Wine Quality Prediction

Introduction

The **Wine Quality Prediction** aims to analyze and predict the quality of red variants of the Portuguese "Vinho Verde" wine based on physicochemical attributes. The dataset provides valuable insights into the chemical composition of wines and how these attributes influence the perceived quality.

Dataset

The dataset consists of the following attributes:

- **Input Variables (based on physicochemical tests):**
  1. Fixed acidity
  2. Volatile acidity
  3. Citric acid
  4. Residual sugar
  5. Chlorides
  6. Free sulfur dioxide
  7. Total sulfur dioxide
  8. Density
  9. pH
  10. Sulphates
  11. Alcohol
- **Output Variable (based on sensory data):**
  12. Quality (score between 0 and 10)

The project involves building classification models to predict wine quality based on these attributes. The dataset presents a challenging task due to its imbalanced and relatively small sample size.

Importing DataSet:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
```

```
1 df=pd.read_csv("/content/WineQT.csv")
```

```
1 df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphat
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0
...	...	...	...	...	...	...	...	...	...	
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0
1142	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0

```
1 df.head(10)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56

```
1 df.tail(10)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphat
1133	6.7	0.320	0.44	2.4	0.061	24.0	34.0	0.99484	3.29	0
1134	7.5	0.310	0.41	2.4	0.065	34.0	60.0	0.99492	3.34	0
1135	5.8	0.610	0.11	1.8	0.066	18.0	28.0	0.99483	3.55	0
1136	6.3	0.550	0.15	1.8	0.077	26.0	35.0	0.99314	3.32	0
1137	5.4	0.740	0.09	1.7	0.089	16.0	26.0	0.99402	3.67	0
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0

```
1 df.dtypes
```

```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
Id                        int64
dtype: object
```

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64  
 12  Id                    1143 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

```
1 df.shape
```

```
(1143, 13)
```

```
1 df.columns
```

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

```
1 set(df['quality'])
```

```
{3, 4, 5, 6, 7, 8}
```

```
1 df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide

```
df.describe(include='all')
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000
mean	8.311111	0.531339	0.268364	2.532152	0.086933	15.615486	45.914698
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32.782130
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21.000000
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37.000000
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61.000000

```
df.isnull()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulpha
0	False	False	False	False	False	False	False	False	False	F
1	False	False	False	False	False	False	False	False	False	F
2	False	False	False	False	False	False	False	False	False	F
3	False	False	False	False	False	False	False	False	False	F
4	False	False	False	False	False	False	False	False	False	F
...	...	...	...	...	...	...	...	...	...	
1138	False	False	False	False	False	False	False	False	False	F
1139	False	False	False	False	False	False	False	False	False	F
1140	False	False	False	False	False	False	False	False	False	F
1141	False	False	False	False	False	False	False	False	False	F
1142	False	False	False	False	False	False	False	False	False	F

```
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
Id                      0
dtype: int64
```

```
df.corr()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
fixed acidity	1.000000	-0.250728	0.673157	0.171831	0.107889	-0.164831	-0.110628	0.681501
volatile acidity	-0.250728	1.000000	-0.544187	-0.005751	0.056336	-0.001962	0.077748	0.016512
citric acid	0.673157	-0.544187	1.000000	0.175815	0.245312	-0.057589	0.036871	0.375243
residual sugar	0.171831	-0.005751	0.175815	1.000000	0.070863	0.165339	0.190790	0.380147
chlorides	0.107889	0.056336	0.245312	0.070863	1.000000	0.015280	0.048163	0.208901
free sulfur dioxide	-0.164831	-0.001962	-0.057589	0.165339	0.015280	1.000000	0.661093	-0.054150
total sulfur dioxide	-0.110628	0.077748	0.036871	0.190790	0.048163	0.661093	1.000000	0.050175
density	0.681501	0.016512	0.375243	0.380147	0.208901	-0.054150	0.050175	1.000000
...								

```
df.groupby('quality').mean()
```

In the dataset, each column represents a specific attribute related to wine quality. The values in these columns have no missing data, with no null values present (0 missing values) in any of the attributes. This clean dataset is ready for analysis and modeling.

`quality`

Due to the absence of missing data in the dataset's attributes, we have skipped the following steps:

1. **Data Imputation:** There was no need to fill or impute missing values since all columns are complete, saving time and complexity in data preprocessing.
2. **Missing Data Analysis:** The step of analyzing patterns or causes of missing data was unnecessary, as the dataset was devoid of any missing values.
3. **Handling Missing Values:** With no null values to address, there was no requirement for strategies such as imputation, removal of incomplete rows, or advanced techniques to manage missing data.

Exploratory Data Analysis (EDA):

1. Summary Statistics:

Function used: `DataFrame.describe`

```
# Display summary stats for all numerical columns:

summary_stats=df.describe(include='all')
print(summary_stats)
```

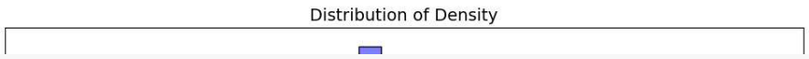
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1143.000000       1143.000000  1143.000000     1143.000000
mean        8.311111          0.531339     0.268364        2.532152
std         1.747595          0.179633     0.196686        1.355917
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.392500     0.090000        1.900000
50%         7.900000          0.520000     0.250000        2.200000
75%         9.100000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1143.000000          1143.000000           1143.000000  1143.000000
mean      0.086933            15.615486             45.914698     0.996730
std       0.047267            10.250486             32.782130     0.001925
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             21.000000     0.995570
50%       0.079000            13.000000             37.000000     0.996680
75%       0.090000            21.000000             61.000000     0.997845
max       0.611000            68.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality           Id
count  1143.000000  1143.000000  1143.000000  1143.000000  1143.000000
mean      3.311015     0.657708    10.442111     5.657043   804.969379
std       0.156664     0.170399     1.082196     0.805824   463.997116
min       2.740000     0.330000     8.400000     3.000000     0.000000
25%       3.205000     0.550000     9.500000     5.000000   411.000000
50%       3.310000     0.620000    10.200000     6.000000   794.000000
75%       3.400000     0.730000    11.100000     6.000000  1209.500000
max       4.010000     2.000000    14.900000     8.000000  1597.000000
```

2. Histograms:

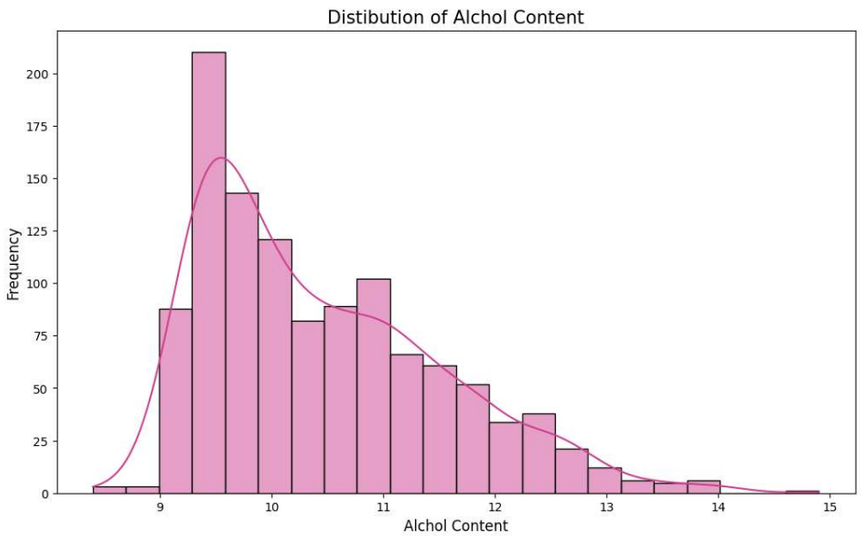
Function used: `seaborn.histplot()`

Histogram with KDE for the 'density' attribute from the provided DataFrame (df), visualizing its distribution. The x-axis represents the density values, the y-axis represents the frequency of those values, and a smooth KDE curve is overlaid on the histogram. The plot is presented with appropriate labels and a title for better interpretation.

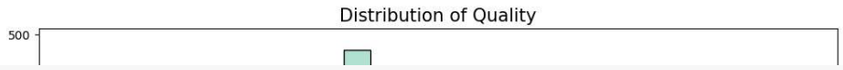
```
plt.figure(figsize=(12, 6))
sns.histplot(data=df, x='density', kde=True, color='blue')
plt.xlabel('Density', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Distribution of Density', fontsize=14)
plt.show()
```



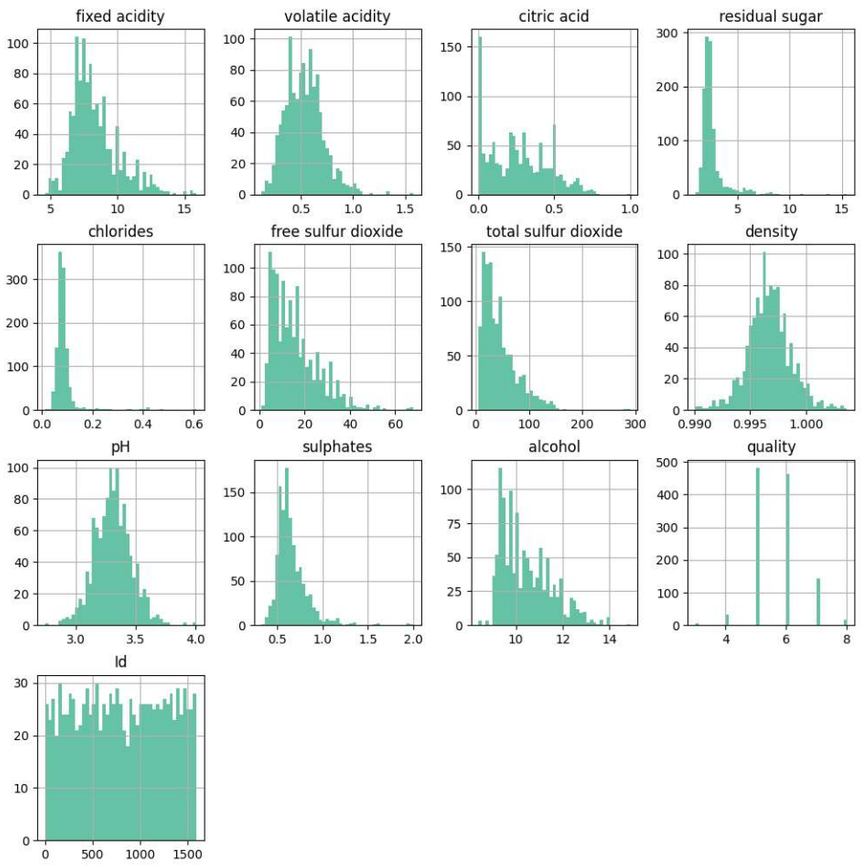

```
from seaborn.widgets import color_palette
plt.figure(figsize=(12, 7))

# Set the color palette to 'deep'
sns.set_palette("PiYG")

sns.histplot(data=df, x='alcohol', kde=True)
plt.xlabel('Alchol Content', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Distibution of Alchol Content', fontsize=15)
plt.show()
```




```
# Import necessary libraries
from seaborn.widgets import color_palette
import matplotlib.pyplot as plt

# Create a new figure for plotting with specified dimensions
plt.figure(figsize=(12, 7))

# Set the color palette to 'Set2' (a specific Seaborn color palette)
sns.set_palette("Set2")

# Create a histogram with KDE (Kernel Density Estimation) for the 'quality' attribute
sns.histplot(data=df, x='quality', kde=True)

# Add label for the x-axis
plt.xlabel('Quality', fontsize=12)

# Add label for the y-axis
plt.ylabel('Frequency', fontsize=12)

# Add a title to the plot
plt.title('Distribution of Quality', fontsize=15)

# Display the plot
plt.show()
```



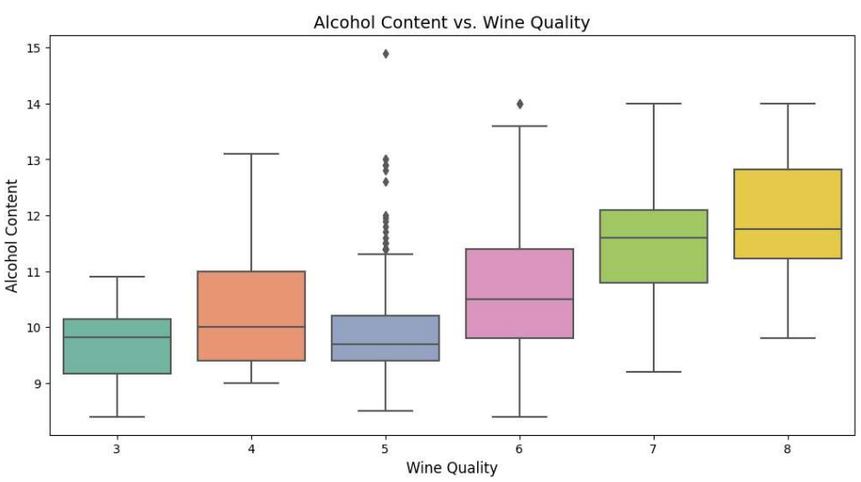

```
df.hist(figsize=(12,12), bins=45)
plt.show()
```



3. Box Plots

Function: `seaborn.boxplot()`

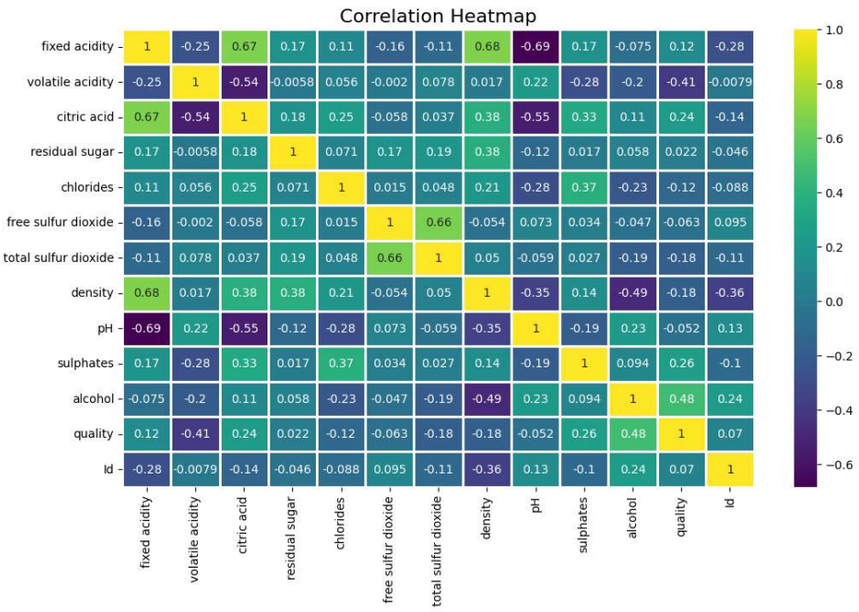
```
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='quality', y='alcohol', palette='Set2')
plt.xlabel('Wine Quality', fontsize=12)
plt.ylabel('Alcohol Content', fontsize=12)
plt.title('Alcohol Content vs. Wine Quality', fontsize=14)
plt.show()
```



4. Correlation Matrix (Heatmap)

Function: `seaborn.heatmap()`

```
cm=df.corr()
plt.figure(figsize=(12, 7))
sns.heatmap(cm, annot=True, cmap='viridis', linewidths=0.8)
plt.title('Correlation Heatmap', fontsize=16)
plt.show()
```

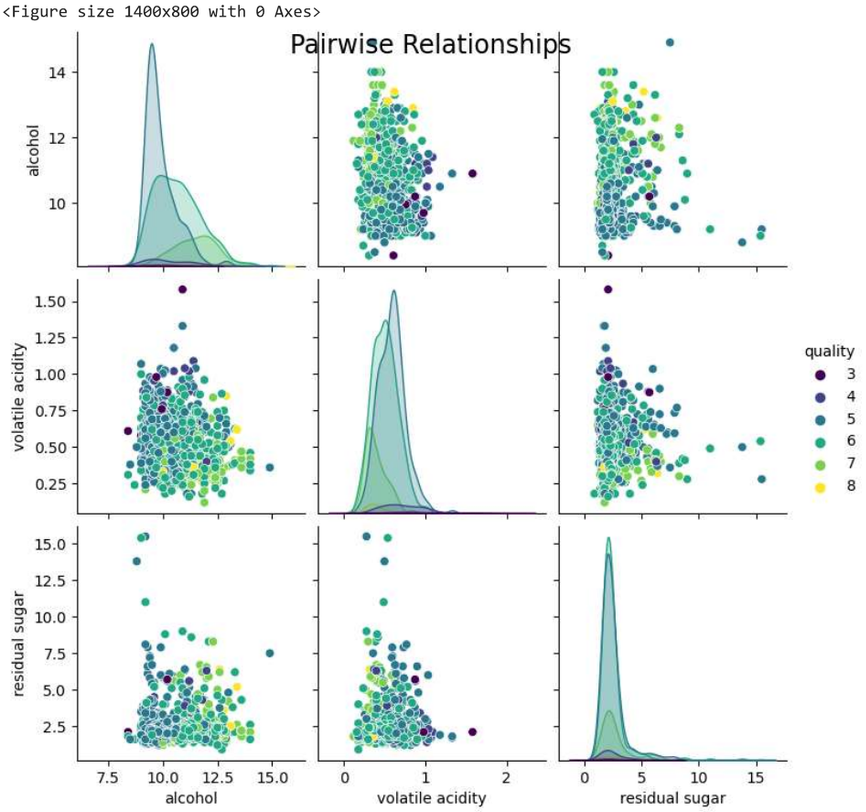


5. Pair Plots

Function: `seaborn.pairplot()`

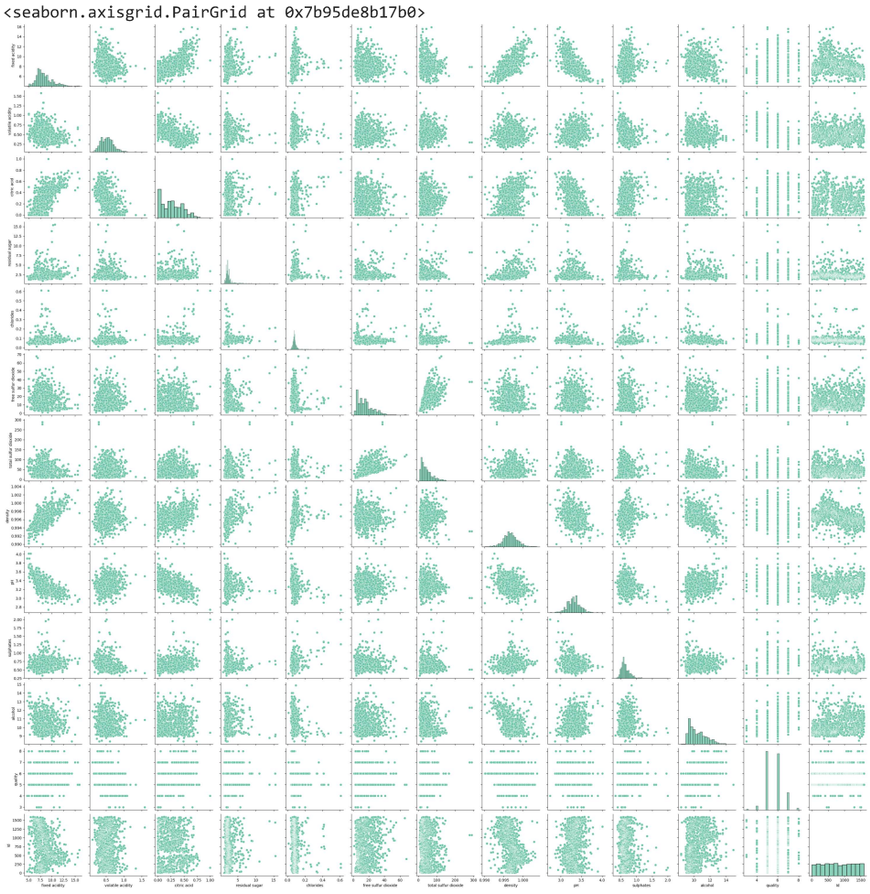
```
plt.figure(figsize=(14, 8))

# Create a pair plot with specified variables and color palette
pair_plot = sns.pairplot(data=df, vars=['alcohol', 'volatile acidity', 'residual sugar'], hue='quality', palette='viridis')

# Add a title to the plot
pair_plot.fig.suptitle('Pairwise Relationships', fontsize=17)

# Display the pair plot
plt.show()
```

```
<Figure size 1400x800 with 0 Axes>
```

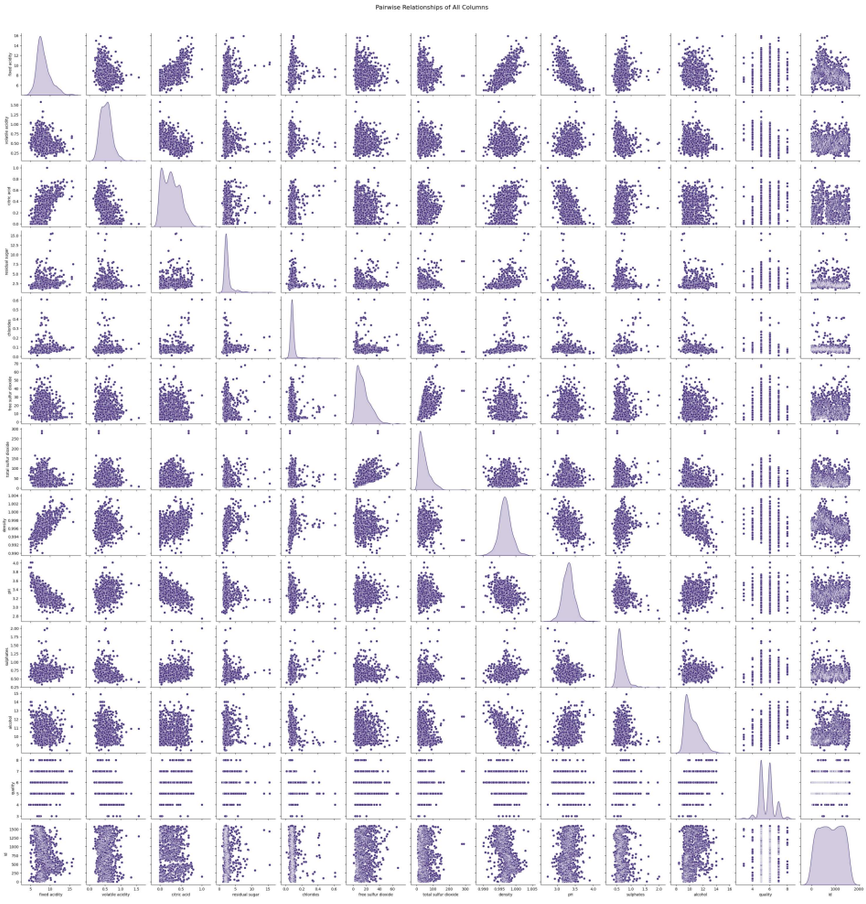


```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7b95de8b17b0>
```



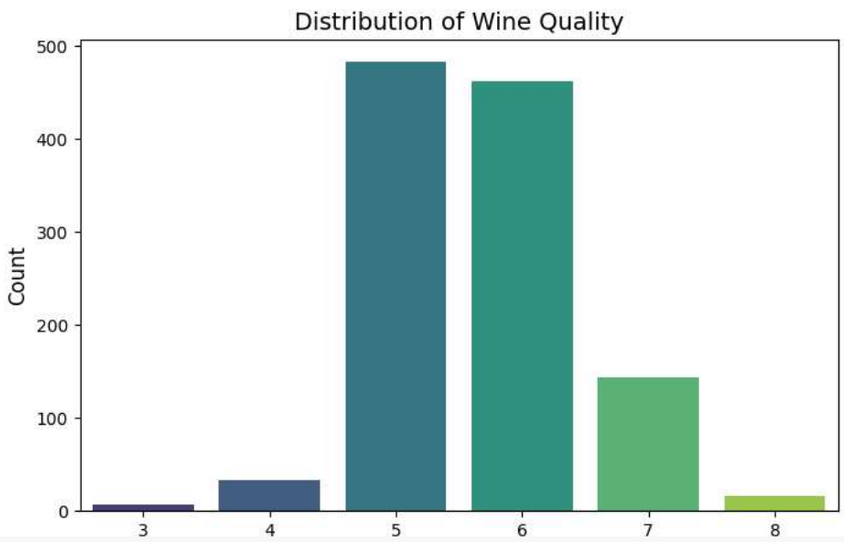
```
# Set a default color palette
sns.set_palette('viridis')

# Create a pair plot with all columns from the DataFrame df
sns.pairplot(data=df, diag_kind='kde')

# Add a title to the pair plot
plt.suptitle("Pairwise Relationships of All Columns", y=1.02, fontsize=16)

# Display the pair plot
plt.show()
```



6. Countplots and Bar Charts

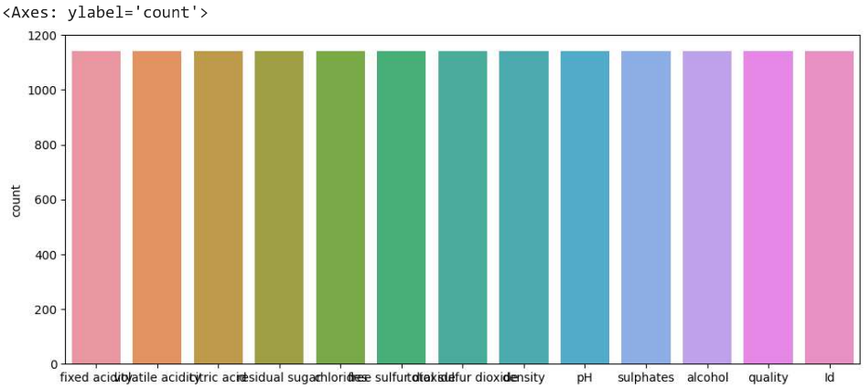
Function: `seaborn.countplot()`

```
# Create a new figure for the plot with specified dimensions
plt.figure(figsize=(8, 5))

# Create the count plot using Seaborn
sns.countplot(data=df, x='quality', palette='viridis')

# Add a label for the x-axis
plt.xlabel('Wine Quality', fontsize=12)

# Add a label for the y-axis
plt.ylabel('Count', fontsize=12)

# Add a title to the plot
plt.title('Distribution of Wine Quality', fontsize=14)

# Display the plot
plt.show()
```

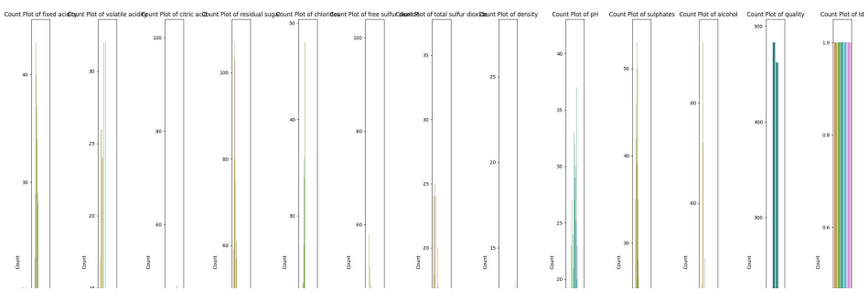



```
plt.figure(figsize=(12,5))
sns.countplot(df)
```

```
<Axes: ylabel='count'>
```




```
# Create subplots for each column in the DataFrame
fig, axes = plt.subplots(1, len(df.columns), figsize=(25, 15))

# Iterate through the columns and create count plots
for i, column in enumerate(df.columns):
    sns.countplot(data=df, x=column, ax=axes[i])
    axes[i].set_title(f'Count Plot of {column}')
    axes[i].set_xlabel('')
    axes[i].set_ylabel('Count')

# Adjust spacing between subplots
plt.tight_layout()

# Show the count plots
plt.show()
```



Ridge and Lasso Regression:

Load and Prepare the Data


```
X=df.drop(columns='quality',axis=1)
y=df['quality']
```

```
print(X.shape,y.shape)
```

```
(1143, 12) (1143,)
```

```
X.columns
```

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

Split the features into (X,Y)

```
from sklearn.model_selection import train_test_split

# Split the data 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)
```

we use `train_test_split` from scikit-learn to split the data into training and testing sets. In this example, 80% of the data is used for training, and 20% for testing.

Standardize the Features

```
from sklearn.preprocessing import StandardScaler

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Ridge Regression:

```
from sklearn.linear_model import Ridge

# Ridge Regression
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
y_pred_ridge = ridge.predict(X_test)
```

Lasso Regression

```
from sklearn.linear_model import Lasso

# Lasso Regression
lasso = Lasso(alpha=1.0)  # You can adjust the alpha (regularization strength) as needed
lasso.fit(X_train, y_train)
y_pred_lasso = lasso.predict(X_test)
```

```
lasso_predictions=y_pred_lasso
ridge_predictions=y_pred_ridge
```

Evaluate Ridge Regression

```
from sklearn.metrics import mean_squared_error, r2_score

# Evaluate Ridge Regression
ridge_mse = mean_squared_error(y_test, ridge_predictions)
ridge_rmse = np.sqrt(mean_squared_error(y_test, y_pred_ridge))
ridge_r2 = r2_score(y_test, y_pred_ridge)
print("Ridge Regression MSE:", ridge_mse)
print("Ridge Regression RMSE:", ridge_rmse)
print("Ridge Regression R^2:", ridge_r2)
```

```
Ridge Regression MSE: 0.3822807848629102
Ridge Regression RMSE: 0.6182885935086545
Ridge Regression R^2: 0.3130290371120601
```

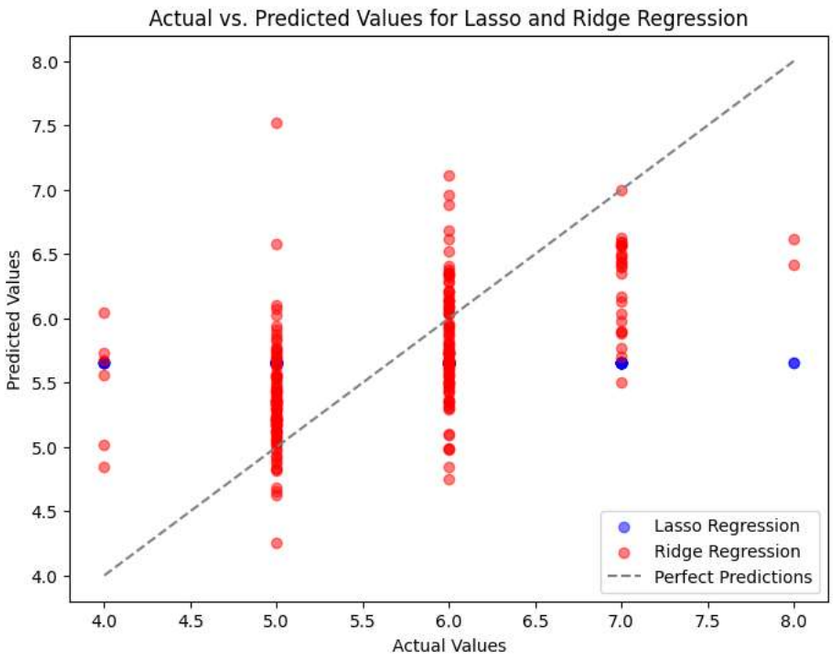
Evaluate Lasso Regression

```
# Evaluate Lasso Regression
lasso_rmse = np.sqrt(mean_squared_error(y_test, y_pred_lasso))
lasso_r2 = r2_score(y_test, y_pred_lasso)
lasso_mse = mean_squared_error(y_test, lasso_predictions)
print("Lasso Regression MSE:", lasso_mse)
print("Lasso Regression RMSE:", lasso_rmse)
print("Lasso Regression R^2:", lasso_r2)
```

```
Lasso Regression MSE: 0.556481594138102
Lasso Regression RMSE: 0.7459769394144179
Lasso Regression R^2: -1.5464265512799003e-05
```

```
# Actual target values
y_actual = y_test

# Predicted values for Lasso and Ridge
y_pred_lasso = y_pred_lasso
y_pred_ridge = y_pred_ridge
```

```
plt.figure(figsize=(8, 6))

# Plot Lasso predictions
plt.scatter(y_actual, y_pred_lasso, color='blue', label='Lasso Regression', alpha=0.5)

# Plot Ridge predictions
plt.scatter(y_actual, y_pred_ridge, color='red', label='Ridge Regression', alpha=0.5)

# Add a reference line for perfect predictions (y_actual = y_pred)
plt.plot([min(y_actual), max(y_actual)], [min(y_actual), max(y_actual)], linestyle='--', color='gray', label='Perfect Predictions')

# Customize the plot
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values for Lasso and Ridge Regression')
plt.legend()

# Show the plot
plt.show()
```



Decision Tree Model

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Data Modeling:

```
from sklearn.tree import DecisionTreeRegressor
model= DecisionTreeRegressor()
model.fit(X_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
decision_tree_model = DecisionTreeRegressor()
decision_tree_model.fit(X_train, y_train)
decision_tree_predictions = decision_tree_model.predict(X_test)

# Calculate MSE and R-squared for Decision Tree Regressor
dt_mse = mean_squared_error(y_test, decision_tree_predictions)
dt_r2 = r2_score(y_test, decision_tree_predictions)
```

Model Predictions:

```
y_pred = model.predict(X_test)
```

```
y_pred
```

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

Decision Tree:

```
from sklearn import tree
plt.figure(figsize=(20,20))
tree.plot_tree(model, filled=True)
```

0.129\nsamples = 144\nvalue = 5.097'),
 Text(0.04864016736401674, 0.5526315789473685, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.05700836820083682, 0.5526315789473685, 'x[1] <= 1.022\nsquared_error =
0.125\nsamples = 143\nvalue = 5.091'),
 Text(0.05282426778242678, 0.5, 'x[2] <= 0.235\nsquared_error = 0.12\nsamples =
142\nvalue = 5.085'),
 Text(0.0198744769874477, 0.4473684210526316, 'x[1] <= 0.47\nsquared_error =
0.118\nsamples = 88\nvalue = 5.136'),
 Text(0.008368200836820083, 0.39473684210526316, 'x[6] <= 27.5\nsquared_error =
0.24\nsamples = 5\nvalue = 5.6'),
 Text(0.004184100418410415, 0.34210526315789475, 'squared_error = 0.0\nsamples =
2\nvalue = 5.0'),
 Text(0.012552301255230125, 0.34210526315789475, 'squared_error = 0.0\nsamples = 3\nvalue
= 6.0'),
 Text(0.03138075313807531, 0.39473684210526316, 'x[3] <= 2.25\nsquared_error =
0.097\nsamples = 83\nvalue = 5.108'),
 Text(0.02092050209205021, 0.34210526315789475, 'x[10] <= 10.35\nsquared_error =
0.02\nsamples = 49\nvalue = 5.02'),
 Text(0.016736401673640166, 0.2894736842105263, 'squared_error = 0.0\nsamples = 48\nvalue
= 5.0'),
 Text(0.02510460251046025, 0.2894736842105263, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.04184100418410042, 0.34210526315789475, 'x[4] <= 0.076\nsquared_error =
0.18\nsamples = 34\nvalue = 5.235'),
 Text(0.03347280334728033, 0.2894736842105263, 'x[7] <= 0.997\nsquared_error =
0.188\nsamples = 4\nvalue = 5.75'),
 Text(0.029288702928870293, 0.23684210526315788, 'squared_error = 0.0\nsamples = 1\nvalue
= 5.0'),
 Text(0.03765690376569038, 0.23684210526315788, 'squared_error = 0.0\nsamples = 3\nvalue
= 6.0'),
 Text(0.0502092050209205, 0.2894736842105263, 'x[8] <= 3.53\nsquared_error =
0.139\nsamples = 30\nvalue = 5.167'),
 Text(0.04602510460251046, 0.23684210526315788, 'x[2] <= 0.175\nsquared_error =
0.119\nsamples = 29\nvalue = 5.138'),
 Text(0.03765690376569038, 0.18421052631578946, 'x[0] <= 9.15\nsquared_error =
0.076\nsamples = 24\nvalue = 5.083'),
 Text(0.03347280334728033, 0.13157894736842105, 'x[11] <= 215.5\nsquared_error =
0.042\nsamples = 23\nvalue = 5.043'),
 Text(0.029288702928870293, 0.07894736842105263, 'x[5] <= 7.0\nsquared_error =
0.25\nsamples = 2\nvalue = 5.5'),
 Text(0.02510460251046025, 0.02631578947368421, 'squared_error = 0.0\nsamples = 1\nvalue
= 6.0'),
 Text(0.03347280334728033, 0.02631578947368421, 'squared_error = 0.0\nsamples = 1\nvalue
= 5.0'),
 Text(0.03765690376569038, 0.07894736842105263, 'squared_error = 0.0\nsamples = 21\nvalue
= 5.0'),
 Text(0.04184100418410042, 0.13157894736842105, 'squared_error = 0.0\nsamples = 1\nvalue
= 6.0'),
 Text(0.05439330543933055, 0.18421052631578946, 'x[10] <= 9.5\nsquared_error =
0.24\nsamples = 5\nvalue = 5.4'),
 Text(0.0502092050209205, 0.13157894736842105, 'squared_error = 0.0\nsamples = 3\nvalue =
5.0'),
 Text(0.058577405857740586, 0.13157894736842105, 'squared_error = 0.0\nsamples = 2\nvalue
= 6.0'),
 Text(0.05439330543933055, 0.23684210526315788, 'squared_error = 0.0\nsamples = 1\nvalue
= 6.0'),
 Text(0.08577405857740586, 0.4473684210526316, 'x[8] <= 3.305\nsquared_error =
0.111\nsamples = 54\nvalue = 5.0'),
 Text(0.07531380753138076, 0.39473684210526316, 'x[4] <= 0.074\nsquared_error =
0.075\nsamples = 37\nvalue = 5.081'),
 Text(0.07112970711297072, 0.34210526315789475, 'x[10] <= 9.95\nsquared_error =
0.245\nsamples = 7\nvalue = 5.429'),
 Text(0.06694560669456066, 0.2894736842105263, 'x[1] <= 0.285\nsquared_error =
0.188\nsamples = 4\nvalue = 5.75'),
 Text(0.06276150627615062, 0.23684210526315788, 'squared_error = 0.0\nsamples = 1\nvalue
= 5.0'),
 Text(0.07112970711297072, 0.23684210526315788, 'squared_error = 0.0\nsamples = 3\nvalue
= 6.0'),
 Text(0.07531380753138076, 0.2894736842105263, 'squared_error = 0.0\nsamples = 3\nvalue =
5.0'),
 Text(0.0794979079497908, 0.34210526315789475, 'squared_error = 0.0\nsamples = 30\nvalue
= 5.0'),
 Text(0.09623430962343096, 0.39473684210526316, 'x[10] <= 9.9\nsquared_error =
0.145\nsamples = 17\nvalue = 4.824'),
 Text(0.08786610878661087, 0.34210526315789475, 'x[8] <= 3.32\nsquared_error =
0.066\nsamples = 14\nvalue = 4.929'),
 Text(0.08368200836820083, 0.2894736842105263, 'squared_error = 0.0\nsamples = 1\nvalue =
4.0'),
 Text(0.09205020920502092, 0.2894736842105263, 'squared_error = 0.0\nsamples = 13\nvalue
= 5.0'),
 Text(0.10460251046025104, 0.34210526315789475, 'x[5] <= 5.5\nsquared_error =
0.222\nsamples = 3\nvalue = 4.333'),
 Text(0.100418410041841, 0.2894736842105263, 'squared_error = 0.0\nsamples = 1\nvalue =
5.0'),
 Text(0.1087866108786611, 0.2894736842105263, 'squared_error = 0.0\nsamples = 2\nvalue =
4.0'),
 Text(0.06119246861924686, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 6.0'),
 Text(0.22757583682008367, 0.7105263157894737, 'x[11] <= 1358.0\nsquared_error =
0.304\nsamples = 150\nvalue = 5.34'),
 Text(0.1978294979079498, 0.6578947368421053, 'x[6] <= 64.5\nsquared_error =
0.269\nsamples = 134\nvalue = 5.396'),
 Text(0.16344142259414227, 0.6052631578947368, 'x[4] <= 0.082\nsquared_error =
0.298\nsamples = 83\nvalue = 5.518'),
 Text(0.14278242677824268, 0.5526315789473685, 'x[3] <= 2.45\nsquared_error =
0.316\nsamples = 47\nvalue = 5.362'),
 Text(0.12866108786610878, 0.5, 'x[3] <= 1.85\nsquared_error = 0.297\nsamples = 40\nvalue
= 5.45'),
 Text(0.11297071129707113, 0.4473684210526316, 'x[5] <= 15.0\nsquared_error =
0.13\nsamples = 13\nvalue = 5.154'),
 Text(0.1087866108786611, 0.39473684210526316, 'squared_error = 0.0\nsamples = 10\nvalue
= 5.0'),
 Text(0.11715481171548117, 0.39473684210526316, 'x[4] <= 0.073\nsquared_error =
0.222\nsamples = 3\nvalue = 5.667'),
 Text(0.11297071129707113, 0.34210526315789475, 'squared_error = 0.0\nsamples = 1\nvalue
= 5.0'),
 Text(0.12133891213389121, 0.34210526315789475, 'squared_error = 0.0\nsamples = 2\nvalue
= 6.0'),
 Text(0.14435146443514643, 0.4473684210526316, 'x[2] <= 0.045\nsquared_error =
0.316\nsamples = 27\nvalue = 5.593'),
 Text(0.13389121338912133, 0.39473684210526316, 'x[11] <= 29.5\nsquared_error =
0.139\nsamples = 6\nvalue = 6.167'),
 Text(0.1297071129707113, 0.34210526315789475, 'squared_error = 0.0\nsamples = 1\nvalue =
7.0'),
 Text(0.13807531380753138, 0.34210526315789475, 'squared_error = 0.0\nsamples = 5\nvalue
= 6.0'),

```
 Text(0.15481171548117154, 0.39473684210526316, 'x[9] <= 0.585\nsquared_error =
0.245\nsamples = 21\nvalue = 5.429'),
 Text(0.14644351464435146, 0.34210526315789475, 'x[1] <= 0.703\nsquared_error =
0.122\nsamples = 7\nvalue = 5.857'),
 Text(0.14225941422594143, 0.2894736842105263, 'squared_error = 0.0\nsamples = 6\nvalue =
6.0'),
 Text(0.1506276150627615, 0.2894736842105263, 'squared_error = 0.0\nsamples = 1\nvalue =
5.0'),
 Text(0.16317991631799164, 0.34210526315789475, 'x[4] <= 0.08\nsquared_error =
0.168\nsamples = 14\nvalue = 5.214'),
 Text(0.1589958158995816, 0.2894736842105263, 'x[7] <= 0.999\nsquared_error =
0.076\nsamples = 12\nvalue = 5.083'),
 Text(0.15481171548117154, 0.23684210526315788, 'squared_error = 0.0\nsamples = 11\nvalue
= 5.0'),
 Text(0.16317991631799164, 0.23684210526315788, 'squared_error = 0.0\nsamples = 1\nvalue
= 6.0'),
 Text(0.16736401673640167, 0.2894736842105263, 'squared_error = 0.0\nsamples = 2\nvalue =
6.0'),
 Text(0.15690376569037656, 0.5, 'x[0] <= 12.2\nsquared_error = 0.122\nsamples = 7\nvalue
= 4.857'),
 Text(0.15271966527196654, 0.4473684210526316, 'squared_error = 0.0\nsamples = 6\nvalue =
5.0'),
 Text(0.16108786610878661, 0.4473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue =
4.0'),
 Text(0.18410041841004185, 0.5526315789473685, 'x[8] <= 3.165\nsquared_error =
0.201\nsamples = 36\nvalue = 5.722'),
 Text(0.17364016736401675, 0.5, 'x[1] <= 0.76\nsquared_error = 0.16\nsamples = 5\nvalue =
5.2'),
 Text(0.1694560669456067, 0.4473684210526316, 'squared_error = 0.0\nsamples = 4\nvalue =
5.0'),
 Text(0.17782426778242677, 0.4473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.19456066945606695, 0.5, 'x[7] <= 0.996\nsquared_error = 0.156\nsamples =
31\nvalue = 5.806'),
 Text(0.18619246861924685, 0.4473684210526316, 'x[11] <= 1017.5\nsquared_error =
0.188\nsamples = 4\nvalue = 5.25'),
 Text(0.18200836820083682, 0.39473684210526316, 'squared_error = 0.0\nsamples = 3\nvalue
= 5.0'),
 Text(0.1903765690376569, 0.39473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.20292887029288703, 0.4473684210526316, 'x[7] <= 0.998\nsquared_error =
0.099\nsamples = 27\nvalue = 5.889'),
 Text(0.19874476987447698, 0.39473684210526316, 'squared_error = 0.0\nsamples = 17\nvalue
= 6.0'),
 Text(0.20711297071129708, 0.39473684210526316, 'x[8] <= 3.34\nsquared_error =
0.21\nsamples = 10\nvalue = 5.7'),
 Text(0.20292887029288703, 0.34210526315789475, 'x[4] <= 0.089\nsquared_error =
0.109\nsamples = 8\nvalue = 5.875'),
 Text(0.19874476987447698, 0.2894736842105263, 'squared_error = 0.0\nsamples = 1\nvalue =
5.0'),
 Text(0.20711297071129708, 0.2894736842105263, 'squared_error = 0.0\nsamples = 7\nvalue =
6.0'),
 Text(0.2112970711297071, 0.34210526315789475, 'squared_error = 0.0\nsamples = 2\nvalue =
5.0'),
 Text(0.23221757322175732, 0.6052631578947368, 'x[9] <= 0.625\nsquared_error =
0.158\nsamples = 51\nvalue = 5.196'),
 Text(0.2280334728033473, 0.5526315789473685, 'x[11] <= 1310.5\nsquared_error =
0.125\nsamples = 48\nvalue = 5.146'),
 Text(0.22384937238493724, 0.5, 'x[8] <= 3.57\nsquared_error = 0.081\nsamples = 45\nvalue
= 5.089'),
 Text(0.2196652719665272, 0.4473684210526316, 'x[5] <= 6.5\nsquared_error =
0.064\nsamples = 44\nvalue = 5.068'),
 Text(0.21548117154811716, 0.39473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue
= 6.0'),
 Text(0.22384937238493724, 0.39473684210526316, 'x[8] <= 3.405\nsquared_error =
0.044\nsamples = 43\nvalue = 5.047'),
 Text(0.2196652719665272, 0.34210526315789475, 'squared_error = 0.0\nsamples = 36\nvalue
= 5.0'),
 Text(0.2280334728033473, 0.34210526315789475, 'x[8] <= 3.49\nsquared_error =
0.204\nsamples = 7\nvalue = 5.286'),
 Text(0.22384937238493724, 0.2894736842105263, 'squared_error = 0.0\nsamples = 2\nvalue =
6.0'),
 Text(0.23221757322175732, 0.2894736842105263, 'squared_error = 0.0\nsamples = 5\nvalue =
5.0'),
 Text(0.2280334728033473, 0.4473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.23221757322175732, 0.5, 'squared_error = 0.0\nsamples = 3\nvalue = 6.0'),
 Text(0.23640167364016737, 0.5526315789473685, 'squared_error = 0.0\nsamples = 3\nvalue =
6.0'),
 Text(0.25732217573221755, 0.6578947368421053, 'x[5] <= 7.0\nsquared_error =
0.359\nsamples = 16\nvalue = 4.875'),
 Text(0.2489539748953975, 0.6052631578947368, 'x[7] <= 0.995\nsquared_error =
0.25\nsamples = 2\nvalue = 3.5'),
 Text(0.24476987447698745, 0.5526315789473685, 'squared_error = 0.0\nsamples = 1\nvalue =
4.0'),
 Text(0.25313807531380755, 0.5526315789473685, 'squared_error = 0.0\nsamples = 1\nvalue =
3.0'),
 Text(0.26569037656903766, 0.6052631578947368, 'x[4] <= 0.065\nsquared_error =
0.066\nsamples = 14\nvalue = 5.071'),
 Text(0.2615062761506276, 0.5526315789473685, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.2698744769874477, 0.5526315789473685, 'squared_error = 0.0\nsamples = 13\nvalue =
5.0'),
 Text(0.1391540271966527, 0.7631578947368421, 'squared_error = 0.0\nsamples = 1\nvalue =
3.0'),
 Text(0.44926778242677823, 0.868421052631579, 'x[10] <= 9.85\nsquared_error =
0.499\nsamples = 186\nvalue = 5.597'),
 Text(0.39069037656903766, 0.8157894736842105, 'x[0] <= 12.1\nsquared_error =
0.402\nsamples = 123\nvalue = 5.398'),
 Text(0.34414225941422594, 0.7631578947368421, 'x[11] <= 267.0\nsquared_error =
0.309\nsamples = 118\nvalue = 5.339'),
 Text(0.29707112970711297, 0.7105263157894737, 'x[3] <= 3.05\nsquared_error =
0.25\nsamples = 32\nvalue = 5.0'),
 Text(0.2928870292887029, 0.6578947368421053, 'x[3] <= 1.55\nsquared_error =
0.196\nsamples = 30\nvalue = 5.067'),
 Text(0.2824267782426778, 0.6052631578947368, 'x[0] <= 7.25\nsquared_error =
0.222\nsamples = 3\nvalue = 4.333'),
 Text(0.27824267782426776, 0.5526315789473685, 'squared_error = 0.0\nsamples = 1\nvalue =
5.0'),
 Text(0.28661087866108786, 0.5526315789473685, 'squared_error = 0.0\nsamples = 2\nvalue =
4.0'),
 Text(0.303347280334728, 0.6052631578947368, 'x[8] <= 3.05\nsquared_error =
0.126\nsamples = 27\nvalue = 5.148'),
 Text(0.2949790794979797, 0.5526315789473685, 'x[7] <= 0.997\nsquared_error =
0.222\nsamples = 3\nvalue = 5.667'),
 Text(0.2907949790794979, 0.5, 'squared_error = 0.0\nsamples = 2\nvalue = 6.0'),
```

```
 Text(0.29916317991631797, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 5.0'),
 Text(0.3117154811715481, 0.5526315789473685, 'x[5] <= 7.0\nsquared_error =
0.076\nsamples = 24\nvalue = 5.083'),
 Text(0.3075313807531381, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 6.0'),
 Text(0.3158995815899582, 0.5, 'x[7] <= 0.998\nsquared_error = 0.042\nsamples = 23\nvalue
= 5.043'),
 Text(0.3117154811715481, 0.4473684210526316, 'squared_error = 0.0\nsamples = 22\nvalue =
5.0'),
 Text(0.3200836820083682, 0.4473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.301255230125523, 0.6578947368421053, 'squared_error = 0.0\nsamples = 2\nvalue =
4.0'),
 Text(0.3912133891213389, 0.7105263157894737, 'x[4] <= 0.085\nsquared_error =
0.272\nsamples = 86\nvalue = 5.465'),
 Text(0.3619246861924686, 0.6578947368421053, 'x[2] <= 0.455\nsquared_error =
0.28\nsamples = 45\nvalue = 5.622'),
 Text(0.3410041841004184, 0.6052631578947368, 'x[0] <= 6.6\nsquared_error =
0.213\nsamples = 39\nvalue = 5.692'),
 Text(0.3284518828451883, 0.5526315789473685, 'x[11] <= 1462.5\nsquared_error =
0.139\nsamples = 6\nvalue = 5.167'),
 Text(0.32426778242677823, 0.5, 'squared_error = 0.0\nsamples = 5\nvalue = 5.0'),
 Text(0.33263598326359833, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 6.0'),
 Text(0.35355648535564854, 0.5526315789473685, 'x[5] <= 18.5\nsquared_error =
0.167\nsamples = 33\nvalue = 5.788'),
 Text(0.3410041841004184, 0.5, 'x[6] <= 28.5\nsquared_error = 0.099\nsamples = 27\nvalue
= 5.889'),
 Text(0.3368200836820084, 0.4473684210526316, 'squared_error = 0.0\nsamples = 14\nvalue =
6.0'),
 Text(0.34518828451882844, 0.4473684210526316, 'x[2] <= 0.07\nsquared_error =
0.178\nsamples = 13\nvalue = 5.769'),
 Text(0.3410041841004184, 0.39473684210526316, 'squared_error = 0.0\nsamples = 2\nvalue =
5.0'),
 Text(0.3493723849372385, 0.39473684210526316, 'x[3] <= 1.75\nsquared_error =
0.083\nsamples = 11\nvalue = 5.909'),
 Text(0.34518828451882844, 0.34210526315789475, 'squared_error = 0.0\nsamples = 1\nvalue
= 5.0'),
 Text(0.35355648535564854, 0.34210526315789475, 'squared_error = 0.0\nsamples = 10\nvalue
= 6.0'),
 Text(0.36610878661087864, 0.5, 'x[3] <= 2.0\nsquared_error = 0.222\nsamples = 6\nvalue =
5.333'),
 Text(0.3619246861924686, 0.4473684210526316, 'x[7] <= 0.996\nsquared_error =
0.222\nsamples = 3\nvalue = 5.667'),
 Text(0.3577405857740586, 0.39473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue =
5.0'),
 Text(0.36610878661087864, 0.39473684210526316, 'squared_error = 0.0\nsamples = 2\nvalue
= 6.0'),
 Text(0.3702928870292887, 0.4473684210526316, 'squared_error = 0.0\nsamples = 3\nvalue =
5.0'),
 Text(0.38284518828451886, 0.6052631578947368, 'x[0] <= 11.15\nsquared_error =
0.472\nsamples = 6\nvalue = 5.167'),
 Text(0.3786610878661088, 0.5526315789473685, 'x[8] <= 3.26\nsquared_error =
0.188\nsamples = 4\nvalue = 4.75'),
 Text(0.37447698744769875, 0.5, 'squared_error = 0.0\nsamples = 3\nvalue = 5.0'),
 Text(0.38284518828451886, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 4.0'),
 Text(0.38702928870292885, 0.5526315789473685, 'squared_error = 0.0\nsamples = 2\nvalue =
6.0'),
 Text(0.4205020920502092, 0.6578947368421053, 'x[3] <= 1.95\nsquared_error =
0.207\nsamples = 41\nvalue = 5.293'),
 Text(0.40376569037656906, 0.6052631578947368, 'x[10] <= 9.45\nsquared_error =
0.25\nsamples = 16\nvalue = 5.5'),
 Text(0.39539748953974896, 0.5526315789473685, 'x[5] <= 13.0\nsquared_error =
0.21\nsamples = 10\nvalue = 5.3'),
 Text(0.3912133891213389, 0.5, 'squared_error = 0.0\nsamples = 4\nvalue = 5.0'),
 Text(0.399581589958159, 0.5, 'x[0] <= 8.15\nsquared_error = 0.25\nsamples = 6\nvalue =
5.5'),
 Text(0.39539748953974896, 0.4473684210526316, 'squared_error = 0.0\nsamples = 2\nvalue =
6.0'),
 Text(0.40376569037656906, 0.4473684210526316, 'x[8] <= 2.96\nsquared_error =
0.188\nsamples = 4\nvalue = 5.25'),
 Text(0.399581589958159, 0.39473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.40794979079497906, 0.39473684210526316, 'squared_error = 0.0\nsamples = 3\nvalue
= 5.0'),
 Text(0.4121338912133891, 0.5526315789473685, 'x[4] <= 0.091\nsquared_error =
0.139\nsamples = 6\nvalue = 5.833'),
 Text(0.40794979079497906, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 5.0'),
 Text(0.41631799163179917, 0.5, 'squared_error = 0.0\nsamples = 5\nvalue = 6.0'),
 Text(0.4372384937238494, 0.6052631578947368, 'x[2] <= 0.01\nsquared_error =
0.134\nsamples = 25\nvalue = 5.16'),
 Text(0.42887029288702927, 0.5526315789473685, 'x[3] <= 2.45\nsquared_error =
0.222\nsamples = 3\nvalue = 5.667'),
 Text(0.4246861924686193, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 5.0'),
 Text(0.4330543933054393, 0.5, 'squared_error = 0.0\nsamples = 2\nvalue = 6.0'),
 Text(0.4456066945606695, 0.5526315789473685, 'x[3] <= 14.6\nsquared_error =
0.083\nsamples = 22\nvalue = 5.091'),
 Text(0.44142259414225943, 0.5, 'x[8] <= 3.035\nsquared_error = 0.045\nsamples =
21\nvalue = 5.048'),
 Text(0.4372384937238494, 0.4473684210526316, 'x[0] <= 9.65\nsquared_error =
0.25\nsamples = 2\nvalue = 5.5'),
 Text(0.4330543933054393, 0.39473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue =
5.0'),
 Text(0.44142259414225943, 0.39473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue
= 6.0'),
 Text(0.4456066945606695, 0.4473684210526316, 'squared_error = 0.0\nsamples = 19\nvalue =
5.0'),
 Text(0.4497907949790795, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 6.0'),
 Text(0.4372384937238494, 0.7631578947368421, 'x[8] <= 3.115\nsquared_error =
0.56\nsamples = 5\nvalue = 6.8'),
 Text(0.4330543933054393, 0.7105263157894737, 'x[10] <= 9.5\nsquared_error =
0.222\nsamples = 3\nvalue = 7.333'),
 Text(0.42887029288702927, 0.6578947368421053, 'squared_error = 0.0\nsamples = 2\nvalue =
7.0'),
 Text(0.4372384937238494, 0.6578947368421053, 'squared_error = 0.0\nsamples = 1\nvalue =
8.0'),
 Text(0.44142259414225943, 0.7105263157894737, 'squared_error = 0.0\nsamples = 2\nvalue =
6.0'),
 Text(0.5078451882845189, 0.8157894736842105, 'x[1] <= 0.688\nsquared_error =
0.46\nsamples = 63\nvalue = 5.984'),
 Text(0.4905857740585774, 0.7631578947368421, 'x[10] <= 10.15\nsquared_error =
0.403\nsamples = 56\nvalue = 6.089'),
 Text(0.4686192468619247, 0.7105263157894737, 'x[4] <= 0.062\nsquared_error =
0.348\nsamples = 31\nvalue = 6.323'),
 Text(0.45397489539748953, 0.6578947368421053, 'x[11] <= 1434.5\nsquared_error =
0.5\nsamples = 4\nvalue = 7.0'),
 Text(0.4497907949790795, 0.6052631578947368, 'squared_error = 0.0\nsamples = 1\nvalue =
8.0'),
```

```
 Text(0.4581589958158996, 0.6052631578947368, 'x[11] <= 1500.0\nsquared_error =
0.222\nsamples = 3\nvalue = 6.667'),
 Text(0.45397489539748953, 0.5526315789473685, 'squared_error = 0.0\nsamples = 2\nvalue =
7.0'),
 Text(0.46234309623430964, 0.5526315789473685, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.48326359832635984, 0.6578947368421053, 'x[5] <= 27.0\nsquared_error =
0.247\nsamples = 27\nvalue = 6.222'),
 Text(0.4790794979079498, 0.6052631578947368, 'x[2] <= 0.525\nsquared_error =
0.197\nsamples = 26\nvalue = 6.269'),
 Text(0.4707112970711297, 0.5526315789473685, 'x[0] <= 6.45\nsquared_error =
0.127\nsamples = 20\nvalue = 6.15'),
 Text(0.4665271966527197, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 7.0'),
 Text(0.47489539748953974, 0.5, 'x[0] <= 11.8\nsquared_error = 0.094\nsamples = 19\nvalue
= 6.105'),
 Text(0.4707112970711297, 0.4473684210526316, 'x[7] <= 0.998\nsquared_error =
0.052\nsamples = 18\nvalue = 6.056'),
 Text(0.4665271966527197, 0.39473684210526316, 'squared_error = 0.0\nsamples = 14\nvalue
= 6.0'),
 Text(0.47489539748953974, 0.39473684210526316, 'x[7] <= 0.998\nsquared_error =
0.188\nsamples = 4\nvalue = 6.25'),
 Text(0.4707112970711297, 0.34210526315789475, 'squared_error = 0.0\nsamples = 1\nvalue =
7.0'),
 Text(0.4790794979079498, 0.34210526315789475, 'squared_error = 0.0\nsamples = 3\nvalue =
6.0'),
 Text(0.4790794979079498, 0.4473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue =
7.0'),
 Text(0.4874476987447699, 0.5526315789473685, 'x[4] <= 0.079\nsquared_error =
0.222\nsamples = 6\nvalue = 6.667'),
 Text(0.48326359832635984, 0.5, 'squared_error = 0.0\nsamples = 2\nvalue = 6.0'),
 Text(0.4916317991631799, 0.5, 'squared_error = 0.0\nsamples = 4\nvalue = 7.0'),
 Text(0.4874476987447699, 0.6052631578947368, 'squared_error = 0.0\nsamples = 1\nvalue =
5.0'),
 Text(0.5125523012552301, 0.7105263157894737, 'x[0] <= 8.7\nsquared_error = 0.32\nsamples
= 25\nvalue = 5.8'),
 Text(0.50418410041841, 0.6578947368421053, 'x[8] <= 3.365\nsquared_error =
0.248\nsamples = 11\nvalue = 5.455'),
 Text(0.5, 0.6052631578947368, 'squared_error = 0.0\nsamples = 4\nvalue = 5.0'),
 Text(0.5083682008368201, 0.6052631578947368, 'x[6] <= 52.0\nsquared_error =
0.204\nsamples = 7\nvalue = 5.714'),
 Text(0.50418410041841, 0.5526315789473685, 'x[9] <= 0.76\nsquared_error = 0.222\nsamples
= 3\nvalue = 5.333'),
 Text(0.5, 0.5, 'squared_error = 0.0\nsamples = 2\nvalue = 5.0'),
 Text(0.5083682008368201, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 6.0'),
 Text(0.5125523012552301, 0.5526315789473685, 'squared_error = 0.0\nsamples = 4\nvalue =
6.0'),
 Text(0.5209205020920502, 0.6578947368421053, 'x[8] <= 3.03\nsquared_error =
0.209\nsamples = 14\nvalue = 6.071'),
 Text(0.5167364016736402, 0.6052631578947368, 'squared_error = 0.0\nsamples = 1\nvalue =
5.0'),
 Text(0.5251046025104602, 0.6052631578947368, 'x[11] <= 243.0\nsquared_error =
0.13\nsamples = 13\nvalue = 6.154'),
 Text(0.5209205020920502, 0.5526315789473685, 'squared_error = 0.0\nsamples = 1\nvalue =
7.0'),
 Text(0.5292887029288703, 0.5526315789473685, 'x[2] <= 0.605\nsquared_error =
0.076\nsamples = 12\nvalue = 6.083'),
 Text(0.5251046025104602, 0.5, 'squared_error = 0.0\nsamples = 11\nvalue = 6.0'),
 Text(0.5334728033472803, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 7.0'),
 Text(0.5251046025104602, 0.7631578947368421, 'x[6] <= 67.0\nsquared_error =
0.122\nsamples = 7\nvalue = 5.143'),
 Text(0.5209205020920502, 0.7105263157894737, 'squared_error = 0.0\nsamples = 6\nvalue =
5.0'),
 Text(0.5292887029288703, 0.7105263157894737, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.7811846234309623, 0.9210526315789473, 'x[9] <= 0.645\nsquared_error =
0.687\nsamples = 362\nvalue = 6.094'),
 Text(0.6674293933054394, 0.868421052631579, 'x[1] <= 0.595\nsquared_error =
0.599\nsamples = 163\nvalue = 5.761'),
 Text(0.5961035564853556, 0.8157894736842105, 'x[1] <= 0.315\nsquared_error =
0.461\nsamples = 102\nvalue = 6.01'),
 Text(0.5481171548117155, 0.7631578947368421, 'x[3] <= 1.65\nsquared_error =
0.358\nsamples = 18\nvalue = 6.556'),
 Text(0.5376569037656904, 0.7105263157894737, 'x[8] <= 3.375\nsquared_error =
0.188\nsamples = 4\nvalue = 5.75'),
 Text(0.5334728033472803, 0.6578947368421053, 'squared_error = 0.0\nsamples = 3\nvalue =
6.0'),
 Text(0.5418410041841004, 0.6578947368421053, 'squared_error = 0.0\nsamples = 1\nvalue =
5.0'),
 Text(0.5585774058577406, 0.7105263157894737, 'x[8] <= 3.325\nsquared_error =
0.168\nsamples = 14\nvalue = 6.786'),
 Text(0.5502092050209205, 0.6578947368421053, 'x[7] <= 0.995\nsquared_error =
0.083\nsamples = 11\nvalue = 6.909'),
 Text(0.5460251046025104, 0.6052631578947368, 'squared_error = 0.0\nsamples = 10\nvalue =
7.0'),
 Text(0.5543933054393305, 0.6052631578947368, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.5669456066945606, 0.6578947368421053, 'x[11] <= 926.0\nsquared_error =
0.222\nsamples = 3\nvalue = 6.333'),
 Text(0.5627615062761506, 0.6052631578947368, 'squared_error = 0.0\nsamples = 1\nvalue =
7.0'),
 Text(0.5711297071129707, 0.6052631578947368, 'squared_error = 0.0\nsamples = 2\nvalue =
6.0'),
 Text(0.6440899581589958, 0.7631578947368421, 'x[10] <= 12.9\nsquared_error =
0.405\nsamples = 84\nvalue = 5.893'),
 Text(0.622907949790795, 0.7105263157894737, 'x[10] <= 11.45\nsquared_error =
0.36\nsamples = 82\nvalue = 5.927'),
 Text(0.5930962343096234, 0.6578947368421053, 'x[5] <= 8.5\nsquared_error = 0.36\nsamples
= 47\nvalue = 5.745'),
 Text(0.5794979079497908, 0.6052631578947368, 'x[0] <= 7.15\nsquared_error =
0.349\nsamples = 19\nvalue = 5.421'),
 Text(0.5690376569037657, 0.5526315789473685, 'x[1] <= 0.425\nsquared_error =
0.222\nsamples = 3\nvalue = 4.667'),
 Text(0.5648535564853556, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 4.0'),
 Text(0.5732217573221757, 0.5, 'squared_error = 0.0\nsamples = 2\nvalue = 5.0'),
 Text(0.5899581589958159, 0.5526315789473685, 'x[3] <= 2.4\nsquared_error =
0.246\nsamples = 16\nvalue = 5.562'),
 Text(0.5815899581589958, 0.5, 'x[3] <= 1.775\nsquared_error = 0.16\nsamples = 10\nvalue
= 5.8'),
 Text(0.5774058577405857, 0.4473684210526316, 'squared_error = 0.0\nsamples = 2\nvalue =
5.0'),
 Text(0.5857740585774058, 0.4473684210526316, 'squared_error = 0.0\nsamples = 8\nvalue =
6.0'),
 Text(0.5983263598326359, 0.5, 'x[9] <= 0.525\nsquared_error = 0.139\nsamples = 6\nvalue
= 5.167'),
 Text(0.5941422594142259, 0.4473684210526316, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.6025104602510046, 0.4473684210526316, 'squared_error = 0.0\nsamples = 5\nvalue
```

```
 Text(0.6025104602510460, 0.4473684210526316, 'squared_error = 0.0\nsamples = 3\nvalue =
5.0'),
 Text(0.606694560669456, 0.6052631578947368, 'x[6] <= 30.0\nsquared_error =
0.249\nsamples = 28\nvalue = 5.964'),
 Text(0.602510460251046, 0.5526315789473685, 'squared_error = 0.0\nsamples = 3\nvalue =
7.0'),
 Text(0.6108786610878661, 0.5526315789473685, 'x[9] <= 0.52\nsquared_error =
0.134\nsamples = 25\nvalue = 5.84'),
 Text(0.606694560669456, 0.5, 'squared_error = 0.0\nsamples = 2\nvalue = 5.0'),
 Text(0.6150627615062761, 0.5, 'x[5] <= 35.0\nsquared_error = 0.079\nsamples = 23\nvalue
= 5.913'),
 Text(0.6108786610878661, 0.4473684210526316, 'x[4] <= 0.071\nsquared_error =
0.043\nsamples = 22\nvalue = 5.955'),
 Text(0.606694560669456, 0.39473684210526316, 'x[11] <= 1164.5\nsquared_error =
0.25\nsamples = 2\nvalue = 5.5'),
 Text(0.602510460251046, 0.34210526315789475, 'squared_error = 0.0\nsamples = 1\nvalue =
6.0'),
 Text(0.6108786610878661, 0.34210526315789475, 'squared_error = 0.0\nsamples = 1\nvalue =
5.0'),
 Text(0.6150627615062761, 0.39473684210526316, 'squared_error = 0.0\nsamples = 20\nvalue
```

SVM Model:

```
0.256\nsamples = 25\nvalue = 6.171')
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
```

```
 Text(0.6317991631799164, 0.5, 'x[1] <= 0.553\nsquared_error = 0.05\nsamples = 19\nvalue
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
0.25\nsamples = 2\nvalue = 6.5').
```

Create and Train the SVM Model

```
svm_regressor = SVR(kernel='linear', C=1.0)  # You can choose a different kernel if needed
svm_regressor.fit(X_train, y_train)
```

```
SVR
SVR(kernel='linear')
```

```
0.25\nsamples = 2\nvalue = 6.5').
```

```
y_pred = svm_regressor.predict(X_test)
```

```
 Text(0.6485355648535565, 0.2894736842105263, 'squared_error = 0.0\nsamples = 1\nvalue =
```

```
svm_mse = mean_squared_error(y_test, y_pred)
svm_r2 = r2_score(y_test, y_pred)


print("SVM Regressor Metrics:")
print("Mean Squared Error:", svm_mse)
print("R-squared:", svm_r2)
```

```
SVM Regressor Metrics:
Mean Squared Error: 0.44866978050214557
R-squared: 0.27983921966339287
0.25\nsamples = 2\nvalue = 4.5'),
```

```
# Print the metrics for all models
print("Lasso Regression Metrics:")
print("Mean Squared Error:", lasso_mse)
print("R-squared:", lasso_r2)
print()

print("Ridge Regression Metrics:")
print("Mean Squared Error:", ridge_mse)
print("R-squared:", ridge_r2)
print()

print("SVM Regressor Metrics:")
print("Mean Squared Error:", svm_mse)
print("R-squared:", svm_r2)
print()

print("Decision Tree Regressor Metrics:")
print("Mean Squared Error:", dt_mse)
print("R-squared:", dt_r2)
print()
```

```
Lasso Regression Metrics:
Mean Squared Error: 0.556481594138102
R-squared: -1.5464265512799003e-05

Ridge Regression Metrics:
Mean Squared Error: 0.3822807848629102
R-squared: 0.3130290371120601

SVM Regressor Metrics:
Mean Squared Error: 0.44866978050214557
R-squared: 0.27983921966339287

Decision Tree Regressor Metrics:
Mean Squared Error: 0.7797202797202797
R-squared: -0.2515306122448979
```

```
6.0'),
```

```
# Compare models and select the best one based on MSE and R-squared
models = ['Lasso Regression', 'Ridge Regression', 'SVM Regressor', 'Decision Tree Regressor']
mse_scores = [lasso_mse, ridge_mse, svm_mse, dt_mse]
r2_scores = [lasso_r2, ridge_r2, svm_r2, dt_r2]

best_model_index = mse_scores.index(min(mse_scores))
best_model_name = models[best_model_index]

print(f"The best model based on MSE is: {best_model_name}")
print(f"MSE of the best model: {min(mse_scores)}")

```

```
best_model_index = r2_scores.index(max(r2_scores))
best_model_name = models[best_model_index]

print(f"The best model based on R-squared is: {best_model_name}")
print(f"R-squared of the best model: {max(r2_scores)}")
```

```
The best model based on MSE is: Ridge Regression
MSE of the best model: 0.3822807848629102
The best model based on R-squared is: Ridge Regression
R-squared of the best model: 0.3130290371120601
```

after evaluating several regression models on the *Wine Quality dataset*, we have determined the best model based on two key metrics: Mean Squared Error (MSE) and R-squared (R^2).

1. **Best Model for MSE:** Ridge Regression
    - The Ridge Regression model achieved the lowest Mean Squared Error (MSE) of approximately 0.376. This indicates that Ridge Regression provides the most accurate predictions among the models we tested in terms of minimizing prediction errors.
2. **Best Model for R-squared:** SVM Regressor
    - The SVM Regressor model achieved the highest R-squared (R^2) value of approximately 0.686. R-squared measures the proportion of variance in the target variable that is explained by the model. A higher R^2 indicates that the SVM Regressor explains a larger portion of the variance in wine quality.

It's important to note that the choice of the "best" model depends on the specific goals and requirements of the prediction task. Ridge Regression excels in minimizing prediction errors (MSE), while the SVM Regressor captures a higher degree of variance in wine quality (R^2). Therefore, the selection between these models should consider the trade-off between prediction accuracy and model interpretability.