TERM PROJECT

INTRODUCTION

The quality of a wine is important for the consumers as well as the wine industry. The traditional (expert) way of measuring wine quality is time-consuming. Nowadays, machine learning models are important tools to replace human tasks. In this case, there are several features to predict the wine quality but the entire features will not be relevant for better prediction. So, our thesis work is focusing on what wine features are important to get the promising result. For the purpose of classification model and evaluation of the relevant features, we used three algorithms namely support vector machine (SVM), naïve Bayes (NB), and artificial neural network (ANN). In this study, we used two wine quality datasets red wine and white wine. To evaluate the feature importance we used the Pearson coefficient correlation and performance measurement matrices

such as accuracy, recall, precision, and f1 score for comparison of the machine learning algorithm. A grid search algorithm was applied to improve the model accuracy.

For this project, I used Red 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, I 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

Attributes	Description

fixed acidity	Fixed acids, numeric from 3.8 to 15.9
volatile acidity	Volatile acids, numeric from 0.1 to 1.6
citric acid	Citric acids, numeric from 0.0 to 1.7
residual sugar	residual sugar, numeric from 0.6 to 65.8
chlorides	Chloride, numeric from 0.01 to 0.61
free sulfur dioxide	Free sulfur dioxide, numeric: from 1 to 289
total sulfur dioxide	Total sulfur dioxide, numeric: from 6 to 440
density	Density, numeric: from 0.987 to 1.039
pH	pH, numeric: from 2.7 to 4.0
sulfates	Sulfates, numeric: from 0.2 to 2.0
alcohol	Alcohol, numeric: from 8.0 to 14.9
quality	Quality, numeric: from 0 to 10, the output target

BACKGROUND

A wide range of machine learning algorithms is available for the learning process. This section describes the classification algorithms used in wine quality prediction and related work.

- ***Classification algorithm***
  - **Naive Bayesian**

The naive Bayesian is the simple supervised machine learning classification

algorithm based on the Bayes theorem. The algorithm assumes that the feature conditions are independent of the given class. The naive Bayes algorithm helps to build fast machine learning models that can make a fast prediction. The algorithm finds whether a particular portion has a spot by a particular class it utilizes the probability of likelihood.

- **Support Vector Machine**

  The support vector machine (SVM) is the most popular and most widely used machine learning algorithm. It is a supervised learning model that can perform classification and regression tasks. However, it is primarily used for classification problems in machine learning.

  The SVM algorithm aims to create the best line or decision boundary that can separate n-dimensional space into classes. So we can put the new data points easily in the correct groups. This best decision boundary is called a hyperplane. The support vector machine selects the extreme data points that helping to create the hyperplane. In above diagram, two different groups are classified by using the decision boundary or hyperplane:

  The SVM model is used for both non-linear and linear data. It uses a nonlinear mapping to convert the main preparing information into a higher measurement. The model searches for the linear optimum splitting hyperplane in this new measurement. A hyperplane can split the data into two classes with an appropriate nonlinear mapping to suitably high measurements and for the finding, this hyperplane SVM uses the support vectors and edges. The SVM model is a representation of the models as a point in space, the different classes are isolated by the gap to mapped with

the aim that instances are wide as would be careful. The model can perform out a nonlinear form of classification.

- **Artificial neural network**

The artificial neural network is a collection of neurons that can process information. It has been successfully applied to the classification task in several industries, including the commercial, industrial, and scientific filed. The algorithm model is a connection between the neurons that are interconnected with the input layer, a hidden layer, and an output layer. The neural network is constant because while an element of the neural network is failing, it can continue its parallel nature without any difficulties.

The implementation of the artificial neural network consists of three layers: input, hidden, and, output. The function at the input layer is mapped the input attribute which passes input to the hidden layer. The hidden layer is a middle layer where all input with the weights is received to each node in the hidden layer. The output layer is mapped to the predicted. The connection among the neurons is called weights, it has numerical values and this weight among the neurons are determining the learning ability of the neural network. The activation function is used to standardize the output from the neurons and these activation functions are evaluate the output of the neural network in the mathematical equations. Each neuron has an activation function. The neural network is hard to understand without mathematical reasoning. Activation functions are also called the transmission function and also helps to standardize the output range between -1 to 1 or 0 to 1.

OBJECTIVES

The objectives of this project are as follows:

- Explaining data set through the code of python.
- To implement different machine learning techniques.
- To experiment with different methods to see which yields the highest accuracy.
- To determine which features are the most indicative of a good quality wine.

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

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,accuracy_score
import warnings
```

```
warnings.filterwarnings("ignore")
```

- **Reading Data**

```
import wine dataset

wine = datasets.load_wine()

# np.c_ is the numpy concatenate function

wine_df = pd.DataFrame(data= np.c_[wine['data'], wine['target']],

columns= wine['feature_names'] + ['target'])

wine_df.head()
```

Out[5]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	

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

- **Array**

```
In [25]: wine.keys()
Out[25]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])

In [26]: y=wine['target']
         y
Out[26]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2])
```

❖ Missing Values

```
In [36]: wine_df.isnull().sum() # checking for null values
Out[36]: alcohol                 0
         malic_acid              0
         ash                     0
         alcalinity_of_ash       0
         magnesium               0
         total_phenols           0
         flavanoids              0
         nonflavanoid_phenols    0
         proanthocyanins         0
         color_intensity         0
         hue                     0
         % of diluted_wines      0
         proline                 0
         target                  0
         label                   0
         dtype: int64
```

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

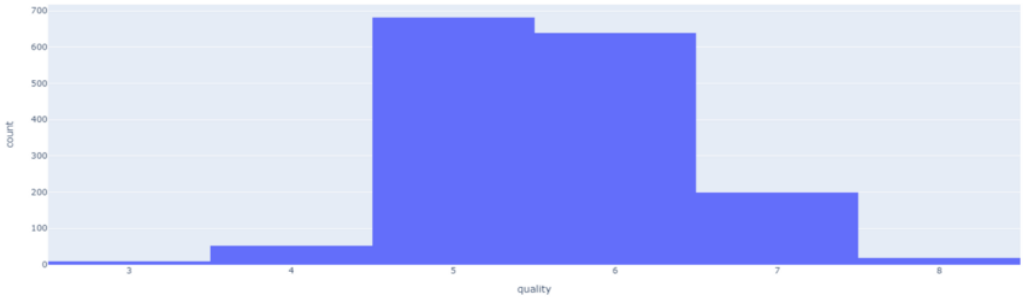
❖ Exploring Variables

Histogram of 'quality' variable

First, I wanted to see the distribution of the *quality* variable. I wanted to make sure that I had enough 'good quality' wines in my dataset — you'll see later how I defined 'good quality'. The datasets are the imbalanced distribution of red wine and white wine where the separate classes are not equally represented. This imbalanced data can lead to overfitting and under fitting algorithms. The

red wine's highest quality class 5 instances are 681 and white wine highest quality class 6 instances are 2198. Both datasets are unbalanced with the number of instances ranging from 5 in the minority class up to 681 in red wine and ranging from 6 in the minority class up to 2198 in the majority class. The highest quality scores are rarely paralleled to the middle classes. By using resampling this problem can be solved, the resampling is by adding copies of examples from the under-represented class of unnaturally creating such instances (over-sampling) or either by removing from the over- represented class (under-sampling). Mostly, it will be better to over-sample unless you have sufficiently of data. However, there are some disadvantages to over-sampling it increases the instances of the dataset, so the processing time is increasing to build the model. Over- sampling can lead to overfitting when putting the extremes. Therefore the resampling is preferred.

```
fig = px.histogram(df,x='quality')
fig.show()
```



❖ **Correlation Matrix**

Next I wanted to see the correlations between the variables that I'm working with. This allows me to get a much better understanding of the relationships between my variables in a quick glimpse.

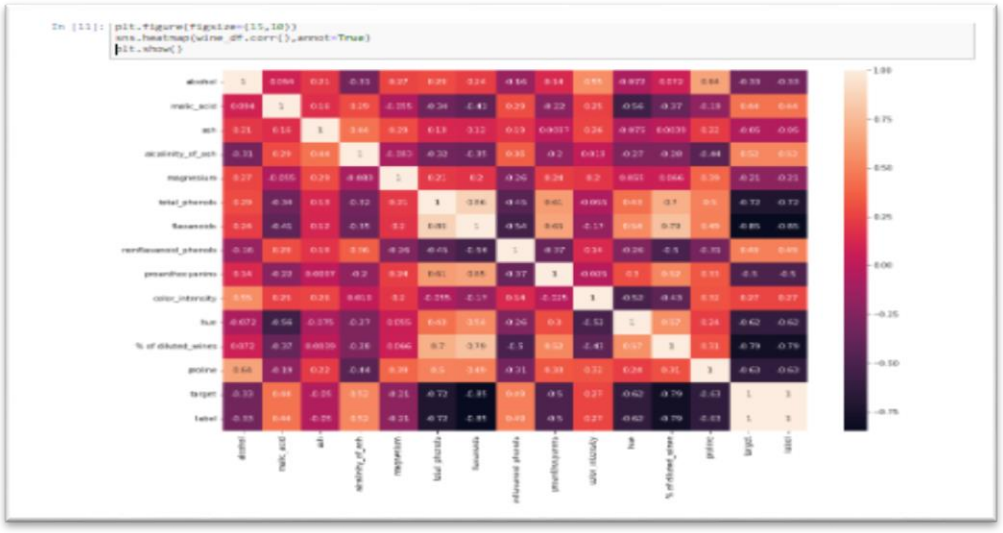
Immediately, I can see that there are some variables that are strongly correlated to *quality.* It's likely that these variables are also the most important features in our machine learning model, but we'll take a look at that later.

For a better understanding of the features and to examines the correlation between the features. We use the Pearson coefficient correlation matrices to calculate the correlation between the features.

plt.figure(figsize=(15,10))

sns.heatmap(wine_df.corr(),annot=True)

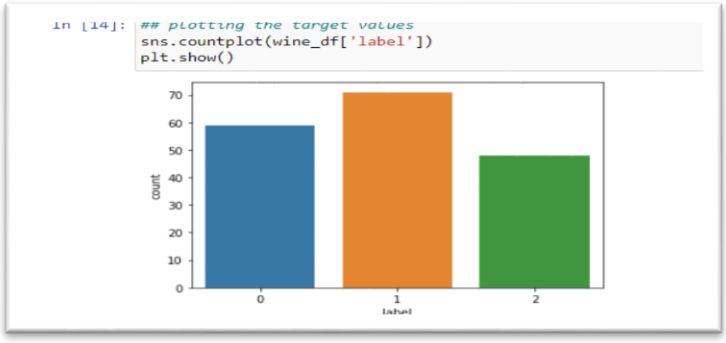
plt.show()

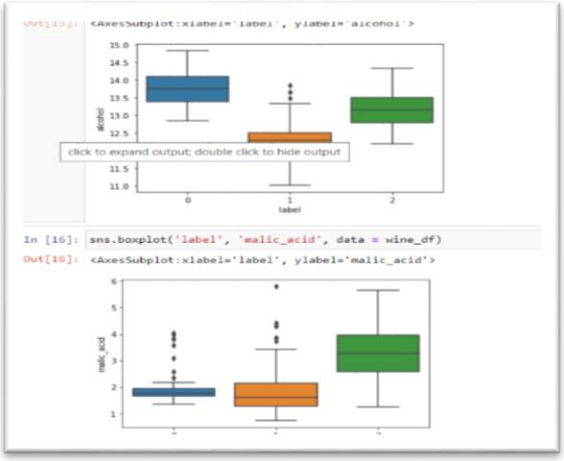


From above screenshot of red wine correlation matrix we ranked the features according to the high correlation values to the quality class such as freatures

are 'alcohol', 'volatile acidity', 'sulphates', 'citric acid', 'total sulfur dioxide', 'density', 'chlorides', 'fixed acidity', 'pH', 'free sulfur dioxide', 'residual sugar'. Similarily, from Figure 6 white wine correlation matrix we ranked the features according to the high correlation values to the quality class such as freatures are 'alcohol', 'density', 'chlorides', 'volatile acidity', 'total sulfur dioxide', 'fixed acidity', 'pH', 'residual sugar', 'sulphates', 'citric acid', 'free sulfur dioxide'.

- **Plotting target values**





- **Convert to a Classification Problem**

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

For this problem, I 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 I converted the output variable to a binary output, I separated my feature variables (X) and the target variable (y) into separate dataframes.

```
# Create Classification version of target variable
df['goodquality'] = [1 if x >= 7 else 0 for x in df['quality']]# Separate feature variables and target variable
X = df.drop(['quality','goodquality'], axis = 1)
y = df['goodquality']
```

❖ **Proportion of Good vs Bad Wines**

I 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 I assumed that it was okay for this purpose.

```
# See proportion of good vs bad wines
df['goodquality'].value_counts()
```

```
0    1382
1     217
Name: goodquality, dtype: int64
```

Preparing Data for Modelling

❖ **Standardizing Feature Variables**

At this point, I felt that I was ready to prepare the data for modelling. The first thing that I did was standardize the data. **Standardizing** the data means that it will transform the data so that its distribution will have a mean of 0 and a

standard deviation of 1. It's important to standardize your data in order to equalize the range of the data.

For example, imagine a dataset with two input features: height in millimeters and weight in pounds. Because the values of 'height' are much higher due to its measurement, a greater emphasis will *automatically be placed on height than weight, creating a bias.*

```
# Normalize feature variables
from sklearn.preprocessing import StandardScaler
X_features = X
X = StandardScaler().fit_transform(X)
```

Split data

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

```
# Splitting the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25,
random_state=0)
```

Now,

- **Modelling**

For this project, I wanted to compare five different machine learning models: decision trees, random forests, AdaBoost, Gradient Boost, and XGBoost. For the purpose of this project, I wanted to compare these models by their accuracy.

- **Model 1: Decision Tree**

  Decision trees are a popular model, used in operations research, strategic planning, and machine learning. Each square above is called a node, and the more nodes you have, the more accurate your decision tree will be (generally). The last nodes of the decision tree, where a decision is made, are called the leaves of the tree. Decision trees are intuitive and easy to build but fall short when it comes to accuracy.

```
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifiermodel1 =
DecisionTreeClassifier(random_state=1)
model1.fit(X_train, y_train)
y_pred1 = model1.predict(X_test)print(classification_report(y_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94       355
           1       0.53      0.73      0.62        45

    accuracy                           0.90       400
   macro avg       0.75      0.83      0.78       400
weighted avg       0.92      0.90      0.90       400
```

- **Model 2: Random Forest**

  Random forests are an ensemble learning technique that builds off of decision trees. Random forests involve creating multiple decision trees using bootstrapped datasets of the original data and randomly selecting a subset of variables at each step of the decision tree. The model then selects the mode of all of the predictions of each decision tree. What's the point of this? By

relying on a “majority wins” model, it reduces the risk of error from an individual tree.

For example, if we created one decision tree, the third one, it would predict 0. But if we relied on the mode of all 4 decision trees, the predicted value would be 1. This is the power of random forests.

```
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier(random_state=1)
model2.fit(X_train, y_train)
y_pred2 = model2.predict(X_test)print(classification_report(y_test, y_pred2))
```

	precision	recall	f1-score	support
0	0.95	0.97	0.96	355
1	0.68	0.58	0.63	45
accuracy			0.92	400
macro avg	0.82	0.77	0.79	400
weighted avg	0.92	0.92	0.92	400

- **Model 3: AdaBoost**

  The next three models are boosting algorithms that take weak learners and turn them into strong ones. I don’t want to get sidetracked and explain the differences between the three because it’s quite complicated and intricate. That being said, I’ll leave some resources where you can learn about AdaBoost, Gradient Boosting, and XGBoosting.

StatQuest: AdaBoost

StatQuest: Gradient Boost

- *StatQuest: XGBoost*

```
from sklearn.ensemble import AdaBoostClassifier
model3 = AdaBoostClassifier(random_state=1)
model3.fit(X_train, y_train)
y_pred3 = model3.predict(X_test)print(classification_report(y_test, y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       355
           1       0.51      0.49      0.50        45

    accuracy                           0.89       400
   macro avg       0.72      0.71      0.72       400
weighted avg       0.89      0.89      0.89       400
```

- **Model4: KNN**

```
# Train
fit_knn <- knn3(formula = fml, data = train_set, k = 5)


# Predict
y_knn <- predict(object = fit_knn,
newdata = test_set,
type ="class")


# Compare the results: confusion matrix
caret::confusionMatrix(data = y_knn,
reference = test_set$type,
positive = "red")
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction red white
##      red   145    15
##      white  15   475
##
##               Accuracy : 0.954
##                 95% CI : (0.935, 0.969)
##    No Information Rate : 0.754
##    P-Value [Acc > NIR] : <2e-16
##
##                  Kappa : 0.876
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.906
##             Specificity : 0.969
##          Pos Pred Value : 0.906
##          Neg Pred Value : 0.969
##              Prevalence : 0.246
##          Detection Rate : 0.223
##    Detection Prevalence : 0.246
##       Balanced Accuracy : 0.938
```

```
##
##        'Positive' Class : red
##
# F1 score
F_meas(data = y_knn, reference = test_set$type)
## [1] 0.906
```

- **Model 4: Gradient Boosting**

```
from sklearn.ensemble import GradientBoostingClassifier
model4 = GradientBoostingClassifier(random_state=1)
model4.fit(X_train, y_train)
y_pred4 = model4.predict(X_test)print(classification_report(y_test, y_pred4))
```

	precision	recall	f1-score	support
0	0.94	0.94	0.94	355
1	0.52	0.51	0.52	45
accuracy			0.89	400
macro avg	0.73	0.73	0.73	400
weighted avg	0.89	0.89	0.89	400

- **Model 5: XGBoost**

```
import xgboost as xgb
model5 = xgb.XGBClassifier(random_state=1)
model5.fit(X_train, y_train)
y_pred5 = model5.predict(X_test)print(classification_report(y_test, y_pred5))
```

	precision	recall	f1-score	support
0	0.96	0.95	0.95	355
1	0.62	0.69	0.65	45
accuracy			0.92	400
macro avg	0.79	0.82	0.80	400
weighted avg	0.92	0.92	0.92	400

By comparing the five models, the random forest and XGBoost seems to yield the highest level of accuracy. However, since XGBoost has a better f1-score for predicting good quality wines (1), I'm concluding that the XGBoost is the winner of the five models.

- **Feature Importance**

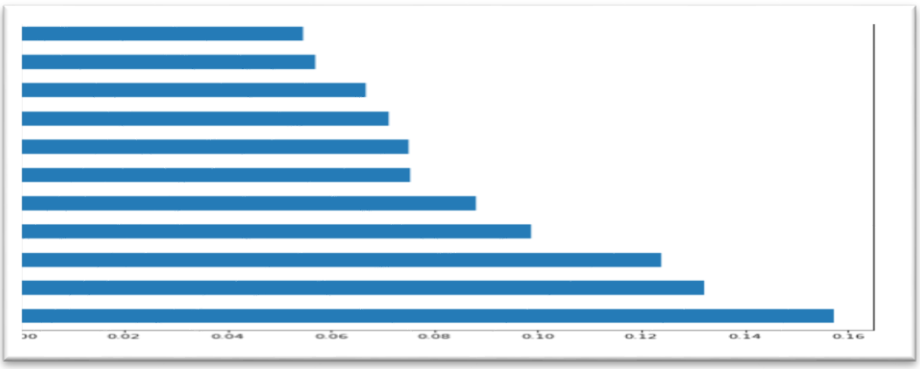
Below, I graphed the feature importance based on the Random Forest model and the XGBoost model. While they slightly vary, the top 3 features are the same: alcohol, volatile acidity, and sulphates. If you look below the graphs, I split the dataset into good quality and bad quality to compare these variables in more detail. The importance of the features are identified and from both dataset's first 10 features were selected and the last feature was excluded, above red wine performance analysis and white wine performance analysis shows that the performance in terms of accuracy.

Firstly, these selected features were implemented on the unbalanced classes, the unbalanced classes and the performance of the prediction model, in terms of accuracy, precision, recall, and F1 score is examined.

	SVM			NB			ANN		
Class	Precisi	Recall	F1	Precisi	Recall	F1	Precisi	Recall	F1
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.17	0.50	0.26	0.00	0.00	0.00
5	0.79	0.79	0.79	0.73	0.60	0.66	0.70	0.82	0.76
6	0.60	0.60	0.60	0.54	0.53	0.54	0.57	0.62	0.59
7	0.62	0.62	0.62	0.32	0.43	0.37	0.62	0.23	0.33
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Accuracy	69.06			54.06			64.37		

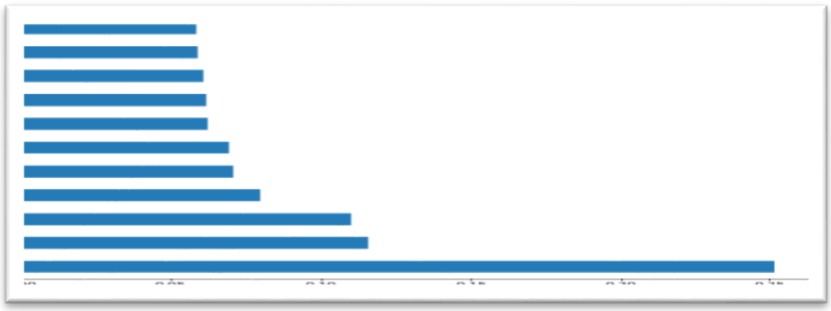
- **via Random Forest**

```
feat_importances = pd.Series(model2.feature_importances_,
index=X_features.columns)
feat_importances.nlargest(25).plot(kind='barh',figsize=(10,10))
```



- **via XGBoost**

```
feat_importances = pd.Series(model5.feature_importances_,
index=X_features.columns)
feat_importances.nlargest(25).plot(kind='barh',figsize=(10,10))
```



Evaluation

The performance measurement is calculated and evaluate the techniques to detect the effectiveness and efficiency of the model. There are four ways to check the predictions are correct or incorrect:

- True Positive: Number of samples that are predicted to be positive which are truly positive.
- False Positive: Number of samples that are predicted to be positive which are truly negative.
- False Negative: Number of samples that are predicted to be negative which are truly positive.
- True Negative: Number of samples that are predicted to be negative which are truly negative.

Below listed techniques, we use for the evaluation of the model.

1. Accuracy – Accuracy is defined as the ratio of correctly predicted observation to the total observation. The accuracy can be calculated easily by dividing the

number of correct predictions by the total number of prediction.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{False Positive} + \text{False Negative} + \text{True Negative}}$$

2. Precision – Precision is defined as the ratio of correctly predicted positive observations to the total predicted positive observations.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

3. Recall – Recall is defined as the ratio of correctly predicted positive observations to all observations in the actual class. The recall is also known as the True Positive rate calculated as,

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

4. F1 Score – F1 score is the weighted average of precision and recall. The f1 score is used to measure the test accuracy of the model. F1 score is calculated by multiplying the recall and precision is divided by the recall and precision, and the result is calculated by multiplying two.

$$\text{F1 score} = 2 * \frac{\text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

Accuracy is the most widely used evaluation metric for most traditional applications. But the accuracy rate is not suitable for evaluating imbalanced data sets, because many experts have observed that for extremely skewed class distributions, the recall rate for minority classes is typically 0, which means that no classification rules are generated for the minority class. Using the terminology in information retrieval, the precision and recall of the minority categories are much lower than the majority class. Accuracy gives more weight to the majority class than to the minority class, this makes it challenging for the classifier to implement well in the minority class.

For this purpose, additional metrics are coming into widespread usage (Guo et al., 2008).

The F1 score is the popular evaluation matric for the imbalanced class problem (Estabrooks and Japkowicz, 2001). F1 score combines two matrices: precision and recall. Precision state how accurate the model was predicting a certain class and recall state that the opposite of the regrate misplaced instances which are misclassified. Since the multiple classes have multiple F1 scores. By using the unweighted mean of the F1 scores for our final scoring. We want our models to get optimized to classify instances that belong to the minority side, such as wine quality of 3, 8, or 9 equally well with the rest of the qualities that are represented in a larger number.

Comparing the Top 4 Features

```
# Filtering df for only good quality
df_temp = df[df['goodquality']==1]
df_temp.describe()# Filtering df for only bad quality
df_temp2 = df[df['goodquality']==0]
df_temp2.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	goodquality
count	217.000000	217.000000	217.000000	217.000000	217.000000	217.000000	217.000000	217.000000	217.000000	217.000000	217.000000	217.000000	217.0
mean	8.847005	0.405530	0.376498	2.708756	0.075912	13.981567	34.889401	0.996030	3.288802	0.743456	11.518049	7.082949	1.0
std	1.999977	0.144963	0.194438	1.363026	0.028480	10.234615	32.572238	0.002201	0.154478	0.134038	0.998153	0.276443	0.0
min	4.900000	0.120000	0.000000	1.200000	0.012000	3.000000	7.000000	0.990640	2.880000	0.390000	9.200000	7.000000	1.0
25%	7.400000	0.300000	0.300000	2.000000	0.062000	6.000000	17.000000	0.994700	3.200000	0.650000	10.800000	7.000000	1.0
50%	8.700000	0.370000	0.400000	2.300000	0.073000	11.000000	27.000000	0.995720	3.270000	0.740000	11.600000	7.000000	1.0
75%	10.100000	0.490000	0.490000	2.700000	0.085000	18.000000	43.000000	0.997350	3.380000	0.820000	12.200000	7.000000	1.0
max	15.600000	0.915000	0.760000	8.900000	0.358000	54.000000	289.000000	1.003200	3.780000	1.360000	14.000000	8.000000	1.0

Good Quality

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	goodquality
count	1382.000000	1382.000000	1382.000000	1382.000000	1382.000000	1382.000000	1382.000000	1382.000000	1382.000000	1382.000000	1382.000000	1382.000000	1382.0
mean	8.236831	0.547022	0.254407	2.512120	0.089281	16.172214	48.285818	0.996859	3.314616	0.644754	10.251037	5.408828	0.0
std	1.682726	0.176337	0.189665	1.415778	0.049113	10.467685	32.585604	0.001808	0.154135	0.170629	0.969664	0.601719	0.0
min	4.600000	0.160000	0.000000	0.900000	0.034000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000	0.0
25%	7.100000	0.420000	0.082500	1.900000	0.071000	8.000000	23.000000	0.995785	3.210000	0.540000	9.500000	5.000000	0.0
50%	7.800000	0.540000	0.240000	2.200000	0.080000	14.000000	39.500000	0.996800	3.310000	0.600000	10.000000	5.000000	0.0
75%	9.100000	0.650000	0.400000	2.600000	0.091000	22.000000	65.000000	0.997900	3.410000	0.700000	10.900000	6.000000	0.0
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	165.000000	1.003690	4.010000	2.000000	14.900000	6.000000	0.0

Bad Quality

- **Conclusion**

we implement the correlation matrices and calculate the relationship among all the features, as in red wine correlation matrices.Then we ranked the features from these unbalancing and balancing

classes, we achieved a better performance result on the balanced class for all the models.

based on high correlation with the quality feature. The analysis of groups of features from left to right is implemented and first 10 features are selected and the last feature is excluded because there is no improvement and it is decreasing the performance of the model. 'residual sugar' feature from red wine datasets and 'free sulfur dioxide' feature from the white wine dataset is excluded for the final implementation of the models.

After identifying the importance of the features we start the implementation of the model. To analyze the performance of the model firstly, we implemented the model on the original data (unbalanced class), and then implemented the model on the balance class, balancing each class. In terms of the performance of the prediction model accuracy, precision, recall, and f1 score is examined, performance analysis results for unbalanced classes for each model is examined.

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.