#18

Using Matplotlib for Visualization

MIS561 Data Visualization
Original Author: Lusi Yang

Introduction to Matplotlib

Matplotlib is probably the most popular plotting library for Python. It is used for data science and machine learning visualizations all around the world.

John Hunter began developing Matplotlib in 2003. It aimed to emulate the commands of the **MATLAB** software, which was the scientific standard back then. Several features such as the global style of MATLAB were introduced into Matplotlib to make the transition to Matplotlib easier for MATLAB users.

Overview of plots in Matplotlib

Plots in Matplotlib have a hierarchical structure that nests Python objects to create a tree-like structure.

The two main components of a plot are:

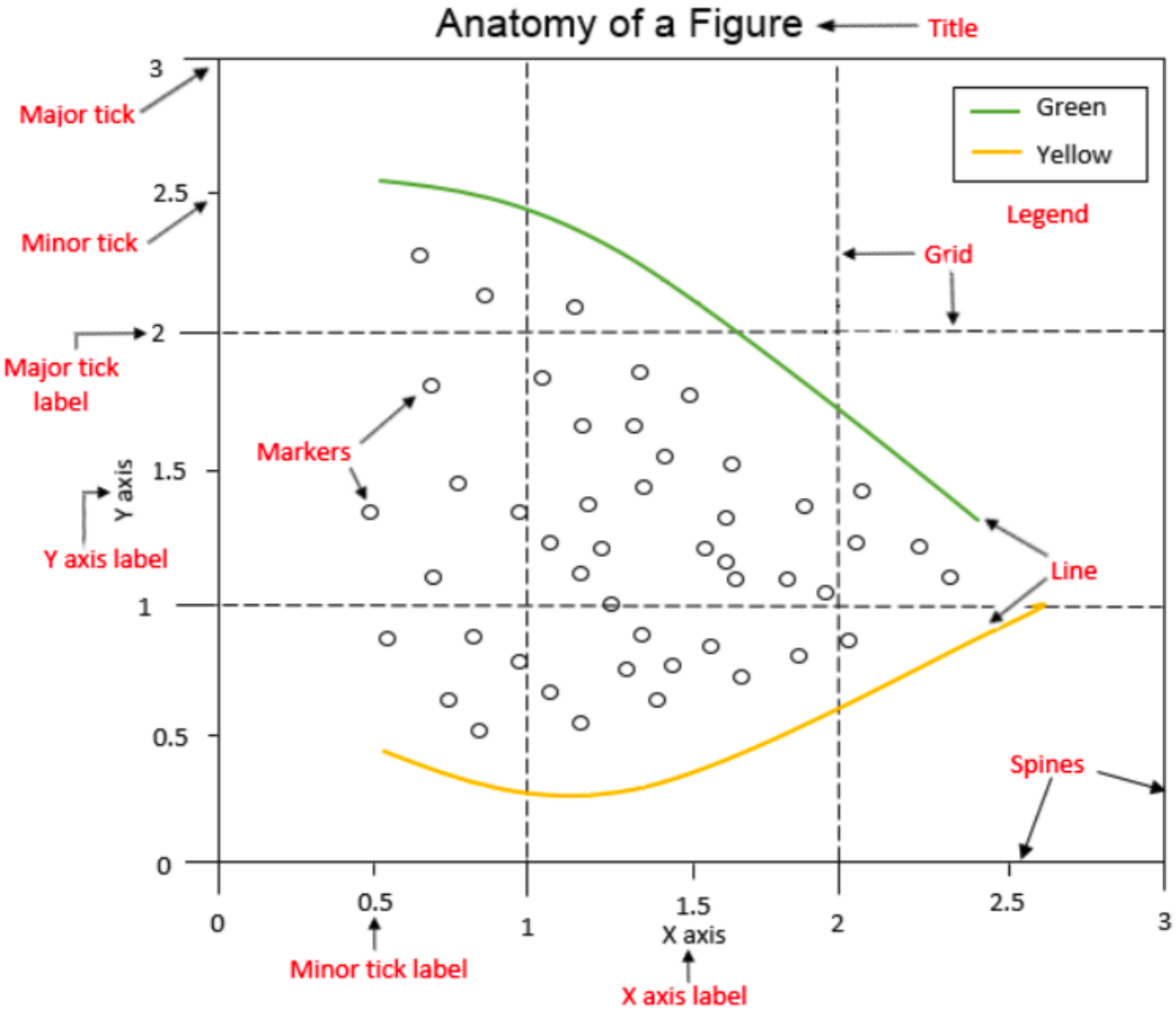
Figure

- The Figure is an outermost container and is used as a canvas to draw on. It allows you to draw multiple plots within it. It not only holds the Axes object but also has the capability to configure the **Title**.

Axes

- The Axes is an actual plot, or subplot, depending on whether you want to plot single or multiple visualizations. Its sub-objects include the x and y axis, spines, and legends.

Anatomy of a Matplotlib Figure



The components in the anatomy of a Figure object

Spines: Lines connecting the axis tick marks

Title: Text label of the whole Figure object

Legend: They describe the content of the plot

Grid: Vertical and horizontal lines used as an extension of the tick marks

X/Y axis label: Text label for the X/Y axis below the spines

Minor tick: Small value indicators between the major tick marks

Minor tick label: Text label that will be displayed at the minor ticks

Major tick: Major value indicators on the spines

Major tick label: Text label that will be displayed at the major ticks

Line: Plotting type that connects data points with a line

Markers: Plotting type that plots every data point with a defined marker

Pyplot basics

pyplot contains a simpler interface for creating visualizations, which allows the users to plot the data without explicitly configuring the **Figure** and **Axes** themselves. They are implicitly and automatically configured to achieve the desired output.

It is handy to use the alias **plt** to reference the imported submodule:

```
import matplotlib.pyplot as plt
```

Pyplot basics

Creating Figures

We use **plt.figure()** to create a new **Figure**. This function returns a Figure instance.

