Lab 10 - Plotting

```
import numpy as np
```

```
seed = 42 # for reproducibility
np.random.seed(seed)
```

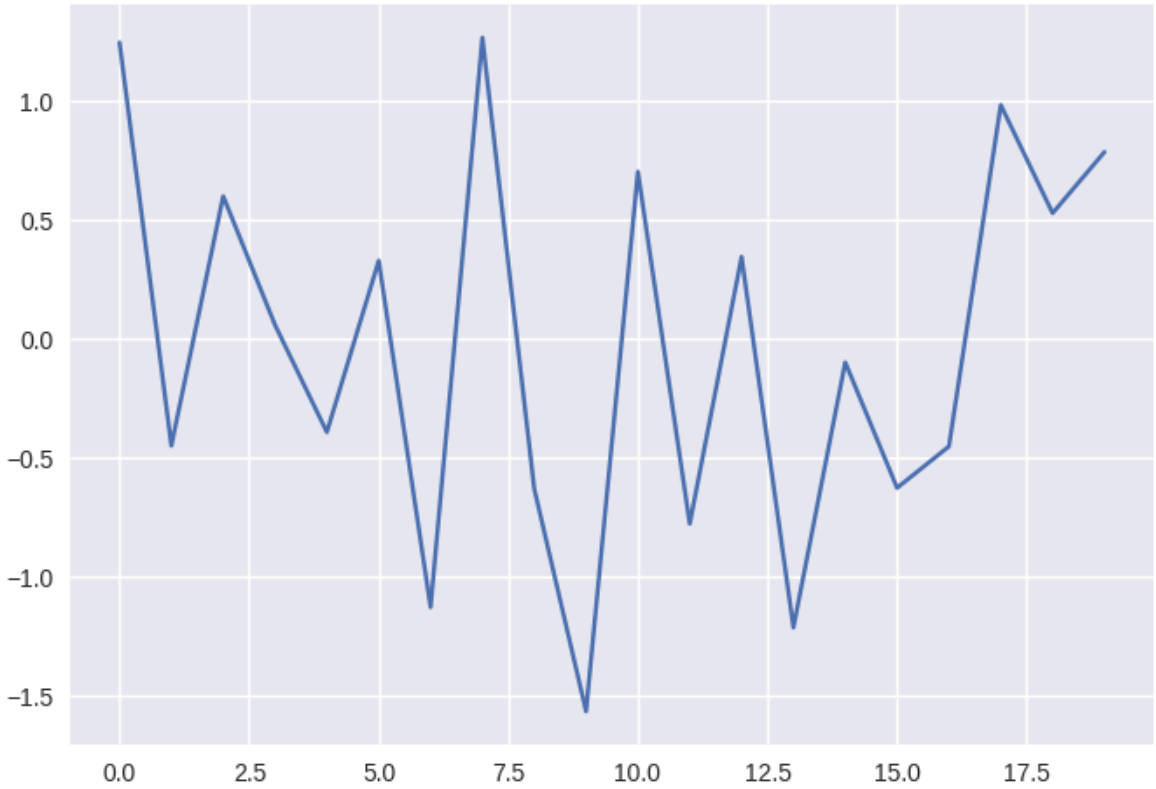
One-Dimensional Data Sets

```
import matplotlib.pyplot as plt
import seaborn

# https://seaborn.pydata.org/tutorial/aesthetics.html
# https://matplotlib.org/stable/gallery/style_sheets/style_sheets_reference.
plt.style.use('seaborn-v0_8')
```

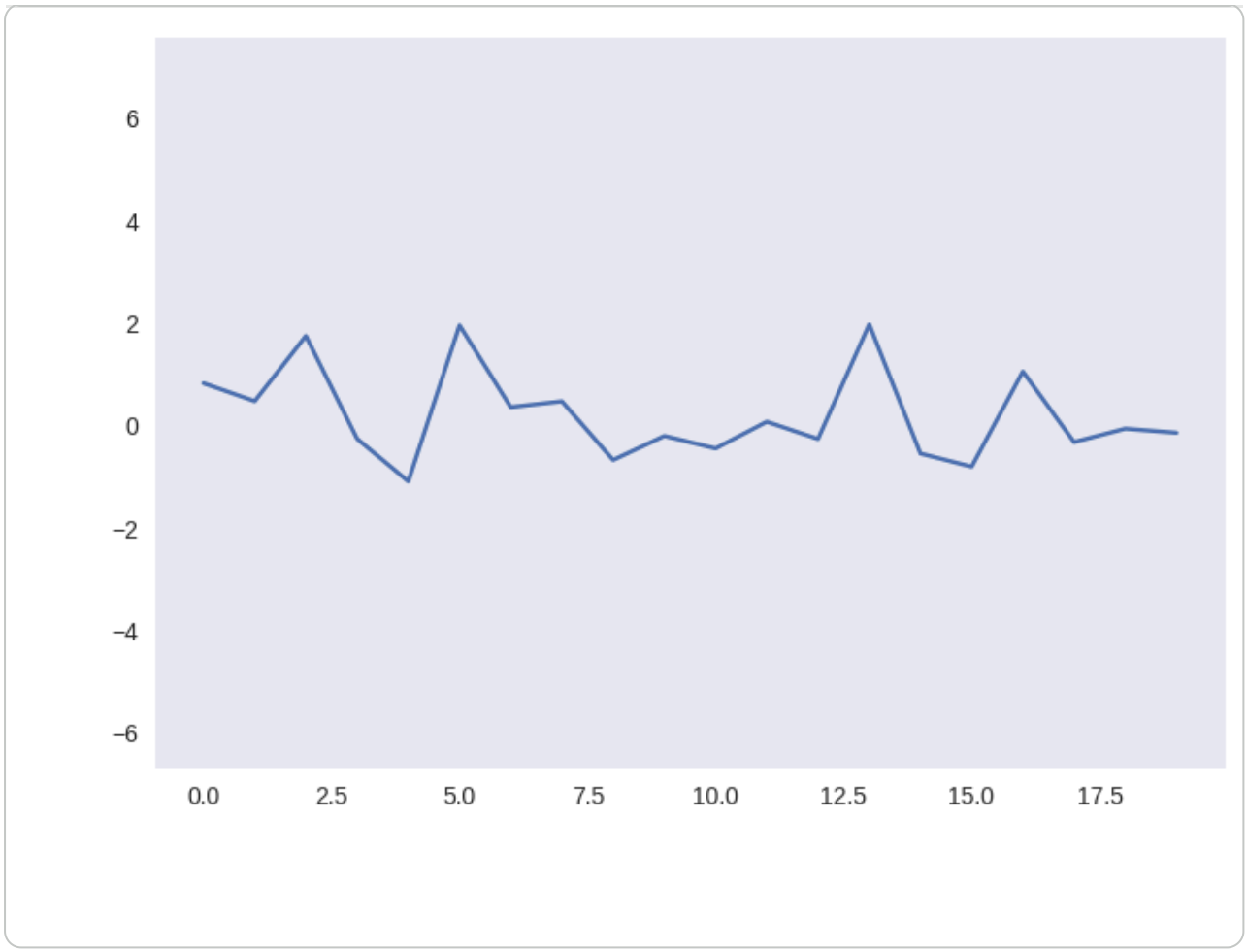
```
import numpy as np
y = np.random.standard_normal(20) # https://numpy.org/doc/2.1/reference/rand
x = np.arange(len(y)) # generaza un vector de valori
print(y)
print(x)
```

```
[ 1.24516048 -0.44969097  0.59906068  0.05751635 -0.39266889  0.32778703
 -1.12818318  1.26486793 -0.63042505 -1.56629305  0.70178254 -0.77792484
  0.34408821 -1.21408838 -0.09947714 -0.62703261 -0.45274251  0.98228964
  0.52783765  0.78468767]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

```
plt.plot(x, y)
plt.show()
```



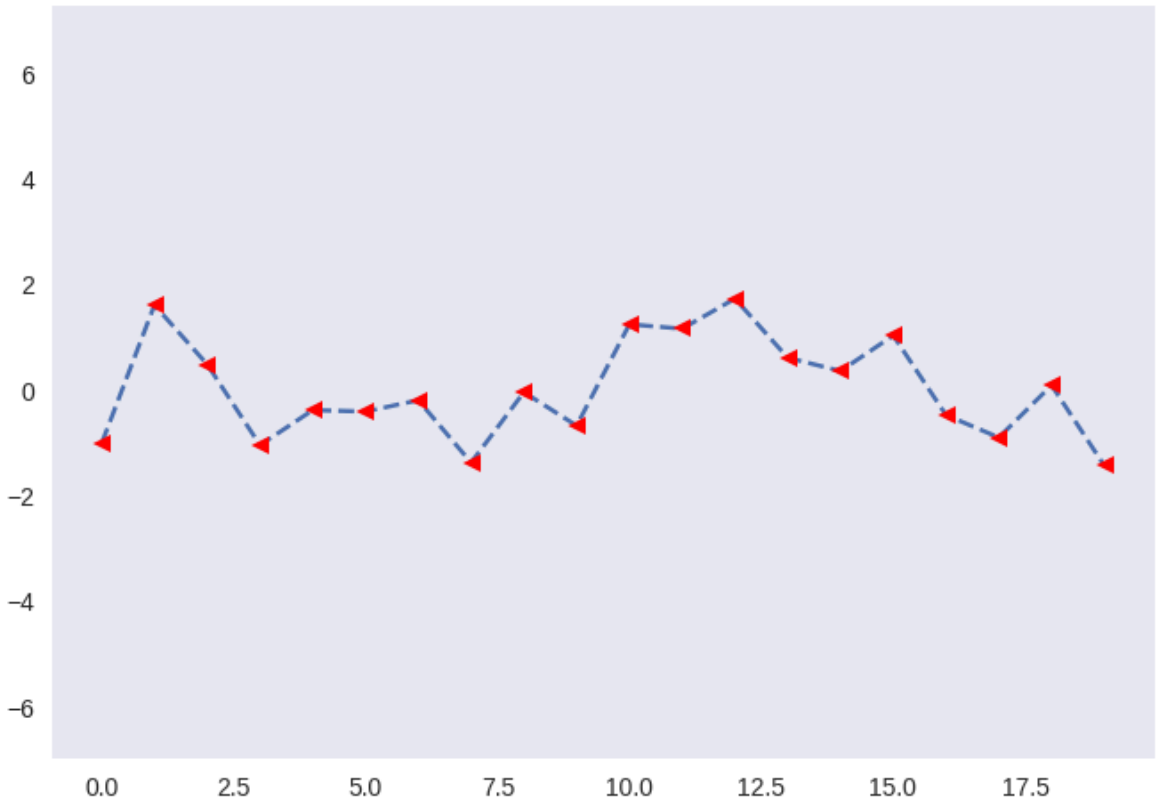

```
import numpy as np
y = np.random.standard_normal(20) # https://numpy.org/doc/2.1/reference/rand
x = np.arange(len(y)) # generaza un vector de valori
plt.plot(x, y)
plt.grid(False) # https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplo
plt.axis('equal') # https://matplotlib.org/stable/api/_as_gen/matplotlib.pyp
plt.show()
```



Character	Symbol
-	Solid line style
--	Dashed line style
-.	Dash-dot line style
:	Dotted line style
.	Point marker
,	Pixel marker
o	Circle marker
v	Triangle_down marker
^	Triangle_up marker
<	Triangle_left marker
>	Triangle_right marker
1	Tri_down marker
2	Tri_up marker
3	Tri_left marker
4	Tri_right marker
s	Square marker
p	Pentagon marker
*	Star marker
h	Hexagon1 marker
H	Hexagon2 marker
+	Plus marker

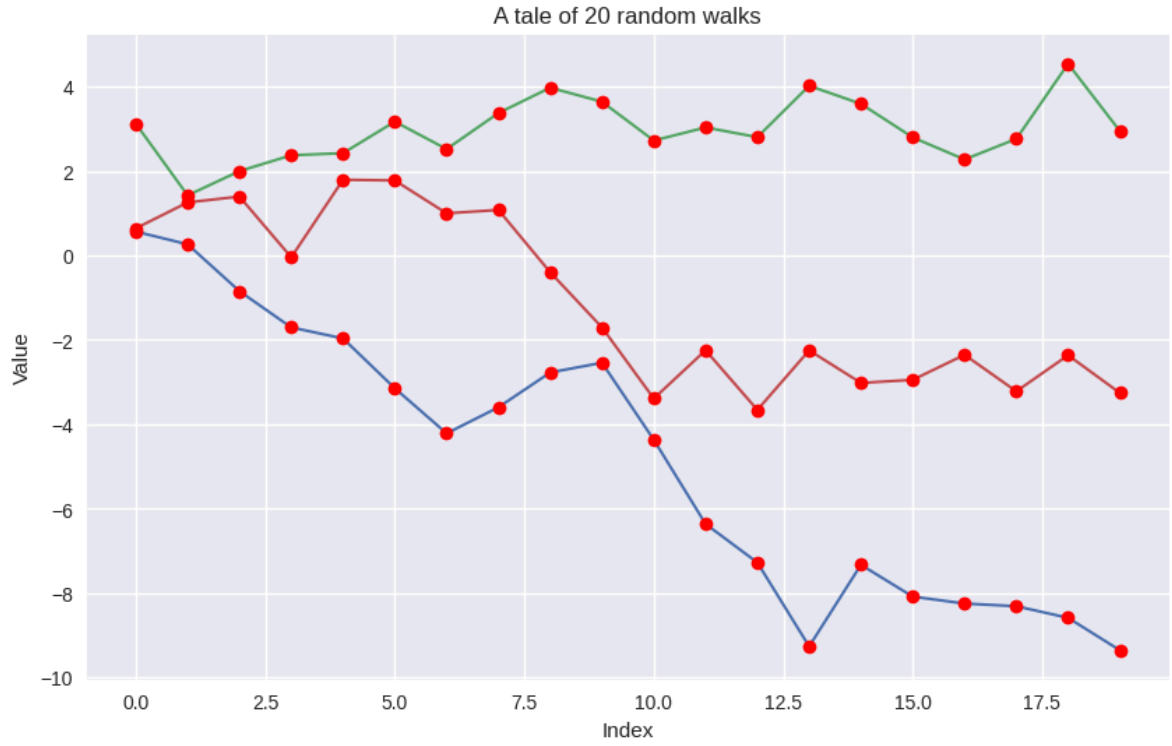
Character	Symbol
x	X marker
D	Diamond marker
d	Thin diamond marker
`	`
_	Hline marker

```
import numpy as np
y = np.random.standard_normal(20) # https://numpy.org/doc/2.1/reference/rand
x = np.arange(len(y)) # generaza un vector de valori
#plt.plot(x, y)
# Display the line as -- . Check the Notes section at https://matplotlib.org
plt.plot(x, y, '--')

#plt.plot(x, y, 'o') # Check the Notes section at https://matplotlib.org/sta
#plt.plot(x, y, 'D')
plt.plot(x, y, 'r<')

plt.grid(False) # https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplo
plt.axis('equal') # https://matplotlib.org/stable/api/_as_gen/matplotlib.pyp
plt.show()
```



Two-Dimensional Data Sets

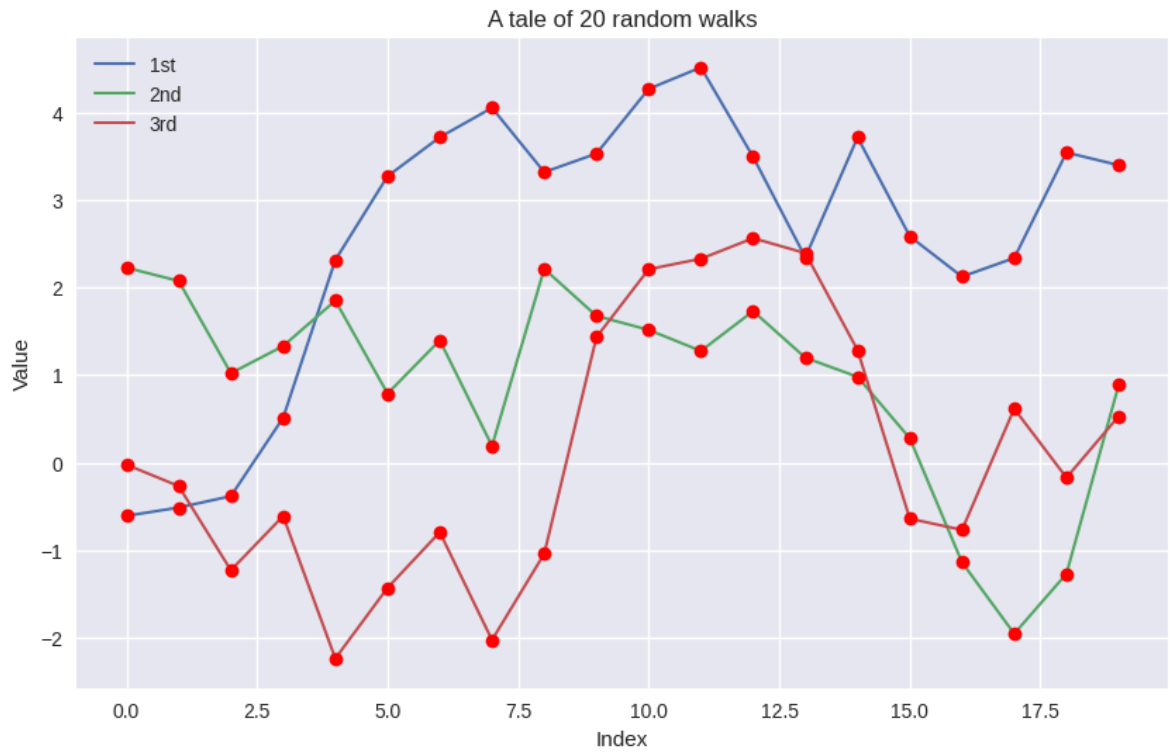
```
import matplotlib.pyplot as plt
import numpy
import seaborn

# y = np.random.standard_normal((20, 3))
# the values differ grately
y = np.random.standard_normal((20, 3)).cumsum(axis = 0)

plt.figure(figsize=(10, 6))
plt.plot(y, lw=1.5)
plt.plot(y, 'ro')
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('A tale of 20 random walks') # generated by colab
plt.show()
```



In such a case, further annotations might be helpful to better read the plot. You can add individual labels to each data set and have them listed in the legend. The function `plt.legend()` accepts different locality parameters. 0 stands for best location, in the sense that as little data as possible is hidden by the legend.

Documentation:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.legend.html

```
import matplotlib.pyplot as plt
import numpy
import seaborn

# y = np.random.standard_normal((20, 3))
# the values differ grately
y = np.random.standard_normal((20, 3)).cumsum(axis = 0)

plt.figure(figsize=(10, 6))
plt.plot(y, lw=1.5)
plt.plot(y, 'ro')
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('A tale of 20 random walks') # generated by colab
plt.legend(['1st', '2nd', '3rd'], loc=2) # 0 - auto
plt.show()
```



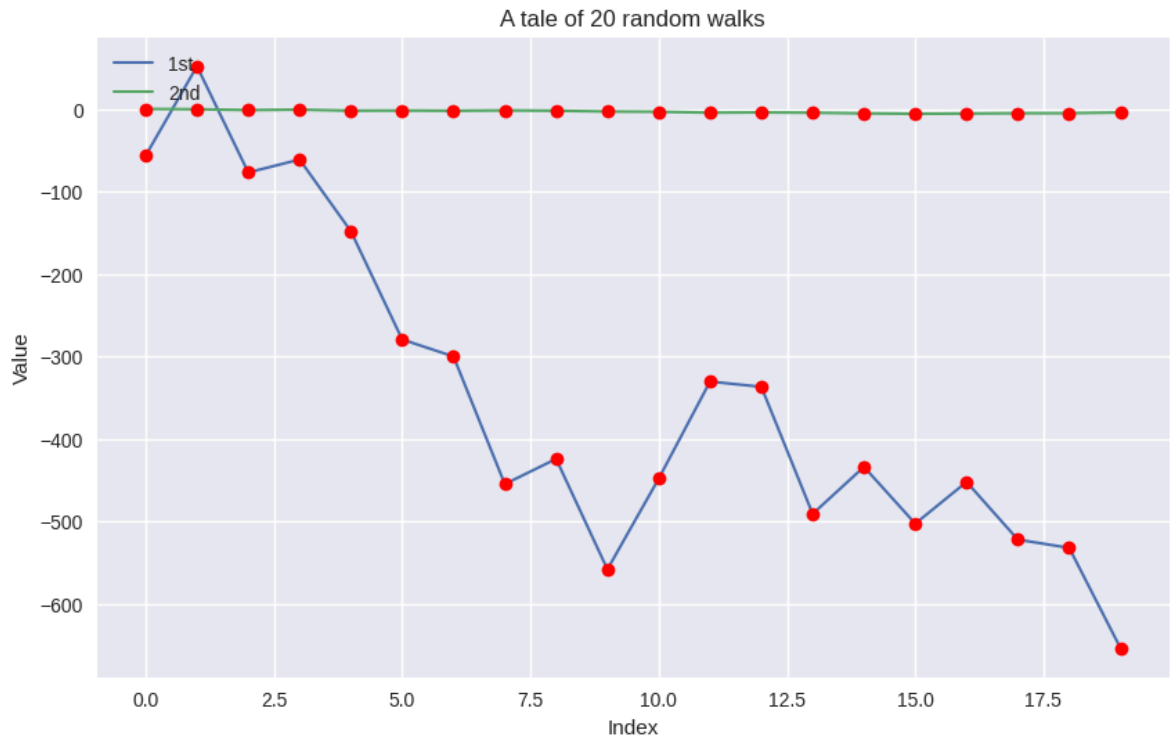
Further location options for `plt.legend()`:

Loc Code	Description
Default	Upper right
0	Best possible

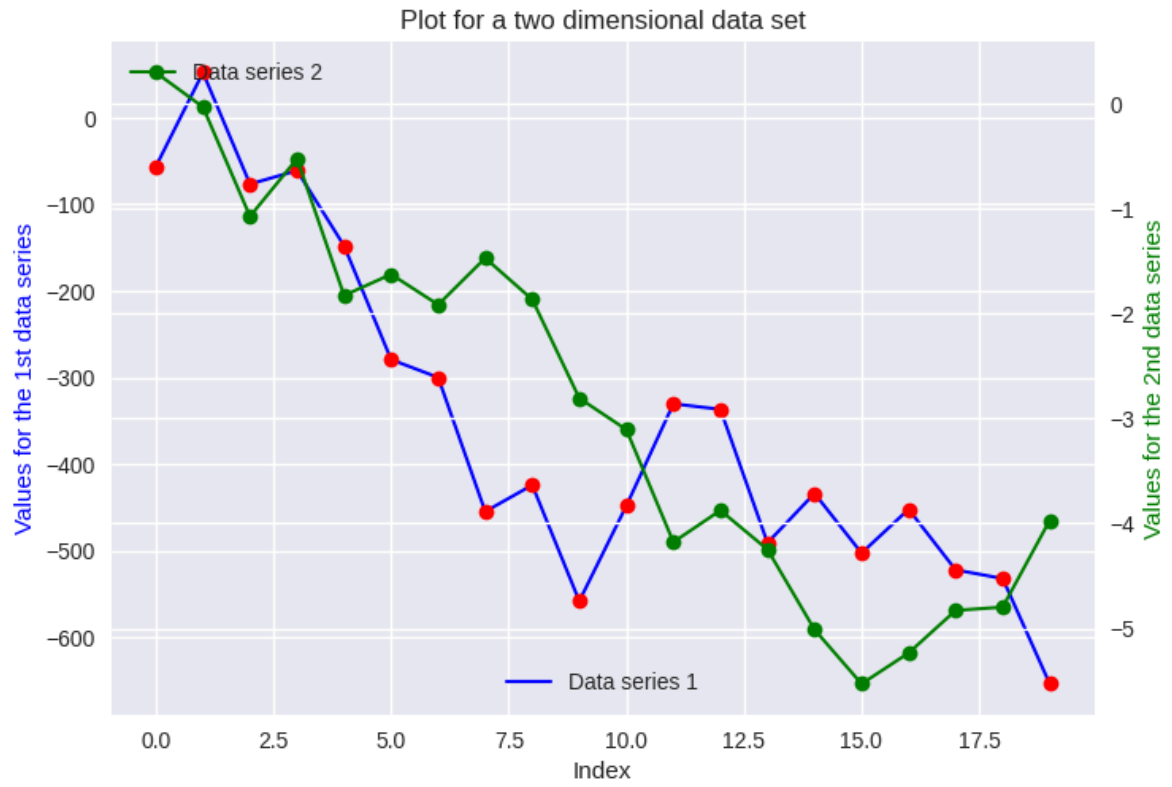
Loc Code	Description
1	Upper right
2	Upper left
3	Lower left
4	Lower right
5	Right
6	Center left
7	Center right
8	Lower center
9	Upper center
10	Center

Multiple data sets with a similar scaling, like simulated paths for the same financial risk factor, can be plotted using a single y-axis. However, often data sets show rather different scalings and the plotting of such data with a single yscale generally leads to a significant loss of visual information. To illustrate the effect, the following example scales the first of the two data subsets by a factor of 100 and plots the data again:

```
import matplotlib.pyplot as plt
import numpy
import seaborn

y = np.random.standard_normal((20, 2)).cumsum(axis = 0)
y[:,0] = y[:,0] * 100 # we make the values 100 times larger

plt.figure(figsize=(10, 6))
plt.plot(y, lw=1.5)
plt.plot(y, 'ro')
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('A tale of 20 random walks') # generated by colab
plt.legend(['1st', '2nd'], loc=2) # 0 - auto
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy
import seaborn

fig, ax1 =plt.subplots() # 1) Defining the figure and the axis object
ax1.plot(y[:,0], 'b', lw=1.5, label='Data series 1')
ax1.plot(y[:,0], 'ro')
ax1.legend(loc=8)
ax1.set_xlabel('Index')
ax1.set_ylabel('Values for the 1st data series', color='b')
ax1.set_title('Plot for a two dimensional data set')
ax2 = ax1.twinx() # Defining the second axis object
ax2.plot(y[:,1], 'g', lw=1.5, label='Data series 2')
ax2.plot(y[:,1], 'go')
ax2.legend(loc=2)
ax2.set_ylabel('Values for the 2nd data series', color='g')
plt.show()
```

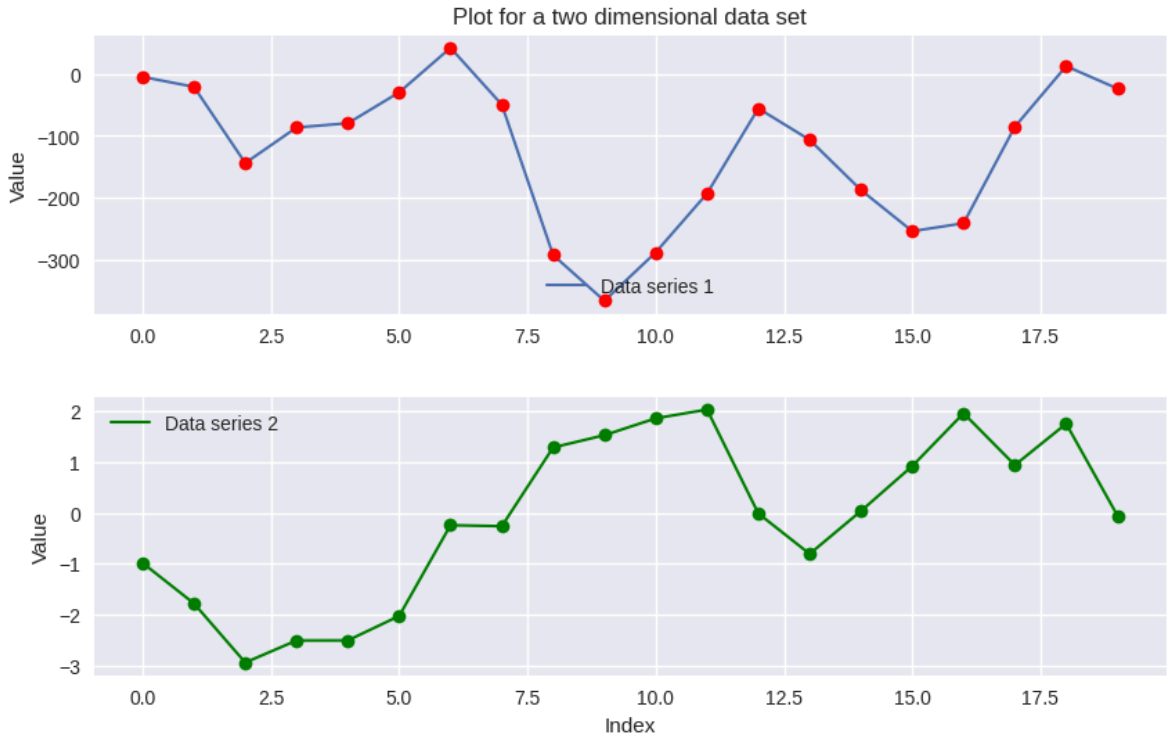


This option gives even more freedom to handle the two data sets. In `plt.subplot(211)` (`plt.subplot(2, 1, 1)`), the three digits represent:

- 2 ⟶ number of rows of subplots
- 1 ⟶ number of columns
- 1 ⟶ this is the first subplot (top one)

```
import matplotlib.pyplot as plt
import numpy
import seaborn

y = np.random.standard_normal((20, 2)).cumsum(axis = 0)
y[:,0] = y[:,0] * 100 # we make the values 100 times larger

plt.figure(figsize=(10, 6))
# Two ways of defining the number of rows and columns
# plt.subplot(211)
plt.subplot(2, 1, 1)
plt.plot(y[:,0], lw=1.5, label='Data series 1')
plt.plot(y[:,0], 'ro')
plt.legend(loc=8)
#plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Plot for a two dimensional data set')
```

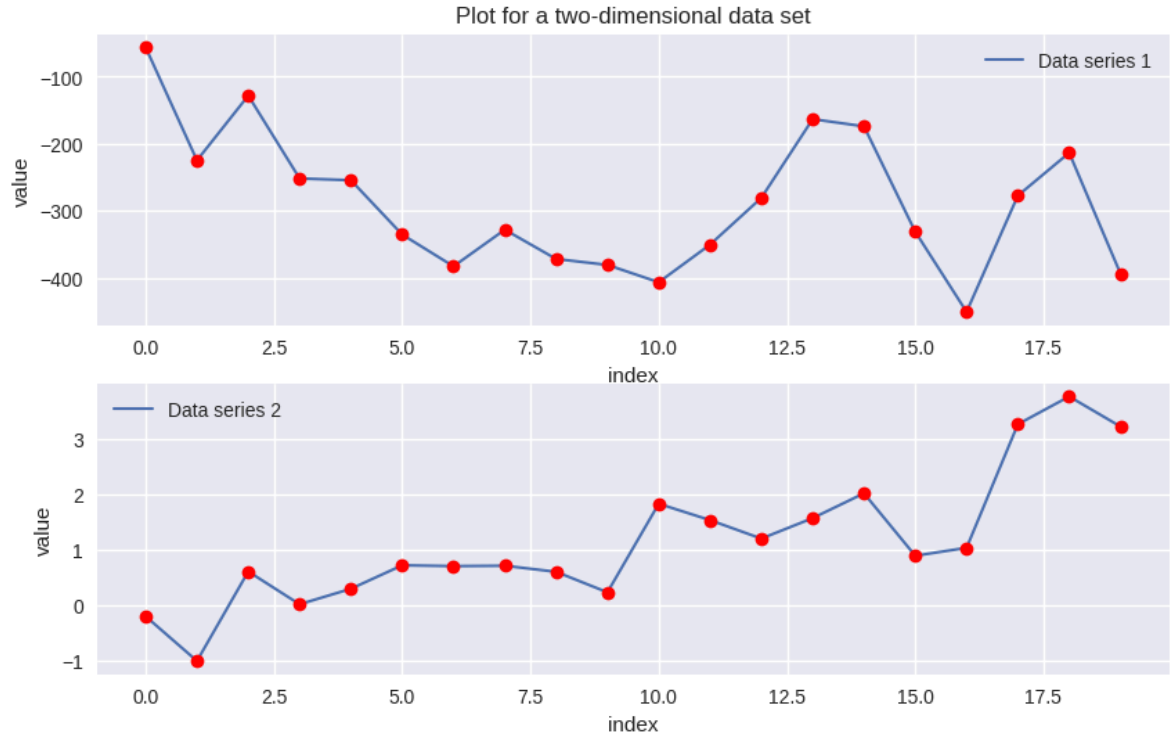
```
plt.subplot(2, 1, 2)
plt.plot(y[:,1], 'g', lw=1.5, label='Data series 2')
plt.plot(y[:,1], 'go')
plt.legend(loc=2)
plt.xlabel('Index')
plt.ylabel('Value')
#plt.tight_layout() # Added to increase space between subplots
plt.subplots_adjust(hspace=0.3) # Added to increase space between subplots
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

y = np.random.standard_normal((20, 2)).cumsum(axis=0) # rows
y[:, 0] = y[:, 0] * 100

plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.plot(y[:,0], lw=1.5, label='Data series 1')
plt.plot(y[:,0], 'ro') #markers
plt.legend(loc=0)
plt.xlabel('index')
plt.ylabel('value')
plt.title('Plot for a two-dimensional data set')
```

```
plt.subplot(2, 1, 2)
plt.plot(y[:,1], lw=1.5, label='Data series 2')
plt.plot(y[:,1], 'ro')
plt.legend(loc=0)
plt.xlabel('index')
plt.ylabel('value')
```

```
Text(0, 0.5, 'value')
```

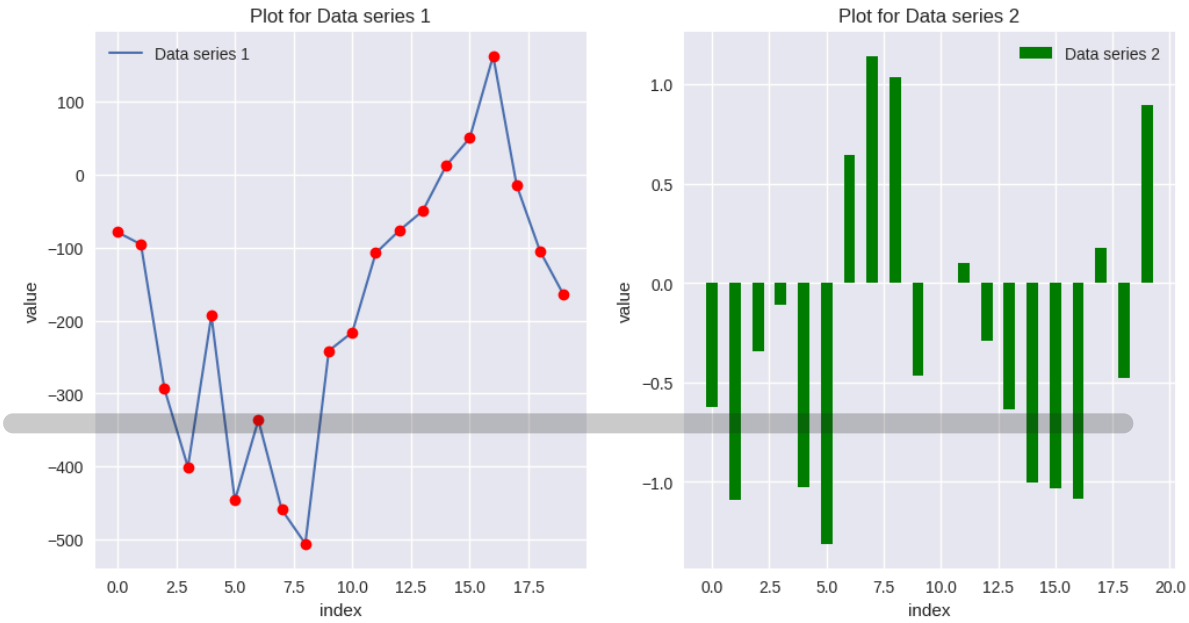


With the subplot approach one has the freedom to combine arbitrary kinds of plots that matplotlib offers.

```
import numpy as np
import matplotlib.pyplot as plt

y = np.random.standard_normal((20, 2)).cumsum(axis=0) # rows
y[:, 0] = y[:, 0] * 100

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(y[:,0], lw=1.5, label='Data series 1')
plt.plot(y[:,0], 'ro') #markers
plt.legend(loc=0)
plt.xlabel('index')
```

```
plt.ylabel('value')
plt.title('Plot for Data series 1')
plt.subplot(1, 2, 2)
plt.bar(np.arange(len(y)), y[:,1], width=0.5, color='g', label='Data series
plt.legend(loc=0)
plt.xlabel('index')
plt.ylabel('value')
plt.title('Plot for Data series 2')
#
```

```
Text(0.5, 1.0, 'Plot for Data series 2')
```

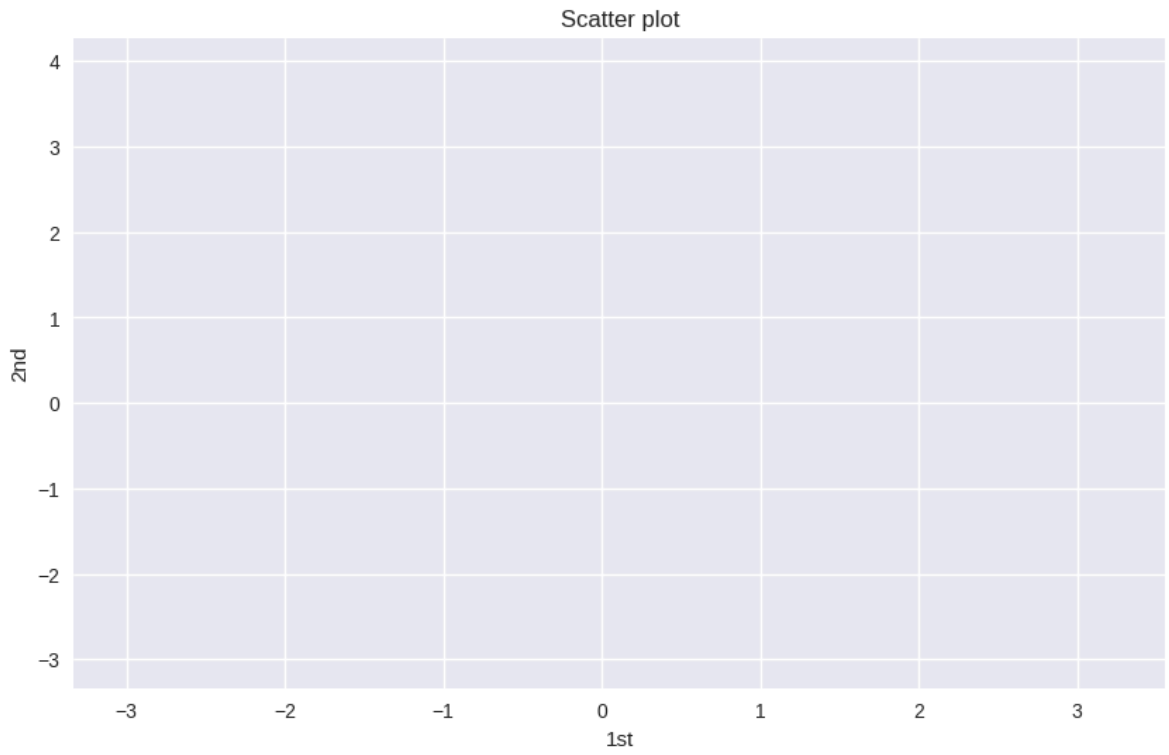


Other Plot Styles

Scatter plot, where the values of one data set serve as the x values for the other data set. This plot type might be used, for example, for plotting the returns of one financial time series against those of another one. This example uses a new two-dimensional data set with some more data:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
y = np.random.standard_normal((1000, 2))

plt.figure(figsize=(10, 6))
plt.plot(y[:,0], y[:,1], 'rx')
plt.xlabel('1st')
plt.ylabel('2nd')
plt.title('Scatter plot')
plt.show()
```

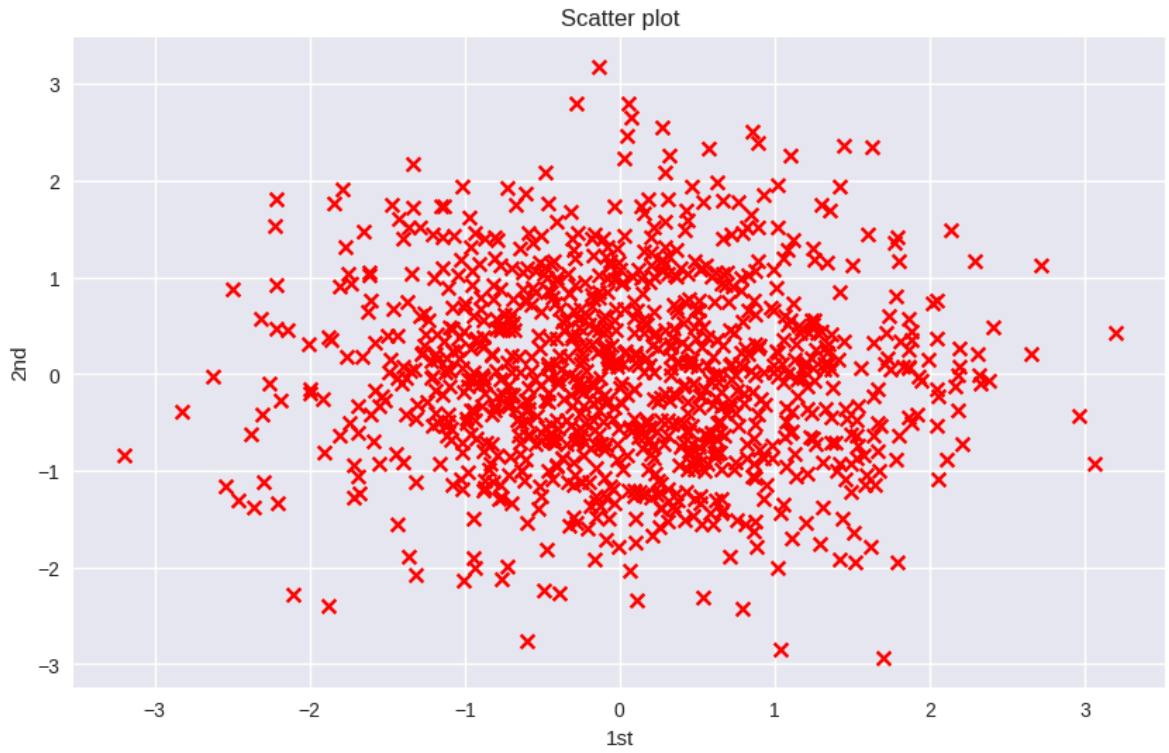


matplotlib also provides a specific function to generate scatter plots. It basically works in the same way, but provides some additional features.

Documentation:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html

```
import matplotlib.pyplot as plt
import numpy as np

y = np.random.standard_normal((1000, 2))

plt.figure(figsize=(10, 6))
# plt.scatter(y[:,0], y[:,1], marker='x')
plt.scatter(y[:,0], y[:,1], c='red', marker='x')
```

```
plt.xlabel('1st')
plt.ylabel('2nd')
plt.title('Scatter plot')
plt.show()
```



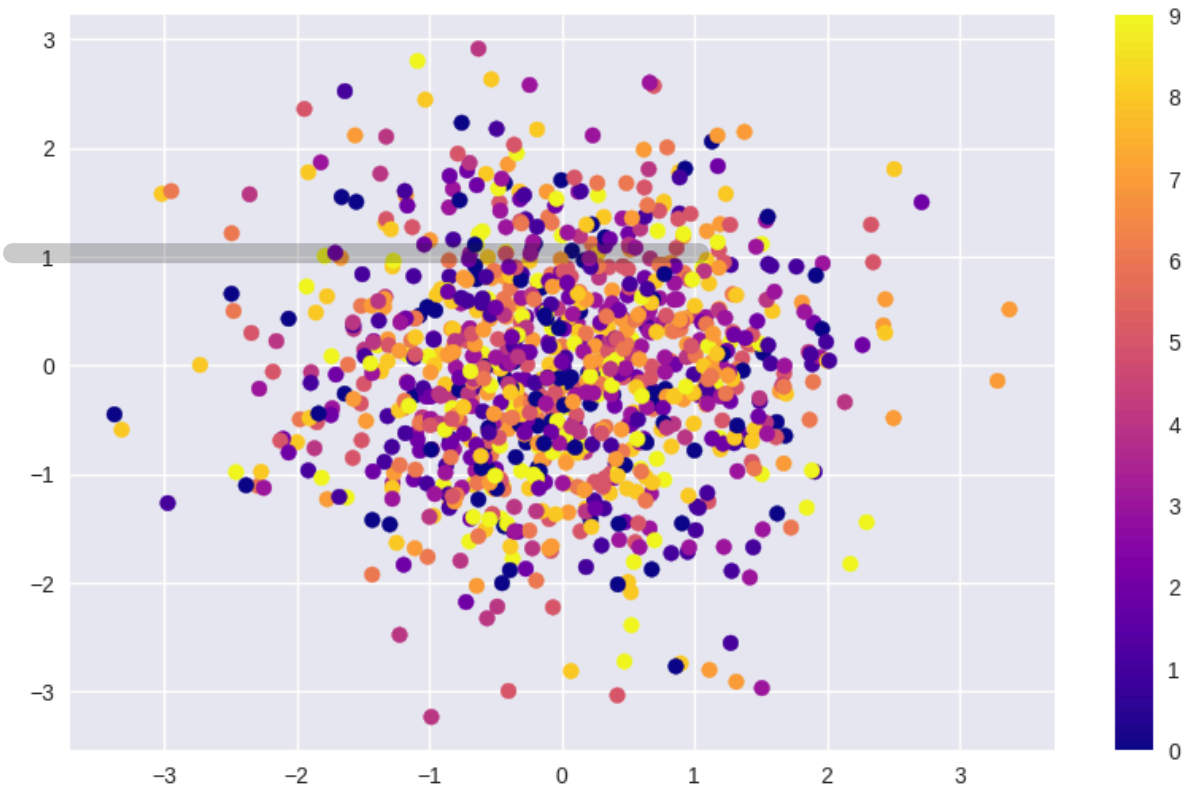
Among other things, the `plt.scatter()` plotting function allows the addition of a third dimension, which can be visualized through different colors and be described by the use of a color bar. The figure below shows a scatter plot where there is a third dimension illustrated by different colors of the single dots and with a color bar as a legend for the colors. To this end, the following code generates a third data set with random data, this time consisting of integers between 0 and 10.

Documentation: https://matplotlib.org/stable/users/explain/colors/colormaps.html

```
import matplotlib.pyplot as plt
import numpy as np

y = np.random.standard_normal((1000, 2))
c = np.random.randint(0, 10, len(y))

plt.figure(figsize=(10, 6))
#plt.scatter(y[:,0], y[:,1], c=c, cmap=plt.cm.jet)
```

```
#plt.scatter(y[:,0], y[:,1], c=c, cmap='coolwarm')
plt.scatter(y[:,0], y[:,1], c=c, cmap='plasma', marker='o') # https://matplo
plt.colorbar()
```

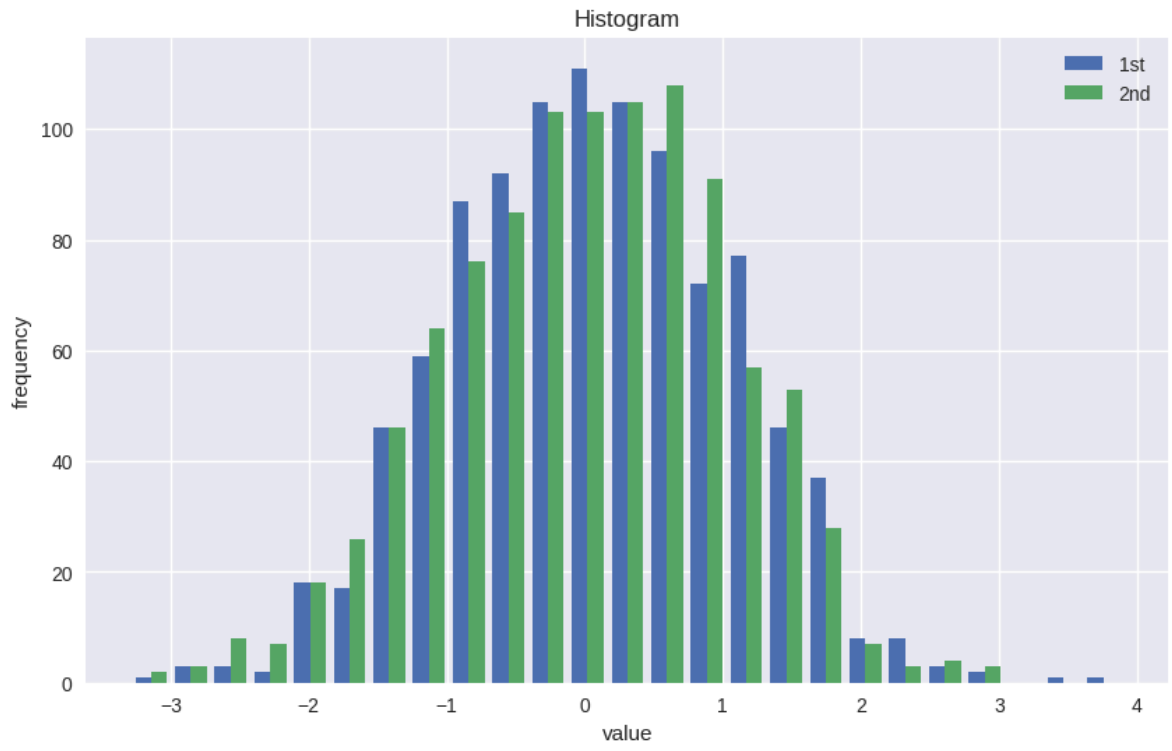
```
<matplotlib.colorbar.Colorbar at 0x789cefa24f80>
```



Another type of plot, the histogram, is also often used in the context of financial returns. The Figure below puts the frequency values of the two data sets next to each other in the same plot.

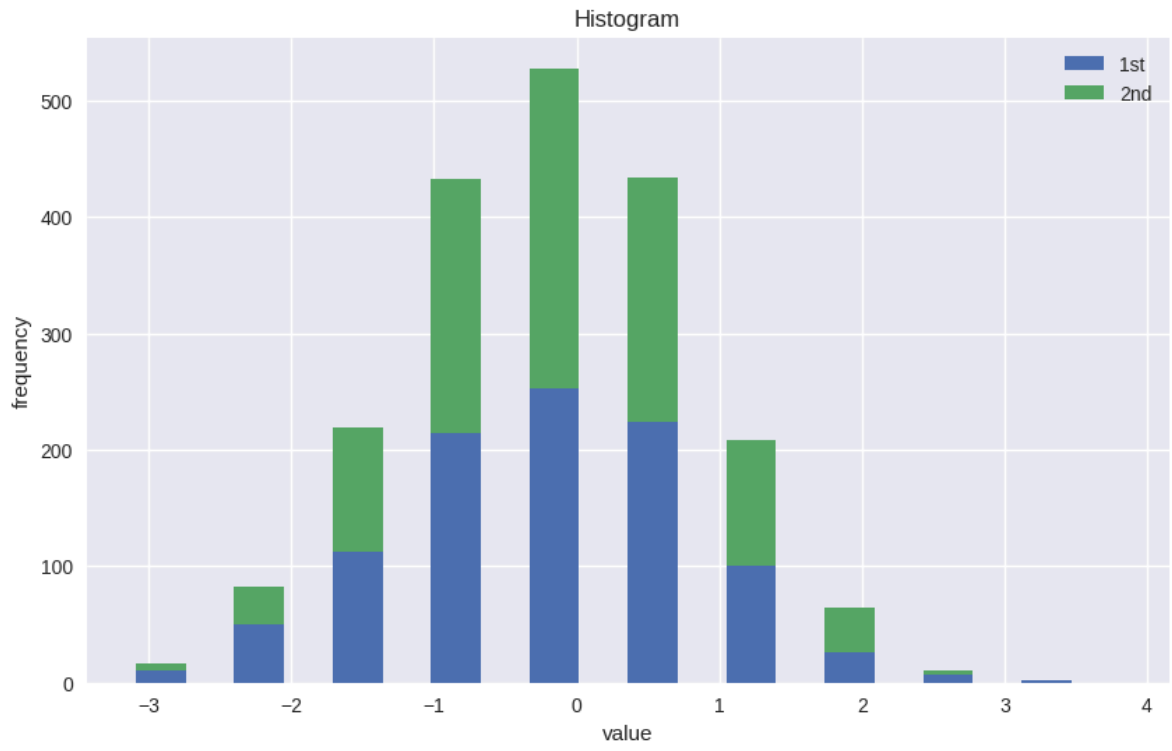
Documentation: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html

```
import matplotlib.pyplot as plt
import numpy

y = np.random.standard_normal((1000, 2))

plt.figure(figsize=(10, 6))
#plt.hist(y, label=['1st', '2nd'])
plt.hist(y, label=['1st', '2nd'], bins=25)
plt.legend(loc=0)
plt.xlabel('value')
plt.ylabel('frequency')
plt.title('Histogram')
plt.show()
```



The figure below shows a similar plot; this time, the data of the two data sets is stacked in the histogram:

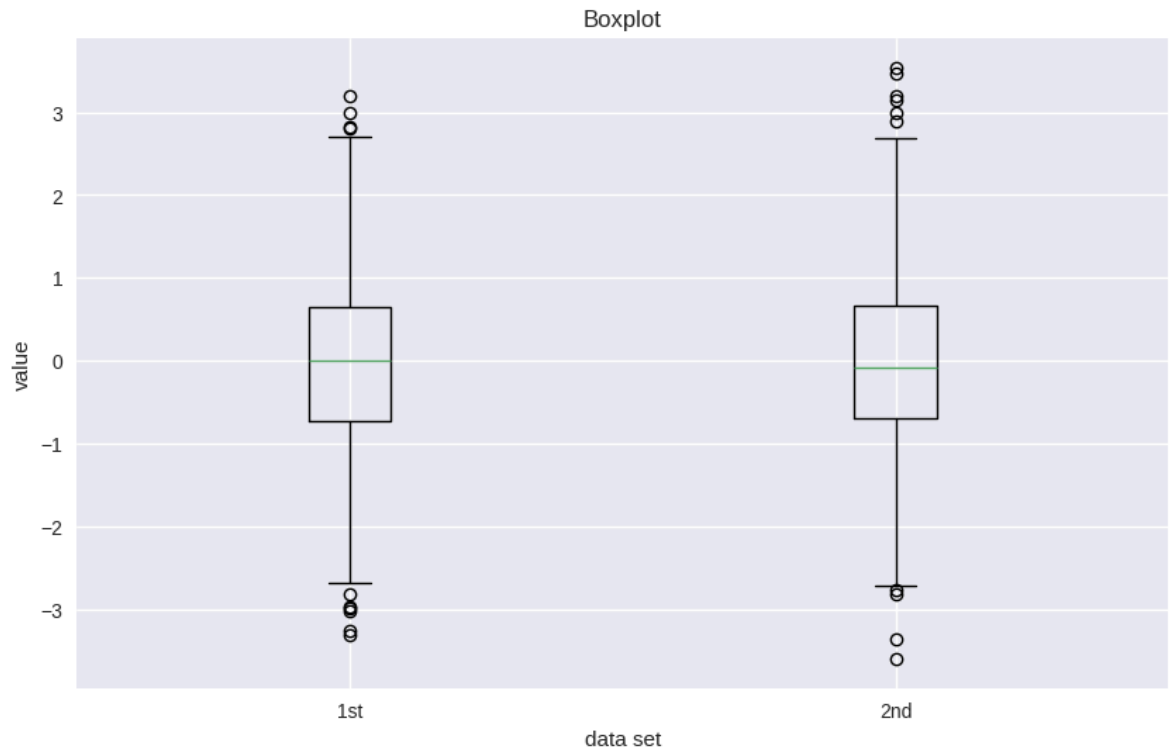
```
import matplotlib.pyplot as plt
import numpy as np

y = np.random.standard_normal((1000, 2))

plt.figure(figsize=(10, 6))
#plt.hist(y, label=['1st', '2nd'], stacked=True)
plt.hist(y, label=['1st', '2nd'], stacked=True, width=0.5)
plt.legend(loc=0)
plt.xlabel('value')
plt.ylabel('frequency')
plt.title('Histogram')
plt.show()
```



Another useful plot type is the boxplot. Similar to the histogram, the boxplot allows both a concise overview of the characteristics of a data set and easy comparison of multiple data sets. The figure below shows such a plot for our data sets.

Documentation:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html

```
import matplotlib.pyplot as plt
import numpy as np

y = np.random.standard_normal((1000, 2))

plt.figure(figsize=(10, 6))
plt.boxplot(y, tick_labels=['1st', '2nd'])
plt.xlabel('data set')
plt.ylabel('value')
plt.title('Boxplot')
plt.show()
```



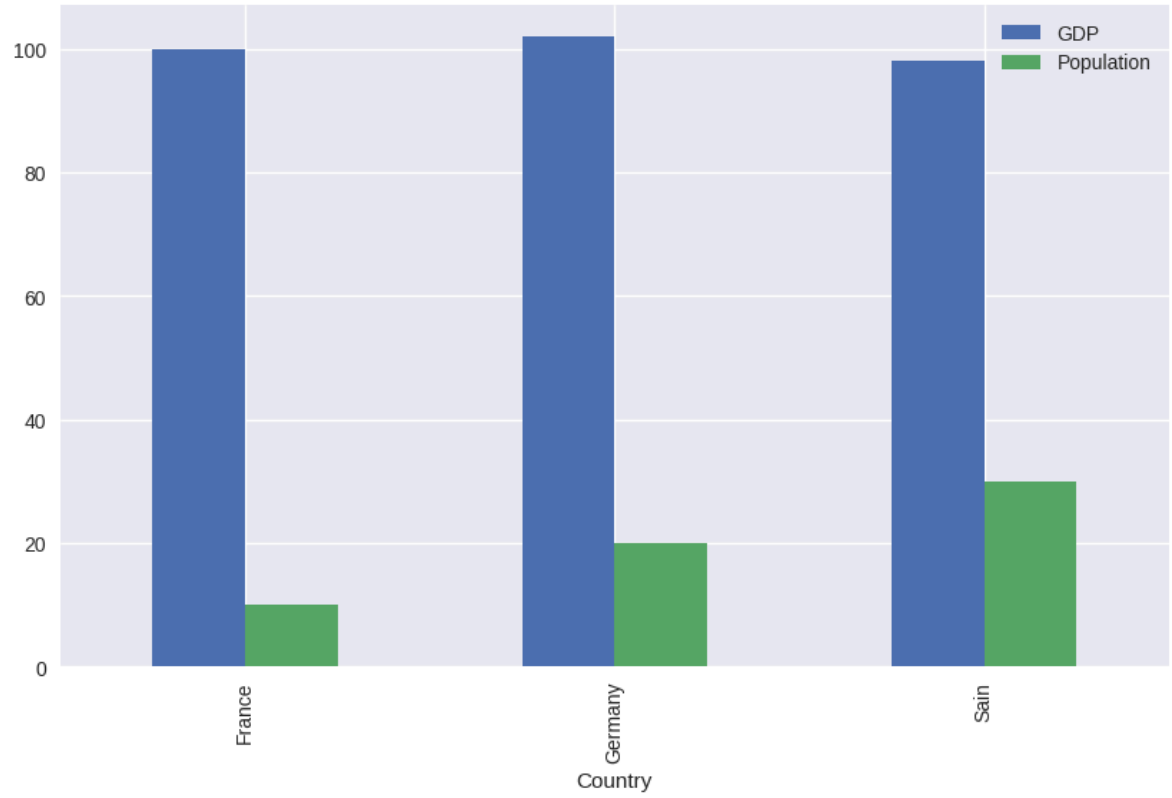
Creating a 2d chart based on the data from an Excel file.

```
import pandas as pd
df = pd.read_excel('/content/Dataset - Countries.xlsx')
df
```

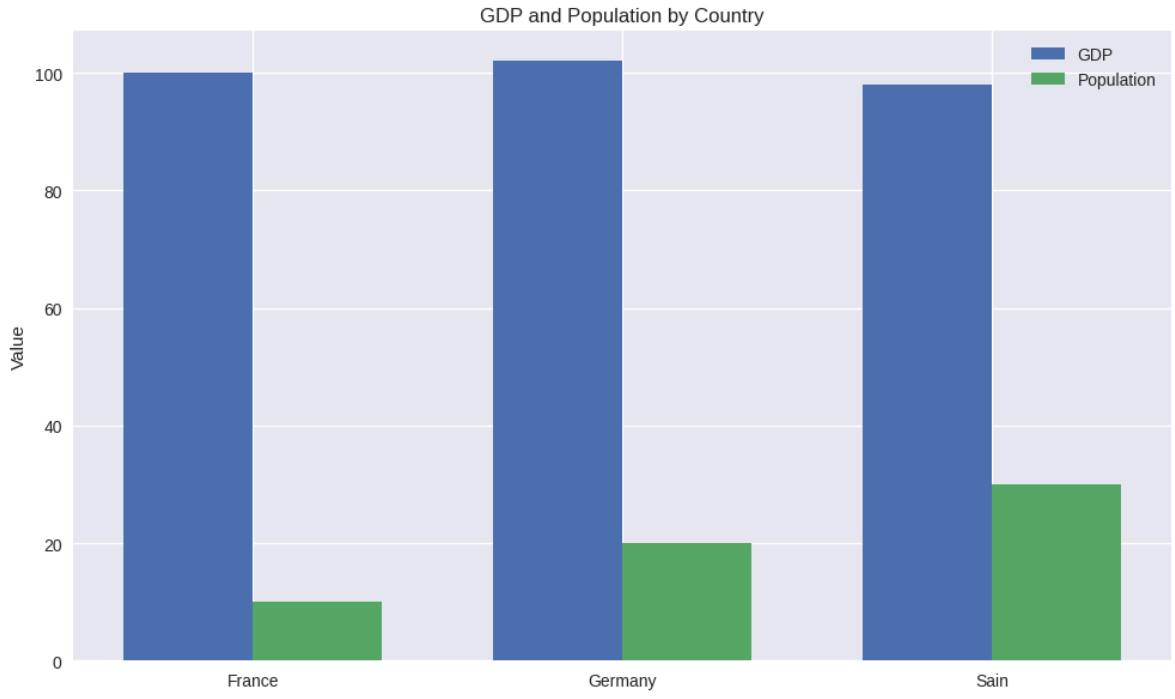
	Country	GDP	Population
0	France	100	10
1	Germany	102	20
2	Sain	98	30

```
# df.plot(x='Country', y='GDP', kind = 'bar', figsize=(10, 6))
df.plot(x='Country', y=['GDP', 'Population'], kind = 'bar', figsize=(10, 6))
```

```
<Axes: xlabel='Country'>
```




```
# Rewrite the code using matplotlib directly instead of the `plot` method fr

import matplotlib.pyplot as plt
import numpy as np

x = np.arange(len(df['Country'])) # the label locations
width = 0.35  # the width of the bars

plt.figure(figsize=(10, 6))

rects1 = plt.bar(x - width/2, df['GDP'], width, label='GDP')
rects2 = plt.bar(x + width/2, df['Population'], width, label='Population')

# Add some text for labels, title and custom x-axis tick labels, etc.
plt.ylabel('Value')
plt.title('GDP and Population by Country')
plt.xticks(x, df['Country'])
plt.legend()

plt.tight_layout()
plt.show()
```



3D

```
import matplotlib.pyplot as plt
import numpy as np

# Generate some random data for the 3D plot
x = np.random.rand(100)
y = np.random.rand(100)
z = np.random.rand(100)

# Create the 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(x, y, z) # same color
ax.scatter(x, y, z, c='r') # specify a color
ax.scatter(x, y, z, c='r', marker='D') # specify a color and a different sh
#ax.scatter(x, y, z, c=z, cmap='viridis') # use a color map

ax.set_xlabel('X Label')
```

```
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.show()
#
```