November 10, 2024

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report,␣
 ↪confusion_matrix,accuracy_score
from sklearn.model_selection import GridSearchCV
from google.colab import files
```

[4]:
```
uploaded = files.upload()
```

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

```
Saving winequality-white.csv to winequality-white.csv
```

[5]:
```
red_wine = pd.read_csv('winequality-red.csv', delimiter=';')
white_wine = pd.read_csv('winequality-white.csv', delimiter=';')
```

[6]:
```
red_wine['wine_type'] = 'red'
white_wine['wine_type'] = 'white'
```

[7]:
```
print("Red Wine Dataset:")
print(red_wine.info())
print("\nWhite Wine Dataset:")
print(white_wine.info())
```

```
Red Wine Dataset:
<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  quality               1599 non-null   int64
 12  wine_type             1599 non-null   object
dtypes: float64(11), int64(1), object(1)
memory usage: 162.5+ KB
None

White Wine Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64
 12  wine_type             4898 non-null   object
dtypes: float64(11), int64(1), object(1)
memory usage: 497.6+ KB
None
```

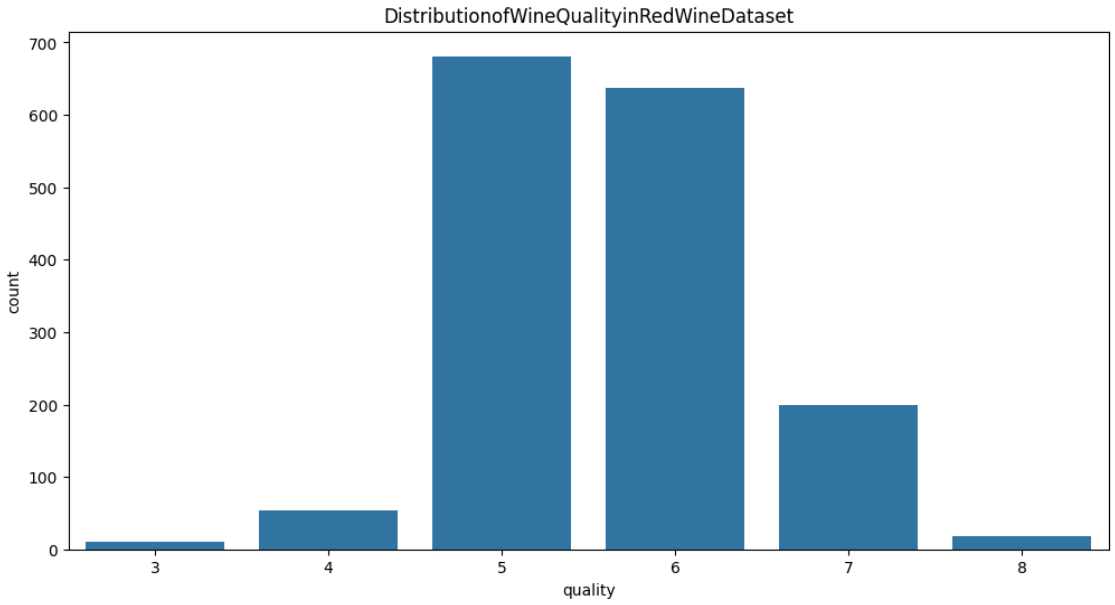
[8]:
```
print(red_wine.describe())
print()
print()
print(white_wine.describe())
```

```
       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
```

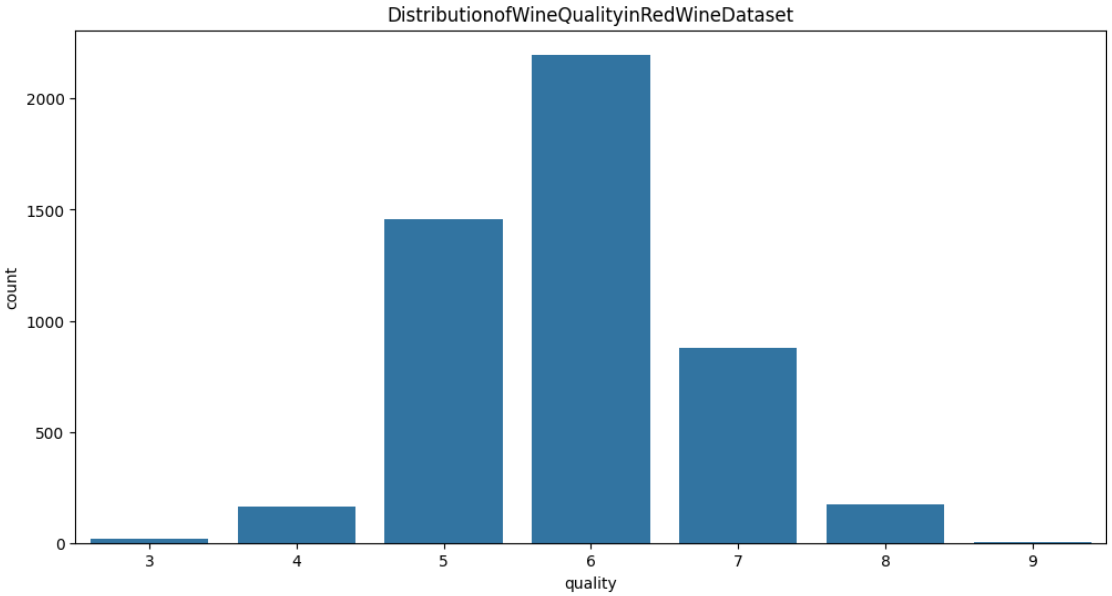
```
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         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


       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    4898.000000       4898.000000  4898.000000     4898.000000
mean        6.854788          0.278241     0.334192        6.391415
std         0.843868          0.100795     0.121020        5.072058
min         3.800000          0.080000     0.000000        0.600000
25%         6.300000          0.210000     0.270000        1.700000
50%         6.800000          0.260000     0.320000        5.200000
75%         7.300000          0.320000     0.390000        9.900000
max        14.200000          1.100000     1.660000       65.800000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  4898.000000          4898.000000           4898.000000  4898.000000
mean      0.045772            35.308085            138.360657     0.994027
std       0.021848            17.007137             42.498065     0.002991
min       0.009000             2.000000              9.000000     0.987110
25%       0.036000            23.000000            108.000000     0.991723
50%       0.043000            34.000000            134.000000     0.993740
75%       0.050000            46.000000            167.000000     0.996100
max       0.346000           289.000000            440.000000     1.038980

                pH    sulphates      alcohol      quality
count  4898.000000  4898.000000  4898.000000  4898.000000
mean      3.188267     0.489847    10.514267     5.877909
std       0.151001     0.114126     1.230621     0.885639
```

```
min        2.720000      0.220000      8.000000      3.000000
25%        3.090000      0.410000      9.500000      5.000000
50%        3.180000      0.470000     10.400000      6.000000
75%        3.280000      0.550000     11.400000      6.000000
max        3.820000      1.080000     14.200000      9.000000
```

[9]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=red_wine,x='quality')
plt.title("DistributionofWineQualityinRedWineDataset")
plt.show()
```

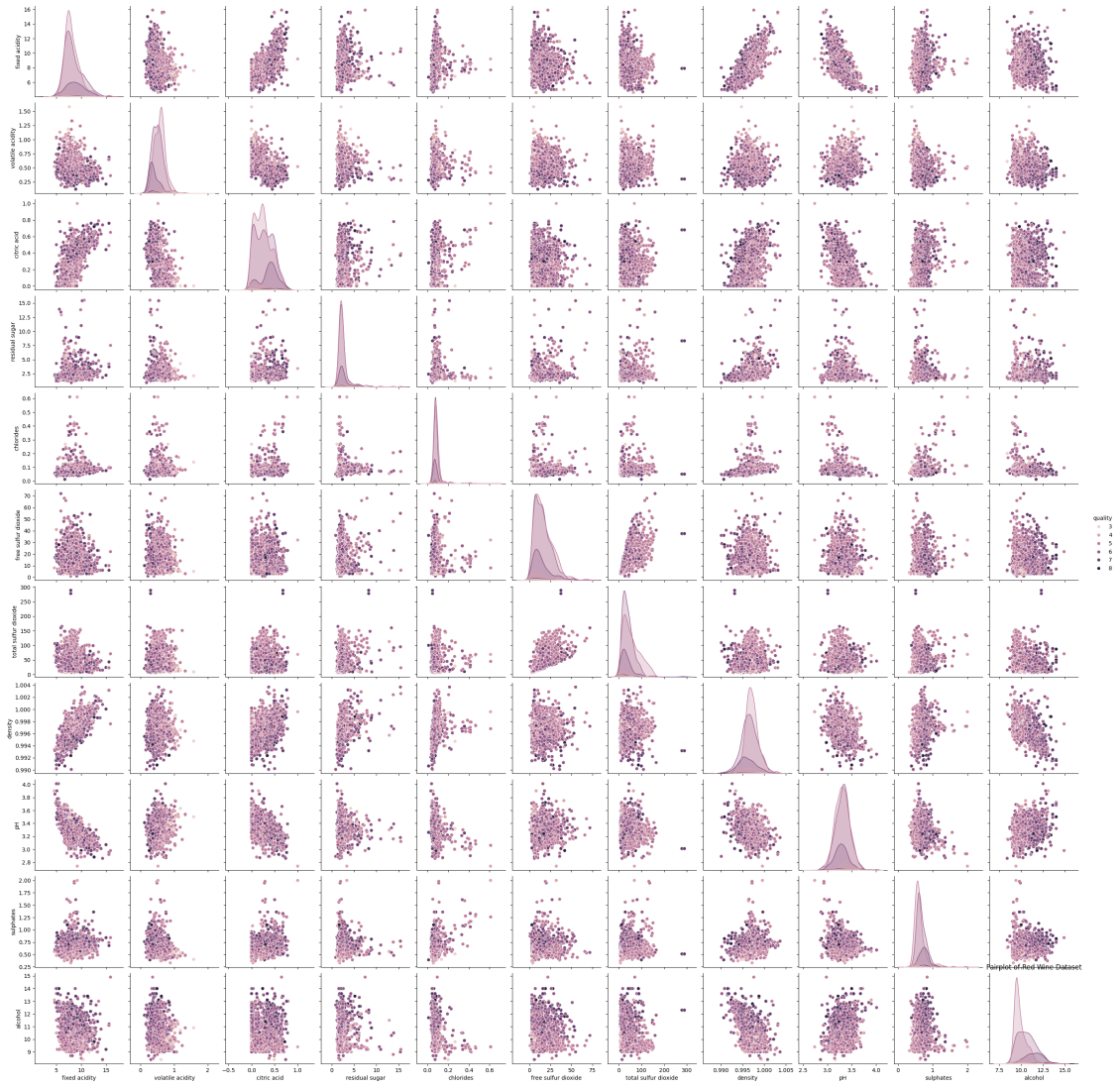


[10]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=white_wine,x='quality')
plt.title("DistributionofWineQualityinRedWineDataset")
plt.show()
```