By default, the Figure has a width of 6.4 inches and a height of 4.8 inches with a dpi of 100. To change the default values of the Figure, we can use the parameters **figsize** and **dpi**.

```
plt.figure(figsize=(10, 5)) #To change the width and the height
plt.figure(dpi=300) #To change the dpi
```

Pyplot basics

Closing Figures

Figures that are not used anymore should be closed by explicitly calling **plt.close()**, which also cleans up memory efficiently.

If nothing is specified, the current Figure will be closed.

To close a specific Figure, you can either provide a reference to a Figure instance or provide the Figure number. To find the **number** of a figure object, we can make use of the **number** attribute:

```
plt.gcf().number
```

By using **plt.close('all')**, all Figures will be closed.

The following code shows how a Figure can be created and closed:

```
plt.figure(num=10) #Create Figure with Figure number 10
plt.close(10) #Close Figure with Figure number 10
```

Pyplot basics

Format Strings

Format strings are a neat way to specify **colors**, **marker types**, and **line styles**.

A format string is specified as "**[color][marker][line]**", where each item is optional. If the **color** is the only argument of the format string, you can use any **matplotlib.colors**.

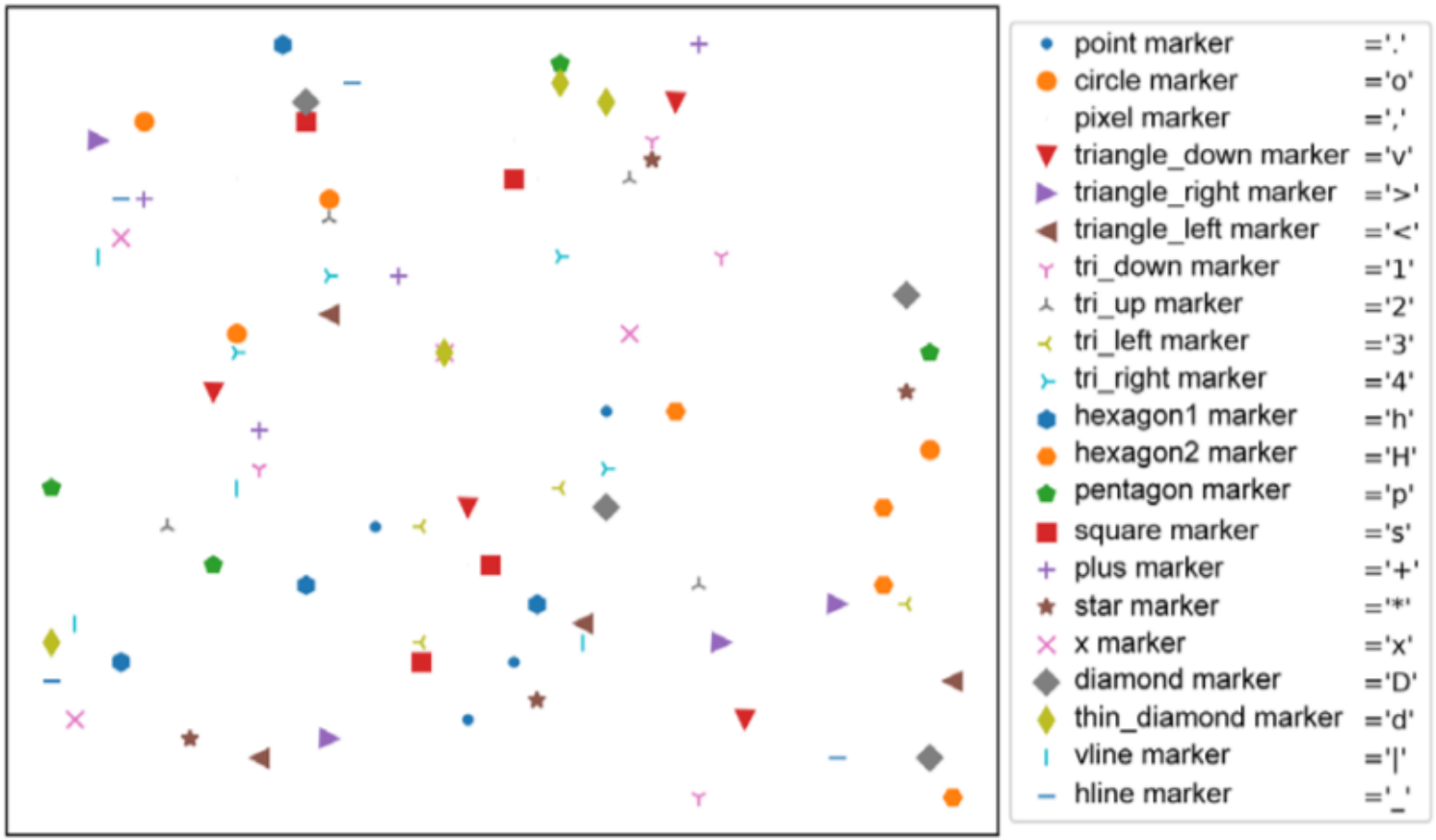
Matplotlib recognizes the following formats.

- RGB or RGBA float tuples (for example, (0.2, 0.4, 0.3) or (0.2, 0.4, 0.3, 0.5))
- RGB or RGBA hex strings (for example, '#0F0F0F' or '#0F0F0F0F')

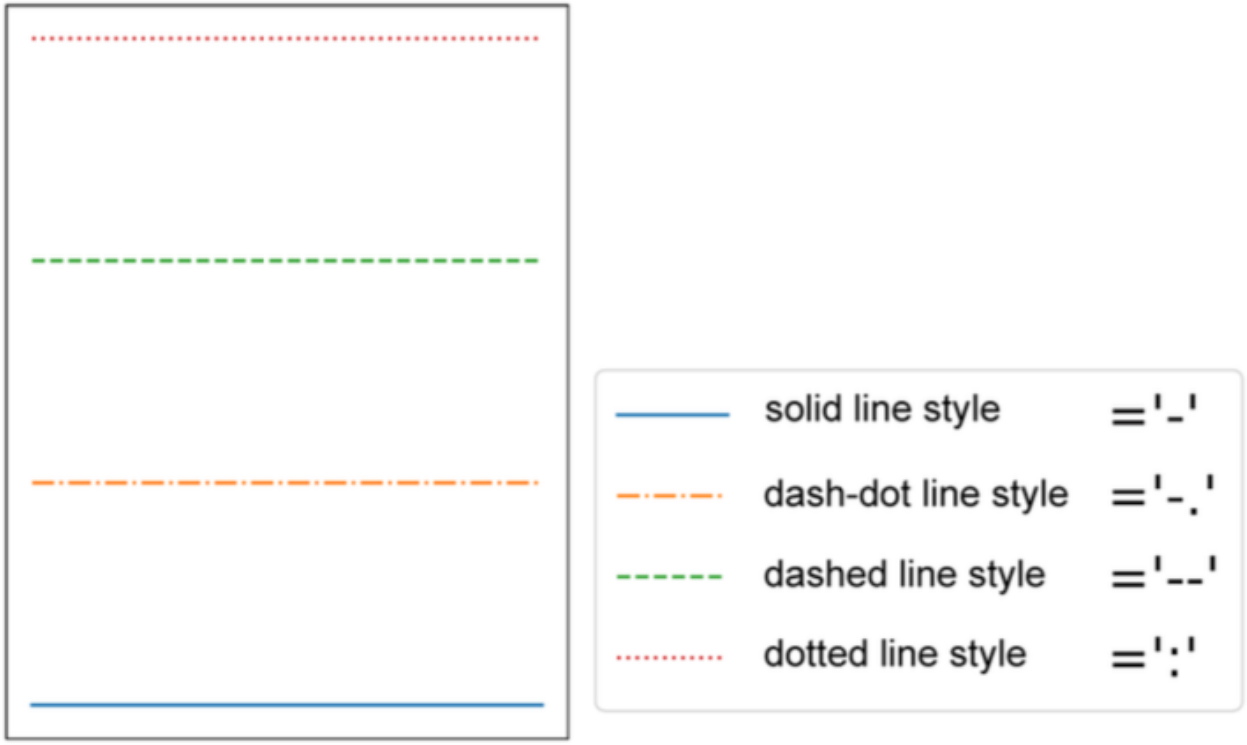
More about the RGBA color space:
https://en.wikipedia.org/wiki/RGBA_color_space

Format	Colors
'b'	blue
'r'	red
'g'	green
'm'	magenta
'c'	cyan
'b'	black
'w'	white
'y'	yellow

Color specified in string format



Markers in format strings


solid line style ='-'
dash-dot line style ='-.'
dashed line style ='--'
dotted line style =':'

Line styles

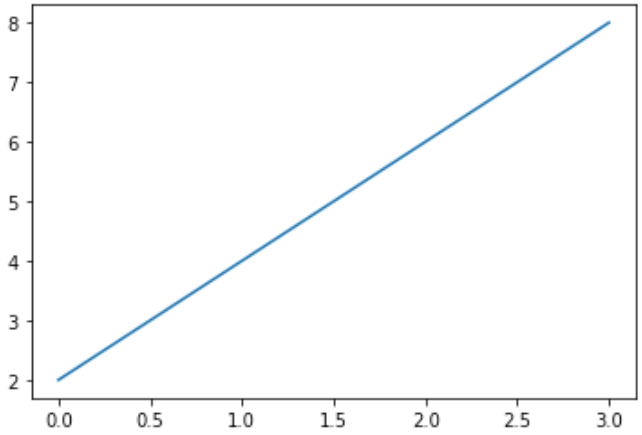
Pyplot basics – plotting

With **plt.plot([x], y, [fmt])**, you can plot data points as lines and/or markers. The function returns a list of Line2D objects representing the plotted data.

By default, if you do not provide a format string, the data points will be connected with straight, solid lines.

```
plt.figure()
plt.plot([0, 1, 2, 3],[2, 4, 6, 8])
```

```
[<matplotlib.lines.Line2D at 0x11153e5c0>]
```



Plotting data points as a line

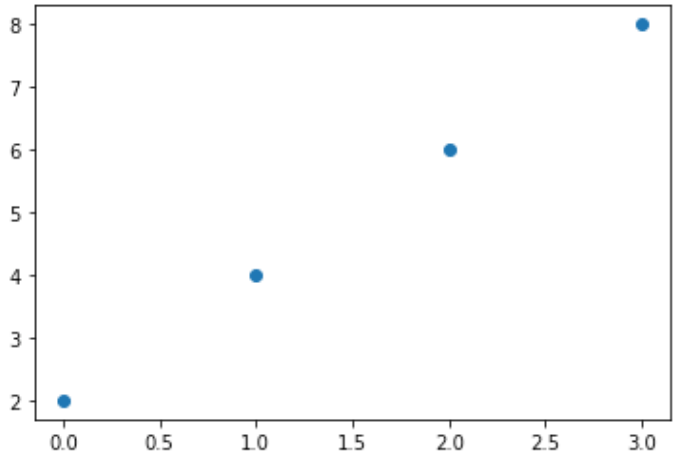
Pyplot basics – plotting

If you want to plot markers instead of lines, you can just specify a format string with any marker type.

For example, **plt.plot([0, 1, 2, 3], [2, 4, 6, 8], 'o')** displays data points as circles.

```
plt.figure()
plt.plot([0, 1, 2, 3],[2, 4, 6, 8],'o')
```

```
[<matplotlib.lines.Line2D at 0x111a4d978>]
```



Plotting data points with markers (circles)

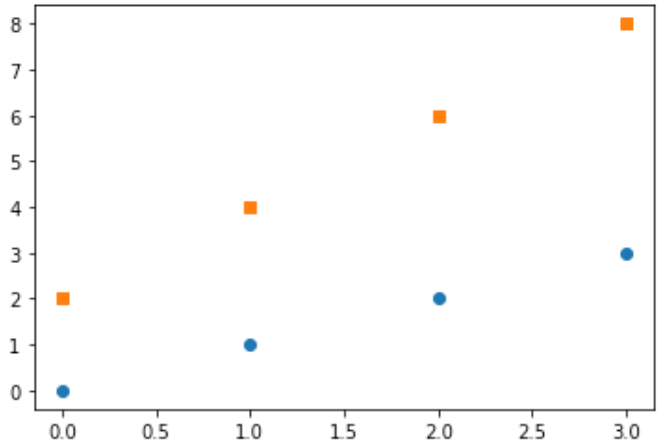
Pyplot basics – plotting

To plot multiple data pairs, the syntax **plt.plot([x], y, [fmt], [x], y2, [fmt2], ...)** can be used.

For example, **plt.plot([0, 1, 2, 3], 'o', [2, 4, 6, 8], 's')** results in:

```
plt.figure()
plt.plot([0, 1, 2, 3],'o',[2, 4, 6, 8],'s')
```

```
[<matplotlib.lines.Line2D at 0x111b13ac8>,
 <matplotlib.lines.Line2D at 0x111b13be0>]
```



Plotting data points with multiple markers

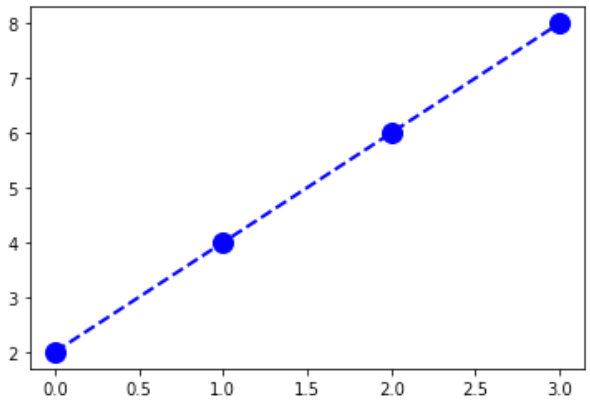
Pyplot basics – plotting

Any **Line2D** properties can be used instead of format strings to further customize the plot.

For example, the following code snippet shows how we can additionally specify the **linewidth** and the **markersize**.

```
plt.figure()
plt.plot([2, 4, 6, 8], color='blue', marker='o', linestyle='dashed',
  linewidth=2, markersize=12)
```

```
[<matplotlib.lines.Line2D at 0x111c35630>]
```



Plotting data points with specified properties

Pyplot basics

Saving Figures

plt.savefig(fname) saves the current Figure.

There are some useful optional parameters you can specify, such as **dpi**, **format**, or **transparent**.

The following code snippet gives an example of how you can save a Figure as a PNG file.

```
plt.figure()
plt.plot([1, 2, 4, 5], [1, 3, 4, 3], '-o')
plt.savefig('lineplot.png', dpi=300, bbox_inches='tight')
#bbox_inches='tight' removes the outer white margins
```

Activity 1

Visualizing Stock Trends by Using a Line Plot

Objective

We are interested in investing in stocks. We downloaded the stock prices for the "big five": Amazon, Google, Apple, Facebook, and Microsoft. We will create a line chart to show stock trends.

```
# Import statements
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

#display figures in Jupyter Notebook
%matplotlib inline
```

Use pandas to read the data located in the subfolder data.

```
# load datasets
google = pd.read_csv('./data/GOOGL_data.csv')
facebook = pd.read_csv('./data/FB_data.csv')
apple = pd.read_csv('./data/AAPL_data.csv')
amazon = pd.read_csv('./data/AMZN_data.csv')
microsoft = pd.read_csv('./data/MSFT_data.csv')
```

Load data from the five CVS files we downloaded

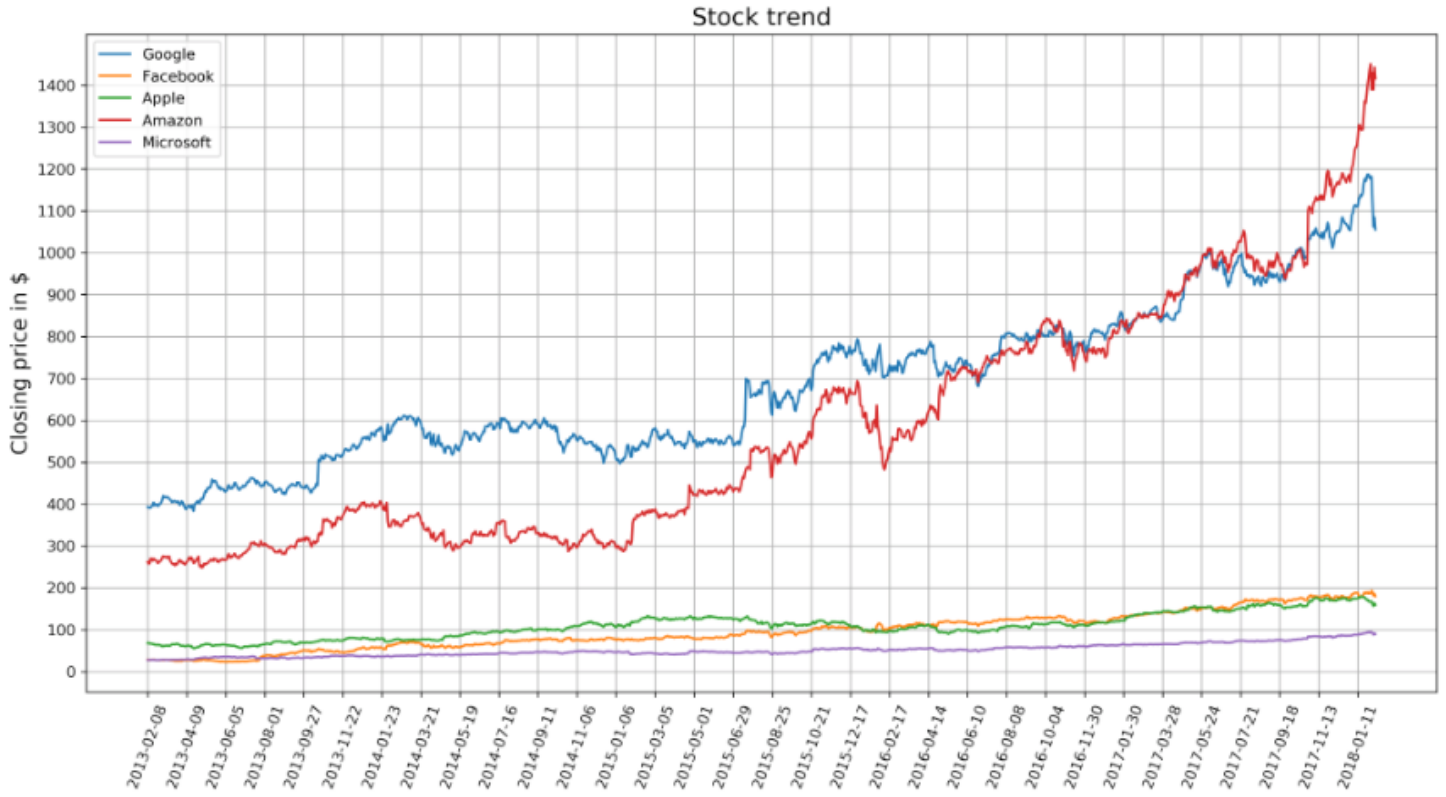
```
# Create figure
plt.figure(figsize=(16, 8), dpi=300)
# Plot data
plt.plot('date', 'close', data=google, label='Google')
plt.plot('date', 'close', data=facebook, label='Facebook')
plt.plot('date', 'close', data=apple, label='Apple')
plt.plot('date', 'close', data=amazon, label='Amazon')
plt.plot('date', 'close', data=microsoft, label='Microsoft')
# Specify ticks for x- and y-axis
plt.xticks(np.arange(0, 1260, 40), rotation=70)
plt.yticks(np.arange(0, 1450, 100))
# Add title and label for y-axis
plt.title('Stock trend', fontsize=16)
plt.ylabel('Closing price in $', fontsize=14)
# Add grid
plt.grid()
# Add legend
plt.legend()
# Show plot
plt.show()
```

Use Matplotlib to create a line chart visualizing the closing prices for the past five years (whole data sequence) for all five companies.

Add labels, titles, and a legend to make the visualization self-explanatory.

Add grid and legend to your plot.

Expected view



How to remove the grid in the plot?

Bar chart

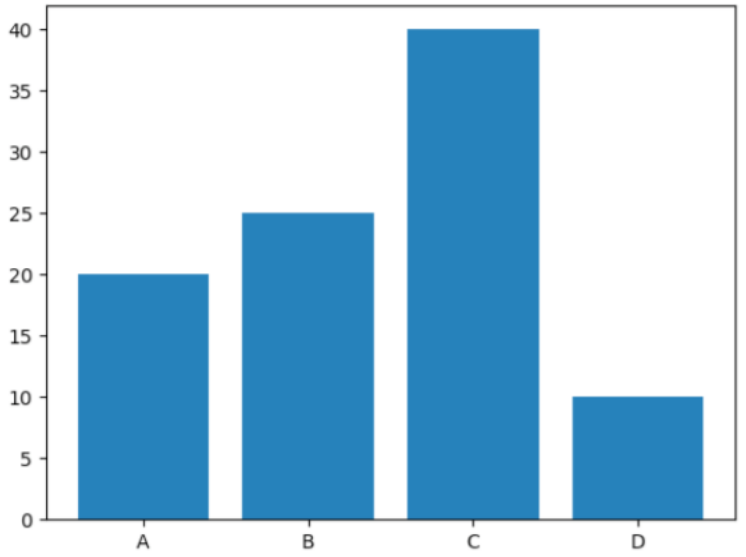
plt.bar(x, height, [width]) creates a vertical bar plot. For horizontal bars, use the **plt.barh()** function.

Parameters:

- **x** – Specifies the x coordinates of the bars
- **height –** Specifies the height of the bars
- **width** (optional) – Specifies the width of all bars; the default is 0.8

Example:

```
plt.bar(['A', 'B', 'C', 'D'], [20, 25, 40, 10])
```



Activity 2

Creating a Bar Plot for Movie Comparison

Objective

We are given five movies with scores from Rotten Tomatoes: Tomatometer score and Audience Score. We will create a bar chart to compare these two scores among the five movies.

```
# Import statements
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

#display figures in Jupyter Notebook
%matplotlib inline
```

Use pandas to read the data located in the subfolder data.

```
# Load dataset
movie_scores = pd.read_csv('./data/movie_scores.csv')
```

Load data from the the CVS files we downloaded

```
movie_scores.head(3)
```

	Unnamed: 0	MovieTitle	Tomatometer	AudienceScore
0	0	The Shape of Water	91	73
1	1	Black Panther	97	79
2	2	Dunkirk	92	81

Display the first three rows in the dataset

```
movie_scores.dtypes
```

```
Unnamed: 0        int64
MovieTitle       object
Tomatometer       int64
AudienceScore     int64
dtype: object
```

Display the data type of each field in the dataset

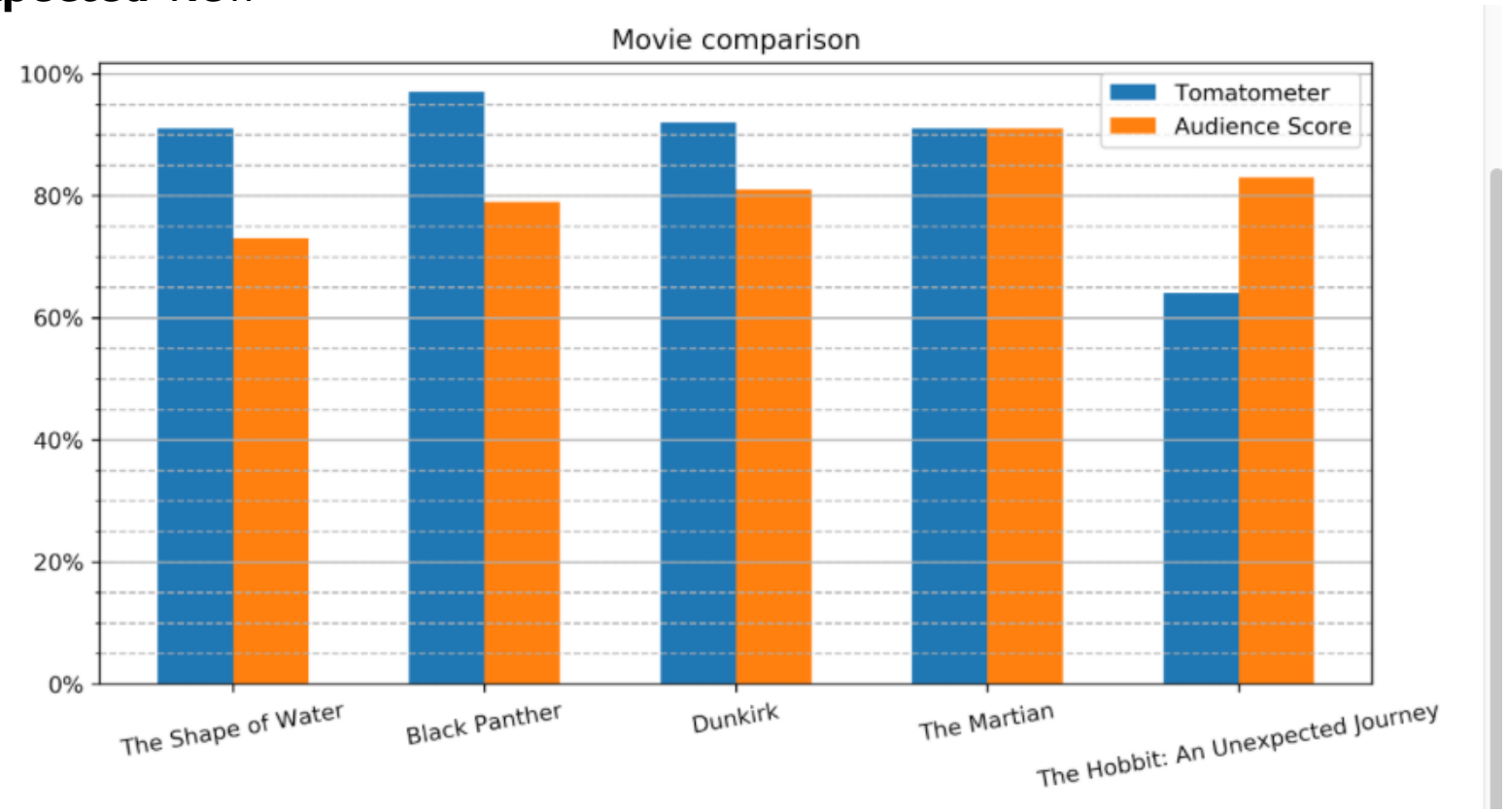
Use Matplotlib to create a visually-appealing bar plot comparing the two scores for all five movies.

```
# Create figure
plt.figure(figsize=(10, 5), dpi=300)
# Create bar plot
pos = np.arange(len(movie_scores['MovieTitle']))
width = 0.3
plt.bar(pos - width / 2, movie_scores['Tomatometer'], width, label='Tomatometer')
plt.bar(pos + width / 2, movie_scores['AudienceScore'], width, label='Audience Score')
# Specify ticks
plt.xticks(pos, rotation=10)
plt.yticks(np.arange(0, 101, 20))
# Get current Axes for setting tick labels and horizontal grid
ax = plt.gca()
# Set tick labels
ax.set_xticklabels(movie_scores['MovieTitle'])
ax.set_yticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
# Add minor ticks for y-axis in the interval of 5
ax.set_yticks(np.arange(0, 100, 5), minor=True)
# Add major horizontal grid with solid lines
ax.yaxis.grid(which='major')
# Add minor horizontal grid with dashed lines
ax.yaxis.grid(which='minor', linestyle='--')
# Add title
plt.title('Movie comparison')
# Add legend
plt.legend()
# Show plot
plt.show()
```

Set ticks and labels for x-axis and y-axis. Use the movie titles as labels for the x-axis.

Add grid, title and legend to the plot.

Expected view



How to remove the grid in the plot and change the size of the bars?

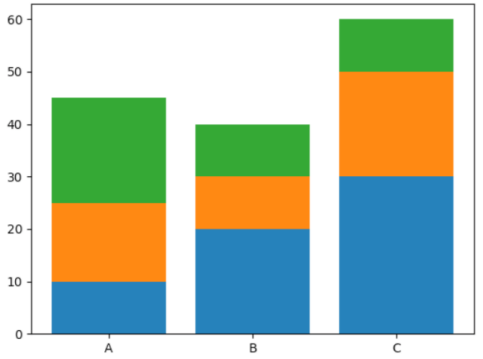
Stacked bar chart

A **stacked bar chart** uses the same **plt.bar** function as bar charts.

For each stacked bar, the **plt.bar** function must be called and the **bottom** parameter must be specified starting with the second stacked bar.

This will become clear with the example:

```
...
plt.bar(x, bars1)
plt.bar(x, bars2, bottom=bars1)
plt.bar(x, bars3, bottom=np.add(bars1, bars2))
...
```



Activity 3

Creating a Stacked Bar Plot to Visualize Restaurant Performance

Objective

You are the owner of a restaurant and due to a new law you have to introduce a smoking free day. To make as few losses as possible you want to visualize how much sales are made every day according to smoking and non-smoking people.

We will create a stacked bar chart to visualization the performance of the restaurant.

```
# Import statements
import pandas as sb
import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline
```

```
# Load dataset
bills = sns.load_dataset('tips')
```

Load the default data from the python library

```
bills
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2

Display the data structure

```
days = ['Thur', 'Fri', 'Sat', 'Sun']
days_range = np.arange(len(days))
smoker = ['Yes', 'No']
```

```
bills_by_days = [bills[bills['day'] == day] for day in days]
bills_by_days_smoker = [[bills_by_days[day][bills_by_days[day]['smoker'] == s] for s in smoker] for day in days_range]
total_by_days_smoker = [[bills_by_days_smoker[day][s]['total_bill'].sum() for s in range(len(smoker))] for day in days_range]
totals = np.asarray(total_by_days_smoker)
```

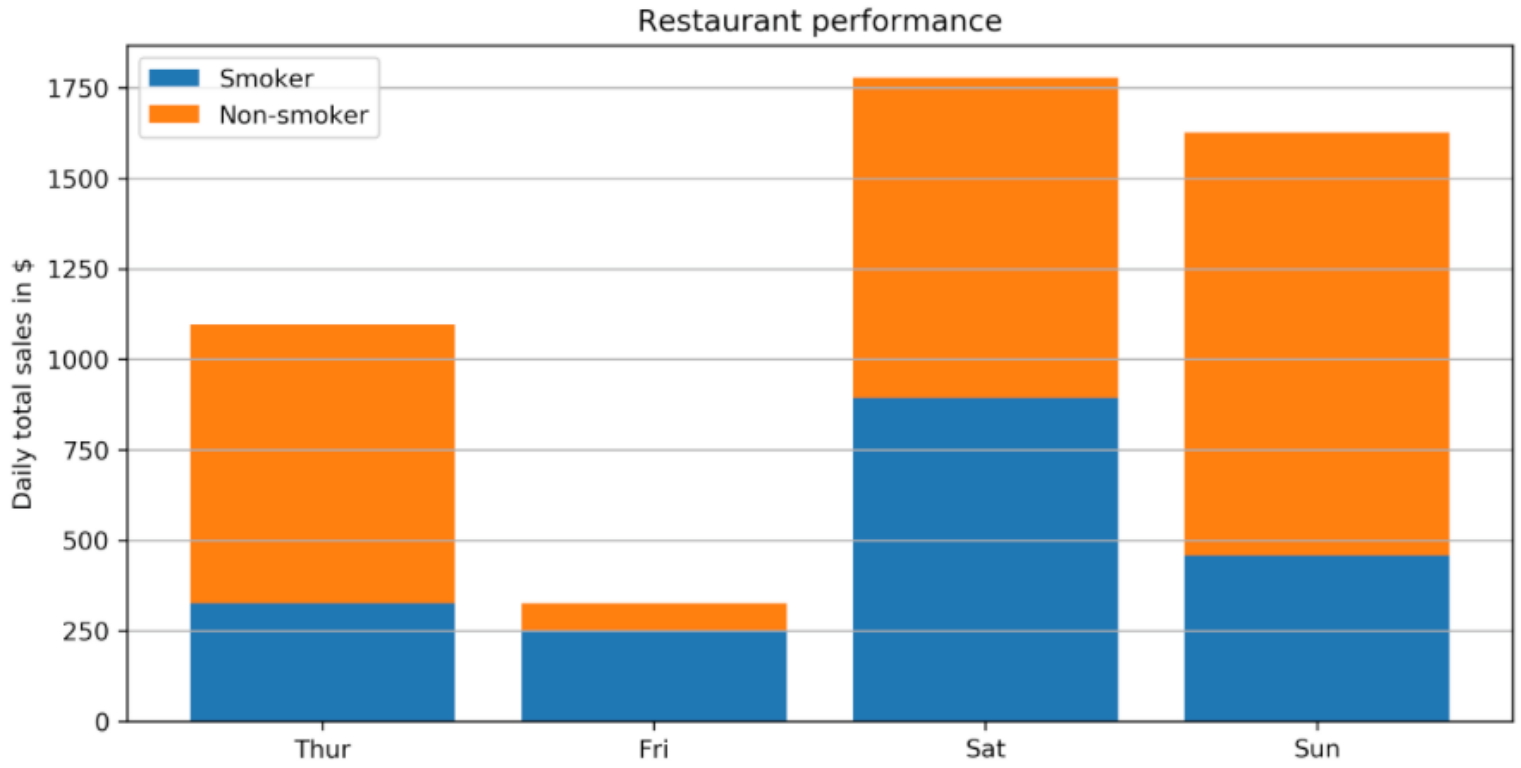
Calculate the sum of bills for each day and for both smokers and non-smokers

Create a stacked bar plot, stacking the summed smoking and non-smoking total bills separated for each day.

```
# Create figure
plt.figure(figsize=(10, 5), dpi=300)
# Create stacked bar plot
plt.bar(days_range, totals[:, 0], label='Smoker')
plt.bar(days_range, totals[:, 1], bottom=totals[:, 0], label='Non-smoker')
# Add legend
plt.legend()
# Add labels and title
plt.xticks(days_range)
ax = plt.gca()
ax.set_xticklabels(days)
ax.yaxis.grid()
plt.ylabel('Daily total sales in $')
plt.title('Restaurant performance')
# Show plot
plt.show()
```

Add a legend, labels, and a title.

Expected view



How to remove the grid in the plot?

Stacked area chart

plt.stackplot(x, y) creates a stacked area plot.

Parameters:

- **x** – Specifies the x-values of the data series.
- **y** – Specifies the y-values of the data series. For multiple series, either as a 2d array, or any number of 1D arrays, call the following function: **plt.stackplot(x, y1, y2, y3, ...).**
- **labels** (Optional): Specifies the labels as a list or tuple for each data series.

Activity 4

Comparing Smartphone Sales Units Using a Stacked Area Chart

Objective

You want to invest in one of the biggest five smartphone manufacturers. Looking at the quarterly sales units as part of a whole may be a good indicator to invest in one of the companies. We will compare smartphone sales units using a stacked area chart.

