```
# importting Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings as wr
wr.filterwarnings('ignore')

from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving winequality-red.csv to winequality-red.csv
```

```
df = pd.read_csv("winequality-red.csv")
print(df.head())
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0            7.4              0.70         0.00             1.9
0.076
1            7.8              0.88         0.00             2.6
0.098
2            7.8              0.76         0.04             2.3
0.092
3           11.2              0.28         0.56             1.9
0.075
4            7.4              0.70         0.00             1.9
0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates
\
0                 11.0                  34.0   0.9978  3.51       0.56

1                 25.0                  67.0   0.9968  3.20       0.68

2                 15.0                  54.0   0.9970  3.26       0.65

3                 17.0                  60.0   0.9980  3.16       0.58

4                 11.0                  34.0   0.9978  3.51       0.56


   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

```
#Analysing the Data
df.describe()
```

```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1599.000000       1599.000000  1599.000000     1599.000000
mean        8.319637          0.527821     0.270976        2.538806
std         1.741096          0.179060     0.194801        1.409928
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.390000     0.090000        1.900000
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide
density  \
count  1599.000000          1599.000000           1599.000000
1599.000000
mean      0.087467            15.874922             46.467792
0.996747
std       0.047065            10.460157             32.895324
0.001887
min       0.012000             1.000000              6.000000
0.990070
25%       0.070000             7.000000             22.000000
0.995600
50%       0.079000            14.000000             38.000000
0.996750
75%       0.090000            21.000000             62.000000
0.997835
max       0.611000            72.000000            289.000000
1.003690

                pH    sulphates      alcohol      quality
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000

<google.colab._quickchart_helpers.SectionTitle at 0x7d98b12759f0>

from matplotlib import pyplot as plt
_df_0['fixed acidity'].plot(kind='hist', bins=20, title='fixed
acidity')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_1['volatile acidity'].plot(kind='hist', bins=20, title='volatile
acidity')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_2['citric acid'].plot(kind='hist', bins=20, title='citric acid')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_3['residual sugar'].plot(kind='hist', bins=20, title='residual
sugar')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7d9877a0fbb0>
```

```
from matplotlib import pyplot as plt
_df_4.plot(kind='scatter', x='fixed acidity', y='volatile acidity',
s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_5.plot(kind='scatter', x='volatile acidity', y='citric acid',
s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_6.plot(kind='scatter', x='citric acid', y='residual sugar', s=32,
alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_7.plot(kind='scatter', x='residual sugar', y='chlorides', s=32,
alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7d98b1274820>
```

```
from matplotlib import pyplot as plt
_df_8['fixed acidity'].plot(kind='line', figsize=(8, 4), title='fixed
acidity')
plt.gca().spines[['top', 'right']].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_9['volatile acidity'].plot(kind='line', figsize=(8, 4),
title='volatile acidity')
plt.gca().spines[['top', 'right']].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_10['citric acid'].plot(kind='line', figsize=(8, 4), title='citric
acid')
plt.gca().spines[['top', 'right']].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_11['residual sugar'].plot(kind='line', figsize=(8, 4),
title='residual sugar')
plt.gca().spines[['top', 'right']].set_visible(False)
```

```
df.shape
```

```
(1599, 12)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

```
df.isnull()
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0             False             False        False           False
False
1             False             False        False           False
False
2             False             False        False           False
False
3             False             False        False           False
False
4             False             False        False           False
False
...             ...               ...          ...             ...
...
1594          False             False        False           False
False
1595          False             False        False           False
False
```

```
1596          False             False        False           False
False
1597          False             False        False           False
False
1598          False             False        False           False
False

      free sulfur dioxide  total sulfur dioxide  density     pH
sulphates  \
0                   False                 False    False  False
False
1                   False                 False    False  False
False
2                   False                 False    False  False
False
3                   False                 False    False  False
False
4                   False                 False    False  False
False
...                   ...                   ...      ...    ...
...
1594                False                 False    False  False
False
1595                False                 False    False  False
False
1596                False                 False    False  False
False
1597                False                 False    False  False
False
1598                False                 False    False  False
False

      alcohol  quality
0       False    False
1       False    False
2       False    False
3       False    False
4       False    False
...       ...      ...
1594    False    False
1595    False    False
1596    False    False
1597    False    False
1598    False    False

[1599 rows x 12 columns]
```

```
#check for missing values in each column of the DataFrame ‘df’ and
returns the sum of null values for each column
df.isnull().sum()
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

```
#Describe the columns
df.columns.tolist()
```

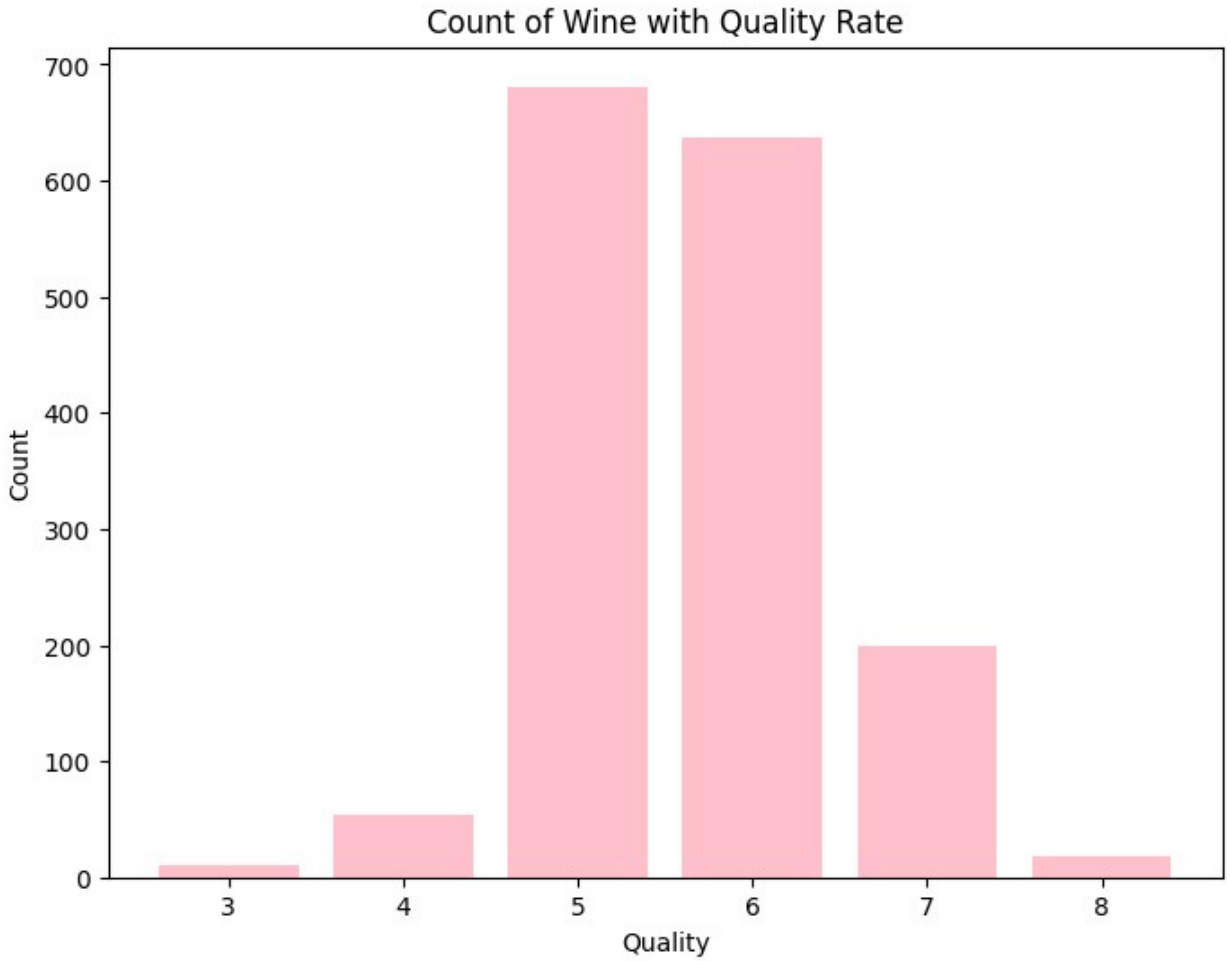
```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol',
 'quality']
```

```
#checking duplicate values
df.nunique()
'''The function df.nunique() determines how many unique values there
are in each column of the DataFrame “df,”
offering information about the variety of data that makes up each
feature'''
```

```
fixed acidity            96
volatile acidity        143
citric acid              80
residual sugar           91
chlorides               153
free sulfur dioxide      60
total sulfur dioxide    144
density                 436
pH                       89
sulphates                96
alcohol                  65
quality                   6
dtype: int64
```

Univariate Analysis

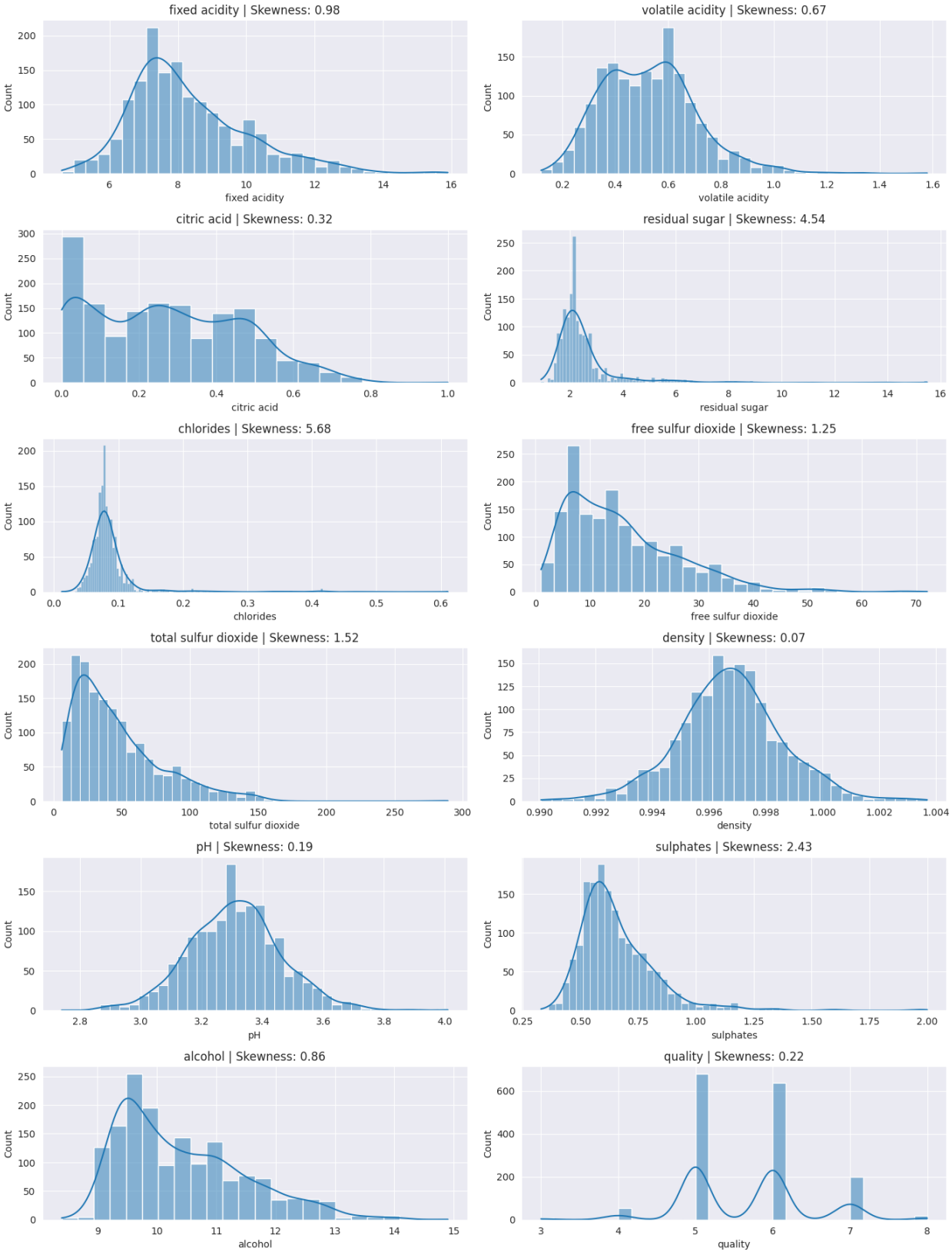
```
quality_counts = df['quality'].value_counts()

# Using Matplotlib to create a count plot
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='pink')
plt.title('Count of Wine with Quality Rate')
plt.xlabel('Quality')
plt.ylabel('Count')
plt.show()
```



#Kernel Density Plots

Kernel density plot is about the skewness of the of the corresponding feature. The features in this dataset that have skewness are exactly 0 depicts the symmetrical distribution and the plots with skewness 1 or above 1 is positively or right skewed distribution. In right skewed or positively skewed distribution if the tail is more on the right side, that indicates extremely high values.

```
# Set Seaborn style
sns.set_style("darkgrid")

# Identify numerical columns
numerical_columns = df.select_dtypes(include=["int64",
"float64"]).columns

# Plot distribution of each numerical feature
plt.figure(figsize=(14, len(numerical_columns) * 3))
for idx, feature in enumerate(numerical_columns, 1):
     plt.subplot(len(numerical_columns), 2, idx)
     sns.histplot(df[feature], kde=True)
     plt.title(f"{feature} | Skewness: {round(df[feature].skew(),
2)}")


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



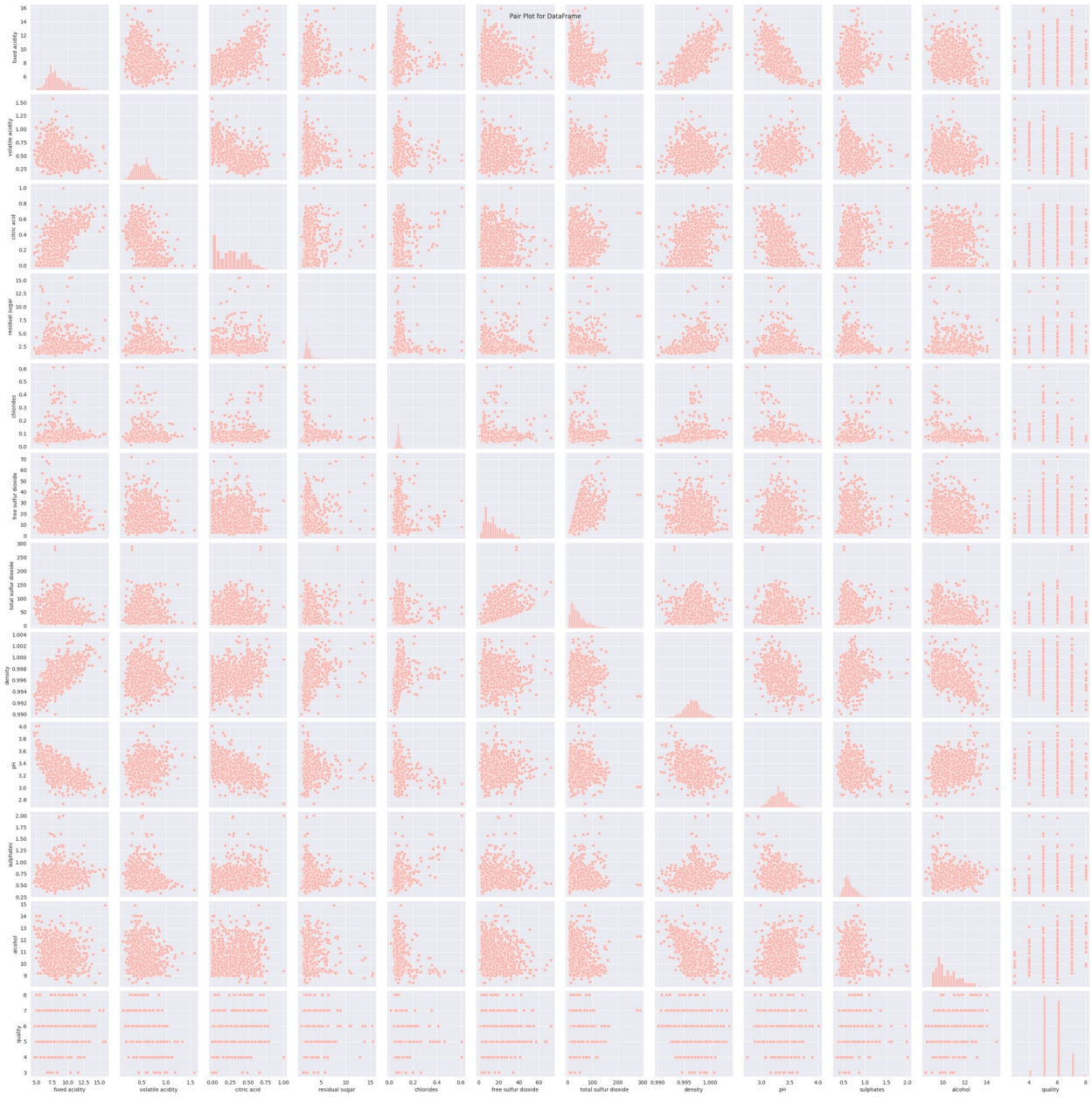
#Bivariate Analysis

When doing a bivariate analysis, two variables are examined simultaneously in order to look for patterns, dependencies, or interactions between them. Understanding how changes in one variable may correspond to changes in another requires the use of this statistical method.

Bivariate analysis allows for a thorough comprehension of the interdependence between two variables within a dataset by revealing information on the type and intensity of associations.

```
# Set the color palette
sns.set_palette("Pastel1")

plt.figure(figsize=(10, 6))
sns.pairplot(df)

plt.suptitle('Pair Plot for DataFrame')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```

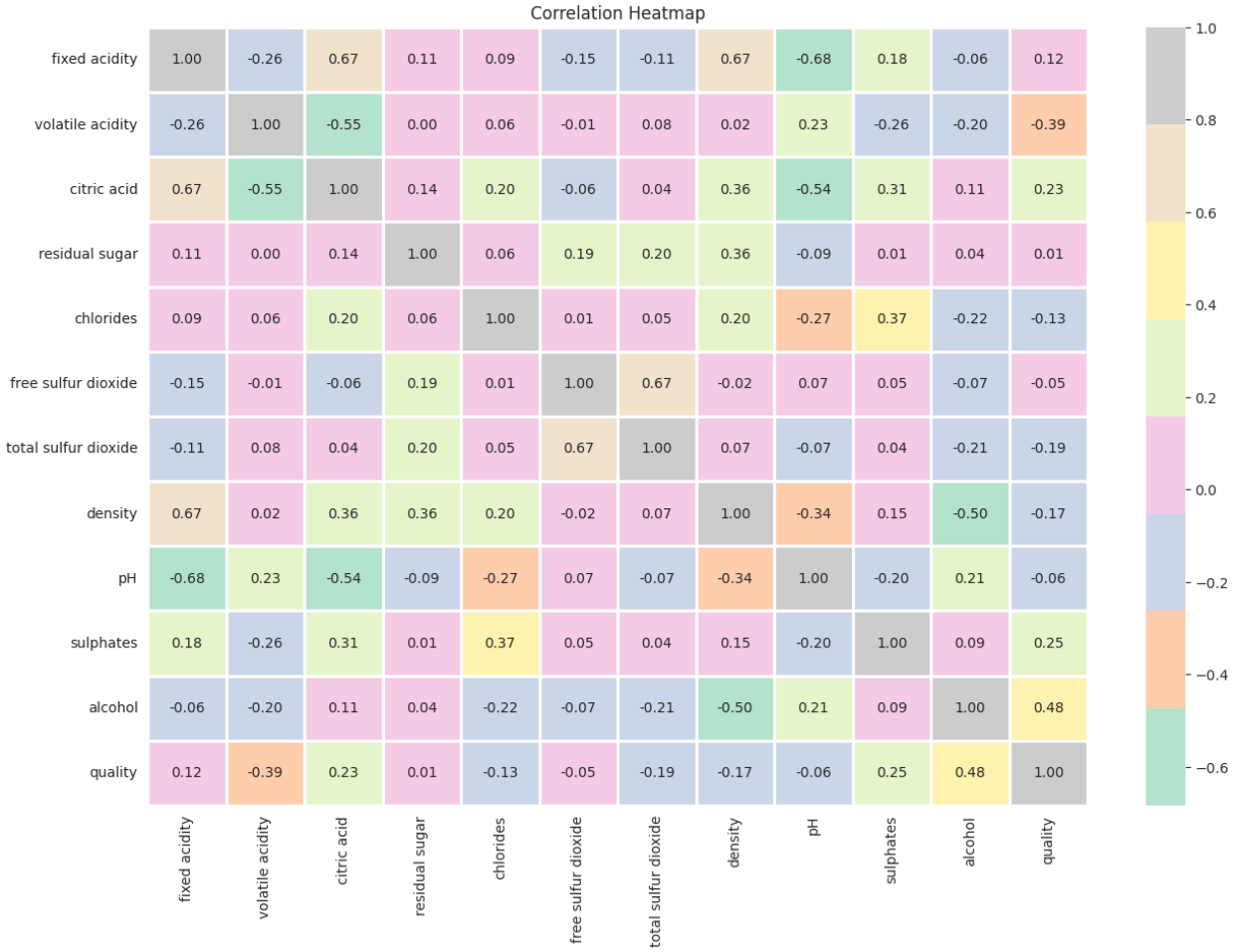


- If the plot is diagonal , histograms of kernel density plots , is shows the distribution of the individual variables.
- If the scatter plot is in the lower triangle, it displays the relationship between the pairs of the variables.
- If the scatter plots above and below the diagonal are mirror images, indicating symmetry.
- If the histogram plots are more centered, it represents the locations of peaks.
- Skewness is depicted by observing whether the histogram is symmetrical or skewed to the left or right.

#Multivariate Analysis Interactions between three or more variables in a dataset are simultaneously analyzed and interpreted in multivariate analysis.

In order to provide a comprehensive understanding of the collective behavior of several variables, it seeks to reveal intricate patterns, relationships, and interactions between them

```
plt.figure(figsize=(15, 10))
sns.heatmap(df.corr(), annot=True, fmt='.2f', cmap='Pastel2',
linewidths=2)

plt.title('Correlation Heatmap')
plt.show()
```



For interpreting a correlation matrix plot,

- Values close to +1 indicates strong positive correlation, -1 indicates a strong negative correlation and 0 indicates suggests no linear correlation.
- Darker colors signify strong correlation, while light colors represents weaker correlations.
- Positive correlation variable move in same directions. As one increases, the other also increases.
- Negative correlation variable move in opposite directions. An increase in one variable is associated with a decrease in the other

#Linear Regression

Features (X):

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

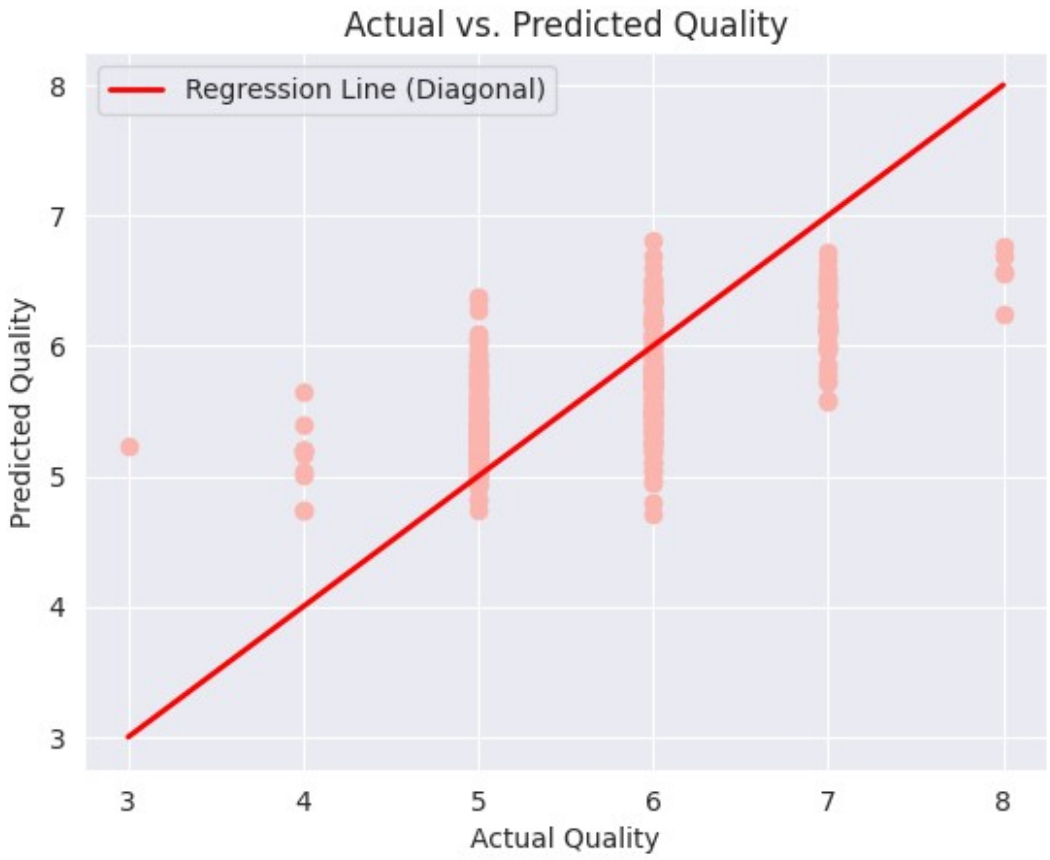
Target Variable (y):

- quality

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load your red wine quality dataset into a pandas DataFrame
# Assume your DataFrame is named 'df'
# Example: df = pd.read_csv('your_dataset.csv')

# Select features (X) and target variable (y)
X = df.drop('quality', axis=1)
y = df['quality']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
```

```
print(f'R-squared: {r2}')

# Visualize the predicted vs. actual values
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Quality")
plt.ylabel("Predicted Quality")
plt.title("Actual vs. Predicted Quality")
diagonal_line = np.linspace(min(min(y_test), min(y_pred)),
max(max(y_test), max(y_pred)), 100)
plt.plot(diagonal_line, diagonal_line, color='red', linewidth=2,
label='Regression Line (Diagonal)')

plt.legend()
plt.show()
```

```
Mean Squared Error: 0.39002514396395416
R-squared: 0.4031803412796231
```



- The x-axis shows the actual quality values.
- The y-axis shows the corresponding predicted quality values made by your linear regression model.

- The Mean Squared Error (MSE) and R-squared (R2) values that you printed in your code are quantitative measures of your model's performance. A lower MSE and a higher R-squared indicate better performance.
- If the points are clustered closely around the diagonal line, it suggests that your model is making accurate predictions.
- If the points are scattered and deviate from the diagonal line, it indicates discrepancies between the predicted and actual values.
- In Linear regression, we mostly use metrics like MSE and R2 for evaluation.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

#Select features (X) and target variable (y)
X = df.drop('quality', axis=1)
y = df['quality']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create and train Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_linear = linear_model.predict(X_test)

# Create and train Decision Tree Regressor model
tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(X_train, y_train)
y_pred_tree = tree_model.predict(X_test)

# Create and train Random Forest Regressor model
forest_model = RandomForestRegressor(n_estimators=100,
random_state=42)
forest_model.fit(X_train, y_train)
y_pred_forest = forest_model.predict(X_test)

# Evaluate the models
models = [('Linear Regression', y_pred_linear),
          ('Decision Tree Regressor', y_pred_tree),
          ('Random Forest Regressor', y_pred_forest)]

for model_name, y_pred in models:
```

```
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f'{model_name}:')
    print(f'Mean Squared Error: {mse}')
    print(f'R-squared: {r2}\n')

# Visualize the predicted vs. actual values for Random Forest
Regressor
plt.scatter(y_test, y_pred_forest)
plt.xlabel("Actual Quality")
plt.ylabel("Predicted Quality")
plt.title("Actual vs. Predicted Quality (Random Forest Regressor)")
plt.show()
```

```
Linear Regression:
Mean Squared Error: 0.39002514396395416
R-squared: 0.4031803412796231

Decision Tree Regressor:
Mean Squared Error: 0.60625
R-squared: 0.07231130172297862

Random Forest Regressor:
Mean Squared Error: 0.30123812499999997
R-squared: 0.5390429623873638
```

#Decision Tree Regression Select your features (X) and the target variable (y). In this case: Features (X): fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol.

Target variable (y): quality.

```
#Train-Test Split
from sklearn.model_selection import train_test_split

# Select features (X) and target variable (y)
X = df.drop('quality', axis=1)
y = df['quality']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

#Model Training
'''Use scikit-learn to create a DecisionTreeRegressor and train it on
your training data.'''
from sklearn.tree import DecisionTreeRegressor

# Create a Decision Tree Regressor model
tree_model = DecisionTreeRegressor(random_state=42)
```

```
# Train the model on the training set
tree_model.fit(X_train, y_train)
```

```
DecisionTreeRegressor(random_state=42)
```

```
#Model Evalution
'''Make Predictions'''
# Make predictions on the test set
y_pred = tree_model.predict(X_test)

#Evaluate the Model
'''We use Regression metrics like R2 and MSE'''
from sklearn.metrics import mean_squared_error, r2_score


mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.60625
R-squared: 0.07231130172297862
```

- MSE:A lower MSE indicates better performance, and 0.60625 suggests that, on average, your predictions are 0.60625 units away from the actual values.
- R2: An R-squared value closer to 1 indicates a better fit, and 0.0723 suggests that your model explains only a small portion of the variance in the wine quality.

#The relatively high MSE and low R-squared value may indicate that the Decision Tree Regressor might not be capturing the underlying patterns in the data well.

#So we will try adjusting hyperparameters and performing cross-validation on a Decision Tree Regressor in scikit-learn can be done using the GridSearchCV or RandomizedSearchCV functions. These functions perform an exhaustive search over a specified parameter grid or a random search over parameter distributions, respectively.

max_depth: This parameter controls the maximum depth of the decision tree. A higher max_depth allows the tree to make more complex splits, potentially capturing more intricate patterns in the data. However, deeper trees can also lead to overfitting, especially with small datasets. Setting an appropriate max_depth is crucial to finding the right balance between model complexity and generalization.

min_samples_leaf: This parameter specifies the minimum number of samples required to be in a leaf node. A leaf node is a terminal node that represents the predicted value for a set of features. Setting a higher min_samples_leaf can prevent the tree from creating nodes with very few samples, which helps to control overfitting. It encourages the model to generalize patterns rather than memorizing noise in the training data.

min_samples_split: This parameter sets the minimum number of samples required to split an internal node. It controls the criteria for creating additional branches in the tree. A higher min_samples_split prevents the tree from making splits for a small number of samples, similar to min_samples_leaf. It can help in regularizing the model and preventing overfitting.

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Select features (X) and target variable (y)
X = df.drop('quality', axis=1)
y = df['quality']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create a Decision Tree Regressor model
tree_model = DecisionTreeRegressor(random_state=42)

# Define the hyperparameter grid for tuning
param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Use GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(tree_model, param_grid, cv=5,
scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print(f'Best Hyperparameters: {best_params}')

# Evaluate the model with the best hyperparameters
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Calculate performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Best Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 4,
'min_samples_split': 2}
Mean Squared Error: 0.4976069107860738
R-squared: 0.2385578435945852
```

#Lets analyze the results:

Earlier Values:

Mean Squared Error: 0.60625

R-squared: 0.07231130172297862

Current Values:

Mean Squared Error: 0.4976069107860738

R-squared: 0.2385578435945852

This indicates the results of hyperparameter tuning using GridSearchCV with a Decision Tree Regressor:

Best Hyperparameters:

- max_depth: 10
- min_samples_leaf: 4
- min_samples_split: 2

Mean Squared Error (MSE): 0.4976069107860738

- In this case, the average squared difference is approximately 0.4976, which is an improvement from the previous model (lower MSE is better).

R-squared: 0.2385578435945852

- The R-squared value measures the proportion of the variance in the dependent variable that is predictable from the independent variables. In this case, the model explains around 23.86% of the variance in wine quality.

#Random Forest Regression

```
from sklearn.ensemble import RandomForestRegressor
#Select features (X) and target variable (y)
X = df.drop('quality', axis=1)
y = df['quality']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create and train Random Forest Regressor model
forest_model = RandomForestRegressor(n_estimators=100,
random_state=42)
forest_model.fit(X_train, y_train)
y_pred_forest = forest_model.predict(X_test)

# Evaluate the models
models = [ ('Random Forest Regressor', y_pred_forest)]
```

```
for model_name, y_pred in models:
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f'{model_name}:')
    print(f'Mean Squared Error: {mse}')
    print(f'R-squared: {r2}\n')

# Visualize the predicted vs. actual values for Random Forest
Regressor
plt.scatter(y_test, y_pred_forest)
plt.xlabel("Actual Quality")
plt.ylabel("Predicted Quality")
plt.title("Actual vs. Predicted Quality (Random Forest Regressor)")
diagonal_line = np.linspace(min(min(y_test), min(y_pred)),
max(max(y_test), max(y_pred)), 100)
plt.plot(diagonal_line, diagonal_line, color='red', linewidth=2,
label='Regression Line (Diagonal)')
plt.show()
```

```
Random Forest Regressor:
Mean Squared Error: 0.30123812499999997
R-squared: 0.5390429623873638
```


Actual vs. Predicted Quality (Random Forest Regressor)
Predicted Quality
Actual Quality
3
4
5
6
7
8