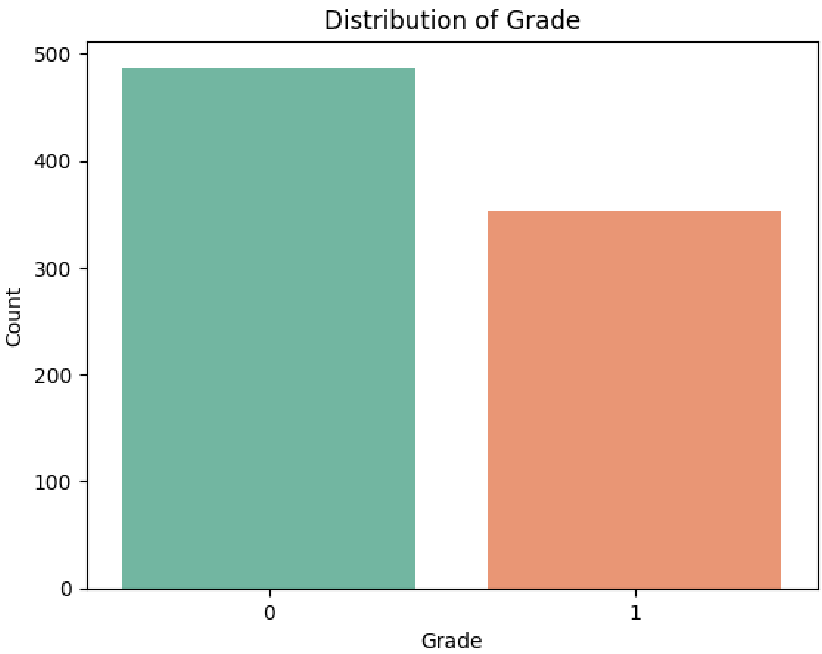
Omar207149:

Dataset : Glioma Grading Clinical and Mutation Features

Info about the dataset

This dataset is made for the detection of Gliomas which are the most common primary tumors of the brain , They can be graded as LGG (Lower-Grade Glioma) or GBM (Glioblastoma Multiforme) , this dataset have 23 features including the target which the grade , this dataset is in the area of health and medicine , there are classes 2 in the dataset 0 = "LGG and 1 = "GBM"

```
Grade
0    487
1    352
Name: count, dtype: int64
```



First the data cleaning of the dataset

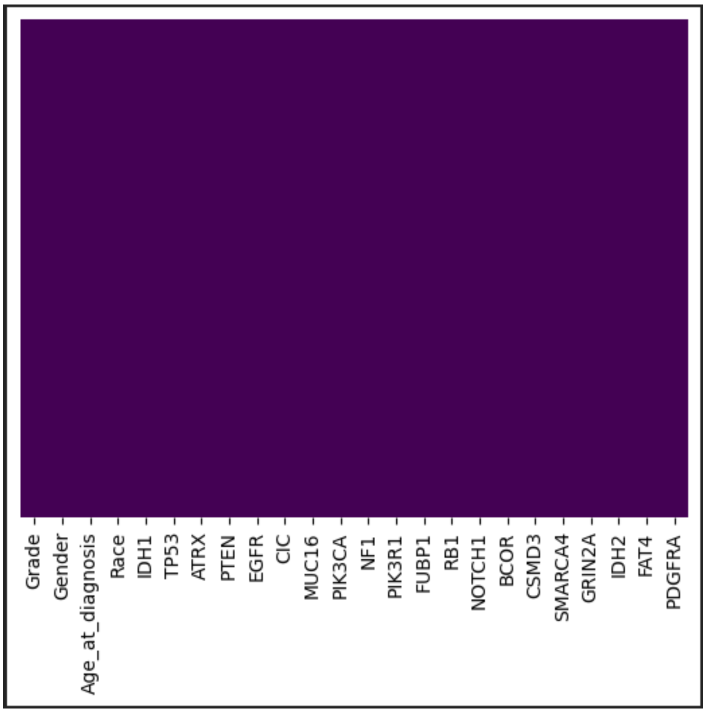
1. Checking for something that need changing

ge_at_diagnosis	Race	IDH1	TP53	ATRX	PTEN	EGFR	CIC	...	FUBP1	RB1	NOTCH1	BCOR	CSMD3	SMARCA4	GRIN2A
.30	0	1	0	0	0	0	0	...	1	0	0	0	0	0	0
.72	0	1	0	0	0	0	1	...	0	0	0	0	0	0	0
.17	0	1	1	1	0	0	0	...	0	0	0	0	0	0	0
.78	0	1	1	1	0	0	0	...	0	0	0	0	0	0	0
.51	0	1	1	1	0	0	0	...	0	0	0	0	0	0	0

Form the look of the data set there isn't any column that need removing all the features are important second all the data are number there are no string variables so there is no labeling needed

2. Checking for nulls

```
Grade               0
Gender              0
Age_at_diagnosis    0
Race                0
IDH1                0
TP53                0
ATRX                0
PTEN                0
EGFR                0
CIC                 0
MUC16               0
PIK3CA              0
NF1                 0
PIK3R1              0
FUBP1               0
RB1                 0
NOTCH1              0
BCOR                0
CSMD3               0
SMARCA4             0
GRIN2A              0
IDH2                0
FAT4                0
PDGFRA              0
dtype: int64
```



From the heat map and the code we can see that there isn't any nulls in the dataset

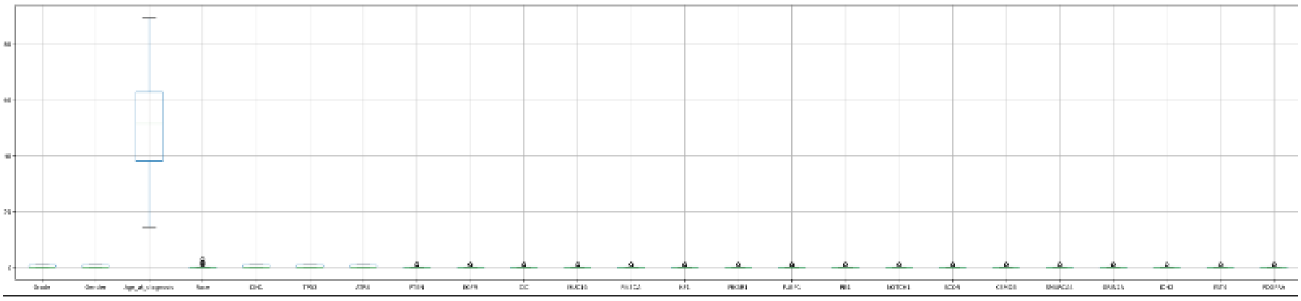
3. Lastly checking for outliers
   For this one we going to try different ones and see which we chose

   - First using quantile ranges

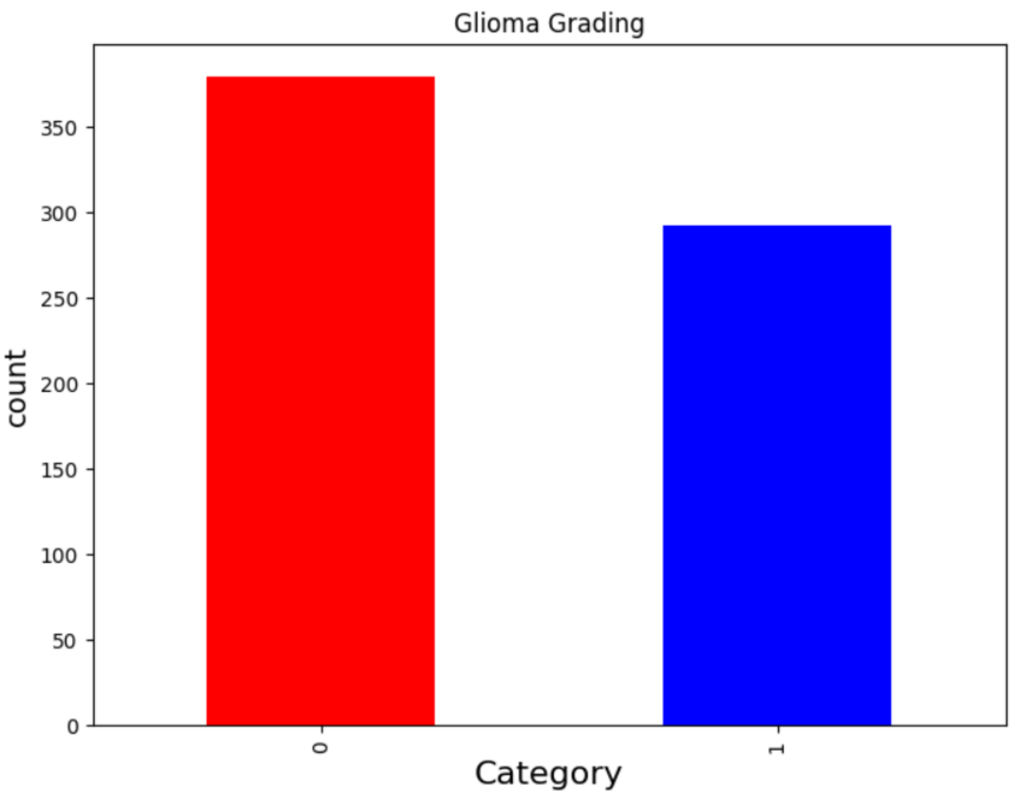
It removes 212

```
Grade
0    365
1    262
Name: count, dtype: int64
```

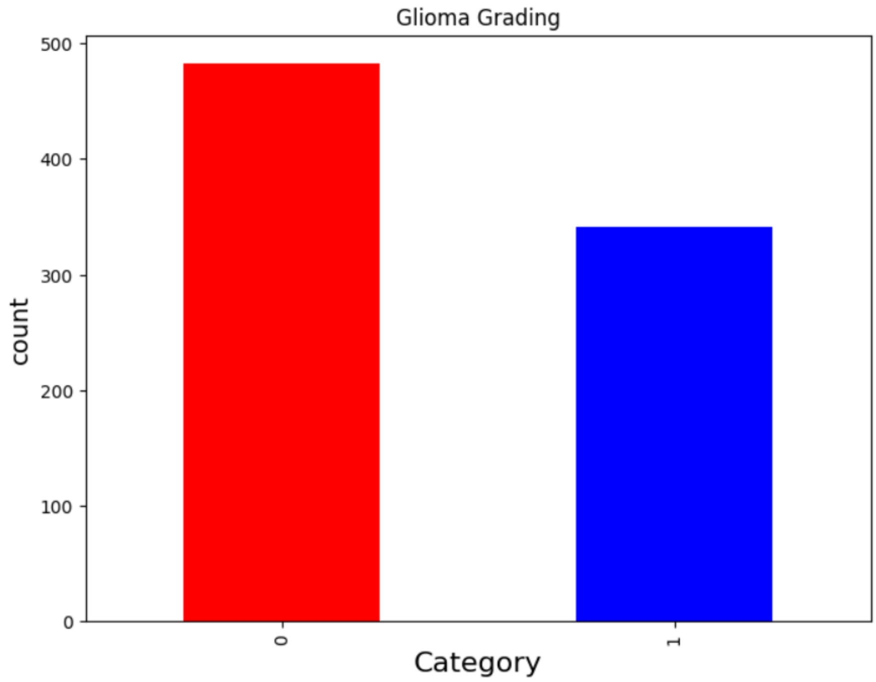
Using boxplot for visualization



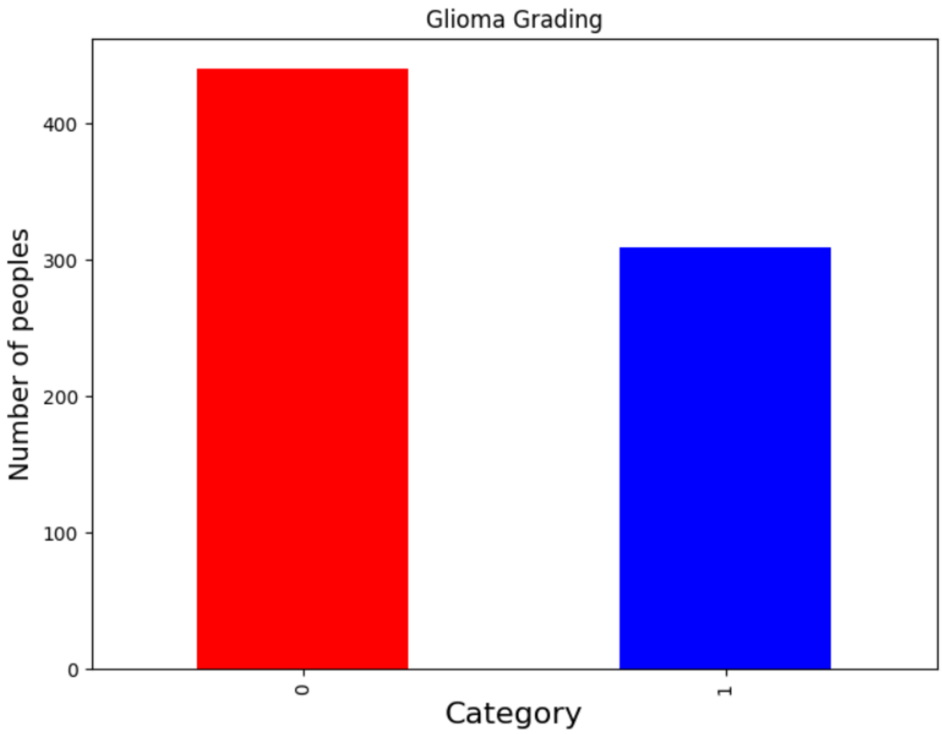
- Elliptic envelope
  Which detected 168 outliers



- Local Outlier Factor
  Which detected 15 outliers



- Isolation Forest
  Which detected 90 outliers



From the outliers detection algorithm we sow we going to use the Isolation Forest

Then we go to the training of the models

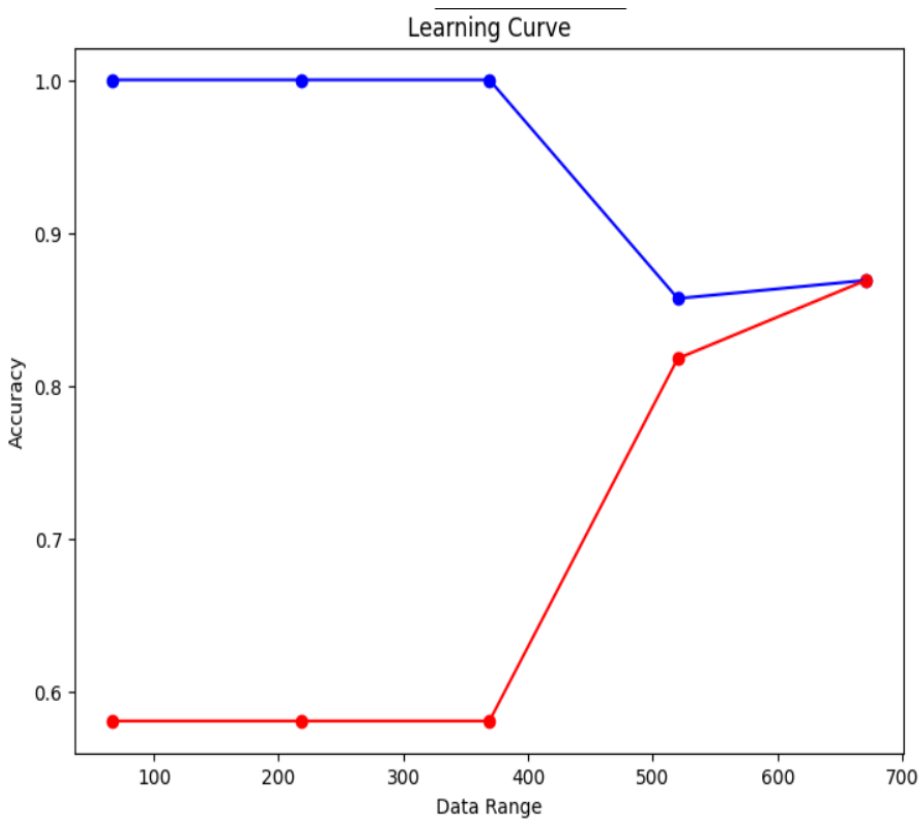
First we going to split the dataset into the x and y for training and testing

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=42)
```

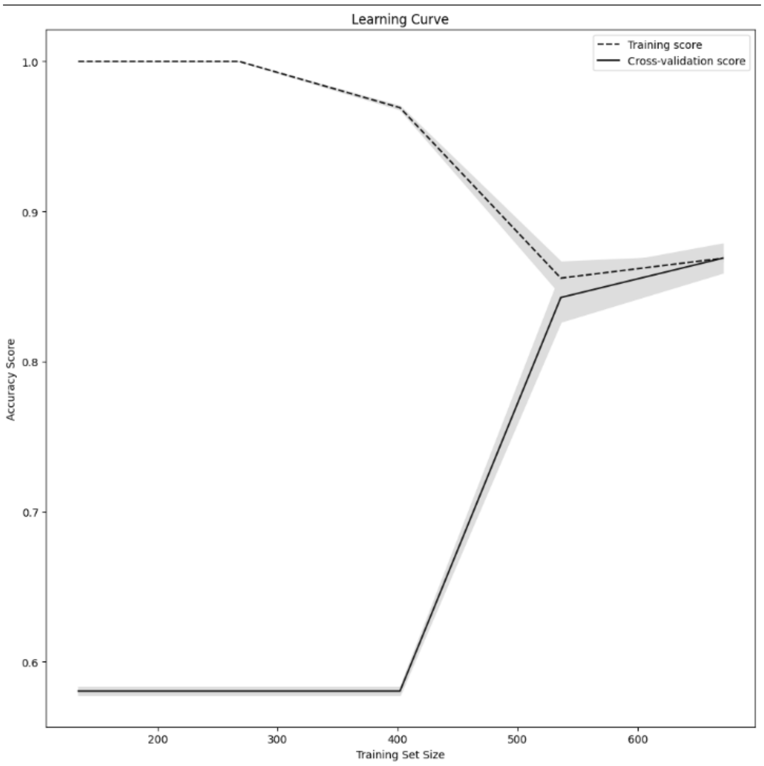
The fist mode we using is Decision Tree Classifier
Which give us

```
Accuracy on training set: 0.869
Accuracy on test set: 0.869
ACC of model: 0.8810
```

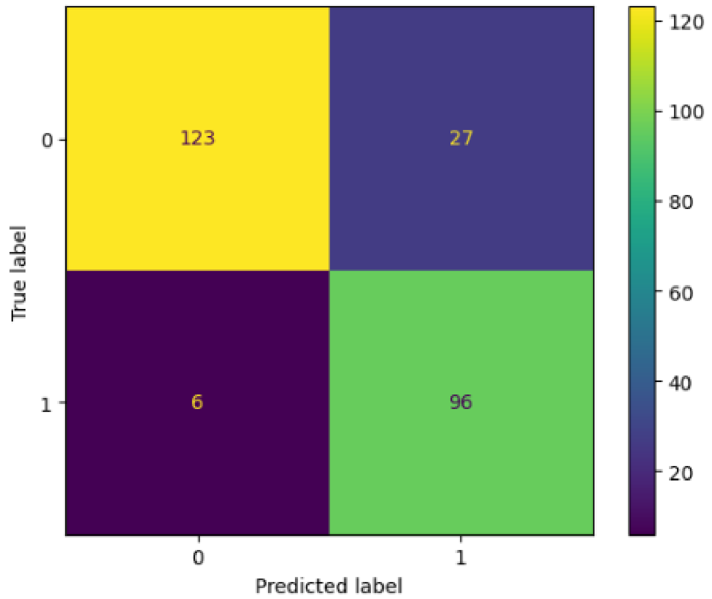
Learn carves visualizations



Another learning curve



Confusion matrix :



From the confusion matrix we can say that disruption between 123 and 96 is normal

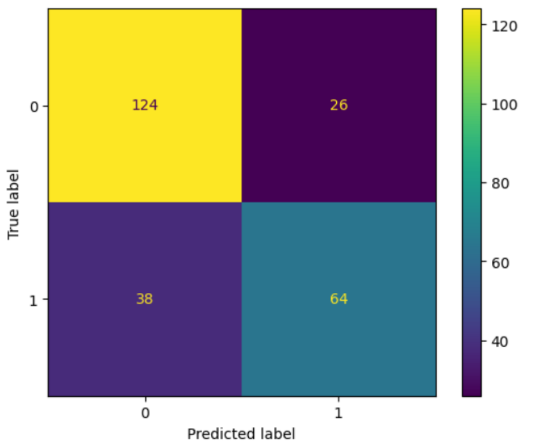
Classification report

```
Train - Accuracy : 0.868824531516184
Train - Confusion matrix : [[279  58]
 [ 19 231]]
Train - classification report :               precision    recall  f1-score   support

           0       0.94      0.83      0.88       337
           1       0.80      0.92      0.86       250

    accuracy                           0.87       587
   macro avg       0.87      0.88      0.87       587
weighted avg       0.88      0.87      0.87       587

Test - Accuracy : 0.8690476190476191
Test - Confusion matrix : [[123  27]
 [  6  96]]
Test - classification report :               precision    recall  f1-score   support

           0       0.95      0.82      0.88       150
           1       0.78      0.94      0.85       102

    accuracy                           0.87       252
   macro avg       0.87      0.88      0.87       252
weighted avg       0.88      0.87      0.87       252
```

Second model we going to use SVM (support vector machine)

We will be using the polynomial as our kernel
which give us

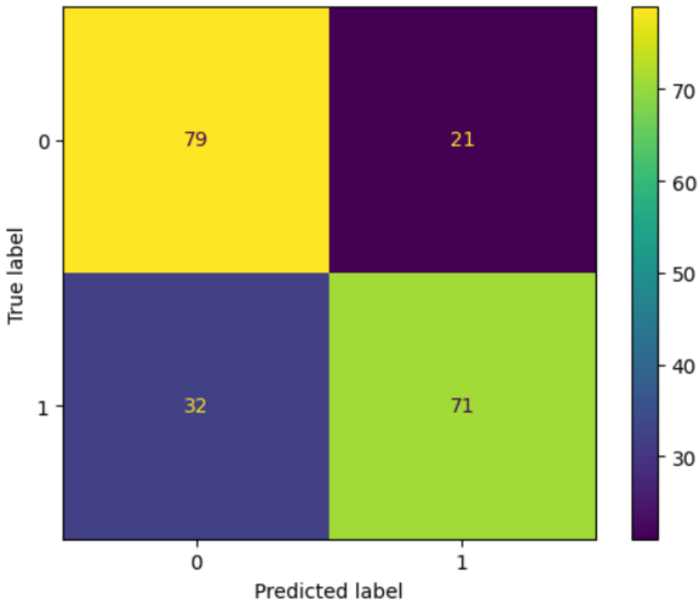
```
Accuracy (Polynomial Kernel):  74.60
Accuracy on training set: 0.746
Accuracy on test set: 0.746
F1 (Polynomial Kernel):  74.30
```



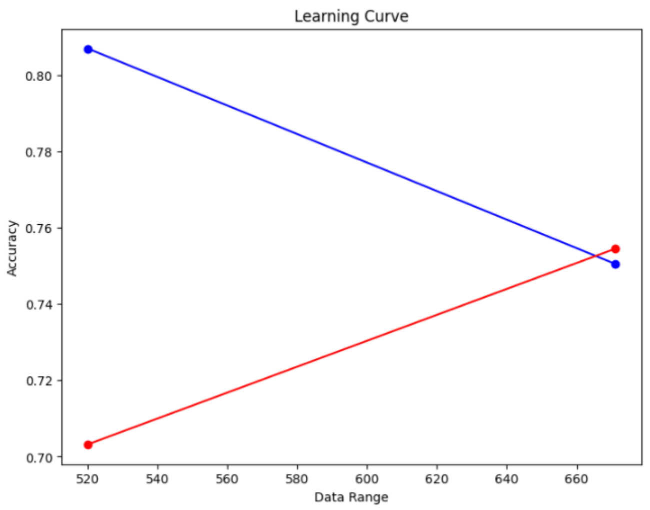
But because of the confusing matrix we can see that there is big difference between 124 and the 64 so we need to fix this problem by using over sampling which will give us

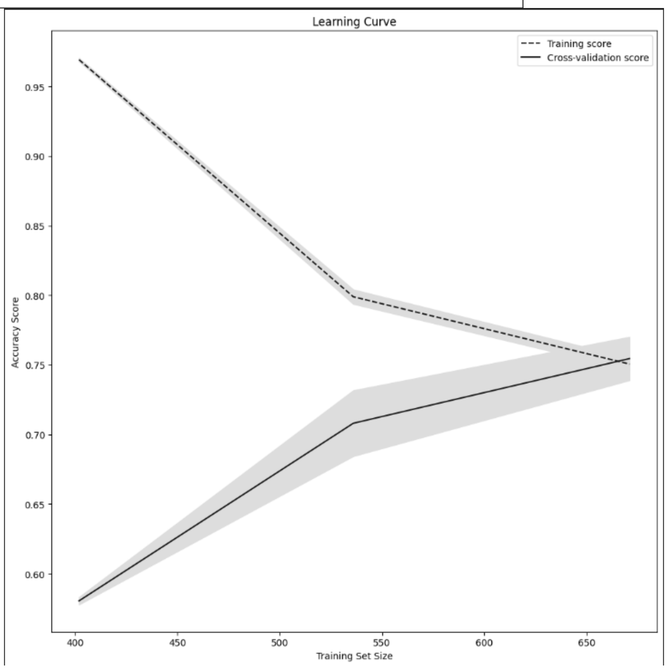
```
Accuracy (Polynomial Kernel):  73.89
Accuracy on training set: 0.753
Accuracy on test set: 0.742
F1 (Polynomial Kernel):  73.84
```

And fixed the confusion matrix



Learning carve visualisation





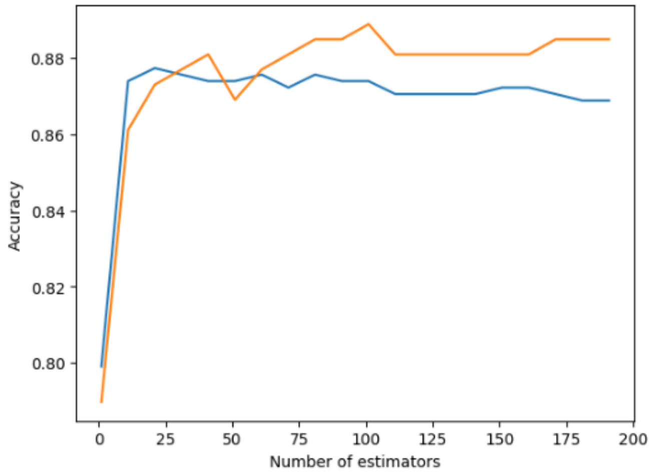
Classification report

```
Train - Accuracy : 0.7515923566878981
Train - Confusion matrix : [[182  55]
 [ 62 172]]
Train - classification report :               precision    recall  f1-score   support

           0       0.75      0.77      0.76       237
           1       0.76      0.74      0.75       234

    accuracy                           0.75       471
   macro avg       0.75      0.75      0.75       471
weighted avg       0.75      0.75      0.75       471

Test - Accuracy : 0.7389162561576355
Test - Confusion matrix : [[79 21]
 [32 71]]
Test - classification report :               precision    recall  f1-score   support

           0       0.71      0.79      0.75       100
           1       0.77      0.69      0.73       103

    accuracy                           0.74       203
   macro avg       0.74      0.74      0.74       203
weighted avg       0.74      0.74      0.74       203
```

Third model random forest

Fist we going to see the best n for number of estimators

Which give us

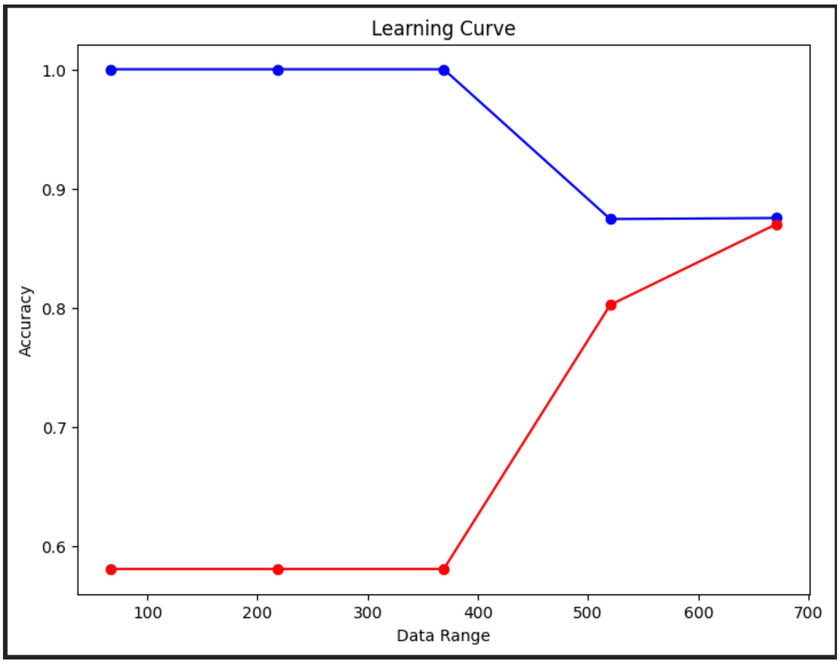


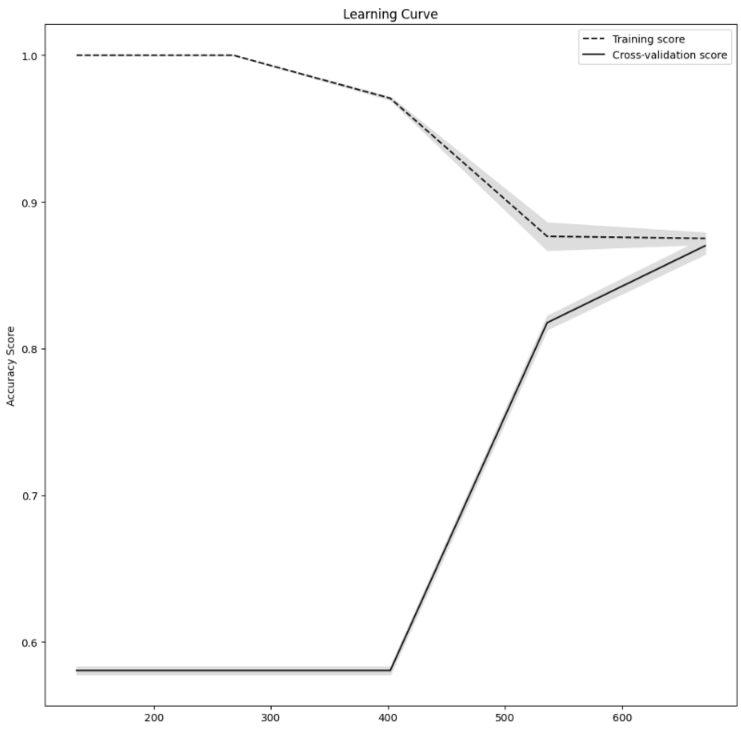
From we visualization we can estimate that best one near n =170

Which will give us

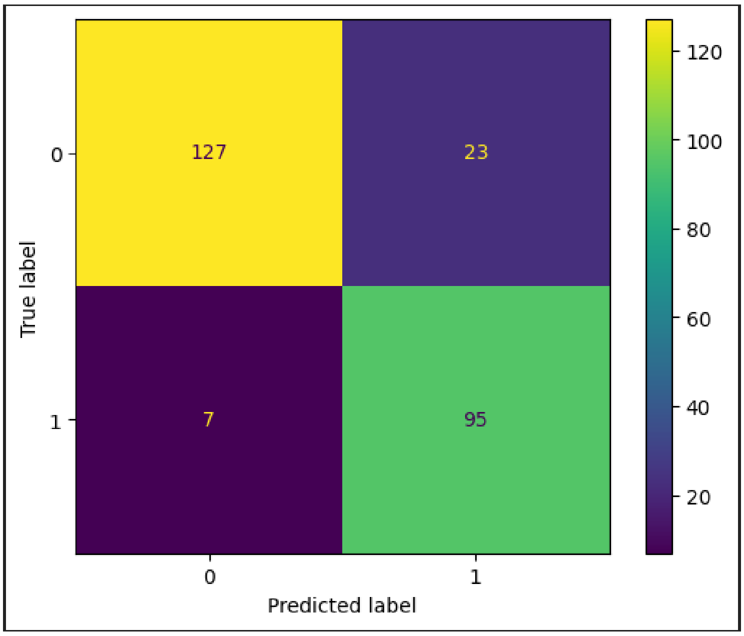
```
Accuracy: 0.8809523809523809
Accuracy on training set: 0.918
Accuracy on test set: 0.881
```

Learning curves





Confusion matrix



This confusion matrix have good balance

Classification report

```
Train - Accuracy : 0.868824531516184
Train - Confusion matrix : [[285  52]
 [ 25 225]]
Train - classification report :               precision    recall  f1-score   support

           0       0.92      0.85      0.88       337
           1       0.81      0.90      0.85       250

    accuracy                           0.87       587
   macro avg       0.87      0.87      0.87       587
weighted avg       0.87      0.87      0.87       587

Test - Accuracy : 0.8849206349206349
Test - Confusion matrix : [[128  22]
 [  7  95]]
Test - classification report :               precision    recall  f1-score   support

           0       0.95      0.85      0.90       150
           1       0.81      0.93      0.87       102

    accuracy                           0.88       252
   macro avg       0.88      0.89      0.88       252
weighted avg       0.89      0.88      0.89       252
```

From the past models we going to be making ensemble models

Firs we going to be using voting

First making Hard voting

By add the three models we used before we get in comparison to them

```
DecisionTreeClassifier 0.8690476190476191
SVC 0.746031746031746
RandomForestClassifier 0.8849206349206349
VotingClassifier 0.8849206349206349
```

We can see that random and voting have the same that because it choses from the three modes the best and use it

Classification report

```
Train - Accuracy : 0.8671209540034072
Train - Confusion matrix : [[285  52]
 [ 26 224]]
Train - classification report :               precision    recall  f1-score   support

           0       0.92      0.85      0.88       337
           1       0.81      0.90      0.85       250

    accuracy                           0.87       587
   macro avg       0.86      0.87      0.87       587
weighted avg       0.87      0.87      0.87       587

Test - Accuracy : 0.8849206349206349
Test - Confusion matrix : [[128  22]
 [  7  95]]
Test - classification report :               precision    recall  f1-score   support

           0       0.95      0.85      0.90       150
           1       0.81      0.93      0.87       102

    accuracy                           0.88       252
   macro avg       0.88      0.89      0.88       252
weighted avg       0.89      0.88      0.89       252
```

Second soft voting

Which give us

```
DecisionTreeClassifier 0.8690476190476191
SVC 0.746031746031746
RandomForestClassifier 0.8849206349206349
VotingClassifier 0.8809523809523809
```

Classification report

```
Train - Accuracy : 0.868824531516184
Train - Confusion matrix : [[285  52]
 [ 25 225]]
Train - classification report :               precision    recall  f1-score   support

           0       0.92      0.85      0.88       337
           1       0.81      0.90      0.85       250

    accuracy                           0.87       587
   macro avg       0.87      0.87      0.87       587
weighted avg       0.87      0.87      0.87       587

Test - Accuracy : 0.8809523809523809
Test - Confusion matrix : [[127  23]
 [  7  95]]
Test - classification report :               precision    recall  f1-score   support

           0       0.95      0.85      0.89       150
           1       0.81      0.93      0.86       102

    accuracy                           0.88       252
   macro avg       0.88      0.89      0.88       252
weighted avg       0.89      0.88      0.88       252
```

Bagging was done for Decision Tree

```
Descision Tree (Bagging) =  0.8849206349206349
Descision Tree (Stand Alone) =  0.8690476190476191
```

Which slightly improved on the model

Classification report

```
Train - Accuracy : 0.8705281090289608
Train - Confusion matrix : [[280  57]
 [ 19 231]]
Train - classification report :               precision    recall  f1-score   support

           0       0.92      0.85      0.88       337
           1       0.81      0.90      0.85       250

    accuracy                           0.87       587
   macro avg       0.87      0.87      0.87       587
weighted avg       0.87      0.87      0.87       587

Test - Accuracy : 0.8849206349206349
Test - Confusion matrix : [[127  23]
 [  6  96]]
Test - classification report :               precision    recall  f1-score   support

           0       0.95      0.85      0.90       150
           1       0.81      0.94      0.87       102

    accuracy                           0.88       252
   macro avg       0.88      0.89      0.88       252
weighted avg       0.89      0.88      0.89       252
```

Bagging was done for Random Forest

```
Random Forest (Bagging) =  0.873015873015873
Random Forest (Stand Alone) =  0.8809523809523809
```

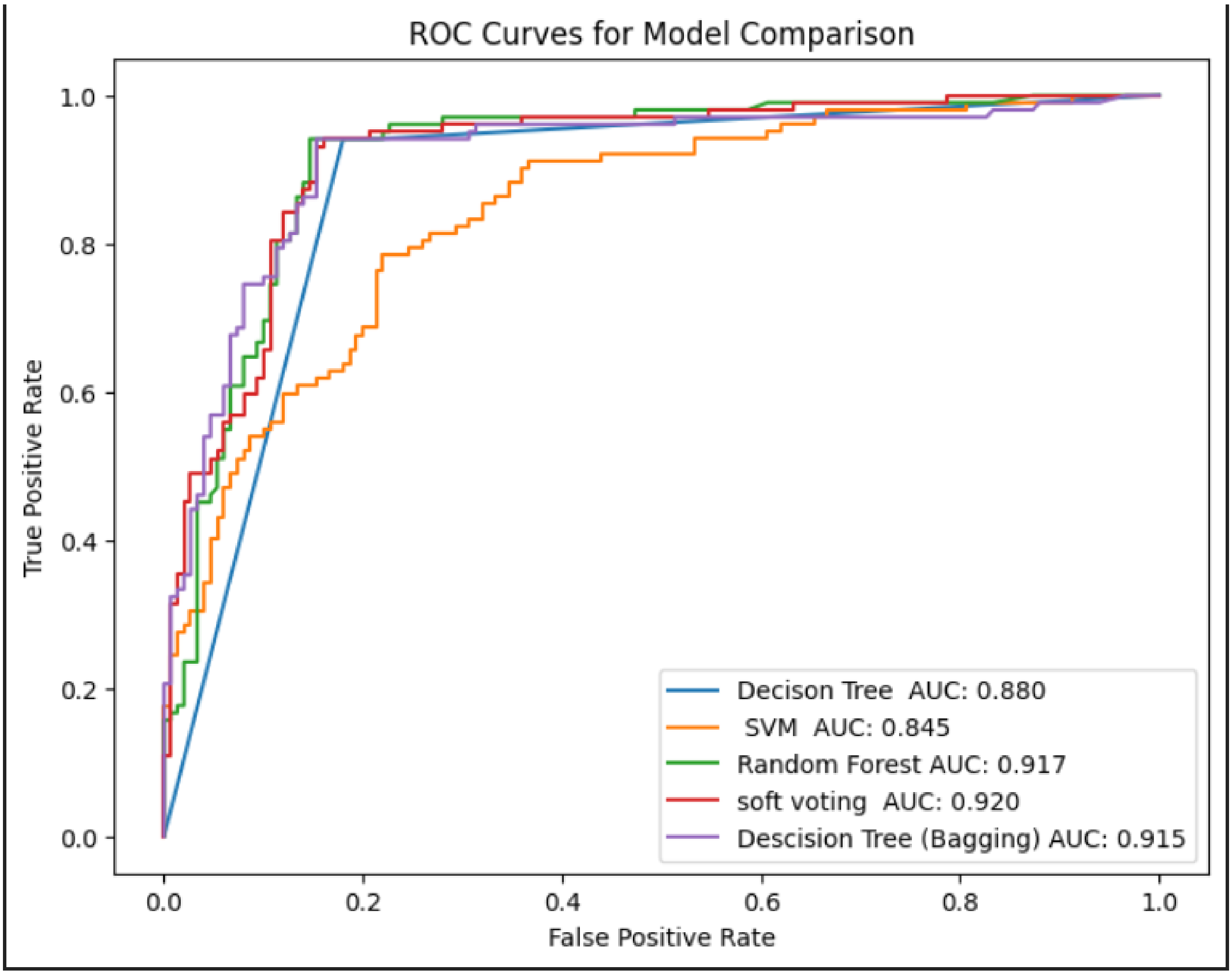
This time didn't improve on it

Classification report

```
Train - Accuracy : 0.8705281090289608
Train - Confusion matrix : [[280  57]
 [ 19 231]]
Train - classification report :               precision    recall  f1-score   support

           0       0.94      0.83      0.88       337
           1       0.80      0.92      0.86       250

    accuracy                           0.87       587
   macro avg       0.87      0.88      0.87       587
weighted avg       0.88      0.87      0.87       587

Test - Accuracy : 0.8849206349206349
Test - Confusion matrix : [[127  23]
 [  6  96]]
Test - classification report :               precision    recall  f1-score   support

           0       0.95      0.85      0.90       150
           1       0.81      0.94      0.87       102

    accuracy                           0.88       252
   macro avg       0.88      0.89      0.88       252
weighted avg       0.89      0.88      0.89       252
```

finally using the roc curve to compare between models

note : hard voting does not have roc curve



Form out mode we can see that the one that have highest AUC is soft voting but random forest and Decision tree bagging is very close to it

Ahmed Sameh 211392

Images of many defective steel plates with surface flaws are used to extract steel plate defects. 27 distinct features were identified through image analysis to characterize the steel fault. Six distinct types of defects are classified, with a final category called "other faults" reserved for errors that do not fall into any of the other six categories.

Data processing

```
faulty.isnull().sum() # find the sum of the nulls in the df
```

```
X_Minimum                0
X_Maximum                0
Y_Minimum                0
Y_Maximum                0
Pixels_Areas             0
X_Perimeter              0
Y_Perimeter              0
Sum_of_Luminosity        0
Minimum_of_Luminosity    0
Maximum_of_Luminosity    0
Length_of_Conveyer       0
TypeOfSteel_A300         0
TypeOfSteel_A400         0
Steel_Plate_Thickness    0
Edges_Index              0
Empty_Index              0
Square_Index             0
Outside_X_Index          0
Edges_X_Index            0
Edges_Y_Index            0
Outside_Global_Index     0
LogOfAreas               0
Log_X_Index              0
Log_Y_Index              0
Orientation_Index        0
Luminosity_Index         0
SigmoidOfAreas           0
Pastry                   0
Z_Scratch                0
K_Scatch                 0
Stains                   0
Dirtiness                0
Bumps                    0
Other_Faults             0
dtype: int64
```

in the figure above I checked for the nulls in the data frame and there was none so I did not remove any.

```
faulty['All_faults'] = faulty[['Pastry', 'Z_Scratch', 'K_Scatch', 'Stains', 'Dirtiness', 'Bumps', 'Other_Faults']].idxmax(axis=1)
#creating a new column called 'All_faults' in the faulty DataFrame, where each row is labeled with the type of fault
```

In the figure above I created a column called ‘All_faults’ where I put different class in different columns in it.

```
faulty = faulty.drop(['Pastry', 'Z_Scratch', 'K_Scatch', 'Stains', 'Dirtiness', 'Bumps', 'Other_Faults'], axis=1)
# deleting the columns after adding them to the all faults column
```

In the figure above I dropped the columns because I already added them in the new column

```
faulty['All_faults'].replace('Pastry',1,inplace=True)
faulty['All_faults'].replace('Z_Scratch',2,inplace=True)
faulty['All_faults'].replace('K_Scatch',3,inplace=True)
faulty['All_faults'].replace('Stains',4,inplace=True)
faulty['All_faults'].replace('Dirtiness',5,inplace=True)
faulty['All_faults'].replace('Bumps',6,inplace=True)
faulty['All_faults'].replace('Other_Faults',7,inplace=True)
faulty.head()  # replacing the strings with ints to be able to work with it
```

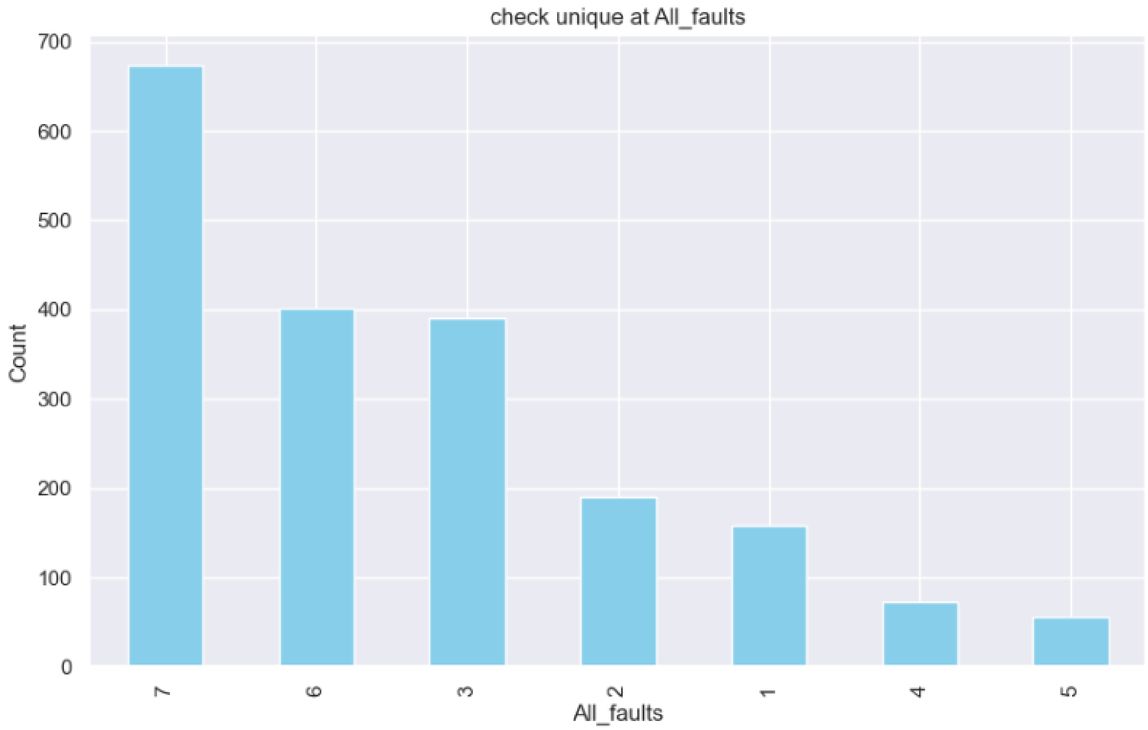
In the figure above I changed the string data to int to be able to work with numerical data

```
faulty.dtypes # get the datatypes
```

```
X_Minimum                  int64
X_Maximum                  int64
Y_Minimum                  int64
Y_Maximum                  int64
Pixels_Areas               int64
X_Perimeter                int64
Y_Perimeter                int64
Sum_of_Luminosity          int64
Minimum_of_Luminosity      int64
Maximum_of_Luminosity      int64
Length_of_Conveyer         int64
TypeOfSteel_A300           int64
TypeOfSteel_A400           int64
Steel_Plate_Thickness      int64
Edges_Index              float64
Empty_Index              float64
Square_Index             float64
Outside_X_Index          float64
Edges_X_Index            float64
Edges_Y_Index            float64
Outside_Global_Index     float64
LogOfAreas               float64
Log_X_Index              float64
Log_Y_Index              float64
Orientation_Index        float64
Luminosity_Index         float64
SigmoidOfAreas           float64
All_faults                 int64
dtype: object
```

In the figure above I checked for datatypes after the change

```python
column_name = 'All_faults'
value_counts = faulty[column_name].value_counts()
plt.figure(figsize=(10, 6))
value_counts.plot(kind='bar', color='skyblue')
plt.title(f'check unique at {column_name}')
plt.xlabel(column_name)
plt.ylabel('Count')
plt.show()
#check the number of each unqiue class at all faults column
```



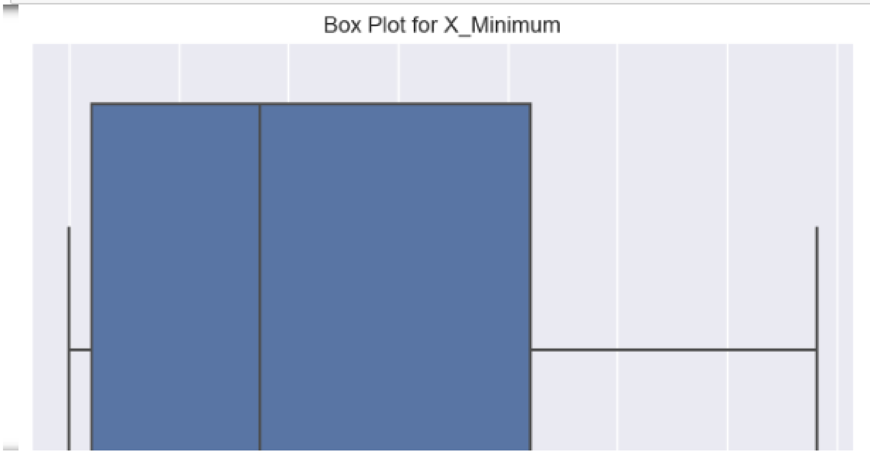
In the figure above the unique value of each class in the new columns because it will be the target

```python
faulty.duplicated().sum() #check for duplicates
```

```
0
```

In the figure above check for duplicated and there was none to remove

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



In the figure above I checked for outliers on the boxplot and there were many.

```
import matplotlib.pyplot as plt
from sklearn.neighbors import LocalOutlierFactor

xz = faulty.iloc[:, :-1]
yz = faulty['All_faults']

outlierDetector2 =  LocalOutlierFactor(n_neighbors=40)
result2=outlierDetector2.fit_predict(xz)
# split the df into xz(anything except target) and yz(target)uses the Local Outlier Factor algorithm (outlierDetector2)
#to predict whether each data point is an inlier or an outlier, storing the results in the result2 variable
```

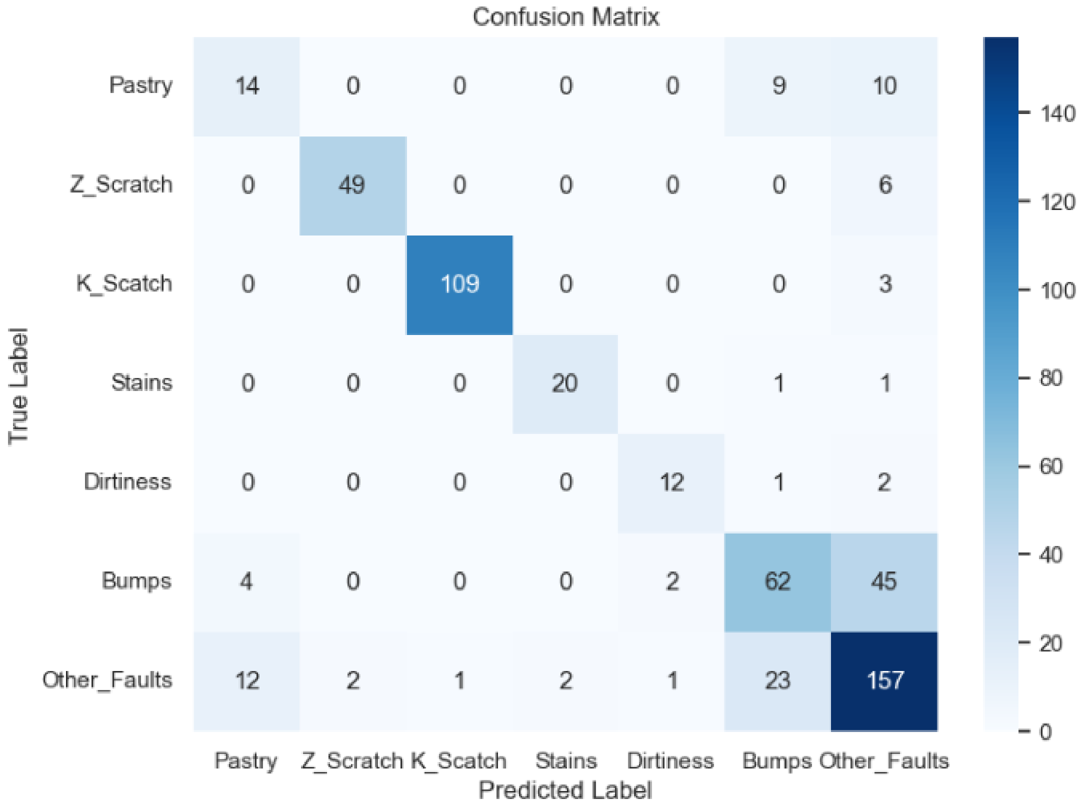
Used the local outlier factor to see whether the data point is outlier or not and saving it in the result2.

Rf after outlier before smote:

```
rf = RandomForestClassifier(n_estimators=100, max_depth = 20, min_samples_split=3, random_state=42)
rf.fit(x_train, y_train)

# Prediction on the test set
y_predef = rf.predict(x_test)

# Evaluate the model
acc = accuracy_score(y_predef, y_test)
print(f"Accuracy of the model: {acc:.4f} after outlier removal")
#using the rf model with hyper p after the outlier removal
```

```
Accuracy of the model: 0.7719 after outlier removal
```

Used the random forest model after the outlier removal with random hyper parameter after many trials and this was the best I got



The classification report above looks good as there was a lot of true positive between the classes.

Difference in classification report

The ratio of accurately predicted positive observations to the total number of predicted positives is known as precision. It calculates the proportion of actual positive cases that match the predictions.

The ratio of accurately predicted positive observations to all observations made during the actual class is known as recall. It counts the number of real positive cases that were accurately anticipated.

The harmonic mean of recall and precision is known as the F1-score. It offers a balance between recall and precision.

The number of real instances of the class in the given dataset is known as support. It shows how many actual instances there are of each class.

Classification Report:

	precision	recall	f1-score	support
1	0.47	0.42	0.44	33
2	0.96	0.89	0.92	55
3	0.99	0.97	0.98	112
4	0.91	0.91	0.91	22
5	0.80	0.80	0.80	15
6	0.65	0.55	0.59	113
7	0.70	0.79	0.74	198
accuracy			0.77	548
macro avg	0.78	0.76	0.77	548
weighted avg	0.77	0.77	0.77	548

In the report above it is seen that in class 3 has the best between all that means it was the most in true positives on the other hand class one has the least after it class 6 the rest were good and ok

Oversample:

oversample

```
oversample = SMOTE(random_state=42)
overX,overY=oversample.fit_resample(x_train, y_train)
#using oversampling on the training data
```

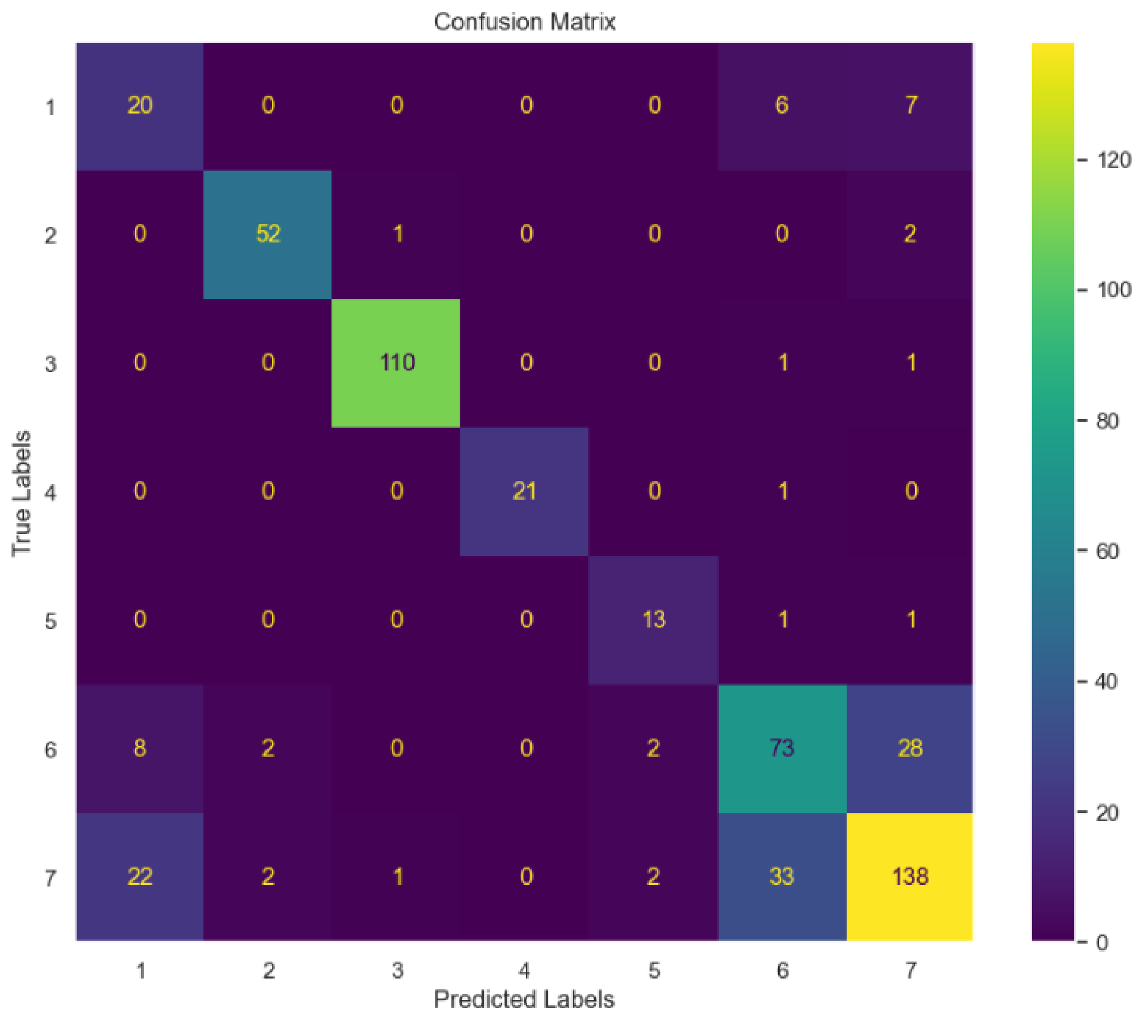
Oversample using smote on the training set and the test stays the same as original.

Rf after oversample

```
scaler = StandardScaler()
overX_scaled = scaler.fit_transform(overX)
x_test_scaled = scaler.transform(x_test)


rfo_clf = RandomForestClassifier(n_estimators=100, max_depth=40, min_samples_split=3, random_state=42)
rfo_clf.fit(overX_scaled, overY)

# Make predictions on the standardized testing data
opy_pred = rfo_clf.predict(x_test_scaled)
```

```
ACC of model: 0.7792 After SMOTE detection
```

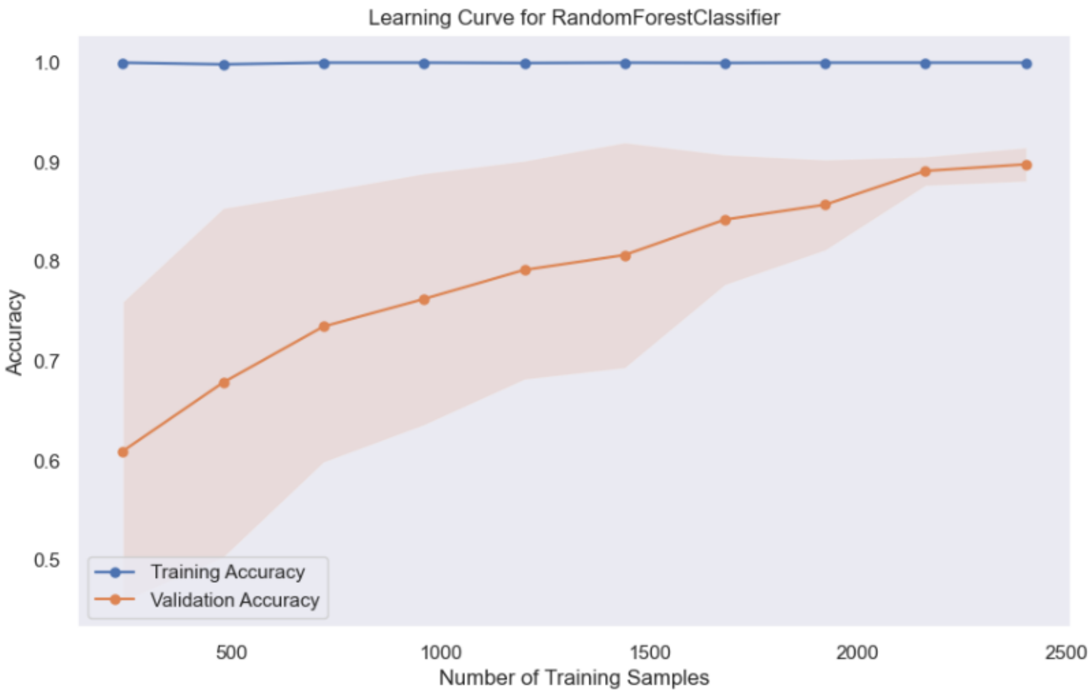
Using random forest on the new training sets using the standard scalar to standarise the features prior to Random Forest classifier training. This prevents any problems with feature scaling and guarantees that the features contribute to the model in a balanced way, aiding in the Random Forest algorithm's best performance. And it is used on input variables not output that’s why I used overx and testx.



The cm above looks alright however compared to the one before the outlier it has done better in all classes except the class 7 where it had less true positives

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

           1       0.40      0.61      0.48        33
           2       0.93      0.95      0.94        55
           3       0.98      0.98      0.98       112
           4       1.00      0.95      0.98        22
           5       0.76      0.87      0.81        15
           6       0.63      0.65      0.64       113
           7       0.78      0.70      0.74       198

    accuracy                           0.78       548
   macro avg       0.78      0.81      0.80       548
weighted avg       0.79      0.78      0.78       548
```

In class 3 it is constantly high until now but in class 4 there was a better change than the one before in the whole report where again one was the lowest of all.



Learning curve looks alright but it reached a point much better than the accuracy recorded maybe depends on the number of cv mentioned.

Explaining the MSE, bias, variance

Bias quantifies the average deviation between the expected and actual numbers. High bias models often oversimplify the relationships that underlie the data, which can result in systematic inaccuracies. When a model is too basic to adequately represent the complexity of the data, it can lead to underfitting, or high bias.

Variance quantifies how sensitive or variable the model is to the training set of data. A high variance model may perform well on the training set but poorly on fresh, untested data since it is heavily influenced by the details of the training set. When a model overfits, it collects noise in the training data instead of the underlying patterns, which is caused by high variation.

Low variance and high bias frequently result in underfitting.

A large variance with little bias frequently results in overfitting.

The average squared difference between the actual and predicted values is known as the MSE.

```
MSE: 3.177919708029197
Bias: 2.4828193430656933
Variance: 0.6951003649635037
```

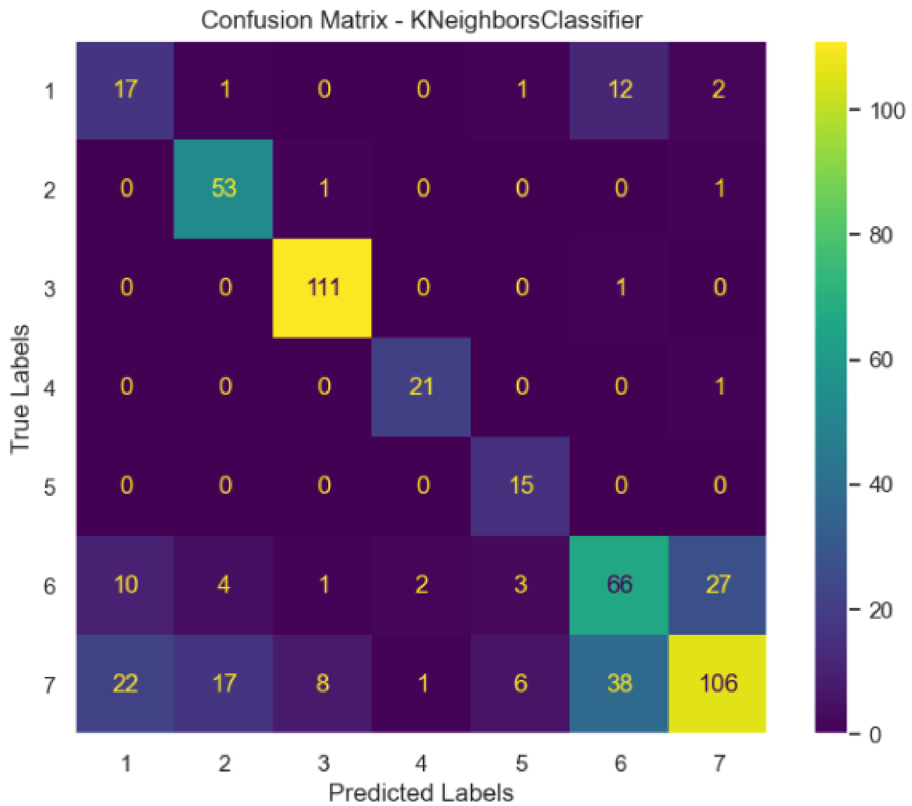
Based on the above it might mean it is underfitting, but the values are not that bad .

KNN after oversample

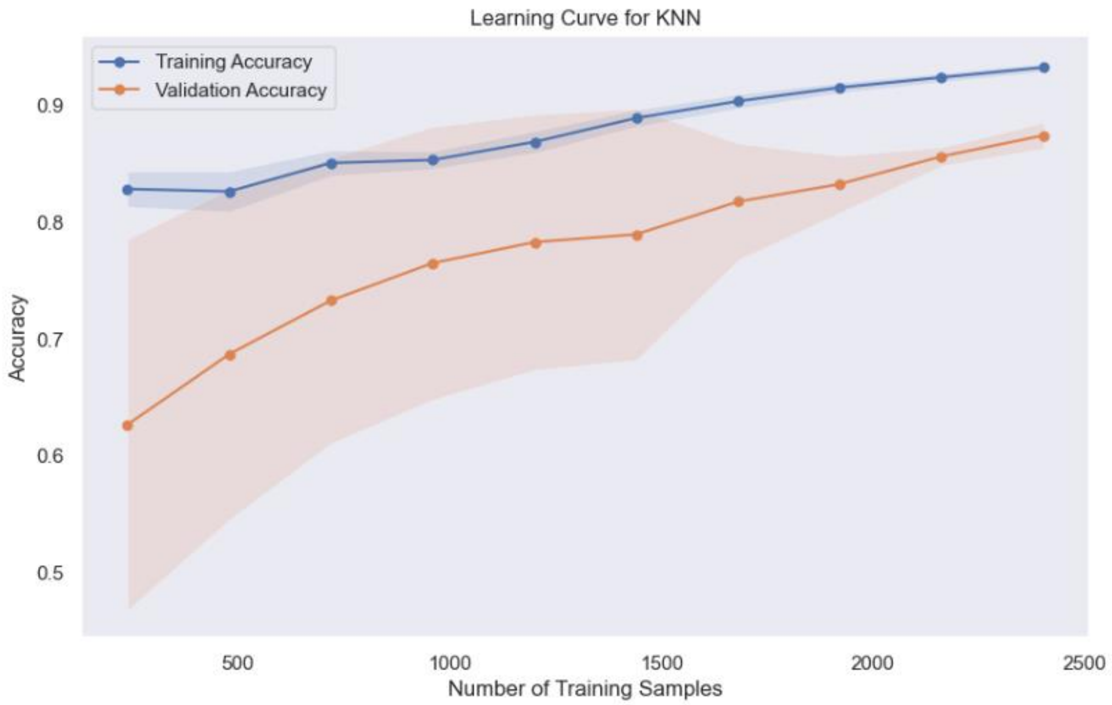
```
from sklearn.neighbors import KNeighborsClassifier

scaler = StandardScaler()
overX_scaled = scaler.fit_transform(overX)
x_test_scaled = scaler.transform(x_test)

knn0_clf = KNeighborsClassifier(n_neighbors=3, metric='manhattan')
knn0_clf.fit(overX_scaled, overY)

# Make predictions on training and test data
oy_train_pred23 = knn0_clf.predict(overX_scaled)
oy_test_pred23 = knn0_clf.predict(x_test_scaled)


train_accuracy = accuracy_score(overY, oy_train_pred23)
test_accuracy = accuracy_score(y_test, oy_test_pred23)

print("KNeighborsClassifier Results:")
print("Training Accuracy: {:.2f}".format(train_accuracy))
print("Test Accuracy: {:.2f}".format(test_accuracy))

# standardize the features, and train a KNN classifier on the oversampled training data,
```

```
KNeighborsClassifier Results:
Training Accuracy: 0.94
Test Accuracy: 0.71
```

Using KNN on the new training sets using the standard scalar to standarise the features prior to Random Forest classifier training. This prevents any problems with feature scaling and guarantees that the features contribute to the model in a balanced way, aiding in the KNN's algorithm's best performance. And it is used on input variables not output that's why I used overx and testx.



The cm above is much worse than the models before, however in some of the classes it performs better.



The learning curve above looks amazing where they are close to each other

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

           1       0.35      0.52      0.41        33
           2       0.71      0.96      0.82        55
           3       0.92      0.99      0.95       112
           4       0.88      0.95      0.91        22
           5       0.60      1.00      0.75        15
           6       0.56      0.58      0.57       113
           7       0.77      0.54      0.63       198

    accuracy                           0.71       548
   macro avg       0.68      0.79      0.72       548
weighted avg       0.73      0.71      0.71       548
```

The case is repeated where class 3 is in the top and 1 in the bottom but the most surprising on is class 5 with a low precision but full recall 0.60 means that when the model predicts a positive class, it is correct about 60% of the time. However, the recall is full.

```
MSE: 4.143339416058395
Bias: 2.776181569343066
Variance: 1.3671578467153285
```

influenced by the details of the training set. When a model overfits, it collects noise in the training data instead of the underlying patterns, which is caused by high variation.

Low variance and high bias frequently result in underfitting.

A large variance with little bias frequently results in overfitting.

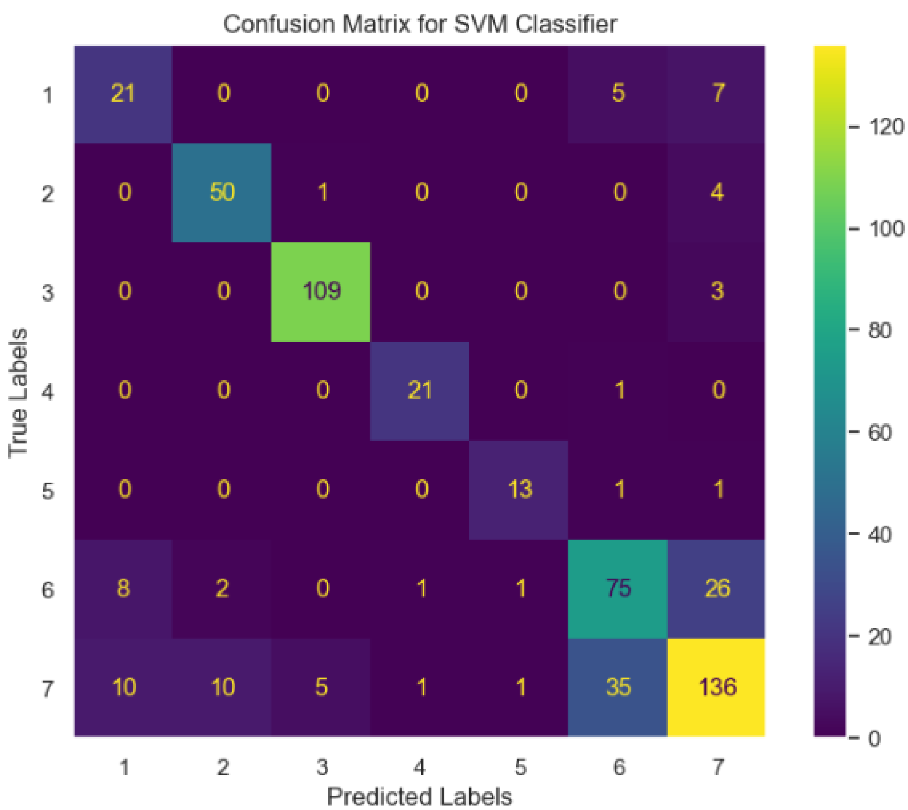
The average squared difference between the actual and predicted values is known as the MSE.

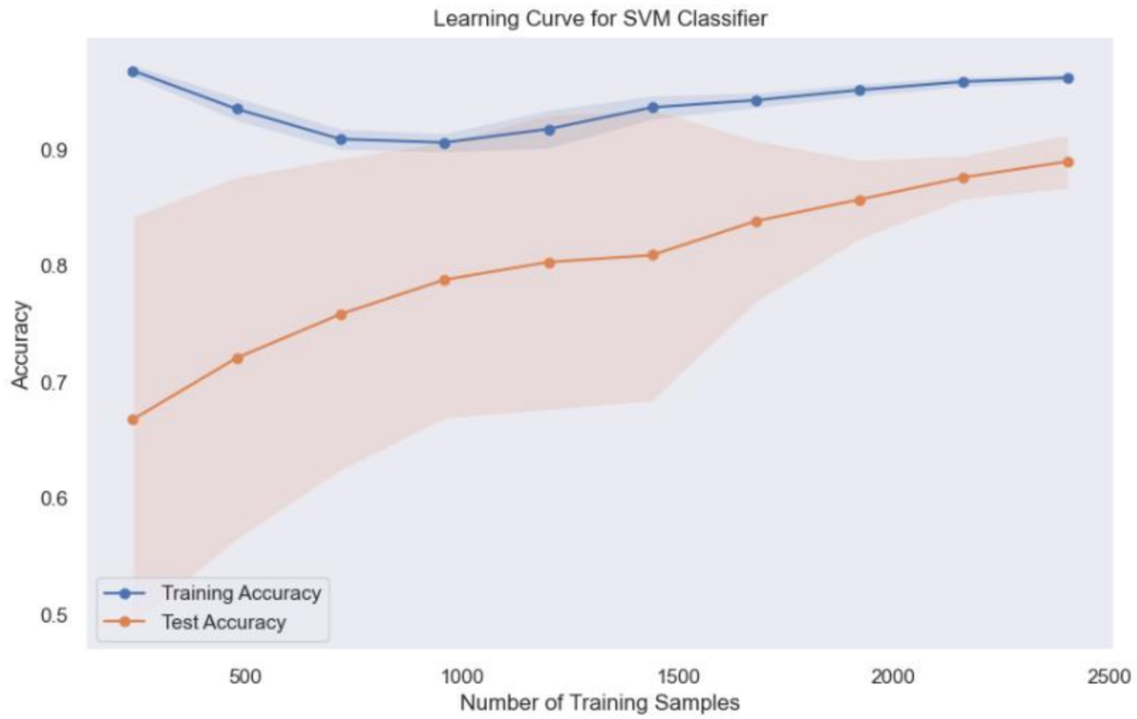
SVM after oversample

```
scaler = StandardScaler()
overX_scaled = scaler.fit_transform(overX)
x_test_scaled = scaler.transform(x_test)
svm_clf = SVC(C= 10, random_state=42, probability=True)


svm_clf.fit(overX_scaled, overY)


predictions_svm = svm_clf.predict(x_test_scaled)


accuracy_svm = accuracy_score(y_test, predictions_svm)

print(f"SVM Accuracy: {accuracy_svm}")
# standardize the features, and train a SVM classifier on the oversampled training data,
```

```
SVM Accuracy: 0.7755474452554745
```

Using SVM on the new training sets using the standard scalar to standarise the features prior to Random Forest classifier training. This prevents any problems with feature scaling and guarantees that the features contribute to the model in a balanced way, aiding in the SVM algorithm's best performance. And it is used on input variables not output that’s why I used overx and testx.





This learning curve looks good where the testing and training points are beside each other .

Classification Report:

	precision	recall	f1-score	support
1	0.54	0.64	0.58	33
2	0.81	0.91	0.85	55
3	0.95	0.97	0.96	112
4	0.91	0.95	0.93	22
5	0.87	0.87	0.87	15
6	0.64	0.66	0.65	113
7	0.77	0.69	0.73	198
accuracy			0.78	548
macro avg	0.78	0.81	0.80	548
weighted avg	0.78	0.78	0.77	548

class 3 tops in all nearly and class one is the least in all.

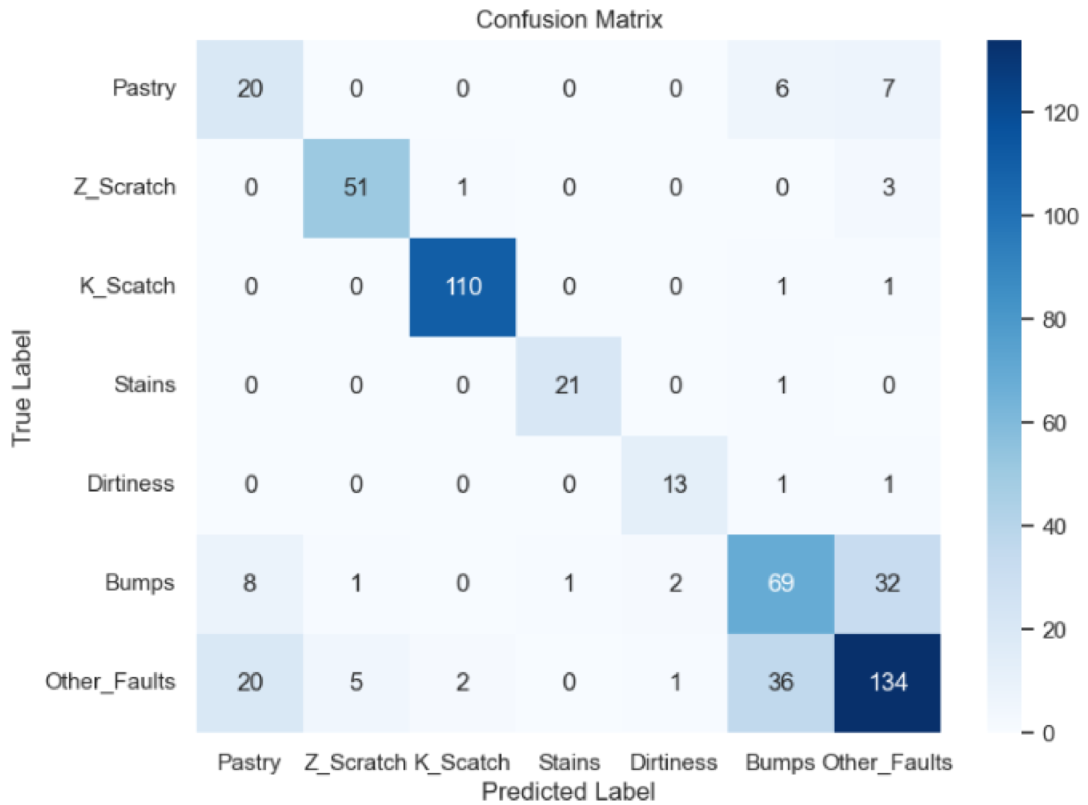
```
MSE: 3.3278284671532843
Bias: 2.2661633211678835
Variance: 1.0616651459854014
```

Pipeline

```
Pipe1 = Pipeline([('Scaler', StandardScaler()), ('model', RandomForestClassifier(n_estimators=100, random_state=42))])

Pipe1.fit(overX, overY)

yf_pred = Pipe1.predict(x_test)

print("Accuracy of the model: %.4f with Pipeline" % accuracy_score(yf_pred, y_test))
```

Tried pipeline of the random forest with different hyper p however lower accuracy than the past.

```
Accuracy of the model: 0.7628 with Pipeline
```



Ensemble voting:

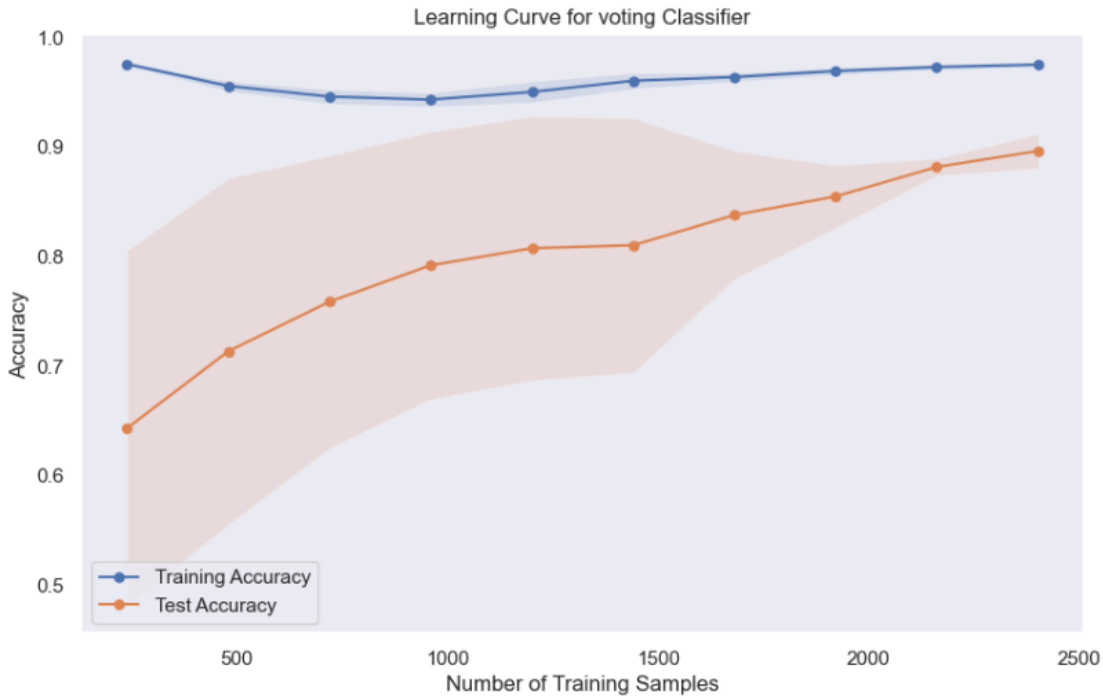
```python
# Create a voting classifier
voting_clf = VotingClassifier(
    estimators=[
        ('svm', svm_clf),
        ('random_forest', rfo_clf),
        ('knn', knn0_clf)
    ],
    voting='hard'
)


voting_clf.fit(overX_scaled, overY)


predictions_voting = voting_clf.predict(x_test_scaled)

accuracy_voting = accuracy_score(y_test, predictions_voting)
print(f"Voting Classifier Accuracy: {accuracy_voting}")
```

Used hard ensemble voting where The class that received the greatest majority of votes that is, the class that each classifier predicted with the highest probability is the anticipated output class.

```
Voting Classifier Accuracy: 0.7609489051094891
```

Lower than the other models

```
Classification Report for Voting Classifier:
              precision    recall  f1-score   support

           1       0.44      0.70      0.54        33
           2       0.77      0.96      0.85        55
           3       0.95      0.99      0.97       112
           4       0.91      0.95      0.93        22
           5       0.81      0.87      0.84        15
           6       0.63      0.65      0.64       113
           7       0.79      0.62      0.70       198

    accuracy                           0.76       548
   macro avg       0.76      0.82      0.78       548
weighted avg       0.77      0.76      0.76       548
```



This learning curve looks good where the testing and training points are beside each other.

```
MSE: 3.4479927007299267
Bias: 2.4024635036496353
Variance: 1.045529197080292
```

Based on the above it might mean it is underfitting, but the values are not that bad as the variance is low so the model may be stable

Accuracy:

rf after outlier before smote	0.771
rf after smote	0.779
Pipeline rf	0.76
Knn after smote	0.70
Svm after smote	0.775
Voting ensemble	0.760

The rf after smote had the best accuracy .

Predict students' dropout and academic success. Classification Dataset {Sara Amjad 212071}

1. Data Description:

This dataset originates from a research initiative addressing the challenge of mitigating academic dropout and failure in higher education. The overarching objective is to leverage machine learning methodologies to detect students at risk early in their academic journey, enabling the implementation of timely support strategies. The dataset encompasses comprehensive information available at the commencement of student enrolment, covering academic trajectory, demographics, and socio-economic factors. The classification task revolves around predicting students' outcomes at the conclusion of the standard course duration, categorizing them into three classes: dropout, enrolled, and graduate. Derived from multiple disjoint databases of a higher education institution, the dataset spans various undergraduate disciplines, such as agronomy, design, education, nursing, journalism, management, social service, and technologies. Given the class imbalance, particularly towards one category, the dataset serves as a foundation for constructing classification models to anticipate students' dropout and academic success. I picked this dataset to expand my learning after working with a two-class dataset in a previous assignment. Wanting to tackle a more complex challenge, I opted for a multiclass classification task. The dataset's focus on academic success and dropout caught my interest as a student, offering a real-world context for applying machine learning.

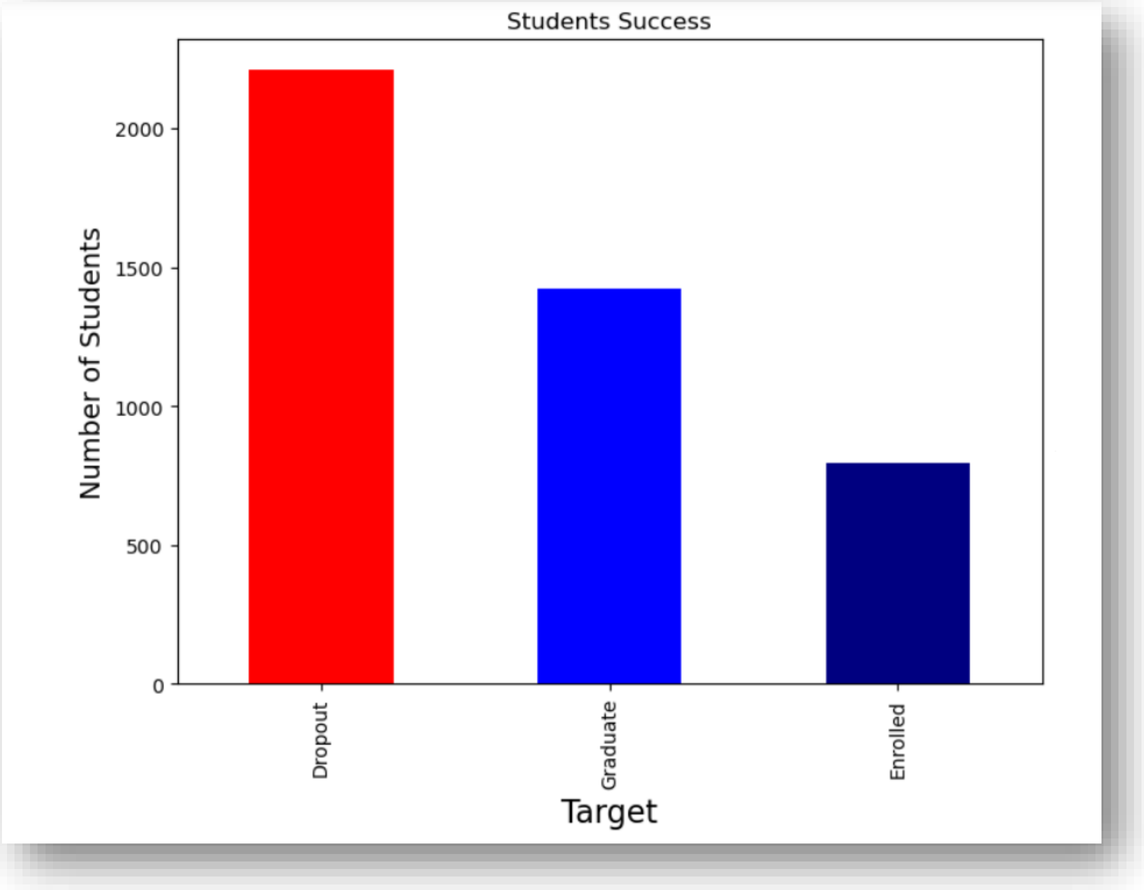


Figure 1

As you can see its slightly imbalanced as stated in the description.

2. Data Pre-Processing:

- Nulls: there were no null values as you can see below in figure 2.

```
StudentDF.isna().sum()

Marital status                                    0
Application mode                                  0
Application order                                 0
Course                                            0
Daytime/evening attendance\t                      0
Previous qualification                            0
Previous qualification (grade)                    0
Nationality                                       0
Mother's qualification                            0
Father's qualification                            0
Mother's occupation                               0
Father's occupation                               0
Admission grade                                   0
Displaced                                         0
Educational special needs                         0
Debtor                                            0
Tuition fees up to date                           0
Gender                                            0
Scholarship holder                                0
Age at enrollment                                 0
International                                     0
Curricular units 1st sem (credited)               0
Curricular units 1st sem (enrolled)               0
Curricular units 1st sem (evaluations)            0
Curricular units 1st sem (approved)               0
Curricular units 1st sem (grade)                  0
Curricular units 1st sem (without evaluations)    0
Curricular units 2nd sem (credited)               0
Curricular units 2nd sem (enrolled)               0
Curricular units 2nd sem (evaluations)            0
Curricular units 2nd sem (approved)               0
Curricular units 2nd sem (grade)                  0
Curricular units 2nd sem (without evaluations)    0
Unemployment rate                                 0
Inflation rate                                    0
GDP                                               0
Target                                            0
dtype: int64
```

Figure 2

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

- ## Duplicates:

As shown below in figure 3 there were no duplicates

```
StudentDF.duplicated().sum() #check if i have duplicates
: 0
```

- ## Spelling Mistake

In ‘Nationality’ Column at the beginning it was written wrong so I corrected it.

```
#i renamed the nacionality coloumn to correct spelling
StudentDF.rename(columns={'Nacionality': 'Nationality'}, inplace=True)
```

- ## Outliers:

I used 3 methods of detecting the outliers and each method gave me different outcomes so after using the 3 methods. I checked first the percentage of the data that were detected if they were acceptable or not since removing a lot of data may be causing a loss of important data causing a loss of important insights. Moreover, the models might become biased and fail to generalize well to new, unseen data.as well as it could negatively impact the model’s performance. So, the first method which was the “Elliptic Envelope” detected 885 outliers. The second method was the ‘LocalOutlierFactor’ detected 240 outliers and lastly the ‘IsolationForest’ detected 590 outliers. So, after comparing the performance of the used models using each outlier detector method I chose the last method which was the ‘IsolationForest’.in addition to that I checked again the classes frequencies after using any of the methods to make sure it did not affect one class in a way that made it disappear due to its treatment as an outlier itself.

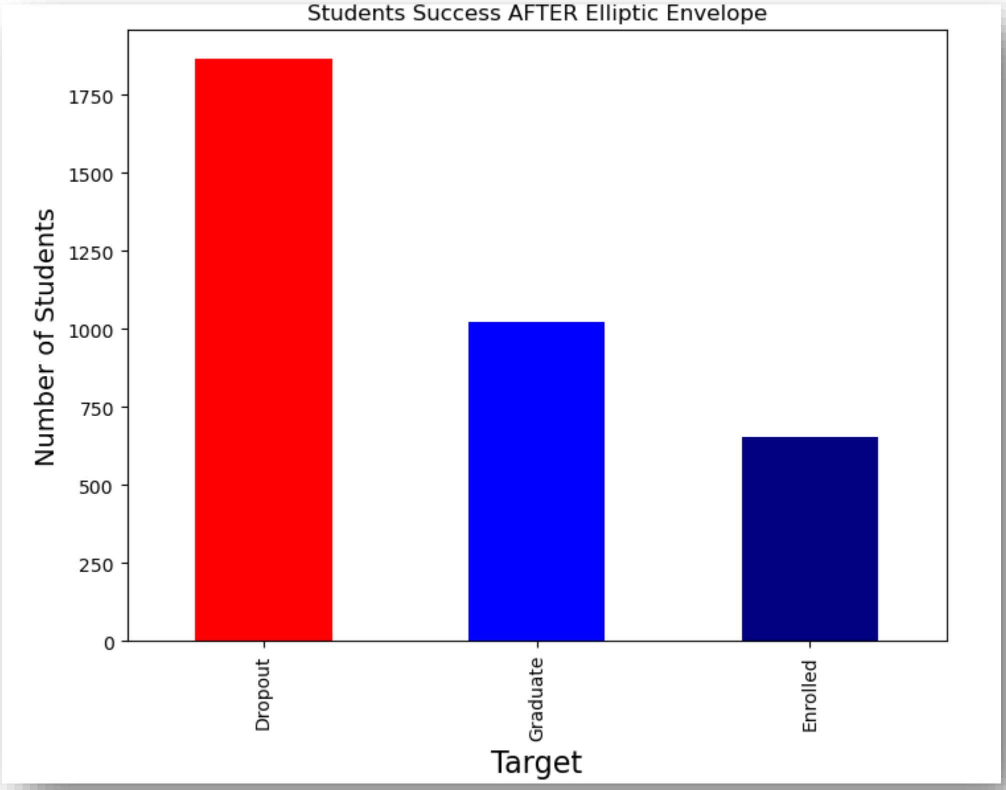


Figure 5 (after removing outliers using the Elliptic Envelope).

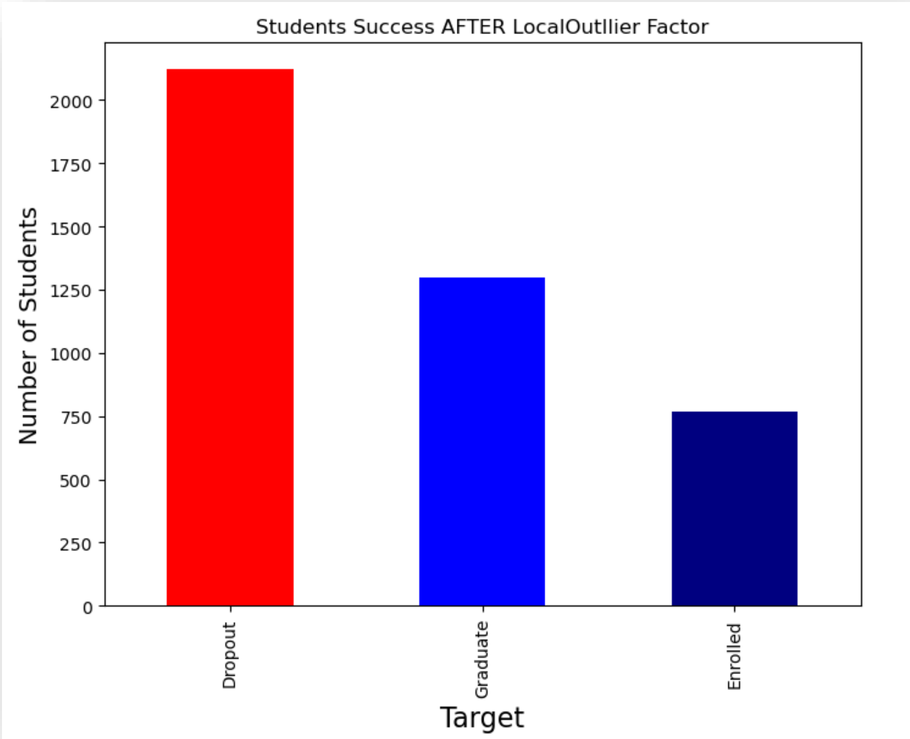


Figure 6 (after removing outliers using the Local Outlier Factor).

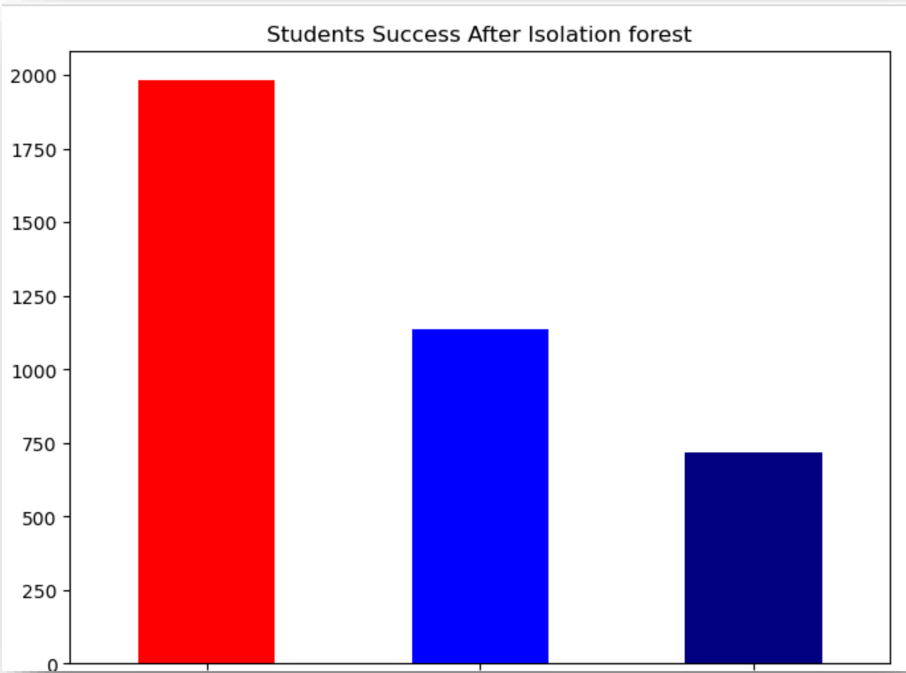
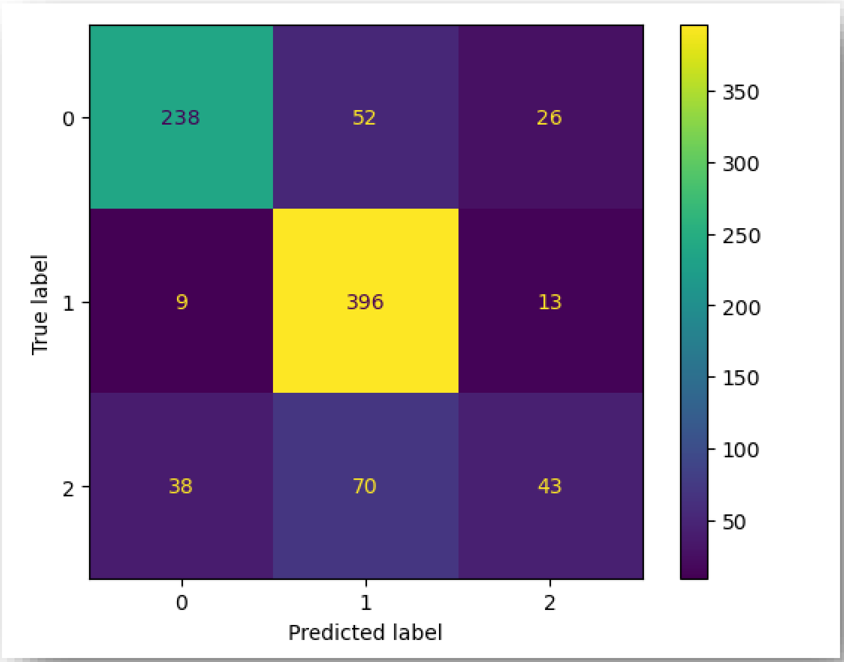


Figure 7 (after removing outliers using the Isolation Forest).

After removing the outliers, the models' accuracies and confusion matrix improved. So for a comparison iam going to show the accuracies, Confusion matrix and classification report.

```
Best Max Depth: 10
Best Number of Estimators: 100
Accuracy on training set before removing the outliers: 0.919
Accuracy on test set before removing the outliers: 0.765
```

In the Figure above is the Random Forest model accuracies before removing the outliers.



In the Figure above is the Random Forest model Confusion Matrix before removing the outliers.

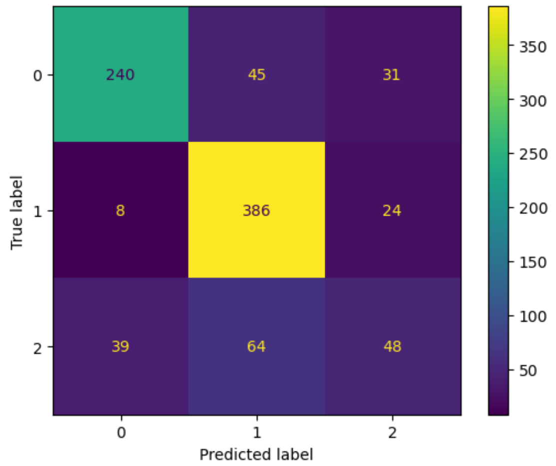
```
Training Accuracy: 0.92
Test Accuracy: 0.76
Classification Report for Training Data:
              precision    recall  f1-score   support

           0       0.98      0.90      0.94      1105
           1       0.88      1.00      0.93      1791
           2       0.97      0.73      0.83       643

    accuracy                           0.92      3539
   macro avg       0.94      0.88      0.90      3539
weighted avg       0.93      0.92      0.92      3539

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

           0       0.84      0.75      0.79       316
           1       0.76      0.95      0.85       418
           2       0.52      0.28      0.37       151

    accuracy                           0.76       885
   macro avg       0.71      0.66      0.67       885
weighted avg       0.75      0.76      0.75       885
```

In the Figure above is the Random Forest model Classification Report before removing the outliers.

```
Accuracy on training set before removing outliers: 0.843
Accuracy on test set before removing outliers: 0.762
```

In the Figure above is the Gradient Boosting model accuracies before removing the outliers.

```python
# this is just the confusion matrix of the gradient boosting before removing any outliers or
#oversampling which shows why the class 2 is not predicted correctly that good due to that its
#slightly imbalanced
Confusion = confusion_matrix(y_test, y_pred25)
CM=ConfusionMatrixDisplay(Confusion)
CM.plot()
plt.show()
```



In the Figure above is the Gradient Boosting Confusion Matrix before removing the outliers.

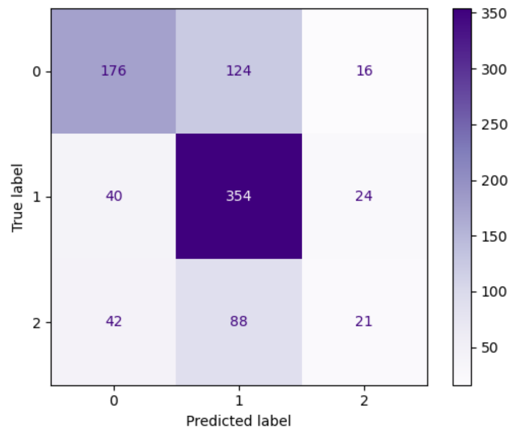
```python
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

# Define a list of potential k values to try
k_values = range(1, 20)

# Initialize variables to store the best k and its corresponding accuracy
best_k = None
best_accuracy = 0.0

# it Iterates through each k value and find the one with the highest accuracy
for k in k_values:
    # i Created a K-Nearest Neighbors classifier with the current k value
    knn = KNeighborsClassifier(n_neighbors=k)
    # i Used cross-validation to estimate the model's accuracy
    accuracy = cross_val_score(knn, x_train, y_train, cv=5).mean()
    # Check if the current k gives a higher accuracy than the previous best
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k

# Train the final K-Nearest Neighbors model using the best k
best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(x_train, y_train)
# Make predictions on the test set
y_pred = best_knn.predict(x_test)

print("Best k:", best_k)
print("Accuracy on knn training set before removing the outliers: {:.3f}".format(best_knn.score(x_train, y_train)))
print("Accuracy on knn test set before removing the outliers: {:.3f}".format(best_knn.score(x_test, y_test)))
```

```
Best k: 8
Accuracy on knn training set before removing the outliers: 0.701
Accuracy on knn test set before removing the outliers: 0.623
```

In the Figure above is the KNN accuracies before removing the outliers.

```
#plot the confusion matrix of the best knn model that i got previously
plot_confusion_matrix(best_knn,x_test,y_test,cmap='Purples')
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2b0824a2c10>
```

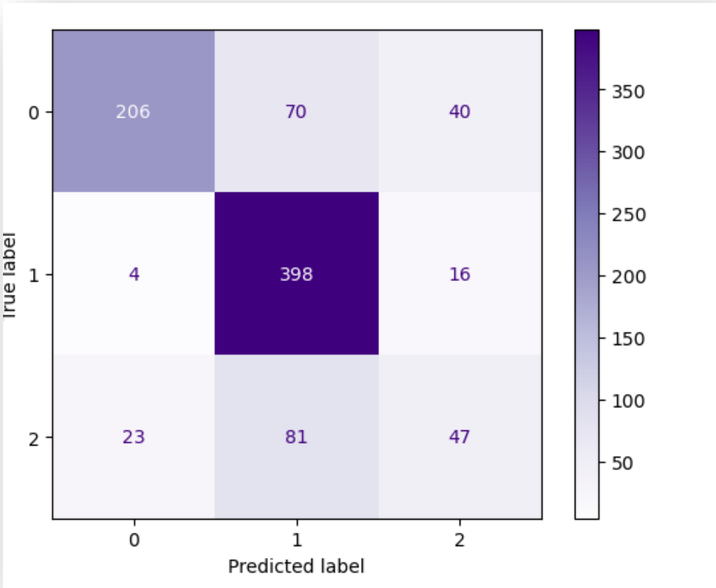


In the Figure above is the KNN Confusion matrix before removing the outliers.

```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

# Define a list of potential max_depth and min_samples_leaf values
max_depth_values = range(1, 20)
min_samples_leaf_values = range(1, 10)

# Initialize variables to store the best max_depth, min_samples_leaf, and accuracy
best_max_depth = None
best_min_samples_leaf = None
best_accuracy = 0.0

# Iterate through combinations of max_depth and min_samples_leaf and find the best ones
for max_depth in max_depth_values:
    for min_samples_leaf in min_samples_leaf_values:
        # Create a Decision Tree classifier with the current parameters
        dt = DecisionTreeClassifier(max_depth=max_depth, min_samples_leaf=min_samples_leaf)
        # Use cross-validation to estimate the model's accuracy
        accuracy = cross_val_score(dt, x_train, y_train, cv=5).mean()
         # Check if the current combination of parameters gives a higher accuracy
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_max_depth = max_depth
            best_min_samples_leaf = min_samples_leaf

# Train the final model with the best parameters
best_dt = DecisionTreeClassifier(max_depth=best_max_depth, min_samples_leaf=best_min_samples_leaf)
best_dt.fit(x_train, y_train)
# Make predictions on the test set
y_pred = best_dt.predict(x_test)

# Print the results
print("Best max_depth:", best_max_depth)
print("Best min_samples_leaf:", best_min_samples_leaf)
print("Accuracy on training set before removing the outliers: {:.3f}".format(best_dt.score(x_train, y_train)))
print("Accuracy on test set before removing the outliers: {:.3f}".format(best_dt.score(x_test, y_test)))
```

```
Best max_depth: 5
Best min_samples_leaf: 8
Accuracy on training set before removing the outliers: 0.773
Accuracy on test set before removing the outliers: 0.736
```

In the Figure above is the Decision tree accuracies before removing the outliers.



In the Figure above is the Decision tree confusion matrix before removing the outliers.

```
# Assign the best_forest model obtained from random search to the variable 'rf'
rf = best_forest
# Train the Random Forest model on the filtered training data after removing outliers
rf.fit(x_train, y_train)
# Make predictions on the test set
y_pred=rf.predict(x_test)
# Print the accuracy of the model on the training set after removing outliers
print("Accuracy on training set after removing the outliers : {:.3f}".format(rf.score(x_train, y_train)))
# Print the accuracy of the model on the test set after removing outliers
print("Accuracy on test set after removing the outliers: {:.3f}".format(rf.score(x_test, y_test)))
```

```
Accuracy on training set after removing the outliers : 0.925
Accuracy on test set after removing the outliers: 0.772
```

In the Figure above is the Random Forest model accuracies after removing the outliers.

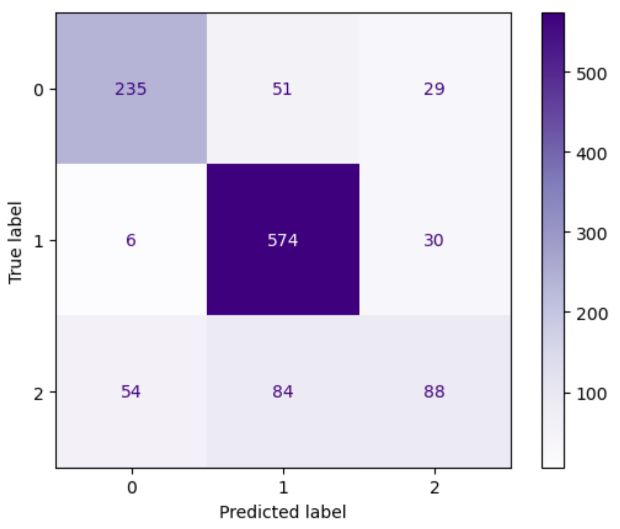
```
# train the gradient boosting classifier model on the filtered training data
gr = grad
# Make predictions on the test set
gr.fit(x_train, y_train)
y_pred=rf.predict(x_test)
# print the accuracies
print("Accuracy on training set after removing the outliers : {:.3f}".format(gr.score(x_train, y_train)))
print("Accuracy on test set after removing the outliers: {:.3f}".format(gr.score(x_test, y_test)))
```

```
Accuracy on training set after removing the outliers : 0.846
Accuracy on test set after removing the outliers: 0.779
```

In the Figure above is the Gradient Boosting model accuracies after removing the outliers.

```
#plot the confusion matrix of gradient boosting after removing the outliers
plot_confusion_matrix(gr,x_test,y_test,cmap='Purples')
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2b0824c3b20>
```

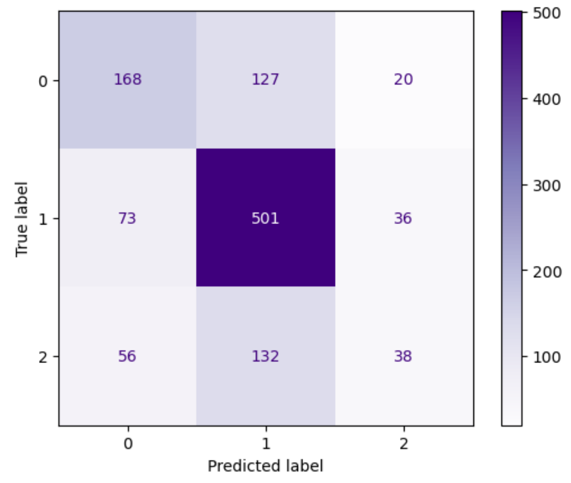


In the Figure above is the Gradient Boosting confusion matrix after removing the outliers.

```
# train the KNN classifier model on the filtered training data
knn=best_knn
knn.fit(x_train, y_train)
# Make predictions on the test set
y_pred=knn.predict(x_test)
# print the accuracies
print("Accuracy on training set after removing the outliers : {:.3f}".format(knn.score(x_train, y_train)))
print("Accuracy on test set after removing the outliers: {:.3f}".format(knn.score(x_test, y_test)))

Accuracy on training set after removing the outliers : 0.687
Accuracy on test set after removing the outliers: 0.614
```

In the Figure above is the KNN accuracies after removing the outliers.

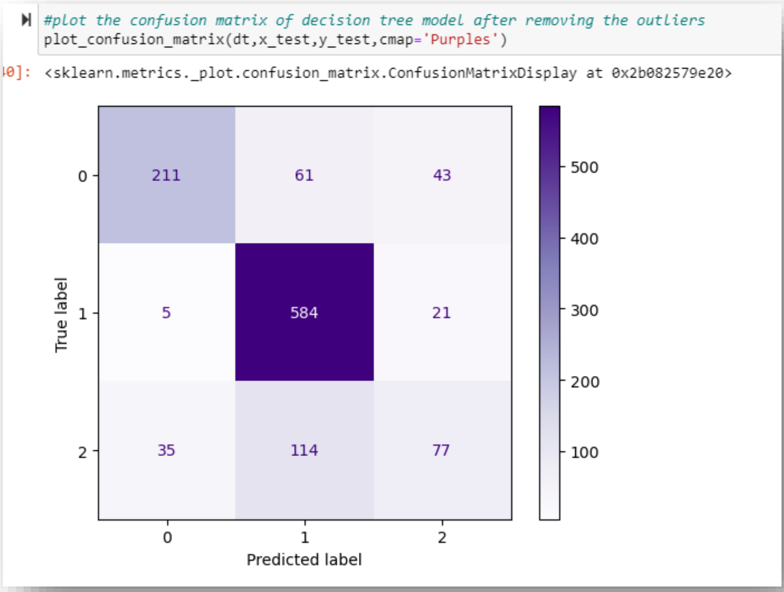
```
#plot the confusion matrix of knn after removing the outliers
plot_confusion_matrix(knn,x_test,y_test,cmap='Purples')

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2b0826a7460>
```



In the Figure above is the KNN confusion matrix after removing the outliers.

```
In [39]:
# train the decision tree  model on the filtered training data
dt= best_dt
dt.fit(x_train, y_train)
# Make predictions on the test set
y_pred=dt.predict(x_test)
# print accuracies
print("Accuracy on training set after removing the outliers : {:.3f}".format(dt.score(x_train, y_train)))
print("Accuracy on test set after removing the outliers: {:.3f}".format(dt.score(x_test, y_test)))

Accuracy on training set after removing the outliers : 0.775
Accuracy on test set after removing the outliers: 0.758
```

In the Figure above is the decision tree accuracies after removing the outliers.



In the Figure above is the decision tree confusion matrix after removing the outliers.

As shown above in all the figures after removing the outliers the random forest model accuracies improved from training 0.919 and testing 0.765 to training 0.925 and testing 0.772.Moreover, the gradient boosting model accuracies improved from training 0.843 and testing 0.762 to training 0.846 and testing 0.779.In addition to that the Decision tree model improved from training 0.773 and testing 0.736 to training 0.775 and testing 0.758.However, the Knn model decreased slightly from training 0.701 and testing 0.623 to training 0.687 and testing 0.614.

- Represent Categorical variables as numerical labels:

Through replacing the categorical classes by numbers as shown below in Figure 7.

```python
# i Replaced 'Dropout' with 0 in the 'Target' column
StudentDF['Target'].replace('Dropout',0,inplace=True)
# i Replaced 'Graduate' with 1 in the 'Target' column
StudentDF['Target'].replace('Graduate',1,inplace=True)
# i Replaced 'Enrolled' with 2 in the 'Target' column
StudentDF['Target'].replace('Enrolled',2,inplace=True)
StudentDF.head()
```

3. Oversampling methods due to the imbalanced classes

In my attempt to balance the dataset, I first tried using Random Oversampling. However, after checking how well the models were doing, it was clear that the expected improvement in accuracies didn't happen as much as I thought. This could be because the dataset already had a fair balance between classes, and adding more instances of the minority class through Random Oversampling didn't make a big difference. Realizing this, I decided to give the SMOTE method a shot. Surprisingly, models that didn't do so well with Random Oversampling showed better results with SMOTE. Still, despite these efforts, the increase in accuracies after oversampling wasn't very big. It seems that the nature of the dataset and the specific models I used have a big impact on how effective oversampling techniques can be.

```python
# Assign the best_forest model obtained from random search to the variable 'rf'
rf = best_forest
# Train the Random Forest model on the filtered training data after removing outliers
rf.fit(x_train, y_train)
# Make predictions on the test set
y_pred=rf.predict(x_test)
# Print the accuracy of the model on the training set after removing outliers
print("Accuracy on training set after removing the outliers : {:.3f}".format(rf.score(x_train, y_train)))
# Print the accuracy of the model on the test set after removing outliers
print("Accuracy on test set after removing the outliers: {:.3f}".format(rf.score(x_test, y_test)))
```

```
Accuracy on training set after removing the outliers : 0.925
Accuracy on test set after removing the outliers: 0.772
```

In the Figure above is the Random Forest model accuracies after removing the outliers but before oversampling.

```
# Training the model on the oversampled data.
rfr= rf
rfr.fit(x_train, y_train)
# Fit the classifier to the training data

# Predicting on the test set.
y_pred=rfr.predict(x_test)
print("Accuracy on training set after random Oversampling : {:.3f}".format(rfr.score(x_train, y_train)))
print("ACC of random Forest model testing set: %.4f After random Oversampling" %accuracy_score(y_pred,y_test))
```

```
Accuracy on training set after random Oversampling : 0.925
ACC of random Forest model testing set: 0.7724 After random Oversampling
```

In the Figure above is the Random Forest model accuracies after removing the outliers and after oversampling using Random Oversampling.

```
# Using StandardScaler to scale the features
scaler = StandardScaler()
# Splitting the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(overX1, overY1, test_size=0.3,random_state=42)
x_train=scaler.fit_transform(x_train)
x_test=scaler.fit_transform(x_test)


# Fit the random forest classifier to the training data after oversampling using the SMOTE

rfs=rf
rfs.fit(x_train, y_train)
y_pred=rfs.predict(x_test)
```

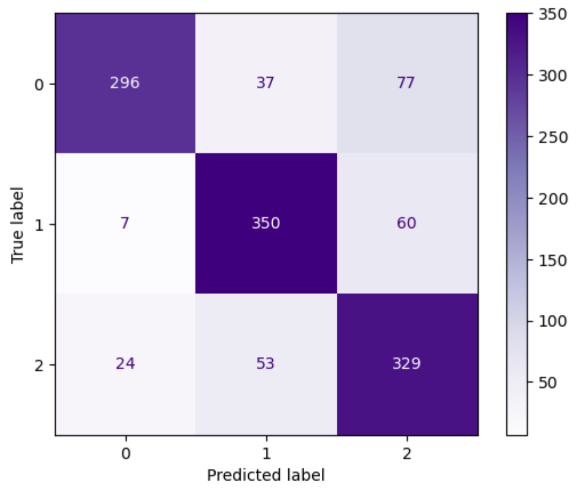
```
print("Accuracy on training set after SMOTE Oversampling : {:.3f}".format(rfs.score(x_train, y_train)))
print("ACC of training set model: %.4f After SMOTE Oversampling" %accuracy_score(y_pred,y_test))
```

```
Accuracy on training set after SMOTE Oversampling : 0.921
ACC of model: 0.7908 After SMOTE Oversampling
```

In the Figure above is the Random Forest model accuracies after removing the outliers and after using standard scalar and Oversampling using SMOTE. And as u can see the accuracies decreased but the confusion matrix improved a lot.

```
plot_confusion_matrix(rfs,x_test,y_test,cmap='Purples')
```

```
]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2b0a6c88d90>
```



In the Figure above is the Random Forest model confusion matrix after removing the outliers and after using standard scalar and Oversampling using SMOTE.

```
gradr= grad
gradr.fit(x_train, y_train)
# Fit the classifier to the training data after oversampling

y_pred=gradr.predict(x_test)
print("Accuracy on training set after random Oversampling : {:.3f}".format(gradr.score(x_train, y_train)))
print("ACC of random Forest model testing set: %.4f After random Oversampling" %accuracy_score(y_pred,y_test))
```

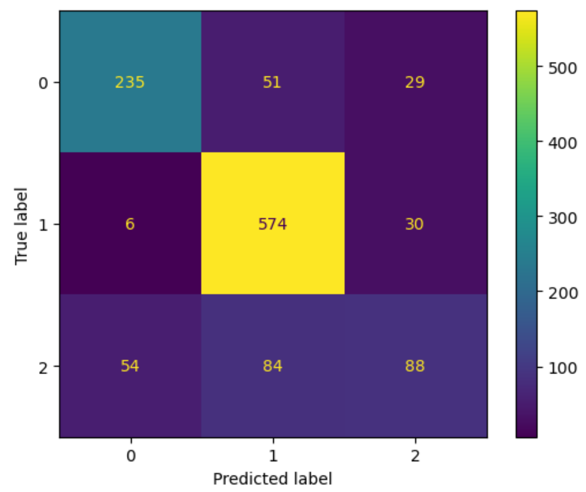
```
Accuracy on training set after random Oversampling : 0.846
ACC of random Forest model testing set: 0.7793 After random Oversampling
```

```
#print the classification report on gradient boosting after oversampling
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.75      0.80      0.77       295
           1       0.94      0.81      0.87       709
           2       0.39      0.60      0.47       147

    accuracy                           0.78      1151
   macro avg       0.69      0.73      0.70      1151
weighted avg       0.82      0.78      0.79      1151
```

In the Figure above is the gradient boosting model classification report and accuracies after removing the outliers and after Oversampling using random oversampling.

```python
#plot the confusion matrix on gradient boosting after oversampling
Confusion = confusion_matrix(y_test, y_pred)
CM=ConfusionMatrixDisplay(Confusion)
CM.plot()
plt.show()
```



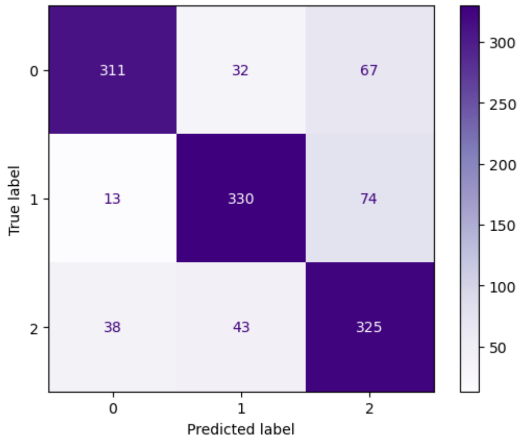
In the Figure above is the gradient boosting model confusion matrix and after removing the outliers and Oversampling using random oversampling. Although the model performance did not improve through the random oversampling however, using the SMOTE improved in both accuracy and confusion matrix as shown below in the figure.

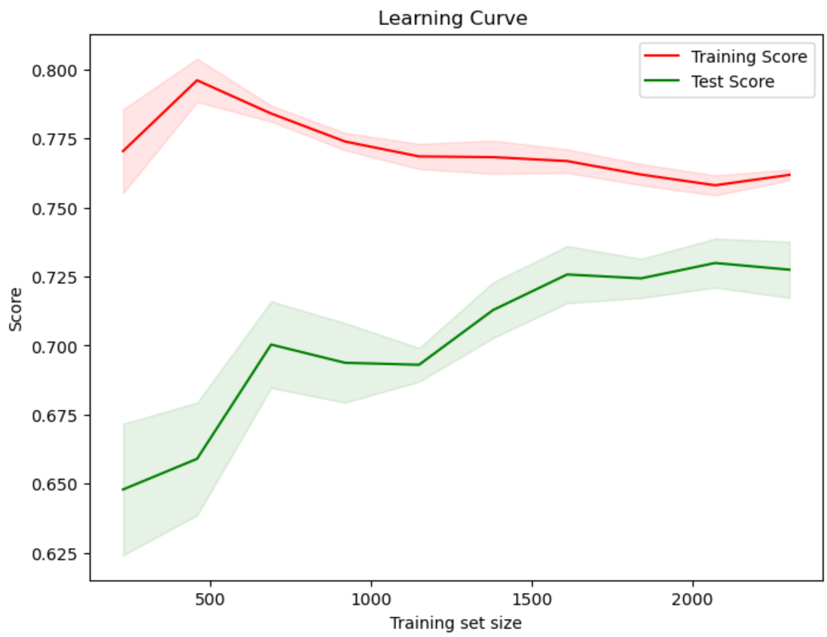
```python
grads = grad
grads.fit(x_train, y_train)
y_pred=grads.predict(x_test)
print("Accuracy on training set after SMOTE Oversampling : {:.3f}".format(grads.score(x_train, y_train)))
print("Accuracy on test set after SMOTE Oversampling: {:.3f}".format(grads.score(x_test, y_test)))
```

```
Accuracy on training set after SMOTE Oversampling : 0.881
Accuracy on test set after SMOTE Oversampling: 0.783
```

```python
plot_confusion_matrix(grads,x_test,y_test,cmap='Purples')
```

```
1]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2b0a9da5340>
```

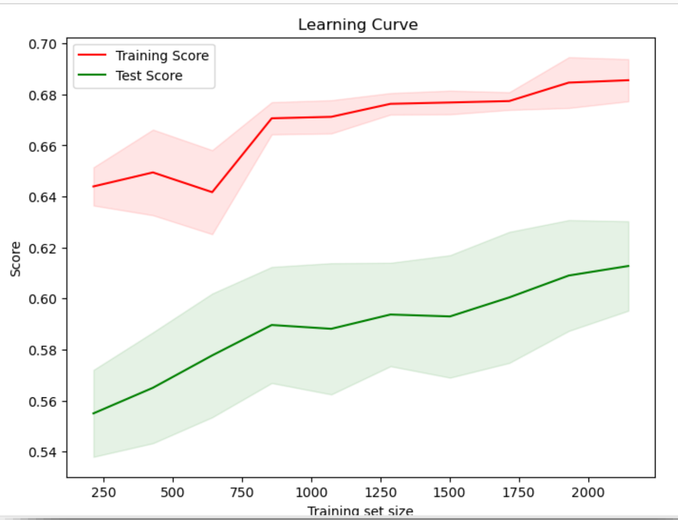




In the figure above is the decision tree model learning curve after smote oversampling.

```
knnr=knn
knnr.fit(x_train, y_train)
# Fit the classifier to the training data after oversampling

y_pred=knnr.predict(x_test)
print("Accuracy on training set after random Oversampling : {:.3f}".format(knnr.score(x_train, y_train)))
print("ACC of knn model testing set: %.4f After random Oversampling" %accuracy_score(y_pred,y_test))
```

```
Accuracy on training set after random Oversampling : 0.687
ACC of knn model testing set: 0.6142 After random Oversampling
```




```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.53      0.57      0.55       297
           1       0.82      0.66      0.73       760
           2       0.17      0.40      0.24        94

    accuracy                           0.61      1151
   macro avg       0.51      0.54      0.51      1151
weighted avg       0.69      0.61      0.64      1151
```

The knn model did not improve using random over sampler as shown above. However, when I used the smote oversampler the accuracy, learning curve, precision, recall, and the confusion matrix improved a lot as shown below in the figure.

```
knns=knn
knns.fit(x_train, y_train)
y_pred=knns.predict(x_test)
```

```
print("Accuracy on training set after SMOTE Oversampling : {:.3f}".format(knns.score(x_train, y_train)))
print("ACC of model: %.4f After SMOTE Oversampling" %accuracy_score(y_pred,y_test))
```

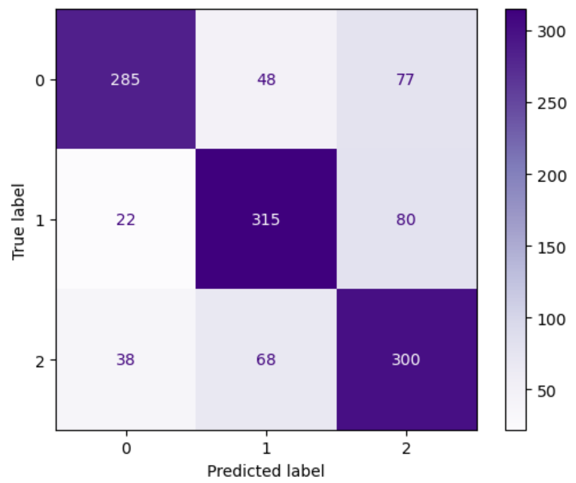
```
Accuracy on training set after SMOTE Oversampling : 0.809
ACC of model: 0.7299 After SMOTE Oversampling
```


```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.70      0.83      0.75       345
           1       0.76      0.73      0.74       431
           2       0.74      0.66      0.70       457

    accuracy                           0.73      1233
   macro avg       0.73      0.74      0.73      1233
weighted avg       0.73      0.73      0.73      1233
```

```
plot_confusion_matrix(knns,x_test,y_test,cmap='Purples')
```

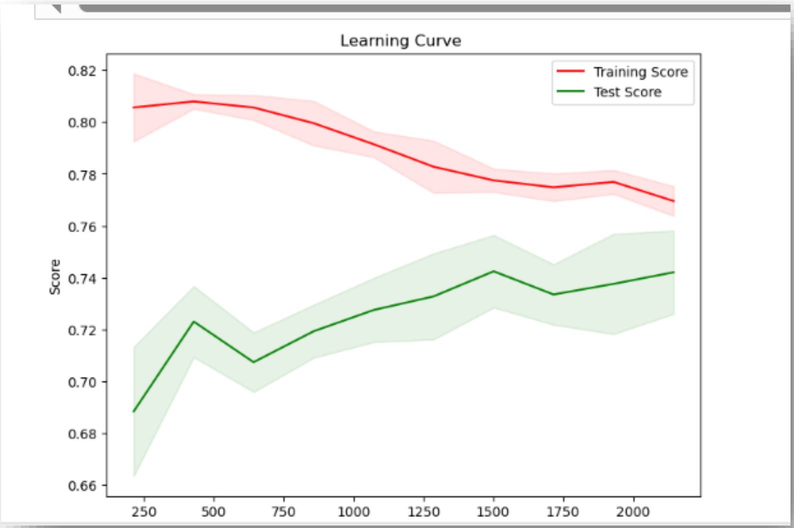
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2b0a9e53e80>
```



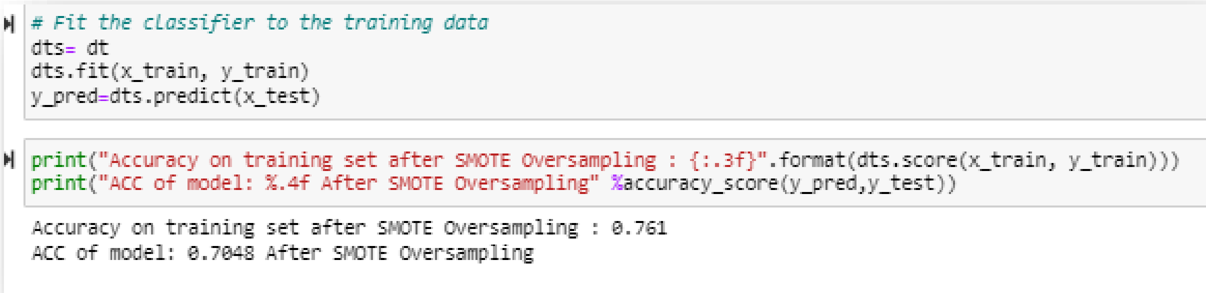
In the decision tree model its performance when using the random oversampler is higher than the Smote as shown below.

```
dtr= dt
dtr.fit(x_train, y_train)
# Fit the classifier to the training data after oversampling

y_pred=dtr.predict(x_test)
print("Accuracy on training set after random Oversampling : {:.3f}".format(dtr.score(x_train, y_train)))
print("ACC of decision tree model testing set: %.4f After random Oversampling" %accuracy_score(y_pred,y_test))
```

```
Accuracy on training set after random Oversampling : 0.775
ACC of decision tree model testing set: 0.7576 After random Oversampling
```



The figure above shows the learning curve of the decision tree model after it was oversampled using the random oversampler.



```
# Fit the classifier to the training data
dts= dt
dts.fit(x_train, y_train)
y_pred=dts.predict(x_test)
```

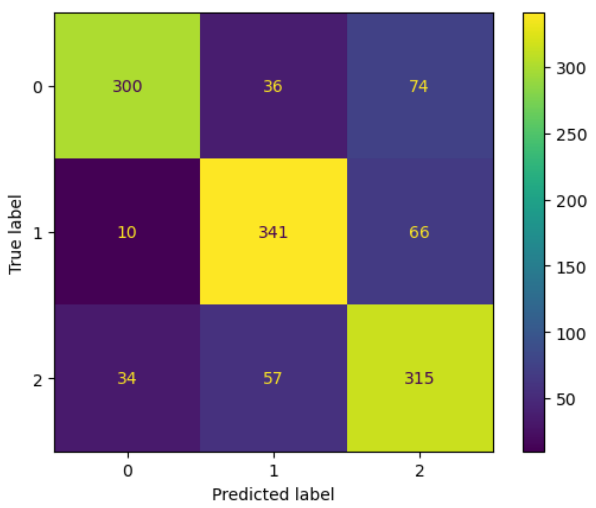
```
print("Accuracy on training set after SMOTE Oversampling : {:.3f}".format(dts.score(x_train, y_train)))
print("ACC of model: %.4f After SMOTE Oversampling" %accuracy_score(y_pred,y_test))
```

```
Accuracy on training set after SMOTE Oversampling : 0.761
ACC of model: 0.7048 After SMOTE Oversampling
```

The figure above shows the accuracy of the decision tree model after it was oversampled using the SMOTE and as I stated above its performance is lower when I used smote than when I used random oversampler.

The figure above shows the learning curve of the decision tree model after it was oversampled using the Smote.

```
Ensemble Model Accuracy: 0.78
Classification Report for Ensemble Model:
              precision    recall  f1-score   support

           0       0.87      0.73      0.80       410
           1       0.79      0.82      0.80       417
           2       0.69      0.78      0.73       406

    accuracy                           0.78      1233
   macro avg       0.78      0.78      0.78      1233
weighted avg       0.78      0.78      0.78      1233
```

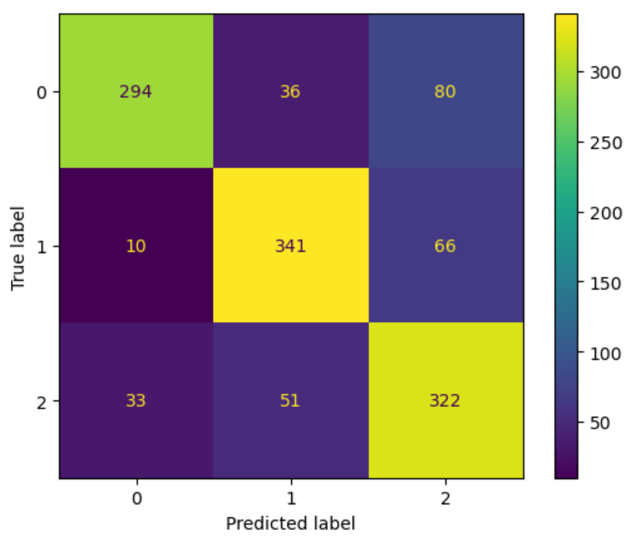


The above figure shows the ensembling classification report and the confusion matrix when i tried the (hard)voting on the Logistic Regression and the models that I oversampled using the SMOTE over Sampling.

```
LogisticRegression 0.7396593673965937
DecisionTreeClassifier 0.7047850770478508
KNeighborsClassifier 0.7299270072992701
RandomForestClassifier 0.7907542579075426
GradientBoostingClassifier 0.7834549878345499
VotingClassifier 0.7753446877534469
```

And the above figure here only shows the used individual models in ensembling.

```
Ensemble Model Accuracy: 0.78
Classification Report for Ensemble Model:
              precision    recall  f1-score   support

           0       0.87      0.72      0.79       410
           1       0.80      0.82      0.81       417
           2       0.69      0.79      0.74       406

    accuracy                           0.78      1233
   macro avg       0.79      0.78      0.78      1233
weighted avg       0.79      0.78      0.78      1233
```

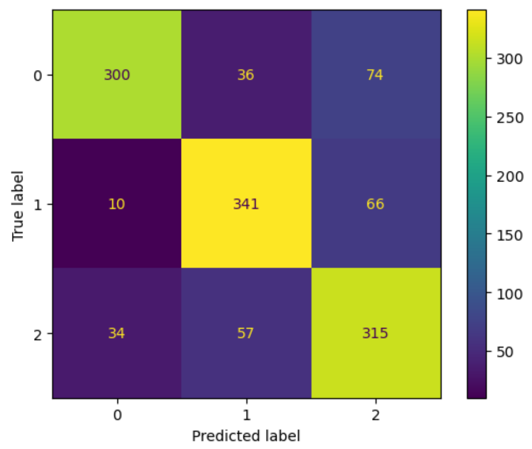


The above figure shows the ensembling classification report and the confusion matrix when i tried the (soft)voting on the Logistic Regression and the models that I oversampled using the SMOTE over Sampling.

```
LogisticRegression 0.7396593673965937
DecisionTreeClassifier 0.7047850770478508
KNeighborsClassifier 0.7299270072992701
RandomForestClassifier 0.7907542579075426
GradientBoostingClassifier 0.7834549878345499
VotingClassifier 0.7761557177615572
```

And the above figure here only shows the used individual models in ensembling.

```
Ensemble Model Accuracy: 0.78
Classification Report for Ensemble Model:
              precision    recall  f1-score   support

           0       0.87      0.73      0.80       410
           1       0.79      0.82      0.80       417
           2       0.69      0.78      0.73       406

    accuracy                           0.78      1233
   macro avg       0.78      0.78      0.78      1233
weighted avg       0.78      0.78      0.78      1233
```

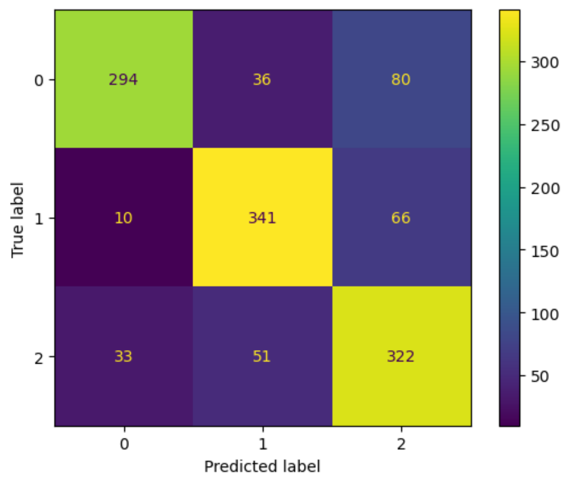


The above figure shows the ensembling classification report and the confusion matrix when i tried the (hard)voting on the Logistic Regression and the models that I oversampled using the Random over Sampling.

```
LogisticRegression 0.7396593673965937
DecisionTreeClassifier 0.7047850770478508
KNeighborsClassifier 0.7299270072992701
RandomForestClassifier 0.7907542579075426
GradientBoostingClassifier 0.7834549878345499
VotingClassifier 0.7753446877534469
```

And the above figure here only shows the used individual models in ensembling.

```
Ensemble Model Accuracy: 0.78
Classification Report for Ensemble Model:
              precision    recall  f1-score   support

           0       0.87      0.72      0.79       410
           1       0.80      0.82      0.81       417
           2       0.69      0.79      0.74       406

    accuracy                           0.78      1233
   macro avg       0.79      0.78      0.78      1233
weighted avg       0.79      0.78      0.78      1233
```



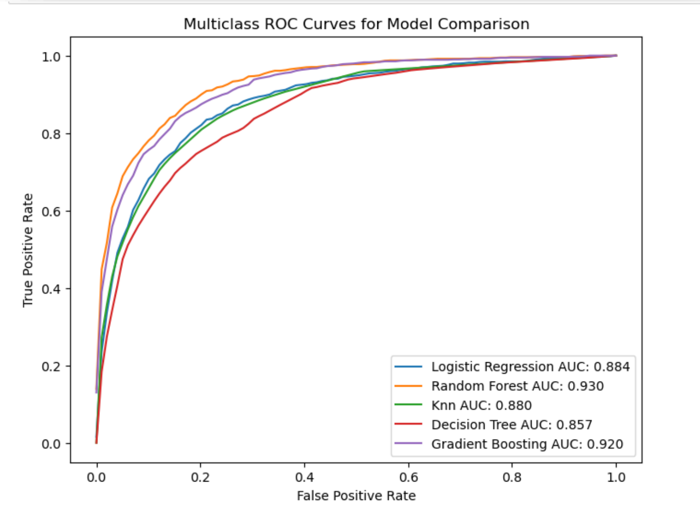
The above figure shows the ensembling classification report and the confusion matrix when i tried the (soft)voting on the Logistic Regression and the models that I oversampled using the Random over Sampling.

```
LogisticRegression 0.7396593673965937
DecisionTreeClassifier 0.7047850770478508
KNeighborsClassifier 0.7299270072992701
RandomForestClassifier 0.7907542579075426
GradientBoostingClassifier 0.7834549878345499
VotingClassifier 0.7761557177615572
```

And the above figure here only shows the used individual models in ensembling.

And as u can see the soft voting is slightly better than hard voting

4. ROC Curves for Model Comparison



As you can see above in the figure, the Logistic Regression and the KNN AUC is very close to each other which is why their ROC curves are very similar to each other.

And The overall F1 score of the models improved after the oversampling which could have been low at the beginning due to the imbalance.

Models	Bias	Variance	MSE
Logistic regression	0.669	0.629	0.587
Random Forest	0.678	0.640	0.483
KNN	0.672	0.665	0.600
Decision Tree	0.679	0.632	0.610
Gradient Boosting	0.671	0.642	0.482

{Abdelrahman Ayman 211896}

1. **Data Description:**

The Abalone dataset is a famous machine learning dataset that contains physical measurements of abalones, which are a species of marine mollusk. Typically, the purpose of using this dataset is to predict the age of an abalone based on its morphological traits. The dataset consists of length, diameter, and height, as well as whole weight, shucked weight, viscera weight, and shell weight. The number of rings, which is regarded as an indirect indication of the abalone's age, is usually the target variable.

This figure shows the distribution of the labels (old, young) of the dataset according to all features in the dataset.

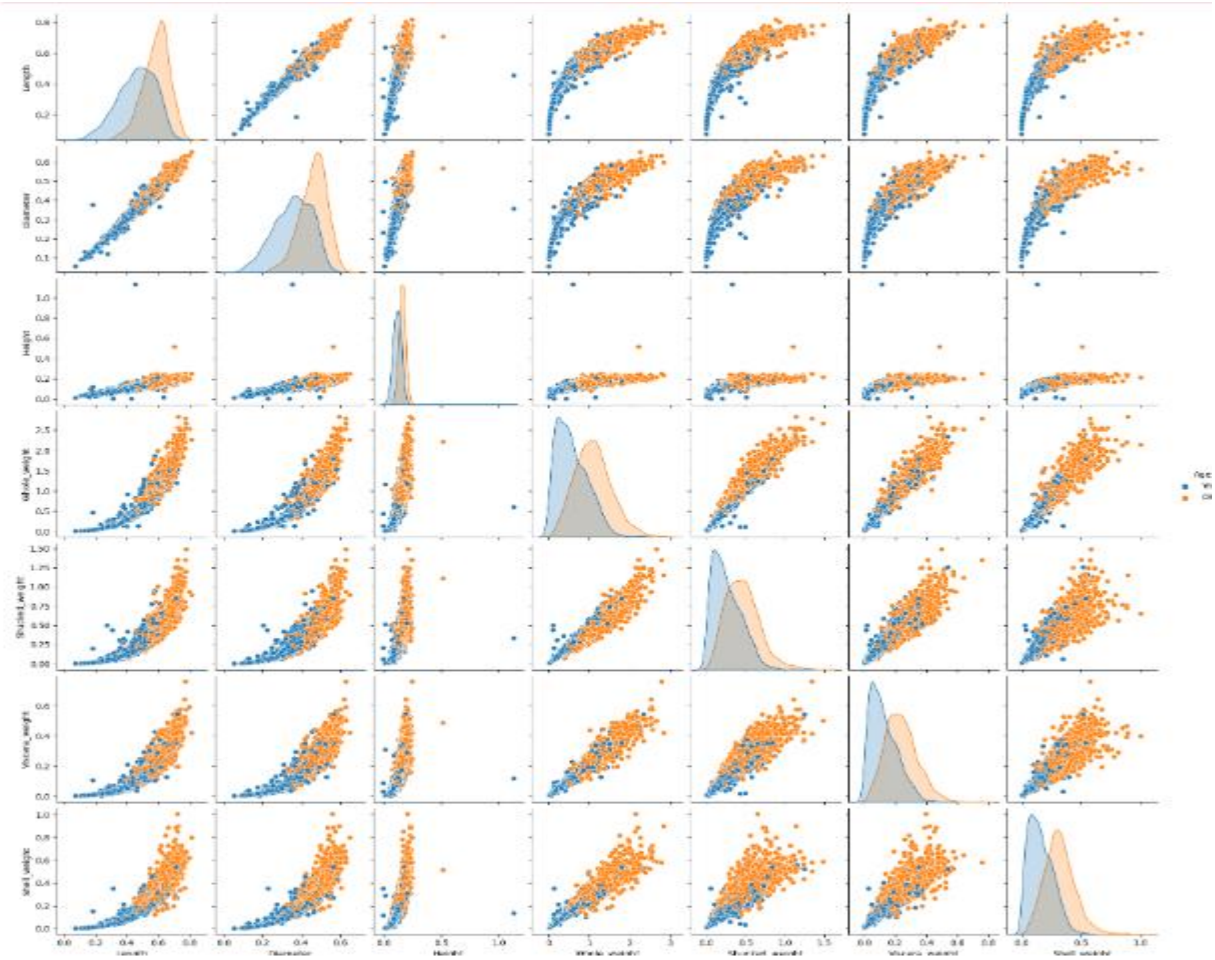


Figure 1

1. <u>Data Preprocessing:</u>

• Because the abalone's age is determined by adding 1.5 to the number of rings for each abalone, I added a new column (Age) to replace the rings column, which holds the abalone ages after adding 1.5 to the number of rings, as seen in the figure below.

```
# Create a new column 'Age' by adding 1.5 to the 'Class_number_of_rings' column
abalone['Age'] = abalone['Class_number_of_rings']+1.5
# Drop the 'Class_number_of_rings' column from the DataFrame
abalone.drop('Class_number_of_rings', axis = 1, inplace = True)
```

```
abalone.head()
```

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight	Age
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	16.5
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	8.5
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	10.5
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	11.5
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	8.5

- Using the pd.cut function, I categorized the newly produced 'Age' column into discrete age groups ('Young' and 'Old'). The 'bins' argument specifies how the values in the 'Age' column are binned into defined intervals. In this situation, ages less than or equal to 11 are labeled "Young," while more than 11 are labeled "Old" as shown in the figure below.

```
# Create age groups ('Young' and 'Old') based on the 'Age' column using the cut function
# The bins parameter defines the ranges for each age group
abalone['Age'] = pd.cut(abalone['Age'], bins=[0, 11, 200], labels=['Young', 'Old'])
abalone.head()
```

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight	Age
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	Old
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	Young
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	Young
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	Old
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	Young

- Nulls: the dataset contains no null values.

```
]: # Check for missing values in each column of the DataFrame
abalone.isnull().sum()
```

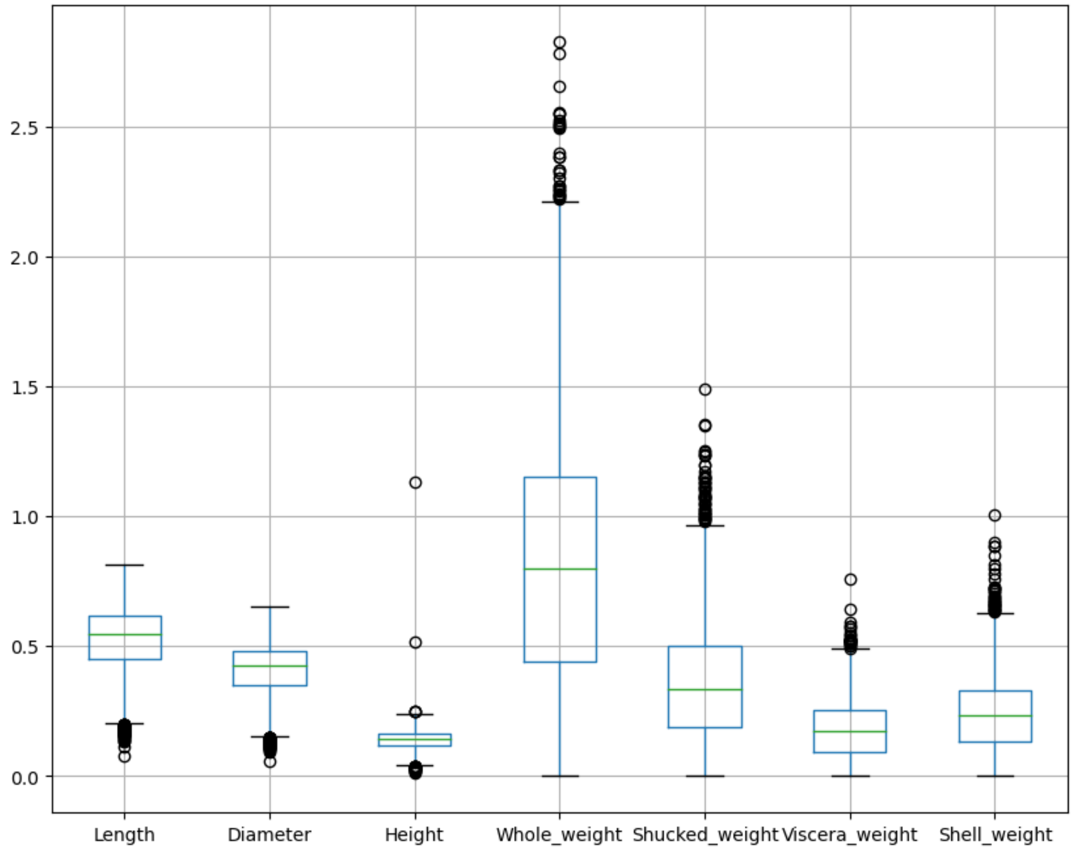
```
]: Sex               0
   Length            0
   Diameter          0
   Height            0
   Whole_weight      0
   Shucked_weight    0
   Viscera_weight    0
   Shell_weight      0
   Age               0
   dtype: int64
```

• The "Height" column contains two values equal to zero, which I removed because it makes no sense that the abalone's height is equal to zero.

```
# Filter rows where the 'Height' column is equal to 0 and print them
rows0 = abalone[abalone['Height'] == 0]
print(rows0)
```

```
     Sex  Length  Diameter  Height  Whole_weight  Shucked_weight  \
1257   I   0.430      0.34     0.0         0.428          0.2065
3996   I   0.315      0.23     0.0         0.134          0.0575

      Viscera_weight  Shell_weight    Age
1257          0.0860        0.1150  Young
3996          0.0285        0.3505  Young
```

- The box plot shows the values of each column and the outliers in each column.



- Outliers: using the IQR method to drop the outliers with 0.25 for Q1 and 0.75 for Q3.

In [22]:
```
# Calculate the first quartile (Q1), third quartile (Q3), and interquartile range (IQR) for each column in abalone_x
Q1 = abalone_x.quantile(0.25)
Q3 = abalone_x.quantile(0.75)
IQR = Q3 - Q1

# Use the IQR to filter out outliers and create a cleaned version of the abalone dataset
abalone_cleaned = abalone[~((abalone_x < (Q1 - 1.5 * IQR)) | (abalone_x > (Q3 + 1.5 * IQR))).any(axis=1)]
abalone_cleaned.shape
```

Out[22]:
```
(4024, 9)
```

2. **Models:**

- To see the results before removing the outliers, we trained an ensemble model that uses hard voting and consists of three models: random forest, support vector machine, and logistic regression.

The figure shows that the accuracy of the training is 0.818 and accuracy of the testing is 0.801.

```
In [19]: # Create a Voting Classifier model with hard voting with rf , svm , and log models
voting = VotingClassifier(
    estimators=[('logReg', log), ('rf', rf), ('svm', svm)],
    voting='hard')

# Fit the Voting Classifier model on the training data
voting.fit(X_train0, y_train0)
```

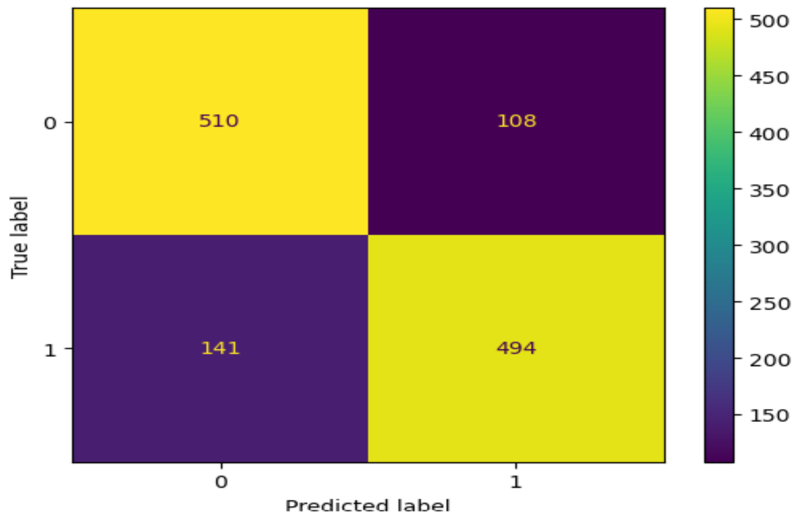
```
Out[19]:
VotingClassifier
logReg                  rf                        svm
LogisticRegression      RandomForestClassifier    SVC
```

```
In [20]: # Print the accuracy on the training set and on the test set
print("Accuracy on training set: {:.3f}".format(voting.score(X_train0, y_train0)))
print("Accuracy on test set: {:.3f}".format(voting.score(X_test0, y_test0)))
```

```
Accuracy on training set: 0.818
Accuracy on test set: 0.801
```

This figure shows the confusion matrix of the ensemble model before removing the outliers.

```
: # generating the confusion matrix on the ensemble model 'voting'
y_pred = voting.predict(X_test0)
confusion = confusion_matrix(y_test0, y_pred)
cm=ConfusionMatrixDisplay(confusion)
cm.plot()
plt.show()
```



- To see the results after removing the outliers, we trained an ensemble model that uses hard voting and consists of three models: random forest, support vector machine, and logistic regression.

The figure shows that the accuracy of the training increased and became 0.823 and accuracy of the testing decreased and became 0.781.

```
# Create a Voting Classifier model with hard voting with rf , svm , and log models
voting2 = VotingClassifier(
    estimators=[('logReg', log), ('rf', rf), ('svm', svm)],
    voting='hard')

# Fit the Voting Classifier model on the training dat
voting2.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(voting2.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(voting2.score(X_test, y_test)))
```

```
Accuracy on training set: 0.823
Accuracy on test set: 0.781
```

- By using the grid search we found the best hyperparameters for each model then we made a new ensemble model that consists of the three models and using the soft voting for training.

This figure shows the models with new values for each hyperparameters that will be used in the ensemble model.

```
# Create a Random Forest Classifier model after hyperparameter tunning with max depth 8 and n_estimators 400
# max_depth: Maximum depth of the tree
# n_estimators: The number of trees in the forest
rf2 = RandomForestClassifier(max_depth=8, n_estimators=400, random_state=42)

# Create an SVM Classifier model after hyperparameter tunning with rbf kernel and C 200 and gamma 1
# kernel: Specifies the kernel type. 'rbf' stands for radial basis function.
# C: Regularization parameter
# gamma: Kernel coefficient for 'rbf'
svm2 = SVC(kernel='rbf', C=200, gamma=1,random_state=42,probability=True)

# Create a DecisionTreeClassifier with max depth 7 and min sample split 23
dt = DecisionTreeClassifier(max_depth=7,min_samples_split=23,random_state=42)
```

This figure shows the new ensemble model and its results.

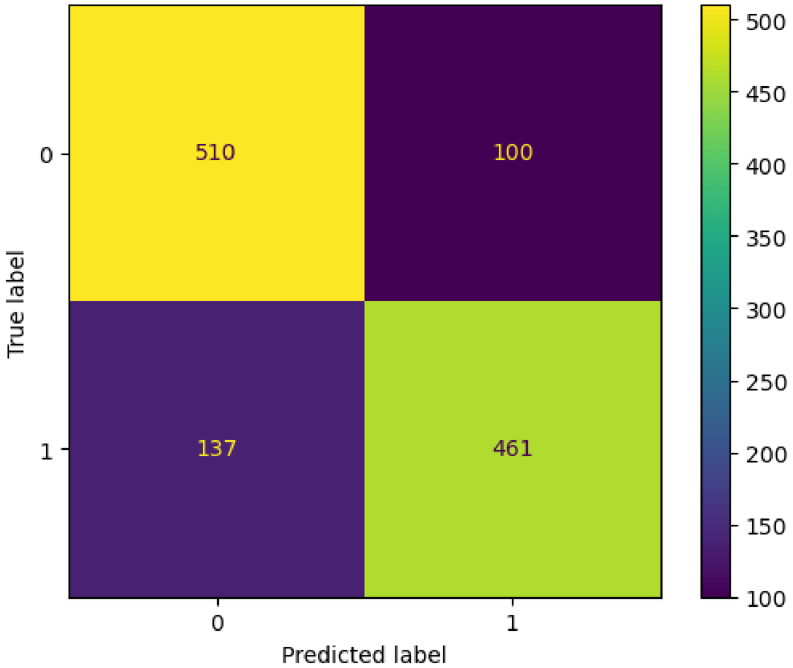
the accuracy of the training increased and became 0.839 and accuracy of the testing is decreased and became 0.804.

```
# Create a Voting Classifier model with hard voting with rf , svm , and dt models
voting_bst = VotingClassifier(
    estimators=[('dt', dt), ('rf', rf2), ('svm', svm2)],
    voting='soft')

# Fit the Voting Classifier model on the training dat
voting_bst.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(voting_bst.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(voting_bst.score(X_test, y_test)))
```

```
Accuracy on training set: 0.839
Accuracy on test set: 0.804
```

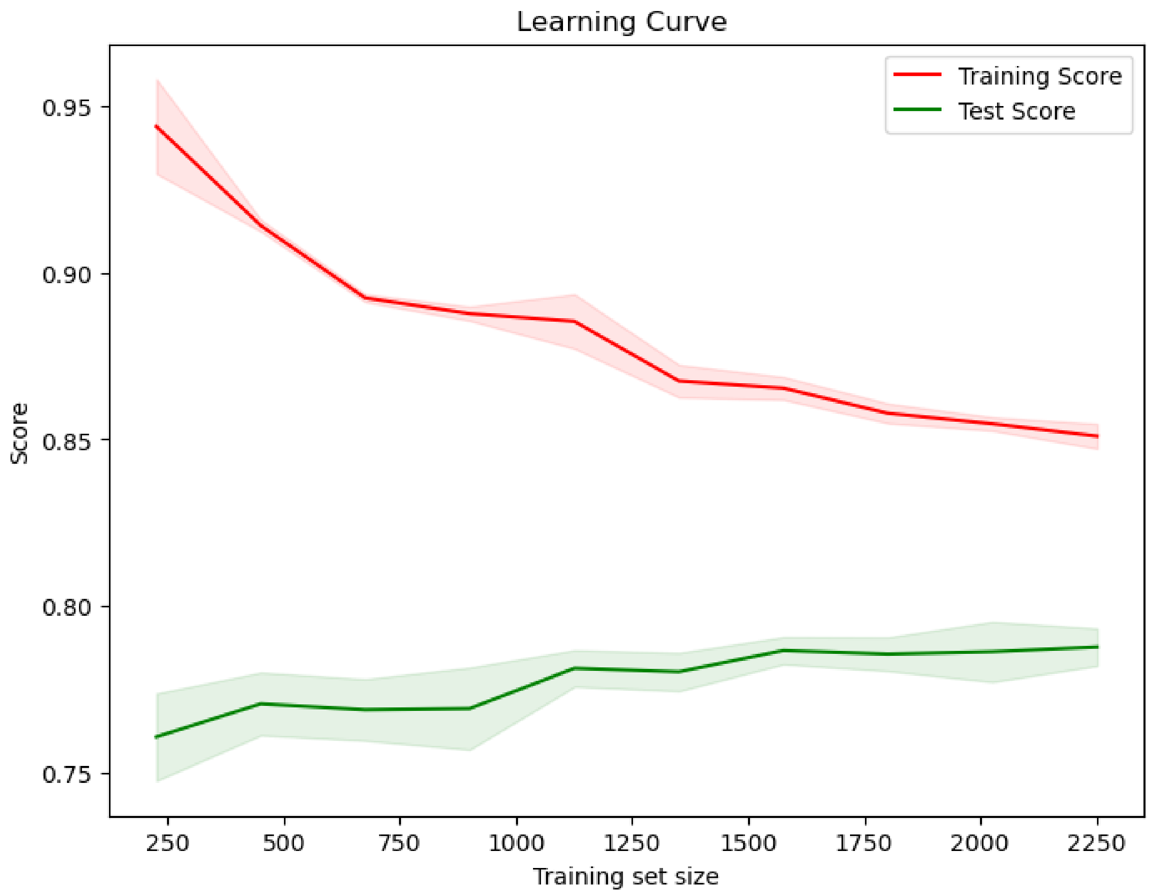
This figure shows the confusion matrix of the new ensemble model.

```
# generating the confusion matrix on the ensemble model 'voting_bst'
y_pred4 = voting_bst.predict(X_test)
confusion = confusion_matrix(y_test, y_pred4)
cm=ConfusionMatrixDisplay(confusion)
cm.plot()
plt.show()
```



This figure is the learning curve of the new ensemble 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. Model performance Comparison:

- We used the classification report to compare the results of the ensemble models and its three models that make the ensemble.

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 ensemble model which has best precision, recall, f1-score, and accuracy.

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.81      0.78       610
           1       0.79      0.74      0.77       598

    accuracy                           0.78      1208
   macro avg       0.78      0.78      0.78      1208
weighted avg       0.78      0.78      0.78      1208

SVM Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.82      0.80       610
           1       0.81      0.77      0.79       598

    accuracy                           0.80      1208
   macro avg       0.80      0.80      0.80      1208
weighted avg       0.80      0.80      0.80      1208

Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.81      0.78       610
           1       0.79      0.73      0.76       598

    accuracy                           0.77      1208
   macro avg       0.77      0.77      0.77      1208
weighted avg       0.77      0.77      0.77      1208

Voting Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.84      0.81       610
           1       0.82      0.77      0.80       598

    accuracy                           0.80      1208
   macro avg       0.81      0.80      0.80      1208
weighted avg       0.80      0.80      0.80      1208
```

- We used bias and variance and the mean square error to compare the results of each model.

Bias: The difference between the expected values from a model and the true value in the data is represented by bias. A high bias suggests that the model oversimplifies the underlying patterns, resulting in systematic forecasting errors.

Variance evaluates the model's sensitivity to changes in the training data. A high variance indicates that the model is overly complex, collecting noise in the training set and performing badly on fresh, previously unseen data.

MSE (Mean Squared Error): MSE estimates the average squared difference between anticipated and actual data. It is a complete metric that considers both bias and variance. A lower MSE suggests greater model performance, as it represents smaller differences between predictions and true values.

This table shows the results of each model.

model	Bias	Variance	MSE
Ensemble model	0.175	0.045	0.220
Random Forrest	0.187	0.032	0.219
Support vector machine	0.184	0.029	0.213
Decision tree	0.160	0.096	0.256

Clustering:

```
Number of outliers removed: 57
Outlier clusters: Index([0, 6], dtype='int32', name='Cluster')
```

We used k-mean clustering where It begins by standardizing the features and then employing K-Means with seven clusters. Clusters with fewer than 200 points are deemed outliers, and the data points associated with them are eliminated. The number of outliers and the indices of outlier clusters are then printed for the deleted outliers.

Models:

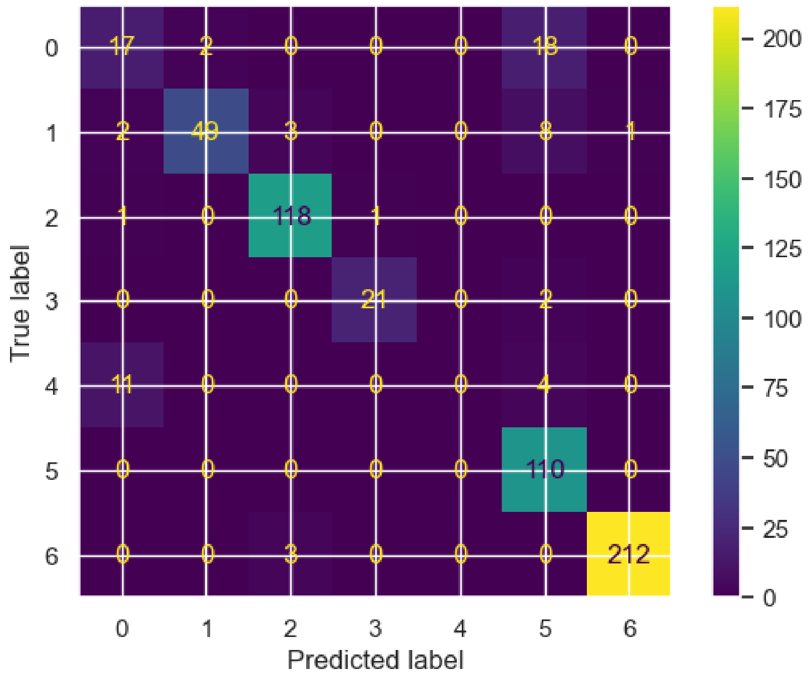
We used the random forest model 4 times

Before the outlier

```
rf1 = RandomForestClassifier(n_estimators=25, max_leaf_nodes=7, n_jobs=-1, random_state=42)
rf1.fit(x_train, y_train)

# Prediction on the test set
y_predef = rf1.predict(x_test)

print("Accuracy on training set: {:.3f}".format(rf1.score(x_train, y_train)))
print("Accuracy on test set: {:.3f}".format(rf1.score(x_test, y_test)))
```

```
Accuracy on training set: 0.902
Accuracy on test set: 0.904
```

Confusion matrix:



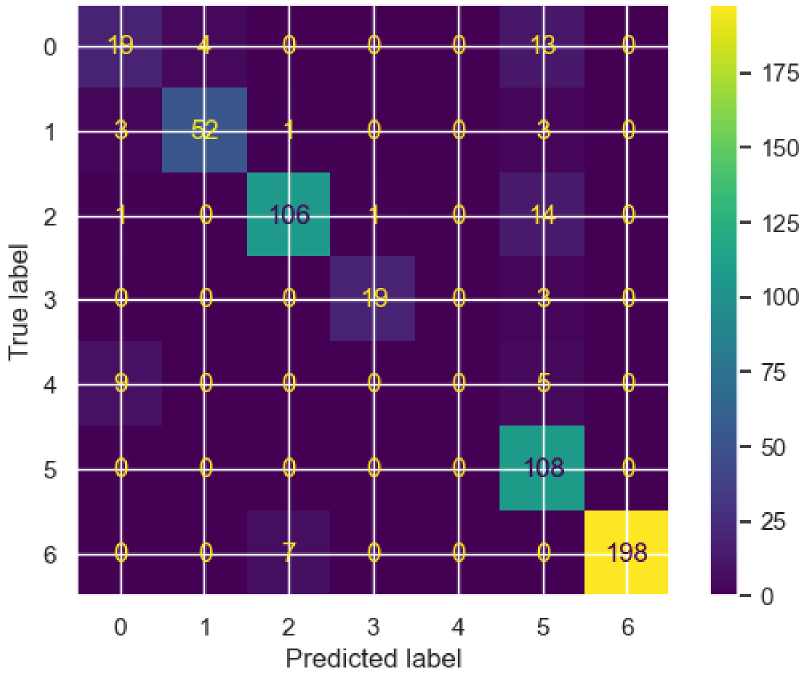
In this cm the distribution is not the best and the fourth class is not seen .

The model after outlier removal:

```
rf2 = RandomForestClassifier(n_estimators=25, max_leaf_nodes=7, n_jobs=-1, random_state=42)
rf2.fit(x_train0, y_train0)

# Prediction on the test set
y_predef2 = rf2.predict(x_test0)

# Evaluate the model

print("Accuracy on training set: {:.3f}".format(rf2.score(x_train0, y_train0)))
print("Accuracy on test set: {:.3f}".format(rf2.score(x_test0, y_test0)))
```

```
Accuracy on training set: 0.904
Accuracy on test set: 0.887
```

The accuracies after the removal of outliers.



The cm is the same as above the only difference is the outlier removal

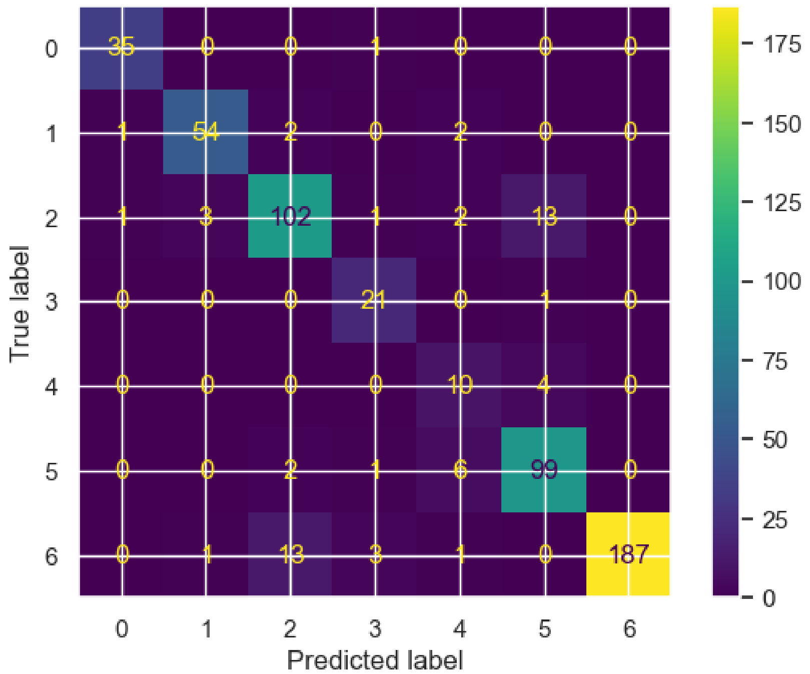
We oversampled using Smote.

```
oversample = SMOTE(random_state=42)
overX,overY=oversample.fit_resample(x_train0, y_train0)
```

The accuracies where slightly better than before using the over sampling of the train set

```python
rf3 = RandomForestClassifier(n_estimators=25, max_leaf_nodes=7, n_jobs=-1, random_state=42)
rf3.fit(overX, overY)


# Evaluate the model

print("Accuracy on training set: {:.3f}".format(rf3.score(overX, overY)))
print("Accuracy on test set: {:.3f}".format(rf3.score(x_test0, y_test0)))
```

```
Accuracy on training set: 0.936
Accuracy on test set: 0.898
```



The cm of the oversample is the best until now in the distribution and it sees the fourth class.

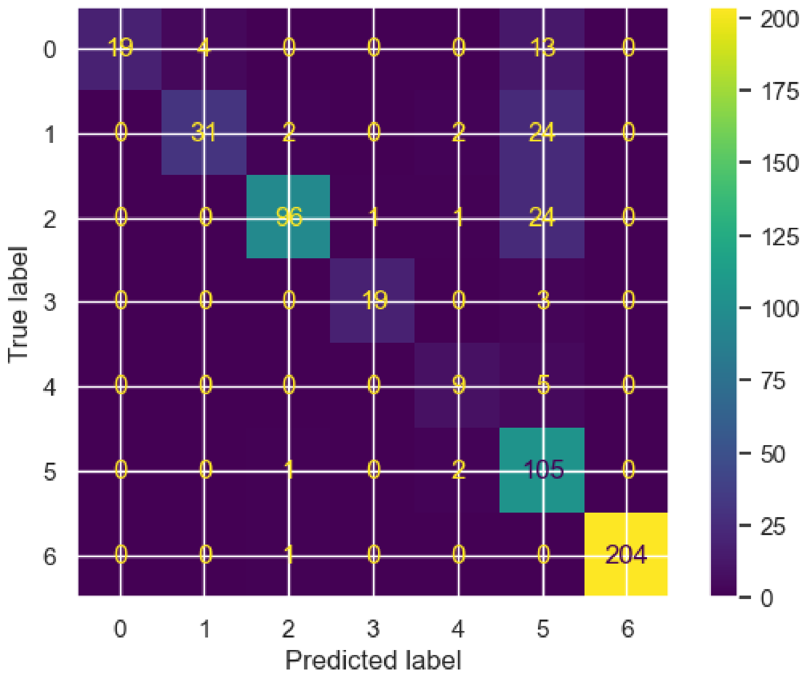
We did under sampling using Tomek links.

```python
from imblearn.under_sampling import TomekLinks
tl = TomekLinks()
X_res, y_res = tl.fit_resample(x_train0, y_train0)
```

```
rf4 = RandomForestClassifier(n_estimators=25, max_leaf_nodes=7, n_jobs=-1, random_state=42)
rf4.fit(X_res, y_res)



# Evaluate the model

print("Accuracy on training set: {:.3f}".format(rf4.score(X_res, y_res)))
print("Accuracy on test set: {:.3f}".format(rf4.score(x_test0, y_test0)))
```

```
Accuracy on training set: 0.922
Accuracy on test set: 0.853
```

Used the rf with the undersampling and the accuracy was lower than oversampling.



The cm distribution is not bad but the true positives in some classes where only better than the smote.

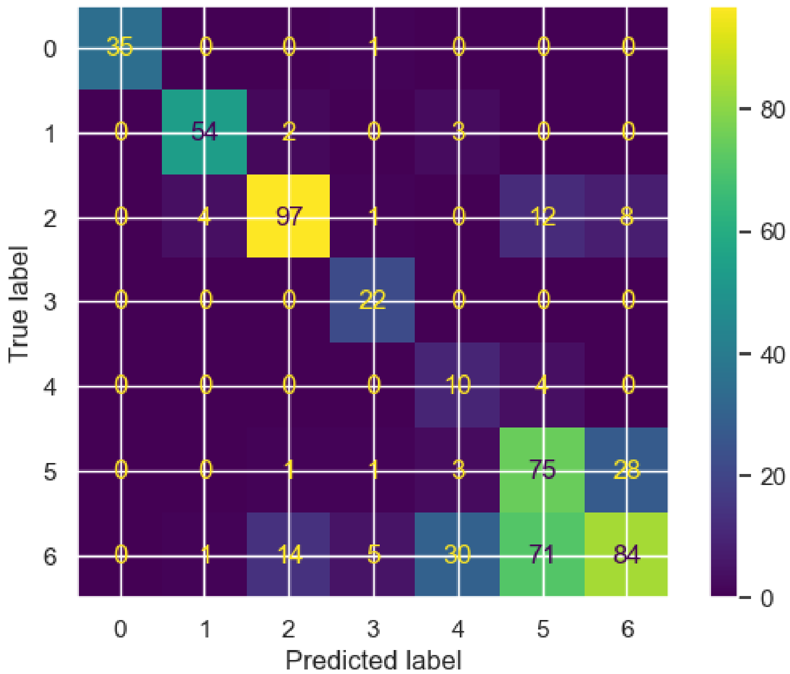
We used the SmoteTomek which combines between the oversample and undersampling

```
from imblearn.combine import SMOTETomek
smt = SMOTETomek(random_state=42,smote=SMOTE(k_neighbors=2))
X, y = smt.fit_resample(x_train0, y_train0)
```

```python
rf5 = RandomForestClassifier(n_estimators=25, max_leaf_nodes=7, n_jobs=-1, random_state=42)
rf5.fit(X, y)


# Evaluate the model

print("Accuracy on training set: {:.3f}".format(rf5.score(X, y)))
print("Accuracy on test set: {:.3f}".format(rf5.score(x_test0, y_test0)))
```

```
Accuracy on training set: 0.868
Accuracy on test set: 0.666
```

The accuracy is the worst here scoring a low 0.666 compared to the others



The cm distribution is bad as the class 6 has a lot of negatives than positives but it is better in some of the classes.

Why Accuracy not reliable?

Accuracy is not necessarily a reliable performance metric, especially when there are uneven class distributions or varying consequences for different sorts of errors. When one class outnumbers the others by a large margin, a model can attain high accuracy by merely predicting the majority class, disguising its failure to correctly categorize minority classes. Furthermore, accuracy does not distinguish between the severity of false positives and false negatives. By focusing on distinct aspects of model performance, task-specific metrics like precision, recall, F1 score, or area under the ROC curve enable a more nuanced evaluation. To achieve a comprehensive and relevant assessment of the model's efficacy, metrics must be linked with the application's specific goals and requirements.