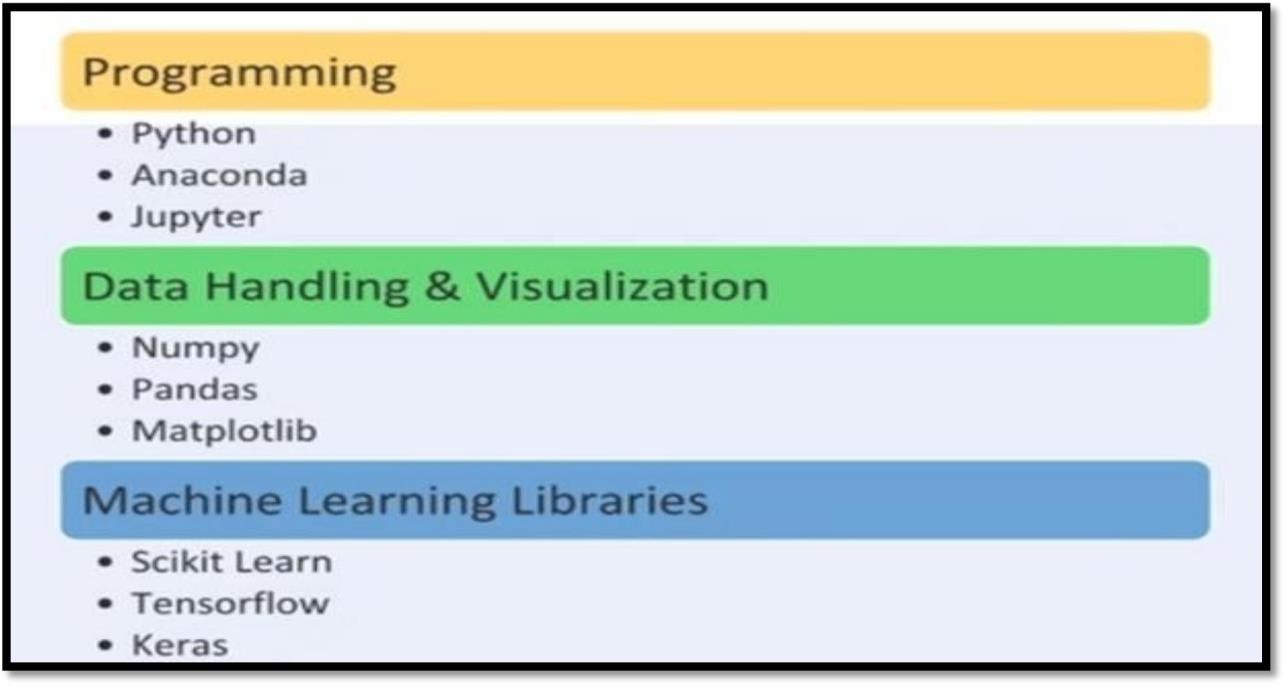
# Exercise-1

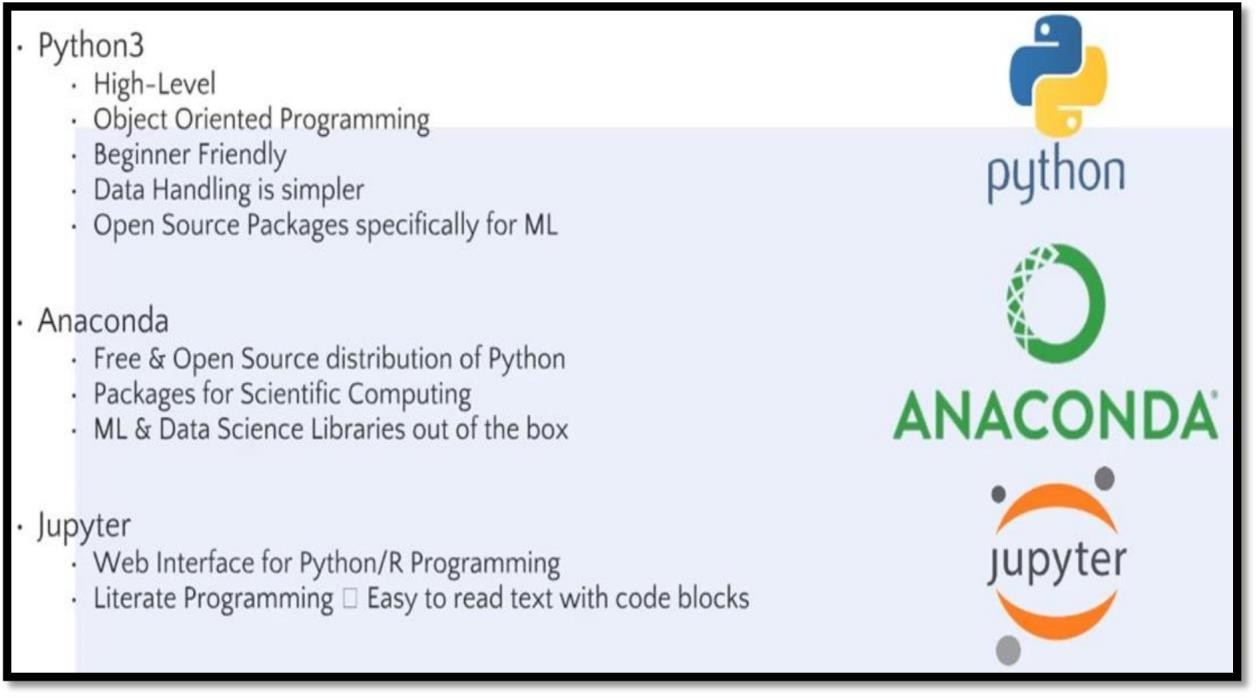
**AIM: Anaconda Software Installation and introduction to Jupyter and Co Lab.**

**ANACONDA SOFTWARE INSTALLATION**

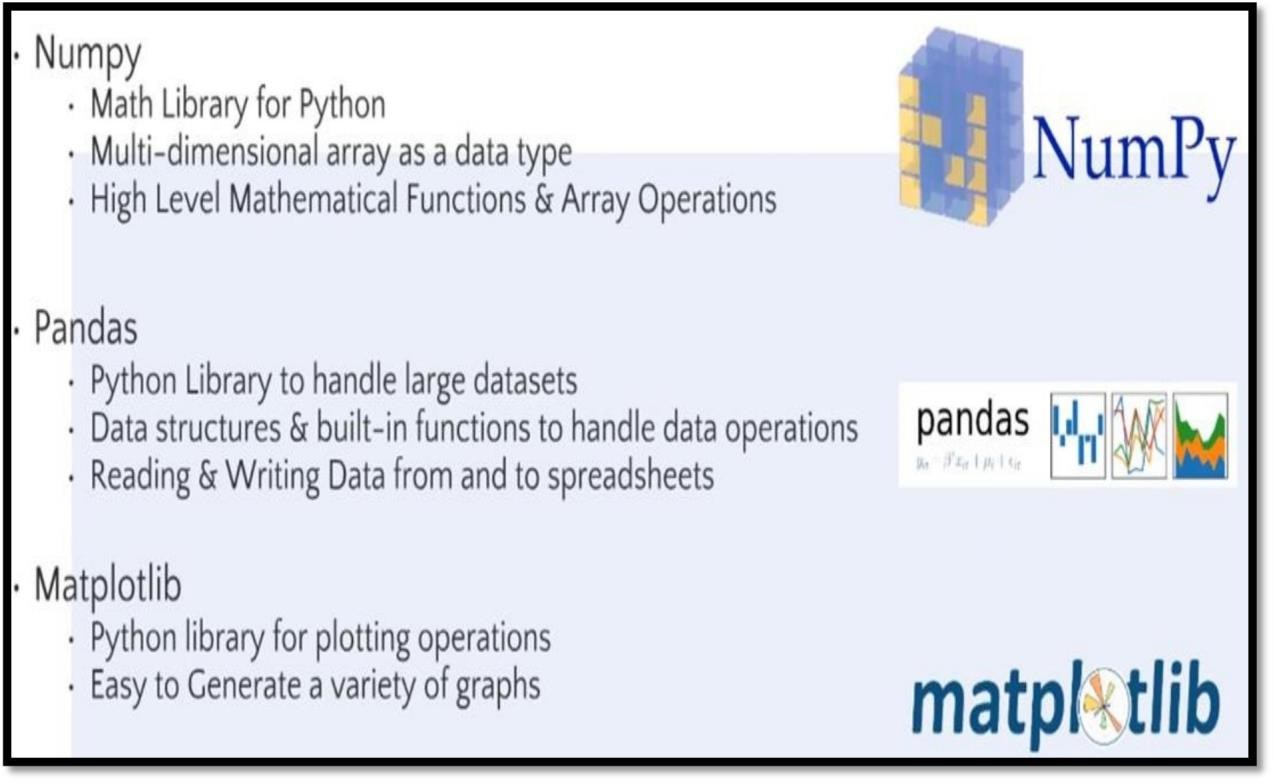
**TOOLS USED FOR MACHINE LEARNING**

****

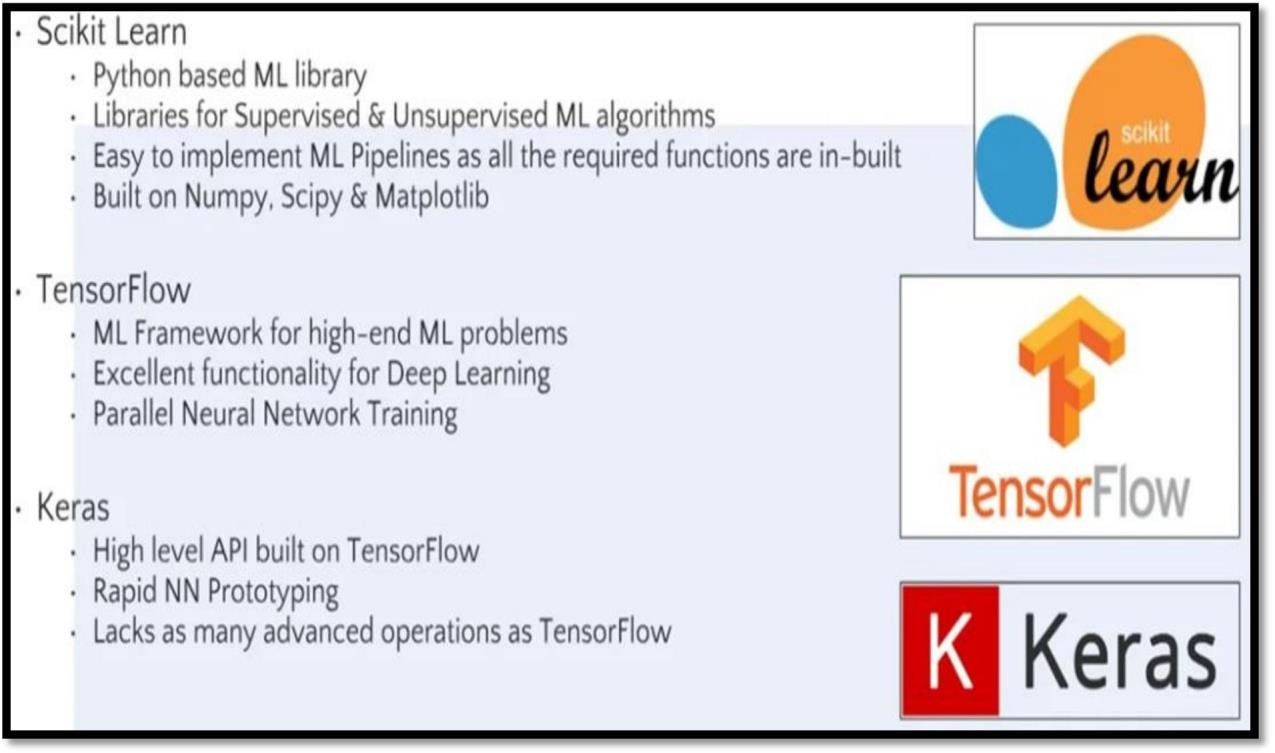
**PROGRAMMING TOOLS**

****

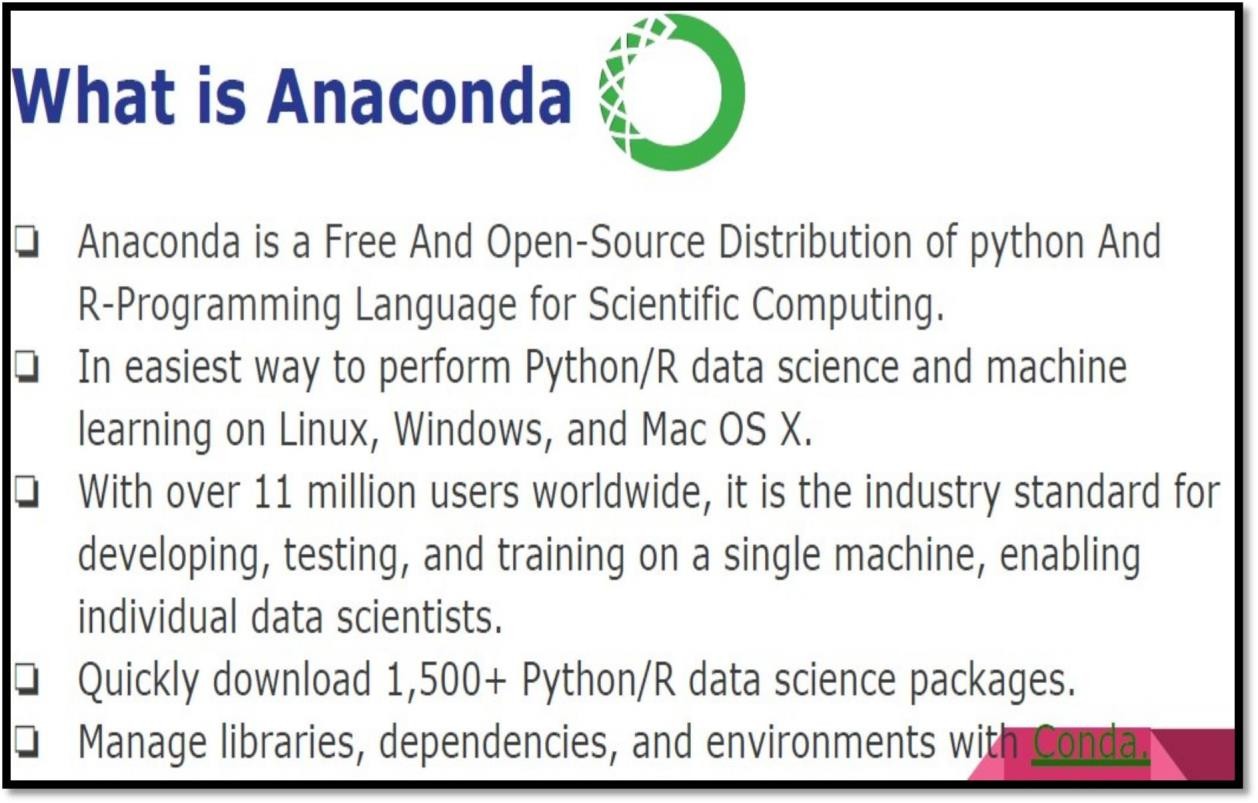
**DATA HANDLING IN PYTHON**

****

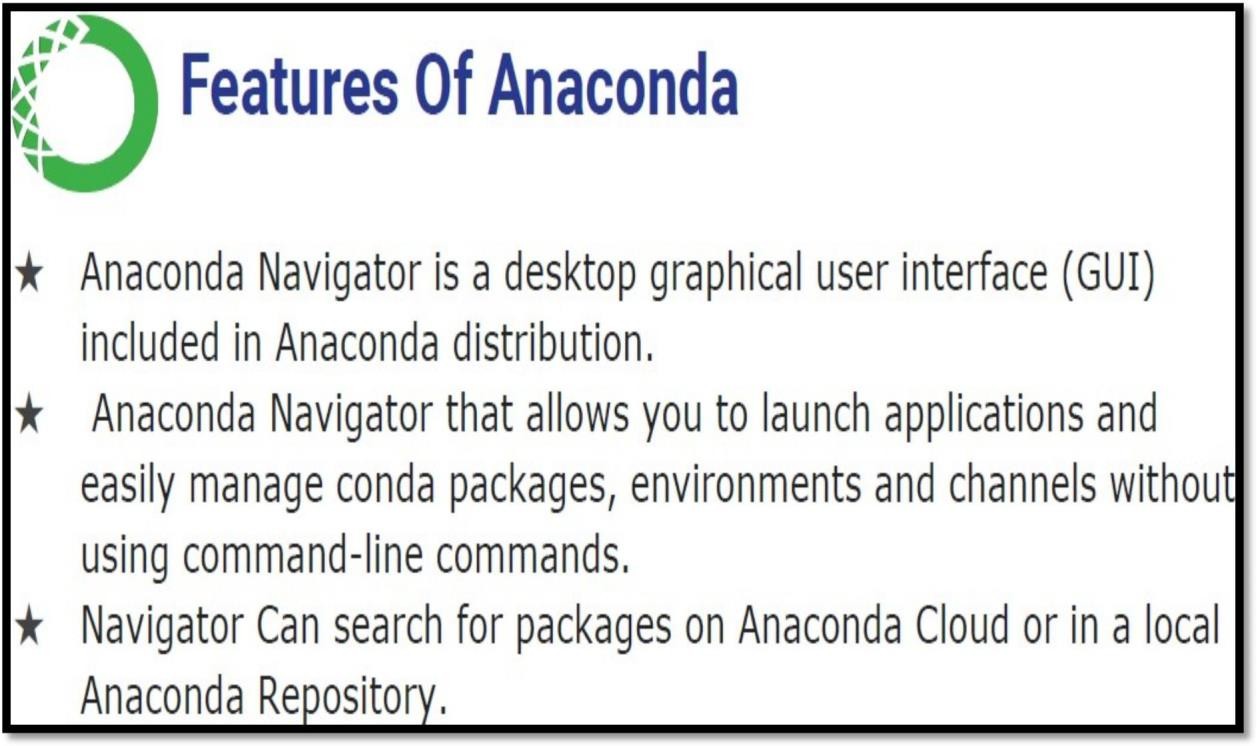
**MACHINE LEARNING LIBRARIES**

****

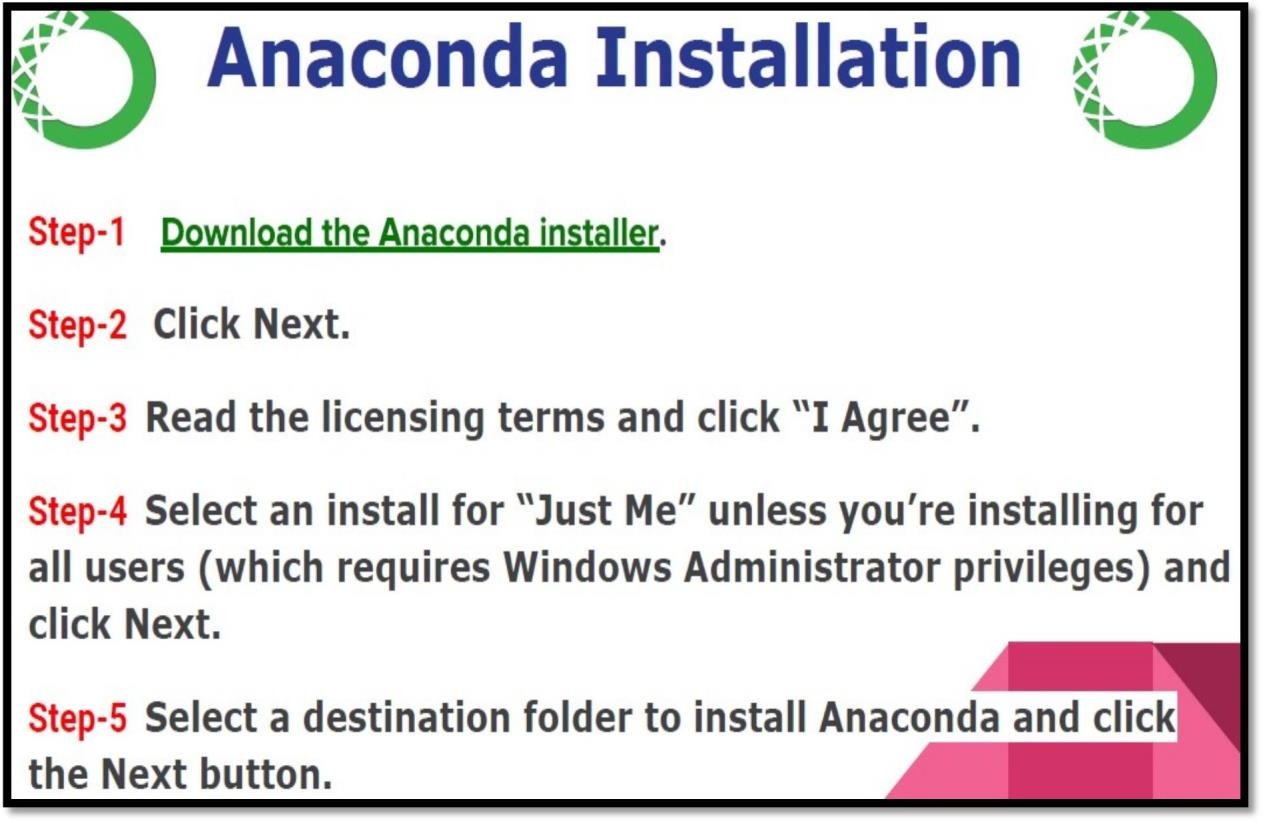
**INSTALLATION OF ANACONDA**

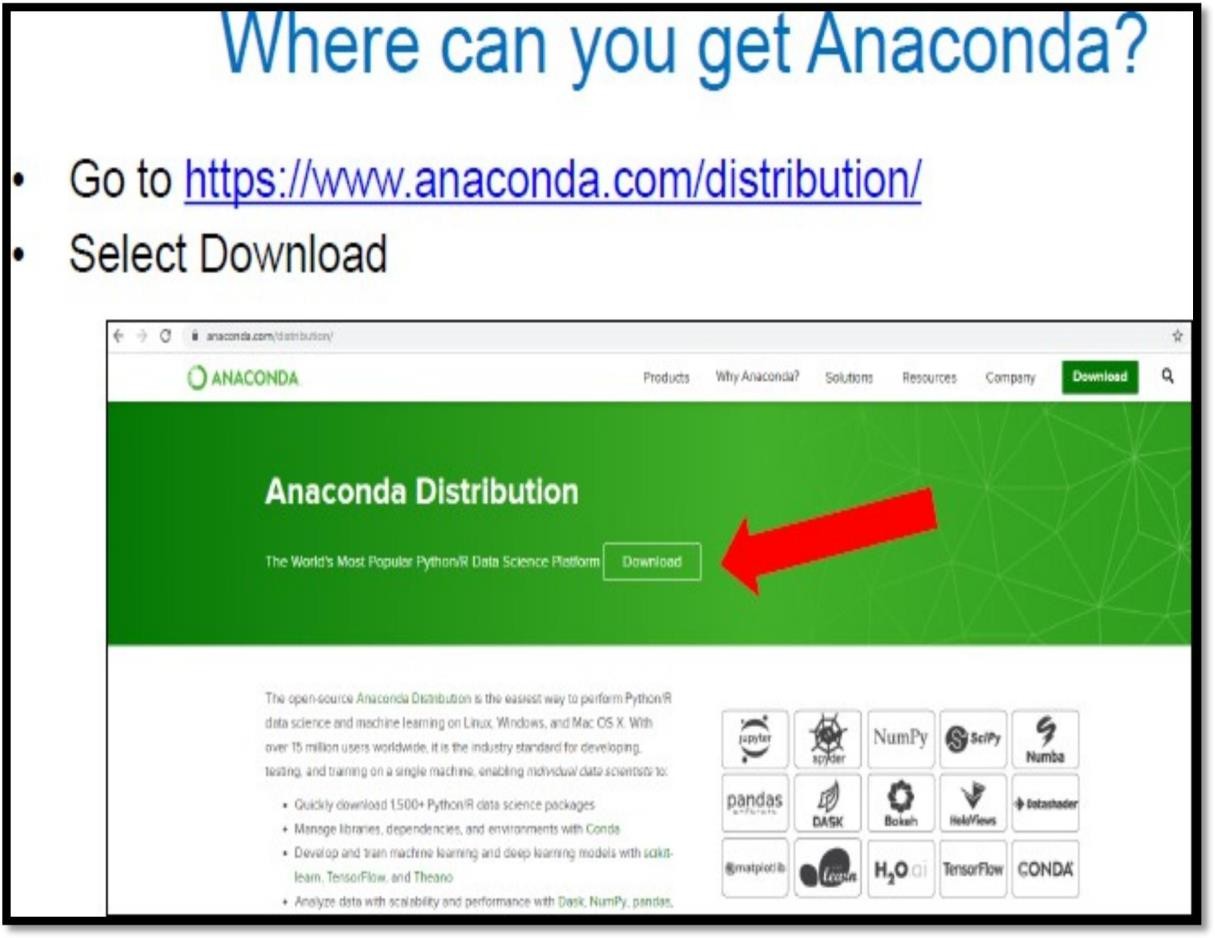
****

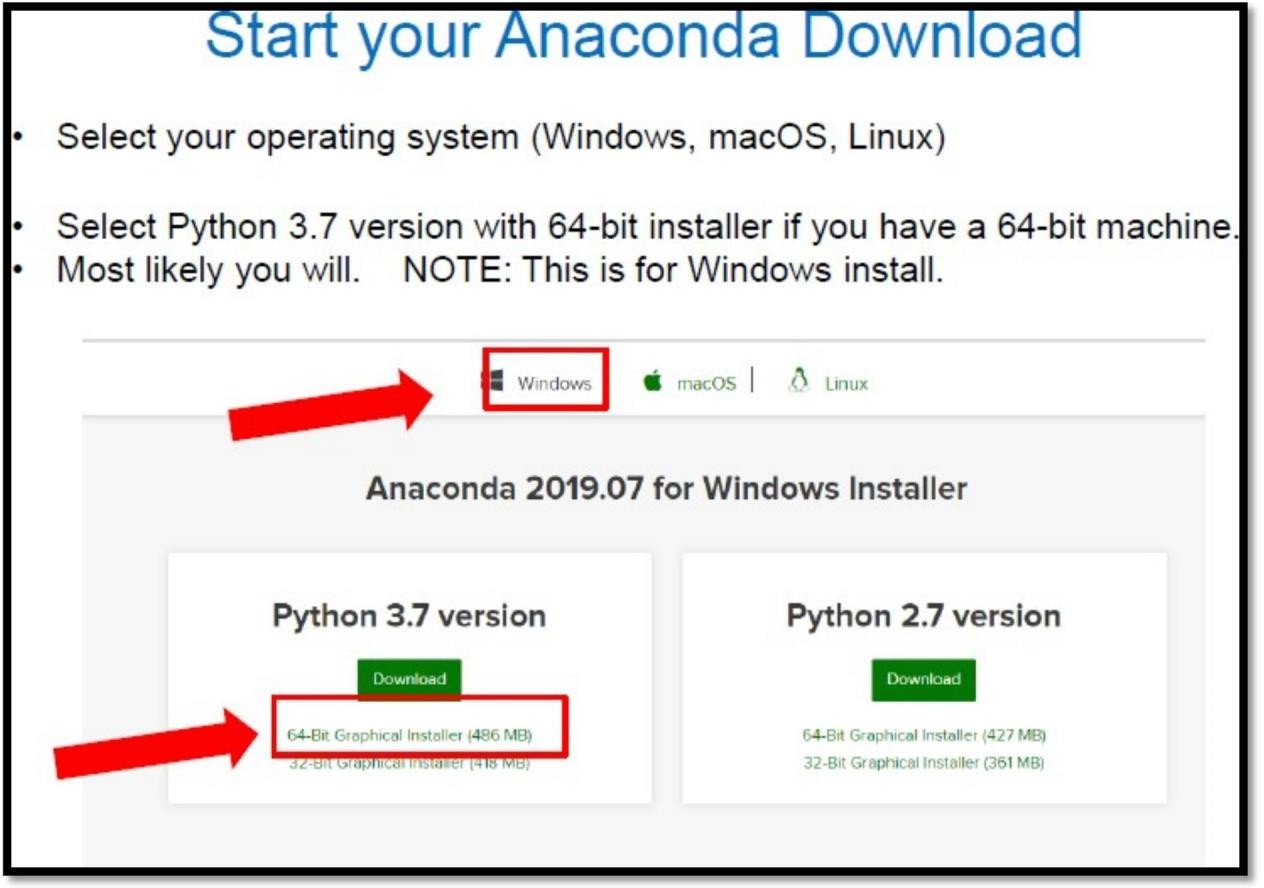
**FEATURES OF ANACONDA**

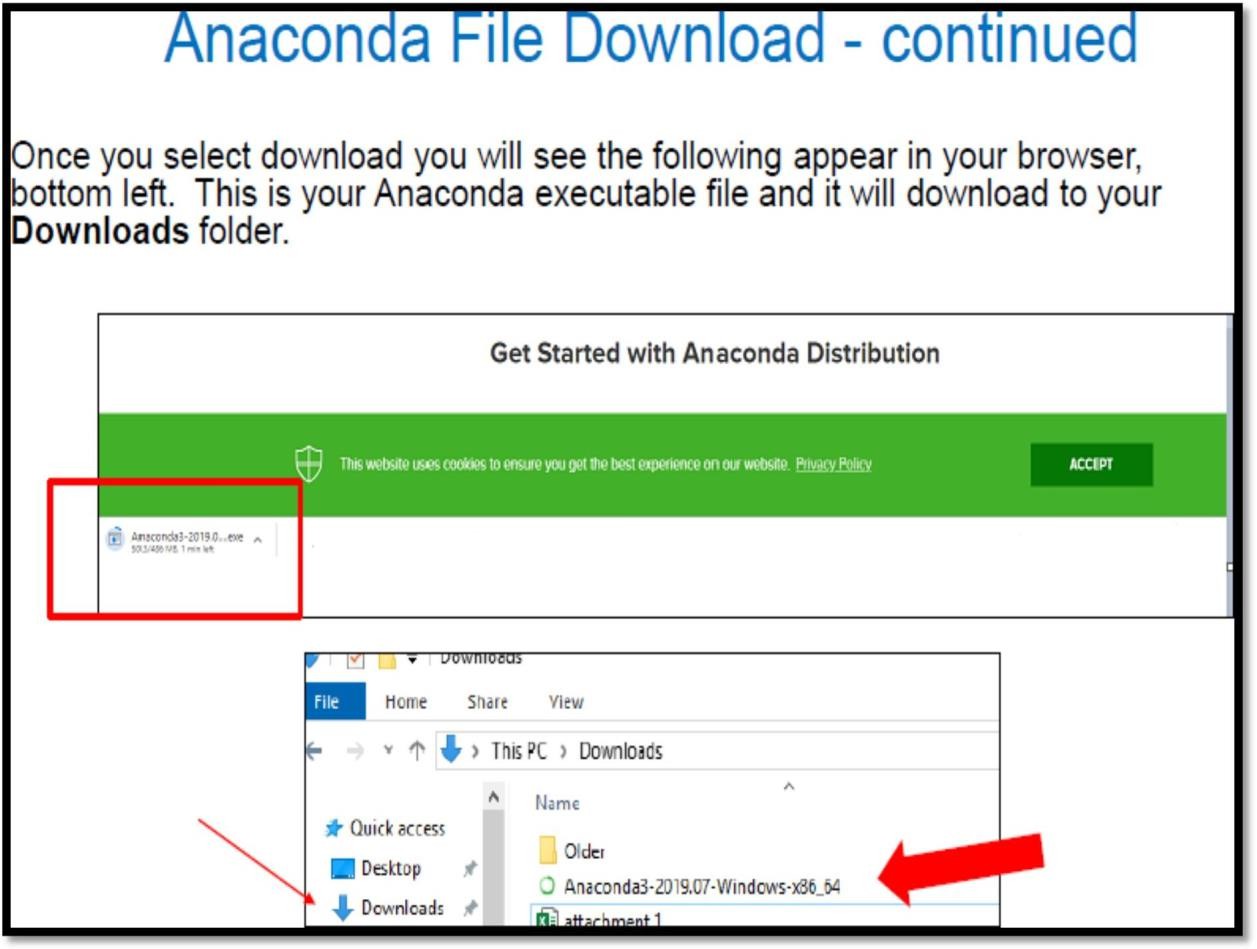
****

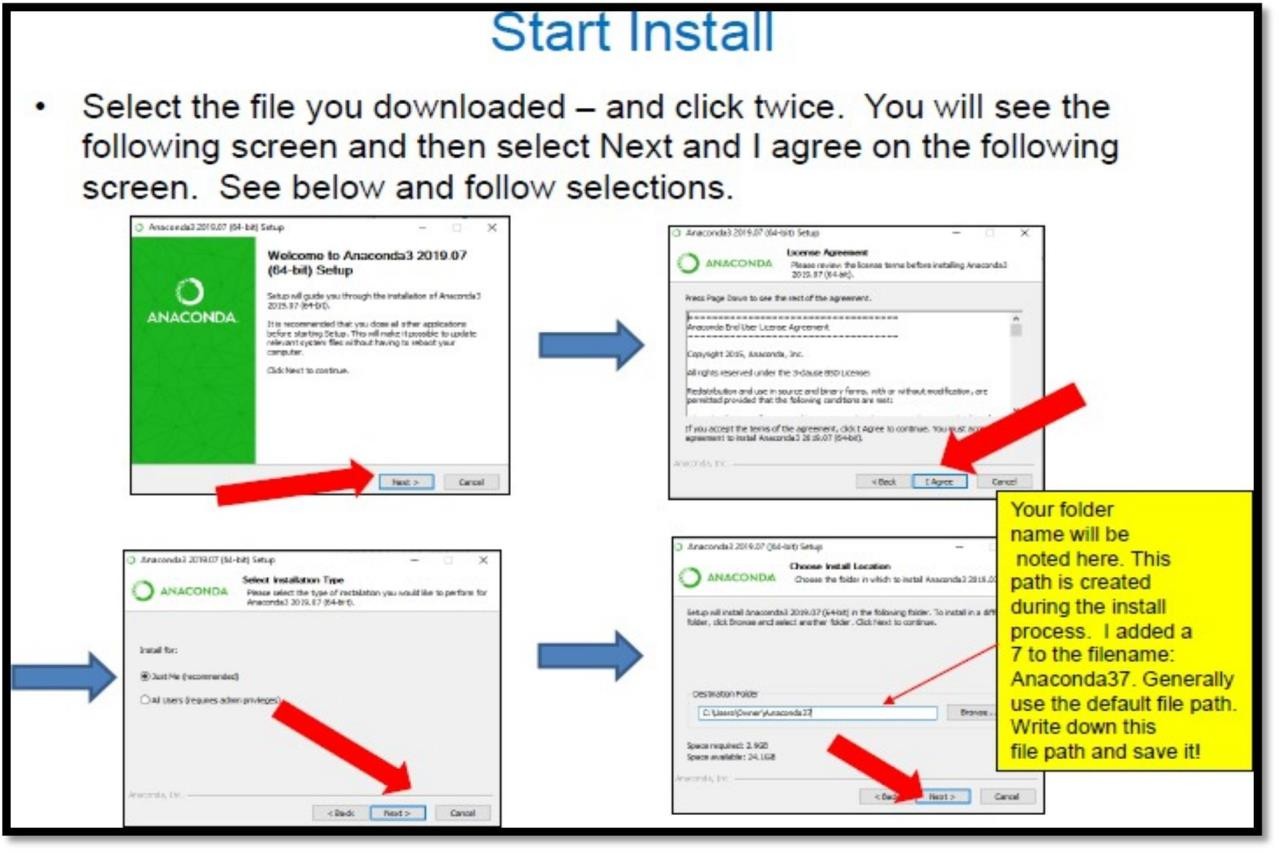


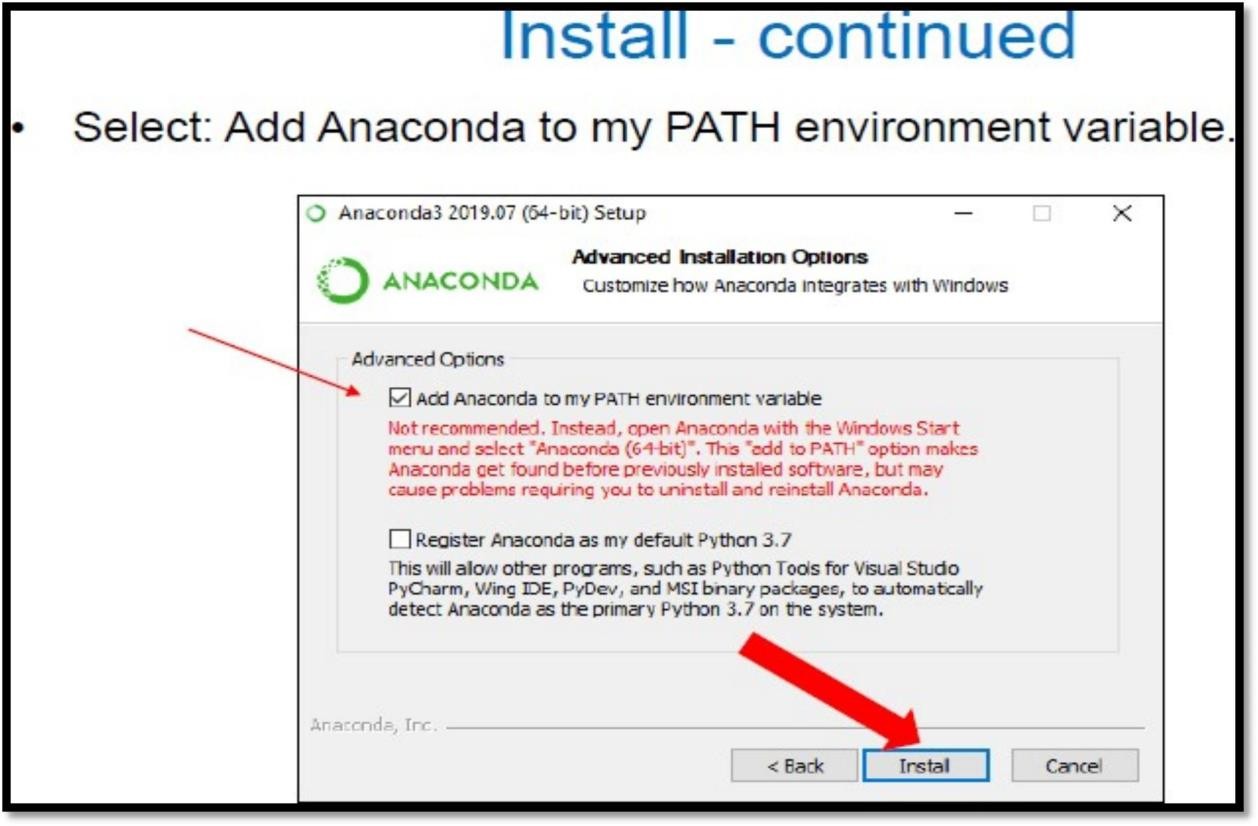
****

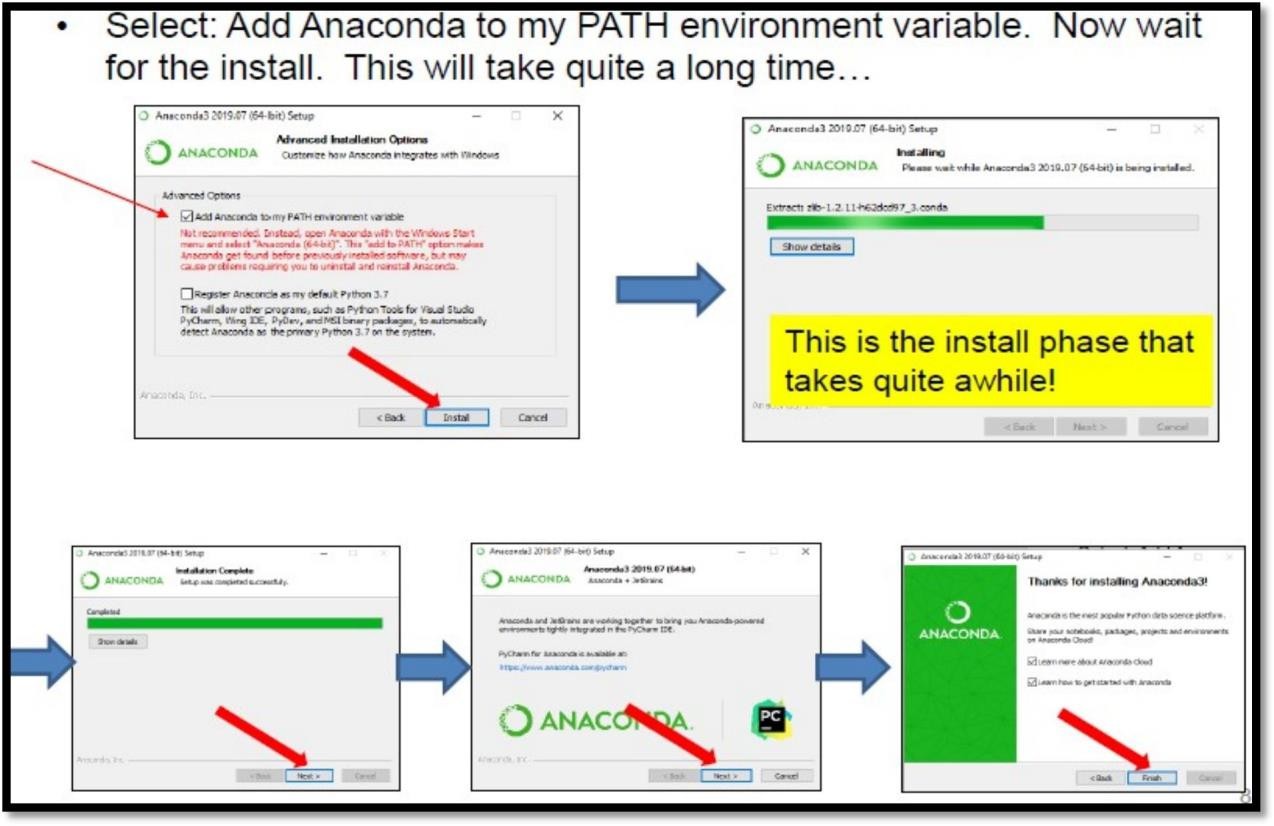


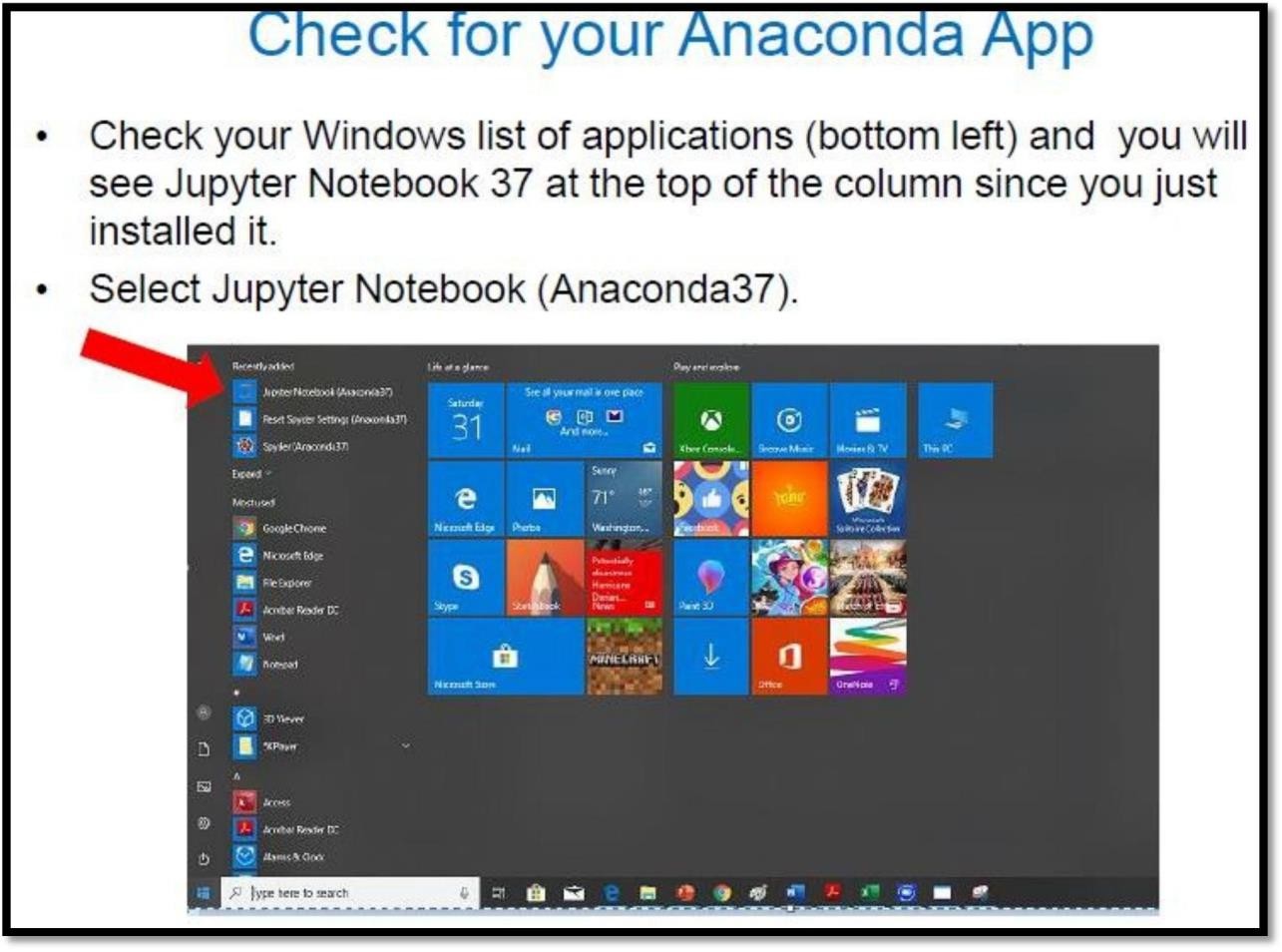
****

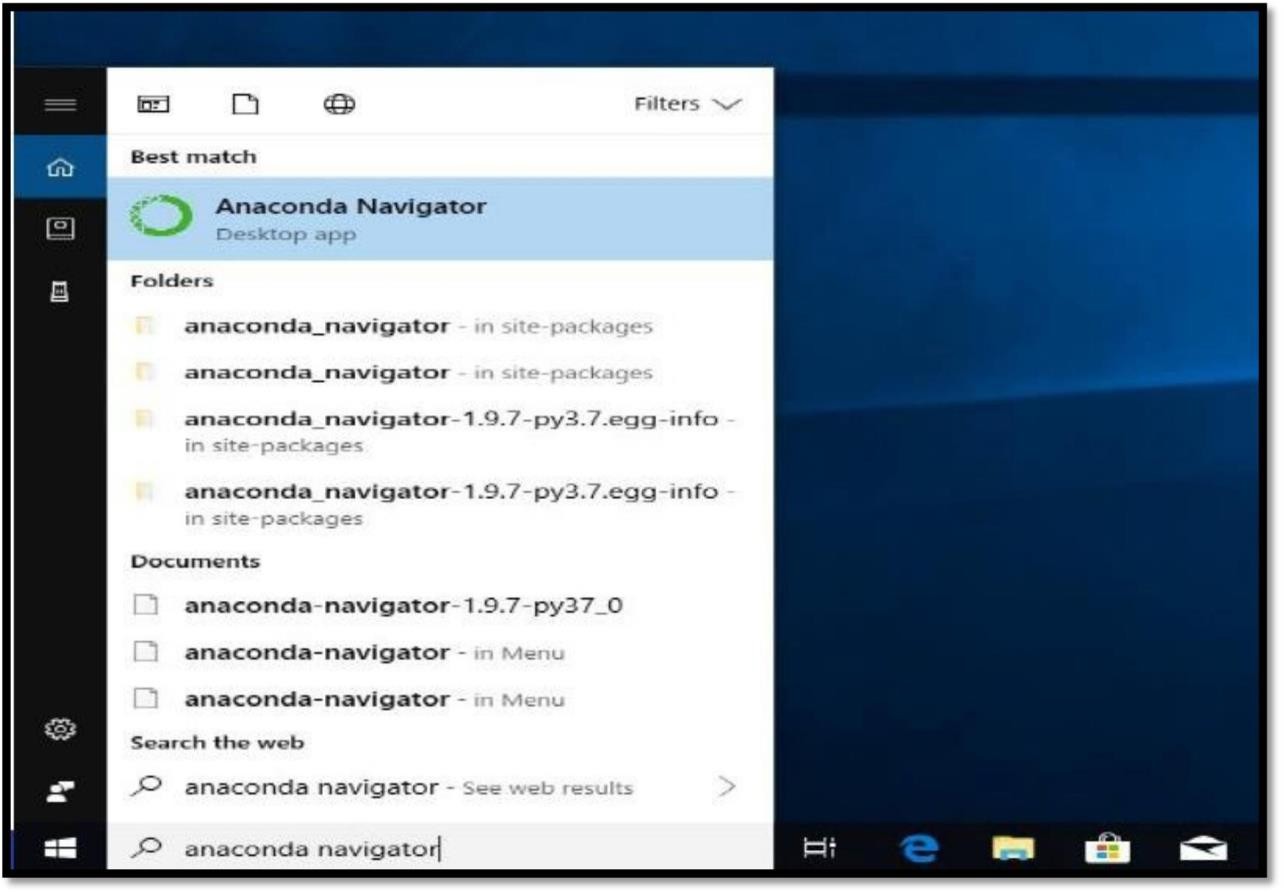


****

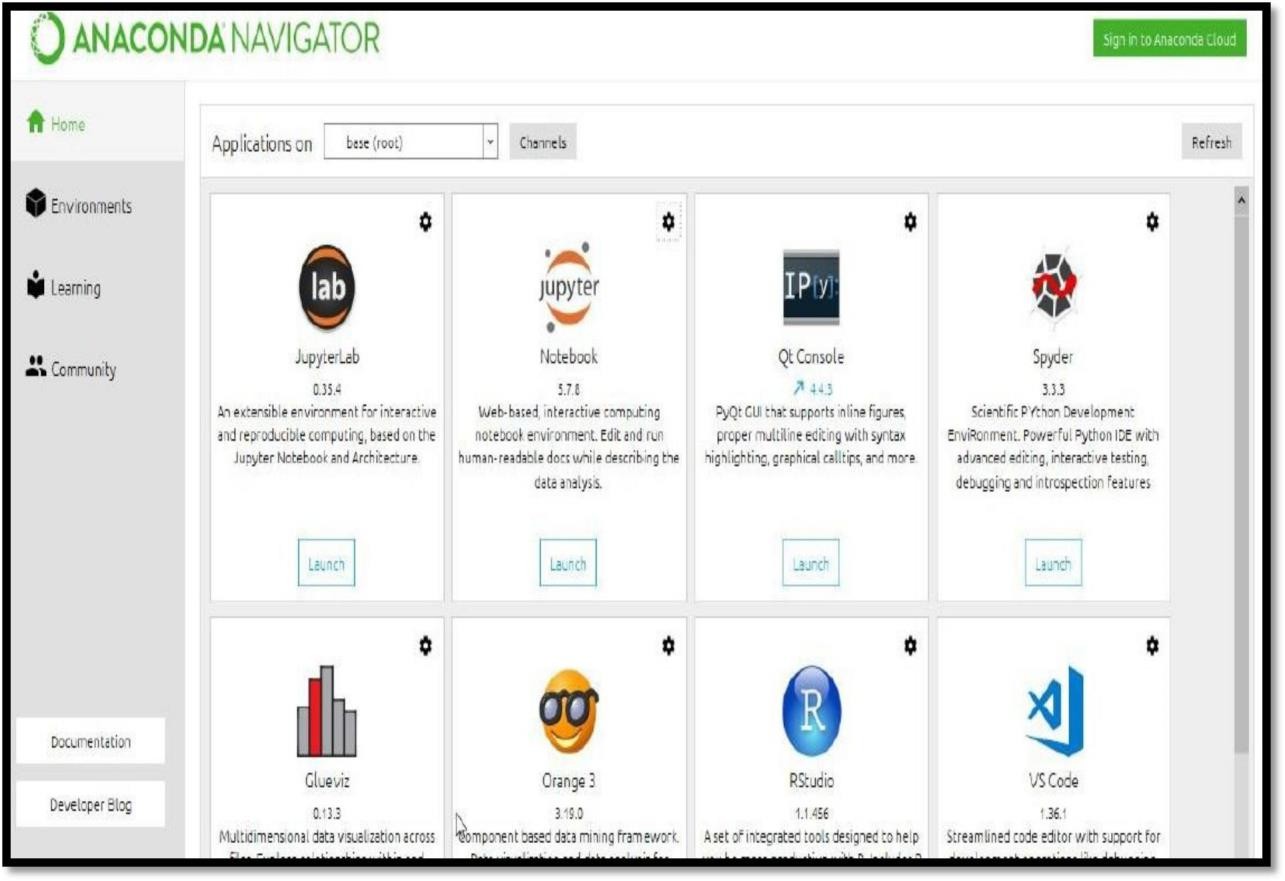


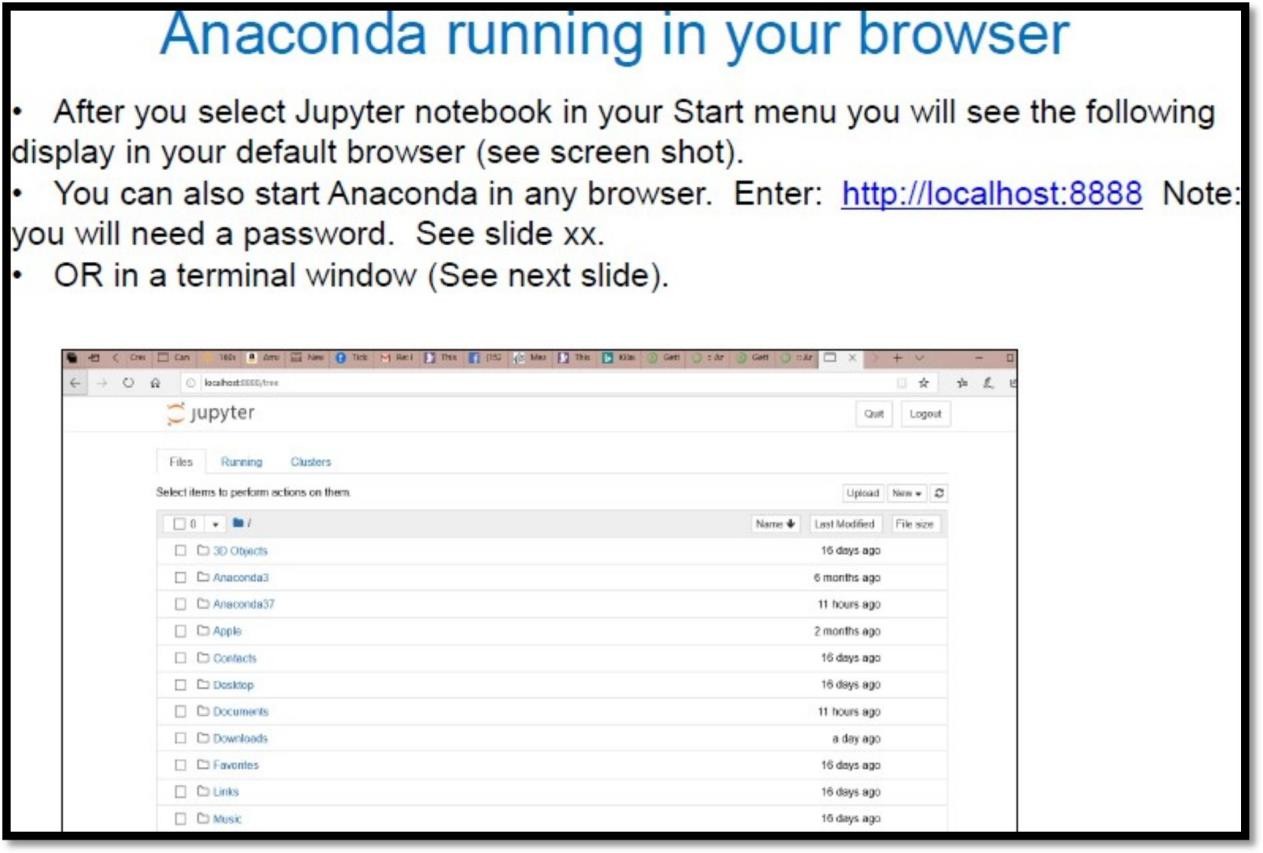
****

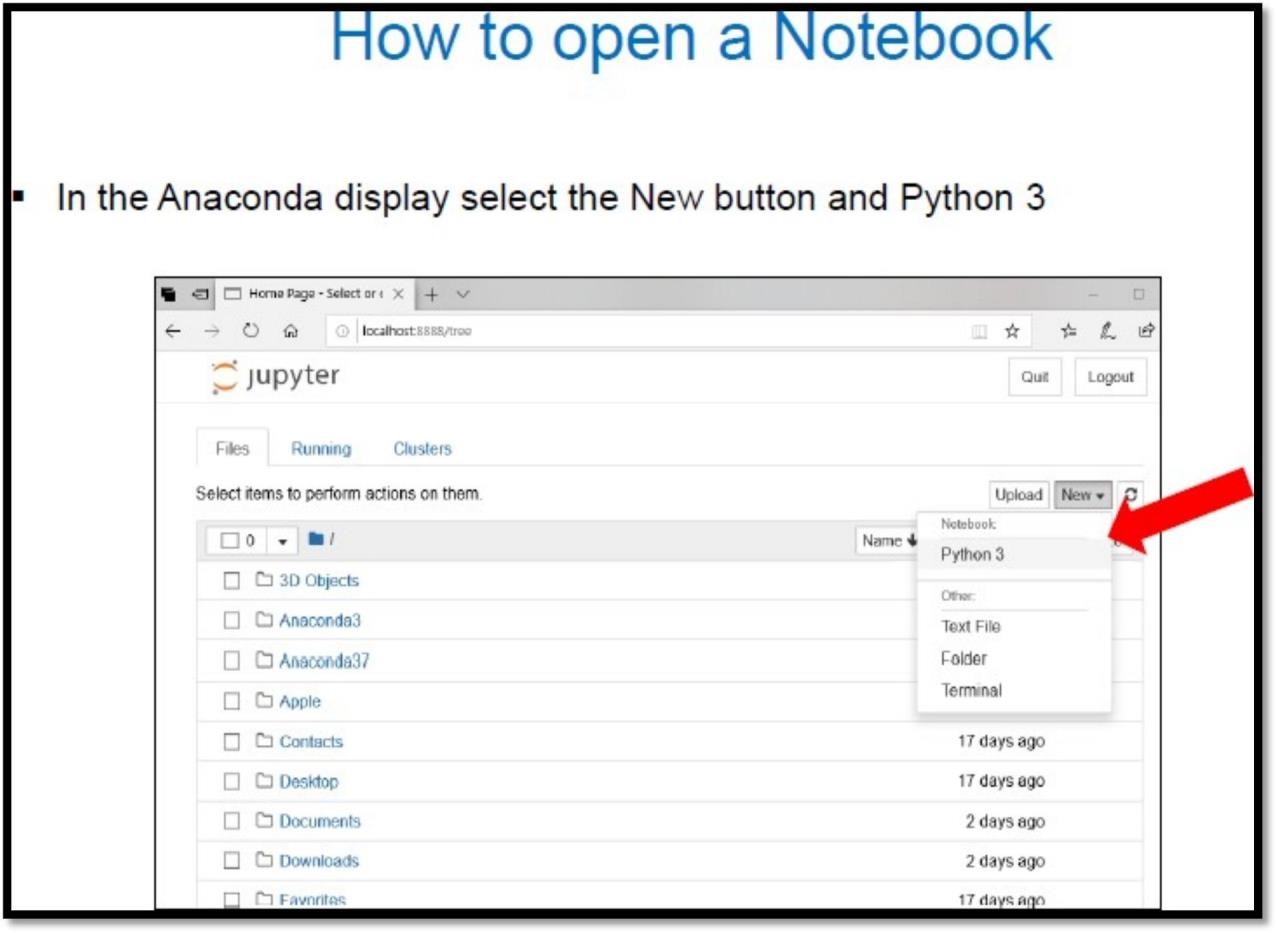


****

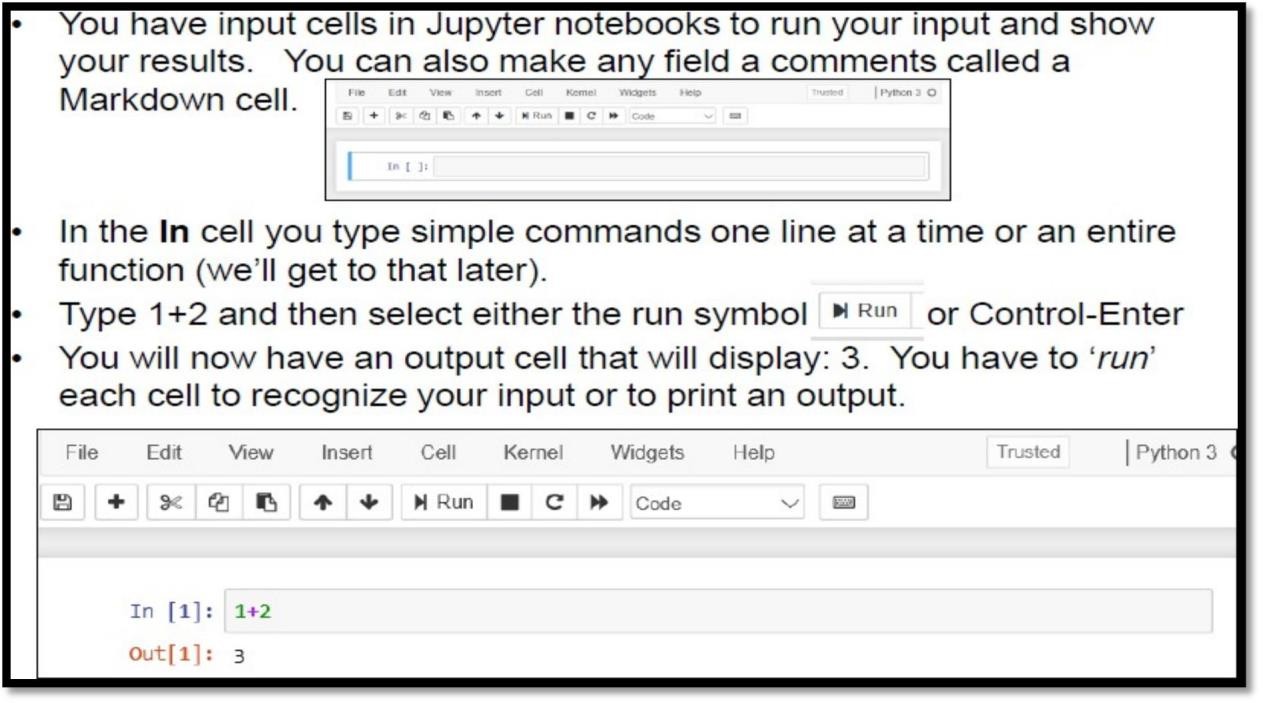
**After that select the Jupyter Notebook**

