UNIT-2
PYTHON LIBRARIES SUITABLE FOR MACHINE LEARNING

Prepared by : Ritesh Prajapati

INTRODUCTION TO MACHINE LEARNING(4350702)

SEMESTER-5 DIPLOMA COMPUTER ENGINEERING

Topics

2.1 Numpy:

- Creating Array, Accessing Array, Stacking & Splitting,
- Maths Functions,
- Statistics Functions

2.2 Pandas:

- Series, DataFrames,
- Read CSV File, Cleaning Empty Cells, Cleaning Wrong Data, Removing Duplicates,
- Pandas Plotting

Prepared by : Ritesh Prajapati

2.3 Matplotlib:

- Pyplot.plot, show, labels, grid, bars, histogram, subplot, pie chart

2.4 sklearn:

- Key concepts and features
- Steps to Build a Model in Sklearn: Loading a Dataset, train_test_split

Numpy

- NumPy stands for Numerical Python.
- NumPy is a Python library used for working with arrays.
- NumPy was created in 2005 by Travis Oliphant. It is an open source.

- In Python we have lists that serve the purpose of arrays, but are slow to process.
- NumPy is 50x faster than traditional Python lists.
- NumPy arrays are stored at one continuous place in memory unlike lists.

Prepared by : Ritesh Prajapati

Difference NumPy and List

Numpy Array	**Python List**
It can contain similar datatypes.	It Contains different types of datatypes.
We need to Numpy Library to access Numpy Arrays.	It is built-in function of python.
In this Element wise operation is possible.	Element wise operation is not possible on the list.
By using numpy.array() we can create N-Dimensional array.	It is by default 1-dimensional.In some cases, we can create an N-Dimensional list. But it is a long process.
It requires smaller memory consumption as compared to Python List.	It requires more memory as compared to Numpy Array.
In this each item is stored in a sequential manner.	It stores item in random location of the memory.
It is faster as compared to list.	It is slow as compared to NumPy Array.

Prepared by : Ritesh Prajapati

Prepared by : Ritesh Prajapati

Features of NumPy

- Multidimensional Arrays:
  - 1D,2D and more – can be used for matrices and multidimensional data
- Efficient Mathematical Operations:
  - Element by element operation – faster execution
  - Vast collection of mathematical functions.
- Broadcasting
  - Arrays of different shapes and sizes adjusted to make operation simple
- Integration with Scientific libraries
  - Integrates with Pandas, Scipy, Scikit learn etc.
- Random Number Generation:
  - random number generator for creating arrays of random data, essential for simulations and statistical analysis.

Prepared by : Ritesh Prajapati

Features of NumPy

- Multidimensional Arrays:
  - 1D,2D and more – can be used for matrices and multidimensional data
- Efficient Mathematical Operations:
  - Element by element operation – faster execution
  - Vast collection of mathematical functions.
- Broadcasting
  - Arrays of different shapes and sizes adjusted to make operation simple
- Integration with Scientific libraries
  - Integrates with Pandas, Scipy, Scikit learn etc.
- Random Number Generation:
  - random number generator for creating arrays of random data, essential for simulations and statistical analysis.

Prepared by : Ritesh Prajapati

NumPy Arrays

- Installation
  - pip install numpy
- NumPy is usually imported under the **np** alias.
  - import numpy as np
- The array object in NumPy is called “ndarray” – for n dimensional array.
- Dimensions are called as **axes.**
  - [1,2,3] has one axes with 3 elements, so length is 3.
  - [[10,20,30] , [11,22,33]] has 2 axes, first axis has length of 2 and second has length of 3

Prepared by : Ritesh Prajapati

NumPy - Creating Arrays

- 1-D Array

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

Output

- [1 2 3 4 5]

- 2-D Array

Prepared by : Ritesh Prajapati

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

Output:

```
[[12 3]
 [4 5 6]]
```

NumPy - Creating Arrays

- 3-D Array

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

print(arr)
```

Prepared by : Ritesh Prajapati

Output

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

NumPy – Array Attributes

- **ndarray.ndim**
  - ndim represents the number of dimensions (axes)
  - 2-dimensional array [[3,4,6], [0,8,1]], value of ndim will be 2.
- **ndarray.shape**
  - shape is representing the size of the ndarray in each dimension.
  - 2-dimensional array [[3,4,6], [0,8,1]], value of shape will be (2,3)
- **ndarray.size**
  - size is the total number of elements in the ndarray.
  - 2-dimensional array [[3,4,6], [0,8,1]], shape is (2,3), size will be (2*3) = 6.
- **ndarray.dtype**
  - dtype tells the data type of the elements of a NumPy array.
  - 2-dimensional array [[3,4,6], [0,8,1]] , dtype will be int64
- **ndarray.itemsize**
  - itemsize returns the size (in bytes) of each element
  - 2-dimensional array [[3,4,6], [0,8,1]] , itemsize will be 8, size of integer (in bytes) is 8 bytes.

Prepared by : Ritesh Prajapati

NumPy - Creating Arrays

- Zeros
  - Creates array filled with all 0s
  - np.zeros(2,3) -> array of 6 elements with all 0s
- Ones
  - Creates array filled with all 1s
  - np.ones(2,3) -> array of 6 elements with all 1s
- Empty
  - Creates empty array (random data inside)
  - np.empty(2,3) -> array of 6 elements with all random data

Prepared by : Ritesh Prajapati

Prepared by : Ritesh Prajapati

NumPy - Creating Arrays

- Arrange
  - Returns an array with evenly spaced elements as per the interval.

  - **numpy.arange(*start*, *stop*, *step*)**

  - start : start of interval range. By default start = 0
  - stop : end of interval range
  - step : step size of interval. By default step size = 1, distance between two adjacent value

  - Example :

```
import numpy as np
print(np.arange(1, 2, 0.1))
```

  - Output :

```
[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9]
```

Prepared by : Ritesh Prajapati

NumPy - Creating Arrays

- Reshape
  - Reshaping means changing the shape of an array.
  - By reshaping we can add or remove dimensions or change number of elements in each dimension.
  - Reshape From 1-D to 2-D

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

  - Output

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

Prepared by : Ritesh Prajapati

NumPy - Creating Arrays

- Append
  - adds values at the end of an input array.

  numpy.append(arr, values, axis)
  - arr : Input array
  - values: To be appended to arr.
  - axis: The axis along which append operation is to be done.

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print np.append(a, [7,8,9])
print np.append(a, [[7,8,9]],axis = 0)
```

  - Output

```
[1 2 3 4 5 6 7 8 9]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Prepared by : Ritesh Prajapati

Prepared by : Ritesh Prajapati

Accessing Array: by index number

- Access 1-D Arrays

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

  - Output

```
2
```

- Access 2-D Arrays

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st row:', arr[0, 1])
```

  - Output

```
2nd element on 1st row : 2
```

- Negative Indexing
  - access an array from the end.

```
arr = np.array([[1,2,3], [6,7,8]])
print(arr[1, -1])
```

  - Output

```
7
```

Prepared by : Ritesh Prajapati

Prepared by : Ritesh Prajapati

Accessing Array: Slicing

- Taking elements from one given index to another given index.
- [start:end]
- [start:end:step]

- Start : Starting index(Inclusive). . Default 0
- End : Ending index(Exclusive). Default Last Element
- Step : Step of the slicing, Default 1

- Negative Slicing: Use the minus operator to refer to an index from the end:

Prepared by : Ritesh Prajapati

Accessing Array: Slicing

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

- print(arr[1:5]) -> [2 3 4 5]
- print(arr[:4]) -> [1 2 3 4]
- print(arr[4:]) -> [5 6 7]
- print(arr[1:5:2]) -> [2 4]
- print(arr[-3:-1]) -> [5 6]

Prepared by : Ritesh Prajapati

Accessing Array: Slicing

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

- print(arr[1:5]) -> [2 3 4 5]
- print(arr[:4]) -> [1 2 3 4]
- print(arr[4:]) -> [5 6 7]
- print(arr[1:5:2]) -> [2 4]
- print(arr[-3:-1]) -> [5 6]

- Question : Reverse Array ?
- print(arr[::-1])

Prepared by : Ritesh Prajapati

Accessing Array: Joining and Stacking

- Joining means putting contents of two or more arrays in a single array.

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

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

- Joining 2-D arrays along rows (axis=1) and cols (axis=0)

```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr)

arr = np.concatenate((arr1, arr2), axis=0)
print(arr)
```

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

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

Prepared by : Ritesh Prajapati

Accessing Array: Stacking

- Stacking Along Rows

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

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

- Stacking Along Cols

```
arr = np.vstack((arr1, arr2))
print(arr)
```

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

Prepared by : Ritesh Prajapati

- Stacking Along Height (depth)

```
arr = dstack ((arr1, arr2))
print(arr)
```

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

Accessing Array: Splitting

- Splitting breaks one array into multiple.
- Split the array in 3 parts:

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

```
[array([1, 2]),
array([3, 4]),
array([5, 6])]
```

Prepared by : Ritesh Prajapati

- Access Splitted array

```
print(newarr[0])
print(newarr[1])
print(newarr[2])
```

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

Accessing Array: Splitting

- Splitting 2-D Arrays along rows.

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

```
[array([[1],[4]]),
array([[2], [5]]),
array([[3], [6]])]
```

- Using hsplit()

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

```
[array([[1],[4]]),
array([[2], [5]]),
array([[3], [6]])]
```

- Splitting 2-D Arrays along cols.

