Faculty of Computer and Data Sciences, Alexandria University.
Course Title: Data Mining & Analytics 02-24-00204
Department: Intelligent Systems (2nd year)

Data Mining - E-commerce Dataset for Predictive Marketing EDA and Clustering Analysis

Project Team

ID	NAME
2203147	أحمد ايهاب عبدالحليم عبده عجمي
22011615	مروان طارق امبابي رزق ابراهيم
2203163	يوسف محمد حلمي أحمد
22010467	أدهم معتز محمد عبدالمنعم
22012048	فارس محمد أحمد أحمد سالم

Team Members' Roles

TASKS	NAME
Problem Statement – Method Evaluation & Visualization	مروان طارق امبابي رزق ابراهيم
Dataset Deployment – Data Preprocessing	أدهم معتز محمد عبدالمنعم
Data Preprocessing – Data Correlation	فارس محمد أحمد أحمد سالم
K-medoids Clustering	يوسف محمد حلمي أحمد
Hierarchical Clustering	أحمد ايهاب عبدالحليم عبده عجمي

Introduction & Problem Statement

In this project, we will use unsupervised machine learning to categorize clients based on features of their purchase behavior. Predictive marketing may help businesses by finding clients that have similar demands or respond similarly to marketing activity. We can also help businesses in determining suitable categories to direct their targeted marketing campaigns.

Hunter's e-grocery is a well-known online shopping brand. Unexpected events such as Covid-19, the Ukraine crisis, and the gas scarcity have all had an influence on the purchasing behavior of clients. Therefore, using Clustering techniques and Principal Component Analysis (PCA) for dimensionality reduction, we will develop propositions for predictive marketing to target customers more accurately.

Importing Libraries

Importing Used Libraries

```python
import numpy as np
import pandas as pd
from sklearn_extra.cluster import KMedoids
from scipy.cluster.hierarchy import dendrogram,linkage,fcluster
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from sklearn.metrics import davies_bouldin_score
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style('darkgrid')
```
[2] ✓ 2.6s Python

- "numpy" for numerical computations.
- "Pandas" for data manipulation and analysis.
- "sklearn_extra.cluster.KMedoids" for K-Medoids clustering.
- "scipy.cluster.hierarchy" for hierarchical clustering.
- "sklearn.preprocessing.StandardScaler" for standardizing features.
- "sklearn.decomposition.PCA" for principal component analysis (PCA).
- "sklearn.metrics" for evaluation metrics such as silhouette score and Davies-Bouldin score.
- "Seaborn" and "matplotlib.pyplot" for data visualization.

Dataset importing and description

Loading Original Dataset

```python
original_data = pd.read_csv("ECommerce_consumer behaviour.csv")
```
[3] ✓ 1.0s Python

Dataset Info & Statistics

```python
original_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2019501 entries, 0 to 2019500
Data columns (total 12 columns):
 #   Column                  Dtype
---  ------                  -----
 0   order_id                int64
 1   user_id                 int64
 2   order_number            int64
 3   order_dow               int64
 4   order_hour_of_day       int64
 5   days_since_prior_order  float64
 6   product_id              int64
 7   add_to_cart_order       int64
 8   reordered               int64
 9   department_id           int64
 10  department              object
 11  product_name            object
dtypes: float64(1), int64(9), object(2)
memory usage: 184.9+ MB
```

Print information about data like (data type, index range,……) using "info"

```python
original_data.describe()
```

	order_id	user_id	order_number	order_dow	order_hour_of_day	days_since_prior_order	product_id	add_to_cart_order	reordered	department_id
count	2.019501e+06	2.019501e+06	2.019501e+06	2.019501e+06	2.019501e+06	1.895159e+06	2.019501e+06	2.019501e+06	2.019501e+06	2.019501e+06
mean	1.707013e+06	1.030673e+05	1.715138e+01	2.735367e+00	1.343948e+01	1.138603e+01	7.120590e+01	8.363173e+00	5.897427e-01	9.928349e+00
std	9.859832e+05	5.949117e+04	1.752576e+01	2.093882e+00	4.241008e+00	8.970980e+00	3.820727e+01	7.150059e+00	4.918804e-01	6.282933e+00
min	1.000000e+01	2.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	0.000000e+00	1.000000e+00
25%	8.526490e+05	5.158400e+04	5.000000e+00	1.000000e+00	1.000000e+01	5.000000e+00	3.100000e+01	3.000000e+00	0.000000e+00	4.000000e+00
50%	1.705004e+06	1.026900e+05	1.100000e+01	3.000000e+00	1.300000e+01	8.000000e+00	8.300000e+01	6.000000e+00	1.000000e+00	9.000000e+00
75%	2.559031e+06	1.546000e+05	2.400000e+01	5.000000e+00	1.600000e+01	1.500000e+01	1.070000e+02	1.100000e+01	1.000000e+00	1.600000e+01
max	3.421080e+06	2.062090e+05	1.000000e+02	6.000000e+00	2.300000e+01	3.000000e+01	1.340000e+02	1.370000e+02	1.000000e+00	2.100000e+01

Print statistical information about data like (mean, standard deviation, minimum, maximum,…..) using "describe"

Data Preprocessing

Handling Duplicates

Checking Duplicate Values

```python
print(f"The number of duplicate entries: {original_data.duplicated().sum()}")
```

```
The number of duplicate entries: 0
```

Check for duplicate values by founding sum of duplicate values in data using "duplicated" and find sum of it using "sum"

Handling Missing Values

Checking Missing Values

```python
# Sum of null values
missing_values = original_data.isnull().values.sum()
print("Total number of missing values: ",missing_values)

# Percentage of null values
missing_percentage = (original_data.isnull().sum() / len(original_data)) * 100
print("Percentage of missing values for training data:")
print(missing_percentage)

# Highest column percentage of null values
highest_missing_percentage = missing_percentage.idxmax()
print(f"The column with the highest missing values percentage is: {highest_missing_percentage}")
```

[7] ✓ 0.2s Python

```
Total number of missing values:  124342
Percentage of missing values for training data:
order_id                  0.000000
user_id                   0.000000
order_number              0.000000
order_dow                 0.000000
order_hour_of_day         0.000000
days_since_prior_order    6.157066
product_id                0.000000
add_to_cart_order         0.000000
reordered                 0.000000
department_id             0.000000
department                0.000000
product_name              0.000000
dtype: float64
The column with the highest missing values percentage is: days_since_prior_order
```

Check for missing values by checking if there are null values in data using "isnull" and find sum of the values using "sum", then find percentage of missing values in every column in data by finding sum of null values divided by data length and multiplying by 100, finally print column with the most percentage of missing values using "idxmax" to get maximum value.

As we can see only "days_since_prior_order" has missing values with a percentage of 6.157% so it is the column with the highest number of missing values.

Handling Missing values

```python
original_data['days_since_prior_order']=original_data['days_since_prior_order'].fillna('-1')
original_data['days_since_prior_order']=original_data['days_since_prior_order'].astype(int)

# Sum of null values
missing_values = original_data.isnull().values.sum()
print("Total number of missing values: ",missing_values)

# Percentage of null values
missing_percentage = (original_data.isnull().sum() / len(original_data)) * 100
print("Percentage of missing values for training data:")
print(missing_percentage)
```

[8] ✓ 0.3s Python

```
Total number of missing values:  0
Percentage of missing values for training data:
order_id                  0.0
user_id                   0.0
order_number              0.0
order_dow                 0.0
order_hour_of_day         0.0
days_since_prior_order    0.0
product_id                0.0
add_to_cart_order         0.0
reordered                 0.0
department_id             0.0
department                0.0
product_name              0.0
dtype: float64
```

Next, we need to handle missing values in "days_since_prior_order" so, we will replace null values with any other value like -1 using "fillna" and make sure that our column's data type is integer using "astype" and check again if there are any missing values left, since there are no more left so we will go to next step.

Handling Outliers

Identifying and Removing outliers

```python
numerical_columns = original_data.select_dtypes(include=['int64']).columns

def remove_outliers_IQR(original_data, numerical_columns, threshold=1.5):
    for col in numerical_columns:
        q1 = original_data[col].quantile(0.25)
        q3 = original_data[col].quantile(0.75)
        IQR = q3 - q1
        lower_bound = q1 - threshold * IQR
        upper_bound = q3 + threshold * IQR
        outliers_mask = (original_data[col] < lower_bound) | (original_data[col] > upper_bound)
        original_data = original_data[~outliers_mask].reset_index(drop=True)
    return original_data

# Applying the function to remove outliers
new_data = remove_outliers_IQR(original_data, numerical_columns)

# Displaying number of outliers removed from each numerical column
for col in numerical_columns:
    outliers_removed = len(original_data[col]) - len(new_data[col])
    print(f"Number of outliers removed in {col}: {outliers_removed}")
```

[9] ✓ 0.9s   Python

```
Number of outliers removed in order_id: 181310
Number of outliers removed in user_id: 181310
Number of outliers removed in order_number: 181310
Number of outliers removed in order_dow: 181310
Number of outliers removed in order_hour_of_day: 181310
Number of outliers removed in product_id: 181310
Number of outliers removed in add_to_cart_order: 181310
Number of outliers removed in reordered: 181310
Number of outliers removed in department_id: 181310
```

Next is removing outliers, first we need to select numerical columns from our data using "select_dtype" and choose data type (int) for integers, then we will make function to identify and removing outliers that will take three arguments (data, selected numerical columns from data and threshold=1.5 as default).

For each numerical column, the function will calculate the first quartile (q1), third quartile (q3), and interquartile range (IQR).

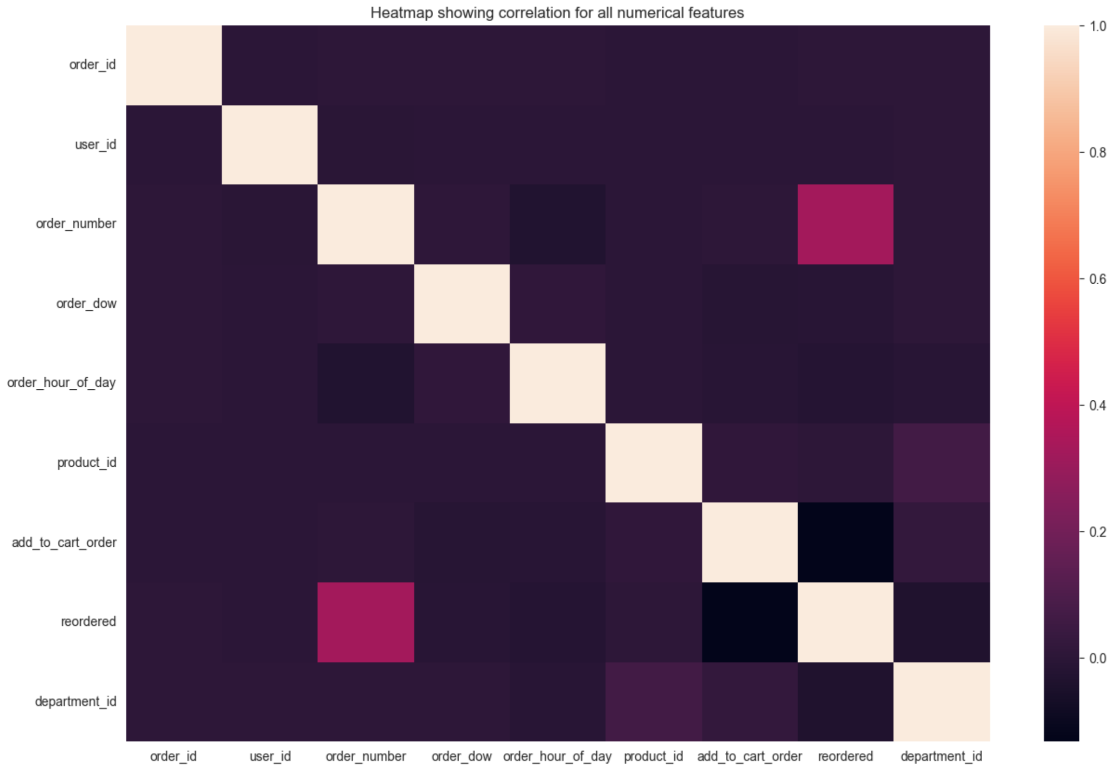
Then, we will defines lower and upper bounds based on the IQR and the threshold value.

Using these bounds, we will create a mask (outliers_mask) to identify rows where the data values are considered outliers (unneeded data).

Then, update data by removing outliers from it and reset index using "reset_index" to drop outlier index from data and return our updated data

Then, applying function and display outlier rows that removed from data.

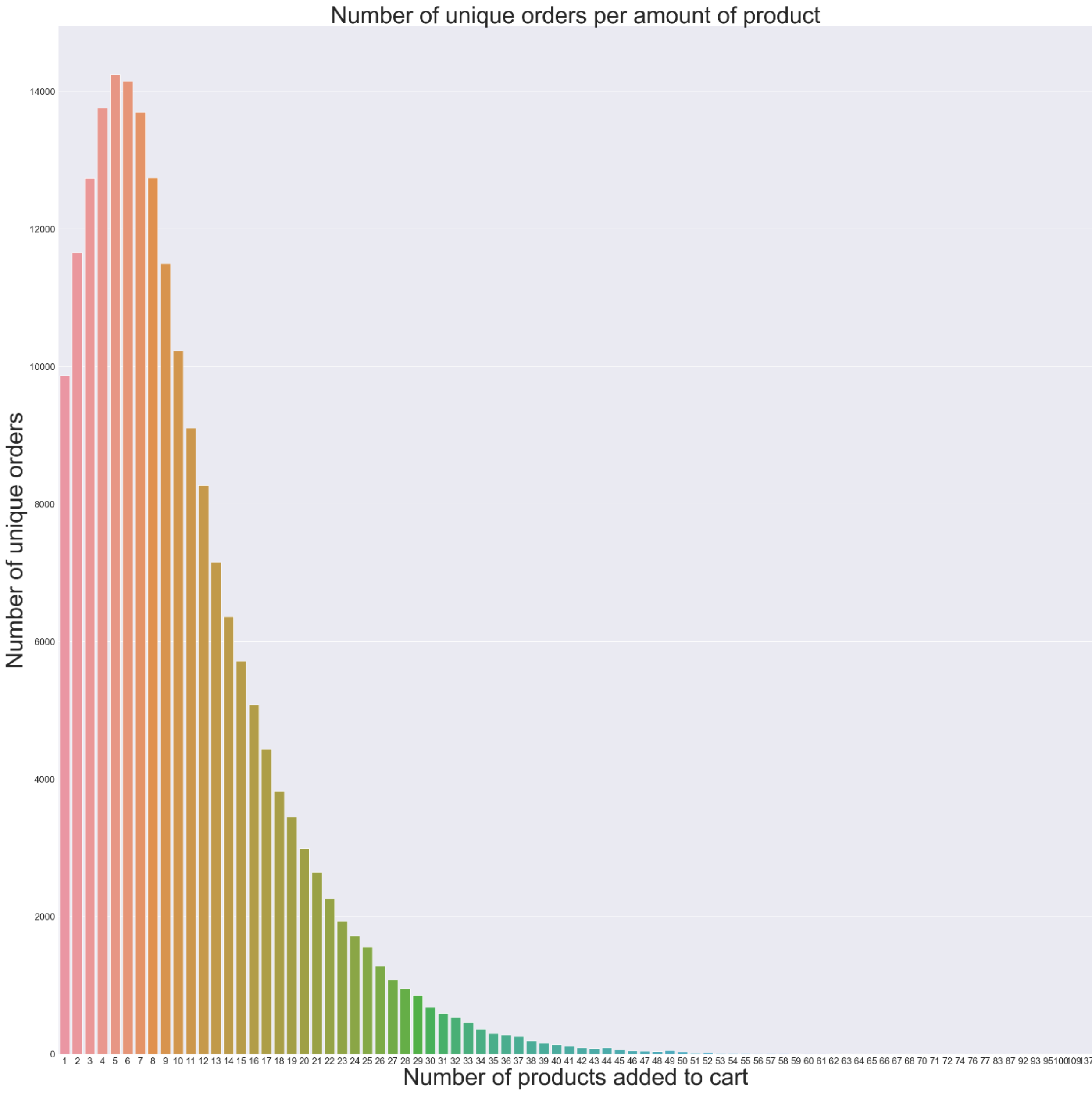
Exploratory Data Analysis (EDA)



Next is computing correlation between numerical features using “corr” and plot result using “heatmap”.

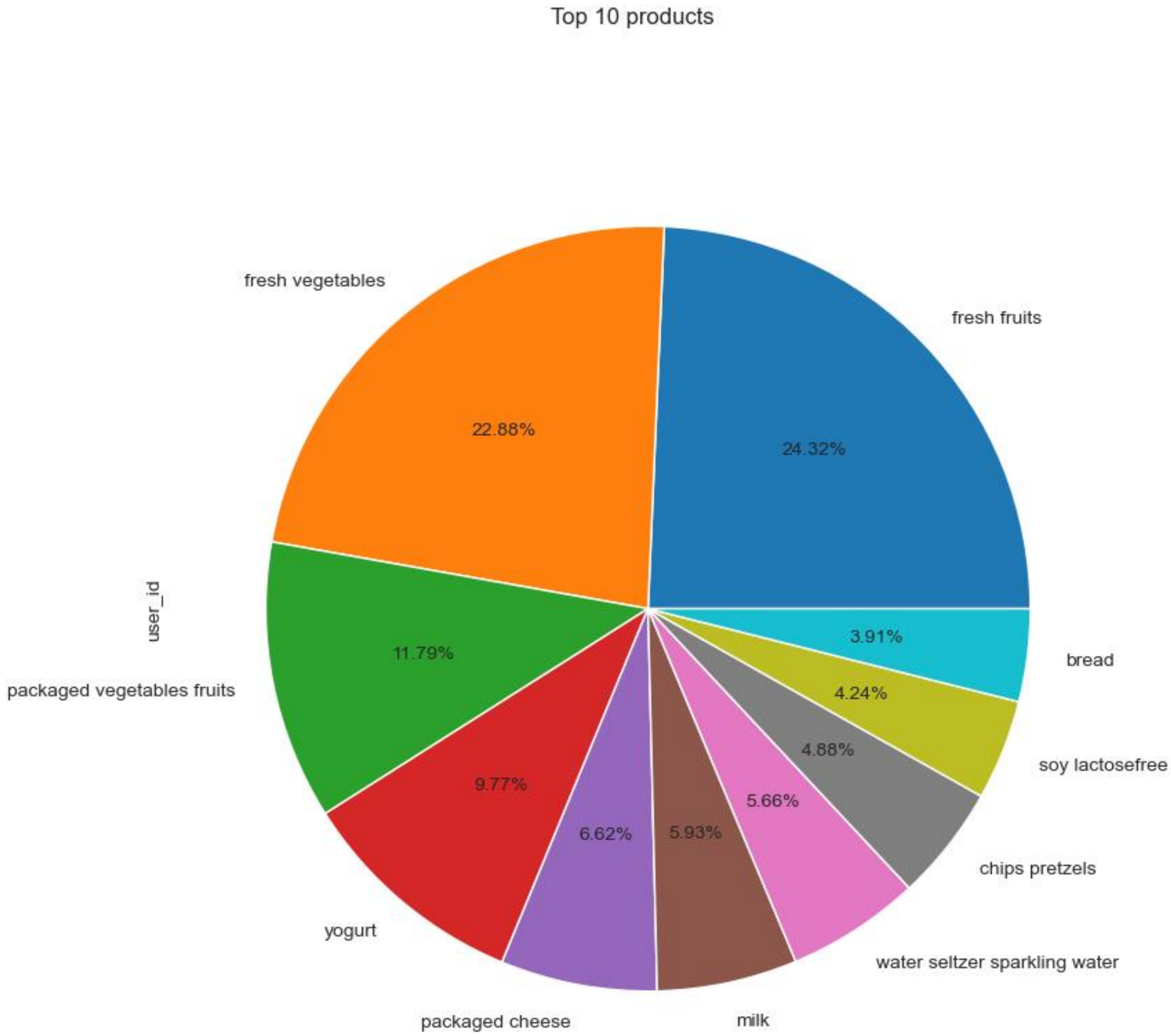
Identifying Hidden Patterns

```
In [9]: # Number of orders for each number of products added to cart
        grouped = original_data.groupby("order_id")["add_to_cart_order"].aggregate("max").reset_index()
        grouped = grouped.add_to_cart_order.value_counts()

        plt.subplots(figsize=(40, 40))
        sns.barplot(x=grouped.index, y=grouped.values)

        plt.title("Number of unique orders per amount of product", fontsize=50)
        plt.ylabel('Number of unique orders', fontsize=50)
        plt.xlabel('Number of products added to cart', fontsize=50)
        plt.yticks(fontsize=20)
        plt.xticks(fontsize=20)
        plt.show()
```



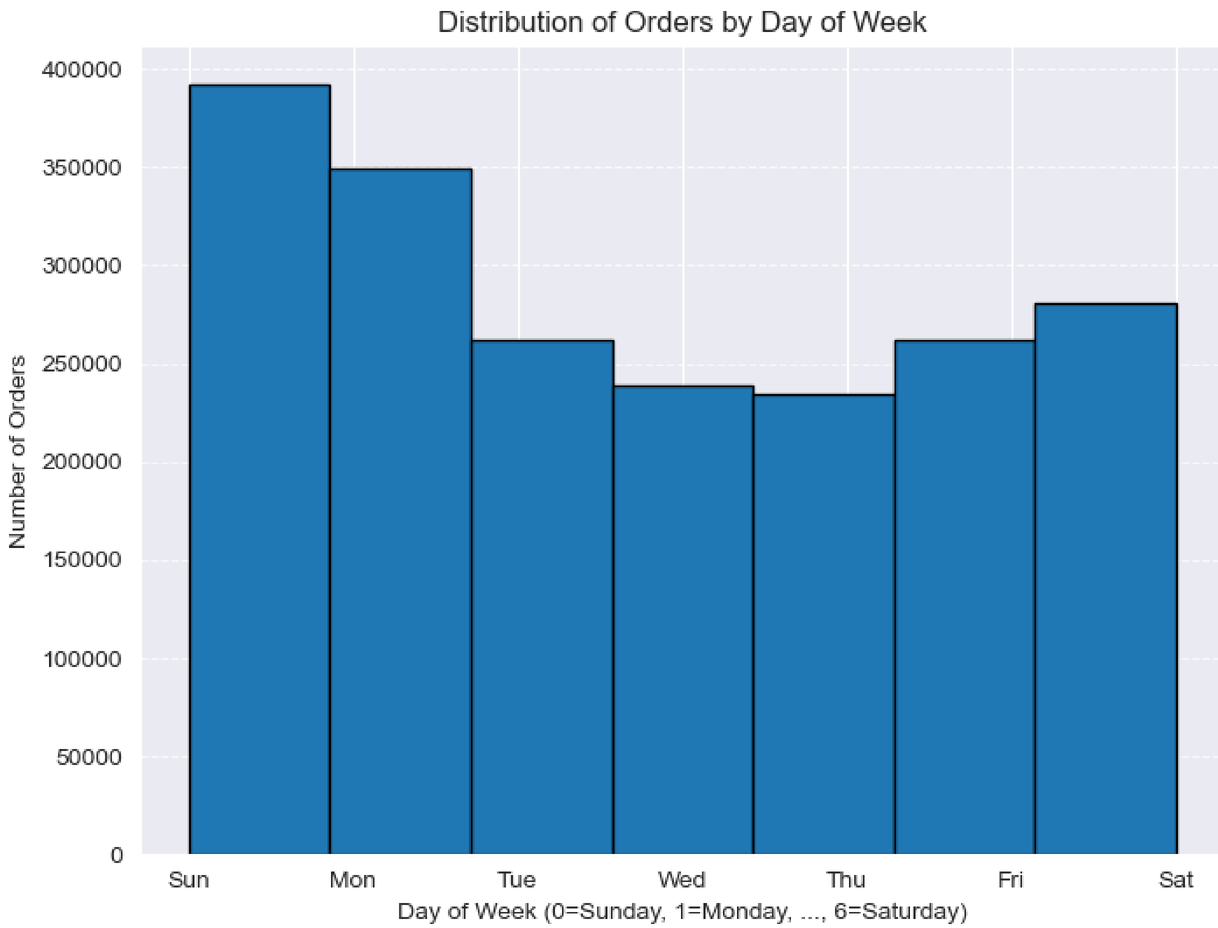
We plot a relationship between the number of products added and the number of unique orders. We first group the data by order ID and find the maximum number of products added to cart in each order. After that, we create a bar plot showing the count of orders for each unique number of products added to cart.

```
In [10]: # Pie Chart of the top 10 most purchased products
         product_counts = original_data.groupby('product_name')['user_id'].count()
         top_10_products = product_counts.sort_values(ascending=False).head(10)
         top_10_products.plot(kind='pie', autopct='%1.2f%%', subplots=True, title='Top 10 products', figsize=(9, 9))
```



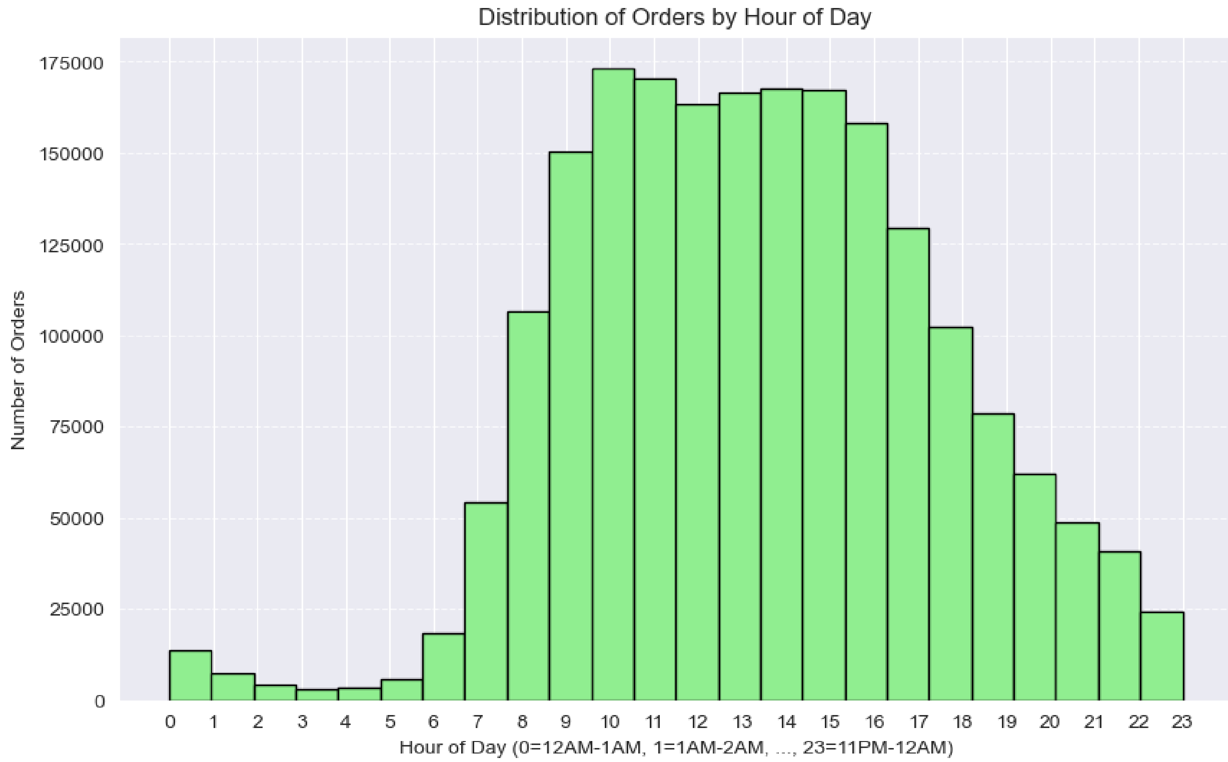
Here, we graph the top 10 most purchased products using a pie chart to show the portions of each category. We notice that fresh fruits and vegetables are at the top of the list.

```
In [11]: # Number of orders by day of the week
         plt.figure(figsize=(8, 6))
         plt.hist(original_data['order_dow'], bins=7, edgecolor='black')
         plt.title('Distribution of Orders by Day of Week')
         plt.xlabel('Day of Week (0=Sunday, 1=Monday, ..., 6=Saturday)')
         plt.ylabel('Number of Orders')
         plt.xticks(range(7), ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'])
         plt.grid(axis='y', linestyle='--', alpha=0.7)
         plt.show()
```



Here, we plot a histogram distribution of the number of orders per day of the week, we notice that peak days are on the weekends (Saturday, Sunday) and low days are in the middle of the workweek (Tuesday, Wednesday, etc.)

```
In [12]: # Number of orders by hour of day
         plt.figure(figsize=(10, 6))
         plt.hist(original_data['order_hour_of_day'], bins=24, color='lightgreen', edgecolor='black')
         plt.title('Distribution of Orders by Hour of Day')
         plt.xlabel('Hour of Day (0=12AM-1AM, 1=1AM-2AM, ..., 23=11PM-12AM)')
         plt.ylabel('Number of Orders')
         plt.xticks(range(0, 24))
         plt.grid(axis='y', linestyle='--', alpha=0.7)
         plt.show()
```



Similarly, we plot the distribution of the number of orders per hour of the day, we also notice that peak hours are in the morning through noon (9AM-4PM) and fall drastically at midnight and through the night as we'd expect.

Scaling Numerical Features

Performing Standard Scaler

```python
scaler = StandardScaler()
features_to_scale = [column for column in new_data[numerical_columns]]
new_data[features_to_scale] = pd.DataFrame(scaler.fit_transform(new_data[features_to_scale]), columns=features_to_scale)
```

[12] ✓ 0.2s Python

Then, we need to scale numerical columns to standardize features by using "StandardScaler" that removes the mean and scaling to unit variance.

So, we need to select scaler (StandardScaler) and select only numerical columns from data and transform features using scaler and update data.

Encoding Categorical Features

Encoding Categorical Features

```python
new_data=pd.get_dummies(new_data)
```

[13] ✓ 0.3s Python

Next is encoding using “get_dummies” that use One-hot encoding technique that used to convert categorical variables into binary vectors. Each category becomes a new binary column (dummy variable), where 1 indicates the presence of that category and 0 indicates absence.

Sample Selection

Taking a sample from the dataset

```python
# Set a random seed for reproducibility
np.random.seed(42)
new_data=new_data.sample(n=100)
```

[14] ✓ 0.0s Python

Taking a sample of 100 rows out of the dataset.

Dimensionality Reduction

Performing PCA

```python
pca = PCA(n_components=2)  # Choosing 2D for visualization
new_data = pca.fit_transform(new_data)
```

[15] ✓ 0.0s Python

PCA is technique that used to reduce the dimensionality by finding the principal components (linear combinations of the original features) that capture the most variance in the data

So, if we choose “n_components”=2 then it will reduce the dimensionality of data to 2 dimensions which will allow us to visualize the data in a 2D plot.

K-medoids Clustering

Applying K_medoids Algorithm

```python
k = 2
K_medoids = KMedoids(k).fit(new_data)
clusters = K_medoids.cluster_centers_
labels = K_medoids.labels_
print('labels : ', labels, '\n')
print('clusters : ', clusters, '\n')

# Evaluate clustering performance using silhouette score
k_medoids_silhouette = silhouette_score(new_data, labels)
print("Silhouette Score:", k_medoids_silhouette)

# Evaluate clustering performance using davies_bouldin_score
k_medoids_davies_bouldin = davies_bouldin_score(new_data, labels)
print("davies_bouldin:", k_medoids_davies_bouldin)
```

[16] ✓ 0.0s Python

```
labels :  [1 1 0 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1
 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0
 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1]

clusters :  [[16.16908475  0.23822558]
 [-4.88509454  0.11200762]]

Silhouette Score: 0.7140696333428328
davies_bouldin: 0.3751872722533399
```

Next is applying K_medoids Algorithm, first we need to select number of clusters(K), 2 is chosen based on the findings of the evaluations performed.

Then fit our algorithm our data using “KMedoids” that take number of cluster(K).

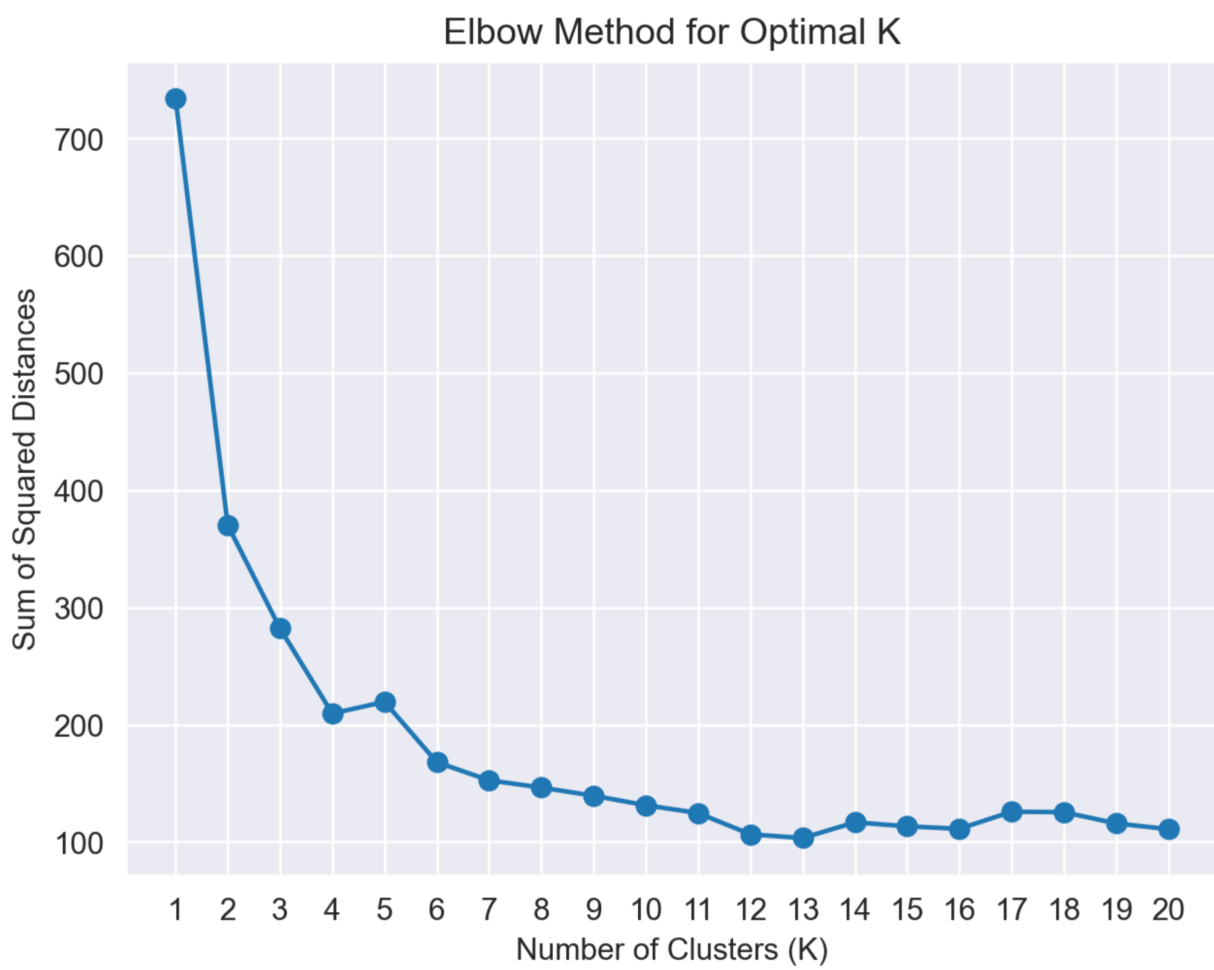
After fitting data we will retrieve the cluster centers using “K_medoids.cluster_centers_” and cluster labels using “K_medoids.labels_” and print them, then evaluate clustering performance using “silhouette_score” that measures how similar an object is to its own cluster compared to other clusters. Higher silhouette scores indicate better clustering and using “davies_bouldin_score” that quantifies the average "similarity" between clusters and the "dissimilarity" between clusters. Lower Davies-Bouldin scores indicate better clustering.

Elbow Method

Using the Elbow Method for finding the optimal number of clusters (K)

```python
k_values = range(1, 21)  # Evaluate from 1 to 20 clusters

# Calculate the Within-Clusters Sum of Squares (WCSS) for each value of K
costs = []
for k in k_values:
    kmedoids = KMedoids(n_clusters=k, random_state=0)
    kmedoids.fit(new_data)
    costs.append(kmedoids.inertia_)

# Plot the elbow curve
plt.rcParams['figure.dpi'] = 227
plt.plot(k_values, costs, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Sum of Squared Distances')
plt.title('Elbow Method for Optimal K')
plt.xticks(k_values)
plt.show()
```

[17] ✓ 0.3s Python



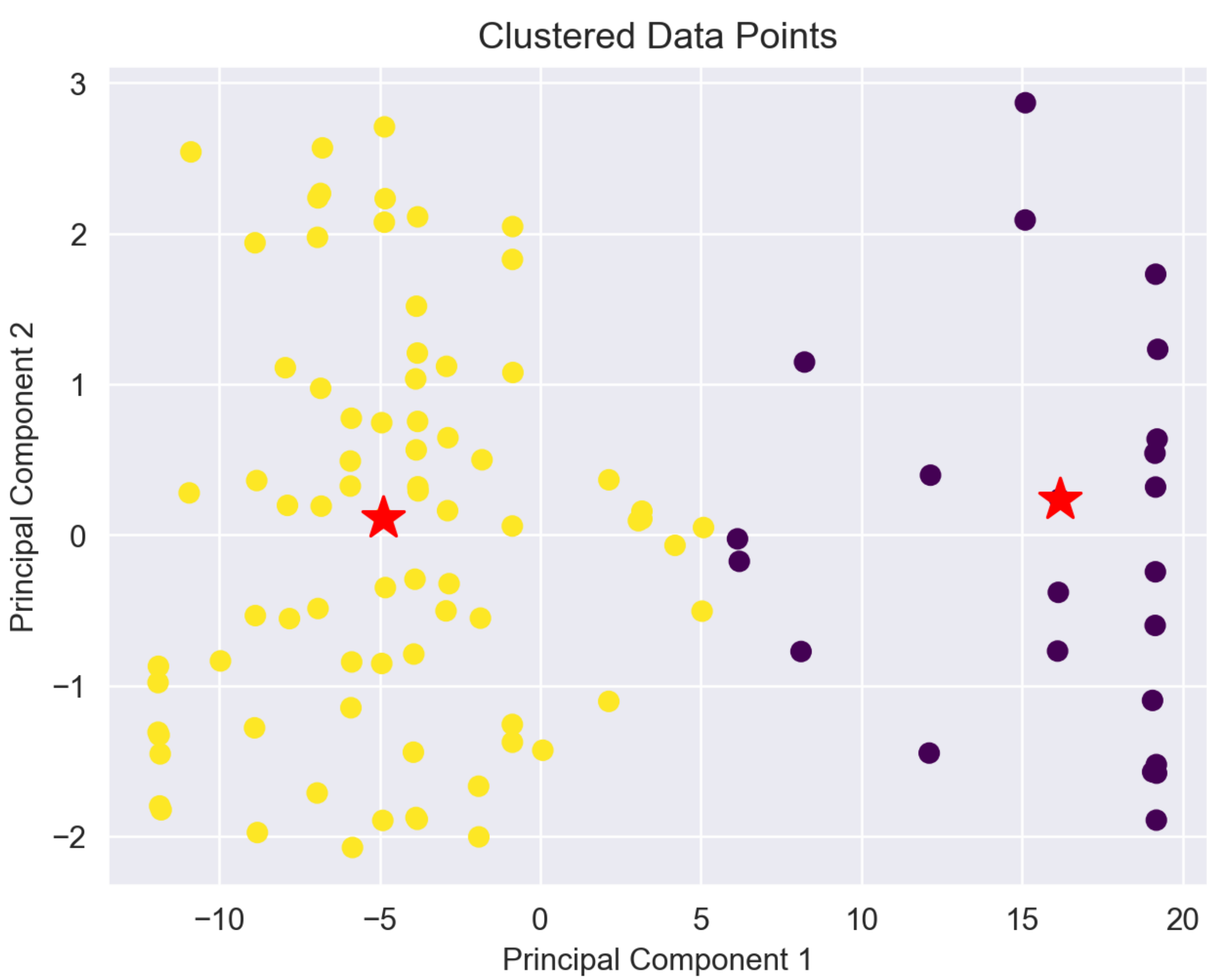
Then, we need to make sure that 2 is the optimal number of clusters to train our data so we will use "Elbow method"

First, we will choose range of K to try from 1 to 20 and make empty list to save costs then we will apply K_medoids Algorithm to every number of clusters in range add “inertia_” to cost list which is Within-Clusters Sum of Squares (WCSS) that is the value for the clustering result, which represents the sum of squared distances of samples to their closest cluster center then plot the result and as we can see the range of the elbow after which the slope has minimal change can be chosen from k=2, 3, 4.

Plotting the Produced Clusters

```
# Plot clustered data points
plt.scatter(new_data[:, 0], new_data[:, 1], c=labels, cmap='viridis')
plt.scatter(clusters[:, 0], clusters[:, 1], c='red', marker='*', s=200)  # Plot cluster centroids
plt.title('Clustered Data Points')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

[18] ✓ 0.1s Python

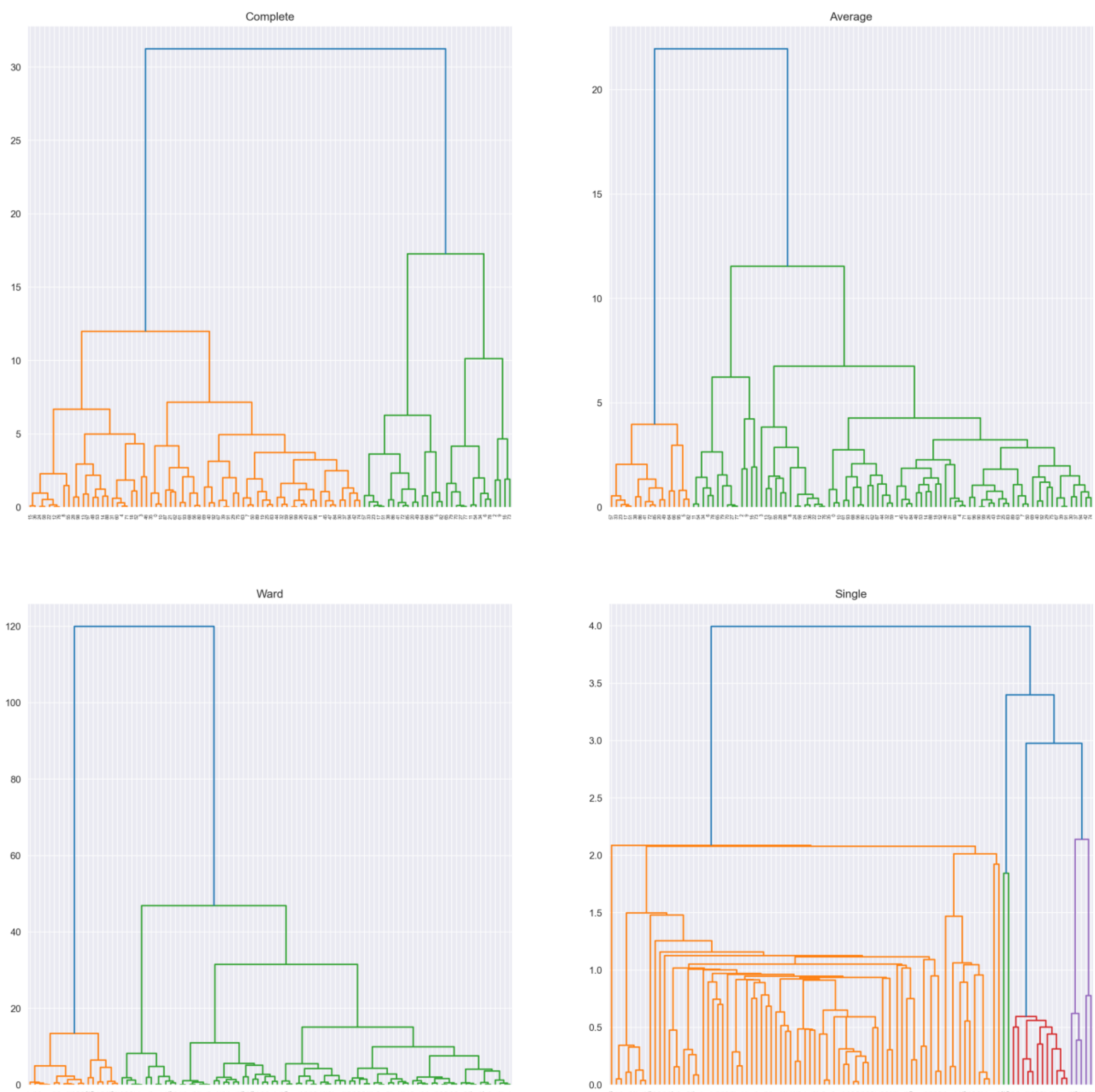


Hierarchical Clustering

Applying Hierachical Clustering

```python
plt.figure(figsize=(20,20))

z_complete=linkage(new_data, method='complete', metric='euclidean')
z_average=linkage(new_data, method='average', metric='euclidean')
z_ward=linkage(new_data, method='ward', metric='euclidean')
z_single=linkage(new_data, method='single', metric='euclidean')

plt.subplot(2,2,1), dendrogram(z_complete), plt.title('Complete')
plt.subplot(2,2,2), dendrogram(z_average), plt.title('Average')
plt.subplot(2,2,3), dendrogram(z_ward), plt.title('Ward')
plt.subplot(2,2,4), dendrogram(z_single), plt.title('Single')
plt.show()
```

[19] ✓ 3.9s Python



Next is applying Hierarchical Clustering algorithm using “linkage” so we will put (data, method and metric), for method we will use “single” that considers the minimum distance between clusters when deciding how to merge them, “complete” that considers the maximum distance between clusters when deciding how to merge them, ”average” that calculates the average distance between all pairs of points in different clusters and “ward” that minimizes the variance when merging clusters, aiming to minimize the increase in variance after merging and for metric we will use “Euclidean” distance then plot every dendrogram.

Plotting every Linkage Method's Silhouette Score over clusters (K)

```python
def plot_silhouette_over_k(Z):
    # Calculate silhouette scores for different numbers of clusters
    silhouette_scores = []
    heights = np.unique(Z[:, 2])
    for height in np.unique(Z[:, 2]):
        # Extract cluster labels based on current height
        labels = fcluster(Z, height, criterion='distance')
        n_clusters = len(np.unique(labels))
        if n_clusters > 1:  # Ensure at least two clusters
            # Compute silhouette score
            silhouette_scores.append(silhouette_score(new_data, labels))
        else:
            silhouette_scores.append(-1)

    # Interpolate silhouette scores for all unique heights
    interp_silhouette_scores = np.interp(np.unique(Z[:, 2]), heights, silhouette_scores)

    # Find the index of maximum silhouette score
    max_silhouette_idx = np.argmax(interp_silhouette_scores)
    max_silhouette_height = np.unique(Z[:, 2])[max_silhouette_idx]
    max_silhouette_score = interp_silhouette_scores[max_silhouette_idx]

    # Plot silhouette scores
    plt.plot(np.unique(Z[:, 2]), interp_silhouette_scores, marker='o', label='Silhouette Scores')
    plt.xlabel('Height')
    plt.ylabel('Silhouette Score')

    plt.scatter(max_silhouette_height, max_silhouette_score, color='red', s=100, label=f'Max Silhouette Score: {max_silhouette_score:.2f}')

    plt.legend()
```

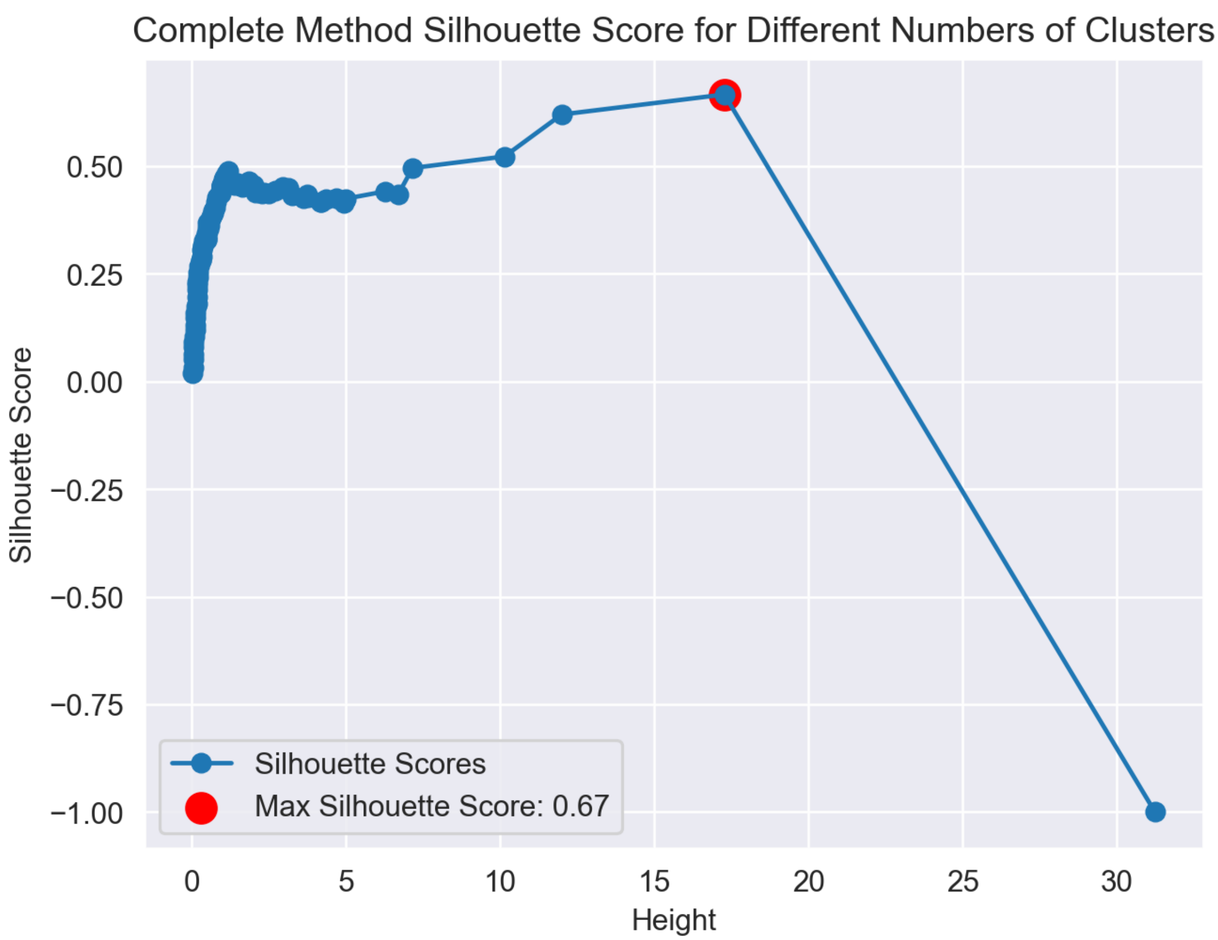
[21] 0.0s Python

Then we will make a function that calculates silhouette scores for different numbers of clusters by iterating over unique heights in the linkage matrix Z.
For each height, it extracts cluster labels based on that height using the “fcluster” function with the criterion='distance' parameter.
Silhouette scores are computed using “silhouette_score”
Since silhouette scores are calculated for unique heights, the function interpolates the scores for all unique heights using “interp” This step ensures plot without missing values.
Then we will find the index and value of the maximum silhouette score among the interpolated scores to identify the optimal number of clusters or height for clustering.

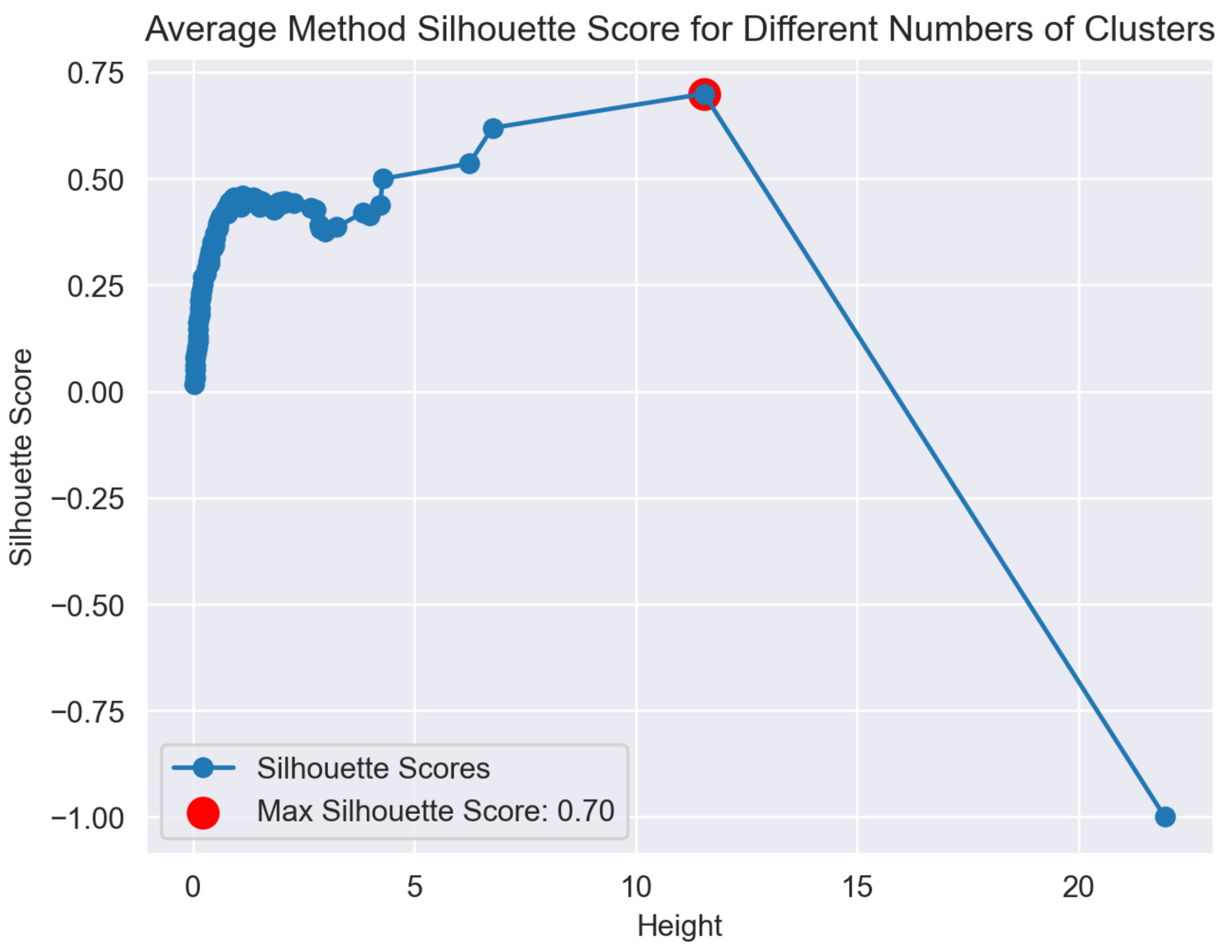
Then plot the silhouette scores against heights, the maximum silhouette score is highlighted with a red dot and labeled with the corresponding height and score.

Evaluation Comparison across types of linkage

```python
plot_silhouette_over_k(z_complete)
plt.title('Complete Method Silhouette Score for Different Numbers of Clusters')
plt.show()
```
[22] ✓ 0.5s Python

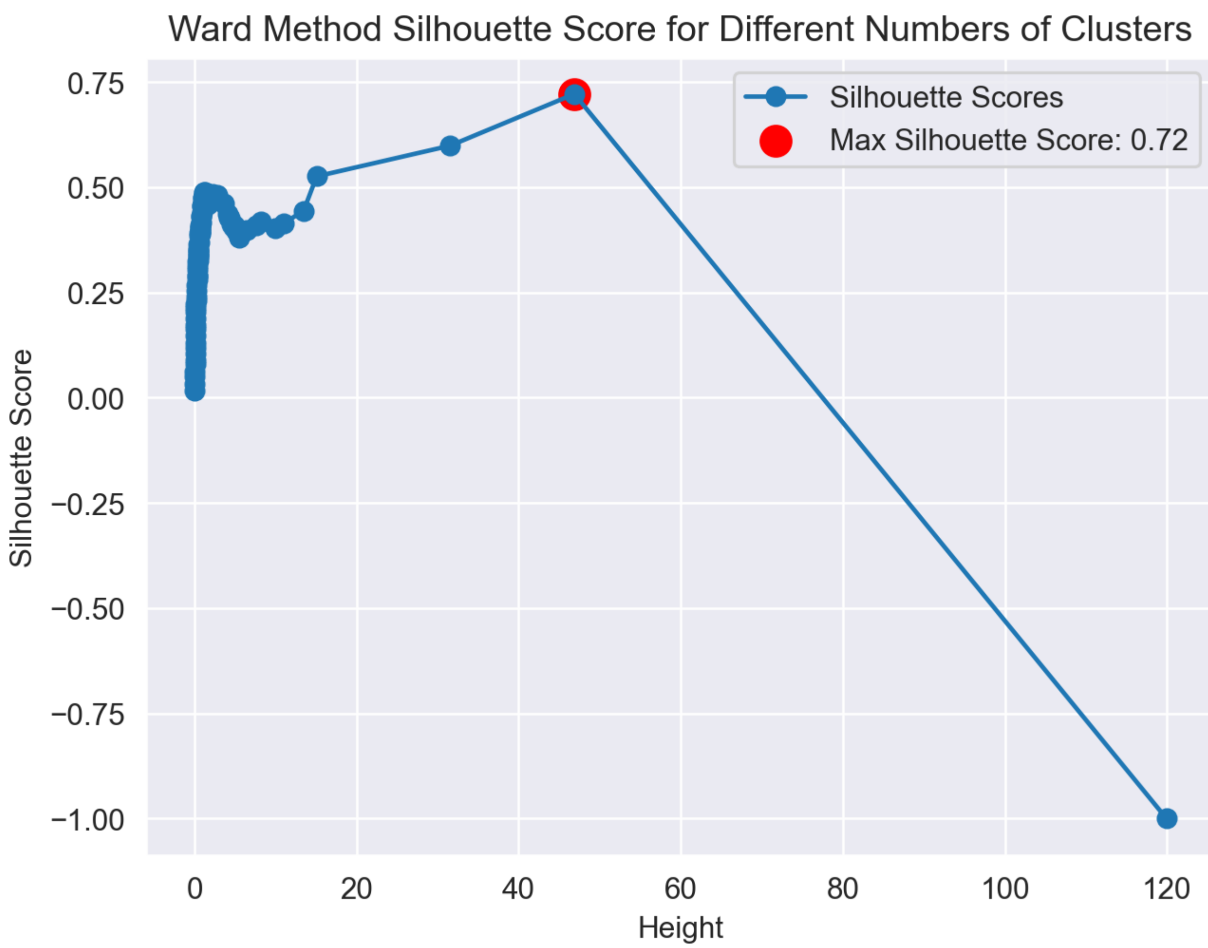



```python
plot_silhouette_over_k(z_average)
plt.title('Average Method Silhouette Score for Different Numbers of Clusters')
plt.show()
```
[23] ✓ 0.4s Python



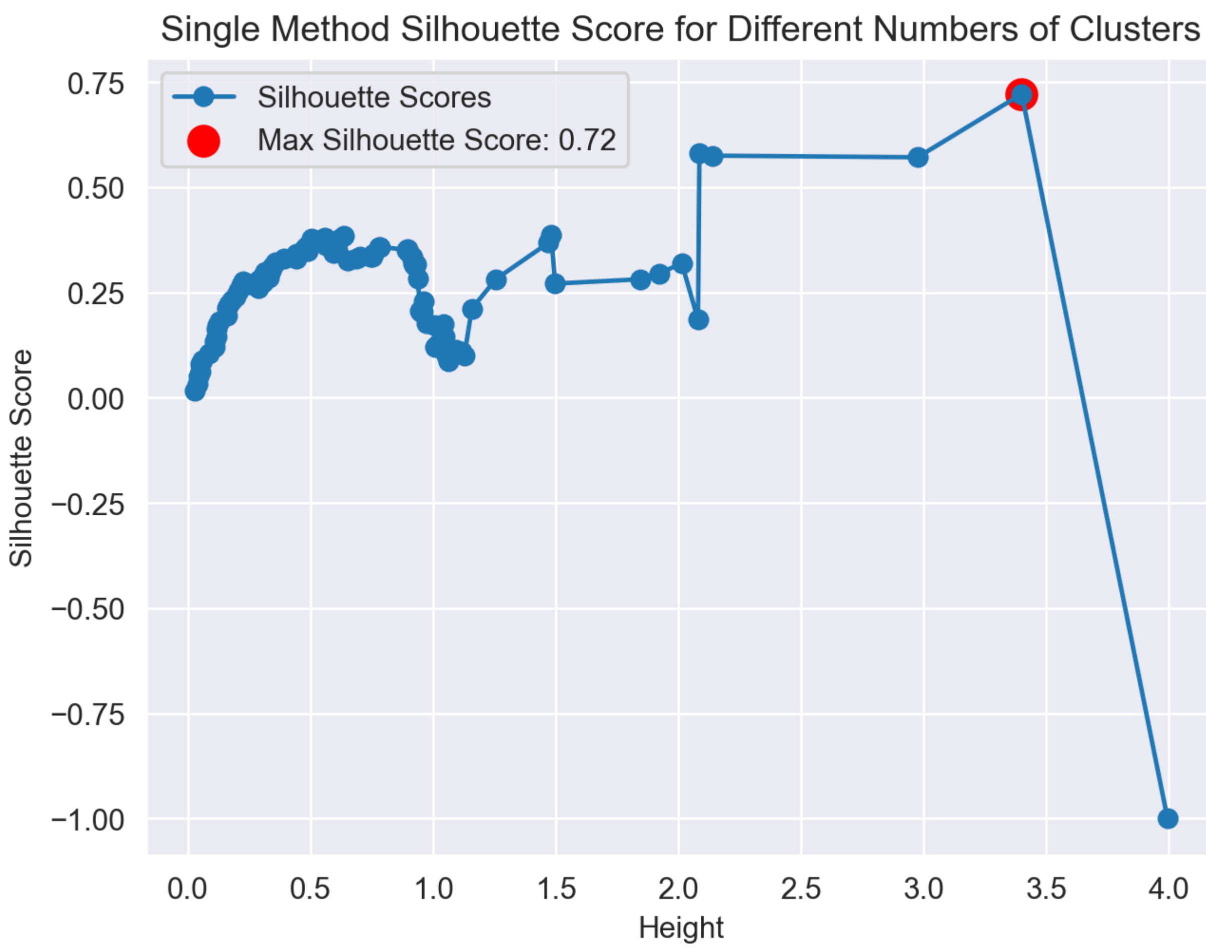

```python
plot_silhouette_over_k(z_ward)
plt.title('Ward Method Silhouette Score for Different Numbers of Clusters')
plt.show()
```

[24] ✓ 0.4s Python




```python
plot_silhouette_over_k(z_single)
plt.title('Single Method Silhouette Score for Different Numbers of Clusters')
plt.show()
```

[25] ✓ 0.8s Python



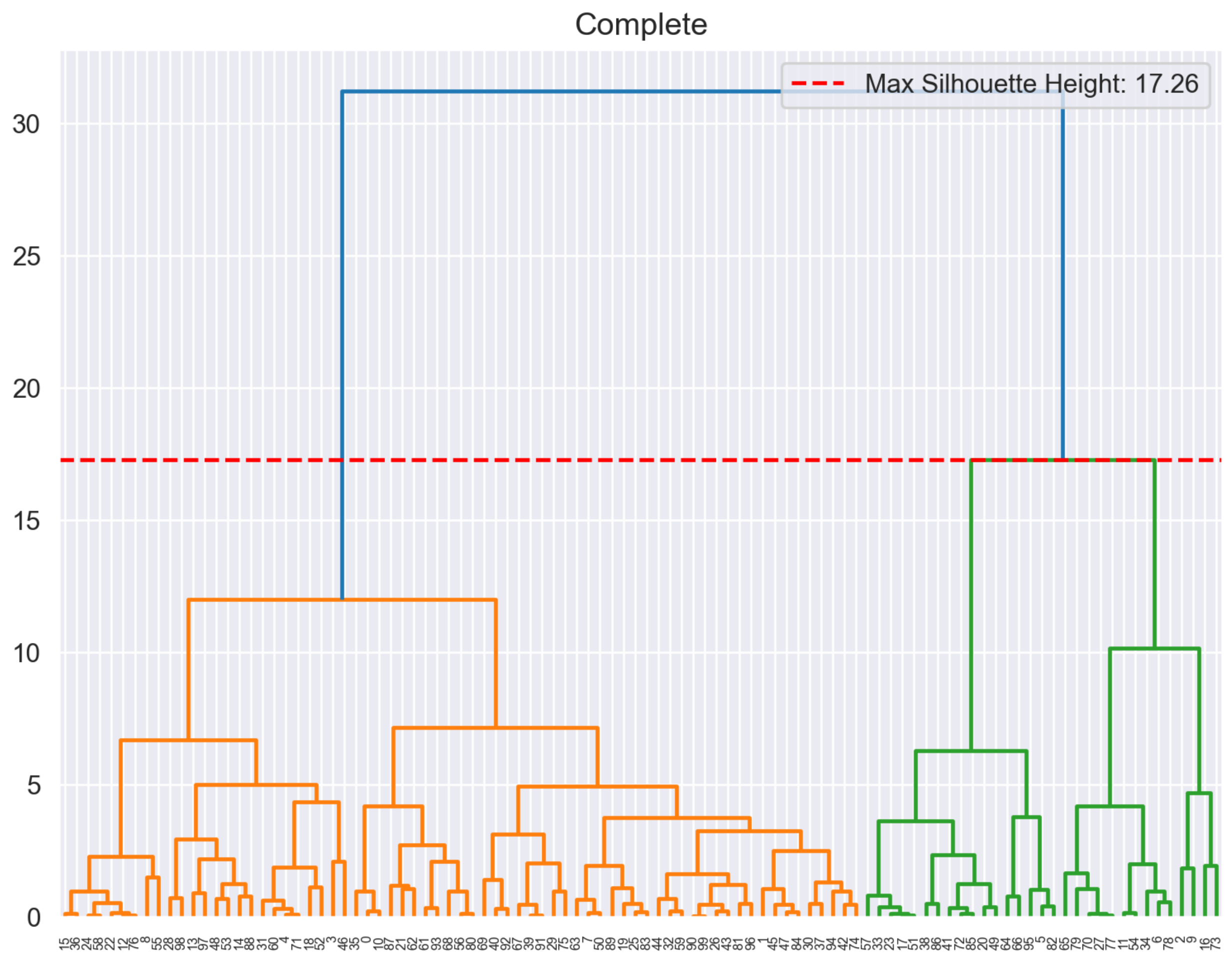
Plotting the Horizontal Line that best cuts the Dendrograms into Optimal Clusters (K)

```python
def max_silhouette_height(Z):
    silhouette_scores = []
    heights = np.unique(Z[:, 2])
    for height in np.unique(Z[:, 2]):
        # Extract cluster labels based on current height
        labels = fcluster(Z, height, criterion='distance')
        n_clusters = len(np.unique(labels))
        if n_clusters > 1:  # Ensure at least two clusters
            # Compute silhouette score
            silhouette_scores.append(silhouette_score(new_data, labels))
        else:
            silhouette_scores.append(-1)

    # Interpolate silhouette scores for all unique heights
    interp_silhouette_scores = np.interp(np.unique(Z[:, 2]), heights, silhouette_scores)

    # Find the index of maximum silhouette score
    max_silhouette_idx = np.argmax(interp_silhouette_scores)
    max_silhouette_height = np.unique(Z[:, 2])[max_silhouette_idx]
    max_silhouette_score = interp_silhouette_scores[max_silhouette_idx]

    return max_silhouette_height, max_silhouette_score
```

[26] ✓ 0.0s Python

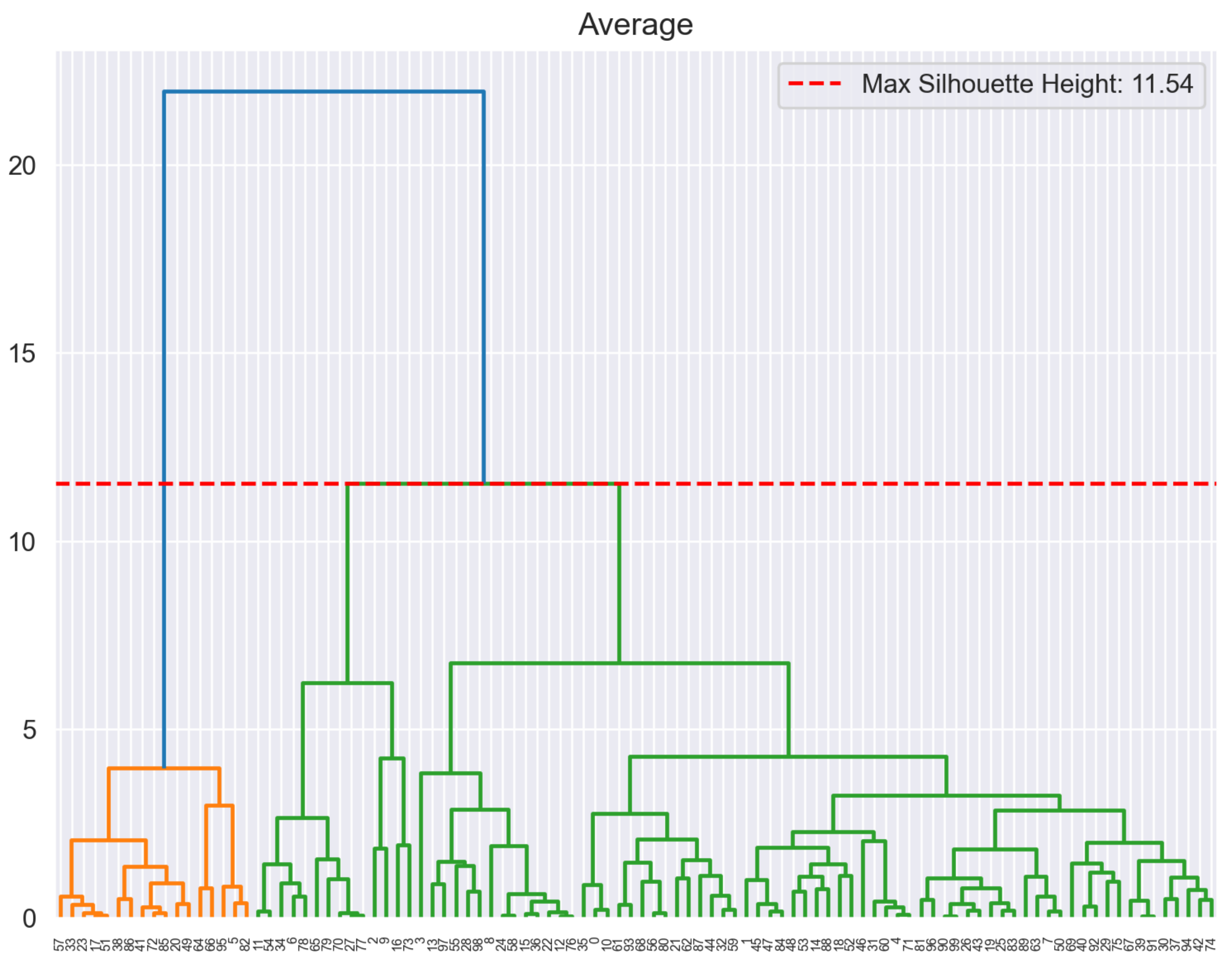
Best-fit horizontal line

In this function we will do the same as the above function but this time we will need it to plot horizontal line that best cuts the dendrograms into optimal clusters (K) then plot dendrograms.

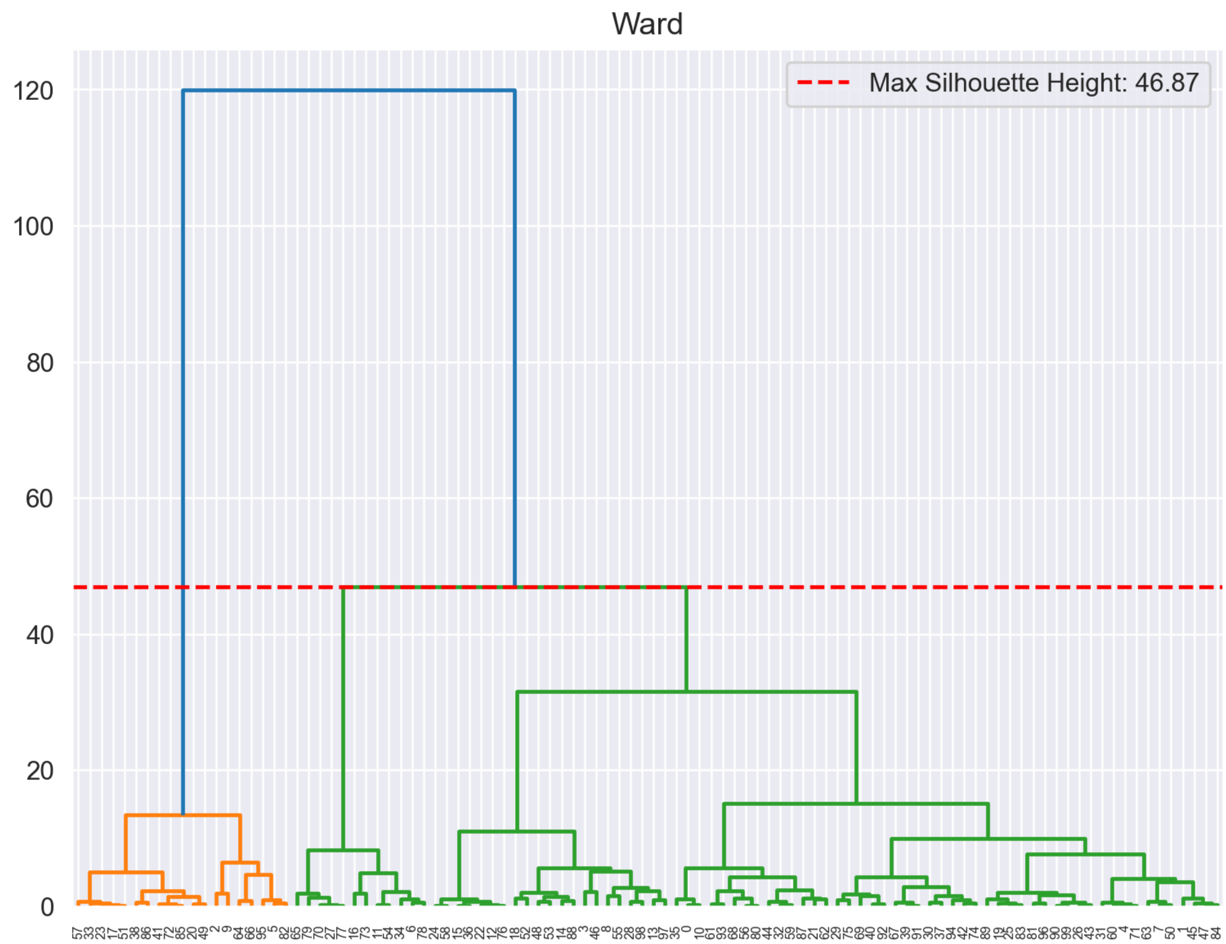



```python
# Plot for Average linkage
plt.figure(figsize=(8, 6))
dendrogram(z_average)
plt.title('Average')
max_sil_height_2 = max_silhouette_height(z_average)[0]
plt.axhline(y=max_sil_height_2, color='red', linestyle='--', label='Max Silhouette Height: {:.2f}'.format(max_sil_height_2))
plt.legend()
plt.show()
```

[28] ✓ 1.3s Python

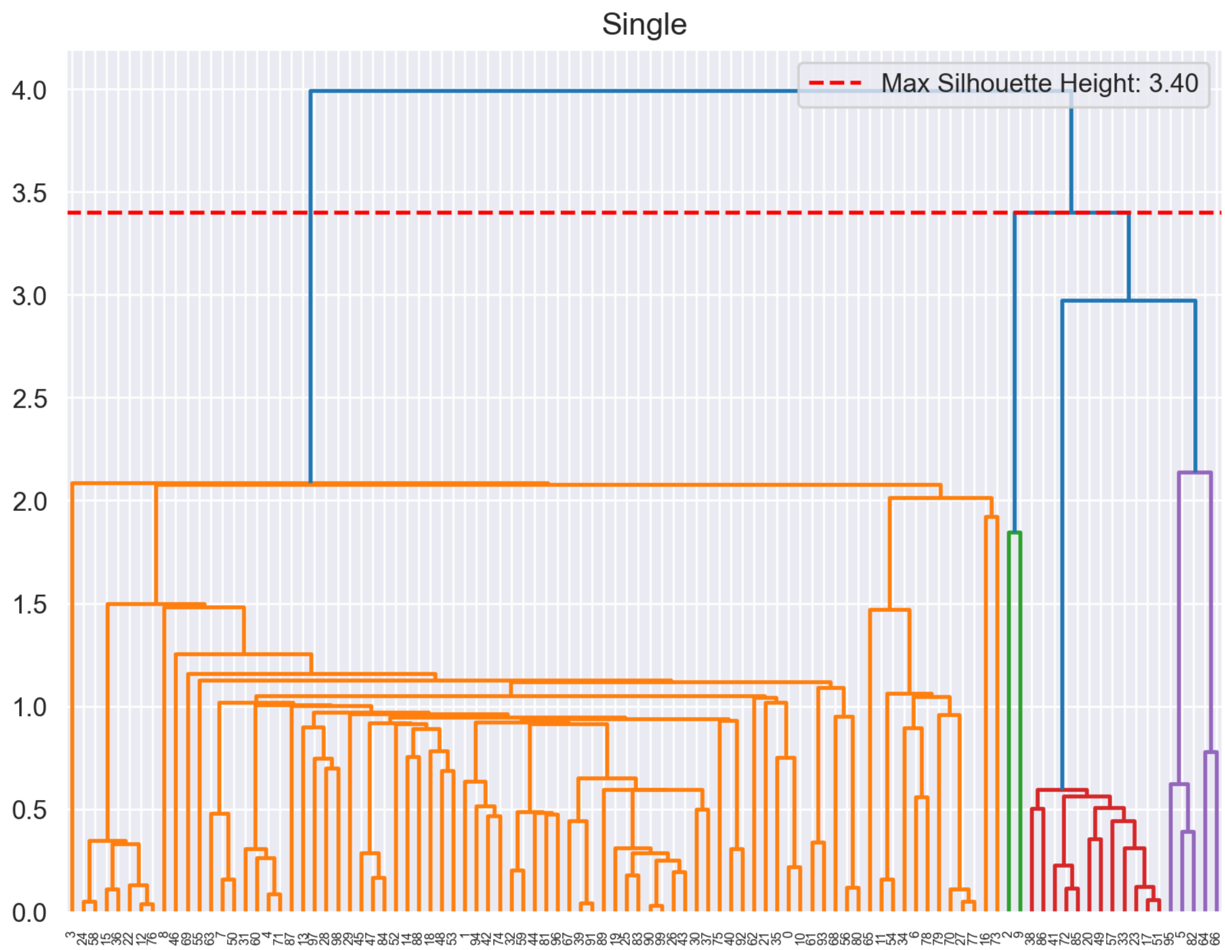



```python
# Plot for Ward linkage
plt.figure(figsize=(8, 6))
dendrogram(z_ward)
plt.title('Ward')
max_sil_height_2 = max_silhouette_height(z_ward)[0]
plt.axhline(y=max_sil_height_2, color='red', linestyle='--', label='Max Silhouette Height: {:.2f}'.format(max_sil_height_2))
plt.legend()
plt.show()
```




```python
# Plot for Single linkage
plt.figure(figsize=(8, 6))
dendrogram(z_single)
plt.title('Single')
max_sil_height_2 = max_silhouette_height(z_single)[0]
plt.axhline(y=max_sil_height_2, color='red', linestyle='--', label='Max Silhouette Height: {:.2f}'.format(max_sil_height_2))
plt.legend()
plt.show()
```

[30] ✓ 1.3s Python



Comparison between used clustering methods according to evaluation

Comparison between k_medoids and Hierarchical

```python
best_davies_bouldin_score=min([hierarchical_davies_bouldin,k_medoids_davies_bouldin])
best_silhouette_score=max([hierarchical_silhouette, k_medoids_silhouette])
print('the best davies_bouldin_score', best_davies_bouldin_score)
print('the best silhouette_score', best_silhouette_score)
```

[31] ✓ 0.0s Python

```
the best davies_bouldin_score 0.3751872722533399
the best silhouette_score 0.7140696333428328
```

Next we will make small comparison between k_medoids and Hierarchical using “silhouette_score” and “davies_bouldin_score” so we will find minimum “davies_bouldin_score” between k_medoids and Hierarchical because minimum “davies_bouldin_score” is better and maximum “silhouette_score” between k_medoids and Hierarchical because maximum “silhouette_score” is better.

Visualizing the Performance of each Clustering Method

```python
metrics = ['Silhouette Score', 'Davies-Bouldin Score']
k_medoids_scores = [k_medoids_silhouette, k_medoids_davies_bouldin]
hierarchical_scores = [hierarchical_silhouette, hierarchical_davies_bouldin]

x = np.arange(len(metrics))
width = 0.35

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, k_medoids_scores, width, label='K-Medoids')
rects2 = ax.bar(x + width/2, hierarchical_scores, width, label='Hierarchical')

ax.set_xlabel('Evaluation Metrics')
ax.set_ylabel('Scores')
ax.set_title('Clustering Performance Comparison')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()

def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(round(height, 2)),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)

fig.tight_layout()

plt.show()
```