Objective:

- To present the findings on Wine Quality Data using Python.
- To classify different Wine Qualities using Machine Learning Models.

Dataset selection:

Student ID: 20231278
Mod (78,2) = 0
Hence, selecting the **Wine Quality Dataset**. This report will focus on the following problem set:

1. Illustrate the following summary information in Python.
    a. Read the data and construct some appropriate graphs for each of the variables with interpretation.
    b. Conduct the EDA for the selected study data.
    c. Construct the pair plots for different wine qualities (Good >5, Bad<=5) in Wine Quality Data.
2. Classify the different wine qualities (Good >5, Bad<=5) in Wine Quality Data. Hence, estimate the Logistic Regression model to predict the probability of good wine quality in Wine Quality Data. Interpret the fitted model.
3. Split the entire dataset into the training dataset (70%) and the test dataset (30%). To classify the different wine qualities (Good >5, Bad<=5) in Wine Quality Data use the following supervised learning methods for the training dataset:
    a. Run the Logistic Regression model to classify the different categories. Find the confusion matrix and ROC curve. Hence, calculate and interpret the predictive value positive and negative, Accuracy, Sensitivity, and Specificity of the test.
    b. Run the Decision Trees to classify the different categories. Find the confusion matrix and ROC curve. Hence, calculate and interpret the predictive value positive and negative, Accuracy, Sensitivity, and Specificity of the test.
    c. Run the Random Forest to classify the different categories. Find the confusion matrix and ROC curve. Hence, calculate and interpret the predictive value positive and negative, Accuracy, Sensitivity, and Specificity of the test.
    d. Run the Support Vector Machines to classify the different categories. Find the confusion matrix and ROC curve. Hence, calculate and interpret the predictive value positive and negative, Accuracy, Sensitivity, and Specificity of the test.
    e. Hence, evaluate the best-performed model for this study using the test dataset.

1.1 Data Description:

Wine Quality Data set is divided into 2 datasets: red_wine and white_wine. There are 1599 rows in red_wine data set, and 4898 rows in white dataset. For both datasets, the columns are the same: 12.

```
red.head()
```

	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

```
white.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

So, first append both datasets, using the following code:

```
#Combining two data sets
total_wine = red.append(white, ignore_index = True)
```

Using the last two digits of my ID(78) with a decimal point, appending the following 5rows with total_wine.

```
#Creating the dataframe with following 5 rows using last two digit of my ID: 78

datarowSeries = [pd.Series([7.8,0.88, 0.0, 1.9, .09, 25, 67, .991, 3.22, .68, 9.8, 5],index = total_wine.columns),
                 pd.Series([7.8+0.78,0.88+0.78, 0.0+0.78, 1.9, .09+0.78, 25+0.78, 67+0.78, .991+0.78, 3.22, .68+0.78, 9.8+0.78, 5],index = total_wine.columns),
                 pd.Series([7.2+0.78,0.83+0.78, 0.01+0.78, 2.2, .19+0.78, 15+0.78, 60+0.78, .996+0.78, 3.52, .55+0.78, 9.6+0.78, 6],index = total_wine.columns),
                 pd.Series([7.9+0.78,0.89+0.78, 0.01+0.78, 1.7, .08+0.78, 22+0.78, 57+0.78, .997+0.78, 3.26, .64+0.78, 9.8+0.78, 2],index = total_wine.columns),
                 pd.Series([7.7+0.78,0.86+0.78, 0.02+0.78, 2.3, .07+0.78, 11+0.78, 38+0.78, .994+0.78, 3.12, .08+0.78, 9.4+0.78, 3],index = total_wine.columns),
                 ]
```

So, the final DataFrame is denoted as df(shown below):

```
#Final data set: red wine data, white wine data, and 5 new rows using ID: 78
df = total_wine.append(datarowSeries, ignore_index = True)
df.head(5)
```

	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

Reallocate the wine quality value as 0: 0 to 5 (Bad) and 1: 6to 10 (Good Quality). Using a "modify_quality" function.

```
def modify_quality(value):
  '''
  Reallocate the wine quality value as 0: 0 to 5 ( Average quality)
  and 1: 6to 10 (Good Quality)
  >>> modify_quality(5)
  0
  >>> modify_quality(7)
  1

  '''
  if value <= 5:
    return 0
  else:
    return 1

# Directly assign the result of the function to the 'quality' column
df['quality'] = df['quality'].apply(modify_quality)
```

```
#Final DataFrame
df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.40	0.70	0.00	1.9	0.076	11.00	34.00	0.9978	3.51	0.56	9.40	0
1	7.80	0.88	0.00	2.6	0.098	25.00	67.00	0.9968	3.20	0.68	9.80	0
2	7.80	0.76	0.04	2.3	0.092	15.00	54.00	0.9970	3.26	0.65	9.80	0
3	11.20	0.28	0.56	1.9	0.075	17.00	60.00	0.9980	3.16	0.58	9.80	1
4	7.40	0.70	0.00	1.9	0.076	11.00	34.00	0.9978	3.51	0.56	9.40	0

After completing these tasks, the data is completed with 6502 rows and 12 columns. Table 1 represents all the variable description with their value level, level of measurements and suitable measures.

Table 1: Variable's Summary information of Wine Quality Dataset

Variable Name	**Variable Description**	**Value Level**	**Level of Measurement**	**Appropriate Measures**
Fixed Acidity	Grams of tartaric acid per liter	Continuous	Ratio	Mean
Volatile Acidity	Grams of volatile acetic acid per liter	Continuous	Ratio	Mean
Citric Acid	Grams of citric acid per liter	Continuous	Ratio	Mean
Residual Sugar	Grams of residual sugar per liter	Continuous	Ratio	Mean
Chlorides	Milligrams of chlorides per liter	Continuous	Ratio	Mean
Free Sulfur Dioxide	Milligrams of free sulfur dioxide per liter	Continuous	Ratio	Mean
Total Sulfur Dioxide	Milligrams of total sulfur dioxide per liter	Continuous	Ratio	Mean
Density	Density of the wine at 20°C	Continuous	Ratio	Mean
pH	Hydrogen ion potential	Continuous	Ratio	Mean
Sulphates	Milligrams of total sulphates per liter	Continuous	Ratio	Mean
Alcohol	Percentage of alcohol	Continuous	Ratio	Mean
Quality	Overall quality of the wine (as rated by human experts)	Discrete 0 = Bad 1 = Good	Ordinal	Mode

1.2 Data Cleaning:

To check missing values of the dataset the following codes are used.

```
df.shape
```

```
(6502, 12)
```

```
#null values
df.isnull().sum()
```

```
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
```

```
#data type
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6502 entries, 0 to 6501
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         6502 non-null   float64
 1   volatile acidity      6502 non-null   float64
 2   citric acid           6502 non-null   float64
 3   residual sugar        6502 non-null   float64
 4   chlorides             6502 non-null   float64
 5   free sulfur dioxide   6502 non-null   float64
 6   total sulfur dioxide  6502 non-null   float64
 7   density               6502 non-null   float64
 8   pH                    6502 non-null   float64
 9   sulphates             6502 non-null   float64
 10  alcohol               6502 non-null   float64
 11  quality               6502 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 609.7 KB
```

There are 12 variables and 6502 observations and no missing values were found in the dataset. Now, to understand the dataset more in-depth, the central tendency, spread, and potential skewness of the data, using df.describe()

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	6502.000000	6502.000000	6502.000000	6502.000000	6502.000000	6502.000000	6502.000000	6502.000000	6502.000000	6502.000000	6502.000000	6502.000000
mean	7.216144	0.340552	0.318874	5.440588	0.056551	30.517398	115.700495	0.995176	3.218539	0.531744	10.491656	0.632728
std	1.296323	0.167861	0.145785	4.756935	0.040659	17.745527	56.523208	0.019568	0.160774	0.149994	1.192292	0.482098
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000	0.000000
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992340	3.110000	0.430000	9.500000	0.000000
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000	10.300000	1.000000
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.996997	3.320000	0.600000	11.300000	1.000000
max	15.900000	1.670000	1.660000	65.800000	0.970000	289.000000	440.000000	1.777000	4.010000	2.000000	14.900000	1.000000

Highlights:

- The median is smaller than mean for most of the variables. Skewness of the variables:
- For alcohol, slightly to the right skewed
- Residual sugar is right skewed
- Some of the data showed about symmetry: fixed acidity, volatile, citric acid, chlorides, free sulfur dioxide,total suldur dioxide, pH, sulphates,
- **Data variations:** Based on the standard deviations, residual sugar, free sulfur dioxide, and total sulfur dioxide exhibit high variability in the data. Among these, total sulfur dioxide has the highest standard deviation, indicating a wider spread of values compared to the other variables.

- While the standard deviations for the other variables suggest a more moderate range, further investigation into the expected ranges for these wine quality variables would be beneficial.
- **Possible outliers:** From the summary statistics, we can not pinpoint specific outliers, we can identify which may need further investigation. This includes fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol.

The following boxplot will check the outliers in the dataset:

```
# Exclude the 'quality' column from the list of columns for the boxplot
plt.figure(figsize=(10, 6))
df.drop('quality', axis = 1)
df.boxplot()
plt.title('Boxplot for the variables related to the quality of the wine')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

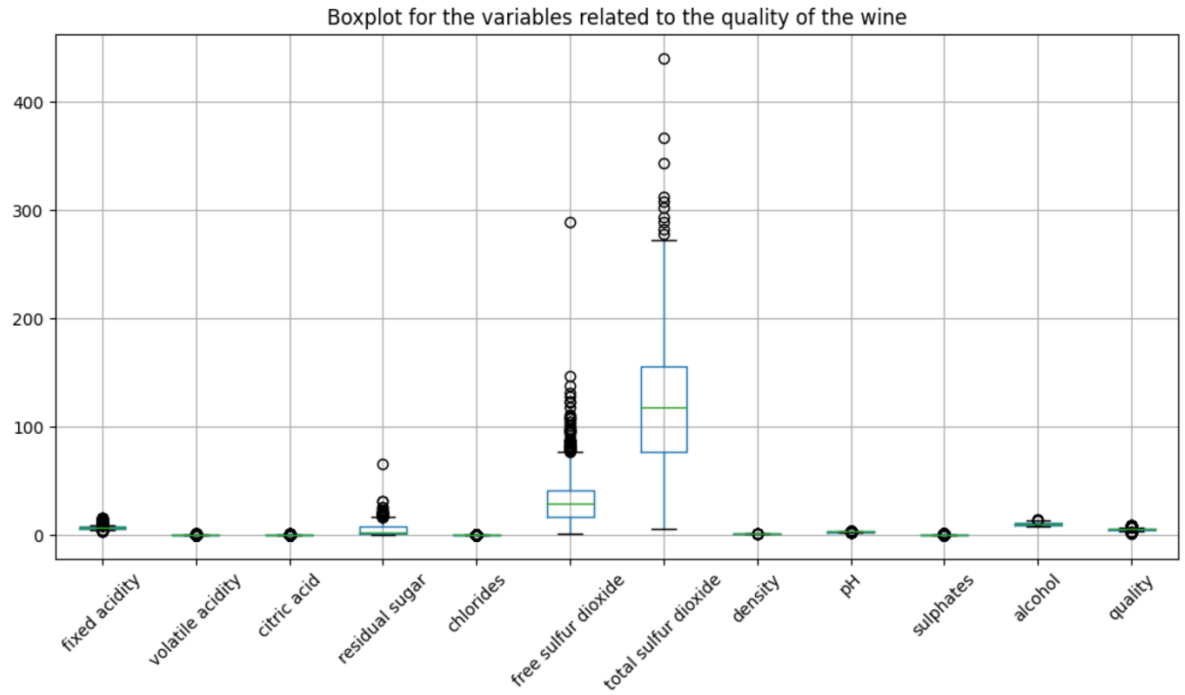


Figure 1: Boxplot for the variables related to the quality of the wine

Figure 1 reveals outliers in all variables. However, "residual sugar," "free sulfur dioxide," and "total sulfur dioxide" exhibit a significantly higher number of outliers compared to the other variables.

1.3 Univariate Analysis:

In this data, only the quality variable is ordinal (0: Bad, 1: Good). All the other 11 variables are ratio. Hence, for the 11 Ratio variables a histogram and a boxplot is created, and for the quality variable, a pie chart is shown.

1.3.1 Graphical Representation:

Fixed Acidity Distribut ion in Wine (Histogram & Boxplot)

Code:

```
# Fixed Acidity

x = 'fixed acidity'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

ax1.hist(df[x], bins = 9)
ax1.set_xlabel(x)
ax1.set_ylabel('frequency')

ax2.boxplot(df[x])
ax2.set_xlabel(x)

plt.suptitle("Fixed Acidity Distribution in Wine (Histogram & Boxplot)")


plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

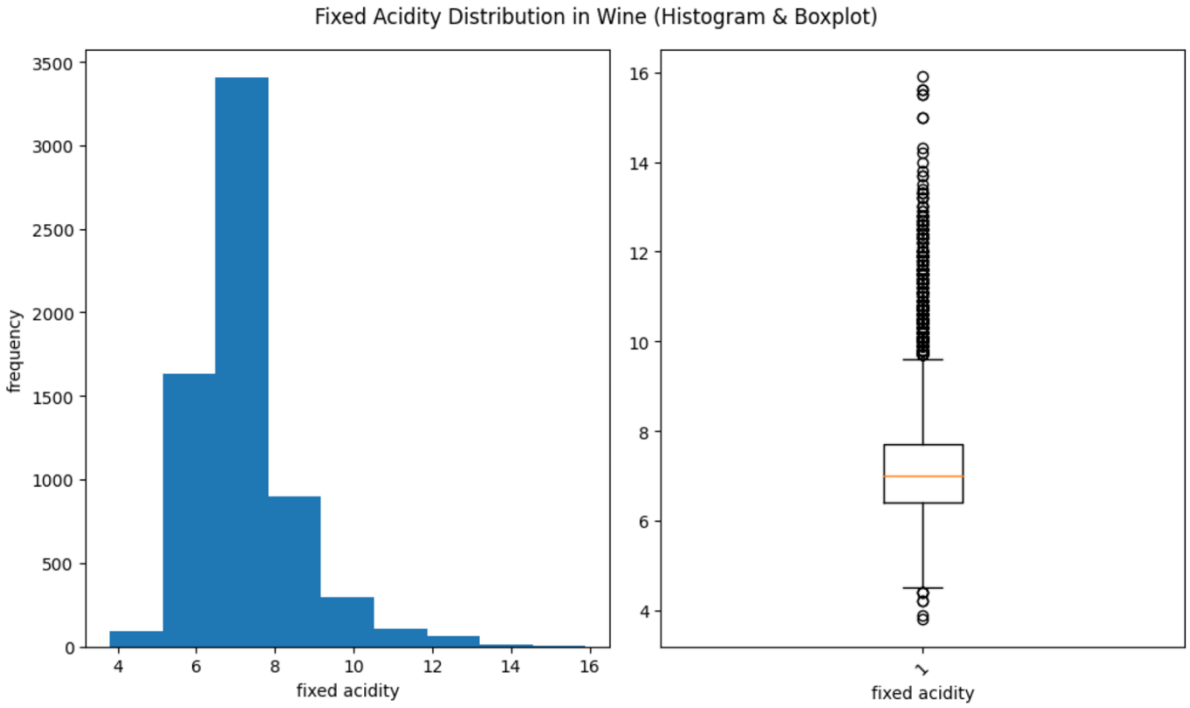


Figure 2: Fixed Acidity Distribution in Wine (Histogram & Boxplot)

