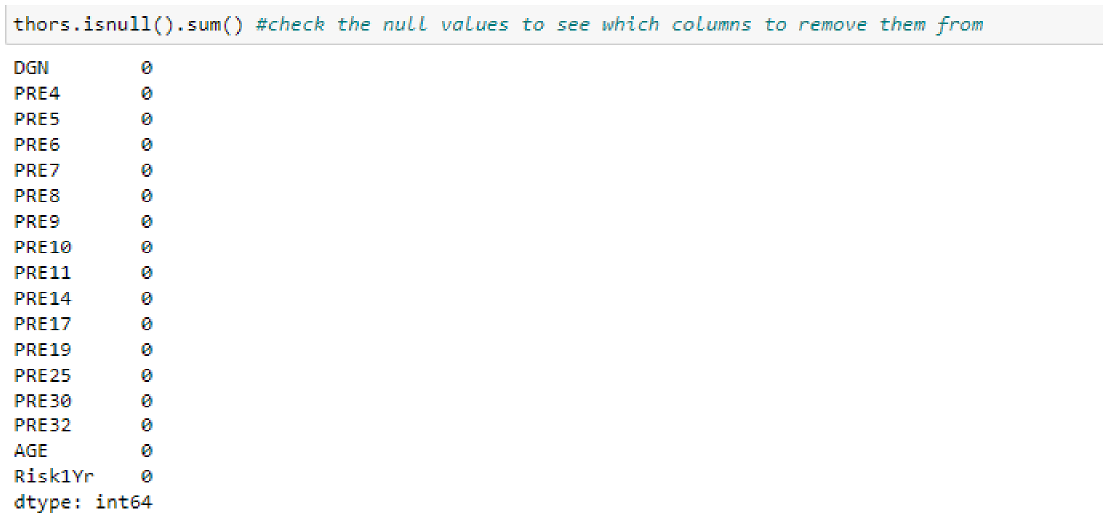
Ahmed Sameh 211392

What is the data about?

This dataset is dedicated to classification problem related to the post-operative life expectancy in the lung cancer patients about (Thoracic Surgery). The data has 470 instances. There is a 2 classes F and T where T says the patient died and these are in the target label is the "Risk1Yr" column, thus it is a binary classification problem.

Data Cleaning /Processing

Fig.1

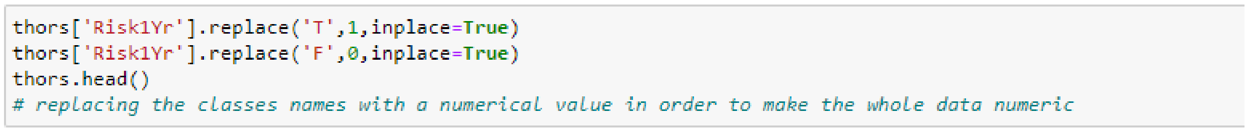


```
thors.isnull().sum() #check the null values to see which columns to remove them from
```

```
DGN        0
PRE4       0
PRE5       0
PRE6       0
PRE7       0
PRE8       0
PRE9       0
PRE10      0
PRE11      0
PRE14      0
PRE17      0
PRE19      0
PRE25      0
PRE30      0
PRE32      0
AGE        0
Risk1Yr    0
dtype: int64
```

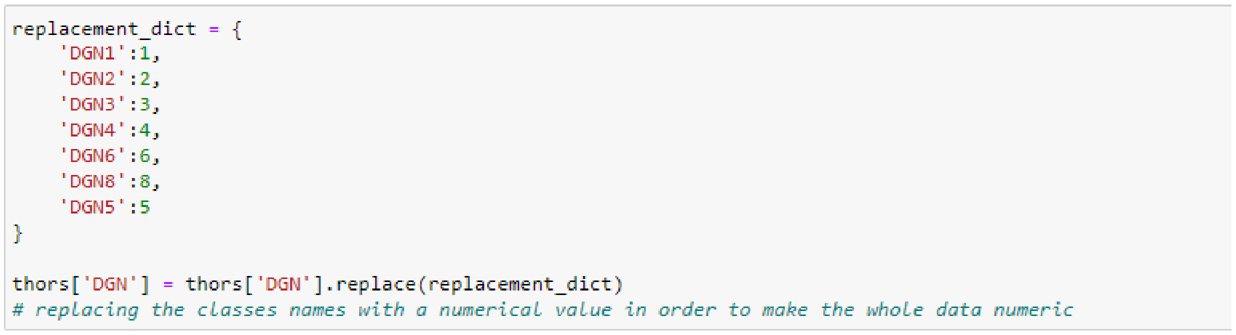
Based on Fig.1 above there are no **nulls** in the dataset.

Fig.2



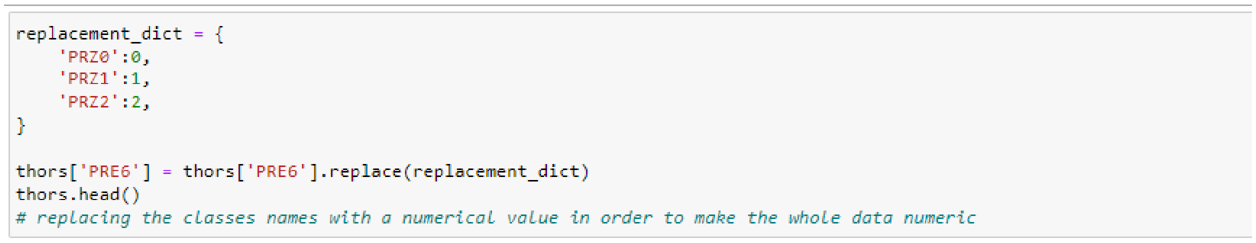
```
thors['Risk1Yr'].replace('T',1,inplace=True)
thors['Risk1Yr'].replace('F',0,inplace=True)
thors.head()
# replacing the classes names with a numerical value in order to make the whole data numeric
```

Fig.3



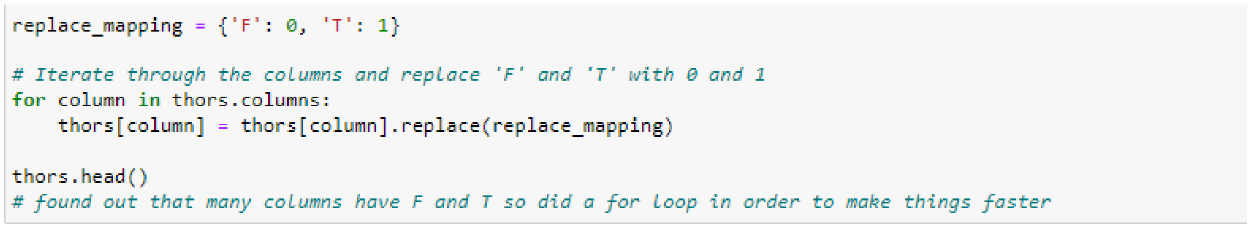
```
replacement_dict = {
    'DGN1':1,
    'DGN2':2,
    'DGN3':3,
    'DGN4':4,
    'DGN6':6,
    'DGN8':8,
    'DGN5':5
}

thors['DGN'] = thors['DGN'].replace(replacement_dict)
# replacing the classes names with a numerical value in order to make the whole data numeric
```

Fig.4



```
replacement_dict = {
    'PRZ0':0,
    'PRZ1':1,
    'PRZ2':2,
}

thors['PRE6'] = thors['PRE6'].replace(replacement_dict)
thors.head()
# replacing the classes names with a numerical value in order to make the whole data numeric
```

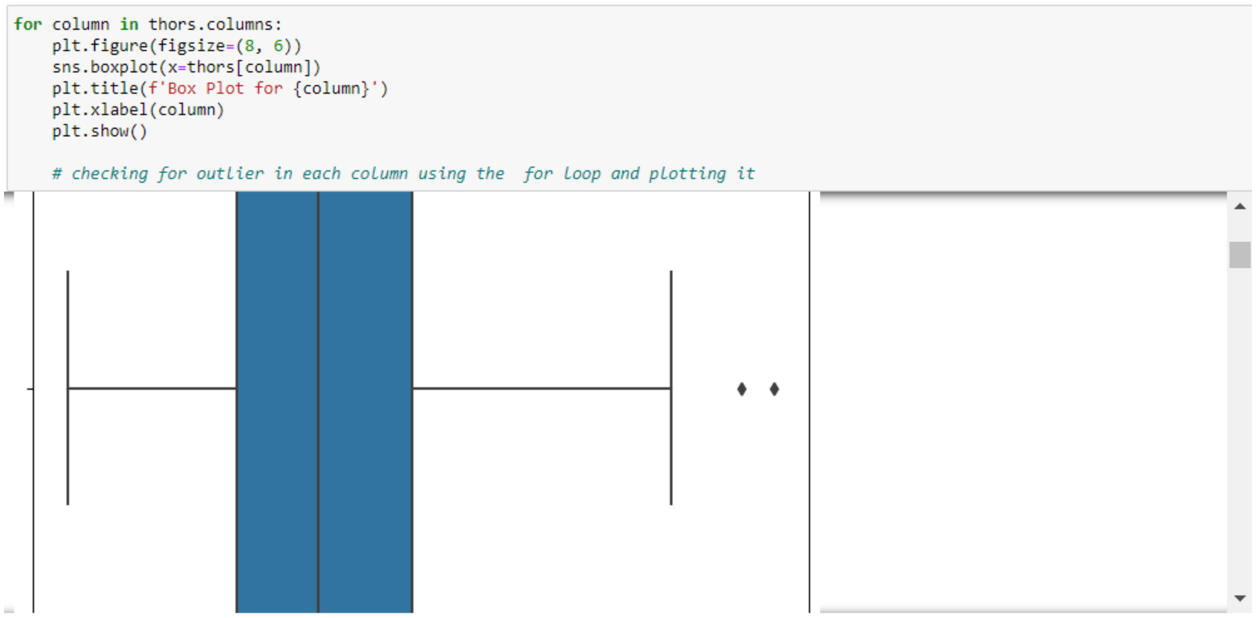
Fig.5



```
replace_mapping = {'F': 0, 'T': 1}

# Iterate through the columns and replace 'F' and 'T' with 0 and 1
for column in thors.columns:
    thors[column] = thors[column].replace(replace_mapping)

thors.head()
# found out that many columns have F and T so did a for loop in order to make things faster
```

Based on Fig.2- Fig5 changed the obj data type to int data type in order to be able to process data as whole.

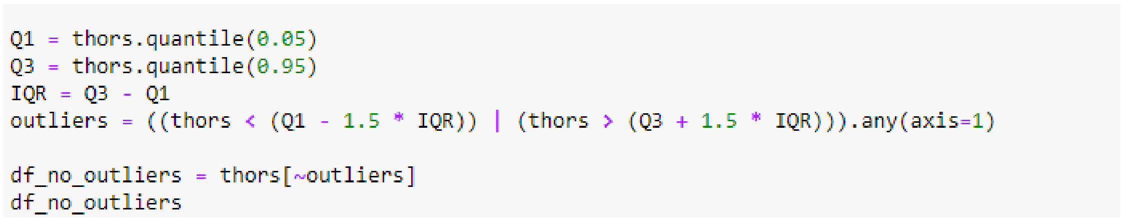
Fig.6



```
for column in thors.columns:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x=thors[column])
    plt.title(f'Box Plot for {column}')
    plt.xlabel(column)
    plt.show()

    # checking for outlier in each column using the  for loop and plotting it
```

Fig.6 shows the outliers of each column using the for loop

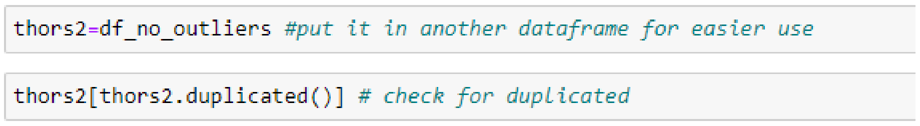
Fig.7



```
Q1 = thors.quantile(0.05)
Q3 = thors.quantile(0.95)
IQR = Q3 - Q1
outliers = ((thors < (Q1 - 1.5 * IQR)) | (thors > (Q3 + 1.5 * IQR))).any(axis=1)

df_no_outliers = thors[~outliers]
df_no_outliers
```

As in Fig.7 I removed the outliers in the data frame to clean the data I chose a small quantile as it removed a lot of data when I put it as 0.10 and 0.95 it removed more than half the instances, so I tried to find a percentage that removed a few only.

Fig.8



```
thors2=df_no_outliers #put it in another dataframe for easier use
```

```
thors2[thors2.duplicated()] # check for duplicated
```

Found no duplicates in the dataset.

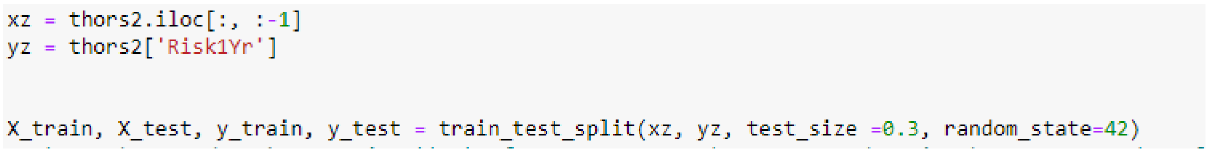
Models:

Relating to the models I have used my dataset once with outliers and once without to compare between the accuracy and performances seeing if removing the outlier really affects the data the much or if kept untouched from removing outliers better.

I used the 4 same classification models on both including the decision tree, random forest, gradient boosting, SVM.

In Fig 9. Below, First we split the data into training and testing based on the x and y where the x has all the features except the target and y has the target

Fig.9

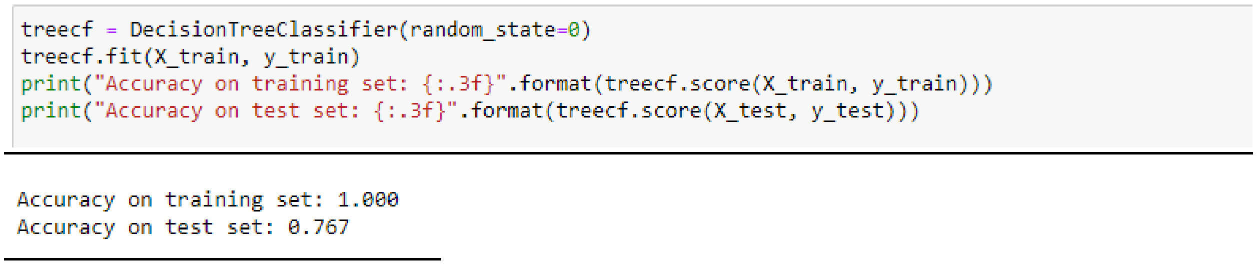


```
xz = thors2.iloc[:, :-1]
yz = thors2['Risk1Yr']


X_train, X_test, y_train, y_test = train_test_split(xz, yz, test_size =0.3, random_state=42)
```

Decision tree:

As seen in Fig10. I use the dt classifier model only putting in it random state for starters while it trains the dt model on a training dataset then print the test and accuracy. The training set received a score of 100% according to the model's accuracy rating, suggesting that the model is overfit. Moreover, accuracy evaluation is not the ideal evaluation because the data is class weighted towards class 0.

Fig.10

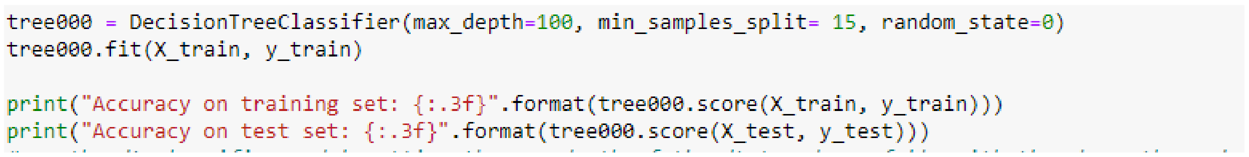


```
treecf = DecisionTreeClassifier(random_state=0)
treecf.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(treecf.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(treecf.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.767
```

Due to that I have decided to change in the hyper parameters of the classifier to get to better results.

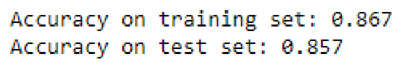
Fig.11



```
tree000 = DecisionTreeClassifier(max_depth=100, min_samples_split= 15, random_state=0)
tree000.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(tree000.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree000.score(X_test, y_test)))
```

I used the dt classifier model putting the max depth of the dt tuned carefully with the where the nodes will be split only if having a minimum of 15 samples thus might then avoid the overfitting that happened.

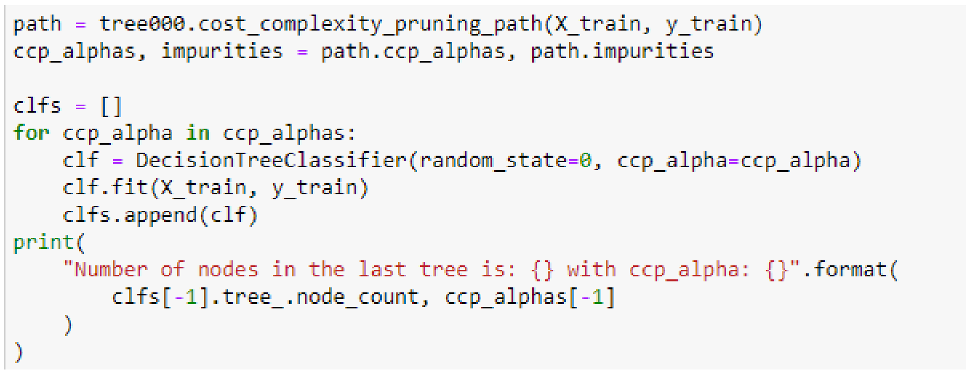
Fig.12



```
Accuracy on training set: 0.867
Accuracy on test set: 0.857
```

In Fig.12 the accuracy changed drastically making it better than before just avoiding overfitting

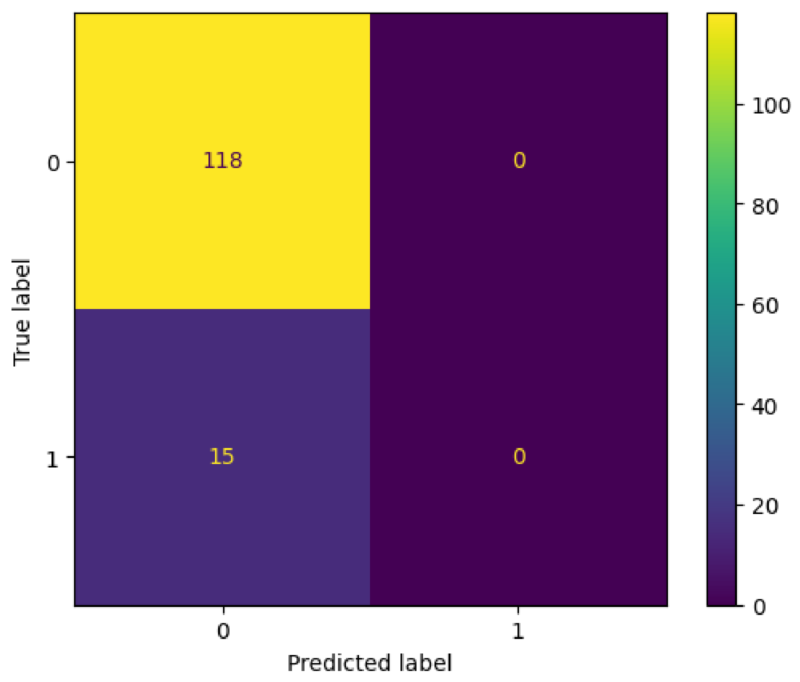
Fig.13



```
path = tree000.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print(
    "Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
```

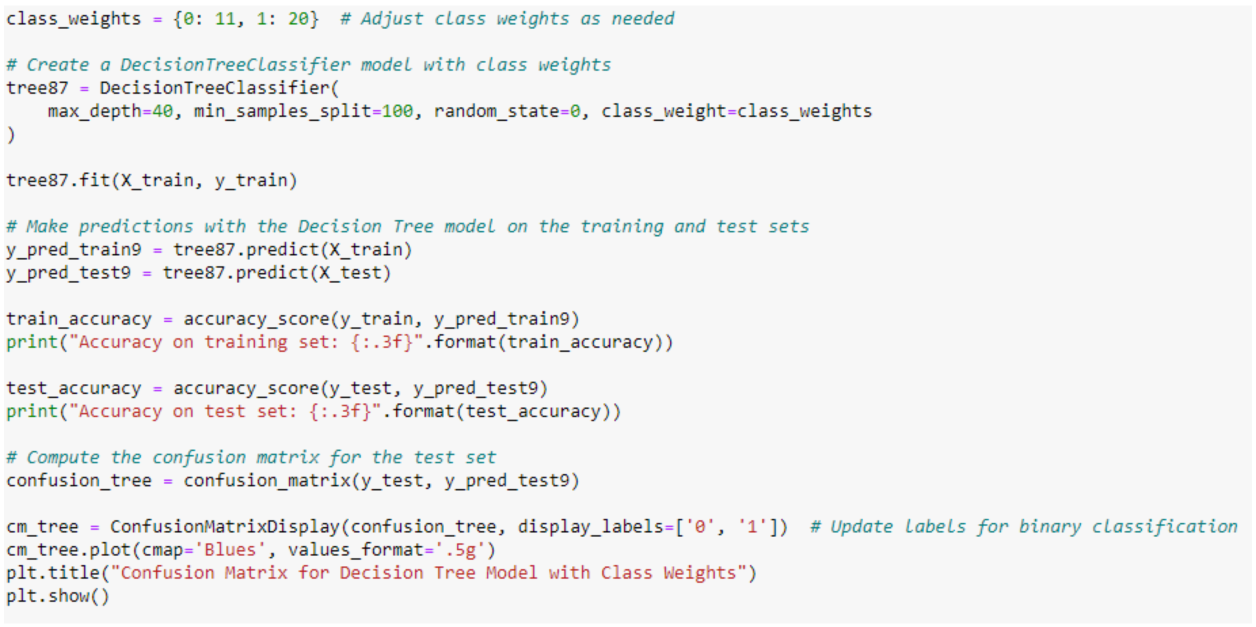
In the fig above I used the cost complexity pruning on the original dt finding the right balance of the ccp alpha to balance the model and stop the overfitting. However it failed as the results were much worse leading to confusion matrix only going towards on of the classes which is zero.

Fig.14



In the fig above the true positive of the ones seem very perfect however the true negatives aren't there scoring zero so I decided to turn down this way and went on to do more tuning as seen below.

Fig.14



```python
class_weights = {0: 11, 1: 20}  # Adjust class weights as needed

# Create a DecisionTreeClassifier model with class weights
tree87 = DecisionTreeClassifier(
    max_depth=40, min_samples_split=100, random_state=0, class_weight=class_weights
)

tree87.fit(X_train, y_train)

# Make predictions with the Decision Tree model on the training and test sets
y_pred_train9 = tree87.predict(X_train)
y_pred_test9 = tree87.predict(X_test)

train_accuracy = accuracy_score(y_train, y_pred_train9)
print("Accuracy on training set: {:.3f}".format(train_accuracy))

test_accuracy = accuracy_score(y_test, y_pred_test9)
print("Accuracy on test set: {:.3f}".format(test_accuracy))

# Compute the confusion matrix for the test set
confusion_tree = confusion_matrix(y_test, y_pred_test9)

cm_tree = ConfusionMatrixDisplay(confusion_tree, display_labels=['0', '1'])  # Update labels for binary classification
cm_tree.plot(cmap='Blues', values_format='.5g')
plt.title("Confusion Matrix for Decision Tree Model with Class Weights")
plt.show()
```

As in Fig.14 I decided to add weights as solution to not making data imbalanced I tried random trials until I have reached a point max of where the true negatives appeared of course the false positives increased in the conv matrix however the data started to balance as in fig 15.

Fig.15

```
Accuracy on training set: 0.806
Accuracy on test set: 0.805
```

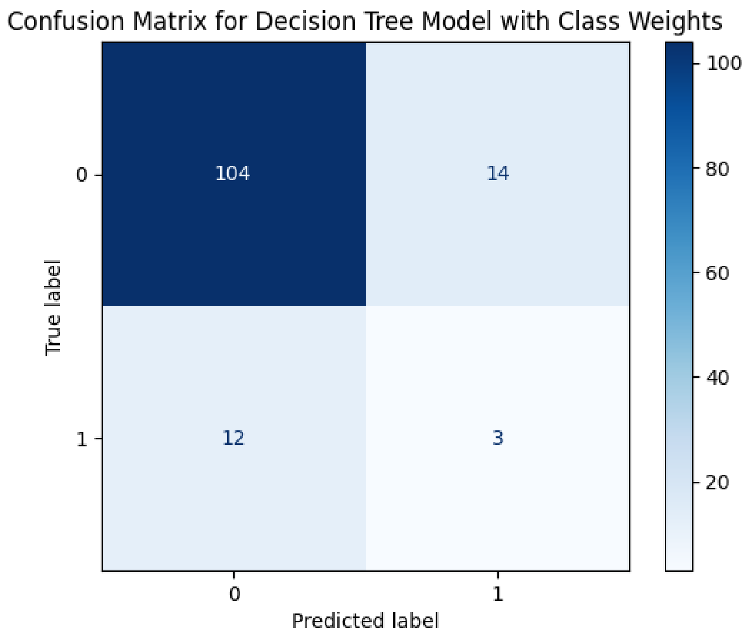
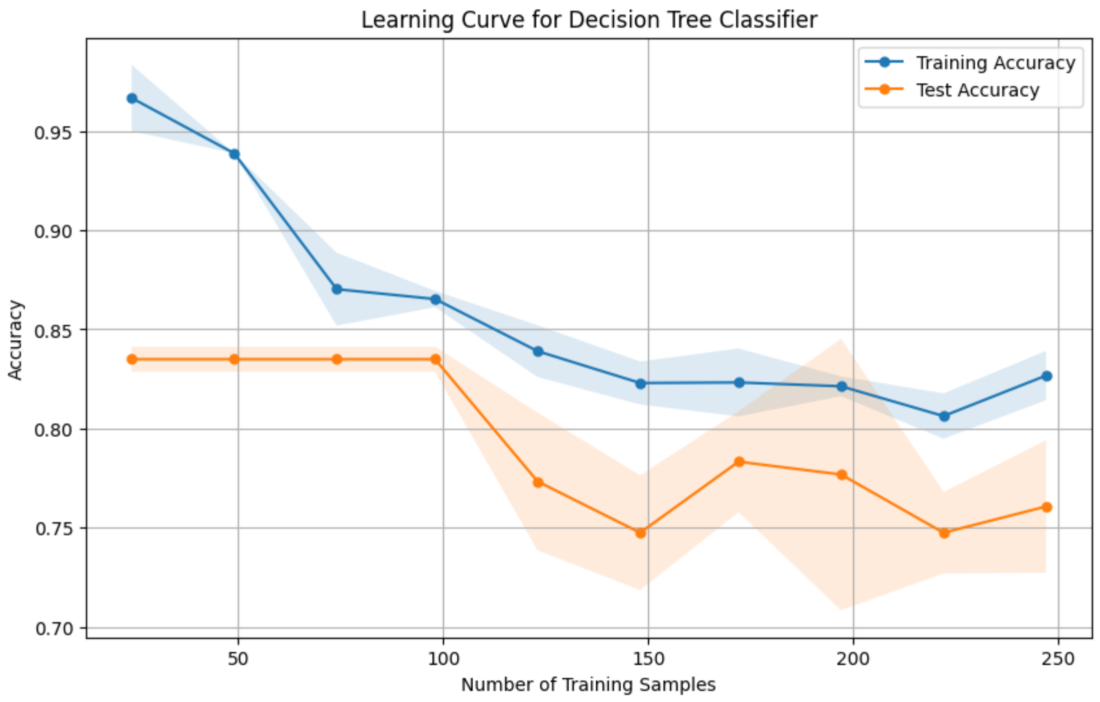


Fig.16



As in Fig.16 it shows that somehow the curve looks decent enough not to overfit or underfit.

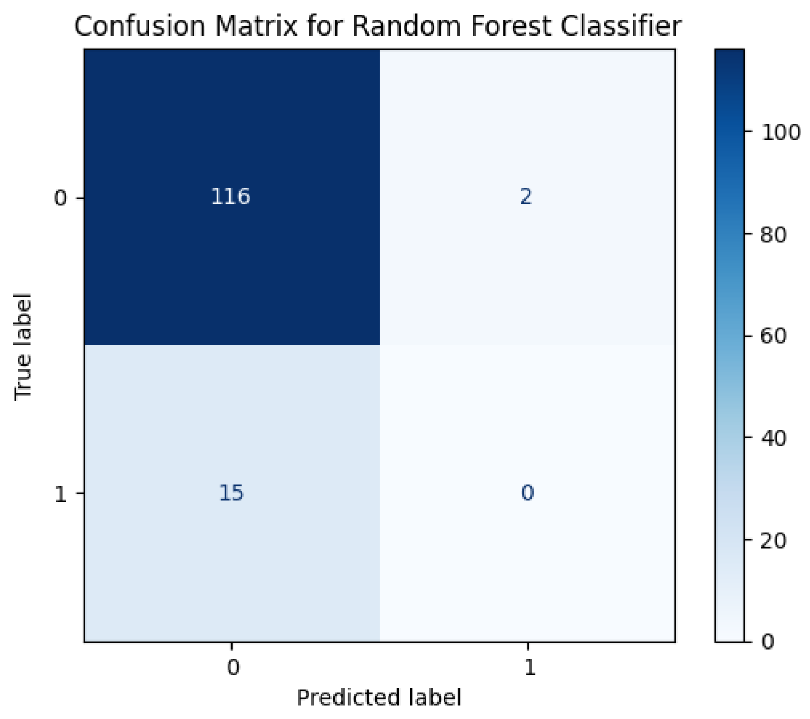
Random Forest:

Fig.17

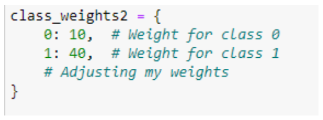
```
rf_classifier0 = RandomForestClassifier(n_estimators=100,max_depth=10, min_samples_split=8, random_state=42)

rf_classifier0.fit(X_train, y_train)


y_train_pred0 = rf_classifier0.predict(X_train)

y_test_pred0 = rf_classifier0.predict(X_test)

train_accuracy = accuracy_score(y_train, y_train_pred0)
print("Training Accuracy: {:.2f}".format(train_accuracy))

test_accuracy = accuracy_score(y_test, y_test_pred0)
print("Test Accuracy: {:.2f}".format(test_accuracy))
```

In the figure above I tuned the parameters until I reached a logical training and accuracy of course using the random forest classifier where it is definitely amazing regarding of overfitting in the data and due to the results came good.

```
Training Accuracy: 0.88
Test Accuracy: 0.87
```



However, the data did the same not seeing the other class thus overfitting, so it is better to do more tuning or maybe add more weights!



```
class_weights2 = {
    0: 10,  # Weight for class 0
    1: 40,  # Weight for class 1
    # Adjusting my weights
}
```

```
Training Accuracy: 0.86
Test Accuracy: 0.81
```

And indeed, after adding weights again the other class started to show up as seen in the confusion matrix it is expected for the accuracy to go down as now the other class started to join in with its different rate.

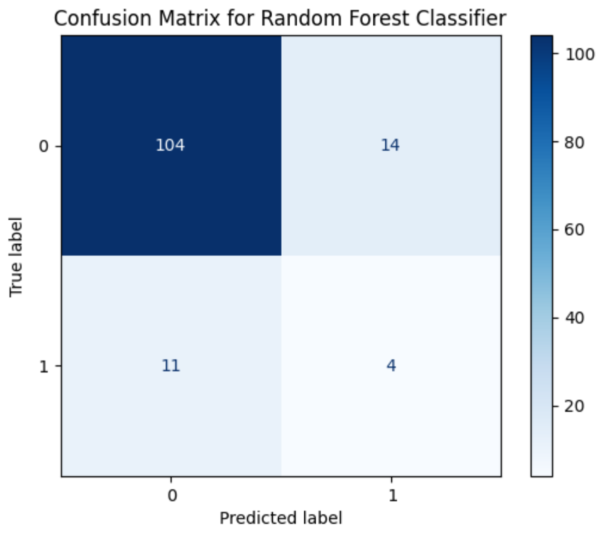
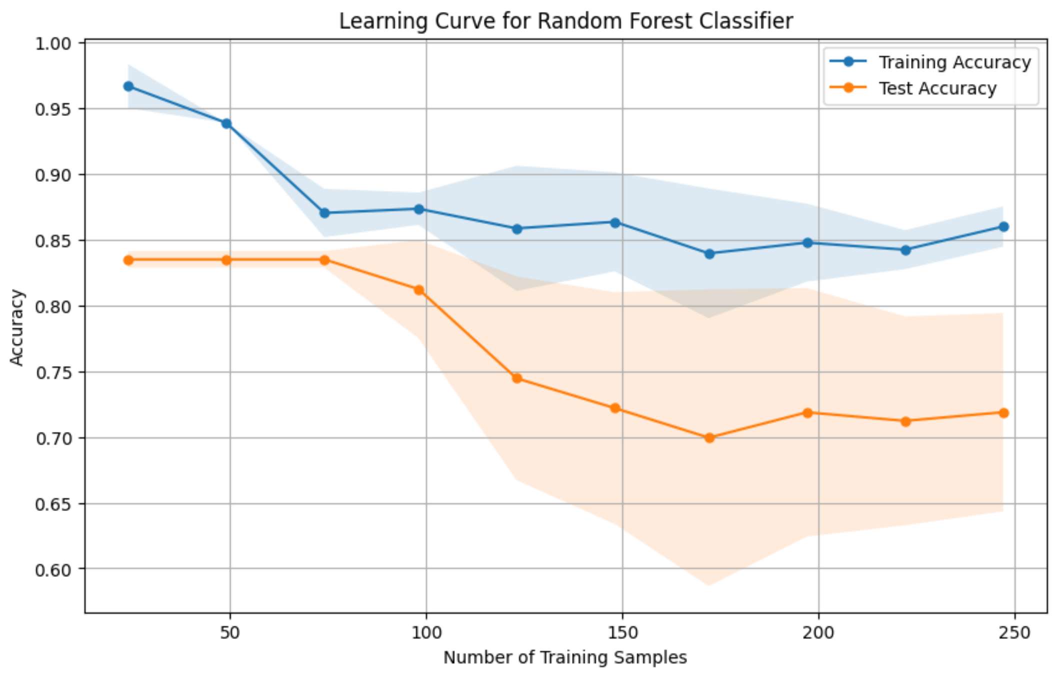


Fig 20.

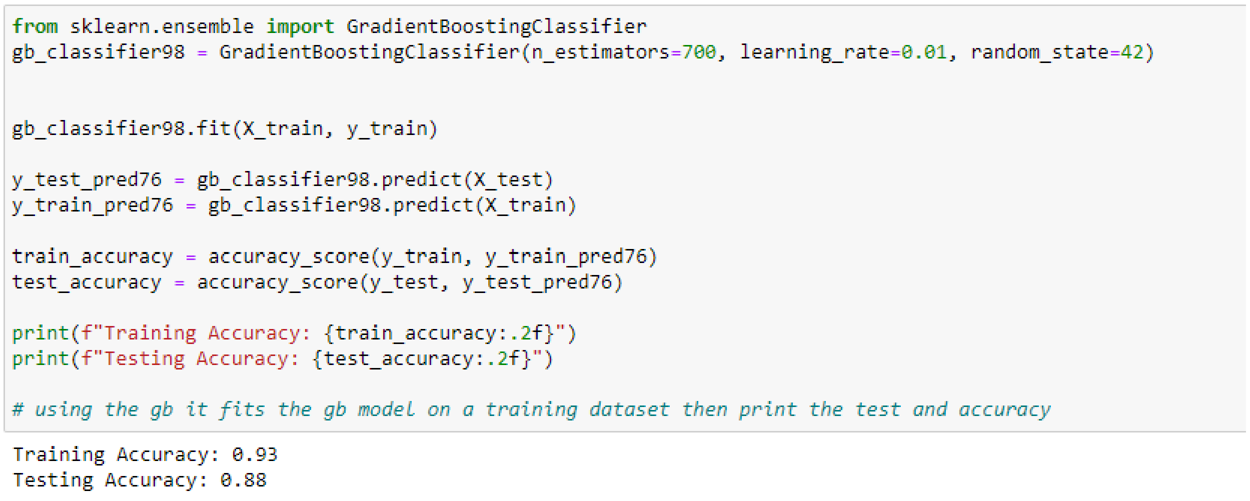


The learning curve seems to be ok as it just a little far away from overfitting in my opinion I think it is good somehow.

Gradient Boost:

The Gradient Boost creates a strong predictive model by combining several weak learners, usually decision trees and yes this might be helpful so why don't we try it out.

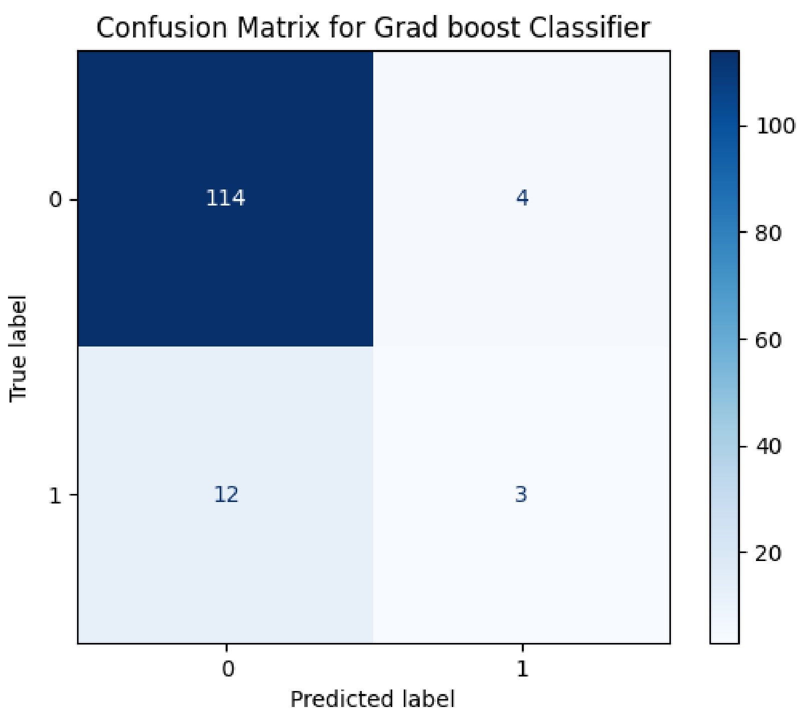
And as seen below in Fig 21.



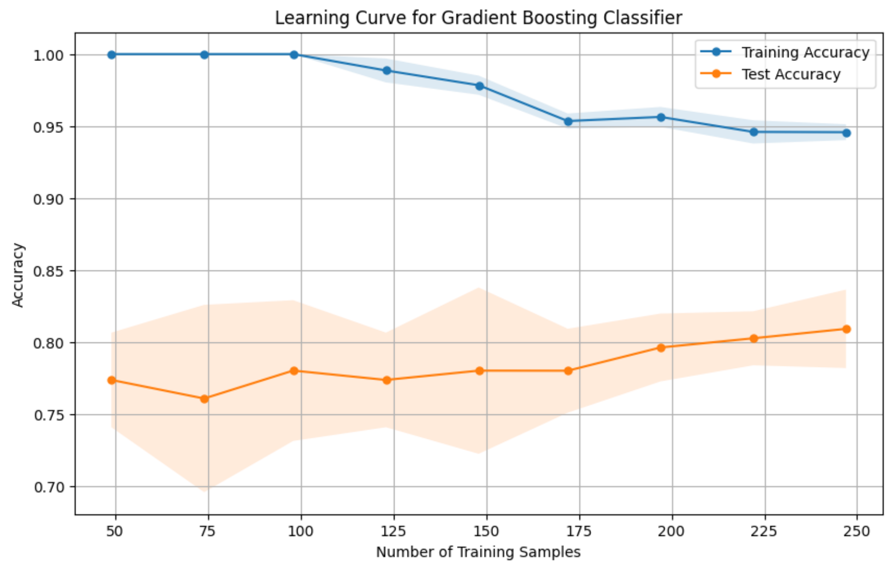
```
from sklearn.ensemble import GradientBoostingClassifier
gb_classifier98 = GradientBoostingClassifier(n_estimators=700, learning_rate=0.01, random_state=42)


gb_classifier98.fit(X_train, y_train)

y_test_pred76 = gb_classifier98.predict(X_test)
y_train_pred76 = gb_classifier98.predict(X_train)

train_accuracy = accuracy_score(y_train, y_train_pred76)
test_accuracy = accuracy_score(y_test, y_test_pred76)

print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Testing Accuracy: {test_accuracy:.2f}")

# using the gb it fits the gb model on a training dataset then print the test and accuracy
```

```
Training Accuracy: 0.93
Testing Accuracy: 0.88
```

The accuracy seems to be impressive however confusion matrix will make it appear more as below

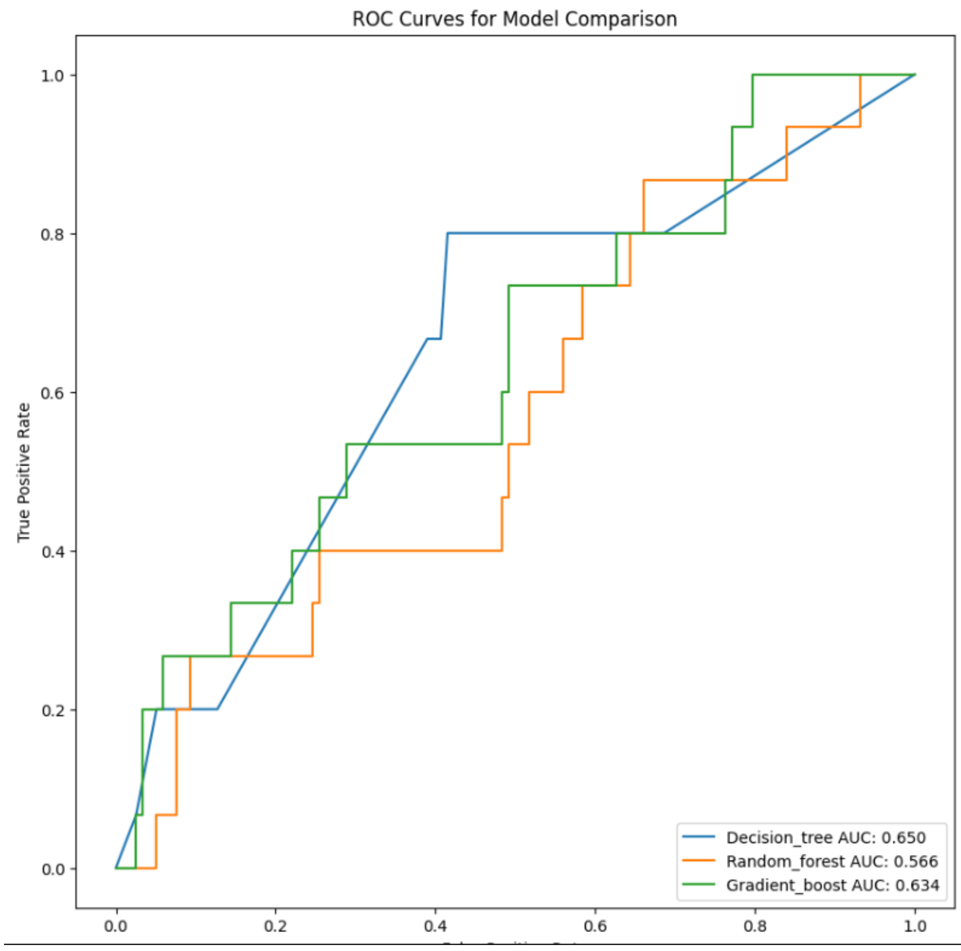


The matrix above is not the best as still indeed there is a lot of false negatives it is better than the rest until now.

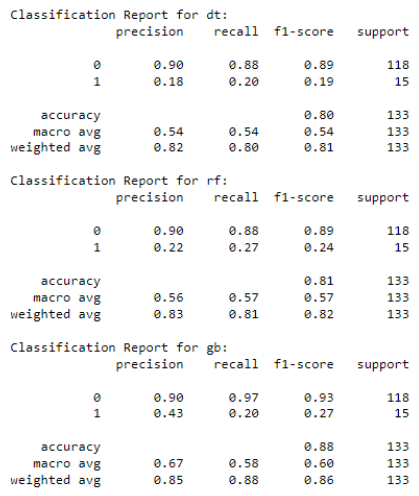


The Learning curve above might be a little closer to overfit however I think it might be somehow acceptable.

The Roc curve below seemed disappointing due to the AUC resulting from the values However the Decision tree seemed to have the best area under curve. When assessing and contrasting classification models, ROC and AUC are both useful metrics, particularly when finding the right balance between true positives and false positives is crucial.



As well as the **Classification report**



```
Classification Report for dt:
              precision    recall  f1-score   support

           0       0.90      0.88      0.89       118
           1       0.18      0.20      0.19        15

    accuracy                           0.80       133
   macro avg       0.54      0.54      0.54       133
weighted avg       0.82      0.80      0.81       133

Classification Report for rf:
              precision    recall  f1-score   support

           0       0.90      0.88      0.89       118
           1       0.22      0.27      0.24        15

    accuracy                           0.81       133
   macro avg       0.56      0.57      0.57       133
weighted avg       0.83      0.81      0.82       133

Classification Report for gb:
              precision    recall  f1-score   support

           0       0.90      0.97      0.93       118
           1       0.43      0.20      0.27        15

    accuracy                           0.88       133
   macro avg       0.67      0.58      0.60       133
weighted avg       0.85      0.88      0.86       133
```

Where in the figure above By calculating the ratio of real positive predictions to all positive predictions, **precision** evaluates the accuracy of positive predictions.

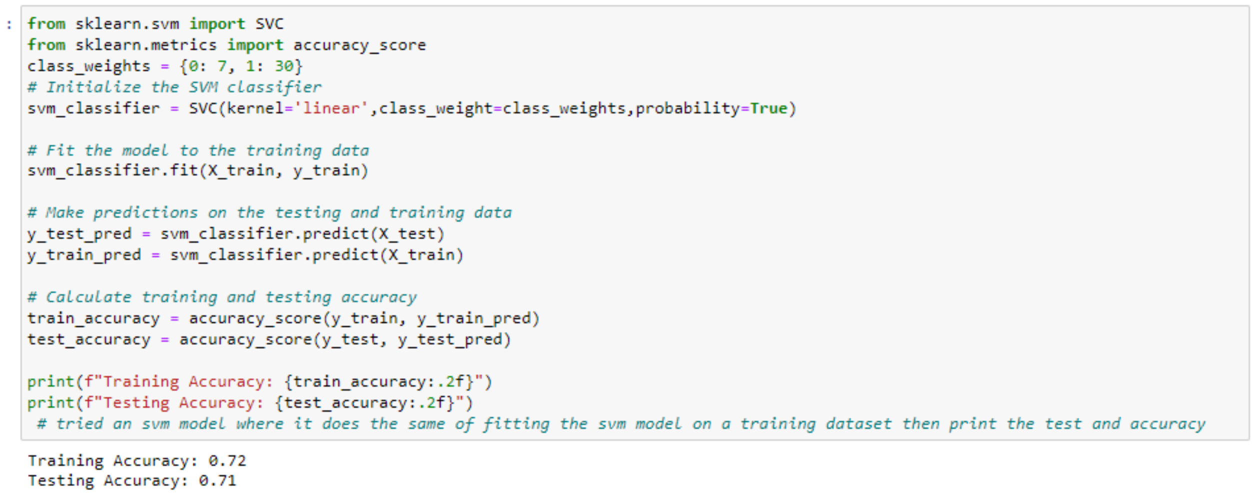
By taking into account the ratio of true positives to the overall number of actual positives, **recall** evaluates the model's capacity to accurately identify all genuine positive events.

The **F1-Score** offers a fair evaluation of a model's performance by integrating precision and recall into a single score that represents their harmonic meaning.

Seeing that I decided to add another model to them hoping it might score better which is the SVM model.

SVM Model:

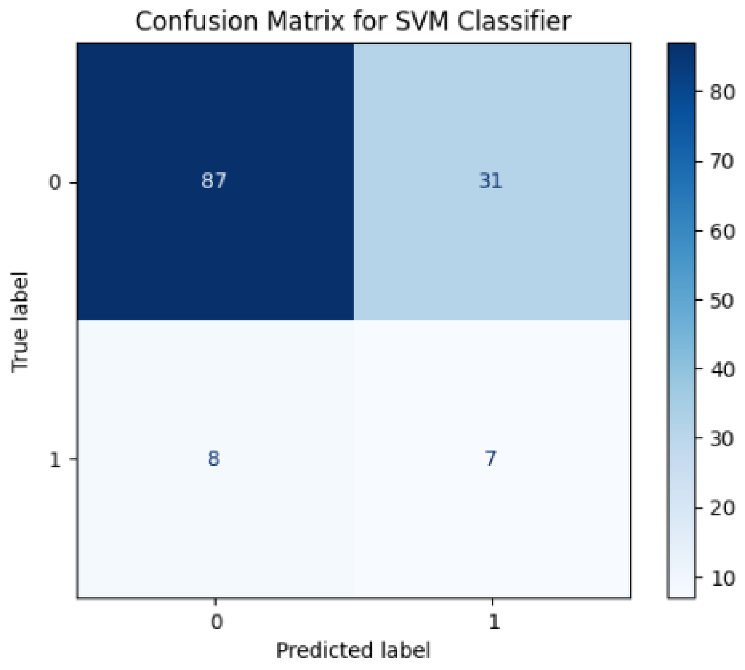
SVM model functions by identifying the most appropriate hyperplane for classifying the data into distinct groups.



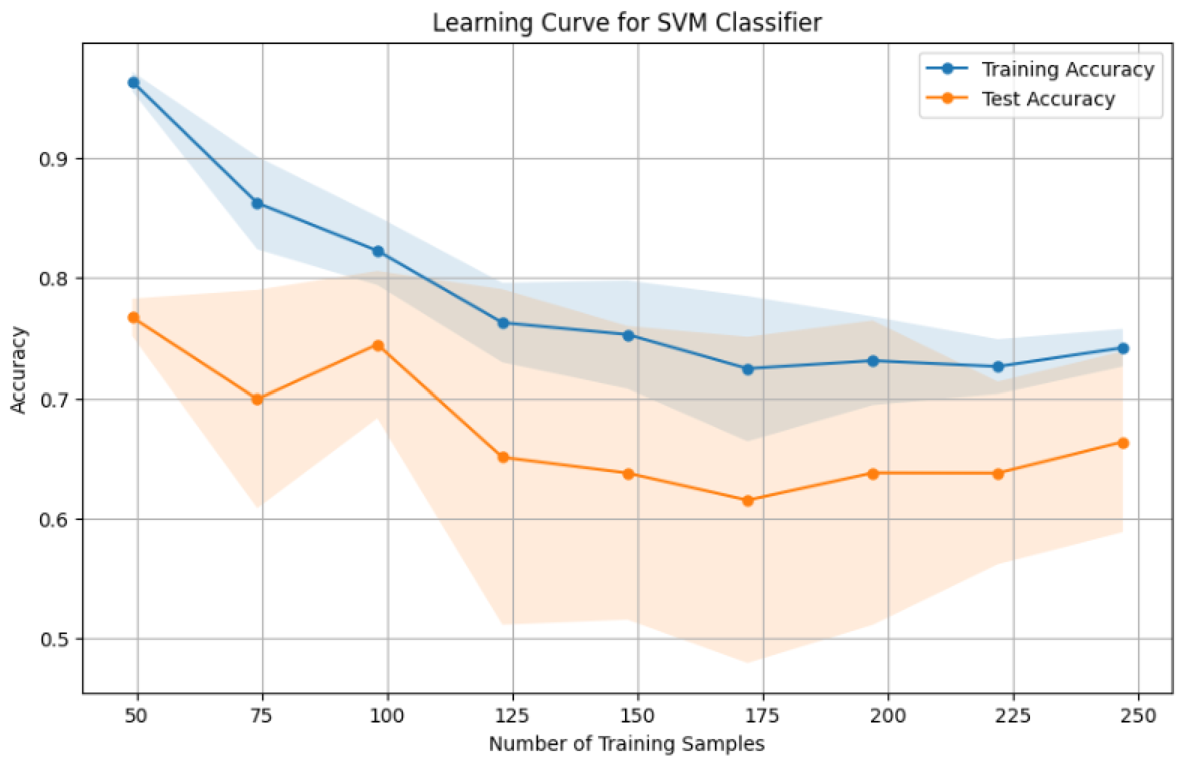
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
class_weights = {0: 7, 1: 30}
# Initialize the SVM classifier
svm_classifier = SVC(kernel='linear',class_weight=class_weights,probability=True)

# Fit the model to the training data
svm_classifier.fit(X_train, y_train)

# Make predictions on the testing and training data
y_test_pred = svm_classifier.predict(X_test)
y_train_pred = svm_classifier.predict(X_train)

# Calculate training and testing accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Testing Accuracy: {test_accuracy:.2f}")
 # tried an svm model where it does the same of fitting the svm model on a training dataset then print the test and accuracy
```

```
Training Accuracy: 0.72
Testing Accuracy: 0.71
```

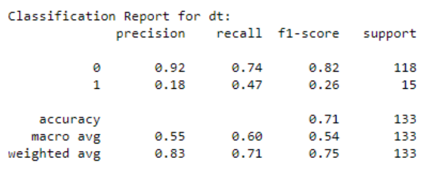
I have also included weights here due to the same problem that faced the others after trying it and also I choose the weights based on trial and error and as seen below it has a great improvement as more data started getting recognized based on more tuning in the data as said.



The learning curve looks good as it doesn't overfit or underfit anyways and the opposite accuracy point are closer to each other.

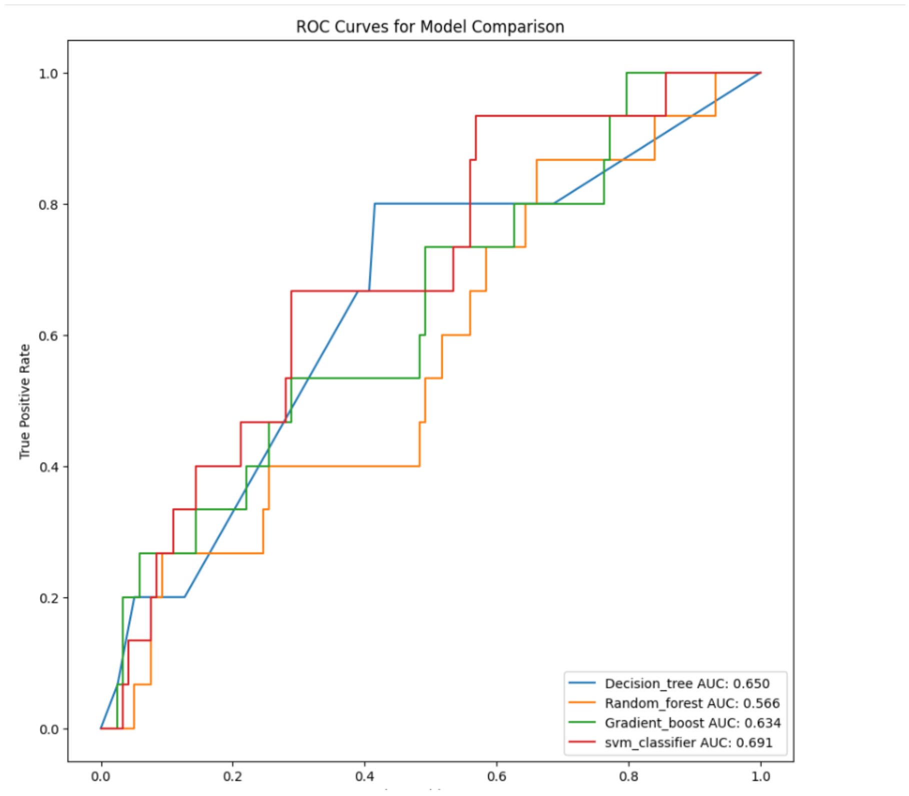


The report looked better than the rest relating to both the classes not one.



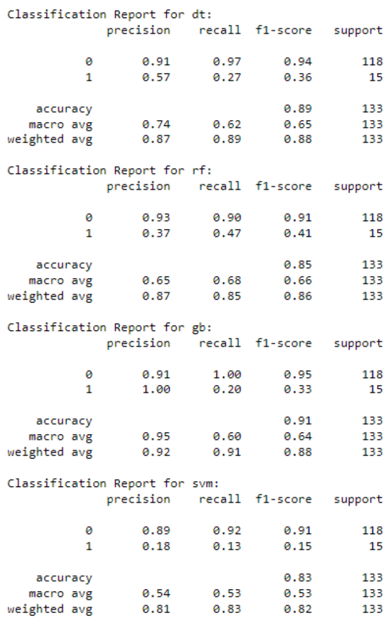
```
Classification Report for dt:
              precision    recall  f1-score   support

           0       0.92      0.74      0.82       118
           1       0.18      0.47      0.26        15

    accuracy                           0.71       133
   macro avg       0.55      0.60      0.54       133
weighted avg       0.83      0.71      0.75       133
```

And yes indeed it was the best in the ROC curve and the AUC comparing the other 3 models with The SVM.

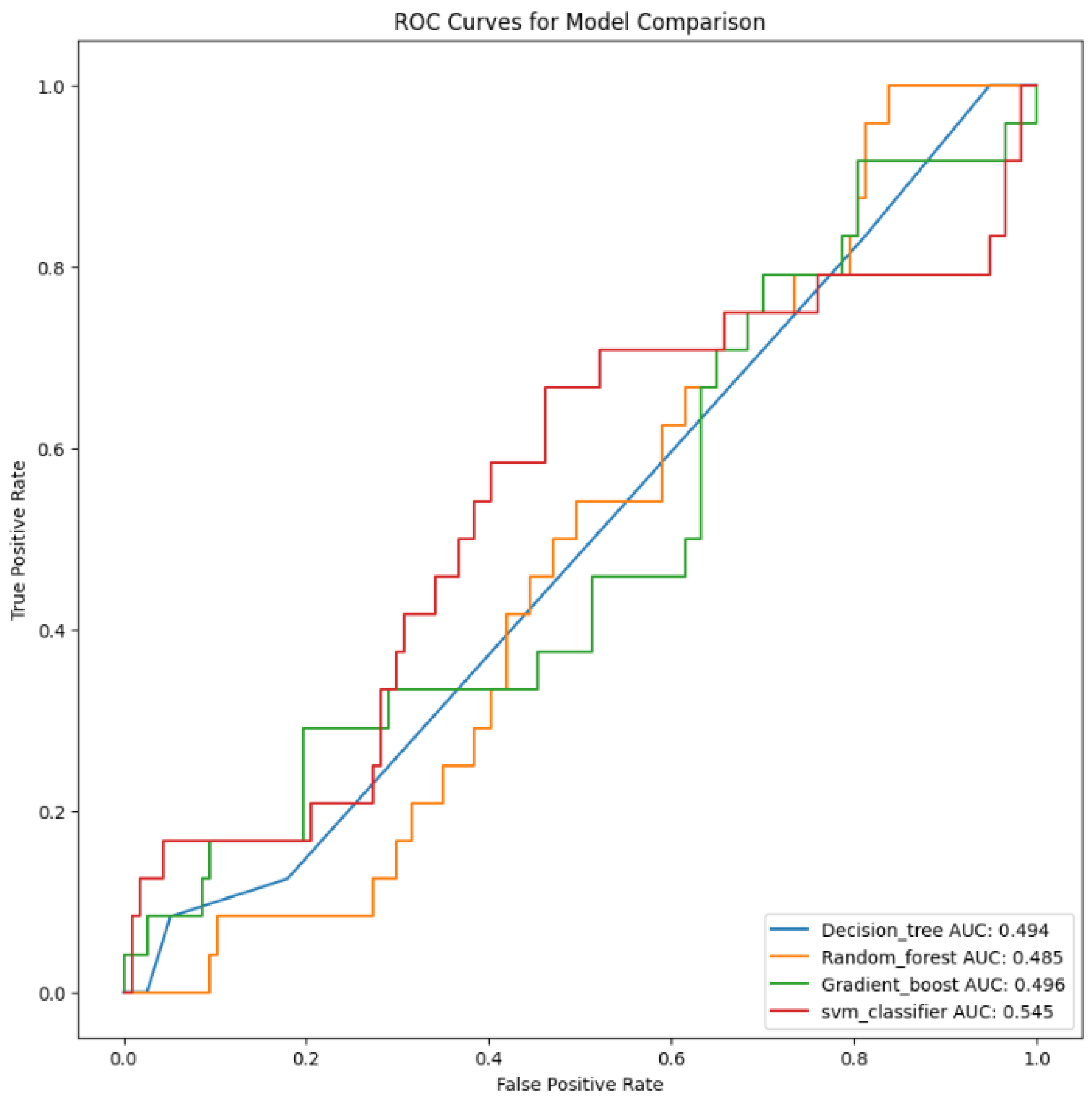


Models without removing outliers

And as I mentioned in the beginning, I tried the data but without the outliers and that is what I reached in figures I will only include the Roc and the classification of what I reached.



```
Classification Report for dt:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94       118
           1       0.57      0.27      0.36        15

    accuracy                           0.89       133
   macro avg       0.74      0.62      0.65       133
weighted avg       0.87      0.89      0.88       133

Classification Report for rf:
              precision    recall  f1-score   support

           0       0.93      0.90      0.91       118
           1       0.37      0.47      0.41        15

    accuracy                           0.85       133
   macro avg       0.65      0.68      0.66       133
weighted avg       0.87      0.85      0.86       133

Classification Report for gb:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       118
           1       1.00      0.20      0.33        15

    accuracy                           0.91       133
   macro avg       0.95      0.60      0.64       133
weighted avg       0.92      0.91      0.88       133

Classification Report for svm:
              precision    recall  f1-score   support

           0       0.89      0.92      0.91       118
           1       0.18      0.13      0.15        15

    accuracy                           0.83       133
   macro avg       0.54      0.53      0.53       133
weighted avg       0.81      0.83      0.82       133
```



And yet again the SVM was the best model between ass regarding the ROC in the data without removing the outlier however the values of the AUC were very low.

All in All, removing the outlier was better relating to the confusion matrix and accuracies and also the ROC/AUC and I found out that the SVM was the better in both in classifying. However, none were that impressive as in the end many data left unrecognized I was unable to increase the accuracies more or make the values recognize each other perfectly.

Abdulrahman Ayman 211896

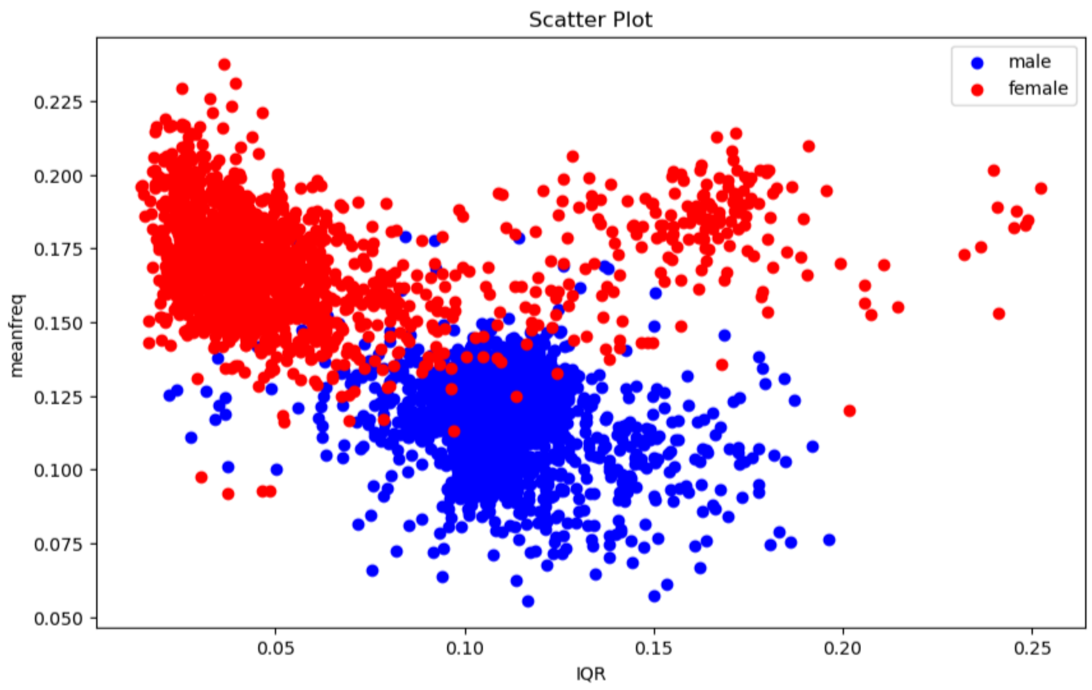
1. Data Description:

This dataset was created to determine a voice's gender based on the speech and voice's acoustic characteristics. Many acoustic features that were extracted from voice recordings make up the dataset. Sorting the speakers into male and female categories is the dataset's primary objective. The dataset consists of 3,168 recorded speech samples from male and female speakers. This dataset has a target label called (label) that is consisting of 2 classes [male, female]. The dataset consists of 21 columns and 3168 rows.

Figure1: this figure shows the distribution of the labels (male, female) of the datasets according to the mean frequency feature as the y-axis and the IQR (interquartile range) feature as x-axis.

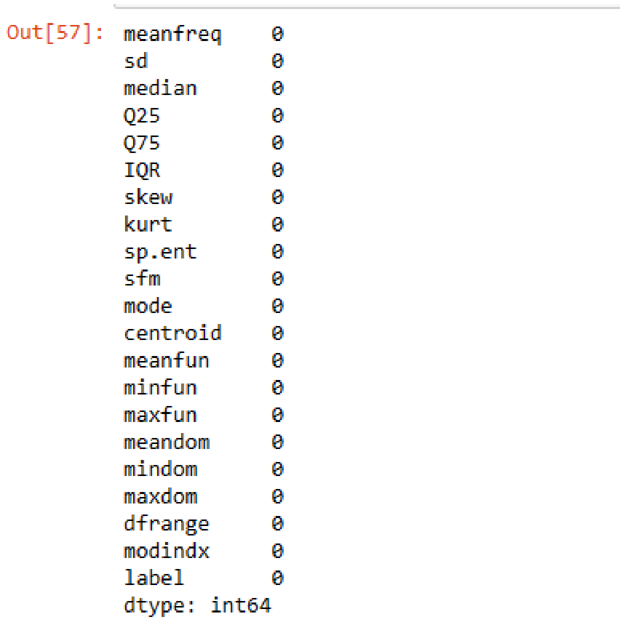
According to this figure, most female voices have mean frequencies greater than 0.125 and IQRs lower than 0.15.

Most male voices have IQR greater than 0.1 and mean frequencies less than 0.125.



2. Data Preprocessing:

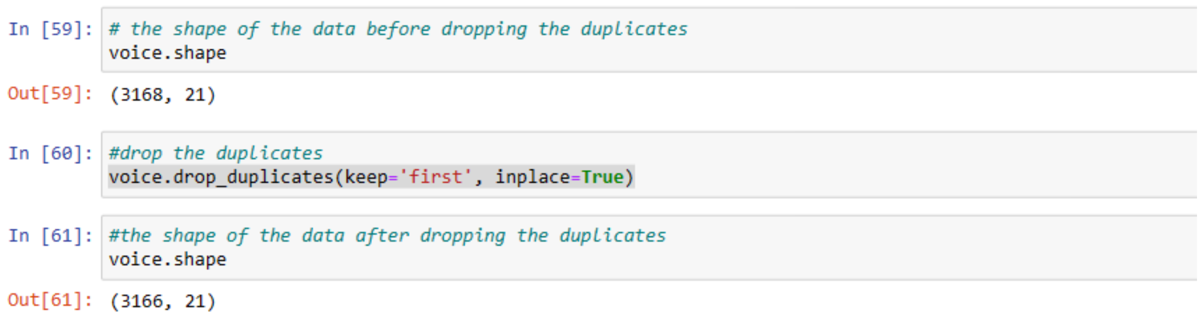
- Nulls: the dataset contains no null values.



```
Out[57]: meanfreq    0
         sd          0
         median      0
         Q25         0
         Q75         0
         IQR         0
         skew        0
         kurt        0
         sp.ent      0
         sfm         0
         mode        0
         centroid    0
         meanfun     0
         minfun      0
         maxfun      0
         meandom     0
         mindom      0
         maxdom      0
         dfrange     0
         modindx     0
         label       0
         dtype: int64
```

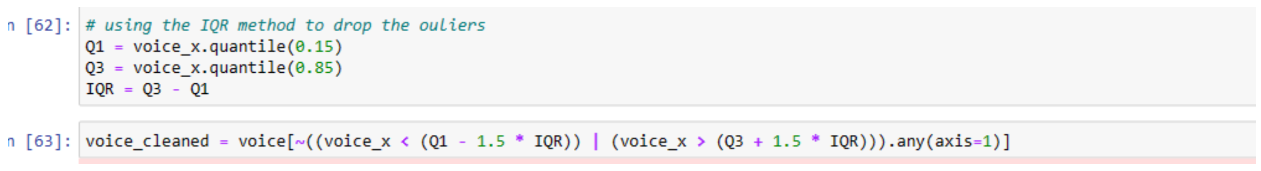
- Duplicates: The dataset contains 2 duplicated records which are dropped keeping one of the duplicates.

The shape of the data before and after removing the duplicates.



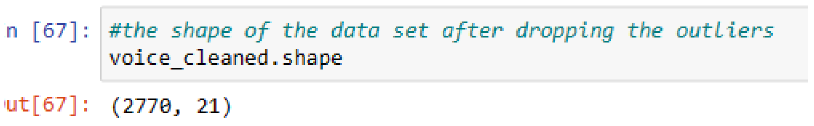
```
In [59]: # the shape of the data before dropping the duplicates
         voice.shape

Out[59]: (3168, 21)

In [60]: #drop the duplicates
         voice.drop_duplicates(keep='first', inplace=True)

In [61]: #the shape of the data after dropping the duplicates
         voice.shape

Out[61]: (3166, 21)
```

- Outliers: using the IQR method to drop the outliers with 0.15 for Q1 and 0.85 for Q3



```
n [62]: # using the IQR method to drop the ouliers
        Q1 = voice_x.quantile(0.15)
        Q3 = voice_x.quantile(0.85)
        IQR = Q3 - Q1

n [63]: voice_cleaned = voice[~((voice_x < (Q1 - 1.5 * IQR)) | (voice_x > (Q3 + 1.5 * IQR))).any(axis=1)]
```

And this is the shape of the dataset after removing the outliers.



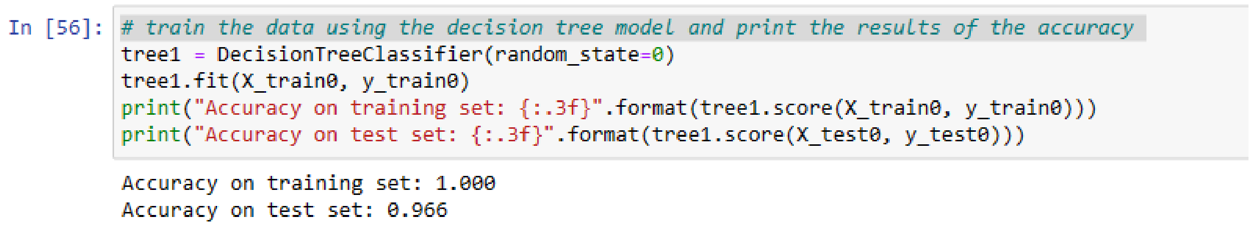
```
n [67]: #the shape of the data set after dropping the outliers
        voice_cleaned.shape

ut[67]: (2770, 21)
```

3. **Models:**

3.1. Decision Tree:

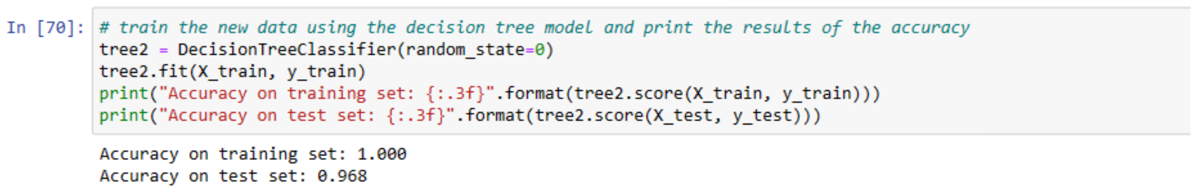
We used the decision Tree model for training before and after the preprocessing on the data to show the improvement that happened after making the preprocessing on the data.

3.1.1. According to this figure, before the preprocessing the accuracy of the training is 1 and accuracy of the testing is 0.966.



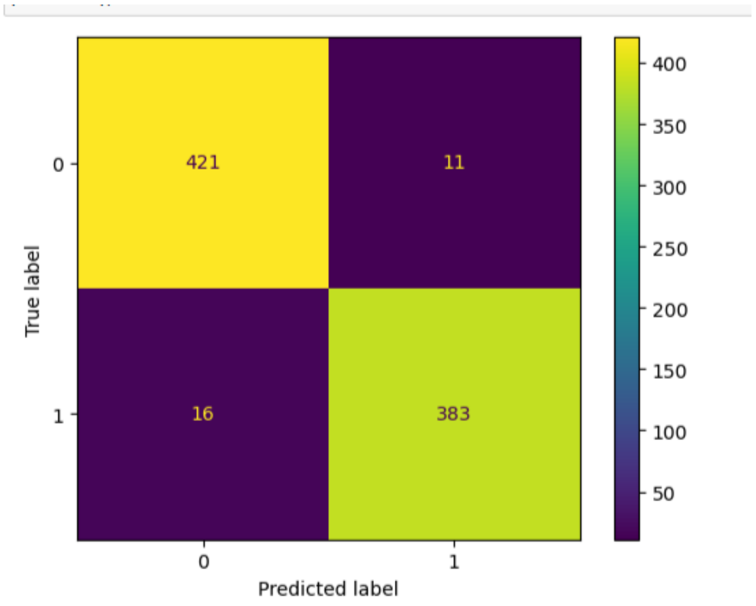
```
In [56]: # train the data using the decision tree model and print the results of the accuracy
         tree1 = DecisionTreeClassifier(random_state=0)
         tree1.fit(X_train0, y_train0)
         print("Accuracy on training set: {:.3f}".format(tree1.score(X_train0, y_train0)))
         print("Accuracy on test set: {:.3f}".format(tree1.score(X_test0, y_test0)))

         Accuracy on training set: 1.000
         Accuracy on test set: 0.966
```

3.1.2. According to this figure, after the preprocessing on the data the testing accuracy improved to 0.968 and the accuracy of the training is still the same, which is 1.



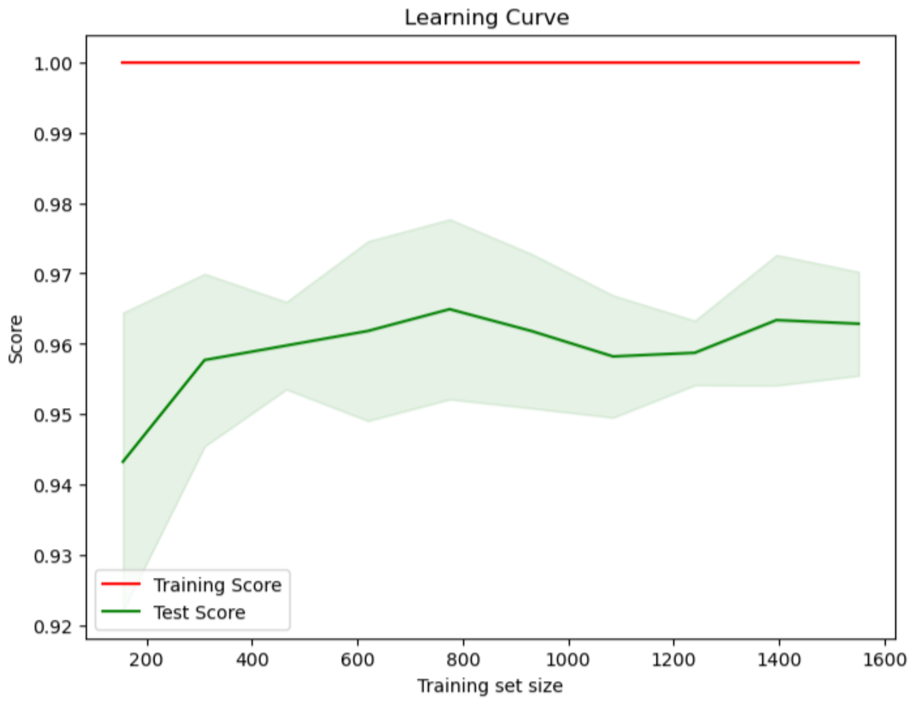
```
In [70]: # train the new data using the decision tree model and print the results of the accuracy
         tree2 = DecisionTreeClassifier(random_state=0)
         tree2.fit(X_train, y_train)
         print("Accuracy on training set: {:.3f}".format(tree2.score(X_train, y_train)))
         print("Accuracy on test set: {:.3f}".format(tree2.score(X_test, y_test)))

         Accuracy on training set: 1.000
         Accuracy on test set: 0.968
```

This figure shows the confusion matrix of the decision tree after the preprocessing. The confusion matrix shows the number of true positives that predicted by the model is 383 and the number of true negatives that predicted by the model is 421.



This figure is the learning curve of the decision tree model after the preprocessing.

This learning curve shows how the model training and test accuracy affected over the change of the training set size.



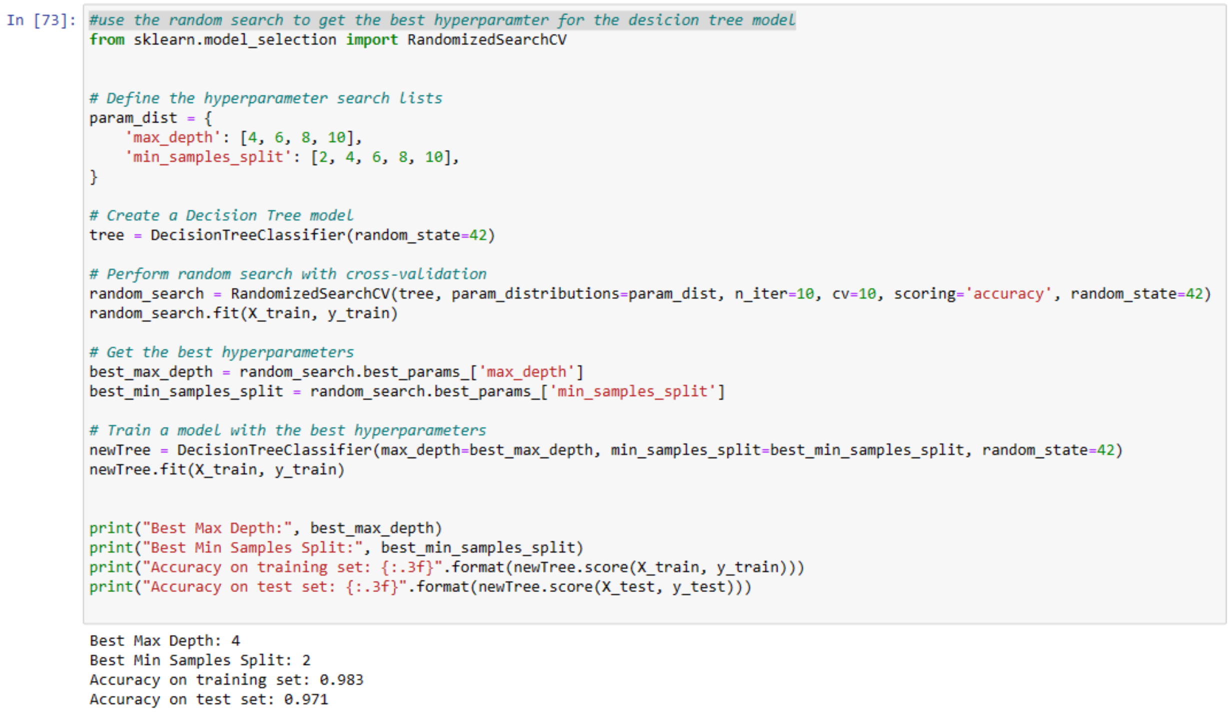
3.1.3. Decision tree model with hyper parameters:

In this model we add hyperparameters (max depth, min sample split) to improve the results of the last model.

max depth: it determines the depth or the levels of the tree, it restricts how deep the tree can grow.

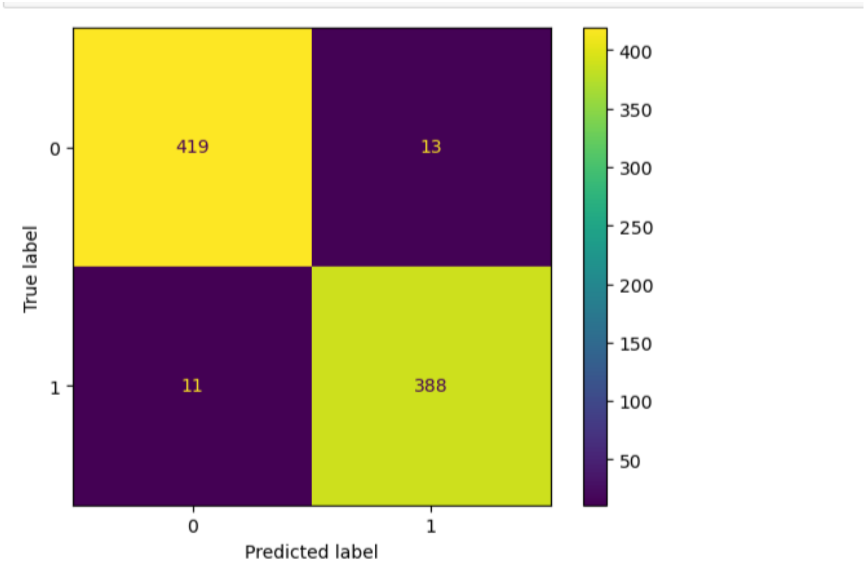
Min sample split: It determines the minimum number of samples required to split a node in the decision tree. If a node contains fewer samples than min samples split, it won't be split further.

According to this figure, I created a list for max depth and a list for min sample split that contain some values and use the random search to get the best max depth and the best min sample split values. According to the search the best max depth is 4 and the best min sample split is 2, and when I use these values to train the model the results of the test accuracy improved to 0.971 which is better than the tree without hyperparameters and the training accuracy become 0.983.



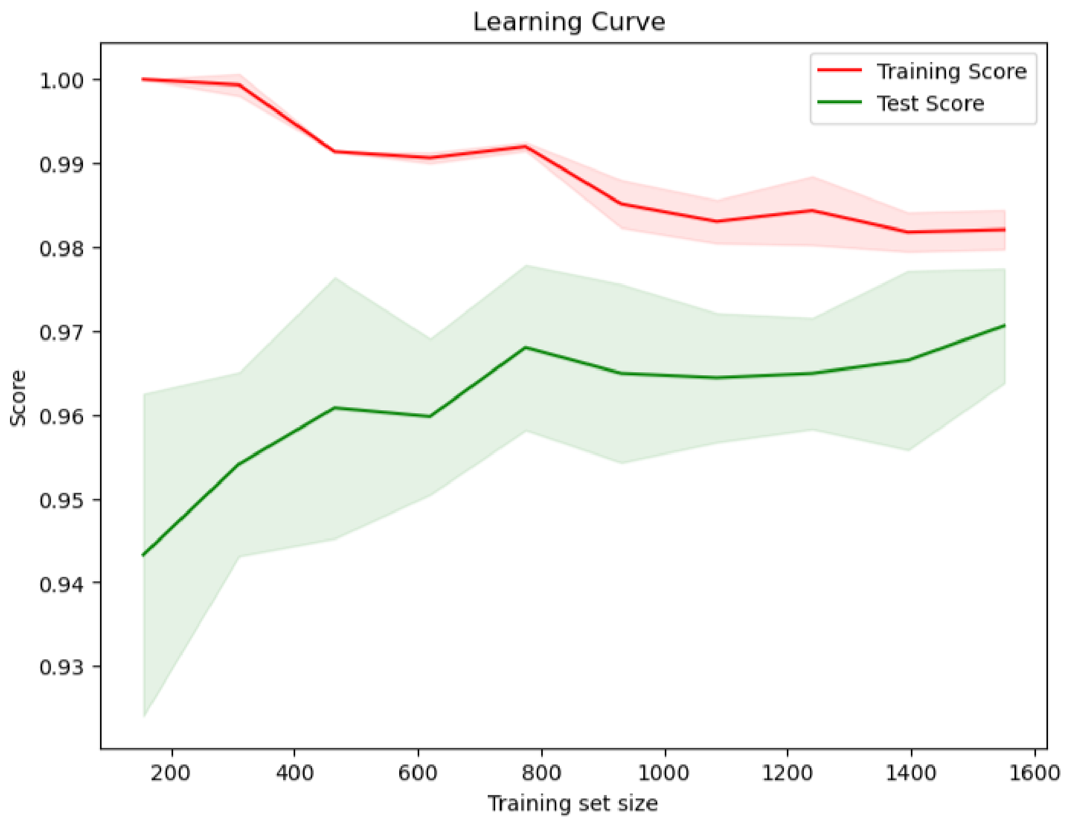
```
In [73]: #use the random search to get the best hyperparamter for the desicion tree model
         from sklearn.model_selection import RandomizedSearchCV


         # Define the hyperparameter search lists
         param_dist = {
             'max_depth': [4, 6, 8, 10],
             'min_samples_split': [2, 4, 6, 8, 10],
         }

         # Create a Decision Tree model
         tree = DecisionTreeClassifier(random_state=42)

         # Perform random search with cross-validation
         random_search = RandomizedSearchCV(tree, param_distributions=param_dist, n_iter=10, cv=10, scoring='accuracy', random_state=42)
         random_search.fit(X_train, y_train)

         # Get the best hyperparameters
         best_max_depth = random_search.best_params_['max_depth']
         best_min_samples_split = random_search.best_params_['min_samples_split']

         # Train a model with the best hyperparameters
         newTree = DecisionTreeClassifier(max_depth=best_max_depth, min_samples_split=best_min_samples_split, random_state=42)
         newTree.fit(X_train, y_train)


         print("Best Max Depth:", best_max_depth)
         print("Best Min Samples Split:", best_min_samples_split)
         print("Accuracy on training set: {:.3f}".format(newTree.score(X_train, y_train)))
         print("Accuracy on test set: {:.3f}".format(newTree.score(X_test, y_test)))

         Best Max Depth: 4
         Best Min Samples Split: 2
         Accuracy on training set: 0.983
         Accuracy on test set: 0.971
```

This figure shows the confusion matrix of the new tree with the hyperparameters. The confusion matrix shows the number of true positives that predicted by the model is 388 and the number of true negatives that predicted by the model is 419.



This figure is the learning curve of the new tree with hyper parameters.

This learning curve shows how the model training and test accuracy affected over the change of the training set size.

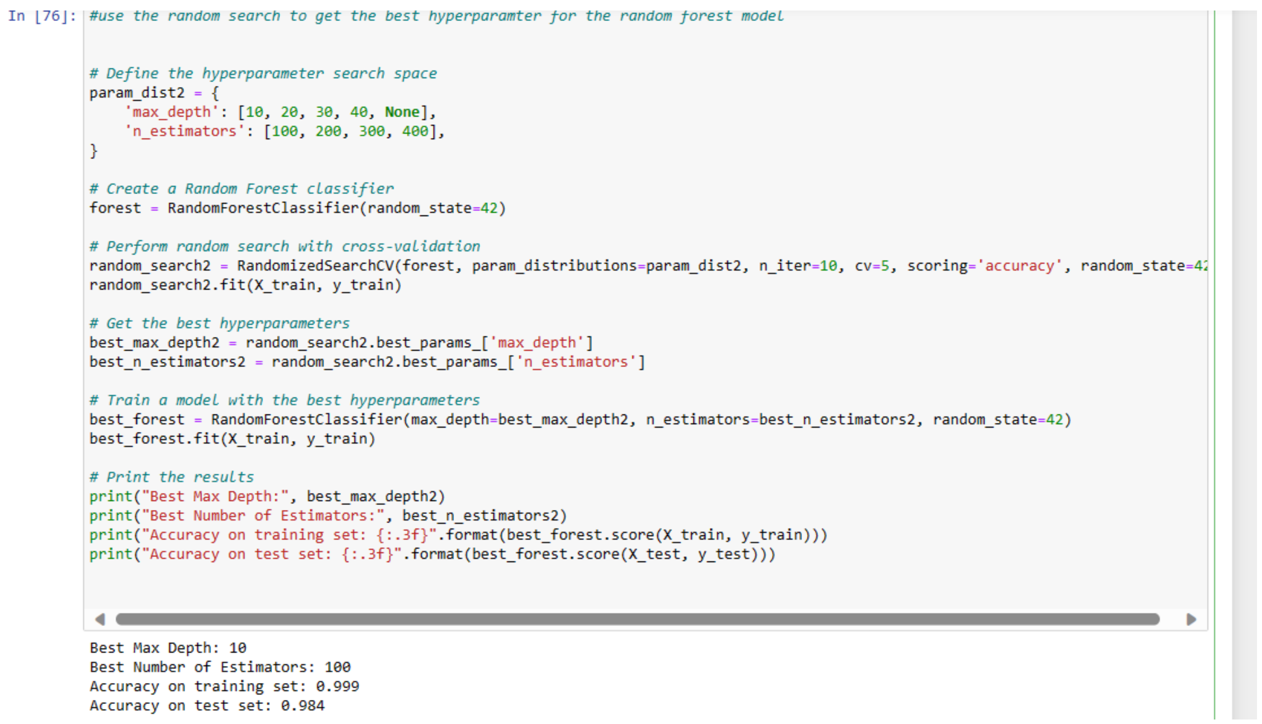


3.2. Random Forest:

I used the random forest model for the training.

Number of estimators: hyperparameter that specifies the number of decision trees that will be created in the ensemble. Each decision tree is built independently from a randomly selected subset of the training data, and the final prediction is made by aggregating the results of these individual trees.

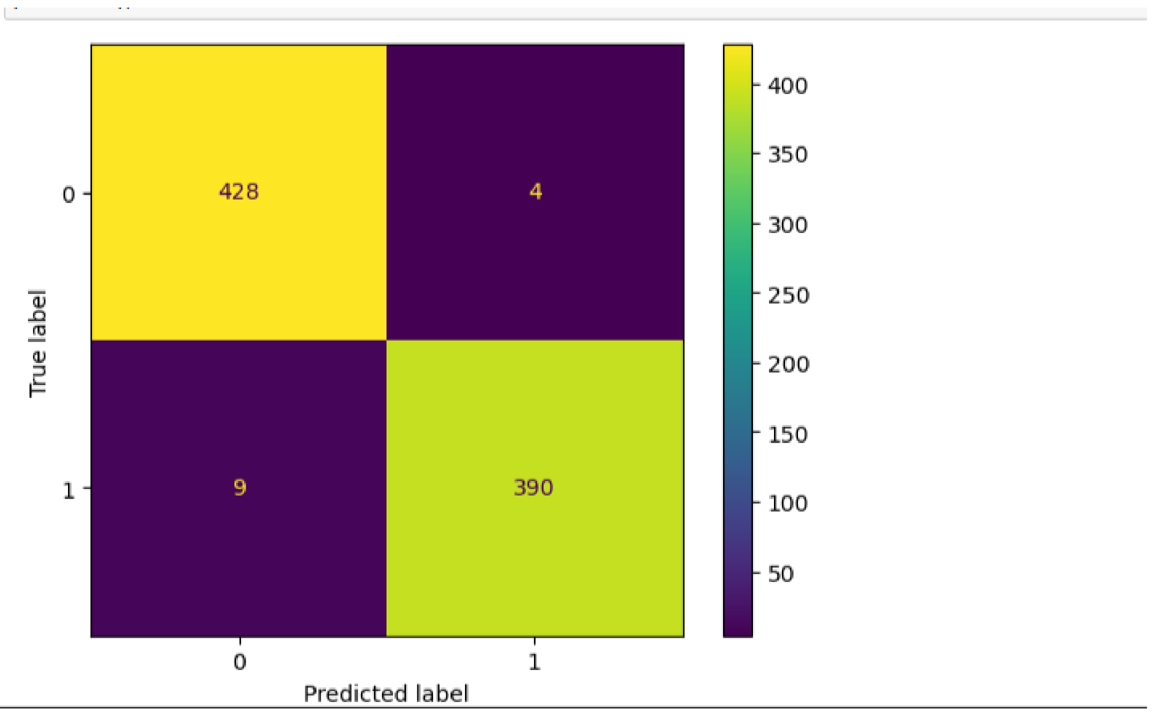
According to this figure, I created a list for max depth and a list for number of estimators that contain some values and used the random search to get the best max depth and the best number of estimators values. According to the search the best max depth is 10 and the best number of estimators is 100. When I use these values to train the model the results of the test accuracy improved to 0.984 which is better than the tree without hyperparameters and the training accuracy become 0.999.



```
In [76]: #use the random search to get the best hyperparamter for the random forest model


         # Define the hyperparameter search space
         param_dist2 = {
             'max_depth': [10, 20, 30, 40, None],
             'n_estimators': [100, 200, 300, 400],
         }

         # Create a Random Forest classifier
         forest = RandomForestClassifier(random_state=42)

         # Perform random search with cross-validation
         random_search2 = RandomizedSearchCV(forest, param_distributions=param_dist2, n_iter=10, cv=5, scoring='accuracy', random_state=42
         random_search2.fit(X_train, y_train)

         # Get the best hyperparameters
         best_max_depth2 = random_search2.best_params_['max_depth']
         best_n_estimators2 = random_search2.best_params_['n_estimators']

         # Train a model with the best hyperparameters
         best_forest = RandomForestClassifier(max_depth=best_max_depth2, n_estimators=best_n_estimators2, random_state=42)
         best_forest.fit(X_train, y_train)

         # Print the results
         print("Best Max Depth:", best_max_depth2)
         print("Best Number of Estimators:", best_n_estimators2)
         print("Accuracy on training set: {:.3f}".format(best_forest.score(X_train, y_train)))
         print("Accuracy on test set: {:.3f}".format(best_forest.score(X_test, y_test)))
```

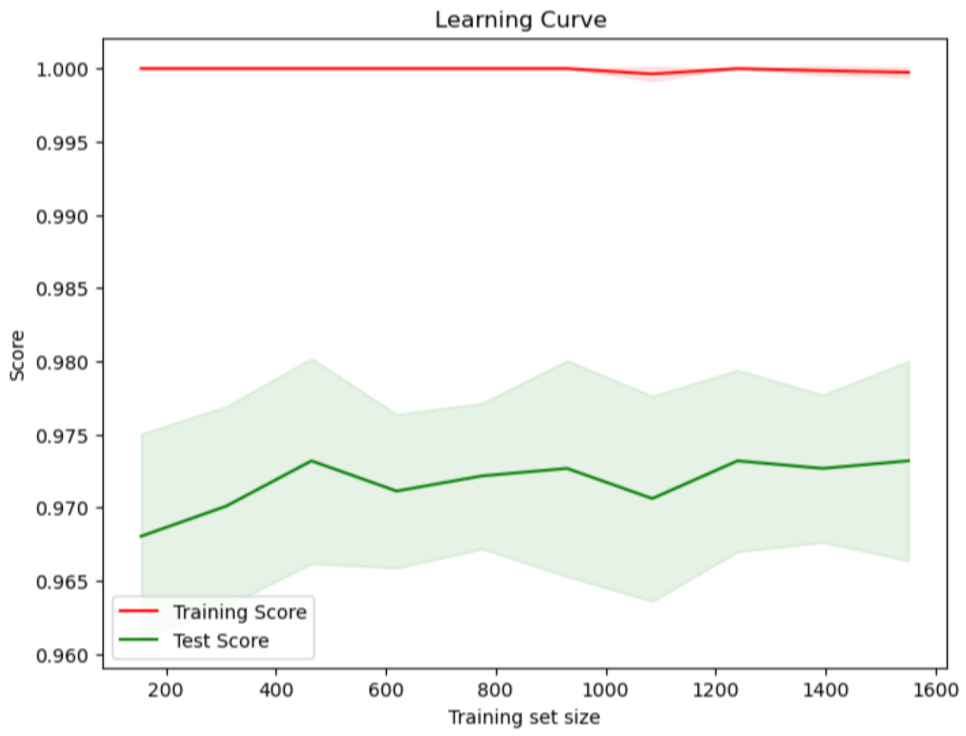
```
Best Max Depth: 10
Best Number of Estimators: 100
Accuracy on training set: 0.999
Accuracy on test set: 0.984
```

This figure shows the confusion matrix of the random forest model. The confusion matrix shows the number of true positives that predicted by the model is 390 and the number of true negatives that predicted by the model is 428.



This figure is the learning curve of the random forest model.

This learning curve shows how the model training and test accuracy affected over the change of the training set size.

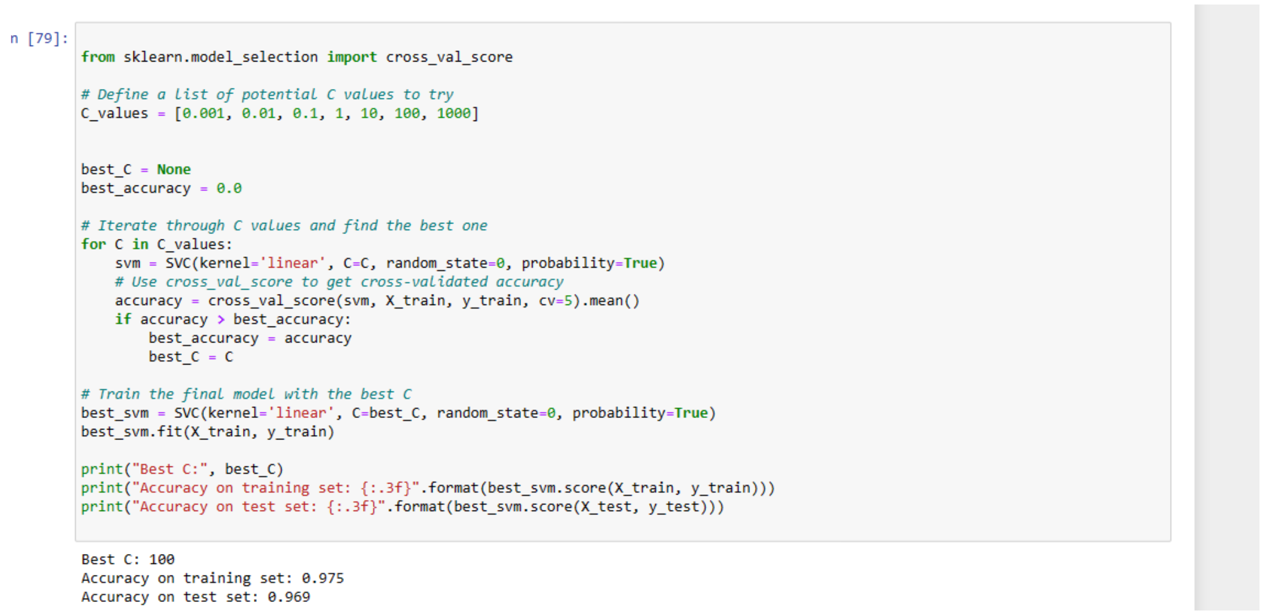


3.3. Support Vector Machine (SVM):

I used the SVM model for the training.

the "C" parameter, often referred to as the regularization parameter, is a crucial hyperparameter that controls the trade-off between maximizing the margin (distance between the decision boundary and the support vectors) and minimizing the classification error.

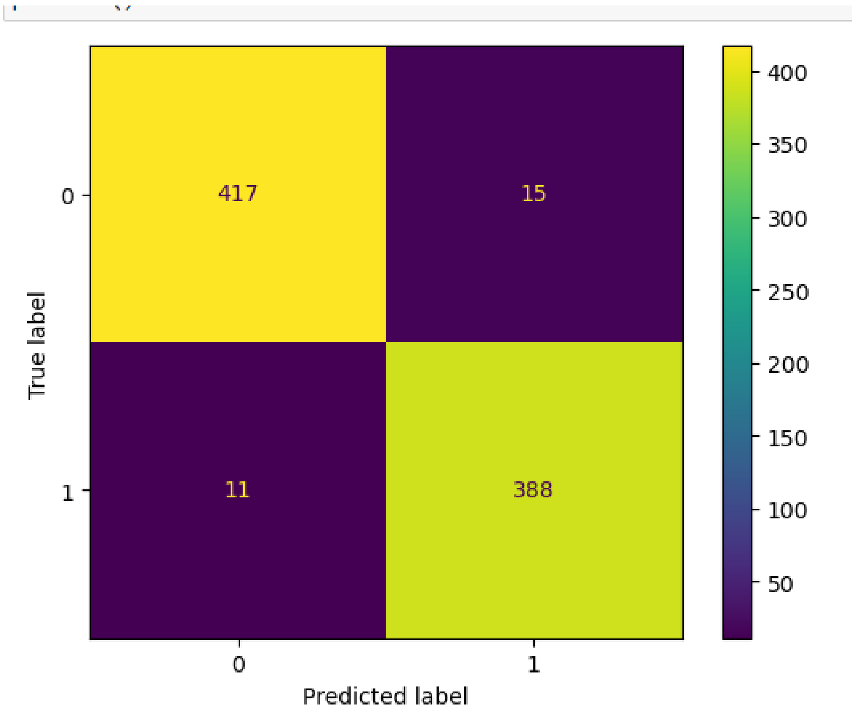
According to this figure, I created a list for C values and used a for loop to get the best C value. According to the search the best C value is 100. When I use this value to train the model and the results of the test accuracy is 0.969 and the training accuracy became 0.975 which Is lower than the random forest and the decision tree models.



```
n [79]:
from sklearn.model_selection import cross_val_score

# Define a list of potential C values to try
C_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]


best_C = None
best_accuracy = 0.0

# Iterate through C values and find the best one
for C in C_values:
    svm = SVC(kernel='linear', C=C, random_state=0, probability=True)
    # Use cross_val_score to get cross-validated accuracy
    accuracy = cross_val_score(svm, X_train, y_train, cv=5).mean()
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_C = C

# Train the final model with the best C
best_svm = SVC(kernel='linear', C=best_C, random_state=0, probability=True)
best_svm.fit(X_train, y_train)

print("Best C:", best_C)
print("Accuracy on training set: {:.3f}".format(best_svm.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(best_svm.score(X_test, y_test)))
```

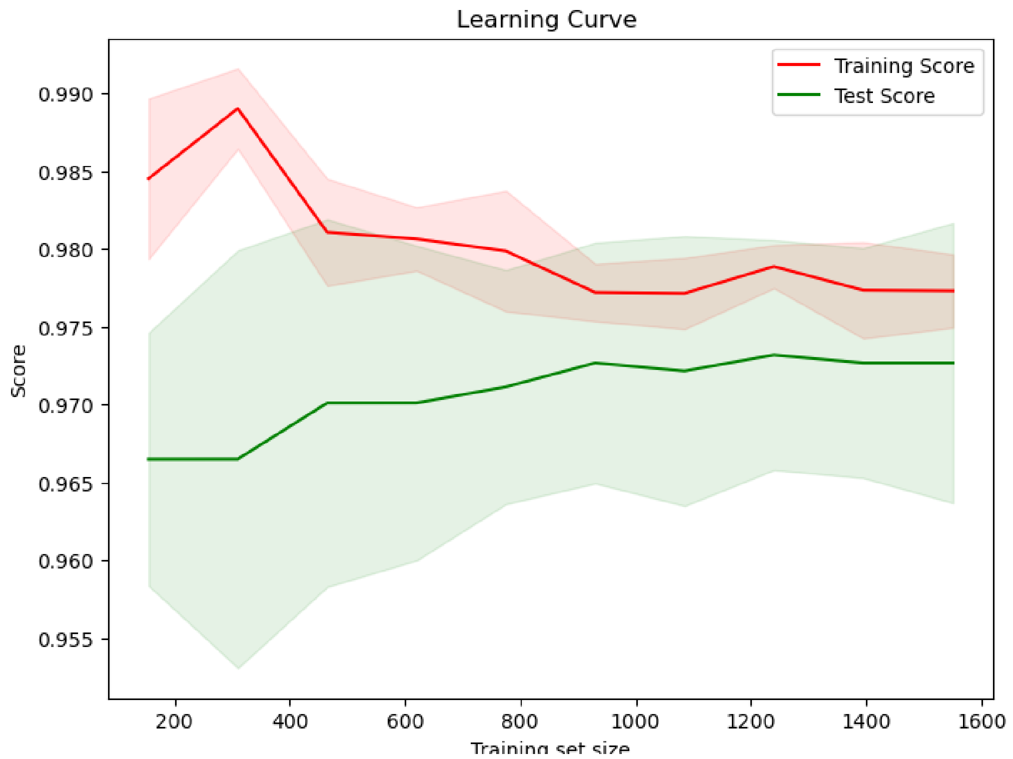
```
Best C: 100
Accuracy on training set: 0.975
Accuracy on test set: 0.969
```

This figure shows the confusion matrix of the SVM model. The confusion matrix shows the number of true positives that predicted by the model is 388 and the number of true negatives that predicted by the model is 417.



This figure is the learning curve of the SVM model.

This learning curve shows how the model training and test accuracy affected over the change of the training set size.

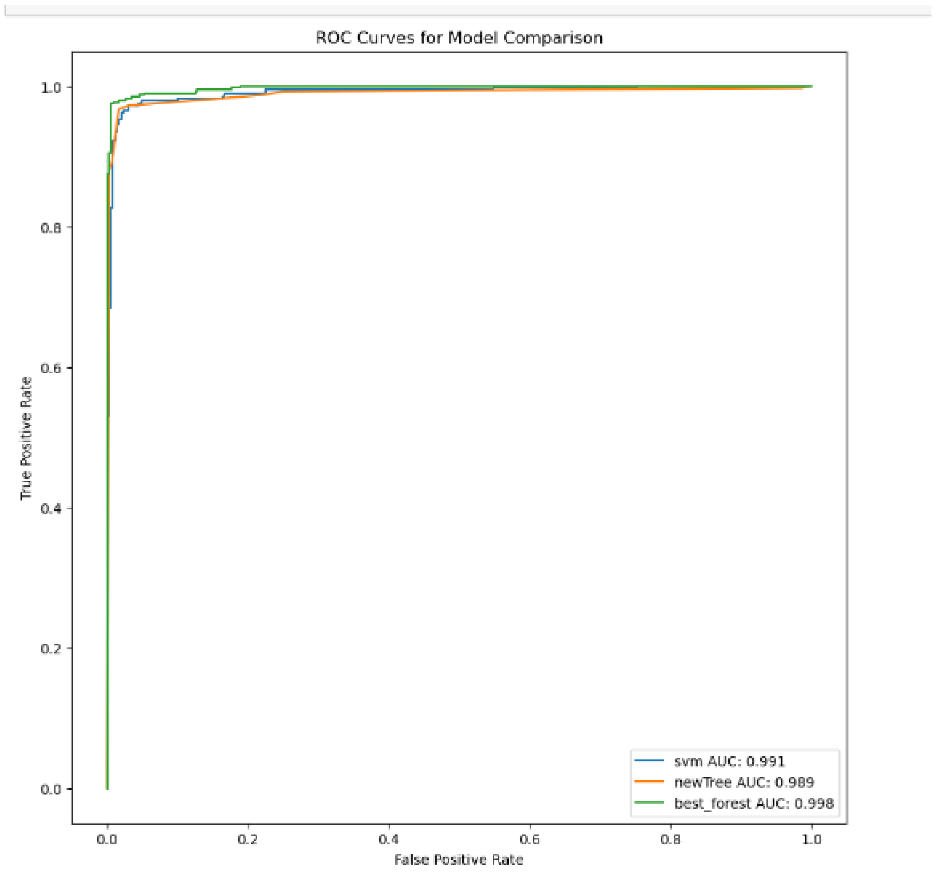


4. Model performance Comparison:

4.1. ROC curve:

Binary classification model performance is evaluated using a graphical representation called a ROC curve. The ability of a model to differentiate between positive and negative classes is shown visually. An ROC curve that touched the graph's upper-left corner would represent a perfect model. One way to quantify a model's discriminative power is to look at its area under the ROC curve (AUC), where larger values correspond to better results.

According to this figure, the random forest model has the best value of the AUC which means that it is the best model.



4.2. Classification report:

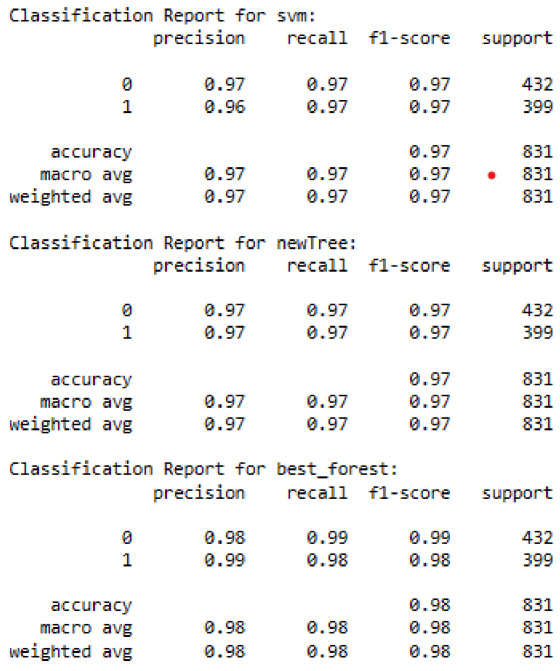
The classification report shows the f1 score, precision, and recall of the model.

Precision: Precision is the ratio of correct positive prediction to the overall positive predictions.

Recall: Recall is the ratio of correct positive prediction to the overall number of positive examples in the dataset.

F1 score: The F1-Score is the harmonic mean of precision and recall. It combines both precision and recall.

According to this figure, the best model is the random forest model.



```
Classification Report for svm:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       432
           1       0.96      0.97      0.97       399

    accuracy                           0.97       831
   macro avg       0.97      0.97      0.97       831
weighted avg       0.97      0.97      0.97       831

Classification Report for newTree:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       432
           1       0.97      0.97      0.97       399

    accuracy                           0.97       831
   macro avg       0.97      0.97      0.97       831
weighted avg       0.97      0.97      0.97       831

Classification Report for best_forest:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       432
           1       0.99      0.98      0.98       399

    accuracy                           0.98       831
   macro avg       0.98      0.98      0.98       831
weighted avg       0.98      0.98      0.98       831
```

Portuguese Banking Marketing Campaign Classification Dataset (Sara Amjad 212071)

1. Data Description:

An institution of banking in Portugal generated this dataset through direct marketing campaigns. Phone calls to consumers were the main means of promoting a bank term deposit subscription during the campaigns. The objective was to ascertain if a client would accept the term deposit ("yes") or not ("no"). There are 41,188 data samples in this dataset, with 20 features and one target variable per sample.

The distribution of the data across the two classes is shown in Figure 1. The data is clearly class skewed towards class 0, as seen in the figure below. Which could be considered as a data imbalance.

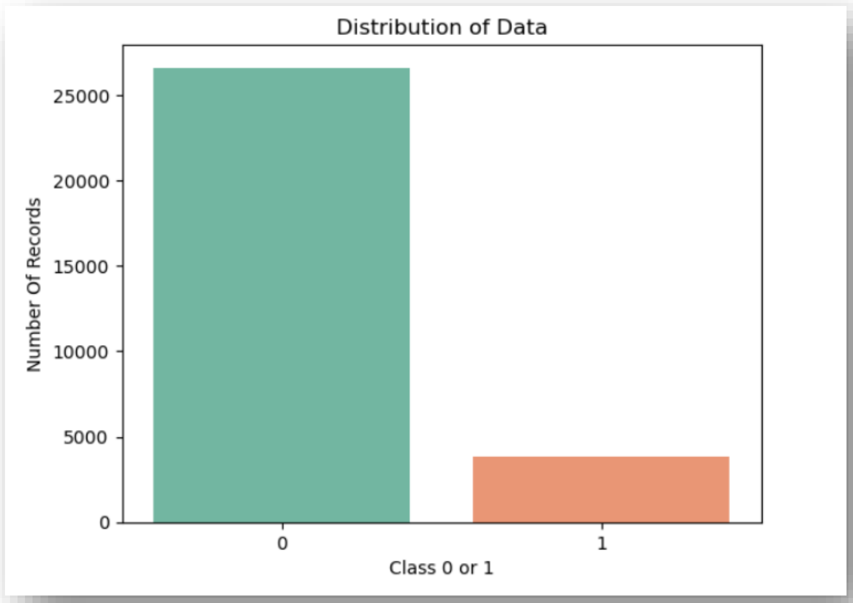


Figure 1

2. Data Pre-Processing:

- **Nulls: Initially, there were no null values; but, when substituting Nan for the "unknown" values, the null values were identified and displayed on the heatmap.**

The Nulls before replacing the 'Unknown' values with Nan are shown below in figure 2.

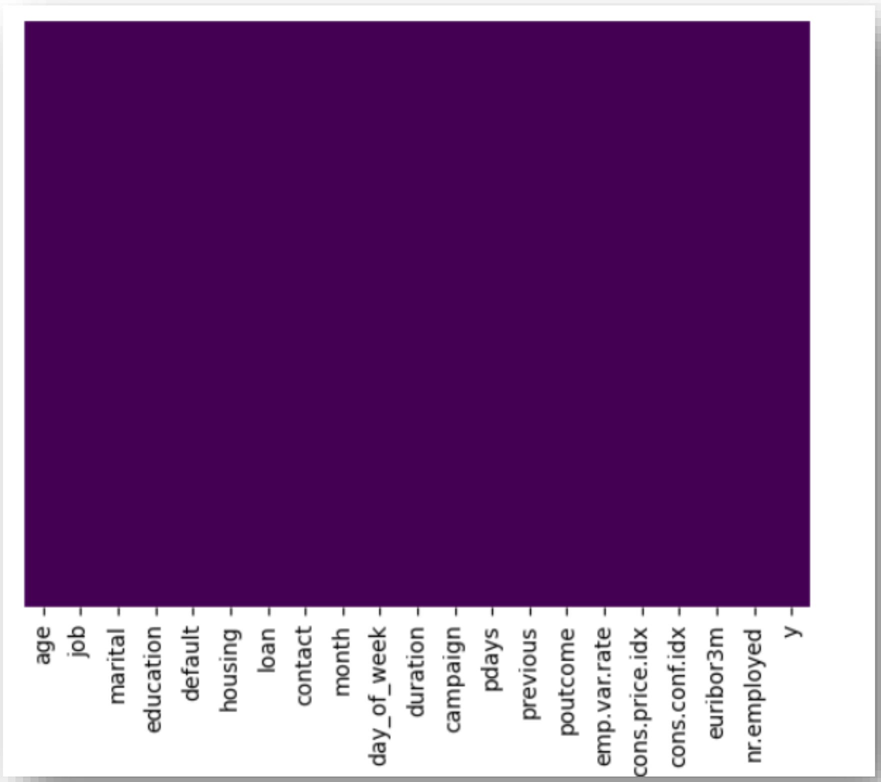


Figure 2

And The Nulls after replacing the ‘Unknown’ values with Nan are shown below in figure 3.

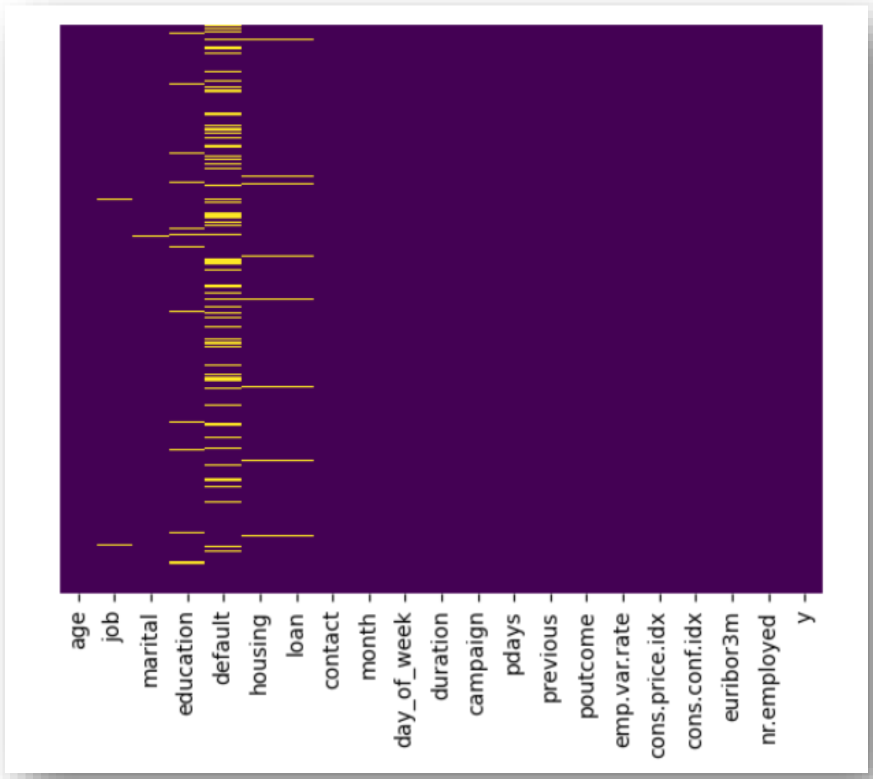


Figure 3

As shown above in figure 3 the ‘default’ feature is full of nulls which is why its going to be dropped.

- ## Duplicates:

  There are 10 duplicates which will be dropping afterwards keeping only the first one.

- ## Outliers:

  at the beginning using the IQR, the outliers were dropped in which the first quartile (Q1) is calculated at the 15th percentile and the third quartile (Q3) is calculated at 85th percentile. However, the dataset records decreased from 30478 to 26620 and it seems that the records of class 1 were considered as outliers due to the data imbalance where the records of class 1 are way less than the records of class 0. And in Figure 4 and 5 it shows the records of class 1 before and after dropping the

outliers showing the fact that class 1 records were considered as outliers and were dropped. So, I decided not to drop these outliers and after I changed the (Q1) to be calculated at the 3rd percentile and the (Q3) to be calculated at the 97th percentile no records were dropped.

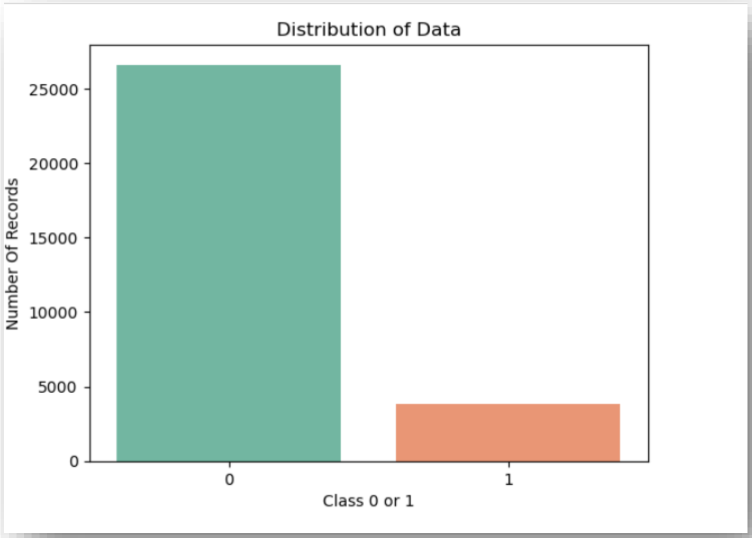


Figure 4 (with outliers)

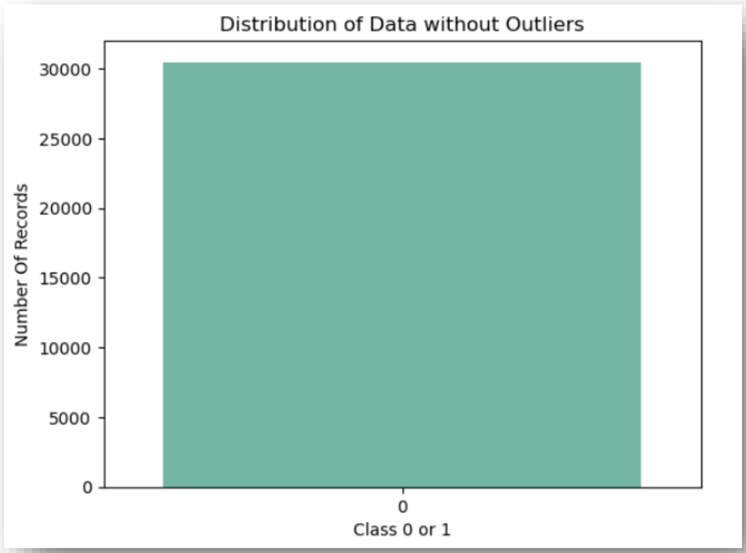
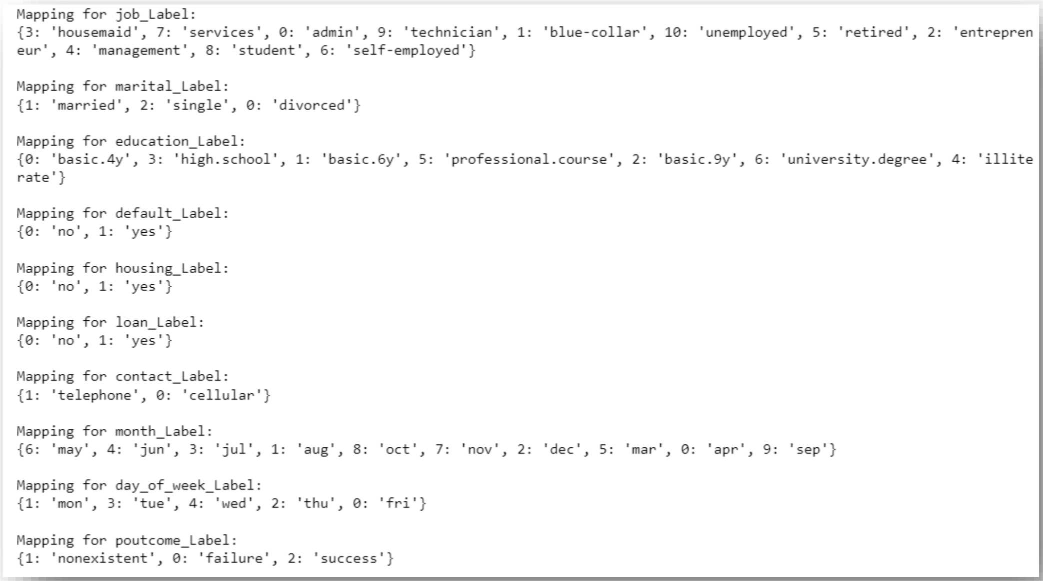


Figure 5 (without outliers)

- **Represent Categorical variables as numerical labels:**

Through performing a label encoding for categorical columns in the Data frame to facilitate and make it suitable for the use of decision trees. As shown in Figure 6.



```
Mapping for job_Label:
{3: 'housemaid', 7: 'services', 0: 'admin', 9: 'technician', 1: 'blue-collar', 10: 'unemployed', 5: 'retired', 2: 'entrepren
eur', 4: 'management', 8: 'student', 6: 'self-employed'}

Mapping for marital_Label:
{1: 'married', 2: 'single', 0: 'divorced'}

Mapping for education_Label:
{0: 'basic.4y', 3: 'high.school', 1: 'basic.6y', 5: 'professional.course', 2: 'basic.9y', 6: 'university.degree', 4: 'illite
rate'}

Mapping for default_Label:
{0: 'no', 1: 'yes'}

Mapping for housing_Label:
{0: 'no', 1: 'yes'}

Mapping for loan_Label:
{0: 'no', 1: 'yes'}

Mapping for contact_Label:
{1: 'telephone', 0: 'cellular'}

Mapping for month_Label:
{6: 'may', 4: 'jun', 3: 'jul', 1: 'aug', 8: 'oct', 7: 'nov', 2: 'dec', 5: 'mar', 0: 'apr', 9: 'sep'}

Mapping for day_of_week_Label:
{1: 'mon', 3: 'tue', 4: 'wed', 2: 'thu', 0: 'fri'}

Mapping for poutcome_Label:
{1: 'nonexistent', 0: 'failure', 2: 'success'}
```

Figure 6

Then drop the categorical columns that were substituted with the labelled columns since no need to have the categorical ones anymore.

3. Feature Importance of the Models

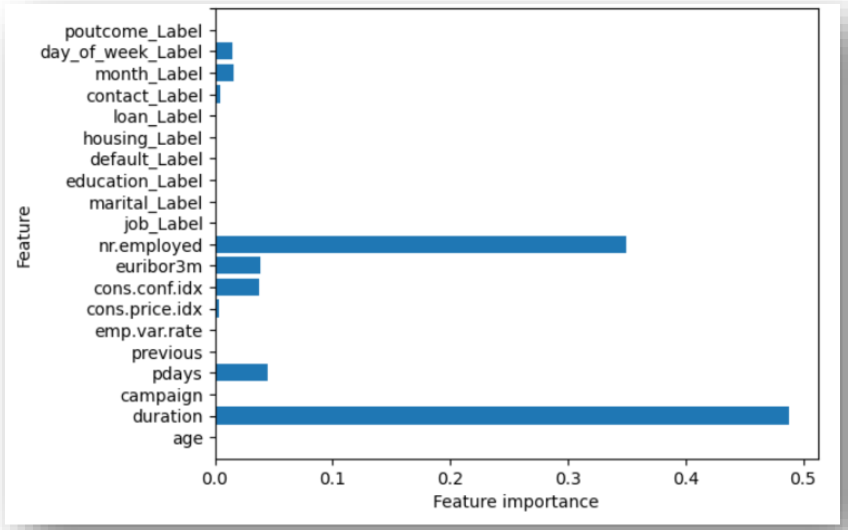


Figure 7

As shown above in Figure 7, This is the Feature importance plot of the Decision tree.

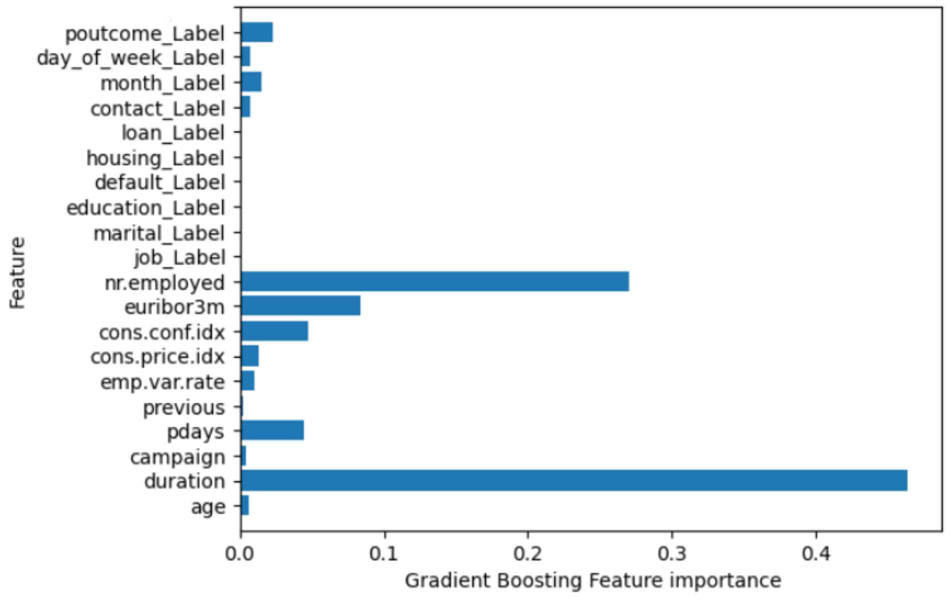


Figure 8

As shown above in Figure 8, This is the Feature importance plot of the Gradient Boosting Model.

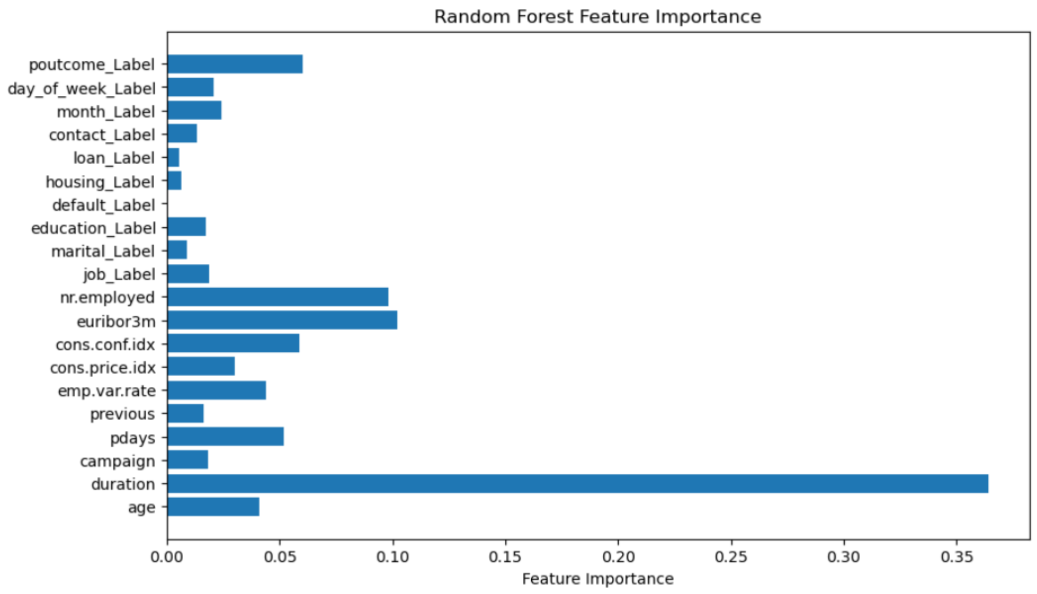


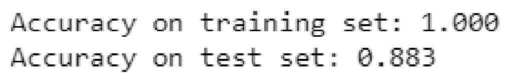
Figure 9

As shown above in Figure 9, This is the Feature importance plot of the Random Forest Model.

This Random Forest Feature Importance shows that the random forest generalizes well and provide a more comprehensive view of feature importance than the decision tree and gradient boosting.

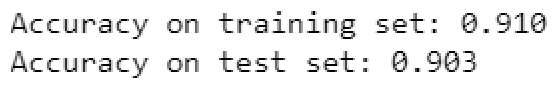
4. Decision Tree Model:

1st decision tree (no specified maximum depth and trained on all the features) Training and Testing accuracy.



```
Accuracy on training set: 1.000
Accuracy on test set: 0.883
```

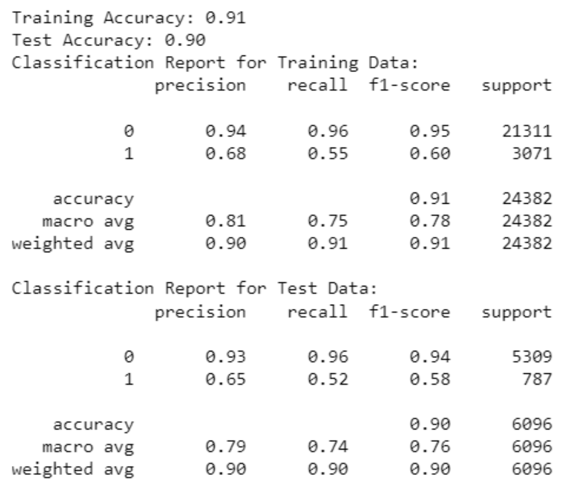
2nd decision tree using Pre-pruning (maximum depth to 5 and minimum samples split to 6) Training and Testing accuracy.



```
Accuracy on training set: 0.910
Accuracy on test set: 0.903
```

The first decision tree shows an overfitting since it performs very well on the training data but generalizes poorly to new, unseen data. So, to solve this and get rid of the overfitting, in the second decision tree a pre-pruning was applied resulting in no overfitting anymore and an increase in the testing accuracy.

The classification report of the Decision tree model (with Pre-Pruning)



```
Training Accuracy: 0.91
Test Accuracy: 0.90
Classification Report for Training Data:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     21311
           1       0.68      0.55      0.60      3071

    accuracy                           0.91     24382
   macro avg       0.81      0.75      0.78     24382
weighted avg       0.90      0.91      0.91     24382

Classification Report for Test Data:
              precision    recall  f1-score   support

           0       0.93      0.96      0.94      5309
           1       0.65      0.52      0.58       787

    accuracy                           0.90      6096
   macro avg       0.79      0.74      0.76      6096
weighted avg       0.90      0.90      0.90      6096
```

As shown above, the f1-score of class 1 is not that high like class 0 due to the data imbalance of the data in which the records of class 1 are a very small number compared to the records of class 0. A solution of this data imbalance problem could be Resampling whether oversampling or under-sampling using techniques like SMOTE. Or by Assigning different misclassification costs to different classes or by adjusting the classification threshold.

The Learning Curve of the Decision tree model (with Pre-Pruning)

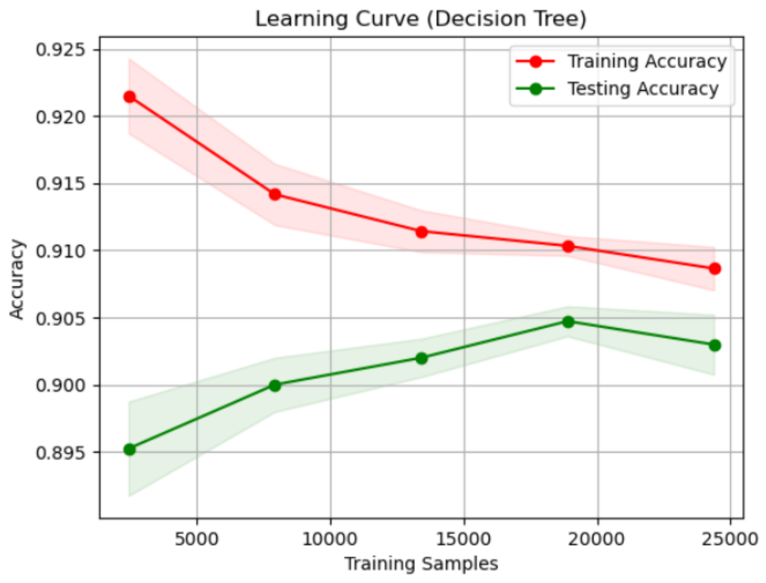


Figure 10

Plotting the learning curve of the Decision tree demonstrates how the training and testing accuracy of the model varies with the amount of training data. The training accuracy shows a slow, but not abrupt, fall with increasing numbers of training samples. With additional samples, the testing accuracy also shows an increasing tendency at the same time. At about 24,000 samples, these two accuracy curves converge and get closest to one another. This convergence implies that the model finds a balance between fitting the training data and obtaining strong test performance, stabilizing its performance, and reaching a point where it generalizes well to new data.

Confusion Matrix of the Decision tree with Pre-Pruning

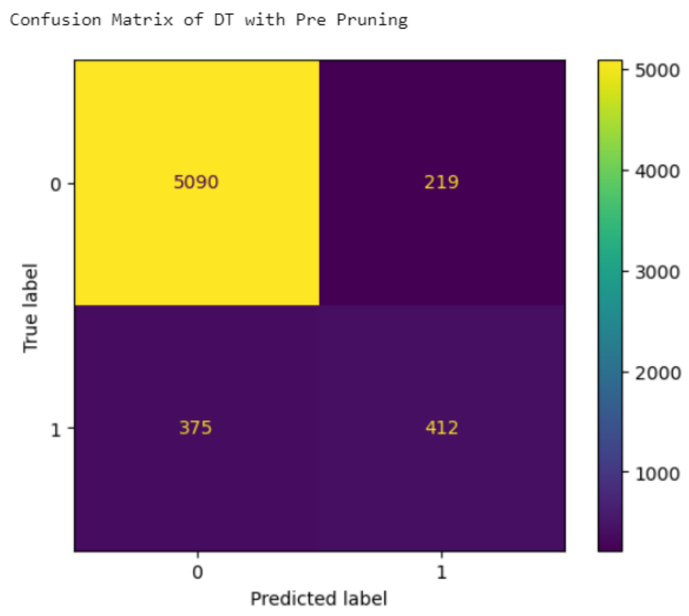


Figure 11

This Confusion Matrix here shows that the number of true positives is greater than the false positive however the difference between them is not that much which could be due to the data imbalance.

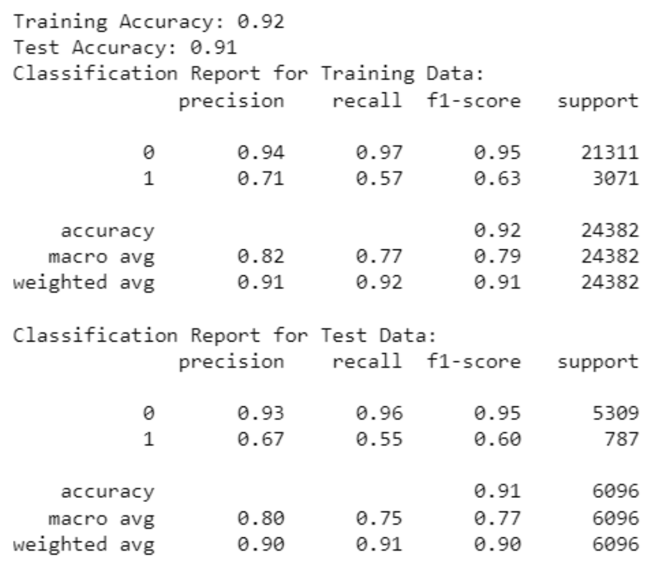
5. Gradient Boosting Model:

The Training and Testing accuracy of the gradient boosting model.

```
Accuracy on training set: 0.916
Accuracy on test set: 0.907
```

Through using the gradient boosting model, the accuracy of both the training and testing increased compared to when the decision tree was applied.

The classification report of the Gradient Boosting model



```
Training Accuracy: 0.92
Test Accuracy: 0.91
Classification Report for Training Data:
              precision    recall  f1-score   support

           0       0.94      0.97      0.95     21311
           1       0.71      0.57      0.63      3071

    accuracy                           0.92     24382
   macro avg       0.82      0.77      0.79     24382
weighted avg       0.91      0.92      0.91     24382

Classification Report for Test Data:
              precision    recall  f1-score   support

           0       0.93      0.96      0.95      5309
           1       0.67      0.55      0.60       787

    accuracy                           0.91      6096
   macro avg       0.80      0.75      0.77      6096
weighted avg       0.90      0.91      0.90      6096
```

The Learning Curve of the Gradient Boosting Classifier model

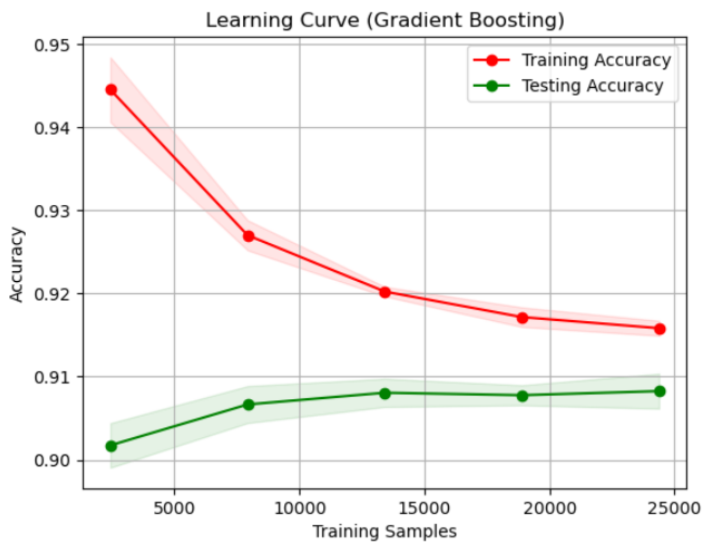


Figure 12

The gradient boosting model's learning curve illustrates how the quantity of training data affects the model's accuracy in testing and training. Increasing the amount of training samples causes the training accuracy to decrease gradually then become steady. The testing accuracy also has a trend towards increasing with more samples then becomes steady as well. These two accuracy curves converge and get closest to each other at roughly 24,000 samples. This convergence suggests that the model stabilizes its performance and reaches a point where it performs well when generalized to new data, reaching a balance between fitting the training data and achieving excellent test performance.

Confusion Matrix of the Gradient Boosting Classifier model

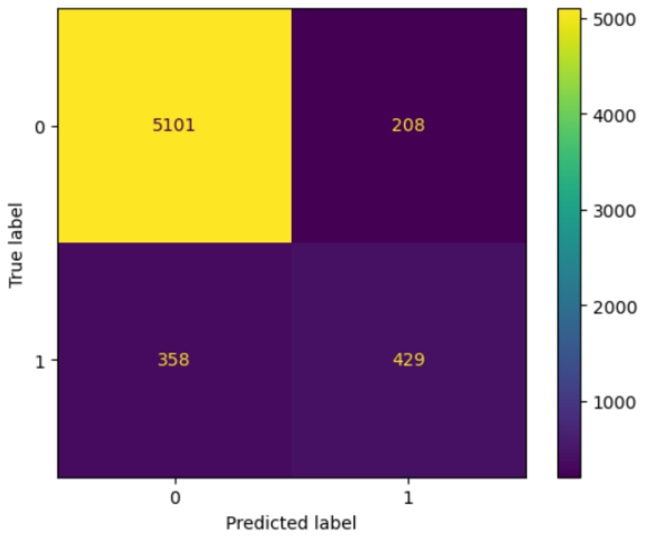
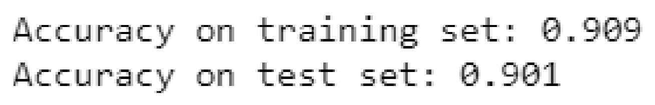


Figure 13

Clearly in the above figure, the number of the correctly predicted in the gradient boosting is higher than the decision tree.

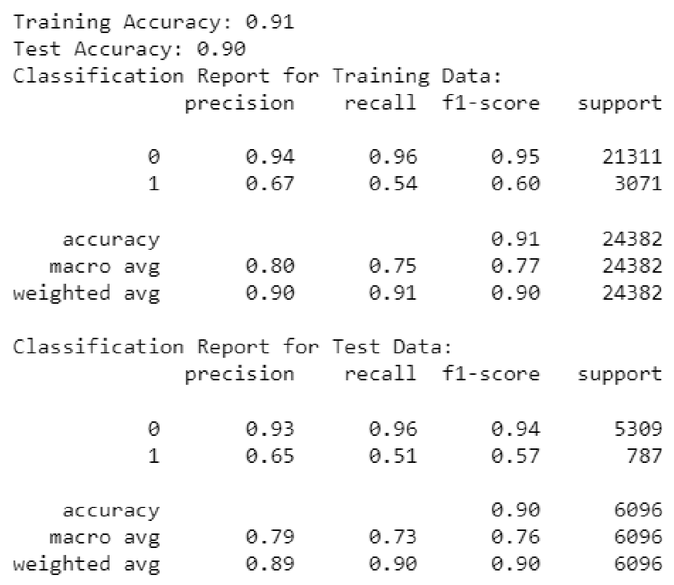
6. KNN:

The Training and Testing accuracy of the K-Nearest Neighbours model.



```
Accuracy on training set: 0.909
Accuracy on test set: 0.901
```

The classification report of the K-Nearest Neighbours model



```
Training Accuracy: 0.91
Test Accuracy: 0.90
Classification Report for Training Data:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     21311
           1       0.67      0.54      0.60      3071

    accuracy                           0.91     24382
   macro avg       0.80      0.75      0.77     24382
weighted avg       0.90      0.91      0.90     24382

Classification Report for Test Data:
              precision    recall  f1-score   support

           0       0.93      0.96      0.94      5309
           1       0.65      0.51      0.57       787

    accuracy                           0.90      6096
   macro avg       0.79      0.73      0.76      6096
weighted avg       0.89      0.90      0.90      6096
```

Confusion Matrix of the K-Nearest Neighbours model

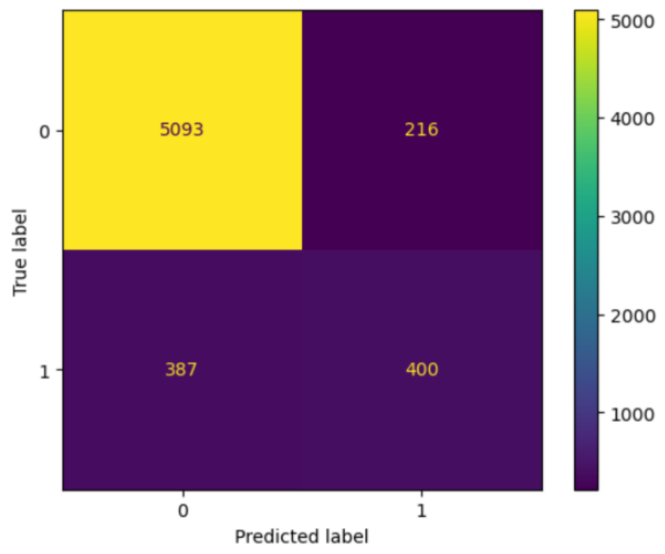


Figure 14

Training and Testing Accuracy vs. Value of K for KNN

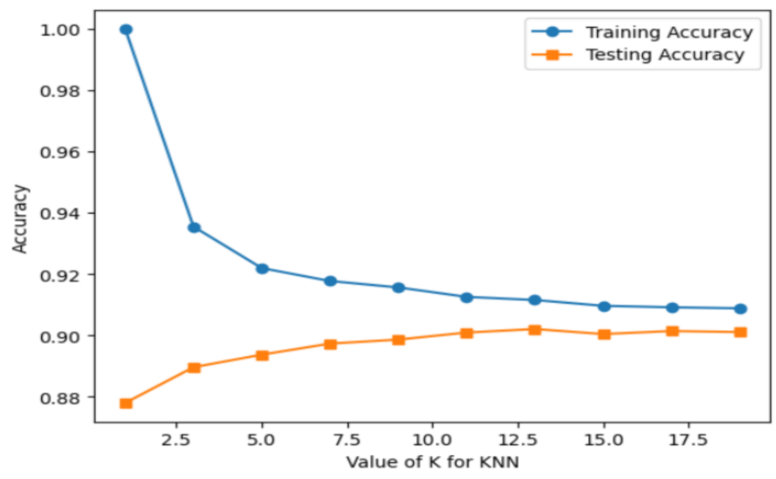


Figure 15

As you can see above in Figure 15, The plot illustrates the correlation between a K-Nearest Neighbours (KNN) classification model's training, testing accuracy and its value, K, which represents the number of nearest neighbours considered. It also helps in illustrating how the model's performance is affected by the selection of K.

The Learning Curve of the KNN model

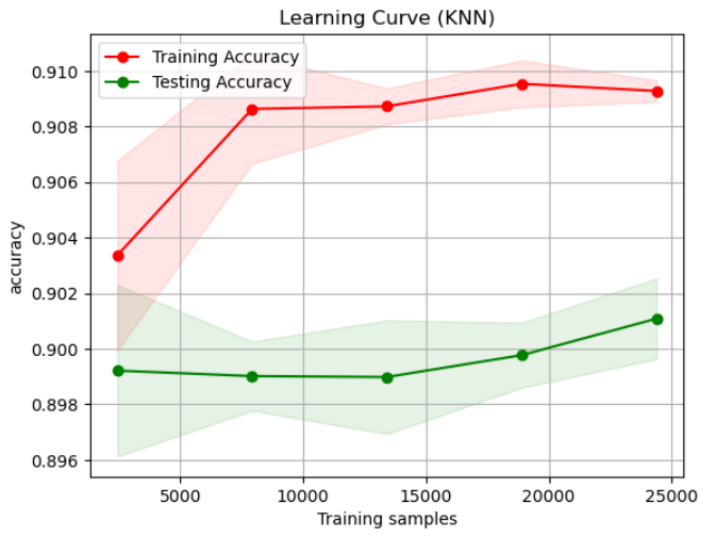
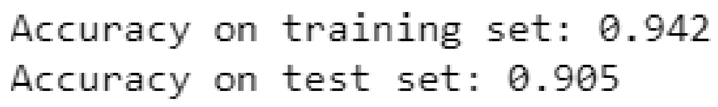


Figure 16

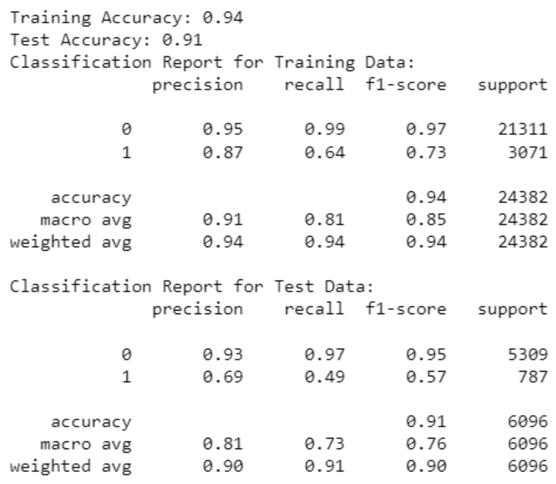
7. Random Forest Model:

The Training and Testing accuracy of the Random Forest model.



```
Accuracy on training set: 0.942
Accuracy on test set: 0.905
```

The classification report of the Random Forest model



```
Training Accuracy: 0.94
Test Accuracy: 0.91
Classification Report for Training Data:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     21311
           1       0.87      0.64      0.73      3071

    accuracy                           0.94     24382
   macro avg       0.91      0.81      0.85     24382
weighted avg       0.94      0.94      0.94     24382

Classification Report for Test Data:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      5309
           1       0.69      0.49      0.57       787

    accuracy                           0.91      6096
   macro avg       0.81      0.73      0.76      6096
weighted avg       0.90      0.91      0.90      6096
```

Confusion Matrix of the Random Forest model

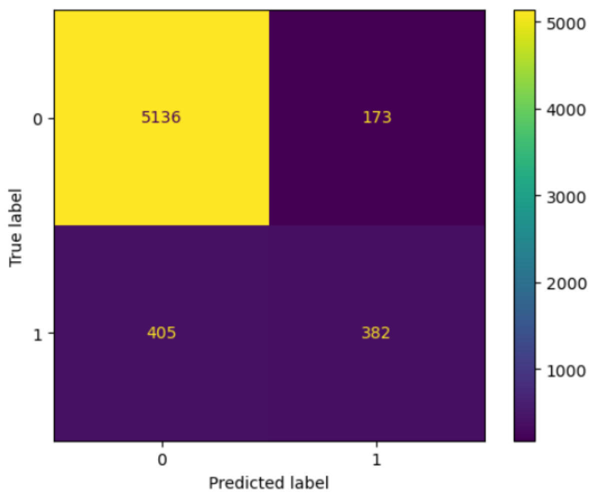


Figure 17

The Learning Curve of the Random Forest model

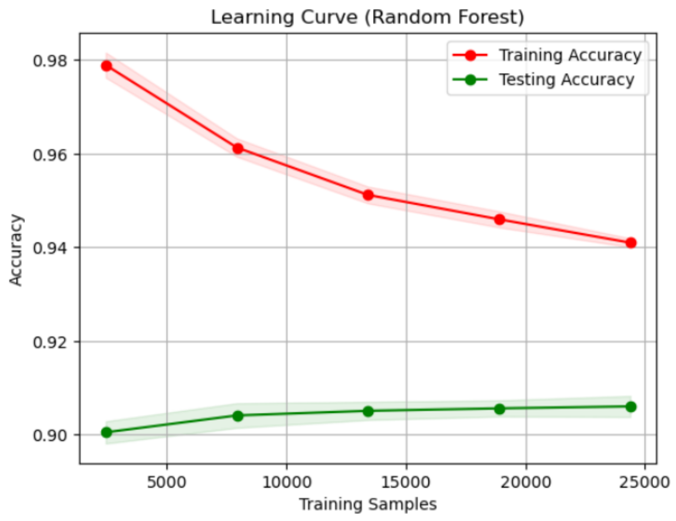


Figure 18

Plotting the learning curve of a Random Forest classifier demonstrates how the training and testing accuracy of the model varies with the amount of training data. The training accuracy shows a slow, but not abrupt, fall with increasing numbers of training samples. With additional samples, the testing accuracy also shows an increasing tendency at the same time then becomes steady. At about 24,000 samples, these two accuracy curves converge and get closest to one another. This convergence implies that the model finds a balance between fitting the training data and obtaining strong test performance, stabilizing its performance, and reaching a point where it generalizes well to new data.

8. ROC Curves for Model Comparison

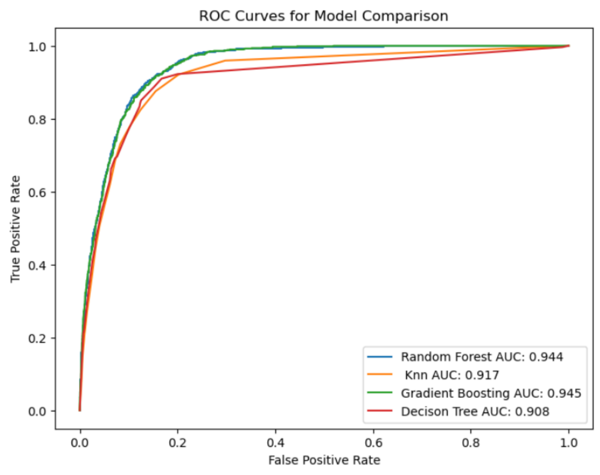


Figure 19

According to the plot shown above, the Gradient boosting has an overall better performance in distinguishing between these 2 classes. And as you can see in figure 19 as well, the Random Forest and the Gradient Boosting AUC curves are very similar since the AUC scores are 0.944 and 0.945 which is why the curves are a bit overlapping. So, to deal with this I visualized both ROC curves separately. In the ROC plot for all four models shown above, a Strong predictive performance is indicated by the curves' high ascent from the origin and approach to the top-left corner of the plot. The Area Under the ROC Curve (AUC) scores are 0.908,0.917,0.944 and 0.945 showing that the models are performing quite well.

Random Forest ROC Curve

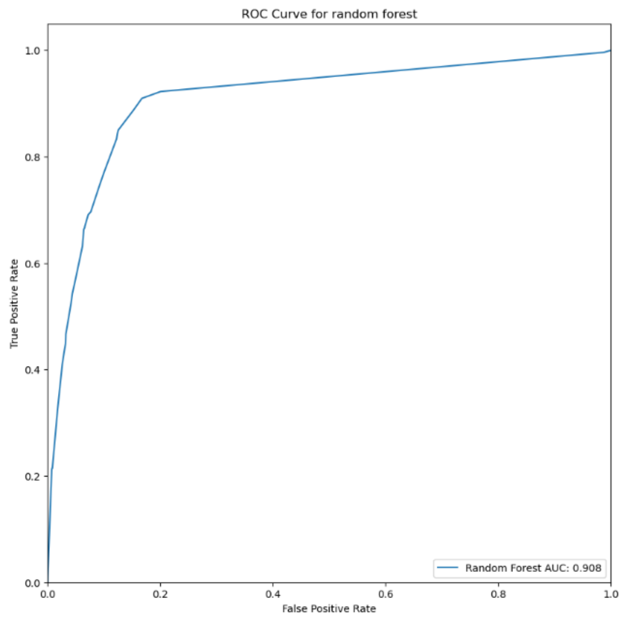


Figure 20

Gradient Boosting ROC Curve

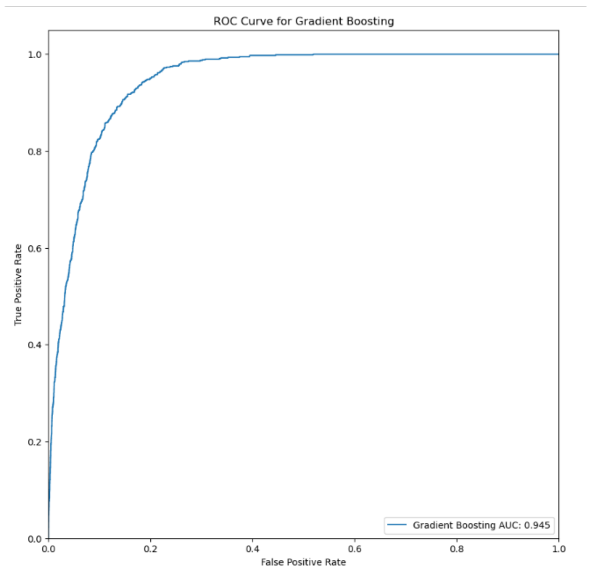
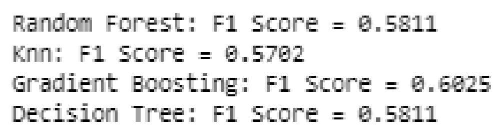


Figure 21

9. F1 Score of all Models



```
Random Forest: F1 Score = 0.5811
Knn: F1 Score = 0.5702
Gradient Boosting: F1 Score = 0.6025
Decision Tree: F1 Score = 0.5811
```

The existence of data imbalance in the dataset may explain why the F1 score for the models isn't as great as anticipated. And as I stated earlier the cause of the data imbalance is because one class greatly outnumbers the other. Because it considers both recall and precision, the F1 score is sensitive to the disparity in performance between the classes. When data is unbalanced, the dominant class often has an impact on the F1 score. This could result in a lower F1 score for the minority class, which is frequently the class of interest.

According to the F1 Scores the Gradient boosting is considered better.

It is vital to remember that the models might still function successfully in real-world scenarios even with a lower F1 score.

Omar 207140

Classification report

Cirrhosis Patient Survival Prediction

The Data description:

Will Utilize 17 clinical features for predicting survival state of patients with liver cirrhosis. The survival states include 0 = D (death), 1 = C (censored), 2 = CL (censored due to liver transplant),This dataset have 418 instances and 18 features and target label being status ,this dataset subject area is Health and medicine

Figure 1:This count plot shows the count of each class in status where we can see that CL is much smaller than the other classes

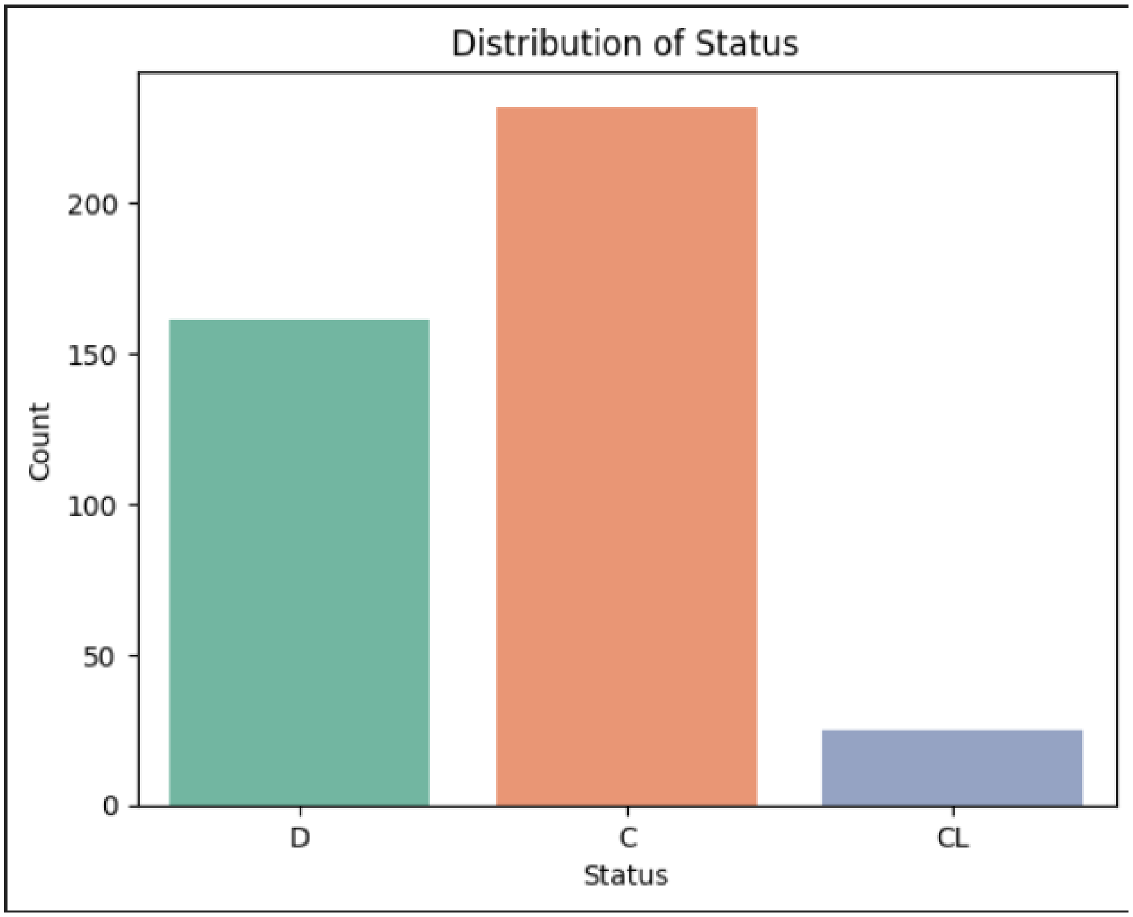


Figure 1st

Data cleaning :

Cheeking for nulls :

Using heat map to visualize(figure 2) the nulls in the dataset , we can see that most of the nulls comes from certain instances which all next to each other this is big problem because filling techniques won't work like unless they going to fill with very wrong , if not repeated over and over , this null huge null values make 9 columns out of 18 (not counting status , and id will be removed ') ,that is 50% of the data being null trying to fill it will ruin the accuracy , that is dropping this nulls is the best option in my opinion which was about 34% of the dataset dropping down to 276

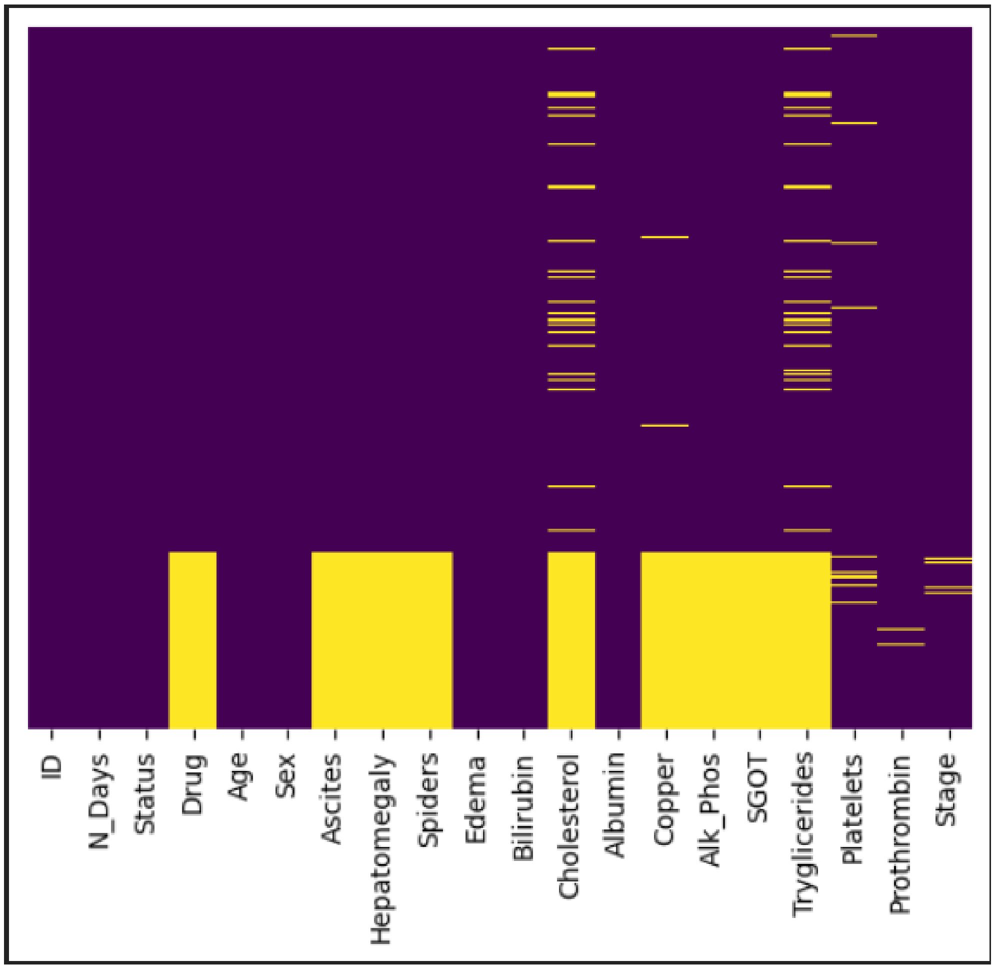


Figure 2

Duplicates :

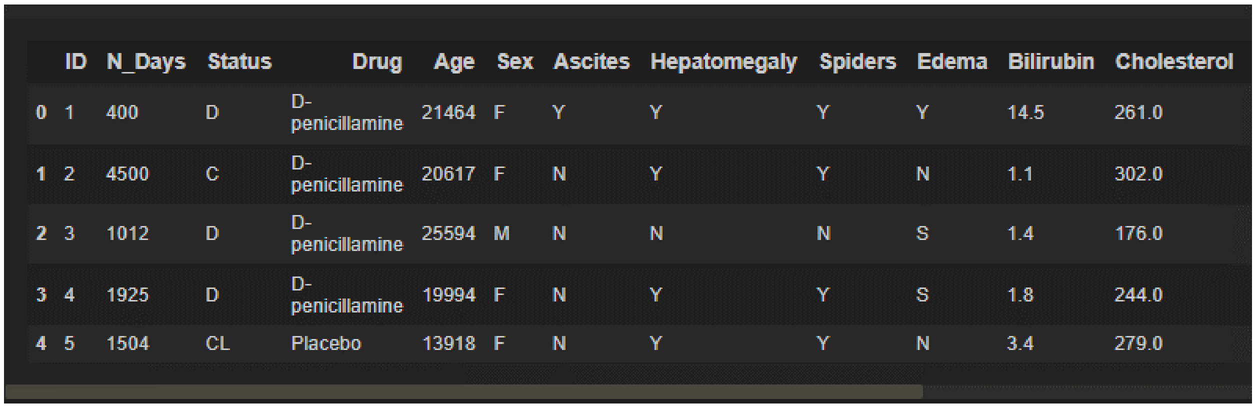
there where no duplicates

Outliers :

There where only 5 outliers

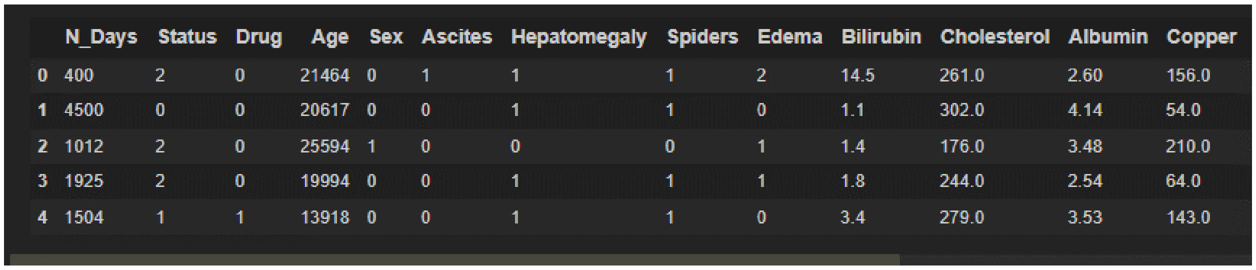
Labeling and dropping :

- Status , sex , ascites hepatomegaly , spiders ,edema are string but with repeated values exp sex (F,M) , spiders(Y,N) , so will be changed to numbers .
- Column id will be removed because all unique values



	ID	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Cholesterol
0	1	400	D	D-penicillamine	21464	F	Y	Y	Y	Y	14.5	261.0
1	2	4500	C	D-penicillamine	20617	F	N	Y	Y	N	1.1	302.0
2	3	1012	D	D-penicillamine	25594	M	N	N	N	S	1.4	176.0
3	4	1925	D	D-penicillamine	19994	F	N	Y	Y	S	1.8	244.0
4	5	1504	CL	Placebo	13918	F	N	Y	Y	N	3.4	279.0

To

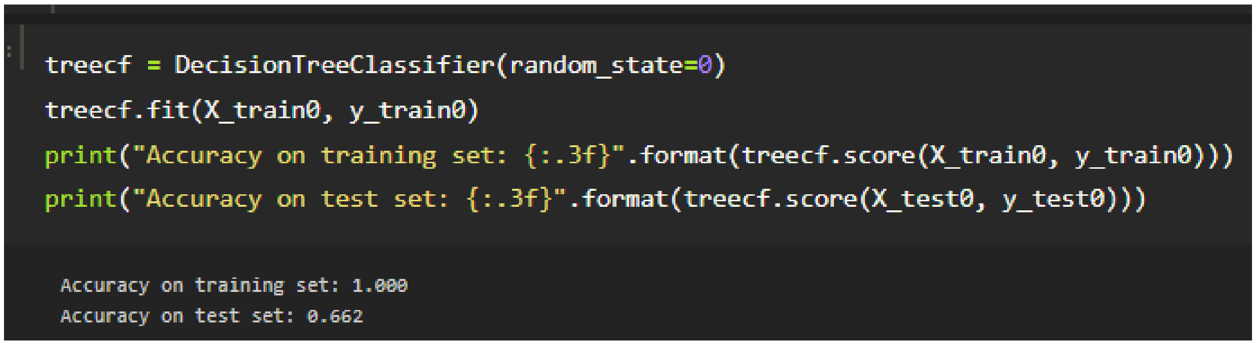


	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Cholesterol	Albumin	Copper
0	400	2	0	21464	0	1	1	1	2	14.5	261.0	2.60	156.0
1	4500	0	0	20617	0	0	1	1	0	1.1	302.0	4.14	54.0
2	1012	2	0	25594	1	0	0	0	1	1.4	176.0	3.48	210.0
3	1925	2	0	19994	0	0	1	1	1	1.8	244.0	2.54	64.0
4	1504	1	1	13918	0	0	1	1	0	3.4	279.0	3.53	143.0

Models

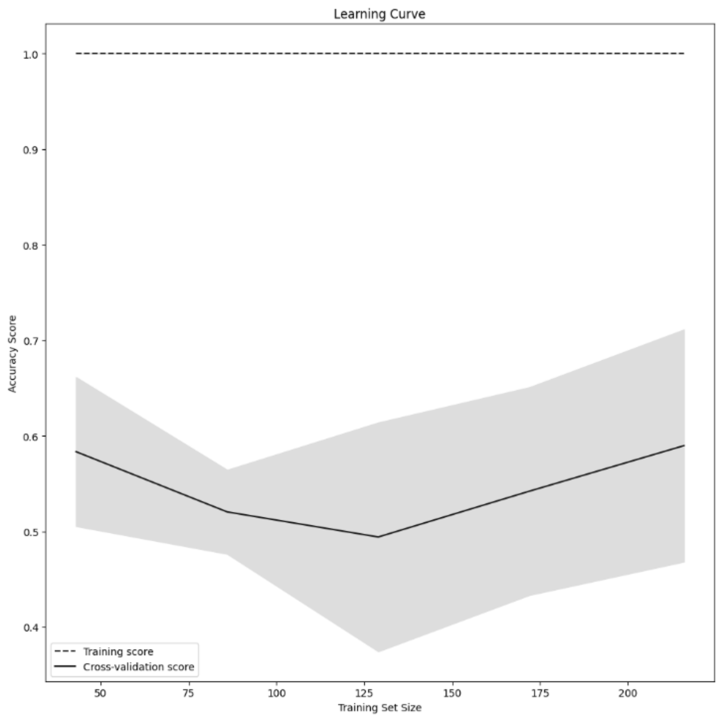
We will be training multiple of models , some of them we will try to improve the accuracy , this will be combined with visualization to see the improvement and the problems that exits while training

1. Decision Tree
   Training the model using all the features



```
treecf = DecisionTreeClassifier(random_state=0)
treecf.fit(X_train0, y_train0)
print("Accuracy on training set: {:.3f}".format(treecf.score(X_train0, y_train0)))
print("Accuracy on test set: {:.3f}".format(treecf.score(X_test0, y_test0)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.662
```



From the learning curve(figure3) we can see that there overfitting ,we will trying to fix that with pre Pruning

Figure 3

Seeing the important features of the decision tree (figure4)

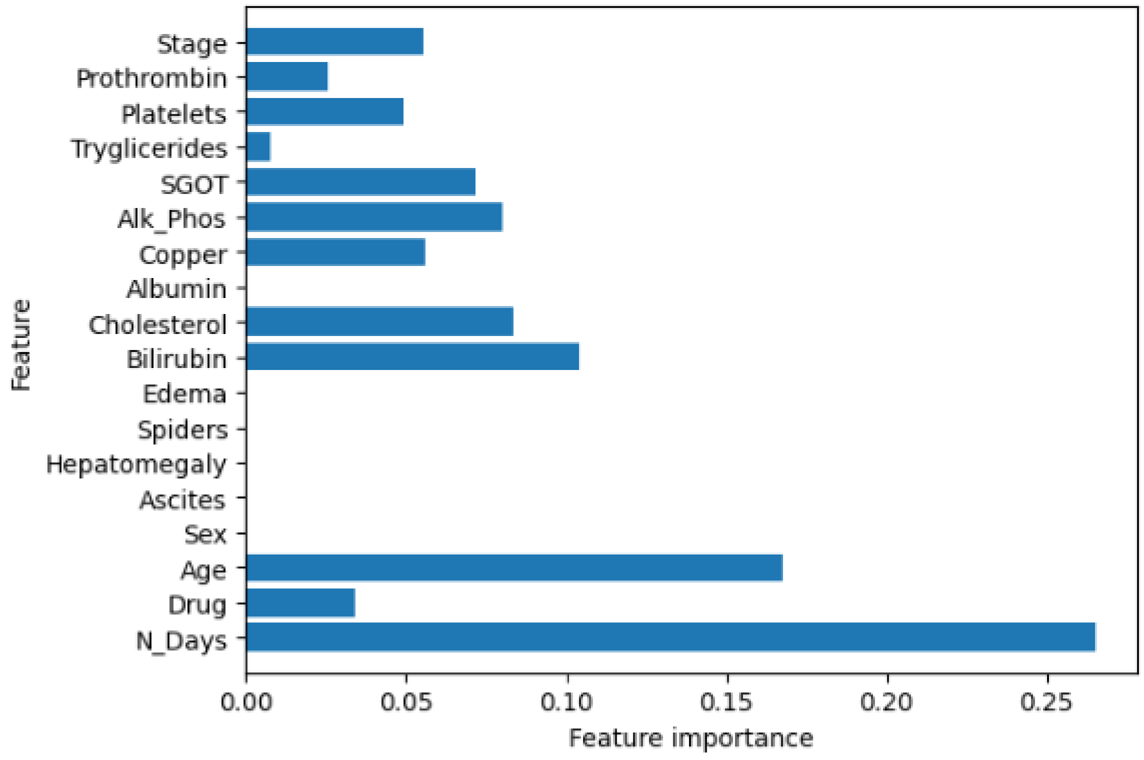
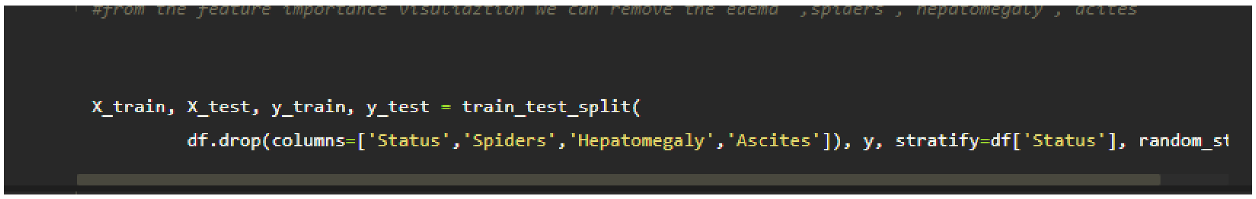


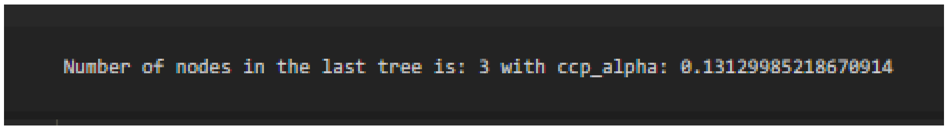
figure 4

We can see that there are features that is not important so we remove it from the training



```
#from the feature importance visulazition we can remove the edema  ,spiders , hepatomegaly , acites



X_train, X_test, y_train, y_test = train_test_split(
        df.drop(columns=['Status','Spiders','Hepatomegaly','Ascites']), y, stratify=df['Status'], random_st
```

Will be alpha pruning



```
Number of nodes in the last tree is: 3 with ccp_alpha: 0.13129985218670914
```

Trying different alphas to see that best form the graph

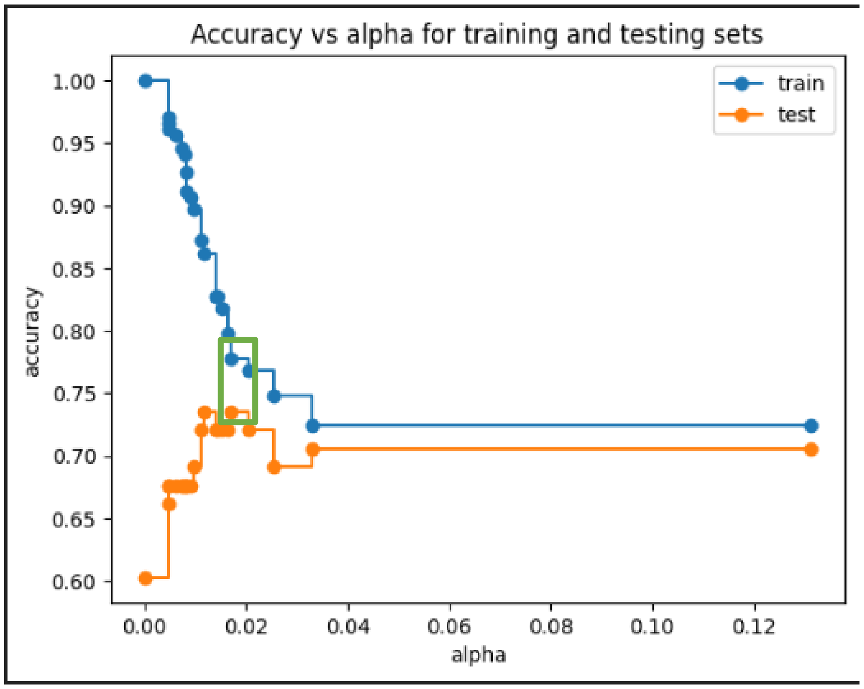
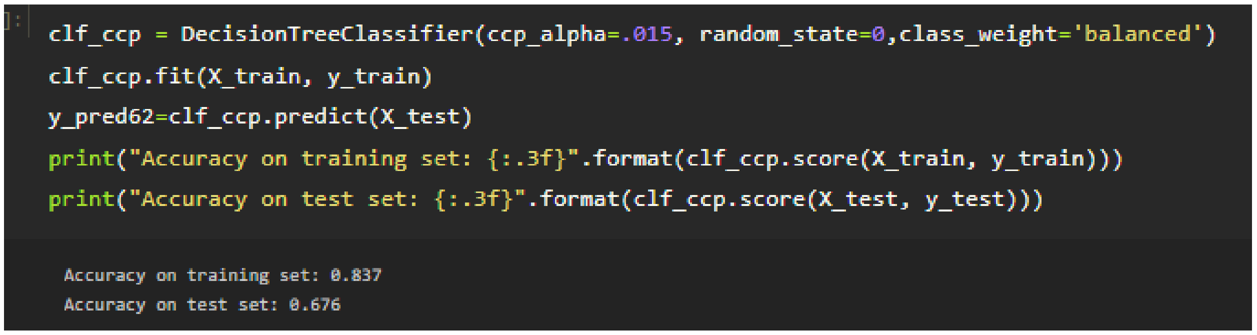


Figure 5

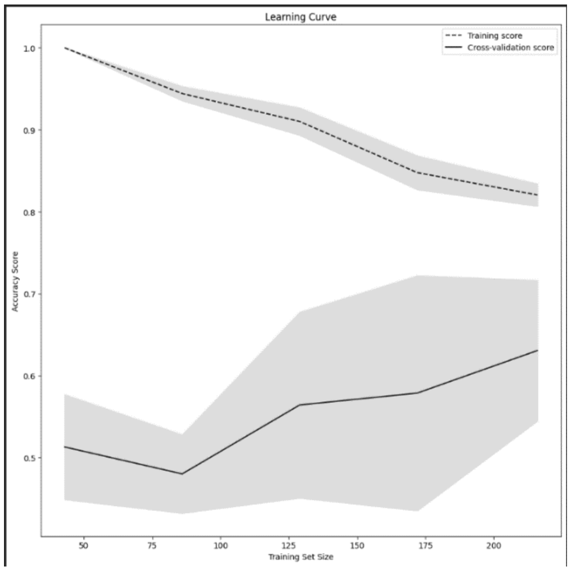
The green part is the figure 5 is the region we the best alpha is whish is near the one we found

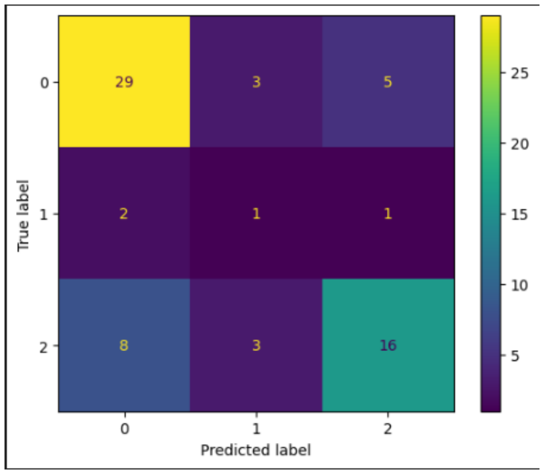
After training again with less features and alpha



```
clf_ccp = DecisionTreeClassifier(ccp_alpha=.015, random_state=0,class_weight='balanced')
clf_ccp.fit(X_train, y_train)
y_pred62=clf_ccp.predict(X_test)
print("Accuracy on training set: {:.3f}".format(clf_ccp.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(clf_ccp.score(X_test, y_test)))

 Accuracy on training set: 0.837
 Accuracy on test set: 0.676
```

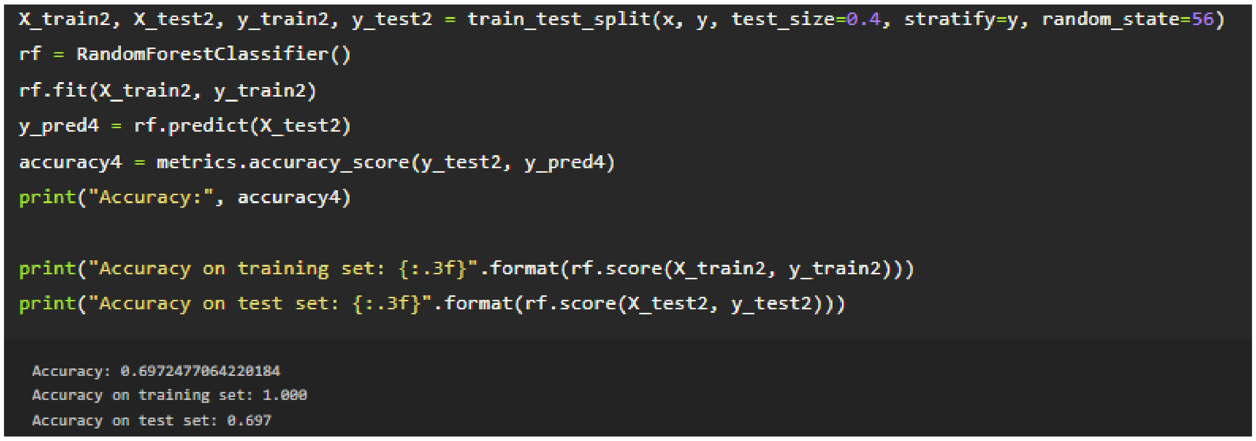
Visualization of decision tree learning curve and confusion matrix :





From the learning curve we can see that the overfitting has been fixed and the accuracy of testing increased

2. Random forest classification
   Training the model



```
X_train2, X_test2, y_train2, y_test2 = train_test_split(x, y, test_size=0.4, stratify=y, random_state=56)
rf = RandomForestClassifier()
rf.fit(X_train2, y_train2)
y_pred4 = rf.predict(X_test2)
accuracy4 = metrics.accuracy_score(y_test2, y_pred4)
print("Accuracy:", accuracy4)

print("Accuracy on training set: {:.3f}".format(rf.score(X_train2, y_train2)))
print("Accuracy on test set: {:.3f}".format(rf.score(X_test2, y_test2)))
```

```
Accuracy: 0.6972477064220184
Accuracy on training set: 1.000
Accuracy on test set: 0.697
```

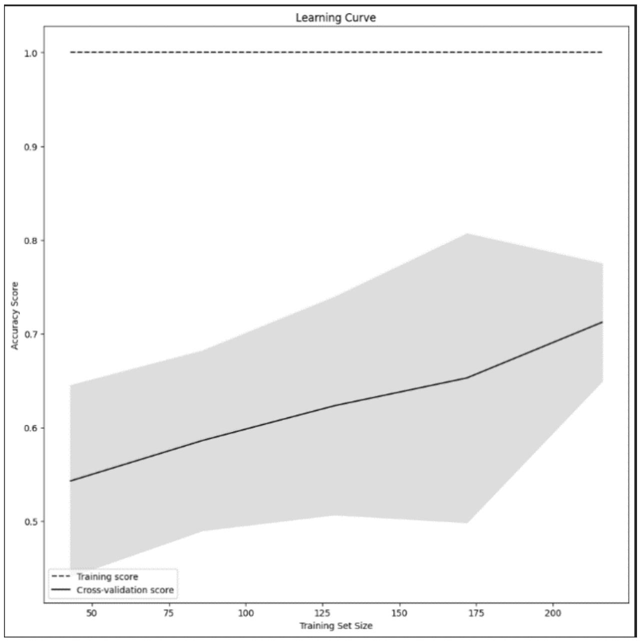
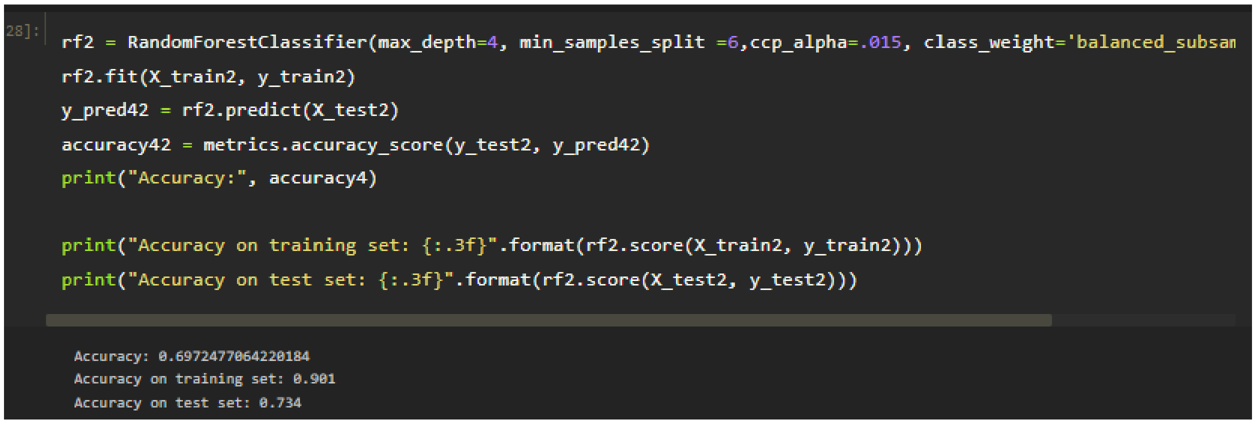


figure 6

From the learning curve(figure6) we can see that there overfitting, we will trying to fix that with pre
Pruning

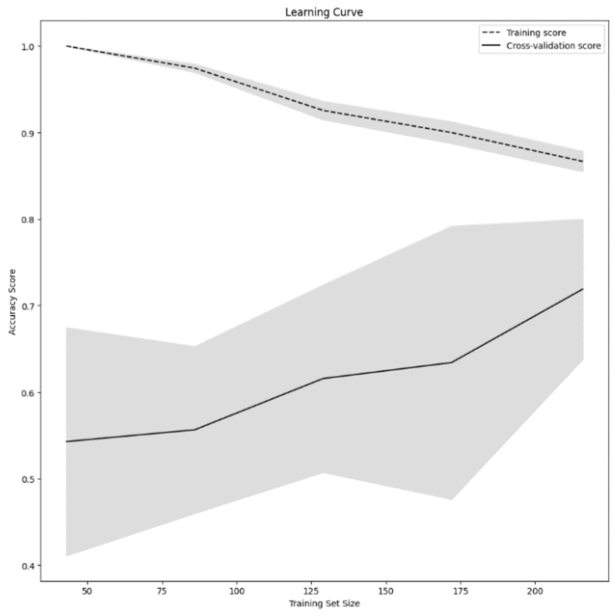
by making maximum depth = 4 and min sample split = 6 , using the same alpha as before , class weight to increase the wight of the second class because of it is small amount



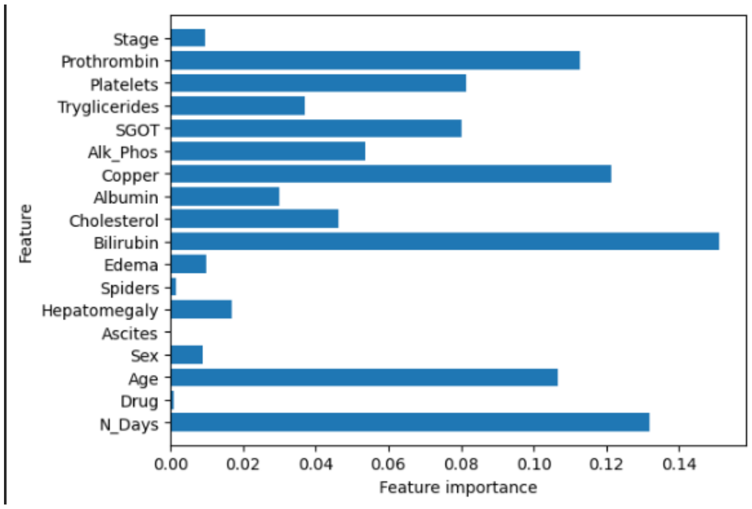
```
rf2 = RandomForestClassifier(max_depth=4, min_samples_split =6,ccp_alpha=.015, class_weight='balanced_subsam
rf2.fit(X_train2, y_train2)
y_pred42 = rf2.predict(X_test2)
accuracy42 = metrics.accuracy_score(y_test2, y_pred42)
print("Accuracy:", accuracy4)

print("Accuracy on training set: {:.3f}".format(rf2.score(X_train2, y_train2)))
print("Accuracy on test set: {:.3f}".format(rf2.score(X_test2, y_test2)))
```

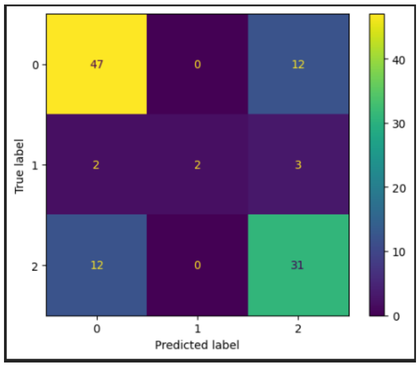
```
Accuracy: 0.6972477064220184
Accuracy on training set: 0.901
Accuracy on test set: 0.734
```

Visualization of decision tree learning curve and confusion matrix and feature importance:



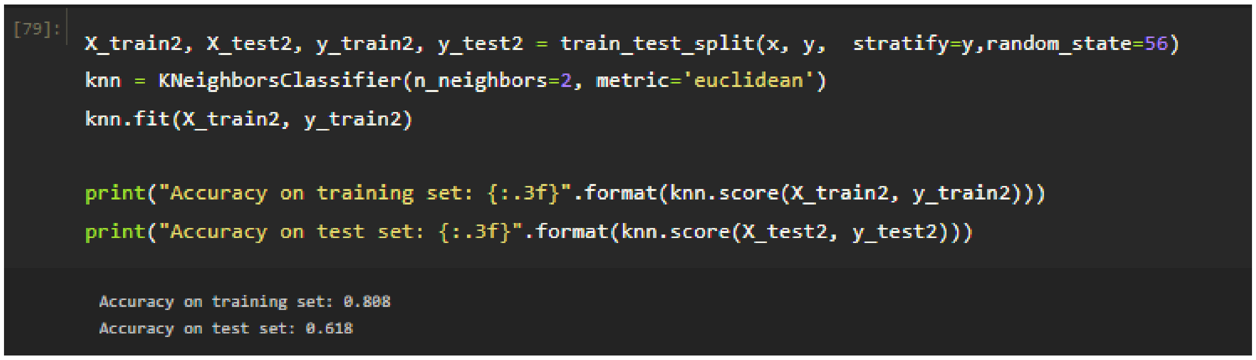
Here we can see that the overfitting have been fixed and the accuracy of the testing have been increased from 0.697 to 0.734





3. K-Nearest Neighbors (KNN)

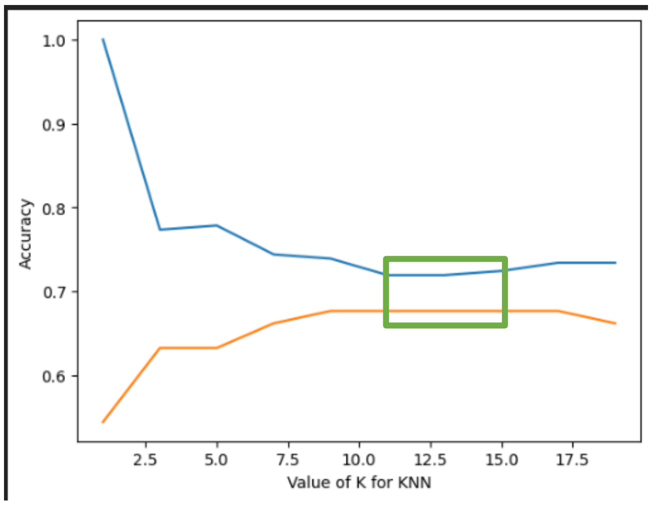
Training the model with random number of neighbors (K)



```
[79]: X_train2, X_test2, y_train2, y_test2 = train_test_split(x, y,  stratify=y,random_state=56)
      knn = KNeighborsClassifier(n_neighbors=2, metric='euclidean')
      knn.fit(X_train2, y_train2)

      print("Accuracy on training set: {:.3f}".format(knn.score(X_train2, y_train2)))
      print("Accuracy on test set: {:.3f}".format(knn.score(X_test2, y_test2)))
```

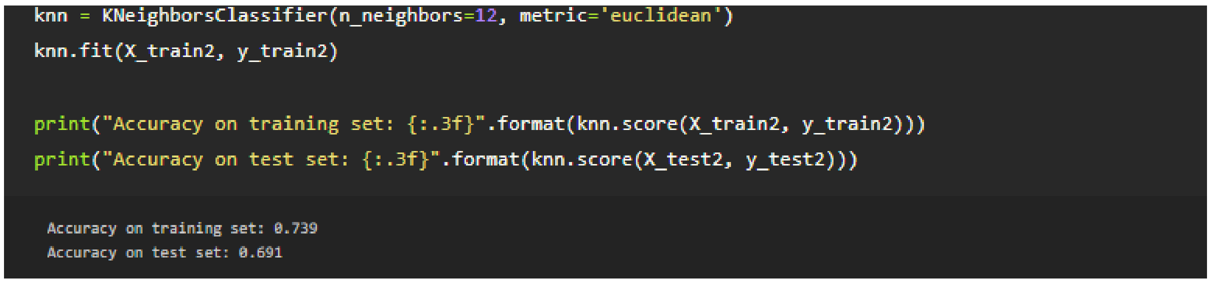
```
Accuracy on training set: 0.808
Accuracy on test set: 0.618
```

Then are going to see the accuracy of different to choose the most optimal one.



The green should be where the most optimal K is.

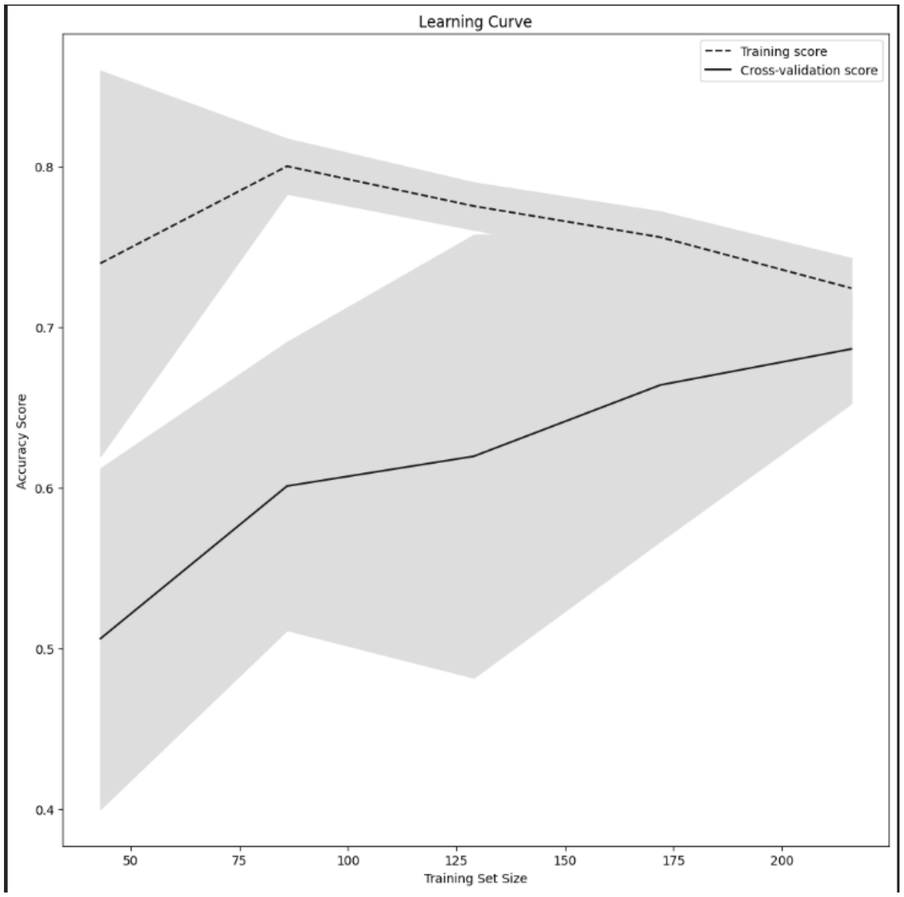
We will choose 12

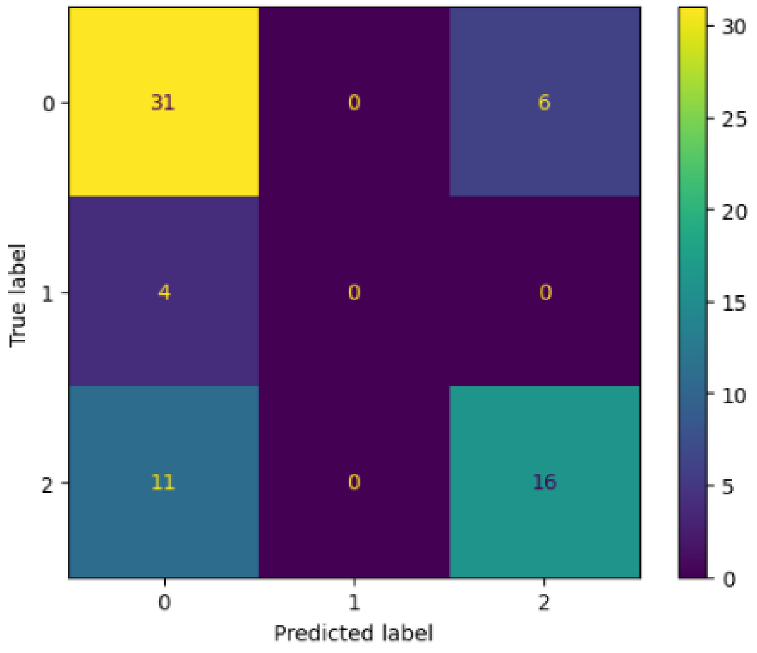


```
knn = KNeighborsClassifier(n_neighbors=12, metric='euclidean')
knn.fit(X_train2, y_train2)

print("Accuracy on training set: {:.3f}".format(knn.score(X_train2, y_train2)))
print("Accuracy on test set: {:.3f}".format(knn.score(X_test2, y_test2)))
```

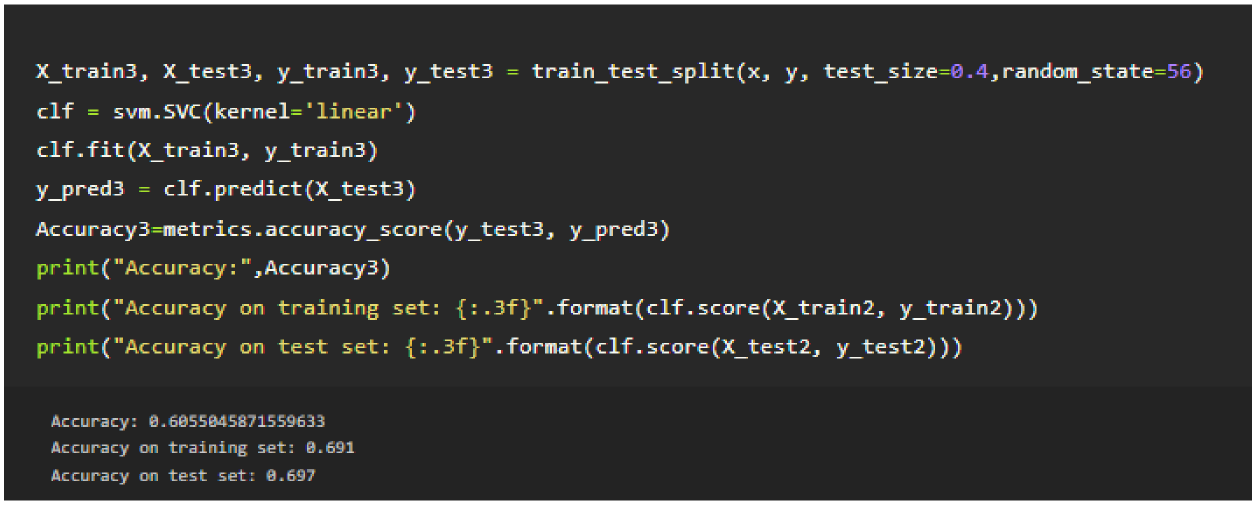
```
Accuracy on training set: 0.739
Accuracy on test set: 0.691
```

Visualization learning curve and confusion matrix :





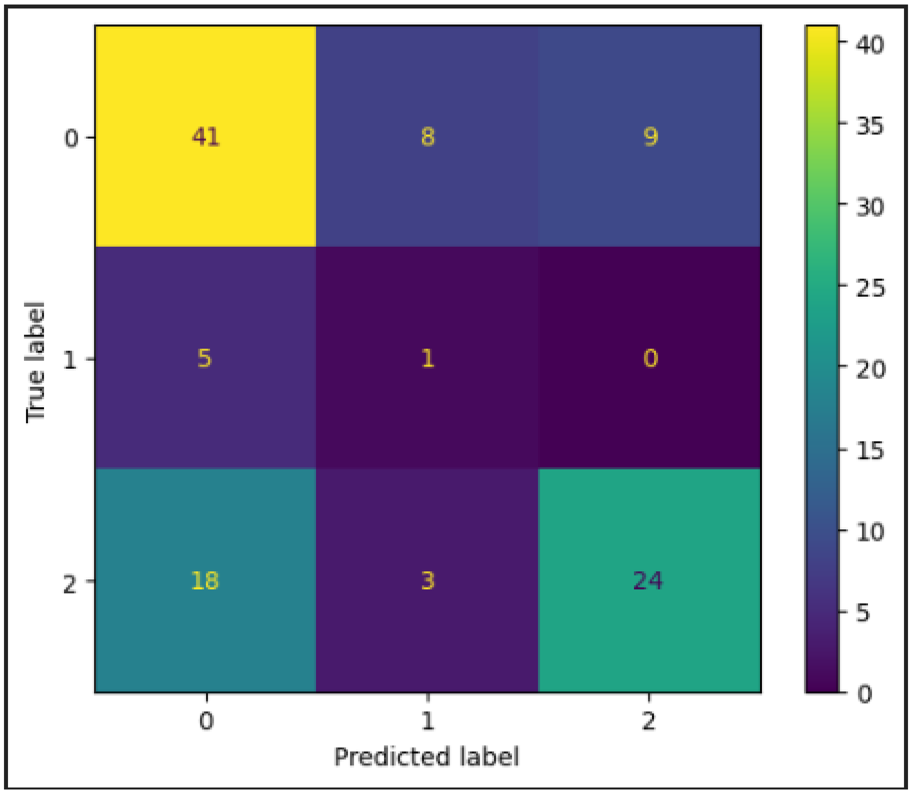
4. Support Vector Machine (SVM)
   Training the model



```
X_train3, X_test3, y_train3, y_test3 = train_test_split(x, y, test_size=0.4,random_state=56)
clf = svm.SVC(kernel='linear')
clf.fit(X_train3, y_train3)
y_pred3 = clf.predict(X_test3)
Accuracy3=metrics.accuracy_score(y_test3, y_pred3)
print("Accuracy:",Accuracy3)
print("Accuracy on training set: {:.3f}".format(clf.score(X_train2, y_train2)))
print("Accuracy on test set: {:.3f}".format(clf.score(X_test2, y_test2)))
```

```
Accuracy: 0.6055045871559633
Accuracy on training set: 0.691
Accuracy on test set: 0.697
```

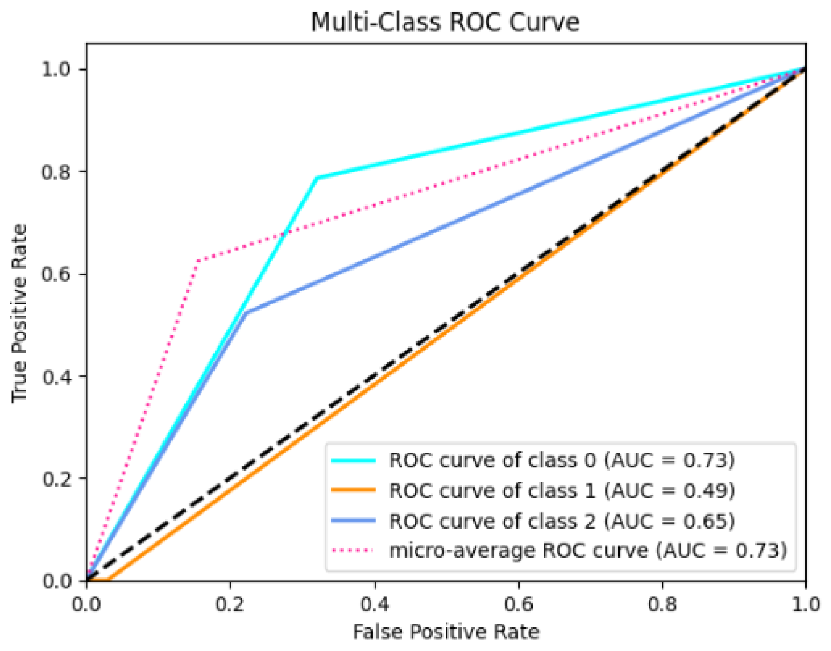
Visualization confusion matrix :

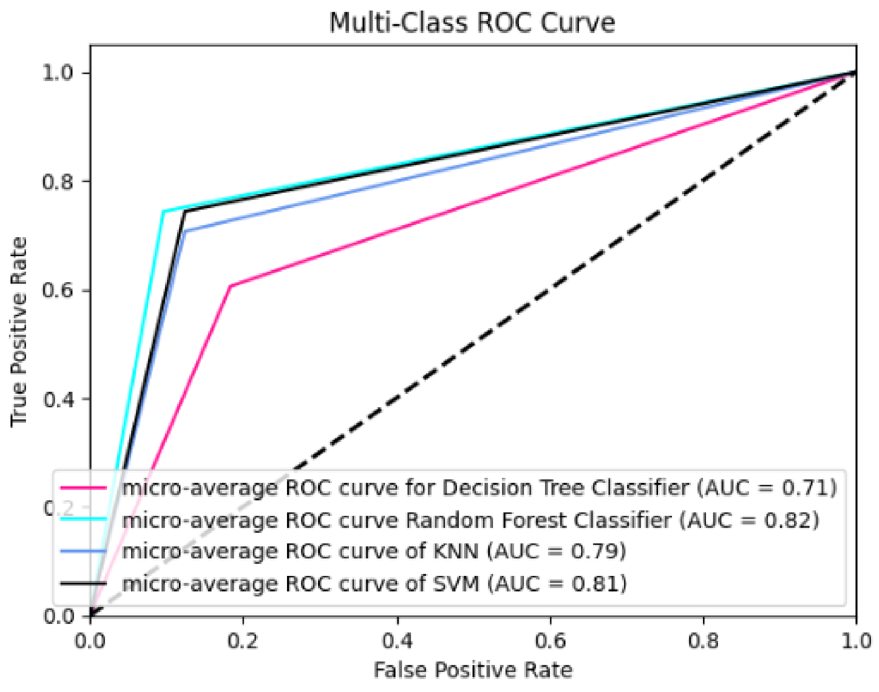


Comparing between models using Accuracy

Since we have three class we will have to use binarization to be able to woke that by using one vs the rest classification then calculate the average

Example of one model





Form the roc curve we can see that best one is Random forest classification

Regression:

Airfoil Self-Noise:

NASA data set, acquired during a series of studies in an anechoic wind tunnel, involving the aerodynamic and acoustic properties of two and three-dimensional airfoil blade sections. This data set contains 1503 instances. Where the target is Scaled sound pressure level, in decibels.

Fig1.

```
column_headers = [
    "Frequency", "Angle of attack",
    " Chord length", " Free-stream velocity", "Suction side displacement thickness"," Scaled sound pressure level"
]
```

In Fig1 we add all the column headers names in order to add them to our dataset that is without the headers

Fig2.

```
airfoil.isnull().sum() # get the sum of the null values in the data
```

In Fig2. we check to see the sum of the nulls and there appeared to be no nulls in our data.

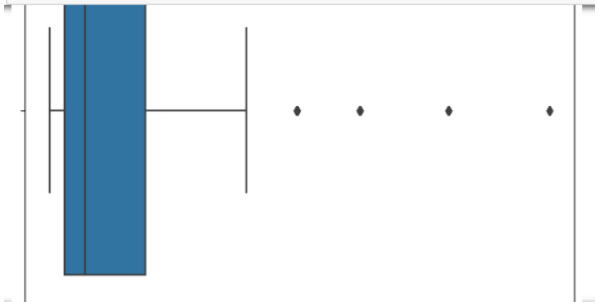
Fig3.

```
airfoil.dtypes  #get the data type
```

In fig3 we checked the datatypes in order to make sure that it's going to function properly.

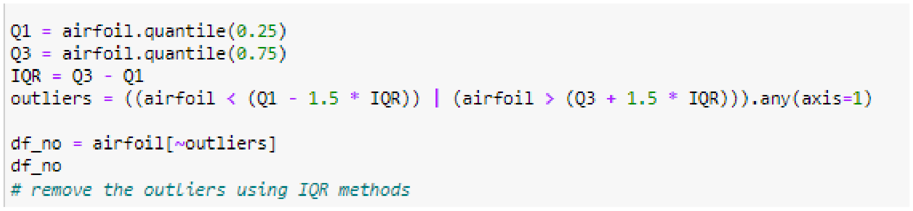
Fig4.

```
for column in column_headers:
    plt.figure(figsize=(8, 6))  # Adjust the figure size if needed
    sns.boxplot(x=airfoil[column])
    plt.title(f'Box Plot for {column}')
    plt.xlabel(column)
    plt.show()
     # plotting the outliers for every column
```



In fig4 we used to headers of the columns in a for loop in order to find the outliers in them and present it in a plot.

Fig5.

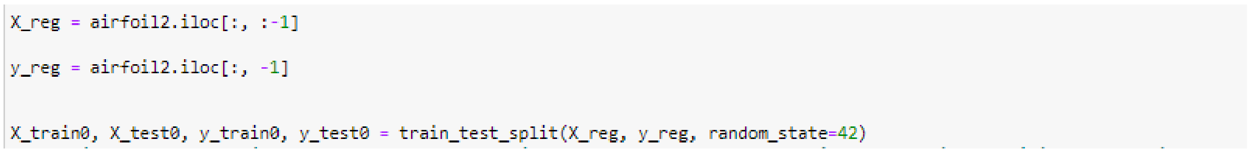


```
Q1 = airfoil.quantile(0.25)
Q3 = airfoil.quantile(0.75)
IQR = Q3 - Q1
outliers = ((airfoil < (Q1 - 1.5 * IQR)) | (airfoil > (Q3 + 1.5 * IQR))).any(axis=1)

df_no = airfoil[~outliers]
df_no
# remove the outliers using IQR methods
```

In fig5 we tried different quantiles until we reached these in the figure due to the fact that it only removed a few of the data rather than half of it.

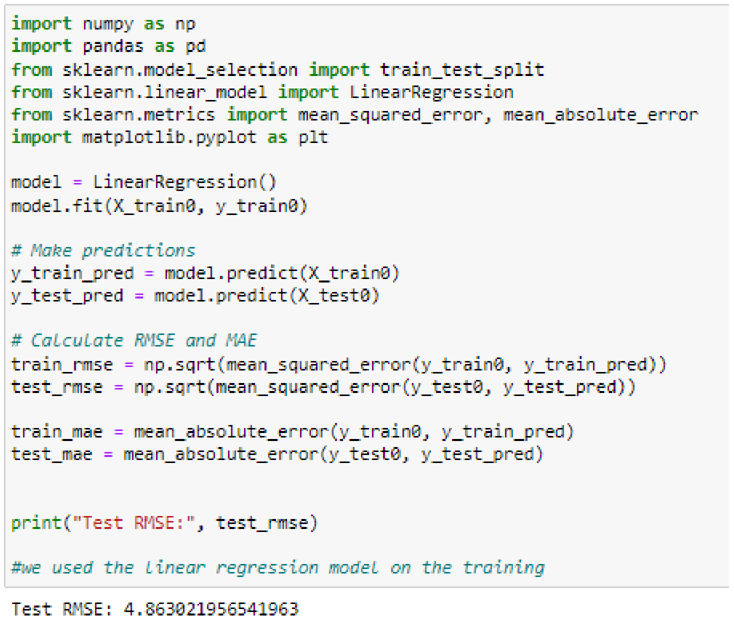
Models

Fig6.



```
X_reg = airfoil2.iloc[:, :-1]

y_reg = airfoil2.iloc[:, -1]


X_train0, X_test0, y_train0, y_test0 = train_test_split(X_reg, y_reg, random_state=42)
```

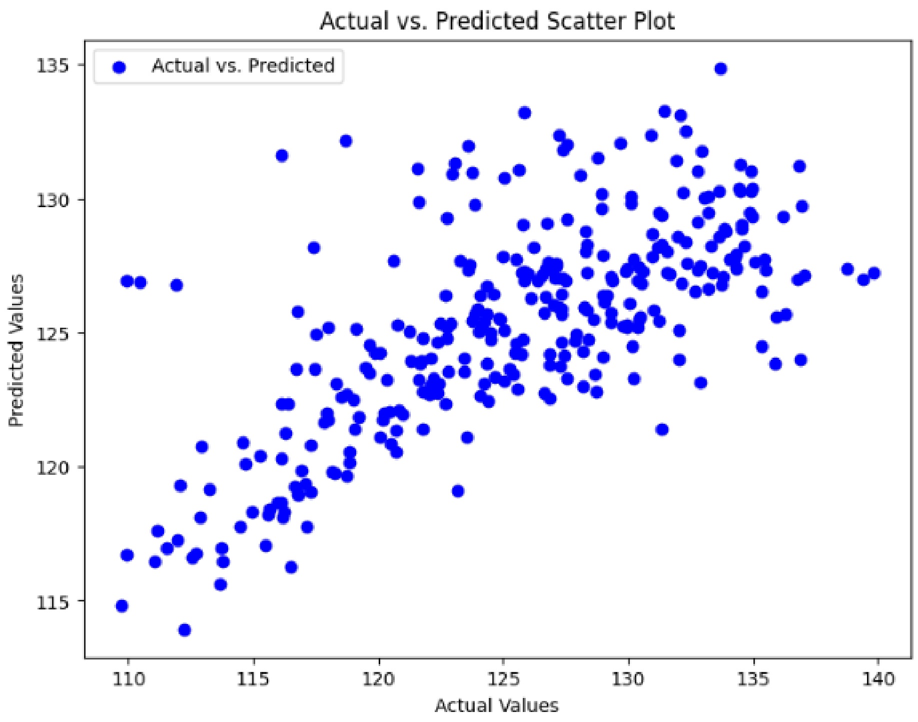
We assign x and y as x holds all the features except the target column and y has only the target column then we split the data into train and test.



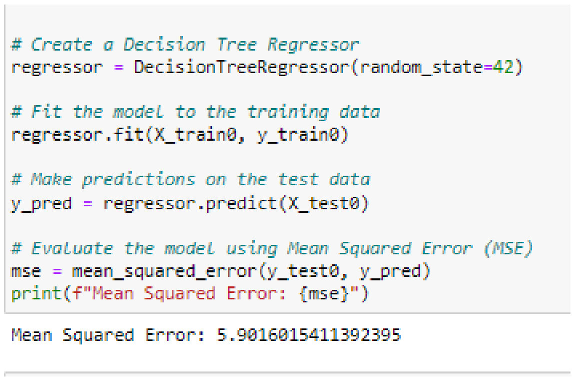
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

model = LinearRegression()
model.fit(X_train0, y_train0)

# Make predictions
y_train_pred = model.predict(X_train0)
y_test_pred = model.predict(X_test0)

# Calculate RMSE and MAE
train_rmse = np.sqrt(mean_squared_error(y_train0, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test0, y_test_pred))

train_mae = mean_absolute_error(y_train0, y_train_pred)
test_mae = mean_absolute_error(y_test0, y_test_pred)


print("Test RMSE:", test_rmse)

#we used the linear regression model on the training
```

```
Test RMSE: 4.863021956541963
```

We used the linear regression model without any parameters while it trains the linear regression model on a training dataset then print the RMSE.



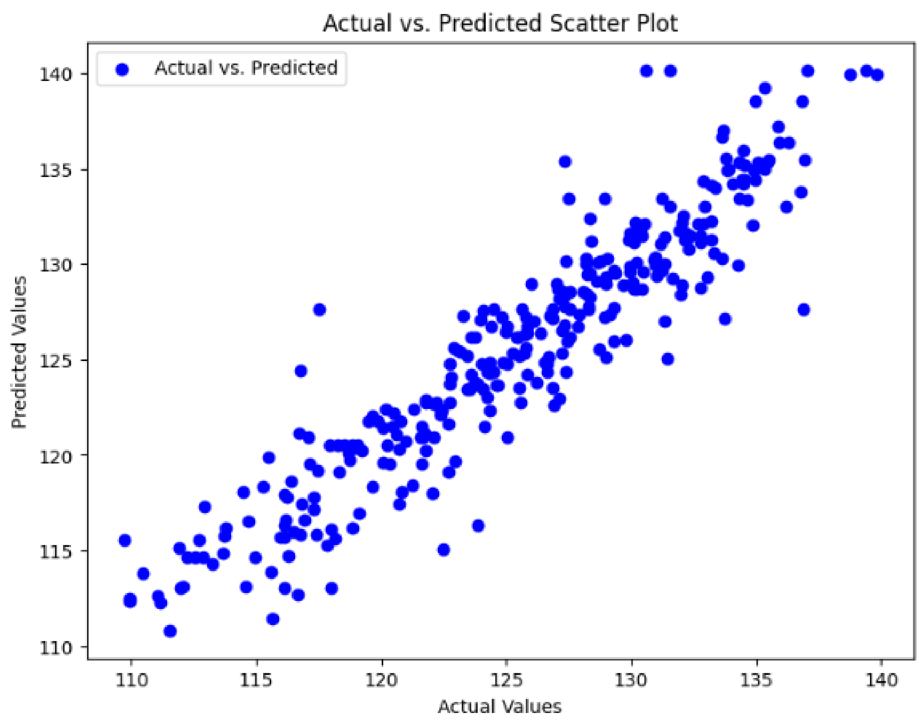
We used the scatter plot to visualize the predicted values against the actual values



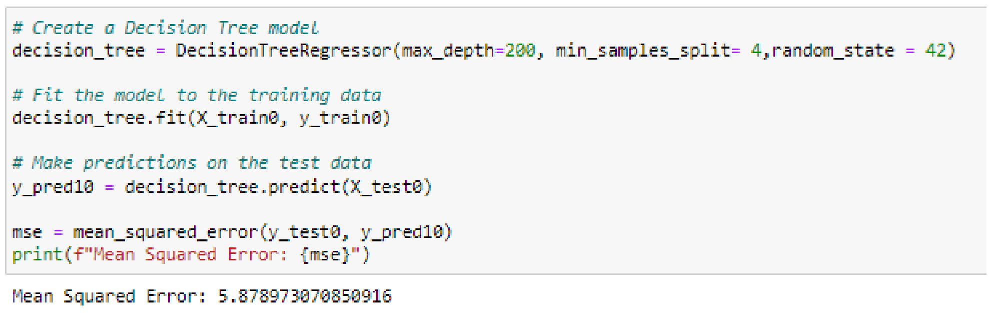
```
# Create a Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)

# Fit the model to the training data
regressor.fit(X_train0, y_train0)

# Make predictions on the test data
y_pred = regressor.predict(X_test0)

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test0, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 5.9016015411392395
```

We used the Decision Tree Regressor model with random state while it trains Decision Tree Regressor model on a training dataset then print the RMSE.



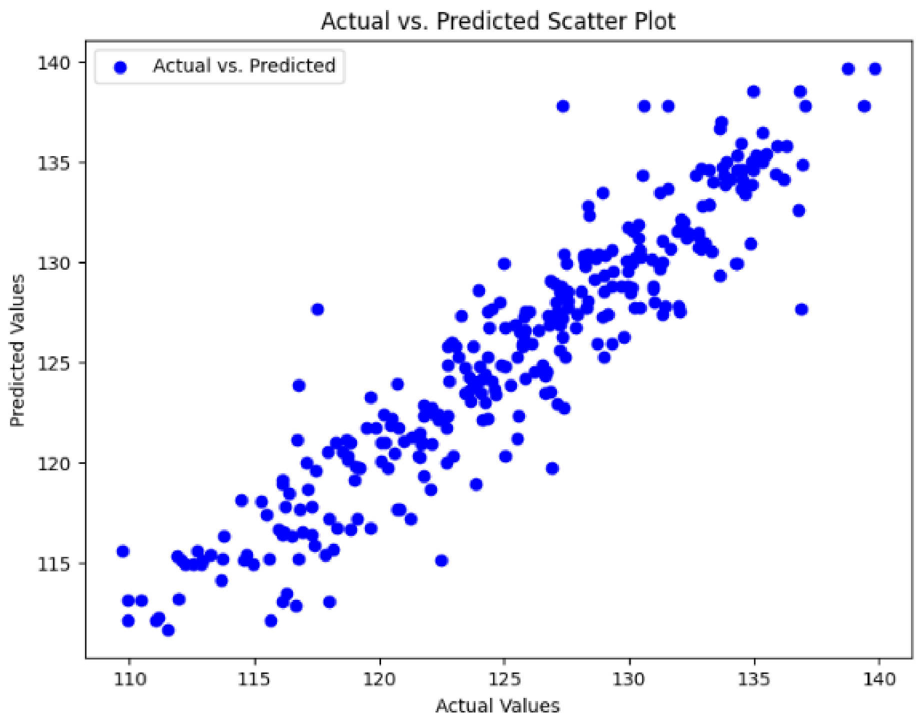
We used the scatter plot to visualize the predicted values against the actual values.



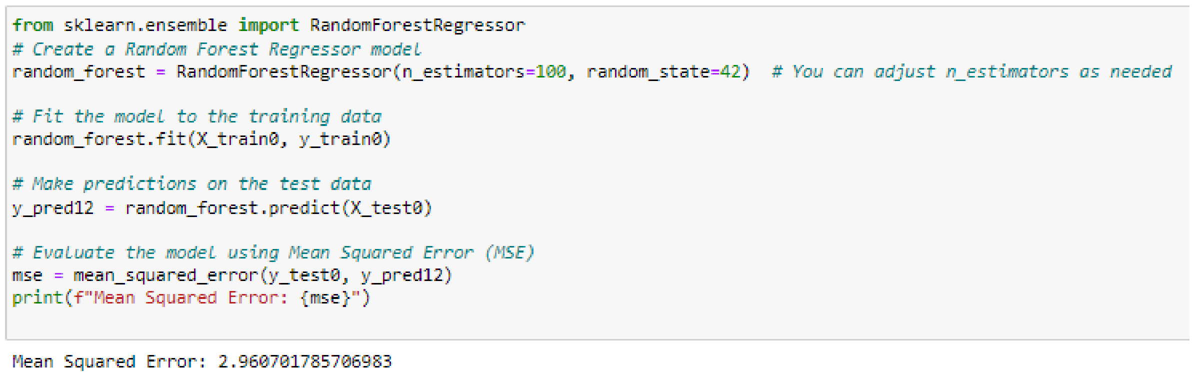
```
# Create a Decision Tree model
decision_tree = DecisionTreeRegressor(max_depth=200, min_samples_split= 4,random_state = 42)

# Fit the model to the training data
decision_tree.fit(X_train0, y_train0)

# Make predictions on the test data
y_pred10 = decision_tree.predict(X_test0)

mse = mean_squared_error(y_test0, y_pred10)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 5.878973070850916
```

We used the Decision Tree Regressor model with the max_depth and min split samples as it need 4 samples in a node inorder to split it while it trains Decision Tree Regressor model on a training dataset then print the RMSE.



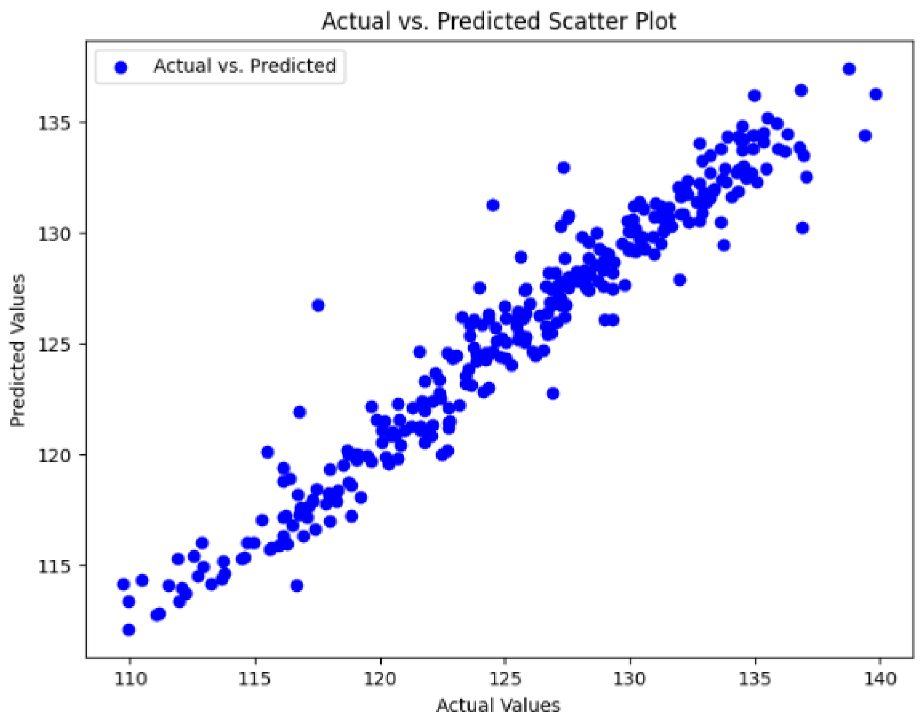
We used the scatter plot to visualize the predicted values against the actual values



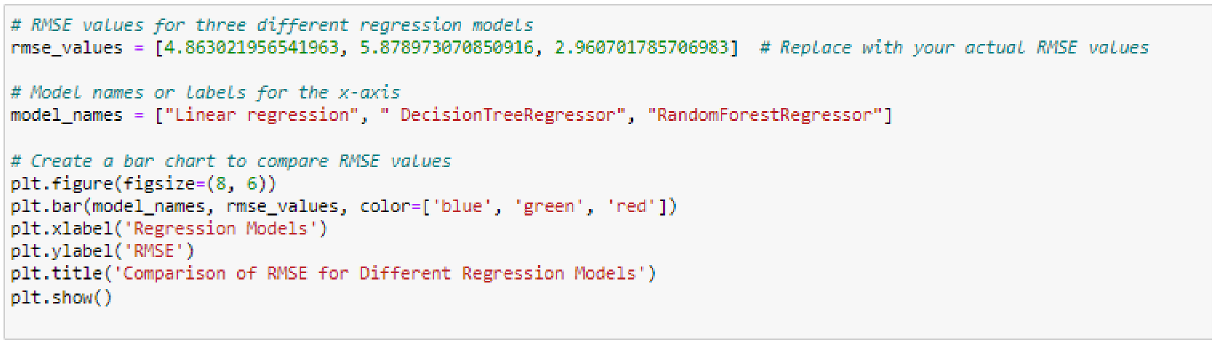
```
from sklearn.ensemble import RandomForestRegressor
# Create a Random Forest Regressor model
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)  # You can adjust n_estimators as needed

# Fit the model to the training data
random_forest.fit(X_train0, y_train0)

# Make predictions on the test data
y_pred12 = random_forest.predict(X_test0)

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test0, y_pred12)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 2.960701785706983
```

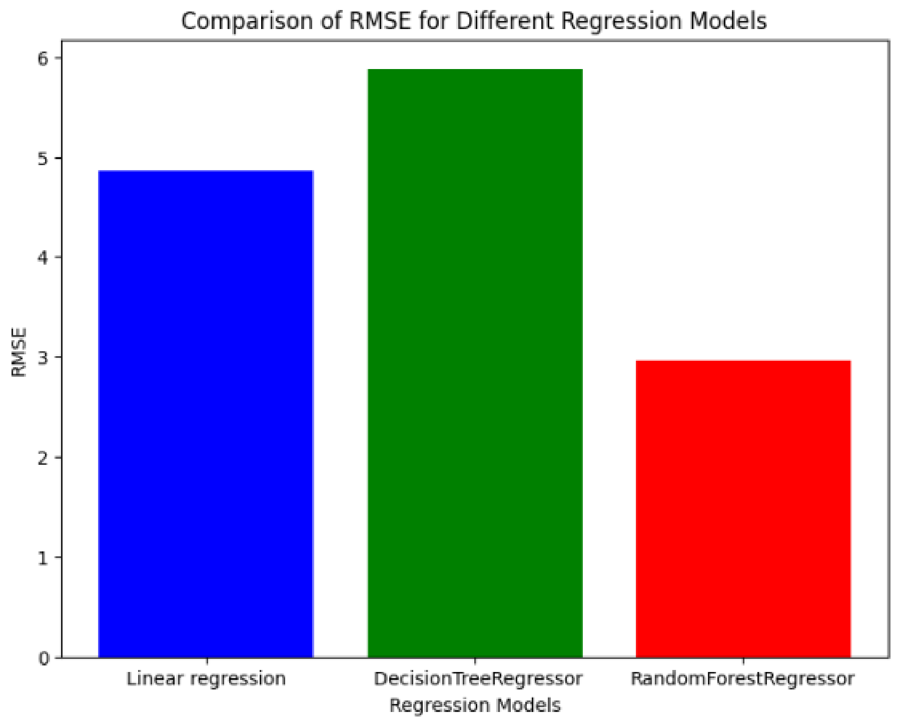
We used the Random Forest Regressor model with number of estimators only while it trains Random Forest Regressor on a training dataset then print the RMSE.



We used the scatter plot to visualize the predicted values against the actual values.



```
# RMSE values for three different regression models
rmse_values = [4.863021956541963, 5.878973070850916, 2.960701785706983]  # Replace with your actual RMSE values

# Model names or labels for the x-axis
model_names = ["Linear regression", " DecisionTreeRegressor", "RandomForestRegressor"]

# Create a bar chart to compare RMSE values
plt.figure(figsize=(8, 6))
plt.bar(model_names, rmse_values, color=['blue', 'green', 'red'])
plt.xlabel('Regression Models')
plt.ylabel('RMSE')
plt.title('Comparison of RMSE for Different Regression Models')
plt.show()
```

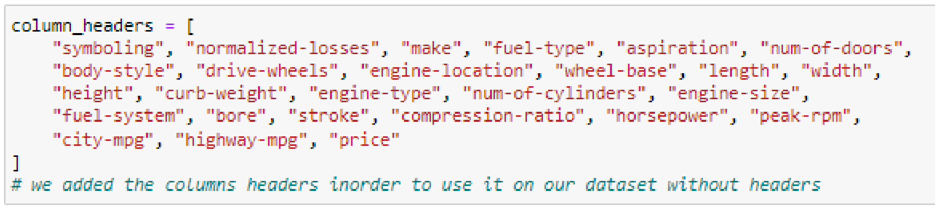


We took the RMSE of all the regression models used and compared to see which one has the lowest RMSE and then choose it.

Automobile Regression Dataset

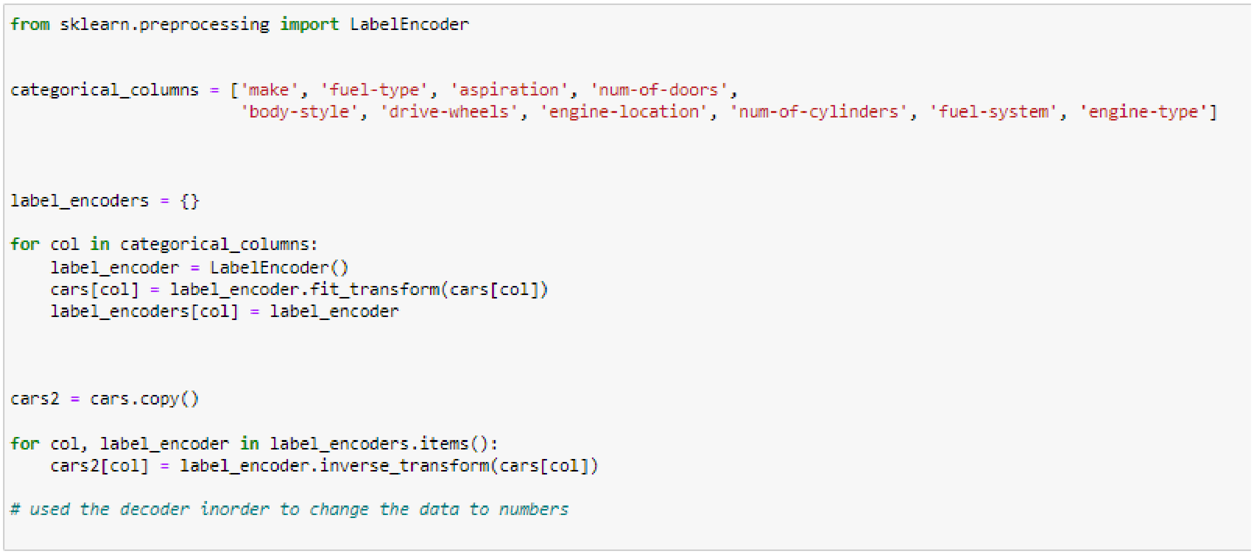
Dataset Description

This data set includes three different types of entities, 25 features, and 205 instances. (a) the car's specification in terms of different attributes; (b) the insurance risk rating it has been given; and (c) the normalised losses it has incurred in comparison to other cars. The degree to which the car is riskier than its pricing suggests is indicated by the second rating. At first, a risk factor symbol related to the car's price is assigned to it. Next, this symbol is moved up (or down) the scale to indicate how riskier (or less) it is. This method is known as "symbolling" by actuaries. If the auto has a value of +3, it is dangerous; if it has a value of -3, it is probably safe.



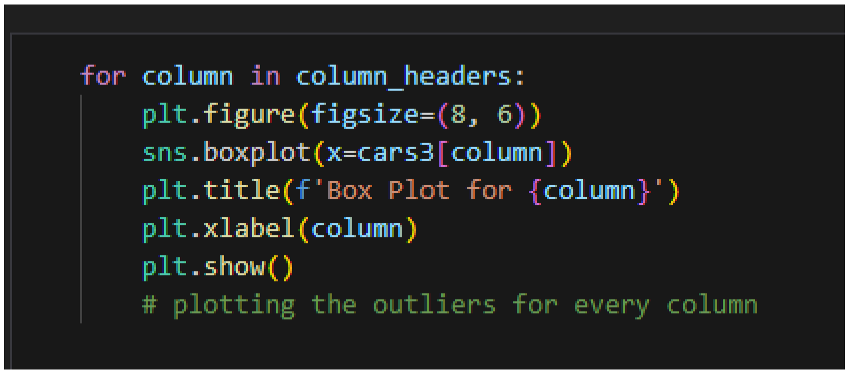
```
column_headers = [
    "symboling", "normalized-losses", "make", "fuel-type", "aspiration", "num-of-doors",
    "body-style", "drive-wheels", "engine-location", "wheel-base", "length", "width",
    "height", "curb-weight", "engine-type", "num-of-cylinders", "engine-size",
    "fuel-system", "bore", "stroke", "compression-ratio", "horsepower", "peak-rpm",
    "city-mpg", "highway-mpg", "price"
]
# we added the columns headers inorder to use it on our dataset without headers
```

we add all the column headers names in order to add them to our dataset that is without the headers

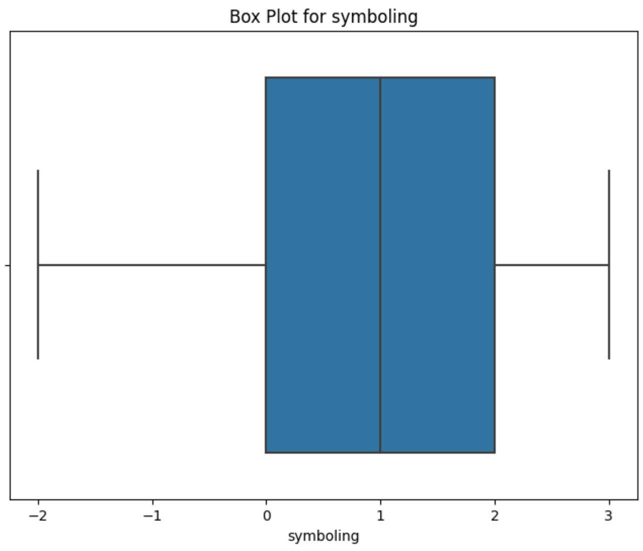


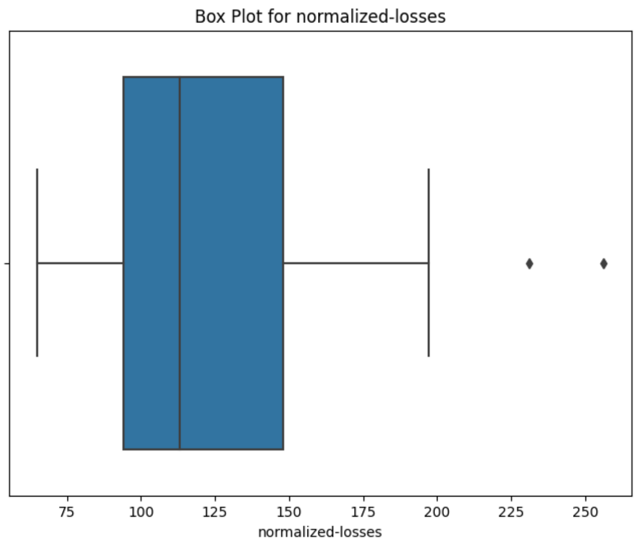
```
from sklearn.preprocessing import LabelEncoder


categorical_columns = ['make', 'fuel-type', 'aspiration', 'num-of-doors',
                       'body-style', 'drive-wheels', 'engine-location', 'num-of-cylinders', 'fuel-system', 'engine-type']



label_encoders = {}

for col in categorical_columns:
    label_encoder = LabelEncoder()
    cars[col] = label_encoder.fit_transform(cars[col])
    label_encoders[col] = label_encoder



cars2 = cars.copy()

for col, label_encoder in label_encoders.items():
    cars2[col] = label_encoder.inverse_transform(cars[col])

# used the decoder inorder to change the data to numbers
```

Seeing the outlier visualization

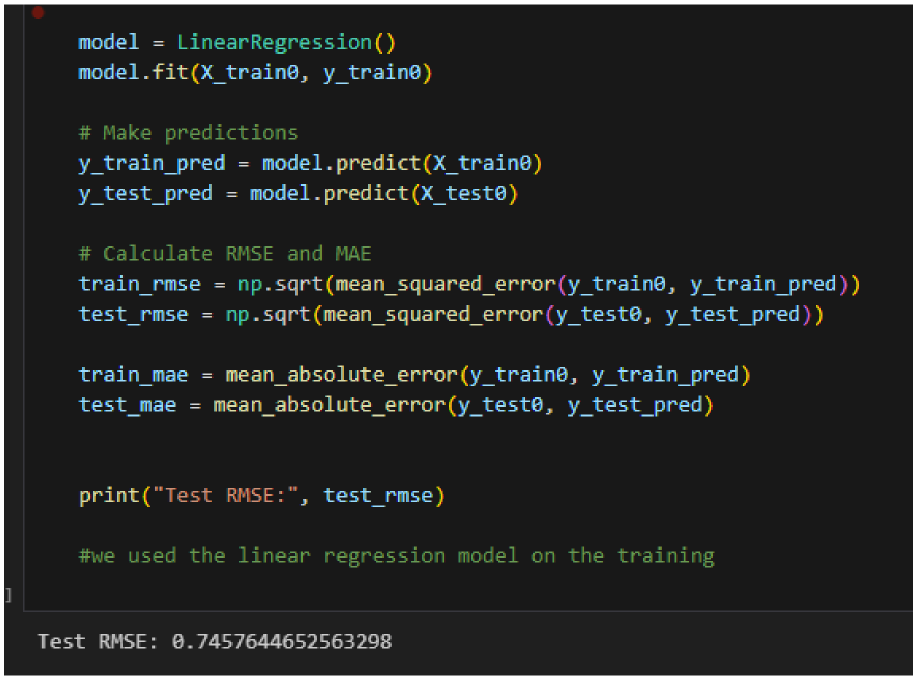


```
for column in column_headers:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x=cars3[column])
    plt.title(f'Box Plot for {column}')
    plt.xlabel(column)
    plt.show()
    # plotting the outliers for every column
```





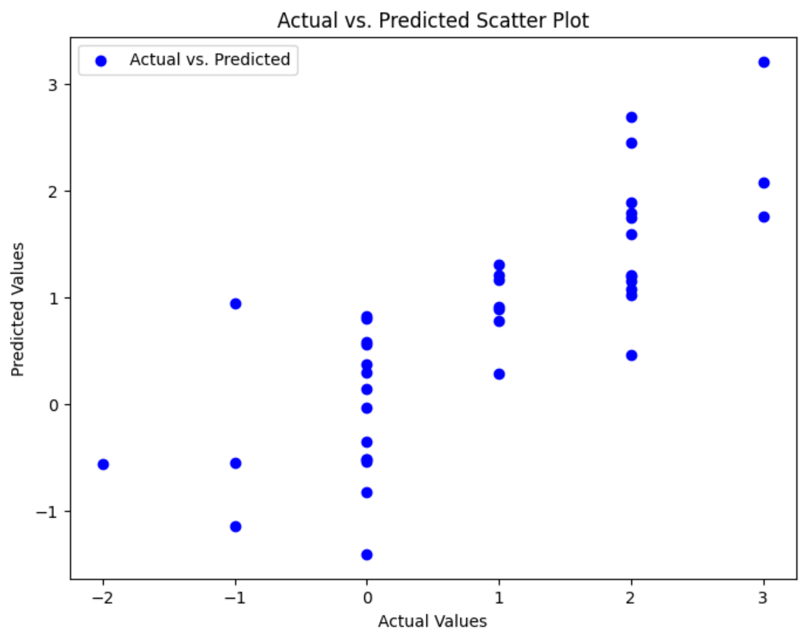
We used the linear regression model without any parameters while it trains the linear regression model on a training dataset then print the RMSE.



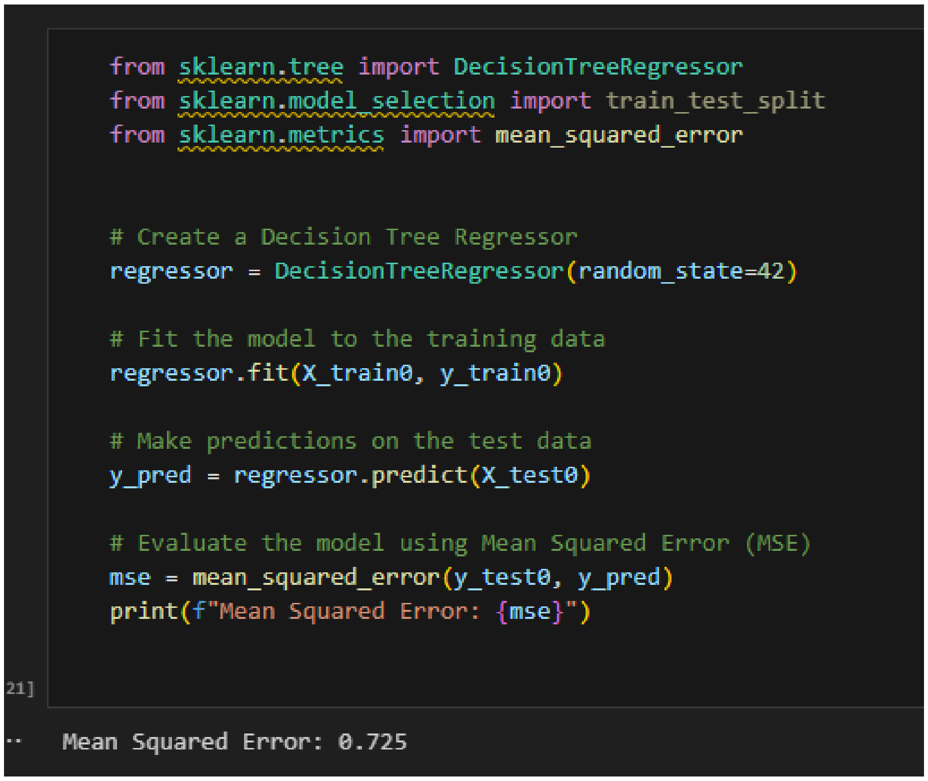
```
model = LinearRegression()
model.fit(X_train0, y_train0)

# Make predictions
y_train_pred = model.predict(X_train0)
y_test_pred = model.predict(X_test0)

# Calculate RMSE and MAE
train_rmse = np.sqrt(mean_squared_error(y_train0, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test0, y_test_pred))

train_mae = mean_absolute_error(y_train0, y_train_pred)
test_mae = mean_absolute_error(y_test0, y_test_pred)


print("Test RMSE:", test_rmse)

#we used the linear regression model on the training
```

```
Test RMSE: 0.7457644652563298
```

The liner regression showing vs predicted scatter plot



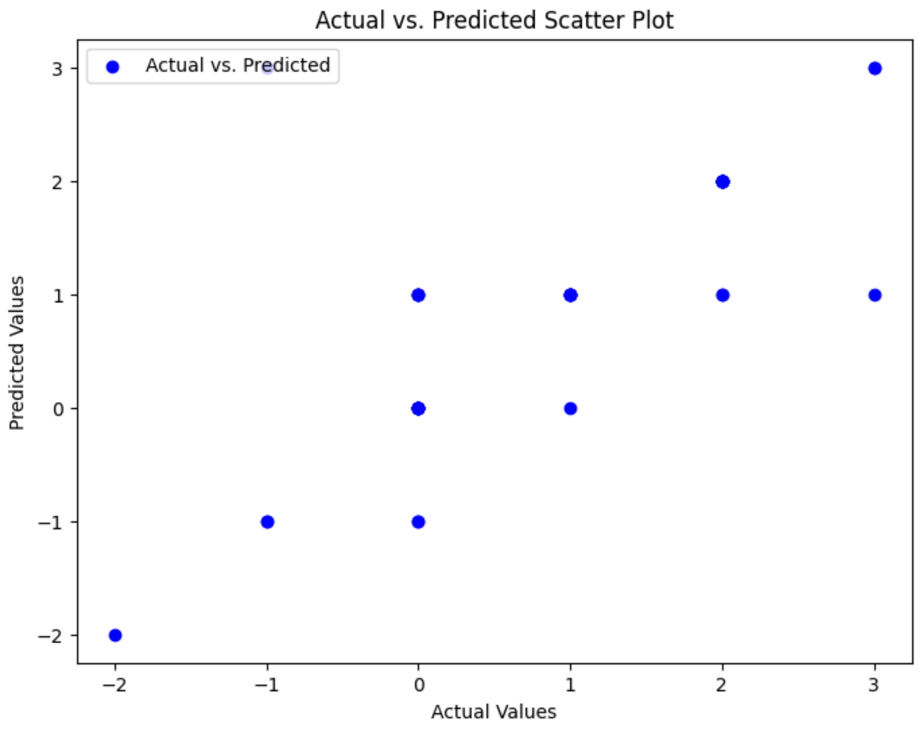
We used the Decision Tree Regressor model with random state while it trains Decision Tree Regressor model on a training dataset then print the RMSE.



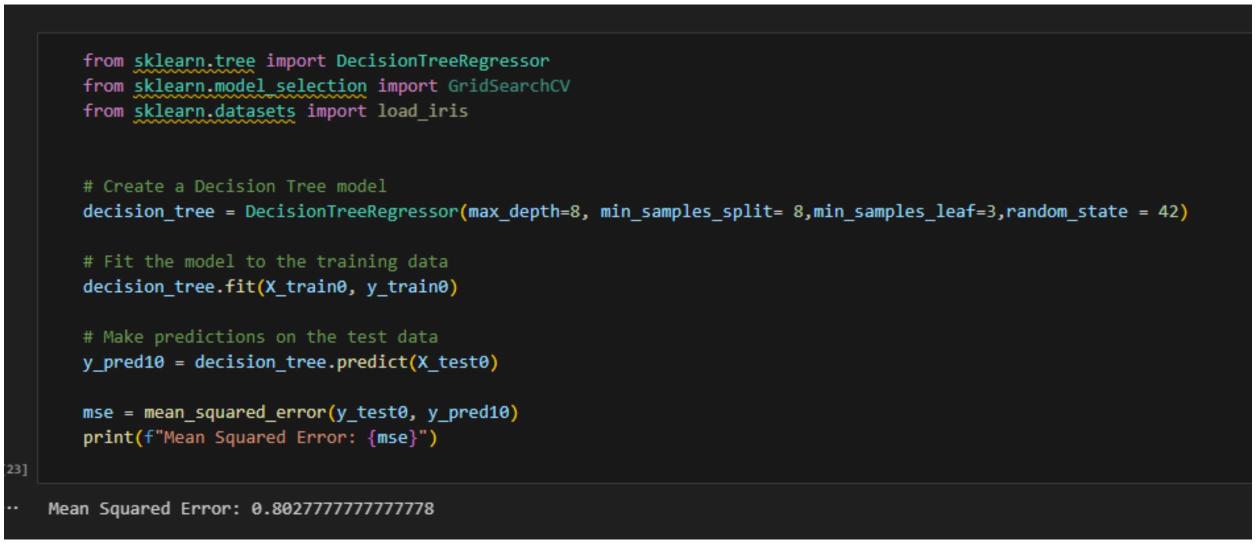
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


# Create a Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)

# Fit the model to the training data
regressor.fit(X_train0, y_train0)

# Make predictions on the test data
y_pred = regressor.predict(X_test0)

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test0, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 0.725
```

The Decision tree regressor showing vs predicted scatter plot



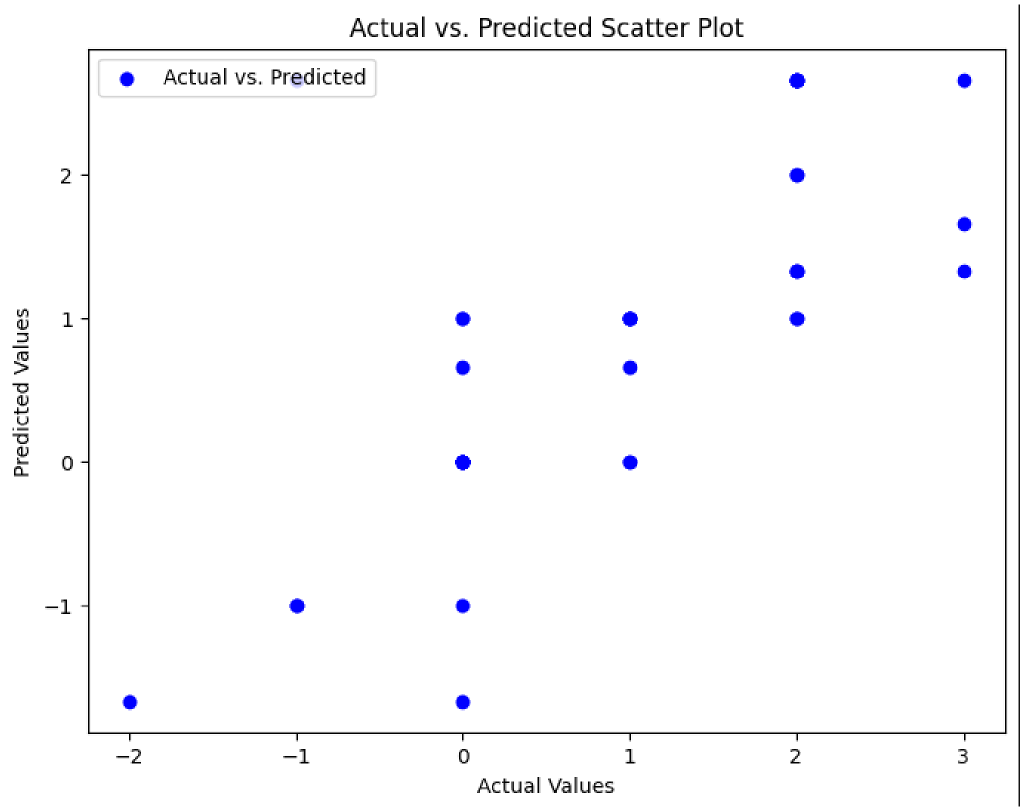
We used the Decision Tree Regressor model with the max_depth and min split samples as it need 4 samples in a node in order to split it while it trains Decision Tree Regressor model on a training dataset then print the RMSE.



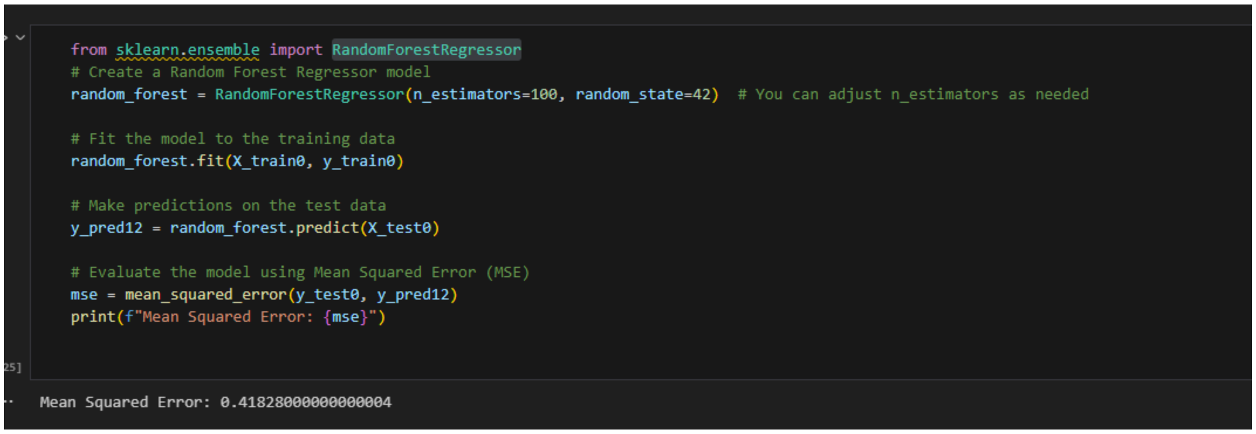
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_iris


# Create a Decision Tree model
decision_tree = DecisionTreeRegressor(max_depth=8, min_samples_split= 8,min_samples_leaf=3,random_state = 42)

# Fit the model to the training data
decision_tree.fit(X_train0, y_train0)

# Make predictions on the test data
y_pred10 = decision_tree.predict(X_test0)

mse = mean_squared_error(y_test0, y_pred10)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 0.8027777777777778
```

The Decision Tree model showing vs predicted scatter plot



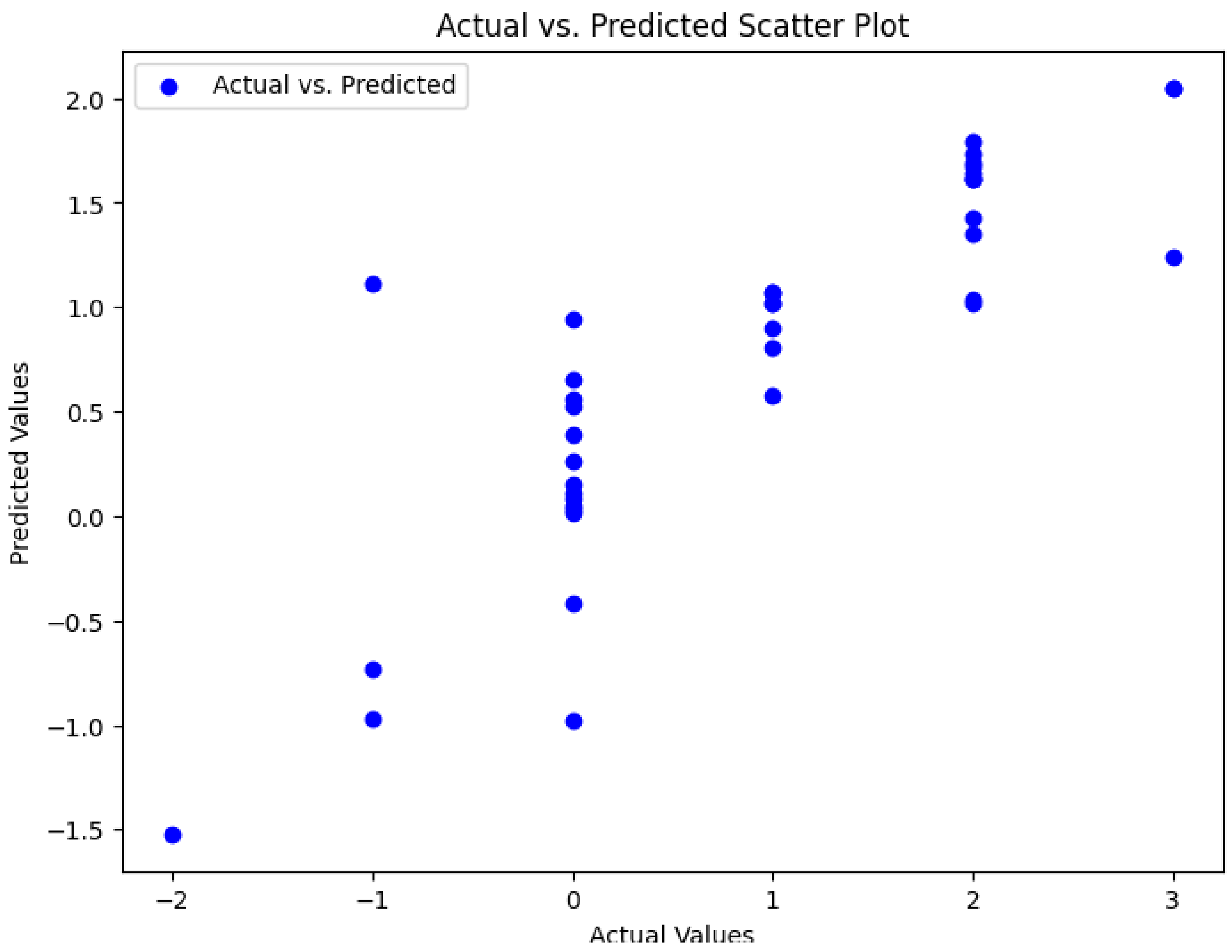
We used the Random Forest Regressor model with number of estimators only while it trains Random Forest Regressor on a training dataset then print the RMSE.



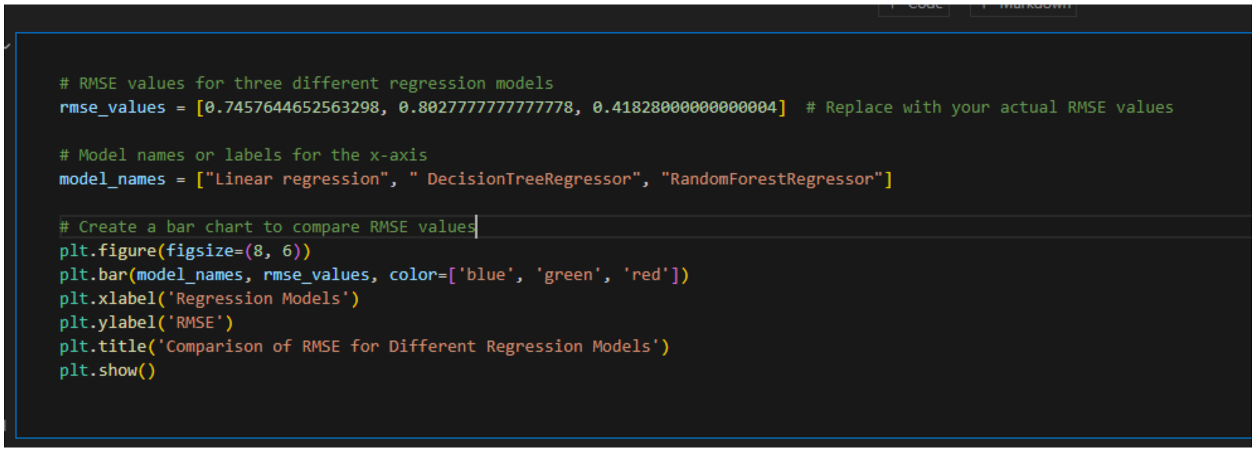
```
from sklearn.ensemble import RandomForestRegressor
# Create a Random Forest Regressor model
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)  # You can adjust n_estimators as needed

# Fit the model to the training data
random_forest.fit(X_train0, y_train0)

# Make predictions on the test data
y_pred12 = random_forest.predict(X_test0)

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test0, y_pred12)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 0.41828000000000004
```

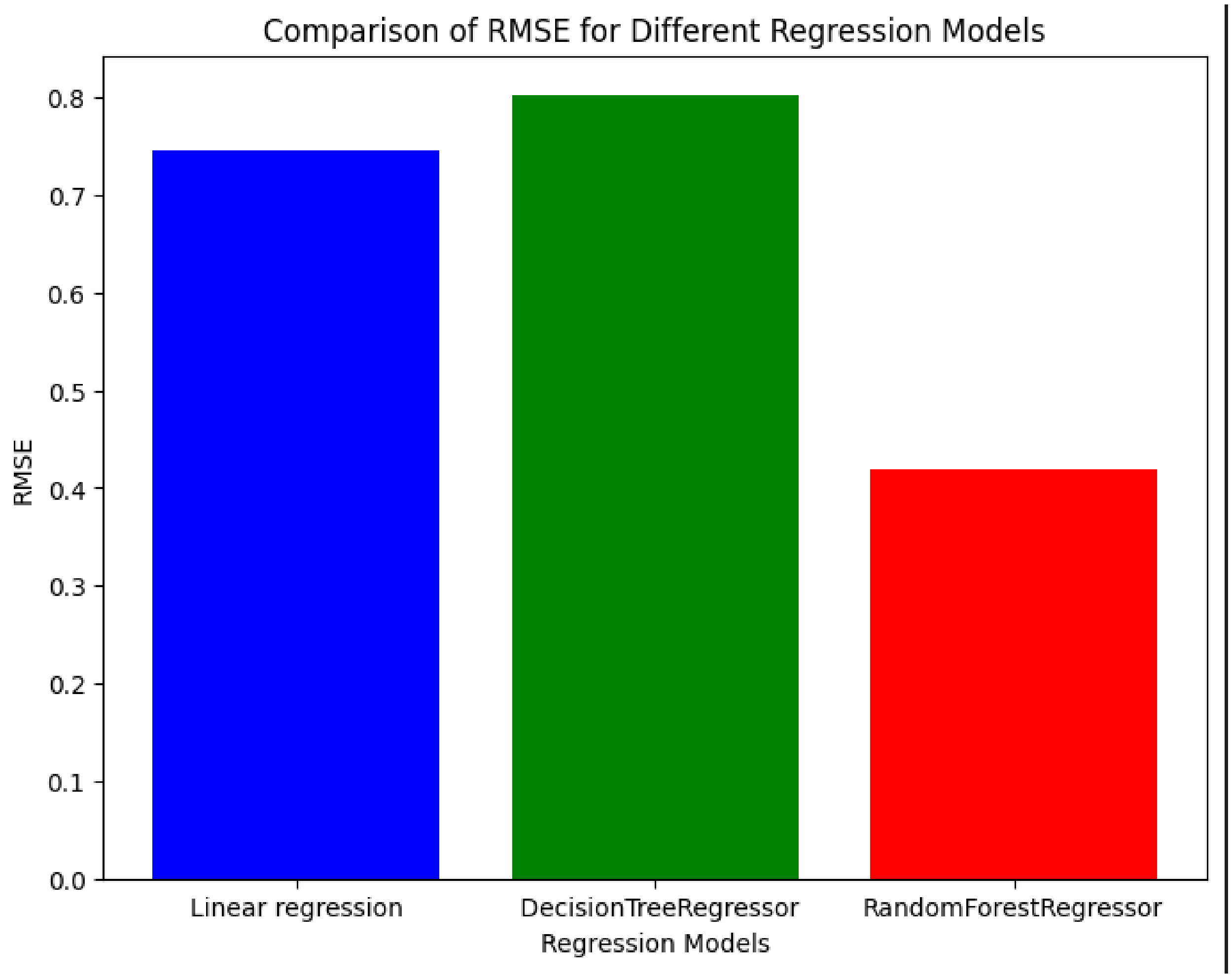
The Random Forest Regressor showing vs predicted scatter plot



Comparison of RMSE for Different Regression Models And we can see that the Linear regressor has the lowest RMSE.



```
# RMSE values for three different regression models
rmse_values = [0.7457644652563298, 0.8027777777777778, 0.41828000000000004]  # Replace with your actual RMSE values

# Model names or labels for the x-axis
model_names = ["Linear regression", " DecisionTreeRegressor", "RandomForestRegressor"]

# Create a bar chart to compare RMSE values
plt.figure(figsize=(8, 6))
plt.bar(model_names, rmse_values, color=['blue', 'green', 'red'])
plt.xlabel('Regression Models')
plt.ylabel('RMSE')
plt.title('Comparison of RMSE for Different Regression Models')
plt.show()
```


Comparison of RMSE for Different Regression Models
RMSE
Linear regression
DecisionTreeRegressor
RandomForestRegressor
Regression Models