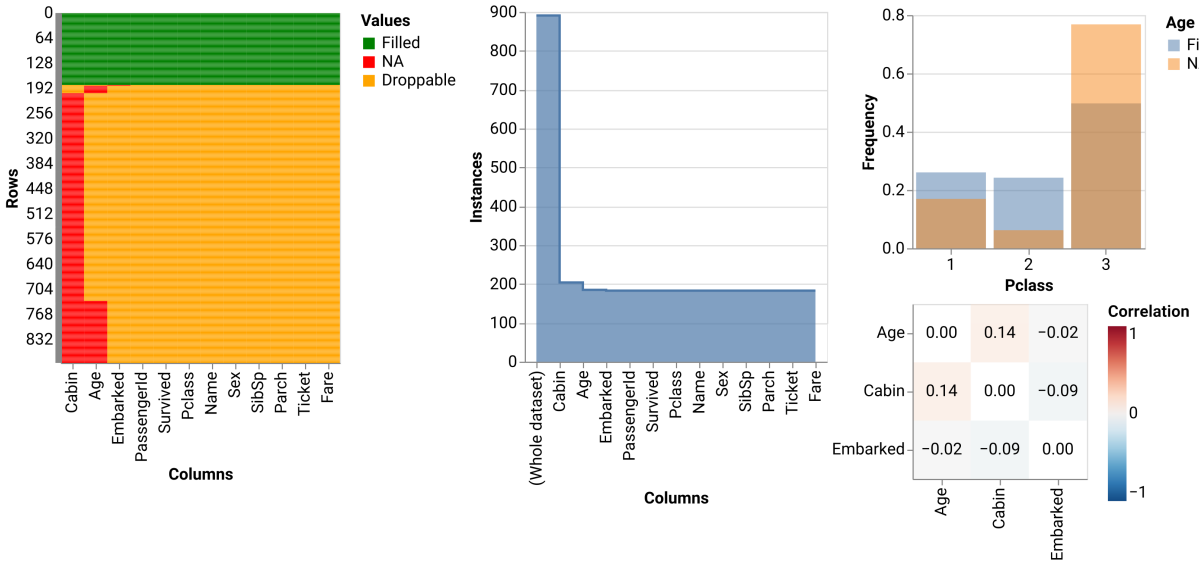
scikit-na

Maksim Terpilowski

Mar 11, 2022

USAGE

scikit-na is a Python package for missing data (NA) analysis. The package includes many functions for statistical analysis, modeling, and data visualization. The latter is done using two packages — matplotlib and Altair.



CHAPTER

ONE

FEATURES

- Interactive report (based on ipywidgets)
- Descriptive statistics
- Hypotheses tests
- Regression modelling
- Data visualization

1.1 Installation

Package can be installed from PyPi:

```
pip install scikit-na
```

or from this repo:

```
pip install git+https://github.com/maximtrp/scikit-na.git
```

1.1.1 Requirements

- NumPy
- Statsmodels
- Seaborn
- Pandas
- Altair
- Matplotlib
- ipywidgets

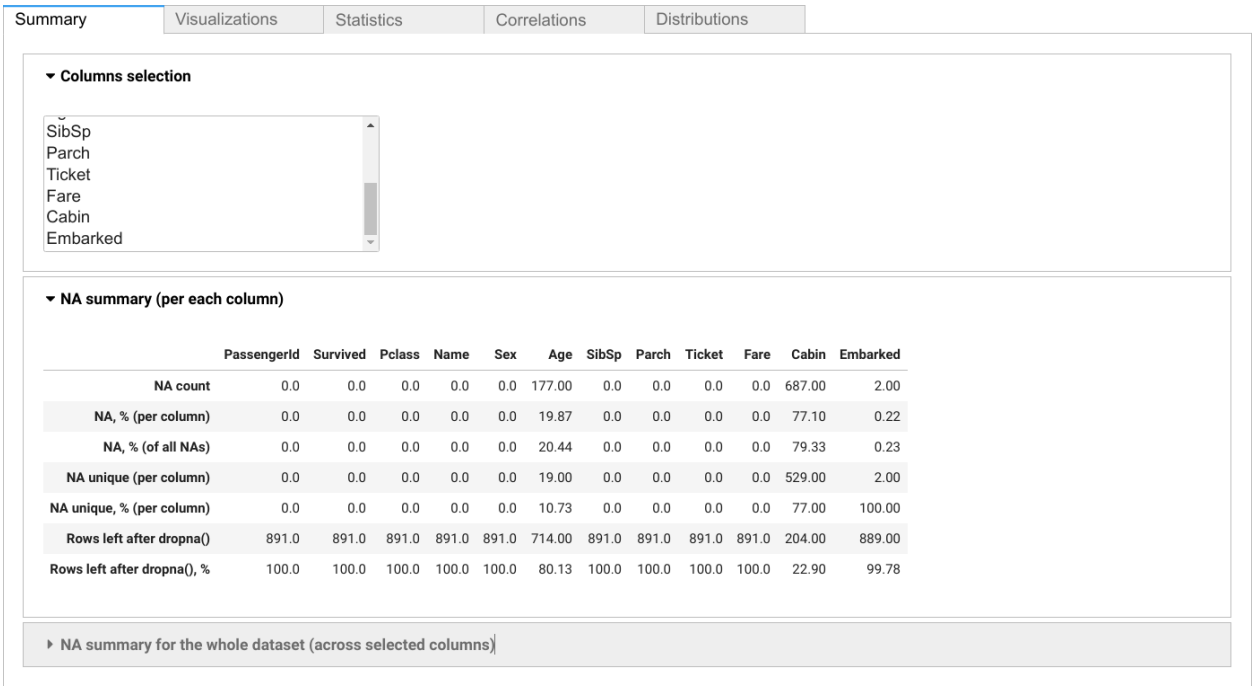
1.2 Interactive report

Report interface is based on ipywidgets. It can help you quickly and interactively explore NA values in your dataset, view patterns, calculate statistics, show relevant figures and tables.

To begin, just load the data and pass your DataFrame to `scikit_na.report()` function:

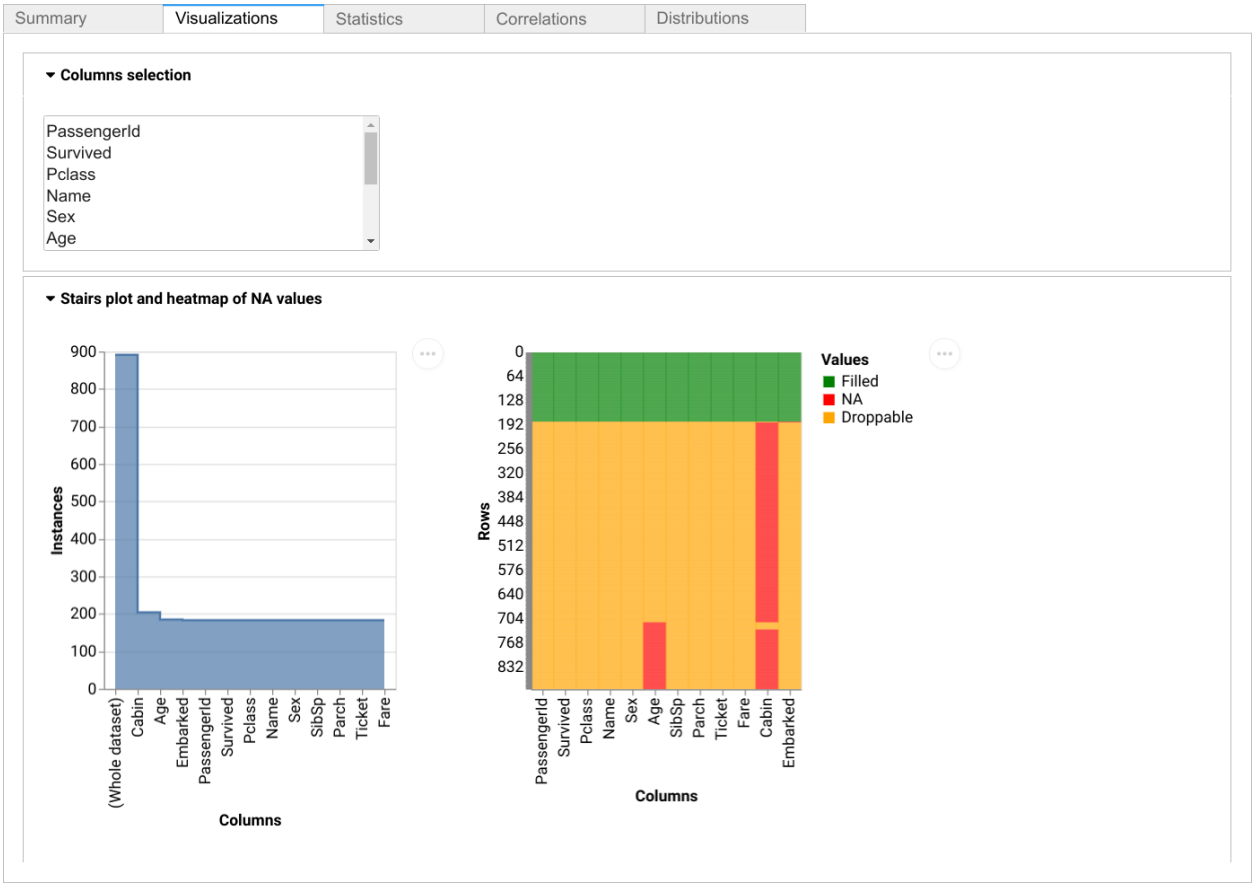
```
import pandas as pd
import scikit_na as na

data = pd.read_csv('titanic_dataset.csv')
na.report(data)
```

1.2.1 Summary tab

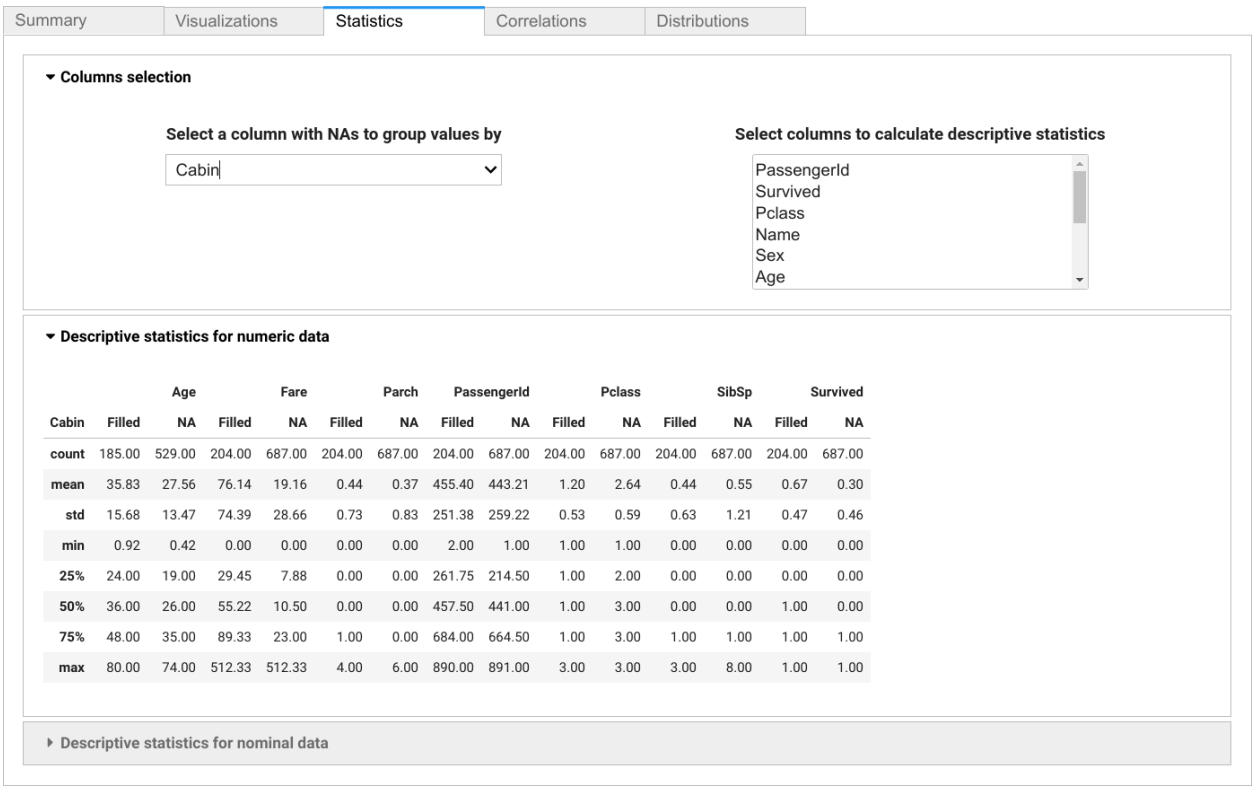


	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
NA count	0.0	0.0	0.0	0.0	0.0	177.00	0.0	0.0	0.0	0.0	687.00	2.00
NA, % (per column)	0.0	0.0	0.0	0.0	0.0	19.87	0.0	0.0	0.0	0.0	77.10	0.22
NA, % (of all NAs)	0.0	0.0	0.0	0.0	0.0	20.44	0.0	0.0	0.0	0.0	79.33	0.23
NA unique (per column)	0.0	0.0	0.0	0.0	0.0	19.00	0.0	0.0	0.0	0.0	529.00	2.00
NA unique, % (per column)	0.0	0.0	0.0	0.0	0.0	10.73	0.0	0.0	0.0	0.0	77.00	100.00
Rows left after dropna()	891.0	891.0	891.0	891.0	891.0	714.00	891.0	891.0	891.0	891.0	204.00	889.00
Rows left after dropna(), %	100.0	100.0	100.0	100.0	100.0	80.13	100.0	100.0	100.0	100.0	22.90	99.78

1.2.2 Visualization tab

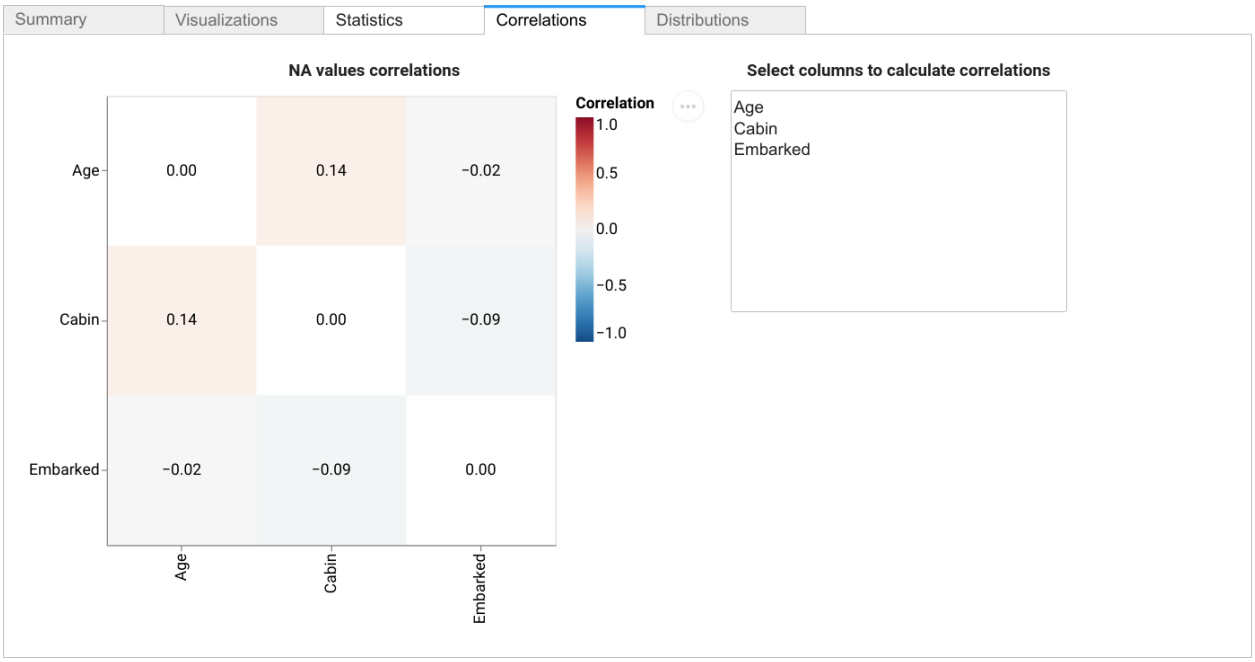


1.2.3 Statistics tab

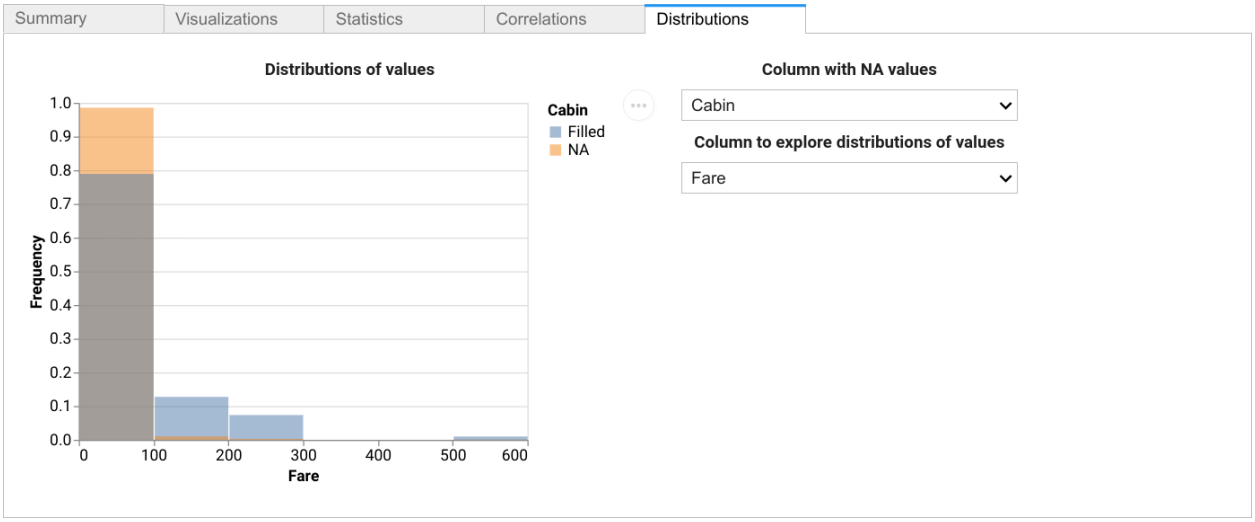


	Age		Fare		Parch		PassengerId		Pclass		SibSp		Survived	
Cabin	Filled	NA	Filled	NA	Filled	NA	Filled	NA	Filled	NA	Filled	NA	Filled	NA
count	185.00	529.00	204.00	687.00	204.00	687.00	204.00	687.00	204.00	687.00	204.00	687.00	204.00	687.00
mean	35.83	27.56	76.14	19.16	0.44	0.37	455.40	443.21	1.20	2.64	0.44	0.55	0.67	0.30
std	15.68	13.47	74.39	28.66	0.73	0.83	251.38	259.22	0.53	0.59	0.63	1.21	0.47	0.46
min	0.92	0.42	0.00	0.00	0.00	0.00	2.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00
25%	24.00	19.00	29.45	7.88	0.00	0.00	261.75	214.50	1.00	2.00	0.00	0.00	0.00	0.00
50%	36.00	26.00	55.22	10.50	0.00	0.00	457.50	441.00	1.00	3.00	0.00	0.00	1.00	0.00
75%	48.00	35.00	89.33	23.00	1.00	0.00	684.00	664.50	1.00	3.00	1.00	1.00	1.00	1.00
max	80.00	74.00	512.33	512.33	4.00	6.00	890.00	891.00	3.00	3.00	3.00	8.00	1.00	1.00

1.2.4 Correlations tab



1.2.5 Distributions tab



1.3 Statistics

In missing data analysis, an important step is to calculate simple descriptive and aggregating statistics of missing and non-missing data for each column and for the whole dataset. *Scikit-na* attempts to provide useful functions for such operations.

1.3.1 Summary

We will use Titanic dataset that contains NA values in three columns: *Age*, *Cabin*, and *Embarked*.

Per column

To get a simple summary per each column, we will load a dataset using `pandas` and pass it to `summary()` function. The latter supports subsetting a dataset with `columns` argument. And we will make use of it to cut the width of the results table.

```
import scikit_na as na
import pandas as pd

data = pd.read_csv('titanic_dataset.csv')

# Excluding three columns without NA to fit the table here
na.summary(data, columns=data.columns.difference(['SibSp', 'Parch', 'Ticket']))
```

	Age	Cabin	Em-barked	Fare	Name	Passen-gerId	Pclass	Sex	Sur-vived
NA count	177	687	2	0	0	0	0	0	0
NA, % (per column)	19.87	77.1	0.22	0	0	0	0	0	0
NA, % (of all NAs)	20.44	79.33	0.23	0	0	0	0	0	0
NA unique (per column)	19	529	2	0	0	0	0	0	0
NA unique, % (per column)	10.73	77	100	0	0	0	0	0	0
Rows left after dropna()	714	204	889	891	891	891	891	891	891
Rows left after dropna(), %	80.13	22.9	99.78	100	100	100	100	100	100

Those measures were meant to be self-explanatory:

- *NA count* is the number of NA values in each column.
- *NA unique* is the number of NA values in each column that are unique for it, i.e. do not intersect with NA values in the other columns (or that will remain in dataset if we drop NA values in the other columns).
- *Rows left after dropna()* shows how many rows will be left in a dataset if we apply `pandas.Series.dropna()` method to each column.

Whole dataset

By default, `summary()` function returns the results for each column. To get the summary of missing data for the whole DataFrame, we should set `per_column` argument to `False`.

```
na.summary(data, per_column=False)
```

	dataset
Total columns	12
Total rows	891
Rows with NA	708
Rows without NA	183
Total cells	10692
Cells with NA	866
Cells with NA, %	8.1
Cells with non-missing data	9826
Cells with non-missing data, %	91.9

1.3.2 Descriptive statistics

The next step is to calculate descriptive statistics for columns with quantitative and qualitative data. First, let's filter the columns by data types:

```
# Presumably, qualitative data, needs checking
cols_nominal = data.columns[data.dtypes == object]

# Quantitative data
cols_numeric = data.columns[(data.dtypes == float) | (data.dtypes == int)]
```

We should also specify a column with missing values (NAs) that will be used to split the data in the selected columns into two groups: NA (missing) and Filled (non-missing).

Qualitative data

```
na.describe(data, columns=cols_nominal)
```

	Embarked		Name		Sex		Ticket	
Cabin	Filled	NA	Filled	NA	Filled	NA	Filled	NA
count	202	687	204	687	204	687	204	687
unique	3	3	204	687	2	2	142	549
top	S	S	Levy, Mr. Rene Jacques	Nasser, Mr. Nicholas	male	male	113760	347082
freq	129	515	1	1	107	470	4	7

Let's check the results by hand:

```
data.groupby(
  data['Cabin'].isna().replace({False: 'Filled', True: 'NA'}))['Sex']\
.value_counts()
```

Cabin	Sex	Count
Filled	male	107
	female	97
NA	male	470
	female	217

Here we take *Cabin* column, encode missing/non-missing data as Filled/NA, and then use it to group and count values in *Sex* column: among the passengers with missing *cabin* data, 470 were males, while 217 were females.

Quantitative data

Now, let's look at the statistics calculated for the numeric data:

```
# Selecting just two columns
na.describe(data, columns=['Age', 'Fare'], col_na='Cabin')
```

	Age		Fare	
Cabin	Filled	NA	Filled	NA
count	185	529	204	687
mean	35.8293	27.5553	76.1415	19.1573
std	15.6794	13.4726	74.3917	28.6633
min	0.92	0.42	0	0
25%	24	19	29.4531	7.8771
50%	36	26	55.2208	10.5
75%	48	35	89.3282	23
max	80	74	512.329	512.329

The mean *age* of passengers with missing *cabin* data was 27.6 years.

1.4 Regression modeling

The presence of missing data can be used in regression modeling as a dependent variable encoded as `0` and `1`.

For demonstration purposes, we will use Titanic dataset. Let's create a regression model with *Age* as a dependent variable and *Fare*, *Parch*, *Pclass*, *SibSp*, *Survived* as independent variables. Internally, `pandas.Series.isna()` method is called on *Age* column, and the resulting boolean values are converted to integers (`True` and `False` become 1 and `0`). Data preprocessing is totally up to you!

Currently, `scikit_na.model()` function runs a logistic model using statsmodels package as a backend.

```
import pandas as pd
import scikit_na as na

# Loading data
data = pd.read_csv("titanic_dataset.csv")

# Selecting columns with numeric data
# Dropping "PassengerId" column
subset = data.loc[:, data.dtypes != object].drop(columns=['PassengerId'])

# Fitting a model
model = na.model(subset, col_na='Age')
model.summary()
```

```
Optimization terminated successfully.
Current function value: 0.467801
Iterations 7
                          Logit Regression Results
==============================================================================
Dep. Variable:                    Age   No. Observations:                  891
Model:                          Logit   Df Residuals:                      885
Method:                           MLE   Df Model:                            5
Date:                Sat, 05 Jun 2021   Pseudo R-squ.:                 0.06164
Time:                        17:51:31   Log-Likelihood:                -416.81
converged:                       True   LL-Null:                       -444.19
Covariance Type:            nonrobust   LLR p-value:                 1.463e-10
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
(intercept)    -2.7294      0.429     -6.369      0.000      -3.569      -1.890
Fare            0.0010      0.003      0.376      0.707      -0.004       0.006
Parch          -0.8874      0.223     -3.984      0.000      -1.324      -0.451
Pclass          0.5953      0.147      4.046      0.000       0.307       0.884
SibSp           0.2548      0.095      2.684      0.007       0.069       0.441
Survived       -0.1026      0.198     -0.519      0.604      -0.490       0.285
==============================================================================
```

1.5 Data visualization

```
import pandas as pd
import scikit_na as na
data = pd.read_csv('../../_tests/data/titanic_dataset.csv')
```

1.5.1 Heatmap

NA values

Missing data can be visualized on a heatmap to quickly grasp its patterns. We will be using Altair + Vega backend. To plot a heatmap of NAs, simply pass your DataFrame to `scikit_na.altair.plot_heatmap()` function.

Droppables are those values that will be dropped if we simply use `pandas.DataFrame.dropna()` on the *whole dataset*. By default, columns are sorted by the number of NA values.

```
na.altair.plot_heatmap(data)
```

```
alt.Chart(...)
```

Correlations

Correlations can be plotted using `scikit_na.altair.plot_corr()` function. Under the hood, it calls `scikit_na.correlate()` function with your input DataFrame as the first argument:

```
na.altair.plot_corr(data).properties(width=125, height=125)
```

```
alt.LayerChart(...)
```

1.5.2 Stairs plot

Stairs plot is a useful visualization of a dataset shrinkage on applying `pandas.Series.dropna()` method to each column sequentially (sorted by the number of NA values, by default):

```
na.altair.plot_stairs(data)
```

```
alt.Chart(...)
```

After dropping all NAs in *Cabin* column, we are left with 21 more NAs (in *Age* and *Embarked* columns). This plot also shows tooltips with exact numbers of NA values that are dropped per each column.

```
3 na.altair.plot_stairbars(data)
```

```
3 alt.Chart(...)
```

1.5.3 Histogram

Plotting a nice histogram may require configuring additional parameters.

```
chart = na.altair.plot_hist(data, col='Pclass', col_na='Age')\
    .properties(width=200, height=200)
chart.configure_axisX(labelAngle = 0)
```

```
alt.Chart(...)
```

1.5.4 Density plot

```
chart = na.altair.plot_kde(data, col='Age', col_na='Cabin')\
    .properties(width=200, height=200)
chart.configure_axisX(labelAngle = 0)
```

```
alt.Chart(...)
```

1.6 scikit_na

scikit_na.**correlate**(*data: pandas.core.frame.DataFrame*, *columns: Optional[Sequence] = None*, *drop: bool = True*, ***kwargs*) → pandas.core.frame.DataFrame

Calculate correlations between columns in terms of NA values.

Parameters

- **data** (*DataFrame*) – Input data.
- **columns** (*Optional[List, ndarray, Index] = None*) – Columns names.
- **drop** (*bool = True, optional*) – Drop columns without NA values.
- **kwargs** (*dict, optional*) – Keyword arguments passed to `pandas.DataFrame.corr()` method.

Returns Correlation values.

Return type DataFrame

scikit_na.**describe**(*data: pandas.core.frame.DataFrame*, *col_na: str*, *columns: Optional[Sequence] = None*, *na_mapping: Optional[dict] = None*) → pandas.core.frame.DataFrame

Describe data grouped by a column with NA values.

Parameters

- **data** (*DataFrame*) – Input data.
- **col_na** (*str*) – Column with NA values to group the other data by.
- **columns** (*Optional[Sequence]*) – Columns to calculate descriptive statistics on.
- **na_mapping** (*dict, optional*) – Dictionary with NA mappings. By default, it is {True: "NA", False: "Filled"}.

Returns Descriptive statistics (mean, median, etc.).

Return type DataFrame

scikit_na.**model**(*data: pandas.core.frame.DataFrame*, *col_na: str*, *columns: Optional[Sequence] = None*, *intercept: bool = True*, *fit_kws: Optional[dict] = None*, *logit_kws: Optional[dict] = None*)

Logistic regression modeling.

Fit a logistic regression model to NA values encoded as 0 (non-missing) and 1 (NA) in column *col_na* with predictors passed with *columns* argument. Statsmodels package is used as a backend for model fitting.

Parameters

- **data** (*DataFrame*) – Input data.
- **col_na** (*str*) – Column with NA values to use as a dependent variable.
- **columns** (*Optional[Sequence]*) – Columns to use as independent variables.
- **intercept** (*bool, optional*) – Fit intercept.
- **fit_kws** (*dict, optional*) – Keyword arguments passed to *fit()* method of model.
- **logit_kws** (*dict, optional*) – Keyword arguments passed to `statsmodels.discrete.discrete_model.Logit()` class.

Returns Model after applying *fit* method.

Return type statsmodels.discrete.discrete_model.BinaryResultsWrapper

Example

```
>>> import scikit_na as na
>>> model = na.model(
...     data,
...     col_na='column_with_NAs',
...     columns=['age', 'height', 'weight'])
>>> model.summary()
```

scikit_na.**report**(*data: pandas.core.frame.DataFrame*, *columns: Optional[Sequence[str]] = None*, *layout: Optional[ipywidgets.widgets.widget_layout.Layout] = None*, *round_dec: int = 2*, *corr_kws: Optional[dict] = None*, *heat_kws: Optional[dict] = None*, *dist_kws: Optional[dict] = None*)

Interactive report.

Parameters

- **data** (*DataFrame*) – Input data.
- **columns** (*Optional[Sequence[str]], optional*) – Columns names.
- **layout** (*widgets.Layout, optional*) – Layout object for use in GridBox.
- **round_dec** (*int, optional*) – Number of decimals for rounding.
- **corr_kws** (*dict, optional*) – Keyword arguments passed to *scikit_na.altair.plot_corr()*.
- **heat_kws** (*dict, optional*) – Keyword arguments passed to *scikit_na.altair.plot_heatmap()*.
- **hist_kws** (*dict, optional*) – Keyword arguments passed to *scikit_na.altair.plot_hist()*.

Returns Interactive report with multiple tabs.

Return type widgets.Tab

scikit_na.**stairs**(*data: pandas.core.frame.DataFrame*, *columns: Optional[Sequence] = None*, *xlabel: str = 'Columns'*, *ylabel: str = 'Instances'*, *tooltip_label: str = 'Size difference'*, *dataset_label: str = '(Whole dataset)'*)

DataFrame shrinkage on cumulative `pandas.DataFrame.dropna()`.

Parameters

- **data** (*DataFrame*) – Input data.
- **columns** (*Optional[Sequence], optional*) – Columns names.
- **xlabel** (*str, optional*) – X axis label.
- **ylabel** (*str, optional*) – Y axis label.
- **tooltip_label** (*str, optional*) – Tooltip label.
- **dataset_label** (*str, optional*) – Label for a whole dataset.

Returns Dataset shrinkage results after cumulative `pandas.DataFrame.dropna()`.

Return type DataFrame

scikit_na.**summary**(*data: pandas.core.frame.DataFrame*, *columns: Optional[Sequence] = None*, *per_column: bool = True*, *round_dec: int = 2*) → pandas.core.frame.DataFrame

Summary statistics on NA values.

Parameters

- **data** (*DataFrame*) – Data object.
- **columns** (*Optional[Sequence]*) – Columns or indices to observe.
- **per_column** (*bool = True, optional*) – Show stats per each selected column.
- **round_dec** (*int = 2, optional*) – Number of decimals for rounding.

Returns Summary on NA values in the input data.

Return type DataFrame

scikit_na.**test_hypothesis**(*data: pandas.core.frame.DataFrame*, *col_na: str*, *test_fn: callable*, *test_kws: Optional[dict] = None*, *columns: Optional[Union[Sequence[str], Dict[str, callable]]] = None*, *dropna: bool = True*) → Dict[str, object]

Test a statistical hypothesis.

This function can be used to find evidence against missing completely at random (MCAR) mechanism by comparing two samples grouped by missingness in another column.

Parameters

- **data** (*DataFrame*) – Input data.
- **col_na** (*str*) – Column to group values by. `pandas.Series.isna()` method is applied before grouping.
- **columns** (*Optional[Union[Sequence[str], Dict[str, callable]]]*) – Columns to test hypotheses on.
- **test_fn** (*callable, optional*) – Function to test hypothesis on NA/non-NA data. Must be a two-sample test function that accepts two arrays and (optionally) keyword arguments such as `scipy.stats.mannwhitneyu()`.
- **test_kws** (*dict, optional*) – Keyword arguments passed to *test_fn* function.
- **dropna** (*bool = True, optional*) – Drop NA values in two samples before running a hypothesis test.

Returns Dictionary with tests results as *column* => test function output.

Return type Dict[str, object]

Example

```
>>> import scikit_na as na
>>> import pandas as pd
>>> data = pd.read_csv('some_dataset.csv')
>>> # Simple example
>>> na.test_hypothesis(
...     data,
...     col_na='some_column_with_NAs',
...     columns=['age', 'height', 'weight'],
...     test_fn=ss.mannwhitneyu)
```

```
>>> # Example with `columns` as a dictionary of column => function pairs
>>> from functools import partial
>>> import scipy.stats as st
>>> # Passing keyword arguments to functions
>>> kstest_mod = partial(st.kstest, N=100)
>>> mannwhitney_mod = partial(st.mannwhitneyu, use_continuity=False)
>>> # Running tests
>>> results = na.test_hypothesis(
...     data,
...     col_na='some_column_with_NAs',
...     columns={
...         'age': kstest_mod,
...         'height': mannwhitney_mod,
...         'weight': mannwhitney_mod})
>>> pd.DataFrame(results, index=['statistic', 'p-value'])
```

1.7 scikit_na.altair

scikit_na.altair.**plot_corr**(*data: pandas.core.frame.DataFrame*, *columns: Optional[Sequence[str]] = None*, *mask_diag: bool = True*, *annot_color: str = 'black'*, *round_sgn: int = 2*, *font_size: int = 14*, *opacity: float = 0.5*, *corr_kws: Optional[dict] = None*, *chart_kws: Optional[dict] = None*, *x_kws: Optional[dict] = None*, *y_kws: Optional[dict] = None*, *color_kws: Optional[dict] = None*, *text_kws: Optional[dict] = None*) → altair.vegalite.v4.api.Chart

Correlation heatmap.

Parameters

- **data** (*DataFrame*) – Input data.
- **columns** (*Optional[Sequence[str]]*) – Columns names.
- **mask_diag** (*bool = True*) – Mask diagonal on heatmap.
- **corr_kws** (*dict, optional*) – Keyword arguments passed to `pandas.DataFrame.corr()` method.

- **heat_kws** (*dict, optional*) – Keyword arguments passed to `seaborn.heatmap()` method.

Returns Altair Chart object.

Return type altair.Chart

scikit_na.altair.**plot_hist**(*data: pandas.core.frame.DataFrame*, *col: str*, *col_na: str*, *na_label: Optional[str] = None*, *na_replace: Optional[dict] = None*, *heuristic: bool = True*, *thres_uniq: int = 20*, *step: bool = False*, *norm: bool = True*, *font_size: int = 14*, *xlabel: Optional[str] = None*, *ylabel: str = 'Frequency'*, *chart_kws: Optional[dict] = None*, *markarea_kws: Optional[dict] = None*, *markbar_kws: Optional[dict] = None*, *joinagg_kws: Optional[dict] = None*, *calc_kws: Optional[dict] = None*, *x_kws: Optional[dict] = None*, *y_kws: Optional[dict] = None*, *color_kws: Optional[dict] = None*) → altair.vegalite.v4.api.Chart

Histogram plot.

Plots a histogram of values in a column *col* grouped by NA/non-NA values in column *col_na*.

Parameters

- **data** (*DataFrame*) – Input data.
- **col** (*str*) – Column to display distribution of values.
- **col_na** (*str*) – Column to group values by.
- **na_label** (*str, optional*) – Legend title.
- **na_replace** (*dict, optional*) – Dictionary to replace values returned by `pandas.Series.isna()` method.
- **step** (*bool, optional*) – Draw step plot.
- **norm** (*bool, optional*) – Normalize values in groups.
- **xlabel** (*str, optional*) – X axis label.
- **ylabel** (*str, optional*) – Y axis label.
- **chart_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart()`.
- **markarea_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart.mark_area()`.
- **markbar_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart.mark_bar()`.
- **joinagg_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart.transform_joinaggregate()`.
- **calc_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart.transform_calculate()`.
- **x_kws** (*dict, optional*) – Keyword arguments passed to `altair.X()`.
- **y_kws** (*dict, optional*) – Keyword arguments passed to `altair.Y()`.
- **color_kws** (*dict, optional*) – Keyword arguments passed to `altair.Color()`.

Returns Altair Chart object.

Return type Chart

scikit_na.altair.**plot_kde**(*data: pandas.core.frame.DataFrame*, *col: str*, *col_na: str*, *na_label: Optional[str] = None*, *na_replace: Optional[dict] = None*, *font_size: int = 14*, *xlabel: Optional[str] = None*, *ylabel: str = 'Density'*, *chart_kws: Optional[dict] = None*, *markarea_kws: Optional[dict] = None*, *density_kws: Optional[dict] = None*, *x_kws: Optional[dict] = None*, *y_kws: Optional[dict] = None*, *color_kws: Optional[dict] = None*) → altair.vegalite.v4.api.Chart

Density plot.

Plots distribution of values in a column *col* grouped by NA/non-NA values in column *col_na*.

Parameters

- **data** (*DataFrame*) – Input data.
- **col** (*str*) – Column to display distribution of values.
- **col_na** (*str*) – Column to group values by.
- **na_label** (*str, optional*) – Legend title.
- **na_replace** (*dict, optional*) – Dictionary to replace values returned by `pandas.Series.isna()` method.
- **xlabel** (*str, optional*) – X axis label.
- **ylabel** (*str, optional*) – Y axis label.
- **chart_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart()`.
- **markarea_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart.mark_area()`.
- **density_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart.transform_density()`.
- **x_kws** (*dict, optional*) – Keyword arguments passed to `altair.X()`.
- **y_kws** (*dict, optional*) – Keyword arguments passed to `altair.Y()`.
- **color_kws** (*dict, optional*) – Keyword arguments passed to `altair.Color()`.

Returns Altair Chart object.

Return type Chart

scikit_na.altair.**plot_heatmap**(*data: pandas.core.frame.DataFrame*, *columns: Optional[Sequence[str]] = None*, *tooltip_cols: Optional[Sequence[str]] = None*, *names: Optional[list] = None*, *sort: bool = True*, *droppable: bool = True*, *font_size: int = 14*, *xlabel: str = 'Columns'*, *ylabel: str = 'Rows'*, *zlabel: str = 'Values'*, *chart_kws: Optional[dict] = None*, *rect_kws: Optional[dict] = None*, *x_kws: Optional[dict] = None*, *y_kws: Optional[dict] = None*, *color_kws: Optional[dict] = None*) → altair.vegalite.v4.api.Chart

Heatmap plot for NA/non-NA values.

By default, it also indicates values that are to be dropped by `pandas.DataFrame.dropna()` method.

Parameters

- **data** (*DataFrame*) – Input data.
- **columns** (*Optional[Sequence[str]], optional*) – Columns that are to be displayed on a plot.
- **tooltip_cols** (*Optional[Sequence[str]], optional*) – Columns to be used in tooltips.

- **names** (*list, optional*) – Values labels passed as a list. The first element corresponds to non-missing values, the second one to NA values, and the last one to droppable values, i.e. values to be dropped by `pandas.DataFrame.dropna()`.
- **sort** (*bool, optional*) – Sort values as NA/non-NA.
- **droppable** (*bool, optional*) – Show values to be dropped by `pandas.DataFrame.dropna()` method.
- **xlabel** (*str, optional*) – X axis label.
- **ylabel** (*str, optional*) – Y axis label.
- **zlabel** (*str, optional*) – Groups label (shown as a legend title).
- **chart_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart()` class.
- **rect_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart.mark_rect()` method.
- **x_kws** (*dict, optional*) – Keyword arguments passed to `altair.X()` class.
- **y_kws** (*dict, optional*) – Keyword arguments passed to `altair.Y()` class.
- **color_kws** (*dict, optional*) – Keyword arguments passed to `altair.Color()` class.

Returns Altair Chart object.

Return type altair.Chart

scikit_na.altair.**plot_scatter**(*data: pandas.core.frame.DataFrame*, *x_col: str*, *y_col: str*, *col_na: str*, *na_label: Optional[str] = None*, *na_replace: Optional[dict] = None*, *font_size: int = 14*, *xlabel: Optional[str] = None*, *ylabel: Optional[str] = None*, *circle_kws: Optional[dict] = None*, *color_kws: Optional[dict] = None*, *x_kws: Optional[dict] = None*, *y_kws: Optional[dict] = None*)

Scatter plot.

Parameters

- **data** (*DataFrame*) – Input data.
- **x_col** (*str*) – Column name corresponding to X axis.
- **y_col** (*str*) – Column name corresponding to Y axis.
- **col_na** (*str*) – Column name
- **na_label** (*str, optional*) – Label for NA values in legend.
- **na_replace** (*dict, optional*) – NA replacement mapping, by default {True: 'NA', False: 'Filled'}.
- **font_size** (*int, optional*) – Font size for plotting, by default 14.
- **xlabel** (*str, optional*) – X axis label.
- **ylabel** (*str, optional*) – Y axis label.
- **circle_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart.mark_circle()`.
- **color_kws** (*dict, optional*) – Keyword arguments passed to `altair.Color()`.
- **x_kws** (*dict, optional*) – Keyword arguments passed to `altair.X()`.
- **y_kws** (*dict, optional*) – Keyword arguments passed to `altair.Y()`.

Returns Scatter plot.

Return type altair.Chart

scikit_na.altair.**plot_stairs**(*data: pandas.core.frame.DataFrame*, *columns: Optional[Sequence[str]] = None*, *xlabel: str = 'Columns'*, *ylabel: str = 'Instances'*, *tooltip_label: str = 'Size difference'*, *dataset_label: str = '(Whole dataset)'*, *font_size: int = 14*, *area_kws: Optional[dict] = None*, *chart_kws: Optional[dict] = None*, *x_kws: Optional[dict] = None*, *y_kws: Optional[dict] = None*)

Stairs plot.

Plots changes in dataset size (rows/instances number) after applying `pandas.DataFrame.dropna()` to each column cumulatively.

Columns are sorted by maximum influence on dataset size.

Parameters

- **data** (*DataFrame*) – Input data.
- **columns** (*Optional[Sequence[str]], optional*) – Columns that are to be displayed on a plot.
- **xlabel** (*str, optional*) – X axis label.
- **ylabel** (*str, optional*) – Y axis label.
- **tooltip_label** (*str, optional*) – Label for differences in dataset size that is displayed on a tooltip.
- **dataset_label** (*str, optional*) – Label for the whole dataset (before dropping any NAs).
- **area_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart.mark_area()` method.
- **chart_kws** (*dict, optional*) – Keyword arguments passed to `altair.Chart()` class.
- **x_kws** (*dict, optional*) – Keyword arguments passed to `altair.X()` class.
- **y_kws** (*dict, optional*) – Keyword arguments passed to `altair.Y()` class.

Returns Chart object.

Return type altair.Chart

1.8 scikit_na.mpl

scikit_na.mpl.**plot_corr**(*data: pandas.core.frame.DataFrame*, *columns: Optional[Sequence[str]] = None*, *mask_diag: bool = True*, *corr_kws: Optional[dict] = None*, *heat_kws: Optional[dict] = None*) → matplotlib.axes._subplots.SubplotBase

Plot a correlation heatmap.

Parameters

- **data** (*DataFrame*) – Input data.
- **columns** (*Optional[Sequence[str]], optional*) – Columns names.
- **mask_diag** (*bool = True*) – Mask diagonal on heatmap.
- **corr_kws** (*dict, optional*) – Keyword arguments passed to `pandas.DataFrame.corr()`.

- **heat_kws** (*dict, optional*) – Keyword arguments passed to `pandas.DataFrame.heatmap()`.

Returns Heatmap AxesSubplot object.

Return type matplotlib.axes._subplots.AxesSubplot

scikit_na.mpl.**plot_heatmap**(*data: pandas.core.frame.DataFrame*, *columns: Optional[Sequence[str]] = None*, *droppable: bool = True*, *sort: bool = True*, *cmap: Optional[Sequence[str]] = None*, *names: Optional[Sequence[str]] = None*, *yaxis: bool = False*, *xaxis: bool = True*, *legend_kws: Optional[dict] = None*, *sb_kws: Optional[dict] = None*) → matplotlib.axes._subplots.SubplotBase

NA heatmap. Plots NA values as red lines and normal values as black lines.

Parameters

- **data** (*DataFrame*) – Input data.
- **columns** (*Optional[Sequence[str]], optional*) – Columns names.
- **droppable** (*bool, optional*) – Show values to be dropped by `pandas.DataFrame.dropna()` method.
- **sort** (*bool, optional*) – Sort DataFrame by selected columns.
- **cmap** (*Optional[Sequence[str]], optional*) – Heatmap and legend colormap: non-missing values, droppable values, NA values, correspondingly. Passed to `seaborn.heatmap()` method.
- **names** (*Optional[Sequence[str]], optional*) – Legend labels: non-missing values, droppable values, NA values, correspondingly.
- **yaxis** (*bool, optional*) – Show Y axis.
- **xaxis** (*bool, optional*) – Show X axis.
- **legend_kws** (*dict, optional*) – Keyword arguments passed to `matplotlib.axes._subplots.AxesSubplot()` method.
- **sb_kws** (*dict, optional*) – Keyword arguments passed to `seaborn.heatmap()` method.

Returns AxesSubplot object.

Return type matplotlib.axes._subplots.AxesSubplot

scikit_na.mpl.**plot_hist**(*data: pandas.core.frame.DataFrame*, *col: str*, *col_na: str*, *col_na_fmt: str = '"{}" is NA'*, *stat: str = 'density'*, *common_norm: bool = False*, *hist_kws: Optional[dict] = None*) → matplotlib.axes._subplots.SubplotBase

Histogram plot to compare distributions of values in column *col* split into two groups (NA/Non-NA) by column *col_na* in input DataFrame.

Parameters

- **data** (*DataFrame*) – Input DataFrame.
- **col** (*str*) – Name of column to compare distributions of values.
- **col_na** (*str*) – Name of column to group values by (NA/Non-NA).
- **col_na_fmt** (*str*) – Legend title format string.
- **common_norm** (*bool, optional*) – Use common norm.
- **hist_kws** (*dict, optional*) – Keyword arguments passed to `seaborn.histplot()`.

Returns AxesSubplot returned by `seaborn.histplot()`.

Return type SubplotBase

scikit_na.mpl.**plot_kde**(*data: pandas.core.frame.DataFrame*, *col: str*, *col_na: str*, *col_na_fmt: str = '"{}" is NA'*, *common_norm: bool = False*, *kde_kws: Optional[dict] = None*) → matplotlib.axes._subplots.SubplotBase

KDE plot to compare distributions of values in column *col* split into two groups (NA/Non-NA) by column *col_na* in input DataFrame.

Parameters

- **data** (*DataFrame*) – Input DataFrame.
- **col** (*str*) – Name of column to compare distributions of values.
- **col_na** (*str*) – Name of column to group values by (NA/Non-NA).
- **col_na_fmt** (*str*) – Legend title format string.
- **common_norm** (*bool, optional*) – Use common norm.
- **kde_kws** (*dict, optional*) – Keyword arguments passed to `seaborn.kdeplot()`.

Returns AxesSubplot returned by `seaborn.kdeplot()`.

Return type SubplotBase

scikit_na.mpl.**plot_stats**(*na_info: pandas.core.frame.DataFrame*, *idxstr: Optional[str] = None*, *idxint: Optional[int] = None*, ***kwargs*) → matplotlib.axes._subplots.SubplotBase

Plot barplot with NA descriptive statistics.

Parameters

- **na_info** (*DataFrame*) – Typically, the output of *scikit_na.describe()* method.
- **idxstr** (*str = None, optional*) – Index string labels passed to `pandas.DataFrame.loc()` method.
- **idxint** (*int = None, optional*) – Index integer labels passed to `pandas.DataFrame.iloc()` method.
- **kwargs** (*dict, optional*) – Keyword arguments passed to `seaborn.barplot()` method.

Returns Barplot AxesSubplot object.

Return type matplotlib.axes._subplots.AxesSubplot

Raises **ValueError** – Raised if neither `idxstr` nor `idxint` are passed.

INDEX