### Importing the Dependencies

In [4]:
```
# Importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn import preprocessing
from sklearn.model_selection import train_test_split,GridSearchCV,cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression # For Logistic Regression ML Model
from sklearn.tree import DecisionTreeClassifier # For Decision Tree ML Model
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, confusion_matrix , classification_report , accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier

import warnings;
warnings.filterwarnings('ignore');
```

# Exploratory Data Analysis

In [8]:
```
df = pd.read_csv('QualityPrediction.csv') # Import the dataset
df.head()
```

Out[8]:

| | 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 |

In [9]:
```
df.info() # Dataset has only two dtypes - float64 and int64
```

```
<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 [10]:
```
df.isnull().sum() # No Null values in the dataset
```

Out[10]:
```
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 [11]:
```
df.describe() # Statistical data
```

Out[11]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alc |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.00 |
| **mean** | 8.319637 | 0.527821 | 0.270976 | 2.538806 | 0.087467 | 15.874922 | 46.467792 | 0.996747 | 3.311113 | 0.658149 | 10.42 |
| **std** | 1.741096 | 0.179060 | 0.194801 | 1.409928 | 0.047065 | 10.460157 | 32.895324 | 0.001887 | 0.154386 | 0.169507 | 1.06 |
| **min** | 4.600000 | 0.120000 | 0.000000 | 0.900000 | 0.012000 | 1.000000 | 6.000000 | 0.990070 | 2.740000 | 0.330000 | 8.40 |
| **25%** | 7.100000 | 0.390000 | 0.090000 | 1.900000 | 0.070000 | 7.000000 | 22.000000 | 0.995600 | 3.210000 | 0.550000 | 9.50 |
| **50%** | 7.900000 | 0.520000 | 0.260000 | 2.200000 | 0.079000 | 14.000000 | 38.000000 | 0.996750 | 3.310000 | 0.620000 | 10.20 |
| **75%** | 9.200000 | 0.640000 | 0.420000 | 2.600000 | 0.090000 | 21.000000 | 62.000000 | 0.997835 | 3.400000 | 0.730000 | 11.10 |
| **max** | 15.900000 | 1.580000 | 1.000000 | 15.500000 | 0.611000 | 72.000000 | 289.000000 | 1.003690 | 4.010000 | 2.000000 | 14.90 |

In [12]:
```
df.mode() # Shows most repeated values in the features
```

Out[12]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.2 | 0.6 | 0.0 | 2.0 | 0.08 | 6.0 | 28.0 | 0.9972 | 3.3 | 0.6 | 9.5 | 5 |

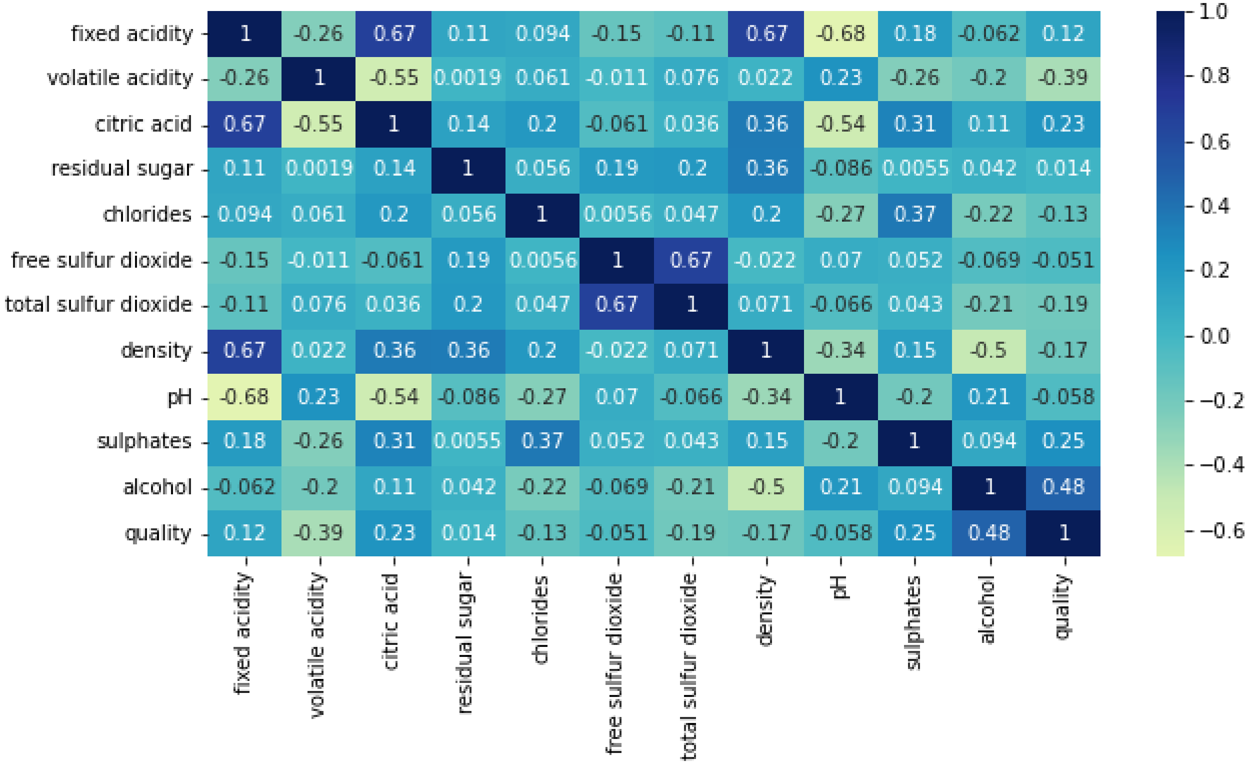
In [13]:
```
df.corr() # Correlation of features with eachother and target variable
```

Out[13]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **fixed acidity** | 1.000000 | -0.256131 | 0.671703 | 0.114777 | 0.093705 | -0.153794 | -0.113181 | 0.668047 | -0.682978 | 0.183006 | -0.061668 | 0.124052 |
| **volatile acidity** | -0.256131 | 1.000000 | -0.552496 | 0.001918 | 0.061298 | -0.010504 | 0.076470 | 0.022026 | 0.234937 | -0.260987 | -0.202288 | -0.390558 |
| **citric acid** | 0.671703 | -0.552496 | 1.000000 | 0.143577 | 0.203823 | -0.060978 | 0.035533 | 0.364947 | -0.541904 | 0.312770 | 0.109903 | 0.226373 |
| **residual sugar** | 0.114777 | 0.001918 | 0.143577 | 1.000000 | 0.055610 | 0.187049 | 0.203028 | 0.355283 | -0.085652 | 0.005527 | 0.042075 | 0.013732 |
| **chlorides** | 0.093705 | 0.061298 | 0.203823 | 0.055610 | 1.000000 | 0.005562 | 0.047400 | 0.200632 | -0.265026 | 0.371260 | -0.221141 | -0.128907 |
| **free sulfur dioxide** | -0.153794 | -0.010504 | -0.060978 | 0.187049 | 0.005562 | 1.000000 | 0.667666 | -0.021946 | 0.070377 | 0.051658 | -0.069408 | -0.050656 |
| **total sulfur dioxide** | -0.113181 | 0.076470 | 0.035533 | 0.203028 | 0.047400 | 0.667666 | 1.000000 | 0.071269 | -0.066495 | 0.042947 | -0.205654 | -0.185100 |
| **density** | 0.668047 | 0.022026 | 0.364947 | 0.355283 | 0.200632 | -0.021946 | 0.071269 | 1.000000 | -0.341699 | 0.148506 | -0.496180 | -0.174919 |
| **pH** | -0.682978 | 0.234937 | -0.541904 | -0.085652 | -0.265026 | 0.070377 | -0.066495 | -0.341699 | 1.000000 | -0.196648 | 0.205633 | -0.057731 |
| **sulphates** | 0.183006 | -0.260987 | 0.312770 | 0.005527 | 0.371260 | 0.051658 | 0.042947 | 0.148506 | -0.196648 | 1.000000 | 0.093595 | 0.251397 |
| **alcohol** | -0.061668 | -0.202288 | 0.109903 | 0.042075 | -0.221141 | -0.069408 | -0.205654 | -0.496180 | 0.205633 | 0.093595 | 1.000000 | 0.476166 |
| **quality** | 0.124052 | -0.390558 | 0.226373 | 0.013732 | -0.128907 | -0.050656 | -0.185100 | -0.174919 | -0.057731 | 0.251397 | 0.476166 | 1.000000 |

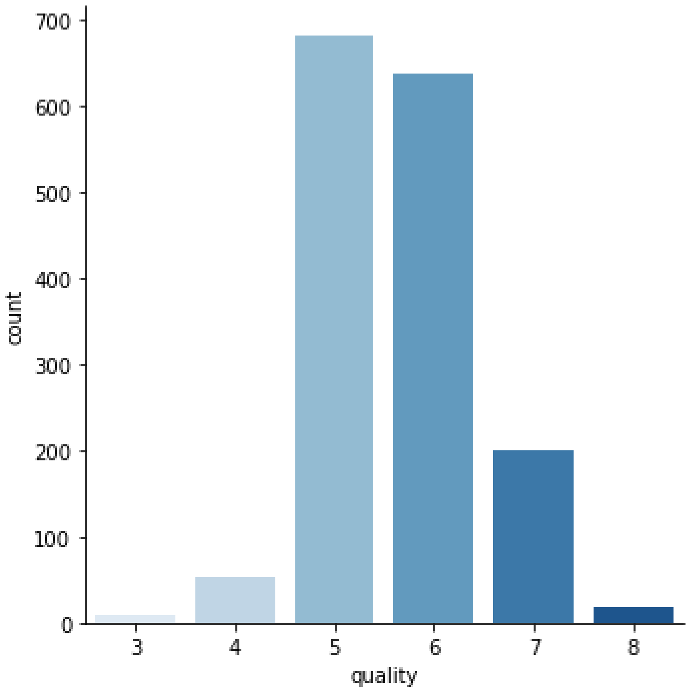
In [9]:
```
plt.figure(figsize=(10,5))
sns.heatmap(df.corr(), annot = True, cbar = True, cmap = "YlGnBu", center = 0)
plt.show()

# Observations:
# pH and fixed acidity has strong correlation
# pH and citric acid has strong correlation
# volatile acidity and citric acid has strong correlation
# citric acid and fixed acidity has strong correlation
# density and fixed acidity has strong correlation
# total sulpur dioxide and free sulphur dioxide has strong correlation
# alcohol and quality has good correlation
# volatile acidity and quality has good colleration
# sulphates and citric acid has good correlation
# suphates and chlorides has good correlation
```



In [10]:
```
sns.catplot(x = 'quality', data = df, kind = 'count', palette = 'Blues')
# quality feature has high number of values in categories => 5,6 and 7, whereas few values in 3,4, and 8
```

Out[10]:
```
<seaborn.axisgrid.FacetGrid at 0x1b777877970>
```

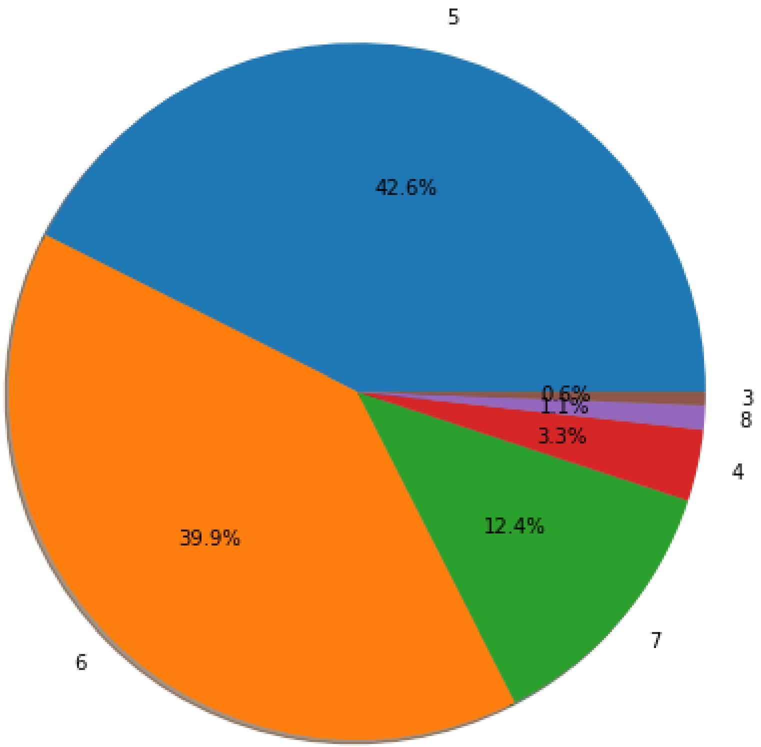



In [151]:
```
df.quality.value_counts() # Numerical representation of above catplot
```

Out[151]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

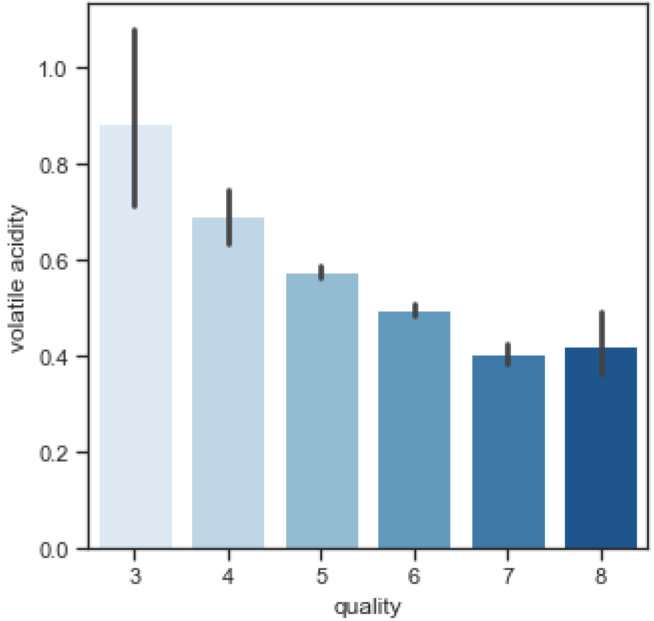
In [182]:
```
# Pie Chart of target variable
Quality_count=[681,638,199,53,18,10]
Quality_labels=['5','6','7','4','8','3']
plt.pie(Quality_count,labels=Quality_labels,radius=2,autopct='%0.1f%%',shadow=True)
```

Out[182]:
```
([<matplotlib.patches.Wedge at 0x1b77f1bf3d0>,
  <matplotlib.patches.Wedge at 0x1b77fcc66a0>,
  <matplotlib.patches.Wedge at 0x1b77fcc6700>,
  <matplotlib.patches.Wedge at 0x1b77e0af910>,
  <matplotlib.patches.Wedge at 0x1b77e0afca0>,
  <matplotlib.patches.Wedge at 0x1b77d62e370>],
 [Text(0.5075885136176095, 2.1406433380746703, '5'),
  Text(-1.5518097107013993, -1.5594507436186755, '6'),
  Text(1.6694494845062682, -1.4328078791944705, '7'),
  Text(2.1497439794837754, -0.46754766887801047, '4'),
  Text(2.1938714019596497, -0.16409835972245979, '8'),
  Text(2.199575397837407, -0.04322116643977899, '3')],
 [Text(0.2768664619732415, 1.16762363894982, '42.6%'),
  Text(-0.8464416603825813, -0.8506094965192774, '39.9%'),
  Text(0.9106088097306916, -0.7815315704697111, '12.4%'),
  Text(1.1725876251729683, -0.2550260012061875, '3.3%'),
  Text(1.196657128341627, -0.08950819621225078, '1.1%'),
  Text(1.1997683988204035, -0.023575181694424904, '0.6%')])
```


5
42.6%
0.6%
1.1%
3.3%
3
8
4
39.9%
12.4%
6
7

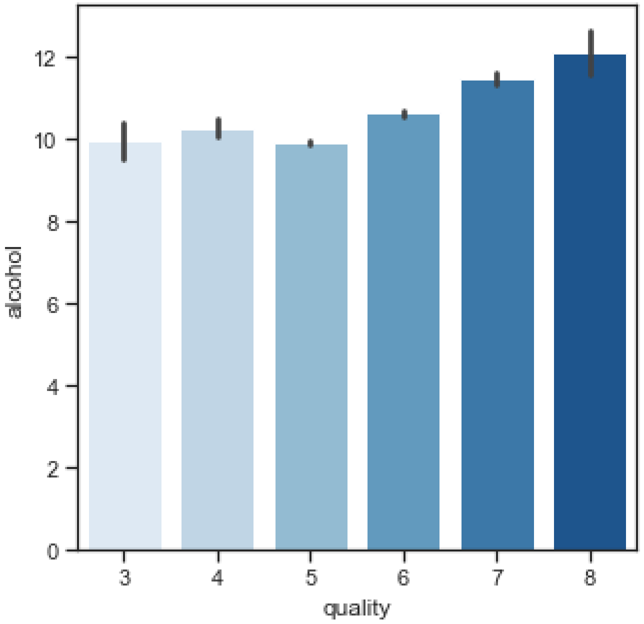
In [216]:
```
plot = plt.figure(figsize = (5,5))
sns.barplot(x = 'quality', y = 'volatile acidity', data = df,  palette = 'Blues')
```

Out[216]: <AxesSubplot:xlabel='quality', ylabel='volatile acidity'>



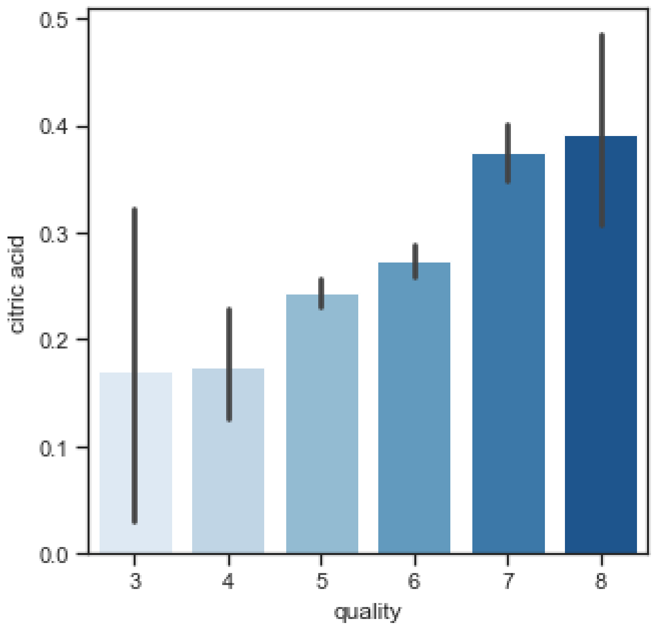
In [217]:
```
plot = plt.figure(figsize = (5,5))
sns.barplot(x = 'quality', y = 'alcohol', data = df,  palette = 'Blues')
```

Out[217]:
```
<AxesSubplot:xlabel='quality', ylabel='alcohol'>
```



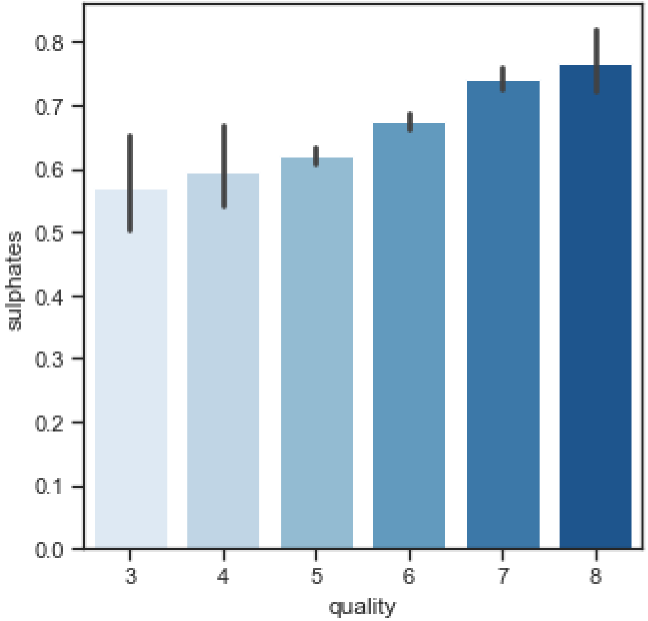
In [218]:
```
plot = plt.figure(figsize = (5,5))
sns.barplot(x = 'quality', y = 'citric acid', data = df,  palette = 'Blues')
```

Out[218]: <AxesSubplot:xlabel='quality', ylabel='citric acid'>



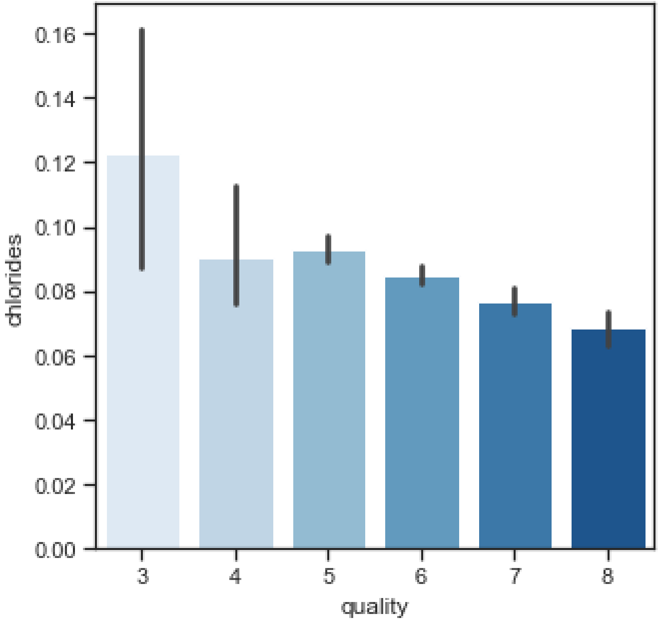
In [219]:
```
plot = plt.figure(figsize = (5,5))
sns.barplot(x = 'quality', y = 'sulphates', data = df,  palette = 'Blues')
```

Out[219]: <AxesSubplot:xlabel='quality', ylabel='sulphates'>



In [220]:
```
plot = plt.figure(figsize = (5,5))
sns.barplot(x = 'quality', y = 'chlorides', data = df,  palette = 'Blues')
```

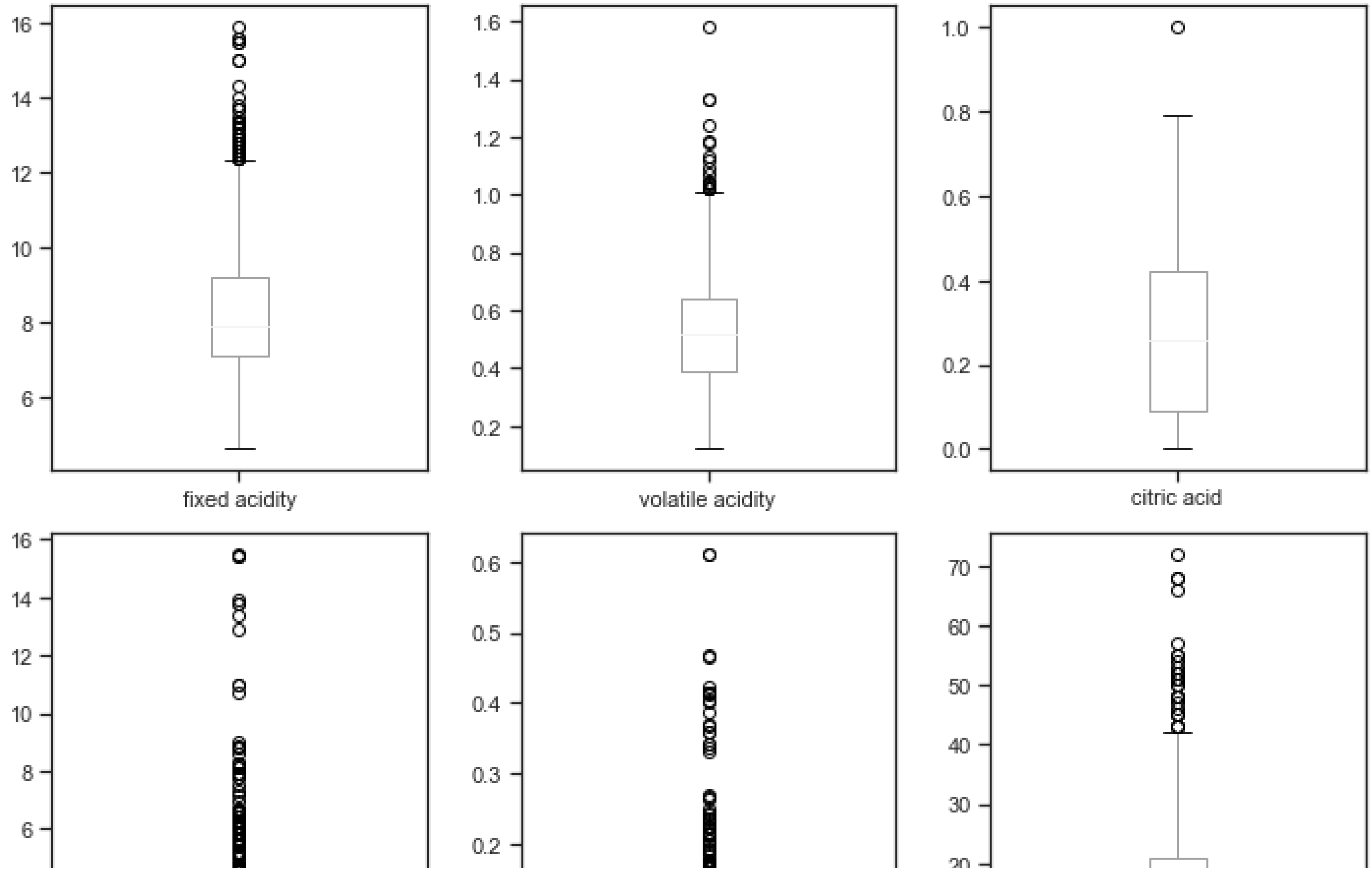
Out[220]: <AxesSubplot:xlabel='quality', ylabel='chlorides'>

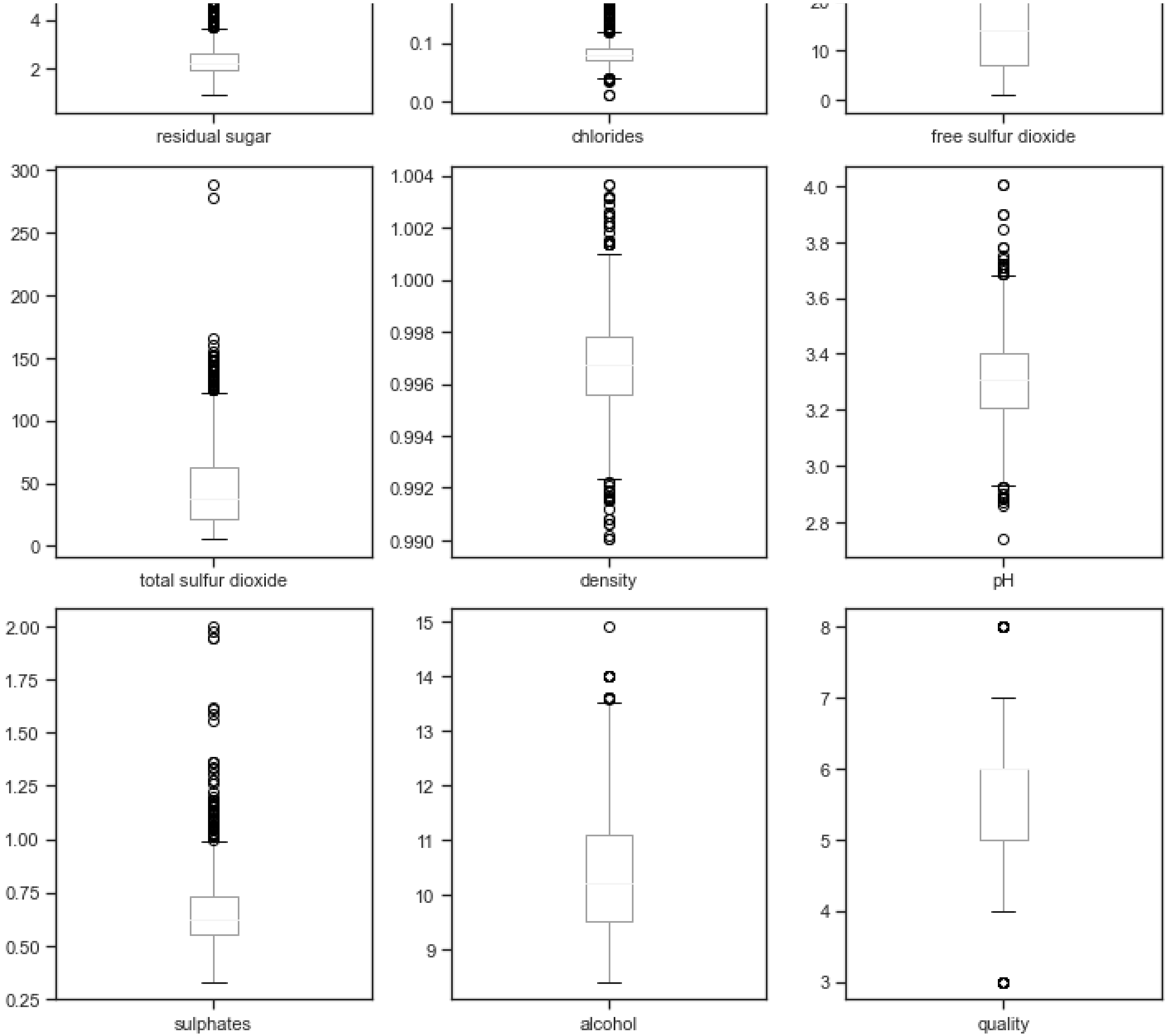


In [157]:
```
# Observations on barplot and piechart :
# 1) Volatile acidity and Chlorides are inversely proportional to quality
# 2) Citric acid and Sulphates are directly proportional to quality
# 3) Around 95% of share is allocated by 5,6 & 7 categories of Quality feature,each consisting of 42.6%, 39.9% and 12.4%
# 4) Whereas, remaining 5% is allocated by 3,4 & 8 where, 3 = 0.6%, 4 = 3.3% and 8 = 1.1%.
```

In [158]:
```
# Boxplot - To understand the outliers for each feature
plt.figure(figsize = (10,15))
col = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']
for i,col in enumerate (df.columns):
    plt.subplot(4,3,i+1)
    df.boxplot(col)
    plt.grid()
    plt.tight_layout()

# Observations on boxplot :
# citric acid , alcohol and quality has less outliers as compared to other features.
# Most of the features are right skewed.
```



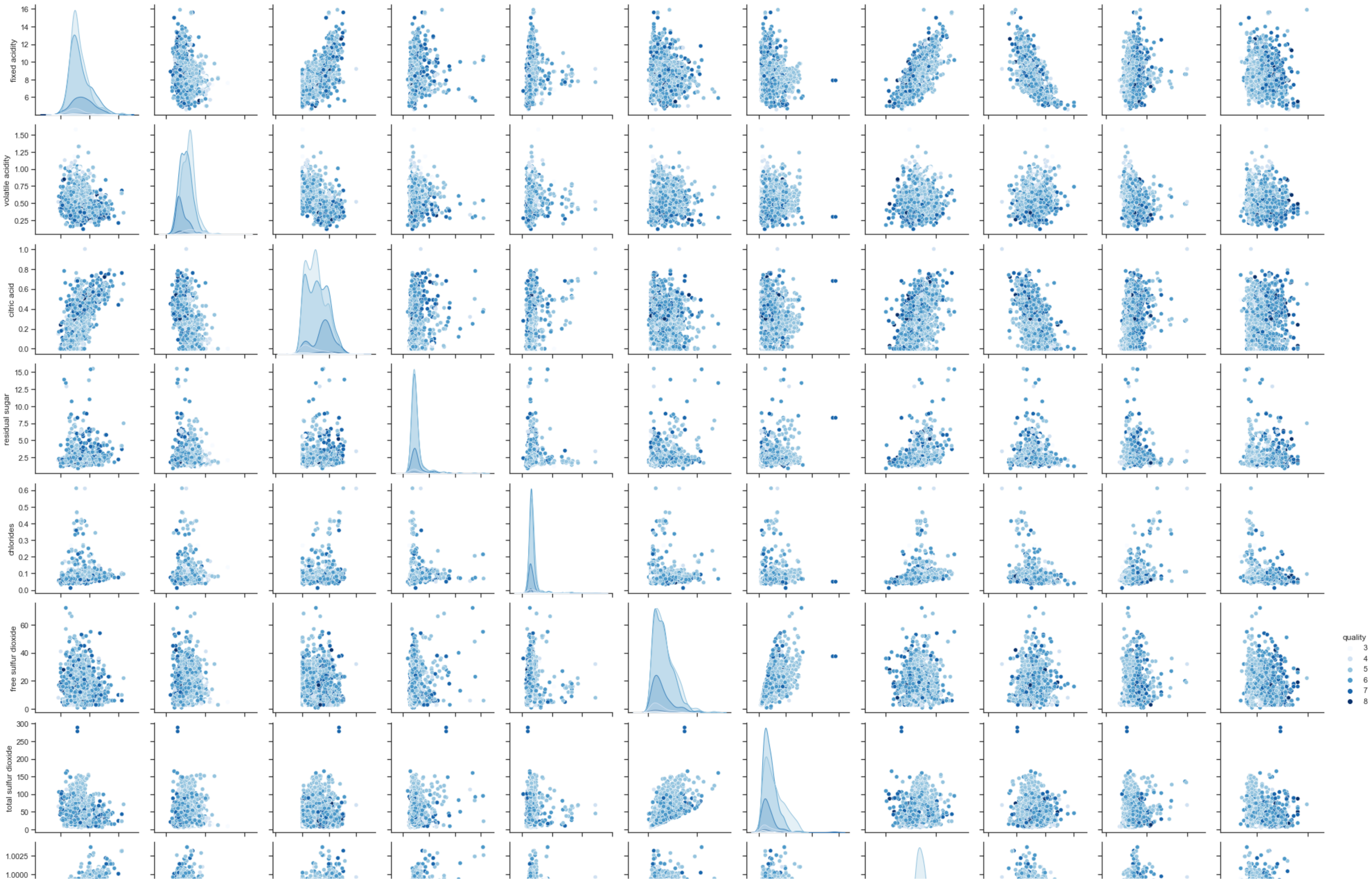

residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
quality

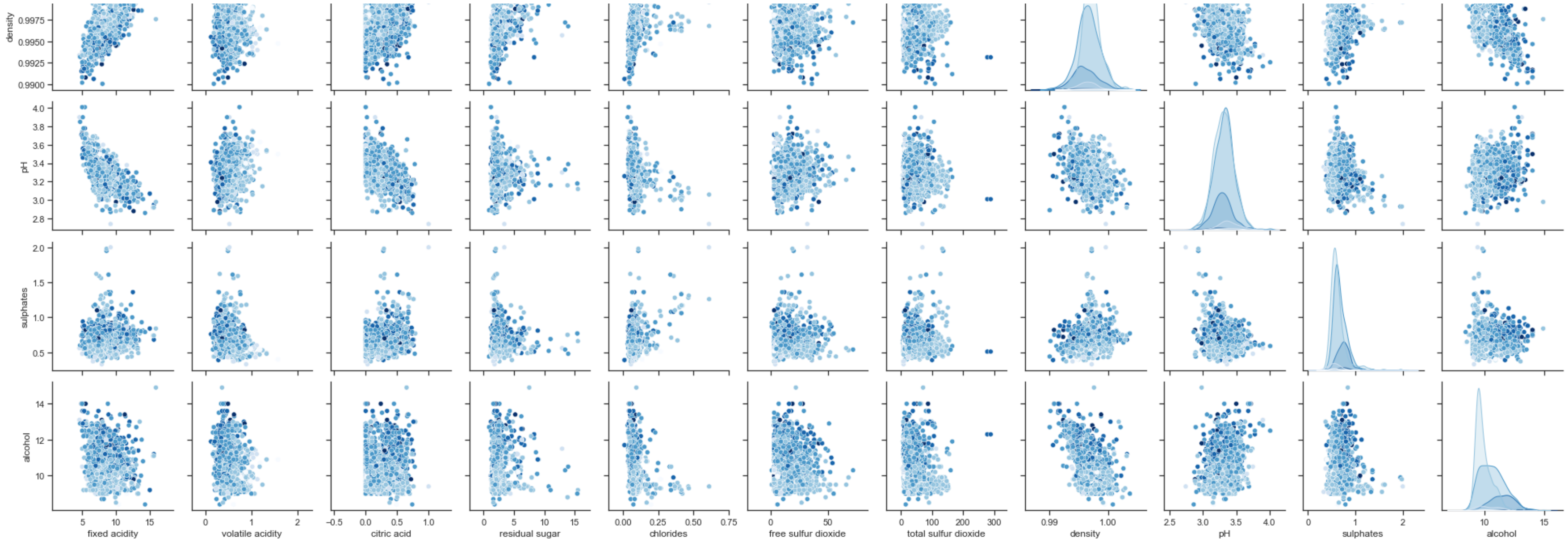
In [159]:
```
df.info()
```

```
<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 [221]:
```
# PairPlot
fig = plt.figure(figsize = (20,5))
sns.pairplot(df, hue = 'quality', palette = 'Blues')
fig.savefig('pairplot_wines_dataset.png')
```

```
<Figure size 1440x360 with 0 Axes>
```

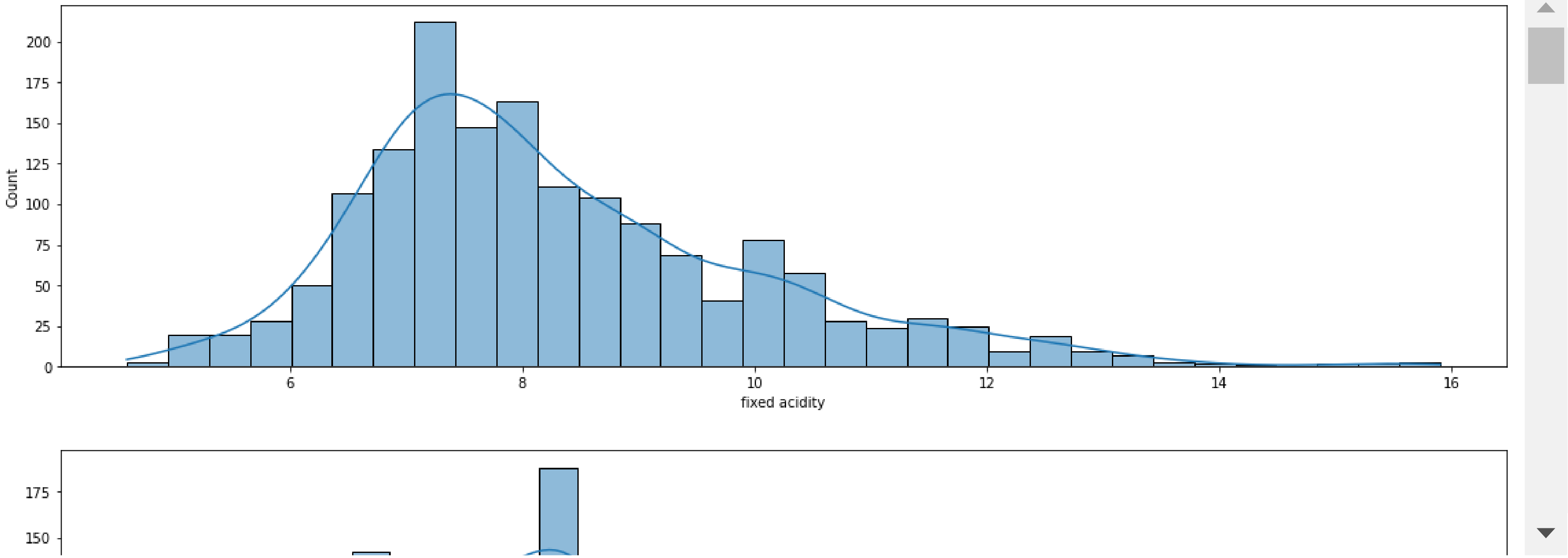



density
pH
sulphates
alcohol
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol

In [240]:
```python
# Histogram - To understand the distribution of each feature
fig, axes = plt.subplots(nrows=11, ncols=1, figsize=(16, 50), squeeze=False)

for i, column in enumerate(df.columns, start = 0):
    if column != "quality":
        sns.histplot(x=column, data=df, ax=axes[i, 0], kde=True)

fig.tight_layout(pad=3.0)
```




In [ ]:
```python
# Observations on histogram :
# 1) Right Skewed features - fixed acidity, citric acid, residual sugar, free sulphur dioxide, total sulphur dioxide, su
#    alcohol.
# 2) Normal Distributed features - density, pH,
```

## Outliers

In [161]:
```
#IQR Method
list_col = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']

for col in list_col:
    Q1 = np.percentile(df[col], 25)
    Q2 = np.percentile(df[col], 50)
    Q3 = np.percentile(df[col], 75)
    IQR = Q3-Q1
    IQRmin = Q1 - 1.5*IQR
    IQRmax = Q3 + 1.5*IQR
    outlier_IQR = []
    for i in df[col]:
        if i<IQRmin or i>IQRmax:
            outlier_IQR.append(i)
    print('Outliers in {0} are {1} '.format(col,outlier_IQR),"\n")
```

```
Outliers in fixed acidity are [12.8, 12.8, 15.0, 15.0, 12.5, 13.3, 13.4, 12.4, 12.5, 13.8, 13.5, 12.6, 12.5, 12.8, 12.
8, 14.0, 13.7, 13.7, 12.7, 12.5, 12.8, 12.6, 15.6, 12.5, 13.0, 12.5, 13.3, 12.4, 12.5, 12.9, 14.3, 12.4, 15.5, 15.5, 1
5.6, 13.0, 12.7, 13.0, 12.7, 12.4, 12.7, 13.2, 13.2, 13.2, 15.9, 13.3, 12.9, 12.6, 12.6]

Outliers in volatile acidity are [1.13, 1.02, 1.07, 1.33, 1.33, 1.04, 1.09, 1.04, 1.24, 1.185, 1.02, 1.035, 1.025, 1.11
5, 1.02, 1.02, 1.58, 1.18, 1.04]

Outliers in citric acid are [1.0]

Outliers in residual sugar are [6.1, 6.1, 3.8, 3.9, 4.4, 10.7, 5.5, 5.9, 5.9, 3.8, 5.1, 4.65, 4.65, 5.5, 5.5, 5.5, 5.5,
7.3, 7.2, 3.8, 5.6, 4.0, 4.0, 4.0, 4.0, 7.0, 4.0, 4.0, 6.4, 5.6, 5.6, 11.0, 11.0, 4.5, 4.8, 5.8, 5.8, 3.8, 4.4, 6.2, 4.
2, 7.9, 7.9, 3.7, 4.5, 6.7, 6.6, 3.7, 5.2, 15.5, 4.1, 8.3, 6.55, 6.55, 4.6, 6.1, 4.3, 5.8, 5.15, 6.3, 4.2, 4.2, 4.6, 4.
2, 4.6, 4.3, 4.3, 7.9, 4.6, 5.1, 5.6, 5.6, 6.0, 8.6, 7.5, 4.4, 4.25, 6.0, 3.9, 4.2, 4.0, 4.0, 4.0, 6.6, 6.0, 6.0, 3.8,
9.0, 4.6, 8.8, 8.8, 5.0, 3.8, 4.1, 5.9, 4.1, 6.2, 8.9, 4.0, 3.9, 4.0, 8.1, 8.1, 6.4, 6.4, 8.3, 8.3, 4.7, 5.5, 5.5, 4.3,
5.5, 3.7, 6.2, 5.6, 7.8, 4.6, 5.8, 4.1, 12.9, 4.3, 13.4, 4.8, 6.3, 4.5, 4.5, 4.3, 4.3, 3.9, 3.8, 5.4, 3.8, 6.1, 3.9, 5.
1, 5.1, 3.9, 15.4, 15.4, 4.8, 5.2, 5.2, 3.75, 13.8, 13.8, 5.7, 4.3, 4.1, 4.1, 4.4, 3.7, 6.7, 13.9, 5.1, 7.8]

Outliers in chlorides are [0.176, 0.17, 0.368, 0.341, 0.172, 0.332, 0.464, 0.401, 0.467, 0.122, 0.178, 0.146, 0.236, 0.
61, 0.36, 0.27, 0.039, 0.337, 0.263, 0.611, 0.358, 0.343, 0.186, 0.213, 0.214, 0.121, 0.122, 0.122, 0.128, 0.12, 0.159,
0.124, 0.122, 0.122, 0.174, 0.121, 0.127, 0.413, 0.152, 0.152, 0.125, 0.122, 0.2, 0.171, 0.226, 0.226, 0.25, 0.148, 0.1
22, 0.124, 0.124, 0.143, 0.222, 0.039, 0.157, 0.422, 0.034, 0.387, 0.415, 0.157, 0.157, 0.243, 0.241, 0.19, 0.132, 0.12
6, 0.038, 0.165, 0.145, 0.147, 0.012, 0.012, 0.039, 0.194, 0.132, 0.161, 0.12, 0.12, 0.123, 0.123, 0.414, 0.216, 0.171,
```

```
0.178, 0.369, 0.166, 0.166, 0.136, 0.132, 0.132, 0.123, 0.123, 0.123, 0.403, 0.137, 0.414, 0.166, 0.168, 0.415, 0.153,
0.415, 0.267, 0.123, 0.214, 0.214, 0.169, 0.205, 0.205, 0.039, 0.235, 0.23, 0.038]

Outliers in free sulfur dioxide are [52.0, 51.0, 50.0, 68.0, 68.0, 43.0, 47.0, 54.0, 46.0, 45.0, 53.0, 52.0, 51.0, 45.
0, 57.0, 50.0, 45.0, 48.0, 43.0, 48.0, 72.0, 43.0, 51.0, 51.0, 52.0, 55.0, 55.0, 48.0, 48.0, 66.0]

Outliers in total sulfur dioxide are [145.0, 148.0, 136.0, 125.0, 140.0, 136.0, 133.0, 153.0, 134.0, 141.0, 129.0, 128.
0, 129.0, 128.0, 143.0, 144.0, 127.0, 126.0, 145.0, 144.0, 135.0, 165.0, 124.0, 124.0, 134.0, 124.0, 129.0, 151.0, 133.
0, 142.0, 149.0, 147.0, 145.0, 148.0, 155.0, 151.0, 152.0, 125.0, 127.0, 139.0, 143.0, 144.0, 130.0, 278.0, 289.0, 135.
0, 160.0, 141.0, 141.0, 133.0, 147.0, 147.0, 131.0, 131.0, 131.0]

Outliers in density are [0.9916, 0.9916, 1.0014, 1.0015, 1.0015, 1.0018, 0.9912, 1.0022, 1.0022, 1.0014, 1.0014, 1.001
4, 1.0014, 1.0032, 1.0026, 1.0014, 1.00315, 1.00315, 1.00315, 1.0021, 1.0021, 0.9917, 0.9922, 1.0026, 0.9921, 0.99154,
0.99064, 0.99064, 1.00289, 0.99162, 0.99007, 0.99007, 0.9902, 0.9922, 0.9915, 0.99157, 0.9908, 0.99084, 0.99191, 1.0036
9, 1.00369, 1.00242, 0.99182, 1.00242, 0.99182]

Outliers in pH are [3.9, 3.75, 3.85, 2.74, 3.69, 3.69, 2.88, 2.86, 3.74, 2.92, 2.92, 2.92, 3.72, 2.87, 2.89, 2.89, 2.9
2, 3.9, 3.71, 3.69, 3.69, 3.71, 3.71, 2.89, 2.89, 3.78, 3.7, 3.78, 4.01, 2.9, 4.01, 3.71, 2.88, 3.72, 3.72]

Outliers in sulphates are [1.56, 1.28, 1.08, 1.2, 1.12, 1.28, 1.14, 1.95, 1.22, 1.95, 1.98, 1.31, 2.0, 1.08, 1.59, 1.0
2, 1.03, 1.61, 1.09, 1.26, 1.08, 1.0, 1.36, 1.18, 1.13, 1.04, 1.11, 1.13, 1.07, 1.06, 1.06, 1.05, 1.06, 1.04, 1.05, 1.0
2, 1.14, 1.02, 1.36, 1.36, 1.05, 1.17, 1.62, 1.06, 1.18, 1.07, 1.34, 1.16, 1.1, 1.15, 1.17, 1.17, 1.33, 1.18, 1.17, 1.0
3, 1.17, 1.1, 1.01]

Outliers in alcohol are [14.0, 14.0, 14.0, 14.0, 14.9, 14.0, 13.6, 13.6, 13.6, 14.0, 14.0, 13.56666667, 13.6]

Outliers in quality are [8, 8, 8, 8, 8, 3, 8, 8, 8, 3, 8, 3, 8, 3, 3, 8, 8, 8, 8, 8, 3, 3, 8, 8, 3, 3, 3, 8]
```

In [162]:
```
# Z-Score Method
list_col = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']
for col in list_col:
    mean = np.mean(df[col])
    std = np.std(df[col])
    outlier_Z = []
    for i in df[col]:
        z = (i - mean)/std
        if z <- 3  or z > 3:
            outlier_Z.append(i)
    print('Outliers for feature {0} are {1}'.format(col,outlier_Z),'\n')
```

```
Outliers for feature fixed acidity are [15.0, 15.0, 13.8, 14.0, 13.7, 13.7, 15.6, 14.3, 15.5, 15.5, 15.6, 15.9]

Outliers for feature volatile acidity are [1.13, 1.07, 1.33, 1.33, 1.09, 1.24, 1.185, 1.115, 1.58, 1.18]

Outliers for feature citric acid are [1.0]

Outliers for feature residual sugar are [10.7, 7.3, 7.2, 7.0, 11.0, 11.0, 7.9, 7.9, 15.5, 8.3, 7.9, 8.6, 7.5, 9.0, 8.8,
8.8, 8.9, 8.1, 8.1, 8.3, 8.3, 7.8, 12.9, 13.4, 15.4, 15.4, 13.8, 13.8, 13.9, 7.8]

Outliers for feature chlorides are [0.368, 0.341, 0.332, 0.464, 0.401, 0.467, 0.236, 0.61, 0.36, 0.27, 0.337, 0.263, 0.
611, 0.358, 0.343, 0.413, 0.25, 0.422, 0.387, 0.415, 0.243, 0.241, 0.414, 0.369, 0.403, 0.414, 0.415, 0.415, 0.267, 0.2
35, 0.23]

Outliers for feature free sulfur dioxide are [52.0, 51.0, 50.0, 68.0, 68.0, 54.0, 53.0, 52.0, 51.0, 57.0, 50.0, 48.0, 4
8.0, 72.0, 51.0, 51.0, 52.0, 55.0, 55.0, 48.0, 48.0, 66.0]

Outliers for feature total sulfur dioxide are [148.0, 153.0, 165.0, 151.0, 149.0, 147.0, 148.0, 155.0, 151.0, 152.0, 27
8.0, 289.0, 160.0, 147.0, 147.0]

Outliers for feature density are [1.0032, 1.0026, 1.00315, 1.00315, 1.00315, 1.0026, 0.99064, 0.99064, 1.00289, 0.9900
7, 0.99007, 0.9902, 0.9908, 0.99084, 1.00369, 1.00369, 1.00242, 1.00242]

Outliers for feature pH are [3.9, 3.85, 2.74, 3.9, 3.78, 3.78, 4.01, 4.01]

Outliers for feature sulphates are [1.56, 1.28, 1.2, 1.28, 1.95, 1.22, 1.95, 1.98, 1.31, 2.0, 1.59, 1.61, 1.26, 1.36,
```

```
1.18, 1.36, 1.36, 1.17, 1.62, 1.18, 1.34, 1.17, 1.17, 1.33, 1.18, 1.17, 1.17]

Outliers for feature alcohol are [14.0, 14.0, 14.0, 14.0, 14.9, 14.0, 14.0, 14.0]

Outliers for feature quality are [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

In [163]:
```
# Observations on outlier :
# Alcohol, Citric acid and Quality has less outliers as compared to other features.
# Most of the features are right skewed.
# Keeping outliers for model efficiency
```

In [164]:
```
f,axes = plt.subplots(1, 2, figsize=(16, 6))
sns.set(style="ticks", palette="pastel")
sns.boxplot(y = df['quality'], color = 'green', ax = axes[0])
axes[0].set_title('Boxplot of Quality')
sns.boxplot(y = df['alcohol'], color = 'orange', ax = axes[1])
axes[1].set_title('Boxplot of Alcohol')
```

Out[164]:
```
Text(0.5, 1.0, 'Boxplot of Alcohol')
```

In [166]:
```
df.columns
```

Out[166]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

## Check the Multicolinearity of the Independent Variables

In [11]:
```
### Check if there is Multicollinearity in Indenpendent variables

from statsmodels.stats.outliers_influence import variance_inflation_factor
X_new = df[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol']]
vif_data = pd.DataFrame()
vif_data['feature'] = X_new.columns
vif_data['VIF'] = [variance_inflation_factor(X_new.values, i)
                        for i in range(len(X_new.columns))]
print(vif_data)
```

```
                 feature          VIF
0          fixed acidity    74.452265
1       volatile acidity    17.060026
2            citric acid     9.183495
3         residual sugar     4.662992
4              chlorides     6.554877
5    free sulfur dioxide     6.442682
6   total sulfur dioxide     6.519699
7                density  1479.287209
8                     pH  1070.967685
9              sulphates    21.590621
10               alcohol   124.394866
```

In [ ]:
```
# Observations on Multicollinearity

# 1) Out of 11 Independent variables, 6 are above 10 VIF value.
# 2) As more than 50% of Independent variables are above 10 VIF value, we cannot take decision of removing them,
#     as they might contain necessary information of the dataset.
# 3) Regression models are affected by multicollinearity, hence we need to apply Classification models on Wine dataset.
# 4) Classification models are immune to multicollinearity and will give more accuracy without removing any feature or
#     loosing any important information on the dataset.
```

## Splitting the dataset into train and test data

In [14]:
```
X = df.drop('quality', axis = 1)
X.head(2)
```

Out[14]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.4 | 0.70 | 0.0 | 1.9 | 0.076 | 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 |
| **1** | 7.8 | 0.88 | 0.0 | 2.6 | 0.098 | 25.0 | 67.0 | 0.9968 | 3.20 | 0.68 | 9.8 |

In [15]:
```
# Converting the values of quality feature - Greater than or equal to 7 to 1 and rest values to 0.

Y = df['quality'].apply(lambda y_value : 1 if y_value >= 7 else 0)
Y
```

Out[15]:
```
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
```

In [65]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20 , random_state = 2) # Splitting the train - tes
```

### Feature Scaling

In [66]:
```
standard_Scaler=StandardScaler()
X_train = standard_Scaler.fit_transform(X_train)
X_test = standard_Scaler.transform(X_test)
```

# Building ML Models

# 1. Logistic Regression

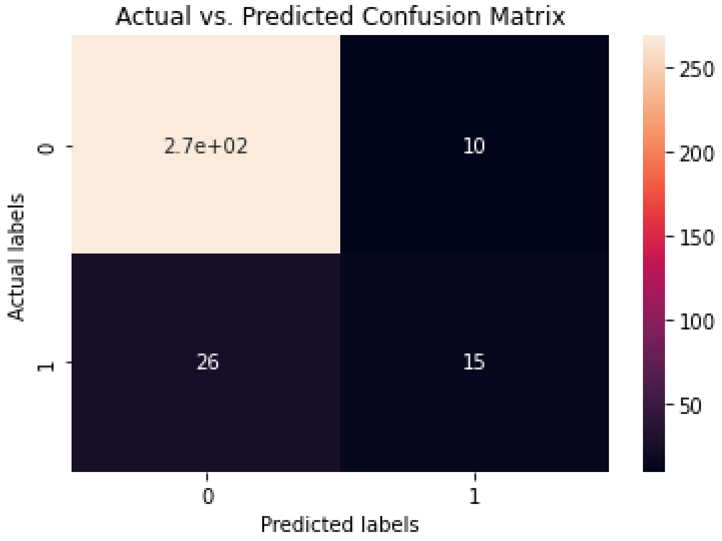
In [67]:
```
# Instantiating the model
log_reg=LogisticRegression()
log_reg.fit(X_train,y_train) # Fitting the model
y_pred=log_reg.predict(X_test)
```

In [68]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test,y_pred)
conf_matrix
```

Out[68]:
```
array([[269,  10],
       [ 26,  15]], dtype=int64)
```

In [69]:
```
# Plot Confusion Matrix
ax= plt.subplot()
sns.heatmap(conf_matrix,annot=True, ax= ax)

ax.set_xlabel('Predicted labels');ax.set_ylabel('Actual labels');
ax.set_title('Actual vs. Predicted Confusion Matrix');
plt.show()
```



In [80]:
```
# Checking the accuracy of training dataset
x_predlr = log_reg.predict(X_train)
accuracy_lr_train= accuracy_score(y_train,x_predlr)
print('Accuracy of training dataset : ',accuracy_lr_train)

# Checking the accuracy of testing dataset
accuracy_lr_test = accuracy_score(y_test,y_pred)
print('Accuracy of testing Model',accuracy_lr_test)
```

```
Accuracy of training dataset :  0.8795934323690383
Accuracy of testing Model 0.8875
```

In [71]:
```
# Acurracy , Precision and Recall
print(metrics.classification_report(y_test,y_pred))

# Area of Curve
predictions_prob_lr = log_reg.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test,predictions_prob_lr)
print('Area under curve :',auc(fpr,tpr))

# CV Score
scores_log_reg = cross_val_score(log_reg, X_train, y_train, cv=5)
print('Cross Validation Score:',scores_log_reg.mean())
```

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.94       279
           1       0.60      0.37      0.45        41

    accuracy                           0.89       320
   macro avg       0.76      0.67      0.70       320
weighted avg       0.87      0.89      0.88       320

Area under curve : 0.8832065740012238
Cross Validation Score: 0.8702113970588237
```

In [81]:
```
# Accuracy of Traing dataset is 87.8% and of Testing dataset is 88.75%,
# and both are low, so we can say that its an underfitted model.
```

# 2. Decision Tree

In [83]:
```
X.head(2)
```

Out[83]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.4 | 0.70 | 0.0 | 1.9 | 0.076 | 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 |
| **1** | 7.8 | 0.88 | 0.0 | 2.6 | 0.098 | 25.0 | 67.0 | 0.9968 | 3.20 | 0.68 | 9.8 |

In [84]:
```
Y.head(2)
```

Out[84]:
```
0    0
1    0
Name: quality, dtype: int64
```

In [85]:
```
# Splitting
X_traindt,X_testdt,y_traindt,y_testdt = train_test_split(X,Y,test_size=0.2, random_state = 2)
```

In [86]:
```
# Scaling
standard_Scaler=StandardScaler()
X_traindt = standard_Scaler.fit_transform(X_traindt)
X_testdt = standard_Scaler.transform(X_testdt)
```

### GINI

In [87]:

```
# Building Decision Tree Model
# GINI - To check the max depth
for mxdpt in range(2,10,1):
    model_DT_gini = DecisionTreeClassifier(random_state = 7, max_depth = mxdpt)
    model_DT_gini.fit(X_traindt, y_traindt)
    y_pred_new = model_DT_gini.predict(X_testdt)
    accuracy_score_new = accuracy_score(y_testdt, y_pred_new)
    print("Accuracy Score of Max Depth {0} is {1}".format(mxdpt,accuracy_score_new))
```

```
Accuracy Score of Max Depth 2 is 0.8875
Accuracy Score of Max Depth 3 is 0.903125
Accuracy Score of Max Depth 4 is 0.9
Accuracy Score of Max Depth 5 is 0.896875
Accuracy Score of Max Depth 6 is 0.90625
Accuracy Score of Max Depth 7 is 0.890625
Accuracy Score of Max Depth 8 is 0.878125
Accuracy Score of Max Depth 9 is 0.8875
```

In [88]:

```
# Considering Final Depth as 3
model_dt_dt = DecisionTreeClassifier(random_state=7,max_depth=3)
model_dt_dt.fit(X_traindt,y_traindt)

y_pred_dt = model_dt_dt.predict(X_testdt)
accuracy_score_dt = accuracy_score(y_testdt,y_pred_dt)
print('Accuracy Score for model with depth 3 is:',accuracy_score_dt)
```

```
Accuracy Score for model with depth 3 is: 0.903125
```

In [89]:
```
# Checking the accuracy of training dataset
x_preddt = model_dt_dt.predict(X_traindt)
accuracy_dt_train= accuracy_score(y_traindt,x_preddt)
print('Accuracy of training dataset : ',accuracy_dt_train)

# Checking the accuracy of testing dataset
accuracy_dt_test = accuracy_score(y_testdt,y_pred_dt)
print('Accuracy of testing Model',accuracy_dt_test)
```

```
Accuracy of training dataset :  0.890539483971853
Accuracy of testing Model 0.903125
```

In [90]:
```
# Acurracy , Precision and Recall for GINI
print(classification_report(y_testdt,y_pred_dt))

# Area Under Curve for GINI
predictions_prob = model_dt_dt.predict_proba(X_testdt)[:, 1] #Gini
fpr, tpr, _ = roc_curve(y_testdt,predictions_prob)
print('Area under Curve considering GINI ',auc(fpr,tpr))

# CV Score for GINI
scores_dt_dt = cross_val_score(model_dt_dt, X_traindt, y_traindt, cv=5)
print('Cross Validation Score considering GINI ',scores_dt_dt.mean())
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.95       279
           1       0.71      0.41      0.52        41

    accuracy                           0.90       320
   macro avg       0.81      0.69      0.73       320
weighted avg       0.89      0.90      0.89       320

Area under Curve considering GINI  0.8627939505201503
Cross Validation Score considering GINI  0.8756954656862745
```

### Entropy

In [94]:
```
# Entrophy - To check the max depth
for mxdpt in range(2,10,1):
    model_DT_en = DecisionTreeClassifier(random_state = 7, max_depth = mxdpt, criterion='entropy')
    model_DT_en.fit(X_traindt, y_traindt)
    y_pred_new = model_DT_en.predict(X_testdt)
    accuracy_score_new = accuracy_score(y_testdt, y_pred_new)
    print("Accuracy Score of Max Depth {0} is {1}".format(mxdpt,accuracy_score_new))
```

```
Accuracy Score of Max Depth 2 is 0.871875
Accuracy Score of Max Depth 3 is 0.8875
Accuracy Score of Max Depth 4 is 0.89375
Accuracy Score of Max Depth 5 is 0.903125
Accuracy Score of Max Depth 6 is 0.89375
Accuracy Score of Max Depth 7 is 0.890625
Accuracy Score of Max Depth 8 is 0.89375
Accuracy Score of Max Depth 9 is 0.890625
```

In [95]:
```
# Considering Final Depth as 5
model_dt_ent = DecisionTreeClassifier(random_state = 20,max_depth=5,criterion='entropy')
model_dt_ent.fit(X_traindt,y_traindt)

y_pred_ent = model_dt_ent.predict(X_testdt)

accuracy_score_en = accuracy_score(y_testdt,y_pred_ent)
print('Accuracy Score for model with depth 5 is:',accuracy_score_en)
```

```
Accuracy Score for model with depth 5 is: 0.90625
```

In [96]:
```
# Checking the accuracy of training dataset
x_predent = model_dt_ent.predict(X_traindt)
accuracy_ent_train = accuracy_score(y_traindt,x_predent)
print('Accuracy of training dataset : ',accuracy_ent_train)

# Checking the accuracy of testing dataset
accuracy_ent_test = accuracy_score(y_testdt,y_pred_ent)
print('Accuracy of testing Model',accuracy_ent_test)
```

```
Accuracy of training dataset :  0.9139953088350273
Accuracy of testing Model 0.90625
```

In [211]:
```
# Acurracy , Precision and Recall for Entropy
print(classification_report(y_testdt,y_pred_ent))

# Area Under Curve for Entropy
predictions_prob = model_dt_ent.predict_proba(X_testdt)[:, 1] #Gini
fpr, tpr, _ = roc_curve(y_testdt,predictions_prob)
print('Area under Curve considering Entropy ',auc(fpr,tpr))

# CV Score for Entropy
scores_dt_ent = cross_val_score(model_dt_ent, X_traindt, y_traindt, cv=5)
print('Cross Validation Score considering Entropy ',scores_dt_ent.mean())
```

```
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       279
           1       0.67      0.54      0.59        41

    accuracy                           0.91       320
   macro avg       0.80      0.75      0.77       320
weighted avg       0.90      0.91      0.90       320

Area under Curve considering Entropy  0.8731969577760295
Cross Validation Score considering Entropy  0.8639460784313725
```

In [ ]:
```
# Observation:
# Accuracy of Traing dataset is 91.39% and of Testing dataset is 90.62%, both accuracies are almost equal and high.
# Hence we can say that the model is neither underfitted nor overfitted or its perfectly fitted for the dataset.
```

### By using GridSearchCV

In [212]:
```
tree_para = {'criterion':['gini','entropy'],'max_depth':[1,2,3,4,5,6,7,8,9,10,11,12,15,20,30,40,50,70,90,120,150], 'rand
clf = GridSearchCV(DecisionTreeClassifier(), tree_para, cv=5)
clf.fit(X_traindt, y_traindt)
clf.best_params_
```

Out[212]:
```
{'criterion': 'entropy', 'max_depth': 6, 'random_state': 7}
```

In [213]:
```
# Considering Final Depth as 5
model_dt_ent_CV = DecisionTreeClassifier(random_state = 7,max_depth=6,criterion='entropy')
model_dt_ent_CV.fit(X_traindt,y_traindt)

y_pred_ent_CV = model_dt_ent_CV.predict(X_testdt)

accuracy_score_en_CV = accuracy_score(y_testdt,y_pred_ent_CV)
print('Accuracy Score for model with depth 6 is:',accuracy_score_en_CV)

# Acurracy , Precision and Recall
print(classification_report(y_testdt,y_pred_ent_CV))

# Area Under Curve
predictions_prob_CV = model_dt_ent_CV.predict_proba(X_testdt)[:, 1] #Gini
fpr, tpr, _ = roc_curve(y_testdt,predictions_prob_CV)
print('Area under Curve ',auc(fpr,tpr))

# CV Score
scores_dt_ent_CV = cross_val_score(model_dt_ent_CV, X_traindt, y_traindt, cv=5)
print('Cross Validation Score  ',scores_dt_ent_CV.mean())
```

```
Accuracy Score for model with depth 6 is: 0.89375
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       279
           1       0.59      0.59      0.59        41

    accuracy                           0.89       320
   macro avg       0.76      0.76      0.76       320
weighted avg       0.89      0.89      0.89       320

Area under Curve  0.8491563947897544
Cross Validation Score   0.8827144607843138
```

# 3. Random Forest Tree

In [16]:
```
# Splitting
X_trainrf, X_testrf, y_trainrf, y_testrf = train_test_split(X,Y,test_size = 0.20,random_state = 3)
```

In [17]:
```
# Scaling
standard_Scaler=StandardScaler()
X_trainrf = standard_Scaler.fit_transform(X_trainrf)
X_testrf = standard_Scaler.transform(X_testrf)
```

In [18]:
```
model_RF = RandomForestClassifier() #Instantiating Model
```

### GridSearchCV

In [19]:
```
param_dist = {'max_depth': [2, 3, 4],
              'max_features': ['auto', 'sqrt', 'log2', None],
                'bootstrap' : [True, False],
              'criterion': ['gini', 'entropy']}

cv_rf = GridSearchCV(model_RF, cv = 10,
                     param_grid=param_dist,
                     n_jobs = 3)

cv_rf.fit(X_trainrf, y_trainrf)
print('Best Parameters using grid search: \n', cv_rf.best_params_)
```

```
Best Parameters using grid search:
 {'bootstrap': True, 'criterion': 'gini', 'max_depth': 4, 'max_features': None}
```

In [20]:
```
model_RF.set_params(criterion = 'gini',
                  max_features = None,
                  bootstrap = False,
                  max_depth = 4)
```

Out[20]:
```
RandomForestClassifier(bootstrap=False, max_depth=4, max_features=None)
```

In [21]:
```
model_RF.fit(X_trainrf, y_trainrf) # Fitting the model
y_predrf = model_RF.predict(X_testrf)
```

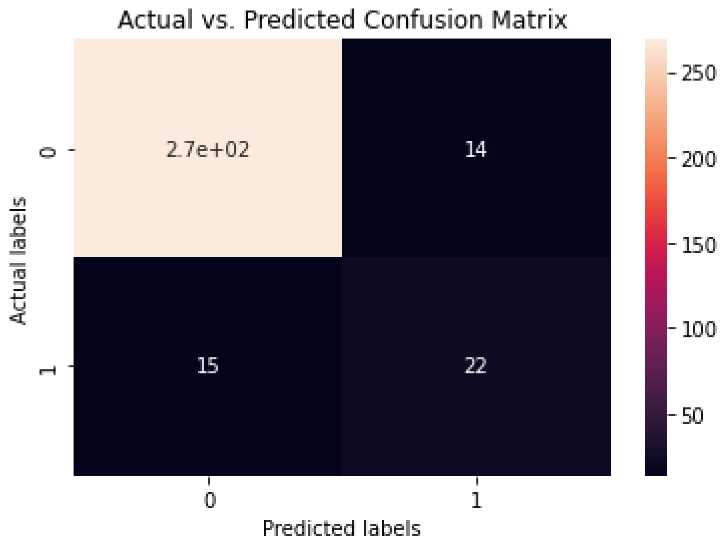
In [23]:
```
cm=confusion_matrix(y_testrf,y_predrf)
print(cm)

ax= plt.subplot()
sns.heatmap(cm,annot=True, ax= ax)

ax.set_xlabel('Predicted labels');ax.set_ylabel('Actual labels');
ax.set_title('Actual vs. Predicted Confusion Matrix');

plt.show()
```

```
[[269  14]
 [ 15  22]]
```



In [79]:
```
# Checking the accuracy of training dataset
pred_x_train = model_RF.predict(X_trainrf)
accuracy_rf_x_train = accuracy_score(y_trainrf,pred_x_train)
print('Accuracy of training model: ',accuracy_rf_x_train)

# Checking the accuracy of testing dataset
accuracy_rf = accuracy_score(y_testrf,y_predrf)
print('Accuracy of testing Model',accuracy_rf)
```

```
Accuracy of training model:  0.9163408913213448
Accuracy of testing Model 0.909375
```

In [27]:
```
# Area under Curve
predictions_prob_rf = model_RF.predict_proba(X_testrf)[:, 1]
fpr, tpr, _ = roc_curve(y_testrf,predictions_prob_rf)
print('Area under Curve',auc(fpr,tpr))

# CV Score
scores_rf = cross_val_score(model_RF, X_trainrf, y_trainrf, cv=5)
print('Cross Validation Score',scores_rf.mean())

# Acurracy , Precision and Recall
print(classification_report(y_testrf,y_predrf))
```

```
Accuracy of Random Forest Model 0.909375
Area under Curve 0.8567472065705282
Cross Validation Score 0.851439950980392
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       283
           1       0.61      0.59      0.60        37

    accuracy                           0.91       320
   macro avg       0.78      0.77      0.78       320
weighted avg       0.91      0.91      0.91       320
```

In [28]:
```
# Observation:
# Accuracy of Traing dataset is 91.63 and of Testing dataset is 90.93, both accuracies are almost equal and high.
# Hence we can say that the model is neither underfitted nor overfitted or its perfectly fitted for the dataset.
```

# 4. Naive Bayes Model

In [30]:
```
# Splitting
X_trainnb, X_testnb, y_trainnb, y_testnb = train_test_split(X,Y,test_size = 0.20,random_state = 42)
```

In [31]:
```
# Scaling
standard_Scaler=StandardScaler()
X_trainnb = standard_Scaler.fit_transform(X_trainnb)
X_testnb = standard_Scaler.transform(X_testnb)
```

In [32]:
```
model_NB = GaussianNB() # Instantiating the model
```

In [33]:
```
model_NB.fit(X_trainnb, y_trainnb) # Fitting the model
```

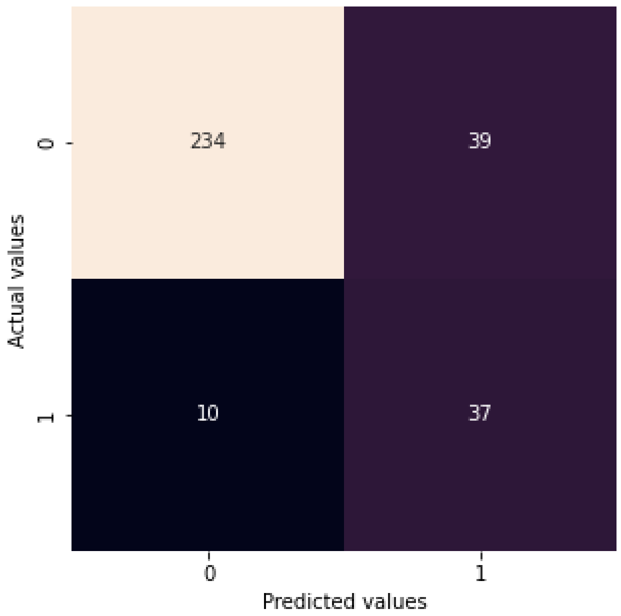
Out[33]:
```
GaussianNB()
```

In [34]:
```
# Confusion Matrix
y_prednb = model_NB.predict(X_testnb)
mat = confusion_matrix(y_testnb, y_prednb)
print(mat)
```

```
[[234  39]
 [ 10  37]]
```

In [35]:
```
# Plot Confusion Matrix
plt.figure(figsize=(10,5))
names = np.unique(y_prednb)
sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=names, yticklabels=names)
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.plot()
```

Out[35]: []



In [73]:
```
# Checking the accuracy of traing dataset
x_prednb = model_NB.predict(X_trainnb)
accuracy_nb_train= accuracy_score(y_trainnb,x_prednb)
print('Accuracy of training dataset : ',accuracy_nb_train)

# Checking the accuracy of testing dataset
accuracy_nb = accuracy_score(y_testnb,y_prednb)
print('Accuracy of testing dataset: ',accuracy_nb)
```

```
Accuracy of training dataset :  0.8389366692728695
Accuracy of testing dataset:  0.846875
```

In [171]:
```
# Accuracy, Precision, recall
print(classification_report(y_prednb, y_testnb))

# Area under Curve
predictions_prob_NB = model_NB.predict_proba(X_testnb)[:, 1]
fpr, tpr, _ = roc_curve(y_testnb,predictions_prob_NB)
print('Area under curve :',auc(fpr,tpr))

# CV Score
scores_NB = cross_val_score(model_NB, X_trainnb, y_trainnb, cv=5)
print('Cross Validation Score',scores_NB.mean())
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.91       244
           1       0.79      0.49      0.60        76

    accuracy                           0.85       320
   macro avg       0.82      0.72      0.75       320
weighted avg       0.84      0.85      0.83       320

Area under curve : 0.8604161795651157
Cross Validation Score 0.8373621323529411
```

In [ ]:
```
# Observation:
# Accuracy of Traing dataset is 83.89% and of Testing dataset is 84.68%,
# and both are low, so we can say that its an underfitted model.
```

# 5. K Nearest Neighbours Model

In [39]:
```
# Splitting
X_trainknn, X_testknn, y_trainknn, y_testknn = train_test_split(X,Y,test_size = 0.20,random_state = 42)
```

In [40]:
```
# Scaling
standard_Scaler=StandardScaler()
X_trainknn = standard_Scaler.fit_transform(X_trainknn)
X_testknn = standard_Scaler.transform(X_testknn)
```

In [41]:
```
knn = KNeighborsClassifier(n_neighbors=3)    # Instantiate
knn.fit(X_trainknn,y_trainknn)                     # fit
y_predknn = knn.predict(X_testknn)
```

In [42]:
```
print(confusion_matrix(y_testknn,y_predknn))
```

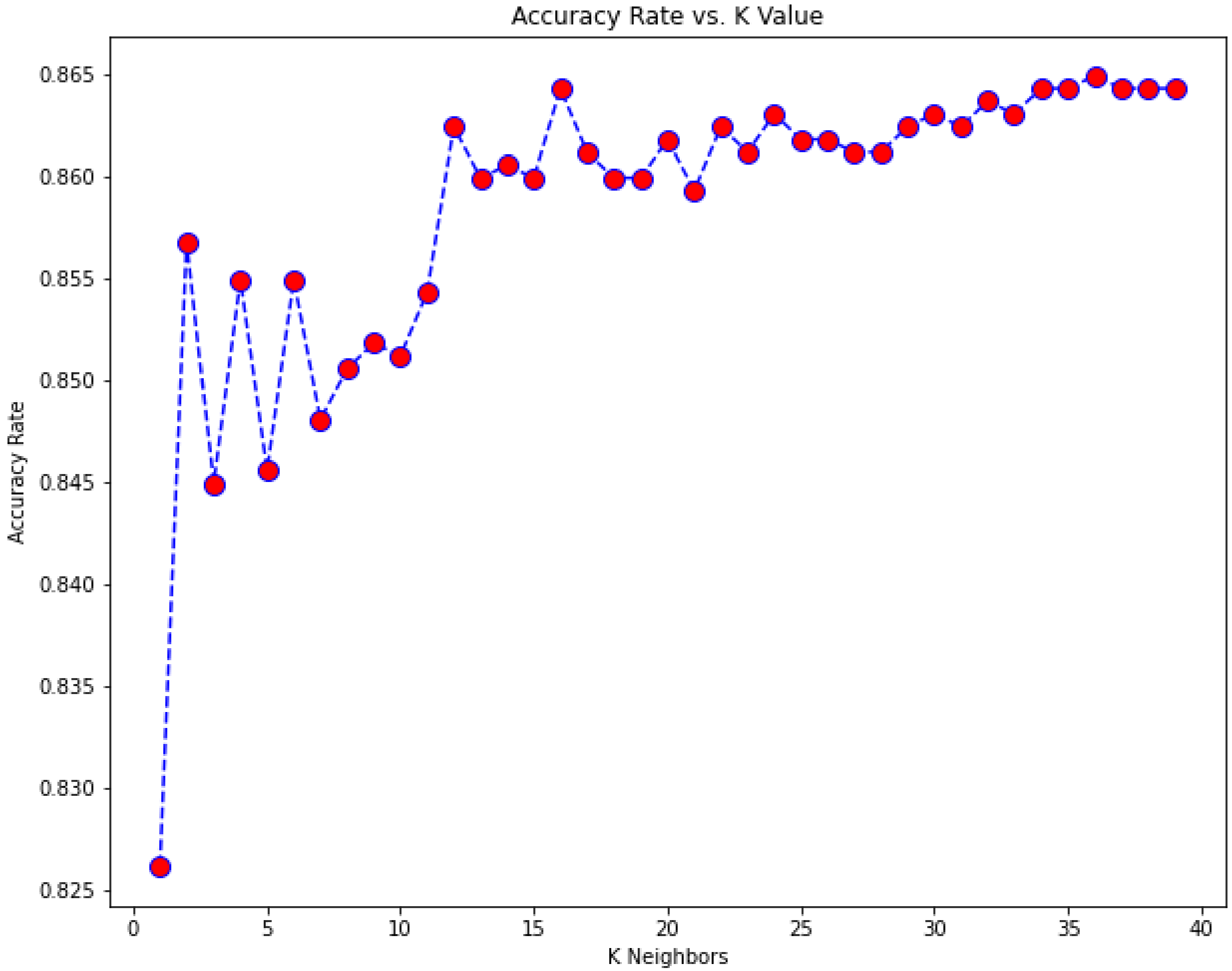
```
[[259  14]
 [ 27  20]]
```

In [59]:
```
accuracy_rate = []
for i in range(1,40):    # May take some time
    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,X,Y,cv=10)
    accuracy_rate.append(score.mean())
print(accuracy_rate)
```

```
[0.8261320754716982, 0.8567845911949685, 0.8449095911949686, 0.8549213836477989, 0.8455424528301887, 0.854913522012578
5, 0.8480345911949685, 0.8505424528301886, 0.8517924528301887, 0.8511674528301887, 0.8542924528301887, 0.86241745283018
88, 0.8599135220125786, 0.8605385220125786, 0.8599135220125786, 0.8642924528301886, 0.8611674528301887, 0.8599174528301
887, 0.8599174528301885, 0.8617924528301886, 0.8592924528301887, 0.8624174528301888, 0.8611674528301887, 0.863042452830
1888, 0.8617924528301886, 0.8617924528301886, 0.8611674528301887, 0.8611674528301887, 0.8624174528301886, 0.86304245283
01888, 0.8624174528301886, 0.8636674528301886, 0.8630424528301888, 0.8642924528301886, 0.8642924528301886, 0.8649174528
301886, 0.8642924528301886, 0.8642924528301886, 0.8642924528301886]
```

In [60]:
```
#To find optimal value of k
plt.figure(figsize = (10,8))
plt.plot(range(1,40),accuracy_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Accuracy Rate vs. K Value')
plt.xlabel('K Neighbors')
plt.ylabel('Accuracy Rate')
```

Out[60]: Text(0, 0.5, 'Accuracy Rate')



In [61]:
```
# Optimal value of K = 18
knn = KNeighborsClassifier(n_neighbors=18)    #Instantiate
knn.fit(X_trainknn,y_trainknn)                        #fit
y_predknn_18 = knn.predict(X_testknn)                   #Predict
```

In [76]:
```
# Checking the accuracy of traing dataset
x_predknn = knn.predict(X_trainknn)
accuracy_knn_train= accuracy_score(y_trainknn,x_predknn)
print('Accuracy of training dataset : ',accuracy_knn_train)

# Checking the accuracy of testing dataset
accuracy_knn = accuracy_score(y_testknn,y_predknn)
print('Accuracy of testing dataset: ',accuracy_knn)
```

```
Accuracy of training dataset :  0.8780297107114934
Accuracy of testing dataset:  0.871875
```

In [63]:
```
# Accuracy, Recall, Precision
print(classification_report(y_testknn,y_predknn_18))

# Area Under Curve
y_pred_prob_knn = knn.predict_proba(X_testknn)[::, 1]
fpr2, tpr2, _ = roc_curve(y_testknn,
                          y_pred_prob_knn)
print('Area under curve',auc(fpr2,tpr2))

# CV Score
scores_KNN = cross_val_score(knn, X_trainknn, y_trainknn, cv=5)
print('Cross Validation Score',scores_KNN.mean())
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.92       273
           1       0.59      0.21      0.31        47

    accuracy                           0.86       320
   macro avg       0.73      0.59      0.62       320
weighted avg       0.84      0.86      0.83       320

Area under curve 0.881264125944977
Cross Validation Score 0.8749019607843138
```

In [ ]:
```
# After scaling the features, macro avg, weighted avg and precision value improved in KNN model
# Accuracy of Traing dataset is 87.8% and of Testing dataset is 87.18%, both accuracies are almost same, but are low.
# Can be said that its an underfitted model.
```

# Optimised Code Including all Models

In [200]:
```
def classify(model, x_trainn, x_testt, y_trainn, y_testt, y_predd):
    #Train the model
    model.fit(x_trainn, y_trainn)

    #Accuracy
    accuracy = accuracy_score(y_testt,y_predd)
    print('Accuracy:', accuracy)

    #CV Score
    scores = cross_val_score(model, x_trainn, y_trainn, cv=5)
    print('Cross Validation Score',scores.mean())

    # Area Under Curve
    y_predd = model.predict_proba(x_testt)[::, 1]
    fpr2, tpr2, _ = roc_curve(y_testt,y_predd)
    print('Area under curve',auc(fpr2,tpr2))
```

In [215]:
```
#Logistic Regression
log_reg=LogisticRegression()
classify(log_reg, X_train, X_test, y_train, y_test, y_pred)
```

```
Accuracy: 0.8875
Cross Validation Score 0.8702113970588237
Area under curve 0.8832065740012238
```

In [231]:
```
#Decision Tree
#Gini
print('For Gini')
model_DT=DecisionTreeClassifier(max_depth=5,criterion='gini')
classify(model_DT, X_traindt, X_testdt, y_traindt, y_testdt, y_pred_dt)
#Entropy
print('For Entropy')
model_DT_ent=DecisionTreeClassifier(max_depth=5,criterion='entropy')
classify(model_DT_ent, X_traindt, X_testdt, y_traindt, y_testdt, y_pred_ent)
```

```
For Gini
Accuracy: 0.903125
Cross Validation Score 0.8780330882352942
Area under curve 0.8222309642451263
For Entropy
Accuracy: 0.90625
Cross Validation Score 0.8662898284313725
Area under curve 0.8731969577760295
```

In [229]:
```
#Random Forest
model_RF=RandomForestClassifier()
classify(model_RF, X_trainrf, X_testrf, y_trainrf, y_testrf, y_predrf)
```

```
Accuracy: 0.909375
Cross Validation Score 0.901467524509804
Area under curve 0.9312864100849967
```

In [218]:
```
#Naive Bayes Model
model_NB = GaussianNB()
classify(model_NB, X_trainnb, X_testnb, y_trainnb, y_testnb, y_prednb)
```

```
Accuracy: 0.846875
Cross Validation Score 0.8373621323529411
Area under curve 0.8604161795651157
```

In [219]:
```
#KNN Model
model_knn = KNeighborsClassifier(n_neighbors=3)
classify(model_knn, X_trainknn, X_testknn, y_trainknn, y_testknn, y_predknn)
```

```
Accuracy: 0.871875
Cross Validation Score 0.8686519607843138
Area under curve 0.8218767048554283
```

In [ ]:
```
# Observations on Models :
# 1) KNN and Logistic Regression are affected by Feature Scaling most as compare to rest of the models used,
#    still the accuracy they are providing is around 85%.
# 2) Decision Tree and Random Forest Tree Models are more accurate models with accuracy = 90 % and 91% respectively
#     as compared to rest of the models used.
```