SDK Reference v25.2.6

See all our docs at docs.snorkel.ai

PDF generated on April 11, 2025

API Reference

`snorkelflow.client`	Interfaces to interact with the Snorkel Flow REST API.
`snorkelflow.lfs`	Classes that represent LF and LF Package.
`snorkelflow.operators`	Functionality for writing custom operators (preprocessors and postprocessors) in Python.
`snorkelflow.rich_docs`	Utilities to manipulate Rich Document structures for labeling functions and operators in Snorkel Flow.
`snorkelflow.utils`	Other general utilities.
`snorkelflow.ingest`	Functionality to ingest data of various formats into Snorkel Flow.
`snorkelflow.studio`	Functionality for writing custom labeling functions.
`snorkelflow.sdk`	(Beta) Object-oriented SDK for Snorkel Flow

`templates`	LF template schema

`operators`	Built-in Operators.

operators

Built-in Operators.

Built-in operators are available in the full SDK and can be programmatically added to the DAG like below:

```
# Add the operator to the DAG
sf.add_node(
    application=APP_NAME,
    input_node_uids=[123],
    output_node_uids=[456],
    op_type="ColumnRenamer",
    op_config={"column_map": {"body": "email-body"}},
)
```

Featurizers

Text-based

operators.truncate.TruncatePreprocessor(field)	Truncates a column by given amount.
operators.candidates.span_preview.SpanPreviewPreprocessor([...])	Operator that adds a field with nearby textual features for each extracted span.
operators.whitespace.WhitespacePreprocessor(fields)	Preprocessor that normalizes whitespace.
operators.spacy.SpacyPreprocessor(field[, ...])	Preprocessor that parses document and adds json doc column.
operators.spacy.SpacyTokenizer(text_field[, ...])	Preprocessor that parses document and adds tokens column.
operators.spacy.NounChunkFeaturizer(field[, ...])	A Featurizer that yields all noun phrases according to spaCy.

`operators.spacy.VerbPhraseFeaturizer`(field)	A Featurizer that yields all verb phrases according to a simple part-of-speech verb match.
`operators.spacy.SentenceFeaturizer`(field[, ...])	A Featurizer that yields all sentences according to spaCy.
`operators.embedding.EmbeddingFeaturizer`(field)	Featurizer that converts text to an embedding.
`operators.embedding.EmbeddingCandidateFeaturizer`(...)	Featurizer that converts text to an embedding.
`operators.special_char.AsciiCharFilter`(field)	Preprocessor that removes non-ascii chars from selected column in place.
`operators.special_char.LatinCharFilter`(field)	Preprocessor that removes non-latin chars from selected column in place.
`operators.candidates.context.ContextAggregator`()	Adds a column with aggregated spans for the current context_uid.
`operators.candidates.extractor.EmptySpanFeaturizer`(field)	A SpanFeaturizer that yields a single empty span from (0,0) for each row
`operators.candidates.extractor.RegexSpanFeaturizer`(...)	A SpanFeaturizer that yields all matches for a given regular expression
`operators.candidates.extractor.DateSpanFeaturizer`(field)	Extracts spans (slices of documents) that contain dates (using regex)
`operators.candidates.extractor.ParagraphSpanFeaturizer`(field)	Extracts spans (slices of documents) that contain paragraphs (using regex)

`operators.candidates.extractor.NumericSpanFeaturizer`(field)	Extracts spans (slices of documents) that contain numeric values
`operators.candidates.extractor.EmailAddressSpanFeaturizer`(field)	Extracts spans (slices of documents) that contain email addresses (using regex)
`operators.candidates.extractor.USCurrencySpanFeaturizer`(field)	Extracts spans (slices of documents) that contain US currency (using regex)
`operators.candidates.extractor.HardCodedSpanFeaturizer`(...)	A SpanFeaturizer that reads spans directly from its config.
`operators.candidates.extractor.SpansFileFeaturizer`(path)	A SpanFeaturizer that reads spans directly from a file with the expected span columns
`operators.candidates.extractor.EntityDictSpanFeaturizer`(...)	SpanFeaturizer that yields (and optionally links) spans in an entity-to-aliases dictionary
`operators.candidates.extractor.EntityDictRegexSpanFeaturizer`(...)	A SpanFeaturizer that yields (and optionally links) spans in an entity-to-aliases dictionary, which supports regexes.
`operators.candidates.extractor.DocEntityDictSpanFeaturizer`(...)	(Optimized for keyword aliases) SpanFeaturizer that yields (and optionally links) spans, given an entity-to-aliases dictionary and doc-id-to-entity dictionary.
`operators.candidates.extractor_spacy.TokenSpanFeaturizer`(field)	A SpanFeaturizer that yields every token, given a selected tokenization strategy

`operators.candidates.extractor_spacy.NounChunkSpanFeaturizer`(...)	A SpanFeaturizer that yields all noun phrases according to spaCy
`operators.candidates.extractor_spacy.TagSpanFeaturizer`(...)	A SpanFeaturizer that yields all matches for a given NER tag according to spaCy
`operators.candidates.extractor_spacy.SpacyNERSpanFeaturizer`(...)	A SpanFeaturizer that yields all matches for a default list of NER tags according to spaCy

PDF-based

`operators.candidates.rich_doc_features.RichDocRegexNGramDetector`(regex)	Featurizer that detects ngrams matching a regex pattern.
`operators.candidates.rich_doc_features.RichDocRegexPageFeaturizer`(...)	This operator adds a list of pages to retain based on the regex pattern provided.
`operators.candidates.rich_doc_features.RichDocSpanBaseFeaturesPreprocessor`()	Operator that computes basic features for each span using the associated RichDoc object (e.g., bounding box values of the span, page numbers, etc.)
`operators.candidates.rich_doc_features.RichDocSpanRowFeaturesPreprocessor`([...])	Operator that computes row-level features for a span (eg.
`operators.candidates.rich_doc_features.RichDocSpanStructuralPreprocessor`([...])	Operator to compute structural Rich Doc features for span.

`operators.candidates.rich_doc_features.RichDocSpanVisualPreprocessor`([...])	Operator to compute visual Rich Doc features for span.
`operators.candidates.rich_doc_page.RichDocPagePreprocessor`()	Operator that filters out all RichDoc pages except the one containing the span to show on the frontend (for the sake of performance)
`operators.pdf.checkbox.CheckboxFeaturizer`([...])	A featurizer that identifies checkboxes in PDF documents.
`operators.pdf.checkbox.CheckboxSpanMapper`([...])	An operator that assign checkbox-related features to the spans in the document.
`operators.pdf.hocr.HocrToRichDocParser`(field)	Operator that parses hOCR and creates a RichDoc, a Snorkel-Flow-native representation of a pdf document with formatting preserved.
`operators.pdf.hocr.TruncateHOCR`(field[, ...])	Truncates a HOCR document to a certain # of pages.
`operators.pdf.lines.LinesFeaturizer`(field[, ...])	A featurizer that identifies horizontal and vertical lines in PDF documents.
`operators.pdf.lines.LinesPageFilterFeaturizer`(...)	Operator that filters horizontal and vertical lines to subset of pages.
`operators.pdf.page_splitter.PageSplitter`([...])	Split PDFs into pages for subsequent filtering and faster processing.
`operators.pdf.parser.PDFToRichDocParser`(field)	Operator that parses a PDF into a RichDoc (see docs for details).
`operators.pdf.parser2.PDFToRichDocParser2`(field)	Operator that parses a PDF into Snorkel's RichDoc representation.

`operators.pdf.text_cluster.TextClusterer`([...])	Operator that clusters horizontally aligned words using word spacing.
`operators.pdf.text_cluster.TextClusterSpanExtractor`()	Extractor that creates one span per horizontal text cluster.
`operators.pdf.text_cluster.TextClusterSpanFeaturizer`()	Featurizer that creates list of spans with one span per horizontal text cluster.
`operators.pdf.truncate_pdf.TruncatePDF`(...)	Truncates a PDF to a certain # of pages.
`operators.row_filter.TableRowFilter`([...])	A filter that filters out rows without tables.
`operators.pdf.table.TableFeaturizer`([field, ...])	A featurizer that detects tables in PDF documents.
`operators.pdf.table.TableSpanMapper`([...])	An operator that maps table predictions to spans.

OCR

`operators.ocr.tesseract_featurizer.TesseractFeaturizer`(...)	Operator that takes in a PDF URL and outputs the hOCR text.
`operators.azure.azure_form_recognizer_parser.AzureFormRecognizerParser`(...)	Takes in a PDF URL and runs Azure Form Recognizer on it.

Filters

`operators.filter.LabelIntFilter`(label_ints)	A filter that includes/excludes all datapoints corresponding to specified label ints.
`operators.filter.LabelFilter`(label_strs[, ...])	A filter that includes/excludes all datapoints corresponding to specified label strings.

`operators.filter.MultiLabelFilter`(label_strs)	A filter that includes/excludes all datapoints corresponding to specified label strings for multilabel classification.
`operators.row_filter.BooleanColumnBasedRowFilter`(...)	Filters rows based on the specified boolean column.
`operators.row_filter.PandasQueryFilter`(query)	Includes or excludes rows based on a pandas query
`operators.row_filter.TextSizeFilter`(field, ...)	A filter that excludes rows with a text column larger than specificed size (in KB)
`operators.row_filter.RegexRowFilter`(...[, ...])	Filters rows based on the regex pattern provided.
`operators.candidates.filter.RegexSpanFilter`(regex)	A filter that REMOVES all candidates that match a given regular expresion
`operators.candidates.filter.ExtractedSpanFilter`([...])	A filter that removes all spans with a negative prediction.

Extractors

`operators.candidates.extractor.ListToRowsExploder`(...)	A SpanExtractor that explodes a field containing a list of spans.
`operators.candidates.extractor.EmptySpanExtractor`(field)	A SpanExtractor that yields a single empty span from (0,0) for each row
`operators.candidates.extractor.RegexSpanExtractor`(...)	A SpanExtractor that yields all matches for a given regular expression

`operators.candidates.extractor.DateSpanExtractor`(field)	Extracts spans (slices of documents) that contain dates (using regex)
`operators.candidates.extractor.ParagraphSpanExtractor`(field)	Extracts spans (slices of documents) that contain paragraphs (using regex)
`operators.candidates.extractor.NumericSpanExtractor`(field)	Extracts spans (slices of documents) that contain numbers (using regex)
`operators.candidates.extractor.EmailAddressSpanExtractor`(field)	Extracts spans (slices of documents) that contain email addresses (using regex)
`operators.candidates.extractor.USCurrencySpanExtractor`(field)	Extracts spans (slices of documents) that contain US currency (using regex)
`operators.candidates.extractor.HardCodedSpanExtractor`(...)	A SpanExtractor that reads spans directly from its config.
`operators.candidates.extractor.SpansFileExtractor`(path)	A SpanExtractor that reads spans directly from a file with columns ["char_start", "char_end", "context_uid", "span_field", "initial_label", "span_entity"]
`operators.candidates.extractor.EntityDictSpanExtractor`(...)	SpanExtractor that yields (and optionally links) spans in an entity-to-aliases dictionary
`operators.candidates.extractor.EntityDictRegexSpanExtractor`(...)	A SpanExtractor that yields (and optionally links) spans in an entity-to-aliases dictionary, which supports regexes.

`operators.candidates.extractor.DocEntityDictSpanExtractor`(...)	(Optimized for keyword aliases) SpanExtractor that yields (and optionally links) spans, given an entity-to-aliases dictionary and doc-id-to-entity dictionary.
`operators.candidates.extractor_spacy.TokenSpanExtractor`(field)	A SpanExtractor that yields every token, given a selected tokenization strategy
`operators.candidates.extractor_spacy.NounChunkSpanExtractor`(field)	A SpanExtractor that yields all noun phrases according to spaCy
`operators.candidates.extractor_spacy.TagSpanExtractor`(...)	A SpanExtractor that yields all matches for a given NER tag according to spaCy
`operators.candidates.extractor_spacy.SpacyNERSpanExtractor`(...)	

Normalizers/linkers

`operators.candidates.normalizer.DateSpanNormalizer`()	Normalizes date spans into their canonical forms, e.g. 2020-01-01.
`operators.candidates.normalizer.USCurrencySpanNormalizer`()	Normalizes US currency spans into their numerical values.
`operators.candidates.normalizer.TextCasingSpanNormalizer`()	Normalizes spans by lowercasing them, then capitalizing the first letter of each word.
`operators.candidates.normalizer.OrdinalSpanNormalizer`()	Normalizes numerical ordinals (1st, 2nd, etc) to string ordinals (first, second, etc).
`operators.candidates.normalizer.NumericalSpanNormalizer`()	Normalizes numerical cardinals (1, 2, etc) to string values (one, two, etc).

`operators.candidates.normalizer.SpanEntityNormalizer`()	Copies the linked span entity from the extractor as the normalized span.
`operators.candidates.normalizer.IdentitySpanNormalizer`()	Copies span text as is.
`operators.candidates.linker.EntityDictLinker`(...)	Maps span_text to span_entity given an entity to alias dictionary.

Model postprocessors

`operators.post_processors.sequence_tagging_post_processors.SpanFilterByLengthPostProcessor`(...)	Filter positive spans with length <= the provided span_length
`operators.post_processors.sequence_tagging_post_processors.SpanMergeByRegexPatternPostProcessor`(...)	Merge same-label nearby spans if the negative-label text in between spans matches this regex pattern.
`operators.post_processors.sequence_tagging_post_processors.SpanMergeByNumberCharacterPostProcessor`(...)	Merge same-label nearby spans if the negative-label text in between spans is in [lower, upper] number of characters (inclusive).
`operators.post_processors.sequence_tagging_post_processors.SpanRegexPostProcessor`(...)	Post Porcessor that labels anything that matches the provided pattern with the provided label
`operators.post_processors.sequence_tagging_post_processors.SpanRemoveWhitespacePostProcessor`(field)	Remove leading & trailing whitespace in positive spans.
`operators.post_processors.sequence_tagging_post_processors.SubstringExpansionPostProcessor`(...)	This postprocessor expands predictions to the token boundaries should a prediction boundary fall mid-token.

Reducers

`operators.candidates.reducer.IdentityReducer`([...])	No-Op Reducer that passes all Spans through
`operators.candidates.reducer.DocumentFirstReducer`([...])	Reduces span predictions in a document to the span which occurs first.
`operators.candidates.reducer.DocumentLastReducer`([...])	Reduces spans predictions in a document to the span which occurs last.
`operators.candidates.reducer.DocumentMostConfidentReducer`([...])	Reduces spans predictions in a document to the span with the most confident model prediction.
`operators.candidates.reducer.DocumentMostCommonReducer`([...])	Reduces spans predictions in a document to the span which occurs most frequently.
`operators.candidates.reducer.EntityMeanPredictionReducer`([...])	Reduces span predictions for a document entity by computing the mean prediction.
`operators.candidates.reducer.EntityMostCommonReducer`([...])	Reduces span predictions for a document entity to the prediction of the majority vote class.
`operators.candidates.reducer.EntityMostConfidentReducer`([...])	Reduces span predictions for a document entity to the most confident prediction.
`operators.candidates.reducer.EntityFirstReducer`([...])	Reduces span predictions for a document entity to the first occuring span of that entity.
`operators.candidates.reducer.EntityLastReducer`([...])	Reduces span predictions for a document entity to the last occuring span of that entity.

Miscellaneous

`operators.rename.ColumnRenamer`(column_map)	Preprocessor that renames columns
`operators.drop.ColumnDropper`(fields)	Processor that drops given columns from the DataFrame.
`operators.concat.ConcatRows`()	Processor that concatenates dataframes along the index axis.
`operators.concat.ConcatColumns`([join_type, ...])	Processor that concatenates columns of dataframes.
`operators.change_datapoint.ChangeDatapoint`(...)	Preprocessor that changes the datapoint type/columns.
`operators.table.TableConverter`(field[, ...])	Feautizer that convert an array of dicts into a custom Table object
`operators.identity.IdentityOperator`()	No-op operator
`operators.scaler.StandardScaler`(field[, ...])	Preprocessor that scales a numerical column to mean=0 and std=1.
`operators.filler.ColumnFiller`(field, value)	Operator that replaces values in either a new or existing column in the DataFrame with the given value
`operators.trace.TraceToStepFlattener`(field)	Flattens a nested JSON field that contains a hierarchical trace into individual steps.

operators.candidates.extractor.DateSpanExtractor

class operators.candidates.extractor.DateSpanExtractor(*field, col_suffix=None*)

Extracts spans (slices of documents) that contain dates (using regex)

This operator uses a regex to extract all spans from the parent document that contain dates. The vast majority of standard date formats are supported, including ISO-8601, RFC-3339, and others.

Parameters

Name	Type	Default	Info
field	`str`		The dataframe column to extract date spans from.
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.DocEntityDictSpanExtractor

class operators.candidates.extractor.DocEntityDictSpanExtractor(*entity_dict_path*, *doc_entity_dict_path*, *field*, *ignore_case=False*, *link_entities=True*, *col_suffix=None*)

(Optimized for keyword aliases) SpanExtractor that yields (and optionally links) spans, given an entity-to-aliases dictionary and doc-id-to-entity dictionary.

operators.candidates.extractor.EmailAddressSpanExtractor

class operators.candidates.extractor.EmailAddressSpanExtractor(*field*, *col_suffix=None*)

Extracts spans (slices of documents) that contain email addresses (using regex)

operators.candidates.extractor.EmptySpanExtractor

class operators.candidates.extractor.EmptySpanExtractor(*field*, *col_suffix=None*)

A SpanExtractor that yields a single empty span from (0,0) for each row

This can be used to create a candidate set with a one-to-one mapping with documents, which can serve as a starting point for labeling GT manually.

operators.candidates.extractor.EntityDictRegexSpanExtractor

class operators.candidates.extractor.EntityDictRegexSpanExtractor(*entity_dict_path*, *field*, *ignore_case=False*, *link_entities=True*, *col_suffix=None*)

A SpanExtractor that yields (and optionally links) spans in an entity-to-aliases dictionary, which supports regexes. Entity Dict Extractor is better suited for keywords.

This is used for entity classification tasks. It additionally annotates each span with the linked entity, using the dictionary value. By default regexes provided are surrounded by (`\b`).

operators.candidates.extractor.EntityDictSpanExtractor

class operators.candidates.extractor.EntityDictSpanExtractor(*entity_dict_path*, *field*, *ignore_case=False*, *link_entities=True*, *col_suffix=None*)

SpanExtractor that yields (and optionally links) spans in an entity-to-aliases dictionary

This is used for entity classification tasks. It additionally annotates each span with the linked entity, using the dictionary value. This operator is Optimized for keyword aliases.

An example of the entity-to-aliases dictionary can be found in Entity Classification Tutorials.

Parameters

Name	Type	Default	Info
entity_dict_path	`str`		The path to the entity-to-aliases dictionary.
field	`str`		The dataframe column to extract spans from.
ignore_case	`bool`	`False`	If true, the extraction will be NOT case sensitive (default to false).
link_entities	`bool`	`True`	If true, the extracted span will be linked with its original entity/aliases.
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.HardCodedSpanExtractor

class operators.candidates.extractor.HardCodedSpanExtractor(*char_starts*, *char_ends*, *context_uids*, *span_fields*, *initial_labels*, *span_entities*, *col_suffix=None*)

A SpanExtractor that reads spans directly from its config.

operators.candidates.extractor.ListToRowsExploder

class operators.candidates.extractor.ListToRowsExploder(*span_fields_schema*)

A SpanExtractor that explodes a field containing a list of spans.

This can be combined with a featurizer that produces a list of spans. For the featurizer before this operator, each field's type in the output schema should be object rather than the type in the list.

Parameters

Name	Type	Default	Info
span_fields_schema	`Dict[str, str]`		Dictionary mapping a span field name to the type contained within that field.

operators.candidates.extractor.NumericSpanExtractor

class operators.candidates.extractor.NumericSpanExtractor(*field, col_suffix=None*)

Extracts spans (slices of documents) that contain numbers (using regex)

operators.candidates.extractor.ParagraphSpanExtractor

class operators.candidates.extractor.ParagraphSpanExtractor(*field, col_suffix=None*)

Extracts spans (slices of documents) that contain paragraphs (using regex)

This operator uses a regex pattern to extract all paragraphs as spans from the parent document. Trailing newline characters are preserved for each paragraph

Parameters

Name	Type	Default	Info
field	`str`		The dataframe column to extract paragraph spans from.
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.RegexSpanExtractor

class operators.candidates.extractor.RegexSpanExtractor(*regex, field, ignore_case=False, capture_group=0, col_suffix=None*)

A SpanExtractor that yields all matches for a given regular expression

This operator applies a given regular expression over a specified field in the passing DataFrame and adds each match as a new span in the output.

Parameters

Name	Type	Default	Info
regex	`str`		The regular expression to apply over the data.
field	`str`		The name of the column in the dataframe to apply to provided regex over.
ignore_case	`bool`		If true, ignore case when considering regular expression matches (defaults to false).
capture_group	`str`		The capture group to provide to the regex results.
col_suffix	`str`		An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.SpansFileExtractor

class operators.candidates.extractor.SpansFileExtractor(*path*, *source_type='CSV'*, *col_suffix=None*)

A SpanExtractor that reads spans directly from a file with columns [“char_start”, “char_end”, “context_uid”, “span_field”, “initial_label”, “span_entity”]

operators.candidates.extractor.USCurrencySpanExtractor

class operators.candidates.extractor.USCurrencySpanExtractor(*field*, *col_suffix=None*)

Extracts spans (slices of documents) that contain US currency (using regex)

operators.candidates.extractor_spacy.NounChunkSpanExtractor

class operators.candidates.extractor_spacy.NounChunkSpanExtractor(*field*, *col_suffix=None*, ***spacy_span_kwargs*)

A SpanExtractor that yields all noun phrases according to spaCy

operators.candidates.extractor_spacy.SpacyNERSpanExtractor

class operators.candidates.extractor_spacy.SpacyNERSpanExtractor(*tag*, *field*, *match_longest=True*, *col_suffix=None*, ***spacy_span_kwargs*)

operators.candidates.extractor_spacy.TagSpanExtractor

class operators.candidates.extractor_spacy.TagSpanExtractor(*field, tag, attr='ent_type_', match_longest=True, col_suffix=None, **spacy_span_kwargs*)

A SpanExtractor that yields all matches for a given NER tag according to spaCy

Parameters

Name	Type	Default	Info
tag	`Union[str, List[str]]`		The set of tags to look for.
attr	`str`	`'ent_type_'`	The spaCy attribute to compare to the tags (e.g., `ent_type_`).
model			The model to load into spaCy (only supports models in strap).
fields			The fields in which to look for candidates.
match_longest	`bool`	`True`	If True, return contiguous chunks of a tag as a single candidate.

operators.candidates.extractor_spacy.TokenSpanExtractor

class operators.candidates.extractor_spacy.TokenSpanExtractor(*field*, *tokenizer='spacy'*, *col_suffix=None*, ***spacy_span_kwargs*)

A SpanExtractor that yields every token, given a selected tokenization strategy

operators.azure.azure_form_recognizer_parser.AzureFormRecognizerParser

class operators.azure.azure_form_recognizer_parser.AzureFormRecognizerParser(*pdf_url_field, form_recognizer_endpoint, form_recognizer_key, result_upload_storage_key='', result_upload_container=None, result_upload_blob_prefix=None, result_upload_overwrite=True*)

Takes in a PDF URL and runs Azure Form Recognizer on it. The result is returned as a RichDoc. The result is also uploaded to blob storage if configured. The form recognizer endpoint and key are configured as secrets.

Parameters

Name	Type	Default	Info
pdf_url_field	`NewType(DataframeFieldType, str)`		The name of the column in the dataframe contains PDF urls.
form_recognizer_endpoint	`str`		The endpoint for the Azure Form Recognize service.

form_recognizer_key	`str`		The key in the secret store that has the Azure Form Recognize key.
result_upload_storage_key	`str`	`''`	The key in the secret store that has connectio string for the Azure blob storage account.
result_upload_container	`Optional[str]`	`None`	The container in the Azure blob storage account to upload results to.
result_upload_blob_prefix	`Optional[str]`	`None`	The prefix to use for the blob name.
result_upload_overwrite	`bool`	`True`	Whether to overwrite

			existing blobs.

Returns

- *rich_doc_text* – A stripped raw text representation of the rich doc
- *rich_doc_pkl* – A serialized RichDoc that corresponds to rich_doc_text
- *page_docs* – A serialized list of RichDoc objects, one per page
- *page_char_starts* – A character offsets of text starting on each page

operators.ocr.tesseract_featurizer.TesseractFeaturizer

class operators.ocr.tesseract_featurizer.TesseractFeaturizer(*pdf_url_field, output_field, ignore_errors=False*)

Operator that takes in a PDF URL and outputs the hOCR text.

This operator uses the Tesseract OCR engine to convert PDFs to hOCR text. The hOCR text is then used by the HocrToRichDocParser to generate a RichDoc object for the PDF.

Parameters

Name	Type	Default	Info
pdf_url_field	`NewType(DataframeFieldType, str)`		The name of the field that contains the URL of the PDF to be OCR'd.
output_field	`str`		The name of the field that will contain the hOCR text.
ignore_errors	`bool`	`False`	Whether we want to raise errors or not for bad PDF files.

Returns

output_field

Return type

`The hOCR text`

operators.candidates.rich_doc_features.RichDocRegexNGramDetector

class operators.candidates.rich_doc_features.RichDocRegexNGramDetector(*regex, target_field=None, capture_group=0, case_sensitive=True*)

Featurizer that detects ngrams matching a regex pattern.

Parameters

Name	Type	Default	Info
regex	`str`		The regex pattern to search for.
target_field	`Optional[str]`	`None`	The name of the field to store the detected ngrams in.
capture_group	`int`	`0`	The capture group to use when extracting the ngram text.
case_sensitive	`bool`	`True`	Whether to ignore case when searching for the regex pattern.

operators.candidates.rich_doc_features.RichDocRegexPageFeaturizer

class operators.candidates.rich_doc_features.RichDocRegexPageFeaturizer(*regex_pattern*, *case_sensitive=False*)

This operator adds a list of pages to retain based on the regex pattern provided. The regex pattern is searched for over the text in each page of the document. The list of pages to retain is stored in the *context_pages* field.

This operator is the first step in filtering out pages based on keywords in native PDF extraction applications. The user should add this operator followed by a PDFToRichDocParser. The *context_pages* field should be provided as the "Pages field" input to the PDFToRichDocParser.

Parameters

Name	Type	Default	Info
regex_pattern	`str`		The regular expression pattern we use to filter rows.
case_sensitive	`bool`	`False`	If False, ignore case when considering regular expression matches (defaults to False).

operators.candidates.rich_doc_features.RichDocSpanBaseFeaturesPreprocessor

class operators.candidates.rich_doc_features.RichDocSpanBaseFeaturesPreprocessor

Operator that computes basic features for each span using the associated RichDoc object (e.g., bounding box values of the span, page numbers, etc.)

This operator compute span-based features for richer information for each span. The list of computed features can be found in the Returns section.

This operator usually co-exists with RichDocSpanBaseFeaturesPreprocessor and RichDocSpanRowFeaturesPreprocessor to create RichDoc representation and features.

This operator utilizes existing RichDoc prepopulated columns (no input required).

Returns

- *rich_doc_span_page_id (int)* – Page number (0-based)
- *rich_doc_span_start_word_id (str)* – The ID of the word containing the start of the span (hereafter referred to as “start word”)
- *rich_doc_span_start_char_offset (int)* – The inclusive start index of the span, relative to the start index of the “start word”
- *rich_doc_span_end_word_id (str)* – The ID of the word containing the end of the span
- *rich_doc_span_end_char_offset (int)* – The inclusive end index of the span, relative to the start index of the “start word”
- rich_doc_span_ngram (`Ngram`) – The `Ngram` object for words containing the span
- *left (int)* – The left of the bounding box
- *top (int)* – The top of the bounding box
- *right (int)* – The right of the bounding box
- *bottom (int)* – The bottom of the bounding box
- *rich_doc_font_size (int)* – The font size of the `Ngram` object

operators.candidates.rich_doc_features.RichDocSpanRowFeaturesPreprocessor

class operators.candidates.rich_doc_features.RichDocSpanRowFeaturesPreprocessor(*row_id=False, row_text_before=0, row_text_inline=False, row_text_after=0, row_header=False, inferred_row_headers=False, row_header_json='{"scope": "page", "multi_row": true, "min_margin": 10, "max_gap": 20, "max_left_page_pct": 50}', mask_span=True, feature_suffix=''*)

Operator that computes row-level features for a span (eg. text from the span's row, text before and after the span's row etc)

This operator compute row-level features for richer information for each span. The list of computed features can be found in the Returns section (optionally with a suffix on each feature name).

This operator usually co-exists with RichDocSpanBaseFeaturesPreprocessor and RichDocSpanBaseFeaturesPreprocessor to create RichDoc representation and features.

Parameters

Name	Type	Default	Info
row_id	`int`		If True, calculate the rich_doc_row_id feature.
row_text_before	`str`		If positive, include this many rows before span in rich_doc_row_text_before.
row_text_inline	`bool`		If True, calculate the rich_doc_row_text_inline feature.
row_text_after	`int`		If positive, include this many rows before span in rich_doc_row_text_after.
row_header	`bool`		If True, calculate the rich_doc_row_header feature.
inferred_row_headers	`bool`		If True, calculate the rich_doc_inferred_row_headers feature.

mask_span	`str`		If True, replace the span content with '-SPAN-' in rich_doc_row_text_inline.
row_header_json	`bool`		JSON string containing additional settings for row header features.
feature_suffix	`str`		If None, auto-generate suffixes for features based on their parameters. Otherwise, append this string to each feature (use empty string for no suffixes).

Returns

- *rich_doc_row_id* – The (int) index of the span's row
- *rich_doc_row_text_before* – The text in the rows 1 to X before the span's row
- *rich_doc_row_text_after* – The text in the rows 1 to X after the span's row
- *rich_doc_row_text_inline* – The text from the span's row
- *rich_doc_row_header* – The text in the span's row header
- *rich_doc_inferred_row_headers* – The text in the span's inferred row headers

operators.candidates.rich_doc_features.RichDocSpanStructuralPreprocessor

class operators.candidates.rich_doc_features.RichDocSpanStructuralPreprocessor(*window=1, scope_unit='line', direction='before or after', feature_name_override=None*)

Operator to compute structural Rich Doc features for span.

This operator computes structural Rich Doc features for span. Available Features (optionally with a suffix on the feature name):

Note: rich_doc_proximate_text: The text in [window] [scope_unit]s [direction] of span (e.g., [1] [line] [before] the span)

Parameters

Name	Type	Default	Info
window	`int`	`1`	Number of scope units to extract feature text from.
scope_unit	`str`	`'line'`	The unit to use (word / line / par / area / page).
direction	`str`	`'before or after'`	The direction (before_only / after_only / before_or_after) to extract feature text from relative to span.
feature_name_override	`Optional[str]`	`None`	If not None, use this as the generated column name (instead of an auto-generated name).

operators.candidates.rich_doc_features.RichDocSpanVisualPreprocessor

class operators.candidates.rich_doc_features.RichDocSpanVisualPreprocessor(*location='center', scope='page', threshold=50, threshold_unit='pixels', threshold_dir='left_or_right', mask_span_ngrams=True, ngram_range_min=1, ngram_range_max=2, feature_name_override=None*)

Operator to compute visual Rich Doc features for span.

Operator to compute visual Rich Doc features for span. Available Features (optionally with a suffix on the feature name):

Note: rich_doc_aligned_ngrams: The ngrams in the given [scope] whose [location] is within [threshold] [threshold_unit]s

Parameters

Name	Type	Default	Info
location	`str`	`'center'`	The location of the span and ngrams to compare (left / center / right / top / middle / bottom).
scope	`str`	`'page'`	The scope to search for ngrams within (word / line / par / area / page).
threshold	`int`	`50`	The maximum threshold used when comparing two location values.

threshold_dir	`str`	`'left_or_right'`	A specific direction for restricting the search for aligned ngrams (left_only, right_only, left_or_right, up_only, down_only, up_or_down).
mask_span_ngrams	`bool`	`True`	If True, replace the span with -SPAN- in all ngrams.
ngram_range_min	`int`	`1`	The lower bound of ngrams to include (e.g., 1 = unigrams, 2 = bigrams, etc.).
ngram_range_max	`int`	`2`	The upper bound of ngrams to include (e.g., 1 = unigrams, 2 = bigrams, etc.).
feature_name_override	`Optional[str]`	`None`	If not None, use this as the generated column name (instead of an auto-generated name).

operators.candidates.rich_doc_page.RichDocPagePreprocessor

class operators.candidates.rich_doc_page.RichDocPagePreprocessor

Operator that filters out all RichDoc pages except the one containing the span to show on the frontend (for the sake of performance)

This operator removes all pages expect the ones containing the span to show in the frontend. This preprocessor creates the {RichDocCols.DOC_COL} column and is required by the frontend for rendering a RichDoc with its original formatting on the Data Viewer.

This operator usually co-exists with RichDocSpanBaseFeaturesPreprocessor and RichDocSpanRowFeaturesPreprocessor to create RichDoc representation and features.

NOTE: RichDocPagePreprocessor can only be used with spans extracted from the {RichDocCols.TEXT_COL} column.

NOTE: The candidate RichDoc is added to the same column as the context RichDoc.

This operator utilizes existing RichDoc prepopulated columns (no inputs required). The output includes the column contains RichDoc formatting for frontend rendering (RichDocCols.DOC_COL)

Returns

The column contains RichDoc formatting for frontend rendering

Return type

`{RichDocCols.DOC_COL}`

operators.pdf.checkbox.CheckboxFeaturizer

class operators.pdf.checkbox.CheckboxFeaturizer(*pdf_url_field='rich_doc_pdf_url', min_box_length_px=25, max_box_length_px=55, px_threshold_ratio=0.1, num_pages_per_batch=100, pages_field=None*)

A featurizer that identifies checkboxes in PDF documents.

Parameters

Name	Type	Default	Info
pdf_url_field	`str`	`'rich_doc_pdf_url'`	The name of the column containing the PDF file paths.
min_box_length_px	`int`	`25`	The minimum length of a checkbox. Defaults to 25.
max_box_length_px	`int`	`55`	The maximum length of a checkbox. Defaults to 55.
px_threshold_ratio	`float`	`0.1`	The threshold ratio of non-empty pixels inside a checkbox to be

			considered as checked. Defaults to 0.1.
num_pages_per_batch	`int`	`100`	The number of pages to process in each batch. Defaults to 100.
pages_field	`Optional[str]`	`None`	The name of the column containing the page numbers on which to run the operator on. If None, the operator will run on all pages. Defaults to None.

Returns

A Checkbox object containing the checkbox info.

Return type

`{RichDocCols.CHECKBOXES}`

operators.pdf.checkbox.CheckboxSpanMapper

class operators.pdf.checkbox.CheckboxSpanMapper(*px_distance=100, left_checkboxes=True, right_checkboxes=False, top_checkboxes=False, bottom_checkboxes=False*)

An operator that assign checkbox-related features to the spans in the document.

Parameters

Name	Type	Default	Info
px_distance	`int`	`100`	The distance in pixels to map checkboxes to spans. Defaults to 100px.
left_checkboxes	`bool`	`True`	Whether to assign checkboxes to the left of the span. Defaults to True.
right_checkboxes	`bool`	`False`	Whether to assign checkboxes to the right of the span. Defaults to False.
top_checkboxes	`bool`	`False`	Whether to assign checkboxes to the top of the span. Defaults to False.
bottom_checkboxes	`bool`	`False`	Whether to assign checkboxes to the bottom of the span. Defaults to False.

Returns

- *{is_left_checkbox_checked}* – A boolean value indicating whether the span contains a checkbox. If no checkboxes are found, the value is null.
- *{is_right_checkbox_checked}* – A boolean value indicating whether the span contains a checkbox. If no checkboxes are found, the value is null.
- *{is_top_checkbox_checked}* – A boolean value indicating whether the span contains a checkbox. If no checkboxes are found, the value is null.
- *{is_bottom_checkbox_checked}* – A boolean value indicating whether the span contains a checkbox. If no checkboxes are found, the value is null.

operators.pdf.hocr.HocrToRichDocParser

class operators.pdf.hocr.HocrToRichDocParser(*field, drop_field=True, ignore_errors=False*)

Operator that parses hOCR and creates a RichDoc, a Snorkel-Flow-native representation of a pdf document with formatting preserved.

This operator parses hOCR field to create a Snorkel-Flow-native representation of hOCR documents, including a richer text representation, spatial information, etc. with the original formatting preserved. RichDoc representation empowers in-depth tools with hOCR-formatted data.

The output includes: a stripped raw text representation of the rich doc (rich_doc_text), a serialized RichDoc that corresponds to rich_doc_text (rich_doc_pkl), a serialized list of RichDoc objects, one per page (page_docs), and character offsets of text starting on each page (page_char_starts).

Parameters

Name	Type	Default	Info
field	`str`		The name of the column in the dataframe contains hOCR values.
drop_field	`bool`	`True`	A boolean to drop original hOCR text when True. This option is deprecated and a no-op. Please use ColumnDropper instead.
ignore_errors	`bool`	`False`	If set true, ignore docs having hOCR parsing errors or missing and return empty RichDoc.

Returns

- **rich_doc_text** (*a stripped raw text representation of the rich doc*)
- **rich_doc_pkl** (*a serialized RichDoc that corresponds to rich_doc_text*)
- **page_docs** (*a serialized list of RichDoc objects, one per page*)
- **page_char_starts** (*Character offsets of text starting on each page*)

operators.pdf.hocr.TruncateHOCR

class operators.pdf.hocr.TruncateHOCR(*field, target_field=None, pages=5, ignore_errors=False*)

Truncates a HOCR document to a certain # of pages.

Truncates a given column of HOCR documents to a certain # of pages and write the truncated documents to a new column.

Parameters

Name	Type	Default	Info
field	`str`		The name of the column containing the HOCR documents.
target_field	`str, optional`		The name of the column to write the truncated HOCR documents to. If not provided, the truncated documents will be written to a column named <field>_truncated.
pages	`int, optional`		The number of pages to truncate the HOCR documents to. If not provided, the HOCR documents will be truncated to 5 pages.
ignore_errors	`bool, optional`		Whether to ignore errors when parsing the HOCR documents. If True, the original HOCR documents will be written to the target column even if there are errors during truncation.

operators.pdf.lines.LinesFeaturizer

class operators.pdf.lines.LinesFeaturizer(*field, min_length_px=21, num_pages_per_batch=100, pages_field='context_pages'*)

A featurizer that identifies horizontal and vertical lines in PDF documents.

The operator uses Erosion + Dilation, one of the common techniques to identify horizontal / veritical lines in an image.

Parameters

Name	Type	Default	Info
field	`str`		The name of the co containing the PDF
min_length_px	`int`	`21`	The minimum lengt pixels) of a line to b considered valid. D 21.
num_pages_per_batch	`int`	`100`	The number of pag process in each bat Defaults to 100.
pages_field	`Optional[str]`	`'context_pages'`	The name of the co containing the page on which to run the on. Defaults to RichDocCols.CONTE

Returns

A HVLines object containing the horizontal and vertical lines.

Return type

`{RichDocCols.HV_LINES}`

operators.pdf.lines.LinesPageFilterFeaturizer

class operators.pdf.lines.LinesPageFilterFeaturizer(*pages_field*)

Operator that filters horizontal and vertical lines to subset of pages.

operators.pdf.page_splitter.PageSplitter

class operators.pdf.page_splitter.PageSplitter(*window_size=0*)

Split PDFs into pages for subsequent filtering and faster processing. Window size specifies the number of consecutive pages before/after the current page to use as a single document for development.

operators.pdf.parser.PDFToRichDocParser

class operators.pdf.parser.PDFToRichDocParser(*field, remove_superscripts=False, pages_field=None, extract_pars=False, parser_params='{"char_margin": 1.0, "word_margin": 0.1, "all_texts": true}', ignore_errors=False, parser_version=1*)

Operator that parses a PDF into a RichDoc (see docs for details).

This operator parses PDF to create a Snorkel-Flow-native representation of hOCR documents, including a richer text representation, spatial information, etc. with the original formatting preserved. RichDoc representation empowers in-depth tools with PDF-formatted data.

The output includes: a stripped raw text representation of the rich doc (rich_doc_text), a serialized RichDoc that corresponds to rich_doc_text (rich_doc_pkl), a serialized list of RichDoc objects, one per page (page_docs), and character offsets of text starting on each page (page_char_starts).

Parameters

Name	Type	Default	Info
field	`str`		The name of the column in the dataframe contains PDF urls.
remove_superscripts	`bool`	`False`	If true, remove the superscripts in the PDF (default to false).
pages_field	`Optional[str]`	`None`	Field used to filter pages from the pdf.
extract_pars	`bool`	`False`	If true, extract paragraphs/areas from the PDF.

parser_params	`str`	`'{"char_margin": 1.0, "word_margin": 0.1, "all_texts": true}'`	pdfminer parsing parameters that control text grouping.
ignore_errors	`bool`	`False`	If true, ignore errors during parsing.
parser_version	`Optional[int]`	`1`	The version of the parser used to parse the PDF. If not provided, the latest parser version is used.

Returns

- *rich_doc_text* – A stripped raw text representation of the rich doc
- *rich_doc_pkl* – A serialized RichDoc that corresponds to rich_doc_text
- *page_docs* – A serialized list of RichDoc objects, one per page
- *page_char_starts* – A character offsets of text starting on each page

operators.pdf.parser2.PDFToRichDocParser2

class operators.pdf.parser2.PDFToRichDocParser2(*field*, *parser_version=1*)

Operator that parses a PDF into Snorkel's RichDoc representation.

This operator parses PDF to create a Snorkel-Flow-native representation of PDF documents, including a richer text representation, spatial information, etc. with the original formatting preserved. RichDoc representation empowers in-depth tools with PDF-formatted data.

The output includes: a stripped raw text representation of the rich doc (rich_doc_text), a serialized RichDoc that corresponds to rich_doc_text (rich_doc_pkl), a serialized list of RichDoc objects, one per page (page_docs), and character offsets of text starting on each page (page_char_starts).

This parser will ignore parsing errors by default. The documents with errors will be skipped. PDFs with parsing errors are logged and errors are raised to the user.

Parameters

Name	Type	Default	Info
field	`str`		The name of the column in the dataframe contains PDF urls.
parser_version	`Optional[int]`	`1`	The version of the parser used to parse the PDF. If not provided, the latest parser version is used.

Returns

- *rich_doc_text* – A stripped raw text representation of the rich doc
- *rich_doc_pkl* – A serialized RichDoc that corresponds to rich_doc_text
- *page_docs* – A serialized list of RichDoc objects, one per page
- *page_char_starts* – A character offsets of text starting on each page

operators.pdf.table.TableFeaturizer

class operators.pdf.table.TableFeaturizer(*field='rich_doc_pdf_url', model='microsoft/table-transformer-structure-recognition', pages_field=None*)

A featurizer that detects tables in PDF documents.

Parameters

Name	Type	Default	Info
field	`str`	`'rich_doc_pdf_url'`	The name of the column containing the PDF URL paths.
model	`str`	`'microsoft/table-transformer-structure-recognition'`	The pretrained Table Transformer model to use for table detection.
pages_field	`Optional[str]`	`None`	The name of the column containing the page numbers on which to run the operator on. If None, the operator will run on all pages. Defaults to None.

Returns

A Table object containing the table metadata information.

Return type

`{RichDocCols.TABLES}`

operators.pdf.table.TableSpanMapper

class operators.pdf.table.TableSpanMapper(*confidence_threshold=0.9*)

An operator that maps table predictions to spans.

Parameters

Name	Type	Default	Info
confidence_threshold	`float`	`0.9`	Threshold used to filter model predictions. Defaults to 0.9.

Returns

- *{is_table_span}* – A boolean value indicating whether the span is within a table.
- *{table_column_id}* – An integer value indicating the column ID of the span.

operators.pdf.text_cluster.TextClusterSpanExtractor

class operators.pdf.text_cluster.TextClusterSpanExtractor

Extractor that creates one span per horizontal text cluster.

operators.pdf.text_cluster.TextClusterSpanFeaturizer

class operators.pdf.text_cluster.TextClusterSpanFeaturizer

Featurizer that creates list of spans with one span per horizontal text cluster.

This operator considers all text clusters created by TextClusterer as individual spans. For each spans, Prerequisite: The DAG must contain the TextCluterer before TextClusterSpanFeaturizer.

This operator utilizes existing RichDoc prepopulated columns (no input required), includes {{RichDocCols.TEXT_CLUSTERS}: None, {RichDocCols.TEXT_COL}: str}

operators.pdf.text_cluster.TextClusterer

class operators.pdf.text_cluster.TextClusterer(*wordspacing_tolerance=0.75, merge_words_between_vertical_lines=False, merge_rows_between_horizontal_lines=False, pages_field='context_pages'*)

Operator that clusters horizontally aligned words using word spacing.

This operator clusters horizontally aligned word that stays within a predefined word spacing. Text Clusters are the group of words that are separated by a single space. The heuristics employed here relies on the fact that if the words are separated by width > the max_width in standard typography, then they are separate word clusters. The max_width = wordspacing_tolerance * vertical width between 2 previous words.

Optionally Merges Word clusters vertically into regions using bounding horizontal lines. Needs LinesFeaturizer if merge_words_between_vertical_lines or merge_rows_between_horizontal_lines are set.

Parameters

Name	Type	Default	
wordspacing_tolerance	`float`	`0.75`	Th ve w cl
merge_words_between_vertical_lines	`bool`	`False`	If Li w to
merge_words_between_horizontal_lines			If Li w lir
pages_field	`Optional[str]`	`'context_pages'`	Th cc or

			or Ri

Returns

A serialized TextClusters class containing all the text clusters information.

Return type

`{RichDocCols.TEXT_CLUSTERS}`

operators.pdf.truncate_pdf.TruncatePDF

class operators.pdf.truncate_pdf.TruncatePDF(*field*, *pdf_storage_dir*, *target_field=None*, *pages=5*, *ignore_errors=False*)

Truncates a PDF to a certain # of pages.

Truncates a given column of pdf urls to a certain # of pages and write the paths to the truncated pdfs to a new column. This will work for native and scanned PDFs.

Parameters

Name	Type	Default	Info
field	`str`		The field you want to truncate.
pdf_storage_dir	`str`		The directory to store the truncated pdfs in.
target_field	`str`		The field you want to write the truncated PDF paths to.
pages	`int`		The number of pages to truncate to.
ignore_errors	`bool, optional`		Whether to ignore errors when parsing the PDF documents. If True, the original PDF documents will be written to the target column even if there are errors during truncation.

operators.row_filter.TableRowFilter

class operators.row_filter.TableRowFilter(*confidence_score=0.9*)

A filter that filters out rows without tables. This operator requires the predictions from the TableFeaturizer operator.

Parameters

Name	Type	Default	Info
confidence_score	`float`	`0.9`	The minimum confidence score required for a table to be considered valid.

operators.candidates.context.ContextAggregator

class operators.candidates.context.ContextAggregator

Adds a column with aggregated spans for the current context_uid.

operators.candidates.extractor.DateSpanFeaturizer

class operators.candidates.extractor.DateSpanFeaturizer(*field, col_suffix=None*)

Extracts spans (slices of documents) that contain dates (using regex)

This operator uses a regex to extract all spans from the parent document that contain dates. The vast majority of standard date formats are supported, including ISO-8601, RFC-3339, and others.

Parameters

Name	Type	Default	Info
field	`str`		The dataframe column to extract date spans from.
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.DocEntityDictSpanFeaturizer

class operators.candidates.extractor.DocEntityDictSpanFeaturizer(*entity_dict_path*, *doc_entity_dict_path*, *field*, *ignore_case=False*, *link_entities=True*, *col_suffix=None*)

(Optimized for keyword aliases) SpanFeaturizer that yields (and optionally links) spans, given an entity-to-aliases dictionary and doc-id-to-entity dictionary.

operators.candidates.extractor.EmailAddressSpanFeaturizer

class operators.candidates.extractor.EmailAddressSpanFeaturizer(*field, col_suffix=None*)

Extracts spans (slices of documents) that contain email addresses (using regex)

This operator uses a regex to extract all spans from the parent document that contain properly formatted email addresses according to RFC6530.

Parameters

Name	Type	Default	Info
field	`str`		The dataframe column to extract email address spans from.
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.EmptySpanFeaturizer

class operators.candidates.extractor.EmptySpanFeaturizer(*field, col_suffix=None*)

A SpanFeaturizer that yields a single empty span from (0,0) for each row

This can be used to create a candidate set with a one-to-one mapping with documents, which can serve as a starting point for labeling GT manually.

operators.candidates.extractor.EntityDictRegexSpanFeaturizer

class operators.candidates.extractor.EntityDictRegexSpanFeaturizer(*entity_dict_path*, *field*, *ignore_case=False*, *link_entities=True*, *col_suffix=None*)

A SpanFeaturizer that yields (and optionally links) spans in an entity-to-aliases dictionary, which supports regexes. Entity Dict Featurizer is better suited for keywords.

This is used for entity classification tasks. It additionally annotates each span with the linked entity, using the dictionary value. By default regexes provided are surrounded by (`\b`).

operators.candidates.extractor.EntityDictSpanFeaturizer

class operators.candidates.extractor.EntityDictSpanFeaturizer(*entity_dict_path, field, ignore_case=False, link_entities=True, col_suffix=None*)

SpanFeaturizer that yields (and optionally links) spans in an entity-to-aliases dictionary

This is used for entity classification tasks. It additionally annotates each span with the linked entity, using the dictionary value. This operator is Optimized for keyword aliases.

Parameters

Name	Type	Default	Info
entity_dict_path	`str`		A path (either local or to remote storage) that contains the entity linking definitions.
field	`str`		The field of the passing dataframe that entities will be extracted from.
ignore_cae			If true, ignore case when matching entities (defaults to false).
link_entities	`bool`	`True`	If true, link entities (defaults to true).
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.HardCodedSpanFeaturizer

class operators.candidates.extractor.HardCodedSpanFeaturizer(*char_starts, char_ends, context_uids, span_fields, initial_labels, span_entities, col_suffix=None*)

A SpanFeaturizer that reads spans directly from its config.

This SpanFeaturizer directly takes in span definitions as part of its config and applies those span definitions over the passing DataFrame.

Parameters

Name	Type	Default	Info
char_starts	`List[int]`		The indices of the first characters of the spans for this candidate.
char_ends	`List[int]`		The indices of the last characters of the spans for this candidate.
context_uids	`List[int]`		The UIDs of the columns in the passing dataframe to extract the specified span from.
span_fields	`List[str]`		The names of the fields that these spans are extracted from.
initial_labels	`List[int]`		The labels assigned to spans at creation time (if available).
span_entities	`List[Optional[str]]`		The entities linked to the span_text.
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.NumericSpanFeaturizer

class operators.candidates.extractor.NumericSpanFeaturizer(*field, col_suffix=None*)

Extracts spans (slices of documents) that contain numeric values

Parameters

Name	Type	Default	Info
field	`str`		The dataframe column to extract numeric spans from.
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.ParagraphSpanFeaturizer

class operators.candidates.extractor.ParagraphSpanFeaturizer(*field, col_suffix=None*)

Extracts spans (slices of documents) that contain paragraphs (using regex)

This operator uses a regex pattern to extract all paragraphs as spans from the parent document. Trailing newline characters are preserved for each paragraph

Parameters

Name	Type	Default	Info
field	`str`		The dataframe column to extract paragraph spans from.
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.RegexSpanFeaturizer

class operators.candidates.extractor.RegexSpanFeaturizer(*regex, field, ignore_case=False, capture_group=0, col_suffix=None*)

A SpanFeaturizer that yields all matches for a given regular expression

This operator applies a given regular expression over a specified field in the passing DataFrame and adds each match as a new span in the output.

Parameters

Name	Type	Default	Info
regex	`str`		The regular expression to apply over the data.
field	`str`		The name of the column in the dataframe to apply to provided regex over.
ignore_case	`bool`	`False`	If true, ignore case when considering regular expression matches (defaults to false).
capture_group	`int`	`0`	The capture group to provide to the regex results.
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.SpansFileFeaturizer

class operators.candidates.extractor.SpansFileFeaturizer(*path, source_type='CSV', col_suffix=None*)

A SpanFeaturizer that reads spans directly from a file with the expected span columns

This operator reads specified spans from the provided file. The file can live either locally or in cloud storage. The file is expected to encode a dataframe with the following columns: [“char_start”, “char_end”, “context_uid”, “span_field”, “initial_label”, “span_entity”]

Parameters

Name	Type	Default	Info
path	`str`		A path (either local or to S3) to the file that defines the spans.
source_type	`str`	`'CSV'`	Either “CSV” or “PARQUET” for .csv and .parquet files, respectively.
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor.USCurrencySpanFeaturizer

class operators.candidates.extractor.USCurrencySpanFeaturizer(*field, col_suffix=None*)

Extracts spans (slices of documents) that contain US currency (using regex)

Parameters

Name	Type	Default	Info
field	`str`		The dataframe column to extract currency amount spans from.
col_suffix	`Optional[str]`	`None`	An optional suffix for the column containing the extracted spans.

operators.candidates.extractor_spacy.NounChunkSpanFeaturizer

class operators.candidates.extractor_spacy.NounChunkSpanFeaturizer(*field*, ***spacy_span_kwargs*)

A SpanFeaturizer that yields all noun phrases according to spaCy

Parameters

Name	Type	Default	Info
field	`str`		The dataframe column to apply the tokenization strategy over.

operators.candidates.extractor_spacy.SpacyNERSpanFeaturizer

class operators.candidates.extractor_spacy.SpacyNERSpanFeaturizer(*tag*, *field*, *match_longest=True*, ***spacy_span_kwargs*)

A SpanFeaturizer that yields all matches for a default list of NER tags according to spaCy

Parameters

Name	Type	Default	Info
tag	`str`		A valid NER tag (see options).
attr			The spaCy attribute to compare to the tags (e.g., `ent_type_`).
model			The model to load into spaCy (only supports models in strap).
fields			The fields in which to look for candidates.
match_longest	`bool`	`True`	If True, return contiguous chunks of a tag as a single candidate.

operators.candidates.extractor_spacy.TagSpanFeaturizer

class operators.candidates.extractor_spacy.TagSpanFeaturizer(*field*, *tag*, *attr='ent_type_'*, *match_longest=True*, ***spacy_span_kwargs*)

A SpanFeaturizer that yields all matches for a given NER tag according to spaCy

Parameters

Name	Type	Default	Info
tag	`Union[str, List[str]]`		The set of tags to look for.
attr	`str`	`'ent_type_'`	The spaCy attribute to compare to the tags (e.g., `ent_type_`).
model			The model to load into spaCy (only supports models in strap).
fields			The fields in which to look for candidates.
match_longest	`bool`	`True`	If True, return contiguous chunks of a tag as a single candidate.

operators.candidates.extractor_spacy.TokenSpanFeaturizer

class operators.candidates.extractor_spacy.TokenSpanFeaturizer(*field*, *tokenizer='spacy'*, ***spacy_span_kwargs*)

A SpanFeaturizer that yields every token, given a selected tokenization strategy

Given a valid tokenization strategy, this operator will tokenize the input dataframe into spans based on the produced tokens.

Parameters

Name	Type	Default	Info
field	`str`		The dataframe column to apply the tokenization strategy over.
tokenizer	`str`	`'spacy'`	The tokenizer strategy (one of “spacy” or “whitespace”).

operators.candidates.span_preview.SpanPreviewPreprocessor

class operators.candidates.span_preview.SpanPreviewPreprocessor(*char_window=200, feature_suffix=''*)

Operator that adds a field with nearby textual features for each extracted span.

This Operator adds a preview of the parent document to the Span dataframe. This is useful when one needs to classify spans based on the surrounding context.

Parameters

Name	Type	Default	Info
char_window	`int`	`200`	The size of window (in characters), defaults to 200.
feature_suffix	`Optional[str]`	`''`	String to suffix the default feature names of this operator's output schema.

operators.embedding.EmbeddingCandidateFeaturizer

class operators.embedding.EmbeddingCandidateFeaturizer(*field*, *candidate_field*, *target_field=None*, *embedding_type='simcse'*)

Featurizer that converts text to an embedding.

operators.embedding.EmbeddingFeaturizer

class operators.embedding.EmbeddingFeaturizer(*field, target_field=None, embedding_type='simcse'*)

Featurizer that converts text to an embedding.

Parameters

Name	Type	Default	Info
field	`str`		The name of the column containing the text to embed.
target_field	`Optional[str]`	`None`	The name of the column to store the embedding in.
embedding_type	`str`	`'simcse'`	The type of embedding to use. Options are “simcse” and “spacy”.

operators.spacy.NounChunkFeaturizer

class operators.spacy.NounChunkFeaturizer(*field*, *target_field=None*, ***spacy_span_kwargs*)

A Featurizer that yields all noun phrases according to spaCy.

Used by the Seq Tagging application template as one of the base candidate generators for producing embeddings from.

Parameters

Name	Type	Default	Info
field	`str`		The field to parse into noun chunks.
target_field	`Optional[str]`	`None`	The field in which to store the extracted noun chunk candidates.
spacy_span_kwargs	`Any`		kwargs to pass to the spaCy model.

operators.spacy.SentenceFeaturizer

class operators.spacy.SentenceFeaturizer(*field, target_field=None, **spacy_span_kwargs*)

A Featurizer that yields all sentences according to spaCy.

Used by the Seq Tagging application template as one of the base candidate generators for producing embeddings from.

Parameters

Name	Type	Default	Info
field	`str`		The field to parse into sentences.
target_field	`Optional[str]`	`None`	The field in which to store the extracted sentence candidates.
spacy_span_kwargs	`Any`		kwargs to pass to the spaCy model.

operators.spacy.SpacyPreprocessor

class operators.spacy.SpacyPreprocessor(*field, target_field='doc', model='en_core_web_sm', disable=None, **spacy_kwargs*)

Preprocessor that parses document and adds json doc column.

Used by Sequence Tagging applications to add additional document metadata.

Parameters

Name	Type	Default	Info
field	`str`		The field to parse with spacy.
target_field	`str`	`'doc'`	The field in which to store the parsed doc object.
model	`str`	`'en_core_web_sm'`	The model to load into spaCy (only supports models in strap).
disable	`Optional[List[str]]`	`None`	Optional list of pipeline steps to disable.
spacy_kwargs	`Dict[str, Any]`		Kwargs to forward to the spacy.load function.

operators.spacy.SpacyTokenizer

class operators.spacy.SpacyTokenizer(*text_field, tokens_field=None*)

Preprocessor that parses document and adds tokens column.

Used by Sequence Tagging applications to add additional document metadata.

Parameters

Name	Type	Default	Info
text_field	`str`		The field to parse with spacy.
tokens_field	`Optional[str]`	`None`	The field in which to store the tokens list object.

operators.spacy.VerbPhraseFeaturizer

class operators.spacy.VerbPhraseFeaturizer(*field, target_field=None, min_length=2, **spacy_span_kwargs*)

A Featurizer that yields all verb phrases according to a simple part-of-speech verb match.

Used by the Seq Tagging application template as one of the base candidate generators for producing embeddings from.

Parameters

Name	Type	Default	Info
field	`str`		The field to parse into verb phrases.
target_field	`Optional[str]`	`None`	The field in which to store the extracted verb phrase candidates.
min_length	`int`	`2`	The minimum number of tokens for extracted verb phrases. Used to avoid single token phrases. Default: 2.
spacy_span_kwargs	`Any`		kwargs to pass to the spaCy model.

operators.special_char.AsciiCharFilter

class operators.special_char.AsciiCharFilter(*field*)

Preprocessor that removes non-ascii chars from selected column in place.

operators.special_char.LatinCharFilter

class operators.special_char.LatinCharFilter(*field*)

Preprocessor that removes non-latin chars from selected column in place. The following unicode ranges are not filtered: Basic_Latin: U+0000-U+007F; Latin-1_Supplement: U+0080-U+00FF; Latin_Extended-A: U+0100-U+017F; Latin_Extended-B: U+0180-U+024F; Latin_Extended_Additional: U+1E00-U+1EFF.

operators.truncate.TruncatePreprocessor

class operators.truncate.TruncatePreprocessor(*field, target_field=None, length=5000, by='words'*)

Truncates a column by given amount.

Truncates the given column (with string values) and writes result to the target field.

Parameters

Name	Type	Default	Info
field	`str`		The field you want to truncate.
target_field	`str`		The output field to write the truncated strings to.
length	`int`		The length to truncate to.
by	`str`		Whether to truncate to <length> 'words' or 'chars'.

operators.whitespace.WhitespacePreprocessor

class operators.whitespace.WhitespacePreprocessor(*fields, to_replace=None, output_field_suffix=''*)

Preprocessor that normalizes whitespace.

This operator finds all of the different types of whitespace in a given text field and normalizes it to the regular space character (U+0020). By default, the following non-standard space characters with the regular space: U+00A0, U+2000 to U+200A, U+202F, U+205F, U+3000. See https://en.wikipedia.org/wiki/Whitespace_character for more details on what these UTF-8 code points mean.

Parameters

Name	Type	Default	Info
fields	`List[str]`		The fields to apply whitespace pre-processing to.
to_replace	`Optional[str]`	`None`	A string containing all characters to be replaced with a regular whitespace (U+0020).
output_field_suffix	`Optional[str]`	`''`	To avoid updating in place, optionally specify a suffix to add to specified fields.

operators.candidates.filter.ExtractedSpanFilter

class operators.candidates.filter.ExtractedSpanFilter(*prediction_col='preds', negative_label=0*)

A filter that removes all spans with a negative prediction.

Parameters

Name	Type	Default	Info
prediction_col	`str`	`'preds'`	The column contains (integer representation) prediction.
negative_label	`int`	`0`	The (integer representation) of the NEGATIVE label.

operators.candidates.filter.RegexSpanFilter

class operators.candidates.filter.RegexSpanFilter(*regex*, *field='span_text'*, *ignore_case=False*)

A filter that REMOVES all candidates that match a given regular expresion

operators.filter.LabelFilter

class operators.filter.LabelFilter(*label_strs, filter_type='include', label_col='preds_str'*)

A filter that includes/excludes all datapoints corresponding to specified label strings.

This operator either includes or excludes datapoints in passing dataframes that have string labels in the set of provided :label_strs.

Parameters

Name	Type	Default	Info
label_strs	`List[str]`		The list of strings to use when applying the filter.
filter_type	`str`	`'include'`	The type of filter to apply (either “include” or “exclude”, default: {FilterTypes.INCLUDE}).
label_col	`str`	`'preds_str'`	The column in the passing dataframe to look for label integers (default: {ModelCols.PREDICTION_STR}).

operators.filter.LabelIntFilter

class operators.filter.LabelIntFilter(*label_ints, filter_type='include', label_col='preds'*)

A filter that includes/excludes all datapoints corresponding to specified label ints.

This operator either includes or excludes datapoints in passing dataframes that have integer labels in the set of provided :label_ints.

Parameters

Name	Type	Default	Info
label_ints	`List[int]`		The list of integers to use when applying the filter.
filter_type	`str`	`'include'`	The type of filter to apply (either “include” or “exclude”).
label_col	`str`	`'preds'`	The column in the passing dataframe to look for label integers.

operators.filter.MultiLabelFilter

class operators.filter.MultiLabelFilter(*label_strs, filter_type='include', label_col='preds_str'*)

A filter that includes/excludes all datapoints corresponding to specified label strings for multilabel classification.

This operator either includes or excludes datapoints in passing DataFrames that have string labels in the set of provided *label_strs*, depending on whether those labels are marked as PRESENT or ABSENT.

Parameters

Name	Type	Default	Info
label_strs	`Dict[str, str]`		Dictionary of string labels and whether they are present/absent to use when applying the filter.
filter_type	`str`	`'include'`	The type of filter to apply (either “include” or “exclude”, default: {FilterTypes.INCLUDE}).
label_col	`str`	`'preds_str'`	The column in the passing dataframe to look for label integers (default: {ModelCols.PREDICTION_STR}).

operators.row_filter.BooleanColumnBasedRowFilter

class operators.row_filter.BooleanColumnBasedRowFilter(*boolean_col_name, filter_type='include'*)

Filters rows based on the specified boolean column.

This operator allows one to either include or exclude rows based on the the boolean values present in a column in the DataFrame.

Parameters

Name	Type	Default	Info
boolean_col_name	`str`		The name of the dataframe column containing the boolean values used for filtering.
filter_type	`str`	`'include'`	One of ({FilterTypes.INCLUDE}, {FilterTypes.EXCLUDE}), defaults to {FilterTypes.INCLUDE}.

operators.row_filter.PandasQueryFilter

class operators.row_filter.PandasQueryFilter(*query*)

Includes or excludes rows based on a pandas query

This operator allows one to specify an arbitary Pandas DataFrame query and have that query applied over the input DataFrame, filtering the DataFrame in the process.

Parameters

Name	Type	Default	Info
query	`str`		The Pandas DataFrame query to perform, see https://pandas.pydata.org/docs/reference/api/pandas.DataFr

operators.row_filter.RegexRowFilter

class operators.row_filter.RegexRowFilter(*regex_pattern, text_field=None, case_sensitive=False*)

Filters rows based on the regex pattern provided. This is applied over the *rich_doc_text* field by default.

This operator allows users to search for a regex pattern over a text field. If the regex pattern is not present in the text for a row, the row is dropped from the Pandas DataFrame.

This operator is commonly used to filter rows (pages or documents) based on keywords in PDF extraction applications.

Parameters

Name	Type	Default	Info
regex_pattern	`str`		The regular expression pattern we use to filter rows.
text_field	`str`		The text field over which we perform the regex match.
case_sensitive	`bool`	`False`	If False, ignore case when considering regular expression matches (defaults to False).

operators.row_filter.TextSizeFilter

class operators.row_filter.TextSizeFilter(*field*, *max_text_size*)

A filter that excludes rows with a text column larger than specificed size (in KB)

Used by Sequence Tagging applications to filter large texts

Parameters

Name	Type	Default	Info
field	`str`		The text field to filter rows on.
max_text_size	`int`		Maximum text size in KB.

operators.change_datapoint.ChangeDatapoint

class operators.change_datapoint.ChangeDatapoint(*datapoint_type, datapoint_cols=None, sorted=False*)

Preprocessor that changes the datapoint type/columns.

Parameters

Name	Type	Default	
datapoint_type	`str`		The data type
datapoint_cols	`Optional[NewType(DataframeMultiFieldType, List[str])]`	`None`	The data colu
sorted	`bool`	`False`	Wh the data will be s afte cha ove to t cha only data type

operators.concat.ConcatColumns

class operators.concat.ConcatColumns(*join_type='inner', suffix='', drop_duplicates=False*)

Processor that concatenates columns of dataframes.

operators.concat.ConcatRows

class operators.concat.ConcatRows

Processor that concatenates dataframes along the index axis.

operators.drop.ColumnDropper

class operators.drop.ColumnDropper(*fields*)

Processor that drops given columns from the DataFrame.

Given a list of column names, this operator will drop those columns from the passing DataFrame. Invalid fields will be ignored.

Parameters

Name	Type	Default	Info
fields	`List[str]`		The list of columns to be dropped.

operators.filler.ColumnFiller

class operators.filler.ColumnFiller(*field, value*)

Operator that replaces values in either a new or existing column in the DataFrame with the given value

Parameters

Name	Type	Default	Info
field	`str`		The field whose values will be replaced.
value	`Any`		The value to insert into the specified field.

operators.identity.IdentityOperator

class operators.identity.IdentityOperator

No-op operator

operators.rename.ColumnRenamer

class operators.rename.ColumnRenamer(*column_map*)

Preprocessor that renames columns

This operator is used to rename columns in the passing dataframe according to a provided renaming dictionary

Parameters

Name	Type	Default	Info
column_map	`Dict[str, str]`		A dictionary mapping old column names to new column names.

operators.scaler.StandardScaler

class operators.scaler.StandardScaler(*field*, *target_field=None*, *mean=0.0*, *std=1.0*)

Preprocessor that scales a numerical column to mean=0 and std=1.

operators.table.TableConverter

class operators.table.TableConverter(*field, output_field_suffix=''*)

Feautizer that convert an array of dicts into a custom Table object

Parameters

Name	Type	Default	Info
field	`str`		A field of array of dicts to cast to a Table type.
output_field_suffix	`str`	`''`	(Optional) To avoid updating in place.

operators.trace.TraceToStepFlattener

class operators.trace.TraceToStepFlattener(*field*)

Flattens a nested JSON field that contains a hierarchical trace into individual steps.

The input JSON is expected to follow a schema with step_type, metadata, value and substeps. Each step in the trace will be flattened into a separate row.

Parameters

Name	Type	Default	Info
field	`str`		The field containing the JSON trace to flatten.

operators.post_processors.sequence_tagging_post_processors.SpanFilterByLengthPostProcessor

class
operators.post_processors.sequence_tagging_post_processors.SpanFilterByLengthPostProcessor(*span_length*)

Filter positive spans with length <= the provided span_length

operators.post_processors.sequence_tagging_post_processors.SpanMergeByNumberCharacterPostProcessor

class
operators.post_processors.sequence_tagging_post_processors.SpanMergeByNumberCharacterPostProcessor(*lower_bound_number_character, upper_bound_number_character*)

Merge same-label nearby spans if the negative-label text in between spans is in [lower, upper] number of characters (inclusive).

operators.post_processors.sequence_tagging_post_processors.SpanMergeByRegexPatternPostProcessor

class operators.post_processors.sequence_tagging_post_processors.SpanMergeByRegexPatternPostProcessor(*regex_pattern, field*)

Merge same-label nearby spans if the negative-label text in between spans matches this regex pattern. For the Field, please select your document field.

Parameters

Name	Type	Default	Info
field	`str`		Your seq input field.
regex_pattern	`str`		Merge nearby spans with same labels if the negative-labeled text in between spans matches this regex pattern.

operators.post_processors.sequence_tagging_post_processors.SpanRegexPostProcessor

class
operators.post_processors.sequence_tagging_post_processors.SpanRegexPostProcessor(*regex_pattern*, *predict_label*, *field*)

Post Porcessor that labels anything that matches the provided pattern with the provided label

Parameters

Name	Type	Default	Info
regex_pattern	`str`		The regex pattern that we should.
field	`str`		The field over which extraction was run.
predict_label	`int`		What to label text that matches the provided pattern.

operators.post_processors.sequence_tagging_post_processors.SpanRemoveWhitespacePostProcessor

class operators.post_processors.sequence_tagging_post_processors.SpanRemoveWhitespacePostProcessor(*field*)

Remove leading & trailing whitespace in positive spans. For the Field, please select your document field.

operators.post_processors.sequence_tagging_post_processors.SubstringExpansionPostProcessor

class
operators.post_processors.sequence_tagging_post_processors.SubstringExpansionPostProcessor(*field*, *tokenized_field*)

This postprocessor expands predictions to the token boundaries should a prediction boundary fall mid-token.

Parameters

Name	Type	Default	Info
field	`str`		The field over which the model has made predictions.
tokens_field			The field in which the tokens list object is stored.

operators.candidates.linker.EntityDictLinker

class operators.candidates.linker.EntityDictLinker(*entity_dict_path, ignore_case=False*)

Maps span_text to span_entity given an entity to alias dictionary.

operators.candidates.normalizer.DateSpanNormalizer

class operators.candidates.normalizer.DateSpanNormalizer

Normalizes date spans into their canonical forms, e.g. 2020-01-01.

Returns

The normalized span after post-processing

Return type

`{SpanCols.NORMALIZED_SPAN}`

operators.candidates.normalizer.IdentitySpanNormalizer

class operators.candidates.normalizer.IdentitySpanNormalizer

Copies span text as is.

Returns

The normalized span after post-processing

Return type

`{SpanCols.NORMALIZED_SPAN}`

operators.candidates.normalizer.NumericalSpanNormalizer

class operators.candidates.normalizer.NumericalSpanNormalizer

Normalizes numerical cardinals (1, 2, etc) to string values (one, two, etc).

Returns

The normalized span after post-processing

Return type

`{SpanCols.NORMALIZED_SPAN}`

operators.candidates.normalizer.OrdinalSpanNormalizer

class operators.candidates.normalizer.OrdinalSpanNormalizer

Normalizes numerical ordinals (1st, 2nd, etc) to string ordinals (first, second, etc).

Returns

The normalized span after post-processing

Return type

`{SpanCols.NORMALIZED_SPAN}`

operators.candidates.normalizer.SpanEntityNormalizer

class operators.candidates.normalizer.SpanEntityNormalizer

Copies the linked span entity from the extractor as the normalized span.

Returns

The normalized span after post-processing

Return type

`{SpanCols.NORMALIZED_SPAN}`

operators.candidates.normalizer.TextCasingSpanNormalizer

class operators.candidates.normalizer.TextCasingSpanNormalizer

Normalizes spans by lowercasing them, then capitalizing the first letter of each word.

Returns

The normalized span after post-processing

Return type

`{SpanCols.NORMALIZED_SPAN}`

operators.candidates.normalizer.USCurrencySpanNormalizer

class operators.candidates.normalizer.USCurrencySpanNormalizer

Normalizes US currency spans into their numerical values.

Returns

The normalized span after post-processing

Return type

`{SpanCols.NORMALIZED_SPAN}`

operators.candidates.reducer.DocumentFirstReducer

class operators.candidates.reducer.DocumentFirstReducer(*preds_col='preds', probs_col='probs'*)

Reduces span predictions in a document to the span which occurs first.

operators.candidates.reducer.DocumentLastReducer

class operators.candidates.reducer.DocumentLastReducer(*preds_col='preds'*, *probs_col='probs'*)

Reduces spans predictions in a document to the span which occurs last.

operators.candidates.reducer.DocumentMostCommonReducer

class operators.candidates.reducer.DocumentMostCommonReducer(*preds_col='preds'*, *probs_col='probs'*)

Reduces spans predictions in a document to the span which occurs most frequently.

operators.candidates.reducer.DocumentMostConfidentReducer

class operators.candidates.reducer.DocumentMostConfidentReducer(*preds_col='preds', probs_col='probs'*)

Reduces spans predictions in a document to the span with the most confident model prediction.

operators.candidates.reducer.EntityFirstReducer

class operators.candidates.reducer.EntityFirstReducer(*preds_col='preds', probs_col='probs'*)

Reduces span predictions for a document entity to the first occuring span of that entity.

operators.candidates.reducer.EntityLastReducer

class operators.candidates.reducer.EntityLastReducer(*preds_col='preds'*, *probs_col='probs'*)

Reduces span predictions for a document entity to the last occuring span of that entity.

operators.candidates.reducer.EntityMeanPredictionReducer

class operators.candidates.reducer.EntityMeanPredictionReducer(*preds_col='preds'*, *probs_col='probs'*)

Reduces span predictions for a document entity by computing the mean prediction.

operators.candidates.reducer.EntityMostCommonReducer

class operators.candidates.reducer.EntityMostCommonReducer(*tie_break='first (no tiebreak)'*, *preds_col='preds'*)

Reduces span predictions for a document entity to the prediction of the majority vote class.

operators.candidates.reducer.EntityMostConfidentReducer

class operators.candidates.reducer.EntityMostConfidentReducer(*preds_col='preds'*, *probs_col='probs'*)

Reduces span predictions for a document entity to the most confident prediction.

operators.candidates.reducer.IdentityReducer

class operators.candidates.reducer.IdentityReducer(*preds_col='preds', probs_col='probs'*)

No-Op Reducer that passes all Spans through

snorkelflow.client

Interfaces to interact with the Snorkel Flow REST API.

Most of the functions in the `snorkelflow.client` module require a *client context* object — `SnorkelFlowContext` — that points to the Snorkel Flow instance:

```
import snorkelflow.client as sf
ctx = sf.SnorkelFlowContext.from_kwargs()
```

All the functions under submodules are also available under `snorkelflow.client`.

Examples

```
import snorkelflow.client as sf
# get_node is available under snorkelflow.client.nodes
sf.nodes.get_node(node)
# also available under snorkelflow.client (recommended)
sf.get_node(node)
```

Since `snorkelflow.client` submodules may be reorganized in the future, we recommend accessing functions directly from `snorkelflow.client` to minimize the risk of future breaking changes.

Submodules

`snorkelflow.client.analyses`	Analysis related functions.
`snorkelflow.client.annotations`	Annotations-related SDK functions.
`snorkelflow.client.annotation_sources`	Annotation source related functions.
`snorkelflow.client.applications`	Application related functions.

`snorkelflow.client.batches`	(Annotation) batch related functions.
`snorkelflow.client.blocks`	Block related functions.
`snorkelflow.client.comments`	Comment related functions.
`snorkelflow.client.ctx`	Context related classes.
`snorkelflow.client.custom_pip`	Custom pip install related functions.
`snorkelflow.client.datasets`	Dataset related functions.
`snorkelflow.client.dataset_views`	Dataset views functions for datasets
`snorkelflow.client.datasources`	Datasource related functions.
`snorkelflow.client.evaluation`	Evaluation related functions.
`snorkelflow.client.external_models`	External model endpoints related functions.
`snorkelflow.client.files`	File storage related functions to upload and download files and directories.
`snorkelflow.client.file_storage_configs`	File storage config related functions.
`snorkelflow.client.fm_suite`	Foundation model suite related functions.

`snorkelflow.client.gts`	Ground truth related functions.
`snorkelflow.client.lf_packages`	LF package related functions.
`snorkelflow.client.lf_templates`	LF template related functions.
`snorkelflow.client.lfs`	LF related functions.
`snorkelflow.client.metrics`	Metric related functions.
`snorkelflow.client.models`	Model related functions.
`snorkelflow.client.nodes`	Node related functions.
`snorkelflow.client.object_storage`	Minio related functions.
`snorkelflow.client.operators`	Operator related functions.
`snorkelflow.client.secrets`	Secret store related functions.
`snorkelflow.client.studio`	Studio related functions.
`snorkelflow.client.synthetic`	Synthetic data related functions for generating synthetic data.
`snorkelflow.client.tags`	Tag-related functions.

`snorkelflow.client.training_sets`	Training set related functions.
`snorkelflow.client.transfer`	SDK functions for transferring assets between applications or nodes on a single Snorkel Flow instance.
`snorkelflow.client.utils`	Utility functions.
`snorkelflow.client.users`	User related functions.
`snorkelflow.client.workflows`	Workflow related functions.

snorkelflow.client.analyses

Analysis related functions.

Functions

get_ngram_counts(node, df, field[, label, ...])	Return a dictionary mapping n-gram name to its count in dataset.
plot_conflicts_matrix(node[, split, ...])	View matrix that shows conflicts among labels assigned by LFs.
plot_distinctive_keywords(node, field[, ...])	View the most distinctive ngrams' histograms after re-fetching the data.
plot_distinctive_ngram_histogram(node, ...)	Return a list of n-grams with higher relative count for a class than the overall dataset.
plot_distributions(node, df, numeric_field)	Plot the distribution of a numeric feature, optionally on a per-class basis.
plot_feature_combination_heatmap(node, df, ...)	Generate a heatmap of a metric calculated across combinations of 2 features.
plot_feature_importance(node, model_uid[, split])	Generates a SHAP based summary plot.
plot_histograms(node[, split, column_names, ...])	Generates and displays interactive histograms for features.
plot_overlap_matrix(node[, split, ...])	View matrix that shows overlaps among labels assigned by LFs.
plot_scatter(node[, split, plot_type, ...])	Generates a scatter plot of the requested type.
plot_scatterplot(node, df, x_axis_field, ...)	Plot the scatterplot of datapoints across two fields.

snorkelflow.client.analyses.get_ngram_counts

snorkelflow.client.analyses.get_ngram_counts(*node, df, field, label=None, stop_words='english', ngram_range=(1, 3)*)

Return a dictionary mapping n-gram name to its count in dataset.

Filter by ground truth if specified.

Examples

For example, if we want to see all the n-grams that have a significantly higher relative count for `EMPLOYMENT` than for all classes in the dev split.

```
import matplotlib.pyplot as plt
%matplotlib inline

df = sf.get_dataset(node, split="dev")

counts_dict = sf.get_ngram_counts(node, df, field="text")
class_counts_dict = sf.get_ngram_counts(
    node,
    df,
    field="text",
    label="employment"
)
sf.plot_distinctive_ngram_histogram(
    node,
    counts_dict,
    class_counts_dict,
    num_ngrams=10
)
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
df	`DataFrame`		The data frame with a text field over which we calculate n-gram counts.

field	`str`		Text field in df over which we calculate n-gram counts.
label	`Optional[str]`	`None`	Label string to use to filter data points where ground truth is this value for calculating n-gram counts.
stop_words	`Optional[str]`	`'english'`	Words to ignore when building n-gram counts. Default to `"english"`. See `sklearn.CountVectorizer` documentation.
ngram_range	`Optional[Tuple]`	`(1, 3)`	The lower and upper boundary of the range of n-grams to consider. See `sklearn.CountVectorizer` documentation.

Raises

ValueError – If label in not in the set of valid labels for task If field in not in the set of columns in df

Returns

A dictionary where n-gram names are keys and counts are values

Return type

`dict`

snorkelflow.client.analyses.plot_conflicts_matrix

snorkelflow.client.analyses.plot_conflicts_matrix(*node, split='dev', normalize=True, colorbar=True, lf_uids_filter=None*)

View matrix that shows conflicts among labels assigned by LFs.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
split	`Optional[str]`	`'dev'`	Split to be loaded. Default to `"dev"`
normalize	`Optional[bool]`	`True`	Whether to normalize by total number of datapoints. Default to `True`
colorbar	`Optional[bool]`	`True`	Whether to show colorbar. Default to `True`
lf_uids_filter	`Optional[List[int]]`	`None`	Inclusive list of uids which should be included.

Return type

`None`

snorkelflow.client.analyses.plot_distinctive_keywords

snorkelflow.client.analyses.plot_distinctive_keywords(*node, field, split='dev', label=None, overall_keyword_kwargs=None, class_keyword_kwargs=None, keyword_histogram_kwargs=None*)

View the most distinctive ngrams' histograms after re-fetching the data.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
field	`str`		The field to extract te
split	`str`	`'dev'`	The split to get summ Only *"dev"*, *"valid"*, ar splits are allowed. De *"dev"*.
label	`Optional[str]`	`None`	The class label for wh extract distinctive ng by default, meaning t
overall_keyword_kwargs	`Optional[Dict[str, Any]]`	`None`	Kwargs for the overa counts phase (refer t documentation on *get_ngram_counts* fo *None* by default.
class_keyword_kwargs	`Optional[Dict[str, Any]]`	`None`	Kwargs for the class- ngram counts phase documentation on *get_ngram_counts* fo *None* by default.

keyword_histogram_kwargs	`Optional[Dict[str, Any]]`	`None`	Kwargs for the histog (refer to documentat *plot_distinctive_ngra* for details). *None* by

Returns

Returns a list of distinctive ngrams.

Return type

`Dict[str, List[str]]`

snorkelflow.client.analyses.plot_distinctive_ngram_histogram

snorkelflow.client.analyses.plot_distinctive_ngram_histogram(*node, overall_count_dict, class_count_dict, num_ngrams=5, min_appearance=2, threshold=0.001, class_name=None*)

Return a list of n-grams with higher relative count for a class than the overall dataset.

> SEE ALSO
>
> Use `get_ngram_counts()` to calculate the overall and class-specific counts, and pass them in as arguments here.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
overall_count_dict	`Dict[str, int]`		Dictionary mapping n-gram name and count, should be for the overall dataset.
class_count_dict	`Dict[str, int]`		Dictionary mapping n-gram name and count, should be for the a specific class.
num_ngrams	`Optional[int]`	`5`	Number of distinctive ngrams to return.
min_appearance	`Optional[int]`	`2`	Smallest nmber of times an n-gram has to appear to be counted.
threshold	`Optional[float]`	`0.001`	Smallest percentage difference in frequency an n-gram has to have between the class and the overall dataset. Default to `0.001`.

class_name	`Optional[str]`	`None`	The name of the class whose histogram is being plotted.

Returns

A dictionary where n-gram names are keys and counts are values

Return type

`list`

snorkelflow.client.analyses.plot_distributions

snorkelflow.client.analyses.plot_distributions(*node, df, numeric_field, label_field=None, drop_unknown=True, hist_kwargs=None, axes_kwargs=None*)

Plot the distribution of a numeric feature, optionally on a per-class basis.

One way to identify promising LFs over numeric features is to plot the distributions by class and identify a range and/or threshold that separates one class from the others. For example, we can see that emails with more images attached are more likely to be spam.

Examples

```
df = sf.get_dataset(node, split="dev")
sf.plot_distributions(node, df, "num_images", label_field="label")
```

Parameters

Name	Type	Default	
node	`int`		UID of the node.
df	`DataFrame`		The data frame with a numeric fie
numeric_field	`str`		Name of the field in the df that co
label_field	`Optional[str]`	`None`	Name of the field containing labe
drop_unknown	`bool`	`True`	If `True` and `label_field` is not on a per-class basis.
hist_kwargs	`Optional[Dict[str, Any]]`	`None`	Arguments to pass through to ma (https://matplotlib.org/3.2.1/api/_ Commonly used hist_kwargs: `bins`: Specify the number `log`: Plot the y-axis on a lo `range`: Specifies an upper

axes_kwargs	`Optional[Dict[str, Any]]`	`None`	Arguments to pass through to ma (https://matplotlib.org/api/_as_ge Commonly used axes_kwargs: `title`: Add a title to the pl `xlabel`: Add an x-axis labe `ylabel`: Add a y-axis label

Raises

- **ValueError** – If numeric_field is not one of the columns in the given DataFrame
- **ValueError** – If label_field is given and not one of the columns in the given DataFrame

Return type

`None`

snorkelflow.client.analyses.plot_feature_combination_heatmap

snorkelflow.client.analyses.plot_feature_combination_heatmap(*node*, *df*, *label_str*, *label_field*, *metric='logistic_regression'*, *excluded_features=None*, *included_features=None*, *plot_title=None*, *figure_size=None*, *top_k_features=None*)

Generate a heatmap of a metric calculated across combinations of 2 features.

The values generated in the heatmap represent the score of the selected metric, details about how the scores are calculated can be found under the 'metric' documentation.

Parameters

Name	Type	Default	
node	`int`		UID of t
df	`DataFrame`		The dat with a n field ove plot the scatterp
label_str	`str`		Name o label to calculat metric v
label_field	`str`		Name o containi for plott per-clas
metric	`str`	`'logistic_regression'`	The met when ca values t the heat Current 'median

			‘linear_r and ‘logistic_ are sup ‘mean’ v on each combina use the differen betwee of the ta and the all othe the new space. * will run each fea combina use the differen betwee median target la the med other la new fea * ‘linear_r will run regresso feature combina a one vs schema the targ and use the pred ‘logistic_

			will run regressi feature combina a one vs schema the targ and use score of regresso
excluded_features	`Optional[List[str]]`	`None`	A list of exclude heatma features this list plotted. parame passed, 'include cannot l provide default, features exclude
included_features	`Optional[List[str]]`	`None`	A list of include heatma features list will b If this pa passed, 'exclude cannot l provide default,

			features included
plot_title	`Optional[str]`	`None`	An optic for the h
figure_size	`Optional[Tuple[int, int]]`	`None`	A tuple the size with wh generat heatma default, allocate feature.
top_k_features	`Optional[int]`	`None`	If not N number features Feature selected identifyi perform combina accordi defined

Raises

- **ValueError** – If label_str is not a valid label
- **ValueError** – If label_field is not one of the columns in the given DataFrame
- **ValueError** – If both excluded_features and included_features are provided
- **ValueError** – If top_k_features is provided and not positive
- **RuntimeWarning** – If an encountered feature combination has a variance of 0, and thus cannot have PCA or normalization applied to them. Such feature combinations will show up as blank on the heatmap.

Return type

`None`

snorkelflow.client.analyses.plot_feature_importance

snorkelflow.client.analyses.plot_feature_importance(*node, model_uid, split='valid'*)

Generates a SHAP based summary plot.

> NOTE
> This function requires these additional dependencies to be installed: shap, transformers, xgboost. They are pre-installed in the in-platform notebook server but may not be present in your own Python environment. Please make sure to have them installed. We've tested with shap==0.40.0, transformers==4.27.4, xgboost==1.1.0.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
model_uid	`int`		UID of the model to get important features for.
split	`str`	`'valid'`	The split to get a visualization for.

Raises

NotSupportedException – For models other than LogReg and XGBoost.

Return type

`None`

snorkelflow.client.analyses.plot_histograms

snorkelflow.client.analyses.plot_histograms(*node, split='dev', column_names=None, drop_unknown=True*)

Generates and displays interactive histograms for features.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
split	`str`	`'dev'`	The split to get data from. Only *"dev"*, *"valid"*, and *"test"* splits are allowed. Default to *"dev"*.
column_names	`Optional[List[str]]`	`None`	The names of columns to plot. *None* by default, meaning all columns will be plotted.
drop_unknown	`bool`	`True`	Whether datapoints with unknown labels should be dropped from the plot. Defaults to True.

Return type

`None`

snorkelflow.client.analyses.plot_overlap_matrix

snorkelflow.client.analyses.plot_overlap_matrix(*node, split='dev', self_overlaps=False, normalize=True, colorbar=True, lf_uids_filter=None*)

View matrix that shows overlaps among labels assigned by LFs.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
split	`Optional[str]`	`'dev'`	Split to be loaded. Default to `"dev"`
self_overlaps	`Optional[bool]`	`False`	Whether to show overlaps of one LF with itself. Default to `False`
normalize	`Optional[bool]`	`True`	Whether to normalize by total number of datapoints. Default to `True`
colorbar	`Optional[bool]`	`True`	Whether to show colorbar. Default to `True`
lf_uids_filter	`Optional[List[int]]`	`None`	Inclusive list of uids which should be included.

Return type

`None`

snorkelflow.client.analyses.plot_scatter

snorkelflow.client.analyses.plot_scatter(*node*, *split='dev'*, *plot_type='2d'*, *column_names=None*, *drop_unknown=True*)

Generates a scatter plot of the requested type.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
split	`str`	`'dev'`	The split to get data from. Only *“dev”*, *“valid”*, and *“test”* splits are allowed. Default to *“dev”*.
plot_type	`str`	`'2d'`	Type of scatter plot to generate. Defaults to ‘2d’, but can also be ‘2d_matrix’ or ‘3d’.
column_names	`Optional[List[str]]`	`None`	The names of the columns to plot. *None* by default, meaning that the first 2 or 3 columns will be used.
drop_unknown	`bool`	`True`	Whether datapoints with unknown labels should be dropped from the plot. Defaults to True.

Raises

ValueError – If column_names doesn’t have the right number of columns, or if column_names is *None*, sufficient columns aren’t available. Also raises this error if calling with an unsupported plot type.

Return type

`None`

snorkelflow.client.analyses.plot_scatterplot

snorkelflow.client.analyses.plot_scatterplot(*node, df, x_axis_field, y_axis_field, label_field, drop_unknown=True, display_legend=True, scatter_kwargs=None, axes_kwargs=None, filter_flag_column=None, color_map='rainbow'*)

Plot the scatterplot of datapoints across two fields.

To find promising LFs involving a combination of numeric features, the `plot_scatterplot()` function can be used instead to visualize datapoints. Running `%matplotlib notebook` in your notebook prior to running this function will open up the plot in interactive mode, allowing for easier exploration of the data.

If you find a view while exploring a scatterplot in interactive mode that would make a promising bounding rectangle LF, then you can use the `add_scatterpot_lf` function to do so easily. Using the function in the next cell will automatically generate a LF from your current view of the scatterplot. For example, we can examine network data and easily creating a bounding LF for Youtube traffic as follows:

Examples

```
%matplotlib notebook
df = sf.get_dataset(node, split='dev')
axis_obj = sf.plot_scatterplot(node, df, 'Fwd.Packet.Length.Max',
'Avg.Fwd.Segment.Size', "ProtocolName")

# Run this cell after the view in the interactive scatterplot contains
mostly YOUTUBE labels
sf.add_scatterplot_lf(node, axis_obj, 'YOUTUBE')
```

Parameters

Name	Type	Default	
node	`int`		UID of the node.
df	`DataFrame`		The data frame with a nur
x_axis_field	`str`		Name of the field in the d
y_axis_field	`str`		Name of the field in the d

label_field	`str`		Name of the field containi
drop_unknown	`bool`	`True`	If `True`, remove example
display_legend	`bool`	`True`	If `True` display a legend c
scatter_kwargs	`Optional[Dict[str, Any]]`	`None`	Arguments to pass throug (https://matplotlib.org/api
axes_kwargs	`Optional[Dict[str, Any]]`	`None`	Arguments to pass throug (https://matplotlib.org/api Commonly used axes_kwa `title`: Add a title t `xlabel`: Edit x-axis `ylabel`: Edit y-axis
filter_flag_column	`Optional[str]`	`None`	Name of column describir accessed via show_filtered
color_map	`str`	`'rainbow'`	Name of the matplotlib co (https://matplotlib.org/3.1

Raises

- **ValueError** – If x_axis_field is not one of the columns in the given DataFrame
- **ValueError** – If y_axis_field is not one of the columns in the given DataFrame
- **ValueError** – If label_field is not one of the coluns in the given DataFrame
- **ValueError** – If filter_flag_column is not one of the coluns in the given DataFrame

Returns

An axis object containing information about the plot that can be passed into add_scatterplot_lf to quickly generate a bounding rect LF

Return type

`matplotlib.pyplot.axis`

snorkelflow.client.annotation_sources

Annotation source related functions.

Functions

create_annotation_source([username, ...])	Create an annotation source.
delete_annotation_source(source_name)	Delete an annotation source.
get_annotation_source([source_name, source_uid])	Get an annotation source from uid or name.
get_annotation_sources()	Get annotation sources.
update_annotation_source(source_name[, ...])	Update an annotation source.

snorkelflow.client.annotation_sources.create_annotation_source

snorkelflow.client.annotation_sources.create_annotation_source(*username=None, source_name=None, source_type=None, metadata=None*)

Create an annotation source.

Parameters

Name	Type	Default	Info
username	`Optional[str]`	`None`	The username of source if the source type is user.
source_name	`Optional[str]`	`None`	The name of the source, which defaults to username for user type of source.
source_type	`Optional[str]`	`None`	The type of source (user, aggregation, etc).
metadata	`Optional[dict]`	`None`	Any source metadata.

Returns

The created source.

Return type

`Source`

snorkelflow.client.annotation_sources.delete_annotation_source

snorkelflow.client.annotation_sources.delete_annotation_source(*source_name*)

Delete an annotation source.

Parameters

Name	Type	Default	Info
source_name	`str`		The name of the source.

Returns

Returns true if the operation succeeds.

Return type

`bool`

snorkelflow.client.annotation_sources.get_annotation_source

snorkelflow.client.annotation_sources.get_annotation_source(*source_name=None, source_uid=None*)

Get an annotation source from uid or name.

Parameters

Name	Type	Default	Info
source_name	`Optional[str]`	`None`	Name of the source (such as username, etc).
source_uid	`Optional[int]`	`None`	Uid of the source.

Return type

`A single source that matches name and/or uid`

snorkelflow.client.annotation_sources.get_annotation_sources

snorkelflow.client.annotation_sources.get_annotation_sources()

Get annotation sources.

Returns

A list of sources.

Return type

`List[Source]`

snorkelflow.client.annotation_sources.update_annotation_source

snorkelflow.client.annotation_sources.update_annotation_source(*source_name, new_source_name=None, metadata=None*)

Update an annotation source.

Parameters

Name	Type	Default	Info
source_name	`str`		The current name of the source.
new_source_name	`Optional[str]`	`None`	The new name of the source.
metadata	`Optional[dict]`	`None`	The updated metadata.

Returns

The updated source.

Return type

`Source`

snorkelflow.client.annotations

Annotations-related SDK functions. Annotations allow subject matter experts (SMEs) to manually label data. A small amount of manually-labeled ground truth data is essential in a programmatic workflow to guarantee the quality of the model's predictions, calculate meaningful metrics, and provide baselines for advanced automated functionality like automatic LF suggestions. Methods in this module allow users to create, edit, and delete annotations, as well as calculate annotation-related metrics. This module also provides methods for managing how annotations are registered to the platform as ground truth.

Functions

add_annotation(node, x_uid, label[, metadata])	Add an annotation for a document or a span.
add_annotations(node, annotations[, ...])	Add multiple annotations to a node.
aggregate_annotations(node, batch_name[, ...])	Combine annotations from multiple sources into a single set of annotations.
commit_annotations(node, source_name)	Commit annotations in a node as ground truth (GT).
delete_annotation(node, annotation_uids)	Deletes annotations from a list of annotation UIDs.
get_annotations(node[, source_name, x_uids, ...])	Get a collection of annotations for a particular node.
get_interannotator_agreement(node[, ...])	Get interannotator agreement statistics for a node.
update_annotation(node, annotation_uid, label)	Updates the value of an already-created annotation.

snorkelflow.client.annotations.add_annotation

snorkelflow.client.annotations.add_annotation(*node, x_uid, label, metadata=None*)

Add an annotation for a document or a span. The label for the annotation must be in the appropriate format expected by the model node specified by the “node” parameter. To learn more about label formats, see Format for ground truth interaction in the SDK.

Examples

```
>>> # Add a positive label for a document
>>> sf.add_annotation(node, "doc::1", "POS", metadata={"created by":
"sdk"})
{
    'annotation_uid': 12345,
    'source': {
        'source_uid': 1,
        'source_type': 'user',
        'source_name': 'my-name',
        'user_uid': 0,
        'metadata': {"created by": "sdk"}
    },
    'workspace_uid': 1
}
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
x_uid	`str`		UID of the document or span to be annotated.
label	`Any`		The label for annotation (“POS”, “NEG”, etc). The format of the label depends on the configuration of the model node specified by the “node” parameter.
metadata	`Optional[Dict]`	`None`	Any metadata that we want to save for the annotation.

Return type

`List[Dict[str, any]]`

snorkelflow.client.annotations.add_annotations

snorkelflow.client.annotations.add_annotations(*node, annotations, source_name=None, username=None*)

Add multiple annotations to a node. The “annotations” argument for this function is a list of dictionaries. The dictionaries must have an “x_uid” key, which is the UID of the document or span to be annotated. The dictionaries must also have a “label” key, which is the label for the annotation. The format of the label depends on the configuration of the model node specified by the “node” parameter. To learn more about label formats, see Format for ground truth interaction in the SDK.

At least one of `source_name` or `username` must be provided. A `username` is a name corresponding to a specific user, while an annotation “source” can be a non-user source of annotations.

Examples

```
>>> sf.add_annotations(model_node_uid,
>>>     annotations=[{"x_uid": "<x_uid>", "label": "<label>"}, ...],
>>>     username="<username>"
>>> )
[
    {
        "annotation_uid": <annotation_uid> # Every row/datapoint will
have a different uid
        "x_uid": <x_uid>,
        "source_uid": <source_uid>, # This is the username that was
passed in the function
    }
    ...
]
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node to add annotations to.

annotations	`List[Dict[str, Any]]`		The list of annotations to be added. Annotations must be dictionaries with an “x_uid” and “label” key.
source_name	`Optional[str]`	`None`	The name of the source for annotations. If this is None, then `username` must be provided.
username	`Optional[str]`	`None`	The name of the user for the annotations. If this is None, then `source_name` must be provided.

Return type

`List[Dict[str, any]]`

snorkelflow.client.annotations.aggregate_annotations

snorkelflow.client.annotations.aggregate_annotations(*node, batch_name, sources=None, strategy=None*)

Combine annotations from multiple sources into a single set of annotations. The "strategy" argument specifies how to combine the annotations; the "sources" argument specifies which sources to aggregate. If "sources" is None, then all sources will be aggregated. This function will return a list of new annotations that were created by the aggregation. This aggregated set of annotations will then show up in the Batches page in the UI.

Aggregation strategies differ based on what kind of task you are aggregating annotations for.

> - For multi-class classification tasks, the "simple_majority" strategy will select the label that was most frequently assigned to each data point.
> - For multi-label classification tasks, the "simple_union" strategy will take the union over all votes for all classes for each data point. Conflicts will be broken by selecting the vote that was most frequent.
> - For sequence tagging tasks, the "simple_intersection" strategy will label the intersection of the spans marked by all annotators as the final span.

Examples

```
>>> sf.aggregate_annotations(1, "test-batch-name", sources=["user1",
"user2"], strategy="simple_majority")
__DATAPOINT_UID  annotation_uid
doc::10005       4679185          {'annotation_uid': 4679185, 'x_uid':
'doc::100...
doc::10006       4679186          {'annotation_uid': 4679186, 'x_uid':
'doc::100...
doc::10007       4679187          {'annotation_uid': 4679187, 'x_uid':
'doc::100...
doc::10009       4679188          {'annotation_uid': 4679188, 'x_uid':
'doc::100...
Name: annotation, dtype: object
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node that we are committing the annotations to.
batch_name	`str`		The name of the batch where the annotations are being aggregated.
sources	`Optional[List[str]]`	`None`	[Optional] The list of sources where the annotations are being aggregated.
strategy	`Optional[str]`	`None`	The strategy to use for aggregation.

Return type

`List[Annotation]`

snorkelflow.client.annotations.commit_annotations

snorkelflow.client.annotations.commit_annotations(*node, source_name*)

Commit annotations in a node as ground truth (GT). This will make the annotations available in Studio. Note that you may have to resample data splits in Studio to see new ground truth appear. Committing ground truth may cause labeling function and model metrics to change. Committing ground truth to data points that already have ground truth will overwrite the existing ground truth.

Examples

```
>>> sf.commit_annotations(1, "Simple-majority-07/01/23-01:18:13")
# No output. Resampling the data split in Studio may be required to see
the new ground truth.
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node that we are committing the annotations to.
source_name	`str`		The name for the source of annotations to commit as ground truth. Can be a username, or a source name.

Return type

`None`

snorkelflow.client.annotations.delete_annotation

snorkelflow.client.annotations.delete_annotation(*node, annotation_uids*)

Deletes annotations from a list of annotation UIDs. Deleted annotations cannot be retrieved once they are deleted. Deleting an annotation will not impact the ground truth labels for the document or span, even if that annotation has been committed as ground truth. Use `sf.get_annotations()` to see all annotations for a node and to retrieve the appropriate annotation UIDs.

Examples

```
>>> sf.delete_annotation(1, [123, 456, 789])
# No return value. If the function does not raise an error, the
annotations were deleted successfully.
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
annotation_uids	`List[int]`		A list of Annotation UIDs to be deleted.

Return type

`None`

snorkelflow.client.annotations.get_annotations

snorkelflow.client.annotations.get_annotations(*node, source_name=None, x_uids=None, batch_uids=None*)

Get a collection of annotations for a particular node. Annotations are returned in a Pandas Series. The returned Pandas Series has a MultiIndex with the first level being the x_uid and the second level being the annotation_uid. You can increase the granularity of annotations retrieved by specifying a source name, a list of x_uids, or a list of batch_uids. batch_uids can be accessed by using the `sf.get_batches` function.

Examples

```
>>> annotations = sf.get_annotations(node)
>>> annotations.loc["doc::1"]  # Get annotations for doc::1
>>> annotations.loc[:, 1000]  # Get an annotation of annotation_uid 1000
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node whose annotations we are fetching.
source_name	`Optional[str]`	`None`	The name of the source (annotator, aggregation, etc).
x_uids	`Optional[List[str]]`	`None`	An optional list of x_uids.
batch_uids	`Optional[List[int]]`	`None`	An optional list of batch uids.

Return type

`pd.Series`

snorkelflow.client.annotations.get_interannotator_agreement

snorkelflow.client.annotations.get_interannotator_agreement(*node, label_str=None, batch_uids=None, metric=None*)

Get interannotator agreement statistics for a node. This function returns a list of usernames, an interannotator agreement matrix, and an interannotator agreement metric.

> - The interannotator agreement matrix is a matrix of agreement percentages between each pair of annotators. Agreement is only calculated over data points where both annotators gave a valid label. If the `label_str` argument is provided, then the matrix will only contain agreement percentages for that class.
> - The interannotator agreement metric is a metric that quantifies overall interannotator agreement. For non-span-based tasks, the metric is `krippendorff-alpha`. For span-based tasks, the metric is `mean-span-f1`.

Examples

```
>>> sf.get_interannotator_agreement(1)
{'usernames': ['my-username'], 'matrix': [[1.0]], 'metric': None}
>>> sf.get_interannotator_agreement(NODE_UID)
{
    'usernames': [<annotator_name>, ...],
    'matrix': [[<agreement_%_by_user_per_class>, ... ],  ... ],
    'metric': <score>,
}
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node that we are fetching the interannotator agreement from.

label_str	`Optional[str]`	`None`	Optional class value to compute interannotator agreement per class.
batch_uids	`Optional[List[int]]`	`None`	Optional list of batch_uids to compute interannotator agreement per batch.
metric	`Optional[str]`	`None`	Optional string to include an interannotator agreement metric such as Krippendorff's Alpha.

Return type

`Dict[str, Any]`

snorkelflow.client.annotations.update_annotation

snorkelflow.client.annotations.update_annotation(*node, annotation_uid, label, metadata=None*)

Updates the value of an already-created annotation. This will maintain the original author and timestamp of the annotation. Use `sf.delete_annotation` and *sf.add_annotation*` for hard overwrites, where you want to overwrite all annotation data.

Examples

```
>>> sf.get_annotations(0).loc["doc::0"].loc[123]
{'annotation_uid': 123, 'x_uid': 'doc::0', 'label': 'POS'}
>>> sf.update_annotation(0, 123, "NEG", metadata={"updated by": "sdk"})
>>> sf.get_annotations(0).loc["doc::0"].loc[123]
{'annotation_uid': 123, 'x_uid': 'doc::0', 'label': 'NEG', metadata=
{"updated by": "sdk"}}
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node the annotation is in.
annotation_uid	`int`		The UID for the annotation to update.
label	`str`		The new label for the annotation.
metadata	`Optional[Dict]`	`None`	Optional metadata to update, by default None.

Raises

- **ValueError** – If the label supplied is not valid for the given node
- **ValueError** – If we are unable to successfully fetch the node information

- **ValueError** – If updating the annotation fails

Return type

`None`

snorkelflow.client.applications

Application related functions.

Functions

add_block_to_application(application, ...[, ...])	Add a block to an existing application.
create_app_version(application, name[, ...])	Create a new App version for an application.
create_application(application_name, dataset)	Create a new application
create_classification_application(...[, ...])	Create an application with a classification block.
create_hocr_classification_application(...)	Create a hOCR PDF classification application.
create_hocr_extraction_application(...[, ...])	Create an application with a hOCR extraction block.
create_multilabel_classification_application(...)	Create an application with a multi-label classification block.
create_native_pdf_classification_application(...)	Create a native PDF classification application.
create_native_pdf_extraction_application(...)	Create an application with a native PDF extraction block.
create_new_application_with_resplit_datasources(...)	Resplit datasources and creates a new application with resplit datasources.
create_sequence_tagging_application(...[, ...])	Create an application with a sequence tagging block.

create_text_extraction_application(...[, ...])	Create an application with a text extraction block.
delete_application(application)	Deletes the specified application.
duplicate_application(application, ...[, ...])	Duplicate an existing application.
execute_graph_on_data(application, ...)	Execute specified operators on an input DataFrame.
get_application(application)	Get a specific application based on the name
get_applications([dataset])	Get a list of all applications
list_app_versions(application)	List all App versions for an application.
load_app_version(application, app_version_uid)	Load an existing App version for an application.
set_application_visibility(application, ...)	Update the is_public field of an existing application
update_application(application[, ...])	Update various attributes of an existing Application
visualize_application_graph(application)	BETA: Plots a representation of the application graph, and returns a networkx DiGraph for further analysis.

snorkelflow.client.applications.add_block_to_application

snorkelflow.client.applications.add_block_to_application(*application, template_id, input_node, block_config, from_block_uid=None, label_schema_uid=None*)

Add a block to an existing application.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the new application.
template_id	`str`		The string identifier for the type of block.
input_node	`int`		The UID of the input node.
block_config	`Dict[str, Any]`		A dictionary representation of the configuration for the block, with below key-value pairs defined: • ”extraction_field”: *str* # For information extraction tasks only, in which case str value should be name of column from which to perform extraction • ”hocr_field”: *str* # For hOCR Extraction tasks only, in which case str value should be {name of hOCR data column} • ”label_col”: *str* # For single-label classification tasks only, in which case str should be name of column with GT labels • ”label_map”: *Dict[str, int]*

			• "pdf_field": *str* # For PDF classification tasks and native PDF extraction tasks only, in which case str value should be name of column containing rich doc pdf url's" • "seq_field": *str* # For sequence-tagging tasks only, in which case str value should be name of column on which to perform sequence tagging.
from_block_uid	`Optional[int]`	`None`	An optional identifier for a block to copy from.
label_schema_uid	`Optional[int]`	`None`	An optional identifier for a label schema to use.

Returns

Information on the created block

Return type

`Dict[str, Any]`

Examples

```
>>> sf.add_block_to_application(
>>>     application=APP_NAME,
>>>     template_id="clf",
>>>     input_node=INPUT_NODE_ID,
>>>     block_config=dict(
>>>         label_map=dict(UNKNOWN=-1, NEGATIVE=0, POSITIVE=1),
>>>     ),
>>> )
{'block_uid': <block_uid>, 'node_uids': [node_uid,...], 'job_ids':
[job_id,...]}
```

snorkelflow.client.applications.create_app_version

snorkelflow.client.applications.create_app_version(*application, name, description=<snorkelflow.client_v3.tdm.types.Unset object>*)

Create a new App version for an application.

Parameters

Name	Type	Default	
application	`Union[str, int]`		Th or the ap to GT for
name	`str`		Na the Ap ver
description	`Union[Unset, str]`	`<snorkelflow.client_v3.tdm.types.Unset object at 0x74581c032e90>`	De of Ap ver

Returns

dag_version_uid: The UID of the new App version

Return type

`Dict[str, int]`

snorkelflow.client.applications.create_application

snorkelflow.client.applications.create_application(*application_name, dataset, input_schema=None*)

Create a new application

Parameters

Name	Type	Default	Info
application_name	`str`		The name of the new application.
dataset	`Union[str, int]`		The name or UID of the dataset which the application will use.
input_schema	`Optional[List[str]]`	`None`	A list of columns (from the dataset). If not provided, all the columns but uid_col in the dataset are used.

Returns

Information on the created application

Return type

`Dict[str, Any]`

Examples

```
>>> sf.create_application(
>>>     APP_NAME,
>>>     dataset=DATASET_NAME,
>>>     input_schema=["field1", "field2"],
>>> )
{'application_uid': <application_uid>, 'job_ids': [<job_id>, ...]}
```

snorkelflow.client.applications.create_classification_application

snorkelflow.client.applications.create_classification_application(*application_name, dataset, input_schema, labels, label_col=None*)

Create an application with a classification block.

Parameters

Name	Type	Default	Info
application_name	`str`		The name of the new application.
dataset	`Union[str, int]`		The name or UID of the dataset which the application will use.
input_schema	`List[str]`		A list of columns (from the dataset).
labels	`List[str]`		A string list of labels for the task.
label_col	`Optional[str]`	`None`	An optional string name of a dataset column that contains labels.

Returns

The created application object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.create_classification_application(
>>>     APP_NAME,
>>>     dataset=DATASET_NAME,
>>>     input_schema=["field1", "field2"],
>>>     labels=["label1", "label2"],
>>>     label_col="label",
>>> )
{
    'application_uid': <application_uid>,
    'name': <name>,
    'dataset_uid': <dataset_uid>,
    'description': <description>,
    'application_config': <application_config_dict>, # Contains input
schema
    'created_at': <timetamp>,
    'node_dag': <node_dag>,  # contains list of nodes and it's input
node
    'access_config': <access_config>,
    'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
}
```

snorkelflow.client.applications.create_hocr_classification_application

snorkelflow.client.applications.create_hocr_classification_application(*application_name, dataset, input_schema, hocr_field, labels, label_col=None, split_docs_by_page=False*)

Create a hOCR PDF classification application.

Parameters

Name	Type	Default	Info
application_name	`str`		The name of the new application.
dataset	`Union[str, int]`		The name or UID of the dataset which the application will use.
input_schema	`List[str]`		A list of columns (from the dataset).
hocr_field	`str`		The field that contains hocr data.
labels	`List[str]`		A string list of labels for the application.
label_col	`Optional[str]`	`None`	An optional string name of a dataset column that contains labels.
split_docs_by_page	`bool`	`False`	An optional value to add PageSplitter node to the application DAG.

Returns

The created application object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.create_hocr_classification_application(
>>>     APP_NAME,
>>>     dataset=DATASET_NAME,
>>>     input_schema=["uid", "file_name", "hocr", "rich_doc_pdf_url"],
>>>     hocr_field="hocr",
>>>     labels=["label1", "label2"],
>>>     label_col="label",
>>> )
{
    'application_uid': <application_uid>,
    'name': <name>,
    'dataset_uid': <dataset_uid>,
    'description': <description>,
    'application_config': <application_config_dict>, # Contains input
schema
    'created_at': <timetamp>,
    'node_dag': <node_dag>,  # contains list of nodes and it's input
node
    'access_config': <access_config>,
    'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
}
```

snorkelflow.client.applications.create_hocr_extraction_application

snorkelflow.client.applications.create_hocr_extraction_application(*application_name, dataset, input_schema, hocr_field, labels, label_col=None, split_docs_by_page=False, pdf_url_field='rich_doc_pdf_url'*)

Create an application with a hOCR extraction block.

Parameters

Name	Type	Default	Info
application_name	`str`		The name of the new application.
dataset	`Union[str, int]`		The name or UID of the dataset which the application will use.
input_schema	`List[str]`		A list of columns (from the dataset).
hocr_field	`str`		The field that contains hocr data.
labels	`List[str]`		A string list of labels for the application.
label_col	`Optional[str]`	`None`	An optional string name of a dataset column that contains labels.

split_docs_by_page	`bool`	`False`	An optional value to add PageSplitter node to the application DAG.
pdf_url_field	`str`	`'rich_doc_pdf_url'`	An string name of a dataset column that contains PDF URLs.

Returns

The created application object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.create_hocr_extraction_application(
>>>     APP_NAME,
>>>     dataset=DATASET_NAME,
>>>     input_schema=["uid", "file_name", "hocr", "rich_doc_pdf_url"],
>>>     hocr_field="hocr",
>>>     labels=["label1", "label2"],
>>>     label_col="label",
>>> )
{
    'application_uid': <application_uid>,
    'name': <name>,
    'dataset_uid': <dataset_uid>,
    'description': <description>,
    'application_config': <application_config_dict>, # Contains input
schema
    'created_at': <timetamp>,
    'node_dag': <node_dag>,  # contains list of nodes and it's input
node
    'access_config': <access_config>,
    'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
}
```

snorkelflow.client.applications.create_multilabel_classification_application

snorkelflow.client.applications.create_multilabel_classification_application(*application_name, dataset, input_schema, labels, label_col=None*)

Create an application with a multi-label classification block.

Parameters

Name	Type	Default	Info
application_name	`str`		The name of the new application.
dataset	`Union[str, int]`		The name or UID of the dataset which the application will use.
input_schema	`List[str]`		A list of columns (from the dataset).
labels	`List[str]`		A string list of labels for the task.
label_col	`Optional[str]`	`None`	An optional string name of a dataset column that contains labels.

Returns

The created application object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.create_multilabel_classification_application(
>>>     APP_NAME,
>>>     dataset=DATASET_NAME,
>>>     input_schema=["text"],
>>>     labels=["label1", "label2"],
>>>     label_col="label",
>>> )
{
    'application_uid': <application_uid>,
    'name': <name>,
    'dataset_uid': <dataset_uid>,
    'description': <description>,
    'application_config': <application_config_dict>, # Contains input
schema
    'created_at': <timetamp>,
    'node_dag': <node_dag>,  # contains list of nodes and it's input
node
    'access_config': <access_config>,
    'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
}
```

snorkelflow.client.applications.create_native_pdf_classification_application

snorkelflow.client.applications.create_native_pdf_classification_application(*application_name, dataset, input_schema, pdf_field, labels, label_col=None, split_docs_by_page=False*)

Create a native PDF classification application.

Parameters

Name	Type	Default	Info
application_name	`str`		The name of the new application.
dataset	`Union[str, int]`		The name or UID of the dataset which the application will use.
input_schema	`List[str]`		A list of columns (from the dataset).
pdf_field	`str`		The field that contains the PDF url.
labels	`List[str]`		A string list of labels for the application.
label_col	`Optional[str]`	`None`	An optional string name of a dataset column that contains labels.
split_docs_by_page	`bool`	`False`	An optional value to add PageSplitter node to the application DAG.

Returns

The created application object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.create_native_pdf_classification_application(
>>>     APP_NAME,
>>>     dataset=DATASET_NAME,
>>>     input_schema = ["pdf_url"],
>>>     pdf_field="pdf_url",
>>>     labels=["label1", "label2"],
>>>     label_col="label",
>>> )
{
    'application_uid': <application_uid>,
    'name': <name>,
    'dataset_uid': <dataset_uid>,
    'description': <description>,
    'application_config': <application_config_dict>, # Contains input
schema
    'created_at': <timetamp>,
    'node_dag': <node_dag>,  # contains list of nodes and it's input
node
    'access_config': <access_config>,
    'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
}
```

snorkelflow.client.applications.create_native_pdf_extraction_application

snorkelflow.client.applications.create_native_pdf_extraction_application(*application_name, dataset, input_schema, pdf_field, labels, label_col=None, split_docs_by_page=False*)

Create an application with a native PDF extraction block.

Parameters

Name	Type	Default	Info
application_name	`str`		The name of the new application.
dataset	`Union[str, int]`		The name or UID of the dataset which the application will use.
input_schema	`List[str]`		A list of columns (from the dataset).
pdf_field	`str`		The field that contains the PDF url.
labels	`List[str]`		A string list of labels for the application.
label_col	`Optional[str]`	`None`	An optional string name of a dataset column that contains labels.
split_docs_by_page	`bool`	`False`	An optional value to add PageSplitter node to the application DAG.

Returns

The created application object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.create_native_pdf_extraction_application(
>>>     APP_NAME,
>>>     dataset=DATASET_NAME,
>>>     input_schema = ["pdf_url"],
>>>     pdf_field="pdf_url",
>>>     labels=["label1", "label2"],
>>>     label_col="label",
>>> )
{
    'application_uid': <application_uid>,
    'name': <name>,
    'dataset_uid': <dataset_uid>,
    'description': <description>,
    'application_config': <application_config_dict>, # Contains input
schema
    'created_at': <timetamp>,
    'node_dag': <node_dag>,  # contains list of nodes and it's input
node
    'access_config': <access_config>,
    'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
}
```

snorkelflow.client.applications.create_new_application_with_resplit_datasources

snorkelflow.client.applications.create_new_application_with_resplit_datasources(*application, datasource_uids, split_random_seed=None, split_pct=None*)

Resplit datasources and creates a new application with resplit datasources.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application to resplit data on.
datasource_uids	`List[int]`		UIDs of the datasources to resplit data on.
split_random_seed	`Optional[int]`	`None`	Random seed value for the splitting function.
split_pct	`Optional[Dict[str, float]]`	`None`	Dict containing train/test/valid split ratios.

Return type

`List of new application name, and new dataset name`

snorkelflow.client.applications.create_sequence_tagging_application

snorkelflow.client.applications.create_sequence_tagging_application(*application_name, dataset, input_schema, labels, seq_field, label_col=None*)

Create an application with a sequence tagging block.

Parameters

Name	Type	Default	Info
application_name	`str`		The name of the new application.
dataset	`Union[str, int]`		The name or UID of the dataset which the application will use.
input_schema	`List[str]`		A list of columns (from the dataset).
labels	`List[str]`		A string list of labels for the task.
seq_field	`str`		The name of the field containing sequence data to be used for sequence tagging.
label_col	`Optional[str]`	`None`	An optional string name of a dataset column that contains labels.

Returns

The created application object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.create_sequence_tagging_application(
>>>     APP_NAME,
>>>     dataset=DATASET_NAME,
>>>     input_schema=["text"],
>>>     seq_field="text",
>>>     labels=["label1", "label2"],
>>>     label_col="label",
>>> )
{
    'application_uid': <application_uid>,
    'name': <name>,
    'dataset_uid': <dataset_uid>,
    'description': <description>,
    'application_config': <application_config_dict>, # Contains input
schema
    'created_at': <timetamp>,
    'node_dag': <node_dag>,  # contains list of nodes and it's input
node
    'access_config': <access_config>,
    'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
}
```

snorkelflow.client.applications.create_text_extraction_application

snorkelflow.client.applications.create_text_extraction_application(*application_name, dataset, input_schema, extraction_field, labels, label_col=None*)

Create an application with a text extraction block.

Parameters

Name	Type	Default	Info
application_name	`str`		The name of the new application.
dataset	`Union[str, int]`		The name or UID of the dataset which the application will use.
input_schema	`List[str]`		A list of columns (from the dataset).
extraction_field	`str`		The field to extract from.
labels	`List[str]`		A string list of labels for the task.
label_col	`Optional[str]`	`None`	An optional string name of a dataset column that contains labels.

Returns

The created pplication object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.create_text_extraction_application(
>>>     APP_NAME,
>>>     dataset=DATASET_NAME,
>>>     input_schema=["text"],
>>>     extraction_field="text",
>>>     labels=["label1", "label2"],
>>>     label_col="label",
>>> )
{
    'application_uid': <application_uid>,
    'name': <name>,
    'dataset_uid': <dataset_uid>,
    'description': <description>,
    'application_config': <application_config_dict>, # Contains input
schema
    'created_at': <timetamp>,
    'node_dag': <node_dag>,  # contains list of nodes and it's input
node
    'access_config': <access_config>,
    'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
}
```

snorkelflow.client.applications.delete_application

snorkelflow.client.applications.delete_application(*application*)

Deletes the specified application.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application to delete.

Returns

Application deletion status

Return type

`Dict[str, Any]`

Examples

```
>>> sf.delete_application(APP_NAME)
{
    'uid':<job_uid>,
    'job_type': 'delete-application',
    'state': 'completed',
    'enqueued_time': <timestamp>,
    'execution_start_time': <timestamp>,
    'end_time': <timestamp>,
    'application_uid': <application_uid>,
    'dataset_uid': None,
    'node_uid': None,
    'user_uid': <user_uid>,
    'workspace_uid': <workspace_uid>,
    'percent': 100,
    'message': f'Deleted Application {application_uid}',
    'detail': {'type': 'engine'},
    'pod_name': 'engine',
    'function_name': 'identity_engine_job',
    'process_id': <process_id>,
    'timing': {}
}
```

snorkelflow.client.applications.duplicate_application

snorkelflow.client.applications.duplicate_application(*application, new_application_name, new_dataset_name=None, datasource_uids_to_load=None*)

Duplicate an existing application.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application to duplicate.
new_application_name	`str`		The name of the new application.
new_dataset_name	`Optional[str]`	`None`	Dataset for the new application. Defaults to original application's dataset.
datasource_uids_to_load	`Optional[List[int]]`	`None`	Optional list of datasources to activate in new application if its dataset has changed and we still want to transfer assets (LFs, GT, Tags).

Returns

Information on the duplicated application

Return type

`Dict[str, Any]`

Examples

```
>>> sf.duplicate_application(APP_NAME, DUPLICATED_APP_NAME)
{
    'application_uid': <duplicated_application_uid>,
    'job_ids': [<engine_job_id>,...]
}
```

snorkelflow.client.applications.execute_graph_on_data

snorkelflow.client.applications.execute_graph_on_data(*application, target_uids, df*)

Execute specified operators on an input DataFrame. Please refer to the `df` parameter for a more extensive functionality.

There is another method to retrieve the output data at a specific node using the current dataset/datasource: `sf.get_node_output_data`.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the new application.
target_uids	`List[int]`		A list of operators to process the DataFrame over.
df	`Union[DataFrame, Dict[int, DataFrame]]`		Either a Pandas DataFrame, or a dictionary mapping node uids to Pandas DataFrames. The dictionary specifies the input dataframe assigned to the corresponding node. If a single DataFrame df is provided it is converted to {-1: df} i.e. df is used at the input to the application DAG.

Returns

A mapping from target operator uid to processed Pandas DataFrame

Return type

`Dict[str, any]`

Examples

```
>>> df = sf.get_dataset_data(APP_NAME)
>>> sf.execute_graph_on_data(APP_NAME, target_uids=[123, 456], df=df)
{
    <op_node_uid>: pd.Dataframe,
    ...
}
```

snorkelflow.client.applications.get_application

snorkelflow.client.applications.get_application(*application*)

Get a specific application based on the name

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.

Returns

The requested application object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.get_application(APP_NAME)
{
    'application_uid': <application_uid>,
    'name': <name>,
    'dataset_uid': <dataset_uid>,
    'description': <description>,
    'application_config': <application_config_dict>, # Contains input
schema
    'created_at': <timetamp>,
    'node_dag': <node_dag>,  # contains list of nodes and it's input
node
    'access_config': <access_config>,
    'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
}
```

snorkelflow.client.applications.get_applications

snorkelflow.client.applications.get_applications(*dataset=None*)

Get a list of all applications

Parameters

Name	Type	Default	Info
dataset	`Union[str, int, None]`	`None`	The name or UID of the dataset to filter applications.

Returns

A list of all application objects

Return type

`List[Dict[str, Any]]`

Examples

```
>>> sf.get_applications()
[
    {
        'application_uid': <application_uid>,
        'name': <name>,
        'dataset_uid': <dataset_uid>,
        'description': <description>,
        'application_config': <application_config_dict>, # Contains
input schema
        'created_at': <timetamp>,
        'node_dag': <node_dag>,  # Contains list of nodes and it's input
node
        'access_config': <application_uid>,
        'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
    },
    ...
]
```

snorkelflow.client.applications.list_app_versions

snorkelflow.client.applications.list_app_versions(*application*)

List all App versions for an application.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application to list App versions for.

Returns

app_version_uid: The UID of the App version application_uid: The UID of the application which the App version is associated with name: name of App version description: description of App version dag_version_info: contains version information for the DAG including DAG structure and committed op versions hidden: Deleted state of App version created_by: Author name created_at: App version creation time

Return type

`Dict[str, int]`

snorkelflow.client.applications.load_app_version

snorkelflow.client.applications.load_app_version(*application*, *app_version_uid*)

Load an existing App version for an application.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application to load the App version for.
app_version_uid	`int`		The UID of the App version to load into the application.

Return type

`None`

snorkelflow.client.applications.set_application_visibility

snorkelflow.client.applications.set_application_visibility(*application, is_public*)

Update the is_public field of an existing application

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the existing application.
is_public	`bool`		Boolean flag determining if the application is public or private.

Returns

Information on the created application

Return type

`Dict[str, Any]`

Examples

```
>>> sf.set_application_visibility(APP_NAME, is_public=True)
{
    'access_config': <access_config>, # Function will update this field
    'application_uid': <application_uid>,
    'name': <name>,
    'dataset_uid': <dataset_uid>,
    'description': <description>,
    'application_config': <application_config_dict>, # Contains input
schema
    'created_at': <timetamp>,
    'node_dag': <node_dag>,  # Contains list of nodes and it's input
node
    'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
}
```

snorkelflow.client.applications.update_application

snorkelflow.client.applications.update_application(*application, new_application_name=None, access_config=None, input_schema=None*)

Update various attributes of an existing Application

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the existing application.
new_application_name	`Optional[str]`	`None`	A new name for the existing application.
access_config	`Optional[Dict[str, Any]]`	`None`	Dictionary containing one field *is_public*, to set the public/private state.
input_schema	`Optional[List[str]]`	`None`	A list of columns (from the dataset).

Returns

Information on the created application

Return type

`Dict[str, Any]`

Examples

```
>>> sf.update_application(APP_NAME, input_schema=["new_field1"])
{
    'access_config': <access_config>,
    'application_uid': <application_uid>,
    'name': <name>,
    'dataset_uid': <dataset_uid>,
    'description': <description>,
    'application_config': <application_config_dict>, # Contains new
input schema
    'created_at': <timetamp>,
    'node_dag': <node_dag>,  # Contains list of nodes and it's input
node
    'hydrated_node_dag': <hydrated_node_dag> # Contains list of it's
node and it's config
}
```

snorkelflow.client.applications.visualize_application_graph

snorkelflow.client.applications.visualize_application_graph(*application*)

BETA: Plots a representation of the application graph, and returns a networkx DiGraph for further analysis. Note that this function is in beta, and has known rendering issues for some graph layouts. The function returns an nx.DiGraph, so you can try different graph visualizations using references from:
https://networkx.org/documentation/stable/reference/drawing.html

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the new application.

Return type

`nx.DiGraph`

snorkelflow.client.batches

(Annotation) batch related functions.

Functions

create_batches(node, username, assignees[, ...])	Create batches of data points for annotation.
delete_batch(node, batch_name)	Delete a batch
get_batches(node)	Get batches.
get_x_uids_from_batch(batch_uid)	Get a list of x_uids for a batch.
update_batch(node, batch_name[, ...])	Update an existing batch

snorkelflow.client.batches.create_batches

snorkelflow.client.batches.create_batches(*node, username, assignees, batch_name=None, split=None, num_batches=None, batch_size=None, sample_by_docs=False, parent_batch_uid=None, datasource_uid=None, randomize=False, random_seed=None, x_uids=None*)

Create batches of data points for annotation.

Parameters

Name	Type	Default	Info
node	`int`		The id of the node to create the batches from.
username	`str`		The username of the author of the batch(es) being created.
assignees	`List[str]`		The usernames of annotators assigned to annotate this batch.
batch_name	`Optional[str]`	`None`	The name of the batch. If multiple batches are created, each will be appended with a number. Default to “Batch #”.
split	`Optional[str]`	`None`	One of three sources to create batch: currently support train, test, & valid splits.
num_batches	`Optional[int]`	`None`	The number of batches. Default to 1.
batch_size	`Optional[int]`	`None`	Default to the total size divided by the number of batches (with any extras in

			the final batch to reach the total number of data points).
sample_by_docs	`bool`	`False`	If True, then inputs (num_batches, batch_size) are interpreted as number of documents (rather than number of spans). This parameter results in different results only for candidate-based extraction applications where datapoints correspond to spans not documents.
parent_batch_uid	`Optional[int]`	`None`	If the new batch(es) are being created from an existing batch, that batch's UID.
datasource_uid	`Optional[int]`	`None`	If the new batch(es) are being created from a datasource, that datasource's UID.
randomize	`Optional[bool]`	`False`	Whether the data should be shuffled before dividing into batches.
random_seed	`Optional[int]`	`None`	The random seed for batch shuffling.
x_uids	`Optional[List[str]]`	`None`	The list of x_uids to be assigned to this batch.

Returns

A list of created batches.

Return type

List[Batch]

Examples

```
# Creating batches with default parameters.
sf.create_batches(node, username="assigner", assignees=["user 1", "user
2"])

# Creating a batch with specific datapoints using the index. Datapoints
must be from the same split.
sf.create_batches(node, username="assigner", assignees=["user 1"],
x_uids=["span::5", "span::7"])

# Creating batches with a fixed number of datapoints.
sf.create_batches(node, username="assigner", assignees=["user 1", "user
2"], batch_size=10)

# Creating batches with a fixed number of documents.
# In candidate-based extraction applications the datapoints are spans,
but users would like to create batches of documents.
sf.create_batches(node, username="assigner", assignees=["user 1", "user
2"], batch_size=5, sample_by_docs=True)
```

snorkelflow.client.batches.delete_batch

snorkelflow.client.batches.delete_batch(*node, batch_name*)

Delete a batch

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node.
batch_name	`str`		The name of the batch.

Returns

Return true if the operation succeeds.

Return type

`bool`

snorkelflow.client.batches.get_batches

snorkelflow.client.batches.get_batches(*node*)

Get batches.

Parameters

Name	Type	Default	Info
node	`int`		The id of the node to get the batches from.

Returns

A list of batches; return empty list if there is no batch.

Return type

List[Batch]

snorkelflow.client.batches.get_x_uids_from_batch

snorkelflow.client.batches.get_x_uids_from_batch(*batch_uid*)

Get a list of x_uids for a batch.

Parameters

Name	Type	Default	Info
batch_uid	`int`		The id of the batch.

Returns

A list of x_uids for this batch

Return type

`List[str]`

snorkelflow.client.batches.update_batch

snorkelflow.client.batches.update_batch(*node, batch_name, new_batch_name=None, assignees=None*)

Update an existing batch

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node.
batch_name	`str`		The name of the current batch.
new_batch_name	`Optional[str]`	`None`	What the batch's new name should be updated to, if provided.
assignees	`Optional[List[str]]`	`None`	The list of usernames that should be assigned to this batch.

Returns

The updated batch.

Return type

Batch

snorkelflow.client.blocks

Block related functions.

Functions

delete_operator_block(application, block_uid)	Delete a specified block from an application
duplicate_block(application, from_block_uid, ...)	Duplicates a block in an application
get_operator_block(application, block_uid)	Fetch details about a specified block
get_operator_blocks(application)	Fetch all blocks in an application

snorkelflow.client.blocks.delete_operator_block

snorkelflow.client.blocks.delete_operator_block(*application, block_uid*)

Delete a specified block from an application

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.
block_uid	`int`		The unique identifier for a block within the application.

Return type

`None`

snorkelflow.client.blocks.duplicate_block

snorkelflow.client.blocks.duplicate_block(*application, from_block_uid, block_config, input_node_uid=-1, assets_to_import=None*)

Duplicates a block in an application

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.
from_block_uid	`int`		The uid of the block to duplicate.
block_config	`Dict[str, Any]`		The block config for the block to be created.
input_node_uid	`int`	`-1`	The input node uid for the newly created block- defaults to the input.
assets_to_import	`Optional[Dict[str, Any]]`	`None`	An optional dictionary specifying duplication options.

Returns

Details on the duplicated block

Return type

`Dict[str, any]`

Example

```
>>> sf.duplicate_block(
>>>    APP_NAME,
>>>    BLOCK_UID,
>>>    block_config=dict(
>>>        label_map=dict(POS=1, NEG=0, UNKNOWN=-1)
>>>    ),
>>> )
{
    'block_uid': <block_uid>,
    'node_uids': [<node_uid>, ...],
    'job_ids': [<job_uid>, ...]
}
```

snorkelflow.client.blocks.get_operator_block

snorkelflow.client.blocks.get_operator_block(*application, block_uid*)

Fetch details about a specified block

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.
block_uid	`int`		The unique identifier for a block within the application.

Returns

The requested block object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.get_operator_block(APP_NAME, 123)
{
    'block_uid': <block_uid>,
    'application_uid': <application_uid>,
    'template_id':  <template_id>, # Corresponds to application/block
template id
    'node_uids': [<node_uid>, ...],
    'block_metadata': <block_metadata_dict>,
    'name': <block_name>,
}
```

snorkelflow.client.blocks.get_operator_blocks

snorkelflow.client.blocks.get_operator_blocks(*application*)

Fetch all blocks in an application

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.

Returns

A list of all block objects in the specified application

Return type

`List[Dict[str, any]]`

Examples

```
>>> sf.get_operator_blocks(APP_NAME)
[
    {
        'block_uid': <block_uid>,
        'application_uid': <application_uid>,
        'template_id':  <template_id>, # Corresponds to
application/block template id
        'node_uids': [<node_uid>, ...],
        'block_metadata': <block_metadata_dict>,
        'name': <block_name>,
    }
]
```

snorkelflow.client.comments

Comment related functions. Comments let you add custom notes to data points, for tracking and collaboration. Comments are visible to all users who have access to the task.

Functions

`create_comment`(node, x_uid, username, body)	Create a new comment for a data point.
`delete_comment`(node, comment_uid)	Remove an existing comment by its comment UID.
`delete_datapoint_comments`(node, x_uid)	Removes all existing comments for a particular datapoint.
`edit_comment`(node, comment_uid, body)	Edit the body text of an existing comment.
`get_comment`(node, comment_uid)	Fetch an existing comment by its UID.
`get_comments`(node[, username, x_uid])	Return a list of all comments for the task.

snorkelflow.client.comments.create_comment

snorkelflow.client.comments.create_comment(*node, x_uid, username, body, is_context_tag=False*)

Create a new comment for a data point.

Examples

```
>>> sf.create_comment(node=1, x_uid="doc::10005", username="user1",
body="This is a comment")
{
    'comment_uid': 7,
    'user_uid': 3,
    'x_uid': 'doc::10005',
    'body': 'This is a comment',
    'created_at': '2023-09-26T17:31:15.759565',
    'is_edited': False
}
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the parent model node for this comment.
x_uid	`str`		The UID for the datapoint. Datapoint UIDs can be found by looking at the index `__DATAPOINT_UID` column of the node's DataFrame.
username	`str`		The username of the author of the comment. A username must be provided.
body	`str`		The body text of the comment.
is_context_tag	`bool`	`False`	For information extraction tasks, adds a comment to the parent document instead of the provided span UID. By default False.

Returns

A dictionary of metadata corresponding to the newly created comment.

Return type

`Dict[str, Union[str, int]]`

snorkelflow.client.comments.delete_comment

snorkelflow.client.comments.delete_comment(*node, comment_uid*)

Remove an existing comment by its comment UID.

To delete all comments for a given datapoint, see delete_datapoint_comments()

Examples

```
>>> sf.delete_comment(node=1, comment_uid=7)
None
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the parent model node for this comment.
comment_uid	`int`		UID of the comment to delete.

Return type

`None`

snorkelflow.client.comments.delete_datapoint_comments

snorkelflow.client.comments.delete_datapoint_comments(*node, x_uid*)

Removes all existing comments for a particular datapoint.

To delete an individual comment by comment_uid, see delete_comment()

Examples

```
>>> sf.delete_datapoint_comments(node=1, x_uid="doc::10005")
None
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the parent model node for this comment.
x_uid	`str`		The UID of the datapoint whose comments we wish to delete.

Return type

`None`

snorkelflow.client.comments.edit_comment

snorkelflow.client.comments.edit_comment(*node, comment_uid, body*)

Edit the body text of an existing comment. This will overwrite the existing comment body. This will not change the comment's UID, creation date, or the associated username.

Examples

```
>>> sf.edit_comment(node=1, comment_uid=7, body="This is an edited
comment")
{
    'comment_uid': 7,
    'user_uid': 3,
    'x_uid': 'doc::10005',
    'body': 'This is an edited comment',
    'created_at': '2023-09-26T17:31:15.759565',
    'is_edited': True
}
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the parent model node for this comment.
comment_uid	`int`		UID of the comment to edit.
body	`str`		New body of the comment.

Returns

A dictionary of metadata corresponding to the edited comment.

Return type

`Dict[str, Union[str, int]]`

snorkelflow.client.comments.get_comment

snorkelflow.client.comments.get_comment(*node, comment_uid*)

Fetch an existing comment by its UID.

Examples

```
>>> sf.get_comment(node=1, comment_uid=7)
{
    'comment_uid': 7,
    'user_uid': 3,
    'x_uid': 'doc::10005',
    'body': 'This is a comment',
    'created_at': '2023-09-26T17:31:15.759565',
    'is_edited': False
}
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the parent model node for this comment.
comment_uid	`int`		UID of the comment to retrieve. Use `snorkelflow.client.get_comments()` to see a list of all comments for a node.

Returns

A dictionary of metadata corresponding to the comment.

Return type

`Dict[str, Union[str, int]]`

snorkelflow.client.comments.get_comments

snorkelflow.client.comments.get_comments(*node, username=None, x_uid=None*)

Return a list of all comments for the task. This method can optionally be filtered by the username of the comment author or by the UID of the associated datapoint.

Examples

```
>>> sf.get_comments(node=1, username="user1")
[
    {
        'comment_uid': 7,
        'user_uid': 3,
        'x_uid': 'doc::10005',
        'body': 'This is a comment',
        'created_at': '2023-09-26T17:31:15.759565',
        'is_edited': False
    },
    {
        'comment_uid': 8,
        'user_uid': 3,
        'x_uid': 'doc::10005',
        'body': 'This is another comment',
        'created_at': '2023-09-26T17:31:15.759565',
        'is_edited': False
    }
]
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the parent model node for this comment.
username	`Optional[str]`	`None`	If included, only return comments by the user with this username.
x_uid	`Optional[str]`	`None`	UID of datapoint. If included, returns all comments associated with only this datapoint.

Returns

A list of dictionaries of metadata for all comments that match the provided filters.

Return type

`List[Dict[str, Union[str, int]]]`

snorkelflow.client.ctx

Context related classes.

Classes

SnorkelFlowContext(tdm_client[, minio_args, ...])	The SnorkelFlowContext object provides client context for the Snorkel Flow SDK.
SnorkelFlowPredictionAPIClient	alias of **HTTPClient**

snorkelflow.client.ctx.SnorkelFlowContext

class snorkelflow.client.ctx.SnorkelFlowContext(*tdm_client*, *minio_args=None*, *studio_client=None*, *storage_client=None*, *workspace_name=None*, *set_global=True*, *debug=False*)

Bases: `object`

The SnorkelFlowContext object provides client context for the Snorkel Flow SDK. It allows the Snorkel Flow SDK to recognize a Snorkel Flow instance by identifying Snorkel Flow's essential API services (TDM, Studio, MinIO) via user-provided parameters, a YAML config file, or a Snorkel Flow API key.

Examples

```
import snorkelflow.client as sf
ctx = sf.SnorkelFlowContext.from_kwargs(...)
```

The keyword arguments are presented in the documentation below. Alternatively, if you have a `.snorkel-flow.yaml` config file locally, you can use it to create a context:

```
import snorkelflow.client as sf
ctx = sf.SnorkelFlowContext.from_user_config("path/to/.snorkel-
flow.yaml")
```

__init__

__init__(*tdm_client*, *minio_args=None*, *studio_client=None*, *storage_client=None*, *workspace_name=None*, *set_global=True*, *debug=False*)

Initialize a SnorkelFlowContext.

Methods

`__init__`(tdm_client[, minio_args, ...])	Initialize a SnorkelFlowContext.
`from_endpoint_url`([endpoint, ...])	Initialize a SnorkelFlowContext from keyword arguments.
`from_kwargs`([tdm_scheme, tdm_host, ...])	Initialize a SnorkelFlowContext from keyword arguments.

from_user_config(cls[, user_config_yaml, ...])	Method to create a SnorkelFlowContest object from a `.snorkel-flow.yaml` file.
get_global()	Retrieve the global SnorkelFlowContext object.
get_minio_client()	Retrieve an S3 resource for the Minio client.
set_debug(debug)	Set the verbosity of warnings, details, and stacktraces
set_global([ctx])	Set a SnorkelFlowContext object globally.

Attributes

storage_client	
workspace_name	SnorkelFlowContext objects are only scoped to work in a particular workspace.

from_endpoint_url

classmethod from_endpoint_url(*endpoint=None, minio_endpoint=None, api_key=None, minio_access_key=None, minio_secret_key=None, workspace_name=None, set_global=True, debug=False*)

Initialize a SnorkelFlowContext from keyword arguments.

Examples

```
# Instantiate with default kwargs and a custom url
import snorkelflow.client as sf
ctx =
sf.SnorkelFlowContext.from_endpoint_url("https://edge.k8s.g498.io/")
```

```
# Instantiate with custom kwargs
import snorkelflow.client as sf
ctx = sf.SnorkelFlowContext.from_endpoint_url(
    endpoint="https://edge.k8s.g498.io/",
    minio_endpoint="https://edge-minio-api.k8s.g498.io",
    minio_access_key="minio",
    minio_secret_key="minio123",
    workspace_name="my-workspace",
)
```

```
# Instantiate with an API key
import snorkelflow.client as sf
ctx = sf.SnorkelFlowContext.from_endpoint_url(
    endpoint="https://edge.k8s.g498.io/",
    api_key="my-api-key",
)
```

Parameters

Name	Type	Default	Info
endpoint	`Optional[str]`	`None`	The baseurl to a snorkelflow instance.
minio_endpoint	`Optional[str]`	`None`	The url to a minio service.
minio_access_key	`Optional[str]`	`None`	The access key to use for MinIO. If not provided, look for the access key at an environment variable *MINIO_ACCESS_KEY*. If this environment variable is not defined, then look for a file identified by an environment variable *MINIO_ACCESS_KEY_FILE*.
minio_secret_key	`Optional[str]`	`None`	The secret key to use for MinIO. If not provided, look for the secret key at an

			environment variable *MINIO_SECRET_KEY*. If this environment variable is not defined, then look for a file identified by an environment variable *MINIO_SECRET_KEY_FILE*.
workspace_name	`Optional[str]`	`None`	The workspace name, which determines the workspace a dataset and application will be created in and queried from.
api_key	`Optional[str]`	`None`	The API key to use for the TDM, Studio, and Storage APIs.
set_global	`bool`	`True`	Whether to set the context as the global context, accessible across all notebooks in-platform.
debug	`bool`	`False`	Whether to enable debug mode. Debug mode will print more verbose errors to the screen inside of the Python notebook.

Returns

A SnorkelFlowContext object that can be used to interact with the MinIO, TDM, and Studio clients directly.

Return type

SnorkelFlowContext

from_kwargs

classmethod **from_kwargs**(*tdm_scheme='http', tdm_host='tdm-api', tdm_port=8686, tdm_url_prefix='', minio_scheme='http', minio_host='minio', minio_port=9001, minio_access_key=None, minio_secret_key=None, studio_scheme='http', studio_host='studio-api', studio_port=8484, studio_url_prefix='', storage_scheme='http', storage_host='storage-api', storage_port=31315, storage_url_prefix='', workspace_name=None, api_key=None, set_global=True, debug=False*)

Initialize a SnorkelFlowContext from keyword arguments.

Examples

```
# Instantiate with default kwargs
import snorkelflow.client as sf
ctx = sf.SnorkelFlowContext.from_kwargs()
```

```
# Instantiate with custom kwargs
import snorkelflow.client as sf
ctx = sf.SnorkelFlowContext.from_kwargs(
    tdm_port=8080,
    minio_port=9000,
    minio_access_key="minio",
    minio_secret_key="minio123",
    studio_port=8080,
    workspace_name="my-workspace",
)
```

```
# Instantiate with an API key
import snorkelflow.client as sf
ctx = sf.SnorkelFlowContext.from_kwargs(
    api_key="my-api-key",
)
```

Parameters

Name	Type	Default	Info
tdm_scheme	`str`	`'http'`	The scheme to use for the TDM API.

tdm_host	`str`	`'tdm-api'`	The host to use for the TDM API.
tdm_port	`int`	`8686`	The port to use for the TDM API. Contact an administrator for information.
tdm_url_prefix	`str`	`''`	The url prefix to use for the TDM API.
minio_scheme	`str`	`'http'`	The scheme to use for MinIO.
minio_host	`str`	`'minio'`	The host to use for MinIO.
minio_port	`int`	`9001`	The port to use for MinIO. Contact an administrator for information.
minio_access_key	`Optional[str]`	`None`	The access key to use for MinIO. If not provided, look for the access key at an environment variable *MINIO_ACCESS_KEY*. If this environment variable is not defined, then look for a file identified by an environment variable *MINIO_ACCESS_KEY_FILE*.
minio_secret_key	`Optional[str]`	`None`	The secret key to use for MinIO. If not provided, look for the secret key at an environment variable

			MINIO_SECRET_KEY. If this environment variable is not defined, then look for a file identified by an environment variable *MINIO_SECRET_KEY_FILE*.
studio_scheme	`str`	`'http'`	The scheme to use for the Studio API.
studio_host	`str`	`'studio-api'`	The host to use for the Studio API.
studio_port	`int`	`8484`	The port to use for the Studio API. Contact an administrator for information.
studio_url_prefix	`str`	`''`	The url prefix to use for the Studio API.
storage_scheme	`str`	`'http'`	The scheme to use for the Storage API.
storage_host	`str`	`'storage-api'`	The host to use for the Storage API.
storage_port	`int`	`31315`	The port to use for the Storage API. Contact an administrator for information.
storage_url_prefix	`str`	`''`	The url prefix to use for the Storage API.
workspace_name	`Optional[str]`	`None`	The workspace name, which determines the workspace a dataset and

			application will be created in and queried from.
api_key	`Optional[str]`	`None`	The API key to use for the TDM and Studio APIs.
set_global	`bool`	`True`	Whether to set the context as the global context, accessible across all notebooks in-platform.
debug	`bool`	`False`	Whether to enable debug mode. Debug mode will print more verbose errors to the screen inside of the Python notebook.

Returns

A SnorkelFlowContext object that can be used to interact with the MinIO, TDM, and Studio clients directly.

Return type

SnorkelFlowContext

from_user_config

classmethod **from_user_config**(*cls, user_config_yaml=None, protocol='http', set_global=True, debug=False*)

Method to create a SnorkelFlowContest object from a `.snorkel-flow.yaml` file.

Examples

```
import snorkelflow.client as sf
ctx =
sf.SnorkelFlowContext.from_user_config("path/to/your/.snorkel-
flow.yaml")
```

Parameters

Name	Type	Default	Info
user_config_yaml	`Optional[str]`	`None`	Path to the `.snorkel-flow.yaml` file.
protocol	`str`	`'http'`	The scheme to use for TDM, Studio, and MinIO.
set_global	`bool`	`True`	Whether to use this YAML file to determine the context globally.
debug	`bool`	`False`	Whether to enable debug mode. Debug mode will print more verbose errors to the screen inside of the Python notebook.

Returns

A SnorkelFlowContext object that can be used to interact with the MinIO, TDM, and Studio clients directly.

Return type

SnorkelFlowContext

get_global

classmethod get_global()

Retrieve the global SnorkelFlowContext object.

Examples

```
import snorkelflow.client as sf
ctx = sf.SnorkelFlowContext.get_global()
```

Returns

A SnorkelFlowContext object that can be used to interact with the MinIO, TDM, and Studio clients directly.

Return type

SnorkelFlowContext

Raises

AttributeError – If no global context has been set.

get_minio_client

get_minio_client()

Retrieve an S3 resource for the Minio client. All methods for the retrieved resource can be found in the Boto3 documentation.

Examples

```
import snorkelflow.client as sf
ctx = sf.SnorkelFlowContext.from_kwargs()
mc = ctx.get_minio_client()
mc.create_bucket(Bucket="my-bucket")
```

```
# Create a new bucket
mc = ctx.get_minio_client()
mc.create_bucket(Bucket="my-bucket")

# List all objects in a bucket
bucket = mc.Bucket("my-bucket")
for obj in bucket.objects.all():
    print(obj)
```

Return type

`S3.ServiceResource`

Raises

RuntimeError – If no valid connection to the MinIO client can be established.

set_debug

set_debug(*debug*)

Set the verbosity of warnings, details, and stacktraces

Parameters

Name	Type	Default	Info
debug	`bool`		If True, SDK operations will print more verbose errors to the screen inside of the Python notebook.

Return type

`None`

set_global

classmethod set_global(*ctx=None*)

Set a SnorkelFlowContext object globally. This context object will be used by all Snorkel Flow SDK functions.

Examples

```
import snorkelflow.client as sf
ctx = sf.SnorkelFlowContext.from_kwargs(...)
sf.SnorkelFlowContext.set_global(ctx)
```

Parameters

Name	Type	Default	Info
ctx	`Optional[SnorkelFlowContext]`	`None`	A context to set as global. If no context is provided, will attempt to create one with default parameters.

Return type

`None`

property **storage_client**: *StorageClient*

property **workspace_name**: *str*

SnorkelFlowContext objects are only scoped to work in a particular workspace.

Returns

The name of the workspace belonging to this context object.

Return type

`str`

snorkelflow.client.ctx.SnorkelFlowPredictionAPIClient

snorkelflow.client.ctx.SnorkelFlowPredictionAPIClient

alias of `HTTPClient`

snorkelflow.client.custom_pip

Custom pip install related functions.

Functions

custom_pip_install(package, *[, version, ...])	Install a Python package from a package index for use in LFs and operators.
custom_pip_install_factory_reset()	Perform a factory reset of the all custom-installed Python packages.
custom_pip_install_wheel(path)	Install a Python package from a wheel file for use in LFs and operators.
list_custom_pip_installs()	List all custom installed packages along with their installed dependencies.

snorkelflow.client.custom_pip.custom_pip_install

snorkelflow.client.custom_pip.custom_pip_install(*package*, *, *version=None*, *index_url=None*)

Install a Python package from a package index for use in LFs and operators.

> ⚠ WARNING
> Custom Python package installation is an experimental feature. If something goes wrong and logs suggest an issue involving a package installed in the `.snorkel-flow-pip` directory, use `snorkelflow.client.custom_pip_install_factory_reset` to perform a factory reset of the internal custom Python package store.

Examples

Install a Python package and use it in a custom LF.

```
sf.custom_pip_install("phonenumbers")
@labeling_function(name="my_lf")
def lf(x):  # type: ignore
    import phonenumbers
    if phonenumbers.PhoneNumberMatcher(x.text, "US"):
        return "contact_info"
    return "UNKNOWN"
```

Parameters

Name	Type	Default	Info
package	`str`		The name of the Python package in the index to install.
version	`Optional[str]`	`None`	The version of the Python package to install.
index_url	`Optional[str]`	`None`	Base URL of the package index (defaults to PyPI).

Return type

`None`

snorkelflow.client.custom_pip.custom_pip_install_factory_reset

snorkelflow.client.custom_pip.custom_pip_install_factory_reset()

Perform a factory reset of the all custom-installed Python packages. :rtype: `None`

> ⚠ WARNING
>
> LFs and operators that imported previously installed custom Python packages will fail during execution since the Python packages are no longer available. Either delete these LFs/operators, or reinstall the required packages after performing a factory reset.

snorkelflow.client.custom_pip.custom_pip_install_wheel

snorkelflow.client.custom_pip.custom_pip_install_wheel(*path*)

Install a Python package from a wheel file for use in LFs and operators.

The recommended method is to upload the wheel file to MinIO, and use the `minio://` address in the `path` parameter.

> ⚠ WARNING
>
> Custom Python package installation is an experimental feature. If something goes wrong and logs suggest an issue involving a package installed in the `.snorkel-flow-pip` directory, use `snorkelflow.client.custom_pip_install_factory_reset` to perform a factory reset of the internal custom Python package store.

Examples

Install a Python package from a wheel file and use it in a custom LF. The wheel file has already been uploaded to the `custom_whls` MinIO bucket.

```
import snorkelflow.client as sf
sf.custom_pip_install_wheel("minio://custom_whls/phonenumbers-8.12.18-py2.py3-none-any.whl")
@labeling_function(name="my_lf")
def lf_2(x):  # type: ignore
    import phonenumbers
    if phonenumbers.PhoneNumberMatcher(x.text, "US"):
        return "contact_info"
    return "UNKNOWN"
```

Parameters

Name	Type	Default	Info
path	`str`		Path to the `.whl` file.

Return type

`None`

snorkelflow.client.custom_pip.list_custom_pip_installs

snorkelflow.client.custom_pip.list_custom_pip_installs()

List all custom installed packages along with their installed dependencies.

Return type

`Dict[str, str]`

snorkelflow.client.dataset_views

Dataset views functions for datasets

Functions

create_dataset_view(dataset, name, ...[, ...])	Creates a dataset view for a specific dataset.
delete_dataset_view(dataset_view_uid)	Remove an existing dataset view by its UID.
get_dataset_view(dataset_view_uid)	Get an existing dataset view by its UID.
get_dataset_views(dataset[, label_schema_uids])	Get all dataset views for a dataset using the dataset UID or name.
update_dataset_view(dataset_view_uid, ...)	Updates the list of label schemas associated with a dataset view by passing a list of label schema uids.

snorkelflow.client.dataset_views.create_dataset_view

snorkelflow.client.dataset_views.create_dataset_view(*dataset, name, view_type, column_mapping, label_schema_uids=None*)

Creates a dataset view for a specific dataset.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		The name or UID of the dataset.
name	`str`		The name of the dataset vie
view_type	`str`		Type of the dataset view. “single_llm_response_view” “ranking_llm_responses_vie
column_mapping	`Dict[str, str]`		Map of expected columns tc user-provided columns. Expected columns are “instruction” and “response” Optional columns are “prompt_prefix” and “context”. Example: {“instruction”: “col1”, “response”: “col2”}.
label_schema_uids	`Optional[List[int]]`	`None`	List of label schemas associated for that view.

Returns

A dataset view object.

Return type

`Dict[str, Any]`

Examples

```
>>> sf.create_dataset_view(
>>>     dataset=DATASET_NAME,
>>>     name="my-dataset-view",
>>>     view_type="single_llm_response_view",
>>>     column_mapping={"instruction": "col1", "response": "col2"},
>>> )
{'dataset_view_uid': <dataset_view>, 'name': "my-dataset-view", ...}
```

snorkelflow.client.dataset_views.delete_dataset_view

snorkelflow.client.dataset_views.delete_dataset_view(*dataset_view_uid*)

Remove an existing dataset view by its UID.

Examples

```
>>> sf.delete_dataset_view(dataset_view_uid=7)
None
```

Parameters

Name	Type	Default	Info
dataset_view_uid	`int`		UID of the dataset view to delete.

Return type

None

snorkelflow.client.dataset_views.get_dataset_view

snorkelflow.client.dataset_views.get_dataset_view(*dataset_view_uid*)

Get an existing dataset view by its UID.

Examples

```
>>> sf.get_dataset_view(dataset_view_uid=7)
{'dataset_view_uid': 12, 'name': "my-dataset-view", ...}
```

Parameters

Name	Type	Default	Info
dataset_view_uid	`int`		UID of the dataset view to delete.

Return type

`List[Dict[str, Any]]`

snorkelflow.client.dataset_views.get_dataset_views

snorkelflow.client.dataset_views.get_dataset_views(*dataset, label_schema_uids=None*)

Get all dataset views for a dataset using the dataset UID or name.

Examples

```
>>> sf.get_dataset_views(dataset=DATASET_NAME)
[{'dataset_view_uid': 12, 'name': "my-dataset-view", ...}, {...}]
```

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		The name or UID of the dataset.
label_schema_uids	`Optional[List[int]]`	`None`	List of label schemas associated for that view.

Return type

`List`[`Dict`[`str`, `Any`]]

snorkelflow.client.dataset_views.update_dataset_view

snorkelflow.client.dataset_views.update_dataset_view(*dataset_view_uid, label_schema_uids*)

Updates the list of label schemas associated with a dataset view by passing a list of label schema uids. To remove all the mappings, pass [] as label_schema_uids.

Examples

```
>>> sf.update_dataset_view(dataset_view_uid=7, label_schema_uids=[5])
None
```

Parameters

Name	Type	Default	Info
dataset_view_uid	`int`		UID of the dataset view.
label_schema_uids	`List[int]`		list of label schema UIDs.

Return type

`None`

snorkelflow.client.datasets

Dataset related functions.

Functions

create_dataset(dataset)	Create a dataset.
delete_dataset(dataset[, force])	Delete a dataset.
get_dataset_data(dataset[, split, ...])	Load raw data for the given dataset (prior to applying any processors).
get_datasets()	Get all datasets objects.

snorkelflow.client.datasets.create_dataset

snorkelflow.client.datasets.create_dataset(*dataset*)

Create a dataset.

Specify the *workspace_name* in SnorkelFlowContext to create a dataset in a non-default workspace.

Deprecated since version 2024.R4: Use `snorkelflow.sdk.Dataset.create()` instead.

Parameters

Name	Type	Default	Info
dataset	`str`		Name of the dataset to create.

Returns

UID of the created dataset

Return type

`int`

snorkelflow.client.datasets.delete_dataset

snorkelflow.client.datasets.delete_dataset(*dataset, force=False*)

Delete a dataset.

The operation will fail if any tasks belong to the dataset.

Deprecated since version 2024.R4: Use `snorkelflow.sdk.Dataset.delete()` instead.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		Name or UID of the dataset to delete.
force	`bool`	`False`	If True, delete any applications using the dataset as well.

Return type

`None`

snorkelflow.client.datasets.get_dataset_data

snorkelflow.client.datasets.get_dataset_data(*dataset, split=None, start_date=None, end_date=None, target_columns=None*)

Load raw data for the given dataset (prior to applying any processors).

Deprecated since version 2024.R4: Use `snorkelflow.sdk.Dataset.get_dataframe()` instead.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		Name or UID of the dataset to load unlabeled dataset from.
split	`Optional[str]`	`None`	Name of split (*"train"*, "*valid*", "*test*") to load. None means load all splits.
start_date	`Optional[str]`	`None`	Fetch data starting from this date. Defaults to minus infinity.
end_date	`Optional[str]`	`None`	Fetch data up to this date. Defaults to infinity.
target_columns	`Optional[List[str]]`	`None`	Optional list of columns needed in dataframe. Default to all columns.

Returns

An [n_data_points x n_fields] Pandas DataFrame containing the dataset data.

Return type

`DataFrame`

snorkelflow.client.datasets.get_datasets

snorkelflow.client.datasets.get_datasets()

Get all datasets objects.

Deprecated since version 2024.R4: Use `snorkelflow.sdk.Dataset.list()` instead.

Returns

List of all dataset objects

Return type

`List[Dict[str, Any]]`

Examples

```
>>> sf.get_datasets()
[
    {
        'dataset_uid': <dataset_uid>,
        'name': <name>,
        'file_storage_config_uid': <config_uid>,
        'workspace_uid': <workspace_uid>,
    },
    ...
]
```

snorkelflow.client.datasources

Datasource related functions.

Functions

add_active_datasources(node[, ...])	Adds multiple datasources to a node and activates them all.
create_datasource(dataset, path, file_type)	Create a data source.
delete_datasource(dataset, datasource_uid[, ...])	Delete a data source.
get_datasources(dataset)	Get all data sources for a dataset.
get_node_datasources(node[, compute_staleness])	Returns the datasources for a given node.
prep_and_ingest_datasource(dataset, paths, ...)	Create a data source.
put_node_datasource(node, datasource_uid, ...)	Set the active state of a datasource in a node.
refresh_active_datasources(node)	Refreshes the active data sources at a node.
split_datasources_by_percent(dataset, ...[, ...])	Split datasources given a list of paths by percent, and optionally also stratify ground truth across splits.
update_datasource(datasource_uid, split)	Update split for a data source, in all nodes that have that datasource.

snorkelflow.client.datasources.add_active_datasources

snorkelflow.client.datasources.add_active_datasources(*node, datasource_uids_to_load=None, sync=False*)

Adds multiple datasources to a node and activates them all.

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to add a datasource to.
datasource_uids_to_load	`Optional[List[int]]`	`None`	A list of datasource identifiers to load at the node, default (None) is to load all datasources.
sync	`bool`	`False`	Boolean if set to True this call will block until the tdm job has completed.

Returns

Info on the request

Return type

`Dict[str, Any]`

Examples

```
>>> datasources = sf.get_datasources(DATASET_NAME)
>>> datasources_uid = [ds['datasource_uid'] for ds in datasources]
>>> sf.add_active_datasources(node, datasources_uid)
{'node_uid': <node_uid>, 'job_id': <job_uid>}
```

snorkelflow.client.datasources.create_datasource

snorkelflow.client.datasources.create_datasource(*dataset, path, file_type, uid_col=None, split='train', datasource_ds=None, reader_kwargs=None, credential_kwargs=None, scheduler=None, load_to_model_nodes=False, sync=True*)

Create a data source.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		Name or UID of the dataset to create the data source in.
path	`str`		Path to the data source (e.g. minio, s3, http).
file_type	`str`		File type (csv or parquet).
uid_col	`Optional[str]`	`None`	Name of the UID column in the data source. The values in this column must be unique non-negative integers that are not duplicated across files. If not specified, we will generate a SnorkelFlow ID column.
split	`str`	`'train'`	Split of the dataset to add data source to (train, valid, or test).

datasource_ds	`Optional[str]`	`None`	Datestamp of the data source in YYYY-MM-DD format.
reader_kwargs	`Optional[Dict[str, Any]]`	`None`	Dictionary of keyword arguments to pass to Dask read functions.
credential_kwargs	`Optional[Dict[str, Any]]`	`None`	Dictionary of credentials for specific data connectors.
scheduler	`Optional[str]`	`None`	Dask scheduler (threads, client, or group) to use.
load_to_model_nodes	`bool`	`False`	Load datasource in all tasks in dataset?.
sync	`bool`	`True`	Poll job status and block until complete?.

Returns

UID of the created data source if sync mode used. Otherwise, return job_id

Return type

`Union[str, int]`

snorkelflow.client.datasources.delete_datasource

snorkelflow.client.datasources.delete_datasource(*dataset*, *datasource_uid*, *force=False*, *sync=True*)

Delete a data source.

The operation will fail if any applications are using the data source.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		Name or UID of the dataset the data source belongs to.
datasource_uid	`int`		UID of the data source to delete.
force	`bool`	`False`	(Optional) boolean allowing one to force deletion of a datasource even if that datasource has dependent assets (ground truth, annotations, etc).
sync	`bool`	`True`	Poll job status and block until complete?.

Returns

Optionally returns job_id if sync mode is turned off

Return type

`Optional[int]`

snorkelflow.client.datasources.get_datasources

snorkelflow.client.datasources.get_datasources(*dataset*)

Get all data sources for a dataset.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		Name or UID of the dataset to retrieve data sources for.

Returns

List of all data source objects for the given dataset

Return type

`List[Dict[str, Any]]`

Examples

```
>>> sf.get_datasources(DATASET_NAME)
[
    {
        'datasource_uid': <datasource_uid>,
        'type': <type_int>,
        'config':  <config_dict>,
        'split': <split>, # E.g., Train, Valid, Test
        'ds': <timestamp>,
        'metadata': <metadata_dict>,
    }
    ...
]
```

snorkelflow.client.datasources.get_node_datasources

snorkelflow.client.datasources.get_node_datasources(*node, compute_staleness=False*)

Returns the datasources for a given node.

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to be committed.
compute_staleness	`bool`	`False`	An optional setting to compute if a node is stale.

Returns

A list of datasource objects for a given node

Return type

`List[Dict[str, Any]]`

Examples

```
>>> sf.get_node_datasources(NODE_UID)
[
    {
        'datasource_uid': <datasource_uid>,
        'type': <type_int>,
        'config':  <config_dict>,
        'split': <split>, # E.g., Train, Valid, Test
        'ds': <timestamp>,
        'metadata': <metadata_dict>,
    }
    ...
]
```

snorkelflow.client.datasources.prep_and_ingest_datasource

snorkelflow.client.datasources.prep_and_ingest_datasource(*dataset, paths, input_type, split, run_datasource_checks=True*)

Create a data source.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		Name or UID of the dataset to create the data source in.
paths	`List[str]`		List of paths to the data source (e.g. MinIO, S3).
input_type	`str`		Type of input type of the files in the folder to be processed (eg. pdf, image) The supported types are `pdf`, `image`, and `hocr`
split	`str`		Split of the dataset to add data source to (train, valid, or test).
run_datasource_checks	`bool`	`True`	Whether we should run data source checks before ingestion (defaults to True).

Returns

UID of the created data source if sync mode used

Return type

`Optional[int]`

Examples

```
>>> sf.prep_and_ingest_datasource(
>>>     dataset="test-dataset",
>>>     paths=["minio://pdf-bucket/"],
>>>     input_type="pdf",
>>>     split="train",
>>> )
1
```

snorkelflow.client.datasources.put_node_datasource

snorkelflow.client.datasources.put_node_datasource(*node, datasource_uid, is_active*)

Set the active state of a datasource in a node.

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to add a datasource to.
datasource_uid	`int`		The identifier for a specific datasource.
is_active	`bool`		The desired state of the datasource.

Return type

`None`

snorkelflow.client.datasources.refresh_active_datasources

snorkelflow.client.datasources.refresh_active_datasources(*node*)

Refreshes the active data sources at a node.

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to add a datasource to.

Returns

Info on the request

Return type

`Dict[str, Any]`

Examples

```
>>> sf.refresh_active_datasources(NODE_UID)
{
    'node_uid': <node_uid>,
    'job_id': <job_id>
}
```

snorkelflow.client.datasources.split_datasources_by_percent

snorkelflow.client.datasources.split_datasources_by_percent(*dataset, uid_col, paths, source_type, split_random_seed=None, split_pct=None, gt_col=None, unknown_gt_value=None*)

Split datasources given a list of paths by percent, and optionally also stratify ground truth across splits.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		The name or UID of the dataset into which we are adding these datasources. If a name is provided, the one in the default workspace is used.
uid_col	`str`		UID column for the datasources.
paths	`List[str]`		List of paths that we want to merge and split into train/test/valid.
split_random_seed	`Optional[int]`	`None`	Random seed value for the splitting function.
source_type	`str`		The format of the file(s) we want to split, eg. “PARQUET”.
split_pct	`Optional[Dict[str, float]]`	`None`	Dict containing train/test/valid split ratios.
gt_col	`Optional[str]`	`None`	The column that contains the labels, needed if

			stratifying GT.
unknown_gt_value	`Optional[Any]`	`None`	The value of unknown/null in the GT col, needed if stratifying GT.

Returns

Mapping of split -> file path, for each train/test/valid

Return type

`Dict`

Examples

```
>>> sf.split_datasources_by_percent(
>>>     dataset=dataset_uid,
>>>     source_type="CSV",
>>>     paths=[csv1, csv2, csv3],
>>>     uid_col="context_uid",
>>>     split_pct={'train': 0.6, 'test': 0.2, 'valid': 0.2},
>>> )
{
    <split_path>: <split> # Split is one of train, valid, or test
    ...
}
```

snorkelflow.client.datasources.update_datasource

snorkelflow.client.datasources.update_datasource(*datasource_uid, split*)

Update split for a data source, in all nodes that have that datasource.

Datasource uids used in nodes can be found with `get_node_datasources()`.

Parameters

Name	Type	Default	Info
datasource_uid	`int`		UID of the data source to update.
split	`str`		split to re-assign the data source to.

Returns

List of uids of all nodes that have the updated datasource

Return type

`List[int]`

snorkelflow.client.evaluation

Evaluation related functions.

Functions

`create_evaluation_report`(dataset, metric_schemas)	Start a job to compute metrics for a dataset split.
`preview_custom_prompt_metric`(dataset, ...[, ...])	Preview the results of a custom prompt metric on a subset of data.

snorkelflow.client.evaluation.create_evaluation_report

snorkelflow.client.evaluation.create_evaluation_report(*dataset, metric_schemas, split=None, slices=None, models=None*)

Start a job to compute metrics for a dataset split.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		Name or UID of the dataset to load unlabeled dataset from.
metric_schemas	`List[MetricSchema]`		Create built-in MetricSchema objects.
split	`Optional[str]`	`None`	[DEPRECATED] Name of the split to evaluate.
slices	`Optional[List[int]]`	`None`	UID of the slices for which to compute metrics If no slices are provide, all slices in the given dataset will be evaluated.
models	`Optional[List[int]]`	`None`	UID of the models for which to compute metrics If no models are provided, metrics for all models will be computed All models must map to sources associated with datasources in the given dataset.

Returns

A dictionary containing the evaluation results

Return type

`Dict[str, Any]`

snorkelflow.client.evaluation.preview_custom_prompt_metric

snorkelflow.client.evaluation.preview_custom_prompt_metric(*dataset, x_uids, metric_schema, sync=True*)

Preview the results of a custom prompt metric on a subset of data.

This function allows testing custom prompt metric configurations before creating an evaluation report by running the prompt directly on a sample of data points.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		Name or UID of the dataset to evaluate.
x_uids	`List[str]`		List of data point UIDs to evaluate.
metric_schema	`MetricSchema`		The metric schema to use for evaluation. Must be a CustomPromptMetricSchema.
sync	`bool`	`True`	Whether to wait for results (True) or return job ID (False).

Returns

If sync=True, returns DataFrame with evaluation results If sync=False, returns job ID string

Return type

`Union[pd.DataFrame, str]`

Raises

TypeError – If metric_schema is not a CustomPromptMetricSchema

snorkelflow.client.external_models

External model endpoints related functions. External models endpoints are used to configure specific models from foundation model providers.

Functions

delete_external_model_endpoint(model_name)	Removes an external model endpoint from DB (Only for superadmin users).
get_external_model_endpoints([model_name, ...])	Gets the external model endpoints from DB (Only for superadmin users).
set_external_model_endpoint(model_name, ...)	Adds an external model endpoint to DB (Only for superadmin users).

snorkelflow.client.external_models.delete_external_model_endpoint

snorkelflow.client.external_models.delete_external_model_endpoint(*model_name*)

Removes an external model endpoint from DB (Only for superadmin users).

Parameters

Name	Type	Default	Info
model_name	`str`		Name of the model.

Return type

None

snorkelflow.client.external_models.get_external_model_endpoints

snorkelflow.client.external_models.get_external_model_endpoints(*model_name=None, detail=False*)

Gets the external model endpoints from DB (Only for superadmin users).

Parameters

Name	Type	Default	Info
model_name	`Optional[str]`	`None`	Name of the model to retrieve the endpoint or configuration for. Defaults to None to return specified information for all models.
detail	`bool`	`False`	Whether to return the full configuration for each model. Defaults to False to return only the endpoint URL.

Returns

Mapping of model name to model configuration

Return type

`Dict[str, Any]`

snorkelflow.client.external_models.set_external_model_endpoint

snorkelflow.client.external_models.set_external_model_endpoint(*model_name, endpoint, model_provider, fm_type, **config_kwargs*)

Adds an external model endpoint to DB (Only for superadmin users). NOTE: this will impact all users who elect to use *model_name*

Parameters

Name	Type	Default	Info
model_name	`str`		Name of the model.
endpoint	`str`		Endpoint of the model.
model_provider	`str`		Name of the model provider (one of: hugging_face, local_inference_service, openai, vertexai_lm).
fm_type	`str`		The model type of the foundation mode (one of: text2text, qa, docvqa).
config_kwargs	`Any`		Any additional config options to set for the model.

Return type

`None`

snorkelflow.client.file_storage_configs

File storage config related functions.

Functions

create_file_storage_config(name, base_path)	Create a file storage config.
delete_file_storage_config(...)	Deletes the file storage config
get_file_storage_config(file_storage_config_name)	Get a specific file storage config based on the name
get_file_storage_configs()	Get a list of all file storage configs
set_default_file_storage_config(...)	Set the file storage config as the default

snorkelflow.client.file_storage_configs.create_file_storage_config

snorkelflow.client.file_storage_configs.create_file_storage_config(*name, base_path*)

Create a file storage config.

Parameters

Name	Type	Default	Info
name	`str`		The name of the new file storage config.
base_path	`str`		The base path to store files in. Ex: If the desired storage path is s3://snorkel-data/folder, the base path is snorkel-data/folder.

Returns

The created file storage config object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.create_file_storage_config("remote", "example-bucket/inner-
folder")
{
    'file_storage_config_uid': <config_uid>,
    'name': 'remote',
    'is_default': <bool>,
    'base_path': 'example-bucket/inner-folder'
}
```

snorkelflow.client.file_storage_configs.delete_file_storage_config

snorkelflow.client.file_storage_configs.delete_file_storage_config(*file_storage_config_name*)

Deletes the file storage config

Parameters

Name	Type	Default	Info
file_storage_config_name	`str`		The name of the storage config to delete.

Return type

`None`

snorkelflow.client.file_storage_configs.get_file_storage_config

snorkelflow.client.file_storage_configs.get_file_storage_config(*file_storage_config_name*)

Get a specific file storage config based on the name

Parameters

Name	Type	Default	Info
file_storage_config_name	`str`		The name of the file storage config.

Returns

The requested file storage config object

Return type

`Dict[str, Any]`

Examples

```
>>> sf.get_file_storage_config(file_storage_config_name="minio")
{
    'file_storage_config_uid': <config_uid>,
    'name': <storage_config_name>,
    'is_default': <bool>,
    'base_path': <base_path_string>,
}
```

snorkelflow.client.file_storage_configs.get_file_storage_configs

snorkelflow.client.file_storage_configs.get_file_storage_configs()

Get a list of all file storage configs

Returns

A list of all file storage config objects

Return type

`List[Dict[str, Any]`

Examples

```
>>> sf.get_file_storage_configs()
[
    {
        'file_storage_config_uid': <config_uid>,
        'name': <storage_config_name>,
        'is_default': <bool>,
        'base_path': <base_path_string>,
    }
    ...
]
```

snorkelflow.client.file_storage_configs.set_default_file_storage_config

snorkelflow.client.file_storage_configs.set_default_file_storage_config(*file_storage_config_name*)

Set the file storage config as the default

Parameters

Name	Type	Default	Info
file_storage_config_name	`str`		The name of the default storage config.

Return type

None

snorkelflow.client.files

File storage related functions to upload and download files and directories.

Functions

download_dir(remote_path, local_path)	Downloads remote directory from Snorkel Files Store to local directory.
download_file(remote_path, local_path)	Downloads remote file from Snorkel Files Store to local file.
list_dir(remote_path)	Lists files in remote directory from Snorkel Files Store.
upload_dir(local_path, remote_path)	Uploads a local directory to the Snorkel Files Store.
upload_file(local_path, remote_path)	Uploads a file to the Snorkel Files Store.

snorkelflow.client.files.download_dir

snorkelflow.client.files.download_dir(*remote_path, local_path*)

Downloads remote directory from Snorkel Files Store to local directory. Files and subdirectories inside the remote directory will be placed directly in the local directory. Both absolute paths (ex. minio://workspace-1/data_dir) and relative paths (ex. data_dir, workspace-1/data_dir) are supported.

> ⚠ WARNING
> If you're including the workspace prefix in the remote path, the workspace prefix (workspace-{#}) must match the current workspace. Relative paths will be resolved by removing the workspace prefix: workspace-1/data_dir -> data_dir.

Example

```
remote_path = "minio://workspace-1/data_dir" # equivalent to "data_dir"
local_path = "/home/user/download_dir"

# Files and sub-directories under `remote_path` will be downloaded to
/home/user/download_dir
sf.download_dir(remote_path, local_path)

# To preserve the directory name, you can specify the local path like
this:
sf.download_dir(remote_path, "/home/user/download_dir/data_dir")

# These calls will raise a ValueError
sf.download_dir("minio://workspace-{not-current-workspace-id}/data_dir",
local_path)
sf.download_dir("workspace-{not-current-workspace-id}/data_dir",
local_path)
```

Parameters

Name	Type	Default	Info
remote_path	`str`		Path of remote directory to be downloaded.
local_path	`str`		Local directory to download file to.

Return type

`None`

snorkelflow.client.files.download_file

snorkelflow.client.files.download_file(*remote_path, local_path*)

Downloads remote file from Snorkel Files Store to local file. Both absolute paths (ex. minio://workspace-1/data/file.txt) and relative paths (ex. data/file.txt, workspace-1/data/file.txt) are supported.

> ⚠ WARNING
>
> If you're including the workspace prefix in the remote path, the workspace prefix (workspace-{#}) must match the current workspace. Relative paths will be resolved by removing the workspace prefix: workspace-1/file.txt -> file.txt.

Example

```
remote_path = "minio://workspace-1/data/file.txt"
local_path = "/home/user/file.txt"

# File will be downloaded to /home/user/file.txt
sf.download_file(remote_path, local_path)

# These calls will raise a ValueError
sf.download_file("minio://workspace-{not-current-workspace-
id}/data/file.txt", local_path)
sf.download_file("workspace-{not-current-workspace-id}/data/file.txt",
local_path)
```

Parameters

Name	Type	Default	Info
remote_path	`str`		Path of file to be downloaded.
local_path	`str`		Local path to download file to.

Return type

`None`

snorkelflow.client.files.list_dir

snorkelflow.client.files.list_dir(*remote_path*)

Lists files in remote directory from Snorkel Files Store. Both absolute paths (ex. minio://workspace-1/data_dir) and relative paths (ex. data_dir, workspace-1/data_dir) are supported.

> ⚠ WARNING
>
> If you're including the workspace prefix in the remote path, the workspace prefix (workspace-{#}) must match the current workspace. Relative paths will be resolved by removing the workspace prefix: workspace-1/data_dir -> data_dir.

Example

```
remote_path = "minio://workspace-1/data_dir" # equivalent to "data_dir"

# Files in the remote directory will be listed
sf.list_dir(remote_path)

# These calls will raise a ValueError
sf.list_dir("minio://{not-current-workspace-id}/data_dir")
sf.list_dir("{not-current-workspace-id}/data_dir")
```

Parameters

Name	Type	Default	Info
remote_path	`str`		Path to directory to be listed.

Returns

List of files in specified remote directory

Return type

`List[Any]`

snorkelflow.client.files.upload_dir

snorkelflow.client.files.upload_dir(*local_path*, *remote_path*)

Uploads a local directory to the Snorkel Files Store. Both absolute paths (ex. minio://workspace-1/data_dir) and relative paths (ex. data_dir, workspace-1/data_dir) are supported.

> ⚠ WARNING
>
> If you're including the workspace prefix in the remote path, the workspace prefix (workspace-{#}) must match the current workspace. Relative paths will be resolved by removing the workspace prefix: workspace-1/data_dir -> data_dir.

Example

```
local_path = "/home/user/data_dir"
remote_path = "data_dir"

# Directory will be uploaded to minio://workspace-{current-workspace-id}/data_dir
sf.upload_dir(local_path, remote_path)

# These calls will raise a ValueError
sf.upload_dir(local_path, "minio://workspace-{not-current-workspace-id}/data_dir")
sf.upload_dir(local_path, "workspace-{not-current-workspace-id}/data_dir")
```

Parameters

Name	Type	Default	Info
local_path	`str`		Directory containing files to be uploaded.
remote_path	`str`		Remote directory on Snorkel Files Store to upload files to.

Returns

Tuple containing:

- List of uploaded file paths
- Uploaded directory path

Return type

`Tuple[List[str], str]`

snorkelflow.client.files.upload_file

snorkelflow.client.files.upload_file(*local_path, remote_path*)

Uploads a file to the Snorkel Files Store. Both absolute paths (ex. minio://workspace-1/file.txt) and relative paths (ex. file.txt, workspace-1/file.txt) are supported.

> ⚠ WARNING
>
> If you're including the workspace prefix in the remote path, the workspace prefix (workspace-{#}) must match the current workspace. Relative paths will be resolved by removing the workspace prefix: workspace-1/file.txt -> file.txt.

Example

```
local_path = "/home/user/file.txt"
remote_path = "file.txt"

# File will be uploaded to minio://workspace-{current-workspace-id}/file.txt
sf.upload_file(local_path, remote_path)

# These calls will raise a ValueError
sf.upload_file(local_path, "minio://workspace-{not-current-workspace-id}/file.txt")
sf.upload_file(local_path, "workspace-{not-current-workspace-id}/file.txt")
```

Parameters

Name	Type	Default	Info
local_path	`str`		Path to file to be uploaded.
remote_path	`str`		File path in Snorkel Files Store to upload file to.

Returns

The uploaded file path

Return type

`str`

snorkelflow.client.fm_suite

Foundation model suite related functions. Provides functions for estimating costs, viewing available methods, running jobs, and monitoring progress. Allows configuring and kicking off warm start workflows to generate labeling functions using foundation models.

Functions

create_prompt_lf(node, model_name, ... [, ...])	Kick off a prompt inference job on a given node.
get_available_warm_start_methods(node)	Get the available warm start methods for a given node.
get_warm_start_cost(node, configs[, split, ...])	Estimate the additional cost in USD of warm start methods for a given node.
preview_prompt_lf(node, model_name, ... [, ...])	Preview a prompt inference job on a single data split of a given node.
prompt_fm(prompt, model_name[, model_type, ...])	Send one or more prompts to a foundation model
prompt_fm_over_dataset(prompt_template, ...)	Run a prompt over a dataset.
run_lf_inference(node, lf_uid[, ...])	Run the LF on new data.
run_warm_start(node, warm_start_method, ...)	Kick off an FM Warm Start job on a given node.

snorkelflow.client.fm_suite.create_prompt_lf

snorkelflow.client.fm_suite.create_prompt_lf(*node, model_name, model_type, prompt_text, lf_name, primary_text_field=None, label=None, threshold=None, output_code=None, split=None, sync=False, **model_kwargs*)

Kick off a prompt inference job on a given node.

Parameters

Name	Type	Default	
node	`int`		Node uid to run Warm Start
model_name	`str`		The unique name of the FM t
model_type	`str`		The type of predictor use. Mu Model Type | Application | F +===== | “text2text” | Single-Label cl ——————-+ | “te —————————— ——————-+—————— ——————-+—————— Text2Text | +—————— DocVQA | +——————
prompt_text	`str`		The unformatted prompt to t inference,.
lf_name	`str`		The name of the LF to create
primary_text_field	`Optional[str]`	`None`	The name of the text field to `model_type="text2text_`
label	`Optional[str]`	`None`	What to label the data point
threshold	`Optional[float]`	`None`	The cutoff for where a predic

output_code	`Optional[str]`	`None`	Code used to map the FM ou default code mapper based c
split	`Optional[List[str]]`	`None`	List of splits to run inference,
sync	`bool`	`False`	If True, method will block unt Studio or via sf.poll_job_statu
model_kwargs	`Any`		Additional kwargs to pass to to adjust the batch size, spec tokens_in_longest_batched_c

Returns

The uid of the prompt inference job which can be used to monitor progress with sf.poll_job_status(job_uid).

Return type

`job_uid`

Examples

```
sf.create_prompt_lf(
    NODE_UID,
    model_name="google/flan-t5-small",
    lf_name="multilabel_prompt_lf",
    model_type="text2text_multi_label",
    prompt_text="The possible labels are sentence, affirmation,
question. What are the most likely label for the following document:
{Body_q}\n\nLabel: ",
)
```

Note the job progress can always be be monitored within the ‘In Progress’ LFs table on Studio or via sf.poll_job_status(‘123’)

snorkelflow.client.fm_suite.get_available_warm_start_methods

snorkelflow.client.fm_suite.get_available_warm_start_methods(*node*)

Get the available warm start methods for a given node.

Parameters

Name	Type	Default	Info
node	`int`		Node uid which Warm Start will be run over.

Return type

`WarmStartMethodsResults`

Returns

- **note** (*str*) – Optional informative user message indicating how to make available more warm start methods.
- **results** (*List[Dict[str, Any]*) – List of all available warm start methods and their descriptions.

Examples

```
>>> results = sf.fm_suite.get_available_warm_start_methods(NODE_UID)
>>> print(results)
Note: All possible warm start methods for this node type are supported.
```

Methods: sdnet: SDNet refers to Self-describing Networks, where a base foundation model is used in a few-shot setting for Named Entity Recognition. To do this, the foundation model is finetuned in a specific way to learn the underlying concepts which represent the classes. Full details can be found by reading the ‘Few-shot Named Entity Recognition with Self-describing Networks’ paper. Recommended models include: [‘snorkelai/sdnet’]

```
>>> print(results.methods["sdnet"])
```

Description: SDNet refers to Self-describing Networks, where a base foundation model is used in a few-shot setting for Named Entity Recognition. To do this, the foundation model is finetuned in a specific way to learn the underlying concepts which represent the classes. Full details can be found by reading the ‘Few-shot Named Entity Recognition with Self-describing Networks’ paper.

SDNet currently only supports the one recommended model. Recommended Models: snorkelai/sdnet: T5 (Text-To-Text Transfer Transformer) is a transformer-based architecture where all NLP training tasks are reframed into a unified text-to-text format and learned jointly without the need for separate task-specific heads, as in BERT-style models. The base variant has 250M parameters.

snorkelflow.client.fm_suite.get_warm_start_cost

snorkelflow.client.fm_suite.get_warm_start_cost(*node, configs, split=None, input_columns=None, openai_costs=None, allow_unsupported_foundation_model=False*)

Estimate the additional cost in USD of warm start methods for a given node.

Parameters

Name	Type	Default	
node	`int`		Node ui run over
configs	`List[Dict[str, str]]`		List of di containi *provider* estimate methods *get_avai*
split	`Optional[List[str]]`	`None`	Dataset costs. D "valid", "
input_columns	`Optional[List[str]]`	`None`	List of in estimati columns
openai_costs	`Optional[Dict[str, Dict[str, float]]]`	`None`	Inferenc each mo values.
allow_unsupported_foundation_model	`bool`	`False`	If True, e for unsu bypassir validatic

Returns

costs – Estimated cost in USD for each method-model combination

Return type

`List[WarmStartConfigCost]`

Examples

```
>>> costs = sf.get_warm_start_cost(NODE_UID, [{"method":
"zsl_prompt_remote", "provider": "openai", "model": "openai/gpt-4"}])
>>> costs[0].additional_cost
63.20
```

snorkelflow.client.fm_suite.preview_prompt_lf

snorkelflow.client.fm_suite.preview_prompt_lf(*node, model_name, model_type, prompt_text, split, primary_text_field=None, label=None, threshold=None, num_examples=100, output_code=None, **model_kwargs*)

Preview a prompt inference job on a single data split of a given node.

Parameters

Name	Type	Default	
node	`int`		Node uid to run the preview on.
model_name	`str`		The unique name of the FM to use
model_type	`str`		The type of predictor use. Must be Model Type | Application | FM Typ +==========+ | “text2text” | Single-Label classifi ——————-+ | “text2tex —————————————— ——————-+—————— ——————-+—————— Text2Text | +——————-+— DocVQA | +——————-+—
prompt_text	`str`		The unformatted prompt to the FM inference.
split	`str`		The single data split to run the infe
primary_text_field	`Optional[str]`	`None`	The name of the text field to prepe `model_type="text2text_qa"`.
label	`Optional[str]`	`None`	What to label the data point as wh
threshold	`Optional[float]`	`None`	The cutoff for where a prediction f

num_examples	`int`	`100`	The number of examples to run in
output_code	`Optional[str]`	`None`	Code used to map the FM output t
model_kwargs	`Any`		Additional kwargs to pass to the F to adjust the batch size, specify th tokens_in_longest_batched_doc.

Returns

Dataframe containing the predictions for the data points. Columns include: output (LLM output), confidence, label (by the LLM), ground_truth

Return type

`df`

Examples

```
sf.preview_prompt_lf(
    NODE_UID,
    model_name="google/flan-t5-small",
    split="dev",
    model_type="text2text",
    prompt_text="What is the most likely label for the following
document: {Body}\n\nLabel: "
)
```

snorkelflow.client.fm_suite.prompt_fm

snorkelflow.client.fm_suite.prompt_fm(*prompt, model_name, model_type=None, question=None, runs_per_prompt=1, sync=True, cache_name='default', **fm_hyperparameters*)

Send one or more prompts to a foundation model

Parameters

Name	Type	Default	Info
prompt	`Union[str, List[str]]`		The prompt(s) to send to the foundation model.
model_name	`str`		The name of the foundation model to use.
model_type	`Optional[LLMType]`	`None`	The way we should use the foundation model, must be one of the LLMType values.
question	`Optional[str]`	`None`	When provided, this get's passed to the model for each prompt which is useful for information retrieval tasks. The prompt argument essentially then becomes the context(s) which contains the answer to the question.

runs_per_prompt	`int`	`1`	The number of times to run a prompt, note each response can be different. All will be cached.
sync	`bool`	`True`	Whether to wait for the job to complete before returning the result.
cache_name	`str`	`'default'`	The cache name is used in the hash construction. To run a prompt and get a different result, you should change the cache name to something that hasn't been used before. For example: >> sf.prompt_fm("What is the meaning of life?", "openai/gpt-4o") The meaning of life is to work... >> sf.prompt_fm("What is the meaning of life?", "openai/gpt-4o") << hit's the cache The meaning of life is to work... >> sf.prompt_fm("What is the meaning of life?", "openai/gpt-4o", cache_name="run_2")

			<< hit's a different part of the cache The meaning of life is to have fun!.
fm_hyperparameters	`Any`		Additional keyword arguments to pass to the foundation model such as temperature, max_tokens, etc.

Return type

`Union[DataFrame, str]`

Returns

- *df* – Dataframe containing the predictions for the data points. There are two columns, the input prompt and the output of the foundation model.
- *job_id* – The job id of the prompt inference job which can be used to monitor progress with sf.poll_job_status(job_id).

Examples

```
>>> sf.prompt_fm(prompt="What is the meaning of life?",
model_name="openai/gpt-4")
   | text                          | generated_text
| perplexity
------------------------------------------------------------------------
--------------------------------------
0  | What is the meaning of life?   | Life is all about having fun!
| 0.789
```

```
>>> sf.prompt_fm(prompt=["What is the meaning of life?", "What is the
meaning of death?"], model_name="openai/gpt-4")
   | text                          | generated_text
| perplexity
------------------------------------------------------------------------
--------------------------------------
0  | What is the meaning of life?   | Life is all about having fun!
| 0.789
1  | What is the meaning of death?  | Death is about not having fun!
| 0.981
```

```
>>> sf.prompt_fm(question="What is surname", prompt="Joe Bloggs is a
person", model_name="deepset/roberta-base-squad2")
   | text                          | answer         | start  | end   |
score
------------------------------------------------------------------------
--------------------------------------
0  | Joe Bloggs is a person         | Bloggs         | 4      | 11    |
0.985
```

snorkelflow.client.fm_suite.prompt_fm_over_dataset

snorkelflow.client.fm_suite.prompt_fm_over_dataset(*prompt_template, dataset, x_uids, model_name, model_type=None, runs_per_prompt=1, sync=True, cache_name='default', system_prompt=None, **fm_hyperparameters*)

Run a prompt over a dataset. Any field in the dataset can be referenced in the prompt by using curly braces, {field_name}.

Parameters

Name	Type	Default	Info
prompt_template	`str`		The prompt template used to format input rows sent to the foundation model.
dataset	`Union[str, int]`		The name or UID of the dataset containing the data we want to prompt over.
x_uids	`List[str]`		The x_uids of the rows within the dataset to prompt over.
model_name	`str`		The name of the foundation model to use.
model_type	`Optional[LLMType]`	`None`	The way we should use the foundation model, must be one of the LLMType values.

runs_per_prompt	`int`	`1`	The number of times to run inference over an xuid, note each response can be different. All will be cached.
sync	`bool`	`True`	Whether to wait for the job to complete before returning the result.
cache_name	`str`	`'default'`	The cache name is used in the hash construction. To run a prompt and get a different result, you should change the cache name to something that hasn't been used before. For example: >> sf.prompt_fm("What is the meaning of life?", "openai/gpt-4o") The meaning of life is to work... >> sf.prompt_fm("What is the meaning of life?", "openai/gpt-4o") << hit's the cache The meaning of life is to work... >> sf.prompt_fm("What is the meaning of life?", "openai/gpt-4o",

			cache_name="run_2") << hit's a different part of the cache The meaning of life is to have fun!.
system_prompt	`Optional[str]`	`None`	The system prompt to prepend to the prompt.
fm_hyperparameters	`Any`		Additional keyword arguments to pass to the foundation model such as temperature, max_tokens, etc.

Return type

`Union`[`DataFrame`, `str`]

Returns

- *df* – Dataframe containing the predictions for the data points. There are two columns, the input prompt and the output of the foundation model.
- *job_id* – The job id of the prompt inference job which can be used to monitor progress with sf.poll_job_status(job_id).

Examples

```
>>> sf.prompt_fm_over_dataset(prompt_template="{email_subject}. What is
this email about?", dataset=1, x_uids=["0", "1"],
model_name="openai/gpt-4")
   | email_subject                          | generated_text
| perplexity
------------------------------------------------------------------------
------------------------------------------------------------
0  | Fill in survey for $50 amazon voucher  | The email is asking you to
fill in a survey for an amazon voucher       | 0.891
1  | Hey it's Bob, free on Sat?             | The email is from your
friend Bob as if you're free on Saturday         | 0.787
```

snorkelflow.client.fm_suite.run_lf_inference

snorkelflow.client.fm_suite.run_lf_inference(*node, lf_uid, inference_splits=None, sync=False*)

Run the LF on new data. Note, if there are any datapoints for which inference has been computed already the cached result will be used.

Parameters

Name	Type	Default	Info
node	`int`		Node uid which contains the LF and data to run over.
lf_uid	`int`		The LF for which new predictions will be computed.
inference_splits	`Optional[List[str]]`	`None`	Dataset splits to run inference over. Defaults to all, [“train”, “dev”, “valid”, “test”].
sync	`bool`	`False`	If True, method will block until the inference job is complete. Note the job progress can always be be monitored manually with sf.poll_job_status(job_uid).

Returns

job_uid – The uid of the Warm Start job which can be used to monitor progress with sf.poll_job_status(job_uid).

Return type

`str`

Examples

```
>>> sf.run_lf_inference(NODE_UID, LF_UID, ["train", "test"])
Note the job progress can be monitored with sf.poll_job_status('123')
```

snorkelflow.client.fm_suite.run_warm_start

snorkelflow.client.fm_suite.run_warm_start(*node, warm_start_method, foundation_model, lf_name='Warm Start SDK', split=None, columns=None, one_lf_per_class=False, sync=False, allow_unsupported_foundation_model=False, **model_kwargs*)

Kick off an FM Warm Start job on a given node.

Parameters

Name	Type	Default	
node	`int`		Node ui
warm_start_method	`str`		The strir use. If yc sf.get_av first to s node.
foundation_model	`str`		The Fou Start me bigscien
lf_name	`str`	`'Warm Start SDK'`	Name of
split	`Optional[List[str]]`	`None`	Splits to to all, ["t
columns	`Optional[List[str]]`	`None`	Text fiel text colu nodes w SPAN_PI
one_lf_per_class	`bool`	`False`	If True, a class. Th the Wari

sync	`bool`	`False`	If True, r job is co always b LFs table sf.poll_jo
allow_unsupported_foundation_model	`bool`	`False`	If True, r foundati support bypass f
model_kwargs	`Any`		Addition Model. F window kwarg.

Returns

job_uid – The uid of the Warm Start job which can be used to monitor progress with sf.poll_job_status(job_uid).

Return type

`str`

Examples

```
>>> sf.run_warm_start(NODE_UID, "zsl_prompt", "google/flan-t5-base")
Note the job progress can always be be monitored within the 'In
Progress' LFs table on Studio or via sf.poll_job_status('123')
```

snorkelflow.client.gts

Ground truth related functions. Ground truth is the set of labels that are considered to be the "true" labels for a dataset. Ground truth is used to evaluate the performance of a model, calculate labeling function performance, and power analysis and auto-suggest tools. Ground truth labels are associated with a single model node in an application.

Functions

add_ground_truth(node[, x_uids, labels, df, ...])	This function registers ground truth to the Snorkel Flow platform.
align_external_ground_truth(node_uid, ...[, ...])	Note: This SDK is only necessary when you use a non-dataframe format with sf.add_ground_truth for sequence tagging applications.
create_ground_truth_version(node, name[, ...])	Create a new ground truth version for a node.
delete_ground_truth_version(node, gt_version_uid)	Delete a specified ground truth version for a node.
get_document_ground_truth(node[, context_uids])	Get document-level ground truths as Pandas DataFrame.
get_ground_truth(node[, split, start_date, ...])	Get ground truth from the Snorkel Flow platform.
get_inferred_document_ground_truth_from_span_ground_truth(...)	Available only for text extraction and entity classification tasks.
get_span_level_ground_truth_conflicts(node)	Retrieve a list of span-level ground truth conflicts.
list_ground_truth_versions(node)	Fetch a list of all ground truth versions for a node.
load_ground_truth_version(node, gt_version_uid)	Load a specified ground truth version for a node.

snorkelflow.client.gts.add_ground_truth

snorkelflow.client.gts.add_ground_truth(*node, x_uids=None, labels=None, *, df=None, metadata=None, user_format=False, skip_missing=False, auto_generate_negative_labels=False*)

This function registers ground truth to the Snorkel Flow platform. Depending on the use case, the input format for add_ground_truth varies. Please refer to the specific section for your application type. Note that ground truth will be ingested for the particular `node` specified.

1. Single-label Applications

   Provide a list of `labels` corresponding to each data point specified by the `x_uids` argument. The labels are strings if `user_format=True`, and integers if `user_format=False`.

   ```
   >>> x_uids = ["doc::0", "doc::1", "doc::2"]
   >>> labels = ["loan", "services", "employment"]
   >>> sf.add_ground_truth(node, x_uids, labels, user_format=True)
   ```

2. Multi-label Applications

   Provide a list of `labels` corresponding to each data point specified by the `x_uids` argument. If `user_format=True`, each value in `labels` is a `dict` mapping each string label to one of `PRESENT`, `ABSENT`, or `ABSTAIN`. If `user_format=False`, each value in `labels` is a `dict` mapping each integer label to one of `PRESENT`, `ABSENT`, or `ABSTAIN`.

   ```
   >>> x_uids = ["doc::0", "doc::1", "doc::2"]
   >>> labels = [
       {"Japanese Movies": "PRESENT", "World cinema": "ABSTAIN", "Black-and-
   White": "ABSENT", "Short Film": "ABSTAIN"},
       {"Japanese Movies": "ABSTAIN", "World cinema": "PRESENT", "Black-and-
   White": "ABSTAIN", "Short Film": "PRESENT"},
       {"Japanese Movies": "ABSENT", "World cinema": "ABSENT", "Black-and-
   White": "PRESENT", "Short Film": "ABSTAIN"}
   ]
   >>> sf.add_ground_truth(node, x_uids, labels, user_format=True)
   ```

   Note that `add_ground_truth` will not accept serialized JSON. As an example for a multi-label application:

```
>>> x_uids = ["doc::0"]
>>> labels = ['{"Japanese Movies": "PRESENT", "World cinema": "ABSTAIN",
"Black-and-White": "ABSENT", "Short Film": "ABSTAIN"}']
>>> sf.add_ground_truth(node, x_uids, labels, user_format=True)
# This will fail since the labels are JSON serialized as strings
>>> deserialized_labels = [json.loads(label) for label in labels]
>>> sf.add_ground_truth(node, x_uids, deserialized_labels, user_format=True)
```

3. Sequence Tagging Applications

> Provide a dataframe containing the following columns: 1. __DATAPOINT_UID: The unique identifier for each data point. 2. start: The starting position of the labeled span. 3. end: The ending position of the labeled span. 4. label: The label for the span.
>
> Starting with version 0.95, when providing a DataFrame, the labels will be aligned automatically (no need to run `align_external_ground_truth`).
>
> > __DATAPOINT_UID | start | end | label doc::0 | 0 | 10 | COMPANY doc::1 | 0 | 5 | COMPANY doc::1 | 5 | 15 | OTHER
>
> ```
> >>> df = pd.DataFrame({
>     "__DATAPOINT_UID": ["doc::0", "doc::1", "doc::1"],
>     "start": [0, 0, 5],
>     "end": [10, 5, 15],
>     "label": ["COMPANY", "COMPANY", "OTHER"]
> })
> >>> sf.add_ground_truth(node, df=df)
> ```

4. Sequence Tagging Applications (legacy)

> Provide a list of `labels` corresponding to each data point specified by the `x_uids` argument. Each value in `labels` is a list of spans corresponding to each data point. The spans are formatted as `(start, end, label)` where `start` and `end` are the starting and ending indices of the span, and `label` is the label of the span. If `user_format=True`, the labels are strings. If `user_format=False`, the labels are integers.
>
> ```
> >>> x_uids = ["doc::0", "doc::1", "doc::2"]
> >>> labels = [
>     [(0, 10, "COMPANY"), (15, 20, "OTHER")],
>     [(0, 5, "COMPANY")],
>     [(5, 15, "COMPANY")]
> ]
> >>> sf.add_ground_truth(node, x_uids, labels, user_format=True)
> ```

If using the traditional input format (i.e., `x_uids` + `labels`), you must still run `align_external_ground_truth` to align the ground truth spans after preprocessing.

To learn more about the Label format for your use case you can see Format for ground truth interaction in the SDK.

Parameters

Name	Type	Default	
node	`int`		UID of the node for.
x_uids	`Union[List[str], ndarray, None]`	`None`	UIDs of data poi numpy array of
labels	`Union[List[Any], ndarray, None]`	`None`	Label values. Lis labels. Must be t x_uids. If user_fo labels have not l user_format is F instead of string
df	`Optional[DataFrame]`	`None`	Specific for sequ pandas datafrar following colum start, end, label provide one of (`df`
metadata	`Optional[Dict[str, Any]]`	`None`	Metadata to reg (information abo was generated).
user_format	`bool`	`False`	True if the grour provided in user To see a mappir integer represer (for `user_form` `sf.get_node_`

skip_missing	`bool`	`False`	Specific for lega cases. True if sp datasource nee otherwise.
auto_generate_negative_labels	`bool`	`False`	Specific for lega cases. If True, au negative ground

Return type

`None`

snorkelflow.client.gts.align_external_ground_truth

snorkelflow.client.gts.align_external_ground_truth(*node_uid, x_uids, labels, user_format=False, scheduler=None*)

Note: This SDK is only necessary when you use a non-dataframe format with sf.add_ground_truth for sequence tagging applications. Starting with version 0.95, if you use a dataframe with sf.add_ground_truth, the labels will automatically be aligned

(Sequence Tagging Only) This function changes external ground truth spans to compensate for the offsets caused by a text preprocessor.

Text Preprocessors may sometimes remove characters, resulting in misalignments between externally collected ground truth spans and the preprocessed text. For example, the default `AsciiCharFilter` preprocessor removes non-ascii characters which will cause some spans to shift leftwards, but external annotations will still have the original spans. Thus, it is necessary to use this function for applications with non-ascii characters.

Examples

An example for a sequence tagging application with `AsciiCharFilter` as the preprocessor

```
node_uid = sf.get_model_node(APP_NAME)
x_uids = ["doc::0", "doc::1"] # all x_uids with labels
labels = [
    [[0, 20, "COMPANY"]], # labels for doc::0
    [[10, 15, "COMPANY"], [20, 25, "COMPANY"]], # labels for doc::1
    ...
]

aligned_labels = sf.align_external_ground_truth(node_uid,  x_uids,
labels, user_format=True)
```

Then, use `sf.add_ground_truth` to add `aligned_labels`

```
sf.add_ground_truth(node_uid, x_uids, aligned_labels, user_format=True)
```

Parameters

Name	Type	Default	Info

node_uid	`int`		The UID of the model node.
x_uids	`List[str]`		UIDs of data points. Can be a list or a 1D numpy array of strings.
labels	`List[Any]`		Label values. List or numpy array of labels. Must be the same length as x_uids. If user_format is True, check that labels have not been JSON serialized.
user_format	`bool`	`False`	True if labels are provided in user format, False otherwise.
scheduler	`Optional[str]`	`None`	Dask scheduler (threads, client, or group) to use.

Returns

List of aligned labels corresponding to x_uids.

Return type

`List[Any]`

snorkelflow.client.gts.create_ground_truth_version

snorkelflow.client.gts.create_ground_truth_version(*node, name, description=<snorkelflow.client_v3.tdm.types.Unset object>*)

Create a new ground truth version for a node. Creating a ground truth version will yield an integer UID that identifies the ground truth version. The `snorkelflow.client.load_ground_truth_version` function can then be used to restore a previously created ground truth version. Note that ground truth versions are only available in the node the ground truth version was created for.

Examples

```
>>> sf.create_ground_truth_version(node=1, name="my_gt_version",
description="my description")
{'gt_version_uid': 1}
```

Parameters

Name	Type	Default	
node	`int`		Th the to GT for
name	`str`		Na the gro tru ver
description	`Union[Unset, str]`	`<snorkelflow.client_v3.tdm.types.Unset object at 0x74581c032e90>`	An op de of

			gro tru ver

Returns

gt_version_uid: The UID of the new ground truth version

Return type

`Dict[str, int]`

snorkelflow.client.gts.delete_ground_truth_version

snorkelflow.client.gts.delete_ground_truth_version(*node, gt_version_uid*)

Delete a specified ground truth version for a node. This is a destructive operation, and it will not be possible to load this ground truth version once this operation is complete.

Examples

```
>>> sf.list_ground_truth_versions(node=1)
[{'gt_version_uid': 1, ... }]
>>> sf.delete_ground_truth_version(node=1, gt_version_uid=1)
>>> sf.list_ground_truth_versions(node=1)
[]
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to delete the GT version of.
gt_version_uid	`int`		The UID of the ground truth version to delete.

Return type

`None`

snorkelflow.client.gts.get_document_ground_truth

snorkelflow.client.gts.get_document_ground_truth(*node, context_uids=None*)

Get document-level ground truths as Pandas DataFrame.

This function gets the ground truth for an entire input document. It can only be used for information extraction tasks. For text extraction tasks, this gets a list of extractions for each document. For entity classification tasks, this gets a dictionary mapping entities to classes for each document. The ground truth is always represented as integers rather than strings. A mapping from integer to string labels can be retrieved using `sf.get_node_label_map`. For information about how to interpret ground truth in information extraction tasks, see Format for ground truth interaction in the SDK.

Examples

```
>>> sf.get_document_ground_truth(123, context_uids=[456, 789])
<pd.DataFrame>
__DATAPOINT_UID     |   ground_truth
doc::0              |   [[0, 10, 0]]
doc::1              |   [[0, 5, 1], [5, 15, 0]]
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node whose document ground truth to get.
context_uids	`Optional[List[int]]`	`None`	Optional list of context_uids of documents whose ground truth to get. If None, get all document ground truth.

Return type

`DataFrame`

Returns

- *Pandas DataFrame containing document ground truth with*
- *document context_uid as index*

snorkelflow.client.gts.get_ground_truth

snorkelflow.client.gts.get_ground_truth(*node, split=None, start_date=None, end_date=None, is_context=False, user_format=False*)

Get ground truth from the Snorkel Flow platform. The format of the ground truth depends on the task type and whether `user_format` is True or False. For more information, see Format for ground truth interaction in the SDK.

Examples

```
>>> sf.get_ground_truth(node=1, split='train', user_format=True)
<pd.Series>
doc::0      SPAM
doc::1      HAM
doc::2      HAM
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node whose ground-truth we're fetching.
split	`Optional[str]`	`None`	Name of a split whose ground-truth to fetch (if None, return all splits).
start_date	`Optional[str]`	`None`	Start date (e.g., yyyy/mm/dd) to filter ground-truth labels by.
end_date	`Optional[str]`	`None`	End date (e.g., yyyy/mm/dd) to filter ground-truth labels by.
is_context	`bool`	`False`	If False, get datapoint level ground truth. Otherwise, get context level ground truth. Only relevant for information extraction tasks.
user_format	`bool`	`False`	True if the ground truth labels are returned in user format, False

			otherwise.

Returns

A pandas Series containing ground truth labels.

Return type

`pd.Series`

snorkelflow.client.gts.get_inferred_document_ground_truth_from_span_ground_truth

snorkelflow.client.gts.get_inferred_document_ground_truth_from_span_ground_truth(*node, split, reducer_class=None, overwrite_existing_document_ground_truth=False*)

Available only for text extraction and entity classification tasks.

In the case of these task types, we wish to assign strictly one class to the overall document based on how we classify the spans within the document. The Reducer operator lets us do this during model development. However, we also need a way to validate the outputs of that Reducer operator by adding document-level ground truth to each document. This function allows us to infer what the document-level ground truth is based on the span-level ground truth that already exists.

If `overwrite_existing_document_ground_truth` is set to `True`, then this function will overwrite any existing document-level ground truth. Otherwise, this function will only infer document-level ground truth for documents that do not already have document-level ground truth.

Examples

```
>>> sf.get_inferred_document_ground_truth_from_span_ground_truth(1,
"train")
100 # 100 documents had their document-level GT inferred based on the
existing span-level GT for those documents
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node whos document-level ground-truth we're inferring.

split	`str`		Name of th split whose document-level ground-truth to infer.
reducer_class	`Optional[str]`	`None`	Name of th reducer class to apply to data before inferring document-level groun truths (only applicable to entity classificatic tasks).
overwrite_existing_document_ground_truth	`bool`	`False`	Overwrite existing document-level groun truths. By default this function wi not modify any existin document-level groun truths.

Returns

Number of documents that had their ground truths changed

Return type

`int`

snorkelflow.client.gts.get_span_level_ground_truth_conflicts

snorkelflow.client.gts.get_span_level_ground_truth_conflicts(*node, split=None, start_date=None, end_date=None*)

Retrieve a list of span-level ground truth conflicts. The conflicts are sorted by entropy. For a span_text where all ground truth labels are the same, the entropy is 0. For a span_text with multiple different ground truth labels, the entropy > 0. The larger entropy indicates more conflicts.

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node whose ground-truth we're fetching.
split	`Optional[str]`	`None`	Name of a split whose ground-truth to fetch (if None, return all splits).
start_date	`Optional[str]`	`None`	Start date (e.g., yyyy/mm/dd) to filter ground-truth labels by.
end_date	`Optional[str]`	`None`	End date (e.g., yyyy/mm/dd) to filter ground-truth labels by.

Returns

List of dict of span-level conflicts in a DataFrame, sorted by entropy, i.e., the span with most conflicting ground truth labels comes first

Return type

`pd.DataFrame`

snorkelflow.client.gts.list_ground_truth_versions

snorkelflow.client.gts.list_ground_truth_versions(*node*)

Fetch a list of all ground truth versions for a node. The return list will reveal the corresponding ground truth version UIDs for all ground truth versions for this node, which can then be used in `snorkelflow.client.load_ground_truth_version` and `snorkelflow.client.delete_ground_truth_version.`

Examples

```
>>> sf.list_ground_truth_versions(node=1)
[
    {
        'gt_version_uid': 1,
        'node_uid': 1,
        'name': 'my-gt-version',
        'description': 'my description',
        'hidden': False,
        'created_by': 'notebook',
        'created_at': '2023-07-10T20:33:09.323068'
    }
]
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to fetch all GT versions for.

Returns

Returns a list of ground truth versions for this node, displaying the information relevant to each ground truth version object.

Return type

`List[Dict[str, Any]]`

snorkelflow.client.gts.load_ground_truth_version

snorkelflow.client.gts.load_ground_truth_version(*node, gt_version_uid*)

Load a specified ground truth version for a node. This will change the ground truth that you see in Studio. Make sure to save your previous ground truth version with `snorkelflow.client.create_ground_truth_version` before loading a new one to avoid losing any ground truth labels. You can call `sf.poll_job_status` on the `job_uid` returned by this function to track the progress of this operation.

The ground truth labels you load will become available in Developer Studio once this function has finished executing. Keep in mind that you may need to resample the dev split of your selected node to see the new ground truth labels.

Examples

```
>>> sf.load_ground_truth_version(node=1, gt_version_uid=1)
{'job_uid': 'rq-ZZyIRDE7_engine-D2nP_load-ground-truth-version'}
>>> sf.poll_job_status('rq-ZZyIRDE7_engine-D2nP_load-ground-truth-
version')
+0.31s Finished loading ground truth version
# ... ground truth labels are now available in Developer Studio after
resampling the Dev split
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to load a GT version for.
gt_version_uid	`int`		The UID of the ground truth version to load.

Return type

`AsyncJobResponse`

snorkelflow.client.lf_packages

LF package related functions.

Functions

delete_lf_package(node, pkg_uid)	Delete an LF package
export_lf_packages(node, pkg_uids)	Export LF packages
import_lf_packages(node, lf_packages)	Import LF packages.
package_lfs(node[, name])	Package active LFs into an LF package.
transfer_lf_packages(node, from_node[, ...])	Copy LF packages from one node to another.

snorkelflow.client.lf_packages.delete_lf_package

snorkelflow.client.lf_packages.delete_lf_package(*node, pkg_uid*)

Delete an LF package

Parameters

Name	Type	Default	Info
node	`int`		UID of the node containing the LF.
pkg_uid	`int`		UID of the LF package to delete.

Return type

`None`

snorkelflow.client.lf_packages.export_lf_packages

snorkelflow.client.lf_packages.export_lf_packages(*node*, *pkg_uids*)

Export LF packages

Parameters

Name	Type	Default	Info
node	`int`		UID of the node whose ground-truth we're fetching.
pkg_uids	`Union[int, Iterable[int]]`		UIDs of LF packages to export.

Returns

List of LF packages

Return type

List[LFPackage]

snorkelflow.client.lf_packages.import_lf_packages

snorkelflow.client.lf_packages.import_lf_packages(*node, lf_packages*)

Import LF packages.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node whose ground-truth we're fetching.
lf_packages	`Union[LFPackage, Iterable[LFPackage]]`		LF packages to import.

Returns

UIDs of imported LF packages

Return type

`List[int]`

snorkelflow.client.lf_packages.package_lfs

snorkelflow.client.lf_packages.package_lfs(*node, name=None*)

Package active LFs into an LF package. Use `unpackage_lfs()` to unpackage them. Note that new package will not be created if there is no change in the set of active LFs from the last package.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node containing the active LFs to package.
name	`Optional[str]`	`None`	Name of the lf package.

Returns

LF package UID

Return type

`int`

snorkelflow.client.lf_packages.transfer_lf_packages

snorkelflow.client.lf_packages.transfer_lf_packages(*node, from_node, pkg_uids=None, updated_label_schema=None*)

Copy LF packages from one node to another.

Parameters

Name	Type	Default	Info
node	`int`		UID of the destination node.
from_node	`int`		UID of the source node.
pkg_uids	`Union[int, List[int], None]`	`None`	Optional list of package UIDs to copy. If None, all packages will be copied.
updated_label_schema	`Optional[Dict[str, str]]`	`None`	Dictionary that maps from label strings in the source node to corresponding label strings in the destination node if the label schema has been updated.

Return type

`None`

snorkelflow.client.lf_templates

LF template related functions.

Functions

add_lf_template_class(lf_template, ...[, ...])	Register an LF template class along with a LF template schema.
delete_lf_template(template_uid)	Delete LF templates
get_lf_templates([template_uid, ...])	Fetches Custom LF templates optionally filtered by template uid, template name, template type or template schema name

snorkelflow.client.lf_templates.add_lf_template_class

snorkelflow.client.lf_templates.add_lf_template_class(*lf_template*, *lf_template_schema*, *overwrite=False*)

Register an LF template class along with a LF template schema.

Examples

```
from templates import TemplateSchema, Template, TemplateConfig

class CustomNumericTemplateSchema(TemplateSchema):
    '''
    Field [field](int) is [operator] [value]
    '''
    field: str
    operator: str
    value: float

class CustomNumericTemplate(Template):
    template_type = "custom_numeric"
    abbreviation = "NUM"
    description = "If [field] has a numeric value that is [>, =, etc.]
[value], then label."
    menu_type = {
        "name": "Custom Numeric",
        "value": template_type,
        "category": ["custom"],
    }
    docs_link = ""
    template_schema = "CustomNumericTemplateSchema"

    def __init__(self, template_config: TemplateConfig) -> None:
        self._field = template_config["field"]
        self.op_str = template_config["operator"]
        self._value = template_config["value"]

    def check(self, x: pd.Series) -> bool:
        field_value = x[self._field]
        if self.op_str == "=":
            return field_value == self._value
        elif self.op_str == ">":
            return field_value > self._value
        elif self.op_str == "<":
            return field_value < self._value
        return False

sf.add_lf_template_class(CustomNumericTemplate,
CustomNumericTemplateSchema)
```

Parameters

Name	Type	Default	Info
lf_template	`Type[Any]`		Custom Template class that will be serialized.
lf_template_schema	`Type[Any]`		Template schema, derived from BaseModel to define the inputs custom LF template would take.
overwrite	`bool`	`False`	Overwrite an existing custom template class of the same name if one already exists Also overwrites LF template schema.

Returns

Metadata associated with this code asset

Return type

`Dict(str, any)`

Raises

- **UserInputError** – If the custom template or template schema class name does not begin with *Custom*
- **ValueError** – If the serialized Template or Template schema class is too large
- **ValueError** – If a Template class with the same name already exists
- **ValueError** – When workspace with workspace_name not found

snorkelflow.client.lf_templates.delete_lf_template

snorkelflow.client.lf_templates.delete_lf_template(*template_uid*)

Delete LF templates

Parameters

Name	Type	Default	Info
template_uid	`int`		Custom template uid.

Raises

- **NotFoundError** – When template with template_uid not found
- **UserInputError** – When an lf exists that is created using template with template_uid

Return type

`None`

snorkelflow.client.lf_templates.get_lf_templates

snorkelflow.client.lf_templates.get_lf_templates(*template_uid=None, template_name=None, template_type=None, template_schema_name=None, include_code=False*)

Fetches Custom LF templates optionally filtered by template uid, template name, template type or template schema name

Examples

```
>>> sf.get_lf_templates()
[
    {
        'template_uid': <template_uid>,
        'template_name': <template_name>,
        'template_type': <template_type>,
        'template_schema_name': <template_schema_name>,
        'code': <code>,
        'workspace_uid': <workspace_uid>
    }
    ...
]
```

Parameters

Name	Type	Default	Info
template_uid	`Optional[int]`	`None`	UID of the template. Defaults to None.
template_name	`Optional[str]`	`None`	Name of the template. Defaults to None.
template_type	`Optional[str]`	`None`	Template type. Defaults to None.
template_schema_name	`Optional[str]`	`None`	Template schema name. Defaults to None.
include_code	`bool`	`False`	Include template and template schema class code

			. Defaults to False.

Returns

List of custom LF templates info

Return type

`List[Dict[str, Any]]`

Raises

ValueError – When workspace with workspace_name not found

snorkelflow.client.lfs

LF related functions.

Functions

add_lf(node, lf[, overwrite])	Add an LF as an active LF.
add_scatterplot_lf(node, ax, label_str[, ...])	Create a bounding rectangle LF from the current view of a scatterplot.
archive_lf(node, name)	Archive an active LF.
archive_lfs(node)	Archive all active LFs in a node.
delete_lf(node, lf_uid)	Delete an LF This will completely delete lf (not archiving) Raises error if there is an LF package associated
execute_lfs(node_uid, pkg_uid, df)	Given a package_uid, node_uid and a df => retrieve the LFs from the package and run it over the data in the df
get_lf(node, name)	Get an active LF by name.
get_lf_autosuggest(node, column_name, ...)	Get LF suggestion from snorkel.
get_lf_summary(node[, split, ...])	Return various per-LF stats in a pandas DataFrame.
get_lfs(node)	Get all active LFs.
package_lfs(node[, name])	Package active LFs into an LF package.
unpackage_lfs(node, pkg_uid[, clear_existing])	Unpackage LFs from an LF package as active LFs.

snorkelflow.client.lfs.add_lf

snorkelflow.client.lfs.add_lf(*node, lf, overwrite=False*)

Add an LF as an active LF.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node containing the LF.
lf	`LF`		LF to add.
overwrite	`Optional[bool]`	`False`	If True, overwrite an exisiting active LF with the same name. Note that a new lf_uid will be assigned.

Returns

UID of the added LF

Return type

`int`

snorkelflow.client.lfs.add_scatterplot_lf

snorkelflow.client.lfs.add_scatterplot_lf(*node*, *ax*, *label_str*, *name=None*, *split='dev'*, *apply_timeout_secs=None*)

Create a bounding rectangle LF from the current view of a scatterplot.

The name of the x axis and y axis will be used as the field names.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
ax	`axis`		A matplotlib.pyplot.axis object containing information about the plot to create a bounding box from (usually acquired through plot_scatterplot()).
label_str	`str`		Label assigned if the labeling function votes.
name	`Optional[str]`	`None`	Name of the labeling function.
split	`str`	`'dev'`	Split to be loaded. Default to `"dev"`.
apply_timeout_secs	`Optional[float]`	`None`	Optional param to specify timeout for LF apply.

Raises

ValueError – If label_str is not in the set of valid labels for task

Return type

`Dict[str, Any]`

snorkelflow.client.lfs.archive_lf

snorkelflow.client.lfs.archive_lf(*node, name*)

Archive an active LF.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node containing the LF.
name	`str`		Name of the LF to delete.

Return type

`None`

snorkelflow.client.lfs.archive_lfs

snorkelflow.client.lfs.archive_lfs(*node*)

Archive all active LFs in a node.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node containing the LFs.

Return type

`None`

snorkelflow.client.lfs.delete_lf

snorkelflow.client.lfs.delete_lf(*node, lf_uid*)

Delete an LF This will completely delete lf (not archiving) Raises error if there is an LF package associated

Parameters

Name	Type	Default	Info
node	`int`		UID of the node containing the LF.
lf_uid	`int`		UID of the LF to delete.

Return type

`None`

snorkelflow.client.lfs.execute_lfs

snorkelflow.client.lfs.execute_lfs(*node_uid, pkg_uid, df*)

Given a package_uid, node_uid and a df => retrieve the LFs from the package and run it over the data in the df

Parameters

Name	Type	Default	Info
node_uid	`int`		UID of the node at which LFs are packaged.
pkg_uid	`int`		UID of the LF package whose LFs are applied to a dataframe.
df	`DataFrame`		Dataframe LFs are applied over.

Returns

Dataframe with an additional column containing LF vote for each LF

Return type

`pd.DataFrame`

snorkelflow.client.lfs.get_lf

snorkelflow.client.lfs.get_lf(*node, name*)

Get an active LF by name.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node containing the LF.
name	`str`		Name of the LF to get.

Returns

An active LF

Return type

LF

snorkelflow.client.lfs.get_lf_autosuggest

snorkelflow.client.lfs.get_lf_autosuggest(*node, column_name, column_type, suggest_model*)

Get LF suggestion from snorkel.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
column_name	`str`		Name of the column.
column_type	`str`		Type of the column.
suggest_model	`str`		Suggest model to use.

Returns

A List of suggested LF Configs in JSON format.

Return type

`list`

snorkelflow.client.lfs.get_lf_summary

snorkelflow.client.lfs.get_lf_summary(*node, split='dev', return_quality_scores=False, apply_timeout_secs=None*)

Return various per-LF stats in a pandas DataFrame.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
split	`str`	`'dev'`	The split to get LF stats from. Only *"dev"*, *"valid"*, and *"test"* splits are allowed. Default to *"dev"*.
return_quality_scores	`bool`	`False`	Whether to return LF quality scores for each LF. Default to `False`.
apply_timeout_secs	`Optional[float]`	`None`	Optional param to specify the timeout of LF apply per LF.

Returns

A Pandas DataFrame containing various per-LF stats.

Return type

`pd.DataFrame`

snorkelflow.client.lfs.get_lfs

snorkelflow.client.lfs.get_lfs(*node*)

Get all active LFs.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node to get LFs from.

Returns

List of LFs

Return type

List[LF]

snorkelflow.client.lfs.package_lfs

snorkelflow.client.lfs.package_lfs(*node, name=None*)

Package active LFs into an LF package. Use `unpackage_lfs()` to unpackage them. Note that new package will not be created if there is no change in the set of active LFs from the last package.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node containing the active LFs to package.
name	`Optional[str]`	`None`	Name of the lf package.

Returns

LF package UID

Return type

`int`

snorkelflow.client.lfs.unpackage_lfs

snorkelflow.client.lfs.unpackage_lfs(*node, pkg_uid, clear_existing=False*)

Unpackage LFs from an LF package as active LFs.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node at which LFs are unpackaged.
pkg_uid	`int`		UID of the LF package from which LFs are unpackaged.
clear_existing	`bool`	`False`	If True, all existing active LFs will be deleted.

Return type

`None`

snorkelflow.client.metrics

Metric related functions.

Functions

add_metric_to_node(node, metric_name)	Add a metric from the node's settings.
delete_metric_from_node(node, metric_name)	Remove a metric from the node's settings.
get_candidate_extractor_metrics(node[, ...])	Returns a metrics DataFrame for given prediction from candidate extractor against ground truth.
get_df_metrics(node, df, ground_truth_col, ...)	Returns a metrics DataFrame after scoring two columns from a DataFrame.
get_model_metrics(node[, model_uid, split, ...])	Return specified metrics for a model.
list_available_metrics(node)	List available metrics which can be passed to *get_model_metrics*.
register_custom_metric(node, metric_func, ...)	Register a user-defined metric to available metrics.
tune_threshold_on_valid(node, x_uids, ...[, ...])	Tune decision threshold on ground truth in valid split given predicted_labels and predicted_probs.

snorkelflow.client.metrics.add_metric_to_node

snorkelflow.client.metrics.add_metric_to_node(*node*, *metric_name*)

Add a metric from the node's settings.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
metric_name	`str`		Name of the metric.

Return type

`None`

snorkelflow.client.metrics.delete_metric_from_node

snorkelflow.client.metrics.delete_metric_from_node(*node, metric_name*)

Remove a metric from the node's settings.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
metric_name	`str`		Name of the metric.

Return type

`None`

snorkelflow.client.metrics.get_candidate_extractor_metrics

snorkelflow.client.metrics.get_candidate_extractor_metrics(*node, predictions=None, op_version_uid=None, split=None, metrics=None*)

Returns a metrics DataFrame for given prediction from candidate extractor against ground truth.

Parameters

Name	Type	Default	Info
node	`int`		UID of the extraction node.
predictions	`Optional[DataFrame]`	`None`	Predictions from candidate extractor. Should be a DataFrame, each row represents a span, with context_uid, char_start, char_end as columns. One of either predictions or op_version_id must be provided.
op_version_uid	`Optional[int]`	`None`	ID of the operator for which to calculate the metrics for One of either predictions or op_version_id must be provided.
split	`Optional[str]`	`None`	Name of the split to run metrics for. If None, then all splits are returned.
metrics	`Optional[List[str]]`	`None`	Metrics to be used for scoring. If None, defaults to word_entity_recall.

Returns

metric names as keys, metric scores as values

Return type

`Dict`

snorkelflow.client.metrics.get_df_metrics

snorkelflow.client.metrics.get_df_metrics(*node, df, ground_truth_col, prediction_col, metrics=None*)

Returns a metrics DataFrame after scoring two columns from a DataFrame.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
df	`DataFrame`		DataFrame with ground truth and predictions. Use **`get_node_data()`** to get it.
ground_truth_col	`str`		Name of the column containing ground_truth. Should be class integers.
prediction_col	`str`		Name of the column containing predictions. Should be class integers.
metrics	`Optional[List[str]]`	`None`	Metrics to be used for scoring. If None, defaults to the tasks settings metrics, see *get_node_settings*.

Returns

metric names as keys, metric scores as values

Return type

`Dict`

Examples

```
>>> sf.get_df_metrics(model_node_uid, df, ground_truth_col, pred_col)
{
    'metrics': {
        <metric_name>: <score>,
        ...
        'confusion_matrix' : {
            'matrix':[[<sum_of_datapoint_act_vs_pred>, ...]],
            'lablel': [<label>, ...].
        }
    }
}
```

snorkelflow.client.metrics.get_model_metrics

snorkelflow.client.metrics.get_model_metrics(*node, model_uid=None, split=None, start_date=None, end_date=None, include_tag_type_uid=None, exclude_tag_type_uid=None, metrics=None, ignore_cache=False, apply_postprocessors=True, is_context=False*)

Return specified metrics for a model.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
model_uid	`Optional[int]`	`None`	Model which metrics will be calculated for. If None, then all models are returned.
split	`Optional[str]`	`None`	Name of the split to run metrics for. If None, then all splits are returned.
start_date	`Optional[str]`	`None`	Earliest data source datestamp to include (YYYY-MM-DD format). If None, no filter set.
end_date	`Optional[str]`	`None`	Latest data source datestamp to include (YYYY-MM-DD format). If None, no filter set.
include_tag_type_uid	`Optional[List[int]]`	`None`	Only these tag types will be included in this metrics calculation. If None, no filter set.

exclude_tag_type_uid	`Optional[List[int]]`	`None`	Tag types which should be excluded from this metrics calculation. (Overrides inclusion on collison). If None, no filter set.
metrics	`Optional[List[str]]`	`None`	Metrics which will be returned as columns. If None, defaults to the tasks settings metrics, see *get_node_settings*.
ignore_cache	`bool`	`False`	If True, ignore the cached model metrics and re-compute the model metrics. Re-use cached model metrics by default.
apply_postprocessors	`bool`	`True`	If True, post processors are applied. Only relevant when is_context is False.
is_context	`bool`	`False`	If True, get context level metrics. Only relevant for information extraction tasks.

Returns

a DataFrame where each row represents a model

Return type

`pd.DataFrame`

snorkelflow.client.metrics.list_available_metrics

snorkelflow.client.metrics.list_available_metrics(*node*)

List available metrics which can be passed to *get_model_metrics*.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.

Returns

Names of metrics that are available.

Return type

`List[str]`

snorkelflow.client.metrics.register_custom_metric

snorkelflow.client.metrics.register_custom_metric(*node, metric_func, metric_name, overwrite=False, raw_code=None*)

Register a user-defined metric to available metrics.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
metric_func	`Callable`		A function to compute metrics.
metric_name	`str`		A name for this metric.
overwrite	`Optional[bool]`	`False`	Overwrite a metric of the same name if one already exists.
raw_code	`Optional[str]`	`None`	Custom metric func as a string.

Returns

id of the registered metric.

Return type

`int`

Examples

```
# Define a custom metric.
def jaccard_score(golds: np.ndarray, preds: np.ndarray, probs:
np.ndarray) -> float:
    # Params should be ``(golds, preds, probs)``, where each is an
np.ndarray:
    # ``golds`` represents the ground truth labels, 1-D array;
    # ``preds`` represents the model's predicted label, 1-D array; and,
    # ``probs`` represents the probabilities of each class, 2-D array,
n_examples x n_classes.
    # Any 3rd party libraries should be imported within the scoring
function.
    from sklearn import metrics
    return metrics.jaccard_score(golds, preds)

# This metric can now be registered.
import snorkelflow.client as sf
sf.register_custom_metric(
    node,
    metric_func=jaccard_score,
    metric_name="jaccard",
)

# Within the Models tab under Advanced Settings, "jaccard" can now be
added to the displayed
# metrics, or set as the node's primary metric.
# The metrics results can also be viewed in the SDK.
results = sf.get_model_metrics(node)
```

snorkelflow.client.metrics.tune_threshold_on_valid

snorkelflow.client.metrics.tune_threshold_on_valid(*node*, *x_uids*, *predicted_labels*, *predicted_probs*, *user_format=False*)

Tune decision threshold on ground truth in valid split given predicted_labels and predicted_probs. Only avalible for binary-class. To learn more about the Label format for your use case you can see Format for ground truth interaction in the SDK.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node for which we're adding predictions.
x_uids	`List[str]`		UIDs for examples for which predictions were made.
predicted_labels	`List[Any]`		Predicted labels corresponding to *x_uids*.
predicted_probs	`List[Any]`		Predicted label probabilities corresponding to *x_uids*.
user_format	`bool`	`False`	True if predictions are provided in user format, False otherwise.

Return type

`Tuple`[`Union`[`float`, `Dict`[`str`, `float`]], `ndarray`, `ndarray`, `ndarray`]

Returns

- *Union[float, Dict[str, float]]* – Optimal threshold tuned on ground truth in valid split

- *np.ndarray* – Numpy array of strings representing x uids, might be in a different order than the input x_uids
- *np.ndarray* – Numpy array of integers representing labels.
- *np.ndarray* – Numpy array representing probabilities.

snorkelflow.client.models

Model related functions.

Functions

add_predictions(node, *, model_uid, x_uids)	Register predictions generated by models (and optionally, ground truth labels).
get_models(node)	Return a list of models in pandas dataframe.
get_predictions(node[, split, model_uid, ...])	Get predictions from a trained model.
register_model(node, description, training_set)	Register a model with Snorkel Training Data Manager.
register_trained_model(model, node, description)	Register a trained model with Snorkel.
train_custom_model(node, custom_cls, ...[, ...])	Train a custom machine learning model.
train_model(node, feature_fields, ...[, ...])	Train a machine learning model.

snorkelflow.client.models.add_predictions

snorkelflow.client.models.add_predictions(*node, *, model_uid, x_uids, predicted_labels=None, predicted_probs=None, user_format=False*)

Register predictions generated by models (and optionally, ground truth labels). To learn more about the Label format for your use case you can see Format for ground truth interaction in the SDK.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node for which we're adding predictions.
model_uid	`int`		ID corresponding to the model that produced predictions.
x_uids	`Union[List[str], ndarray]`		UIDs of data points for which predictions were made.
predicted_labels	`Union[List[int], ndarray, None]`	`None`	Predicted labels corresponding to *x_uids*.
predicted_probs	`Union[List[List[float]], ndarray, None]`	`None`	Predicted label probabilities corresponding to *x_uids*. These probabilities should be a list of lists or 2-D numpy array of dimension *(len(x_uids)) x*

			(num_classes). All probabilities must be greater than zero and those for each data point should sum to 1.
user_format	`bool`	`False`	True if predictions and ground truth are provided in user format, False otherwise.

Return type

`None`

snorkelflow.client.models.get_models

snorkelflow.client.models.get_models(*node*)

Return a list of models in pandas dataframe.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node from which we're getting models.

Returns

List of models in a DataFrame

Return type

`pd.DataFrame`

snorkelflow.client.models.get_predictions

snorkelflow.client.models.get_predictions(*node, split=None, model_uid=None, start_date=None, end_date=None, is_context=False, user_format=False, apply_postprocessors=True*)

Get predictions from a trained model.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node whose predictions we're fetching.
split	`Optional[str]`	`None`	Name of a split whose predictions to fetch (if None, return all splits).
model_uid	`Optional[int]`	`None`	ID corresponding to the model whose predictions should be fetched. If None, use the most recently added model.
start_date	`Optional[str]`	`None`	Start date (e.g., yyyy/mm/dd) on the datasources to filter predictions.
end_date	`Optional[str]`	`None`	End date (e.g., yyyy/mm/dd) on the datasources to filter predictions.
is_context	`bool`	`False`	True to load predictions for context data points, False to load predictions for primary data points.
user_format	`bool`	`False`	True if the predictions and ground truths are returned in user format, False otherwise.

apply_postprocessors	`bool`	`True`	True if apply postprocessors.

Returns

A pandas DataFrame containing predictions and probabilities.

Return type

`pd.DataFrame`

snorkelflow.client.models.register_model

snorkelflow.client.models.register_model(*node, description, training_set, name=None*)

Register a model with Snorkel Training Data Manager.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node for which we're registering a model.
description	`str`		String containing description of model being registered.
training_set	`int`		Training set which the model was trained on. -1 if not trained using a training set created in Snorkel Flow.
name	`Optional[str]`	`None`	The name of the model.

Returns

Integer id of registered model.

Return type

`int`

snorkelflow.client.models.register_trained_model

snorkelflow.client.models.register_trained_model(*model, node, description*)

Register a trained model with Snorkel.

Parameters

Name	Type	Default	Info
model	`Any`		Custom model instance.
node	`int`		UID of the node for which we're registering a model.
description	`str`		Description of the model.

Returns

Integer id of registered trained model.

Return type

`int`

snorkelflow.client.models.train_custom_model

snorkelflow.client.models.train_custom_model(*node, custom_cls, feature_fields, model_description, name, training_set=None, filter_unlabeled=True, filter_uncertain_labels=True, load_ground_truth=True, train_on_dev_data=False, predict_on_dev_data=False, discretize_labels=True, max_runs=8, sampler_config=None, tune_threshold_on_valid=False, use_lf_labels=False, lf_uids_to_use=None, predict_for_all_train=False, apply_postprocessors=True, sync=True, scheduler=None*)

Train a custom machine learning model.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node for which we're training a model.
custom_cls	`Any`		Class of the custom defined model with train, predict, load, and save defined.
feature_fields	`List[str]`		List of field names to use as features.
model_description	`str`		Description of the model shown on the Models page.
name	`str`		The name of the model.
training_set	`Optional[int]`	`None`	Training set to train on. If None, defaults to latest.
filter_unlabeled	`bool`	`True`	Remove data points not covered by LFs when training?.

filter_uncertain_labels	`bool`	`True`	Remove low-confidence data points when training?.
load_ground_truth	`bool`	`True`	Use ground truth labels when training model?.
train_on_dev_data	`bool`	`False`	Include the dev set when training model?.
predict_on_dev_data	`bool`	`False`	Predict on dev set only?.
discretize_labels	`bool`	`True`	Round probabilistic training labels to maximum probability class?.
max_runs	`int`	`8`	For hyperparameter search, maximum number of model configurations to sample.
sampler_config	`Optional[Dict[str, Any]]`	`None`	A dictionary with fields “strategy” (required), “params” (optional), and “class_counts” (optional) representing a sampler configuration. For details, see sampler-config. This option is

			only supported if *discretize_labels* is True.
tune_threshold_on_valid	`bool`	`False`	For F1-tuned extraction models, tune the prediction threshold on valid set?.
use_lf_labels	`bool`	`False`	Use LF labels as features?.
lf_uids_to_use	`Optional[List[int]]`	`None`	If set, and use_lf_labels is True, use only these LFs for model training.
predict_for_all_train	`bool`	`False`	Predict for all data points in the train set (True), or just ones that can be loaded into the dev set (False)?.
apply_postprocessors	`bool`	`True`	If True, post processors are applied.
sync	`bool`	`True`	Poll job status and block until complete?.
scheduler	`Optional[str]`	`None`	Dask scheduler (threads, client, or group) to use.

Returns

A dictionary that looks like *{"job_id": "rq-XX-YYY", "model_uid": ZZ}*. The value for *model_uid* will be *None* if *sync=False*.

Return type

`Dict[str, Any]`

Examples

```
>>> from snorkelflow.models.cls_model import
TrainedClassificationModelV2
>>> class CustomModel(TrainedClassificationModelV2):
>>>     ...
>>>
>>> sf.train_custom_model(
>>>     node=node_uid,
>>>     custom_cls=CustomModel,
>>>     feature_fields=["feature_1", "feature_2"],
>>>     model_description="trainable model",
>>>     name="CustomModel",
>>>     training_set=training_set_uid,
>>> )
{'job_id': <job_id>, 'model_uid': <model_uid>}
```

snorkelflow.client.models.train_model

snorkelflow.client.models.train_model(*node, feature_fields, model_description, model_config, name=None, training_set=None, filter_unlabeled=True, filter_uncertain_labels=True, load_ground_truth=True, train_on_dev_data=False, predict_on_dev_data=False, discretize_labels=True, max_runs=8, sampler_config=None, tune_threshold_on_valid=False, use_lf_labels=False, lf_uids_to_use=None, predict_for_all_train=False, apply_postprocessors=True, sync=True, scheduler=None*)

Train a machine learning model.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node for which we're training a model.
feature_fields	`List[str]`		List of field names to use as features.
model_description	`str`		Description of the model shown on the Models page.
model_config	`Dict[str, Any]`		JSON configuration describing the model and hyperparameters.
name	`Optional[str]`	`None`	The name of the model.
training_set	`Optional[int]`	`None`	Training set to train on. If None, defaults to latest.
filter_unlabeled	`bool`	`True`	Remove data points not covered by LFs

			when training?.
filter_uncertain_labels	`bool`	`True`	Remove low-confidence data points when training?.
load_ground_truth	`bool`	`True`	Use ground truth labels when training model?.
train_on_dev_data	`bool`	`False`	Include the dev set when training model?.
predict_on_dev_data	`bool`	`False`	Predict on dev set only?.
discretize_labels	`bool`	`True`	Round probabilistic training labels to maximum probability class?.
max_runs	`int`	`8`	For hyperparameter search, maximum number of model configurations to sample.
sampler_config	`Optional[Dict[str, Any]]`	`None`	A dictionary with fields “strategy” (required), “params” (optional), and “class_counts” (optional) representing a sampler configuration. For details, see sampler-

			config. This option is only supported if *discretize_labels* is True.
tune_threshold_on_valid	`bool`	`False`	For F1-tuned extraction models, tune the prediction threshold on valid set?.
use_lf_labels	`bool`	`False`	Use LF labels as features?.
lf_uids_to_use	`Optional[List[int]]`	`None`	If set, and use_lf_labels is True, use only these LFs for model training.
predict_for_all_train	`bool`	`False`	Predict for all data points in the train set (True), or just ones that can be loaded into the dev set (False)?.
apply_postprocessors	`bool`	`True`	If True, post processors are applied.
sync	`bool`	`True`	Poll job status and block until complete?.
scheduler	`Optional[str]`	`None`	Dask scheduler (threads, client, or group) to use.

Returns

A dictionary that looks like *{"job_id": "rq-XX-YYY", "model_uid": ZZ}*. The value for *model_uid* will be *None* if *sync=False*.

Return type

`Dict[str, Any]`

Examples

```
>>> from snorkelflow.models.model_configs import
SKLEARN_LOGISTIC_REGRESSION_CONFIG
>>> import snorkelflow.client as sf
>>> sf.train_model(
>>>     node=node,
>>>     feature_fields=["text"],
>>>     model_description='Logistic Regression',
>>>     model_config=SKLEARN_LOGISTIC_REGRESSION_CONFIG.dict()
>>> )
{'job_id': <job_id>, 'model_uid': <model_uid>}
```

snorkelflow.client.nodes

Node related functions.

Functions

add_node(application[, input_node_uids, ...])	Adds a node to the graph.
add_node_hierarchy(application, ...)	Initiates a hierarchical classification DAG from an initial application.
commit_builtin_operator(node, op_type[, ...])	Commit a built-in operator to the node
commit_custom_operator(node, operator_uid)	Commits a customer operator to the specified node
commit_custom_operator_to_node(node, ...)	Commits a customer operator to the specified node
commit_model_to_node(node, model_uid)	Commits an existing model to the specified node
delete_node(node)	Deletes the specified node
fit_and_commit(node_uid, op_type, fit_config)	Fits an operator given a node_uid and commits a fitted op_version to the node.
get_dataset(node[, split, batch_uid, ...])	Filter the dataset and return with GT labels and uids as index.
get_example(node, x_uid[, split])	Get a specific data point by x_uid.
get_model_node(application)	Returns the model node for an application, if there is exactly one
get_model_nodes(application)	Returns all model nodes in the application

get_node(node)	Gets detailed information about a specific node in a data pipeline (DAG).
get_node_data(node[, split, data, ...])	Get a dataframe from a model node, optionally including annotations and labels.
get_node_input_cols(application, node)	Returns a list of available columns at the specified node.
get_node_inputs_data(application, node[, ...])	Gets the input data frames at a specific node.
get_node_label_map(node_uid)	Returns the label map for a given model node.
get_node_output_data(application, node[, ...])	Get the output dataframe at a specific node.
get_node_settings(node, setting)	Get specified node setting.
get_node_uid(application, search_op_type)	Returns all nodes in the application with a specified expected_op_type
get_preprocessing_issues(node)	Returns all datapoint UIDs with errors or warnings raised during preprocessing up to (but excluding) the node uid.
list_nodes(application)	Gets a list of all nodes in the specified application.
resample_split(node[, split, max_total, ...])	Resample a split of the dataset.
set_node_settings(node, setting, setting_value)	Set specified task setting.
uncommit_operator(node)	Uncommit a committed operator from the node

snorkelflow.client.nodes.add_node

snorkelflow.client.nodes.add_node(*application, input_node_uids=None, expected_op_type=None, node_config=None, output_node_uid=None, output_node_uids=None, node_cls='ApplicationNode', op_type=None, op_config=None, add_to_parent_block=False*)

Adds a node to the graph. Must specify one of input_node_uids or output_node_uids to determine the location of the node.

Parameters

Name	Type	Default	
application	`Union[str, int]`		The name application
input_node_uids	`Optional[List[int]]`	`None`	The list of i [-1] if your dataset no
expected_op_type	`Optional[str]`	`None`	The type o
node_config	`Optional[Dict[str, Any]]`	`None`	A dictionar node.
output_node_uids	`Optional[List[int]]`	`None`	An optiona node ids.
output_node_uid	`Optional[int]`	`None`	Deprecate output noc
node_cls	`str`	`'ApplicationNode'`	The name
op_type	`Optional[str]`	`None`	The operat committed *TruncateP*
op_config	`Optional[Dict[str, Any]]`	`None`	The operat *{"field": "te "length": 5*

			`get_defa` to check re built-in ope
add_to_parent_block	`Optional[bool]`	`False`	The boolea determine added to tl Note: in an single exist *input_nod* this param result in a ahead of th within it.

Returns

Information about the created node object

Return type

`Dict[str, Any]`

Examples

```
>>> # Add a place holder node
>>> sf.add_node(
>>>     application=APP_NAME,
>>>     input_node_uids=[input_node],
>>>     expected_op_type="Featurizer",
>>>     output_node_uids=[output_node],
>>> )
{'node_uid': <node_uid>, 'op_version_uid': None}

>>> # Add a committed node
>>> sf.add_node(
>>>     application=APP_NAME,
>>>     input_node_uids=[input_node],
>>>     op_type="TruncatePreprocessor",
>>>     op_config={"field": "text", "by": "chars", "length": 5},
>>>     output_node_uids=[output_node],
>>> )
{'node_uid': <node_uid>, 'op_version_uid': <op_version_uid>}
```

snorkelflow.client.nodes.add_node_hierarchy

snorkelflow.client.nodes.add_node_hierarchy(*application*, *input_node_uid*, *hierarchy_config*)

Initiates a hierarchical classification DAG from an initial application.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.
input_node_uid	`int`		The input node from where hierarchy will begin.
hierarchy_config	`Dict[str, Dict]`		A dictionary config for the labels in the hierarchy.

Returns

Dict[str – Dictionary detailing nodes created in the DAG. Semantics of the output are as follows - 1. Initial_Model -> uid of the model node of the initial application 2. Label str -> Dictionary which details nodes created for the branch a. Label_Filter: uid of the LabelFilter node which segregates datapoints as per a particular label for downstream classification, b. Column_Filler: uid of the ColumnFiller node which replaces existing gts with UNK to allow new labels, c. Model: uid of the Model node which classifies these datapoints into child labels 3. Concat -> uid of the Concat node which consolidates all labels at the end

Return type

`Any]`

Examples

```
>>> sf.add_node_hierarchy(
>>>     APP_NAME,
>>>     model_node_uid,
>>>     {
>>>         "1": {"1.1": {}, "1.2": {}},
>>>         "2": {"2.1": {}, "2.2": {}},
>>>     }
>>> )
{
    'nodes_output_dict':
    {'Initial_Model': {'Model': 525},
    '1': {'Label_Filter': 531, 'Column_Filler': 532, 'Model': 533},
    '2': {'Label_Filter': 536, 'Column_Filler': 537, 'Model': 538},
    '1.1': {'Label_Filter': 534},
    '1.2': {'Label_Filter': 535},
    '2.1': {'Label_Filter': 539},
    '2.2': {'Label_Filter': 540},
    'Concat': {'node_uid': 541, 'op_version_uid': 345}}
}
```

snorkelflow.client.nodes.commit_builtin_operator

snorkelflow.client.nodes.commit_builtin_operator(*node, op_type, op_config=None*)

Commit a built-in operator to the node

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to have the operator committed to.
op_type	`str`		The operator type to be committed, e.g., *TruncatePreprocessor*.
op_config	`Optional[Dict[str, Any]]`	`None`	The operator config, e.g., *{"field": "text", "by": "chars", "length": 5}*. Use **`get_default_operator()`** to check required fields for built-in operators.

Return type

`None`

Examples

```
>>> new_node_id = sf.add_node(APP_NAME, input_node_uids=[123],
output_node_uids=[456])['node_uid']
>>> sf.commit_builtin_operator(
>>>     new_node_id,
>>>     op_type="TruncatePreprocessor",
>>>     op_config={"field": "text", "by": "chars", "length": 5},
>>> )
```

snorkelflow.client.nodes.commit_custom_operator

snorkelflow.client.nodes.commit_custom_operator(*node, operator_uid*)

Commits a customer operator to the specified node

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to have the op-version committed to.
operator_uid	`int`		UID of an existing custom operator, to be turned into an op-version and committed to the node.

Returns

Dictionary containing the op_version_uid of the newly created op-version

Return type

`Dict[str, Any]`

Examples

```
>>> # Add a custom operator
>>> operator_uid =  sf.add_operator(operator)["op_uid"]
>>> # Add a placeholder node
>>> add_node_resp = sf.add_node(
>>>     application=APP_NAME,
>>>     input_node_uids=[INPUT_NODE_ID],
>>>     expected_op_type="Featurizer",
>>>     output_node_uids=[model_node_uid],
>>> )
>>> sf.commit_custom_operator_to_node(
>>>     node=add_node_resp['node_uid'],
>>>     operator_uid=operator_uid,
>>> )
{'op_version_uid': <op_version_uid> }
```

snorkelflow.client.nodes.commit_custom_operator_to_node

snorkelflow.client.nodes.commit_custom_operator_to_node(*node, operator_uid*)

Commits a customer operator to the specified node

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to have the op-version committed to.
operator_uid	`int`		UID of an existing custom operator, to be turned into an op-version and committed to the node.

Returns

Dictionary containing the op_version_uid of the newly created op-version

Return type

`Dict[str, Any]`

Examples

```
>>> # Add a custom operator
>>> operator_uid =  sf.add_operator(operator)["op_uid"]
>>> # Add a placeholder node
>>> add_node_resp = sf.add_node(
>>>     application=APP_NAME,
>>>     input_node_uids=[INPUT_NODE_ID],
>>>     expected_op_type="Featurizer",
>>>     output_node_uids=[model_node_uid],
>>> )
>>> sf.commit_custom_operator_to_node(
>>>     node=add_node_resp['node_uid'],
>>>     operator_uid=operator_uid,
>>> )
{'op_version_uid': <op_version_uid> }
```

snorkelflow.client.nodes.commit_model_to_node

snorkelflow.client.nodes.commit_model_to_node(*node, model_uid*)

Commits an existing model to the specified node

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to have the op-version committed to.
model_uid	`int`		UID of an existing model that belongs to this node. This model_uid will be turned into an op-version and committed to the node.

Returns

Dictionary containing the op_version_uid of the newly created op-version

Return type

`Dict[str, Any]`

Examples

```
>>> sf.commit_model_to_node(model_node_uid, model_uid)
{'op_version_uid': <op_version_uid>}
```

snorkelflow.client.nodes.delete_node

snorkelflow.client.nodes.delete_node(*node*)

Deletes the specified node

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to delete.

Return type

None

snorkelflow.client.nodes.fit_and_commit

snorkelflow.client.nodes.fit_and_commit(*node_uid, op_type, fit_config, op_version_uid=None, operator_uid=None*)

Fits an operator given a node_uid and commits a fitted op_version to the node.

Parameters

Name	Type	Default	Info
node_uid	`int`		uid of working node that should contain operator.
op_type	`str`		class string of fited operator (e.g. StandardScaler).
fit_config	`Dict[str, Any]`		dictionary with fit arguments for fitting the operator.
op_version_uid	`Optional[int]`	`None`	UID of an existing op-version to be committed to the node.
operator_uid	`Optional[int]`	`None`	UID of an existing custom operator, to be turned into an op-version and committed to the node.

Returns

Dictionary containing the op_version_uid of the newly created op-version

Return type

`Dict[str, Any]`

Examples

```
>>> # Add placeholder node
>>> add_node_resp = sf.add_node(
>>>     application=APP_NAME,
>>>     input_node_uids=[INPUT_NODE_ID],
>>>     expected_op_type="Featurizer",
>>>     output_node_uids=[model_node_uid],
>>> )
>>> sf.fit_and_commit(
>>>     node_uid=add_node_resp['node_uid'],
>>>     op_type="StandardScaler",
>>>     fit_config={
>>>         "field": "field_name",
>>>         "target_field": "target_field_name",
>>>     },
>>> )
{'op_version_uid': <op_version_uid> }
```

snorkelflow.client.nodes.get_dataset

snorkelflow.client.nodes.get_dataset(*node, split='dev', batch_uid=None, combiner='AND', show_filtered_flag=False, gt_label=None, all_lfs_filter=None, no_lfs_filter=None, model_filters=None, lf_filters=None, training_set_filters=None, include_tag_type_uids=None, exclude_tag_type_uids=None, exclude_slice_uids=None, include_slice_uids=None*)

Filter the dataset and return with GT labels and uids as index.

You can also filter for only data points that have a certain ground truth label or a certain predicted label by a machine learning model. This filter is handy when you want to do your own analysis on an error bucket.

Multiple conditions will be combined with `AND` semantics.

Examples

```
# Filter data where all LFs abstain.
df = sf.get_dataset(node, all_lfs_filter="UNKNOWN")
# Filter data where their GT labels are "LABEL".
df = sf.get_dataset(node, gt_label="LABEL")
# Filter data with a set of rules.
df = sf.get_dataset(
    node,
    gt_label="LABEL",
    model_filters=[(1, "LABEL"), (2, "LABEL")],
    lf_filters=[("my_lf", "LABEL")],
    training_set_filters=[(1, "LABEL")],
    combiner="AND"
)
```

```
# Generate column statistics and summaries for the dataset
df = sf.get_dataset(node)
df.describe(include="all")
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
split	`str`	`'dev'`	The split to be loaded. Only

			`"dev"`, `"valid"`, and `"test"` splits are allowed. Default to `"dev"`.
batch_uid	`Optional[int]`	`None`	UID of the annotation batch to filter.
combiner	`str`	`'AND'`	Combiner to apply across all filters, by default `"AND"`
show_filtered_flag	`bool`	`False`	If True, return all datapoints and a column called “filtered_flag” with “True” or “False” values specifying whether the data is included in the provided filters.
gt_label	`Optional[str]`	`None`	If set, include only data points where ground truth is this label string.
all_lfs_filter	`Optional[str]`	`None`	If set, include only data points where all LFs

			vote for the label string passed.
no_lfs_filter	`Optional[str]`	`None`	If set, include only data points where no LFs vote for the label string passed.
model_filters	`Optional[List[Tuple[int, str]]]`	`None`	Tuple where the first value is the model_id, and the second is predicted label or voting pattern. If provided, include only data points that match this pattern.
lf_filters	`Optional[List[Tuple[str, str]]]`	`None`	List of tuples where first value in tuple is LF name and second value is assigned label or voting pattern. If provided, include only data points that match this pattern.

training_set_filters	`Optional[List[Tuple[int, str]]]`	`None`	List of tuples where first value in tuple is training set ID and second value is assigned label or voting pattern. If provided, include only data points that match this pattern.
include_slice_uids	`Optional[List[int]]`	`None`	List of slices, only data points which are a member of this slice will be included.
include_tag_type_uids	`Optional[List[int]]`	`None`	List of tag types, only data points which include this tag type will be included.

exclude_slice_uids	`Optional[List[int]]`	`None`	List of slices, only data points which are not a member of this slice will be included. Exclusion will supersede inclusion in case of collision.
exclude_tag_type_uids	`Optional[List[int]]`	`None`	List of tag types, only data points which exclude this tag type will be included. Exclusion will supersede inclusion in case of collision.

Returns

A Pandas DataFrame for give split filtered according to parameters

Return type

`pd.DataFrame`

snorkelflow.client.nodes.get_example

snorkelflow.client.nodes.get_example(*node, x_uid, split='dev'*)

Get a specific data point by x_uid.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
x_uid	`str`		UID of the data point you want to get.
split	`str`	`'dev'`	Split where that data point belongs.

Returns

Dictionary of properties of that data point.

Return type

`dict`

snorkelflow.client.nodes.get_model_node

snorkelflow.client.nodes.get_model_node(*application*)

Returns the model node for an application, if there is exactly one

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.

Returns

The UID of the model node

Return type

`int`

snorkelflow.client.nodes.get_model_nodes

snorkelflow.client.nodes.get_model_nodes(*application*)

Returns all model nodes in the application

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.

Returns

A list of model node ids

Return type

`List[int]`

snorkelflow.client.nodes.get_node

snorkelflow.client.nodes.get_node(*node*)

Gets detailed information about a specific node in a data pipeline (DAG).

Gets comprehensive details about a node, including its class, unique identifier, associated dataset and application, committed operator version, expected operator type, creation timestamp, and node-specific configuration and label schema.

Parameters

Name	Type	Default	Info
node	`int`		The unique identifier (UID) of the node. This is an integer representing a specific node within the Snorkel Flow application graph.

Raises

HTTPError – If the node with the given UID is not found (404 status code) or if there are other API-related issues. Check the error message in the exception for details.

Return type

`Dict[str, Any]`

Example

Example 1

Get information about the node with UID 123.

```
# Get the node with UID 123
sf.get_node(123)
```

Example 1 return

```
{
    'node_cls': 'ClassificationNode',
    'node_uid': 123,
    'dataset_uid': 456,
    'application_uid': 789,
    'committed_op_version_uid': 101,
    'expected_op_type': 'Model',
    'created_at': '2025-02-06T17:11:31.511326',
    'node_config': {
        'safe_to_delete': False,
        'label_map': {'positive': 0, 'negative': 1, 'neutral': 2},
        . . .
    },
    'label_schema': {
        'name': 'Sentiment',
        'description': 'Sentiment of product review',
        'data_type': 'text',
        'task_type': 'classification',
        . . .
    }
}
```

This output includes the following values:

- `node_cls`: The class of the node, such as 'ModelNode' or 'ClassificationNode'.
- `node_uid`: The unique identifier of the node.
- `dataset_uid`: The unique identifier of the dataset associated with the node (may be null if not associated).
- `application_uid`: The unique identifier of the application the node belongs to.
- `committed_op_version_uid`: The unique identifier of the committed operator version (may be null if no operator is committed).
- `expected_op_type`: The expected type of operator for this node (e.g., 'Featurizer', 'Model').
- `created_at`: Timestamp indicating when the node was created.
- `node_config`: A dictionary containing node-specific configuration. The structure of this config varies based on the node class (`node_cls`) and `expected_op_type`.
- `label_schema`: A dictionary describing the label schema associated with this node.

snorkelflow.client.nodes.get_node_data

snorkelflow.client.nodes.get_node_data(*node, split=None, data=True, data_columns=None, ground_truth=True, tags=False, comments=False, training_set_labels=False, training_set_probs=False, training_set_uid=None, training_set_overwrite_with_gt=False, training_set_filter_unlabeled=False, training_set_filter_uncertain_labels=True, training_set_tie_break_policy='abstain', training_set_sampler_config=None, model_predictions=False, model_probabilities=False, model_confidences=False, model_uid=None, user_format=False, rename_columns=None, lfs=None, lfs_column_prefix='lf_', start_date=None, end_date=None, max_input_rows=None, apply_postprocessors=True, slices=False*)

Get a dataframe from a model node, optionally including annotations and labels.

Parameters

Name	Type	Default	
node	`int`		UID of t a dataf
split	`Union[str, List[str], None]`	`None`	Name c one or “dev”, “ “train” v in the c None, l
data	`bool`	`True`	If True, column in the r
data_columns	`Optional[List[str]]`	`None`	Option needec Default Specify data_cc signific
ground_truth	`bool`	`True`	If True, contain

			labels (
tags	`bool`	`False`	If True, contain to each np.nan
comments	`bool`	`False`	If True, contain assigne (missin
training_set_uid	`Optional[int]`	`None`	The uid whose probab training training this is r
training_set_labels	`bool`	`False`	If True, contain labels (
training_set_probs	`bool`	`False`	If True, contain probab np.nan
training_set_overwrite_with_gt	`bool`	`False`	If True, labels/p ground possibl

training_set_filter_unlabeled	`bool`	`False`	If True, where t -1 (i.e., training is True, or GT la
training_set_filter_uncertain_labels	`bool`	`True`	If True, "abstai labels c thresho
training_set_tie_break_policy	`str`	`'abstain'`	How to set labe set pro a rando tied cla Return label.
training_set_sampler_config	`Optional[Dict[str, Any]]`	`None`	A dictio "strateg "param "class_c represe configu sample only pe labels,
model_predictions	`bool`	`False`	If True, contain predict
model_probabilities	`bool`	`False`	If True, contain probab cardina

			could b you ma fetchin and/or instead
model_confidences	`bool`	`False`	If True, contain confide predict is defin assigne class.
model_uid	`Optional[int]`	`None`	The uid predict and/or loaded or mod this is r
user_format	`bool`	`False`	True if provide otherw
rename_columns	`Optional[Dict[str, str]]`	`None`	Option columr are “gr “comm “trainin “trainin “model “model “model
lfs	`Optional[List[str]]`	`None`	String r functio would l final ou

			not par their la (e.g., -1 user_fo label) o “dev”.
lfs_column_prefix	`Optional[str]`	`'lf_'`	String p columr columr resultir names “lf_{labe
start_date	`Optional[str]`	`None`	Include added YYYY-M
end_date	`Optional[str]`	`None`	Include added YYYY-M
max_input_rows	`Optional[int]`	`None`	Contro returne dataset
apply_postprocessors	`bool`	`True`	True if
slices	`bool`	`False`	If True, contain to each np.nan

Returns

An [n_data_points, n_fields] pd.DataFrame containing the task data.

Return type

`DataFrame`

Raises

- **RuntimeError** – If the data split does not exist.
- **ValueError** – If an unrecognized column name is passed in through rename_columns
- **ValueError** – If training-related arguments are set to True and no training_set_uid is specified.
- **ValueError** – If model-related arguments are set to True and no model_uid is specified.
- **ValueError** – If the given node UID doesn't point to a model node.

snorkelflow.client.nodes.get_node_input_cols

snorkelflow.client.nodes.get_node_input_cols(*application*, *node*)

Returns a list of available columns at the specified node.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.
node	`int`		The UID of the node to be committed.

Returns

A list of column names

Return type

`List[str]`

snorkelflow.client.nodes.get_node_inputs_data

snorkelflow.client.nodes.get_node_inputs_data(*application, node, datasource_uids=None, max_input_rows=5, partition=None*)

Gets the input data frames at a specific node. By default, only 5 rows will be used. See `get_node_output_data()` for more details and example usages of the parameters.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.
node	`int`		The UID of the node to retrieve data from.
datasource_uids	`Optional[List[int]]`	`None`	Optional list of datasource uids from the dataset to run the pipeline on. Cached datasource identifiers should not be used.
max_input_rows	`Optional[int]`	`5`	Optional maximum number of rows to compute each input node on. Should be a positive integer. Does not guarantee that exactly max_input_rows rows will be used. Set None to retrieve all rows (potentially cause out of memory).
partition	`Optional[int]`	`None`	0-based index to specify the (datasource) partition to read from. Leave it *None* to read all partitions.

Returns

A mapping from input node id to Pandas DataFrame

Return type

`Dict[int, pd.DataFrame]`

Raises

IndexError – If the provided partition is out of range

snorkelflow.client.nodes.get_node_label_map

snorkelflow.client.nodes.get_node_label_map(*node_uid*)

Returns the label map for a given model node. The label map is a mapping from label strings to their integer representations.

Parameters

Name	Type	Default	Info
node_uid	`int`		The UID corresponding to a Model node.

Returns

The label map, mapping each label string to its corresponding integer value

Return type

`Dict[str, int]`

snorkelflow.client.nodes.get_node_output_data

snorkelflow.client.nodes.get_node_output_data(*application, node, datasource_uids=None, max_input_rows=5, partition=None*)

Get the output dataframe at a specific node.

To get this dataframe, the application dataset is processed through the DAG up until the specified node. By default, only 5 rows will be read from the dataset. You can also specify particular datasources by their uids and/or the 0-based partition number. For example, if you have a dataset with 3 datasources, each of which is partitioned by 2, *partition=3* refers to the first partition of the second datasource.

There is another method to retrieve the output data at multiple specified nodes: `sf.execute_graph_on_data`, which allows passing a customized input DataFrame to retrieve its output at specific nodes.

Examples

```
# Get the output of the node 123 using the first 5 rows in the dataset
sf.get_node_output_data("my-application", 123)

# Get the output of the node 123 using all the rows in the dataset
sf.get_node_output_data("my-application", 123, max_input_rows=None)

# Get the output of the node 123 using a particular datasource
sf.get_node_output_data("my-application", 123, datasource_uids=[456])

# Get the output of the node 123 using the first partition (i.e., the
first partition of the first datasource)
sf.get_node_output_data("my-application", 123, max_input_rows=None,
partition=0)
```

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.
node	`int`		The UID of the node to retrieve data from.

datasource_uids	`Optional[List[int]]`	`None`	Optional list of datasource identifiers to run the pipeline on.
max_input_rows	`Optional[int]`	`5`	Optional Maximum number of input rows to compute node output on. Should be a positive integer. Does not guarantee that exactly max_input_rows rows will be used. Set None to retrieve all rows (potentially cause out of memory).
partition	`Optional[int]`	`None`	0-based index to specify the (datasource) partition to read from. Leave it *None* to read all partitions.

Returns

A Pandas DataFrame that contains all data at the specified node

Return type

`pd.DataFrame`

Raises

IndexError – If the provided partition is out of range

snorkelflow.client.nodes.get_node_settings

snorkelflow.client.nodes.get_node_settings(*node, setting*)

Get specified node setting.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node you are getting settings for.
setting	`str`		Name of setting to get.

Returns

Dictionary with specified setting

Return type

`Dict[str, Any]`

Examples

```
>>> settings = sf.get_node(model_node_uid)['node_config']['settings']
>>> valid_settings = list(settings.keys())
>>> setting_field = valid_settings[0]
>>> sf.get_node_settings(model_node_uid, setting_field)
{
    'node_uid': <node_uid>,
    '<setting_field>': <setting_field_value>,
}
```

snorkelflow.client.nodes.get_node_uid

snorkelflow.client.nodes.get_node_uid(*application, search_op_type*)

Returns all nodes in the application with a specified expected_op_type

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application.
search_op_type	`Optional[str]`		The search_op_type or expected_op_type of nodes to return. Set as *None* to get nodes with no expected op type set. Algorithm: If search_op_type is None, return all nodes with expected_op_type == None Elif node_op_type == node.op_type or node_op_type = node.expected_op_type add nodes to list of returned nodes.

Returns

A list of model node ids

Return type

`List[int]`

snorkelflow.client.nodes.get_preprocessing_issues

snorkelflow.client.nodes.get_preprocessing_issues(*node*)

Returns all datapoint UIDs with errors or warnings raised during preprocessing up to (but excluding) the node uid. Datapoints are checked by certain operators for errors or warnings during preprocessing.

Parameters

Name	Type	Default	Info
node	`int`		The UID of the model node.

Returns

"datapoint": UID of the skipped datapoint "type: The type of issue (error or warning) encountered "node": The node at which the datapoint failed the check "detail": Error message detail encountered when the datapoint failed the check

Return type

`DataFrame with columns –`

snorkelflow.client.nodes.list_nodes

snorkelflow.client.nodes.list_nodes(*application*)

Gets a list of all nodes in the specified application.

This function gets a list of node objects, each representing a node in a data pipeline (DAG).

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		Specify the application, by name or UID, to see its list of nodes.

Raises

ValueError – If the application is invalid or does not exist.

Return type

`List[Dict[str, Any]]`

Example

Example 1

Get a list of all nodes in the application 'YOUR_APP_NAME'.

```
# Get all nodes in the application 'YOUR_APP_NAME'
sf.list_nodes('YOUR_APP_NAME')
```

Example 1 return

```
[
    {
        'node_uid': 123,
        'committed_op_version_uid': 456,
        'expected_op_type': 'Model',
        'created_at': '2025-02-06T17:11:31.511326',
        'node_config': {
            'safe_to_delete': False,
            'label_map': {'positive': 0, 'negative': 1, 'neutral': 2},
            . . .
        },
        'label_schema': {
            'name': 'Sentiment',
            'description': 'Sentiment of product review',
            'data_type': 'text',
            'task_type': 'classification',
            . . .
        }
    }
]
```

This output includes the following values:

- `node_uid`: Unique identifier for the node.
- `committed_op_version_uid`: Unique identifier for the committed operator version, if any.
- `expected_op_type`: The expected operator type for this node (e.g., 'Featurizer', 'Model').
- `created_at`: Timestamp indicating when the node was created.
- `node_config`: A dictionary containing node-specific configuration. The structure of this config varies based on the node class (`node_cls`) and `expected_op_type`.
- `label_schema`: A dictionary describing the label schema associated with this node.

snorkelflow.client.nodes.resample_split

snorkelflow.client.nodes.resample_split(*node, split='dev', max_total=None, max_labeled=None, min_per_class=1, sample_by_docs=False, seed=123, split_datasources=None, x_uids=None, context_uids=None*)

Resample a split of the dataset.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
split	`str`	`'dev'`	Split to resample.
max_total	`Optional[int]`	`None`	The maximum total number of examples in the resampled split. This will be approximate for extraction tasks where we fetch examples by whole context only.
max_labeled	`Optional[int]`	`None`	The maximum number of labeled examples to include in sampled split.
min_per_class	`Optional[int]`	`1`	The minimum number of labeled examples per class to include in sampled split.
sample_by_docs	`bool`	`False`	If True, then the numbers (max_total, max_labeled) are interpreted as number of docs (rather than number of candidates).
seed	`Optional[int]`	`123`	A random seed to make sampling deterministic for

			a given dataset.
split_datasources	`Optional[List[int]]`	`None`	A list of the datasources that we want to resample from. If None, sample from all datasources.
x_uids	`Optional[List[str]]`	`None`	A list of x_uids. If provided, overrides all other keyword arguments and pulls in the provided x_uids instead. This can only be used for sampling datapoints into the dev split.
context_uids	`Optional[List[int]]`	`None`	A list of document-level int uids to sample. If provided, overrides all other keyword arguments and pulls in the x_uids corresponding to the document-level uids instead. This can only be used for sampling datapoints into the dev split.

Raises

ValueError – If split is not in the set of valid splits.

Return type

`None`

Examples

```
# Resample the valid split
sf.resample_split(node, split="valid", seed=123)

# Sample specific datapoints into the dev split using the index
sf.resample_split(node, x_uids=["span::5", "span::7"])

# Sample specific documents into the dev split using the input index
sf.resample_split(node, context_uids=[1, 2])
```

snorkelflow.client.nodes.set_node_settings

snorkelflow.client.nodes.set_node_settings(*node, setting, setting_value*)

Set specified task setting.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node to change settings for.
setting	`str`		Name of setting to change.
setting_value	`Any`		New value for setting.

Return type

`None`

snorkelflow.client.nodes.uncommit_operator

snorkelflow.client.nodes.uncommit_operator(*node*)

Uncommit a committed operator from the node

If the specified node has a committed operator, this method will uncommit it, leaving the node in an uncommitted state.

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node to have the op-version committed to.

Return type

`None`

Examples

```
>>> sf.uncommit_operator(node_uid)
```

snorkelflow.client.object_storage.download_remote_object

snorkelflow.client.object_storage.download_remote_object(*url, bucket, obj, access_key, secret_key, region='', sync=True*)

Download a file from an external s3-compatbile storage (e.g: MinIO, GCS).

This can be useful for downloading a datasource, ground truth, or any other file for usage in-platform.

Parameters

Name	Type	Default	Info
url	`str`		URL of the remote storage server (e.g. https://my-minio-server.domain.com), must have URI scheme (http:// or https://).
bucket	`str`		Bucket name for the object.
obj	`str`		Path of the object inside the bucket (e.g. enron_spam_test.csv).
access_key	`str`		Access key for the remote storage server.
secret_key	`str`		Secret key for the remote storage server.
region	`str`	`''`	Region for the s3 service (ignored for MinIO & GCS).
sync	`bool`	`True`	Poll job status and block until complete?.

Returns

The in-platform path of the object downloaded if sync mode used

Return type

`str`

Examples

```
# Downloading from minio
path = sf.download_remote_object(
    "https://my-minio.domain.com/",  # remote host
    "training-data-bucket",          # bucket
    "dev-split.csv",                 # object
    "admin",                         # access key
    "password")                      # secret key

# Downloading from GCS bucket. Please use HMAC key authentication
# https://cloud.google.com/storage/docs/authentication/hmackeys
gcs_path = sf.download_remote_object("https://storage.googleapis.com",
    "gcs-bucket-name",
    "gcs-object.csv",
    "GCS_HMAC_KEY",
    "GCS_HMAC_SECRET")

# Downloading from s3
s3_path = sf.download_remote_object(
    "https://s3.amazonaws.com/",
    "some-s3-bucket",
    "dev-split.csv",
    "AWS_ACCESS_KEY",
    "AWS_SECRET_KEY",
    "us-west-2")         # region is required for s3
```

snorkelflow.client.operators

Operator related functions.

Functions

add_operator(operator[, overwrite])	Add an operator (defined as `UserDefinedOperator`).
add_operator_class(operator[, overwrite])	Add an operator class.
check_conflicting_operator_name(operator_name)	Check whether an Operator name conflicts with the name of a built-in Operator.
delete_operator(operator_name)	Delete an operator given the operator name.
execute_operators(df[, processor_config, ...])	Execute a series of operators from a given dataframe, and either a processor_config or a workflow_config.
get_custom_operators()	List all user defined operators.
get_default_operator(operator_name)	Return a description of the a specific built-in operator
get_default_operators([op_type_filter])	Return a description of the built-in operators
get_operator_code(operator_name)	Get code for a user defined operator.
get_operator_config(operator_name)	Get operator config for a user defined operator.

snorkelflow.client.operators.add_operator

snorkelflow.client.operators.add_operator(*operator*, *overwrite=False*)

Add an operator (defined as `UserDefinedOperator`).

After writing an operator, you can add it by passing the decorated function object:

```
from snorkelflow.operators import pandas_operator
import snorkelflow.client as sf

@pandas_operator(name="Add Random Number", input_schema={},
output_schema={"rand_num": float})
def add_rand_num(df: pd.DataFrame) -> pd.DataFrame:
    import random
    df["rand_num"] = [random.random() for _ in range(len(df))]
    return df

sf.add_operator(add_rand_num)
```

The custom operator will now show up in the web UI for use as a preprocessor, load processor, and/or postprocessor.

Notes

Available only in the full SDK

Parameters

Name	Type	Default	Info
operator	`Any`		UserDefinedOperator that will be serialized.
overwrite	`Optional[bool]`	`False`	Overwrite an existing operator of the same name if one already exists.

Returns

Metadata associated with this code asset

Return type

`Dict(str, any)`

snorkelflow.client.operators.add_operator_class

snorkelflow.client.operators.add_operator_class(*operator*, *overwrite=False*)

Add an operator class.

Parameters for custom operators are hard-coded within the user-defined function. Custom operator classes have parameters that are specified in the Application Studio overview.

Notes

Available only in the full SDK

Parameters

Name	Type	Default	Info
operator	`Type[Any]`		An custom Operator class that will be serialized.
overwrite	`Optional[bool]`	`False`	Overwrite an existing operator class of the same name if one already exists.

Returns

Metadata associated with this code asset

Return type

`Dict(str, any)`

Raises

- **ValueError** – If the custom operator class name does not begin with *Custom*
- **ValueError** – If the serialized operator class is too large
- **ValueError** – If an operator class with the same name already exists

snorkelflow.client.operators.check_conflicting_operator_name

snorkelflow.client.operators.check_conflicting_operator_name(*operator_name*)

Check whether an Operator name conflicts with the name of a built-in Operator.

Notes

Available only in the full SDK

Parameters

Name	Type	Default	Info
operator_name	`str`		The name of the new operator to be checked.

Raises

ValueError – If operator_name conflicts with a built-in operator name.

Return type

None

snorkelflow.client.operators.delete_operator

snorkelflow.client.operators.delete_operator(*operator_name*)

Delete an operator given the operator name.

Parameters

Name	Type	Default	Info
operator_name	`str`		Name of the custom operator to delete.

Return type

`None`

snorkelflow.client.operators.execute_operators

snorkelflow.client.operators.execute_operators(*df, processor_config=None, workflow_config=None, return_outputs=False*)

Execute a series of operators from a given dataframe, and either a processor_config or a workflow_config.

Parameters

Name	Type	Default	Info
df	`DataFrame`		Pandas dataframe to execute operators over.
processor_config	`Optional[List[Dict[str, Any]]]`	`None`	Processor config to execute.
workflow_config	`Optional[Dict[int, Any]]`	`None`	Workflow config dict to execute.
return_outputs	`bool`	`False`	Whether to return workflow output as a dictionary of output names to dataframes. If False, returns the dataframe corresponding to the last output in the workflow.

Returns

Returns a list of pandas DataFrames that were computed from the processor_config being run over the input_df.

Return type

`List[pandas.DataFrame]`

snorkelflow.client.operators.get_custom_operators

snorkelflow.client.operators.get_custom_operators()

List all user defined operators.

Returns

Dictionary mapping from operator name to its properties

Return type

`Dict(str, Dict(str, Any))`

Examples

```
>>> sf.get_custom_operators()
{
    <custom_operator_name>: {
        'op_uid':  <op_uid>',
        'name':  <name>',
        'created_at': <timestamp>',
        'updated_at': <timestamp>',
        'op_type': <op_type>', # Optype like PandasOperator,
PandasFeaturizer, FieldExtractor, etc.
        'op_config': {
            'name': <operator_name>,
            'f': <serialized_code>,
            'resources': <resources_dict>,
            'resources_fn': <serialized_code>,
            'input_schema': <input_schema_dict>,
            'output_schema': <output_schema_dict>,
        },
        'code': {
            '_f': <code>
        },
    },
    ...
}
```

snorkelflow.client.operators.get_default_operator

snorkelflow.client.operators.get_default_operator(*operator_name*)

Return a description of the a specific built-in operator

Parameters

Name	Type	Default	Info
operator_name	`str`		The name of the operator whose config you wish to inspect.

Returns

Dictionary containing a description of the operator and a description of its input arguments.

Return type

`Dict[str, Any]`

Examples

```
>>> sf.get_default_operator(OPERATOR_NAME)
{
    'description': <description>,
    'fit_inputs': <fit_inputs_dict>,
    'params': <param_config_dict>,
}
```

snorkelflow.client.operators.get_default_operators

snorkelflow.client.operators.get_default_operators(*op_type_filter=None*)

Return a description of the built-in operators

Retreives the configuration format required for instantiating an operator onto a node. Each object returned provides a description of the operator and a list of inputs for that operator.

Parameters

Name	Type	Default	Info
op_type_filter	`Optional[str]`	`None`	An optional type of operator to filter under, e.g. “Featurizer” shows only featurizers.

Returns

Dictionary that maps an operator name to an object that describes the configuration for each operator.

Return type

`Dict[str, Any]`

Examples

```
>>> sf.get_default_operators()
{
    <operator_name>: {
        'description': <description>,
        'fit_inputs': <fit_inputs_dict>,
        'params': <param_config_dict>,
    }
    ...
}
```

snorkelflow.client.operators.get_operator_code

snorkelflow.client.operators.get_operator_code(*operator_name*)

Get code for a user defined operator.

Parameters

Name	Type	Default	Info
operator_name	`str`		Name of the operator source code to fetch.

Returns

Code for the code asset

Return type

`Any`

Examples

```
>>> sf.get_operator_code(operator_name)
<string_of_code_asset>
```

snorkelflow.client.operators.get_operator_config

snorkelflow.client.operators.get_operator_config(*operator_name*)

Get operator config for a user defined operator.

Parameters

Name	Type	Default	Info
operator_name	`str`		Name of the operator source code to fetch.

Returns

Dictionary containing code asset class, config, and code.

Return type

`Dict[str, Any]`

Examples

```
>>> sf.get_operator_config(operator_name)
{
    'dataset_uid': <dataset_uid>,
    'op_type': <op_type>', # Optype like PandasOperator,
PandasFeaturizer, FieldExtractor, etc.
    'op_config': {
        'name': <operator_name>,
        'f': <serialized_code>,
        'resources': <resources_dict>,
        'resources_fn': <serialized_code>,
        'input_schema': <input_schema_dict>,
        'output_schema': <output_schema_dict>,
    },
    'code': {
        '_f': <code>
    },
}
```

snorkelflow.client.secrets

Secret store related functions.

Functions

delete_secret(key[, secret_store, ...])	Deletes a secret from the secret store (Only for superadmin users).
list_integrations([secret_store, ...])	Gets configuration status for each foundation model provider (Only for superadmin users).
list_secrets([secret_store, workspace_uid, ...])	Gets all secret keys in a workspace (Only for superadmin users).
set_secret(key, value[, secret_store, ...])	Adds secret to the secret store (Only for superadmin users).

snorkelflow.client.secrets.delete_secret

snorkelflow.client.secrets.delete_secret(*key, secret_store='local_store', workspace_uid=1, kwargs=None*)

Deletes a secret from the secret store (Only for superadmin users).

Parameters

Name	Type	Default	Info
key	`str`		Key to reference the secret in the store.
secret_store	`str`	`'local_store'`	The secret store to delete the secret (only *local_store* supported now).
workspace_uid	`int`	`1`	The workspace uid for the secret.
kwargs	`Optional[Dict[str, Any]]`	`None`	Other connection kwargs for accesing the secret store.

Return type

`None`

snorkelflow.client.secrets.list_integrations

snorkelflow.client.secrets.list_integrations(*secret_store='local_store', workspace_uid=1, kwargs=None*)

Gets configuration status for each foundation model provider (Only for superadmin users).

Parameters

Name	Type	Default	Info
secret_store	`str`	`'local_store'`	The secret store to list the secret (only *local_store* supported now).
workspace_uid	`int`	`1`	The workspace uid for the secret.
kwargs	`Optional[Dict[str, Any]]`	`None`	Other connection kwargs for accesing the secret store.

Return type

`None`

snorkelflow.client.secrets.list_secrets

snorkelflow.client.secrets.list_secrets(*secret_store='local_store', workspace_uid=1, kwargs=None*)

Gets all secret keys in a workspace (Only for superadmin users).

Parameters

Name	Type	Default	Info
secret_store	`str`	`'local_store'`	The secret store to list the secret (only *local_store* supported now).
workspace_uid	`int`	`1`	The workspace uid for the secret.
kwargs	`Optional[Dict[str, Any]]`	`None`	Other connection kwargs for accesing the secret store.

Returns

All secret keys associated with a workspace

Return type

`List[str]`

snorkelflow.client.secrets.set_secret

snorkelflow.client.secrets.set_secret(*key, value, secret_store='local_store', workspace_uid=1, kwargs=None*)

Adds secret to the secret store (Only for superadmin users).

Parameters

Name	Type	Default	Info
key	`str`		Key to reference the secret in the store.
value	`Union[str, Dict[str, str]]`		The secret being added.
secret_store	`str`	`'local_store'`	The secret store to add the secret (only *local_store* supported now).
workspace_uid	`int`	`1`	The workspace uid for the secret.
kwargs	`Optional[Dict[str, Any]]`	`None`	Other connection kwargs for accesing the secret store.

Return type

`None`

snorkelflow.client.studio

Studio related functions.

> NOTE
> This submodule is available only in the full SDK

Functions

add_code_lf(node, lf[, label, name, ...])	Save a labeling function.
add_model_based_lfs(node, model, model_name, ...)	Performs the following operations in sequence: (a) trains model on the dev set, (b) saves model to minio at the specified path, (c) applies model to predict over the entire dataset, (d) generates labeling functions for all model output labels.
add_onboarding_base_fm(node, model_name[, ...])	Apply Onboarding Base FM to your application.

snorkelflow.client.studio.add_code_lf

snorkelflow.client.studio.add_code_lf(*node, lf, label=None, name=None, is_trusted=False, is_multipolar=False, split='dev', use_code_template=True, apply_lf=True, label_str=None, apply_timeout_secs=None*)

Save a labeling function.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
lf	`LabelingFunction`		Labeling function to be saved.
label	`Optional[Any]`	`None`	Label assigned if the labeling function votes. Label string for single label and sequence nodes, label dictionary for multi-label nodes. Ignored if is_multipolar=True or use_code_template=False.
name	`Optional[str]`	`None`	Name of the labeling function.
is_trusted	`bool`	`False`	If True, mark this LF as a trusted LF.
is_multipolar	`bool`	`False`	Whether this LF outputs multiple labels in the label space.
split	`str`	`'dev'`	Split to apply LF to and reports stats on. Only used if apply_lf = True.

use_code_template	`bool`	`True`	Whether to wrap the LF code in the LF code builder template or not. If False, store the raw LF code passed. Prefer to set use_code_template to True whenever possible.
apply_lf	`bool`	`True`	Whether to apply the LF on the provided split. Note: if apply_lf is set to false, LFs are not validated for errors and may not work in GUI should issues arise. Use with caution.
label_str	`Optional[str]`	`None`	For backwards compatibility only. If label is specified, label_str must be the same as label.
apply_timeout_secs	`Optional[float]`	`None`	Optional param to specify timout for LF apply.

Raises

- **ValueError** – If label_str is not in the set of valid labels for task
- **ValueError** – If label_str is None and use_code_template is True.

Return type

`Dict[str, Any]`

snorkelflow.client.studio.add_model_based_lfs

snorkelflow.client.studio.add_model_based_lfs(*node, model, model_name, feature_fields, **model_kwargs*)

Performs the following operations in sequence: (a) trains model on the dev set, (b) saves model to minio at the specified path, (c) applies model to predict over the entire dataset, (d) generates labeling functions for all model output labels. Useful for generating model-based labeling functions.

Note: this method currently only supports classification end-models.

> NOTE
> The provided model must be a class instance that has the following four methods:

```
class CustomModel:
    def fit(self, X: np.ndarray, y: np.ndarray, **model_kwargs: Any) ->
None:
        # Fits the model on (X, y)
        pass

    def predict(self, X: np.ndarray) -> np.ndarray:
        # Predicts y_hat (shape: (X.shape[0], 1)) from X
        pass

    def save(self, dirpath: str) -> None:
        # Saves necessary model information at dirpath
        pass

    @classmethod
    def load(cls, dirpath: str) -> Any:
        # Loads necessary model information from dirpath
        pass
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
model	`Any`		The model to train. Must have methods fit, predict, save, and load.

model_name	`str`		The name of the model, used to name the prediction column and associated generated LFs.
feature_fields	`Union[str, List[str]]`		The names of the input columns to train the model on.
model_kwargs	`Any`		Any other keyword arguments for model training.

Raises

- **ValueError** – If model does not have the desired methods.
- **ValueError** – If the associated application is not a classification application.

Returns

The trained model.

Return type

`Any`

snorkelflow.client.studio.add_onboarding_base_fm

snorkelflow.client.studio.add_onboarding_base_fm(*node, model_name, columns=None, sync=False*)

Apply Onboarding Base FM to your application. Note: this function currently only applies to classification application.

Parameters

Name	Type	Default	Info
node	`int`		UID of the current model node.
model_name	`str`		For Onboarding Base FM, available models are (recommended) “openai/gpt-3.5-turbo” and “openai/gpt-4”.
columns	`Optional[List[str]]`	`None`	The columns we are applying the Onboarding Base FM on. By default, we are using all text columns. Non-text columns are currently not supported.
sync	`bool`	`False`	Boolean if set to True this call will product the tdm job details until it has completed.

Returns

Info on the request

Return type

`Dict[str, Any]`

Examples

```
>>> results = sf.add_onboarding_base_fm(NODE_UID, "openai/gpt-3.5-
turbo")
{"detail": "In Progress LF created", "job_id": job_id}
>>> # You can check job status by running `sf.poll_job_status(job_id)`
```

snorkelflow.client.synthetic

Synthetic data related functions for generating synthetic data.

Functions

augment_data(data, model_name[, ...])	Augment each row of the data by the number of times specified and return a dataframe with the synthetic data as an additional column.
augment_dataset(dataset, x_uids, model_name)	Augment each row of the dataset by the number of times specified and return a dataframe containing only the synthetic data.

snorkelflow.client.synthetic.augment_data

snorkelflow.client.synthetic.augment_data(*data, model_name, runs_per_prompt=1, prompt='Rewrite the following text whilst retaining the core meaning.', sync=True, **fm_hyperparameters*)

Augment each row of the data by the number of times specified and return a dataframe with the synthetic data as an additional column.

Parameters

Name	Type	Default	Info
data	`Union[List[str], str]`		The data to augment.
model_name	`str`		The name of the foundation model to use.
runs_per_prompt	`int`	`1`	The number of times to augment each row.
prompt	`str`	`'Rewrite the following text whilst retaining the core meaning.'`	The prompt prefix to send to the foundation model together with each row.
sync	`bool`	`True`	Whether to wait for the job to complete before returning the result.
fm_hyperparameters	`Any`		Additional keyword arguments to pass

			to the foundation model such as temperature, max_tokens, etc.

Return type

`Union`[`DataFrame`, `str`]

Returns

- *df* – Dataframe containing the augmentations for the data points.
- *job_id* – The job id of the augment data job which can be used to monitor progress with sf.poll_job_status(job_id).

Examples

```
>>> sf.augment_data(["hello, how can I help you?", "sorry that is not
possible"], "openai/gpt-4")
   | text                          | generated_text
| perplexity
------------------------------------------------------------------------
-------------------------------------
0  | hello, how can I help you?     | welcome, ask me a question to get
started   | 0.0113636364
1  | sorry that is not possible     | unfortunately you cannot do that
| 0.8901232123
```

```
>>> sf.augment_data(["hello, how can I help you?", "sorry that is not
possible"], "openai/gpt-4", runs_per_prompt=2)
   | text                          | generated_text
| perplexity
--------------------------------------------------------------------------
-------------------------------------
0  | hello, how can I help you?     | welcome, ask me a question to get
started   | 0.0113636364
1  | sorry that is not possible     | unfortunately you cannot do that
| 0.8901232123
0  | hello, how can I help you?     | Let me know how to get started.
| 0.2313232442
1  | sorry that is not possible     | bad luck, you cannot do that.
| 0.8313232442
```

snorkelflow.client.synthetic.augment_dataset

snorkelflow.client.synthetic.augment_dataset(*dataset, x_uids, model_name, runs_per_prompt=1, prompt='Your task is to rewrite the a set of text fields whilst retaining the core meaning. You should keep the same language and ensure each re-written field is of a similar length to the original.', fields=None, sync=True, **fm_hyperparameters*)

Augment each row of the dataset by the number of times specified and return a dataframe containing only the synthetic data. By default, all fields are augmented and the foundation model performs the augmentation of each row (all fields) in one inference step.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		The name or UID of the dataset to generate a synthetic augmentation of.
x_uids	`List[str]`		The x_uids within the dataset to augment.
model_name	`str`		The name of the foundation model to use.
runs_per_prompt	`int`	`1`	The number of times to augment each row.
prompt	`str`	`'Your task is to`	The prompt passed to the

		`rewrite the a set of text fields whilst retaining the core meaning. You should keep the same language and ensure each re-written field is of a similar length to the original.'`	foundation model for each row. Note that by default, the prompt is appended with the fields to make the following: “Rewrite the following text fields whilst retaining the core meaning. You should keep the same language and ensure each re-written field is of a similar length to the original. Return your answer in a json format with the same keys as the fields: [field_1, field_2, ...] Here is the data you have to rewrite...”. To override this default behavior, simply pass at least one field wrapped in parentheses, e.g. {field_1}, within the

			prompt and no additional text will be append to the prompt.
fields	`Optional[List[str]]`	`None`	The fields to augment. If not provided, all fields will be augmented.
sync	`bool`	`True`	Whether to wait for the job to complete before returning the result.
fm_hyperparameters	`Any`		Additional keyword arguments to pass to the foundation model such as temperature, max_tokens, etc.

Return type

`Union`[`DataFrame`, `str`]

Returns

- *df* – Dataframe containing the augmentations for the data points.
- *job_id* – The job id of the augment data job which can be used to monitor progress with sf.poll_job_status(job_id).

Examples

```
>>> sf.augment_dataset(dataset=1, x_uids=["0", "1"],
model_name="openai/gpt-4", runs_per_prompt=2)
   | subject                                | body
| perplexity
------------------------------------------------------------------------
------------------------------------------------------------
0  | Fill in survey for $50 amazon voucher  | The email is asking you to
fill in a survey for an amazon voucher       | 0.891
1  | Hey it's Bob, free on Sat?             | The email is from your
friend Bob asking if you're free on Saturday     | 0.787
0  | Free survey for $50                    | Want a free $50 amazon
voucher? Fill in our survey.                     | 0.911
1  | No Plans on Sat, Bob?                  | Let's meet up on Sat. Bob.
| 0.991
```

snorkelflow.client.tags

Tag-related functions. Tags let you mark datapoints with arbitrary labels. Tags can be used to filter documents, create annotation batches, and to give your team members more information about the data.

Functions

add_tag_type(node, name[, description, ...])	Create a new tag type.
add_tags(node, x_uids, tag_types[, ...])	Adds a collection of tags, specified by `(x_uid, tag_type)`, to a given node.
delete_tag_type(node, name[, ...])	Delete an existing tag type from the set of all possible tag types.
delete_tags(node, x_uids, tag_types[, ...])	Deletes a collection of tags, specified by `(x_uid, tag_type)`, from a given node.
get_tag_type(node, name[, is_context_tag_type])	Get a tag type's metadata by its name.
get_tag_types(node[, is_context_tag_type])	Fetches a list of tag type metadata, for all tag types registered to a given node.
get_tags(node[, is_context_tag_type])	Return the tags that are assigned to data points for the given node as a Pandas Series.
update_tag_type(node, previous_name[, ...])	Changes the name and/or description for an existing tag type in a given node.

snorkelflow.client.tags.add_tag_type

snorkelflow.client.tags.add_tag_type(*node, name, description='', is_context_tag_type=False*)

Create a new tag type. A tag type is a label that can be assigned to data points. To use these tag types to tag data points, use `add_tags()`.

Examples

```
>>> sf.add_tag_type(node=1, name="tag1", description="A tag type for
tagging data points")
{
    'tag_type_uid': 52,
    'name': 'tag1',
    'description': 'A tag type for tagging data points',
    'is_context_tag_type': False
}
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node that the tag type will belong to.
name	`str`		Name of the tag type.
description	`str`	`''`	Optional description of the tag type.
is_context_tag_type	`bool`	`False`	If True, the given tag type applies to parent context instead of the individual data point. Only applicable for information extraction tasks.

Returns

A dictionary of metadata corresponding to the newly created tag type.

Return type

`Dict[str, Union[str, int]]`

snorkelflow.client.tags.add_tags

snorkelflow.client.tags.add_tags(*node, x_uids, tag_types, is_context_tag_type=False, missing='error'*)

Adds a collection of tags, specified by `(x_uid, tag_type)`, to a given node. To add a new tag type, use `ad_tag_type()`

Examples

```
>>> # Applies a single tag type to all x_uids
>>> sf.add_tags(node=1, x_uids=["doc::1", "doc::2"], tag_types="tag1")
2

>>> # Applies a unique tag type for each x_uid
>>> sf.add_tags(node=1, x_uids=["doc::1", "doc::2"], tag_types=["tag1",
"tag2"])
2
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node the tag_types are being added to.
x_uids	`List[str]`		A list of x_uids to add tag types to.
tag_types	`Union[List[str], str]`		A tag type name or list of tag type names to add to the corresponding x_uids If a single tag type is provided, it will be added to all given x_uids.
is_context_tag_type	`bool`	`False`	If True, the tag types are added to the parent context of the datapoint, instead of the datapoint

			itself. Only applicable for information extraction tasks.
missing	`str`	`'error'`	How to handle requested tag types that do not exist, can be either `create` or `error`. If “create”, creates missing tag types If “error”, throws an error. By default, “error”.

Returns

The number of new tags that were added

Return type

`int`

snorkelflow.client.tags.delete_tag_type

snorkelflow.client.tags.delete_tag_type(*node, name, is_context_tag_type=False*)

Delete an existing tag type from the set of all possible tag types. To remove a tag from a datapoint, use `delete_tags()`.

Examples

```
>>> sf.delete_tag_type(node=1, name="tag1")
None
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node that the tag type belongs to.
name	`str`		Name of the tag type to delete.
is_context_tag_type	`bool`	`False`	If True, the given tag type applies to parent context instead of the individual data point. Only applicable for information extraction tasks.

Return type

`None`

snorkelflow.client.tags.delete_tags

snorkelflow.client.tags.delete_tags(*node, x_uids, tag_types, is_context_tag_type=False*)

Deletes a collection of tags, specified by `(x_uid, tag_type)`, from a given node. To delete a tag type, use `delete_tag_type()`

Examples

```
>>> # Removes a single tag type from all x_uids
>>> sf.delete_tags(node=1, x_uids=["doc::1", "doc::2"],
tag_types="tag1")
2

>>> # Removes a unique tag type for each x_uid
>>> sf.delete_tags(node=1, x_uids=["doc::1", "doc::2"], tag_types=
["tag1", "tag2"])
2
```

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node containing all the tags to delete.
x_uids	`List[str]`		List of `x_uids` to remove tags from.
tag_types	`Union[List[str], str]`		A tag type name or list of tag types to remove from the corresponding `x_uids` A tag type name or list of tag types to remove from the corresponding `x_uids` If a single tag type is provided, it will be removed from all given `x_uids`

is_context_tag_type	`bool`	`False`	If True, the tag types being removed are removed at the parent context level instead of the individual datapoint level. Only applicable for information extraction tasks.

Returns

The number of tags that were deleted

Return type

`int`

snorkelflow.client.tags.get_tag_type

snorkelflow.client.tags.get_tag_type(*node, name, is_context_tag_type=False*)

Get a tag type's metadata by its name.

Examples

```
>>> sf.get_tag_type(node=1, name="tag1")
{
    'tag_type_uid': 35,
    'name': 'tag1',
    'description': 'First tag type',
    'is_context_tag_type': False
}
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node the tag type belongs to.
name	`str`		The name of the tag type.
is_context_tag_type	`bool`	`False`	If True, the given tag type is at the parent context level instead of the individual datapoint level. Only applicable for information extraction tasks.

Returns

The tag type's metadata

Return type

`Dict[str, Union[str, int]]`

snorkelflow.client.tags.get_tag_types

snorkelflow.client.tags.get_tag_types(*node, is_context_tag_type=False*)

Fetches a list of tag type metadata, for all tag types registered to a given node.

Examples

```
>>> sf.get_tag_types(node=1)
[
    {
        'tag_type_uid': 35,
        'name': 'tag1',
        'description': 'First tag type',
        'is_context_tag_type': False
    },
    {
        'tag_type_uid': 36,
        'name': 'tag2',
        'description': 'Another tag type',
        'is_context_tag_type': False
    }
]
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node whose tag type are returned.
is_context_tag_type	`bool`	`False`	If True, tag types are returned for the parent document context instead of the individual span or datapoint. Only applicable for information extraction tasks.

Returns

A list of tag type metadata for all tag types registered to the given node

Return type

`List[Dict[str, Union[str, int]]]`

snorkelflow.client.tags.get_tags

snorkelflow.client.tags.get_tags(*node, is_context_tag_type=False*)

Return the tags that are assigned to data points for the given node as a Pandas Series.

Examples

```
>>> sf.get_tags(node=1)
x_uid
doc::10005          [tag0, tag1]
doc::10006               [tag1]
doc::10007               [tag2]
doc::10009               [tag1]
doc::10012          [tag0, tag2]
doc::10052               [tag1]
Name: tags, dtype: object
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node whose tags are returned.
is_context_tag_type	`bool`	`False`	If True, the tags fetched apply to parent context instead of at a data point level. Only applicable for information extraction tasks.

Returns

A pandas Series containing a list of tags

Return type

`pd.Series`

snorkelflow.client.tags.update_tag_type

snorkelflow.client.tags.update_tag_type(*node, previous_name, new_name=None, new_description=None, is_context_tag_type=False*)

Changes the name and/or description for an existing tag type in a given node. This will also update all existing tags of this type.

Examples

```
>>> sf.update_tag_type(node=1, previous_name="tag1", new_name="tag3",
new_description="A new description")
{
    'tag_type_uid': 35,
    'name': 'tag3',
    'description': 'A new description',
    'is_context_tag_type': False
}
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node the tag type belongs to.
previous_name	`str`		Name of the tag type whose name/description you wish to change.
new_name	`Optional[str]`	`None`	Optional new name for the tag type.
new_description	`Optional[str]`	`None`	Optional new description for the tag type.
is_context_tag_type	`bool`	`False`	If True, the tag type is at the parent context level instead of the individual datapoint level.

			Only applicable for information extraction tasks.

Returns

A dictionary of metadata for the tag type after being updated

Return type

`Dict[str, Union[str, bool]]`

snorkelflow.client.training_sets

Training set related functions.

Functions

add_training_set(node[, lf_package_version, ...])	Create a training set from an existing LF package (run **package_lfs** first if necessary).
delete_training_set(node, training_set_uid)	Delete a training set from a node.
get_training_set(node, uid[, split, ...])	Return programmatically generated labels (and optionally, probabilities) for a given split.

snorkelflow.client.training_sets.add_training_set

snorkelflow.client.training_sets.add_training_set(*node, lf_package_version=None, tuning_splits=None, parallel_core_fraction=None, partition_size_mb=100, scheduler=None, max_ds_per_apply=None, start_date=None, end_date=None, label_model_list=None, sync=True, register_model=False, register_model_replace_abstain_with_negative=False, max_train_rows=100000*)

Create a training set from an existing LF package (run `package_lfs` first if necessary).

Parameters

Name	Type	Default
node	`int`	
lf_package_version	`Optional[int]`	`None`
tuning_splits	`Optional[List[str]]`	`None`
parallel_core_fraction	`Optional[float]`	`None`

partition_size_mb	`int`	`100`
scheduler	`Optional[str]`	`None`
max_ds_per_apply	`Optional[int]`	`None`
start_date	`Optional[str]`	`None`
end_date	`Optional[str]`	`None`

label_model_list	`Optional[List[Any]]`	`None`
sync	`bool`	`True`
register_model	`bool`	`False`
register_model_replace_abstain_with_negative	`Optional[bool]`	`False`
max_train_rows	`Optional[int]`	`100000`

Returns

A dictionary that looks like *{"job_id": "rq-XX-YYY", "training_set_uid": ZZ}*. The value for *training_set_uid* will be *None* if *sync=False* and/or *create_training_set=False*.

Return type

`Dict[str, Any]`

Examples

```
>>> sf.add_training_set(node_uid)
{"job_id": <job_id>, "training_set_uid": <training_set_uid>}
```

snorkelflow.client.training_sets.delete_training_set

snorkelflow.client.training_sets.delete_training_set(*node, training_set_uid, delete_upto_version=False*)

Delete a training set from a node.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node to remove training set from.
training_set_uid	`int`		UID of the training set to remove.
delete_upto_version	`bool`	`False`	If True, delete all versions of the training set.

Return type

`None`

snorkelflow.client.training_sets.get_training_set

snorkelflow.client.training_sets.get_training_set(*node, uid, split='train', include_labels=True, include_probs=False, include_data_columns=None, overwrite_with_gt=False, filter_unlabeled=True, filter_uncertain_labels=True, tie_break_policy='abstain', sampler_config=None, user_format=False*)

Return programmatically generated labels (and optionally, probabilities) for a given split.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node to load training set from.
uid	`int`		The uid of the training set whose labels and/or probabilities will be loaded. If include_labels or include_probs is Tru this is required.
split	`Optional[str]`	`'train'`	Name of data split t load, one of ["train", "dev", "valid", "test"] Fetching "train" will exclude examples ir the current dev set. None, load all available splits.
include_labels	`bool`	`True`	If True, add a colum containing (int) training set labels (missing = -1).

include_probs	`bool`	`False`	If True, add a colum containing (float) training set probabilities (missir = np.nan).
include_data_columns	`Optional[List[str]]`	`None`	Optional list of columns needed in dataframe. If None, return all data columns. Specifying limited include_data_colum can significantly speed up loading.
overwrite_with_gt	`bool`	`False`	If True, replace training set labels/probabilities with ground truth values where possible.
filter_unlabeled	`bool`	`True`	If True, drop data points where the training set label is (i.e., "abstain"). If overwrite_with_gt is True, filter points w no LF or GT labels.
filter_uncertain_labels	`bool`	`True`	If True, set uncertai labels to "abstain", with uncertain label defined as probs < threshold.

tie_break_policy	`str`	`'abstain'`	How to derive (int) training set labels from (float) training set probs. If “random”, return a random choice among the tied classes. If “abstain”: Return -1 as the training set label.
sampler_config	`Optional[Dict[str, Any]]`	`None`	A dictionary with fields “strategy” (required), “params” (optional), and “class_counts” (optional) representing a sampler configuration. For details, see sampler config. This setting only pertains to returned labels, not probs.
user_format	`bool`	`False`	True if labels are returned in user format, False otherwise.

Returns

An [n_data_points, n_fields] pd.DataFrame containing the task data.

Return type

`DataFrame`

snorkelflow.client.transfer

SDK functions for transferring assets between applications or nodes on a single Snorkel Flow instance.

Functions

convert_span_gt_csv_to_span_format(...[, ...])	Convert exported span ground truth CSV files to old span format for backwards compatibility.
export_ground_truth(node, filepath[, is_context])	Exports ground truth.
export_lfs(node, filepath[, active_only])	Export LFs from a node to the specified filepath as a base64-encoded string.
export_node_data(node, dst_dir)	Saves both the node's LFs and GT to the location specified by `dst_dir`.
export_tag_types(node[, ...])	Export the tag types for a node, optionally to a file.
import_ground_truth(node, filepath[, ...])	Import ground truth labels for any task type.
import_lfs(node, filepath[, merge_type])	Import labeling functions from a source filepath to a destination node.
import_node_data(*, node, src_dir[, uid_col])	Load labeling functions and ground truth labels from `src_dir` to node.
import_tag_types(node, all_tag_types_dict[, ...])	Import `tag_types`.
transfer_annotations(node, from_node[, ...])	Copies annotations from one node to another.
transfer_comments(node, from_node[, merge_type])	Transfer comments from the node with ID `from_node` to the node with ID `node`.

`transfer_gts`(node, from_node[, ...])	Copy ground truth from one node to another.
`transfer_lfs`(node, from_node[, ...])	Copy all active LFs and packaged LFs from one node to another.
`transfer_lfs_by_name`(node, from_node[, ...])	Copy active LFs and packaged LFs selectively from one node to another.
`transfer_tags`(node, from_node[, ...])	Copy tags from `from_node` to `node`.

snorkelflow.client.transfer.convert_span_gt_csv_to_span_format

snorkelflow.client.transfer.convert_span_gt_csv_to_span_format(*gt_csv_path*, *converted_gt_csv_path*, *x_uid_col='x_uid'*, *label_col='label'*)

Convert exported span ground truth CSV files to old span format for backwards compatibility.

Parameters

Name	Type	Default	Info
gt_csv_path	`str`		Path to ground truth CSV file for spans containing x_uid column to convert.
converted_gt_csv_path	`str`		Path to write converted CSV, containing span columns and labels.
x_uid_col	`str`	`'x_uid'`	Column name containing x_uids in the CSV file at gt_csv_path.
label_col	`str`	`'label'`	Column name containing labels in the CSV file at gt_csv_path.

Return type

`None`

snorkelflow.client.transfer.export_ground_truth

snorkelflow.client.transfer.export_ground_truth(*node, filepath, is_context=False*)

Exports ground truth. Returns a two-column table, where the columns are the `x_uid` and `label`. Also writes this table to a CSV file at the provided location.

Examples

```
>>> sf.export_ground_truth(node_uid, gt_filepath)
# returns a pd.DataFrame
            x_uid         label
0           doc::001      "SPAM"
1           doc::002      "HAM"
```

Parameters

Name	Type	Default	Info
node	`int`		ID of the node to export GT from.
filepath	`str`		Path to the ground truth file we export to. This path must be a path in the local filesystem, i.e. cannot be in S3 or MinIO.
is_context	`bool`	`False`	If True, export a ground truth label for the whole document, if False then just for the span. Only available for information extraction tasks.

Return type

`None if there is no gt, otherwise, a pd.DataFrame with columns x_uid and label.`

snorkelflow.client.transfer.export_lfs

snorkelflow.client.transfer.export_lfs(*node, filepath, active_only=None*)

Export LFs from a node to the specified filepath as a base64-encoded string.

Examples

```
>>> sf.export_lfs(node_uid, filepath="my_lfs.txt")
None # Writes the LFs to my_lfs.txt
```

Parameters

Name	Type	Default	Info
node	`int`		ID of the node whose LFs you want to export.
filepath	`str`		Location of file to export. This path must be a path in the local filesystem, i.e. cannot be in S3 or MinIO.
active_only	`Optional[bool]`	`None`	If True, only LFs that are currently “active” in Studio are exported.

Returns

A Base64-encoded serialized dump of all relevant LFs. Use `base64.b64decode` to decode it back into a human-readable format.

Return type

`str`

snorkelflow.client.transfer.export_node_data

snorkelflow.client.transfer.export_node_data(*node, dst_dir*)

Saves both the node's LFs and GT to the location specified by `dst_dir`. Can be thought of as a combination of `export_ground_truth` and `export_lfs`. In contrast to `export_lfs`, this function exports the LFs as a JSON file directly instead of serializing them first.

Examples

```
>>> sf.export_node_data(node_uid, dst_dir)
# Creates a directory at dst_dir containing a CSV file of GTs and a JSON
dump of LFs
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the model node to save LFs and GTs for.
dst_dir	`str`		Directory where LFs and GTs will be saved as JSON dump and CSV, respectively. Only local path is supported.

Return type

`None`

snorkelflow.client.transfer.export_tag_types

snorkelflow.client.transfer.export_tag_types(*node, is_context_tag_type=None, filepath=None*)

Export the tag types for a node, optionally to a file. Tag types describe the set of possible tags that can be assigned to a document at this node. Also exports a `tag_map`, a mapping that points individual documents to the corresopnding tag type UID. You can retrieve a human-readable list of tags by replacing the values in `tag_map` with the corresponding tag name in the `tag_types` list.

Examples

```
>>> sf.export_tag_types(node_uid)
{
    'tag_types': [
        {
            'tag_type_uid': 1,
            'name': "my first tag",
            'description': "this is my first tag!",
            'is_context_tag_type': False
            }
    ],
    'tag_map': {
        'doc::1': [1, ...],
        'doc::2': [1],
    },
}
```

Parameters

Name	Type	Default	Info
node	`int`		ID of the node for which we're exporting ground truth.
is_context_tag_type	`Optional[bool]`	`None`	If True, export only context tag types. If False, export only non-context tag types. If None, export all context tag types. This option only works with non-classification tasks.

filepath	`Optional[str]`	`None`	Path to the file where the tag data is saved. Only local path is supported.

Returns

A dictionary with two keys “tag_types” and “tag_map” “tag_types” value is of type List[Dict[str, Any], which is a list of dictionary configs of tag_types.

”tag_map” value is of type Dict[str, List[int]], mapping x_uids to the associated list of tag_type_uids.

This dictionary can be used for *import_tag_types*.

Return type

`Dict[str, Any]`

snorkelflow.client.transfer.import_ground_truth

snorkelflow.client.transfer.import_ground_truth(*node, filepath, updated_label_schema=None, file_format='CSV', label_col='label', uid_col='x_uid', run_async=False, is_context=False, auto_generate_negative_labels=False, merge_type='FROM'*)

Import ground truth labels for any task type.

The file is read as a dataframe. It is expected to have 2 columns, `x_uid` and `label`, where `x_uid` consists of `x_uids` (unique identifiers) and `label` consists of corresponding ground truth values. The default names of the required columns are `x_uid` and `label`, however, these names can be updated by passing in the *uid_col* and *label_col* parameters.

Alternatively, for span-based tasks, It is expected to have 5 columns: `label`, `context_uid`, `span_field`, `char_start`, `char_end`. The last 4 values must match those of a span for the ground truth to be updated. Different column names for the `label` can be updated by passing in the `label_col` parameters. If using this format, `uid_col` parameter must be set to None.

Examples

gt_filepath.csv: x_uid,label doc::001,SPAM doc::002,HAM

```
>>> sf.import_ground_truth(node_uid, "gt_filepath.csv")
2 # Returns the number of ground truth labels successfully imported
```

Parameters

Name	Type	Default	Info
node	`int`		ID of the node f we're uploading truth.
filepath	`str`		Path to the grou file we're uploa Both both local MinIO path are supported.

updated_label_schema	`Optional[Dict[str, str]]`	`None`	A dictionary tha remap labels fr file into labels i target node. Th for this dictiona the unique labe in the `label` c the file that is b imported. The v this dictionary a corresponding names in the destination noc example, if the contains the lab but the destina node uses the l “animal”, then t dictionary shou {“dog”: “animal”
file_format	`str`	`'CSV'`	Format of the g truth file, either `parquet`.
label_col	`Optional[str]`	`'label'`	Name of the co the ground trut containing the g truth value. Def “label”.
uid_col	`Optional[str]`	`'x_uid'`	Name of the co the ground trut containing uid. is required for classification ta can be blank fo

			task types. Defa "x_uid".
run_async	`Optional[bool]`	`False`	If true, runs the asynchronously returns a job ID be polled with `sf.poll_job_` Otherwise bloc by default.
is_context	`bool`	`False`	If True, import g truth at the doc level, otherwise span level. Only applicable for information ext tasks.
auto_generate_negative_labels	`bool`	`False`	If True, fills in "g sequence taggi with the "negat "other" class. O applicable for s tagging tasks. D False.
merge_type	`str`	`'FROM'`	Specifies how t conflicts betwe imported file ar destination noc either be `TO` or `TO`, any conflic take the values destination noc `FROM` any confl take the values imported file.

Return type

`Union[int, str]`

Returns

- *int* – An int for how many ground truth uids were successfully updated.
- *str* – If the import job is running asynchronously, the job_uid is returned instead

snorkelflow.client.transfer.import_lfs

snorkelflow.client.transfer.import_lfs(*node, filepath, merge_type='UNION'*)

Import labeling functions from a source filepath to a destination node.

Examples

```
>>> sf.import_lfs(model_node_uid, filepath="<lf_dump_filepath>")
{
    'lf_configs_dict': {
        <lf_uid>: {
            'status' : <status>, # success or failure
            'new_uid': <new_lf_uid>,
            'name': , <lf_name>,
        }
        ...
    }
    'packaged_lfs': {
        <lf_package_uid>: {
            {
                'status': 'success',
                'new_uid': <new_lf_package_uid>
                'new_lf_uids': [<new_lf_uid>, ...]
        }
        ...
    }
    'active_lfs': {
        <lf_uid>: {
            'status' : <status>, # success or failure
            'detail': , <detail>,
            'name': , <lf_name>,
        }
        ...
    }
}
```

Parameters

Name	Type	Default	Info
node	`int`		ID of the node you want to import LFs to.

filepath	`str`		Location of file to import. Only local path is supported.
merge_type	`str`	`'UNION'`	This specifies how to handle conflicts, can be one of {'TO','FROM','UNION'}. If 'TO', any conflicts will take the LF definition in the destination node. If 'FROM' any conflicts will take the LF definition from the source file. If 'UNION', Snorkel Flow will attempt to import both as two distinct LFs.

Returns

Returns a dictionary that provides metadata for the LFs that were imported, the LF packages that were imported, and which LFs are active.

Return type

`Dict[str, Any]`

snorkelflow.client.transfer.import_node_data

snorkelflow.client.transfer.import_node_data(*, node, src_dir, uid_col='x_uid')

Load labeling functions and ground truth labels from `src_dir` to node. This function should primarily be used in conjunction with `export_node_data`.

Examples

```
>>> sf.export_node_data(node=123, dst_dir="exported_node_data")
>>> sf.import_node_data(node=456, src_dir="exported_node_data")
None # Imports LFs and GTs from the directory exported_node_data  to
node 123
```

Parameters

Name	Type	Default	Info
node	`int`		UID of the node to load LFs and GTs to.
src_dir	`str`		Directory where LFs and GTs will be loaded from. Only local path is supported, meaning MinIO and S3 paths will not work.
uid_col	`Optional[str]`	`'x_uid'`	Name of the column in the ground truth file containing uid. If None, defaults to `x_uid`.

Return type

`None`

snorkelflow.client.transfer.import_tag_types

snorkelflow.client.transfer.import_tag_types(*node, all_tag_types_dict, tag_types_to_import=None*)

Import `tag_types`. The format of the input dictionary `all_tag_types_dict` should match the output format of `export_tag_types`.

Examples

```
>>> my_export_dict = sf.export_tag_types(123)
>>> sf.import_tag_types(456, my_export_dict)
```

Parameters

Name	Type	Default	Info
node	`int`		ID of the node to which we're uploading tag type.
all_tag_types_dict	`Dict[str, Any]`		A dictionary with two keys "tag_types" and "tag_map" "tag_types" value is of type List[Dict[str, Any], which is a list of dictionary configs of tags_type. "tag_map" value is of type Dict[str, List[int]], mapping x_uids to the associated list of tag_type_uids. This dictionary can be obtained through *export_tag_types*.
tag_types_to_import	`Optional[List[str]]`	`None`	List of tag_types to import. If None, import

			all tag_types. If [], import no tag types.

Return type

`None`

snorkelflow.client.transfer.transfer_annotations

snorkelflow.client.transfer.transfer_annotations(*node, from_node, updated_label_schema=None, merge_type='TO'*)

Copies annotations from one node to another. Assumes that the UIDs in one node exist in the other. If the labels between the two nodes have been renamed, the `updated_label_schema` parameter may be used. If `x_uids` in the source node do not exist in the target node, they will be skipped. Existing annotations in node are not overwritten in case of conflict.

Examples

```
>>> sf.transfer_annotations(123, 456)
{
    'created_annotations': [
        {
            'annotation_uid': 1,
            'x_uid': 'doc::001',
        }
        ...
    ]
}
```

Parameters

Name	Type	Default	Info
node	`int`		ID of the node to which we're copying annotations.
from_node	`int`		ID of the node from which we're copying annotations.
updated_label_schema	`Optional[Dict[str, Optional[str]]]`	`None`	A dictionary that can remap labels from the file into labels in the target node. The keys

			for this dictionary are the unique label strings in the `label` column of the file that is being imported. The values for this dictionary are the corresponding label names in the destination node. For example, if the file contains the label “dog” but the destination node uses the label “animal”, then the dictionary should be {“dog”: “animal”}.
merge_type	`str`	`'TO'`	This is specify how to handle conflicts, can be one of {‘TO’,’FROM’}. If ‘TO’, any conflicts will take the values of the `node`. If ‘FROM’ any conflicts will take the values from the `from_node`.

Returns

Contains one key (“created_annotations” which maps to a list of (annotation_uid, x_uid))

Return type

`Dict[str, List]`

snorkelflow.client.transfer.transfer_comments

snorkelflow.client.transfer.transfer_comments(*node, from_node, merge_type='FROM'*)

Transfer comments from the node with ID `from_node` to the node with ID `node`.

Examples

```
>>> sf.transfer_comments(123, 456)
# Transfer comments from 456 to 123
>>> sf.transfer_comments(123, 456, merge_type="UNION")
# Transfer comments from 456 to 123, taking the union of all comments
from both nodes
>>> sf.transfer_comments(123, 456, merge_type="FROM")
# Transfer comments from 456 to 123, taking the comments from 456
```

Parameters

Name	Type	Default	Info
node	`int`		ID of the node comments are copied to.
from_node	`int`		ID of the node comments are copied from.
merge_type	`str`	`'FROM'`	Merge type for comments, can be one of {'UNION','FROM'}. If 'UNION', comments are transferred without any special identifier. If 'FROM', comment body is prefixed with a header containing information about which node the comment was transferred from.

Return type

`None`

snorkelflow.client.transfer.transfer_gts

snorkelflow.client.transfer.transfer_gts(*node, from_node, updated_label_schema=None, gt_to_import=None, context_gt_to_import=None, merge_type='FROM'*)

Copy ground truth from one node to another. Assumes that the same `x_uids` are present in both nodes. Note that even if the data in the two nodes is the same, the `x_uids` may be different if the nodes were created from different datasets. If `x_uids` are present in the `from_node` but not the `node`, they will be skipped.

Parameters

Name	Type	Default	Info
node	`int`		The node you want to copy ground truth to.
from_node	`int`		The node from which you are copying ground truth.
updated_label_schema	`Optional[Dict[str, Optional[str]]]`	`None`	A dictionary that can remap labels from the file into labels in the target node. The keys for this dictionary are the unique label strings in the `label` column of the file that is being imported. The values for this dictionary are the corresponding label names in the destination node.

			For example, if the file contains the label “dog” but the destination node uses the label “animal”, then the dictionary should be {“dog”: “animal”}.
gt_to_import	`Optional[List[str]]`	`None`	List of classes from the old node for which the ground truth labels should be imported. If None, import all labels. If [], import no labels. Default None (imports all classes).
context_gt_to_import	`Optional[List[str]]`	`None`	List of classes from the old node for which the context ground truth labels should be imported. If None, import all labels. If [], import no labels. This is only applicable for information extraction tasks with document-level ground truth in addition to span-level ground truth. Default None (imports all classes).

merge_type	`str`	`'FROM'`	This specifies how to handle conflicts, can be one of {'TO','FROM'}. If 'TO', any conflicts will take the values from the `node`. If 'FROM' any conflicts will take the values from the `from_node`.

Returns

Contains two keys: `n_gt_transferred` - number of labels successfully transferred `n_context_gt_transferred` - number of context labels successfully transferred.

Return type

`Dict[str, int]`

snorkelflow.client.transfer.transfer_lfs

snorkelflow.client.transfer.transfer_lfs(*node, from_node, updated_label_schema=None, verbose=False, merge_type='UNION'*)

Copy all active LFs and packaged LFs from one node to another.

- LFs are not transferred when their label name is no longer valid (ie it is not part of the label names at the destination node) and a mapping from this old label name to a new label name is not provided through *updated_label_schema*.
- Similarly, custom LFs (aka code LFs) are not imported if their output string is no longer valid at the destination node. Unlike above, this restriction is true even if a mapping is given.
- LFs that were previously active at the source node get reactivated only if their name doesn't conflict with LFs that are currently active at the destination node.
- Packages are recreated using only LFs that are successfully imported, and only if successfully imported LFs have at least 2 labels.

Examples

```
>>> sf.transfer_lfs(node_uid, from_node_uid)
# Transfer all active LFs and packaged LFs from from_node_uid to
node_uid
>>> sf.transfer_lfs(node_uid, from_node_uid, updated_label_schema=
{"dog": "animal"})
# Transfer all active LFs and packaged LFs from from_node_uid to
node_uid, remapping the label "dog" to "animal"'
>>> sf.transfer_lfs(node_uid, from_node_uid, merge_type="FROM")
# Transfer all active LFs and packaged LFs from from_node_uid to
node_uid, taking the LFs from from_node_uid
```

Parameters

Name	Type	Default	Info
node	`int`		ID of the node LFs are copied to.

from_node	`int`		ID of the node LFs are copied from.
updated_label_schema	`Optional[Dict[str, Optional[str]]]`	`None`	A dictionary that can remap labels from the file into labels in the target node. The keys for this dictionary are the unique label strings in the `label` column of the file that is being imported. The values for this dictionary are the corresponding label names in the destination node. For example, if the file contains the label “dog” but the destination node uses the label “animal”, then the dictionary should be {“dog”: “animal”}.
verbose	`bool`	`False`	Whether to return the full response JSON or return None.
merge_type	`str`	`'UNION'`	This specifies how to handle conflicts, can be one of {`TO`,``FROM``, `UNION`}. If `TO`, any

			conflicts will take the values from the `node`. If 'FROM' any conflicts will take the values from the `from_node`. If `UNION`, will take values from both.

Return type

`Optional[Dict[str, Any]]`

snorkelflow.client.transfer.transfer_lfs_by_name

snorkelflow.client.transfer.transfer_lfs_by_name(*node, from_node, lf_names=None, updated_label_schema=None, merge_type='UNION'*)

Copy active LFs and packaged LFs selectively from one node to another. LFs to be copied can be specified by their names.

Examples

```
>>> sf.transfer_lfs_by_name(node_uid, from_node_uid, lf_names=["RGX-lf-1", "RGX-lf-2"])
['RGX-lf-1', 'RGX-lf-2']
```

Parameters

Name	Type	Default	Info
node	`int`		ID of the node LFs are copied to.
from_node	`int`		ID of the node LFs are copied from.
lf_names	`Optional[List[str]]`	`None`	A list of the unique names of the LFs that are copied. LFs that share a name with another LF in the destination node will be skipped. If None, copy all LFs.
updated_label_schema	`Optional[Dict[str, str]]`	`None`	A dictionary that can remap labels from the file into labels in the target node. The keys for this dictionary are the

			unique label strings in the `label` column of the file that is being imported. The values for this dictionary are the corresponding label names in the destination node. For example, if the file contains the label "dog" but the destination node uses the label "animal", then the dictionary should be {"dog": "animal"}.
merge_type	`str`	`'UNION'`	This specifies how to handle conflicts, can be one of {'TO','FROM','UNION'}. If 'TO', any conflicts will take the values from the `node`. If 'FROM' any conflicts will take the values from the `from_node`. If 'UNION', will take values from both.

Returns

List of the LF names that were copied

Return type

`List[str]`

snorkelflow.client.transfer.transfer_tags

snorkelflow.client.transfer.transfer_tags(*node, from_node, tag_types_to_import=None, is_context_tag_type=None*)

Copy tags from `from_node` to `node`.

Examples

```
>>> sf.transfer_tags(node_uid, from_node_uid)
# Transfer all tags from from_node_uid to node_uid
>>> sf.transfer_tags(node_uid, from_node_uid, tag_types_to_import=["my
first tag"])
# Transfer only the tag "my first tag" from from_node_uid to node_uid
```

Parameters

Name	Type	Default	Info
node	`int`		ID of the node to which we're copying tags.
from_node	`int`		ID of the node from which we're copying tags.
tag_types_to_import	`Optional[List[str]]`	`None`	List of tag_types from the old node that should be imported. If None, import all tag_types. If [], import no tag_types.
is_context_tag_type	`Optional[bool]`	`None`	If True, transfer only context tag_types. If False, transfer only non-context tag_types. If None, transfer all context tag_types. This option is only applicable

			for information extraction tasks. Context tag_types are tags at the document-level, whereas other tags are at the span level.

Return type

`None`

snorkelflow.client.users

User related functions.

Functions

`get_user`(user)	Get a user info by its username or user_uid.
`reset_password`(username, new_password)	Reset the password for the specified user to the specified new password.

snorkelflow.client.users.get_user

snorkelflow.client.users.get_user(*user*)

Get a user info by its username or user_uid.

Example

```
>>> sf.get_user("username")
{
    'username': 'username',
    'user_uid': 4,
    'default_view': 'standard',
    'role': 'standard',
    'is_active': True,
    'is_locked': False,
    'email': None,
    'timezone': None,
    'is_superadmin': False
}
```

Parameters

Name	Type	Default	Info
user	`Union[str, int]`		A valid Snorkel Flow user's username or user_uid.

Returns

The user info corresponding to the provided username/user_uid

Return type

`Dict[str, Any]`

snorkelflow.client.users.reset_password

snorkelflow.client.users.reset_password(*username, new_password*)

Reset the password for the specified user to the specified new password.

This functionality is only available to administrators as determined by the API key of the caller.

Parameters

Name	Type	Default	Info
username	`str`		Username of the user whose password you wish to reset.
new_password	`str`		The new password value you wish to set for this user.

Return type

`None`

snorkelflow.client.utils

Utility functions.

Functions

get_application_uid(name)	Fetch the UID of an Application by name
get_batch_uid(node, batch_name)	Translate a batch_name to a batch_uid.
get_dataset_name(dataset_uid)	Fetch the UID of a Dataset by name
get_dataset_uid(dataset)	Fetch the UID of a dataset by name or UID
get_file_storage_config_uid(...)	Fetch the UID of a file storage config by name
get_lf_uid(node, lf_name)	Fetch the UID of an active LF by name.
get_operator_uid(operator_name)	Fetch the UID of an Operator by name.
get_source_uid(source_name)	Translate a source_name to a source_uid.
get_tag_type_uid(node, name[, ...])	Look up the tag_type_uid of a tag type by name.
get_user_uid(username)	Translate a username to a user_uid.
get_workspace_name(workspace_uid)	Fetch the name of a Workspace by UID
get_workspace_uid(workspace_name)	Fetch the UID of a Workspace by name
poll_job_status(job_id[, timeout, verbose])	Poll /jobs endpoint and print statuses.

snorkelflow.client.utils.get_application_uid

snorkelflow.client.utils.get_application_uid(*name*)

Fetch the UID of an Application by name

Parameters

Name	Type	Default	Info
name	`str`		Name of the application.

Returns

UID of the application

Return type

`int`

snorkelflow.client.utils.get_batch_uid

snorkelflow.client.utils.get_batch_uid(*node, batch_name*)

Translate a batch_name to a batch_uid.

Parameters

Name	Type	Default	Info
node	`int`		The UID of the node.
batch_name	`str`		A valid batch name.

Returns

The batch_uid for batch_name

Return type

`int`

snorkelflow.client.utils.get_dataset_name

snorkelflow.client.utils.get_dataset_name(*dataset_uid*)

Fetch the UID of a Dataset by name

Parameters

Name	Type	Default	Info
dataset_uid	`int`		UID of the dataset.

Returns

Name of the dataset

Return type

`str`

snorkelflow.client.utils.get_dataset_uid

snorkelflow.client.utils.get_dataset_uid(*dataset*)

Fetch the UID of a dataset by name or UID

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		Name or UID of the dataset.

Returns

UID of the dataset

Return type

`int`

Raises

ValueError – If a dataset doesn't exist.

snorkelflow.client.utils.get_file_storage_config_uid

snorkelflow.client.utils.get_file_storage_config_uid(*file_storage_config_name*)

Fetch the UID of a file storage config by name

Parameters

Name	Type	Default	Info
file_storage_config_name	`str`		Name of the file storage config.

Returns

UID of the file storage config

Return type

`int`

snorkelflow.client.utils.get_lf_uid

snorkelflow.client.utils.get_lf_uid(*node*, *lf_name*)

Fetch the UID of an active LF by name.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node.
lf_name	`str`		Name of the LF.

Returns

UID of the LF

Return type

`int`

snorkelflow.client.utils.get_operator_uid

snorkelflow.client.utils.get_operator_uid(*operator_name*)

Fetch the UID of an Operator by name.

Parameters

Name	Type	Default	Info
operator_name	`str`		Name of the operator whose UID is desired.

Returns

The UID of the Operator specified by operator_name

Return type

`int`

Raises

OperatorNotFound – If the Operator does not exist in the dataset

snorkelflow.client.utils.get_source_uid

snorkelflow.client.utils.get_source_uid(*source_name*)

Translate a source_name to a source_uid.

Parameters

Name	Type	Default	Info
source_name	`str`		A valid source name.

Returns

The source_uid for source_name

Return type

`int`

snorkelflow.client.utils.get_tag_type_uid

snorkelflow.client.utils.get_tag_type_uid(*node, name, is_context_tag_type=False*)

Look up the tag_type_uid of a tag type by name.

Parameters

Name	Type	Default	Info
node	`int`		UID of the node that the tag type belongs to.
name	`str`		Name of the tag type whose tag_type_uid will be returned.
is_context_tag_type	`bool`	`False`	If True, this is a context-level tag type instead of a datapoint-level tag type.

Returns

The tag_uid of the requested tag type

Return type

`int`

snorkelflow.client.utils.get_user_uid

snorkelflow.client.utils.get_user_uid(*username*)

Translate a username to a user_uid.

Helpful for making API requests that require a user_uid when only a username is available.

Deprecated since version 2024.R2: Use `get_user` instead.

Parameters

Name	Type	Default	Info
username	`str`		A valid Snorkel Flow user's username.

Returns

The user_uid corresponding to the provided username

Return type

`int`

snorkelflow.client.utils.get_workspace_name

snorkelflow.client.utils.get_workspace_name(*workspace_uid*)

Fetch the name of a Workspace by UID

Parameters

Name	Type	Default	Info
workspace_uid	`int`		UID of the workspace.

Returns

Name of the workspace

Return type

`str`

snorkelflow.client.utils.get_workspace_uid

snorkelflow.client.utils.get_workspace_uid(*workspace_name*)

Fetch the UID of a Workspace by name

Parameters

Name	Type	Default	Info
workspace_name	`str`		Name of the workspace.

Returns

UID of the workspace

Return type

`int`

snorkelflow.client.utils.poll_job_status

snorkelflow.client.utils.poll_job_status(*job_id, timeout=None, verbose=True*)

Poll /jobs endpoint and print statuses.

Parameters

Name	Type	Default	Info
job_id	`str`		UID of the job.
timeout	`Optional[timedelta]`	`None`	Optional polling timeout duration, jobs that exceed the timeout will be canceled.
verbose	`bool`	`True`	Optional argument to increase verbosity of the output logs.

Returns

Final job status response if job completes

Return type

`Dict[str, Any]`

Raises

- **JobFailedException** – If job fails
- **JobCancelledException** – If job is cancelled by user

Examples

```
>>> sf.poll_job_status(job_id)
{
    'uid': <job_id>,
    'job_type': <job_type>,
    'state': <state>, # completed or failed
    'enqueued_time': <timestamp>,
    'execution_start_time': <timestamp>,
    'end_time': <timestamp>,
    'application_uid':<application_uid>,
    'dataset_uid': <dataset_uid>,
    'node_uid': <node_uid>,
    'user_uid':<user_uid>,
    'workspace_uid': <workspace_uid>,
    'percent': <percent>, # Based on state
    'message':  <message>,
    'detail': <detail>,
    'pod_name':  <pod_name>,
    'function_name':  <function_name>,
    'process_id':  <process_id>,
    'timing': <timing_dict>
}
```

snorkelflow.client.workflows

Workflow related functions.

Functions

execute_prediction_api(client, df[, ...])	Execute an exported workflow running via Prediction API on a new Pandas DataFrame.
execute_prediction_api_async(client, df[, ...])	Execute an exported workflow running via Prediction API on a new Pandas DataFrame.

snorkelflow.client.workflows.execute_prediction_api

snorkelflow.client.workflows.execute_prediction_api(*client, df, return_outputs=False*)

Execute an exported workflow running via Prediction API on a new Pandas DataFrame.

Parameters

Name	Type	Default	Info
client	`HTTPClient`		SnorkelFlowPredictionAPIClient object.
df	`DataFrame`		Data over which to execute the workflow.
return_outputs	`bool`	`False`	Whether to return workflow output as a dictionary of output names to dataframes. If False, assumes there is only one output in the workflow and directly returns the corresponding the dataframe.

Returns

Dictionary mapping outputs in Workflow DAG to corresponding Pandas DataFrame. If *return_outputs* is False, assumes there is only 1 output, and returns the corresponding dataframe.

Return type

`Dict(str, pandas.DataFrame) or pandas.DataFrame`

snorkelflow.client.workflows.execute_prediction_api_async

snorkelflow.client.workflows.execute_prediction_api_async(*client, df, return_outputs=False*)

Execute an exported workflow running via Prediction API on a new Pandas DataFrame.

It is possible to configure your Prediction API service to provide support for asynchronous inference requests in addition to the standard synchronous requests. If your service is configured to provide this functionality, you can use this method to make a request to the asynchronous endpoint.

Parameters

Name	Type	Default	Info
client	`HTTPClient`		SnorkelFlowPredictionAPIClient object.
df	`DataFrame`		Data over which to execute the workflow.
return_outputs	`bool`	`False`	Whether to return workflow output as a dictionary of output names to dataframes. If False, assumes there is only one output in the workflow and directly returns the corresponding the dataframe.

Returns

Dictionary mapping outputs in Workflow DAG to corresponding Pandas DataFrame. If *return_outputs* is False, assumes there is only 1 output, and returns the corresponding dataframe.

Return type

`Dict(str, pandas.DataFrame) or pandas.DataFrame`

snorkelflow.ingest

Functionality to ingest data of various formats into Snorkel Flow.

Reference

Functions

conversation_json_to_parquet(...)	Generates SnorkelFlow ingestible PARQUET file from JSON.
dirtree_to_parquet(dir_root[, native, ...])	Generate a partitioned parquet file from a directory of PDF documents.
import_utterance_ground_truth(node, ...[, ...])	Uploads ground truth for utterance classification applications.
time_series_csv_to_parquet(...[, label_col, ...])	Generate SnorkelFlow-ingestible PARQUET file from CSV.

snorkelflow.ingest.conversation_json_to_parquet

snorkelflow.ingest.conversation_json_to_parquet(*input_json_file_path, output_parquet_file_path*)

Generates SnorkelFlow ingestible PARQUET file from JSON. Note that we convert all columns to str using json.dumps(). Please use json.loads() if you want to use these columns later.

> NOTE
> Since v0.73, this function no longer adds the output parquet file to a dataset. Please use `create_datasource` to do so.

Parameters

Name	Type	Default	Info
input_json_file_path	`str`		Path to the input JSON file. Local path and MinIO path are supported.
output_parquet_file_path	`str`		Path of the generated parquet file. Only MinIO path is supported.

Return type

`None`

snorkelflow.ingest.dirtree_to_parquet

snorkelflow.ingest.dirtree_to_parquet(*dir_root, native=False, page_selector=None, docs_per_part=1000, parquet_name='generated', scheduler='threads'*)

Generate a partitioned parquet file from a directory of PDF documents.

NB: This function is now workspace-aware, which means the uploaded parquet file will be added to a minio://workspace-#/ workspace-scoped directory. Notably, this means the parquet file will no longer be co-located with a non-workspace-scoped input directory.

Parameters

Name	Type	Default	Info
dir_root	`str`		Path to the directory conta documents. Only MinIO pa supported. Every file (recu the directory with a `.pdf` be ingested. For every file there must be a correspor `xyz.xml` or `xyz.hocr` ir directory if `native=Fals`
native	`bool`	`False`	This should be set to true dealing with only native PI the method won't look for hOCR or XML files for each
page_selector	`Optional[Dict[str, List[int]]]`	`None`	Dictionary to select a few p interest from each docum are file names, and values indexed) page numebers. `{"xyz": [4, 1, ...], [2], ...}`
docs_per_part	`int`	`1000`	Number of documents to partition of the parquet fil

parquet_name	`str`	`'generated'`	The name of the generate The file is created at `dir_root/<parquet_na` removing anyting else that previously.
scheduler	`str`	`'threads'`	[Advanced] Dask schedule perform computations. Ple Snorkel for help with setti parameter to non-default

Return type

`None`

Examples

```
>>> from snorkelflow.ingest import dirtree_to_parquet
>>> dirtree_to_parquet(
>>>     dir_root="minio://pdf-bucket/",
>>>     native=True,
>>>     parquet_name="data",
>>> )
Created parquet file minio://pdf-bucket/data.parquet
```

snorkelflow.ingest.import_utterance_ground_truth

snorkelflow.ingest.import_utterance_ground_truth(*node, filepath, label_col, conversation_id_col='context_uid', utterance_id_col='utterance_idx', metadata=None, user_format=False*)

Uploads ground truth for utterance classification applications. Requires input in the form of [conversation_id, utterance_id, label] triplets.

Parameters

Name	Type	Default	Info
node	`int`		Node uid for the model node to which GTs are to be uploaded.
filepath	`str`		Path to the CSV file which contains GTs. Local path and MinIO path are supported.
label_col	`str`		Name of the column which contains labels. labels are strings, user_format must be True.
conversation_id_col	`str`	`'context_uid'`	Name of the column containing conversation_id Defaults to 'context_uid' if

			using Snorkelflow generated UIDs.
utterance_id_col	`str`	`'utterance_idx'`	Name of the column containing utterance_ids. Defaults to 'utterance_idx'.
metadata	`Optional[Dict[str, Any]]`	`None`	Metadata to register as the provenance of ground truth.
user_format	`bool`	`False`	True if the ground truth labels are strings, False otherwise.

Return type

`None`

snorkelflow.ingest.time_series_csv_to_parquet

snorkelflow.ingest.time_series_csv_to_parquet(*input_csv_file_path, output_parquet_file_path, value_col, timestamp_col, label_col=None, doc_size=None, overlap=0, uid_offset=None, fill_na=None*)

Generate SnorkelFlow-ingestible PARQUET file from CSV.

Examples

Input CSV file:

```
timestamp,value,label
2012-05-11 03:45:00,3.1,0
2012-05-11 04:45:00,1.5,0
2012-05-11 05:45:00,2.3,1
2012-05-11 06:45:00,4.4,1
2012-05-11 07:45:00,8.9,0
2012-05-12 03:45:00,10.1,1
2012-05-12 04:45:00,12.0,1
2012-05-12 05:45:00,11.0,0
2012-05-12 06:45:00,10.0,0
2012-05-12 07:45:00,15.0,1
```

The output parquet file when `doc_size=5` would look like this:

```
                                        values_array
0  gASVRAEAAAAAAABdlCiMMnsidGltZXN0YW1wIjogIjIwMT...
1  gASVEwEAAAAAAABdlCiMM3sidGltZXN0YW1wIjogIjIwMT...
```

This parquet file can be uploaded to a dataset as a data source like below:

```
sf.create_datasource(DATASET_NAME, output_parquet_file_path,
file_type="PARQUET")
```

In the example above, Snorkel Flow generates the UID for each row (or "doc"). If you want to control and make it deterministic (e.g., when you want to upload ground truth labels), please specify `uid_offset` and `uid_col="uid"` at each function.

The output parquet file when `doc_size=5` and `uid_offset=2` would look like this:

```
                                        values_array        uid
0  gASVRAEAAAAAAABdlCiMMnsidGltZXN0YW1wIjogIjIwMT...          2
1  gASVEwEAAAAAAABdlCiMM3sidGltZXN0YW1wIjogIjIwMT...          3
```

Then specify `uid_col="uid"` when uploading this parquet file as follows::

import snorkelflow.client as sf sf.create_datasource(DATASET_NAME, output_parquet_file_path, file_type="PARQUET", uid_col="uid")

Parameters

Name	Type	Default	Info
input_csv_file_path	`str`		Path to the input CSV file. Bot local path and MinIO path are supported.
output_parquet_file_path	`str`		Path of the generated parque Only MinIO path is supported
value_col	`str`		Name of column in CSV conta values.
timestamp_col	`str`		Name of column in CSV conta timestamps. The timestamp h be in a format that can be pa by `pandas.Timestamp`.
label_col	`Optional[str]`	`None`	The name of the column cont ground truth labels, or None such column exists.
doc_size	`Optional[int]`	`None`	The desired size of each docu generated (in terms of the nu of datapoints each contains), excluding overlap. None by d meaning the entire CSV beco one document.
overlap	`int`	`0`	The desired overlap between generated documents. 0 by d
uid_offset	`Optional[int]`	`None`	If specified, "uid" column will added to the parquet file. The starts from `uid_offset`.

fill_na	`Optional[str]`	`None`	If specified, the “value” colum process NaN values via the following strategies (‘fillzero’, ‘pad’,’ffill’,’backfill’,’bfill’,’interp

Return type

`None`

snorkelflow.lfs

Classes that represent LF and LF Package.

Classes

LF(name[, label, templates, ...])	Class to represent an immutable LF.
LFPackage([lfs])	Set of LFs to represent an LF package.
labeling_function([name, resources, ...])	Decorator to create a LabelingFunction object from a user-defined function.

snorkelflow.lfs.LF

class snorkelflow.lfs.LF(*name*, *label=None*, *templates=None*, *multipolar_template=None*, *graph=None*, *is_trusted=False*)

Bases: `object`

Class to represent an immutable LF.

Examples

Create an LF.

```
>>> from snorkelflow.lfs import LF
>>> lf_a = LF(name="lf_a", label=0, templates=...)
```

Due to the immutability, a new LF should be created if you want to change its attributes.

```
>>> lf_config = lf_a.to_dict()
>>> lf_config["name"] = "lf_b"
>>> lf_b = LF(**lf_config)
```

__init__

__init__(*name*, *label=None*, *templates=None*, *multipolar_template=None*, *graph=None*, *is_trusted=False*)

Initialize an LF object.

Parameters

Name	Type	Default	Info
name	`str`		Name of the LF.
label	`Optional[Any]`	`None`	Label associated with the LF. This shouldn't be provided for

			a multi-polar LF.
templates	`Optional[List[Dict[str, Any]]]`	`None`	Templates for the LF. Each template is assigned with a 0-th based index that can be used in *graph*.
multipolar_template	`Optional[Dict[str, Any]]`	`None`	Template for the multi-polar LF.
graph	`Union[str, int, List[Any], None]`	`None`	Provide this argument to construct a *compound LF*, one that's made of multiple templates. Optional if there is only one template. It is a list that starts with a logical operator, followed by template indexes or sub-lists. The logical

			operator can be `"$AND"`, `"$OR"`, or `"$NOT"`. The list can be nested but the depth should be 3 or less. For example, a compound LF with `["$AND", 0, ["$NOT", 1]]` votes if the 0th template returns True and the 1st template returns False, and its depth is 2.
is_trusted	`Optional[bool]`	`False`	If True, designate the LF as trusted.

Methods

`__init__`(name[, label, templates, ...])	Initialize an LF object.
`to_dict`()	Get a dictionary representation of the LF.

Attributes

`graph`	

`is_multipolar`	
`is_trusted`	
`name`	
`label`	
`templates`	
`multipolar_template`	

to_dict

to_dict()

Get a dictionary representation of the LF.

Returns

Dictionary representation of the LF

Return type

`Dict[str, Any]`

graph: `Union[str, int, List[Any], None] = None`

property is_multipolar: *bool*

is_trusted: `Optional[bool] = False`

label: `Optional[Any]`

multipolar_template: `Optional[Dict[str, Any]]`

name: `str`

templates: `Optional[List[Dict[str, Any]]]`

snorkelflow.lfs.LFPackage

class snorkelflow.lfs.LFPackage(*lfs=None*)

Bases: `object`

Set of LFs to represent an LF package.

This class allows manipulating a package at LF-level (e.g., adding an LF) as well as at package-level (e.g., taking a union of packages), but remember that LF names should be unique within a package.

Examples

Initialize a package with a list of LFs.

```
>>> from snorkelflow.lfs import LF, LFPackage
>>> lf_a = LF(name="lf_a", label=0, templates=...)
>>> lf_b = LF(name="lf_b", label=1, templates=...)
>>> pkg = LFPackage([lf_a, lf_b])
>>> [lf.name for lf in pkg]
['lf_a', 'lf_b']
```

Rename all the LFs in a package.

```
>>> pkg.rename_lfs(prefix="my_")
>>> [lf.name for lf in pkg]
['my_lf_a', 'my_lf_b']
```

Rename a single LF. Due to its immutability, an LF should be removed, recreated, and added again.

```
>>> lf_config = pkg.pop("lf_a").to_dict()
>>> lf_config["name"] = "my_lf_a_v1"
>>> pkg.add(LF(**lf_config))
>>> [lf.name for lf in pkg]
['lf_b', 'my_lf_a_v1']
```

__init__

__init__(*lfs=None*)

Initialize a package.

Parameters

Name	Type	Default	Info
lfs	`Union[LF, List[LF], None]`	`None`	List of LFs.

Methods

`__init__`([lfs])	Initialize a package.
`add`(lf)	Add an LF to the package.
`difference`(other)	Return a new package with LFs in the package that are not in the other package.
`get`(name)	Get the LF specified by name.
`intersection`(other)	Return a new package with LFs common to the package and the other.
`pop`(name)	Remove the LF specified by name from the package and return it.
`rename_lfs`([prefix, suffix])	Rename all LFs in the package.
`union`(other)	Return a new package with LFs from the package and the other.
`update_label`(label_mapping)	Update label strings.

add

add(*lf*)

Add an LF to the package.

Parameters

Name	Type	Default	Info
lf	`LF`		LF to add.

Return type

`None`

difference

difference(*other*)

Return a new package with LFs in the package that are not in the other package.

Parameters

Name	Type	Default	Info
other	`LFPackage`		The package to compare.

Returns

New package with LFs in the package that are not in the other package

Return type

LFPackage

get

get(*name*)

Get the LF specified by name.

Parameters

Name	Type	Default	Info

name	`str`		Name of the LF to get.

Returns

LF to get

Return type

LF

intersection

intersection(*other*)

Return a new package with LFs common to the package and the other.

Parameters

Name	Type	Default	Info
other	`LFPackage`		The package to compare.

Returns

New package with LFs common to the package and the other

Return type

LFPackage

Raises

ValueError – If both packages have an LF with the same name but a different value for the other attributes. For example, this package has `LF(name='lf_a', label=0, ...)` and the other has `LF(name='lf_a', label=1, ...)`.

pop

pop(*name*)

Remove the LF specified by name from the package and return it.

Parameters

Name	Type	Default	Info
name	`str`		Name of the LF to remove.

Returns

LF that is removed

Return type

LF

rename_lfs

rename_lfs(*prefix='', suffix=''*)

Rename all LFs in the package.

Parameters

Name	Type	Default	Info
prefix	`str`	`''`	The string to append at the beginning of each LF name.
suffix	`str`	`''`	The string to append at the end of each LF name.

Return type

`None`

union

union(*other*)

Return a new package with LFs from the package and the other.

Parameters

Name	Type	Default	Info
other	`LFPackage`		The package to compare.

Returns

New package with LFs from the package and the other

Return type

LFPackage

Raises

ValueError – If both packages have an LF with the same name but a different value for the other attributes. For example, this package has `LF(name='lf_a', label=0, ...)` and the other has `LF(name='lf_a', label=1, ...)`.

update_label

update_label(*label_mapping*)

Update label strings.

Parameters

Name	Type	Default	Info
label_mapping	`Dict[Any, Any]`		Dictionary that maps from old (raw) label to new (raw) label.

Return type

`None`

Examples

```
>>> # Find LFs whose label is 0 and update their label to 1
>>> pkg.update_label({0: 1})
```

snorkelflow.lfs.labeling_function

class snorkelflow.lfs.labeling_function(*name=None, resources=None, preprocess_configs=None*)

Bases: `object`

Decorator to create a LabelingFunction object from a user-defined function.

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	Name of the LF.
resources	`Optional[Mapping[str, Any]]`	`None`	Labeling resources passed in to `f` via `kwargs`.

			NOTE While this can be a nested dictionary and can include functions, functions should be its direct children if any otherwise they would break when Snorkel Flow upgrades to a newer version of Python.
preprocess_configs	`Optional[List[Dict[str, Any]]]`	`None`	Preprocessors to run on data points before LF execution.

Examples

```
# Simple example
@labeling_function()
def f(x):
    return "SPAM" if "drug" in x.body else "UNKNOWN"

# Example with resources
def find_word_index(text):
    import numpy
    try:
        idx = numpy.where(text.split(" ").index("employee"))
    except:
        idx = numpy.array([])
    return idx

@labeling_function(name="my_lf",
resources=dict(find_word_index=find_word_index))
def lf(x, find_word_index):
    import numpy
    idx = find_word_index(x.text)
    if numpy.mean(idx) <= 1000:
        return "employment"
    else:
        return "UNKNOWN"

# Bad example ("find_word_index" function is NOT a direct child of
"resources")
@labeling_function(name="my_lf",
resources=dict(funcs=dict(find_word_index=find_word_index)))
def bad_lf(x, funcs):
    import numpy
    idx = funcs["find_word_index"](x.text)
    if numpy.mean(idx) <= 1000:
        return "employment"
    else:
        return "UNKNOWN"
```

__init__

__init__(*name=None, resources=None, preprocess_configs=None*)

Methods

__init__([name, resources, preprocess_configs])	

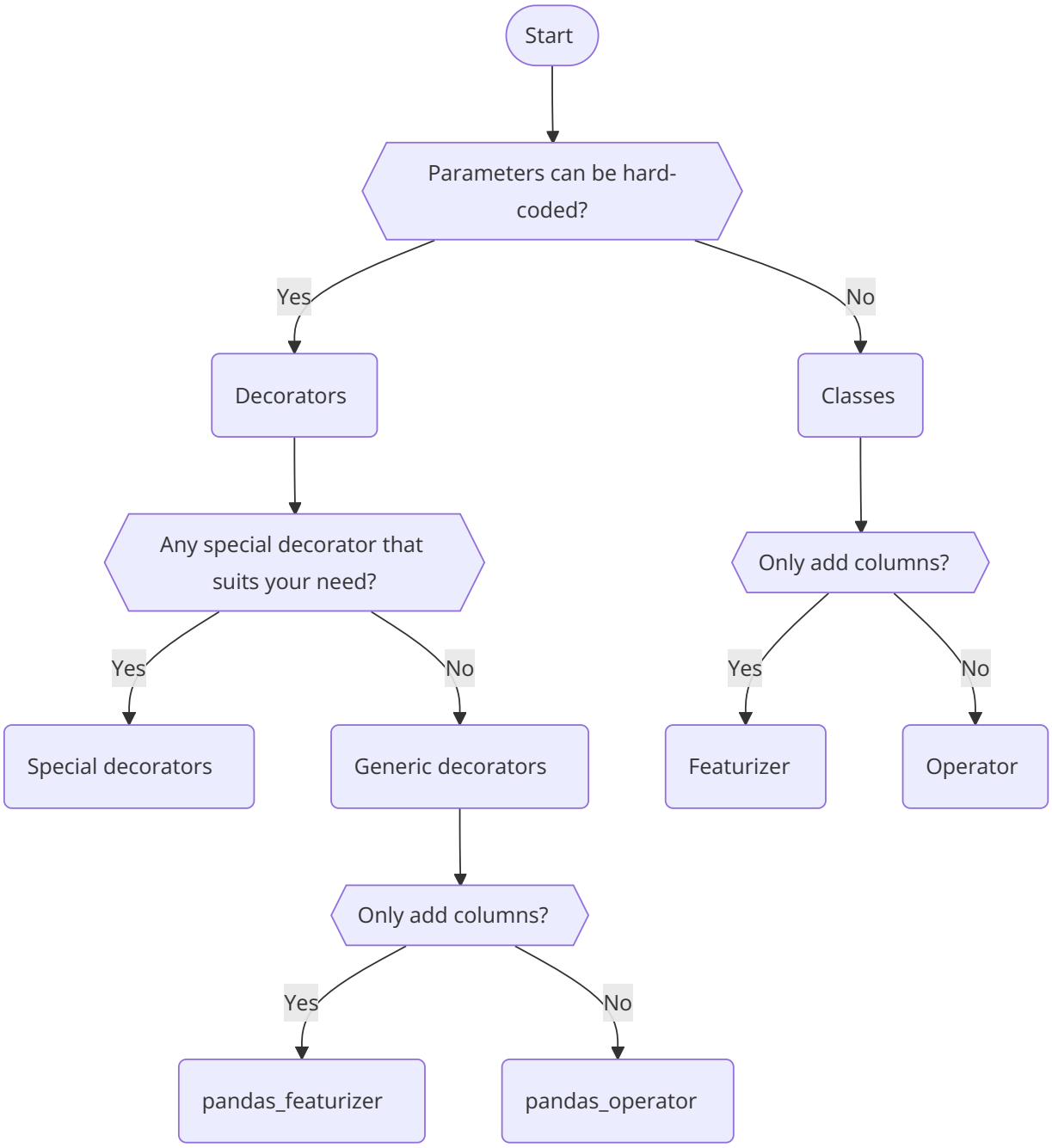
snorkelflow.operators

Functionality for writing custom operators (preprocessors and postprocessors) in Python.

In addition to built-in operators, one can develop a custom operator like below and use it as part of an application. In a nutshell, a custom operator is a Python function that accepts and outputs dataframes, decorated by one of Snorkel Flow operator decorators.

```
@pandas_featurizer(
    name="add_num", input_schema={}, output_schema={"added_num": int},
    resources={"num": 7})
def add_num(df: pd.DataFrame, num: int) -> pd.DataFrame:
    df["added_num"] = num
    return df
```

Generally, parameters for custom operators are hard-coded within the user-defined function when they are developed for simplicity, e.g., `"num": 7` in the example above. To create custom operators that take in parameters when they are used, you can instead create custom operator classes. The diagram below should help you decide which approach to take and which decorator/class to use.



> NOTE
>
> Custom operators can only be developed and registered from the in-platform Notebook server. See also Custom Operators for tutorials.

Special decorators

It's recommended to use special decorators rather than generic decorators whenever possible as the former is easier to use and less error-prone. For example, use `field_extractor` rather than `dask_extractor`.

field_extractor(field[, name, resources, ...])	Decorator for generating candidate spans for extraction tasks.
page_splitter(name[, resources, resources_fn])	Decorator for splitting PDFs into groups of pages.
row_filter(name[, resources, resources_fn])	Decorator for filtering rows of a dataframe.
span_normalizer([name, resources])	Decorator for converting span text to a standard format.
span_reducer([name, datapoint_instance, ...])	Decorator for aggregating span-level model predictions to document-level predictions for extraction tasks.
reducer([name, datapoint_instance, ...])	Decorator for aggregating lower-level model predictions to higher-level predictions.

Generic decorators

If none of the special decorators above suits your need, you can use one of the generic decorators below. In Snorkel Flow, operators, whether built-in or custom, are applied to a Dask dataframe, which is composed of many smaller Pandas dataframes (see here for more details). It's recommended to use `pandas_featurizer` or `pandas_operator`, which allows the user-defined function to deal with each Pandas dataframe at a time for simplicity unless you have to work with the whole Dask dataframe.

pandas_featurizer(*, input_schema[, name, ...])	Decorator for adding columns to a dataframe.
pandas_operator(*, input_schema[, name, ...])	Decorator that wraps a function mapping a `pandas.DataFrame` to another `pandas.DataFrame`.
dask_operator(*, input_schema[, name, ...])	Decorator that wraps a function mapping a `dask.dataframe.DataFrame` to another `dask.dataframe.DataFrame`.

`dask_combiner`(*, input_schema[, name, ...])	Decorator to define Dask Combiner from a function.
`dask_extractor`(*, input_schema[, name, ...])	Decorator to define Dask Extractor from a function.

Classes

`Featurizer`()	Operator class that adds one or more columns (features) to a DataFrame.
`Operator`()	Operator class that performs some transformation on dask dataframes.

snorkelflow.operators.Featurizer

class snorkelflow.operators.Featurizer

Bases: `ChangeColumns`

Operator class that adds one or more columns (features) to a DataFrame.

See Custom Operator Classes for examples.

__init__

__init__()

Methods

`__init__`()	
`check_fit_arguments`(**kwargs)	Check fit arguments.
`column_docs`()	rtype: `Dict`[`str`, `str`]
`estimate_gpu_perf`(df)	rtype: `Performance`
`estimate_perf`(df)	rtype: `Performance`
`examples`()	List of examples (pairs of input df and operator kwargs).
`execute`(input_ddfs[, callback])	Public method for validating + executing operator.
`fit`(df, **kwargs)	Error handling wrapper for _fit method to fit operator to training data.
`fit_arguments`()	Return fit argument types if operator implements _fit method, and None otherwise.

`fit_input_schema`(**kwargs)	Error handling wrapper for _fit_input_schema method.
`get_artifact_config`()	Mapping from operator config keys to data artifact paths.
`get_datapoint_cols`(input_datapoint_cols)	Get datapoint_cols for output DataFrame of the operator, given columns for inputs.
`get_datapoint_instance`(input_datapoint_instances)	Return datapoint_instance of first input.
`get_datapoint_type`(input_datapoint_types)	Get datapoint_type for output DataFrame of the operator, given types for inputs.
`get_featurizer_hash`()	Used as part of hash when storing features.
`get_op_impl_version`()	rtype: `int`
`get_operator_impl_version_to_ds_migration`()	rtype: `Dict`[`int`, `bool`]
`get_predictions`(input_df, output_df)	Operators capable of producting predictions, such as Models and Extractors, can subclass this to define how they return their predictions.
`get_row_hashes`(df)	Returns a unique hash for each row in the input_df.
`is_disabled_by_feature_flag`()	Returns a boolean indicating whether this operator is disabled by a feature flag.
`is_featurizer`()	rtype: `bool`

Attributes

`artifact_config_keys`	
`drop_schema`	
`errors_file_path`	
`fit_params`	
`init_params`	
`input_schema`	Col to dtype map to validate the dtypes of the input dataframe.
`is_deprecated`	
`is_expensive`	
`new_datapoint_cols`	
`node_uid`	
`operator_impl_version`	
`operator_impl_version_to_ds_migration`	
`output_meta`	
`output_schema`	Col to dtype map to validate the dtypes of the output dataframe.
`show_args_in_gui`	
`shuffle_cols`	Used to tell whether operator requires shuffle before execution.
`use_gpu_if_available`	Whether this operator will run on GPU if one is available.

execute

execute(*input_ddfs, callback=<function no_op_progress_callback>*)

Public method for validating + executing operator. Must return dataframe.

Return type

`DataFrame`

get_featurizer_hash

get_featurizer_hash()

Used as part of hash when storing features. Has to be overwritten in order to use the cache_features decorator.

Return type

`str`

get_row_hashes

get_row_hashes(*df*)

Returns a unique hash for each row in the input_df. Only relevant if doing caching using the cache_features decorator.

Return type

`Series`

is_featurizer

static is_featurizer()

Return type

`bool`

drop_schema*:* `Final` *= None*

snorkelflow.operators.Operator

class snorkelflow.operators.Operator

Bases: `object`

Operator class that performs some transformation on dask dataframes.

See Using custom data points as one of the example usages.

__init__

__init__()

Methods

`__init__`()	
`check_fit_arguments`(**kwargs)	Check fit arguments.
`column_docs`()	rtype: `Dict`[`str`, `str`]
`estimate_gpu_perf`(df)	rtype: `Performance`
`estimate_perf`(df)	rtype: `Performance`
`examples`()	List of examples (pairs of input df and operator kwargs).
`execute`(input_ddfs[, callback])	Public method for validating + executing operator.
`fit`(df, **kwargs)	Error handling wrapper for _fit method to fit operator to training data.
`fit_arguments`()	Return fit argument types if operator implements _fit method, and None otherwise.

`fit_input_schema`(**kwargs)	Error handling wrapper for _fit_input_schema method.
`get_artifact_config`()	Mapping from operator config keys to data artifact paths.
`get_datapoint_cols`(input_datapoint_cols)	Get datapoint_cols for output DataFrame of the operator, given columns for inputs.
`get_datapoint_instance`(input_datapoint_instances)	Get datapoint_instance for output DataFrame of the operator, given datapoint_instances for inputs.
`get_datapoint_type`(input_datapoint_types)	Get datapoint_type for output DataFrame of the operator, given types for inputs.
`get_op_impl_version`()	rtype: `int`
`get_operator_impl_version_to_ds_migration`()	rtype: `Dict`[`int`, `bool`]
`get_predictions`(input_df, output_df)	Operators capable of producting predictions, such as Models and Extractors, can subclass this to define how they return their predictions.
`is_disabled_by_feature_flag`()	Returns a boolean indicating whether this operator is disabled by a feature flag.

Attributes

`artifact_config_keys`	
`drop_schema`	List of cols that are dropped as a result of this operator.
`errors_file_path`	

`fit_params`	
`init_params`	
`input_schema`	Col to dtype map to validate the dtypes of the input dataframe.
`is_deprecated`	
`is_expensive`	
`new_datapoint_cols`	
`node_uid`	
`operator_impl_version`	
`operator_impl_version_to_ds_migration`	
`output_meta`	
`output_schema`	Col to dtype map to validate the dtypes of the output dataframe.
`show_args_in_gui`	
`shuffle_cols`	Used to tell whether operator requires shuffle before execution.
`use_gpu_if_available`	Whether this operator will run on GPU if one is available.

check_fit_arguments

classmethod check_fit_arguments(***kwargs*)

Check fit arguments.

Return type

`None`

column_docs

column_docs()

Return type

`Dict[str, str]`

estimate_gpu_perf

estimate_gpu_perf(*df*)

Return type

`Performance`

estimate_perf

estimate_perf(*df*)

Return type

`Performance`

examples

static examples()

List of examples (pairs of input df and operator kwargs).

Return type

`List[OperatorExample]`

execute

execute(*input_ddfs*, *callback=<function no_op_progress_callback>*)

Public method for validating + executing operator. Must return dataframe.

Return type

`DataFrame`

fit

classmethod fit(*df*, ***kwargs*)

Error handling wrapper for _fit method to fit operator to training data.

Return type

`Dict[str, Any]`

fit_arguments

classmethod fit_arguments()

Return fit argument types if operator implements _fit method, and None otherwise.

Return type

`Optional[Dict[str, Any]]`

fit_input_schema

classmethod fit_input_schema(***kwargs*)

Error handling wrapper for _fit_input_schema method.

Return type

`Optional[Dict[str, Any]]`

get_artifact_config

get_artifact_config()

Mapping from operator config keys to data artifact paths.

Keys correspond to artifact_config_keys property.

Return type

`Dict[str, str]`

get_datapoint_cols

get_datapoint_cols(*input_datapoint_cols*)

Get datapoint_cols for output DataFrame of the operator, given columns for inputs.

Return type

`List[str]`

get_datapoint_instance

get_datapoint_instance(*input_datapoint_instances*)

Get datapoint_instance for output DataFrame of the operator, given datapoint_instances for inputs.

Return type

`DatapointType`

get_datapoint_type

get_datapoint_type(*input_datapoint_types*)

Get datapoint_type for output DataFrame of the operator, given types for inputs.

Return type

`Type[DatapointType]`

get_op_impl_version

classmethod get_op_impl_version()

Return type

`int`

get_operator_impl_version_to_ds_migrat

ion

classmethod get_operator_impl_version_to_ds_migration()

Return type

`Dict[int, bool]`

get_predictions

get_predictions(*input_df*, *output_df*)

Operators capable of producting predictions, such as Models and Extractors, can subclass this to define how they return their predictions. df should be the input df to the operator. In the future, if spaces other than the SequenceLabelSpace are required, we can add that parameter here, however try-catch blocks will be required at import time to avoid circular dependencies.

Return type

`Dict[str, Any]`

is_disabled_by_feature_flag

static is_disabled_by_feature_flag()

Returns a boolean indicating whether this operator is disabled by a feature flag.

Return type

`bool`

artifact_config_keys: *`List[str] = []`*

property drop_schema: *List[str] | None*

List of cols that are dropped as a result of this operator.

- If value is *None*, does NOT drop any columns.

errors_file_path: *`Optional[str] = None`*

fit_params: *`Optional[Dict[Any, Any]] = None`*

init_params: *`Optional[Dict[Any, Any]] = None`*

abstract property input_schema: *Dict[str, Any] | None*

Col to dtype map to validate the dtypes of the input dataframe.

- If value is *None*, any schema is allowed.

- These are the _minimum_ required fields. Other fields are allowed by default.
- If type is *None*, any type will be allowed for this field.

is_deprecated: `bool` = False

is_expensive: `bool` = False

new_datapoint_cols: `List`[`str`] = []

node_uid: `Optional`[`int`] = None

operator_impl_version: `int` = 0

operator_impl_version_to_ds_migration: `Dict`[`int`, `bool`] = {}

property output_meta: Dict[str, Any] | None

abstract property output_schema: Dict[str, Any] | None

Col to dtype map to validate the dtypes of the output dataframe.

- If value is *None*, any schema is allowed.
- These are the _minimum_ required fields. Other fields are allowed by default.
- If type is *None*, any type will be allowed for this field.

Don't sort the keys in the output_schema if using the cache_features decorator if the order of features could change depending on the initialization.

show_args_in_gui: `bool` = True

property shuffle_cols: List[str]

Used to tell whether operator requires shuffle before execution. The specified cols will be shuffled into the same partition

property use_gpu_if_available: bool

Whether this operator will run on GPU if one is available.

snorkelflow.operators.dask_combiner

class snorkelflow.operators.dask_combiner(*, *input_schema*, *name=None*, *resources=None*, *resources_fn=None*, *output_schema=None*)

Bases: `dask_operator`

Decorator to define Dask Combiner from a function.

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	Name of the Operator.
resources	`Optional[Dict[str, Any]]`	`None`	Resources passed in to `f` via `kwargs`
resources_fn	`Optional[Callable[[], Dict[str, Any]]]`	`None`	A function for generating a dictionary of values passed to `f` via `kwargs`, that are too expensive to serialize as resources.
input_schema	`Dict[str, Any]`		Dictionary mapping from column to dtype, used to validate the dtypes of the first input dataframe.
output_schema	`Optional[Dict[str, Any]]`	`None`	Dictionary mapping from column to dtype, used to validate the dtypes of the output dataframe. If not `None`, then `f` must not delete any dataframe columns, and all new columns must be specified along with types in `output_schema`.

__init__

__init__(*, input_schema, name=None, resources=None, resources_fn=None, output_schema=None)

Methods

`__init__`(*, input_schema[, name, resources, ...])	

snorkelflow.operators.dask_extractor

class snorkelflow.operators.dask_extractor(*, *input_schema*, *name=None*, *resources=None*, *resources_fn=None*, *output_schema=None*)

Bases: `dask_operator`

Decorator to define Dask Extractor from a function.

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	Name of the Operator.
resources	`Optional[Dict[str, Any]]`	`None`	Resources passed in to `f` via `kwargs`
resources_fn	`Optional[Callable[[], Dict[str, Any]]]`	`None`	A function for generating a dictionary of values passed to `f` via `kwargs`, that are too expensive to serialize as resources.
input_schema	`Dict[str, Any]`		Dictionary mapping from column to dtype, used to validate the dtypes of the input dataframe.
output_schema	`Optional[Dict[str, Any]]`	`None`	Dictionary mapping from column to dtype, used to validate the dtypes of the output dataframe. If not `None`, then `f` must not delete any dataframe columns, and all new columns must be specified along with types in `output_schema`.

__init__

__init__(*, input_schema, name=None, resources=None, resources_fn=None, output_schema=None)

Methods

`__init__`(*, input_schema[, name, resources, ...])	

snorkelflow.operators.dask_operator

class snorkelflow.operators.dask_operator(*, *input_schema*, *name=None*, *resources=None*, *resources_fn=None*, *output_schema=None*)

Bases: `object`

Decorator that wraps a function mapping a `dask.dataframe.DataFrame` to another `dask.dataframe.DataFrame`.

> ⚠ WARNING
>
> Using `@dask_operator` is only recommended for advanced users, as changes to indexes/partitions in the operator will be reflected internally in the Snorkel Flow execution pipeline. For most use cases, `@pandas_operator` should be sufficient.

Examples

The following example is a function that adds a new column `newcol`.

```
import dask.dataframe as dd
from snorkelflow.operators import dask_operator

@dask_operator(name="set_newcol", input_schema={})
def set_newcol(ddf: dd.DataFrame) -> dd.DataFrame:
    import pandas as pd
    import numpy as np
    from dask import dataframe as dd
    import random
    meta = ddf.dtypes.to_dict()
    meta['newcol'] = np.dtype(float)
    def _set_newcol(df: pd.DataFrame) -> pd.DataFrame:
        df['newcol'] = [random.random() for _ in range(len(df))]
        return df
    ddf = dd.map_partitions(_set_newcol, ddf, meta=meta)
    return ddf

sf.add_operator(set_newcol)
```

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	Name of the Operator.

resources	`Optional[Dict[str, Any]]`	`None`	Resources passed in to `f` via `kwargs`
resources_fn	`Optional[Callable[[], Dict[str, Any]]]`	`None`	A function for generating a dictionary of values passed to `f` via `kwargs`, that are too expensive to serialize as resources.
input_schema	`Dict[str, Any]`		Dictionary mapping from column to dtype, used to validate the dtypes of the input dataframe.
output_schema	`Optional[Dict[str, Any]]`	`None`	Dictionary mapping from column to dtype, used to validate the dtypes of the output dataframe. If not `None`, then `f` must not delete any dataframe columns, and all new columns must be specified along with types in `output_schema`.

__init__

__init__(**, input_schema, name=None, resources=None, resources_fn=None, output_schema=None*)

Methods

`__init__`(*, input_schema[, name, resources, ...])	

snorkelflow.operators.field_extractor

class snorkelflow.operators.field_extractor(*field, name=None, resources=None, resources_fn=None*)

Bases: `object`

Decorator for generating candidate spans for extraction tasks.

The field over which candidates should be extracted is specified in the decorator. The wrapped function accepts a string input (which is text in the specified field for a single row of the DataFrame) and outputs a list of tuples `[(char_start, char_end, span_entity)]` (using type annotation syntax: `List[Tuple[int, int, Optional[str]]]`). Each tuple represents the index of the starting character of the span, the index of the ending character of the span, and a canonical identifier string for the entity represented by the span (if applicable).

> ⚠ WARNING
>
> `char_end` represents the inclusive bounds of the span (the index of the last character in the span). In Python syntax, the span substring is `content[char_start : char_end + 1]`.

Examples

The following example demonstrate how to use a custom regex to extract candidates:

```
import re
from snorkelflow.operators import field_extractor

regex = re.compile(r"(?:hello|good morning)", flags=re.IGNORECASE)

@field_extractor(
    name="greetings_extractor",
    field="body",
    resources=dict(compiled_regex=regex),
)
def greetings_extractor(content: str, compiled_regex: re.Pattern) ->
List[Tuple[int, int, Optional[str]]]:
    spans: List[Tuple[int, int, Optional[str]]] = []
    for match in compiled_regex.finditer(content):
        char_start, char_end = match.span(0)
        spans.append((char_start, char_end - 1, None))
    return spans

sf.add_operator(greetings_extractor)
```

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	Name of the Extractor.
field	`str`		Dataframe field that the extraction is operating over.
resources	`Optional[Dict[str, Any]]`	`None`	Resources passed in to `f` via `kwargs`
resources_fn	`Optional[Callable[[], Dict[str, Any]]]`	`None`	A function for generating a dictionary of values passed to `f` via `kwargs`, that are too expensive to serialize as resources.

__init__

__init__(*field, name=None, resources=None, resources_fn=None*)

Methods

`__init__`(field[, name, resources, resources_fn])	

snorkelflow.operators.page_splitter

class snorkelflow.operators.page_splitter(*name*, *resources=None*, *resources_fn=None*)

Bases: `object`

Decorator for splitting PDFs into groups of pages.

Each row's features can be used to detemine how the document is split. The wrapped function accepts a DataFrame row (which generally corresponds to a single document) and outputs a list of page splits. Each page split is a list of page indices in the document corresponding to the split. For example, if you want to split the even and odd pages of a 5 page document, the output would look like [[0,2,4], [1,3]].

> ⚠ WARNING
> Each page can only be in one of the page splits (i.e. you can't have duplicate page indices in different items in the list). Each page split MUST also be in sorted order.

Examples

The following examples demonstrate how to use the decorator:

```python
# Getting the first and last page of each doc
from snorkelflow.operators import page_splitter
@page_splitter(name="first_last_page")
def first_last_page(row: pd.Series) -> List[List[int]]:
    from snorkelflow.rich_docs.rich_doc import RichDocCols
    page_docs = row[RichDocCols.PAGE_DOCS]
    return [[0, len(page_docs) - 1]]

# Splitting the even/odd pages in the document
@page_splitter(name="even_odd_pages")
def even_odd_pages(row: pd.Series) -> List[List[int]]:
    from snorkelflow.rich_docs.rich_doc import RichDocCols
    page_docs = row[RichDocCols.PAGE_DOCS]
    even_splits = list(range(0, len(page_docs), 2))
    odd_splits = list(range(1, len(page_docs), 2))
    return [even_splits, odd_splits]

# Split documents into batches of 5 pages
@page_splitter(name="batch_five_pages")
def batch_five_pages(row: pd.Series) -> List[List[int]]:
    from snorkelflow.rich_docs.rich_doc import RichDocCols
    page_docs = row[RichDocCols.PAGE_DOCS]
    page_ids = list(range(len(page_docs)))
    return [page_ids[i:i + 5] for i in range(0, len(page_ids), 5)]

# Finding all pages that have the compiled regex pattern
import re
pattern = re.compile(r"(?:hellog|good morning)", flags=re.IGNORECASE)

@page_splitter(name="greetings_splitter",
    resources=dict(compiled_pattern=pattern),
)
def greetings_splitter(row: pd.Series, compiled_pattern: re.Pattern) ->
List[List[int]]:
    # Find all pages that have the pattern
    from snorkelflow.rich_docs.rich_doc import RichDocCols
    # Enumerates over each page's RichDoc to find matches
    page_docs = row[RichDocCols.PAGE_DOCS]
    page_ids = []
    for page_idx, page_rd in enumerate(page_docs):
        result = compiled_pattern.findall(page_rd.text)
        if len(result) != 0:
            page_ids.append(page_idx)
    return [page_ids]

# Register any one of the operators above to the dataset
sf.add_operator(greetings_splitter)
```

Parameters

Name	Type	Default	Info
name	`str`		Name of the page splitter.
resources	`Optional[Dict[str, Any]]`	`None`	Resources passed in to `f` via `kwargs`
resources_fn	`Optional[Callable[[], Dict[str, Any]]]`	`None`	A function for generating a dictionary of values passed to `f` via `kwargs`, that are too expensive to serialize as resources.

__init__

__init__(*name, resources=None, resources_fn=None*)

Methods

`__init__`	Initialize self.

snorkelflow.operators.pandas_featurizer

class snorkelflow.operators.pandas_featurizer(**, input_schema, name=None, resources=None, resources_fn=None, output_schema=None, is_join_featurizer=False*)

Bases: `object`

Decorator for adding columns to a dataframe.

It can be used a create a `Featurizer`, namely an operator that only adds columns (features) to a dataframe, but does not add or delete any rows.

A pandas_featurizer **must** include an output schema, which should contain all the new columns added by the wrapped function. If it has an input schema, it should include all the columns needed by the function.

While `@pandas_featurizer` allows you to define custom operators using simple Pandas syntax, they are automatically executed and parallelized using Dask under the hood.

Examples

In the following example, a function that adds one to an integer column to produce another column is defined and wrapped with a `@pandas_featurizer`.

```
from snorkelflow.operators import pandas_featurizer

@pandas_featurizer(name="Add 1", input_schema={"mycol": int},
output_schema={"mycol2": int})
def add_one(df: pd.DataFrame) -> pd.DataFrame:
    return df.assign(mycol2=df.mycol + 1)

sf.add_operator(add_one)
```

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	Name of the Featurizer.
f			Function that accepts as input from `pd.DataFrame` and

			outputs a `pd.DataFrame`.
resources	`Optional[Dict[str, Any]]`	`None`	Resources passed in to `f` via `kwargs`
resources_fn	`Optional[Callable[[], Dict[str, Any]]]`	`None`	A function for generating a dictionary of values passed to `f` via `kwargs`, that are too expensive to serialize as resources.
input_schema	`Dict[str, Any]`		Dictionary mapping from column to dtype. This must include **all** the columns required by `f`.
output_schema	`Optional[Dict[str, Any]]`	`None`	Dictionary mapping from column to dtype. `f` must add exactly the columns specified here to the dataframe.
is_join_featurizer	`bool`	`False`	If True, then the join of the documents and candidates dataframes is given to is as input. Should be False for classification tasks.

__init__

__init__(*, input_schema, name=None, resources=None, resources_fn=None, output_schema=None, is_join_featurizer=False)

Methods

`__init__`(*, input_schema[, name, resources, ...])	

snorkelflow.operators.pandas_operator

class snorkelflow.operators.pandas_operator(*, *input_schema*, *name=None*, *resources=None*, *resources_fn=None*, *output_schema=None*)

Bases: `object`

Decorator that wraps a function mapping a `pandas.DataFrame` to another `pandas.DataFrame`.

While `@pandas_operator` allows you to define custom operators using simple Pandas syntax, they are automatically executed and parallelized using Dask under the hood.

Examples

In the following example, a function that adds a random number to each row is defined. An `output_schema` is explicitly added to indicate the column schema of the resulting DataFrame.

This is used for validation and may provide you with helpful error messages.

```
from snorkelflow.operators import pandas_operator

@pandas_operator(name="Add Random Number", input_schema={},
output_schema={"rand_num": float})
def add_rand_num(df: pd.DataFrame) -> pd.DataFrame:
    import random
    df["rand_num"] = [random.random() for _ in range(len(df))]
    return df

sf.add_operator(add_rand_num)
```

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	Name of the Operator.
resources	`Optional[Dict[str, Any]]`	`None`	Resources passed in to `f` via `kwargs`

resources_fn	`Optional[Callable[[], Dict[str, Any]]]`	`None`	A function for generating a dictionary of values passed to `f` via `kwargs`, that are too expensive to serialize as resources.
input_schema	`Dict[str, Any]`		Dictionary mapping from column to dtype, used to validate the dtypes of the input dataframe.
output_schema	`Optional[Dict[str, Any]]`	`None`	Dictionary mapping from column to dtype, used to validate the dtypes of the output dataframe. If not `None`, then `f` must not delete any dataframe columns, and all new columns must be specified along with types in `output_schema`.

__init__

__init__(*, input_schema, name=None, resources=None, resources_fn=None, output_schema=None)

Methods

`__init__`(*, input_schema[, name, resources, ...])	

snorkelflow.operators.reducer

class snorkelflow.operators.reducer(*name=None, datapoint_instance=None, resources=None, input_schema=None, output_schema=None*)

Bases: `object`

Decorator for aggregating lower-level model predictions to higher-level predictions.

The decorator wraps a function that takes a Pandas DataFrame as input and returns a new Pandas DataFrame. The index of the new DataFrame must be the same as the original DataFrame, unless the `datapoint_instance` is specified. The available datapoints can be found in `DATAPOINT_TYPES` imported from `snorkelflow.utils.datapoint`.

> ⚠ WARNING
> Using custom reducers is only recommended for advanced users as it currently requires knowledge of internal fields and index handling. The `datatpoint_instance` must also be specified.

Examples

This example, finds the first span in the doc by `char_start`

```python
from snorkelflow.operators import reducer
from snorkelflow.utils.datapoint import DocDatapoint
from snorkelflow.extraction.span import SpanCols

# Getting the first span per doc
@reducer(name="first_span_reducer", datapoint_instance=DocDatapoint())
def first_span_reducer(df: pd.DataFrame) -> pd.DataFrame:
    df = df.loc[df.groupby([SpanCols.CONTEXT_UID])
[SpanCols.CHAR_START].idxmin()]
    return df

# Geting the first page per doc (PDF apps)
@reducer(name="first_page_reducer", datapoint_instance=DocDatapoint())
def first_page_reducer(df: pd.DataFrame) -> pd.DataFrame:
    from snorkelflow.extraction.span import SpanCols
    # Gets the first page from each doc
    index_col = df.index.name
    df = df.reset_index()
    df = df.groupby([SpanCols.CONTEXT_UID]).first()
    df = df.reset_index().set_index(index_col)
    return df

# Getting multiple spans per doc
@reducer(name="multi_span_reducer", datapoint_instance=DocDatapoint())
def multi_span_reducer(df: pd.DataFrame) -> pd.DataFrame:
    index_col = df.index.name
    df = df.reset_index()
    df = df.groupby([SpanCols.CONTEXT_UID]).agg(list)
    df[index_col] = df[index_col].str[0]
    df = df.reset_index().set_index(index_col)
    return df

sf.add_operator(multi_span_reducer)
```

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	Name of the Reducer.
f			Function that accepts as input from

			`pd.DataFrame` and outputs a `pd.DataFrame`.
datapoint_instance	`Optional[DatapointType]`	`None`	An instance of the datapoint type to reduce to. For example if you are reducing from spans to docs, this should be DocDatapoint.
resources	`Optional[Dict[str, Any]]`	`None`	Resources passed in to `f` via `kwargs`
input_schema	`Optional[Dict[str, Any]]`	`None`	Dictionary mapping from column to dtype, used to validate the dtypes of the input dataframe.
output_schema	`Optional[Dict[str, Any]]`	`None`	Dictionary mapping from column to dtype, used to validate the dtypes of the output dataframe.

__init__

__init__(*name=None, datapoint_instance=None, resources=None, input_schema=None, output_schema=None*)

Methods

`__init__`	Initialize self.

snorkelflow.operators.row_filter

class snorkelflow.operators.row_filter(*name*, *resources=None*, *resources_fn=None*)

Bases: `object`

Decorator for filtering rows of a dataframe.

The decorator wraps a function that takes a Pandas DataFrame as input and returns a Pandas Series that's used to filter the input dataframe.

Examples

The following examples demonstrate how to create a custom filter:

```
from snorkelflow.operators import row_filter
import pandas as pd

# Keep all spans that have more than 5 words
@row_filter(name="greater_than_five_words")
def greater_than_five_words(df: pd.DataFrame) -> pd.Series:
    return df["span_text"].str.split().str.len() > 5

# Keep all PDFs with greater than 5 pages
@row_filter(name="greater_than_five_pages")
def greater_than_five_pages(df: pd.DataFrame) -> pd.Series:
    from snorkelflow.rich_docs import RichDoc, RichDocCols
    page_docs = df[RichDocCols.PAGE_DOCS]
    return page_docs.apply(lambda x: len(x) > 5)

# Filter rows based on compiled regex pattern
import re
pattern = re.compile(r"(?:hello|good morning)", flags=re.IGNORECASE)

@row_filter(
    name="greetings_filter",
    resources=dict(compiled_rgx=pattern),
)
def greetings_filter(df: pd.DataFrame, compiled_rgx: re.Pattern) ->
pd.Series:
    text_col = df["text"]
    return text_col.apply(lambda x: bool(compiled_rgx.match(x)))

# Register any one of the row_filters above
sf.add_operator(greetings_filter)
```

Parameters

Name	Type	Default	Info
name	`str`		Name of the row_filter.
resources	`Optional[Dict[str, Any]]`	`None`	Resources passed in to `f` via `kwargs`
resources_fn	`Optional[Callable[[], Dict[str, Any]]]`	`None`	A function for generating a dictionary of values passed to `f` via `kwargs`, that are too expensive to serialize as resources.

__init__

__init__(*name, resources=None, resources_fn=None*)

Methods

`__init__`	Initialize self.

snorkelflow.operators.span_normalizer

class snorkelflow.operators.span_normalizer(*name=None, resources=None*)

Bases: `object`

Decorator for converting span text to a standard format.

The decorator wraps a function that takes a string of the span text (e.g. `"Jan. 1, 2020"`) as input and outputs the normalized string (e.g. `"2020-01-01"`).

Normalization is useful before reduction so that the reducer can aggregate over different forms of the same entity.

Examples

For example, a date normalizer can be used when trying to extract dates from a document to ensure that a reducer will aggregate all forms of the same date (e.g. the spans `"Jan. 1, 2020"` and `"2020-01-01"` are treated as the same date and the model predictions can be aggregated).

```
from snorkelflow.operators import span_normalizer

@span_normalizer(name="date_normalizer")
def date_normalizer(span_text: str) -> str:
    import re
    from dateutil.parser import parse

    pattern = re.compile(r"([0-9]{1,2})(?:st|nd|rd|th)? day of")
    sub_pattern = pattern.sub(r"\1", span_text)
    if isinstance(sub_pattern, int):
        return sub_pattern
    try:
        parsed = str(parse(sub_pattern).date())
    except ValueError:
        return span_text
    return parsed

sf.add_operator(date_normalizer)
```

Parameters

Name	Type	Default	Info

name	`Optional[str]`	`None`	Name of the Normalizer.
f			Function that accepts as input a string and outputs a string.
resources	`Optional[Dict[str, Any]]`	`None`	Resources passed in to `f` via `kwargs`

__init__

__init__(*name=None, resources=None*)

Methods

`__init__`([name, resources])	

snorkelflow.operators.span_reducer

class snorkelflow.operators.span_reducer(*name=None, datapoint_instance=name=span, columns=['context_uid', 'span_field', 'span_field_value_hash', 'char_start', 'char_end'], resources=None, input_schema=None, output_schema=None*)

Bases: `object`

Decorator for aggregating span-level model predictions to document-level predictions for extraction tasks.

The decorator wraps a function that takes a Pandas DataFrame as input and returns a new Pandas DataFrame. The index of the new DataFrame must be the same as the original DataFrame, unless the `datapoint_instance` is specified. The available datapoints can be found in `DATAPOINT_TYPES` imported from `snorkelflow.utils.datapoint`.

> ⚠ WARNING
> Using custom span reducers is only recommended for advanced users as it currently requires knowledge of internal fields and index handling.

Examples

The following pattern of grouping by and reducing over the `"context_uid"` index is common.

```python
from snorkelflow.operators import span_reducer
from snorkelflow.extraction.span import SpanCols

# Get the first span in each document
@span_reducer(name="first_reducer")
def first_reducer(df: pd.DataFrame) -> pd.DataFrame:
    df = df.loc[df.groupby([SpanCols.CONTEXT_UID])
[SpanCols.CHAR_START].idxmin()]
    return df

# Get the last span in the doc and convert to a DocDatapoint
from snorkelflow.utils.datapoint import DocDatapoint
@span_reducer(name="last_reducer", datapoint_instance=DocDatapoint())
def last_reducer(df: pd.DataFrame) -> pd.DataFrame:
    df = df.loc[df.groupby([SpanCols.CONTEXT_UID])
[SpanCols.CHAR_START].idxmin()]
    return df

# Get multiple spans per doc expressed as a list
@span_reducer(name="multi_reducer", datapoint_instance=DocDatapoint())
def multi_reducer(df: pd.DataFrame) -> pd.DataFrame:
    from snorkelflow.extraction.span import SpanCols
    index_col = df.index.name
    df = df.reset_index()
    df = df.groupby([SpanCols.CONTEXT_UID]).agg(list)
    df[index_col] = df[index_col].str[0]
    df = df.reset_index().set_index(index_col)
    return df

# Register any of the above reducers to make them available to add in
the DAG
sf.add_operator(multi_reducer)
```

Parameters

Name	Type	Default
name	`Optional[str]`	`None`
f		

datapoint_instance	`Optional[DatapointType]`	`name=span, columns=['context_uid', 'span_field', 'span_field_value_hash', 'char_start', 'char_end']`
resources	`Optional[Dict[str, Any]]`	`None`
input_schema	`Optional[Dict[str, Any]]`	`None`
output_schema	`Optional[Dict[str, Any]]`	`None`

__init__

__init__(*name=None, datapoint_instance=name=span, columns=['context_uid', 'span_field', 'span_field_value_hash', 'char_start', 'char_end'], resources=None, input_schema=None,*

output_schema=None)

Methods

__init__([name, datapoint_instance, ...])	

snorkelflow.rich_docs

Utilities to manipulate Rich Document structures for labeling functions and operators in Snorkel Flow.

Labeling functions that use rich documents

When rich document information is available, the `RichDoc` object is available for you to write labeling functions over. We also provide a `RichDocWrapper` class that provides additional helpers and derived properties for a given `RichDoc` object. You can import these classes and use them to operate over the serialized `RichDoc` object.

```
from snorkelflow.lfs.lfs import labeling_function

@labeling_function(name="rich_doc_lf")
def rich_doc_lf(x):
    from snorkelflow.rich_docs import RichDocCols, RichDoc
    from snorkelflow.rich_doc_wrapper import RichDocWrapper

    # Deserialize the RichDoc object
    rd = x[RichDocCols.PAGE_DOCS].apply(lambda x: RichDoc.from_page_docs(x))
    # Wrap RichDoc object and text in RichDocWrapper class
    rdw = RichDocWrapper(rd)

    # Get the left bounding box coordinate for the given span
    span_left = rdw.get_span_ngram(x["char_start"], x["char_end"]).left
    # Label INVALID if span bounding box begins too far to the left
    if span_left < 1100:
        return "INVALID"
    return "UNKNOWN"

sf.add_code_lf(node, rich_doc_lf, label="INVALID")
```

Classes

`HVLines`(dfs_horz, dfs_vert)	Serializable wrapper for lists of horizontal and vertical lines in image.
`RichDoc`(pages, areas, pars, lines, words [, ...])	An object representing a document with rich formatting preserved.
`RichDocCols`()	Base class that specifies Rich Doc columns.

`RichDocList`(rich_docs)	Serializable wrapper for list of RichDoc's.
`Serializable`()	Interface for types that have a custom serialization / deserialization function.
`TextClusters`(word_to_cluster, df_clusters)	Serializable wrapper for horizontal clusters of words.

Exceptions

`MissingRichDocException`	An Exception raised when a RichDoc was expected and could not be found.

snorkelflow.rich_docs.HVLines

class snorkelflow.rich_docs.HVLines(*dfs_horz, dfs_vert*)

Bases: `Serializable`

Serializable wrapper for lists of horizontal and vertical lines in image.

__init__

__init__(*dfs_horz, dfs_vert*)

Methods

`__init__`(dfs_horz, dfs_vert)	
`deserialize`(serialized)	Deserialize instance to string.
`serialize`()	Serialize instance to string.

deserialize

classmethod deserialize(*serialized*)

Deserialize instance to string.

Return type

`HVLines`

serialize

serialize()

Serialize instance to string.

Return type

`str`

snorkelflow.rich_docs.MissingRichDocException

exception snorkelflow.rich_docs.MissingRichDocException

An Exception raised when a RichDoc was expected and could not be found.

snorkelflow.rich_docs.RichDoc

class snorkelflow.rich_docs.RichDoc(*pages*, *areas*, *pars*, *lines*, *words*, *text=None*, *text_by_row=None*)

Bases: `Serializable`

An object representing a document with rich formatting preserved.

For additional utilities for operating on a RichDoc document, see the RichDocWrapper class.

Types of signals available via RichDoc:

- textual:
    - A text-only representation of the document (no markup present)
- structural:
    - All words are assigned a line, par(agraph), area, and page. Lines may include font size information
- visual:
    - Objects of all granularities contain bounding box coordinates
- tabular:
    - Currently expressed primarily via visual signals (e.g., vertical and horizontal alignments)

Other Notes:

- [0, 0] is in the top-left corner (so bottom > top and right > left)

Parameters

Name	Type	Default	Info
pages	`DataFrame`		A DataFrame of page objects.
areas	`DataFrame`		A DataFrame of area objects.
pars	`DataFrame`		A DataFrame of paragraph objects.
lines	`DataFrame`		A DataFrame of line objects.

words	`DataFrame`		A DataFrame of word objects, with bounding boxes, parent obj. assignments, etc.
text	`Optional[str]`	`None`	The derived text representation of the RichDoc if it is known. Typically this is None and the text is auto-generated to ensure consistency. It may be passed, however, when serializing/deserializing, for efficiency.

__init__

__init__(*pages, areas, pars, lines, words, text=None, text_by_row=None*)

Methods

`__init__`(pages, areas, pars, lines, words [, ...])	
`deserialize`(serialized)	Deserialize the RichDoc instance from the encoded pickle representation.
`extract_span_page`(char_start, char_end)	Create a RichDoc with only objects on the same page as the requested span.
`from_page_docs`(page_docs)	Construct a Richdoc from the page richdoc.
`from_records`(*, pages, areas, pars, lines, words)	Construct RichDoc object from records lists.
`get_row_text`(page, row)	Return the text corresponding to a given row_id and page_id.
`get_span_location_fields`(char_start, char_end)	Get the location of a span of words in the rich doc in terms of word_ids.
`normalize_char_starts`()	Normalize char offsets for pages and words dfs inplace.

`serialize`()	Serialize the RichDoc instance into an encoded pickle representation.
`split_pages`()	Split a RichDoc document into a list of RichDocs, one page per doc.
`to_hocr`()	Convert a RichDoc document into an hOCR formatted string.
`to_json`([start_page])	Convert a RichDoc document into a JSON formatted string.
`to_text`()	Return a text-only representation of a RichDoc (without markup).

Attributes

`page_char_starts`	Get an array of the char_start for all pages in the RichDoc.
`word_char_ends`	Get an array of the char_end for all words in the RichDoc.
`word_char_starts`	Get an array of the char_start for all words in the RichDoc.
`word_indexes`	Get an array of the word indexes for all words in the RichDoc.

deserialize

classmethod **deserialize**(*serialized*)

Deserialize the RichDoc instance from the encoded pickle representation.

Parameters

Name	Type	Default	Info
serialized	`str`		A base64-encoded string representation of the RichDoc pickle.

Return type

`RichDoc`

extract_span_page

extract_span_page(*char_start, char_end*)

Create a RichDoc with only objects on the same page as the requested span.

Because visual signals are only guaranteed for words on a span's page, dropping the other pages can save a significant amount of space, especially in documents with many pages.

Parameters

Name	Type	Default	Info
char_start	`int`		The inclusive start index of the span in RichDoc.text.
char_end	`int`		The inclusive end index of the span in RichDoc.text.

Return type

`RichDoc`

from_page_docs

classmethod from_page_docs(*page_docs*)

Construct a Richdoc from the page richdoc.

Parameters

Name	Type	Default	Info
page_docs	`RichDocList`		RichDocList datastructure.

Return type

`RichDoc`

from_records

classmethod from_records(*, *pages*, *areas*, *pars*, *lines*, *words*, *convert_to_int=True*)

Construct RichDoc object from records lists.

Return type

`RichDoc`

get_row_text

get_row_text(*page*, *row*)

Return the text corresponding to a given row_id and page_id.

Return type

`str`

get_span_location_fields

get_span_location_fields(*char_start*, *char_end*, *span_words=None*)

Get the location of a span of words in the rich doc in terms of word_ids.

These fields are used by the frontend for rendering.

Parameters

Name	Type	Default	Info
char_start	`int`		The inclusive start index of the span in RichDoc.text.
char_end	`int`		The inclusive end index of the span in RichDoc.text.

span_words	`Optional[DataFrame]`	`None`	The dataframe of words associated with the span. If not specified, it is computed using char start and char end.

Return type

`Dict[str, Any]`

normalize_char_starts

normalize_char_starts()

Normalize char offsets for pages and words dfs inplace.

This is needed when a RichDoc is constructed from disjoint pages (e.g. from_page_docs)

Return type

`None`

serialize

serialize()

Serialize the RichDoc instance into an encoded pickle representation.

Return type

`str`

split_pages

split_pages()

Split a RichDoc document into a list of RichDocs, one page per doc.

Return type

`RichDocList`

to_hocr

to_hocr()

Convert a RichDoc document into an hOCR formatted string.

Return type

`str`

to_json

to_json(*start_page=0*)

Convert a RichDoc document into a JSON formatted string.

Return type

`str`

to_text

to_text()

Return a text-only representation of a RichDoc (without markup).

Return type

`str`

property page_char_starts*: List[int]*

Get an array of the char_start for all pages in the RichDoc.

NOTE: If the RichDoc has been trimmed to a subset of pages, then only the char_start values for those pages will be present in this attribute.

property word_char_ends*: ndarray*

Get an array of the char_end for all words in the RichDoc.

property word_char_starts*: ndarray*

Get an array of the char_start for all words in the RichDoc.

property word_indexes*: List[int]*

Get an array of the word indexes for all words in the RichDoc.

Note that the index is 0-based and relative to the whole page. For example, when a RichDoc represents the 2nd page of a 2-page document, and the word indexes for

such a RichDoc will be n, n+1, n+2, ..., n+m, assuming the first page has n words.

snorkelflow.rich_docs.RichDocCols

class snorkelflow.rich_docs.RichDocCols

Bases: `object`

Base class that specifies Rich Doc columns.

__init__

__init__()

Methods

`__init__`()	

Attributes

`CHECKBOXES`	A `Layout` object that captures checkboxes in a doc; requires a *CheckboxFeaturizer* in the DAG
`CONTEXT_PAGES`	JSON dump containing the positions of words, lines, etc
`DOC_COL`	The RichDoc object, which can be used to extract properties of the document
`HV_LINES`	A `HVLines` object that captures the vertical/horiztonal lines in a doc; requires a *LinesFeaturizer* in the DAG
`IS_BOTTOM_CHECKBOX_CHECKED`	A boolean field that denotes whether a bottom checkbox is checked or not
`IS_CHECKED`	A boolean field that denotes whether a checkbox is checked or not
`IS_LEFT_CHECKBOX_CHECKED`	A boolean field that denotes whether a left checkbox is checked or not

`IS_RIGHT_CHECKBOX_CHECKED`	A boolean field that denotes whether a right checkbox is checked or not
`IS_TABLE_SPAN`	
`IS_TOP_CHECKBOX_CHECKED`	A boolean field that denotes whether a top checkbox is checked or not
`JSON_COL`	
`PAGE_CHAR_STARTS`	A list of character offsets that denote where in the rich_doc_text each page starts
`PAGE_DOCS`	A list of RichDoc objects where each item corresponds to the RichDoc of a single page, in order
`PDF_URL_COL`	The URL/file location of the PDF
`PKL_COL`	Serialized version of the RichDoc object
`SPAN_END_CHAR_OFFSET`	The offset from the char_end of the last word in a span
`SPAN_END_WORD_ID`	The id of the word at the end of the span, relative to all other words in the document
`SPAN_NGRAM`	An n-gram of the words in the span; See `Ngram` for more
`SPAN_PAGE_ID`	The page # that the span belongs to
`SPAN_START_CHAR_OFFSET`	The offset from the char_start of the first word in a span

`SPAN_START_WORD_ID`	The id of the word at the start of the span, relative to all other words in the document
`TABLES`	
`TABLE_COLUMN_ID`	
`TEXT_CLUSTERS`	A `TextClusters` object object that captures horizontal clusters of words; requires a *TextClusterer* in the DAG
`TEXT_CLUSTER_ID`	A global cluster id associated with each text cluster
`TEXT_COL`	Extracted plain text from the PDF
`TEXT_REGION_ID`	When *LinesFeaturizer* is added, the user can group vertical and horizontal lines into regions; this field denotes which region is associate with a span

CHECKBOXES = *'checkboxes'*

A `Layout` object that captures checkboxes in a doc; requires a *CheckboxFeaturizer* in the DAG

CONTEXT_PAGES = *'context_pages'*

JSON dump containing the positions of words, lines, etc

DOC_COL = *'rich_doc'*

The RichDoc object, which can be used to extract properties of the document

HV_LINES = *'hv_lines'*

A `HVLines` object that captures the vertical/horiztonal lines in a doc; requires a *LinesFeaturizer* in the DAG

IS_BOTTOM_CHECKBOX_CHECKED = *'is_bottom_checkbox_checked'*

A boolean field that denotes whether a bottom checkbox is checked or not

IS_CHECKED = *'is_checked'*

A boolean field that denotes whether a checkbox is checked or not

IS_LEFT_CHECKBOX_CHECKED = *'is_left_checkbox_checked'*

A boolean field that denotes whether a left checkbox is checked or not

IS_RIGHT_CHECKBOX_CHECKED = *'is_right_checkbox_checked'*

A boolean field that denotes whether a right checkbox is checked or not

IS_TABLE_SPAN = *'is_table_span'*

IS_TOP_CHECKBOX_CHECKED = *'is_top_checkbox_checked'*

A boolean field that denotes whether a top checkbox is checked or not

JSON_COL = *'rich_doc_json'*

PAGE_CHAR_STARTS = *'page_char_starts'*

A list of character offsets that denote where in the rich_doc_text each page starts

PAGE_DOCS = *'page_docs'*

A list of RichDoc objects where each item corresponds to the RichDoc of a single page, in order

PDF_URL_COL = *'rich_doc_pdf_url'*

The URL/file location of the PDF

PKL_COL = *'rich_doc_pkl'*

Serialized version of the RichDoc object

SPAN_END_CHAR_OFFSET = *'rich_doc_span_end_char_offset'*

The offset from the char_end of the last word in a span

SPAN_END_WORD_ID = *'rich_doc_span_end_word_id'*

The id of the word at the end of the span, relative to all other words in the document

SPAN_NGRAM = *'rich_doc_span_ngram'*

An n-gram of the words in the span; See `Ngram` for more

SPAN_PAGE_ID = *'rich_doc_span_page_id'*

The page # that the span belongs to

SPAN_START_CHAR_OFFSET = *'rich_doc_span_start_char_offset'*

The offset from the char_start of the first word in a span

SPAN_START_WORD_ID = *'rich_doc_span_start_word_id'*

The id of the word at the start of the span, relative to all other words in the document

TABLES = *'tables'*

TABLE_COLUMN_ID = *'table_column_id'*

TEXT_CLUSTERS = *'text_clusters'*

A `TextClusters` object object that captures horizontal clusters of words; requires a *TextClusterer* in the DAG

TEXT_CLUSTER_ID = *'text_cluster_id'*

A global cluster id associated with each text cluster

TEXT_COL = *'rich_doc_text'*

Extracted plain text from the PDF

TEXT_REGION_ID *= 'text_region_id'*

When *LinesFeaturizer* is added, the user can group vertical and horizontal lines into regions; this field denotes which region is associate with a span

snorkelflow.rich_docs.RichDocList

class snorkelflow.rich_docs.RichDocList(*rich_docs*)

Bases: `Serializable`

Serializable wrapper for list of RichDoc's.

__init__

__init__(*rich_docs*)

Methods

`__init__`(rich_docs)	
`deserialize`(serialized)	Deserialize instance to string.
`serialize`()	Serialize instance to string.

deserialize

classmethod deserialize(*serialized*)

Deserialize instance to string.

Return type

`Any`

serialize

serialize()

Serialize instance to string.

Return type

`str`

snorkelflow.rich_docs.Serializable

class snorkelflow.rich_docs.Serializable

Bases: `ABC`

Interface for types that have a custom serialization / deserialization function.

__init__

__init__()

Methods

`__init__`()	
`deserialize`(serialized)	rtype: `Any`
`serialize`()	rtype: `str`

deserialize

abstract classmethod deserialize(*serialized*)

Return type

`Any`

serialize

abstract serialize()

Return type

`str`

snorkelflow.rich_docs.TextClusters

class snorkelflow.rich_docs.TextClusters(*word_to_cluster*, *df_clusters*)

Bases: `Serializable`

Serializable wrapper for horizontal clusters of words.

__init__

__init__(*word_to_cluster*, *df_clusters*)

Methods

`__init__`(word_to_cluster, df_clusters)	
`deserialize`(serialized)	Deserialize instance to string.
`serialize`()	Serialize instance to string.

deserialize

classmethod deserialize(*serialized*)

Deserialize instance to string.

Return type

`Any`

serialize

serialize()

Serialize instance to string.

Return type

`str`

snorkelflow.sdk

(Beta) Object-oriented SDK for Snorkel Flow

Classes

Batch(name, uid, dataset_uid, label_schemas, ...)	The Batch object represents an annotation batch in Snorkel Flow.
Dataset(name, uid, mta_enabled)	The Dataset object represents a dataset in Snorkel Flow.
Deployment(deployment_uid, **kwargs)	The Deployment object represents a deployment in Snorkel Flow.
ExternalModel(external_model_name[, ...])	
ExternalModelTrainer([column_mappings, ...])	The ExternalModelTrainer class provides methods to finetune a 3rd party model.
FTDataset(df, dataset_uid, label_schema_uid, ...)	
FineTuningApp(app_uid, model_node_uid, ...)	
LabelSchema(name, uid, dataset_uid, ...[, ...])	The LabelSchema object represents a label schema in Snorkel Flow.
MLflowDeployment(deployment_uid, **kwargs)	MLflowDeployment class represents a deployment of a Snorkel Flow application as an MLflow model.
ModelNode(uid, application_uid, config)	ModelNode class represents a model node.
Node(uid, application_uid, config)	The Node object represents atomic data processing units in Snorkel Flow.

OperatorNode(uid, application_uid, config)	OperatorNode class represents a non-model, operator node.
QualityDataset(df, dataset_uid, ...)	
Slice(dataset, slice_uid, name[, ...])	
SnorkelFlowPackageDeployment(deployment_uid, ...)	SnorkelFlowPackageDeployment class represents a deployment of a Snorkel Flow application as a SnorkelFlowPackage.

snorkelflow.sdk.Batch

class snorkelflow.sdk.Batch(*name, uid, dataset_uid, label_schemas, batch_size, ts, x_uids*)

Bases: `object`

The Batch object represents an annotation batch in Snorkel Flow. Currently, this interface only represents Dataset-level (not Node-level) annotation batches.

__init__

__init__(*name, uid, dataset_uid, label_schemas, batch_size, ts, x_uids*)

Create a batch object in-memory with necessary properties. This constructor should not be called directly, and should instead be accessed through the `create()` and `get()` methods.

Parameters

Name	Type	Default	Info
name	`str`		The name of the batch.
uid	`int`		The UID for the batch within Snorkel Flow.
dataset_uid	`int`		The UID for the dataset within Snorkel Flow.
label_schemas	`List[LabelSchema]`		The list of label schemas associated with this batch.
batch_size	`int`		The number of examples in the batch.
ts	`datetime`		The timestamp at which the batch was created.
x_uids	`List[str]`		The UIDs for the examples in the batch.

Methods

`__init__`(name, uid, dataset_uid, ...)	Create a batch object in-memory with necessary properties.
`commit`(source_uid[, label_schema_uids])	Commit a source on a batch as ground truth.
`create`(cls, dataset_uid[, name, assignees, ...])	Create one or more annotation batches for a dataset.
`delete`(batch_uid)	Delete an annotation batch by its UID.
`export`(path[, selected_fields, ...])	Export the batch to a zipped CSV file.
`get`(batch_uid)	Retrieve an annotation batch by its UID.
`get_dataframe`([selected_fields, ...])	Get a pandas DataFrame representation of the batch.
`update`([name, assignees, expert_source_uid])	Update properties of the annotation batch.

Attributes

`batch_size`	The number of examples in the batch.
`dataset_uid`	The UID for the dataset within Snorkel Flow.
`label_schemas`	The list of label schemas associated with this batch.
`name`	The name of the batch.
`ts`	The timestamp at which the batch was created.
`uid`	The UID for the batch within Snorkel Flow.
`x_uids`	The UIDs for the examples in the batch.

commit

commit(*source_uid, label_schema_uids=None*)

Commit a source on a batch as ground truth.

Parameters

Name	Type	Default	Info
source_uid	`int`		The UID for the source on the batch.
label_schema_uids	`Optional[List[int]]`	`None`	The label schema UIDs to commit, defaults to all label schemas if not set.

Return type

`None`

create

classmethod create(*cls, dataset_uid, name=None, assignees=None, label_schemas=None, batch_size=None, num_batches=None, randomize=False, random_seed=123, selection_strategy=None, split=None, x_uids=None, filter_by_x_uids_not_in_batch=False, divide_x_uids_evenly_to_assignees=False*)

Create one or more annotation batches for a dataset.

Typically, Dataset.create_batches() is the recommended entrypoint for creating batches.

Parameters

Name	Type	Defaul
dataset_uid	`int`	

name	`Optional[str]`	`None`
assignees	`Optional[List[int]]`	`None`
label_schemas	`Optional[List[LabelSchema]]`	`None`
batch_size	`Optional[int]`	`None`
num_batches	`Optional[int]`	`None`
randomize	`Optional[bool]`	`False`
random_seed	`Optional[int]`	`123`
selection_strategy	`Optional[SelectionStrategy]`	`None`
split	`Optional[str]`	`None`
x_uids	`Optional[List[str]]`	`None`

filter_by_x_uids_not_in_batch	`Optional[bool]`	`False`
divide_x_uids_evenly_to_assignees	`Optional[bool]`	`False`

Returns

The list of created batches

Return type

List[Batch]

delete

classmethod **delete**(*batch_uid*)

Delete an annotation batch by its UID.

Parameters

Name	Type	Default	Info
batch_uid	`int`		The UID for the batch within Snorkel Flow.

Return type

None

export

export(*path, selected_fields=None, include_annotations=False, include_ground_truth=False, max_rows=10000, csv_delimiter=',', quote_char='"', escape_char='\\\\'*)

Export the batch to a zipped CSV file.

Parameters

Name	Type	Default	Info
path	`Union[str, Path]`		The path to the zipped CSV file. If the path does not end in .zip, it will be appended to the path.
selected_fields	`Optional[List[str]]`	`None`	A list of fields to export. If not set, all fields will be exported.
include_annotations	`bool`	`False`	Whether to include annotations in the export.
include_ground_truth	`bool`	`False`	Whether to include ground truth in the export.
max_rows	`int`	`10000`	The maximum number of rows to export.

csv_delimiter	`str`	`','`	The delimiter to use for CSV fields.
quote_char	`str`	`'"'`	The character to use for quoted fields in the CSV.
escape_char	`str`	`'\\\\'`	The character to use for escaping special characters in the CSV.

Returns

The path to the zipped CSV file

Return type

`pathlib.Path`

get

classmethod **get**(*batch_uid*)

Retrieve an annotation batch by its UID.

Parameters

Name	Type	Default	Info
batch_uid	`int`		The UID for the batch within Snorkel Flow.

Returns

The batch object

Return type

Batch

get_dataframe

get_dataframe(*selected_fields=None, include_annotations=False, include_ground_truth=False, max_rows=10000*)

Get a pandas DataFrame representation of the batch.

Parameters

Name	Type	Default	Info
selected_fields	`Optional[List[str]]`	`None`	A list of fields to include in the DataFrame. If not set, all fields will be included.
include_annotations	`bool`	`False`	Whether to include annotations in the DataFrame.
include_ground_truth	`bool`	`False`	Whether to include ground truth in the DataFrame.
max_rows	`int`	`10000`	The maximum number of rows to include in the DataFrame.

Returns

The pandas DataFrame representation of the batch

Return type

`pd.DataFrame`

update

update(*name=None, assignees=None, expert_source_uid=None*)

Update properties of the annotation batch.

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	The new name of the batch.
assignees	`Optional[List[int]]`	`None`	The user UIDs for the new assignees of the batches.
expert_source_uid	`Optional[int]`	`None`	The UID for the new expert source of the batches.

Return type

`None`

property **batch_size**: *int*

The number of examples in the batch.

property **dataset_uid**: *int*

The UID for the dataset within Snorkel Flow.

property **label_schemas**: *List[LabelSchema]*

The list of label schemas associated with this batch.

property **name**: *str*

The name of the batch.

property **ts**: *datetime*

The timestamp at which the batch was created.

property **uid**: *int*

The UID for the batch within Snorkel Flow.

property **x_uids**: *List[str]*

The UIDs for the examples in the batch.

snorkelflow.sdk.Dataset

class snorkelflow.sdk.Dataset(*name*, *uid*, *mta_enabled*)

Bases: `object`

The Dataset object represents a dataset in Snorkel Flow.

Datasets Quickstart

In this quickstart, we will create a Dataset and upload a file to that Dataset as a data source. We will then show how you might go about ingesting that data into the platform.

We will need the following imports

```
from snorkelflow.sdk import Dataset
import snorkelflow.client as sf
import pandas as pd
ctx = sf.SnorkelFlowContext.from_kwargs()
```

We will begin by creating a new Dataset.

```
>>> contracts_dataset = Dataset.create("contracts-dataset")
Successfully created dataset contracts-dataset with UID 0 in workspace 0
```

Next, we will attempt to save a file to the Dataset as a data source. This file will be in S3. File upload will initially fail because this file contains null values.

```
>>> contracts_dataset.create_datasource("s3://snorkel-contracts-
dataset/dev.parquet", uid_col="uid", split="train")
UserInputError: Errors...
```

In this particular example, we decide we don't care about these rows, so we can use Pandas to edit the file and remove the null values. We can then re-upload the data, this time uploading the DataFrame directly without needing to save it to a file again. In some other cases, you may want to either edit those null cells or fix them in your upstream data pipeline.

```
>>> df = pd.read_parquet("s3://snorkel-contracts-dataset/dev.parquet")
>>> df = df.dropna()
>>> contracts_dataset.create_datasource(df, uid_col="uid",
split="train")
+0.07s Starting data ingestion
+1.85s Ingesting data
+2.05s Data ingestion complete
```

To verify that has worked correctly, we can view this Dataset's data sources.

```
>>> contracts_dataset.datasources
[{'datasource_uid': 668,...}]
```

Dataset Concepts

Datasets

Datasets are how your data is represented in Snorkel Flow. Snorkel Flow projects always begin with a single Dataset. Datasets bring external data into Snorkel Flow and help manage that data once it has been ingested. Datasets are composed of individual chunks of data, called **data sources**, and provides an interface for managing individual data sources.

Data Sources

Data sources are the individual chunks of data that make up a Dataset. A data source can be a file you upload from local storage, a file located in a remote (S3, MinIO, etc.) storage service, or an in-memory Pandas DataFrame. Data sources shouldn't be touched directly, but should be managed by interfacing with their parent Dataset. The best way to deal with data sources is to treat them as blocks of data, which can be added and removed but only occasionally changed. Data sources can be given names during their creation, but are usually referred to using a data source UID, an integer ID assigned to each data source when it is created.

Derived Data Sources

When an application is created using a dataset, Snorkel Flow will create a *derived* data source for each data source in the dataset. Derived data sources are intermediate representations of data that track the lineage of the data as it is being processed and are associated with only one application. Note that some operations, such as changing the split of a data source, don't propagate to any of the derived data source once they are derived, and vice versa. Derived data sources are viewable in the Snorkel Flow UI on the "View Data Sources" button, accessible from the "Develop" screen of an application.

Modifying Data

In general, data sources should be treated as immutable. This means that you should avoid modifying the underlying data source once it has been uploaded. If your goal is to filter out rows, add feature columns, or remove feature columns, you should use an Operator to do so. Alternatively, you can modify your data upstream of Snorkel and create a new Dataset with your edited data.

The Python SDK provides limited support for specific one-off operations on data sources. Sometimes you might need to reformat the data in an existing column to make it compatible with processing logic. In this case, you can use the `dataset.update_datasource_data` method to swap out an existing data source for a new one with the updated data. However, be aware that this is an irreversible change, and updating data in this way is an expensive operation that will require all downstream applications to be refreshed.

Splits

Data sources belong to *splits*. Splits help dictate how the data will be used in the model development process. Data sources allocated to the **train** split will be used for model training and labeling function development. Data sources allocated to the **valid** split will be used to validate models iteratively and to perform error analysis. Data sources allocated to the **test** split will be used to evaluate the final model. Data source splits may be updated as needed, but be aware that model metrics and labeling function performance will change based on how the splits are allocated.

Data Upload Guardrails

When you upload data to Snorkel Flow, it must pass a series of safety checks to ensure that the data is valid and safe to load into the platform. These checks include:

> - **Number of rows**: A single data source should not exceed 10 million rows. If your data source exceeds this limit, you should split it into multiple data sources before uploading.
> - **Column memory**: The average memory usage of a single column must be under 20MB across all columns in your data source. For performance, the average column memory usage should be under 5MB. If your data source exceeds this limit, you should split it into multiple data sources before uploading.

- **Null values**: Snorkel Flow will not permit data to be uploaded if any null values exist in that data source. If you have null values in your data, you might want to clean them up with the Pandas `fillna()` method before uploading.
- **Unique integer index**: Snorkel Flow requires that each data source have a unique integer index column. The values in this index must be unique among all datasources in the Dataset. The values must also be unique, non-negative integers. If your Dataset does not already have this stable index column, you must create one before uploading.
- **Consistent schema**: All data sources in a single Dataset should have the same columns. All columns that are in multiple data sources must have the same type. If you have columns that exist in some data sources but not others, you may see unexpected behavior in downstream tasks.

Fetching UIDs

Methods in the `Dataset` class will sometimes require a UID parameter. This is the unique identifier for the Dataset within Snorkel Flow. The Dataset UID can be retrieved by calling `.uid` on a Dataset object. Data source methods will sometimes require a data source UID, which can be retrieved by printing out the datasources by calling `my_dataset.datasources`. The data source UID is the `datasource_uid` field in the returned dictionary.

__init__

__init__(*name, uid, mta_enabled*)

Create a dataset object in-memory with necessary properties. This constructor should not be called directly, and should instead be accessed through the `create()` and `get()` methods

Parameters

Name	Type	Default	Info
name	`str`		The human-readable name of the dataset. Must be unique within the workspace.

uid	`int`		The unique integer identifier for the dataset within Snorkel Flow.
mta_enabled	`bool`		Whether or not multi-task annotation is enabled for this dataset.

Methods

`__init__`(name, uid, mta_enabled)	Create a dataset object in-memory with necessary properties.
`create`(dataset_name[, enable_mta])	Creates and registers a new Dataset object.
`create_batches`([name, assignees, ...])	Create annotation batches for this dataset.
`create_datasource`(data, uid_col[, name, ...])	Creates a new data source withing the Dataset from either a filepath or a Pandas DataFrame.
`create_label_schema`(name, data_type, ...[, ...])	Create a label schema associated with this dataset.
`delete`(dataset[, force])	Delete a dataset based on the provided identifier
`delete_datasource`(datasource_uid[, force, sync])	Delete a data source.
`get`(dataset)	Fetches an already-created Dataset from Snorkel Flow and returns a Dataset object that can be used to interact with files and data
`get_dataframe`([split, max_rows, ...])	Read the Dataset's data into an in-memory Pandas DataFrame.
`list`()	Get a list of all Datasets.
`update`([name])	Update the metadata for this dataset.

`update_datasource_data`(old_datasource_uid, ...)	This function allows you to replace the data of an existing data source with new data.
`update_datasource_split`(datasource_uid, split)	Change the split of a data source that has already been uploaded to the dataset.

Attributes

`batches`	A list of batches belonging to this Dataset.
`datasources`	A list of data sources and associated metadata belonging to this Dataset.
`label_schemas`	A list of label schemas belonging to this Dataset.
`mta_enabled`	Whether or not multi-task annotation is enabled for this dataset.
`name`	The human-readable name of the dataset.
`uid`	The unique integer identifier for the dataset within Snorkel Flow.

create

classmethod **create**(*dataset_name, enable_mta=True*)

Creates and registers a new Dataset object. A Dataset object organizes and collects files and other sources of data for use in Snorkel Flow. A Dataset is restricted to a particular workspace, so only users in that workspace will be able to access that Dataset. Datasets must be initialized with a unique name

Examples

```
>>> from snorkelflow.sdk import Dataset
>>> my_dataset = Dataset.create(dataset_name="my-dataset")
Successfully created dataset my-dataset with UID 0 in workspace 0
```

Parameters

Name	Type	Default	Info
dataset_name	`str`		A name for the Dataset. This name must be unique within the workspace.
enable_mta	`bool`	`True`	Whether to enable multi-task annotation for this dataset. Enabled by default.

Returns

A Dataset object that can be used to interact with the dataset in Snorkel Flow

Return type

Dataset

create_batches

create_batches(*name=None, assignees=None, label_schemas=None, batch_size=None, num_batches=None, randomize=False, random_seed=123, selection_strategy=None, split=None, x_uids=None, filter_by_x_uids_not_in_batch=False, divide_x_uids_evenly_to_assignees=False*)

Create annotation batches for this dataset.

This is the recommended entrypoint for creating batches.

Parameters

Name	Type	Defaul
name	`Optional[str]`	`None`
assignees	`Optional[List[int]]`	`None`

label_schemas	`Optional[List[LabelSchema]]`	`None`
batch_size	`Optional[int]`	`None`
num_batches	`Optional[int]`	`None`
randomize	`Optional[bool]`	`False`
random_seed	`Optional[int]`	`123`
selection_strategy	`Optional[SelectionStrategy]`	`None`
split	`Optional[str]`	`None`
x_uids	`Optional[List[str]]`	`None`
filter_by_x_uids_not_in_batch	`Optional[bool]`	`False`
divide_x_uids_evenly_to_assignees	`Optional[bool]`	`False`

Returns

The list of created batches

Return type

List[Batch]

create_datasource

create_datasource(*data, uid_col, name=None, split='train', sync=True, run_checks=True*)

Creates a new data source withing the Dataset from either a filepath or a Pandas DataFrame.

If you provide a filepath: A file can be a CSV or Parquet file that either exists in the local filesystem, or is accessible via an S3-compatible API (such as MinIO, or AWS S3). Files must have a stable integer index column that is unique across all data sources in the dataset.

If you provide a DataFrame: The DataFrame must have a unique integer column that does not contain duplicates across other sources of data. All DataFrame column names must be strings.

The data must pass all validation checks to be registered as a valid data source. If a DataFrame fails to pass all data validation checks, the upload will fail and the data source will not be registered.

Examples

```
>>> from snorkelflow.sdk import Dataset
>>> my_dataset = Dataset.get("my-dataset")
>>> my_dataset.create_datasource("my_data.csv", uid_col="id",
split="train")
+0.07s Starting data ingestion
+1.85s Ingesting data
+2.05s Data ingestion complete
1 # UID of the datasource

>>> my_dataset.create_datasource(df, uid_col="id", name="dataframe-
data", split="train")
+0.07s Starting data ingestion
+1.85s Ingesting data
+2.05s Data ingestion complete
1
```

Parameters

Name	Type	Default	Info
data	`Union[str, DataFrame]`		Either: - A path to a file in the local filesystem, or a path to an S3-compatible API, by default None. If filepath is not provided, a DataFrame must be provided instead - A Pandas DataFrame, by default None. If df is not provided, a filepath must be provided instead.
uid_col	`str`		Name of the UID column for this data. The values in this column must be unique non-negative integers that are not duplicated across files.
name	`Optional[str]`	`None`	The name to give this data source. If not provided, the

			name of the file will be used, by default None. Adding a name is strongly recommended when uploading a DataFrame.
split	`Optional[str]`	`'train'`	The name of the data split this data belongs to, by default Splits.train.
sync	`Optional[bool]`	`True`	Whether execution should be blocked by this function, by default True. Note that Dataset().datasources may not be updated immediately if sync=False.
run_checks	`bool`	`True`	Whether we should run datasource checks. Recommended for safety, by default True.

Returns

If sync is True, returns the integer UID of the datasource. If sync is False, returns a job ID that can be monitored with `sf.poll_job_id`

Return type

`Union[str, int]`

create_label_schema

create_label_schema(*name, data_type, task_type, label_map, multi_label=False, description=None, label_column=None, label_descriptions=None, primary_field=None*)

Create a label schema associated with this dataset.

This is the recommended entrypoint for creating label schemas.

Parameters

Name	Type	Default	Info
name	`str`		The name of the label schema.
data_type	`str`		The data type of the label schema.
task_type	`str`		The task type of the label schema.
label_map	`Union[Dict[str, int], List[str]]`		A dictionary mapping label names to their integer values, or a list of label names.
multi_label	`bool`	`False`	Whether the label schema is a multi-label schema, by default False.
description	`Optional[str]`	`None`	A description of the label schema, by default None.
label_column	`Optional[str]`	`None`	The name of the column that contains the labels, by default None.
label_descriptions	`Optional[Dict[str, str]]`	`None`	A dictionary mapping label names to their descriptions, by default None.

primary_field	`Optional[str]`	`None`	The primary field of the label schema, by default None.

Returns

The label schema object

Return type

LabelSchema

delete

classmethod **delete**(*dataset*, *force=False*)

Delete a dataset based on the provided identifier

The operation will fail if any applications use this Dataset

Examples

```
>>> from snorkelflow.sdk import Dataset
>>> Dataset.delete("my-dataset")
Successfully deleted dataset my-dataset with UID 0.
```

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		Name or UID of the dataset to delete.
force	`bool`	`False`	If True, delete any applications using the Dataset as well.

Return type

`None`

delete_datasource

delete_datasource(*datasource_uid, force=False, sync=True*)

Delete a data source. Calling delete_datasource will fully remove the data source from the dataset.

> ⚠ WARNING
> The operation will not be permitted if any applications are using the data source to avoid breaking downstream applications. If you are sure you want to delete the data source, use the flag `force=True` to override this check. This function may take a while.

Examples

```
>>> from snorkelflow.sdk import Dataset
>>> my_dataset = Dataset.get("my-dataset")
>>> my_dataset.delete_datasource(1)
Successfully deleted datasource with UID 1.
```

Parameters

Name	Type	Default	Info
datasource_uid	`int`		UID of the data source to delete. See all datasources for this dataset by viewing self.datasources.
force	`bool`	`False`	boolean allowing one to force deletion of a datasource even if that datasource has dependent assets (ground truth, annotations, etc), by default false.
sync	`bool`	`True`	Poll job status and block until complete, by default true.

Returns

Optionally returns job_id if sync mode is turned off

Return type

`Optional[str]`

get

classmethod **get**(*dataset*)

Fetches an already-created Dataset from Snorkel Flow and returns a Dataset object that can be used to interact with files and data

Examples

```
>>> from snorkelflow.sdk import Dataset
>>> my_dataset = Dataset.get("my-dataset")
Successfully retrieved dataset my-dataset with UID 0 in workspace
0.
```

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		Either the name or UID of the dataset. A list of all accessible datasets can be retrieved with `Dataset.list()`

Returns

A Dataset object that can be used to interact with files and data in Snorkel Flow.

Return type

Dataset

get_dataframe

get_dataframe(*split=None, max_rows=10, target_columns=None, datasource_uid=None, use_source_index=True*)

Read the Dataset's data into an in-memory Pandas DataFrame. If only a subset of columns are required, they can be specified with `target_columns`. Note that changes to the DataFrame will not be reflected in the Dataset. To change the actual data in the dataset, you must swap out the relevant data sources.

> NOTE
> By default, only 10 rows are read for memory safety. This limit can be increased by setting `max_rows` to a larger value, but this can be computationally intensive and may lead to unstable behavior.

> NOTE
> By default, we will return the original index column name the data source was uploaded with. However, certain SDK workflows might require an internal representation of the index column, such as the `snorkelflow.sdk.Deployment.execute` function. If you run into issues because of this, run `dataset.get_dataframe` with the `use_source_index` parameter set to `False`.

Examples

```
>>> from snorkelflow.sdk import Dataset
>>> my_dataset = Dataset.get("my-dataset")
>>> df = my_dataset.get_dataframe(target_columns=["a", "b"])
<pd.DataFrame object with 10 rows and columns a, b>
>>> df = my_dataset.get_dataframe(datasource_uid=0, max_rows=None)
<pd.DataFrame object with 100 rows and columns a, b, c>
```

Parameters

Name	Type	Default	Info
split	`Optional[str]`	`None`	The data split to load, b default None (all splits) Other options are "trai "valid", and "test".
max_rows	`Optional[int]`	`10`	The maximum number rows to read, by defaul Use `max_rows=None` t

			fetch all rows. Warning setting this to a large v can be computationally intensive and may lead unstable behavior.
target_columns	`Optional[List[str]]`	`None`	A list of desired data columns, in case not al columns are required, default None.
datasource_uid	`Optional[int]`	`None`	Fetch a dataframe from particular `datasource_uid`. A li all datasource UIDs car retrieved with `Dataset().datasou`
use_source_index	`bool`	`True`	If true, returns the inde column that the data source was originally uploaded with. If false, returns the Snorkel Flo internal column name. by default.

Returns

A Pandas DataFrame object displaying the data in this dataset

Return type

`pd.DataFrame`

list

static list()

Get a list of all Datasets. The returned list includes the Dataset UID, the Dataset name, and additional metadata used to keep track of the Dataset's properties.

Examples

```
>>> Dataset.list()
[
    {
        "name": "test-csv-str",
        "uid": 116,
        "datasources": []
    },
    ...
]
```

Returns

List of all dataset objects

Return type

List[Dataset]

update

update(*name=''*)

Update the metadata for this dataset. Only updating the name of this Dataset is currently supported. The new name for the dataset must be unique within the workspace.

Examples

```
>>> from snorkelflow.sdk import Dataset
>>> my_dataset = Dataset.get(dataset="my-dataset")
>>> my_dataset.update(name="my-new-dataset")
Successfully renamed dataset with UID 0 to my-new-dataset
```

Parameters

Name	Type	Default	Info
name	`str`	`''`	The new name for this dataset.

Returns

Confirmation string if this operation was successful

Return type

`str`

update_datasource_data

update_datasource_data(*old_datasource_uid, new_data, sync=True*)

This function allows you to replace the data of an existing data source with new data. This function can be used if you find an error in an existing value in a data source, or if you need to update values due to changes in your upstream data pipeline. This function requires that all row indexes in the new data source match the row indexes of the old data source. Additionally, all columns must have the same name and the same type.

If your goal is to change the number of columns, the number of rows, or the type of a column, you should consider using an Operator instead.

> ⚠ WARNING
> This is a potentially dangerous operation, and may take a while to run. For safety, this will always run data source checks on the new data source. Applications and models that use the data source being replaced may become temporarily unavailable as computations are re-run over the new data, and might report different behavior. If you are unsure how to use this function, contact a Snorkel representative.

Examples

```
>>> from snorkelflow.sdk import Dataset
>>> my_dataset = Dataset.get("my-dataset")
>>> my_dataset.datasources
[{"datasource_uid": 1, "datasource_name": "test.csv", "split":
"train"}]
>>> df = my_dataset.get_dataframe(datasource_uid=1, max_rows=None)
>>> df
|    |   a |   b |   c          |
|  0|   1 |   0 | bad_path.pdf|
>>> df.iloc[0, 2] = "good_path.pdf"
>>> my_dataset.update_datasource_data(1, df)
Successfully replaced data in datasource with UID 1.
```

Parameters

Name	Type	Default	
old_datasource_uid	`int`		The UID of the data source you w data sources for this dataset by v
new_data	`Union[str, DataFrame]`		Either (1) A path to a file in the loc compatible API, by default None. must be provided instead, or (2) A df is not provided, a filepath mus UIDs of the new data must exactl Use `dataset.get_dataframe(data` to see the existing data.
sync	`bool`	`True`	Poll job status and block until all j

Returns

Returns a Job ID that can be polled if sync is False. Otherwise returns None

Return type

`Optional[str]`

Raises

ValueError – If the data provided is neither a valid file path or a valid Pandas DataFrame

update_datasource_split

update_datasource_split(*datasource_uid, split*)

Change the split of a data source that has already been uploaded to the dataset. This will impact how the data source is used in all future applications.

> ⚠ WARNING
> This will only impact the Dataset's data source, and not existing derived data sources. To change the split within applications that have already been created, find the node's derived data source UID by clicking on "Develop" > "View Data Sources" in the Snorkel Flow UI and use the `sf.update_datasource` function.

Examples

```
>>> from snorkelflow.sdk import Dataset
>>> my_dataset = Dataset.get("my-dataset")
>>> my_dataset.datasources
[{"datasource_uid": 1, "datasource_name": "test.csv", "split":
"train"}]
>>> my_dataset.update_datasource_split(1, "train")
[123, 456, 789]
```

Parameters

Name	Type	Default	Info
datasource_uid	`int`		The integer UID corresponding to the data source you wish to update. You can see a list of all data sources for this dataset by viewing self.datasources.

split	`str`		The new split to assign to this data source. Must be one of “train”, “test”, or “valid”.

Returns

Returns a list of model nodes that have been impacted by changing the split.

Return type

`List[int]`

property **batches**: *List[Batch]*

A list of batches belonging to this Dataset.

property **datasources**: *List[Dict[str, Any]]*

A list of data sources and associated metadata belonging to this Dataset.

property **label_schemas**: *List[LabelSchema]*

A list of label schemas belonging to this Dataset.

property **mta_enabled**: *bool*

Whether or not multi-task annotation is enabled for this dataset.

property **name**: *str*

The human-readable name of the dataset.

property **uid**: *int*

The unique integer identifier for the dataset within Snorkel Flow.

snorkelflow.sdk.Deployment

class snorkelflow.sdk.Deployment(*deployment_uid, **kwargs*)

Bases: `ABC`

The Deployment object represents a deployment in Snorkel Flow. Deployments help you deploy your Snorkel Flow applications to production. This class acts an interface between Snorkel Flow and external deployment services. Every deployment in Snorkel Flow is identified by a unique deployment_uid.

To create a deployment, use MLflowDeployment.create method. This will create a deployment in Snorkel Flow and register it with an MLflow Model Registry of your choice. You can then download the deployment artifacts and use the MLflow CLI to deploy it to production. SnorkelFlowPackage Deployment is deprecated and is only supported for backward compatibility. We recommend using MLflowDeployment instead.

This class also provides various methods to manage your deployments, like `Deployment.list`, `Deployment.get`, `Deployment.delete` etc.

A typical workflow for deploying an application to production is as follows:

```
from snorkelflow.sdk import MLflowDeployment, Deployment

my_deployment = MLflowDeployment.create("my-application", "my-
deployment")
my_deployment.execute(df)
```

Follow with MLflow client calls to deploy your workflow to production.

__init__

__init__(*deployment_uid, **kwargs*)

Initializes a Deployment object. This constructor should not be called directly. Instead, use the *Deployment.create* method to create a deployment.

Parameters

Name	Type	Default	Info
deployment_uid	`int`		The unique identifier of the deployment.

Methods

`__init__`(deployment_uid, **kwargs)	Initializes a Deployment object.
`create`(application, name)	Creates a deployment.
`delete`(deployment_uid)	Delete a deployment based on the provided identifier.
`execute`(df)	Execute a deployment on a pandas DataFrame.
`get`(deployment_uid)	Fetches a Deployment object based on the provided identifier.
`list`([application])	List all deployments for an application.
`update`(name)	Update the metadata for the deployment.

create

abstract classmethod **create**(*application, name*)

Creates a deployment. This method is abstract and must be implemented by the child class.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application to be exported.

name	`Optional[str]`		The name of the deployment. If this is not provided, deployment name will be created using the application name. Deployment names are not unique and re-running this method with the same name will create a new deployment.

Return type

`Deployment`

delete

classmethod **delete**(*deployment_uid*)

Delete a deployment based on the provided identifier.

Examples

```
>>> from snorkelflow.sdk import Deployment
>>> Deployment.delete(deployment_uid=123)
Successfully deleted deployment with uid 123
```

Parameters

Name	Type	Default	Info
deployment_uid	`int`		The unique identifier of the deployment.

Return type

`None`

execute

execute(*df*)

Execute a deployment on a pandas DataFrame. This method can be used to test a deployment within Snorkel Flow environment before deploying it to production.

Examples

```
>>> from snorkelflow.sdk import Deployment
>>> deployment = Deployment.get(deployment_uid=123)
>>> deployment.execute(df=pd.DataFrame({"text": ["hello world"]}))
                        context_uid
text  preds  preds_str   probs
    __DATAPOINT_UID
    doc::0                     0                   this is sample
text      1   services    [0.2509882183599772, 0.25098841065990574,
0.24...
    doc::1                     1        this is another text
paragraph      0   employment  [0.25402374467456923,
0.24917708783941298, 0.2...
```

Parameters

Name	Type	Default	Info
df	`DataFrame`		A pandas DataFrame containing the data to execute the deployment on. The DataFrame must contain a column with the same name as the data schema of the deployment.

Returns

A pandas DataFrame containing the output of the deployment. The output will be a DataFrame with the following columns: context_uid, preds, preds_str, probs. The preds column contains the predicted label, preds_str contains the predicted label as a string, and probs contains the predicted probabilities for each label.

Return type

`pd.DataFrame`

get

classmethod **get**(*deployment_uid*)

Fetches a Deployment object based on the provided identifier. This object can be used to interact with the deployment.

Examples

```
>>> from snorkelflow.sdk import Deployment
>>> deployment = Deployment.get(deployment_uid=123)
>>> deployment.name
"my-deployment"
```

Parameters

Name	Type	Default	Info
deployment_uid	`int`		The unique identifier of the deployment.

Returns

A Deployment object that can be used to interact with the deployment.

Return type

Deployment

list

classmethod **list**(*application=None*)

List all deployments for an application. If application is not specified, lists all deployments in user's current workspace. Note that this method can be expensive if there are a large number of deployments in the given workspace.

Examples

```
>>> from snorkelflow.sdk import Deployment
>>> Deployment.list(application="my-application")
```

Parameters

Name	Type	Default	Info
application	`Union[str, int, None]`	`None`	Name or unique identifier of the application to list deployments for.

Returns

A list of Deployment objects.

Return type

`List["Deployment"]`

update

update(*name*)

Update the metadata for the deployment. Currently, the only metadata that can be updated is the name.

Examples

```
>>> from snorkelflow.sdk import Deployment
>>> deployment = Deployment.get(deployment_uid=123)
>>> deployment.update(name="new-name")
>>> deployment.name
"new-name"
```

Parameters

Name	Type	Default	Info
name	`str`		The new name for the deployment.

Return type

`None`

snorkelflow.sdk.ExternalModel

class snorkelflow.sdk.ExternalModel(*external_model_name, column_mappings={'context': FineTuningColumnType.CONTEXT, 'instruction': FineTuningColumnType.INSTRUCTION, 'prompt_prefix': FineTuningColumnType.PROMPT_PREFIX, 'response': FineTuningColumnType.RESPONSE}, finetuning_provider_type=FinetuningProvider.AWS_SAGEMAKER*)

Bases: `object`

__init__

__init__(*external_model_name, column_mappings={'context': FineTuningColumnType.CONTEXT, 'instruction': FineTuningColumnType.INSTRUCTION, 'prompt_prefix': FineTuningColumnType.PROMPT_PREFIX, 'response': FineTuningColumnType.RESPONSE}, finetuning_provider_type=FinetuningProvider.AWS_SAGEMAKER*)

Class to represent a trained 3rd party model. Returns from ExternalModelTrainer.finetune

Parameters

Name	Type	Defa
external_model_name	`str`	
column_mappings	`Dict[str, FineTuningColumnType]`	`{'instruction': <FineTuningColumnTy 'instruction'>, 'co <FineTuningColumnTy 'context'>, 'respor <FineTuningColumnTy 'response'>, 'promp <FineTuningColumnTy 'prompt_prefix'>}`
finetuning_provider_type	`FinetuningProvider`	`<FinetuningProvide 'sagemaker'>`

Methods

`__init__`(external_model_name[, ...])	Class to represent a trained 3rd party model.
`inference`(datasource_uids[, x_uids, ...])	3rd Party Inference using the finetuned model.

inference

inference(*datasource_uids, x_uids=None, generation_config=None, deployment_config=None, prompt_template=None, sync=True*)

3rd Party Inference using the finetuned model.

Parameters

Name	Type	Default	Info
datasource_uids	`List[int]`		The datasource uids t use for inference.
x_uids	`Optional[List[str]]`	`None`	Optional x_uids for filtering the datasour
generation_config	`Optional[Dict[str, Any]]`	`None`	Optional generation configuration, e.g.: `{ "max_new_tokens 300, "top_k": 50 "top_p": 0. "do_sample" True, "temperature": }`
deployment_config	`Optional[Dict[str, Any]]`	`None`	Optional deployment configuration e.g.:

			`{ "instance_type" "ml.g5.12xlarge "instance_count 1, }`
prompt_template	`Optional[str]`	`None`	Optional prompt template to LLM inference. Defaults to *{instruction}*
sync	`bool`	`True`	Whether to wait for th inference job to comp before returning.

Returns

if (sync=True) returns the model source uid, else returns the job id

Return type

`str`

snorkelflow.sdk.ExternalModelTrainer

class snorkelflow.sdk.ExternalModelTrainer(*column_mappings={'context': FineTuningColumnType.CONTEXT, 'instruction': FineTuningColumnType.INSTRUCTION, 'prompt_prefix': FineTuningColumnType.PROMPT_PREFIX, 'response': FineTuningColumnType.RESPONSE}, finetuning_provider_type=FinetuningProvider.AWS_SAGEMAKER*)

Bases: `object`

The ExternalModelTrainer class provides methods to finetune a 3rd party model.

ExternalModelTrainer Quickstart

In this quickstart, we will create a ExternalModelTrainer object, finetune a 3rd party model, and run inference on that model, saving the result as new datasources.

We will need the following imports

```
import pandas as pd

import snorkelflow.client as sf
from snorkelflow.sdk import Dataset
from snorkelflow.sdk.fine_tuning_app import (
    ExternalModel,
    ExternalModelTrainer,
)
from snorkelflow.types.finetuning import (
    FineTuningColumnType,
    FinetuningProvider,
)
```

First set all your AWS secrets, making sure have you created a Sagemaker execution role with AmazonSageMakerFullAccess.

```
>>> AWS_ACCESS_KEY_ID = "aws::finetuning::access_key_id"
>>> AWS_SECRET_ACCESS_KEY = "aws::finetuning::secret_access_key"
>>> SAGEMAKER_EXECUTION_ROLE = "aws::finetuning::sagemaker_execution_role"
>>> FINETUNING_AWS_REGION = "aws::finetuning::region"
>>> sf.set_secret(AWS_ACCESS_KEY_ID, "<YOUR_ACCESS_KEY>", secret_store='local_store',
workspace_uid=1, kwargs=None)
>>> sf.set_secret(AWS_SECRET_ACCESS_KEY, "<YOUR_SECRET_KEY>", secret_store='local_store',
workspace_uid=1, kwargs=None)
>>> sf.set_secret(SAGEMAKER_EXECUTION_ROLE, "arn:aws:iam::<YOUR_EXECUTION_ROLE>",
secret_store='local_store', workspace_uid=1, kwargs=None)
>>> sf.set_secret(FINETUNING_AWS_REGION, "<YOUR_REGION>", secret_store='local_store',
workspace_uid=1, kwargs=None)
```

Define your data and training settings.

```
df = pd.DataFrame({
    "context_uid": ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10"],
    "context": ["", "", "", "", "", "", "", "", "", ""],
    "prompt": [
        "What is the capital of India?",
        "What is the capital of France?",
        "What is the capital of Nigeria?",
        "What is the capital of Germany?",
        "What is the capital of Japan?",
        "What is the capital of Italy?",
        "What is the capital of Brazil?",
        "What is the capital of Spain?",
        "What is the capital of Australia?",
        "What is the capital of Mozambique?",
    ],
    "response": [
        "New Delhi",
        "Paris",
        "Abuja",
        "Berlin",
        "Tokyo",
        "Rome",
        "Brasília",
        "Madrid",
        "Canberra",
        "Maputo"
    ],
})
training_configs = {
    "instance_type": "ml.g5.12xlarge",
    "instance_count": "1"
}
finetuning_configs = {
        "epoch": "1",
        "instruction_tuned": "True",
        "peft_type": "lora",
        "gradient_checkpointing": "False",
    }
column_mapping = {
    "instruction": FineTuningColumnType.INSTRUCTION,
    "response": FineTuningColumnType.RESPONSE,
    "context": FineTuningColumnType.CONTEXT,
}
```

Create a new Dataset and Datasource.

```
>>> ft_dataset = Dataset.create("ExternalModelTrainer-Quickstart")
Successfully created dataset FineTuningModel-Quickstart with UID 0.
>>> datasource_uid = int(ft_dataset.create_datasource(df, uid_col="context_uid", split="train"))
```

Now it's time to check our AWS permissions and kick off a fine-tuning job

```
>>> external_model_trainer = ExternalModelTrainer(
>>>     column_mappings=column_mapping,
>>>     finetuning_provider_type=FinetuningProvider.AWS_SAGEMAKER,
>>> )
>>> external_model_trainer.check_finetuning_provider_authentication()
>>> external_model = external_model_trainer.finetune(
>>>     base_model_id="meta-textgeneration-llama-3-8b",
>>>     base_model_version="2.0.1",
>>>     finetuning_configs=finetuning_configs,
>>>     training_configs=training_configs,
>>>     datasource_uids=[datasource_uid],
>>>     sync=True,
>>> )
>>> assert isinstance(external_model, ExternalModel)  # mypy
```

Now that we have a finetuned model, we can run inference on it.

```
>>> model_source_uid = external_model.inference(
>>>     datasource_uids=[datasource_uid],
>>>     sync=True
>>> )
```

Once the inference job completes, we can inspect the new datasources.

```
>>> datasources = sf.datasources.get_datasources(ft_dataset.uid)
>>> new_datasources = [ds for ds in datasources if ds['source_uid'] == model_source_uid]
>>> new_df = ft_dataset.get_dataframe(datasource_uid=new_datasources[0]['datasource_uid'])
>>> new_df.head()
```

__init__

__init__(*column_mappings={'context': FineTuningColumnType.CONTEXT, 'instruction': FineTuningColumnType.INSTRUCTION, 'prompt_prefix': FineTuningColumnType.PROMPT_PREFIX, 'response': FineTuningColumnType.RESPONSE}, finetuning_provider_type=FinetuningProvider.AWS_SAGEMAKER*)

Class to fine-tune an external model with SnorkelFlow data.

Parameters

Name	Type	Default	
column_mappings	`Dict[str, FineTuningColumnType]`	`{'instruction': <FineTuningColumnType.INSTRUCTION: 'instruction'>, 'context': <FineTuningColumnType.CONTEXT: 'context'>, 'response': <FineTuningColumnType.RESPONSE: 'response'>, 'prompt_prefix': <FineTuningColumnType.PROMPT_PREFIX: 'prompt_prefix'>}`	Mappir FineTur column
finetuning_provider_type	`FinetuningProvider`	`<FinetuningProvider.AWS_SAGEMAKER: 'sagemaker'>`	The ext to Finetur

Methods

`__init__`([column_mappings, ...])	Class to fine-tune an external model with SnorkelFlow data.
`check_finetuning_provider_authentication`()	Check if the user has the necessary permissions to use the finetuning provider.
`finetune`(base_model_id, finetuning_configs, ...)	Finetune a 3rd Party Model

check_finetuning_provider_authentication

check_finetuning_provider_authentication()

Check if the user has the necessary permissions to use the finetuning provider.

Returns

True if the user has the necessary permissions, or raises an error with the missing permissions

Return type

`bool`

finetune

finetune(*base_model_id, finetuning_configs, training_configs, datasource_uids, base_model_version='*', x_uids=None, sync=True*)

Finetune a 3rd Party Model

Parameters

Name	Type	Default	Info
base_model_id	`str`		The id of the base model. See available models: https://docs.aws.amazon.com/sagemaker/latest/dg/ju foundation-models-latest.html
finetuning_configs	`Dict[str, Any]`		The finetuning hyperparameters, e.g.: `{ "epoch": "1", "instruction_tuned": "True", "validation_split_ratio": "0.1", "max_input_length": "1024" }`
training_configs	`Dict[str, Any]`		The training infrastructure configurations, e.g.: `{ "instance_type": "ml.g5.12xlarge", }`
datasource_uids	`List[int]`		The datasource uids to use for finetuning.

base_model_version	`str`	`'*'`	The version of the base model. Defaults to "*" for lates version. See aws docs above for available versions.
x_uids	`Optional[List[str]]`	`None`	Optional x_uids for filtering the datasources.
sync	`bool`	`True`	Whether to wait for the finetuning job to complete be returning.

Return type

`Union`[`ExternalModel`, `str`]

Returns

- *ExternalModel* – if (sync=True) returns an ExternalModel object for running inference
- *str* – if (sync=False) returns the job id without waiting for the job to finish

snorkelflow.sdk.FTDataset

class snorkelflow.sdk.FTDataset(*df*, *dataset_uid*, *label_schema_uid*, *model_node_uid*)

Bases: `object`

__init__

__init__(*df*, *dataset_uid*, *label_schema_uid*, *model_node_uid*)

Methods

`__init__`(df, dataset_uid, label_schema_uid, ...)	
`append`(ft_dataset)	Append the given FTDataset to the current FTDataset.
`create_annotation_batches`([assignees])	Create an annotation batch for the ft dataset.
`export_data`(format, filepath)	Export the data in the FTDataset to the specified format and write to the provided filepath.
`filter`([source_uids, splits, x_uids, ...])	Filter the dataset based on the given filters.
`get_data`()	Get the data associated with the fine tuning dataset.
`get_x_uids`()	Get the x_uids in the FTDataset.
`mix`(mix_on, weights, n_samples[, seed])	Mix the dataset by split, source_uid, or slice based on the given weights, returning up to limit samples.
`sample`(n[, seed])	Sample n samples from the FTDataset.
`save`(name)	Save the FTDataset as a slice.

`set_as_dev_set`()	Resample the x_uids within the FTDataset as the dev set for the fine tuning application.

append

append(*ft_dataset*)

Append the given FTDataset to the current FTDataset.

Parameters

Name	Type	Default	Info
ft_dataset	`FTDataset`		The FTDataset to append to the current FTDataset.

Returns

The appended FTDataset

Return type

FTDataset

create_annotation_batches

create_annotation_batches(*assignees=None*)

Create an annotation batch for the ft dataset. A batch will be created for each split the x_uids in the dataset are a part of.

Parameters

Name	Type	Default	Info
assignees	`Optional[List[int]]`	`None`	The user uids of the assignees of the annotation batch.

Returns

The created annotation batch

Return type

`List[Dict[str, Any]]`

export_data

export_data(*format, filepath*)

Export the data in the FTDataset to the specified format and write to the provided filepath.

Parameters

Name	Type	Default	Info
format	`ExportFormat`		The format to export the data to.
filepath	`str`		The filepath to write the exported data to.

Return type

`None`

filter

filter(*source_uids=None, splits=None, x_uids=None, feature_hashes=None, slices=None, has_gt=None*)

Filter the dataset based on the given filters.

Parameters

Name	Type	Default	Info

source_uids	`Optional[List[int]]`	`None`	The source uids to filter by.
splits	`Optional[List[str]]`	`None`	The splits to filter by.
x_uids	`Optional[List[str]]`	`None`	The x uids to filter by.
feature_hashes	`Optional[List[str]]`	`None`	The feature hashes to filter by.
slices	`Optional[List[Slice]]`	`None`	The slices to filter by, rows within at least one slice will be included.
has_gt	`Optional[bool]`	`None`	Filter by the existence / non-existence of ground truth.

Returns

The filtered dataset

Return type

FTDataset

get_data

get_data()

Get the data associated with the fine tuning dataset.

Returns

The data associated with the fine tuning dataset

Return type

`pd.DataFrame`

get_x_uids

get_x_uids()

Get the x_uids in the FTDataset.

Returns

The x_uids in the FTDataset

Return type

`List[str]`

mix

mix(*mix_on, weights, n_samples, seed=123*)

Mix the dataset by split, source_uid, or slice based on the given weights, returning up to limit samples. We perform stratified sampling to determine the datapoints to include in the returned FTDataset. Datpoints are sampled without replacement. Note that you may receive less than limit datapoints if there is not enough data to sample and exactly respect the weights. If it's not possible to honor the weight distribution, you will see a warning message when we are unable to include datapoints from a given population because the weight is too low. If the subsets of data we are mixing overlap, you may see a higher distribution of data than what is provided by the weights.

Parameters

Name	Type	Default	Info
mix_on	`MixOn`		The attribute to mix the dataset by.

weights	`Dict[Union[str, int], int]`		The weights to use for the mix, e.g.: `{ "slice1": 3, "slice2": 5 } # The resulting data returned would be 3/8ths from "slice1" and 5/8ths from "slice2".`
n_samples	`int`		The number of samples to sample.
seed	`int`	`123`	The seed to use for the random sampling.

Returns

The mixed dataset

Return type

FTDataset

sample

sample(*n, seed=None*)

Sample n samples from the FTDataset.

Parameters

Name	Type	Default	Info
n	`int`		The number of samples to sample.

seed	`Union[int, RandomState, None]`	`None`	The seed to use for the random sampling.

Returns

The sampled dataset

Return type

FTDataset

save

save(*name*)

Save the FTDataset as a slice. Use the *FTDataset.filter* method to restore the FTDataset from the slice.

Parameters

Name	Type	Default	Info
name	`str`		The name of the slice to save.

Returns

The created slice

Return type

Slice

set_as_dev_set

set_as_dev_set()

Resample the x_uids within the FTDataset as the dev set for the fine tuning application. Note that a dev set must only contain x_uids that are in the train set.

Return type

`None`

snorkelflow.sdk.FineTuningApp

class snorkelflow.sdk.FineTuningApp(*app_uid, model_node_uid, dataset_uid, label_schema_uid, workspace_uid, fine_tuning_app_config*)

Bases: `object`

__init__

__init__(*app_uid, model_node_uid, dataset_uid, label_schema_uid, workspace_uid, fine_tuning_app_config*)

Methods

`__init__`(app_uid, model_node_uid, ...)	
`create`(app_name, fine_tuning_app_config)	Create a new fine tuning application with the given name and configuration The dataset, application, label schema will be setup for you.
`create_evaluation_report`([split, ...])	Create an evaluation report for the quality dataset.
`delete`()	Delete the fine tuning application.
`get`(application)	Initialize a FineTuningApp object from an existing fine tuning application previously created with the SDK.
`get_annotation_batches`()	Get the annotation batches associated with the entire fine tuning dataset and label schema.
`get_dataframe`([split, source_uids, x_uids, ...])	Get data from the dataset associated with the fine tuning application with the given filters applied
`get_evaluation_report`(evaluation_report_uid)	Get the evaluation report associated with the given evaluation report uid.

`get_ft_dataset`()	Get the fine tuning dataset associated with the fine tuning application.
`get_quality_dataset`(model_uid)	Create a QualityDataset object from a trained model's predictions.
`get_sources`()	Get the sources within the current workspace.
`import_data`(data, split, source_uid[, name, ...])	Import data into the fine tuning application.
`import_ground_truth`(gt_df, gt_column, ...[, ...])	Import ground truth labels into the fine tuning dataset.
`list_evaluation_reports`()	List the evaluation reports associated with the fine tuning application.
`list_quality_models`()	List the quality models associated with the fine tuning application.
`register_custom_metric`(metric_name, metric_func)	Register a user-defined metric with the FineTuningApp.
`register_metric`(metric_schema)	Register a defined metric with the FineTuningApp.
`register_model_source`(model_name[, metadata])	Register a model source with the given model name and metadata.
`register_source`(source_name, source_type, ...)	Register a source in the platform
`setup_studio`()	Setup the studio for the fine tuning application.
`unregister_metric`(metric_name)	Unregister a metric with the FineTuningApp.

Attributes

`datasource_metadata`	Get metadata about each datasource, include details about the source

create

classmethod **create**(*app_name, fine_tuning_app_config*)

Create a new fine tuning application with the given name and configuration The dataset, application, label schema will be setup for you.

Parameters

Name	Type	Default	Info
app_name	`str`		The name of the fine tuning application.
fine_tuning_app_config	`FineTuningAppConfig`		The configuration of the fine tuning application.

Returns

The fine tuning application object

Return type

FineTuningApp

create_evaluation_report

create_evaluation_report(*split=None, quality_models=None, finetuned_model_sources=None, slices=None*)

Create an evaluation report for the quality dataset.

Parameters

Name	Type	Default	Info
split	`Optional[str]`	`None`	The split of the data to evaluate (if not provided, metrics will be computed for all splits).
quality_models	`Union[List[str], List[int], None]`	`None`	The quality models to evaluate (if not provided, the committed quality model or the most recently trained model will be used, in that order).
finetuned_model_sources	`Union[List[str], List[int], None]`	`None`	The finetuned model sources to evaluate (if not provided, all finetuned models associated with the datasources will be used).
slices	`Union[List[str], List[int], None]`	`None`	The slices to evaluate (if not provided, all slices in the

			given dataset will be evaluated).

Returns

A dictionary containing the evaluation results

Return type

`Dict[str, Any]`

delete

delete()

Delete the fine tuning application. Dataset must be deleted separately.

Return type

`None`

get

classmethod get(*application*)

Initialize a FineTuningApp object from an existing fine tuning application previously created with the SDK.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or uid of the fine tuning application.

Returns

The fine tuning application object

Return type

FineTuningApp

get_annotation_batches

get_annotation_batches()

Get the annotation batches associated with the entire fine tuning dataset and label schema.

Return type

`List[Batch]`

get_dataframe

get_dataframe(*split=None, source_uids=None, x_uids=None, datasource_uids=None*)

Get data from the dataset associated with the fine tuning application with the given filters applied

Parameters

Name	Type	Default	Info
split	`Optional[str]`	`None`	The split of the data to get.
source_uids	`Optional[List[int]]`	`None`	The source uids to filter by.
x_uids	`Optional[List[str]]`	`None`	The x uids to filter by.
datasource_uids	`Optional[List[str]]`	`None`	The datasource uids to filter by.

Return type

`DataFrame`

get_evaluation_report

get_evaluation_report(*evaluation_report_uid*)

Get the evaluation report associated with the given evaluation report uid.

Parameters

Name	Type	Default	Info
evaluation_report_uid	`int`		The unique identifier for the evaluation report to retrieve.

Returns

A dictionary containing the details of the evaluation report.

Return type

`Dict[str, Any]`

get_ft_dataset

get_ft_dataset()

Get the fine tuning dataset associated with the fine tuning application.

Returns

The fine tuning dataset object

Return type

FTDataset

get_quality_dataset

get_quality_dataset(*model_uid*)

Create a QualityDataset object from a trained model's predictions.

Parameters

Name	Type	Default	Info
model_uid	`int`		The unique identifier of the trained model.

Returns

The QualityDataset object.

Return type

QualityDataset

get_sources

get_sources()

Get the sources within the current workspace.

Returns

A list of dictionaries containing the details of the sources.

Return type

`List[Dict[str, Any]]`

import_data

import_data(*data, split, source_uid, name=None, sync=True, refresh_datasources=True, prompt_template=None*)

Import data into the fine tuning application.

Parameters

Name	Type	Default	Info

data	`Union[str, DataFrame]`		A file path or a pandas DataFrame of the data to import into the dataset.
split	`str`		The split of the data.
source_uid	`int`		The source to associate the data with for data lineage.
name	`Optional[str]`	`None`	The name of the data source.
sync	`bool`	`True`	Whether to wait for the ingestion job to complete before returning.
refresh_datasources	`bool`	`True`	Whether to refresh datasources for the downstream model node after ingestion. Can only be set if sync is True.
prompt_template	`Optional[str]`	`None`	The prompt template used when the data was generated.

Returns

The job_id of the ingestion job

Return type

`str`

Notes

If sync is set to False, the method will return immediately after submitting the ingestion job, and refresh_datasources and backfill predictions will not be performed. To ensure all post-ingestion tasks are completed, keep sync as True (default).

import_ground_truth

import_ground_truth(*gt_df, gt_column, join_column, source_uid=None, user_format=True*)

Import ground truth labels into the fine tuning dataset.

Parameters

Name	Type	Default	Info
gt_df	`DataFrame`		The ground truth labels DataFrame.
gt_column	`str`		The column in the ground truth DataFrame that contains the labels.
join_column	`str`		The column to join the gt_df and the fine tuning dataset on to associate the ground truth labels with the fine tuning dataset.
source_uid	`Optional[int]`	`None`	The source uid to associate the annotations with. Defaults to the requesting user's source uid if not set.
user_format	`bool`	`True`	Whether the labels are in the user format or not (the label map string value vs the int value). If true, the label map will be used to convert the labels to their integer values.

Return type

`None`

list_evaluation_reports

list_evaluation_reports()

List the evaluation reports associated with the fine tuning application.

Return type

`List[Dict[str, Any]]`

list_quality_models

list_quality_models()

List the quality models associated with the fine tuning application.

Return type

`DataFrame`

register_custom_metric

register_custom_metric(*metric_name, metric_func, overwrite=False*)

Register a user-defined metric with the FineTuningApp.

Parameters

Name	Type	Default	Info
metric_name	`str`		The display name of this metric.
metric_func	`Callable`		A python function to compute this metric.
overwrite	`Optional[bool]`	`False`	Overwrite a metric of the same name if one already exists.

Returns

id of the registered metric.

Return type

`int`

register_metric

register_metric(*metric_schema*)

Register a defined metric with the FineTuningApp.

Parameters

Name	Type	Default	Info
metric_schema	`MetricSchema`		A MetricSchema object.

Return type

`None`

register_model_source

register_model_source(*model_name, metadata=None*)

Register a model source with the given model name and metadata.

Parameters

Name	Type	Default	Info
model_name	`str`		The name of the model.
metadata	`Optional[ModelSourceMetadata]`	`None`	The metadata associated

			with the model source. If not provided, the provided model name will be used as the model name in the metadata.

Returns

The registered model source.

Return type

`Dict[str, Any]`

register_source

classmethod **register_source**(*source_name*, *source_type*, *user_uid*, *metadata=None*)

Register a source in the platform

Parameters

Name	Type	Default	Info
source_name	`str`		The name of the source.
source_type	`SvcSourceType`		The type of the source.
user_uid	`Optional[int]`		The user uid to associate with the source.

metadata	`Optional[Dict[str, Any]]`	`None`	The metadata to associate with the source.

Returns

The created source

Return type

`Dict[str, Any]`

setup_studio

setup_studio()

Setup the studio for the fine tuning application. This will refresh any stale datasources associated with the fine tuning application.

Return type

`None`

unregister_metric

unregister_metric(*metric_name*)

Unregister a metric with the FineTuningApp.

Parameters

Name	Type	Default	Info
metric_name	`str`		The display name of the metric to unregister.

Return type

`None`

property datasource_metadata*: Dict[int, Any]*

Get metadata about each datasource, include details about the source

snorkelflow.sdk.LabelSchema

class snorkelflow.sdk.LabelSchema(*name, uid, dataset_uid, label_map, description, is_text_label=False*)

Bases: `object`

The LabelSchema object represents a label schema in Snorkel Flow. Currently, this interface only represents Dataset-level (not Node-level) label schemas.

__init__

__init__(*name, uid, dataset_uid, label_map, description, is_text_label=False*)

Create a label schema object in-memory with necessary properties. This constructor should not be called directly, and should instead be accessed through the `create()` and `get()` methods

Parameters

Name	Type	Default	Info
name	`str`		The name of the label schema.
uid	`int`		The UID for the label schema within Snorkel Flow.
dataset_uid	`int`		The UID for the dataset within Snorkel Flow.
label_map	`Dict[str, int]`		The label map of the label schema.
description	`Optional[str]`		The description of the label schema.
is_text_label	`bool`	`False`	Whether the label schema is a text label schema.

Methods

`__init__`(name, uid, dataset_uid, label_map, ...)	Create a label schema object in-memory with necessary properties.
`copy`(name[, description, label_map, ...])	Copy a label schema.
`create`(dataset_uid, name, data_type, ... [, ...])	Create a label schema for a dataset.
`delete`(label_schema)	Delete a label schema by name or UID.
`get`(label_schema)	Retrieve a label schema by name or UID.

Attributes

`dataset_uid`	The UID for the dataset within Snorkel Flow.
`description`	The description of the label schema.
`is_text_label`	Whether the label schema is a text label schema.
`label_map`	The label map of the label schema.
`name`	The name of the label schema.
`uid`	The UID for the label schema within Snorkel Flow.

copy

copy(*name, description=None, label_map=None, label_descriptions=None, updated_label_schema=None*)

Copy a label schema.

Parameters

Name	Type	Default	Info

name	`str`		The name of the new label schema.
description	`Optional[str]`	`None`	The description of the new label schema.
label_map	`Optional[Dict[str, int]]`	`None`	The label map of the new label schema.
label_descriptions	`Optional[Dict[str, str]]`	`None`	The label descriptions of the new label schema.
updated_label_schema	`Optional[Dict[str, str]]`	`None`	The update mapping to apply to the new label schema. This is a dictionary mapping label names for the current label schema to those for the new label schema. If a label for the current label schema is removed, it is mapped to None. Examples: 1. Rename “old_1” to “new_1” and remove “old_2”: {“old_1”: “new_1”, “old_2”: None} 2.

			Merge “old_1” and “old_2” to “new_1”: {“old_1”: “new_1”, “old_2”: “new_1”} 3. Split “old_1” to “new_1” and “new_2”, and keep assets labeled as “old_1” at “new_1”: {“old_1”: “new_1”} 4. Add “new_3”: None (no change to the existing assets).

Returns

The new label schema object

Return type

LabelSchema

create

classmethod **create**(*dataset_uid, name, data_type, task_type, label_map, multi_label=False, description=None, label_column=None, label_descriptions=None, primary_field=None, is_text_label=False*)

Create a label schema for a dataset.

Typically, Dataset.create_label_schema() is the recommended entrypoint for creating label schemas.

Parameters

Name	Type	Default	Info

dataset_uid	`int`		The UID for the dataset within Snorkel Flow.
name	`str`		The name of the label schema.
data_type	`str`		The data type of the label schema.
task_type	`str`		The task type of the label schema.
label_map	`Union[Dict[str, int], List[str]]`		A dictionary mapping label names to their integer values, or a list of label names.
multi_label	`bool`	`False`	Whether the label schema is a multi-label schema, by default False.
description	`Optional[str]`	`None`	A description of the label schema, by default None.
label_column	`Optional[str]`	`None`	The name of the column that contains the labels, by default None.
label_descriptions	`Optional[Dict[str, str]]`	`None`	A dictionary mapping label names to their descriptions, by default None.

primary_field	`Optional[str]`	`None`	The primary field of the label schema, by default None.
is_text_label	`bool`	`False`	Whether the label schema is a text label schema, by default False.

Returns

The label schema object

Return type

LabelSchema

delete

classmethod **delete**(*label_schema*)

Delete a label schema by name or UID.

Parameters

Name	Type	Default	Info
label_schema	`Union[str, int]`		The name or UID of the label schema.

Return type

`None`

get

classmethod **get**(*label_schema*)

Retrieve a label schema by name or UID.

Parameters

Name	Type	Default	Info
label_schema	`Union[str, int]`		The name or UID of the label schema.

Returns

The label schema object

Return type

LabelSchema

Raises

ValueError – If no label schema is found with the given name or UID

property **dataset_uid**: *int*

The UID for the dataset within Snorkel Flow.

property **description**: *str | None*

The description of the label schema.

property **is_text_label**: *bool*

Whether the label schema is a text label schema.

property **label_map**: *Dict[str, int]*

The label map of the label schema.

property **name**: *str*

The name of the label schema.

property **uid**: *int*

The UID for the label schema within Snorkel Flow.

snorkelflow.sdk.MLflowDeployment

class snorkelflow.sdk.MLflowDeployment(*deployment_uid*, ***kwargs*)

Bases: `Deployment`

MLflowDeployment class represents a deployment of a Snorkel Flow application as an MLflow model. This class inherits from the Deployment class and provides additional functionality specific to MLflow deployments. Since MLflow is the only supported deployment type at the moment, this class is the only implementation of the Deployment class. This class is intended to be an interface between Snorkel Flow and MLflow.

A typical workflow for deploying an application to production is as follows:

```
from snorkelflow.sdk import MLflowDeployment

deployment = MLflowDeployment.create("my-application", "my-deployment")
deployment.execute(df)
deployment.download('/path/to/download')
```

Follow with MLflow client calls to deploy your workflow to production.

If you want to get a deployment by its name, you can use the following code:

```
from snorkelflow.sdk import MLflowDeployment

deployments = MLflowDeployment.list(application="my-application")
matched_deployment = next((deployment for deployment in deployments if
deployment.name == "my-deployment"), None)
if matched_deployment is None:
    print("Deployment not found")
```

__init__

__init__(*deployment_uid*, ***kwargs*)

Initializes a Deployment object. This constructor should not be called directly. Instead, use the *Deployment.create* method to create a deployment.

Parameters

Name	Type	Default	Info
deployment_uid	`int`		The unique identifier of the deployment.

Methods

`__init__`(deployment_uid, **kwargs)	Initializes a Deployment object.
`create`(application[, name, registry_type, ...])	Create a new MLflow deployment from an application.
`delete`(deployment_uid)	Delete a deployment based on the provided identifier.
`download`(local_path[, download_as_zip, ...])	Download an MLflow deployment to the specified local path.
`execute`(df)	Execute a deployment on a pandas DataFrame.
`get`(deployment_uid)	Fetches a Deployment object based on the provided identifier.
`list`([application])	List all deployments for an application.
`register`(tracking_server_uri, *[, ...])	Register an MLflowModel Deployment with an external MLflow model registry.
`update`(name)	Update the metadata for the deployment.

create

classmethod create(*application, name=None, registry_type='INTERNAL', model_registry_uid=None, signature=True, experiment_name=None*)

Create a new MLflow deployment from an application.

Examples

```
>>> from snorkelflow.sdk import MLflowDeployment
>>> my_deployment = MLflowDeployment.create(application="my-app",
name="my-deployment")
{'deployment_uid': <deployment_uid>}
>>> my_deployment.name
"my-deployment"
```

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the exported.
name	`Optional[str]`	`None`	The name of the deploy provided, deployment r created using the appli Deployment names are running this method wi will create a new deplo model to Databricks Ur name should be in the <schema>.<model>".
registry_type	`str`	`'INTERNAL'`	The type of the model r or "EXTERNAL".
model_registry_uid	`Optional[int]`	`None`	The UID of the external registry_type is "INTERN model_registry_uid is n
signature	`bool`	`True`	Whether to add a signa input_schema in the MI True, a signature will be signature will be added
experiment_name	`Optional[str]`	`None`	The name of the experi model under, used the if not provided. This is r model registries such a this must an absolute p "/Users/<username>/<e

Returns

MLflowDeployment object containing information about the deployed application

Return type

MLflowDeployment

download

download(*local_path, download_as_zip=False, overwrite=False*)

Download an MLflow deployment to the specified local path. To see how to use this downloaded package, please go to https://mlflow.org/.

Examples

```
>>> deployment.download(local_path='./my-folder',
download_as_zip=True)
Successfully downloaded deployment to ./my-folder/my-deployment-
name.zip
>>> deployment.download(local_path='./my-folder',
download_as_zip=False)
Successfully downloaded deployment to ./my-folder/my-deployment-
name
```

Parameters

Name	Type	Default	Info
local_path	`str`		The local path to download the deployment to.
download_as_zip	`bool`	`False`	Whether to download the deployment as a zip file. If False, the deployment will be downloaded as a directory.

overwrite	`bool`	`False`	Whether to overwrite the local path if it already exists. If False, an error will be thrown if the path already exists.

Return type

`None`

register

register(*tracking_server_uri*, ***, *experiment_name=None*, *model_name=None*, *mlflow_s3_endpoint_url=None*, *aws_access_key_id=None*, *aws_secret_access_key=None*, *basic_auth_username=None*, *basic_auth_password=None*, *bearer_auth_token=None*, *timeout_seconds=600*)

Register an MLflowModel Deployment with an external MLflow model registry.

Examples

```
>>> from snorkelflow.sdk import MLflowDeployment
>>> deployment = MLflowDeployment.create(application="my-app")
>>>
deployment.register(tracking_server_uri="http://localhost:5000")
<Response [200]>
```

Parameters

Name	Type	Default	Info
tracking_server_uri	`str`		URI of the tracking server to export the deployment to.
experiment_name	`Optional[str]`	`None`	The name of the experiment to register the model under, used default if not provided.
model_name	`Optional[str]`	`None`	The name of the model to register the

			deployment under. Defaults to the parent application name.
mlflow_s3_endpoint_url	`Optional[str]`	`None`	The endpoint of the S3-compatible storage service used by the tracking server.
aws_access_key_id	`Optional[str]`	`None`	The access key for the s3-compatible storage service used by the tracking server.
aws_secret_access_key	`Optional[str]`	`None`	The secret key for the s3-compatible storage service used by the tracking server.
basic_auth_username	`Optional[str]`	`None`	Username needed for access to servers with basic authentication enabled.
basic_auth_password	`Optional[str]`	`None`	Password needed for access to servers with basic authentication enabled.
bearer_auth_token	`Optional[str]`	`None`	Bearer token needed for access to servers with bearer authentication enabled.

timeout_seconds	`int`	`600`	Maximum amount of time to wait for the job to complete before giving up.

Returns

A dictionary describing the model that has been exported to the external service. Resembles a mlflow.model.models.ModelInfo object with extra information on the active Experiment.

Return type

`Dict`

snorkelflow.sdk.ModelNode

class snorkelflow.sdk.ModelNode(*uid*, *application_uid*, *config*)

Bases: `Node`

ModelNode class represents a model node.

__init__

__init__(*uid*, *application_uid*, *config*)

Methods

`__init__`(uid, application_uid, config)	
`get`(node_uid)	Fetches a node by its UID.
`get_comments`([username])	Retreive the comments left on the current model node.
`get_dataframe`([split, columns])	Retrieve the data being passed directly through this node.
`get_ground_truth`([split, user_format])	Retrieve ground truth data for the current model node.
`get_lfs`()	Retrieve a list of currently active labeling functions for the current model node.
`get_model_preds`([model_uid, split, ...])	Retrieve model predictions and probabilities for the current model node and a given model UID.
`get_tags`([is_context])	Retrieve tags put on the current model node.
`get_training_set`(training_set_uid[, split, ...])	Retrieve a training set for this model node, specified by the training set UID.

Attributes

`application_uid`	The unique identifier for the application this node belongs to
`config`	Returns the detailed configuration information for this node
`uid`	The unique identifier for this node

get_comments

get_comments(*username=None*)

Retreive the comments left on the current model node. This method will return a Pandas DataFrame whose columns contain the metadata and content for the comment.

Examples

```
>>> my_node.get_comments()
    comment_uid user_uid    x_uid   body    created_at
is_edited
    7           3           doc::1  hello   2023-09-26T17   False
```

Parameters

Name	Type	Default	Info
username	`Optional[str]`	`None`	Optionally, return only a specific user's coments. By default returns all comments.

Returns

A Pandas DataFrame containing the comments left on the model node.

Return type

`pd.DataFrame`

get_dataframe

get_dataframe(*split=None, columns=None*)

Retrieve the data being passed directly through this node. Can be filtered by a split or by a subset of columns (useful for large datasets). The data can also optionally include tag and comment metadata.

This dataframe is not the same as the dataframe returned by `Dataset.get_dataframe()`. While `Dataset.get_dataframe()` returns the source data, the dataframe returned by `Node.get_dataframe()` has also undergone all the preprocessing/DAG transformations up to this point in the processing pipeline.

Parameters

Name	Type	Default	Info
split	`Optional[str]`	`None`	Optionally restrict the data retrieved to a particular split, by default None (i.e., all splits).
columns	`Optional[List[str]]`	`None`	Optionally restrict the columns returned by this function, by default None. Useful for large datasets to significantly speed up retrieval time.

Returns

A dataframe of the data being passed directly through this node, optionally filtered by split and/or columns and indexed by `x_uid`. This DataFrame is the result of all preprocessing in the DAG pipeline up to this point.

Return type

`pd.DataFrame`

get_ground_truth

get_ground_truth(*split=None, user_format=False*)

Retrieve ground truth data for the current model node. Optionally filter by a particular split.

Parameters

Name	Type	Default	Info
split	`Optional[str]`	`None`	Which data split to select, by default None (all splits). Can be one of “dev”, “train”, “valid”, or “test.
user_format	`bool`	`False`	Whether to return the ground truth in a human-readable format, by default False.

Returns

A Pandas DataFrame mapping the data index to the ground truth label. If `user_format` is True, the `label` column will contain human-readable label names.

Return type

`pd.DataFrame`

get_lfs

get_lfs()

Retrieve a list of currently active labeling functions for the current model node.

Examples

```
>>> my_node.get_lfs()
[
    LF(name='LF 1', label=3, templates=[...]),
    LF(name='LF 2', label=2, templates=[...]),
    LF(name='LF 3', label=1, templates=[...]),
]
```

Returns

A list of all currently active labeling functions for the current model node.

Return type

List[LF]

get_model_preds

get_model_preds(*model_uid=None, split=None, is_context=False, user_format=True*)

Retrieve model predictions and probabilities for the current model node and a given model UID. If no model UID is provided, the most recent model's predictions are returned.

Examples

```
>>> my_node.get_model_preds()
            preds   probs
x_uid
doc::994    0       [0.543..., 0.080..., 0.37...]
doc::999    2       [0.327..., 0.201..., 0.4...]
```

Parameters

Name	Type	Default	Info
model_uid	`Optional[int]`	`None`	The UID of a trained model, by default the latest model. All trained models can be seen from

			the “Models” accordion in Developer Studio.
split	`Optional[str]`	`None`	Optionally filter model predictions by split, by default returns predictions for all splits. Splits can be one of “train”, “dev”, “valid”, or “test”.
is_context	`bool`	`False`	When True, retrieves predictions at the document level instead of the span level, by default False. Only applicable for information extraction tasks.
user_format	`bool`	`True`	Whether to return the predictions in a human-readable or compressed integer format, by default True (returning a human-readable format).

Returns

A Pandas DataFrame of model predictions and probabilities, indexed by `x_uid`. If `user_format` is True, the `preds` column will contain human-readable label names.

Return type

`pd.DataFrame`

get_tags

get_tags(*is_context=False*)

Retrieve tags put on the current model node. For information extraction tasks, this method allows for fine-grained control over whether you want to retrieve tags at the document level or at the span level.

Examples

```
>>> my_node.get_tags()
x_uid
doc::10005        [loan-err, new_tag1]
doc::10006                  [new_tag1]
doc::10198             [Key-EMP-error]
Name: tags, dtype: object
```

Parameters

Name	Type	Default	Info
is_context	`bool`	`False`	When True, retrieves tags at the document level instead of the span level, by default False. Only applicable for information extraction tasks.

Returns

A Pandas Series containing the tags put on the model node, indexed by `x_uid` .

Return type

`pd.Series`

get_training_set

get_training_set(*training_set_uid, split=None, user_format=True*)

Retrieve a training set for this model node, specified by the training set UID. Allow allows for filtering the training set by a particular data split.

Examples

```
>>> my_node.get_training_set(1)
           training_set_labels     training_set_probs
doc::100   stock                   [0.030..., 0.024...]
```

Parameters

Name	Type	Default	Info
training_set_uid	`int`		A training set UID, which can be found under the “Models” accordion in Developer Studio.
split	`Optional[str]`	`None`	An optional data split to return predictions for, by default None (all splits). Can be one of “train”, “dev”, “valid”, or “test”.
user_format	`bool`	`True`	Whether to return the predictions in a human-readable or compressed integer format, by default True (returning a human-readable format).

Returns

A Pandas DataFrame of training set predictions and probabilities, indexed by `x_uid`. If `user_format` is True, the `preds` column will contain human-readable label names.

Return type

`pd.DataFrame`

snorkelflow.sdk.Node

class snorkelflow.sdk.Node(*uid, application_uid, config*)

Bases: `ABC`

The Node object represents atomic data processing units in Snorkel Flow.

Nodes Quickstart

```
from snorkelflow.sdk import Node
# Get a Node object by its UID
my_node = Node.get(123)

# Get a dataframe
df = my_node.get_dataframe()
```

Nodes Concepts

Nodes

A Node is a unit of data processing in Snorkel Flow. Nodes are split into two broad categories, *Operators* and *Models*. Operator nodes apply transformations to the data, such as adding a column, modifying a column's values, combining multiple dataframes, or filtering rows. Model nodes are the machine learning hubs in the data pipeline, where you can query and finetune foundation models, explore your data, and train your own models. Nodes are stitched together in a particular order, which dictates how your data flows from your source Dataset all the way to your final desired outputs. The Application DAG (directed acyclic graph) helps visualize the order that the nodes are organized in.

Fetching Data in the Notebook

The Node object is the primary way to fetch data in the Snorkel Flow Notebook. The Node object has a `get_dataframe()` method, which returns a Pandas DataFrame corresponding to what the DAG sees when it is processing data at that point in the graph. This is useful for debugging and understanding how your data is being transformed throughout the data development process. However, since the Notebook environment often has fewer compute resources than the core Snorkel Flow backend, there are some caveats to be aware of when interacting with your data this way. By default, the `get_dataframe()` method will return a maximum of 10 rows of data, to prevent the

Notebook from running out of memory. This safety latch can be manually overridden, but should be done with caution.

__init__

__init__(*uid, application_uid, config*)

Methods

`__init__`(uid, application_uid, config)	
`get`(node_uid)	Fetches a node by its UID.
`get_dataframe`()	Retrieve the data being passed directly through this node.

Attributes

`application_uid`	The unique identifier for the application this node belongs to
`config`	Returns the detailed configuration information for this node
`uid`	The unique identifier for this node

get

classmethod **get**(*node_uid*)

Fetches a node by its UID. Returns either a ModelNode or OperatorNode object, which can be used to fetch and manipulate node-level data.

Parameters

Name	Type	Default	Info

node_uid	`int`		The UID for the particular node. You can find this UID by clicking on the node in the DAG view in Developer Studio, or by looking at the `node_dag` field of the return value of `snorkelflow.client.get_application()`.

Returns

A `Node` object, either an instance of `ModelNode` or `OperatorNode`.

Return type

Node

get_dataframe

abstract **get_dataframe**()

Retrieve the data being passed directly through this node.

This dataframe is not the same as the dataframe returned by `Dataset.get_dataframe()`. While `Dataset.get_dataframe()` returns the source data, the dataframe returned by `Node.get_dataframe()` has also undergone all the preprocessing/DAG transformations up to this point in the processing pipeline.

Returns

A DataFrame of the data being passed directly through this node. This DataFrame is the result of all preprocessing in the DAG pipeline up to this point.

Return type

`pd.DataFrame`

property **application_uid**: *int*

The unique identifier for the application this node belongs to

property **config**: *Dict[str, Any]*

Returns the detailed configuration information for this node

property **uid**: *int*

The unique identifier for this node

snorkelflow.sdk.OperatorNode

class snorkelflow.sdk.OperatorNode(*uid, application_uid, config*)

Bases: `Node`

OperatorNode class represents a non-model, operator node.

__init__

__init__(*uid, application_uid, config*)

Methods

`__init__`(uid, application_uid, config)	
`get`(node_uid)	Fetches a node by its UID.
`get_dataframe`([max_input_rows, ...])	Retrieve the data being passed directly through this node.

Attributes

`application_uid`	The unique identifier for the application this node belongs to
`config`	Returns the detailed configuration information for this node
`uid`	The unique identifier for this node

get_dataframe

get_dataframe(*max_input_rows=10, datasource_uids=None, partition=None*)

Retrieve the data being passed directly through this node. By default, this function will only process a maximum of 10 rows of data, to prevent the Notebook from running out of memory. To override this limit, set `max_input_rows` to a higher value.

This dataframe is not the same as the dataframe returned by `Dataset.get_dataframe()`. While `Dataset.get_dataframe()` returns the source data, the dataframe returned by `Node.get_dataframe()` has also

undergone all the preprocessing/DAG transformations up to this point in the processing pipeline.

Parameters

Name	Type	Default	Info
max_input_rows	`int`	`10`	The number of rows that should be pushed through this node, by default 10.
datasource_uids	`Optional[List[int]]`	`None`	A list of datasource UIDs to process, useful if you have some specific datasources you want to examine, by default None. See the `Dataset` class for more information on fetching a datasource UID.
partition	`Optional[int]`	`None`	A specific file partition to process, by default None. Only applicable if the source dataset files are in a readily partitioned format.

Returns

A DataFrame displaying the results when the source dataset is pushed through this node.

Return type

`pd.DataFrame`

snorkelflow.sdk.QualityDataset

class snorkelflow.sdk.QualityDataset(*df*, *dataset_uid*, *label_schema_uid*, *model_node_uid*, *label_map*)

Bases: `FTDataset`

__init__

__init__(*df*, *dataset_uid*, *label_schema_uid*, *model_node_uid*, *label_map*)

Methods

`__init__`(df, dataset_uid, label_schema_uid, ...)	
`append`(ft_dataset)	Append the given FTDataset to the current FTDataset.
`create_annotation_batches`([assignees])	Create an annotation batch for the ft dataset.
`export_data`(format, filepath)	Export the data in the FTDataset to the specified format and write to the provided filepath.
`filter`([source_uids, splits, x_uids, ...])	Filter the dataset based on the given filters.
`get_data`()	Get the data associated with the fine tuning dataset.
`get_x_uids`()	Get the x_uids in the FTDataset.
`mix`(mix_on, weights, n_samples[, seed])	Mix the dataset by split, source_uid, or slice based on the given weights, returning up to limit samples.
`sample`(n[, seed])	Sample n samples from the FTDataset.
`save`(name)	Save the FTDataset as a slice.

`set_as_dev_set`()	Resample the x_uids within the FTDataset as the dev set for the fine tuning application.

filter

filter(*source_uids=None, splits=None, x_uids=None, feature_hashes=None, slices=None, has_gt=None, labels=None, confidence_threshold=None*)

Filter the dataset based on the given filters.

Parameters

Name	Type	Default	Info
source_uids	`Optional[List[int]]`	`None`	The source uids to filter by.
splits	`Optional[List[str]]`	`None`	The splits to filter by.
x_uids	`Optional[List[str]]`	`None`	The x uids to filter by.
feature_hashes	`Optional[List[str]]`	`None`	The feature hashes to filter by.
slices	`Optional[List[Slice]]`	`None`	The slices to filter by, rows within at least one slice will be included.
has_gt	`Optional[bool]`	`None`	Filter by the existence / non-existence

			of ground truth.
labels	`Optional[List[str]]`	`None`	The labels to filter by.
confidence_threshold	`Optional[float]`	`None`	The confidence threshold to filter by.

Returns

The filtered dataset

Return type

QualityDataset

snorkelflow.sdk.Slice

class snorkelflow.sdk.Slice(*dataset, slice_uid, name, description=None, config=None*)

Bases: `object`

__init__

__init__(*dataset, slice_uid, name, description=None, config=None*)

Create a Slice object in-memory with necessary properties. This constructor should not be called directly, and should instead be accessed through the `create()` and `get()` methods

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		The UID or name for the dataset within Snorkel Flow.
slice_uid	`int`		The UID for the slice within Snorkel Flow.
name	`str`		The name of the slice.
description	`Optional[str]`	`None`	The description of the slice.

Methods

`__init__`(dataset, slice_uid, name[, ...])	Create a Slice object in-memory with necessary properties.
`add_x_uids`(x_uids)	Add datapoints to a slice.
`create`(dataset, name[, description, config])	Create a slice for a dataset.
`get`(dataset, slice)	Retrieve a slice by UID.

`get_x_uids`()	Retrieve the UIDs of the datapoints in the slice.
`list`(dataset)	Retrieve all slices for a dataset.
`remove_x_uids`(x_uids)	Remove datapoints from a slice.
`update`([name, description, config])	Update the slice properties.

Attributes

`dataset_uid`	Return the UID of the dataset that the slice belongs to
`description`	Return the description of the slice
`name`	Return the name of the slice
`slice_uid`	Return the UID of the slice

add_x_uids

add_x_uids(*x_uids*)

Add datapoints to a slice.

Parameters

Name	Type	Default	Info
x_uids	`List[str]`		List of UIDs of the datapoints you want to add to the slice.

Return type

`None`

create

classmethod create(*dataset*, *name*, *description=''*, *config=None*)

Create a slice for a dataset. Slices are use to identify a subset of datapoints in a dataset. You can add datapoints to a slice manually, or if you define a config, you can add datapoints programmatically. Slice membership can contain both manual and programmatic identified datapoints.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		The UID or name for the dataset within Snorkel Flow.
name	`str`		The name of the slice.
description	`str`	`''`	A description of the slice, by default the empty string.
config	`Optional[SliceConfig]`	`None`	A SliceConfig object, by default None, you can reference the schema in the *template* module for constructing this config. This config is used to define the Slicing Function (templates and graph) for the slice, allowing it to programmatically add datapoints to the slice membership.

Returns

The slice object

Return type

Slice

Examples

```
>>> from templates.keyword_template import KeywordTemplateSchema
>>> from snorkelflow.sdk.slices import Slice, SliceConfig
>>> from snorkelflow.utils.graph import DEFAULT_GRAPH
>>> Slice.create(
>>>     dataset=dataset_uid,
>>>     name="slice_name",
>>>     description="description",
>>>     config=SliceConfig(
>>>         templates=[
>>>             KeywordTemplateSchema(
>>>                 field="text_col",
>>>                 keywords=["keyword1", "keyword2"],
>>>                 operator="CONTAINS",
>>>                 case_sensitive=False,
>>>                 tokenize=True,
>>>             )
>>>         ],
>>>         graph=DEFAULT_GRAPH,
>>>     ),
>>> )
```

get

classmethod **get**(*dataset, slice*)

Retrieve a slice by UID.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		The UID or name for the dataset within Snorkel Flow.
slice	`Union[str, int]`		The UID or name of the slice.

Returns

The slice object

Return type

Slice

Raises

ValueError – If no slice is found with the given UID

get_x_uids

get_x_uids()

Retrieve the UIDs of the datapoints in the slice.

Returns

List of UIDs of the datapoints in the slice

Return type

`List[str]`

list

classmethod list(*dataset*)

Retrieve all slices for a dataset.

Parameters

Name	Type	Default	Info
dataset	`Union[str, int]`		The UID or name for the dataset within Snorkel Flow.

Returns

A list of all the slices available for that dataset

Return type

List[Slice]

Raises

ValueError – If no dataset is found with the given id

remove_x_uids

remove_x_uids(*x_uids*)

Remove datapoints from a slice.

Parameters

Name	Type	Default	Info
x_uids	`List[str]`		List of UIDs of the datapoints you want to remove from the slice.

Return type

`None`

update

update(*name=None, description=None, config=None*)

Update the slice properties.

Parameters

Name	Type	Default	Info

name	`Optional[str]`	`None`	The new name for the slice, by default None.
description	`Optional[str]`	`None`	The new description for the slice, by default None.
config	`Optional[SliceConfig]`	`None`	A SliceConfig object with the new configuration for the slice, by default None.

Return type

`None`

property **dataset_uid**: *int*

Return the UID of the dataset that the slice belongs to

property **description**: *str | None*

Return the description of the slice

property **name**: *str*

Return the name of the slice

property **slice_uid**: *int*

Return the UID of the slice

snorkelflow.sdk.SnorkelFlowPackageDeployment

class snorkelflow.sdk.SnorkelFlowPackageDeployment(*deployment_uid*, ***kwargs*)

Bases: `Deployment`

SnorkelFlowPackageDeployment class represents a deployment of a Snorkel Flow application as a SnorkelFlowPackage. This class inherits from the Deployment class.

> NOTE
>
> This class supports backward compatibility for the deprecated SnorkelFlowPackage deployment method. Creating new deployments is not supported. To create a new deployment, use `MLflowDeployment.create` instead.

__init__

__init__(*deployment_uid*, ***kwargs*)

Initializes a Deployment object. This constructor should not be called directly. Instead, use the *Deployment.create* method to create a deployment.

Parameters

Name	Type	Default	Info
deployment_uid	`int`		The unique identifier of the deployment.

Methods

`__init__`(deployment_uid, **kwargs)	Initializes a Deployment object.
`create`(application, name)	This method is not supported for SnorkelFlowPackageDeployment.
`delete`(deployment_uid)	Delete a deployment based on the provided identifier.
`execute`(df)	Execute a deployment on a pandas DataFrame.

`get`(deployment_uid)	Fetches a Deployment object based on the provided identifier.
`list`([application])	List all deployments for an application.
`update`(name)	Update the metadata for the deployment.

create

classmethod create(*application, name*)

This method is not supported for SnorkelFlowPackageDeployment. Please use `MLflowDeployment.create` instead.

Parameters

Name	Type	Default	Info
application	`Union[str, int]`		The name or UID of the application to be exported.
name	`Optional[str]`		The name of the deployment. If this is not provided, deployment name will be created using the application name. Deployment names are not unique and re-running this method with the same name will create a new deployment.

Return type

`SnorkelFlowPackageDeployment`

snorkelflow.studio

Functionality for writing custom labeling functions.

Basic LF decorator

See Code LFs.

Advanced LF decorators

Labeling functions that use common NLP Libraries

To write an LF that just requires a SpaCy or Stanza model, use the `@spacy_labeling_function` or `@stanza_labeling_function` decorators. The decorated function must take `nlp` as an additional argument, which stands for the model.

> **TIP**
> We recommend SpaCy for most cases, since Stanza may be slow to apply.

```
from snorkelflow.studio import spacy_labeling_function

@spacy_labeling_function(name="spacy_companies")
def spacy_companies(x, nlp):
    companies = {"Microsoft", "Yahoo", "Apple", "Google"}
    doc = nlp(x.text)
    for ent in doc.ents:
        if ent.text in companies:
            return "LABEL"
    return "UNKNOWN"

sf.add_code_lf(node, spacy_companies, label="LABEL")
```

```
from snorkelflow.studio import stanza_labeling_function

@stanza_labeling_function(name="stanza_companies")
def stanza_companies(x, nlp):
    companies = {"Microsoft", "Yahoo", "Apple", "Google"}
    doc = nlp(x.text)
    for ent in doc.ents:
        if ent.text in companies:
            return "LABEL"
    return "UNKNOWN"

sf.add_code_lf(node, stanza_companies, label="LABEL")
```

Passing external resources using a resources_fn

When using a very large resource like a SpaCy model, passing it via resources can cause the serialized labeling function to become very large. In those cases, you can pass the resource via a function using the `@resources_fn_labeling_function` decorator. Write a function that creates the resources and returns a dictionary mapping resource names to the resources. Then use the decorator with that function, and the labeling function being decorated can take the resources (denoted by the resource names) as additional arguments. The resources function is only run once, and the results are made available to all invocations of the labeling function.

```
from snorkelflow.studio import resources_fn_labeling_function

# Function to compute a spacy model, and the spacy module itself, and return
them
# as a dictionary.
def get_nlp():
    import spacy
    return {"nlp": spacy.load("en_core_web_sm"), "spacy": spacy}

# Decorate function that takes nlp and spacy (keys from the get_nlp dict
above)
# as additional arguments.
@resources_fn_labeling_function(name="my_other_lf", resources_fn=get_nlp)
def starts_with_noun(x, nlp, spacy):
    doc = nlp(x.txt)
    first_word = doc[0]
    return "LABEL_A" if not first_word.pos == spacy.parts_of_speech.NOUN else
"UNKNOWN"

# Function to download and return a nltk tokenizer and parser.
def get_nltk_tokenizer_parser():
    import nltk
    nltk.download("punkt", download_dir="/tmp/nltk")
    nltk.download("averaged_perceptron_tagger", download_dir="/tmp/nltk")
    return {"tokenize": nltk.word_tokenize, "parse": nltk.pos_tag}

@resources_fn_labeling_function(
    name="check_vbp", resources_fn=get_nltk_tokenizer_parser,
)
def check_vbp(x, tokenize, parse):
    pos_pairs = parse(tokenize(x.text))
    for token, pos in pos_pairs:
        if pos == "VBP":
            return "LABEL"
    return "UNKNOWN"
```

Classes

`resources_fn_labeling_function`([name, ...])	Subclass of **`@labeling_function`** decorator that allows passing resources as functions.
`spacy_labeling_function`([name, resources])	Convenience decorator for spacy based labeling functions.
`stanza_labeling_function`([name, resources])	Convenience decorator for stanza based labeling functions.

snorkelflow.studio.resources_fn_labeling_function

class snorkelflow.studio.resources_fn_labeling_function(*name=None, resources=None, resources_fn=None*)

Bases: `labeling_function`

Subclass of `@labeling_function` decorator that allows passing resources as functions.

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	Name of the LF.
resources	`Optional[Mapping[str, Any]]`	`None`	Labeling resources passed in to `f` via `kwargs`
resources_fn	`Optional[Callable[[], Dict[str, Any]]]`	`None`	A function that takes no arguments and returns a dict of resources, which is passed as kwargs to the underlying function. Expensive dependencies (like spacy nlp) and imports should be passed here to avoid serializing or repeatedly computing them.

Examples

```
>>> from snorkelflow.studio import resources_fn_labeling_function
>>> def get_nlp() -> Dict[str, Any]:
...     import spacy
...     return {"nlp": spacy.load("en_core_web_sm")}
>>> @resources_fn_labeling_function(name="my_lf", resources_fn=get_nlp)
... def many_entities(x, nlp) -> str:
...     return "LABEL" if len(nlp(x).ents) > 5 else "UNKNOWN"
```

__init__

__init__(*name=None, resources=None, resources_fn=None*)

Methods

`__init__`([name, resources, resources_fn])	

snorkelflow.studio.spacy_labeling_function

class snorkelflow.studio.spacy_labeling_function(*name=None, resources=None*)

Bases: `resources_fn_labeling_function`

Convenience decorator for spacy based labeling functions.

The decorated function must have an additional *nlp* argument which represents the spacy en-core-web-sm model.

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	Name of the LF.
resources	`Optional[Mapping[str, Any]]`	`None`	Labeling resources passed in to `f` via `kwargs`

Examples

```
>>> from snorkelflow.studio import spacy_labeling_function
>>> @spacy_labeling_function(name="my_spacy_lf")
... def many_entities(x, nlp) -> int:
...     return 0 if len(nlp(x.txt).ents) > 5 else -1
```

__init__

__init__(*name=None, resources=None*)

Methods

`__init__`([name, resources])	

snorkelflow.studio.stanza_labeling_function

class snorkelflow.studio.stanza_labeling_function(*name=None, resources=None*)

Bases: `resources_fn_labeling_function`

Convenience decorator for stanza based labeling functions.

The decorated function must have an additional *nlp* argument which represents the stanza en model.

Parameters

Name	Type	Default	Info
name	`Optional[str]`	`None`	Name of the LF.
resources	`Optional[Mapping[str, Any]]`	`None`	Labeling resources passed in to `f` via `kwargs`

Examples

```
>>> from snorkelflow.studio import stanza_labeling_function
>>> @stanza_labeling_function(name="my_stanza_lf")
... def many_entities(x, nlp) -> int:
...     return 0 if len(nlp(x.txt).ents) > 5 else -1
```

__init__

__init__(*name=None, resources=None*)

Methods

`__init__`([name, resources])	

snorkelflow.utils

Other general utilities.

Functions

open_file(path[, mode])	Opens a file at the specified path and returns the corresponding file object for user files.
resolve_data_path(path)	Resolve a MinIO path to a local file path.

snorkelflow.utils.open_file

snorkelflow.utils.open_file(*path, mode='r', **kwargs*)

Opens a file at the specified path and returns the corresponding file object for user files.

Currently supports MinIO URLs in the following format: `minio://{bucket}/{key}`, local file paths, and S3 URLs.

New MinIO paths paths are resolved to the current workspace by default, ensuring seamless compatibility. Workspace-scoped paths (e.g., *minio://workspace-1/file.txt*) are also supported if explicitly provided and match the current workspace.

Existing non-workspace-scoped MinIO paths paths will continue to work as expected. However, if a file with the same name is created in a workspace-scoped path, it will take priority moving forward.

Examples

```
>>> # Open a file in MinIO
>>> f = open_file("minio://my-bucket/my-key")
```

```
>>> # Open a file in MinIO outside of the in-platform Notebook
>>> f = open_file(
        "minio://my-bucket/my-key",
        key={MINIO-ACCESS-KEY},
        secret={MINIO-SECRET-KEY},
        client_kwargs={"endpoint_url": {MINIO-URL}}
    )
```

```
>>> # Open a local file
>>> f = open_file("./path/to/file")
```

```
>>> # Open a file in S3
>>> f = open_file("s3://my-bucket/my-key", key={YOUR-S3-ACCESS-KEY},
secret={YOUR-S3-ACCESS-KEY})
```

Parameters

Name	Type	Default	Info
path	`str`		The path of the file to be opened.

mode	`str`	`'r'`	`'w'`, `'rb'`, etc.
kwargs	`Optional[Dict[str, Any]]`		Extra options that make sense to a particular storage connection, e.g. host, port, username, password, etc. These options are passed directly to `fsspec.open`.

Returns

A file object opened in the specified mode

Return type

`OpenFile`

snorkelflow.utils.resolve_data_path

snorkelflow.utils.resolve_data_path(*path*)

Resolve a MinIO path to a local file path.

This method can resolve a MinIO path only if it is called from the in-app notebook.

Parameters

Name	Type	Default	Info
path	`str`		MinIO path (e.g., “minio://path/to/file.parquet”).

Returns

File path

Return type

`str`

Examples

```
>>> import pandas as pd
>>> from snorkelflow.utils import resolve_data_path
>>> minio_path = "minio://path/to/file.parquet"
>>> resolved_path = resolve_data_path(minio_path)
>>> df = pd.read_parquet(resolved_path)
```

templates

LF template schema

Template-based LFs can be written in notebook and added to the core platform like below:

```
from templates import KeywordTemplateSchema
keyword_template = KeywordTemplateSchema(
    field = "text",
    keywords = ["keyword1", "keyword2"],
    operator = "CONTAINS",
)

from snorkelflow.lfs import LF
lf = LF(name="my_lf", label=0, templates=[keyword_template.to_dict()])

sf.add_lf(node, lf)
```

Template schema

BoundingPolygonTemplateSchema(*args, **kwargs)	Bounding Polygon template
BoundingRectangleTemplateSchema(*args, **kwargs)	Bounding Rectangle template
CrowdWorkerTemplateSchema(*args, **kwargs)	Crowd Worker template
CustomSVMSequenceTemplateSchema(*args, **kwargs)	Custom SMV Sequence Template
DictTemplateSchema(*args, **kwargs)	Dictionary template
EmbeddingNNTemplateSchema(*args, **kwargs)	Template schema for EmbeddingsNearestNeighbor
ExternalModelTemplateSchema(*args, **kwargs)	External Model template
FieldLengthTemplateSchema(*args, **kwargs)	Field Length template

`FullTextRegexTemplateSchema`(*args, **kwargs)	Full Text Regex template
`FuzzyKeywordTemplateSchema`(*args, **kwargs)	Fuzzy Keyword template
`ImageImageComparatorTemplateSchema`(*args, ...)	Image Image Comparator template
`ImageModelBasedTemplateSchema`(*args, **kwargs)	Model Based template
`ImagePatchComparatorTemplateSchema`(*args, ...)	Image Patch Comparator template
`ImageTextComparatorTemplateSchema`(*args, ...)	Image Text Comparator template
`IpAddressTemplateSchema`(*args, **kwargs)	IP Address template
`KeywordContextTemplateSchema`(*args, **kwargs)	Keyword Context template
`KeywordLocationSchema`(*args, **kwargs)	Keyword Location template
`KeywordTemplateSchema`(*args, **kwargs)	Keyword template
`LexiconOverlapTemplateSchema`(*args, **kwargs)	Lexicon Overlap template
`ModelBasedTemplateSchema`(*args, **kwargs)	Model Based template
`MultiPolarModelBasedTemplateSchema`(*args, ...)	Multipolar Model Based template

`MultiPolarSequenceModelTemplateSchema`(*args, ...)	Multipolar Sequence Model template
`MultipolarCrowdWorkerTemplateSchema`(*args, ...)	Multipolar Crowd Worker template
`MultipolarImageImageComparatorTemplateSchema`(...)	Multipolar Image Image Comparator template
`MultipolarImageModelBasedTemplateSchema`(...)	Model Based template
`MultipolarImagePatchComparatorTemplateSchema`(...)	Multipolar Image Patch Comparator template
`MultipolarImageTextComparatorTemplateSchema`(...)	Multipolar Image Text Comparator template
`NumericComparatorTemplateSchema`(*args, **kwargs)	Numeric Comparator template
`NumericTemplateSchema`(*args, **kwargs)	Numeric template
`PatternMatchTemplateSchema`(*args, **kwargs)	
`RegexTemplateSchema`(*args, **kwargs)	Regex template
`RichDocBoundingBoxTemplateSchema`(*args, **kwargs)	
`RichDocExpressionTemplateSchema`(*args, **kwargs)	
`SQLQueryTemplateSchema`(*args, **kwargs)	SQL Query template

SequenceContextTemplateSchema(*args, **kwargs)	Sequence Context template
SequenceEntityDictTemplateSchema(*args, **kwargs)	Sequence Entity Dictionary template
SequenceFuzzyKeywordTemplateSchema(*args, ...)	Sequence Fuzzy Keyword template
SequenceKeywordTemplateSchema(*args, **kwargs)	Sequence Keyword template
SequenceModelTemplateSchema(*args, **kwargs)	Sequence Model template
SequenceNERTemplateSchema(*args, **kwargs)	Sequence NER template
SequenceRegexTemplateSchema(*args, **kwargs)	Sequence Regex template
SequenceSpacyPropTemplateSchema(*args, **kwargs)	Sequence Spacy Property template
SequenceSubstringExpansionTemplateSchema(...)	Sequence Substring Expansion template
SequenceTokenTypeTemplateSchema(*args, **kwargs)	Sequence Token Type template
SequenceWordVectorSchema(*args, **kwargs)	Sequence Word Vector template
SpacyOverlapTemplateSchema(*args, **kwargs)	Spacy Overlap template
SpanContentTemplateSchema(*args, **kwargs)	Span Content template

`SpanContextTemplateSchema`(*args, **kwargs)	Span Context template
`SpanFontSizeTemplateSchema`(*args, **kwargs)	
`SpanLocationTemplateSchema`(*args, **kwargs)	Span Location template
`SpanPageTemplateSchema`(*args, **kwargs)	
`SpanRegexAlignmentTemplateSchema`(*args, **kwargs)	
`SpanRegexPositionTemplateSchema`(*args, **kwargs)	
`SpanRegexProximityTemplateSchema`(*args, **kwargs)	
`SpanRegexRowTemplateSchema`(*args, **kwargs)	
`TableOneAttributeSchema`(*args, **kwargs)	Table One Attribute template
`TableTwoAttributeSchema`(*args, **kwargs)	Table Two Attribute template
`TimestampTemplateSchema`(*args, **kwargs)	Timestamp template
`TwoAttributeNumericTemplateSchema`(*args, ...)	Two Attribute Numeric template
`UtteranceContentTemplateSchema`(*args, **kwargs)	Utterance Content template

`UtteranceContextTemplateSchema`(*args, **kwargs)	Utterance Context template
`UtteranceLocationTemplateSchema`(*args, **kwargs)	Utterance Location template
`UtteranceSimilarityTemplateSchema`(*args, ...)	Utterance Similarity template
`WordExpressionTemplateSchema`(*args, **kwargs)	Word Expression template
`WordRegexTemplateSchema`(*args, **kwargs)	Sequence Regex template

templates.BoundingPolygonTemplateSchema

class templates.BoundingPolygonTemplateSchema(**args*, ***kwargs*)

Bounding Polygon template

Parameters

Name	Type	Default	Info
field_1	`str`		Field 1.
field_2	`str`		Field 2.
coordinates	`str`		List of coordinates in a json string. E.g., “[[0, 0], [4, 0], [4, 4]]”.

templates.BoundingRectangleTemplateSchema

class templates.BoundingRectangleTemplateSchema(**args*, ***kwargs*)

Bounding Rectangle template

Parameters

Name	Type	Default	Info
field_1	`str`		Field 1.
field_2	`str`		Field 2.
field_1_min	`float`		Min for Field 1.
field_1_max	`float`		Max for Field 1.
field_2_min	`float`		Min for Field 2.
field_2_max	`float`		Max for Field 2.

templates.CrowdWorkerTemplateSchema

class templates.CrowdWorkerTemplateSchema(**args*, ***kwargs*)

Crowd Worker template

Parameters

Name	Type	Default	Info
filepath	`str`		Path to a file.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.CustomSVMSequenceTemplateSchema

class templates.CustomSVMSequenceTemplateSchema(**args*, ***kwargs*)

Custom SMV Sequence Template

Parameters

Name	Type	Default	Info
model_path	`str`		Path to trained binary SVM model.
feature_field	`str`		Field containing embeddings.
feature_span_field	`str`		Field containing spans that map to embeddings.
min_distance	`float`		Minimum distance.
model_name	`str`		SVM model name.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.DictTemplateSchema

class templates.DictTemplateSchema(**args, **kwargs*)

Dictionary template

Parameters

Name	Type	Default	Info
field	`str`		Field.
combiner	`{"ANY", "ALL", "NONE"}`		Combiner.
filepath	`str`		Path to a file.
case_sensitive	`bool`	`False`	Case sensitive or not.
tokenize	`bool`	`True`	Tokenize or not.
remove_header	`bool`	`False`	If True, remove the first row.
class_index	`int`	`0`	Index of the column containing keywords.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.EmbeddingNNTemplateSchema

class templates.EmbeddingNNTemplateSchema(**args*, ***kwargs*)

Template schema for EmbeddingsNearestNeighbor

Parameters

Name	Type	Default	Info
class_embeddings_path	`str`		Path to a file containing class embeddings.
feature_field	`str`		Name of the field containing the feature / embeddings.
min_distance	`float`		Minimum distance to be considered a match.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.ExternalModelTemplateSchema

class templates.ExternalModelTemplateSchema(**args, **kwargs*)

External Model template

Parameters

Name	Type	Default	Info
model_type	`{"polarity_sentiment", "subjectivity_sentiment", "nltk_polarity"}`		Model type.
field	`str`		Field.
operator	`{">", "<", "=", "!=", ">=", "<="}`		Operator.
threshold	`float`		Threshold.

templates.FieldLengthTemplateSchema

class templates.FieldLengthTemplateSchema(**args, **kwargs*)

Field Length template

Parameters

Name	Type	Default	Info
field	`str`		Field.
operator	`{"=", "!=", "<", "<=", ">", ">="}`		Operator.
unit	`{"chars", "words", "lines", "sentences", "paragraphs"}`		Unit.
value	`int`		Value.

templates.FullTextRegexTemplateSchema

class templates.FullTextRegexTemplateSchema(**args*, ***kwargs*)

Full Text Regex template

Parameters

Name	Type	Default	Info
columns	`List[str]`		List of fields to search.
regex_pattern	`str`		Regex pattern.
case_sensitive	`bool`	`False`	Case sensitive or not.
dot_matches_newline	`bool`	`False`	If True, make a dot . match any character at all, including a newline.
start_end_match_line	`bool`	`False`	If True, the character ^ and $ match at the beginning of the string and at the end of the string, respectively.
strict_location_match	`bool`	`True`	Strict location match or not.

templates.FuzzyKeywordTemplateSchema

class templates.FuzzyKeywordTemplateSchema(**args, **kwargs*)

Fuzzy Keyword template

Parameters

Name	Type	Default	Info
field	`str`		Field.
keywords	`List[str]`		List of keywords.
min_similarity_ratio	`float`		Minimum similarity ratio.
case_sensitive	`bool, False`		Case sensitive or not.

templates.ImageImageComparatorTemplateSchema

class templates.ImageImageComparatorTemplateSchema(**args*, ***kwargs*)

Image Image Comparator template

Parameters

Name	Type	Default	Info
image_embedding_field	`str`		Field containing the image embedding.
image_path	`str`		Image path.
operator	`string`	`>=`	Operator.
min_similarity	`float`	`0.5`	Minimum similarity threshold.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.ImageModelBasedTemplateSchema

class templates.ImageModelBasedTemplateSchema(**args, **kwargs*)

Model Based template

Parameters

Name	Type	Default	Info
model_name	`str`		Model name.
model_label	`str`		Model label.
dirpath	`str`		The path to the stored model in minio.
label_map	`Dict[str, int]`		The label map for the Node that this LF is being applied to.
operator	`str`		Operator.
min_similarity	`float`	`0`	Minimum similarity for model decision function.

templates.ImagePatchComparatorTemplateSchema

class templates.ImagePatchComparatorTemplateSchema(**args, **kwargs*)

Image Patch Comparator template

Parameters

Name	Type	Default	Info
image_embedding_field	`str`		Field containing the image embedding.
image_path	`str`		Image path.
bbox	`List[float]`		Bounding box.
operator	`string`	`>=`	Operator.
min_similarity	`float`	`0.5`	Minimum similarity threshold.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.ImageTextComparatorTemplateSchema

class templates.ImageTextComparatorTemplateSchema(**args, **kwargs*)

Image Text Comparator template

Parameters

Name	Type	Default	Info
image_embedding_field	`str`		Field containing the image embedding.
text_query	`str`		Text query.
operator	`string`	`>=`	Operator.
min_similarity	`float`	`0.5`	Minimum similarity threshold.

templates.IpAddressTemplateSchema

class templates.IpAddressTemplateSchema(**args*, ***kwargs*)

IP Address template

Parameters

Name	Type	Default	Info
field	`str`		Field.
network	`str`		Network (e.g., “2.0.0.0/31” and “64::1182:0/112”).

templates.KeywordContextTemplateSchema

class templates.KeywordContextTemplateSchema(**args, **kwargs*)

Keyword Context template

Label data points if a pattern appears within the specified proximity of the other pattern in a specific data field.

Parameters

Name	Type	Default	Info
field	`str`		Field.
regex_pattern_1	`str`		Regex pattern.
regex_pattern_2	`str`		Other regex pattern.
window_size	`int`		Window size.
window_unit	`{"CHARACTERS", "WORDS"}`		Window unit.
match_direction	`{"left", "right", "left or right"}`		Match direction.
case_sensitive	`bool`	`False`	Case sensitive or not.
tokenize	`bool`	`True`	Tokenize or not.
regex	`bool`	`True`	If True, patterns are treated as regex.

templates.KeywordLocationSchema

class templates.KeywordLocationSchema(**args*, ***kwargs*)

Keyword Location template

Parameters

Name	Type	Default	Info
field	`str`		Field.
pattern	`str`		Pattern.
unit	`{"chars", "words", "lines", "sentences", "paragraphs"}`		Unit.
start	`int`	`0`	Start index.
end	`int`	`sys.maxsize`	End index.
operator	`str`	`">="`	Operator.
frequency	`int`	`1`	Number of occurences.
case_sensitive	`bool`	`False`	Case sensitive or not.
regex	`bool`	`False`	If True, the pattern is treated as a regex.
tokenize	`bool`	`False`	Tokenize or not.

templates.KeywordTemplateSchema

class templates.KeywordTemplateSchema(**args*, ***kwargs*)

Keyword template

Parameters

Name	Type	Default	Info
field	`str`		Field.
keywords	`List[str]`		List of keywords.
operator	`{"CONTAINS", "CONTAINS LINE MATCHING", "EQUALS"}`		Operator.
case_sensitive	`bool`	`False`	Case sensitive or not.
tokenize	`bool`	`True`	Tokenize or not.

templates.LexiconOverlapTemplateSchema

class templates.LexiconOverlapTemplateSchema(**args, **kwargs*)

Lexicon Overlap template

Parameters

Name	Type	Default	Info
field_1	`str`		Field 1.
field_2	`str`		Field 2.
denominator_field	`str`		Denominator field.
operator	`{"=", "!=", "<", "<=", ">", ">="}`		Operator.
value	`float`		Value.
lexicon_dict_path	`str`		Path to a file containing lexicons.
case_sensitive	`bool`	`False`	Case sensitive or not.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.ModelBasedTemplateSchema

class templates.ModelBasedTemplateSchema(**args, **kwargs*)

Model Based template

Parameters

Name	Type	Default	Info
include_fields	`List[str]`		The fields to include in th model featurization.
target_field	`str`		The field containing the model predictions.
model_type	`str`		The specific model used, be contained in: ["zsl_text_match", "zsl_entailment", "masked_lm", "logistic_regression "one_class_svm", "tfidf_logreg", "setfi "sdnet", "sequence_embedd
model_name	`str`		A user specified string na given to the trained mode
value	`int`		The class index this LF sh vote for.
dirpath	`str`		The path to the stored m in minio.
threshold	`float`		The confidence threshold below which this LF shoul abstain.

unique_model_name	`Optional[str]`	`None`	A unique name for the m usually the same as dirpa
is_multilabel	`bool`	`False`	Whether or not the task t LF is being applied to is m label.
multilabel_absent_as_abstain	`bool`	`True`	If this is a multi-label task True if we should replace “absent” votes output by model with “abstain” vote

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that’s used within an LF.

Return type

`List[str]`

templates.MultiPolarModelBasedTemplateSchema

class templates.MultiPolarModelBasedTemplateSchema(**args*, ***kwargs*)

Multipolar Model Based template

Parameters

Name	Type	Default	Info
include_fields	`List[str]`		The fields to include in th model featurization.
target_field	`str`		The field containing the model predictions.
model_type	`str`		The specific model used, be contained in: ["zsl_text_match", "zsl_entailment", "masked_lm", "logistic_regression "one_class_svm", "tfidf_logreg", "setfi "sdnet", "sequence_embedd
model_name	`str`		A user specified string na given to the trained mode
value	`None`		Inherited from ModelBasedTemplateSch but not used.
dirpath	`str`		The path to the stored m in minio.

threshold	`float`		The confidence threshold below which this LF shoul abstain.
unique_model_name	`Optional[str]`	`None`	A unique name for the m usually the same as dirpa
is_multilabel	`bool`	`False`	Whether or not the task t LF is being applied to is m label.
multilabel_absent_as_abstain	`bool`	`True`	If this is a multi-label task True if we should replace "absent" votes output by model with "abstain" vote
inv_label_map	`Dict[str, int]`		A mapping from the user' class labels to the LF's cla labels.

templates.MultiPolarSequenceModelTemplateSchema

class templates.MultiPolarSequenceModelTemplateSchema(**args, **kwargs*)

Multipolar Sequence Model template

Parameters

Name	Type	Default	Info
field	`str`		Field storing the output of a ModelFeaturizer.
label	`None`		Inherited from SequenceModelTemplateSchema but not used.
include_fields	`List[str]`		The fields to include in the model featurization.
model_type	`str`		The specific model used, must be contained in: ["sdnet"]
model_name	`str`		A user specified string name given to the trained model.
dirpath	`str`		The path to the stored model in minio.
unique_model_name	`Optional[str]`	`None`	A unique name for the model, usually the same as dirpath.
inv_label_map	`Dict[str, int]`		A mapping from the user's class labels to the LF's class labels.

templates.MultipolarCrowdWorkerTemplateSchema

class templates.MultipolarCrowdWorkerTemplateSchema(**args, **kwargs*)

Multipolar Crowd Worker template

Parameters

Name	Type	Default	Info
filepath	`str`		Path to a file.
uid_col	`str`		UID column.
label_col	`str`		Label column.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.MultipolarImageImageComparatorTemplateSchema

class templates.MultipolarImageImageComparatorTemplateSchema(**args, **kwargs*)

Multipolar Image Image Comparator template

Parameters

Name	Type	Default	Info
image_embedding_field	`str`		Field containing the image embedding.
image_path	`str`		Image path.
label	`str`		Label.
pos_threshold	`float`	`0.5`	Positive threshold similarity. Datapoints with similarity greater than or equal are voted positive.
neg_threshold	`float`	`-0.5`	Negative threshold similarity. Datapoints with similarity less than or equal are voted negative.
invert_votes	`bool`		Whether to invert the votes.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.MultipolarImageModelBasedTemplateSchema

class templates.MultipolarImageModelBasedTemplateSchema(**args, **kwargs*)

Model Based template

Parameters

Name	Type	Default	Info
model_name	`str`		Model name.
model_label	`str`		Model label.
label	`str`		Label.
dirpath	`str`		The path to the stored model in minio.
label_map	`Dict[str, int]`		The label map for the Node that this LF is being applied to.
pos_threshold	`float`	`0.5`	Positive threshold for model decision function. Datapoints with decision function greater than or equal are voted positive.
neg_threshold	`float`	`-0.5`	Negative threshold for model decision function. Datapoints with decision function less than or equal are voted negative.
invert_votes	`bool`		Whether to invert the votes.

templates.MultipolarImagePatchComparatorTemplateSchema

class templates.MultipolarImagePatchComparatorTemplateSchema(**args*, ***kwargs*)

Multipolar Image Patch Comparator template

Parameters

Name	Type	Default	Info
image_embedding_field	`str`		Field containing the image embedding.
image_path	`str`		Image path.
bbox	`List[float]`		Bounding box.
label	`str`		Label.
pos_threshold	`float`	`0.5`	Positive threshold similarity. Datapoints with similarity greater than or equal are voted positive.
neg_threshold	`float`	`-0.5`	Negative threshold similarity. Datapoints with similarity less than or equal are voted negative.
invert_votes	`bool`		Whether to invert the votes.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.MultipolarImageTextComparatorTemplateSchema

class templates.MultipolarImageTextComparatorTemplateSchema(**args*, ***kwargs*)

Multipolar Image Text Comparator template

Parameters

Name	Type	Default	Info
image_embedding_field	`str`		Field containing the image embedding.
text_query	`str`		Text query.
label	`str`		Label.
pos_threshold	`float`	`0.5`	Positive threshold similarity. Datapoints with similarity greater than or equal are voted positive.
neg_threshold	`float`	`-0.5`	Negative threshold similarity. Datapoints with similarity less than or equal are voted negative.
invert_votes	`bool`		Whether to invert the votes.

templates.NumericComparatorTemplateSchema

class templates.NumericComparatorTemplateSchema(**args*, ***kwargs*)

Numeric Comparator template

Parameters

Name	Type	Default	Info
field_1	`str`		Field 1.
field_2	`str`		Field 2.
value	`float`		Value.
operator	`{"+/-", "+", "-"}`		Operator.
scale	`{"unit", "percentage"}`		Scale.

templates.NumericTemplateSchema

class templates.NumericTemplateSchema(**args, **kwargs*)

Numeric template

Parameters

Name	Type	Default	Info
field	`str`		Field.
operator	`{"=", "!=", "<", "<=", ">", ">="}`		Operator.
value	`float`		Value.

templates.PatternMatchTemplateSchema

class templates.PatternMatchTemplateSchema(**args*, ***kwargs*)

templates.RegexTemplateSchema

class templates.RegexTemplateSchema(**args, **kwargs*)

Regex template

Parameters

Name	Type	Default	Info
field	`str`		Field.
span_text_field	`str`	`SpanCols.SPAN_TEXT`	Span field.
regex_pattern	`str`		Regex pattern.
case_sensitive	`bool`	`False`	Case sensitive or not.
dot_matches_newline	`bool`	`False`	If True, make a dot . match any character at all, including a newline.
start_end_match_line	`bool`	`False`	If True, the character ^ and $ match at the beginning of the string and at the end of the string, respectively.
strict_location_match	`bool`	`True`	Strict location match or not.

templates.RichDocBoundingBoxTemplateSchema

class templates.RichDocBoundingBoxTemplateSchema(**args*, ***kwargs*)

templates.RichDocExpressionTemplateSchema

class templates.RichDocExpressionTemplateSchema(**args*, ***kwargs*)

templates.SQLQueryTemplateSchema

class templates.SQLQueryTemplateSchema(**args*, ***kwargs*)

SQL Query template

Parameters

Name	Type	Default	Info
query	`str`		SQL query.

templates.SequenceContextTemplateSchema

class templates.SequenceContextTemplateSchema(**args, **kwargs*)

Sequence Context template

Parameters

Name	Type	Default	Info
field	`str`		Field.
operator	`{"LEFT", "RIGHT", "LEFT OR RIGHT"}`		Operator.
value	`str`		Pattern.
case_sensitive	`bool`	`False`	Case sensitive or not.
token_len	`int`		Length of a token.
case_sensitive			Case sensitive or not.
tokenize	`bool`	`False`	Tokenize or not.
regex	`bool`	`False`	If True, patterns are treated as regex.
include_context	`Optional[bool]`	`False`	).
word_boundary_tokenizer	`Optional[bool]`	`True`	).

templates.SequenceEntityDictTemplateSchema

class templates.SequenceEntityDictTemplateSchema(**args, **kwargs*)

Sequence Entity Dictionary template

Parameters

Name	Type	Default	Info
field	`str`		Field.
entity_dict_path	`str`		Path to a file.
case_sensitive	`bool`	`False`	Case sensitive or not.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.SequenceFuzzyKeywordTemplateSchema

class templates.SequenceFuzzyKeywordTemplateSchema(**args, **kwargs*)

Sequence Fuzzy Keyword template

Parameters

Name	Type	Default	Info
field	`str`		Field.
keywords	`List[str]`		List of keywords.
min_similarity_ratio	`float`		Minimum similarity ratio.
case_sensitive	`bool`	`False`	Case sensitive or not.

templates.SequenceKeywordTemplateSchema

class templates.SequenceKeywordTemplateSchema(**args*, ***kwargs*)

Sequence Keyword template

Parameters

Name	Type	Default	Info
field	`str`		Field.
keywords	`List[str]`		List of keywords.
case_sensitive	`bool`	`False`	Case sensitive or not.
tokenize	`bool`	`True`	Tokenize or not.

templates.SequenceModelTemplateSchema

class templates.SequenceModelTemplateSchema(**args*, ***kwargs*)

Sequence Model template

Parameters

Name	Type	Default	Info
field	`str`		Field storing the output of a ModelFeaturizer.
label	`str`		The label type to extract from the model output field.
include_fields	`List[str]`		The fields to include in the model featurization.
model_type	`str`		The specific model used, must be contained in: ["sdnet"]
model_name	`str`		A user specified string name given to the trained model.
dirpath	`str`		The path to the stored model in minio.
unique_model_name	`Optional[str]`	`None`	A unique name for the model, usually the same as dirpath.

templates.SequenceNERTemplateSchema

class templates.SequenceNERTemplateSchema(**args*, ***kwargs*)

Sequence NER template

Parameters

Name	Type	Default	Info
primary_field	`str`		The primary text field for the dataset or the field that NER was applied on.
ner_field	`str`		The field that contains the NER entities.
keywords	`List[str]`		List of keywords.

templates.SequenceRegexTemplateSchema

class templates.SequenceRegexTemplateSchema(**args*, ***kwargs*)

Sequence Regex template

Parameters

Name	Type	Default	Info
field	`str`		Field.
regex_pattern	`str`		Regex pattern.
case_sensitive	`bool`	`False`	Case sensitive or not.
group_idx	`int`	`0`	Matched group index.

templates.SequenceSpacyPropTemplateSchema

class templates.SequenceSpacyPropTemplateSchema(**args*, ***kwargs*)

Sequence Spacy Property template

Parameters

Name	Type	Default	Info
field	`str`		Field.
keywords	`List[str]`		List of keywords.

templates.SequenceSubstringExpansionTemplateSchema

class templates.SequenceSubstringExpansionTemplateSchema(**args, **kwargs*)

Sequence Substring Expansion template

Parameters

Name	Type	Default	Info
field	`str`		).
operator	`{"MATCHES", "CONTAINS", "STARTS", "ENDS"}`		).
value	`str`		).
case_sensitive	`bool`	`False`	).
regex	`bool`	`False`	).

templates.SequenceTokenTypeTemplateSchema

class templates.SequenceTokenTypeTemplateSchema(**args, **kwargs*)

Sequence Token Type template

Parameters

Name	Type	Default	Info
field	`str`		Field.
token_type	`{"lowercase", "capitalized", "uppercase"}`		Token type.
word_boundary_tokenizer	`bool`	`True`	Tokenize or not.
merge_token	`bool`	`False to maintain backward compatability`	Merge nearby tokens or not.

templates.SequenceWordVectorSchema

class templates.SequenceWordVectorSchema(**args*, ***kwargs*)

Sequence Word Vector template

Parameters

Name	Type	Default	Info
field	`str`		Field.
keywords	`List[str]`		List of keywords.
cosine_similarity	`float`		Cosine similarity.
word_vector_path	`str`		Path to a file containing word embedding vectors.
case_sensitive	`bool`		Case sensitive or not.

artifact_config_keys

static artifact_config_keys()

The names of the fields in the template schema pydantic model that represent paths to files or directories that contain data which back the LF template. For example, paths to a dictionary on disk that's used within an LF.

Return type

`List[str]`

templates.SpacyOverlapTemplateSchema

class templates.SpacyOverlapTemplateSchema(**args, **kwargs*)

Spacy Overlap template

Parameters

Name	Type	Default	Info
field_1	`str`		Field 1.
field_2	`str`		Field 2.
field_denominator	`str`		Field denominator.
operator	`{"=", "!=", "<", "<=", ">", ">="}`		Operator.
value	`float`		Value.
case_sensitive	`bool`	`False`	Case sensitive or not.
lemmatization	`bool`	`True`	Lemmatize or not.
unique_values	`bool, True`		Deduplicate words or not.
spacy_tag_include	`List[str]`	`[]`	List of spacy tags to include.

templates.SpanContentTemplateSchema

class templates.SpanContentTemplateSchema(**args, **kwargs*)

Span Content template

Parameters

Name	Type	Default	Info
span_text_field	`str`		Span field.
span_field	`str`		Field from which spans are extracted.
operator	`{"MATCHES", "CONTAINS", "STARTS", "ENDS"}`		Operator.
value	`str`		Pattern.
regex	`bool`	`False`	If True, the pattern is treated as a regex.
case_sensitive	`bool`	`False`	Case sensitive or not.
tokenize	`bool`	`False`	Tokenize or not.

templates.SpanContextTemplateSchema

class templates.SpanContextTemplateSchema(**args*, ***kwargs*)

Span Context template

Parameters

Name	Type	Default	Info
operator	`{"LEFT", "RIGHT", "LEFT OR RIGHT"}`		Operator.
value	`str`		Value.
window	`int`		The number of words within the span that we will search.
regex	`bool`	`False`	If True, the pattern is treated as a regex.
case_sensitive	`bool`	`False`	Case sensitive or not.
tokenize	`bool`	`False`	Tokenize or not.

templates.SpanFontSizeTemplateSchema

class templates.SpanFontSizeTemplateSchema(**args*, ***kwargs*)

templates.SpanLocationTemplateSchema

class templates.SpanLocationTemplateSchema(**args, **kwargs*)

Span Location template

Parameters

Name	Type	Default	Info
span_text_field	`str`		Span field.
span_field	`str`		Field from which spans are extracted from.
operator	`{"=", "!=", "<", "<=", ">", ">="}`	`">="`	Operator.
frequency	`int`	`1`	Number of occurencens.
unit	`{"chars", "words", "lines", "sentences", "paragraphs"}`	`"paragraphs"`	Unit.
start	`int`	`0`	Start index.
end	`int`	`sys.maxsize`	End index.
case_sensitive	`bool`	`False`	Case sensitive or not.

templates.SpanPageTemplateSchema

class templates.SpanPageTemplateSchema(**args*, ***kwargs*)

templates.SpanRegexAlignmentTemplateSchema

class templates.SpanRegexAlignmentTemplateSchema(**args*, ***kwargs*)

templates.SpanRegexPositionTemplateSchema

class templates.SpanRegexPositionTemplateSchema(**args*, ***kwargs*)

templates.SpanRegexProximityTemplateSchema

class templates.SpanRegexProximityTemplateSchema(**args*, ***kwargs*)

templates.SpanRegexRowTemplateSchema

class templates.SpanRegexRowTemplateSchema(**args*, ***kwargs*)

templates.TableOneAttributeSchema

class templates.TableOneAttributeSchema(**args, **kwargs*)

Table One Attribute template

Parameters

Name	Type	Default	Info
field	`str`		Field.
elem_field	`str`		Element field.
operator	`{"ANY", "ALL"}`		Operator.
values	`List[str]`		List of values.
case_sensitive	`bool`	`False`	Case sensitive or not.

templates.TableTwoAttributeSchema

class templates.TableTwoAttributeSchema(**args*, ***kwargs*)

Table Two Attribute template

Parameters

Name	Type	Default	Info
column	`str`		Column.
field_1	`str`		Field 1.
field_2	`str`		Field 2.
operator_1	`str`		Operator for field 1.
operator_2	`str`		Operator for field 2.
value_1	`str`		Value for field 1.
value_2	`str`		Value for field 2.

templates.TimestampTemplateSchema

class templates.TimestampTemplateSchema(**args*, ***kwargs*)

Timestamp template

Parameters

Name	Type	Default	Info
field	`str`		Field.
start_time	`str`		Start time.
end_time	`str`		End time.
start_dow	`str`		Start day of week.
end_dow	`str`		End day of week.
start_dom	`int`		Start day of month.
end_dom	`int`		End day of month.
start_month	`str`		Start month.
end_month	`str`		End month.
start_year	`str`		Start year.
end_year	`str`		End year.

templates.TwoAttributeNumericTemplateSchema

class templates.TwoAttributeNumericTemplateSchema(**args, **kwargs*)

Two Attribute Numeric template

Parameters

Name	Type	Default	Info
field_1	`str`		Field 1.
field_2	`str`		Field 2.
coefficient_1	`float`		Coefficient for field 1.
coefficient_2	`float`		Coefficient for field 2.
operator	`{"=", "!=", "<", "<=", ">", ">="}`		Operator.
value	`float`		Value.

templates.UtteranceContentTemplateSchema

class templates.UtteranceContentTemplateSchema(**args, **kwargs*)

Utterance Content template

Parameters

Name	Type	Default	Info
operator	`{"MATCHES", "CONTAINS", "STARTS", "ENDS"}`		Operator.
value	`str`		Pattern.
regex	`bool`	`False`	If True, the pattern is treated as a regex.
case_sensitive	`bool`	`False`	Case sensitive or not.
tokenize	`bool`	`False`	Tokenize or not.

templates.UtteranceContextTemplateSchema

class templates.UtteranceContextTemplateSchema(**args, **kwargs*)

Utterance Context template

LF Template specific to conversational use-case. Looks for keywords/regex pattern in specified number of previous or next utterances.

Parameters

Name	Type	Default	Info
operator	`{"BEFORE", "AFTER", "BEFORE OR AFTER"}`		Indicates whether to search before or after the current utterance.
window	`int`	`1`	How many utterances should we restrict the search window to.
speakers	`{"any", "current", "other", or speakername}`		List of speakers whose utterances will be considered for matching.
value	`str`		The value we should be searching for.
regex	`bool`	`False`	If this is True, *value* should be a RegEx Pattern.
case_sensitive	`bool`	`False`	Case sensitive or not.
tokenize	`bool`	`False`	Only can be True if *regex* is False, we will search for an exact word match between value and span context and not match on substrings.

templates.UtteranceLocationTemplateSchema

class templates.UtteranceLocationTemplateSchema(**args*, ***kwargs*)

Utterance Location template

Parameters

Name	Type	Default	Info
value	`str`		Pattern.
start	`int`	`0`	Start index.
end	`int`	`sys.maxsize`	End index.
regex	`bool`	`False`	If True, the pattern is treated as a regex.
case_sensitive	`bool`	`False`	Case sensitive or not.
tokenize	`bool`	`False`	Tokenize or not.

templates.UtteranceSimilarityTemplateSchema

class templates.UtteranceSimilarityTemplateSchema(**args, **kwargs*)

Utterance Similarity template

Parameters

Name	Type	Default	Info
keywords	`List[str]`		List of keywords.
similarity_value	`float`		Similarity.

templates.WordExpressionTemplateSchema

class templates.WordExpressionTemplateSchema(**args*, ***kwargs*)

Word Expression template

Parameters

Name	Type	Default	Info
regex_pattern	`str`		Regex pattern used in the expression.
expression	`str`		Expression used to evaluate words.
case_sensitive	`bool`	`False`	Case sensitive or not.

templates.WordRegexTemplateSchema

class templates.WordRegexTemplateSchema(**args*, ***kwargs*)

Sequence Regex template

Parameters

Name	Type	Default	Info
regex_pattern	`str`		Regex pattern.
case_sensitive	`bool`	`False`	Case sensitive or not.
group_idx	`int`	`0`	Matched group index.