Matplotlib Summary

Matplotlib Tutorials playlist

Ahmed Mohamed Ads

Tuesday 15th April, 2025
17 Shawwal 1446 AH

Matplotlib

matplotlib

Python library used for plotting different graphs.

Basic Imports and Functions

```
import matplotlib
\%matplotlib tk
from matplotlib import pyplot as plt
```

- `import matplotlib`: importing the library to use it.
- `\%matplotlib tk`: if you want the figure in a separate window, not in the notebook line (this is extra information from me :))
- `from matplotlib import pyplot as plt`: importing the pyplot module from the library to use it to plot this module has several functions for plotting.

Plotting Functions

```
plt.plot(x, y)
plt.xlabel('name on x_axis')
plt.ylabel('name on y_axis')
plt.title('title of the plot')
plt.legend()
plt.show()
plt.grid()
plt.tight_layout()
```

- `plt.plot(x, y)`: the plotting function plot and the two arguments, the x on the x-axis and the y on the y-axis.
- `plt.xlabel('name on x_axis')`: function for the label representing the data on the x_axis.
- `plt.ylabel('name on y_axis')`: function for the label representing the data on the y_axis.
- `plt.title('title of the plot')`: function for the title that represents what the plot is for.
- `plt.legend()`: function adds a legend and the legend is a small box explaining the meaning of the different elements plotted lines, bars, etc. matplotlib legend documentation
- `plt.show()`: function responsible for presenting the plot on the screen (displaying on).
- `plt.grid()`: function taking True or False, responsible for making a grid for the plot for better vision.
- `plt.tight_layout()`: function for adjusting spacing or overlapping between subplots.

Style and Saving

```
plt.style.available
plt.style.use()
plt.savefig('plot.png')
```

- `plt.style.available`: an attribute used to see the available style for changing plot style.
- `plt.style.use()`: function used for changing plot style.
- `plt.savefig('plot.png')`: function used for saving the plot as png.

some functions extra explanation

```
plt.plot(x_data, y_data, '^ k - -' , label = 'All Devs')
plt.plot(x_data, y_data, color = 'k', linestyle = '- -', marker = '.
    ')
plt.plot(x_data, y_data, color = '#5a7d9a', linewidth = '3')
```

- `plt.plot(x_data, y_data, '^ k--', label = 'All Devs'`: this is a formatting for the marker, line, and color (fmt = '[marker][line][color]'). (label = 'All Devs') label for the legend (better for avoiding the problem of order).
- `plt.plot(x_data, y_data, color = 'k', linestyle = '--', marker = '.'`: we can use the previous fmt as this (color ='color', linestyle= 'linestyle', marker = 'marker').
- `plt.plot(x_data, y_data, color = '#5a7d9a', linewidth = '3'`: we can also use the hexa representation for the color. (linewidth = 'width') is used for making the line of the plot Bold according to a specific width.

Bar Charts

```
plt.bar(x_data, y_data)
plt.barh()  # Horizontal bars
```

- `plt.bar(x_data, y_data)`: function for plotting a bar chart
- `plt.barh()`: plotting the bars horizontally

Multiple Bar Charts Solution

```
import numpy as np
x_indexes = np.arange(len(age_x))
width = 0.25
plt.bar(x_indexes - width, py_dev_y, width=width)
plt.bar(x_indexes, js_dev_y, width=width)
plt.bar(x_indexes + width, dev_y, width=width)
plt.xticks(ticks=x_indexes, labels=age_x)
```

- `import numpy as np`: importing the numpy library

```
age_x = [25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35]
x_indexes = np.arange(len(age_x))

width = 0.25

py_dev_y = [45372, 48876, 53850, 57287, 63016,
            65998, 70003, 70000, 71496, 75370, 83640]
plt.bar(x_indexes - width, py_dev_y,width = width, label = 'Python')

js_dev_y = [37810, 43515, 46823, 49293, 53437,
            56373, 62375, 66674, 68745, 68746, 74583]
plt.bar(x_indexes, js_dev_y,width = width, label = 'JavaScript')

dev_y = [38496, 42000, 46752, 49320, 53200,
         56000, 62316, 64928, 67317, 68748, 73752]
plt.bar(x_indexes + width, dev_y,width = width, color = '#444444', linestyle = '--', label = 'All Devs')

plt.xlabel('Age')
plt.ylabel('Median Salary (USD)')
plt.title('Median Salary (USD) by Age')

plt.legend()

plt.tight_layout()

#plt.savfig('plot.png')

plt.show()
```

Figure 1: code for more than bar chart

- `x_indexes = np.arange(len(age_x))`: making the range of indexes and to the bars side by side we will subtract the width from one of the bars and leave one as the middle one and the third we will add the width to it to make the bars side by side like this and if more than three we can add double the width, etc...
- `plt.xticks(ticks=x_indexes, labels=age_x)`: Now in the plot, we see that the x-axis doesn't have the ages, and it has the indexes. We can fix that by using (plt.xticks(ticks = x_indexes, labels = age_x)), and parameters may vary according to variables