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = np.array_split(arr, 2, axis=0)
print(newarr)
```

```
[array([[1, 2, 3]]),
array([[4, 5, 6]])]
```

- Using vsplit()

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

```
[array([[1, 2, 3]]),
array([[4, 5, 6]])]
```

Prepared by : Ritesh Prajapati

Arithmetic Operations Functions

add(x1, x2)	Add arguments element-wise.
reciprocal(x)	Return the reciprocal of the argument, element-wise.
positive(x)	Numerical positive, element-wise.
negative(x)	Numerical negative, element-wise.
multiply(x1, x2)	Multiply arguments element-wise.
divide(x1, x2)	Divide arguments element-wise.
power(x1, x2)	First array elements raised to powers from second array, element-wise.
subtract(x1, x2)	Subtract arguments, element-wise.
true_divide(x1, x2)	Divide arguments element-wise.
floor_divide(x1, x2)	Return the largest integer smaller or equal to the division of the inputs.
float_power(x1, x2)	First array elements raised to powers from second array, element-wise.
fmod(x1, x2)	Returns the element-wise remainder of division.
mod(x1, x2)	Returns the element-wise remainder of division.
modf(x[, out1, out2])	Return the fractional and integral parts of an array, element-wise.
remainder(x1, x2)	Returns the element-wise remainder of division.
divmod(x1, x2[, out1, out2])	Return element-wise quotient and remainder simultaneously.

Prepared by : Ritesh Prajapati

Exponents and Log Functions

exp(x)	Calculate the exponential of all elements in the input array.
expm1(x)	Calculate exp(x) - 1 for all elements in the array.
exp2(x)	Calculate 2**p for all p in the input array.
log(x)	Natural logarithm, element-wise.
log10(x)	Return the base 10 logarithm of the input array, element-wise.
log2(x)	Base-2 logarithm of x.
log1p(x)	Return the natural logarithm of one plus the input array, element-wise.
logaddexp(x1, x2)	Logarithm of the sum of exponentiations of the inputs.
logaddexp2(x1, x2)	Logarithm of the sum of exponentiations of the inputs in base-2.

Prepared by : Ritesh Prajapati

Prepared by : Ritesh Prajapati

Rounding Functions

round(a[, decimals])	Evenly round to the given number of decimals.
around(a[, decimals])	Round an array to the given number of decimals.
rint(x)	Round elements of the array to the nearest integer.
fix(x)	Round to nearest integer towards zero.
floor(x)	Return the floor of the input, element-wise.
ceil(x)	Return the ceiling of the input, element-wise.
trunc(x)	Return the truncated value of the input, element-wise.

Prepared by : Ritesh Prajapati

Trigonometric Functions

sin(x)	Trigonometric sine, element-wise.
cos(x)	Cosine element-wise.
tan(x)	Compute tangent element-wise.
arcsin(x)	Inverse sine, element-wise.
arccos(x)	Trigonometric inverse cosine, element-wise.
arctan(x)	Trigonometric inverse tangent, element-wise.
degrees(x)	Convert angles from radians to degrees.
radians(x)	Convert angles from degrees to radians.
deg2rad(x)	Convert angles from degrees to radians.
rad2deg(x)	Convert angles from radians to degrees.

Prepared by : Ritesh Prajapati

Hyperbolic Functions

sinh(x)	Hyperbolic sine, element-wise.
cosh(x)	Hyperbolic cosine, element-wise.
tanh(x)	Compute hyperbolic tangent element-wise.
arcsinh(x)	Inverse hyperbolic sine element-wise.
arccosh(x)	Inverse hyperbolic cosine, element-wise.
arctanh(x)	Inverse hyperbolic tangent element-wise.

Prepared by : Ritesh Prajapati

Sums, products, differences… Functions

prod(a[, axis, dtype, out, keepdims, ...])	Return the product of array elements over a given axis.
sum(a[, axis, dtype, out, keepdims, ...])	Sum of array elements over a given axis.
nanprod(a[, axis, dtype, out, keepdims, ...])	Return the product of array elements over a given axis treating Not a Numbers (NaNs) as ones.
nansum(a[, axis, dtype, out, keepdims, ...])	Return the sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
cumprod(a[, axis, dtype, out])	Return the cumulative product of elements along a given axis.
cumsum(a[, axis, dtype, out])	Return the cumulative sum of the elements along a given axis.
nancumprod(a[, axis, dtype, out])	Return the cumulative product of array elements over a given axis treating Not a Numbers (NaNs) as one.
nancumsum(a[, axis, dtype, out])	Return the cumulative sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
diff(a[, n, axis, prepend, append])	Calculate the n-th discrete difference along the given axis.
ediff1d(ary[, to_end, to_begin])	The differences between consecutive elements of an array.
gradient(f, *varargs[, axis, edge_order])	Return the gradient of an N-dimensional array.
cross(a, b[, axisa, axisb, axisc, axis])	Return the cross product of two (arrays of) vectors.

Prepared by : Ritesh Prajapati

Extrema Finding Functions

maximum(x1, x2, /[, out, where, casting, ...])	Element-wise maximum of array elements.
max(a[, axis, out, keepdims, initial, where])	Return the maximum of an array or maximum along an axis.
amax(a[, axis, out, keepdims, initial, where])	Return the maximum of an array or maximum along an axis.
fmax(x1, x2, /[, out, where, casting, ...])	Element-wise maximum of array elements.
nanmax(a[, axis, out, keepdims, initial, where])	Return the maximum of an array or maximum along an axis, ignoring any NaNs.
minimum(x1, x2, /[, out, where, casting, ...])	Element-wise minimum of array elements.
min(a[, axis, out, keepdims, initial, where])	Return the minimum of an array or minimum along an axis.
amin(a[, axis, out, keepdims, initial, where])	Return the minimum of an array or minimum along an axis.
fmin(x1, x2, /[, out, where, casting, ...])	Element-wise minimum of array elements.
nanmin(a[, axis, out, keepdims, initial, where])	Return minimum of an array or minimum along an axis, ignoring any NaNs.

Prepared by : Ritesh Prajapati

Prepared by : Ritesh Prajapati

Miscellaneous Functions

convolve(a, v[, mode])	Returns the discrete, linear convolution of two one-dimensional sequences.
clip(a, a_min, a_max[, out])	Clip (limit) the values in an array.
sqrt(x, /[, out, where, casting, order, ...])	Return the non-negative square-root of an array, element-wise.
cbrt(x, /[, out, where, casting, order, ...])	Return the cube-root of an array, element-wise.
square(x, /[, out, where, casting, order, ...])	Return the element-wise square of the input.
absolute(x, /[, out, where, casting, order, ...])	Calculate the absolute value element-wise.
fabs(x, /[, out, where, casting, order, ...])	Compute the absolute values element-wise.
sign(x, /[, out, where, casting, order, ...])	Returns an element-wise indication of the sign of a number.
heaviside(x1, x2, /[, out, where, casting, ...])	Compute the Heaviside step function.
nan_to_num(x[, copy, nan, posinf, neginf])	Replace NaN with zero and infinity with large finite numbers (default behaviour) or with the numbers defined by the user

Prepared by : Ritesh Prajapati

Pandas

- Pandas is a Python library used to analyze data.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- Created by Wes McKinney in 2008.

- Pandas allows us to analyze big data and make conclusions.
- 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.

Prepared by : Ritesh Prajapati

- Installation of Pandas

  pip install pandas
- Importing Pandas

  import pandas as pd

Pandas features

- Great data handling:
  Series and DataFrames, both are efficient and fast ways of managing data and exploring it.
- Data Import/Export
  Support for import and export of data from different file formats.
- Data Manipulation
  - Contains wide range of functions for data manipulation.
- Handling missing data
  - Provides functions for handling missing data and null values.
- Time series Analysis
  - Provide functions for Time Series data analysis.(time based indexing,resampling,moving window)
- Data Visualization
  - Creates plots, charts,graphs used to visualize data
- GroupBy Operations
  - Grouping data as per criteria needed
- Integration with Other Libraries
  - Can be integrated with other libraries like NumPy,Matplotlib etc

Prepared by : Ritesh Prajapati

Prepared by : Ritesh Prajapati

Creating Series

- It is one-dimensional array to store homogenous data (column only).
- From python list

```
import pandas as pd
a = [1,7,2,9,6]
var1 = pd.Series(a)
print(var1)
```

```
0    1
1    7
2    2
3    9
4    6
dtype: int64
```

Prepared by : Ritesh Prajapati

- From NumPy

```
import pandas as pd
import numpy as np
a = np.array([1,7,2,9,6])
var1 = pd.Series(a)
print(var1)
```

```
0    1
1    7
2    2
3    9
4    6
dtype: int64
```


Prepared by : Ritesh Prajapati

Creating Series

- Passing Index

```
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
series= pd.Series(data,index=[100,101,102,103])
print(series)
```

```
100  a
101  b
102  c
103  d
dtype: object
```

- Create a Series from dictionary

Prepared by : Ritesh Prajapati

```
import pandas as pd
import numpy as np
data = {'a' : 0.0, 'b' : 1.1, 'c' : 2.2}
series = pd.Series(data)
print(series)
```

```
a 0.0
b 1.1
c 2.2
dtype: float64
```


Prepared by : Ritesh Prajapati

Creating Dataframes

- It is two-dimensional array to store heterogeneous data(rows and columns)
- From python list

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data)
print(df)
```

```
0    Alex     10
1    Bob      12
2    Clarke   13
```

Prepared by : Ritesh Prajapati

- From python list column names

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print(df)
```

```
    Name     Age
0    Alex     10
1    Bob      12
2    Clarke   13
```


Prepared by : Ritesh Prajapati

Creating Dataframes

- It is two-dimensional array to store heterogeneous data(rows and columns)
- From Numpy Array

```
import pandas as pd
import numpy as np
data = np.array([['Alex',10],['Bob',12],['Clarke',13]])
df = pd.DataFrame(data,columns=['Name','Age'])
print(df)
```

	Name	Age
0	Alex	10
1	Bob	12
2	Clarke	13

Prepared by : Ritesh Prajapati

- From Dictionary

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve'],'Age':[28,34,29]}
df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])
print(df)
```

	Age	Name
rank1	28	Tom
rank2	34	Jack
rank3	29	Steve

Prepared by : Ritesh Prajapati

Reading CSV files

pandas.read_csv(filepath_or_buffer, sep=‘ ,’, header='infer', index_col=None, usecols=None, dtype=None, engine=None, nrows=None, skiprows=None)

- filepath_or_buffer : str, path object or file-like object
- header : int, Sequence of int, ‘infer’ or None, default ‘infer’
- sep : str, default ‘,’
- index_col : Sequence of index, optional
- usecols : retrieve selected columns
- nrows : no of rows to display
- skiprows : skip rows

Prepared by : Ritesh Prajapati

Program

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

Duration	Date	Pulse	Maxpulse	Calories
0	60 '2020/12/01'	110	130	409.1
1	60 '2020/12/02'	117	145	479.0
2	60 '2020/12/03'	103	135	340.0
3	45 '2020/12/04'	109	175	282.4
4	45 '2020/12/05'	117	148	406.0
5	60 '2020/12/06'	102	127	300.0
6	60 '2020/12/07'	110	136	374.0
7	450 '2020/12/08'	104	134	253.3
8	30 '2020/12/09'	109	133	195.1
9	60 '2020/12/10'	98	124	269.0
10	60 '2020/12/11'	103	147	329.3
11	60 '2020/12/12'	100	120	250.7
12	60 '2020/12/12'	100	120	250.7
13	60 '2020/12/13'	106	128	345.3
14	60 '2020/12/14'	104	132	379.3
15	60 '2020/12/15'	98	123	275.0
16	60 '2020/12/16'	98	120	215.2
17	60 '2020/12/17'	100	120	300.0
18	45 '2020/12/18'	90	112	NaN
19	60 '2020/12/19'	103	123	323.0
20	45 '2020/12/20'	97	125	243.0
21	60 '2020/12/21'	108	131	364.2
22	45 NaN	100	119	282.0
23	60 '2020/12/23'	130	101	300.0
24	45 '2020/12/24'	105	132	246.0
25	60 '2020/12/25'	102	126	334.5
26	60 2020/12/26	100	120	250.0
27	60 '2020/12/27'	92	118	241.0
28	60 '2020/12/28'	103	132	NaN
29	60 '2020/12/29'	100	132	280.0
30	60 '2020/12/30'	102	129	380.3
31	60 '2020/12/31'	92	115	243.0

Prepared by : Ritesh Prajapati

Data cleaning

- Data cleaning means fixing bad data in your data set.

- Empty cells ("Date" in row 22, and "Calories" in row 18 and 28).
- Wrong format ("Date" in row 26).
- Wrong data ("Duration" in row 7).
- Duplicates (row 11 and 12).

Prepared by : Ritesh Prajapati

Cleaning Empty Cells

- 1. Remove rows that contain empty cells. (dropna)

```
import pandas as pd
df = pd.read_csv('data.csv')
new_df = df.dropna()
print(new_df)
```

- 2. Insert a *new* value in empty cells. (fillna)

```
import pandas as pd
df = pd.read_csv('data.csv')
new_df = df.fillna(130)
print(new_df)
```

Prepared by : Ritesh Prajapati

Cleaning Wrong Data

Replacing Values. (loc)

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

#fetching with loc
result = df.loc[‘7', 'Duration ']
print(result)

#replacing with loc
df.loc[7,'Duration'] = 45
print(df)

#looping with loc
for x in df.index:
  if df.loc[x, "Duration"] > 120:
     df.loc[x, "Duration"] = 120
print(df)
```

Prepared by : Ritesh Prajapati

Removing Duplicates

Using drop_duplicates

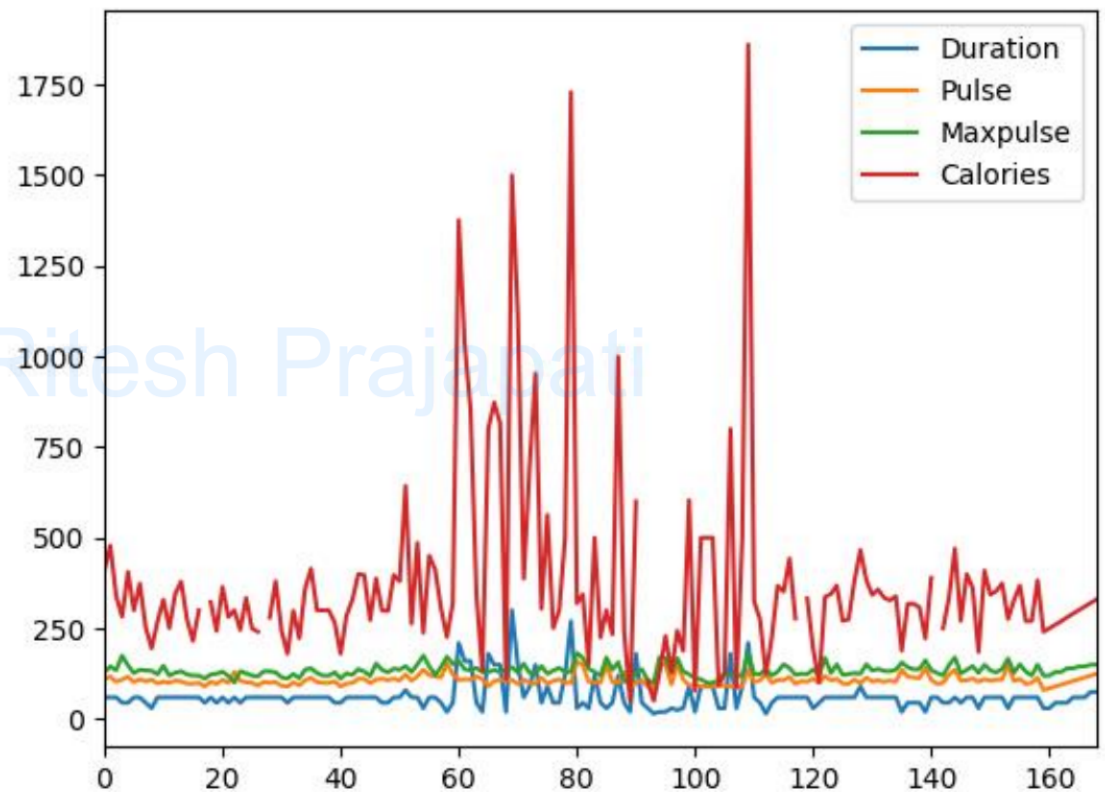
```
import pandas as pd
df = pd.read_csv('data.csv')

#remove all duplicates
new_df = df.drop_duplicates()
print(new_df)

#remove duplicates and keep last
new_df = df.drop_duplicates(keep='last')
print(new_df)
```

Prepared by : Ritesh Prajapati

Plotting

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

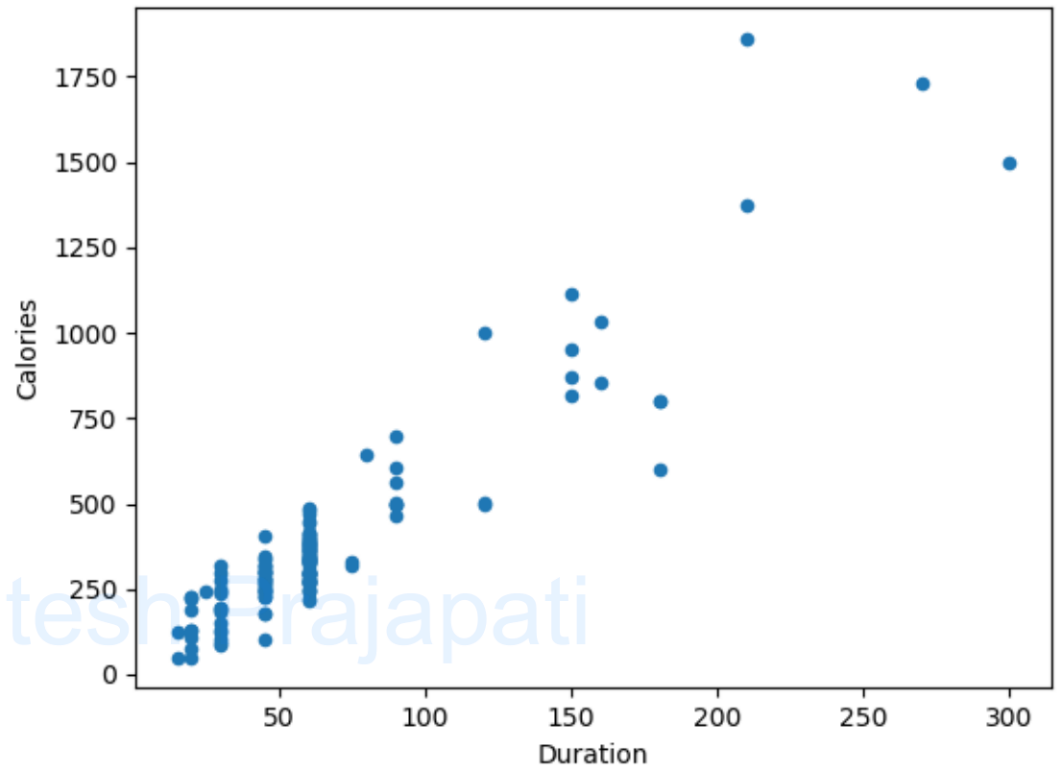
```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
df.plot()
plt.show()
```



Prepared by : Ritesh Prajapati

Plotting

- Scatter Plot – (kind = 'scatter‘)
- A scatter plot needs an x- and a y-axis.

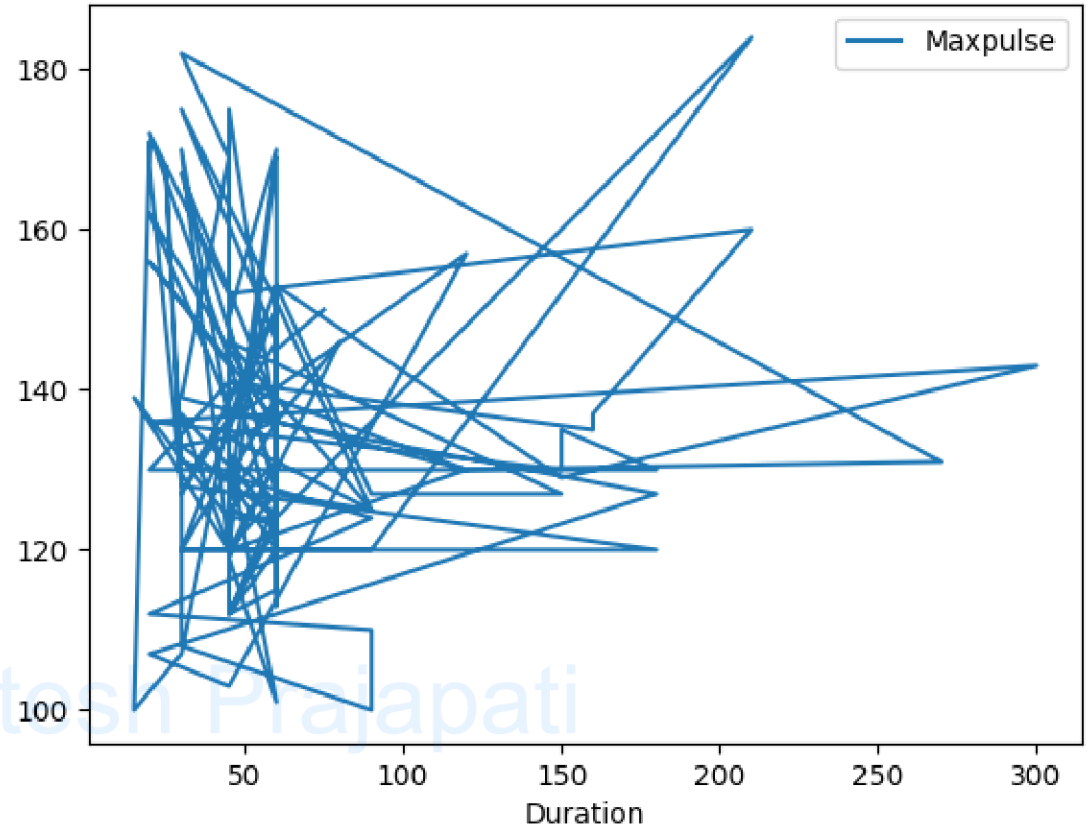



```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')
plt.show()
```

Prepared by : Ritesh Prajapati

Plotting

- Line Plot – (kind = line‘)
- A line plot needs an x- and a y-axis.



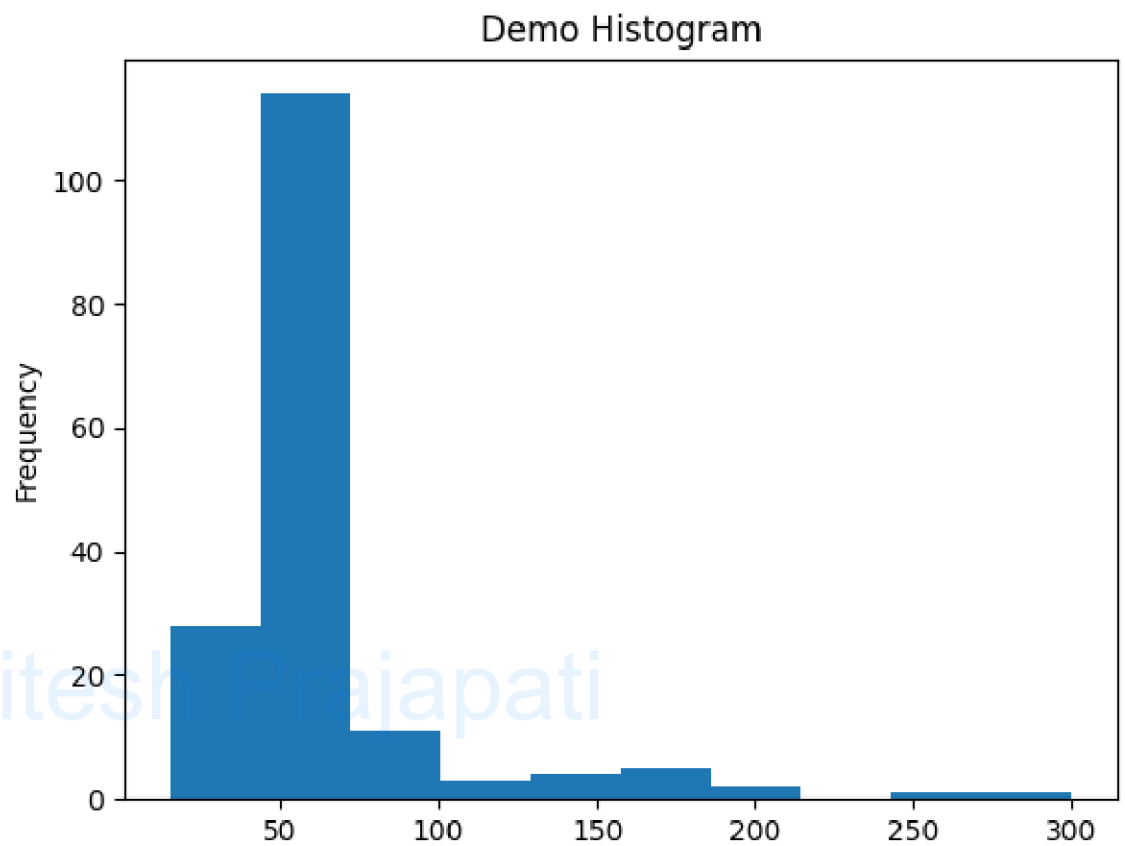

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
df.plot(kind = line', x = 'Duration', y = 'Calories')
plt.show()
```

Prepared by : Ritesh Prajapati

Plotting

- Histogram– (kind = hist ‘)
- A histogram needs only one column.

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
df["Duration"].plot(kind = 'hist')
plt.title('Demo Histogram')
plt.show()
```



Prepared by : Ritesh Prajapati

Matplotlib

- Matplotlib is a graph plotting library in python (visualization utility.)
- Matplotlib was created by John D. Hunter.
- Matplotlib is open source and we can use it freely.

- Installation of Matplotlib

  pip install matplotlib

- Importing Matplotlib

  import matplotlib as mlb

  import matplotlib.pyplot as plt

- Most of the Matplotlib utilities lies under the pyplot submodule

Prepared by : Ritesh Prajapati

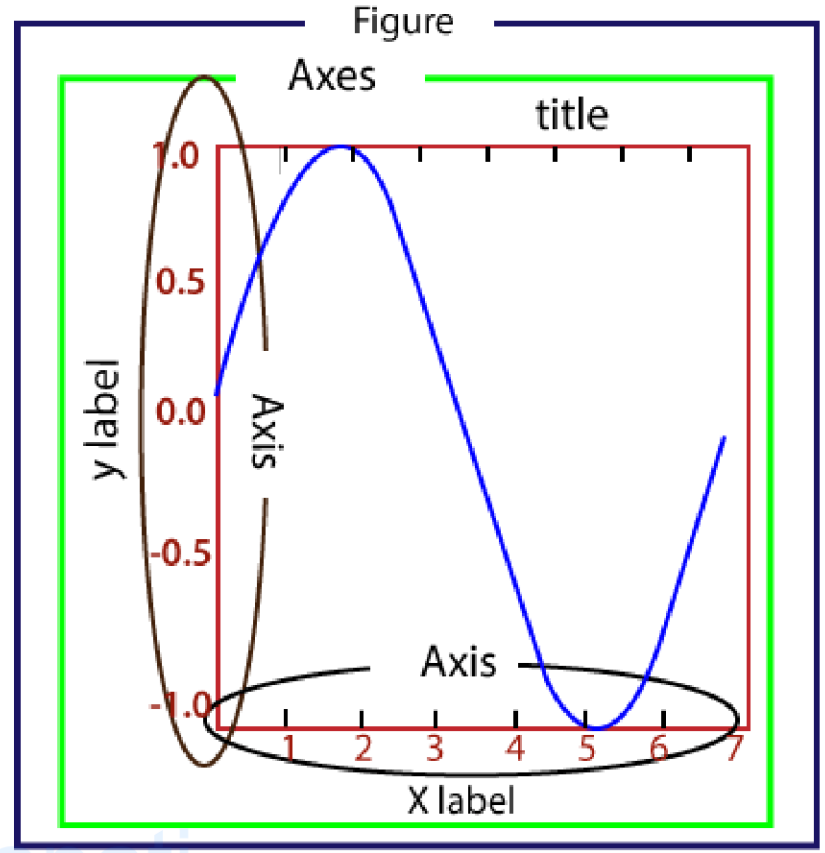
Matplotlib

- Plotting Functions:
  Functions to create lineplots,scatter plots, bar plots,histograms,piecharts etc and many more.
- Support for multiple Formats
  Rendering of plots in different file formats(PNG,JPG,PDF etc).
- Object Oriented Interface
  - OO interface to have control over elements of plot.
- Customization Options
  - Able to modify colors, line styles, markers,fonts etc.
- 3D Plotting
  - 3D visualizations also possible
- Animations
  - Able to create animated plots and visualizations
- Integration with Other Libraries
  - Can be integrated with other libraries like NumPy,Pandas etc

Prepared by : Ritesh Prajapati

Prepared by : Ritesh Prajapati

General Concept of Matplotlib



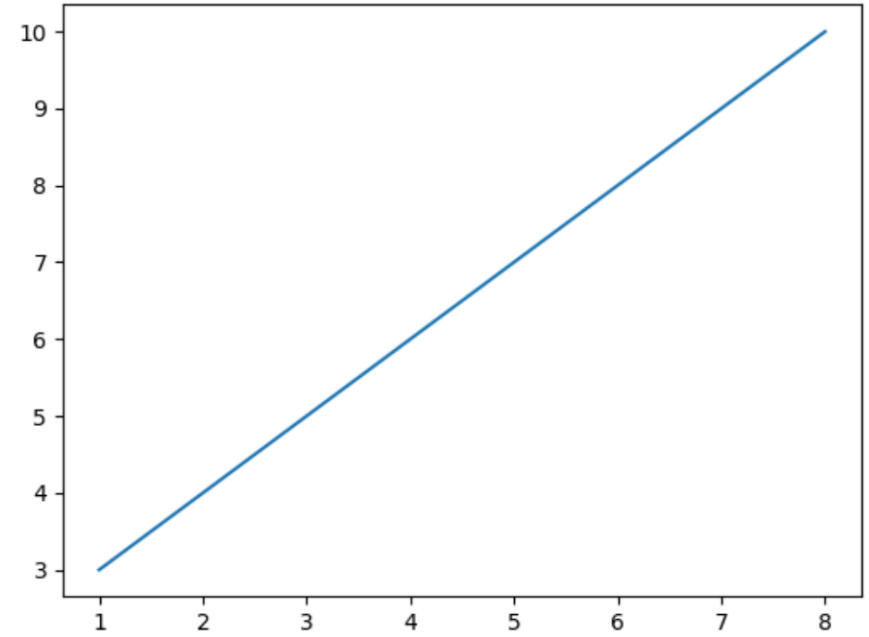
- **Figure:**
  - Represents entire window or page which hold one or more axes (plots)
- **Axes:**
  - Axes are actual plots where data is visualized
  - A Figure can contain several Axes.
  - Each Axes is comprised of a title, an x-label, and a y-label.
- **Axis:**
  - Axis are the number of objects(like line) and responsible for generating the graph limits.
- **Artist:**
  - An artist is the all which we see on the graph like Text objects, Line2D objects, and collection objects.
  - Most Artists are tied to Axes.

Prepared by : Ritesh Prajapati

Line Plot

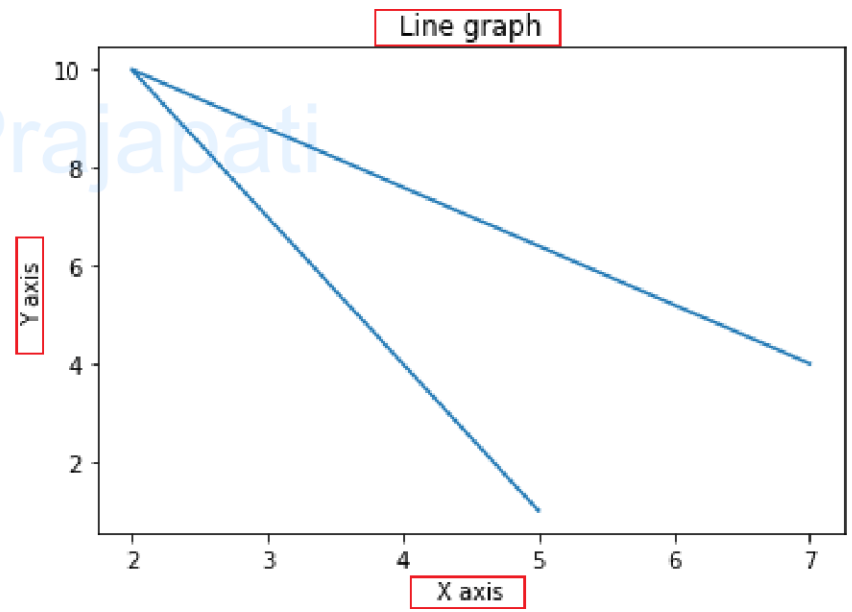
- plot() is used to draw points (markers) in a diagram.

```
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()
```



- Custom Plot

```
import matplotlib.pyplot as plt
x = [5, 2, 7]
y = [1, 10, 4]
plt.plot(x, y)
plt.title('Line graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.show()
```

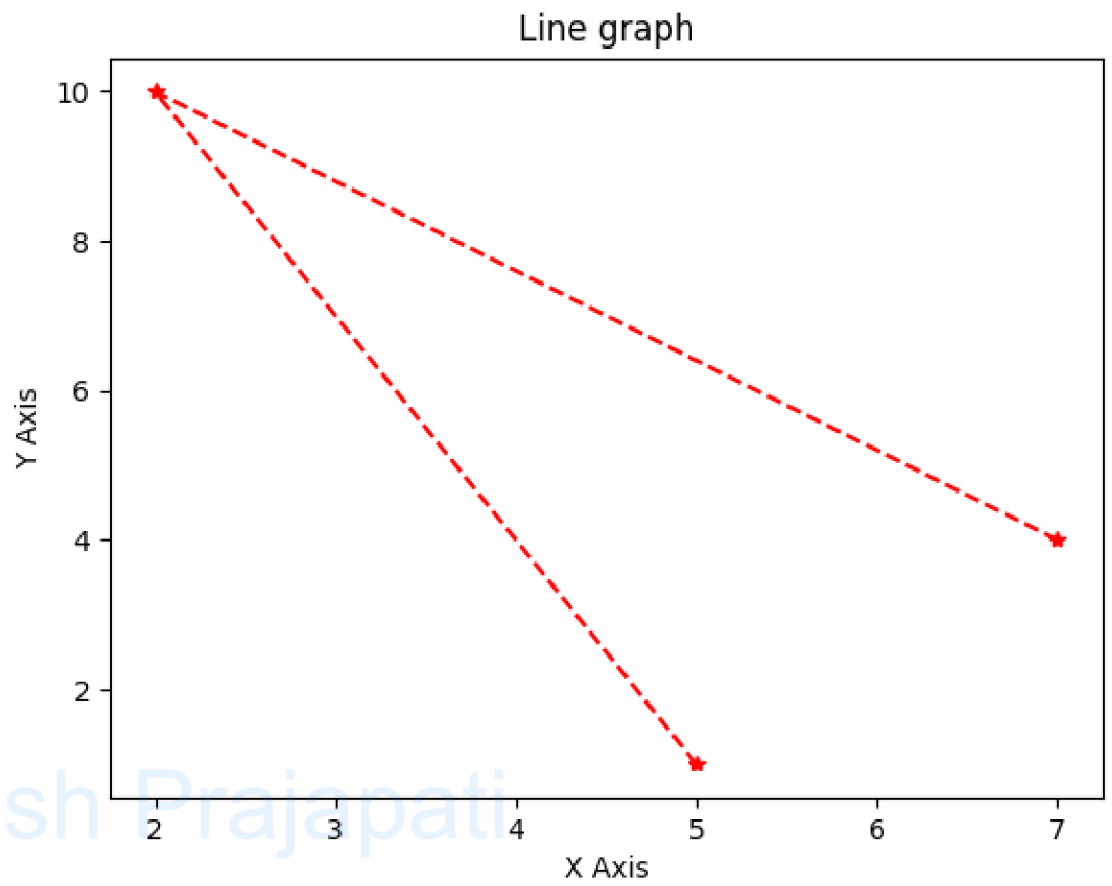


Prepared by : Ritesh Prajapati

Line Plot

- Color, marker, linestyle

