Project Name - **Red Wine Quality Prediction**

Name - Aman Mulla.

Batch - DS2307

Project Summary -

The Red Wine Quality Prediction project have objective to develop a machine learning model that can predict the quality of red wines based on various chemical features. Red wine quality is a critical factor in the wine industry, and being able to predict it accurately can help winemakers enhance their production processes and improve overall product quality.

We have below chemical feature based on physicochemical tests,

1. **Fixed Acidity** - Represents the concentration of non-volatile acid in the wine.
2. **Volatile Acidity** - Represents the concentration of volatile acid in the wine.
3. **Citric Acid** - Represents the amount of Citric acid in the wine.
4. **Residual Sugar** - Represents the amount of sugar that remains in the wine.
5. **Chlorides** - Represents the amount of Chlorides (Salt) in the wine.
6. **Free Sulfur Dioxide** - Its preservative in wine.
7. **Total Sulfur Dioxide** - Represents the total amount of sulfur dioxide in the wine.
8. **Density** - Density measures the wine's mass per unit volume.
9. **pH Value** - pH Value the acidity or basicity of the wine.
10. **Sulphates** - Represents the amount of Sulphates that in the wine.
11. **Alcohol** - Represents the amount of alcohol present in the wine.
12. **Quality** - Quality is ranging from 0 to 10, representing the overall quality of the red wine. Where 10 is better quality and 0 is less quality.

Problem Statement

To develop a predictive machine learning model that can accurately predict the quality of red wines based on physicochemical properties.Can assign a quality score between 0 and 10 to each wine.

To be more specific will work for below problems,

1. Which features are more important in determining the wine quality.
2. Contribution of each factor to the wine quality in our model.
3. To predict wine quality is ranging from 0 to 10, representing the overall quality of the red wine. Where 10 is better quality and 0 is less quality.

Knowing data and variable in dataset

```
# Importing Necessary Libraries # Other necessary libraries wil import at the time of code execution.

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

pd.set_option('display.max_rows', None)


# Loading Dataset

wine_data = pd.read_csv('/content/drive/MyDrive/DataSets/wine.csv')


# Checkinh Head of dataset

wine_data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	Alcohol_content	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	Low	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	Medium	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	Medium	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	Medium	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	Low	5

```
wine_data.columns
```

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

To avoid type error will remane for some of column names

```
wine_data.rename(columns={'fixed acidity': 'fixed_acidity'}, inplace=True)

wine_data.rename(columns={'volatile acidity': 'volatile_acidity'}, inplace=True)

wine_data.rename(columns={'citric acid': 'citric_acid'}, inplace=True)

wine_data.rename(columns={'residual sugar': 'residual_sugar'}, inplace=True)

wine_data.rename(columns={'free sulfur dioxide': 'free_sulfur_dioxide'}, inplace=True)

wine_data.rename(columns={'total sulfur dioxide': 'total_sulfur_dioxide'}, inplace=True)


wine_data.head()
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	Alcohol_content	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	Low	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	Medium	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	Medium	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	Medium	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	Low	5

```
# Will Check for shape of dataset

wine_data.shape
```

```
(1599, 13)
```

There were 1599 records and 13 attributes in the dataset.

Dataset Information

```
wine_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 13 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  Alcohol_content       1599 non-null   object 
 12  quality               1599 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 162.5+ KB
```

From .info(), we can observe that most of variables are in float64 dtype, only one variable that is alcohol_content with object and one with int.

So will convert object datatype into float or int by applying encoding on alcohol_content variable it will be easy to build ML Models.

```
wine_data['Alcohol_content'].unique()
```

```
array(['Low', 'Medium', 'High'], dtype=object)
```

```
# Will check for unique values in Alcohol_content variable

wine_data['Alcohol_content'].unique()
# We have 3 unique values in Alcohol_content, will encode as Low= 1, Medium =2, High =3

encoding_nums = {"Alcohol_content": {"Low": 1, "Medium": 2, "High": 3}}
encoding_nums
wine_data = wine_data.replace(encoding_nums)



# Will check for head of dataset, and info of dataset.

wine_data.head()

wine_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 13 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  Alcohol_content       1599 non-null   int64  
 12  quality               1599 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 162.5 KB
```

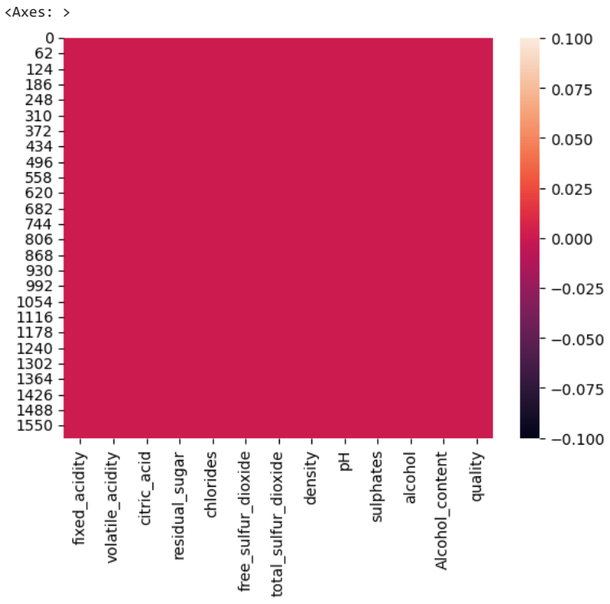
Will Check for Null value in dataset

```
wine_data.isna().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
Alcohol_content         0
quality                 0
dtype: int64
```

```
sns.heatmap(wine_data.isnull())
```

```
<Axes: >
```



As per isna(), No any null value present in any of variable, along with same all the variables are in float and int datatype, will move further on EDA.

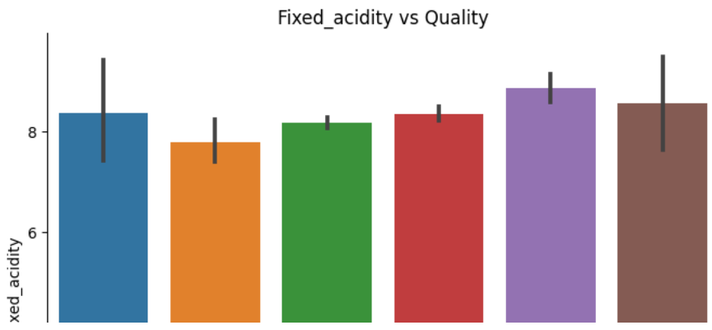
```
wine_data.columns
```

```
Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'Alcohol_content', 'quality'],
      dtype='object')
```

Chart - 1

Fixed_acidity vs Quality

```
# will plot bar plot for Fixed_acidity vs Quality and check for insights

f,ax = plt.subplots(figsize = (8,6))
sns.despine(f)
sns.barplot(y = 'fixed_acidity',x ='quality', data = wine_data)
plt.xlabel('quality')
plt.ylabel('Fixed_acidity')
plt.title('Fixed_acidity vs Quality')

plt.show()
```



Insights:

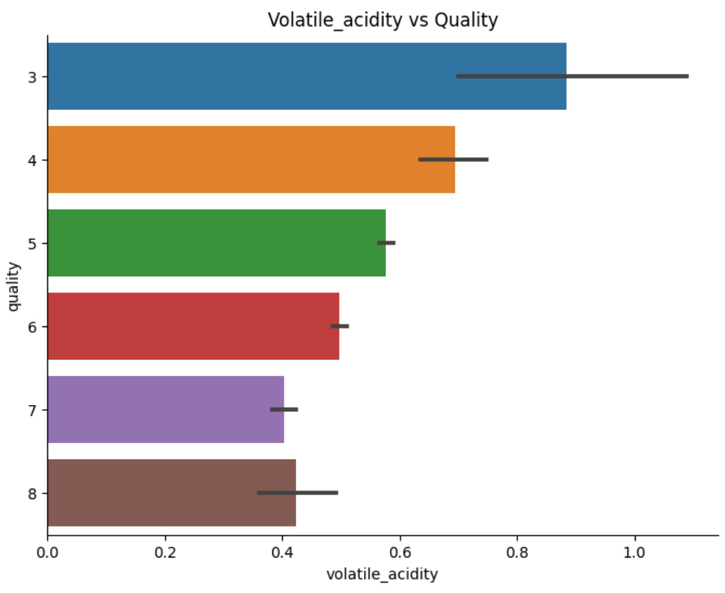
Above bar graph represents the fixed acidity level for wines with a specific quality rating. The bar indicate fixed acidity for wines in that quality category.

- Wines with quality ratings of 5 and 6 appear to have similar fixed acidity levels.
- Quality ratings 7 and 8 have the highest fixed acidity level, and this level decreases as you move towards lower quality ratings, with a noticeable drop at quality rating 4.
- Higher quality wines generally have higher fixed acidity levels, which suggests that wines with higher fixed acidity might be preferred by consumers.

Chart - 2

Volatile_acidity vs Quality

```
f,ax = plt.subplots(figsize = (8,6))
sns.despine(f)
sns.barplot(y = 'quality',x ='volatile_acidity', data = wine_data,orient ='h')
plt.xlabel('volatile_acidity')
plt.ylabel('quality')
plt.title('Volatile_acidity vs Quality')

plt.show()
```



Insights:

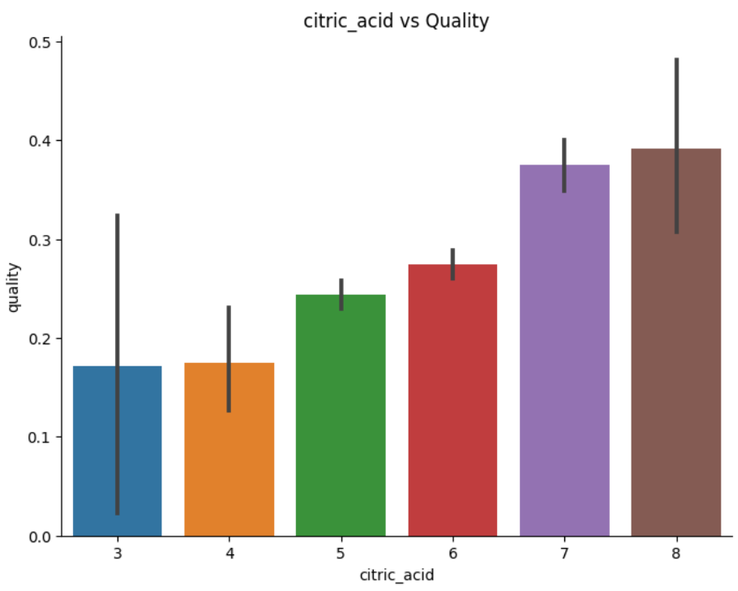
For above bar graph, each horizontal bar on the graph represents the volatile acidity level for wines with a specific quality rating.

- Quality ratings 7 and 8 have notably lower levels of volatile acidity compared to lower quality ratings. This suggests that wines with these higher quality ratings have a more balanced and pleasant acidity level.
- Quality ratings 3 and 4 have the highest average levels of volatile acidity, indicating that wines in these quality categories tend to have higher volatile acid content.
- The graph suggests that volatile acidity is inversely related to wine quality, with higher-quality wines having lower volatile acidity levels.

Chart - 3

Citric_acid vs Quality

```
f,ax = plt.subplots(figsize = (8,6))
sns.despine(f)
sns.barplot(x = 'quality',y ='citric_acid', data = wine_data)
plt.xlabel('citric_acid')
plt.ylabel('quality')
plt.title('citric_acid vs Quality')

plt.show()
```



Insights:

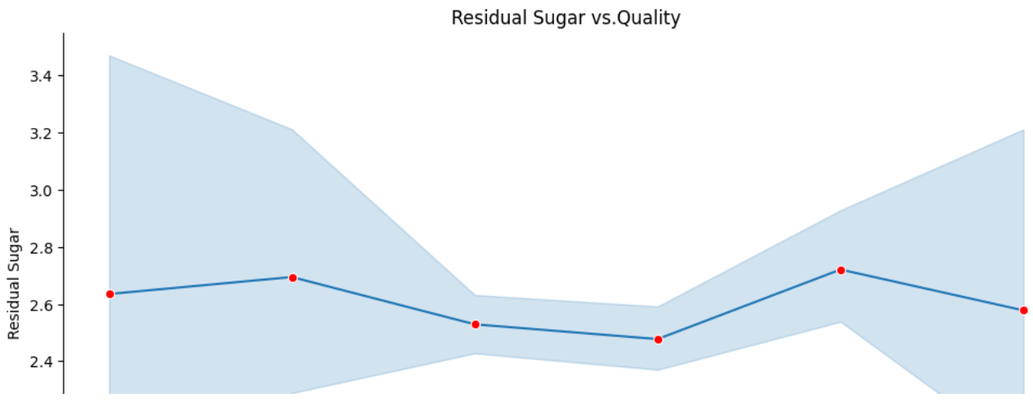
Above bar graph represents the citric_acid level for wines with a specific quality rating. The bar indicate citric_acid for wines in that quality category.

- Higher-quality wines tend to have higher average levels of citric acid,which indicat that citric acid may contribute positively to the get good quality of red wines.
- Quality ratings 7 and 8 have higher citric acid levels, while Quality ratings 3 and 4 have lower citric acid levels.This indicating that wines with higher quality ratings tend to have more citric acid and wines with lower quality ratings tend to have less citric acid.

Chart - 4

Residual_sugar vs Quality

```
f,ax = plt.subplots(figsize=(12,6))
sns.despine(f)
sns.lineplot(x='quality', y='residual_sugar', marker='o',markerfacecolor='red', data=wine_data)
plt.xlabel('Quality')
plt.ylabel('Residual Sugar')
plt.title('Residual Sugar vs.Quality')

plt.show()
```



Insights:

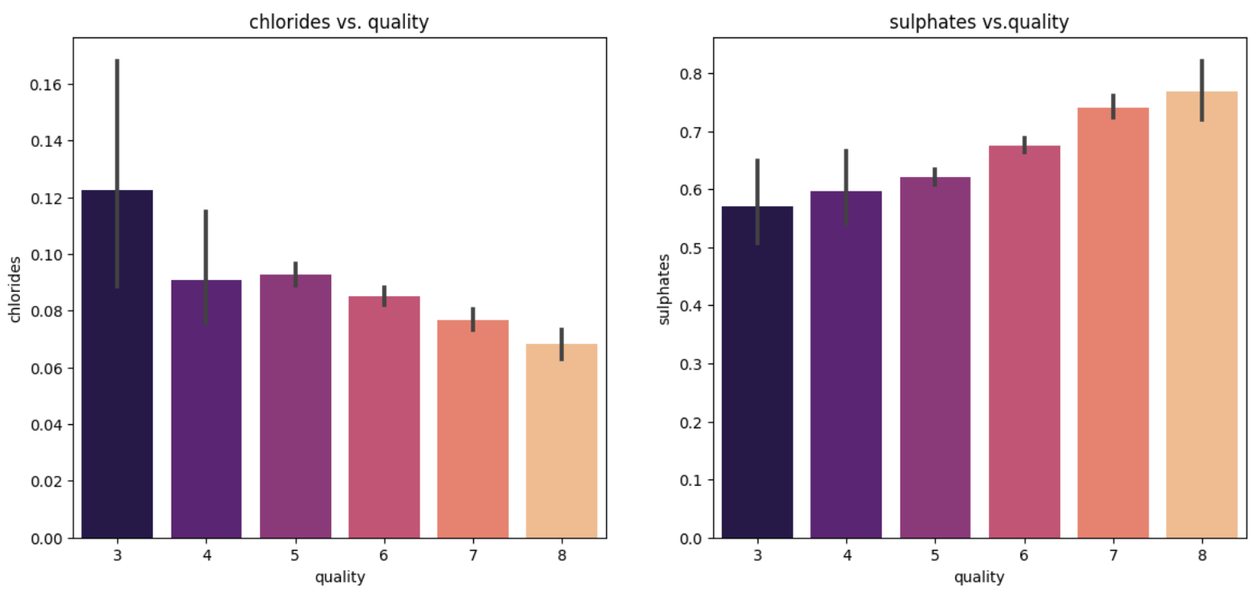
The above line plot connects data points for each quality rating, showing how the average level of residual sugar changes as wine quality varies.

- The line plot indicates that there is the average level of residual sugar tends to increase as wine quality improves. This means that wines with higher quality ratings tend to have slightly higher levels of residual sugar.
- Quality ratings 7 and 8 show an increase in residual sugar levels compared to quality ratings 5 and 6.

Chart - 4

Chlorides vs Quality & sulphates vs.quality

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))
sns.barplot(y='chlorides', x='quality', ax=axes[0], palette='magma',data=wine_data)
axes[0].set_xlabel('quality')
axes[0].set_ylabel('chlorides')
axes[0].set_title('chlorides vs. quality')

sns.barplot(x='quality', y='sulphates', ax=axes[1], palette='magma',data=wine_data)
axes[1].set_xlabel('quality')
axes[1].set_ylabel('sulphates')
axes[1].set_title('sulphates vs.quality')

plt.show()
```



Insights:

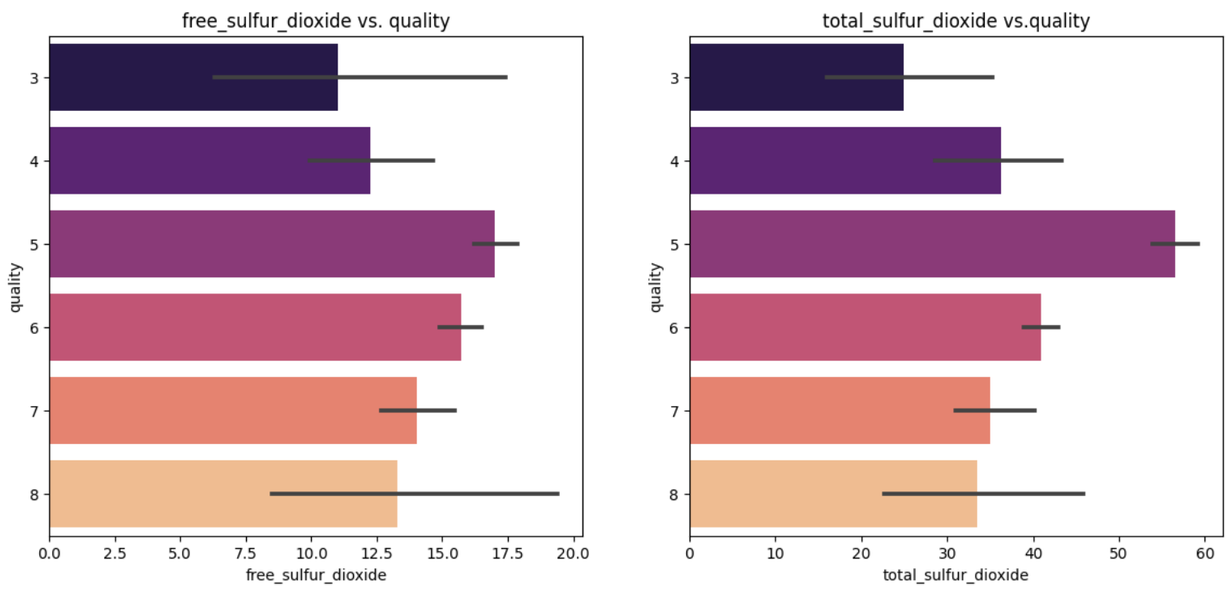
Above bar graph represents the chlorides and sulphates level for wines with a specific quality rating. The bar indicate fixed acidity for wines in that quality category.

- The graph indicates a trend where lower-quality wines tend to have higher levels of chlorides.
- Quality ratings 7 and 8 have lower chloride levels compared to lower quality ratings.of 3 and 4.
- Chloride content is inversely related to wine quality, with higher-quality wines having lower chloride levels.
- Quality ratings 7 and 8 have notably higher levels of sulphates compared to lower quality ratings. This suggests that wines with these higher quality ratings tend to have a higher level of sulphates.
- Quality ratings 3, 4, and 5 have lower sulphate levels. Wines in these quality categories have a lower levels of sulphates.
- The graph implies that sulphates are positively correlated with wine quality. Higher-quality wines are more likely to have a higher sulphate content.

Chart - 5

free_sulfur_dioxide vs. quality & total_sulfur_dioxide vs.quality

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))
sns.barplot(x='free_sulfur_dioxide', y='quality', ax=axes[0], palette='magma',data=wine_data,orient='h')
axes[0].set_xlabel('free_sulfur_dioxide')
axes[0].set_ylabel('quality')
axes[0].set_title('free_sulfur_dioxide vs. quality')

sns.barplot(y='quality', x='total_sulfur_dioxide', ax=axes[1], palette='magma',data=wine_data,orient='h')
axes[1].set_xlabel('total_sulfur_dioxide')
axes[1].set_ylabel('quality')
axes[1].set_title('total_sulfur_dioxide vs.quality')

plt.show()
```



Insights:

Above bar graph represents the free_sulfur_dioxide and total_sulfur_dioxide level for wines with a quality rating.Each bar in the left plot represents the levels of free sulfur dioxide while Each bar in the left plot represents the levels of total_sulfur_dioxide for wines with a quality rating.

For free_sulfur_dioxide vs. quality:

- Quality ratings 3 and 4 have slightly lower levels of free sulfur dioxide compared to other quality ratings. This suggests that wines in these quality categories may require lower levels of free sulfur dioxide.
- Quality ratings 5 and 6 have slightly higher levels of free sulfur dioxide compared to other quality ratings. This suggests that wines in these quality categories may require higher levels of free sulfur dioxide.

For free_sulfur_dioxide vs. quality:

- Quality ratings 3 and 8 have much lower levels of total_sulfur_dioxide compared to other quality ratings. This suggests that wines in these quality categories may require lower levels of total_sulfur_dioxide.
- Quality ratings 5 have slightly higher levels of total_sulfur_dioxide compared to other quality ratings. This suggests that wines in these quality categories may require higher levels of total_sulfur_dioxide.

- Also for total_sulfur_dioxide vs. quality, indicates that quality rating 5 have much higher amount of total_sulfur_dioxide as compared to others, so for making wine of quality rating 5 total_sulfur_dioxide required in much amount.

```
wine_data.columns
```

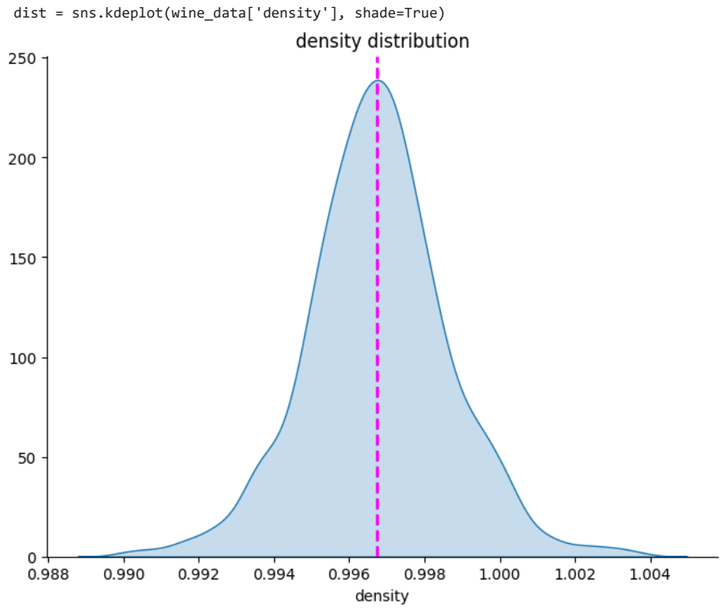
```
Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'Alcohol_content', 'quality'],
      dtype='object')
```

Chart - 6

density and pH distribution

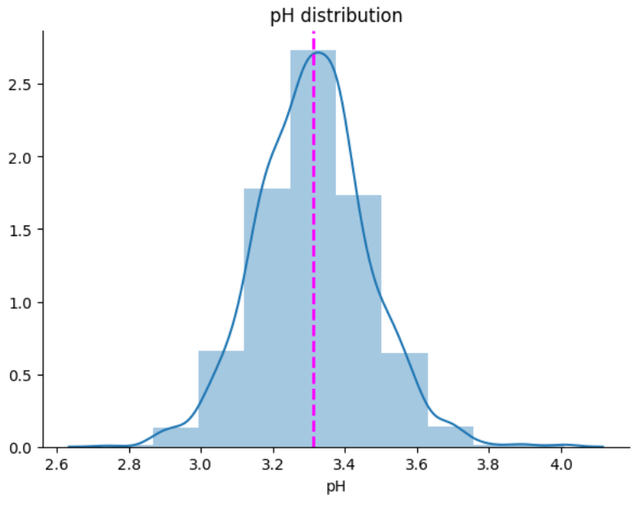
```
f,ax  = plt.subplots(figsize=(8,6))

sns.despine(f)
dist = sns.kdeplot(wine_data['density'], shade=True)
dist.set(xlabel = 'density', ylabel ='', title = 'density distribution')

dist.axvline(wine_data['density'].mean(), color='magenta', linestyle='dashed', linewidth=2)

plt.show()
```

```
<ipython-input-90-28c31760b344>:4: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  dist = sns.kdeplot(wine_data['density'], shade=True)
```




```
f,ax = plt.subplots(figsize=(7,5))
sns.despine(f)
dist = sns.distplot(wine_data['pH'],bins=10)
dist.set(xlabel = 'pH', ylabel ='', title = 'pH distribution')

dist.axvline(wine_data['pH'].mean(), color='magenta', linestyle='dashed', linewidth=2)
```

```
<ipython-input-91-352705520d6a>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  dist = sns.distplot(wine_data['pH'],bins=10)
<matplotlib.lines.Line2D at 0x7c58531e38e0>
```



Insights:

Above KDA graph represents distribution of density and pH for all quality ratings.

- From graph we can observe that density and pH is normally distributed.
- From the density distribution, we can observe that the density is centered around a mean of 0.997.
- From the pH distribution, we can observe that the pH is distributed with a mean of 3.3.
- The shape of the curve suggests that most wines have similar densityand pH values, with variations around the central peak.

```
wine_data.columns
```

```
Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'Alcohol_content', 'quality'],
      dtype='object')
```

Chart - 7

Alcohol vs. quality

```
f,ax = plt.subplots(figsize = (8,6))
sns.despine(f)
sns.barplot(x = 'quality',y ='alcohol', data = wine_data)
plt.xlabel('quality')
plt.ylabel('alcohol level')
plt.title('alcohol vs Quality')

plt.show()
```

Insights:

Above bar graph represents the alcohol level for wines with a specific quality rating. The bar indicate alcohol for wines in that quality category.

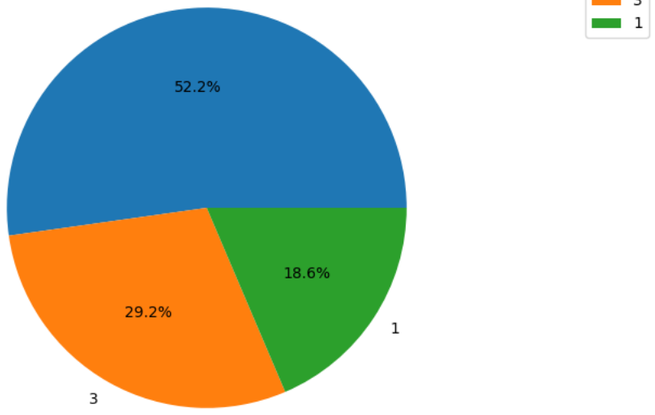
- The graph indicate a positive correlation between alcohol level and wine quality. Higher-quality wines tend to have higher level of alcohol content,while lower-quality wines have lower level of alcohol content.
- Quality ratings 7 and 8 shows the highest alcohol content. This suggests that wines with these higher quality ratings tend to be better in alcohol.
- Quality ratings 3 and 5 have the lowest average alcohol content, indicating that wines in these quality categories generally contain less alcohol.

Chart - 7

Alcohol_content vs. quality

```
Alcohol_content_count = wine_data['Alcohol_content'].value_counts()

print(Alcohol_content_count)    # Low= 1, Medium =2, High =3

plt.figure(figsize=(8,6))
ax = plt.subplot(111)
plt.pie(x=Alcohol_content_count.values, labels=Alcohol_content_count.index, autopct='%1.1f%%')
plt.legend()
ax.legend(bbox_to_anchor=(1.4, 1))
plt.show()
```

```
2    835
3    467
1    297
Name: Alcohol_content, dtype: int64
```



Insights:

This pie chart offers a clear representation of the distribution of wines in the dataset across different alcohol content categories as Low, Medium, and High.(Low= 1, Medium =2, High =3)

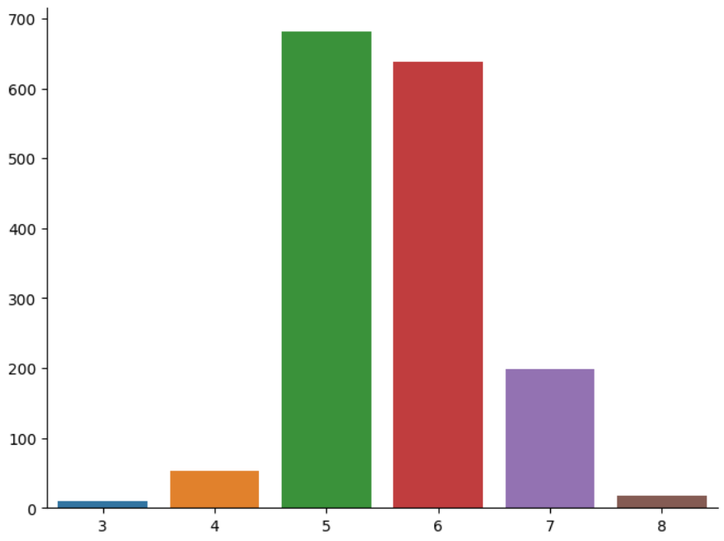
- Majority of wines fall into the "Medium" alcohol content category(52.2%).The "Low" and "High" alcohol content categories are less common.(18.6% and 29.2% respectively).

Chart - 8

Quality Distribution

```
quality_count = wine_data['quality'].value_counts()

print(quality_count)

f,ax = plt.subplots(figsize=(8,6))
sns.despine(f)
sns.barplot(y = quality_count.values, x = quality_count.index ,data=wine_data )

plt.show()
```

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



Insights:

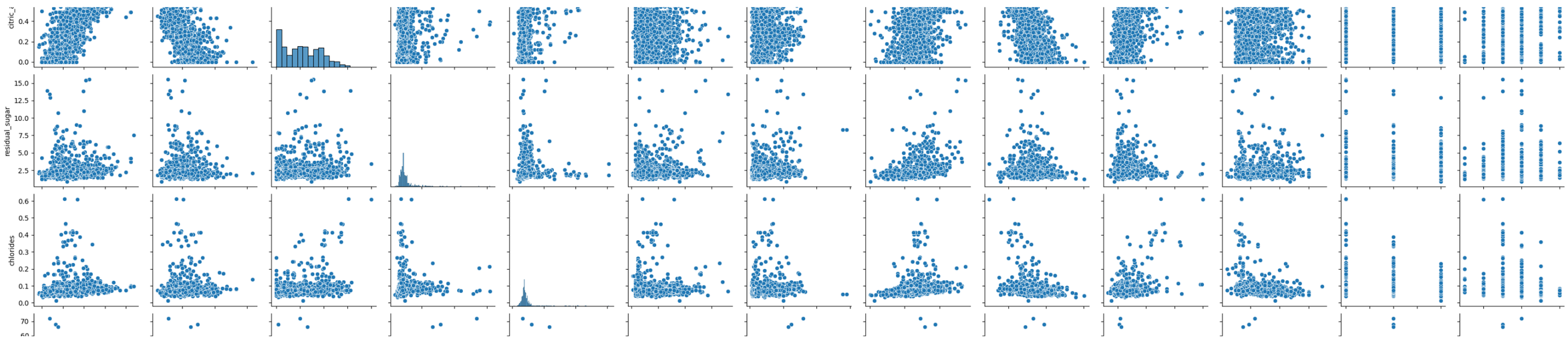
The graph provides a distribution of wine quality ratings in the dataset. It shows how many wines fall into each quality category.

- Quality ratings 5 and 6 are the most common in the dataset, with quality rating 5 having a slightly higher count than quality rating 6.
- Very fewer wines with quality ratings 3, 4, and 8 in the dataset. Quality rating 7 is also less common.

Chart - 9

Pair Plot

```
sns.pairplot(wine_data)

plt.show()
```



Insights:

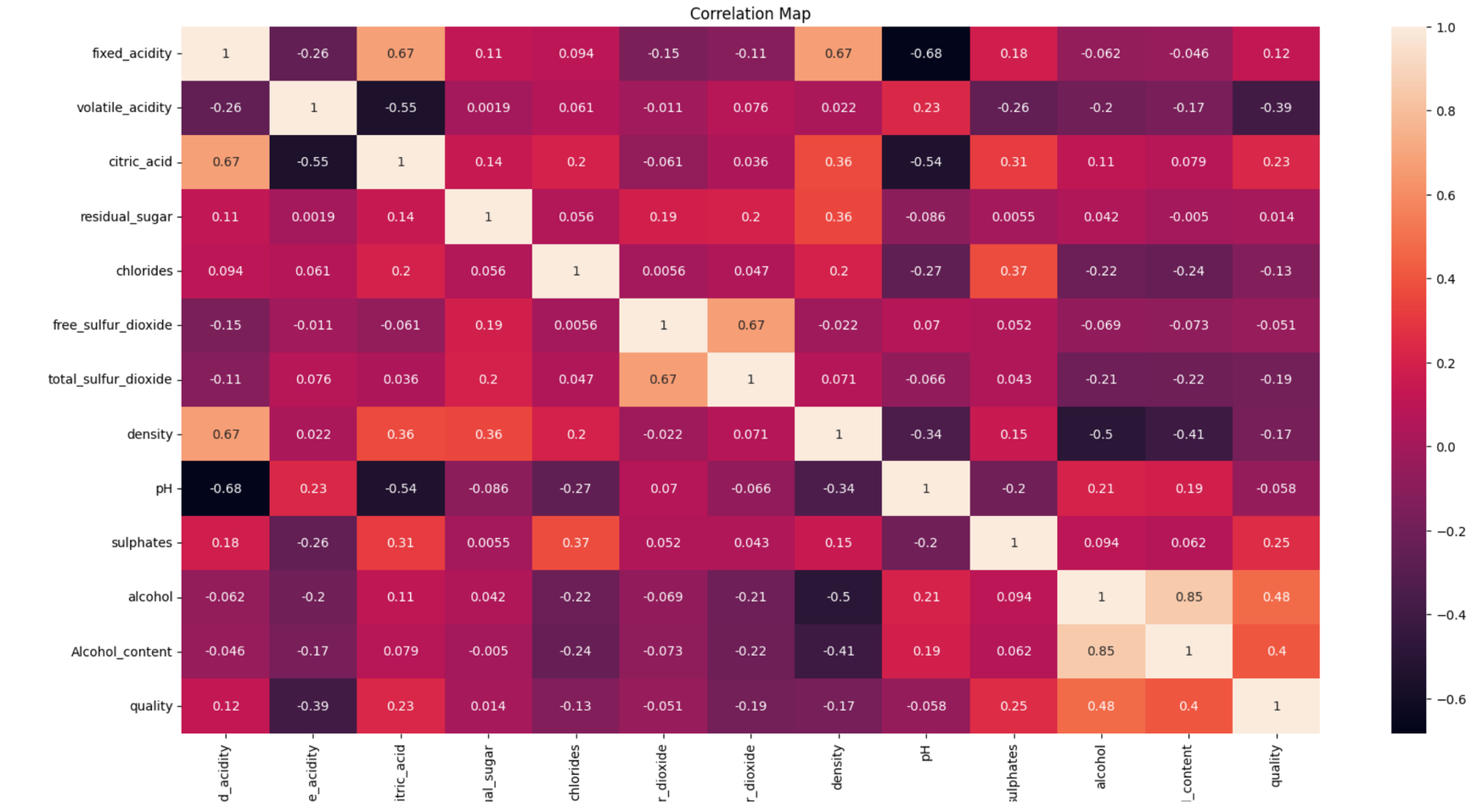
This type of visualization is useful for understanding the relationships and distributions of variables.

- We can observe that relation and distribution of each varibale, we have have positive correlation for 'density' and 'residual_sugar' , 'free_sulfur_dioxide' and 'total_sulfur_dioxide' , 'sulphates' and 'alcohol'.
- Also we have negative correlation for 'pH' and 'fixed_acidity' ,'alcohol' and 'density'.

Chart - 10

Heatmap

```
correlation_data = wine_data

correlation_matrix = correlation_data.corr()

plt.figure(figsize=(20,10))

sns.heatmap(correlation_matrix,annot=True)
plt.title('Correlation Map')
plt.show()
```



Insights:

- **Positive Correlation:** 'alcohol' and 'Alcohol_content' ,'citric_acid' and 'fixed_acidity', 'free_sulfur_dioxide' and 'total_sulfur_dioxide', 'density' and 'fixed_acidity' this vearibal showing positive corrlation.
- **Negative Correlation:** 'volatile_acidity' and 'citric_acid' , 'fixed_acidity' and 'volatile_acidity' , 'pH' and 'fixed_acidity' , 'alcohol' and 'density' this vearibal showing negative corrlation.
- The heatmap highlights that "Quality" is most strongly correlated with "Alcohol," "Sulphates," "Volatile Acidity," "Chlorides," and "Total Sulfur Dioxide."

Now, Done with EDA part will check for 3 hypothesis statement.

Hypothetical Statements and result.

Statement 1

The fixed acidity of wines with a quality rating greater than or equal to 6 is significantly different from the fixed acidity of wines with a quality rating less than 6.

Ho : There is a significant difference in mean fixed acidity.

H1 : There is no significant difference in mean fixed acidity.

```
# Will check for p-value

import scipy.stats as stats

high_quality_wines = wine_data[wine_data['quality'] >= 7]['fixed_acidity']
low_quality_wines = wine_data[wine_data['quality'] < 7]['fixed_acidity']

# perform t test on selected criteria
t_stat, p_value = stats.ttest_ind(high_quality_wines, low_quality_wines, equal_var=False)

print(p_value)


if p_value < 0.05:  # 0.05 is alpha value
    print("Reject the null hypothesis: There is a significant difference in mean fixed acidity.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in mean fixed acidity.")
```

```
 2.7967491696235023e-05
 Reject the null hypothesis: There is a significant difference in mean fixed acidity.
```

Statement 2

There is a correlation between the levels of citric acid and sulphates in red wines.

Ho : There is a significant correlation between citric acid and sulphates.

H1 : There is no significant correlation between citric acid and sulphates.

```
# using pearsonr from scipy.stats

from scipy.stats import pearsonr

citric_acid = wine_data['citric_acid']
sulphates = wine_data['sulphates']

corr_coefficient, p_value = pearsonr(citric_acid, sulphates)

if p_value < 0.05:
    print("There is a significant correlation between citric acid and sulphates.")
else:
    print("There is no significant correlation between citric acid and sulphates.")
```

```
 There is a significant correlation between citric acid and sulphates.
```

Statement 3

The distribution of alcohol content in red wines with a quality rating of 5 is significantly different from the distribution of alcohol content in red wines with a quality rating of 6.

Ho : The distributions are significantly different.

H1 : The distributions are not significantly different.

```
# using t-test to find p_value

alcohol_rating_5 = wine_data[wine_data['quality'] == 5]['alcohol']
alcohol_rating_6 = wine_data[wine_data['quality'] == 6]['alcohol']

t_stat, p_value = stats.ttest_ind(alcohol_rating_5, alcohol_rating_6, equal_var=False)


if p_value < 0.05:
    print("Reject the null hypothesis: The distributions are significantly different.")
else:
    print("Fail to reject the null hypothesis:The distributions are not significantly different.")
```

```
 Reject the null hypothesis: The distributions are significantly different.
```

ML Model Implementation

Before Moving toward ML Models, have to convert y variable which is 'quality' into bivariate.

```
wine_data['quality'].unique()     # 8,7,6 =1  # 3,4,5 = 0
```

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

```
wine_data['quality'].unique()

# So, in quality variable we have total 6 unique values so, will do encoding as for quality rating 8,7,6 will assign 1 that is good quality, and for quality rating 5,4,3 will assign 0 that is Not-Good quality

encoding_nums = {"quality": {8:1, 7:1, 6:1, 5:0, 4:0, 3:0}}
encoding_nums

wine_data = wine_data.replace(encoding_nums)


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

```
1    855
0    744
Name: quality, dtype: int64
```

ML Model - 1

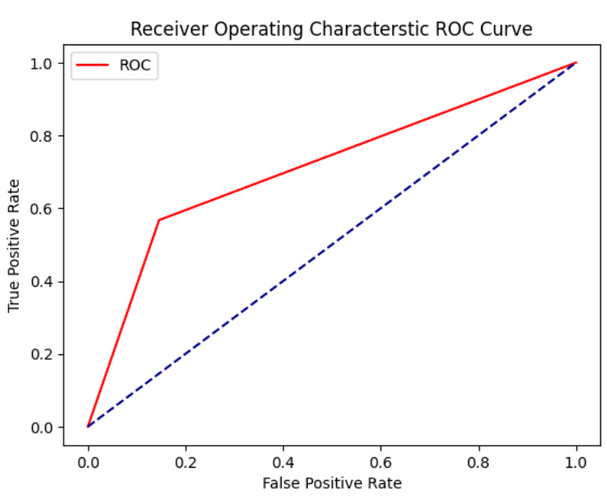
Using all Variables for ML Model-1

```
# Importing Necessary Libraries

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score,confusion_matrix,roc_curve,roc_auc_score
from sklearn.metrics import classification_report


# For ML Model 1 Using all variables from dataset

x = wine_data.drop(columns=['quality'])  # dependent Variables

y = wine_data['quality'] # idependent Variables

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting Logistic Regression model to dataset

log_reg = LogisticRegression()

log_reg.fit(x_train,y_train)

# will predict on x_train

y_pred_train = log_reg.predict(x_train)

y_pred_train

# prediction on x_test
y_pred = log_reg.predict(x_test)

y_pred

# Predicted probablity on x_test

log_reg.predict_proba(x_test)
```

```
              [0.57821939, 0.42178061],
              [0.75465287, 0.24534713],
              [0.4126137 , 0.5873863 ],
              [0.81201763, 0.18798237],
              [0.34644002, 0.65355998],
              [0.32046391, 0.67953609],
              [0.28942652, 0.71057348],
              [0.70905454, 0.29094546],
              [0.91867137, 0.08132863],
              [0.28746144, 0.71253856],
              [0.19177311, 0.80822689],
              [0.7803951 , 0.2196049 ],
              [0.68707139, 0.31292861],
              [0.45726583, 0.54273417],
              [0.47462939, 0.52537061],
              [0.18654306, 0.81345694],
              [0.88506121, 0.11493879],
              [0.87505085, 0.12494915],
              [0.78610473, 0.21389527],
              [0.47382885, 0.52617115],
              [0.77276184, 0.22723816],
              [0.85315749, 0.14684251],
              [0.30909385, 0.69090615],
              [0.10863638, 0.89136362],
              [0.54767939, 0.45232061],
              [0.93923849, 0.06076151],
              [0.34665024, 0.65334976],
              [0.34418864, 0.65581136]])
```

```
# Model Accuracy

accuracy = accuracy_score(y_test,y_pred)

print("Accurcy of ML Model-1:",accuracy*100,'%')

# Using Confusion Matrix for checking accuracy of model

confusion_mat = confusion_matrix(y_test,y_pred)

confusion_mat

# Checking accuracy of model with classification report

print(classification_report(y_test,y_pred))

# Plotting ROC Curve

fpr,tpr,thresholds = roc_curve(y_test,y_pred)  # where,fpr = False Positive Rate ,tpr = True positive Rate , Thresholds means how data is changing

plt.plot(fpr,tpr,color ='red',label='ROC')
plt.plot([0,1],[0,1],color='darkblue',linestyle='--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characterstic ROC Curve')
plt.legend()
plt.show()
```

```
Accurcy of ML Model-1: 70.0 %
              precision    recall  f1-score   support

           0       0.63      0.85      0.72       185
           1       0.82      0.57      0.67       215

    accuracy                           0.70       400
   macro avg       0.72      0.71      0.70       400
weighted avg       0.73      0.70      0.70       400
```



Evaluation Metrics:

1. **Accuracy Of ML Model-1 - 70%.**
2. **For class 0: Precision is 0.63, Recall is 0.85, and F1-score is 0.72.**
3. **For class 1: Precision is 0.82, Recall is 0.57, and F1-score is 0.67.**

Precision: The ratio of true positives to the total number of instances predicted as positive. High precision indicates low false positive rate.

Recall: The ratio of true positives to the total number of actual positive instances. High recall indicates low false negative rate.

F1-Score: The harmonic mean of precision and recall. It provides a balance between precision and recall.

```
# By using GridSearchCV will find for best parameters and by same will to hypertuning for model.

from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100,1000],  # Values of C to search
    'penalty': ['l1', 'l2'],  # L1 (Lasso) and L2 (Ridge) regularization
}

# Create a logistic regression model
logistic_regression = LogisticRegression(solver='liblinear')

# Create a grid search object
grid_search = GridSearchCV(logistic_regression, param_grid, cv=5, scoring='accuracy')

# Fitting to model
grid_search.fit(x_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model
best_model = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_model.predict(x_test)

# Getting Scores and classificaion report after hypertuning parameter.

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100,"%")

report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Best Hyperparameters: {'C': 100, 'penalty': 'l1'}
Accuracy: 69.5 %
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.88      0.73       185
           1       0.84      0.53      0.65       215

    accuracy                           0.69       400
   macro avg       0.73      0.71      0.69       400
weighted avg       0.74      0.69      0.69       400
```

Evaluation Metrics after hypertuning Parameter on ML Model-1:

1. **Accuracy Of ML Model-1 - 69.5%.**
2. **For class 0: Precision is 0.62, Recall is 0.88, and F1-score is 0.73.**
3. **For class 1: Precision is 0.84, Recall is 0.53, and F1-score is 0.65.**

```
wine_data.columns
```

```
Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'Alcohol_content', 'quality'],
      dtype='object')
```

ML Model - 2

Considering all continous variables (dropping for categorical variable)

Only one column with categorical values that is 'Alcohol_content'

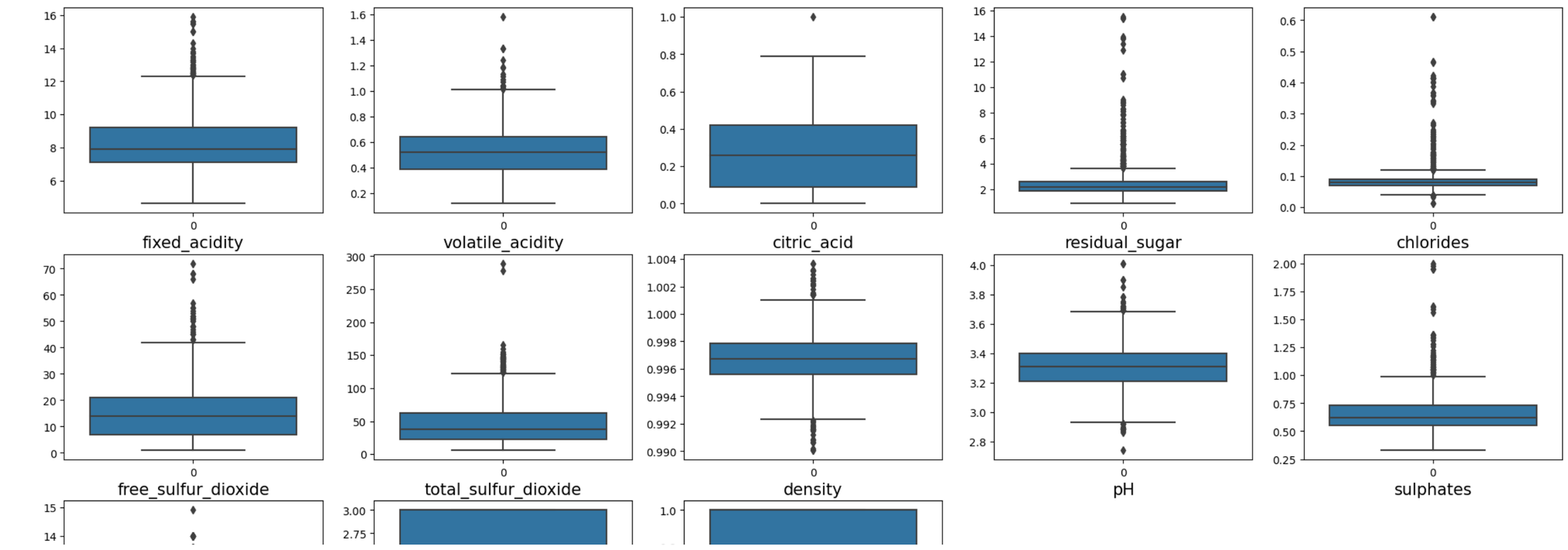
```
x = wine_data[['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar','chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol']]

y = wine_data['quality']


# For ML Model 1 Using all variables from dataset

x = wine_data.drop(columns=['quality'])  # dependent Variables

y = wine_data['quality'] # idependent Variables

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting Logistic Regression model to dataset

log_reg = LogisticRegression()

log_reg.fit(x_train,y_train)

# will predict on x_train

y_pred_train = log_reg.predict(x_train)

y_pred_train

# prediction on x_test
y_pred = log_reg.predict(x_test)

y_pred

# Predicted probablity on x_test

log_reg.predict_proba(x_test)

# Model Accuracy

accuracy = accuracy_score(y_test,y_pred)

print("Accurcy of ML Model-2:",accuracy*100,'%')

# Using Confusion Matrix for checking accuracy of model

confusion_mat = confusion_matrix(y_test,y_pred)

confusion_mat

# Checking accuracy of model with classification report

print(classification_report(y_test,y_pred))

# Plotting ROC Curve

fpr,tpr,thresholds = roc_curve(y_test,y_pred)  # where,fpr = False Positive Rate ,tpr = True positive Rate , Thresholds means how data is changing

plt.plot(fpr,tpr,color ='red',label='ROC')
plt.plot([0,1],[0,1],color='darkblue',linestyle='--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characterstic ROC Curve')
plt.legend()
plt.show()

# By using GridSearchCV will find for best parameters and by same will to hypertuning for model.

from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100,1000],  # Values of C to search
    'penalty': ['l1', 'l2'],  # L1 (Lasso) and L2 (Ridge) regularization
}

# Create a logistic regression model
logistic_regression = LogisticRegression(solver='liblinear')

# Create a grid search object
grid_search = GridSearchCV(logistic_regression, param_grid, cv=5, scoring='accuracy')

# Fitting to model
grid_search.fit(x_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model
best_model = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_model.predict(x_test)

# Getting Scores and classificaion report after hypertuning parameter.

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100,"%")

report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
(1199, 12)
(400, 12)
(1199,)
(400,)
Accurcy of ML Model-2: 70.0 %
```

For ML Model-2, we can observe that there is no significant change in the model's accuracy before and after tuning hyperparameters. This may be because we dropped only one variable from the previous model.

Before moveing for ML Model-3, will cehck for outliers present in each variable and will treat for outliers if required.

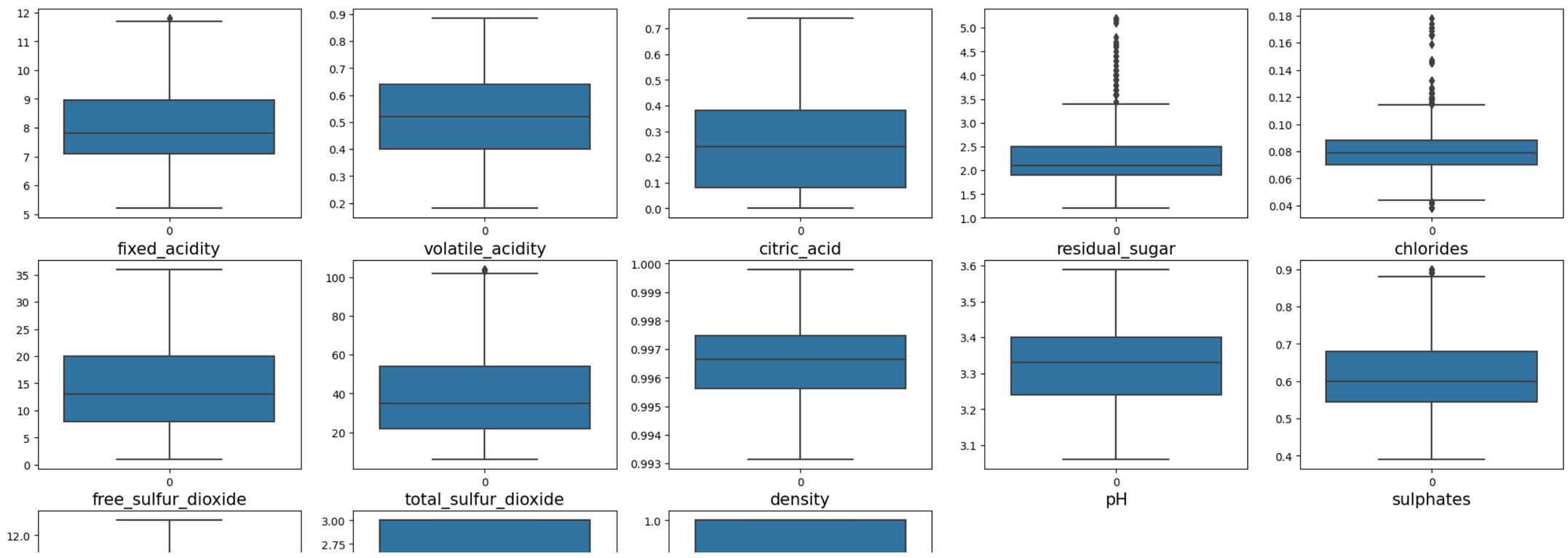
```
# Plotting box plot to know about outliers

plt.figure(figsize=(25,20))
graph = 1

for column in wine_data:
  if graph<=25:
    plt.subplot(5,5,graph)
    ax=sns.boxplot(data= wine_data[column])
    plt.xlabel(column,fontsize=15)
  graph+=1
plt.show()
```




```
# Define a threshold for the Z-score
z_score_threshold = 2  # You can adjust this threshold based on your data and requirements

# Select numerical columns where you want to detect and treat outliers
numerical_cols = ['fixed_acidity', 'volatile_acidity','residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol']

# Create a copy of the dataset for outlier treatment
no_outliers = wine_data.copy()

# Loop through each numerical column and detect and remove outliers
for col in numerical_cols:
    z_scores = stats.zscore(no_outliers[col])
    no_outliers = no_outliers[(z_scores < z_score_threshold) & (z_scores > -z_score_threshold)]

# Display the shape of the dataset after removing outliers
print("Shape of data after outlier removal:", no_outliers.shape)
```

```
Shape of data after outlier removal: (1015, 13)
```

```
# Plotting box plot to know outliers after treatment

plt.figure(figsize=(25,20))
graph = 1

for column in no_outliers:
  if graph<=25:
    plt.subplot(5,5,graph)
    ax=sns.boxplot(data= no_outliers[column])
    plt.xlabel(column,fontsize=15)
  graph+=1
plt.show()
```



From above Boxplot we can observe that most of outlier removed after using z_score threshold as 2. But for variable 'residual_sugar' and 'chlorides' still having much outliers, for Model-3 will dropping these 2 variable for better result.

ML Model - 3

Considering columns after outlier treatment.

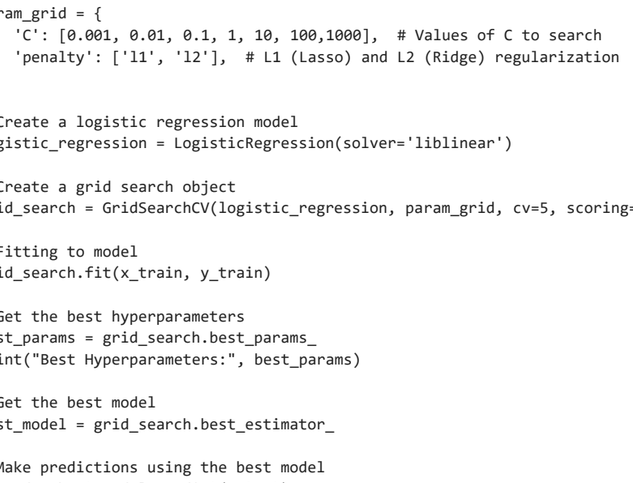
```
wine_data.columns
```

```
Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'Alcohol_content', 'quality'],
      dtype='object')
```

```
x = no_outliers[['fixed_acidity', 'volatile_acidity', 'citric_acid', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol','Alcohol_content']]

y = no_outliers['quality']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
```

```
print(y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting Logistic Regression model to dataset

log_reg = LogisticRegression()

log_reg.fit(x_train,y_train)

# will predict on x_train

y_pred_train = log_reg.predict(x_train)

y_pred_train

# prediction on x_test
y_pred = log_reg.predict(x_test)

y_pred

# Predicted probablity on x_test

log_reg.predict_proba(x_test)

# Model Accuracy

accuracy = accuracy_score(y_test,y_pred)

print("Accurcy of ML Model-3:",accuracy*100,'%')

# Using Confusion Matrix for checking accuracy of model

confusion_mat = confusion_matrix(y_test,y_pred)
confusion_mat

# Checking accuracy of model with classification report

print(classification_report(y_test,y_pred))

# Plotting ROC Curve

fpr,tpr,thresholds = roc_curve(y_test,y_pred)  # where,fpr = False Positive Rate ,tpr = True positive Rate , Thresholds means how data is changing

plt.plot(fpr,tpr,color ='red',label='ROC')
plt.plot([0,1],[0,1],color='darkblue',linestyle='--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characterstic ROC Curve')
plt.legend()
plt.show()

# By using GridSearchCV will find for best parameters and by same will to hypertuning for model.

from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100,1000],  # Values of C to search
    'penalty': ['l1', 'l2'],  # L1 (Lasso) and L2 (Ridge) regularization
}

# Create a logistic regression model
logistic_regression = LogisticRegression(solver='liblinear')

# Create a grid search object
grid_search = GridSearchCV(logistic_regression, param_grid, cv=5, scoring='accuracy')

# Fitting to model
grid_search.fit(x_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model
best_model = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_model.predict(x_test)

# Getting Scores and classificaion report after hypertuning parameter.

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100,"%")

report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
(761, 10)
(254, 10)
(761,)
(254,)
Accurcy of ML Model-3: 76.37795275590551 %
              precision    recall  f1-score   support

           0       0.66      0.85      0.74       102
           1       0.88      0.70      0.78       152

    accuracy                           0.76       254
   macro avg       0.77      0.78      0.76       254
weighted avg       0.79      0.76      0.77       254
```




```
Best Hyperparameters: {'C': 10, 'penalty': 'l1'}
Accuracy: 76.77165354330708 %
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.86      0.75       102
           1       0.88      0.70      0.78       152

    accuracy                           0.77       254
   macro avg       0.77      0.78      0.77       254
weighted avg       0.79      0.77      0.77       254
```

Evaluation Metrics after hypertuning Parameter on ML Model-3:

Precision: Before tuning: For class 0, precision is 0.66, and for class 1, precision is 0.88. After tuning: For class 0, precision remains the same (0.66), and for class 1, precision is 0.88.

Recall: Before tuning: For class 0, recall is 0.85, and for class 1, recall is 0.70. After tuning: For class 0, recall is 0.86, and for class 1, recall 0.70

F1-Score: Before tuning: The F1-score for class 0 is 0.74, and for class 1, it is 0.78. After tuning: The F1-score for class 0 is 0.75, and for class 1, it decreases slightly to 0.78.

Accuracy before tuning is **76.37%** and after tuning is **76.77%**.

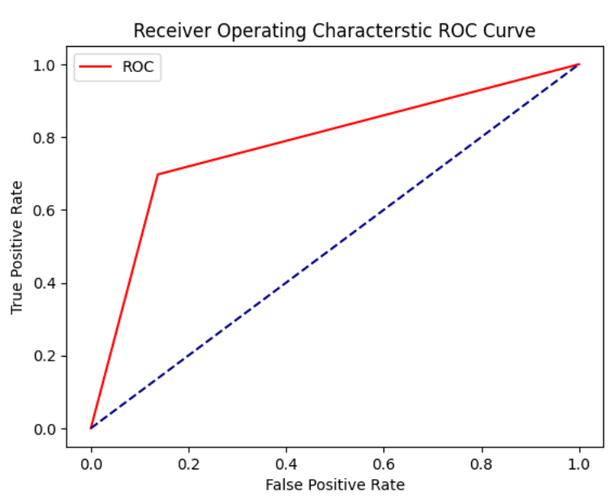
Double-click (or enter) to edit

ML Model - 4

Considering columns after outlier treatment. Cosidering variable with respect to correlation, liner relations.

```
x = no_outliers[['volatile_acidity', 'citric_acid', 'free_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol']]

y = no_outliers['quality']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting Logistic Regression model to dataset

log_reg = LogisticRegression()

log_reg.fit(x_train,y_train)

# will predict on x_train

y_pred_train = log_reg.predict(x_train)

y_pred_train

# prediction on x_test
y_pred = log_reg.predict(x_test)

y_pred

# Predicted probablity on x_test

log_reg.predict_proba(x_test)

# Model Accuracy

accuracy = accuracy_score(y_test,y_pred)

print("Accurcy of ML Model-3:",accuracy*100,'%')

# Using Confusion Matrix for checking accuracy of model

confusion_mat = confusion_matrix(y_test,y_pred)

confusion_mat

# Checking accuracy of model with classification report

print(classification_report(y_test,y_pred))

# Plotting ROC Curve

fpr,tpr,thresholds = roc_curve(y_test,y_pred)  # where,fpr = False Positive Rate ,tpr = True positive Rate , Thresholds means how data is changing

plt.plot(fpr,tpr,color ='red',label='ROC')
plt.plot([0,1],[0,1],color='darkblue',linestyle='--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characterstic ROC Curve')
plt.legend()
plt.show()

# By using GridSearchCV will find for best parameters and by same will to hypertuning for model.

from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100,1000],  # Values of C to search
    'penalty': ['l1', 'l2'],  # L1 (Lasso) and L2 (Ridge) regularization
}

# Create a logistic regression model
logistic_regression = LogisticRegression(solver='liblinear')

# Create a grid search object
grid_search = GridSearchCV(logistic_regression, param_grid, cv=5, scoring='accuracy')

# Fitting to model
grid_search.fit(x_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model
best_model = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_model.predict(x_test)

# Getting Scores and classificaion report after hypertuning parameter.

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100,"%")

report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
(761, 7)
(254, 7)
(761,)
(254,)
Accurcy of ML Model-3: 76.37795275590551 %
              precision    recall  f1-score   support

           0       0.66      0.86      0.75       102
           1       0.88      0.70      0.78       152

    accuracy                           0.76       254
   macro avg       0.77      0.78      0.76       254
weighted avg       0.79      0.76      0.77       254
```




```
Best Hyperparameters: {'C': 10, 'penalty': 'l1'}
Accuracy: 77.16535433070865 %
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.86      0.75       102
           1       0.89      0.71      0.79       152

    accuracy                           0.77       254
   macro avg       0.78      0.79      0.77       254
weighted avg       0.80      0.77      0.77       254
```

Evaluation Metrics after hypertuning Parameter on ML Model-4:

Precision: Before tuning: For class 0, precision is 0.66, and for class 1, precision is 0.88. After tuning: For class 0, precision 0.67, and for class 1, precision is 0.89.

Recall: Before tuning: For class 0, recall is 0.86, and for class 1, recall is 0.70. After tuning: For class 0, recall is 0.86, and for class 1, recall 0.71

F1-Score: Before tuning: The F1-score for class 0 is 0.75, and for class 1, it is 0.78. After tuning: The F1-score for class 0 is 0.75, and for class 1, it decreases slightly to 0.79.

Accuracy before tuning is **76.37%** and after tuning is **77.16%**.

ML Model - 5

Decision Tree Model

```
# Importing decision tree classifier

from sklearn.tree import DecisionTreeClassifier

# Considering variable with above ML model as its giveing better result.

x = no_outliers[['fixed_acidity', 'volatile_acidity', 'citric_acid', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol','Alcohol_content']]

y = no_outliers['quality']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting a Decision Tree model

decision_tree = DecisionTreeClassifier()
decision_tree.fit(x_train, y_train)
```

```
# Make predictions on the training data
y_pred_train = decision_tree.predict(x_train)

# Make predictions on the test data
y_pred = decision_tree.predict(x_test)

# Model Accuracy

accuracy = accuracy_score(y_test,y_pred)

print("Accurcy of Decision Tree :",accuracy*100,'%')

# Using Confusion Matrix for checking accuracy of the model
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(confusion_mat)

report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)

# Defining function for checking results

def metric_score(clf,x_train,x_test,y_train,y_test,train=True):
  if train:
    y_pred = clf.predict(x_train)
    print('\n======Train Result======')
    print(f'Accuracy Score : {accuracy_score(y_train,y_pred)*100:.2f}%')

  elif train==False:
    pred = clf.predict(x_test)
    print('\n======Test Result======')
    print(f'Accuracy Score : {accuracy_score(y_test,pred)*100:.2f}%')

    print('\n\n Test Classification Report \n',classification_report(y_test,pred,digits=2))


# After checking accuracy and classification report of decision tree model will check for hyprtuning for more good result.

grid_param = {
    'criterion': ['gini','entropy'],
    'max_depth': range(10,15),
    'min_samples_leaf': range(2,6),
    'min_samples_split': range(3,8),
    'max_leaf_nodes': range(5,10)
}

grid_search = GridSearchCV(estimator=decision_tree,
                           param_grid = grid_param,
                           cv=5,
                           n_jobs=-1)

grid_search.fit(x_train,y_train)

# Check for best parameters

best_parameters = grid_search.best_params_
print(best_parameters)
```

```
    (761, 10)
    (254, 10)
    (761,)
    (254,)
    Accurcy of Decision Tree : 66.53543307086615 %
    Confusion Matrix:
    [[71 31]
     [54 98]]
    Classification Report:
                  precision    recall  f1-score   support

               0       0.57      0.70      0.63       102
               1       0.76      0.64      0.70       152

        accuracy                           0.67       254
       macro avg       0.66      0.67      0.66       254
    weighted avg       0.68      0.67      0.67       254

    {'criterion': 'gini', 'max_depth': 10, 'max_leaf_nodes': 8, 'min_samples_leaf': 2, 'min_samples_split': 3}
```

```
# using recived best paramerter for further model for getting good result

clf = DecisionTreeClassifier(criterion='gini',max_depth=10,min_samples_leaf=5,min_samples_split=3,max_leaf_nodes= 9)

clf.fit(x_train,y_train)

# will check for result

metric_score(clf,x_train,x_test,y_train,y_test,train=True) # for training score

metric_score(clf,x_train,x_test,y_train,y_test,train=False)  # for testing score
```

```
    Accurcy of Decision Tree : 66.53543307086615 %

    ======Train Result======
    Accuracy Score : 73.46%

    ======Test Result======
    Accuracy Score : 74.41%


     Test Classification Report
                  precision    recall  f1-score   support

               0       0.63      0.88      0.73       102
               1       0.89      0.65      0.75       152

        accuracy                           0.74       254
       macro avg       0.76      0.77      0.74       254
    weighted avg       0.79      0.74      0.75       254
```

Evaluation Metrics for decision tree model:

1. **Precision-** Before tuning: For class 0, precision is 0.57, and for class 1, precision is 0.76. After tuning: For class 0, recision is 0.63, and for class 1, precision is 0.89.
2. **Recall-** Before tuning: For class 0, recall is 0.70, and for class 1, recall is 0.64. After tuning: For class 0, recall is 0.88, and for class 1, recall is 0.65.
3. **F1-Score-** Before tuning: The F1-score for class 0 is 0.63, and for class 1, it is 0.70. After tuning: For class 0, F1-Score is 0.73, and for class 1, F1-Score is 0.75.
4. **Accuracy-** before tuning is **66.53%** and after tuning accuracy of model is **66.53%** while accuracy on train dataset is **73.46%** and on test dataset is **74.41%**.

ML Model - 5

RandomForestClassifier

```
x = no_outliers[['volatile_acidity', 'citric_acid', 'free_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol']]

y = no_outliers['quality']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting RandomForest model, first will import from sklearn package

from sklearn.ensemble import RandomForestClassifier

random_for = RandomForestClassifier()

random_for.fit(x_train,y_train)

# Make predictions on the training data
y_pred_train = random_for.predict(x_train)

# Make predictions on the test data
y_pred = random_for.predict(x_test)

# Model Accuracy

accuracy = accuracy_score(y_test,y_pred)

print("Accurcy of RandomForestClassifier :",accuracy*100,'%')

# Using Confusion Matrix for checking accuracy of the model
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(confusion_mat)

report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)

# Defining function for checking results

def metric_score(clf,x_train,x_test,y_train,y_test,train=True):
  if train:
    y_pred = clf.predict(x_train)
    print('\n======Train Result======')
    print(f'Accuracy Score : {accuracy_score(y_train,y_pred)*100:.2f}%')

  elif train==False:
    pred = clf.predict(x_test)
    print('\n======Test Result======')
    print(f'Accuracy Score : {accuracy_score(y_test,pred)*100:.2f}%')

    print('\n\n Test Classification Report \n',classification_report(y_test,pred,digits=2))


# After checking accuracy and classification report of decision tree model will check for hyprtuning for more good result.

grid_param = {
    'criterion': ['gini','entropy'],
    'max_depth': range(10,15),
    'min_samples_leaf': range(2,6),
    'min_samples_split': range(3,8),
    'max_leaf_nodes': range(5,10)
}

grid_search = GridSearchCV(estimator=random_for,
                           param_grid = grid_param,
                           cv=5,
                           n_jobs=-1)

grid_search.fit(x_train,y_train)

# Check for best parameters

best_parameters = grid_search.best_params_
print(best_parameters)
```

```
(761, 7)
(254, 7)
(761,)
(254,)
Accurcy of RandomForestClassifier : 79.13385826771653 %
Confusion Matrix:
[[ 85  17]
 [ 36 116]]
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.83      0.76       102
           1       0.87      0.76      0.81       152

    accuracy                           0.79       254
   macro avg       0.79      0.80      0.79       254
weighted avg       0.80      0.79      0.79       254

{'criterion': 'gini', 'max_depth': 13, 'max_leaf_nodes': 9, 'min_samples_leaf': 5, 'min_samples_split': 5}
```

```
# using recived best paramerter for further model for getting good result

clf = RandomForestClassifier(criterion='gini',max_depth=13,min_samples_leaf=5,min_samples_split=5,max_leaf_nodes= 9)

clf.fit(x_train,y_train)

# will check for result

metric_score(clf,x_train,x_test,y_train,y_test,train=True) # for training score

metric_score(clf,x_train,x_test,y_train,y_test,train=False)  # for testing score
```

```
======Train Result======
Accuracy Score : 76.61%

======Test Result======
Accuracy Score : 75.59%


 Test Classification Report
              precision    recall  f1-score   support

           0       0.65      0.86      0.74       102
           1       0.88      0.68      0.77       152

    accuracy                           0.76       254
   macro avg       0.76      0.77      0.75       254
weighted avg       0.79      0.76      0.76       254
```

Evaluation Metrics for RandomForestClassifier:

1. **Precision-** Before tuning: For class 0, precision is 0.70, and for class 1, precision is 0.87. After tuning: For class 0, recision is 0.65, and for class 1, precision is 0.88.
2. **Recall-** Before tuning: For class 0, recall is 0.83, and for class 1, recall is 0.76. After tuning: For class 0, recall is 0.86, and for class 1, recall is 0.68.
3. **F1-Score-** Before tuning: The F1-score for class 0 is 0.76, and for class 1, it is 0.81. After tuning: For class 0, F1-Score is 0.74, and for class 1, F1-Score is 0.77.
4. **Accuracy-** before tuning is **79.13%** and after tuning accuracy on train dataset is **76.61%** and on test dataset is **75.59%**.

ML Model - 6

KNeighborsClassifier

```
x = no_outliers[['volatile_acidity', 'citric_acid', 'free_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol']]

y = no_outliers['quality']

# splitting data into train and test set.
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting KNeighborsClassifier model, first will import from sklearn package

from sklearn.neighbors import KNeighborsClassifier

kNN = KNeighborsClassifier()

kNN.fit(x_train,y_train)

# Make predictions on the training data
y_pred_train = kNN.predict(x_train)

# Make predictions on the test data
y_pred = kNN.predict(x_test)

# Model Accuracy

accuracy = accuracy_score(y_test,y_pred)

print("Accurcy of KNeighborsClassifier :",accuracy*100,'%')

# Using Confusion Matrix for checking accuracy of the model
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(confusion_mat)

report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)

# Defining function for checking results

def metric_score(clf,x_train,x_test,y_train,y_test,train=True):
  if train:
    y_pred = clf.predict(x_train)
    print('\n======Train Result======')
    print(f'Accuracy Score : {accuracy_score(y_train,y_pred)*100:.2f}%')

  elif train==False:
    pred = clf.predict(x_test)
    print('\n======Test Result======')
    print(f'Accuracy Score : {accuracy_score(y_test,pred)*100:.2f}%')

    print('\n\n Test Classification Report \n',classification_report(y_test,pred,digits=2))


# After checking accuracy and classification report of decision tree model will check for hyprtuning for more good result.

grid_param = {
    'n_neighbors': range(1, 21),
    'weights': ['uniform', 'distance'],
    'p': [1, 2],
}

grid_search = GridSearchCV(estimator=kNN,
                           param_grid = grid_param,
                           cv=5,
                           n_jobs=-1)

grid_search.fit(x_train,y_train)

# Check for best parameters

best_parameters = grid_search.best_params_
print(best_parameters)
```

```
    (761, 7)
    (254, 7)
    (761,)
    (254,)
    Accurcy of RandomForestClassifier : 72.44094488188976 %
    Confusion Matrix:
    [[ 72  30]
     [ 40 112]]
    Classification Report:
                  precision    recall  f1-score   support

               0       0.64      0.71      0.67       102
               1       0.79      0.74      0.76       152

        accuracy                           0.72       254
       macro avg       0.72      0.72      0.72       254
    weighted avg       0.73      0.72      0.73       254

    {'n_neighbors': 7, 'p': 1, 'weights': 'distance'}
```

```
# using recived best paramerter for further model for getting good result

kNN_best = KNeighborsClassifier(**best_parameters)

kNN_best.fit(x_train,y_train)

# will check for result

metric_score(kNN_best,x_train,x_test,y_train,y_test,train=True) # for training score

metric_score(kNN_best,x_train,x_test,y_train,y_test,train=False)  # for testing score
```

```
    ======Train Result======
    Accuracy Score : 100.00%

    ======Test Result======
    Accuracy Score : 77.95%


     Test Classification Report
                  precision    recall  f1-score   support

               0       0.69      0.83      0.75       102
               1       0.87      0.74      0.80       152

        accuracy                           0.78       254
       macro avg       0.78      0.79      0.78       254
    weighted avg       0.80      0.78      0.78       254
```

Evaluation Metrics for KNeighborsClassifier:

1. **Precision-** Before tuning: For class 0, precision is 0.64, and for class 1, precision is 0.79. After tuning: For class 0, recision is 0.69, and for class 1, precision is 0.87.
2. **Recall-** Before tuning: For class 0, recall is 0.71, and for class 1, recall is 0.74. After tuning: For class 0, recall is 0.83, and for class 1, recall is 0.74.
3. **F1-Score-** Before tuning: The F1-score for class 0 is 0.67, and for class 1, it is 0.76. After tuning: For class 0, F1-Score is 0.75, and for class 1, F1-Score is 0.80.
4. **Accuracy-** before tuning is **72.44%** and after tuning accuracy on train dataset is **100%** and on test dataset is **77.95%**.

Seems like KNeighborsClassifier model overfitting after tuning on train dataset.

ML Model - 6

SVM classifier

```
x = no_outliers[['volatile_acidity', 'citric_acid', 'free_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol']]

y = no_outliers['quality']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
```

```
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting SVM model, first will import from sklearn package

from sklearn.svm import SVC

svm_classifier = SVC()

svm_classifier.fit(x_train,y_train)

# Make predictions on the training data
y_pred_train = svm_classifier.predict(x_train)

# Make predictions on the test data
y_pred = svm_classifier.predict(x_test)

# Model Accuracy

accuracy = accuracy_score(y_test,y_pred)

print("Accurcy of SVM Model :",accuracy*100,'%')

# Using Confusion Matrix for checking accuracy of the model
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(confusion_mat)

report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)

# Defining function for checking results

def metric_score(clf,x_train,x_test,y_train,y_test,train=True):
  if train:
    y_pred = clf.predict(x_train)
    print('\n======Train Result======')
    print(f'Accuracy Score : {accuracy_score(y_train,y_pred)*100:.2f}%')
  elif train==False:
    pred = clf.predict(x_test)
    print('\n======Test Result======')
    print(f'Accuracy Score : {accuracy_score(y_test,pred)*100:.2f}%')

    print('\n\n Test Classification Report \n',classification_report(y_test,pred,digits=2))


# After checking accuracy and classification report of decision tree model will check for hyprtuning for more good result.

grid_param = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': [0.001, 0.01, 0.1],
}

grid_search = GridSearchCV(estimator=svm_classifier,
                           param_grid = grid_param,
                           cv=5,
                           n_jobs=-1)

grid_search.fit(x_train,y_train)

# Check for best parameters

best_parameters = grid_search.best_params_
print(best_parameters)
```

```
(761, 7)
(254, 7)
(761,)
(254,)
Accurcy of SVM Model : 75.59055118110236 %
Confusion Matrix:
[[ 86  16]
 [ 46 106]]
Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.84      0.74       102
           1       0.87      0.70      0.77       152

    accuracy                           0.76       254
   macro avg       0.76      0.77      0.75       254
weighted avg       0.78      0.76      0.76       254

{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
```

```
# using recived best paramerter for further model for getting good result

svm_classifier_best = SVC(**best_parameters)

svm_classifier_best.fit(x_train,y_train)

# will check for result

metric_score(svm_classifier_best,x_train,x_test,y_train,y_test,train=True) # for training score

metric_score(svm_classifier_best,x_train,x_test,y_train,y_test,train=False)  # for testing score
```

```
======Train Result======
Accuracy Score : 72.01%

======Test Result======
Accuracy Score : 75.59%


 Test Classification Report
              precision    recall  f1-score   support

           0       0.65      0.86      0.74       102
           1       0.88      0.68      0.77       152

    accuracy                           0.76       254
   macro avg       0.76      0.77      0.75       254
weighted avg       0.79      0.76      0.76       254
```

Evaluation Metrics for SVC Model:

1. **Precision-** Before tuning: For class 0, precision is 0.65, and for class 1, precision is 0.87. After tuning: For class 0, recision is 0.65, and for class 1, precision is 0.88.
2. **Recall-** Before tuning: For class 0, recall is 0.84, and for class 1, recall is 0.70. After tuning: For class 0, recall is 0.86, and for class 1, recall is 0.68.
3. **F1-Score-** Before tuning: The F1-score for class 0 is 0.74, and for class 1, it is 0.77. After tuning: For class 0, F1-Score is 0.74, and for class 1, F1-Score is 0.77.
4. **Accuracy-** before tuning is **75.59%** and after tuning accuracy on train dataset is **72.01%** and on test dataset is **75.59%**.

Conclusion

- Wines with quality ratings of 5 and 6 appear to have similar fixed acidity levels.
- Higher quality wines generally have higher fixed acidity levels, which suggests that wines with higher fixed acidity might be preferred by consumers.
- Quality ratings 7 and 8 have notably lower levels of volatile acidity compared to lower quality ratings. This suggests that wines with these higher quality ratings have a more balanced and pleasant acidity level.
- Quality ratings 3 and 4 have the highest average levels of volatile acidity, indicating that wines in these quality categories tend to have higher volatile acid content.
- Higher-quality wines tend to have higher average levels of citric acid,which indicat that citric acid may contribute positively to the get good quality of red wines.
- Quality ratings 7 and 8 have higher citric acid levels, while Quality ratings 3 and 4 have lower citric acid levels.This indicating that wines with higher quality ratings tend to have more citric acid and wines with lower quality ratings tend to have less citric acid.
- The line plot indicates that there is the average level of residual sugar tends to increase as wine quality improves. This means that wines with higher quality ratings tend to have slightly higher levels of residual sugar. Quality ratings 7 and 8 show an increase in residual sugar levels compared to quality ratings 5 and 6.
- The graph indicates a trend where lower-quality wines tend to have higher levels of chlorides.Chloride content is inversely related to wine quality, with higher-quality wines having lower chloride levels.
- The graph implies that sulphates are positively correlated with wine quality. Higher-quality wines are more likely to have a higher sulphate content.
- Quality ratings 5 and 6 have slightly higher levels of free sulfur dioxide compared to other quality ratings. This suggests that wines in these quality categories may require higher levels of free sulfur dioxide.

- Quality ratings 5 have slightly higher levels of total_sulfur_dioxide compared to other quality ratings. This suggests that wines in these quality categories may require higher levels of total_sulfur_dioxide.
- From the density distribution, we can observe that the density is centered around a mean of 0.997.
- From the pH distribution, we can observe that the pH is distributed with a mean of 3.3.
- The graph indicate a positive correlation between alcohol level and wine quality. Higher-quality wines tend to have higher level of alcohol content,while lower-quality wines have lower level of alcohol content
- Majority of wines fall into the "Medium" alcohol content category(52.2%).The "Low" and "High" alcohol content categories are less common.(18.6% and 29.2% respectively).
- Quality ratings 5 and 6 are the most common in the dataset, with quality rating 5 having a slightly higher count than quality rating 6
- We can observe that relation and distribution of each varibale, we have have positive correlation for 'density' and 'residual_sugar' , 'free_sulfur_dioxide' and 'total_sulfur_dioxide' , 'sulphates' and 'alcohol'.
- Positive Correlation: 'alcohol' and 'Alcohol_content' ,'citric_acid' and 'fixed_acidity', 'free_sulfur_dioxide' and 'total_sulfur_dioxide', 'density' and 'fixed_acidity' this vearibal showing positive corrlation.

Hypothetical Statements and result.

1. The fixed acidity of wines with a quality rating greater than or equal to 6 is significantly different from the fixed acidity of wines with a quality rating less than 6.

   Ho : There is a significant difference in mean fixed acidity.

   H1 : There is no significant difference in mean fixed acidity.

   Result : Reject the null hypothesis: There is a significant difference in mean fixed acidity.

2. There is a correlation between the levels of citric acid and sulphates in red wines.

   Ho : There is a significant correlation between citric acid and sulphates.

   H1 : There is no significant correlation between citric acid and sulphates.

   Result :There is a significant correlation between citric acid and sulphates.

3. The distribution of alcohol content in red wines with a quality rating of 5 is significantly different from the distribution of alcohol content in red wines with a quality rating of 6.

   Ho : The distributions are significantly different.

   H1 : The distributions are not significantly different.

   Result : Reject the null hypothesis: The distributions are significantly different.

Evaluation Metrics for Model 1 before tuning:

1. **Accuracy Of ML Model-1 - 70%.**
2. **For class 0: Precision is 0.63, Recall is 0.85, and F1-score is 0.72.**
3. **For class 1: Precision is 0.82, Recall is 0.57, and F1-score is 0.67.**

Evaluation Metrics after hypertuning Parameter on ML Model-1:

1. **Accuracy Of ML Model-1 - 69.5%.**
2. **For class 0: Precision is 0.62, Recall is 0.88, and F1-score is 0.73.**
3. **For class 1: Precision is 0.84, Recall is 0.53, and F1-score is 0.65.**

For ML Model-2, we can observe that there is no significant change in the model's accuracy before and after tuning hyperparameters. This may be because we dropped only one variable from the previous model.

Evaluation Metrics after hypertuning Parameter on ML Model-3:

1. **Precision:** Before tuning: For class 0, precision is 0.66, and for class 1, precision is 0.88. After tuning: For class 0, precision remains the same (0.66), and for class 1, precision is 0.88.
2. **Recall:** Before tuning: For class 0, recall is 0.85, and for class 1, recall is 0.70. After tuning: For class 0, recall is 0.86, and for class 1, recall 0.70
3. **F1-Score:** Before tuning: The F1-score for class 0 is 0.74, and for class 1, it is 0.78. After tuning: The F1-score for class 0 is 0.75, and for class 1, it decreases slightly to 0.78.
4. **Accuracy** before tuning is **76.37%** and after tuning is **76.77%**.

Evaluation Metrics after hypertuning Parameter on ML Model-4:

1. **Precision:** Before tuning: For class 0, precision is 0.66, and for class 1, precision is 0.88. After tuning: For class 0, precision 0.67, and for class 1, precision is 0.89.
2. **Recall:** Before tuning: For class 0, recall is 0.86, and for class 1, recall is 0.70. After tuning: For class 0, recall is 0.86, and for class 1, recall 0.71
3. **F1-Score:** Before tuning: The F1-score for class 0 is 0.75, and for class 1, it is 0.78. After tuning: The F1-score for class 0 is 0.75, and for class 1, it decreases slightly to 0.79.
4. **Accuracy** before tuning is **76.37%** and after tuning is **77.16%**.

Evaluation Metrics for decision tree model:

1. **Precision-** Before tuning: For class 0, precision is 0.57, and for class 1, precision is 0.76. After tuning: For class 0, recision is 0.63, and for class 1, precision is 0.89.
2. **Recall-** Before tuning: For class 0, recall is 0.70, and for class 1, recall is 0.64. After tuning: For class 0, recall is 0.88, and for class 1, recall is 0.65.
3. **F1-Score-** Before tuning: The F1-score for class 0 is 0.63, and for class 1, it is 0.70. After tuning: For class 0, F1-Score is 0.73, and for class 1, F1-Score is 0.75.
4. **Accuracy-** before tuning is **66.53%** and after tuning accuracy of model is **66.53%** while accuracy on train dataset is **73.46%** and on test dataset is **74.41%**.

Evaluation Metrics for RandomForestClassifier:

1. **Precision-** Before tuning: For class 0, precision is 0.70, and for class 1, precision is 0.87. After tuning: For class 0, recision is 0.65, and for class 1, precision is 0.88.
2. **Recall-** Before tuning: For class 0, recall is 0.83, and for class 1, recall is 0.76. After tuning: For class 0, recall is 0.86, and for class 1, recall is 0.68.
3. **F1-Score-** Before tuning: The F1-score for class 0 is 0.76, and for class 1, it is 0.81. After tuning: For class 0, F1-Score is 0.74, and for class 1, F1-Score is 0.77.
4. **Accuracy-** before tuning is **79.13%** and after tuning accuracy on train dataset is **76.61%** and on test dataset is **75.59%**.

Evaluation Metrics for KNeighborsClassifier:

1. **Precision-** Before tuning: For class 0, precision is 0.64, and for class 1, precision is 0.79. After tuning: For class 0, recision is 0.69, and for class 1, precision is 0.87.
2. **Recall-** Before tuning: For class 0, recall is 0.71, and for class 1, recall is 0.74. After tuning: For class 0, recall is 0.83, and for class 1, recall is 0.74.
3. **F1-Score-** Before tuning: The F1-score for class 0 is 0.67, and for class 1, it is 0.76. After tuning: For class 0, F1-Score is 0.75, and for class 1, F1-Score is 0.80.
4. **Accuracy-** before tuning is **72.44%** and after tuning accuracy on train dataset is **100%** and on test dataset is **77.95%**.

Seems like KNeighborsClassifier model overfitting after tuning on train dataset.

Evaluation Metrics for SVC Model:

1. **Precision-** Before tuning: For class 0, precision is 0.65, and for class 1, precision is 0.87. After tuning: For class 0, recision is 0.65, and for class 1, precision is 0.88.
2. **Recall-** Before tuning: For class 0, recall is 0.84, and for class 1, recall is 0.70. After tuning: For class 0, recall is 0.86, and for class 1, recall is 0.68.
3. **F1-Score-** Before tuning: The F1-score for class 0 is 0.74, and for class 1, it is 0.77. After tuning: For class 0, F1-Score is 0.74, and for class 1, F1-Score is 0.77.
4. **Accuracy-** before tuning is **75.59%** and after tuning accuracy on train dataset is **72.01%** and on test dataset is **75.59%**.

From all the models above, we can observe that the **RandomForestClassifier** model consistently provides the best results both before and after tuning. Additionally, when evaluating metrics such as precision, recall, and F1-score for both class 0 and 1, it outperforms other models

Thank-You