graphpkg Documentation

Release unknown

Nishant Baheti

Aug 20, 2022

CONTENTS

Install **graphpkg** using pip.

Note: pip install graphpkg

CHAPTER

ONE

CONTENTS

1.1 graphpkg

A package to plot graphs on a dashboard using matplotlib

1.1.1 Description

This package initially started with the idea of plotting live trend graph using matplotlib as base.

visit https://graphpkg.readthedocs.io/ for detailed documentation

1.2 graphpkg package

1.2.1 Subpackages

graphpkg.live package

Module contents

class graphpkg.live.**LiveDashboard**(*config: dict*)

Bases: object

Live Dashboard plot

Parameters **config** (*dict*) – Configuration Dictionary

Example

```
>>> conf = {
>>>     "dashboard": "DASHBOARD1",
>>>     "plots": {
>>>         "trend": [
>>>             {
>>>                 "func_for_data": func1,
>>>                 "fig_spec": (4, 3, (1, 2)),
>>>                 "interval": 500,
>>>                 "title": "trend plot1"
```

(continues on next page)

(continued from previous page)

```
>>>             },
>>>             {
>>>                 "func_for_data": func1,
>>>                 "fig_spec": (4, 3, (4, 5)),
>>>                 "interval": 500,
>>>                 "title": "trend plot2"
>>>             },
>>>             {
>>>                 "func_for_data": func1,
>>>                 "fig_spec": (4, 3, (7, 8)),
>>>                 "interval": 500,
>>>                 "title": "trend plot3"
>>>             },
>>>             {
>>>                 "func_for_data": func1,
>>>                 "fig_spec": (4, 3, (10,11)),
>>>                 "interval": 500,
>>>                 "title": "trend plot4"
>>>             }
>>>         ],
>>>         "distribution": [{
>>>             "fig_spec": (4, 3, (3,6)),
>>>             "func_for_data": func4,
>>>             "interval": 1000,
>>>             "title": "distribution plot",
>>>             "window": 500
>>>         }],
>>>         "scatter": [
>>>             {
>>>                 "fig_spec": (4, 3, (9,12)),
>>>                 "func_for_data": func3,
>>>                 "func_args": (1000,),
>>>                 "interval": 1000,
>>>                 "title": "other other scatter plot",
>>>                 "window": 500
>>>             }
>>>         ]
>>>     }
>>> }
>>> dash = LiveDashboard(config=conf)
>>> dash.start()
>>> matplotlib.pyplot.show()
```

property dash_config: List[Dict]
Dash board configuration

Returns list of plots in a dictionary

Return type List[Dict]

display()
display information

start()
Start the dashboard

class graphpkg.live.**LiveDistribution**(*interval: int*, *func_for_data: callable*, *func_args: Optional[Iterable] = None*, *fig: Optional[matplotlib.pyplot.figure] = None*, *fig_spec: tuple = (1, 1, 1)*, *xlabel: str = 'x-axis'*, *ylabel: str = 'y-axis'*, *label: str = 'Current Data'*, *title: str = 'Live Scatter'*, *window: int = 2000*)

Bases: graphpkg.live._graph.Graph

Live Distribution Graph Module

Parameters

- **func_for_data** (*callable*) – Function to return x and y data point.x is a single value and y can be a list of max length 3 or a single value. both of them shouldn't be None.
- **Example** –

```
>>> def get_new_data():
>>>     return 10,10
```

```
>>> def get_new_data():
>>>     ## first param for x axis and second can be an array of␣
↪values
>>>     return 10,[10,11]
```

```
>>> def func1(*args):
>>>    return random.randrange(1, args[0]),random.randrange(1,␣
↪args[0])
```

- **func_args** (*Iterable, optional*) – data function arguments. Defaults to None.
- **fig** (*matplotlib.pyplot.figure, optional*) – .Matplotlib figure. Defaults to None.
- **fig_spec** (*tuple, optional*) – [description]. Matplotlib figure specification. Defaults to (1,1,1).
- **interval** (*int*) – Interval to refresh data in milliseconds.
- **xlabel** (*str, optional*) – Label for x-axis. Defaults to "x-axis".
- **ylabel** (*str, optional*) – Label for y-axis. Defaults to "y-axis".
- **label** (*str, optional*) – Label for plot line. Defaults to "Current Data".
- **title** (*str, optional*) – Title of Scatter chart. Defaults to "Live Scatter".
- **window** (*int, optional*) – Data point window. Defaults to 500.

Examples

```
>>> def func1():
        return None, random.randrange(1,100)
>>> g1 = LiveDistribution(func_for_data=func1,interval=1000,title="plot 1 for range␣
↪1-100")
>>> g1.start()
>>> plt.show()
```

display() → None
display information

start() → None
Initiate the scatter chart

property xs: List[graphpkg.live._graph.A]
x-axis data list

Returns x-axis list

Return type List[A]

property ys: List[graphpkg.live._graph.A]
y-axis data list

Returns y-axis list of lists

Return type List[A]

class graphpkg.live.**LiveScatter**(*interval: int*, *func_for_data: callable*, *func_args: Optional[Iterable] = None*, *fig: Optional[matplotlib.pyplot.figure] = None*, *fig_spec: tuple = (1, 1, 1)*, *xlabel: str = 'x-axis'*, *ylabel: str = 'y-axis'*, *label: str = 'Current Data'*, *title: str = 'Live Scatter'*, *window: int = 500*)

Bases: graphpkg.live._graph.Graph

Live Scatter Graph Module

Parameters

- **func_for_data** (*callable*) – Function to return x and y data point.x is a single value and y can be a list of max length 3 or a single value. both of them shouldn't be None.
- **Example** –

```
>>> def get_new_data():
>>>     return 10,10
```

```
>>> def get_new_data():
>>>     ## first param for x axis and second can be an array of␣
↪values
>>>     return 10,[10,11]
```

```
>>> def func1(*args):
>>>    return random.randrange(1, args[0]),random.randrange(1,␣
↪args[0])
```

- **func_args** (*Iterable, optional*) – data function arguments. Defaults to None.
- **fig** (*matplotlib.pyplot.figure, optional*) – .Matplotlib figure. Defaults to None.
- **fig_spec** (*tuple, optional*) – [description]. Matplotlib figure specification. Defaults to (1,1,1).
- **interval** (*int*) – Interval to refresh data in milliseconds.
- **xlabel** (*str, optional*) – Label for x-axis. Defaults to "x-axis".
- **ylabel** (*str, optional*) – Label for y-axis. Defaults to "y-axis".
- **label** (*str, optional*) – Label for plot line. Defaults to "Current Data".
- **title** (*str, optional*) – Title of Scatter chart. Defaults to "Live Scatter".
- **window** (*int, optional*) – Data point window. Defaults to 500.

Examples

```
>>> scatter = LiveScatter(func_for_data = get_new_data, interval=1000)
>>> scatter.start()
>>> matplotlib.pyplot.show()
```

```
>>> scatter = LiveScatter(func_for_data = get_new_data, interval=1000, window=30)
>>> scatter.start()
>>> matplotlib.pyplot.show()
```

```
>>> scatter = LiveScatter(func_for_data = get_new_data,func_args=(1000,),␣
↪interval=1000, title="my test data")
>>> scatter.start()
>>> matplotlib.pyplot.show()
```

display() → None
display information

start() → None
Initiate the scatter chart

property xs: List[graphpkg.live._graph.A]
x-axis data list

Returns x-axis list

Return type List[A]

property ys: List[graphpkg.live._graph.A]
y-axis data list

Returns y-axis list of lists

Return type List[A]

class graphpkg.live.**LiveTrend**(*interval: int*, *func_for_data: callable*, *func_args: Optional[Iterable] = None*, *fig: Optional[matplotlib.pyplot.figure] = None*, *fig_spec: tuple = (1, 2, (1, 2))*, *xlabel: str = 'x-axis'*, *ylabel: str = 'y-axis'*, *label: str = 'Current Data'*, *title: str = 'Live Trend'*, *window: int = 50*)

Bases: graphpkg.live._graph.Graph

Live Trend Graph Module

Parameters

- **func_for_data** (*callable*) – Function to return x and y data points.x is a single value and y can be a list of max length 3 or a single value.
- **Example** –

```
>>> def get_new_data():
>>>     return datetime.datetime.now(),10
```

```
>>> def get_new_data():
>>>     return None,10
```

```
>>> def get_new_data():
>>>     ## first param for x axis and second can be an array of␣
↪values
>>>     return datetime.datetime.now(),[10,11]
```

```
>>> def func1(*args):
>>>    return datetime.datetime.now(),random.randrange(1, args[0])
```

- **func_args** (*Iterable, optional*) – data function arguments. Defaults to None.
- **fig** (*matplotlib.pyplot.figure, optional*) – .Matplotlib figure. Defaults to None.
- **fig_spec** (*tuple, optional*) – [description]. Matplotlib figure specification. Defaults to (1,2,(1,2)).
- **interval** (*int*) – Interval to refresh data in milliseconds.
- **xlabel** (*str, optional*) – Label for x-axis. Defaults to "x-axis".
- **ylabel** (*str, optional*) – Label for y-axis. Defaults to "y-axis".
- **label** (*str, optional*) – Label for plot line. Defaults to "Current Data".
- **title** (*str, optional*) – Title of trend chart. Defaults to "Live Trend".
- **window** (*int, optional*) – Data point window. Defaults to 50.

Examples

```
>>> trend = LiveTrend(func_for_data = get_new_data, interval=1000)
>>> trend.start()
>>> matplotlib.pyplot.show()
```

```
>>> trend = LiveTrend(func_for_data = get_new_data, interval=1000, window=30)
>>> trend.start()
>>> matplotlib.pyplot.show()
```

```
>>> trend = LiveTrend(func_for_data = get_new_data, interval=1000, title="my test␣
↪data")
>>> trend.start()
>>> matplotlib.pyplot.show()
```

display() → None
display information

start() → None
Initiate the trend chart

property xs: List[graphpkg.live._graph.A]
x-axis data list

Returns x-axis list

Return type List[A]

property ys: List[graphpkg.live._graph.A]
y-axis data list

Returns y-axis list of lists

Return type List[A]

graphpkg.static package

Module contents

graphpkg.static.**grid_classification_boundary**(*models_list: list*, *data: Optional[numpy.ndarray] = None*, *size: int = 4*, *n_plot_cols: int = 3*, *figsize: tuple = (5, 5)*, *canvas_details: int = 50*, *canvas_opacity: float = 0.4*, *canvas_palette='coolwarm'*) → None

Plot multiple plots of clasification boundaries for mulitple ml models.

Only models are allowed with 1D prediction.

Parameters

- **models_list** (*list*) – Models list of dictionary.
- **data** (*np.ndarray, optional*) – source data. restricted to 2 features and 1 target, in total 3 columns. Defaults to None.
- **size** (*int, optional*) – Size of canvas. Defaults to 4.
- **n_plot_cols** (*int, optional*) – number of plot columns. Defaults to 3.
- **figsize** (*tuple, optional*) – figure size. Defaults to (5, 5).
- **canvas_details** (*int, optional*) – detailing in canvas. Defaults to 50.
- **canvas_opacity** (*float, optional*) – Canvas transparency parameter. Defaults to 0.4.
- **canvas_palette** (*str, optional*) – palette from matplotlib. Defaults to coolwarm.

Raises ValueError – Only 3 dimensional data, 2 features, 1 target is allowed.

Examples

```
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.tree import DecisionTreeClassifier
>>> from sklearn.datasets import make_classification
>>> import matplotlib.pyplot as plt
>>> X, y = make_classification(n_samples=500, n_features=2, random_state=25,
>>>                            n_informative=1, n_classes=2, n_clusters_per_
↪class=1,
>>>                            n_repeated=0, n_redundant=0)
>>> lr_model = LogisticRegression().fit(X, y)
>>> dt_model = DecisionTreeClassifier().fit(X, y)
>>> models_list = [{
>>>     "name": "Logistic Regression Classifier",
>>>     "function": lr_model.predict
>>> },{
>>>     "name": "Decision Tree Classifier",
>>>     "function": dt_model.predict
>>> }]
>>> grid_classification_boundary(models_list=models_list, data=np.hstack((X, y.
↪reshape(-1, 1))),
```

(continues on next page)

(continued from previous page)

```
>>>                                 figsize=(7,5), canvas_details=100)
>>> plt.show()
```

graphpkg.static.**multi_distplots**(*df: pandas.core.frame.DataFrame*, *n_cols: int = 4*, *bins: int = 20*, *kde: bool = True*, *class_col: Optional[str] = None*, *legend: bool = True*, *legend_loc: str = 'best'*, *figsize: Optional[tuple] = None*, *palette: str = 'dark'*, *grid_flag: bool = True*, *xticks_rotation: int = 60*) → None

Mulitple Distribution Plots using pandas dataframe.

Seaborn's histplot is used for distribution with additional functionality to have multiple distributions in one grid.

Parameters

- **df** (*pd.DataFrame*) – Input dataframe.
- **n_cols** (*int, optional*) – Number of columns in the grid. Defaults to 4.
- **bins** (*int, optional*) – number of bins in distribution. Defaults to 20.
- **kde** (*bool, optional*) – kde estimation line & plot. Defaults to True.
- **class_col** (*str, optional*) – class column name for distribution separation and legend. Defaults to None.
- **legend** (*bool, optional*) – put legend or not. Defaults to True.
- **legend_loc** (*str, optional*) – where to put legend, takes inputs similar to matplotlib.pyplot. Defaults to 'best'.
- **figsize** (*tuple, optional*) – figure size, similar to matplotlib.pyplot. Defaults to None.
- **palette** (*str, optional*) – color palette, property from seaborn. Defaults to 'dark'.
- **grid_flag** (*bool, optional*) – put grid or not. Defaults to True.
- **xticks_rotation** (*int, optional*) – xticks rotation angle. Defaults to 60.

Examples

```
>>> from sklearn.datasets import fetch_california_housing
>>> import pandas as pd
>>> import numpy as np
>>> dataset = fetch_california_housing()
>>> df = pd.DataFrame(dataset.data, columns=dataset.feature_names)
>>> df['target'] = dataset.target
>>> multi_distplots(df, n_cols=2)
>>> plt.show()
```

graphpkg.static.**plot_boxed_timeseries**(*df: pandas.core.frame.DataFrame*, *ts_col: str*, *data_col: str*, *box: Optional[str] = 'MONTH'*, *figsize: Optional[tuple] = None*)

Plot timeseries data integrated with boxplot to see window based data variation.

Parameters

- **df** (*pd.DataFrame*) – pandas dataframe.
- **ts_col** (*str*) – timeseries column name.
- **data_col** (*str*) – data column name.
- **box** (*Optional[str], optional*) – time box. Defaults to 'MONTH'.

- **figsize** (*Optional[tuple], optional*) – figure size. Defaults to None.

Returns Matplotlib figure and axes.

Return type Figure, Axes

Examples

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import pandas as pd
>>> from graphpkg.static import plot_boxed_timeseries
>>> size = 1000
>>> df = pd.DataFrame({
>>>     "data": np.random.normal(size=(size,)) * 100,
>>>     "timestamps": pd.date_range(start='1/1/2018', periods=size, freq='MIN')
>>> })
>>> fig, ax = plot_boxed_timeseries(df, data_col='data', ts_col='timestamps', box=
↪'hour', figsize=(10, 5))
>>> plt.tight_layout()
>>> plt.show()
```

graphpkg.static.**plot_classification_boundary**(*func: Callable*, *data: Optional[numpy.ndarray] = None*, *size: int = 4*, *n_plot_cols: int = 1*, *figsize: tuple = (5, 5)*, *canvas_details: int = 50*, *canvas_opacity: float = 0.5*, *canvas_palette: str = 'coolwarm'*)

Plot classification model's decision boundary.

Parameters

- **func** (*function*) – Prediction function of ML model that.
- **data** (*np.ndarray, optional*) – source data. restricted to 2 features and 1 target, in total 3 columns. Defaults to None.
- **size** (*int, optional*) – size of canvas. Defaults to 4.
- **n_plot_cols** (*int, optional*) – number of columns for number of plots. Defaults to 1.
- **figsize** (*tuple, optional*) – matplotlib figure size. Defaults to (5, 5).
- **canvas_details** (*int, optional*) – how detailed the boundary should be. Defaults to 50.
- **canvas_opacity** (*float, optional*) – Canvas transparency parameter. Defaults to 0.3.
- **canvas_palette** (*str, optional*) – palette of canvas. Defaults to 'coolwarm'.

Raises ValueError – If the input data's shape is not (k,3), k=number of rows.

Examples

```
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.datasets import make_classification
>>> import matplotlib.pyplot as plt
>>> X, y = make_classification(n_samples=500, n_features=2, random_state=25,
>>>                            n_informative=1, n_classes=2, n_clusters_per_
↪class=1,
>>>                            n_repeated=0, n_redundant=0)
>>> model = LogisticRegression().fit(X, y)
>>> plot_classification_boundary(func=model.predict, data=np.hstack((X,y.reshape(-1,
↪1))),canvas_details=100)
>>> plt.show()
```

graphpkg.static.**plot_distribution**(*x: numpy.ndarray*, *kde: Optional[bool] = True*, *indicate_data: Optional[Union[list, numpy.ndarray]] = None*, *figsize: Optional[tuple] = None*) → None

Plot distribution with additional informations.

distribution and box plot from matplotlib and seaborn.

Parameters

- **x** (*np.ndarray*) – input 1D array.
- **kde** (*Optional[bool], optional*) – kde parameter from seaborn. Defaults to True.
- **indicate_data** (*Optional[Union[list, np.ndarray]], optional*) – data points to observe/indicate in plot. Defaults to None.
- **figsize** (*Optional[tuple], optional*) – figure size from matplotlib. Defaults to None.

Raises **AssertionError** – only 1d arrays are allowed for input.

Examples

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from graphpkg.static import plot_distribution
>>> x = np.random.normal(size=(200,))
>>> plot_distribution(x, indicate_data=[0.6])
>>> plt.show()
```

1.2.2 Module contents

graphpkg

1.3 License

The MIT License (MIT)

1.4 Contributors

- Nishant Baheti <nishantbaheti.it19@gmail.com>

1.5 Changelog

1.5.1 Version 0.0.5

- Added dashboard
- Added distribution plot

1.5.2 Version 0.0.9

- Restructured more pythonic way

1.5.3 Version 1.0.0

- first stable version

1.5.4 Version 1.0.1

- minor updates in typing
- unit test done

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX