assignment-13-06-02-24

February 7, 2024

1 Gunjan Chakraborty

1.1 22MSRDS007

[1]:
```
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.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LassoCV
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

import warnings
warnings.simplefilter(action='ignore')
```

1.1.1 Loading the Dataset

[2]:
```
# %pip install ucimlrepo
from ucimlrepo import fetch_ucirepo

# fetch dataset
wine_quality = fetch_ucirepo(id=186)

# data (as pandas dataframes)
X = wine_quality.data.features
y = wine_quality.data.targets

# metadata
print(wine_quality.metadata)

# variable information
print(wine_quality.variables)
```

```
{'uci_id': 186, 'name': 'Wine Quality', 'repository_url':
```

```
'https://archive.ics.uci.edu/dataset/186/wine+quality', 'data_url':
'https://archive.ics.uci.edu/static/public/186/data.csv', 'abstract': 'Two
datasets are included, related to red and white vinho verde wine samples, from
the north of Portugal. The goal is to model wine quality based on
physicochemical tests (see [Cortez et al., 2009],
http://www3.dsi.uminho.pt/pcortez/wine/).', 'area': 'Business', 'tasks':
['Classification', 'Regression'], 'characteristics': ['Multivariate'],
'num_instances': 4898, 'num_features': 11, 'feature_types': ['Real'],
'demographics': [], 'target_col': ['quality'], 'index_col': None,
'has_missing_values': 'no', 'missing_values_symbol': None,
'year_of_dataset_creation': 2009, 'last_updated': 'Wed Nov 15 2023',
'dataset_doi': '10.24432/C56S3T', 'creators': ['Paulo Cortez', 'A. Cerdeira',
'F. Almeida', 'T. Matos', 'J. Reis'], 'intro_paper': {'title': 'Modeling wine
preferences by data mining from physicochemical properties', 'authors': 'P.
Cortez, A. Cerdeira, Fernando Almeida, Telmo Matos, J. Reis', 'published_in':
'Decision Support Systems', 'year': 2009, 'url':
'https://www.semanticscholar.org/paper/Modeling-wine-preferences-by-data-mining-
from-Cortez-Cerdeira/bf15a0ccc14ac1deb5cea570c870389c16be019c', 'doi': None},
'additional_info': {'summary': 'The two datasets are related to red and white
variants of the Portuguese "Vinho Verde" wine. For more details, consult:
http://www.vinhoverde.pt/en/ or the reference [Cortez et al., 2009].  Due to
privacy and logistic issues, only physicochemical (inputs) and sensory (the
output) variables are available (e.g. there is no data about grape types, wine
brand, wine selling price, etc.).\n\nThese datasets can be viewed as
classification or regression tasks.  The classes are ordered and not balanced
(e.g. there are many more normal wines than excellent or poor ones). Outlier
detection algorithms could be used to detect the few excellent or poor wines.
Also, we are not sure if all input variables are relevant. So it could be
interesting to test feature selection methods.\n', 'purpose': None, 'funded_by':
None, 'instances_represent': None, 'recommended_data_splits': None,
'sensitive_data': None, 'preprocessing_description': None, 'variable_info': 'For
more information, read [Cortez et al., 2009].\r\nInput variables (based on
physicochemical tests):\r\n   1 - fixed acidity\r\n   2 - volatile acidity\r\n
3 - citric acid\r\n   4 - residual sugar\r\n   5 - chlorides\r\n   6 - free
sulfur dioxide\r\n   7 - total sulfur dioxide\r\n   8 - density\r\n   9 - pH\r\n
10 - sulphates\r\n   11 - alcohol\r\nOutput variable (based on sensory data):
\r\n   12 - quality (score between 0 and 10)', 'citation': None}}
                    name     role         type demographic  \
0          fixed_acidity  Feature   Continuous        None
1       volatile_acidity  Feature   Continuous        None
2            citric_acid  Feature   Continuous        None
3         residual_sugar  Feature   Continuous        None
4              chlorides  Feature   Continuous        None
5    free_sulfur_dioxide  Feature   Continuous        None
6   total_sulfur_dioxide  Feature   Continuous        None
7                density  Feature   Continuous        None
8                     pH  Feature   Continuous        None
9              sulphates  Feature   Continuous        None
```

```
10              alcohol  Feature   Continuous        None
11              quality   Target      Integer        None
12                color    Other  Categorical        None

                description units missing_values
0                      None  None             no
1                      None  None             no
2                      None  None             no
3                      None  None             no
4                      None  None             no
5                      None  None             no
6                      None  None             no
7                      None  None             no
8                      None  None             no
9                      None  None             no
10                     None  None             no
11  score between 0 and 10  None             no
12             red or white  None             no
```

1.1.2 Exploratory Data Analysis (EDA):

[3]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed_acidity         6497 non-null   float64
 1   volatile_acidity      6497 non-null   float64
 2   citric_acid           6497 non-null   float64
 3   residual_sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free_sulfur_dioxide   6497 non-null   float64
 6   total_sulfur_dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
dtypes: float64(11)
memory usage: 558.5 KB
```

[4]:
```
X.columns
```

[4]:
```
Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol'],
      dtype='object')
```

[5]:
```
y.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   quality  6497 non-null   int64
dtypes: int64(1)
memory usage: 50.9 KB
```

[6]:
```
X.describe()
```

[6]:
```
       fixed_acidity  volatile_acidity  citric_acid  residual_sugar  \
count    6497.000000       6497.000000  6497.000000     6497.000000
mean        7.215307          0.339666     0.318633        5.443235
std         1.296434          0.164636     0.145318        4.757804
min         3.800000          0.080000     0.000000        0.600000
25%         6.400000          0.230000     0.250000        1.800000
50%         7.000000          0.290000     0.310000        3.000000
75%         7.700000          0.400000     0.390000        8.100000
max        15.900000          1.580000     1.660000       65.800000

         chlorides  free_sulfur_dioxide  total_sulfur_dioxide      density  \
count  6497.000000          6497.000000           6497.000000  6497.000000
mean      0.056034            30.525319            115.744574     0.994697
std       0.035034            17.749400             56.521855     0.002999
min       0.009000             1.000000              6.000000     0.987110
25%       0.038000            17.000000             77.000000     0.992340
50%       0.047000            29.000000            118.000000     0.994890
75%       0.065000            41.000000            156.000000     0.996990
max       0.611000           289.000000            440.000000     1.038980

