

MACHINE LEARNING LAB



**Bachelor of Technology
(Computer Science Engineering)**

SUBMITTED BY:

**Rambhu yadav
Semester : 6th**

URN : 1903355

INSTITUTE : CTIEMT

SUBMITTED TO :

Ms. Karuna

INDEX

| Sr.No. | Experiments | Page | Signature |
|---------------|---|-------------|------------------|
| 1. | Implement data pre-processing | | |
| 2. | Deploy Simple Linear Regression | | |
| 3. | Simulate Multiple Linear Regression | | |
| 4. | Implement Decision Tree | | |
| 5. | Deploy Random forest classification | | |
| 6. | Simulate Naïve Bayes algorithm | | |
| 7. | Implement K-Nearest Neighbors (K-NN), k-Means | | |
| 8. | Deploy Support Vector Machine, Apriori algorithm | | |

EXPERIMENT NO-1

AIM: Implement data pre-processing

Data Preprocessing in Machine learning

Data preprocessing 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.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

Steps involves in Data pre-processing:

- **Getting the dataset** ◦ **Importing libraries** ◦ **Importing datasets** ◦ **Finding Missing Data** ◦ **Encoding Categorical Data** ◦ **Splitting dataset into training and test set** ◦ **Feature scaling**

1) Get the Dataset

To create a machine learning model, 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**.

Dataset may be of different formats for different purposes, such as, if we want to create a machine learning model for business purpose, then dataset will be different with the dataset required for a liver patient. So each dataset is different from another 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.

What is a CSV 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. We can also create our dataset by gathering data using various API with Python and put that data into a .csv file.

2) Importing Libraries

In order to perform data preprocessing 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 preprocessing, which are:

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.

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.

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.

```
1 # importing libraries
2 import numpy as nm
3 import matplotlib.pyplot as mtp
4 import pandas as pd
5
```

3) Importing the Datasets

Now we need to import the datasets which we have collected for our machine learning project. But before importing a dataset, we need to set the current directory as a working directory. To set a working directory in Spyder IDE, we need to follow the below steps:

1. Save your Python file in the directory which contains dataset.
2. Go to File explorer option in Spyder IDE, and select the required directory.
3. Click on F5 button or run option to execute the file.

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.

Loading the dataset

```
dataset = pd.read_excel('age_salary.xls')
```

The data set used here is as simple as shown below:

```
1 # importing libraries
2 import numpy as nm
3 import matplotlib.pyplot as mtp
4 import pandas as pd
5
```

Note: The 'nan' you see in some cells of the dataframe denotes the missing fields

Classifying the dependent and Independent Variables

Here we just have 2 factors, age and salary. Salary is the dependent factor that changes with the independent factor age. Now let's classify them programmatically.

X = dataset.iloc[:, :-1].values #Takes all rows of all columns except the last column

Y = dataset.iloc[:, -1].values # Takes all rows of the last column

- X : independent variable set
- Y : dependent variable set

The dependent and independent values are stored in different arrays. In case of multiple independent variables use X = dataset.iloc[:, a:b].values where a is the starting range and b is the ending range (column indices).

| | 0 |
|----|-----|
| 0 | 25 |
| 1 | 27 |
| 2 | 50 |
| 3 | 35 |
| 4 | 40 |
| 5 | 35 |
| 6 | nan |
| 7 | 48 |
| 8 | 50 |
| 9 | 37 |
| 10 | 21 |
| 11 | nan |
| 12 | 63 |

| | 0 |
|----|-------|
| 0 | 35000 |
| 1 | 40000 |
| 2 | 54000 |
| 3 | nan |
| 4 | 60000 |
| 5 | 58000 |
| 6 | 52000 |
| 7 | 79000 |
| 8 | 83000 |
| 9 | nan |
| 10 | 24000 |
| 11 | 60000 |
| 12 | 70000 |

4) Dealing with Missing Data

We have already noticed the missing fields in the data denoted by “nan”. Machine learning models cannot accommodate missing fields in the data they are provided with. So the missing fields must be filled with values that will not affect the variance of the data or make it more noisy.

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy="mean")
X = imp.fit_transform(X)
Y = Y.reshape(-1,1)
Y = imp.fit_transform(Y)
Y = Y.reshape(-1)
```

The scikit-learn library’s SimpleImputer Class allows us to impute the missing fields in a dataset with valid data. In the above code, we have used the default strategy for filling missing values which is the mean. The imputer cannot be applied on 1D arrays and since Y is a 1D array, it needs to be converted to a compatible shape. The reshape functions allows us to reshape any array. The fit_transform() method will fit the imputer object and then transforms the arrays.

Output

| | 0 | | 0 |
|----|---------|----|---------|
| 0 | 25 | 0 | 35000 |
| 1 | 27 | 1 | 40000 |
| 2 | 50 | 2 | 54000 |
| 3 | 35 | 3 | 55909.1 |
| 4 | 40 | 4 | 60000 |
| 5 | 35 | 5 | 58000 |
| 6 | 39.1818 | 6 | 52000 |
| 7 | 48 | 7 | 79000 |
| 8 | 50 | 8 | 83000 |
| 9 | 37 | 9 | 55909.1 |
| 10 | 21 | 10 | 24000 |
| 11 | 39.1818 | 11 | 60000 |
| 12 | 63 | 12 | 70000 |

5) Dealing with Categorical Data

When dealing with large and real-world datasets, categorical data is almost inevitable. Categorical variables represent types of data which may be divided into groups. Examples of categorical variables are race, sex, age group, educational level etc. These variables often have letters or words as its values. Since machine learning models are all about numbers and calculations, these categorical variables need to be coded in to numbers. Having coded the categorical variable into numbers may just not be enough.

For example, consider the dataset below with 2 categorical features nation and purchased_item. Let us assume that the dataset is a record of how age, salary and country of a person determine if an item is purchased or not. Thus purchased_item is the dependent factor and age, salary and nation are the independent factors.

| Index | nation | purchased_item | age | salary |
|-------|---------|----------------|------|---------|
| 0 | India | No | 25 | 35000 |
| 1 | Russia | Yes | 27 | 40000 |
| 2 | Germany | No | 50 | 54000 |
| 3 | Russia | No | 35 | 55909.1 |
| 4 | Germany | Yes | 40 | 60000 |
| 5 | India | Yes | 35 | 58000 |
| 6 | Russia | No | 39.1 | 52000 |
| 7 | India | Yes | 48 | 79000 |
| 8 | Germany | No | 50 | 83000 |
| 9 | India | Yes | 37 | 55909.1 |
| 10 | Germany | No | 21 | 24000 |
| 11 | India | Yes | 39.1 | 60000 |
| 12 | Russia | No | 63 | 70000 |

It has 3 countries listed. In a larger dataset, these may be large groups of data.

Since countries don't have a mathematical relation between them unless we are considering some known factors such as size or population etc , coding them in numbers will not work, as a number may be less than or greater than another number. Dummy variables are the solution. Using one hot encoding we will

create a dummy variable for each of the category in the column. And uses binary encoding for each dummy variable. We do not need to create dummy variables for the feature purchased_item as it has only 2 categories either yes or no.

```
dataset = pd.read_csv('dataset.csv')
X = dataset.iloc[:,[0,2,3]].values
Y = dataset.iloc[:,1].values
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
le_X = LabelEncoder()
X[:,0] = le_X.fit_transform(X[:,0])
ohe_X = OneHotEncoder(categorical_features = [0])
X = ohe_X.fit_transform(X).toarray()
```

Output

| | | | | | |
|----|---|---|---|------|---------|
| 0 | 0 | 1 | 0 | 25 | 35000 |
| 1 | 0 | 0 | 1 | 27 | 40000 |
| 2 | 1 | 0 | 0 | 50 | 54000 |
| 3 | 0 | 0 | 1 | 35 | 55909.1 |
| 4 | 1 | 0 | 0 | 40 | 60000 |
| 5 | 0 | 1 | 0 | 35 | 58000 |
| 6 | 0 | 0 | 1 | 39.1 | 52000 |
| 7 | 0 | 1 | 0 | 48 | 79000 |
| 8 | 1 | 0 | 0 | 50 | 83000 |
| 9 | 0 | 1 | 0 | 37 | 55909.1 |
| 10 | 1 | 0 | 0 | 21 | 24000 |
| 11 | 0 | 1 | 0 | 39.1 | 60000 |
| 12 | 0 | 0 | 1 | 63 | 70000 |

The the first 3 columns are the dummy features representing Germany,India and Russia respectively.The 1's in each column represent that the person belongs to that specific country. **Y = le_X.fit_transform(Y)**

Output:

| | |
|----|---|
| | 0 |
| 0 | 0 |
| 1 | 1 |
| 2 | 0 |
| 3 | 0 |
| 4 | 1 |
| 5 | 1 |
| 6 | 0 |
| 7 | 1 |
| 8 | 0 |
| 9 | 1 |
| 10 | 0 |
| 11 | 1 |
| 12 | 0 |

6) Splitting the Dataset into Training and Testing sets

All machine learning models require us to provide a training set for the machine so that the model can train from that data to understand the relations between features

and can predict for new observations. When we are provided a single huge dataset with too much of observations, it is a good idea to split the dataset into two, a training_set and a test_set, so that we can test our model. Scikit-learn comes with a method called train_test_split to help us with this task.

```
from sklearn.model_selection import train_test_split  
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3,  
random_state = 0)
```

The above code will split X and Y into two subsets each.

- **test_size:** the desired size of the test_set. 0.3 denotes 30%.
- **random_state:** This is used to preserve the uniqueness. The split will happen uniquely for a random_state.

7) Scaling the features

Since machine learning models rely on numbers to solve relations it is important to have similarly scaled data in a dataset. Scaling ensures that all data in a dataset falls in the same range. Unscaled data can cause inaccurate or false predictions. Some machine learning algorithms can handle feature scaling on its own and doesn't require it explicitly.

The StandardScaler class from the scikit-learn library can help us scale the dataset.

```
from sklearn.preprocessing import StandardScaler sc_X  
= StandardScaler()  
X_train = sc_X.fit_transform(X_train)  
X_test = sc_X.transform(X_test)
```

```
sc_y = StandardScaler()  
Y_train = Y_train.reshape((len(Y_train), 1))  
Y_train = sc_y.fit_transform(Y_train)  
Y_train = Y_train.ravel()
```

Output

X_train before scaling :

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|----|---------|
| 0 | 1 | 0 | 0 | 50 | 54000 |
| 1 | 1 | 0 | 0 | 50 | 83000 |
| 2 | 0 | 0 | 1 | 27 | 40000 |
| 3 | 0 | 1 | 0 | 48 | 79000 |
| 4 | 0 | 1 | 0 | 37 | 55909.1 |
| 5 | 0 | 0 | 1 | 35 | 55909.1 |
| 6 | 0 | 1 | 0 | 25 | 35000 |
| 7 | 0 | 1 | 0 | 35 | 58000 |
| 8 | 0 | 0 | 1 | 63 | 70000 |

X_train after scaling :

| | 0 | 1 | 2 | 3 | 4 |
|---|-----------|-----------|-----------|-----------|------------|
| 0 | 1.87083 | -0.894427 | -0.707107 | 0.758848 | -0.327639 |
| 1 | 1.87083 | -0.894427 | -0.707107 | 0.758848 | 1.58037 |
| 2 | -0.534522 | -0.894427 | 1.41421 | -1.20467 | -1.24875 |
| 3 | -0.534522 | 1.11803 | -0.707107 | 0.588107 | 1.3172 |
| 4 | -0.534522 | 1.11803 | -0.707107 | -0.350967 | -0.202032 |
| 5 | -0.534522 | -0.894427 | 1.41421 | -0.521708 | -0.202032 |
| 6 | -0.534522 | 1.11803 | -0.707107 | -1.37541 | -1.57772 |
| 7 | -0.534522 | 1.11803 | -0.707107 | -0.521708 | -0.0644645 |
| 8 | -0.534522 | -0.894427 | 1.41421 | 1.86866 | 0.725058 |

EXPERIMENT NO-2

AIM: Deploy Simple Linear Regression

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

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

dataset = pd.read_csv('data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=1/3,
                                                    random_state=0)
reg = LinearRegression()
reg.fit(x_train, y_train)

y_predict = reg.predict(x_train)
plt.scatter(x_train, y_train, color='red')
plt.plot(x_train, y_predict, color='blue')
plt.title('Salary VS Experience (Training set)')
plt.xlabel('Year of Experience')
plt.ylabel('Salary')
plt.show()

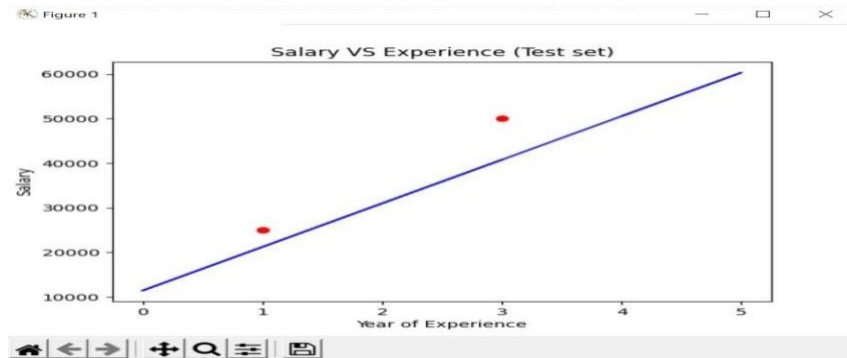
plt.scatter(x_test, y_test, color='red')
plt.plot(x_train, y_predict, color='blue')
plt.title('Salary VS Experience (Test set)')
plt.xlabel('Year of Experience')
plt.ylabel('Salary')
plt.show()
```

OUTPUT

```

Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\HP\Desktop\linear regression.py =====
3079.710144927536
>>> |

```



EXPERIMENT NO-3

AIM: Simulate Multiple Linear Regression

```
import matplotlib.pyplot as plt
```

```
import pandas as pd import
```

```
numpy as np
```

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn import metrics df =
```

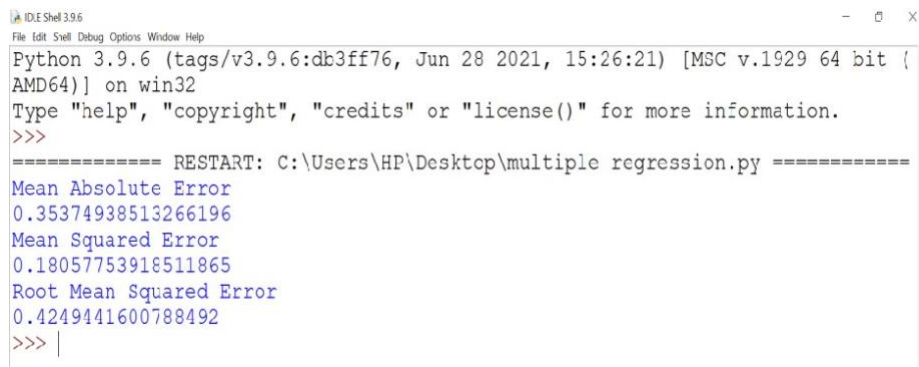
```
pd.read_csv("Diabetes.csv") x = df.iloc[:, :-1].values
```

```
y = df.iloc[:, -1].values
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 100)
```

```
reg = LinearRegression()
reg.fit(x_train,y_train)
y_predict= reg.predict(x_test)
print("Mean Absolute Error")
print(metrics.mean_absolute_error(y_test,y_predict)) print("Mean
Squared Error")
print(metrics.mean_squared_error(y_test,y_predict)) print("Root
Mean Squared Error")
print(np.sqrt(metrics.mean_squared_error(y_test,y_predict)))
```

OUTPUT



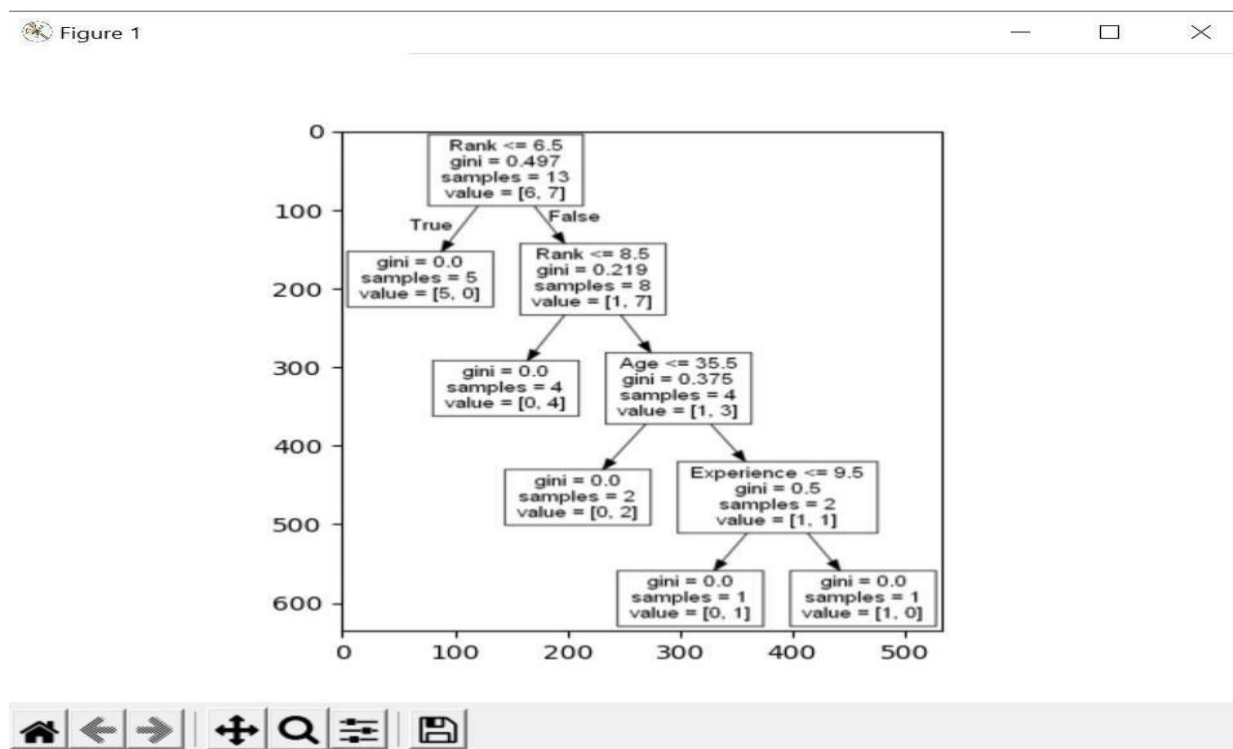
```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\HP\Desktop\multiple regression.py =====
Mean Absolute Error
0.35374938513266196
Mean Squared Error
0.18057753918511865
Root Mean Squared Error
0.4249441600788492
>>> |
```

EXPERIMENT NO-4

AIM: Implement Decision Tree

```
import pandas from
sklearn import tree
import pydotplus
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt import
matplotlib.image as pltimg df =
pandas.read_csv("dt.csv") d = {'UK': 0, 'USA':
1, 'N': 2} df['Nationality'] =
df['Nationality'].map(d) d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)
features = ['Age', 'Experience', 'Rank',
'Nationality'] X = df[features] y = df['Go']
dtree = DecisionTreeClassifier() dtree
= dtree.fit(X, y)
data = tree.export_graphviz(dtree, out_file=None, feature_names=features)
graph = pydotplus.graph_from_dot_data(data)
graph.write_png('mydecisiontree.png')
img=pltimg.imread('mydecisiontree.png')
imgplot = plt.imshow(img) plt.show()
```

OUTPUT



EXPERIMENT NO-5

AIM: Deploy Random forest classification

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix

dataset = pd.read_csv('Diabetes.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)

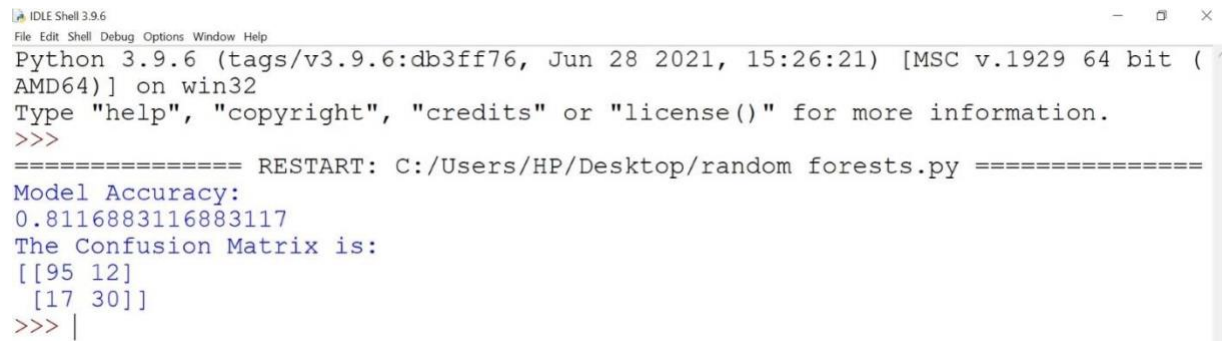
from sklearn.preprocessing import StandardScaler
scaling_x = StandardScaler()
x_train = scaling_x.fit_transform(x_train)
x_test = scaling_x.transform(x_test)

rfc = RandomForestClassifier()
rfc.fit(x_train, y_train)

y_pred = rfc.predict(x_test)
print("Model Accuracy:")
print(rfc.score(x_test, y_test))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("The Confusion Matrix is:")
print(cm)
```

OUTPUT



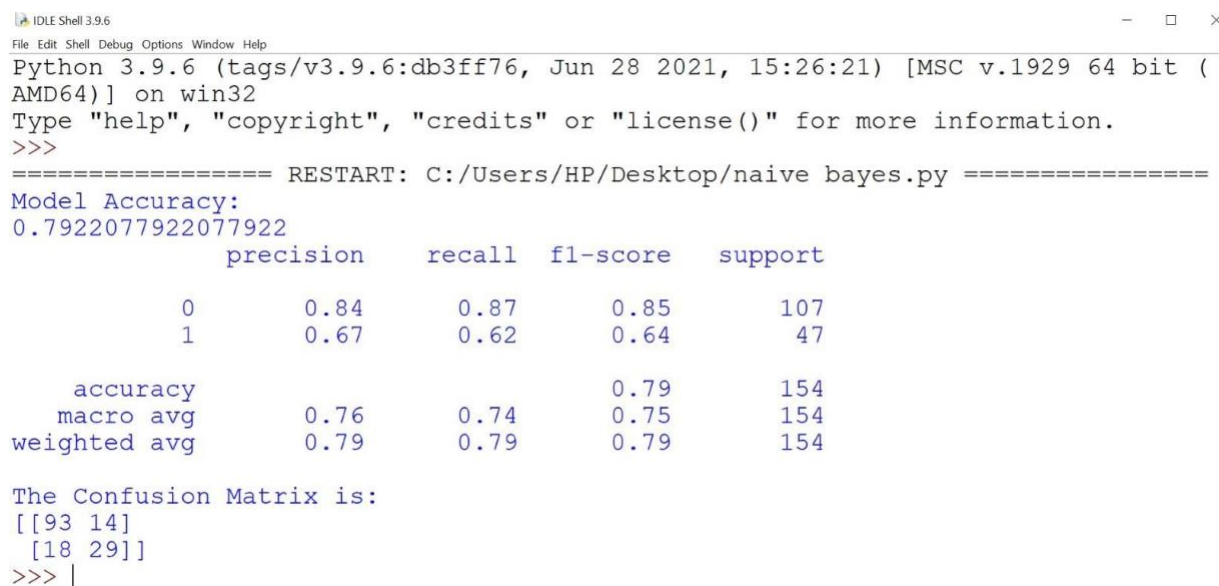
```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:/Users/HP/Desktop/random forests.py =====
Model Accuracy:
0.8116883116883117
The Confusion Matrix is:
[[95 12]
 [17 30]]
>>> |
```


EXPERIMENT NO-6

AIM: Simulate Naïve Bayes algorithm

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.metrics import confusion_matrix
dataset = pd.read_csv('Diabetes.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
from sklearn.preprocessing import StandardScaler
scaling_x = StandardScaler()
x_train = scaling_x.fit_transform(x_train)
x_test = scaling_x.transform(x_test)
model = GaussianNB()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print("Model Accuracy:")
print(model.score(x_test, y_test))
print(metrics.classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print("The Confusion Matrix is:")
print(cm)
```

OUTPUT



```

IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/HP/Desktop/naive bayes.py =====
Model Accuracy:
0.7922077922077922

```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.84 | 0.87 | 0.85 | 107 |
| 1 | 0.67 | 0.62 | 0.64 | 47 |
| accuracy | | | 0.79 | 154 |
| macro avg | 0.76 | 0.74 | 0.75 | 154 |
| weighted avg | 0.79 | 0.79 | 0.79 | 154 |

```

The Confusion Matrix is:
[[93 14]
 [18 29]]
>>> |

```

Experiment No-7

AIM: Implement K-Nearest Neighbors (K-NN), k-Means a) KNN

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

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

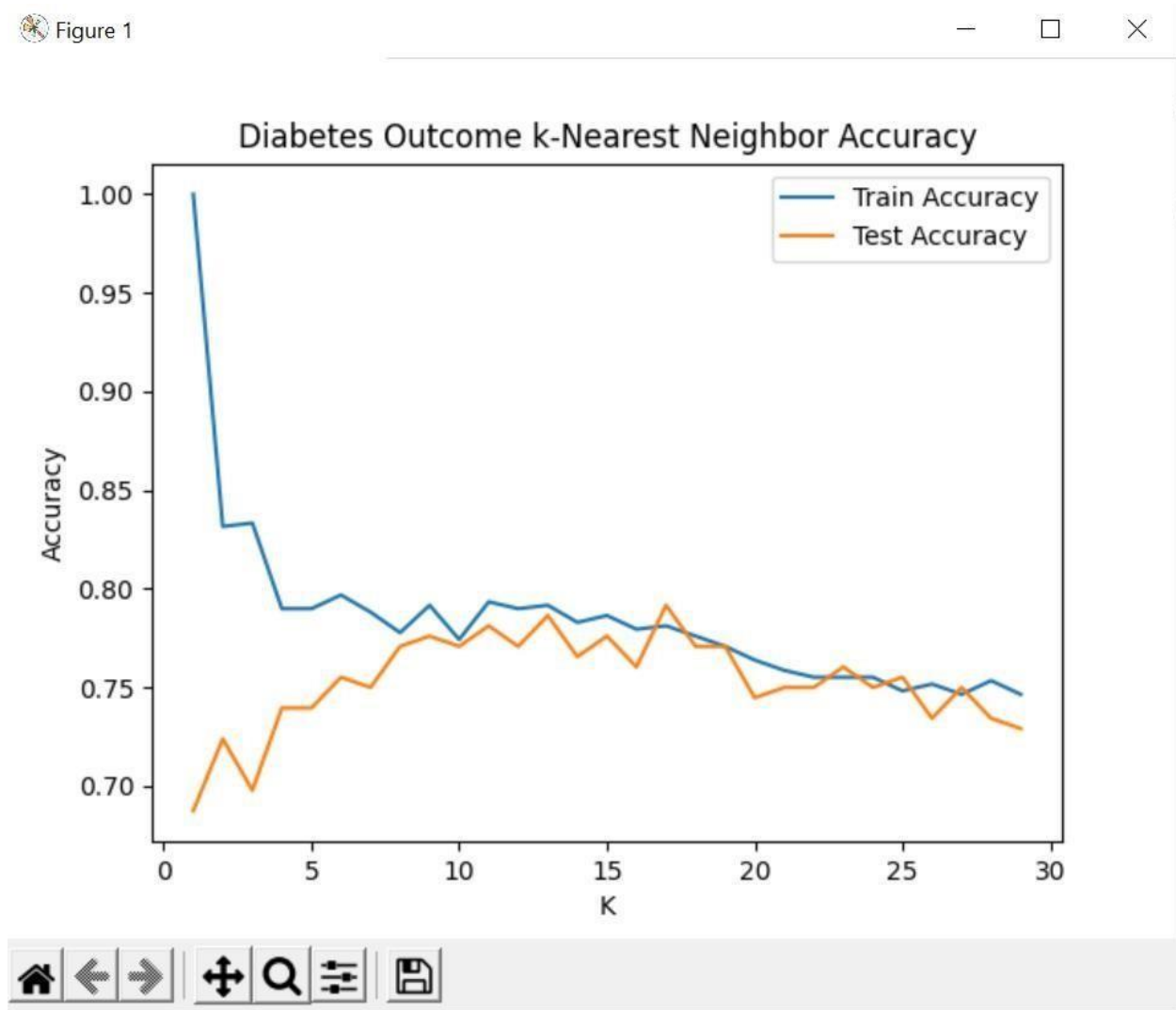
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier

dataset = pd.read_csv('Diabetes.csv')

X_train, X_test, y_train, y_test = train_test_split(dataset.loc[:, dataset.columns !=
'Outcome'], dataset['Outcome'], stratify=dataset['Outcome'],
random_state=66)
train_accuracy = []
test_accuracy = []
for k in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_accuracy.append(knn.score(X_train, y_train))
    test_accuracy.append(knn.score(X_test, y_test))
plt.plot(k, train_accuracy, label="Train Accuracy")
plt.plot(k, test_accuracy, label="Test Accuracy")
plt.title('Diabetes Outcome k-Nearest Neighbor Accuracy')
```

```
plt.ylabel("Accuracy")    plt.xlabel("K")    plt.legend()
plt.show()
```

OUTPUT



b) K-MEANS

```
import numpy as np import seaborn as sns import matplotlib.pyplot as plt import
pandas as pd from sklearn.model_selection import train_test_split from
sklearn.naive_bayes import GaussianNB from sklearn import metrics from
sklearn.metrics import confusion_matrix import sklearn from sklearn.cluster import
```

```

KMeans dataset = pd.read_csv('Diabetes.csv') x = dataset.iloc[:, :-1].values y =
dataset.iloc[:, -1].values x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size = 0.20, random_state = 42, stratify = dataset['Outcome'] )
KMeans_Clustering = KMeans(n_clusters =2, random_state=0)
KMeans_Clustering.fit(x_train)          kpred          =
KMeans_Clustering.predict(x_test)
print('Classification report:\n\n',
sklearn.metrics.classification_report(y_test,kpred))
print("Confusion Matrix :") outcome_labels =
sorted(dataset.Outcome.unique())
print(confusion_matrix(y_test, kpred))
KMeans_Clustering      =      KMeans(n_clusters      =2,      random_state=0)
KMeans_Clustering.fit(x)

plt.scatter(dataset.iloc[:, [1]].values,dataset.iloc[:, [5]].values,
c=KMeans_Clustering.labels_, cmap='rainbow') plt.show()

```

OUTPUT

```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/HP/Desktop/K MEANS.py =====
Classification report:

              precision    recall  f1-score   support

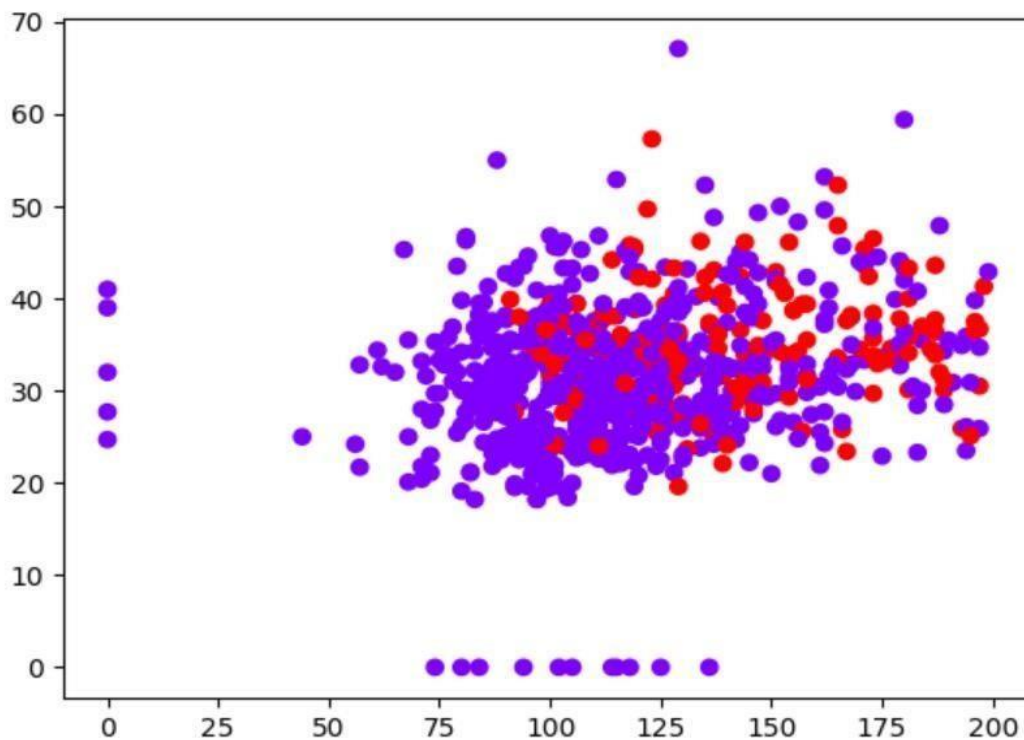
     0       0.66       0.79       0.72       100
     1       0.40       0.26       0.31        54

 accuracy          0.60       154
 macro avg       0.53       0.52       0.52       154
weighted avg       0.57       0.60       0.58       154

Confusion Matrix :
[[79 21]
 [40 14]]
>>> |
```



Figure 1



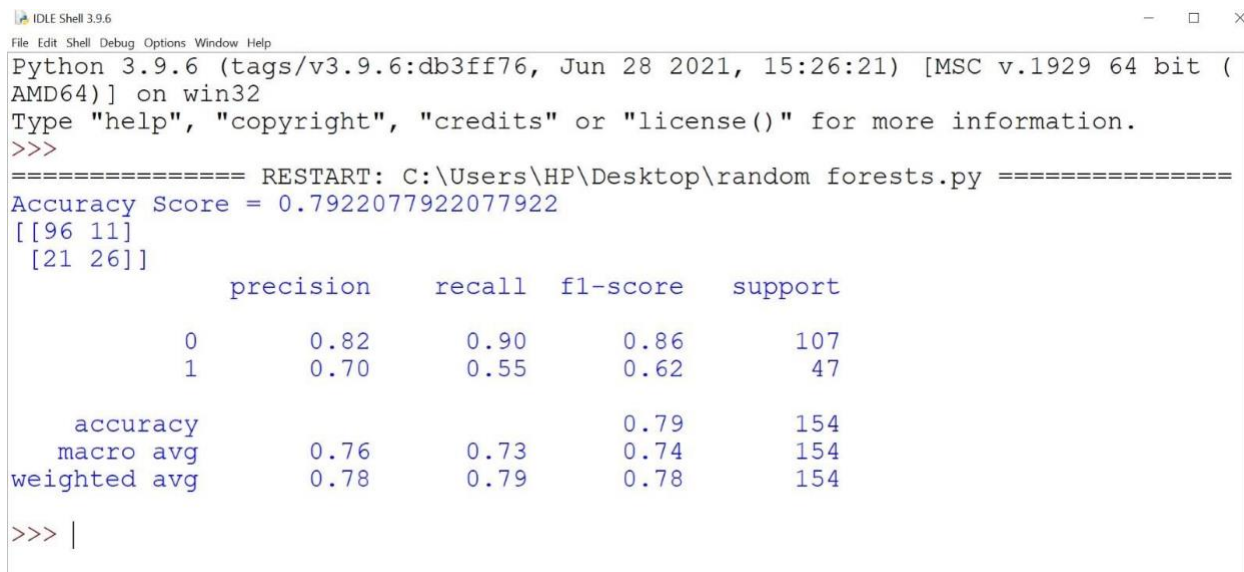
EXPERIMENT NO-8

AIM: Deploy Support Vector Machine, Apriori algorithm a)

Support Vector Machine

```
import numpy as np import matplotlib.pyplot as plt import pandas as pd from
sklearn.model_selection import train_test_split from sklearn.svm import SVC from
sklearn import metrics from sklearn.metrics import confusion_matrix dataset =
pd.read_csv('Diabetes.csv') x = dataset.iloc[:, :-1].values y = dataset.iloc[:, -
1].values x_train, x_test, y_train, y_test = train_test_split(x,
y,test_size=0.2,random_state=0) from sklearn.preprocessing import StandardScaler
scaling_x=StandardScaler() x_train=scaling_x.fit_transform(x_train)
x_test=scaling_x.transform(x_test) svc_model = SVC() svc_model.fit(x_train,
y_train) svc_pred = svc_model.predict(x_test) print("Accuracy Score =",
format(metrics.accuracy_score(y_test, svc_pred))) from sklearn.metrics import
confusion_matrix,classification_report print(confusion_matrix(y_test, svc_pred))
print(classification_report(y_test,svc_pred))
```

OUTPUT



```

Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\HP\Desktop\random forests.py =====
Accuracy Score = 0.7922077922077922
[[96 11]
 [21 26]]

```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.82 | 0.90 | 0.86 | 107 |
| 1 | 0.70 | 0.55 | 0.62 | 47 |
| accuracy | | | 0.79 | 154 |
| macro avg | 0.76 | 0.73 | 0.74 | 154 |
| weighted avg | 0.78 | 0.79 | 0.78 | 154 |

```

>>> |

```

b) Apriori algorithm

```

import numpy as np
import pandas as pd
from apyori import apriori
dataset = pd.read_csv('C:/Users/HP/Downloads/Groceries_dataset.csv')
print(dataset.shape) #print(dataset.head())
transactions = []
for i in range(0,38765):
    transactions.append([str(dataset.values[i,j]) for j in range(0,3)])
rules = apriori(transactions, min_support = 0.00030, min_confidence = 0.05,
min_lift = 3, max_length = 2, target = "rules")
association_results = list(rules)
print(association_results[0])
for item in association_results:
    pair = item[0]
    items = [x for x in pair]
    print("Rule : ", items[0], " -> " + items[1])
    print("Support : ", str(item[1]))
    print("Confidence : ",str(item[2][0][2]))
    print("Lift : ", str(item[2][0][3]))
    print("=====")

```

OUTPUT

```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\HP\Desktop\apriori.py =====
(38765, 3)
RelationRecord(items=frozenset({'whole milk', '28-11-2015'}), support=0.0003353540564942603, ordered_statistics=[OrderedStatistic(items_base=frozenset({'28-11-2015'}), items_add=frozenset({'whole milk'}), confidence=0.21666666666666666, lift=3.356947775113243)])
>>> |
```