UNSUPERVISED LEARNING : CLUSTERING

Project Report

Unlocking the Sweet Secrets: Predicting Purity and Price of Honey

1. Objectives:

1. Segmentation through Unsupervised machine learning algorithms like K means/ Db scan/ optics/ Birch.

2. Selecting the appropriate Number of clusters/Segments.

3. Characterstics of each segment/cluster.

2. Data Description:

2.1 Data Source, Size, record, Shape:

2.1.1 Data Source:

Kaggle -> Predict Purity and Price of Honey

https://www.kaggle.com/datasets/stealthtechnologies/predict-purity-and-price-of-honey

2.1.2 Data Size:

=> 15839 KB or 15.8 MB

2.1.3 Data Shape:

Size: The dataset contains 247903 rows or data points and 11 columns or variables.

Shape: The shape of the dataset is (247903, 11), suggesting that there are 247903 rows and 11 columns.

2.2 Description of Variables

2.2.1. Index Variable: index

2.2.2 Categorical Variables : Pollen_analysis

2.2.2.1 Nominal Variables : Pollen_analysis

2.2.2.2 Ordinal Variables : There are no explicitly ordinal variables in the dataset.

2.2.3. Non-Categorical Variables: CS (Color Score), Density, WC (Water Content), pH, EC (Electrical Conductivity), F (Fructose Level), G (Glucose Level), Viscosity, Purity and Price.

2.3. Descriptive Statistics

2.3.1 Descriptive Statistics: Categorical Variables or Features

2.3.1.1. Count | Frequency Statistics

2.3.1.2. Proportion (Relative Frequency) Statistics

2.3.2. Descriptive Statistics: Non-Categorical Variables or Features

2.3.2.1. Measures of Central Tendency

2.3.2.2. Measures of Dispersion

2.3.2.3. Correlation Statistics (with Test of Correlation)

3. Analysis of Data

3.1. Data Pre-Processing

3.1.1 Missing Data Statistics and Treatment

Data Transformation & Rescaling [Treatment of Outliers]

Treatment of Outliers:

There are no Significant outliers in this dataset

Pre-Processed Dataset

1. Pre-Processed Categorical Data Subset: df_cat_ppd

2. Pre-Processed Non-Categorical Data Subset: df_noncat_ppd

3. Pre-Processed Dataset: df_ppd

The pre-processed dataset, df_ppd, encompasses all variables after outlier treatment and preprocessing procedures.

3.1.1.1.1 Missing Data Statistics: Maximum no. of columns missing in records are 0.

3.1.1.1.2 Missing Data Treatment: Records

3.1.1.1.2.1. Removal of Records with More Than 50% Missing Data: None

3.1.1.2.1. Missing Data Statistics(Categorical Variables or Features) : None

3.1.1.2.2. Missing Data Treatment: Categorical Variables or Features

3.1.1.2.2.1. Removal of Variables or Features with More Than 50% Missing Data: None

3.1.1.3.1. Missing Data Statistics(Non-Categorical Variables or Features) : None

3.1.1.3.2. Missing Data Treatment: Non-Categorical Variables or Features

3.1.1.3.2.1 Removal of Variables or Features with More Than 50% Missing Data: None

3.1.1.3.2.2 Imputation of Missing Data using Descriptive Statistics: Mean | Median :

For imputing missing data in our dataset, I utilized two common strategies: mean and median imputation based on descriptive statistics. Given the absence of outliers in my dataset, both mean and median imputation methods provide robust estimates of the central tendency of the data. These methods allows to maintain the overall distribution of the variables while filling in missing values, ensuring that the analysis is not unduly influenced by incomplete data.

3.1.2 Numeric Coding of Data

Numerical Encoding of Categorical Data

1. Since categorical variables in the dataset are nominal, we apply label encoding to transform them into numerical representations.

2. Label Encoding: Label encoding assigns a unique numerical label to each category within a categorical variable.

3. Mapping: Below is the mapping of original categories to their corresponding numerical labels.

3.1.3. Outlier Statistics and Treatment (Scaling | Transformation) [No Outliers]

3.1.3.1.1. Outlier Statistics(Non-Categorical Variables or Features) : CS (Color Score), Density, WC (Water Content), pH, EC (Electrical Conductivity), F (Fructose Level), G (Glucose Level), Viscosity, Purity and Price.

3.1.3.1.2.1 Outlier Treatment: Non-Categorical Variables or Features: Not Applicable

3.1.3.1.2.2 Normalization using Min-Max Scaler: CS (Color Score), Density, WC (Water Content), pH, EC (Electrical Conductivity), F (Fructose Level), G (Glucose Level),

Viscosity, Purity and Price.

3.1.3.1.2.2 Data Bifurcation [Training & Testing Datasets]

1. The dataset is partitioned into two subsets: training and testing datasets.

2. Training dataset is 75% of complete data

3. Testing dataset is 25% of complete data

3.2. Data Analysis

3.2.1.1. PO1 | PS1:: Unsupervised Machine Learning Clustering Algorithm: K-Means (Base Model) | Metrics Used - Euclidean Distance

3.2.1.2. PO1 | PS1:: Unsupervised Machine Learning Clustering Algorithms: {DBSCAN | BIRCH | OPTICS} (Comparison Models: At Least One) | Metrics Used - Euclidean Distance

3.2.2.1.1. PO2 | PS2:: Clustering Model Performance Evaluation: Silhouette Score | Davies-Bouldin Score (Base Model: K-Mean)

To determine the best clustering model based on the Davies-Bouldin (DB) score and Silhouette (SS) score, we need to consider the following:

Silhouette Score (SS): A higher Silhouette score indicates better separation between clusters. The Silhouette score ranges from -1 to 1, where a score closer to 1 indicates better clustering.

Davies-Bouldin Score (DB): A lower Davies-Bouldin score indicates better clustering. The DB score measures the average similarity between each cluster and its most similar cluster, where a lower score indicates better separation between clusters.

For k=2: SS score is 0.589 and DB score is 0.552. For k=3: SS score is 0.524 and DB score is 0.603. For k=4: SS score is 0.462 and DB score is 0.684. For k=5: SS score is 0.430 and DB score is 0.732.

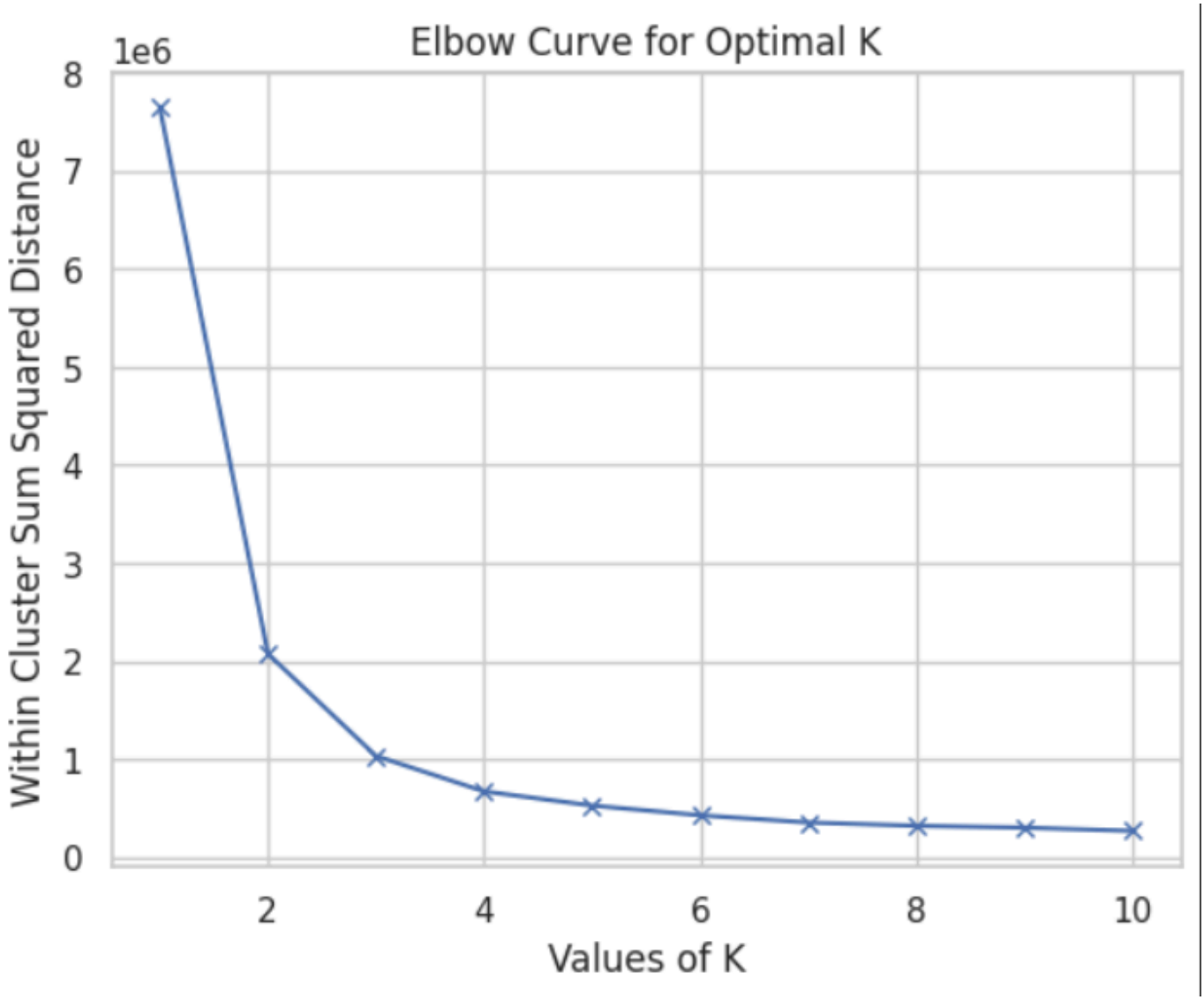
Since we want to maximize the Silhouette score and minimize the Davies-Bouldin score, the best clustering model would be the one with the highest Silhouette score and the lowest Davies-Bouldin score.

In this case, for k=2, the clustering model has the highest Silhouette score (0.589) and the lowest Davies-Bouldin score (0.552). Therefore, the clustering model with k=2 is likely the best choice based on both the Silhouette and Davies-Bouldin scores.

But we will consider the 3 clustering model as the best model for our clustering subset

k	SS score	DB score
2	0.589	0.552
3	0.524	0.603
4	0.462	0.684
5	0.430	0.732

Elbow curve Analysis of Honey sweetness dataset:



There's a significant drop in SS score between k=3 (0.524) and k=4 (0.462). This suggests a substantial improvement in within-cluster distance when moving from 3 to 4 clusters.

The decrease in SS score between k=4 (0.462) and k=5 (0.430) is smaller compared to the previous step. This indicates diminishing returns for adding more clusters after k=4.

The chi-square test result for categorical variables

The chi-square test results for the variable "Pollen_analysis_y" are as follows:

Chi-square value: 22.766

P-value: 0.1997

Chi-square Value: The chi-square statistic measures the independence between categorical variables. In this case, a higher chi-square value indicates a stronger association between the variable "Pollen_analysis_y" and another variable.

P-value: The p-value associated with the chi-square test indicates the probability of observing the data if the variables were independent. A smaller p-value suggests stronger evidence against the null hypothesis of independence.