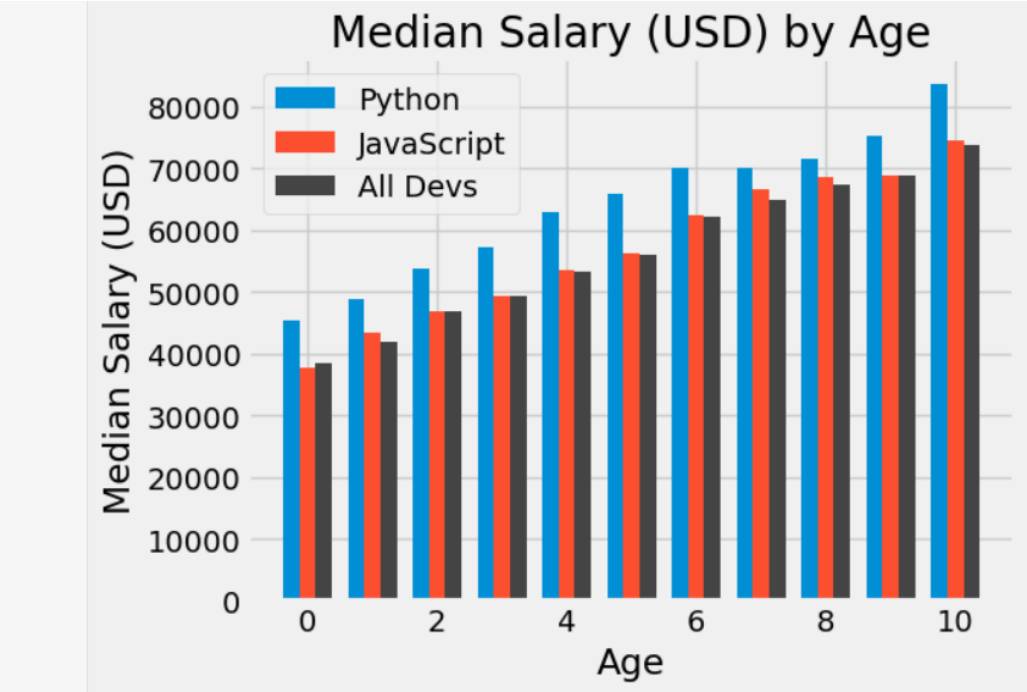

Figure 2: more than bar chart

Pandas Integration

```
import pandas as pd
df = pd.read_csv()
df[].str.split(';').explode().value_counts()
.reverse() or [::-1]  # To reverse data order
```

- `import pandas as pd`: importing pandas library as name pd
- `df = pd.read_csv()`: reading a csv file using pandas
- `df[].str.split(';').explode().value_counts()`: splitting a data separated with (;) and counting it for Series objects, not the entire DataFrame.
- `.reverse()or [::-1]`: We can use these two to reverse the data (.reverse() works on lists only; [::-1] works on lists, strings, arrays) to make the most is up or down.

Pie Charts

```
plt.pie(slices, labels=labels)
plt.pie(slices, wedgeprops={})
plt.pie(slices, explode=[0,0,0,wanted,0], shadow=True,
        startangle=angle, autopct='%1.1f%%')
```

- `plt.pie(slices, labels=labels)`: function for plotting a pie chart. (We can pass labels = labels for the labels of the pie chart).

- `plt.pie(slices, wedgeprops={})`: we can use the `wedgeprops` to control the appearance of some wedges in the pie chart. You can see the documentation here for more details matplotlib wedge documentation

- `plt.pie(slices, explode=[0,0,0,wanted,0], shadow=True, startangle=angle, autopct='\%1.1f\%\%'`: `label` is for pies labels. `wedgeprops` we talked about it. `explode` is used to explode a certain pie. We choose a number (wanted), and we then apply the explode. `shadow` for making a shadow for the pie chart. `startangle` for starting the chart with a certain angle. `autopct= '%1.1f%%'` for adding the percent of each piece of pie on it.

Advanced Plotting

```
plt.stackplot()
plt.fill_between(x_values, y_values, limit)
plt.fill_between(x_values, y_values, limit,
                 where=(y_values > limit),
                 interpolate=True, alpha=0.25)
plt.hist(ages, bins=bins, edgecolor='black', log=True)
plt.axvline()  # Vertical line
plt.scatter(x,y, s=100, c='color', cmap='Greens',
            marker='x', edgecolor='black',
            linewidth=1, alpha=0.75)
```

- `plt.stackplot()`: function for plotting stack plot.

- `plt.fill_between(x_values, y_values, limit)`: function for filling under the line plot matplotlib fill_between documentation.

- `plt.fill_between(x_values, y_values, limit, where=(y_values > limit), interpolate=True, alpha=0.25)`: `where` is used to customize the filling limit above or below a certain value. `alpha = value` and this is how much we can see through the filling. `interpolate` to make sure that the intersections do not get clipped and the regions are filled correctly.

