Dataswim Documentation

Release 0.1

synw

Jan 12, 2020

Data io

CHAPTER 1

Load data

Load data from external sources into the main dataframe

1.1 From files

`Df.`**`load_csv`**(*url*, ***kwargs*)
Loads csv data in the main dataframe

Parameters

- **`url`** (*str*) – url of the csv file to load: can be absolute if it starts with `/` or relative if it starts with `./`
- **`kwargs`** – keyword arguments to pass to Pandas `read_csv` function

Example `ds.load_csv("./myfile.csv")`

`Df.`**`load_excel`**(*filepath*, ***kwargs*)
Set the main dataframe with the content of an Excel file

Parameters

- **`filepath`** (*str*) – url of the csv file to load, can be absolute if it starts with `/` or relative if it starts with `./`
- **`kwargs`** – keyword arguments to pass to Pandas `read_excel` function

Example `ds.load_excel("./myfile.xlsx")`

`Df.`**`load_h5`**(*filepath*)
Load a Hdf5 file to the main dataframe

Parameters **`filepath`** (*str*) – url of the csv file to load, can be absolute if it starts with `/` or relative if it starts with `./`

Example `ds.load_h5("./myfile.hdf5")`

Df.**load_json**(*path*, ***kwargs*)

Load data in the main dataframe from json

Parameters

- **filepath** (*str*) – url of the csv file to load, can be absolute if it starts with / or relative if it starts with ./
- **kwargs** – keyword arguments to pass to Pandas read_json function

Example `ds.load_json("./myfile.json")`

1.2 From a database

See the database operations section

CHAPTER 2

Export data

2.1 To files

`Export.`**`to_csv`**(*filepath: str*, *index: bool = False*, ***kwargs*)

Write the main dataframe to a csv file

Parameters

- **`filepath`** (*`str`*) – path of the file to save
- **`index`** – [description], defaults to False
- **`index`** – bool, optional
- **`*args`** – arguments to pass to `pd.to_csv`

Example `ds.to_csv_("myfile.csv", header=false)`

`Export.`**`to_excel`**(*filepath: str*, *title: str*)

Write the main dataframe to an Excell file

Parameters

- **`filepath`** (*`str`*) – path of the Excel file to write
- **`title`** (*`str`*) – Title of the stylesheet

Example `ds.to_excel_("./myfile.xlsx", "My data")`

`Export.`**`to_hdf5`**(*filepath: str*)

Write the main dataframe to Hdf5 file

Parameters **`filepath`** (*`str`*) – path where to save the file

Example `ds.to_hdf5_("./myfile.hdf5")`

CHAPTER 3

Convert data

Convert the main dataframe to various formats

3.1 To web formats

`Export.`**`to_html_`**`()` → str
Convert the main dataframe to html

Returns html data

Return type str

Example `ds.to_html_()`

`Export.`**`to_json_`**`()` → str
Convert the main dataframe to json

Returns json data

Return type str

Example `ds.to_json_()`

`Export.`**`to_javascript_`**`(`*table_name: str = 'data'*`)` → str
Convert the main dataframe to javascript code

Parameters

- **`table_name`** – javascript variable name, defaults to "data"
- **`table_name`** – str, optional

Returns a javascript constant with the data

Return type str

Example `ds.to_javastript_("myconst")`

3.2 To data formats

Export.**to_markdown_**() → str
Convert the main dataframe to markdown

Returns markdown data

Return type str

Example `ds.to_markdown_()`

Export.**to_rst_**() → str
Convert the main dataframe to restructured text

Returns rst data

Return type str

Example `ds.to_rst_()`

Export.**to_records_**() → dict
Returns a list of dictionary records from the main dataframe

Returns a python dictionnary with the data

Return type str

Example `ds.to_records_()`

Export.**to_numpy_**(*table_name: str = 'data'*) → numpy.array
Convert the main dataframe to a numpy array

Parameters

- **table_name** – name of the python variable, defaults to "data"
- **table_name** – str, optional

Returns a numpy array

Return type np.array

Example `ds.to_numpy_("myvar")`

3.3 To python code

Export.**to_python_**(*table_name: str = 'data'*) → list
Convert the main dataframe to python a python list

Parameters

- **table_name** – python variable name, defaults to "data"
- **table_name** – str, optional

Returns a python list of lists with the data

Return type str

Example `ds.to_python_("myvar")`

CHAPTER 4

Database operations

All operations available for databases. Supported databases: all that SqlAlchemy support

4.1 Connect

Db.**connect**(*url: str*)
Connect to the database and set it as main database

Parameters **url** (*str*) – path to the database, uses the Sqlalchemy format

Example `ds.connect("sqlite:///mydb.slqite")`

4.2 Load data

Db.**load**(*table: str*)
Set the main dataframe from a table's data

Parameters **table** (*str*) – table name

Example `ds.load("mytable")`

Db.**load_django**(*query: django query*)
Load the main dataframe from a django orm query

Parameters **query** (*django query*) – django query from a model

Example `ds.load_django(Mymodel.objects.all())`

Db.**load_django_**(*query: django query*) → Ds
Returns a DataSwim instance from a django orm query

Parameters **query** (*django query*) – django query from a model

Returns a dataswim instance with data from a django query

Return type *Ds*

Example `ds2 = ds.load_django_(Mymodel.objects.all())`

4.3 Infos

`Db.`**`tables`**`()`
Print the existing tables in a database

Example `ds.tables()`

`Db.`**`tables_`**`()` → list
Return a list of the existing tables in a database

Returns list of the table names

Return type list

Example `tables = ds.tables_()`

`Db.`**`table`**`(`*name: str*`)`
Display info about a table: number of rows and columns

Parameters **`name`** (*`str`*) – name of the table

Example `tables = ds.table("mytable")`

4.4 Insert

`Insert.`**`to_db`**`(`*table: str*, *dtypes: List[sqlalchemy.sql.sqltypes.SchemaType] = None*`)`
Save the main dataframe to the database

Parameters

- **`table`** (*`str`*) – the table to create
- **`dtypes`** (*`List[SchemaType], optional`*) – SqlAlchemy columns type, defaults to None, will be infered if not provided

`Insert.`**`update_table`**`(`*table: str, pks: List[str] = ['id'], mirror: bool = True*`)`
Update records in a database table from the main dataframe

Parameters

- **`table`** (*`str`*) – table to update
- **`pks`** (*`List[str], optional :param mirror: delete the rows not in the new dataset`*) – if rows with matching pks exist they will be updated, otherwise a new row is inserted in the table, defaults to ["id"]

`Insert.`**`insert`**`(`*table: str*, *records: dict*, *create_cols: bool = False*, *dtypes: List[sqlalchemy.sql.sqltypes.SchemaType] = None*`)`

Insert one or many records in the database from a dictionary or a list of dictionaries

Parameters

- **`table`** (*`str`*) – the table to insert into
- **`records`** (*`dict`*) – a dictionnary or list of dictionnaries of the data to insert

- **create_cols** (*bool, optional*) – create the columns if they don't exist, defaults to False
- **dtypes** (*SchemaType, optional*) – list of SqlAlchemy table types, defaults to None. The types are infered if not provided

Insert.**upsert**(*table: str, record: dict, create_cols: bool = False, dtypes: List[sqlalchemy.sql.sqltypes.SchemaType] = None, pks: List[str] = ['id']*)

Upsert a record in a table

Parameters

- **table** (*str*) – the table to upsert into
- **record** (*dict*) – dictionary with the data to upsert
- **create_cols** (*bool, optional*) – create the columns if it doesn't exist, defaults to False
- **dtypes** (*List[SchemaType], optional*) – list of SqlAlchemy column types, defaults to None
- **pks** (*List[str], optional*) – if rows with matching pks exist they will be updated, otherwise a new row is inserted in the table, defaults to ["id"]

4.5 Relations

Relation.**relation**(*table: str, origin_field: str, search_field: str, destination_field: str = None, id_field: str = 'id'*)

Add a column to the main dataframe from a relation foreign key

Parameters

- **table** (*str*) – the table to select from
- **origin_field** (*str*) – the column name in the origin table to search from, generally an id column
- **search_field** (*str*) – the column name in the foreign table
- **destination_field** (*str, optional*) – name of the column to be created with the data in the datframe, defaults to None, will be named as the origin_field if not provided
- **id_field** (*str, optional*) – name of the primary key to use, defaults to "id"

example: `ds.relation("product", "category_id", "name")`

Relation.**relation_**(*table: str, origin_field: str, search_field: str, destination_field=None, id_field='id'*) → pandas.core.frame.DataFrame

Returns a DataSwim instance with a column filled from a relation foreign key

Parameters

- **table** (*str*) – the table to select from
- **origin_field** (*str*) – the column name in the origin table to search from, generally an id column
- **search_field** (*str*) – the column name in the foreign table
- **destination_field** (*str, optional*) – name of the column to be created with the data in the datframe, defaults to None, will be named as the origin_field if not provided

- **id_field** (*str, optional*) – name of the primary key to use, defaults to "id"

Returns a pandas DataFrame

Return type DataFrame

4.6 InfluxDb

influx_init (`url`, `port`, `user`, `pwd`, `db`)

Initialize an Influxdb database client

influx_to_csv (`measurement`, `batch_size=5000`)

Batch export data from an Influxdb measurement to csv

CHAPTER 5

Nulls

`Clean.`**`drop_nan`**(*col: str = None*, *method: str = 'all'*, ***kwargs*)
Drop rows with NaN values from the main dataframe

Parameters

- **`col`** (*`str, optional`*) – name of the column, defaults to None. Drops in
- **`method`** (*`str, optional`*) – `how` param for `df.dropna`, defaults to "all"
- **`**kwargs`** (*`optional`*) – params for `df.dropna`

Example `ds.drop_nan("mycol")`

`Clean.`**`nan_empty`**(*col: str*)
Fill empty values with NaN values

Parameters **`col`** (*`str`*) – name of the colum

Example `ds.nan_empty("mycol")`

`Clean.`**`zero_nan`**(**cols*)
Converts zero values to nan values in selected columns

Parameters **`*cols`** (*`str, at least one`*) – names of the colums

Example `ds.zero_nan("mycol1", "mycol2")`

`Clean.`**`fill_nan`**(*val: str*, **cols*)
Fill NaN values with new values in the main dataframe

Parameters

- **`val`** (*`str`*) – new value
- **`*cols`** (*`str, at least one`*) – names of the colums

Example `ds.fill_nan("new value", "mycol1", "mycol2")`

`Clean.`**`fill_nulls`**(*col: str*)
Fill all null values with NaN values in a column. Null values are `None` or en empty string

Parameters **col** (*str*) – column name

Example `ds.fill_nulls("mycol")`

CHAPTER 6

Dates

Clean.**date**(*col: str*, ***kwargs*)
Convert a column to date type

Parameters

- **col** (*str*) – column name
- ****kwargs** (*optional*) – keyword arguments for pd.to_datetime

Example `ds.date("mycol")`

Clean.**fdate**(**cols*, *precision: str = 'S'*, *format: str = None*)
Convert column values to formated date string

Parameters

- ***cols** (*str, at least one*) – names of the colums
- **precision** (*str, optional*) – time precision: Y, M, D, H, Min S, defaults to "S"
- **format** (*str, optional*) – python date format, defaults to None

Example `ds.fdate("mycol1", "mycol2", precision)`

Clean.**timestamps**(*col: str*, ***kwargs*)
" Add a timestamps column from a date column

Parameters

- **col** (*str*) – name of the timestamps column to add
- ****kwargs** (*optional*) – keyword arguments for pd.to_datetime

Example `ds.timestamps("mycol")`

CHAPTER 7

Convert types

Clean.**to_int**(**cols*, ***kwargs*)

Convert some column values to integers

Parameters

- ***cols** (*str, at least one*) – names of the colums
- ****kwargs** (*optional*) – keyword arguments for pd.to_numeric

Example ds.to_int("mycol1", "mycol2", errors="coerce")

Clean.**to_float**(*col: str*, ***kwargs*)

Convert colums values to float

Parameters

- **col** (*str, at least one*) – name of the colum
- ****kwargs** (*optional*) – keyword arguments for df.astype

Example ds.to_float("mycol1")

Clean.**to_type**(*dtype: type*, **cols*, ***kwargs*)

Convert colums values to a given type in the main dataframe

Parameters

- **dtype** (*type*) – a type to convert to: ex: str
- ***cols** (*str, at least one*) – names of the colums
- ****kwargs** (*optional*) – keyword arguments for df.astype

Example ds.to_type(str, "mycol")

CHAPTER 8

Create index

Clean.**index**(*col: str*)
Set an index to the main dataframe

Parameters **col** (*str*) – column name where to index from

Example `ds.index("mycol")`

8.1 Date index

Clean.**dateindex**(*col: str*)
Set a datetime index from a column

Parameters **col** (*str*) – column name where to index the date from

Example `ds.dateindex("mycol")`

CHAPTER 9

Clean values

Clean.**strip**(*col: str*)
Remove leading and trailing white spaces in a column's values

Parameters **col** (*str*) – name of the column

Example `ds.strip("mycol")`

Clean.**strip_cols**()
Remove leading and trailing white spaces in columns names

Example `ds.strip_cols()`

Clean.**roundvals**(*col: str*, *precision: int = 2*)
Round floats in a column. Numbers are going to be converted to floats if they are not already

Parameters

- **col** (*str*) – column name
- **precision** – float precision, defaults to 2
- **precision** – int, optional

Example `ds.roundvals("mycol")`

Clean.**replace**(*col: str*, *searchval: str*, *replaceval: str*)
Replace a value in a column in the main dataframe

Parameters

- **col** (*str*) – column name
- **searchval** (*str*) – value to replace
- **replaceval** (*str*) – new value

Example `ds.replace("mycol", "value", "new_value")`

CHAPTER 10

Columns

Columns.**rename**(*source_col: str*, *dest_col: str*)

Renames a column in the main dataframe

Parameters

- **source_col** (*str*) – name of the column to rename
- **dest_col** (*str*) – new name of the column

Example `ds.rename("Col 1", "New col")`

Columns.**add**(*col: str*, *value*)

Add a column with default values

Parameters

- **col** (*str*) – column name
- **value** (*any*) – column value

Example `ds.add("Col 4", 0)`

Columns.**keep**(**cols*)

Limit the dataframe to some columns

Parameters **cols** (*str*) – names of the columns

Example `ds.keep("Col 1", "Col 2")`

Columns.**keep_**(**cols*) → Ds

Returns a dataswim instance with a dataframe limited to some columns

Parameters **cols** (*str*) – names of the columns

Returns a dataswim instance

Return type *Ds*

Example `ds2 = ds.keep_("Col 1", "Col 2")`

Columns.**drop**(**cols*)
Drops columns from the main dataframe

Parameters **cols** (*str*) – names of the columns

Example `ds.drop("Col 1", "Col 2")`

Columns.**exclude**(*col: str*, *val*)
Delete rows based on value

Parameters

- **col** (*str*) – column name
- **val** (*any*) – value to delete

Example `ds.exclude("Col 1", "value")`

Columns.**copycol**(*origin_col: str*, *dest_col: str*)
Copy a columns values in another column

Parameters

- **origin_col** (*str*) – name of the column to copy
- **dest_col** (*str*) – name of the new column

Example `ds.copy("col 1", "New col")`

Columns.**indexcol**(*col: str*)
Add a column from the index

Parameters **col** (*str*) – name of the new column

Example `ds.index_col("New col")`

CHAPTER 11

Values

`Values.`**`sum_`**`(`*col: str*`)` → float

Returns the sum of all values in a column

Parameters **`col`** (*`str`*) – column name

Returns sum of all the column values

Return type float

Example `sum = ds.sum_("Col 1")`

`Values.`**`dropr`**`(`**rows*`)`

Drops some rows from the main dataframe

Parameters **`rows`** (*`list of ints`*) – rows names

Example `ds.drop_rows([0, 2])`

`Values.`**`apply`**`(`*function*, **cols*, *axis=1*, ***kwargs*`)`

Apply a function on columns values

Parameters

- **`function`** (*`function`*) – a function to apply to the columns
- **`cols`** (*`name of columns`*) – columns names
- **`axis`** – index (0) or column (1), default is 1
- **`kwargs`** (*`optional`*) – arguments for `df.apply`

Example

```
def f(row):
        # add a new column with a value
        row["newcol"] = row["Col 1] + 1
        return row

ds.apply(f)
```

Values.**pivot**(*index*, ***kwargs*)
Pivots a dataframe

11.1 Quantiles

Values.**trimquants**(*col: str*, *inf: float*, *sup: float*)
Remove superior and inferior quantiles from the dataframe

Parameters

- **col** (*str*) – column name
- **inf** (*float*) – inferior quantile
- **sup** (*float*) – superior quantile

Example `ds.trimquants("Col 1", 0.01, 0.99)`

Values.**trimsquants**(*col: str*, *sup: float*)
Remove superior quantiles from the dataframe

Parameters

- **col** (*str*) – column name
- **sup** (*float*) – superior quantile

Example `ds.trimsquants("Col 1", 0.99)`

Values.**trimiquants**(*col: str*, *inf: float*)
Remove superior and inferior quantiles from the dataframe

Parameters

- **col** (*str*) – column name
- **inf** (*float*) – inferior quantile

Example `ds.trimiquants("Col 1", 0.05)`

CHAPTER 12

Resample timeseries

Resample.**rsum**(*time_period: str*, *num_col: str = 'Number'*, *dateindex: str = None*)

Resample and add a sum the main dataframe to a time period

Parameters

- **time_period** – unit + period: periods are Y, M, D, H, Min, S
- **time_period** – str
- **num_col** – number of the new column, defaults to "Number"
- **num_col** – str, optional
- **dateindex** – column name to use as date index, defaults to None
- **dateindex** – str, optional

Example `ds.rsum("1D")`

Resample.**rmean**(*time_period: str*, *num_col: str = 'Number'*, *dateindex: str = None*)

Resample and add a mean column the main dataframe to a time period

Parameters

- **time_period** – unit + period: periods are Y, M, D, H, Min, S
- **time_period** – str
- **num_col** – number of the new column, defaults to "Number"
- **num_col** – str, optional
- **dateindex** – column name to use as date index, defaults to None

Example `ds.rmean("1Min")`

CHAPTER 13

Dataframe

Dataframe.**concat**(**dss*, ***kwargs*)

Concatenate dataswim instances from and set it to the main dataframe

Parameters

- **dss** (Ds) – dataswim instances to concatenate
- **kwargs** – keyword arguments for pd.concat

Dataframe.**split_**(*col: str) -> list(Ds*)

Split the main dataframe according to a column's unique values and return a dict of dataswim instances

Returns list of dataswim instances

Return type list(*Ds*)

Example dss = ds.slit_("Col 1")

Dataframe.**merge**(*df: pandas.core.frame.DataFrame*, *on: str*, *how: str = 'outer'*, ***kwargs*)

Set the main dataframe from the current dataframe and the passed dataframe

Parameters

- **df** (*pd.DataFrame*) – the pandas dataframe to merge
- **on** (*str*) – param for pd.merge
- **how** (*str, optional*) – param for pd.merge, defaults to "outer"
- **kwargs** – keyword arguments for pd.merge

CHAPTER 14

Calculations

14.1 Diffs

`Calculations.`**`diffn`**(*diffcol: str*, *name: str = 'Diff'*)

Add a diff column to the main dataframe: calculate the diff from the next value

Parameters

- **`diffcol`** (*`str`*) – column to diff from
- **`name`** (*`str, optional`*) – diff column name, defaults to "Diff"

Example `ds.diffn("Col 1", "New col")`

`Calculations.`**`diffp`**(*diffcol: str*, *name: str = 'Diff'*)

Add a diff column to the main dataframe: calculate the diff from the previous value

Parameters

- **`diffcol`** (*`str`*) – column to diff from
- **`name`** (*`str, optional`*) – diff column name, defaults to "Diff"

Example `ds.diffp("Col 1", "New col")`

`Calculations.`**`diffm`**(*diffcol: str*, *name: str = 'Diff'*, *default=nan*)

Add a diff column to the main dataframe: calculate the diff from the column mean

Parameters

- **`diffcol`** (*`str`*) – column to diff from
- **`name`** – diff column name, defaults to "Diff"
- **`name`** – str, optional
- **`default`** – column default value, defaults to nan
- **`default`** – optional

Example `ds.diffm("Col 1", "New col")`

Calculations.**diffs**(*col: str*, *serie: iterable*, *name: str = 'Diff'*)
Add a diff column from a serie. The serie is an iterable of the same length than the dataframe

Parameters

- **col** (*str*) – column to diff
- **serie** (*iterable*) – serie to diff from
- **name** – name of the diff col, defaults to "Diff"
- **name** – str, optional

Example `ds.diffs("Col 1", [1, 1, 4], "New col")`

Calculations.**diffsp**(*col: str*, *serie: iterable*, *name: str = 'Diff'*)
Add a diff column in percentage from a serie. The serie is an iterable of the same length than the dataframe

Parameters

- **col** (*str*) – column to diff
- **serie** (*iterable*) – serie to diff from
- **name** – name of the diff col, defaults to "Diff"
- **name** – str, optional

Example `ds.diffp("Col 1", [1, 1, 4], "New col")`

14.2 Group by

Calculations.**gmean_**(*col: str*, *index_col: bool = True*) → Ds
Group by and mean column

Parameters

- **col** (*str*) – column to group
- **index_col** (*bool*) –

Returns a dataswim instance

Return type *Ds*

Example `ds2 = ds.gmean("Col 1")`

Calculations.**gsum_**(*col: str*, *index_col: bool = True*) → Ds
Group by and sum column

Parameters

- **col** (*str*) – column to group
- **index_col** (*bool*) –

Returns a dataswim instance

Return type *Ds*

Example `ds2 = ds.gsum("Col 1")`

14.3 Ratio

Calculations.**ratio**(*col: str*, *ratio_col: str = 'Ratio'*)

Add a column whith the percentages ratio from a column

Parameters

- **col** (*str*) – column to calculate ratio from
- **ratio_col** – new ratio column name, defaults to "Ratio"
- **ratio_col** – str, optional

Example `ds2 = ds.ratio("Col 1")`

CHAPTER 15

View

View.**show**(*rows: int = 5*, *dataframe: pandas.core.frame.DataFrame = None*) → pandas.core.frame.DataFrame

Display info about the dataframe

Parameters

- **rows** – number of rows to show, defaults to 5
- **rows** – int, optional
- **dataframe** – a pandas dataframe, defaults to None
- **dataframe** – pd.DataFrame, optional

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.show()`

View.**one**()

Shows one row of the dataframe and the field names wiht count

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.one()`

View.**tail**(*rows: int = 5*)

Returns the main dataframe's tail

Parameters

- **rows** – number of rows to print, defaults to 5
- **rows** – int, optional

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.tail()`

15.1 Infos

`View.`**`cols_`**`()` → pandas.core.frame.DataFrame
Returns a dataframe with columns info

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.cols_()`

`View.`**`describe_`**`()`
Return a description of the data

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.describe()`

`View.`**`types_`**`(`*col: str*`)` → pandas.core.frame.DataFrame
Display types of values in a column

Parameters **`col`** (*`str`*) – column name

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.types_("Col 1")`

CHAPTER 16

Select

16.1 Rows

`Select.`**`first_`**`()` → pandas.core.series.Series
Select the first row

Returns the first row as a serie

Return type pd.Series

`Select.`**`limit`**(*r: int = 5*)
Limit selection to a range in the main dataframe

Parameters **`r`** (*`int, optional`*) – number of rows to keep, defaults to 5

`Select.`**`limit_`**(*r: int = 5*) → Ds
Returns a DataSwim instance with limited selection

Returns a Ds instance

Return type *Ds*

`Select.`**`unique_`**(*col: str*) → list
Returns unique values in a column

Parameters **`col`** (*`str`*) – the column to select from

Returns a list of unique values in the column

Return type list

`Select.`**`wunique_`**(*col*)
Weight unique values: returns a dataframe with a count of unique values

`Select.`**`subset`**(**args*)
Set the main dataframe to a subset based in positions Select a subset of the main dataframe based on position: ex: ds.subset(0,10) or ds.subset(10) is equivalent: it starts at the first row if only one argument is provided

Select.**subset_**(**args*)
Returns a Dataswim instance with a subset data based in positions Select a subset of the main dataframe based on position: ex: ds.subset(0,10) or ds.subset(10) is equivalent: it starts at the first row if only one argument is provided

16.2 From dates

Select.**nowrange**(*col: str*, *timeframe: str*)
Set the main dataframe with rows within a date range from now

Parameters

- **col** (*str*) – the column to use for range
- **timeframe** (*str*) – units are: S, H, D, W, M, Y

example: `ds.nowrange("Date", "3D")`

Select.**nowrange_**(*col: str*, *timeframe: str*) → Ds
Returns a Dataswim instance with rows within a date range from now

Parameters

- **col** (*str*) – the column to use for range
- **timeframe** (*str*) – units are: S, H, D, W, M, Y

Returns [description]

Return type [type]

example: `ds2 = ds.nowrange_("Date", "3D")`

Select.**daterange**(*datecol: str*, *date_start: datetime.datetime*, *op: str*, ***args*)
Set the main dataframe rows in a date range

Parameters

- **datecol** (*str*) – the column to use for range
- **date_start** (*datetime.datetime*) – the date to start from
- **op** (*str*) –
  - or -

Select.**daterange_**(*datecol: str*, *date_start: datetime.datetime*, *op: str*, ***args*) → Ds
Returns a DataSwim instance with rows in a date range

Parameters

- **datecol** (*str*) – the column to use for range
- **date_start** (*datetime.datetime*) – the date to start from
- **op** (*str*) –
  - or -

Returns a dataswim instance

Return type *Ds*

CHAPTER 17

Count

Count.**count**()
Counts the number of rows of the main dataframe

Count.**count_nulls**(*field: str*)
Count the number of null values in a column

Parameters **field** (*str*) – the column to count from

Count.**count_empty**(*field: str*)
List of empty row indices

Parameters **field** (*str*) – column to count from

Count.**count_zero**(*field: str*)
List of row with 0 values

Parameters **field** (*str*) – column to count from

Count.**count_unique_**(*field: str*) → int
Return the number of unique values in a column

Parameters **field** (*str*) – column to count from

Returns number of unique values

Return type int

CHAPTER 18

Search

nulls_ (`field`)

Return all null rows

contains (`column`, `value`)

Returns rows that contains a string value in a column

exact (`column`, `value`)

Returns rows that has the exact string value in a column

CHAPTER 19

Utils

set (`df=None`, `db=None`)

Set a main dataframe

clone ()

Returns a new DataSwim instance from the current instance

duplicate_ (`df=None`, `db=None`, `quiet=False`)

Returns a new DataSwim instance using the previous database connection

backup ()

Backup the main dataframe

restore ()

Restore the main dataframe

CHAPTER 20

Chart data

20.1 Charts

chart (`x_field=None, y_field=None, chart_type="line", opts=None, style=None, label=None`)

Initialize a chart

chart_ (`x_field=None, y_field=None, chart_type="line", opts=None, style=None, label=None`)

Initialize and return a chart

bar_ (`style=None, opts=None, label=None`)

Get a bar chart

line_ (`style=None, opts=None, label=None`)

Get a line chart

area_ (`style=None, opts=None, label=None`)

Get an area chart

line_point_ (`colors={'line': 'yellow', 'point': 'navy'}, style=None, opts=None, label=None`)

Get a line and point chart

point_ (`style=None, opts=None, label=None`)

Get a point chart

area_ (`style=None, opts=None, label=None`)

Get a line chart

hist_ (`style=None, opts=None, label=None`)

Get an historiogram chart

errorbar_ (`style=None`, `opts=None`, `label=None`)

Get an errorbar chart

heatmap_ (`style=None`, `opts=None`, `label=None`)

Get a heatmap chart

linear_ (`style=None`, `opts=None`, `label=None`)

Get a linear regression chart

dlinear_ (`style=None`, `opts=None`, `label=None`)

Get a linear regression chart with marginal distribution

density_ (`style=None`, `opts=None`, `label=None`)

Get a density chart

distrib_ (`style=None`, `opts=None`, `label=None`)

Get a distribution chart

residual_ (`style=None`, `opts=None`, `label=None`)

Returns a models residuals chart

20.2 Layouts

layout_ (`chart_objs`, `cols=3`)

Returns a Holoview layout from chart objects

20.3 Chart options

opts (`key`, `value`)

Add or update an option value to defaults

style (`key`, `value`)

Add or update a style value to defaults

CHAPTER 21

Reports

csv (`path`)

Saves the main dataframe to a csv file

stack (`slug`, `title`, `chart_obj=None`)

Get the html for a chart and store it

files (`folderpath=None`, `p=True`)

Writes the html report to one file per report from the report stack

get_html (`chart_obj=None`)

Get the html and script tag for a chart

CHAPTER 22

Conventions

All functions that end with an underscore return an object. You can often see the same functions with and without underscore: ex:

This sets a datetime index from a column in the main dataframe:

```
ds.dateindex("date")
```

This returns a new instance with a dataframe set with a datetime index:

```
ds2 = ds.dateindex_("date")
```

Note: some functions without underscore can still return something: ex: `ds.show()` returns a dataframe's head

CHAPTER 23

API

class dataswim.**Ds**(*df=None*, *db=None*, *nbload_libs=True*)

Bases: dataswim.db.Db, dataswim.data.Df, dataswim.charts.Plot, dataswim.maps.Map, dataswim.report.Report, dataswim.base.DsBase

Main class

add(*col: str*, *value*)

Add a column with default values

Parameters

- **col** (*str*) – column name
- **value** (*any*) – column value

Example `ds.add("Col 4", 0)`

aenc(*key*, *value*)

Add an entry to the altair encoding dict

altair_encode = {}

altair_header_()

Returns html script tags for Altair

amap(*lat*, *long*, *zoom=13*, *tiles='map'*)

Sets a map

append(*vals: list*, *index=None*)

Append a row to the main dataframe

Parameters

- **vals** (*list*) – list of the row values to add
- **index** – index key, defaults to None
- **index** – any, optional

Example `ds.append([0, 2, 2, 3, 4])`

apply (*function*, **cols*, *axis=1*, ***kwargs*)
Apply a function on columns values

Parameters

- **function** (*function*) – a function to apply to the columns
- **cols** (*name of columns*) – columns names
- **axis** – index (0) or column (1), default is 1
- **kwargs** (*optional*) – arguments for `df.apply`

Example

```
def f(row):
        # add a new column with a value
        row["newcol"] = row["Col 1] + 1
        return row

ds.apply(f)
```

area_ (*label=None*, *style=None*, *opts=None*, *options={}*)
Get an area chart

arrow_ (*xloc*, *yloc*, *text*, *orientation='v'*, *arrowstyle='->'*)
Returns an arrow for a chart. Params: the text, xloc and yloc are coordinates to position the arrow. Orientation is the way to display the arrow: possible values are `[<, ^, >, v]`. Arrow style is the graphic style of the arrow: possible values: `[-, ->, -[, -|>, <->, <|-|>]`

autoprint = False

backup ()
Backup the main dataframe

backup_df = None

bar_ (*label=None*, *style=None*, *opts=None*, *options={}*)
Get a bar chart

bar_num_ (*label=None*, *style=None*, *opts=None*)
Get an Altair bar + number marks chart

bokeh_header_ ()
Returns html script tags for Bokeh

chart (*x=None*, *y=None*, *chart_type=None*, *opts=None*, *style=None*, *label=None*, *options={}*, ***kwargs*)
Get a chart

chart_ (*x=None*, *y=None*, *chart_type=None*, *opts=None*, *style=None*, *label=None*, *options={}*, ***kwargs*)
Get a chart

chart_obj = None

chart_opts = {'width': 880}

chart_style = {}

chartjs_header_ ()
Returns html script tags for Chartjs

circle_ (*label=None*, *style=None*, *opts=None*, *options={}*)
Get a circle chart

clone_(*quiet=False*)
Clone the DataSwim instance

Parameters **quiet** (*bool, optional*) – print a message, defaults to False

Returns a dataswim instance

Return type *Ds*

color(*val*)
Change the chart's color

color_(*i=None*)
Get a color from the palette

color_index = 0

cols_() → pandas.core.frame.DataFrame
Returns a dataframe with columns info

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.cols_()`

concat(**dss*, ***kwargs*)

Concatenate dataswim instances from and set it to the main dataframe

Parameters

- **dss** (`Ds`) – dataswim instances to concatenate
- **kwargs** – keyword arguments for `pd.concat`

concat_(**dss*, ***kwargs*)

Concatenate dataswim instances and return a new Ds instance

Parameters

- **dss** (`Ds`) – dataswim instances to concatenate
- **kwargs** – keyword arguments for `pd.concat`

Return type *Ds*

connect(*url: str*)
Connect to the database and set it as main database

Parameters **url** (*str*) – path to the database, uses the Sqlalchemy format

Example `ds.connect("sqlite:///mydb.slqite")`

contains(*column*, *value*)
Set the main dataframe instance to rows that contains a string value in a column

copycol(*origin_col: str*, *dest_col: str*)
Copy a columns values in another column

Parameters

- **origin_col** (*str*) – name of the column to copy
- **dest_col** (*str*) – name of the new column

Example `ds.copy("col 1", "New col")`

count()
Counts the number of rows of the main dataframe

count_() → int
Returns the number of rows of the main dataframe

Returns number of rows

Return type int

count_empty(*field: str*)
List of empty row indices

Parameters **field** (*str*) – column to count from

count_nulls(*field: str*)
Count the number of null values in a column

Parameters **field** (*str*) – the column to count from

count_unique_(*field: str*) → int
Return the number of unique values in a column

Parameters **field** (*str*) – column to count from

Returns number of unique values

Return type int

count_zero(*field: str*)
List of row with 0 values

Parameters **field** (*str*) – column to count from

cvar_(*col*)
Returns the coefficient of variance of a column in percentage

datapath = None

date(*col: str*, ***kwargs*)
Convert a column to date type

Parameters

- **col** (*str*) – column name
- ****kwargs** (*optional*) – keyword arguments for `pd.to_datetime`

Example `ds.date("mycol")`

dateindex(*col: str*)
Set a datetime index from a column

Parameters **col** (*str*) – column name where to index the date from

Example `ds.dateindex("mycol")`

dateparser(*dformat='%d/%m/%Y'*)
Returns a date parser for pandas

daterange(*datecol: str*, *date_start: datetime.datetime*, *op: str*, ***args*)
Set the main dataframe rows in a date range

Parameters

- **datecol** (*str*) – the column to use for range

- **date_start** (*datetime.datetime*) – the date to start from
- **op** (*str*) –
  - or -

daterange_(*datecol: str*, *date_start: datetime.datetime*, *op: str*, ***args*) → Ds
Returns a DataSwim instance with rows in a date range

Parameters

- **datecol** (*str*) – the column to use for range
- **date_start** (*datetime.datetime*) – the date to start from
- **op** (*str*) –
  - or -

Returns a dataswim instance

Return type *Ds*

db = None

debug(**msg*)
Prints a warning

defaults()
Reset the chart options and style to defaults

density_(*label=None*, *style=None*, *opts=None*)
Get a Seaborn density chart

describe_()
Return a description of the data

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.describe()`

df = None

diffm(*diffcol: str*, *name: str = 'Diff'*, *default=nan*)
Add a diff column to the main dataframe: calculate the diff from the column mean

Parameters

- **diffcol** (*str*) – column to diff from
- **name** – diff column name, defaults to "Diff"
- **name** – str, optional
- **default** – column default value, defaults to nan
- **default** – optional

Example `ds.diffm("Col 1", "New col")`

diffn(*diffcol: str*, *name: str = 'Diff'*)
Add a diff column to the main dataframe: calculate the diff from the next value

Parameters

- **diffcol** (*str*) – column to diff from

- **name** (*str, optional*) – diff column name, defaults to "Diff"

Example `ds.diffn("Col 1", "New col")`

diffp (*diffcol: str*, *name: str = 'Diff'*)
Add a diff column to the main dataframe: calculate the diff from the previous value

Parameters

- **diffcol** (*str*) – column to diff from
- **name** (*str, optional*) – diff column name, defaults to "Diff"

Example `ds.diffp("Col 1", "New col")`

diffs (*col: str*, *serie: iterable*, *name: str = 'Diff'*)
Add a diff column from a serie. The serie is an iterable of the same length than the dataframe

Parameters

- **col** (*str*) – column to diff
- **serie** (*iterable*) – serie to diff from
- **name** – name of the diff col, defaults to "Diff"
- **name** – str, optional

Example `ds.diffs("Col 1", [1, 1, 4], "New col")`

diffsp (*col: str*, *serie: iterable*, *name: str = 'Diff'*)
Add a diff column in percentage from a serie. The serie is an iterable of the same length than the dataframe

Parameters

- **col** (*str*) – column to diff
- **serie** (*iterable*) – serie to diff from
- **name** – name of the diff col, defaults to "Diff"
- **name** – str, optional

Example `ds.diffp("Col 1", [1, 1, 4], "New col")`

distrib_ (*label=None*, *style=None*, *opts=None*)
Get a Seaborn distribution chart

dlinear_ (*label=None*, *style=None*, *opts=None*)
Get a Seaborn linear + distribution chart

drop (**cols*)
Drops columns from the main dataframe

Parameters **cols** (*str*) – names of the columns

Example `ds.drop("Col 1", "Col 2")`

drop_nan (*col: str = None*, *method: str = 'all'*, ***kwargs*)
Drop rows with NaN values from the main dataframe

Parameters

- **col** (*str, optional*) – name of the column, defaults to None. Drops in
- **method** (*str, optional*) – `how` param for `df.dropna`, defaults to "all"
- ****kwargs** (*optional*) – params for `df.dropna`

Example `ds.drop_nan("mycol")`

dropr(**rows*)
Drops some rows from the main dataframe

Parameters **rows** (*list of ints*) – rows names

Example `ds.drop_rows([0, 2])`

dsmap = None

end(**msg*)
Prints an end message with elapsed time

engine = 'bokeh'

err(**args*)
Handle an error

errorbar_(*label=None*, *style=None*, *opts=None*, *options={}*)
Get a point chart

errors_handling = 'exceptions'

exact(*column*, **values*)
Sets the main dataframe to rows that has the exact string value in a column

exact_(*column*, **values*)
Returns a Dataswim instance with rows that has the exact string value in a column

exclude(*col: str*, *val*)
Delete rows based on value

Parameters

- **col** (*str*) – column name
- **val** (*any*) – value to delete

Example `ds.exclude("Col 1", "value")`

fdate(**cols*, *precision: str = 'S'*, *format: str = None*)
Convert column values to formated date string

Parameters

- ***cols** (*str, at least one*) – names of the colums
- **precision** (*str, optional*) – time precision: Y, M, D, H, Min S, defaults to "S"
- **format** (*str, optional*) – python date format, defaults to None

Example `ds.fdate("mycol1", "mycol2", precision)`

fill_nan(*val: str*, **cols*)
Fill NaN values with new values in the main dataframe

Parameters

- **val** (*str*) – new value
- ***cols** (*str, at least one*) – names of the colums

Example `ds.fill_nan("new value", "mycol1", "mycol2")`

fill_nulls(*col: str*)
Fill all null values with NaN values in a column. Null values are `None` or en empty string

Parameters **col** (*str*) – column name

Example `ds.fill_nulls("mycol")`

first_() → pandas.core.series.Series
Select the first row

Returns the first row as a serie

Return type pd.Series

flat_(*col*, *nums=True*)
Returns a flat representation of a column's values

footer = None

format_date_(*date: datetime.datetime*) → str
Format a date string

Parameters **date** (*datetime.datetime*) – the input date

Returns output date string

Return type str

get_html(*chart_obj=None*, *slug=None*)
Get the html and script tag for a chart

getall_(*table*)
Get all rows values for a table

gmean_(*col: str*, *index_col: bool = True*) → Ds
Group by and mean column

Parameters

- **col** (*str*) – column to group
- **index_col** (*bool*) –

Returns a dataswim instance

Return type *Ds*

Example `ds2 = ds.gmean("Col 1")`

gsum_(*col: str*, *index_col: bool = True*) → Ds
Group by and sum column

Parameters

- **col** (*str*) – column to group
- **index_col** (*bool*) –

Returns a dataswim instance

Return type *Ds*

Example `ds2 = ds.gsum("Col 1")`

header = None

heatmap_(*label=None*, *style=None*, *opts=None*, *options={}*)
Get a heatmap chart

height(*val*)
Change the chart's height

hist_(*label=None*, *style=None*, *opts=None*, *options={}*)
Get an historiogram chart

hline_(*label=None*, *style=None*, *opts=None*, *options={}*)
Get a mean line chart

html(*label*, **msg*)
Prints html in notebook

index(*col: str*)
Set an index to the main dataframe

Parameters **col** (*str*) – column name where to index from

Example `ds.index("mycol")`

indexcol(*col: str*)
Add a column from the index

Parameters **col** (*str*) – name of the new column

Example `ds.index_col("New col")`

influx_cli = None

influx_count_(*measurement*)
Count the number of rows for a measurement

influx_init(*url*, *port*, *user*, *pwd*, *db*)
Initialize an Influxdb database client

influx_query_(*q*)
Runs an Influx db query

influx_to_csv(*measurement*, *batch_size=5000*)
Batch export data from an Influxdb measurement to csv

info(**msg*)
Prints a message with an info prefix

insert(*table: str*, *records: dict*, *create_cols: bool = False*, *dtypes: List[sqlalchemy.sql.sqltypes.SchemaType] = None*)

Insert one or many records in the database from a dictionary or a list of dictionaries

Parameters

- **table** (*str*) – the table to insert into
- **records** (*dict*) – a dictionnary or list of dictionnaries of the data to insert
- **create_cols** (*bool, optional*) – create the columns if they don't exist, defaults to False
- **dtypes** (*SchemaType, optional*) – list of SqlAlchemy table types, defaults to None. The types are infered if not provided

keep(**cols*)
Limit the dataframe to some columns

Parameters **cols** (*str*) – names of the columns

Example `ds.keep("Col 1", "Col 2")`

keep_(**cols*) → Ds
Returns a dataswim instance with a dataframe limited to some columns

Parameters **cols** (*str*) – names of the columns

Returns a dataswim instance

Return type *Ds*

Example `ds2 = ds.keep_("Col 1", "Col 2")`

label = None

layout_(*chart_objs*, *cols=3*)
Returns a Holoview Layout from chart objects

limit(*r: int = 5*)
Limit selection to a range in the main dataframe

Parameters **r** (*int, optional*) – number of rows to keep, defaults to 5

limit_(*r: int = 5*) → Ds
Returns a DataSwim instance with limited selection

Returns a Ds instance

Return type *Ds*

line_(*label=None*, *style=None*, *opts=None*, *options={}*)
Get a line chart

line_num_(*label=None*, *style=None*, *opts=None*)
Get an Altair line + number marks chart

line_point_(*label=None*, *style=None*, *opts=None*, *options={}*, *colors={'line': 'orange', 'point': '#30A2DA'}*)
Get a line and point chart

load(*table: str*)
Set the main dataframe from a table's data

Parameters **table** (*str*) – table name

Example `ds.load("mytable")`

load_csv(*url*, ***kwargs*)
Loads csv data in the main dataframe

Parameters

- **url** (*str*) – url of the csv file to load: can be absolute if it starts with `/` or relative if it starts with `./`
- **kwargs** – keyword arguments to pass to Pandas `read_csv` function

Example `ds.load_csv("./myfile.csv")`

load_django(*query: django query*)
Load the main dataframe from a django orm query

Parameters **query** (*django query*) – django query from a model

Example `ds.load_django(Mymodel.objects.all())`

load_django_(*query: django query*) → Ds
Returns a DataSwim instance from a django orm query

Parameters **query** (*django query*) – django query from a model

Returns a dataswim instance with data from a django query

Return type *Ds*

Example `ds2 = ds.load_django_(Mymodel.objects.all())`

load_excel (*filepath*, ***kwargs*)
Set the main dataframe with the content of an Excel file

Parameters

- **filepath** (*str*) – url of the csv file to load, can be absolute if it starts with `/` or relative if it starts with `./`
- **kwargs** – keyword arguments to pass to Pandas `read_excel` function

Example `ds.load_excel("./myfile.xlsx")`

load_h5 (*filepath*)
Load a Hdf5 file to the main dataframe

Parameters **filepath** (*str*) – url of the csv file to load, can be absolute if it starts with `/` or relative if it starts with `./`

Example `ds.load_h5("./myfile.hdf5")`

load_json (*path*, ***kwargs*)
Load data in the main dataframe from json

Parameters

- **filepath** (*str*) – url of the csv file to load, can be absolute if it starts with `/` or relative if it starts with `./`
- **kwargs** – keyword arguments to pass to Pandas `read_json` function

Example `ds.load_json("./myfile.json")`

lreg (*xcol*, *ycol*, *name='Regression'*)
Add a column to the main dataframe populted with the model's linear regression for a column

lreg_ (*label=None*, *style=None*, *opts=None*, *options={}*)
Get a linear regression chart

map_ (*lat*, *long*, *zoom=13*, *tiles='map'*)
Returns a map

marker (*lat*, *long*, *text*, *color=None*, *icon=None*)
Set the main map with a marker to the default map

marker_ (*lat*, *long*, *text*, *pmap*, *color=None*, *icon=None*)
Returns the map with a marker to the default map

mbar_ (*col*, *x=None*, *y=None*, *rsum=None*, *rmean=None*)
Splits a column into multiple series based on the column's unique values. Then visualize theses series in a chart. Parameters: column to split, x axis column, y axis column Optional: rsum="1D" to resample and sum data an rmean="1D" to mean the data

mcluster (*lat_col: str*, *lon_col: str*)
Add a markers cluster to the map

merge (*df: pandas.core.frame.DataFrame*, *on: str*, *how: str = 'outer'*, ***kwargs*)
Set the main dataframe from the current dataframe and the passed dataframe

Parameters

- **df** (*pd.DataFrame*) – the pandas dataframe to merge

- **on** (*str*) – param for `pd.merge`
- **how** (*str, optional*) – param for `pd.merge`, defaults to "outer"
- **kwargs** – keyword arguments for `pd.merge`

mfw_(*col*, *sw_lang='english'*, *limit=100*)
Returns a Dataswim instance with the most frequent words in a column exluding the most common stop words

mline_(*col*, *x=None*, *y=None*, *rsum=None*, *rmean=None*)
Splits a column into multiple series based on the column's unique values. Then visualize theses series in a chart. Parameters: column to split, x axis column, y axis column Optional: rsum="1D" to resample and sum data an rmean="1D" to mean the data

mline_point_(*col*, *x=None*, *y=None*, *rsum=None*, *rmean=None*)
Splits a column into multiple series based on the column's unique values. Then visualize theses series in a chart. Parameters: column to split, x axis column, y axis column Optional: rsum="1D" to resample and sum data an rmean="1D" to mean the data

mpoint_(*col*, *x=None*, *y=None*, *rsum=None*, *rmean=None*)
Splits a column into multiple series based on the column's unique values. Then visualize theses series in a chart. Parameters: column to split, x axis column, y axis column Optional: rsum="1D" to resample and sum data an rmean="1D" to mean the data

msg = None

msg_(*label*, **msg*)
Returns a message with a label

nan = None

nan_empty(*col: str*)
Fill empty values with NaN values

Parameters **col** (*str*) – name of the colum

Example `ds.nan_empty("mycol")`

ncontains(*column*, *value*)
Set the main dataframe instance to rows that do not contains a string value in a column

ndlayout_(*dataset*, *kdims*, *cols=3*)
Create a Holoview NdLayout from a dictionnary of chart objects

notebook = False

nowrange(*col: str*, *timeframe: str*)
Set the main dataframe with rows within a date range from now

Parameters

- **col** (*str*) – the column to use for range
- **timeframe** (*str*) – units are: S, H, D, W, M, Y

example: `ds.nowrange("Date", "3D")`

nowrange_(*col: str*, *timeframe: str*) → Ds
Returns a Dataswim instance with rows within a date range from now

Parameters

- **col** (*str*) – the column to use for range
- **timeframe** (*str*) – units are: S, H, D, W, M, Y

Returns [description]

Return type [type]

example: `ds2 = ds.nowrange_("Date", "3D")`

ok(**msg*)
Prints a message with an ok prefix

one()
Shows one row of the dataframe and the field names wiht count

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.one()`

opt(*name*, *value*)
Add or update one option

opts(*dictobj*)
Add or update options

pivot(*index*, ***kwargs*)
Pivots a dataframe

point_(*label=None*, *style=None*, *opts=None*, *options={}*)
Get a point chart

point_num_(*label=None*, *style=None*, *opts=None*)
Get an Altair point + number marks chart

progress(**msg*)
Prints a progress message

quants_(*inf*, *sup*, *chart_type='point'*, *color='green'*)
Draw a chart to visualize quantiles

query(*q: str*) → dataset.util.ResultIter
Query the database

Parameters **q** (*str*) – the query to perform

Returns a dictionary with the query results

Return type dataset.util.ResultIter

quiet = False

radar_(*label=None*, *style=None*, *opts=None*, *options={}*)
Get a radar chart

raenc(*key*)
Remove an entry from the altair encoding dict

raencs()
Reset the altair encoding dict

ratio(*col: str*, *ratio_col: str = 'Ratio'*)
Add a column whith the percentages ratio from a column

Parameters

- **col** (*str*) – column to calculate ratio from
- **ratio_col** – new ratio column name, defaults to "Ratio"

- **ratio_col** – str, optional

Example `ds2 = ds.ratio("Col 1")`

rcolor()
Reset the color to the base color

relation(*table: str*, *origin_field: str*, *search_field: str*, *destination_field: str = None*, *id_field: str = 'id'*)
Add a column to the main dataframe from a relation foreign key

Parameters

- **table** (*str*) – the table to select from
- **origin_field** (*str*) – the column name in the origin table to search from, generally an id column
- **search_field** (*str*) – the column name in the foreign table
- **destination_field** (*str, optional*) – name of the column to be created with the data in the datframe, defaults to None, will be named as the origin_field if not provided
- **id_field** (*str, optional*) – name of the primary key to use, defaults to "id"

example: `ds.relation("product", "category_id", "name")`

relation_(*table: str*, *origin_field: str*, *search_field: str*, *destination_field=None*, *id_field='id'*) → pandas.core.frame.DataFrame

Returns a DataSwim instance with a column filled from a relation foreign key

Parameters

- **table** (*str*) – the table to select from
- **origin_field** (*str*) – the column name in the origin table to search from, generally an id column
- **search_field** (*str*) – the column name in the foreign table
- **destination_field** (*str, optional*) – name of the column to be created with the data in the datframe, defaults to None, will be named as the origin_field if not provided
- **id_field** (*str, optional*) – name of the primary key to use, defaults to "id"

Returns a pandas DataFrame

Return type DataFrame

rename(*source_col: str*, *dest_col: str*)
Renames a column in the main dataframe

Parameters

- **source_col** (*str*) – name of the column to rename
- **dest_col** (*str*) – new name of the column

Example `ds.rename("Col 1", "New col")`

replace(*col: str*, *searchval: str*, *replaceval: str*)
Replace a value in a column in the main dataframe

Parameters

- **col** (*str*) – column name

- **searchval** (*str*) – value to replace
- **replaceval** (*str*) – new value

Example `ds.replace("mycol", "value", "new_value")`

report_engines = []

report_path = None

reports = []

residual_(*label=None*, *style=None*, *opts=None*)
Returns a Seaborn models residuals chart

restore()
Restore the main dataframe

reverse()
Reverses the main dataframe order

Example `ds.reverse()`

rmean(*time_period: str*, *num_col: str = 'Number'*, *dateindex: str = None*)
Resample and add a mean column the main dataframe to a time period

Parameters

- **time_period** – unit + period: periods are Y, M, D, H, Min, S
- **time_period** – str
- **num_col** – number of the new column, defaults to "Number"
- **num_col** – str, optional
- **dateindex** – column name to use as date index, defaults to None

Example `ds.rmean("1Min")`

rmean_(*time_period: str*, *num_col: str = 'Number'*, *dateindex: str = None*)
Resample and add a mean column the main dataframe to a time period and returns a new Ds instance

Parameters

- **time_period** – unit + period: periods are Y, M, D, H, Min, S
- **time_period** – str
- **num_col** – number of the new column, defaults to "Number"
- **num_col** – str, optional
- **dateindex** – column name to use as date index, defaults to None

Example `ds.rmean_("1Min")`

ropt(*name*)
Remove one option

ropts()
Reset the chart options

roundvals(*col: str*, *precision: int = 2*)
Round floats in a column. Numbers are going to be converted to floats if they are not already

Parameters

- **col** (*str*) – column name

- **precision** – float precision, defaults to 2
- **precision** – int, optional

Example `ds.roundvals("mycol")`

rstyle(*name*)
Remove one style

rstyles()
Reset the chart options

rsum(*time_period: str*, *num_col: str = 'Number'*, *dateindex: str = None*)
Resample and add a sum the main dataframe to a time period

Parameters

- **time_period** – unit + period: periods are Y, M, D, H, Min, S
- **time_period** – str
- **num_col** – number of the new column, defaults to "Number"
- **num_col** – str, optional
- **dateindex** – column name to use as date index, defaults to None
- **dateindex** – str, optional

Example `ds.rsum("1D")`

rsum_(*time_period: str*, *num_col: str = 'Number'*, *dateindex: str = None*)
Resample and add a sum the main dataframe to a time period and returns a new Ds instance

Parameters

- **time_period** – unit + period: periods are Y, M, D, H, Min, S
- **time_period** – str
- **num_col** – number of the new column, defaults to "Number"
- **num_col** – str, optional
- **dateindex** – column name to use as date index, defaults to None
- **dateindex** – str, optional

Example `ds.rsum_("1D")`

rule_(*label=None*, *style=None*, *opts=None*, *options={}*)
Get a rule chart

sarea_(*col*, *x=None*, *y=None*, *rsum=None*, *rmean=None*)
Get an stacked area chart

sbar_(*stack_index=None*, *label=None*, *style=None*, *opts=None*, *options={}*)
Get a stacked bar chart

scolor()
Set a unique color from a serie

scommit()

sconnect(*url: str*)

seaborn_bar_(*label=None*, *style=None*, *opts=None*)
Get a Seaborn bar chart

show(*rows: int = 5*, *dataframe: pandas.core.frame.DataFrame = None*) → pandas.core.frame.DataFrame
Display info about the dataframe

Parameters

- **rows** – number of rows to show, defaults to 5
- **rows** – int, optional
- **dataframe** – a pandas dataframe, defaults to None
- **dataframe** – pd.DataFrame, optional

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.show()`

size(*val*)
Change the chart's point size

sline_(*window_size=5*, *y_label='Moving average'*, *chart_label=None*)
Get a moving average curve chart to smooth between points

sort(*col: str*)
Sorts the main dataframe according to the given column

Parameters **col** (*str*) – column name

Example `ds.sort("Col 1")`

split_(*col: str) -> list(Ds*)
Split the main dataframe according to a column's unique values and return a dict of dataswim instances

Returns list of dataswim instances

Return type list(*Ds*)

Example `dss = ds.slit_("Col 1")`

sq_(*query: str*)

sqm_(*query: str*, *values*)

square_(*label=None*, *style=None*, *opts=None*, *options={}*)
Get a square chart

stack(*slug*, *chart_obj=None*, *title=None*)
Get the html for a chart and store it

start(**msg*)
Prints an start message

start_time = None

static_path = None

status(**msg*)
Prints a status message

strip(*col: str*)
Remove leading and trailing white spaces in a column's values

Parameters **col** (*str*) – name of the column

Example `ds.strip("mycol")`

strip_cols()
Remove leading and trailing white spaces in columns names

Example `ds.strip_cols()`

style(*name*, *value*)
Add or update one style

styles(*dictobj*)
Add or update styles

subset(**args*)
Set the main dataframe to a subset based in positions Select a subset of the main dataframe based on position: ex: ds.subset(0,10) or ds.subset(10) is equivalent: it starts at the first row if only one argument is provided

subset_(**args*)
Returns a Dataswim instance with a subset data based in positions Select a subset of the main dataframe based on position: ex: ds.subset(0,10) or ds.subset(10) is equivalent: it starts at the first row if only one argument is provided

subtitle(*txt*)
Prints a subtitle for pipelines

sum_(*col: str*) → float
Returns the sum of all values in a column

Parameters **col** (*str*) – column name

Returns sum of all the column values

Return type float

Example `sum = ds.sum_("Col 1")`

table(*name: str*)
Display info about a table: number of rows and columns

Parameters **name** (*str*) – name of the table

Example `tables = ds.table("mytable")`

tables()
Print the existing tables in a database

Example `ds.tables()`

tables_() → list
Return a list of the existing tables in a database

Returns list of the table names

Return type list

Example `tables = ds.tables_()`

tail(*rows: int = 5*)
Returns the main dataframe's tail

Parameters

- **rows** – number of rows to print, defaults to 5
- **rows** – int, optional

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.tail()`

text_ (*label=None*, *style=None*, *opts=None*)
Get an Altair text marks chart

tick_ (*label=None*, *style=None*, *opts=None*, *options={}*)
Get an tick chart

timestamps (*col: str*, ***kwargs*)
" Add a timestamps column from a date column

Parameters

- **col** (*str*) – name of the timestamps column to add
- ****kwargs** (*optional*) – keyword arguments for `pd.to_datetime`

Example `ds.timestamps("mycol")`

title (*txt*)
Prints a title for pipelines

tmarker (*lat*, *long*, *text*, *color=None*, *icon=None*, *style=None*)
Returns the map with a text marker to the default map

tmarker_ (*lat*, *long*, *text*, *pmap*, *color=None*, *icon=None*, *style=None*)
Returns the map with a text marker to the default map

to_csv (*filepath: str*, *index: bool = False*, ***kwargs*)
Write the main dataframe to a csv file

Parameters

- **filepath** (*str*) – path of the file to save
- **index** – [description], defaults to False
- **index** – bool, optional
- ***args** – arguments to pass to `pd.to_csv`

Example `ds.to_csv_("myfile.csv", header=false)`

to_db (*table: str*, *dtypes: List[sqlalchemy.sql.sqltypes.SchemaType] = None*)
Save the main dataframe to the database

Parameters

- **table** (*str*) – the table to create
- **dtypes** (*List[SchemaType], optional*) – SqlAlchemy columns type, defaults to None, will be infered if not provided

to_excel (*filepath: str*, *title: str*)
Write the main dataframe to an Excell file

Parameters

- **filepath** (*str*) – path of the Excel file to write
- **title** (*str*) – Title of the stylesheet

Example `ds.to_excel_("./myfile.xlsx", "My data")`

to_file (*slug*, *folderpath=None*, *header=None*, *footer=None*)
Writes the html report to a file from the report stack

to_files (*folderpath=None*)
Writes the html report to one file per report

to_float (*col: str*, ***kwargs*)
Convert colums values to float

Parameters

- **col** (*str, at least one*) – name of the colum
- ****kwargs** (*optional*) – keyword arguments for `df.astype`

Example `ds.to_float("mycol1")`

to_hdf5 (*filepath: str*)
Write the main dataframe to Hdf5 file

Parameters **filepath** (*str*) – path where to save the file

Example `ds.to_hdf5_("./myfile.hdf5")`

to_html_ () → str
Convert the main dataframe to html

Returns html data

Return type str

Example `ds.to_html_()`

to_int (**cols*, ***kwargs*)
Convert some column values to integers

Parameters

- ***cols** (*str, at least one*) – names of the colums
- ****kwargs** (*optional*) – keyword arguments for `pd.to_numeric`

Example `ds.to_int("mycol1", "mycol2", errors="coerce")`

to_javascript_ (*table_name: str = 'data'*) → str
Convert the main dataframe to javascript code

Parameters

- **table_name** – javascript variable name, defaults to "data"
- **table_name** – str, optional

Returns a javascript constant with the data

Return type str

Example `ds.to_javastript_("myconst")`

to_json_ () → str
Convert the main dataframe to json

Returns json data

Return type str

Example `ds.to_json_()`

to_markdown_ () → str
Convert the main dataframe to markdown

Returns markdown data

Return type str

Example `ds.to_markdown_()`

to_numpy_(*table_name: str = 'data'*) → numpy.array
Convert the main dataframe to a numpy array

Parameters

- **table_name** – name of the python variable, defaults to "data"
- **table_name** – str, optional

Returns a numpy array

Return type np.array

Example `ds.to_numpy_("myvar")`

to_python_(*table_name: str = 'data'*) → list
Convert the main dataframe to python a python list

Parameters

- **table_name** – python variable name, defaults to "data"
- **table_name** – str, optional

Returns a python list of lists with the data

Return type str

Example `ds.to_python_("myvar")`

to_records_() → dict
Returns a list of dictionary records from the main dataframe

Returns a python dictionnary with the data

Return type str

Example `ds.to_records_()`

to_rst_() → str
Convert the main dataframe to restructured text

Returns rst data

Return type str

Example `ds.to_rst_()`

to_type(*dtype: type*, **cols*, ***kwargs*)
Convert colums values to a given type in the main dataframe

Parameters

- **dtype** (*type*) – a type to convert to: ex: `str`
- ***cols** (*str, at least one*) – names of the colums
- ****kwargs** (*optional*) – keyword arguments for `df.astype`

Example `ds.to_type(str, "mycol")`

trimiquants(*col: str*, *inf: float*)
Remove superior and inferior quantiles from the dataframe

Parameters

- **col** (*str*) – column name
- **inf** (*float*) – inferior quantile

Example `ds.trimiquants("Col 1", 0.05)`

trimquants(*col: str*, *inf: float*, *sup: float*)
Remove superior and inferior quantiles from the dataframe

Parameters

- **col** (*str*) – column name
- **inf** (*float*) – inferior quantile
- **sup** (*float*) – superior quantile

Example `ds.trimquants("Col 1", 0.01, 0.99)`

trimsquants(*col: str*, *sup: float*)
Remove superior quantiles from the dataframe

Parameters

- **col** (*str*) – column name
- **sup** (*float*) – superior quantile

Example `ds.trimsquants("Col 1", 0.99)`

types_(*col: str*) → pandas.core.frame.DataFrame
Display types of values in a column

Parameters **col** (*str*) – column name

Returns a pandas dataframe

Return type pd.DataFrame

Example `ds.types_("Col 1")`

unique_(*col: str*) → list
Returns unique values in a column

Parameters **col** (*str*) – the column to select from

Returns a list of unique values in the column

Return type list

update_table(*table: str, pks: List[str] = ['id'], mirror: bool = True*)
Update records in a database table from the main dataframe

Parameters

- **table** (*str*) – table to update
- **pks** (*List[str], optional :param mirror: delete the rows not in the new dataset*) – if rows with matching pks exist they will be updated, otherwise a new row is inserted in the table, defaults to ["id"]

upsert(*table: str, record: dict, create_cols: bool = False, dtypes: List[sqlalchemy.sql.sqltypes.SchemaType] = None, pks: List[str] = ['id']*)
Upsert a record in a table

Parameters

- **table** (*str*) – the table to upsert into

- **record** (*dict*) – dictionary with the data to upsert
- **create_cols** (*bool, optional*) – create the columns if it doesn't exist, defaults to False
- **dtypes** (*List[SchemaType], optional*) – list of SqlAlchemy column types, defaults to None
- **pks** (*List[str], optional*) – if rows with matching pks exist they will be updated, otherwise a new row is inserted in the table, defaults to ["id"]

version = '0.6.0'

warning(**msg*)
Prints a warning

width(*val*)
Change the chart's width

wunique_(*col*)
Weight unique values: returns a dataframe with a count of unique values

x = None

y = None

zero_nan(**cols*)
Converts zero values to nan values in selected columns

Parameters ***cols** (*str, at least one*) – names of the colums

Example `ds.zero_nan("mycol1", "mycol2")`

Python Module Index

d

Index

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z