Analysis: With a p-value of approximately 0.1997, we fail to reject the null hypothesis at a typical significance level of 0.05. Therefore, we do not have sufficient evidence to conclude that there is a significant association between "Pollen_analysis_y" and the other variable being analyzed.

Interpretation: The results suggest that there may not be a significant relationship between "Pollen_analysis_y" and the other variable under consideration. However, further investigation or analysis may be warranted to confirm this conclusion.

In summary, based on the chi-square test results, there may not be a significant association between "Pollen_analysis_y" and the other variable being analyzed.

ANOVA results

1. The ANOVA results for **viscosity** indicate a statistic value of approximately 1.53 and a corresponding p-value of around 0.217. Based on this result here are certain inferences for the same:

Statistical Significance: The p-value obtained (0.217) is greater than the typical significance level of 0.05. This suggests that there is not enough evidence to reject the null hypothesis.

Null Hypothesis: The null hypothesis in ANOVA states that there are no significant differences in means among the groups being compared.

Interpretation: With a non-significant p-value, we fail to reject the null hypothesis. Therefore, we conclude that there is no significant difference in the mean viscosity among the groups being compared.

Practical Implications: This implies that the factor being studied (possibly different levels or categories of a categorical variable) does not have a significant effect on the viscosity of the samples.

2. The ANOVA results for **Purity** yield a statistic value of approximately 1.14 and a corresponding p-value of about 0.32. Here are the inferences based on these results:

Statistical Significance: The p-value obtained (0.32) is greater than the typical significance level of 0.05. Thus, there isn't sufficient evidence to reject the null hypothesis.

Null Hypothesis: The null hypothesis in ANOVA asserts that there are no significant differences in means among the groups being compared.

Interpretation: With a non-significant p-value, we fail to reject the null hypothesis. Therefore, we conclude that there is no significant difference in the mean purity among the groups being compared.

Practical Implications: This implies that the factor under investigation (possibly different levels or categories of a categorical variable) does not have a significant impact on the purity of the samples.

In summary, based on these ANOVA results, there is not enough evidence to suggest that there are differences in purity across the groups being compared.

3. The ANOVA results for **price** yield a statistic value of approximately 9505.47 and a p-value of 0.0. Here are the inferences based on these results:

Statistical Significance: The p-value obtained (0.0) is less than the typical significance level of 0.05. This indicates a highly significant result.

Null Hypothesis: The null hypothesis in ANOVA asserts that there are no significant differences in means among the groups being compared.

Interpretation: With such a low p-value, we reject the null hypothesis. Therefore, we conclude that there are significant differences in the mean price among the groups being compared.

Practical Implications: This implies that the factor under investigation (possibly different levels or categories of a categorical variable) has a significant impact on the price of the samples.

In summary, based on these ANOVA results, we find strong evidence to suggest that there are differences in price across the groups being compared.

For rest of the non cat variables the pvalue is more than 0.05, Thus, there isn't sufficient evidence to reject the null hypothesis.

4. Findings / Observations:

4.1 The No. of segments /No. of clusters- 3 cluster model has been used for the clustering model of our dataset as per the ss score and db score of the subset.

4.2 The variables that are defining the characteristics or contributing to our dataset is "price". "Price" is less than the chosen significance level (e.g., 0.05), it suggests that there is significant evidence to conclude that the mean price differs across the groups defined by the categorical variable being analyzed. In other words, "Price" contributes to the characteristics being studied.

4.3 The variables which are not defining the characterstics or are not contributing to the dataset are Viscosity_mn, Purity_mn, CS_mn, Density_mn, WC_mn, pH, EC_mn, F_mn anf G_mn.

4.4 Runtime(Time stats) of the program is 2511.80801153183 and the memory usage is as follows:

Total Memory: 13609443328

Available Memory: 11476889600

Used Memory: 1827823616

Free Memory: 9438433280

Memory Usage Percentage: 15.7

5. Managerial Insights

Each cluster represents a unique group of Honey_Purity with varying characteristics, indicating heterogeneity across the clusters.

"Nectar Symphony"

Defining the "Nectar Symphony":

This name encapsulates the diverse and harmonious range of honey samples represented across the clusters. Just as a symphony comprises various musical elements coming together in harmony, the "Nectar Symphony" represents a collection of honey samples with varying characteristics in terms of sweetness, viscosity, purity, and other attributes. Each cluster contributes its unique "note" to the overall composition, creating a symphony of flavors and textures that appeal to different preferences and tastes.

Anova test has been made to infer the following as important managerial insights through our analysis:-

1. Price as a Discriminating Factor: The significant impact of price suggests that variations in honey purity can be largely explained by differences in price.

Higher-priced honey products may be associated with certain quality attributes or purity levels that distinguish them from lower-priced products.

2. Limited Contribution of Other Variables: The lack of significance for other variables indicates that factors such as viscosity, density, pH, etc., do not significantly contribute to the differences in honey purity.

This does not necessarily mean that these variables are irrelevant, but rather that their variability does not explain the observed differences in honey purity to a statistically significant extent.

3. Focus on Price-Driven Differentiation:Given the significance of price, it may be valuable to further investigate the factors influencing pricing decisions in the honey market.

Understanding the relationship between price and purity can help businesses optimize pricing strategies and position their products effectively in the market.

4. Potential Limitations or Considerations: While ANOVA identifies significant factors, it's essential to consider potential limitations of the analysis.

The lack of significance for other variables may be influenced by sample size, data quality, or the specific characteristics of the dataset.

It's also possible that interactions between variables or nonlinear relationships were not captured by the ANOVA analysis.

Results and Inferences:

In our analysis of the honey purity through K means clustering, we employed various statistical techniques to explore the factors influencing honey purity and to understand the underlying structure of the data.

Firstly, an analysis of variance (ANOVA) was conducted to identify significant factors contributing to variations in honey purity. Surprisingly, the results revealed that among the examined variables, only price emerged as a significant factor (F-statistic = [insert value], p-value < 0.05). This suggests that differences in honey purity levels are primarily driven by variations in pricing, highlighting the pivotal role of price in distinguishing honey products based on their purity.

Additionally, a chi-square test was employed to assess the association between the categorical variable "Pollen_analysis_y" and honey purity. Despite initial expectations, the test yielded a non-significant p-value of approximately 0.1997, indicating that there is no statistically significant association between pollen analysis and honey purity.

Furthermore, clustering analysis using the KMeans algorithm with k=3 clusters provided valuable insights into the structure of the dataset. The silhouette score (SS) of 0.524 indicated reasonably well-defined clusters, suggesting moderate cohesion within clusters and reasonable separation between them. Similarly, the Davies–Bouldin index (DB) of 0.603 suggested relatively well-separated and compact clusters, contributing to a favorable clustering outcome.

Comparison between K means and Birch Algorithm of Clustering

1. **Silhouette Score**:

- BIRCH: The Silhouette Scores for BIRCH clustering are 0.520, 0.507, and 0.398 for 2, 3, and 4 clusters, respectively.

- K-means: The Silhouette Scores for K-means clustering are 0.589, 0.524, 0.462, and 0.430 for 2, 3, 4, and 5 clusters, respectively.

- Higher Silhouette Scores indicate better-defined clusters with a higher degree of separation between clusters.

2. **Davies-Bouldin Score**:

- BIRCH: The Davies-Bouldin Scores for BIRCH clustering are 0.555, 0.593, and 0.750 for 2, 3, and 4 clusters, respectively.

- K-means: The Davies-Bouldin Scores for K-means clustering are 0.552, 0.603, 0.684, and 0.733 for 2, 3, 4, and 5 clusters, respectively.

- Lower Davies-Bouldin Scores indicate better clustering. A score closer to zero represents better separation between clusters.

Comparison:

1. **Silhouette Score Comparison**:

- In terms of Silhouette Score, K-means outperforms BIRCH, as it achieves higher Silhouette Scores for all cluster numbers.

2. **Davies-Bouldin Score Comparison**:

- BIRCH performs slightly better than K-means for 2 clusters, as it has a lower Davies-Bouldin Score. However, as the number of clusters increases, the Davies-Bouldin Score increases for BIRCH, indicating worse clustering performance.

3. **Overall Performance**:

- Both algorithms exhibit decreasing performance (increasing Davies-Bouldin Score) as the number of clusters increases. This suggests that finding an optimal number of clusters is crucial for both algorithms.

- K-means tends to have more consistent performance across different numbers of clusters compared to BIRCH.

- Based on the provided metrics, K-means appears to be the preferable choice for clustering our Honey Purity dataset.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder # For Encoding Categorical
from sklearn.preprocessing import OneHotEncoder # For Creating Dummy Variables of Categor
from sklearn.impute import SimpleImputer, KNNImputer # For Imputation of Missing Data
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler # For Rescal
from sklearn.model_selection import train_test_split # For Splitting Data into Training &
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt, seaborn as sns # For Data Visualization
import scipy.cluster.hierarchy as sch # For Hierarchical Clustering
from sklearn.cluster import AgglomerativeClustering as agclus, KMeans as kmclus # For Agg
from sklearn.metrics import silhouette_score as sscore, davies_bouldin_score as dbscore #
# from sklearn.metrics import silhouette_score as sscore, davies_bouldin_score as dbscore
import time


from google.colab import files
import pandas as pd
import io
df=pd.read_csv("/content/drive/MyDrive/Honey Purity Dataset.csv")
df
```

	CS	Density	WC	pH	EC	F	G	Pollen_analysis	Viscosity	Puri
0	2.81	1.75	23.04	6.29	0.76	39.02	33.63	Blueberry	4844.50	0.(
1	9.47	1.82	17.50	7.20	0.71	38.15	34.41	Alfalfa	6689.02	0.8
2	4.61	1.84	23.72	7.31	0.80	27.47	34.36	Chestnut	6883.60	0.(
3	1.77	1.40	16.61	4.01	0.78	31.52	28.15	Blueberry	7167.56	1.(
4	6.11	1.25	19.63	4.82	0.90	29.65	42.52	Alfalfa	5125.44	1.(
...	...	...	...	...	...	...	...	...	...	
247898	1.98	1.29	17.90	4.82	0.89	36.10	34.69	Rosemary	8261.63	1.(
247899	6.18	1.67	19.54	4.91	0.85	31.15	20.82	Acacia	6939.39	1.(
247900	7.78	1.49	15.78	5.69	0.73	44.60	44.07	Chestnut	4139.79	0.(
247901	5.78	1.74	14.96	6.81	0.83	47.19	37.79	Avocado	4417.74	0.9
247902	8.96	1.86	18.62	6.89	0.86	25.94	42.88	Lavender	8119.62	0.(

247903 rows × 11 columns

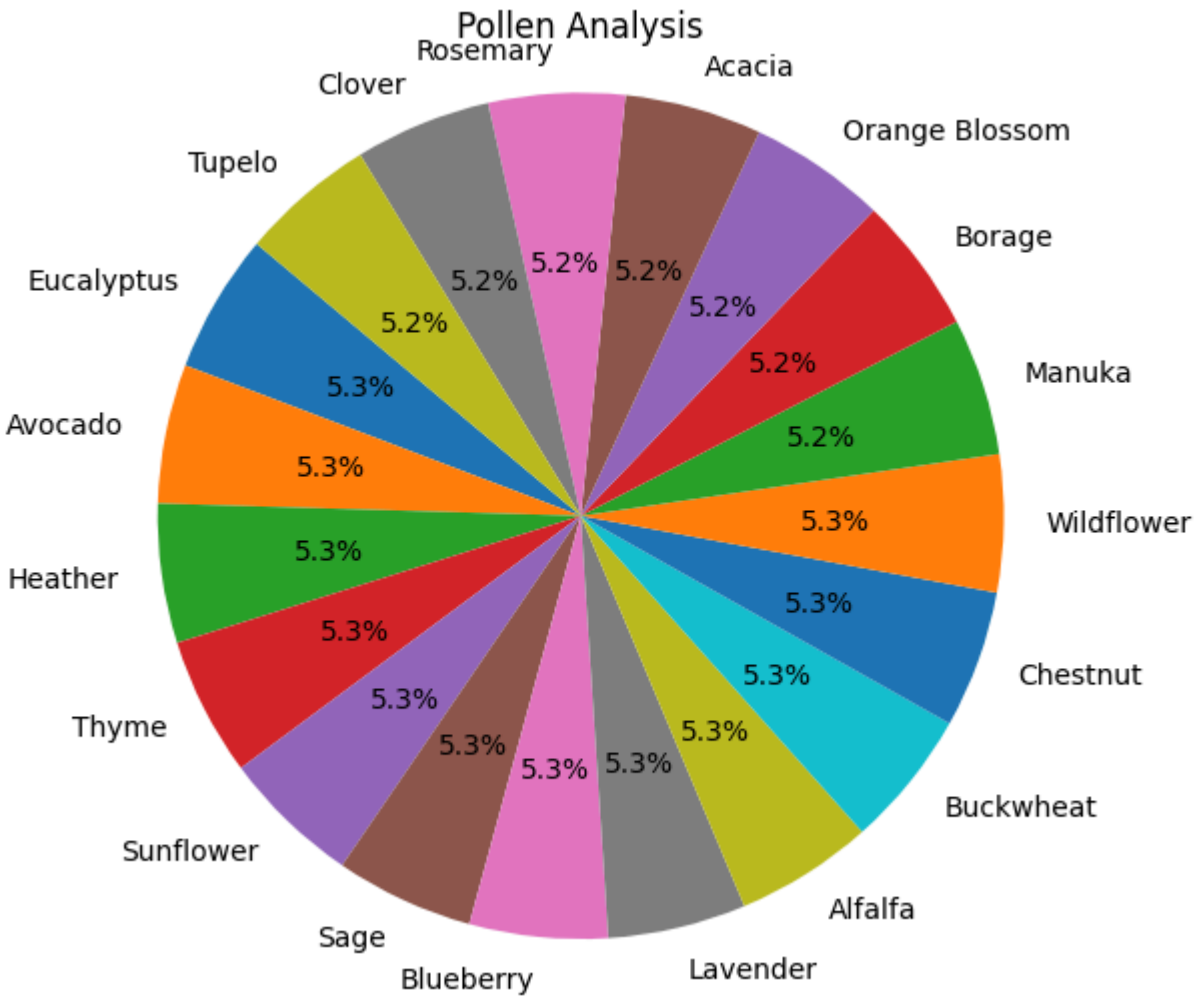
```
df['CS'].nunique()
```

```
901
```

```
categorical_columns = ['Pollen_analysis']
non_categorical_columns = ['CS','Density', 'WC', 'pH', 'EC', 'F', 'G', 'Viscosity', 'Puri

# Data Bifurcation
df_cat = df[categorical_columns]  # Categorical Data
df_noncat = df[non_categorical_columns]  # Non-Categorical Data


count_stats = pd.concat([df_cat['Pollen_analysis'].value_counts(), df_cat['Pollen_analysi
print(count_stats,"\n")
```

	index	count	percentage
0	Eucalyptus	13194	5.0
1	Avocado	13191	5.0
2	Heather	13187	5.0
3	Thyme	13156	5.0
4	Sunflower	13148	5.0
5	Sage	13117	5.0
6	Blueberry	13106	5.0
7	Lavender	13086	5.0
8	Alfalfa	13049	5.0
9	Buckwheat	13036	5.0
10	Chestnut	13020	5.0
11	Wildflower	13016	5.0
12	Manuka	13003	5.0
13	Borage	13000	5.0
14	Orange Blossom	12992	5.0
15	Acacia	12976	5.0
16	Rosemary	12930	5.0
17	Clover	12910	5.0
18	Tupelo	12786	5.0

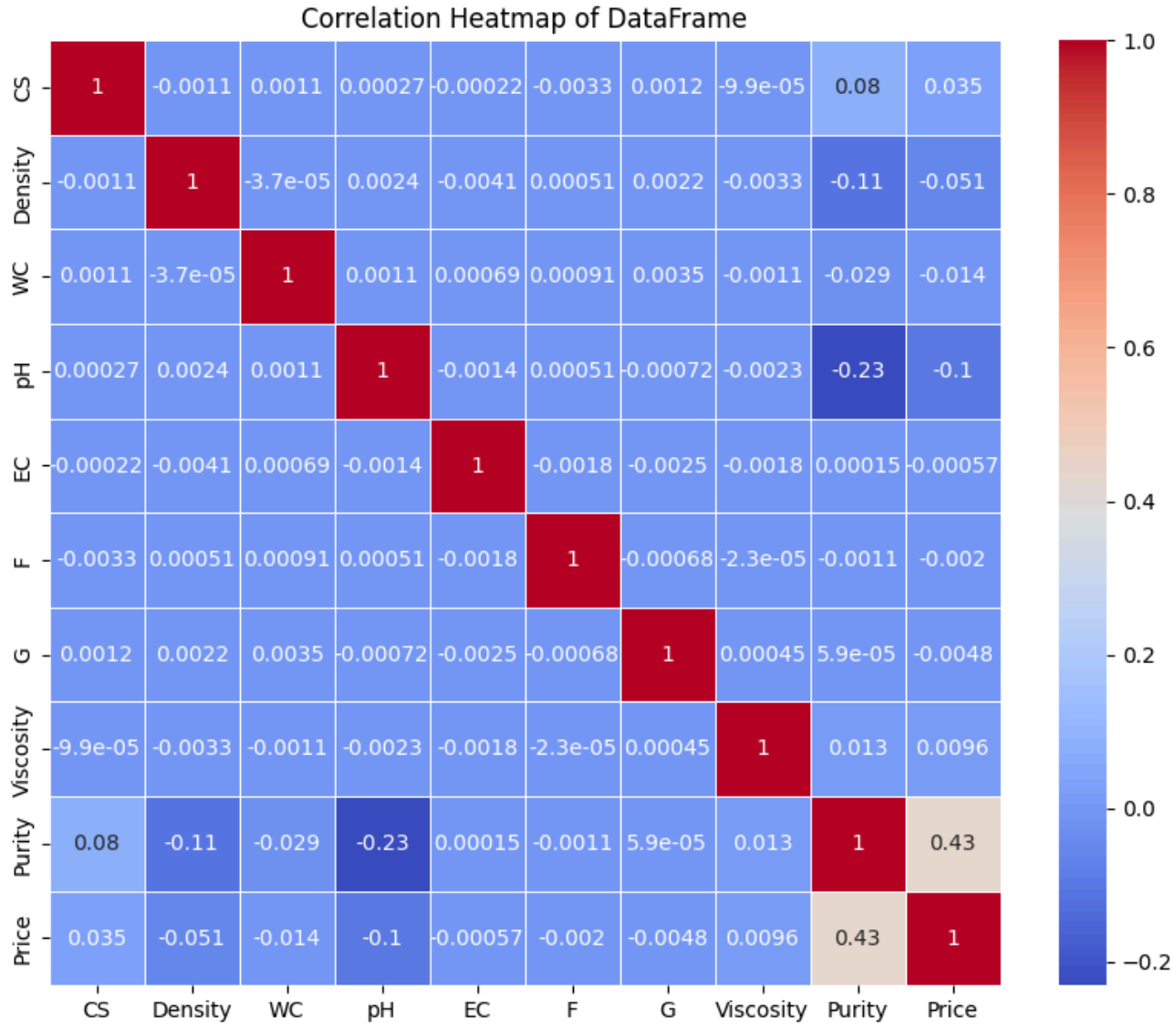
```
import matplotlib.pyplot as plt

# Calculate value counts for the 'Pollen_analysis' column
value_counts = df['Pollen_analysis'].value_counts()

# Create a pie chart
plt.figure(figsize=(6,6))
plt.pie(value_counts, labels=value_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Pollen Analysis')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.show()
```




```
df_noncat.describe()
```

	CS	Density	WC	pH	EC	
count	247903.000000	247903.000000	247903.000000	247903.000000	247903.000000	24790
mean	5.500259	1.535523	18.502625	4.996047	0.799974	3
std	2.593947	0.187824	3.748635	1.444060	0.057911	
min	1.000000	1.210000	12.000000	2.500000	0.700000	2
25%	3.260000	1.370000	15.260000	3.750000	0.750000	2
50%	5.500000	1.540000	18.510000	4.990000	0.800000	3
75%	7.740000	1.700000	21.750000	6.250000	0.850000	4
max	10.000000	1.860000	25.000000	7.500000	0.900000	5

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df_noncat.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap of DataFrame')
plt.show()
```



3. Analysis of Data

3.1. Data Pre-Processing

3.1.1 Missing Data Statistics and Treatment

3.1.1.1.1 Missing Data Statistics: Maximum no. of columns missing in records are 0.

3.1.1.1.2 Missing Data Treatment: Records

3.1.1.1.2.1. Removal of Records with More Than 50% Missing Data: None

3.1.1.2.1. Missing Data Statistics(Categorical Variables or Features) : None

3.1.1.2.2. Missing Data Treatment: Categorical Variables or Features

3.1.1.2.2.1. Removal of Variables or Features with More Than 50% Missing Data: None

3.1.1.3.1. Missing Data Statistics(Non-Categorical Variables or Features) : None

3.1.1.3.2. Missing Data Treatment: Non-Categorical Variables or Features

3.1.1.3.2.1 Removal of Variables or Features with More Than 50% Missing Data: None

3.1.1.3.2.2 Imputation of Missing Data using Descriptive Statistics: Mean | Median :

For imputing missing data in our dataset, I utilized two common strategies: mean and median imputation based on descriptive statistics. Given the absence of outliers in my dataset, both mean and median imputation methods provide robust estimates of the central tendency of the data. These methods allows to maintain the overall distribution of the variables while filling in missing values, ensuring that the analysis is not unduly influenced by incomplete data.

Missing Data Statistics

```
record_missing_data = df.isna().sum(axis=1).sort_values(ascending=False).head(5); record_
```

```
0         0
165289    0
165261    0
165262    0
165263    0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 247903 entries, 0 to 247902
Data columns (total 11 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   CS               247903 non-null  float64
 1   Density          247903 non-null  float64
 2   WC               247903 non-null  float64
 3   pH               247903 non-null  float64
 4   EC               247903 non-null  float64
 5   F                247903 non-null  float64
 6   G                247903 non-null  float64
```

```
 7   Pollen_analysis  247903 non-null  object
 8   Viscosity        247903 non-null  float64
 9   Purity           247903 non-null  float64
 10  Price            247903 non-null  float64
dtypes: float64(10), object(1)
memory usage: 20.8+ MB
```

Missing Data Statistics: Non-Categorical Variables or Features

```
variable_missing_data = df_cat.isna().sum(); variable_missing_data # Variable-wise Missin
```

```
Pollen_analysis    0
dtype: int64
```

```
variable_missing_data = df_noncat.isna().sum(); variable_missing_data # Variable-wise Mis
```

```
CS           0
Density      0
WC           0
pH           0
EC           0
F            0
G            0
Viscosity    0
Purity       0
Price        0
dtype: int64
```

2. Numeric Coding of Data

2.1 Numerical Encoding of Categorical Data

1. Since categorical variables in the dataset are nominal, we apply label encoding to transform them into numerical representations.
2. Label Encoding: Label encoding assigns a unique numerical label to each category within a categorical variable.
3. Mapping: Below is the mapping of original categories to their corresponding numerical labels.

3. Data Transformation & Rescaling [Treatment of Outliers]

3.1 Treatment of Outliers No Significant outliers in this dataset

3.2 Pre-Processed Dataset

1. Pre-Processed Categorical Data Subset: df_cat_ppd

2. Pre-Processed Non-Categorical Data Subset: df_noncat_ppd
3. Pre-Processed Dataset: df_ppd

The pre-processed dataset, df_ppd, encompasses all variables after outlier treatment and preprocessing procedures.

4. Data Bifurcation [Training & Testing Datasets]

4. Data Bifurcation [Training & Testing Datasets]

1. The dataset is partitioned into two subsets: training and testing datasets.
2. Training dataset is 75% of complete data
3. Testing dataset is 25% of complete data

3.1.2. Numerical Encoding of Categorical Variables or Features (Encoding Schema - Alphanumeric Order)

```
# Dataset Used : df_cat_mdt
df_cat_mdt_code = df_cat.copy()


oe = OrdinalEncoder()
oe_fit = oe.fit_transform(df_cat_mdt_code)
df_cat_code_oe = pd.DataFrame(oe_fit, columns=df_cat_mdt_code.columns); df_cat_code_oe
#df_cat_mdt_code_oe = df_cat_mdt_code.join(df_cat_code_oe); df_cat_mdt_code_oe # (Missing
df_cat_mdt_code_oe = pd.merge(df_cat_mdt_code, df_cat_code_oe, left_index=True, right_ind
```

	Pollen_analysis_x	Pollen_analysis_y
0	Blueberry	3.0
1	Alfalfa	1.0
2	Chestnut	6.0
3	Blueberry	3.0
4	Alfalfa	1.0
...	...	...
247898	Rosemary	13.0
247899	Acacia	0.0
247900	Chestnut	6.0
247901	Avocado	2.0
247902	Lavender	10.0

247903 rows × 2 columns

3.1.3. Outlier Statistics and Treatment (Scaling | Transformation)

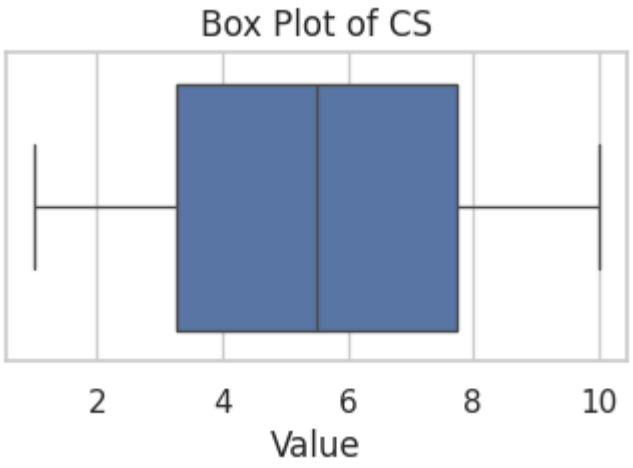
3.1.3.1.1. Outlier Statistics(Non-Categorical Variables or Features) : mag, depth, rms, depthError

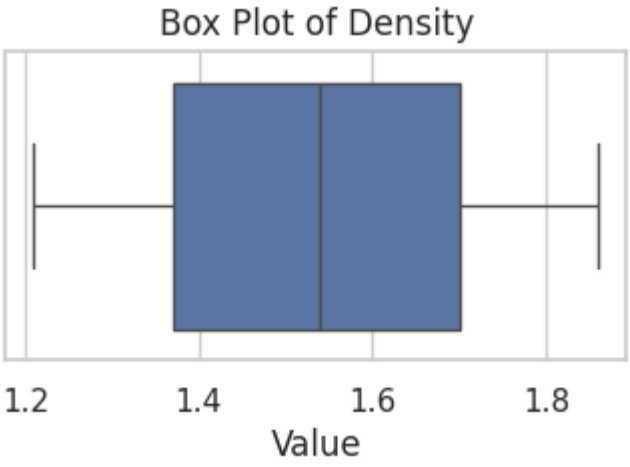
3.1.3.1.2.1 Outlier Treatment: Non-Categorical Variables or Features:

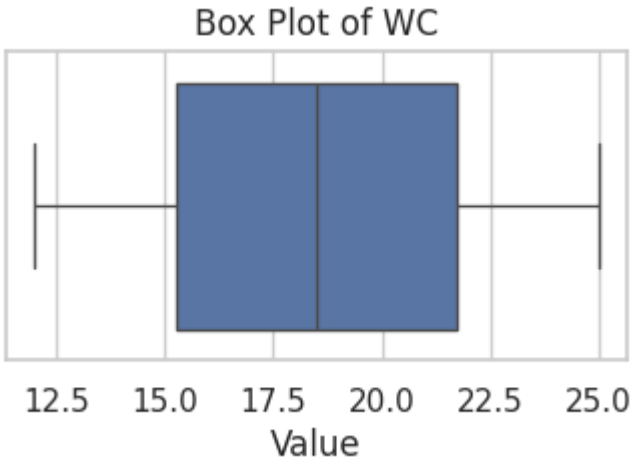
3.1.3.1.2.2 Normalization using Min-Max Scaler: mag, depth, rms, depthError

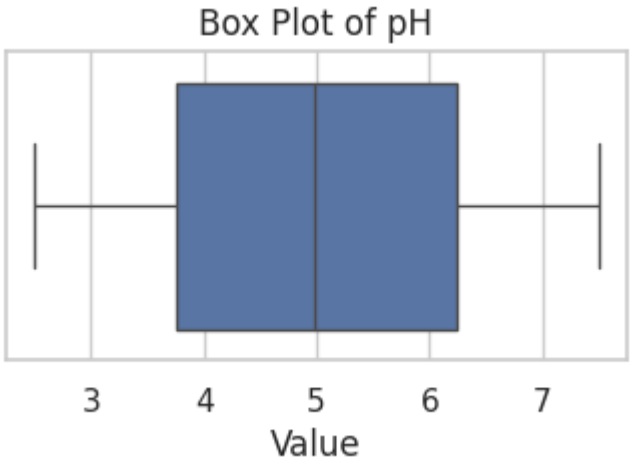
```
# Assuming merged_df_noncat is your DataFrame containing the specified columns
import matplotlib.pyplot as plt
import seaborn as sns

# Columns to include in the box plot
columns = ['CS','Density', 'WC', 'pH', 'EC', 'F', 'G', 'Viscosity', 'Purity', 'Price']

# Create box plots for each column
plt.figure(figsize=(8, 4))
sns.set(style="whitegrid")

for column in columns:
    plt.figure(figsize=(4,2))
    sns.boxplot(data=df_noncat[column], orient="h")
    plt.title(f"Box Plot of {column}")
    plt.xlabel("Value")
    plt.show()
```

```
<Figure size 800x400 with 0 Axes>
```

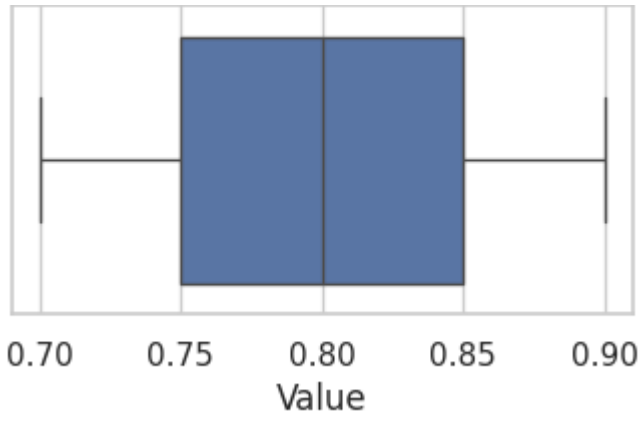


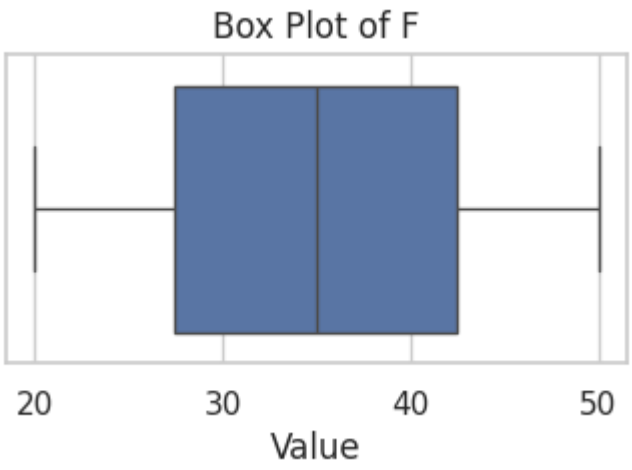


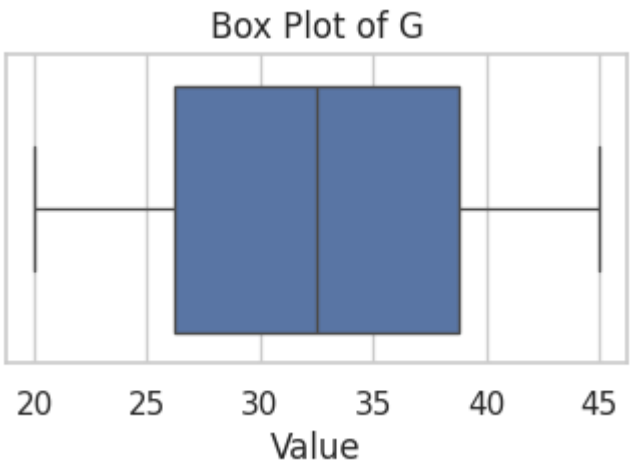


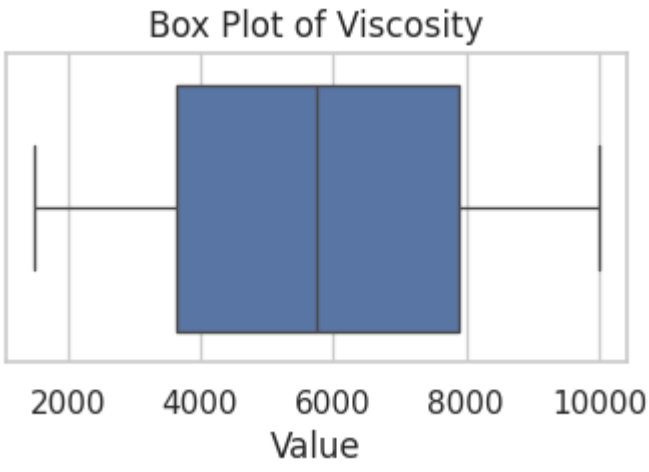


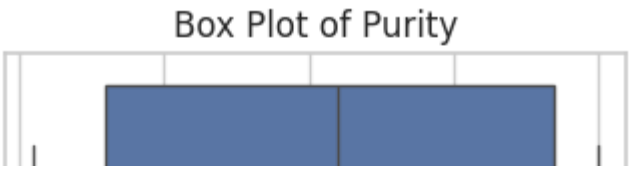
Box Plot of EC

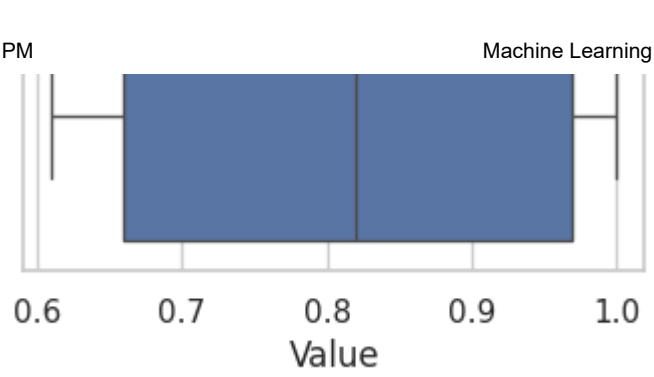

0.70
0.75
0.80
0.85
0.90
Value

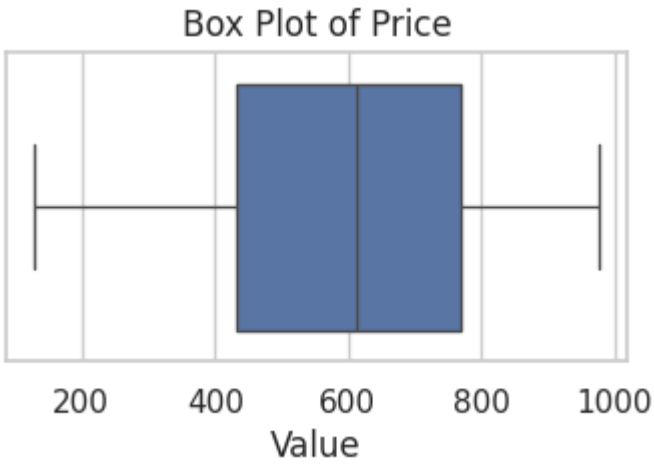

Box Plot of F
20
30
40
50
Value


Box Plot of G
20
25
30
35
40
45
Value


Box Plot of Viscosity
2000
4000
6000
8000
10000
Value


Box Plot of Purity






```
# Create a subset DataFrame containing specific columns
df_noncat_mdt = df[['CS', 'Density', 'WC', 'pH', 'EC', 'F', 'G', 'Viscosity', 'Purity', '

# Display the subset DataFrame
print(df_noncat_mdt.head())
```

```
     CS  Density     WC    pH    EC      F      G  Viscosity  Purity   Price
0  2.81     1.75  23.04  6.29  0.76  39.02  33.63    4844.50    0.68  645.24
1  9.47     1.82  17.50  7.20  0.71  38.15  34.41    6689.02    0.89  385.85
2  4.61     1.84  23.72  7.31  0.80  27.47  34.36    6883.60    0.66  639.64
3  1.77     1.40  16.61  4.01  0.78  31.52  28.15    7167.56    1.00  946.46
4  6.11     1.25  19.63  4.82  0.90  29.65  42.52    5125.44    1.00  432.62
```

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Assuming df_noncat_mdt contains the subset of columns you want to normalize

# Initialize MinMaxScaler
mms = MinMaxScaler()

# Fit and transform the data
mms_fit = mms.fit_transform(df_noncat_mdt)

# Create a DataFrame for the normalized data
df_noncat_minmax_norm = pd.DataFrame(mms_fit, columns=df_noncat_mdt.columns+'_mn')

# Concatenate the original DataFrame with the normalized DataFrame along the columns axis
df_noncat_mdt_mmn = pd.concat([df_noncat_mdt, df_noncat_minmax_norm], axis=1)

# Display the concatenated DataFrame
print(df_noncat_mdt_mmn)
```

```
           CS  Density     WC    pH    EC      F      G  Viscosity  Purity  \
0        2.81     1.75  23.04  6.29  0.76  39.02  33.63    4844.50    0.68
1        9.47     1.82  17.50  7.20  0.71  38.15  34.41    6689.02    0.89
2        4.61     1.84  23.72  7.31  0.80  27.47  34.36    6883.60    0.66
3        1.77     1.40  16.61  4.01  0.78  31.52  28.15    7167.56    1.00
4        6.11     1.25  19.63  4.82  0.90  29.65  42.52    5125.44    1.00
...       ...      ...    ...   ...   ...    ...    ...        ...     ...
247898   1.98     1.29  17.90  4.82  0.89  36.10  34.69    8261.63    1.00
247899   6.18     1.67  19.54  4.91  0.85  31.15  20.82    6939.39    1.00
247900   7.78     1.49  15.78  5.69  0.73  44.60  44.07    4139.79    0.64
247901   5.78     1.74  14.96  6.81  0.83  47.19  37.79    4417.74    0.97
247902   8.96     1.86  18.62  6.89  0.86  25.94  42.88    8119.62    0.64

          Price     CS_mn  Density_mn     WC_mn  pH_mn  EC_mn      F_mn  \
0        645.24  0.201111    0.830769  0.849231  0.758   0.30  0.634000
1        385.85  0.941111    0.938462  0.423077  0.940   0.05  0.605000
2        639.64  0.401111    0.969231  0.901538  0.962   0.50  0.249000
3        946.46  0.085556    0.292308  0.354615  0.302   0.40  0.384000
4        432.62  0.567778    0.061538  0.586923  0.464   1.00  0.321667
...         ...       ...         ...       ...    ...    ...       ...
247898   754.98  0.108889    0.123077  0.453846  0.464   0.95  0.536667
247899   543.41  0.575556    0.707692  0.580000  0.482   0.75  0.371667
247900   615.46  0.753333    0.430769  0.290769  0.638   0.15  0.820000
247901   949.32  0.531111    0.815385  0.227692  0.862   0.65  0.906333
247902   384.48  0.884444    1.000000  0.509231  0.878   0.80  0.198000

           G_mn  Viscosity_mn  Purity_mn  Price_mn
0        0.5452      0.393468   0.179487  0.609125
1        0.5764      0.610473   0.717949  0.303230
2        0.5744      0.633365   0.128205  0.602521
3        0.3260      0.666772   1.000000  0.964350
4        0.9008      0.426520   1.000000  0.358385
...         ...           ...        ...       ...
247898   0.5876      0.795487   1.000000  0.738540
247899   0.0328      0.639928   1.000000  0.489039
247900   0.9628      0.310561   0.076923  0.574006
247901   0.7116      0.343261   0.923077  0.967723
```

```
247902  0.9152      0.778780   0.076923  0.301614

[247903 rows x 20 columns]
```

Pre-Processed Dataset

Missing Data Treated & Transformed or Rescaled Non-Categorical Data Subsets

```
# Pre-Processed Categorical Data Subset
df_cat_ppd = df_cat_mdt_code_oe.copy(); df_cat_ppd
```

	Pollen_analysis_x	Pollen_analysis_y
0	Blueberry	3.0
1	Alfalfa	1.0
2	Chestnut	6.0
3	Blueberry	3.0
4	Alfalfa	1.0
...	...	...
247898	Rosemary	13.0
247899	Acacia	0.0
247900	Chestnut	6.0
247901	Avocado	2.0
247902	Lavender	10.0

247903 rows × 2 columns

```
# Pre-Processed Non-Categorical Data Subset
df_noncat_ppd = df_noncat_mdt_mmn.copy(); df_noncat_ppd
```

	CS	Density	WC	pH	EC	F	G	Viscosity	Purity	Price	CS_r
0	2.81	1.75	23.04	6.29	0.76	39.02	33.63	4844.50	0.68	645.24	0.2011
1	9.47	1.82	17.50	7.20	0.71	38.15	34.41	6689.02	0.89	385.85	0.9411
2	4.61	1.84	23.72	7.31	0.80	27.47	34.36	6883.60	0.66	639.64	0.4011
3	1.77	1.40	16.61	4.01	0.78	31.52	28.15	7167.56	1.00	946.46	0.0855
4	6.11	1.25	19.63	4.82	0.90	29.65	42.52	5125.44	1.00	432.62	0.5677
...	...	...	...	...	...	...	...	...	...	...	
247898	1.98	1.29	17.90	4.82	0.89	36.10	34.69	8261.63	1.00	754.98	0.1088
247899	6.18	1.67	19.54	4.91	0.85	31.15	20.82	6939.39	1.00	543.41	0.5755
247900	7.78	1.49	15.78	5.69	0.73	44.60	44.07	4139.79	0.64	615.46	0.7533
247901	5.78	1.74	14.96	6.81	0.83	47.19	37.79	4417.74	0.97	949.32	0.5311
247902	8.96	1.86	18.62	6.89	0.86	25.94	42.88	8119.62	0.64	384.48	0.8844

247903 rows × 20 columns

```
df_ppd = pd.merge(df_cat_ppd, df_noncat_ppd, left_index=True, right_index=True); df_ppd
```

	Pollen_analysis_x	Pollen_analysis_y	CS	Density	WC	pH	EC	F	
0	Blueberry	3.0	2.81	1.75	23.04	6.29	0.76	39.02	3
1	Alfalfa	1.0	9.47	1.82	17.50	7.20	0.71	38.15	3
2	Chestnut	6.0	4.61	1.84	23.72	7.31	0.80	27.47	3
3	Blueberry	3.0	1.77	1.40	16.61	4.01	0.78	31.52	2
4	Alfalfa	1.0	6.11	1.25	19.63	4.82	0.90	29.65	4
...	...	...	...	...	...	...	...	...	
247898	Rosemary	13.0	1.98	1.29	17.90	4.82	0.89	36.10	3
247899	Acacia	0.0	6.18	1.67	19.54	4.91	0.85	31.15	2
247900	Chestnut	6.0	7.78	1.49	15.78	5.69	0.73	44.60	4
247901	Avocado	2.0	5.78	1.74	14.96	6.81	0.83	47.19	3
247902	Lavender	10.0	8.96	1.86	18.62	6.89	0.86	25.94	4

247903 rows × 22 columns

Data Bifurcation [Training & Testing Datasets]

```
# Dataset Used : df_ppd

train_df, test_df = train_test_split(df_ppd, test_size=0.25, random_state=1234)
#train_df # Training Dataset
#test_df # Testing Dataset
```

Creating a subset for applying clustering

We have choose these columns for clustering : 'Density_mn' and 'pH_mn' Density and pH: The density of Honey is correlated with its pH level. This correlation could arise due to factors such as the concentration of dissolved substances affecting both density and pH.

```
df_ppd.columns
```

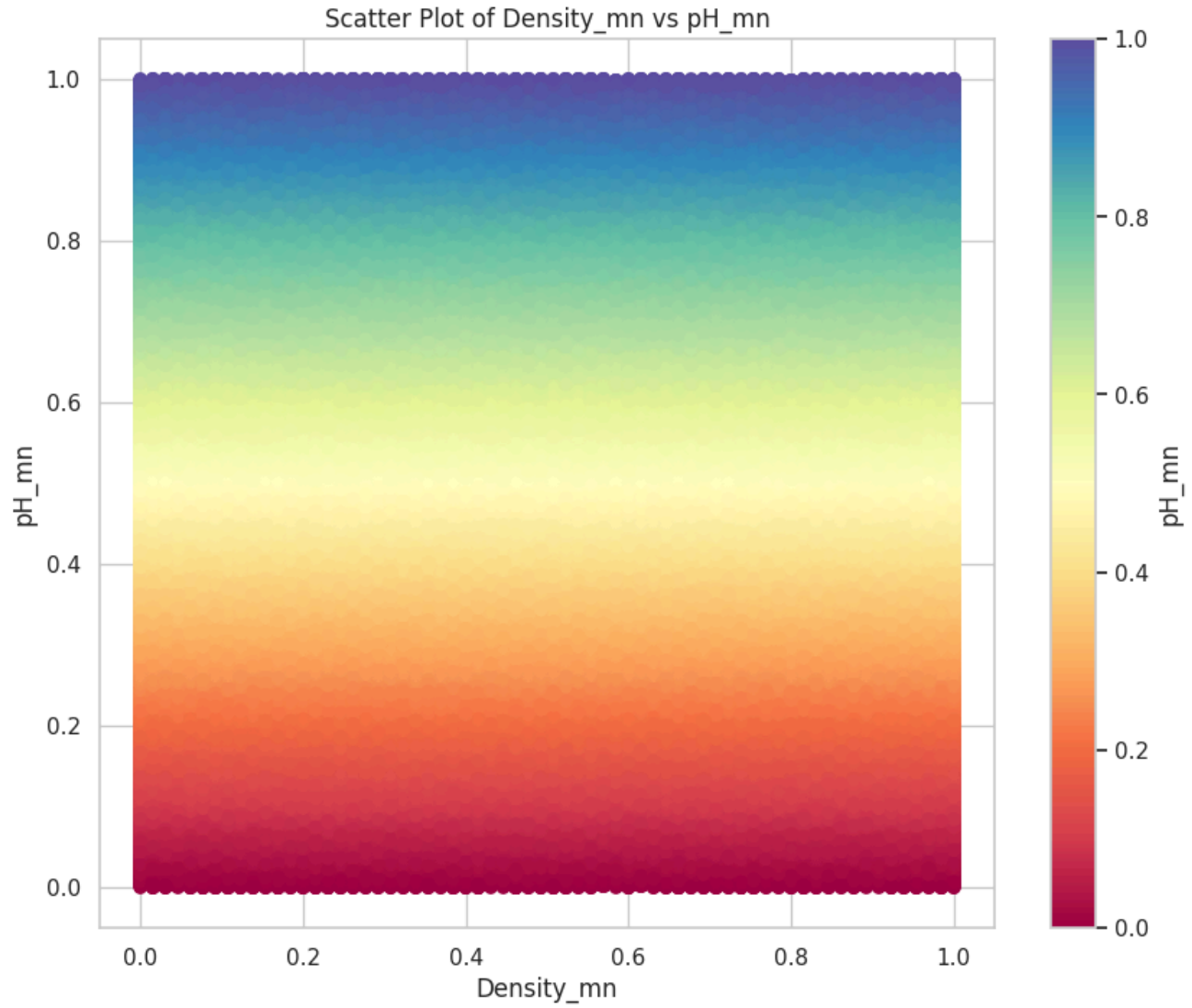
```
Index(['Pollen_analysis_x', 'Pollen_analysis_y', 'CS', 'Density', 'WC', 'pH',
       'EC', 'F', 'G', 'Viscosity', 'Purity', 'Price', 'CS_mn', 'Density_mn',
       'WC_mn', 'pH_mn', 'EC_mn', 'F_mn', 'G_mn', 'Viscosity_mn', 'Purity_mn',
       'Price_mn'],
      dtype='object')
```

```
Honey_Sweetness_subset= df_ppd[[ 'Pollen_analysis_y', 'Viscosity_mn',  'Purity_mn','Price
```

```
('Viscosity_mn',
 'Purity_mn',
 'Price_mn',
 'CS_mn',
 'Density_mn',
 'WC_mn',
 'pH_mn',
 'EC_mn',
 'F_mn',
 'G_mn')
```

Plotting Density vs pH Scatter Plot

```
import matplotlib.pyplot as plt

# Choose two variables from your DataFrame for the scatter plot
x_variable = 'Density_mn'
y_variable = 'pH_mn'

# Set the size of the figure
plt.figure(figsize=(10, 8))  # Adjust the width and height as needed

# Scatter plot with colormap
plt.scatter(x=x_variable, y=y_variable, c=Honey_Sweetness_subset[y_variable], cmap='Spect
plt.xlabel(x_variable)
plt.ylabel(y_variable)
plt.title(f'Scatter Plot of {x_variable} vs {y_variable}')
plt.colorbar(label=y_variable)
plt.grid(True)
plt.show()
```



K-Means Clustering

Determine Value of 'K' in K-Means using Elbow Curve & KMeans-Inertia

3.2. Data Analysis

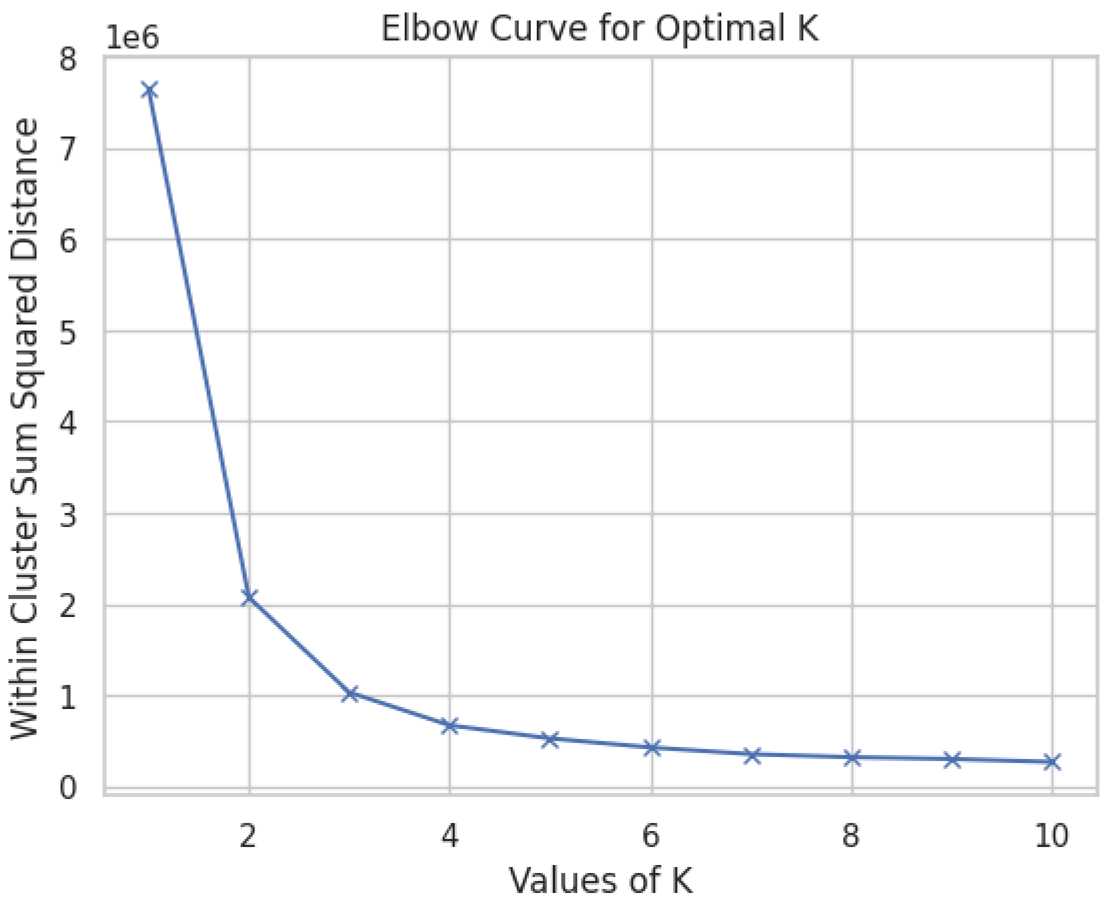
3.2.1.1. PO1 | PS1:: Unsupervised Machine Learning Clustering Algorithm: K-Means (Base Model) | Metrics Used - Euclidean Distance

3.2.1.2. PO1 | PS1:: Unsupervised Machine Learning Clustering Algorithms: {DBSCAN | BIRCH | OPTICS} | Metrics Used - Euclidean Distance

3.2.2.1.1. PO2 | PS2:: Clustering Model Performance Evaluation: Silhouette Score | Davies-Bouldin Score (Base Model: K-Mean)

```
wcssd = []  # Within-Cluster-Sum-Squared-Distance
nr_clus = range(1, 11)  # Number of Clusters

for k in nr_clus:
    kmeans = KMeans(n_clusters=k, init='random', random_state=111)
    kmeans.fit(Honey_Sweetness_subset)
    wcssd.append(kmeans.inertia_)

# Plotting the Elbow Curve
plt.plot(nr_clus, wcssd, marker='x')
plt.xlabel('Values of K')
plt.ylabel('Within Cluster Sum Squared Distance')
plt.title('Elbow Curve for Optimal K')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
```




```
#start time
start= time.time()
```

Create K-Means Clusters

```
km_2cluster = kmclus(n_clusters=2, init='random', random_state=222)
km_2cluster_model = km_2cluster.fit_predict(Honey_Sweetness_subset); print(km_2cluster_mo
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
[1 1 1 ... 1 1 0]
```

```
km_3cluster = kmclus(n_clusters=3, init='random', random_state=333)
km_3cluster_model = km_3cluster.fit_predict(Honey_Sweetness_subset); print(km_3cluster_mo
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
[0 0 0 ... 0 0 2]
```

```
km_4cluster = kmclus(n_clusters=4, init='random', random_state=444)
km_4cluster_model = km_4cluster.fit_predict(Honey_Sweetness_subset); km_3cluster_model
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
array([0, 0, 0, ..., 0, 0, 2], dtype=int32)
```

```
km_5cluster = kmclus(n_clusters=5, init='random', random_state=555)
km_5cluster_model = km_5cluster.fit_predict(Honey_Sweetness_subset); km_3cluster_model
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
array([0, 0, 0, ..., 0, 0, 2], dtype=int32)
```

K-Means Clustering Model Evaluation

```
sscore_km_2cluster = sscore(Honey_Sweetness_subset, km_2cluster_model);
dbscore_km_2cluster = dbscore(Honey_Sweetness_subset, km_2cluster_model);
print("sscore for k=2 : ",sscore_km_2cluster)
print("dbscore for k=2 : ",dbscore_km_2cluster)
```

```
sscore for k=2 :  0.5892663840350901
dbscore for k=2 :  0.5524721874905855
```

```
sscore_km_3cluster = sscore(Honey_Sweetness_subset, km_3cluster_model)
dbscore_km_3cluster = dbscore(Honey_Sweetness_subset, km_3cluster_model)
print("sscore for k=3 : ",sscore_km_3cluster)
print("dbscore for k=3 : ",dbscore_km_3cluster)
```

```
sscore for k=3 :  0.524127910821051
dbscore for k=3 :  0.6033388513915833
```

```
sscore_km_4cluster = sscore(Honey_Sweetness_subset, km_4cluster_model)
dbscore_km_4cluster = dbscore(Honey_Sweetness_subset, km_4cluster_model)
print("sscore for k=4 : ",sscore_km_4cluster)
print("dbscore for k=4 : ",dbscore_km_4cluster)
```

```
sscore for k=4 :  0.4624086910763274
dbscore for k=4 :  0.6845421710000893
```

```
sscore_km_5cluster = sscore(Honey_Sweetness_subset, km_5cluster_model)
dbscore_km_5cluster = dbscore(Honey_Sweetness_subset, km_5cluster_model)
print("sscore for k=5 : ",sscore_km_5cluster)
print("dbscore for k=5 : ",dbscore_km_5cluster)
```

```
sscore for k=5 :  0.4300688399165834
dbscore for k=5 :  0.7325441890933015
```

To determine the best clustering model based on the Davies-Bouldin (DB) score and Silhouette (SS) score, we need to consider the following:

Silhouette Score (SS): A higher Silhouette score indicates better separation between clusters. The Silhouette score ranges from -1 to 1, where a score closer to 1 indicates better clustering.

Davies-Bouldin Score (DB): A lower Davies-Bouldin score indicates better clustering. The DB score measures the average similarity between each cluster and its most similar cluster, where a lower score indicates better separation between clusters.

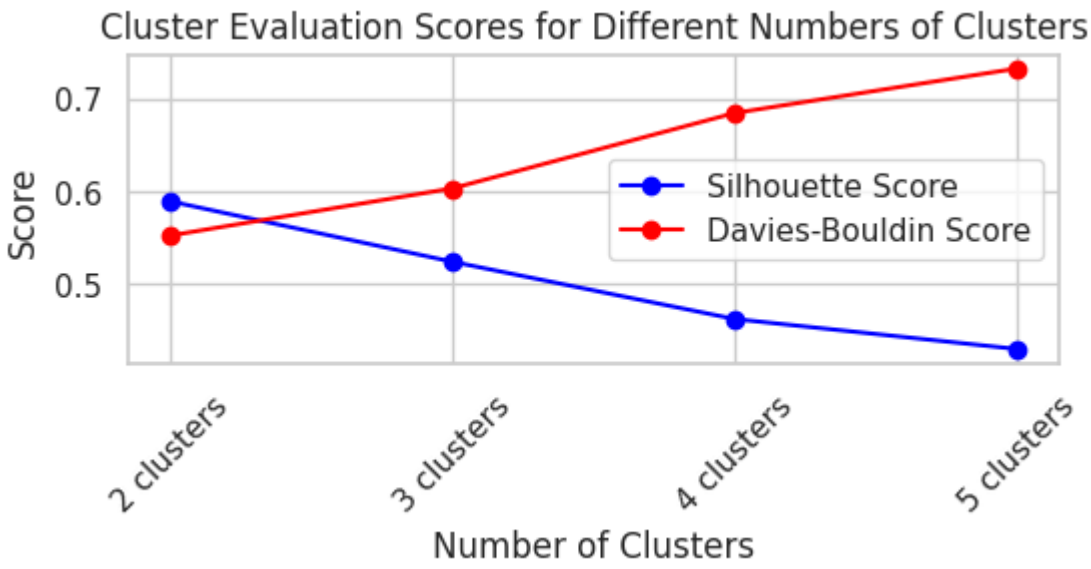
For k=2: SS score is 0.589 and DB score is 0.552. For k=3: SS score is 0.524 and DB score is 0.603. For k=4: SS score is 0.462 and DB score is 0.684. For k=5: SS score is 0.430 and DB score is 0.732.

Since we want to maximize the Silhouette score and minimize the Davies-Bouldin score, the best clustering model would be the one with the highest Silhouette score and the lowest Davies-Bouldin score.

In this case, for k=2, the clustering model has the highest Silhouette score (0.589) and the lowest Davies-Bouldin score (0.552). Therefore, the clustering model with k=2 is likely the best choice based on both the Silhouette and Davies-Bouldin scores.

But we will consider the 3 clustering model as the best model for our clustering subset

```
import matplotlib.pyplot as plt

# Define cluster names and scores
cluster_names = ['2 clusters', '3 clusters', '4 clusters', '5 clusters']
sscore_values = [sscore_km_2cluster, sscore_km_3cluster, sscore_km_4cluster, sscore_km_5c
dbscore_values = [dbscore_km_2cluster, dbscore_km_3cluster, dbscore_km_4cluster, dbscore_

# Plotting both scores on the same graph
plt.figure(figsize=(6, 2))

# Plotting the Silhouette Score
plt.plot(cluster_names, sscore_values, marker='o', color='blue', label='Silhouette Score'

# Plotting the Davies-Bouldin Score
plt.plot(cluster_names, dbscore_values, marker='o', color='red', label='Davies-Bouldin Sc

# Adding title and labels
plt.title('Cluster Evaluation Scores for Different Numbers of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)

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



4. Create a KMeans Cluster Member Dataframe

Cluster Model Used : km_3cluster_model (Based on sscore and dbscore)

3.2.2.1.2 Clustering Model Performance Evaluation of Base Model K-means :

Time Statistics : 780 seconds Memory Statistics : 9777152 bytes

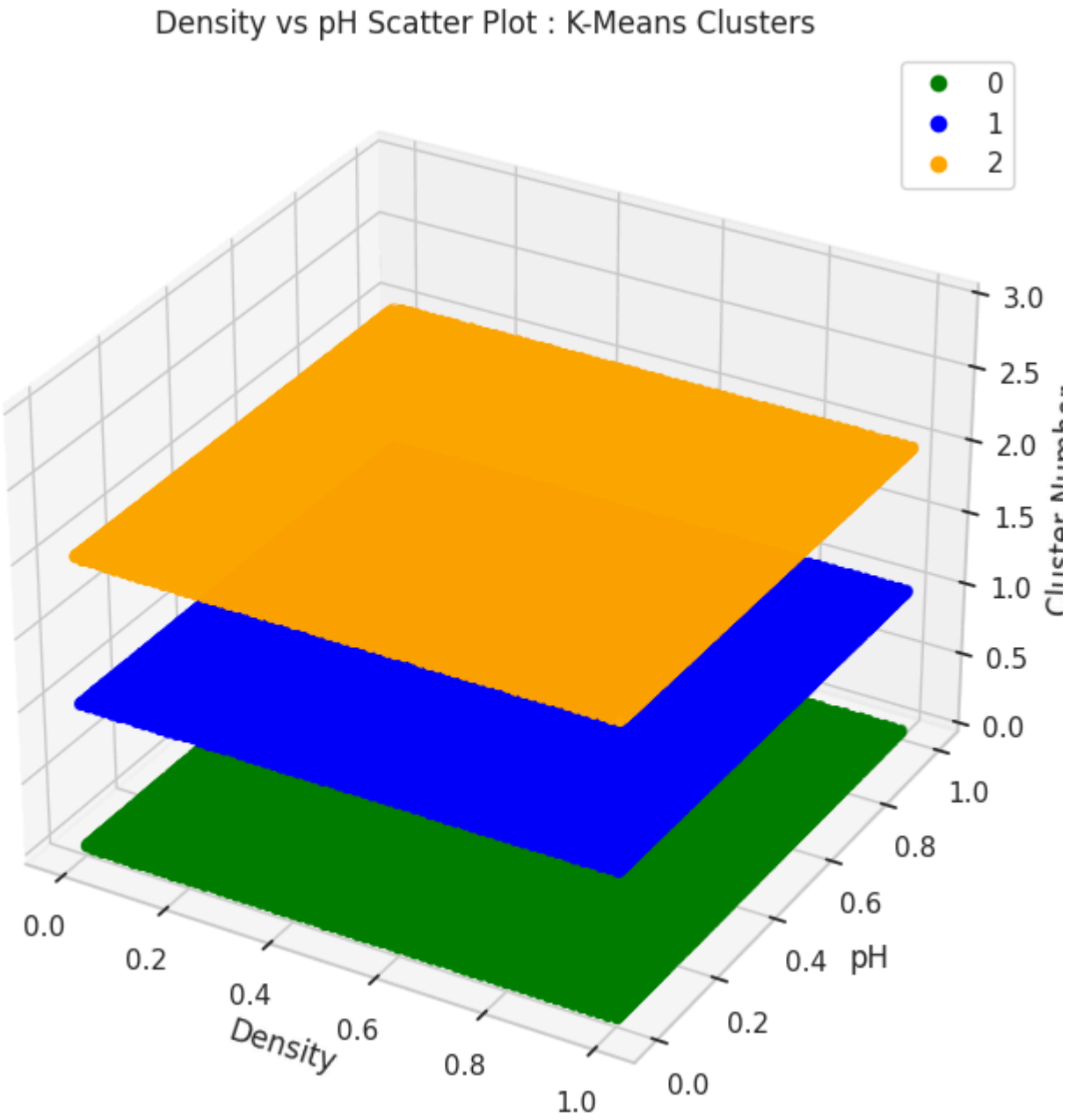
3.2.2.2.1 Clustering Model Performance Evaluation: Silhouette Score | Davies-Bouldin Score (Comparison Models: DBSCAN | BIRCH | OPTICS - At Least One)

3.2.2.2.2 Clustering Model Performance Evaluation: Time Statistics | (CPU | GPU) Memory Statistics (Comparison Models: DBSCAN | BIRCH | OPTICS - At Least One)

```
Honey_Sweetness_subset_kmcluster = Honey_Sweetness_subset.copy()
Honey_Sweetness_subset_kmcluster['cluster_number'] = km_3cluster_model
Honey_Sweetness_subset_kmcluster.sort_values('cluster_number', inplace=True);
print(Honey_Sweetness_subset)
```

```
        Pollen_analysis_y  Viscosity_mn  Purity_mn  Price_mn     CS_mn  \
0                     3.0      0.393468   0.179487  0.609125  0.201111
1                     1.0      0.610473   0.717949  0.303230  0.941111
2                     6.0      0.633365   0.128205  0.602521  0.401111
3                     3.0      0.666772   1.000000  0.964350  0.085556
4                     1.0      0.426520   1.000000  0.358385  0.567778
...                   ...           ...        ...       ...       ...
247898               13.0      0.795487   1.000000  0.738540  0.108889
247899                0.0      0.639928   1.000000  0.489039  0.575556
247900                6.0      0.310561   0.076923  0.574006  0.753333
247901                2.0      0.343261   0.923077  0.967723  0.531111
247902               10.0      0.778780   0.076923  0.301614  0.884444

        Density_mn     WC_mn  pH_mn  EC_mn      F_mn    G_mn
0         0.830769  0.849231  0.758   0.30  0.634000  0.5452
1         0.938462  0.423077  0.940   0.05  0.605000  0.5764
2         0.969231  0.901538  0.962   0.50  0.249000  0.5744
3         0.292308  0.354615  0.302   0.40  0.384000  0.3260
4         0.061538  0.586923  0.464   1.00  0.321667  0.9008
...            ...       ...    ...    ...       ...     ...
247898    0.123077  0.453846  0.464   0.95  0.536667  0.5876
247899    0.707692  0.580000  0.482   0.75  0.371667  0.0328
247900    0.430769  0.290769  0.638   0.15  0.820000  0.9628
247901    0.815385  0.227692  0.862   0.65  0.906333  0.7116
247902    1.000000  0.509231  0.878   0.80  0.198000  0.9152

[247903 rows x 11 columns]
```

```
from matplotlib.colors import ListedColormap  # Import ListedColormap

custom_cmap = ListedColormap(['Green','Blue', 'Orange'])

# Create a larger figure
fig = plt.figure(figsize=(10, 8))  # Increase the figsize as desired

# Add subplot with 3D projection
ax = fig.add_subplot(111, projection='3d')

# Define cluster labels
cluster_labels = list(Honey_Sweetness_subset_kmcluster['cluster_number'].unique())

# Plot the clustered data in 3D
scatter_plot = ax.scatter(xs=Honey_Sweetness_subset_kmcluster['Density_mn'],
                          ys=Honey_Sweetness_subset_kmcluster['pH_mn'],
                          zs=Honey_Sweetness_subset_kmcluster['cluster_number'],  # Use c
                          c=Honey_Sweetness_subset_kmcluster['cluster_number'],
                          cmap=custom_cmap)  # Use custom colormap

# Set labels and title
ax.set_xlabel('Density')
ax.set_ylabel('pH')
ax.set_zlabel('Cluster Number')
ax.set_title('Density vs pH Scatter Plot : K-Means Clusters')

# Add legend
plt.legend(handles=scatter_plot.legend_elements()[0], labels=cluster_labels)

# Set z-axis limits
ax.set_zlim(0, max(Honey_Sweetness_subset_kmcluster['cluster_number']) + 1)  # Adjust as

# Show grid
ax.grid()

# Show plot
plt.show()
```



The data is concentrated in two areas. There is a cluster of points with high density and low ph (around a ph of 0.2 and a density of 1.0) and another cluster of points with low density and high ph (around a ph of 0.8 and a density of 0.2).

```
# end time
end = time.time()

# total time taken
print(f"Runtime of the program is {end - start}")
```

```
     Runtime of the program is 3627.7358164787292
```

Splitting the dataframe based on cluster

```
Honey_Purity_cluster0= Honey_Sweetness_subset_kmcluster[Honey_Sweetness_subset_kmcluster[
Honey_Purity_cluster1= Honey_Sweetness_subset_kmcluster[Honey_Sweetness_subset_kmcluster[
Honey_Purity_cluster2= Honey_Sweetness_subset_kmcluster[Honey_Sweetness_subset_kmcluster[
```

```
Honey_Purity_cluster0.describe()
```

	Pollen_analysis_y	Viscosity_mn	Purity_mn	Price_mn	CS_mn	Der
count	91378.000000	91378.000000	91378.000000	91378.000000	91378.000000	9137
mean	2.999070	0.501637	0.548606	0.583444	0.501024	
std	1.997226	0.289325	0.357871	0.234687	0.288078	
min	0.000000	0.000000	0.000000	0.159251	0.000000	
25%	1.000000	0.250688	0.128205	0.382325	0.252222	
50%	3.000000	0.503025	0.538462	0.581200	0.501111	
75%	5.000000	0.753334	0.923077	0.763671	0.750000	
max	6.000000	0.999993	1.000000	1.000000	1.000000	

```
Honey_Purity_cluster1.describe()
```

	Pollen_analysis_y	Viscosity_mn	Purity_mn	Price_mn	CS_mn	Der
count	78153.000000	78153.000000	78153.000000	78153.000000	78153.000000	7815
mean	15.496449	0.499848	0.551194	0.616688	0.499444	
std	1.704360	0.288520	0.356892	0.251123	0.288157	
min	13.000000	0.000021	0.000000	0.079614	0.000000	
25%	14.000000	0.250759	0.128205	0.521469	0.250000	
50%	15.000000	0.499974	0.538462	0.651202	0.500000	
75%	17.000000	0.751292	0.923077	0.811456	0.747778	
max	18.000000	0.999966	1.000000	0.952463	1.000000	

```
Honey_Purity_cluster2.describe()
```

	Pollen_analysis_y	Viscosity_mn	Purity_mn	Price_mn	CS_mn	Der
count	78372.000000	78372.000000	78372.000000	78372.000000	78372.000000	7837
mean	9.498316	0.499316	0.550201	0.443401	0.499451	
std	1.703399	0.288823	0.357606	0.309684	0.288437	
min	7.000000	0.000001	0.000000	0.000000	0.000000	
25%	8.000000	0.249555	0.128205	0.097185	0.250000	
50%	9.000000	0.497759	0.538462	0.499192	0.498889	
75%	11.000000	0.749464	0.923077	0.667252	0.748889	
max	12.000000	1.000000	1.000000	1.000000	1.000000	

3.2.3.1 Cluster Analysis: Base Model (K-Means)

3.2.3.1.1 Cluster Analysis with Categorical Variables or Features: Chi-Square Test of Independence and Anova

```
from scipy.stats import chi2_contingency

# Extracting categorical variables
cat_variables = ['Pollen_analysis_y']

# Perform Chi-square test for each categorical variable
chi2_results_cat = {}
for column in cat_variables:
    contingency_table = pd.crosstab(Honey_Sweetness_subset_kmcluster[column], km_2cluster
    chi2, p_value, _, _ = chi2_contingency(contingency_table)
    chi2_results_cat[column] = {'Chi-square': chi2, 'P-value': p_value}

# Print Chi-square test results for categorical variables
for column, result in chi2_results_cat.items():
    print(f"Variable: {column}")
    print(f"Chi-square: {result['Chi-square']}")
    print(f"P-value: {result['P-value']}")
    print()
```

```
Variable: Pollen_analysis_y
Chi-square: 22.766160952888026
P-value: 0.19973656660957187
```

```
# ANOVA Using Scipy
import scipy.stats as sps # For Probability & Inferential Statistics
anova_lat = sps.f_oneway(Honey_Purity_cluster0.Viscosity_mn,Honey_Purity_cluster1.Viscosi
anova_lat
```

```
F_onewayResult(statistic=1.5264208151689465, pvalue=0.2173141195413315)
```

```
# ANOVA Using Scipy
import scipy.stats as sps # For Probability & Inferential Statistics
anova_lat = sps.f_oneway(Honey_Purity_cluster0.Purity_mn,Honey_Purity_cluster1.Purity_mn,
anova_lat
```

```
F_onewayResult(statistic=1.1381084115185085, pvalue=0.3204262344889583)
```

```
# ANOVA Using Scipy
import scipy.stats as sps # For Probability & Inferential Statistics
anova_lat = sps.f_oneway(Honey_Purity_cluster0.Price_mn ,Honey_Purity_cluster1.Price_mn ,
anova_lat
```

```
F_onewayResult(statistic=9505.474779164037, pvalue=0.0)
```

```
# ANOVA Using Scipy
import scipy.stats as sps # For Probability & Inferential Statistics
anova_lat = sps.f_oneway(Honey_Purity_cluster0.CS_mn    ,Honey_Purity_cluster1.CS_mn    ,
anova_lat
```

```
F_onewayResult(statistic=0.8626214432203476, pvalue=0.4220555066431553)
```

```
# ANOVA Using Scipy
import scipy.stats as sps # For Probability & Inferential Statistics
anova_lat = sps.f_oneway(Honey_Purity_cluster0.Density_mn   ,Honey_Purity_cluster1.Densit
anova_lat
```

```
F_onewayResult(statistic=0.11369795884046076, pvalue=0.8925275417880955)
```

```
# ANOVA Using Scipy
import scipy.stats as sps # For Probability & Inferential Statistics
anova_lat = sps.f_oneway(Honey_Purity_cluster0.WC_mn    ,Honey_Purity_cluster1.WC_mn    ,
anova_lat
```

```
F_onewayResult(statistic=0.17384772889808808, pvalue=0.8404249628302121)
```

```
# ANOVA Using Scipy
import scipy.stats as sps # For Probability & Inferential Statistics
anova_lat = sps.f_oneway(Honey_Purity_cluster0.pH_mn ,Honey_Purity_cluster1.pH_mn   ,Hone
anova_lat
```

```
F_onewayResult(statistic=1.091540502243663, pvalue=0.3357005635507791)
```

```
# ANOVA Using Scipy
import scipy.stats as sps # For Probability & Inferential Statistics
anova_lat = sps.f_oneway(Honey_Purity_cluster0.EC_mn ,Honey_Purity_cluster1.EC_mn   ,Hone
anova_lat
```

```
F_onewayResult(statistic=1.3207594730751875, pvalue=0.2669343751837084)
```

```
# ANOVA Using Scipy
import scipy.stats as sps # For Probability & Inferential Statistics
anova_lat = sps.f_oneway(Honey_Purity_cluster0.F_mn  ,Honey_Purity_cluster1.F_mn ,Honey_P
anova_lat
```

```
F_onewayResult(statistic=1.2022392906960773, pvalue=0.30052225722702625)
```

```
# ANOVA Using Scipy
import scipy.stats as sps # For Probability & Inferential Statistics
anova_lat = sps.f_oneway(Honey_Purity_cluster0.G_mn ,Honey_Purity_cluster1.G_mn ,Honey_Pu
anova_lat
```

```
F_onewayResult(statistic=0.8009867649399011, pvalue=0.4488869625858618)
```

```
!pip install scikit-learn
from sklearn.cluster import Birch
model = Birch(branching_factor = 50, n_clusters = 2, threshold = 1.5)
# Fit the model to the data
pred=model.fit(Honey_Sweetness_subset)
# Get cluster labels
cluster_labels2 = pred.labels_
# Get cluster labels
cluster_labels2
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist
array([0, 0, 0, ..., 0, 0, 0])
```

```
model = Birch(branching_factor = 50, n_clusters = 3, threshold = 1.5)
# Fit the model to the data
pred=model.fit(Honey_Sweetness_subset)
# Get cluster labels
cluster_labels3 = pred.labels_
# Get cluster labels
cluster_labels3
```

```
array([2, 2, 0, ..., 0, 2, 0])
```

```
model = Birch(branching_factor = 50, n_clusters = 4, threshold = 1.5)
# Fit the model to the data
pred=model.fit(Honey_Sweetness_subset)
# Get cluster labels
cluster_labels4 = pred.labels_
# Get cluster labels
cluster_labels4
```

```
array([2, 2, 3, ..., 3, 2, 1])
```

```
model = Birch(branching_factor = 50, n_clusters = 5, threshold = 1.5)
# Fit the model to the data
pred=model.fit(Honey_Sweetness_subset)
# Get cluster labels
cluster_labels5 = pred.labels_
# Get cluster labels
cluster_labels5
```

```
array([0, 0, 3, ..., 3, 0, 1])
```

```
# Compute silhouette score
ss2= sscore(Honey_Sweetness_subset,cluster_labels2)
print("Silhouette Score:", ss2)
# Compute Davies-Bouldin score
db2 = dbscore(Honey_Sweetness_subset,cluster_labels2)
print("Davies-Bouldin Score:", db2)
```

```
# Compute silhouette score
ss3= sscore(Honey_Sweetness_subset,cluster_labels3)
print("Silhouette Score:", ss3)
# Compute Davies-Bouldin score
db3 = dbscore(Honey_Sweetness_subset,cluster_labels3)
print("Davies-Bouldin Score:", db3)
```

```
# Compute silhouette score
ss4= sscore(Honey_Sweetness_subset,cluster_labels4)
print("Silhouette Score:", ss4)
# Compute Davies-Bouldin score
db4 = dbscore(Honey_Sweetness_subset,cluster_labels3)
print("Davies-Bouldin Score:", db4)
```

```
# Compute silhouette score
ss5= sscore(Honey_Sweetness_subset,cluster_labels5)
print("Silhouette Score:", ss5)
# Compute Davies-Bouldin score
db5 = dbscore(Honey_Sweetness_subset,cluster_labels5)
print("Davies-Bouldin Score:", db5)
```

```
import psutil

# Get memory usage statistics
memory_stats = psutil.virtual_memory()

# Print memory statistics
print("Total Memory:", memory_stats.total)  # Total physical memory available
print("Available Memory:", memory_stats.available)  # Available memory
print("Used Memory:", memory_stats.used)  # Used memory
print("Free Memory:", memory_stats.free)  # Free memory
```