MACHINE LEARNING FINAL PROJECT

Name – Anish Joshi

NetID – AXJ200101

Group No – 37

Semester – Fall 2022

Instructor – Prof. Sriram Natarajan

INTRODUCTION

Problem Statement – Prediction of Wine Quality based on some physiochemical parameters such as pH values, Alcohol content, Acid content, Sugar content, etc.

The dataset has been taken from Kaggle. The link is provided in references. The dataset consists of 6497 columns which is spread across 12 features and an output label. The entire project is divided into four parts i.e., the Data Preprocessing, the implementation of six ML Algorithms, Performance evaluation and conclusion.

DATASET DESCRIPTION

The dataset consists of the following features and an output label.

1. Fixed Acidity – Amount of Tartaric Acid
2. Volatile Acidity – Amount of Acetic Acid
3. Citric Acid – Amount of Critic Acid
4. Residual Sugar – Amount of sugar content left post fermentation
5. Chlorides – Amount of salt present in the wine
6. Free sulfur dioxide – Amount of sulfur dioxide in free form
7. Total sulfur dioxide – Total amount of sulfur dioxide
8. Density – Density of Wine
9. pH – Indicating the pH scale (pH<7 → Acidic, pH=7 → Neutral, pH>7 → Basic)
10. Sulphates – Amount of Potassium Sulphate in wine
11. Alcohol – Alcohol content
12. Wine type – White wine (75%) and red wine (25%)

Output – Wine Quality (0 – if low and 1 if high)

EXPLORATORY DATA ANALYSIS AND DATA PREPROCESSING

This is what our data looks like. The following figure consists of the data_types of the features and the shape of the dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   type                  6497 non-null   object
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

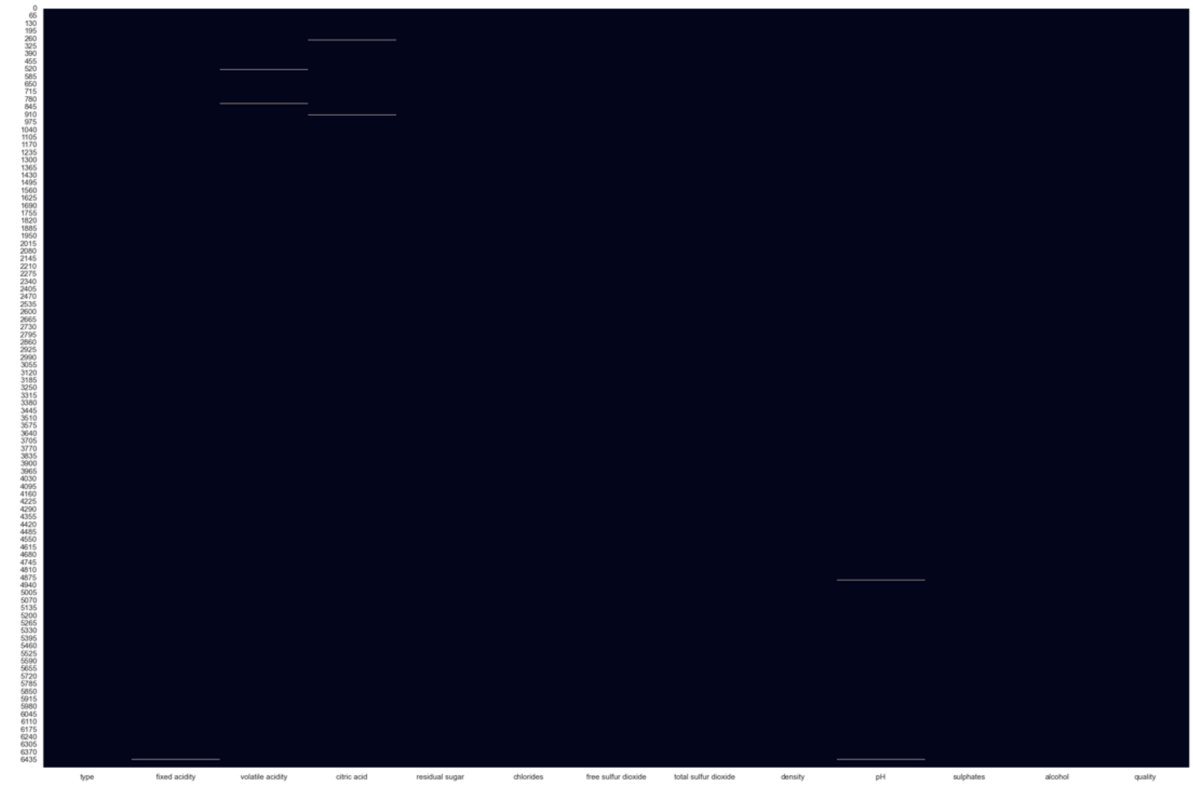
```
]: (6497, 13)
```

Dealing with missing values

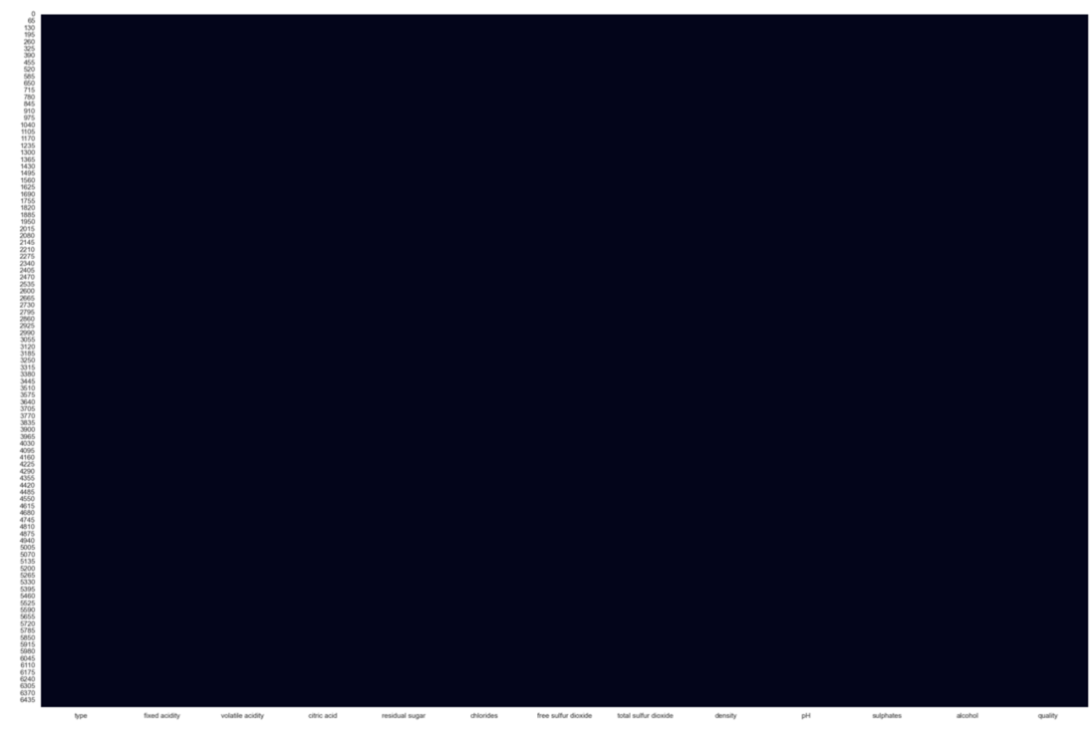
We will start our analysis by dealing with missing values first. Following are the missing values in our dataset in ascending order. We can see that 'fixed acidity' has the maximum number of missing values followed by pH, volatile acidity, sulphates and so on.

```
type                     0
free sulfur dioxide      0
total sulfur dioxide     0
density                  0
alcohol                  0
quality                  0
residual sugar           2
chlorides                2
citric acid              3
sulphates                4
volatile acidity         8
pH                       9
fixed acidity           10
dtype: int64
```

To deal with missing values, instead of removing the missing values, we will replace the missing data with the mean of the feature. This ensures that we do not loss any precious data. We will create a heatmap first to look at the missing values.



Now, to deal with the missing values, we will replace the missing values with the mean of that feature. The following is the heatmap after dealing with the missing values which shows absence of any other missing values.

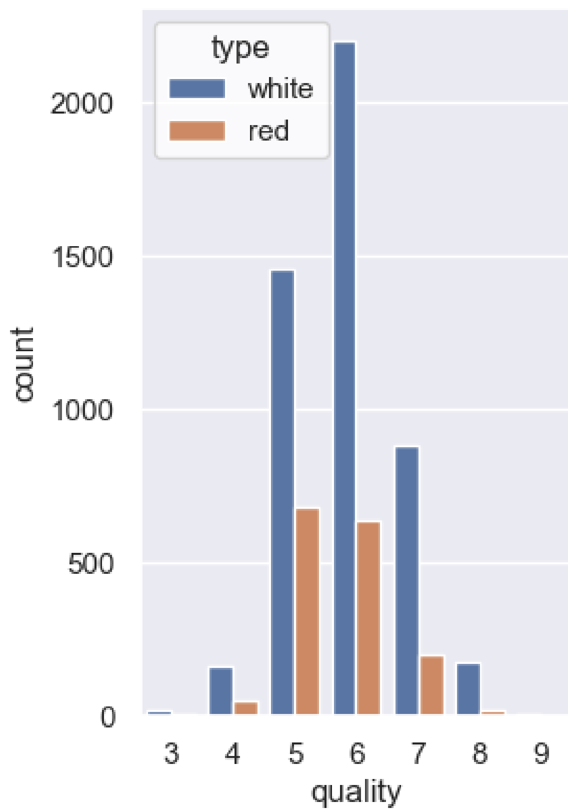


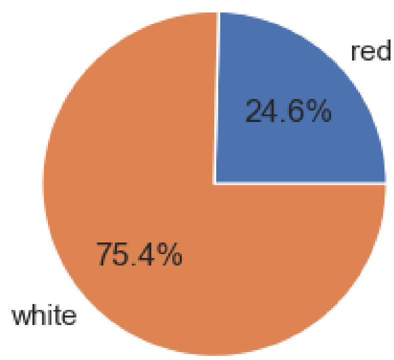
This is how our data looks like after dealing with the missing values.

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000
7.216579	0.339691	0.318722	5.444326	0.056042	30.525319	115.744574	0.994697	3.218395	0.531215	10.491801	5.818378
1.295751	0.164548	0.145231	4.757392	0.035031	17.749400	56.521855	0.002999	0.160637	0.148768	1.192712	0.873255
3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000	3.000000
6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992340	3.110000	0.430000	9.500000	5.000000
7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000	10.300000	6.000000
7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.996990	3.320000	0.600000	11.300000	6.000000
15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000	14.900000	9.000000

The first, second and third row gives us the count, mean and the standard deviation. The fourth row and the last row gives us the minimum and maximum value while the remaining three rows in-between give us the 25%, 50% and 75% values.

<u>Dealing with categorical variable and the target variable</u>





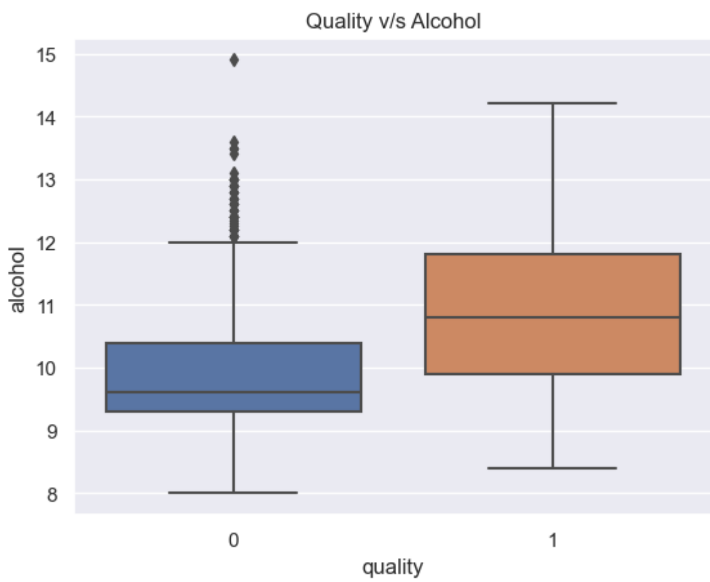
About 75% of the data pertains to white wine while the remaining to red wine. Further, from the bar graph, we can observe that the white wine is the one with the highest quality of 6 and that a significant of data has a quality of 6 which may cause a problem in training our model. To overcome this, we will map the values of quality to low (3-5) and high (6-9), which is further mapped to 0(low) and 1(high). With the help of this, we might be able to deal with the distribution to some extent.

<u>Dealing with Categorical features</u>

Additionally, we will also map the 'type' feature, which is a categorical variable into 0 and 1, where 0 is for type 'red' and 1 is for type 'white'.

The below plot gives us a visual representation of how each feature is related to each other.

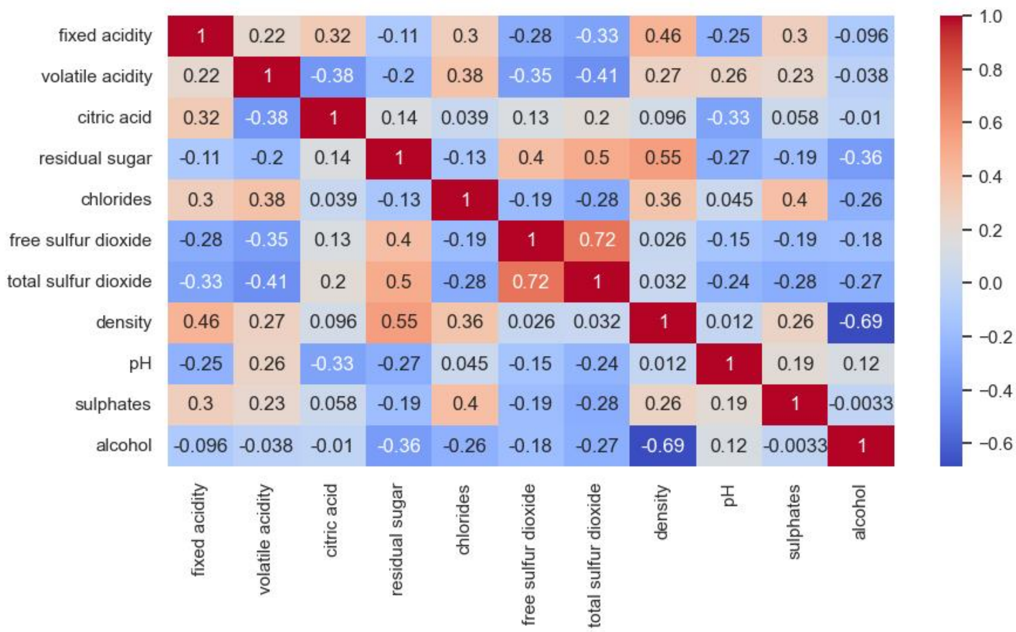
The below figure shows the relation between Alcohol and Quality.



From the above figure, we can infer that wines with high alcohol content have higher ratings.

FEATURE ENGINEERING

Correlation matrix is used to define the relation between all the features.
The following figure is the correlation matrix which provides us an idea about how the features are related to each other.



From the above correlation matrix, we can infer that:

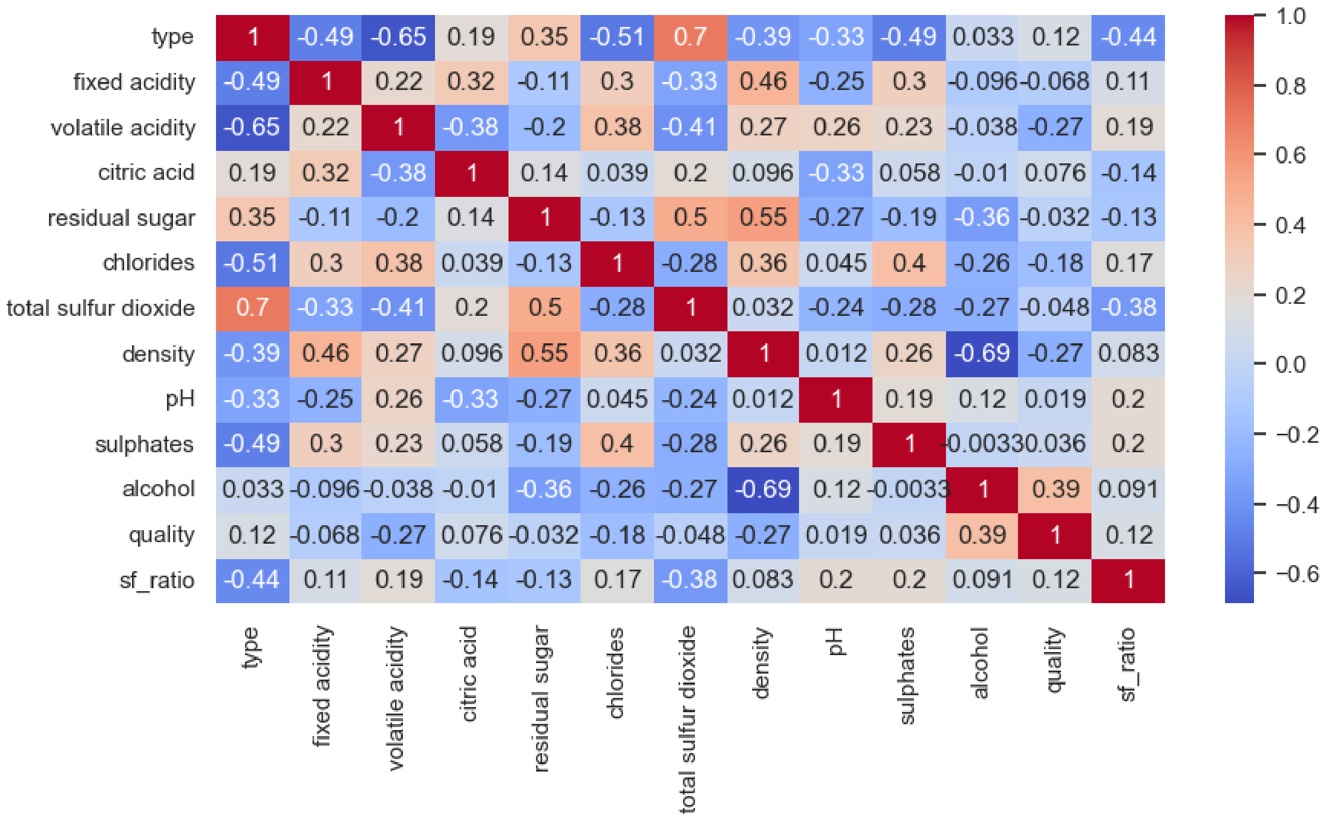
a. ‘free sulfur dioxide’ and ‘total sulfur dioxide’ are highly positively correlated with each other with a value of ‘0.72’, which is obvious, since the former is a part of latter.
b. ‘density’ is moderately correlated with residual sugar, fixed acidity and chlorides while negatively correlated with alcohol.
c. The matrix is symmetric around the diagonal.

<u>Dealing with high correlation values</u>

The high positive correlation between ‘free sulfur dioxide’ and ‘total sulfur dioxide’ might create a problem for us while running our model.

To deal with this, we will drop the ‘free sulfur dioxide’ feature column and insert a new feature which we will term as ‘sf_ratio’ which is the ratio of ‘free sulfur dioxide’ to the ‘total sulfur dioxide’.

The following figure is the correlation matrix we get after we have the done the above process.



From the above correlation matrix, we can now confirm that we have successfully dealt with the high positive correlation between ‘free sulfur dioxide’ and ‘total sulfur dioxide’.

FEATURE SCALING

The following figure gives us an idea of how much are the numerical features normalized.

```
Column  fixed acidity  : ShapiroResult(statistic=0.8797012567520142, pvalue=0.0)
Column  volatile acidity  : ShapiroResult(statistic=0.8758564591407776, pvalue=0.0)
Column  citric acid  : ShapiroResult(statistic=0.9649235010147095, pvalue=4.999661625986668e-37)
Column  residual sugar  : ShapiroResult(statistic=0.8248178958892822, pvalue=0.0)
Column  chlorides  : ShapiroResult(statistic=0.6182848811149597, pvalue=0.0)
Column  total sulfur dioxide  : ShapiroResult(statistic=0.9825838208198547, pvalue=1.5920966626574383e-27)
Column  density  : ShapiroResult(statistic=0.9682108163833618, pvalue=1.330269304272158e-35)
Column  pH  : ShapiroResult(statistic=0.9914466142654419, pvalue=2.197733584552725e-19)
Column  sulphates  : ShapiroResult(statistic=0.8988358378410339, pvalue=0.0)
Column  alcohol  : ShapiroResult(statistic=0.9535516500473022, pvalue=2.9630456028148257e-41)
Column  sf_ratio  : ShapiroResult(statistic=0.9525539875030518, pvalue=1.3871453498351364e-41)

/opt/anaconda3/lib/python3.9/site-packages/scipy/stats/_morestats.py:1800: UserWarning: p-value may not be accurate
for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

From the above output snippet, we can infer that none of the features are normalized. To deal with this, we will scale the data to bring the dataset onto the same scale.
We will use the **MinMaxScaler()** from the scikit-learn's preprocessing library to scale the data.

3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	total sulfur dioxide	density	pH	sulphates	alcohol	sf_ratio	type	quality
0	0.264463	0.126667	0.216867	0.308282	0.059801	0.377880	0.267785	0.217054	0.129213	0.115942	0.289998	1	1
1	0.206612	0.146667	0.204819	0.015337	0.066445	0.290323	0.132832	0.449612	0.151685	0.217391	0.099870	1	1
2	0.355372	0.133333	0.240964	0.096626	0.068106	0.209677	0.154039	0.418605	0.123596	0.304348	0.343415	1	1
3	0.280992	0.100000	0.192771	0.121166	0.081395	0.414747	0.163678	0.364341	0.101124	0.275362	0.275595	1	1
4	0.280992	0.100000	0.192771	0.121166	0.081395	0.414747	0.163678	0.364341	0.101124	0.275362	0.275595	1	1
5	0.355372	0.133333	0.240964	0.096626	0.068106	0.209677	0.154039	0.418605	0.123596	0.304348	0.343415	1	1
6	0.198347	0.160000	0.096386	0.098160	0.059801	0.299539	0.150183	0.356589	0.140449	0.231884	0.237125	1	1
7	0.264463	0.126667	0.216867	0.308282	0.059801	0.377880	0.267785	0.217054	0.129213	0.115942	0.289998	1	1
8	0.206612	0.146667	0.204819	0.015337	0.066445	0.290323	0.132832	0.449612	0.151685	0.217391	0.099870	1	1
9	0.355372	0.093333	0.259036	0.013804	0.058140	0.283410	0.128976	0.387597	0.129213	0.434783	0.232890	1	1
10	0.355372	0.126667	0.246988	0.013037	0.039867	0.131336	0.071139	0.209302	0.191011	0.579710	0.182015	1	0
11	0.396694	0.100000	0.240964	0.055215	0.043189	0.237327	0.146327	0.325581	0.174157	0.246377	0.159676	1	0
12	0.338843	0.066667	0.222892	0.009202	0.051495	0.158986	0.094274	0.356589	0.230337	0.405797	0.228431	1	0
13	0.231405	0.053333	0.240964	0.013804	0.058140	0.315668	0.078851	0.635659	0.168539	0.637681	0.375037	1	1
14	0.371901	0.226667	0.373494	0.286043	0.051495	0.382488	0.252362	0.201550	0.252809	0.246377	0.258438	1	0
15	0.231405	0.060000	0.228916	0.013804	0.038206	0.244240	0.082707	0.410853	0.185393	0.492754	0.272374	1	1
16	0.206612	0.266667	0.024096	0.007669	0.061462	0.214286	0.109697	0.403101	0.078652	0.231884	0.335927	1	1
17	0.282362	0.386667	0.289157	0.009202	0.033223	0.158986	0.040293	0.472868	0.095506	0.695652	0.436161	1	1
18	0.297521	0.173333	0.253012	0.007669	0.039867	0.380184	0.088490	0.310078	0.174157	0.478261	0.091906	1	1
19	0.223140	0.153333	0.084337	0.105828	0.058140	0.292627	0.161751	0.387597	0.157303	0.217391	0.279132	1	0

This is what the data looks like after scaling using MinMaxScaler().

ML ALGORITHMS IMPLEMENTATION

Before starting with the model implementation, few functions were pre-defined to facilitate in evaluating our model performance. Following are the functions:

a. ***def test_report(model,test_size):***
   This function is used to give us the classification report on the test data. It takes the model and the test_size as input parameters.
b. ***def train_report(model,test_size):***
   This function is used to give us the classification report on the train data. It takes the model and the test_size as input parameters.
c. ***def f_p_r(model,test_size,name):***
   This function gives us the f1 score, the precision, and the recall for each model. It takes the model, model name and the test_size as input parameters.
d. ***def confuse_mat(model,test_size):***
   This function will give us the plotted confusion matrices for the inputed test_size. It takes model and test_size as input parameters.
e. ***def roc_auc(model,name,test_size):***
   This function will give us the roc curve for a model and the auc score. It takes test_size, the model and the name of the model as the input parameters.

The following algorithms were used in model creation:

a. Logistic Regression (75--25% train-test-split)
b. Support Vector Machine (75--25% train-test-split)
c. K-Nearest Neighbors (75--25% train-test-split)
d. Decision Tree (75--25% train-test-split)
e. Gradient Boosting (80--20% train-test-split)
f. AdaBoost Classifier (75--25% train-test-split)

We will be implementing our models using the following methodology:

a. Before doing the main implementation, I ran each model with different test-sizes and chose a test size which is optimal for a particular model. (This is not included in the report to keep it short and crisp.)
b. The second step was to run each model with GridSearchCV with 10-fold cross validation (cv=10) and get the best parameter settings for each model.
c. Report the classification report for both train and test set, F1 score, precision, recall, AUC score, Confusion matrices for each model.
d. Next will be to evaluate the performance and give a brief discussion on the performance of each algorithm.
e. The final step will be the Conclusion.

Performance metrics:

a. Accuracy
b. F1 score
c. Precision
d. Recall
e. AUC score

LOGISTIC REGRESSION

Hyperparameter Tuning (GridSearchCV with 10-fold cross validation)

C - Value	Penalty	Solver	Accuracy
0.01	L2	Newton-cg	64.4
0.10	L2	lbfgs	71.8
1	L2	liblinear	74.2
10	L2	Newton-cg	74.36
100	L2	lbfgs	74.26

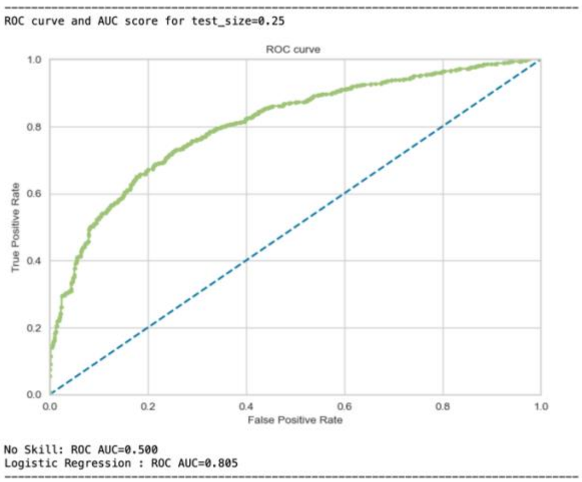
```
-------------------------------------------
Best parameter setting with test size=0.25:  {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
Best accuracy score with test size=0.25:  0.7436462786548625
-------------------------------------------------------------------------------------
```

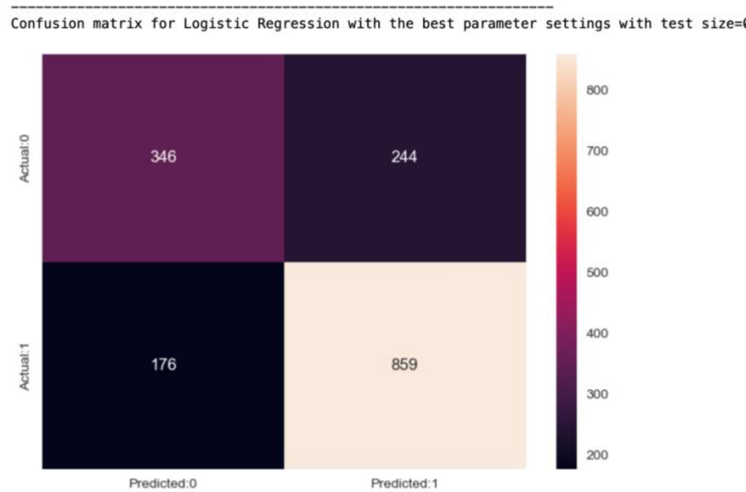
Classification report for Test and Train

```
----------------------------------------------------
Classification Report for Logistic Regression model with Test with test size=0.25
              precision    recall  f1-score   support

           0       0.66      0.59      0.62       590
           1       0.78      0.83      0.80      1035

    accuracy                           0.74      1625
   macro avg       0.72      0.71      0.71      1625
weighted avg       0.74      0.74      0.74      1625

----------------------------------------------------
Classification Report for Logistic Regression model with Train with test size=0.25
              precision    recall  f1-score   support

           0       0.69      0.57      0.62      1794
           1       0.77      0.85      0.81      3078

    accuracy                           0.75      4872
   macro avg       0.73      0.71      0.72      4872
weighted avg       0.74      0.75      0.74      4872

--------------------------------------------------------------------
```

Accuracy (Train): 0.75
Accuracy (Test): 0.74

ROC curve, AUC score and the Confusion Matrix (Average of 10 runs)





F1 score: 0.712, Precision: 0.720, Recall: 0.70
AUC Score: 0.805

SUPPORT VECTOR MACHINE

Hyperparameter Tuning (GridSearchCV with 10-fold cross validation)

Support Vector Machine						
C	Kernel	gamma	degree	random_state	max_iter	Accuracy
0.01	Linear	0.001	2	10	1000	64.9
0.01	Sigmoid	0.001	2	10	1000	65.08
10	poly	0.1	3	100	2000	63.704
10	rbf	0.01	2	10	2000	75.00
1	sigmoid	0.01	3	100	1000	66.45

```
Best parameter setting with test size=0.25:  {'C': 10, 'degree': 2, 'gamma': 0.1, 'kernel': 'rbf', 'max_iter': 2000
, 'random_state': 10}
Best accuracy score with test size=0.25:  0.7500126232874407
```

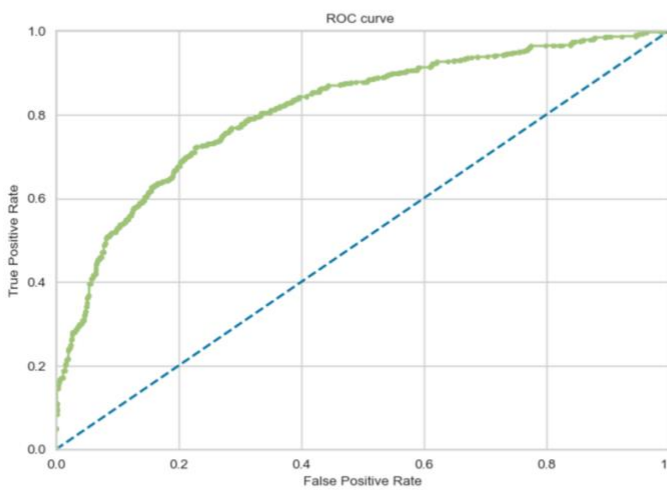
Classification report for Test and Train

```
-----------------------------------------------------------------
Classification Report for SVM model with Test with test size=0.25
              precision    recall  f1-score   support

           0       0.69      0.58      0.63       590
           1       0.78      0.85      0.81      1035

    accuracy                           0.75      1625
   macro avg       0.74      0.72      0.72      1625
weighted avg       0.75      0.75      0.75      1625

-----------------------------------------------------------------
Classification Report for SVM model with Train with test size=0.25
              precision    recall  f1-score   support

           0       0.71      0.56      0.62      1794
           1       0.77      0.87      0.82      3078

    accuracy                           0.75      4872
   macro avg       0.74      0.71      0.72      4872
weighted avg       0.75      0.75      0.75      4872
```

Accuracy (Train): 0.75
Accuracy (Test): 0.75

ROC curve, AUC score and the Confusion Matrix

ROC curve and AUC score for test_size=0.25



Confusion matrix for SVM with the best parameter settings with test size=0.25

No Skill: ROC AUC=0.500
SVM : ROC AUC=0.811

F1 score: 0.7231, Precision: 0.7354, Recall: 0.716
AUC Score: 0.811

Now, we will see how the values of F1 score and precision change with the C values
C=10

```
201]: f_p_r(svm_25,0.25,"SVM")

      With test_size=0.25 for  SVM  f1 score: 0.7231050142160318
      With test_size=0.25 for  SVM  precision: 0.7354531442992558
```

C=0.01

```
For C= 0.01
With test_size=0.25 for  SVM  f1 score: 0.3890977443609023
With test_size=0.25 for  SVM  precision: 0.31846153846153846
-----------------------------------------------------------------
```

C=0.1

```
For C= 0.1
With test_size=0.25 for  SVM  f1 score: 0.447351416480831
With test_size=0.25 for  SVM  precision: 0.7721951381288761
-----------------------------------------------------------------
```

C=1

