unipy Documentation

Release 0.1.27

Young Ju Kim

Apr 22, 2020

CONTENTS:

1 Installation 3

2 Usage 5

3 Welcome to unipy's documentation! 7

4 Indices and tables 9

5 Contents 11
- 5.1 Module contents . . . 11
- 5.2 Subpackages . . . 34

Python Module Index 91

Index 93

unipy is a toolkit for data scientists. This offers a number of scientific, statistical objects. This also contains many pythonic objects like generators, decorators and function wrappers, etc.

Some famous datasets embedded will make you easy to test.

CHAPTER

ONE

INSTALLATION

```
pip install unipy
```

CHAPTER

TWO

USAGE

```
import unipy as up
import unipy.dataset.api as dm
```

CHAPTER

THREE

WELCOME TO UNIPY'S DOCUMENTATION!

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

FIVE

CONTENTS

5.1 Module contents

5.1.1 unipy

Provides

1. Data Handling Tools
2. Statistical Functions.
3. Function Wrappers to profile
4. Generally-used Plots

How to use

In terms of Data science, Data Preprocessing & Plotting is one of the most annoying parts of Data Analysis. `unipy` offers you many functions maybe once you have tried to search in `google` or `stackoverflow`.

The docstring examples assume that *unipy* has been imported as *up*::

```
>>> import unipy as up
```

Code snippets are indicated by three greater-than signs::

```
>>> x = 42
>>> x = x + 1
```

Use the built-in `help` function to view a function's docstring::

```
>>> help(np.sort)
...
```

General-purpose documents like a glossary and help on the basic concepts of numpy are available under the `docs` sub-module:

```
>>> from unipy import docs
>>> help(docs)
...
```

Available subpackages

dataset Some famous datasets like `iris`, `titanic` and `adult`

image Image transformation tools.

math Mathmatical core functions for `unipy` itself

plots Most used plots

stats Statistic tools

tools Data handling tools

utils High-level wrappers & Python function decorators

unipy_test Test-codes of `unipy`

class unipy.**Ellipse**(*diameter*)

Bases: `object`

Create an ellipse.

diameter

radius

center

angle

coordinates()

unipy.**point_boxplot**(*data*, *groupby=None*, *value=None*, *rot=90*, *spread=0.2*, *dot_size=15.0*, *dot_color='b'*, *dot_alpha=0.2*, *figsize=12, 9*, **args*, ***kwargs*)

Boxplot with points.

Draw boxplots by given keys(groupby, value).

Parameters

- **data** (*pandas.DataFrame*) – a dataset.
- **groupby** (*str or list-like (default: None)*) – A key column to separate. (X-axis, categorical) When `str`, it should be a column name to groupby. When `list-like`, it contains a column name to groupby.
- **value** (*str or list-like (default: None)*) – A key column to get values. (Y-axis, numerical) When `str`, it should be a column name of values. When `list-like`, it contains a column name of values.
- **rot** (*int (default: 90)*) – A rotation angle to show X-axis labels.
- **spread** (*float (default: .2)*) – A spread ratio of points. The bigger, the pointing distribution width are broader.
- **dot_size** (*float (default: 15.)*) – A size of each points.
- **dot_color** (*int (default: 'b')*) – A color name of each points.
- **dot_alpha** (*float (default: .2)*) – A transparency value of each points.

Returns

- *matplotlib.figure.Figure* – A plot figure.
- *Exceptions*
- ———-

- *AssertionError* – It is raised when two or more names are given to `groupby` or `value`.

See also:

`pandas.DataFrame.boxplot` `matplotlib.pyplot`

Examples

```
>>> import unipy.dataset.api as dm
>>> from unipy.plots import point_boxplot
>>> dm.init()
>>> data = dm.load('iris')
Dataset : iris
>>> tmp = point_boxplot(data, groupby='species', value='sepal_length')
```

unipy.**point_boxplot_axis**(*data*, *groupby=None*, *value=None*, *rot=90*, *spread=0.2*, *dot_size=15.0*, *dot_color='b'*, *dot_alpha=0.2*, *share_yrange=True*, *figsize=12, 9*, **args*, ***kwargs*)

Boxplot with points, horizontally seperated.

Draw boxplots by given keys(groupby, value).

Parameters

- **data** (*pandas.DataFrame*) – a dataset.
- **groupby** (*str or list-like (default: None)*) – A key column to separate. (X-axis, categorical) When `str`, it should be a column name to groupby. When `list-like`, it contains a column name to groupby.
- **value** (*str or list-like (default: None)*) – A key column to get values. (Y-axis, numerical) When `str`, it should be a column name of values. When `list-like`, it contains a column name of values.
- **rot** (*int (default: 90)*) – A rotation angle to show X-axis labels.
- **spread** (*float (default: .2)*) – A spread ratio of points. The bigger, the pointing distribution width are broader.
- **dot_size** (*float (default: 15.)*) – A size of each points.
- **dot_color** (*int (default: 'b')*) – A color name of each points.
- **dot_alpha** (*float (default: .2)*) – A transparency value of each points.
- **share_yrange** (*Boolean (defalut: True)*) – False then each Y-axis limit of boxplots will draw independent.

Returns

- *matplotlib.figure.Figure* – A plot figure.
- *Exceptions*
- ———-
- *AssertionError* – It is raised when two or more names are given to `groupby` or `value`.

See also:

`pandas.DataFrame.boxplot` `matplotlib.pyplot`

Examples

```
>>> import unipy.dataset.api as dm
>>> from unipy.plots import point_boxplot_axis
>>> dm.init()
>>> data = dm.load('iris')
Dataset : iris
>>> tmp = point_boxplot_axis(data,
...                          groupby='species',
...                          value='sepal_length',
...                          share_yrange=True)
```

unipy.**mosaic_plot**(*data*, *groupby=None*, *col_list=None*, *show_values=True*, *rot=90*, *width=0.9*, *figsize=12, 9*, **args*, ***kwargs*)

Mosaic Plot via Stacked bar plots.

Draw plots by given keys(groupby, value).

Parameters

- **data** (*pandas.DataFrame*) – a dataset.
- **groupby** (*str or list-like (default: None)*) – A key column to separate. (X-axis, categorical) When `str`, it should be a column name to groupby. When `list-like`, it contains a column name to groupby.
- **col_list** (*str or list-like (default: None)*) – A key column to get values. (Y-axis, numerical) When `str`, it should be column names of values. When `list-like`, it contains column names of values.
- **rot** (*int (default: 90)*) – A rotation angle to show X-axis labels.
- **show_values** (*boolean (default: True)*) – Choose If *n* is annotated.

Returns

- *matplotlib.figure.Figure* – A plot figure.
- *Exceptions*
- ———-
- *AssertionError* – It is raised when two or more names are given to `groupby` or `value`.

See also:

`pandas.DataFrame.barplot` `matplotlib.pyplot`

Examples

```
>>> import unipy.dataset.api as dm
>>> from unipy.plots import mosaic_plot
>>> dm.init()
>>> data = dm.load('adult')
Dataset : iris
>>> tmp = mosaic_plot(data, groupby='native_country',
... col_list=['workclass', 'education'])
```

unipy.**rgb2gras**(*img_array*)

unipy.**hough_transform**(*img_bin*, *theta_res=1*, *rho_res=1*)

unipy.**deviation**(*container*, *method='mean'*, *if_abs=True*)
Deviation.

unipy.**vif**(*y*, *X*)
Variance inflation factor.

unipy.**mean_absolute_percentage_error**(*measure*, *predict*, *thresh=3.0*)
Mean Absolute Percentage Error. It is a percent of errors. It measures the prediction accuracy of a forecasting method in Statistics with the real mesured values and the predicted values, for example in trend estimation. If MAPE is 5, it means this prediction method potentially has 5% error. It cannot be used if there are zero values, because there would be a division by zero.

unipy.**average_absolute_deviation**(*measure*, *predict*, *thresh=2*)
Average Absolute Deviation. It is ... It measures the prediction accuracy of a forecasting method in Statistics with the real mesured values and the predicted values, for example in trend estimation. If MAD is 5, it means this prediction method potentially has...

unipy.**median_absolute_deviation**(*measure*, *predict*, *thresh=2*)
Median Absolute Deviation. It is ... It measures the prediction accuracy of a forecasting method in Statistics with the real mesured values and the predicted values, for example in trend estimation. If MAD is 5, it means this prediction method potentially has...

unipy.**calculate_interaction**(*rankTbl*, *pvTbl*, *target*, *ranknum=10*)
Feature interaction calculation.

unipy.**f_test**(*a*, *b*, *scale=1*, *alternative='two-sided'*, *conf_level=0.95*, **args*, ***kwargs*)
F-Test.

unipy.**f_test_formula**(*a*, *b*, *scale=1*, *alternative='two-sided'*, *conf_level=0.95*, **args*, ***kwargs*)
F-Test by formula.

unipy.**anova_test**(*formula*, *data=None*, *typ=1*)
ANOVA Test.

unipy.**anova_test_formula**(*formula*, *data=None*, *typ=1*)
ANOVA Test by formula.

unipy.**chisq_test**(*data*, *x=None*, *y=None*, *correction=None*, *lambda_=None*, *margin=True*, *print_ok=True*)
Chi-square Test.

lambda_ gives the power in the Cressie-Read power divergence statistic. The default is 1. For convenience, **lambda_** may be assigned one of the following strings, in which case the corresponding numerical value is used:

Parameters

- **data** (*pandas.DataFrame*) –
- **x** (*str (default: None)*) –
- **y** (*str (default: None)*) –
- **correction** (*(default: None)*) –
- **lambda_** (*lambda (default: None)*) –
- **margin** (*Boolean (default: True)*) –
- **print_ok** (*Boolean (default: True)*) –

unipy.**fisher_test**(*data*, *x=None*, *y=None*, *alternative='two-sided'*, *margin=True*, *print_ok=True*)
Fisher's Exact Test.

unipy.**lasso_rank**(*formula=None*, *X=None*, *y=None*, *data=None*, *alpha=array([1e-05, 0.00011, 0.00021, 0.00031, 0.00041, 0.00051, 0.00061, 0.00071, 0.00081, 0.00091, 0.00101, 0.00111, 0.00121, 0.00131, 0.00141, 0.00151, 0.00161, 0.00171, 0.00181, 0.00191, 0.00201, 0.00211, 0.00221, 0.00231, 0.00241, 0.00251, 0.00261, 0.00271, 0.00281, 0.00291, 0.00301, 0.00311, 0.00321, 0.00331, 0.00341, 0.00351, 0.00361, 0.00371, 0.00381, 0.00391, 0.00401, 0.00411, 0.00421, 0.00431, 0.00441, 0.00451, 0.00461, 0.00471, 0.00481, 0.00491, 0.00501, 0.00511, 0.00521, 0.00531, 0.00541, 0.00551, 0.00561, 0.00571, 0.00581, 0.00591, 0.00601, 0.00611, 0.00621, 0.00631, 0.00641, 0.00651, 0.00661, 0.00671, 0.00681, 0.00691, 0.00701, 0.00711, 0.00721, 0.00731, 0.00741, 0.00751, 0.00761, 0.00771, 0.00781, 0.00791, 0.00801, 0.00811, 0.00821, 0.00831, 0.00841, 0.00851, 0.00861, 0.00871, 0.00881, 0.00891, 0.00901, 0.00911, 0.00921, 0.00931, 0.00941, 0.00951, 0.00961, 0.00971, 0.00981, 0.00991])*, *k=2*, *plot=False*, **args*, ***kwargs*)

Feature selection by LASSO regression.

Parameters

- **formula** – R-style formula string
- **X** (*list-like*) – Column values for X.
- **y** (*list-like*) – A column value for y.
- **data** (*pandas.DataFrame*) – A DataFrame.
- **alpha** (*Iterable*) – An Iterable contains alpha values.
- **k** (*int*) – Threshold of coefficient matrix
- **plot** (*Boolean (default: False)*) – True if want to plot the result.

Returns

- **rankTbl** (*pandas.DataFrame*) – Feature ranking by given `k`.
- **minIntercept** (*pandas.DataFrame*) – The minimum intercept row in coefficient matrix.
- **coefMatrix** (*pandas.DataFrame*) – A coefficient matrix.
- **kBest** (*pandas.DataFrame*) – When Given `k`, The best intercept row in coefficient matrix.
- **kBestPredY** (*dict*) – A predicted `Y` with `kBest` alpha.

Example

```
>>> import unipy.dataset.api as dm
>>> dm.init()
['cars', 'anscombe', 'iris', 'nutrients', 'german_credit_scoring_fars2008',
↪'winequality_red', 'winequality_white', 'titanic', 'car90', 'diabetes', 'adult',
↪ 'tips', 'births_big', 'breast_cancer', 'air_quality', 'births_small']
>>> wine_red = dm.load('winequality_red')
Dataset : winequality_red
>>>
>>> ranked, best_by_intercept, coefTbl, kBest, kBestPred = lasso_rank(X=wine_red.
↪columns.drop('quality'), y=['quality'], data=wine_red)
>>> ranked
                  rank  lasso_coef  abs_coef
volatile_acidity     1   -0.675725  0.675725
alcohol              2    0.194865  0.194865
>>> best_by_intercept
                      RSS  Intercept  fixed_acidity  volatile_acidity      alpha_
↪0.00121  691.956364   3.134874       0.002374         -1.023793    (continues on next page)
```

(continued from previous page)

citric_acid residual_sugar chlorides free_sulfur_dioxide alpha_0.00121 0.0 0.0 -0.272912 -0.0

total_sulfur_dioxide density pH sulphates alcohol alpha_0.00121 -0.000963 -0.0 -0.0 0.505956 0.264552

var_count

alpha_0.00121 6 >>>

unipy.**feature_selection_vif**(*data*, *thresh=5.0*)

Stepwise Feature Selection for multivariate analysis.

It calculates OLS regressions and the variance inflation factors iterating all explanatory variables. If the maximum VIF of a variable is over the given threshold, It will be dropped. This process is repeated until all VIFs are lower than the given threshold.

Recommended threshold is lower than 5, because if VIF is greater than 5, then the explanatory variable selected is highly collinear with the other explanatory variables, and the parameter estimates will have large standard errors because of this.

Parameters

- **data** (*DataFrame, (rows: observed values, columns: multivariate variables)*) – design dataframe with all explanatory variables, as for example used in regression
- **thresh** (*int, float*) – A threshold of VIF

Returns

- **Filtered_data** (*DataFrame*) – A subset of the input DataFame
- **dropped_List** (*DataFrame*) – 'var' column : dropped variable names from input data columns 'vif' column : variance inflation factor of dropped variables

Notes

This function does not save the auxiliary regression.

See also:

`statsmodels.stats.outliers_influence.variance_inflation_factor()`

References

http://en.wikipedia.org/wiki/Variance_inflation_factor

unipy.**from_formula**(*formula*)

R-style Formula Formatting.

unipy.**exc**(*source*, *blacklist*)

Get items except the given list.

This function splits an Iterable into the given size of multiple chunks. The items of An iterable should be the same type.

Parameters

- **source** (*Iterable*) – An Iterable to filter.
- **blacklist** (*Iterable*) – A list contains items to eliminate.

Returns A filtered list.

Return type list

See also:

`Infix Operator`

Examples

```
>>> import unipy as up
>>> up.splitter(list(range(10)), how='equal', size=3)
[(0, 1, 2, 3), (4, 5, 6), (7, 8, 9)]
```

```
>>> up.splitter(list(range(10)), how='remaining', size=3)
[(0, 1, 2), (3, 4, 5), (6, 7, 8), (9,)]
```

unipy.**splitter**(*iterable*, *how='equal'*, *size=2*)

Split data with given size.

This function splits an Iterable into the given size of multiple chunks. The items of An iterable should be the same type.

Parameters

- **iterable** (*Iterable*) – An Iterable to split.
- **how** (*{'equal', 'remaining'}*) – The method to split. 'equal' is to split chunks with the approximate length within the given size. 'remaining' is to split chunks with the given size, and the remains are bound as the last chunk.
- **size** (*int*) – The number of chunks.

Returns A list of chunks.

Return type list

See also:

`numpy.array_split()`, `itertools.islice()`

Examples

```
>>> import unipy as up
>>> up.splitter(list(range(10)), how='equal', size=3)
[(0, 1, 2, 3), (4, 5, 6), (7, 8, 9)]
```

```
>>> up.splitter(list(range(10)), how='remaining', size=3)
[(0, 1, 2), (3, 4, 5), (6, 7, 8), (9,)]
```

unipy.**even_chunk**(*iterable*, *chunk_size*, **args*, ***kwargs*)

Split data into even size.

This function splits an Iterable into the given size of multiple chunks. The items of An iterable should be the same type.

Parameters

- **iterable** (*Iterable*) – An Iterable to split. If N-dimensional, It is chunked by 1st dimension.
- **chunk_size** (int) – The length of each chunks.

Returns A generator yields a list of chunks. The data type of the elements in a list are equal to the source data type.

Return type generator

See also:

`itertools.islice` `yield from`

Examples

```
>>> import numpy as np
>>> from unipy.tools.data_handler import even_chunk
>>> data = list(range(7))  # list, 1D
>>> print(data)
[0, 1, 2, 3, 4, 5, 6]
>>> chunked_gen = even_chunk(data, 3)
>>> print(chunked_gen)
<generator object even_chunk at 0x7fc4924897d8>
>>> next(chunked_gen)
[0, 1, 2]
>>> chunked = list(even_chunk(data, 3))
>>> print(chunked)
[[0, 1, 2], [3, 4, 5], [6]]
>>> data = np.arange(30).reshape(-1, 3)  # np.ndarray, 2D
>>> print(data)
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]])
>>> chunked_gen = even_chunk(data, 4)
>>> next(chunked_gen)
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8]), array([ 9, 10, 11])]
>>> next(chunked_gen)
[array([12, 13, 14]),
 array([15, 16, 17]),
 array([18, 19, 20]),
 array([21, 22, 23])]
>>> next(chunked_gen)
[array([24, 25, 26]), array([27, 28, 29])]
>>> next(chunked_gen)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
StopIteration
```

unipy.**pair_unique**(**args*)

Get Unique pairsets.

This function gets an unique pair-sets of given data.

Parameters **iterable** (*Iterable*) – Iterables having an equal length.

Returns A list of tuples. Each tuple is an unique pair of values.

Return type list

Raises **ValueError** – If the lengths of argments are not equal.

See also:

zip set

Examples

```
>>> from unipy.tools.data_handler import pair_unique
>>> data = dm.load('titanic')
Dataset : titanic
>>> data.head()
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
>>> pair_unique(data.iloc[:, 0], data.iloc[:, 1])
[(5, '1st'), (19, '3rd'), (29, '1st'), (20, 'Crew'),
 (21, '1st'), (3, '3rd'), (16, 'Crew'), (26, '2nd'),
 (23, '3rd'), (10, '2nd'), (24, 'Crew'), (7, '3rd'),
 (4, 'Crew'), (27, '3rd'), (18, '2nd'), (28, 'Crew'),
 (30, '2nd'), (11, '3rd'), (2, '2nd'), (1, '1st'),
 (14, '2nd'), (31, '3rd'), (22, '2nd'), (17, '1st'),
 (8, 'Crew'), (9, '1st'), (32, 'Crew'), (15, '3rd'),
 (6, '2nd'), (12, 'Crew'), (13, '1st'), (25, '1st')]
>>> idx1 = [1, 2, 3]
>>> idx2 = [0, 9, 8, 4]
>>> pair_unique(idx1, idx2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: All argments should have the same length.
```

unipy.**df_pair_unique**(*data_frame*, *col_list*, *to_frame=False*)

Get unique pairsets in pandas.DataFrame.

This function gets an unique pair-sets of given columns.

Parameters

- **data_frame** (*pandas.DataFrame*) – DataFrame to get unique-pairs.
- **col_list** (*pandas.Index, list, tuple*) – Column names of given DataFrame.
- **to_frame** (*Boolean (default: False)*) – Choose output type. If True, It returns pandas.DataFrame as an output. If False, It returns a list of tuples.

Returns

- *list* – If to_frame=False, A list of tuples is returned. Each tuple is an unique pair of values.

- *pandas.DataFrame* – If to_frame=True, pandas.DataFrame is returned. Each row is an unique pair of values.

See also:

`pandas.DataFrame.itertuples()`

Examples

```
>>> from unipy.tools.data_handler import df_pair_unique
>>> data = dm.load('titanic')
Dataset : titanic
>>> data.head()
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
>>> df_pair_unique(data, ['Class', 'Sex'])
[('3rd', 'Male'), ('2nd', 'Male'), ('2nd', 'Female'), ('1st', 'Female'),
 ('Crew', 'Male'), ('1st', 'Male'), ('Crew', 'Female'), ('3rd', 'Female')]
>>> df_pair_unique(data, ['Class', 'Sex'], to_frame=True)
  Class     Sex
0   3rd    Male
1   2nd    Male
2   2nd  Female
3   1st  Female
4  Crew    Male
5   1st    Male
6  Crew  Female
7   3rd  Female
```

`unipy.`**`map_to_tuple`**(*iterable*)
Only for some specific reason.

`unipy.`**`map_to_list`**(*iterable*)
Only for some specific reason.

`unipy.`**`merge_csv`**(*file_path*, *pattern='*.csv'*, *sep=','*, *if_save=True*, *save_name=None*, *low_memory=True*)
Merge seperated csv type datasets into one dataset. Summary

This function get separated data files together. When merged, the file is sorted by its name in ascending order.

Parameters

- **`file_path`** (*`str`*) – A directory path of source files.
- **`pattern`** (*`str`*) – A File extension with conditional naming. (default: '*.csv')
- **`sep`** (*`int`*) – A symbol seperating data columns.
- **`if_save`** (*`Boolean (Optional, default: True)`*) – False if you don't want to save the result.
- **`save_name`** (*`str`*) – A filename to save the result. It should be given if if_save=True. If inappropriate name is given, the first name of file list is used.
- **`low_memory`** (*`Boolean (Optional, default: True)`*) – It is used for pandas.read_csv() option only.

Returns A concatenated DataFrame.

Return type pandas.DataFrame

Examples

```
>>> from unipy.tools.data_handler import merge_csv
>>> data = dm.load('titanic')
Dataset : titanic
>>> data.head(9)
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
5   2nd  Female  Child       No     0
6   3rd  Female  Child       No    17
7  Crew  Female  Child       No     0
8   1st    Male  Adult       No   118
>>> data.iloc[:2, :].to_csv('tmp1.csv', header=True, index=False)
>>> data.iloc[2:4, :].to_csv('tmp2.csv', header=True, index=False)
>>> data.iloc[4:9, :].to_csv('tmp3.csv', header=True, index=False)
>>> merged = merge_csv('./')
>>> merged
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
5   2nd  Female  Child       No     0
6   3rd  Female  Child       No    17
7  Crew  Female  Child       No     0
8   1st    Male  Adult       No   118
```

unipy.**nancumsum**(*iterable*)

A cumulative sum function.

A cumulative sum function.

Parameters **iterable** (*Iterable*) – Iterables to calculate cumulative sum.

Yields *int* – A cumulative summed value.

See also:

numpy.isnan()

Examples

```
>>> from unipy.tools.data_handler import nancumsum
>>> tmp = [1, 2, 4]
>>> nancumsum(tmp)
<generator object nancumsum at 0x1084553b8>
>>> list(nancumsum(tmp))
[1, 3, 7]
```

unipy.**depth**(*iterable*)

Get dimension depth.

Get a dimension depth number of a nested iterable.

Parameters **iterable** (*iterable*) – An Iterable to get a dimension depth number.

Returns A dimension depth number.

Return type int

See also:

`collections.Iterable()`

Examples

```
>>> from unipy.tools.data_handler import depth
>>> tmp = [(1, 3), (4, 6), (7, 9), (10, 12)]
>>> depth(tmp)
2
>>> tmp3d = [[np.arange(i) + i for i in range(2, j)]
...          for j in range(5, 10)]
>>> depth(tmp3d)
3
>>> # It can handle dict type (considering values only).
>>> tmp3d_dict = [{'key' + str(i): np.arange(i) + i for i in range(2, j)}
...               for j in range(5, 10)]
>>> depth(tmp3d_dict)
3
```

unipy.**zero_padder_2d**(*arr*, *max_len=None*, *method='backward'*)

Zero-padding for fixed-length inputs(2D).

Zero-padding Function with nested sequence. Each elements of a given sequence is padded fixed-length.

Parameters

- **arr** (*Iterable*) – A nested sequence containing 1-Dimensional numpy.ndarray.
- **max_len** (*int (default: None)*) – A required fixed-length of each sequences. If None, It calculates the max length of elements as max_len.
- **method** (*{'forward', 'backward'} (default: 'backward')*) – where to pad.

Returns A list containing 3-Dimensional numpy.ndarray with fixed-length 2D.

Return type list

See also:

`unipy.depth(), numpy.pad(), numpy.stack()`

Examples

```
>>> from unipy.tools.data_handler import zero_padder_2d
>>> tmp = [np.arange(i) + i for i in range(2, 5)]
>>> tmp
[array([2, 3]), array([3, 4, 5]), array([4, 5, 6, 7])]
>>> zero_padder_2d(tmp)
array([[2, 3, 0, 0],
       [3, 4, 5, 0],
       [4, 5, 6, 7]])
>>> zero_padder_2d(tmp, max_len=6)
array([[2, 3, 0, 0, 0, 0],
       [3, 4, 5, 0, 0, 0],
       [4, 5, 6, 7, 0, 0]])
 >>> zero_padder_2d(tmp, max_len=5, method='forward')
array([[0, 0, 0, 2, 3],
       [0, 0, 3, 4, 5],
       [0, 4, 5, 6, 7]])
```

unipy.**zero_padder_3d**(*arr*, *max_len=None*, *method='backward'*)

Zero-padding for fixed-length inputs(3D).

Zero-padding Function with nested sequence. Each elements of a given sequence is padded fixed-length.

Parameters

- **arr** (*Iterable*) – A nested sequence containing 2-Dimensional numpy.ndarray.
- **max_len** (*int (default: None)*) – A required fixed-length of each sequences. If None, It calculates the max length of elements as max_len.
- **method** (*{'forward', 'backward'} (default: 'backward')*) – where to pad.

Returns A list containing 3-Dimensional numpy.ndarray with fixed-length 2D.

Return type list

Raises **ValueError** – All 3D shape of inner numpy.ndarray is not equal.

See also:

unipy.depth(), numpy.pad(), numpy.stack()

Examples

```
>>> from unipy.tools.data_handler import zero_padder_3d
>>> tmp3d = [np.arange(i * 2).reshape(-1, 2) for i in range(1, 5)]
>>> tmp3d
[array([[0, 1]]),
 array([[0, 1],
        [2, 3]]),
 array([[0, 1],
        [2, 3],
        [4, 5]]),
 array([[0, 1],
        [2, 3],
        [4, 5],
        [6, 7]])]
```

(continues on next page)

(continued from previous page)

```
>>> zero_padder_3d(tmp3d)
array([[[0, 1],
        [0, 0],
        [0, 0],
        [0, 0]],
```

[[0, 1], [2, 3], [0, 0], [0, 0]],

[[0, 1], [2, 3], [4, 5], [0, 0]],

[[0, 1], [2, 3], [4, 5], [6, 7]]])

```
>>> tmp3d_eye
[array([[1.]]),
 array([[1., 0.],
        [0., 1.]]),
 array([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]]),
 array([[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]])]
>>> zero_padder_3d(tmp3d_eye)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 24, in zero_padder_3d
ValueError: 3D shape should be equal.
```

unipy.**multiprocessor**(*func*, *worker=2*, *arg_zip=None*, **args*, ***kwargs*)

Use multiprocessing as a function.

Just for convenience.

Parameters

- **func** (*Function*) – Any function without `lambda`.
- **worker** (*int (default: 2)*) – A number of processes.
- **arg_zip** (*zip (default: None)*) – A `zip` instance.

Returns A list contains results of each processes.

Return type list

See also:

`multiprocessing.pool`

Examples

```
>>> from unipy.utils.wrapper import multiprocessor
>>> alist = [1, 2, 3]
>>> blist = [-1, -2, -3]
>>> def afunc(x, y):
...     return x + y
...
>>> multiprocessor(afunc, arg_zip=zip(alist, blist))
[0, 0, 0]
>>> def bfunc(x):
...     return x + 2
...
>>> multiprocessor(bfunc, arg_zip=zip(alist))
[3, 4, 5]
```

unipy.**uprint**(**args*, *print_ok=True*, ***kwargs*)

Print option interface.

This function is equal to `print` function but added `print_ok` option. This allows you to control printing in a function.

Parameters

- ***args** (whatever `print` allows.) – It is same as `print` does.
- **print_ok** (*Boolean (default: True)*) – An option whether you want to print something out or not.
- **arg_zip** (*zip (default: None)*) – A `zip` instance.

unipy.**lprint**(*input_x*, *output*, *name=None*)

Print option interface.

This function is to stdout the shape of input layer & output layer in Deep Learning architecture.

Parameters

- **input_x** (*numpy.ndarray*) – A `numpy.ndarray` object of input source.
- **output** (*numpy.ndarray*) – A `numpy.ndarray` object of output target.
- **name** (*str (default: None)*) – An optional name you want to print out.

unipy.**aprint**(**arr*, *maxlen=None*, *name_list=None*, *decimals=None*)

Stdout the *numpy.ndarray* in pretty.

It prints the multiple *numpy.ndarray* out "Side by Side."

Parameters

- **arr** (*numpy.ndarray*) – Any arrays you want to print out.
- **maxlen** (*int (default: None)*) – A length for each array to print out. It is automatically calculated in case of *None*.
- **name_list** (*list (default: None)*) – A list contains the names of each arrays. Upper Alphabet is given in case of *None*.
- **decimals** (*int (default: None)*) – A number to a specified number of digits to truncate.

Examples

```
>>> from unipy.utils.wrapper import aprint
>>> arr_x = np.array([
... [.6, .5, .1],
... [.4, .2, .8],
... ])
>>> arr_y = np.array([
... [.4, .6],
... [.7, .3,],
... ])
>>> aprint(arr_x, arr_y)
========================================
|  A                 |    B             |
|  (2, 3)            |    (2, 2)        |
========================================
|  [[0.6 0.5 0.1]    |    [[0.4 0.6]    |
|   [0.4 0.2 0.8]]   |     [0.7 0.3]]   |
========================================
>>> aprint(arr_x, arr_y, name_list=['X', 'Y'])
========================================
|  X                 |    Y             |
|  (2, 3)            |    (2, 2)        |
========================================
|  [[0.6 0.5 0.1]    |    [[0.4 0.6]    |
|   [0.4 0.2 0.8]]   |     [0.7 0.3]]   |
========================================
>>> aprint(arr_x, arr_y, arr_y[:1], name_list=['X', 'Y', 'Y_1'])
============================================================
|  X                 |    Y             |    Y_1           |
|  (2, 3)            |    (2, 2)        |    (1, 2)        |
============================================================
|  [[0.6 0.5 0.1]    |    [[0.4 0.6]    |    [[0.4 0.6]]   |
|   [0.4 0.2 0.8]]   |     [0.7 0.3]]   |                  |
============================================================
```

unipy.**time_profiler**(*func*)

Print wrapper for time profiling.

This wrapper prints out start, end and elapsed time.

Parameters **func** (*Function*) – A function to profile.

Returns A wrapped function.

Return type Function

See also:

functools.wraps decorator

Examples

```
>>> import unipy as up
>>> @up.time_profiler
... def afunc(i):
...     return len(list(range(i)))
...
>>> res = afunc(58)
(afunc) Start   : 2018-06-20 22:11:35.511374
(afunc) End     : 2018-06-20 22:11:35.511424
(afunc) Elapsed :             0:00:00.000050
>>> res
58
```

unipy.**time_logger**(*func*)

Logging wrapper for time profiling.

This wrapper logs start, end and elapsed time.

Parameters **func** (*Function*) – A function to profile.

Returns A wrapped function.

Return type Function

See also:

functools.wraps decorator

Examples

```
>>> import unipy as up
>>> @up.time_logger
... def afunc(i):
...     return len(list(range(i)))
...
>>> res = afunc(58)
(afunc) Start   : 2018-06-20 22:11:35.511374
(afunc) End     : 2018-06-20 22:11:35.511424
(afunc) Elapsed :             0:00:00.000050
>>> res
58
```

class unipy.**profiler**(*type='logging'*)

Bases: object

unipy.**job_wrapper**(*func*)

Print wrapper for time profiling.

This wrapper prints out start & end line.

Parameters **func** (*Function*) – A function to separate print-line job.

Returns A wrapped function.

Return type Function

See also:

functools.wraps decorator

Examples

```
>>> import unipy as up
>>> @up.job_wrapper
... def afunc(i):
...     return len(list(range(i)))
...
>>> afunc(458)
----------- [afunc] START -----------
```

———— [afunc] END ————

afunc : 0:00:00.000023

458

class unipy.**Infix**(*func*)

Bases: object

Wrapper for define an operator.

This wrapper translates a function to an operator.

Returns A wrapped function.

Return type Function

See also:

`functools.partial` decorator

Examples

```
>>> @Infix
... def add(x, y):
...     return x + y
...
>>> 5 |add| 6
11
>>> instanceof = Infix(isinstance)
>>> 5 |instanceof| int
True
```

unipy.**infix**(*func*)

A functional API for Infix decorator.

Returns A wrapped function.

Return type Function

See also:

`unipy.utils.wrapper.infix`

Examples

```
>>> @infix
... def add(x, y):
...     return x + y
...
>>> 5 |add| 6
11
>>> instanceof = infix(isinstance)
>>> 5 |instanceof| int
True
```

class unipy.**ReusableGenerator**(*generator*)

Bases: object

Temporary Interface to re-use generator for convenience.

Once assigned, It can be infinitely consumed **as long as an input generator remains un-exhausted.

_source

A source generator.

Type generator

See also:

generator itertools.tee

Examples

```
>>> from unipy.utils.generator import ReusableGenerator
>>> gen = (i for i in range(10))
>>> gen
<generator object <genexpr> at 0x11120ebf8>
>>> regen = ReusableGenerator(gen)
>>> regen
<unipy.utils.generator.ReusableGenerator object at 0x1061a97f0>
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(gen)  # If the source is used, copied one will be exhausted too.
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(gen)
[]
>>> list(regen)
[]
```

unipy.**re_generator**(*generator*)

A functional API for unipy.ReusableGenerator.

Once assigned, It can be infinitely consumed **as long as an output generator is called at least one time.

Parameters **generator** (*generator*) – An generator to copy. This original generator should not be used anywhere else, until the copied one consumed at least once.

Returns A generator to be used infinitely.

Return type generator

See also:

generator `itertools.tee`

Examples

```
>>> from unipy.utils.generator import re_generator
>>> gen = (i for i in range(10))
>>> gen
<generator object <genexpr> at 0x11120ebf8>
>>> regen = copy_generator(gen)
>>> regen
<unipy.utils.generator.ReusableGenerator object at 0x1061a97f0>
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(gen)  # Once the copied one is used, the source will be exhausted.
[]
>>> list(gen)
[]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

unipy.**split_generator**(*iterable*, *size*)

unipy.**num_fromto_generator**(*start*, *end*, *term*)

A range function yields pair chunks.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters ***args** (*int*) – end or start, end[, term] It works like range function.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

`yield`

Examples

```
>>> from unipy.utils.generator import num_fromto_generator
>>>
>>> query = 'BETWEEN {pre} AND {nxt};'
>>>
>>> q_list = [query.format(pre=item[0], nxt=item[1])
...           for item in num_fromto_generator(1, 100, 10)]
>>> print(q_list[0])
BETWEEN 1 AND 10;
>>> print(q_list[1])
BETWEEN 11 AND 20;
```

unipy.**dt_fromto_generator**(*start*, *end*, *day_term*, *tm_format='%Y%m%d'*)

A range function yields datetime formats by pair.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters

- **start** (*str*) – start datetime like 'yyyymmdd'.
- **end** (*str*) – start datetime like 'yyyymmdd'.
- **day_term** (*int*) – term of days.
- **tm_format** (*(default: '%Y%m%d')*) – datetime format string.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

`yield`

Examples

```
>>> from unipy.utils.generator import dt_fromto_generator
>>> dt_list = [item for item in
...            dt_fromto_generator('20170101','20170331', 10)]
>>> dt_list[:3]
[('20170101', '20170110'),
 ('20170111', '20170120'),
 ('20170121', '20170130')]
```

unipy.**tm_fromto_generator**(*start*, *end*, *day_term*, *tm_string=['000000', '235959']*, *tm_format='%Y%m%d'*)

A range function yields datetime formats by pair.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters

- **start** (*str*) – start datetime like 'yyyymmdd'.
- **end** (*str*) – start datetime like 'yyyymmdd'.
- **day_term** (*int*) – term of days.
- **tm_string** (list (default: `['000000', '235959']`)) – time strings to concatenate.
- **tm_format** (*(default: '%Y%m%d')*) – datetime format string.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

`yield`

Examples

```
>>> from unipy.utils.generator import tm_fromto_generator
>>> tm_list = [item for item in
...            tm_fromto_generator('20170101','20170331', 10)]
>>> tm_list[:3]
[('20170101000000', '20170110235959'),
 ('20170111000000', '20170120235959'),
 ('20170121000000', '20170130235959')]
```

unipy.**timestamp_generator**(**args*)

A range function yields pair timestep strings.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters ***args** (*int*) – end or start, end[, term] It works like range function.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

`yield`

Examples

```
>>> from unipy.utils.generator import timestamp_generator
>>> timestamp_generator(1, 10, 2)
<generator object timestamp_generator at 0x10f519678>
>>> list(timestamp_generator(1, 14, 5))
[(1, 5), (6, 10), (11, 15)]
>>> begin, fin, period = 1, 10, 3
>>> list(timestamp_generator(begin, fin, period))
[(1, 3), (4, 6), (7, 9), (10, 12)]
>>> time_sequence = timestamp_generator(begin, fin, period)
>>> time_msg = "{start:2} to {end:2}, {term:2} days."
>>> for time in time_sequence:
... b, f = time
... print(time_msg.format(start=b, end=f, term=period))
...
 1 to  3,  3 days.
 4 to  6,  3 days.
 7 to  9,  3 days.
10 to 12,  3 days.
```

unipy.**gdrive_downloader**(*gdrive_url_id*, *pattern='*'*, *download_path='./data'*)

Download files in Google Drive.

Download files in Googel Drive to the given path.

Parameters

- **gdrive_url_id** (*str*) – An URL ID of an Google Drive directory which contains files to download. *https://drive.google.com/drive/folders/<google drive URL ID>*.
- **pattern** (*str (default: '*')*) – A pattern of regular expression to filter file in the target directory.
- **download_path** (*str (default: './data')*) – A target directory to download files in given URL ID.

Returns Nothing is returned.

Return type None

See also:

`None()`

Examples

```
>>> import unipy.util.gdrive import gdrive_downloader
>>> gdrive_path_id = '1LA5334-SZdizcFqkl4xO8Hty7w1q0e8h'
>>> up.gdrive_downloader(gdrive_path_id)
```

unipy.**gdrive_uploader**(*gdrive_url_id*, *pattern='*'*, *src_dir='./data'*)

Download files in Google Drive.

Download files in Googel Drive to the given path.

Parameters

- **gdrive_url_id** (*str*) – An URL ID of an Google Drive directory to upload files. *https://drive.google.com/drive/folders/<google drive URL ID>*.
- **pattern** (*str (default: '*')*) – A pattern of regular expression to filter file in the target directory.
- **src_dir** (*str (default: './data')*) – A source directory to upload files in given URL ID.

Returns Nothing is returned.

Return type None

See also:

None()

Examples

```
>>> import unipy.util.gdrive import gdrive_uploader
>>> gdrive_path_id = '1LA5334-SZdizcFqkl4xO8Hty7w1q0e8h'
>>> up.gdrive_uploader(gdrive_path_id)
```

5.2 Subpackages

5.2.1 unipy.core package

Submodules

unipy.core.api module

Module contents

5.2.2 unipy.dataset package

Submodules

unipy.dataset.api module

Pre-made Dataset Provider.

unipy.dataset.api.**init**()

Summary *unipy* package has some famous datasets. This function unzip the embedded dataset to use.

Returns

Return type None

Examples

```
>>> import unipy.dataset.api as dm
>>> dm.init()
['iris', 'births_small', 'anscombe', 'nutrients', 'car90', 'cars',
 'breast_cancer', 'winequality_red', 'german_credit_scoring_fars2008',
 'winequality_white', 'tips', 'air_quality', 'diabetes', 'births_big',
 'adult', 'titanic']
```

unipy.dataset.api.**reset**()

Summary This function unzip the embedded dataset to use. Equal to *dm.init()*

Returns

Return type None

Examples

```
>>> import unipy.dataset.api as dm
>>> dm.reset()
```

unipy.dataset.api.**ls**()

Summary This function shows the list of the dataset.

Returns

Return type list

Examples

```
>>> import unipy.dataset.api as dm
>>> dm.init()
['iris', 'births_small', 'anscombe', 'nutrients', 'car90', 'cars',
 'breast_cancer', 'winequality_red', 'german_credit_scoring_fars2008',
 'winequality_white', 'tips', 'air_quality', 'diabetes', 'births_big',
 'adult', 'titanic']
>>> dm.ls()
['iris', 'births_small', 'anscombe', 'nutrients', 'car90', 'cars',
 'breast_cancer', 'winequality_red', 'german_credit_scoring_fars2008',
 'winequality_white', 'tips', 'air_quality', 'diabetes', 'births_big',
 'adult', 'titanic']
```

unipy.dataset.api.**load**(*pick*)

Summary This function returns a dataset you select. :param pick: You can load a dataset by its name or its index from the list of

dm.ls(). Indices start with 0.

Returns

Return type pandas.DataFrame

Examples

```
>>> import unipy.dataset.api as dm
>>> dm.init()
['iris', 'births_small', 'anscombe', 'nutrients', 'car90', 'cars',
 'breast_cancer', 'winequality_red', 'german_credit_scoring_fars2008',
 'winequality_white', 'tips', 'air_quality', 'diabetes', 'births_big',
 'adult', 'titanic']
>>> data = dm.load('anscombe')
Dataset : anscombe
>>> data = dm.load(2)
Dataset : anscombe
```

Module contents

Datasets.

This module offers you well-known datasets.

api

- *init* – Unzip datasets.
- *reset* – Re-unzip datasets.
- *ls* – List-up datasets.
- *load* – Load a dataset.

unipy.dataset.**init**()

Summary *unipy* package has some famous datasets. This function unzip the embedded dataset to use.

Returns

Return type None

Examples

```
>>> import unipy.dataset.api as dm
>>> dm.init()
['iris', 'births_small', 'anscombe', 'nutrients', 'car90', 'cars',
 'breast_cancer', 'winequality_red', 'german_credit_scoring_fars2008',
 'winequality_white', 'tips', 'air_quality', 'diabetes', 'births_big',
 'adult', 'titanic']
```

unipy.dataset.**reset**()

Summary This function unzip the embedded dataset to use. Equal to *dm.init()*

Returns

Return type None

Examples

```
>>> import unipy.dataset.api as dm
>>> dm.reset()
```

unipy.dataset.**ls**()

Summary This function shows the list of the dataset.

Returns

Return type list

Examples

```
>>> import unipy.dataset.api as dm
>>> dm.init()
['iris', 'births_small', 'anscombe', 'nutrients', 'car90', 'cars',
 'breast_cancer', 'winequality_red', 'german_credit_scoring_fars2008',
 'winequality_white', 'tips', 'air_quality', 'diabetes', 'births_big',
 'adult', 'titanic']
>>> dm.ls()
['iris', 'births_small', 'anscombe', 'nutrients', 'car90', 'cars',
 'breast_cancer', 'winequality_red', 'german_credit_scoring_fars2008',
 'winequality_white', 'tips', 'air_quality', 'diabetes', 'births_big',
 'adult', 'titanic']
```

unipy.dataset.**load**(*pick*)

Summary This function returns a dataset you select. :param pick: You can load a dataset by its name or its index from the list of

dm.ls(). Indices start with 0.

Returns

Return type pandas.DataFrame

Examples

```
>>> import unipy.dataset.api as dm
>>> dm.init()
['iris', 'births_small', 'anscombe', 'nutrients', 'car90', 'cars',
 'breast_cancer', 'winequality_red', 'german_credit_scoring_fars2008',
 'winequality_white', 'tips', 'air_quality', 'diabetes', 'births_big',
 'adult', 'titanic']
>>> data = dm.load('anscombe')
Dataset : anscombe
>>> data = dm.load(2)
Dataset : anscombe
```

5.2.3 unipy.image package

Submodules

unipy.image.api module

Created on Sun Jan 8 03:46:03 2017

@author: Young Ju Kim

unipy.image.houghmatrix module

Image Transformation.

unipy.image.houghmatrix.**rgb2gras**(*img_array*)

unipy.image.houghmatrix.**hough_transform**(*img_bin*, *theta_res=1*, *rho_res=1*)

Module contents

Image processing toolkit.

houghmatrix

- *rgb2gras* – RGB to Grayscale.
- *hough_transform* – Hough Transformation.

unipy.image.**rgb2gras**(*img_array*)

unipy.image.**hough_transform**(*img_bin*, *theta_res=1*, *rho_res=1*)

5.2.4 unipy.math package

Submodules

unipy.math.api module

Created on Wed Jan 4 20:33:37 2017

@author: Young Ju Kim

unipy.math.geometry module

Geometrical toolkit.

class unipy.math.geometry.**Ellipse**(*diameter*)

Bases: object

Create an ellipse.

diameter

radius

center

angle

coordinates()

Module contents

Mathmatical backend.

geometry

- `` – .

class unipy.math.**Ellipse**(*diameter*)

Bases: object

Create an ellipse.

diameter

radius

center

angle

coordinates()

5.2.5 unipy.plots package

Submodules

unipy.plots.api module

Created on Wed Jan 4 20:33:37 2017

@author: Young Ju Kim

unipy.plots.boxplot module

Complexed Plotting Toolkit.

unipy.plots.boxplot.**point_boxplot**(*data*, *groupby=None*, *value=None*, *rot=90*, *spread=0.2*, *dot_size=15.0*, *dot_color='b'*, *dot_alpha=0.2*, *figsize=12, 9*, **args*, ***kwargs*)

Boxplot with points.

Draw boxplots by given keys(groupby, value).

Parameters

- **data** (*pandas.DataFrame*) – a dataset.
- **groupby** (*str or list-like (default: None)*) – A key column to separate. (X-axis, categorical) When `str`, it should be a column name to groupby. When `list-like`, it contains a column name to groupby.

- **value** (*str or list-like (default: None)*) – A key column to get values. (Y-axis, numerical) When `str`, it should be a column name of values. When `list-like`, it contains a column name of values.
- **rot** (*int (default: 90)*) – A rotation angle to show X-axis labels.
- **spread** (*float (default: .2)*) – A spread ratio of points. The bigger, the pointing distribution width are broader.
- **dot_size** (*float (default: 15.)*) – A size of each points.
- **dot_color** (*int (default: 'b')*) – A color name of each points.
- **dot_alpha** (*float (default: .2)*) – A transparency value of each points.

Returns

- *matplotlib.figure.Figure* – A plot figure.
- *Exceptions*
- ———-
- *AssertionError* – It is raised when two or more names are given to `groupby` or `value`.

See also:

`pandas.DataFrame.boxplot` `matplotlib.pyplot`

Examples

```
>>> import unipy.dataset.api as dm
>>> from unipy.plots import point_boxplot
>>> dm.init()
>>> data = dm.load('iris')
Dataset : iris
>>> tmp = point_boxplot(data, groupby='species', value='sepal_length')
```

unipy.plots.boxplot.**point_boxplot_axis**(*data*, *groupby=None*, *value=None*, *rot=90*, *spread=0.2*, *dot_size=15.0*, *dot_color='b'*, *dot_alpha=0.2*, *share_yrange=True*, *figsize=12, 9*, **args*, ***kwargs*)

Boxplot with points, horizontally seperated.

Draw boxplots by given keys(groupby, value).

Parameters

- **data** (*pandas.DataFrame*) – a dataset.
- **groupby** (*str or list-like (default: None)*) – A key column to separate. (X-axis, categorical) When `str`, it should be a column name to groupby. When `list-like`, it contains a column name to groupby.
- **value** (*str or list-like (default: None)*) – A key column to get values. (Y-axis, numerical) When `str`, it should be a column name of values. When `list-like`, it contains a column name of values.
- **rot** (*int (default: 90)*) – A rotation angle to show X-axis labels.
- **spread** (*float (default: .2)*) – A spread ratio of points. The bigger, the pointing distribution width are broader.
- **dot_size** (*float (default: 15.)*) – A size of each points.

- **dot_color** (*int (default: 'b')*) – A color name of each points.
- **dot_alpha** (*float (default: .2)*) – A transparency value of each points.
- **share_yrange** (*Boolean (defalut: True)*) – False then each Y-axis limit of boxplots will draw independent.

Returns

- *matplotlib.figure.Figure* – A plot figure.
- *Exceptions*
- ———-
- *AssertionError* – It is raised when two or more names are given to `groupby` or `value`.

See also:

`pandas.DataFrame.boxplot` `matplotlib.pyplot`

Examples

```
>>> import unipy.dataset.api as dm
>>> from unipy.plots import point_boxplot_axis
>>> dm.init()
>>> data = dm.load('iris')
Dataset : iris
>>> tmp = point_boxplot_axis(data,
...                          groupby='species',
...                          value='sepal_length',
...                          share_yrange=True)
```

`unipy.plots.boxplot.`**mosaic_plot**(*data*, *groupby=None*, *col_list=None*, *show_values=True*, *rot=90*, *width=0.9*, *figsize=12, 9*, **args*, ***kwargs*)

Mosaic Plot via Stacked bar plots.

Draw plots by given keys(groupby, value).

Parameters

- **data** (*pandas.DataFrame*) – a dataset.
- **groupby** (*str or list-like (default: None)*) – A key column to separate. (X-axis, categorical) When `str`, it should be a column name to groupby. When `list-like`, it contains a column name to groupby.
- **col_list** (*str or list-like (default: None)*) – A key column to get values. (Y-axis, numerical) When `str`, it should be column names of values. When `list-like`, it contains column names of values.
- **rot** (*int (default: 90)*) – A rotation angle to show X-axis labels.
- **show_values** (*boolean (default: True)*) – Choose If *n* is annotated.

Returns

- *matplotlib.figure.Figure* – A plot figure.
- *Exceptions*
- ———-
- *AssertionError* – It is raised when two or more names are given to `groupby` or `value`.

See also:

`pandas.DataFrame.barplot` `matplotlib.pyplot`

Examples

```
>>> import unipy.dataset.api as dm
>>> from unipy.plots import mosaic_plot
>>> dm.init()
>>> data = dm.load('adult')
Dataset : iris
>>> tmp = mosaic_plot(data, groupby='native_country',
... col_list=['workclass', 'education'])
```

Module contents

Plotting Toolkit.

Boxplot

- *point_boxplot* – Boxplot with points.
- *point_boxplot_grid* – Grid-framed boxplot with points.

`unipy.plots.`**`point_boxplot`**(*data*, *groupby=None*, *value=None*, *rot=90*, *spread=0.2*, *dot_size=15.0*, *dot_color='b'*, *dot_alpha=0.2*, *figsize=12, 9*, **args*, ***kwargs*)

Boxplot with points.

Draw boxplots by given keys(groupby, value).

Parameters

- **`data`** (*`pandas.DataFrame`*) – a dataset.
- **`groupby`** (*`str or list-like (default: None)`*) – A key column to separate. (X-axis, categorical) When `str`, it should be a column name to groupby. When `list-like`, it contains a column name to groupby.
- **`value`** (*`str or list-like (default: None)`*) – A key column to get values. (Y-axis, numerical) When `str`, it should be a column name of values. When `list-like`, it contains a column name of values.
- **`rot`** (*`int (default: 90)`*) – A rotation angle to show X-axis labels.
- **`spread`** (*`float (default: .2)`*) – A spread ratio of points. The bigger, the pointing distribution width are broader.
- **`dot_size`** (*`float (default: 15.)`*) – A size of each points.
- **`dot_color`** (*`int (default: 'b')`*) – A color name of each points.
- **`dot_alpha`** (*`float (default: .2)`*) – A transparency value of each points.

Returns

- *matplotlib.figure.Figure* – A plot figure.
- *Exceptions*
- ———-

- *AssertionError* – It is raised when two or more names are given to `groupby` or `value`.

See also:

`pandas.DataFrame.boxplot` `matplotlib.pyplot`

Examples

```
>>> import unipy.dataset.api as dm
>>> from unipy.plots import point_boxplot
>>> dm.init()
>>> data = dm.load('iris')
Dataset : iris
>>> tmp = point_boxplot(data, groupby='species', value='sepal_length')
```

`unipy.plots.`**`point_boxplot_axis`**(*data*, *groupby=None*, *value=None*, *rot=90*, *spread=0.2*, *dot_size=15.0*, *dot_color='b'*, *dot_alpha=0.2*, *share_yrange=True*, *figsize=12, 9*, **args*, ***kwargs*)

Boxplot with points, horizontally seperated.

Draw boxplots by given keys(groupby, value).

Parameters

- **`data`** (*`pandas.DataFrame`*) – a dataset.
- **`groupby`** (*`str or list-like (default: None)`*) – A key column to separate. (X-axis, categorical) When `str`, it should be a column name to groupby. When `list-like`, it contains a column name to groupby.
- **`value`** (*`str or list-like (default: None)`*) – A key column to get values. (Y-axis, numerical) When `str`, it should be a column name of values. When `list-like`, it contains a column name of values.
- **`rot`** (*`int (default: 90)`*) – A rotation angle to show X-axis labels.
- **`spread`** (*`float (default: .2)`*) – A spread ratio of points. The bigger, the pointing distribution width are broader.
- **`dot_size`** (*`float (default: 15.)`*) – A size of each points.
- **`dot_color`** (*`int (default: 'b')`*) – A color name of each points.
- **`dot_alpha`** (*`float (default: .2)`*) – A transparency value of each points.
- **`share_yrange`** (*`Boolean (defalut: True)`*) – False then each Y-axis limit of boxplots will draw independent.

Returns

- *matplotlib.figure.Figure* – A plot figure.
- *Exceptions*
- ———-
- *AssertionError* – It is raised when two or more names are given to `groupby` or `value`.

See also:

`pandas.DataFrame.boxplot` `matplotlib.pyplot`

Examples

```
>>> import unipy.dataset.api as dm
>>> from unipy.plots import point_boxplot_axis
>>> dm.init()
>>> data = dm.load('iris')
Dataset : iris
>>> tmp = point_boxplot_axis(data,
...                          groupby='species',
...                          value='sepal_length',
...                          share_yrange=True)
```

unipy.plots.**mosaic_plot**(*data*, *groupby=None*, *col_list=None*, *show_values=True*, *rot=90*, *width=0.9*, *figsize=12, 9*, **args*, ***kwargs*)

Mosaic Plot via Stacked bar plots.

Draw plots by given keys(groupby, value).

Parameters

- **data** (*pandas.DataFrame*) – a dataset.
- **groupby** (*str or list-like (default: None)*) – A key column to separate. (X-axis, categorical) When `str`, it should be a column name to groupby. When `list-like`, it contains a column name to groupby.
- **col_list** (*str or list-like (default: None)*) – A key column to get values. (Y-axis, numerical) When `str`, it should be column names of values. When `list-like`, it contains column names of values.
- **rot** (*int (default: 90)*) – A rotation angle to show X-axis labels.
- **show_values** (*boolean (default: True)*) – Choose If *n* is annotated.

Returns

- *matplotlib.figure.Figure* – A plot figure.
- *Exceptions*
- ———-
- *AssertionError* – It is raised when two or more names are given to `groupby` or `value`.

See also:

`pandas.DataFrame.barplot` `matplotlib.pyplot`

Examples

```
>>> import unipy.dataset.api as dm
>>> from unipy.plots import mosaic_plot
>>> dm.init()
>>> data = dm.load('adult')
Dataset : iris
>>> tmp = mosaic_plot(data, groupby='native_country',
... col_list=['workclass', 'education'])
```

5.2.6 unipy.stats package

Submodules

unipy.stats.api module

Created on Sun Jan 8 03:46:03 2017

@author: Young Ju Kim

unipy.stats.feature_selection module

Feature selection.

unipy.stats.feature_selection.**lasso_rank**(*formula=None*, *X=None*, *y=None*, *data=None*, *alpha=array([1e-05, 0.00011, 0.00021, 0.00031, 0.00041, 0.00051, 0.00061, 0.00071, 0.00081, 0.00091, 0.00101, 0.00111, 0.00121, 0.00131, 0.00141, 0.00151, 0.00161, 0.00171, 0.00181, 0.00191, 0.00201, 0.00211, 0.00221, 0.00231, 0.00241, 0.00251, 0.00261, 0.00271, 0.00281, 0.00291, 0.00301, 0.00311, 0.00321, 0.00331, 0.00341, 0.00351, 0.00361, 0.00371, 0.00381, 0.00391, 0.00401, 0.00411, 0.00421, 0.00431, 0.00441, 0.00451, 0.00461, 0.00471, 0.00481, 0.00491, 0.00501, 0.00511, 0.00521, 0.00531, 0.00541, 0.00551, 0.00561, 0.00571, 0.00581, 0.00591, 0.00601, 0.00611, 0.00621, 0.00631, 0.00641, 0.00651, 0.00661, 0.00671, 0.00681, 0.00691, 0.00701, 0.00711, 0.00721, 0.00731, 0.00741, 0.00751, 0.00761, 0.00771, 0.00781, 0.00791, 0.00801, 0.00811, 0.00821, 0.00831, 0.00841, 0.00851, 0.00861, 0.00871, 0.00881, 0.00891, 0.00901, 0.00911, 0.00921, 0.00931, 0.00941, 0.00951, 0.00961, 0.00971, 0.00981, 0.00991])*, *k=2*, *plot=False*, **args*, ***kwargs*)

Feature selection by LASSO regression.

Parameters

- **formula** – R-style formula string
- **X** (*list-like*) – Column values for X.
- **y** (*list-like*) – A column value for y.
- **data** (*pandas.DataFrame*) – A DataFrame.
- **alpha** (*Iterable*) – An Iterable contains alpha values.
- **k** (*int*) – Threshold of coefficient matrix
- **plot** (*Boolean (default: False)*) – True if want to plot the result.

Returns

- **rankTbl** (*pandas.DataFrame*) – Feature ranking by given k.
- **minIntercept** (*pandas.DataFrame*) – The minimum intercept row in coefficient matrix.

- **coefMatrix** (*pandas.DataFrame*) – A coefficient matrix.
- **kBest** (*pandas.DataFrame*) – When Given `k`, The best intercept row in coefficient matrix.
- **kBestPredY** (*dict*) – A predicted `Y` with `kBest` alpha.

Example

```
>>> import unipy.dataset.api as dm
>>> dm.init()
['cars', 'anscombe', 'iris', 'nutrients', 'german_credit_scoring_fars2008',
↪'winequality_red', 'winequality_white', 'titanic', 'car90', 'diabetes', 'adult',
↪ 'tips', 'births_big', 'breast_cancer', 'air_quality', 'births_small']
>>> wine_red = dm.load('winequality_red')
Dataset : winequality_red
>>>
>>> ranked, best_by_intercept, coefTbl, kBest, kBestPred = lasso_rank(X=wine_red.
↪columns.drop('quality'), y=['quality'], data=wine_red)
>>> ranked
                  rank  lasso_coef  abs_coef
volatile_acidity     1   -0.675725  0.675725
alcohol              2    0.194865  0.194865
>>> best_by_intercept
                      RSS  Intercept  fixed_acidity  volatile_acidity      alpha_
↪0.00121  691.956364   3.134874       0.002374         -1.023793
```

citric_acid residual_sugar chlorides free_sulfur_dioxide alpha_0.00121 0.0 0.0 -0.272912 -0.0

total_sulfur_dioxide density pH sulphates alcohol alpha_0.00121 -0.000963 -0.0 -0.0 0.505956 0.264552

var_count

alpha_0.00121 6 >>>

`unipy.stats.feature_selection.`**`feature_selection_vif`**(*data*, *thresh=5.0*)

Stepwise Feature Selection for multivariate analysis.

It calculates OLS regressions and the variance inflation factors iterating all explanatory variables. If the maximum VIF of a variable is over the given threshold, It will be dropped. This process is repeated until all VIFs are lower than the given threshold.

Recommended threshold is lower than 5, because if VIF is greater than 5, then the explanatory variable selected is highly collinear with the other explanatory variables, and the parameter estimates will have large standard errors because of this.

Parameters

- **data** (*DataFrame, (rows: observed values, columns: multivariate variables)*) – design dataframe with all explanatory variables, as for example used in regression
- **thresh** (*int, float*) – A threshold of VIF

Returns

- **Filtered_data** (*DataFrame*) – A subset of the input DataFame
- **dropped_List** (*DataFrame*) – 'var' column : dropped variable names from input data columns 'vif' column : variance inflation factor of dropped variables

Notes

This function does not save the auxiliary regression.

See also:

`statsmodels.stats.outliers_influence.variance_inflation_factor()`

References

http://en.wikipedia.org/wiki/Variance_inflation_factor

unipy.stats.formula module

Created on Tue Aug 8 01:04:13 2017

@author: Young Ju Kim

`unipy.stats.formula.`**`from_formula`**(*formula*)
R-style Formula Formatting.

unipy.stats.hypo_test module

Statistical Hypothesis Tests.

`unipy.stats.hypo_test.`**`f_test`**(*a*, *b*, *scale=1*, *alternative='two-sided'*, *conf_level=0.95*, **args*, ***kwargs*)
F-Test.

`unipy.stats.hypo_test.`**`f_test_formula`**(*a*, *b*, *scale=1*, *alternative='two-sided'*, *conf_level=0.95*, **args*, ***kwargs*)
F-Test by formula.

`unipy.stats.hypo_test.`**`anova_test`**(*formula*, *data=None*, *typ=1*)
ANOVA Test.

`unipy.stats.hypo_test.`**`anova_test_formula`**(*formula*, *data=None*, *typ=1*)
ANOVA Test by formula.

`unipy.stats.hypo_test.`**`chisq_test`**(*data*, *x=None*, *y=None*, *correction=None*, *lambda_=None*, *margin=True*, *print_ok=True*)
Chi-square Test.

`lambda_` gives the power in the Cressie-Read power divergence statistic. The default is 1. For convenience, **lambda_** may be assigned one of the following strings, in which case the corresponding numerical value is used:

Parameters

- **data** (*pandas.DataFrame*) –
- **x** (*str (default: None)*) –
- **y** (*str (default: None)*) –
- **correction** (*(default: None)*) –
- **lambda_** (*lambda (default: None)*) –
- **margin** (*Boolean (default: True)*) –
- **print_ok** (*Boolean (default: True)*) –

unipy.stats.hypo_test.**fisher_test**(*data*, *x=None*, *y=None*, *alternative='two-sided'*, *margin=True*, *print_ok=True*)
Fisher's Exact Test.

unipy.stats.metrics module

Metric Functions.

unipy.stats.metrics.**deviation**(*container*, *method='mean'*, *if_abs=True*)
Deviation.

unipy.stats.metrics.**vif**(*y*, *X*)
Variance inflation factor.

unipy.stats.metrics.**mean_absolute_percentage_error**(*measure*, *predict*, *thresh=3.0*)
Mean Absolute Percentage Error. It is a percent of errors. It measures the prediction accuracy of a forecasting method in Statistics with the real mesured values and the predicted values, for example in trend estimation. If MAPE is 5, it means this prediction method potentially has 5% error. It cannot be used if there are zero values, because there would be a division by zero.

unipy.stats.metrics.**average_absolute_deviation**(*measure*, *predict*, *thresh=2*)
Average Absolute Deviation. It is ... It measures the prediction accuracy of a forecasting method in Statistics with the real mesured values and the predicted values, for example in trend estimation. If MAD is 5, it means this prediction method potentially has...

unipy.stats.metrics.**median_absolute_deviation**(*measure*, *predict*, *thresh=2*)
Median Absolute Deviation. It is ... It measures the prediction accuracy of a forecasting method in Statistics with the real mesured values and the predicted values, for example in trend estimation. If MAD is 5, it means this prediction method potentially has...

unipy.stats.metrics.**calculate_interaction**(*rankTbl*, *pvTbl*, *target*, *ranknum=10*)
Feature interaction calculation.

Module contents

Utility Objects.

This module provides a number of functions and objects for utility.

hypo_test

- *f_test* – F-Test.
- *f_test_formula* – F-Test by formula.
- *anova_test* – ANOVA Test.
- *anova_test_formula* – ANOVA Test by formula.
- *chisq_test* – Chi-square Test.
- *fisher_test* – Fisher's Exact Test.

feature_selection

- *lasso_rank* – Feature selection by LASSO regression.
- ***feature_selection_vif* – VIF based stepwise feature selection** for multivariate analysis.

metrics

- *deviation* – Deviation.
- *vif* – Variance inflation factor.
- *mean_absolute_percentage_error* – Mean Absolute Percentage Error.
- *average_absolute_deviation* – Average Absolute Deviation.
- *median_absolute_deviation* – Median Absolute Deviation.
- *calculate_interaction* – Feature interaction calculation.

formula

- *from_formula* – R-style Formula Formatting.

unipy.stats.**deviation**(*container*, *method='mean'*, *if_abs=True*)
Deviation.

unipy.stats.**vif**(*y*, *X*)
Variance inflation factor.

unipy.stats.**mean_absolute_percentage_error**(*measure*, *predict*, *thresh=3.0*)
Mean Absolute Percentage Error. It is a percent of errors. It measures the prediction accuracy of a forecasting method in Statistics with the real mesured values and the predicted values, for example in trend estimation. If MAPE is 5, it means this prediction method potentially has 5% error. It cannot be used if there are zero values, because there would be a division by zero.

unipy.stats.**average_absolute_deviation**(*measure*, *predict*, *thresh=2*)
Average Absolute Deviation. It is ... It measures the prediction accuracy of a forecasting method in Statistics with the real mesured values and the predicted values, for example in trend estimation. If MAD is 5, it means this prediction method potentially has...

unipy.stats.**median_absolute_deviation**(*measure*, *predict*, *thresh=2*)
Median Absolute Deviation. It is ... It measures the prediction accuracy of a forecasting method in Statistics with the real mesured values and the predicted values, for example in trend estimation. If MAD is 5, it means this prediction method potentially has...

unipy.stats.**calculate_interaction**(*rankTbl*, *pvTbl*, *target*, *ranknum=10*)
Feature interaction calculation.

unipy.stats.**f_test**(*a*, *b*, *scale=1*, *alternative='two-sided'*, *conf_level=0.95*, **args*, ***kwargs*)
F-Test.

unipy.stats.**f_test_formula**(*a*, *b*, *scale=1*, *alternative='two-sided'*, *conf_level=0.95*, **args*, ***kwargs*)
F-Test by formula.

unipy.stats.**anova_test**(*formula*, *data=None*, *typ=1*)
ANOVA Test.

unipy.stats.**anova_test_formula**(*formula*, *data=None*, *typ=1*)
ANOVA Test by formula.

unipy.stats.**chisq_test**(*data*, *x=None*, *y=None*, *correction=None*, *lambda_=None*, *margin=True*, *print_ok=True*)
Chi-square Test.

lambda_ gives the power in the Cressie-Read power divergence statistic. The default is 1. For convenience, **lambda_** may be assigned one of the following strings, in which case the corresponding numerical value is used:

Parameters

- **data** (*pandas.DataFrame*) –
- **x** (*str (default: None)*) –
- **y** (*str (default: None)*) –
- **correction** (*(default: None)*) –
- **lambda_** (*lambda (default: None)*) –
- **margin** (*Boolean (default: True)*) –
- **print_ok** (*Boolean (default: True)*) –

unipy.stats.**fisher_test**(*data*, *x=None*, *y=None*, *alternative='two-sided'*, *margin=True*, *print_ok=True*)
Fisher's Exact Test.

unipy.stats.**lasso_rank**(*formula=None*, *X=None*, *y=None*, *data=None*, *alpha=array([1e-05, 0.00011, 0.00021, 0.00031, 0.00041, 0.00051, 0.00061, 0.00071, 0.00081, 0.00091, 0.00101, 0.00111, 0.00121, 0.00131, 0.00141, 0.00151, 0.00161, 0.00171, 0.00181, 0.00191, 0.00201, 0.00211, 0.00221, 0.00231, 0.00241, 0.00251, 0.00261, 0.00271, 0.00281, 0.00291, 0.00301, 0.00311, 0.00321, 0.00331, 0.00341, 0.00351, 0.00361, 0.00371, 0.00381, 0.00391, 0.00401, 0.00411, 0.00421, 0.00431, 0.00441, 0.00451, 0.00461, 0.00471, 0.00481, 0.00491, 0.00501, 0.00511, 0.00521, 0.00531, 0.00541, 0.00551, 0.00561, 0.00571, 0.00581, 0.00591, 0.00601, 0.00611, 0.00621, 0.00631, 0.00641, 0.00651, 0.00661, 0.00671, 0.00681, 0.00691, 0.00701, 0.00711, 0.00721, 0.00731, 0.00741, 0.00751, 0.00761, 0.00771, 0.00781, 0.00791, 0.00801, 0.00811, 0.00821, 0.00831, 0.00841, 0.00851, 0.00861, 0.00871, 0.00881, 0.00891, 0.00901, 0.00911, 0.00921, 0.00931, 0.00941, 0.00951, 0.00961, 0.00971, 0.00981, 0.00991])*, *k=2*, *plot=False*, **args*, ***kwargs*)
Feature selection by LASSO regression.

Parameters

- **formula** – R-style formula string
- **X** (*list-like*) – Column values for X.
- **y** (*list-like*) – A column value for y.
- **data** (*pandas.DataFrame*) – A DataFrame.
- **alpha** (*Iterable*) – An Iterable contains alpha values.
- **k** (*int*) – Threshold of coefficient matrix
- **plot** (*Boolean (default: False)*) – True if want to plot the result.

Returns

- **rankTbl** (*pandas.DataFrame*) – Feature ranking by given k.

- **minIntercept** (*pandas.DataFrame*) – The minimum intercept row in coefficient matrix.
- **coefMatrix** (*pandas.DataFrame*) – A coefficient matrix.
- **kBest** (*pandas.DataFrame*) – When Given `k`, The best intercept row in coefficient matrix.
- **kBestPredY** (*dict*) – A predicted `Y` with `kBest` alpha.

Example

```
>>> import unipy.dataset.api as dm
>>> dm.init()
['cars', 'anscombe', 'iris', 'nutrients', 'german_credit_scoring_fars2008',
↪'winequality_red', 'winequality_white', 'titanic', 'car90', 'diabetes', 'adult',
↪ 'tips', 'births_big', 'breast_cancer', 'air_quality', 'births_small']
>>> wine_red = dm.load('winequality_red')
Dataset : winequality_red
>>>
>>> ranked, best_by_intercept, coefTbl, kBest, kBestPred = lasso_rank(X=wine_red.
↪columns.drop('quality'), y=['quality'], data=wine_red)
>>> ranked
                  rank  lasso_coef  abs_coef
volatile_acidity     1   -0.675725  0.675725
alcohol              2    0.194865  0.194865
>>> best_by_intercept
                     RSS  Intercept  fixed_acidity  volatile_acidity      alpha_
↪0.00121  691.956364   3.134874       0.002374         -1.023793
```

citric_acid residual_sugar chlorides free_sulfur_dioxide alpha_0.00121 0.0 0.0 -0.272912 -0.0

total_sulfur_dioxide density pH sulphates alcohol alpha_0.00121 -0.000963 -0.0 -0.0 0.505956 0.264552

var_count

alpha_0.00121 6 >>>

`unipy.stats.`**`feature_selection_vif`**(*data*, *thresh=5.0*)

Stepwise Feature Selection for multivariate analysis.

It calculates OLS regressions and the variance inflation factors iterating all explanatory variables. If the maximum VIF of a variable is over the given threshold, It will be dropped. This process is repeated until all VIFs are lower than the given threshold.

Recommended threshold is lower than 5, because if VIF is greater than 5, then the explanatory variable selected is highly collinear with the other explanatory variables, and the parameter estimates will have large standard errors because of this.

Parameters

- **data** (*DataFrame, (rows: observed values, columns: multivariate variables)*) – design dataframe with all explanatory variables, as for example used in regression
- **thresh** (*int, float*) – A threshold of VIF

Returns

- **Filtered_data** (*DataFrame*) – A subset of the input DataFame

- **dropped_List** (*DataFrame*) – 'var' column : dropped variable names from input data columns 'vif' column : variance inflation factor of dropped variables

Notes

This function does not save the auxiliary regression.

See also:

`statsmodels.stats.outliers_influence.variance_inflation_factor()`

References

http://en.wikipedia.org/wiki/Variance_inflation_factor

`unipy.stats.`**`from_formula`**(*formula*)
R-style Formula Formatting.

5.2.7 unipy.tools package

Submodules

unipy.tools.api module

Created on Fri Jun 2 13:41:19 2017

@author: Young Ju Kim

unipy.tools.data_handler module

Data manipulation tools.

`unipy.tools.data_handler.`**`exc`**(*source*, *blacklist*)
Get items except the given list.

This function splits an Iterable into the given size of multiple chunks. The items of An iterable should be the same type.

Parameters

- **`source`** (*`Iterable`*) – An Iterable to filter.
- **`blacklist`** (*`Iterable`*) – A list contains items to eliminate.

Returns A filtered list.

Return type list

See also:

`Infix Operator`

Examples

```
>>> import unipy as up
>>> up.splitter(list(range(10)), how='equal', size=3)
[(0, 1, 2, 3), (4, 5, 6), (7, 8, 9)]
```

```
>>> up.splitter(list(range(10)), how='remaining', size=3)
[(0, 1, 2), (3, 4, 5), (6, 7, 8), (9,)]
```

unipy.tools.data_handler.**splitter**(*iterable*, *how='equal'*, *size=2*)

Split data with given size.

This function splits an Iterable into the given size of multiple chunks. The items of An iterable should be the same type.

Parameters

- **iterable** (*Iterable*) – An Iterable to split.
- **how** (*{'equal', 'remaining'}*) – The method to split. 'equal' is to split chunks with the approximate length within the given size. 'remaining' is to split chunks with the given size, and the remains are bound as the last chunk.
- **size** (*int*) – The number of chunks.

Returns A list of chunks.

Return type list

See also:

numpy.array_split(), itertools.islice()

Examples

```
>>> import unipy as up
>>> up.splitter(list(range(10)), how='equal', size=3)
[(0, 1, 2, 3), (4, 5, 6), (7, 8, 9)]
```

```
>>> up.splitter(list(range(10)), how='remaining', size=3)
[(0, 1, 2), (3, 4, 5), (6, 7, 8), (9,)]
```

unipy.tools.data_handler.**even_chunk**(*iterable*, *chunk_size*, **args*, ***kwargs*)

Split data into even size.

This function splits an Iterable into the given size of multiple chunks. The items of An iterable should be the same type.

Parameters

- **iterable** (*Iterable*) – An Iterable to split. If N-dimensional, It is chunked by 1st dimension.
- **chunk_size** (int) – The length of each chunks.

Returns A generator yields a list of chunks. The data type of the elements in a list are equal to the source data type.

Return type generator

See also:

`itertools.islice` `yield from`

Examples

```
>>> import numpy as np
>>> from unipy.tools.data_handler import even_chunk
>>> data = list(range(7))  # list, 1D
>>> print(data)
[0, 1, 2, 3, 4, 5, 6]
>>> chunked_gen = even_chunk(data, 3)
>>> print(chunked_gen)
<generator object even_chunk at 0x7fc4924897d8>
>>> next(chunked_gen)
[0, 1, 2]
>>> chunked = list(even_chunk(data, 3))
>>> print(chunked)
[[0, 1, 2], [3, 4, 5], [6]]
>>> data = np.arange(30).reshape(-1, 3)  # np.ndarray, 2D
>>> print(data)
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]])
>>> chunked_gen = even_chunk(data, 4)
>>> next(chunked_gen)
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8]), array([ 9, 10, 11])]
>>> next(chunked_gen)
[array([12, 13, 14]),
 array([15, 16, 17]),
 array([18, 19, 20]),
 array([21, 22, 23])]
>>> next(chunked_gen)
[array([24, 25, 26]), array([27, 28, 29])]
>>> next(chunked_gen)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
StopIteration
```

`unipy.tools.data_handler.`**`pair_unique`**`(*args)`

Get Unique pairsets.

This function gets an unique pair-sets of given data.

Parameters **`iterable`** (*`Iterable`*) – Iterables having an equal length.

Returns A list of tuples. Each tuple is an unique pair of values.

Return type list

Raises **`ValueError`** – If the lengths of argments are not equal.

See also:

zip set

Examples

```
>>> from unipy.tools.data_handler import pair_unique
>>> data = dm.load('titanic')
Dataset : titanic
>>> data.head()
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
>>> pair_unique(data.iloc[:, 0], data.iloc[:, 1])
[(5, '1st'), (19, '3rd'), (29, '1st'), (20, 'Crew'),
 (21, '1st'), (3, '3rd'), (16, 'Crew'), (26, '2nd'),
 (23, '3rd'), (10, '2nd'), (24, 'Crew'), (7, '3rd'),
 (4, 'Crew'), (27, '3rd'), (18, '2nd'), (28, 'Crew'),
 (30, '2nd'), (11, '3rd'), (2, '2nd'), (1, '1st'),
 (14, '2nd'), (31, '3rd'), (22, '2nd'), (17, '1st'),
 (8, 'Crew'), (9, '1st'), (32, 'Crew'), (15, '3rd'),
 (6, '2nd'), (12, 'Crew'), (13, '1st'), (25, '1st')]
>>> idx1 = [1, 2, 3]
>>> idx2 = [0, 9, 8, 4]
>>> pair_unique(idx1, idx2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: All argments should have the same length.
```

unipy.tools.data_handler.**df_pair_unique**(*data_frame*, *col_list*, *to_frame=False*)

Get unique pairsets in pandas.DataFrame.

This function gets an unique pair-sets of given columns.

Parameters

- **data_frame** (*pandas.DataFrame*) – DataFrame to get unique-pairs.
- **col_list** (*pandas.Index, list, tuple*) – Column names of given DataFrame.
- **to_frame** (*Boolean (default: False)*) – Choose output type. If True, It returns pandas.DataFrame as an output. If False, It returns a list of tuples.

Returns

- *list* – If to_frame=False, A list of tuples is returned. Each tuple is an unique pair of values.
- *pandas.DataFrame* – If to_frame=True, pandas.DataFrame is returned. Each row is an unique pair of values.

See also:

pandas.DataFrame.itertuples()

Examples

```
>>> from unipy.tools.data_handler import df_pair_unique
>>> data = dm.load('titanic')
Dataset : titanic
>>> data.head()
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
>>> df_pair_unique(data, ['Class', 'Sex'])
[('3rd', 'Male'), ('2nd', 'Male'), ('2nd', 'Female'), ('1st', 'Female'),
 ('Crew', 'Male'), ('1st', 'Male'), ('Crew', 'Female'), ('3rd', 'Female')]
>>> df_pair_unique(data, ['Class', 'Sex'], to_frame=True)
  Class     Sex
0   3rd    Male
1   2nd    Male
2   2nd  Female
3   1st  Female
4  Crew    Male
5   1st    Male
6  Crew  Female
7   3rd  Female
```

unipy.tools.data_handler.**map_to_tuple**(*iterable*)
Only for some specific reason.

unipy.tools.data_handler.**map_to_list**(*iterable*)
Only for some specific reason.

unipy.tools.data_handler.**merge_csv**(*file_path*, *pattern='*.csv'*, *sep=','*, *if_save=True*, *save_name=None*, *low_memory=True*)
Merge seperated csv type datasets into one dataset. Summary

This function get separated data files together. When merged, the file is sorted by its name in ascending order.

Parameters

- **file_path** (*str*) – A directory path of source files.
- **pattern** (*str*) – A File extension with conditional naming. (default: '*.csv')
- **sep** (*int*) – A symbol seperating data columns.
- **if_save** (*Boolean (Optional, default: True)*) – False if you don't want to save the result.
- **save_name** (*str*) – A filename to save the result. It should be given if if_save=True. If inappropriate name is given, the first name of file list is used.
- **low_memory** (*Boolean (Optional, default: True)*) – It is used for pandas.read_csv() option only.

Returns A concatenated DataFrame.

Return type pandas.DataFrame

Examples

```
>>> from unipy.tools.data_handler import merge_csv
>>> data = dm.load('titanic')
Dataset : titanic
>>> data.head(9)
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
5   2nd  Female  Child       No     0
6   3rd  Female  Child       No    17
7  Crew  Female  Child       No     0
8   1st    Male  Adult       No   118
>>> data.iloc[:2, :].to_csv('tmp1.csv', header=True, index=False)
>>> data.iloc[2:4, :].to_csv('tmp2.csv', header=True, index=False)
>>> data.iloc[4:9, :].to_csv('tmp3.csv', header=True, index=False)
>>> merged = merge_csv('./')
>>> merged
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
5   2nd  Female  Child       No     0
6   3rd  Female  Child       No    17
7  Crew  Female  Child       No     0
8   1st    Male  Adult       No   118
```

unipy.tools.data_handler.**nancumsum**(*iterable*)

A cumulative sum function.

A cumulative sum function.

Parameters **iterable** (*Iterable*) – Iterables to calculate cumulative sum.

Yields *int* – A cumulative summed value.

See also:

numpy.isnan()

Examples

```
>>> from unipy.tools.data_handler import nancumsum
>>> tmp = [1, 2, 4]
>>> nancumsum(tmp)
<generator object nancumsum at 0x1084553b8>
>>> list(nancumsum(tmp))
[1, 3, 7]
```

unipy.tools.data_handler.**depth**(*iterable*)

Get dimension depth.

Get a dimension depth number of a nested iterable.

Parameters **iterable** (*iterable*) – An Iterable to get a dimension depth number.

Returns A dimension depth number.

Return type int

See also:

`collections.Iterable()`

Examples

```
>>> from unipy.tools.data_handler import depth
>>> tmp = [(1, 3), (4, 6), (7, 9), (10, 12)]
>>> depth(tmp)
2
>>> tmp3d = [[np.arange(i) + i for i in range(2, j)]
...          for j in range(5, 10)]
>>> depth(tmp3d)
3
>>> # It can handle dict type (considering values only).
>>> tmp3d_dict = [{'key' + str(i): np.arange(i) + i for i in range(2, j)}
...               for j in range(5, 10)]
>>> depth(tmp3d_dict)
3
```

unipy.tools.data_handler.**zero_padder_2d**(*arr*, *max_len=None*, *method='backward'*)

Zero-padding for fixed-length inputs(2D).

Zero-padding Function with nested sequence. Each elements of a given sequence is padded fixed-length.

Parameters

- **arr** (*Iterable*) – A nested sequence containing 1-Dimensional numpy.ndarray.
- **max_len** (*int (default: None)*) – A required fixed-length of each sequences. If None, It calculates the max length of elements as max_len.
- **method** (*{'forward', 'backward'} (default: 'backward')*) – where to pad.

Returns A list containing 3-Dimensional numpy.ndarray with fixed-length 2D.

Return type list

See also:

`unipy.depth()`, `numpy.pad()`, `numpy.stack()`

Examples

```
>>> from unipy.tools.data_handler import zero_padder_2d
>>> tmp = [np.arange(i) + i for i in range(2, 5)]
>>> tmp
[array([2, 3]), array([3, 4, 5]), array([4, 5, 6, 7])]
>>> zero_padder_2d(tmp)
array([[2, 3, 0, 0],
       [3, 4, 5, 0],
       [4, 5, 6, 7]])
```

(continues on next page)

(continued from previous page)

```
>>> zero_padder_2d(tmp, max_len=6)
array([[2, 3, 0, 0, 0, 0],
       [3, 4, 5, 0, 0, 0],
       [4, 5, 6, 7, 0, 0]])
 >>> zero_padder_2d(tmp, max_len=5, method='forward')
array([[0, 0, 0, 2, 3],
       [0, 0, 3, 4, 5],
       [0, 4, 5, 6, 7]])
```

unipy.tools.data_handler.**zero_padder_3d**(*arr*, *max_len=None*, *method='backward'*)

Zero-padding for fixed-length inputs(3D).

Zero-padding Function with nested sequence. Each elements of a given sequence is padded fixed-length.

Parameters

- **arr** (*Iterable*) – A nested sequence containing 2-Dimensional numpy.ndarray.
- **max_len** (*int (default: None)*) – A required fixed-length of each sequences. If None, It calculates the max length of elements as max_len.
- **method** (*{'forward', 'backward'} (default: 'backward')*) – where to pad.

Returns A list containing 3-Dimensional numpy.ndarray with fixed-length 2D.

Return type list

Raises **ValueError** – All 3D shape of inner numpy.ndarray is not equal.

See also:

`unipy.depth()`, `numpy.pad()`, `numpy.stack()`

Examples

```
>>> from unipy.tools.data_handler import zero_padder_3d
>>> tmp3d = [np.arange(i * 2).reshape(-1, 2) for i in range(1, 5)]
>>> tmp3d
[array([[0, 1]]),
 array([[0, 1],
        [2, 3]]),
 array([[0, 1],
        [2, 3],
        [4, 5]]),
 array([[0, 1],
        [2, 3],
        [4, 5],
        [6, 7]])]
>>> zero_padder_3d(tmp3d)
array([[[0, 1],
        [0, 0],
        [0, 0],
        [0, 0]],
```

[[0, 1], [2, 3], [0, 0], [0, 0]],

[[0, 1], [2, 3], [4, 5], [0, 0]],

[[0, 1], [2, 3], [4, 5], [6, 7]]])

```
>>> tmp3d_eye
[array([[1.]]),
 array([[1., 0.],
        [0., 1.]]),
 array([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]]),
 array([[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]])]
>>> zero_padder_3d(tmp3d_eye)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 24, in zero_padder_3d
ValueError: 3D shape should be equal.
```

Module contents

Data manipulation tools.

This module provides a number of useful functions for data handling.

data_handler

- *exc* – Get items except the given list.
- *splitter* – Split data with given size.
- *even_chunk* – Split data into even size.
- *pair_unique* – Get Unique pairsets.
- *df_pair_unique* – Get unique pairsets in pandas.DataFrame.
- *merge_csv* – Merge seperated csv type datasets into one dataset.
- *nancumsum* – A cumulative sum function.
- *depth* – Get dimension depth.
- *zero_padder_2d* – Zero-padding for fixed-length inputs(2D).
- *zero_padder_3d* – Zero-padding for fixed-length inputs(3D).

`unipy.tools.`**`exc`**(*source*, *blacklist*)

Get items except the given list.

This function splits an Iterable into the given size of multiple chunks. The items of An iterable should be the same type.

Parameters

- **`source`** (*`Iterable`*) – An Iterable to filter.
- **`blacklist`** (*`Iterable`*) – A list contains items to eliminate.

Returns A filtered list.

Return type list

See also:

`Infix Operator`

Examples

```
>>> import unipy as up
>>> up.splitter(list(range(10)), how='equal', size=3)
[(0, 1, 2, 3), (4, 5, 6), (7, 8, 9)]
```

```
>>> up.splitter(list(range(10)), how='remaining', size=3)
[(0, 1, 2), (3, 4, 5), (6, 7, 8), (9,)]
```

unipy.tools.**splitter**(*iterable*, *how='equal'*, *size=2*)

Split data with given size.

This function splits an Iterable into the given size of multiple chunks. The items of An iterable should be the same type.

Parameters

- **iterable** (*Iterable*) – An Iterable to split.
- **how** (*{'equal', 'remaining'}*) – The method to split. 'equal' is to split chunks with the approximate length within the given size. 'remaining' is to split chunks with the given size, and the remains are bound as the last chunk.
- **size** (*int*) – The number of chunks.

Returns A list of chunks.

Return type list

See also:

`numpy.array_split()`, `itertools.islice()`

Examples

```
>>> import unipy as up
>>> up.splitter(list(range(10)), how='equal', size=3)
[(0, 1, 2, 3), (4, 5, 6), (7, 8, 9)]
```

```
>>> up.splitter(list(range(10)), how='remaining', size=3)
[(0, 1, 2), (3, 4, 5), (6, 7, 8), (9,)]
```

unipy.tools.**even_chunk**(*iterable*, *chunk_size*, **args*, ***kwargs*)

Split data into even size.

This function splits an Iterable into the given size of multiple chunks. The items of An iterable should be the same type.

Parameters

- **iterable** (*Iterable*) – An Iterable to split. If N-dimensional, It is chunked by 1st dimension.
- **chunk_size** (int) – The length of each chunks.

Returns A generator yields a list of chunks. The data type of the elements in a list are equal to the source data type.

Return type generator

See also:

`itertools.islice` `yield from`

Examples

