**CSE-3024 Web Mining**

**Lab Assignment - 4**

**Alokam Nikhitha**

**19BCE2555**

Decision Tree

Aim

Using a Decision Tree Classifier, divide the given network intrusion dataset into normal and abnormal categories. Along with the classification, the following items must be printed:

* Confusion Matrix
* Accuracy of model on Test data
* Decision Tree visualization.

**Dataset Used:** The network intrusion dataset from Kaggle.

Link to which is: <https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection?select=Train_data.csv>

**Procedure:**

- First, we import the necessary numpy, pandas, matplotlib, and tree libraries.

- The dataset is then imported into our workspace. The set of independent and dependent attributes is also defined.

- Next, we used a 7.5:2.5 ratio to divide the dataset into training and test sets.

- Then, using DecisionTreeClassifier from sklearn.tree, we train our decision tree model.

- Next, we look for the test set results that our model anticipated.

- Then, using the expected and test set findings, we print our confusion matrix.

- Similarly, we print the model's accuracy based on the test set and anticipated result.

- Finally, we visualise our model using the sklearn tree.

**Code:**

#19BCE2555

#Importing libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn import tree

#Importing dataset

dataset = pd.read\_csv("Train\_data.csv")

X = dataset.iloc[:, 4:41].values

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

#Splitting the dataset

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)

#Fitting our model

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)

#Printing the confusion matrix

from sklearn.metrics import confusion\_matrix

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

print(cm)

#Printing the accuracy of our model

from sklearn.metrics import accuracy\_score

accuracy = accuracy\_score(y\_test, y\_pred)

print(accuracy)

#Defining the labels of our dataset

classes = ["Anamoly", "Normal"]

#Printing the visualized decision tree

fig = plt.figure(figsize=(25,20))

\_ = tree.plot\_tree(classifier,

feature\_names=dataset.columns,

class\_names=classes,

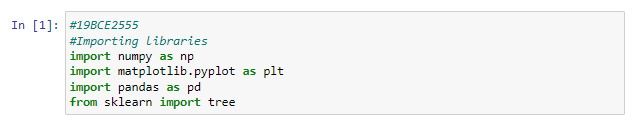
filled=True)

#Printing the feature wise break points of our decision tree

test\_representation = tree.export\_text(classifier)

print(test\_representation)

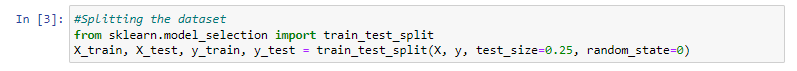
**Code Snippets and Outputs:**



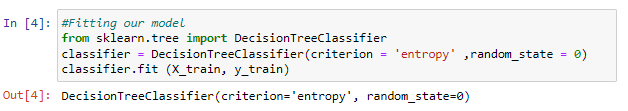
We're importing our libraries right now. Nupmy is imported as np, pandas is imported as pd, matplotlib's pyplot extension is imported as plt, and finally tree is imported from sklearn.



We're using pandas to import our Network Intrusion Dataset into our workspace. Then a set of dependent and independent qualities is defined. The set of independent qualities is labelled X, while the set of dependent attributes is labelled y.



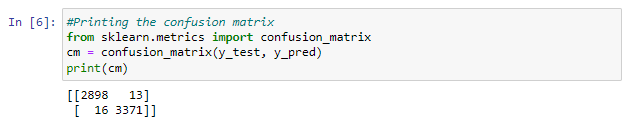
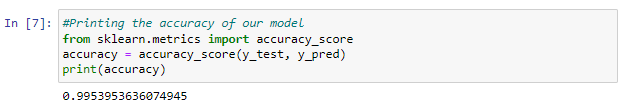
We're going to divide our dataset into two parts: a training set and a test set. We're going to maintain 25% of the dataset in the test set and 75% in the training set.



We're taking data from the training set to train our model. For our decision tree classifier, we employed "entropy" as the deciding factor.



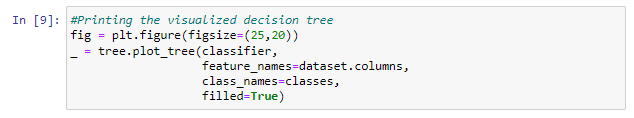
We're collecting our anticipated test set results from the classifier and saving them in the y pred variable.

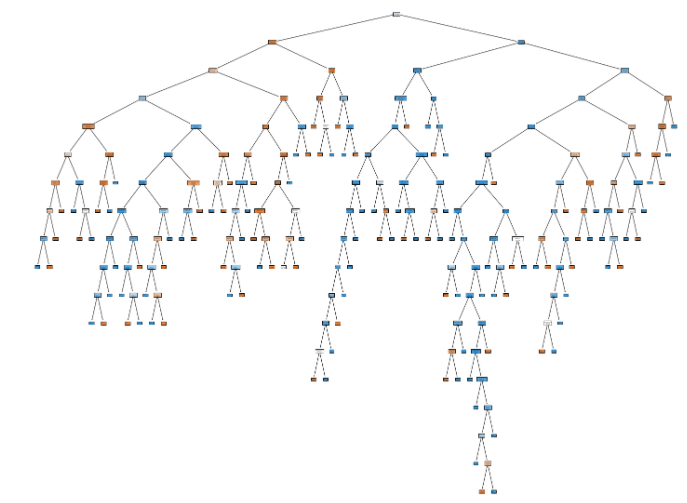
 

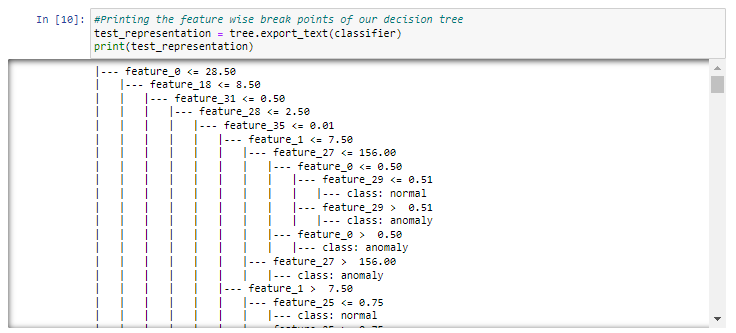
The confusion matrix and accuracy of our decision tree classifier are printed here. Our model's accuracy with the test dataset is 99.53953636 percent.



We're using sklearn's tree library to visualise our decision tree.







The categorization criterion of our decision tree is presented here. We can see that feature 0 is our classifier's root node, followed by multiple middle nodes.

**Results and Output**

Confusion Matrix:



This is our confusion matrix.

True Negatives: 3488

True Positives: 4032

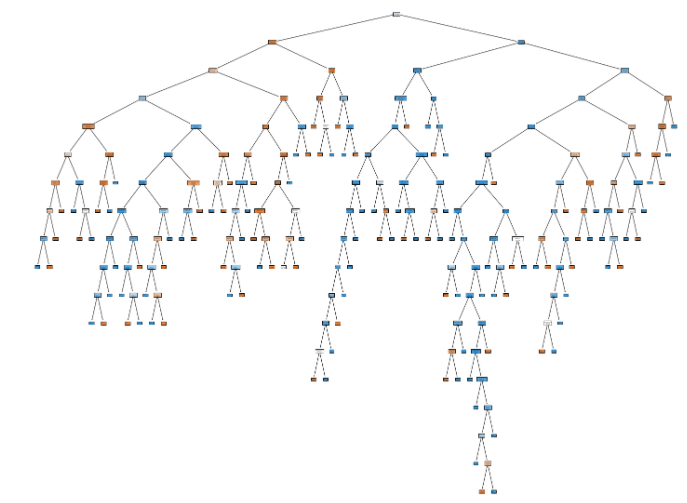
False Positives: 17

False Negatives: 21

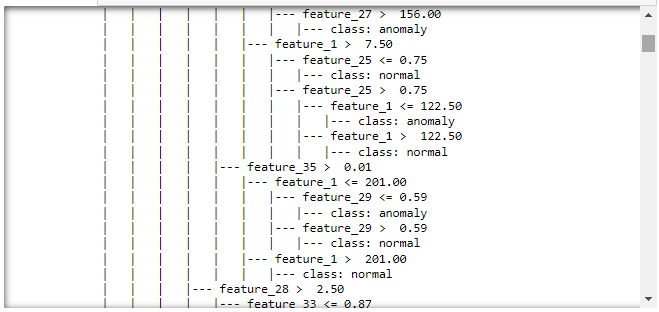
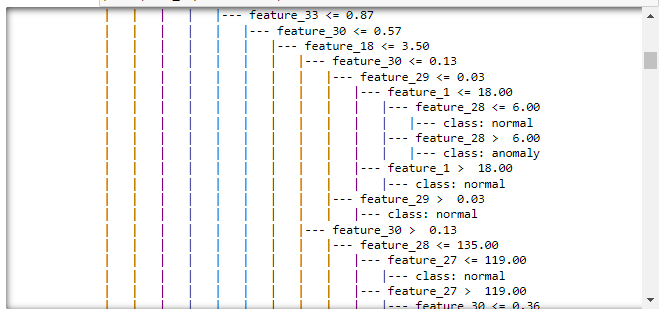
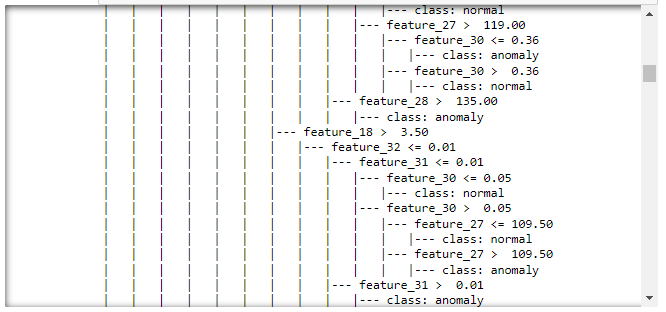
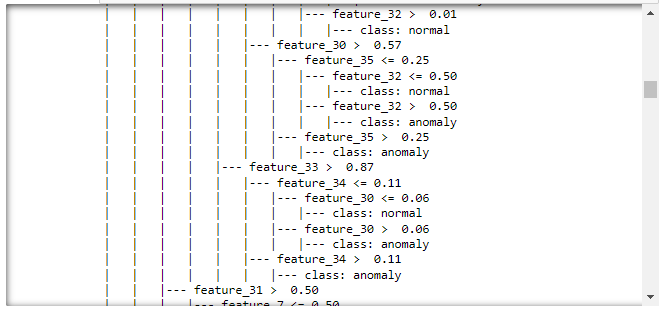
Accuracy:

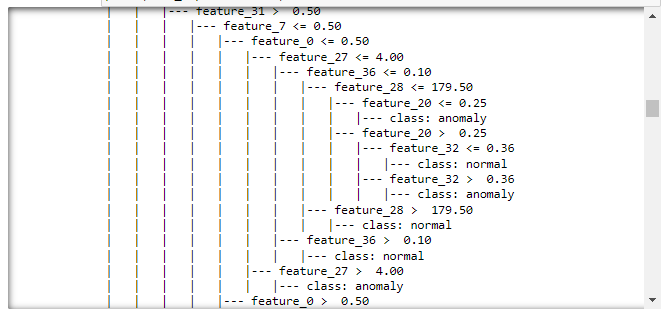
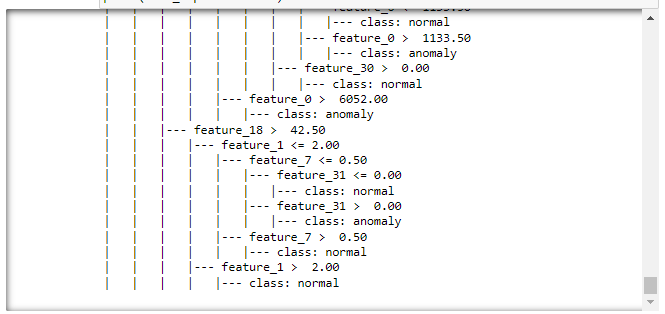
The accuracy of our model stands at 99.49%

Decision Tree Visualization:



Classification Points:

K-Means

**Problem statement:**

Illustrate the k-means clustering to cluster the data points for at least five epoch properly.

How to Implementing K-Means Clustering?

• Using the elbow method to determine the optimal number of clusters for kmeans clustering

• Visualising the clusters

• Plotting the centroids of the clusters

**Dataset used:**

• Shopping-data

• <https://archive.ics.uci.edu/ml/machine-learning-databases/>

**Procedure:**

- Import necessary libraries - sklearn, numpy, pandas, etc.

-Using pandas, we first import the dataset into our workspace.

- Select the number of clusters for the dataset ( K )

- Select K number of centroids

- By calculating the Euclidean distance or Manhattan distance assign the points to the nearest centroid, thus creating K groups

- Now find the original centroid in each group

-  Again reassign the whole data point based on this new centroid, then repeat step 4 until the position of the centroid doesn’t change.

- Using the elbow method to determine the optimal number of clusters for kmeans clustering

- Visualising the clusters and Plotting the centroids of the clusters

**Code:**

#Importing Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

#Importing the Datasets

dataset = pd.read\_csv('shopping-data.csv')

X = dataset.iloc[:, 3:].values

#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++', max\_iter=300, n\_init=10)

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()

#Applying Kmeans to the dataset

kmeans = KMeans(n\_clusters=5, init='k-means++', max\_iter=300, n\_init=10);

y\_kmeans = kmeans.fit\_predict(X)

#Printing out the cluster each input belongs to

y\_kmeans

# Visualising the clusters

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Standard Customers')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Careless Customers')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'cyan', label = 'Target Customers')

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 100, c = 'magenta', label = 'Sensible Customers')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 100, c = 'green', label = 'Careful Customers')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

plt.title ('Clusters of Clients')

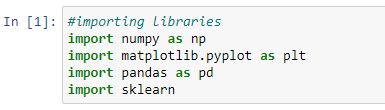
plt.xlabel ('Annual Income (k$)')

plt.ylabel ('Spending Score (1-100)')

plt.legend()

plt.show()

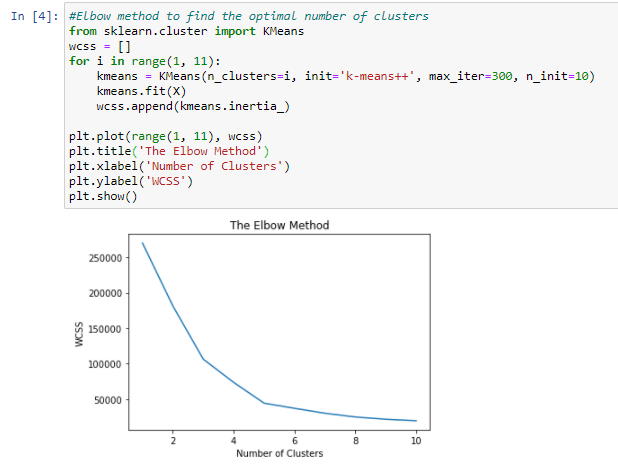
**Code Snippets and Outputs:**

****

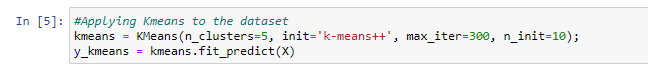
Import necessary libraries - sklearn, numpy, pandas, etc.

****

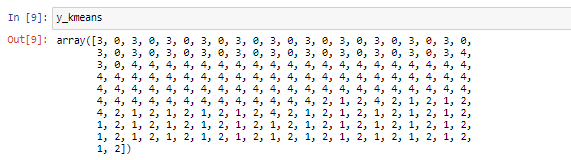
Using pandas, we first import the dataset into our workspace and are assigning the income attribute along with shopping score as independent variables.



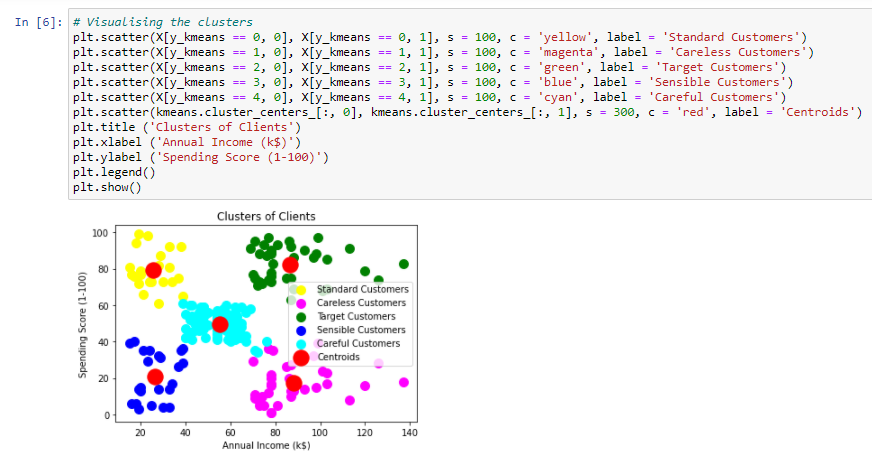
Here we are plotting a graph that marks Within Cluster Sum of Squares (WCSS) with the increase in number of clusters. We can see an elbow formation when the number of clusters is 5 and hence, we assume that optimal number of clusters in our dataset is 5



Here we are training our k-means model with 5 clusters. We are also generating the y\_kmeans array that stores the cluster index of each input attribute from 0 to 4



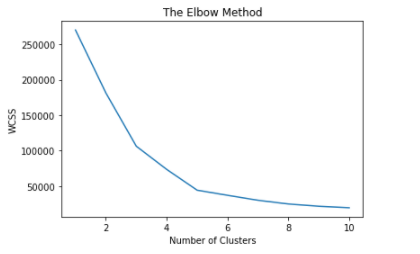
Here we are printing our y\_kmeans array and we can see that each input cell is assigned a value between 0 and 4, both inclusive. This corresponds to the cluster index of each input.



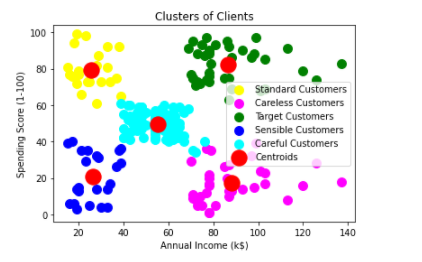
Here we have visualized our results. We have labelled different clusters as blue, green, pink, yellow and cyan. Each cluster correspond to different category of target audience. We have also marked centroids of each cluster which are red in colour.

**Results and Output**

Elbow Method Graph



Clustering Graph:



Here different clusters are marked as blue, yellow, magenta, cyan and green. The red dot over each cluster represents its centroid.

We can categorise these clusters as: -

* Yellow Cluster corresponds to careless customers as they have low income but high spending.
* Blue Cluster as Sensible customers, becoz they have low income and low spending.
* Cyan Clusters are standard cluster that suggest they have median income and median spending.
* The pink coloured cluster correspond to Target Customers, as they have high income but low spending, the shopping company can give them offers and attractions as they are capable of spending more but they aren’t doing it currently.
* Finally, the Green coloured clusters are Careful customers. They have high income and thus high spending as well.

Random Forest

**Question:**

The following are the basic steps involved in performing the random forest algorithm:

1. Pick N random records from the dataset.

2. Build a decision tree based on these N records.

3. Choose the number of trees you want in your algorithm and repeat steps 1 and 2.

4. In case of a regression problem, for a new record, each tree in the forest predicts a value for Y (output). The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

**Dataset Used:**

petrol\_consumption.csv, bill\_authentication.csv.

**Procedure:**

-Using pandas, we first import the dataset into our workspace.

-Next we define the set of dependent and independent attributes.

- We then import the random forest regressor from sklean rn.ensemble and train our model using the independent and dependent attributes.

- Next, we have printed the results of independent set as predicted by our regressor.

- Lastly, To check for the performance of our dataset, we have printed all the evaluation metrics

Since it has less Number of Rows we haven’t split the dataset

**Petrol\_consumption dataset**

**Code**

#Importing Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

#Importing the Dataset

dataset = pd.read\_csv("petrol\_consumption.csv")

#First few rows of our dataset

dataset.head(10)

#Checcking for null values

print(dataset.info())

X = dataset.iloc[:, 0:4].values

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

#Training our Random Forest Regression Model

from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n\_estimators=200, random\_state=0)

regressor.fit(X, y)

#Predictions by Regressor

y\_pred = regressor.predict(X)

#Printing Mean Absolute Error

from sklearn.metrics import mean\_absolute\_error

mean\_absolute\_error(y, y\_pred)

#Printing Mean Absolute Error

from sklearn.metrics import mean\_squared\_error

mean\_squared\_error(y, y\_pred)

#Printing Root Mean Squared Error

np.sqrt(mean\_squared\_error(y, y\_pred))

#Printing Root Mean Sqaured Log Error

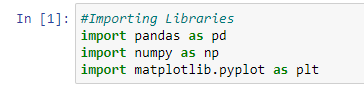
np.log(np.sqrt(mean\_squared\_error(y, y\_pred)))

#Printing R-square value

from sklearn.metrics import r2\_score

r2\_score(y, y\_pred)

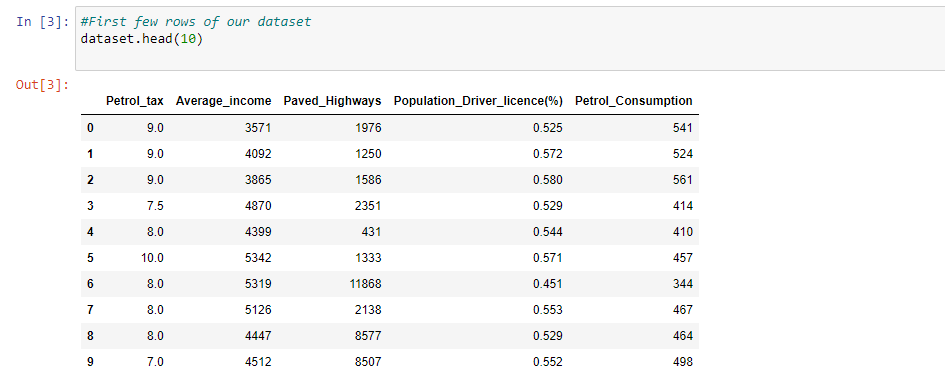
**Code Snippets and Explanation:**

****

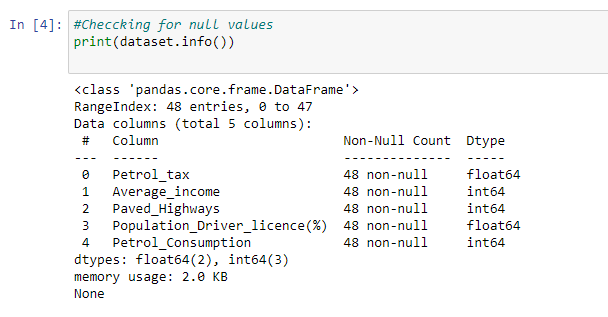
**Here we are importing the required Libraries**

****

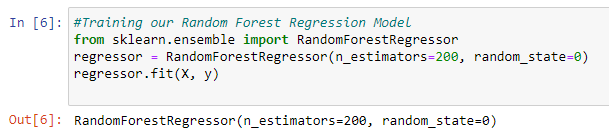
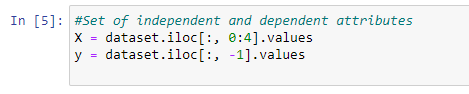
**Using Pandas we are importing the data**

****

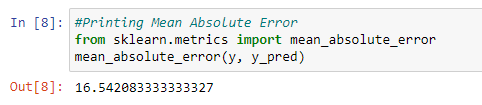
**Printing the first few rows.**

****

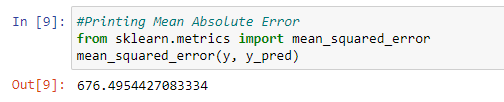
**Here we are checking for the null values.**

****

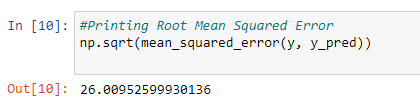
We have Defined set of Dependent and Independent attributes.The n\_estimators here indicate the number of decision trees that we are using to train our random forest regressor. Hence we are using 200 decision trees for prediction. For final value we have used the average value of each decision tree to find the final consumption of petrol of a particular region.

****

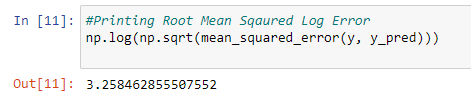
**Printing the Mean Absolute Error**

****

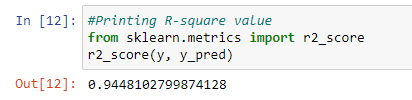
**Printing the Mean Squared Error**

****

**Printing the Root Mean Squared Error**

****

**Printing the Root Mean Sqaured Log Error**

****

**Printing the R-square value**

**Results and Conclusions:**

**Mean Absolute Error from cell8 is** 16.542083333333327

**Mean absolute error from cell 9 is** 676.4954427083334

**Root Mean Squared Error from cell10 is** 26.00952599930136

**Root Mean Squared Log Error from cell11 is** 3.258462855507552

**R-square value from cell12 is** 0.9448102799874128

**Bill\_authentication dataset**

**Code**

#Importing Libraries

import pandas as pd

#importing the bill\_authentication dataset

dataset = pd.read\_csv('bill\_authentication.csv')

#Displaying the first few rows of the dataset

dataset.head()

X = dataset.iloc[:, 0:4].values

y = dataset.iloc[:, 4].values

#Training our Random Forest Regression Model

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier(n\_estimators=20, random\_state=0)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

print(confusion\_matrix(y\_test,y\_pred))

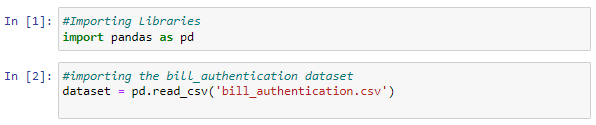
#printing classification\_report

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

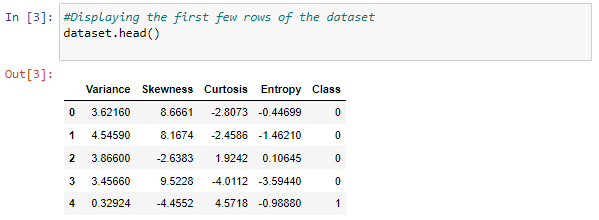
#printing Accuracy

print(accuracy\_score(y\_test, y\_pred))

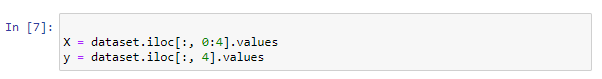
**Code Snippets and Explaination**

****

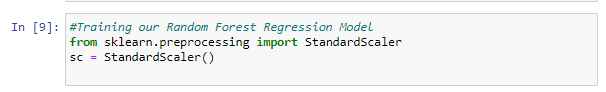
**Here we are importing the required Libraries. Using Pandas we are importing the data**

****

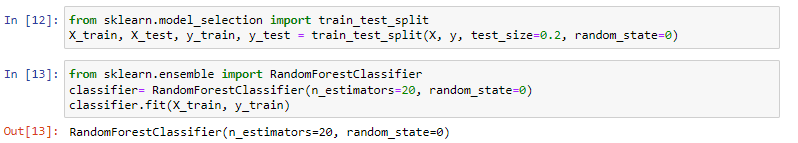
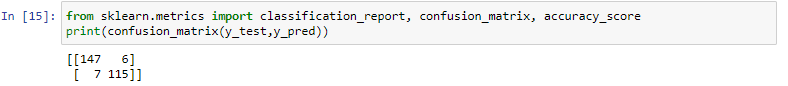
**Printing the first few rows.**

****

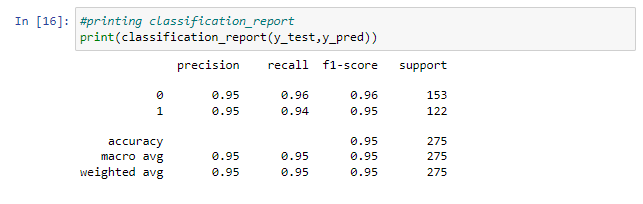
**Defining the Dependent and Independent variables**

****

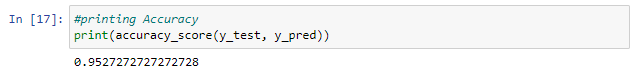
Here we are training our Random forest Regression model

****  

Here we are printing the Confusion Matrix



Here we are printing the Classification Report

****

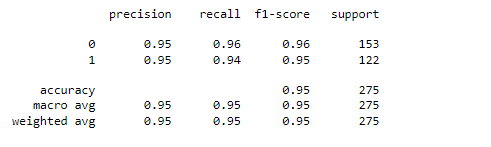
**The Accuracy of the model is** 0.9527272727272728

**Results and Conclusion**

**Confusion Matrix**

****

**Classification Report**

****

**Accuracy of the dataset is:** 0.9527272727272728