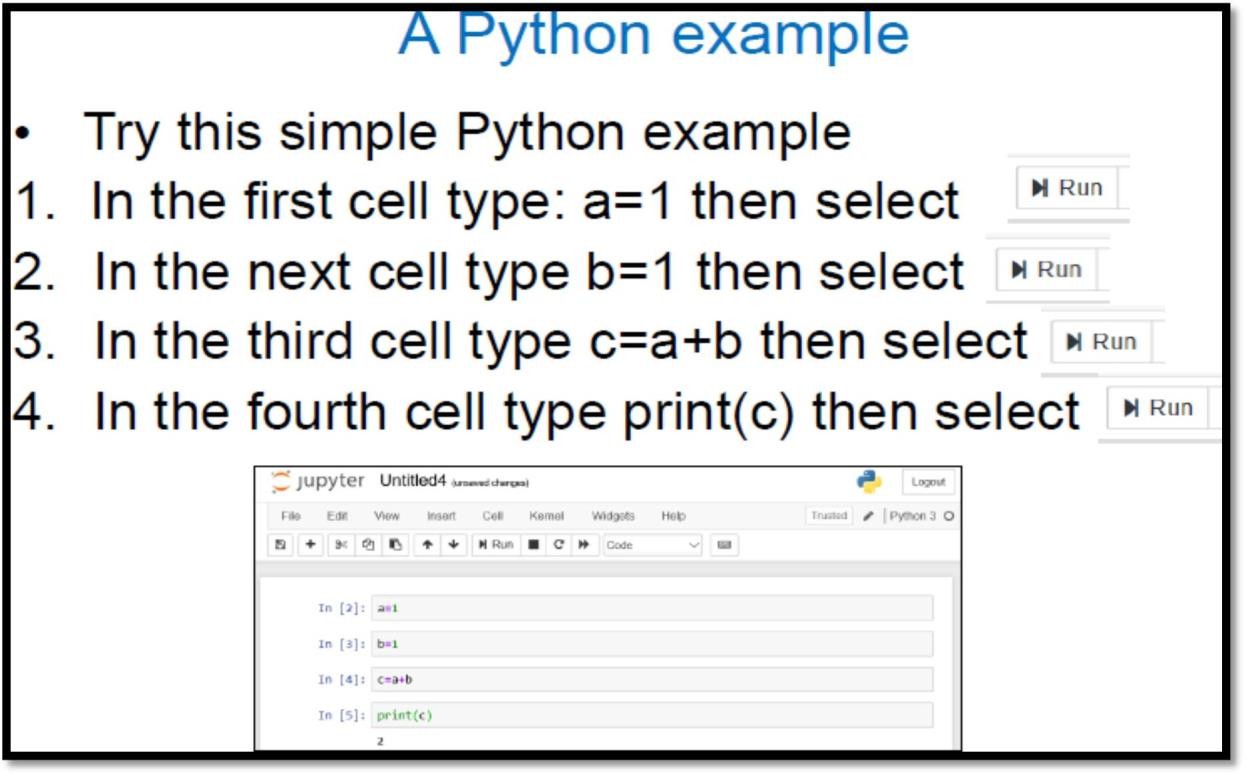
****

****



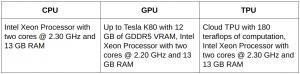
**Jupyter Notebook Example:**

****



## Google Colab:

**Google Colaboratory** is a free online cloud-based Jupyter notebook environment that allows us to train our machine learning and deep learning models on CPUs, GPUs, and TPUs.



#### Why use Google Colab:

* You can write and execute code in python
* It can be provided by the free jupyter notebook environment called colaboratory or Colab
* You can create / share / upload notebooks
* You can import / save notebooks from / to Google Drive
* You can import external datasets Eg: Kaggle
* You can easily integrate TensorFlow, Keras, OpenCV etc.
* There is a free cloud service with free GPU by using ML and Large Deep Learning Models easily

### BASIC PYTHON PROGRAMS

**Printing statements in Python:** Python print () Function The print() function prints the specified message to the screen, or other standard output device.

**Program:**

**# Printing a string** print("Hello, Everyone") print('1,2,3,4')

a = 5

print('The value of a is', a) print(1, 2, 3, 4)

print(1, 2, 3, 4, sep='\*')

print(1, 2, 3, 4, sep='#', end='&')

**Output:**

Hello, Everyone 1,2,3,4

The value of a is 5 1 2 3 4

1\*2\*3\*4

1#2#3#4&

#### Python Indentation

Indentation refers to the spaces at the beginning of a code line. Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important. Python uses indentation to indicate a block of code.

#### Example:

if 5 > 2:

print("Five is greater than two!")

#### Output:

Five is greater than two!

**Variables:** Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.

Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.

###### Rules:

* The first character of the variable must be an alphabet or underscore ( \_ ).
* All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
* Identifier name must not contain any white-space, or special character (! @, #, %, ^, &, \*).
* Identifier name must not be similar to any keyword defined in the language.
* Identifier names are case sensitive; for example, my name, and MyName is not the same.
* Examples of valid identifiers: a123, \_n, n\_9, etc.
* Examples of invalid identifiers: 1a, n%4, n 9, etc.

#### Program-1:

var1 = 10 # An integer assignment var2 = 3.146 # A floating point

var3 = "Hello" # A string print(var1,' ',var2,' ',var3)

**Output:** 10 3.146 Hello

#### Program-2:

# Assigning same value to multiple variables var1 = var2 = var3 = 1

print(var1,' ',var2,' ',var3)

# Assigning Different values to variable in a single expression var1, var2, var3 = 1, 2.5, "Hello"

print(var1,' ',var2,' ',var3)

# Note: commas can be used for multi-assignments

#### Output:

1 1 1

1 2.5 Hello

**Python User Input from Keyboard – input () function**

* Python user input from the keyboard can be read using the input () built-in function.
* The input from the user is read as a string and can be assigned to a variable.
* After entering the value from the keyboard, we have to press the “Enter” button. Then the input () function reads the value entered by the user.

**The syntax of input () function is:**

#### input(prompt)

**Program:**

num = input ("Enter number:") print(num)

num1 = float(input("Enter a floating point number:")) print(num1)

name1 = input("Enter string: ") print(name1)

**Output:**

Enter number: 12

12

Enter a floating point number: 3.5 3.5

Enter string: MACHINE LEARNING LAB MACHINE LEARNING LAB

#### Datatypes:

Data types are the classification or categorization of data items. Data types represent a kind of value which determines what operations can be performed on that data. Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

###### For eg: a=5

* The variable a holds integer value five and we did not define its type. Python interpreter will automatically interpret variables a as an integer type.
* Python enables us to check the type of the variable used in the program. Python provides us the type() function, which returns the type of the variable passed.
* Python provides various standard data types that define the storage method on each of them.

###### The data types defined in Python are given below.

* 1. [Numbers](https://www.javatpoint.com/python-data-types)
  2. [Sequence Type](https://www.javatpoint.com/python-data-types)
  3. [Boolean](https://www.javatpoint.com/python-data-types)
  4. [Set](https://www.javatpoint.com/python-data-types)
  5. [Dictionary](https://www.javatpoint.com/python-data-types)

1. **Numeric:** Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type. Python provides the **type()** function to know the data-type of the variable. Similarly, the **isinstance()** function is used to check an object belongs to a particular class.

#### Program-1:

a = 5

print("The type of a", type(a)) b = 40.5

print("The type of b", type(b)) c = 1+3j

print("The type of c", type(c))

print(" c is a complex number:", isinstance(c,complex)) ‘

#### Output:

The type of a <class 'int'> The type of b <class 'float'>

The type of c <class 'complex'> c is a complex number: True

#### Sequence Type:

* 1. **String:** The string can be defined as the sequence of characters represented in the quotation marks. In Python, we can use single, double, or triple quotes to define a string.

#### Program:

###### # String operations

str = 'Hello World!' # A string print(str) # Prints complete string

print(str[0]) # Prints first character of the string

print(str[2:5]) # Prints characters starting from 3rd to 5th element print(str[2:]) # Prints string starting from 3rd character

print(str \* 2) # Prints string twice

print(str + "TEST") # Prints concatenated string

#### Output:

Hello World! H

llo

llo World! He

Hello World!Hello World!

Hello World!TEST

* 1. **List:** List is a collection which is **ordered and changeable**. Allows **duplicate members.** Python Lists are similar to arrays in C. However, the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

#### Program:

###### # Create a List

thislist = ["apple", "banana", "cherry"] print(thislist)

###### # Access Elements

thislist = ["apple", "banana", "cherry"] print(thislist[1])

###### #range of Indexes

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] print(thislist[2:5]) # here prints only 2,3,4 positions. 5th position not included **# Change the Item Value**

thislist = ["apple", "banana", "cherry"] print(thislist)

thislist[1] = "blackcurrant" print(thislist)

**# add the items in the list** thislist.append("orange") print(thislist)

###### # Remove an items in the list

thislist = ["apple", "banana", "cherry"]

thislist.remove("banana") print(thislist)

**# Join two Lists** list1 = ["a", "b", "c"] list2 = [1, 2, 3]

list3 = list1 + list2 print(list3) **Output:**

['apple', 'banana', 'cherry'] banana

['cherry', 'orange', 'kiwi']

['apple', 'banana', 'cherry']

['apple', 'blackcurrant', 'cherry']

['apple', 'blackcurrant', 'cherry', 'orange'] ['apple', 'cherry']

['a', 'b', 'c', 1, 2, 3]

* 1. **Tuple:** Tuple is a collection which is ordered and **unchangeable. Allows duplicate members. Program:**

###### # Create a tuple

thistuple = ("Maths", "Physics", "Chemistry") print(thistuple)

###### # Print 1st item in the tuple

print(thistuple[1])

###### # add the items in the tuple

thistuple.append("computer") # error , because tuples are immutable. print(thistuple)

**Output:**

('Maths', 'Physics', 'Chemistry') Physics

**AttributeError**: 'tuple' object has no attribute 'append'

* 1. **Dictionary:** Python Dictionary is used to **store the data in a key-value pair format**. The dictionary is the data type in Python, which can simulate the **real-life data arrangement** where some

specific value exists for **some particular key**. It is the **mutable data-structure.** The dictionary is defined into element **Keys and values.**

* Keys must be a single element
* Value can be any type such as list, tuple, integer, etc.

#### Program:

###### # Creating a Dictionary

Employee = {"Name": "John", "Age": 29, "Salary":25000,"Company":"GOOGLE"} print(type(Employee))

print("Employee data: ",Employee)

###### # Creating an empty Dictionary using Dict function. it is one of the built in function in Python

Dict = {}

print("Empty Dictionary: ",Dict)

###### # Creating a Dictionary with dict() method

Dict = dict({1: 'Python', 2: 'Programming', 3:'Language'}) print("Create Dictionary by using dict(): ",Dict)

###### # uisng for loop

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"} for x in Employee:

print(x)

###### #for loop to print all the values of the dictionary

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"} for x in Employee:

print(Employee[x])

#### Output:

<class 'dict'>

Employee data: {'Name': 'John', 'Age': 29, 'salary': 25000, 'Company': 'GOOGLE'} Empty Dictionary: {}

Create Dictionary by using dict(): {1: 'Python', 2: 'Programming', 3: 'Language'} Name

Age salary Company John

29

25000

GOOGLE

#### Built in Functions: Program:

print("Sum of array: ",sum([1,2,3,4]))

print("Length of array: ",len([1,2,3,4])) print("Absolute value: ",abs(-1234)) print("Round value: ",round(1.2234))

import math as mt # importing a package print("Log value: ",mt.log(10))

#### Output:

Sum of array: 10 Length of array: 4 Absolute value: 1234

Round value: 1

Log value: 2.302585092994046

#### Functions:

**Program:**

def area(length, width): return length\*width

a = area(10,20)

print("Area of rectangle:”, a)

**Output:**

Area of rectangle: 200

**NUMPY:** NumPy stands for **Numerical Python.** 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.** It is an open-source project and you can use it freely.

**Program:**

import numpy as np # Importing libraries

a = np.array([0, 1, 2])

b = np.array([5, 5, 5]) print("Matrix A\n", a) print("Matrix B\n", b)

print("Regular matrix addition A+B\n", a + b) print("Addition using Broadcasting A+5\n", a + 5) **Output:**

Matrix A [0 1 2]

Matrix B [5 5 5]

Regular matrix addition A+B [5 6 7]

Addition using Broadcasting A+5 [5 6 7]

**Broadcasting Rules**

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

1. they are equal, or
2. one of them is 1

**Program-1:**

###### # Lets go for a 2D matrix

c = np.array([[0, 1, 2],[3, 4, 5],[6, 7, 8]])

d = np.array([[1, 2, 3],[1, 2, 3],[1, 2, 3]])

e = np.array([1, 2, 3]) print("Matrix C\n", c) print("Matrix D\n", d) print("Matrix E\n", e)

print("Regular matrix addition C+D\n", c + d) print("Addition using Broadcasting C+E\n", c + e) **Output:**

Matrix C [[0 1 2]

[3 4 5]

[6 7 8]]

Matrix D [[1 2 3]

[1 2 3]]

Matrix E [1 2 3]

Regular matrix addition C+D [[ 1 3 5]

[ 4 6 8]

[ 7 9 11]]

Addition using Broadcasting C+E [[ 1 3 5]

[ 4 6 8]

[ 7 9 11]]

**Program-2:**

M = np.ones((3, 3)) print("Matrix M:\n",M) **Output:**

Matrix M:

[[1. 1. 1.]

[1. 1. 1.]

[1. 1. 1.]]

#### Essential Python Packages: Numpy, Pandas, Matplotlib Program:

**# Load library** import numpy as np **# Create matrix**

matrix = np.array([[1, 2, 3],[4, 5, 6], [7, 8, 9]]) print("Matrix\n",matrix)

###### # Select second row

print("Second row of Matrix:",matrix[1,:]) print("Third column of Matrix:",matrix[:,2]) **Output:**

Matrix [[1 2 3]

[7 8 9]]

Second row of Matrix: [4 5 6]

Third column of Matrix: [3 6 9]

#### Matrix Properties:

**Program:**

###### # Create matrix

matrix = np**.**array([[1, 2, 3], [4, 5, 6], [ 7, 8, 9]]) print("Matrix Shape:",matrix.shape) print("Number of elements:",matrix.size) print("Number of dimensions:",matrix.ndim) print("Average of matrix:",np.mean(matrix)) print("Maximum number:",np.max(matrix))

print("Column with minimum numbers:",np.min(matrix, axis=1)) print("Diagonal of matrix:",matrix.diagonal())

**Output:**

Matrix Shape: (3, 3) Number of elements: 9 Number of dimensions: 2 Average of matrix: 5.0 Maximum number: 9

Column with minimum numbers: [1 4 7]

Diagonal of matrix: [1 5 9]

#### Matrix Operations

**Program:**

print("Flattened Matrix\n",matrix.flatten()) print("Reshaping Matrix\n",matrix.reshape(9,1)) print("Transposed Matrix\n",matrix.T)

**Output:**

Flattened Matrix [1 2 3 4 5 6 7 8 9]

Reshaping Matrix [[1]

[2]

[3]

[4]

[5]

[6]

[7]

[8]

[9]]

Transposed Matrix [[1 4 7]

[2 5 8]

[3 6 9]]

**Multiply Two Matrices:**

**Program:**

###### # Create a matrix

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

###### # Create b matrix

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

print("Matrix Addition\n",np.add(a, b)) print("Scalar Multiplication\n",np.multiply(a, b)) **Output:**

Matrix Addition [[ 2 4 2]

[ 2 4 2]

[ 2 4 10]]

Scalar Multiplication [[ 1 3 1]

[ 1 3 1]

[ 1 3 16]]

# Exercise-2

**AIM: APPLY DATA PRE-PROCESSING TECHNIQUES.**

### DESCRIPTION:

Data pre-processing is a process of **preparing the raw data and making it suitable for a machine learning model.** It is the first and crucial step while creating a machine learning model.

###### Why do we need Data Pre-processing?

A real-world data generally c**ontains noises, missing values, and maybe in an unusable format** which cannot be directly used for machine learning models. **Data pre-processing is required tasks for cleaning the data** and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

#### It involves below steps:

1. Getting the Dataset
2. Importing libraries
3. Importing datasets
4. Finding Missing Data
5. Encoding Categorical Data
6. Splitting dataset into training and test set
7. Feature scaling
8. **Getting the Dataset:** The first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the **dataset**. To use the dataset in our code, we usually put it into a **CSV file**. However, sometimes, we may also need to use an HTML or xlsx file.

CSV stands for "**Comma-Separated Values**" files; it is a file format which allows us to save the tabular data, such as spreadsheets. It is useful for huge datasets and can use these datasets in programs.

1. **Importing libraries:** In order to perform data pre-processing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are **three specific libraries that we will use for data pre-processing, which are:**
   1. **Numpy:** Numpy Python library is used for including any type **of mathematical operation in the code**. It is the fundamental package for scientific calculation in Python. It also supports to **add large, multidimensional arrays and matrices**. **So, in Python, we can import it as:**

###### import numpy as nm

Here we have used **nm**, which is a short name for Numpy, and it will be used in the whole program.

* 1. **Matplotlib:** The second library is matplotlib, which is a **Python 2D plotting library**, and with this library, we need to import a sub-library pyplot. This library is **used to plot any type of charts in Python** for the code. **It will be imported as below:**

###### import matplotlib.pyplot as mpt

Here we have used mpt as a short name for this library.

* 1. **Pandas:** The last library is the Pandas library, which is one of the most famous Python libraries and used for **importing and managing the datasets**. It is an **open-source data manipulation and analysis library**. It will be imported as below:

###### import pandas as pd

Here, we have used pd as a short name for this library.

#### Importing datasets:

**read\_csv() function:** Now to import the dataset, we will use **read\_csv() function** of **pandas library**, which is used to read a csv file and performs various operations on it. Using this function, we can read a csv file locally as well as through an URL. **We can use read\_csv function as below:**

**For eg:** data\_set= pd.read\_csv('Student.csv')

Here, **data\_set** is a name of the variable to store our dataset, and inside the function, wehave passed the name of our dataset.

#### Extracting Independent and Dependent Variables:

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset.

**For Eg:** In our dataset, there are three independent variables that are **Country, Age** and **Salary**, and one is a dependent variable which is **purchased**.

**Extracting Independent Variables:** To extract an independent variable, we will use

**iloc[ ]** method of **Pandas library**. It is used to **extract the required rows and columns** from the dataset.

**For Eg: x= data\_set.iloc[:,:-1].values**

In the above code, the **first colon (:) is used to take all the rows**, and the **second colon (:) is for all the columns.** Here we have used **(:-1), because we don't want to take the last column** as it contains the **dependent variable**. So, by doing this, we will get the matrix of features.

**Extracting dependent Variables:** To extract dependent variables, again, we will use

**Pandas .iloc[]** method.

**For Eg: y= data\_set.iloc[:, 3].values**

Here we have taken **all the rows with the last column only**. It will give the array of dependent variables.

#### Finding Missing Data:

**By calculating the mean:** In this way, we will calculate the **mean of that column or row**

which contains **any missing value** and will put it on the place of missing value.

To handle missing values, we will use **Scikit-learn** library in our code, which contains **various libraries** for building machine learning models. Here we will use **Imputer** class of **sklearn.preprocessing** library.

**For Eg:**

###### #handling missing data (Replacing missing data with the mean value)

from sklearn.preprocessing import Imputer

imputer =Imputer(missing\_values ='NaN', strategy='mean', axis = 0) **#Fitting imputer object to the independent variables x.** imputerimputer= imputer.fit(x[:, 1:3])

###### #Replacing missing data with the calculated mean value

x[:,1:3]= imputer.transform(x[:, 1:3])

1. **Encoding Categorical Data:** Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

**For Country variable:** Firstly, we will convert the country variables into categorical data. Soto do this, we will use **LabelEncoder()** class from **pre-processing** library.

Categorical data is data which has some categories such as, in our dataset; there are two categorical variables, **Country**, and **Purchased**.

**Dummy Variables:** Dummy variables are those variables which have values 0 or 1. **The 1 value gives the presence of that variable in a particular column, and rest variables become 0**. With dummy encoding, we will have a number of columns equal to the number of categories.

**For Eg:** In our dataset, we have 3 categories so it will produce three columns having 0 and 1 values. For Dummy Encoding, we will use **OneHotEncoder** class of **pre-processing** library.

1. **Splitting dataset into training and test set:** we divide our dataset into a training set and test set. This is one of the crucial steps of data pre-processing as by doing this, we can enhance the performance of our machine learning model.



**Training Set:** A subset of dataset to train the machine learning model, and we already know the output.

**Test set:** A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

###### For splitting the dataset, we will use the below lines of code:

from sklearn.model\_selection import train\_test\_split

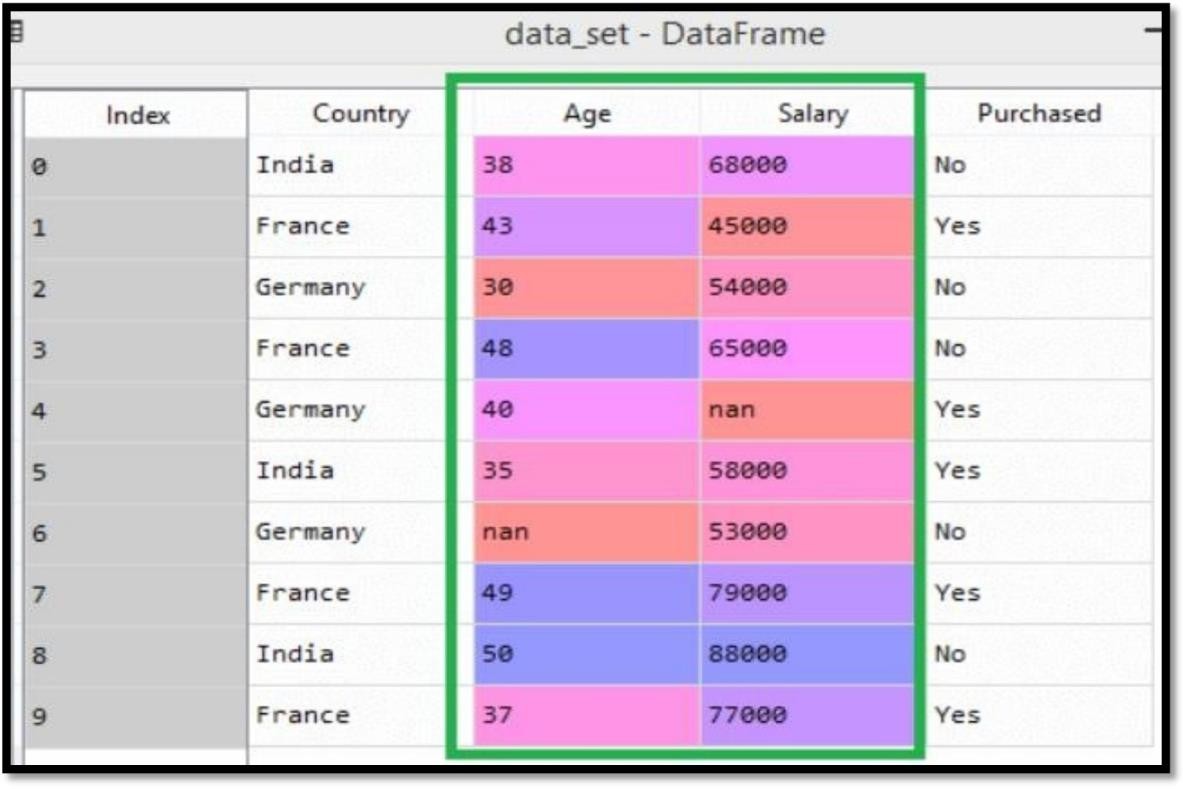
x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.2, random\_state=0)

#### Explanation:

* In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
* In the second line, we have used four variables for our output that are
  + x\_train: features for the training data
  + x\_test: features for testing data
  + y\_train: Dependent variables for training data
  + y\_test: dependent variable for testing data

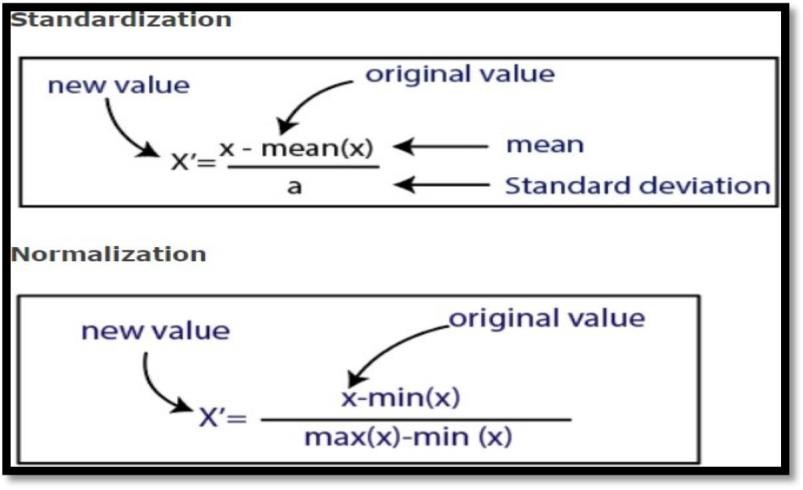
1. **Feature Scaling:** Feature scaling is the final step of data pre-processing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put in the same range and in the same scale so that no any variable dominates the other variable.

**For Eg:**

****

As we can see, the **age and salary column values are not on the same scale**. A machine learning model is based on **Euclidean distance**, and if we do not scale the variable, then it will cause some issue in our machine learning model.

If we compute **any two values from age and salary**, then **salary values will dominate the age values,** and it will produce an **incorrect result**. So to remove this issue, we need to perform feature scaling for machine learning. **There are two ways to perform feature scaling in machine learning:**

****

Here, we will use the **standardization method for our dataset**.

For feature scaling, we will import ***StandardScaler* class** of ***sklearn.preprocessing*** library as:

###### from sklearn.preprocessing import StandardScaler

Now, we will create the object of **StandardScaler class** for **independent variables or** features. And then we will fit and **transform the training dataset.**

###### st\_x= StandardScaler ()

**x\_train= st\_x.fit\_transform(x\_train)**

For test dataset, we will directly apply **transform()** function instead of **fit\_transform()** . because it is already done in training set.

**x\_test = st\_x.transform (x\_test)**

### PROGRAM:

#### #importing the libraries

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

#### # importing the dataset

dataset = pd.read\_csv(‘Preprocessing.csv’)

#### #seperating independent and dependent variables

x = dataset.iloc[:, :-1].values y = dataset.iloc[:, 3].values print(x)

print(y)

**Output:**

#### #Taking care of

**missing data**

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing\_values=np.nan, strategy='mean') imputer.fit(x[:,1:3])

x[:,1:3] = imputer.transform(x[:, 1:3]) print(x)

**Output:**

****

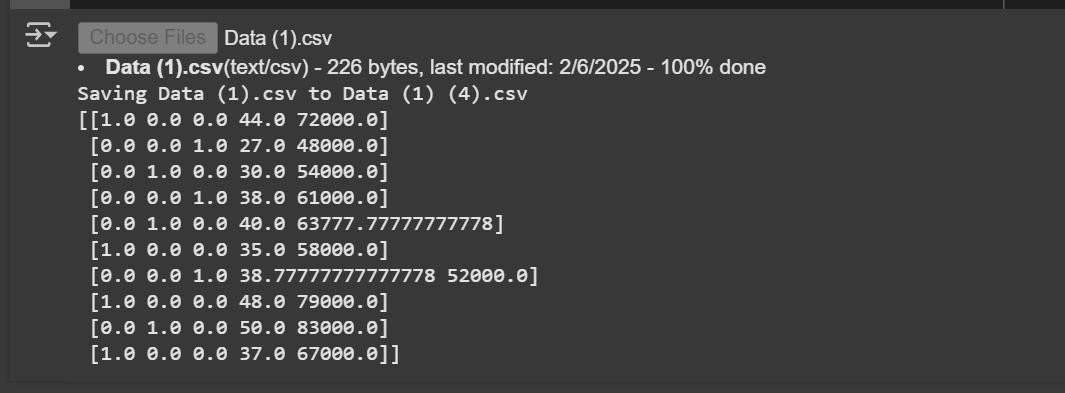
**# Encoding categorical data #Encoding the Independent Variable**

from sklearn.compose import ColumnTransformer from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer([('Country', OneHotEncoder(), [0])],remainder='passthrough') x = ct.fit\_transform(x)

print(x)

**Output:**

****

###### #Encoding the Dependent Variable

from sklearn.preprocessing import LabelEncoder le = LabelEncoder()

y = le.fit\_transform(y)

print(y)

**Output:**

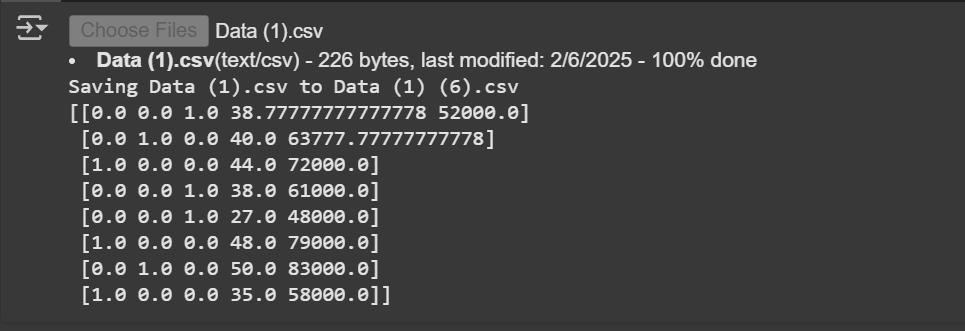


#### #splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2,random\_state = 1) print(x\_train)

**Output:**

****

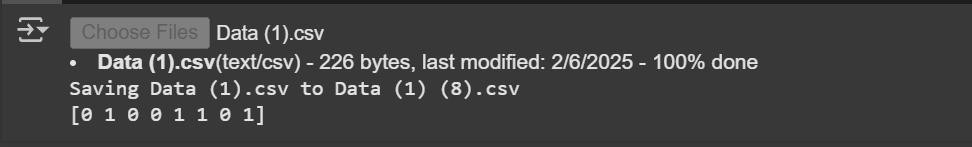
print(x\_test)

**Output:**

****

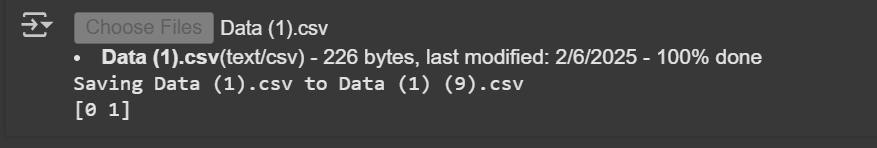
print(y\_train)

**Output:**

****

print(y\_test)

**Output:**



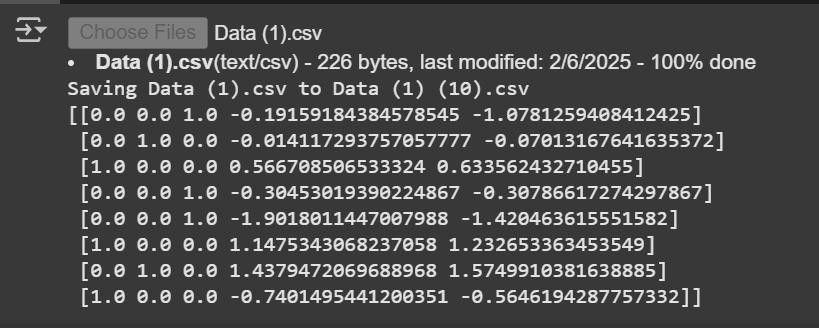
#### #Feature Scaling

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

x\_train[:, 3:] = sc.fit\_transform(x\_train[:, 3:])

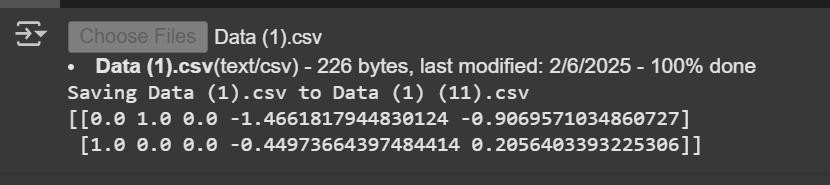
x\_test[:, 3:] = sc.transform(x\_test[:, 3:]) print(x\_train)

**Output:**

****

print(x\_test)

**Output:**

****

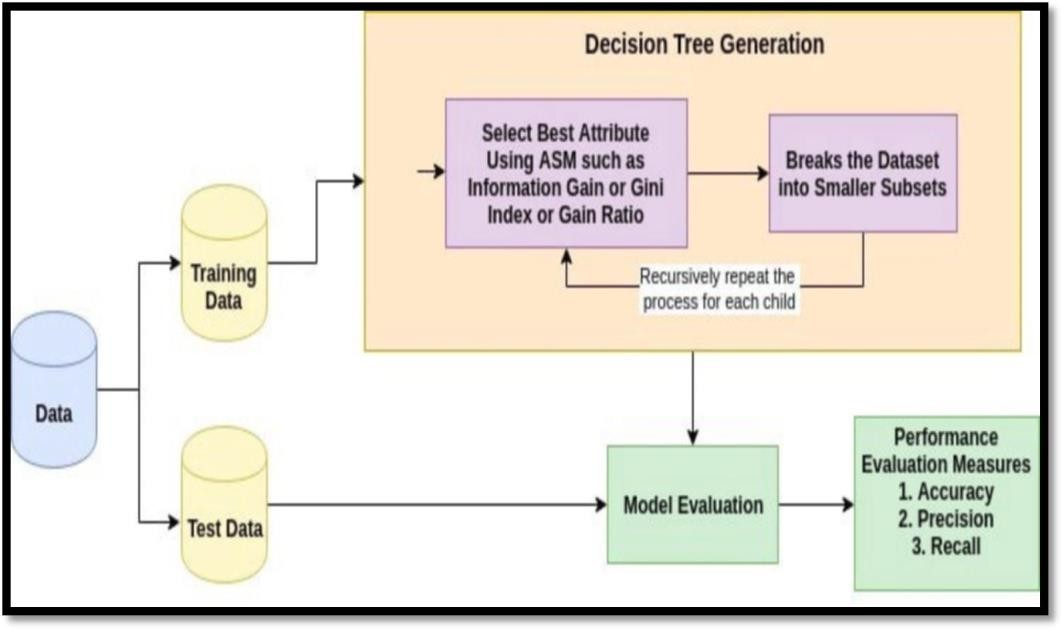
# Excercise-3

### AIM: IMPLEMENT ID3 ALGORITHM. USE AN APPROPRIATE DATA SET FOR BUILDING THE DECISION TREE AND APPLY THIS KNOWLEDGE TO CLASSIFY A NEW SAMPLE.

**DESCRIPTION:**

Decision Tree is a **Supervised learning technique** that can be used for both **classification and Regression problems**, but mostly it is preferred for solving **Classification problems.** It is a **tree- structured classifier**, where **internal nodes** represent the **features of a dataset**. **Branches represent the decision rules. Each leaf node represents the outcome.**

#### Decision Tree Generation:

****

**Decision Tree Algorithm:**

**Step-1:** Begin the tree with the root node, says S, which contains the **complete dataset**. **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM). Step-3: Divide the S into subsets** that contains possible values **for the best attributes. Step-4: Generate the decision tree node**, which **contains the best attribute.**

**Step-5: Recursively make new decision trees** using the subsets of the dataset created **in step -3.** Continue this process until a **stage is reached** where you **cannot further classify** the **nodes** and called the **final node as a leaf node.**

#### Attribute Selection Measure:

It is a heuristic for **selecting the splitting criterion** that **“best” separates a given data partition, D**, of a

###### class-labeled training tuples into individual classes. The Three Important Attribute Selection measures are Information gain: ID3/C4.5

**Gain Ratio : C4.5 Gini Index CART**

**ID3 Algorithm:** Entropy is the main concept of this algorithm, which helps determine a feature or attribute that gives maximum information about a class is called Information gain or ID3 algorithm.

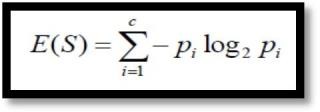
**Steps in ID3 algorithm:**

###### It begins with the original set S as the root node.

1. On **each iteration of the algorithm**, it iterates through the very unused attribute of the set S and

**calculates Entropy(H) and Information gain(IG)** of this attribute.

1. It then selects the attribute which has the **smallest Entropy or Largest Information gain.**
2. The **set S is then split by the selected attribute** to produce a subset of the data.

**Entropy:** By using this method, we can **reduce the level of entropy** from the **root node to the leaf node.**

#### Entropy = - (p(0) \* log(P(0)) + p(1) \* log(P(1)))

Where **pᵢ** is the probability of **randomly picking an element of class i** (i.e. the proportion of the dataset made up of class i).

**Information Gain:** Information gain is used for determining the best features/attributes that render maximum information about a class. It follows the concept of entropy while aiming at decreasing the level of entropy, beginning from the root node to the leaf nodes.

**Information Gain = entropy (parent) – [average entropy (children)] Information Gain = Entropy(S) – [(weighted Avg.) \* Entropy (Each Feature)]**

#### It involves below steps:

1. **Install the Packages:**
   1. **Pip:** It is a standard package manager used to install and maintain packages for Python. The Python standard library comes with a collection of built-in functions and built-in packages. **Note:** If you have Python version

3.4 or later, PIP is included by default.

**!pip installs decision-tree-id3**

* 1. **Six: It is a Python 2 and 3 compatibility library**. It provides **utility functions for smoothing over the differences between the Python versions** with the goal of writing Python code that is **compatible on both Python versions.**

**import six**

* 1. **Sys:** It is a **sys module** in Python provides **various functions and variables** that are used to

**manipulate different parts** of the **Python runtime environment.**

**import sys**

**sys.modules ['sklearn.externals.six'] = six**

* 1. **ID3 Estimator: decision-tree-id3** is a module created to derive decision trees using the **ID3 algorithm**. It is written to be compatible with **Scikit-learn’s API**

#### Importing datasets:

**read\_csv() function:** Now to import the dataset, we will use **read\_csv() function** of **pandas library**, which is used to read a csv file and performs various operations on it. Using this function, we can read a csv file locally as well as through an URL. **We can use read\_csv function as below:**

###### For Eg: tennis\_data = pd.read\_csv ('PlayTennis.csv')

Here, **tennis\_data** is a name of the variable to store our dataset, and inside the function, we have passed the name of our dataset.

And also to print the head (). The head() returns the first n rows for the object based on position.

**tennis\_data.head(5)**

#### Extracting Independent and Dependent Variables:

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset.

**For Eg:** In our dataset, there are **FOUR independent variables** that are **Outlook, Temperature, Humidity, Wind**, and one is a dependent variable which is **Play Tennis**

1. **Encoding Categorical Data:** Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

**For Independent variable:** Firstly, we will convert the country variables into categorical data. So to do this, we will use **LabelEncoder()** class from **pre-processing** library.

Categorical data is data which has some categories such as, in our dataset; there are four categorical variable, **Outlook, Temperature, Humidity, Wind**

from sklearn.preprocessing import LabelEncoder Le = Label Encoder()

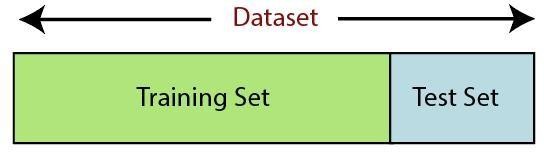
###### # converting each column

tennis\_data ['Outlook'] = Le.fit\_transform(tennis\_data['Outlook']) tennis\_data['Temperature'] = Le.fit\_transform(tennis\_data['Temperature']) tennis\_data['Humidity'] = Le.fit\_transform(tennis\_data['Humidity']) tennis\_data['Wind'] = Le.fit\_transform(tennis\_data['Wind']) **Separating independent and Dependent Variable:**

y = tennis\_data['Play Tennis']

X = tennis\_data. Drop(['Play Tennis'],axis=1)

1. **Splitting dataset into training and test set:** we divide our dataset into a training set and test set. This is one of the crucial steps of data pre-processing as by doing this, we can enhance the performance of our machine learning model.



**Training Set:** A subset of dataset to train the machine learning model, and we already know the output. **Test set:** A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

###### For splitting the dataset, we will use the below lines of code:

from sklearn.model\_selection import train\_test\_split

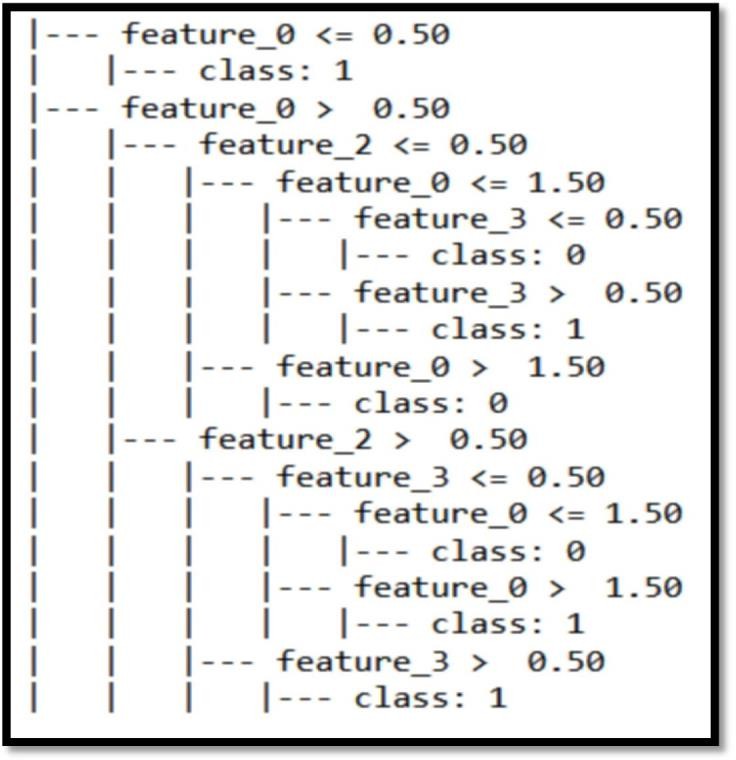
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.9, random\_state = 0)

#### Explanation:

* In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
* In the second line, we have used four variables for our output that are
  + x\_train: features for the training data
  + x\_test: features for testing data
  + y\_train: Dependent variables for training data
  + y\_test: dependent variable for testing data

1. **Decision Tree Creation: (Print Text Representation):** Here we will create a random decision tree with the help of **sci-kit learn library.** We will use the **Play Tennis dataset** for decision tree creation.
2. Firstly, We need to import the **DecisionTreeClassifier** from [s**klearn.tree module**.](https://scikit-learn.org/stable/index.html)
3. **Invoking sklearn export text – O**nce we have created the decision tree, We can export the decision tree into textual format. But to achieve this, we need to **import export\_text** from **sklearn.tree.export package.** After it, We will invoke the export\_text() function by passing the decision tree **object as an argument**. we can easily solve the mystery of the decision tree with the above self-explanatory **export\_text() function**. Here **show\_weights are set are True**. It will give more info about each node. Let’s run the complete code together and check the output.

**from sklearn import tree print(tree.export\_text(clf))**

****

#### Decision Tree Classification (Plot-Tree):

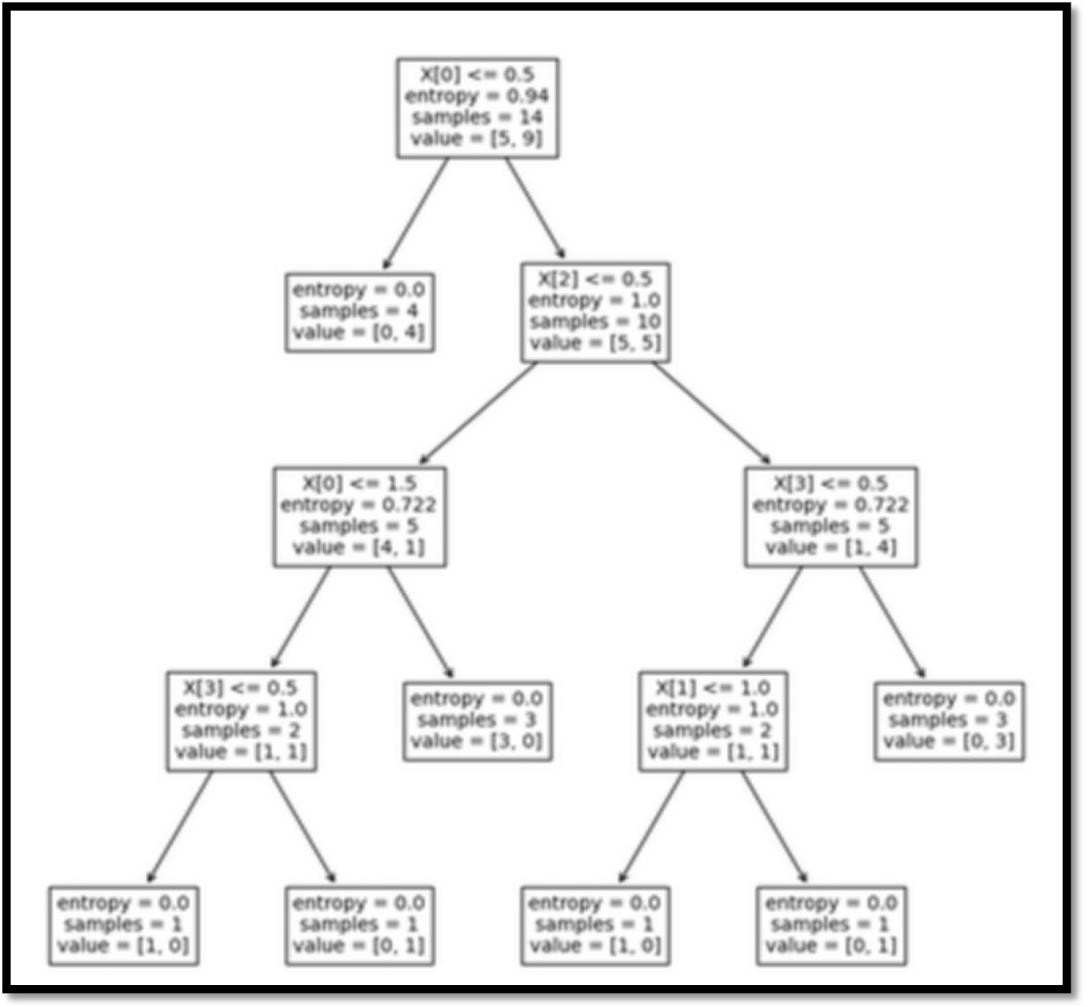
The **plot\_tree method** was added to sklearn**.** It requires **matplotlib** to be installed. It allows us to easily produce figure of the tree. The more information **about plot\_tree arguments** are in the [docs](https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html).

clf = tree.DecisionTreeClassifier(criterion = 'entropy') clf = clf.fit(x\_train, y\_train)

clf.fit(x,y)

Here, we can also **visualize the size of the Tree** and also include the **Font Size.**

fig, ax = plt.subplots(figsize=(10, 10)) tree.plot\_tree(clf, fontsize=10) plt.show()



#### Fitting the Model & Predictions:

Here we can use Python’s **sklearn library** holds tons of **modules that help to build predictive models.** It contains tools for **data splitting, pre-processing, feature selection, tuning and** [**supervised –**](https://dataaspirant.com/2014/09/19/supervised-and-unsupervised-learning/)[**unsupervised learning**](https://dataaspirant.com/2014/09/19/supervised-and-unsupervised-learning/) **algorithms.**

Now we fit Decision tree algorithm on training data, predicting labels for validation dataset and printing the accuracy of the model using various parameters.

**DecisionTreeClassifier():** This is the classifier function for **DecisionTree.** It is the main function for implementing the algorithms. Here we can apply the Criteria is **“Entropy”**

**clf = tree.DecisionTreeClassifier(criterion = 'entropy') clf = clf.fit(x\_train, y\_train)**

**#predections**

**x\_pred = clf.predict(x\_test)**

#### Classification on Decision Tree:

**#classification report to check accuracy, precision, recall etc.**

**from sklearn.metrics import classification\_report**

**print(classification\_report(y\_test, X\_pred))**

**Import the ID3 Algorithm: Here we can calculate the Accuracy Score. T**he function **accuracy\_score()** will be used to **print accuracy of Decision Tree algorithm**. By accuracy, we mean the **ratio of the correctly predicted data points to all the predicted data points**. **Accuracy as a metric** helps to understand the **effectiveness of our algorithm.**

**# import the ID3 Estimator from id3 import Id3Estimator estimator = Id3Estimator() estimator.fit (X\_train, y\_train)**

**X\_pred = estimator.predict(X\_test) #showing classification report print(classification\_report(y\_test, X\_pred)) print(accuracy\_score(y\_test, X\_pred))**

### PROGRAM:

#### #Importing lib for data pre-processing and algorithm building

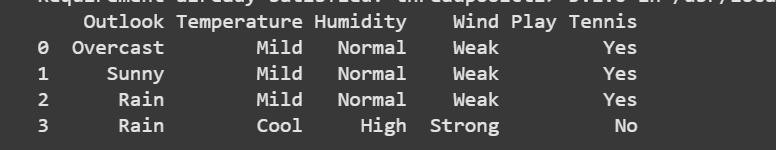
!pip install decision-tree-id3 import matplotlib.pyplot as plt

import pandas as pd # for reading data set import six

import sys sys.modules['sklearn.externals.six'] = six from id3 import Id3Estimator **#Reading tennis data set**

tennis\_data = pd.read\_csv('PlayTennis.csv') #showing first 5 records tennis\_data.head(5)

**Output:**

****

#### #converting data to numeric

from sklearn.preprocessing import LabelEncoder

Le = LabelEncoder()

#### # converting each column

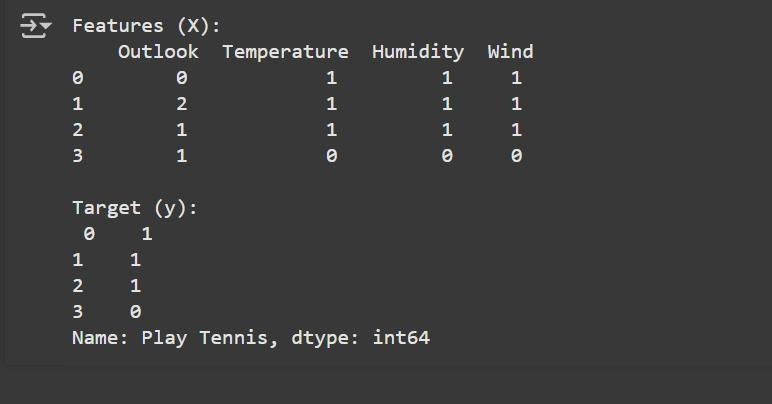
tennis\_data['Outlook'] = Le.fit\_transform(tennis\_data['Outlook']) tennis\_data['Temperature'] = Le.fit\_transform(tennis\_data['Temperature']) tennis\_data['Humidity'] = Le.fit\_transform(tennis\_data['Humidity']) tennis\_data['Wind'] = Le.fit\_transform(tennis\_data['Wind']) tennis\_data['Play Tennis'] = Le.fit\_transform(tennis\_data['Play Tennis']) **#seprating target and features**

y = tennis\_data['Play Tennis']

y = tennis\_data.drop(['Play Tennis'],axis=1) print(x)

print(y)

**Output:**

****

#### # Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.9, random\_state = 0)

#### # Fitting the model

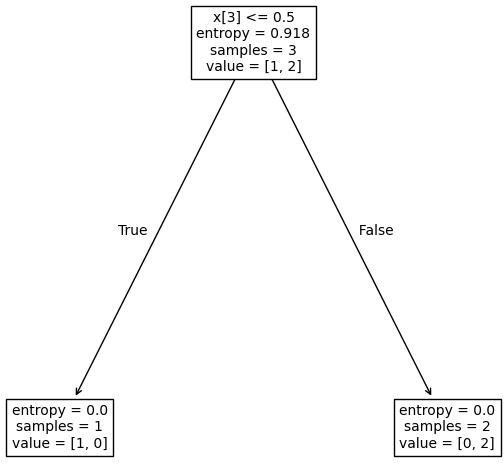
from sklearn import tree

clf = tree.DecisionTreeClassifier(criterion = 'entropy') clf = clf.fit(x\_train, y\_train)

clf.fit(x,y)

fig, ax = plt.subplots(figsize=(10, 10)) tree.plot\_tree(clf, fontsize=10) plt.show()

**Output:**

****

#### #predections

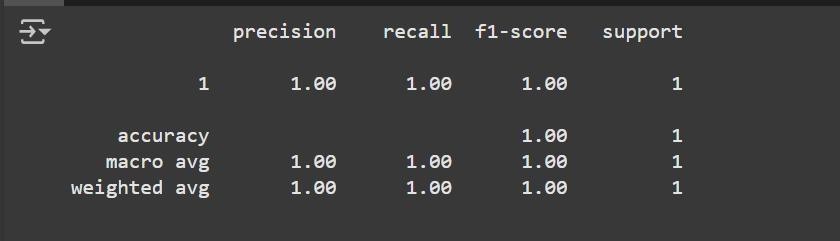
x\_pred = clf.predict(x\_test)

#### #classification report to check accuracy,precision,recall etc.

from sklearn.tree import DecisionTreeClassifier estimator = DecisionTreeClassifier() estimator.fit(x\_train, y\_train)

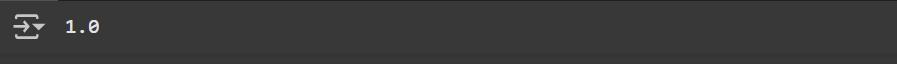
x\_pred = estimator.predict(x\_test)

from sklearn.metrics import classification\_report print(classification\_report(y\_test, x\_pred, zero\_division=1)) **Output:**



#### # Metrics library

from sklearn.metrics import accuracy\_score print(accuracy\_score(y\_test, x\_pred)) **Output:**



# Exercise-4(A)

### AIM: IMPLEMENT PERCEPTRON-AND RULE USING AND GATE USING PERCEPTRON TRAINING RULE

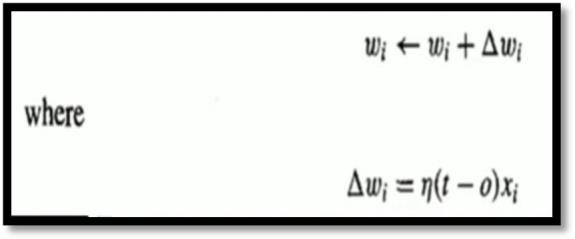
**DESCRIPTION:**

**Perceptron: A Perceptron is an Artificial Neuron.** Frank Rosenblatt (1928 – 1971) was an American psychologist notable in the field of Artificial Intelligence. In 1957, he "invented" a Perceptron program, on an IBM 704 computer at Cornell Aeronautical Laboratory. The Neurons **receive input from our senses by electrical signals and store information to make decisions based on previous input.** So, A Perceptron Unit is **used to build the ANN System.**

Perceptron is a **Machine Learning algorithm** for **supervised learning** of various **binary classification tasks.**

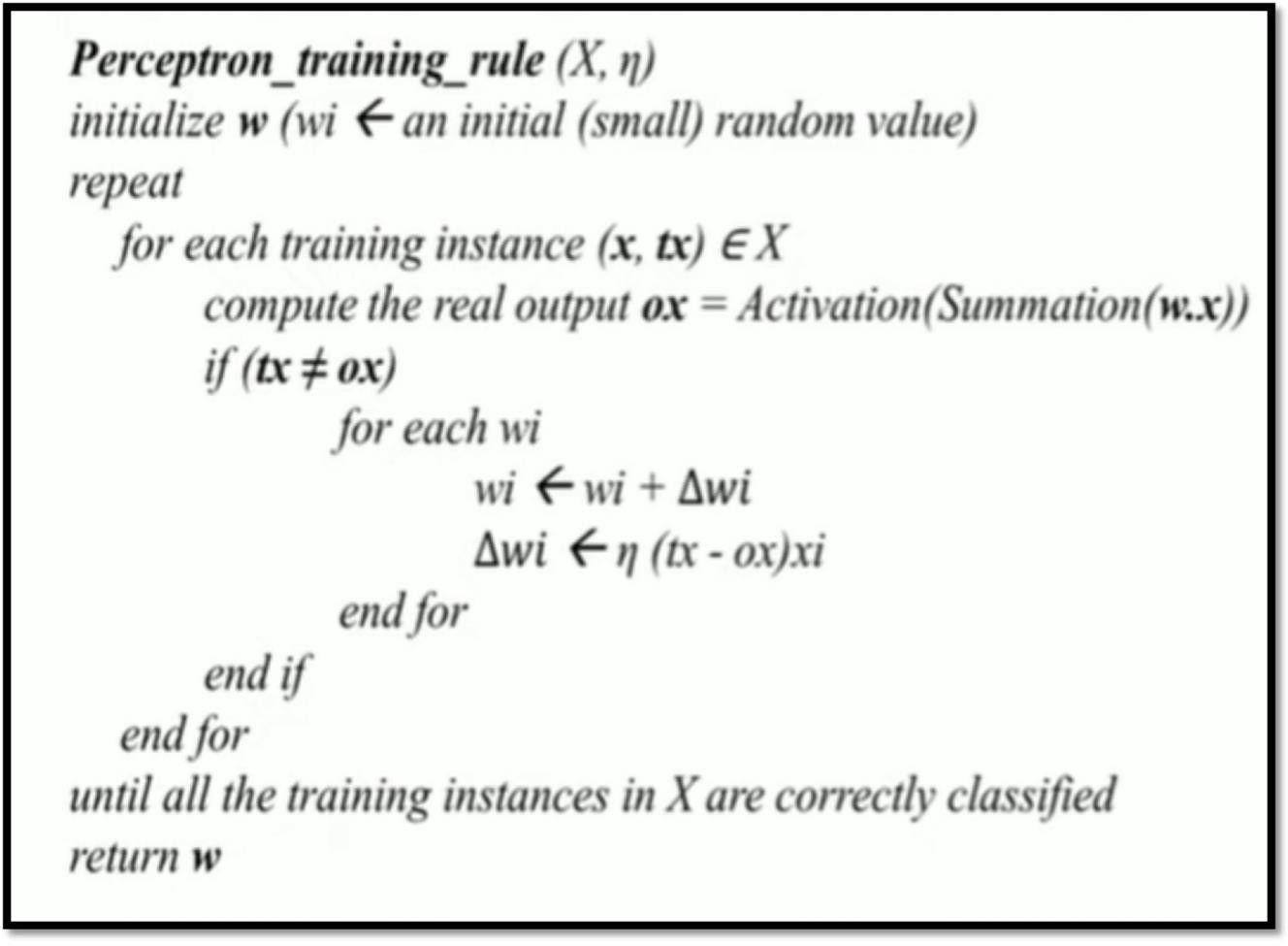
###### The original Perceptron was designed to take a number of binary inputs, and produce one binary output (0 or 1).

**Perceptron Learning Rule:** In this Rule, First begin **with random weights,** then iteratively apply the perceptron to each training example, **modify the perceptron weights whenever it misclassifies an example**. This process is repeated, iterating through the training examples as many times as needed until the **perceptron classifies all training examples correctly**. Here, **Weights are modified at each step according to the Perceptron Training Rule.**

* This rule revises the **weight, wi** associated with the **input xi according to the rule**.
* Here, t = Targeted Output

o = Actual Output

**Algorithm:**

****

**Below the steps involved the Perceptron-AND Algorithm:**

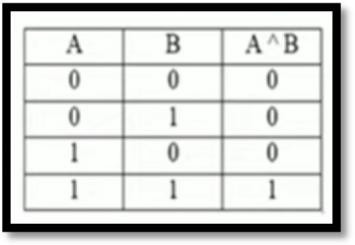
1. **Install the Packages:**

**(a) Numpy:** Numpy Python library is used for including any type **of mathematical operation in the code**. It is the fundamental package for scientific calculation in Python. It also supports to **add large, multidimensional arrays and matrices**. **So, in Python, we can import it as:**

**import numpy as np**

#### Initialize the Input and Output Variables

**The Truth Table of AND Gate:**

****

**np.array([ [0,0] ,[0,1] ,[1,0] ,[1,1] ])**

**labels= np.array([0, 0, 0, 1]) #AND Gate**

#### Initialize the Network Parameters:

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset.

###### # epoch(Training Iterations)

**# Bias=1, Learning Rate=(0to1) # input Weights (w1,w2)**

epochs = int(input('Enter the Epochs:')) threshold= float(input('Enter the Threshold:'))

Learning\_rate = float(input('Enter the Learning Rate')) a=[]

a.append(float(input('Enter the weights for x0'))) a.append(float(input('Enter the weights for x1'))) w=np.array(a)

for j in range(0,epoch):

print("epoch", j) global\_delta = 0

for i in range(0,features.shape[0]): #print(features[i])

actual = labels[i] instance = features[i] x0 = instance[0]

x1 = instance[1]

sum\_unit = x0 \* w[0] + x1 \* w[1] if sum\_unit > threshold:

fire =1 else:

fire =0

#### Print the Classification Report:

delta = actual - fire

global\_delta = global\_delta + abs(delta)

print("prediction:",fire,"where as actual was", actual, " (error:",delta,")") w[0] = w[0] + delta \* learning\_rate

w[1] = w[1] + delta \* learning\_rate print(" ")

if global\_delta == 0: break

print w

### PROGRAM:

import numpy as np

##### # Step-1: Initialize Input and Output Variables

features=np.array([

[0,0]

,[0,1]

,[1,0]

,[1,1]

])

labels= np.array([0, 0, 0, 1]) **#AND Gate**

##### # step-2: Intialize the netwrok parameters

# epoch(Training Iterations)

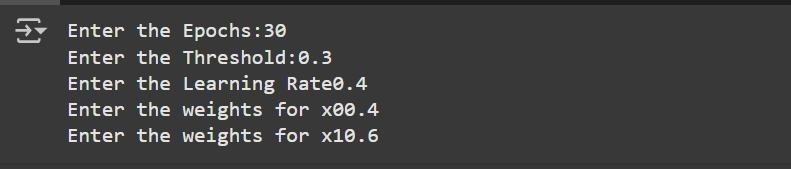
# Bias=1, Learning Rate=(0to1) # input Weights (w1,w2)

epochs = int(input('Enter the Epochs:')) threshold= float(input('Enter the Threshold:'))

Learning\_rate = float(input('Enter the Learning Rate')) a=[]

a.append(float(input('Enter the weights for x0'))) a.append(float(input('Enter the weights for x1'))) w=np.array(a)

**Output:**

****

for j in range(0,epoch): print("epoch:",j) global\_delta = 0

for i in range(0,features.shape[0]): actual = labels[i]

instance = features[i] x0 = instance[0]

x1 = instance[1]

sum\_unit = x0\*w[0]+x1\*w[1] if sum\_unit > threshold:

fire=1 else:

fire=0

delta = actual - fire

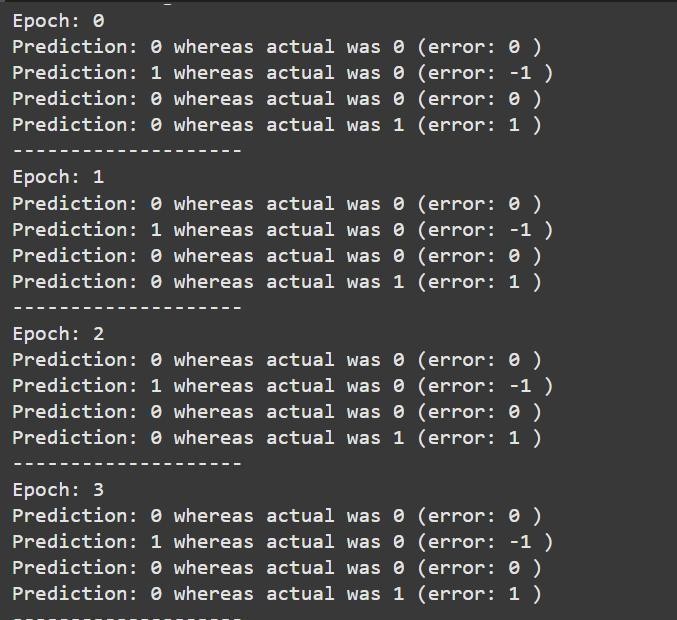
global\_delta = global\_delta+abs(delta)

print("Prediction:",fire,"where as actual was",actual,"(error:",delta,")") w[0] = w[0]+delta\*learning\_rate

w[1] = w[1]+delta\*learning\_rate print(" ")

if global\_delta == 0: break

**Output:**



print(w)

**Output:**

## EXERCISE-4(B)

**AIM: Build An Artificial Neural Network by Implementing the Back Propagation Algorithm and Test the same using Appropriate Data Sets.**

### DESCRIPTION:

An **Artificial Neuron Network (neural network**) is a computational model that mimics the way **nerve cells work in the human brain**. ANN algorithm accepts only **numeric and structured data.**

ANNs can learn and model **non-linear and complicated interactions**, which is critical since many of the relationships between **inputs and outputs in real life** are non-linear and complex.

#### Architecture:

There are three layers in the network architecture:

The Input Layer,

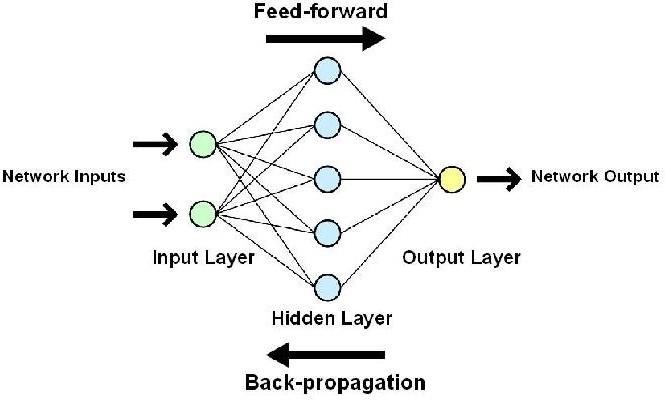
The Hidden Layer (more than one), and The Output Layer.

Because of the numerous layers are sometimes referred to as the **MLP (Multi-Layer Perceptron).**

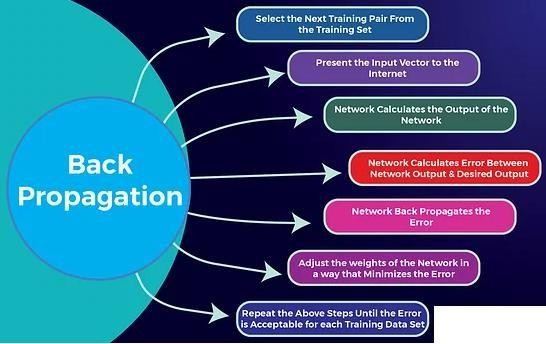
###### The Multi-Layer-Perceptron was first introduced by M. Minsky and S. Papert in 1969. It is an extended Perceptron and has one more hidden neuron layers between its input and output layers. A Multi-Layer-Perceptron is able to solve every logical operation, including the XOR problem.

Multilayer perceptron's (MLPs) are **feed forward neural networks** trained with the **standard back propagation algorithm.** Back propagation is the most important step for training artificial neural networks. **By using the Back propagation, is a procedure to repeatedly adjust the weights so as to minimize the difference between actual output and desired output.**

#### The Architecture of ANN is:

****

**Back Propagation Algorithm Steps:**

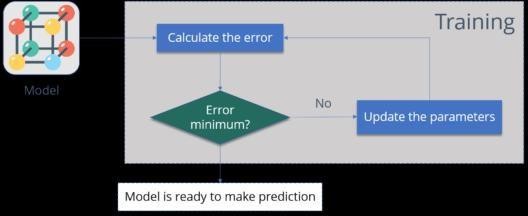


In an **A**[**rtificial**](https://intellipaat.com/blog/what-is-artificial-neural-network/)[**Neural**](https://intellipaat.com/blog/what-is-artificial-neural-network/)[**Network**](https://intellipaat.com/blog/what-is-artificial-neural-network/), the **values of weights and biases** are **randomly initialized**. Due to **random initialization**, the [**neural network**](https://intellipaat.com/blog/tutorial/machine-learning-tutorial/neural-network-tutorial/) **probably has errors for the given inputs.** So, We need to **reduce error values** as much as possible. So, for **reducing these error values,** we need a mechanism that can **compare the desired output of the neural network** with the **Target network’s output.**

Suppose, that **consists of errors** and **adjusts its weights and biases** such that it **gets closer to the desired output after each iteration**. For this, we **train the network** such that it **back propagates and updates the weights and biases**. This is the concept of the **back propagation algorithm.**

Below are the steps that an artificial neural network follows to **gain maximum accuracy** and **minimize error values**

One way to train our model is called as Back propagation. Consider the diagram below:



**Let us summarize the steps for you:**

1. **Calculate the error** – How far is your model output from the actual output.
2. **Minimum Error** – Check whether the error is minimized or not.
3. **Update the parameters** – If the error is huge then, update the parameters (weights and biases). After that again check the error. Repeat the process until the error becomes minimum.
4. **Model is ready to make a prediction** – Once the error becomes minimum, you can feed some inputs to your model and it will produce the output.

#### It involves below steps:

**2. Install the Packages:**

1. **Numpy:** Numpy Python library is used for including any type **of mathematical operation in the code**. It is the fundamental package for scientific calculation in Python. It also supports to **add large, multidimensional arrays and matrices**. **So, in Python, we can import it as:**

#### import numpy as np Here we have taken with One Example

###### The above network contains the following:

* 1. **Two inputs:** X1 and X2
  2. **Two hidden neurons:** h1 and h2
  3. **Two output neurons:** o1 and o2
  4. **Two biases:** b1 and b2

#### Below the steps involved the Back Propagation

**Step – 1: Forward Propagation Step – 2: Backward Propagation**

**Step – 3: Putting all the values together and calculating the updated weight value**

#### Step-1: Forward Propagation:

1. **First, we have to initialize the Weights: ie, here we have to read the input from the user**

Here we have initialized two inputs: X1 and X2

###### For Eg: input\_data = np.array(input("Enter input data separated by commas: ").split(','), dtype=float)

1. **Next, we have to read the Number of Hidden Layers from user.**

###### For Eg: n\_hidden\_layers = int(input("Enter number of hidden layers: "))

1. **Next, we have to read the Number of Output Neurons from user.**

###### For Eg: n\_output\_neurons = int(input("Enter number of output neurons: "))

1. **Next, Read the Bias Values from User**

**For Eg: bias\_values = []**

**for i in range(n\_hidden\_layers+1):**

**bias\_values.append(np.array(input(f"Enter bias values for layer {i+1} separated by commas: ").split(','), dtype=float))**

1. **Next, we have to give the weights for the given inputs X1 and X2. For Eg: for i in range(n\_hidden\_layers):**

**hidden\_layer\_sizes.append(int(input(f"Enter number of neurons in hidden layer {i+1}: "))) if i == 0:**

**weights.append(np.array(input(f"Enter weights for input layer to hidden layer {i+1} separated by commas: ").split(','), dtype=float).reshape(input\_layer\_size, hidden\_layer\_sizes[i]))**

**else:**

**weights.append(np.array(input(f"Enter weights for hidden layer {i} to hidden layer**

**{i+1} separated by commas: ").split(','), dtype=float).reshape(hidden\_layer\_sizes[i-1], hidden\_layer\_sizes[i]))**

**weights.append(np.array(input(f"Enter weights for hidden layer {n\_hidden\_layers} to output layer separated by commas: ").split(','), dtype=float).reshape(hidden\_layer\_sizes[-1],**

**n\_output\_neurons))**

1. **Next, we have to Initialize the Learning Rate and Number of Iterations**

###### learning\_rate = 0.1

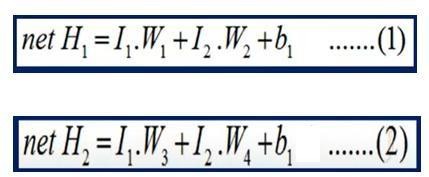
**num\_iterations = 1000**

1. **Next, Read the Target Output from user**

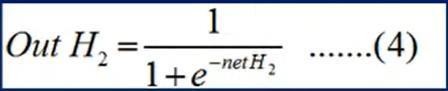
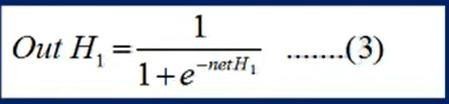
**For Eg: target = np.array(input(f"Enter target output separated by commas: ").split(','), dtype=float)**

#### Next, we have to Train the Network (Feed Forward Propagation)

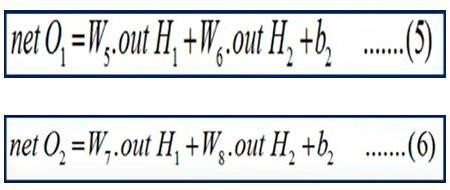
* + Here I1, I2 are the Input Layers and W1, W2, W3, W4 are the corresponding Weights.
  + From the above Network, First we can Calculate the Net H1 and Net H2.



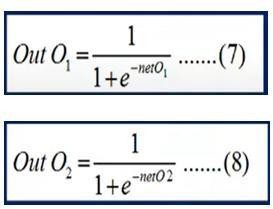
**After Calculating net H1 and net H2. compute Out H1 and Out H2 as follows:**

****

**Similarly, we can Calculating net O1 and net O2.**

****

**From the above Equations, we can calculate Out O1 and Out O2 as follows:**

****

**# Feedforward**

**hidden\_layers = [input\_data]**

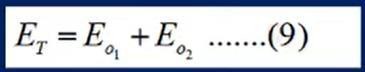
**for j in range(len(hidden\_layer\_sizes)):**

**hidden\_layer = sigmoid(np.dot(hidden\_layers[j], weights[j]) + bias\_values[j]) hidden\_layers.append(hidden\_layer)**

**output = sigmoid(np.dot(hidden\_layers[-1], weights[-1]) + bias\_values[-1])**

#### Step-2: Back Propagation:

* + When we got the Error. i.e, Target Output O1 and Actual Output O1.
  + Similarly, Target Output O2 and Actual Output O2.
  + So that we have to move Backward and we have to adjust the weights of W5, W6, W7, W8.
  + Now we have to Calculate the Output Values.
  + If we get the Expected Output and Actual Outputs are Same, We will stop.

**Next, we can Calculating the Total Error ** **For Eg: # Backpropagation**

**error = output - target**

**delta = error \* sigmoid\_derivative(output) deltas = [delta]**

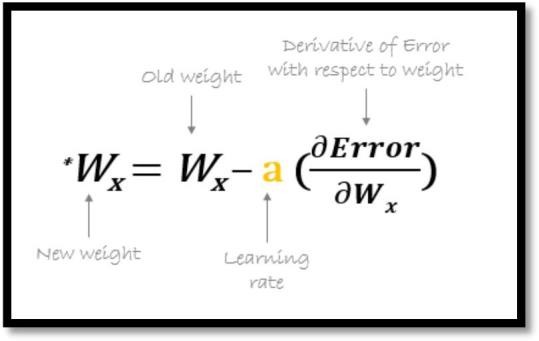
**for j in range(len(hidden\_layer\_sizes)-1, -1, -1):**

**delta = np.dot(deltas[-1], weights[j+1].T) \* sigmoid\_derivative(hidden\_layers[j+1]) deltas.append(delta)**

**deltas.reverse()**

**If, Error Comes, we need to Update the Weights and Biases :**

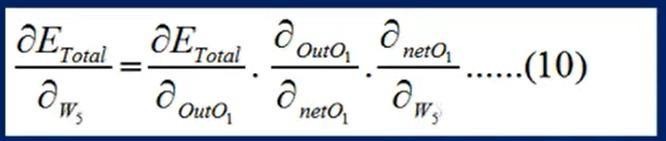
**Here, we need to use the Formula for Gradient Descent to update the Weights.**

****

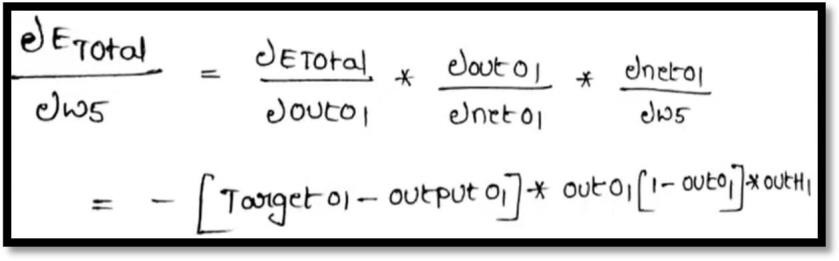
**Suppose we have to update the Weight W5:**

****

**This can be Written as :**

****

**The Final Calculation is :**

****

**# Update weights and biases**

**for j in range(len(hidden\_layer\_sizes)+1): if j == 0:**

**layer\_input = input\_data else:**

**layer\_input = hidden\_layers[j]**

**dtran = np.transpose(deltas[j].reshape(-1,1)) x = np.dot(layer\_input.reshape(-1,1), dtran) #print(x.shape,weights[j].shape)**

**weights[j] -= learning\_rate \* x**

#### Step – 3: Putting all the values together and calculating the updated weight value

**# Print final output and weights print(f"Final output: {output}") print(f"Target output: {target}") for i in range(len(weights)):**

**print(f"Layer {i+1} weights: {weights[i]}") for i in range(len(bias\_values)):**

**print(f"Layer {i+1} biases: {bias\_values[i]}")**

### PROGRAM:

###### #Importing lib for data pre-processing and algorithm building

import numpy as np

###### # Define sigmoid activation function

def sigmoid(x):

return 1 / (1 + np.exp(-x))

###### # Define derivative of sigmoid function

def sigmoid\_derivative(x): return x \* (1 - x)

###### # Read input from user

input\_data = np.array(input("Enter input data separated by commas: ").split(','), dtype=float)

**Output:**

****

###### #Read number of hidden layers from user

n\_hidden\_layers = int(input("Enter number of hidden layers: "))

**Output:**

****

###### # Read number of output neurons from user

n\_output\_neurons = int(input("Enter number of output neurons: "))

**Output:**



###### # Read bias values from user

bias\_values = []

for i in range(n\_hidden\_layers+1):

bias\_values.append(np.array(input(f"Enter bias values for layer {i+1} separated by commas: ").split(','), dtype=float))

**Output:**

****

**# Define neural network architecture** input\_layer\_size = input\_data.shape[0] hidden\_layer\_sizes = []

weights = []

###### # Assign the weights for the given inputs

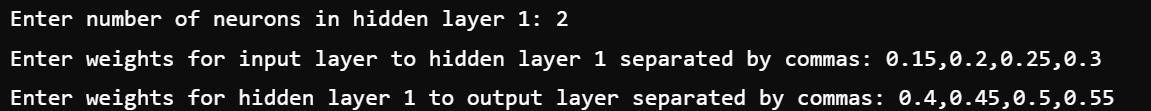
for i in range(n\_hidden\_layers):

hidden\_layer\_sizes.append(int(input(f"Enter number of neurons in hidden layer {i+1}: "))) if i == 0:

weights.append(np.array(input(f"Enter weights for input layer to hidden layer {i+1} separated by commas: ").split(','), dtype=float).reshape(input\_layer\_size,hidden\_layer\_sizes[i]))

else:

weights.append(np.array(input(f"Enter weights for hidden layer {i} to hidden layer {i+1} separated by commas: ").split(','), dtype=float).reshape(hidden\_layer\_sizes[i-1], hidden\_layer\_sizes[i])) weights.append(np.array(input(f"Enter weights for hidden layer {n\_hidden\_layers} to output layer separated by commas: ").split(','), dtype=float).reshape(hidden\_layer\_sizes[-1], n\_output\_neurons)) **Output:**