Volatile Acidity Distribution in Wine (Histogram & Boxplot)

```
# Volatile Acidity

x = 'volatile acidity'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

ax1.hist(df[x], bins = 9)
ax1.set_xlabel(x)
ax1.set_ylabel('frequency')

ax2.boxplot(df[x])
ax2.set_xlabel(x)

plt.suptitle("Volatile Acidity Distribution in Wine (Histogram & Boxplot)")


plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

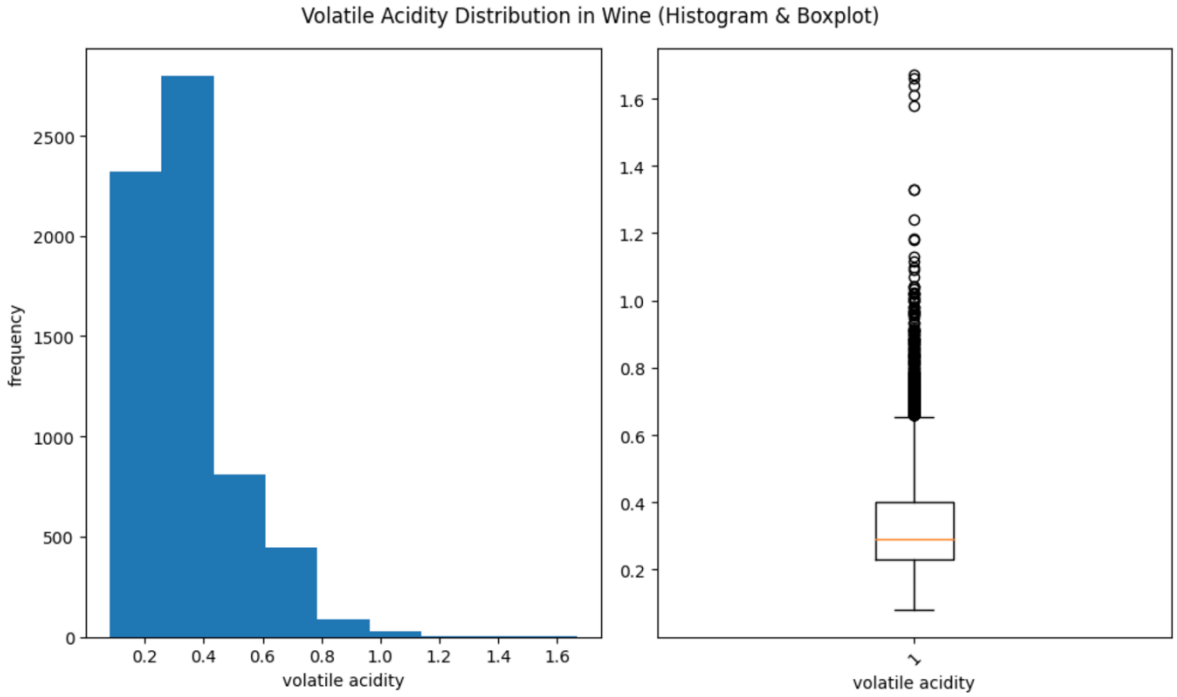


Figure 3: Volatile Acidity Distribution in Wine (Histogram & Boxplot)

Citric Acid Distribution in Wine (Histogram & Boxplot)

```
# Citric Acid

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))
x = 'citric acid'

ax1.hist(df[x], bins = 9)
ax1.set_xlabel(x)
ax1.set_ylabel('frequency')

ax2.boxplot(df[x])
ax2.set_xlabel(x)

plt.suptitle("Citric Acid Distribution in Wine (Histogram & Boxplot)")


plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

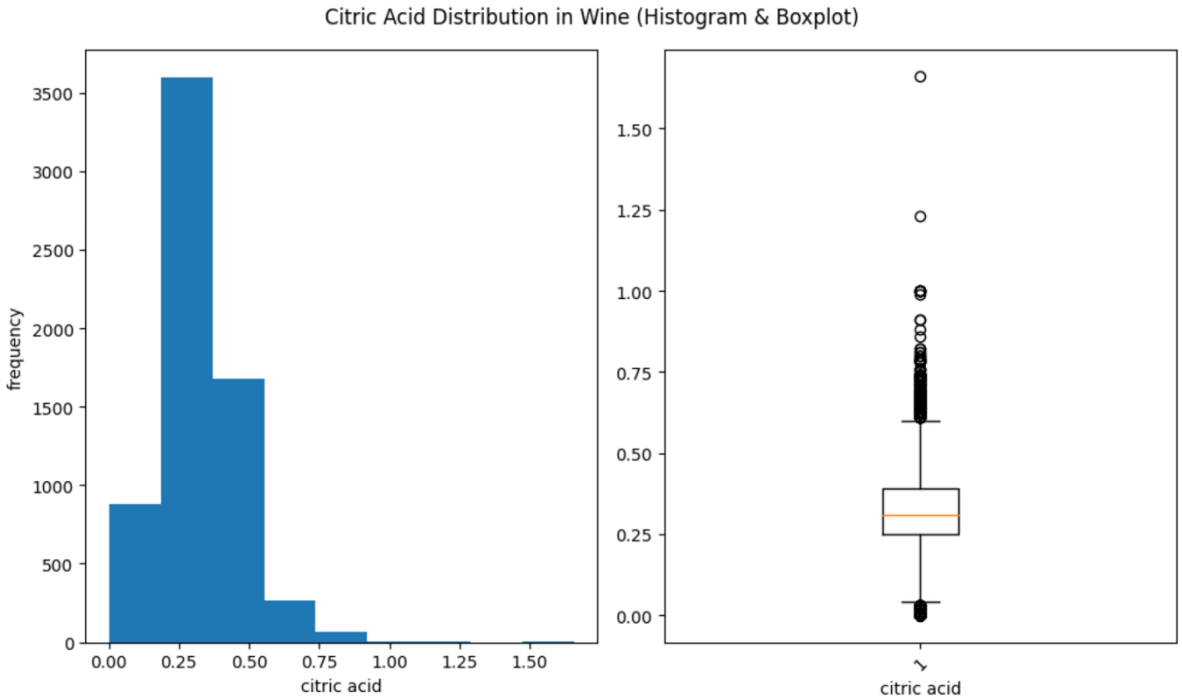


Figure 4: Citric Acid Distribution in Wine (Histogram & Boxplot)

Residual Sugar Distribution in Wine (Histogram & Boxplot)

```
# residual sugar

x = 'residual sugar'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

ax1.hist(df[x], bins = 9)
ax1.set_xlabel(x)
ax1.set_ylabel('frequency')

ax2.boxplot(df[x])
ax2.set_xlabel(x)

plt.suptitle("Residual Sugar Distribution in Wine (Histogram & Boxplot)")


plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

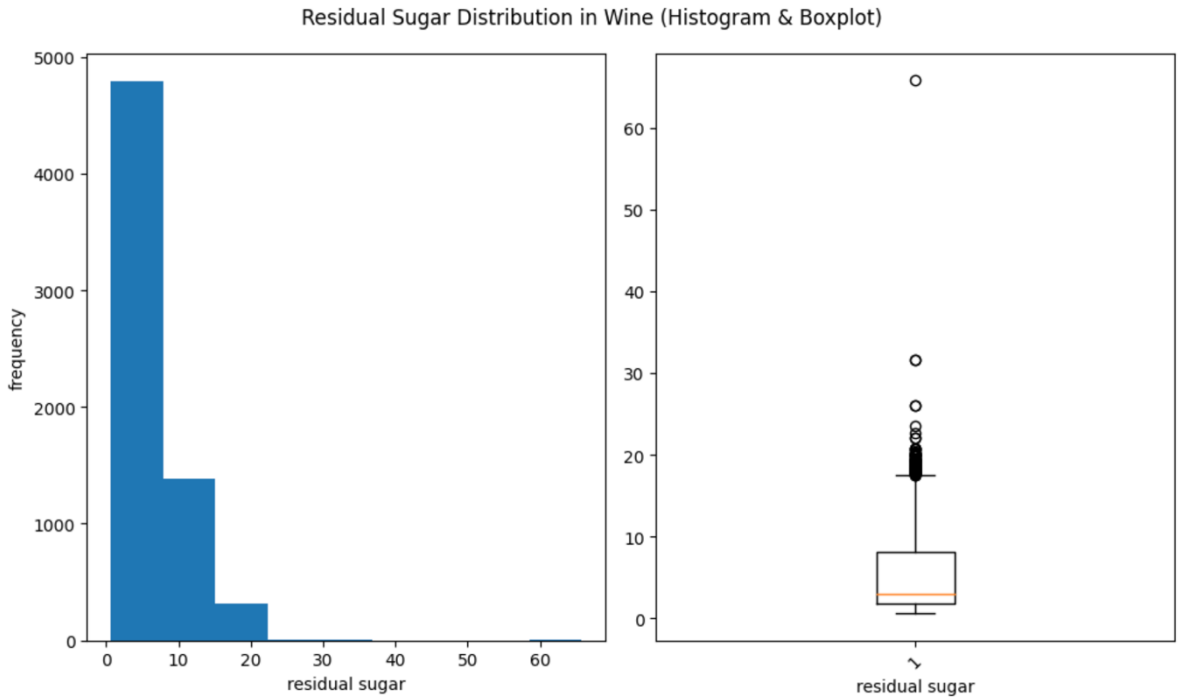


Figure 5: Residual Sugar Distribution in Wine (Histogram & Boxplot)

Chlorides Distribution in Wine (Histogram & Boxplot)

```
# chlorides

x = 'chlorides'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

ax1.hist(df[x], bins = 9)
ax1.set_xlabel(x)
ax1.set_ylabel('frequency')

ax2.boxplot(df[x])
ax2.set_xlabel(x)

plt.suptitle("Chlorides Distribution in Wine (Histogram & Boxplot)")


plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

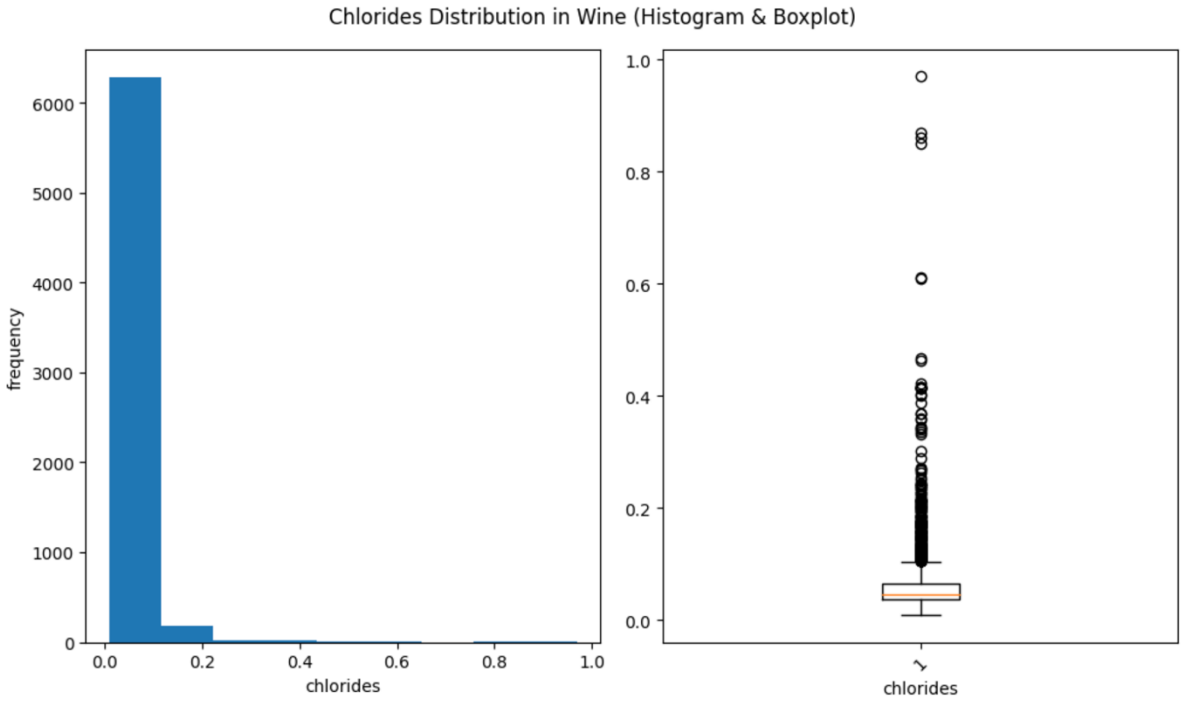
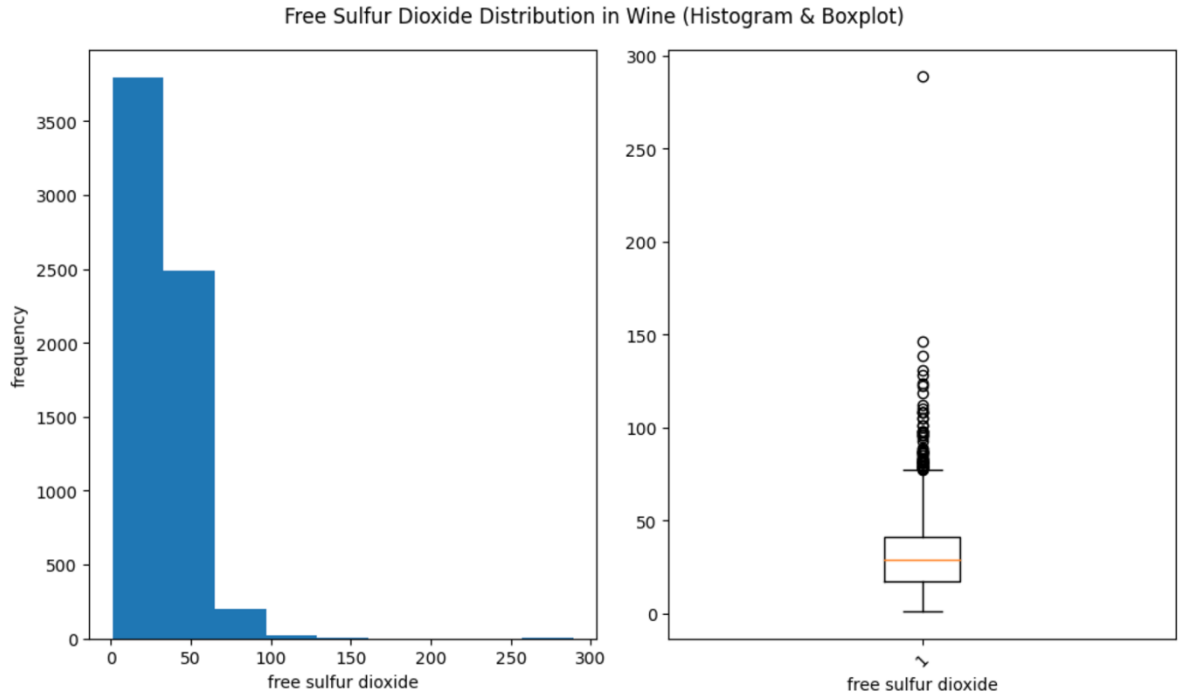


Figure 6: Chlorides Distribution in Wine (Histogram & Boxplot)

Free Sulfur Dioxide Distribution in Wine (Histogram & Boxplot)

```python
# free sulfur dioxide

x = 'free sulfur dioxide'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

ax1.hist(df[x], bins = 9)
ax1.set_xlabel(x)
ax1.set_ylabel('frequency')

ax2.boxplot(df[x])
ax2.set_xlabel(x)

plt.suptitle("Free Sulfur Dioxide Distribution in Wine (Histogram & Boxplot)")


plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```



Total Sulfur Dioxide Distribution in Wine (Histogram & Boxplot)

```
# total sulfur dioxide

x = 'total sulfur dioxide'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

ax1.hist(df[x], bins = 9)
ax1.set_xlabel(x)
ax1.set_ylabel('frequency')

ax2.boxplot(df[x])
ax2.set_xlabel(x)

plt.suptitle("Total Sulfur Dioxide Distribution in Wine (Histogram & Boxplot)")


plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

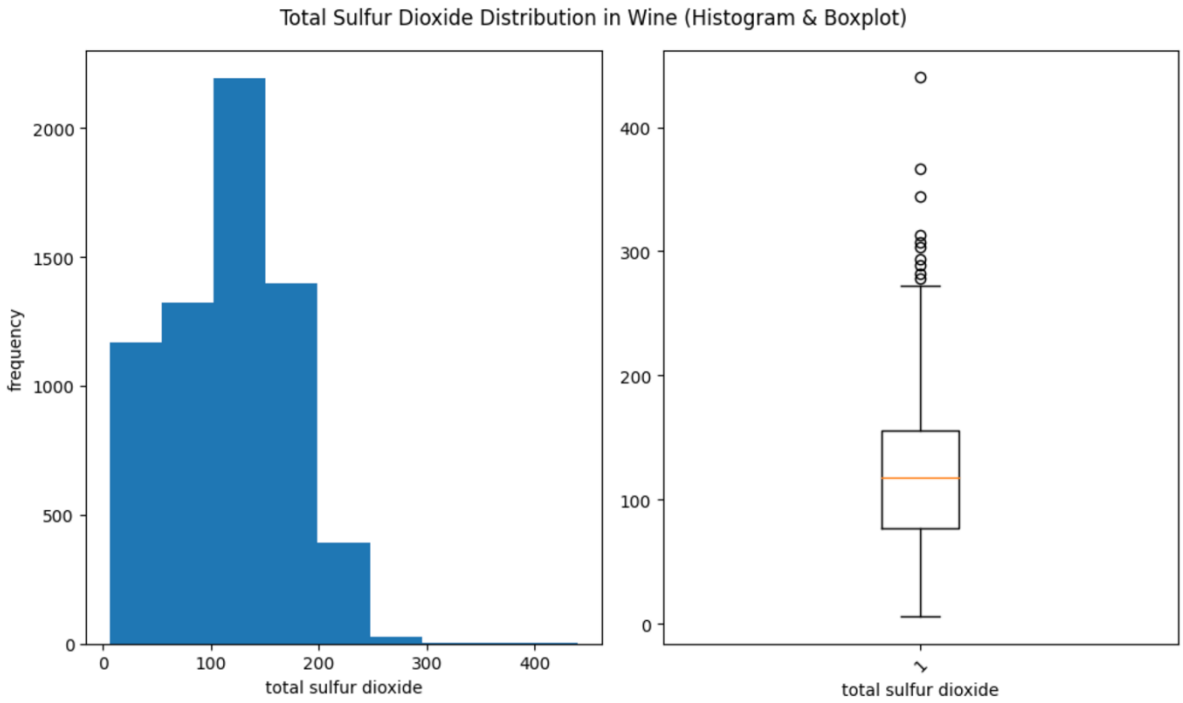


Figure 7:Total Sulfur Dioxide Distribution in Wine (Histogram & Boxplot)

Density Distribution in Wine (Histogram & Boxplot)

```
# density

x = 'density'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

ax1.hist(df[x], bins = 9)
ax1.set_xlabel(x)
ax1.set_ylabel('frequency')

ax2.boxplot(df[x])
ax2.set_xlabel(x)

plt.suptitle("Density Distribution in Wine (Histogram & Boxplot)")


plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

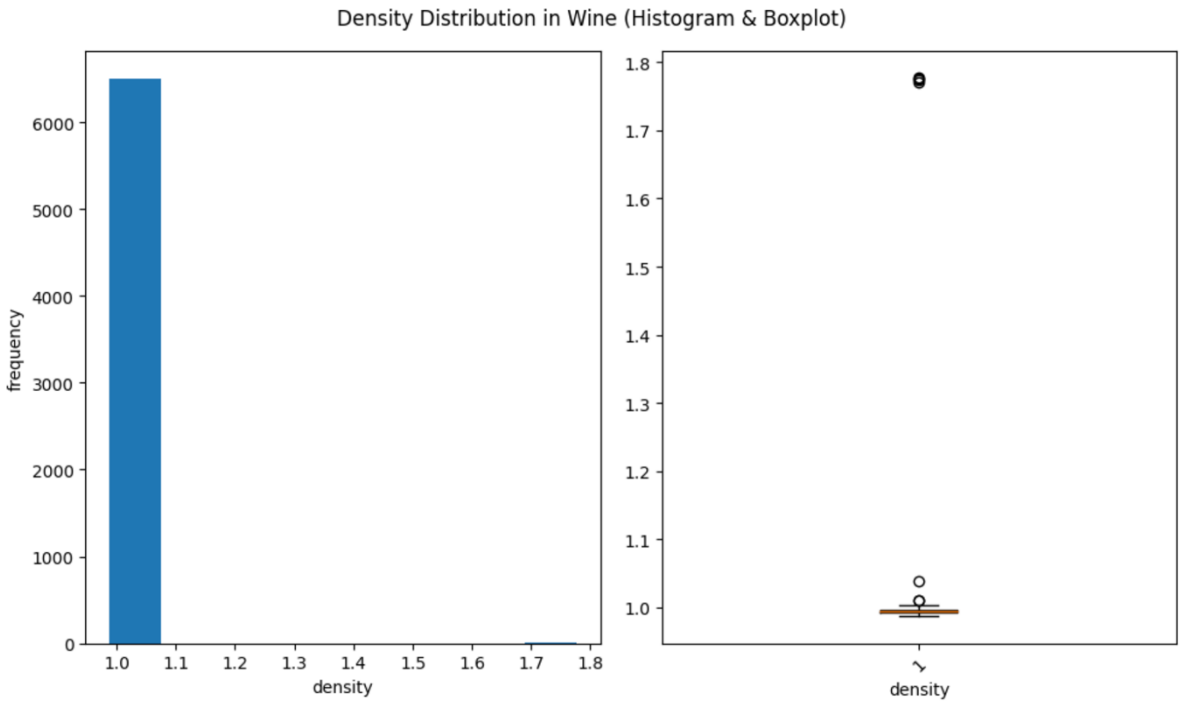


Figure 8: Density Distribution in Wine (Histogram & Boxplot)

pH Distribution in Wine (Histogram & Boxplot)

```
# pH

x = 'pH'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

ax1.hist(df[x], bins = 9)
ax1.set_xlabel(x)
ax1.set_ylabel('frequency')

ax2.boxplot(df[x])
ax2.set_xlabel(x)

plt.suptitle("pH Distribution in Wine (Histogram & Boxplot)")


plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

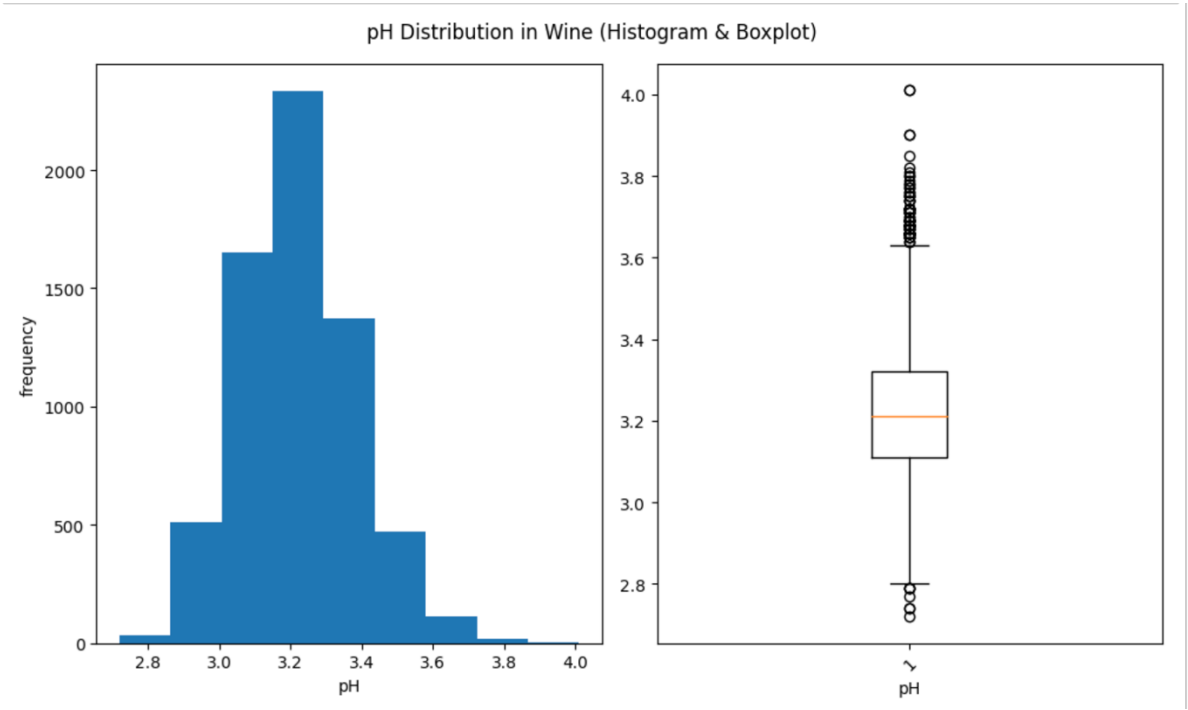


Figure 9: pH Distribution in Wine (Histogram & Boxplot)

Sulphates Distribution in Wine (Histogram & Boxplot)

```
# sulphates

x = 'sulphates'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

ax1.hist(df[x], bins = 9)
ax1.set_xlabel(x)
ax1.set_ylabel('frequency')

ax2.boxplot(df[x])
ax2.set_xlabel(x)

plt.suptitle("Sulphates Distribution in Wine (Histogram & Boxplot)")


plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

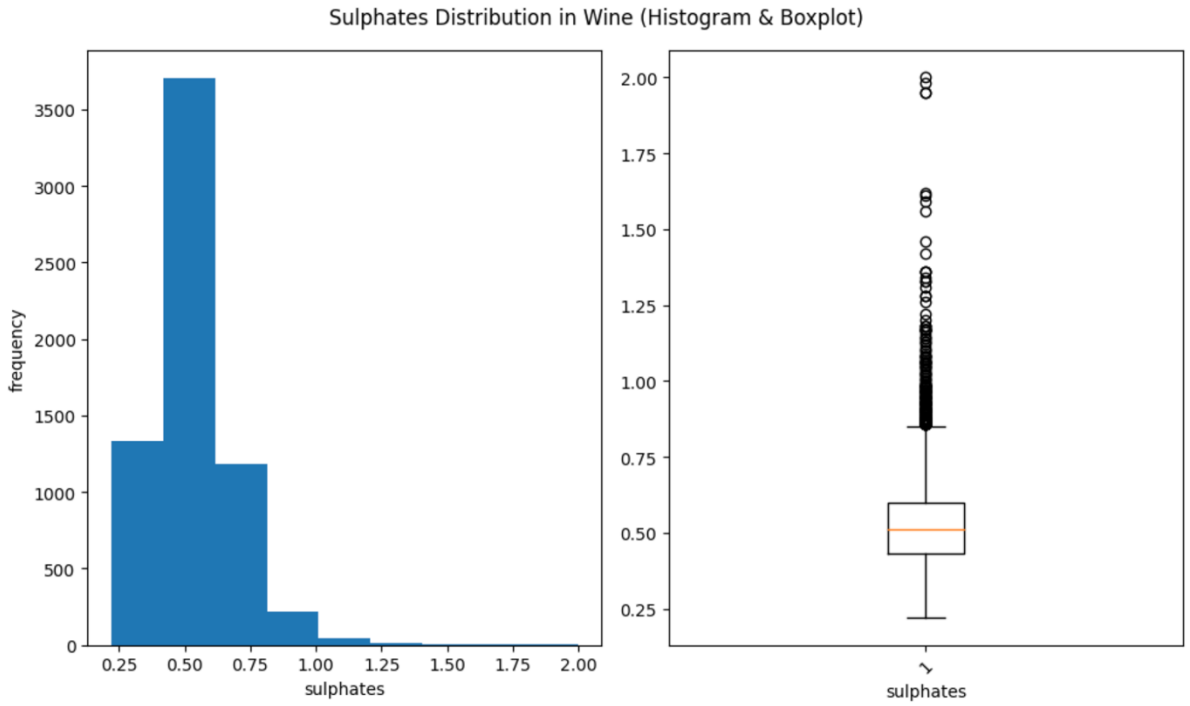


Figure 10: Sulphates Distribution in Wine (Histogram & Boxplot)

Alcohol Distribution in Wine (Histogram & Boxplot)

```
# alcohol

x = 'alcohol'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

ax1.hist(df[x], bins = 9)
ax1.set_xlabel(x)
ax1.set_ylabel('frequency')

ax2.boxplot(df[x])
ax2.set_xlabel(x)

plt.suptitle("Alcohol Distribution in Wine (Histogram & Boxplot)")


plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

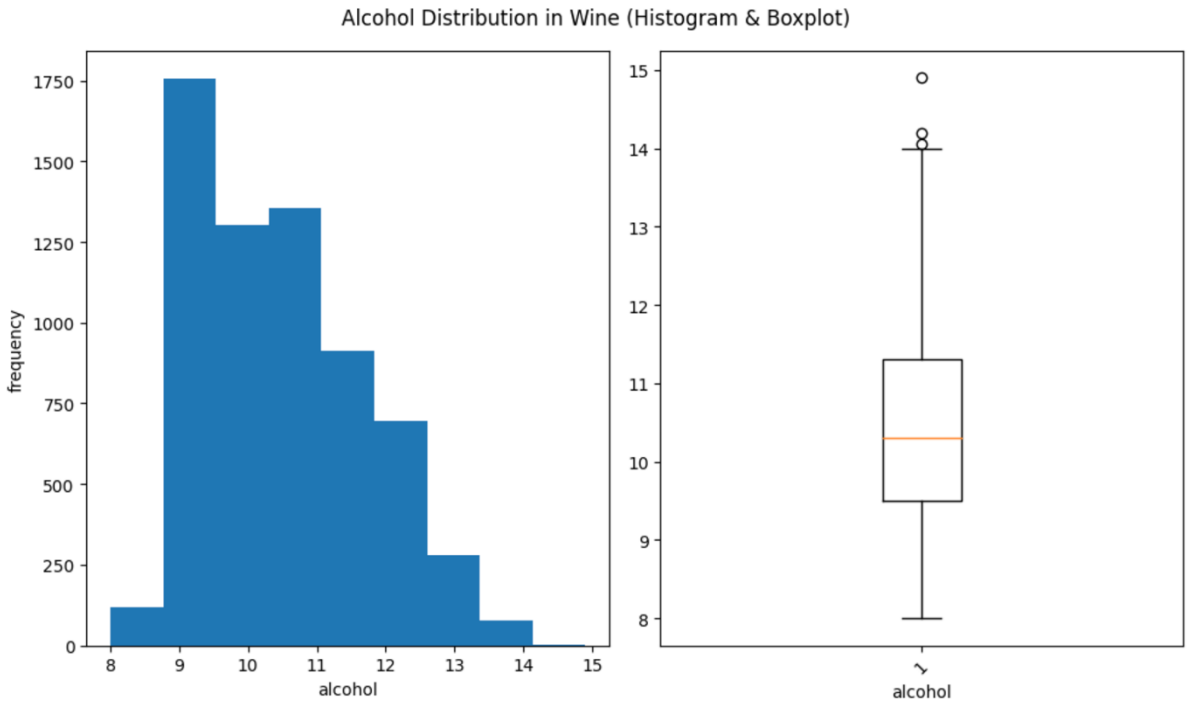


Figure 11: Alcohol Distribution in Wine (Histogram & Boxplot)

Percentage of the quality of wine

```
# quality

x = df.quality.value_counts().to_numpy()

plt.pie(x, labels = ['Bad', 'Good'], autopct = '%1.1f%%')


plt.title('Percentage of the quality of wine')
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

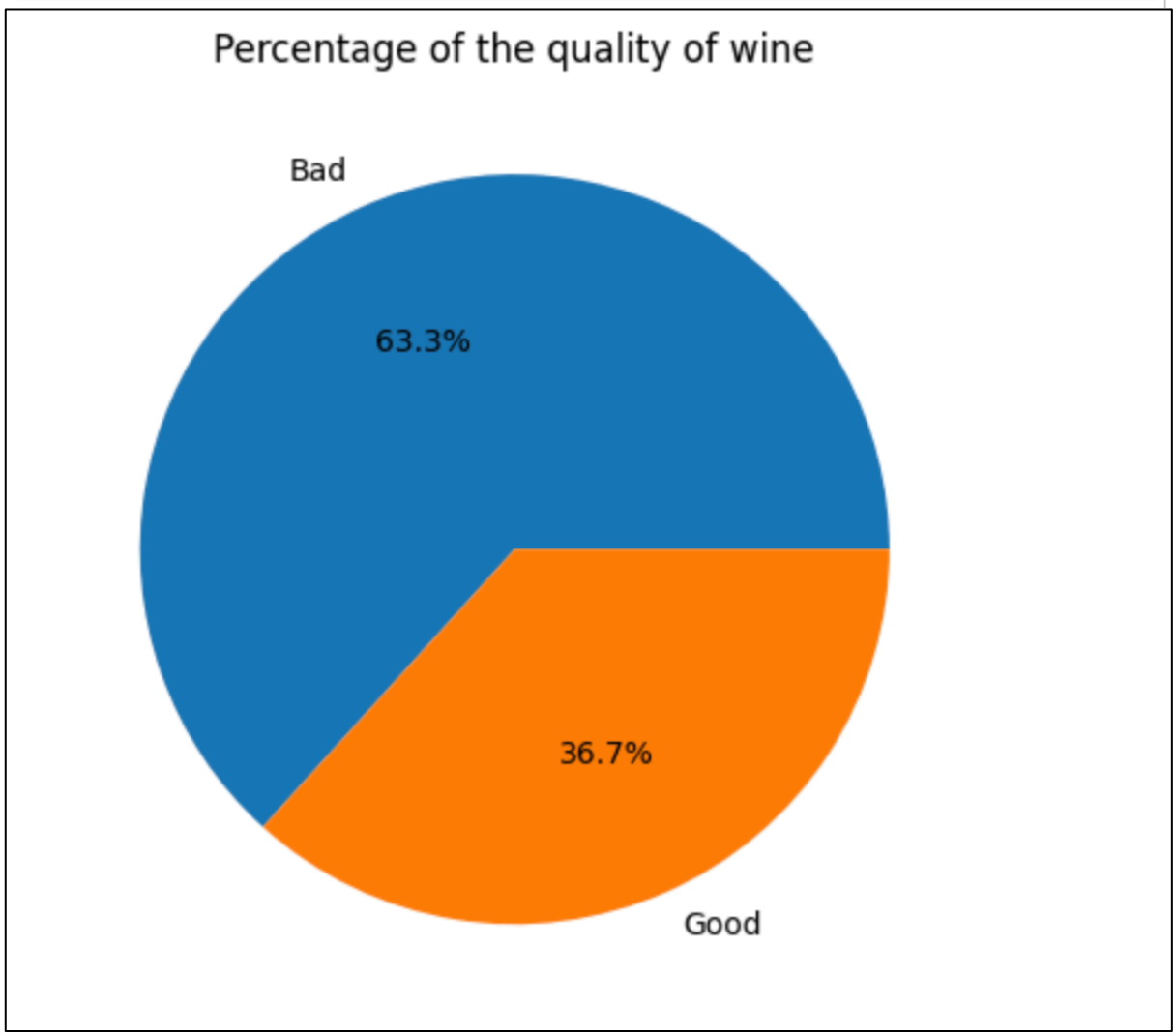


Figure 12: Percentage of the quality of wine

1.3.2 Summary Measures:

Using df.describe(), df.mean(), df.mode() commands the following table was made:

Table 2: Summary measures for different variables

Variables	Mean	Median	Mode	Standard Deviation
fixed acidity	7.216144	7	6.8	1.296323
volatile acidity	0.340552	0.29	0.28	0.167861
citric acid	0.318874	0.31	0.3	0.145785
residual sugar	5.440588	3	2	4.756935
chlorides	0.056551	0.047	0.044	0.040659
free sulfur dioxide	30.517398	29	29	17.745527
total sulfur dioxide	115.700495	118	111	56.523208
density	0.995176	0.99489	0.9972	0.019568
pH	3.218539	3.21	3.16	0.160774
sulphates	0.531744	0.51	0.5	0.149994
alcohol	10.491656	10.3	9.5	1.192292
quality	N/A	1	1	0.482098

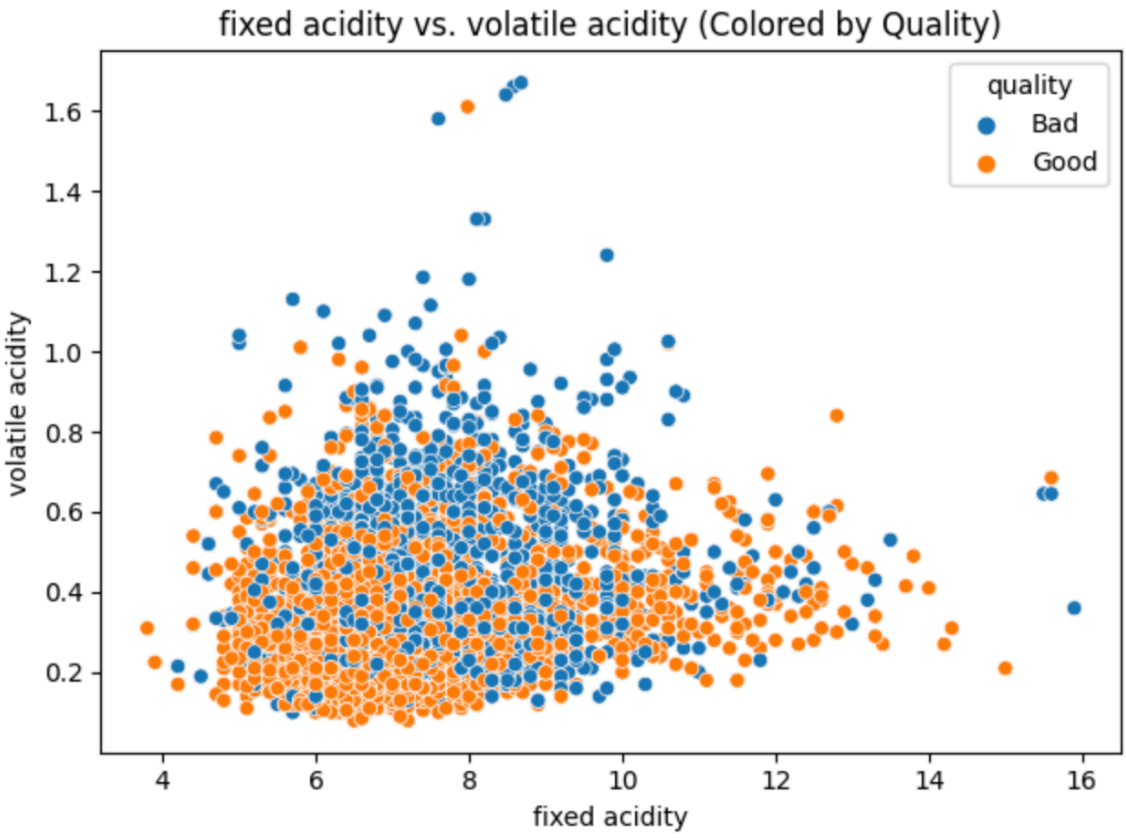
1.4 Bivariate Analysis:

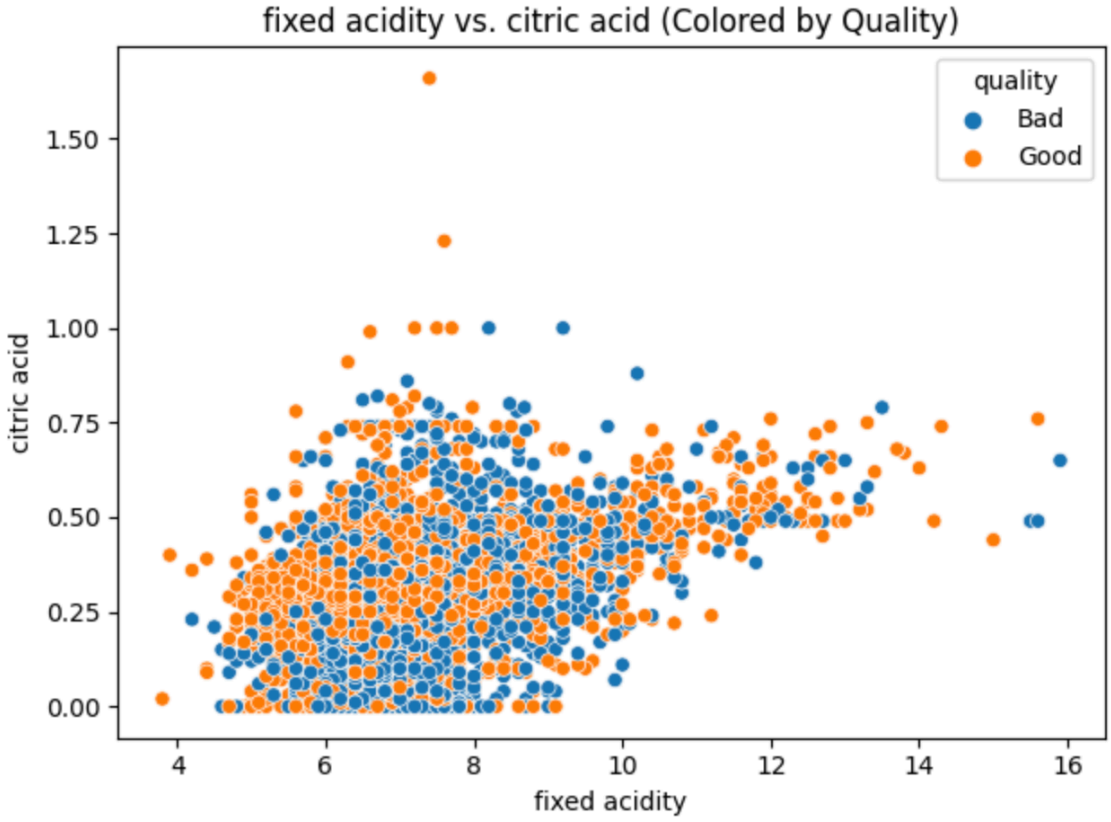
1.4.1 Graphical Representations:

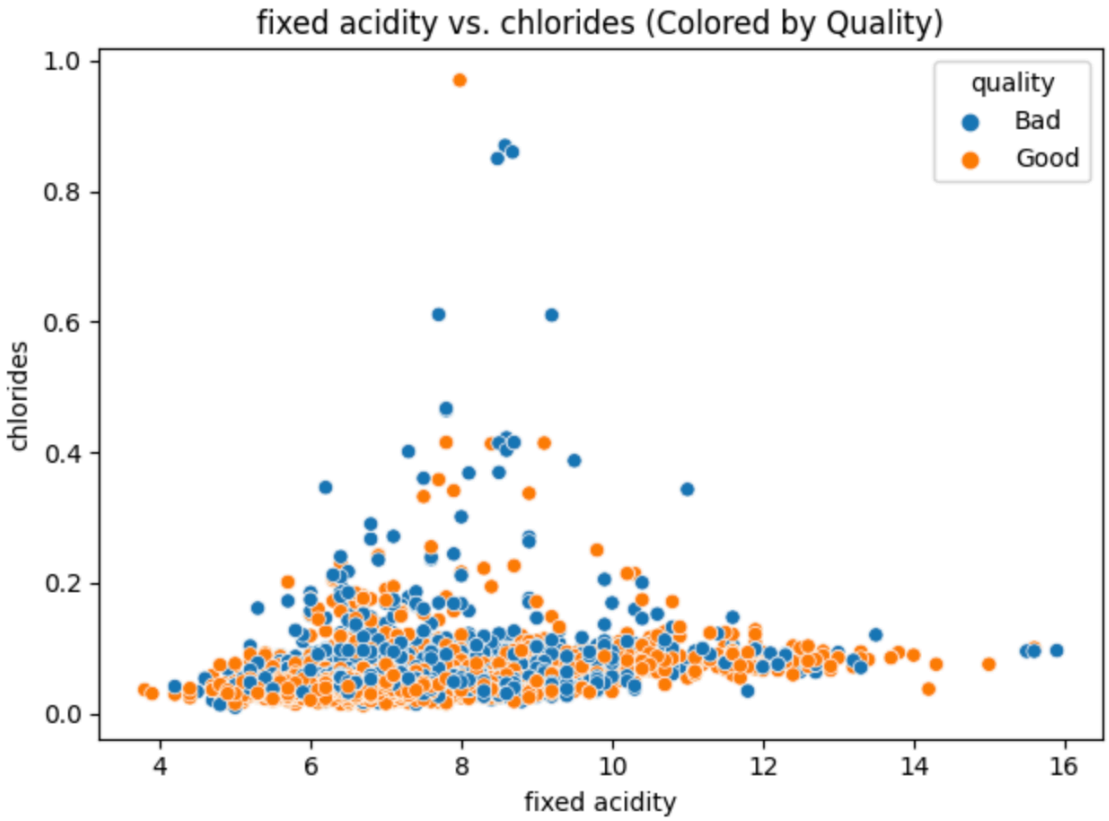
Scatter plots for all the variables color coded by Quality.

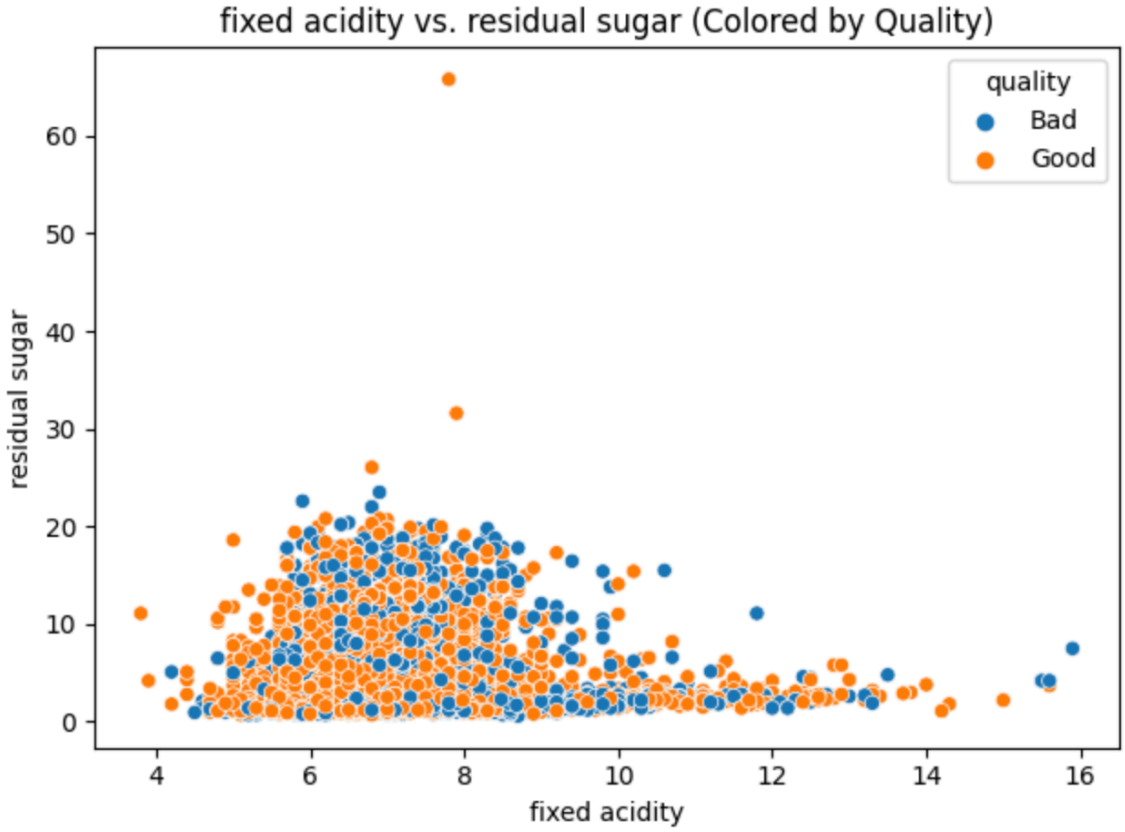
```
df['quality'] = df['quality'].map({0:'Bad', 1: 'Good'})
for col1 in df.columns:
  for col2 in df.columns[df.columns.get_loc(col1) + 1:]:
    sns.scatterplot(x=col1, y=col2, hue='quality', data=df)
    plt.title(f"{col1} vs. {col2} (Colored by Quality)")
    plt.tight_layout()
    plt.show()
```

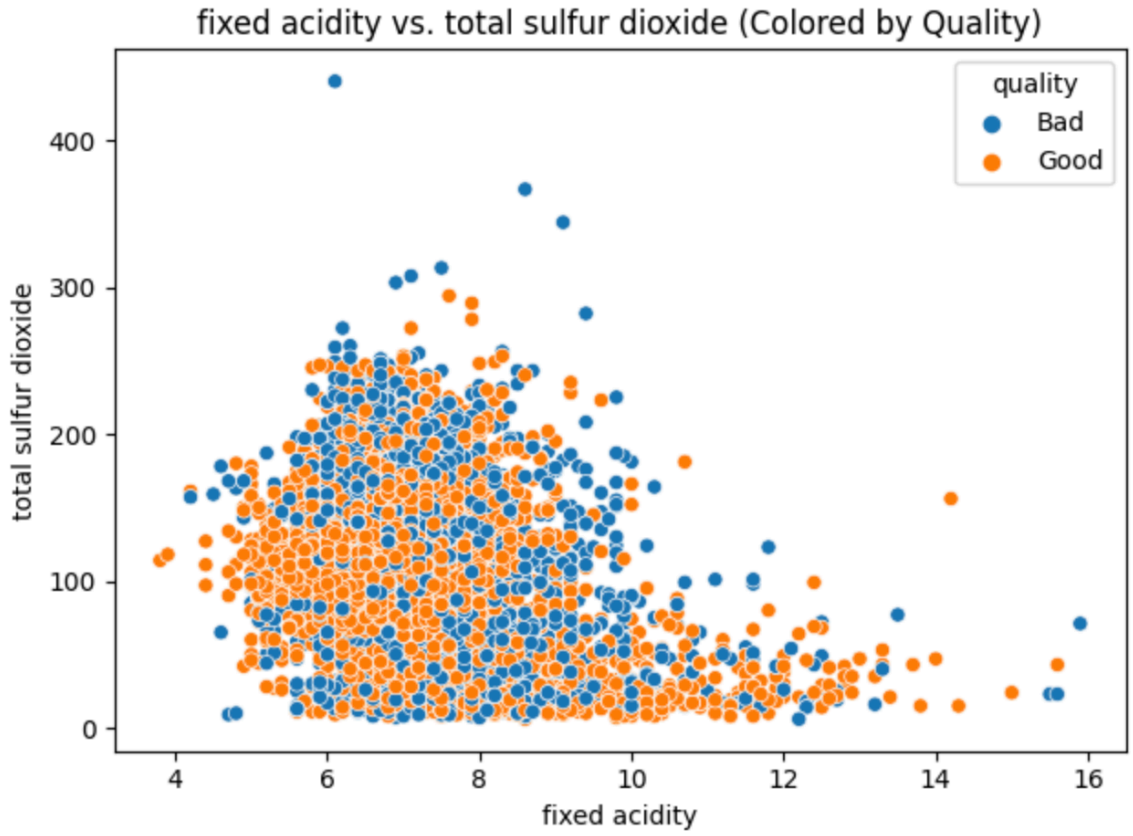
There are about 65 plots for this code, Scatter plots for all the variables with each other. However, here only the scatter plots with fixed acidity to all the other variables are chosen for this assignment, to make the assignment concise.

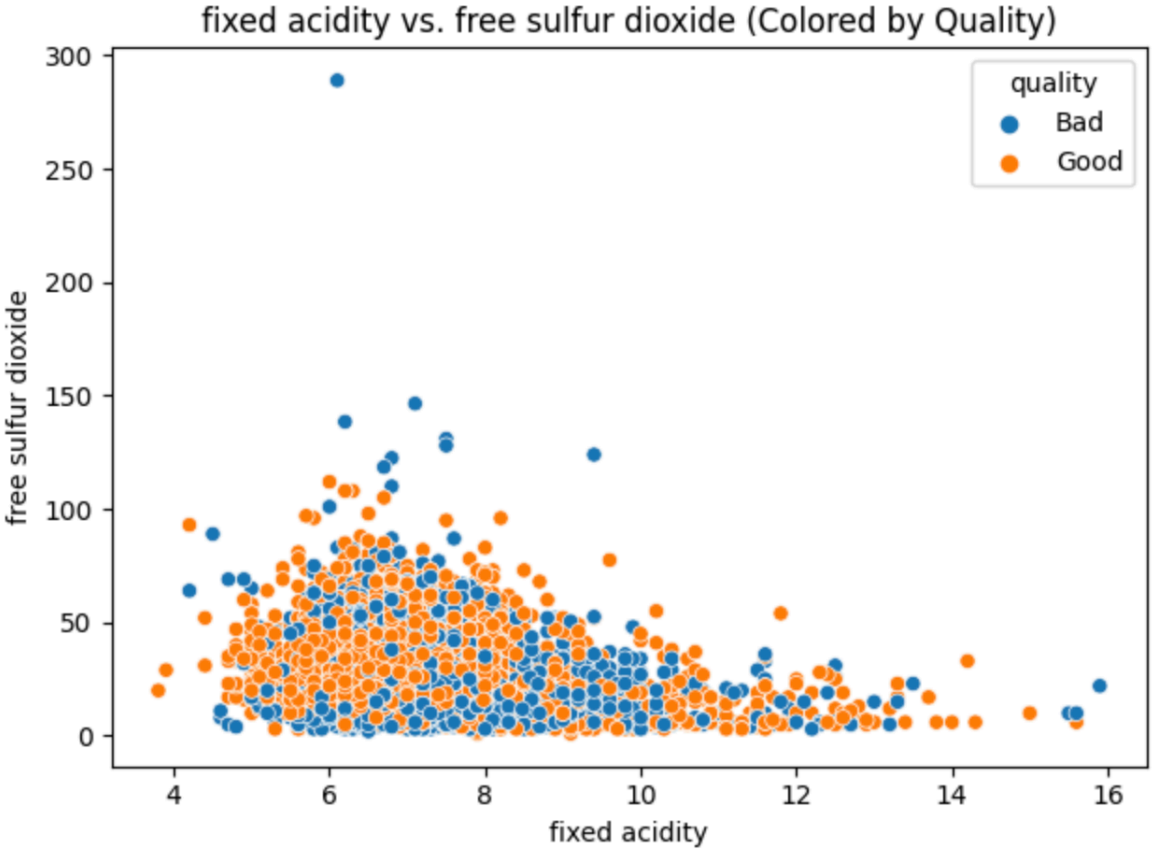

fixed acidity vs. volatile acidity (Colored by Quality)
quality
Bad
Good
volatile acidity
fixed acidity
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
4
6
8
10
12
14
16

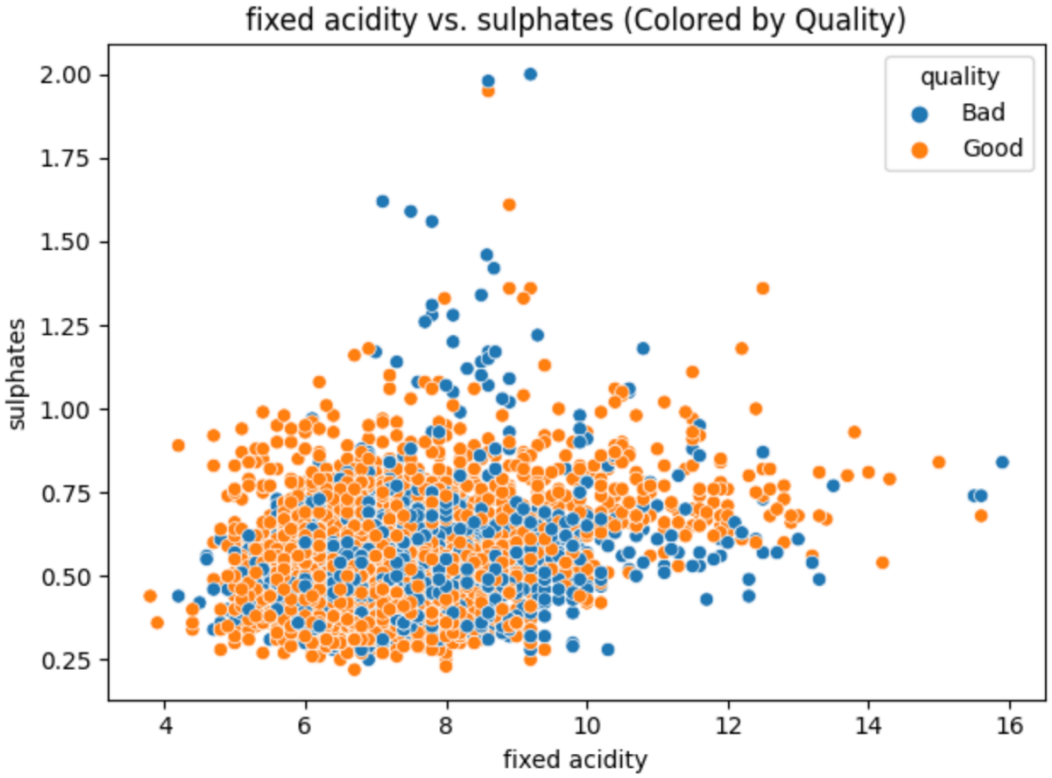

fixed acidity vs. citric acid (Colored by Quality)
quality
Bad
Good
citric acid
fixed acidity
1.50
1.25
1.00
0.75
0.50
0.25
0.00
4
6
8
10
12
14
16

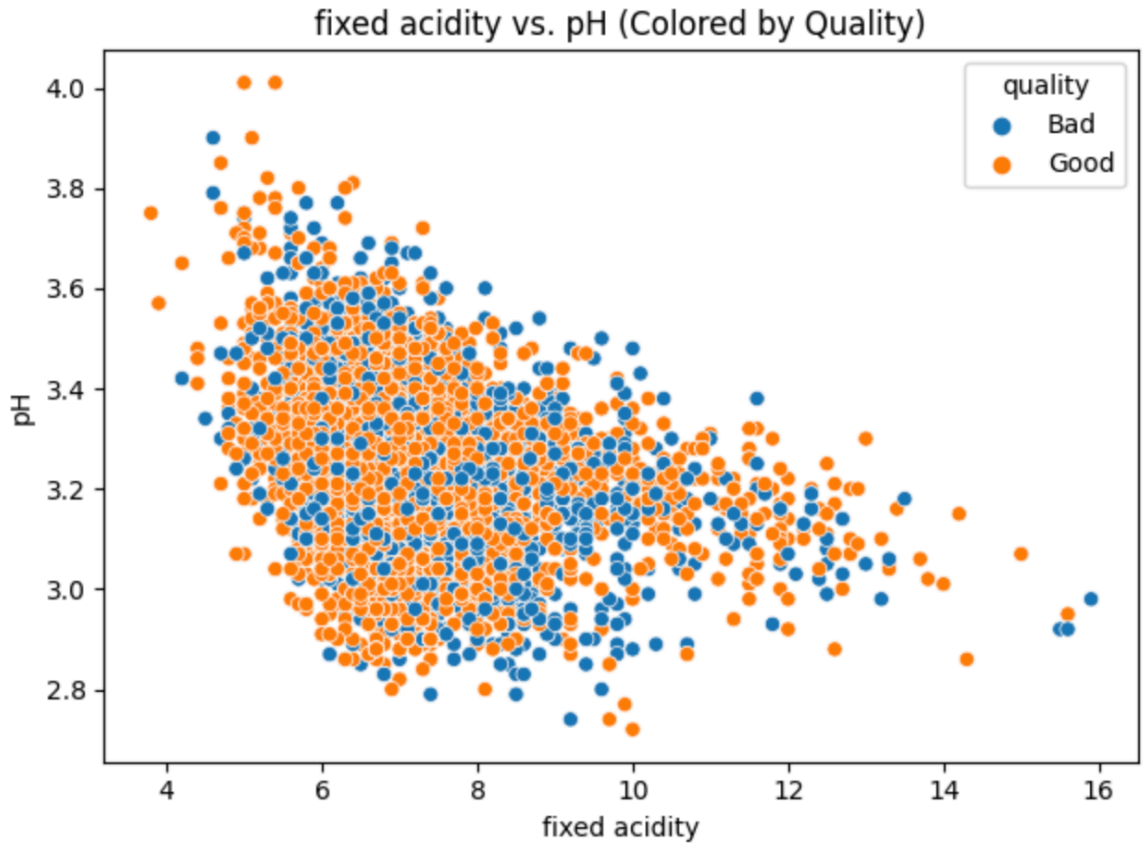

fixed acidity vs. chlorides (Colored by Quality)
quality
Bad
Good
chlorides
fixed acidity
0.0
0.2
0.4
0.6
0.8
1.0
4
6
8
10
12
14
16

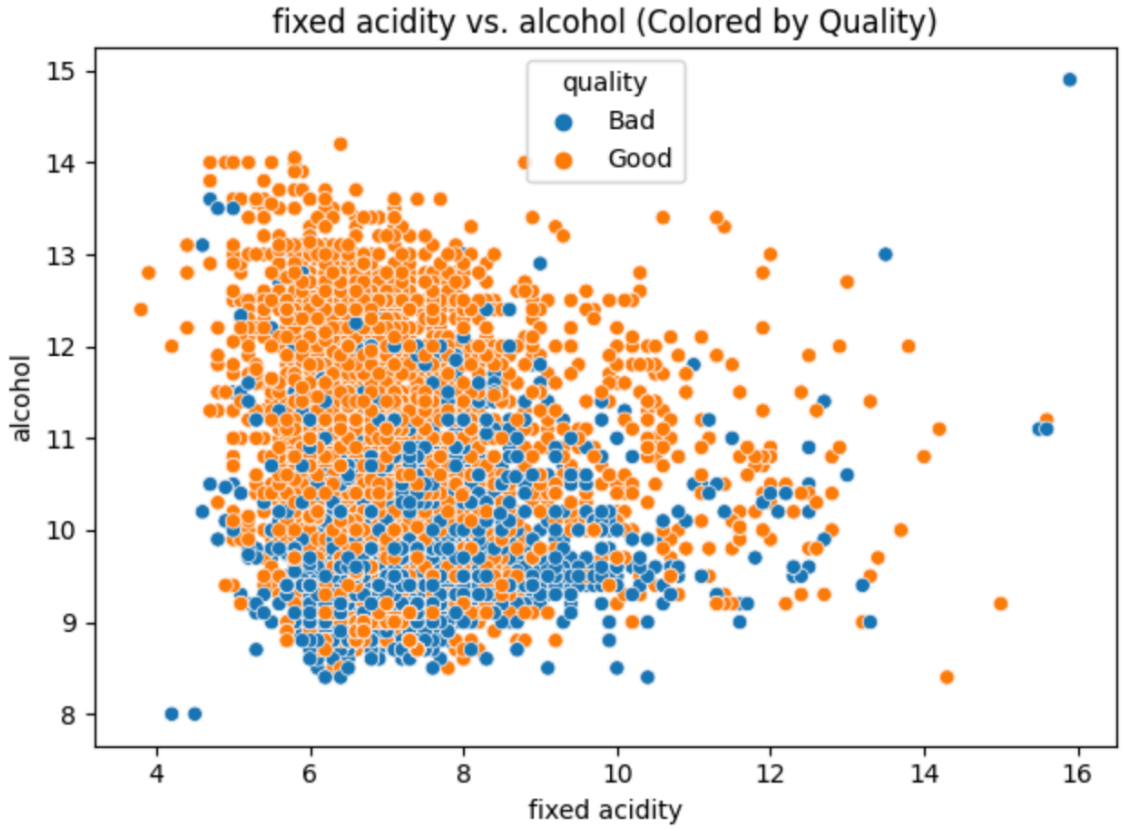

fixed acidity vs. residual sugar (Colored by Quality)
quality
Bad
Good
residual sugar
fixed acidity
0
10
20
30
40
50
60
4
6
8
10
12
14
16

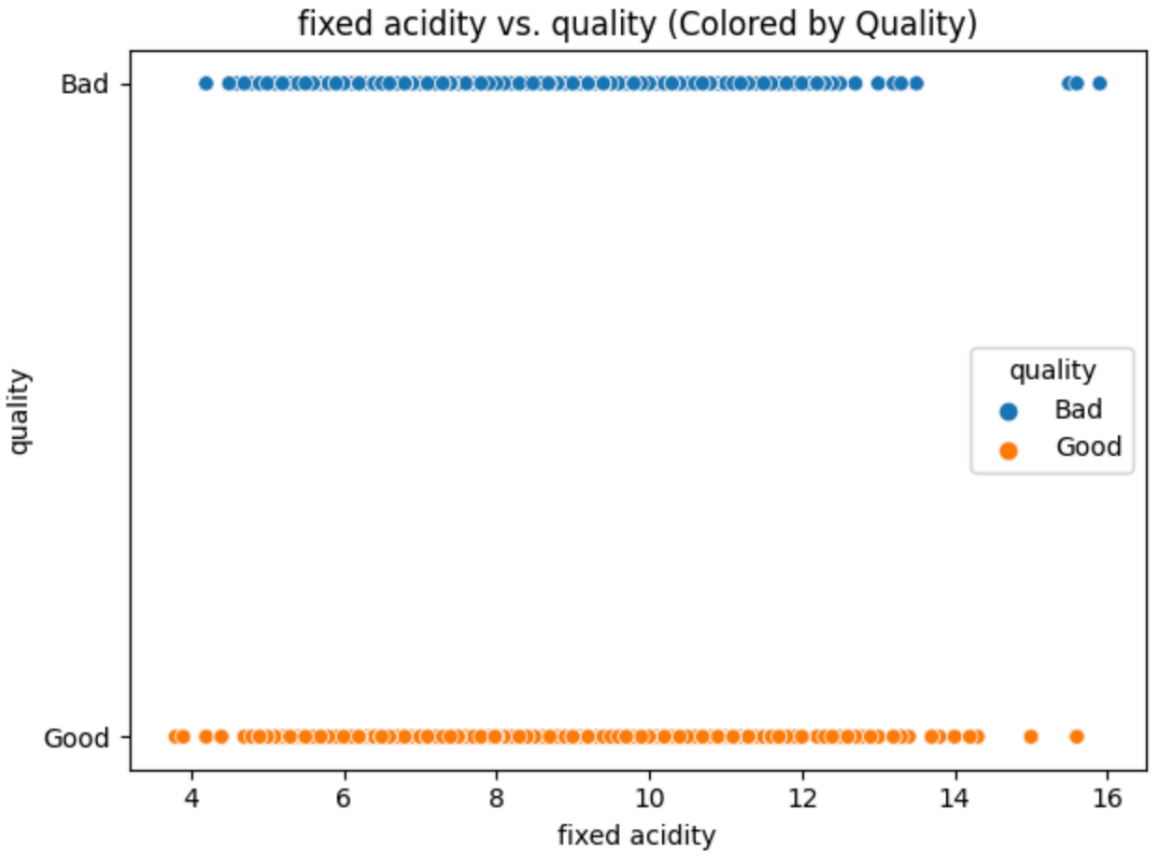

fixed acidity vs. total sulfur dioxide (Colored by Quality)
quality
Bad
Good
total sulfur dioxide
fixed acidity
0
100
200
300
400
4
6
8
10
12
14
16


fixed acidity vs. free sulfur dioxide (Colored by Quality)
quality
Bad
Good
free sulfur dioxide
fixed acidity
0
50
100
150
200
250
300
4
6
8
10
12
14
16


fixed acidity vs. sulphates (Colored by Quality)
quality
Bad
Good
sulphates
fixed acidity
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
4
6
8
10
12
14
16


fixed acidity vs. pH (Colored by Quality)
quality
Bad
Good
pH
fixed acidity
4.0
3.8
3.6
3.4
3.2
3.0
2.8
4
6
8
10
12
14
16


fixed acidity vs. alcohol (Colored by Quality)
quality
Bad
Good
alcohol
fixed acidity
15
14
13
12
11
10
9
8
4
6
8
10
12
14
16


fixed acidity vs. quality (Colored by Quality)
quality
Bad
Good
fixed acidity
Bad
Good
4
6
8
10
12
14
16

1.4.2 Summary Measures:

Using this df.corr() command making the heatmap:

```
correlation = df.corr()
plt.figure(figsize=(10, 6))
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```

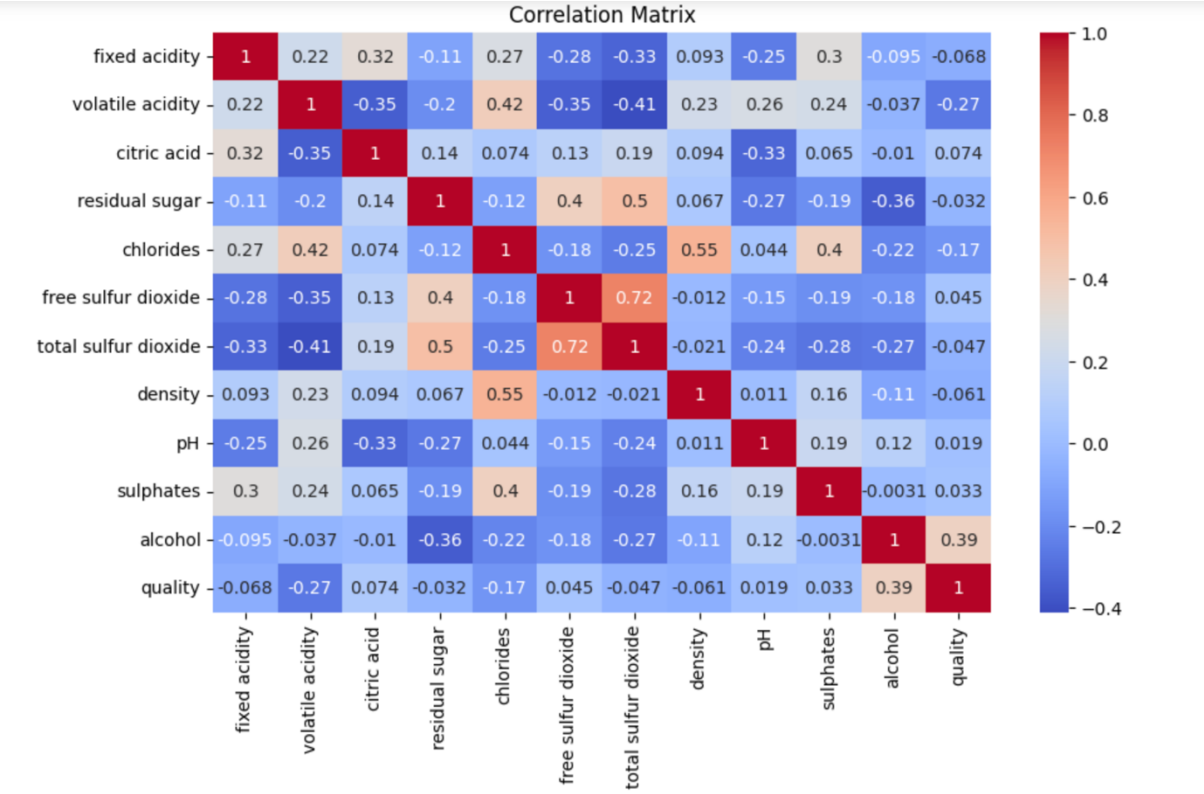


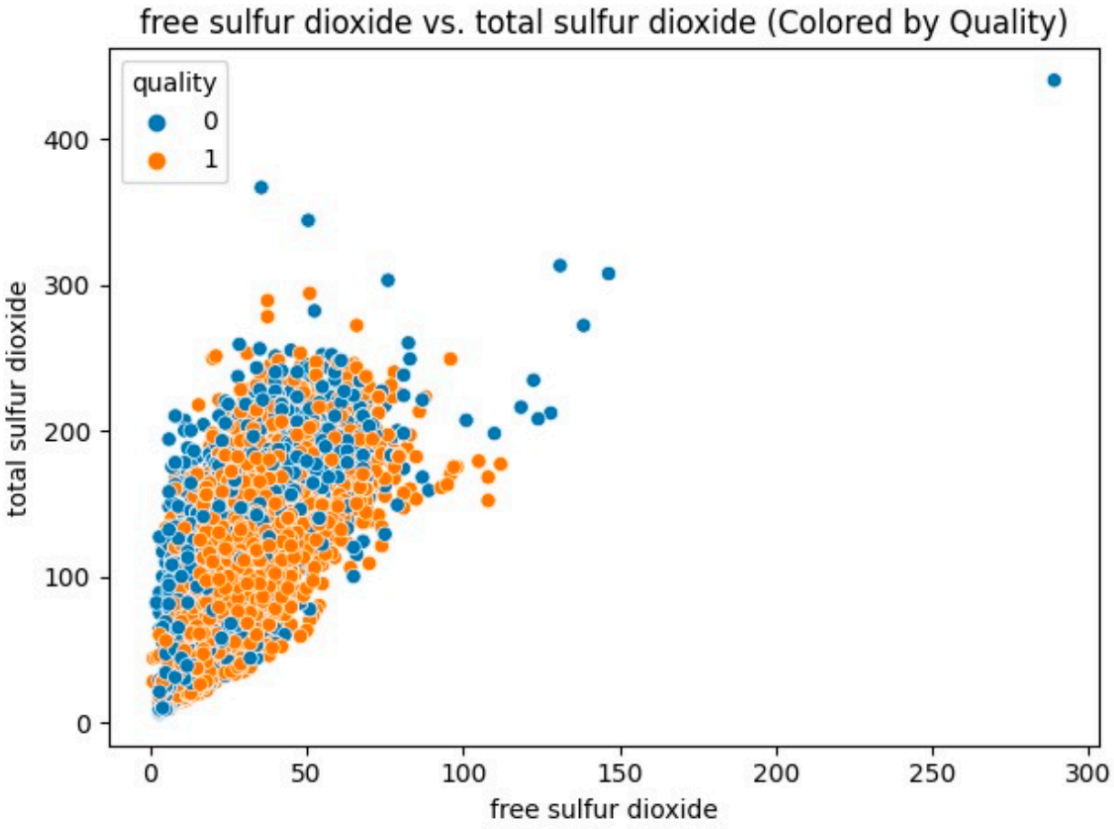
Figure 13: heatmap of Correlation Matrix

There are some strong linear relation between the variables:

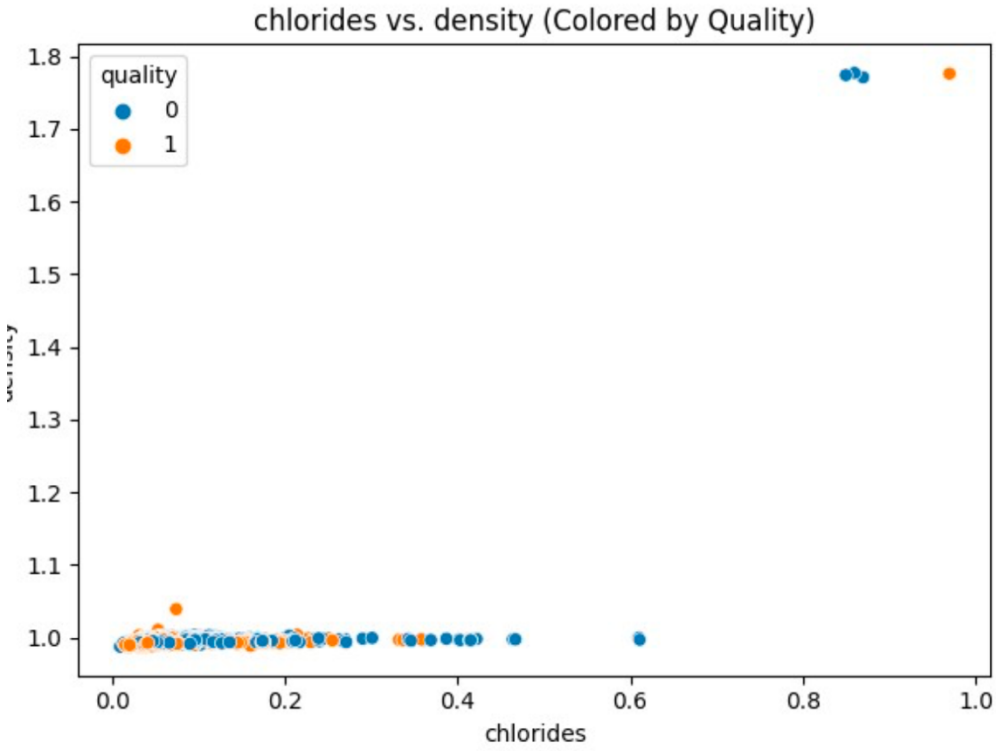
- Free sulfur dioxide to total sulfur dioxide : 0.72; stong positive
- Density to chlorides : 0.55, moderate positive
- Residual sugar to total sulfur dioxide : 0.5; moderate positive
- Volatile acidity to chlorides, free sulfur dioxide to residual sugar and suphates to chlorides have almost moderate positive linear relation.
- Alternatively, volatile acidity to total sulfur dioxide(-0.41) moderate negative linear relationship.

Some of the strong positive, moderate positive and moderate negative linear relation is shown below:

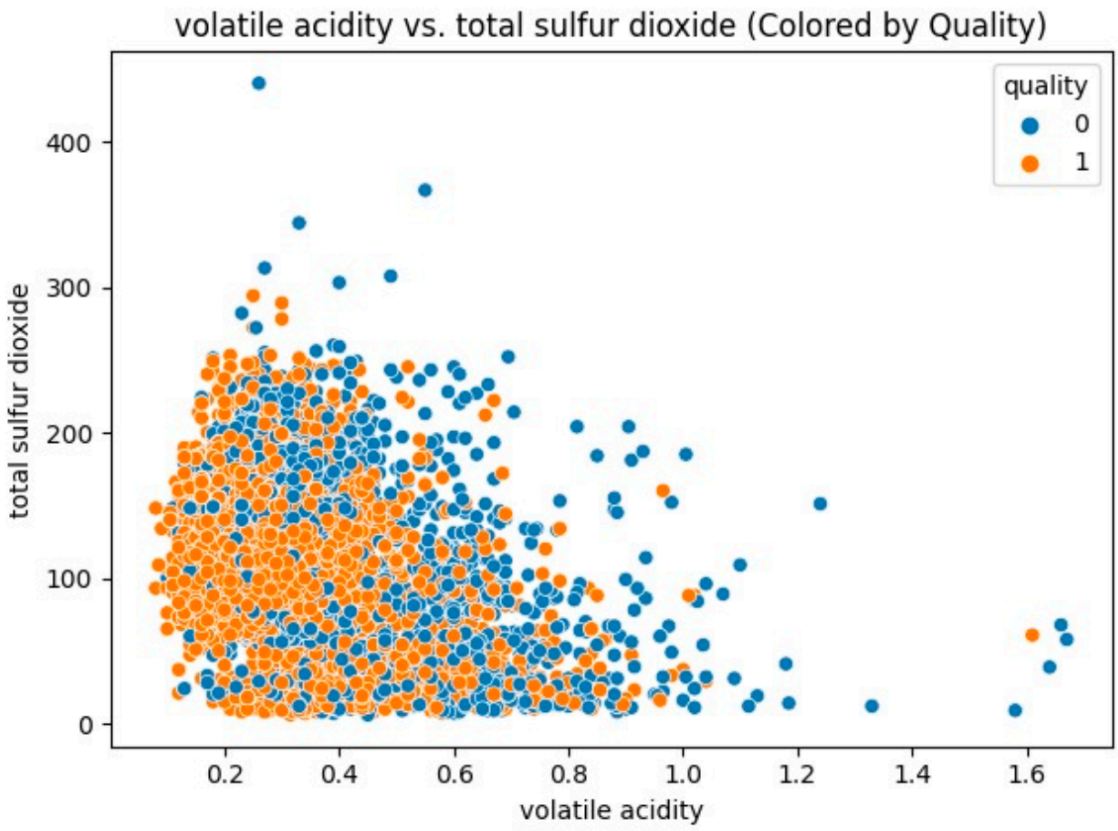
Free sulfur dioxide vs total sulfur dioxide: strong positive relation(0.72)



Density to chlorides: Moderate positive linear relation(0.55)

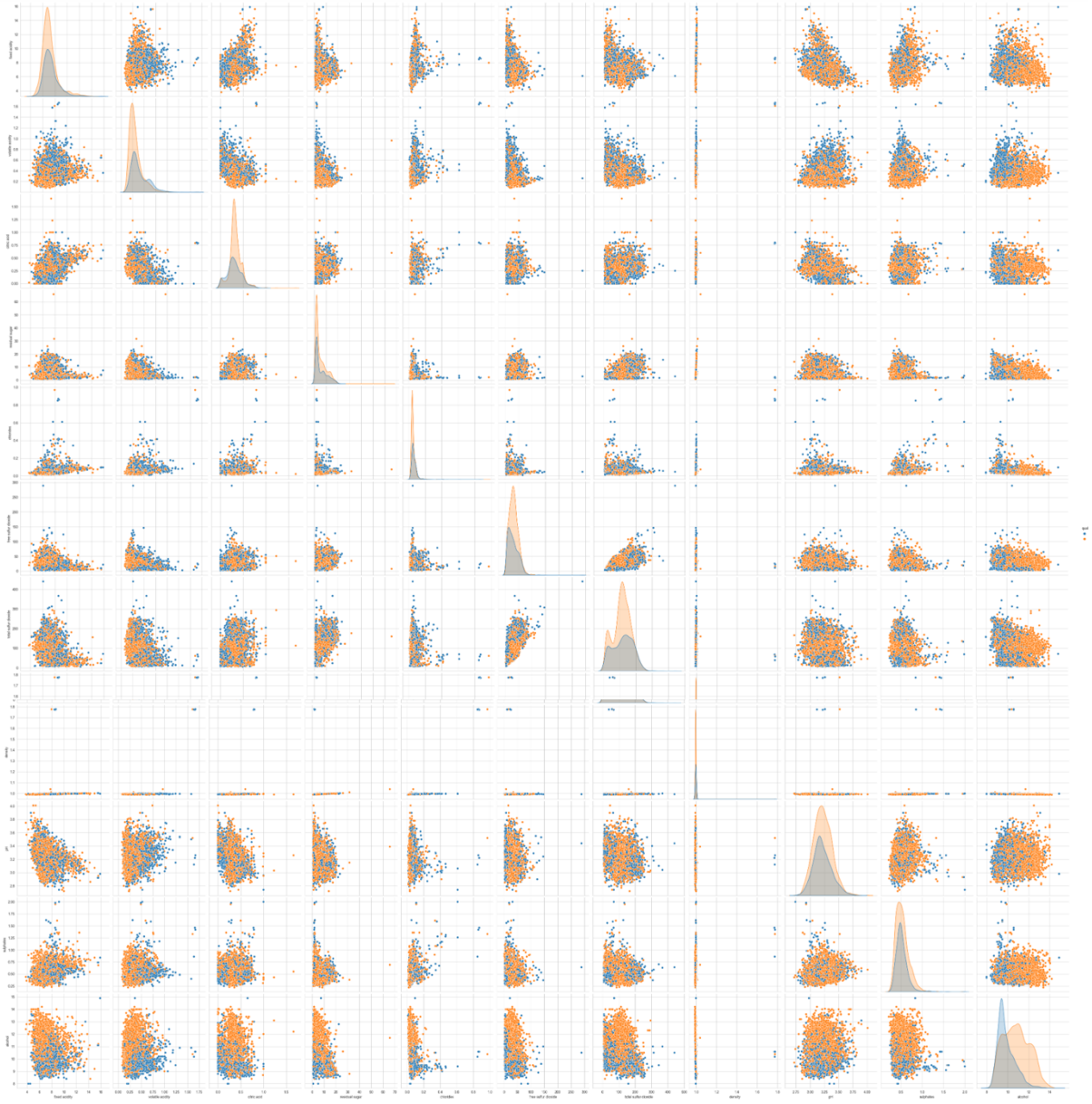


volatile acidity to total sulfur dioxide: moderate negative linear relationship(-0.41)


volatile acidity vs. total sulfur dioxide (Colored by Quality)
quality
0
1
total sulfur dioxide
volatile acidity
0
100
200
300
400
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

Pairplots:

```
#####Pairplots
sns.set_style('whitegrid');
sns.pairplot(df, hue = 'quality', height = 4)
plt.show()
```



Problem 2:

For the problem 2 and 3, the following libraries were imported:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, classification_report, accuracy_score

from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

The code for logistical model:

```
#2 logistic regression using sklearn

X = df.drop('quality',axis = 1) #independent
y = df['quality'] #target/dependent variable


#logistic model
logit = LogisticRegression()
logit.fit(X, y)

print('Coefficients:\n', logit.coef_)
print('Intercept:', logit.intercept_)
```

```
Coefficients:
 [[-0.17247506 -4.11639612  0.33790097  0.03862691 -0.34710538  0.0182845
  -0.01052447 -1.01700789 -0.97215352  1.8427718   0.75905165]]
Intercept: [-1.06225756]
```

The Logistic Regression Equation:

$$Logit\left(\frac{P(Good)}{1-P(Good)}\right) = -1.06225756 - 0.17247506X1 - 4.11639612X2 + 0.33790097X3 + 0.03862691X4 - 0.34710538X5 + 0.0182845X6 - 0.01052447X7 - 1.01700789X8 - 0.97215352X9 + 1.8427718X10 + 0.75905165X11$$

Interpretation of Intercept:

Variable Name	Exp (Variable)	Interpretation
Intercept	0.34567455	When all the variables (X) are zero, the odds of being Good Wine Quality is 0.34567455

Variable	Exp(coefficient)	Interpretation
fixed acidity	0.84157928	Increase in Fixed Acidity is Less Likely to be Good Quality Wine.
volatile acidity	0.01630316	Increase in Density is Least Likely to be Good Quality Wine.
citric acid	1.40200166	Increase in Citric Acid is More Likely to be Good Quality Wine.
residual sugar	1.03938263	Increase in Residual Sugar is More Likely to be Good Quality Wine.
chlorides	0.70673085	Increase in Chlorides is Less Likely to be Good Quality Wine.
free sulfur dioxide	1.01845268	Increase in Free Sulfur Dioxide is More Likely to be Good Quality Wine.
total sulfur dioxide	0.98953072	Increase in Total Sulfur Dioxide is Less Likely to be Good Quality Wine.

density	0.3616755	Increase in Density is Less Likely to be Good Quality Wine.
pH	0.37826755	Increase in pH is Less Likely to be Good Quality Wine.
sulphates	6.31401521	Increase in Sulphates is Most Likely to be Good Quality Wine.
alcohol	2.13624936	Increase in Alcohol is More Likely to be Good Quality Wine.

Problem 3

For the problem 2 and 3, the following libraries were imported:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, classification_report, accuracy_score

from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

Logistic Regressions Model:

```
#3(i)
logmodel = LogisticRegression()
logmodel.fit(X_train, y_train)

y_pred = logmodel.predict(X_test)
roc_auc = roc_auc_score(y_test, model.decision_function(X_test))
print("ROC AUC Score:", roc_auc)

print("Logistic Regression Model")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("ROC AUC Score", roc_auc_score(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
tn,fp, fn, tp = cm.ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
precision = tp/(tp+fp)

print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("Precision:", precision)
```

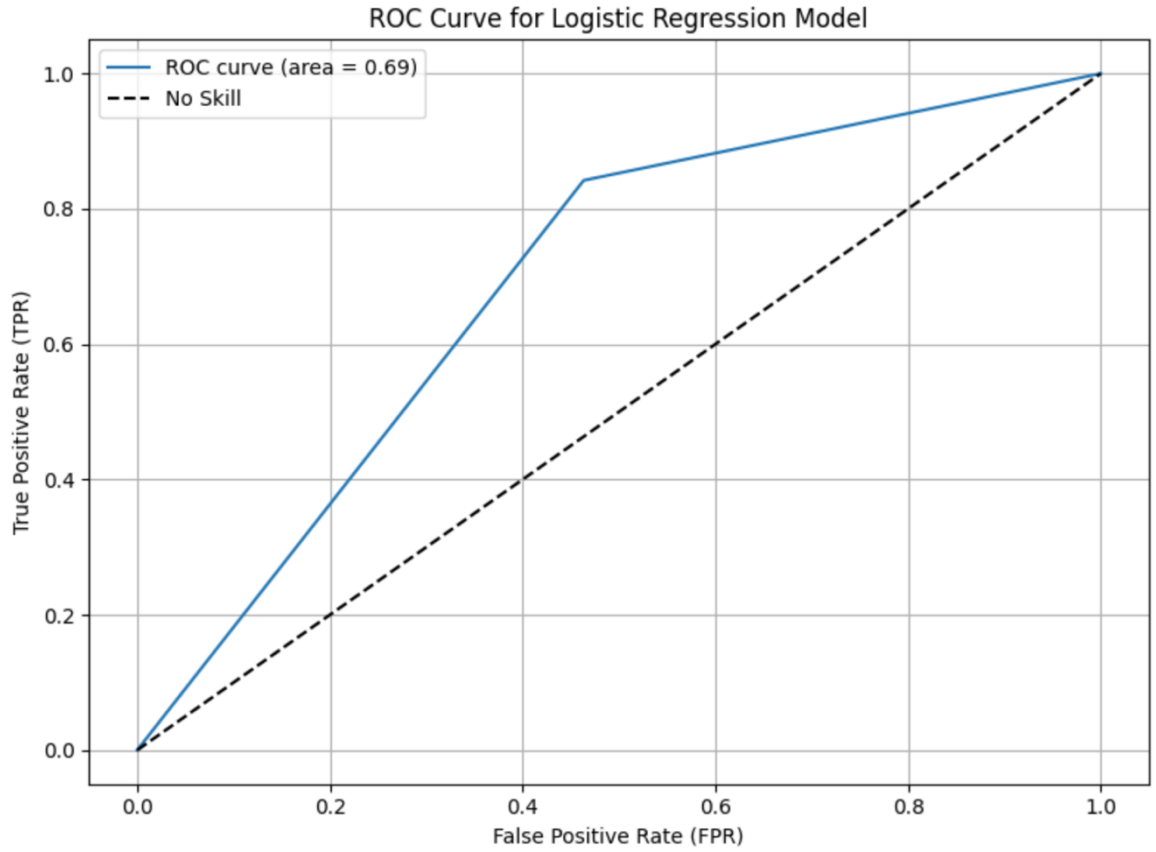
Output:

```
ROC AUC Score: 0.7915799389406544
Logistic Regression Model
Confusion Matrix:
 [[ 381  329]
 [ 196 1045]]
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.54      0.59       710
           1       0.76      0.84      0.80      1241

    accuracy                           0.73      1951
   macro avg       0.71      0.69      0.70      1951
weighted avg       0.72      0.73      0.72      1951

Accuracy Score: 0.7309072270630446
ROC AUC Score 0.6893412854240674
Sensitivity: 0.8420628525382756
Specificity: 0.5366197183098591
Precision: 0.7605531295487628
```

Accuracy: 73.09% Predictions are Correct
Precision: 76.05% Positive Predictions are Correct
Sensitivity: 84.21% Positive Cases were Predicted as Positive
Specificity: 53.66% Negative Cases were Predicted as Negative

ROC Curve:

```
#ROC Curve:
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--', label='No Skill')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Logistic Regression Model')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Decision Trees Model:

```
#3(ii)
DTclf=tree.DecisionTreeClassifier()

DTclf.fit(X_train, y_train)
y_pred= DTclf.predict(X_test) ##y_test= Y, y_pred=Y^

print("Decision Trees ")

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))

#Sensitivity, Specificity
cm = confusion_matrix(y_test, y_pred)
tn,fp, fn, tp = cm.ravel()

sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
precision = tp/(tp+fp)
print("Precision:", precision)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
```

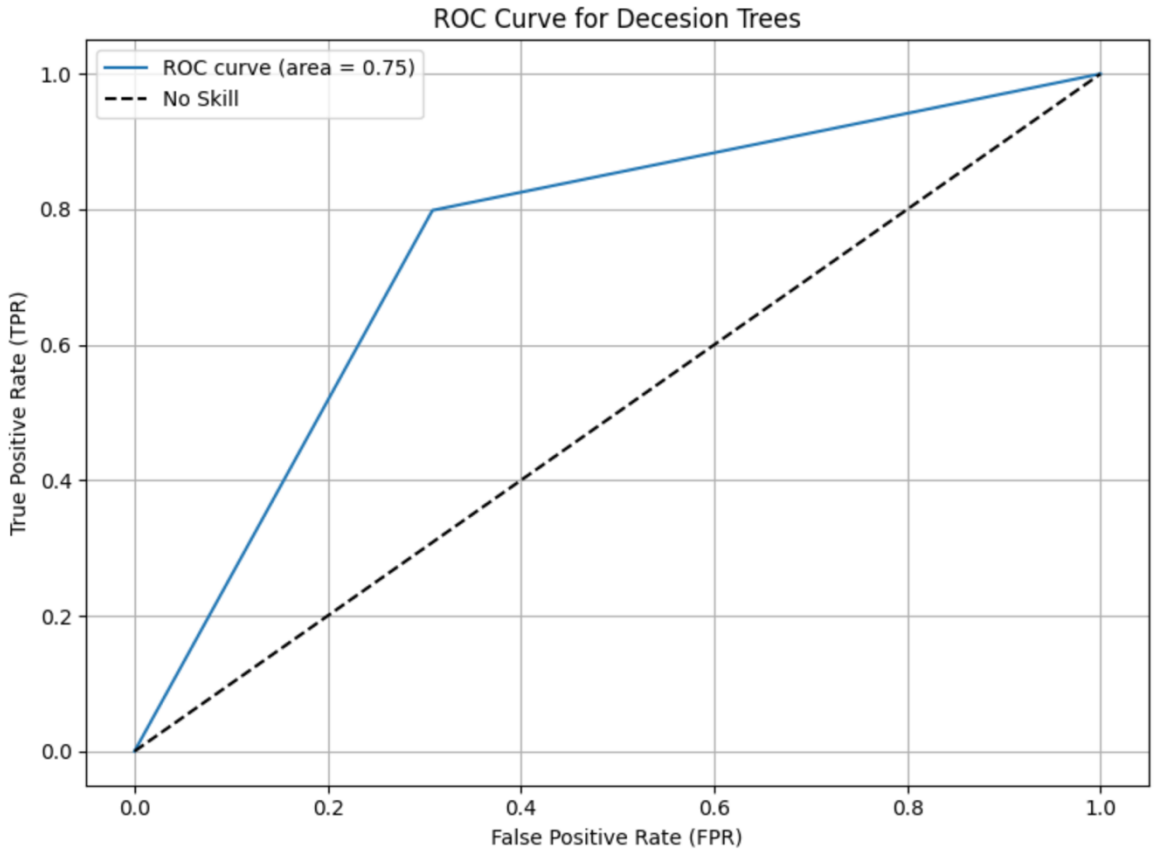
Output:

```
Decision Trees
Confusion Matrix:
 [[491 219]
 [250 991]]
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.69      0.68       710
           1       0.82      0.80      0.81      1241

    accuracy                           0.76      1951
   macro avg       0.74      0.75      0.74      1951
weighted avg       0.76      0.76      0.76      1951

Accuracy Score: 0.7596104561763198
Precision: 0.8190082644628099
Sensitivity: 0.798549556809025
Specificity: 0.6915492957746479
```

Accuracy: 75.96% Predictions are Correct
Precision: 81.90% Positive Predictions are Correct
Sensitivity: 79.85% Positive Cases were Predicted as Positive
Specificity: 89.15% Negative Cases were Predicted as Negative

ROC Curve:

```
#ROC Curve:

fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--', label='No Skill')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Decesion Trees')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Random Forrest Model:

```
#3(iii)
RFclf=RandomForestClassifier(n_estimators=11)

RFclf.fit(X_train, y_train)
y_pred=RFclf.predict(X_test)

print("Random Forest Model")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
tn,fp, fn, tp = cm.ravel()

sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
precision = tp/(tp+fp)
print("Precision:", precision)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
```

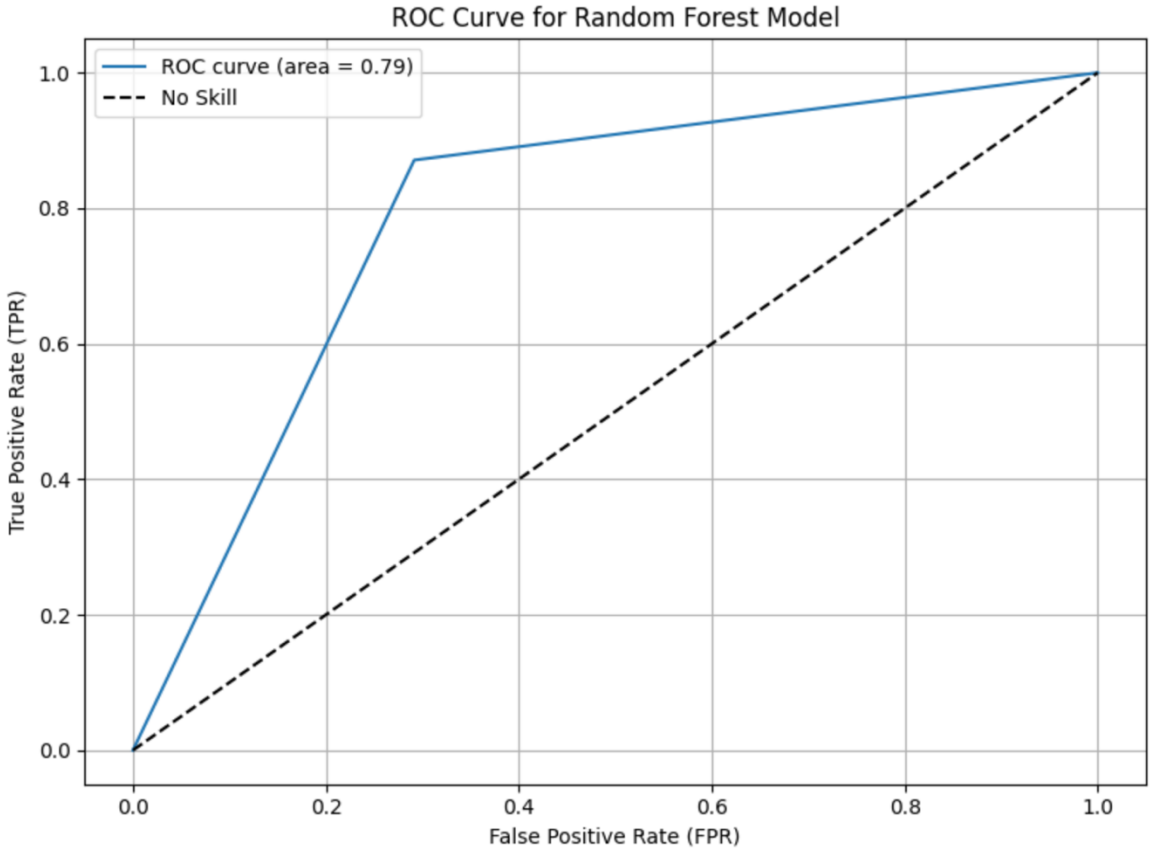
Output:

```
Random Forest Model
Confusion Matrix:
 [[ 503  207]
 [ 160 1081]]
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.71      0.73       710
           1       0.84      0.87      0.85      1241

    accuracy                           0.81      1951
   macro avg       0.80      0.79      0.79      1951
weighted avg       0.81      0.81      0.81      1951

Accuracy Score: 0.8118913377754997
Precision: 0.8392857142857143
Sensitivity: 0.871071716357776
Specificity: 0.7084507042253522
```

Accuracy: 81.19% Predictions are Correct
Precision: 83.92% Positive Predictions are Correct
Sensitivity: 87.11% Positive Cases were Predicted as Positive
Specificity: 70.84% Negative Cases were Predicted as Negative

ROC curve:

```
#ROC Curve:

fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--', label='No Skill')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Decesion Trees')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Support Vector Machines Model:

```
#3(iv)
SVclf = SVC(kernel='linear')
SVclf.fit(X_train, y_train)
y_pred=SVclf.predict(X_test)

print("Support Vector Machines")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))


cm = confusion_matrix(y_test, y_pred)
tn,fp, fn, tp = cm.ravel()

sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
precision = tp/(tp+fp)
print("Precision:", precision)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)

```

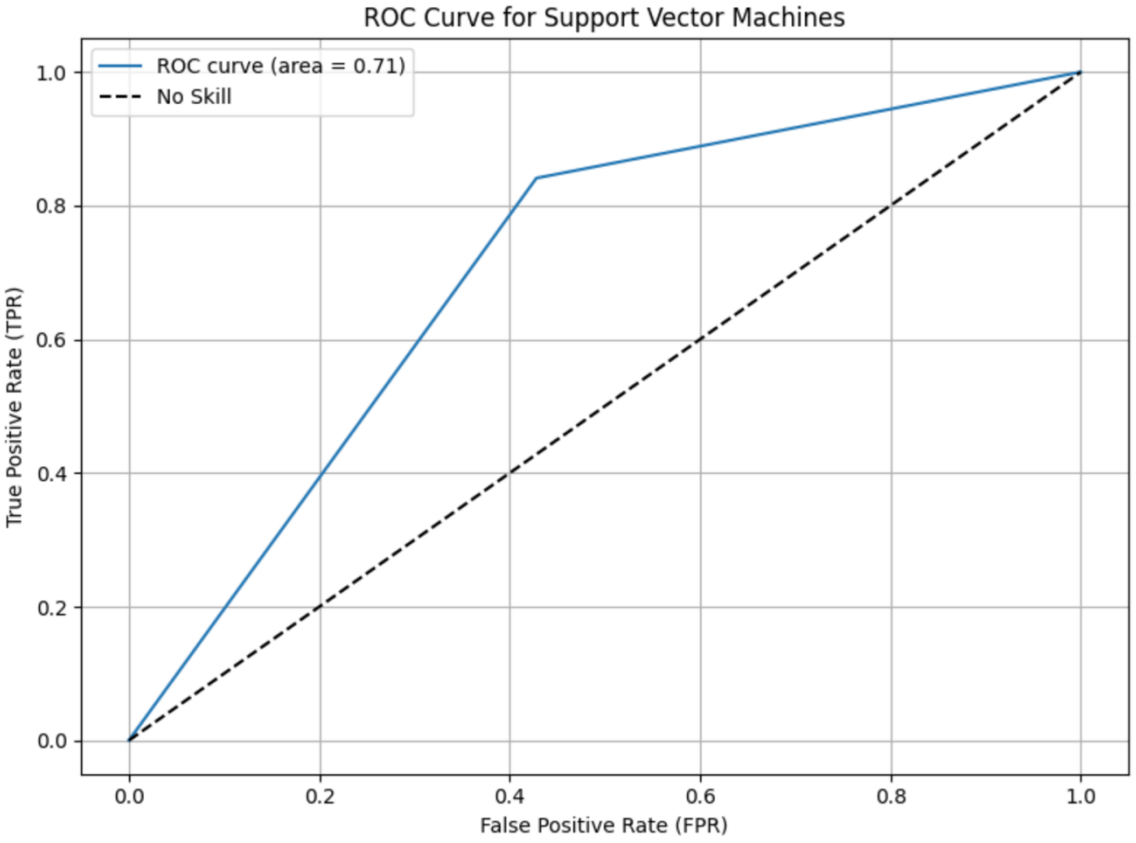
Output:

```
Support Vector Machines
Confusion Matrix:
 [[ 406  304]
 [ 197 1044]]
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.57      0.62       710
           1       0.77      0.84      0.81      1241

    accuracy                           0.74      1951
   macro avg       0.72      0.71      0.71      1951
weighted avg       0.74      0.74      0.74      1951

Accuracy Score: 0.743208610968734
Sensitivity: 0.8412570507655117
Specificity: 0.571830985915493
Precision: 0.7744807121661721
```

Accuracy: 74.32% Predictions are Correct
Precision: 84.12% Positive Predictions are Correct
Sensitivity: 57.18% Positive Cases were Predicted as Positive
Specificity: 77.44% Negative Cases were Predicted as Negative

ROC curve:

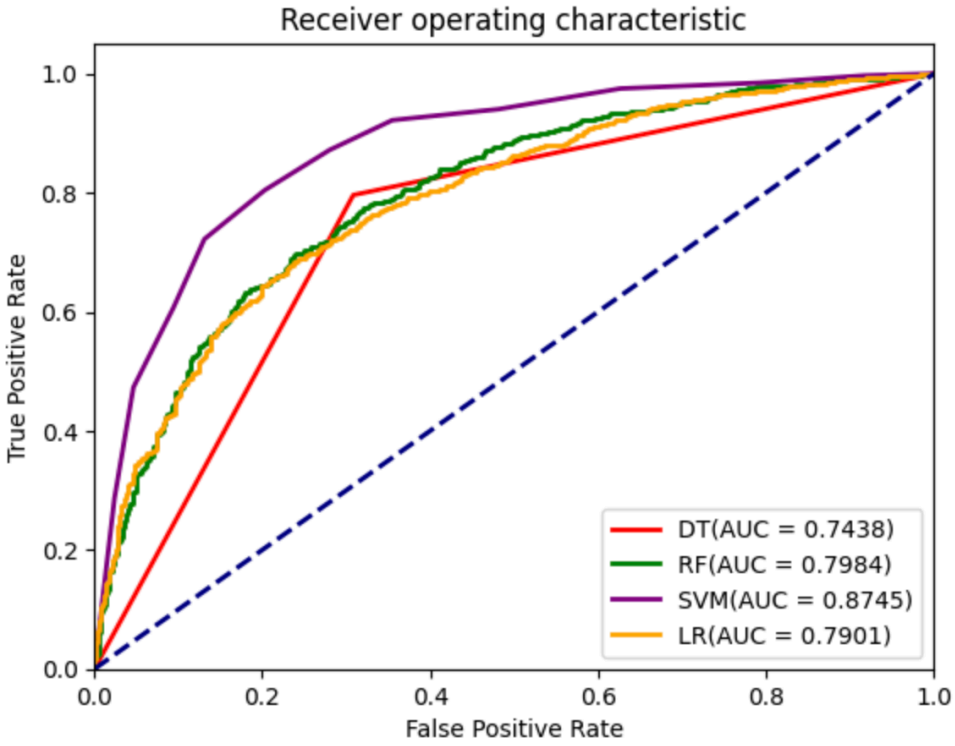
```
#ROC Curve:

fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--', label='No Skill')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Decesion Trees')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Comparing ROC Curve for all the models:

```
#####Predict probabilities for the test data.
probsDT = DTclf.predict_proba(X_test)
####Keep Probabilities of the positive class only.
probsDT = probsDT[:, 1]
#####Predict probabilities for the test data.
probsRF = RFclf.predict_proba(X_test)
####Keep Probabilities of the positive class only.
probsRF = probsRF[:, 1]
probsSV = SVclf.fit(X_train, y_train).decision_function(X_test)
## calculate the fpr and tpr for all thresholds of the classification
probs = logmodel.predict_proba(X_test)
predLR = probs[:,1]
###Compute the AUC Score.
auc = roc_auc_score(y_test, probsDT)
auc2 = roc_auc_score(y_test, probsRF)
auc3 = roc_auc_score(y_test, probsSV)
auc4 = roc_auc_score(y_test, predLR)
print('DT AUC:', auc)
print('RF AUC2:', auc2)
print('SVM AUC3:', auc3)
print('LR AUC4:', auc4)
###Get the ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, probsDT)
fpr2, tpr2, thresholds2 = roc_curve(y_test, probsRF)
fpr3, tpr3, thresholds3 = roc_curve(y_test, probsSV)
fpr4, tpr4, thresholds4 = roc_curve(y_test, predLR)
####Plot ROC Curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='red',
         lw=lw, label='DT(AUC = %0.4f)' % auc)
plt.plot(fpr3, tpr3, color='green',
         lw=lw, label='RF(AUC = %0.4f)' % auc3)
plt.plot(fpr2, tpr2, color='purple',
         lw=lw, label='SVM(AUC = %0.4f)' % auc2)
plt.plot(fpr4, tpr4, color='orange',
         lw=lw, label='LR(AUC = %0.4f)' % auc4)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

```
DT AUC: 0.7438407236326905
RF AUC2: 0.8744634608618673
SVM AUC3: 0.7984417382619651
LR AUC4: 0.7901453847987199

<function matplotlib.pyplot.show(close=None, block=None)>
```



Best performed model:

Model Name	**AUC Value**
Decision Tree	0.743
Random Forest	0.874
Support Vector Machine	0.798
Logistic Regression	0.790

Hence, the best performed model is **Random Forest.**