python

Introduction to Python

- Python is a high-level, interpreted programming language that is widely used for various purposes, including web development, scientific computing, data analysis, artificial intelligence, and machine learning. It was first released in 1991 and has since become one of the most popular programming languages in the world.
- Python is a versatile and powerful programming language that is easy to learn and use.
- Its simplicity & readability
- Python has a vast ecosystem of libraries and tools that make it suitable for various applications, including NumPy and Pandas for data analysis, Flask and Django for web development, PyTorch and TensorFlow for machine learning, and many more.

Some of the key features of Python include:

1. Easy to learn and use
2. Interpreted, meaning that code is executed line by line, without the need for compilation
3. Cross-platform, meaning that Python code can run on various operating systems, including Windows, macOS, and Linux
4. Object-oriented, allowing developers to create complex and modular applications
5. High-level, providing abstractions that simplify programming tasks and make it easier to write and read code
6. Dynamically typed, meaning that data types are determined at runtime
7. Has a large and active community, providing support and resources for developers of all levels

Python Language Libraries for DE

Python has many popular libraries for data engineering tasks. Here are some of the most commonly used libraries for data engineering in Python:

- **pandas:** pandas is a powerful library for data manipulation and analysis. It is commonly used for data cleaning, transformation, and preparation. It provides tools for working with tabular data and supports many data formats, including CSV, Excel, SQL databases, and JSON.
- **NumPy:** NumPy is a library for numerical computing in Python. It provides a multidimensional array object, which is useful for handling large amounts of numerical data. NumPy is commonly used for scientific computing, machine learning, and data analysis.
- **Matplotlib:** Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Python Language Libraries for DE

- **PyTorch:** PyTorch is the largest machine learning library that optimizes tensor computations. It has rich APIs to perform tensor computations with strong GPU acceleration. It also helps to solve application issues related to neural networks.
- **PyArrow:** PyArrow is a library for working with Arrow data structures in Python. Arrow is an in-memory columnar data format that is designed for efficient data processing across multiple programming languages and systems. PyArrow provides tools for converting data between different formats, including Pandas DataFrames and Apache Arrow tables.
- **SciPy:** The name “SciPy” stands for “Scientific Python”. It is an open-source library used for high-level scientific computations. This library is built over an extension of Numpy. It works with Numpy to handle complex computations. While Numpy allows sorting and indexing of array data, the numerical data code is stored in SciPy. It is also widely used by application developers and engineers.
- These libraries are just a few examples of the many tools available for data engineering in Python. Depending on your specific needs and goals, you may find other libraries that are more suitable for your projects.

Python Language Libraries for DE

- **Apache Spark**: Apache Spark is an open-source distributed computing system that provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. It supports several programming languages, including Python, Java, and Scala. Spark is commonly used for large-scale data processing, including data engineering tasks like data transformation and cleaning.
- **Dask**: Dask is a library for parallel computing in Python. It provides a parallelized DataFrame object that is similar to pandas, but can handle much larger datasets by dividing them into smaller chunks and processing them in parallel.
- **PySpark**: PySpark is the Python API for Apache Spark. It provides a way to program Spark using Python code, allowing Python developers to take advantage of Spark's distributed computing capabilities.

What is Numpy

- **NumPy** is a Python library that provides support for multi-dimensional arrays and matrices, along with a large number of mathematical functions to manipulate them. **NumPy** is stands for "**Numerical Python**".
- It is an open source project and you can use it freely.

Why Use NumPy?

- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
- Arrays are very frequently used in data science, where speed and resources are very important.

Numpy

Here are some basic examples of using NumPy in Python:

- **Installing NumPy**

Before using NumPy, you need to install it. You can do this by running the following command in your terminal or command prompt:

```
pip install numpy
```

- **Importing NumPy**

Once NumPy is installed, you can import it into your Python code using the following command:

```python
import numpy as np
```

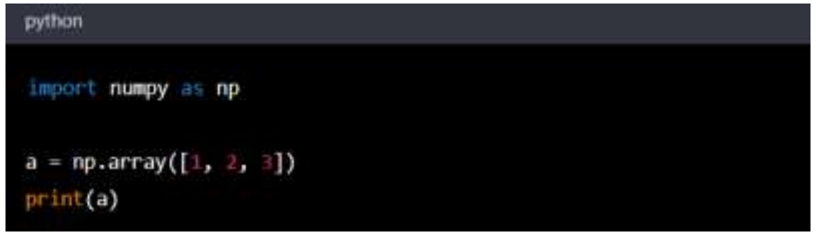
- This imports NumPy and gives it an alias "np" which is commonly used in NumPy code

Numpy

ARRAY

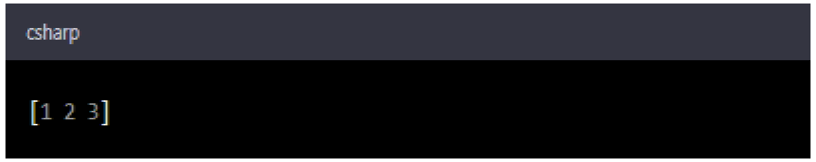
Create a NumPy **Array** by passing a list or tuple to the ‘**np.array()**‘ function.

For example:



```python
import numpy as np

a = np.array([1, 2, 3])
print(a)
```

output

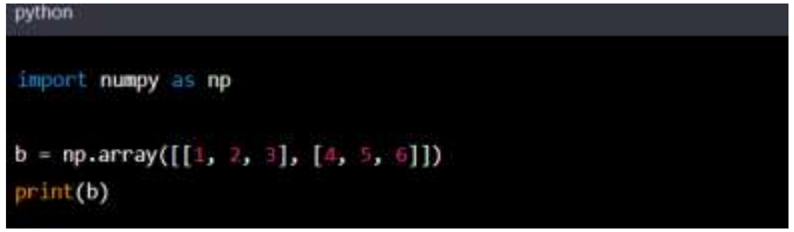


```
[1 2 3]
```

Multi-dimensional array

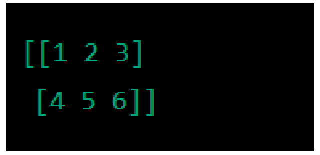
- Create a **multi-dimensional array** by passing a nested list or tuple.

example:



```python
import numpy as np

b = np.array([[1, 2, 3], [4, 5, 6]])
print(b)
```

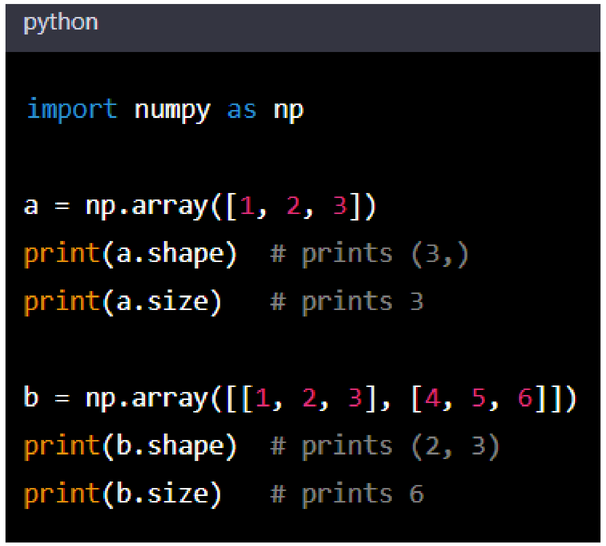
output



```
[[1 2 3]
 [4 5 6]]
```

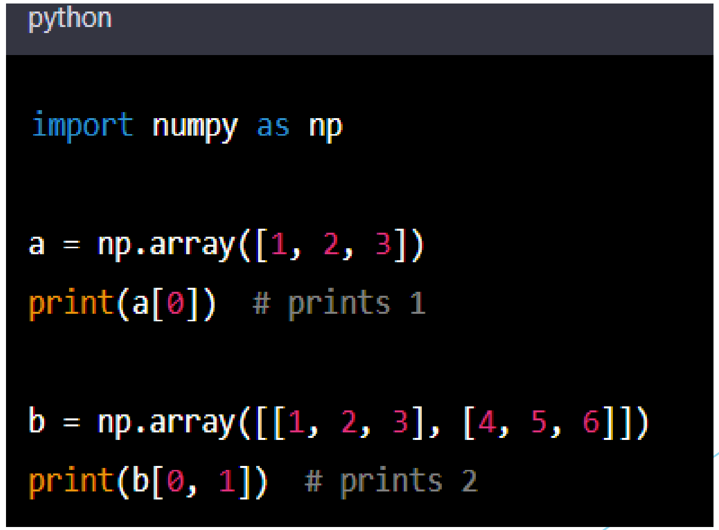
Numpy

- **Array attributes**
- You can access various attributes of a NumPy array, such as its shape and size, using the following properties:



```python
import numpy as np

a = np.array([1, 2, 3])
print(a.shape)  # prints (3,)
print(a.size)   # prints 3

b = np.array([[1, 2, 3], [4, 5, 6]])
print(b.shape)  # prints (2, 3)
print(b.size)   # prints 6
```

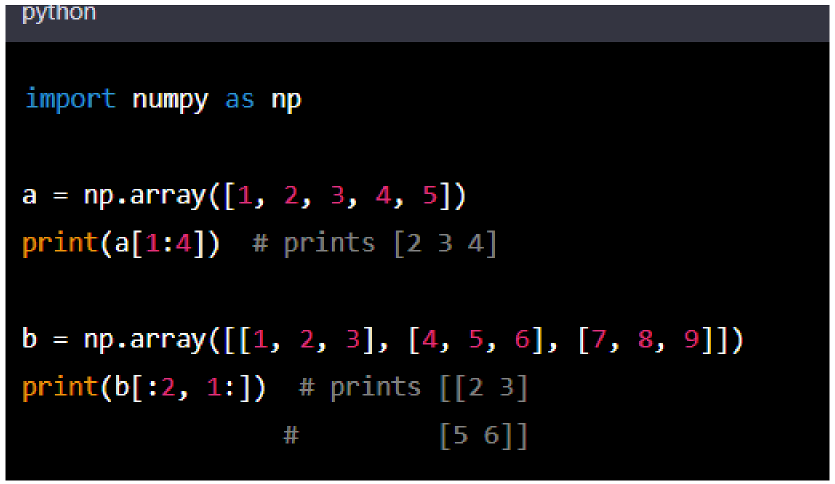
- **Array indexing**
- You can access individual elements of a NumPy array using indexing, just like you would with a list or tuple



```python
import numpy as np

a = np.array([1, 2, 3])
print(a[0])  # prints 1

b = np.array([[1, 2, 3], [4, 5, 6]])
print(b[0, 1])  # prints 2
```

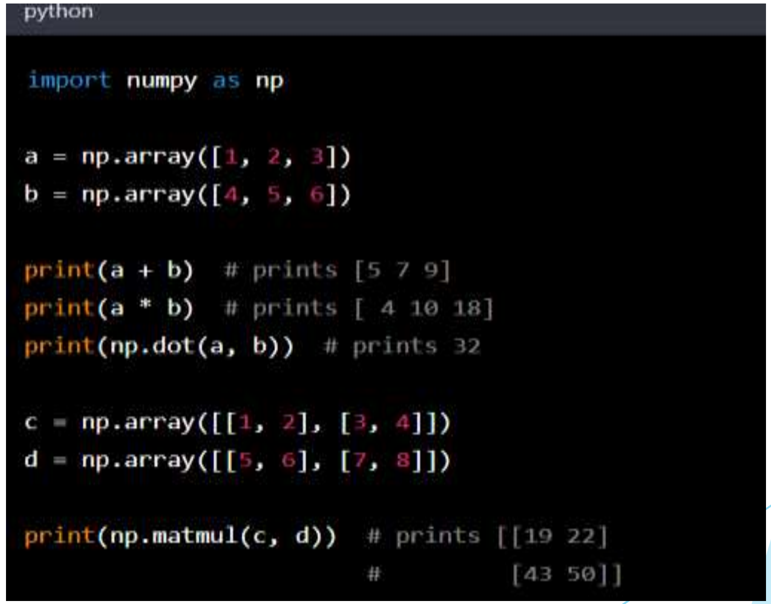
Numpy

- **Array Slicing**
- You can also use slicing to access a range of elements in a NumPy array:



```
python

import numpy as np

a = np.array([1, 2, 3, 4, 5])
print(a[1:4])  # prints [2 3 4]

b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(b[:2, 1:])  # prints [[2 3]
                  #         [5 6]]
```

- **Mathematical operations**
- NumPy provides a wide range of mathematical operations that you can perform on arrays. Here are some examples:



```
python

import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(a + b)  # prints [5 7 9]
print(a * b)  # prints [ 4 10 18]
print(np.dot(a, b))  # prints 32

c = np.array([[1, 2], [3, 4]])
d = np.array([[5, 6], [7, 8]])

print(np.matmul(c, d))  # prints [[19 22]
                        #         [43 50]]
```

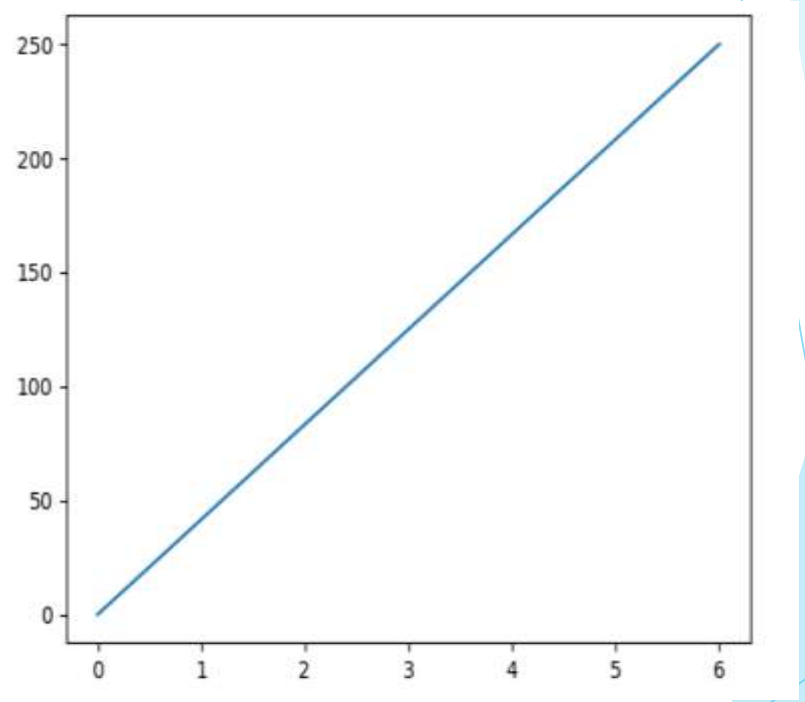
Matplotlib

- **Matplotlib**: This library is responsible for plotting numerical data. And that's why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc.
- Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
- Matplotlib was created by John D. Hunter.
- Matplotlib represents
  - Pyplot
  - Plotting
  - Markers
  - Line
  - Labels
  - Grid
  - Subplot
  - Scatter
  - Bars
  - Histograms
  - PieCharts

Matplotlib (pyplot)

- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the **plt** alias:
- `import matplotlib.pyplot as plt`
- Example: Draw a line in a diagram from position (0,0) to position (6,250):

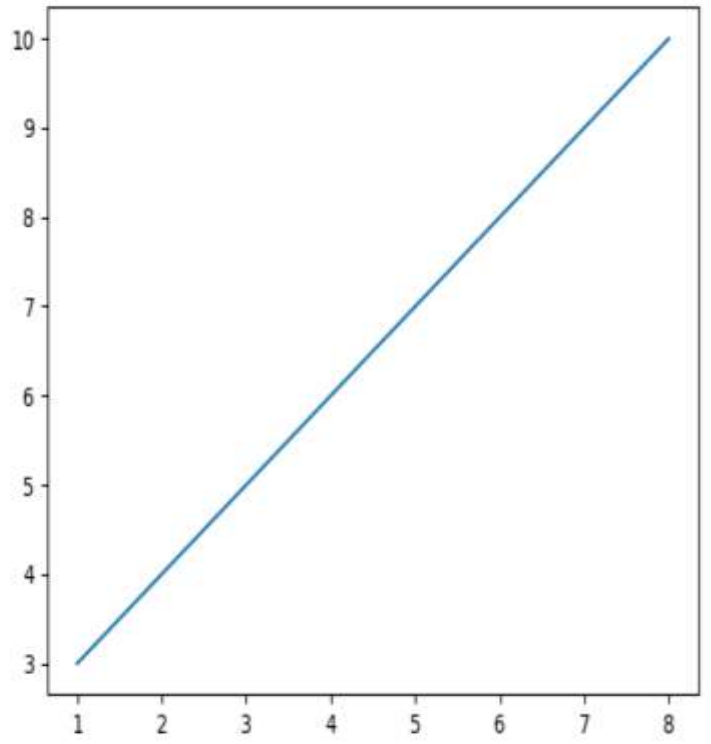
-
  ```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

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



Matplotlib (Plotting)

- 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.
- If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.
- Draw a line in a diagram from position (1, 3) to position (8, 10)

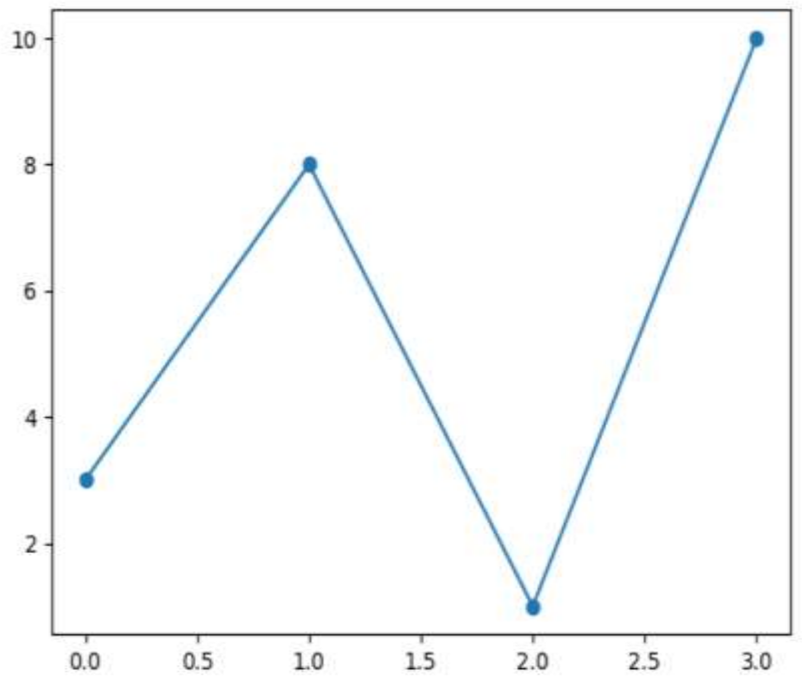
-
  ```
  import matplotlib.pyplot as plt
  import numpy as np

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

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



Matplotlib (Markers)

- Markers
- You can use the keyword argument marker to emphasize each point with a specific marker.
- Mark each point with a circle.

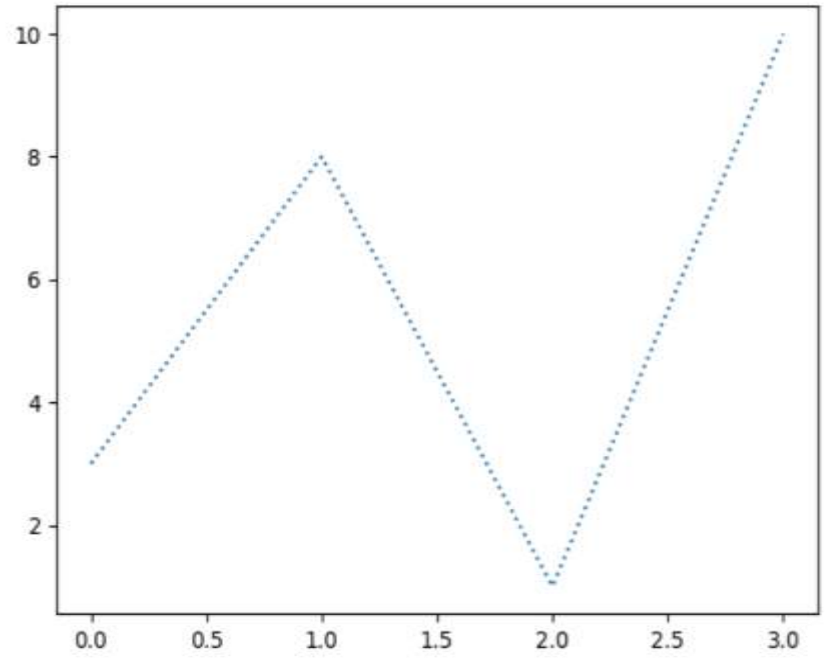
-
  ```
  import matplotlib.pyplot as plt
  import numpy as np

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

  plt.plot(ypoints, marker = 'o')
  plt.show()
  ```



Matplotlib (Line)

- Linestyle
- You can use the keyword argument **linestyle** or shorter ls, to change the style of the plotted line:
- Use dotted lines

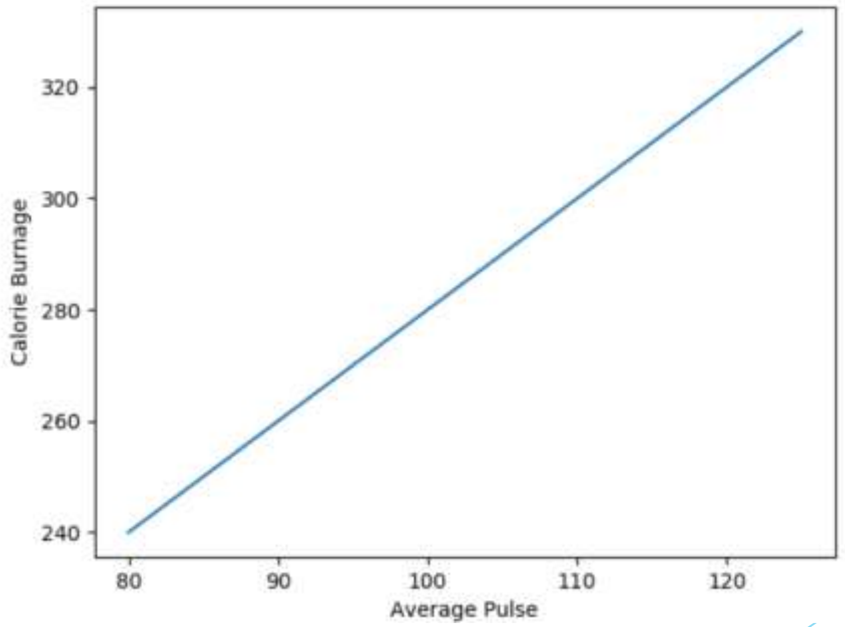
-
  ```
  import matplotlib.pyplot as plt
  import numpy as np

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

  plt.plot(ypoints, linestyle
  = 'dotted')
  plt.show()
  ```



Matplotlib Labels

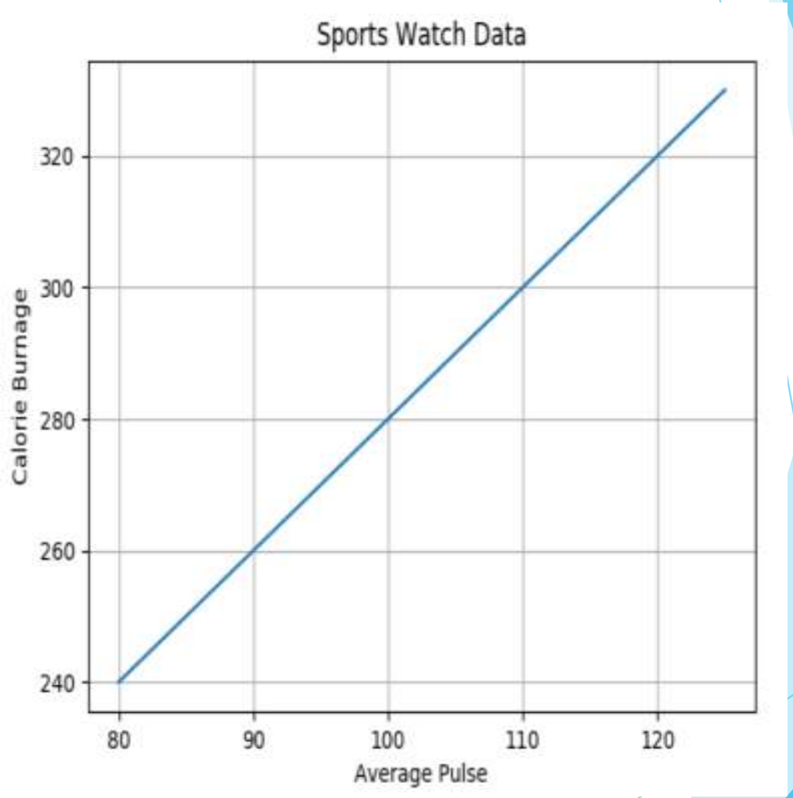
- Create Labels for a plot
- With pyplot,you can use the xlabel() and ylabel() functions to set a label for the x- and y-axis.
- Add labels to x- and y-axis:

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

  x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
  y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

  plt.plot(x, y)

  plt.xlabel("Average Pulse")
  plt.ylabel("Calorie Burnage")

  plt.show()
  ```



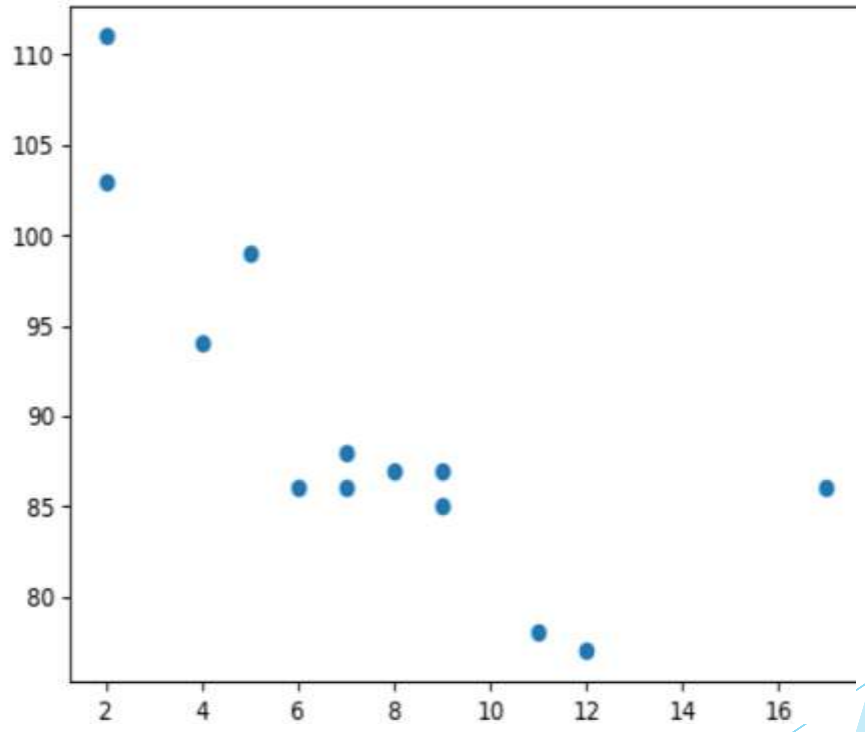
Matplotlib (Grid)

- Adding Grid Lines to a Plot
- With Pyplot,you can use **grid( )** function to add grid lines to the plot.
- Example:
-
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()

plt.show()
```



Matplotlib Scatters

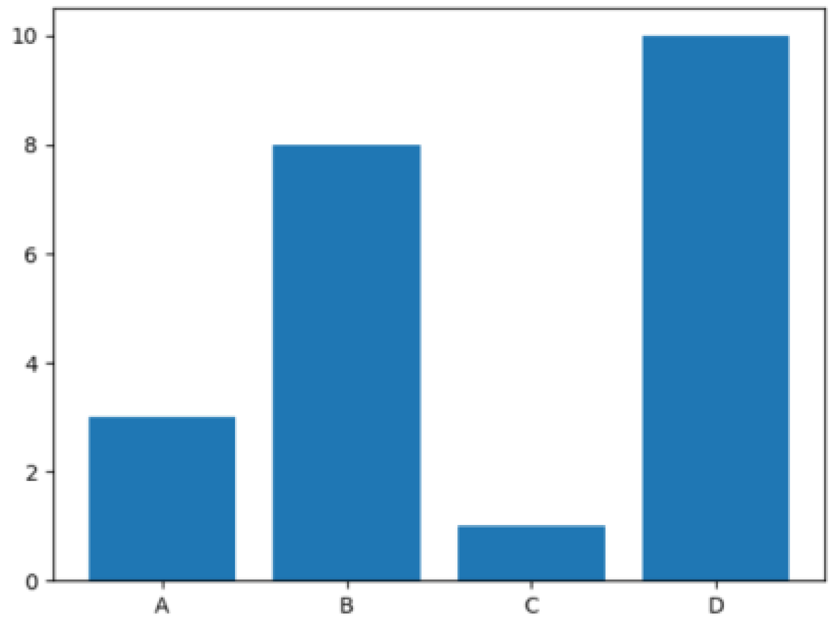
- The **scatter()** function plots one dot for each observation.
- A simple scatter plot example:
-
```
import matplotlib.pyplot as plt
import numpy as np

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()
```



Matplotlib(Bar)

- Creating a Bar: bar( )
- Example:

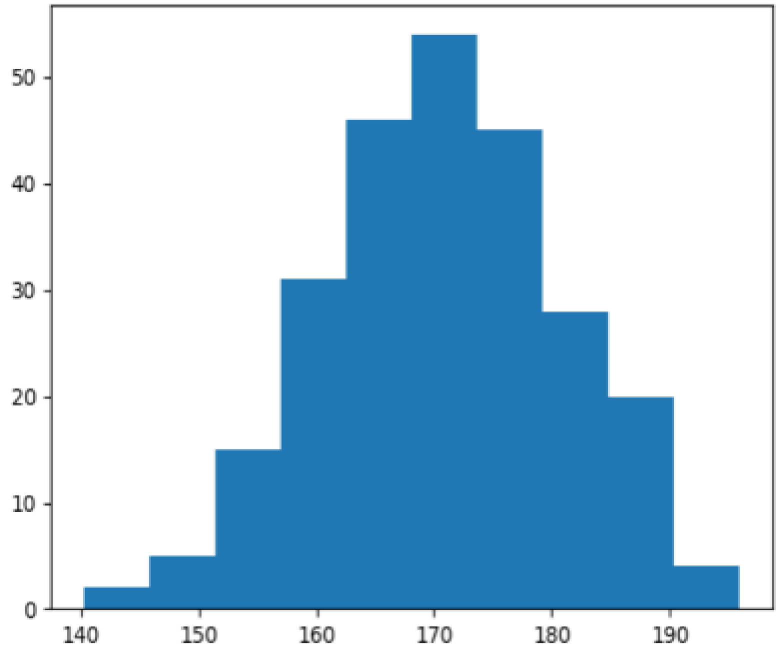
-
  ```
  import matplotlib.pyplot as plt
  import numpy as np

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

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



Matplotlib(Histograms)

- Histogram – **hist( )**
- A histogram is a graph showing frequency distributions.It is a graph showing the number of observations within each given interval.
- Example: Say you ask for the height of 250 people, you might end up with a histogram like this:
- 2 people from 140 to 145cm
  5 people from 145 to 150cm
  15 people from 151 to 156cm
  31 people from 157 to 162cm
  46 people from 163 to 168cm
  53 people from 168 to 173cm
  45 people from 173 to 178cm
  28 people from 179 to 184cm
  21 people from 185 to 190cm
  4 people from 190 to 195cm
- For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10

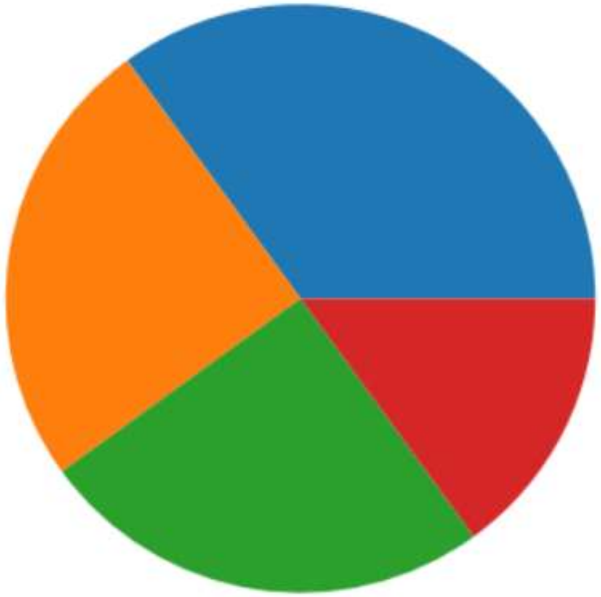
```
import numpy as np
x = np.random.normal(170, 10, 250)
print(x)
```



Matplotlib(Pie chart)

- Pie Chart – **pie( )**
- **Example:**

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

  y = np.array([35, 25, 25, 15])

  plt.pie(y)
  plt.show()
  ```



Pandas

- Pandas is an open-source data analysis and manipulation library for the Python programming language.
- It provides data structures for efficiently storing and manipulating large datasets, as well as tools for **processing**, **cleaning**, and **transforming data**.
- The name **"Pandas"** has a reference to both **"Panel Data",** and "Python Data Analysis" and was created by Wes McKinney in 2008.
- 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**.

Pandas

- Some of the **key features** of Pandas include:

1. Data Structures: Pandas provides two primary data structures - **Series and DataFrame** - which allow for easy manipulation of data.
2. Data Cleaning: Pandas provides powerful tools for cleaning and preprocessing data, such as removing missing values and handling duplicates.
3. Data Aggregation: Pandas makes it easy to perform complex data aggregation operations, such as grouping and pivoting, on large datasets.
4. Data Visualization: Pandas integrates with popular data visualization libraries like Matplotlib and Seaborn to allow for easy and customizable data visualization.
5. Integration: Pandas integrates well with other popular Python libraries, such as NumPy, SciPy, and Scikit-learn, to provide a comprehensive data analysis and machine learning toolkit.

Dask

- Dask is a parallel computing library for Python that allows users to work with large datasets that cannot fit into memory on a single machine.
- It provides parallelized versions of many familiar data structures and algorithms, such as Pandas dataframes and NumPy arrays, and can scale up to work with datasets that are too big to fit into memory on a single machine.

Dask

- Creating a Dask dataframe:This code creates a Dask dataframe from a CSV file. The dataframe is lazily loaded, meaning that Dask only reads in the data when it needs to perform a computation on it.

```
import dask.dataframe as dd
```

```
df = dd.read_csv('my_large_data.csv’)
```

- Filtering a Dask dataframe: This code filters the dask dataframe **df** to only include rows where the values in the column named ‘column_name’ is greater than 10.

```
filtered_df = df[df['column_name'] > 10]
```

- Aggregating a Dask dataframe: This code computes the mean of each group in the Dask dataframe **‘df’** based on the values of the column_name.

```
aggregated_df = df.groupby('column_name').mean()
```

- Computing the result of a Dask computation: This code computes the result of the previous computation by actually loading the data into memory and performing the computation.The result is a Pandas dataframe.

```
result = aggregated_df.compute()
```

Dask

- Using Dask to parallelize a Python function: This code defines a Python function ‘my_function’ that can be applied to each element of a list ‘my_data’ in parallel using Dask. The @delayed decorator tells Dask to delay the execution of the function until it is explicitly computed later. ‘dask.compute’ function actually computes the results in parallel and returns them as a list.
- import dask from dask import delayed “

```
@delayed
def my_function(x):
# Some computation on x
 return result
 result = dask.compute([my_function(x) for x in my_data])[0]
```