```
# Import statements
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

Use pandas to read the data located in the subfolder data.

```
# Load dataset
sales = pd.read_csv('./data/smartphone_sales.csv')
```

Load data from the the CVS file we downloaded

```
sales
```

	Unnamed: 0	Quarter	Apple	Samsung	Huawei	Xiaomi	OPPO
0	0	3Q16	43001	71734	32490	14926	24591
1	1	4Q16	77039	76783	40804	15751	26705
2	2	1Q17	51993	78776	34181	12707	30922
3	3	2Q17	44315	82855	35964	21179	26093
4	4	3Q17	45442	85605	36502	26853	29449
5	5	4Q17	73175	74027	43887	28188	25660
6	6	1Q18	54059	78565	40426	28498	28173
7	7	2Q18	44715	72336	49847	32826	28511

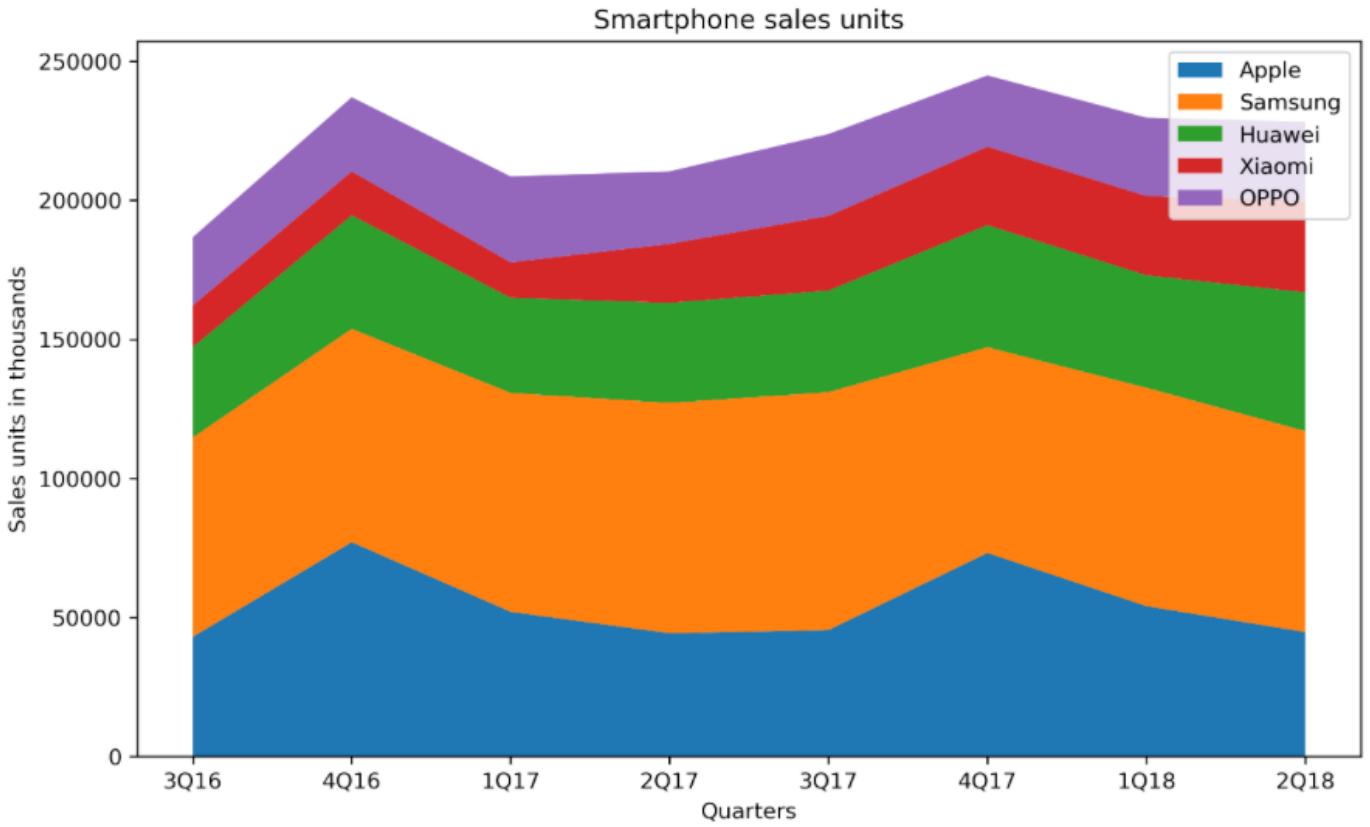
Display the data structure

Create a stacked area chart, specifying x-values, y-values, and labels. Use the column names as labels.

```
# Create figure
plt.figure(figsize=(10, 6), dpi=300)
# Create stacked area chart
labels = sales.columns[2:]
plt.stackplot('Quarter', 'Apple', 'Samsung', 'Huawei', 'Xiaomi', 'OPPO', data=sales, labels=labels)
# Add legend
plt.legend()
# Add labels and title
plt.xlabel('Quarters')
plt.ylabel('Sales units in thousands')
plt.title('Smartphone sales units')
# Show plot
plt.show()
```

Add a legend, labels and a title

Expected view



Hands-on time

1. Using the code in Activity 1, understand how to create a line chart with multiple data series. Execute the code and recreate the chart.

2. Using the code in Activity 2, understand how to create a bar chart with multiple data series. Execute the code and recreate the chart.

3. Using the code in Activity 3, understand how to aggregate data records into higher-order level, and plot the data in a stacked bar chart with multiple data series. Execute the code and recreate the chart.

4. Using the code in Activity 4, understand how to stacked area chart using with multiple data series. Execute the code and recreate the chart.

Check here for the detailed descriptions of functions:

https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.html