- `plt.hist(ages, bins=bins, edgecolor='black', log=True)`: plotting a histogram. `bins` divides the plot into a certain number of bins according to one or several values. `edgecolor` the edge color of the histogram. `log` is used for making the plot in logarithmic scale.

- `plt.axvline()`: used for drawing a vertical line on the plot for a specific value.

- `plt.scatter(x,y, s=100, c='color', cmap='Greens', marker='x', edgecolor='black', linewidth=1, alpha=0.75)`: `s= value` the size of the marker on the plot. `c = 'colors'` choosing plots colors. `cmap = 'color map'` choosing a certain color map for the plot and visit matplotlib color map documentation for more information. `marker = 'marker shape'` choosing a specific color style and see matplotlib marker documentation for more information. `edgecolor` as we said before, for the color of the edges. `linewidth = value` for the width of the markers in the plot. `alpha = value` as we said before, this is how much we can see through the filling.

Scaling Functions

```
plt.xscale()
plt.yscale()
```

- `plt.xscale()`: making a specific scale for example log scale and see matplotlib scale documentation for more information.
- `plt.yscale()`: making a specific scale for example log scale and see matplotlib scale documentation for more information.

Time Series

```
from datetime import datetime, timedelta
from matplotlib import dates as mpl_dates
plt.plot_date(x_values, y_values,
              linestyle='style', marker='marker_shape')
plt.gcf().autofmt_xdate()
date_format = mpl_dates.DateFormatter('%b, %d, %Y')
plt.gca().xaxis.set_major_formatter(date_format)
data['Date'] = pd.to_datetime(data['Date'])
data.sort_values('Date', inplace=True)
```

- `from datetime import datetime, timedelta`: importing the datetime and timedelta module from the datetime library to use it to.
- `from matplotlib import dates as mpl_dates`: importing the dates module with name mpl_dates from the matplotlib library to use it.
- `plt.plot_date(x_values, y_values, linestyle='style', marker='marker_shape')`: `plt.plot_date()` is used for plotting time series data. `linestyle = 'style'` is used for choosing a specific style for the plot, for example, solid, making the data matched by a solid line. `marker = ''` used for choosing a marker shape.
- `plt.gcf().autofmt_xdate()`: `plt.gcf()` for getting the current figure. `plt.autofmt_xdate` for auto-formatting the appearance of the data on the x-axis.
- `date_format = mpl_dates.DateFormatter('%b, %d, %Y')`: setting a specific format for the date and visit matplotlib string format documentation for more.
- `plt.gca().xaxis.set_major_formatter(date_format)`: getting the current axis and setting a specific format using the variable we assigned the format to.
- `data['Date'] = pd.to_datetime(data['Date'])`: converting string to datetime as values.
- `data.sort_values('Date', inplace=True)`: sorting values by Date ascending and `inplace = True` for doing it permanently without the need to assign it again to the same variable.

Animations

```
from itertools import count
from matplotlib.animation import FuncAnimation

x_vals = []
y_vals = []
index = count()

def animate(i):
    x_vals.append(next(index))
    y_vals.append(random.randint(0, 5))
    plt.cla()
    plt.plot(x_vals, y_vals)

ani = FuncAnimation(plt.gcf(), animate, interval=1000)
```

- `from itertools import count`: importing the count module from the itertools library, which counts one number at a time when we get the next value of something.
- `x_vals = []; y_vals = []; index = count()`: assigning two empty lists and then assigning the index variable that we mentioned before `def animate(i) :` is a function for adding values to the list of x and y and plotting.
- `plt.cla()`: used for clearing the axis to prevent the variation of changing the color of the line with each update.
- `FuncAnimation(plt.gcf(), animate, interval=1000)`: function for live plot takes the current figure, and the function and more parameters see matplotlib FuncAnimation documentation for more, and `interval` is for choosing the interval of changing that takes values in milliseconds.

Subplots

```
fig, (ax1,..) = plt.subplots(nrows=value, ncols=value)
fig, ax = plt.subplots()
ax.set_title()
ax.set_xlabel()
ax.set_ylabel()
fig1, ax1 = plt.subplots()
fig2, ax2 = plt.subplots()
```

- `fig, (ax1,..)= plt.subplots(nrows=value, ncols=value)`: making a multiple subplot in the same figure. See and see matplotlib subplots documentation for more `(ax1,..)` is for unpacking them in multiple variables instead of one variable if needed.
- `ax.set_title(); ax.set_xlabel(); ax.set_ylabel()`: Note that the subplots take set_(label or title).
- `fig1, ax1 = plt.subplots(); fig2, ax2 = plt.subplots()`: for making the subplots in different figures.

Conclusion

At the end, this was a fast summary for the matplotlib tutorial playlist by me, and if it is useful, don't forget to pray for me `and for all our brothers in Palestine and Sudan`. And thank you :).