slide

Han Wang

Dec 04, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 What is slide?

slide contains a collection utitily functions that ensure the behaviors of pandas-like dataframes.

CHAPTER

TWO

API REFERENCE

2.1 slide

2.1.1 slide.exceptions

exception slide.exceptions.**SlideCastError**
Bases: *slide.exceptions.SlideException*

Type casting exception

exception slide.exceptions.**SlideException**
Bases: Exception

General Slide level exception

exception slide.exceptions.**SlideIndexIncompatibleError**
Bases: *slide.exceptions.SlideException*

Dataframe index incompatible exception

exception slide.exceptions.**SlideInvalidOperation**
Bases: *slide.exceptions.SlideException*

Invalid operations

2.1.2 slide.utils

class slide.utils.**SlideUtils**(**args*, ***kwds*)
Bases: Generic[slide.utils.TDf, slide.utils.TCol]

A collection of utils for general pandas like dataframes

as_array(*df*, *schema*, *columns=None*, *type_safe=False*)

Parameters

- **df** (*slide.utils.TDf*) –
- **columns** (*Optional[List[str]]*) –
- **type_safe** (*bool*) –

Return type List[List[Any]]

as_array_iterable(*df*, *schema*, *columns=None*, *type_safe=False*)
Convert pandas like dataframe to iterable of rows in the format of list.

Parameters

- **df** (*slide.utils.TDf*) – pandas like dataframe
- **schema** (*pyarrow.lib.Schema*) – schema of the input
- **columns** (*Optional[List[str]]*) – columns to output, None for all columns
- **type_safe** (*bool*) – whether to enforce the types in schema, if False, it will return the original values from the dataframe

Returns iterable of rows, each row is a list

Return type Iterable[List[Any]]

If there are nested types in schema, the conversion can be slower

as_arrow(*df*, *schema*, *type_safe=True*)

Convert the dataframe to pyarrow table

Parameters

- **df** (*slide.utils.TDf*) – pandas like dataframe
- **schema** (*pyarrow.lib.Schema*) – if specified, it will be used to construct pyarrow table, defaults to None
- **type_safe** (*bool*) – check for overflows or other unsafe conversions

Returns pyarrow table

Return type pyarrow.lib.Table

as_pandas(*df*)

Convert the dataframe to pandas dataframe

Returns the pandas dataframe

Parameters **df** (*slide.utils.TDf*) –

Return type pandas.core.frame.DataFrame

binary_arithmetic_op(*col1*, *col2*, *op*)

Binary arithmetic operations +, -, *, /

Parameters

- **col1** (*Any*) – the first column (series or constant)
- **col2** (*Any*) – the second column (series or constant)
- **op** (*str*) – +, -, *, /

Returns the result after the operation (series or constant)

Raises **NotImplementedError** – if op is not supported

Return type Any

All behaviors should be consistent with SQL correspondent operations.

binary_logical_op(*col1*, *col2*, *op*)

Binary logical operations and, or

Parameters

- **col1** (*Any*) – the first column (series or constant)
- **col2** (*Any*) – the second column (series or constant)

- **op** (*str*) – and, or

Returns the result after the operation (series or constant)

Raises **NotImplementedError** – if op is not supported

Return type Any

All behaviors should be consistent with SQL correspondent operations.

case_when(**pairs*, *default=None*)
SQL CASE WHEN

Parameters

- **pairs** (*Tuple[Any, Any]*) – condition and value pairs, both can be either a series or a constant
- **default** (*Optional[Any]*) – default value if none of the conditions satisfies, defaults to None

Returns the final series or constant

Return type Any

This behavior should be consistent with SQL CASE WHEN

cast(*col*, *type_obj*, *input_type=None*)
Cast col to a new type. type_obj must be able to be converted by to_safe_pa_type().

Parameters

- **col** (*Any*) – a series or a constant
- **type_obj** (*Any*) – an objected that can be accepted by to_safe_pa_type()
- **input_type** (*Optional[Any]*) – an objected that is either None or to be accepted by to_safe_pa_type(), defaults to None.

Returns the new column or constant

Return type Any

If input_type is not None, then it can be used to determine the casting behavior. This can be useful when the input is boolean with nulls or strings, where the pandas dtype may not provide the accurate type information.

cast_df(*df*, *schema*, *input_schema=None*)
Cast a dataframe to comply with *schema*.

Parameters

- **df** (*slide.utils.TDf*) – pandas like dataframe
- **schema** (*pyarrow.lib.Schema*) – pyarrow schema to convert to
- **input_schema** (*Optional[pyarrow.lib.Schema]*) – the known input pyarrow schema, defaults to None

Returns converted dataframe

Return type slide.utils.TDf

input_schema is important because sometimes the column types can be different from expected. For example if a boolean series contains Nones, the dtype will be object, without a input type hint, the function can't do the conversion correctly.