```
>>> import numpy as np
>>> from unipy.tools.data_handler import even_chunk
>>> data = list(range(7))  # list, 1D
>>> print(data)
[0, 1, 2, 3, 4, 5, 6]
>>> chunked_gen = even_chunk(data, 3)
>>> print(chunked_gen)
<generator object even_chunk at 0x7fc4924897d8>
>>> next(chunked_gen)
[0, 1, 2]
>>> chunked = list(even_chunk(data, 3))
>>> print(chunked)
[[0, 1, 2], [3, 4, 5], [6]]
>>> data = np.arange(30).reshape(-1, 3)  # np.ndarray, 2D
>>> print(data)
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]])
>>> chunked_gen = even_chunk(data, 4)
>>> next(chunked_gen)
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8]), array([ 9, 10, 11])]
>>> next(chunked_gen)
[array([12, 13, 14]),
 array([15, 16, 17]),
 array([18, 19, 20]),
 array([21, 22, 23])]
>>> next(chunked_gen)
[array([24, 25, 26]), array([27, 28, 29])]
>>> next(chunked_gen)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
StopIteration
```

`unipy.tools.`**`pair_unique`**`(*args)`

Get Unique pairsets.

This function gets an unique pair-sets of given data.

Parameters **`iterable`** (*`Iterable`*) – Iterables having an equal length.

Returns A list of tuples. Each tuple is an unique pair of values.

Return type list

Raises `ValueError` – If the lengths of argments are not equal.

See also:

`zip` `set`

Examples

```
>>> from unipy.tools.data_handler import pair_unique
>>> data = dm.load('titanic')
Dataset : titanic
>>> data.head()
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
>>> pair_unique(data.iloc[:, 0], data.iloc[:, 1])
[(5, '1st'), (19, '3rd'), (29, '1st'), (20, 'Crew'),
 (21, '1st'), (3, '3rd'), (16, 'Crew'), (26, '2nd'),
 (23, '3rd'), (10, '2nd'), (24, 'Crew'), (7, '3rd'),
 (4, 'Crew'), (27, '3rd'), (18, '2nd'), (28, 'Crew'),
 (30, '2nd'), (11, '3rd'), (2, '2nd'), (1, '1st'),
 (14, '2nd'), (31, '3rd'), (22, '2nd'), (17, '1st'),
 (8, 'Crew'), (9, '1st'), (32, 'Crew'), (15, '3rd'),
 (6, '2nd'), (12, 'Crew'), (13, '1st'), (25, '1st')]
>>> idx1 = [1, 2, 3]
>>> idx2 = [0, 9, 8, 4]
>>> pair_unique(idx1, idx2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: All argments should have the same length.
```

unipy.tools.**df_pair_unique**(*data_frame*, *col_list*, *to_frame=False*)

Get unique pairsets in pandas.DataFrame.

This function gets an unique pair-sets of given columns.

Parameters

- **data_frame** (*pandas.DataFrame*) – DataFrame to get unique-pairs.
- **col_list** (*pandas.Index, list, tuple*) – Column names of given DataFrame.
- **to_frame** (*Boolean (default: False)*) – Choose output type. If True, It returns pandas.DataFrame as an output. If False, It returns a list of tuples.

Returns

- *list* – If to_frame=False, A list of tuples is returned. Each tuple is an unique pair of values.
- *pandas.DataFrame* – If to_frame=True, pandas.DataFrame is returned. Each row is an unique pair of values.

See also:

`pandas.DataFrame.itertuples()`

Examples

```
>>> from unipy.tools.data_handler import df_pair_unique
>>> data = dm.load('titanic')
Dataset : titanic
>>> data.head()
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
>>> df_pair_unique(data, ['Class', 'Sex'])
[('3rd', 'Male'), ('2nd', 'Male'), ('2nd', 'Female'), ('1st', 'Female'),
 ('Crew', 'Male'), ('1st', 'Male'), ('Crew', 'Female'), ('3rd', 'Female')]
>>> df_pair_unique(data, ['Class', 'Sex'], to_frame=True)
  Class     Sex
0   3rd    Male
1   2nd    Male
2   2nd  Female
3   1st  Female
4  Crew    Male
5   1st    Male
6  Crew  Female
7   3rd  Female
```

unipy.tools.**map_to_tuple**(*iterable*)
Only for some specific reason.

unipy.tools.**map_to_list**(*iterable*)
Only for some specific reason.

unipy.tools.**merge_csv**(*file_path*, *pattern='*.csv'*, *sep=','*, *if_save=True*, *save_name=None*, *low_memory=True*)
Merge seperated csv type datasets into one dataset. Summary

This function get separated data files together. When merged, the file is sorted by its name in ascending order.

Parameters

- **file_path** (*str*) – A directory path of source files.
- **pattern** (*str*) – A File extension with conditional naming. (default: '*.csv')
- **sep** (*int*) – A symbol seperating data columns.
- **if_save** (*Boolean (Optional, default: True)*) – False if you don't want to save the result.
- **save_name** (*str*) – A filename to save the result. It should be given if if_save=True. If inappropriate name is given, the first name of file list is used.
- **low_memory** (*Boolean (Optional, default: True)*) – It is used for pandas.read_csv() option only.

Returns A concatenated DataFrame.

Return type pandas.DataFrame

Examples

```
>>> from unipy.tools.data_handler import merge_csv
>>> data = dm.load('titanic')
Dataset : titanic
>>> data.head(9)
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
5   2nd  Female  Child       No     0
6   3rd  Female  Child       No    17
7  Crew  Female  Child       No     0
8   1st    Male  Adult       No   118
>>> data.iloc[:2, :].to_csv('tmp1.csv', header=True, index=False)
>>> data.iloc[2:4, :].to_csv('tmp2.csv', header=True, index=False)
>>> data.iloc[4:9, :].to_csv('tmp3.csv', header=True, index=False)
>>> merged = merge_csv('./')
>>> merged
  Class     Sex    Age Survived  Freq
0   1st    Male  Child       No     0
1   2nd    Male  Child       No     0
2   3rd    Male  Child       No    35
3  Crew    Male  Child       No     0
4   1st  Female  Child       No     0
5   2nd  Female  Child       No     0
6   3rd  Female  Child       No    17
7  Crew  Female  Child       No     0
8   1st    Male  Adult       No   118
```

unipy.tools.**nancumsum**(*iterable*)

A cumulative sum function.

A cumulative sum function.

Parameters **iterable** (*Iterable*) – Iterables to calculate cumulative sum.

Yields *int* – A cumulative summed value.

See also:

numpy.isnan()

Examples

```
>>> from unipy.tools.data_handler import nancumsum
>>> tmp = [1, 2, 4]
>>> nancumsum(tmp)
<generator object nancumsum at 0x1084553b8>
>>> list(nancumsum(tmp))
[1, 3, 7]
```

unipy.tools.**depth**(*iterable*)

Get dimension depth.

Get a dimension depth number of a nested iterable.

Parameters **iterable** (*iterable*) – An Iterable to get a dimension depth number.

Returns A dimension depth number.

Return type int

See also:

`collections.Iterable()`

Examples

```
>>> from unipy.tools.data_handler import depth
>>> tmp = [(1, 3), (4, 6), (7, 9), (10, 12)]
>>> depth(tmp)
2
>>> tmp3d = [[np.arange(i) + i for i in range(2, j)]
...          for j in range(5, 10)]
>>> depth(tmp3d)
3
>>> # It can handle dict type (considering values only).
>>> tmp3d_dict = [{'key' + str(i): np.arange(i) + i for i in range(2, j)}
...               for j in range(5, 10)]
>>> depth(tmp3d_dict)
3
```

unipy.tools.**zero_padder_2d**(*arr*, *max_len=None*, *method='backward'*)

Zero-padding for fixed-length inputs(2D).

Zero-padding Function with nested sequence. Each elements of a given sequence is padded fixed-length.

Parameters

- **arr** (*Iterable*) – A nested sequence containing 1-Dimensional numpy.ndarray.
- **max_len** (*int (default: None)*) – A required fixed-length of each sequences. If None, It calculates the max length of elements as max_len.
- **method** (*{'forward', 'backward'} (default: 'backward')*) – where to pad.

Returns A list containing 3-Dimensional numpy.ndarray with fixed-length 2D.

Return type list

See also:

`unipy.depth()`, `numpy.pad()`, `numpy.stack()`

Examples

```
>>> from unipy.tools.data_handler import zero_padder_2d
>>> tmp = [np.arange(i) + i for i in range(2, 5)]
>>> tmp
[array([2, 3]), array([3, 4, 5]), array([4, 5, 6, 7])]
>>> zero_padder_2d(tmp)
array([[2, 3, 0, 0],
       [3, 4, 5, 0],
       [4, 5, 6, 7]])
```

(continues on next page)

(continued from previous page)

```
>>> zero_padder_2d(tmp, max_len=6)
array([[2, 3, 0, 0, 0, 0],
       [3, 4, 5, 0, 0, 0],
       [4, 5, 6, 7, 0, 0]])
 >>> zero_padder_2d(tmp, max_len=5, method='forward')
array([[0, 0, 0, 2, 3],
       [0, 0, 3, 4, 5],
       [0, 4, 5, 6, 7]])
```

unipy.tools.**zero_padder_3d**(*arr*, *max_len=None*, *method='backward'*)

Zero-padding for fixed-length inputs(3D).

Zero-padding Function with nested sequence. Each elements of a given sequence is padded fixed-length.

Parameters

- **arr** (*Iterable*) – A nested sequence containing 2-Dimensional numpy.ndarray.
- **max_len** (*int (default: None)*) – A required fixed-length of each sequences. If None, It calculates the max length of elements as max_len.
- **method** (*{'forward', 'backward'} (default: 'backward')*) – where to pad.

Returns A list containing 3-Dimensional numpy.ndarray with fixed-length 2D.

Return type list

Raises **ValueError** – All 3D shape of inner numpy.ndarray is not equal.

See also:

`unipy.depth()`, `numpy.pad()`, `numpy.stack()`

Examples

```
>>> from unipy.tools.data_handler import zero_padder_3d
>>> tmp3d = [np.arange(i * 2).reshape(-1, 2) for i in range(1, 5)]
>>> tmp3d
[array([[0, 1]]),
 array([[0, 1],
        [2, 3]]),
 array([[0, 1],
        [2, 3],
        [4, 5]]),
 array([[0, 1],
        [2, 3],
        [4, 5],
        [6, 7]])]
>>> zero_padder_3d(tmp3d)
array([[[0, 1],
        [0, 0],
        [0, 0],
        [0, 0]],
```

[[0, 1], [2, 3], [0, 0], [0, 0]],

[[0, 1], [2, 3], [4, 5], [0, 0]],

[[0, 1], [2, 3], [4, 5], [6, 7]]])

```
>>> tmp3d_eye
[array([[1.]]),
 array([[1., 0.],
        [0., 1.]]),
 array([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]]),
 array([[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]])]
>>> zero_padder_3d(tmp3d_eye)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 24, in zero_padder_3d
ValueError: 3D shape should be equal.
```

5.2.8 unipy.unipy_test package

Submodules

unipy.unipy_test.test_data_handler module

Created on Thu Jan 5 20:55:26 2017

@author: Young Ju Kim

unipy.unipy_test.test_dataset module

Created on Thu Jan 5 20:55:26 2017

@author: Young Ju Kim

unipy.unipy_test.test_example module

Created on Thu Jan 5 20:55:26 2017

@author: Young Ju Kim

unipy.unipy_test.test_hypothesis module

Created on Sat Jul 22 23:43:37 2017

@author: pydemia

unipy.unipy_test.test_samplecode module

Sample expensive codes for test.

unipy.unipy_test.test_stats module

Created on Thu Jan 5 20:55:26 2017

@author: Young Ju Kim

Module contents

5.2.9 unipy.utils package

Submodules

unipy.utils.api module

Created on Fri Jun 2 13:41:19 2017

@author: Young Ju Kim

unipy.utils.decorator module

Docstring for `decorator`.

Function Decorator

Profiler	
time_profiler	Function running time command-line profiler.
time_logger	Function running time log profiler.
profiler	High level API combining *time_profiler* & *time_logger*.

Commandline printout	
job_wrapper	Command-line line dragging tool.

Code Translation	
Infix	Function to operator translator.
infix	Functional API for `Infix`.

`unipy.utils.decorator.`**`time_profiler`**(*func*)

Print wrapper for time profiling.

This wrapper prints out start, end and elapsed time.

Parameters **`func`** (*`Function`*) – A function to profile.

Returns A wrapped function.

Return type Function

See also:

`functools.wraps` decorator

Examples

```
>>> import unipy as up
>>> @up.time_profiler
... def afunc(i):
...     return len(list(range(i)))
...
>>> res = afunc(58)
(afunc) Start   : 2018-06-20 22:11:35.511374
(afunc) End     : 2018-06-20 22:11:35.511424
(afunc) Elapsed :             0:00:00.000050
>>> res
58
```

`unipy.utils.decorator.`**`time_logger`**(*func*)

Logging wrapper for time profiling.

This wrapper logs start, end and elapsed time.

Parameters **`func`** (*`Function`*) – A function to profile.

Returns A wrapped function.

Return type Function

See also:

`functools.wraps` decorator

Examples

```
>>> import unipy as up
>>> @up.time_logger
... def afunc(i):
...     return len(list(range(i)))
...
>>> res = afunc(58)
(afunc) Start   : 2018-06-20 22:11:35.511374
(afunc) End     : 2018-06-20 22:11:35.511424
(afunc) Elapsed :             0:00:00.000050
>>> res
58
```

`class` `unipy.utils.decorator.`**`profiler`**(*type='logging'*)

Bases: `object`

`unipy.utils.decorator.`**`job_wrapper`**(*func*)

Print wrapper for time profiling.

This wrapper prints out start & end line.

Parameters **`func`** (*`Function`*) – A function to separate print-line job.

Returns A wrapped function.

Return type Function

See also:

`functools.wraps` decorator

Examples

```
>>> import unipy as up
>>> @up.job_wrapper
... def afunc(i):
...     return len(list(range(i)))
...
>>> afunc(458)
----------- [afunc] START -----------
```

———— [afunc] END ————

afunc : 0:00:00.000023

458

class unipy.utils.decorator.**Infix**(*func*)

Bases: `object`

Wrapper for define an operator.

This wrapper translates a function to an operator.

Returns A wrapped function.

Return type Function

See also:

`functools.partial` decorator

Examples

```
>>> @Infix
... def add(x, y):
...     return x + y
...
>>> 5 |add| 6
11
>>> instanceof = Infix(isinstance)
>>> 5 |instanceof| int
True
```

unipy.utils.decorator.**infix**(*func*)

A functional API for Infix decorator.

Returns A wrapped function.

Return type Function

See also:

`unipy.utils.wrapper.infix`

Examples

```
>>> @infix
... def add(x, y):
...     return x + y
...
>>> 5 |add| 6
11
>>> instanceof = infix(isinstance)
>>> 5 |instanceof| int
True
```

unipy.utils.gdrive module

Docstring for `decorator`.

Function Decorator

File Transfer	
gdrive_downloader	File downloader from Google Drive.
gdrive_uploader	File uploader to Google Drive.

unipy.utils.gdrive.**gdrive_downloader**(*gdrive_url_id*, *pattern='*'*, *download_path='./data'*)

Download files in Google Drive.

Download files in Googel Drive to the given path.

Parameters

- **gdrive_url_id** (*str*) – An URL ID of an Google Drive directory which contains files to download. *https://drive.google.com/drive/folders/<google drive URL ID>*.
- **pattern** (*str (default: '*')*) – A pattern of regular expression to filter file in the target directory.
- **download_path** (*str (default: './data')*) – A target directory to download files in given URL ID.

Returns Nothing is returned.

Return type None

See also:

`None()`

Examples

```
>>> import unipy.util.gdrive import gdrive_downloader
>>> gdrive_path_id = '1LA5334-SZdizcFqkl4xO8Hty7w1q0e8h'
>>> up.gdrive_downloader(gdrive_path_id)
```

unipy.utils.gdrive.**gdrive_uploader**(*gdrive_url_id*, *pattern='*'*, *src_dir='./data'*)

Download files in Google Drive.

Download files in Googel Drive to the given path.

Parameters

- **gdrive_url_id** (*str*) – An URL ID of an Google Drive directory to upload files. *https://drive.google.com/drive/folders/<google drive URL ID>*.
- **pattern** (*str (default: '*')*) – A pattern of regular expression to filter file in the target directory.
- **src_dir** (*str (default: './data')*) – A source directory to upload files in given URL ID.

Returns Nothing is returned.

Return type None

See also:

None()

Examples

```
>>> import unipy.util.gdrive import gdrive_uploader
>>> gdrive_path_id = '1LA5334-SZdizcFqkl4xO8Hty7w1q0e8h'
>>> up.gdrive_uploader(gdrive_path_id)
```

unipy.utils.generator module

Docstring for generator.

Versatile Generators

Productivity	
ReusableGenerator	Reusable Generator.
re_generator	Functional API for ReusableGenerator.

Lazy Evaluation	
split_generator	Split data by given size.

Range Generator	
num_fromto_generator	Range number string pairs by given term.
dt_fromto_generator	Range date format string pairs by given term.
tm_fromto_generator	Range datetime format string pairs by given term.
timestamp_generator	Range timestamp string pairs by given term.

class unipy.utils.generator.**ReusableGenerator**(*generator*)

Bases: object

Temporary Interface to re-use generator for convenience.

Once assigned, It can be infinitely consumed **as long as an input generator remains un-exhausted.

_source

A source generator.

Type generator

See also:

generator itertools.tee

Examples

```
>>> from unipy.utils.generator import ReusableGenerator
>>> gen = (i for i in range(10))
>>> gen
<generator object <genexpr> at 0x11120ebf8>
>>> regen = ReusableGenerator(gen)
>>> regen
<unipy.utils.generator.ReusableGenerator object at 0x1061a97f0>
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(gen)  # If the source is used, copied one will be exhausted too.
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(gen)
[]
>>> list(regen)
[]
```

unipy.utils.generator.**re_generator**(*generator*)

A functional API for unipy.ReusableGenerator.

Once assigned, It can be infinitely consumed **as long as an output generator is called at least one time.

Parameters **generator** (*generator*) – An generator to copy. This original generator should not be used anywhere else, until the copied one consumed at least once.

Returns A generator to be used infinitely.

Return type generator

See also:

generator itertools.tee

Examples

```
>>> from unipy.utils.generator import re_generator
>>> gen = (i for i in range(10))
>>> gen
<generator object <genexpr> at 0x11120ebf8>
>>> regen = copy_generator(gen)
>>> regen
<unipy.utils.generator.ReusableGenerator object at 0x1061a97f0>
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(gen)  # Once the copied one is used, the source will be exhausted.
[]
>>> list(gen)
[]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

unipy.utils.generator.**split_generator**(*iterable*, *size*)

unipy.utils.generator.**num_fromto_generator**(*start*, *end*, *term*)

A range function yields pair chunks.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters ***args** (*int*) – end or start, end[, term] It works like range function.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

yield

Examples

```
>>> from unipy.utils.generator import num_fromto_generator
>>>
>>> query = 'BETWEEN {pre} AND {nxt};'
>>>
>>> q_list = [query.format(pre=item[0], nxt=item[1])
...           for item in num_fromto_generator(1, 100, 10)]
>>> print(q_list[0])
BETWEEN 1 AND 10;
>>> print(q_list[1])
BETWEEN 11 AND 20;
```

unipy.utils.generator.**dt_fromto_generator**(*start*, *end*, *day_term*, *tm_format='%Y%m%d'*)

A range function yields datetime formats by pair.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters

- **start** (*str*) – start datetime like 'yyyymmdd'.
- **end** (*str*) – start datetime like 'yyyymmdd'.

- **day_term** (*int*) – term of days.
- **tm_format** (*(default: '%Y%m%d')*) – datetime format string.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

`yield`

Examples

```
>>> from unipy.utils.generator import dt_fromto_generator
>>> dt_list = [item for item in
...            dt_fromto_generator('20170101','20170331', 10)]
>>> dt_list[:3]
[('20170101', '20170110'),
 ('20170111', '20170120'),
 ('20170121', '20170130')]
```

unipy.utils.generator.**tm_fromto_generator**(*start*, *end*, *day_term*, *tm_string=['000000', '235959']*, *tm_format='%Y%m%d'*)

A range function yields datetime formats by pair.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters

- **start** (*str*) – start datetime like 'yyyymmdd'.
- **end** (*str*) – start datetime like 'yyyymmdd'.
- **day_term** (*int*) – term of days.
- **tm_string** (list (default: `['000000', '235959']`)) – time strings to concatenate.
- **tm_format** (*(default: '%Y%m%d')*) – datetime format string.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

`yield`

Examples

```
>>> from unipy.utils.generator import tm_fromto_generator
>>> tm_list = [item for item in
...            tm_fromto_generator('20170101','20170331', 10)]
>>> tm_list[:3]
[('20170101000000', '20170110235959'),
 ('20170111000000', '20170120235959'),
 ('20170121000000', '20170130235959')]
```

unipy.utils.generator.**timestamp_generator**(**args*)

A range function yields pair timestep strings.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters ***args** (*int*) – end or start, end[, term] It works like range function.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

`yield`

Examples

```
>>> from unipy.utils.generator import timestamp_generator
>>> timestamp_generator(1, 10, 2)
<generator object timestamp_generator at 0x10f519678>
>>> list(timestamp_generator(1, 14, 5))
[(1, 5), (6, 10), (11, 15)]
>>> begin, fin, period = 1, 10, 3
>>> list(timestamp_generator(begin, fin, period))
[(1, 3), (4, 6), (7, 9), (10, 12)]
>>> time_sequence = timestamp_generator(begin, fin, period)
>>> time_msg = "{start:2} to {end:2}, {term:2} days."
>>> for time in time_sequence:
... b, f = time
... print(time_msg.format(start=b, end=f, term=period))
...
 1 to  3,  3 days.
 4 to  6,  3 days.
 7 to  9,  3 days.
10 to 12,  3 days.
```

unipy.utils.remote_ipyconnector module

Created on Sun Jun 11 01:42:23 2017

@author: pydemia

unipy.utils.wrapper module

Docstring for `Wrapper`.

High-level Function Wrapper

Operation Wrapper	
multiprocessor	Functional wrapper for multiprocessing.

Interfaces	
uprint	Print option interface within a function.
lprint	stdout the shape of input layer & output layer in DL
aprint	Stdout the *numpy.ndarray* in pretty.

`unipy.utils.wrapper.`**`multiprocessor`**(*func*, *worker=2*, *arg_zip=None*, **args*, ***kwargs*)

Use multiprocessing as a function.

Just for convenience.

Parameters

- **func** (*Function*) – Any function without `lambda`.
- **worker** (*int (default: 2)*) – A number of processes.
- **arg_zip** (*zip (default: None)*) – A `zip` instance.

Returns A list contains results of each processes.

Return type list

See also:

`multiprocessing.pool`

Examples

```
>>> from unipy.utils.wrapper import multiprocessor
>>> alist = [1, 2, 3]
>>> blist = [-1, -2, -3]
>>> def afunc(x, y):
...     return x + y
...
>>> multiprocessor(afunc, arg_zip=zip(alist, blist))
[0, 0, 0]
>>> def bfunc(x):
...     return x + 2
...
>>> multiprocessor(bfunc, arg_zip=zip(alist))
[3, 4, 5]
```

`unipy.utils.wrapper.`**uprint**(**args*, *print_ok=True*, ***kwargs*)

Print option interface.

This function is equal to `print` function but added `print_ok` option. This allows you to control printing in a function.

Parameters

- ***args** (whatever `print` allows.) – It is same as `print` does.
- **print_ok** (*Boolean (default: True)*) – An option whether you want to print something out or not.
- **arg_zip** (*zip (default: None)*) – A `zip` instance.

`unipy.utils.wrapper.`**lprint**(*input_x*, *output*, *name=None*)

Print option interface.

This function is to stdout the shape of input layer & output layer in Deep Learning architecture.

Parameters

- **input_x** (*numpy.ndarray*) – A `numpy.ndarray` object of input source.
- **output** (*numpy.ndarray*) – A `numpy.ndarray` object of output target.
- **name** (*str (default: None)*) – An optional name you want to print out.

`unipy.utils.wrapper.`**aprint**(**arr*, *maxlen=None*, *name_list=None*, *decimals=None*)

Stdout the *numpy.ndarray* in pretty.

It prints the multiple *numpy.ndarray* out "Side by Side."

Parameters

- **arr** (*numpy.ndarray*) – Any arrays you want to print out.
- **maxlen** (*int (default: None)*) – A length for each array to print out. It is automatically calculated in case of *None*.
- **name_list** (*list (default: None)*) – A list contains the names of each arrays. Upper Alphabet is given in case of *None*.
- **decimals** (*int (default: None)*) – A number to a specified number of digits to truncate.

Examples

```
>>> from unipy.utils.wrapper import aprint
>>> arr_x = np.array([
... [.6, .5, .1],
... [.4, .2, .8],
... ])
>>> arr_y = np.array([
... [.4, .6],
... [.7, .3,],
... ])
>>> aprint(arr_x, arr_y)
=========================================
|  A                 |    B             |
|  (2, 3)            |    (2, 2)        |
=========================================
|  [[0.6 0.5 0.1]    |    [[0.4 0.6]    |
|   [0.4 0.2 0.8]]   |     [0.7 0.3]]   |
=========================================
>>> aprint(arr_x, arr_y, name_list=['X', 'Y'])
=========================================
|  X                 |    Y             |
|  (2, 3)            |    (2, 2)        |
=========================================
|  [[0.6 0.5 0.1]    |    [[0.4 0.6]    |
|   [0.4 0.2 0.8]]   |     [0.7 0.3]]   |
=========================================
>>> aprint(arr_x, arr_y, arr_y[:1], name_list=['X', 'Y', 'Y_1'])
============================================================
|  X                 |    Y             |    Y_1           |
|  (2, 3)            |    (2, 2)        |    (1, 2)        |
============================================================
|  [[0.6 0.5 0.1]    |    [[0.4 0.6]    |    [[0.4 0.6]]   |
|   [0.4 0.2 0.8]]   |     [0.7 0.3]]   |                  |
============================================================
```

Module contents

Utility Objects.

This module provides a number of functions and objects for utility.

decorator

- *time_profiler* – Function running time command-line profiler.
- *time_logger* – Function running time log profiler.
- *job_wrapper* – Command-line line dragging tool.
- *Infix* – Function to operator translator.
- *infix* – Functional API for `Infix`.

generator

- *ReusableGenerator* – Reusable Generator.
- *re_generator* – Functional API for `ReusableGenerator`.
- *split_generator* – Split data by given size.
- *num_fromto_generator* – Range number string pairs by given term.
- *dt_fromto_generator* – Range date format string pairs by given term.
- *tm_fromto_generator* – Range datetime format string pairs by given term.
- *timestamp_generator* – Range timestamp string pairs by given term.

wrapper

- *multiprocessor* – Functional wrapper for multiprocessing.
- *uprint* – Print option interface within a function.

gdrive

- *gdrive_downloader* – File downloader from Google Drive.
- *gdrive_uploader* – File uploader to Google Drive.

`unipy.utils.`**`multiprocessor`**(*func*, *worker=2*, *arg_zip=None*, **args*, ***kwargs*)

Use multiprocessing as a function.

Just for convenience.

Parameters

- **`func`** (*`Function`*) – Any function without `lambda`.
- **`worker`** (*`int (default: 2)`*) – A number of processes.
- **`arg_zip`** (*`zip (default: None)`*) – A `zip` instance.

Returns A list contains results of each processes.

Return type list

See also:

`multiprocessing.pool`

Examples

```
>>> from unipy.utils.wrapper import multiprocessor
>>> alist = [1, 2, 3]
>>> blist = [-1, -2, -3]
>>> def afunc(x, y):
...     return x + y
...
>>> multiprocessor(afunc, arg_zip=zip(alist, blist))
[0, 0, 0]
>>> def bfunc(x):
...     return x + 2
...
>>> multiprocessor(bfunc, arg_zip=zip(alist))
[3, 4, 5]
```

`unipy.utils.`**`uprint`**(**args*, *print_ok=True*, ***kwargs*)

Print option interface.

This function is equal to `print` function but added `print_ok` option. This allows you to control printing in a function.

Parameters

- **`*args`** (whatever `print` allows.) – It is same as `print` does.
- **`print_ok`** (*`Boolean (default: True)`*) – An option whether you want to print something out or not.
- **`arg_zip`** (*`zip (default: None)`*) – A `zip` instance.

`unipy.utils.`**`lprint`**(*input_x*, *output*, *name=None*)

Print option interface.

This function is to stdout the shape of input layer & output layer in Deep Learning architecture.

Parameters

- **`input_x`** (*`numpy.ndarray`*) – A `numpy.ndarray` object of input source.
- **`output`** (*`numpy.ndarray`*) – A `numpy.ndarray` object of output target.
- **`name`** (*`str (default: None)`*) – An optional name you want to print out.

`unipy.utils.`**`aprint`**(**arr*, *maxlen=None*, *name_list=None*, *decimals=None*)

Stdout the *numpy.ndarray* in pretty.

It prints the multiple *numpy.ndarray* out "Side by Side."

Parameters

- **`arr`** (*`numpy.ndarray`*) – Any arrays you want to print out.
- **`maxlen`** (*`int (default: None)`*) – A length for each array to print out. It is automatically calculated in case of *None*.
- **`name_list`** (*`list (default: None)`*) – A list contains the names of each arrays. Upper Alphabet is given in case of *None*.

- **decimals** (*int (default: None)*) – A number to a specified number of digits to truncate.

Examples

```
>>> from unipy.utils.wrapper import aprint
>>> arr_x = np.array([
... [.6, .5, .1],
... [.4, .2, .8],
... ])
>>> arr_y = np.array([
... [.4, .6],
... [.7, .3,],
... ])
>>> aprint(arr_x, arr_y)
========================================
|  A                  |    B             |
|  (2, 3)             |    (2, 2)        |
========================================
|  [[0.6 0.5 0.1]     |    [[0.4 0.6]    |
|   [0.4 0.2 0.8]]    |     [0.7 0.3]]   |
========================================
>>> aprint(arr_x, arr_y, name_list=['X', 'Y'])
========================================
|  X                  |    Y             |
|  (2, 3)             |    (2, 2)        |
========================================
|  [[0.6 0.5 0.1]     |    [[0.4 0.6]    |
|   [0.4 0.2 0.8]]    |     [0.7 0.3]]   |
========================================
>>> aprint(arr_x, arr_y, arr_y[:1], name_list=['X', 'Y', 'Y_1'])
============================================================
|  X                  |    Y             |    Y_1           |
|  (2, 3)             |    (2, 2)        |    (1, 2)        |
============================================================
|  [[0.6 0.5 0.1]     |    [[0.4 0.6]    |    [[0.4 0.6]]   |
|   [0.4 0.2 0.8]]    |     [0.7 0.3]]   |                  |
============================================================
```

unipy.utils.**time_profiler**(*func*)

Print wrapper for time profiling.

This wrapper prints out start, end and elapsed time.

Parameters **func** (*Function*) – A function to profile.

Returns A wrapped function.

Return type Function

See also:

functools.wraps decorator

Examples

```
>>> import unipy as up
>>> @up.time_profiler
... def afunc(i):
...     return len(list(range(i)))
...
>>> res = afunc(58)
(afunc) Start   : 2018-06-20 22:11:35.511374
(afunc) End     : 2018-06-20 22:11:35.511424
(afunc) Elapsed :             0:00:00.000050
>>> res
58
```

unipy.utils.**time_logger**(*func*)

Logging wrapper for time profiling.

This wrapper logs start, end and elapsed time.

Parameters **func** (*Function*) – A function to profile.

Returns A wrapped function.

Return type Function

See also:

`functools.wraps` decorator

Examples

```
>>> import unipy as up
>>> @up.time_logger
... def afunc(i):
...     return len(list(range(i)))
...
>>> res = afunc(58)
(afunc) Start   : 2018-06-20 22:11:35.511374
(afunc) End     : 2018-06-20 22:11:35.511424
(afunc) Elapsed :             0:00:00.000050
>>> res
58
```

class unipy.utils.**profiler**(*type='logging'*)

Bases: `object`

unipy.utils.**job_wrapper**(*func*)

Print wrapper for time profiling.

This wrapper prints out start & end line.

Parameters **func** (*Function*) – A function to separate print-line job.

Returns A wrapped function.

Return type Function

See also:

`functools.wraps` decorator

Examples

```
>>> import unipy as up
>>> @up.job_wrapper
... def afunc(i):
...     return len(list(range(i)))
...
>>> afunc(458)
----------- [afunc] START -----------
```

———— [afunc] END ————

afunc : 0:00:00.000023

458

class unipy.utils.**Infix**(*func*)

Bases: object

Wrapper for define an operator.

This wrapper translates a function to an operator.

Returns A wrapped function.

Return type Function

See also:

functools.partial decorator

Examples

```
>>> @Infix
... def add(x, y):
...     return x + y
...
>>> 5 |add| 6
11
>>> instanceof = Infix(isinstance)
>>> 5 |instanceof| int
True
```

unipy.utils.**infix**(*func*)

A functional API for Infix decorator.

Returns A wrapped function.

Return type Function

See also:

unipy.utils.wrapper.infix

Examples

```
>>> @infix
... def add(x, y):
...     return x + y
...
>>> 5 |add| 6
11
>>> instanceof = infix(isinstance)
>>> 5 |instanceof| int
True
```

class unipy.utils.**ReusableGenerator**(*generator*)

Bases: object

Temporary Interface to re-use generator for convenience.

Once assigned, It can be infinitely consumed **as long as an input generator remains un-exhausted.

_source

A source generator.

Type generator

See also:

generator `itertools.tee`

Examples

```
>>> from unipy.utils.generator import ReusableGenerator
>>> gen = (i for i in range(10))
>>> gen
<generator object <genexpr> at 0x11120ebf8>
>>> regen = ReusableGenerator(gen)
>>> regen
<unipy.utils.generator.ReusableGenerator object at 0x1061a97f0>
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(gen)  # If the source is used, copied one will be exhausted too.
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(gen)
[]
>>> list(regen)
[]
```

unipy.utils.**re_generator**(*generator*)

A functional API for unipy.ReusableGenerator.

Once assigned, It can be infinitely consumed **as long as an output generator is called at least one time.

Parameters **generator** (*generator*) – An generator to copy. This original generator should not be used anywhere else, until the copied one consumed at least once.

Returns A generator to be used infinitely.

Return type generator

See also:

generator `itertools.tee`

Examples

```
>>> from unipy.utils.generator import re_generator
>>> gen = (i for i in range(10))
>>> gen
<generator object <genexpr> at 0x11120ebf8>
>>> regen = copy_generator(gen)
>>> regen
<unipy.utils.generator.ReusableGenerator object at 0x1061a97f0>
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(gen)  # Once the copied one is used, the source will be exhausted.
[]
>>> list(gen)
[]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(regen)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

`unipy.utils.`**`split_generator`**(*iterable*, *size*)

`unipy.utils.`**`num_fromto_generator`**(*start*, *end*, *term*)

A range function yields pair chunks.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters **`*args`** (*int*) – end or start, end[, term] It works like range function.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

`yield`

Examples

```
>>> from unipy.utils.generator import num_fromto_generator
>>>
>>> query = 'BETWEEN {pre} AND {nxt};'
>>>
>>> q_list = [query.format(pre=item[0], nxt=item[1])
...           for item in num_fromto_generator(1, 100, 10)]
>>> print(q_list[0])
BETWEEN 1 AND 10;
>>> print(q_list[1])
BETWEEN 11 AND 20;
```

`unipy.utils.`**`dt_fromto_generator`**(*start*, *end*, *day_term*, *tm_format='%Y%m%d'*)

A range function yields datetime formats by pair.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters

- **start** (*str*) – start datetime like 'yyyymmdd'.
- **end** (*str*) – start datetime like 'yyyymmdd'.
- **day_term** (*int*) – term of days.
- **tm_format** (*(default: '%Y%m%d')*) – datetime format string.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

`yield`

Examples

```
>>> from unipy.utils.generator import dt_fromto_generator
>>> dt_list = [item for item in
...            dt_fromto_generator('20170101','20170331', 10)]
>>> dt_list[:3]
[('20170101', '20170110'),
 ('20170111', '20170120'),
 ('20170121', '20170130')]
```

unipy.utils.**tm_fromto_generator**(*start*, *end*, *day_term*, *tm_string=['000000', '235959']*, *tm_format='%Y%m%d'*)

A range function yields datetime formats by pair.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters

- **start** (*str*) – start datetime like 'yyyymmdd'.
- **end** (*str*) – start datetime like 'yyyymmdd'.
- **day_term** (*int*) – term of days.
- **tm_string** (list (default: `['000000', '235959']`)) – time strings to concatenate.
- **tm_format** (*(default: '%Y%m%d')*) – datetime format string.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

`yield`

Examples

```
>>> from unipy.utils.generator import tm_fromto_generator
>>> tm_list = [item for item in
...            tm_fromto_generator('20170101','20170331', 10)]
>>> tm_list[:3]
[('20170101000000', '20170110235959'),
 ('20170111000000', '20170120235959'),
 ('20170121000000', '20170130235959')]
```

unipy.utils.**timestamp_generator**(**args*)

A range function yields pair timestep strings.

It had made for time-formatting query. It yields a tuple of (start, start+(term-1)) pair, until start > end.

Parameters ***args** (*int*) – end or start, end[, term] It works like range function.

Yields *tuple* – A tuple of (start, start+(term-1)) pair, until start > end.

See also:

yield

Examples

```
>>> from unipy.utils.generator import timestamp_generator
>>> timestamp_generator(1, 10, 2)
<generator object timestamp_generator at 0x10f519678>
>>> list(timestamp_generator(1, 14, 5))
[(1, 5), (6, 10), (11, 15)]
>>> begin, fin, period = 1, 10, 3
>>> list(timestamp_generator(begin, fin, period))
[(1, 3), (4, 6), (7, 9), (10, 12)]
>>> time_sequence = timestamp_generator(begin, fin, period)
>>> time_msg = "{start:2} to {end:2}, {term:2} days."
>>> for time in time_sequence:
... b, f = time
... print(time_msg.format(start=b, end=f, term=period))
...
 1 to  3,  3 days.
 4 to  6,  3 days.
 7 to  9,  3 days.
10 to 12,  3 days.
```

unipy.utils.**gdrive_downloader**(*gdrive_url_id*, *pattern='*'*, *download_path='./data'*)

Download files in Google Drive.

Download files in Googel Drive to the given path.

Parameters

- **gdrive_url_id** (*str*) – An URL ID of an Google Drive directory which contains files to download. *https://drive.google.com/drive/folders/<google drive URL ID>*.
- **pattern** (*str (default: '*')*) – A pattern of regular expression to filter file in the target directory.
- **download_path** (*str (default: './data')*) – A target directory to download files in given URL ID.

Returns Nothing is returned.

Return type None

See also:

None()

Examples

```
>>> import unipy.util.gdrive import gdrive_downloader
>>> gdrive_path_id = '1LA5334-SZdizcFqkl4xO8Hty7w1q0e8h'
>>> up.gdrive_downloader(gdrive_path_id)
```

unipy.utils.**gdrive_uploader**(*gdrive_url_id*, *pattern='*'*, *src_dir='./data'*)

Download files in Google Drive.

Download files in Googel Drive to the given path.

Parameters

- **gdrive_url_id** (*str*) – An URL ID of an Google Drive directory to upload files. *https://drive.google.com/drive/folders/<google drive URL ID>*.
- **pattern** (*str (default: '*')*) – A pattern of regular expression to filter file in the target directory.
- **src_dir** (*str (default: './data')*) – A source directory to upload files in given URL ID.

Returns Nothing is returned.

Return type None

See also:

None()

Examples

```
>>> import unipy.util.gdrive import gdrive_uploader
>>> gdrive_path_id = '1LA5334-SZdizcFqkl4xO8Hty7w1q0e8h'
>>> up.gdrive_uploader(gdrive_path_id)
```

PYTHON MODULE INDEX

U

unipy.core, 34
unipy.core.api, 34
unipy.dataset, 36
unipy.dataset.api, 34
unipy.image, 38
unipy.image.api, 38
unipy.image.houghmatrix, 38
unipy.math, 39
unipy.math.api, 38
unipy.math.geometry, 38
unipy.plots, 42
unipy.plots.api, 39
unipy.plots.boxplot, 39
unipy.stats, 48
unipy.stats.api, 45
unipy.stats.feature_selection, 45
unipy.stats.formula, 47
unipy.stats.hypo_test, 47
unipy.stats.metrics, 48
unipy.tools, 60
unipy.tools.api, 52
unipy.tools.data_handler, 52
unipy.unipy_test, 69
unipy.unipy_test.test_data_handler, 68
unipy.unipy_test.test_dataset, 68
unipy.unipy_test.test_example, 68
unipy.unipy_test.test_hypothesis, 68
unipy.unipy_test.test_samplecode, 69
unipy.unipy_test.test_stats, 69
unipy.utils, 80
unipy.utils.api, 69
unipy.utils.decorator, 69
unipy.utils.gdrive, 72
unipy.utils.generator, 73
unipy.utils.remote_ipyconnector, 77
unipy.utils.wrapper, 77

INDEX

Symbols

_source (*unipy.ReusableGenerator attribute*), 30
_source (*unipy.utils.ReusableGenerator attribute*), 85
_source (*unipy.utils.generator.ReusableGenerator attribute*), 74

A

angle (*unipy.Ellipse attribute*), 12
angle (*unipy.math.Ellipse attribute*), 39
angle (*unipy.math.geometry.Ellipse attribute*), 39
anova_test() (*in module unipy*), 15
anova_test() (*in module unipy.stats*), 49
anova_test() (*in module unipy.stats.hypo_test*), 47
anova_test_formula() (*in module unipy*), 15
anova_test_formula() (*in module unipy.stats*), 49
anova_test_formula() (*in module unipy.stats.hypo_test*), 47
aprint() (*in module unipy*), 26
aprint() (*in module unipy.utils*), 81
aprint() (*in module unipy.utils.wrapper*), 78
average_absolute_deviation() (*in module unipy*), 15
average_absolute_deviation() (*in module unipy.stats*), 49
average_absolute_deviation() (*in module unipy.stats.metrics*), 48

C

calculate_interaction() (*in module unipy*), 15
calculate_interaction() (*in module unipy.stats*), 49
calculate_interaction() (*in module unipy.stats.metrics*), 48
center (*unipy.Ellipse attribute*), 12
center (*unipy.math.Ellipse attribute*), 39
center (*unipy.math.geometry.Ellipse attribute*), 38
chisq_test() (*in module unipy*), 15
chisq_test() (*in module unipy.stats*), 50
chisq_test() (*in module unipy.stats.hypo_test*), 47
coordinates() (*unipy.Ellipse method*), 12
coordinates() (*unipy.math.Ellipse method*), 39
coordinates() (*unipy.math.geometry.Ellipse method*), 39

D

depth() (*in module unipy*), 23
depth() (*in module unipy.tools*), 65
depth() (*in module unipy.tools.data_handler*), 57
deviation() (*in module unipy*), 14
deviation() (*in module unipy.stats*), 49
deviation() (*in module unipy.stats.metrics*), 48
df_pair_unique() (*in module unipy*), 20
df_pair_unique() (*in module unipy.tools*), 63
df_pair_unique() (*in module unipy.tools.data_handler*), 55
diameter (*unipy.Ellipse attribute*), 12
diameter (*unipy.math.Ellipse attribute*), 39
diameter (*unipy.math.geometry.Ellipse attribute*), 38
dt_fromto_generator() (*in module unipy*), 31
dt_fromto_generator() (*in module unipy.utils*), 86
dt_fromto_generator() (*in module unipy.utils.generator*), 75

E

Ellipse (*class in unipy*), 12
Ellipse (*class in unipy.math*), 39
Ellipse (*class in unipy.math.geometry*), 38
even_chunk() (*in module unipy*), 18
even_chunk() (*in module unipy.tools*), 61
even_chunk() (*in module unipy.tools.data_handler*), 53
exc() (*in module unipy*), 17
exc() (*in module unipy.tools*), 60
exc() (*in module unipy.tools.data_handler*), 52

F

f_test() (*in module unipy*), 15
f_test() (*in module unipy.stats*), 49
f_test() (*in module unipy.stats.hypo_test*), 47
f_test_formula() (*in module unipy*), 15
f_test_formula() (*in module unipy.stats*), 49

f_test_formula() (*in module unipy.stats.hypo_test*), 47
feature_selection_vif() (*in module unipy*), 17
feature_selection_vif() (*in module unipy.stats*), 51
feature_selection_vif() (*in module unipy.stats.feature_selection*), 46
fisher_test() (*in module unipy*), 15
fisher_test() (*in module unipy.stats*), 50
fisher_test() (*in module unipy.stats.hypo_test*), 47
from_formula() (*in module unipy*), 17
from_formula() (*in module unipy.stats*), 52
from_formula() (*in module unipy.stats.formula*), 47

G

gdrive_downloader() (*in module unipy*), 33
gdrive_downloader() (*in module unipy.utils*), 88
gdrive_downloader() (*in module unipy.utils.gdrive*), 72
gdrive_uploader() (*in module unipy*), 34
gdrive_uploader() (*in module unipy.utils*), 89
gdrive_uploader() (*in module unipy.utils.gdrive*), 73

H

hough_transform() (*in module unipy*), 14
hough_transform() (*in module unipy.image*), 38
hough_transform() (*in module unipy.image.houghmatrix*), 38

I

Infix (*class in unipy*), 29
Infix (*class in unipy.utils*), 84
Infix (*class in unipy.utils.decorator*), 71
infix() (*in module unipy*), 29
infix() (*in module unipy.utils*), 84
infix() (*in module unipy.utils.decorator*), 71
init() (*in module unipy.dataset*), 36
init() (*in module unipy.dataset.api*), 34

J

job_wrapper() (*in module unipy*), 28
job_wrapper() (*in module unipy.utils*), 83
job_wrapper() (*in module unipy.utils.decorator*), 70

L

lasso_rank() (*in module unipy*), 15
lasso_rank() (*in module unipy.stats*), 50
lasso_rank() (*in module unipy.stats.feature_selection*), 45
load() (*in module unipy.dataset*), 37
load() (*in module unipy.dataset.api*), 35
lprint() (*in module unipy*), 26
lprint() (*in module unipy.utils*), 81
lprint() (*in module unipy.utils.wrapper*), 78
ls() (*in module unipy.dataset*), 37
ls() (*in module unipy.dataset.api*), 35

M

map_to_list() (*in module unipy*), 21
map_to_list() (*in module unipy.tools*), 64
map_to_list() (*in module unipy.tools.data_handler*), 56
map_to_tuple() (*in module unipy*), 21
map_to_tuple() (*in module unipy.tools*), 64
map_to_tuple() (*in module unipy.tools.data_handler*), 56
mean_absolute_percentage_error() (*in module unipy*), 15
mean_absolute_percentage_error() (*in module unipy.stats*), 49
mean_absolute_percentage_error() (*in module unipy.stats.metrics*), 48
median_absolute_deviation() (*in module unipy*), 15
median_absolute_deviation() (*in module unipy.stats*), 49
median_absolute_deviation() (*in module unipy.stats.metrics*), 48
merge_csv() (*in module unipy*), 21
merge_csv() (*in module unipy.tools*), 64
merge_csv() (*in module unipy.tools.data_handler*), 56
module
    unipy, 11
    unipy.core, 34
    unipy.core.api, 34
    unipy.dataset, 36
    unipy.dataset.api, 34
    unipy.image, 38
    unipy.image.api, 38
    unipy.image.houghmatrix, 38
    unipy.math, 39
    unipy.math.api, 38
    unipy.math.geometry, 38
    unipy.plots, 42
    unipy.plots.api, 39
    unipy.plots.boxplot, 39
    unipy.stats, 48
    unipy.stats.api, 45
    unipy.stats.feature_selection, 45
    unipy.stats.formula, 47
    unipy.stats.hypo_test, 47
    unipy.stats.metrics, 48
    unipy.tools, 60
    unipy.tools.api, 52
    unipy.tools.data_handler, 52

unipy.unipy_test, 69
unipy.unipy_test.test_data_handler, 68
unipy.unipy_test.test_dataset, 68
unipy.unipy_test.test_example, 68
unipy.unipy_test.test_hypothesis, 68
unipy.unipy_test.test_samplecode, 69
unipy.unipy_test.test_stats, 69
unipy.utils, 80
unipy.utils.api, 69
unipy.utils.decorator, 69
unipy.utils.gdrive, 72
unipy.utils.generator, 73
unipy.utils.remote_ipyconnector, 77
unipy.utils.wrapper, 77
mosaic_plot() (*in module unipy*), 14
mosaic_plot() (*in module unipy.plots*), 44
mosaic_plot() (*in module unipy.plots.boxplot*), 41
multiprocessor() (*in module unipy*), 25
multiprocessor() (*in module unipy.utils*), 80
multiprocessor() (*in module unipy.utils.wrapper*), 77

N

nancumsum() (*in module unipy*), 22
nancumsum() (*in module unipy.tools*), 65
nancumsum() (*in module unipy.tools.data_handler*), 57
num_fromto_generator() (*in module unipy*), 31
num_fromto_generator() (*in module unipy.utils*), 86
num_fromto_generator() (*in module unipy.utils.generator*), 75

P

pair_unique() (*in module unipy*), 19
pair_unique() (*in module unipy.tools*), 62
pair_unique() (*in module unipy.tools.data_handler*), 54
point_boxplot() (*in module unipy*), 12
point_boxplot() (*in module unipy.plots*), 42
point_boxplot() (*in module unipy.plots.boxplot*), 39
point_boxplot_axis() (*in module unipy*), 13
point_boxplot_axis() (*in module unipy.plots*), 43
point_boxplot_axis() (*in module unipy.plots.boxplot*), 40
profiler (*class in unipy*), 28
profiler (*class in unipy.utils*), 83
profiler (*class in unipy.utils.decorator*), 70

R

radius (*unipy.Ellipse attribute*), 12
radius (*unipy.math.Ellipse attribute*), 39
radius (*unipy.math.geometry.Ellipse attribute*), 38
re_generator() (*in module unipy*), 30
re_generator() (*in module unipy.utils*), 85
re_generator() (*in module unipy.utils.generator*), 74
reset() (*in module unipy.dataset*), 36
reset() (*in module unipy.dataset.api*), 35
ReusableGenerator (*class in unipy*), 30
ReusableGenerator (*class in unipy.utils*), 85
ReusableGenerator (*class in unipy.utils.generator*), 74
rgb2gras() (*in module unipy*), 14
rgb2gras() (*in module unipy.image*), 38
rgb2gras() (*in module unipy.image.houghmatrix*), 38

S

split_generator() (*in module unipy*), 31
split_generator() (*in module unipy.utils*), 86
split_generator() (*in module unipy.utils.generator*), 75
splitter() (*in module unipy*), 18
splitter() (*in module unipy.tools*), 61
splitter() (*in module unipy.tools.data_handler*), 53

T

time_logger() (*in module unipy*), 28
time_logger() (*in module unipy.utils*), 83
time_logger() (*in module unipy.utils.decorator*), 70
time_profiler() (*in module unipy*), 27
time_profiler() (*in module unipy.utils*), 82
time_profiler() (*in module unipy.utils.decorator*), 69
timestamp_generator() (*in module unipy*), 32
timestamp_generator() (*in module unipy.utils*), 87
timestamp_generator() (*in module unipy.utils.generator*), 76
tm_fromto_generator() (*in module unipy*), 32
tm_fromto_generator() (*in module unipy.utils*), 87
tm_fromto_generator() (*in module unipy.utils.generator*), 76

U

unipy
module, 11
unipy.core
module, 34
unipy.core.api
module, 34
unipy.dataset
module, 36
unipy.dataset.api
module, 34

unipy.image
    module, 38
unipy.image.api
    module, 38
unipy.image.houghmatrix
    module, 38
unipy.math
    module, 39
unipy.math.api
    module, 38
unipy.math.geometry
    module, 38
unipy.plots
    module, 42
unipy.plots.api
    module, 39
unipy.plots.boxplot
    module, 39
unipy.stats
    module, 48
unipy.stats.api
    module, 45
unipy.stats.feature_selection
    module, 45
unipy.stats.formula
    module, 47
unipy.stats.hypo_test
    module, 47
unipy.stats.metrics
    module, 48
unipy.tools
    module, 60
unipy.tools.api
    module, 52
unipy.tools.data_handler
    module, 52
unipy.unipy_test
    module, 69
unipy.unipy_test.test_data_handler
    module, 68
unipy.unipy_test.test_dataset
    module, 68
unipy.unipy_test.test_example
    module, 68
unipy.unipy_test.test_hypothesis
    module, 68
unipy.unipy_test.test_samplecode
    module, 69
unipy.unipy_test.test_stats
    module, 69
unipy.utils
    module, 80
unipy.utils.api
    module, 69
unipy.utils.decorator
    module, 69
unipy.utils.gdrive
    module, 72
unipy.utils.generator
    module, 73
unipy.utils.remote_ipyconnector
    module, 77
unipy.utils.wrapper
    module, 77
uprint() (*in module unipy*), 26
uprint() (*in module unipy.utils*), 81
uprint() (*in module unipy.utils.wrapper*), 78

V

vif() (*in module unipy*), 15
vif() (*in module unipy.stats*), 49
vif() (*in module unipy.stats.metrics*), 48

Z

zero_padder_2d() (*in module unipy*), 23
zero_padder_2d() (*in module unipy.tools*), 66
zero_padder_2d() (*in module unipy.tools.data_handler*), 58
zero_padder_3d() (*in module unipy*), 24
zero_padder_3d() (*in module unipy.tools*), 67
zero_padder_3d() (*in module unipy.tools.data_handler*), 59