DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**57**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**01**
DOP	
DOS	

Experiment No. 01

AIM : Introduction to platforms such as Anaconda, COLAB

Theory :

Anaconda :

Anaconda is an open-source package manager and framework for handling machine learning and data science workflows. It also helps to distribute some programming languages like Python and R. With over 7500 different scientific data packages, Anaconda helps process large-scale data, scientific computing, and predictive analysis.

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012. As an Anaconda, Inc. product, it is also known as Anaconda Distribution or Anaconda Individual Edition, while other products from the company are Anaconda Team Edition and Anaconda Enterprise Edition, both of which are not free.

Package versions in Anaconda are managed by the package management system conda. This package manager was spun out as a separate open-source package as it ended up being useful on its own and for things other than Python. There is also a small, bootstrap version of Anaconda called Miniconda, which includes only conda, Python, the packages they depend on, and a small number of other packages.

Steps for installation anaconda on ubuntu :

Step 1 – Download and Install Anaconda Script

Once you're logged into your VPS, refresh the APT command to synchronize all repositories via the command line:

sudo apt-get update

Move to the /tmp directory:

cd /tmp

Download the newest Anaconda installer with the wget command. If you don't have it, install it first:

apt-get install wget

Download the Anaconda installer:

wget https://repo.anaconda.com/archive/Anaconda3-2022.05-Linux-x86_64.sh

Once the download is complete, verify the hash code integrity of the package:

sha256sum Anaconda3-2022.05-Linux-x86_64.sh

The output will notify if any errors occurred. If there are no errors, move on to the actual installation step. To continue, run the Anaconda bash shell script:

bash Anaconda3-2022.05-Linux-x86_64.sh

If you want to install Miniconda instead, use the following commands consecutively:

wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

sha256sum Miniconda3-latest-Linux-x86_64.sh

bash Miniconda3-latest-Linux-x86_64

After running the bash command, you'll be welcomed to the Anaconda setup. However, you must review and agree to its license agreement before the installation. Hit Enter to continue.

Pressing the spacebar a few times will bring you to the end of the license agreement, where you can accept the terms. Type in "yes" as highlighted and hit Enter.

Step 2 – Choose Installation Directory

After agreeing to the license terms, the following prompt will ask you to input the directory where to install the Anaconda on the Ubuntu system. The default location is the user's HOME directory on Ubuntu.

It is recommended to have Anaconda installed in this location. Therefore, press Enter to confirm the default location.

In the next prompt, you will see that the installation process has started. Wait a few minutes until the installer successfully completes the installation process. Type "yes" once more and press Enter.

Congratulations, you have successfully installed Anaconda!

Step 3 – Test the Connection

With the installation done, the next step is to activate the added environment settings using the following command:

source ~/.bashrc

Then, test out the connection:

conda info

If the installation process was successful, a piece of similar-looking information should be displayed:

Updating Anaconda

In case you ever need to update Anaconda, start by updating the conda package manager first:

conda update conda

Then, update the actual Anaconda distribution:

conda update anaconda

Wait a few minutes until the installer successfully completes the Anaconda installation process, type "y" and press Enter.

Uninstalling Anaconda

In order to uninstall Anaconda, install the following anaconda-clean package:

conda install anaconda-clean

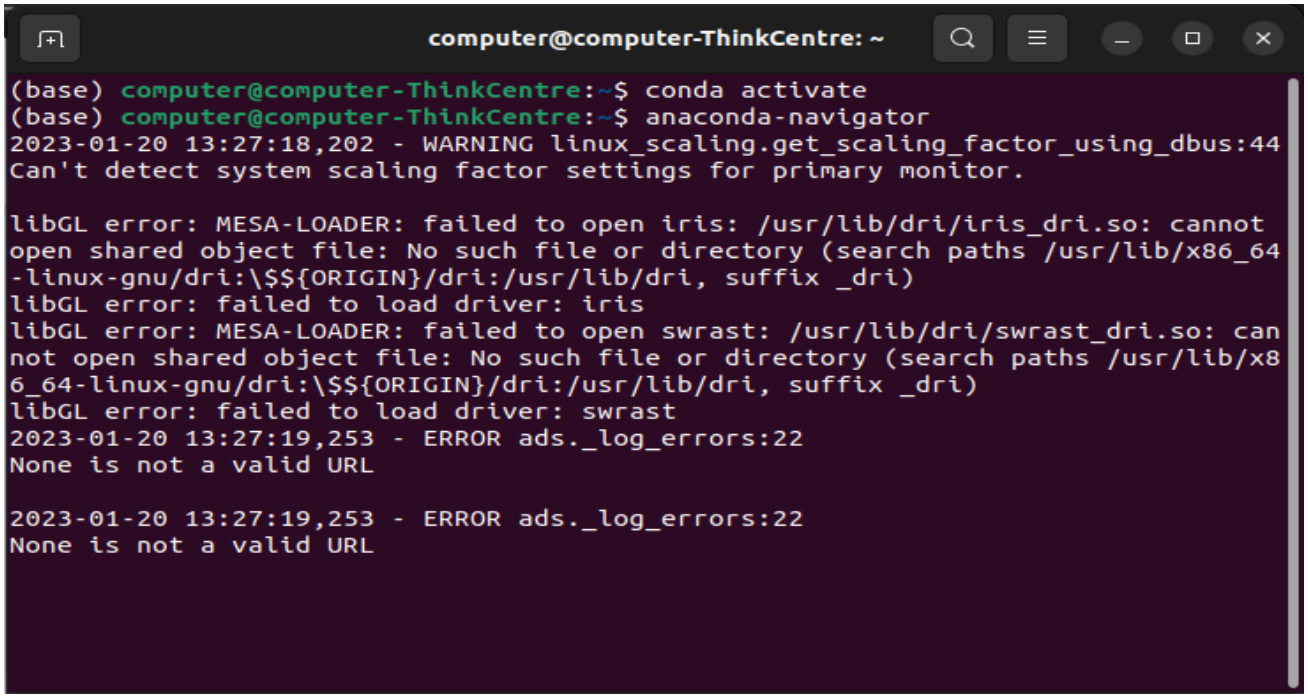
Lastly, remove all Anaconda-related files and directories:

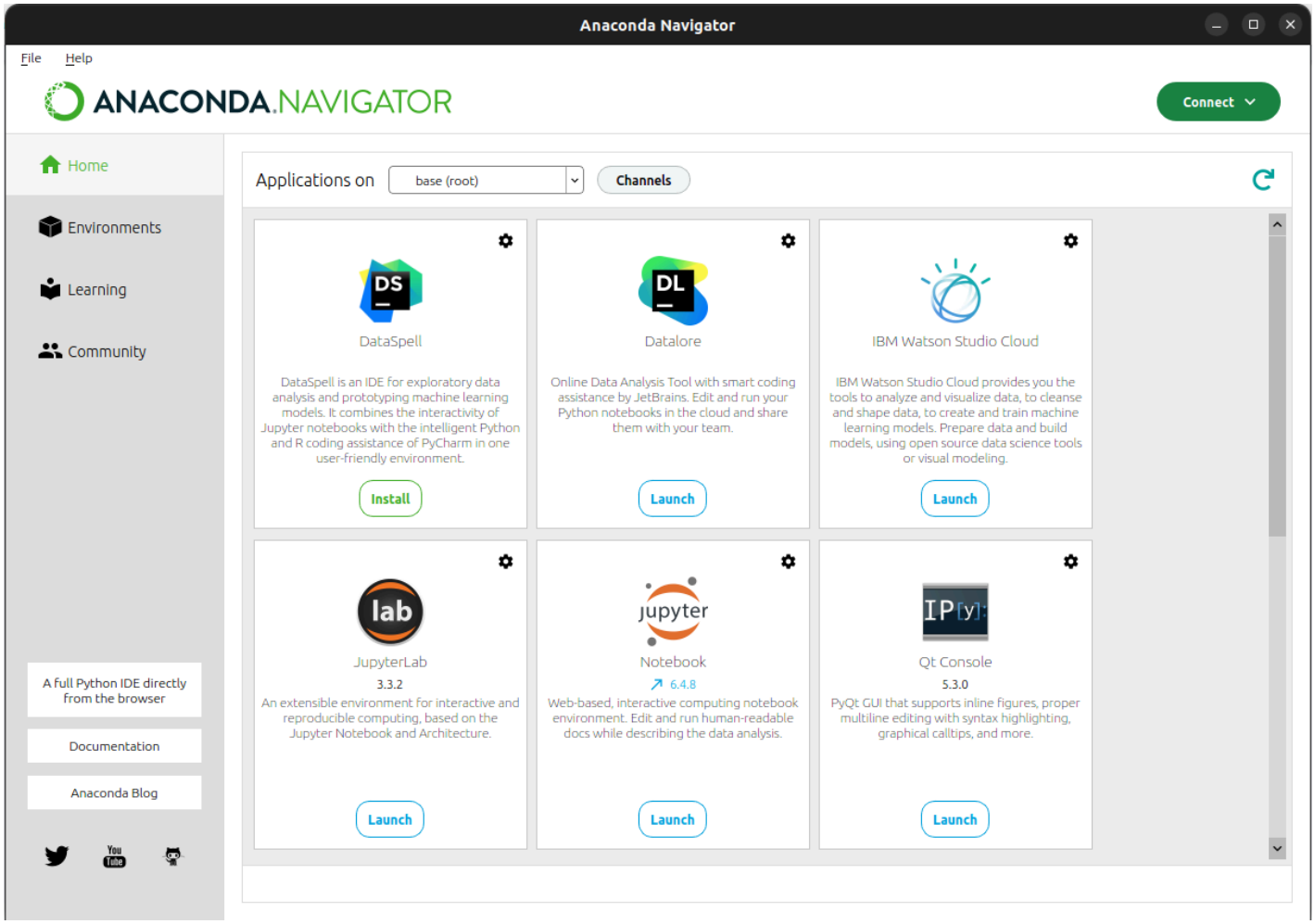
anaconda-clean

Step 4 – Activating Anaconda

conda activate

anaconda-navigator


computer@computer-ThinkCentre: ~
(base) computer@computer-ThinkCentre:~$ conda activate
(base) computer@computer-ThinkCentre:~$ anaconda-navigator
2023-01-20 13:27:18,202 - WARNING linux_scaling.get_scaling_factor_using_dbus:44
Can't detect system scaling factor settings for primary monitor.
libGL error: MESA-LOADER: failed to open iris: /usr/lib/dri/iris_dri.so: cannot open shared object file: No such file or directory (search paths /usr/lib/x86_64-linux-gnu/dri:\$${ORIGIN}/dri:/usr/lib/dri, suffix _dri)
libGL error: failed to load driver: iris
libGL error: MESA-LOADER: failed to open swrast: /usr/lib/dri/swrast_dri.so: cannot open shared object file: No such file or directory (search paths /usr/lib/x86_64-linux-gnu/dri:\$${ORIGIN}/dri:/usr/lib/dri, suffix _dri)
libGL error: failed to load driver: swrast
2023-01-20 13:27:19,253 - ERROR ads._log_errors:22
None is not a valid URL
2023-01-20 13:27:19,253 - ERROR ads._log_errors:22
None is not a valid URL


Anaconda Navigator
File
Help
ANACONDA.NAVIGATOR
Connect
Home
Environments
Learning
Community
Applications on
base (root)
Channels
DataSpell
DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.
Install
Datalore
Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.
Launch
IBM Watson Studio Cloud
IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.
Launch
JupyterLab
3.3.2
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch
Notebook
6.4.8
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
Qt Console
5.3.0
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch
A full Python IDE directly from the browser
Documentation
Anaconda Blog

Jupyter Notebook :

Introduction :

JupyterLab is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design invites extensions to expand and enrich functionality.

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.

Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

Features of Jupyter Notebook :

- **All in one place:** As you know, Jupyter Notebook is an open-source web-based interactive environment that combines code, text, images, videos, mathematical equations, plots, maps, graphical user interface and widgets to a single document.
- **Interactive code:** Jupyter notebook uses ipywidgets packages, which provide many common user interfaces for exploring code and data interactivity.
- **Compiled with Latest Python release:** Anaconda 5.3 is compiled with Python 3.7, taking advantage of Python's speed and feature improvements.
- **Better Reliability:** The reliability of Anaconda has been improved in the latest release by capturing and storing the package metadata for installed packages.
- **Enhanced CPU Peformance:** The Intel Math Kernel Library 2019 for Deep Neural Networks(MKL 2019) has been introduced in Anaconda 5.3 distribution. Users deploying Tensorflow can make use of MKL 2019 for Deep Neural Networks. These Python binary packages are provided to achieve high CPU performance.
- **New packages are added:** There are over 230 packages which has been updated and added in the new release.

Program using Jupyter Notebook :

```
In [1]: num_1=int(input("enter first number: "))
        num_2=int(input("enter second number: "))
        print("your sum is : ",num_1+num_2)

        enter first number: 12
        enter second number: 34
        your sum is :  46
```

COLAB :

Introduction :

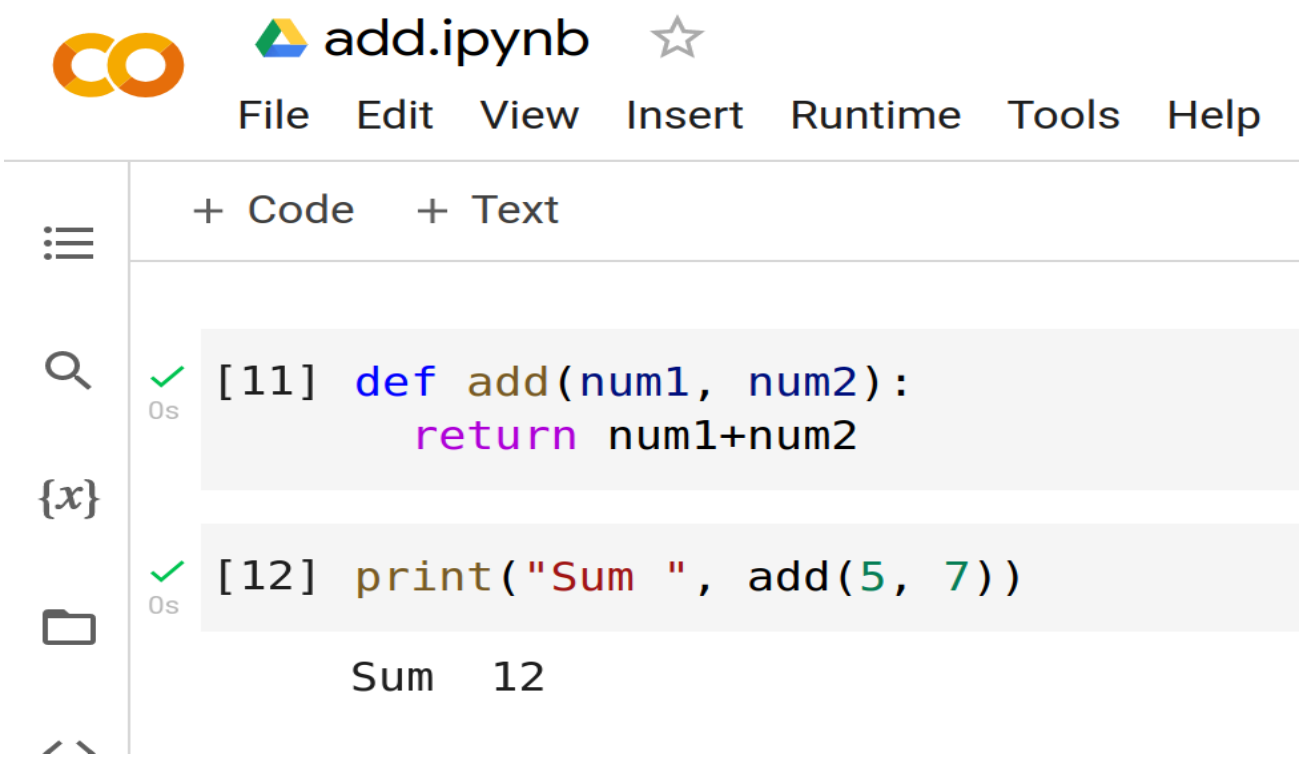
Google Colaboratory, or Colab, is a cloud-based environment for writing documents with live code, visualizations, and narrative text. For those who are familiar with Jupyter notebooks, Colab notebooks are the same, including the .ipynb extension. Unlike Jupyter and Atom (our previous editor for code and reports), however, Colab requires no setup on your computer! It also provides a large amount of free computing power and easy document sharing.

Google is quite aggressive in AI research. Over many years, Google developed an AI framework called TensorFlow and a development tool called Colaboratory. Today TensorFlow is open-sourced and since 2017, Google made Colaboratory free for public use. Colaboratory is now known as Google Colab or simply Colab.

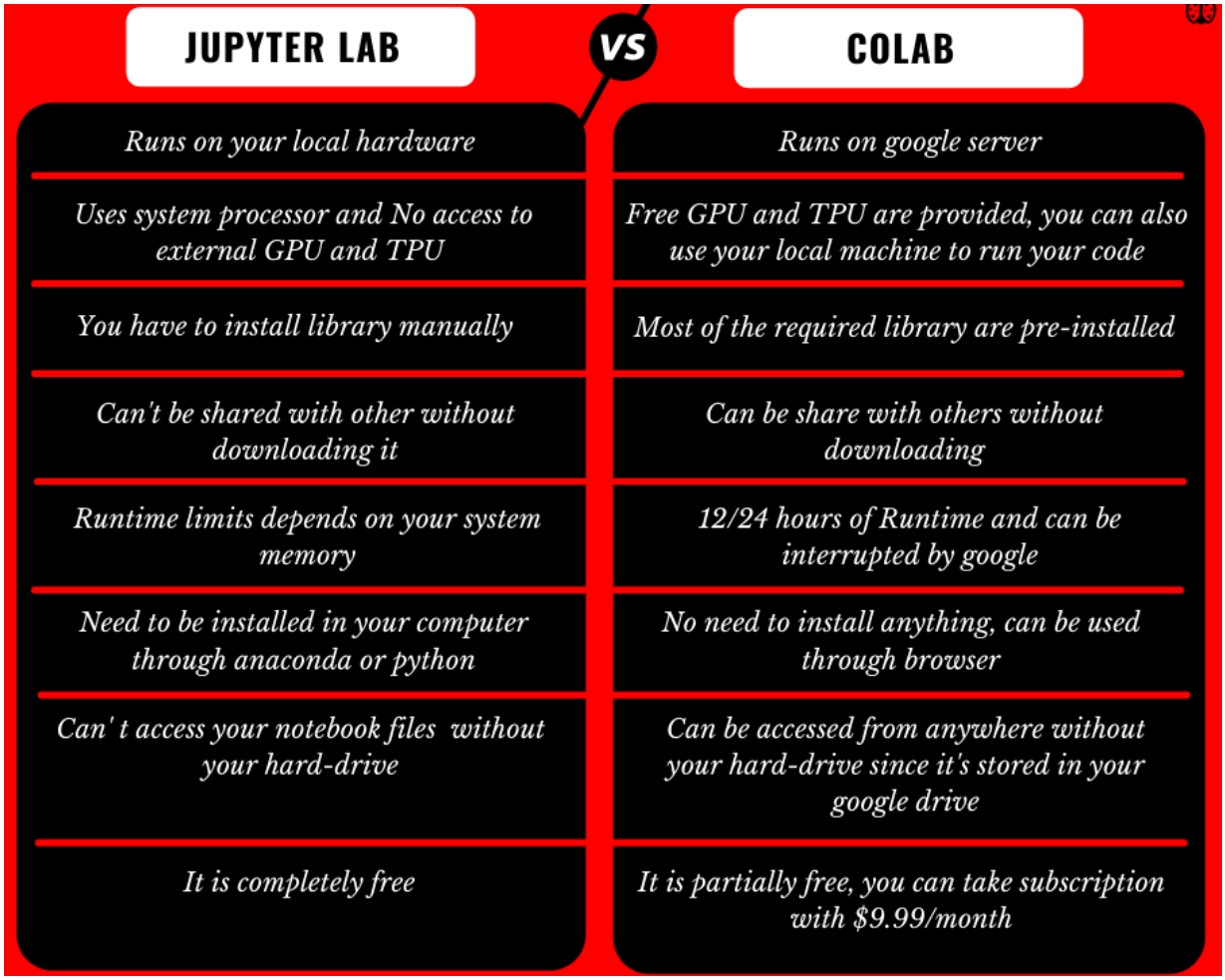
Another attractive feature that Google offers to the developers is the use of GPU. Colab supports GPU and it is totally free. The reasons for making it free for the public could be to make its software a standard in the academics for teaching machine learning and data science. It may also have a long term perspective of building a customer base for Google Cloud APIs which are sold on a per-use basis.

Irrespective of the reasons, the introduction of Colab has eased the learning and development of machine learning applications.

Program using COLAB :



Difference between Jupyter and COLAB :



JUPYTER LAB	VS	COLAB
Runs on your local hardware		*Runs on google server*
Uses system processor and No access to external GPU and TPU		*Free GPU and TPU are provided, you can also use your local machine to run your code*
You have to install library manually		*Most of the required library are pre-installed*
Can't be shared with other without downloading it		*Can be share with others without downloading*
Runtime limits depends on your system memory		*12/24 hours of Runtime and can be interrupted by google*
Need to be installed in your computer through anaconda or python		*No need to install anything, can be used through browser*
Can' t access your notebook files without your hard-drive		*Can be accessed from anywhere without your hard-drive since it's stored in your google drive*
It is completely free		*It is partially free, you can take subscription with $9.99/month*

Conclusion : Learned about different platforms such as Anaconda, COLAB.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**57**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL603**
PRACTICAL NO.	**02**
DOP	**25/01/2023**
DOS	

NUMPY:

NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow use NumPy internally for manipulation of Tensors.

PROGRAM:

```
# operations
import numpy as np
# Creating two arrays of rank 2
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])
# Creating two arrays of rank 1
v = np.array([9, 10])
w = np.array([11, 12])
# Inner product of vectors
print(np.dot(v, w), "\n")
# Matrix and Vector product
print(np.dot(x, v), "\n")
# Matrix and matrix product
print(np.dot(x, y))
```

OUTPUT:

In [1]:
```
# Python program using NumPy

import numpy as np

# Creating two arrays of rank 2
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])

# Creating two arrays of rank 1
v = np.array([9, 10])
w = np.array([11, 12])

# Inner product of vectors
print(np.dot(v, w), "\n")

# Matrix and Vector product
print(np.dot(x, v), "\n")

# Matrix and matrix product
print(np.dot(x, y))
```

```
219

[29 67]

[[19 22]
 [43 50]]
```

In [ ]:

SCIPY:

SciPy is a very popular library among Machine Learning enthusiasts as it contains different modules for optimization, linear algebra, integration and statistics. There is a difference between the SciPy library and the SciPy stack. The SciPy is one of the core packages that make up the SciPy stack. SciPy is also very useful for image manipulation.

PROGRAM:

```
from scipy import io
import numpy as np
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,])
#Export:
io.savemat('arr.mat', {"vec": arr})
#Import:
mydata = io.loadmat('arr.mat')
print(mydata)
```

OUTPUT:

```
from scipy import io
import numpy as np

arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,])

#Export:
io.savemat('arr.mat', {"vec": arr})

#Import:
mydata = io.loadmat('arr.mat')

print(mydata)
```

```
{
  '__header__': b'MATLAB 5.0 MAT-file Platform: nt, Created on: Tue Sep 22 13:12:32 2020',
  '__version__': '1.0',
  '__globals__': [],
  'vec': array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
}
```

TENSORFLOW:

TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, Tensorflow is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application.

PROGRAM:

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
<body>
<h1>TensorFlow JavaScript</h1>
<h3>Get the data behind a tensor:</h3>
<div id="demo"></div>
<script>
const myArr = [[1, 2], [3, 4]]; const tensorA = tf.tensor(myArr);
tensorA.data().then(data => display(data));
// Result: 1,2,3,4 function display(data) {
        document.getElementById("demo").innerHTML = data;
}
</script>
</body>
</html>
```

OUTPUT:

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
<body>

<h1>TensorFlow JavaScript</h1>
<h3>Get the data behind a tensor:</h3>

<div id="demo"></div>

<script>
const myArr = [[1, 2], [3, 4]];
const tensorA = tf.tensor(myArr);
tensorA.data().then(data => display(data));

// Result: 1,2,3,4
function display(data) {
  document.getElementById("demo").innerHTML = data;
}
</script>
</body>
</html>
```

TensorFlow JavaScript

Get the data behind a tensor:

1,2,3,4

PANDAS:

Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and a wide variety of tools for data analysis. It provides many inbuilt methods for grouping, combining and filtering data.

PROGRAM:

```
# arranging a given set of data
# into a table
# importing pandas as pd
import pandas as pd
data = {"country": ["Brazil", "Russia", "India", "China", "South Africa"],
        "capital": ["Brasilia", "Moscow", "New Delhi", "Beijing", "Pretoria"],
        "area": [8.516, 17.10, 3.286, 9.597, 1.221],
        "population": [200.4, 143.5, 1252, 1357, 52.98] }
data_table = pd.DataFrame(data)
print(data_table)
```

OUTPUT:

In [4]:
```
# Python program using Pandas for
# arranging a given set of data
# into a table

# importing pandas as pd
import pandas as pd

data = {"country": ["Brazil", "Russia", "India", "China", "South Africa"],
"capital": ["Brasilia", "Moscow", "New Delhi", "Beijing", "Pretoria"],
"area": [8.516, 17.10, 3.286, 9.597, 1.221],
"population": [200.4, 143.5, 1252, 1357, 52.98] }

data_table = pd.DataFrame(data)
print(data_table)
```

```
        country    capital    area  population
0        Brazil   Brasilia   8.516      200.40
1        Russia     Moscow  17.100      143.50
2         India  New Delhi   3.286     1252.00
3         China    Beijing   9.597     1357.00
4  South Africa   Pretoria   1.221       52.98
```

In [ ]:

MATPLOTLIB:

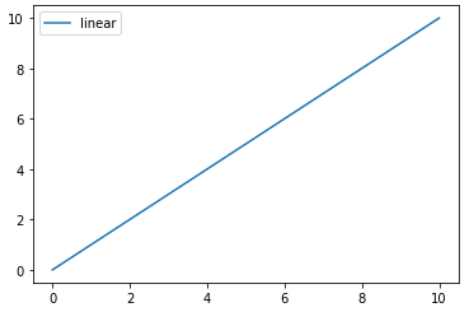
Matplotlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar charts, etc.

PROGRAM:

```
# for forming a linear plot
import matplotlib.pyplot as plt
import numpy as np
# Prepare the data
x = np.linspace(0, 10, 100)
# Plot the data
plt.plot(x, x, label ='linear')
# Add a legend
plt.legend()
# Show the plot
plt.show()
```

OUTPUT:

In [5]:
```
# Python program using Matplotlib
# for forming a linear plot

import matplotlib.pyplot as plt
import numpy as np

# Prepare the data
x = np.linspace(0, 10, 100)

# Plot the data
plt.plot(x, x, label ='linear')

# Add a legend
plt.legend()

# Show the plot
plt.show()
```



In [ ]:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**57**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**03**
DOP	**01/02/2023**
DOS	

Program(input)/Output :

- Import Libraries

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt

        %matplotlib inline
```

- Importing Data and Checking out

data_frame.head()

```
In [2]: HouseDF = pd.read_csv('USA_Housing.csv')
```

```
In [3]: HouseDF.head()
```

Out[3]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Addres
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Ap 674\nLaurabury, N 3701.
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson View Suite 079\nLak Kathleen, CA.
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabet Stravenue\nDanieltowr WI 06482.
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO A 4482
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFP(AE 0938

data_frame.info()

```
In [4]: HouseDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

data_frame.describe()

```
In [5]: HouseDF.describe()
```

Out[5]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

data_frame.columns

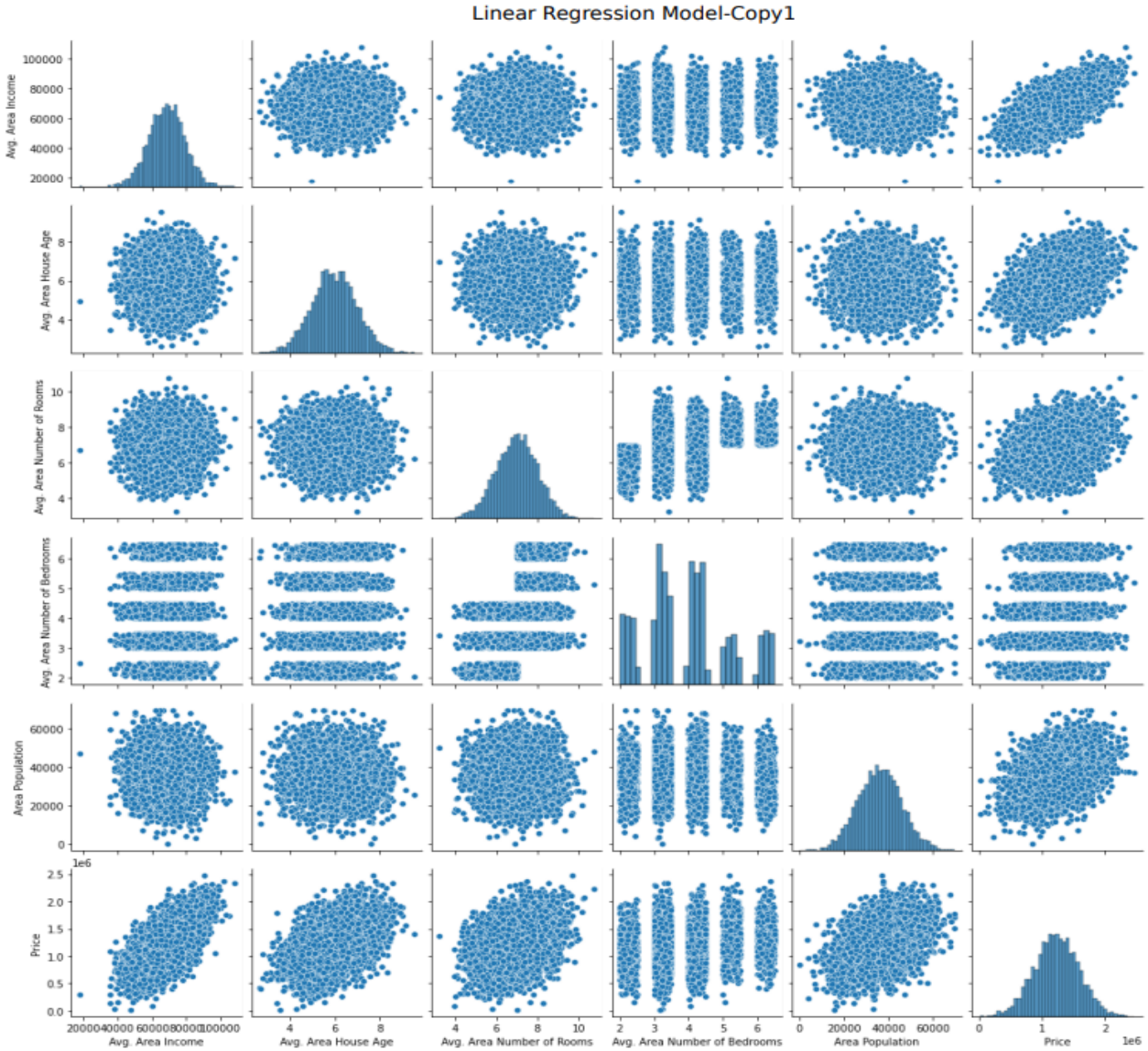
```
In [6]: HouseDF.columns
```

```
Out[6]: Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

- Exploratory Data Analysis for House Price Prediction

sns.pairplot(data_frame)

```
In [7]: sns.pairplot(HouseDF)
Out[7]: <seaborn.axisgrid.PairGrid at 0x7fb14ff35e80>
```

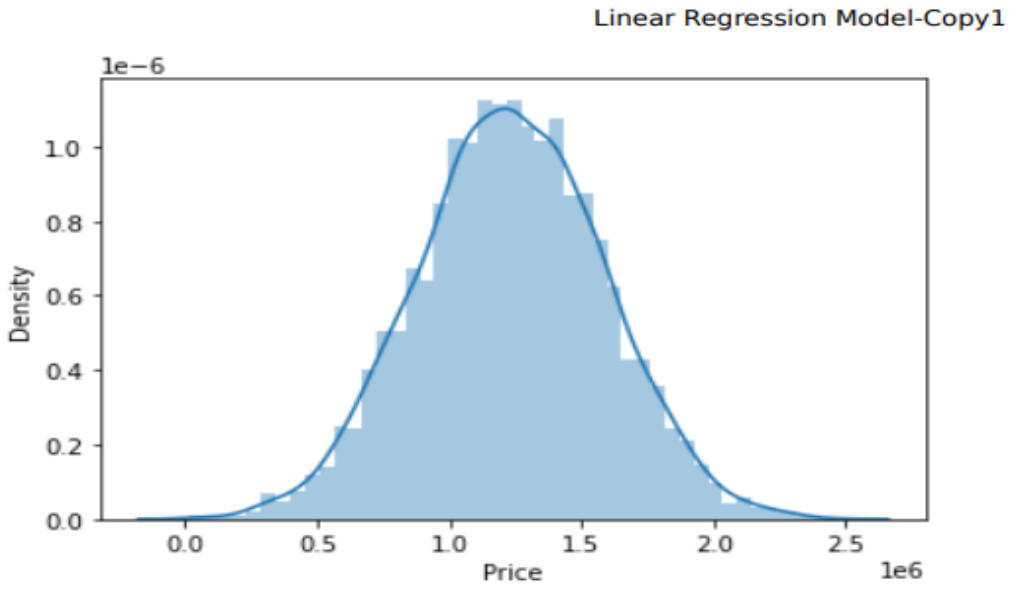


sns.distplot(data_frame[ ])

```
In [8]: sns.distplot(HouseDF['Price'])
```

```
/home/computer/anaconda3/lib/python3.9/site-packages/seaborn/distributions.
py:2619: FutureWarning: `distplot` is a deprecated function and will be rem
oved in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-leve
l function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
Out[8]: <AxesSubplot:xlabel='Price', ylabel='Density'>
```



sns.heatmap(data_frame.corr(),annot=True)

```
In [9]: sns.heatmap(HouseDF.corr(), annot=True)
```

```
Out[9]: <AxesSubplot:>
```

- Training a Linear Regression Model

X and y List

```
In [10]: X = HouseDF[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number o
                      'Avg. Area Number of Bedrooms', 'Area Population']]

         y = HouseDF['Price']
```

Split Data into Train, Test

```
In [11]: from sklearn.model_selection import train_test_split
```

```
In [12]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, ra
```

- Creating and Training the LinearRegression Model

```
In [13]: from sklearn.linear_model import LinearRegression
```

```
In [14]: lm = LinearRegression()
```

```
In [15]: lm.fit(X_train,y_train)
```

```
Out[15]: LinearRegression()
```

- LinearRegression Model Evaluation

```
In [16]: print(lm.intercept_)
```

```
-2640159.79685191
```

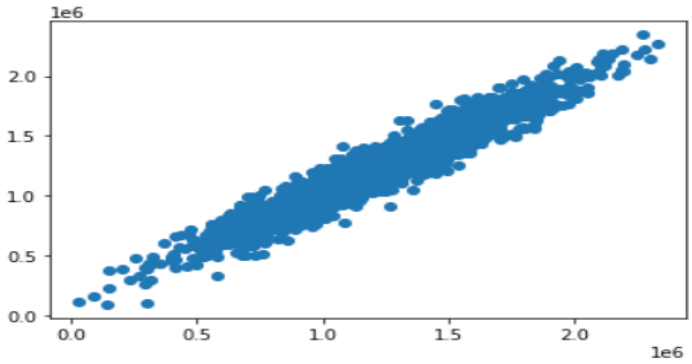
```
In [17]: coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
         coeff_df
```

Out[17]:

	Coefficient
Avg. Area Income	21.528276
Avg. Area House Age	164883.282027
Avg. Area Number of Rooms	122368.678027
Avg. Area Number of Bedrooms	2233.801864
Area Population	15.150420

- Predictions from our Linear Regression Model

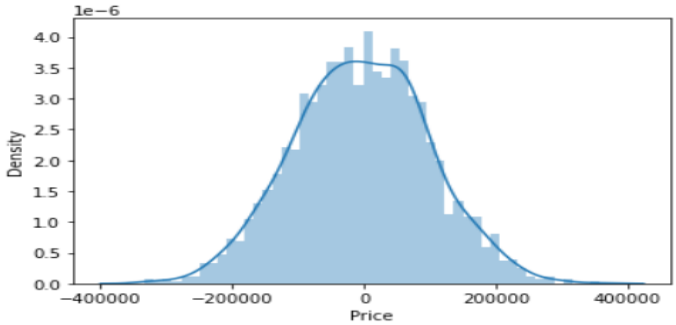
```
In [18]: predictions = lm.predict(X_test)

In [19]: plt.scatter(y_test,predictions)

Out[19]: <matplotlib.collections.PathCollection at 0x7fb14c0c6d00>
```



In the above scatter plot, we see data is in line shape, which means our model has done good predictions.

sns.distplot((y_test-predictions),bins=50)

```
In [20]: sns.distplot((y_test-predictions),bins=50);

/home/computer/anaconda3/lib/python3.9/site-packages/seaborn/distributions.
py:2619: FutureWarning: `distplot` is a deprecated function and will be rem
oved in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-leve
l function for histograms).
  warnings.warn(msg, FutureWarning)
```



In the above histogram plot, we see data is in bell shape (Normally Distributed), which means our model has done good predictions.

- Regression Evaluation Metrics

```
In [21]: from sklearn import metrics

In [22]: print('MAE:', metrics.mean_absolute_error(y_test, predictions))
         print('MSE:', metrics.mean_squared_error(y_test, predictions))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))

         MAE: 82288.22251914954
         MSE: 10460958907.209501
         RMSE: 102278.82922291153

In [ ]:
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**57**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**04**
DOP	**22/02/2023**
DOS	

Program(input)/Output :

- Import Libraries

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from seaborn import load_dataset
%matplotlib inline
plt.style.use('ggplot')
```

- Importing Data and Checking out

data_frame.head()

In [3]:

```
data = load_dataset("titanic")
data
```

Out[3]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_m
0	0	3	male	22.0	1	0	7.2500	S	Third	man	T
1	1	1	female	38.0	1	0	71.2833	C	First	woman	Fa
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	Fa
3	1	1	female	35.0	1	0	53.1000	S	First	woman	Fa
4	0	3	male	35.0	0	0	8.0500	S	Third	man	T
...	...	...	...	...	...	...	...	...	...	...	
886	0	2	male	27.0	0	0	13.0000	S	Second	man	T
887	1	1	female	19.0	0	0	30.0000	S	First	woman	Fa
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	Fa
889	1	1	male	26.0	0	0	30.0000	C	First	man	T
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	T

891 rows × 15 columns

data_frame.info()

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

data_frame.columns
data_frame.describe()

In [5]:

```
columns = ['alive', 'alone', 'embark_town', 'who', 'adult_male', 'deck']
data_2 = data.drop(columns, axis=1)
```

In [6]:

```
data_2.describe(include='all').T
```

Out[6]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	m
survived	891.0	NaN	NaN	NaN	0.383838	0.486592	0.0	0.0	0.0	1.0	
pclass	891.0	NaN	NaN	NaN	2.308642	0.836071	1.0	2.0	3.0	3.0	
sex	891	2	male	577	NaN	NaN	NaN	NaN	NaN	NaN	N
age	714.0	NaN	NaN	NaN	29.699118	14.526497	0.42	20.125	28.0	38.0	8
sibsp	891.0	NaN	NaN	NaN	0.523008	1.102743	0.0	0.0	0.0	1.0	
parch	891.0	NaN	NaN	NaN	0.381594	0.806057	0.0	0.0	0.0	0.0	
fare	891.0	NaN	NaN	NaN	32.204208	49.693429	0.0	7.9104	14.4542	31.0	512.3
embarked	889	3	S	644	NaN	NaN	NaN	NaN	NaN	NaN	N
class	891	3	Third	491	NaN	NaN	NaN	NaN	NaN	NaN	N

- max and min values :

In [7]:

```
print(f"Max value of age column : {data_2['age'].max()}")
print(f"Min value of age column : {data_2['age'].min()}")
```

```
Max value of age column : 80.0
Min value of age column : 0.42
```

- data_set_2 :

In [8]:

```
bins = [0, 5, 17, 25, 50, 80]
labels = ['Infant', 'Kid', 'Young', 'Adult', 'Old']
data_2['age'] = pd.cut(data_2['age'], bins = bins, labels=labels)
```

In [9]:

```
pd.DataFrame(data_2['age'].value_counts())
```

Out[9]:

	age
Adult	349
Young	188
Kid	69
Old	64
Infant	44

data_frame.mode()[ ]
data_frame.fillna()
data_frame[ ].unique()

In [12]:

```
data_2['age'].mode()[0]
```

Out[12]:

```
'Adult'
```

In [14]:

```
data_4 = data_2.fillna({'age' : data_2['age'].mode()[0]})
```

In [15]:

```
data_2['embarked'].unique()
```

Out[15]:

```
array(['S', 'C', 'Q', nan], dtype=object)
```

In [16]:

```
print(f"How many 'S' on embarked column : {data_2[data_2['embarked'] == 'S'].shape[
print(f"How many 'C' on embarked column : {data_2[data_2['embarked'] == 'C'].shape[
print(f"How many 'Q' on embarked column : {data_2[data_2['embarked'] == 'Q'].shape[
```

```
How many 'S' on embarked column : 644
How many 'C' on embarked column : 168
How many 'Q' on embarked column : 77
```

- filling data_set_3 in data_set_2

In [17]:

```
data_3 = data_2.fillna({'embarked' : 'S'})
data_4[['pclass', 'survived']].groupby(['pclass']).sum().sort_values(by='survived')
```

Out[17]:

pclass	survived
2	87
3	119
1	136

In [18]:

```
data_4[['sex', 'survived']].groupby(['sex']).sum().sort_values(by='survived')
```

Out[18]:

sex	survived
male	109
female	233

In [29]:

```
bins = [-1, 7.9104, 14.4542, 31, 512.330]
labels = ['low','medium-low','medium','high']
data_4['fare']=pd.cut(data_3['fare'],bins = bins,labels = labels)
```

sns.distplot(data_frame[ ])

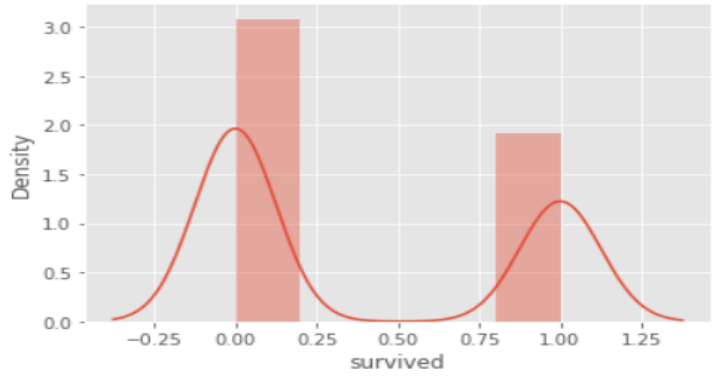
In [27]:

```
data_5 = data_4.drop('class', axis=1)
sns.distplot(data_5['survived'])
```

```
/home/computer/anaconda3/lib/python3.9/site-packages/seaborn/distribut
ions.py:2619: FutureWarning: `distplot` is a deprecated function and w
ill be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[27]:

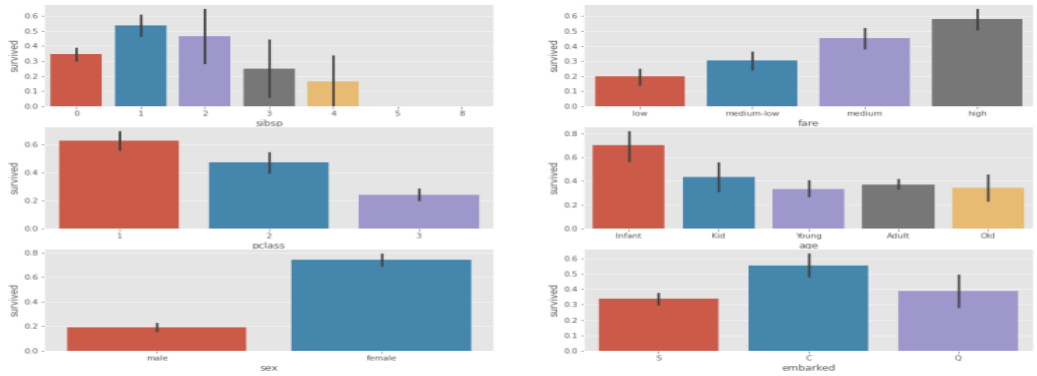
```
<AxesSubplot:xlabel='survived', ylabel='Density'>
```



plt.figure()
plt.subplot()
plt.barplot(x=‘ ’,y=‘ ’,data=data_set)

In [24]:

```
plt.figure(figsize=(20, 10))
plt.subplot(321)
sns.barplot(x = 'sibsp', y = 'survived', data = data_5)
plt.subplot(322)
sns.barplot(x = 'fare', y = 'survived', data = data_5)
plt.subplot(323)
sns.barplot(x = 'pclass', y = 'survived', data = data_5)
plt.subplot(324)
sns.barplot(x = 'age', y = 'survived', data = data_5)
plt.subplot(325)
sns.barplot(x = 'sex', y = 'survived', data = data_5)
plt.subplot(326)
sns.barplot(x = 'embarked', y = 'survived', data = data_5);
```



- creating data_set for dummies

data_frame.shape

In [30]:

```
dummies = ['fare', 'age', 'embarked', 'sex']
dummy_data = pd.get_dummies(data_5[dummies])
```

In [31]:

```
dummy_data.shape
```

Out[31]:

```
(891, 14)
```

- creating another data_set

In [32]:

```
data_6 = pd.concat([data_5, dummy_data], axis = 1)
data_6.drop(dummies, axis=1, inplace=True)
```

- importing libraries

In [33]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
```

data_frame.drop()

In [34]:

```
X = data_6.drop('survived', axis = 1)
y = data_6['survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_
```

- logistic regression

data_frame.fit()

data_frame.predict()

In [35]:

```
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
y_pred
```

Out[35]:

```
array([0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
1,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
0,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
0,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0,
1,
       0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
0,
       1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
1,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1,
       0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0,
1,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0,
0,
       0, 0, 0, 0, 0, 0, 1, 1, 1])
```

- calculating accuracy and confusion matrix

In [36]:

```
accuracy_score(y_pred, y_test)
```

Out[36]:

```
0.8067796610169492
```

In [37]:

```
confusion_matrix(y_pred, y_test)
```

Out[37]:

```
array([[158,  31],
       [ 26,  80]])
```

In [ ]:

```
# 31 + 26 = 57 wrong prediction
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**57**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**05**
DOP	
DOS	

Program(input)/Output :

- **Data Preprocessing Step**
  1. Importing libraries & datasets
  2. Extracting Independent and dependent variable
  3. Splitting the dataset into training and test set
  4. Feature Scaling

```
In [5]: #Data Pre-processing Step
        # importing libraries
        import numpy as nm
        import matplotlib.pyplot as mtp
        import pandas as pd

        #importing datasets
        data_set= pd.read_csv('/home/computer/Downloads/suv_data.csv')

        #Extracting Independent and dependent Variable
        x= data_set.iloc[:, [2,3]].values
        y= data_set.iloc[:, 4].values

        # Splitting the dataset into training 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
        #feature Scaling
        from sklearn.preprocessing import StandardScaler
        st_x= StandardScaler()
        x_train= st_x.fit_transform(x_train)
        x_test= st_x.transform(x_test)
```

The scaled output for the test set will be:

```
In [6]: data_set
```

Out[6]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

- **Fitting the SVM classifier to the training set**
  1. Support vector classifier

```
In [7]: from sklearn.svm import SVC # "Support vector classifier"
        classifier = SVC(kernel='linear', random_state=0)
        classifier.fit(x_train, y_train)
```

```
Out[7]: SVC(kernel='linear', random_state=0)
```

Below is the output for the prediction of the test set

```
In [9]: y_pred

Out[9]: array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 1,
               0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 0,
               1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 1,
               0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
        0, 1,
               0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1])
```

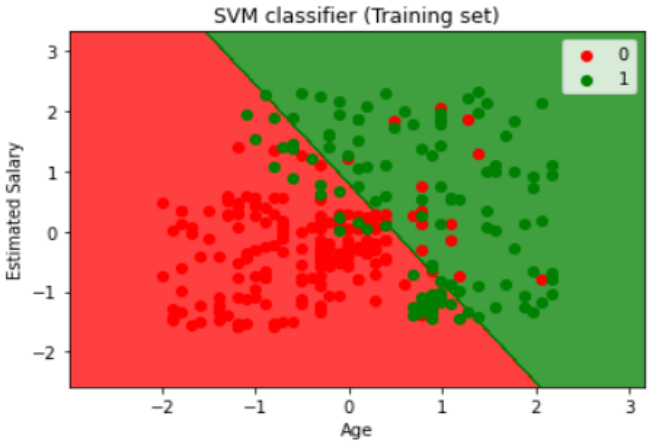
- **Creating the confusion matrix:**

```
In [10]: #Creating the Confusion matrix
         from sklearn.metrics import confusion_matrix
         cm= confusion_matrix(y_test, y_pred)

In [11]: cm

Out[11]: array([[66,  2],
                [ 8, 24]])
```

- **Visualizing the training set result:**

```
In [32]:
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() +
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
```

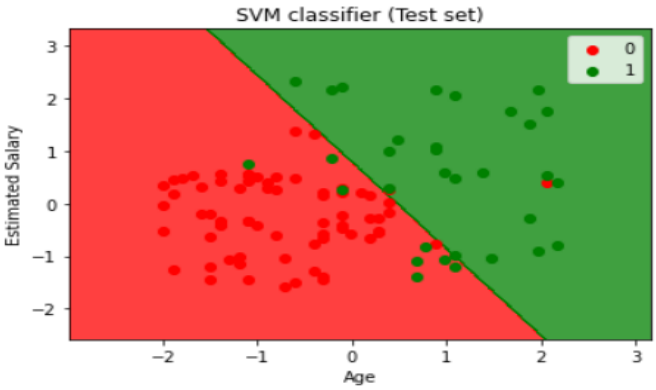


- **Visualizing the test set result:**

In [33]:

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() +
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
```



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**57**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**06**
DOP	
DOS	

Program - Output :

- **Defining Variables -**

```
def hebbian_learning(samples):
        print(f'{"INPUT":^8} {"TARGET":^16}{"WEIGHT CHANGES":^15}{"WEIGHTS":^25}')
         w1, w2, b = 0, 0, 0
        print(' ' * 45, f'({w1:2}, {w2:2}, {b:2})')
        for x1, x2, y in samples:
                w1 = w1 + x1 * y
                w2 = w2 + x2 * y
                b = b + y
                print(f'({x1:2}, {x2:2}) {y:2} ({x1y:2}, {x2y:2}, {y:2}) ({w1:2}, {w2:2}, {b:2})')
```

In [24]:

```
def hebbian_learning(samples):
    print(f'{"INPUT":^8} {"TARGET":^16}{"WEIGHT CHANGES":^15}{"WEIGHTS":^25}')
    w1, w2, b = 0, 0, 0
    print(' ' * 45, f'({w1:2}, {w2:2}, {b:2})')
    for x1, x2, y in samples:
        w1 = w1 + x1 * y
        w2 = w2 + x2 * y
        b = b + y
        print(f'({x1:2}, {x2:2}) {y:2} ({x1:2}, {x2:2}, {y:2}) ({w1:2}, {w2:2}, {
```

- **Defining inputs -**
  1. **AND logic gate**

In [ ]:

```
AND_samples = {
    'binary_input_binary_output': [
        [1, 1, 1],
        [1, 0, 0],
        [0, 1, 0],
        [0, 0, 0]
    ],
    'binary_input_bipolar_output': [
        [1, 1, 1],
        [1, 0, -1],
        [0, 1, -1],
        [0, 0, -1]
    ],
    'bipolar_input_bipolar_output': [
        [ 1, 1, 1],
        [ 1, -1, -1],
        [-1, 1, -1],
        [-1, -1, -1]
    ]
}
```

2. **OR logic gate**
3. **XOR logic gate**

```
OR_samples = {
    'binary_input_binary_output': [
        [1, 1, 1],
        [1, 0, 1],
        [0, 1, 1],
        [0, 0, 0]
    ],
    'binary_input_bipolar_output': [
        [1, 1, 1],
        [1, 0, 1],
        [0, 1, 1],
        [0, 0, -1]
    ],
    'bipolar_input_bipolar_output': [
        [ 1, 1, 1],
        [ 1, -1, 1],
        [-1, 1, 1],
        [-1, -1, -1]
    ]
}
XOR_samples = {
    'binary_input_binary_output': [
        [1, 1, 0],
        [1, 0, 1],
        [0, 1, 1],
        [0, 0, 0]
    ],
    'binary_input_bipolar_output': [
        [1, 1, -1],
        [1, 0, 1],
        [0, 1, 1],
        [0, 0, -1]
    ],
    'bipolar_input_bipolar_output': [
        [ 1, 1, -1],
        [ 1, -1, 1],
        [-1, 1, 1],
        [-1, -1, -1]
    ]
}
```

- **Testing Code with Output -**
  1. **For AND logic gate**

```
In [26]:
```

```
print('-'*20 ,'HEBBIAN LEARNING', '-'*20)
print('AND with Binary Input and Binary Output')
hebbian_learning(AND_samples['binary_input_binary_output'])
print('AND with Binary Input and Bipolar Output')
hebbian_learning(AND_samples['binary_input_bipolar_output'])
print('AND with Bipolar Input and Bipolar Output')
hebbian_learning(AND_samples['bipolar_input_bipolar_output'])
```

```
-------------------- HEBBIAN LEARNING --------------------
AND with Binary Input and Binary Output
 INPUT        TARGET     WEIGHT CHANGES          WEIGHTS
                                              ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1,  0)  0 ( 1,  0,  0) ( 1,  1,  1)
( 0,  1)  0 ( 0,  1,  0) ( 1,  1,  1)
( 0,  0)  0 ( 0,  0,  0) ( 1,  1,  1)
AND with Binary Input and Bipolar Output
 INPUT        TARGET     WEIGHT CHANGES          WEIGHTS
                                              ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1,  0) -1 ( 1,  0, -1) ( 0,  1,  0)
( 0,  1) -1 ( 0,  1, -1) ( 0,  0, -1)
( 0,  0) -1 ( 0,  0, -1) ( 0,  0, -2)
AND with Bipolar Input and Bipolar Output
 INPUT        TARGET     WEIGHT CHANGES          WEIGHTS
                                              ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1, -1) -1 ( 1, -1, -1) ( 0,  2,  0)
(-1,  1) -1 (-1,  1, -1) ( 1,  1, -1)
(-1, -1) -1 (-1, -1, -1) ( 2,  2, -2)
```

2. Using OR logic gate

In [27]:

```
print('-'*20, 'HEBBIAN LEARNING', '-'*20)
print('OR with binary input and binary output')
hebbian_learning(OR_samples['binary_input_binary_output'])
print('OR with binary input and bipolar output')
hebbian_learning(OR_samples['binary_input_bipolar_output'])
print('OR with bipolar input and bipolar output')
hebbian_learning(OR_samples['bipolar_input_bipolar_output'])
```

```
-------------------- HEBBIAN LEARNING --------------------
OR with binary input and binary output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1,  0)  1 ( 1,  0,  1) ( 2,  1,  2)
( 0,  1)  1 ( 0,  1,  1) ( 2,  2,  3)
( 0,  0)  0 ( 0,  0,  0) ( 2,  2,  3)
OR with binary input and bipolar output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1,  0)  1 ( 1,  0,  1) ( 2,  1,  2)
( 0,  1)  1 ( 0,  1,  1) ( 2,  2,  3)
( 0,  0) -1 ( 0,  0, -1) ( 2,  2,  2)
OR with bipolar input and bipolar output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1, -1)  1 ( 1, -1,  1) ( 2,  0,  2)
(-1,  1)  1 (-1,  1,  1) ( 1,  1,  3)
(-1, -1) -1 (-1, -1, -1) ( 2,  2,  2)
```

3. For XOR logic gate

In [28]:

```
print('-'*20, 'HEBBIAN LEARNING', '-'*20)
print('XOR with binary input and binary output')
hebbian_learning(XOR_samples['binary_input_binary_output'])
print('XOR with binary input and bipolar output')
hebbian_learning(XOR_samples['binary_input_bipolar_output'])
print('XOR with bipolar input and bipolar output')
hebbian_learning(XOR_samples['bipolar_input_bipolar_output'])
```

```
-------------------- HEBBIAN LEARNING --------------------
XOR with binary input and binary output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1)  0 ( 1,  1,  0) ( 0,  0,  0)
( 1,  0)  1 ( 1,  0,  1) ( 1,  0,  1)
( 0,  1)  1 ( 0,  1,  1) ( 1,  1,  2)
( 0,  0)  0 ( 0,  0,  0) ( 1,  1,  2)
XOR with binary input and bipolar output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1) -1 ( 1,  1, -1) (-1, -1, -1)
( 1,  0)  1 ( 1,  0,  1) ( 0, -1,  0)
( 0,  1)  1 ( 0,  1,  1) ( 0,  0,  1)
( 0,  0) -1 ( 0,  0, -1) ( 0,  0,  0)
XOR with bipolar input and bipolar output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1) -1 ( 1,  1, -1) (-1, -1, -1)
( 1, -1)  1 ( 1, -1,  1) ( 0, -2,  0)
(-1,  1)  1 (-1,  1,  1) (-1, -1,  1)
(-1, -1) -1 (-1, -1, -1) ( 0,  0,  0)
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**57**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**07**
DOP	
DOS	

Program - Output :

- **Importing Libraries -**
  1. **For plotting**
  2. **For matrix math**
  3. **For normalization + probability density function computation**
  4. **For data preprocessing**

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("white")
%matplotlib inline

import numpy as np

from scipy import stats

import pandas as pd
from math import sqrt, log, exp, pi
from random import uniform
print("import done")
```

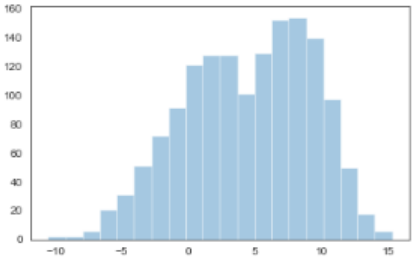
```
import done
```

- **Generating the data yourself, Select μ1, σ1 and μ2, σ2 to generate the data**

In [2]:
```
random_seed=36788765
np.random.seed(random_seed)

Mean1 = 2.0
Standard_dev1 = 4.0
Mean2 = 9.0
Standard_dev2 = 2.0


y1 = np.random.normal(Mean1, Standard_dev1, 1000)
y2 = np.random.normal(Mean2, Standard_dev2, 500)
data=np.append(y1,y2)


Min_graph = min(data)
Max_graph = max(data)
x = np.linspace(Min_graph, Max_graph, 2000)

print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1, Standard_dev1))
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2, Standard_dev2))
sns.distplot(data, bins=20, kde=False);
```

```
Input Gaussian 1: μ = 2.0, σ = 4.0
Input Gaussian 2: μ = 9.0, σ = 2.0
```

```
/home/computer/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



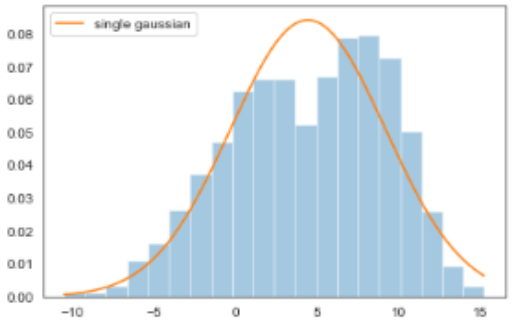
In [3]:
```
class Gaussian:
    "Model univariate Gaussian"
    def __init__(self, mu, sigma):

        self.mu = mu
        self.sigma = sigma

    #probability density function
    def pdf(self, datum):
        "Probability of a data point given the current parameters"
        u = (datum - self.mu) / abs(self.sigma)
        y = (1 / (sqrt(2 * pi) * abs(self.sigma))) * exp(-u * u / 2)
        return y

    def __repr__(self):
        return 'Gaussian({0:4.6}, {1:4.6})'.format(self.mu, self.sigma)
print("done")
```

```
done
```

- **Calculating the mean and standard deviation of the dataset shows it does not fit well (as single Gaussion will not fit the data well)**

In [4]:
```
best_single = Gaussian(np.mean(data), np.std(data))
print('Best single Gaussian: μ = {:.2}, σ = {:.2}'.format(best_single.mu, best_single.sigma))
#fit a single gaussian curve to the data
g_single = stats.norm(best_single.mu, best_single.sigma).pdf(x)
sns.distplot(data, bins=20, kde=False, norm_hist=True);
plt.plot(x, g_single, label='single gaussian');
plt.legend();
```

```
Best single Gaussian: μ = 4.4, σ = 4.8
```



- **EM with 2 Gaussian mixture model**

In [5]:
```
class GaussianMixture_self:
    "Model mixture of two univariate Gaussians and their EM estimation"

    def __init__(self, data, mu_min=min(data), mu_max=max(data), sigma_min=1, sigma_max=1, mix=.5):
        self.data = data
        #todo the Algorithm would be numerical enhanced by normalizing the data first, next do all the EM steps and

        #init with multiple gaussians
        self.one = Gaussian(uniform(mu_min, mu_max),
                            uniform(sigma_min, sigma_max))
        self.two = Gaussian(uniform(mu_min, mu_max),
                            uniform(sigma_min, sigma_max))

        #as well as how much to mix them
        self.mix = mix
```

```
    def Estep(self):
        "Perform an E(stimation)-step, assign each point to gaussian 1 or 2 with a percentage"
        # compute weights
        self.loglike = 0. # = log(p = 1)
        for datum in self.data:
            # unnormalized weights
            wp1 = self.one.pdf(datum) * self.mix
            wp2 = self.two.pdf(datum) * (1. - self.mix)
            # compute denominator
            den = wp1 + wp2
            # normalize
            wp1 /= den
            wp2 /= den      # wp1+wp2= 1, it either belongs to gaussian 1 or gaussion 2
            # add into loglike
            self.loglike += log(den) #freshening up self.loglike in the process
            # yield weight tuple
            yield (wp1, wp2)
```

```
    def Mstep(self, weights):
        "Perform an M(aximization)-step"
        # compute denominators
        (left, rigt) = zip(*weights)
        one_den = sum(left)
        two_den = sum(rigt)

        # compute new means
        self.one.mu = sum(w * d  for (w, d) in zip(left, data)) / one_den
        self.two.mu = sum(w * d  for (w, d) in zip(rigt, data)) / two_den

        # compute new sigmas
        self.one.sigma = sqrt(sum(w * ((d - self.one.mu) ** 2)
                                  for (w, d) in zip(left, data)) / one_den)
        self.two.sigma = sqrt(sum(w * ((d - self.two.mu) ** 2)
                                  for (w, d) in zip(rigt, data)) / two_den)
        # compute new mix
        self.mix = one_den / len(data)


    def iterate(self, N=1, verbose=False):
        "Perform N iterations, then compute log-likelihood"
        for i in range(1, N+1):
            self.Mstep(self.Estep()) #The heart of the algorith, perform E-stepand next M-step
            if verbose:
                print('{0:2} {1}'.format(i, self))
        self.Estep() # to freshen up self.loglike

    def pdf(self, x):
        return (self.mix)*self.one.pdf(x) + (1-self.mix)*self.two.pdf(x)

    def __repr__(self):
        return 'GaussianMixture({0}, {1}, mix={2.03})'.format(self.one,
                                                              self.two,
                                                              self.mix)

    def __str__(self):
        return 'Mixture: {0}, {1}, mix={2:.03})'.format(self.one,
                                                        self.two,
                                                        self.mix)
print("done")
```

```
done
```

- **See the algorithm in action**

In [6]:
```
n_iterations = 20
best_mix = None
best_loglike = float('-inf')
mix = GaussianMixture_self(data)
for _ in range(n_iterations):
    try:
        #train!
        mix.iterate(verbose=True)
        if mix.loglike > best_loglike:
            best_loglike = mix.loglike
            best_mix = mix

    except (ZeroDivisionError, ValueError, RuntimeWarning): # Catch division errors from bad starts, and just throw
        print("one less")
        pass
```

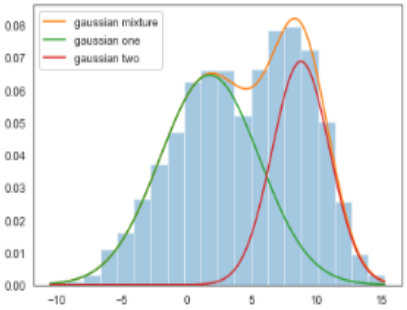
```
 1 Mixture: Gaussian(-8.75384, 1.35749), Gaussian(4.49768, 4.69147), mix=0.00361)
 1 Mixture: Gaussian(-8.4477, 1.685), Gaussian(4.47239, 4.72355), mix=0.00175)
 1 Mixture: Gaussian(-7.95802, 1.91244), Gaussian(4.46376, 4.73467), mix=0.00112)
 1 Mixture: Gaussian(-7.33053, 2.02968), Gaussian(4.45984, 4.7399), mix=0.000852)
 1 Mixture: Gaussian(-6.66907, 2.02428), Gaussian(4.45795, 4.74258), mix=0.000733)
 1 Mixture: Gaussian(-6.06975, 1.90364), Gaussian(4.45718, 4.74393), mix=0.000702)
 1 Mixture: Gaussian(-5.59631, 1.69731), Gaussian(4.45718, 4.74442), mix=0.000734)
 1 Mixture: Gaussian(-5.28418, 1.46631), Gaussian(4.45789, 4.7442), mix=0.000831)
 1 Mixture: Gaussian(-5.12307, 1.27697), Gaussian(4.45936, 4.74326), mix=0.000998)
 1 Mixture: Gaussian(-5.05818, 1.14023), Gaussian(4.46164, 4.74163), mix=0.00124)
 1 Mixture: Gaussian(-5.04384, 1.03771), Gaussian(4.4648, 4.73926), mix=0.00158)
 1 Mixture: Gaussian(-5.0565, 0.95484), Gaussian(4.46902, 4.73603), mix=0.00202)
 1 Mixture: Gaussian(-5.08416, 0.883156), Gaussian(4.47452, 4.73175), mix=0.00259)
 1 Mixture: Gaussian(-5.11979, 0.818027), Gaussian(4.48157, 4.7262), mix=0.00331)
 1 Mixture: Gaussian(-5.15861, 0.757302), Gaussian(4.49033, 4.71921), mix=0.0042)
 1 Mixture: Gaussian(-5.197, 0.700611), Gaussian(4.5008, 4.71078), mix=0.00526)
 1 Mixture: Gaussian(-5.23221, 0.648837), Gaussian(4.5126, 4.70116), mix=0.00645)
 1 Mixture: Gaussian(-5.26261, 0.603408), Gaussian(4.52492, 4.69103), mix=0.00768)
 1 Mixture: Gaussian(-5.28775, 0.565431), Gaussian(4.53658, 4.68135), mix=0.00883)
 1 Mixture: Gaussian(-5.30819, 0.535008), Gaussian(4.54648, 4.67304), mix=0.00981)
```

- **FInally finding best Mixture Gaussian model**

In [7]:
```
n_iterations = 300
n_random_restarts = 4
best_mix = None
best_loglike = float('-inf')
print('Computing best model with random restarts...\n')
for _ in range(n_random_restarts):
    mix = GaussianMixture_self(data)
    for _ in range(n_iterations):
        try:
            mix.iterate()
            if mix.loglike > best_loglike:
                best_loglike = mix.loglike
                best_mix = mix
        except (ZeroDivisionError, ValueError, RuntimeWarning): # Catch division errors from bad starts, and just t
            pass
#print('Best Gaussian Mixture : μ = {:.2}, σ = {:.2} with μ = {:.2}, σ = {:.2}'.format(best_mix.one.mu, best_mix.on

print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1, Standard_dev1))
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2, Standard_dev2))
print('Gaussian {:}: μ = {:.2}, σ = {:.2}, weight = {:.2}'.format("1", best_mix.one.mu, best_mix.one.sigma, best_mi
print('Gaussian {:}: μ = {:.2}, σ = {:.2}, weight = {:.2}'.format("2", best_mix.two.mu, best_mix.two.sigma, (1-best
#Show mixture
sns.distplot(data, bins=20, kde=False, norm_hist=True);
g_both = [best_mix.pdf(e) for e in x]
plt.plot(x, g_both, label='gaussian mixture');
g_left = [best_mix.one.pdf(e) * best_mix.mix for e in x]
plt.plot(x, g_left, label='gaussian one');
g_right = [best_mix.two.pdf(e) * (1-best_mix.mix) for e in x]
plt.plot(x, g_right, label='gaussian two');
plt.legend();
```

```
Computing best model with random restarts...

Input Gaussian 1: μ = 2.0, σ = 4.0
Input Gaussian 2: μ = 9.0, σ = 2.0
Gaussian 1: μ = 1.8, σ = 3.8, weight = 0.62
Gaussian 2: μ = 8.8, σ = 2.2, weight = 0.38
```

```
/home/computer/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a fig
ure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**57**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**08**
DOP	
DOS	

Program - Output :

- **Importing Libraries -**

In [1]:

```
from tabulate import tabulate
```

- **For AND gate**

In [2]:

```
#inputs
x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
w1 = [1, 1, 1, 1]
w2 = [1, 1, 1, 1]
t = 2
#output
print("x1    x2    w1   w2     t     0")
for i in range(len(x1)):
    if ( x1[i]*w1[i] + x2[i]*w2[i] ) >= t:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 1)
    else:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 0)
```

```
x1    x2    w1   w2     t     0
0     0     1     1     2     0
0     1     1     1     2     0
1     0     1     1     2     0
1     1     1     1     2     1
```

- **For OR gate**

In [3]:

```
#inputs
x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
w1 = [1, 1, 1, 1]
w2 = [1, 1, 1, 1]
t = 1
#output
print("x1    x2    w1   w2     t     0")
for i in range(len(x1)):
    if ( x1[i]*w1[i] + x2[i]*w2[i] ) >= t:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 1)
    else:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 0)
```

```
x1    x2    w1   w2     t     0
0     0     1     1     1     0
0     1     1     1     1     1
1     0     1     1     1     1
1     1     1     1     1     1
```

- **For NOT gate**

In [4]:

```
#inputs
x = [0, 1]
w = [-1, -1]
t = 0
#output
print("x       w       t       O")
for i in range(len(x)):
    if ( x[i]*w[i]) >= t:
        print(x[i],'    ',w[i],'    ',t,'    ', 1)
    else:
        print(x[i],'    ',w[i],'    ',t,'    ', 0)
```

```
x       w       t       O
0       -1      0       1
1       -1      0       0
```

- **For NAND gate**

In [5]:

```
#inputs
x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
w1 = [-1, -1, -1, -1]
w2 = [-1, -1, -1, -1]
t = -2
#output
print("x1    x2    w1     w2      t     O")
for i in range(len(x1)):
    if ( x1[i]*w1[i] + x2[i]*w2[i] ) > t:
        print(x1[i],'    ',x2[i],'    ',w1[i],'    ',w2[i],'    ',t,'    ', 1)
    else:
        print(x1[i],'    ',x2[i],'    ',w1[i],'    ',w2[i],'    ',t,'    ', 0)
```

```
x1    x2    w1     w2      t     O
0     0     -1     -1     -2     1
0     1     -1     -1     -2     1
1     0     -1     -1     -2     1
1     1     -1     -1     -2     0
```

- **For NOR gate**

In [6]:

```
#inputs
x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
w1 = [1, 1, 1, 1]
w2 = [1, 1, 1, 1]
t = 0
#output
print("x1    x2    w1   w2     t     O")
for i in range(len(x1)):
    if ( x1[i]*w1[i] + x2[i]*w2[i] ) <= t:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 1)
    else:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 0)
```

```
x1    x2    w1   w2     t     O
0     0     1     1     0     1
0     1     1     1     0     0
1     0     1     1     0     0
1     1     1     1     0     0
```

- **For EXOR gate**

In [7]:

```
#inputs
x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
w1 = [1, 1, 1, 1]
w2 = [1, 1, 1, 1]
w3 = [1, 1, 1, 1]
w4 = [-1, -1, -1, -1]
w5 = [-1, -1, -1, -1]
w6 = [1, 1, 1, 1]
t1 = [0.5,0.5,0.5,0.5]
t2 = [-1.5,-1.5,-1.5,-1.5]
t3 = [1.5,1.5,1.5,1.5]
def XOR (a, b):
    if a != b:
        return 1
    else:
        return 0
#output
print('x1    x2    w1    w2    w3     w4    w5     w6    t1        t2      t3     O
for i in range(len(x1)):
    print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',w3[i],'   ',w4[i],'   ',w
```

```
x1    x2    w1    w2    w3     w4    w5     w6    t1        t2      t3
0
0     0     1     1     1     -1    -1     1     0.5     -1.5     1.5
0
0     1     1     1     1     -1    -1     1     0.5     -1.5     1.5
1
1     0     1     1     1     -1    -1     1     0.5     -1.5     1.5
1
1     1     1     1     1     -1    -1     1     0.5     -1.5     1.5
0
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**57**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**09**
DOP	
DOS	

Program - Output :

- **Importing Libraries -**

In [1]:

```
import numpy as np
import pandas as pd
```

- **Importing data as Pandas DataFrame**

In [6]:

```
data=pd.read_csv('/home/computer/Documents/iris.csv')
data.columns=['Sepal_len_cm','Sepal_wid_cm','Petal_len_cm','Petal_wid_cm','Type']
data.head(10)
```

Out[6]:

	Sepal_len_cm	Sepal_wid_cm	Petal_len_cm	Petal_wid_cm	Type
0	4.9	3.0	1.4	0.2	0
1	4.7	3.2	1.3	0.2	0
2	4.6	3.1	1.5	0.2	0
3	5.0	3.6	1.4	0.2	0
4	5.4	3.9	1.7	0.4	0
5	4.6	3.4	1.4	0.3	0
6	5.0	3.4	1.5	0.2	0
7	4.4	2.9	1.4	0.2	0
8	4.9	3.1	1.5	0.1	0
9	5.4	3.7	1.5	0.2	0

- **Training (using Sigmoid function as the activation function)**

In [7]:

```
def activation_func(value):    #Tangent Hypotenuse
    #return (1/(1+np.exp(-value)))
    return ((np.exp(value)-np.exp(-value))/(np.exp(value)+np.exp(-value)))
```

In [8]:

```
def perceptron_train(in_data,labels,alpha):
    X=np.array(in_data)
    y=np.array(labels)
    weights=np.random.random(X.shape[1])
    original=weights
    bias=np.random.random_sample()
    for key in range(X.shape[0]):
        a=activation_func(np.matmul(np.transpose(weights),X[key]))
        yn=0
        if a>=0.7:
            yn=1
        elif a<=(-0.7):
            yn=-1
        weights=weights+alpha*(yn-y[key])*X[key]
        print('Iteration '+str(key)+': '+str(weights))
    print('Difference: '+str(weights-original))
    return weights
```

- **Testing and Scoring**

In [9]:

```
def perceptron_test(in_data,label_shape,weights):
    X=np.array(in_data)
    y=np.zeros(label_shape)
    for key in range(X.shape[1]):
        a=activation_func((weights*X[key]).sum())
        y[key]=0
        if a>=0.7:
            y[key]=1
        elif a<=(-0.7):
            y[key]=-1
    return y
```

In [10]:

```
def score(result,labels):
    difference=result-np.array(labels)
    correct_ctr=0
    for elem in range(difference.shape[0]):
        if difference[elem]==0:
            correct_ctr+=1
    score=correct_ctr*100/difference.size
    print('Score='+str(score))
```

- **Dividing dataframe**

In [11]:

```
# Dividing DataFrame "data" into "d_train" (60%) and "d_test" (40%)
divider = np.random.rand(len(data)) < 0.70
d_train=data[divider]
d_test=data[~divider]
```

In [12]:

```
# Dividing d_train into data and labels/targets
d_train_y=d_train['Type']
d_train_X=d_train.drop(['Type'],axis=1)

# Dividing d_train into data and labels/targets
d_test_y=d_test['Type']
d_test_X=d_test.drop(['Type'],axis=1)
```

- **Getting learning rate**

In [13]:

```
# Learning rate
alpha = 0.01

# Train
weights = perceptron_train(d_train_X, d_train_y, alpha)
```

```
Iteration 0: [0.99990004 0.07493098 0.36786327 0.41037301]
Iteration 1: [1.04690004 0.10693098 0.38086327 0.41237301]
Iteration 2: [1.09290004 0.13793098 0.39586327 0.41437301]
Iteration 3: [1.14290004 0.17393098 0.40986327 0.41637301]
Iteration 4: [1.19690004 0.21293098 0.42686327 0.42037301]
Iteration 5: [1.24690004 0.24693098 0.44186327 0.42237301]
Iteration 6: [1.30090004 0.28593098 0.45486327 0.42637301]
Iteration 7: [1.35190004 0.32093098 0.46886327 0.42937301]
Iteration 8: [1.40890004 0.35893098 0.48586327 0.43237301]
Iteration 9: [1.45990004 0.39693098 0.50086327 0.43537301]
Iteration 10: [1.50590004 0.43293098 0.51086327 0.43737301]
Iteration 11: [1.55690004 0.46593098 0.52786327 0.44237301]
Iteration 56: [5.97290004 2.67793098 3.38386327 1.30637301]
Iteration 57: [6.09890004 2.72393098 3.47186327 1.33237301]
Iteration 58: [6.21090004 2.78393098 3.55386327 1.35837301]
Iteration 59: [6.32090004 2.83393098 3.63386327 1.38437301]
Iteration 60: [6.43090004 2.88593098 3.72186327 1.40837301]
Iteration 61: [6.55290004 2.94593098 3.81386327 1.43637301]
Iteration 62: [6.66890004 2.99793098 3.89386327 1.46037301]
Iteration 63: [6.76890004 3.04393098 3.95086327 1.48037301]
Iteration 90: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 91: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 92: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 93: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 94: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 95: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 96: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 97: [7.21090004 3.26593098 4.27186327 1.57837301]
Difference: [6.26  3.221 3.918 1.17 ]
```

- **Testing and Calculating score**

In [14]:

```
# Test
result_test=perceptron_test(d_test_X,d_test_y.shape,weights)
```

In [15]:

```
# Calculate score
score(result_test,d_test_y)
```

```
Score=35.294117647058826
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**57**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**10**
DOP	
DOS	

Program - Output :

- **Importing Libraries -**

In [1]:
```
# importing required libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **Importing or loading the dataset**

In [4]:
```
# importing or loading the dataset
dataset = pd.read_csv('/home/computer/Downloads/Wine.csv')

# distributing the dataset into two components X and Y
X = dataset.iloc[:, 0:13].values
y = dataset.iloc[:, 13].values
```

- **Splitting the dataset into the Training set and Test set**

In [5]:
```
# Splitting the X and Y into the
# Training set and Testing 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, ra
```

- **Feature Scaling**

In [6]:
```
# performing preprocessing part
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

- **Applying the PCA function into the training and testing set for analysis**

In [7]:
```
# Applying PCA function on training
# and testing set of X component
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
```

- **Fitting Logistic Regression To the training set**

In [8]:
```
# Fitting Logistic Regression To the training set
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Out[8]:
```
LogisticRegression(random_state=0)
```

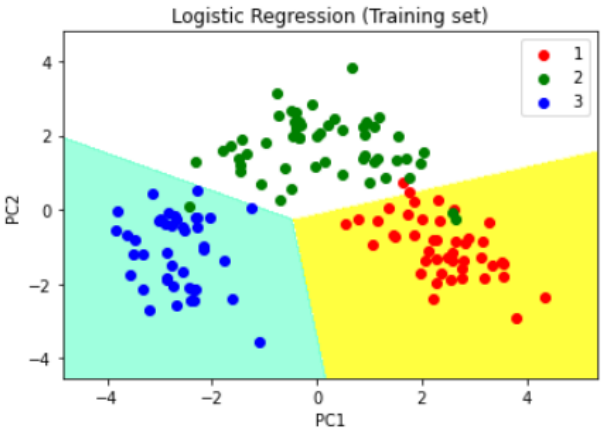
- **Predicting the test set result and making confusion matrix**

In [9]:
```
# Predicting the test set result using
# predict function under LogisticRegression
y_pred = classifier.predict(X_test)
```

In [10]:
```
# making confusion matrix between
# test set of Y and predicted value.
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
```

- **Predicting the training set result**

In [11]:
```
# Predicting the training set
# result through scatter plot
from matplotlib.colors import ListedColormap

X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1, step =
                               np.arange(start = X_set[:, 1].min() -
                               stop = X_set[:, 1].max() + 1, step =

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
                     X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,
                     cmap = ListedColormap(('yellow', 'white', 'aquamarine

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
        plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                                c = ListedColormap(('red', 'green', 'blue'))(

plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1') # for Xlabel
plt.ylabel('PC2') # for Ylabel
plt.legend() # to show legend

# show scatter plot
plt.show()
```



- **Visualizing the Test set results**

In [12]:

```
# Visualising the Test set results through scatter plot
from matplotlib.colors import ListedColormap

X_set, y_set = X_test, y_test

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1, step =
                               np.arange(start = X_set[:, 1].min() -
                               stop = X_set[:, 1].max() + 1, step =

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
                        X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,
                        cmap = ListedColormap(('yellow', 'white', 'aquamarine

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
        plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                                    c = ListedColormap(('red', 'green', 'blue'))(

# title for scatter plot
plt.title('Logistic Regression (Test set)')
plt.xlabel('PC1') # for Xlabel
plt.ylabel('PC2') # for Ylabel
plt.legend()

# show scatter plot
plt.show()
```