GROUP PROJECT (GROUP 28)

Step 1: Import Neccesary Libraries

In [149]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

Step 2: Data Analysis

In [1]:

```
cd C:/Users/shivar
```

```
C:\Users\shivar
```

In [151]:

```
# read the data set in a dataframe
df = pd.read_csv("Data set 3 (99 KB) - winequality.csv")
df
```

Out[151]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quali
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	
...	...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	

1599 rows × 12 columns

In [152]:

```
# this function used to print the first (10) value of the dataframe
df.head(10)
```

Out[152]:

	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
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5

In [153]:

```
# this function used to print the last (10) value of the dataframe
df.tail(10)
```

Out[153]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quali
1589	6.6	0.725	0.20	7.8	0.073	29.0	79.0	0.99770	3.29	0.54	9.2	
1590	6.3	0.550	0.15	1.8	0.077	26.0	35.0	0.99314	3.32	0.82	11.6	
1591	5.4	0.740	0.09	1.7	0.089	16.0	26.0	0.99402	3.67	0.56	11.6	
1592	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	
1593	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9.5	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	

In [154]:

```
df.shape
```

Out[154]:

```
(1599, 12)
```

In [155]:

```
# look through of the data set info in term of statistic
df.describe()
```

Out[155]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	d
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.9
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.0
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.9
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.9
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.9
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.9
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.0

Step 3: Data Preprocessing

In [156]:

```
# Check missing value
df.apply(lambda x: sum(x.isnull()),axis=0)
```

Out[156]:

```
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
```

BINARIZATION OF TARGET VARIABLE

In [157]:

```
# display target variable
df['quality'].unique()
```

Out[157]:

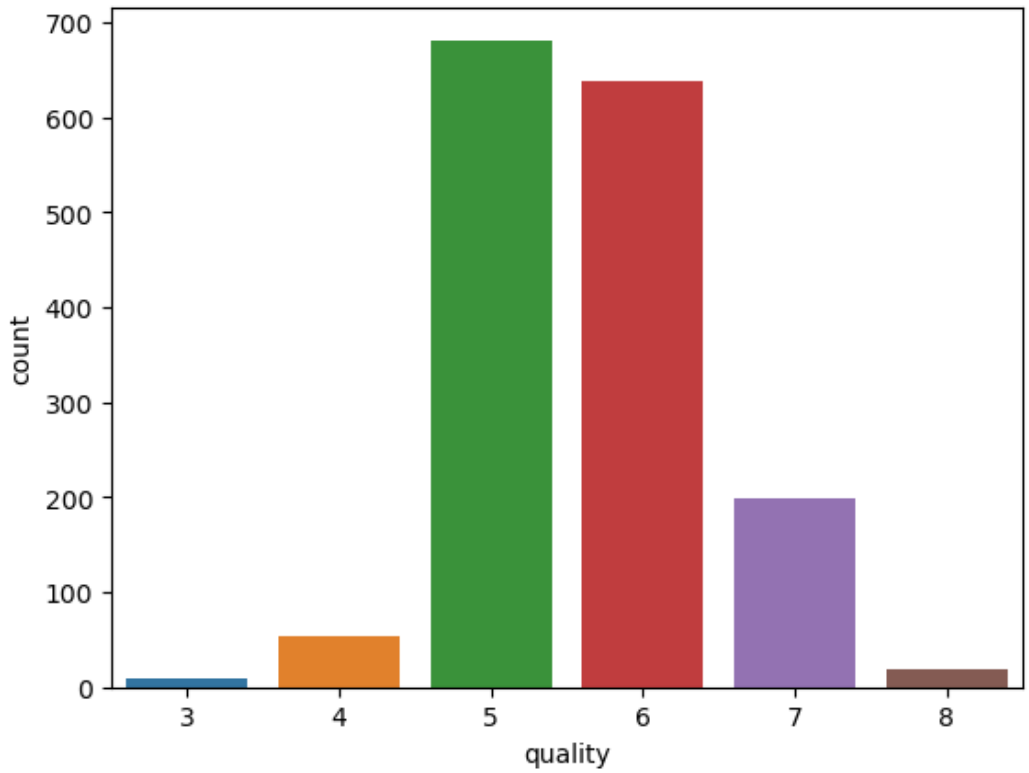
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [158]:

```
# plot graph to se the distribution of the target variable
sns.countplot(data = df, x='quality')
```

Out[158]:

```
<Axes: xlabel='quality', ylabel='count'>
```



In [159]:

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

Out[159]:

```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

Data Encoding/Binarization

In [160]:

```
# perform binarization of target variable
# Assign 1 to Good quality wine that has quality
df['quality'] = [1 if x>=7 else 0 for x in df['quality']]

# display new target variable
df['quality'].unique()
```

Out[160]:

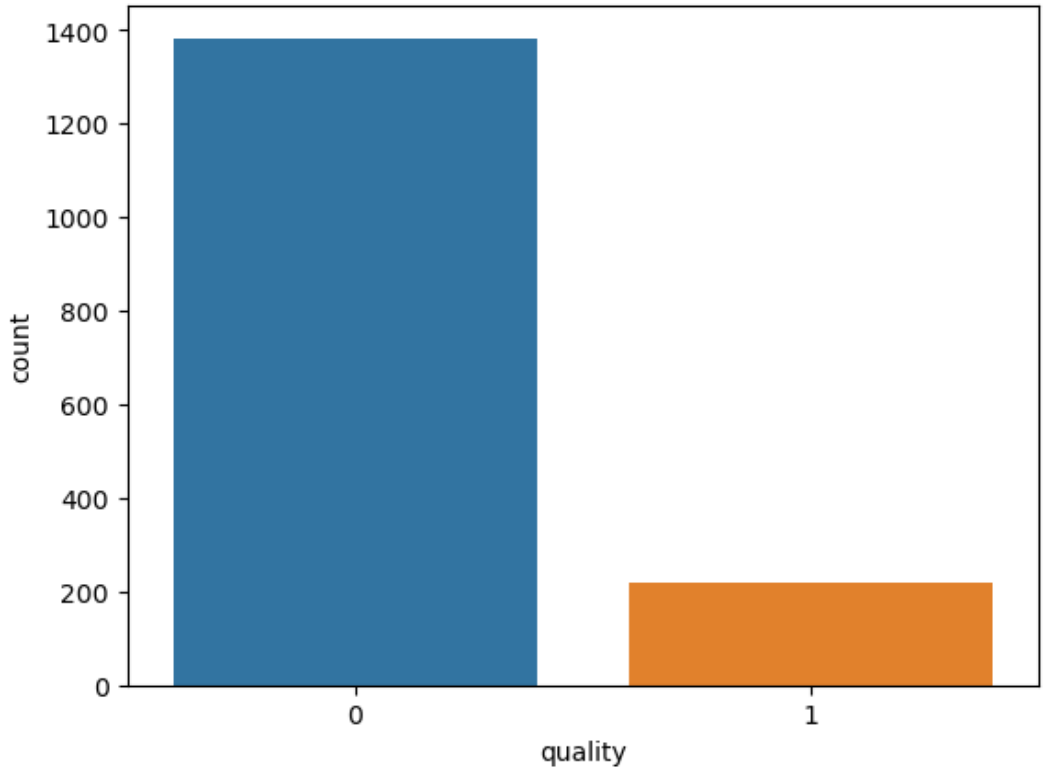
```
array([0, 1], dtype=int64)
```

In [161]:

```
# plot graph to see the distribution of target variable
sns.countplot(data = df, x='quality')
```

Out[161]:

```
<Axes: xlabel='quality', ylabel='count'>
```



In [162]:

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

Out[162]:

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

In [163]:

```
df
```

Out[163]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quali
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	
...	...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	

1599 rows × 12 columns

In [164]:

```
x = df.iloc[:,0:11] #only have 12 column
y = df.iloc[:,-1] # last column
```

In [165]:

```
x
```

Out[165]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0

1599 rows × 11 columns

In [166]:

```
y
```

Out[166]:

```
0       0
1       0
2       0
3       0
4       0
       ..
1594    0
1595    0
1596    0
1597    0
1598    0
Name: quality, Length: 1599, dtype: int64
```

Step 4: Splitting dataset into 60:40, 70:30 and 80:20 Ratios

FEATURE SCALING

future scaling in machine learning is one of the most critical steps during the data preprocessing of data before creatinf the machine learning model. scaling can make a difference between a weak and good machine learning model.

the most common technique scaling are Normalization and Standardization (other: MinMax Scaler, Binarizer)

ex: if an algo is not using feature scaling method then it can consider the value 1000 meter to be greater than 7km but that's actually not true and in this case, the algo will give wrong predictios. so, we use feature scaling to bring values to same magnitudes and thus, tackle this issue

In [167]:

```
# to put over features on the same scale
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)

# Convert scaled features to DataFrame with column names
x = pd.DataFrame(x_scaled, columns=x.columns)
```

Splitting dataset to ratio 60:40

In [168]:

```
# split dataset into train set and test set
x_trainA, x_testA, y_trainA, y_testA = train_test_split(x, y, test_size=0.4, random_state=0)

# Save training set to CSV
train_data = pd.concat([x_trainA, y_trainA], axis=1)
train_data.to_csv('train_A.csv', index=False)

# Save test set to CSV
test_data = pd.concat([x_testA, y_testA], axis=1)
test_data.to_csv('test_A.csv', index=False)
```

In [169]:

```
rf = RandomForestClassifier()
rf.fit(x_trainA, y_trainA)
```

Out[169]:

```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [170]:

```
#perform prediction
y_predictA = rf.predict(x_testA)

print("SCORE (60:40)")
print("------------------------------")
print("Accuracy: ", accuracy_score(y_testA, y_predictA))
print("F1 Score: ", f1_score(y_testA, y_predictA, average='weighted'))
print("Recall: ", recall_score(y_testA, y_predictA, average='weighted'))
print("Precision: ", precision_score(y_testA, y_predictA, average='weighted', zero_division=0))
```

```
SCORE (60:40)
------------------------------
Accuracy:  0.91875
F1 Score:  0.9192854445052226
Recall:  0.91875
Precision:  0.9198542266556972
```

Splitting dataset to ratio 70:30

In [171]:

```
# to put over features on the same scale
# essential in ML - calc distances beween the data
# not scale, feature with high value range start dominating when calc distances
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)

# Convert scaled features to DataFrame with column names
x = pd.DataFrame(x_scaled, columns=x.columns)
```

In [172]:

```
# split dataset into train set and test set (for feature importance)
x_trainB, x_testB, y_trainB, y_testB = train_test_split(x, y, test_size=0.3, random_state=0)

# Save training set to CSV
train_data = pd.concat([x_trainB, y_trainB], axis=1)
train_data.to_csv('train_B.csv', index=False)

# Save test set to CSV
test_data = pd.concat([x_testB, y_testB], axis=1)
test_data.to_csv('test_B.csv', index=False)
```

In [173]:

```
rf = RandomForestClassifier()
rf.fit(x_trainB, y_trainB)
```

Out[173]:

```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [174]:

```
#perform prediction
y_predictB = rf.predict(x_testB)

print("SCORE (70:30)")
print("------------------------------")
print("Accuracy: ", accuracy_score(y_testB, y_predictB))
print("F1 Score: ", f1_score(y_testB, y_predictB, average='weighted'))
print("Recall: ", recall_score(y_testB, y_predictB, average='weighted'))
print("Precision: ", precision_score(y_testB, y_predictB, average='weighted', zero_division=0))
```

```
SCORE (70:30)
------------------------------
Accuracy:  0.91875
F1 Score:  0.9168466686948585
Recall:  0.91875
Precision:  0.9153416347381864
```

Splitting dataset to ratio 80:20

In [175]:

```
# to put over features on the same scale
# essential in ML - calc distances beween the data
# not scale, feature with high value range start dominating when calc distances
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)

# Convert scaled features to DataFrame with column names
x = pd.DataFrame(x_scaled, columns=x.columns)
```

In [176]:

```
# split dataset into train set and test set (for feature importance)
x_trainC, x_testC, y_trainC, y_testC = train_test_split(x, y, test_size=0.2, random_state=0)

# Save training set to CSV
train_data = pd.concat([x_trainC, y_trainC], axis=1)
train_data.to_csv('train_C.csv', index=False)

# Save test set to CSV
test_data = pd.concat([x_testC, y_testC], axis=1)
test_data.to_csv('test_C.csv', index=False)
```

In [177]:

```
rf = RandomForestClassifier()
rf.fit(x_trainC, y_trainC)
```

Out[177]:

```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [178]:

```
#perform prediction
y_predictC = rf.predict(x_testC)

print("SCORE (80:20)")
print("------------------------------")
print("Accuracy: ", accuracy_score(y_testC, y_predictC))
print("F1 Score: ", f1_score(y_testC, y_predictC, average='weighted'))
print("Recall: ", recall_score(y_testC, y_predictC, average='weighted'))
print("Precision: ", precision_score(y_testC, y_predictC, average='weighted', zero_division=0))
```

```
SCORE (80:20)
------------------------------
Accuracy:  0.928125
F1 Score:  0.9251651126651126
Recall:  0.928125
Precision:  0.9232415254237287
```

Thus, ratio 80:20 choosen since it give the highest accuracy

Step 5: Feature selection and Modelling

Without Feature Selection

In [179]:

```
# split dataset into train set and test set (for feature importance)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)

# Save training set to CSV
train_data = pd.concat([x_train, y_train], axis=1)
train_data.to_csv('train_cm.csv', index=False)

# Save test set to CSV
test_data = pd.concat([x_test, y_test], axis=1)
test_data.to_csv('test_cm.csv', index=False)
```

In [180]:

```
classifier = RandomForestClassifier()
classifier.fit(x_train, y_train)
```

Out[180]:

```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [181]:

```
#perform prediction
y_predict = classifier.predict(x_test)

print("SCORE (Without Feature Selection )")
print("------------------------------")
print("Accuracy: ", accuracy_score(y_test, y_predict))
print("F1 Score: ", f1_score(y_test, y_predict, average='weighted'))
print("Recall: ", recall_score(y_test, y_predict, average='weighted'))
print("Precision: ", precision_score(y_test, y_predict, average='weighted', zero_division=0))
```

```
SCORE (Without Feature Selection )
------------------------------
Accuracy:  0.934375
F1 Score:  0.9316724941724942
Recall:  0.934375
Precision:  0.9300635593220339
```

1. Feature Importance

Feature importance is an inbuilt class that comes with Tree Based Classifiers, we will be using Extra Tree Classifier for extracting the top 10 features for the dataset.

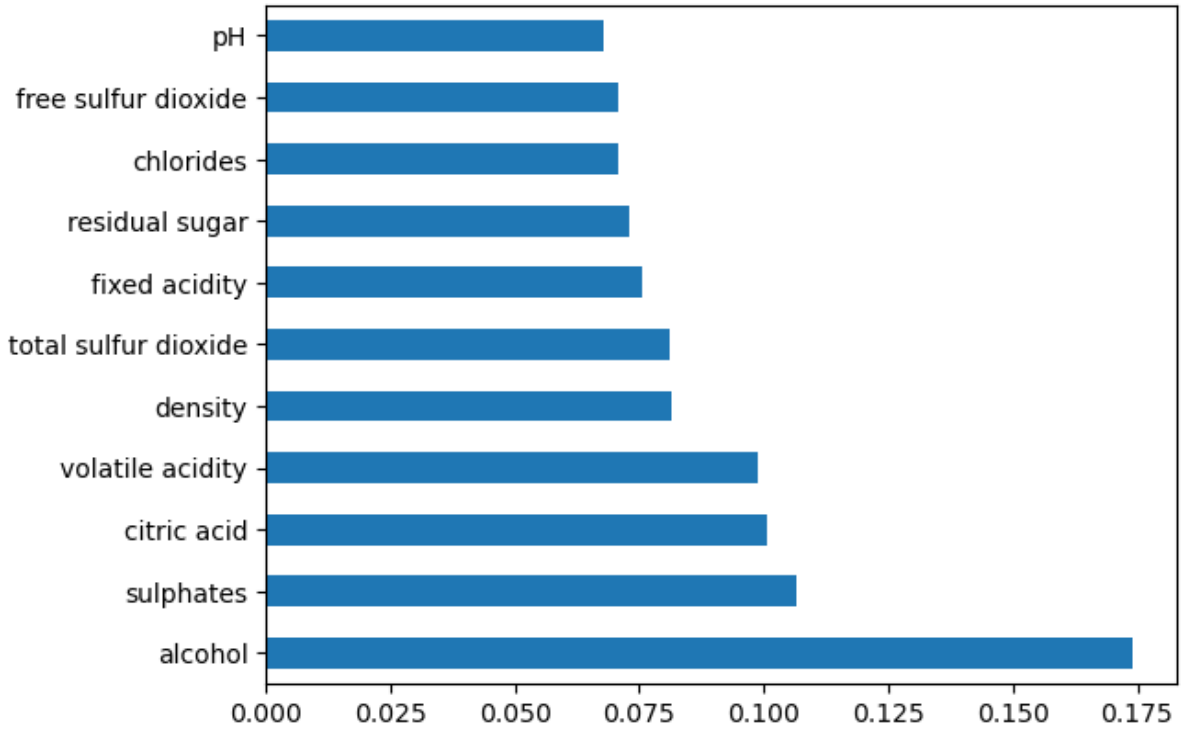
In [182]:

```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt

model = ExtraTreesClassifier()
model.fit(x, y)

#use inbuilt class feature_importances of tree based classifiers
print(model.feature_importances_)
```

```
[0.07554258 0.09863595 0.10073365 0.07281234 0.07069638 0.0706625
 0.08100312 0.08134128 0.06773931 0.10659931 0.1742336 ]
```

In [183]:

```
# plot graph of feature importances for better visualization

feat_importances = pd.Series(model.feature_importances_, index=x.columns)
feat_importances.nlargest(12).plot(kind='barh')
plt.show()
```



With Feature Selection

1) Feature Importance

6 Features

In [184]:

```
# Define the selected features
selected_features_1A = ['alcohol','citric acid' , 'volatile acidity','sulphates','total sulfur diox

# Select the desired features from the training and testing sets
xtrain_selected_1A = x_train[selected_features_1A]
xtest_selected_1A = x_test[selected_features_1A]

# Create and train the Random Forest model
classifier= RandomForestClassifier()
classifier.fit(xtrain_selected_1A, y_train)

# Make predictions on the test set
y_pred_1A = classifier.predict(xtest_selected_1A)

# Print the evaluation metrics for the Random Forest model
print("SCORE (Random Forest-6)")
print("------------------------------")
print("Accuracy:", accuracy_score(y_test, y_pred_1A))
print("F1 Score:", f1_score(y_test, y_pred_1A, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_1A, average='weighted'))
print("Precision:", precision_score(y_test, y_pred_1A, average='weighted', zero_division=0))
```

```
SCORE (Random Forest-6)
------------------------------
Accuracy: 0.91875
F1 Score: 0.9199199129367118
Recall: 0.91875
Precision: 0.9212239583333334
```

4 Features

In [185]:

```
# Define the selected features
selected_features_1B = ['alcohol', 'citric acid' , 'volatile acidity','sulphates']

# Select the desired features from the training and testing sets
xtrain_selected_1B = x_train[selected_features_1B]
xtest_selected_1B = x_test[selected_features_1B]

# Create and train the Random Forest model
classifier= RandomForestClassifier()
classifier.fit(xtrain_selected_1B, y_train)

# Make predictions on the test set
y_pred_1B = classifier.predict(xtest_selected_1B)

# Print the evaluation metrics for the Random Forest model
print("SCORE (Random Forest-4)")
print("------------------------------")
print("Accuracy:", accuracy_score(y_test, y_pred_1B))
print("F1 Score:", f1_score(y_test, y_pred_1B, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_1B, average='weighted'))
print("Precision:", precision_score(y_test, y_pred_1B, average='weighted', zero_division=0))
```

```
SCORE (Random Forest-4)
------------------------------
Accuracy: 0.90625
F1 Score: 0.9111091114767585
Recall: 0.90625
Precision: 0.9177748226950355
```

2 Features

In [186]:

```
# Define the selected features
selected_features_1C = ['alcohol', 'sulphates']

# Select the desired features from the training and testing sets
xtrain_selected_1C = x_train[selected_features_1C]
xtest_selected_1C = x_test[selected_features_1C]

# Create and train the Random Forest model
classifier= RandomForestClassifier()
classifier.fit(xtrain_selected_1C, y_train)

# Make predictions on the test set
y_pred_1C = classifier.predict(xtest_selected_1C)

# Print the evaluation metrics for the Random Forest model
print("SCORE (Random Forest-2)")
print("------------------------------")
print("Accuracy:", accuracy_score(y_test, y_pred_1C))
print("F1 Score:", f1_score(y_test, y_pred_1C, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_1C, average='weighted'))
print("Precision:", precision_score(y_test, y_pred_1C, average='weighted', zero_division=0))
```

```
SCORE (Random Forest-2)
------------------------------
Accuracy: 0.86875
F1 Score: 0.8755527560674621
Recall: 0.86875
Precision: 0.883690276222471
```

2. Univariate Selection

Statistical tests can be used to select those features that have the strongest relationship with the output variable. The scikit-learn library provides the SelectKBest class that can be used with a suite of different statistical tests to select a specific number of features.

The example below uses the chi-squared (chi²) statistical test for non-negative features to select 10 of the best features from the Mobile Price Range Prediction Dataset.

In [187]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
```

In [188]:

```
dfscores
```

Out[188]:

	0
0	1.661298
1	6.306147
2	10.317077
3	0.303084
4	0.741466
5	0.852219
6	2.939007
7	1.417023
8	0.172316
9	3.334157
10	22.896820

In [189]:

```
dfcolumns
```

Out[189]:

	0
0	fixed acidity
1	volatile acidity
2	citric acid
3	residual sugar
4	chlorides
5	free sulfur dioxide
6	total sulfur dioxide
7	density
8	pH
9	sulphates
10	alcohol

In [190]:

```
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)

#naming the dataframe columns
featureScores.columns = ['Specs','Score']

#print 10 best features
featureScores
```

Out[190]:

	Specs	Score
0	fixed acidity	1.661298
1	volatile acidity	6.306147
2	citric acid	10.317077
3	residual sugar	0.303084
4	chlorides	0.741466
5	free sulfur dioxide	0.852219
6	total sulfur dioxide	2.939007
7	density	1.417023
8	pH	0.172316
9	sulphates	3.334157
10	alcohol	22.896820

In [191]:

```
print(featureScores.nlargest(6,'Score')) #print 6 best features
```

```
                  Specs      Score
10              alcohol  22.896820
2           citric acid  10.317077
1      volatile acidity   6.306147
9              sulphates   3.334157
6   total sulfur dioxide   2.939007
0         fixed acidity   1.661298
```

6 Features

In [192]:

```
# Define the selected features
selected_features_2A = ['alcohol','citric acid' , 'volatile acidity','sulphates','total sulfur diox

# Select the desired features from the training and testing sets
xtrain_selected_2A = x_train[selected_features_2A]
xtest_selected_2A = x_test[selected_features_2A]

# Create and train the Random Forest model
classifier= RandomForestClassifier()
classifier.fit(xtrain_selected_2A, y_train)

# Make predictions on the test set
y_pred_2A = classifier.predict(xtest_selected_2A)

# Print the evaluation metrics for the Random Forest model
print("SCORE (Random Forest-6)")
print("------------------------------")
print("Accuracy:", accuracy_score(y_test, y_pred_2A))
print("F1 Score:", f1_score(y_test, y_pred_2A, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_2A, average='weighted'))
print("Precision:", precision_score(y_test, y_pred_2A, average='weighted', zero_division=0))
```

```
SCORE (Random Forest-6)
------------------------------
Accuracy: 0.925
F1 Score: 0.9238357625311056
Recall: 0.925
Precision: 0.9228228962818005
```

4 Features

In [193]:

```
# Define the selected features
selected_features_2B = ['alcohol','citric acid' , 'volatile acidity','sulphates']

# Select the desired features from the training and testing sets
xtrain_selected_2B = x_train[selected_features_2B]
xtest_selected_2B = x_test[selected_features_2B]

# Create and train the Random Forest model
classifier= RandomForestClassifier()
classifier.fit(xtrain_selected_2B, y_train)

# Make predictions on the test set
y_pred_2B = classifier.predict(xtest_selected_2B)

# Print the evaluation metrics for the Random Forest model
print("SCORE (Random Forest=4)")
print("------------------------------")
print("Accuracy:", accuracy_score(y_test, y_pred_2B))
print("F1 Score:", f1_score(y_test, y_pred_2B, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_2B, average='weighted'))
print("Precision:", precision_score(y_test, y_pred_2B, average='weighted', zero_division=0))
```

```
SCORE (Random Forest=4)
------------------------------
Accuracy: 0.90625
F1 Score: 0.9111091114767585
Recall: 0.90625
Precision: 0.9177748226950355
```

2 Features

In [194]:

```
# Define the selected features
selected_features_2C = ['alcohol','citric acid' ]

# Select the desired features from the training and testing sets
xtrain_selected_2C = x_train[selected_features_2C]
xtest_selected_2C = x_test[selected_features_2C]

# Create and train the Random Forest model
classifier= RandomForestClassifier()
classifier.fit(xtrain_selected_2C, y_train)

# Make predictions on the test set
y_pred_2C = classifier.predict(xtest_selected_2C)

# Print the evaluation metrics for the Random Forest model
print("SCORE (Random Forest-2)")
print("------------------------------")
print("Accuracy:", accuracy_score(y_test, y_pred_2C))
print("F1 Score:", f1_score(y_test, y_pred_2C, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_2C, average='weighted'))
print("Precision:", precision_score(y_test, y_pred_2C, average='weighted', zero_division=0))
```

```
SCORE (Random Forest-2)
------------------------------
Accuracy: 0.86875
F1 Score: 0.8769736842105263
Recall: 0.86875
Precision: 0.8872767857142858
```

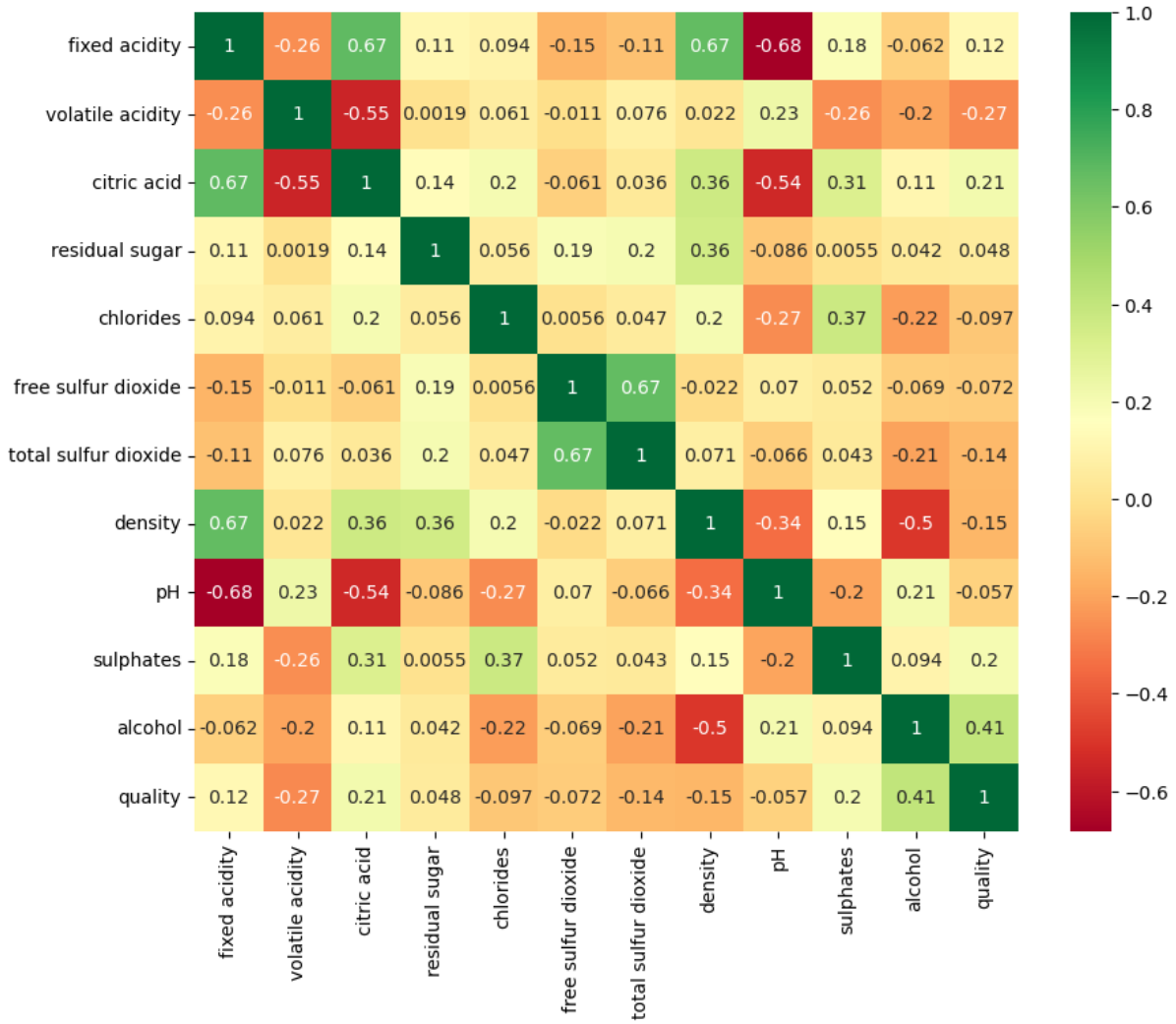
3.Correlation Matrix with Heatmap

Correlation states how the features are related to each other or the target variable. Correlation can be positive (increase in one value of feature increases the value of the target variable) or negative (increase in one value of feature decreases the value of the target variable)

In [195]:

```
# get correlations of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
corrmat

# plot heatmap
fig, ax = plt.subplots(figsize=(10, 8))
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
plt.show()
```

```
<Figure size 2000x2000 with 0 Axes>
```



6 Features

In [196]:

```
# Define the selected features
selected_features_3A = ['alcohol','citric acid' , 'sulphates', 'fixed acidity','residual sugar','pH

# Select the desired features from the training and testing sets
xtrain_selected_3A = x_train[selected_features_3A]
xtest_selected_3A = x_test[selected_features_3A]

# Create and train the Random Forest model
classifier= RandomForestClassifier()
classifier.fit(xtrain_selected_3A, y_train)

# Make predictions on the test set
y_pred_3A = classifier.predict(xtest_selected_3A)

# Print the evaluation metrics for the Random Forest model
print("SCORE (Random Forest-6)")
print("------------------------------")
print("Accuracy:", accuracy_score(y_test, y_pred_3A))
print("F1 Score:", f1_score(y_test, y_pred_3A, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_3A, average='weighted'))
print("Precision:", precision_score(y_test, y_pred_3A, average='weighted', zero_division=0))
```

```
SCORE (Random Forest-6)
------------------------------
Accuracy: 0.940625
F1 Score: 0.9381798756798755
Recall: 0.940625
Precision: 0.936885593220339
```

4 Features

In [197]:

```
# Define the selected features
selected_features_3B = ['alcohol','citric acid' , 'fixed acidity','sulphates']

# Select the desired features from the training and testing sets
xtrain_selected_3B = x_train[selected_features_3B]
xtest_selected_3B = x_test[selected_features_3B]

# Create and train the Random Forest model
classifier= RandomForestClassifier()
classifier.fit(xtrain_selected_3B, y_train)

# Make predictions on the test set
y_pred_3B = classifier.predict(xtest_selected_3B)

# Print the evaluation metrics for the Random Forest model
print("SCORE (Random Forest-4)")
print("------------------------------")
print("Accuracy:", accuracy_score(y_test, y_pred_3B))
print("F1 Score:", f1_score(y_test, y_pred_3B, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_3B, average='weighted'))
print("Precision:", precision_score(y_test, y_pred_3B, average='weighted', zero_division=0))
```

```
SCORE (Random Forest-4)
------------------------------
Accuracy: 0.903125
F1 Score: 0.9075939022166655
Recall: 0.903125
Precision: 0.9134305462706523
```

2 Features

In [198]:

```
# Define the selected features
selected_features_3C = ['alcohol','citric acid']

# Select the desired features from the training and testing sets
xtrain_selected_3C = x_train[selected_features_3C]
xtest_selected_3C = x_test[selected_features_3C]

# Create and train the Random Forest model
classifier= RandomForestClassifier()
classifier.fit(xtrain_selected_3C, y_train)

# Make predictions on the test set
y_pred_3C = classifier.predict(xtest_selected_3C)

# Print the evaluation metrics for the Random Forest model
print("SCORE (Random Forest-2)")
print("------------------------------")
print("Accuracy:", accuracy_score(y_test, y_pred_3C))
print("F1 Score:", f1_score(y_test, y_pred_3C, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_3C, average='weighted'))
print("Precision:", precision_score(y_test, y_pred_3C, average='weighted', zero_division=0))
```

```
SCORE (Random Forest-2)
------------------------------
Accuracy: 0.871875
F1 Score: 0.8792212365796086
Recall: 0.871875
Precision: 0.8883024226663017
```

In [ ]: