Step 1: Import Neccesary Libraries

In [1]:

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

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
#from sklearn import metrics
#from sklearn.utils import shuffle

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 [2]:

```
cd C:\Users\User\Downloads\Assignment
```

```
C:\Users\User\Downloads\Assignment
```

In [3]:

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

Out[3]:

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

1599 rows × 12 columns

In [4]:

```
df.head(10)
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alc
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	

In [5]:

```
df.tail(10)
```

Out[5]:

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

In [6]:

```
df.shape
```

Out[6]:

```
(1599, 12)
```

In [7]:

```
df.describe()
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	tota
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.

Step 3: Data Preprocessing

In [8]:

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

Out[8]:

```
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 [9]:

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

Out[9]:

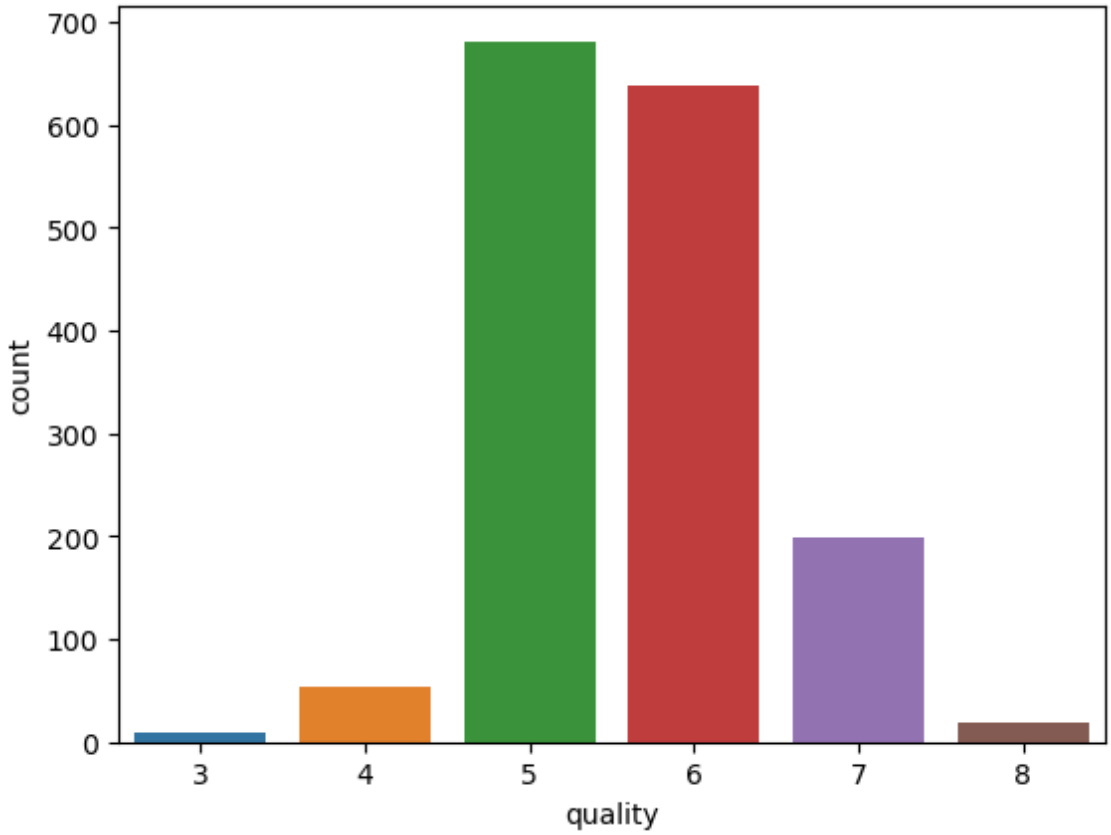
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [10]:

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

Out[10]:

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



In [11]:

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

Out[11]:

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

In [12]:

```
df['quality']= ['1' if i>=7 else '0' for i in df['quality']]
df['quality'] = df['quality'].astype(int)
df['quality'].unique()
```

Out[12]:

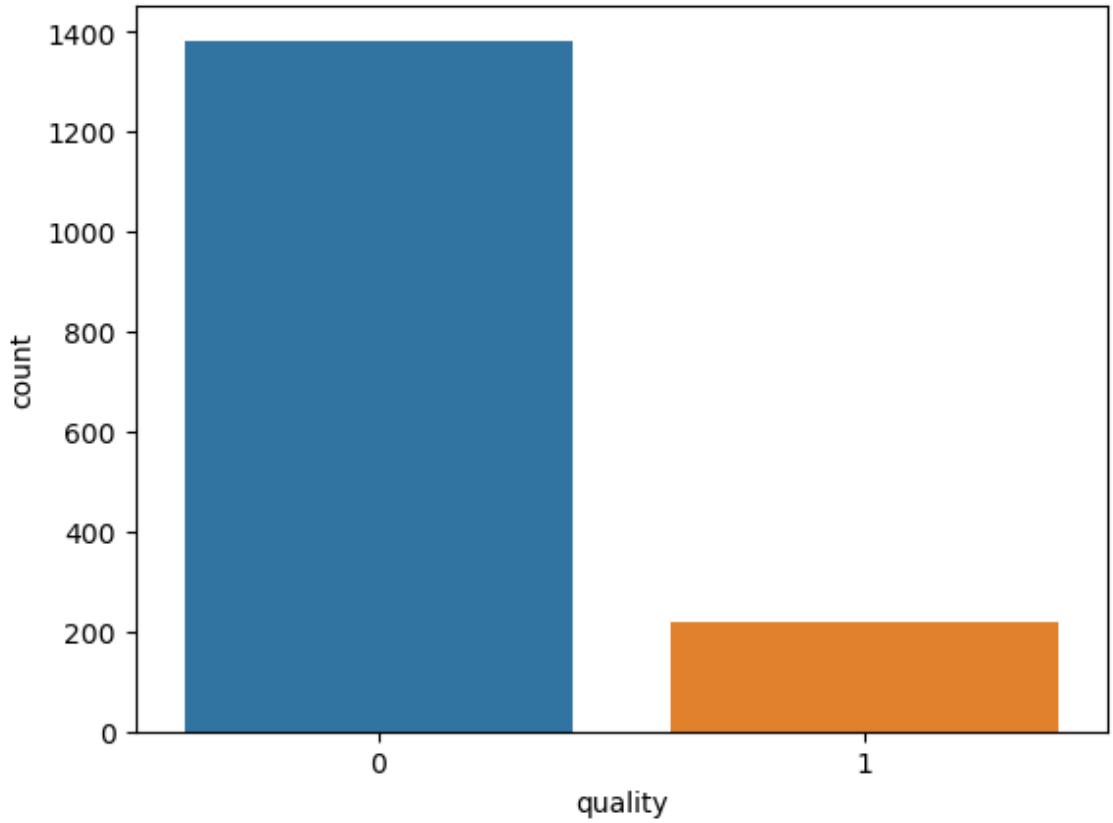
```
array([0, 1])
```

In [13]:

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

Out[13]:

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



In [14]:

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

Out[14]:

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

In [15]:

```
df
```

Out[15]:

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

1599 rows × 12 columns

In [16]:

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

In [17]:

```
x
```

Out[17]:

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

1599 rows × 11 columns

In [18]:

```
y
```

Out[18]:

```
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: int32
```

Step 4: Splitting Dataset Into 60:40, 70:30 And 80:20 Ratios

Feature Scalling

In [19]:

```
# 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)
```

Splitting dataset to ratio 60:40

In [20]:

```
# 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_st

# 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_trainA, y_trainA], axis=1)
test_data.to_csv('test_A.csv', index=False)
```

In [21]:

```
# Building Model (SVM)
model1_SVM = SVC(C=1.0,kernel='rbf', random_state=0)
model2_SVM = SVC(C=1.0,kernel='linear', random_state=0)
model3_SVM = SVC(C=1.0,kernel='poly', random_state=0)
model4_SVM = SVC(C=1.0,kernel='sigmoid', random_state=0)
```

In [22]:

```
model1_SVM.fit(x_trainA, y_trainA) #rbf
model2_SVM.fit(x_trainA, y_trainA) #linear
model3_SVM.fit(x_trainA, y_trainA) #poly
model4_SVM.fit(x_trainA, y_trainA) #sigmoid
```

Out[22]:

```
SVC(kernel='sigmoid', random_state=0)
```

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 [23]:

```
prediction1_SVM = model1_SVM.predict(x_testA) #rbf
prediction2_SVM = model2_SVM.predict(x_testA) #linear
prediction3_SVM = model3_SVM.predict(x_testA) #poly
prediction4_SVM = model4_SVM.predict(x_testA) #sigmoid
```

In [24]:

```
#The accuracy when kernel = rbf

print("SCORE 60:40 (RBF Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testA, prediction1_SVM))
print ('Recall:', recall_score(y_testA, prediction1_SVM, average="weighted"))
print ('Precision:', precision_score(y_testA, prediction1_SVM, average="weighted", zero_
confusion1_SVM = confusion_matrix(y_testA, prediction1_SVM)
print('Confusion matrix:')
print(confusion1_SVM)
```

```
SCORE 60:40 (RBF Kernel)
------------------------------
Accuracy: 0.9
Recall: 0.9
Precision: 0.8884712837837837
Confusion matrix:
[[551  23]
 [ 41  25]]
```

In [25]:

```
#The accuracy when kernel = linear

print("SCORE 60:40 (Linear Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testA, prediction2_SVM))
print ('Recall:', recall_score(y_testA, prediction2_SVM, average="weighted"))
print ('Precision:', precision_score(y_testA, prediction2_SVM, average="weighted", zero_
confusion2_SVM = confusion_matrix(y_testA, prediction2_SVM)
print('Confusion matrix:')
print(confusion2_SVM)
```

```
SCORE 60:40 (Linear Kernel)
------------------------------
Accuracy: 0.896875
Recall: 0.896875
Precision: 0.8043847656249999
Confusion matrix:
[[574   0]
 [ 66   0]]
```

In [26]:

```
#The accuracy when kernel = poly

print("SCORE 60:40 (Poly Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testA, prediction3_SVM))
print ('Recall:', recall_score(y_testA, prediction3_SVM, average="weighted"))
print ('Precision:', precision_score(y_testA, prediction3_SVM, average="weighted", zero_
confusion3_SVM = confusion_matrix(y_testA, prediction3_SVM)
print('Confusion matrix:')
print(confusion3_SVM)
```

```
SCORE 60:40 (Poly Kernel)
------------------------------
Accuracy: 0.896875
Recall: 0.896875
Precision: 0.8914196735395189
Confusion matrix:
[[545  29]
 [ 37  29]]
```

In [27]:

```
#The accuracy when kernel = sigmoid

print("SCORE 60:40 (Sigmoid Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testA, prediction4_SVM))
print ('Recall:', recall_score(y_testA, prediction4_SVM, average="weighted"))
print ('Precision:', precision_score(y_testA, prediction4_SVM, average="weighted", zero_
confusion4_SVM = confusion_matrix(y_testA, prediction4_SVM)
print('Confusion matrix:')
print(confusion4_SVM)
```

```
SCORE 60:40 (Sigmoid Kernel)
------------------------------
Accuracy: 0.859375
Recall: 0.859375
Precision: 0.8217609444768007
Confusion matrix:
[[545  29]
 [ 61   5]]
```

Splitting dataset to ratio 70:30

In [28]:

```
# split dataset into train set and test set
x_trainB, x_testB, y_trainB, y_testB = train_test_split(x, y, test_size = 0.3, random_st

# 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 [29]:

```
# Building Model (SVM)
model1_SVM = SVC(C=1.0,kernel='rbf', random_state=0)
model2_SVM = SVC(C=1.0,kernel='linear', random_state=0)
model3_SVM = SVC(C=1.0,kernel='poly', random_state=0)
model4_SVM = SVC(C=1.0,kernel='sigmoid', random_state=0)
```

In [30]:

```
model1_SVM.fit(x_trainB, y_trainB) #rbf
model2_SVM.fit(x_trainB, y_trainB) #linear
model3_SVM.fit(x_trainB, y_trainB) #poly
model4_SVM.fit(x_trainB, y_trainB) #sigmoid
```

Out[30]:

```
SVC(kernel='sigmoid', random_state=0)
```

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 [31]:

```
prediction1_SVM = model1_SVM.predict(x_testB) #rbf
prediction2_SVM = model2_SVM.predict(x_testB) #linear
prediction3_SVM = model3_SVM.predict(x_testB) #poly
prediction4_SVM = model4_SVM.predict(x_testB) #sigmoid
```

In [32]:

```
#The accuracy when kernel = rbf

print("SCORE 70:30 (RBF Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testB, prediction1_SVM))
print ('Recall:', recall_score(y_testB, prediction1_SVM, average="weighted"))
print ('Precision:', precision_score(y_testB, prediction1_SVM, average="weighted", zero_
confusion1_SVM = confusion_matrix(y_testB, prediction1_SVM)
print('Confusion matrix:')
print(confusion1_SVM)
```

```
SCORE 70:30 (RBF Kernel)
------------------------------
Accuracy: 0.9020833333333333
Recall: 0.9020833333333333
Precision: 0.8832875457875458
Confusion matrix:
[[419  11]
 [ 36  14]]
```

In [33]:

```
#The accuracy when kernel = linear

print("SCORE 70:30 (Linear Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testB, prediction2_SVM))
print ('Recall:', recall_score(y_testB, prediction2_SVM, average="weighted"))
print ('Precision:', precision_score(y_testB, prediction2_SVM, average="weighted", zero_
confusion2_SVM = confusion_matrix(y_testB, prediction2_SVM)
print('Confusion matrix:')
print(confusion2_SVM)
```

```
SCORE 70:30 (Linear Kernel)
------------------------------
Accuracy: 0.8958333333333334
Recall: 0.8958333333333334
Precision: 0.8025173611111112
Confusion matrix:
[[430   0]
 [ 50   0]]
```

In [34]:

```
#The accuracy when kernel = poly

print("SCORE 70:30 (Poly Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testB, prediction3_SVM))
print ('Recall:', recall_score(y_testB, prediction3_SVM, average="weighted"))
print ('Precision:', precision_score(y_testB, prediction3_SVM, average="weighted", zero_
confusion3_SVM = confusion_matrix(y_testB, prediction3_SVM)
print('Confusion matrix:')
print(confusion3_SVM)
```

```
SCORE 70:30 (Poly Kernel)
------------------------------
Accuracy: 0.8958333333333334
Recall: 0.8958333333333334
Precision: 0.883352102102102
Confusion matrix:
[[412  18]
 [ 32  18]]
```

In [35]:

```
#The accuracy when kernel = sigmoid

print("SCORE 70:30 (Sigmoid Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testB, prediction4_SVM))
print ('Recall:', recall_score(y_testB, prediction4_SVM, average="weighted"))
print ('Precision:', precision_score(y_testB, prediction4_SVM, average="weighted", zero_
confusion4_SVM = confusion_matrix(y_testB, prediction4_SVM)
print('Confusion matrix:')
print(confusion4_SVM)
```

```
SCORE 70:30 (Sigmoid Kernel)
------------------------------
Accuracy: 0.8958333333333334
Recall: 0.8958333333333334
Precision: 0.8025173611111112
Confusion matrix:
[[430   0]
 [ 50   0]]
```

Splitting dataset to ratio 80:20

In [36]:

```
# split dataset into train set and test set
x_trainC, x_testC, y_trainC, y_testC = train_test_split(x, y, test_size = 0.2, random_st

# 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 [37]:

```
# Building Model (SVM)
model1_SVM = SVC(C=1.0,kernel='rbf', random_state=0)
model2_SVM = SVC(C=1.0,kernel='linear', random_state=0)
model3_SVM = SVC(C=1.0,kernel='poly', random_state=0)
model4_SVM = SVC(C=1.0,kernel='sigmoid', random_state=0)
```

In [38]:

```
model1_SVM.fit(x_trainC, y_trainC) #rbf
model2_SVM.fit(x_trainC, y_trainC) #linear
model3_SVM.fit(x_trainC, y_trainC) #poly
model4_SVM.fit(x_trainC, y_trainC) #sigmoid
```

Out[38]:

```
SVC(kernel='sigmoid', random_state=0)
```

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 [39]:

```
prediction1_SVM = model1_SVM.predict(x_testC) #rbf
prediction2_SVM = model2_SVM.predict(x_testC) #linear
prediction3_SVM = model3_SVM.predict(x_testC) #poly
prediction4_SVM = model4_SVM.predict(x_testC) #sigmoid
```

In [40]:

```
#The accuracy when kernel = rbf

print("SCORE 80:20 (RBF Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testC, prediction1_SVM))
print ('Recall:', recall_score(y_testC, prediction1_SVM, average="weighted"))
print ('Precision:', precision_score(y_testC, prediction1_SVM, average="weighted", zero_
confusion1_SVM = confusion_matrix(y_testC, prediction1_SVM)
print('Confusion matrix:')
print(confusion1_SVM)
```

```
SCORE 80:20 (RBF Kernel)
------------------------------
Accuracy: 0.915625
Recall: 0.915625
Precision: 0.9001024590163935
Confusion matrix:
[[284   6]
 [ 21   9]]
```

In [41]:

```
#The accuracy when kernel = linear

print("SCORE 80:20 (Linear Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testC, prediction2_SVM))
print ('Recall:', recall_score(y_testC, prediction2_SVM, average="weighted"))
print ('Precision:', precision_score(y_testC, prediction2_SVM, average="weighted", zero_
confusion2_SVM = confusion_matrix(y_testC, prediction2_SVM)
print('Confusion matrix:')
print(confusion2_SVM)
```

```
SCORE 80:20 (Linear Kernel)
------------------------------
Accuracy: 0.90625
Recall: 0.90625
Precision: 0.8212890625
Confusion matrix:
[[290   0]
 [ 30   0]]
```

In [42]:

```
#The accuracy when kernel = poly

print("SCORE 80:20 (Poly Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testC, prediction3_SVM))
print ('Recall:', recall_score(y_testC, prediction3_SVM, average="weighted"))
print ('Precision:', precision_score(y_testC, prediction3_SVM, average="weighted", zero_
confusion3_SVM = confusion_matrix(y_testC, prediction3_SVM)
print('Confusion matrix:')
print(confusion3_SVM)
```

```
SCORE 80:20 (Poly Kernel)
------------------------------
Accuracy: 0.90625
Recall: 0.90625
Precision: 0.8953439597315436
Confusion matrix:
[[279  11]
 [ 19  11]]
```

In [43]:

```
#The accuracy when kernel = sigmoid

print("SCORE 80:20 (Sigmoid Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testC, prediction4_SVM))
print ('Recall:', recall_score(y_testC, prediction4_SVM, average="weighted"))
print ('Precision:', precision_score(y_testC, prediction4_SVM, average="weighted", zero_
confusion4_SVM = confusion_matrix(y_testC, prediction4_SVM)
print('Confusion matrix:')
print(confusion4_SVM)
```

```
SCORE 80:20 (Sigmoid Kernel)
------------------------------
Accuracy: 0.90625
Recall: 0.90625
Precision: 0.8212890625
Confusion matrix:
[[290   0]
 [ 30   0]]
```

Dataset with ratio 80:20 and RBF Kernel has the highest score

In [44]:

```
#read the data set in a dataframe
train = pd.read_csv("train_c.csv")
test = pd.read_csv("test_c.csv")
```

In [45]:

```
xtrain = train.iloc[:,0:11] #only have 12 column
ytrain = train.iloc[:,-1] # last column

xtest = test.iloc[:,0:11] #only have 12 column
ytest = test.iloc[:,-1] # last column
```

In [46]:

```
xtrain
```

Out[46]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	0.469027	0.287671	0.45	0.095890	0.098497	0.211268	0.120141	0.662996	0.511811
1	0.548673	0.095890	0.45	0.164384	0.080134	0.267606	0.151943	0.523495	0.307087
2	0.469027	0.157534	0.55	0.082192	0.083472	0.056338	0.028269	0.516153	0.409449
3	0.088496	0.500000	0.05	0.034247	0.055092	0.154930	0.289753	0.171072	0.645669
4	0.176991	0.414384	0.09	0.315068	0.175292	0.112676	0.038869	0.475771	0.480315
...	...	...	...	...	...	...	...	...	...
1274	0.415929	0.366438	0.26	0.075342	0.140234	0.056338	0.102473	0.536711	0.401575
1275	0.265487	0.373288	0.10	0.041096	0.090150	0.366197	0.173145	0.475771	0.511811
1276	0.292035	0.308219	0.31	0.075342	0.111853	0.126761	0.257951	0.491924	0.433071
1277	0.743363	0.239726	0.49	0.232877	0.121870	0.070423	0.144876	0.883260	0.440945
1278	0.460177	0.589041	0.32	0.095890	0.110184	0.478873	0.515901	0.582232	0.401575

1279 rows × 11 columns

In [47]:

```
xtest
```

Out[47]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	0.548673	0.239726	0.43	0.082192	0.265442	0.366197	0.212014	0.596916	0.338583
1	0.309735	0.479452	0.00	0.219178	0.138564	0.056338	0.028269	0.621880	0.488189
2	0.398230	0.116438	0.33	0.078767	0.085142	0.169014	0.074205	0.373715	0.409449
3	0.495575	0.359589	0.36	0.061644	0.068447	0.056338	0.028269	0.596916	0.338583
4	0.672566	0.226027	0.49	0.034247	0.105175	0.028169	0.000000	0.501468	0.307087
...	...	...	...	...	...	...	...	...	...
315	0.486726	0.102740	0.54	0.095890	0.088481	0.084507	0.070671	0.384728	0.338583
316	0.203540	0.184932	0.24	0.082192	0.150250	0.042254	0.003534	0.334068	0.551181
317	0.398230	0.150685	0.42	0.061644	0.076795	0.112676	0.042403	0.282673	0.346457
318	0.398230	0.441781	0.04	0.047945	0.110184	0.042254	0.028269	0.582232	0.433071
319	0.318584	0.136986	0.42	0.095890	0.143573	0.028169	0.010601	0.366373	0.417323

320 rows × 11 columns

In [48]:

```
ytrain
```

Out[48]:

```
0       0
1       0
2       0
3       1
4       0
       ..
1274    0
1275    0
1276    0
1277    0
1278    0
Name: quality, Length: 1279, dtype: int64
```

In [49]:

```
ytest
```

Out[49]:

```
0      0
1      0
2      1
3      0
4      0
      ..
315    0
316    0
317    0
318    0
319    0
Name: quality, Length: 320, dtype: int64
```

Step 5: Feature Selection

1. Feature Importance

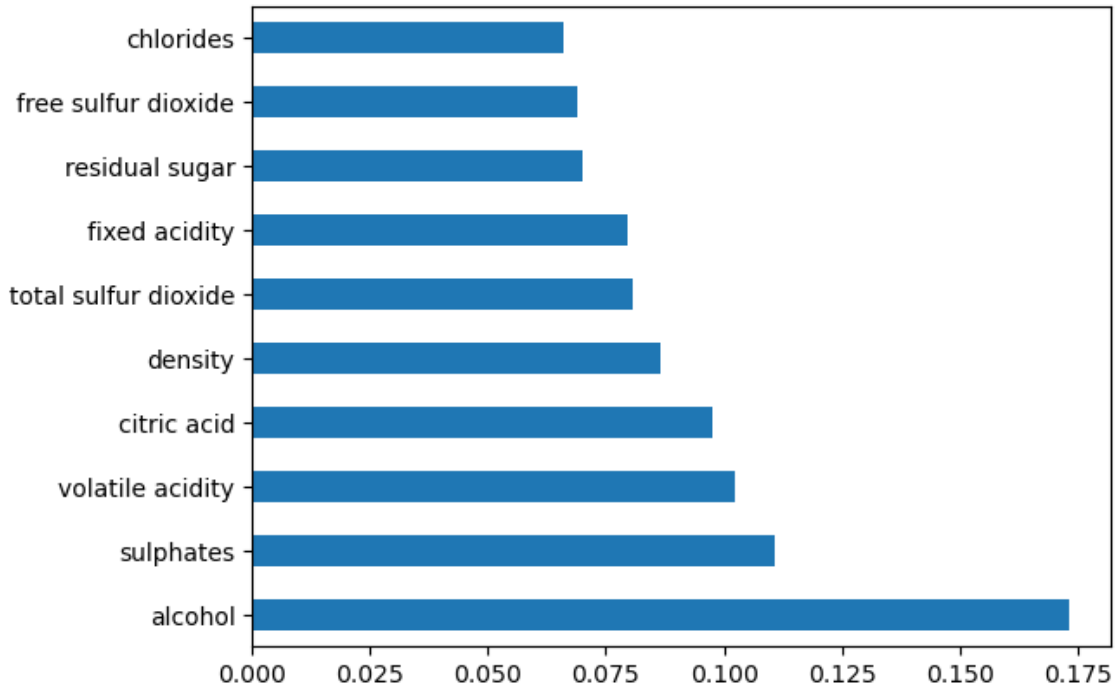
In [50]:

```
model = ExtraTreesClassifier()
model.fit(xtrain,ytrain)
print(model.feature_importances_) #use inbuilt class feature_importances of tree based c
```

```
[0.07941624 0.10225754 0.09767855 0.07009993 0.06600554 0.06888065
 0.08069947 0.08660359 0.06439893 0.11062581 0.17333375]
```

In [51]:

```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=xtrain.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```



2. Univariate Selection

In [52]:

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

In [53]:

```
dfscores
```

Out[53]:

	0
0	1.501255
1	5.944289
2	9.243436
3	0.229023
4	0.590600
5	0.659115
6	2.227811
7	1.202066
8	0.112655
9	3.058817
10	19.291231

In [54]:

```
dfcolumns
```

Out[54]:

	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 [55]:

```
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
featureScores
```

Out[55]:

	Specs	Score
0	fixed acidity	1.501255
1	volatile acidity	5.944289
2	citric acid	9.243436
3	residual sugar	0.229023
4	chlorides	0.590600
5	free sulfur dioxide	0.659115
6	total sulfur dioxide	2.227811
7	density	1.202066
8	pH	0.112655
9	sulphates	3.058817
10	alcohol	19.291231

In [56]:

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

```
                  Specs      Score
10              alcohol  19.291231
2           citric acid   9.243436
1      volatile acidity   5.944289
9             sulphates   3.058817
6   total sulfur dioxide   2.227811
0         fixed acidity   1.501255
```

3.Correlation Matrix with Heatmap

In [57]:

```
#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 desired
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
plt.show()
```

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

Step 6: Modelling and Model Evaluation

Without Feature Selection

In [58]:

```
#SVM RBF Kernel

classfier = SVC(kernel='rbf', C=1)
classfier.fit(xtrain, ytrain)

y_predict = classfier.predict(xtest)
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_predict))
print("F1 Score: ", f1_score(ytest, y_predict, average='weighted'))
print("Recall: ", recall_score(ytest, y_predict, average='weighted'))
print("Precision: ", precision_score(ytest, y_predict, average='weighted', zero_divisior
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.915625
F1 Score:  0.9026260504201682
Recall:  0.915625
Precision:  0.9001024590163935
```

With Feature Selection

6 Features

I) Feature Importance

In [59]:

```
rform feature selection and store the selected feature indices in a list
cted_features_1A = [ 'alcohol','sulphates','volatile acidity','citric acid','density','t

lect the desired features from the training and testing sets
in_selected_1A = xtrain[selected_features_1A]
t_selected_1A = xtest[selected_features_1A]

eate and train the linear SVM model
l_linear = SVC(kernel='rbf', C=1)
l_linear.fit(xtrain_selected_1A, ytrain)

ke predictions on the test set
ed_1A = model_linear.predict(xtest_selected_1A)

int the evaluation metrics for the linear SVM model
t("SCORE (RBF Kernel)")
t("------------------------------")
t("Accuracy: ", accuracy_score(ytest, y_pred_1A))
t("F1 Score: ", f1_score(ytest, y_pred_1A, average='weighted'))
t("Recall: ", recall_score(ytest, y_pred_1A, average='weighted'))
t("Precision: ", precision_score(ytest, y_pred_1A, average='weighted', zero_division=0))
t('Confusion matrix: \n',confusion_matrix(ytest, y_pred_1A))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.921875
F1 Score:  0.907544018542324
Recall:  0.921875
Precision:  0.9091628038085693
Confusion matrix:
 [[286   4]
 [ 21   9]]
```

II) Univariate Selection

In [60]:

```
# Perform feature selection and store the selected feature indices in a list
selected_features_2A = [ 'alcohol','citric acid','volatile acidity','sulphates','total s

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

# Create and train the linear SVM model
model_linear = SVC(kernel='rbf', C=1)
model_linear.fit(xtrain_selected_2A, ytrain)

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

# Print the evaluation metrics for the linear SVM model
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_pred_2A))
print("F1 Score: ", f1_score(ytest, y_pred_2A, average='weighted'))
print("Recall: ", recall_score(ytest, y_pred_2A, average='weighted'))
print("Precision: ", precision_score(ytest, y_pred_2A, average='weighted', zero_divisior
print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_2A))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.925
F1 Score:  0.9123703477730322
Recall:  0.925
Precision:  0.913982259570495
Confusion matrix:
 [[286   4]
 [ 20  10]]
```

III)Correlation Matrix with Heatmap

In [61]:

```
# Perform feature selection and store the selected feature indices in a list
selected_features_3A = [ 'alcohol','sulphates','citric acid','fixed acidity','residual s

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

# Create and train the linear SVM model
model_linear = SVC(kernel='rbf', C=1)
model_linear.fit(xtrain_selected_3A, ytrain)

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

# Print the evaluation metrics for the linear SVM model
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_pred_3A))
print("F1 Score: ", f1_score(ytest, y_pred_3A, average='weighted'))
print("Recall: ", recall_score(ytest, y_pred_3A, average='weighted'))
print("Precision: ", precision_score(ytest, y_pred_3A, average='weighted', zero_divisior
print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_3A))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.921875
F1 Score:  0.905011910094059
Recall:  0.921875
Precision:  0.9099091644601354
Confusion matrix:
 [[287   3]
 [ 22   8]]
```

Step 8: Use different number of features for prediction

4 Features

I) Feature Importance

In [62]:

```
# Perform feature selection and store the selected feature indices in a list
selected_features_1B = [ 'alcohol','sulphates','volatile acidity','citric acid']  # Repl

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

# Create and train the linear SVM model
model_linear = SVC(kernel='rbf', C=1)
model_linear.fit(xtrain_selected_1B, ytrain)

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

# Print the evaluation metrics for the linear SVM model
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_pred_1B))
print("F1 Score: ", f1_score(ytest, y_pred_1B, average='weighted'))
print("Recall: ", recall_score(ytest, y_pred_1B, average='weighted'))
print("Precision: ", precision_score(ytest, y_pred_1B, average='weighted', zero_divisior
print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_1B))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.909375
F1 Score:  0.9000461859870852
Recall:  0.909375
Precision:  0.8953761584193041
Confusion matrix:
 [[281   9]
 [ 20  10]]
```

II) Univariate Selection

In [63]:

```
# Perform feature selection and store the selected feature indices in a list
selected_features_2B = [ 'alcohol','citric acid','volatile acidity','sulphates']  # Repl

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

# Create and train the linear SVM model
model_linear = SVC(kernel='rbf', C=1)
model_linear.fit(xtrain_selected_2B, ytrain)

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

# Print the evaluation metrics for the linear SVM model
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_pred_2B))
print("F1 Score: ", f1_score(ytest, y_pred_2B, average='weighted'))
print("Recall: ", recall_score(ytest, y_pred_2B, average='weighted'))
print("Precision: ", precision_score(ytest, y_pred_2B, average='weighted', zero_divisior
print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_2B))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.909375
F1 Score:  0.9000461859870852
Recall:  0.909375
Precision:  0.8953761584193041
Confusion matrix:
 [[281   9]
 [ 20  10]]
```

III)Correlation Matrix with Heatmap

In [64]:

```
# Perform feature selection and store the selected feature indices in a list
selected_features_3B = [ 'alcohol','sulphates','citric acid','fixed acidity']  # Replace

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

# Create and train the linear SVM model
model_linear = SVC(kernel='rbf', C=1)
model_linear.fit(xtrain_selected_3B, ytrain)

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

# Print the evaluation metrics for the linear SVM model
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_pred_3B))
print("F1 Score: ", f1_score(ytest, y_pred_3B, average='weighted'))
print("Recall: ", recall_score(ytest, y_pred_3B, average='weighted'))
print("Precision: ", precision_score(ytest, y_pred_3B, average='weighted', zero_divisior
print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_3B))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.91875
F1 Score:  0.9050678767541184
Recall:  0.91875
Precision:  0.9043242296918768
Confusion matrix:
 [[285   5]
 [ 21   9]]
```

2 Features

`I) Feature Importance

In [65]:

```
# Perform feature selection and store the selected feature indices in a list
selected_features_1C = [ 'alcohol','sulphates']  # Replace with the selected features

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

# Create and train the linear SVM model
model_linear = SVC(kernel='rbf', C=1)
model_linear.fit(xtrain_selected_1C, ytrain)

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

# Print the evaluation metrics for the linear SVM model
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_pred_1C))
print("F1 Score: ", f1_score(ytest, y_pred_1C, average='weighted'))
print("Recall: ", recall_score(ytest, y_pred_1C, average='weighted'))
print("Precision: ", precision_score(ytest, y_pred_1C, average='weighted', zero_divisior
print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_1C))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.909375
F1 Score:  0.8927510615090958
Recall:  0.909375
Precision:  0.8888358180907041
Confusion matrix:
 [[284   6]
 [ 23   7]]
```

II) Univariate Selection

In [66]:

```
# Perform feature selection and store the selected feature indices in a list
selected_features_2C = [ 'alcohol','citric acid']  # Replace with the selected features

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

# Create and train the linear SVM model
model_linear = SVC(kernel='rbf', C=1)
model_linear.fit(xtrain_selected_2C, ytrain)

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

# Print the evaluation metrics for the linear SVM model
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_pred_2C))
print("F1 Score: ", f1_score(ytest, y_pred_2C, average='weighted'))
print("Recall: ", recall_score(ytest, y_pred_2C, average='weighted'))
print("Precision: ", precision_score(ytest, y_pred_2C, average='weighted', zero_divisior
print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_2C))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.890625
F1 Score:  0.8793660865361372
Recall:  0.890625
Precision:  0.8715411348137787
Confusion matrix:
 [[278  12]
 [ 23   7]]
```

III)Correlation Matrix with Heatmap

In [67]:

```
# Perform feature selection and store the selected feature indices in a list
selected_features_3C = [ 'alcohol','citric acid']  # Replace with the selected features

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

# Create and train the linear SVM model
model_linear = SVC(kernel='rbf', C=1)
model_linear.fit(xtrain_selected_3C, ytrain)

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

# Print the evaluation metrics for the linear SVM model
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_pred_3C))
print("F1 Score: ", f1_score(ytest, y_pred_3C, average='weighted'))
print("Recall: ", recall_score(ytest, y_pred_3C, average='weighted'))
print("Precision: ", precision_score(ytest, y_pred_3C, average='weighted', zero_divisior
print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_3C))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.909375
F1 Score:  0.8927510615090958
Recall:  0.909375
Precision:  0.8888358180907041
Confusion matrix:
 [[284   6]
 [ 23   7]]
```

Six features selected using Univariate Selection will have the highest score in Model Evaluation

Step 9: Use different number of cost

In [68]:

```
# Perform feature selection and store the selected feature indices in a list
selected_features_2A = [ 'alcohol','citric acid','volatile acidity','sulphates','total s

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

costs= [1,100,1000]#using different cost value

for cost in costs:
    model_linear = SVC(kernel='rbf', C=cost)
    model_linear.fit(xtrain_selected_2A, ytrain)
    #perform prediction
    # Make predictions on the test set
    y_pred_2A = model_linear.predict(xtest_selected_2A)

    # Print the evaluation metrics for the linear SVM model
    print(f"SCORE (RBF Kernel) when cost= {cost}")
    print("------------------------------")
    print("Accuracy: ", accuracy_score(ytest, y_pred_2A))
    print("F1 Score: ", f1_score(ytest, y_pred_2A, average='weighted'))
    print("Recall: ", recall_score(ytest, y_pred_2A, average='weighted'))
    print("Precision: ", precision_score(ytest, y_pred_2A, average='weighted', zero_divi
    print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_2A))
```

```
SCORE (RBF Kernel) when cost= 1
------------------------------
Accuracy:  0.925
F1 Score:  0.9123703477730322
Recall:  0.925
Precision:  0.913982259570495
Confusion matrix:
 [[286   4]
 [ 20  10]]
SCORE (RBF Kernel) when cost= 100
------------------------------
Accuracy:  0.915625
F1 Score:  0.9195172696725795
Recall:  0.915625
Precision:  0.9249027074777958
Confusion matrix:
 [[273  17]
 [ 10  20]]
SCORE (RBF Kernel) when cost= 1000
------------------------------
Accuracy:  0.9125
F1 Score:  0.9160199556541018
Recall:  0.9125
Precision:  0.9206279342723004
Confusion matrix:
 [[273  17]
 [ 11  19]]
```