1. Linear Regression – 1

1.1 Logistic Regression - 9

2. Decision Tree – 13
3. KNN – 20
4. Kmeans – 22
5. SVM – 25
6. Heirarchical – 28
7. MLP – 30
8. Adaboost – 40
9. Random Forest – 41
10. GMM - 43

Linear Regression

1. Consider a dataset from UCI repository. a. Create a Simple Linear Regression model using the training data set. b. Predict the scores on the test data and output RMSE and R Squared score. c. Include appropriate code snippets to visualize the model.

DATASET USED:

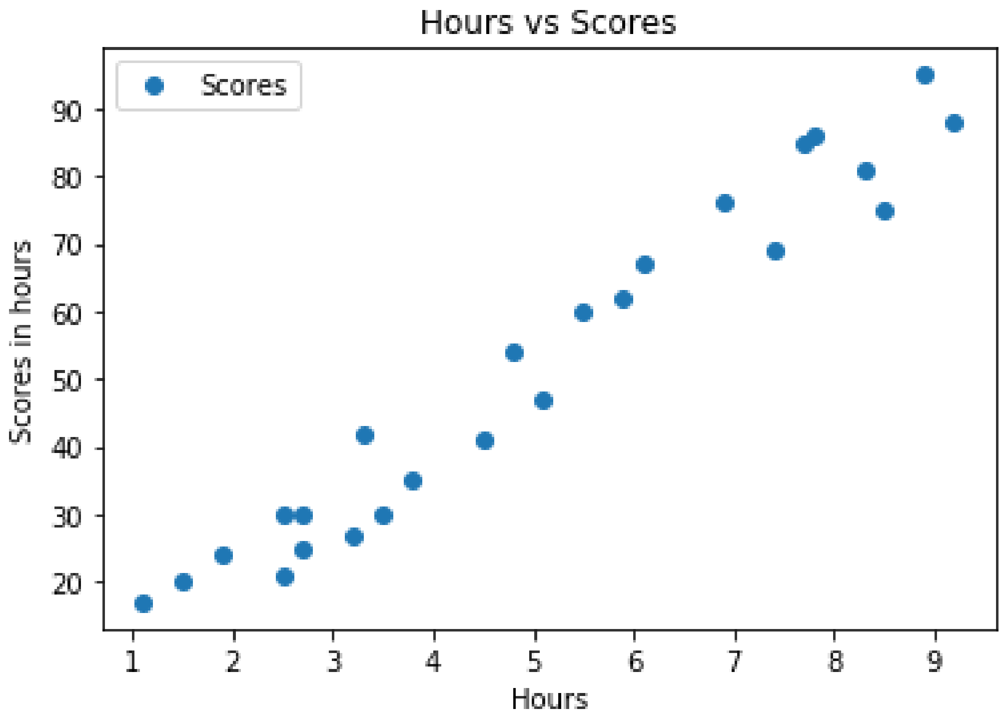
Hours	Scores
2.5	21
5.1	47
3.2	27
8.5	75
3.5	30
1.5	20
9.2	88
5.5	60
8.3	81
2.7	25
7.7	85
5.9	62
4.5	41
3.3	42
1.1	17
8.9	95
2.5	30

1.9	24
6.1	67
7.4	69
2.7	30
4.8	54
3.8	35
6.9	76
7.8	86

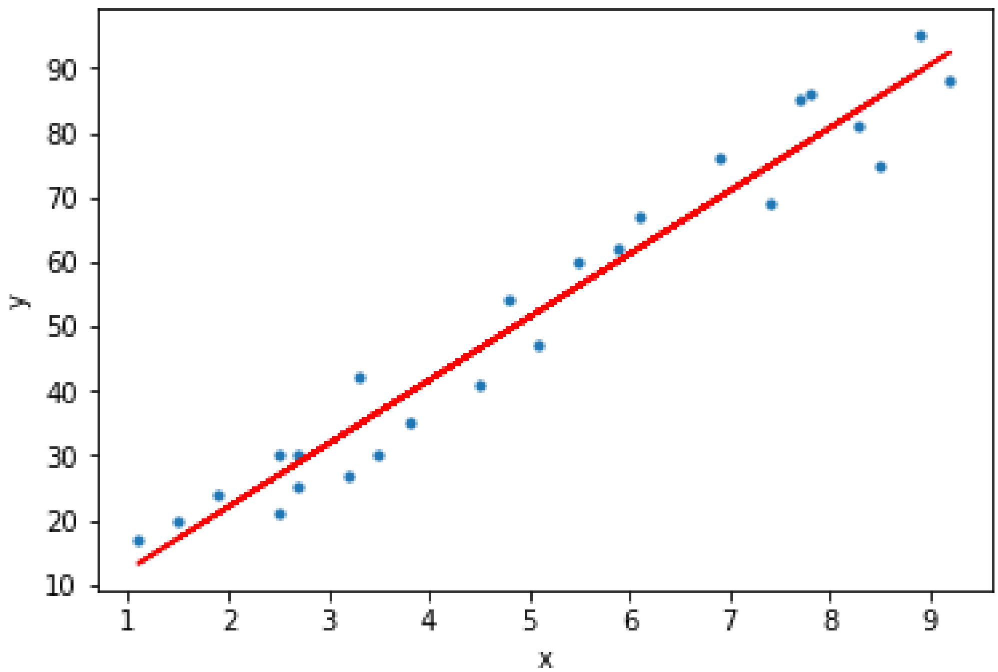
PYTHON PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,  r2_score
scores = pd.read_csv('D:/Nikhil/Documents/scores.csv')
scores.plot(x='Hours',y='Scores',style='o')
plt.title('Hours vs Scores')
plt.xlabel('Hours')
plt.ylabel('Scores in hours')
plt.show()
x=scores.iloc[:,:-1].values
y=scores.iloc[:,1].values
regressionModel = LinearRegression()
regressionModel.fit(x,y)
y_predicted=regressionModel.predict(x)
print(y_predicted)

rmse=mean_squared_error(y,y_predicted)
r2=r2_score(y,y_predicted)

print('Slope',regressionModel.coef_)
print('Intercept:',regressionModel.intercept_)
print('Root mean square error',rmse)
print('R2 score:',r2)

plt.scatter(x,y,s=10)
plt.xlabel('x')
plt.ylabel('y')

plt.plot(x,y_predicted,color='r')
plt.show()
```

OUTPUT:



**[26.92318188 52.3402707 33.76624426 85.57800223 36.69898527 17.14737849
 92.4210646 56.25059205 83.62284155 28.87834256 77.75735951 60.16091341
 46.47478866 34.74382459 13.23705714 89.48832358 26.92318188 21.05769985
 62.11607409 74.8246185 28.87834256 49.40752968 39.63172629 69.9367168
 78.73493985]**
Slope [9.77580339]
Intercept: 2.48367340537321
Root mean square error 28.882730509245466
R2 score: 0.9529481969048356



2. Implement Multiple Linear Regression using a dataset from UCI repository.

DATASET USED -

Year	Month	Interest_Rate	Unemployment_Rate	Stock_Index_Price
2017	12	2.75	5.3	1464
2017	11	2.5	5.3	1394
2017	10	2.5	5.3	1357
2017	9	2.5	5.3	1293
2017	8	2.5	5.4	1256
2017	7	2.5	5.6	1254
2017	6	2.5	5.5	1234
2017	5	2.25	5.5	1195
2017	4	2.25	5.5	1159
2017	3	2.25	5.6	1167
2017	2	2	5.7	1130
2017	1	2	5.9	1075
2016	12	2	6	1047
2016	11	1.75	5.9	965
2016	10	1.75	5.8	943
2016	9	1.75	6.1	958
2016	8	1.75	6.2	971
2016	7	1.75	6.1	949
2016	6	1.75	6.1	884
2016	5	1.75	6.1	866
2016	4	1.75	5.9	876

2016	3	1.75	6.2	822
2016	2	1.75	6.2	704
2016	1	1.75	6.1	719

PYTHON PROGRAM:

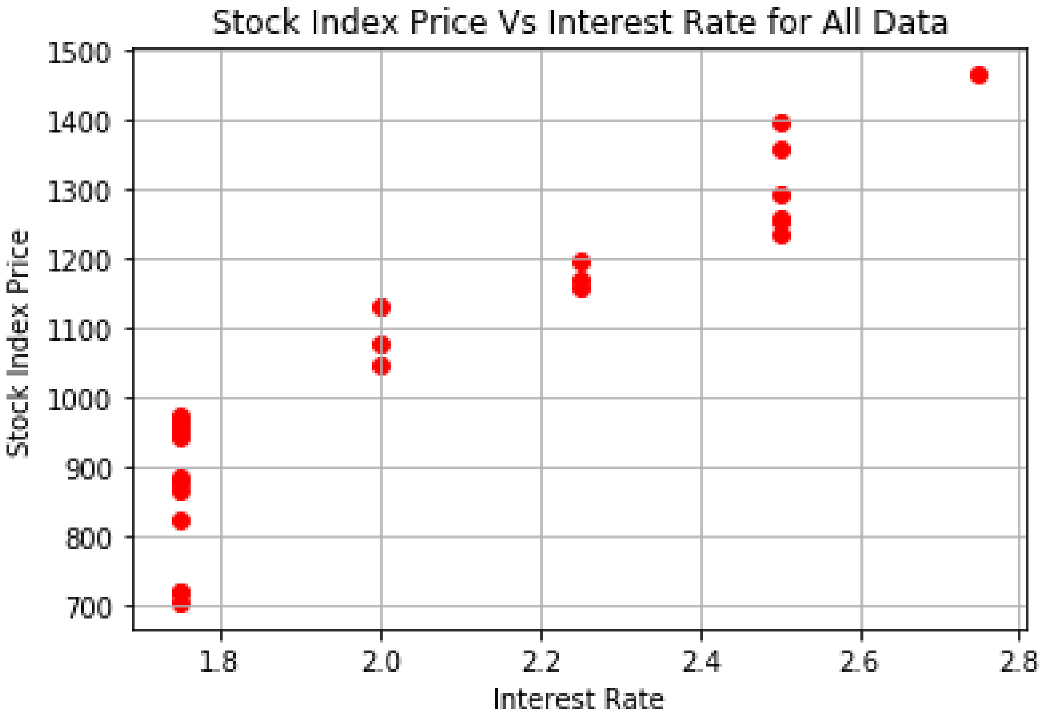
```
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import mean_squared_error,r2_score
import numpy as np
import pandas as pd

stock = pd.read_csv("D:/Nikhil/Documents/economy.csv")
df = pd.DataFrame(stock)
df.isnull().any()
df = df.fillna(method='ffill')
print(df)
Y = df['Stock_Index_Price']
X = df['Interest_Rate']

X=X.values.reshape(-1,1)
Y=Y.values.reshape(-1,1)

plt.scatter(X,Y,color='red')
plt.title('Stock Index Price Vs Interest Rate for All Data')
plt.xlabel('Interest Rate')
plt.ylabel('Stock Index Price')
plt.grid(True)
plt.show()

# Split the data into training/testing sets
X_train = X[0:18]
X_test = X[18:24]

# Split the targets into training/testing sets
Y_train = Y[0:18]
Y_test = Y[18:24]

# Plot outputs
plt.scatter(X_test,Y_test,color='red')
plt.title('Stock Index Price Vs Interest Rate for Test Data')
plt.xlabel('Interest Rate')
plt.ylabel('Stock Index Price')
plt.grid(True)

# Create linear regression object
regr = linear_model.LinearRegression()
# Train the model using the training sets
```

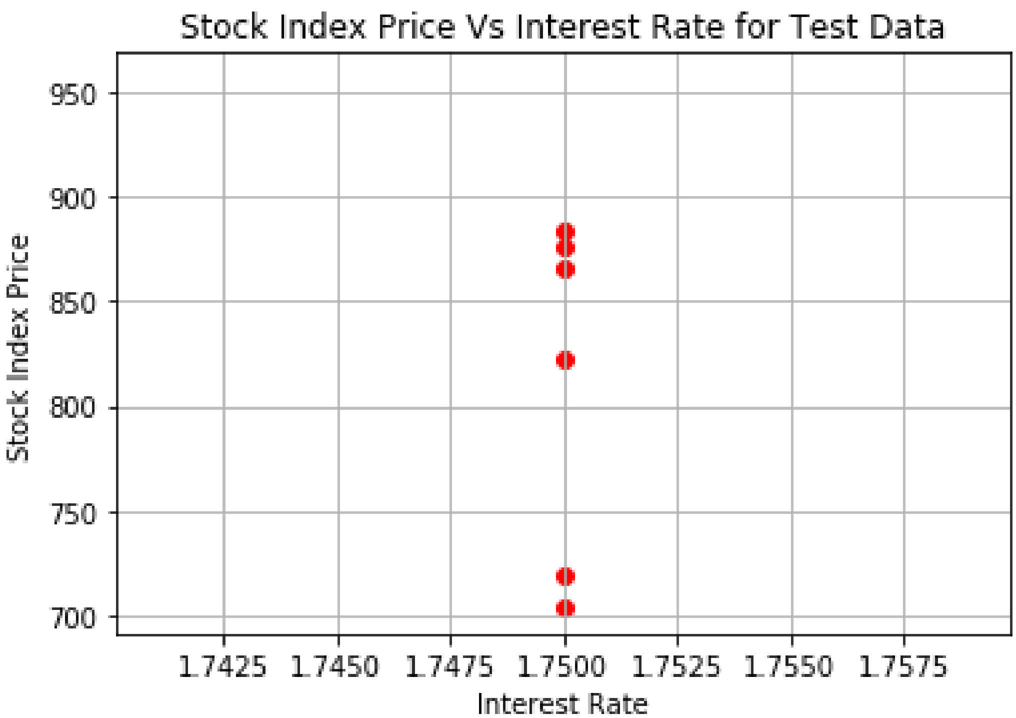
```
regr.fit(X_train,Y_train)

# Plot outputs
plt.plot(X_test, regr.predict(X_test), color='red',linewidth=3)
plt.show()

Y_predicted=regr.predict(X)
print(Y_predicted)

rmse=mean_squared_error(Y,Y_predicted)
r2=r2_score(Y,Y_predicted)

print('Slope',regr.coef_)
print('Intercept:',regr.intercept_)
print('Root mean square error:',rmse)
print('R2 score:',r2)
```

OUTPUT:

DATAFRAME -

	Year	Month	...	Unemployment_Rate	Stock_Index_Price
0	2017.0	12.0	...	5.3	1464.0
1	2017.0	11.0	...	5.3	1394.0
2	2017.0	10.0	...	5.3	1357.0
3	2017.0	9.0	...	5.3	1293.0
4	2017.0	8.0	...	5.4	1256.0
5	2017.0	7.0	...	5.6	1254.0
6	2017.0	6.0	...	5.5	1234.0
7	2017.0	5.0	...	5.5	1195.0
8	2017.0	4.0	...	5.5	1159.0
9	2017.0	3.0	...	5.6	1167.0
10	2017.0	2.0	...	5.7	1130.0
11	2017.0	1.0	...	5.9	1075.0
12	2016.0	12.0	...	6.0	1047.0
13	2016.0	11.0	...	5.9	965.0
14	2016.0	10.0	...	5.8	943.0
15	2016.0	9.0	...	6.1	958.0
16	2016.0	8.0	...	6.2	971.0
17	2016.0	7.0	...	6.1	949.0
18	2016.0	6.0	...	6.1	884.0
19	2016.0	5.0	...	6.1	866.0
20	2016.0	4.0	...	5.9	876.0
21	2016.0	3.0	...	6.2	822.0
22	2016.0	2.0	...	6.2	704.0
23	2016.0	1.0	...	6.1	719.0
24	2016.0	1.0	...	6.1	719.0

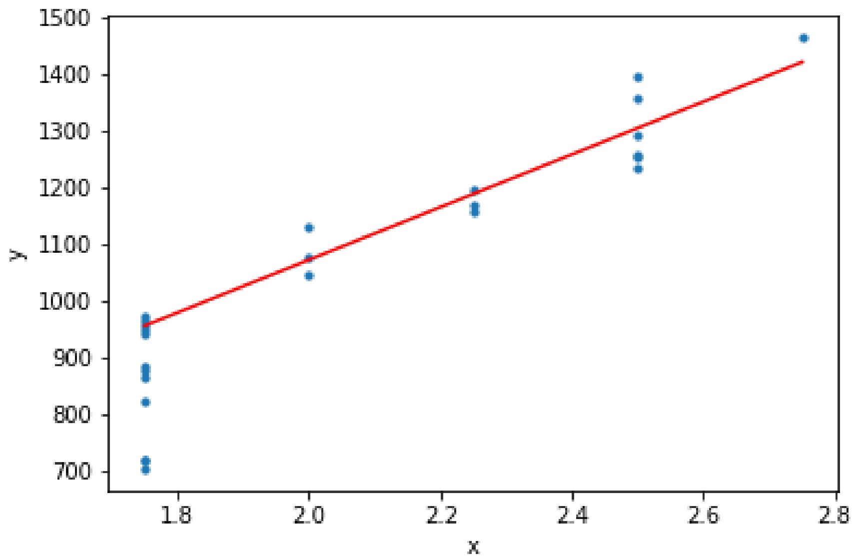




Y_Predicted :
[1420.8172232]
[1304.62917399]

[1304.62917399]
[1304.62917399]
[1304.62917399]
[1304.62917399]
[1304.62917399]
[1188.44112478]
[1188.44112478]
[1188.44112478]
[1072.25307557]
[1072.25307557]
[1072.25307557]
[956.06502636]
[956.06502636]
[956.06502636]
[956.06502636]
[956.06502636]
[956.06502636]
[956.06502636]
[956.06502636]
[956.06502636]
[956.06502636]
[956.06502636]
[956.06502636]]

Slope [[464.75219684]]
Intercept: [142.7486819]
Root mean square error: 9685.940305225162
R2 score: 0.7875414973270607



3. Implement logistic regression and test it using any dataset of your choice from UCI repository. The output should include Confusion Matrix, Accuracy, Error rate, Precision, Recall and F Measure.

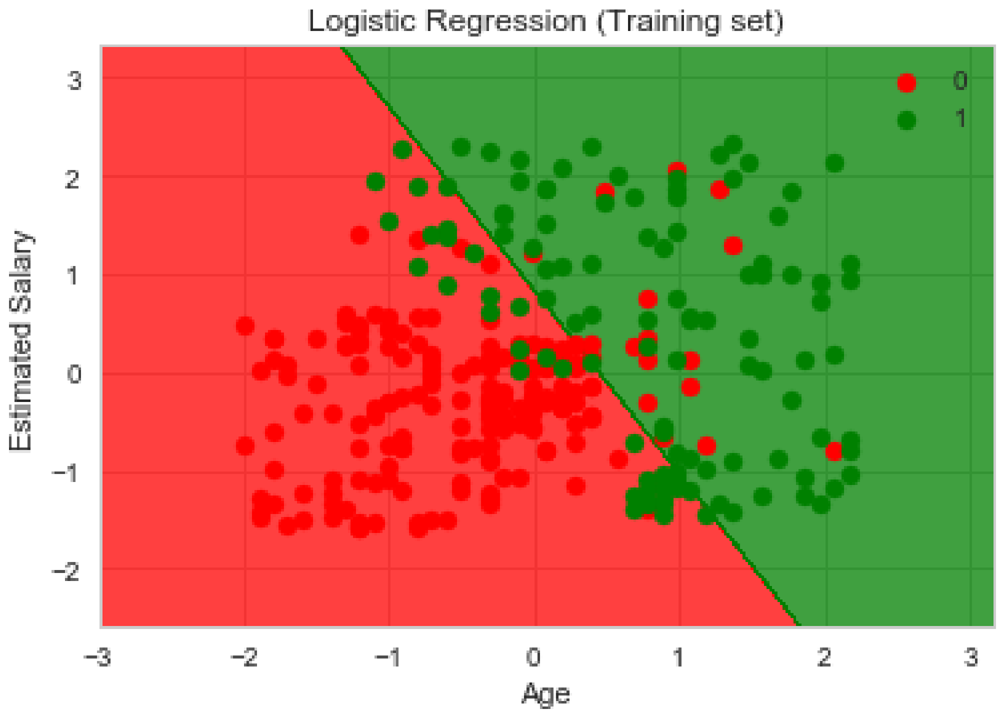
Code:

```
# Importing the libraries
```

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


# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Visualising the Training set results
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 = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max()
+ 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
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'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
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 = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max()
+ 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
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'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()


#Confusion matrix
from sklearn.metrics import confusion_matrix
conf_matrics=confusion_matrix(y_test, y_pred)
print("Confusion Matrics===>")
print(conf_matrics)
print()

#Accuracy and error rate
from sklearn import metrics
accuracy = metrics.accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
print("Accuracy = {}".format(accuracy))
print("Error Rate = {}".format(error_rate))
print()
#classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Output:



Confusion Matrics===>
[[65 3]
[8 24]]

Accuracy = 0.89
Error Rate = 0.10999999999999999

precision   recall f1-score   support

0	0.89	0.96	0.92	68
1	0.89	0.75	0.81	32
avg / total	**0.89**	**0.89**	**0.89**	**100**

Decision Tree

The Dataset:

	A	B	C	D	E	F	G
1	5.1	3.5	1.4	0.2	Iris-setosa		
2	4.9	3	1.4	0.2	Iris-setosa		
3	4.7	3.2	1.3	0.2	Iris-setosa		
4	4.6	3.1	1.5	0.2	Iris-setosa		
5	5	3.6	1.4	0.2	Iris-setosa		
6	5.4	3.9	1.7	0.4	Iris-setosa		
7	4.6	3.4	1.4	0.3	Iris-setosa		
8	5	3.4	1.5	0.2	Iris-setosa		
9	4.4	2.9	1.4	0.2	Iris-setosa		
10	4.9	3.1	1.5	0.1	Iris-setosa		
11	5.4	3.7	1.5	0.2	Iris-setosa		
12	4.8	3.4	1.6	0.2	Iris-setosa		
13	4.8	3	1.4	0.1	Iris-setosa		
14	4.3	3	1.1	0.1	Iris-setosa		
15	5.8	4	1.2	0.2	Iris-setosa		
16	5.7	4.4	1.5	0.4	Iris-setosa		
17	5.4	3.9	1.3	0.4	Iris-setosa		

ID3 Algorithm

The Code:

```
# Importing the libraries import
numpy as np import
matplotlib.pyplot as plt import
pandas as pd
```

```
# Importing the dataset dataset =
pd.read_csv('data.csv') X =
dataset.iloc[:, [2, 3]].values y =
dataset.iloc[:, 4].values

#Converting String to Charaterized data

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Encoding the Dependent Variable
labelencoder_y = LabelEncoder() y =
labelencoder_y.fit_transform(y)

# Splitting the dataset into the Training set and Test set from
sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler sc
= StandardScaler()
X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results y_pred =
classifier.predict(X_test)

# Making the Confusion Matrix from
sklearn.metrics import confusion_matrix cm =
confusion_matrix(y_test, y_pred)

# Visualising the Training set results
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 = 0.01),                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
step = 0.01)) plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
```

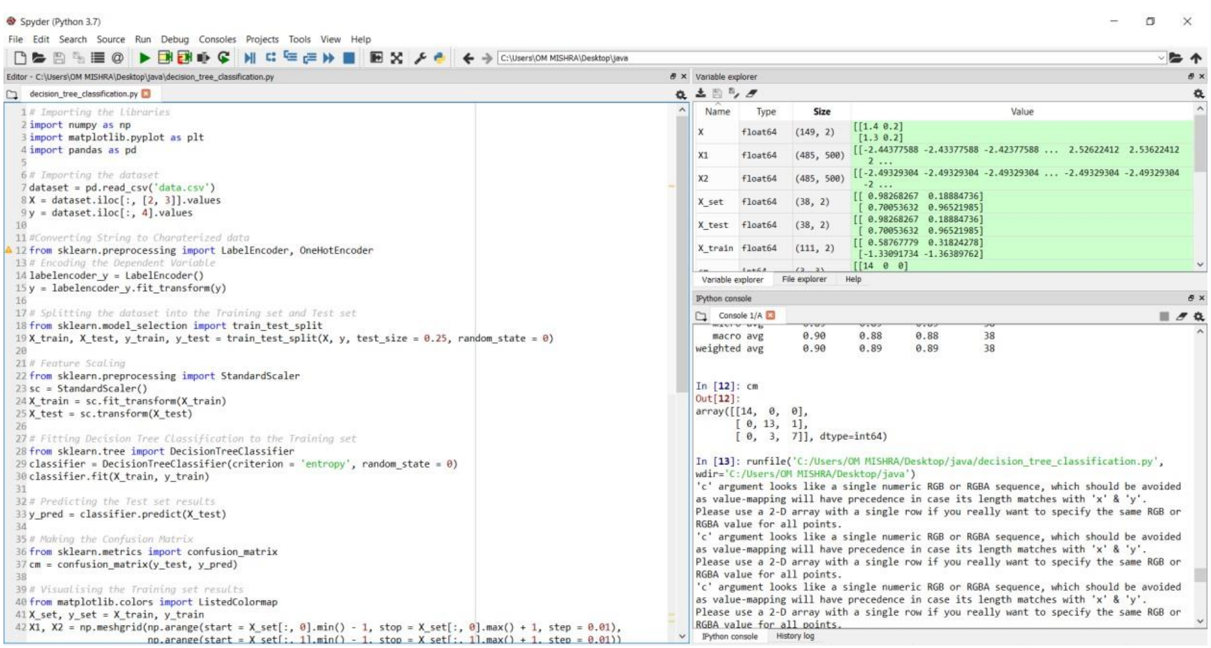
X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green','orange'))) plt.xlim(X1.min(), X1.max()) plt.ylim(X2.min(), X2.max()) for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
c = ListedColormap(('red', 'green','orange'))(i), label = j)
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Petal Length') plt.ylabel('Petal Width') plt.legend()
plt.show()

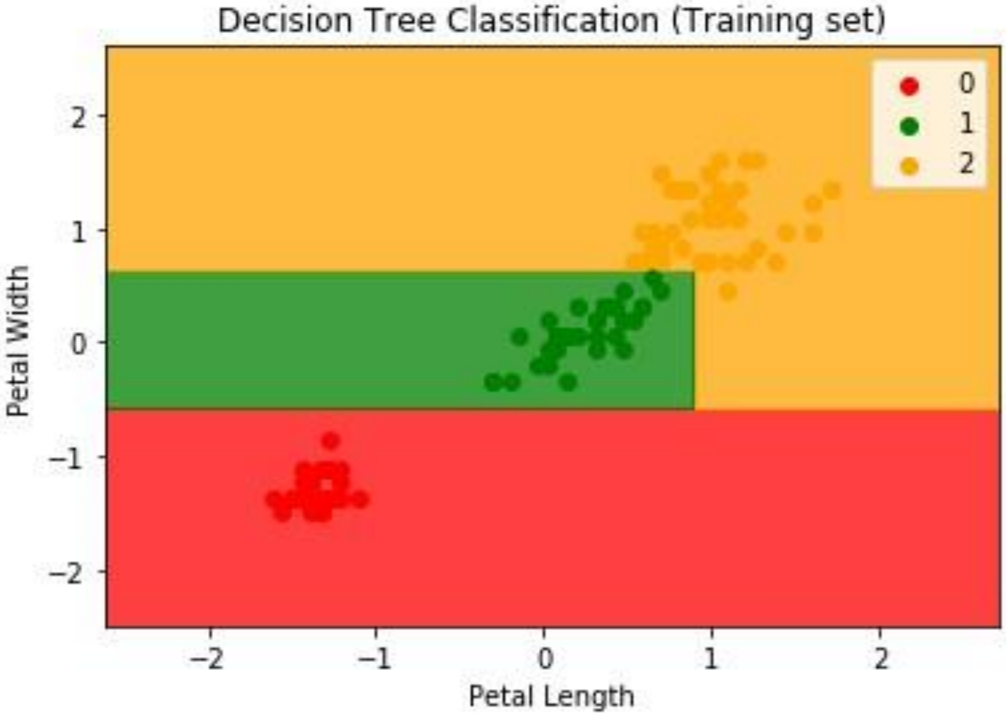
Visualising the Test set results from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test

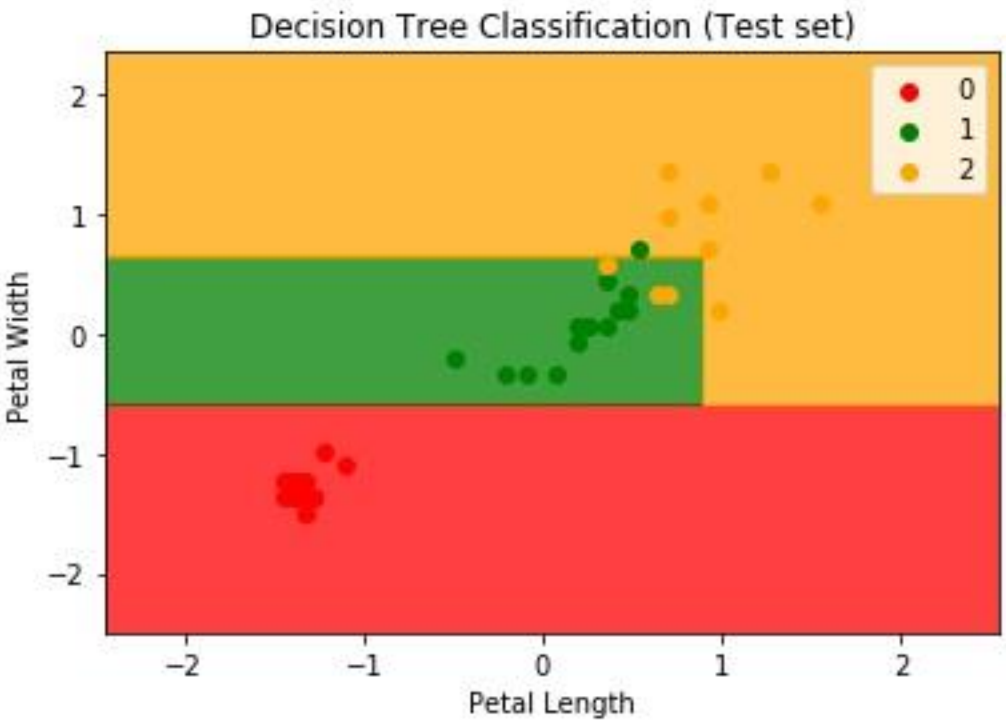
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01), np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01)) plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green','orange'))) 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','orange'))(i), label = j)
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Petal Length') plt.ylabel('Petal Width') plt.legend()
plt.show()

from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))

The Output:




Decision Tree Classification (Training set)
0
1
2
Petal Width
Petal Length
−2
−1
0
1
2


Decision Tree Classification (Test set)
0
1
2
Petal Width
Petal Length
−2
−1
0
1
2

	precision	recall	f1-score	support
0	1.00	1.00	1.00	14
1	0.81	0.93	0.87	14
2	0.88	0.70	0.78	10
micro avg	0.89	0.89	0.89	38
macro avg	0.90	0.88	0.88	38
weighted avg	0.90	0.89	0.89	38

```
In [14]: cm
Out[14]:
array([[14,  0,  0],
       [ 0, 13,  1],
       [ 0,  3,  7]], dtype=int64)
```

CART Algorithm

The Code:

Importing the libraries import
numpy as np import
matplotlib.pyplot as plt import
pandas as pd

Importing the dataset dataset =
pd.read_csv('iris.csv') X =
dataset.iloc[:, [2, 3]].values y =
dataset.iloc[:, 4].values

#Converting String to Charaterized data

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Encoding the Dependent Variable
labelencoder_y = LabelEncoder() y =
labelencoder_y.fit_transform(y)

Splitting the dataset into the Training set and Test set from
sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

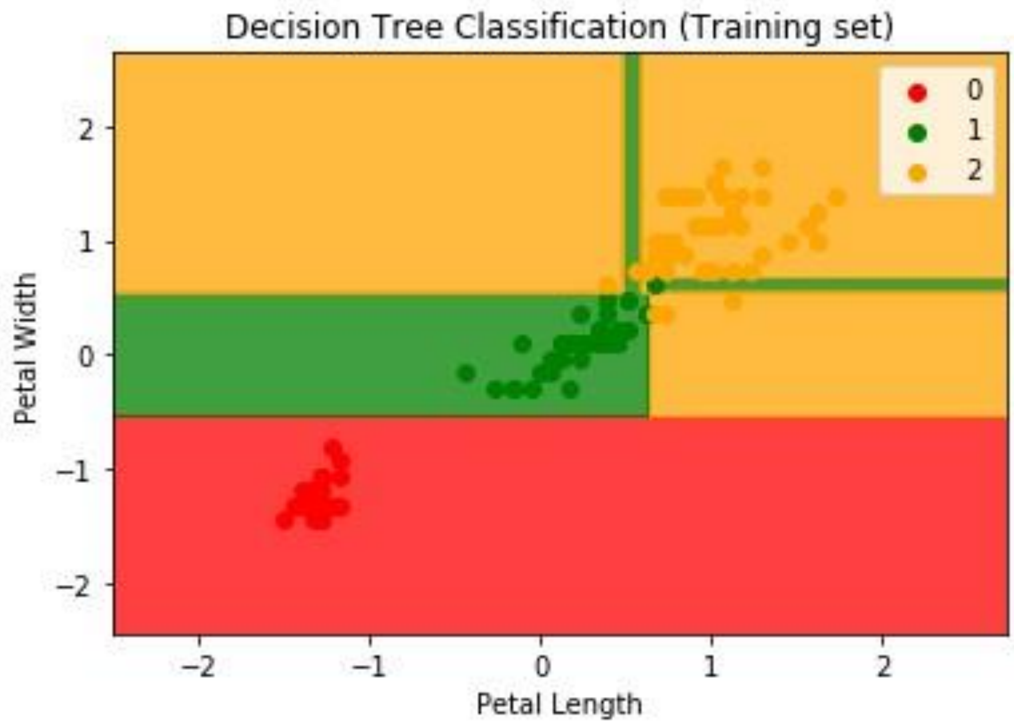
Feature Scaling

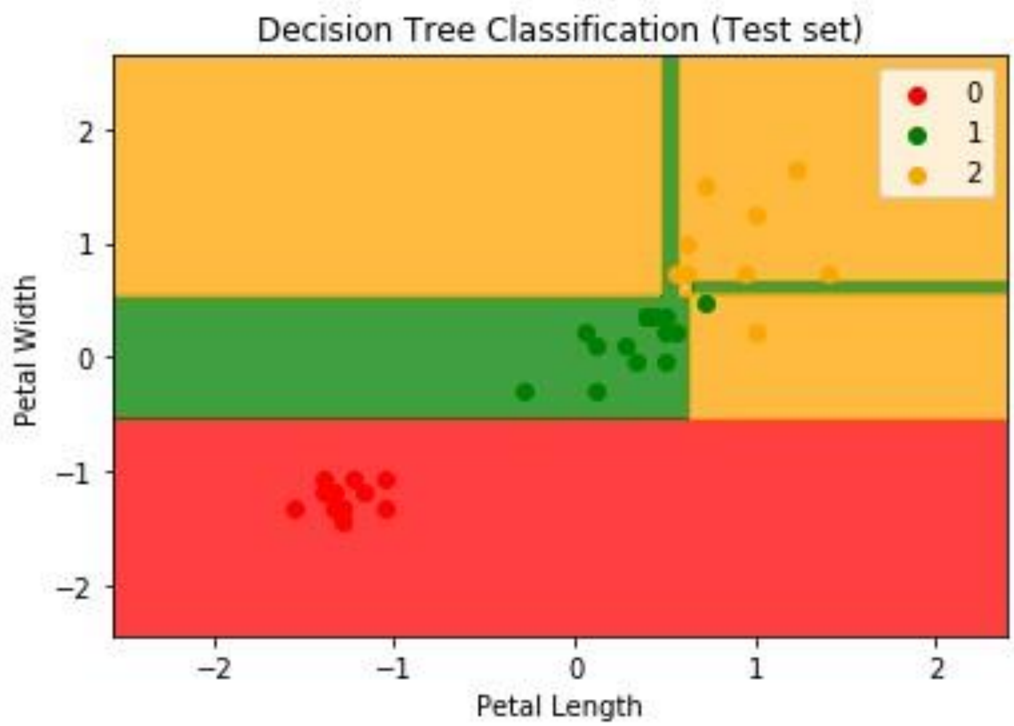
from sklearn.preprocessing import StandardScaler sc
= StandardScaler()

```
X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)


# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)


# Predicting the Test set results y_pred =
classifier.predict(X_test)


# Making the Confusion Matrix from
sklearn.metrics import confusion_matrix cm =
confusion_matrix(y_test, y_pred)


# Visualising the Training set results 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 = 0.01),                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
step = 0.01)) plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),             alpha = 0.75, cmap = ListedColormap(('red',
'green','orange'))) 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','orange'))(i), label = j)
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Petal Length') plt.ylabel('Petal Width') plt.legend()
plt.show()


# Visualising the Test set results 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 = 0.01),                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
step = 0.01)) plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),             alpha = 0.75, cmap = ListedColormap(('red',
'green','orange'))) 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','orange'))(i), label = j)
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Petal Length') plt.ylabel('Petal Width') plt.legend()
plt.show()
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

The Output:






```
Out[10]:
array([[13,  0,  0],
       [ 0, 15,  1],
       [ 0,  1,  8]], dtype=int64)
```

Knn

1. Implement k-Nearest Neighbor algorithm for classifying a dataset.

Dataset Used: Iris Dataset
The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

The Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import classification_report,confusion_matrix
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petallength','petal-width', 'Class']
dataset = pd.read_csv(url, names=names)
dataset.head()
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.20)
from sklearn.preprocessing import StandardScaler
scaler =StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier
for i in range(1,6):
    print('for k = ',i)
    classifier = KNeighborsClassifier(n_neighbors=i)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    print(confusion_matrix(y_test,y_pred))
    print(classification_report(y_test, y_pred))
```

The Output:

```
py3")
for k =  1
[[11  0  0]
 [ 0  7  0]
 [ 0  0 12]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00         7
 Iris-virginica       1.00      1.00      1.00        12

    avg / total       1.00      1.00      1.00        30

for k =  2
[[11  0  0]
 [ 0  7  0]
 [ 0  0 12]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00         7
 Iris-virginica       1.00      1.00      1.00        12

    avg / total       1.00      1.00      1.00        30

for k =  3
[[11  0  0]
 [ 0  7  0]
 [ 0  0 12]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00         7
 Iris-virginica       1.00      1.00      1.00        12

    avg / total       1.00      1.00      1.00        30


for k =  4
[[11  0  0]
 [ 0  7  0]
 [ 0  1 11]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.88      1.00      0.93         7
 Iris-virginica       1.00      0.92      0.96        12

    avg / total       0.97      0.97      0.97        30

for k =  5
[[11  0  0]
 [ 0  7  0]
 [ 0  1 11]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.88      1.00      0.93         7
 Iris-virginica       1.00      0.92      0.96        12

    avg / total       0.97      0.97      0.97        30
```

Implement Multi-Layer Perceptron using a dataset

K-means:

The Dataset:

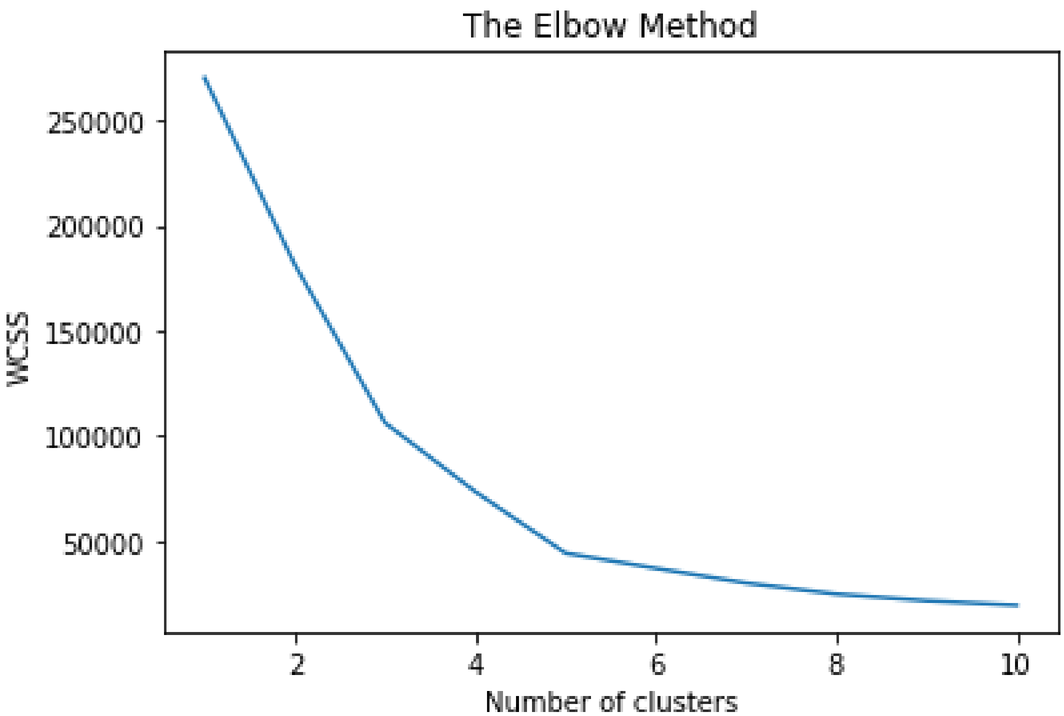
CustomerI	Genre	Age	Annual Inc	Spending Score (1-100)
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40
6	Female	22	17	76
7	Female	35	18	6
8	Female	23	18	94
9	Male	64	19	3
10	Female	30	19	72
11	Male	67	19	14
12	Female	35	19	99
13	Female	58	20	15
14	Female	24	20	77
15	Male	37	20	13
16	Male	22	20	79
17	Female	35	21	35
18	Male	20	21	66
19	Male	52	23	29
20	Female	35	23	98
21	Male	35	24	35
22	Male	25	24	73
23	Female	46	25	5
24	Male	31	25	73
25	Female	54	28	14
26	Male	29	28	82
27	Female	45	28	32

The Code:

```
# K-Means Clustering
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
# y = dataset.iloc[:, 3].values

# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

```
# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label =
'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

The Output:



Therefore Optimum Number of Clusters is 5.

SVM:

The Dataset:

User ID	Gender	Age	Estimated	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1
15714658	Male	48	41000	1
15599081	Female	45	22000	1
15705113	Male	46	23000	1
15631159	Male	47	20000	1

The Code:

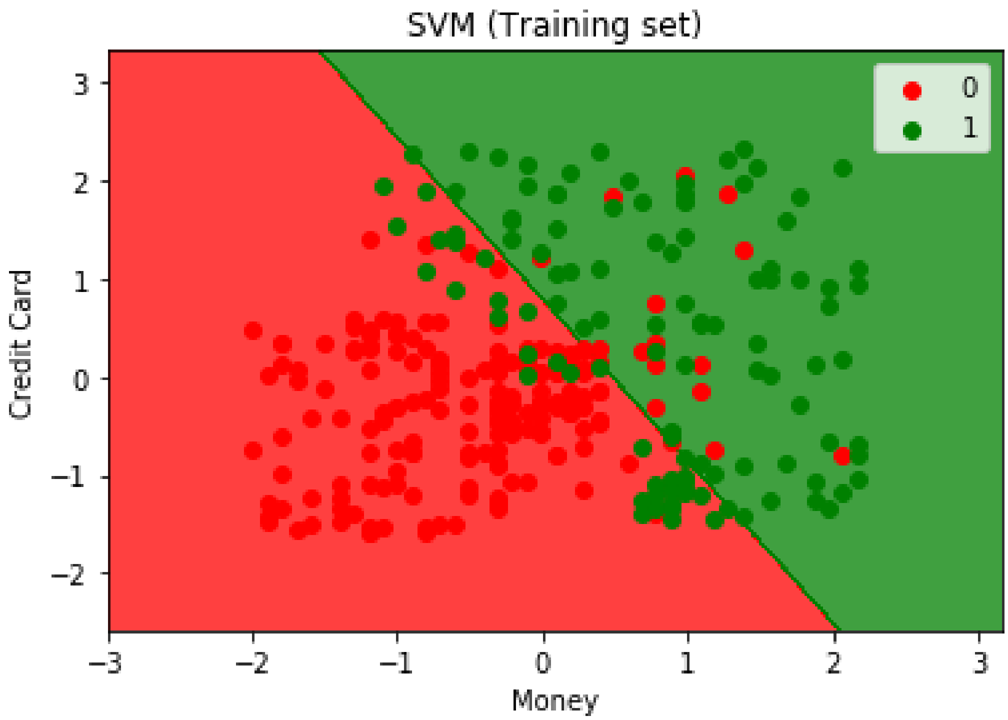
Support Vector Machine (SVM)

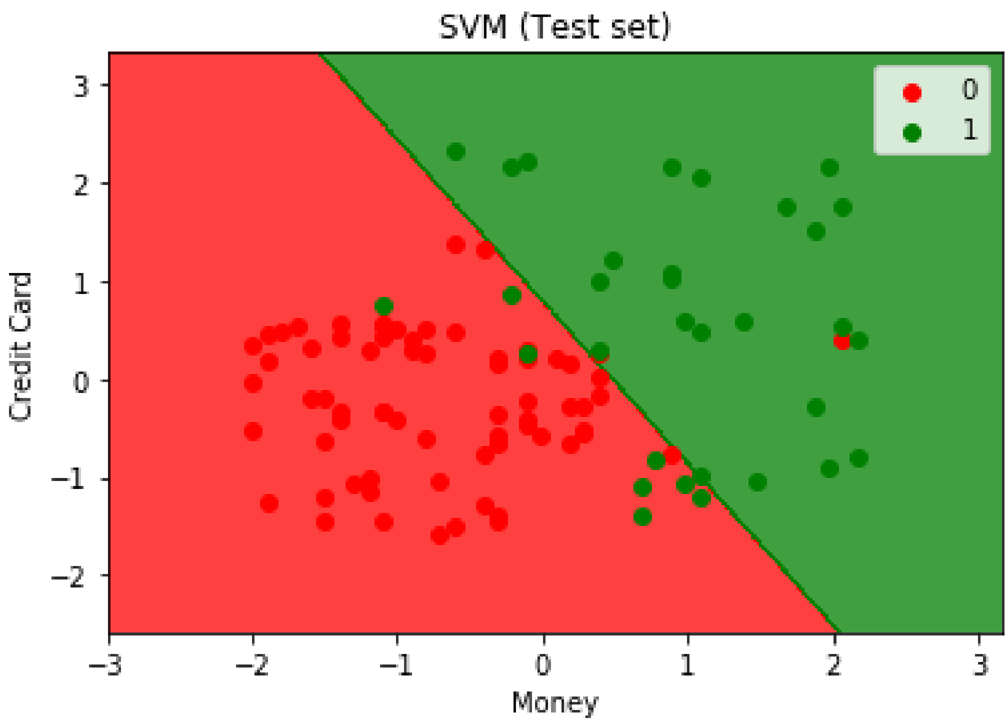
Importing the libraries

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

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Visualising the Training set results
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 = 
0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
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'))(i), label = j)
plt.title('SVM (Training set)')
plt.xlabel('Money')
plt.ylabel('Credit Card')
plt.legend()
plt.show()
```

```
# Visualising the Test set results
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 =
0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
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'))(i), label = j)
plt.title('SVM (Test set)')
plt.xlabel('Money')
plt.ylabel('Credit Card')
plt.legend()
plt.show()
```

The Output:






```
In [2]: cm
Out[2]:
array([[66,  2],
       [ 8, 24]], dtype=int64)
```

Hierarchical clustering:

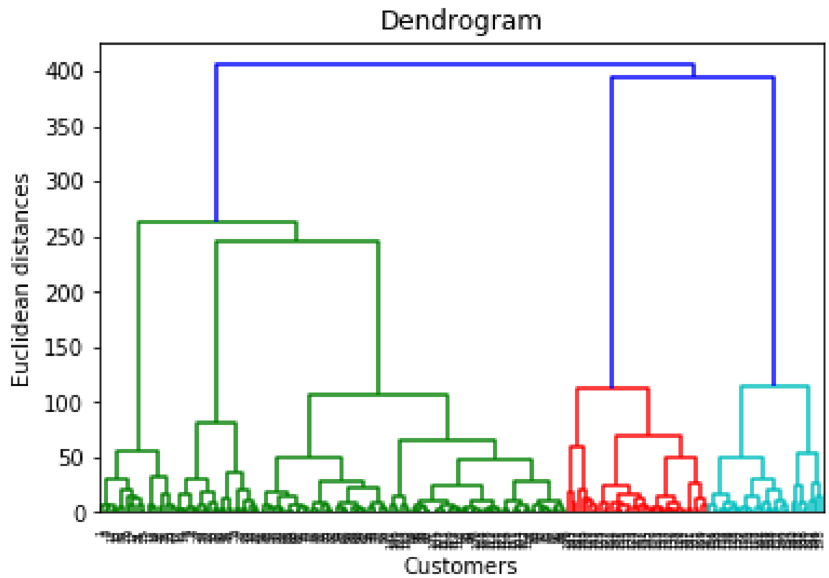
The Dataset:

CustomerI	Genre	Age	Annual Inc	Spending Score (1-100)
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40
6	Female	22	17	76
7	Female	35	18	6
8	Female	23	18	94
9	Male	64	19	3
10	Female	30	19	72
11	Male	67	19	14
12	Female	35	19	99
13	Female	58	20	15
14	Female	24	20	77
15	Male	37	20	13
16	Male	22	20	79
17	Female	35	21	35
18	Male	20	21	66
19	Male	52	23	29
20	Female	35	23	98
21	Male	35	24	35
22	Male	25	24	73
23	Female	46	25	5
24	Male	31	25	73
25	Female	54	28	14
26	Male	29	28	82
27	Female	45	28	32

The Code:

```
# Hierarchical Clustering

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
# y = dataset.iloc[:, 3].values

# Using the dendrogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()

# Fitting Hierarchical Clustering to the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)

# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

The Output:

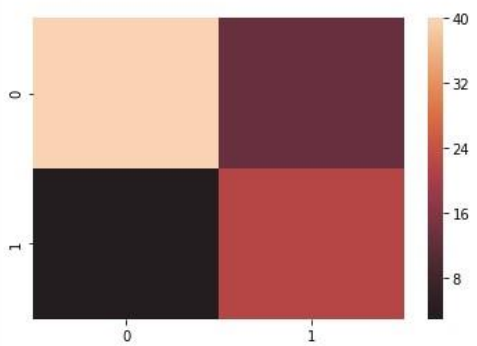


MLP
import numpy as np # linear algebra import pandas as pd # data
processing import matplotlib.pyplot as plt import seaborn as sns df =
pd.read_csv('Dataset_spine.csv') df = df.drop(['Unnamed: 13'], axis=1)
df.head() df.describe()
df = df.drop(['Col7','Col8','Col9','Col10','Col11','Col12'], axis=1) df.head() from
sklearn.neural_network import MLPClassifier from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score from sklearn.metrics import confusion_matrix
y = df['Class_att'] x = df.drop(['Class_att'], axis=1)
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=

0.25, random_state=27) clf = MLPClassifier(hidden_layer_sizes=(100,100,100), max_iter=500, alpha=0.0001,solver='sgd', verbose=10,random_state=21,tol=0.000000001) clf.fit(x_train, y_train) y_pred = clf.predict(x_test) print("accuracy : ") accuracy_score(y_test, y_pred) cm = confusion_matrix(y_test, y_pred) print("confusion matrix : ") print(cm) sns.heatmap(cm, center=True) print("heatmap :")
plt.show()

```
In [10]: runfile('/Users/shivanipriya/Documents/CODES/mlp.py', wdir='/Users/shivanipriya/Documer
CODES')
Iteration 1, loss = 4.26874505
Iteration 2, loss = 7.21713939
Iteration 3, loss = 1.55562179
Iteration 4, loss = 0.94416556
Iteration 5, loss = 1.50851327
Iteration 6, loss = 0.50784081
Iteration 7, loss = 0.39918410
Iteration 8, loss = 0.37867398
Iteration 9, loss = 0.37472479
Iteration 10, loss = 0.34567928
Iteration 11, loss = 0.34293793
Iteration 12, loss = 0.33180054
Iteration 13, loss = 0.33138624
Iteration 14, loss = 0.32280653
Iteration 15, loss = 0.32238711
Iteration 16, loss = 0.33164385
Iteration 17, loss = 0.31229600
Iteration 18, loss = 0.30823387
Iteration 19, loss = 0.32317560
Iteration 20, loss = 0.30697614
Iteration 21, loss = 0.30901736
Iteration 22, loss = 0.35327698
Iteration 23, loss = 0.31835192
Iteration 24, loss = 0.30599160
Iteration 25, loss = 0.30315921
Iteration 26, loss = 0.30774646
Iteration 27, loss = 0.30939928
Iteration 28, loss = 0.30205629
Iteration 29, loss = 0.30492366
Iteration 30, loss = 0.30627954
Iteration 31, loss = 0.32245209
Iteration 32, loss = 0.30092367
Iteration 33, loss = 0.32289947
Iteration 34, loss = 0.30300054
```

```
Iteration 102, loss = 0.28888423
Iteration 103, loss = 0.40146253
Iteration 104, loss = 0.28481547
Iteration 105, loss = 0.28301174
Iteration 106, loss = 0.28824478
Iteration 107, loss = 0.28220960
Iteration 108, loss = 0.29126322
Iteration 109, loss = 0.28446019
Training loss did not improve more than tol=0.000000 for 10 consecutive epochs. Stopping.
accuracy :
confusion matrix :
[[40 13]
 [ 3 22]]
heatmap :
```



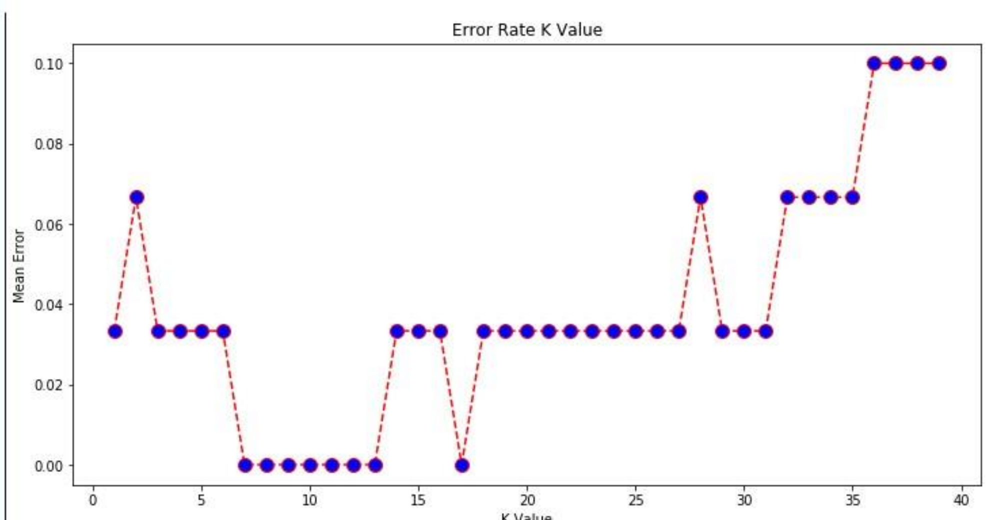

```
In [11]:
```

KNN

CODE import numpy as np import matplotlib.pyplot as plt import pandas as pd from sklearn.model_selection import train_test_split from sklearn.preprocessing import StandardScaler dataset=pd.read_csv('iris.csv') dataset.head()
X=dataset.iloc[:,:-1].values y=dataset.iloc[:,4].values
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20
)
scalar=StandardScaler() scalar.fit(X_train)
X_train=scalar.transform(X_train) X_test=scalar.transform(X_test) from sklearn.neighbors import KNeighbAorsClassifier classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train,y_train) y_pred=classifier.predict(X_test) from sklearn.metrics import classification_report,confusion_matrix print
(confusion_matrix(y_test,y_pred)) print
(classification_report(y_test,y_pred)) error=[] for i in range(1,40):
   knn=KNeighborsClassifier(n_neighbors=i)
   knn.fit(X_train,y_train)   pred_i=knn.predict(X_test)
error.append(np.mean(pred_i!=y_test))
plt.figure(figsize=(12,6))
plt.plot(range(1,40),error,color='red',linestyle='dashed',marker='
o',markerfacecolor='blue',markersize=10)
plt.title('Error Rate K Value') plt.xlabel('K Value')
plt.ylabel('Mean Error')

OUTPUT

```
In [7]: runfile('C:/Users/16BCE0828/.spyder-py3/temp.py', wdir='C:/Users/16BCE0828/.spyder-py3')
[[ 8  0  0]
 [ 0 11  0]
 [ 0  1 10]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       0.92      1.00      0.96        11
 Iris-virginica       1.00      0.91      0.95        11

    avg / total       0.97      0.97      0.97        30
```




```
In [8]:
```

CSE4020 Machine Learning

Lab - 3

Name: S. Mohan Sai

Reg.No: 16BCE0486

Slot. No: L3 + L4

Submitted to: Prof. Vijaysherly .V

1. Implement k-Nearest Neighbour algorithm for classifying a dataset.

Import the dependencies

import numpy as np import

matplotlib.pyplot as plt import pandas

as pd

Extract the dataset from the UCI repository.

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data" names =

['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class'] dataset =

pd.read_csv(url, names=names)

Display the items of the first 5 rows ad split the features and labels dataset.head()

X = dataset.iloc[:, :-1].values y =

dataset.iloc[:, 4].values

Split the train and test data in a ratio of 80-20% respectively.

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20) **Preprocess the Data**

Using standard scalar.

from sklearn.preprocessing import StandardScaler scaler =

StandardScaler() scaler.fit(X_train)

X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)

Import KNN model from sklearn and fit the train dataset from

sklearn.neighbors import KNeighborsClassifier classifier =

KNeighborsClassifier(n_neighbors=5) classifier.fit(X_train, y_train)

Now predict the dataset with the testing data and find the metrics. y_pred =

classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, y_pred))   print(classification_report(y_test,

y_pred))   error = []

Now we are interested in finding the K value of 1 to 40 to which it has the highest accuracy. for i in

range(1, 40):

  knn = KNeighborsClassifier(n_neighbors=i)

knn.fit(X_train, y_train)

  error.append(np.mean(pred_i != y_test))

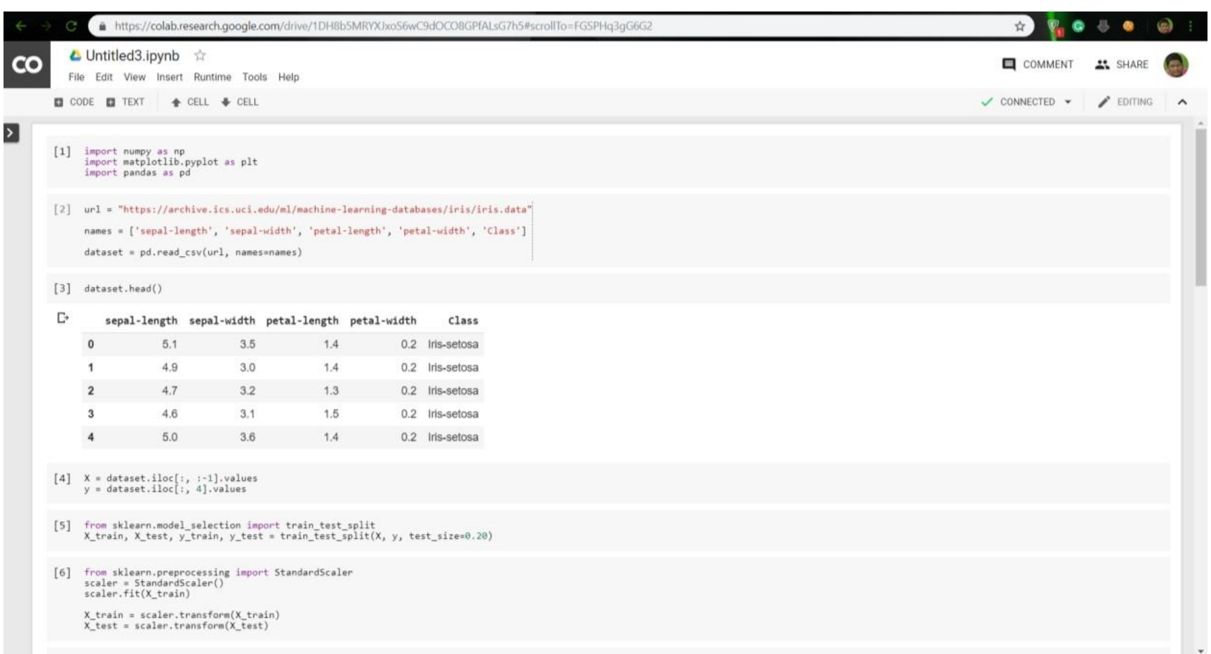
plt.figure(figsize=(12, 6))

plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',

markerfacecolor='blue', markersize=10) plt.title('Error Rate K Value')

plt.xlabel('K Value')   plt.ylabel('Mean Error')

Output ScreenShots



https://colab.research.google.com/drive/1DH8b5MRYXJxoS6wC9dOCO8GPfALsG7h5#scrollTo=p6vVUp_BGphZ

Untitled3.ipynb

File Edit View Insert Runtime Tools Help

CODE TEXT CELL CELL CONNECTED EDITING COMMENT SHARE

```
[4]  X = dataset.iloc[:, :-1].values
     y = dataset.iloc[:, 4].values
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

```
[6]  from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     scaler.fit(X_train)

     X_train = scaler.transform(X_train)
     X_test = scaler.transform(X_test)
```

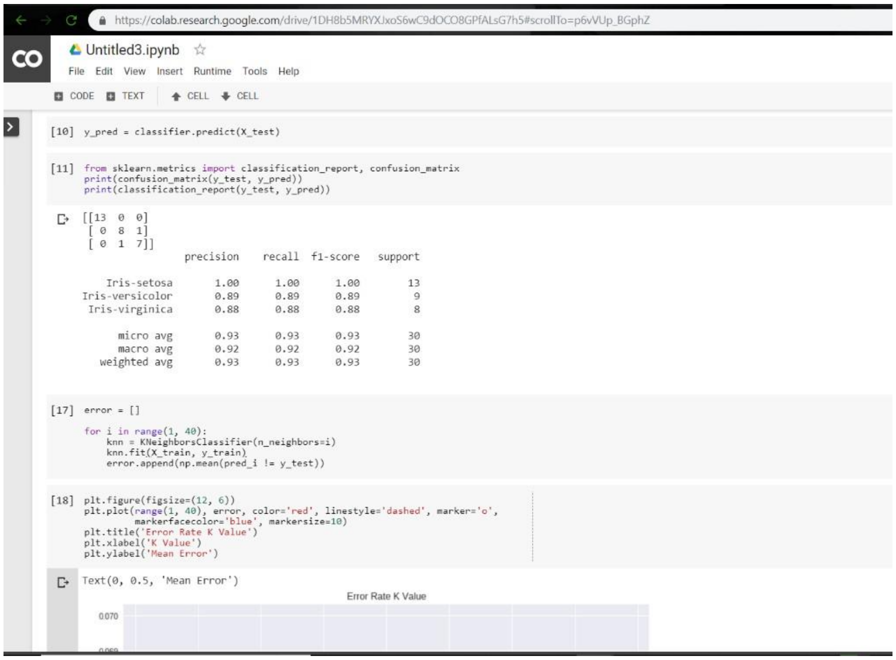
```
[9]  from sklearn.neighbors import KNeighborsClassifier
     classifier = KNeighborsClassifier(n_neighbors=5)
     classifier.fit(X_train, y_train)
```

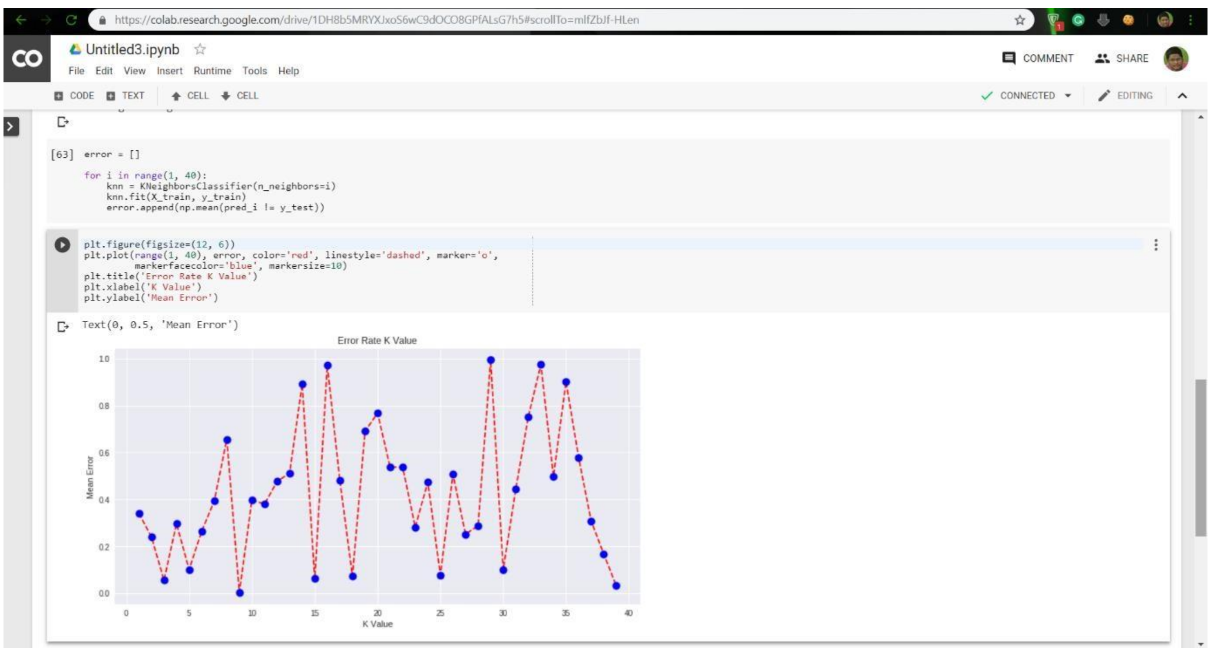
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

```
[10] y_pred = classifier.predict(X_test)
```

```
[11] from sklearn.metrics import classification_report, confusion_matrix
     print(confusion_matrix(y_test, y_pred))
     print(classification_report(y_test, y_pred))
```

```
[[13  0  0]
 [ 0  8  1]
 [ 0  1  7]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        13
Iris-versicolor       0.89      0.89      0.89         9
 Iris-virginica       0.88      0.88      0.88         8

      micro avg       0.93      0.93      0.93        30
      macro avg       0.92      0.92      0.92        30
```





2. Implement Multi-Layer Perceptron using a dataset from UCI repository.

Import all the libraries import numpy

as np import pandas as pd import

matplotlib.pyplot as plt import seaborn

as sns import itertools import warnings

from sklearn.model_selection import train_test_split from

sklearn.preprocessing import StandardScaler from

sklearn.neural_network import MLPClassifier from sklearn import metrics from keras.models import Sequential import tensorflow as tf

Read the file which is downloaded from the UCI repository dataset = pd.read_csv("E:/projects/fraud detection/creditcard.csv") print("Few Entries: ") print(dataset.head())

print("Dataset Shape: ", dataset.shape)

print("Maximum Transaction Value: ", np.max(dataset.Amount))

print("Minimum Transaction Value: ", np.min(dataset.Amount)) **Analysis of the data which we have** color = {1:'blue',0:'yellow'} fraudlist = dataset[dataset.Class == 1] notfraudlist = dataset[dataset.Class == 0] print("The no of Fraud Samples are :", fraudlist.size) print("The no of Non-Fraud Samples are :", notfraudlist.size) fig,axes = plt.subplots(1,2)

axes[0].scatter(list(range(1,fraudlist.shape[0]+1)),fraudlist.Amount,color='blue')

axes[1].scatter(list(range(1,notfraudlist.shape[0]+1)),notfraudlist.Amount,color='yellow') plt.show()

Now split the daset into features and labels with 30 featues and binary label x = dataset.loc[:,dataset.columns.tolist()[1:30]] x = x.as_matrix() y = dataset.loc[:,'Class'] y = y.as_matrix()

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.33,random_state=0)

print("Elements in the training set:" , np.bincount(y_train)) print("Elements in the testing set:" , np.bincount(y_test)) **Function to Train the Model** def trainmodel(model): model.fit(x_train,y_train)

Function To Make Predictions for The Neural Network def predictmodeln(model):

y_pred = model.predict_classes(x_test) f,t,thresholds = metrics.roc_curve(y_test,y_pred) cm = metrics.confusion_matrix(y_test,y_pred) print("Score:", metrics.auc(f,t)) print("Classification report:")

print(metrics.classification_report(y_test,y_pred))

print("Confusion Matrix:") print(cm)

Defining the neural network with 3 layers deep and 1 hidden layer. model = Sequential()

model.add(Dense(256,activation='sigmoid',input_dim=29))

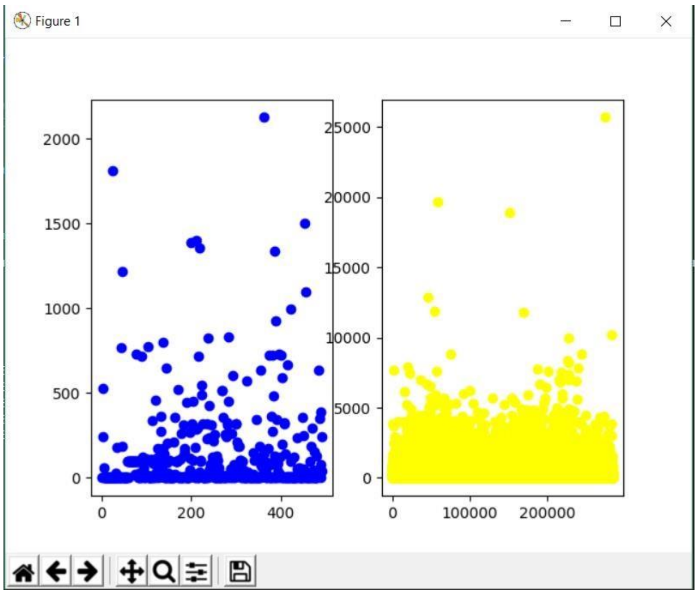
model.add(Dense(128,activation='sigmoid')) model.add(Dense(64,activation='sigmoid'))

model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy']) **Set to the number of epochs the whole network should run with the training data**

model.fit(x_train,y_train,epochs=5) print(predictmodeln(model)) **Output Images :**

Analysing the Dataset



Using TensorFlow backend.

Few Entries:

	Time	V1	V2	V3	...	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	...	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	...	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	...	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	...	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	...	0.219422	0.215153	69.99	0

[5 rows x 31 columns]

Dataset Shape: (284807, 31)

Maximum Transaction Value: 25691.16

Minimum Transaction Value: 0.0

The no of Fraud Samples are : 15252

The no of Non-Fraud Samples are : 8813765

Elements in the training set: [190490   330]

Elements in the testing set: [93825   162] poch 1/5

2019-02-18 17:48:25.211304: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2

32/190820 [..............................] - ETA: 2:32:25 - loss: 1.5449 - acc: 0.0312

192470/192470 [==============================] - 13s 65us/step - loss: 0.0056 - acc: 0.9988

Score: 0.9133937294608758 Classification

report:

precision   recall f1-score   support

0   1.00   1.00   1.00   93825

1   0.79   0.83   0.81   162

micro avg   1.00   1.00   1.00   93987   macro

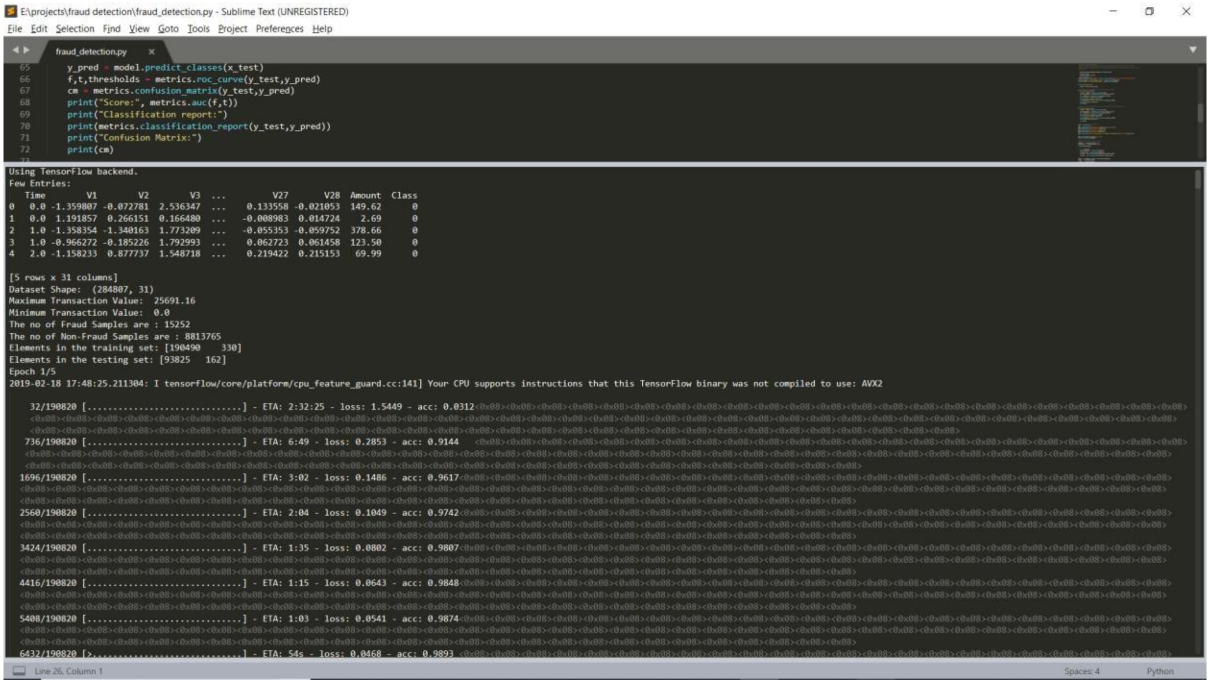
avg   0.90   0.91   0.90   93987 weighted avg

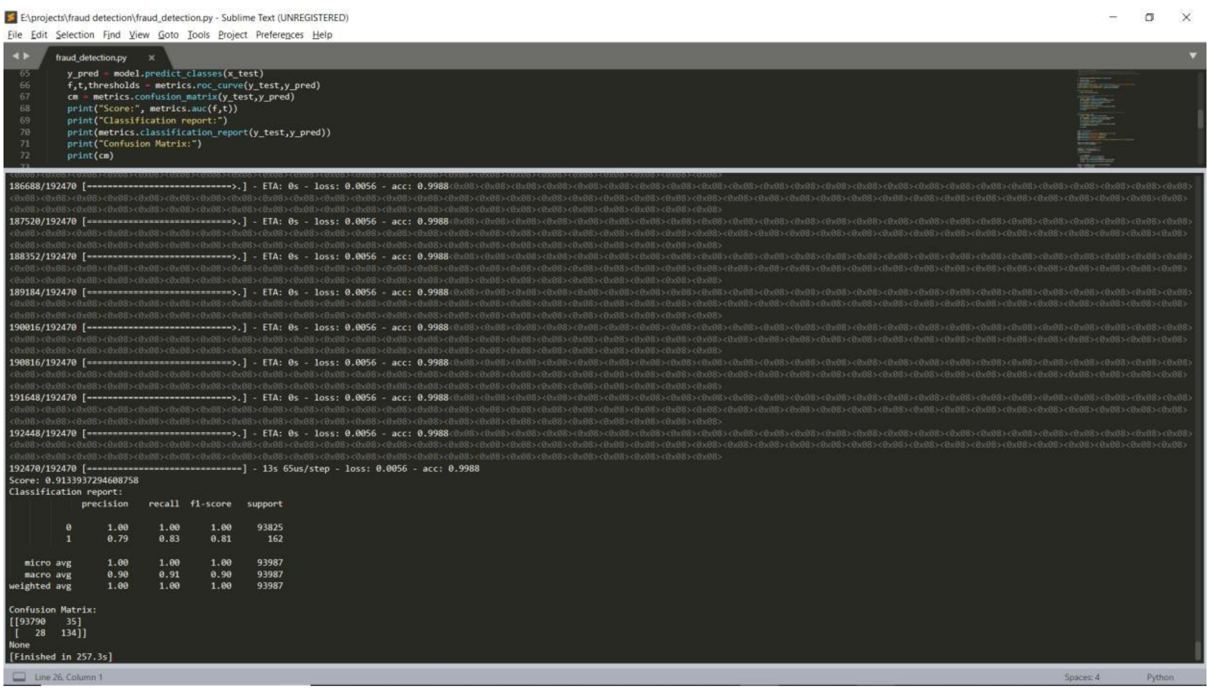
1.00   1.00   1.00   93987

Confusion Matrix:

[[93790   35]

[28   134]] **Images :**





……………………………………………THE END ………………………………………………

ADA BOOST:

```
# Load libraries
from sklearn.ensemble import AdaBoostClassifier
```

```
from sklearn import datasets
# Import train_test_split function
from sklearn.model_selection import train_test_split
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Load data
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70%
training and 30% test

# Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=50,
                         learning_rate=1)
# Train Adaboost Classifer
model = abc.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = model.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

The Output:
Accuracy: 0.8666666666666667

RANDOM FOREST:

```
#Import scikit-learn dataset library
from sklearn import datasets

#Load dataset
iris = datasets.load_iris()

# print the label species(setosa, versicolor,virginica)
print(iris.target_names)

# print the names of the four features
print(iris.feature_names)

# print the iris data (top 5 records)
print(iris.data[0:5])

# print the iris labels (0:setosa, 1:versicolor, 2:virginica)
print(iris.target)

# Creating a DataFrame of given iris dataset.
import pandas as pd
data=pd.DataFrame({
    'sepal length':iris.data[:,0],
    'sepal width':iris.data[:,1],
    'petal length':iris.data[:,2],
```

```
    'petal width':iris.data[:,3],
    'species':iris.target
})
data.head()

# Import train_test_split function
from sklearn.model_selection import train_test_split

X=data[['sepal length', 'sepal width', 'petal length', 'petal width']]  # Features
y=data['species']  # Labels

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70%
training and 30% test

#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

clf.predict([[3, 5, 4, 2]])
```

Here, you are finding important features or selecting features in the IRIS dataset. In scikit-learn, you can perform this task in the following steps:

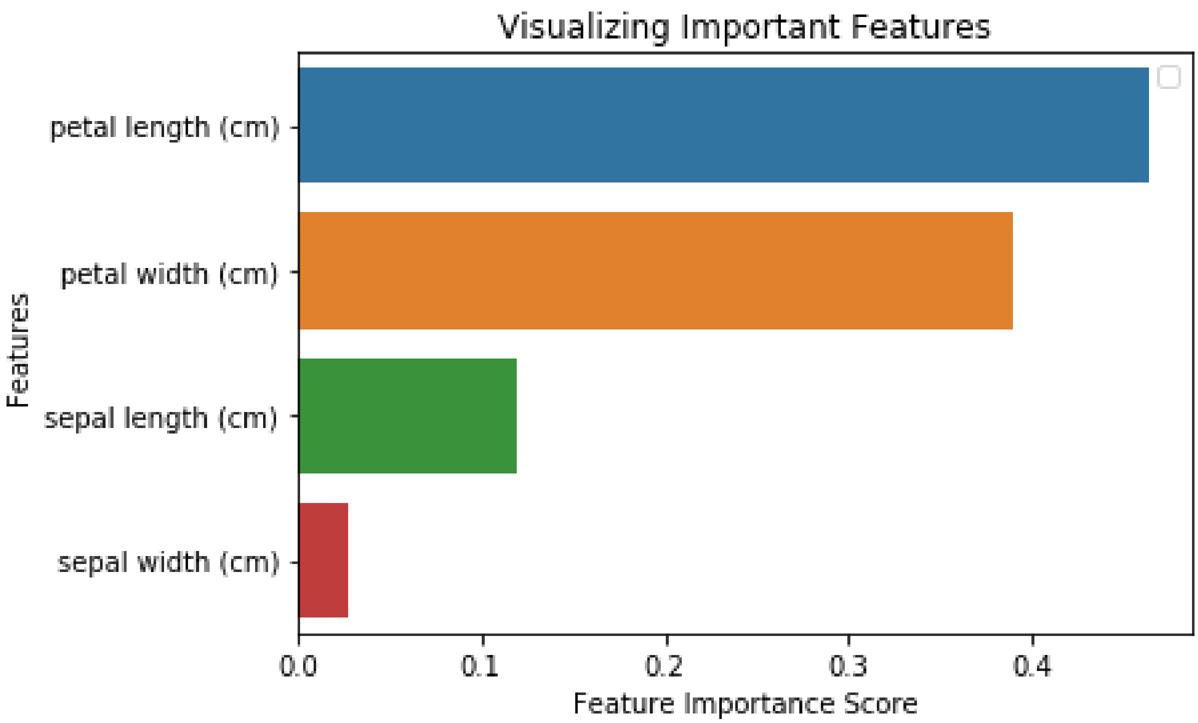
```
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

import pandas as pd

feature_imp=pd.Series(clf.feature_importances_,index=iris.feature_names).sort_value
s(ascending=False)
feature_imp




import matplotlib.pyplot as plt

import seaborn as sns
```

```
%matplotlib inline
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()

# Import train_test_split function

from sklearn.cross_validation import train_test_split
# Split dataset into features and labels
X=data[['petal length', 'petal width','sepal length']]  # Removed feature "sepal
length"
y=data['species']
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.70,
random_state=5) # 70% training and 30% test

from sklearn.ensemble import RandomForestClassifier


#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

# prediction on test set
y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

The Output:

```
No handles with labels found to put in legend.

['setosa' 'versicolor' 'virginica']

['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

[[5.1 3.5 1.4 0.2]

 [4.9 3.  1.4 0.2]

 [4.7 3.2 1.3 0.2]

 [4.6 3.1 1.5 0.2]

 [5.  3.6 1.4 0.2]]
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Accuracy: 0.9333333333333333
```




```
Accuracy: 0.9523809523809523
```

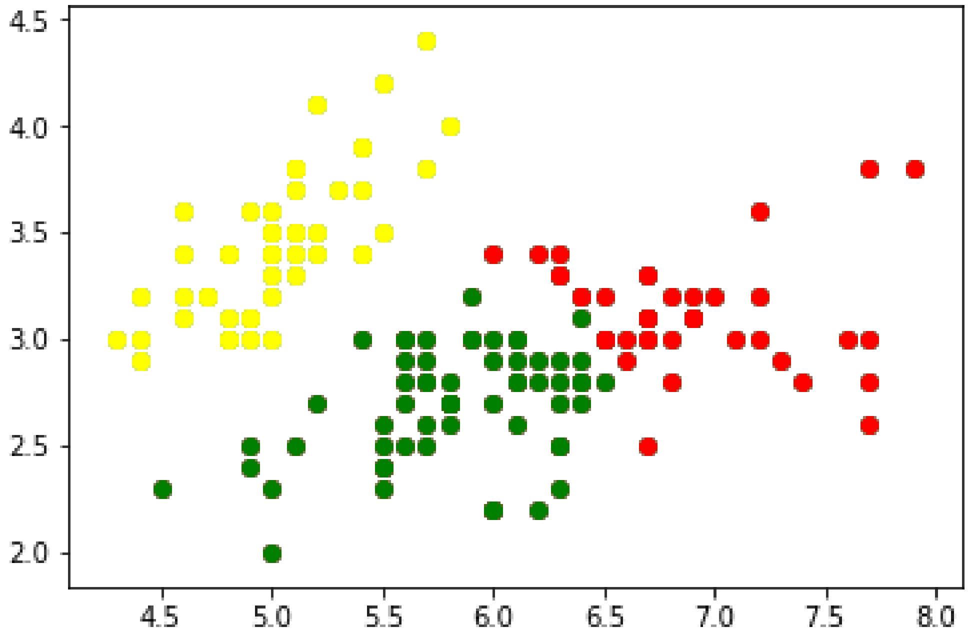
GMM :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas import DataFrame
from sklearn import datasets
from sklearn.mixture import GaussianMixture
```

```
# load the iris dataset
iris = datasets.load_iris()

# select first two columns
X = iris.data[:, :2]

# turn it into a dataframe
d = pd.DataFrame(X)

# plot the data
plt.scatter(d[0], d[1])

gmm = GaussianMixture(n_components = 3)

# Fit the GMM model for the dataset
# which expresses the dataset as a
# mixture of 3 Gaussian Distribution
gmm.fit(d)

# Assign a label to each sample
labels = gmm.predict(d)
d['labels']= labels
d0 = d[d['labels']== 0]
d1 = d[d['labels']== 1]
d2 = d[d['labels']== 2]
```

```
# plot three clusters in same plot

plt.scatter(d0[0], d0[1], c ='r')

plt.scatter(d1[0], d1[1], c ='yellow')

plt.scatter(d2[0], d2[1], c ='g')

gmm = GaussianMixture(n_components = 3)


# Fit the GMM model for the dataset

# which expresses the dataset as a

# mixture of 3 Gaussian Distribution

gmm.fit(d)


# Assign a label to each sample

labels = gmm.predict(d)

d['labels']= labels

d0 = d[d['labels']== 0]

d1 = d[d['labels']== 1]

d2 = d[d['labels']== 2]


# plot three clusters in same plot

plt.scatter(d0[0], d0[1], c ='r')

plt.scatter(d1[0], d1[1], c ='yellow')

plt.scatter(d2[0], d2[1], c ='g')
```

The Output:


4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0