WINE QUALITY PREDICTION

In [2]:
```
import warnings
warnings.simplefilter("ignore")
```

In [3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import scipy.stats as stats
from scipy.stats import zscore
```

In [4]:
```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

IMPORTING DATA

In [4]:
```
WD = pd.read_csv('QualityPrediction.csv')
WD.head()
```

Out[4]:

	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

EXPLORATORY DATA ANALYSIS

In [5]:
```
WD.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 [6]:
```
WD.shape
```

Out[6]:
```
(1599, 12)
```

In [7]:
```
WD.isnull().sum()
```

Out[7]:
```
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 [8]:
```
WD.describe()
```

Out[8]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000

Observation:

(a) There is a big gap between 75% and max values of residual sugar, free sulfur dioxide and total sulfur dioxide.

(b) This is indicative of outliers.

Renaming Columns

In [9]:
```
WD = WD.rename(columns={'fixed acidity': 'fixed_acidity', 'volatile acidity': 'volatile_acidity', 'citric acid
                        'residual sugar': 'residual_sugar', 'free sulfur dioxide': 'free_sulfur_dioxide', 'to
WD.head()
```

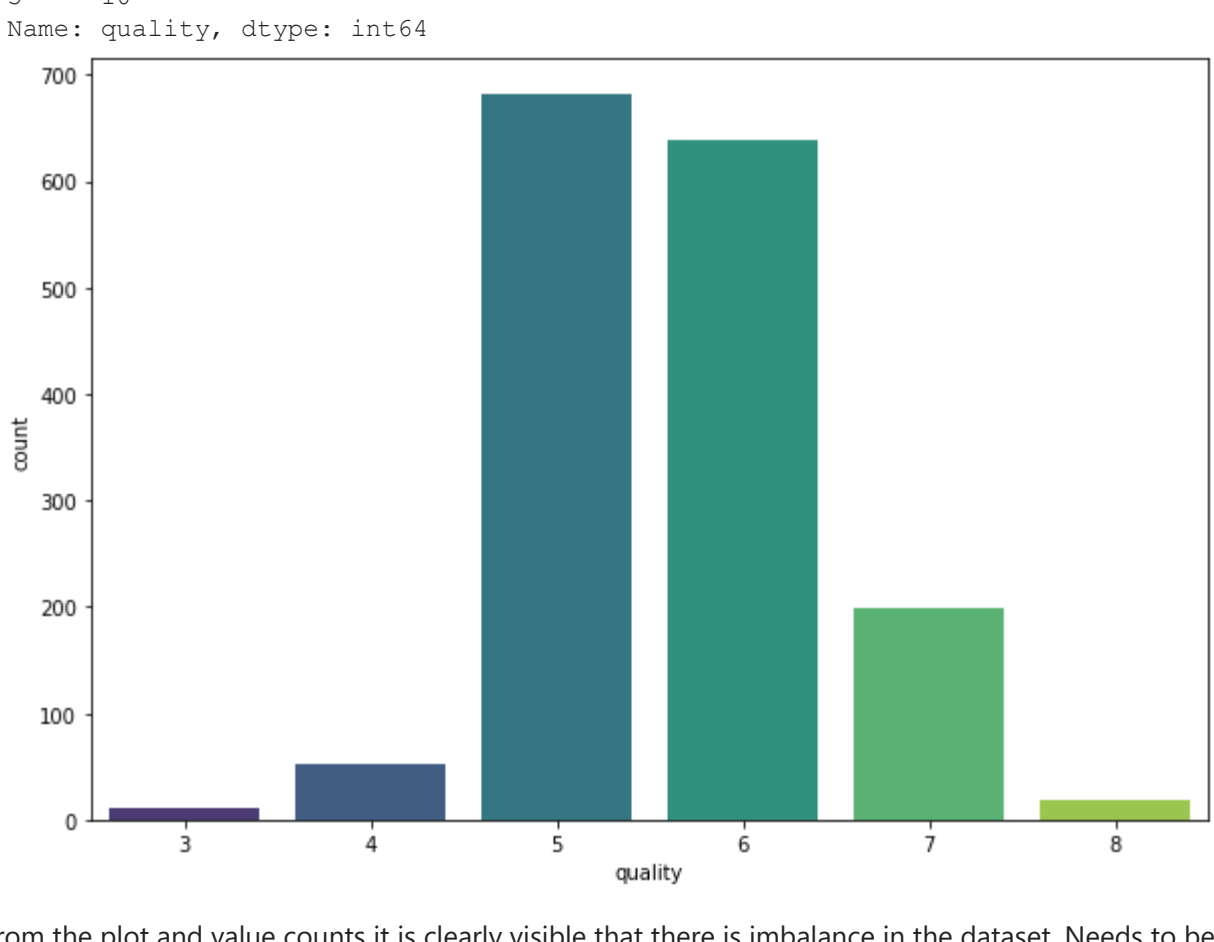
Out[9]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

Visualisation

In [10]:
```
plt.figure(figsize=(10,7))
sns.countplot(WD['quality'],palette='viridis')
WD['quality'].value_counts()
```

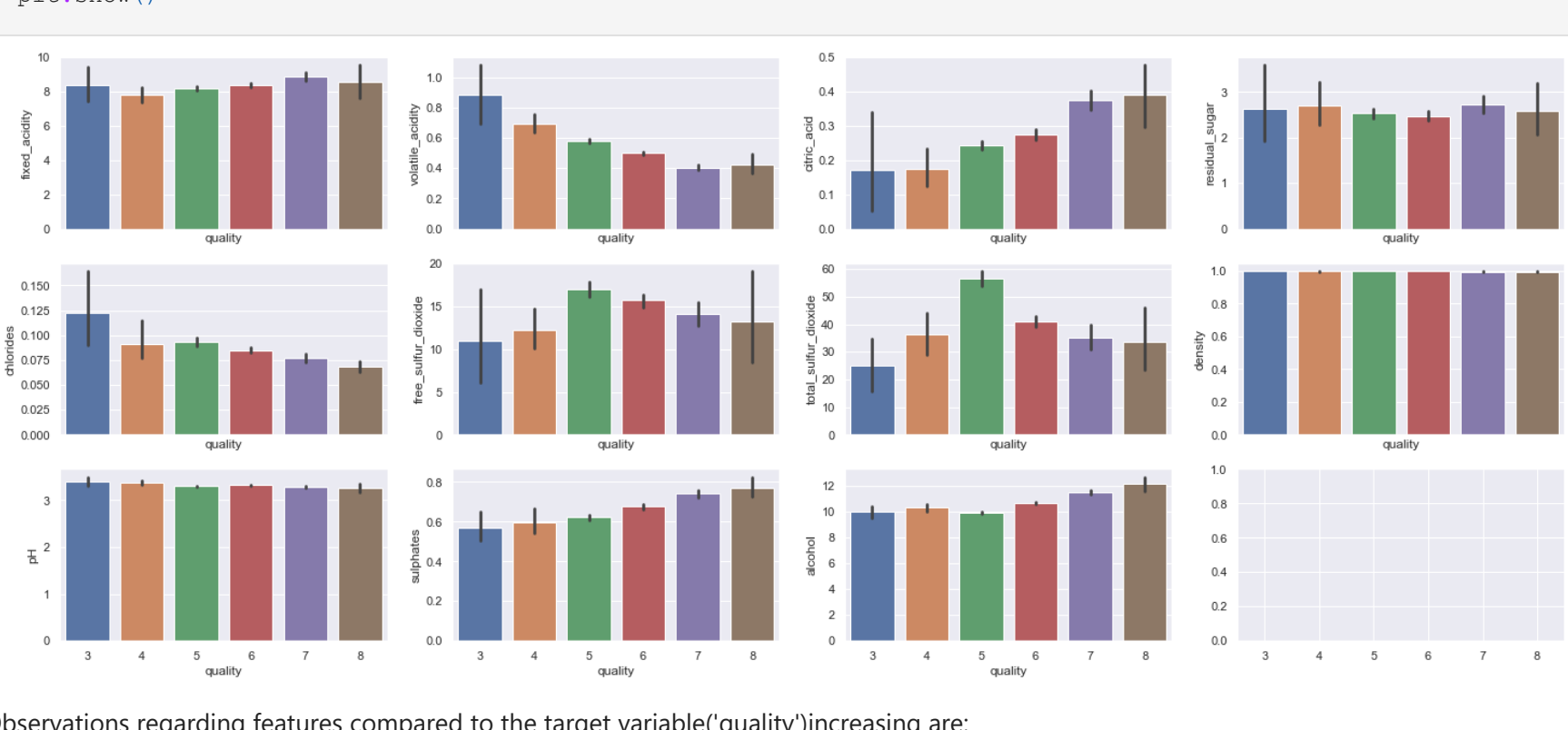
Out[10]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```



From the plot and value counts it is clearly visible that there is imbalance in the dataset. Needs to be rectified.

Plotting Feature variables against Target Label

In [11]:
```
sns.set()
fig, axes = plt.subplots(3, 4,sharex=True, figsize=(25,10))

#create chart in each subplot
sns.barplot(data=WD, x='quality' , y='fixed_acidity',   ax=axes[0,0])
sns.barplot(data=WD, x='quality', y='volatile_acidity', ax=axes[0,1])
sns.barplot(data=WD, x='quality', y='citric_acid', ax=axes[0,2])
sns.barplot(data=WD, x='quality', y='residual_sugar', ax=axes[0,3])
sns.barplot(data=WD, x='quality', y='chlorides', ax=axes[1,0])
sns.barplot(data=WD, x='quality', y='free_sulfur_dioxide', ax=axes[1,1])
sns.barplot(data=WD, x='quality', y='total_sulfur_dioxide', ax=axes[1,2])
sns.barplot(data=WD, x='quality', y='density', ax=axes[1,3])
sns.barplot(data=WD, x='quality', y='pH', ax=axes[2,0])
sns.barplot(data=WD, x='quality', y='sulphates', ax=axes[2,1])
sns.barplot(data=WD, x='quality', y='alcohol', ax=axes[2,2])
plt.tight_layout
plt.show()
```



Observations regarding features compared to the target variable('quality')increasing are:

1. fixed acidity - no pattern
2. volatile acidity - negative trend
3. citric - acid positive trend
4. residual sugar - no pattern
5. chlorides - negative trend
6. free sulfur dioxide - no fixed pattern
7. total sulfur dioxide - no fixed pattern
8. density - more or less consistant
9. pH - more or less constsant
10. sulphates- positive trend
11. alcohol - positive trend

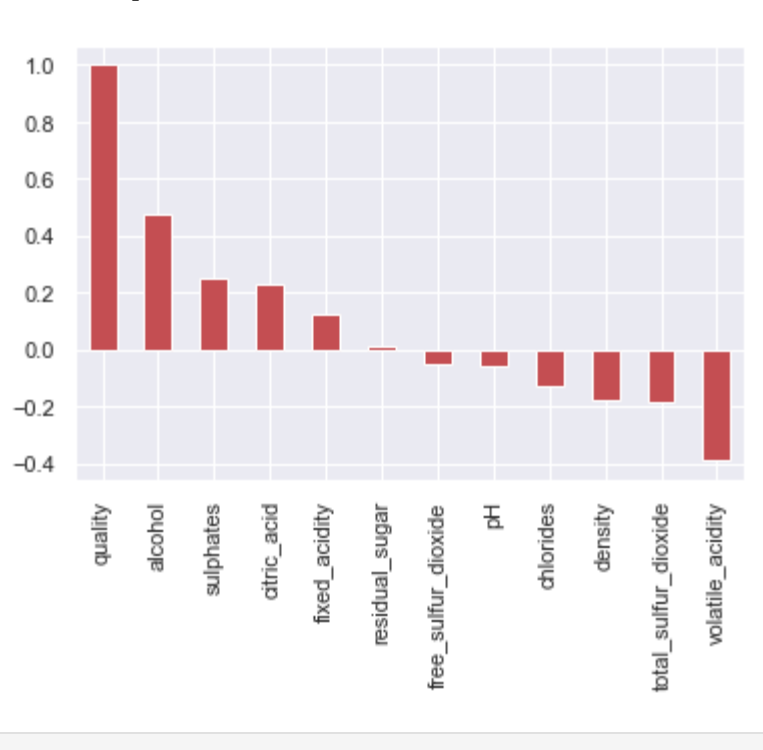
Correlations

In [12]:
```
# External Corr
correlations = WD.corr()['quality'].sort_values(ascending=False)
print(correlations)
```

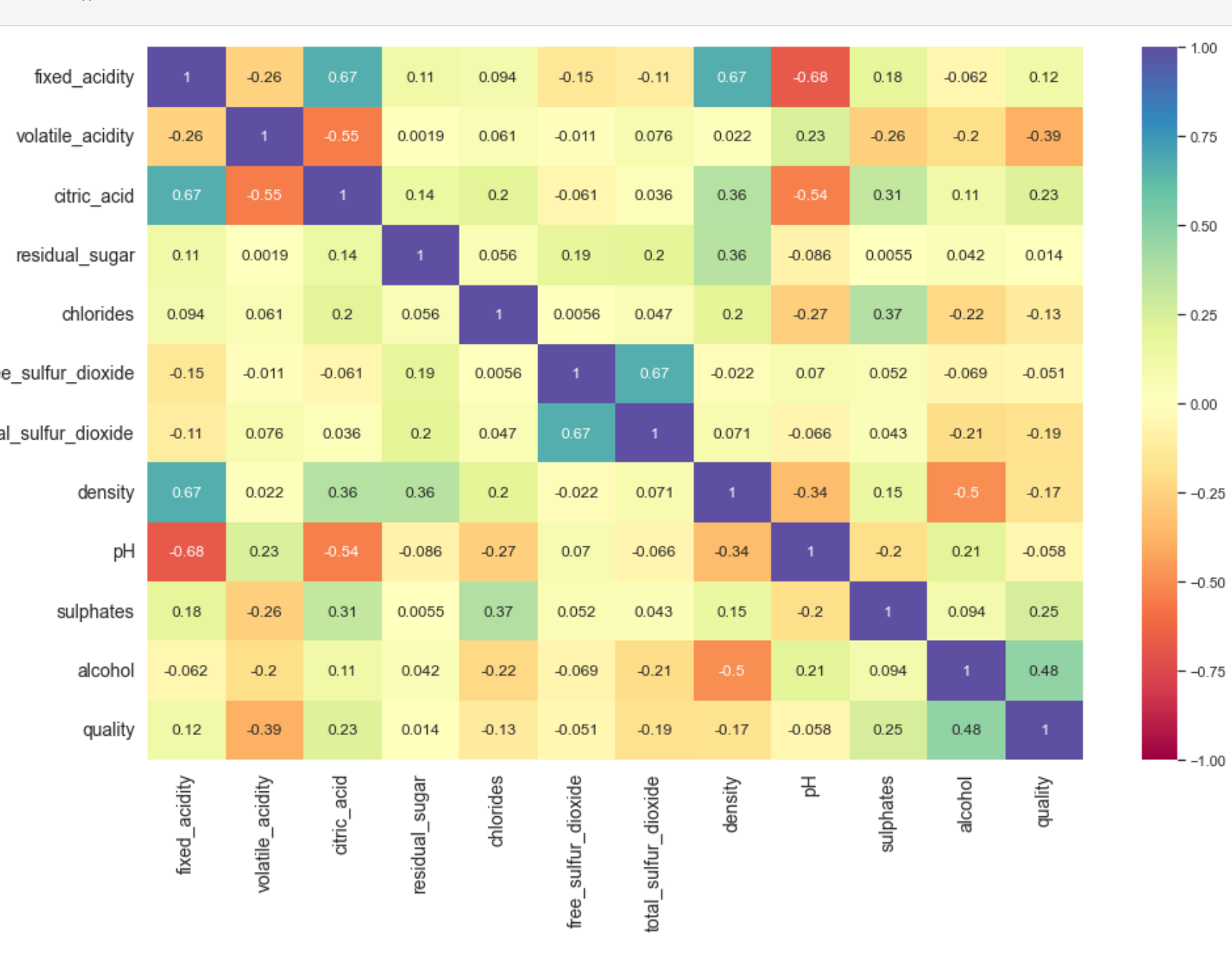
```
quality                 1.000000
alcohol                 0.476166
sulphates               0.251397
citric_acid             0.226373
fixed_acidity           0.124052
residual_sugar          0.013732
free_sulfur_dioxide    -0.050656
pH                     -0.057731
chlorides              -0.128907
density                -0.174919
total_sulfur_dioxide   -0.185100
volatile_acidity       -0.390558
Name: quality, dtype: float64
```

In [13]:
```
correlations.plot(kind='bar',color ='r')
```

Out[13]:
```
<AxesSubplot:>
```



In [14]:
```
# Internal Corr
plt.figure(figsize=(16,10))
sns.heatmap(WD.corr(),vmin=-1,vmax=1, annot=True,cmap = 'Spectral', annot_kws={'size': 12})
sns.set_style('darkgrid')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.show()
```



1. Fixed acidity and citric acid are positively correlated with a value of 0.672 which is close to 1. Similary, fixed acidity and density are positively correlated with a value of 0.668 . The other 2 column that's positively correlated are free sulfur dioxide and total sulfur dioxide with a value of 0.668 . The only negatively correlated columns are fixed acitidy and pH with a value -0.683 being close to the value -1. Quality column is least correlated with residual sugar showing a coefficient value of 0.014.
2. Need to check for multicollinearity.

PRE PROCESSING

Dropping Columns

Free_sulfur_dioxide can be dropped . Also fixed acidity can be dropped to obviate multicollinearity as it has relatively strong corr with citric_acid,density and pH.

In [15]:
```
WD = WD.drop(['free_sulfur_dioxide','fixed_acidity'], axis=1)
WD
```

Out[15]:

	volatile_acidity	citric_acid	residual_sugar	chlorides	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality
0	0.700	0.00	1.9	0.076	34.0	0.99780	3.51	0.56	9.4	5
1	0.880	0.00	2.6	0.098	67.0	0.99680	3.20	0.68	9.8	5
2	0.760	0.04	2.3	0.092	54.0	0.99700	3.26	0.65	9.8	5
3	0.280	0.56	1.9	0.075	60.0	0.99800	3.16	0.58	9.8	6
4	0.700	0.00	1.9	0.076	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...
1594	0.600	0.08	2.0	0.090	44.0	0.99490	3.45	0.58	10.5	5
1595	0.550	0.10	2.2	0.062	51.0	0.99512	3.52	0.76	11.2	6
1596	0.510	0.13	2.3	0.076	40.0	0.99574	3.42	0.75	11.0	6
1597	0.645	0.12	2.0	0.075	44.0	0.99547	3.57	0.71	10.2	5
1598	0.310	0.47	3.6	0.067	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 10 columns

Outlier Removal using z score

- We can use the fact that if z score of a datapoint is greater than 3, it can be treated as an outlier

In [16]:
```
z=np.abs(zscore(WD))
threshold=3
np.where(z>3)

WD=WD[(z<3).all(axis=1)]
WD
```

Out[16]:

	volatile_acidity	citric_acid	residual_sugar	chlorides	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality
0	0.700	0.00	1.9	0.076	34.0	0.99780	3.51	0.56	9.4	5
1	0.880	0.00	2.6	0.098	67.0	0.99680	3.20	0.68	9.8	5
2	0.760	0.04	2.3	0.092	54.0	0.99700	3.26	0.65	9.8	5
3	0.280	0.56	1.9	0.075	60.0	0.99800	3.16	0.58	9.8	6
4	0.700	0.00	1.9	0.076	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...
1594	0.600	0.08	2.0	0.090	44.0	0.99490	3.45	0.58	10.5	5
1595	0.550	0.10	2.2	0.062	51.0	0.99512	3.52	0.76	11.2	6
1596	0.510	0.13	2.3	0.076	40.0	0.99574	3.42	0.75	11.0	6
1597	0.645	0.12	2.0	0.075	44.0	0.99547	3.57	0.71	10.2	5
1598	0.310	0.47	3.6	0.067	42.0	0.99549	3.39	0.66	11.0	6

1471 rows × 10 columns

From 1599 rowsof dataset, the same now reduced to 1471 rows.

Rectification of Class Imbalance in Dataset using SMOTE(Synthetic Minority Oversampling Technique)

In [17]:
```
from imblearn.over_sampling import SMOTE
```

In [18]:
```
X = WD.drop('quality',axis=1)
Y = WD['quality']
```

In [19]:
```
Y.value_counts()
```

Out[19]:
```
5    624
6    595
7    189
4     47
8     16
Name: quality, dtype: int64
```

In [20]:
```
os = SMOTE()
X, Y = os.fit_resample(X, Y)
```

In [21]:
```
Y.value_counts()
```

Out[21]:
```
5    624
6    624
7    624
4    624
8    624
Name: quality, dtype: int64
```

In [22]:
```
Y
```

Out[22]:
```
0       5
1       5
2       5
3       6
4       5
       ..
3115    8
3116    8
3117    8
3118    8
3119    8
Name: quality, Length: 3120, dtype: int64
```

Converting Output Label to Binary

In [23]:
```
Y = Y.apply(lambda y_value:1 if y_value>=7 else 0)
Y
```

Out[23]:
```
0       0
1       0
2       0
3       0
4       0
       ..
3115    1
3116    1
3117    1
3118    1
3119    1
Name: quality, Length: 3120, dtype: int64
```

In [24]:
```
X
```

Out[24]:

	volatile_acidity	citric_acid	residual_sugar	chlorides	total_sulfur_dioxide	density	pH	sulphates	alcohol
0	0.700000	0.000000	1.900000	0.076000	34.000000	0.997800	3.510000	0.560000	9.400000
1	0.880000	0.000000	2.600000	0.098000	67.000000	0.996800	3.200000	0.680000	9.800000
2	0.760000	0.040000	2.300000	0.092000	54.000000	0.997000	3.260000	0.650000	9.800000
3	0.280000	0.560000	1.900000	0.075000	60.000000	0.998000	3.160000	0.580000	9.800000
4	0.700000	0.000000	1.900000	0.076000	34.000000	0.997800	3.510000	0.560000	9.400000
...	...	...	...	...	...	...	...	...	...
3115	0.349748	0.527476	2.588640	0.069886	15.962133	0.997185	3.151010	0.655680	10.987378
3116	0.601346	0.660327	5.020366	0.084895	18.792730	0.998689	3.215164	0.687236	13.234184
3117	0.378171	0.513097	2.149264	0.070563	16.000000	0.995746	3.183805	0.672537	11.845131
3118	0.589374	0.314742	3.423709	0.073137	34.925371	0.995964	3.367003	0.720626	12.971240
3119	0.333526	0.338564	1.705037	0.061504	13.151124	0.995930	3.228993	1.079346	10.125937

3120 rows × 9 columns

Feature Scaling

In [25]:
```
from sklearn.preprocessing import StandardScaler
```

In [26]:
```
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X
```

Out[26]:

	volatile_acidity	citric_acid	residual_sugar	chlorides	total_sulfur_dioxide	density	pH	sulphates	alcohol
0	1.048325	-1.542466	-0.635080	-0.143265	-0.096157	0.828219	1.479045	-0.807893	-1.251586
1	2.058710	-1.542466	0.159890	1.060407	1.159779	0.213801	-0.748226	0.086190	-0.892660
2	1.385120	-1.334554	-0.180812	0.732133	0.665016	0.336684	-0.317141	-0.137331	-0.892660
3	-1.309242	1.368294	-0.635080	-0.197977	0.893368	0.951103	-1.035616	-0.658879	-0.892660
4	1.048325	-1.542466	-0.635080	-0.143265	-0.096157	0.828219	1.479045	-0.807893	-1.251586
...	...	...	...	...	...	...	...	...	...
3115	-0.917731	1.199239	0.146988	-0.477755	-0.782653	0.450262	-1.100208	-0.095010	0.172794
3116	0.494553	1.889774	2.908630	0.343378	-0.674925	1.374717	-0.639278	0.140106	2.188890
3117	-0.758183	1.124504	-0.351999	-0.440713	-0.781212	-0.433566	-0.864581	0.030584	0.942469
3118	0.427355	0.093494	1.095352	-0.299896	-0.060938	-0.299817	0.451651	0.388880	1.952945
3119	-1.008785	0.217316	-0.856493	-0.936389	-0.889637	-0.320447	-0.539922	3.061597	-0.600191

3120 rows × 9 columns

In [27]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [28]:
```
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
```

Creating Train-Test Split

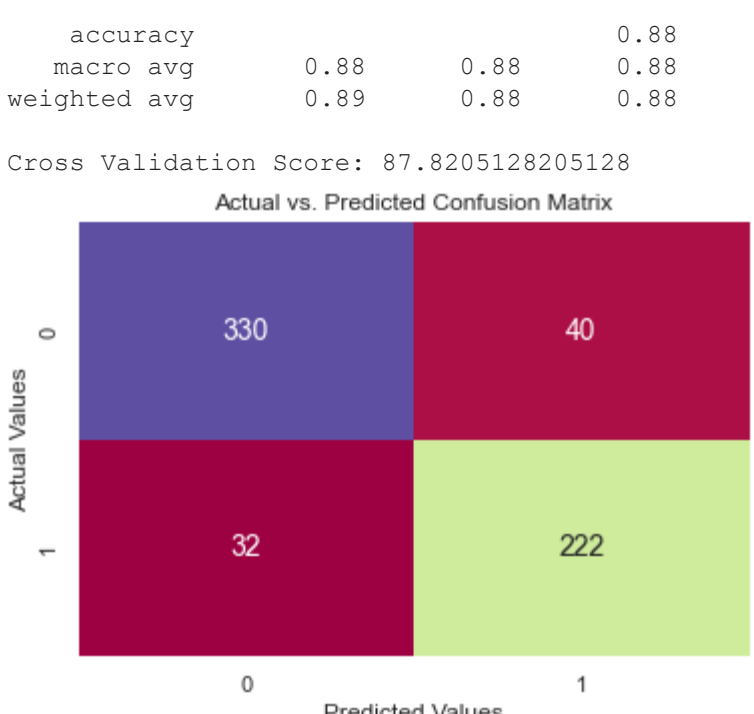
In [29]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

ML MODELS AND PERFORMANCE EVALUATION

Logistic Regression

In [37]:
```
Lgs = LogisticRegression()
Lgs.fit(X_train, Y_train)
pred_Lgs = Lgs.predict(X_test)

acc_score_Lgs = (accuracy_score(Y_test, pred_Lgs))*100
print("Accuracy Score:", acc_score_Lgs)

class_report = classification_report(Y_test, pred_Lgs)
print("\nClassification Report:\n", class_report)

cv_score = (cross_val_score(Lgs, X, Y, cv=5).mean())*100
print("Cross Validation Score:", cv_score)

cm = confusion_matrix(Y_test,pred_Lgs)
sns.heatmap(cm, annot=True, fmt='d', cbar=False,cmap='Spectral',annot_kws={'size': 14})
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('Actual vs. Predicted Confusion Matrix')
plt.show()
```

```
Accuracy Score: 88.46153846153845

Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.89      0.90       370
           1       0.85      0.87      0.86       254

    accuracy                           0.88       624
   macro avg       0.88      0.88      0.88       624
weighted avg       0.89      0.88      0.88       624

Cross Validation Score: 87.8205128205128
```



DecisionTreeClassifier

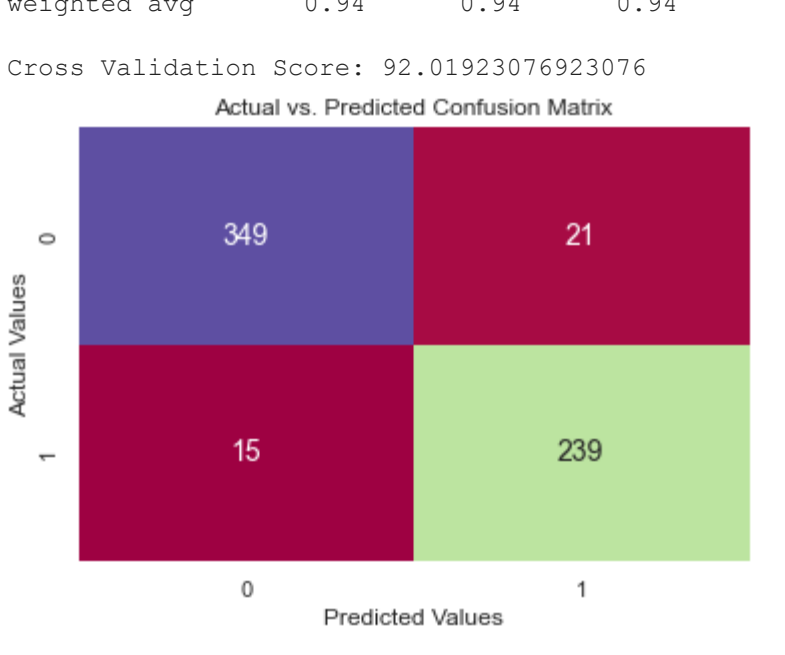
In [41]:
```
DTC = DecisionTreeClassifier(random_state=21, max_depth=15)
DTC.fit(X_train, Y_train)
pred_DTC = DTC.predict(X_test)

acc_score_DTC = (accuracy_score(Y_test, pred_DTC))*100
print("Accuracy Score:", acc_score_DTC)

class_report = classification_report(Y_test, pred_DTC)
print("\nClassification Report:\n", class_report)

cv_score = (cross_val_score(DTC, X, Y, cv=5).mean())*100
print("Cross Validation Score:", cv_score)

cm = confusion_matrix(Y_test,pred_DTC)
sns.heatmap(cm, annot=True, fmt='d', cbar=False,cmap='Spectral',annot_kws={'size': 14})
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('Actual vs. Predicted Confusion Matrix')
plt.show()
```

```
Accuracy Score: 90.38461538461539

Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.91      0.92       370
           1       0.87      0.90      0.88       254

    accuracy                           0.90       624
   macro avg       0.90      0.90      0.90       624
weighted avg       0.90      0.90      0.90       624

Cross Validation Score: 88.49358974358974
```

RandomForestClassifier

In [46]:
```
RFC = RandomForestClassifier(max_depth=15, random_state=42)
RFC.fit(X_train, Y_train)
pred_RFC = RFC.predict(X_test)

acc_score_RFC = (accuracy_score(Y_test, pred_RFC))*100
print("Accuracy Score:", acc_score_RFC)

class_report = classification_report(Y_test, pred_RFC)
print("\nClassification Report:\n", class_report)

cv_score = (cross_val_score(RFC, X, Y, cv=5).mean())*100
print("Cross Validation Score:", cv_score)

cm = confusion_matrix(Y_test,pred_RFC)
sns.heatmap(cm, annot=True, fmt='d', cbar=False,cmap='Spectral',annot_kws={'size': 14})
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('Actual vs. Predicted Confusion Matrix')
plt.show()
```

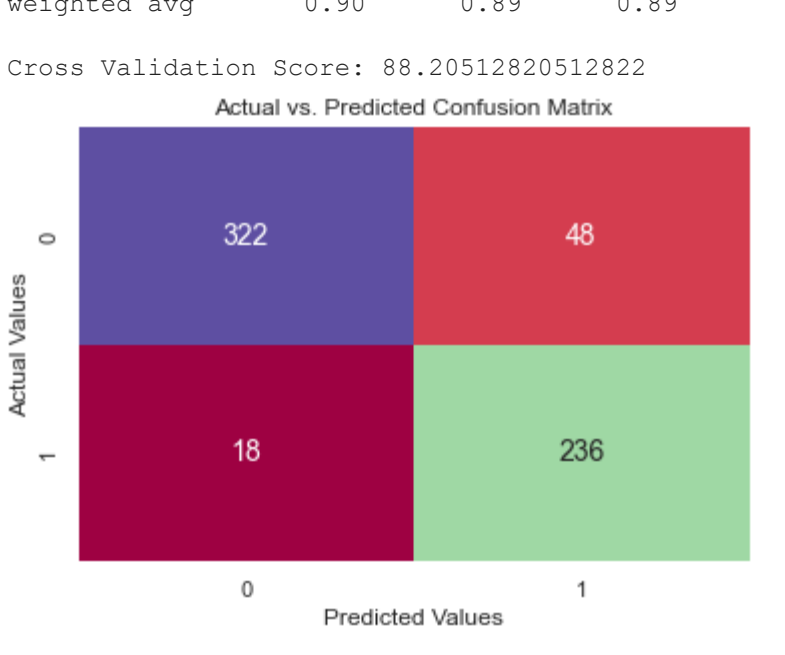
```
Accuracy Score: 94.23076923076923

Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.94      0.95       370
           1       0.92      0.94      0.93       254

    accuracy                           0.94       624
   macro avg       0.94      0.94      0.94       624
weighted avg       0.94      0.94      0.94       624

Cross Validation Score: 92.01923076923076
```



KNN Classifier

In [51]:
```
KNN = KNeighborsClassifier(n_neighbors=15)
KNN.fit(X_train, Y_train)
pred_KNN = KNN.predict(X_test)

acc_score_KNN = (accuracy_score(Y_test, pred_KNN))*100
print("Accuracy Score:", acc_score_KNN)

class_report = classification_report(Y_test, pred_KNN)
print("\nClassification Report:\n", class_report)

cv_score = (cross_val_score(KNN, X, Y, cv=5).mean())*100
print("Cross Validation Score:", cv_score)

cm = confusion_matrix(Y_test,pred_KNN)
sns.heatmap(cm, annot=True, fmt='d', cbar=False,cmap='Spectral',annot_kws={'size': 14})
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('Actual vs. Predicted Confusion Matrix')
plt.show()
```

```
Accuracy Score: 89.42307692307693

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.87      0.91       370
           1       0.83      0.93      0.88       254

    accuracy                           0.89       624
   macro avg       0.89      0.90      0.89       624
weighted avg       0.90      0.89      0.89       624

Cross Validation Score: 88.20512820512822
```



HYPERPARAMAETER TUNING FOR THE BEST MODEL (RANDOMFORESTCLASSIFIER)

In [53]:
```
RFCF = RandomForestClassifier(random_state=42)
np.random.seed(42)
param_RFCF = {'max_depth': [ 10, 20],
              'max_features': [ 'sqrt', 'log2'],'min_samples_leaf': [1, 2, 4], 'min_samples_split': [2, 5, 10]
              'criterion': ['gini', 'entropy']}

cv_RFCF = GridSearchCV(RFCF, cv = 10,
                     param_grid=param_RFCF,
                     n_jobs=-1, verbose=1, scoring='r2')

cv_RFCF.fit(X_train, Y_train)
print('Best Parameters using grid search: \n', cv_RFCF.best_params_)
```

```
Fitting 10 folds for each of 144 candidates, totalling 1440 fits
Best Parameters using grid search:
 {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split':
5, 'n_estimators': 200}
```

In [54]:
```
RFCF.set_params(bootstrap =True,criterion='entropy', max_depth=20, max_features='sqrt', min_samples_leaf=1, mi
```

Out[54]:
```
RandomForestClassifier(criterion='entropy', max_depth=20, max_features='sqrt',
                       min_samples_split=5, n_estimators=200, random_state=42)
```

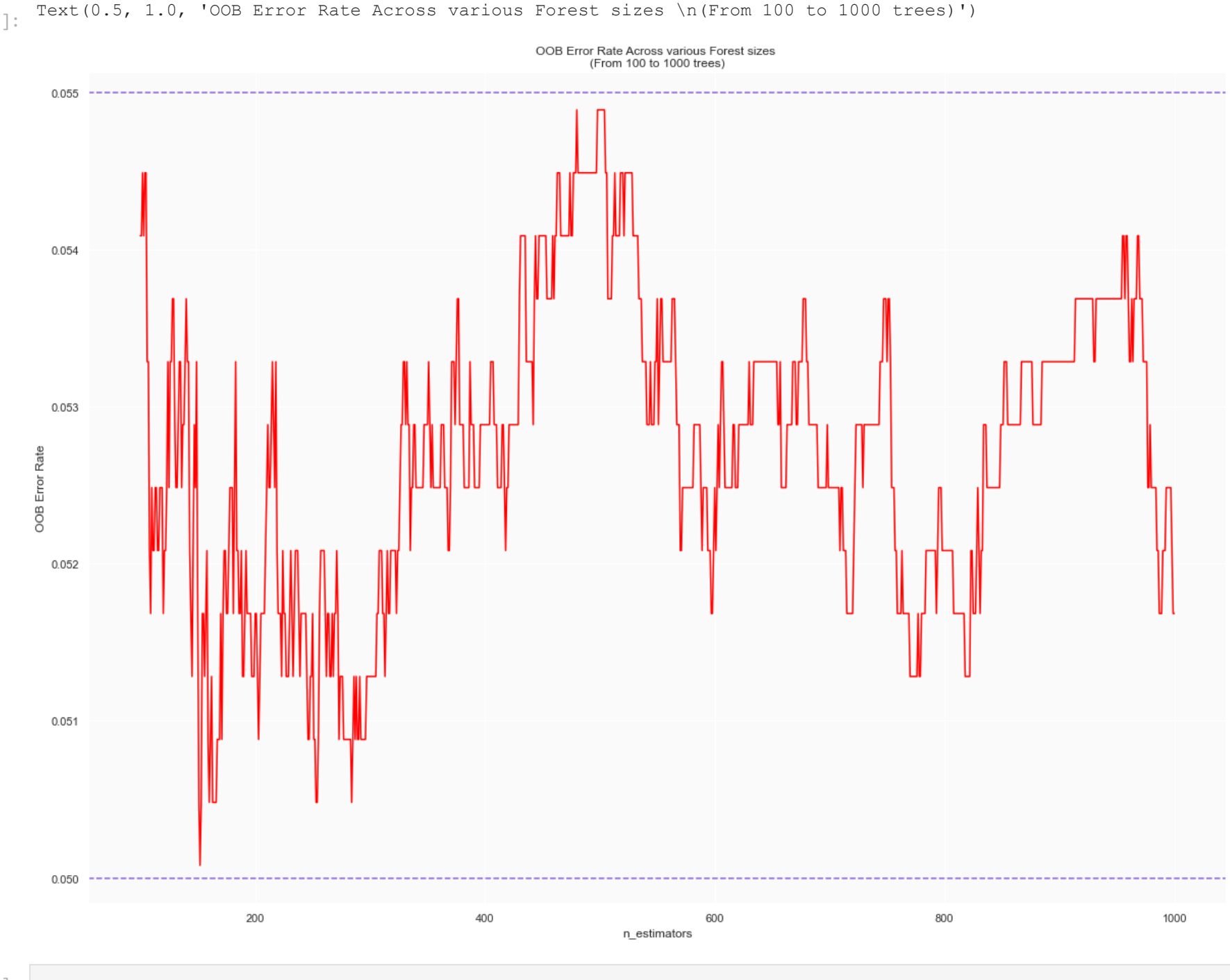
OOB Rate

In [55]:
```
RFCF.set_params(warm_start=True,
                oob_score=True)

min_estimators = 100
max_estimators = 1000

error_rate = {}

for i in range(min_estimators, max_estimators + 1):
    RFCF.set_params(n_estimators=i)
    RFCF.fit(X_train, Y_train)

    oob_error = 1 - RFCF.oob_score_
    error_rate[i] = oob_error
```

In [56]:
```
oob_series_RFCF = pd.Series(error_rate)
```

In [57]:
```
fig, ax = plt.subplots(figsize=(20, 15))

ax.set_facecolor('#fafafa')

oob_series_RFCF.plot(kind='line',color = 'red')
plt.axhline(0.055, color='#875FDB',linestyle='--')
plt.axhline(0.05, color='#875FDB',linestyle='--')
plt.xlabel('n_estimators')
plt.ylabel('OOB Error Rate')
plt.title('OOB Error Rate Across various Forest sizes \n(From 100 to 1000 trees)')
```

Out[57]: Text(0.5, 1.0, 'OOB Error Rate Across various Forest sizes \n(From 100 to 1000 trees)')



In [66]:

```
print('OOB Error rate for 160 trees is: {0:.5f}'.format(oob_series_RFCF[160]))
```

```
OOB Error rate for 160 trees is: 0.05048
```

In [67]:

```
# Refine the tree via OOB Output
RFCF.set_params(n_estimators=160,
                bootstrap = True,
                oob_score=False, warm_start=False)
```

Out[67]:

```
RandomForestClassifier(criterion='entropy', max_depth=20, max_features='sqrt',
                       min_samples_split=5, n_estimators=160, random_state=42)
```

In [68]:

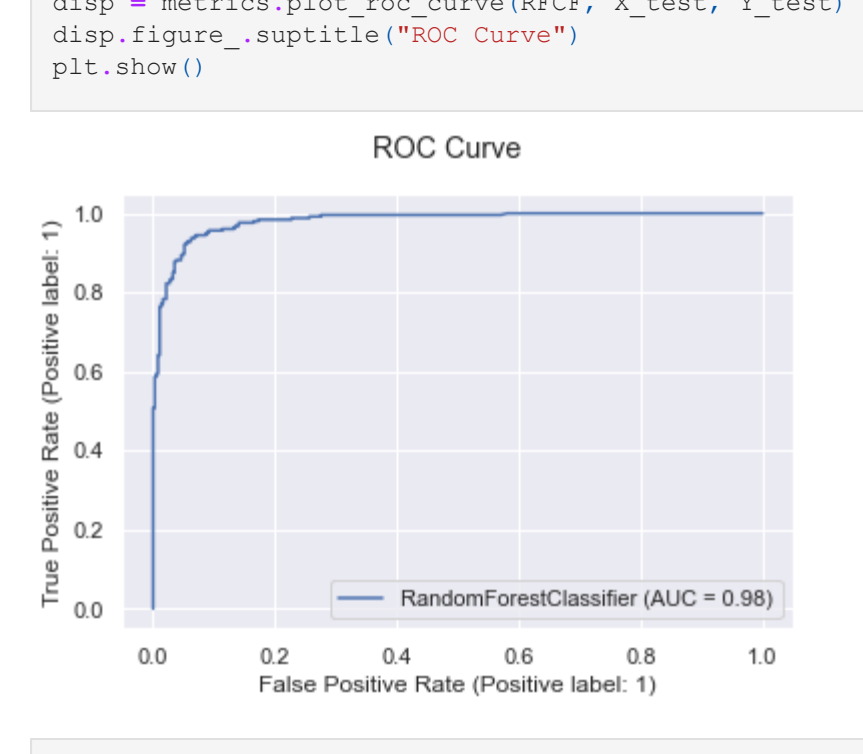
```
RFCF.fit(X_train, Y_train)
Y_pred2 = RFCF.predict(X_test)
RFCF_acc = (accuracy_score(Y_test, Y_pred2))*100
print("Accuracy score for the Best Model is:", RFCF_acc)
print(classification_report(Y_test, Y_pred2))
```

```
Accuracy score for the Best Model is: 93.58974358974359
              precision    recall  f1-score   support

           0       0.96      0.93      0.95       370
           1       0.91      0.94      0.92       254

    accuracy                           0.94       624
   macro avg       0.93      0.94      0.93       624
weighted avg       0.94      0.94      0.94       624
```

ROC Curve

In [72]:

```
disp = metrics.plot_roc_curve(RFCF, X_test, Y_test)
disp.figure_.suptitle("ROC Curve")
plt.show()
```



In [ ]: