

AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

408/1, Kuratoli, Khilkhet, Dhaka 1229,
Bangladesh



Project Title: Apply Knn with Wine Quality-White
Dataset

Project: Mid-Term

Date of Submission:

Course Title: DATA WAREHOUSING AND DATA
MINING

Course Code:

Section: C

Semester: Fall-2023-2024

Course Teacher: DR. AKINUL ISLAM
JONY

Declaration and Statement of Authorship:

1. I/we hold a copy of this Assignment/Case-Study, which can be produced if the original is lost/damaged.
2. This Assignment/Case-Study is my/our original work and no part of it has been copied from any other student's work or from any other source except where due acknowledgement is made.
3. No part of this Assignment/Case-Study has been written for me/us by any other person except where such collaboration has been authorized by the concerned teacher and is clearly acknowledged in the assignment.
4. I/we have not previously submitted or currently submitting this work for any other course/unit.
5. This work may be reproduced, communicated, compared and archived for the purpose of detecting plagiarism.
6. I/we give permission for a copy of my/our marked work to be retained by the Faculty for review and comparison, including review by external examiners.
7. I/we understand that Plagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a form of cheating and is a very serious academic offence that may lead to expulsion from the University. Plagiarized material can be drawn from, and presented in, written, graphic and visual form, including electronic data, and oral presentations. Plagiarism occurs when the origin of them arterial used is not appropriately cited.
8. I/we also understand that enabling plagiarism is the act of assisting or allowing another person to plagiarize or to copy my/our work.

* Student(s) must complete all details except the faculty use part.

** Please submit all assignments to your course teacher or the office of the concerned teacher.

Group Name/No.:

No	Name	ID	Program	Signature
1	Shahriar Soudeep	20-43823-2	BSc.CSE	Soudeep
2	Most, Lilun Nahar Aurthy	20-43997-2	BSc.CSE	Aurthy
3	Tasnimul Hasan	20-43833-2	BSc.CSE	Tasnimul
4	Limia Saina Sathi	20-43851-2	BSc.CSE	Sathi
5				

Faculty use only

FACULTY COMMENTS	Marks Obtained	
	Total Marks	

Dataset Description:

This “wine quality-white.csv” dataset provides information about various attributes of white wines and their corresponding quality ratings. It is designed to analysis and model the factors that influence the quality of white wines.

The dataset includes the following column:

1. **Fixed Acidity:** This is a measure of the non-volatile acids present in the wine.
2. **Volatile Acidity:** This refers to the number of volatile acids in the wine, which can contribute to an unpleasant vinegar-like taste.
3. **Citric Acid:** This is the concentration of citric acid in the wine, which can influence the wine's freshness and flavor.
4. **Residual Sugar:** The amount of sugar remaining in the wine after fermentation. It can affect the wine's sweetness.
5. **Chlorides:** The concentration of salts in the wine, which can impact taste and mouthfeel.
6. **Free Sulfur Dioxide:** The presence of sulfur dioxide, which is used as a preservative and can affect the wine's stability.
7. **Total Sulfur Dioxide:** The sum of both free and bound sulfur dioxide.
8. **Density:** The density of the wine, which is related to its alcohol content.
9. **pH:** The pH level of the wine, which can affect its acidity and overall taste.
10. **Sulphates:** The concentration of sulfates, which can contribute to wine stability and flavor.
11. **Alcohol:** The alcohol content of the wine.
12. **Quality:** This is the target variable and represents the overall quality rating of the wine. It's usually numeric values range from 1 to 10.

Overview:

This code utilizes the pandas, numpy, seaborn, and scikit-learn libraries to perform an analysis on a white wine quality dataset. The dataset is loaded into a pandas DataFrame, and initial data exploration is conducted, including checking for missing values and removing duplicates. The code then visualizes the correlation matrix of the dataset using a heatmap. Subsequently, it addresses outliers in numeric columns by applying the interquartile range (IQR) method. The dataset is then split into training and testing sets, and a k-nearest neighbors (KNN) classification model is implemented to predict wine quality. Three distance metrics (Euclidean, Manhattan, and Maximum) are employed, and their respective accuracies are evaluated. Finally, the code predicts the quality of a new wine sample using the trained KNN model for each distance metric. The analysis provides insights into the relationships between different features of white wines and demonstrates the application of a KNN classifier for quality prediction.

Import Dataset:

Code:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
import pandas as pd

data = pd.read_csv('/content/winequality-white.csv')
print(data.describe())
```

Output:

	fixed acidity	volatile acidity	citric acid	residual sugar	\
count	4898.000000	4898.000000	4898.000000	4898.000000	
mean	6.854788	0.278241	0.334192	6.391415	
std	0.843868	0.100795	0.121020	5.072058	
min	3.800000	0.080000	0.000000	0.600000	
25%	6.300000	0.210000	0.270000	1.700000	
50%	6.800000	0.260000	0.320000	5.200000	
75%	7.300000	0.320000	0.390000	9.900000	
max	14.200000	1.100000	1.660000	65.800000	
	chlorides	free sulfur dioxide	total sulfur dioxide	density	\
count	4898.000000	4898.000000	4898.000000	4898.000000	
mean	0.045772	35.308085	138.360657	0.994027	
std	0.021848	17.007137	42.498065	0.002991	
min	0.009000	2.000000	9.000000	0.987110	
25%	0.036000	23.000000	108.000000	0.991723	
50%	0.043000	34.000000	134.000000	0.993740	
75%	0.050000	46.000000	167.000000	0.996100	
max	0.346000	289.000000	440.000000	1.038980	
	pH	sulphates	alcohol	quality	
count	4898.000000	4898.000000	4898.000000	4898.000000	
mean	3.188267	0.489847	10.514267	5.877909	
std	0.151001	0.114126	1.230621	0.885639	
min	2.720000	0.220000	8.000000	3.000000	
25%	3.090000	0.410000	9.500000	5.000000	
50%	3.180000	0.470000	10.400000	6.000000	
75%	3.280000	0.550000	11.400000	6.000000	
max	3.800000	1.000000	17.000000	8.000000	

Description:

This Python script uses the pandas library to read a white wine quality dataset from a CSV file. The data is loaded into a DataFrame named 'data.' The 'describe()' method generates and prints a statistical summary of the DataFrame, offering insights into key measures such as mean, standard deviation, and percentiles. This concise code snippet serves as an initial exploration of the dataset's statistical properties.

Dataset Details(Total Data):

Code:

```
data.info()
```

Output:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype  
---  -
 0   fixed acidity        4898 non-null   float64
 1   volatile acidity     4898 non-null   float64
 2   citric acid          4898 non-null   float64
 3   residual sugar       4898 non-null   float64
 4   chlorides            4898 non-null   float64
 5   free sulfur dioxide  4898 non-null   float64
 6   total sulfur dioxide 4898 non-null   float64
 7   density              4898 non-null   float64
 8   pH                  4898 non-null   float64
 9   sulphates            4898 non-null   float64
10   alcohol              4898 non-null   float64
11   quality              4898 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
```

Description:

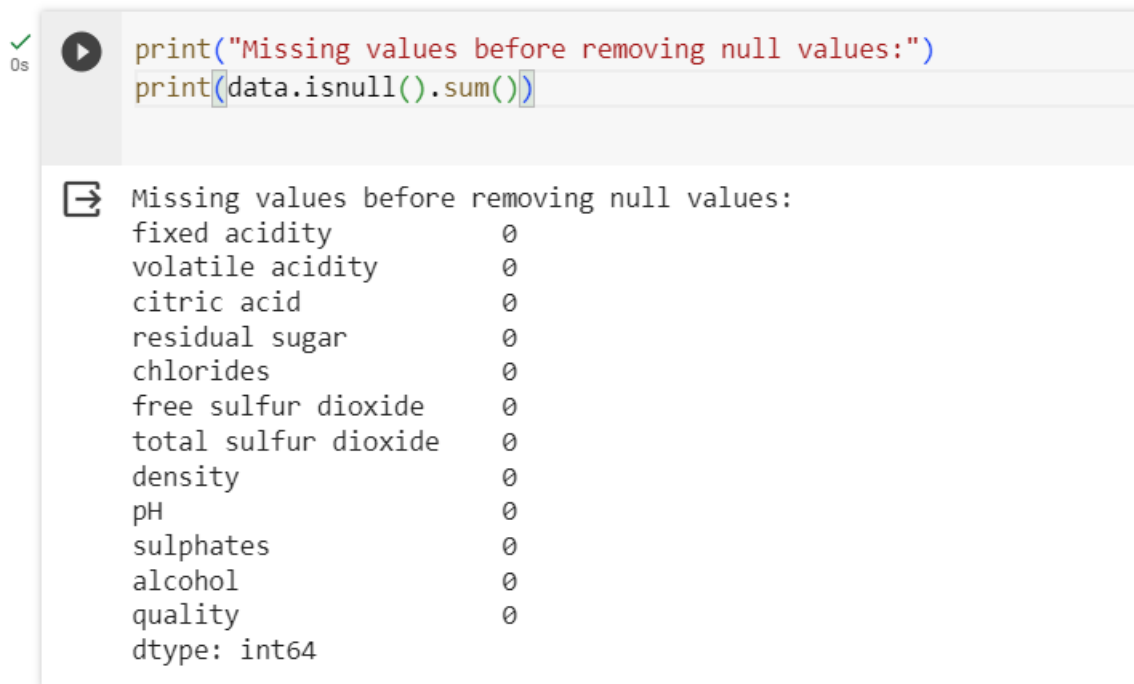
The code 'data.info()' employs the pandas library to provide a summary of the dataset. This includes information on the total non-null entries, data types for each column, and memory usage. The output from 'info()' is valuable for understanding the structure of the dataset, identifying potential missing values, and gaining insights into the data types present in each column.

Missing Values:

Code:

```
print("Missing values values:")  
print(data.isnull().sum())
```

Output:



The image shows a Jupyter Notebook interface. At the top, there is a green checkmark and a play button icon. Below this, the code from the previous block is displayed in a text area. The output of the code is shown in a separate box below the code. It starts with a text prompt 'Missing values before removing null values:' followed by a list of 12 variables and their corresponding sum of null values, which are all 0. The last line of the output is 'dtype: int64'.

```
print("Missing values before removing null values:")  
print(data.isnull().sum())
```

Missing values before removing null values:

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

dtype: int64

Description:

The code checks for missing values in the dataset using `isnull().sum()` and prints the sum of null values for each column. However, there are no null values present in “wine quality-white.csv” dataset. This preemptive check is essential for data integrity, ensuring that the dataset is complete and ready for analysis.

Duplicate Values:

Code:

```
duplicate_rows = data[data.duplicated()]  
print("Duplicate rows before removing the duplicate rows:")  
print(duplicate_rows)  
data = data.drop_duplicates()  
  
duplicate_rows_after = data[data.duplicated()]  
print("Duplicate rows after removing the duplicate values:",  
      duplicate_rows_after)
```

Output:

```
Duplicate rows before removing the duplicate rows:
  fixed acidity  volatile acidity  citric acid  residual sugar  chlorides \
4             7.2              0.23        0.32             8.5      0.058
5             8.1              0.28        0.40             6.9      0.050
7             7.0              0.27        0.36            20.7      0.045
8             6.3              0.30        0.34             1.6      0.049
20            6.2              0.66        0.48             1.2      0.029
...           ...              ...          ...           ...       ...
4828           6.4              0.23        0.35            10.3      0.042
4850           7.0              0.36        0.35             2.5      0.048
4851           6.4              0.33        0.44             8.9      0.055
4856           7.1              0.23        0.39            13.7      0.058
4880           6.6              0.34        0.40             8.1      0.046

  free sulfur dioxide  total sulfur dioxide  density  pH  sulphates \
4                   47.0                186.0  0.99560  3.19      0.40
5                   30.0                 97.0  0.99510  3.26      0.44
7                   45.0                170.0  1.00100  3.00      0.45
8                   14.0                132.0  0.99400  3.30      0.49
20                  29.0                 75.0  0.98920  3.33      0.39

[937 rows x 12 columns]
Duplicate rows after removing the duplicate values: Empty DataFrame
Columns: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates
Row labels: None
```

Description:

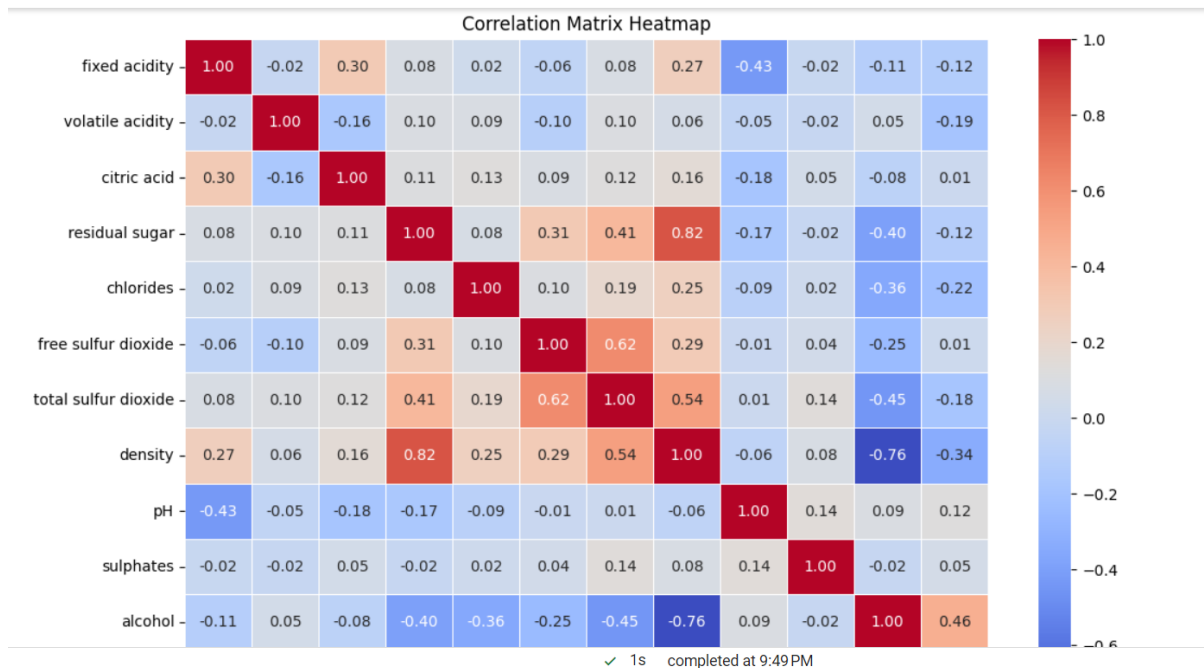
The code identifies and prints duplicate rows in the 'data' dataset using the `duplicated()` function. Initially, it displays the duplicate rows before removal. Afterward, it utilizes `drop_duplicates()` to eliminate the duplicate entries. The subsequent check (`duplicate_rows_after`) ensures that the removal was successful, and the print statement indicates the absence of duplicate rows in the modified dataset. This process is crucial for maintaining data accuracy by eliminating redundant information. In our Dataset there was some duplicate values which shows in the output section.

Correlation (Heatmap):

Code:

```
correlation_matrix = data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",
            linewidths=.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

Output:



Description:

The code calculates the correlation matrix for the 'data' dataset using the `corr()` function. It then visualizes the correlations among variables with a heatmap using the Seaborn library. The heatmap employs colors from the 'coolwarm' colormap to represent correlation strength. Numeric values are annotated within each cell, providing a quick reference to correlation coefficients. This visualization aids in understanding relationships between different variables in the dataset, helping identify potential patterns or dependencies. In our dataset there is not any attributes which have '0' correlation with the target attribute.

Boxplot (Check Outliers):

Code:

```
numeric_columns = data.select_dtypes(include=[np.number]).columns

for col in numeric_columns:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x=data[col])
    plt.title(f'Before removing outliers from: {col}')
    plt.show()
```

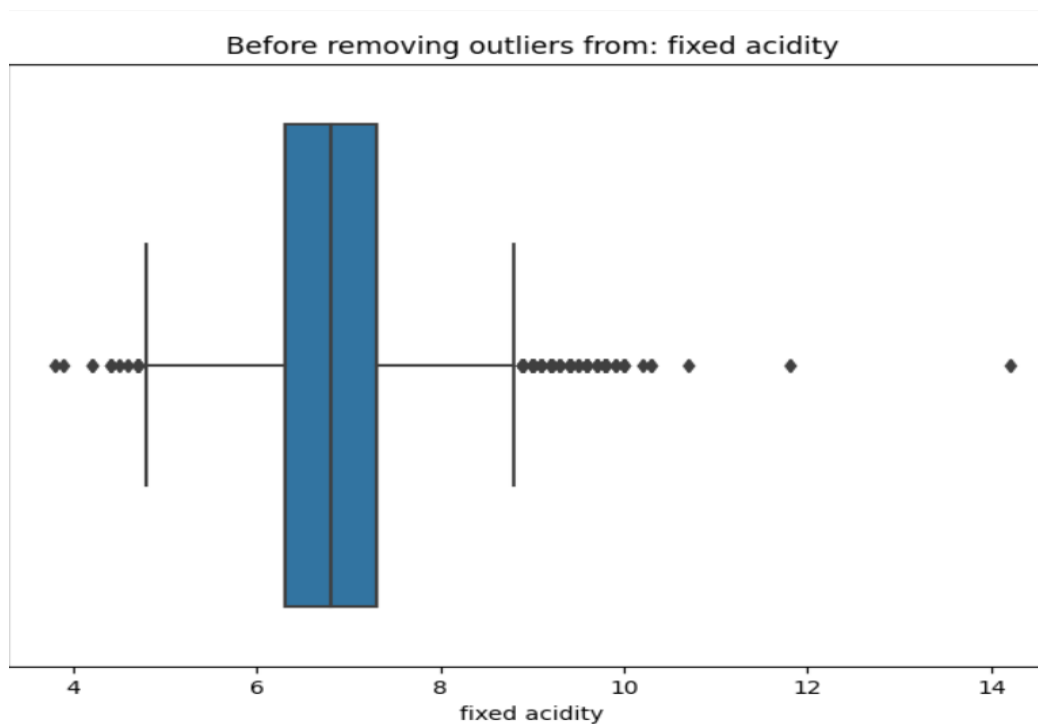
```
def remove_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return data[(data[column] >= lower_bound) & (data[column] <=
upper_bound)]

for col in numeric_columns:
    data = remove_outliers_iqr(data, col)

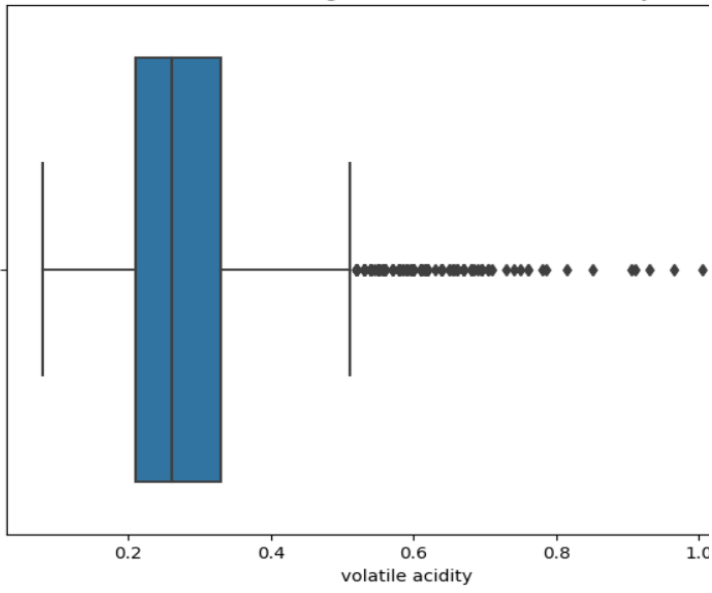
for col in numeric_columns:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x=data[col])
    plt.title(f'After removing outliers from: {col}')
    plt.show()
```

Output:

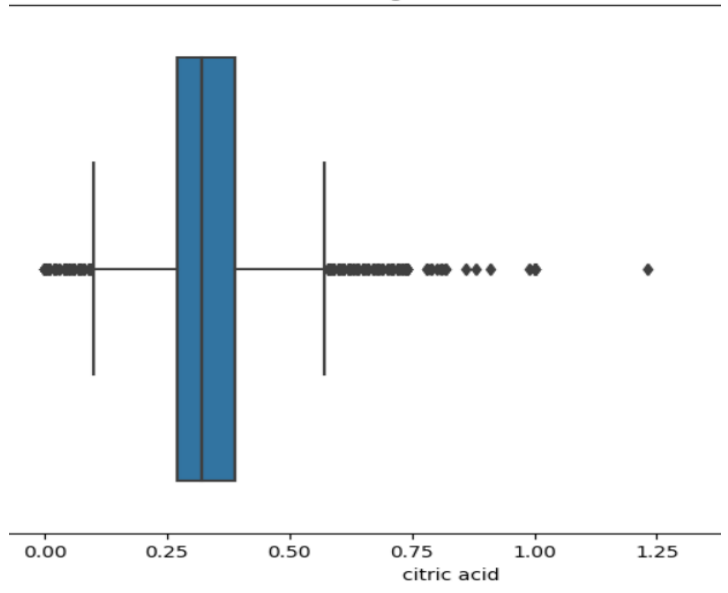
Before Removing Outliers:



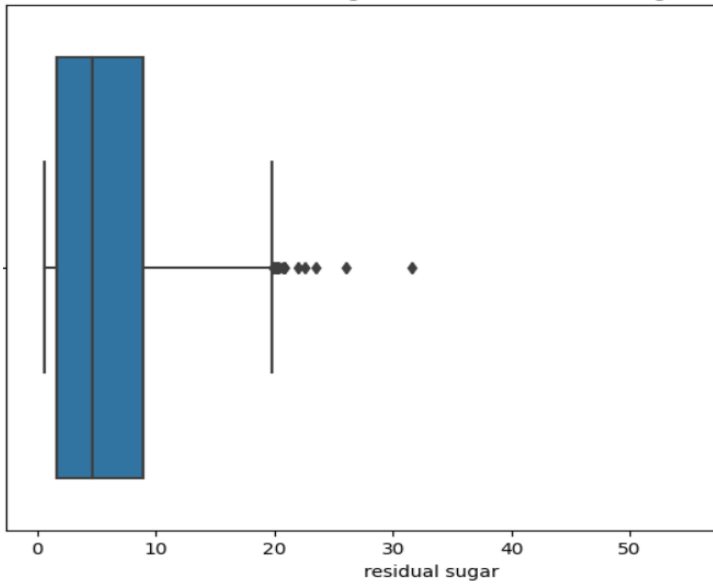
Before removing outliers from: volatile acidity



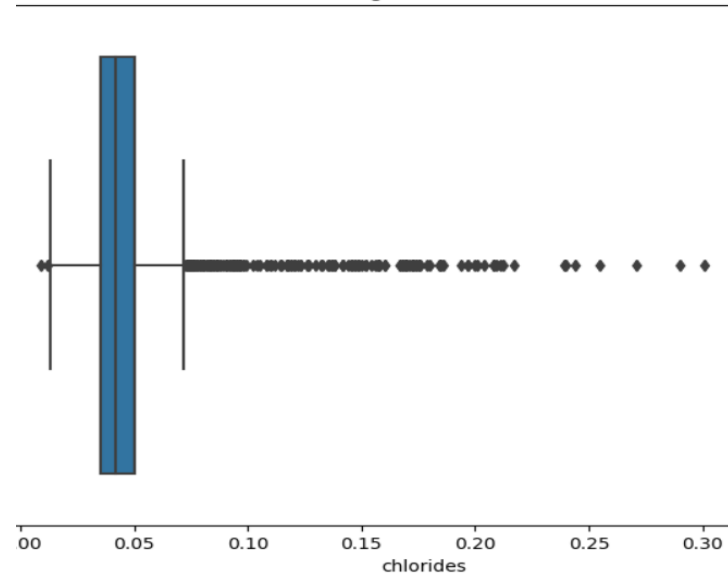
Before removing outliers from: citric acid



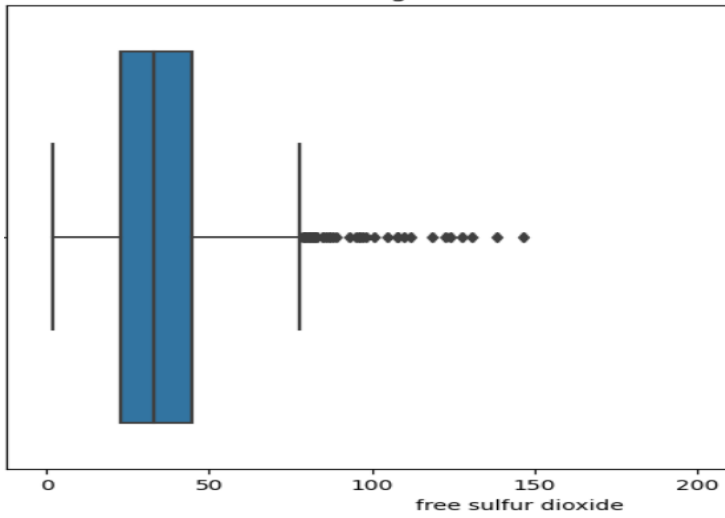
Before removing outliers from: residual sugar



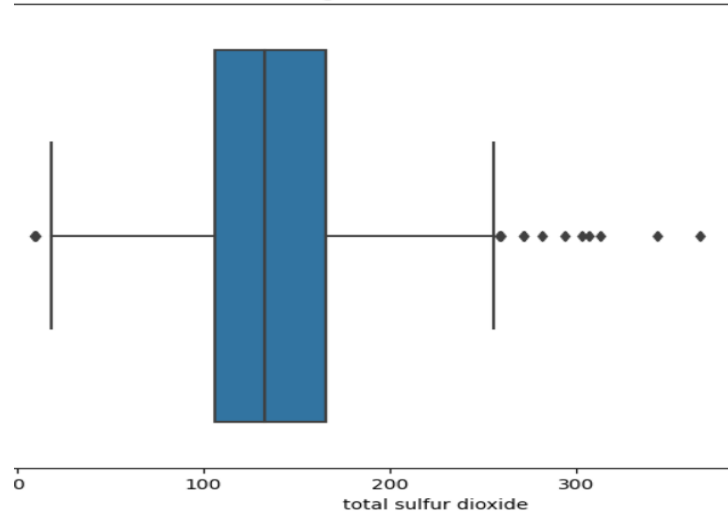
Before removing outliers from: chlorides

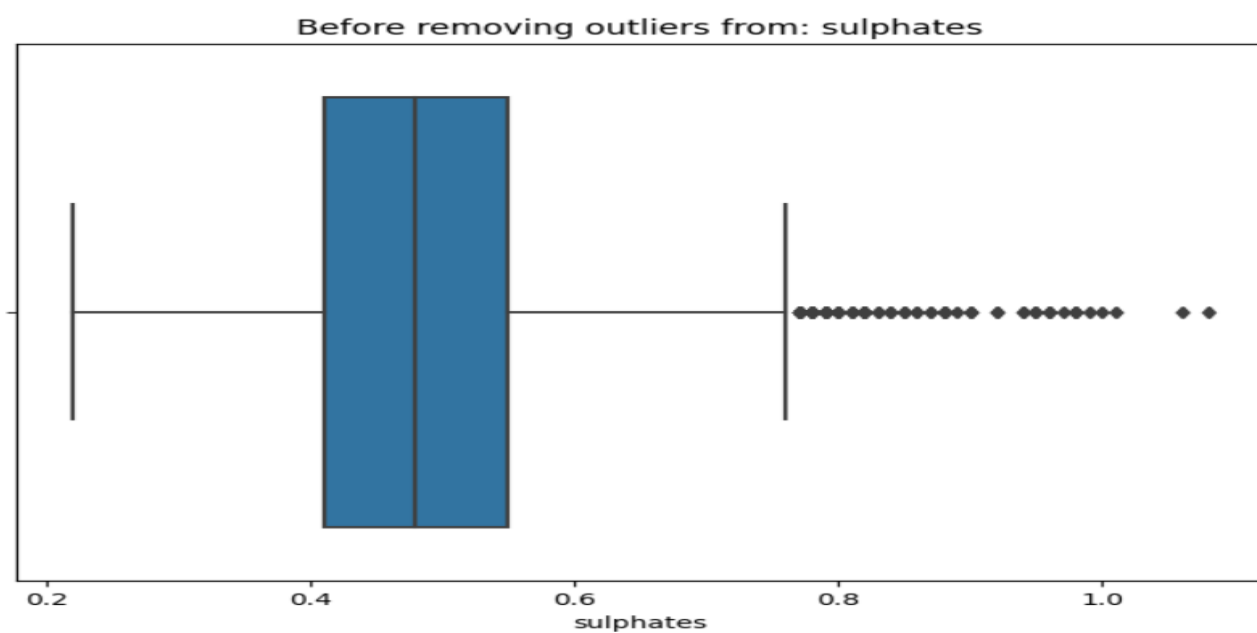
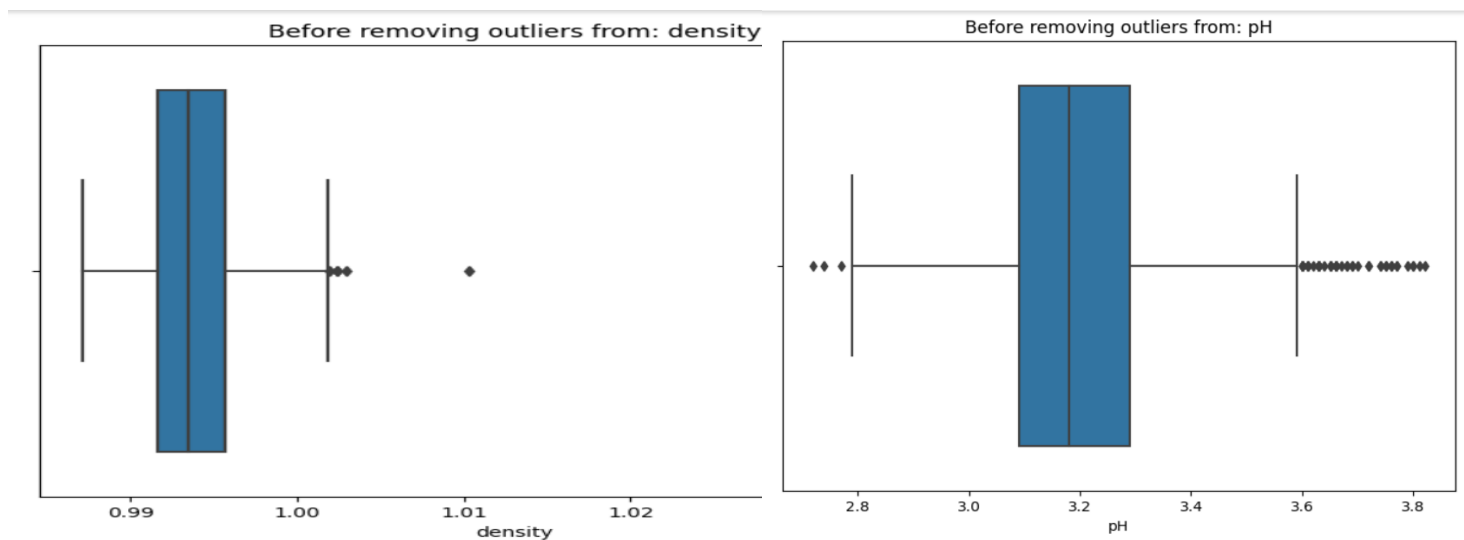


Before removing outliers from: free sulfur dioxide

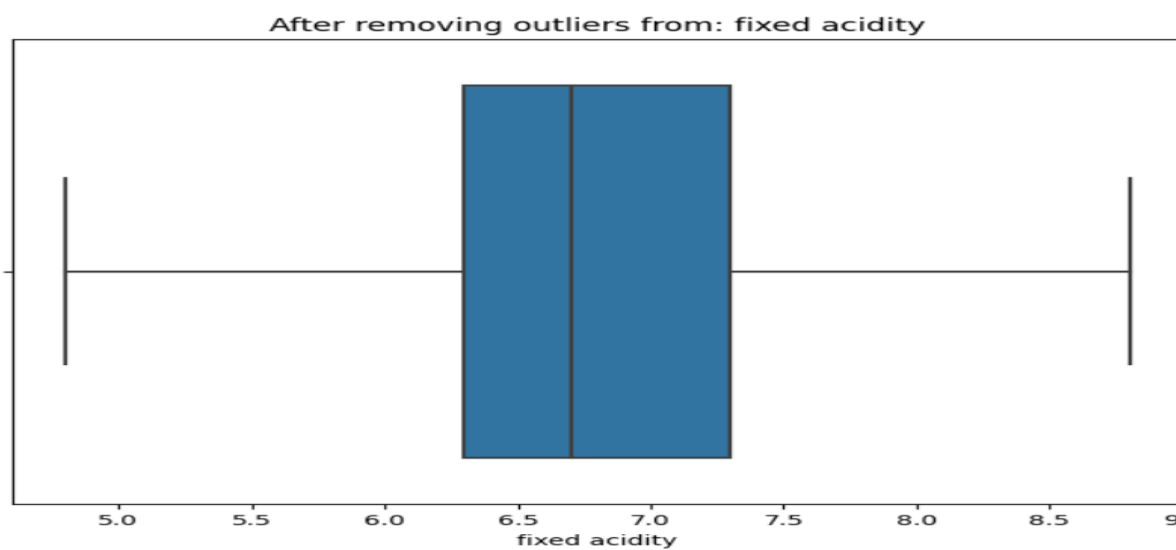


Before removing outliers from: total sulfur dioxide

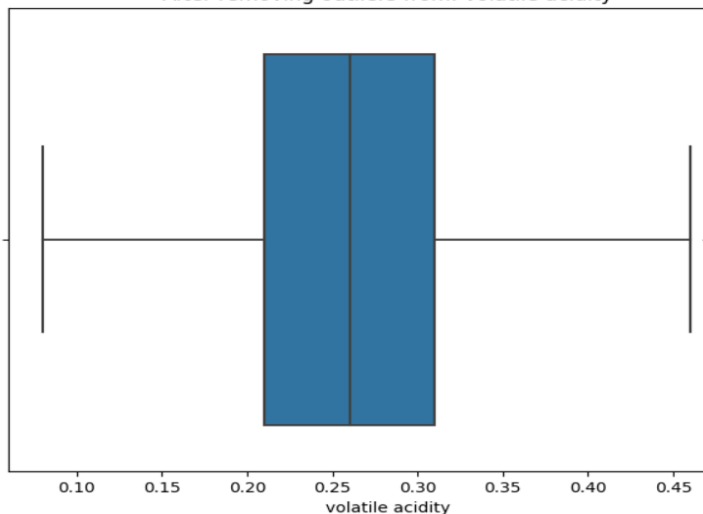




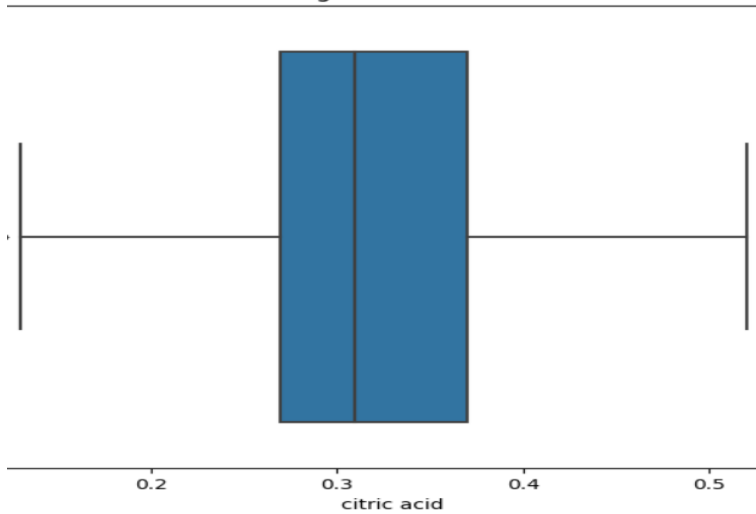
After Removing Outliers:



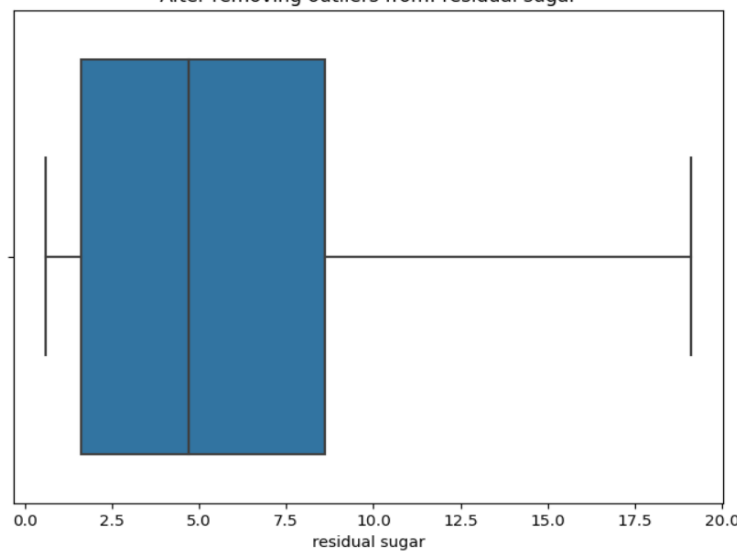
After removing outliers from: volatile acidity



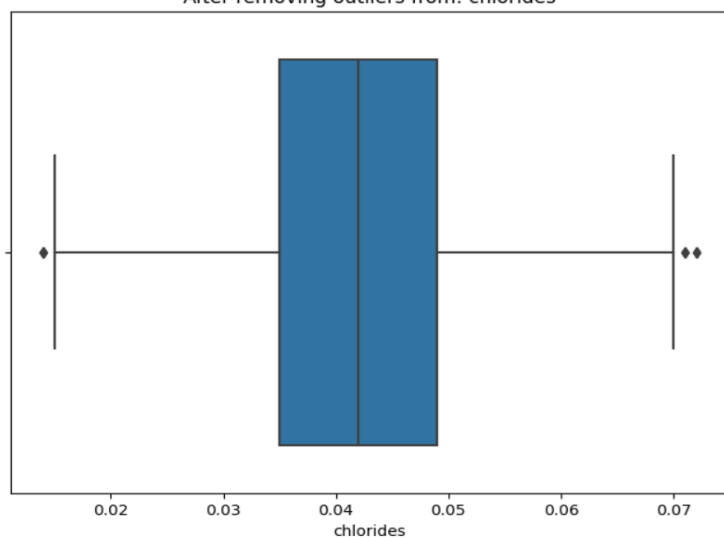
After removing outliers from: citric acid



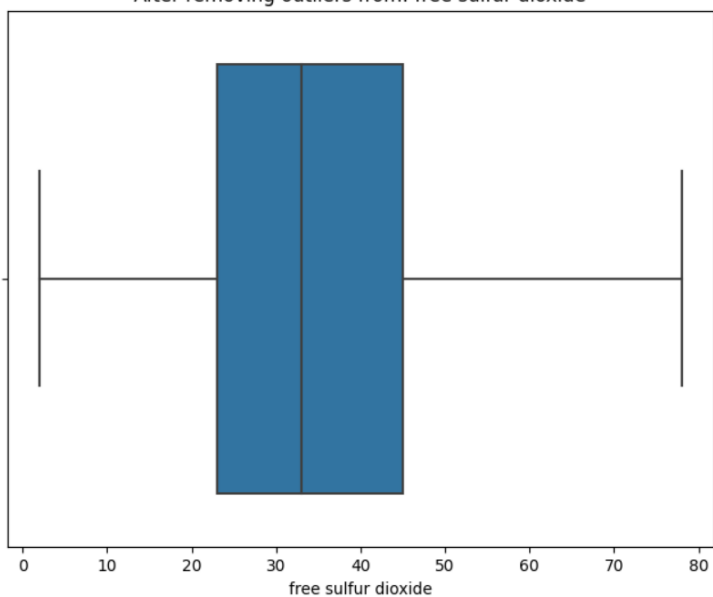
After removing outliers from: residual sugar



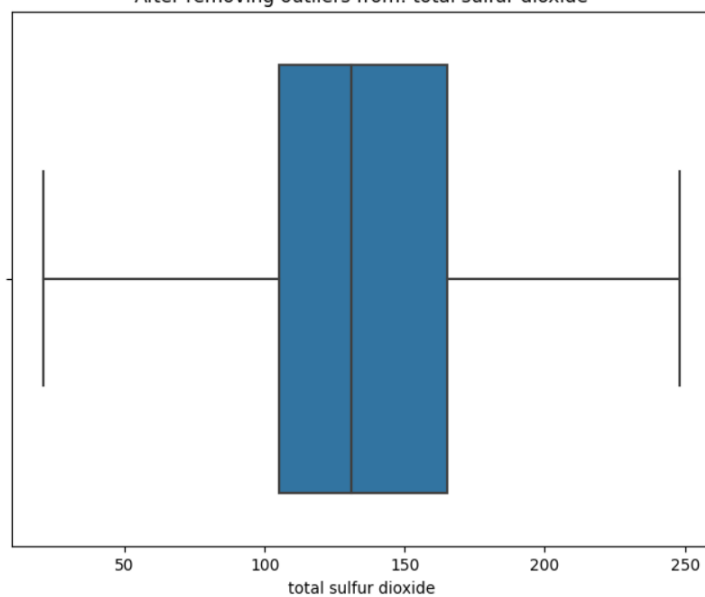
After removing outliers from: chlorides

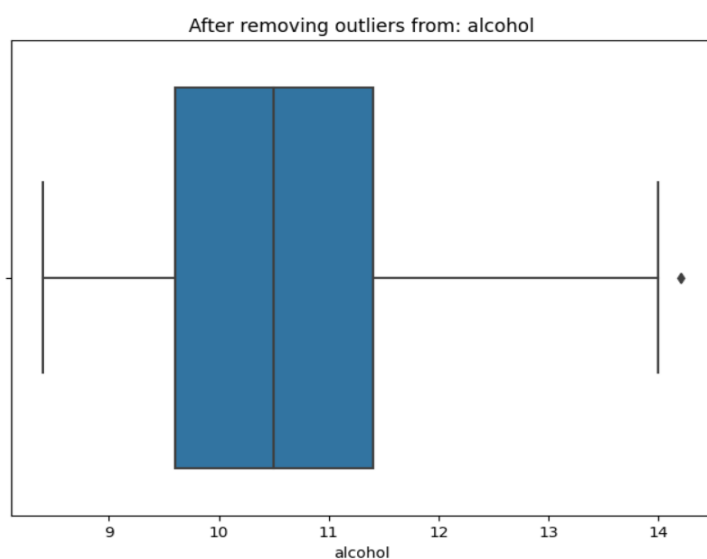
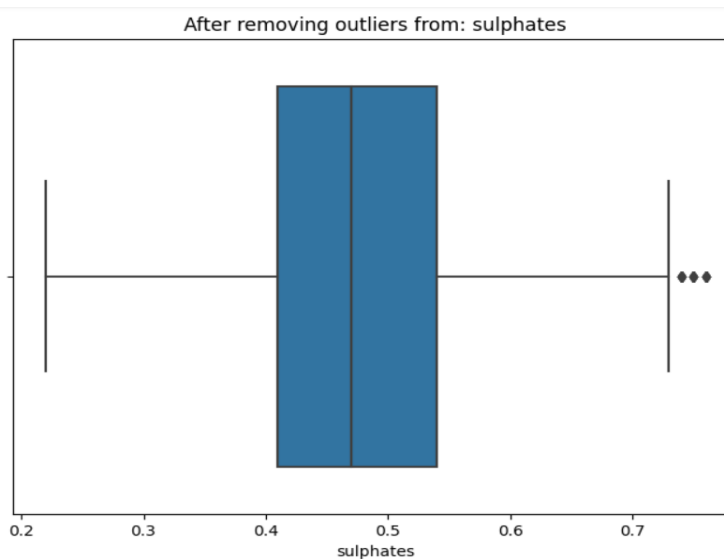
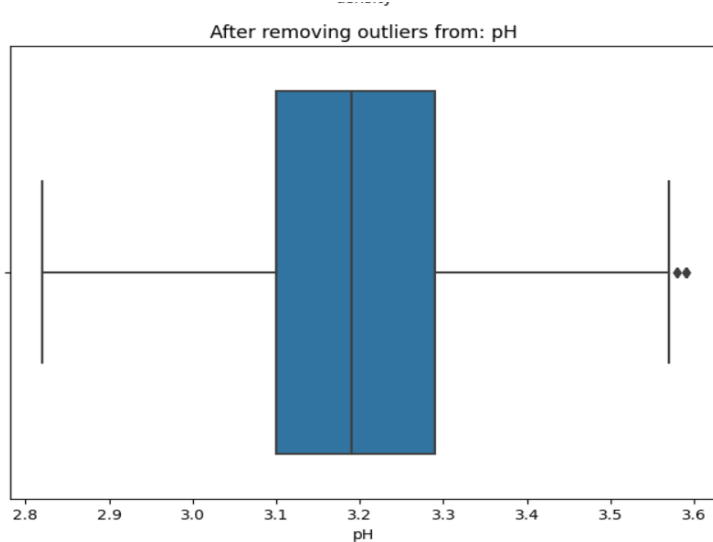
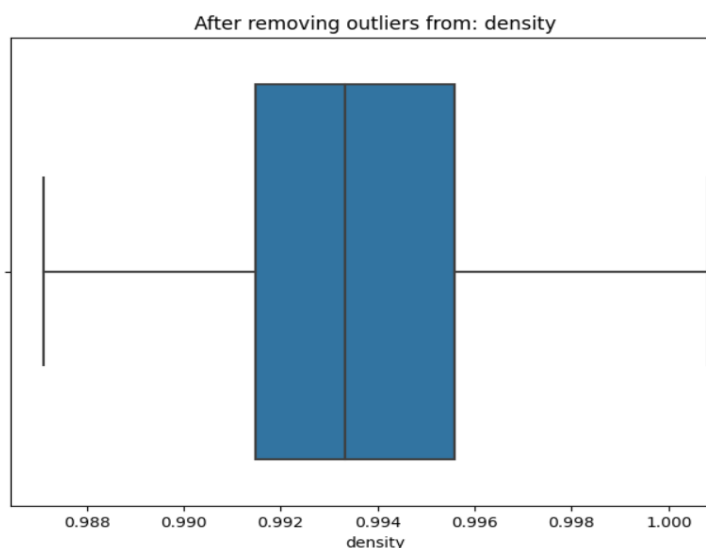


After removing outliers from: free sulfur dioxide



After removing outliers from: total sulfur dioxide





Description:

The code initially identifies and visualizes potential outliers in numeric columns of the 'data' dataset using boxplots. It iterates through each numeric column, displaying a boxplot for the distribution of values before outlier removal. Less than 25% and greater than 75% quartile range data are called outliers. Subsequently, the code defines a function `remove_outliers_iqr` that utilizes the Interquartile Range (IQR) method to filter out outliers from a given column. The function is then applied to each numeric column in the dataset, effectively removing outliers. Finally, the code generates new boxplots for each numeric column, illustrating the distribution of values after the outlier removal process.

Normalization and Splitting:

Code:

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


scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

print(X_scaled)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
```

Output:



```
[[ -0.66046505  0.5029405  0.21313892 ...  0.75517503  0.11366973
 -0.98274713]
 [ 1.72722175  0.23346008  0.94798361 ...  0.46370137 -0.41749555
 -0.46714585]
 [ 0.53337835 -0.44024096 -0.03180931 ... -0.04637753 -0.84242778
 -0.63901294]
 ...
 [-0.39516652 -0.30550075 -1.6239728  ... -1.50374582 -0.20502944
 -1.06868068]
 [-1.72165918  0.36820029 -0.27675754 ...  1.04664868 -1.05489389
 1.85305993]
 [-1.05841285 -0.70972138  0.70303538 ...  0.46370137 -1.69229223
 0.99372446]]
```

Description:

The provided code implies for a machine learning model. Firstly, it separates the target variable, 'quality,' from the feature set 'X.' Next, it employs the 'StandardScaler' from scikit-learn to normalize the feature set, ensuring that all features are on a comparable scale. This step is crucial for models sensitive to the magnitude of input features, enhancing their performance. Following normalization, the data is split into training and testing sets using the 'train_test_split' function. 80% of the data is allocated to training ('X_train' and 'y_train'), while the remaining 20% is reserved for testing ('X_test' and 'y_test'). This division facilitates model training on one subset and evaluation on another, assessing the model's generalization to new, unseen data.

KNN:

Euclidian Distance:

Code:

```
def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

a=3
b=5
c=7
n=c

def knn_predict(train_data, train_labels, test_data, k=n,
distance_metric=euclidean_distance):
    predictions = []
    for test_point in test_data:
        distances = [distance_metric(test_point, train_point) for
train_point in train_data]

        k_indices = np.argsort(distances)[:k]

        k_nearest_labels = train_labels.iloc[k_indices].tolist()

        most_common = Counter(k_nearest_labels).most_common()

        prediction = min(most_common, key=lambda x: (-x[1], x[0]))[0]

        predictions.append(prediction)

    return predictions

predictions_euclidean = knn_predict(X_train, y_train, X_test, k=n,
distance_metric=euclidean_distance)

accuracy_euclidean = accuracy_score(y_test, predictions_euclidean)

print(f'Accuracy (Euclidean): {accuracy_euclidean:.2f}')
```



```
new_wine_sample = np.array([7.0, 1.5, 5.3, 2.0, 0.18, 20, 50, 0.995,
3.3, 0.6, 10.0]).reshape(1, -1)
predicted_quality_euclidean = knn_predict(X_train, y_train,
new_wine_sample, k=n, distance_metric=euclidean_distance)
print(f'Predicted Quality(Euclidean):
{predicted_quality_euclidean[0]}')
```

Output:

K=3:

```
▶ Accuracy (Euclidean): 0.49  
  Predicted Quality(Euclidean): 6
```

K=5

```
⇒ Accuracy (Euclidean): 0.52  
  Predicted Quality(Euclidean): 6
```

K=7

```
⇒ Accuracy (Euclidean): 0.53  
  Predicted Quality(Euclidean): 6
```

Description:

The code implements a k-Nearest Neighbors (k-NN) algorithm using the Euclidean distance metric for wine quality prediction. It assesses the model's performance for k values of 3, 5, and 7 using the `'knn_predict'` function.

For k=3, k=5, and k=7, the algorithm considers the respective three, five, and seven nearest neighbors when making predictions on the test set. The accuracy of the model is calculated and printed for each k value. Additionally, the code predicts the quality of a new wine sample using the Euclidean distance metric, offering insights into the model's behavior for different neighborhood sizes, aiding in the selection of an optimal k value for the specific wine quality prediction task.

Manhattan Distance:

Code:

```
def manhattan_distance(point1, point2):
    return np.sum(np.abs(point1 - point2))

a=3
b=5
c=7
n=c

def knn_predict(train_data, train_labels, test_data, k=n,
distance_metric=manhattan_distance):
    predictions = []
    for test_point in test_data:
        distances = [distance_metric(test_point, train_point) for
train_point in train_data]

        k_indices = np.argsort(distances)[:k]

        k_nearest_labels = train_labels.iloc[k_indices].tolist()

        most_common = Counter(k_nearest_labels).most_common()

        prediction = min(most_common, key=lambda x: (-x[1], x[0]))[0]

        predictions.append(prediction)

    return predictions

predictions_manhattan = knn_predict(X_train, y_train, X_test, k=n,
distance_metric=manhattan_distance)

accuracy_manhattan = accuracy_score(y_test, predictions_manhattan)

print(f'Accuracy (Manhattan): {accuracy_manhattan:.2f}')
```



```
predicted_quality_manhattan = knn_predict(X_train, y_train,
new_wine_sample, k=n, distance_metric=manhattan_distance)
print(f'Predicted Quality(Manhattan):
{predicted_quality_manhattan[0]}')
```


Output:

K=3

```
➞ Accuracy (Manhattan): 0.51  
   Predicted Quality(Manhattan): 6
```

K=5

```
➞ Accuracy (Manhattan): 0.54  
   Predicted Quality(Manhattan): 5
```

K=7

```
➞ Accuracy (Manhattan): 0.55  
   Predicted Quality(Manhattan): 5
```

Description:

The code extends the k-Nearest Neighbors (k-NN) algorithm using the Manhattan distance metric for wine quality prediction, evaluating the model for k values of 3, 5, and 7. The 'knn_predict' function employs the Manhattan distance metric to calculate distances and predict wine quality, iterating through the test set.

For k=3, k=5, and k=7, the algorithm considers the respective nearest neighbors when making predictions. The accuracy of the model is calculated on the test set for each k value, and the results are printed. Additionally, the code predicts the quality of a new wine sample based on Manhattan distance, showcasing the influence of different k values on model predictions. This comprehensive evaluation provides insights into the algorithm's performance with varying neighborhood sizes, aiding in the selection of an optimal k value for the specific wine quality prediction task.

Maximum Distance:

Code:

```
def max_distance(point1, point2):
    return np.max(np.abs(point1 - point2))

a=3
b=5
c=7
n=c

def knn_predict(train_data, train_labels, test_data, k=n,
distance_metric=max_distance):
    predictions = []
    for test_point in test_data:
        distances = [distance_metric(test_point, train_point) for
train_point in train_data]

        k_indices = np.argsort(distances)[:k]

        k_nearest_labels = train_labels.iloc[k_indices].tolist()

        most_common = Counter(k_nearest_labels).most_common()

        prediction = min(most_common, key=lambda x: (-x[1], x[0]))[0]

        predictions.append(prediction)

    return predictions

predictions_max = knn_predict(X_train, y_train, X_test, k=n,
distance_metric=max_distance)

accuracy_max = accuracy_score(y_test, predictions_max)

print(f'Accuracy (Maximum): {accuracy_max:.2f}')
```

```
predicted_quality_max = knn_predict(X_train, y_train, new_wine_sample,
k=n, distance_metric=max_distance)
print(f'Predicted Quality(Maximum): {predicted_quality_max[0]}')
```

Output:

K=3

```
Accuracy (Maximum): 0.55  
Predicted Quality(Maximum): 6
```

K=5

```
➞ Accuracy (Maximum): 0.57  
Predicted Quality(Maximum): 6
```

K=7

```
➞ Accuracy (Maximum): 0.57  
Predicted Quality(Maximum): 6
```

Description:

The code enhances the k-Nearest Neighbors (k-NN) algorithm by introducing a new distance metric, the maximum distance, for wine quality prediction. It evaluates the model's performance for k values of 3, 5, and 7, using the `knn_predict` function with the maximum distance metric.

For k=3, k=5, and k=7, the algorithm considers the respective nearest neighbors when making predictions. The accuracy of the model is calculated on the test set for each k value, and the results are printed. Additionally, the code predicts the quality of a new wine sample based on the maximum distance metric, illustrating the impact of different k values on model predictions. The results contribute to the selection of an optimal k value and distance metric for the specific wine quality prediction task.