# Step 1: Import Neccesary Libraries

In [57]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
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 [59]:

```
#Read dataset
df = pd.read_csv('Data set 3 (99 KB) - winequality.csv')
df
```

Out[59]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.4 | 0.700 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.4 | 5 |
| **1** | 7.8 | 0.880 | 0.00 | 2.6 | 0.098 | 25.0 | 67.0 | 0.99680 | 3.20 | 0.68 | 9.8 | 5 |
| **2** | 7.8 | 0.760 | 0.04 | 2.3 | 0.092 | 15.0 | 54.0 | 0.99700 | 3.26 | 0.65 | 9.8 | 5 |
| **3** | 11.2 | 0.280 | 0.56 | 1.9 | 0.075 | 17.0 | 60.0 | 0.99800 | 3.16 | 0.58 | 9.8 | 6 |
| **4** | 7.4 | 0.700 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.4 | 5 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1594** | 6.2 | 0.600 | 0.08 | 2.0 | 0.090 | 32.0 | 44.0 | 0.99490 | 3.45 | 0.58 | 10.5 | 5 |
| **1595** | 5.9 | 0.550 | 0.10 | 2.2 | 0.062 | 39.0 | 51.0 | 0.99512 | 3.52 | 0.76 | 11.2 | 6 |
| **1596** | 6.3 | 0.510 | 0.13 | 2.3 | 0.076 | 29.0 | 40.0 | 0.99574 | 3.42 | 0.75 | 11.0 | 6 |
| **1597** | 5.9 | 0.645 | 0.12 | 2.0 | 0.075 | 32.0 | 44.0 | 0.99547 | 3.57 | 0.71 | 10.2 | 5 |
| **1598** | 6.0 | 0.310 | 0.47 | 3.6 | 0.067 | 18.0 | 42.0 | 0.99549 | 3.39 | 0.66 | 11.0 | 6 |

1599 rows × 12 columns

In [60]:

```
print("Rows, columns: " + str(df.shape))
```

```
Rows, columns: (1599, 12)
```

In [61]:

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

Out[61]:

| | 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 [62]:

```
df.info()
```

```
<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
```

In [63]:

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

Out[63]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1589** | 6.6 | 0.725 | 0.20 | 7.8 | 0.073 | 29.0 | 79.0 | 0.99770 | 3.29 | 0.54 | 9.2 | 5 |
| **1590** | 6.3 | 0.550 | 0.15 | 1.8 | 0.077 | 26.0 | 35.0 | 0.99314 | 3.32 | 0.82 | 11.6 | 6 |
| **1591** | 5.4 | 0.740 | 0.09 | 1.7 | 0.089 | 16.0 | 26.0 | 0.99402 | 3.67 | 0.56 | 11.6 | 6 |
| **1592** | 6.3 | 0.510 | 0.13 | 2.3 | 0.076 | 29.0 | 40.0 | 0.99574 | 3.42 | 0.75 | 11.0 | 6 |
| **1593** | 6.8 | 0.620 | 0.08 | 1.9 | 0.068 | 28.0 | 38.0 | 0.99651 | 3.42 | 0.82 | 9.5 | 6 |
| **1594** | 6.2 | 0.600 | 0.08 | 2.0 | 0.090 | 32.0 | 44.0 | 0.99490 | 3.45 | 0.58 | 10.5 | 5 |
| **1595** | 5.9 | 0.550 | 0.10 | 2.2 | 0.062 | 39.0 | 51.0 | 0.99512 | 3.52 | 0.76 | 11.2 | 6 |
| **1596** | 6.3 | 0.510 | 0.13 | 2.3 | 0.076 | 29.0 | 40.0 | 0.99574 | 3.42 | 0.75 | 11.0 | 6 |
| **1597** | 5.9 | 0.645 | 0.12 | 2.0 | 0.075 | 32.0 | 44.0 | 0.99547 | 3.57 | 0.71 | 10.2 | 5 |
| **1598** | 6.0 | 0.310 | 0.47 | 3.6 | 0.067 | 18.0 | 42.0 | 0.99549 | 3.39 | 0.66 | 11.0 | 6 |

In [64]:

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

Out[64]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.0000 |
| **mean** | 8.319637 | 0.527821 | 0.270976 | 2.538806 | 0.087467 | 15.874922 | 46.467792 | 0.996747 | 3.3111 |
| **std** | 1.741096 | 0.179060 | 0.194801 | 1.409928 | 0.047065 | 10.460157 | 32.895324 | 0.001887 | 0.1543 |
| **min** | 4.600000 | 0.120000 | 0.000000 | 0.900000 | 0.012000 | 1.000000 | 6.000000 | 0.990070 | 2.7400 |
| **25%** | 7.100000 | 0.390000 | 0.090000 | 1.900000 | 0.070000 | 7.000000 | 22.000000 | 0.995600 | 3.2100 |
| **50%** | 7.900000 | 0.520000 | 0.260000 | 2.200000 | 0.079000 | 14.000000 | 38.000000 | 0.996750 | 3.3100 |
| **75%** | 9.200000 | 0.640000 | 0.420000 | 2.600000 | 0.090000 | 21.000000 | 62.000000 | 0.997835 | 3.4000 |
| **max** | 15.900000 | 1.580000 | 1.000000 | 15.500000 | 0.611000 | 72.000000 | 289.000000 | 1.003690 | 4.0100 |

# Step 3: Data Preprocessing

In [65]:

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

Out[65]:

```
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
```

In [66]:

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

Out[66]:

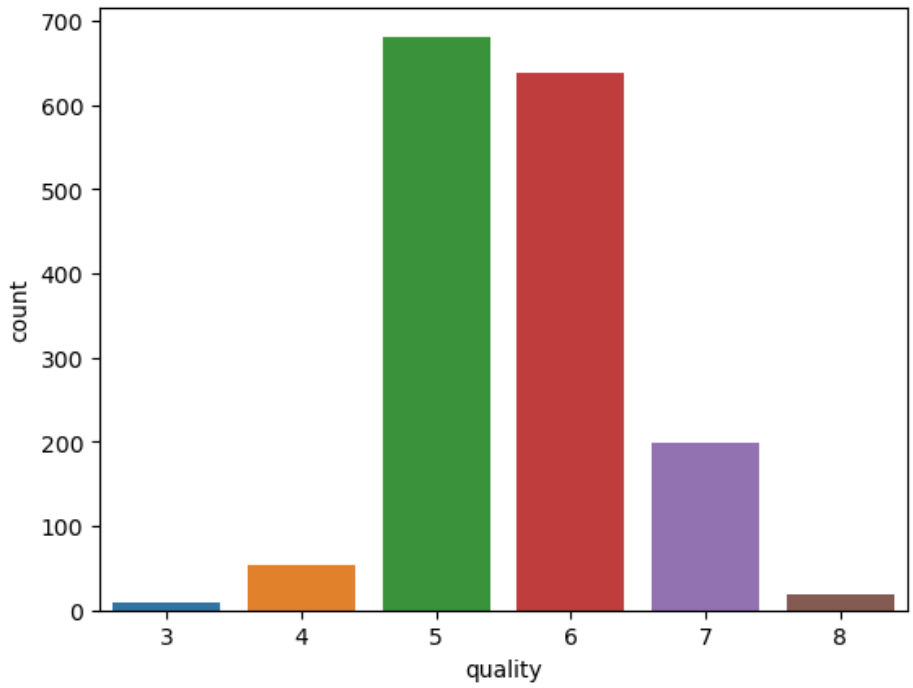
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [67]:

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

Out[67]:

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



In [68]:

```
#assign those feature columns 'fixed acidity', 'volatile acidity', 'citric a
#assign the last column, 'GoodQuality' to y
X = df.iloc[:,0:11]
y = df.iloc[:,-1]
```

In [69]:

```
df.dtypes
```

Out[69]:

```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

In [70]:

```
#showing counts before categorize quality column
df['quality'].value_counts()
```

Out[70]:

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

## Data Encoding/Binarization

In [71]:

```
#binarization of target variable
#assign 1 to good wine quality with quality above or equal to 7
df['quality']=[1 if x>=7 else 0 for x in df['quality']]
#display new target variable
df['quality'].unique()
```

Out[71]:

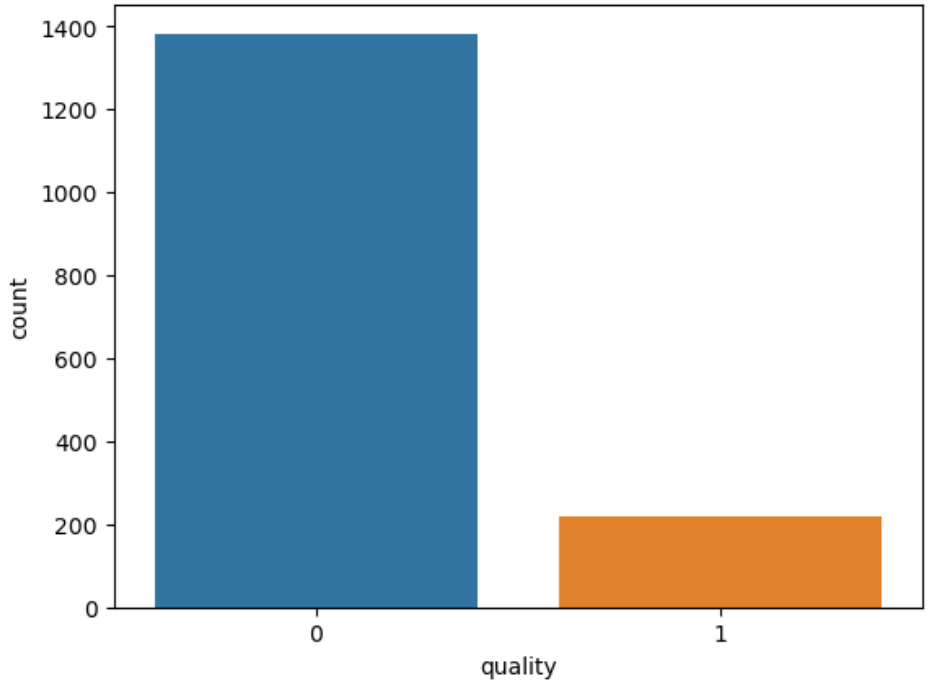
```
array([0, 1], dtype=int64)
```

In [72]:

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

Out[72]:

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




In [73]:

```
#after classification
df['quality'].value_counts()
```

Out[73]:

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

In [74]:

```
df
```

Out[74]:

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

1599 rows × 12 columns

In [75]:

```
#store feature matrix in x and target in y
x = df.drop('quality', axis=1)
y = df['quality']
```

In [76]:

```
x
```

Out[76]:

| | 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 [77]:

```
y
```

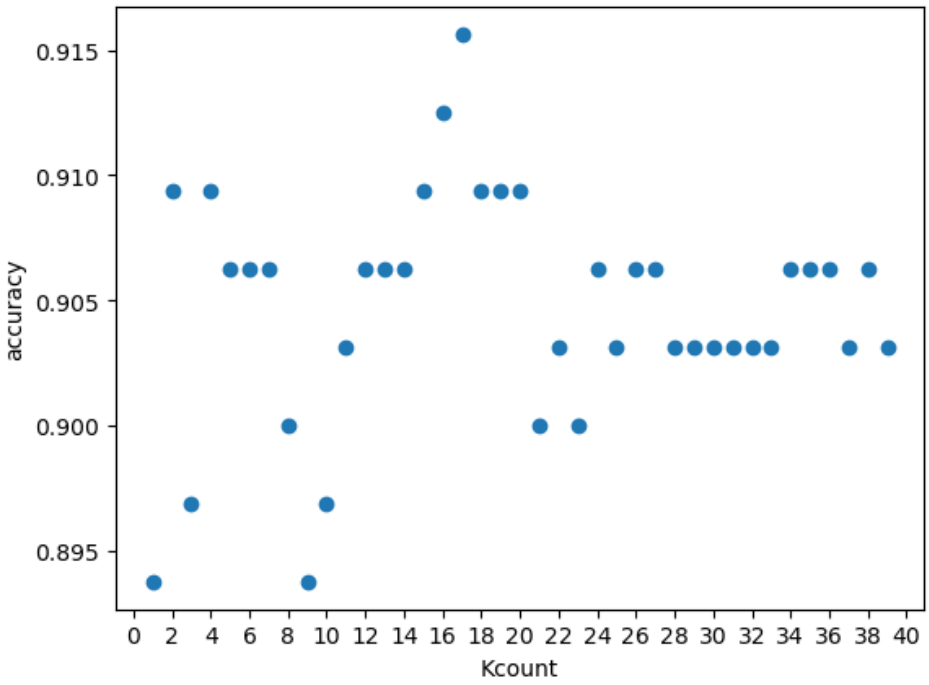
Out[77]:

```
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
```

## Finding value of K with highest accuracy

In [78]:

```
# 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)
#Finding k value with highest acccuracy
k_range = range(1,40)
scores = []
for k in k_range:
    classifier = KNeighborsClassifier(n_neighbors = k)
    classifier.fit(x_train,y_train)
    scores.append(classifier.score(x_test,y_test))


plt.figure()
plt.xlabel("Kcount")
plt.ylabel("accuracy")
plt.scatter(k_range,scores)
plt.grid
plt.xticks([0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40])
plt.show()
```




In [79]:

```
#using root of y_train to find a suitable K value
import math
math.sqrt(len(y_train))
```

Out[79]:

```
35.76310948449533
```

**35 has lower accuracy as compair to 18 as shown in the plotted graph**

**Therefore K=18 has highest accrucay**

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

# Splitting dataset to ratio 60:40

In [80]:

```
# 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 [81]:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
```

In [82]:

```
classifier.fit(x_trainA, y_trainA)
```

Out[82]:

```
KNeighborsClassifier(metric='euclidean', n_neighbors=18)
```

**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 [83]:

```
y_predA = classifier.predict(x_testA)
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_sco
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_testA, y_predA))

print ('Accuracy:', accuracy_score(y_testA, y_predA))
print ('Recall:', recall_score(y_testA, y_predA, average="weighted"))
print ('Precision:', precision_score(y_testA, y_predA, average="weighted"))

confusion = confusion_matrix(y_testA, y_predA)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.90      0.99      0.94       574
           1       0.38      0.08      0.13        66

    accuracy                           0.89       640
   macro avg       0.64      0.53      0.53       640
weighted avg       0.85      0.89      0.86       640

Accuracy: 0.8921875
Recall: 0.8921875
Precision: 0.849282680039259
Confusion matrix:
[[566   8]
 [ 61   5]]
```

# Splitting dataset to ratio 70:30

In [84]:

```
# 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 [85]:

```
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(x_trainB, y_trainB)
```

Out[85]:

```
KNeighborsClassifier(metric='euclidean', n_neighbors=18)
```

**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 [86]:

```
y_predB = classifier.predict(x_testB)
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_scor
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_testB, y_predB))

print ('Accuracy:', accuracy_score(y_testB, y_predB))
print ('Recall:', recall_score(y_testB, y_predB, average="weighted"))
print ('Precision:', precision_score(y_testB, y_predB, average="weighted"))

confusion = confusion_matrix(y_testB, y_predB)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.90      0.99      0.94       430
           1       0.20      0.02      0.04        50

    accuracy                           0.89       480
   macro avg       0.55      0.51      0.49       480
weighted avg       0.82      0.89      0.85       480

Accuracy: 0.8895833333333333
Recall: 0.8895833333333333
Precision: 0.8242543859649122
Confusion matrix:
[[426   4]
 [ 49   1]]
```

# Splitting dataset to ratio 80:20

In [87]:

```
# 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 [88]:

```
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(x_trainC, y_trainC)
```

Out[88]:

```
KNeighborsClassifier(metric='euclidean', n_neighbors=18)
```

**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 [89]:

```
y_predC = classifier.predict(x_testC)
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_scor
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_testC, y_predC))

print ('Accuracy:', accuracy_score(y_testC, y_predC))
print ('Recall:', recall_score(y_testC, y_predC, average="weighted"))
print ('Precision:', precision_score(y_testC, y_predC, average="weighted"))

confusion = confusion_matrix(y_testC, y_predC)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.91      0.99      0.95       290
           1       0.60      0.10      0.17        30

    accuracy                           0.91       320
   macro avg       0.76      0.55      0.56       320
weighted avg       0.88      0.91      0.88       320

Accuracy: 0.909375
Recall: 0.909375
Precision: 0.8848214285714284
Confusion matrix:
[[288   2]
 [ 27   3]]
```

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

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

## Without Feature Selection

In [90]:

```
# Feature Scaling to x_train and x_test to classify better.
#from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [91]:

```
# 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 [92]:

```
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
```

In [93]:

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

Out[93]:

```
KNeighborsClassifier(metric='euclidean', n_neighbors=18)
```

**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 [94]:

```
y_pred = classifier.predict(x_test)
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_sco
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred))

print ('Accuracy:', accuracy_score(y_test, y_pred))
print ('Recall:', recall_score(y_test, y_pred, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred, average="weighted"))

confusion = confusion_matrix(y_test, y_pred)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.91      0.99      0.95       290
           1       0.60      0.10      0.17        30

    accuracy                           0.91       320
   macro avg       0.76      0.55      0.56       320
weighted avg       0.88      0.91      0.88       320

Accuracy: 0.909375
Recall: 0.909375
Precision: 0.8848214285714284
Confusion matrix:
[[288   2]
 [ 27   3]]
```

# 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.

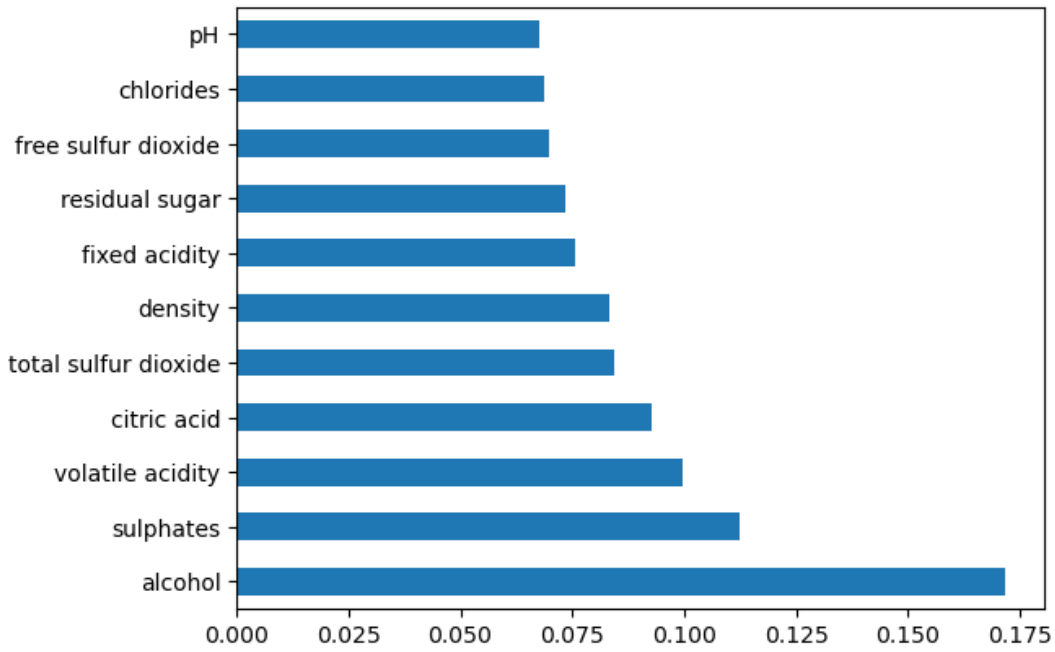
In [95]:

```
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_)
```

```
[0.07560455 0.09977326 0.09275974 0.07356033 0.06871599 0.06968571
 0.08446769 0.0834359  0.06779171 0.11230886 0.17189627]
```

In [96]:

```
# 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 [97]:

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

# 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 KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
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 KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_1A))

print ('Accuracy:', accuracy_score(y_test, y_pred_1A))
print ('Recall:', recall_score(y_test, y_pred_1A, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_1A, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_1A)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.92      0.99      0.95       290
           1       0.67      0.13      0.22        30

    accuracy                           0.91       320
   macro avg       0.79      0.56      0.59       320
weighted avg       0.89      0.91      0.89       320

Accuracy: 0.9125
Recall: 0.9125
Precision: 0.8937101910828027
Confusion matrix:
[[288   2]
 [ 26   4]]
```

## 4 Features

In [98]:

```
# 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 KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
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 KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_1B))

print ('Accuracy:', accuracy_score(y_test, y_pred_1B))
print ('Recall:', recall_score(y_test, y_pred_1B, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_1B, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_1B)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.93      0.94      0.93       290
           1       0.33      0.30      0.32        30

    accuracy                           0.88       320
   macro avg       0.63      0.62      0.62       320
weighted avg       0.87      0.88      0.88       320

Accuracy: 0.878125
Recall: 0.878125
Precision: 0.8725469283276451
Confusion matrix:
[[272  18]
 [ 21   9]]
```

### 2 Features

In [99]:

```
# 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 KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
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 KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_1C))

print ('Accuracy:', accuracy_score(y_test, y_pred_1C))
print ('Recall:', recall_score(y_test, y_pred_1C, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_1C, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_1C)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.93      0.96      0.95       290
           1       0.45      0.30      0.36        30

    accuracy                           0.90       320
   macro avg       0.69      0.63      0.65       320
weighted avg       0.89      0.90      0.89       320

Accuracy: 0.9
Recall: 0.9
Precision: 0.885
Confusion matrix:
[[279  11]
 [ 21   9]]
```

## 2. Univariate Selection

In [100]:

```
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)
```

```
dfscores
```

In [101]:

```
dfcolumns
```

Out[101]:

| | 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 [102]:

```
#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[102]:

| | Specs | Score |
|---|---|---|
| 0 | fixed acidity | 8.393096 |
| 1 | volatile acidity | 7.113769 |
| 2 | citric acid | 10.317077 |
| 3 | residual sugar | 2.856369 |
| 4 | chlorides | 0.383204 |
| 5 | free sulfur dioxide | 56.696032 |
| 6 | total sulfur dioxide | 724.343506 |
| 7 | density | 0.000129 |
| 8 | pH | 0.037747 |
| 9 | sulphates | 2.776190 |
| 10 | alcohol | 28.886089 |

In [103]:

```
print(featureScores.nlargest(6,'Score')) #print 8 best features
```

```
                  Specs       Score
6   total sulfur dioxide  724.343506
5    free sulfur dioxide   56.696032
10               alcohol   28.886089
2            citric acid   10.317077
0          fixed acidity    8.393096
1       volatile acidity    7.113769
```

## 6 Features

In [104]:

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

# 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 KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
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 KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_2A))

print ('Accuracy:', accuracy_score(y_test, y_pred_2A))
print ('Recall:', recall_score(y_test, y_pred_2A, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_2A, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_2A)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.91      1.00      0.96       290
           1       1.00      0.10      0.18        30

    accuracy                           0.92       320
   macro avg       0.96      0.55      0.57       320
weighted avg       0.92      0.92      0.88       320

Accuracy: 0.915625
Recall: 0.915625
Precision: 0.9228115141955836
Confusion matrix:
[[290   0]
 [ 27   3]]
```

## 4 Features

In [105]:

```
# Define the selected features
selected_features_2B = ['total sulfur dioxide','alcohol','sulphates','volatile acidity']


# 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 KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
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 KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_2B))

print ('Accuracy:', accuracy_score(y_test, y_pred_2B))
print ('Recall:', recall_score(y_test, y_pred_2B, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_2B, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_2B)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.91      1.00      0.95       290
           1       0.67      0.07      0.12        30

    accuracy                           0.91       320
   macro avg       0.79      0.53      0.54       320
weighted avg       0.89      0.91      0.87       320

Accuracy: 0.909375
Recall: 0.909375
Precision: 0.8887026813880127
Confusion matrix:
[[289   1]
 [ 28   2]]
```

## 2 Features

In [106]:

```
# 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 KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
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 KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_2C))

print ('Accuracy:', accuracy_score(y_test, y_pred_2C))
print ('Recall:', recall_score(y_test, y_pred_2C, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_2C, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_2C)
print('Confusion matrix:')
print(confusion)
```

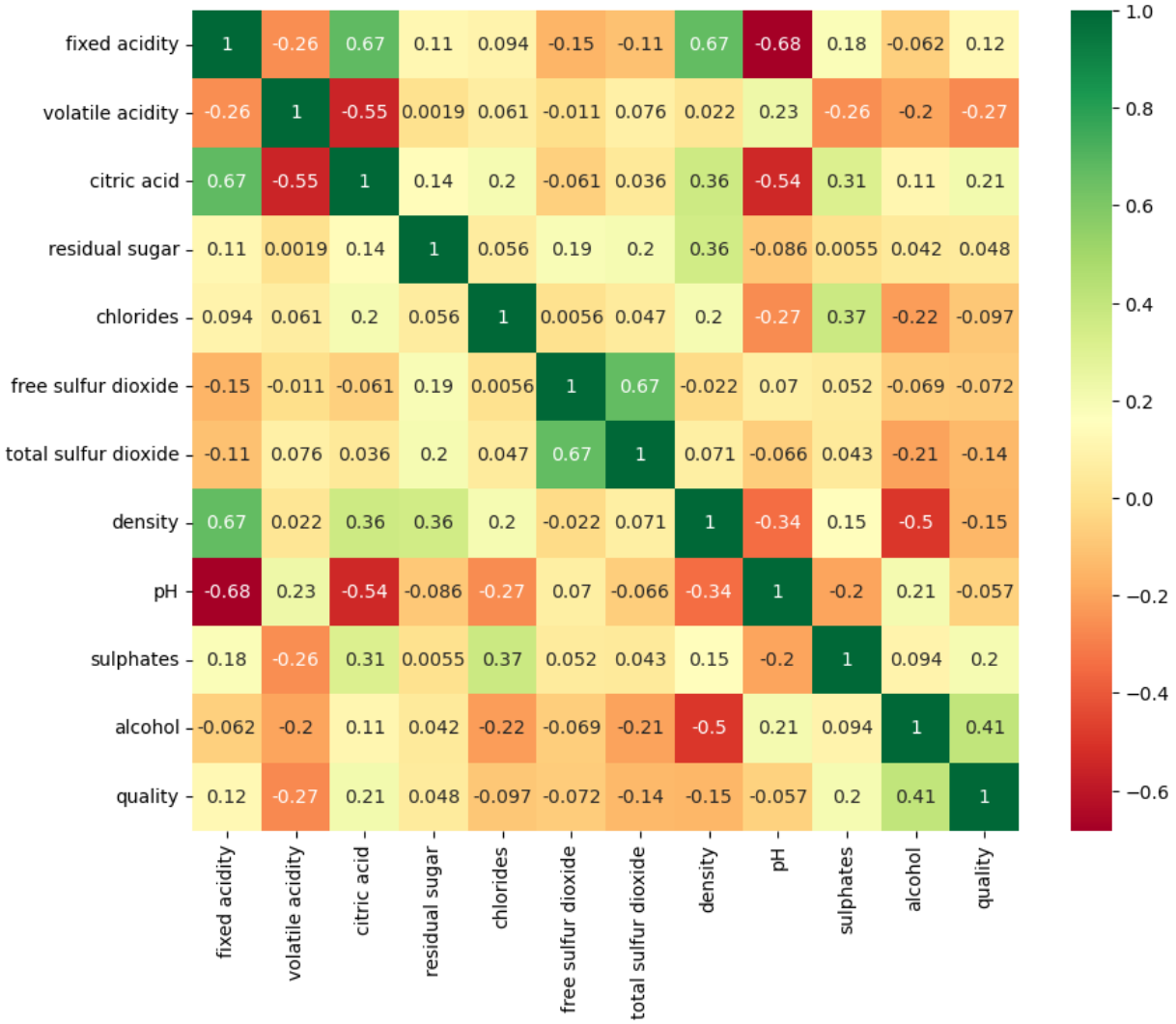
```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       290
           1       0.50      0.03      0.06        30

    accuracy                           0.91       320
   macro avg       0.70      0.51      0.51       320
weighted avg       0.87      0.91      0.87       320

Accuracy: 0.90625
Recall: 0.90625
Precision: 0.8704795597484276
Confusion matrix:
[[289   1]
 [ 29   1]]
```

## 3.Correlation Matrix with Heatmap

In [107]:

```
#get correlations of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
corrmat
#plot heat map
fig, ax = plt.subplots(figsize=(10, 8)) # Adjust the figsize according to your des
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
plt.show()
```

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



## 6 Features

In [108]:

```
# 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 KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
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 KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_3A))

print ('Accuracy:', accuracy_score(y_test, y_pred_3A))
print ('Recall:', recall_score(y_test, y_pred_3A, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_3A, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_3A)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.93      0.96      0.95       290
           1       0.48      0.33      0.39        30

    accuracy                           0.90       320
   macro avg       0.70      0.65      0.67       320
weighted avg       0.89      0.90      0.90       320

Accuracy: 0.903125
Recall: 0.903125
Precision: 0.8902741280458673
Confusion matrix:
[[279  11]
 [ 20  10]]
```

## 4 Features

In [109]:

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

# 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 KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
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 KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_3B))

print ('Accuracy:', accuracy_score(y_test, y_pred_3B))
print ('Recall:', recall_score(y_test, y_pred_3B, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_3B, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_3B)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.94      0.96      0.95       290
           1       0.48      0.37      0.42        30

    accuracy                           0.90       320
   macro avg       0.71      0.66      0.68       320
weighted avg       0.89      0.90      0.90       320

Accuracy: 0.903125
Recall: 0.903125
Precision: 0.8931113672961498
Confusion matrix:
[[278  12]
 [ 19  11]]
```

## 2 Features

In [110]:

```
# 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 KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
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 KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_3C))

print ('Accuracy:', accuracy_score(y_test, y_pred_3C))
print ('Recall:', recall_score(y_test, y_pred_3C, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_3C, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_3C)
print('Confusion matrix:')
print(confusion)
```

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

           0       0.92      0.93      0.93       290
           1       0.30      0.27      0.28        30

    accuracy                           0.87       320
   macro avg       0.61      0.60      0.61       320
weighted avg       0.87      0.87      0.87       320

Accuracy: 0.871875
Recall: 0.871875
Precision: 0.8659817026924536
Confusion matrix:
[[271  19]
 [ 22   8]]
```

In [ ]: