✨ Red Wine Quality Prediction ✨

Author: Kumod Sharma .📄🖋

🎬 Introduction 🎬

📝 Project Objective:

1. Develop a **predictive model** using machine learning algorithms to accurately assess and **predict the quality of red wines** based on various chemical properties and attributes.
2. Evaluate and **compare the performance of different machine learning techniques** to determine the most effective approach for red wine quality prediction, providing insights for potential applications in the wine industry.
3. **Dataset Link:-** [Click to get the Dataset (https://www.kaggle.com/datasets/harmeetsingh07/exshowroom-price)](https://www.kaggle.com/datasets/harmeetsingh07/exshowroom-price)

🌟 Business Understanding:

1. **Enhanced Product Quality:** Accurate red wine quality prediction will lead to improved product quality and consistency, enhancing the winery's reputation and customer satisfaction.
2. **Cost Optimization:** Optimal resource allocation and reduced wastage through predictive modeling will result in cost savings for wineries, improving overall operational efficiency.

3. **Market Competitiveness:** Consistent production of high-quality red wines will give wineries a competitive advantage, allowing them to stand out in the market and attract more customers.

⚙️ Project Content ⚙️

📊 Table of Contents:

📚 Importing Libraries 📚

In [1]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
sns.set(style="darkgrid",font_scale=1.5)
sns.set(rc={"axes.facecolor":"#FFFAF0","figure.facecolor":"#FFFAF0"})
sns.set_context("poster",font_scale = .7)

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier, Stac
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from scipy import stats
from scipy import special
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier


from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score

from imblearn.over_sampling import SMOTE
```

⏳ Loading Datset ⏳

In [2]:
```python
df = pd.read_csv("C://Users//kumod sharma//Desktop//winequality-red.csv")
```

In [3]:
```python
df.head().style.set_properties(**{'background-color': '#E9F6E2','color': 'black','border-color': '#8b8c8c
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.400000	0.700000	0.000000	1.900000	0.076000	11.000000	34.000000	0.997800	3.510000	0.560000	9.400000	5
1	7.800000	0.880000	0.000000	2.600000	0.098000	25.000000	67.000000	0.996800	3.200000	0.680000	9.800000	5
2	7.800000	0.760000	0.040000	2.300000	0.092000	15.000000	54.000000	0.997000	3.260000	0.650000	9.800000	5
3	11.200000	0.280000	0.560000	1.900000	0.075000	17.000000	60.000000	0.998000	3.160000	0.580000	9.800000	6
4	7.400000	0.700000	0.000000	1.900000	0.076000	11.000000	34.000000	0.997800	3.510000	0.560000	9.400000	5

🧠 Basic Understanding of Data 🧠

1. Cheking Dimension of Dataset.

In [4]:
```python
df.shape
```

Out[4]: (1599, 12)

📊 Inference:

- There are total **1599 Records/Rows** in the dataset.

- There are total **12 Features/columns** in the dataset.

2. Generating Basic Information about Data.

In [5]:

```python
df.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Columns: 12 entries, fixed acidity to quality
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

📊 Inference:

- All the **features** present is in the dataset is **numerical.**
- No **Categorical Features** present in the dataset.

3. Computing Total No. of Missing Values & Percentage of Missing values.

In [6]:

```python
null_df = df.isnull().sum().to_frame().rename(columns={0:"Total No. of Missing Values"})
null_df["% of Missing values"] = round(100*null_df["Total No. of Missing Values"]/len(df),2)
null_df.sort_values(by="% of Missing values",ascending=False)
```

Out[6]:

	Total No. of Missing Values	% of Missing values
fixed acidity	0	0.0
volatile acidity	0	0.0
citric acid	0	0.0
residual sugar	0	0.0
chlorides	0	0.0
free sulfur dioxide	0	0.0
total sulfur dioxide	0	0.0
density	0	0.0
pH	0	0.0
sulphates	0	0.0
alcohol	0	0.0
quality	0	0.0

📊 Inference:

- **None of the features** is hvaing **missing values.**
- So we can say the dataset will be more reliable for prediction wine quality.

4. Checking Presence of Duplicate Records in Dataset.

In [7]:
```
print("Is there any Duplicate Records => ",df.duplicated().any())
print("-"*42)
print("Total Duplicate Records present is =>",df[df.duplicated()==True].shape[0])
```

```
Is there any Duplicate Records =>  True
------------------------------------------
Total Duplicate Records present is => 240
```

💬 Inference:

- The **first output** is **True** which indicates that is **presenece of Duplicate Records.**
- The **second output** is **240** which indicates that there is total **240 Duplicate Records.**
- Duplicate vlaues can lead to **Data Integrity issues** so it's better to **drop these records.**

---

5. Dropping Duplicate Records.

In [8]:
```
df.drop_duplicates(inplace=True)
```

---

6. Performing Descriptive Statistical Analysis on Numerical Features.

In [9]:
```
df.describe().T.style.set_properties(**{'background-color': '#E9F6E2','color': 'black','border-color': '#
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
fixed acidity	1359.000000	8.310596	1.736990	4.600000	7.100000	7.900000	9.200000	15.900000
volatile acidity	1359.000000	0.529478	0.183031	0.120000	0.390000	0.520000	0.640000	1.580000
citric acid	1359.000000	0.272333	0.195537	0.000000	0.090000	0.260000	0.430000	1.000000
residual sugar	1359.000000	2.523400	1.352314	0.900000	1.900000	2.200000	2.600000	15.500000
chlorides	1359.000000	0.088124	0.049377	0.012000	0.070000	0.079000	0.091000	0.611000
free sulfur dioxide	1359.000000	15.893304	10.447270	1.000000	7.000000	14.000000	21.000000	72.000000
total sulfur dioxide	1359.000000	46.825975	33.408946	6.000000	22.000000	38.000000	63.000000	289.000000
density	1359.000000	0.996709	0.001869	0.990070	0.995600	0.996700	0.997820	1.003690
pH	1359.000000	3.309787	0.155036	2.740000	3.210000	3.310000	3.400000	4.010000
sulphates	1359.000000	0.658705	0.170667	0.330000	0.550000	0.620000	0.730000	2.000000
alcohol	1359.000000	10.432315	1.082065	8.400000	9.500000	10.200000	11.100000	14.900000
quality	1359.000000	5.623252	0.823578	3.000000	5.000000	6.000000	6.000000	8.000000

💬 Inference: ¶

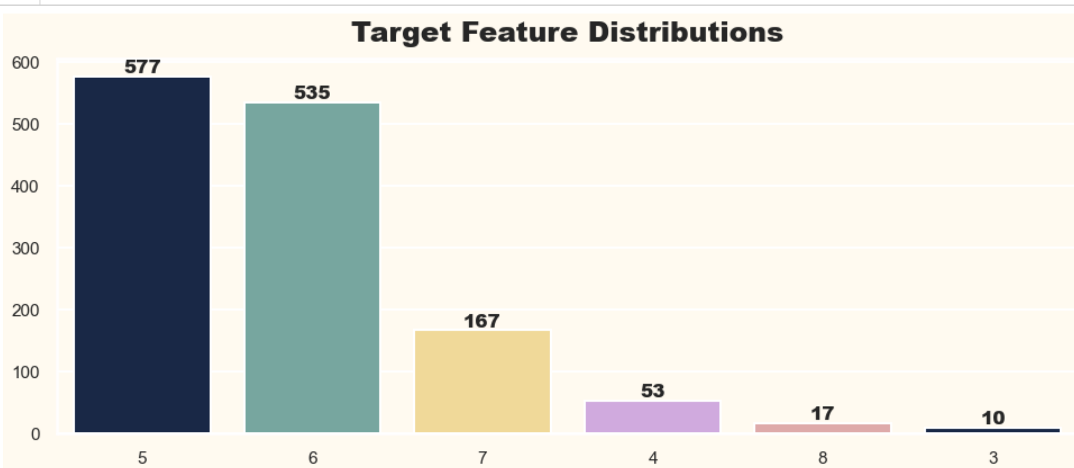
1. The **minimum wine quality is 3** and the **maximum wine quality is 8.**
2. The **average alchol** a red wine holds according to the data is **10.4**.
3. There a **huge difference** between **average total sulfur dioxide** and **maximum total sulfur dioxide**.

📊 Exploratory Data Analysis.📊

1. Visualizing the Target Variable

In [10]:

```
plt.figure(figsize=(13.7,6))
z = df["quality"].value_counts()
sns.barplot(x=z.index, y=z.values, order=z.index, palette=["#11264e","#6faea4","#FEE08B","#D4A1E7","#E7A1
plt.title("Target Feature Distributions",fontweight="black",size=25,pad=15)
for index,value in enumerate(z.values):
    plt.text(index,value,value, ha="center", va="bottom",fontweight="black")

plt.tight_layout()
plt.show()
```



💬 Inference:

- Most of the wines are having quality of **5 or 6**.
- We can clearly observe a **class-imbalance** in the target feature.
- To **overcome** the class-imbalance we can use techniques like:-

    1. **`SMOTE (synthetic minority oversampling technique).`**
    2. **`Stratified K-Folds cross-validation.`**

- **Note:** In this project I have use **Stratified K-Fold Cross-Validaton.**

---

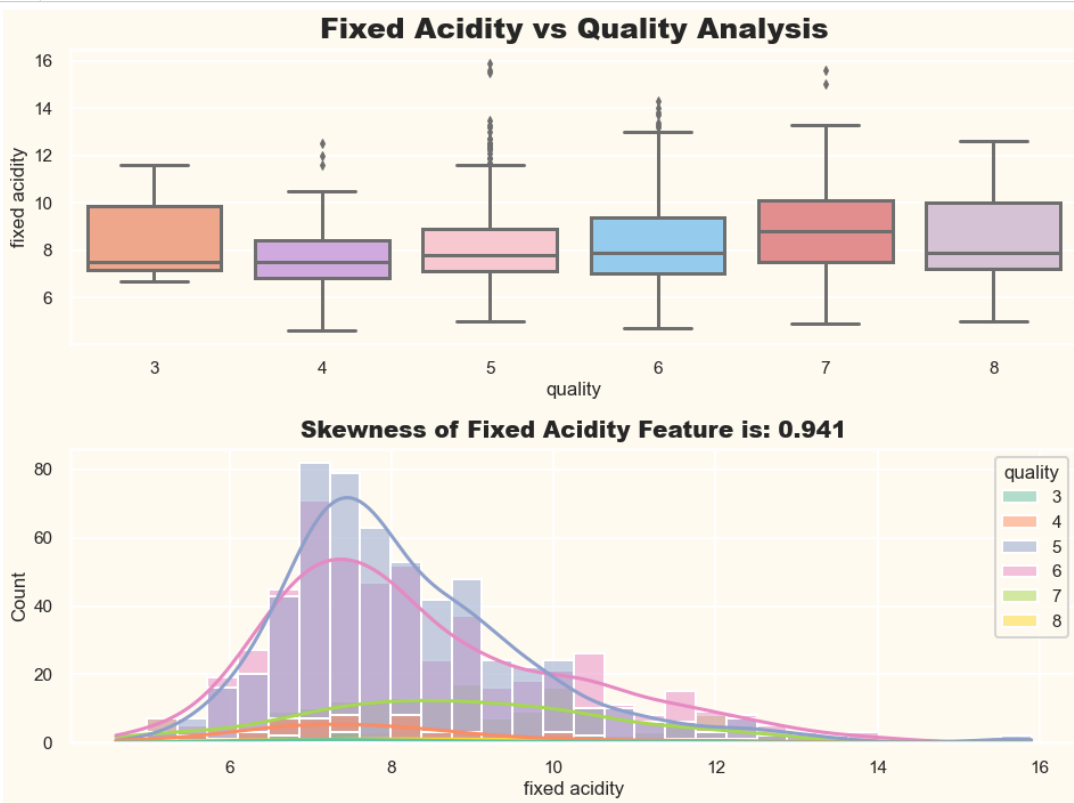
2. Visualizing "Fixed Acidity" Attribute.

