NumPy Library

- What is NumPy?
  - NumPy stands for Numerical Python.
  - NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
  - NumPy is a Python library used for working with arrays.
  - It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- 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
    - Numpy provides a lot of supporting functions that make working with ndarray very easy.

NumPy Library

- Why is NumPy Faster Than Lists?
  - NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
  - This behavior is called locality of reference in computer science.
  - This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.
- Which Language is NumPy written in?
  - NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

Installation of NumPy

```
C:\Users\Your Name>pip install numpy
```

Checking NumPy Version

```
import numpy as np

print(np.__version__)
```

Creating Array

- O-D or Scalar Array
  - `arr = np.array(42)`
- 1-D Arrays:
  - An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.
  - `arr = np.array([1, 2, 3, 4, 5])`
- 2-D Arrays
  - An array that has 1-D arrays as its elements is called a 2-D array.
  -
```
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
```

Creating Array

- 3-D arrays
  - An array that has 2-D arrays (matrices) as its elements is called 3-D array.

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

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

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

Check Dimension of Array

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

Output

Creating Arrays

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(type(arr))
```

```
<class 'numpy.ndarray'>
```

- To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

arr = np.array((1, 2, 3, 4, 5))

Indexing

Access 1-D Array Element

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print("Element at location 1 : ", arr[1])
print("Addition : ", arr[0] + arr[1])
```

```
Element at location 1 :  2
Addition :  3
```

Access 2-D Array Element

```
import numpy as np
arr = np.array([[1, 2, 3],
                [10, 11, 12]])
print('0th Element of 1st Row: ', arr[1, 0])
```

```
0th Element of 1st Row:  10
```

Indexing

Access 3-D Array Element

```
import numpy as np
arr = np.array([[[1, 2, 3],
                 [4, 5, 6]],
                [[7, 8, 9],
                 [10, 11, 12]]])
print("1st 2-D Matrix, 1st Row and Col. 0 : ",arr[1, 1, 0])
```

```
1st 2-D Matrix, 1st Row and Col. 0 :  10
```

Negative Indexing

```
import numpy as np
arr = np.array([[1, 2, 3],
                [7, 8, 9]])
print('Last element of 1st Row : ', arr[1, -1])
```

```
Last element of 1st Row :  9
```

1-D Array Slicing

```
import numpy as np
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7])
print("Element from index 1 to 4 :", arr[1:5])
print("Element from index 5 onwards :", arr[5:])
print("Element from index 0 upto 3 :", arr[:4])
print("Elements using Negative indices :", arr[-4:-1])
print("Elements at even locations :", arr[0: len(arr): 2])
print("Reverse order of elements :", arr[::-1])
print("Reverse order of elements :", arr[6:2:-1])
```

```
Element from index 1 to 4 : [1 2 3 4]
Element from index 5 onwards : [5 6 7]
Element from index 0 upto 3 : [0 1 2 3]
Elements using Negative indices : [4 5 6]
Elements at even locations : [0 2 4 6]
Reverse order of elements : [7 6 5 4 3 2 1 0]
Reverse order of elements : [6 5 4 3]
```

2-D Array Slicing

```
import numpy as np
arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8],
                [9, 10, 11, 12]])
print("Column 3 of Row 0 and 1 : ", arr[0:2, 3])
print("Column 0, 1, 2 of Row 1 and 2 : \n", arr[1:3, 0:3])
```

```
Column 3 of Row 0 and 1 :  [4 8]
Column 0, 1, 2 of Row 1 and 2 :
 [[ 5  6  7]
 [ 9 10 11]]
```

NumPy Array Iteration

```
import numpy as np
arr = np.array([7, 8, 9, 10])
for element in arr:
  print(element)
```

```
7
8
9
10
```

```
import numpy as np
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
for element in arr:
  print(element)
```

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

```
import numpy as np
arr = np.array([[[1, 2, 3],
                 [4, 5, 6]],
                [[7, 8, 9],
                 [10, 11, 12]]])
for element in arr:
  print("Element : \n", element)
```

```
Element :
 [[1 2 3]
 [4 5 6]]
Element :
 [[ 7  8  9]
 [10 11 12]]
```

NumPy Array Iteration

```
import numpy as np
arr = np.array([[1, 2],
                [3, 4]])
for row in arr:
  for element in row:
    print(element)
```

```
1
2
3
4
```

Iteration using nditr

```
import numpy as np

arr = np.array([[[1, 2],
                 [3, 4]],
                [[5, 6],
                 [7, 8]]])
for x in np.nditer(arr):
  print(x)
```

```
1
2
3
4
5
6
7
8
```

NumPy Array Iteration

```
import numpy as np
arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8]])
for x in np.nditer(arr[:, ::2]):
  print(x)
```

```
1
3
5
7
```

Iteration using ndenumerate

```
import numpy as np
arr = np.array([[1, 2],
                [5, 6]])
for idx, x in np.ndenumerate(arr):
  print(idx, x)
```

```
(0, 0) 1
(0, 1) 2
(1, 0) 5
(1, 1) 6
```

Data Types in NumPy

- List of data types in NumPy and characters used to represent them:
  - i - integer
  - b - boolean
  - u - unsigned integer
  - f - float
  - c - complex float
  - m - timedelta
  - M - datetime
  - O - object
  - S - string
  - U - unicode string
  - V - fixed chunk of memory for other type (void)

Data Types in NumPy

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

The NumPy array object has a property called dtype that returns the data type of the array

```
int64
```

We use the array() function to create arrays, this function can take an optional argument: dtype that allows us to define the expected data type of the array elements

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='f2')
print(arr)
print(arr.dtype)
```

```
[1. 2. 3. 4.]
float16
```

For i, u, f, S and U we can define size as well

Data Types in NumPy

```
import numpy as np
arr = np.array(['pineapple', 'banana', 'cherry'])
print(arr.dtype)
print(arr)
```

```
<U9
['pineapple' 'banana' 'cherry']
```

Data Types in NumPy

```
import numpy as np
arr = np.array([10, 2, 3, 4], dtype='S')
print(arr)
print(arr.dtype)
```

```
[b'10' b'2' b'3' b'4']
|S2
```

```
import numpy as np
arr = np.array(['1', '2', '3'], dtype='i')
print(arr)
```

```
[1 2 3]
```

Data Types in NumPy

If a type is given in which elements can't be casted then NumPy will raise a ValueError.

```
import numpy as np
arr = np.array(['a', '2', '3'], dtype='i')
print(arr)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback
<ipython-input-26-bc5f30a52b19> in <module>
      1 import numpy as np
----> 2 arr = np.array(['a', '2', '3'], dtype='i')
      3 print(arr)
```

Data Types in NumPy

```
import numpy as np
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype('i')
print(newarr)
print(newarr.dtype)
```

```
[1 2 3]
int32
```

The astype() function creates a copy of the array, and allows you to specify the data type as a parameter.

```
import numpy as np
arr = np.array([1, 0, 3, -1])
newarr = arr.astype(float)
print(newarr)
print(newarr.dtype)
```

```
[ 1.  0.  3. -1.]
float64
```

Shape of Array

```
import numpy as np
arr = np.array([[[1, 2, 5],
                 [3, 4, 5]],
                [[5, 6, 5],
                 [7, 8, 5]]])
print(arr.shape)
```

```
(2, 2, 3)
```

*The **shape** of an array is the number of elements in each dimension.*

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=2)
print(arr)
print('shape of array :', arr.shape)

arr = np.array([1, 2, 3, 4], ndmin=3)
print(arr)
print('shape of array :', arr.shape)
```

```
[[1 2 3 4]]
shape of array : (1, 4)
[[[1 2 3 4]]]
shape of array : (1, 1, 4)
```

Create an array with 2/3 dimensions using ndmin using a vector with values 1,2,3,4 and verify that last dimension has value 4

Reshape Array

- Reshaping means changing the shape of an array.
- The shape of an array is the number of elements in each dimension.
- By reshaping we can add or remove dimensions or change number of elements in each dimension.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = arr.reshape(2, 3)
print(newarr)
```

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

Convert the following 1-D array with 12 elements into a 2-D array.

Joining Arrays

- We pass a sequence of arrays that we want to join to the concatenate() function, along with the axis.

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[4, 5], [6, 7]])
arr = np.concatenate((arr1, arr2),
                     axis=0)
print(arr)
```

```
[[1 2]
 [3 4]
 [4 5]
 [6 7]]
```

Join two 2-D arrays along columns (axis=0)

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[4, 5], [6, 7]])
arr = np.concatenate((arr1, arr2),
                          axis=1)
print(arr)
```

```
[[1 2 4 5]
 [3 4 6 7]]
```

Join two 2-D arrays along rows (axis=1)

Joining Arrays

- We pass a sequence of arrays that we want to join to stack() method along with the axis.
- Stacking is same as concatenation, only difference is that stacking is done along new axis.

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[4, 5], [6, 7]])
arr = np.stack((arr1, arr2),
                          axis=0)
print(arr)
```

```
[[[1 2]
  [3 4]]

 [[4 5]
  [6 7]]]
```

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[4, 5], [6, 7]])
arr = np.stack((arr1, arr2),
                             axis=1)
print(arr)
```

```
[[[1 2]
  [4 5]]

 [[3 4]
  [6 7]]]
```

Joining Arrays

- NumPy provides a helper function: hstack() to stack along rows.
- NumPy provides a helper function: vstack() to stack along columns.

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[4, 5], [6, 7]])
arr = np.hstack((arr1, arr2))
print(arr)
```

```
[[1 2 4 5]
 [3 4 6 7]]
```

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[4, 5], [6, 7]])
arr = np.vstack((arr1, arr2))
print(arr)
```

```
[[1 2]
 [3 4]
 [4 5]
 [6 7]]
```

Sorting Arrays

- The NumPy ndarray object has a function called sort(), that will sort a specified array.

```
import numpy as np
arr = np.array([1, 3, 2, 6, 4])
arr = np.sort(arr)
print(arr)
```

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

```
import numpy as np
arr = np.array([[1, 6, 2], [12, 5, 8]])
arr = np.sort(arr)
print(arr)
```

```
[[ 1  2  6]
 [ 5  8 12]]
```

Pandas

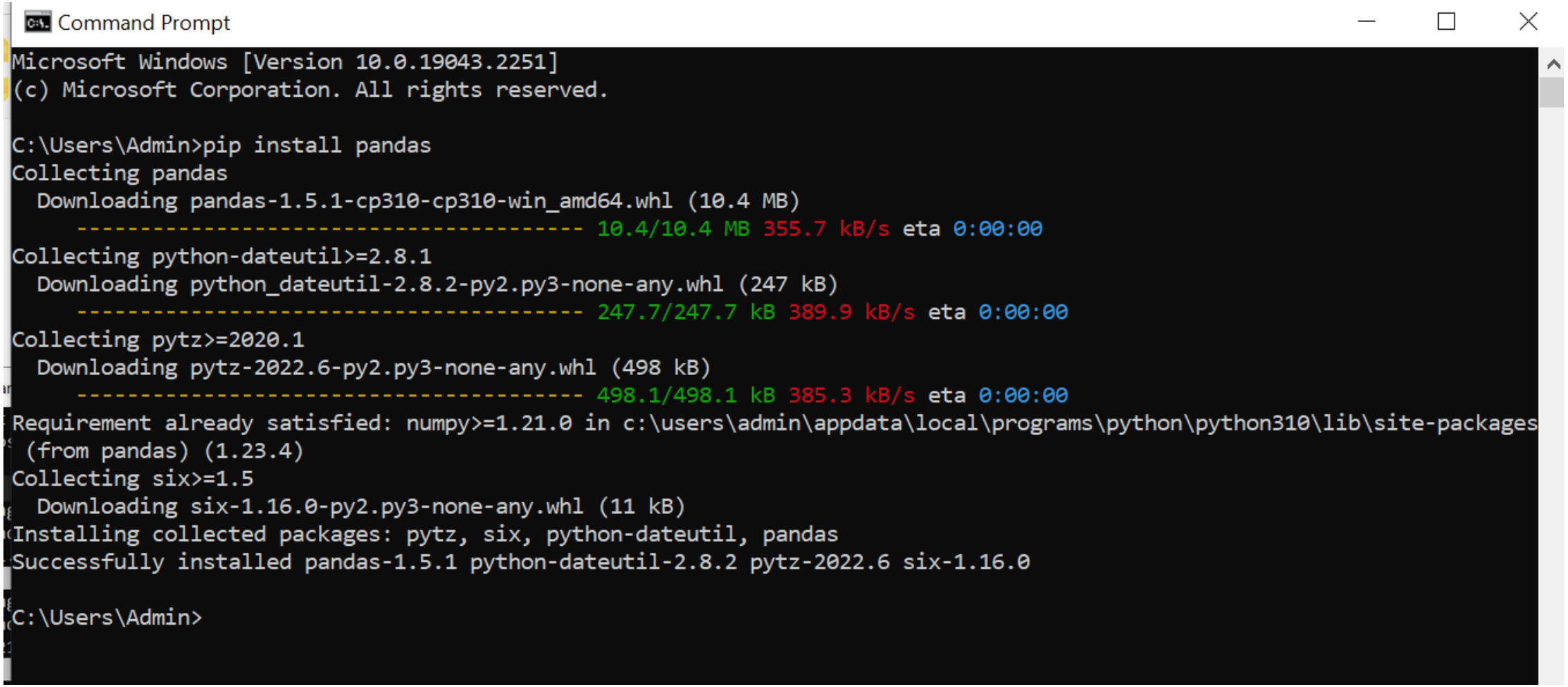
- What is Pandas?
  - Pandas is a Python library used for working with data sets.
  - It has functions for analyzing, cleaning, exploring, and manipulating data.
  - The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.
- Why Use Pandas?
  - Pandas allows us to analyze big data and make conclusions based on statistical theories.
  - Pandas can clean messy data sets, and make them readable and relevant.
  - Relevant data is very important in data science.

Pandas

- 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.

Pandas installation

In cmd execute command : pip install pandas



Pandas Series

- What is a Series?
  - A Pandas Series is like a column in a table.
  - It is a one-dimensional array holding data of any type.

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

```
0    1
1    7
2    2
dtype: int64
```

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a, index=["x", "y", "z"])
print(myvar)
```

```
x    1
y    7
z    2
dtype: int64
```

Access Series Elements

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a, index=["x", "y", "z"])
print(myvar[0], myvar['z'])
```

```
1 2
```

Key/Value Objects as Series

```
import pandas as pd
calories = {"A": 420, "B": 380, "C": 390}
myvar = pd.Series(calories)
print(myvar)
```

```
A    420
B    380
C    390
dtype: int64
```

Data Frames

- DataFrames
  - Data sets in Pandas are usually multi-dimensional tables, called DataFrames.
  - Series is like a column, a DataFrame is the whole table.

```
import pandas as pd
data = {
  "Name": ['Ajay', 'Raman', 'Umesh'],
  "Grade": [9.4, 9.2, 9.6]
}
df = pd.DataFrame(data)
print(df)
```

```
    Name  Grade
0   Ajay    9.4
1  Raman    9.2
2  Umesh    9.6
```

Access Elements of DataFrame

```
import pandas as pd
data = {
  "Name": ['Ajay', 'Raman', 'Umesh'],
  "Grade": [9.4, 9.2, 9.6]
}
df = pd.DataFrame(data)
print(df.loc[0])
```

```
Name      Ajay
Grade      9.4
Name: 0, dtype: object
```

Pandas use the loc attribute to return one or more specified row(s)

```
import pandas as p

data = {
    "Name" : ['Taru','Siraj','Aneeqa'],
    "grades" : [8.9,9.6,8]
    }

df = p.DataFrame(data)
print(df)

print(df.loc[0:1])
```

```
     Name  grades
0    Taru     8.9
1   Siraj     9.6
2  Aneeqa     8.0
    Name  grades
0   Taru     8.9
1  Siraj     9.6
```

Create Index for DataFrame

```
import pandas as pd
data = {
  "Name": ['Ajay', 'Raman', 'Umesh'],
  "Grade": [9.4, 9.2, 9.6]
}
df = pd.DataFrame(data,
                  index=[101, 102, 103])
print(df)
```

```
      Name  Grade
101   Ajay    9.4
102  Raman    9.2
103  Umesh    9.6
```

```
import pandas as pd
data = {
  "Name": ['Ajay', 'Raman', 'Umesh'],
  "Grade": [9.4, 9.2, 9.6]
}
df = pd.DataFrame(data,
                  index=[101, 102, 103])
print(df.loc[101])
```

```
Name      Ajay
Grade      9.4
Name: 101, dtype: object
```

Manipulating DataFrame

```
import pandas as pd
data = {
  "r_no": [1, 2, 3, 4, 5, 6, 7],
  "Grade": [9.4, 9.2, 9.6,
            8.2, 8.8, 9.5, 8.4]
}
df = pd.DataFrame(data)
print(df)
```

```
   r_no  Grade
0     1    9.4
1     2    9.2
2     3    9.6
3     4    8.2
4     5    8.8
5     6    9.5
6     7    8.4
```

```
import pandas as pd
data = {
  "r_no": [1, 2, 3, 4, 5, 6, 7],
  "Grade": [9.4, 9.2, 9.6,
            8.2, 8.8, 9.5, 8.4]
}
df = pd.DataFrame(data)
df = df.rename(columns={'r_no': 'Roll_no'})
print(df)
```

```
   Roll_no  Grade
0        1    9.4
1        2    9.2
2        3    9.6
3        4    8.2
4        5    8.8
5        6    9.5
6        7    8.4
```

Manipulating DataFrame

```
import pandas as pd
data = {
  "r_no": [1, 2, 3, 4, 5, 6, 7],
  "Grade": [9.4, 9.2, 9.6,
            8.2, 8.8, 9.5, 8.4]
}
df = pd.DataFrame(data)
df = df.sort_values(by=['Grade'],
                    ascending=False)
print(df)
```

```
   r_no  Grade
2     3    9.6
5     6    9.5
0     1    9.4
1     2    9.2
4     5    8.8
6     7    8.4
3     4    8.2
```

```
import pandas as pd
data = {
  "r_no": [1, 2, 3, 4, 5, 6, 7],
  "Grade": [9.4, 9.3, 9.6,
            8.2, 8.8, 9.4, 8.4]
}
df = pd.DataFrame(data)
df = df.sort_values(by=['Grade', 'r_no'],
                    ascending=[True, False])
print(df)
```

```
   r_no  Grade
3     4    8.2
6     7    8.4
4     5    8.8
1     2    9.3
5     6    9.4
0     1    9.4
2     3    9.6
```

Manipulating DataFrame

```
import pandas as pd
data = {
  "r_no": [1, 2, 3, 4, 5, 6, 7],
  "Grade": [9.4, 9.3, 9.6,
            8.2, 8.8, 9.4, 8.4]
}
df = pd.DataFrame(data)

def Calculate_Marks(a):
    return a*10

df['Marks'] = df['Grade'].apply(Calculate_Marks)
print(df)
```

	r_no	Grade	Marks
0	1	9.4	94.0
1	2	9.3	93.0
2	3	9.6	96.0
3	4	8.2	82.0
4	5	8.8	88.0
5	6	9.4	94.0
6	7	8.4	84.0

Manipulating DataFrame

```
import pandas as pd
data = {
  "r_no": [1, 2, 3, 4, 5, 6, 7],
  "Grade": [9.4, 9.3, 9.6,
            8.2, 8.8, 9.4, 8.4]
}
df = pd.DataFrame(data)

def Increase_Grades(a):
    return a + 0.2

df['Grade'] = df['Grade'].apply(Increase_Grades)
print(df)
```

```
   r_no  Grade
0     1    9.6
1     2    9.5
2     3    9.8
3     4    8.4
4     5    9.0
5     6    9.6
6     7    8.6
```

DataFrame: Apply Filter

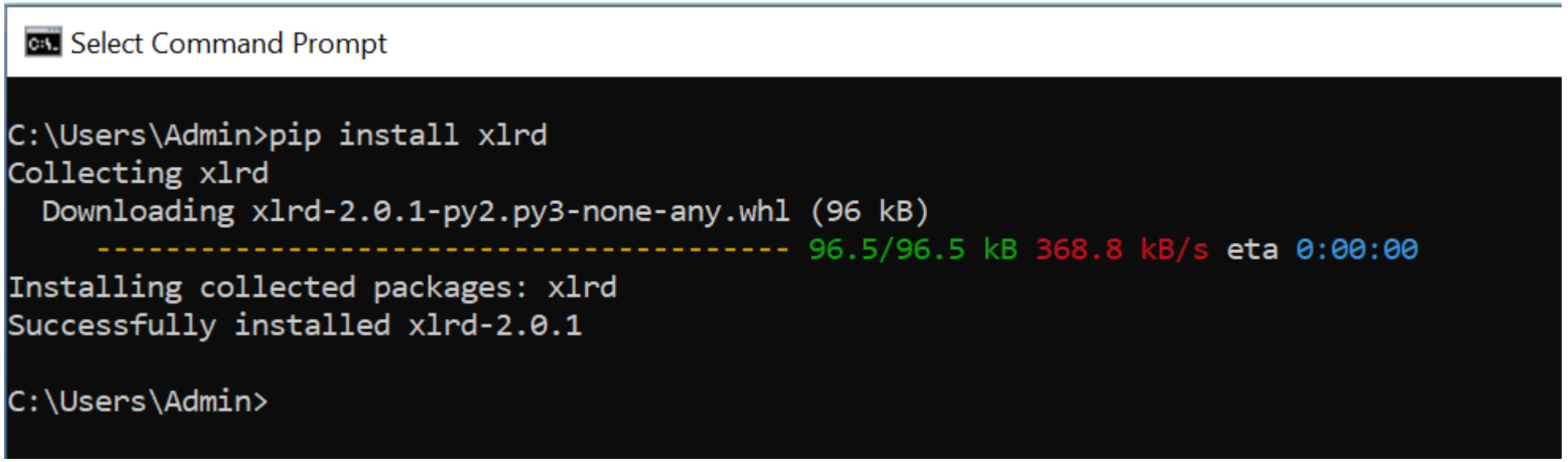
- Select Students with Grade above 9.0

```
import pandas as pd
data = {
  "r_no": [1, 2, 3, 4, 5, 6, 7],
  "Grade": [9.4, 9.3, 9.6,
            8.2, 8.8, 9.4, 8.4]
}
df = pd.DataFrame(data)
print(df[df.Grade > 9.0])
```

```
   r_no  Grade
0     1    9.4
1     2    9.3
2     3    9.6
5     6    9.4
```

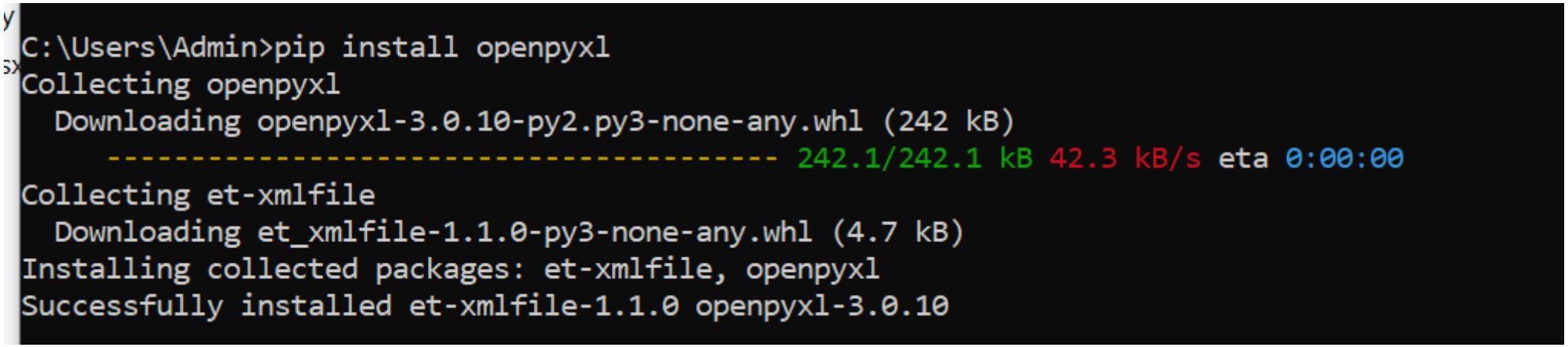
Excel file data read module installation

In cmd execute command : pip install xlrd



Excel file data read module installation

In cmd execute command : pip install openpyxl

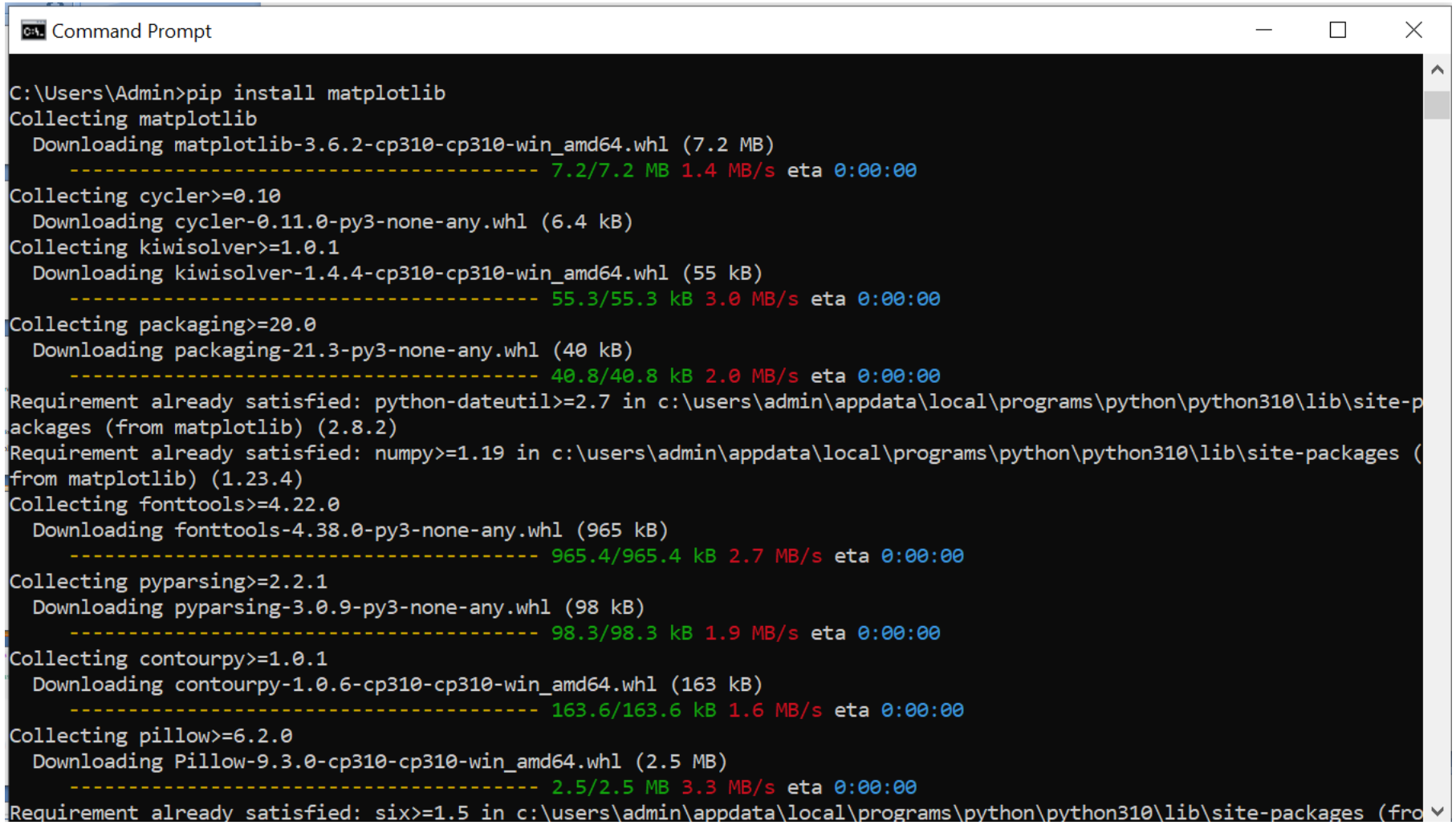


Matplotlib

- What is Matplotlib?
  - Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
  - Matplotlib was created by John D. Hunter.
  - Matplotlib is open source and we can use it freely.
  - Matplotlib is mostly written in python,
    - a few segments are written in C, Objective-C and Javascript for Platform compatibility.
- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:
  - import matplotlib.pyplot as plt

Matplotlib installation

In cmd execute command : pip install matplotlib

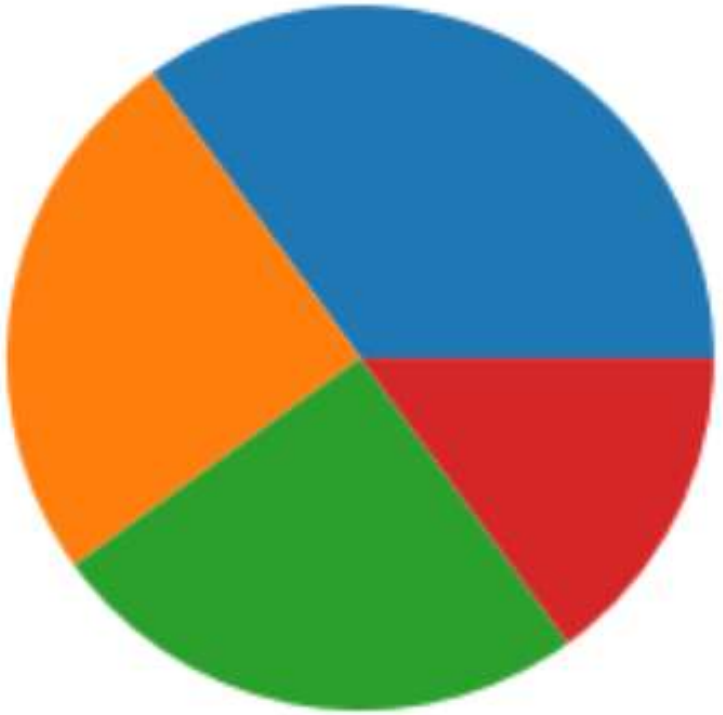


Types of Charts

- Pie Chart
  - With Pyplot, you can use the pie() function to draw pie charts

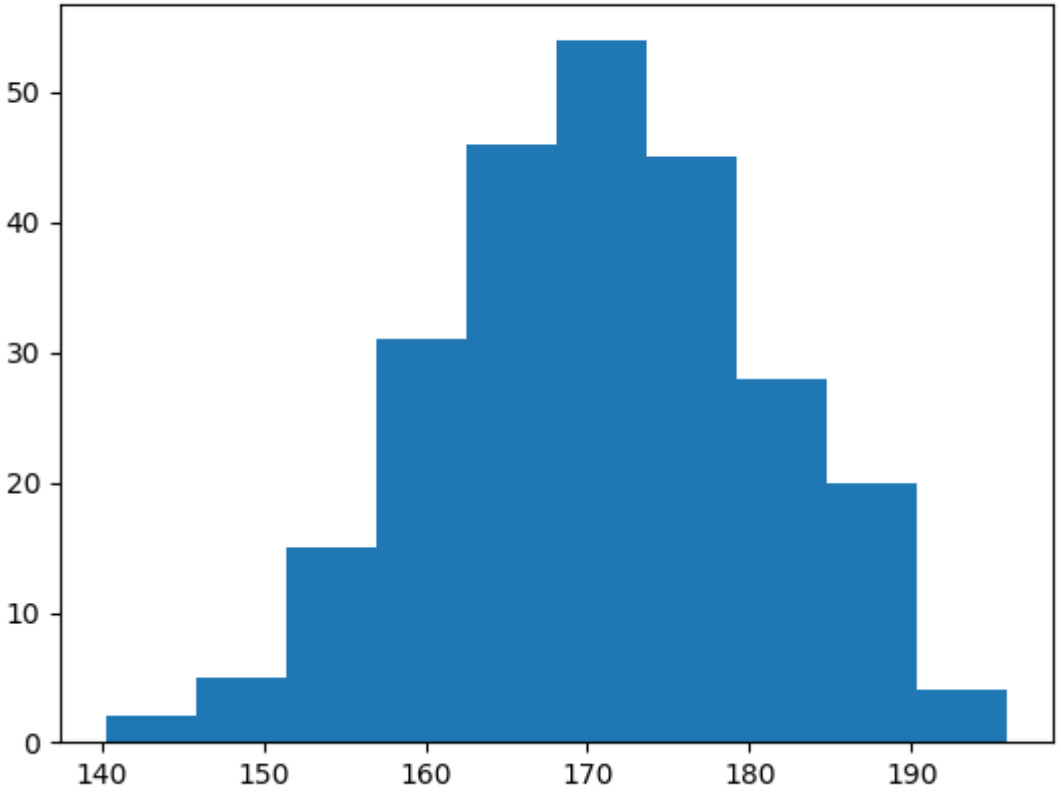
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
plt.pie(y)
plt.show()
```



Types of Charts

- Histogram
  - 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:



You can read from the histogram that there are approximately:
2 people from 140 to 145cm
5 people from 145 to 150cm
15 people from 151 to 156cm

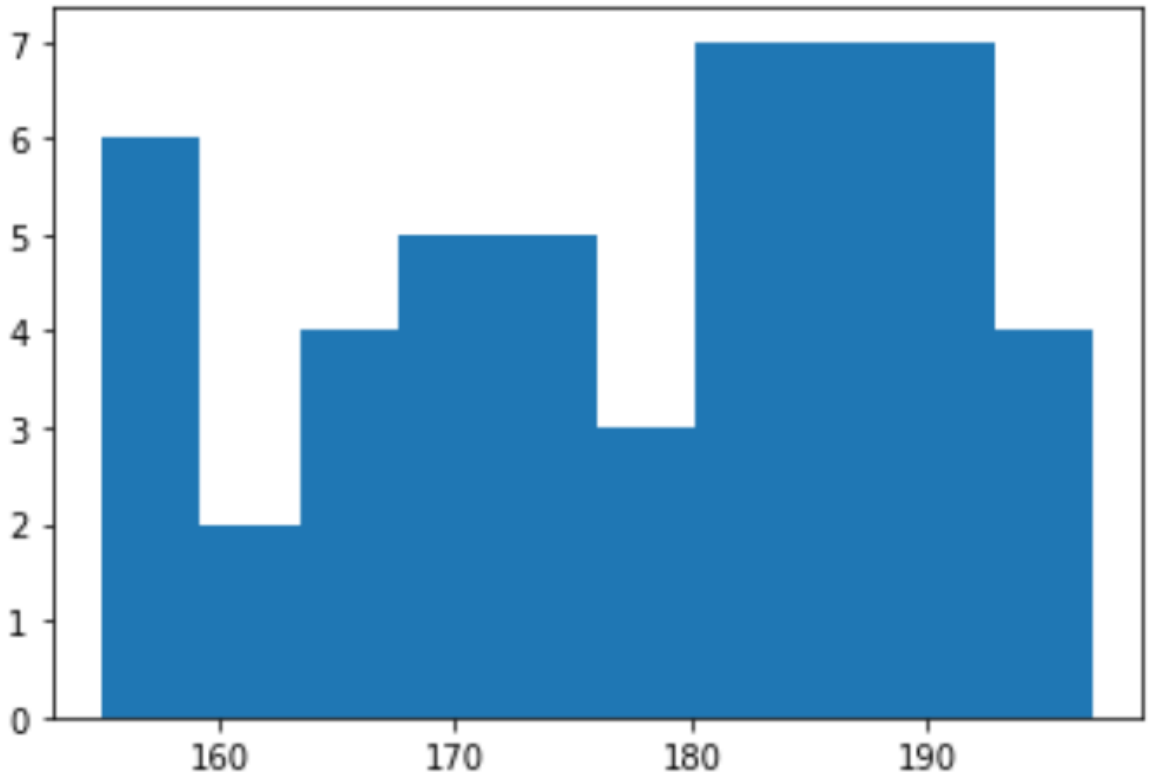
Types of Charts

Plotting Histogram using hist()

The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

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

x = np.random.randint(150, 200, 50)
plt.hist(x)
plt.show()
```

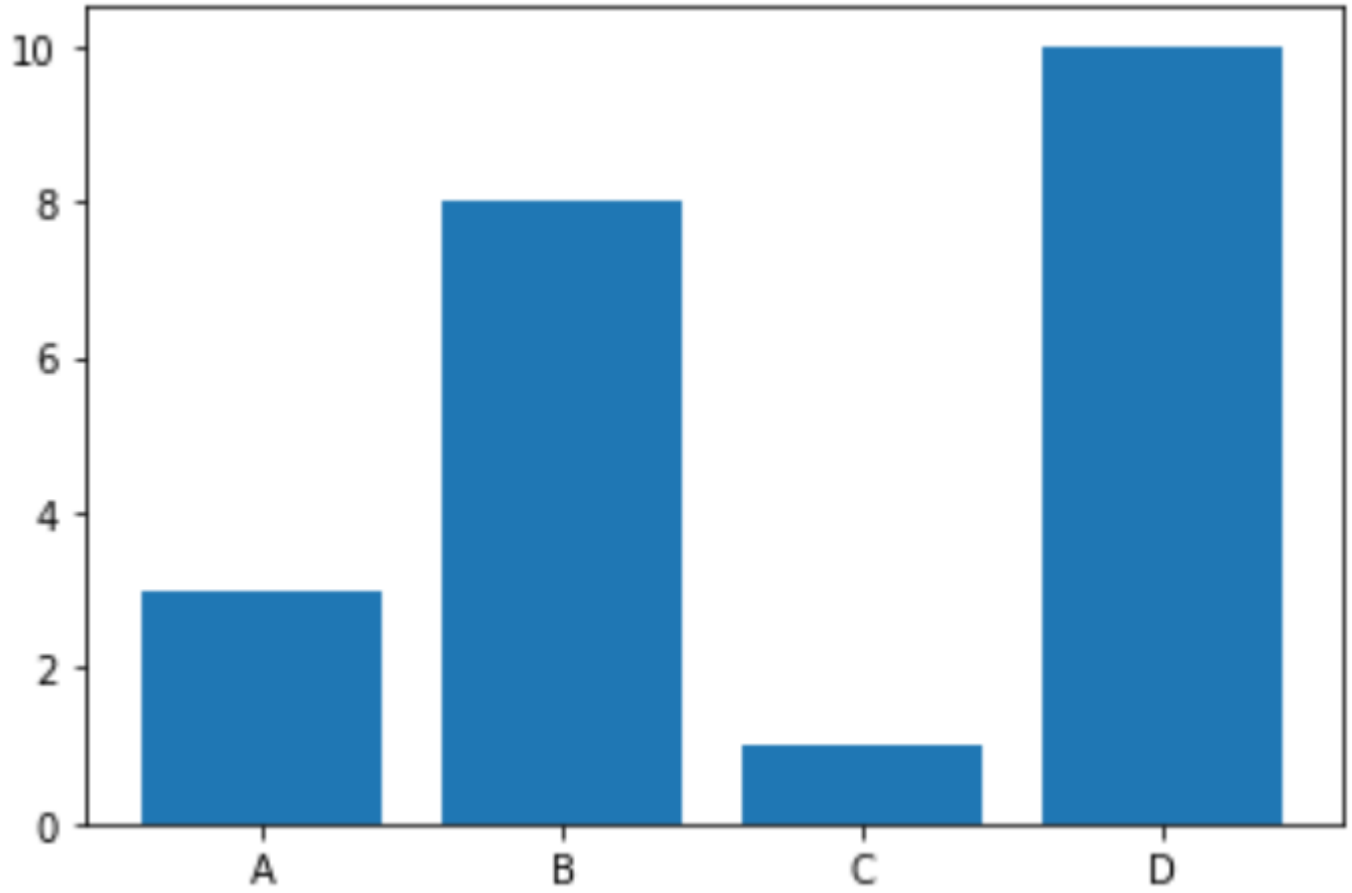


Types of Charts

- Bar Graph
  - you can use the bar() function to draw bar graphs

```
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()
```



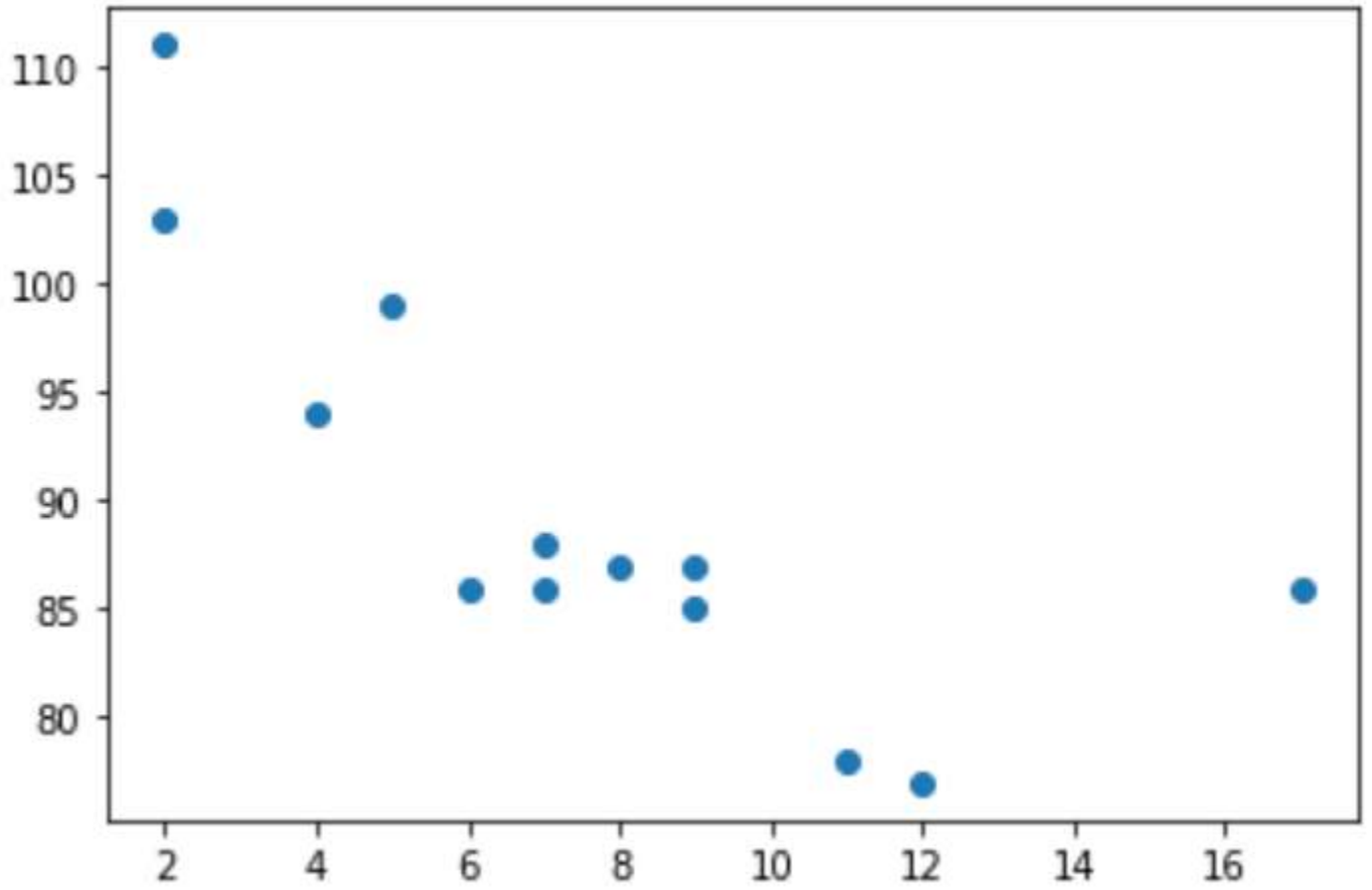
Types of Charts

- Scatter Plot
  - 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:

Types of Charts

```
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()
```



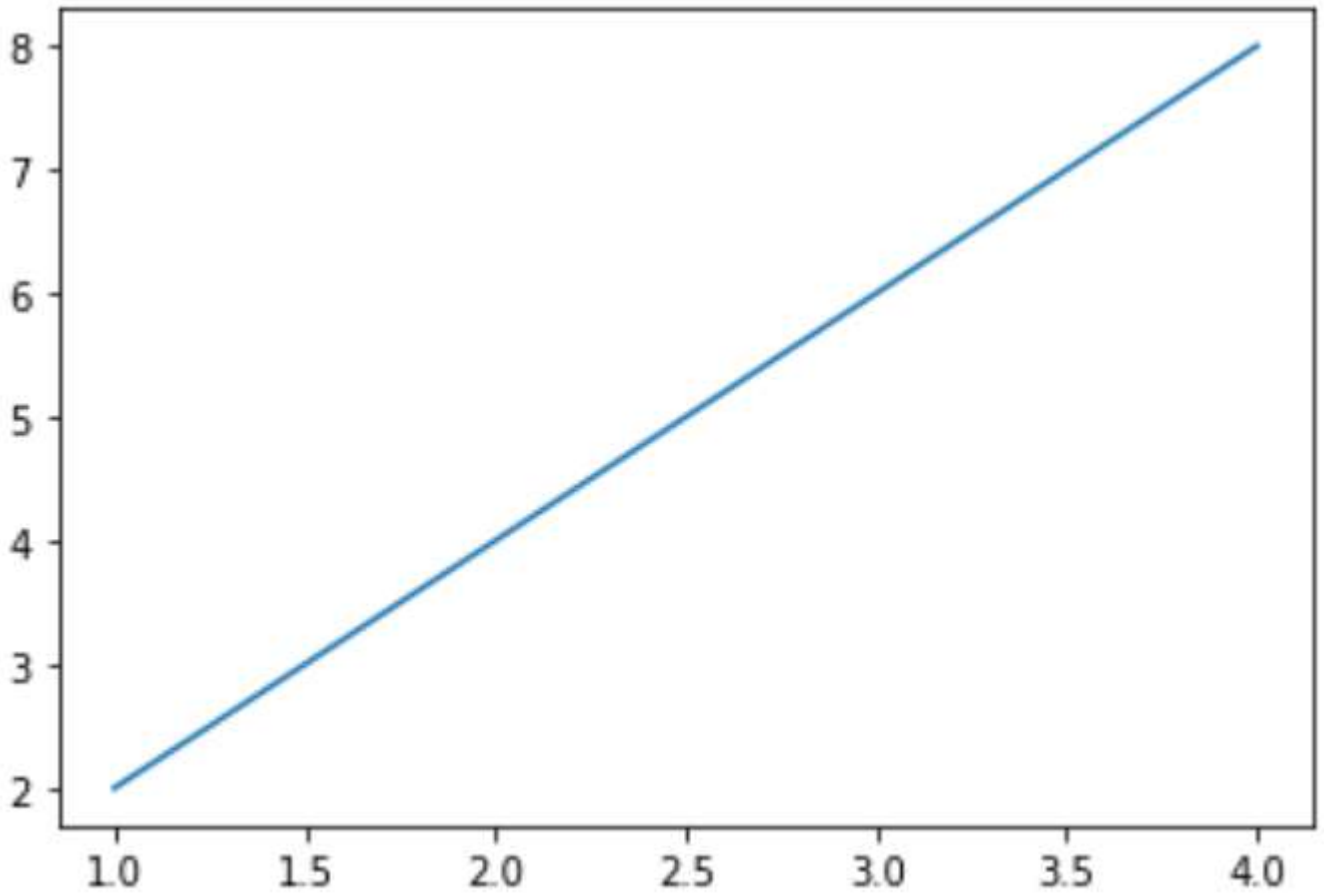
- The observation in the example above is the result of 13 cars passing by.
  - The X-axis shows how old the car is.
  - The Y-axis shows the speed of the car when it passes.
- Are there any relationships between the observations?
  - It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

Types of Charts

- Line Plot

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([1, 2, 3, 4])
y = x*2

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

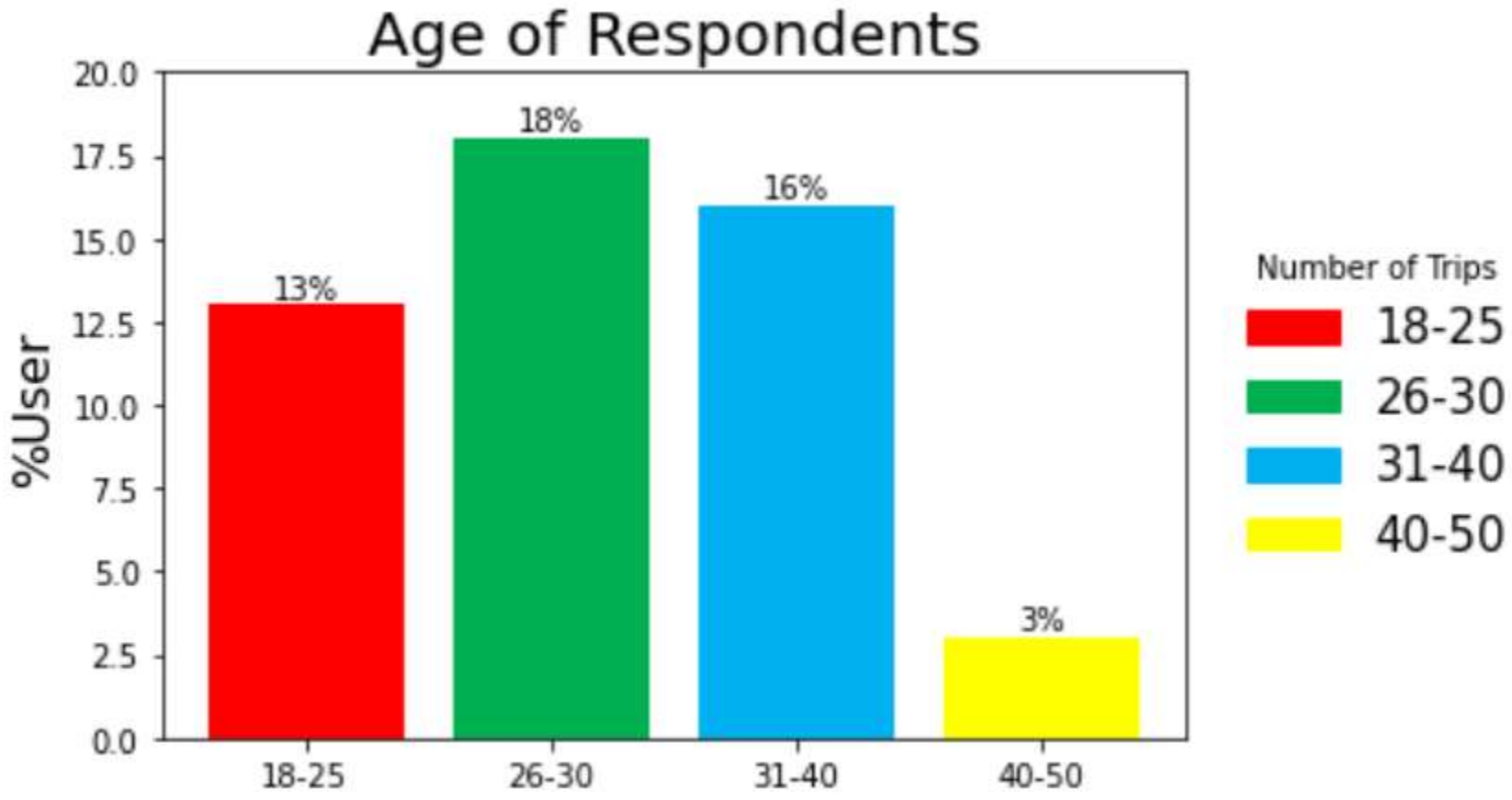


Legends, Annotation, Style

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Patch
color = ('red', '#00b050', '#00b0f0', 'yellow')
objects = ('18-25', '26-30', '31-40', '40-50')
y_pos = np.arange(len(objects))
performance = [13, 18, 16, 3]
width = 0.35  # the width of the bars
plt.bar(y_pos, performance, align='center', color=color)
plt.xticks(y_pos, objects)
plt.ylim(0, 20)
plt.ylabel('%User', fontsize=16)
plt.title('Age of Respondents', fontsize=20)

# map names to colors
cmap = dict(zip(performance, color))
# create the rectangles for the legend
patches = [Patch(color=v, label=k) for k, v in cmap.items()]
# add the legend
plt.legend(title='Number of Trips', labels=objects, handles=patches,
           bbox_to_anchor=(1.04, 0.5), loc='center left',
           borderaxespad=0, fontsize=15, frameon=False)
# add the annotations
for y, x in zip(performance, y_pos):
    plt.annotate(f'{y}%\n', xy=(x, y), ha='center', va='center')
```

Legends, Annotation, Style



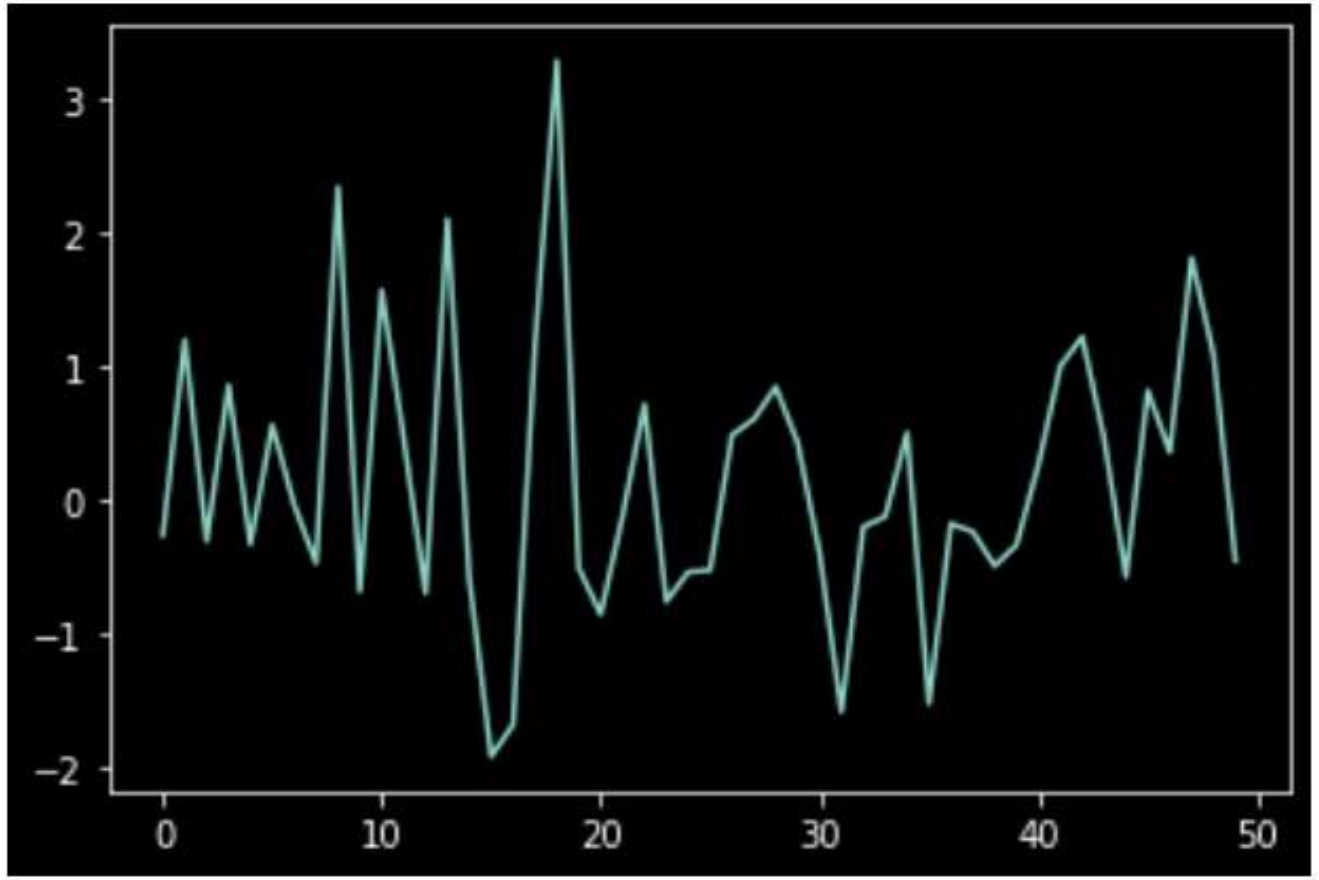
Legends, Annotation, Style

```
import matplotlib.pyplot as plt
print(plt.style.available)
```

['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind10']

Legends, Annotation, Style

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
data = np.random.randn(50)
plt.style.use('dark_background')
plt.plot(data)
plt.show()
```

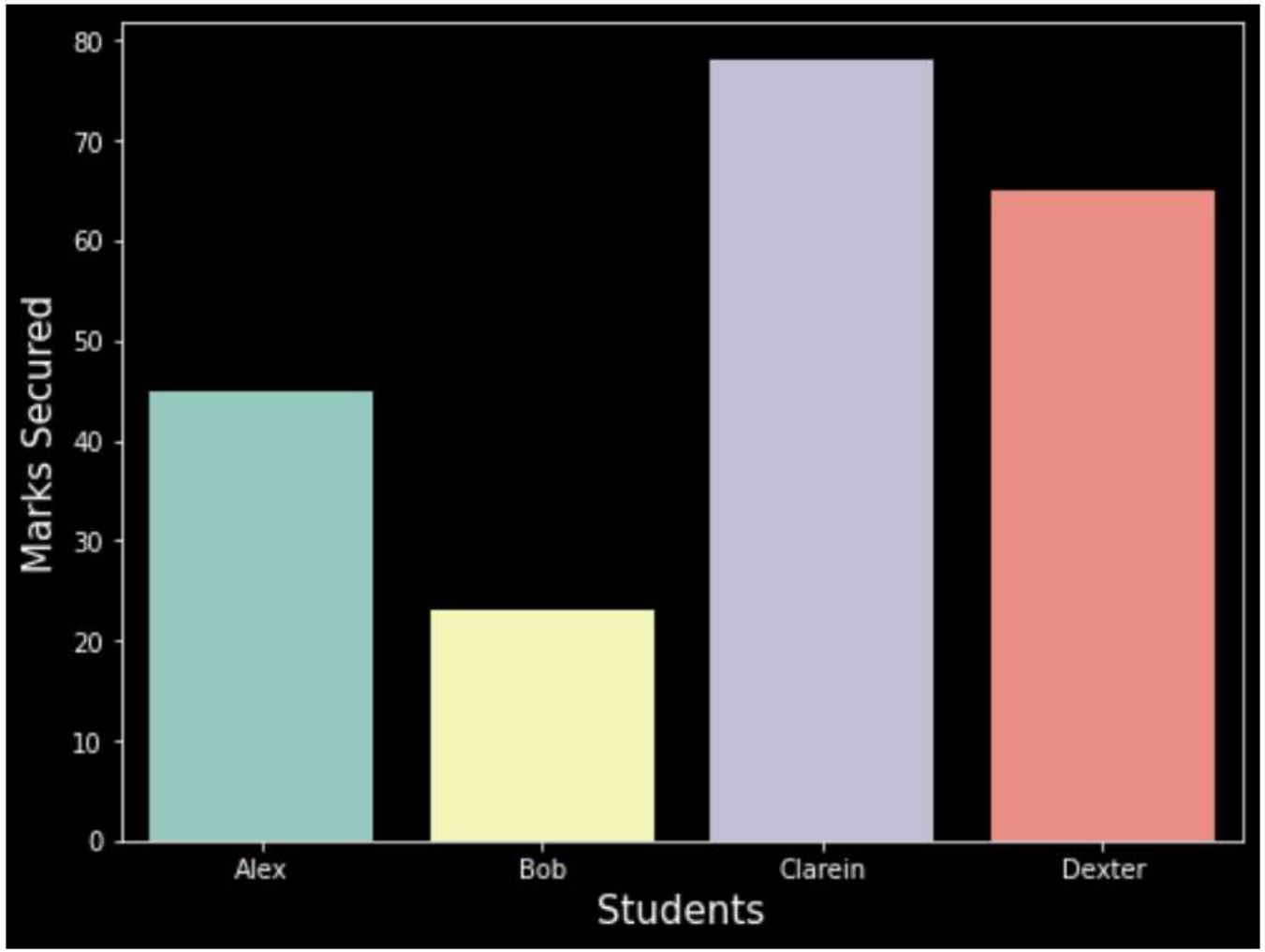


Plotting directly from Pandas Data Frame

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
data = {"Name": ["Alex", "Bob", "Clarein", "Dexter"],
        "Marks": [45, 23, 78, 65]}
df = pd.DataFrame(data, columns=['Name', 'Marks'])

plt.figure(figsize=(8, 6))
plots = sns.barplot(x="Name", y="Marks", data=df)
plt.xlabel("Students", size=15)
plt.ylabel("Marks Secured", size=15)
plt.show()
```

Plotting directly from Pandas Data Frame



Plotting directly from NumPy Array

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

x = np.arange(1, 11)
y = np.array([100, 10, 300, 20, 500,
              60, 700, 80, 900, 100])

plt.title("Line graph")
plt.xlabel("X axis")
plt.ylabel("Y axis")
plt.plot(x, y, color ="green")
plt.show()
```