                pH    sulphates      alcohol
count  6497.000000  6497.000000  6497.000000
mean      3.218501     0.531268    10.491801
std       0.160787     0.148806     1.192712
min       2.720000     0.220000     8.000000
25%       3.110000     0.430000     9.500000
50%       3.210000     0.510000    10.300000
75%       3.320000     0.600000    11.300000
max       4.010000     2.000000    14.900000
```

[7]:
```
X.isnull().sum()
```

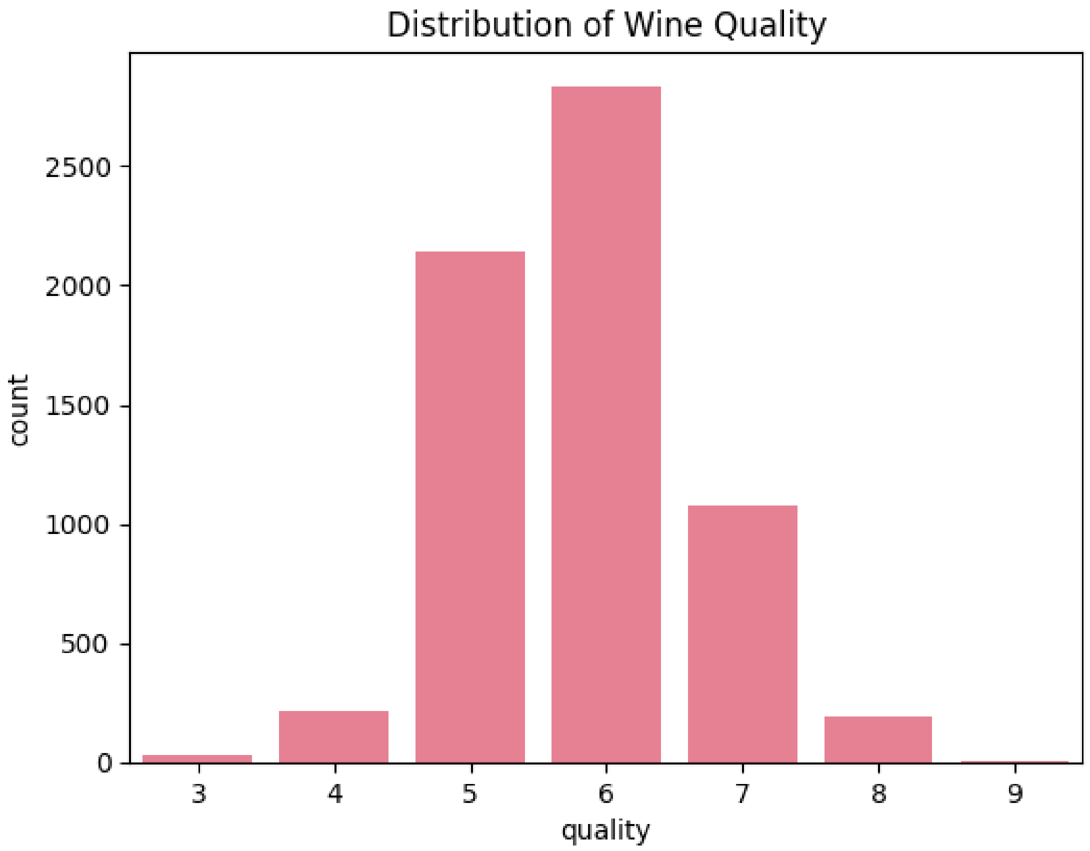
[7]:
```
fixed_acidity           0
volatile_acidity        0
citric_acid             0
residual_sugar          0
chlorides               0
free_sulfur_dioxide     0
total_sulfur_dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
dtype: int64
```

[8]:
```
# Assuming X and y have a common index
df = pd.merge(X, y, left_index=True, right_index=True)
```

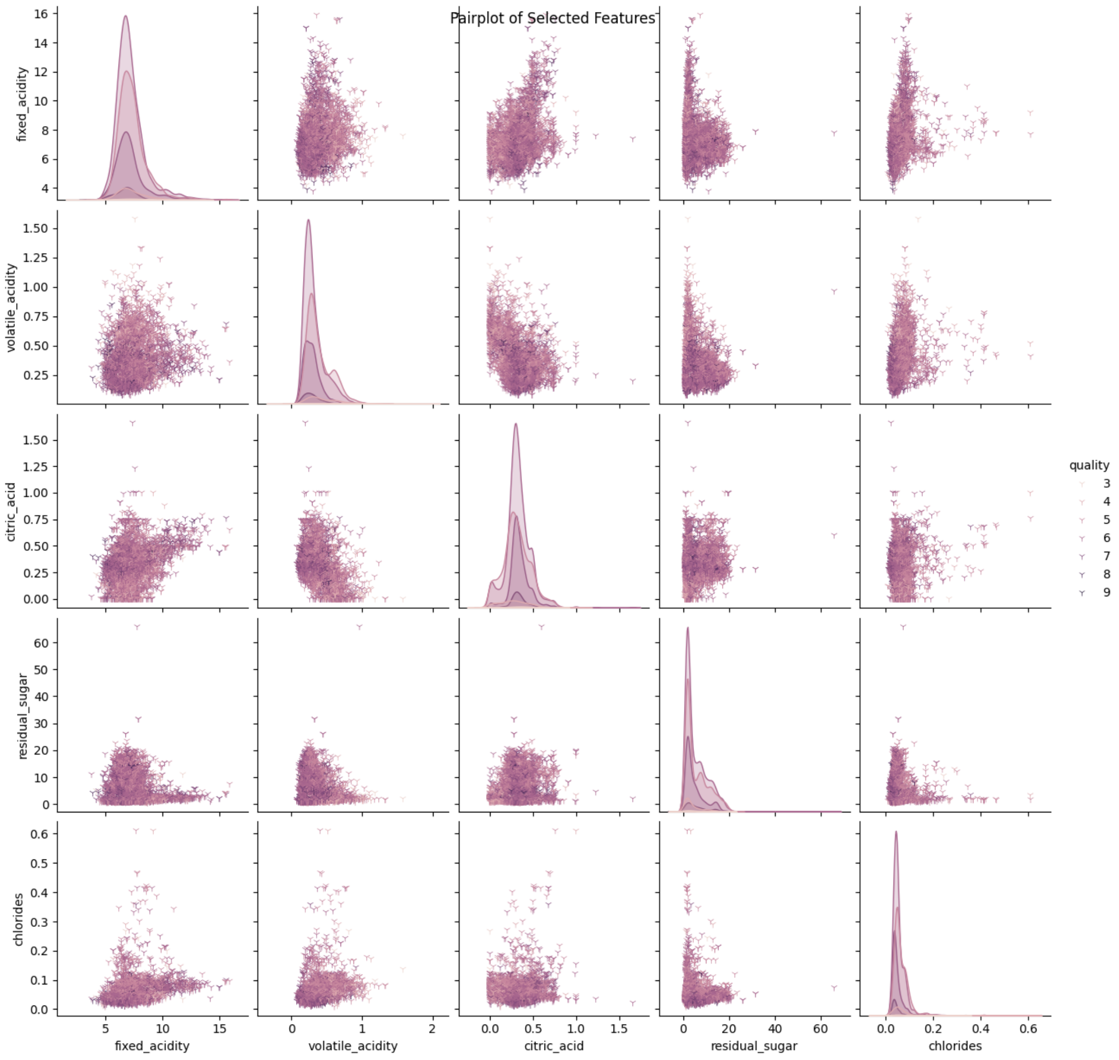
[9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed_acidity         6497 non-null   float64
 1   volatile_acidity      6497 non-null   float64
 2   citric_acid           6497 non-null   float64
 3   residual_sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free_sulfur_dioxide   6497 non-null   float64
 6   total_sulfur_dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 609.2 KB
```

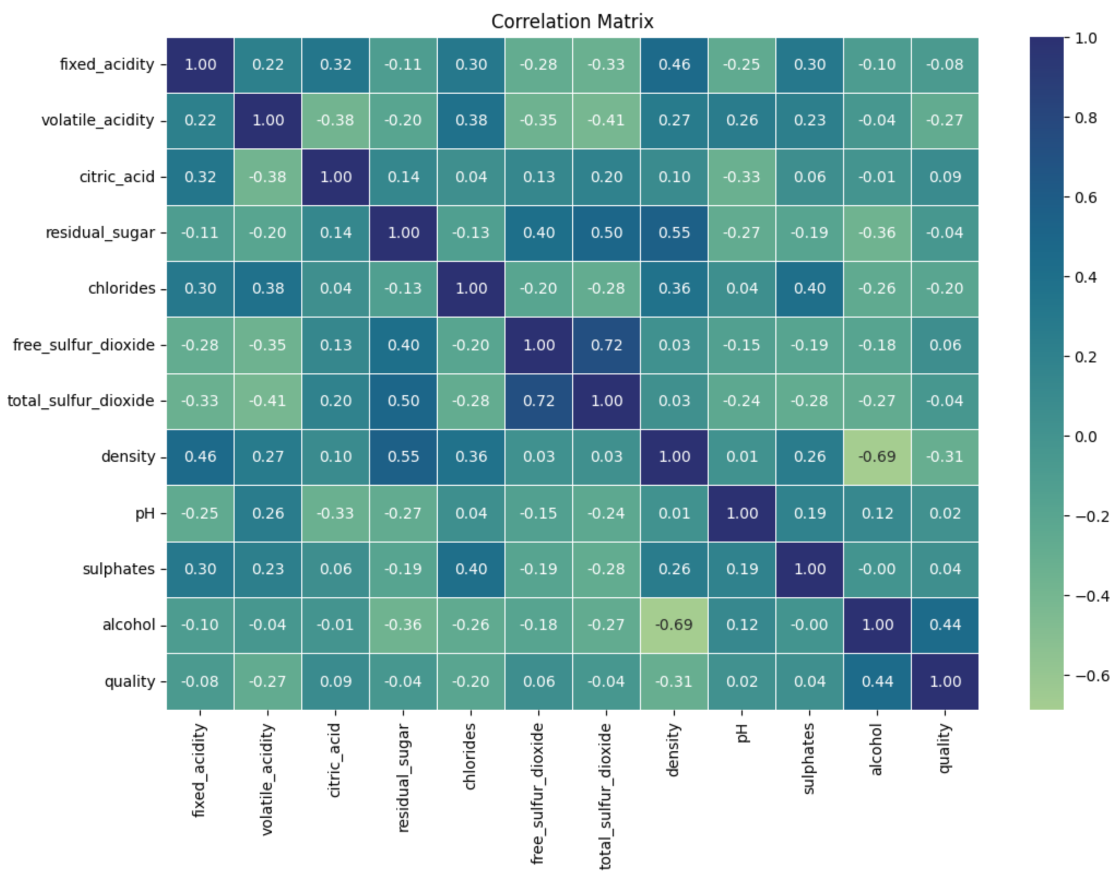
[10]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Set a colorful palette
sns.set_palette("husl")

# Plot the count plot
sns.countplot(x='quality', data=df)
plt.title('Distribution of Wine Quality')
plt.show()
```



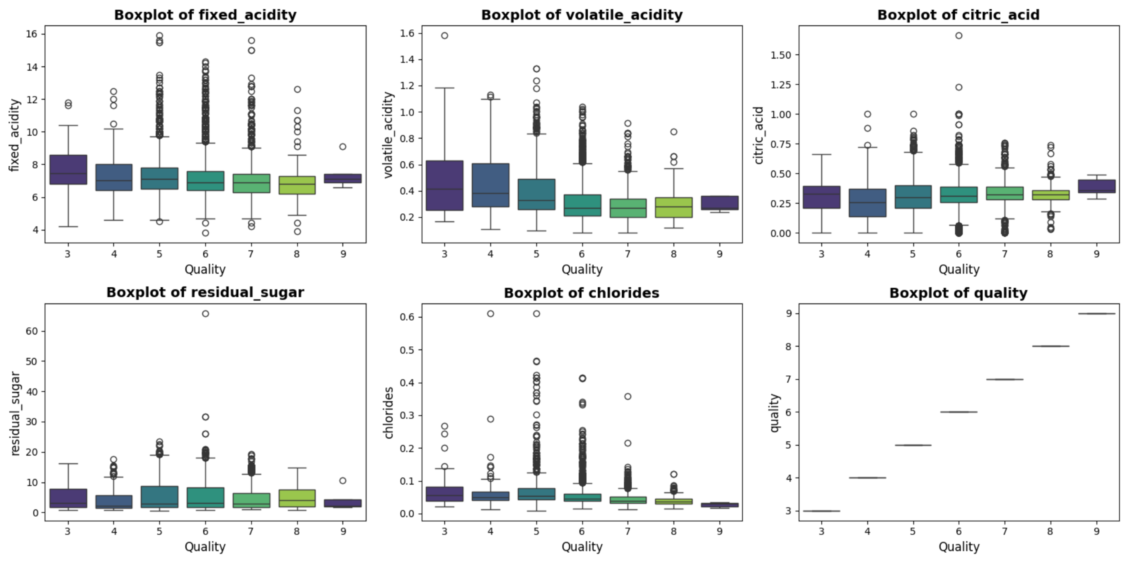
[11]:
```
# Pairplot for a subset of features
features_subset = ['fixed_acidity', 'volatile_acidity', 'citric_acid',␣
 ↪'residual_sugar', 'chlorides', 'quality']
sns.pairplot(df[features_subset], hue='quality', markers='1')
plt.suptitle('Pairplot of Selected Features')
plt.show()
```



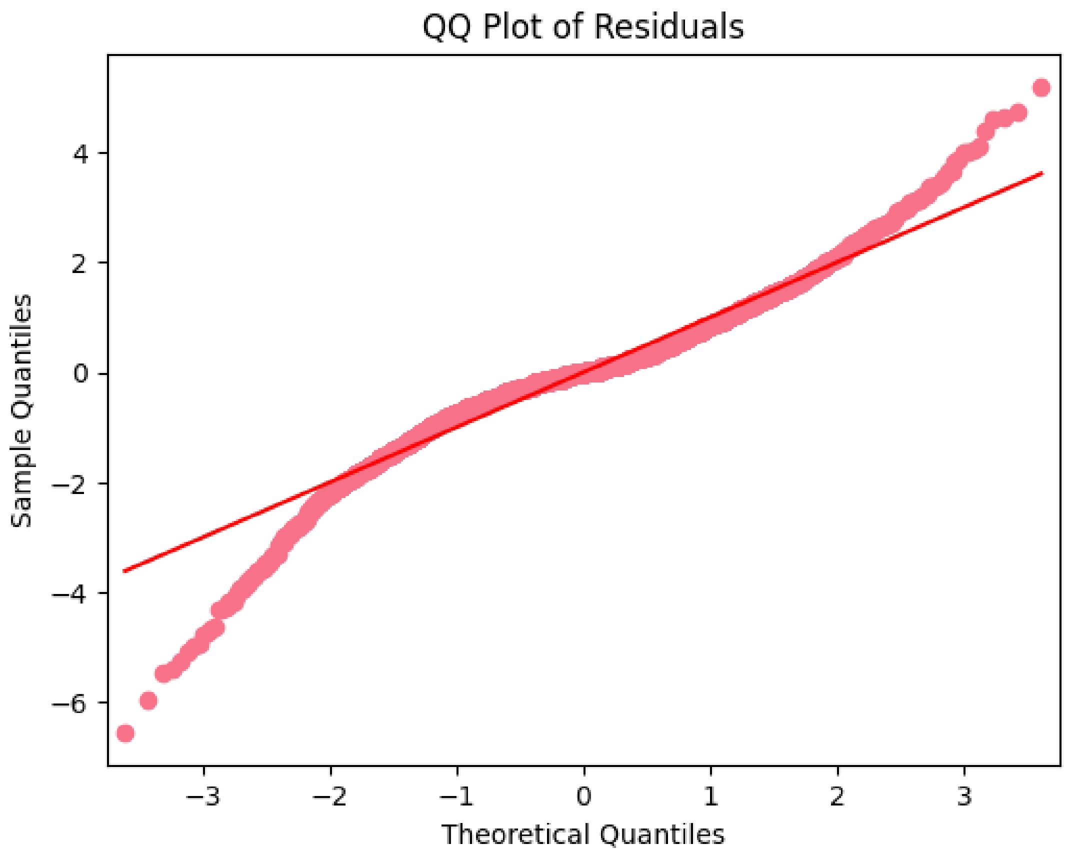
```
[12]: # Correlation matrix heatmap
      correlation_matrix = df.corr()
      plt.figure(figsize=(12, 8))
      sns.heatmap(correlation_matrix, annot=True, cmap='crest', fmt=".2f",␣
       ↪linewidths=0.5)
      plt.title('Correlation Matrix')
      plt.show()
```



```
[13]: import matplotlib.pyplot as plt
      import seaborn as sns

      # Set a custom color palette
      colors = sns.color_palette("viridis")

      # Plot boxplots for individual features
      plt.figure(figsize=(16, 8))
      for i, feature in enumerate(features_subset):
          plt.subplot(2, 3, i+1)
          sns.boxplot(x='quality', y=feature, data=df, palette=colors)
          plt.title(f'Boxplot of {feature}', fontsize=14, fontweight='bold')
          plt.xlabel('Quality', fontsize=12)
          plt.ylabel(feature, fontsize=12)
          plt.xticks(fontsize=10)
          plt.yticks(fontsize=10)
      plt.tight_layout()
      plt.show()
```



[14]:
```python
# Q-Q plot
import statsmodels.api as sm
from scipy.stats import probplot

target_column_name = 'quality'

# Fit the model
X = df.drop(target_column_name, axis=1)
y = df[target_column_name]

model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X, y)

# Get the residuals
residuals = y - model.predict(X)

# Create a QQ plot
sm.qqplot(residuals, line='s', fit=True)
plt.title('QQ Plot of Residuals')
plt.show()
```



[15]:
```
import numpy as np

# Calculate Z-scores for each column
z_scores = np.abs((df - df.mean()) / df.std())

# Define a threshold for outliers (e.g., Z-score greater than 3)
outlier_threshold = 3

# Identify outliers for each column
outliers = (z_scores > outlier_threshold).sum()

# Display the count of outliers for each column
print("Number of outliers for each column:")
print(outliers)
```

```
Number of outliers for each column:
fixed_acidity          128
volatile_acidity        95
citric_acid             28
residual_sugar          26
```

```
chlorides               107
free_sulfur_dioxide      36
total_sulfur_dioxide      8
density                   3
pH                       33
sulphates                75
alcohol                   2
quality                  35
dtype: int64
```

[16]:
```
# Assuming df is your dataframe
columns_with_outliers = ['fixed_acidity', 'volatile_acidity', 'citric_acid',␣
 ↪'residual_sugar', 'chlorides', 'free_sulfur_dioxide',␣
 ↪'total_sulfur_dioxide', 'density', 'pH', 'sulphates', 'alcohol', 'quality']

