A
Report on
Classification of
Wine-Quality Dataset

Submitted By: -

Rajat Mahajan	Roll No.: - 41354
Manav Shah	Roll No.: - 41356
Rohan Maniyar	Roll No.: - 41358

Under the Guidance of
Prof. Bhushan Zope

Pune Institute of Computer Technology
Dhankawadi, Pune – 411043

Table of Contents

Problem Statement

Consider a labeled dataset belonging to an application domain. Apply suitable data preprocessing steps such as handling of null values, data reduction, discretization. For prediction of class labels of given data instances, build classifier models using different techniques (minimum 3), analyze the confusion matrix and compare these models. Also apply cross validation while preparing the training and testing datasets.

Objective

The objectives of this project are as follows

1.To experiment with different classification methods to see which yields the highest accuracy.

2.To determine which features are the most indicative of a good quality wine.

H/W and S/W Requirement

- Core i3 and above processor
- 8GB RAM and 512GB HDD
- Stable internet
- Google account for Google Colab
- Python Libraries
- Wine Quality Dataset

Data Introduction

In this dataset there are specifically red wine variants of Poteguese "Vinho Verde" wine. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.). For more information, read [Cortez et al., 2009].

In the data set, there 1599 different wine as row data and 12 features as columns. Furthermore, there is no nun value to deal with it and all values are numeric means that input values are float and only output value is integer.

For this project, we used UCI ML Repository’s Wine Quality dataset to build various classification models to predict whether a particular red wine is “good quality” or not. Each wine in this dataset is given a “quality” score between 0 and 10. For the purpose of this project, we converted the output to a binary output where each wine is either “good quality” (a score of 7 or higher) or not (a score below 7). The quality of a wine is determined by 11 input variables:

1.Fixed acidity

2.Volatile acidity

3.Citric acid

4.Residual sugar

5.Chlorides

6.Free sulfur dioxide

7.Total sulfur dioxide

8.Density

9.pH

10.Sulfates

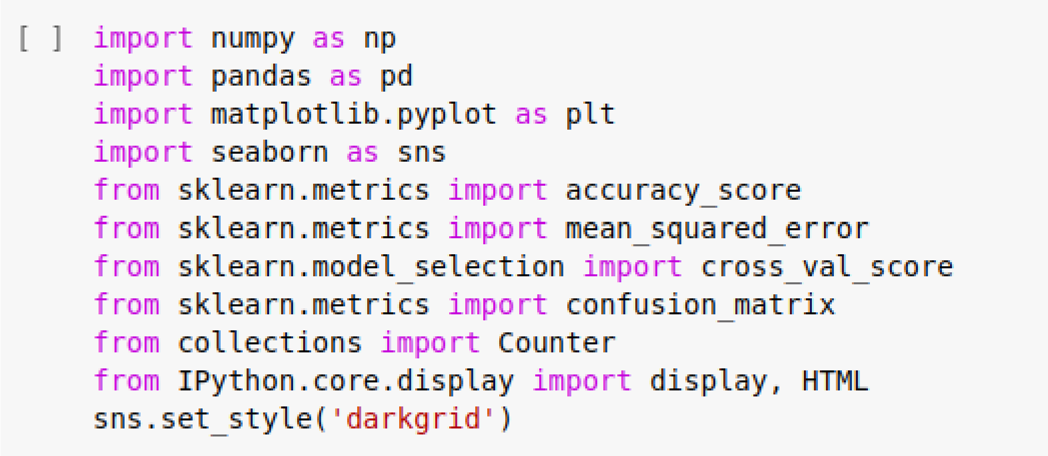
11.Alcohol

12.Quality

Data Processing and Visualization

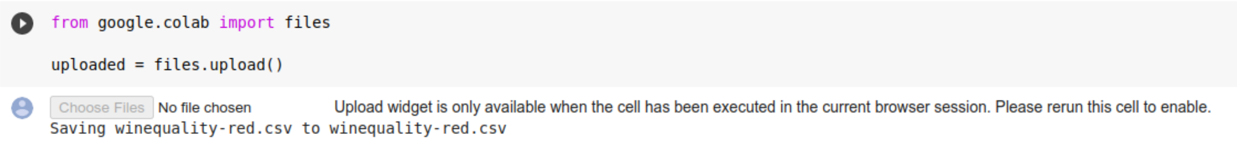
First, we imported all of the relevant libraries that we'll be using as well as the data itself.

- Importing Libraries



```
[ ]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.metrics import accuracy_score
     from sklearn.metrics import mean_squared_error
     from sklearn.model_selection import cross_val_score
     from sklearn.metrics import confusion_matrix
     from collections import Counter
     from IPython.core.display import display, HTML
     sns.set_style('darkgrid')
```

- Reading Data



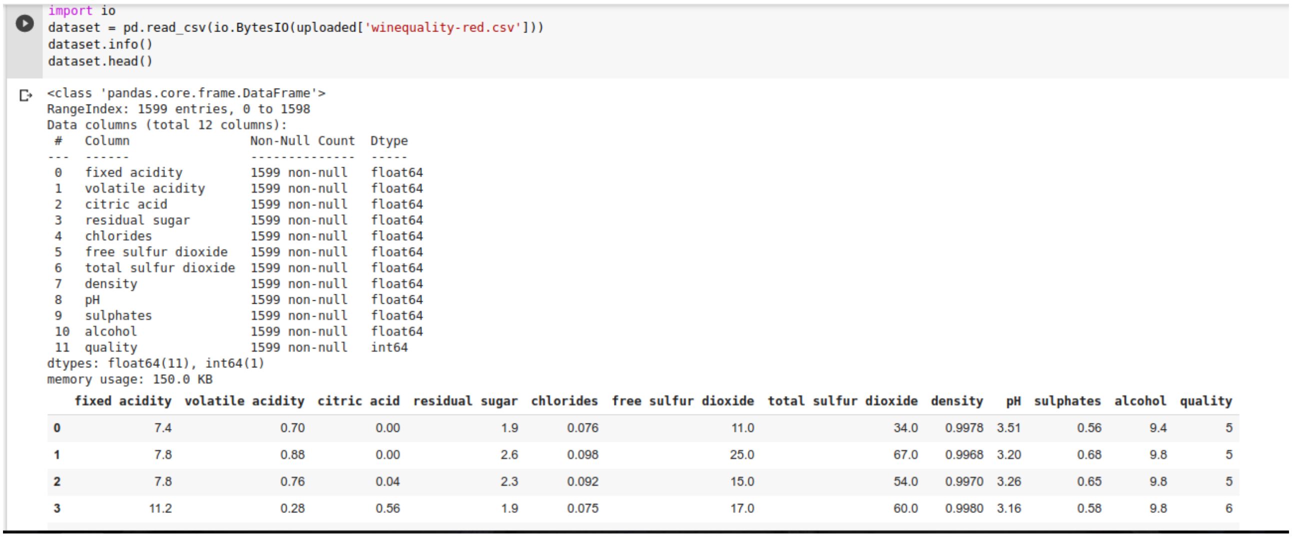
```
from google.colab import files

uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving winequality-red.csv to winequality-red.csv
```

- Understanding Data

Next, we wanted to get a better idea of what we were working with.



