Pandas

Pandas is a Python library used for working with data sets.
It has functions for analyzing, cleaning, exploring, and manipulating data.
The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

What Can Pandas Do?

Pandas gives you answers about the data. Like:

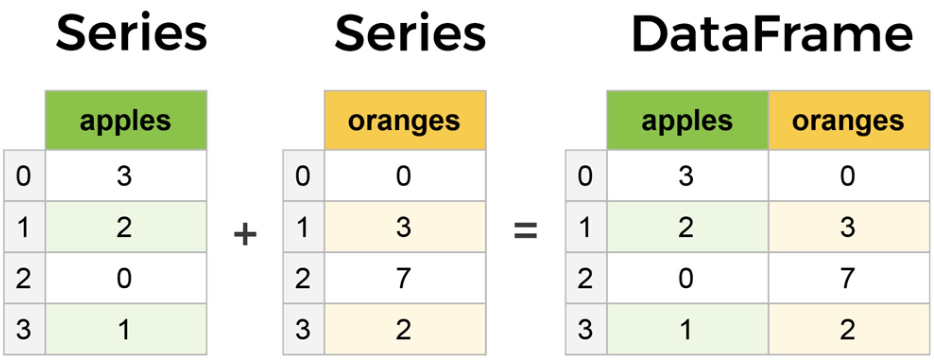
- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called **cleaning the data**.

Data Structures:

Pandas introduces two primary data structures:

- Series: A one-dimensional labeled array.
- DataFrame: A two-dimensional, table-like structure with labeled rows and columns, similar to a spreadsheet or SQL table, and the most commonly used structure.



Series

	apples
0	3
1	2
2	0
3	1

+

Series

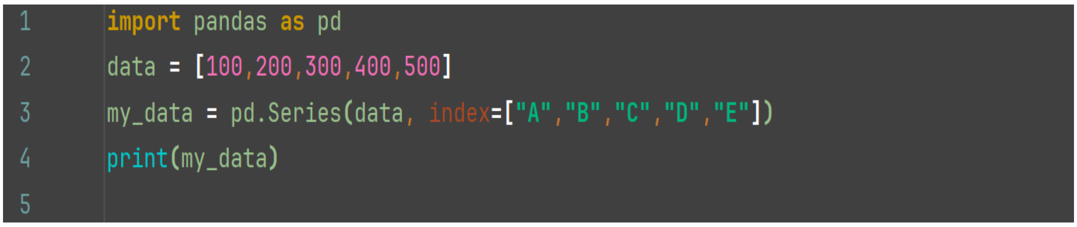
	oranges
0	0
1	3
2	7
3	2

=

DataFrame

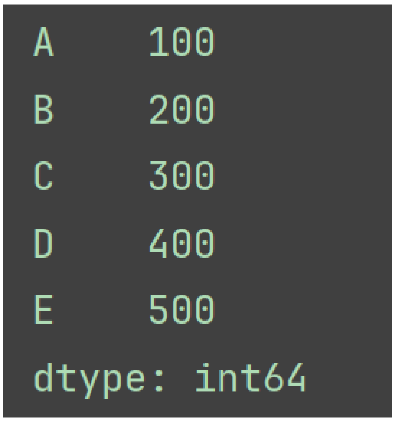
	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Code snippet(Series):



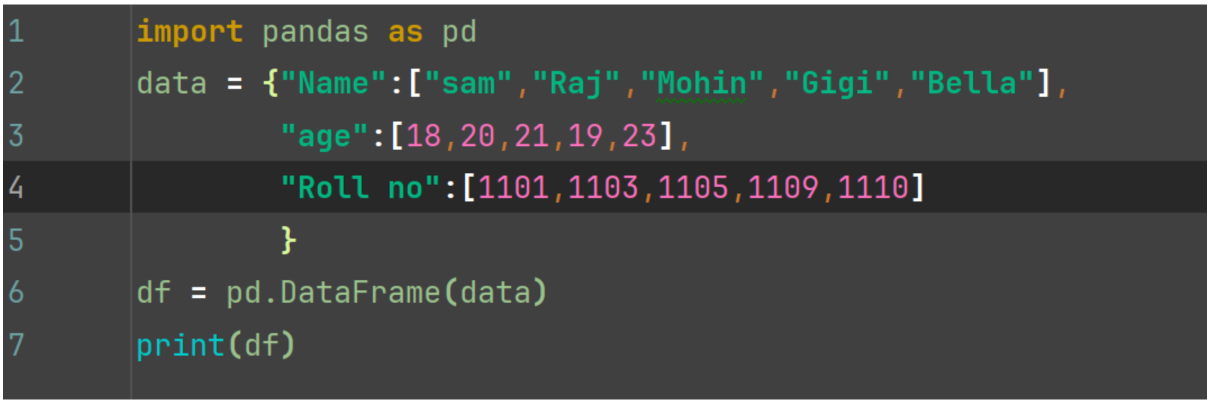
```
import pandas as pd
data = [100,200,300,400,500]
my_data = pd.Series(data, index=["A","B","C","D","E"])
print(my_data)

```

Output:



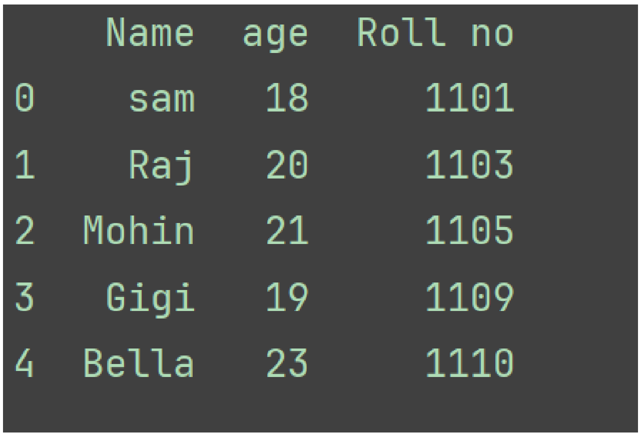
```
A    100
B    200
C    300
D    400
E    500
dtype: int64
```

Code snippet(DataFrame):



```
import pandas as pd
data = {"Name":["sam","Raj","Mohin","Gigi","Bella"],
        "age":[18,20,21,19,23],
        "Roll no":[1101,1103,1105,1109,1110]
        }
df = pd.DataFrame(data)
print(df)
```

Output:



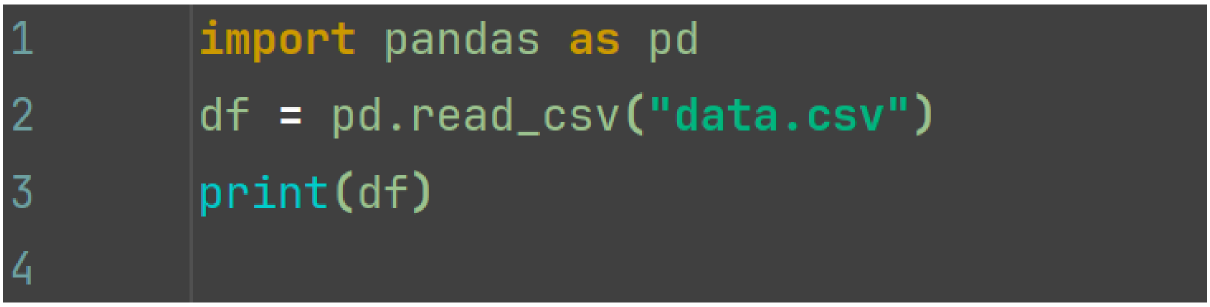
```
    Name  age  Roll no
0    sam   18     1101
1    Raj   20     1103
2  Mohin   21     1105
3   Gigi   19     1109
4  Bella   23     1110
```

Importing files in Pandas:

A simple way to store big data sets is to use CSV files (comma separated files).

CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

In our examples we will be using a CSV file called 'data.csv'.



```
import pandas as pd
df = pd.read_csv("data.csv")
print(df)

```

If you have a large DataFrame with many rows, Pandas will only return the first 5 rows, and the last 5 rows:

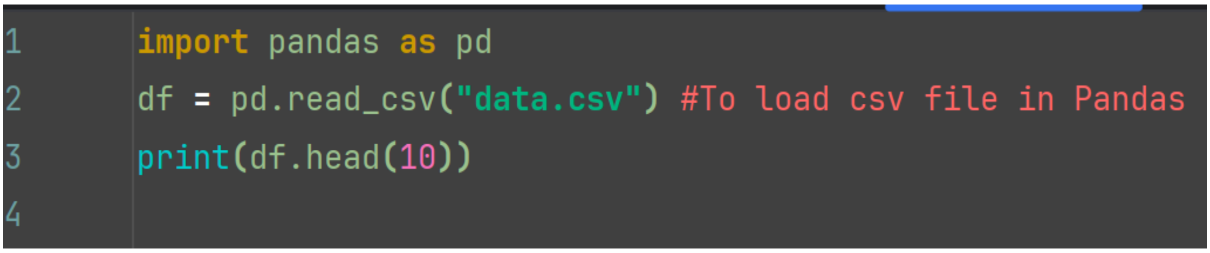
use to_string() to print the entire DataFrame.

Analyzing Data in Pandas:

head()

One of the most used method for getting a quick overview of the DataFrame, is the head() method.

The head() method returns the headers and a specified number of rows, starting from the top.

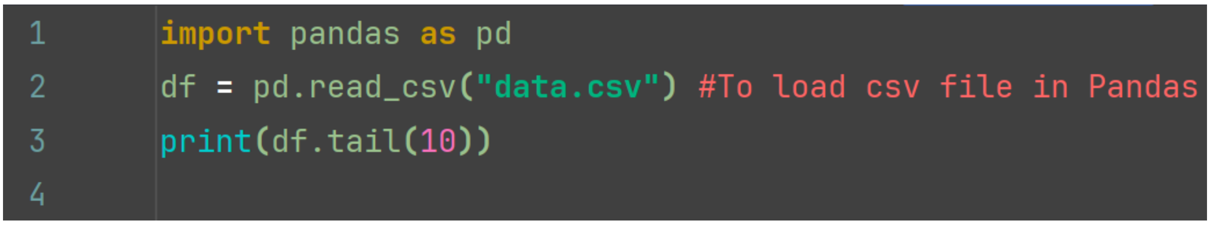


```
import pandas as pd
df = pd.read_csv("data.csv") #To load csv file in Pandas
print(df.head(10))

```

It will only print first 10 rows of DataFrame.

tail()

The tail() method returns the headers and a specified number of rows, starting from the bottom.

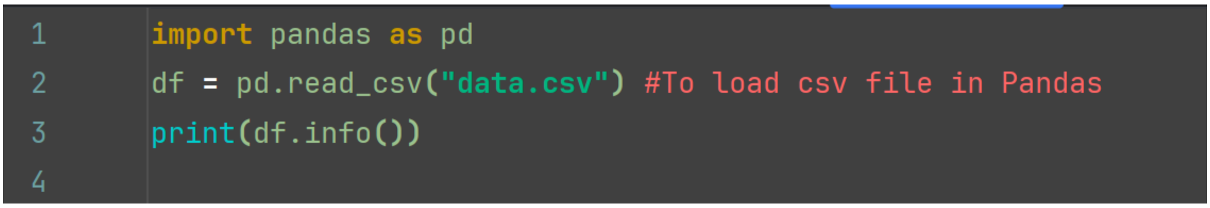


```python
import pandas as pd
df = pd.read_csv("data.csv") #To load csv file in Pandas
print(df.tail(10))

```

Results last 10 rows of DataFrame.

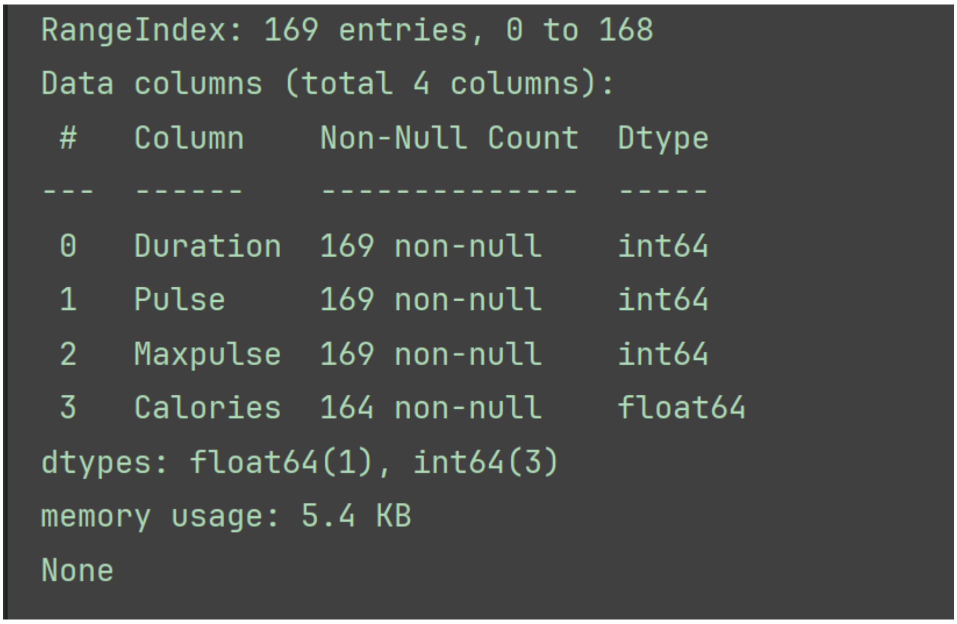
info()

The DataFrames object has a method called info(), that gives you more information about the data set.



```python
import pandas as pd
df = pd.read_csv("data.csv") #To load csv file in Pandas
print(df.info())

```

Output:



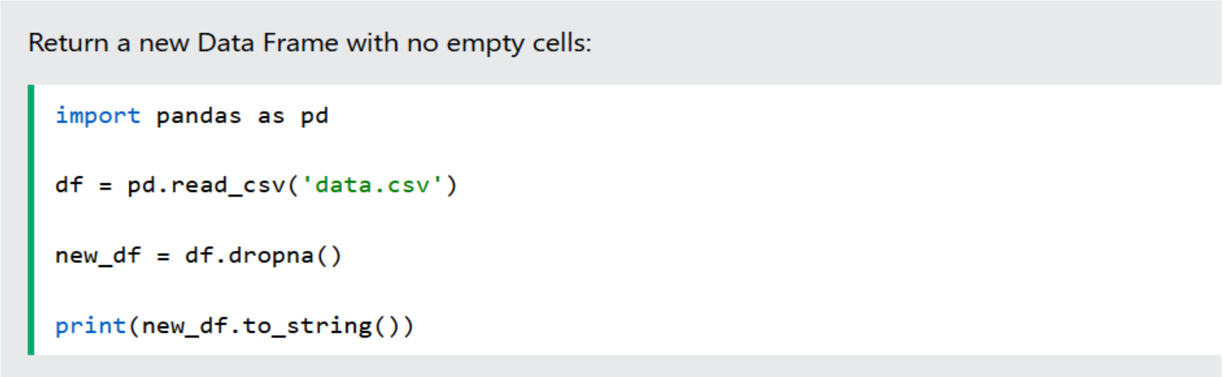
```
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

Cleaning Data:

1.Cleaning Empty Cells

By Removing rows:

One way to deal with empty cells is to remove rows that contain empty cells. This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

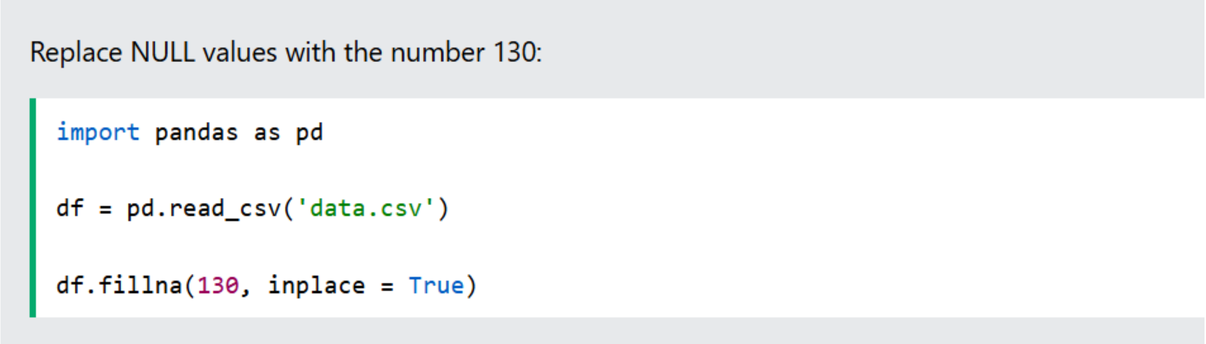


Return a new Data Frame with no empty cells:

```
import pandas as pd

df = pd.read_csv('data.csv')

new_df = df.dropna()

print(new_df.to_string())
```

By Replacing empty value:

Another way of dealing with empty cells is to insert a *new* value instead. This way you do not have to delete entire rows just because of some empty cells.



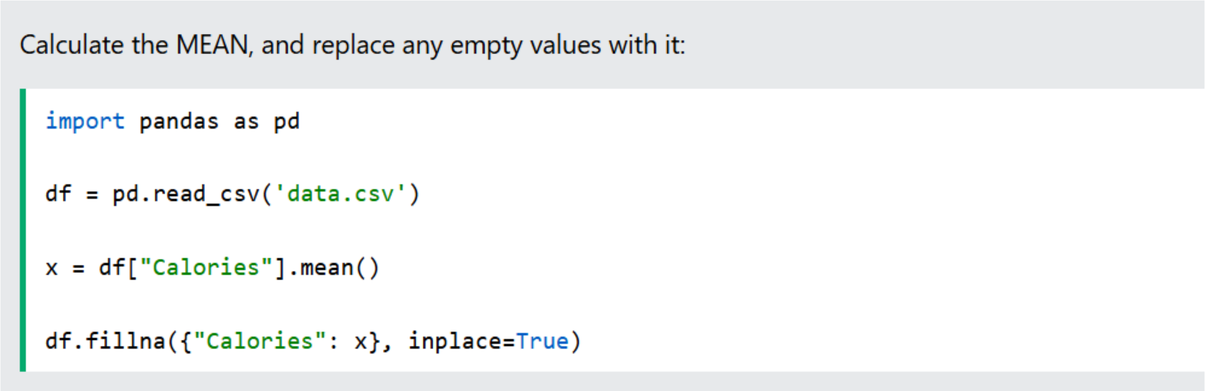
Replace NULL values with the number 130:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.fillna(130, inplace = True)
```

By Replacing using mean, mode or median:

A common way to replace empty cells, is to calculate the mean, median or mode value of the column.

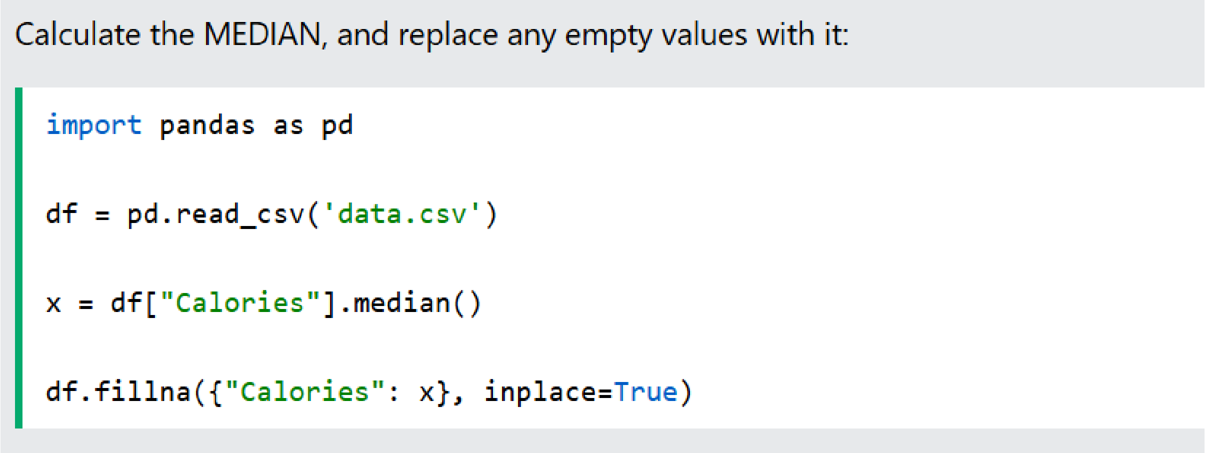
- Calculating MEAN



Calculate the MEAN, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mean()

df.fillna({"Calories": x}, inplace=True)
```

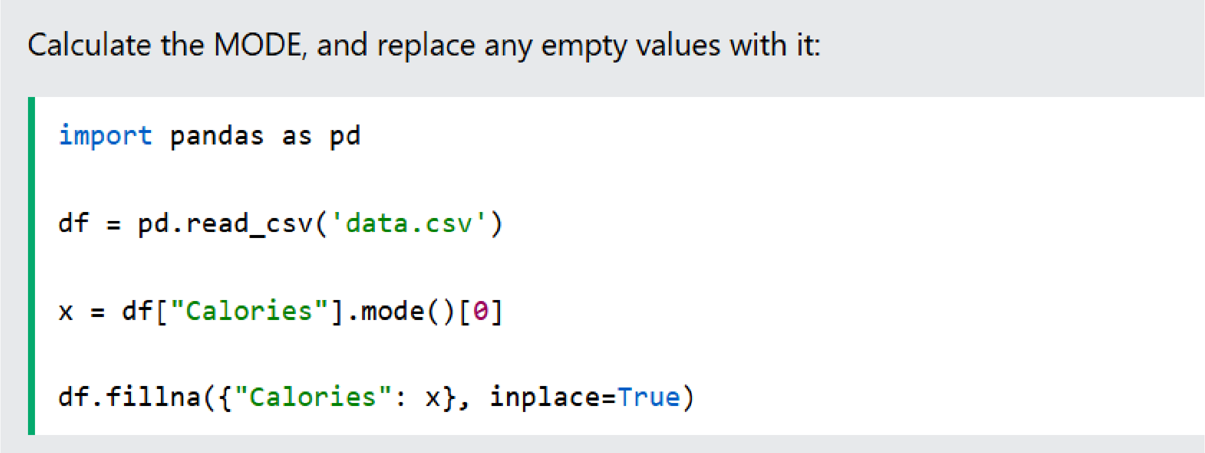
- Calculating MEDIAN



Calculate the MEDIAN, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].median()

df.fillna({"Calories": x}, inplace=True)
```

- Calculating MODE



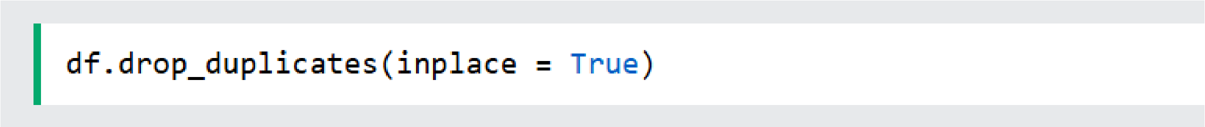
Calculate the MODE, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mode()[0]

df.fillna({"Calories": x}, inplace=True)
```

2.Removing Duplicates

To discover duplicates, we can use the duplicated() method.
The duplicated() method returns a Boolean values for each row.

To remove duplicates, use the **drop_duplicates()** method.



```
df.drop_duplicates(inplace = True)
```

PANDA’S PLOTING

Pandas uses the plot() method to create diagrams. We can use **Pyplot**, a **submodule** of the **Matplotlib** library to visualize the diagram on the screen.

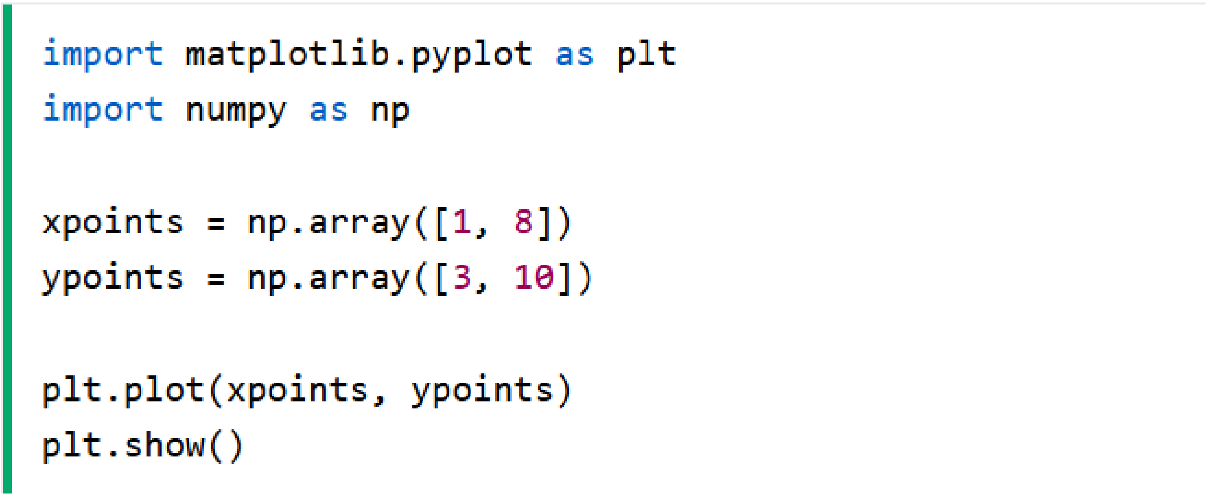
MATPLOTLIB

Matplotlib is a low level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter.

Plotting x and y points

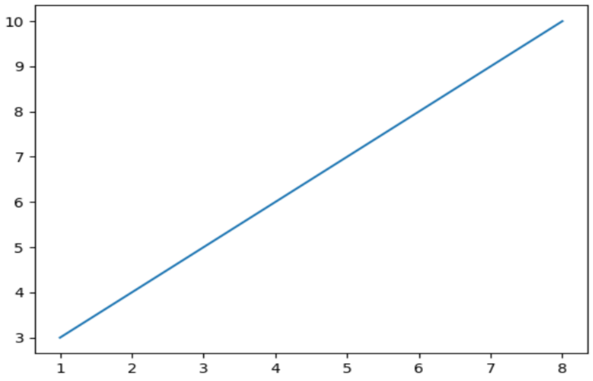
The plot() function is used to draw points (markers) in a diagram. By default, the plot() function draws a line from point to point. The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the **x-axis**.
Parameter 2 is an array containing the points on the **y-axis**.



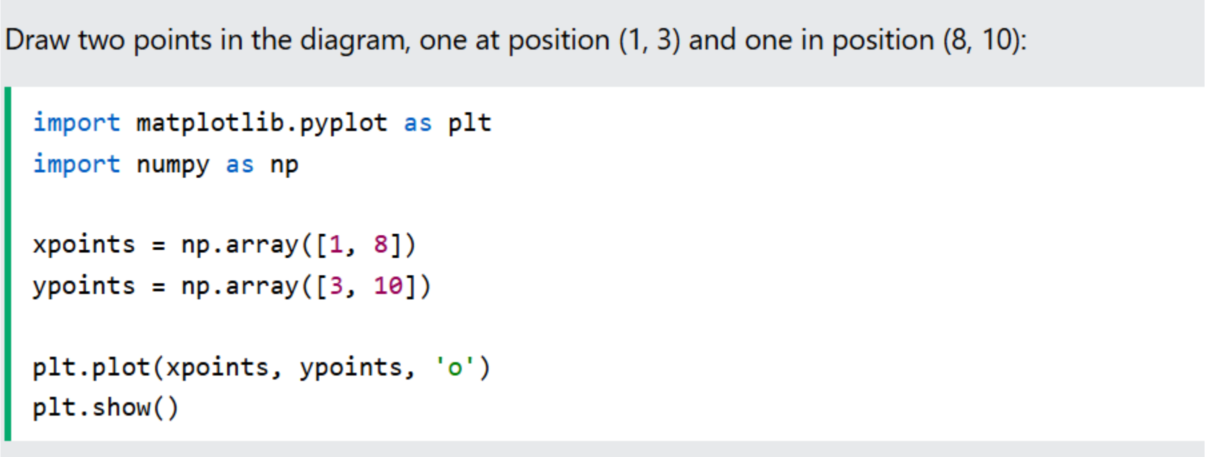
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

Output:



Plotting Without Line

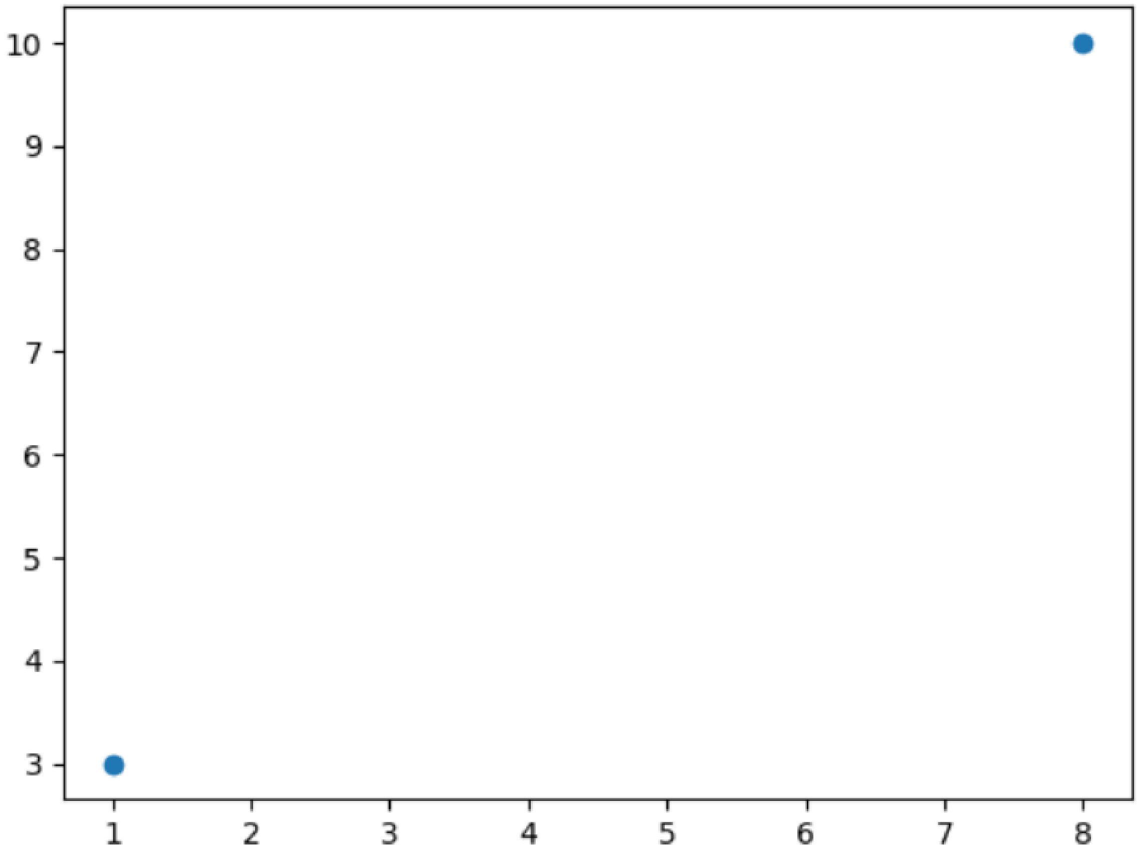
To plot only the markers, you can use *shortcut string notation* parameter 'o', which means 'rings'.



Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

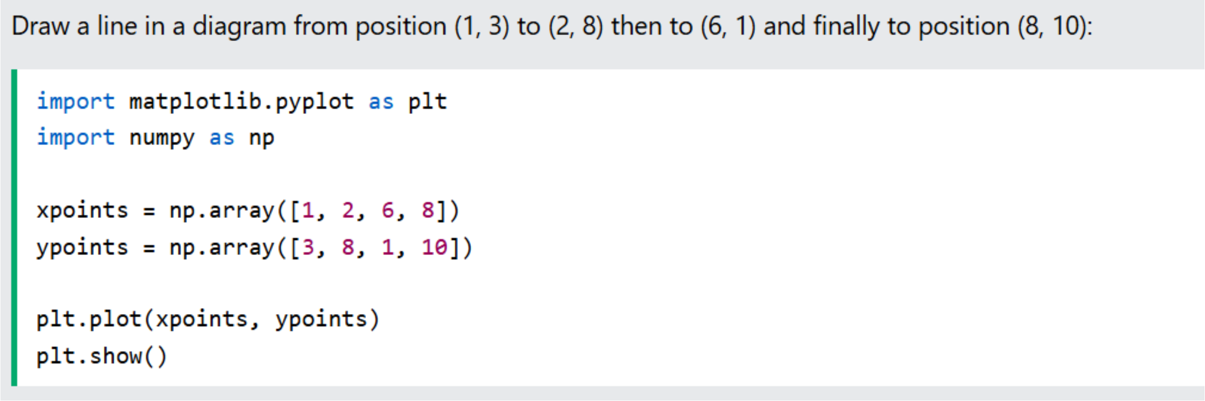
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints, 'o')
plt.show()
```

Output:



Multiple Points

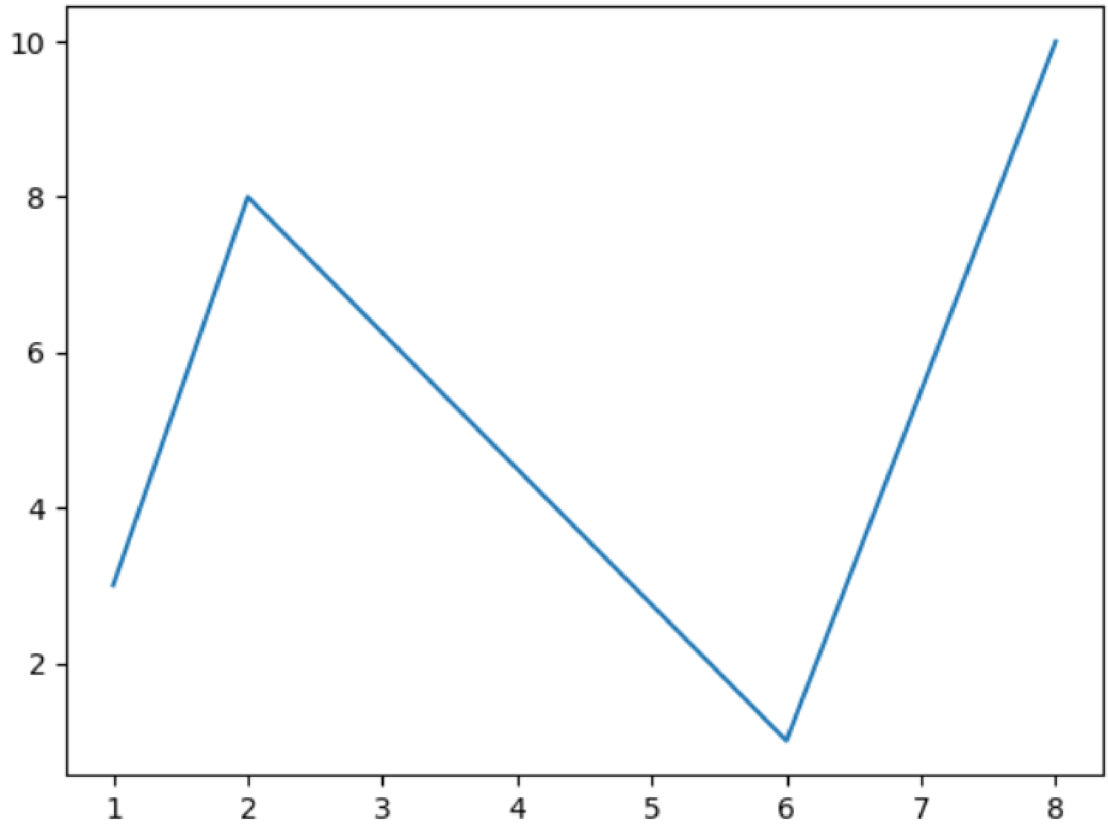
You can plot as many points as you like, just make sure you have the same number of points in both axis.



Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):

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

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

Output:

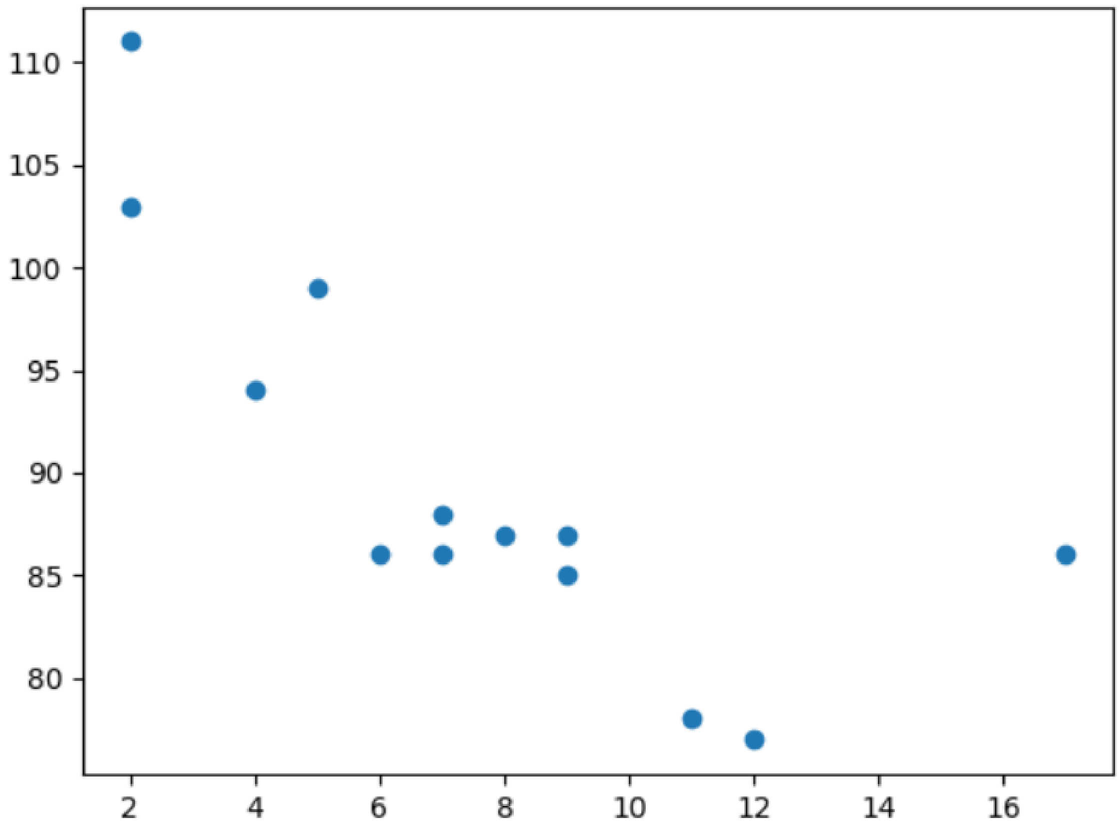


Scatter

With Pyplot, you can use the scatter() function to draw a scatter plot. The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

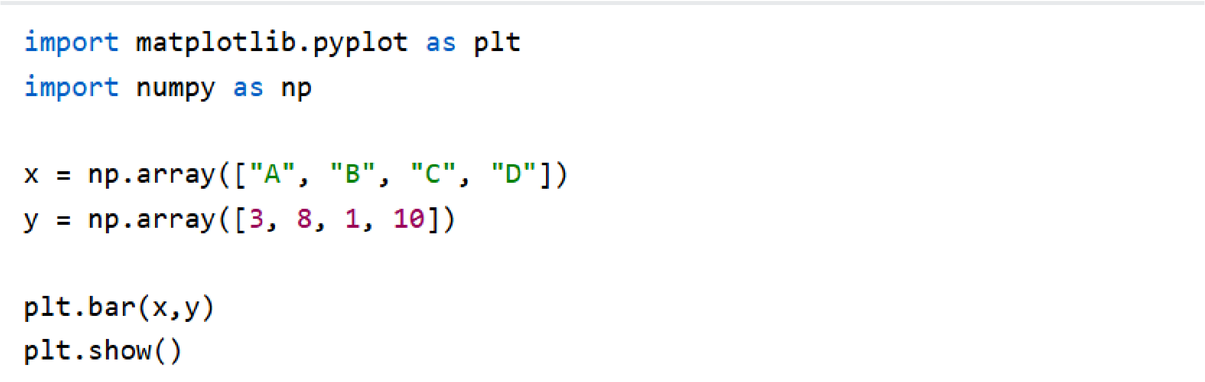
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

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

Output:



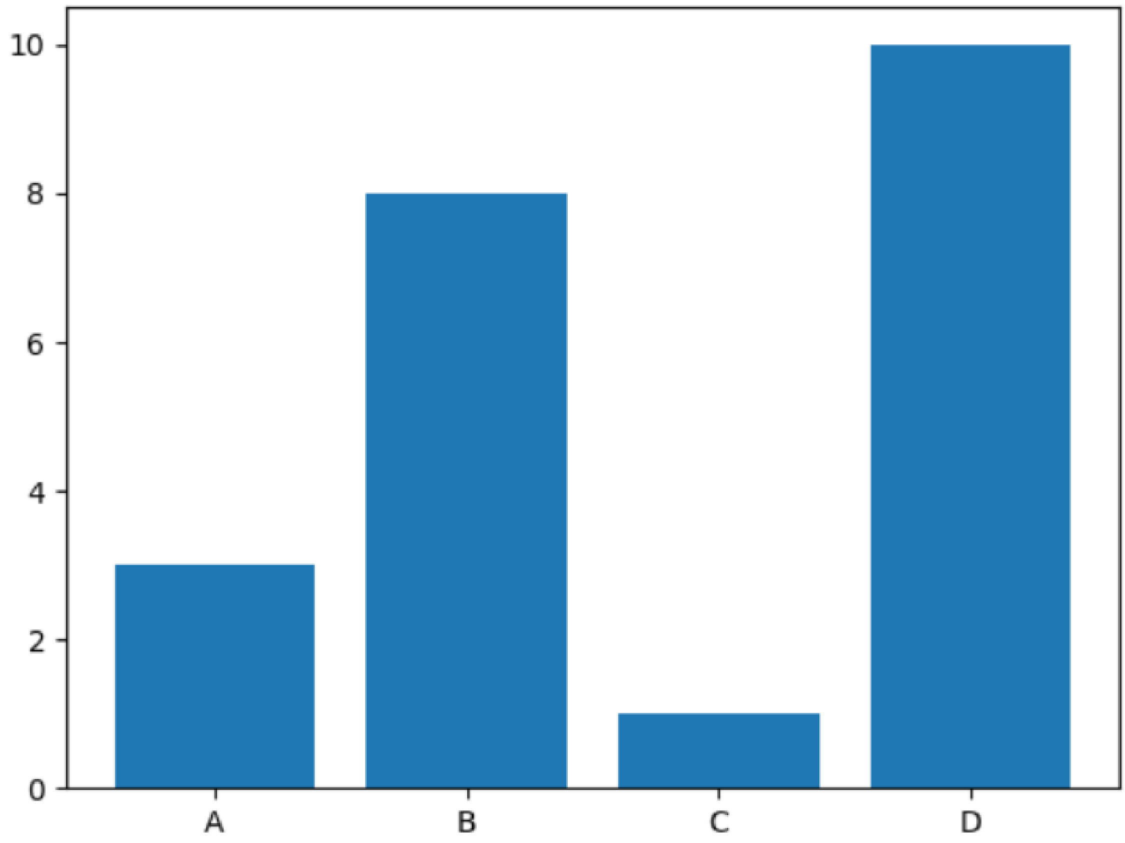
Bars

Using Matplotlib, you can use bar() function to draw bar.



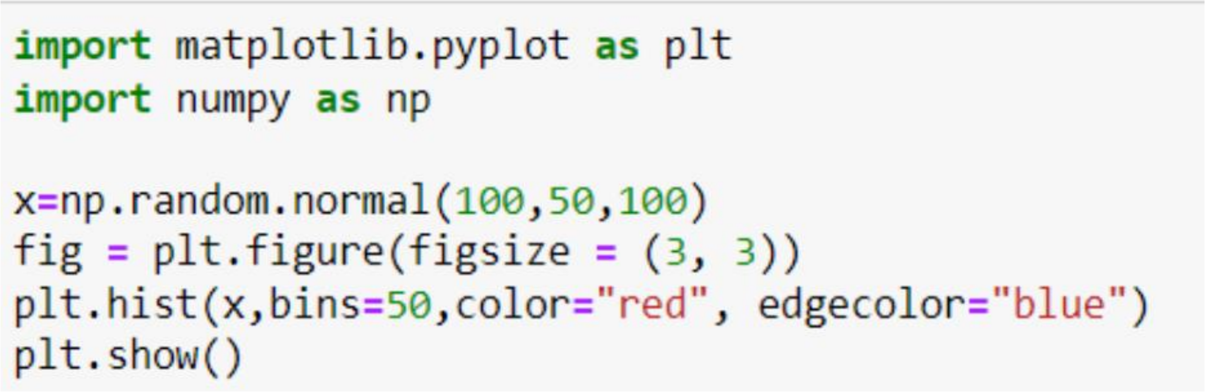
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

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

Output:



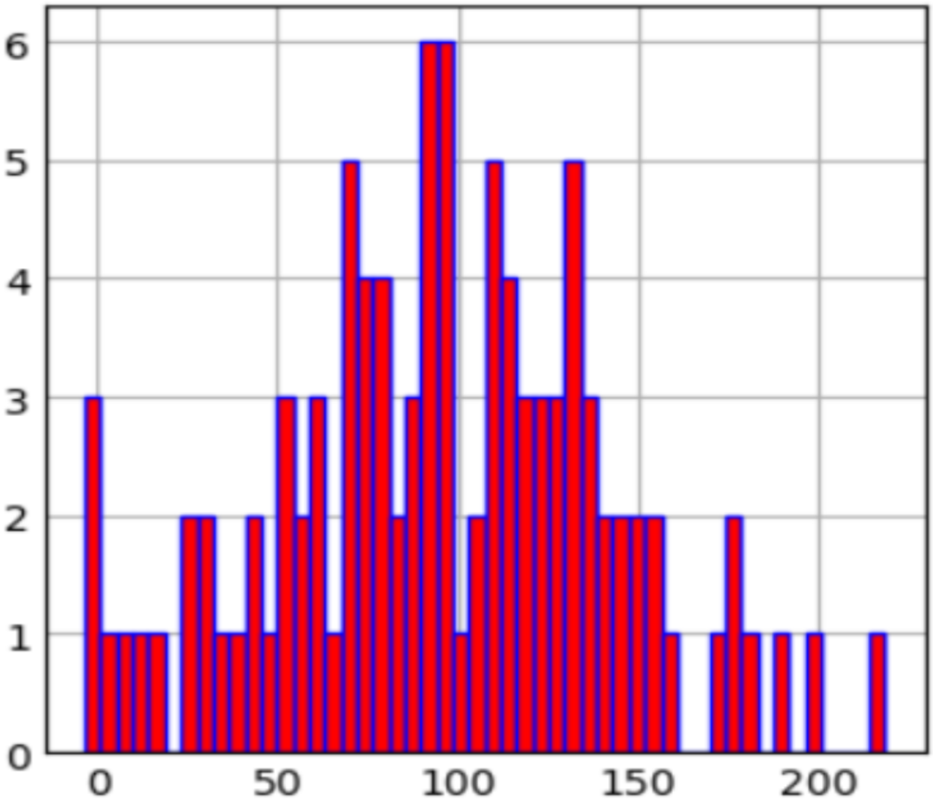
Histograms

A histogram is a graph showing *frequency* distributions. It is a graph showing the number of observations within each given interval.



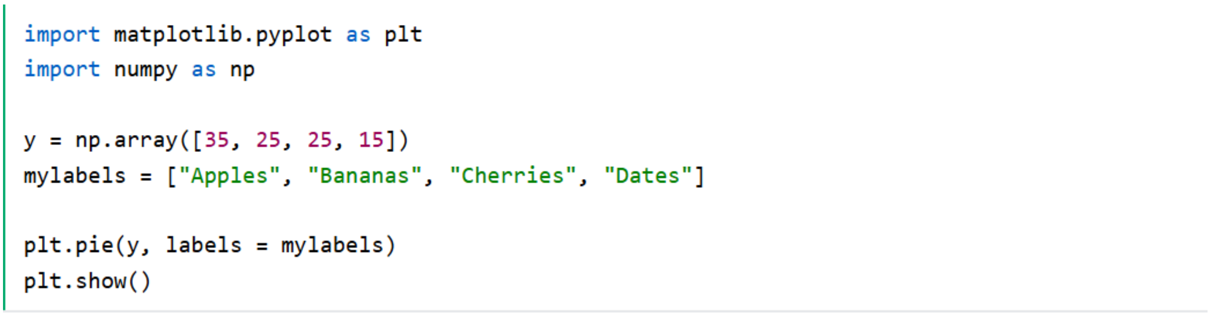
```
import matplotlib.pyplot as plt
import numpy as np

x=np.random.normal(100,50,100)
fig = plt.figure(figsize = (3, 3))
plt.hist(x,bins=50,color="red", edgecolor="blue")
plt.show()
```

Output:



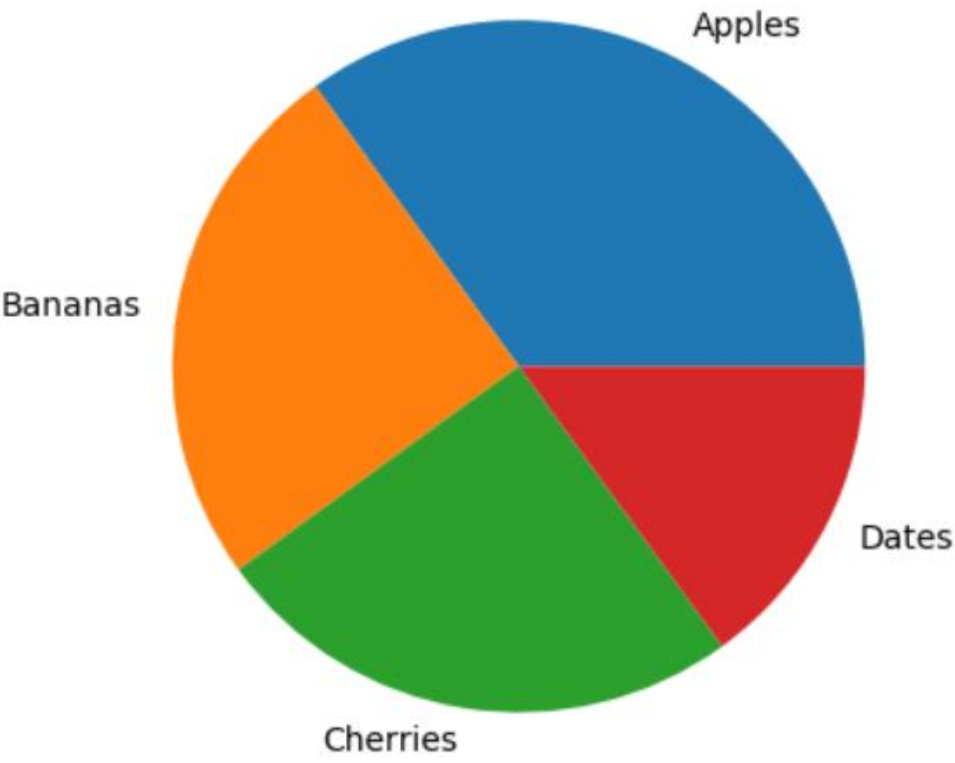
Piechart

Pie charts are used to plot data of same kind which means the same series of data where the different elements are divide based on their percentage.



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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```

Output:



Comparison: Pandas vs Matplotlib

Feature	Pandas	Matplotlib
Primary Purpose	Data analysis & manipulation	Data visualization
Ease of Use	High-level, beginner-friendly	More complex, requires detailed setup
Visualization	Quick, simple plots (df.plot())	Full control, advanced customization
Data Handling	Reads/writes multiple formats, handles missing data	Limited to plotting, not data manipulation
Best For	Fast exploration of datasets	Creating polished, detailed figures
Integration	Uses Matplotlib internally for plotting	Can be combined with Pandas, NumPy