In [11]:

```
def numerical_plot(column):
    plt.figure(figsize=(13.5,10))
    plt.subplot(2,1,1)
    sns.boxplot(x="quality",y=column, data=df, palette=["#FFA07A","#D4A1E7","#FFC0CB","#87CEFA","#F08080"
    plt.title(f"{column.title()} vs Quality Analysis",fontweight="black",size=25,pad=10,)

    plt.subplot(2,1,2)
    sns.histplot(x=column,kde=True,hue="quality",data=df, palette="Set2")
    skew = df[column].skew()
    plt.title(f"Skewness of {column.title()} Feature is: {round(skew,3)}",fontweight="black",size=20,pad=
    plt.tight_layout()
    plt.show()
```

In [12]:

```
numerical_plot("fixed acidity")
```



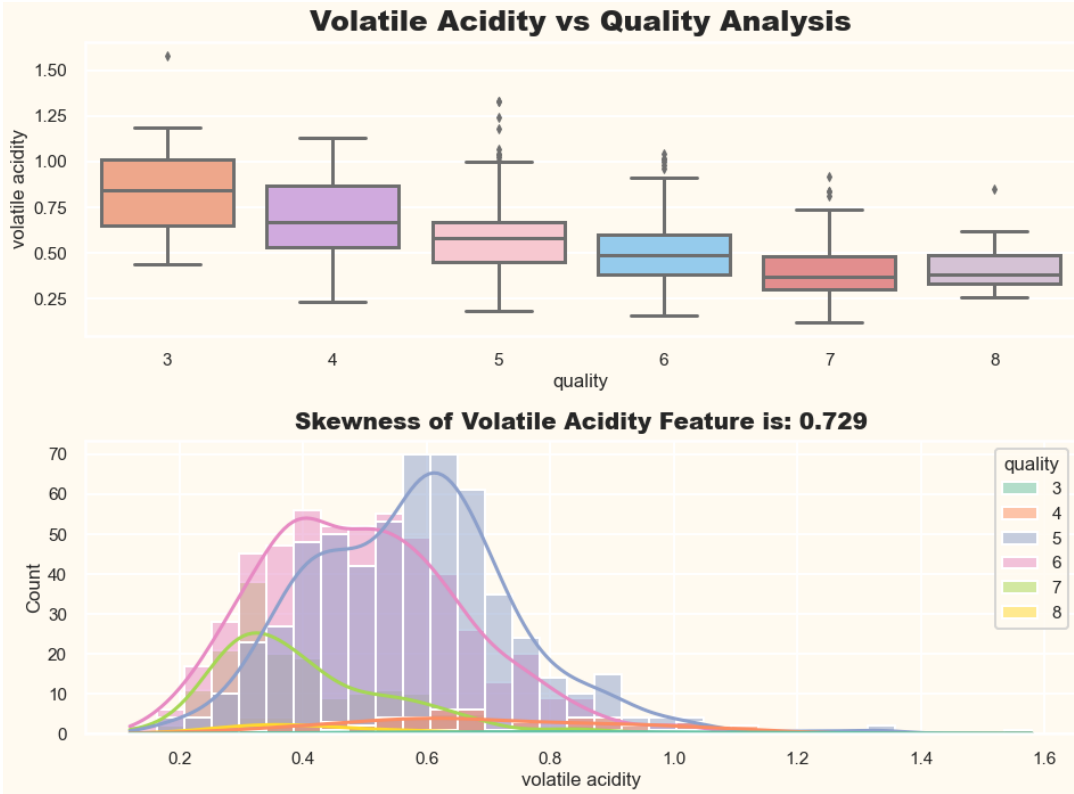
💬 Inference:

- The feature **fixed acidity** is having **almost a symmetric distribution** but the distribution is **little right skewed** with a skewness value of **0.941**.
- Skewness can lead to several implications like **model performance**, **hypothesis testing** and it returns **biased estimation.**
- So we will use **tranformation techniques** to transform thesse feature to have a **symmetric distribution.**

---

3. Visualizing "Volatile Acidity" Attribute.

In [13]:
```
numerical_plot("volatile acidity")
```

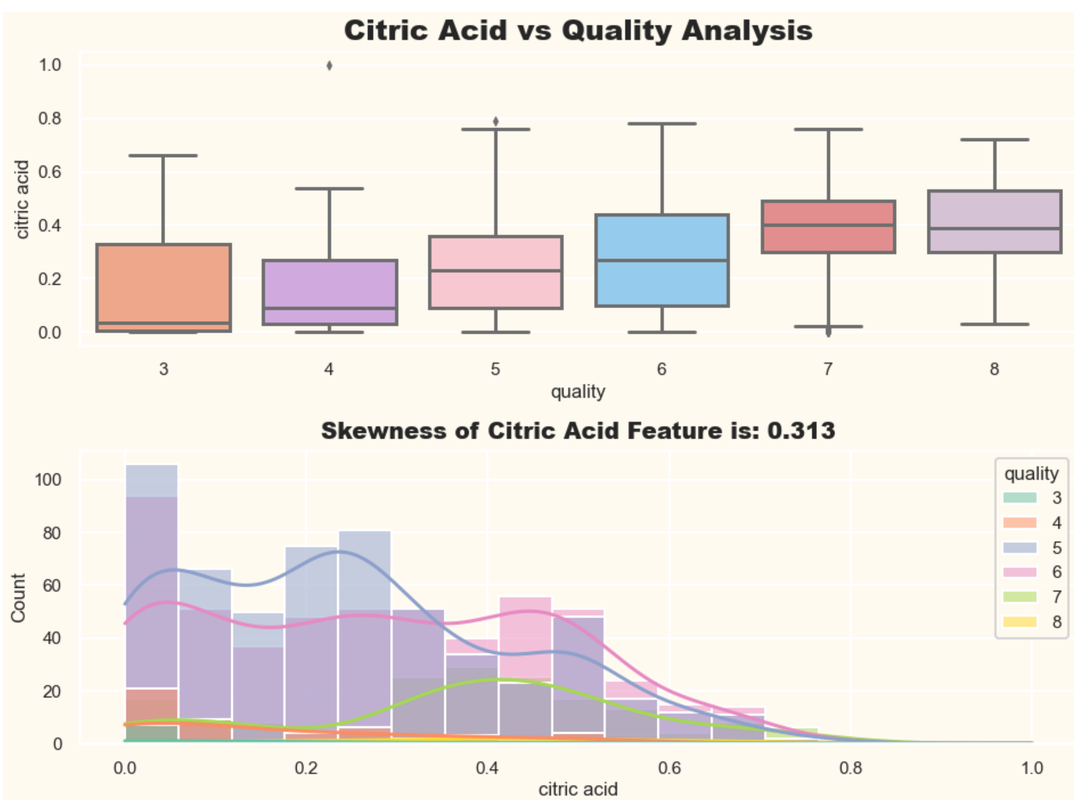


Inference:

- The feature **Volatile Acidity** is having **almost a symmetric distribution** but the distribution is **right skewed** with a skewness value of **0.729**..
- Skewness can lead to several implications like **model performance**, **hypothesis testing** and it returns **biased estimation.**
- So we will use **tranformation techniques** to transform thesse feature to have a **symmetric distribution.**

4. Visualizing "Citric Acid" Attribute.

In [14]:
```
numerical_plot("citric acid")
```



Inference:

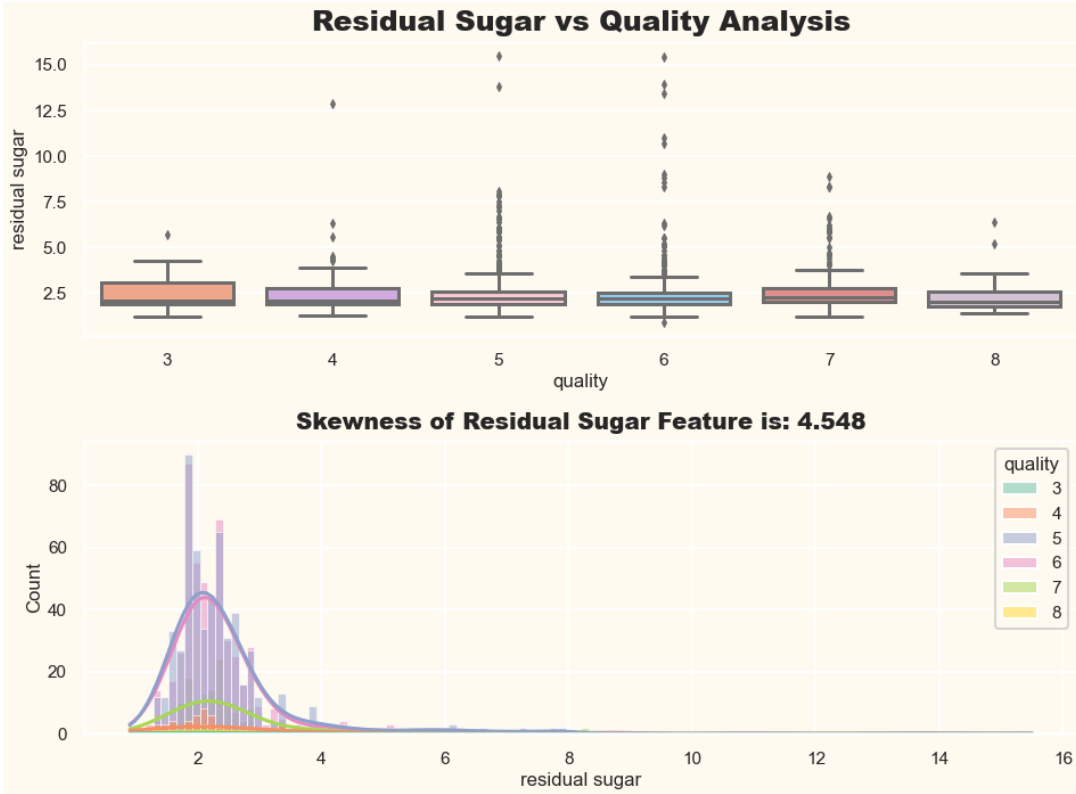
- The feature **Citric Acid** is having a distribution of **right skewed** with a skewness value of **0.313**.

- Although the **skewness is low** but still we try to bring the **skewness close to 0.**
- So we will use **tranformation techniques** to transform thesse feature to have a **symmetric distribution.**

---

5. Visualizing "Residual Sugar" Attribute.

In [15]:
```
numerical_plot("residual sugar")
```

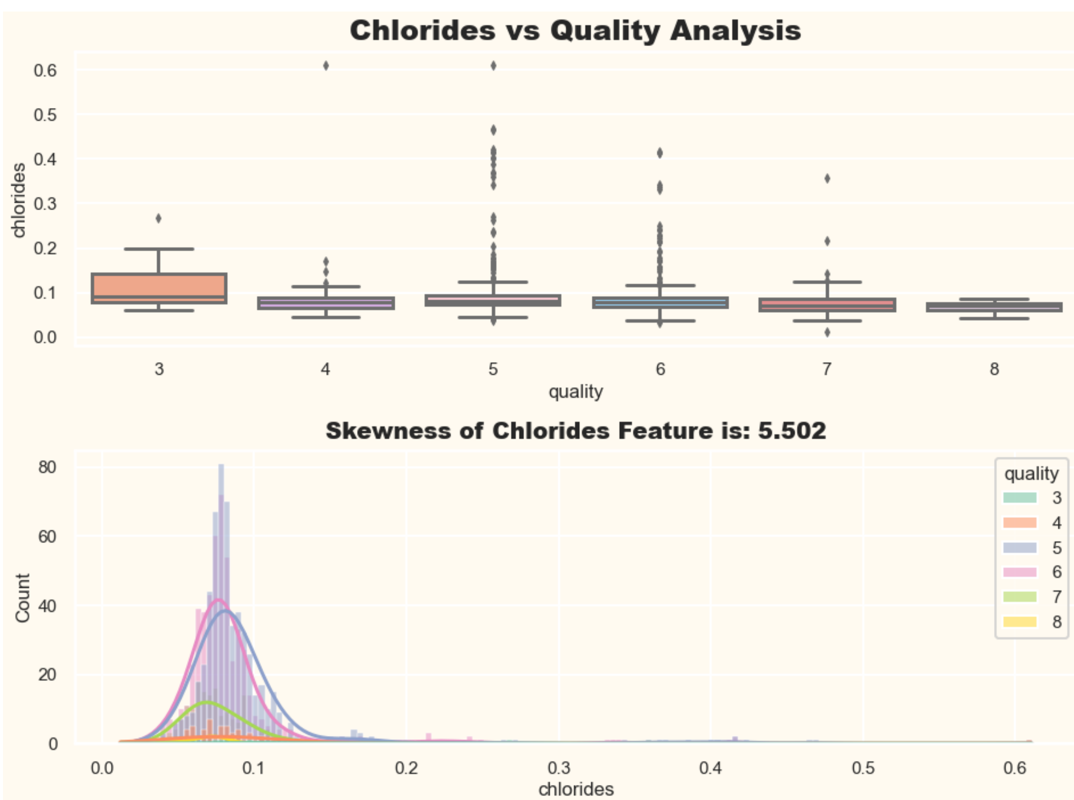


Inference:

- The feature **Residual Sugar** is having **almost a symmetric distribution** but the distribution is ** highly right skewed** with a skewness value of **4.548**.
- The distrbution is **highly right skewed** because of presence of **outliers.**
- So we will use **tranformation techniques** to transform thesse feature to have a **symmetric distribution** and bring **skewness close to 0.**

6. Visualizing "Chlorides" Attribute.

In [16]:
```
numerical_plot("chlorides")
```



💬 Inference:

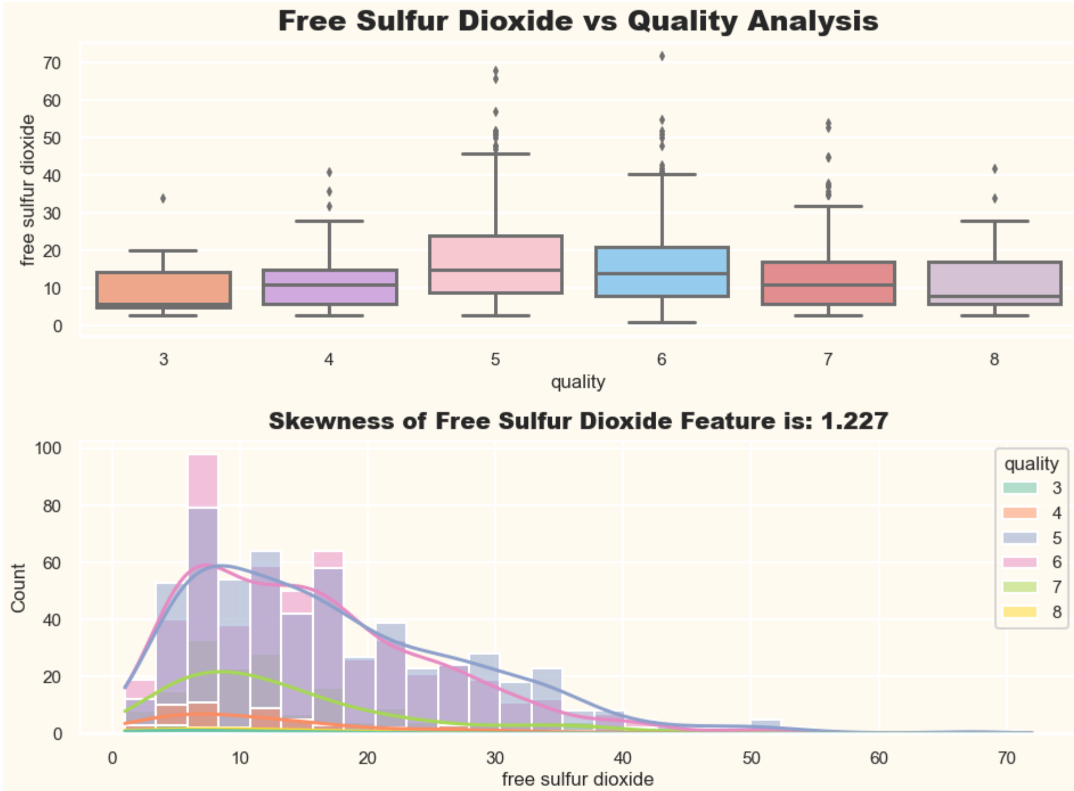
- The feature **Chlorides** is having **almost a symmetric distribution** but the distribution is **highly right skewed** with a skewness value of **5.502**..
- The distrbution is **highly right skewed** because of presence of **outliers.**
- So we will use **tranformation techniques** to transform thesse feature to have a **symmetric distribution** and bring **skewness close to 0.**

---

7. Visualizing "Free Sulfur Dioxide" Attribute.

In [17]:

```
numerical_plot("free sulfur dioxide")
```

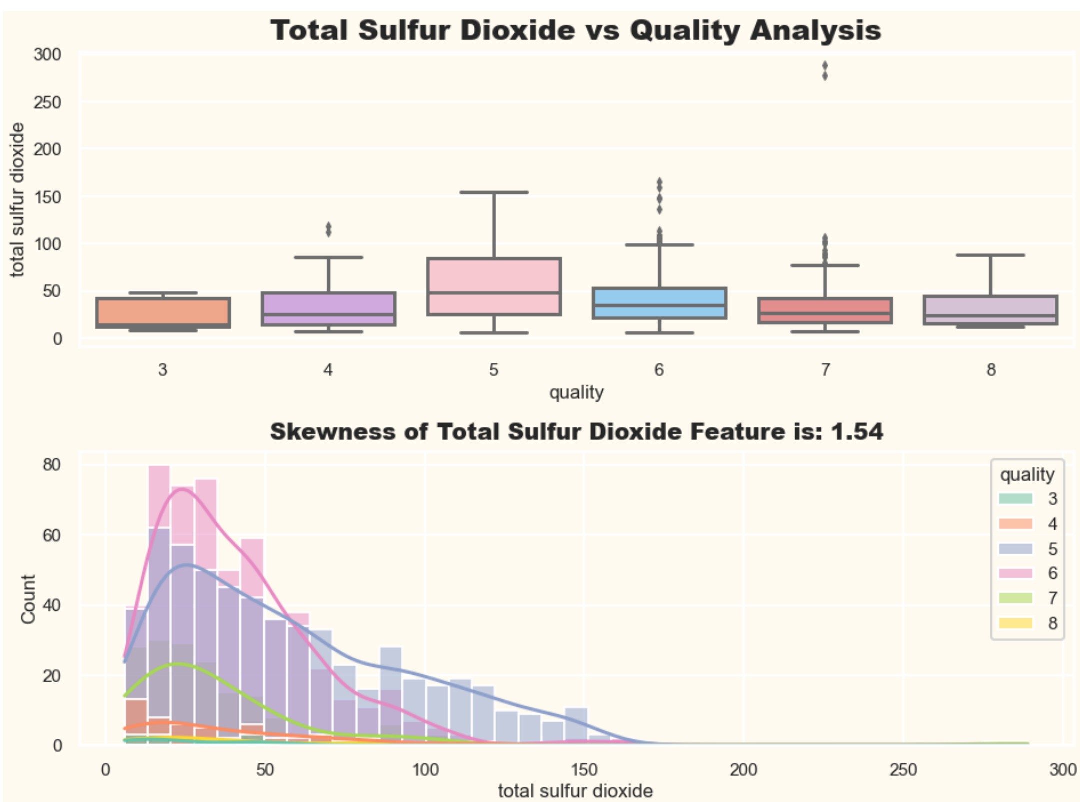


💬 **Inference:**

- The feature **Free Sulfur Dioxide** is having a **highly right skewed distribution** with a skewness value of **1.227**..
- The distrbution is **highly right skewed** because of presence of **outliers.**
- So we will use **tranformation techniques** to transform thesse feature to have a **symmetric distribution** and bring **skewness close to 0.**

8. Visualizing "Total Sulfur Dioxide" Attribute.

In [18]:
```
numerical_plot("total sulfur dioxide")
```



Inference:

- The feature **Total Sulfur Dioxide** is having a **highly right skewed distribution** with a skewness value of **1.54**.

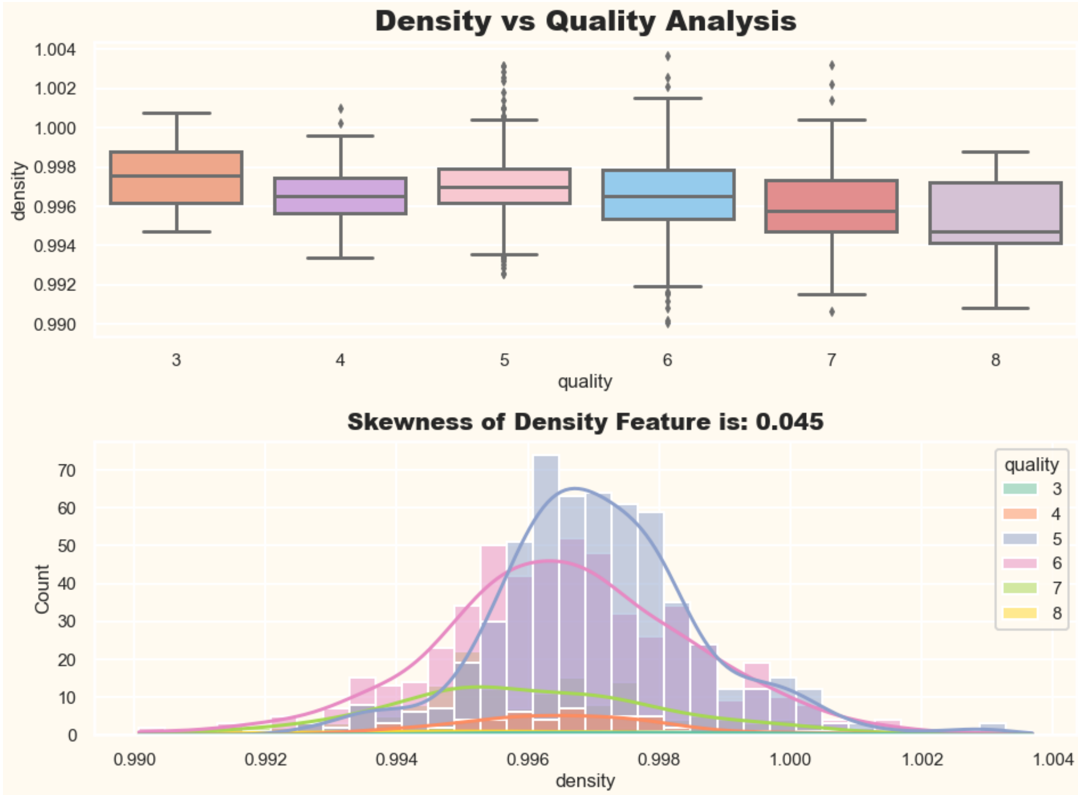
- The distrbution is **highly right skewed** because of presence of **outliers.**
- So we will use **tranformation techniques** to transform thesse feature to have a **symmetric distribution** and bring **skewness close to 0.**

---

9. Visualizing "Density" Attribute.

In [19]:

```
numerical_plot("density")
```



Inference:

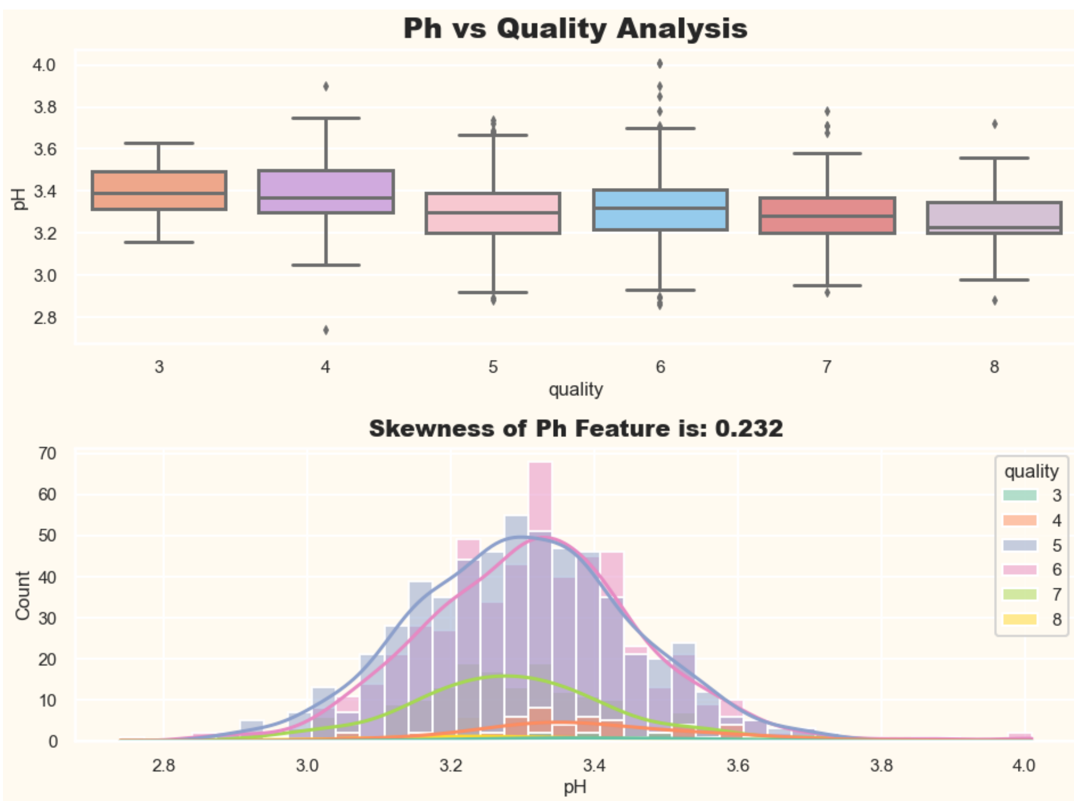
- The feature **Density** is having a perfect **Noraml Distribution** because the **skewness is close to 0**.

- So we **don't** have to use aany transformation techniques on this feature.

10. Visualizing "pH" Attribute.

In [20]:

```
numerical_plot("pH")
```



💬 **Inference:**

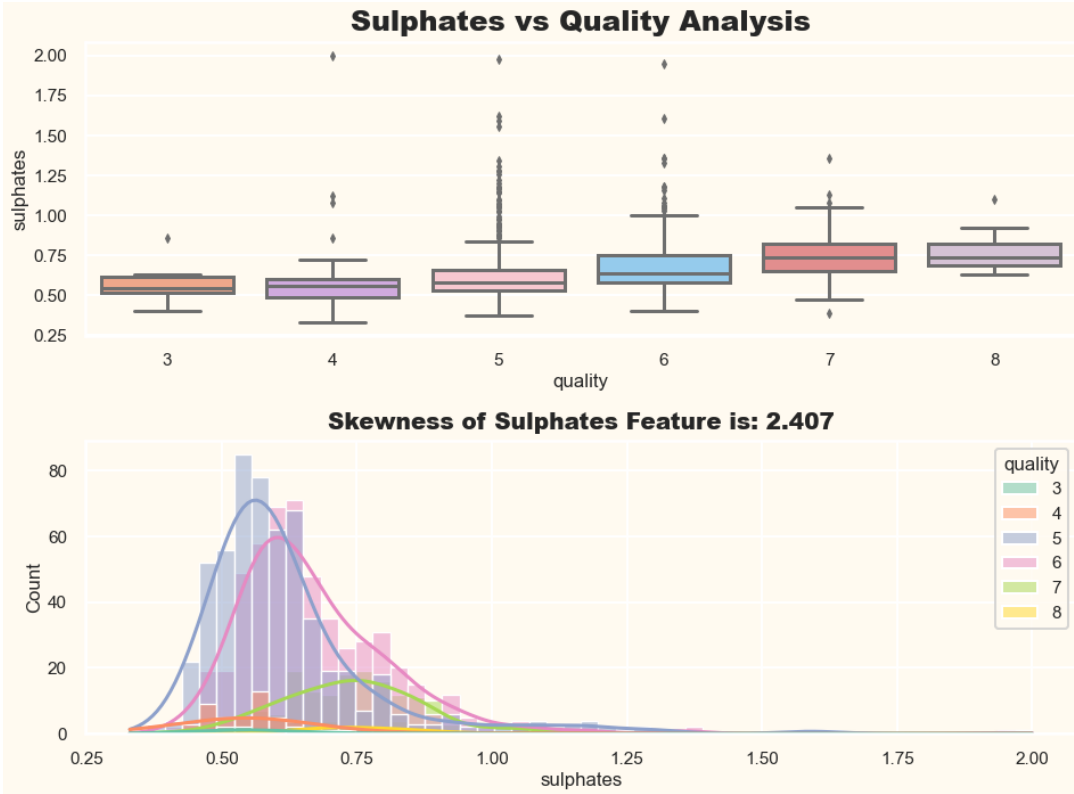
- The feature **pH** is having a **Normal Distribution** with a skewness value of **0.232**..

- But still the **tails** is little **right skewed** because of presence of **outliers.**
- So we will use **transformation techniques** to deal with those outliers.

---

11. Visualizing "Sulphates" Attribute.

In [21]:
```
numerical_plot("sulphates")
```

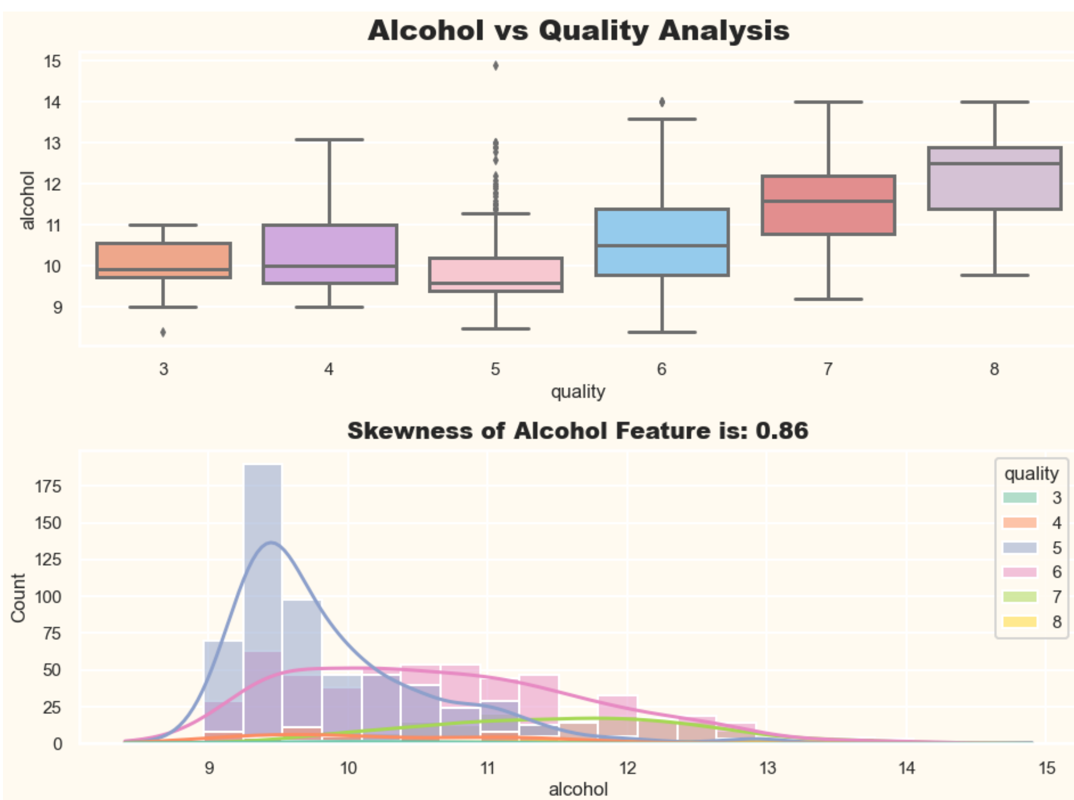


Inference:

- The feature **Sulphates** is having **almost a symmetric distribution** but the distribution is **highly right skewed** with a skewness value of **2.407**.
- The distrbution is **highly right skewed** because of presence of **outliers.**
- So we will use **tranformation techniques** to transform thesse feature to have a **symmetric distribution** and bring **skewness close to 0.**

12. Visualizing "Alcohol" Attribute.

In [22]:
```
numerical_plot("alcohol")
```



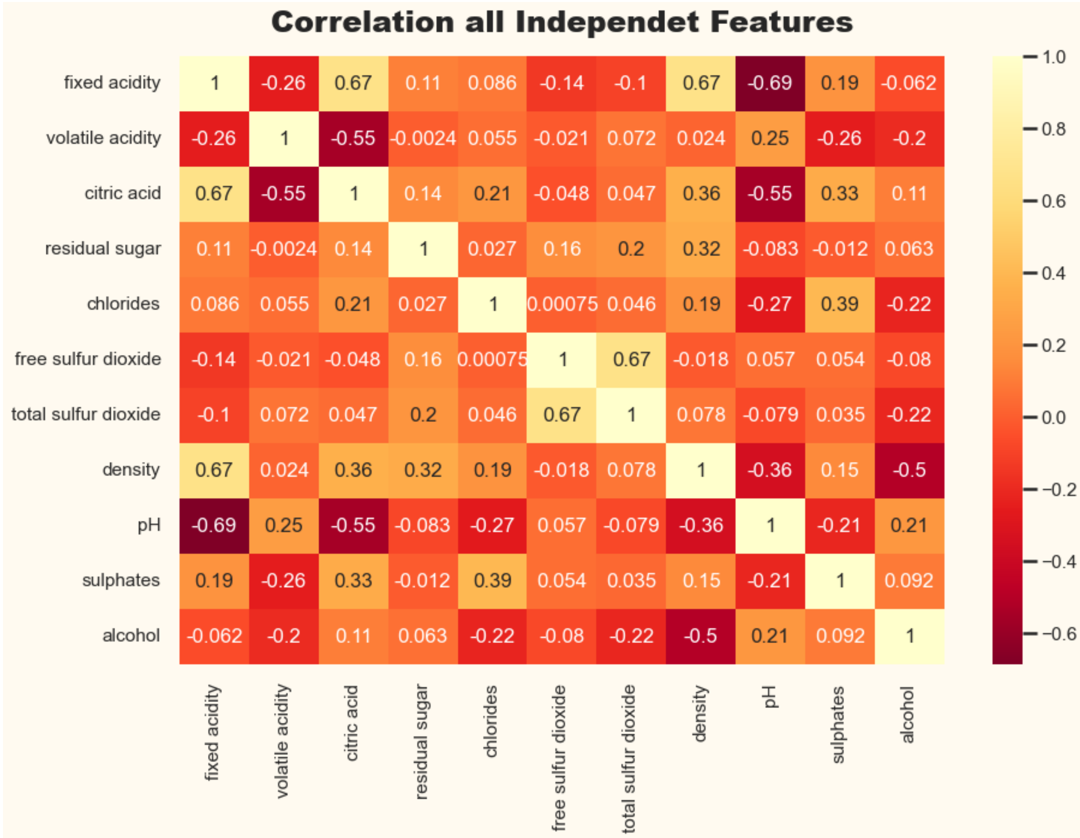
Inference:

- The feature **Alcohol** is having **Asymmetric Distribution** and the distribution is **highly right skewed** with a skewness value of **0.86**.
- The distrbution is **highly right skewed** because of presence of **outliers.**
- So we will use **tranformation techniques** to transform thesse feature to have a **symmetric distribution** and bring **skewness close to 0.**

13. Visualizing Correlation Among the Independent Atrributes.

In [23]:

```
columns = df.columns.tolist()
columns.remove("quality")  ##Quality is having discrete values that's why it's not the correct way to che

corr = df[columns].corr()

plt.figure(figsize=(13.5,10))

sns.heatmap(corr,fmt=".2g",annot=True ,cmap='YlOrRd_r')
plt.title("Correlation all Independet Features",fontweight="black",size=25,pad=20)
plt.tight_layout()
plt.show()
```



💬 **Inference:**

- Many **features** are having **high correlation** with the other features:-
  1. **Fixed Acidity** is having high correlation with **citic acid** and **pH** and vice-versa.
  2. **Volatile Acidity** is having high correlation with **citic acid** adn vice-versa.
  3. **Free Sulfur Dioxide** is having high correlation with **total sulfur dioxide**, **pH**, **Sulphates** and vice-versa.
  4. **Density** is having high correaltion with **fixed acidity**, **alcohol** and vice-versa.

- **Note:**
  - We **can't drop these** correlted features because these features **helps algorithms to create pattern** for prediction.

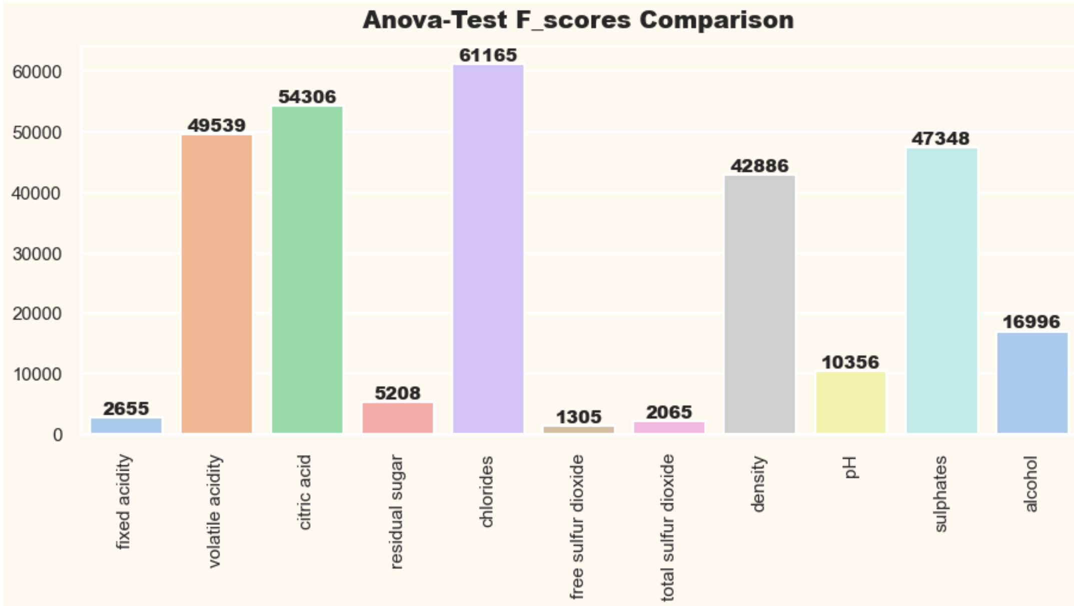
📈 Statistical Analysis - Feature Importance.📈

1. Performing ANOVA Test to Analyze the Features Importance in Wine Quality.

In [24]:
```
f_scores = {}
p_values = {}

for column in columns:
    f_score, p_value = stats.f_oneway(df[column],df["quality"])

    f_scores[column] = f_score
    p_values[column] = p_value
```

2. Visualizing the F_Score of ANOVA Test of Features.

In [25]:
```
plt.figure(figsize=(15,6))
keys = list(f_scores.keys())
values = list(f_scores.values())

sns.barplot(keys, values, palette="pastel")
plt.title("Anova-Test F_scores Comparison",fontweight="black",size=20,pad=15)
plt.xticks(rotation=90)

for index,value in enumerate(values):
    plt.text(index,value,int(value), ha="center", va="bottom",fontweight="black",size=15)
plt.show()
```



3. Comparing F_Score and P_value of ANOVA Test.

In [26]:
```
test_df = pd.DataFrame({"Features":keys,"F_Score":values})
test_df["P_value"] = list(p_values.values())
```

In [27]:
```
test_df
```

Out[27]:

	Features	F_Score	P_value
0	fixed acidity	2655.845015	0.000000e+00
1	volatile acidity	49539.593421	0.000000e+00
2	citric acid	54306.384723	0.000000e+00
3	residual sugar	5208.846019	0.000000e+00
4	chlorides	61165.590080	0.000000e+00
5	free sulfur dioxide	1305.174212	9.870616e-234
6	total sulfur dioxide	2065.769716	0.000000e+00
7	density	42886.542390	0.000000e+00
8	pH	10356.486605	0.000000e+00
9	sulphates	47348.789169	0.000000e+00
10	alcohol	16996.876103	0.000000e+00

Inference:

- The **following features showed statistically significant associations with wine quality:**
  - **Volatile Acidity.**
  - **Citric Acid.**
  - **Chlorides.**
  - **Density.**
  - **pH.**
  - **Sulphates.**
  - **Alcohol.**
- The **following features did not show statistically significant associations with wine quality.**
  - **Fixed Acidity.**
  - **Residual Sugar.**
  - **Free Sulfur Dioxide.**
  - **Total Sulfur Dioxide.**

- **Note:**
  - **Expect Residual Sugar** feature the other **3 features** were having **high correlation with** other independent features.
  - So we **can't drop those 3 features** but if required we **can drop the Residual Sugar** feature.

Data Preprocessing

1. Computing Skewness of Each Numeircal Attributes.

In [28]:
```
new_df  = df.copy()
columns = df.columns.tolist()
columns.remove("quality")
```

In [29]:
```
skew_df = df[columns].skew().to_frame().rename(columns={0:"Skewness"})
skew_df
```

Out[29]:

	Skewness
fixed acidity	0.941041
volatile acidity	0.729279
citric acid	0.312726
residual sugar	4.548153
chlorides	5.502487
free sulfur dioxide	1.226579
total sulfur dioxide	1.540368
density	0.044778
pH	0.232032
sulphates	2.406505
alcohol	0.859841

Inference:

- Except **Density**, **pH**, **citric acid** all the other features are having **high skewness.**
- Skewness can lead to several implications like **model performance**, **hypothesis testing** and it returns **biased estimation.**
- So we will use **tranformation techniques** to transform thesse feature to have a **symmetric distribution.**

---

2. Performing Skewness Transformation Analysis using Different Transformation Techniques.

In [30]:

```
columns = df.columns.tolist()
columns.remove("quality")
skewness_transformation = {}

for col in columns:
    transformed_log = np.log(df[col])                            # Log Transformation
    transformed_boxcox = special.boxcox1p(df[col], 0.15)         # Box-Cox Transformation with lambda=0.15
    transformed_inverse = 1 / df[col]                            # Inverse Transformation
    transformed_yeojohnson, _ = stats.yeojohnson(df[col])        # Yeo-Johnson Transformation
    transformed_cbrt = np.cbrt(df[col])                          # Cube Root Transformation

    # Create a dictionary for the skewness values of each transformation
    transformation_skewness = {
        "Log Transformation": stats.skew(transformed_log),
        "Box-Cox Transformation": stats.skew(transformed_boxcox),
        "Inverse Transformation": stats.skew(transformed_inverse),
        "Yeo Johnson Transformation": stats.skew(transformed_yeojohnson),
        "Cube Root Transformation": stats.skew(transformed_cbrt)}

    # Store the transformation skewness values for the column
    skewness_transformation[col] = transformation_skewness
```

In [31]:

```
result_df = pd.DataFrame.from_dict(skewness_transformation, orient='index')
result_df = pd.concat([skew_df["Skewness"], result_df], axis=1)
result_df
```

Out[31]:

	Skewness	Log Transformation	Box-Cox Transformation	Inverse Transformation	Yeo Johnson Transformation	Cube Root Transformation
fixed acidity	0.941041	0.348419	0.488956	0.252181	0.001881	0.543481
volatile acidity	0.729279	-0.330430	0.385526	1.554967	0.008302	0.009328
citric acid	0.312726	NaN	0.114207	NaN	0.016544	-1.090854
residual sugar	4.548153	1.763289	2.475999	-0.182912	-0.001713	2.490603
chlorides	5.502487	1.885558	5.004319	4.269093	-0.061854	3.065217
free sulfur dioxide	1.226579	-0.219826	0.090191	3.255936	-0.009888	0.246108
total sulfur dioxide	1.540368	-0.078074	0.165055	1.522798	-0.003893	0.392382
density	0.044778	0.036798	0.041363	-0.028870	-0.002809	0.039441
pH	0.232032	0.039720	0.105560	0.147883	-0.005001	0.103082
sulphates	2.406505	0.960399	1.726855	-0.069449	0.014621	1.349370
alcohol	0.859841	0.662627	0.704339	-0.484275	0.116613	0.725828

Inference:

- Out of all the transformation the **Yeo Johnson Transformation** has given the **best results** by reducing the skewness.
- **Inverse transformation** performation is also good on some of the features.
- But we will use **Yeo Johnson Transformation** to achieve th **Normal Distribution.**

3. Applying Yeo - Johnson Transformation on Independent variables.

In [32]:
```python
for col in columns:
    transformed_col,_ = stats.yeojohnson(df[col])
    df[col] = transformed_col
```

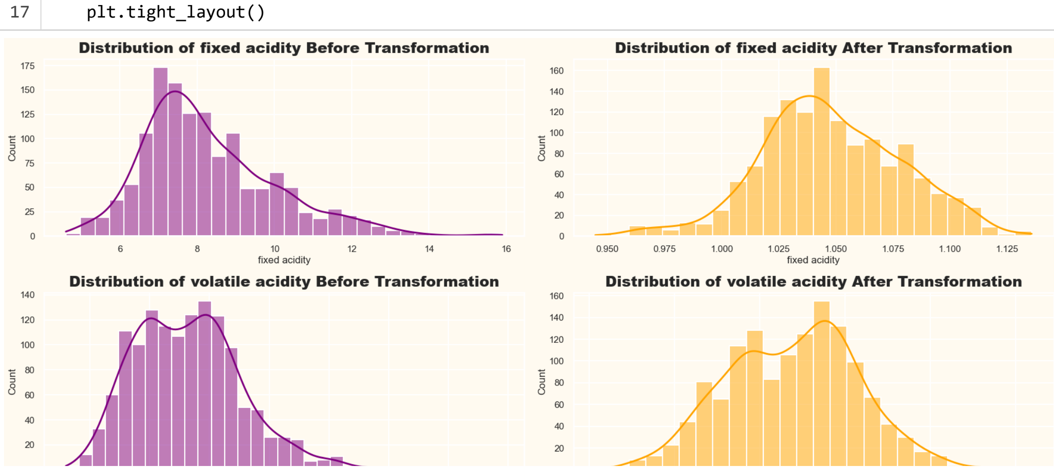
In [33]:
```python
df.sample(5)
```

Out[33]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
589	1.083009	0.225524	0.375740	0.454783	0.035040	1.903885	2.723567	0.090742	0.941016	0.215624	0.267964	7
1090	1.080283	0.206950	0.404863	0.434792	0.041241	4.278243	4.546746	0.090742	0.933546	0.208209	0.267974	8
515	1.056611	0.398169	0.375740	0.481850	0.046981	4.015134	5.336068	0.090743	0.967034	0.229303	0.267949	5
862	1.036744	0.297055	0.266332	0.456744	0.037420	2.231460	3.387543	0.090742	0.960283	0.187139	0.267963	5
1428	1.043126	0.392570	-0.000000	0.434792	0.038738	3.733998	4.224166	0.090742	0.967034	0.208209	0.267968	5

4. Comparining Distribution of Features Before & After Transformations.

In [34]:
```python
x=1
y=2


plt.figure(figsize=(25,60))
for col in columns:
    plt.subplot(11,2,x)
    sns.histplot(new_df[col],kde=True,color="purple")
    plt.title(f"Distribution of {col} Before Transformation",fontweight="black",size=25,pad=10)
    x+=2

    plt.subplot(11,2,y)
    sns.histplot(df[col],kde=True, color="orange")
    plt.title(f"Distribution of {col} After Transformation",fontweight="black",size=25,pad=10)
    y+=2

    plt.tight_layout()
```



Inference:

- We can clearly observe the **changes** their Distrbutions.
- Most of the **features** has achieved **Normal Distribution.**

4. Splitting Target Variable into two Groups.

In [35]:
```
df["quality"].unique()
```

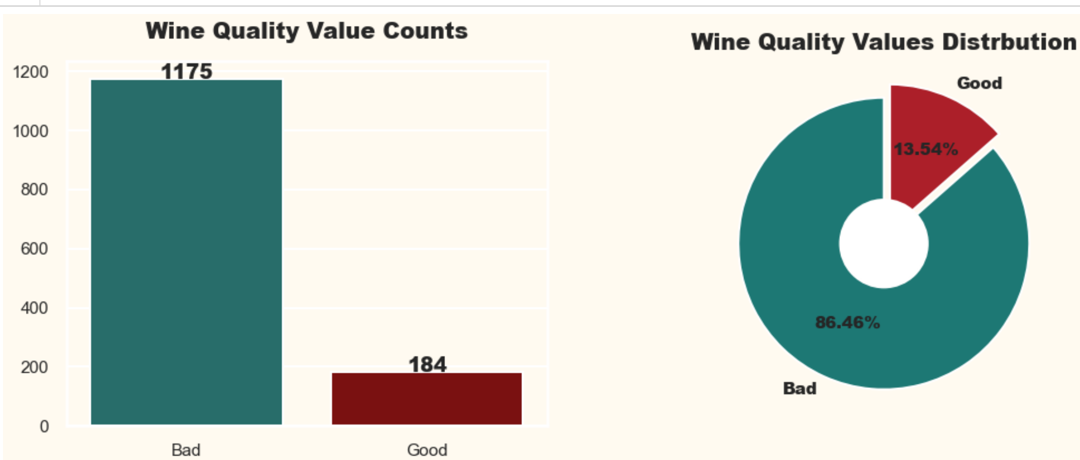
Out[35]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [36]:
```
#Condition of Splitting: If quality > 6.5 => "good" ELSE => "bad"

bin_edges = [0,6.5,10]
group_names = ["Bad","Good"]

df["quality"] = pd.cut(df["quality"], bins=bin_edges, labels=group_names)
df["quality"].unique()
```

Out[36]:
```
['Bad', 'Good']
Categories (2, object): ['Bad' < 'Good']
```

5. Visualizing the New Distribution of Target Variable.

In [37]:
```
plt.figure(figsize=(17,6))
plt.subplot(1,2,1)
quality_counts = df["quality"].value_counts()
sns.barplot(x=quality_counts.index, y=quality_counts.values,palette=["#1d7874","#8B0000"])
plt.title("Wine Quality Value Counts",fontweight="black",size=20,pad=20)
for i, v in enumerate(quality_counts.values):
    plt.text(i, v, v,ha="center", fontweight='black', fontsize=18)

plt.subplot(1,2,2)
plt.pie(quality_counts, labels=["Bad","Good"], autopct="%.2f%%", textprops={"fontweight":"black","size":1
        colors = ["#1d7874","#AC1F29"],explode=[0,0.1],startangle=90)
center_circle = plt.Circle((0, 0), 0.3, fc='white')
fig = plt.gcf()
fig.gca().add_artist(center_circle)
plt.title("Wine Quality Values Distrbution" ,fontweight="black",size=20,pad=10)
plt.show()
```



💬 **Inference:**

- We can still observe **class - imbalance**.
- The dataset is having more **Bad Quality Wines** and very **low Good quality wines.**

6. Encoding Target Variable.

In [38]:
```
df["quality"] = df["quality"].replace({"Bad":0,"Good":1})
```

In [39]:
```
df.sample(5)
```

Out[39]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
931	1.034526	0.381063	0.009935	0.438536	0.039231	2.887073	3.827766	0.090742	0.982720	0.209752	0.267956	0
288	1.060143	0.343854	0.085068	0.452650	0.042769	3.377706	4.099629	0.090742	0.969874	0.221373	0.267965	1
126	1.051056	0.576286	-0.000000	0.425999	0.040822	1.452799	2.644732	0.090742	0.987153	0.194068	0.267968	0
568	1.077467	0.334987	0.375740	0.454783	0.052243	1.903885	3.157361	0.090743	0.967034	0.218263	0.267966	0
1323	1.066832	0.265718	0.313487	0.430634	0.035357	3.434855	4.224166	0.090742	0.954351	0.219672	0.267968	1

7. Splitting Dataset using StratifiedKFold Method.

In [40]:
```
df2 = df.drop(columns="quality")
n_splits = 5  # Set the number of folds
stratified_kfold = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)
```

In [41]:
```
for train_index, test_index in stratified_kfold.split(df2, df["quality"]):
    x_train, x_test = df2.iloc[train_index], df2.iloc[test_index]
    y_train, y_test = df["quality"].iloc[train_index], df["quality"].iloc[test_index]
```

In [42]:
```
print(x_train.shape,y_train.shape)
print(x_test.shape,y_test.shape)
```

```
(1088, 11) (1088,)
(271, 11) (271,)
```

💬 **Inference:**

- The data is **splitted** in **two sets** one for Model Training and another for Model performance Testing.
- We will use **Train Dataset** to train the model and **Test Dataset** to evaluate the performance of the model.

8. Computing Frequeny of Unique Values in Y_Train & Y_Test.

In [43]:
```
unique_df = y_train.value_counts().to_frame().rename(columns={"quality":"Y_train Target frequency"})
unique_df["Y_test target Freequency"] = y_test.value_counts()
unique_df
```

Out[43]:

	Y_train Target frequency	Y_test target Freequency
0	940	235
1	148	36

💬 **Inference:**

- Both the categories in **target variable** is splitter in such a way that **model learns and create pattern** for both the categories easily.

🎯 Model Creation using DecisionTree 🎯

1. Performing Grid-Search with cross-validation to find the best Parameters for the Model.

In [44]:
```
dtree = DecisionTreeClassifier()
```

In [45]:
```
param_grid = {"max_depth":[3,4,5,6,7,8,9,10],
              "min_samples_split":[2,3,4,5,6,7,8],
              "min_samples_leaf":[1,2,3,4,5,6,7,8],
              "criterion":["gini","entropy"],
              "splitter":["best","random"],
              "max_features":["auto",None],
              "random_state":[0,42]}
```

In [47]:
```
grid_search = GridSearchCV(dtree, param_grid, cv=5, n_jobs=-1)

grid_search.fit(x_train,y_train)
```

Out[47]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 4, 5, 6, 7, 8, 9, 10],
                         'max_features': ['auto', None],
                         'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8],
                         'min_samples_split': [2, 3, 4, 5, 6, 7, 8],
                         'random_state': [0, 42],
                         'splitter': ['best', 'random']})
```

2. Fetching the Best Parameters for DecisionTree Model.

In [48]:
```
best_parameters = grid_search.best_params_

print("Best Parameters for DecisionTree Model is:\n")
best_parameters
```

```
Best Parameters for DecisionTree Model is:
```

Out[48]:
```
{'criterion': 'entropy',
 'max_depth': 9,
 'max_features': 'auto',
 'min_samples_leaf': 7,
 'min_samples_split': 2,
 'random_state': 42,
 'splitter': 'random'}
```

3. Creating DecisionTree Model Using Best Parameters.

In [49]:
```
dtree = DecisionTreeClassifier(**best_parameters)

dtree.fit(x_train,y_train)
```

Out[49]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=9, max_features='auto',
                       min_samples_leaf=7, random_state=42, splitter='random')
```

4. Computing Model Accuracy.

In [51]:
```
y_train_pred = dtree.predict(x_train)
y_test_pred = dtree.predict(x_test)

print("Accuracy Score of Model on Training Data is =>",round(accuracy_score(y_train,y_train_pred)*100,2),
print("Accuracy Score of Model on Testing Data  is =>",round(accuracy_score(y_test,y_test_pred)*100,2),"%
```

```
Accuracy Score of Model on Training Data is => 88.05 %
Accuracy Score of Model on Testing Data  is => 86.35 %
```

Inference:

- The model has obtained **88 % accuracy** on training Dataset and **86 % accuracy** on testing dataset.
- So therre's **No underfitting or Overfitting** in the model.
- The model is having a kind of **best fitting.**

5. Model Evaluation using Different Metric Values.

In [52]:
```
print("F1 Score of the Model is =>",f1_score(y_test,y_test_pred, average="weighted"))
print("Recall Score of the Model is =>",recall_score(y_test,y_test_pred, average="weighted"))
print("Precision Score of the Model is =>",precision_score(y_test,y_test_pred, average="weighted"))
```

```
F1 Score of the Model is => 0.8447513237499371
Recall Score of the Model is => 0.8634686346863468
Precision Score of the Model is => 0.8371736536914242
```

💬 **Inference:**

- We can observe that **recall, precision, and F1 score are approxiamately same**, it means that our **model is achieving perfect balance between** correctly identifying **positive samples (recall)** and minimizing **false positives (precision).**

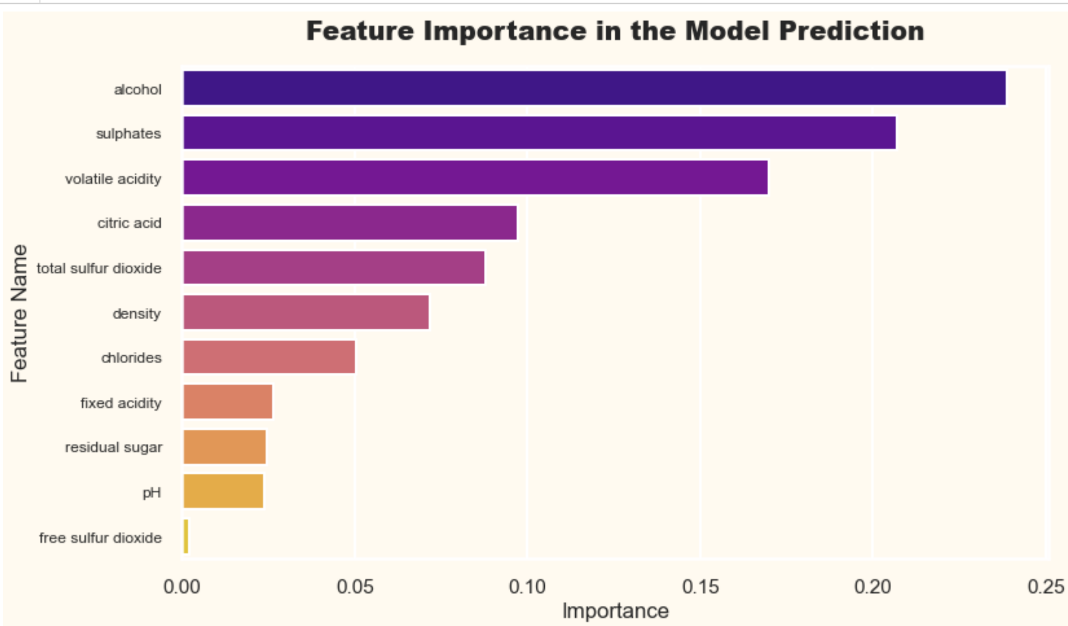
- The **high values** for **F1 score, recall score, and precision score**, all of which are **more than 0.8.**
- These **metrics suggest** that the **model achieves good accuracy** in predicting the **positive class.**

---

6. Finding Importance of Features in DecisionTreeClassifier.

In [53]:
```
imp_df = pd.DataFrame({"Feature Name":x_train.columns,
                       "Importance":dtree.feature_importances_})
```

In [54]:
```
features = imp_df.sort_values(by="Importance",ascending=False)

plt.figure(figsize=(12,7))
sns.barplot(x="Importance", y="Feature Name", data=features, palette="plasma")
plt.title("Feature Importance in the Model Prediction", fontweight="black", size=20, pad=20)
plt.yticks(size=12)
plt.show()
```



💬 **Inference:**

- The **key factors** that significantly influence the **quality of wine** are:-

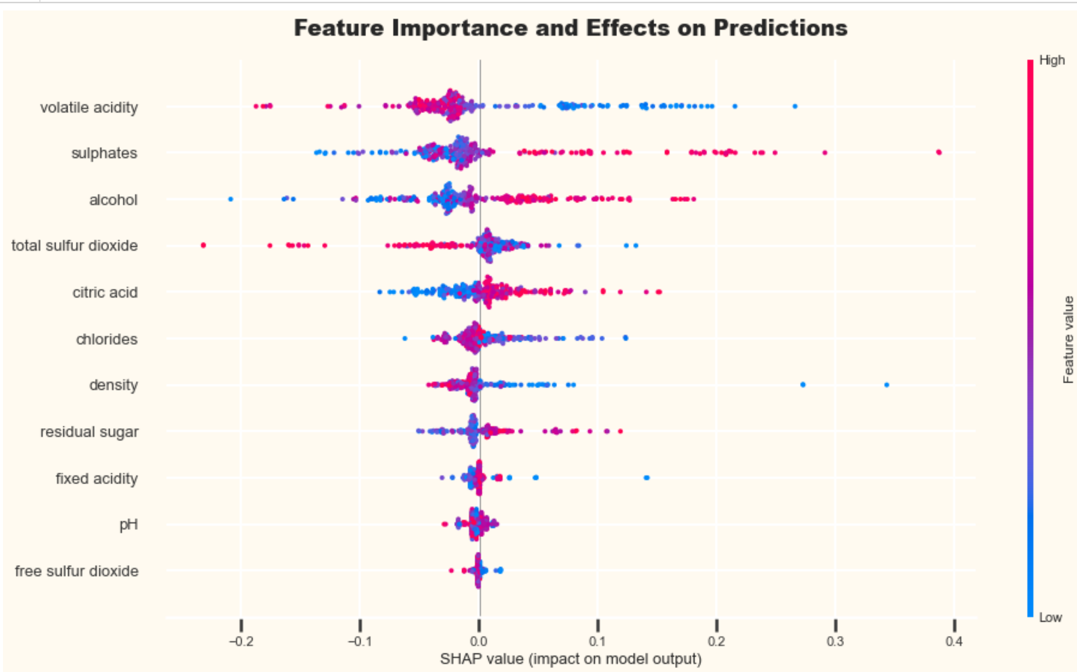
- **`Alcohol`**, **`Sulphates`**, and **`Volatile Acidity`**.

- The **minimal impact** of features on the **wine quality** are:-
    - **`free sulfur dioxide`**, **`pH`**, and **`Residual Sugar`**

7. SHAP Summary Plot: Explaining Model Predictions with Feature Importance.

In [55]:

```python
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(x_test)

plt.title("Feature Importance and Effects on Predictions",fontweight="black",pad=20,size=18)
shap.summary_plot(shap_values[1], x_test.values, feature_names = x_test.columns,plot_size=(14,8))
```



💬 Inference:

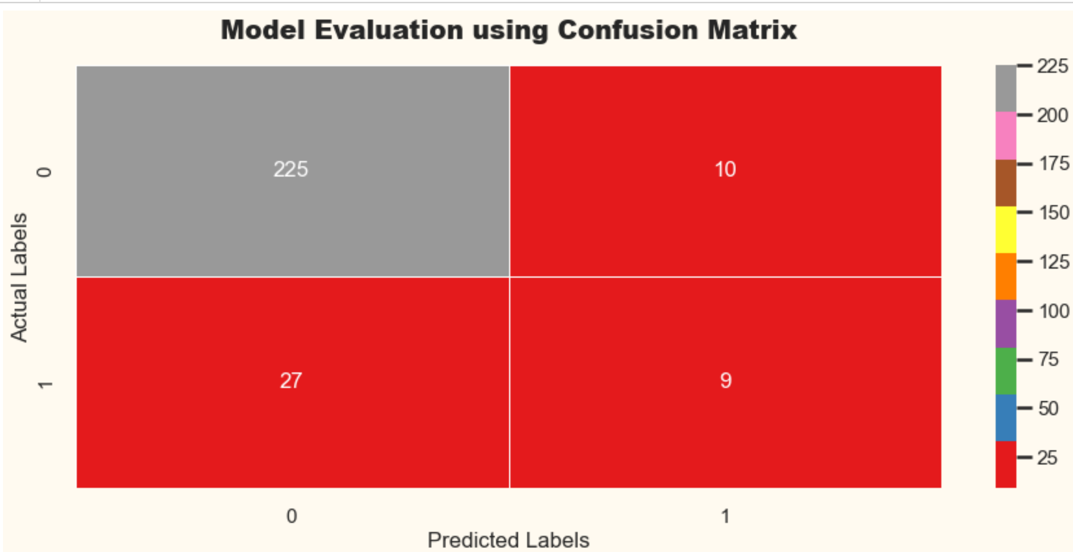
- The **red color represents high feature values**, indicating that the **feature positively contributes** for **increasing the prediction value.**
- The **blue color represents low feature values,** indicating that the **feature negatively contributes** for **decreasing the prediction value.**

---

8. Model Evaluation using Confusion Matrix.

In [56]:

```python
cm = confusion_matrix(y_test,y_test_pred)

plt.figure(figsize=(15,6))
sns.heatmap(data=cm, linewidth=.5, annot=True, fmt="g", cmap="Set1")
plt.title("Model Evaluation using Confusion Matrix",fontsize=20,pad=20,fontweight="black")
plt.ylabel("Actual Labels")
plt.xlabel("Predicted Labels")
plt.show()
```



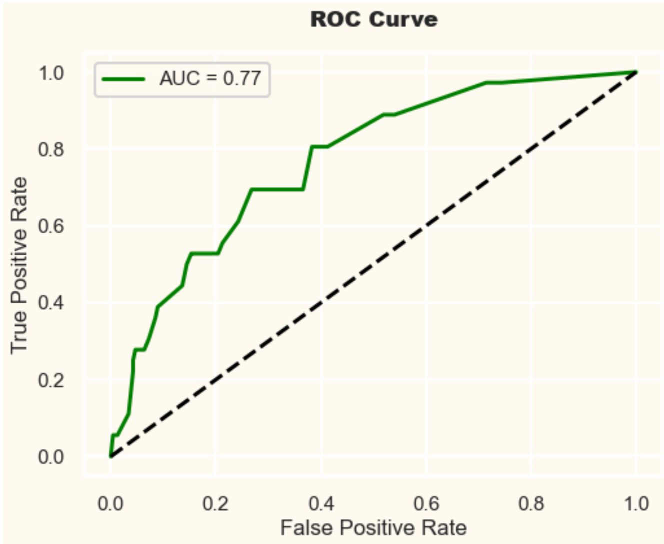
Inference:

- **Strong True Positive Rate:** The model achieved a high number of true positive predictions, indicating its ability to correctly identify positive cases. This suggests that the model is effective in accurately classifying the desired outcome.
- **Need of Improvement in False Negative Rate:** The presence of a relatively high number of false negatives suggests that the model may have missed identifying some actual positive cases. This indicates a need for further refinement to enhance the model's ability to capture all positive cases.

9. Model Evaluation: ROC Curve and Area Under the Curve (AUC)

In [57]:

```
y_pred_proba = dtree.predict_proba(x_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=["y_actual"])])
df_actual_predicted.index = y_test.index


fpr, tpr, thresholds = roc_curve(df_actual_predicted["y_actual"], y_pred_proba)
auc = roc_auc_score(df_actual_predicted["y_actual"], y_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"AUC = {auc:.2f}",color="green")
plt.plot([0, 1], [0, 1], linestyle="--", color="black")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve",pad=20,fontweight="black")
plt.legend()
plt.show()
```



💬 **Inference:**

1. An **AUC (Area Under the Curve) value of 0.77** suggests that the model has **strong discriminative power.**
2. This suggests that the model has a **high ability to distinguish between positive and negative instances**, indicating its effectiveness in making accurate predictions.
3. The **model has a relatively high probability** of ranking a randomly selected positive instance higher than a randomly selected negative instance.

🎯 Model Creation using RandomForest 🎯

1. Performing Grid-Search with cross-validation to find the best Parameters for the Model.

In [58]:
```
rfc = RandomForestClassifier()
```

In [59]:
```
param_grid = {"max_depth":[3,4,5,6,7,8],
              "min_samples_split":[3,4,5,6,7,8],
              "min_samples_leaf":[3,4,5,6,7,8],
              "n_estimators": [50,70,90,100],
              "criterion":["gini","entropy"]}
```

In [60]:
```
grid_search = GridSearchCV(rfc, param_grid, cv=5, n_jobs=-1)

grid_search.fit(x_train,y_train)
```

Out[60]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 4, 5, 6, 7, 8],
                         'min_samples_leaf': [3, 4, 5, 6, 7, 8],
                         'min_samples_split': [3, 4, 5, 6, 7, 8],
                         'n_estimators': [50, 70, 90, 100]})
```

---

2. Fetching the Best Parameters for RandomForest Model.

In [61]:
```
best_parameters = grid_search.best_params_

print("Best Parameters for RandomForest Model is:\n\n")
best_parameters
```

```
Best Parameters for RandomForest Model is:
```

Out[61]:
```
{'criterion': 'gini',
 'max_depth': 4,
 'min_samples_leaf': 3,
 'min_samples_split': 7,
 'n_estimators': 70}
```

---

3. Creating RandomForest Model Using Best Parameters.

In [62]:
```
rfc = RandomForestClassifier(**best_parameters)

rfc.fit(x_train,y_train)
```

Out[62]:
```
RandomForestClassifier(max_depth=4, min_samples_leaf=3, min_samples_split=7,
                       n_estimators=70)
```

---

4. Computing Model Accuracy.

In [63]:
```
y_train_pred = rfc.predict(x_train)
y_test_pred  = rfc.predict(x_test)

print("Accuracy Score of Model on Training Data is =>",round(accuracy_score(y_train,y_train_pred)*100,2),
print("Accuracy Score of Model on Testing Data  is =>",round(accuracy_score(y_test,y_test_pred)*100,2),"%
```

```
Accuracy Score of Model on Training Data is => 90.44 %
Accuracy Score of Model on Testing Data  is => 87.08 %
```

Inference:

- The model has obtained **90 % accuracy** on training Dataset and **87 % accuracy** on testing dataset.
- So there's **No underfitting or Overfitting** in the model.
- The model is having a kind of **best fitting.**

---

5. Model Evaluation using Different Metric Values.

In [65]:
```
print("F1 Score of the Model is =>",f1_score(y_test,y_test_pred,average="weighted"))
print("Recall Score of the Model is =>",recall_score(y_test,y_test_pred,average="weighted"))
print("Precision Score of the Model is =>",precision_score(y_test,y_test_pred,average="weighted"))
```

```
F1 Score of the Model is => 0.8397613569946145
Recall Score of the Model is => 0.87084870848708480848
Precision Score of the Model is => 0.8395608081954945
```

💬 Inference:

- We can observe that **recall, precision, and F1 score are approxiamately same**, it means that our **model is achieving perfect balance between** correctly identifying **positive samples (recall)** and minimizing **false positives (precision).**

- The **high values** for **F1 score, recall score, and precision score**, all of which are **more than 0.8.**
- These **metrics suggest** that the **model achieves good accuracy** in predicting the **positive class.**

---

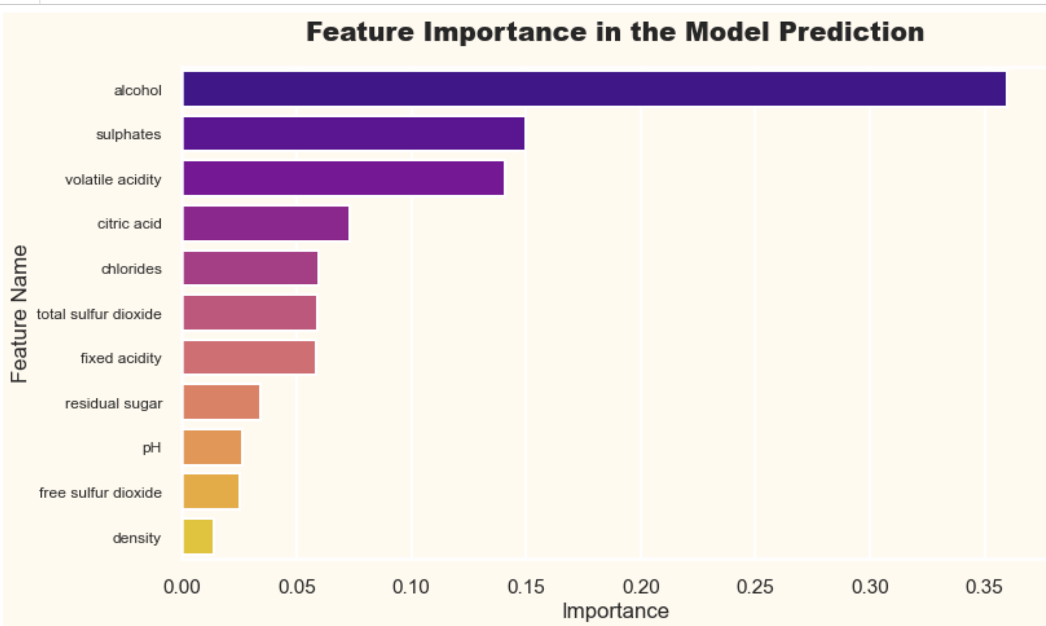
6. Finding Importance of Features in RandomForest Model.

In [66]:

```python
imp_df = pd.DataFrame({"Feature Name":x_train.columns,
                       "Importance":rfc.feature_importances_})
```

In [67]:

```python
features = imp_df.sort_values(by="Importance",ascending=False)

plt.figure(figsize=(12,7))
sns.barplot(x="Importance", y="Feature Name", data=features, palette="plasma")
plt.title("Feature Importance in the Model Prediction", fontweight="black", size=20, pad=20)
plt.yticks(size=12)
plt.show()
```



💬 Inference:

- The **key factors** that significantly influence the **deactivation of customers banking facilities** are:-

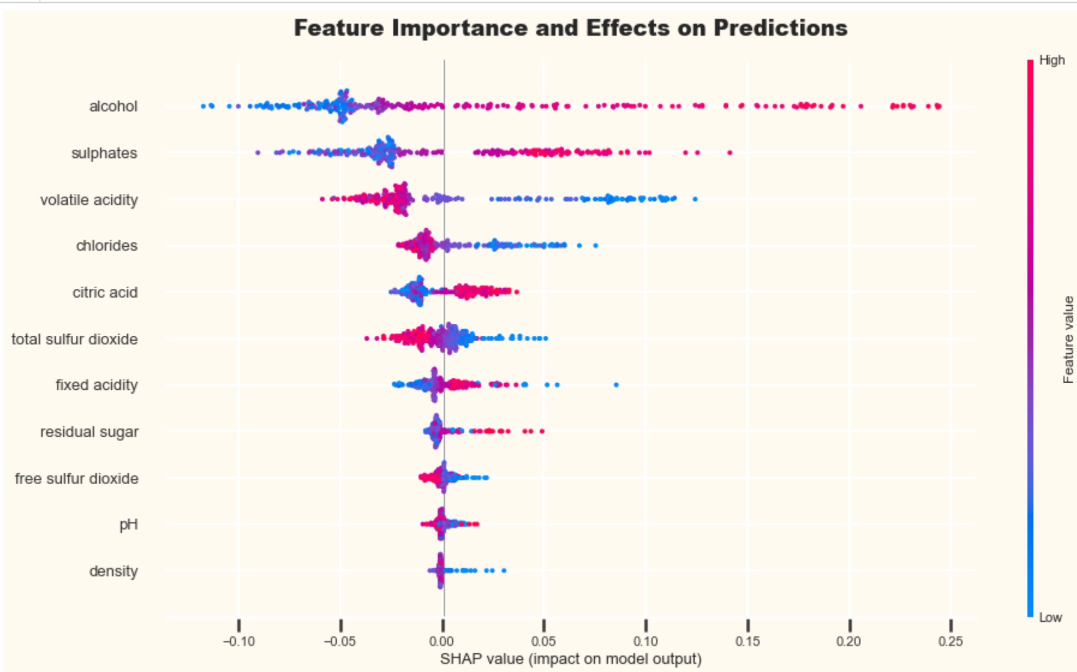
- **Alcohol** , **Sulphates** , and **Volatilee Acidity** .

- The **minimal impact** of features on the **deactivation of customers' banking facilities** are:-
  - **Density** , **Free Sulfur Dioxide** , and **pH** .

7. SHAP Summary Plot: Explaining Model Predictions with Feature Importance.

In [68]:

```
import shap
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(x_test)

plt.title("Feature Importance and Effects on Predictions",fontweight="black",pad=20,size=18)
shap.summary_plot(shap_values[1], x_test.values, feature_names = x_test.columns,plot_size=(14,8))
```



💬 **Inference:**

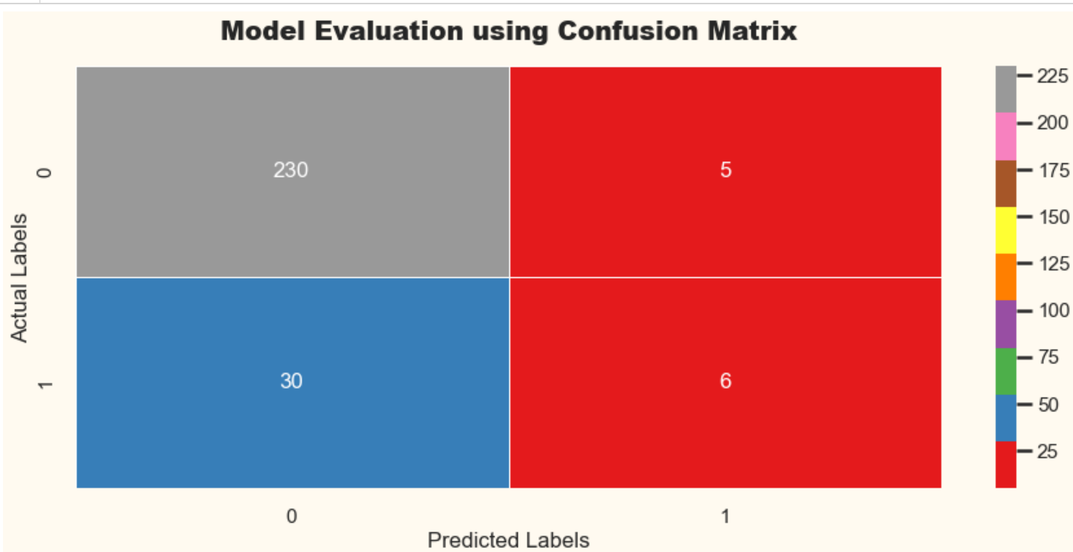
- The **red color represents high feature values**, indicating that the **feature positively contributes** for **increasing the prediction value.**
- The **blue color represents low feature values,** indicating that the **feature negatively contributes** for **decreasing the prediction value.**

---

8. Model Evaluation using Confusion Matrix.

In [70]:

```
cm = confusion_matrix(y_test,y_test_pred)

plt.figure(figsize=(15,6))
sns.heatmap(data=cm, linewidth=.5, annot=True, fmt="g", cmap="Set1")
plt.title("Model Evaluation using Confusion Matrix",fontsize=20,pad=20,fontweight="black")
plt.ylabel("Actual Labels")
plt.xlabel("Predicted Labels")
plt.show()
```



Inference:

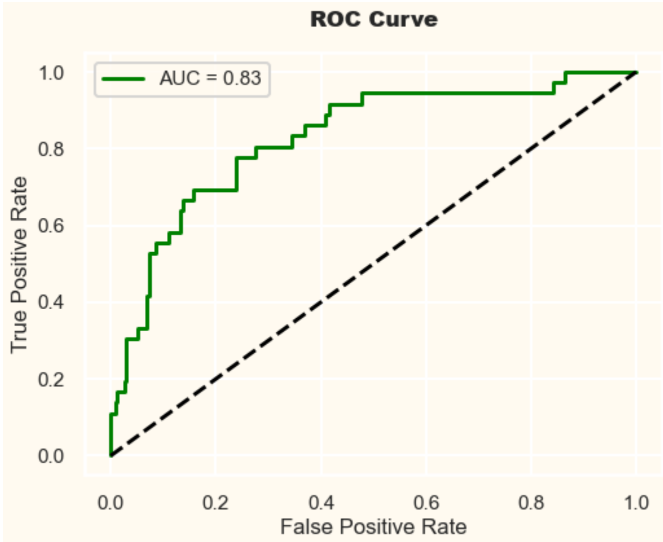
- **Strong True Positive Rate:** The model achieved a high number of true positive predictions, indicating its ability to correctly identify positive cases. This suggests that the model is effective in accurately classifying the desired outcome.
- **Need of Improvement in False Negative Rate:** The presence of a relatively high number of false negatives suggests that the model may have missed identifying some actual positive cases. This indicates a need for further refinement to enhance the model's ability to capture all positive cases.

---

9. Model Evaluation: ROC Curve and Area Under the Curve (AUC)

In [71]:

```
y_pred_proba = rfc.predict_proba(x_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=["y_actual"])])
df_actual_predicted.index = y_test.index


fpr, tpr, thresholds = roc_curve(df_actual_predicted["y_actual"], y_pred_proba)
auc = roc_auc_score(df_actual_predicted["y_actual"], y_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"AUC = {auc:.2f}",color="green")
plt.plot([0, 1], [0, 1], linestyle="--", color="black")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve",pad=20,fontweight="black")
plt.legend()
plt.show()
```



💬 **Inference:**

1. An **AUC (Area Under the Curve) value of 0.83** suggests that the model has **strong discriminative power.**
2. This suggests that the model has a **high ability to distinguish between positive and negative instances**, indicating its effectiveness in making accurate predictions.
3. The **model has a relatively high probability** of ranking a randomly selected positive instance higher than a randomly selected negative instance.

🎈 Conclusion 🎈

💬 **Key-Findings:**

- The **key factors** that significantly influence the **wine quality** are **Alcohol**, **Sulphates** and **Volatilee Acidity.**
- The **minimal impact** of features on the **wine quality** are **Free Sulfur Dioxide** and **pH**.
- **High Training and Testing Accuracies:** Both the model achieved a high accuracy score near to 90% on the training data, indicating a good fit to the training instances. Additionally, the model's accuracy score near to 87% on the testing data suggests its ability to generalize well to unseen instances.
- **High F1 Score, Recall, and Precision:** The model achieved high F1 score, recall, and precision values, all **more than 0.8.** This indicates that the model has a strong ability to correctly identify positive cases while minimizing false positives and maximizing true positives.
- **High AUC value more than 0.8**, states that the model demonstrates a reasonably good discriminatory power. It suggests that the model is able to distinguish between positive and negative instances with a relatively high degree of accuracy.
- **Overall Model Performance:** The model demonstrates strong performance across multiple evaluation metrics, indicating its effectiveness in making accurate predictions and capturing the desired outcomes.

💡 Recommendations 💡

💬 **Key-Suggestions:**

- **Key Quality Drivers:** Prioritize **Alcohol, Sulphates, and Volatile Acidity** as they significantly influence wine quality.
- **Quality Control:** Implement stringent **quality control measures** to maintain desired levels of the key factors during production.
- **Varietal-specific Approach:** Tailor the **winemaking process** to suit each grape variety's unique characteristics and quality requirements.
- **Technology and Expertise:** Invest in modern winemaking technology and **employ experienced winemakers** to ensure precise management of quality factors.
- **Continuous Improvement:** Continuously monitor **customer feedback and wine ratings** to identify areas for improvement and enhance overall product quality.

🏁 THE END 🏁

🔴 **If you liked this Notebook, please do upvote.** 🔴

🔴 **If you have any questions, feel free to comment!** 🔴

🌼 **Best Wishes!** 🌼