assignment-12-05-02-24

February 5, 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 1. 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()
```

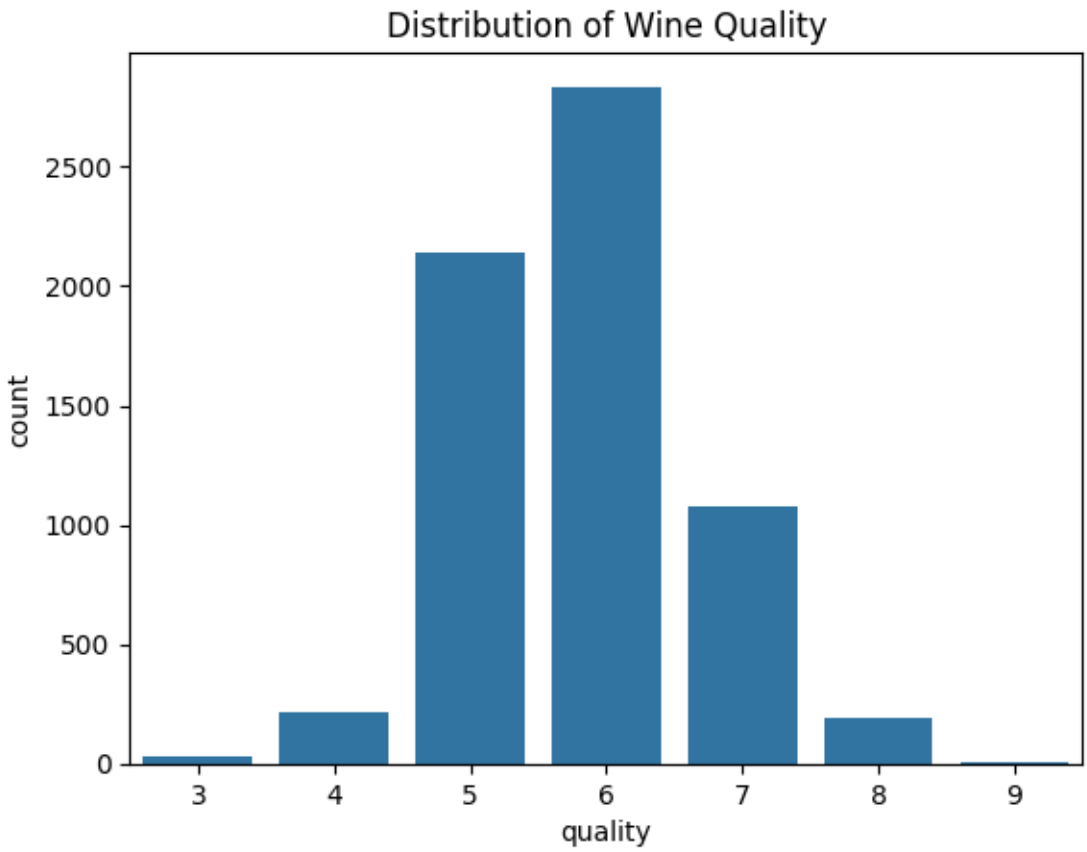
[7]:
```
fixed_acidity           0
volatile_acidity        0
citric_acid             0
residual_sugar          0
chlorides               0
free_sulfur_dioxide     0
total_sulfur_dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
dtype: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

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

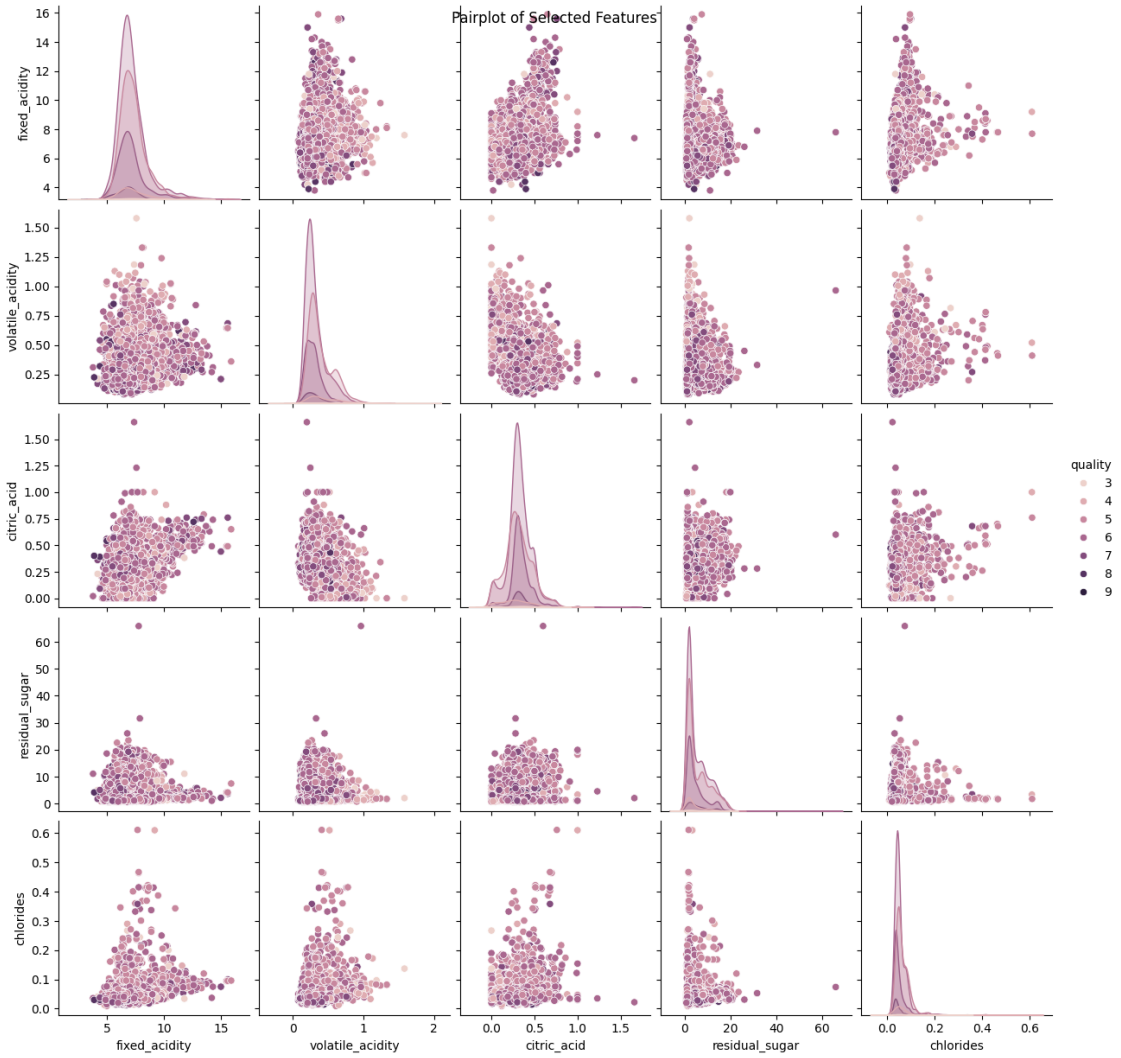
[9]:
```
df.info()
```

```
<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]:
```
# Distribution of the target variable 'quality'
sns.countplot(x='quality', data=df)
plt.title('Distribution of Wine Quality')
plt.show()
```

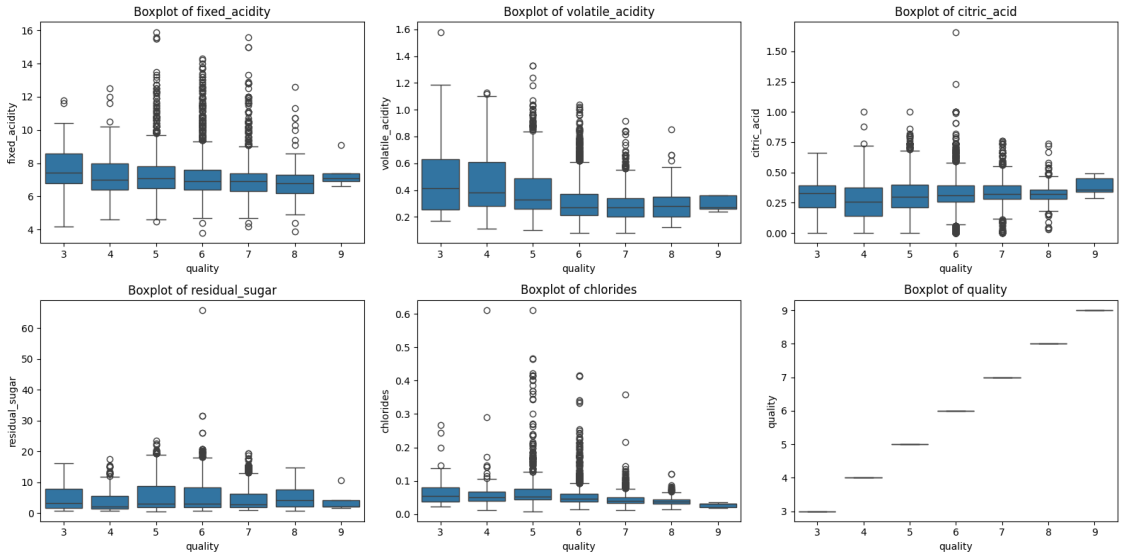


[11]:
```
# Pairplot for a subset of features
features_subset = ['fixed_acidity', 'volatile_acidity', 'citric_acid',␣
 ↪'residual_sugar', 'chlorides', 'quality']
sns.pairplot(df[features_subset], hue='quality', markers='o')
plt.suptitle('Pairplot of Selected Features')
plt.show()
```

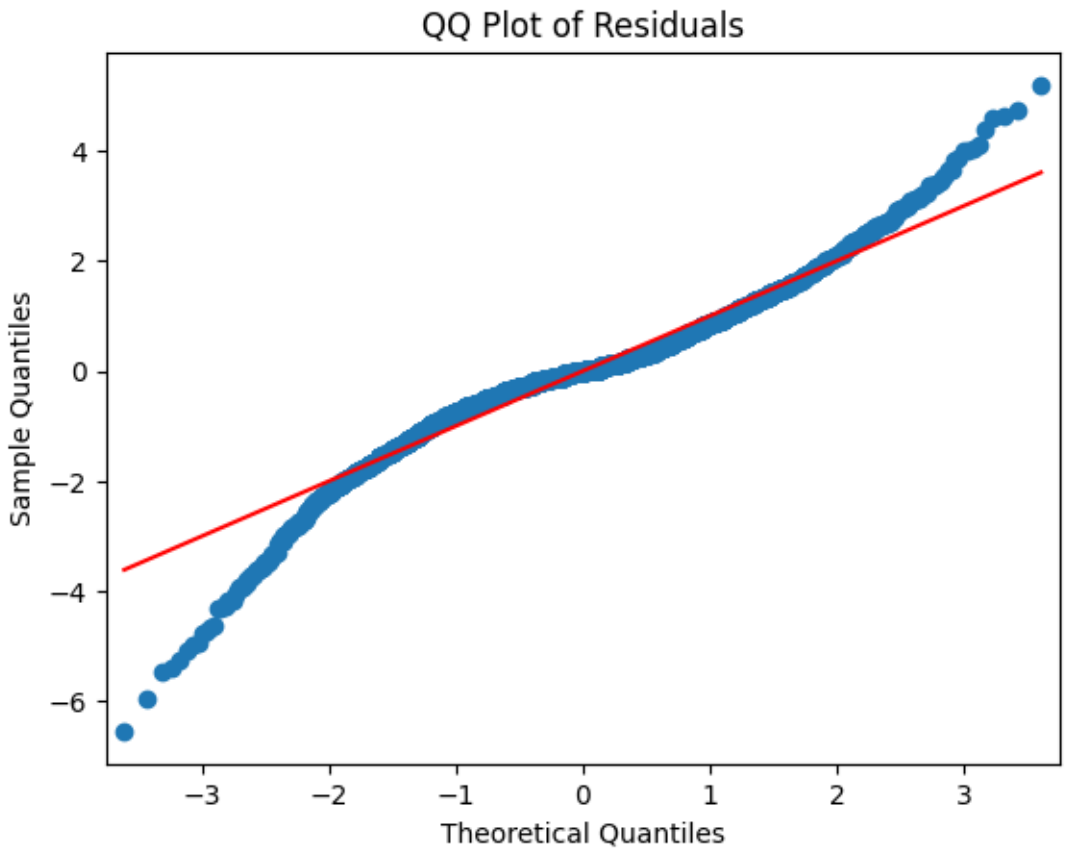


[12]:
```
# Correlation matrix heatmap
correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='viridis', fmt=".2f",␣
 ↪linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

[13]:
```
# Boxplot 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)
    plt.title(f'Boxplot of {feature}')
plt.tight_layout()
plt.show()
```



[14]:
```
# 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
```

[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]:
```
# 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]:
```
# 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]:
```
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
```