```
import matplotlib.pyplot as plt
x = [5, 2, 7]
y = [1, 10, 4]
plt.plot(x, y ,color='red',marker='*',linestyle='--')
plt.title('Line graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.show()
```

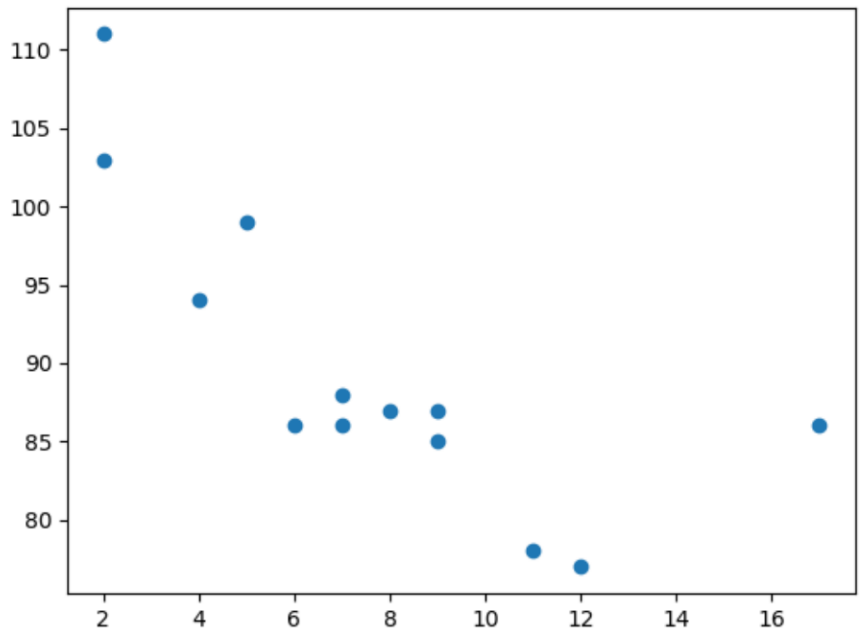


Prepared by : Ritesh Prajapati

Scatter Plots

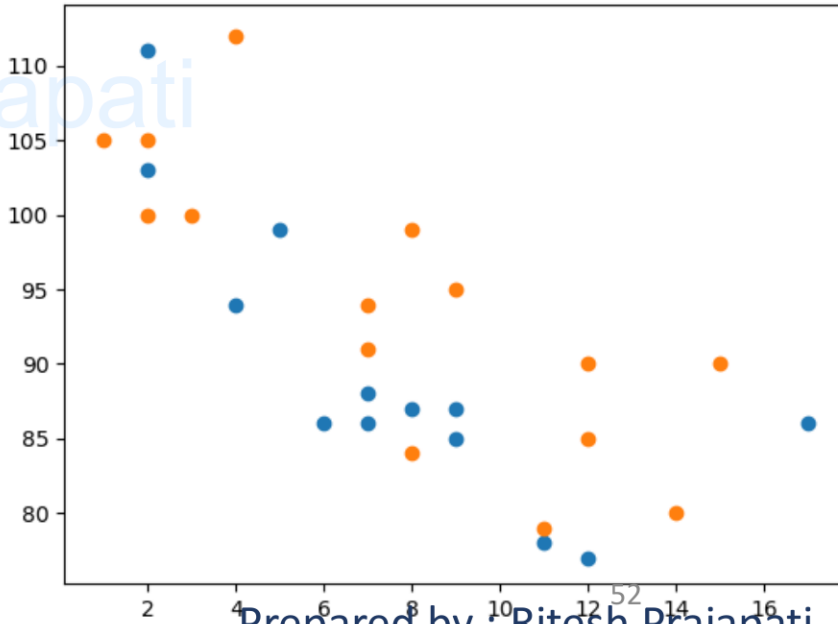
- Single Scatter Plot

```
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()
```



- Two Scatter Plot

```
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)
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)
plt.show()
```



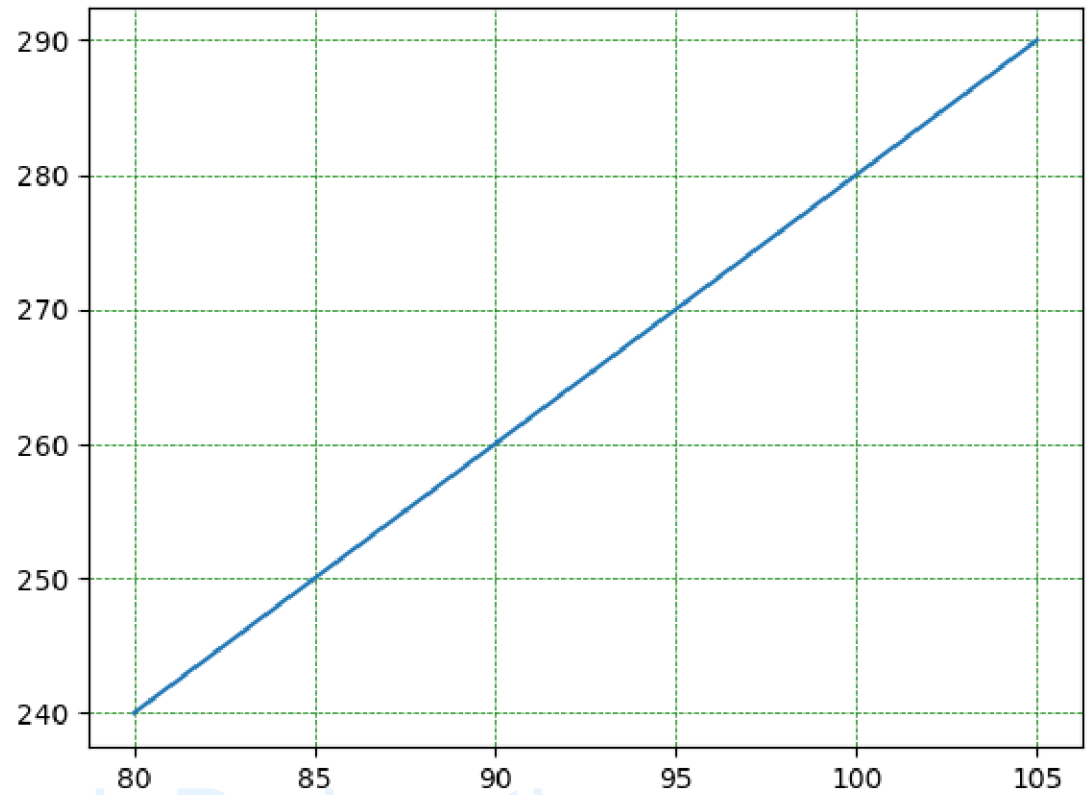
Prepared by : Ritesh Prajapati

Prepared by : Ritesh Prajapati

Grid Lines

- Color, linestyle, linewidth, Axis

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

x = np.array([80, 85, 90, 95, 100, 105])
y = np.array([240, 250, 260, 270, 280, 290])
plt.plot(x, y)
plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)
plt.show()
```

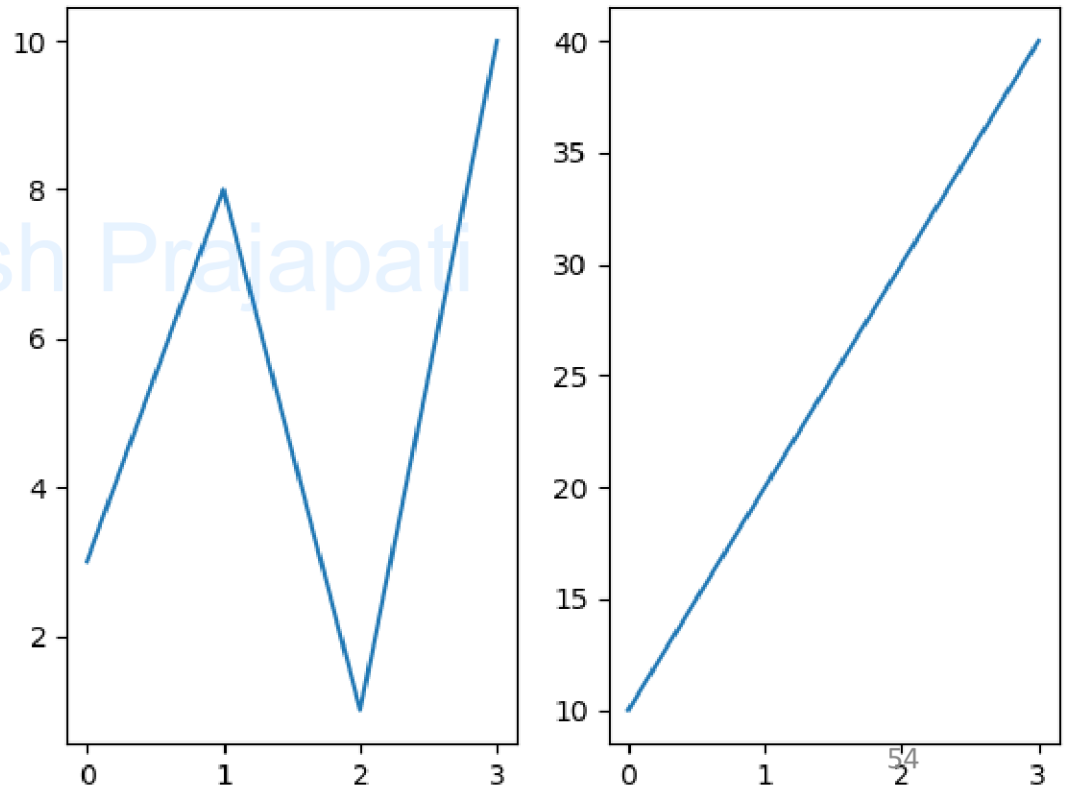


Prepared by : Ritesh Prajapati

Matplotlib Subplot

- subplot() is used to draw multiple plots in one figure:
- subplot() function takes three arguments
- first – rows, second – columns, third - index of current plot

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.show()
```

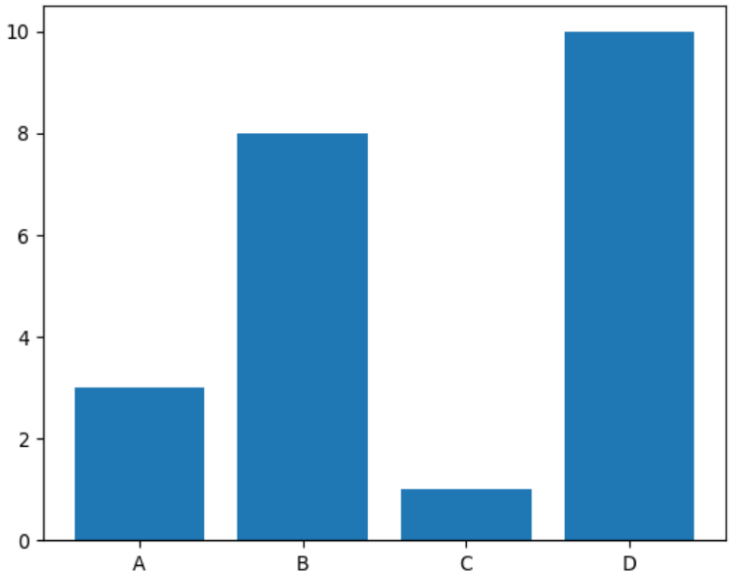


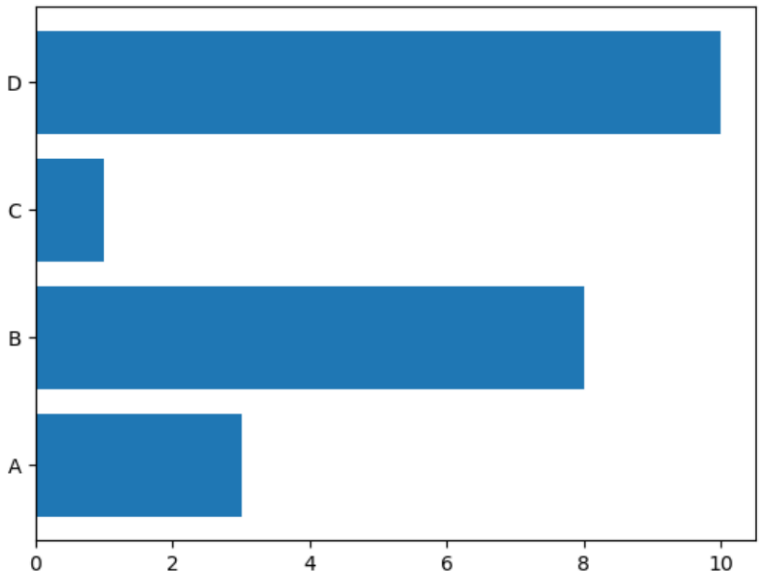
Prepared by : Ritesh Prajapati

Matplotlib Bars

- Vertical Bars

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





- Horizontal Bars

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

Prepared by : Ritesh Prajapati

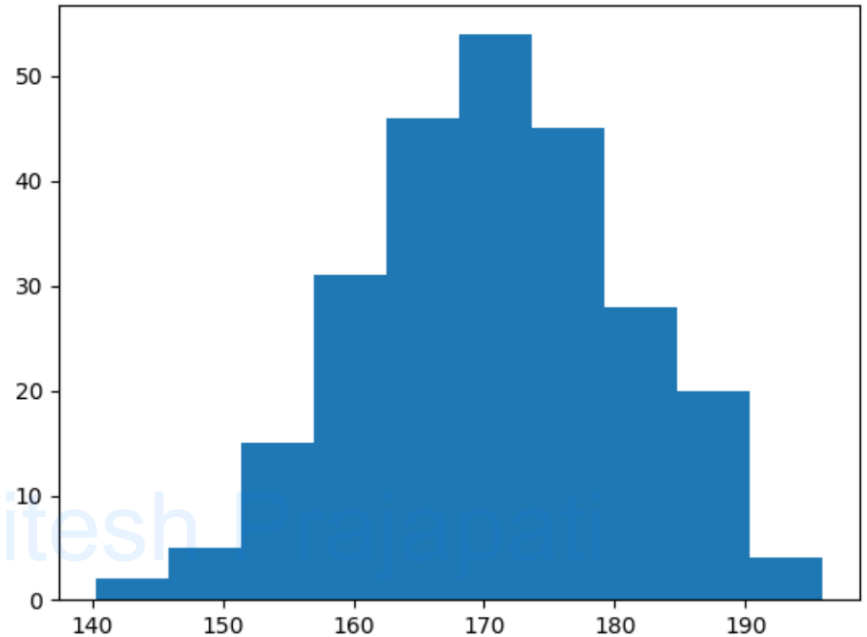
- Other Options

```
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x, y, color = "#4CAF50",width = 0.5)
plt.show()
```

Matplotlib Histogram

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

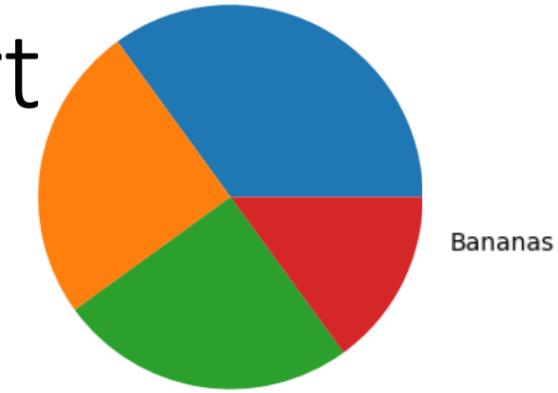
```
import matplotlib.pyplot as plt

import numpy as np

x = np.random.normal(170, 10, 250)

plt.hist(x)

plt.show()
```



Prepared by : Ritesh Prajapati

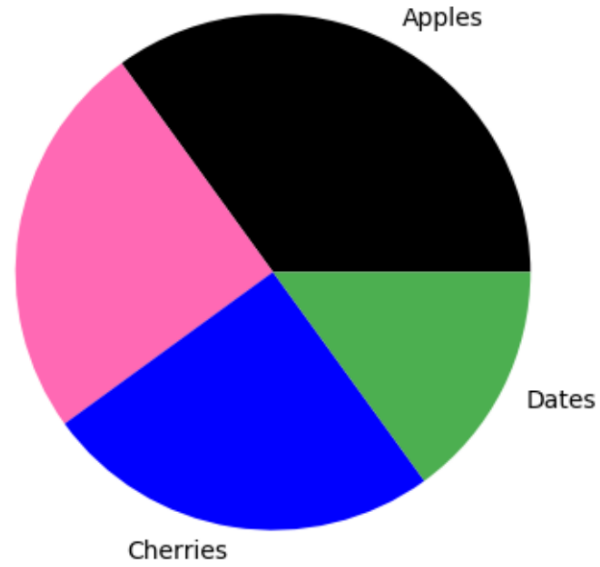
- np.random.normal(loc,scale,size) - generate a random result,
  - loc - (Mean) where the peak of the bell exists.
  - scale - (Standard Deviation) how flat the graph distribution should be.
  - size - The shape of the returned array.

Matplotlib PieChart



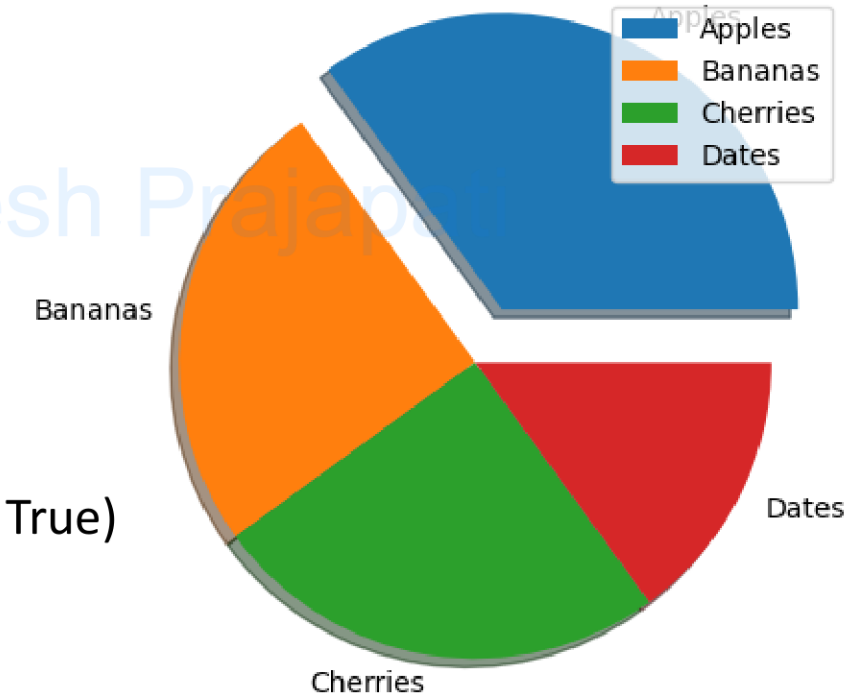

```
y = np.array([35, 25, 25, 15])
plt.pie(y)
plt.show()
```

- With Labels, Colors




```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]
plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```

- With Explode, Shadows,Legend




```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]
plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
plt.legend()
plt.show()
```

Prepared by : Ritesh Prajapati

Scikit-learn(sklearn)

- **Scikit-learn** is a Python module for machine learning.
- The project was started in 2007 by David Cournapeau as a Google Summer of Code project
- It has wide range of tools for all machine learning activities
- Website
  - https://scikit-learn.org/stable/
- Installation
  - Need to install NumPy and SciPy first
  - pip install scikit-learn

Prepared by : Ritesh Prajapati

Scikit-learn(sklearn) - Features

- Large number of datasets that can be used directly.
  - Boston house prices dataset
  - Iris plants dataset
  - Diabetes dataset
  - Optical recognition of handwritten digits' dataset
  - Linnerrud dataset
  - Wine recognition dataset
  - Breast cancer diagnostic dataset
  - The Olivetti faces dataset
  - The 20 newsgroups text dataset

Prepared by : Ritesh Prajapati

- Open Source
  - The package is open source, accessible to everyone and reusable in several contexts.

Scikit-learn(sklearn) - Features

- Support for all types of Algorithms
  - Supervised and Unsupervised models available
- Preprocessing Tools
  - Contains tools for data preprocessing (handling missing values, duplicate values)
- Feature Extraction
  - Contains tools for feature extraction from dataset
- Model Evaluation Tools
  - Various functions like cross validation, hyperparameter tuning, confusion matrix, mean square error etc.
  - Model evaluation metrics like accuracy, precision, recall, F1score.
- Integration with other libraries
  - Integrates with NumPy, Pandas etc

Prepared by : Ritesh Prajapati

Scikit-learn(sklearn) - Steps

1. Import necessary Libraries
2. Load the Dataset
3. Separate features and target variable
4. Split dataset into training and testing
5. Define model
6. Train model
7. Evaluate model
8. Refinement

Prepared by : Ritesh Prajapati

Scikit-learn(sklearn) – KNN Implementation

Source : https://www.geeksforgeeks.org/k-nearest-neighbor-algorithm-in-python

Import necessary Libraries

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
```

Load the Dataset

```
irisData = load_iris()
```

Separate features and target variable

```
X = irisData.data
y = irisData.target
```

Split dataset into training and testing

```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.2,
random_state=42)
```

Prepared by : Ritesh Prajapati

Define model

```
knn = KNeighborsClassifier(n_neighbors=7)
```

Train model

```
knn.fit(X_train, y_train)
```

Evaluate model

```
print(knn.predict(X_test))
```

Refinement

```
print(knn.score(X_test, y_test))
```

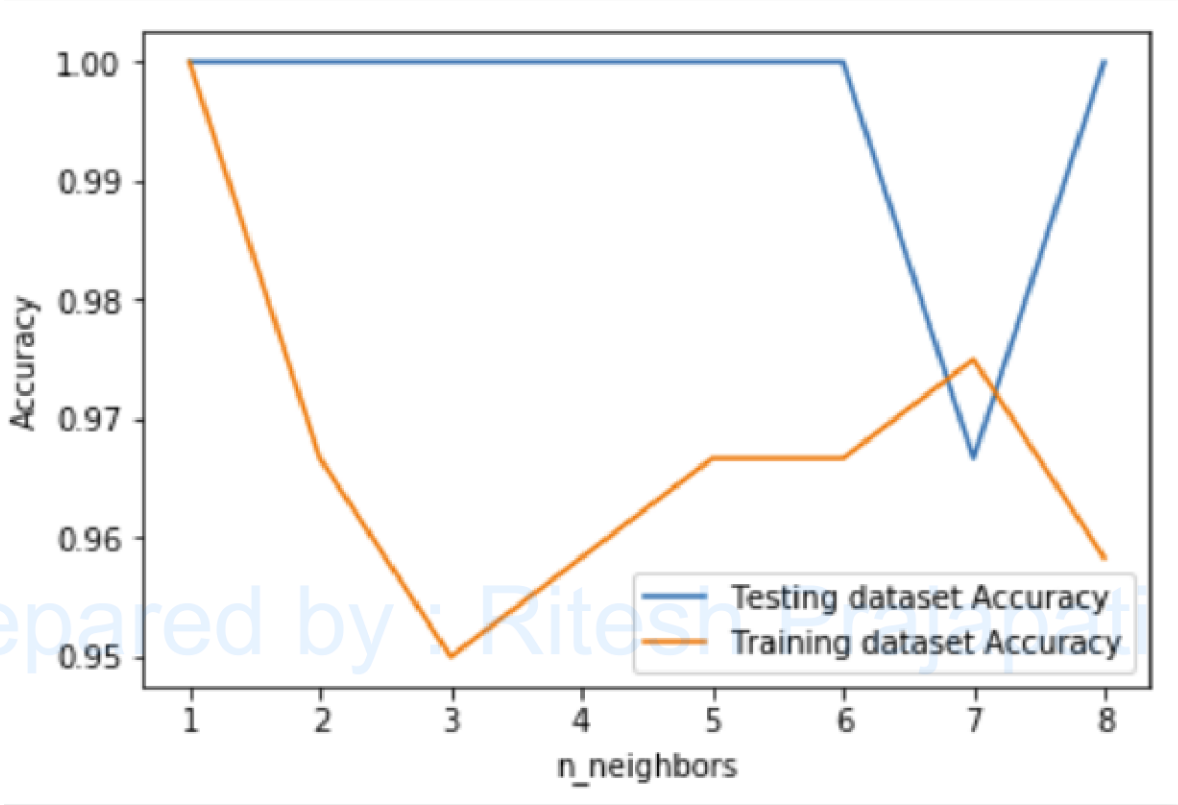
Scikit-learn(sklearn) – KNN Implementation

Source : https://www.geeksforgeeks.org/k-nearest-neighbor-algorithm-in-python

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt

irisData = load_iris()

X = irisData.data
y = irisData.target

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size = 0.2, random_state=42)

neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over K values
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Compute training and test data accuracy
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)

# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

Prepared by : Ritesh Prajapati

Scikit-learn(sklearn) – KNN Implementation



Prepared by : Ritesh Prajapati

Scikit-learn(sklearn) – Other Models Link

K-Nearest Neighbour

1. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
2. https://www.geeksforgeeks.org/k-nearest-neighbor-algorithm-in-python/
3. https://www.analyticsvidhya.com/blog/2021/01/a-quick-introduction-to-k-nearest-neighbor-knn-classification-using-python/

Decision Trees,

1. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
2. https://www.geeksforgeeks.org/decision-tree/
3. https://www.analyticsvidhya.com/blog/2022/01/decision-tree-machine-learning-algorithm/

Prepared by : Ritesh Prajapati

Support Vector Machines,

1. https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
2. https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-python/
3. https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/

Scikit-learn(sklearn) – Other Models Links

Linear Regression

1. https://www.javatpoint.com/implementation-of-linear-regression-using-python
2. https://www.geeksforgeeks.org/linear-regression-python-implementation/
3. https://www.analyticsvidhya.com/blog/2022/02/linear-regression-with-python-implementation/

Logistic Regression

1. https://www.javatpoint.com/logistic-regression-in-machine-learning
2. https://www.javatpoint.com/sklearn-logistic-regression
3. https://www.geeksforgeeks.org/ml-logistic-regression-using-python/
4. https://www.analyticsvidhya.com/blog/2022/02/logistic-regression-using-python-and-excel/

Prepared by : Ritesh Prajapati

Scikit-learn(sklearn) – Other Models Links

K means Clustering

1. https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning
2. https://www.geeksforgeeks.org/k-means-clustering-introduction/
3. https://www.analyticsvidhya.com/blog/2021/04/k-means-clustering-simplified-in-python/

Apriori Association

1. https://www.javatpoint.com/implementing-apriori-algorithm-in-python
2. https://www.geeksforgeeks.org/implementing-apriori-algorithm-in-python/
3. https://www.analyticsvidhya.com/blog/2022/01/underrated-apriori-algorithm-based-unsupervised-machine-learning/

Prepared by : Ritesh Prajapati

Scikit-learn(sklearn) – Other Useful Links

1. https://www.digitalocean.com/community/tutorials/python-scikit-learn-tutorial
2. https://machinelearningmastery.com/start-here/
3. https://courses.analyticsvidhya.com/courses/Machine-Learning-Certification-Course-for-Beginners
4. https://scikit-learn.org/stable/tutorial/index.html

Prepared by : Ritesh Prajapati

THANK YOU

Prepared by : Ritesh Prajapati