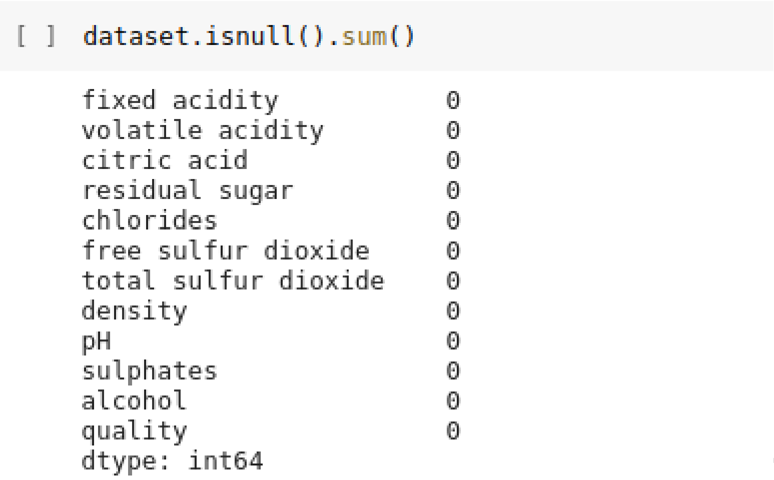
```
import io
dataset = pd.read_csv(io.BytesIO(uploaded['winequality-red.csv']))
dataset.info()
dataset.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6

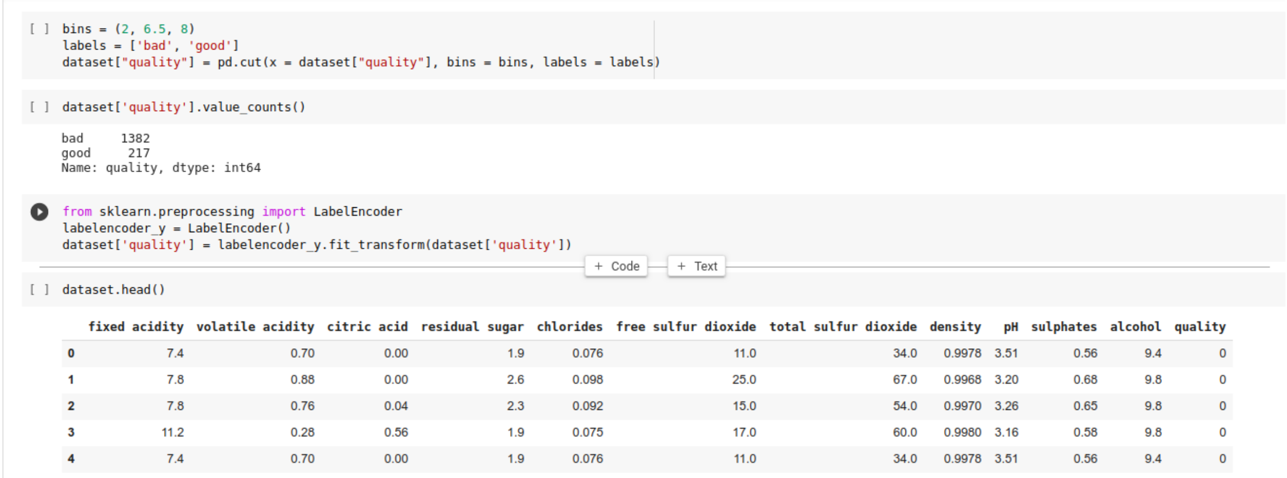
There are a total of 1599 rows and 12 columns. The data looks very clean by looking at the first five rows, but we still wanted to make sure that there were no missing values.

- Missing Values



```
[ ]  dataset.isnull().sum()

     fixed acidity           0
     volatile acidity        0
     citric acid             0
     residual sugar          0
     chlorides               0
     free sulfur dioxide     0
     total sulfur dioxide    0
     density                 0
     pH                      0
     sulphates               0
     alcohol                 0
     quality                 0
     dtype: int64
```

This is a very beginner-friendly dataset. we did not have to deal with any missing values, and there isn't much flexibility to conduct some feature engineering given these variables. Next, we wanted to explore our data a little bit more.



```
[ ]  bins = (2, 6.5, 8)
     labels = ['bad', 'good']
     dataset["quality"] = pd.cut(x = dataset["quality"], bins = bins, labels = labels)
```

```
[ ]  dataset['quality'].value_counts()

     bad     1382
     good     217
     Name: quality, dtype: int64
```

```
     from sklearn.preprocessing import LabelEncoder
     labelencoder_y = LabelEncoder()
     dataset['quality'] = labelencoder_y.fit_transform(dataset['quality'])
```

```
[ ]  dataset.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	0
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0

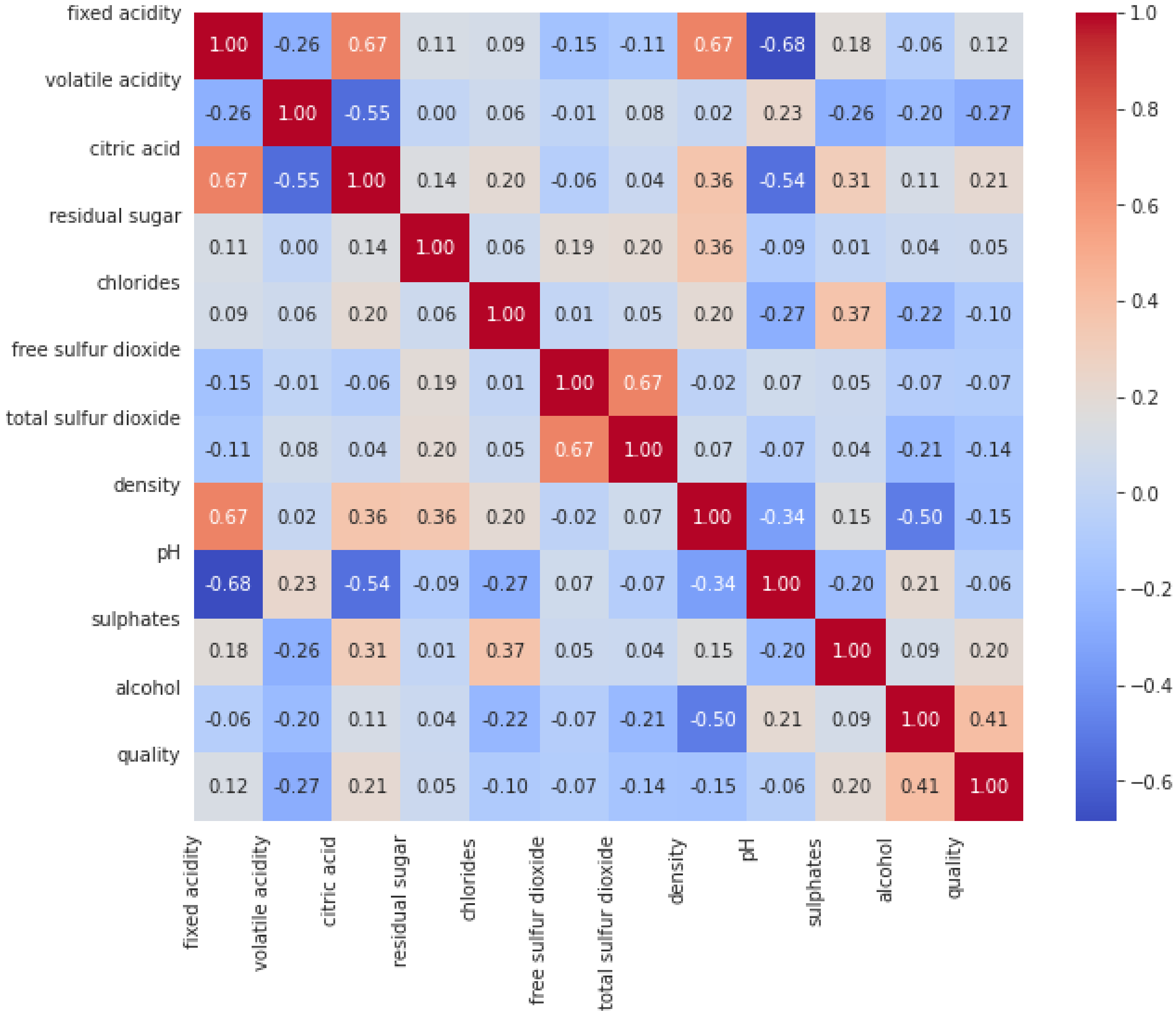
Data Correlation

This data set has many different features and it is important to understand relationship between these in order to analyze dataset better. For that reason, correlation map helps to understand these relations in a single representation. Correlation map is made by calculating the covariance of each features with respect to others, then each covariance value is divided by standard deviation of each variables and get results between **-1, 0, 1**.

-1 means: There is a negative relationship between dependent and independent variables .

0 means: There is no relationship between dependent and independent variables .

1 means: There is a positive relationship between dependent and independent variables . According to these information, it can be made a good analyze about dataset and columns.



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.00	-0.26	0.67	0.11	0.09	-0.15	-0.11	0.67	-0.68	0.18	-0.06	0.12
volatile acidity	-0.26	1.00	-0.55	0.00	0.06	-0.01	0.08	0.02	0.23	-0.26	-0.20	-0.27
citric acid	0.67	-0.55	1.00	0.14	0.20	-0.06	0.04	0.36	-0.54	0.31	0.11	0.21
residual sugar	0.11	0.00	0.14	1.00	0.06	0.19	0.20	0.36	-0.09	0.01	0.04	0.05
chlorides	0.09	0.06	0.20	0.06	1.00	0.01	0.05	0.20	-0.27	0.37	-0.22	-0.10
free sulfur dioxide	-0.15	-0.01	-0.06	0.19	0.01	1.00	0.67	-0.02	0.07	0.05	-0.07	-0.07
total sulfur dioxide	-0.11	0.08	0.04	0.20	0.05	0.67	1.00	0.07	-0.07	0.04	-0.21	-0.14
density	0.67	0.02	0.36	0.36	0.20	-0.02	0.07	1.00	-0.34	0.15	-0.50	-0.15
pH	-0.68	0.23	-0.54	-0.09	-0.27	0.07	-0.07	-0.34	1.00	-0.20	0.21	-0.06
sulphates	0.18	-0.26	0.31	0.01	0.37	0.05	0.04	0.15	-0.20	1.00	0.09	0.20
alcohol	-0.06	-0.20	0.11	0.04	-0.22	-0.07	-0.21	-0.50	0.21	0.09	1.00	0.41
quality	0.12	-0.27	0.21	0.05	-0.10	-0.07	-0.14	-0.15	-0.06	0.20	0.41	1.00

According to figure in above;

- ➔ Quality has a (+)positive relationship between alcohol
- ➔ Quality has a (-)negative weak relationship between volitile_acidicity
- ➔ Quality has almost no relationship between residual_sugar, free_sulfur_dioxide, and pH.(corr =~ 0)
- ➔ Alcohol has a (+)positive relationship between quality and weakly pH
- ➔ Alcohol has a (-)negative relationship between density
- ➔ Alcohol has almost no relationship between fixed_acidicity, residual_sugar, free_sulfur_dioxide, sulphates
- ➔ Volitile_acidicity has a weak (+)positive relationship between pH.
- ➔ Volitile_acidicity has a strong (-)negative relationship between citric_acid
- ➔ Volitile_acidicity has weak (-)negative relationship between fixed_acidicity and sulphates
- ➔ Volitile_acidicity has almost no relationship between residual_sugar, chlorides, free_sulfur_dioxide, total_sulfur_dioxide, density
- ➔ Density has (+)positive relationship between fixed_acidicity
- ➔ Density has (-)negative relationship between density

- ➔ Density has almost no relationship between volitile_acidicity, free_sulfur_dioxide, total_sulfur_dioxide
- ➔ Citric_acid has (+)positive relationship between fixed_acidicity
- ➔ Citric_acid has (-)negative relationship between volitile_acidicity, pH
- ➔ Citric_acid has almost no relationship between residual_sugar, free_sulfur_dioxide, total_sulfur_dioxide

Split data

Next we split the data into a training and test set so that we could cross-validate our models and determine their effectiveness.

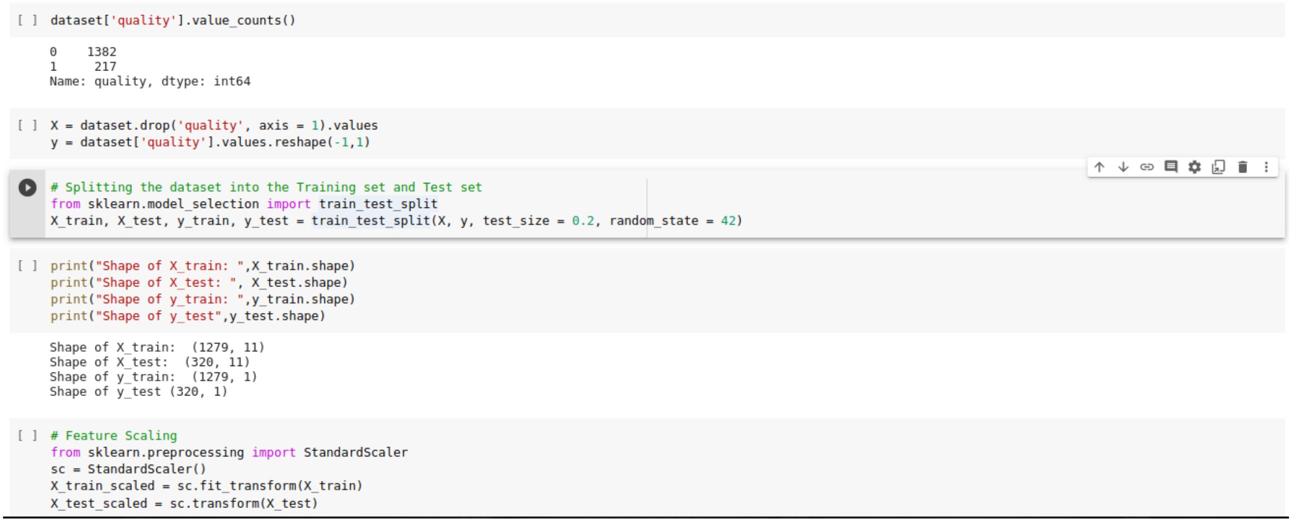
Going back to our objective, we wanted to compare the effectiveness of different classification techniques, so we needed to change the output variable to a binary output.

For this problem, we defined a bottle of wine as 'good quality' if it had a quality score of 7 or higher, and if it had a score of less than 7, it was deemed 'bad quality'.

Once we converted the output variable to a binary output, we separated our feature variables (X) and the target variable (y) into separate dataframes.

Proportion of Good vs Bad Wines

We wanted to make sure that there was a reasonable number of good quality wines. Based on the results below, it seemed like a fair enough number. In some applications, resampling may be required if the data was extremely imbalanced, but we assumed that it was okay for this purpose.



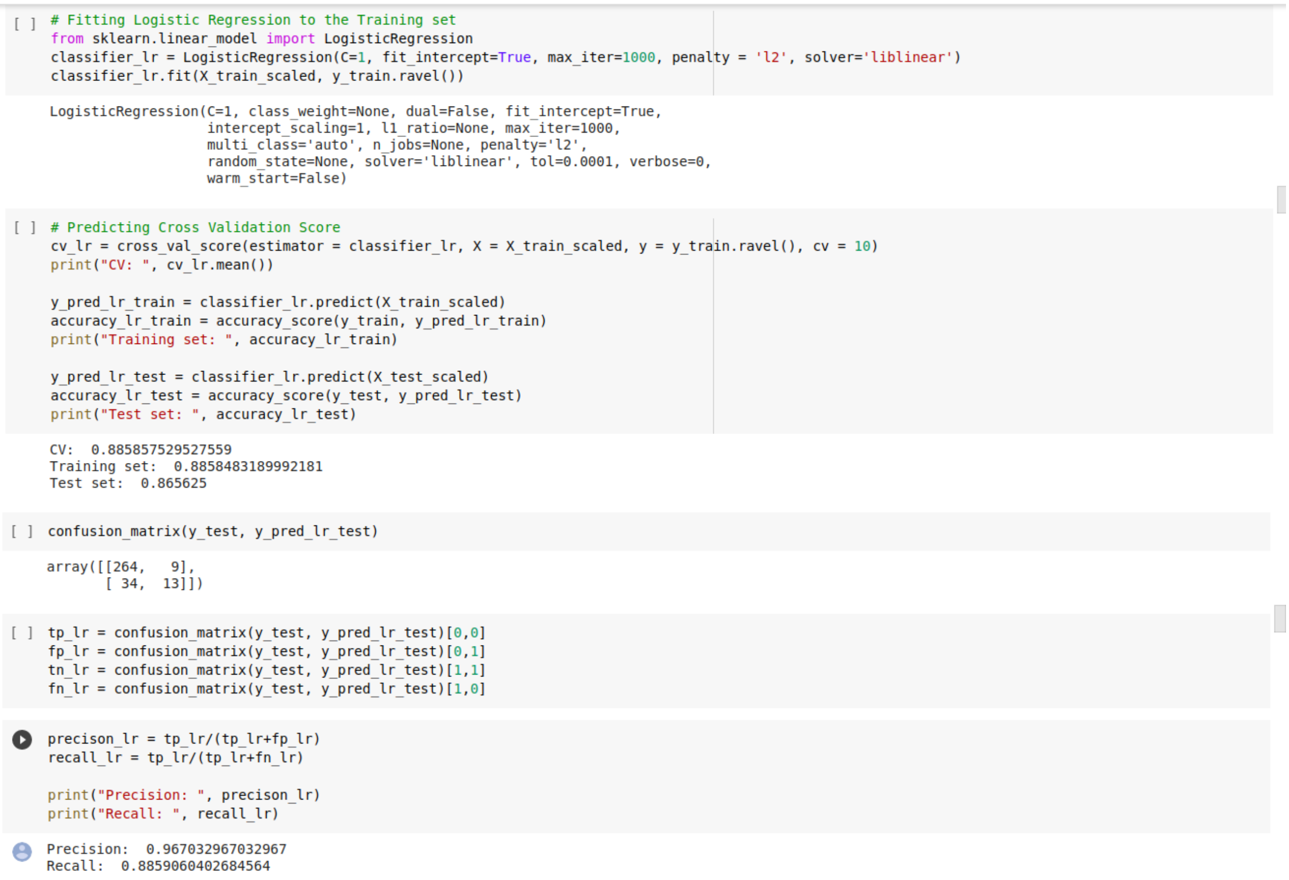
```
[ ] dataset['quality'].value_counts()

    0    1382
    1     217
    Name: quality, dtype: int64

[ ] X = dataset.drop('quality', axis = 1).values
    y = dataset['quality'].values.reshape(-1,1)

    # Splitting the dataset into the Training set and Test set
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

[ ] print("Shape of X_train: ",X_train.shape)
    print("Shape of X_test: ", X_test.shape)
    print("Shape of y_train: ",y_train.shape)
    print("Shape of y_test",y_test.shape)

    Shape of X_train:  (1279, 11)
    Shape of X_test:  (320, 11)
    Shape of y_train:  (1279, 1)
    Shape of y_test (320, 1)

[ ] # Feature Scaling
    from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train_scaled = sc.fit_transform(X_train)
    X_test_scaled = sc.transform(X_test)
```

Classification Algorithms: -

Logistic Regression

The logistic regression is a predictive analysis of statistical method. Logistic regression is analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. For this data set, firstly using ordinal logistic regression then it is better to using binary logistic regression with modified dataset. Formulation of the logistic regression:

$$P = \frac{e^{a+bX}}{1+e^{a+bX}}$$



```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier_lr = LogisticRegression(C=1, fit_intercept=True, max_iter=1000, penalty = 'l2', solver='liblinear')
classifier_lr.fit(X_train_scaled, y_train.ravel())
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
# Predicting Cross Validation Score
cv_lr = cross_val_score(estimator = classifier_lr, X = X_train_scaled, y = y_train.ravel(), cv = 10)
print("CV: ", cv_lr.mean())

y_pred_lr_train = classifier_lr.predict(X_train_scaled)
accuracy_lr_train = accuracy_score(y_train, y_pred_lr_train)
print("Training set: ", accuracy_lr_train)

y_pred_lr_test = classifier_lr.predict(X_test_scaled)
accuracy_lr_test = accuracy_score(y_test, y_pred_lr_test)
print("Test set: ", accuracy_lr_test)
```

```
CV:  0.885857529527559
Training set:  0.8858483189992181
Test set:  0.865625
```

```
confusion_matrix(y_test, y_pred_lr_test)
```

```
array([[264,   9],
       [ 34,  13]])
```

```
tp_lr = confusion_matrix(y_test, y_pred_lr_test)[0,0]
fp_lr = confusion_matrix(y_test, y_pred_lr_test)[0,1]
tn_lr = confusion_matrix(y_test, y_pred_lr_test)[1,1]
fn_lr = confusion_matrix(y_test, y_pred_lr_test)[1,0]
```

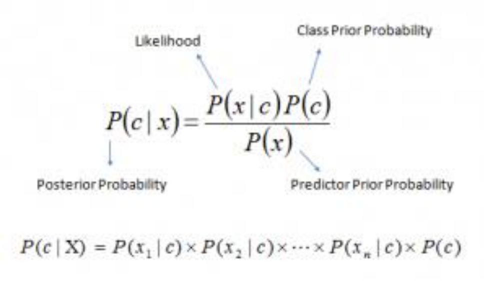
```
precison_lr = tp_lr/(tp_lr+fp_lr)
recall_lr = tp_lr/(tp_lr+fn_lr)

print("Precision: ", precison_lr)
print("Recall: ", recall_lr)
```

```
Precision:  0.967032967032967
Recall:  0.8859060402684564
```

Naive Bayes

It is a classification technique based on Bayes’ Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

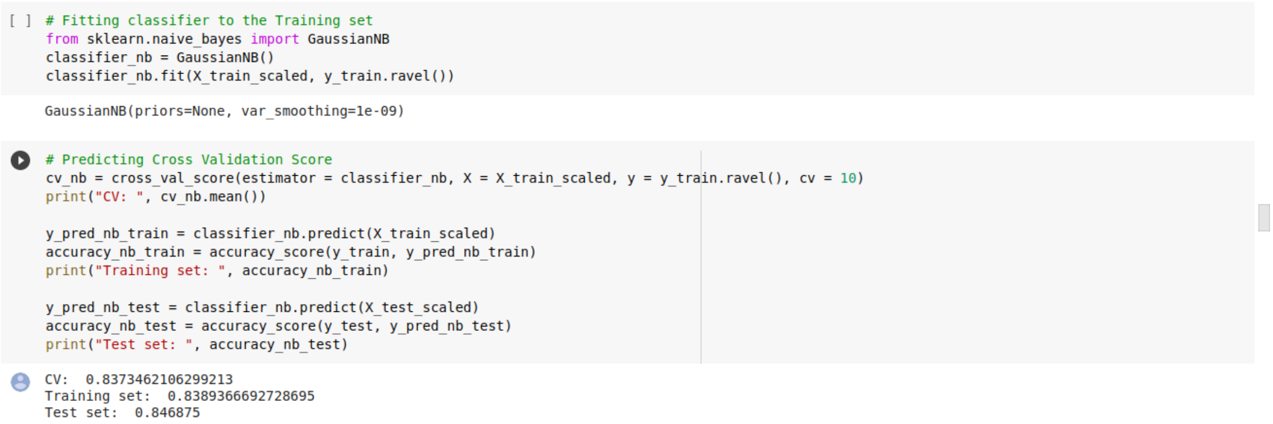


$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Above,

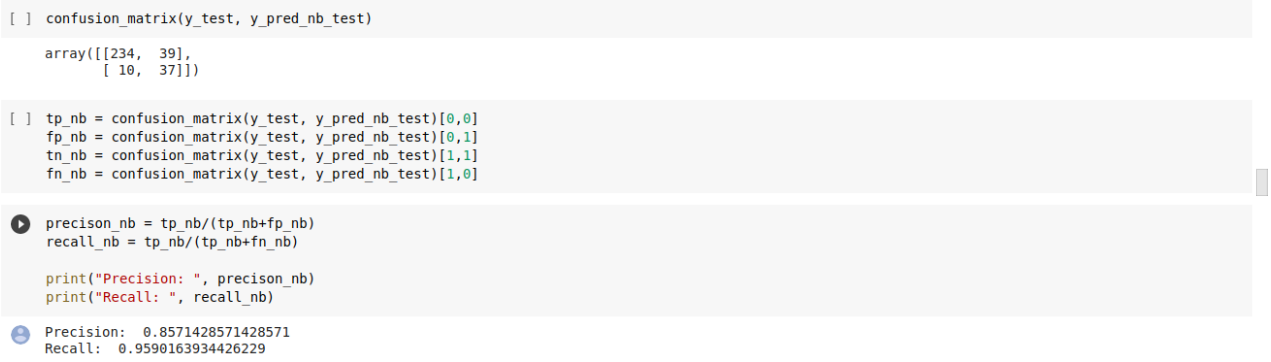
- *P*(*c*|*x*) is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
- *P*(*x*) is the prior probability of *predictor*.



```
[ ] # Fitting classifier to the Training set
    from sklearn.naive_bayes import GaussianNB
    classifier_nb = GaussianNB()
    classifier_nb.fit(X_train_scaled, y_train.ravel())

    GaussianNB(priors=None, var_smoothing=1e-09)
```

```
# Predicting Cross Validation Score
cv_nb = cross_val_score(estimator = classifier_nb, X = X_train_scaled, y = y_train.ravel(), cv = 10)
print("CV: ", cv_nb.mean())

y_pred_nb_train = classifier_nb.predict(X_train_scaled)
accuracy_nb_train = accuracy_score(y_train, y_pred_nb_train)
print("Training set: ", accuracy_nb_train)

y_pred_nb_test = classifier_nb.predict(X_test_scaled)
accuracy_nb_test = accuracy_score(y_test, y_pred_nb_test)
print("Test set: ", accuracy_nb_test)
```

```
CV:  0.8373462106299213
Training set:  0.8389366692728695
Test set:  0.846875
```



```
[ ] confusion_matrix(y_test, y_pred_nb_test)

    array([[234,  39],
           [ 10,  37]])
```

```
[ ] tp_nb = confusion_matrix(y_test, y_pred_nb_test)[0,0]
    fp_nb = confusion_matrix(y_test, y_pred_nb_test)[0,1]
    tn_nb = confusion_matrix(y_test, y_pred_nb_test)[1,1]
    fn_nb = confusion_matrix(y_test, y_pred_nb_test)[1,0]
```

```
precison_nb = tp_nb/(tp_nb+fp_nb)
recall_nb = tp_nb/(tp_nb+fn_nb)

print("Precision: ", precison_nb)
print("Recall: ", recall_nb)
```

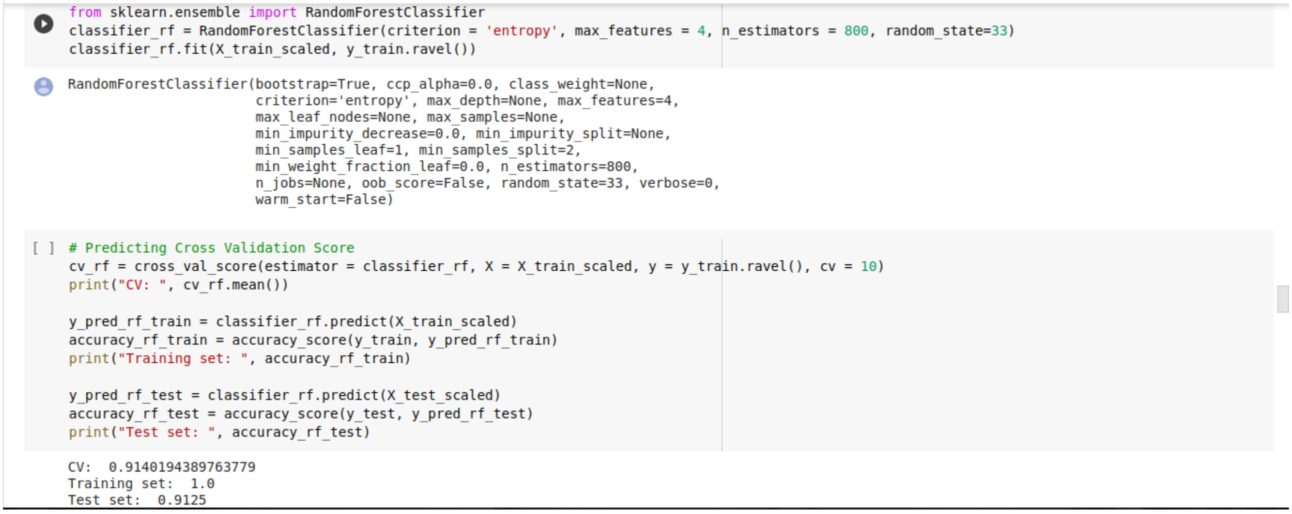
```
Precision:  0.8571428571428571
Recall:  0.9590163934426229
```

Random Forest

Random forest --> baging pf the decision tree Random forests construct many individual decision trees at training and it uses the simplicity of decision trees with flexibility resulting in improvement the accuracy. Predictions from all trees are pooled to make the final prediction; the mode of the classes for classification or the mean prediction for regression. As they use a collection of results to make a final decision. Random forest algorithm contains many variables, and many categorical variables with a large number of class labels. It gives results using data sets that show a loss or unbalanced distribution. When new trees are added into the random forest, algorithm updates itself with decreasing the loss by eliminating noises. Bootstraping the sample data(creating some mini sample dataset with less variable), then calcuating with regression to the gini then pick the highly correlated, therefore first 2-3 split will be the same because of the central limit theorem because the variance of the sum is decreasing. Formula is: -

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (fi - yi)^2$$

Where *N* is the number of data points,
fi is the value returned by the model and
yi is the actual value for data point *i*.

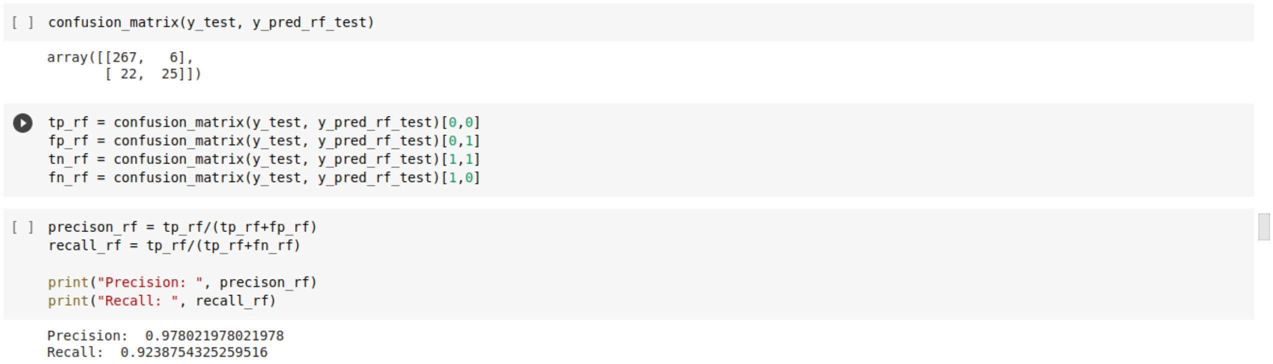


```
from sklearn.ensemble import RandomForestClassifier
classifier_rf = RandomForestClassifier(criterion = 'entropy', max_features = 4, n_estimators = 800, random_state=33)
classifier_rf.fit(X_train_scaled, y_train.ravel())
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features=4,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=800,
                       n_jobs=None, oob_score=False, random_state=33, verbose=0,
                       warm_start=False)
```

```
[ ] # Predicting Cross Validation Score
    cv_rf = cross_val_score(estimator = classifier_rf, X = X_train_scaled, y = y_train.ravel(), cv = 10)
    print("CV: ", cv_rf.mean())

    y_pred_rf_train = classifier_rf.predict(X_train_scaled)
    accuracy_rf_train = accuracy_score(y_train, y_pred_rf_train)
    print("Training set: ", accuracy_rf_train)

    y_pred_rf_test = classifier_rf.predict(X_test_scaled)
    accuracy_rf_test = accuracy_score(y_test, y_pred_rf_test)
    print("Test set: ", accuracy_rf_test)
```

```
CV:  0.9140194389763779
Training set:  1.0
Test set:  0.9125
```



```
[ ] confusion_matrix(y_test, y_pred_rf_test)
```

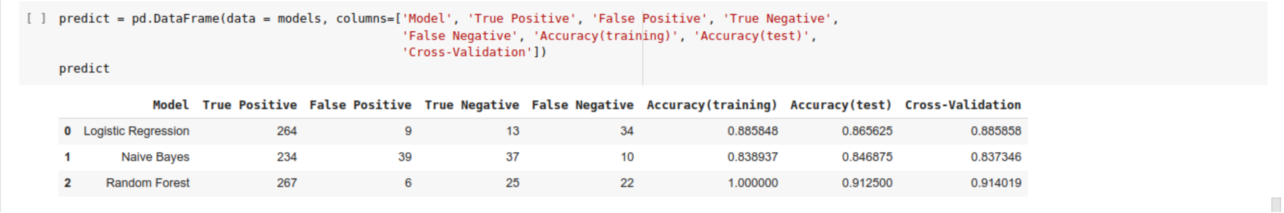
```
array([[267,   6],
       [ 22,  25]])
```

```
tp_rf = confusion_matrix(y_test, y_pred_rf_test)[0,0]
fp_rf = confusion_matrix(y_test, y_pred_rf_test)[0,1]
tn_rf = confusion_matrix(y_test, y_pred_rf_test)[1,1]
fn_rf = confusion_matrix(y_test, y_pred_rf_test)[1,0]
```

```
[ ] precison_rf = tp_rf/(tp_rf+fp_rf)
    recall_rf = tp_rf/(tp_rf+fn_rf)

    print("Precision: ", precison_rf)
    print("Recall: ", recall_rf)
```

```
Precision:  0.978021978021978
Recall:  0.9238754325259516
```

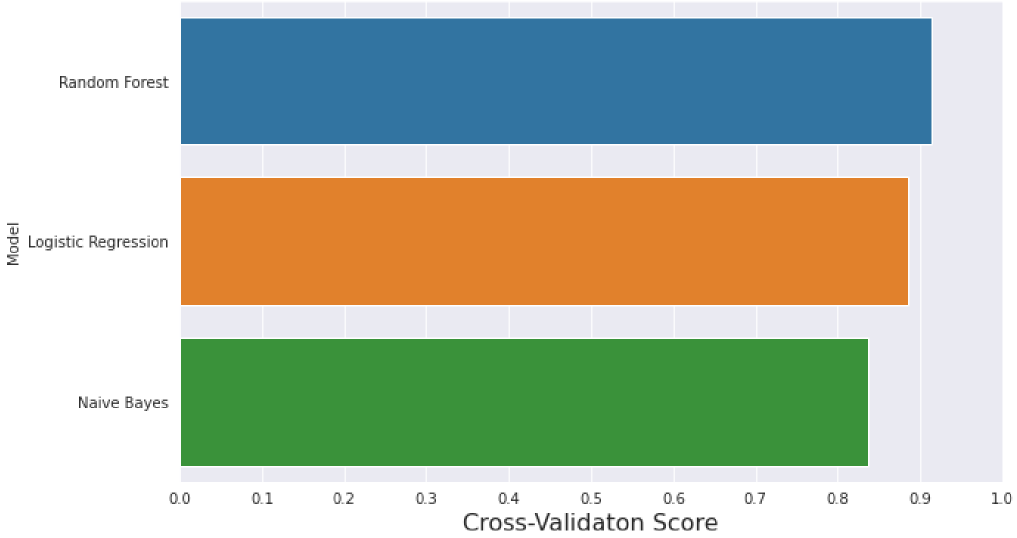
Measuring Error among the classifiers

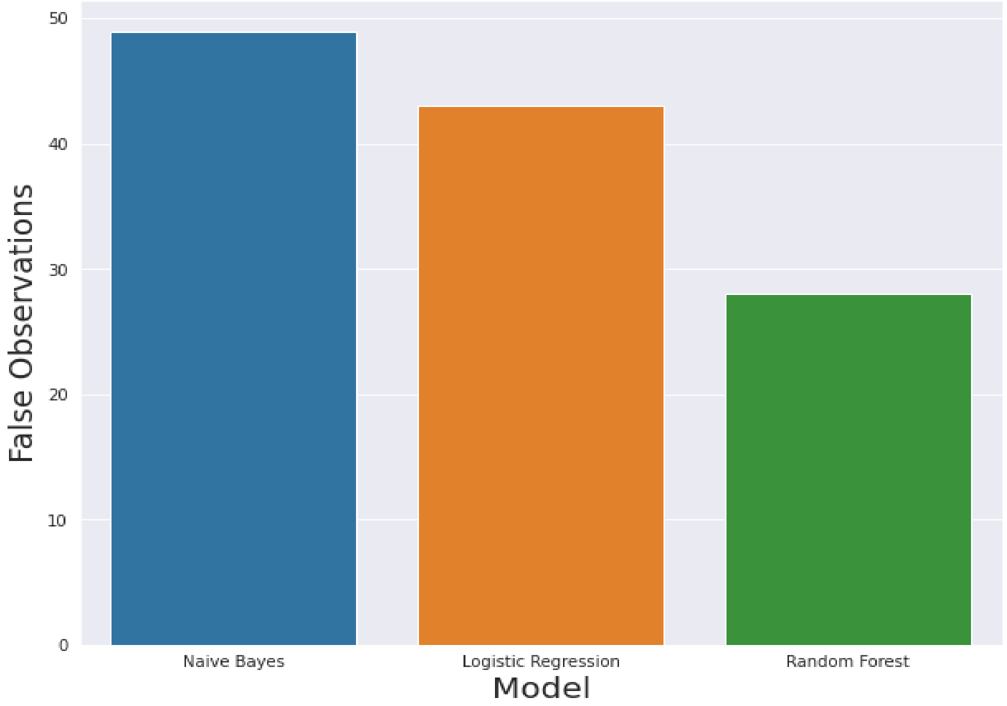


```
[ ] predict = pd.DataFrame(data = models, columns=['Model', 'True Positive', 'False Positive', 'True Negative',
                                                    'False Negative', 'Accuracy(training)', 'Accuracy(test)',
                                                    'Cross-Validation'])

    predict
```

	Model	True Positive	False Positive	True Negative	False Negative	Accuracy(training)	Accuracy(test)	Cross-Validation
0	Logistic Regression	264	9	13	34	0.885848	0.865625	0.885858
1	Naive Bayes	234	39	37	10	0.838937	0.846875	0.837346
2	Random Forest	267	6	25	22	1.000000	0.912500	0.914019

- ## Cross Validation Score

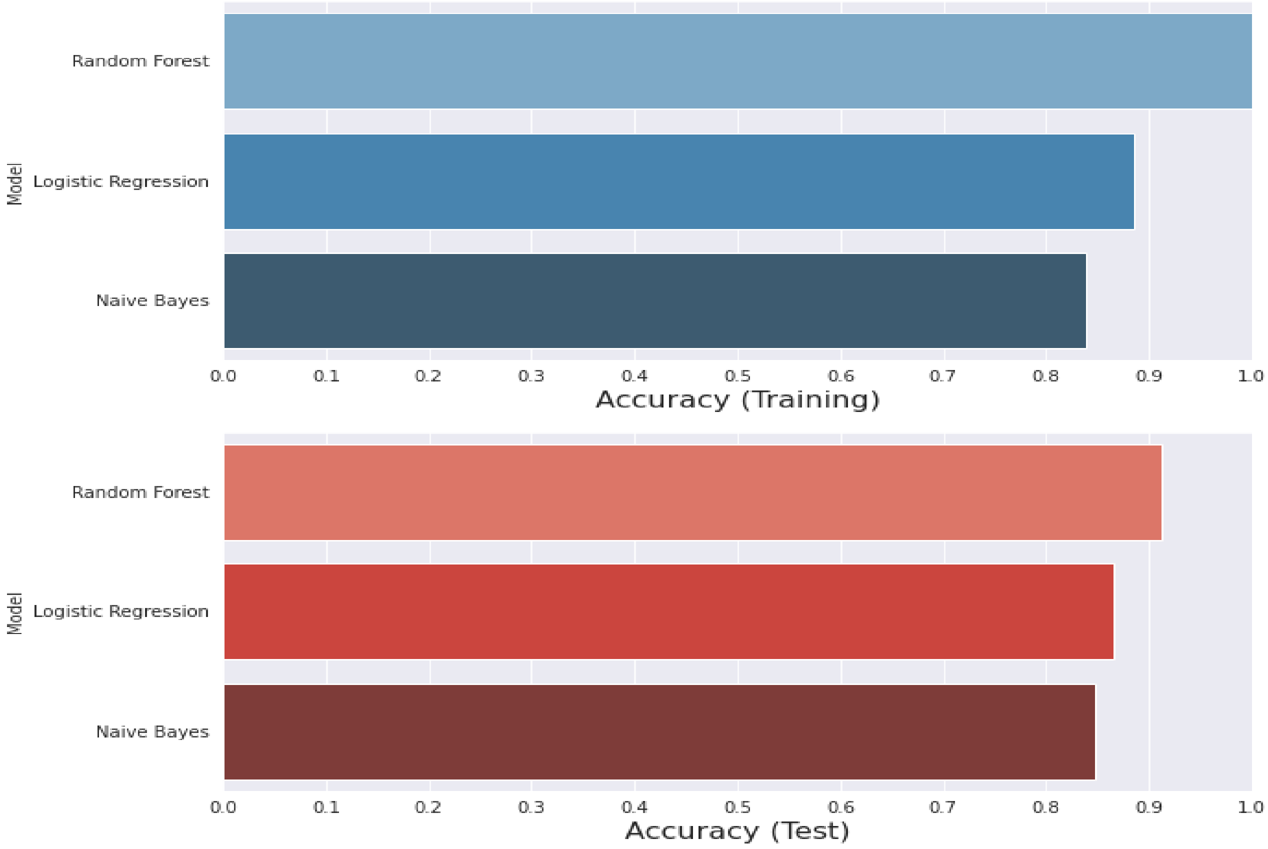




As per our cross validation model, **Random Forest** has scored highest among others and has most effective performance. The training data score matches with testing data as per our result.

Naive Bayes was able predict and classify maximum number of false observation among others.

- **Accuracy of Training and Testing dataset: -**



As per above chart we could observe **Random Forest** scored highest accuracy in both Training and Testing dataset. Thus, our Testing and Training data have almost similar score; our Cross Validation result is correct.

Conclusion

For this work, it was aimed that the analysing which psychochemical are more related with wine quality and which approach is good for prediction of wine quality better. After all work, it is obvious that working with binary classification is better the predict good or bad wines. During this research, three important machine learning techniques were used;

--> Logistic Regression

--> Naive Bayes

--> Random Forest

From all algorithms, it was obvious that for this dataset, Logistic Regression and then Random Forest algorithm gave the best model and accuracy means that those algorithms predict correctly test data. By looking into the details, we can see that good quality wines have higher levels of alcohol on average, have a lower volatile acidity on average, higher levels of sulphates on average, and higher levels of residual sugar on average. If someone wants to analyze similar data like that it is better to work Logistic Regression or Random Forest. Hence, those algorithms variances are found better with high margin terminology, therefore with multiclass analysis, those algorithms will give the best accuracy. After the analysis of this dataset, some features have more effect to deciding quality of the wine, there are some insights about the criteria about wine quality, you can compare just looking the some psychochemical on the label of wines;

Should be higher;

- Alcohol is the most important feature to decide quality of the wine. If the alcohol percentage is high enough, it means that quality of the wine should be better
- Sulphates is another selecting criteria for good wines, with high percentage sulphates wine quality is increasing
- Citric Acid is another selecting criteria, it should be higher to decide more better wine

Should be lower;

- Volatile Acidity should be less in the good wine
- Sulfur dioxide is another effect to decreasing wine quality and also it causes head ache therefore if there is less sulfur dioxide in wine, it should be selected
- Chlorides value has very less effect to quality of the wine but again it is obvious more value of it causes bad quality of the wine

Additionally, for marketing point of view, if a customer wants to buy a wine just looking with some psychochemical values can decide what s/he needs to buy. Of course, brand and price feature was evaluated on this research, therefore, it is not a good analysis for saying “it is good wine”. However, it can give some idea for the people who do not have more knowledge about wine for selecting the good wine maybe for just dinner or gift for friends!

Bibliography

- https://archive.ics.uci.edu/ml/datasets/wine+quality

- https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/#:~:text=Cross%20Validation%20is%20a%20technique,do%20not%20train%20the%20model.&text=You%20reserve%20a%20sample%20data,the%20test%20(validation)%20set.

- https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

- https://www.analyticsvidhya.com/blog/2020/10/introduction-to-logistic-regression-the-most-common-classification-algorithm/

- https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/

- https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009