`coalesce`(*cols*)
Coalesce multiple series and constants

Parameters **cols** (*List[Any]*) – the collection of series and constants in order

Returns the coalesced series or constant

Return type Any

This behavior should be consistent with SQL `COALESCE`

`cols_to_df`(*cols*, *names=None*)
Construct the dataframe from a list of columns (series)

Parameters

- **cols** (*List[Any]*) – the collection of series or constants, at least one value must be a series
- **names** (*Optional[List[str]]*) – the correspondent column names, defaults to None

Returns the dataframe

Return type slide.utils.TDf

If `names` is not provided, then every series in `cols` must be named. Otherise, `names` must align with `cols`. But whether names have duplications or invalid chars will not be verified by this method

`comparison_op`(*col1*, *col2*, *op*)
Binary comparison <, <=, ==, >, >=

Parameters

- **col1** (*Any*) – the first column (series or constant)
- **col2** (*Any*) – the second column (series or constant)
- **op** (*str*) – <, <=, ==, >, >=

Returns the result after the operation (series or constant)

Raises **NotImplementedError** – if op is not supported

Return type Any

All behaviors should be consistent with SQL correspondent operations.

`drop_duplicates`(*df*)
Select distinct rows from dataframe

raise SlideIndexIncompatibleError("pandas like datafame index can't have name"

)

Returns the result with only distinct rows

Parameters **df** (*slide.utils.TDf*) –

Return type slide.utils.TDf

`empty`(*df*)
Check if the dataframe is empty

Parameters **df** (*slide.utils.TDf*) – pandas like dataframe

Returns if it is empty

Return type bool

ensure_compatible(*df*)
Check whether the datafame is compatible with the operations inside this utils collection, if not, it will raise ValueError

Parameters **df** (*slide.utils.TDf*) – pandas like dataframe

Raises **ValueError** – if not compatible

Return type None

except_df(*df1*, *df2*, *unique*, *anti_indicator_col='__anti_indicator__'*)
Exclude df2 from df1

Parameters

- **df1** (*slide.utils.TDf*) – the first dataframe
- **df2** (*slide.utils.TDf*) – the second dataframe
- **unique** (*bool*) – whether return only unique rows
- **anti_indicator_col** (*str*) –

Returns df1 - df2

Return type slide.utils.TDf

The behavior is not well defined when unique is False

filter_df(*df*, *cond*)
Filter dataframe by a boolean series or a constant

Parameters

- **df** (*slide.utils.TDf*) – the dataframe
- **cond** (*Any*) – a boolean seris or a constant

Returns the filtered dataframe

Return type slide.utils.TDf

Filtering behavior should be consistent with SQL.

get_col_pa_type(*col*)
Get column or constant pyarrow data type

Parameters **col** (*Any*) – the column or the constant

Returns pyarrow data type

Return type pyarrow.lib.DataType

intersect(*df1*, *df2*, *unique*)
Intersect two dataframes

Parameters

- **ndf1** – the first dataframe
- **ndf2** – the second dataframe
- **unique** (*bool*) – whether return only unique rows
- **df1** (*slide.utils.TDf*) –
- **df2** (*slide.utils.TDf*) –

Returns intersected dataframe

Return type slide.utils.TDf

`is_between`(*col*, *lower*, *upper*, *positive*)
Check if a series or a constant is >=`lower` and <=`upper`

Parameters

- **col** (*Any*) – the series or the constant
- **lower** (*Any*) – the lower bound, which can be series or a constant
- **upper** (*Any*) – the upper bound, which can be series or a constant
- **positive** (*bool*) – `is between` or `is not between`

Returns the correspondent boolean series or constant

Return type Any

This behavior should be consistent with SQL `BETWEEN` and `NOT BETWEEN`. The return values can be `True`, `False` and `None`

`is_compatile_index`(*df*)
Check whether the datafame is compatible with the operations inside this utils collection

Parameters **df** (*slide.utils.TDf*) – pandas like dataframe

Returns if it is compatible

Return type bool

`is_in`(*col*, *values*, *positive*)
Check if a series or a constant is in `values`

Parameters

- **col** (*Any*) – the series or the constant
- **values** (*List[Any]*) – a list of constants and series (can mix)
- **positive** (*bool*) – `is in` or `is not in`

Returns the correspondent boolean series or constant

Return type Any

This behavior should be consistent with SQL `IN` and `NOT IN`. The return values can be `True`, `False` and `None`

`is_series`(*obj*)
Check whether is a series type

Parameters **obj** (*Any*) – the object

Returns whether it is a series

Return type bool

`is_value`(*col*, *value*, *positive=True*)
Check if the series or constant is `value`

Parameters

- **col** (*Any*) – the series or constant
- **value** (*Any*) – `None`, `True` or `False`
- **positive** (*bool*) – check `is value` or `is not value`, defaults to True (`is value`)

Raises **NotImplementedError** – if value is not supported

Returns a bool value or a series

Return type Any

join(*ndf1*, *ndf2*, *join_type*, *on*, *anti_indicator_col='__anti_indicator__'*, *cross_indicator_col='__corss_indicator__'*)
Join two dataframes.

Parameters

- **ndf1** (*slide.utils.TDf*) – the first dataframe
- **ndf2** (*slide.utils.TDf*) – the second dataframe
- **join_type** (*str*) – see *parse_join_type()*
- **on** (*List[str]*) – join keys for pandas like `merge` to use
- **anti_indicator_col** (*str*) – temporary column name for anti join, defaults to _ANTI_INDICATOR
- **cross_indicator_col** (*str*) – temporary column name for cross join, defaults to _CROSS_INDICATOR

Raises **NotImplementedError** – if join type is not supported

Returns the joined dataframe

Return type slide.utils.TDf

All join behaviors should be consistent with SQL correspondent joins.

like(*col*, *expr*, *ignore_case=False*, *positive=True*)
SQL `LIKE`

Parameters

- **col** (*Any*) – a series or a constant
- **expr** (*Any*) – a pattern expression
- **ignore_case** (*bool*) – whether to ignore case, defaults to False
- **positive** (*bool*) – `LIKE` or `NOT LIKE`, defaults to True

Returns the correspondent boolean series or constant

Return type Any

This behavior should be consistent with SQL `LIKE`

logical_not(*col*)
Logical `NOT`

All behaviors should be consistent with SQL correspondent operations.

Parameters **col** (*Any*) –

Return type Any

series_to_array(*col*)
Convert a series to numpy array

Parameters **col** (*slide.utils.TCol*) – the series

Returns the numpy array

Return type List[Any]

sql_groupby_apply(*df*, *cols*, *func*, *output_schema=None*, ***kwargs*)

Safe groupby apply operation on pandas like dataframes. In pandas like groupby apply, if any key is null, the whole group is dropped. This method makes sure those groups are included.

Parameters

- **df** (*slide.utils.TDf*) – pandas like dataframe
- **cols** (*List[str]*) – columns to group on, can be empty
- **func** (*Callable[[slide.utils.TDf], slide.utils.TDf]*) – apply function, df in, df out
- **output_schema** (*Optional[pyarrow.lib.Schema]*) – output schema hint for the apply
- **kwargs** (*Any*) –

Returns output dataframe

Return type slide.utils.TDf

The dataframe must be either empty, or with type pd.RangeIndex, pd.Int64Index or pd.UInt64Index and without a name, otherwise, *ValueError* will raise.

to_constant_series(*constant*, *from_series*, *dtype=None*, *name=None*)

Convert a constant to a series with the same index of `from_series`

Parameters

- **constant** (*Any*) – the constant
- **from_series** (*slide.utils.TCol*) – the reference series for index
- **dtype** (*Optional[Any]*) – default data type, defaults to None
- **name** (*Optional[str]*) – name of the series, defaults to None

Returns the series

Return type slide.utils.TCol

to_safe_pa_type(*tp*)

Parameters **tp** (*Any*) –

Return type pyarrow.lib.DataType

to_schema(*df*)

Extract pandas dataframe schema as pyarrow schema. This is a replacement of pyarrow.Schema.from_pandas, and it can correctly handle string type and empty dataframes

Parameters **df** (*slide.utils.TDf*) – pandas dataframe

Raises **ValueError** – if pandas dataframe does not have named schema

Returns pyarrow.Schema

Return type pyarrow.lib.Schema

The dataframe must be either empty, or with type pd.RangeIndex, pd.Int64Index or pd.UInt64Index and without a name, otherwise, *ValueError* will raise.

to_series(*obj*, *name=None*)

Convert an object to series

Parameters

- **obj** (*Any*) – the object
- **name** (*Optional[str]*) – name of the series, defaults to None

Returns the series

Return type slide.utils.TCol

unary_arithmetic_op(*col*, *op*)

Unary arithmetic operator on series/constants

Parameters

- **col** (*Any*) – a series or a constant
- **op** (*str*) – can be + or -

Returns the transformed series or constant

Raises **NotImplementedError** – if op is not supported

Return type Any

All behaviors should be consistent with SQL correspondent operations.

union(*df1*, *df2*, *unique*)

Union two dataframes

Parameters

- **df1** (*slide.utils.TDf*) – the first dataframe
- **df2** (*slide.utils.TDf*) – the second dataframe
- **unique** (*bool*) – whether return only unique rows

Returns unioned dataframe

Return type slide.utils.TDf

slide.utils.**parse_join_type**(*join_type*)

Parse and normalize join type string. The normalization will lower the string, remove all space and _, and then map to the limited options.

Here are the options after normalization: `inner`, `cross`, `left_semi`, `left_anti`, `left_outer`, `right_outer`, `full_outer`.

Parameters **join_type** (*str*) – the raw join type string

Raises **NotImplementedError** – if not supported

Returns the normalized join type string

Return type str

PYTHON MODULE INDEX

S

INDEX