```
For C= 1
With test_size=0.25 for  SVM  f1 score: 0.7003963125700008
With test_size=0.25 for  SVM  precision: 0.7175554941512388
-----------------------------------------------------------------
```

C=100

```
For C= 100
With test_size=0.25 for  SVM  f1 score: 0.695047971845345
With test_size=0.25 for  SVM  precision: 0.7181595227742561
-----------------------------------------------------------------
```

ADABOOST CLASSIFIER

Hyperparameter Tuning (GridSearchCV with 10-fold cross validation)

Algorithm	Base estimator	Learning rate	N_estimators	Accuracy
SAMME	DecisionTree Classifier(Max_depth=1)	0.1	100	73.41
SAMME.R	DecisionTreeClassifier(max_depth=3)	0.2	500	79.51
SAMME.R	DecisionTreeClassifier(max_depth=4)	0.5	125	76.4
SAMME.R	DecisionTreeClassifier(max_depth=3)	0.1	500	78.3

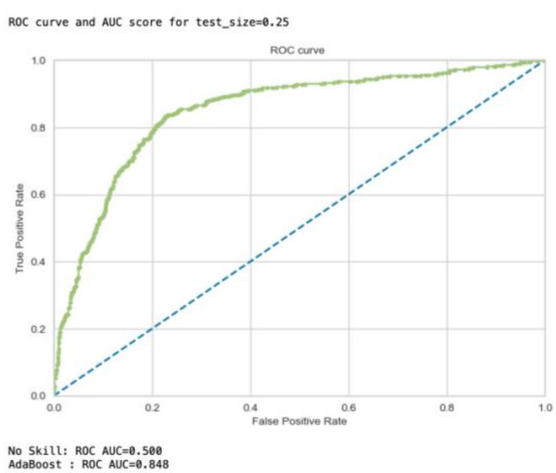
```
-----------------------------------------------
Best parameter setting with test size=0.25:  {'algorithm': 'SAMME.R', 'base_estimator': DecisionTreeClassifier(max_
depth=3), 'learning_rate': 0.2, 'n_estimators': 500}
Best accuracy score with test size=0.25:  0.795163177028983
-----------------------------------------------------------------------------------------------
```

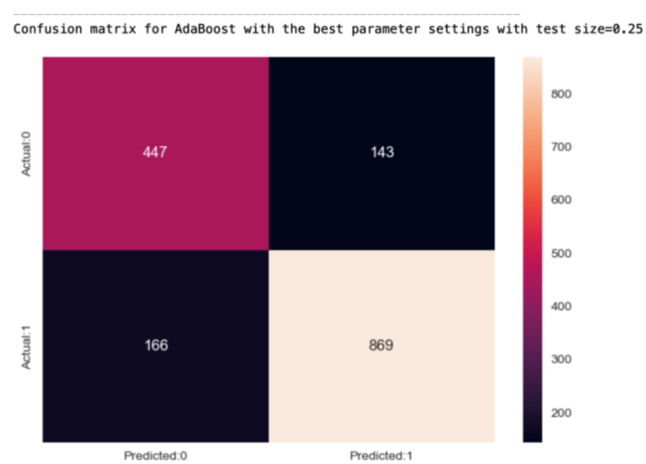
Classification report for Test and Train

```
-------------------------------------------------------------------
Classification Report for AdaBoost model with Test with test size=0.25
              precision    recall  f1-score   support

           0       0.73      0.76      0.74       590
           1       0.86      0.84      0.85      1035

    accuracy                           0.81      1625
   macro avg       0.79      0.80      0.80      1625
weighted avg       0.81      0.81      0.81      1625

-------------------------------------------------------------------
Classification Report for AdaBoost model with Train with test size=0.25
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      1794
           1       0.98      0.99      0.99      3078

    accuracy                           0.98      4872
   macro avg       0.98      0.98      0.98      4872
weighted avg       0.98      0.98      0.98      4872
```

Accuracy (Train): 0.98
Accuracy (Test): 0.81

ROC curve, AUC score and the Confusion Matrix (Average of 10 runs)





F1 score: 0.796, Precision: 0.7939, Recall: 0.7986
AUC Score: 0.848

Now, we will see how the F1 score, precision and recall of our AdaBoost changes with the change in the hyperparameter ‘n_estimators’

N_estimator=100

```
For n_estimaors= 100
With test_size=0.20 for  AdaBoost  Recall: 0.7526283468435273
With test_size=0.25 for  AdaBoost  f1 score: 0.7539037936676731
With test_size=0.25 for  AdaBoost  precision: 0.7553150818239593
-----------------------------------------------------------------
```

N_estimator=400

```
For n_estimaors= 400
With test_size=0.20 for  AdaBoost  Recall: 0.7921968394333907
With test_size=0.25 for  AdaBoost  f1 score: 0.7915563682235904
With test_size=0.25 for  AdaBoost  precision: 0.7909418801077478
-----------------------------------------------------------------
```

N_estimator=1000

```
For n_estimaors= 1000
With test_size=0.20 for  AdaBoost  Recall: 0.798722672562024
With test_size=0.25 for  AdaBoost  f1 score: 0.798940190493671
With test_size=0.25 for  AdaBoost  precision: 0.7991606478657036
-----------------------------------------------------------------
```

N_estimator=5000

```
For n_estimaors= 5000
With test_size=0.20 for  AdaBoost  Recall: 0.7896544665520346
With test_size=0.25 for  AdaBoost  f1 score: 0.7893374163330622
With test_size=0.25 for  AdaBoost  precision: 0.7890268572623762
-----------------------------------------------------------------
```

N_estimator=10000

```
For n_estimaors= 10000
With test_size=0.20 for  AdaBoost  Recall: 0.7963072136248259
With test_size=0.25 for  AdaBoost  f1 score: 0.7959829237042589
With test_size=0.25 for  AdaBoost  precision: 0.795665189484555
-----------------------------------------------------------------
```

DECISION TREE CLASSIFIER

Hyperparameter Tuning (GridSearchCV with 10-fold cross validation)

Criterion	Max Depth	Max Features	Max Leaf Node	Min Samples Leaf	Min Samples Split	Accuracy
GINI	4	sqrt	4	4	4	67.8
ENTROPY	18	Log2	12	12	6	70.2
ENTROPY	10	Sqrt	10	4	6	73
GINI	12	Sqrt	12	6	6	72.12

```
Best parameter setting with test size=0.25:  {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'sqrt', 'max
_leaf_nodes': 10, 'min_samples_leaf': 4, 'min_samples_split': 6}
Best accuracy score with test size=0.25:  0.7298881576732756
-------------------------------------------------------------------------------------
```

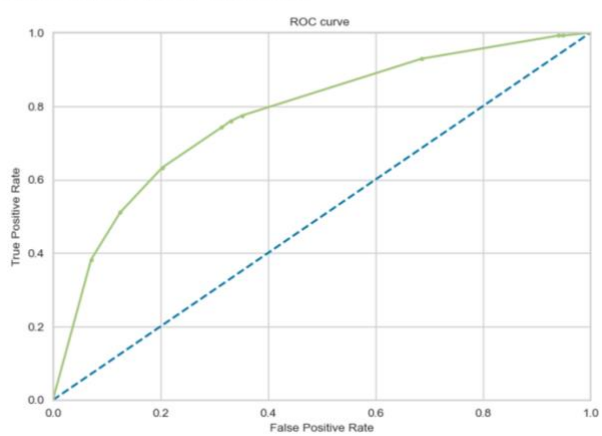
Classification report for Test and Train

```
------------------------------------------------------------------
Classification Report for DT model with Test with test size=0.25
              precision    recall  f1-score   support

           0       0.62      0.65      0.63       590
           1       0.79      0.77      0.78      1035

    accuracy                           0.73      1625
   macro avg       0.71      0.71      0.71      1625
weighted avg       0.73      0.73      0.73      1625

------------------------------------------------------------------
Classification Report for DT model with Train with test size=0.25
              precision    recall  f1-score   support

           0       0.65      0.64      0.64      1794
           1       0.79      0.80      0.79      3078

    accuracy                           0.74      4872
   macro avg       0.72      0.72      0.72      4872
weighted avg       0.74      0.74      0.74      4872
```

Accuracy (Train): 0.74
Accuracy (Test): 0.73

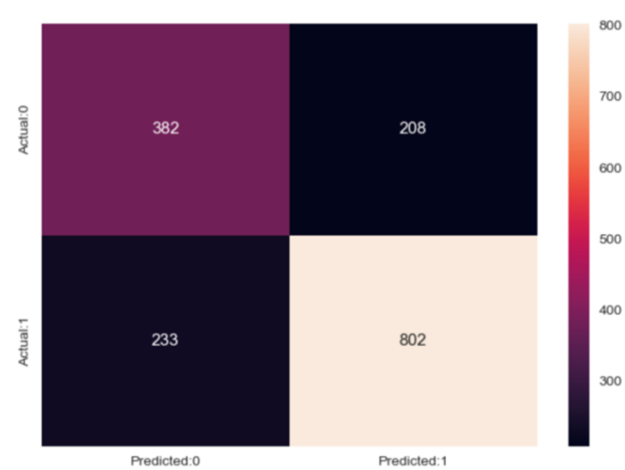
ROC curve, AUC score and the Confusion Matrix (Average of 10 runs)

ROC curve and AUC score for test_size=0.25



No Skill: ROC AUC=0.500
DT : ROC AUC=0.776

Confusion matrix for DT with the best parameter settings with test size=0.25



F1 score: 0.71, Precision: 0.71, Recall: 0.711
AUC Score: 0.776

Now, we will see how the F1 score and precision change for different depths of the decision tree.

Depth – 10

```
245]: f_p_r(dt_25,0.25,"Decision Tree")

      With test_size=0.25 for  Decision Tree  f1 score: 0.7091884872525845
      With test_size=0.25 for  Decision Tree  precision: 0.7075988086613539
```

Depth – 3

```
For max_depth= 3
With test_size=0.25 for  Decision Tree  f1 score: 0.5778824562510151
With test_size=0.25 for  Decision Tree  precision: 0.6615494333375022
-----------------------------------------------------------------
```

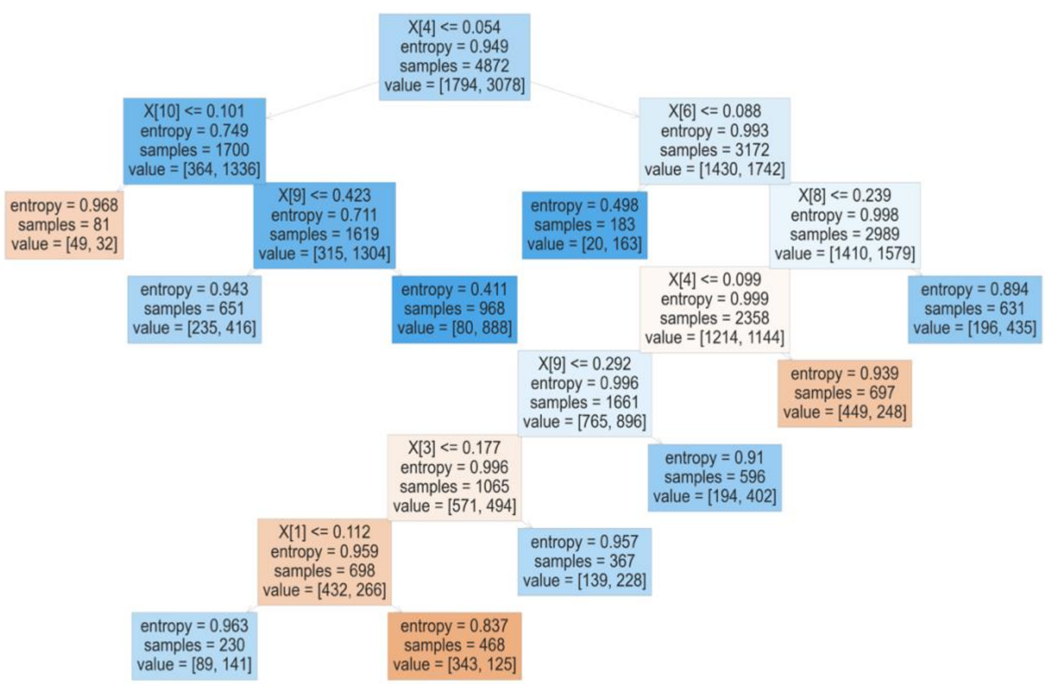
Depth – 6

```
For max_depth= 6
With test_size=0.25 for  Decision Tree  f1 score: 0.6701428053241572
With test_size=0.25 for  Decision Tree  precision: 0.6716800515731953
-----------------------------------------------------------------
```

Depth – 18

```
For max_depth= 18
With test_size=0.25 for  Decision Tree  f1 score: 0.6684285700405173
With test_size=0.25 for  Decision Tree  precision: 0.6669188575751843
-----------------------------------------------------------------
```

The final DT with the best parameter setting and test size=0.25 is given below.



GRADIENT BOOSTING

Hyperparameter Tuning (GridSearchCV with 10-fold cross validation)

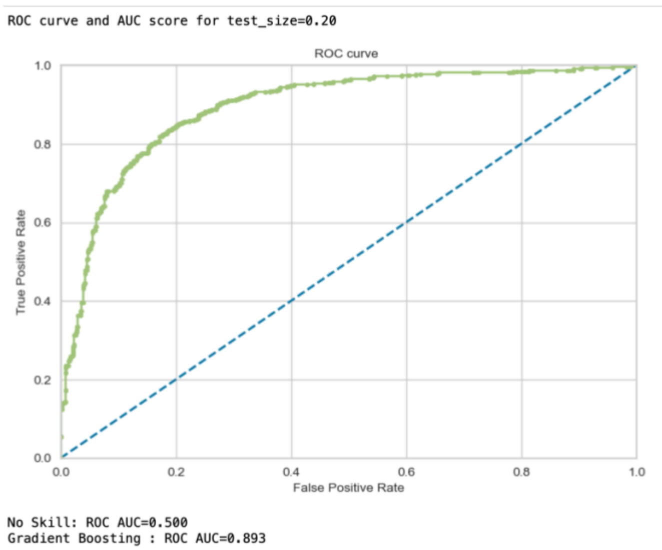
Learning Rate	Max_Depth	Min_samples_leaf	Min_samples_split	N_estimators	Accuracy
0.3	2	4	4	20	75.6
0.7	10	10	6	90	81.3
0.3	10	10	4	70	82.6
0.5	6	10	4	90	78

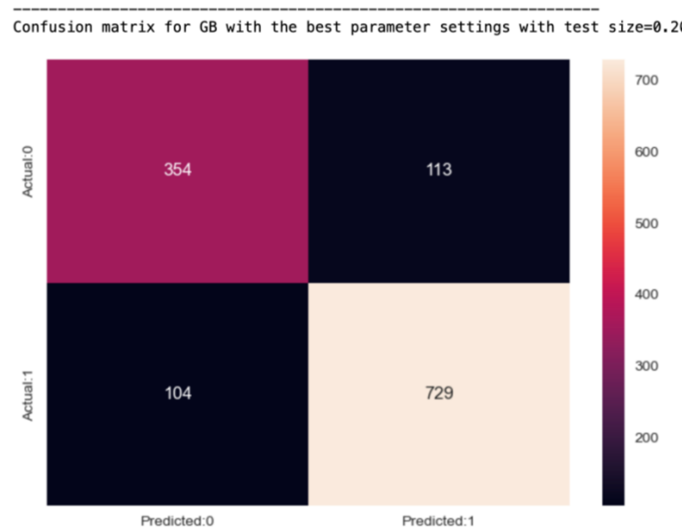
```
-----------------------------------------
Best parameter setting with test size=0.20:  {'learning_rate': 0.3, 'max_depth': 10, 'min_samples_leaf': 10, 'min_s
amples_split': 4, 'n_estimators': 70}
Best accuracy score with test size=0.20:  0.8264358233288869
---------------------------------------------------------------------------------
```

Classification report for Test and Train

```
-----------------------------------------------------------------
Classification Report for GB model with Test with test size=0.20
              precision    recall  f1-score   support

           0       0.76      0.75      0.76       467
           1       0.86      0.87      0.87       833

    accuracy                           0.83      1300
   macro avg       0.81      0.81      0.81      1300
weighted avg       0.83      0.83      0.83      1300

-----------------------------------------------------------------
Classification Report for GB model with Train with test size=0.20
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1917
           1       1.00      1.00      1.00      3280

    accuracy                           1.00      5197
   macro avg       1.00      1.00      1.00      5197
weighted avg       1.00      1.00      1.00      5197

-------------------------------------------------------------------------------------
```

Accuracy (Test): 0.83
Accuracy (Train): 1.00
Our model might or might not be overfitting. More discussion on this in the performance evaluation section.
ROC curve, AUC score and confusion matrix (Average of 10 runs)





F1 score: 0.817, Precision: 0.819, Recall: 0.8165
AUC score: 0.893

K-NEAREST NEIGHBORS

Hyperparameter Tuning (GridSearchCV with 10-fold cross validation)

N_neighbors	metric	weights	algorithm	Accuracy
1	Manhattan	Uniform	Ball-tree	78.5
3	manhattan	uniform	Ball-tree	75.9
13	manhattan	distance	Ball-tree	82.2
27	hamming	uniform	brute	68.8
29	hamming	distance	brute	74.8

```
-----------------------------------------
Best parameter setting with test size=0.25:  {'algorithm': 'ball_tree', 'metric': 'manhattan', 'n_neighbors': 15, '
weights': 'distance'}
Best accuracy score with test size=0.25:  0.8210245060086848
---------------------------------------------------------------------------------
```

Classification report of Train set and Test set

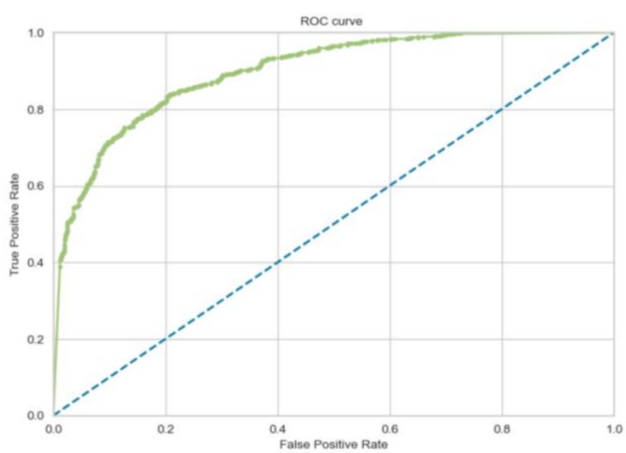
```
---------------------------------------------------------------
Classification Report for KNN model with Test with test size=0.25
              precision    recall  f1-score   support

           0       0.77      0.71      0.73       590
           1       0.84      0.88      0.86      1035

    accuracy                           0.81      1625
   macro avg       0.80      0.79      0.80      1625
weighted avg       0.81      0.81      0.81      1625

---------------------------------------------------------------
Classification Report for KNN model with Train with test size=0.25
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1794
           1       1.00      1.00      1.00      3078

    accuracy                           1.00      4872
   macro avg       1.00      1.00      1.00      4872
weighted avg       1.00      1.00      1.00      4872

-------------------------------------------------------------------------------------
```

Accuracy (Train): 1.00
Accuracy (Test): 0.82

Our model might or might not be overfitting. More discussion on this in the performance evaluation section.

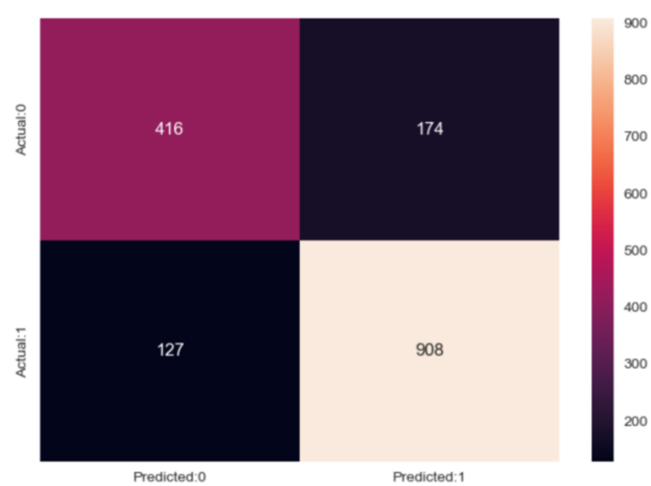
ROC curve, AUC score and Confusion matrix (Average of 10 runs)

ROC curve and AUC score for test_size=0.25



No Skill: ROC AUC=0.500
KNN : ROC AUC=0.898

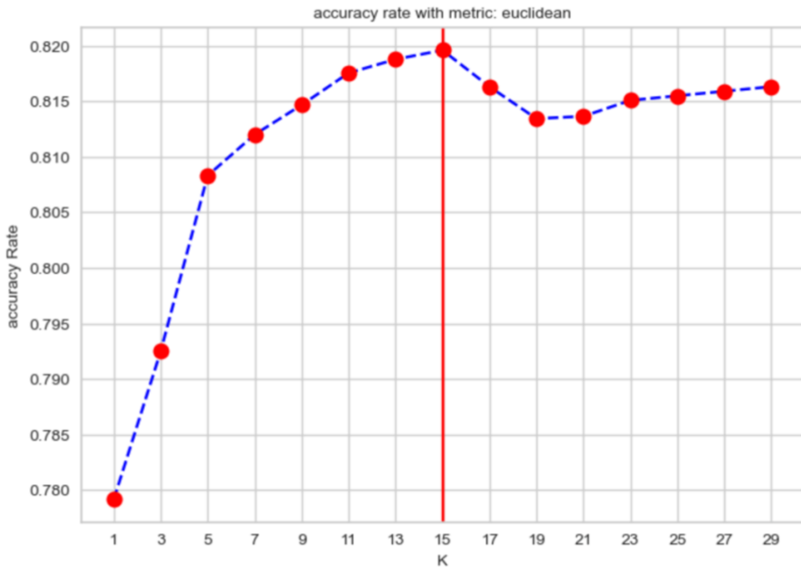
Confusion matrix for KNN with the best parameter settings with test size=0.25



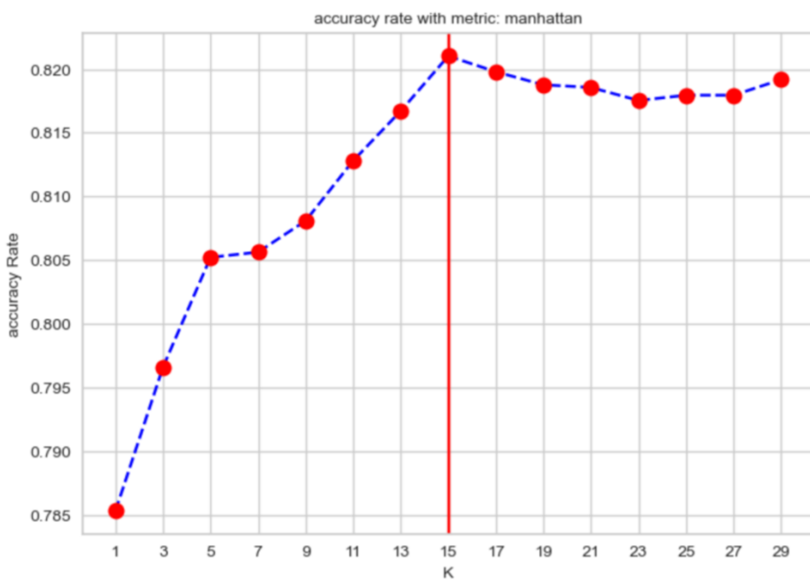
F1 score: 0.796, Precision: 0.802, Recall: 0.7911
AUC score: 0.898

Now, we will look at a plot between 'K' values and the cross-validation score with cv=10 for each of the metric

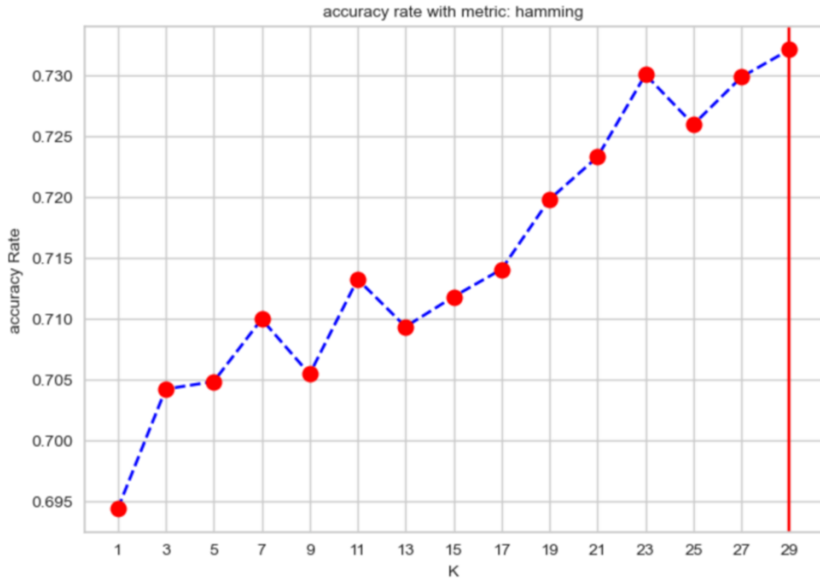
Euclidean:



Manhattan:



Hamming:



PERFORMANCE OF ALGORITHMS

:

model_name	accuracy_train	accuracy_test	difference	f1_score	precision	recall	auc_score
Logistic_Regression	0.75	0.74	0.01	0.712	0.7200	0.7000	0.805
Support_Vector_Machine	0.75	0.75	0.00	0.720	0.7300	0.7160	0.811
AdaBoost_Classifier	0.98	0.81	0.17	0.796	0.7939	0.7986	0.848
Gradient_Boosting	1.00	0.83	0.17	0.817	0.8190	0.8165	0.893
K_Nearest_Neighbors	1.00	0.82	0.18	0.796	0.8020	0.7911	0.899
Decision_Trees	0.74	0.73	0.01	0.710	0.7100	0.7110	0.776

Logistic Regression – LR performed decently well with the dataset and gave us an accuracy of 74% on the test dataset. With a precision of 0.72, it can be said that the model performed fairly in predicting the positive predictions. An F1 score of 0.71 gives us a balanced tradeoff between the precision and recall. An AUC score of 0.80 is good, which means that there is 80% chance that the model will be able to predict between a ‘low’ and ‘high’ quality wine.

Support Vector Machine – As expected, SVM performed decent with the dataset, and gave an accuracy of 75% on the test dataset. With a precision of 0.73, it can be said that our model performed fairly in predicting the positive predictions. An F1 score of 0.72 gives a balanced tradeoff between the precision and recall. An AUC score of 0.811 is good enough, which means that there is 81.1% chance that our model will be able to predict between a ‘low’ and ‘high’ quality wine. The F1 score and the precision values for different values of C (screenshots attached) were calculated, upon calculation, it was observed that as the C value went less than 1, the model started underfitting and gave a very low f1 score and precision values.

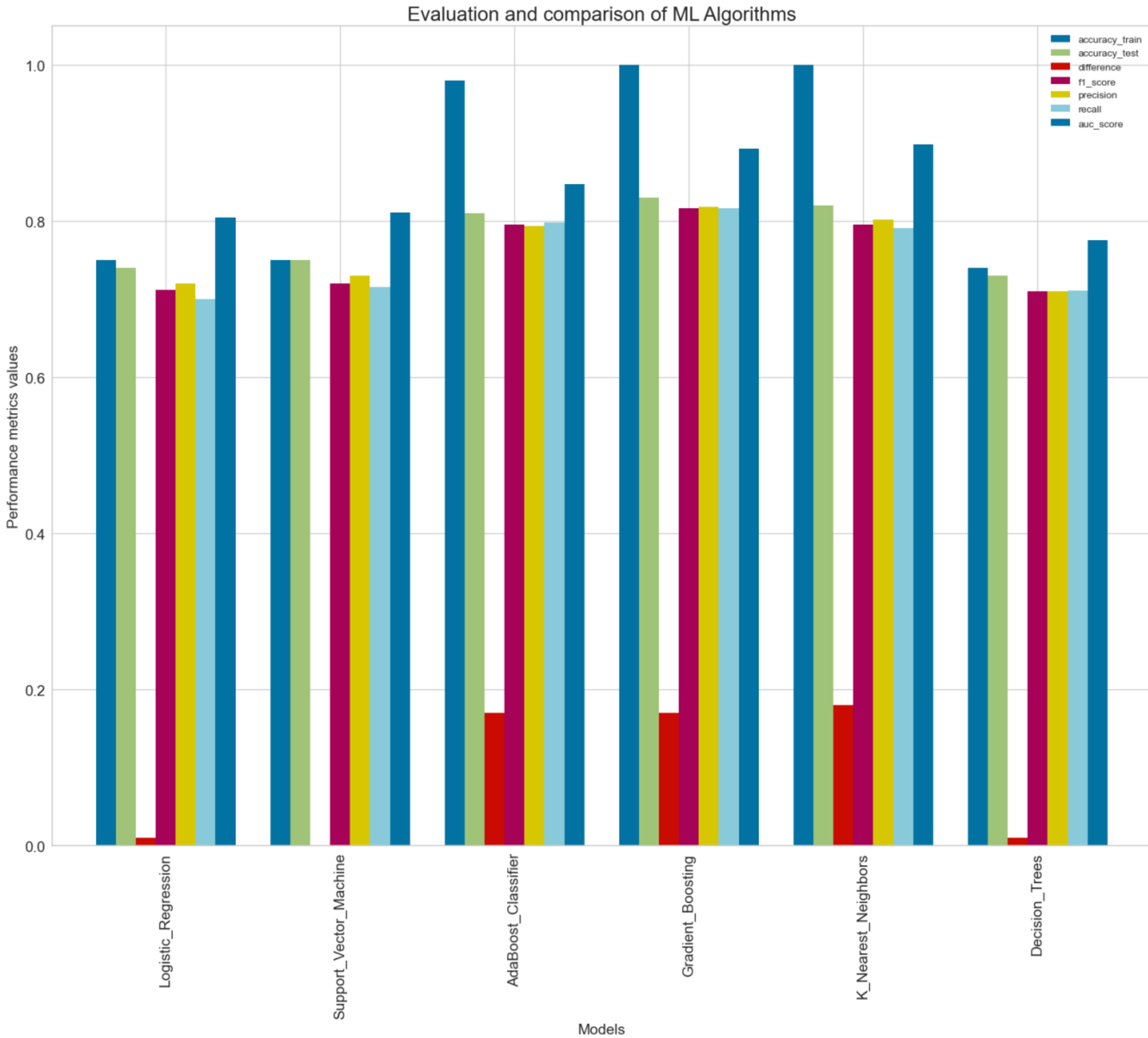
AdaBoost Classifier – When the AdaBoost model was implemented with three different train-test splits, the model was consistently giving a training score and test score in the range of 0.98-1 and 0.79-0.81, respectively for each of the combinations. Now, there may be two reasons for the above results. Firstly, I feel that this may be due to presence of some data points near the decision boundary which might be resulting in a high training score and an accordingly good test score as well. Secondly, since AdaBoost starts overfitting if the data is too noisy or contains outliers, this may be a reason for the results which we got. I think that future analysis of this topic might give us a concrete reason.

Gradient Boosting – Gradient Boosting is the algorithm which took the maximum time to complete its computation process which is due to wide range of hyperparameter combination with a cv=10. Coming to the performance, since Gradient Boosting is somewhat less prone to overfitting, I think that similar to AdaBoost, the GB model might be giving us a high training score due to presence of some data points near the decision boundary. As a result of the mentioned reason, we got a train score and test score as 1 and 0.83 respectively.

K-Nearest Neighbors – With KNN, I ran the model with three different test sizes, and the model was overfitting in each of the combinations. I believe that this may be due to presence of outliers in the dataset. KNN is very sensitive to outliers. One point to note is that – I didn’t use ‘weights’ as a hyperparameter in the initial runs and was getting the K value to be 1 as the best parameter settings. However, as soon as I included ‘weights’, the accuracy shot up and the K value came to be as 15. Hence, I feel that the weight distribution plays an important role in KNN.

Decision Trees – Decision Trees performed as good as SVM. I was able to observe the change in the F1 score and the precision values with the change in the ‘max_depth’ (screenshots attached). The Decision tree was overfit when I kept a large value as the max_depth. Overall, an accuracy of 74% on the train set and 73% on the test set is decent enough to trust the model.

The following unstacked bar plot shows the difference between the performance metrics of each algorithm.



CONCLUSION

Based on the individual performances of the algorithm, for predicting the quality of wine, I feel that each algorithm has its own pitfalls. SVM with a training score of 75% and testing score of 74% seems the ideal choice for our problem statement. However, AdaBoost and Gradient Boosting, are two of the most powerful ML algorithms. Based on the performance evaluation in the last section, I am, maybe inclined towards choosing AdaBoost or Gradient Boosting for the prediction task.

My final verdict is that I would be going ahead with choosing SVM as the best model (as of now) to predict the quality of wine. However, I would like to do further research on my AdaBoost and Gradient Boosting Model as my next project and see if both the boosting algorithms fair better in terms of performance.

REFERENCE

1. https://www.kaggle.com/datasets/rajyellow46/wine-quality

2. Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.