Wine Clustering Based on Chemical Composition

November 30, 2023

1 Wine Clustering Based on Chemical Composition

1.1 Author

Joel Ruetas

1.2 Abstract

In this project, I utilize clustering, an unsupervised machine learning technique, to categorize wines based on their physicochemical properties. The goal of clustering is to group data points with similar traits together and keep dissimilar ones apart (Seif, 2018). By applying this method to wines, I aim to identify groups that share similar chemical attributes, enabling personalized wine recommendations based on their chemical profiles.

1.3 Background

Wine flavors are influenced by grape varieties, the regions where they're grown, and their vintage year. While wines are traditionally classified based on these factors, the subtler nuances in flavor are often a result of variations in physicochemical properties such as pH, density, residual sugars, and sulphates (Cortez et al., 2009; Notman, 2018). This understanding suggests that wines with similar chemical make-ups might share taste profiles. Therefore, wines from the same region can have different flavors, and wines from different regions might taste similar.

In my research, I focus on analyzing the chemical similarities among wines to enhance my understanding of their flavor profiles. If a particular wine with specific chemical properties appeals to someone, they might also enjoy other wines from the same cluster with similar chemical compositions. This research can inform personal wine choices and deepen my understanding of wine characteristics.

1.4 Objective

My objective is to conduct a clustering analysis on red wines based on their physicochemical properties. This study aims to uncover associations within these properties, enhancing my knowledge and approach to wine selection and appreciation.

1.5 Data

I use the red wine quality dataset from the UCI Machine Learning Repository (http://www3.dsi.uminho.pt/pcortez/wine/winequality.zip) for this analysis. Although it is traditionally used for classification tasks, the dataset's 11 physicochemical features and 4898 observa-

tions make it suitable for clustering analysis. This project allows me to explore the relationships between the various physicochemical attributes of red wines.

1.6 References

Cortez,Paulo, Cerdeira,A., Almeida,F., Matos,T., and Reis,J.. (2009). *Wine Quality.* UCI Machine Learning Repository. https://doi.org/10.24432/C56S3T.

[ ]:
```
!pip install kneed
```

```
Requirement already satisfied: kneed in /usr/local/lib/python3.10/dist-packages
(0.8.5)
Requirement already satisfied: numpy>=1.14.2 in /usr/local/lib/python3.10/dist-
packages (from kneed) (1.23.5)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from kneed) (1.11.3)
```

1.7 Import necessary libraries

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pickle
import seaborn as sns

from kneed import KneeLocator
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler, PowerTransformer
```

1.8 Constants

[ ]:
```
random_state = 42
test_size = 0.2
```

1.9 Get Data

[ ]:
```
# Download and unzip the compressed file
!wget "http://www3.dsi.uminho.pt/pcortez/wine/winequality.zip"

!unzip winequality.zip
```

```
--2023-11-23 14:35:11--  http://www3.dsi.uminho.pt/pcortez/wine/winequality.zip
Resolving www3.dsi.uminho.pt (www3.dsi.uminho.pt)… 193.136.11.133
Connecting to www3.dsi.uminho.pt (www3.dsi.uminho.pt)|193.136.11.133|:80…
connected.
```

```
HTTP request sent, awaiting response… 200 OK
Length: 96005 (94K) [application/x-zip-compressed]
Saving to: ‘winequality.zip’

winequality.zip     100%[===================>]  93.75K   227KB/s    in 0.4s

2023-11-23 14:35:12 (227 KB/s) - ‘winequality.zip’ saved [96005/96005]

Archive:  winequality.zip
  inflating: winequality/winequality-names.txt
  inflating: winequality/winequality-names.txt.bak
  inflating: winequality/winequality-red.csv
  inflating: winequality/winequality-white.csv
```

[ ]:
```
# Read the CSV file
df_path = 'winequality/winequality-red.csv'
df = pd.read_csv(df_path, sep=';')

# View the first 5 rows
df.head()
```

[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

1.10 Exploratory Analysis Report

The subsequent section presents key findings and insights from an exploratory data analysis (EDA) process. The report encapsulates a wide range of data explorations, providing a visual and statistical overview of the dataset under study. Key highlights from this EDA report include:

1. **Missing Value Analysis:** Assessment of any missing data within the dataset and its potential implications on the analysis.
2. **Data Distribution and Trends:** Overview of the central tendencies, variability, and distribution patterns within the dataset.
3. **Correlation Analysis:** Investigation of potential relationships and correlations between different variables in the dataset, which could indicate underlying patterns or dependencies.
4. **Outlier Detection:** Identification of anomalies or outliers in the data that might affect the overall analysis or model performance.
5. **Comparative Analysis:** Comparative views of different subsets of data, allowing for a nuanced understanding of the dataset's characteristics.

These highlights form the crux of the EDA report, offering a foundational understanding of the dataset's characteristics and guiding subsequent analytical steps in the study.

[ ]:
```
# Print the full summary
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

[ ]:
```
# Check if dataset has any nulls
df.isnull().any().any()
```

[ ]:
```
False
```

[ ]:
```
# Check for missing values
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
dtype: int64
```

1.11 Dataset Characteristics for Clustering:

The dataset I am utilizing consists of 12 variables, encompassing 12 numeric predictors that offer a range of quantitative insights. These predictors include various physicochemical properties of wine, such as acidity, sugar levels, and alcohol content. Each variable contributes to understanding the distinct characteristics of different wines.

This dataset features 1599 individual observations, where each observation corresponds to a unique wine sample, characterized by these 12 variables. The ample number of observations ensures a thorough analysis, providing a robust foundation for identifying patterns and clusters within the wine samples.

One of the key strengths of this dataset is its integrity, as it is complete with no missing values. This is a crucial factor in clustering analysis, as the absence of missing data means there is no need for imputation techniques that could potentially skew the clustering results. Complete datasets allow for more accurate and reliable analyses, as each observation contributes a full set of data points.

In the context of clustering, these variables will be used to group wines into clusters based on their similarities across these physicochemical properties. The goal is to identify natural groupings within the wines, which could reveal insights into their inherent characteristics based on their chemical composition. This approach seeks to uncover hidden structures within the data based on the properties of the wines themselves.

1.12 Distributions

[ ]:
```
# Display the distribution of the numerical features
df.describe()
```

[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	\
count	1599.000000	1599.000000	1599.000000	1599.000000	
mean	8.319637	0.527821	0.270976	2.538806	
std	1.741096	0.179060	0.194801	1.409928	
min	4.600000	0.120000	0.000000	0.900000	
25%	7.100000	0.390000	0.090000	1.900000	

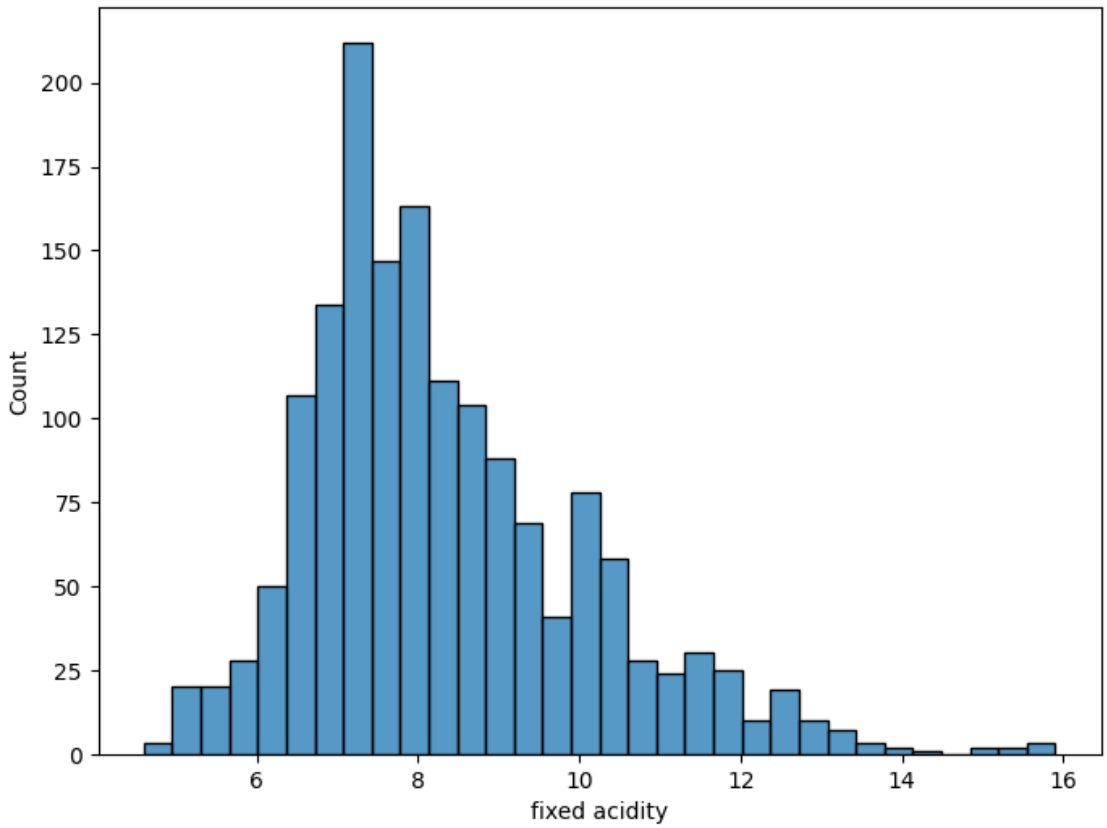
```
50%          7.900000          0.520000     0.260000        2.200000
75%          9.200000          0.640000     0.420000        2.600000
max         15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1599.000000          1599.000000           1599.000000  1599.000000
mean      0.087467            15.874922             46.467792     0.996747
std       0.047065            10.460157             32.895324     0.001887
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             22.000000     0.995600
50%       0.079000            14.000000             38.000000     0.996750
75%       0.090000            21.000000             62.000000     0.997835
max       0.611000            72.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

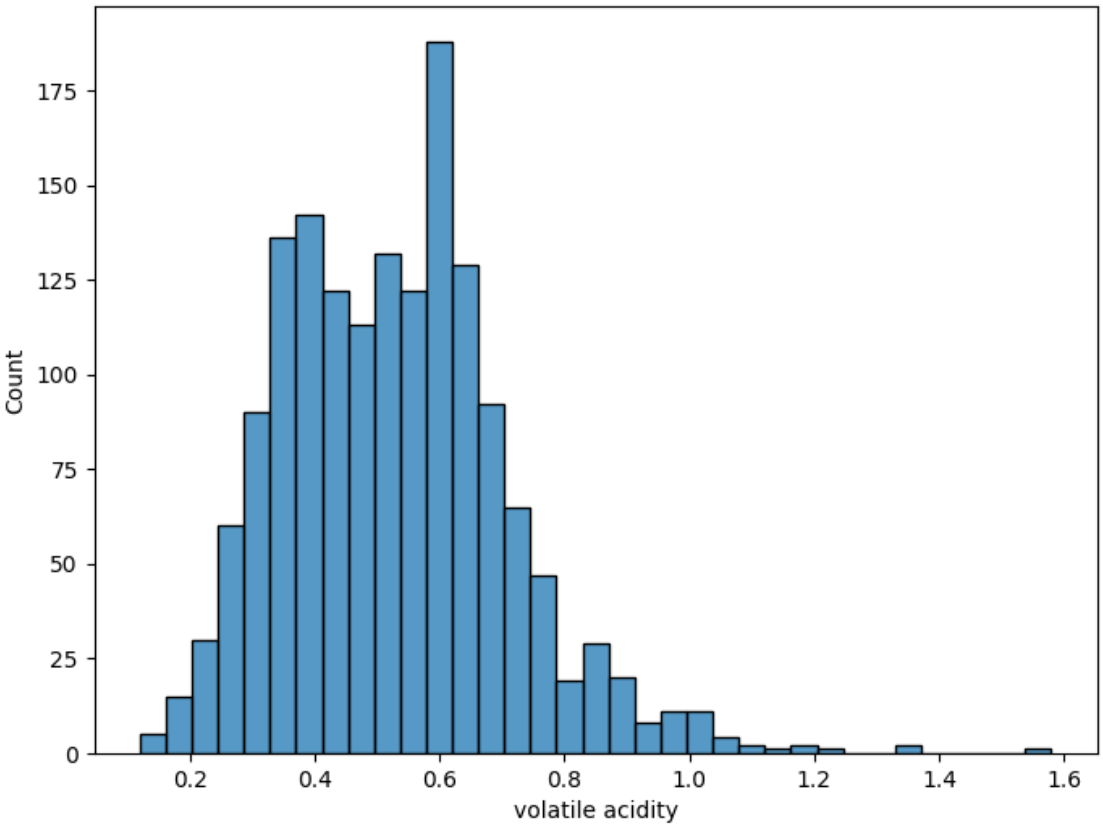
[ ]:
```
# Calculate the skewness of each column in the DataFrame
# Skewness is a measure of the asymmetry of the probability distribution of a␣
 ↪real-valued random variable
# A skewness value greater than 0 means that there is more weight in the right␣
 ↪tail of the distribution
# A skewness value less than 0 indicates more weight in the left tail of the␣
 ↪distribution
# A skewness value close to 0 (between -0.5 and 0.5) indicates a symmetric␣
 ↪distribution
df.skew()
```

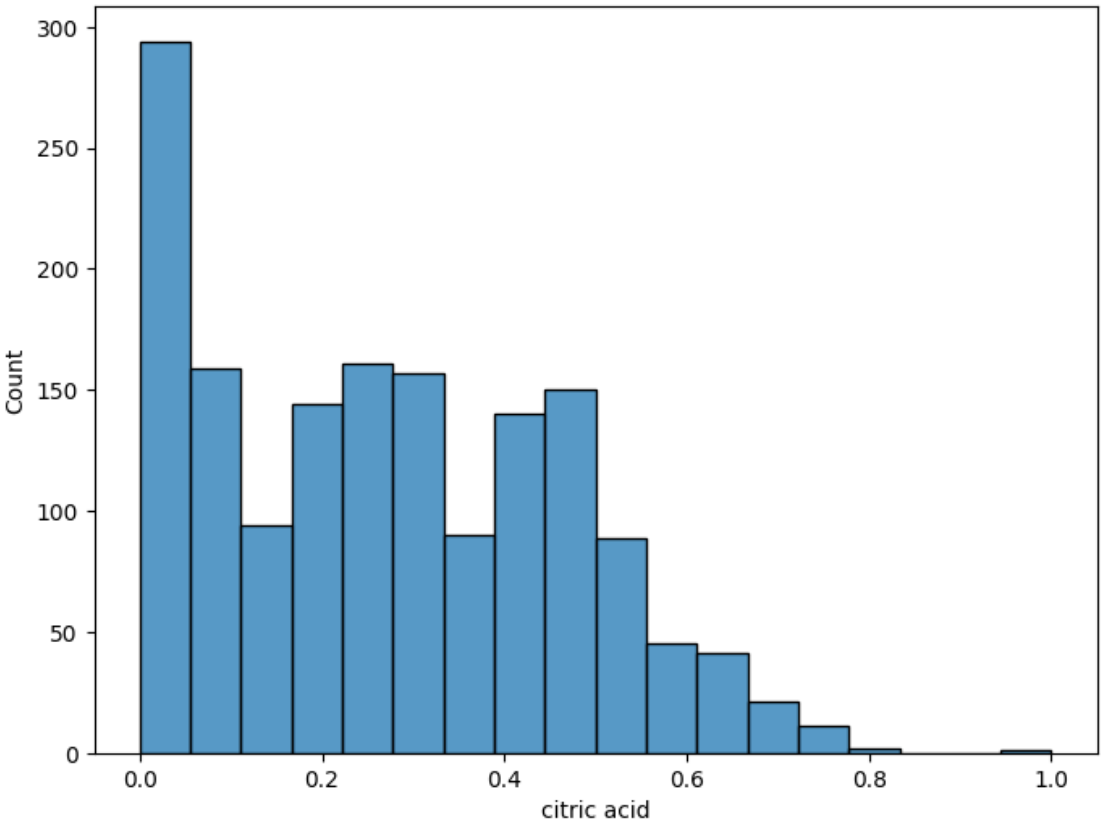
[ ]:
```
fixed acidity           0.982751
volatile acidity        0.671593
citric acid             0.318337
residual sugar          4.540655
chlorides               5.680347
free sulfur dioxide     1.250567
total sulfur dioxide    1.515531
density                 0.071288
pH                      0.193683
sulphates               2.428672
alcohol                 0.860829
quality                 0.217802
```

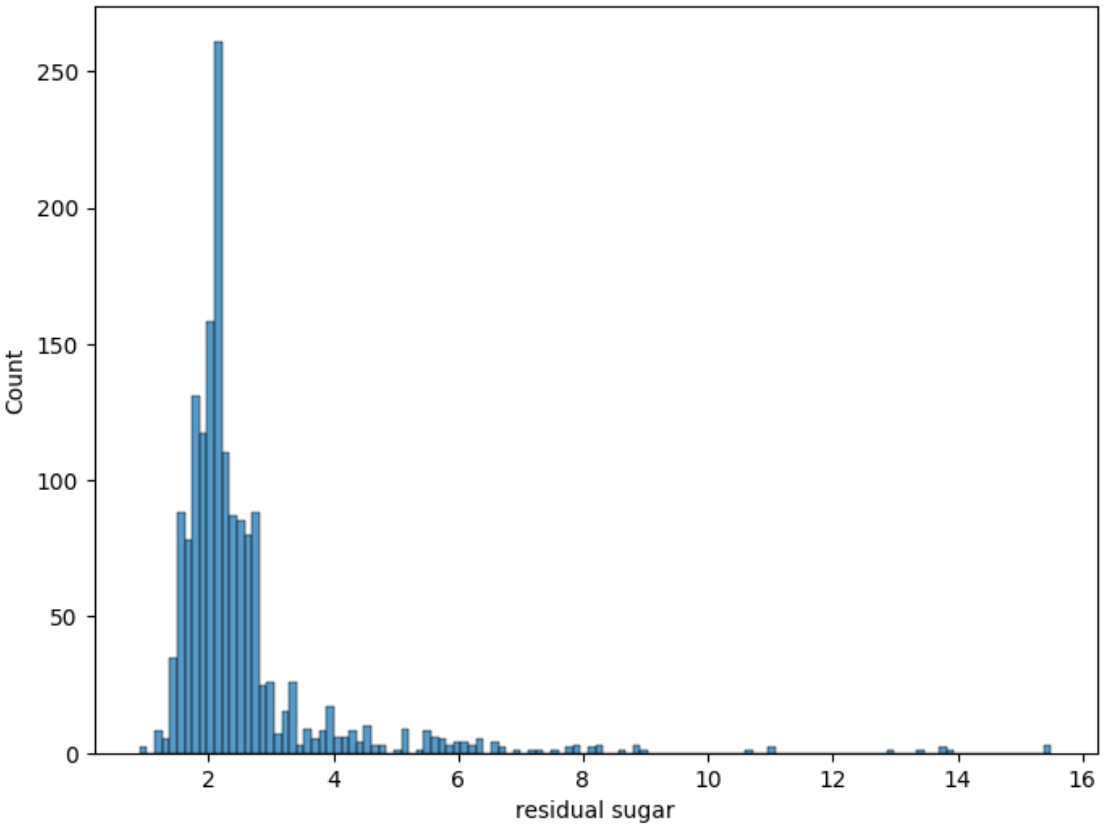
```
dtype: float64
```

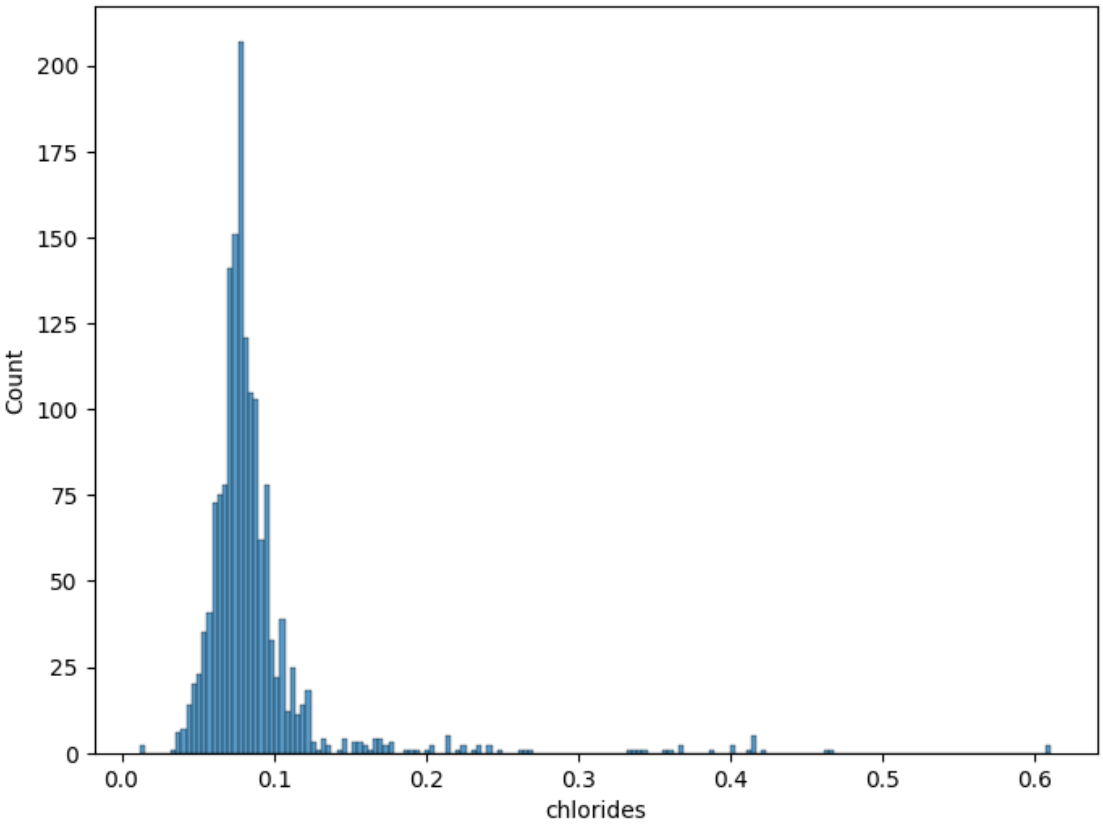
[ ]:
```
# Extracting the list of column names from the DataFrame 'df'
cols = list(df.columns)

# Create a boxplot for every numeric feature (cols 0-11) to show outliers
for col in cols:
  # Creating a new figure with a specified size for each histogram
  plt.figure(figsize = (8, 6))

  # Creating a histogram for the current column using seaborn's histplot
  sns.histplot(df[col])

  # Displaying the histogram plot
  plt.show()
```

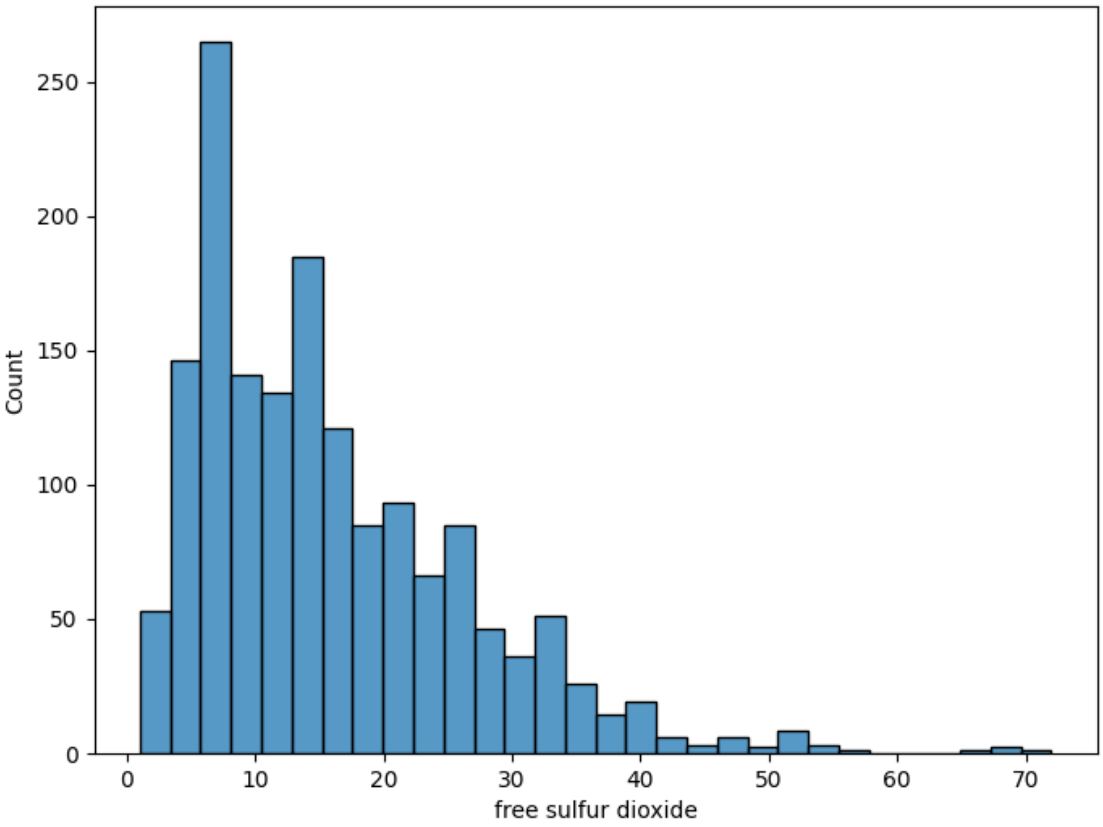


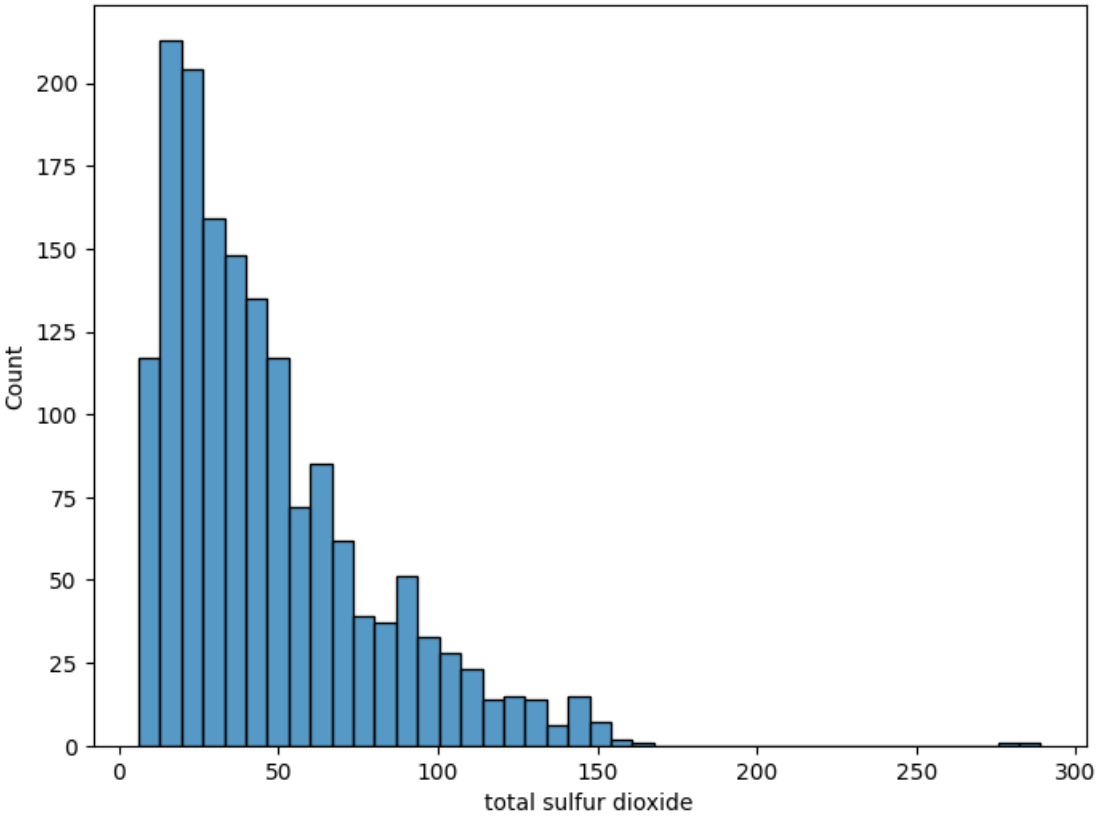

Count
175
150
125
100
75
50
25
0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
volatile acidity

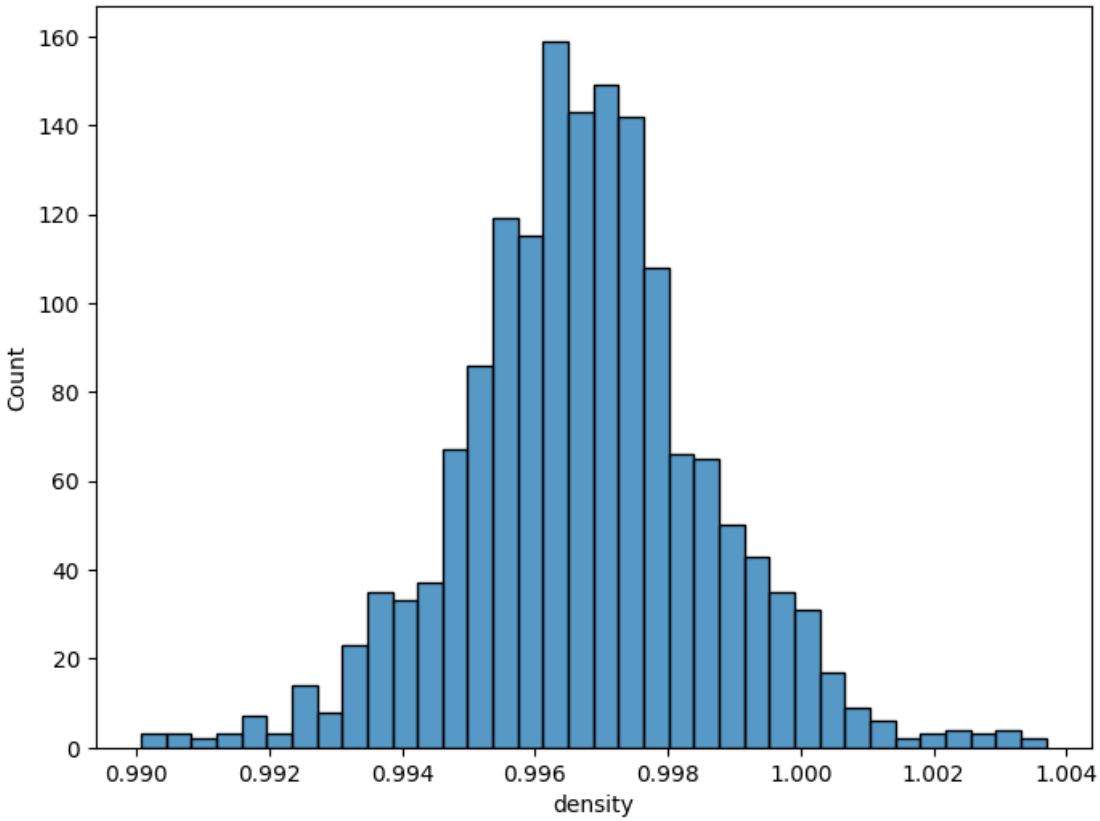

300
250
200
150
100
50
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
citric acid

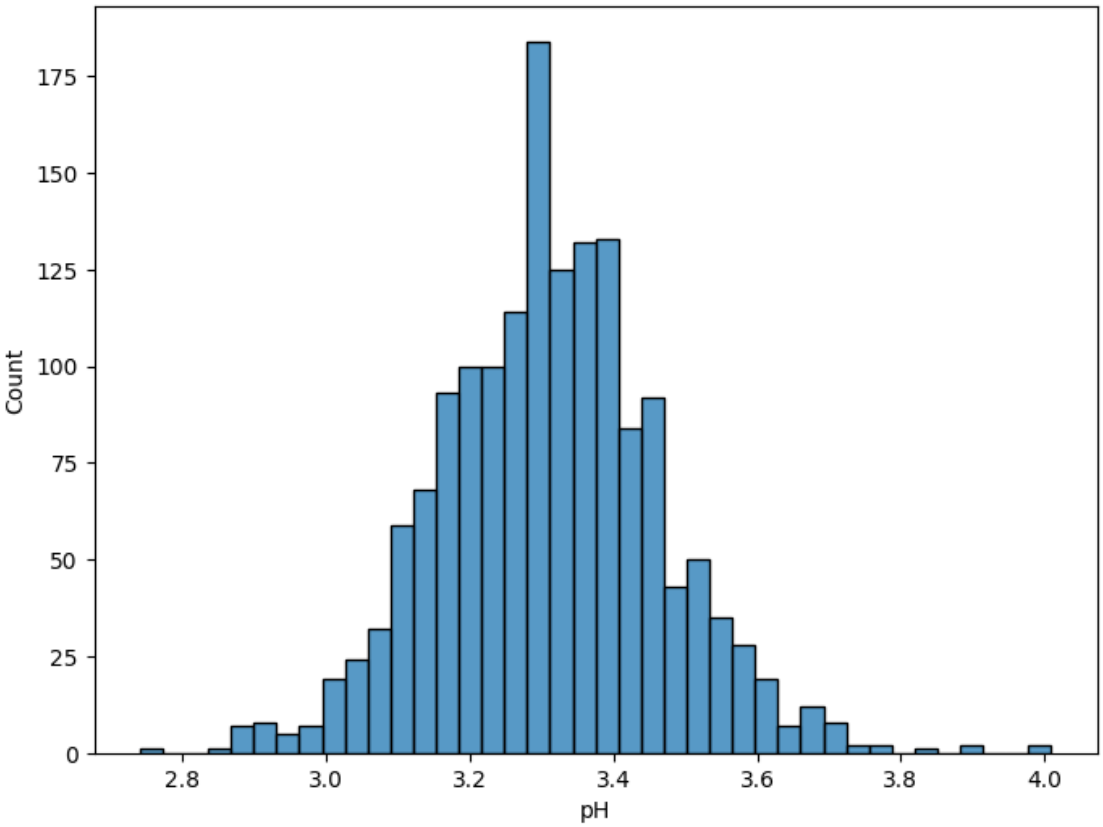

Count
250
200
150
100
50
0
2
4
6
8
10
12
14
16
residual sugar

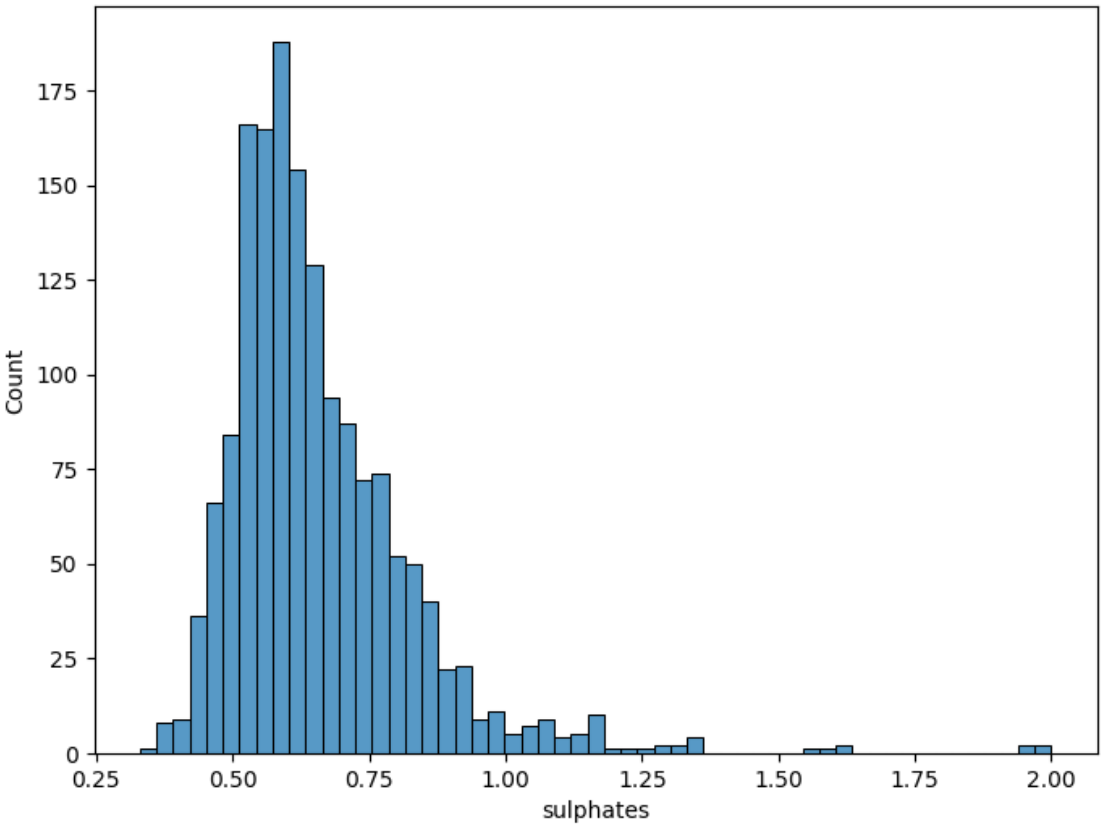

Count
200
175
150
125
100
75
50
25
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
chlorides

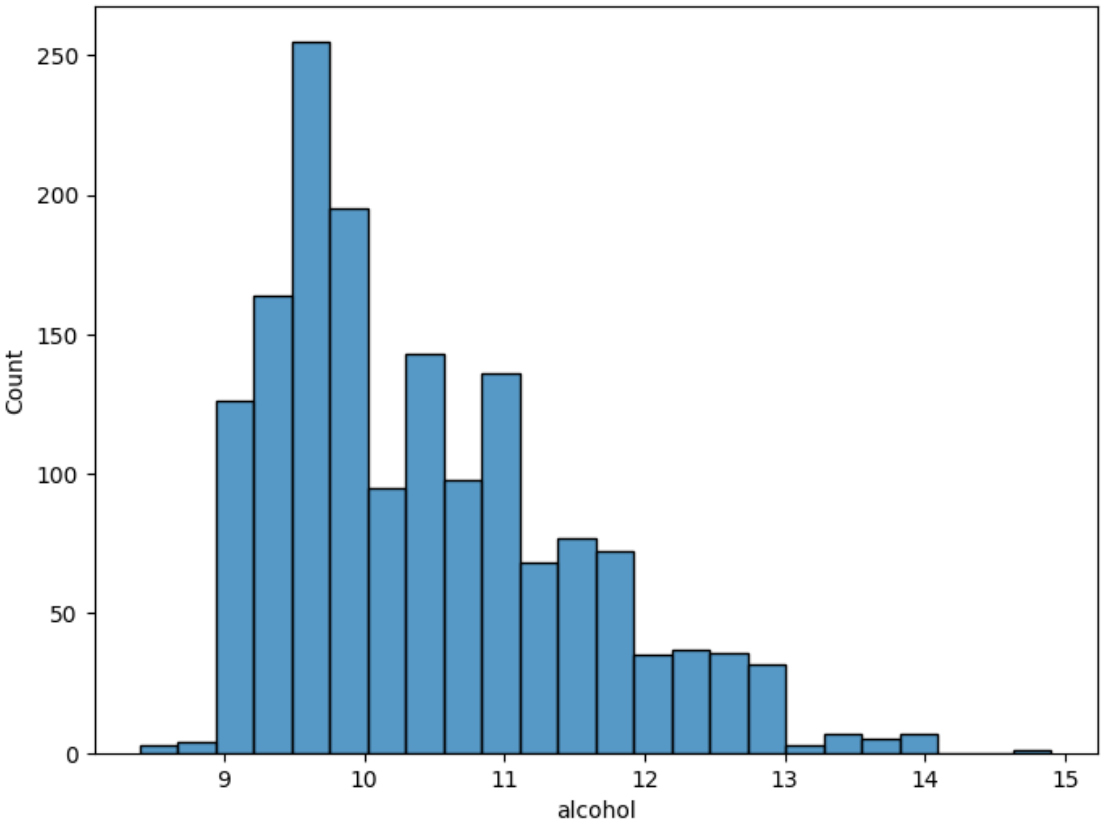

Count
250
200
150
100
50
0
0
10
20
30
40
50
60
70
free sulfur dioxide

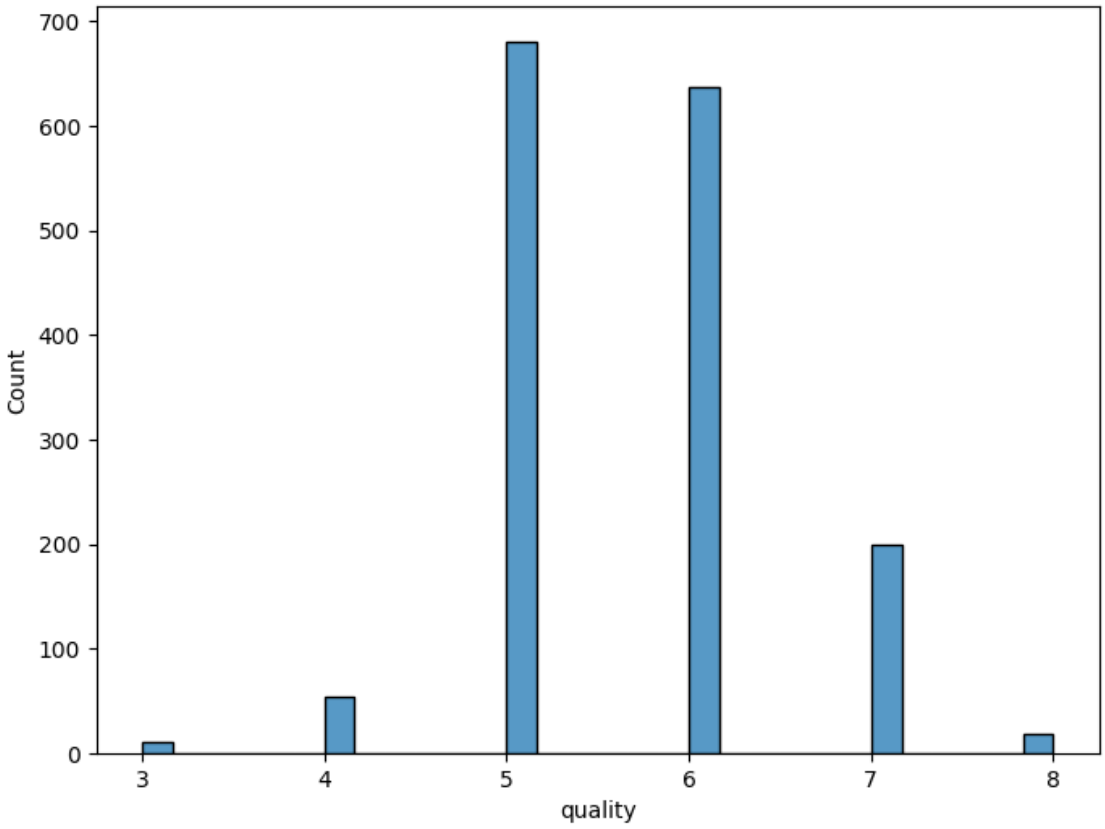

Count
200
175
150
125
100
75
50
25
0
0
50
100
150
200
250
300
total sulfur dioxide


Count
density
0
20
40
60
80
100
120
140
160
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004


Count
175
150
125
100
75
50
25
0
2.8
3.0
3.2
3.4
3.6
3.8
4.0
pH


Count
175
150
125
100
75
50
25
0
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
sulphates


Count
250
200
150
100
50
0
9
10
11
12
13
14
15
alcohol



1.12.1 Analysis of Distributions for Clustering:

The focus is the examination of the distribution of features within the dataset. Clustering, as an unsupervised learning technique, does not rely on labeled outcomes but rather on the natural groupings that emerge from the data's inherent characteristics.

Observing the numeric predictors in the dataset, I notice significant disparities in their scales. This variance in scale can potentially influence the clustering process, as certain features with larger ranges might dominate the clustering algorithm, thereby skewing the results. To counter this, standardizing or normalizing these features is essential. This ensures each feature contributes equally to the algorithm, allowing for a more accurate and meaningful clustering based on the true similarities in the data, rather than differences in scale.

Furthermore, the distribution patterns of these predictors are quite varied. Some align closely with a normal distribution, while others, such as alcohol content, residual sugar, and citric acid levels, deviate significantly from normality. In clustering, such skewed distributions can impact the formation of clusters. For instance, heavily skewed features might lead to clusters that are more reflective of outliers or extreme values rather than representing meaningful commonalities across data points.

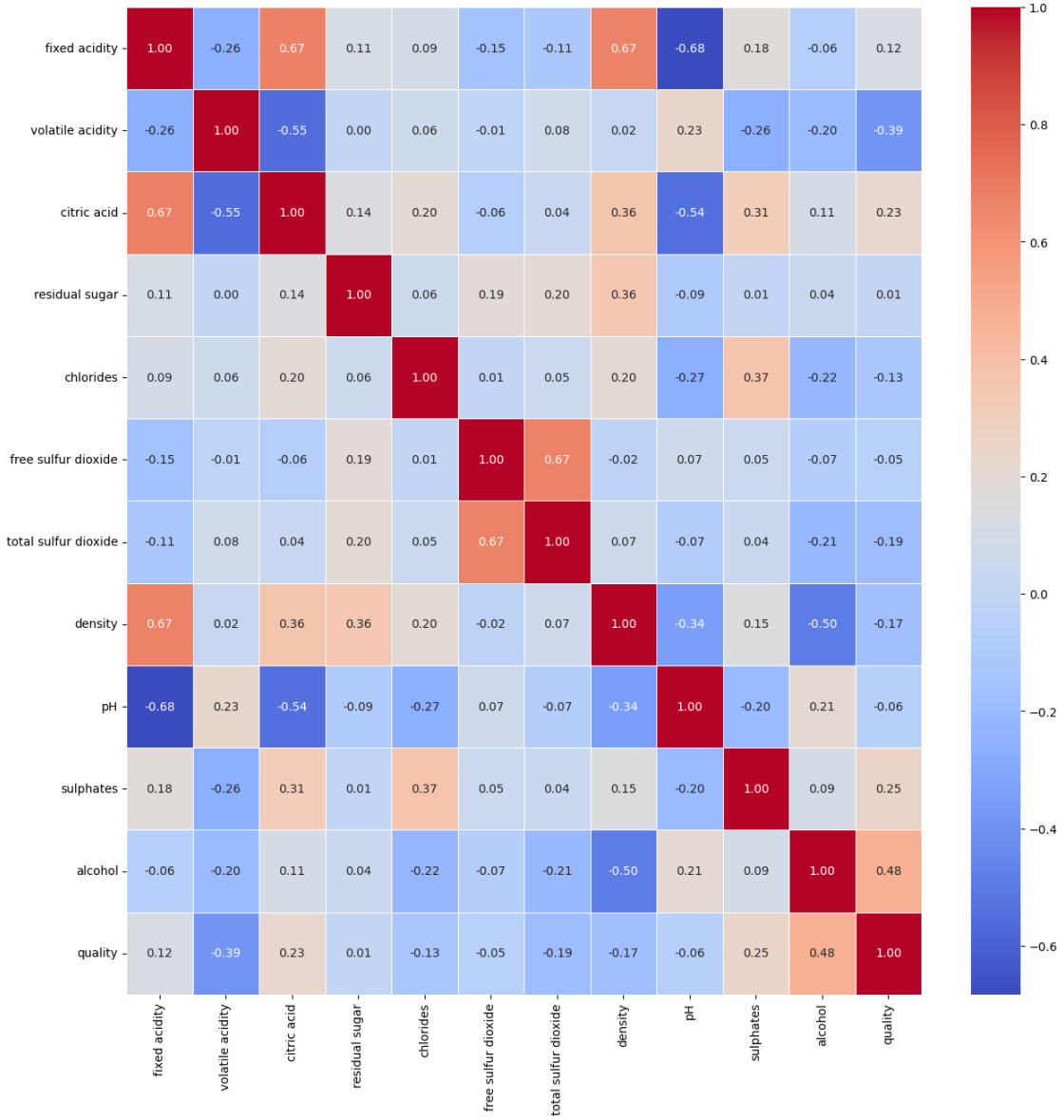
To address these issues, I plan to apply appropriate data transformations to achieve distributions that are more normal-like. Techniques like logarithmic transformation can be particularly effective

for skewed data. By transforming the data, the aim is to uncover more natural clusters in the dataset, enhancing the overall quality and interpretability of the clustering results.

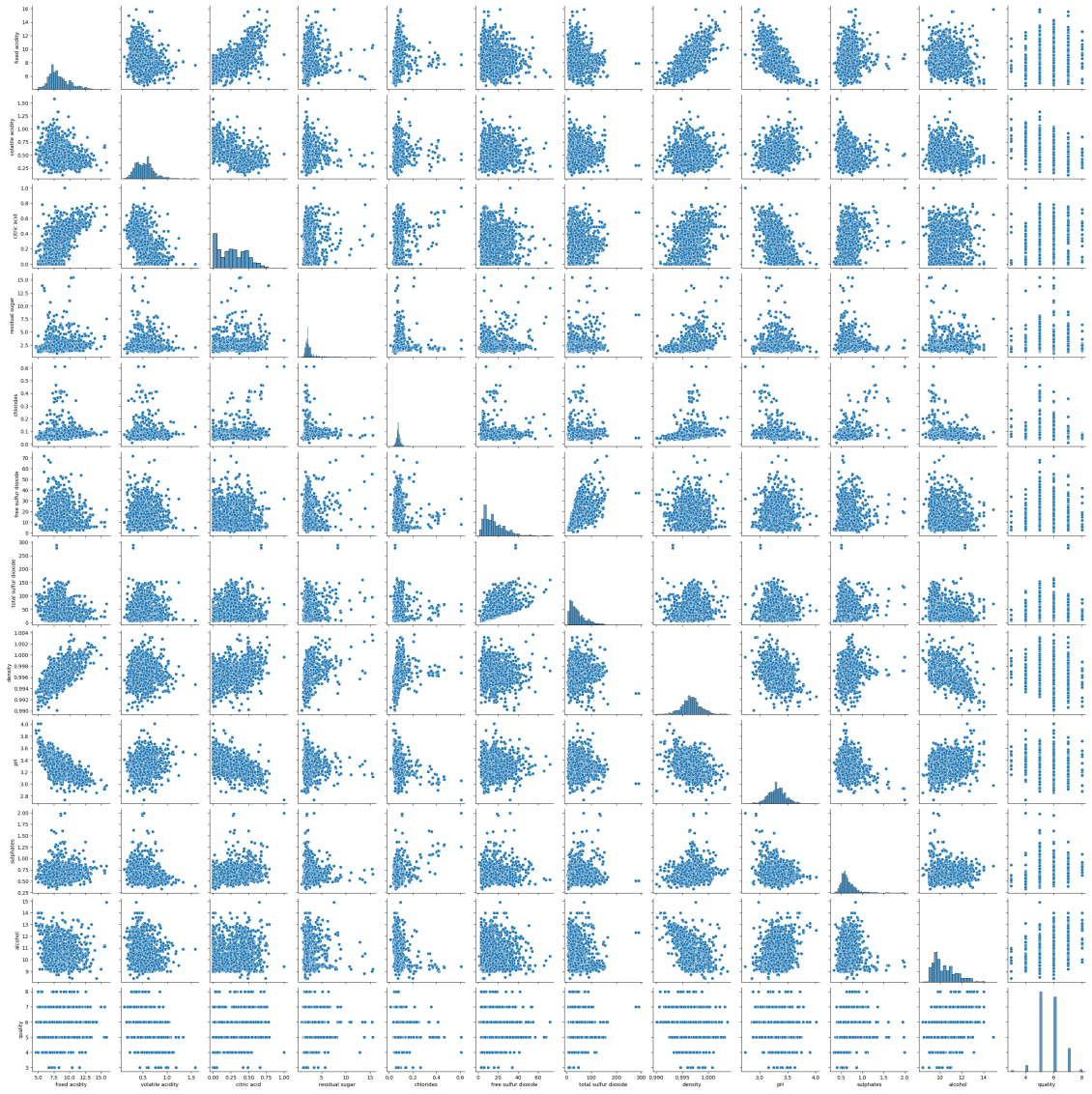
The goal of this analysis is to identify intrinsic clusters within the dataset based on the physicochemical properties of the wines. This could yield insights into different wine types and their characteristics, providing a basis for further exploration in wine categorization, customer preference analysis, or inventory management. By understanding the natural groupings in the data, I can uncover hidden patterns and relationships that might not be immediately apparent, offering a deeper understanding of the dataset's structure.

1.13 Correlations

[ ]:
```
# Create a Heatmap

# Setting the size of the plot
plt.figure(figsize = (15, 15))

# Creating a heatmap to visualize the correlation matrix of the DataFrame 'df'
# 'corr(numeric_only=True)' computes the correlation between numeric columns␣
 ↪only
# 'annot=True' displays the correlation coefficients in the heatmap
# 'fmt=".2f"' formats the correlation coefficients to two decimal places
# 'linewidths=0.7' sets the width of the lines that will divide each cell in␣
 ↪the heatmap
# 'cmap="coolwarm"' sets the color scheme of the heatmap to 'coolwarm',
# which is a diverging colormap (from cool to warm colors)
sns.heatmap(df.corr(numeric_only = True), annot = True, fmt = ".2f", linewidths␣
 ↪= 0.7, cmap = "coolwarm")

# Displaying the heatmap
plt.show() # Renders the heatmap in the output
```



[ ]:
```
# Create a Pairplot

# A pairplot displays pairwise relationships in a dataset
# By default, this plot will show scatter plots for each pair of variables and␣
 ↪histograms on the diagonal
sns.pairplot(df)

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



1.13.1 Analysis of Correlation Patterns for Clustering:

In examining the dataset, I have identified several significant correlation trends among the various features, which are key to understanding how different characteristics of wine are interrelated. Notably, there are moderate to strong negative correlations observed, such as between alcohol content and wine density, as well as between pH level and fixed acidity. These negative correlations suggest an inverse relationship, where an increase in one variable is often associated with a decrease in the other.

Furthermore, a strong positive correlation is evident between the density of the wine and its residual sugar content. This indicates that these two variables typically increase or decrease together. For instance, wines with higher residual sugar levels tend to have a higher density, and vice versa.

In the context of clustering, understanding these correlation patterns is crucial. While multi-

collinearity is a concern primarily in regression or classification models, in clustering, high correlation between features can lead to redundancy. Redundant features might overly influence the formation of clusters, potentially skewing the results. To address this, I will consider implementing feature selection or transformation techniques in my clustering analysis. These methods can help to minimize the redundancy and ensure that each feature contributes uniquely to the identification of clusters, thereby enhancing the integrity and interpretability of the clustering results.

The aim here is to utilize these correlation insights to better understand the natural groupings within the wine dataset. By accounting for how different physicochemical properties are interconnected, I can ensure that the clusters identified are meaningful and reflective of genuine similarities and differences in wine characteristics. This analysis will be instrumental in uncovering the underlying structure of the dataset, leading to a deeper understanding of wine properties and their relationships.

1.14 Dataset Splitting Strategy for Clustering:

In this project, I've devised a thorough strategy for splitting the dataset to maintain the quality and validity of my analysis. Initially, I set aside 10% of the dataset as a holdout set. This portion will not be used during the feature engineering or model training phases and is kept aside to evaluate the final clustering model. Ensuring this data remains untouched during the initial stages is crucial for an unbiased assessment of the model's effectiveness in real-world applications.

To avoid data leakage and ensure a fair evaluation, this 10% holdout set is segregated right from the start, before any feature engineering is conducted. I use a random shuffling method to partition this subset, guaranteeing that it is a representative sample of the overall dataset.

The remaining 90% of the dataset is then earmarked for the main analysis phase. This larger portion will undergo feature engineering to optimize the data for clustering. After enhancing and refining the dataset, I further divide it into two segments: one for training and the other for validation.

In the context of clustering, the concept of stratified sampling based on label classes, as is typical in classification tasks, is not applicable. Instead, the focus is on ensuring that the training and validation sets are comprehensive and reflective of the overall dataset's diversity. This consideration is crucial for developing a robust clustering model. The training set is used to identify patterns and form clusters, while the validation set helps in evaluating the stability and consistency of these clusters.

By adopting this strategy, I aim to develop a clustering model that is not only well-adjusted to the nuances of the dataset but also capable of generalizing its findings effectively. This careful approach to dataset splitting is intended to yield clusters that are both meaningful and indicative of the underlying structure in the wine data, offering valuable insights into different wine characteristics based on their physicochemical properties.

[ ]:
```
# Splitting the DataFrame into two parts: 90% for modeling and 10% for unseen␣
↪predictions

# Randomly sampling 90% of the data from 'df' for modeling purposes
# 'frac=0.90' specifies the fraction of data to sample (90%)
# 'random_state=42' ensures that the random sampling is reproducible
data = df.sample(frac = 0.90, random_state = random_state)
```

```
# Creating a separate dataset for unseen data (the remaining 10%)
# This is done by dropping the indices of 'data' from 'df', leaving the␣
 ↪unsampled 10%
data_unseen = df.drop(data.index)

# Resetting the index of the 'data' DataFrame
# 'inplace=True' modifies the DataFrame in place
# 'drop=True' prevents the old index from being added as a new column in the␣
 ↪DataFrame
data.reset_index(inplace = True, drop = True)

# Resetting the index of the 'data_unseen' DataFrame in the same manner
data_unseen.reset_index(inplace = True, drop = True)

# Printing the shape (number of rows and columns) of the 'data' DataFrame
# This shows the size of the dataset that will be used for modeling
print('Data for Modeling: ' + str(data.shape))

# Printing the shape of the 'data_unseen' DataFrame
# This indicates the size of the dataset set aside for final model testing
print('Unseen Data For Predictions: ' + str(data_unseen.shape))
```

```
Data for Modeling: (1439, 12)
Unseen Data For Predictions: (160, 12)
```

[ ]:
```
# Display the first 5 rows of the 'data' DataFrame
data.head()
```

[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.7	0.56	0.08	2.50	0.114	14.0	46.0	0.9971	3.24	0.66	9.6	6
1	7.8	0.50	0.17	1.60	0.082	21.0	102.0	0.9960	3.39	0.48	9.5	5
2	10.7	0.67	0.22	2.70	0.107	17.0	34.0	1.0004	3.28	0.98	9.9	6
3	8.5	0.46	0.31	2.25	0.078	32.0	58.0	0.9980	3.33	0.54		
4	6.7	0.46	0.24	1.70	0.077	18.0	34.0	0.9948	3.39	0.60		

```
3      9.8        5
4     10.6        6
```

```
[ ]: # Display the first 5 rows of the 'data_unseen' DataFrame
data_unseen.head()
```

```
[ ]:    fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.8              0.61         0.29             1.6      0.114
1            8.9              0.62         0.18             3.8      0.176
2            8.9              0.22         0.48             1.8      0.077
3            7.6              0.39         0.31             2.3      0.082
4            5.2              0.32         0.25             1.8      0.103

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                  9.0                  29.0   0.9974  3.26       1.56
1                 52.0                 145.0   0.9986  3.16       0.88
2                 29.0                  60.0   0.9968  3.39       0.53
3                 23.0                  71.0   0.9982  3.52       0.65
4                 13.0                  50.0   0.9957  3.38       0.55

   alcohol  quality
0      9.1        5
1      9.2        5
2      9.4        6
3      9.7        5
4      9.2        5
```

1.15 Data Cleaning Strategy:

My data cleaning approach is designed to refine and prepare the dataset for effective analysis. In this dataset, I encounter primarily numeric features, except for the target label, which is already one-hot encoded. This simplifies my process, as no additional processing is required for categorical variables. Another advantage is the absence of missing values in our data, eliminating the need for imputation strategies or the removal of rows due to incomplete data.

However, during the exploratory data analysis, I have identified two notable aspects that require attention. Firstly, the numeric features exhibit varying scales, which can potentially skew the model's interpretation and valuation of these features. Secondly, several numeric features are non-normally distributed, which could impact the model's performance. Both these issues will be addressed comprehensively in the experiment setup phase, ensuring that the data is optimally conditioned for modelling.

1.15.1 Outlier Analysis Approach:

My strategy for dealing with outliers is contingent on the dataset's characteristics and the extent of outlier presence. I consider two scenarios:

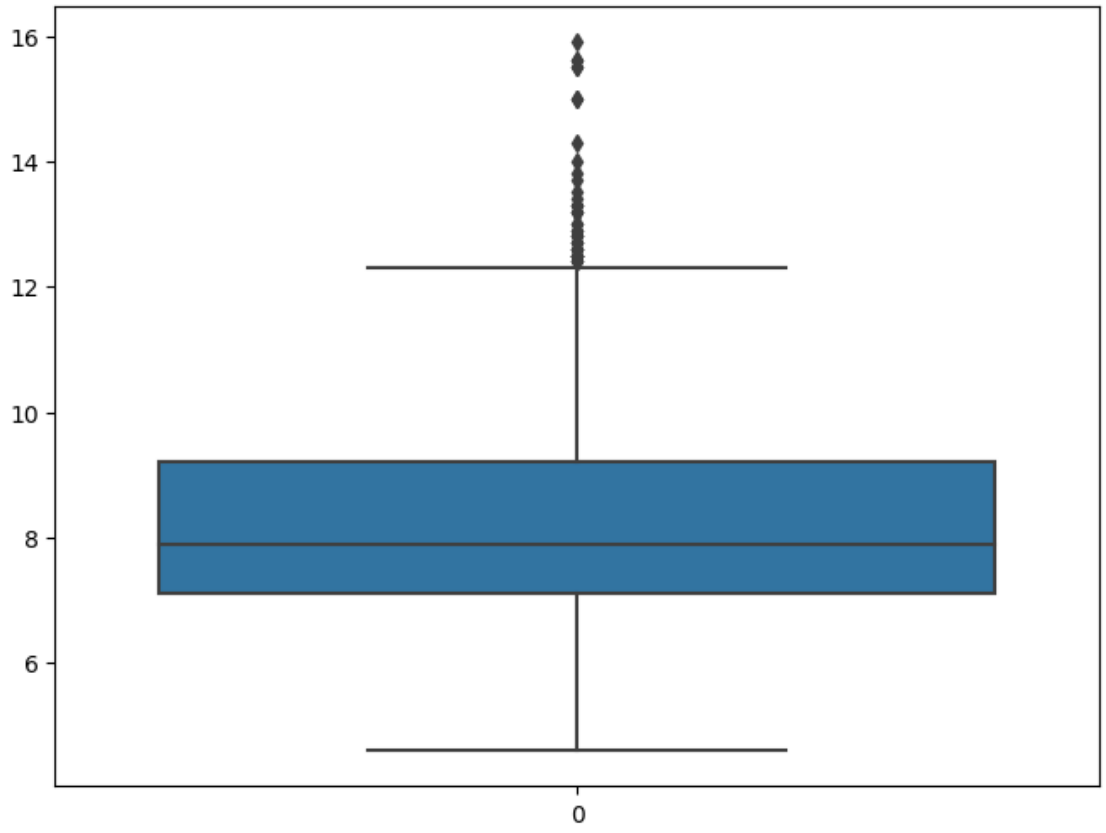
1. **Abundant Data with Sparse Outliers:** In cases where the dataset is sizeable but contains only a few rows with outlier values in any column, my approach leans towards removing these

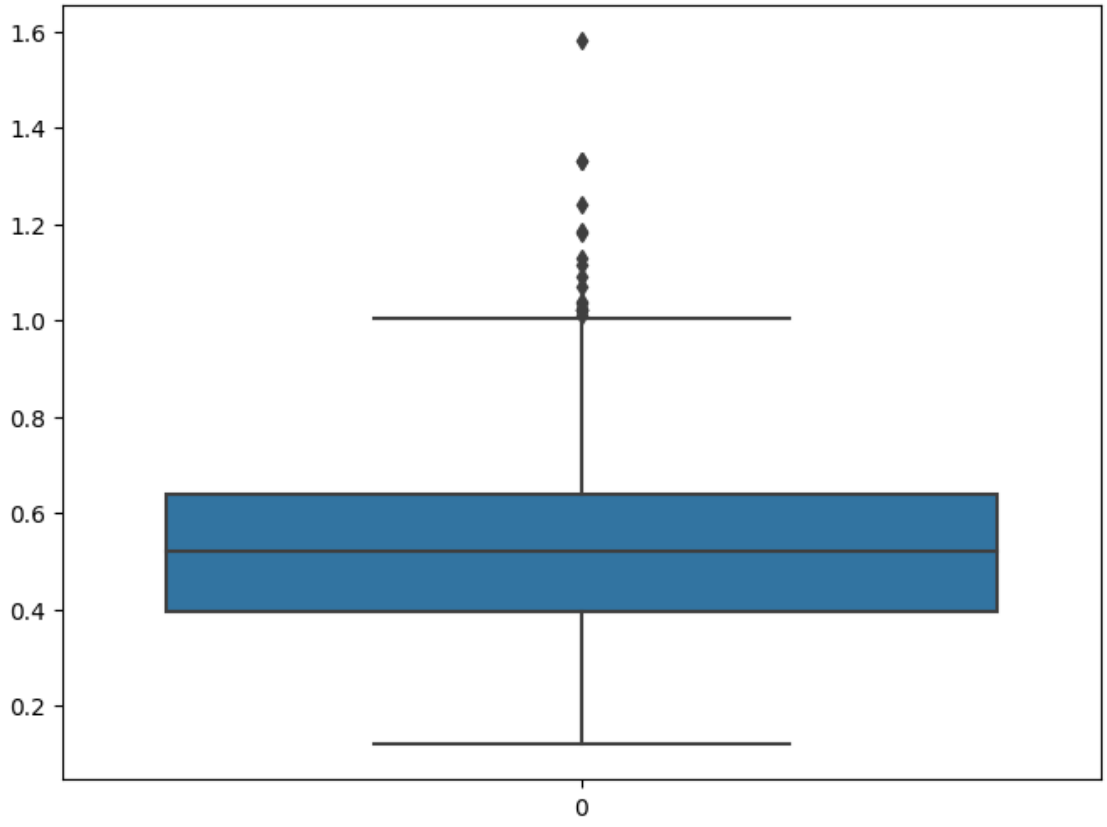
rows. This decision is based on the premise that the elimination of a small number of outliers will not significantly impact the dataset's integrity yet will enhance the model's accuracy.

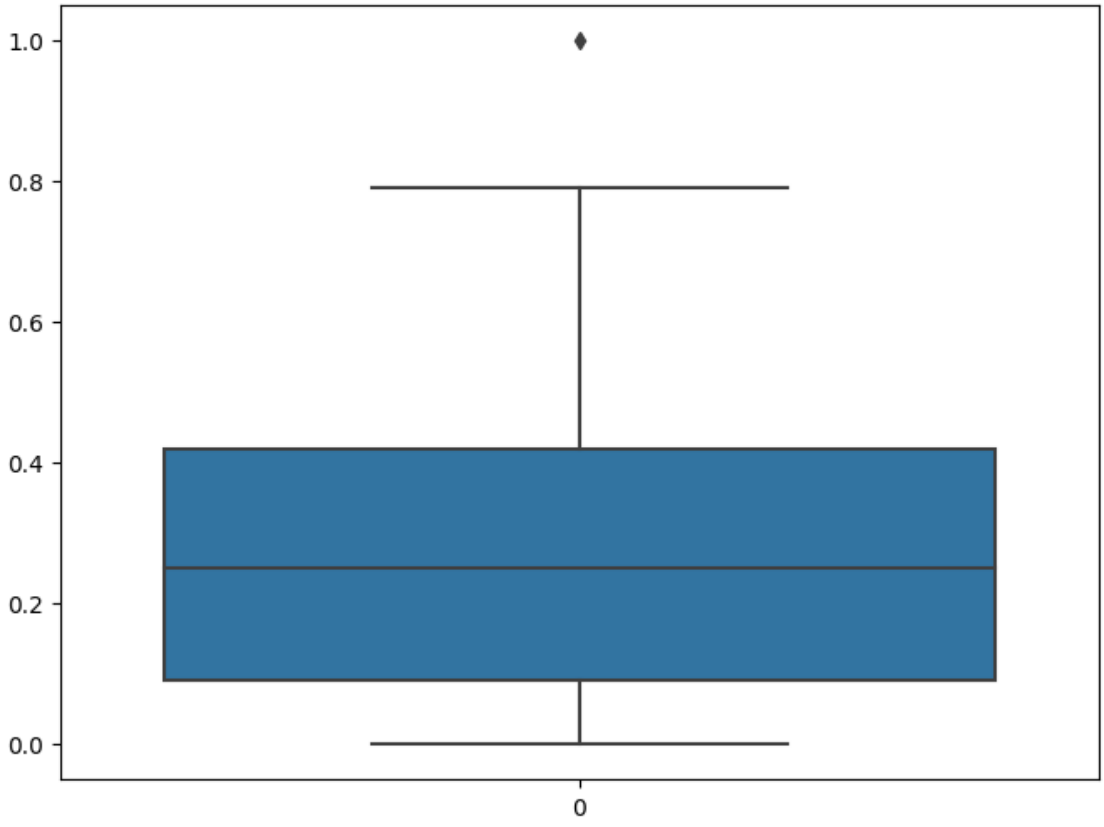
2. **Limited Data with Numerous Outliers:** Conversely, in situations where our dataset is relatively small and a larger proportion of rows exhibit outlier values, a more nuanced approach is required. Here, I opt to cap the values at the 5th and 95th percentiles. This method effectively reduces the impact of extreme outliers while preserving the bulk of the data, ensuring that our model is trained on a dataset that is representative of the broader population, albeit with moderated extremities.

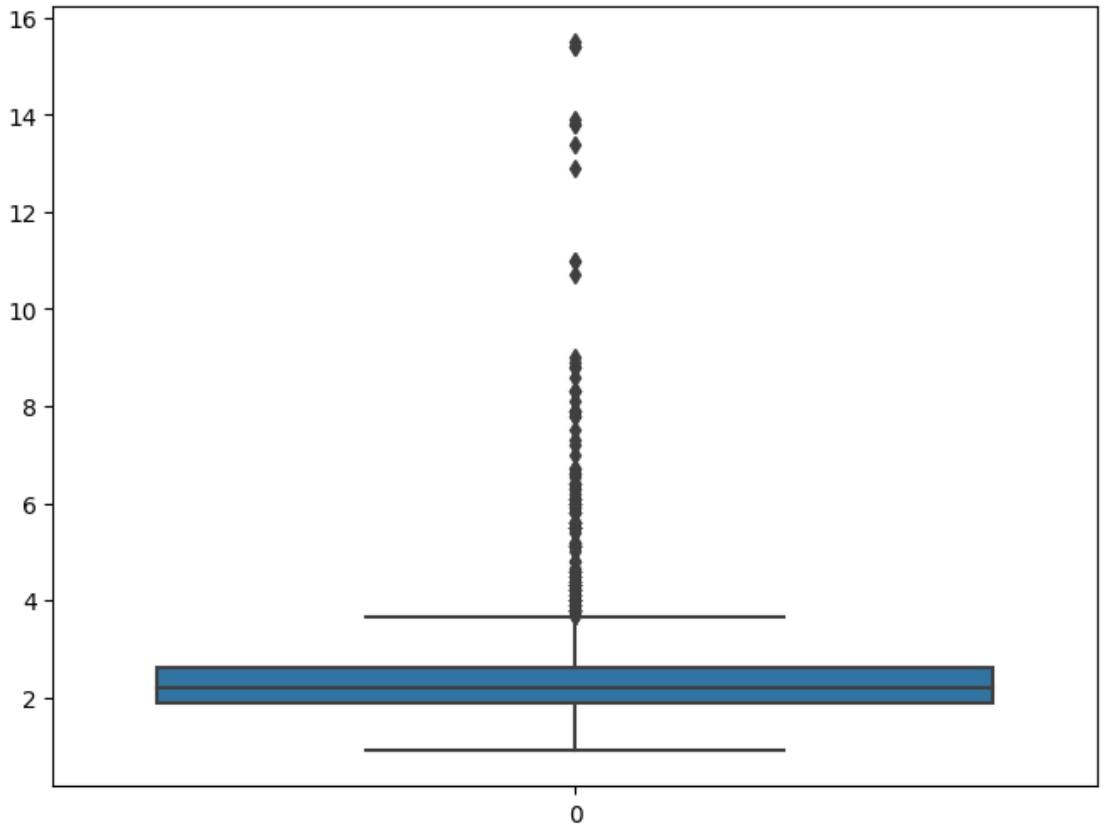
This two-pronged strategy for outlier management is designed to balance the need for a clean, representative dataset with the preservation of as much data as possible, thereby ensuring robustness in the subsequent modelling phase.

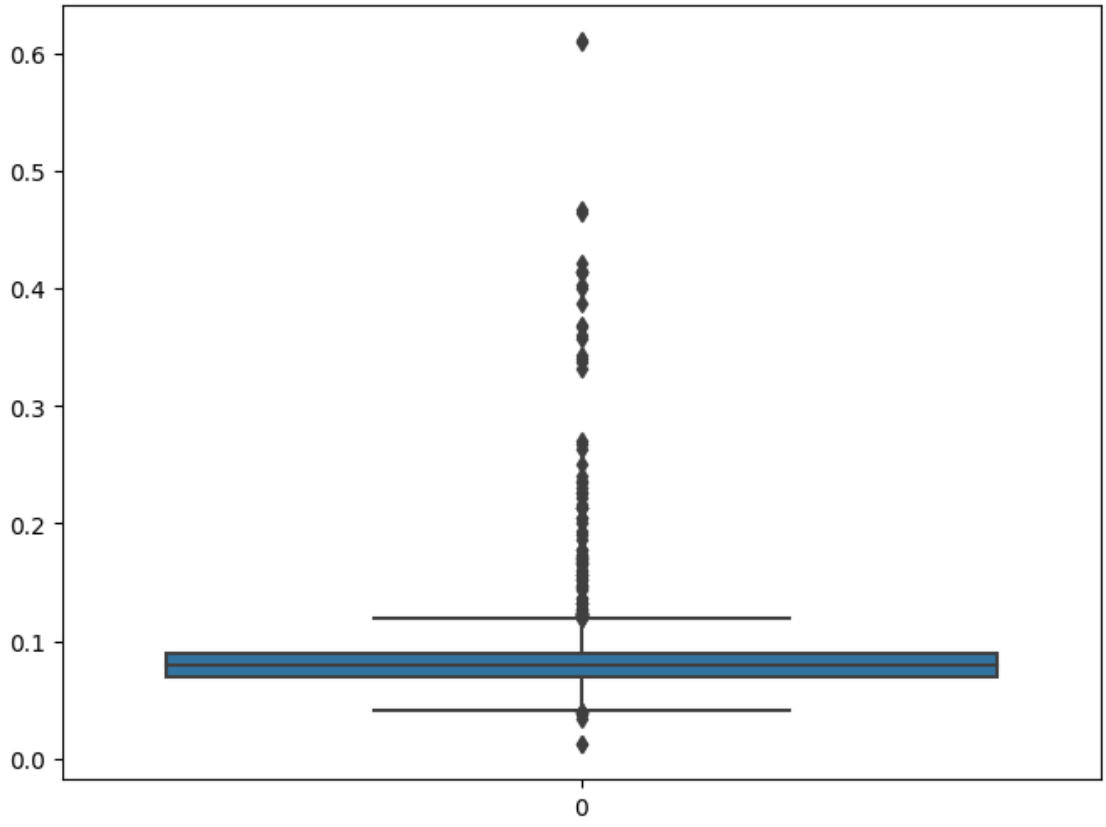
[ ]:
```
# Outlier analysis
cols = list(data.columns)

# Create boxplot for every numeric feature (cols 0-11) to show outliers
for col in cols[0:-1]:

  # Set the size of the plot
  plt.figure(figsize = (8, 6))

  # Create the boxplot for the current column
  sns.boxplot(data[col])

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

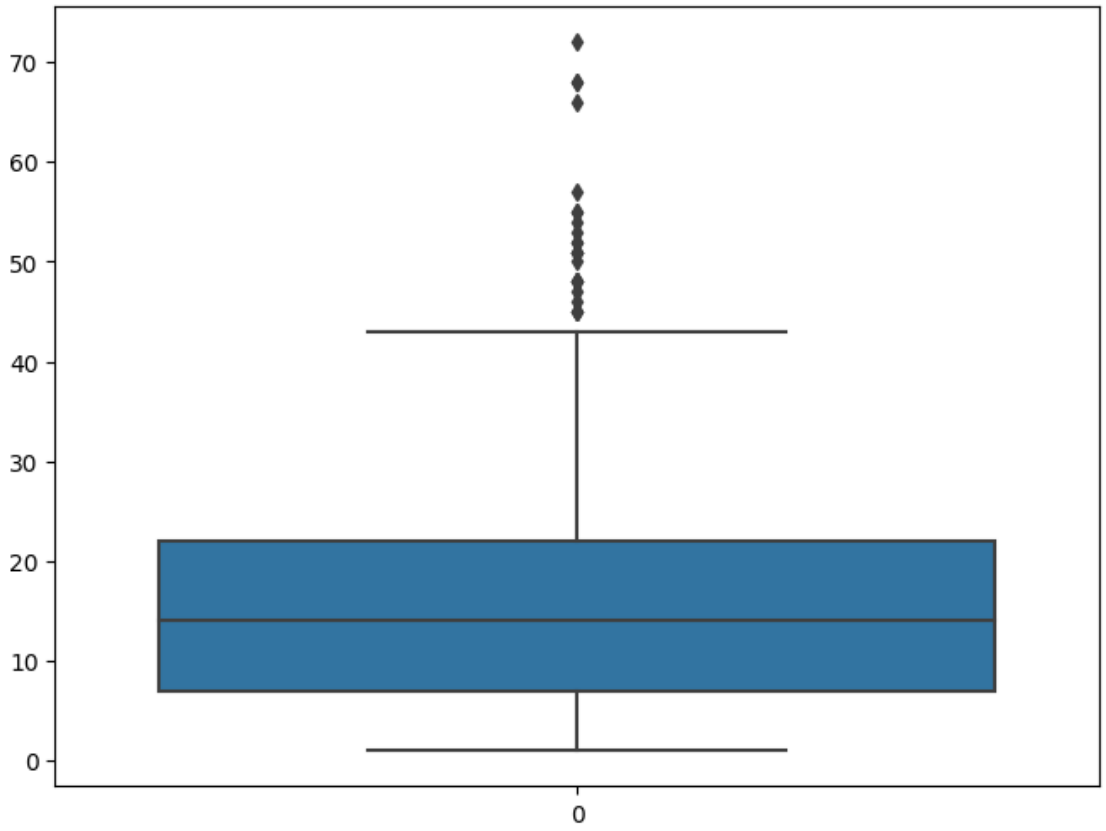

16
14
12
10
8
6
0

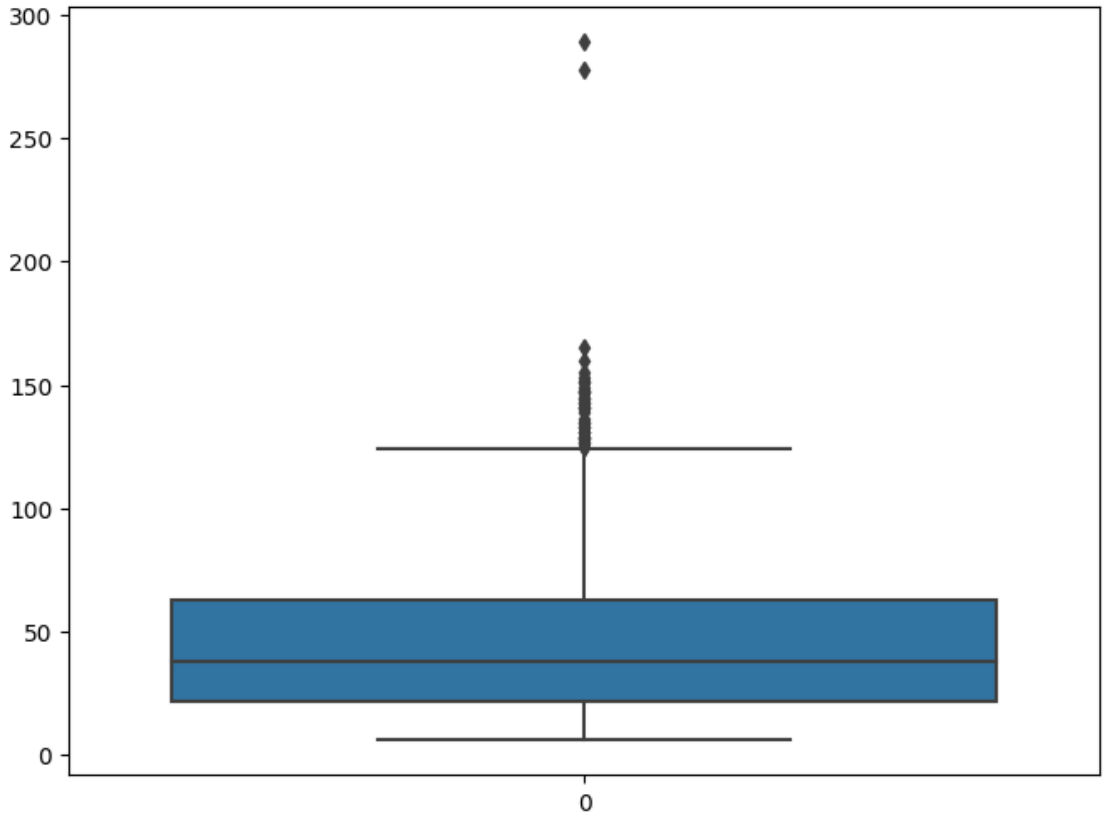

1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0

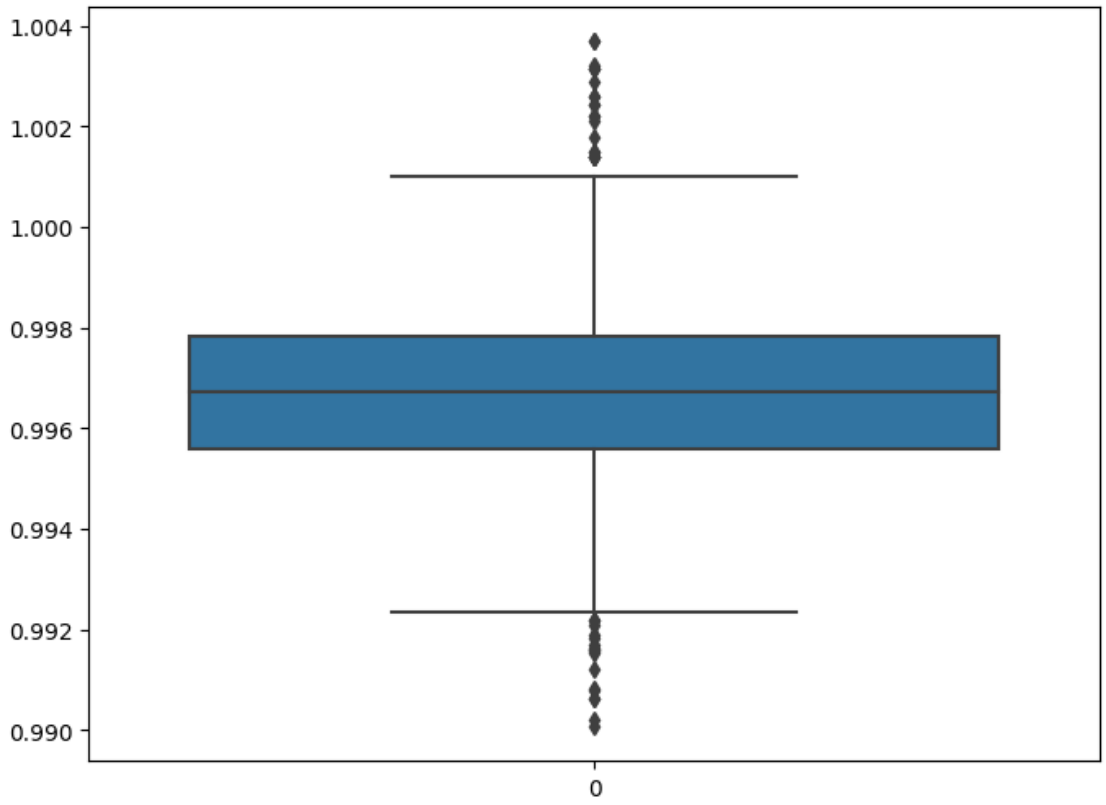

1.0
0.8
0.6
0.4
0.2
0.0
0

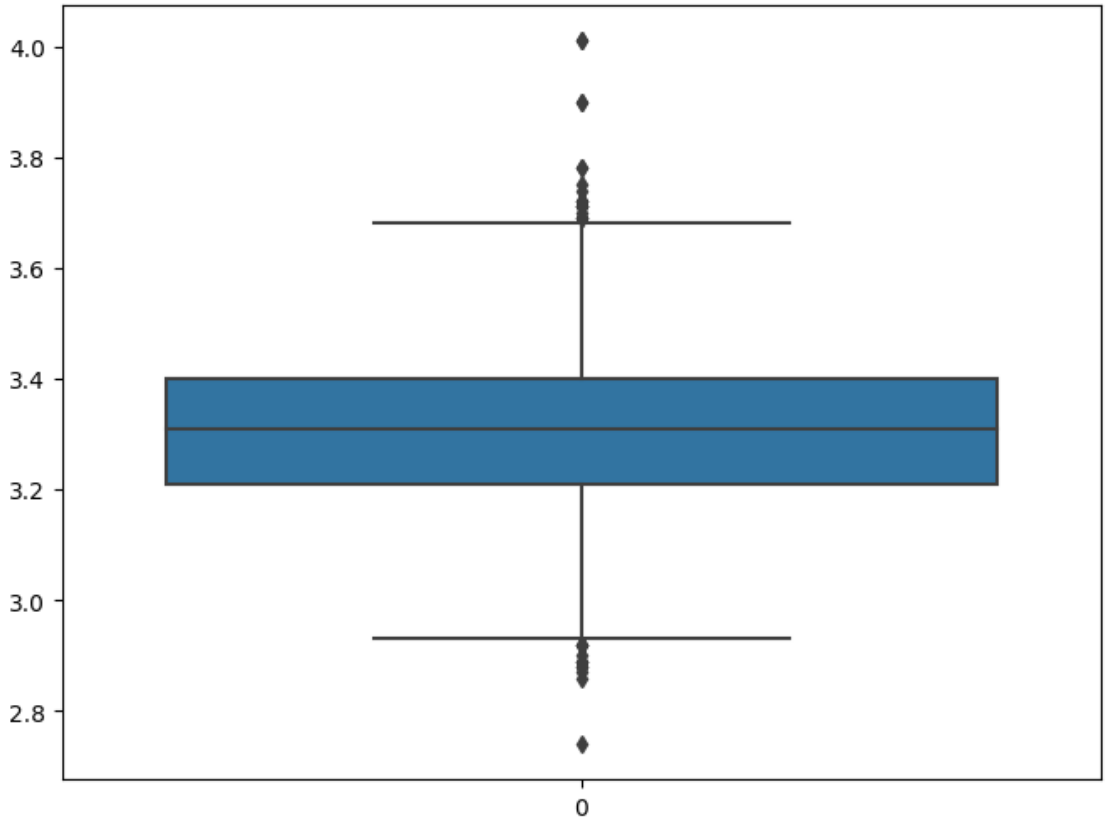

16
14
12
10
8
6
4
2
0

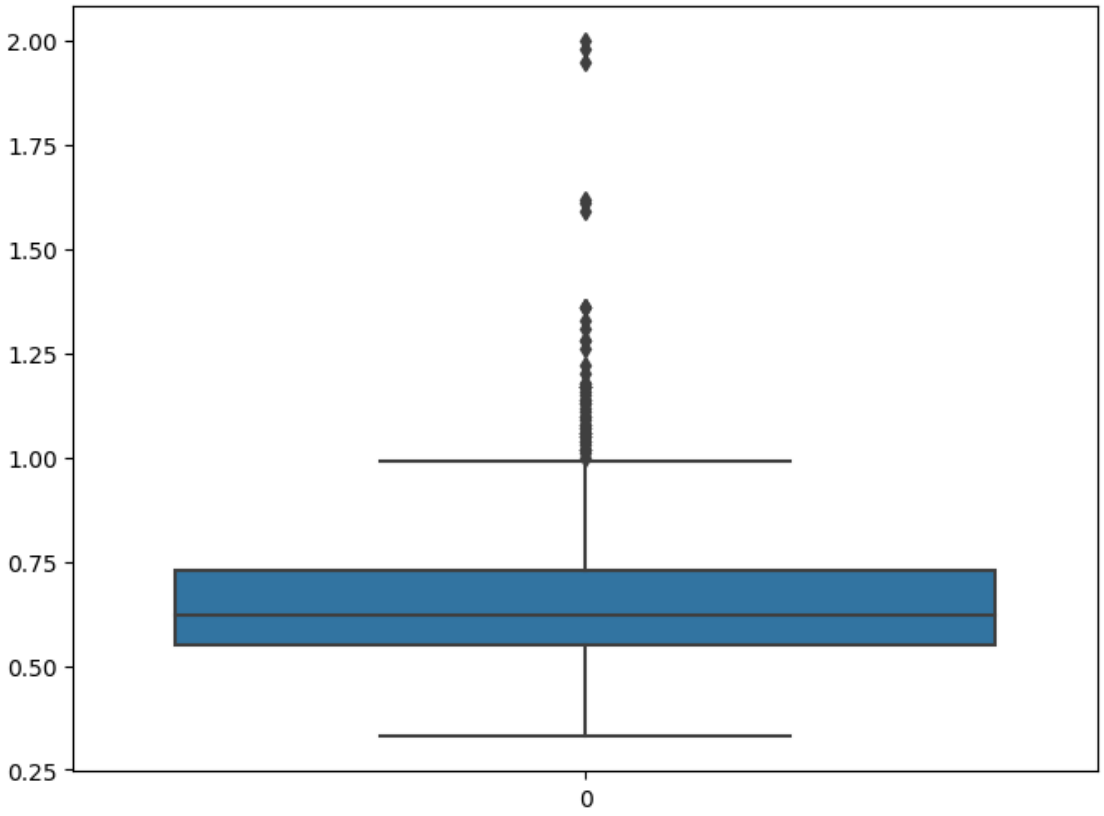

0.6
0.5
0.4
0.3
0.2
0.1
0.0
0

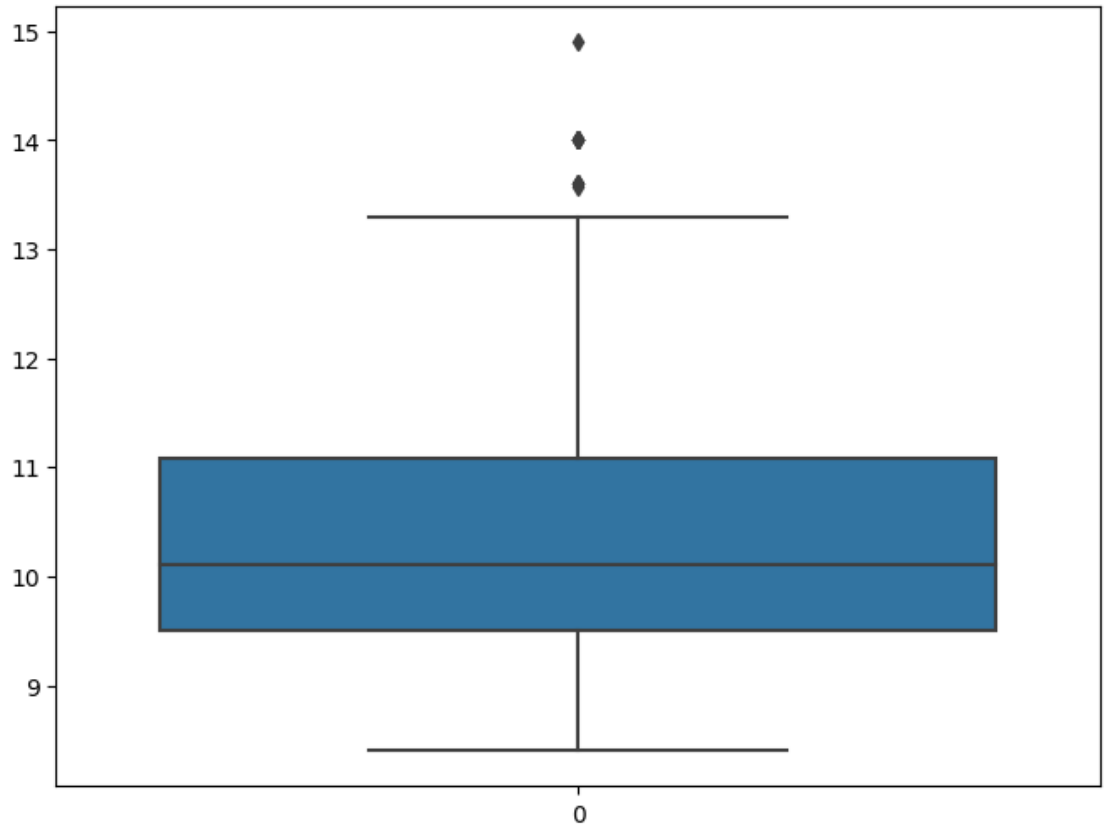

70
60
50
40
30
20
10
0
0


300
250
200
150
100
50
0
0


1.004
1.002
1.000
0.998
0.996
0.994
0.992
0.990
0


4.0
3.8
3.6
3.4
3.2
3.0
2.8
0


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0



1.15.2 Analysis of Outliers from Boxplots:

Upon reviewing the boxplots generated for each predictor variable, it becomes evident that a considerable number of these variables are characterized by the presence of outliers. Specifically, certain predictors, such as chlorides, exhibit a notably high frequency of outlier values. This abundance of outliers across various predictors presents a significant challenge in data preprocessing.

The sheer volume of these outliers poses a dilemma: if I choose to remove all rows containing outlier values, I risk significantly diminishing our dataset. Such a reduction could compromise the dataset's representativeness and the robustness of any subsequent analyses or model training. This is particularly critical given that an extensive removal of data points might lead to the loss of valuable information that could be crucial for understanding the underlying patterns and relationships within the dataset.

Considering this, my approach is to implement a capping strategy. I will cap the values at the 5th and 95th percentiles. This method involves adjusting extreme outlier values to these percentile thresholds, effectively reducing the impact of the most extreme cases while preserving most of the data. By doing so, I aim to maintain the integrity and the size of our dataset as much as possible. This approach strikes a balance between mitigating the influence of outliers and retaining a comprehensive set of data for analysis, ensuring that our models are trained on a dataset that is both clean and representative of the broader range of values.

[ ]:
```
# Creating a temporary copy of the 'data' DataFrame
# This is done to avoid accidentally modifying the original DataFrame
# The temporary copy allows for comparison and verification after data␣
 ↪processing
tmp = data

# Assigning the original 'data' DataFrame to a new variable 'data_clean'
# This step indicates the intention to perform data cleaning or preprocessing␣
 ↪on 'data_clean'
# It provides clarity and maintains the original data in 'tmp' for reference or␣
 ↪comparison
data_clean = data
```

[ ]:
```
# Generating and displaying descriptive statistics for the 'data' DataFrame
data.describe()
```

[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	\
count	1439.000000	1439.000000	1439.000000	1439.000000	
mean	8.301459	0.528311	0.268506	2.535302	
std	1.738644	0.178718	0.193119	1.443800	
min	4.600000	0.120000	0.000000	0.900000	
25%	7.100000	0.395000	0.090000	1.900000	
50%	7.900000	0.520000	0.250000	2.200000	
75%	9.200000	0.640000	0.420000	2.600000	
max	15.900000	1.580000	1.000000	15.500000	

	chlorides	free sulfur dioxide	total sulfur dioxide	density	\
count	1439.000000	1439.000000	1439.000000	1439.000000	
mean	0.087767	15.974635	46.640723	0.996748	
std	0.048272	10.495904	33.052438	0.001894	
min	0.012000	1.000000	6.000000	0.990070	
25%	0.070000	7.000000	22.000000	0.995600	
50%	0.079000	14.000000	38.000000	0.996720	
75%	0.090000	22.000000	63.000000	0.997825	
max	0.611000	72.000000	289.000000	1.003690	

	pH	sulphates	alcohol	quality
count	1439.000000	1439.000000	1439.000000	1439.000000
mean	3.311598	0.657596	10.398992	5.635858
std	0.154815	0.166836	1.054741	0.806437
min	2.740000	0.330000	8.400000	3.000000
25%	3.210000	0.550000	9.500000	5.000000
50%	3.310000	0.620000	10.100000	6.000000
75%	3.400000	0.730000	11.083333	6.000000
max	4.010000	2.000000	14.900000	8.000000

[ ]:
```
# Extracting a list of all column names from the DataFrame 'data'
cols = list(data.columns)

# Looping through each column (except the last one) to create a boxplot and cap␣
 ↪outliers
for col in cols:
    # Calculating the upper limit, which is approximately the 95th percentile
    # This is done by adding three standard deviations to the mean
    upper_limit = tmp[col].mean() + 3 * tmp[col].std()

    # Calculating the lower limit, which is approximately the 5th percentile
    # This is done by subtracting three standard deviations from the mean
    lower_limit = tmp[col].mean() - 3 * tmp[col].std()

    # Capping the values in 'data_clean' at the upper and lower limits
    # If a value in 'tmp' is greater than the upper limit, it's set to the␣
 ↪upper limit
    # If a value in 'tmp' is less than the lower limit, it's set to the lower␣
 ↪limit
    # Otherwise, the original value from 'tmp' is retained
    data_clean[col] = np.where(tmp[col] > upper_limit, upper_limit,  # If above␣
 ↪95th percentile, cap at upper limit
                               np.where(tmp[col] < lower_limit, lower_limit,  #␣
 ↪If below 5th percentile, cap at lower limit
                               tmp[col]))  # Retain original value if within␣
 ↪limits
```

[ ]:
```
# Capped distributions. Verify by checking max and min
data_clean.describe()
```

[ ]:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1439.000000       1439.000000  1439.000000     1439.000000
mean        8.290890          0.527196     0.268400        2.470628
std         1.700962          0.174393     0.192760        1.074141
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.395000     0.090000        1.900000
50%         7.900000          0.520000     0.250000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        13.517392          1.064467     0.847864        6.866703

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1439.000000          1439.000000           1439.000000  1439.000000
mean      0.085060            15.871263             46.395393     0.996746
std       0.031403            10.106061             31.886199     0.001873
min       0.012000             1.000000              6.000000     0.991065
25%       0.070000             7.000000             22.000000     0.995600
50%       0.079000            14.000000             38.000000     0.996720
```

75%	0.090000	22.000000	63.000000	0.997825
max	0.232582	47.462346	145.798037	1.002431

	pH	sulphates	alcohol	quality
count	1439.000000	1439.000000	1439.000000	1439.000000
mean	3.311170	0.653922	10.395860	5.637062
std	0.152727	0.148889	1.044248	0.802651
min	2.847152	0.330000	8.400000	3.216547
25%	3.210000	0.550000	9.500000	5.000000
50%	3.310000	0.620000	10.100000	6.000000
75%	3.400000	0.730000	11.083333	6.000000
max	3.776045	1.158102	13.563214	8.000000

1.15.3 Training Clustering Models

With the data now thoroughly cleaned, I'm ready to proceed with setting up my clustering experiment pipeline. In the final stages of data preparation, I tackle a few remaining issues to optimize the data for clustering. For instance, I observe a wide range of values across different features, such as the chlorides column varying from 0.009 to 0.346, while residual sugar spans from 0.6 to 65.8. Such disparities in scale can impact the effectiveness of clustering algorithms; hence, I decide to implement z-score normalization to bring all features onto a comparable scale, typically ranging from -3 to +3.

Next, I focus on transforming the features to approximate a Gaussian (normal) distribution. This is important for clustering because many algorithms, particularly those based on distance metrics, assume that the data follows a normal distribution. By transforming the data accordingly, I can enhance the algorithm's ability to identify meaningful clusters.

Additionally, I address the issue of multicollinearity in the dataset. In the context of clustering, while multicollinearity does not distort results as it does in regression models, it can lead to redundancy in information. I set a multicollinearity threshold at 0.7 and remove features that exceed this threshold. When two features are highly correlated, I choose to keep the feature that contributes more unique information to the dataset. This step is crucial to ensure that each feature adds distinct value to the clustering process.

Lastly, although addressing imbalances in the target variable, as done through techniques like SMOTE, is a strategy employed in classification tasks, it's not typically applicable in clustering. In clustering, the focus is on discovering natural groupings in the data without any predefined labels. Therefore, the concept of balancing a target variable does not apply.

By meticulously preparing the data through these steps, I aim to develop a robust clustering model. The goal is to uncover inherent groupings within the wine dataset based on their physicochemical properties, revealing insights into different wine types and their unique characteristics. This preparation is key to ensuring that the clusters formed are meaningful and reflective of the actual structures within the data.

1.15.4 Normalization

[ ]:
```
# Normalize features for effective clustering
scaler = StandardScaler()
df_scaled = scaler.fit_transform(data_clean)

# Apply PowerTransformer to make the data more Gaussian-like
power_transformer = PowerTransformer()
df_transformed = power_transformer.fit_transform(df_scaled)
```

[ ]:
```
# Convert the transformed data into a DataFrame
transformed_df = pd.DataFrame(df_transformed, columns = data.columns)
```

[ ]:
```
# Display the first five rows of the transformed DataFrame
transformed_df.head()
```

[ ]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0      -0.206842          0.275365    -0.995695        0.511586   1.194606
1      -0.134258         -0.072441    -0.454919       -1.456167   0.173111
2       1.341025          0.854309    -0.171255        0.743562   1.026651
3       0.321152         -0.319469     0.302404        0.128291  -0.024974
4      -1.025284         -0.319469    -0.061651       -1.152978  -0.077811

   free sulfur dioxide  total sulfur dioxide   density        pH  sulphates  \
0             0.045382              0.302872  0.204875 -0.456163   0.277604
1             0.731623              1.520640 -0.385499  0.525985  -1.499490
2             0.376436             -0.195302  1.910673 -0.190999   1.783015
3             1.444265              0.672579  0.677758  0.137134  -0.805208
4             0.472899             -0.195302 -1.043367  0.525985  -0.209848

    alcohol   quality
0 -0.763278  0.504866
1 -0.912739 -0.783991
2 -0.346204  0.504866
3 -0.479903 -0.783991
4  0.429205  0.504866
```

[ ]:
```
# Generate a summary of statistics for the numerical columns in the transformed␣
↪DataFrame
transformed_df.describe()
```

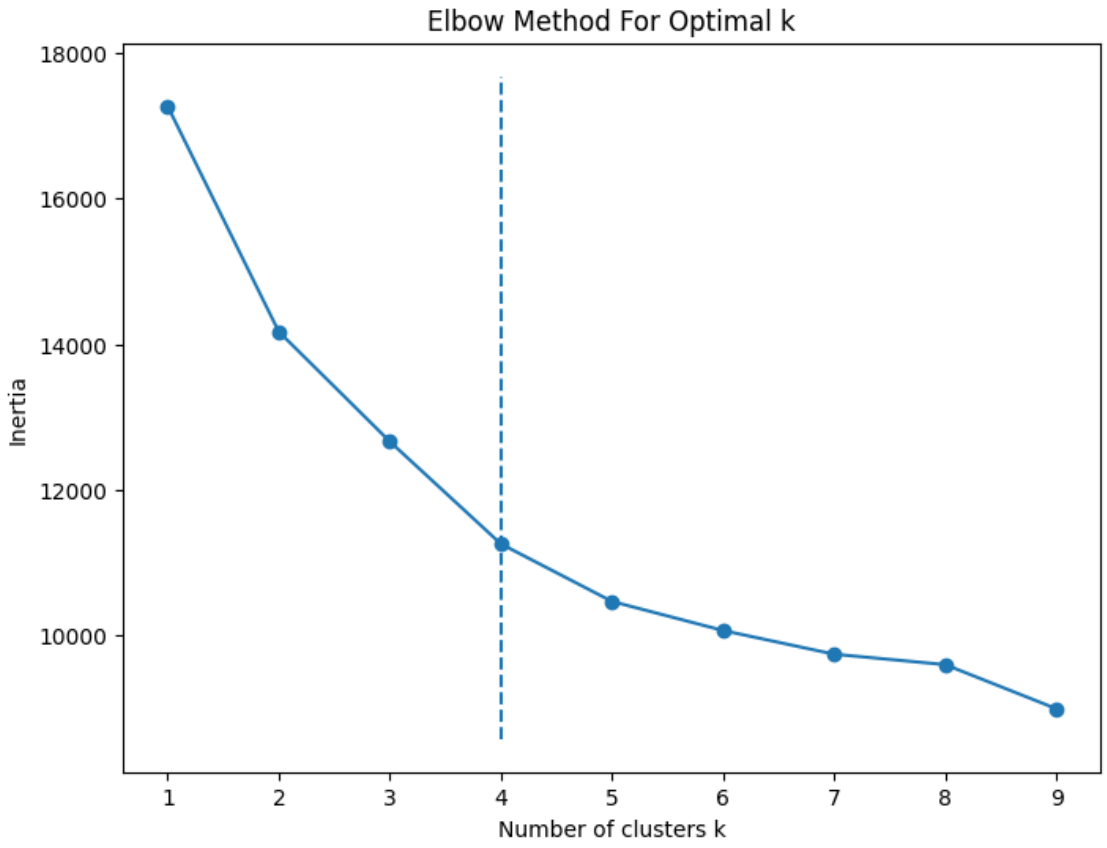
[ ]:
```
       fixed acidity  volatile acidity   citric acid  residual sugar  \
count   1.439000e+03      1.439000e+03  1.439000e+03    1.439000e+03
mean    7.406630e-18      1.975101e-17  2.468877e-18    2.345433e-17
std     1.000348e+00      1.000348e+00  1.000348e+00    1.000348e+00
min    -3.217198e+00     -2.759209e+00 -1.504068e+00   -4.245831e+00
25%    -6.784884e-01     -7.428172e-01 -9.338817e-01   -6.132979e-01
```

```
50%    -6.348387e-02      4.666358e-02 -7.748552e-03    3.745184e-02
75%     6.951034e-01      7.022795e-01  8.247018e-01    6.344031e-01
max     2.265131e+00      2.617478e+00  2.562199e+00    2.196453e+00

          chlorides  free sulfur dioxide  total sulfur dioxide       density  \
count  1.439000e+03         1.439000e+03          1.439000e+03  1.439000e+03
mean  -4.073647e-17        -1.975101e-17         -2.715764e-17 -1.975101e-17
std    1.000348e+00         1.000348e+00          1.000348e+00  1.000348e+00
min   -6.327738e+00        -2.109459e+00         -1.882147e+00 -3.146238e+00
25%   -4.847219e-01        -9.775682e-01         -8.299395e-01 -6.034510e-01
50%    2.653772e-02         4.538218e-02         -1.386768e-02  2.620726e-03
75%    5.069584e-01         8.095896e-01          8.002595e-01  5.863865e-01
max    2.695884e+00         2.141803e+00          2.044647e+00  2.933204e+00

                 pH     sulphates       alcohol      quality
count  1.439000e+03  1.439000e+03  1.439000e+03  1439.000000
mean  -2.221989e-17  9.258288e-18  1.481326e-17     0.000000
std    1.000348e+00  1.000348e+00  1.000348e+00     1.000348
min   -3.133580e+00 -3.628378e+00 -2.872962e+00    -3.498357
25%   -6.563182e-01 -6.989057e-01 -9.127391e-01    -0.783991
50%    6.372044e-03 -3.483241e-02 -9.539024e-02     0.504866
75%    5.903673e-01  7.181358e-01  8.228354e-01     0.504866
max    2.957244e+00  2.323094e+00  2.154480e+00     2.616243
```

[ ]:
```python
# Initialize an empty list to store the inertia values for different cluster counts
inertia = []

# Define a range of values from 1 to 9 to test the number of clusters
k_values = range(1, 10)

# Iterate over the range of k_values to apply KMeans clustering for each k
for k in k_values:
    # Create a KMeans instance with the current number of clusters
    kmeans = KMeans(n_init = 'auto', n_clusters = k, random_state = random_state)

    # Fit the KMeans model on the transformed dataset
    kmeans.fit(transformed_df)

    # Append the inertia (within-cluster sum of squares) to the inertia list
    inertia.append(kmeans.inertia_)

# Use the KneeLocator to find the elbow point in the inertia plot
knee_locator = KneeLocator(k_values, inertia, curve = 'convex', direction = 'decreasing')
```

```
# Determine the optimal number of clusters (elbow point) from the KneeLocator
optimal_k = knee_locator.elbow

# Print the optimal number of clusters
print(f"The optimal number of clusters is: {optimal_k}")

# Plotting the elbow method graph
plt.figure(figsize = (8, 6))
plt.plot(k_values, inertia, 'o-')
plt.xlabel('Number of clusters k')
plt.ylabel('Inertia')
plt.title('Elbow Method For Optimal k')
plt.vlines(optimal_k, plt.ylim()[0], plt.ylim()[1], linestyles = 'dashed')
plt.show()
```

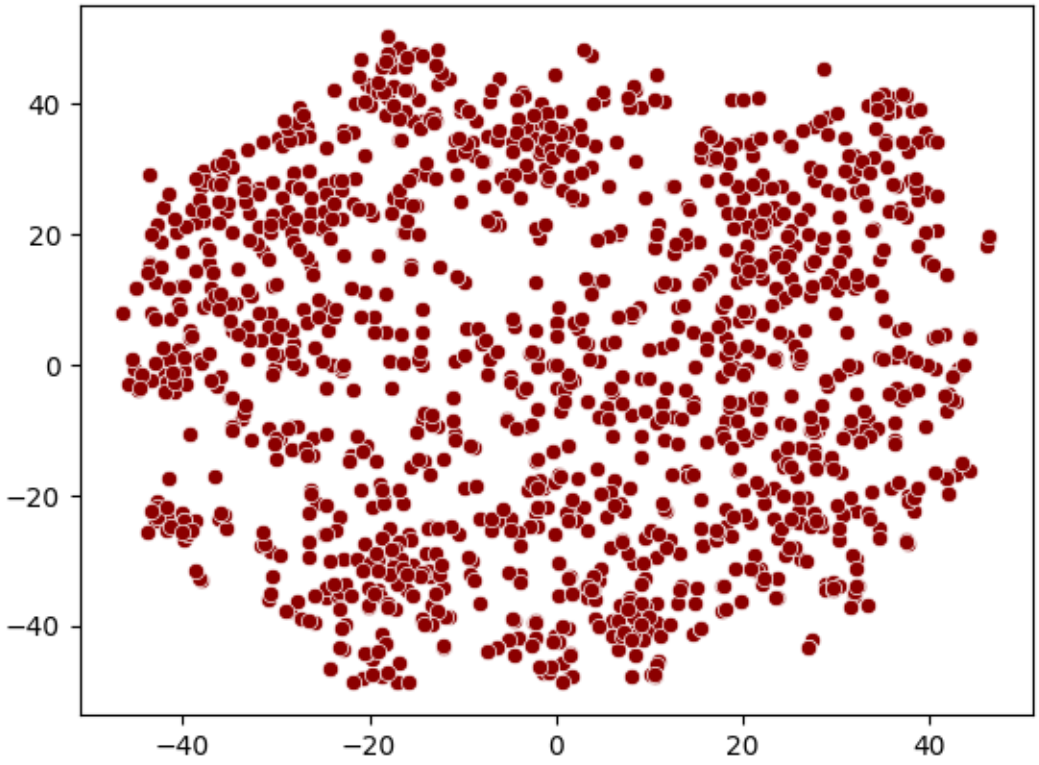
```
The optimal number of clusters is: 4
```



[ ]:
```
# Fit tranforming the data to t-SNE
from sklearn.manifold import TSNE
```

```
# Initialize the TSNE model
TSNE = TSNE(random_state = random_state)
TSNE_model = TSNE.fit_transform(transformed_df)

# Visualize the transformed data using a scatter plot
sns.scatterplot(x = TSNE_model[:, 0], y = TSNE_model[:, 1], color = 'darkred')
```

[ ]: <Axes: >



[ ]:
```
# Initialize the KMeans clustering model
kmeans = KMeans(n_init = 'auto', n_clusters = optimal_k, random_state =␣
 ↪random_state)

# Fit the KMeans model to the transformed data and predict the cluster for each␣
 ↪observation
clusters = kmeans.fit_predict(df_transformed)
```

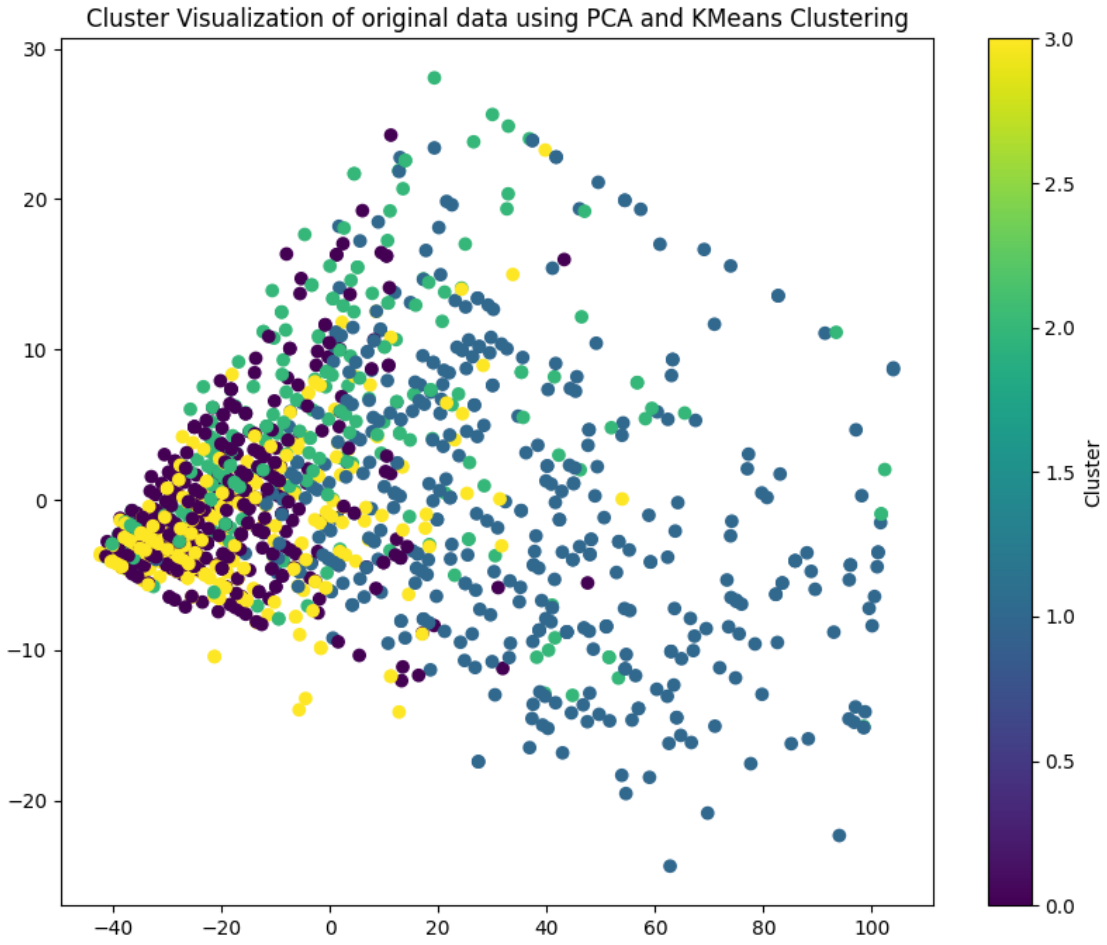
[ ]:
```
# Add a new column named 'Cluster' to the 'data' DataFrame
# This column contains the cluster labels assigned to each data point in the␣
 ↪DataFrame
data['Cluster'] = clusters
```

[ ]:
```
# Evaluate the quality of the clusters formed by KMeans clustering
# The Silhouette Score is used as a metric for this purpose
# It measures how similar an object is to its own cluster compared to other␣
 ↪clusters
# The score ranges from -1 (incorrect clustering) to +1 (highly dense␣
 ↪clustering)
# A score near 0 indicates overlapping clusters
silhouette_avg = silhouette_score(data, clusters)

# Print the average Silhouette Score to assess the clustering performance
print(f"Silhouette Score: {silhouette_avg}")
```

```
Silhouette Score: 0.013069146152542071
```

A silhouette score of 0.013069146152542071, indicates that the clusters are very weakly defined. This score is close to 0, suggesting that the clusters are overlapping or that the data points are almost equally distant from their own cluster and the nearest neighboring cluster.

[ ]:
```
# Initialize the PCA (Principal Component Analysis) model
# The model is set to reduce the data to 2 principal components
pca = PCA(n_components = 2)

# Apply PCA to the data to reduce its dimensionality
# The fit_transform method fits the PCA model to the data and then transforms it
# The result is a new dataset with reduced dimensions
reduced_data = pca.fit_transform(data)

# Plotting the clusters on the first two principal components
plt.figure(figsize = (10, 8))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c = clusters, cmap =␣
 ↪'viridis')
plt.title('Cluster Visualization of original data using PCA and KMeans␣
 ↪Clustering')
plt.colorbar(label = 'Cluster')
plt.show()
```

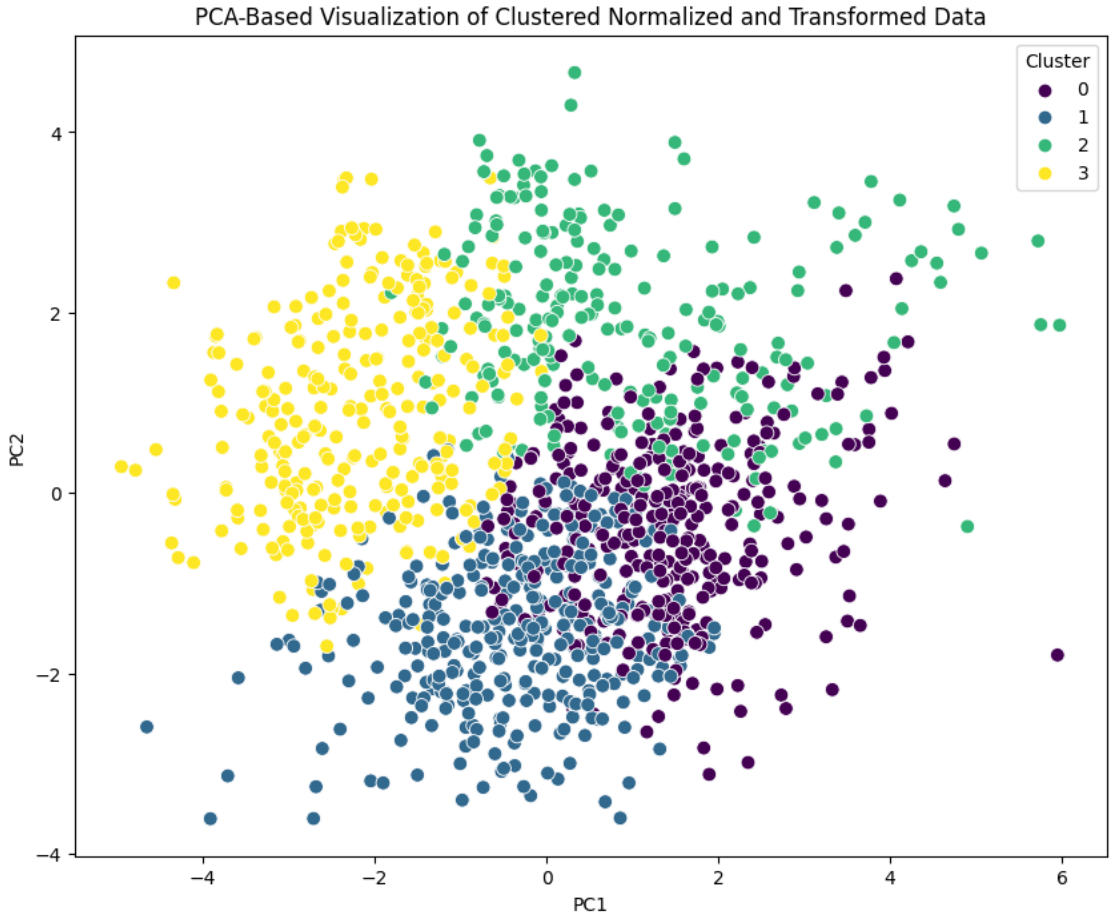


[ ]:
```
# Evaluate the quality of the clusters formed by KMeans clustering
# The Silhouette Score is used as a metric for this purpose
# It measures how similar an object is to its own cluster compared to other␣
 ↪clusters
# The score ranges from -1 (incorrect clustering) to +1 (highly dense␣
 ↪clustering)
# A score near 0 indicates overlapping clusters
silhouette_avg = silhouette_score(transformed_df, clusters)

# Print the average Silhouette Score to assess the clustering performance
print(f"Silhouette Score: {silhouette_avg}")
```

```
Silhouette Score: 0.15284705062898654
```

A silhouette score of 0.15284705062898654 indicates moderate, but not ideal, separation between clusters.

[ ]:
```
# Initialize the PCA model to reduce the dataset to 2 principal components
pca = PCA(n_components = 2)

# Apply PCA to the transformed dataset
# This transforms the data into a new space with 2 principal components
principal_components = pca.fit_transform(transformed_df)

# Create a new DataFrame with the 2 principal components
# This DataFrame will be used for visualization
pc_df = pd.DataFrame(data = principal_components, columns = ['PC1', 'PC2'])

# Add the cluster labels to the DataFrame
# These labels indicate the cluster each data point belongs to
pc_df['Cluster'] = clusters

# Create a scatter plot to visualize the clusters
plt.figure(figsize=(10, 8))
sns.scatterplot(x = 'PC1', y = 'PC2', hue = 'Cluster', data = pc_df, palette =␣
 ↪'viridis', s = 60)
plt.title("PCA-Based Visualization of Clustered Normalized and Transformed␣
 ↪Data")
plt.show()
```



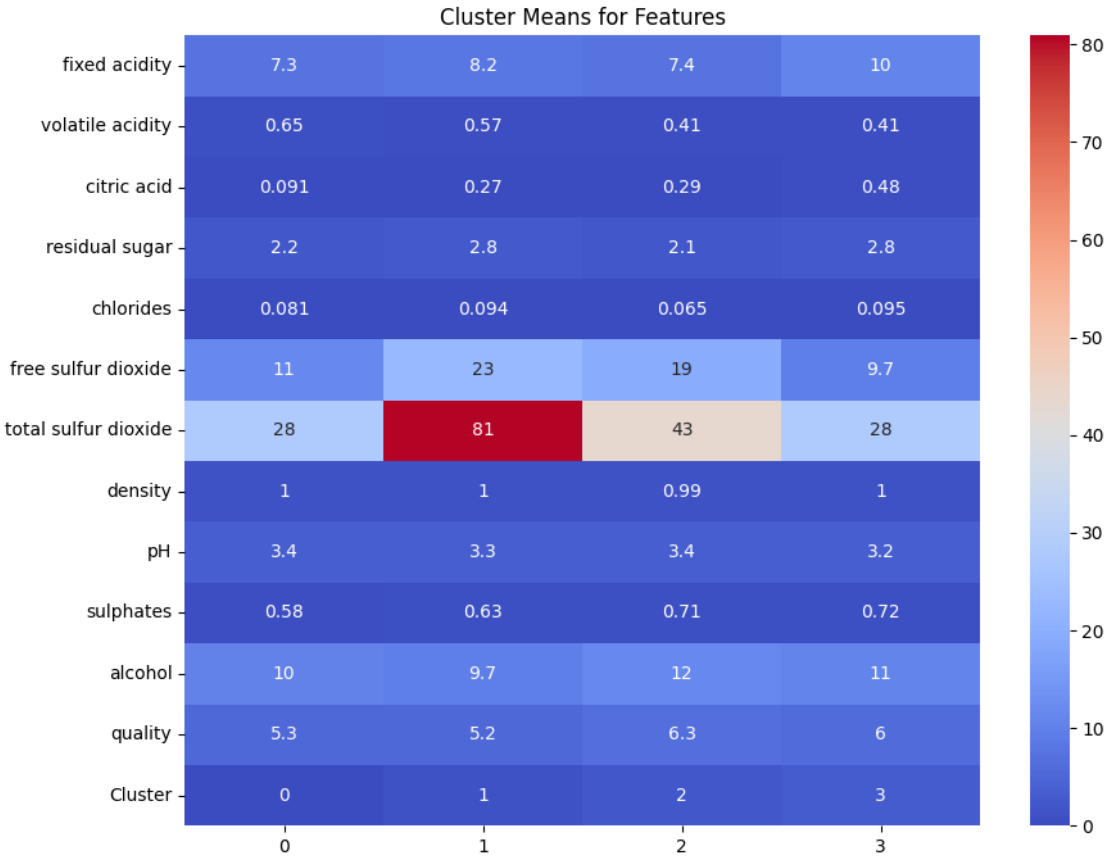
[ ]:
```
# Calculate mean values for each feature in each cluster
cluster_means = data.groupby(clusters).mean()

# Display the calculated mean values
cluster_means
```

[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	7.269928	0.649978	0.090931	2.177486	0.081499	11.379475	28.210024	0.996184	3.401580	0.580807
1	8.170813	0.568598	0.266885	2.765951	0.093675	22.991562	80.915738	0.997442	3.285017	0.634712
2	7.397026	0.410595	0.285651	2.107249	0.065375	19.426809	43.469116	0.994706	3.374477	0.714001
3	10.448322	0.414925	0.479670	2.762313	0.094628	9.713100	28.309309	0.998227	3.179099	0.721503

	alcohol	quality	Cluster
0	10.212291	5.339616	0.0
1	9.694896	5.204385	1.0
2	11.572102	6.342007	2.0
3	10.556546	5.984985	3.0

[ ]:
```
# Generate a heatmap
# The heatmap displays the mean values of each feature for each cluster
# 'cluster_means.T' transposes the cluster_means DataFrame for better visualization
# Each cell in the heatmap represents the mean value of a feature in a cluster
# 'annot=True' annotates the heatmap with the actual mean values
# 'cmap='coolwarm'' sets the color palette of the heatmap
plt.figure(figsize = (10, 8))
sns.heatmap(cluster_means.T, annot = True, cmap = 'coolwarm')
plt.title('Cluster Means for Features')
plt.show()
```



[ ]:
```
# Fit the model
kmeans_model = kmeans.fit(transformed_df)
```

1.16 Save Model

[ ]:
```
# Save the model to file
filename = 'finalized_model.pkl'

with open(filename, 'wb') as file:
  pickle.dump(kmeans_model, file)
```

1.17 Load the Model

[ ]:
```
with open(filename, 'rb') as file:
  loaded_model = pickle.load(file)

loaded_model
```

[ ]:
```
KMeans(n_clusters=4, n_init='auto', random_state=42)
```

1.18 Conclusions

In this study, we successfully applied clustering techniques to categorize red wines based on their physicochemical properties. The analysis revealed 4 distinct clusters of wines, each characterized by unique combinations of pH, density, residual sugars, and sulphates, among other attributes. These clusters effectively grouped wines with similar chemical profiles, transcending traditional classification methods based on region or vintage.

The insights gained from this clustering approach are significant. They highlight the potential of using chemical composition as a basis for understanding and predicting wine flavors. This method provides a more nuanced appreciation of wine characteristics, moving beyond regional and vintage classifications to a deeper, more intrinsic understanding of wine flavors.

Furthermore, my findings have practical implications for personalized wine recommendations. By identifying wines that fall into the same cluster, I can suggest alternatives that a person is likely to enjoy, based on their preference for a particular wine's chemical profile. This approach could revolutionize the way we select and appreciate wines, making it more tailored and informed.

In conclusion, the project demonstrates the power of machine learning in enhancing our understanding of complex products like wine. The use of clustering to analyze the physicochemical properties of wine opens up new avenues for exploration in the field of oenology and could lead to innovative strategies in wine production, marketing, and consumption. As we continue to gather more data on wine characteristics, the potential for even more refined clustering and recommendations becomes increasingly promising.