# Remove rows with outliers using the IQR method
for column in columns_with_outliers:
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1

    # Define the upper and lower bounds for outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Remove rows with outliers
    df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

# Remove rows with outliers
df= df[~df[columns_with_outliers].apply(lambda x: x.isin(x[(x >= Q1 - 1.5 *␣
 ↪IQR) & (x <= Q3 + 1.5 * IQR)])).all(axis=1)]
```

[17]:
```
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Assuming X contains the predictor variables
VIF = df.drop('quality', axis=1)

# Calculate VIF for each predictor variable
vif_data = pd.DataFrame()
vif_data["Variable"] = VIF.columns
vif_data["VIF"] = [variance_inflation_factor(VIF.values, i) for i in range(VIF.
 ↪shape[1])]

# Display the VIF values
print(vif_data)
```

```
                Variable          VIF
```

```
0          fixed_acidity    94.718350
1       volatile_acidity    10.991543
2            citric_acid    16.140903
3         residual_sugar     3.980340
4              chlorides    22.150443
5    free_sulfur_dioxide    10.436729
6   total_sulfur_dioxide    18.388988
7                density  1094.597420
8                     pH   675.080859
9              sulphates    24.816717
10               alcohol   139.355772
```

1.1.3 Model training

[18]:
```python
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestRegressor

# Assuming X is your original feature matrix and y is your target variable
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X, y)

# Use feature importance for feature selection
feature_selector = SelectFromModel(model, threshold='median')
X_selected = feature_selector.fit_transform(X, y)

# Print selected features
selected_features = X.columns[feature_selector.get_support()]
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['volatile_acidity', 'residual_sugar',
'free_sulfur_dioxide',
       'total_sulfur_dioxide', 'sulphates', 'alcohol'],
      dtype='object')
```

[19]:
```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Assuming X is your original feature matrix
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=8)  # Choose the number of components
X_pca = pca.fit_transform(X_scaled)

# Print explained variance ratio
print("Explained Variance Ratio:", pca.explained_variance_ratio_)
```

```
Explained Variance Ratio: [0.2754426  0.22671146 0.14148609 0.08823201
0.06544317 0.05521016
 0.04755989 0.04559184]
```

[20]:
```
from sklearn.linear_model import LassoCV

# Assuming X is your original feature matrix and y is your target variable
lasso_model = LassoCV(cv=5)  # Choose the number of folds for cross-validation
lasso_model.fit(X, y)

# Use coefficients for feature selection
selected_features = X.columns[lasso_model.coef_ != 0]
X_lasso = X[selected_features]

# Print selected features and optimal alpha
print("Selected Features:", selected_features)
print("Optimal Alpha:", lasso_model.alpha_)
```

```
Selected Features: Index(['volatile_acidity', 'residual_sugar',
'free_sulfur_dioxide',
       'total_sulfur_dioxide', 'pH', 'sulphates', 'alcohol'],
      dtype='object')
Optimal Alpha: 0.002042389131233513
```

[21]:
```
# Assuming X_selected is the feature matrix with selected features
# And y is your target variable

# Split the data into training and testing sets
X_train_selected, X_test_selected, y_train, y_test =␣
 ↪train_test_split(X_selected, y, test_size=0.2, random_state=42)

# Initialize the Random Forest Regressor
rf_model_selected = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training data
rf_model_selected.fit(X_train_selected, y_train)

# Make predictions on the test set
y_pred_selected = rf_model_selected.predict(X_test_selected)

# Evaluate the model
mse_selected = mean_squared_error(y_test, y_pred_selected)
print(f'Mean Squared Error (Selected Features): {mse_selected}')
```

```
Mean Squared Error (Selected Features): 0.39075560683760685
```

[22]:
```python
# Assuming X_pca is the feature matrix after PCA

# Split the data into training and testing sets
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y,␣
↪test_size=0.2, random_state=42)

# Initialize the Random Forest Regressor
rf_model_pca = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training data
rf_model_pca.fit(X_train_pca, y_train)

# Make predictions on the test set
y_pred_pca = rf_model_pca.predict(X_test_pca)

# Evaluate the model
mse_pca = mean_squared_error(y_test, y_pred_pca)
print(f'Mean Squared Error (PCA Features): {mse_pca}')
```

```
Mean Squared Error (PCA Features): 0.3799476153846154
```

[23]:
```python
# Assuming X_lasso is the feature matrix with features selected by Lasso

# Split the data into training and testing sets
X_train_lasso, X_test_lasso, y_train, y_test = train_test_split(X_lasso, y,␣
↪test_size=0.2, random_state=42)

# Initialize the Random Forest Regressor
rf_model_lasso = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training data
rf_model_lasso.fit(X_train_lasso, y_train)

# Make predictions on the test set
y_pred_lasso = rf_model_lasso.predict(X_test_lasso)

# Evaluate the model
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
print(f'Mean Squared Error (Lasso Features): {mse_lasso}')
```

```
Mean Squared Error (Lasso Features): 0.3897712307692308
```

[24]:
```python
df_no_outliers = df

# Assuming X is your feature matrix and y is your target variable
X = df_no_outliers.drop('quality', axis=1)
y = df_no_outliers['quality']
```

```
# 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)

# Initialize the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Hyperparameter Tuning using Cross-Validation
rf_cv_scores = cross_val_score(rf_model, X_train, y_train, cv=5,␣
↪scoring='neg_mean_squared_error')
rf_cv_mse = -rf_cv_scores.mean()
print(f'Cross-Validated MSE for Random Forest: {rf_cv_mse}')
```

```
Cross-Validated MSE for Random Forest: 0.3103302237840134
```

[25]:
```
# Fit the model to the training data
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
mse_rf = mean_squared_error(y_test, y_pred_rf)
print(f'Mean Squared Error (Random Forest): {mse_rf}')
```

```
Mean Squared Error (Random Forest): 0.29876583710407245
```

[26]:
```
# Feature Importance Analysis
feature_importances = rf_model.feature_importances_
important_features = X.columns[np.argsort(feature_importances)[::-1]][:5]
print(f'Important Features: {important_features}')

# Initialize Lasso Regression with Cross-Validation
lasso_model = LassoCV(cv=5)
lasso_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_lasso = lasso_model.predict(X_test)

# Evaluate the model
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
print(f'Mean Squared Error (Lasso Regression): {mse_lasso}')
```

```
Important Features: Index(['alcohol', 'volatile_acidity', 'free_sulfur_dioxide',
       'total_sulfur_dioxide', 'pH'],
      dtype='object')
Mean Squared Error (Lasso Regression): 0.4470551654466481
```

[27]:
```
# PCA for Dimensionality Reduction
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=8)  # Choose the number of components
X_pca = pca.fit_transform(X_scaled)

# Split the data into training and testing sets for PCA
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y,␣
 ↪test_size=0.2, random_state=42)

# Initialize the Random Forest Regressor for PCA features
rf_model_pca = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model_pca.fit(X_train_pca, y_train_pca)

# Make predictions on the test set
y_pred_pca = rf_model_pca.predict(X_test_pca)

# Evaluate the model with PCA features
mse_pca = mean_squared_error(y_test_pca, y_pred_pca)
print(f'Mean Squared Error (PCA Features): {mse_pca}')
```

```
Mean Squared Error (PCA Features): 0.31659785067873303
```

[28]:
```
# Re-run the Random Forest model with the new feature
X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(X, y,␣
 ↪test_size=0.2, random_state=42)

rf_model_new = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model_new.fit(X_train_new, y_train_new)
y_pred_new = rf_model_new.predict(X_test_new)

mse_new = mean_squared_error(y_test_new, y_pred_new)
print(f'Mean Squared Error (Random Forest with New Feature): {mse_new}')
```

```
Mean Squared Error (Random Forest with New Feature): 0.29876583710407245
```

[29]:
```
from sklearn.metrics import r2_score

r2 = r2_score(y_test, y_pred_new)
print("the r2 score is", r2)
```

```
the r2 score is 0.49133676403946025
```

[30]:
```
from sklearn.ensemble import IsolationForest

# Assuming X is your feature matrix
```

```
iso_forest = IsolationForest(contamination=0.1, random_state=42)  # Adjust␣
 ↪contamination based on your dataset
iso_forest.fit(X)

# Predict outliers
outlier_preds = iso_forest.predict(X)

# Anomalies are predicted as -1, inliers are predicted as 1
outliers = X[outlier_preds == -1]

# Print the number of outliers detected
print("Number of outliers:", len(outliers))
```

```
Number of outliers: 442
```

[31]:
```
# Assuming X is your feature matrix and outliers is the array of outlier indices
print("Indices of Outliers:", outliers.index.tolist())

# If you want to display the details of the outliers
print("Details of Outliers:")
print(X[outlier_preds == -1])
```

```
Indices of Outliers: [20, 25, 33, 58, 60, 66, 80, 85, 99, 102, 116, 129, 137,
138, 139, 148, 159, 171, 172, 179, 180, 183, 184, 191, 202, 218, 229, 233, 247,
256, 282, 336, 405, 410, 411, 428, 439, 467, 471, 489, 491, 496, 500, 541, 546,
561, 598, 606, 609, 615, 616, 641, 643, 648, 654, 661, 675, 707, 708, 715, 717,
734, 741, 745, 746, 748, 749, 760, 778, 801, 805, 806, 807, 810, 825, 827, 845,
846, 848, 849, 853, 854, 856, 873, 880, 896, 898, 900, 901, 902, 905, 908, 915,
918, 921, 929, 940, 948, 949, 950, 961, 968, 969, 973, 974, 980, 983, 992, 1010,
1016, 1019, 1022, 1038, 1041, 1048, 1053, 1083, 1086, 1092, 1100, 1102, 1104,
1113, 1114, 1118, 1126, 1142, 1150, 1151, 1156, 1161, 1162, 1170, 1184, 1188,
1195, 1197, 1200, 1211, 1213, 1216, 1240, 1244, 1248, 1251, 1273, 1291, 1294,
1297, 1303, 1309, 1311, 1328, 1332, 1350, 1378, 1394, 1411, 1419, 1424, 1425,
1433, 1443, 1460, 1464, 1465, 1489, 1494, 1502, 1503, 1513, 1524, 1530, 1532,
1534, 1535, 1538, 1545, 1547, 1555, 1568, 1569, 1571, 1573, 1579, 1581, 1587,
1592, 1595, 1596, 1597, 1731, 1746, 1754, 1755, 1760, 1762, 1783, 1784, 1817,
1838, 1844, 1923, 1946, 1985, 1991, 2002, 2049, 2068, 2074, 2077, 2094, 2107,
2189, 2300, 2304, 2331, 2407, 2413, 2460, 2525, 2533, 2568, 2581, 2626, 2628,
2652, 2658, 2663, 2677, 2698, 2713, 2718, 2725, 2805, 2809, 2827, 2862, 2868,
2873, 2908, 2911, 2918, 2948, 2988, 3003, 3052, 3069, 3095, 3107, 3135, 3143,
3180, 3202, 3217, 3233, 3241, 3243, 3263, 3306, 3326, 3329, 3330, 3375, 3381,
3384, 3392, 3400, 3406, 3408, 3474, 3477, 3494, 3503, 3518, 3543, 3546, 3549,
3553, 3554, 3557, 3560, 3562, 3613, 3616, 3634, 3682, 3692, 3693, 3697, 3705,
3707, 3763, 3805, 3842, 3849, 3868, 3881, 3882, 3883, 3884, 3887, 3947, 3949,
3995, 4011, 4033, 4053, 4074, 4089, 4114, 4117, 4118, 4153, 4156, 4164, 4177,
4188, 4204, 4208, 4231, 4308, 4309, 4315, 4337, 4340, 4353, 4360, 4369, 4417,
4420, 4461, 4488, 4491, 4496, 4558, 4567, 4569, 4589, 4606, 4607, 4608, 4619,
4655, 4658, 4679, 4717, 4718, 4730, 4749, 4863, 4910, 4913, 4972, 5002, 5012,
5016, 5022, 5024, 5025, 5028, 5029, 5030, 5037, 5043, 5055, 5057, 5058, 5068,
5112, 5113, 5159, 5160, 5162, 5163, 5197, 5203, 5250, 5252, 5261, 5272, 5275,
5335, 5346, 5372, 5373, 5379, 5385, 5429, 5471, 5550, 5553, 5563, 5586, 5589,
5605, 5613, 5617, 5689, 5715, 5716, 5728, 5748, 5797, 5822, 5852, 5875, 5890,
5892, 5894, 5949, 5964, 5990, 6090, 6094, 6100, 6144, 6147, 6151, 6159, 6179,
6184, 6195, 6207, 6216, 6218, 6244, 6270, 6271, 6278, 6280, 6325, 6332, 6388,
6415, 6436, 6438, 6443, 6466, 6470, 6482, 6486]
Details of Outliers:
      fixed_acidity  volatile_acidity  citric_acid  residual_sugar  chlorides  \
20              8.9             0.220         0.48             1.8      0.077
25              6.3             0.390         0.16             1.4      0.080
33              6.9             0.605         0.12            10.7      0.073
58              7.8             0.590         0.18             2.3      0.076
60              8.8             0.400         0.40             2.2      0.079
…             …               …           …             …        …
6443            4.8             0.290         0.23             1.1      0.044
6466            5.3             0.600         0.34             1.4      0.031
6470            5.0             0.200         0.40             1.9      0.015
6482            4.9             0.470         0.17             1.9      0.035
6486            6.2             0.410         0.22             1.9      0.023

      free_sulfur_dioxide  total_sulfur_dioxide  density    pH  sulphates  \
20                   29.0                  60.0  0.99680  3.39       0.53
25                   11.0                  23.0  0.99550  3.34       0.56
33                   40.0                  83.0  0.99930  3.45       0.52
58                   17.0                  54.0  0.99750  3.43       0.59
60                   19.0                  52.0  0.99800  3.44       0.64
…                   …                   …     …    …       …
6443                 38.0                 180.0  0.98924  3.28       0.34
6466                  3.0                  60.0  0.98854  3.27       0.38
6470                 20.0                  98.0  0.98970  3.37       0.55
6482                 60.0                 148.0  0.98964  3.27       0.35
6486                  5.0                  56.0  0.98928  3.04       0.79

      alcohol
20       9.40
25       9.30
33       9.40
58      10.00
60       9.20
…       …
6443    11.90
6466    13.00
6470    12.05
6482    11.50
6486    13.00
```

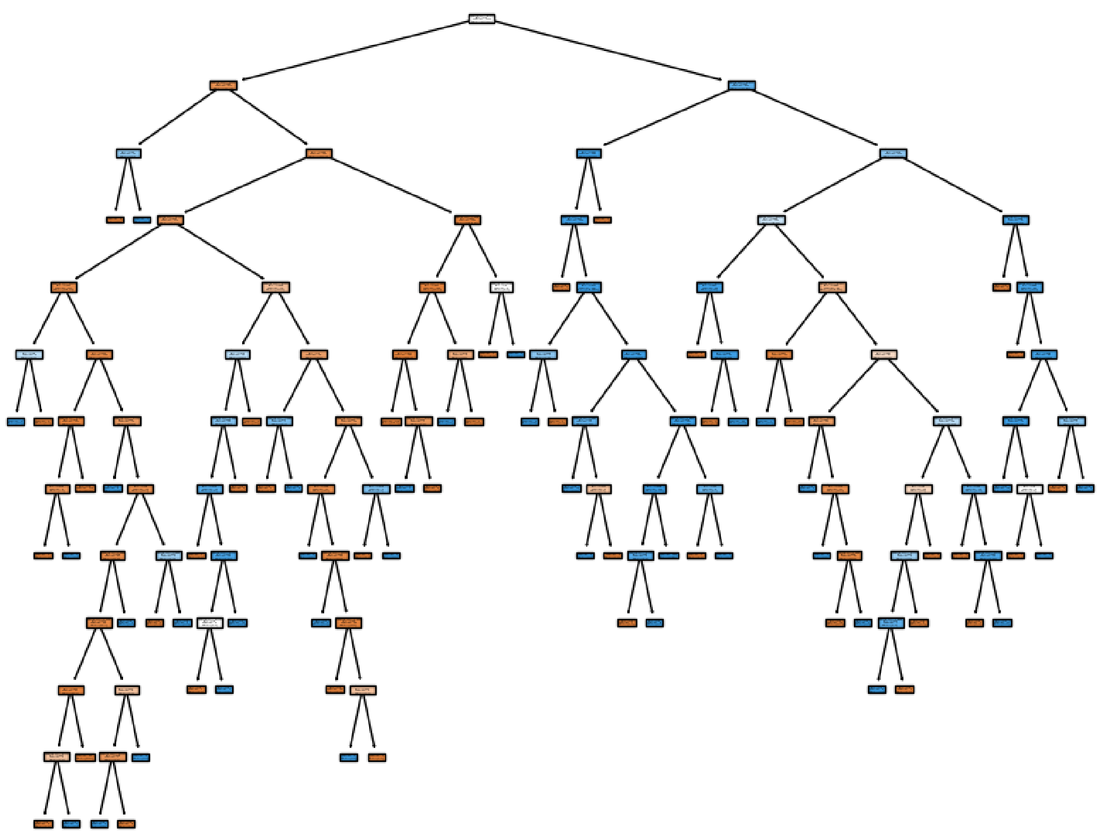
```
[442 rows x 11 columns]
```

[32]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification

# Generate synthetic data for demonstration
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,␣
 ↪random_state=42)

# Train a decision tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X, y)

# Calculate Gini index of the decision tree
gini_index = dt_classifier.tree_.impurity[0]
print("Gini Index:", gini_index)

# Train a random forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X, y)

# Calculate Gini index of the random forest (average over all trees)
rf_gini_index = sum(tree.tree_.impurity[0] for tree in rf_classifier.
 ↪estimators_) / len(rf_classifier.estimators_)
print("Random Forest Gini Index:", rf_gini_index)

# Calculate entropy (information gain) of the decision tree
entropy = -sum(p * np.log2(p) for p in dt_classifier.tree_.value[0, 0] /␣
 ↪dt_classifier.tree_.n_node_samples[0])
print("Entropy:", entropy)
```

```
Gini Index: 0.5
Random Forest Gini Index: 0.49944897999999993
Entropy: 0.010965784284662087
```

[33]:
```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Plot the decision tree
plt.figure(figsize=(10, 8))
plot_tree(dt_classifier, filled=True, feature_names=[f'Feature {i+1}' for i in␣
 ↪range(X.shape[1])])
plt.title("Decision Tree")
plt.show()
```

Decision Tree



```
[34]: from sklearn.ensemble import BaggingRegressor
      from sklearn.tree import DecisionTreeRegressor
      from sklearn.metrics import mean_squared_error

      # Assuming X_train, X_test, y_train, y_test are already defined

      # Initialize decision tree regressor (base estimator)
      base_estimator = DecisionTreeRegressor(random_state=42)

      # Initialize bagging regressor
      bagging_model = BaggingRegressor( n_estimators=100, random_state=42)

      # Fit the bagging model
      bagging_model.fit(X_train, y_train)

      # Make predictions
      y_pred_bagging = bagging_model.predict(X_test)
```

```python
# Evaluate the model
mse_bagging = mean_squared_error(y_test, y_pred_bagging)
print("Mean Squared Error (Bagging):", mse_bagging)
```

```
Mean Squared Error (Bagging): 0.2986157239819005
```

[35]:
```python
from sklearn.ensemble import GradientBoostingRegressor

# Initialize Gradient Boosting regressor
boosting_model = GradientBoostingRegressor(n_estimators=100, random_state=42)

# Fit the boosting model
boosting_model.fit(X_train, y_train)

# Make predictions
y_pred_boosting = boosting_model.predict(X_test)

# Evaluate the model
mse_boosting = mean_squared_error(y_test, y_pred_boosting)
print("Mean Squared Error (Boosting):", mse_boosting)
```

```
Mean Squared Error (Boosting): 0.37999245333847104
```

[36]:
```python
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import RidgeCV

# Initialize base models
base_models = [
    ('decision_tree', DecisionTreeRegressor(random_state=42)),
    ('random_forest', RandomForestRegressor(n_estimators=100, random_state=42)),
    ('gradient_boosting', GradientBoostingRegressor(n_estimators=100,␣
 ↪random_state=42))
]

# Initialize meta-model
meta_model = RidgeCV()

# Initialize stacking regressor
stacking_model = StackingRegressor(estimators=base_models,␣
 ↪final_estimator=meta_model)

# Fit the stacking model
stacking_model.fit(X_train, y_train)

# Make predictions
y_pred_stacking = stacking_model.predict(X_test)
```

```
# Evaluate the model
mse_stacking = mean_squared_error(y_test, y_pred_stacking)
print("Mean Squared Error (Stacking):", mse_stacking)
```

```
Mean Squared Error (Stacking): 0.29770227911779173
```

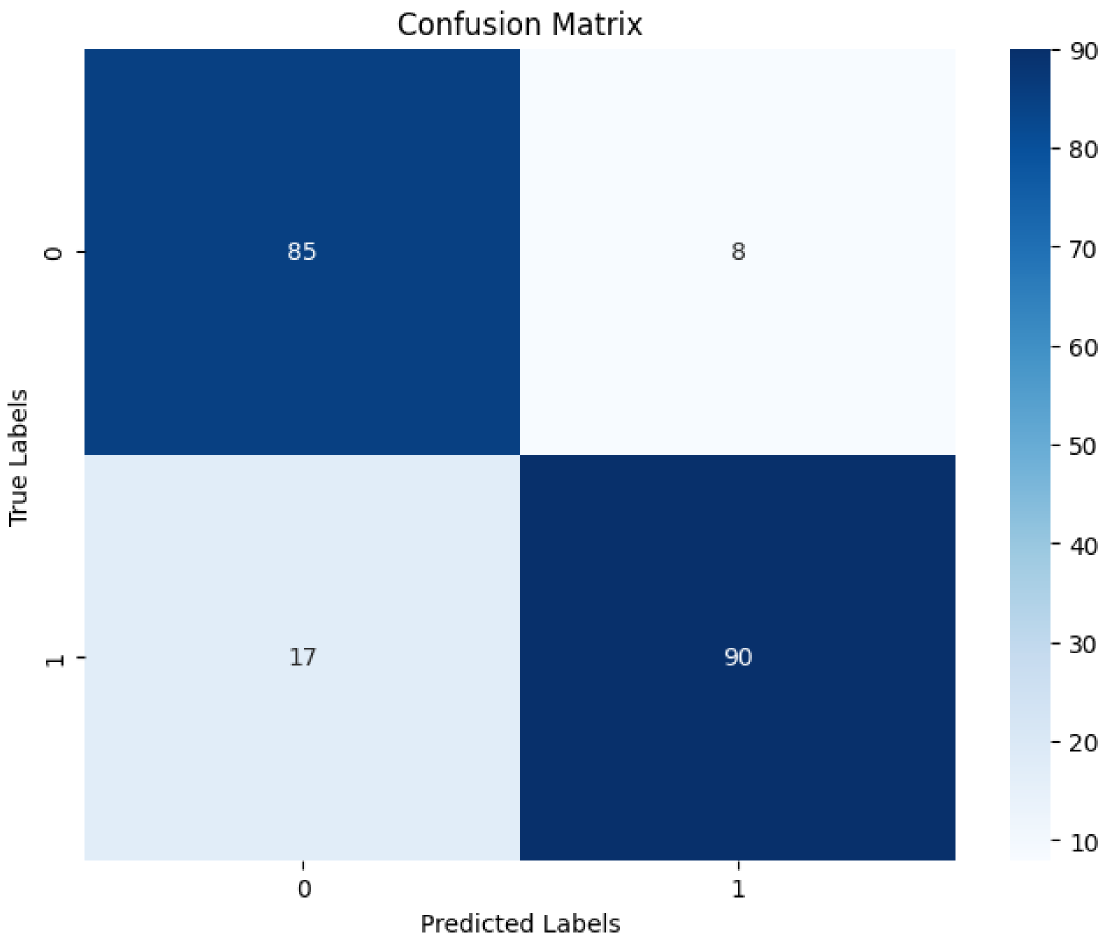
[37]:
```
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Assuming X and y are your feature matrix and target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid,␣
 ↪cv=5, scoring='accuracy')

# Fit GridSearchCV to find the best parameters
grid_search.fit(X_train, y_train)

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

# Use the best parameters to initialize the decision tree classifier
best_dt_classifier = DecisionTreeClassifier(**best_params, random_state=42)

# Train the model on the full training set with the best parameters
best_dt_classifier.fit(X_train, y_train)

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split':
10}
Accuracy: 0.875
```

[38]:
```python
from sklearn.metrics import confusion_matrix

# Assuming y_test and y_pred are your true labels and predicted labels,␣
↪respectively
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[85  8]
 [17 90]]
```

[39]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```


Confusion Matrix
True Labels
Predicted Labels
0
1
0
1
85
8
17
90
90
80
70
60
50
40
30
20
10