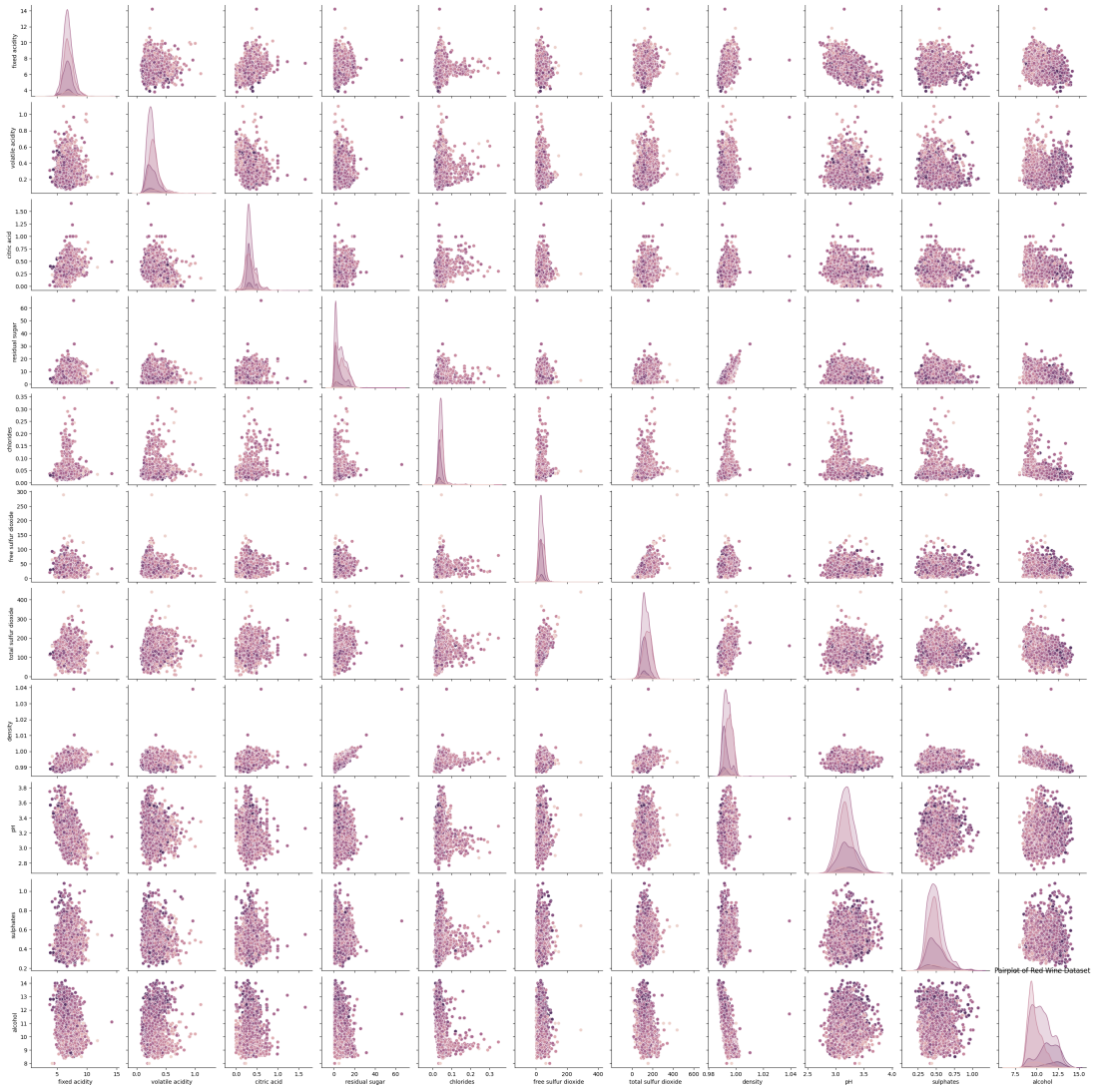
[11]:
```
plt.figure(figsize=(12, 6))
sns.pairplot(red_wine, hue='quality')
plt.title("Pairplot of Red Wine Dataset")
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



[12]:
```
plt.figure(figsize=(12, 6))
sns.pairplot(white_wine, hue='quality')
plt.title("Pairplot of Red Wine Dataset")
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



[13]:
```
# Remove null values
red_wine = red_wine.dropna()
white_wine = white_wine.dropna()
```

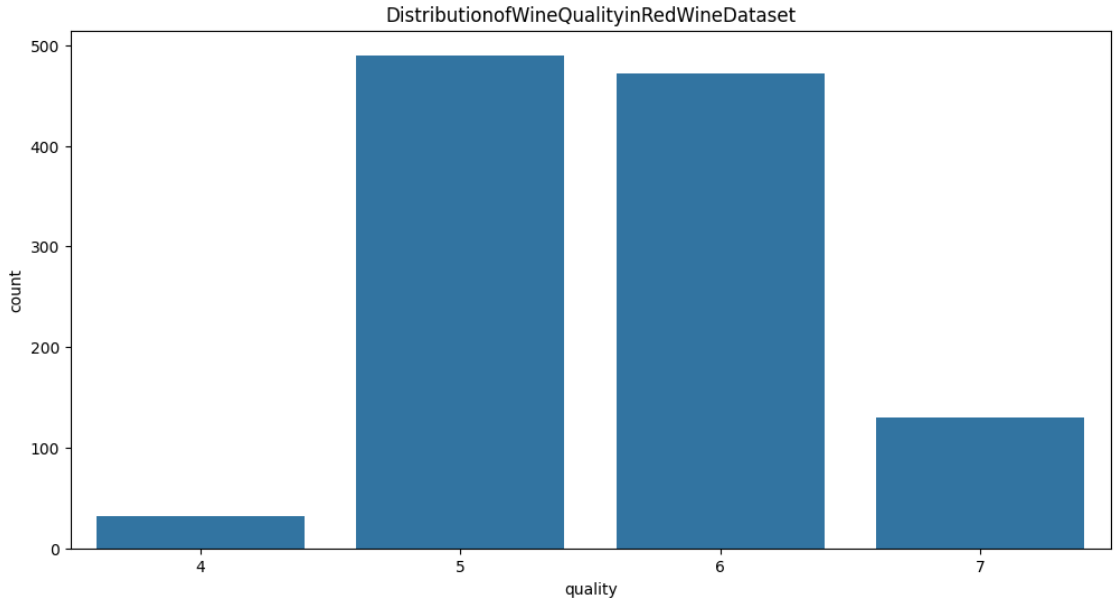
[14]:
```
# Check for missing values
print(red_wine.isnull().sum())
print(white_wine.isnull().sum())
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
```

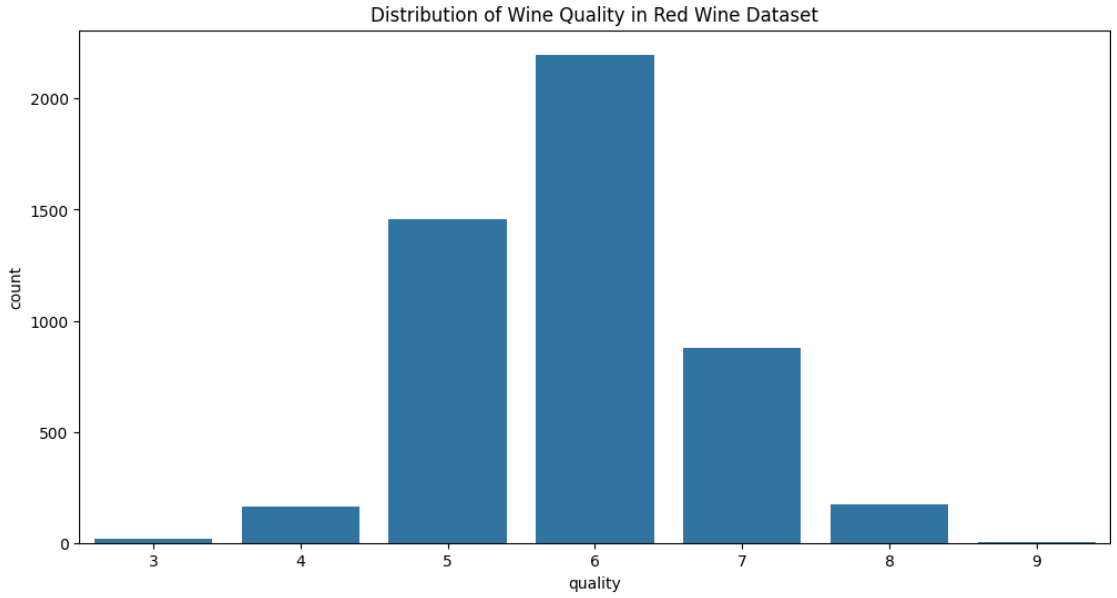
```
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
wine_type               0
dtype: int64
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar         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

[15]:
```
# Function to remove outliers based on IQR
def remove_outliers(df):
    for col in df.columns:
        if df[col].dtype in ['float64', 'int64'] and col != 'wine_type':
            Q1 = df[col].quantile(0.25)
            Q3 = df[col].quantile(0.75)
            IQR = Q3 - Q1
            df = df[~((df[col] < (Q1 - 1.5 * IQR)) | (df[col] > (Q3 + 1.5 *␣
 ↪IQR)))]
    return df

# Apply outlier removal
red_wine_cleaned = remove_outliers(red_wine)
white_wine_cleaned = remove_outliers(white_wine)
```

[16]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=red_wine_cleaned,x='quality')
plt.title("DistributionofWineQualityinRedWineDataset")
plt.show()
```



[17]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=white_wine, x='quality')
plt.title("Distribution of Wine Quality in Red Wine Dataset")
plt.show()
```



[18]:
```
red_wine_cleaned['quality'] = red_wine_cleaned['quality'].apply(lambda x: 1 if␣
↪x >= 7 else 0)
```

```
white_wine_cleaned['quality'] = white_wine_cleaned['quality'].apply(lambda x: 1␣
 ↪if x >= 7 else 0)
```

[19]:
```
# Initialize a scaler
scaler = StandardScaler()

# Scale red wine features
red_wine_features = red_wine_cleaned.drop(columns=['quality', 'wine_type'])
red_wine_scaled = pd.DataFrame(scaler.
 ↪fit_transform(red_wine_features),columns=red_wine_features.columns)
red_wine_scaled['quality'] = red_wine_cleaned['quality'].values
red_wine_scaled['wine_type'] = red_wine_cleaned['wine_type'].values

# Scale white wine features
white_wine_features = white_wine_cleaned.drop(columns=['quality', 'wine_type'])
white_wine_scaled = pd.DataFrame(scaler.
 ↪fit_transform(white_wine_features),columns=white_wine_features.columns)
white_wine_scaled['quality'] = white_wine_cleaned['quality'].values
white_wine_scaled['wine_type'] = white_wine_cleaned['wine_type'].values
```

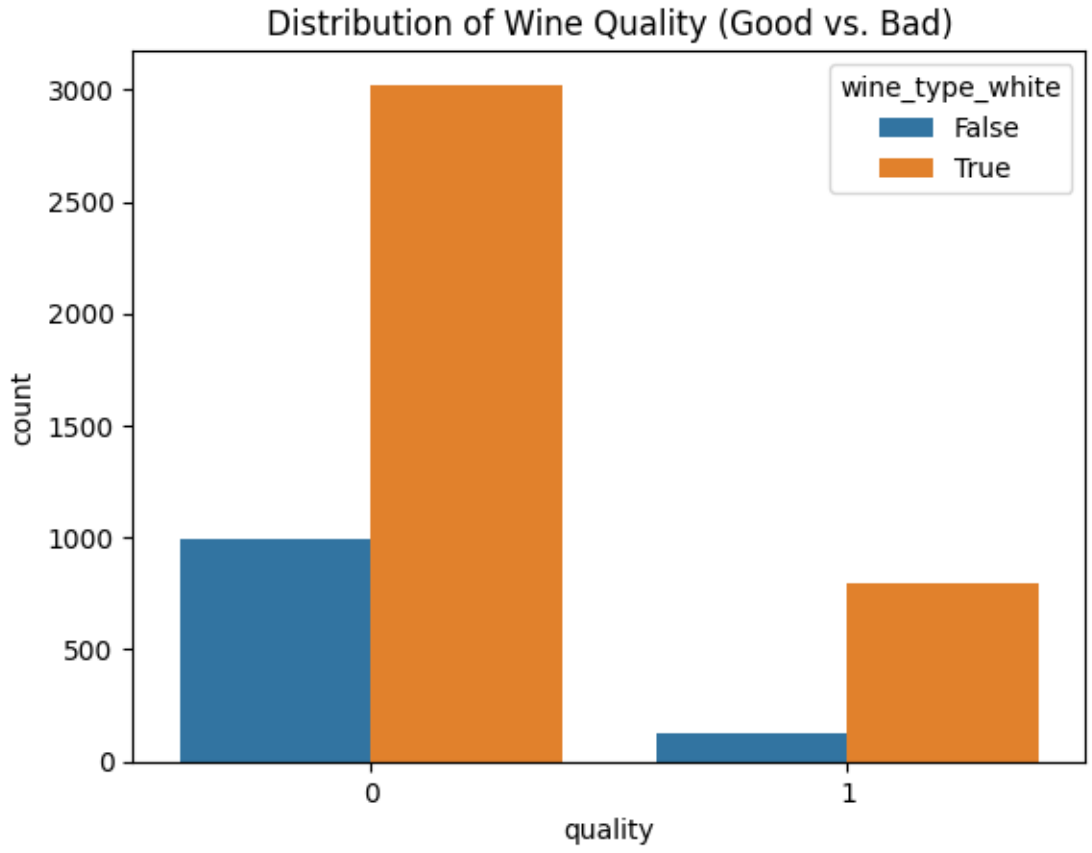
[20]:
```
# Combine the cleaned and scaled datasets
combined_data = pd.concat([red_wine_scaled, white_wine_scaled],␣
 ↪axis=0,ignore_index=True)

# One-hot encode 'wine_type'
combined_data = pd.get_dummies(combined_data,␣
 ↪columns=['wine_type'],drop_first=True)
```

[21]:
```
# Check combined data structure
print(combined_data.info())
```

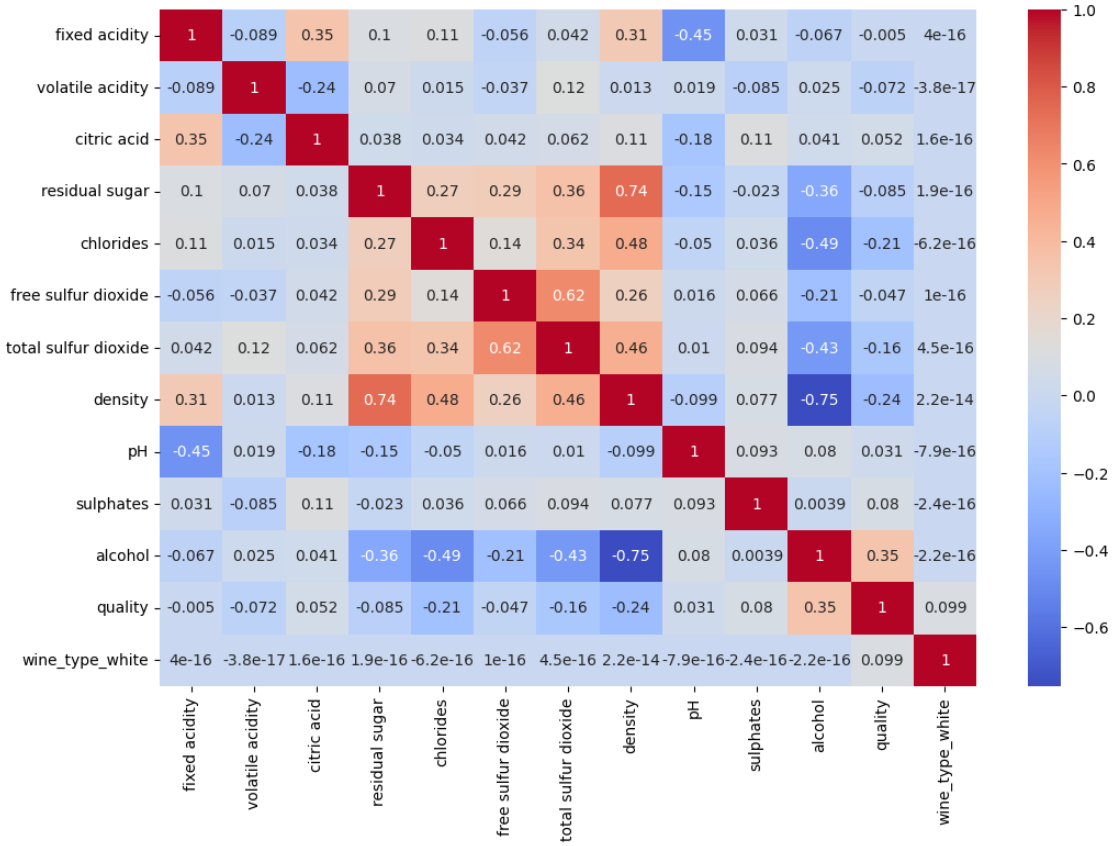
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4939 entries, 0 to 4938
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         4939 non-null   float64
 1   volatile acidity      4939 non-null   float64
 2   citric acid           4939 non-null   float64
 3   residual sugar        4939 non-null   float64
 4   chlorides             4939 non-null   float64
 5   free sulfur dioxide   4939 non-null   float64
 6   total sulfur dioxide  4939 non-null   float64
 7   density               4939 non-null   float64
 8   pH                    4939 non-null   float64
 9   sulphates             4939 non-null   float64
 10  alcohol               4939 non-null   float64
```

```
 11  quality               4939 non-null   int64
 12  wine_type_white       4939 non-null   bool
dtypes: bool(1), float64(11), int64(1)
memory usage: 468.0 KB
None
```

[22]:
```
# Visualize the distribution of the target variable
sns.countplot(data=combined_data, x='quality', hue='wine_type_white')
plt.title("Distribution of Wine Quality (Good vs. Bad)")
plt.show()
```



[23]:
```
# Correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(combined_data.corr(), annot=True, cmap='coolwarm')
plt.show()
```



[24]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix



# Define features and target
X = combined_data.drop('quality', axis=1)
y = combined_data['quality']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,random_state=42)

# Initialize models
models = {
```

```
 'Logistic Regression': LogisticRegression(class_weight='balanced'),
 'Decision Tree': DecisionTreeClassifier(class_weight='balanced'),
 'Random Forest': RandomForestClassifier(class_weight='balanced'),
 'SVM': SVC(class_weight='balanced')
 }

#Train and evaluate models
for name, model in models.items():
  model.fit(X_train,y_train)
  y_pred = model.predict(X_test)
  print(f"{name}-Accuracy:{accuracy_score(y_test,y_pred)}")
  print(confusion_matrix(y_test, y_pred))
  print(classification_report(y_test, y_pred))
```

```
Logistic Regression-Accuracy:0.7095141700404858
[[565 234]
 [ 53 136]]
              precision    recall  f1-score   support

           0       0.91      0.71      0.80       799
           1       0.37      0.72      0.49       189

    accuracy                           0.71       988
   macro avg       0.64      0.71      0.64       988
weighted avg       0.81      0.71      0.74       988

Decision Tree-Accuracy:0.8238866396761133
[[705  94]
 [ 80 109]]
              precision    recall  f1-score   support

           0       0.90      0.88      0.89       799
           1       0.54      0.58      0.56       189

    accuracy                           0.82       988
   macro avg       0.72      0.73      0.72       988
weighted avg       0.83      0.82      0.83       988

Random Forest-Accuracy:0.881578947368421
[[778  21]
 [ 96  93]]
              precision    recall  f1-score   support

           0       0.89      0.97      0.93       799
           1       0.82      0.49      0.61       189

    accuracy                           0.88       988
```

```
   macro avg       0.85      0.73      0.77       988
weighted avg       0.88      0.88      0.87       988

SVM-Accuracy:0.7388663967611336
[[577 222]
 [ 36 153]]
              precision    recall  f1-score   support

           0       0.94      0.72      0.82       799
           1       0.41      0.81      0.54       189

    accuracy                           0.74       988
   macro avg       0.67      0.77      0.68       988
weighted avg       0.84      0.74      0.76       988
```

[25]:
```
#Hyperparameter tuning forRandomForest
param_grid= {
'n_estimators':[50, 100,200],
    'max_depth':[None, 10, 20, 30]
 }

grid_rf= GridSearchCV(RandomForestClassifier(),param_grid,cv=5)
grid_rf.fit(X_train, y_train)

#Bestparametersandaccuracy
print("BestparametersforRandomForest:",grid_rf.best_params_)
y_pred_rf =grid_rf.predict(X_test)
print("TunedRandom Forest-Accuracy:",accuracy_score(y_test,y_pred_rf))
```

```
BestparametersforRandomForest: {'max_depth': 30, 'n_estimators': 100}
TunedRandom Forest-Accuracy: 0.8836032388663968
```

[26]:
```
#Model comparison
model_comparison ={
'Model':['LogisticRegression','DecisionTree', 'RandomForest','SVM'],
'Accuracy':[
  accuracy_score(y_test,models['Logistic Regression'].predict(X_test)),
  accuracy_score(y_test,models['Decision Tree'].predict(X_test)),
  accuracy_score(y_test,y_pred_rf), #TunedRandomForest
  accuracy_score(y_test,models['SVM'].predict(X_test))
 ]
}
comparison_df =pd.DataFrame(model_comparison)
print(comparison_df)
```

```
                Model  Accuracy
0  LogisticRegression  0.709514
```

```
1        DecisionTree  0.823887
2        RandomForest  0.883603
3                 SVM  0.738866
```