**#**

###### Set hyperparameters

learning\_rate = 0.1

num\_iterations = 1000

###### # Read target output from user

target = np.array(input(f"Enter target output separated by commas: ").split(','), dtype=float)

**Output:**



###### # Train the neural network

for i in range(num\_iterations):

###### # Feed forward Network

hidden\_layers = [input\_data]

for j in range(len(hidden\_layer\_sizes)):

hidden\_layer = sigmoid(np.dot(hidden\_layers[j], weights[j]) + bias\_values[j]) hidden\_layers.append(hidden\_layer)

output = sigmoid(np.dot(hidden\_layers[-1], weights[-1]) + bias\_values[-1])

###### # Backpropagation

error = output - target

delta = error \* sigmoid\_derivative(output) deltas = [delta]

for j in range(len(hidden\_layer\_sizes)-1, -1, -1): delta = np.dot(deltas[-1], weights[j+1].T) \*

sigmoid\_derivative(hidden\_layers[j+1]) deltas.append(delta)

deltas.reverse()

###### # Update weights and biases

for j in range(len(hidden\_layer\_sizes)+1): if j == 0:

layer\_input = input\_data else:

layer\_input = hidden\_layers[j]

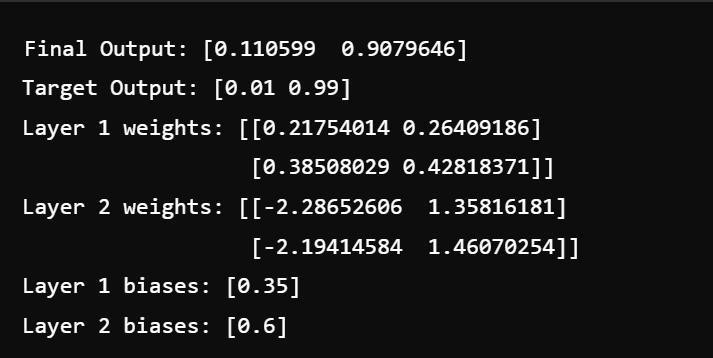
dtran = np.transpose(deltas[j].reshape(-1,1)) x = np.dot(layer\_input.reshape(-1,1), dtran) #print(x.shape,weights[j].shape)

weights[j] -= learning\_rate \* x **# Print final output and weights** print(f"Final output: {output}") print(f"Target output: {target}") for i in range(len(weights)):

print(f"Layer {i+1} weights: {weights[i]}") for i in range(len(bias\_values)):

print(f"Layer {i+1} biases: {bias\_values[i]}")

**Output:**

****

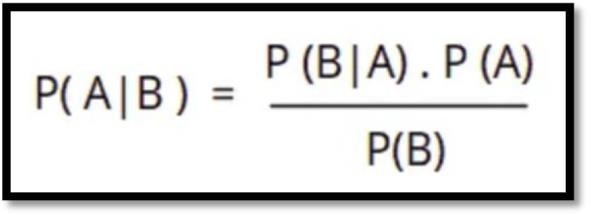
## EXERCISE-5

### AIM: IMPLEMENT NAÏVE BAYESIAN CLASSIFIER FOR A SAMPLE TRAINING DATA SET STORED AS A ‘.CSV’ FILE. COMPUTE THE ACCURACY OF THE CLASSIFIER, CONSIDERING FEW TEST DATA SETS.

**DESCRIPTION:**

**Naive Bayes** is a **statistical classification technique** based on the **Bayes Theorem** and one of the simplest [**Supervised Learning**](https://hands-on.cloud/introduction-to-supervised-machine-learning/) **algorithms. A** [**Naive Bayes classifier**](https://en.wikipedia.org/wiki/Naive_Bayes_classifier)assumes that the effect of a **particular feature in a class is independent of other features** and is based on **Bayes’ theorem**.

###### [Bayes’ theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem) is a mathematical equation used in probability and statistics to calculate conditional probability.

****In other words, you can use this theorem to **calculate the probability of an event with functions** like the **Gaussian Probability Density function** based on its **association with another event. The simple formula of the Bayes theorem is:**

###### Where P (A) and P (B) are two independent events and (B) is not equal to zero.

**P (A | B):** is the **conditional probability of an event A occurring given that B is true. P (B | A):** is the **conditional probability of an event B occurring given that A is true.**

**P (A) and P (B):** are the probabilities of A and B occurring **independently of one another (the marginal probability).**

The **Naive Bayes algorithm** offers **plenty of advantages to its users**. That’s why it has a lot of applications in various **industries, including Health, Technology, Environment**, etc.

#### Implementation of the Naïve Bayes involves below steps:

1. **Install the Packages:**
   1. **Numpy:** Numpy Python library is used for including any type **of mathematical operation in the code**. It is the fundamental package for scientific calculation in Python. It also supports to **add large, multidimensional arrays and matrices**. **So, in Python, we can import it as:**

###### import numpy as np

* 1. **Matplotlib:** The second library is matplotlib, which is a **Python 2D plotting library**, and with this library, we need to import a sub-library pyplot. This library is **used to plot any type of charts in Python** for the code. **It will be imported as below:**

###### import matplotlib.pyplot as plt

Here we have used **plt a**s a short name for this library.

* 1. **Pandas:** The last library is the Pandas library, which is one of the most famous Python libraries and used for **importing and managing the datasets**. It is an **open-source data manipulation and analysis library**. It will be imported as below:

###### import pandas as pd

Here, we have used **pd** as a short name for this library.

* 1. **Seaborn:** Seaborn is a Python **data visualization library** based on **matplotlib.** It provides a **high- level interface for drawing attractive and informative statistical graphics**.

###### import seaborn as sns

Here, we have used **sns** as a short name for this library.

#### Importing the Dataset:

**read\_csv() function:** Now to import the dataset, we will use **read\_csv() function** of **pandas library**, which is used to read a csv file and performs various operations on it. Using this function, we can read a csv file locally as well as through an URL. **We can use read\_csv function as below:**

**For Eg: dataset = pd.read\_csv ('NaiveBayes.csv')**

#### Separating Independent and Dependent Variables:

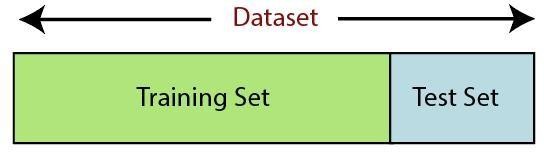
In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset.

**For Eg:** In our dataset, there are **Two independent variables** that are **Age** and **Salary**, and **one is a dependent variable** which is **purchased**.

###### # Seperating Independent and Dependent Variable

**x = dataset.iloc[:, [0,1]].values y = dataset.iloc[:, 2].values**

1. **Splitting dataset into training and test set:** we divide our dataset into a training set and test set. This is one of the crucial steps of data pre-processing as by doing this, we can enhance the performance of our machine learning model.



**Training Set:** A subset of dataset to train the machine learning model, and we already know the output. **Test set:** A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

**For splitting the dataset, we will use the below lines of code:**

**# training and testing data**

**from sklearn.model\_selection import train\_test\_split # assign test data size 25%**

**X\_train, X\_test, y\_train, y\_test =train\_test\_split(X,y,test\_size= 0.25, random\_state=0) Explanation:** We set test\_size=0.25, which means **25%** of the whole data set will be assigned to the **testing** part, and the remaining **75%** will be used for the model’s **training**.

1. **Training model using Naive Bayes Classifier:** Now, let’s train our model using the

Gaussian Naive Bayes classifier (a type of Naive Bayes Classifier).

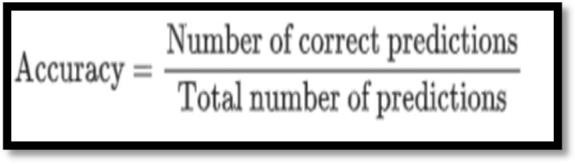
**# import Gaussian Naive Bayes classifier from sklearn.naive\_bayes import GaussianNB # create a Gaussian Classifier**

**classifer1 = GaussianNB() # training the model**

**classifer1.fit(X\_train, y\_train) # testing the model**

**y\_pred1 = classifer1.predict(X\_test)**

1. **Find the Accuracy of the Model:** Accuracy score in machine learning is an **evaluation metric** that **measures the number of correct predictions made by a model** in relation to **the total number of predictions made.** We calculate it by dividing the number of correct predictions by the total number of predictions.



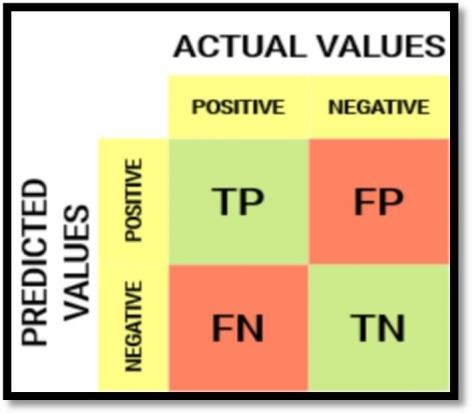
###### # importing accuracy score

**from sklearn.metrics import accuracy\_score # printing the accuracy of the model print(accuracy\_score(y\_test, y\_pred1))**

**Print the Confusion Matrix:** The **confusion matrix** is one of the most popular and widely used

performance measurement techniques for **classification models.**

**Confusion Matrix** as the name suggests gives us a **matrix as output and describes the complete performance of the model** and it also used to **determine the performance of the classification models for a given set of test data.**

****

**# importing the required modules import seaborn as sns**

**from sklearn.metrics import confusion\_matrix # passing actual and predicted values**

**cm = confusion\_matrix(y\_test, y\_pred1)**

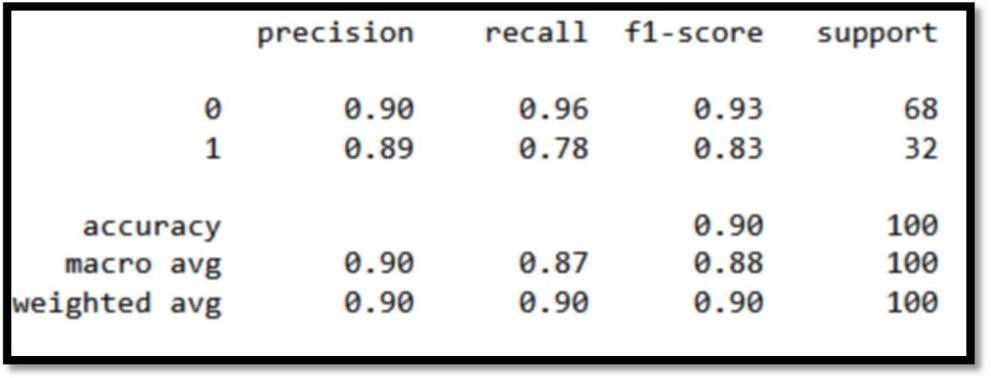
**# true write data values in each cell of the matrix sns.heatmap(cm,annot=True) plt.savefig('confusion.png')**

**Explanation: Heat Maps** are most commonly used to **display a more generalized view of numeric values as a graphical representation of data** where the individual values contained in a m**atrix are represented as colors.**

#### Print the Classification Report:

**# importing classification report**

**from sklearn.metrics import classification\_report # printing the report print(classification\_report(y\_test, y\_pred1))**

****

### PROGRAM:

###### # importing the libraries

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

import seaborn as sns

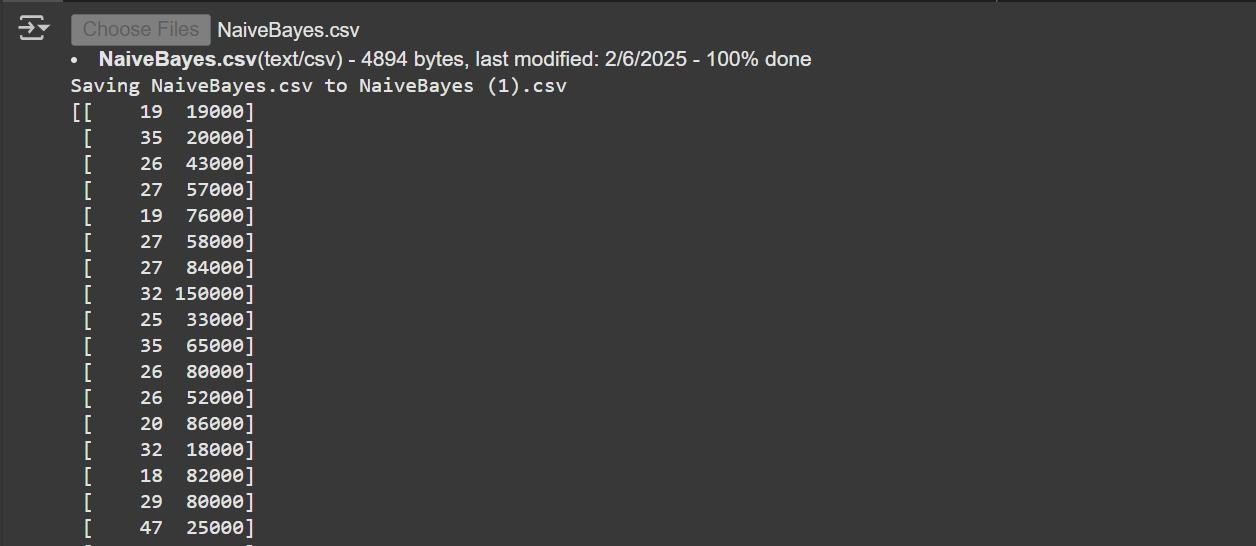
###### # importing the dataset

dataset = pd.read\_csv('NaiveBayes.csv')

###### # Seperating Independent and Dependent Variable

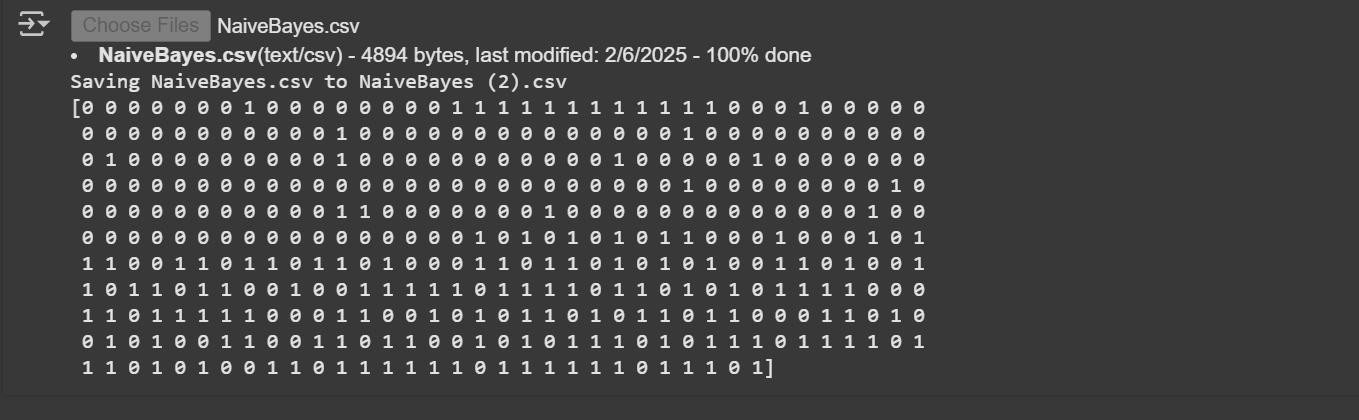
x = dataset.iloc[:, [0,1]].values y = dataset.iloc[:, 2].values print(x)

**Output:**

****

print(y)

**Output:**

****

###### # training and testing data

from sklearn.model\_selection import train\_test\_split

###### # assign test data size 25%

X\_train, X\_test, y\_train, y\_test =train\_test\_split(X,y,test\_size= 0.25, random\_state=0)

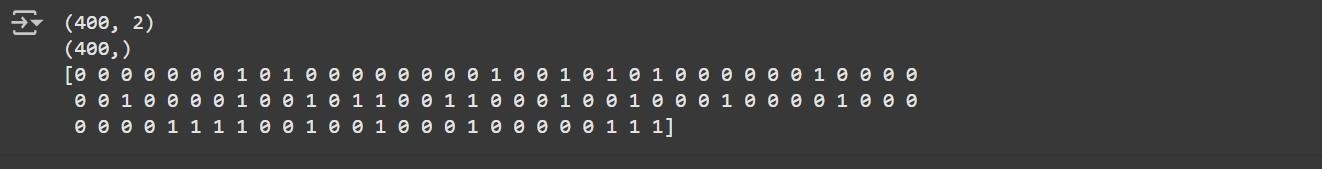
**# import Gaussian Naive Bayes classifier** from sklearn.naive\_bayes import GaussianNB **# create a Gaussian Classifier**

classifer1 = GaussianNB()

**# training the model** classifer1.fit(X\_train, y\_train) **# testing the model**

y\_pred1 = classifer1.predict(X\_test) print(y\_pred1)

**Output:**

****

###### # importing accuracy score

from sklearn.metrics import accuracy\_score **# printing the accuracy of the model** print(accuracy\_score(y\_test,y\_pred1)) **Output:**



###### # importing the required modules

import seaborn as sns

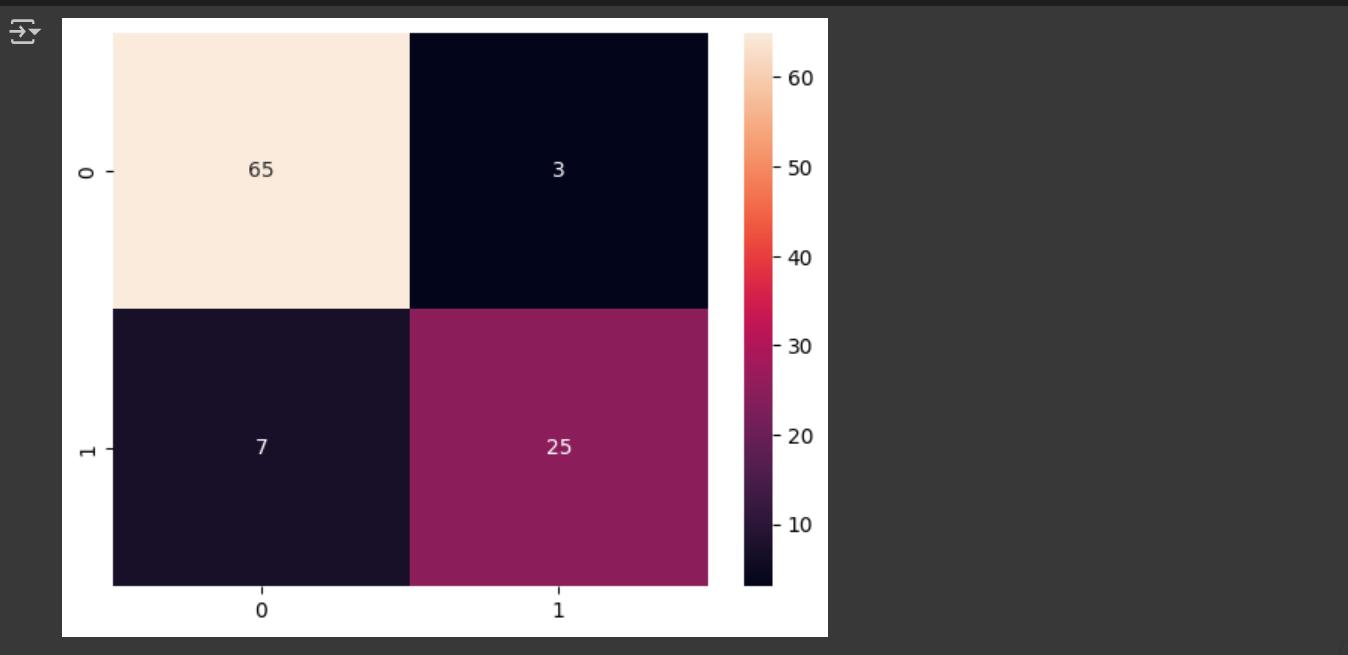
from sklearn.metrics import confusion\_matrix

###### # passing actual and predicted values

cm = confusion\_matrix(y\_test, y\_pred1)

**# true write data values in each cell of the matrix** sns.heatmap(cm,annot=True) plt.savefig('confusion.png')

**Output:**

****

###### # importing classification report

from sklearn.metrics import classification\_report **# printing the report** print(classification\_report(y\_test, y\_pred1)) **Output:**

