Module 5

Module 5 – Introduction to Data science

- The os and sys modules, NumPy - Basics, Creating arrays, Arithmetic, Slicing, Matrix Operations, Random numbers.
- Plotting and visualization. Matplotlib - Basic plot, Ticks, Labels, and Legends.
- Working with CSV files. – Pandas - Reading, Manipulating, and Processing Data.
- Introduction to Micro services using Flask.

NumPy

What is NumPy in Python?

- NumPy in Python is a library that is used to work with arrays and was created in 2005 by Travis Oliphant.
- NumPy library in Python has functions for working in domain of Fourier transform, linear algebra, and matrices.
- Python NumPy is an open-source project that can be used freely. NumPy stands for **Numerical Python**.

NumPy Creating Arrays

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

1-Dimensional Array

Output: [1 2 3 4 5]

We can also pass a tuple in the array function to create an array.

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

The output would be similar to the above case.

Two Dimensional Arrays

2-D Arrays are the ones that have 1-D arrays as its element. The following code will create a 2-D array with 1,2,3 and 4,5,6 as its values.

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

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

NumPy Arithmetic Operations

- NumPy **Add** function
  - This function is used to add two arrays. If we add arrays having dissimilar shapes we get “Value Error”.

```
import numpy as np
a = np.array([10,20,100,200,500])
b = np.array([3,4,5,6,7])
np.add(a, b)
```

  - We can also use the add operator “+” to perform addition of two arrays.

```
import numpy as np
a = np.array([10,20,100,200,500])
b = np.array([3,4,5,6,7])
print(a+b)
```

NumPy Arithmetic Operations

- NumPy **Subtract** function
  - We use this function to output the difference of two arrays. If we subtract two arrays having dissimilar shapes we get “Value Error”.

```
import numpy as np
a = np.array([10,20,100,200,500])
b = np.array([3,4,5,6,7])
np.subtract(a, b)
```

  - NumPy Subtract Operator: We can also use the subtract operator “-” to produce the difference of two arrays.

```
import numpy as np
a = np.array([10,20,100,200,500])
b = np.array([3,4,5,6,7])
print(a-b)
```

NumPy Arithmetic Operations

- NumPy **Multiply** function
  - We use this function to output the multiplication of two arrays. We cannot work with dissimilar arrays.

```
import numpy as np
a = np.array([7,3,4,5,1])
b = np.array([3,4,5,6,7])
np.multiply(a, b)
```

  - NumPy Multiply Operator: We can also use the multiplication operator “*” to get the product of two arrays.

```
import numpy as np
a = np.array([7,3,4,5,1])
b = np.array([3,4,5,6,7])
print(a*b)
```

NumPy Arithmetic Operations

- NumPy **Divide** Function
  - We use this function to output the division of two arrays. We cannot divide dissimilar arrays.

```
import numpy as np
a = np.array([7,3,4,5,1])
b = np.array([3,4,5,6,7])
np.divide(a,b)
```

  - NumPy Divide Operator: We can also use the divide operator “/” to divide two arrays.

```
import numpy as np
a = np.array([7,3,4,5,1])
b = np.array([3,4,5,6,7])
print(a/b)
```

NumPy Arithmetic Operations

- NumPy **Mod and Remainder** function
  - We use both the functions to output the remainder of the division of two arrays.

```
import numpy as np
a = np.array([7,3,4,5,1])
b = np.array([3,4,5,6,7])
np.remainder(a,b)
```

Output: [1, 3, 4, 5, 1]

  - NumPy Mod Function

```
import numpy as np
a = np.array([7,3,4,5,1])
b = np.array([3,4,5,6,7])
np.mod(a,b)
```

NumPy Arithmetic Operations

- NumPy **Power** Function
  - This Function treats the first array as base and raises it to the power of the elements of the second array.

```
import numpy as np
a = np.array([7,3,4,5,1])
b = np.array([3,4,5,6,7])
np.power(a,b)
```

```
Output: [ 343, 81, 1024, 15625, 1]
```

NumPy Array Slicing

- Python NumPy array slicing is used to extract some portion of data from the actual array.
- The syntax of Python NumPy slicing is [start : stop : step]
  - Start : This index by default considers as ‘0’
  - stop : This index considers as a length of the array.
  - step : By default it is considered as ‘1’.

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

```
Output: [2 3 4 5]
```

NumPy Array Slicing

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

```
[5 6 7]
```

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

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

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

```
[5 6]
```

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

```
[2 4]
```

Slicing 2-Dimensional NumPy Arrays

- Use slicing a 2-dimensional array in both axes to obtain a rectangular subset of the original array.
- You can use arr[1:,1:3] to select rows 1: one to the end of the bottom of the array and columns 1:3 (columns 1 and 2).

```
# Create NumPy arrays
arr = np.array([[3, 5, 7, 9, 11],
                [2, 4, 6, 8, 10]])

# Use slicing a 2-D arrays
arr2 = arr[1:,1:3]
print(arr2)
# Output
#[[4 6]]
```

Matrix Operations -Transpose

- The transpose of a matrix is found by switching its rows with its columns.
  - We can use **np.transpose()** function

```
import numpy as np

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

print("\nWith np.transpose(a) function")
print(np.transpose(a))
```

Random numbers Numpy

- In NumPy, we have a module called random which provides functions for generating random numbers.
- These functions can be useful for generating random inputs for testing algorithms.

```
import numpy as np
# generate random integer from 0 to 9
random_number = np.random.randint(0, 10)
print(random_number)
# Output: 7
```

- In this example, we have used the random module to generate a random number. The random.randint() function takes two arguments,
  - 0 - a lower bound (inclusive)
  - 10 - an upper bound (exclusive)
- Here, random.randint() returns a random integer between 0 and 9.

Random numbers Numpy

- Generate Random Float in NumPy
  - We can also generate a random floating-point number between 0 and 1. For that we use the random.rand() function. For example,

```
import numpy as np
# generate random float-point number between 0 and 1
random_number = np.random.rand()

print(random_number)

# Output: 0.7696638323107154
```

Random numbers Numpy

- Generate Random Array in NumPy - NumPy's random module can also be used to generate an array of random numbers. For example,

```
import numpy as np

# generate 1D array of 5 random integers between 0 and 9
integer_array = np.random.randint(0, 10, 5)

print("1D Random Integer Array:\n",integer_array)

# generate 1D array of 5 random numbers between 0 and 1
float_array = np.random.rand(5)

print("\n1D Random Float Array:\n",float_array)

# generate 2D array of shape (3, 4) with random integers
result = np.random.randint(0, 10, (3,4))

print("\n2D Random Integer Array:\n",result)
```

```
1D Random Integer Array:
[9 7 8 4 2]

1D Random Float Array:
 [0.7877579  0.01723754
0.93995075 0.17126388
0.69913594]

2D Random Integer Array:
 [[0 5 3 8]
 [3 9 2 1]
 [8 7 1 2]]
```

Random numbers Numpy

- Choose Random Number from NumPy Array -To choose a random number from a NumPy array, we can use the random.choice() function.

```
import numpy as np

# create an array of integers from 1 to 5
array1 = np.array([1, 2, 3, 4, 5])

# choose a random number from array1
random_choice = np.random.choice(array1)

print(random_choice)

# Output: 3
```

References

- https://www.programiz.com/python-programming/numpy/random
- https://sparkbyexamples.com/python/numpy-array-slicing/
- https://towardsdatascience.com/top-10-matrix-operations-in-numpy-with-examples-d761448cb7a8
- https://data-flair.training/blogs/numpy-arithmetic-operations/

Matplotlib

What is Matplotlib?

- Matplotlib is an open-source drawing library that supports various drawing types
- You can generate plots, histograms, bar charts, and other types of charts with just a few lines of code
- It’s often used in web application servers, shells, and Python scripts

Basic plots in Matplotlib

- Matplotlib comes with a wide variety of plots. Plots helps to understand trends, patterns, and to make correlations.
- They’re typically instruments for reasoning about quantitative information.
- Now let’s check different categories of plots that Matplotlib provides.
  - Line plot
  - Histogram
  - Bar Chart
  - Scatter plot
  - Pie charts
  - Boxplot

Basic plots in Matplotlib - Line Plots

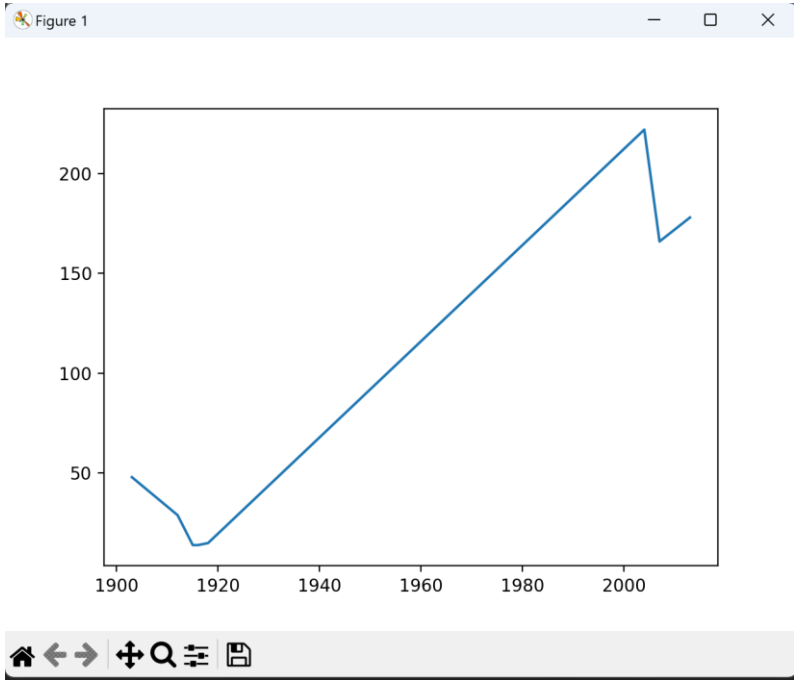
- A line plot is used to see the relationship between the x and y-axis.

- The plot() function in the Matplotlib library's Pyplot module is used to create a 2D hexagonal plot of the coordinates x and y. plot() will take various arguments like **plot(x, y, scalex, scaley, data, **kwargs)**.

  - **x, y** are the coordinates of the horizontal and vertical axis where x values are optional and its default value is range(len(y)).
  - **scalex, scaley** parameters are used to autoscale the x-axis or y-axis, and its default value is true.
  - ****kwargs** is used to specify the property like line label, linewidth, marker, color, etc.

Basic plots in Matplotlib - Line Plots

```
# importing matplotlib module
import matplotlib.pyplot as plt

years= [1903, 1912, 1915, 1916, 1918, 2004, 2007, 2013]
homeRuns = [48,29,14,14,15,222,166,178]

# Function to plot
plt.plot(years,homeRuns)

# function to show the plot
plt.show()
```



Basic plots in Matplotlib - Line Plots

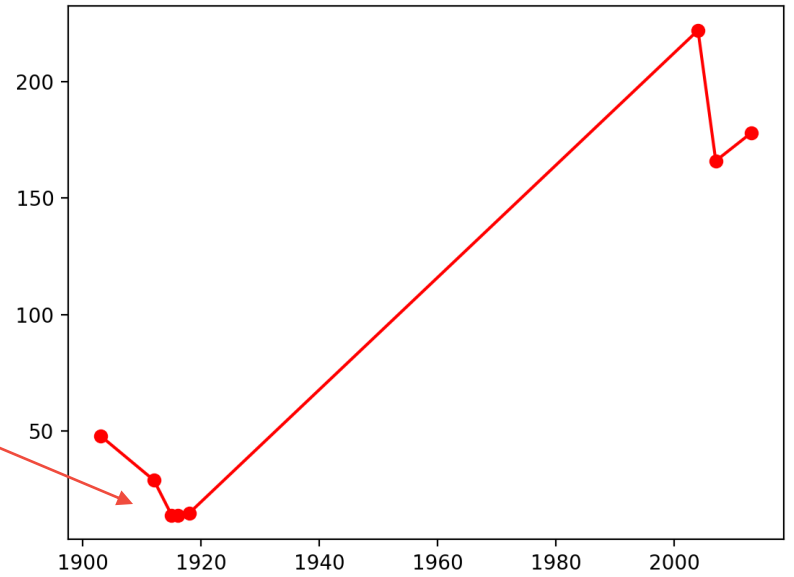
```
# importing matplotlib module
import matplotlib.pyplot as plt

years= [1903, 1912, 1915, 1916, 1918, 2004, 2007, 2013]
homeRuns = [48,29,14,14,15,222,166,178]

# Function to plot
plt.plot(years,homeRuns,'r')

# function to show the plot
plt.show()
```

Colour changed to red

Basic plots in Matplotlib - Line Plots

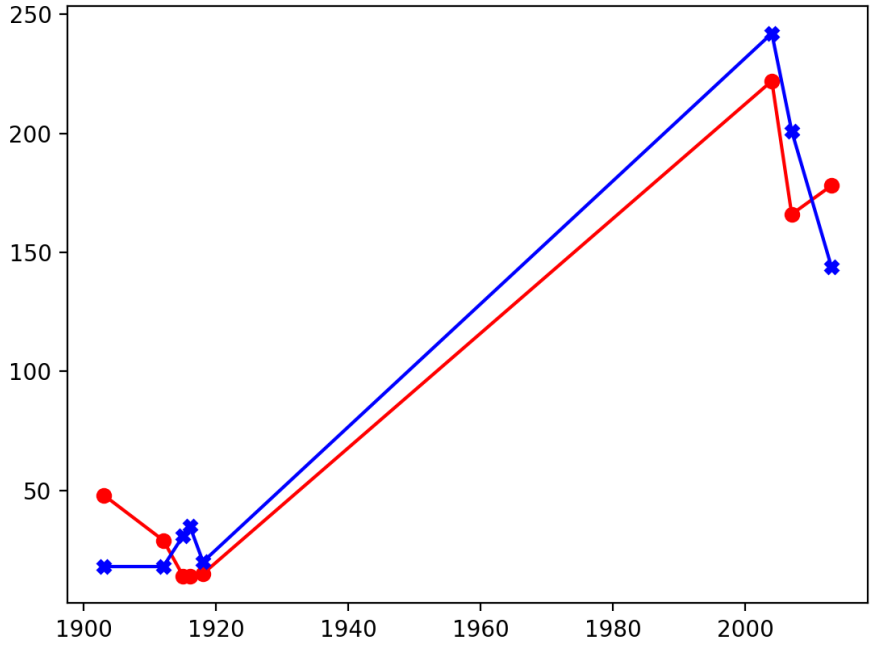
```
# importing matplotlib module
import matplotlib.pyplot as plt

years= [1903, 1912, 1915, 1916, 1918, 2004, 2007, 2013]
homeRuns = [48,29,14,14,15,222,166,178]

# Function to plot
plt.plot(years,homeRuns,'r-o')

# function to show the plot
plt.show()
```

Circle Marker



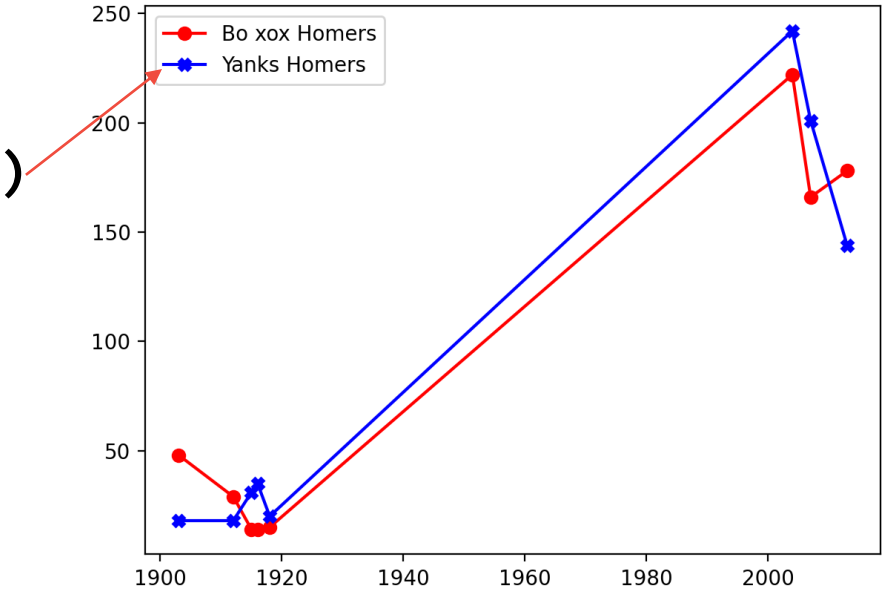
Basic plots in Matplotlib - Line Plots

```
# importing matplotlib module
import matplotlib.pyplot as plt

years= [1903, 1912, 1915, 1916, 1918, 2004, 2007, 2013]
yanks_hr=[18,18,31,35,20,242,201,144]
homeRuns = [48,29,14,14,15,222,166,178]

# Function to plot
plt.plot(years,homeRuns,'r-o')
plt.plot(years,yanks_hr,'b-X')

# function to show the plot
plt.show()
```



Basic plots in Matplotlib - Line Plots

```
# importing matplotlib module
import matplotlib.pyplot as plt

years= [1903, 1912, 1915, 1916, 1918, 2004, 2007, 2013]
yanks_hr=[18,18,31,35,20,242,201,144]
homeRuns = [48,29,14,14,15,222,166,178]

# Function to plot
plt.plot(years,homeRuns,'r-o',label='Bo xox Homers')
plt.plot(years,yanks_hr,'b-X',label='Yanks Homers')

# function to show the plot
plt.legend()
plt.show()
```



Ticks, Labels, and Legends

Chart Title

- Chart title is one of the most crucial elements in communicating what your chart is about at first glance.
- The interpretation and observation usually starts after reading the title.
- It's very easy to create a chart title in Python using matplotlib's pyplot module.
- Here is a sample code:

```
plt.title("Progess Grid")
```

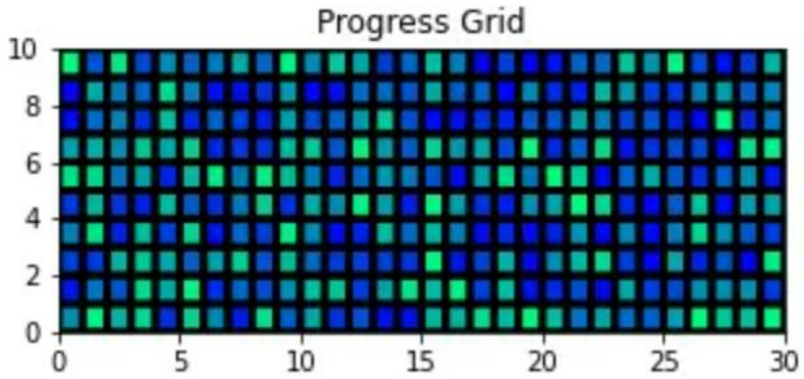


Chart Title

- **Title Color**: You can also adjust the color of the chart title:

```
plt.title("Progess Grid", color="blue")
```

- **Title Size** : Additionally if the chart is large your title with default size might appear too small and vice versa. So it can be necessary to adjust the title size as well:

```
plt.title("Progess Grid", color="blue", size=14)
```

Chart Title

- **Title Position**: Using loc parameter, you can adjust the position of a title on the chart. loc parameter takes values like left, center and right.

```
plt.title("Progess Grid", color="blue", size=14, loc="left")
```

- Title Vertical Margin: y parameter can be used to adjust title position vertically.
  - By default y is 1 and title will appear right above the chart.
  - A value like 1.1 will create more margin above the Python chart while a negative value can be used to move chart title below the chart.

```
plt.title("Progess Grid", color="blue", size=14, y=-0.3, loc="left")
```

Axis Label

- Axes labels can also be crucial (even more crucial than the chart title sometimes) for a chart to be meaningful and professional.
- We have too useful pyplot functions which can be used to define axes labels.
- xlabel()
  - It's quite simple to create axis labels using matplotlib and Python. All you have to do is run xlabel() function to add an x-axis label to your chart. Here is an example:

```
plt.xlabel("Day of Month")
```

- ylabel()
  - You can assign a y-axis label to your chart using ylabel() function.

```
plt.ylabel("Task Progress")
```

Axis Label

- ylabel()
  - With y-axis it might often be necessary to also adjust the rotation of the axis label.
  - This can easily be achieved using rotation parameter and assigning a rotation angle to it..

```
plt.ylabel("Task Progress", rotation=90)
```

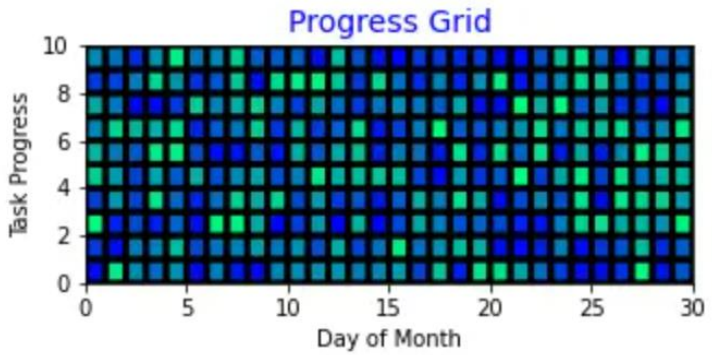
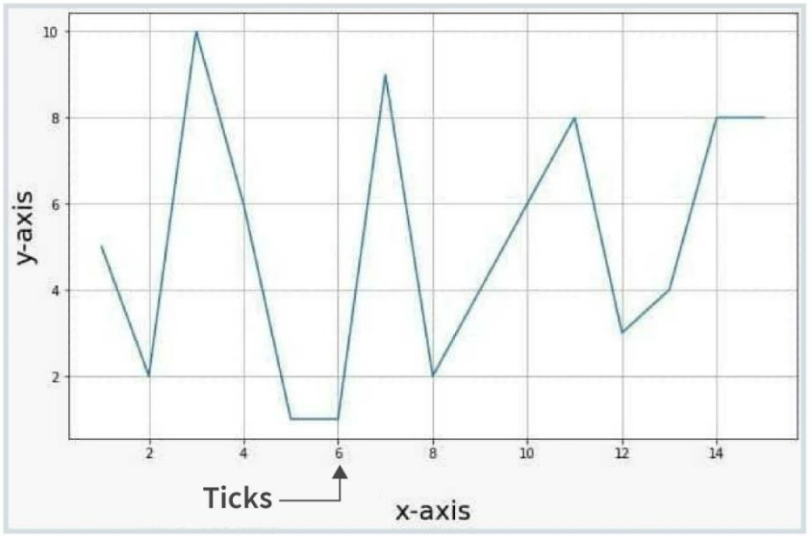


Chart example with rotated y-axis label

Ticks in Matplotlib

- Ticks are the value on the axis to show the coordinates on the graph.
- xticks() & yticks()
  - Axes Ticks can be adjusted using xticks() and yticks() methods in Python.

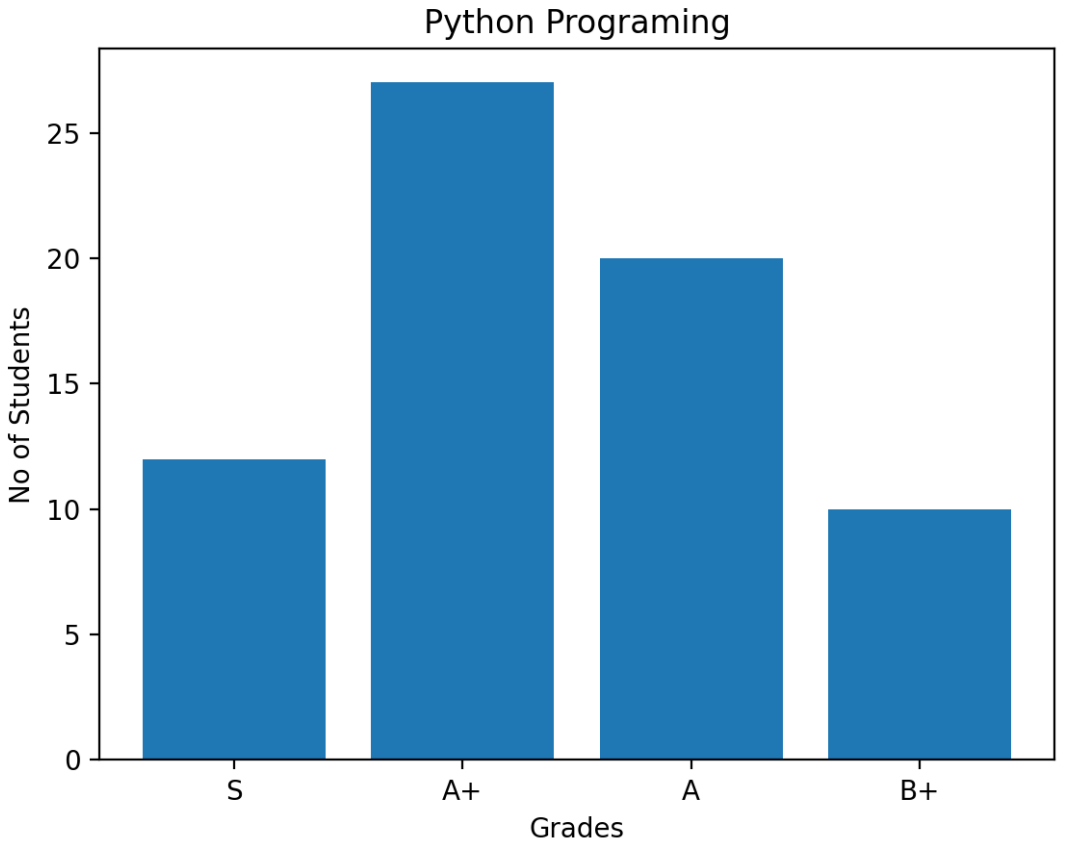


Basic plots in Matplotlib - Bar Plot

- Mainly barplot is used to show the relationship between the numeric and categoric values.
- Barcharts are plotted both vertically and horizontally and are plotted using the following line of code: **plt.bar(x,height,width,bottom,align)**
  - x: representing the coordinates of the x-axis
  - height: the height of the bars
  - width: width of the bars. Its default value is 0.8
  - bottom: It's optional. It is a y-coordinate of the bar its default value is None
  - align: center, edge its default value is center

Basic plots in Matplotlib - Bar Plot

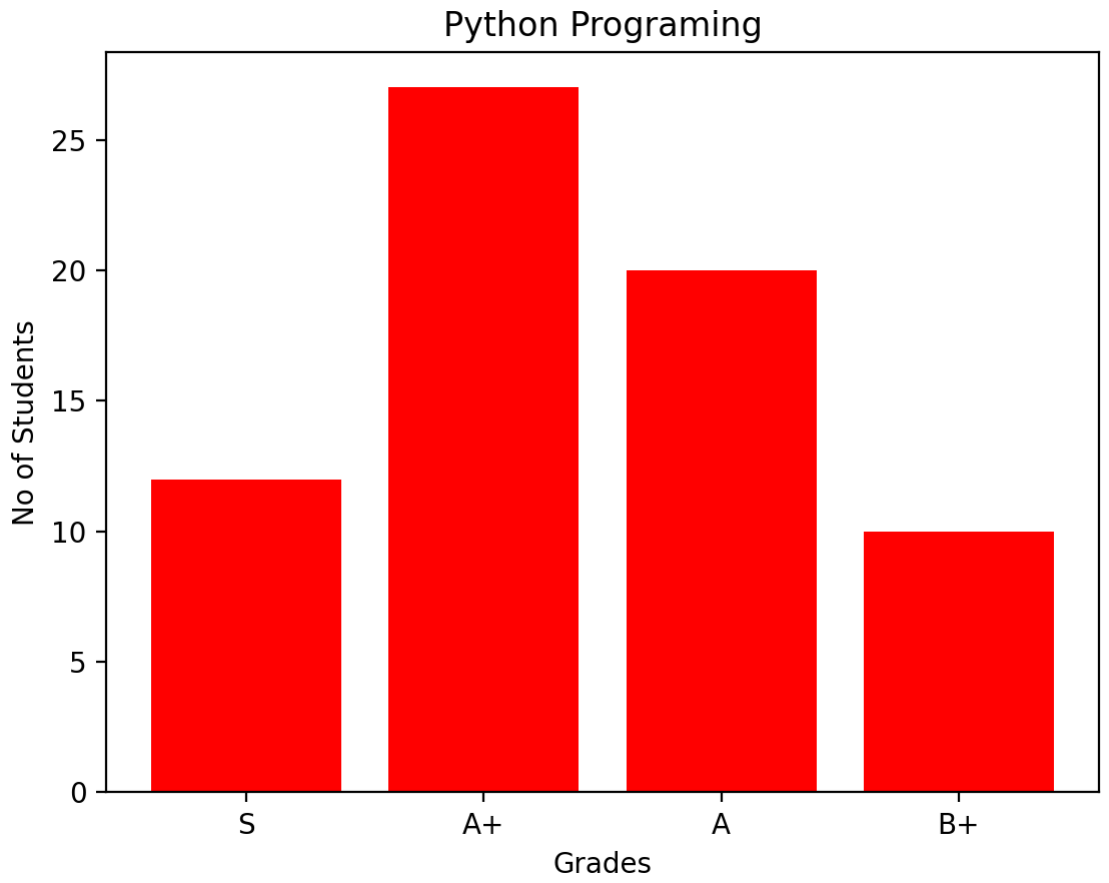
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["S", "A+", "A", "B+"])
y = np.array([12, 27, 20, 10])

plt.title("Python Programing")
plt.ylabel("No of Students")
plt.xlabel("Grades")

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



Basic plots in Matplotlib - Bar Plot

You can **change the color** of the bar chart. To do that, just add the color parameter.

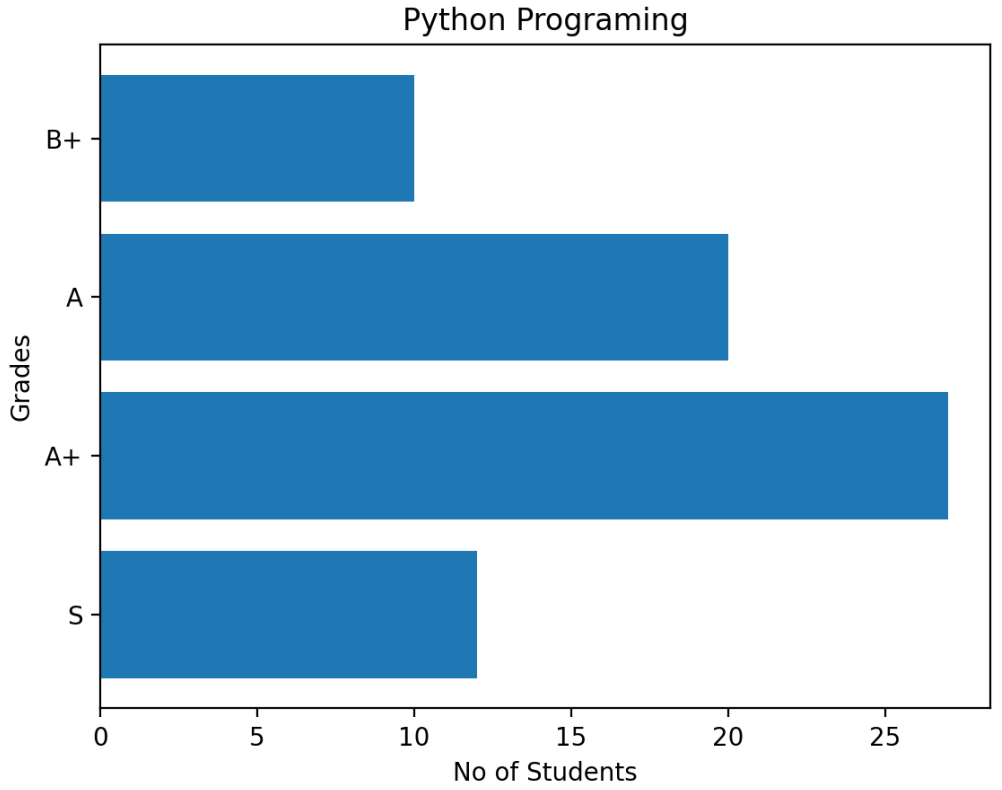
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["S", "A+", "A", "B+"])
y = np.array([12, 27, 20, 10])

plt.title("Python Programing")
plt.ylabel("No of Students")
plt.xlabel("Grades")

plt.bar(x,y,color="red")
plt.show()
```



Basic plots in Matplotlib - Bar Plot

Horizontal bar chart in Matplotlib: Matplotlib makes it very easy to add a horizontal bar chart by using the **plt.barh()**

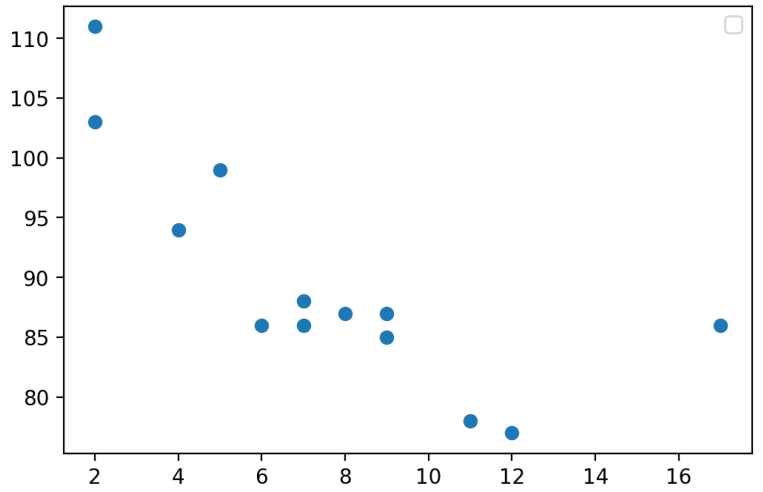
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["S", "A+", "A", "B+"])
y = np.array([12, 27, 20, 10])

plt.title("Python Programing")
plt.xlabel("No of Students")
plt.ylabel("Grades")

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



Basic plots in Matplotlib - Scatter Plot

- Scatter plots are used to show the relationships between the variables and use the dots for the plotting or it used to show the relationship between two numeric variables.
- The **scatter()** method in the **Matplotlib** library is used for plotting.

```
#create the x and y axis coordinates
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.legend()
plt.show()
```

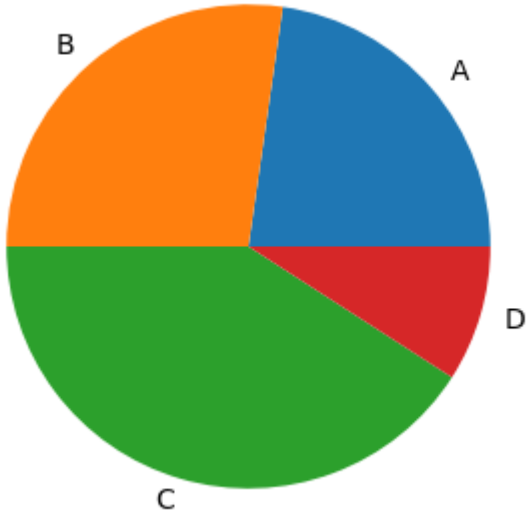


Basic plots in Matplotlib - Pie Chart

- A pie chart (or circular chart) is used to show the percentage of the whole. Hence it is used when we want to compare the individual categories with the whole.
- Pie() will take the different parameters such as:
  - x: Sequence of an array
  - labels: List of strings which will be the name of each slice in the pie chart
  - Autopct: It is used to label the wedges with numeric values. The labels will be placed inside the wedges. Its format is %1.2f%

Basic plots in Matplotlib - Pie Chart

```
#define the figure size
plt.figure(figsize=(7,7))
x = [25,30,45,10]
#labels of the pie chart
labels = ['A','B','C','D']
plt.pie(x, labels=labels)
plt.show()
```

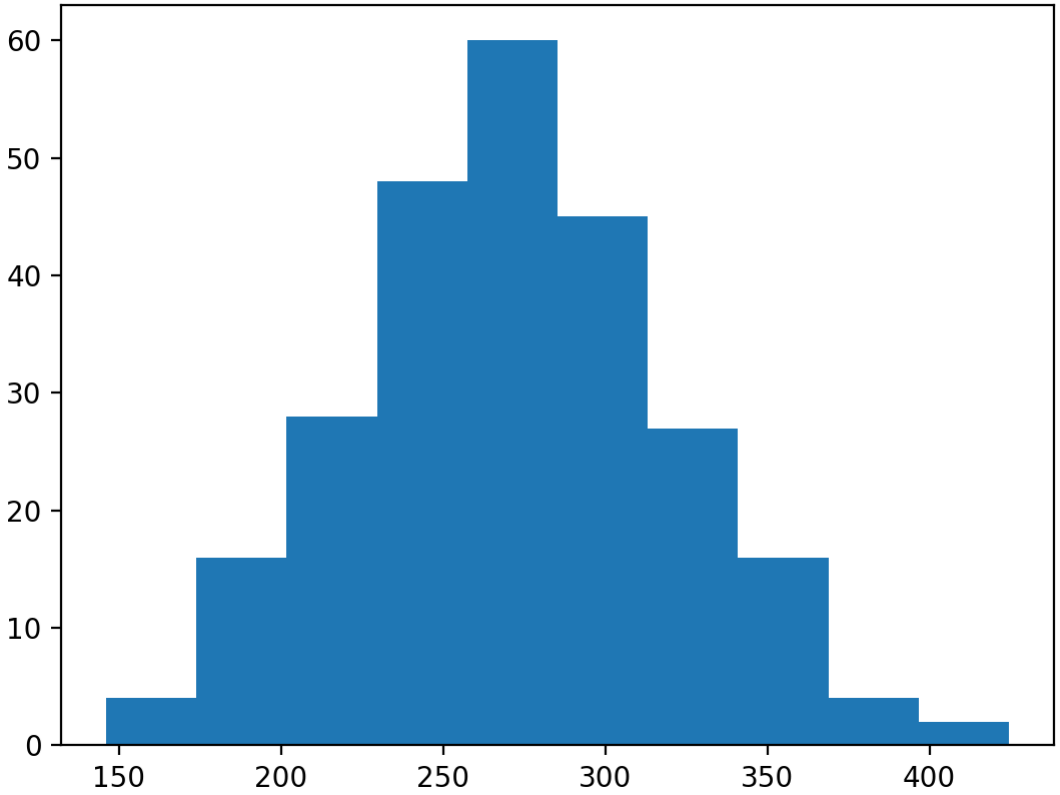


Basic plots in Matplotlib - Histogram Charts

- Histogram is used to show the frequency distribution of a small number of data points for a single variable.
- Histograms frequently divide data into different "bins" or "range groups" and count how many points are in each bin.
- We can use the **hist()** function to create histograms.
- This function will take an array of numbers to create a histogram, the array is sent into the function as an argument.

Basic plots in Matplotlib - Histogram Charts

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

x=np.random.normal(270,50,250)
plt.hist(x)
plt.show()
```



Questions

- Input average marks of 5 students and plot them against grades using a bar chart

- Input name and weight of 5 students and plot them in a line chart

- Input the number of students whose favourite subject is maths/physics/ chemistry/English and plot it in a pie graph.

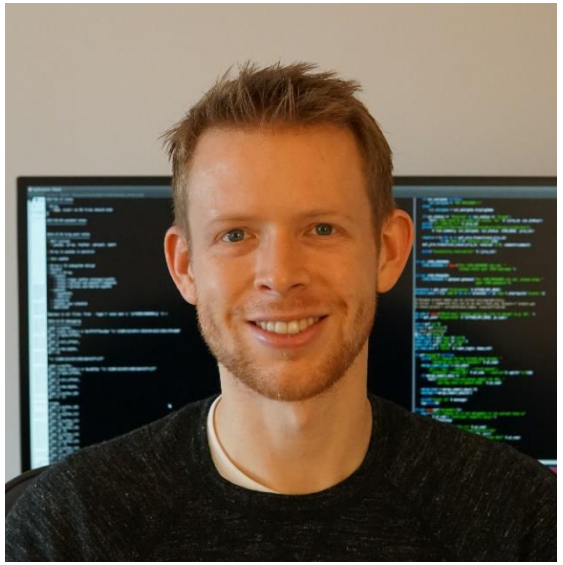
References

- https://www.simplilearn.com/tutorials/python-tutorial/matplotlib
- https://www.geeksforgeeks.org/python-introduction-matplotlib/
- https://www.cyberithub.com/how-to-plot-multiple-graphs-in-python-using-matplotlib/
- https://www.geeksforgeeks.org/plotting-multiple-bar-charts-using-matplotlib-in-python/

Pandas

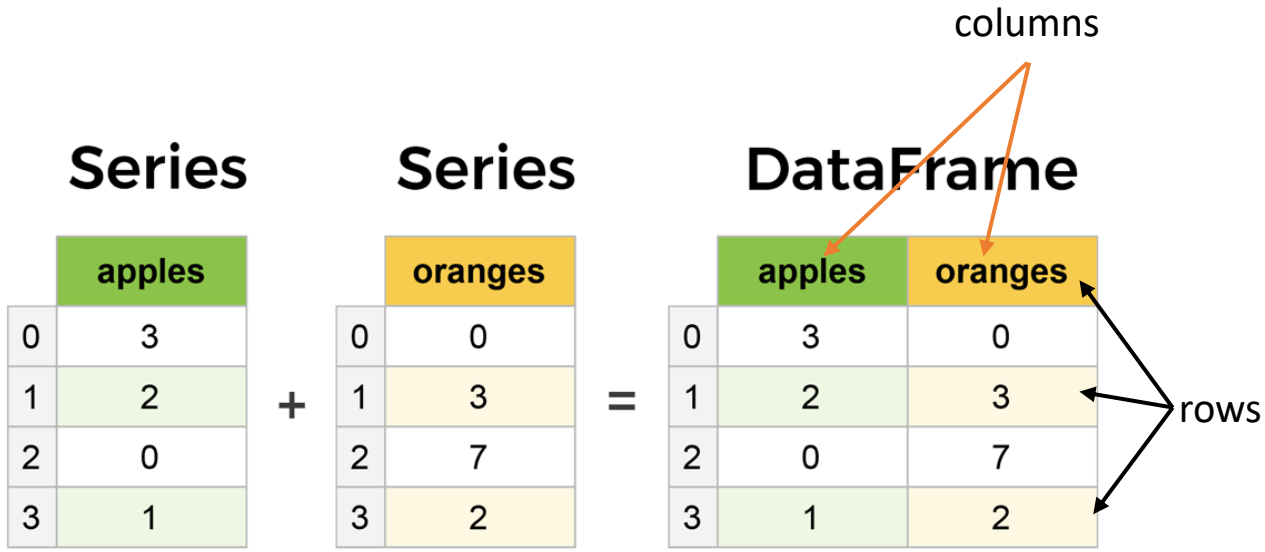
Python Pandas - Introduction

- Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures.
- The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.
- In 2008, developer Wes McKinney started developing pandas when in need of high performance, flexible tool for analysis of data.



Introduction to Data Structures

- Pandas deals with the following three data structures −
  - Series
  - DataFrame
  - Panel



What is a DataFrame?

- Pandas DataFrame is a 2-D labeled data structure with columns of potentially different type.
  - in-memory representation of an excel sheet via Python programming language
- Pandas DataFrame is similar to excel sheet and looks like this

	NAME	AGE	DESIGNATION	
1	a	20	VP	
2	b	27	CEO	
3	c	35	CFO	
4	d	55	VP	
5	e	18	VP	
6	f	21	CEO	
7	g	35	MD	

How to create a Pandas DataFrame?

```
import pandas as pd
my_dict = {
'name' : ["a", "b", "c", "d", "e","f", "g"],
'age' : [20,27, 35, 55, 18, 21, 35],
'designation': ["VP", "CEO", "CFO", "VP", "VP", "CEO", "MD"]
}

df = pd.DataFrame(my_dict)
print(df)
```

	name	age	designation
0	a	20	VP
1	b	27	CEO
2	c	35	CFO
3	d	55	VP
4	e	18	VP
5	f	21	CEO
6	g	35	MD

How to create a Pandas DataFrame?

- The Row Index
  - Since, we haven't provided any Row Index values to the DataFrame, it automatically generates a sequence (0...6) as row index.
  - To provide our own row index, we need to pass **index** parameter in the DataFrame(...) function as

```
df = pd.DataFrame(my_dict, index=[1,2,3,4,5,6,7])
```

How to create a Pandas DataFrame?

- The Row Index

```
import Pandas as pd
my_dict = {
'name' : ["a", "b", "c", "d", "e","f", "g"],
'age' : [20,27, 35, 55, 18, 21, 35],
'designation': ["VP", "CEO", "CFO", "VP", "VP", "CEO", "MD"]
}
df = pd.DataFrame(my_dict)
df = pd.DataFrame(my_dict, index=[1,2,3,4,5,6,7])
print(df)
```

	name	age	designation
1	a	20	VP
2	b	27	CEO
3	c	35	CFO
4	d	55	VP
5	e	18	VP
6	f	21	CEO
7	g	35	MD

How to Export Pandas DataFrame to CSV

- The easiest way to do this : `df.to_csv('file_name.csv')`

- If you want to export without the index, simply add `index=False;`

```
df.to_csv('file_name.csv', index=False)
```

How to Export Pandas DataFrame to CSV

```
import pandas as pd
my_dict = {'name' : ["a", "b", "c", "d", "e","f", "g"],
           'age' : [20,27, 35, 55, 18, 21, 35],
           'designation': ["VP", "CEO", "CFO", "VP",
"VP", "CEO", "MD"]}
df = pd.DataFrame(my_dict)
df.to_csv("text.csv",index=False)
print(df)
```

University Question

- Write Python program to write the following University topper data of CSE branch to a CSV file

Reg. No.	Name	Semester	College	CGPA
ABC123	Ganesh Kumar	S8	ABC	9.8
ECH265	John Mathew	S7	ECH	9.9
FET345	Reena K	S6	FET	9.7
GMT734	Adil M	S5	GMT	9.75

pandas read csv() : Importing Data

- To access data from the CSV file, we require a function **read_csv()** that retrieves data in the form of the data frame.

```
import pandas as pd

# Read the CSV file
employee = pd.read_csv("text.csv")
# View the first 5 rows
print(employee.head())
```

```
   name  age designation
0     a   20          VP
1     b   27         CEO
2     c   35         CFO
3     d   55          VP
4     e   18          VP
```

pandas read csv() : Importing Data

- The default behavior of pandas is to add an initial index to the dataframe returned from the CSV file it has loaded into memory.
- However, you can **explicitly specify what column to make as the index** to the read_csv function by setting the **index_col** parameter.

```
import pandas as pd

# Read the CSV file
employee =
pd.read_csv("text.csv",index_col=0)
# View the first 5 rows
print(employee.head())
```

name	age	designation
a	20	VP
b	27	CEO
c	35	CFO
d	55	VP
e	18	VP

pandas read csv() : Importing Data

- What if you only want to read specific columns into memory because not all of them are important?
- If you do know the columns you need, you can save time and memory by **passing a list**-like object to the **usecols** parameter of the read_csv function.

```
import pandas as pd
# Defining the columns to read
usecols = ['name','age']
# Read the CSV file
employee = employee = pd.read_csv("text.csv",index_col=0,
usecols=usecols)# View the first 5 rows
print(employee.head())
```

```
      age
name
a      20
b      27
c      35
d      55
e      18
```

pandas read csv() : Importing Data

- The “opposite” method of head() is tail(), which shows the last n (5 by default) rows of the dataframe object

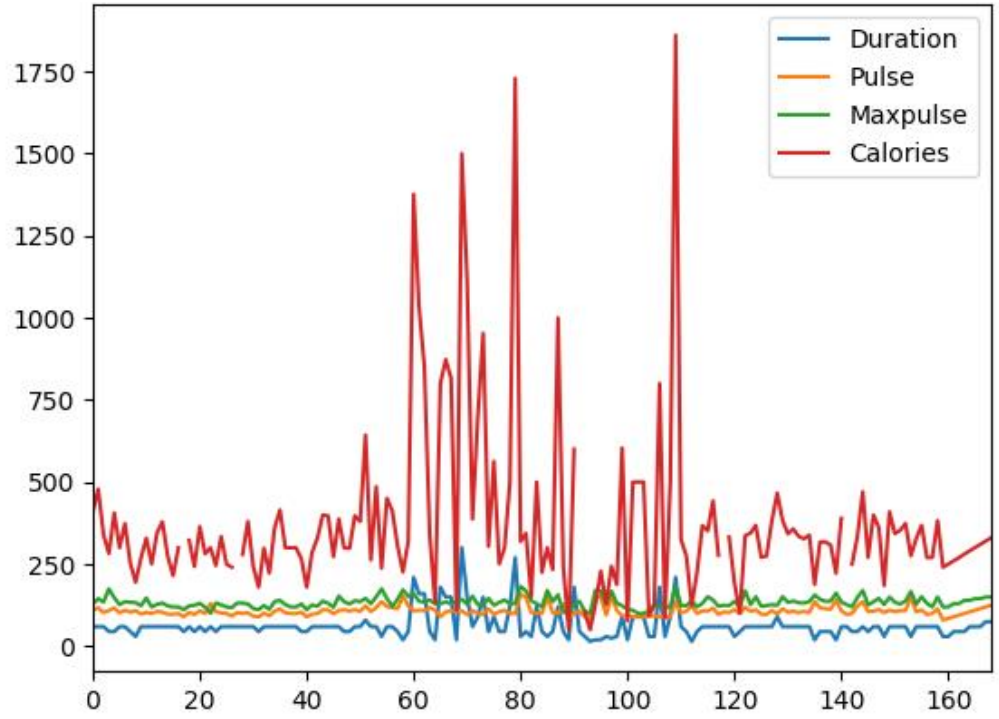
```
import pandas as pd
# Defining the columns to read
usecols = ['name','age']
# Read the CSV file
employee = employee = pd.read_csv("text.csv",index_col=0,
usecols=usecols)# View the first 5 rows
print(employee.tail())
```

```
      age
name
c      35
d      55
e      18
f      21
g      35
```

Pandas - Plotting

- 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.

```
import pandas as pd
import matplotlib.pyplot as plt

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

df.plot()

plt.show()
```



Pandas - Plotting

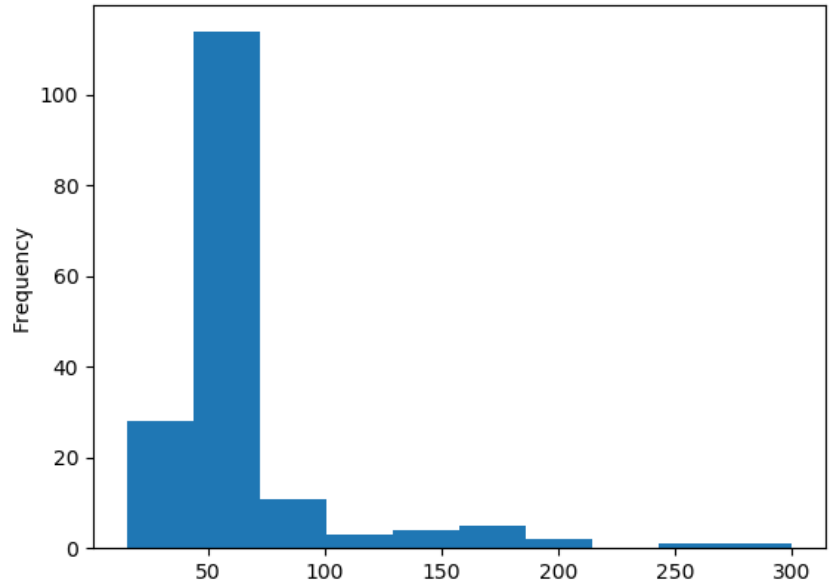
- Scatter Plot
  - Specify that you want a scatter plot with the kind argument: kind = 'scatter'
  - A scatter plot needs an x- and a y-axis.
  - In the example below we will use "Duration" for the x-axis and "Calories" for the y-axis.
  - Include the x and y arguments like this: x = 'Duration', y = 'Calories'

```
import pandas as pd
import matplotlib.pyplot as plt

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

df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')

plt.show()
```

Pandas - Plotting

- Histogram
  - Use the kind argument to specify that you want a histogram: kind = 'hist'
  - A histogram needs only one column.
  - In the example below we will use the "Duration" column to create the histogram:

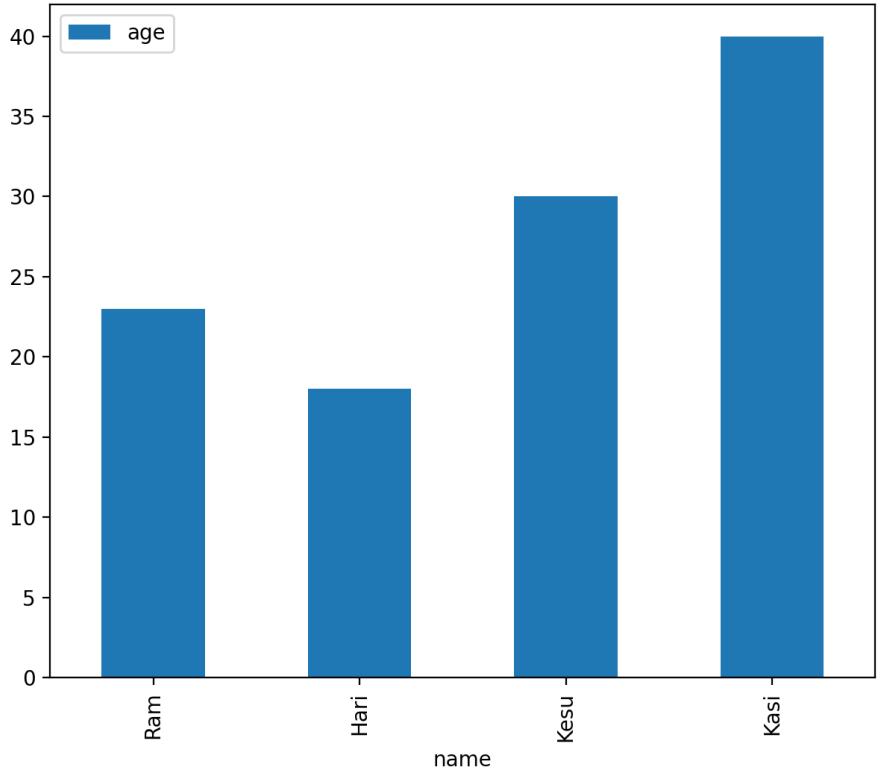
```
import pandas as pd
import matplotlib.pyplot as plt

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

df["Duration"].plot(kind = 'hist')

plt.show()
```



Pandas - Plotting

- Bar Graph
  - Use the kind argument to specify that you want a histogram: kind = ‘bar'

```
import pandas as pd
import matplotlib.pyplot as plt

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

df.plot(kind='bar',x = 'name', y = 'age')

plt.show()
```



References

- https://towardsdatascience.com/pandas-dataframe-a-lightweight-intro-680e3a212b96
- https://www.datacamp.com/tutorial/pandas-read-csv
- https://www.w3schools.com/python/pandas/pandas_plotting.asp