**CSE4020 – Machine Learning Lab**

**Classification and Regression**

**Om Ashish Mishra**

**16BEC0789**

**F2**

**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)

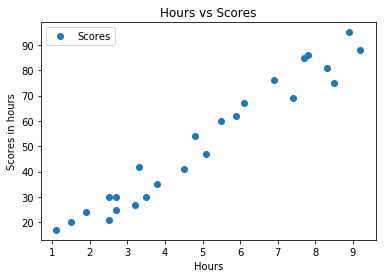
plt.xlabel('x')

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**

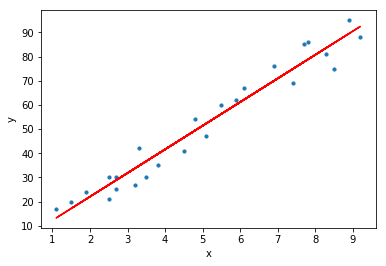
**78.73493985]**

**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

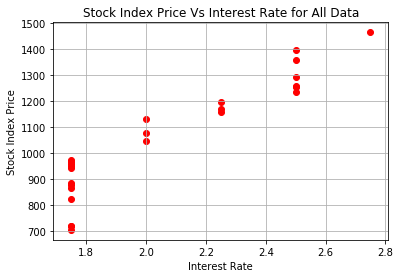
20 2016.0 4.0 ... 5.9 876.0

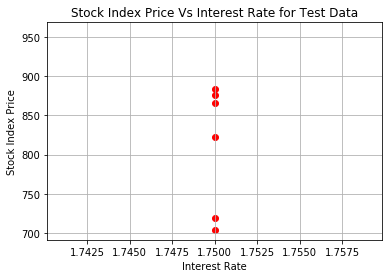
21 2016.0 3.0 ... 6.2 822.0

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]

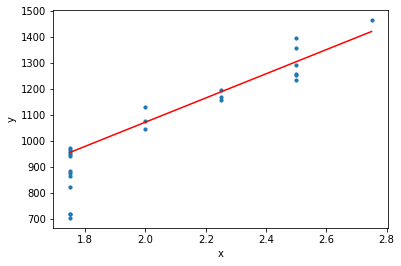
[ 956.06502636]

[ 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

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))

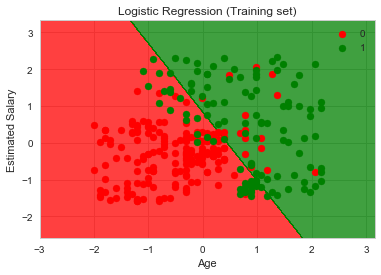
print()

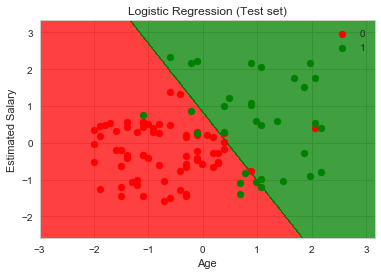
#classification report

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**

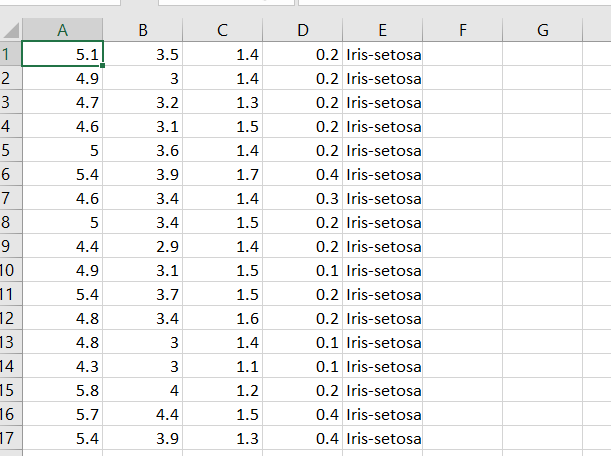
**precision recall f1-score support**

**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**

The Dataset:



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')

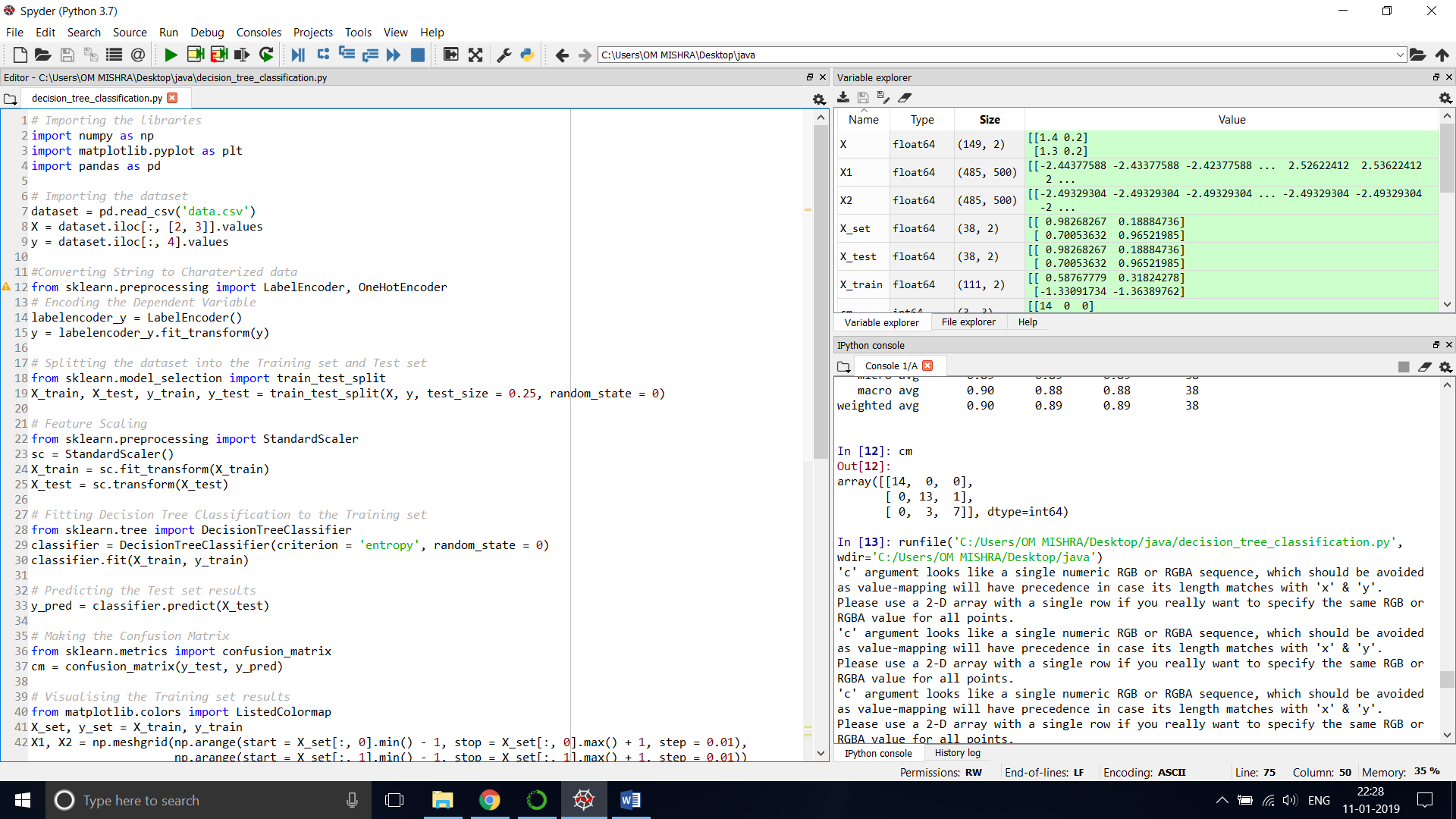
plt.legend()

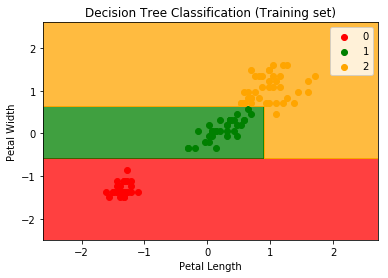
plt.show()

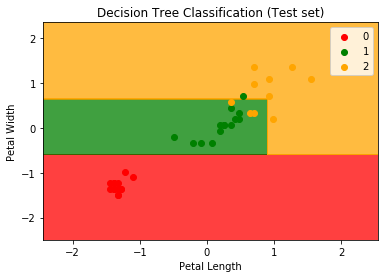
from sklearn.metrics import classification\_report

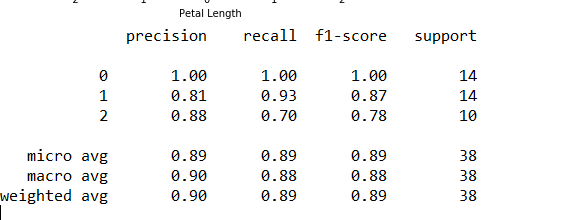
print(classification\_report(y\_test,y\_pred))

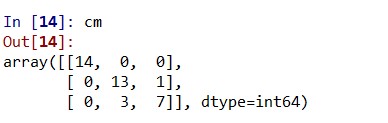
The Output:











CART Algorithm

# 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 = 'gini', 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:

