Experiment No. 5
Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset
Date of Performance: 21-08-2023
Date of Submission: 05-09-2023

Aim: Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset.

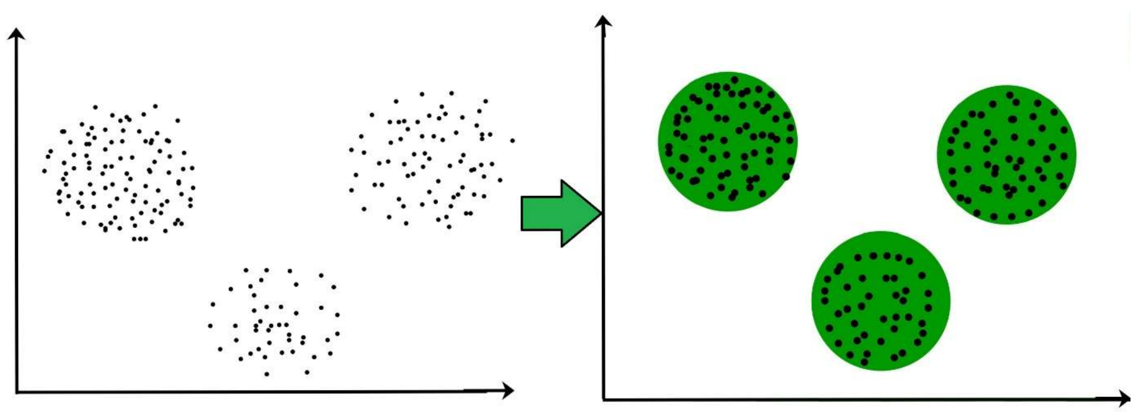
Objective: Able to perform various feature engineering tasks, apply Clustering Algorithm on the given dataset.

Theory:

It is basically a type of unsupervised learning method. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labeled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

For example: The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



Dataset:

This data set refers to clients of a wholesale distributor. It includes the annual spending in monetary units (m.u.) on diverse product categories. The wholesale distributor operating in different regions of Portugal has information on annual spending of several items in their stores across different regions and channels. The dataset consist of 440 large retailers annual spending on 6 different varieties of product in 3 different regions (lisbon , oporto, other) and across different sales channel (Hotel, channel)

Detailed overview of dataset

Records in the dataset = 440 ROWS

Columns in the dataset = 8 COLUMNS

FRESH: annual spending (m.u.) on fresh products (Continuous)

MILK:- annual spending (m.u.) on milk products (Continuous)

GROCERY:- annual spending (m.u.) on grocery products (Continuous)

FROZEN:- annual spending (m.u.) on frozen products (Continuous)

DETERGENTS_PAPER :- annual spending (m.u.) on detergents and paper products (Continuous)

DELICATESSEN:- annual spending (m.u.)on and delicatessen products (Continuous);

CHANNEL: - sales channel Hotel and Retailer

REGION:- three regions (Lisbon, Oporto, Other)

Code:

Conclusion:

Based on the visualization, comment on following:

1. How can you can make use of the clustered data?

   Utilizing Clustered Data:

   Targeted Marketing: Customize marketing strategies for each cluster.

   Inventory Management: Optimize stock levels based on cluster preferences.

   Supply Chain Optimization: Tailor delivery schedules to cluster needs.

   Product Recommendations: Offer personalized product suggestions.

   Customer Service: Adapt service based on cluster preferences.

2. How the different groups of customers, the ***customer segments***, may be affected differently by a specific delivery scheme?

   Effect of Delivery Scheme on Customer Segments:

   Cluster 0: Flexible delivery for diverse product needs.

   Cluster 1: Subscription-based for essential items.

   Cluster 2: Freshness guarantee with quick delivery.

Cluster 3: Cost-effective and efficient delivery options. Collect feedback from each cluster to refine delivery schemes for better satisfaction and loyalty.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('customers.csv')
print(df)
```

```
     Channel  Region  Fresh   Milk  Grocery  Frozen  Detergents_Paper  \
0          2       3  12669   9656     7561     214              2674
1          2       3   7057   9810     9568    1762              3293
2          2       3   6353   8808     7684    2405              3516
3          1       3  13265   1196     4221    6404               507
4          2       3  22615   5410     7198    3915              1777
..       ...     ...    ...    ...      ...     ...               ...
435        1       3  29703  12051    16027   13135               182
436        1       3  39228   1431      764    4510                93
437        2       3  14531  15488    30243     437             14841
438        1       3  10290   1981     2232    1038               168
439        1       3   2787   1698     2510      65               477

     Delicatessen
0            1338
1            1776
2            7844
3            1788
4            5185
..            ...
435          2204
436          2346
437          1867
438          2125
439            52

[440 rows x 8 columns]
```

```
df.head()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

```
print("Data Types")
df.dtypes
```

```
Data Types
Channel             int64
Region              int64
Fresh               int64
Milk                int64
Grocery             int64
Frozen              int64
Detergents_Paper    int64
Delicatessen        int64
dtype: object
```

```
print("Missing values per column:")
print(df.isnull().sum())
```

```
Missing values per column:
Channel             0
Region              0
Fresh               0
Milk                0
Grocery             0
Frozen              0
Detergents_Paper    0
Delicatessen        0
dtype: int64
```

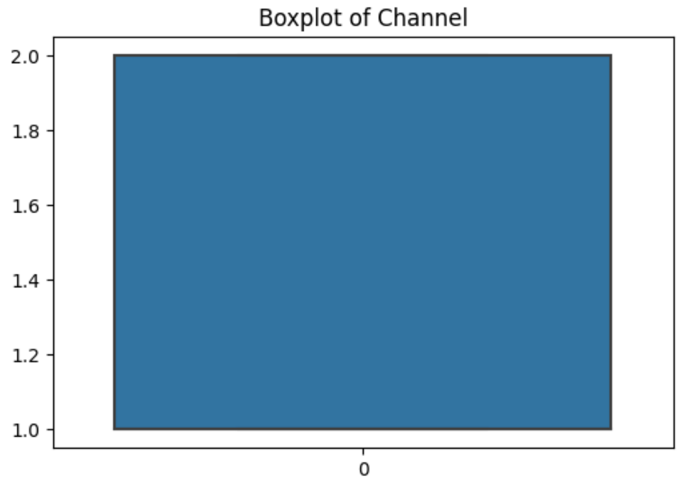
```
print("Descriptive Statistics:")
print(df.describe())
print("Number of duplicate rows: ", df.duplicated().sum())
```

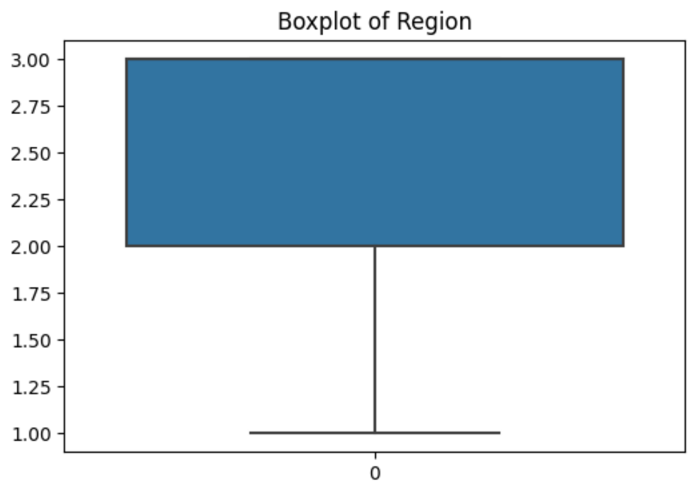
```
Descriptive Statistics:
          Channel      Region          Fresh          Milk       Grocery  \
count  440.000000  440.000000     440.000000    440.000000    440.000000
mean     1.322727    2.543182   12000.297727   5796.265909   7951.277273
std      0.468052    0.774272   12647.328865   7380.377175   9503.162829
min      1.000000    1.000000       3.000000     55.000000      3.000000
25%      1.000000    2.000000    3127.750000   1533.000000   2153.000000
50%      1.000000    3.000000    8504.000000   3627.000000   4755.500000
75%      2.000000    3.000000   16933.750000   7190.250000  10655.750000
max      2.000000    3.000000  112151.000000  73498.000000  92780.000000

             Frozen  Detergents_Paper  Delicatessen
count    440.000000        440.000000    440.000000
mean    3071.931818       2881.493182   1524.870455
std     4854.673333       4767.854448   2820.105937
min       25.000000          3.000000      3.000000
25%      742.250000        256.750000    408.250000
50%     1526.000000        816.500000    965.500000
75%     3554.250000       3922.000000   1820.250000
max    60869.000000      40827.000000  47943.000000
Number of duplicate rows:  0
```

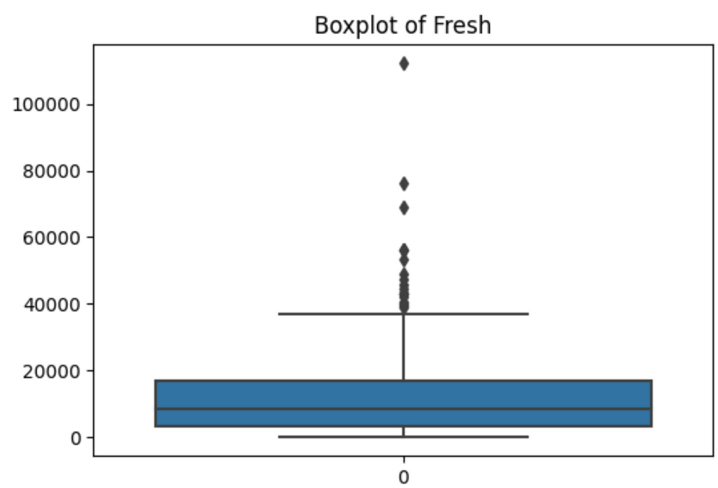
```
df.corr()
```

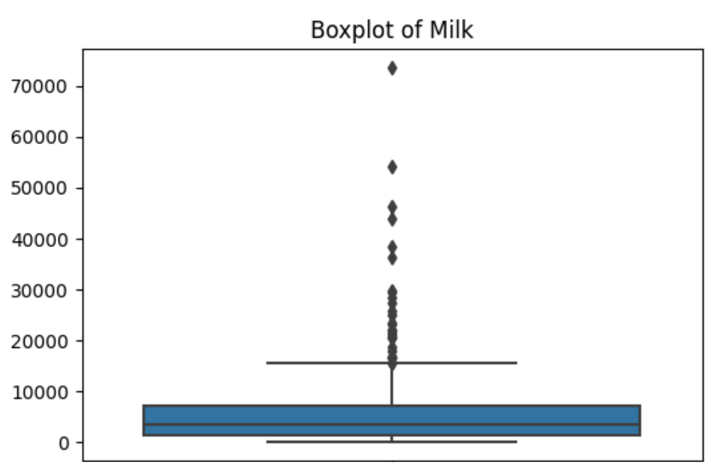
	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
Channel	1.000000	0.062028	-0.169172	0.460720	0.608792	-0.202046	0.636026	0.056011
Region	0.062028	1.000000	0.055287	0.032288	0.007696	-0.021044	-0.001483	0.045212
Fresh	-0.169172	0.055287	1.000000	0.100510	-0.011854	0.345881	-0.101953	0.244690
Milk	0.460720	0.032288	0.100510	1.000000	0.728335	0.123994	0.661816	0.406368
Grocery	0.608792	0.007696	-0.011854	0.728335	1.000000	-0.040193	0.924641	0.205497
Frozen	-0.202046	-0.021044	0.345881	0.123994	-0.040193	1.000000	-0.131525	0.390947
Detergents_Paper	0.636026	-0.001483	-0.101953	0.661816	0.924641	-0.131525	1.000000	0.069291
Delicatessen	0.056011	0.045212	0.244690	0.406368	0.205497	0.390947	0.069291	1.000000

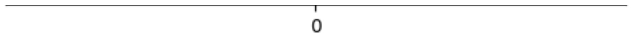
```
import seaborn as sns
import matplotlib.pyplot as plt

# boxplots for all features
for column in df.columns:
    plt.figure(figsize=(6, 4))
    sns.boxplot(df[column])
    plt.title(f'Boxplot of {column}')
    plt.show()

# Function to detect outliers
def detect_outliers(dataframe, column):
    Q1 = dataframe[column].quantile(0.25)
    Q3 = dataframe[column].quantile(0.75)
    IQR = Q3 - Q1
    outliers = dataframe[(dataframe[column] < Q1 - 1.5*IQR) | (dataframe[column] > Q3 + 1.5*IQR)]
    return outliers

# number of outliers for each feature
for column in df.columns:
    outliers = detect_outliers(df, column)
    print(f'Number of outliers in {column}: {len(outliers)}')
```

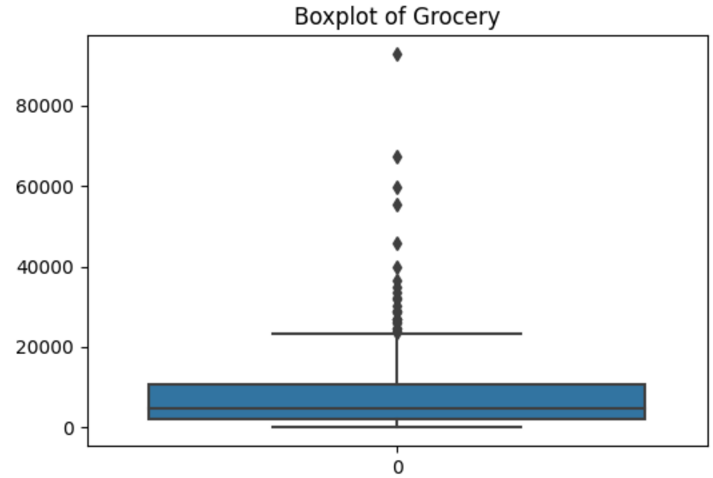

Boxplot of Channel
2.0
1.8
1.6
1.4
1.2
1.0
0

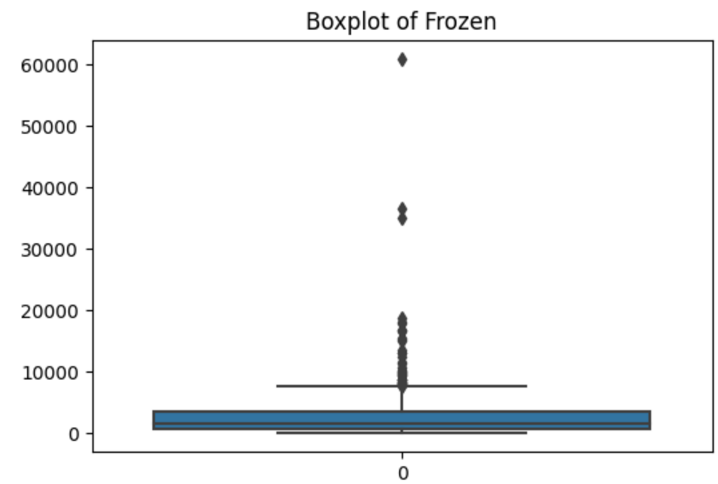

Boxplot of Region
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0

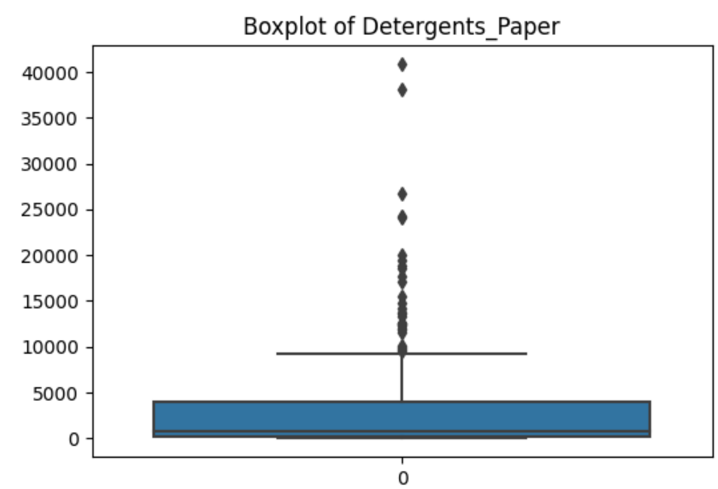

Boxplot of Fresh
100000
80000
60000
40000
20000
0
0


Boxplot of Milk
70000
60000
50000
40000
30000
20000
10000
0


0

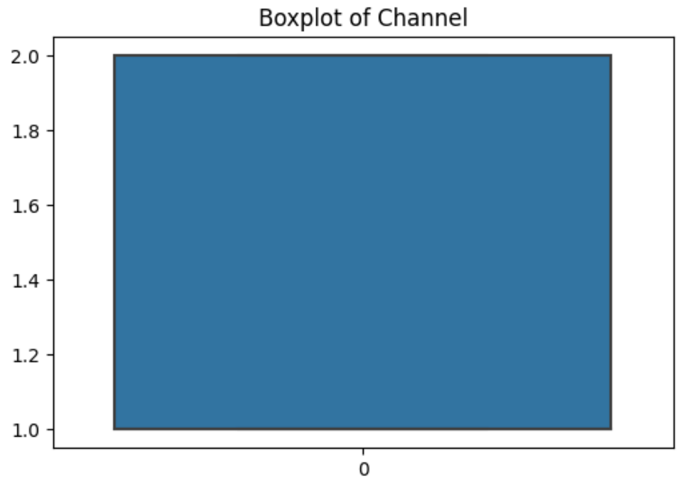

Boxplot of Grocery

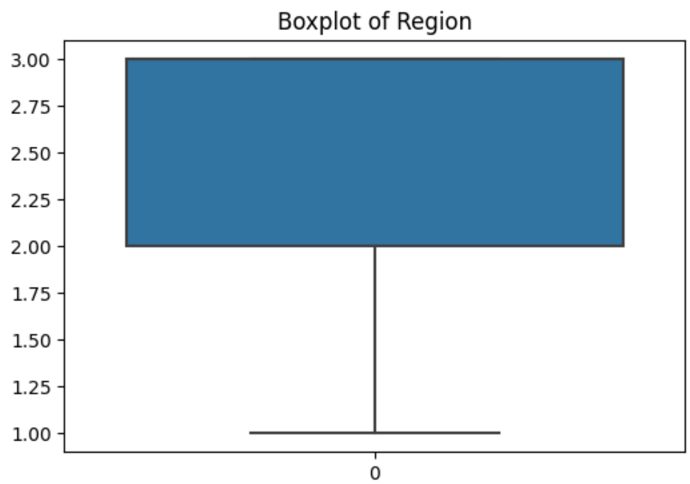

Boxplot of Frozen

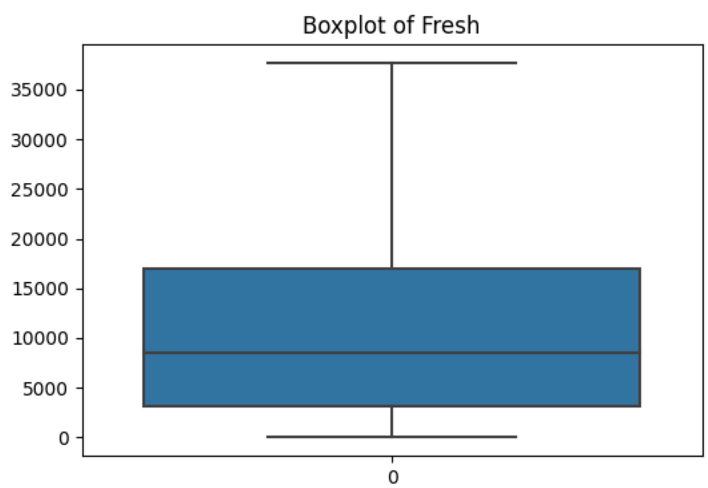

Boxplot of Detergents_Paper

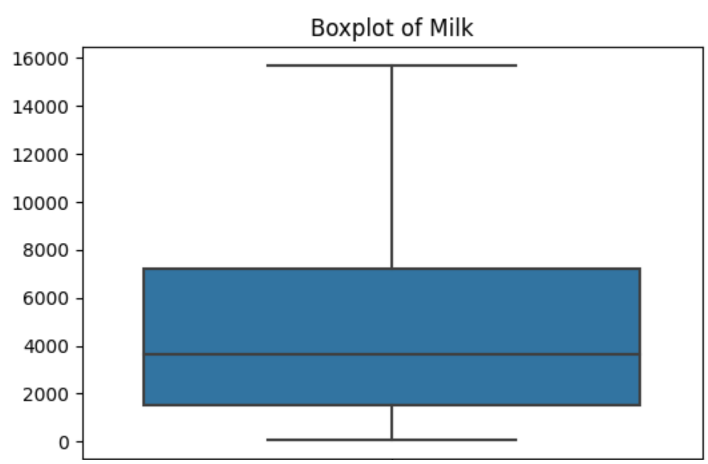

Boxplot of Delicatessen

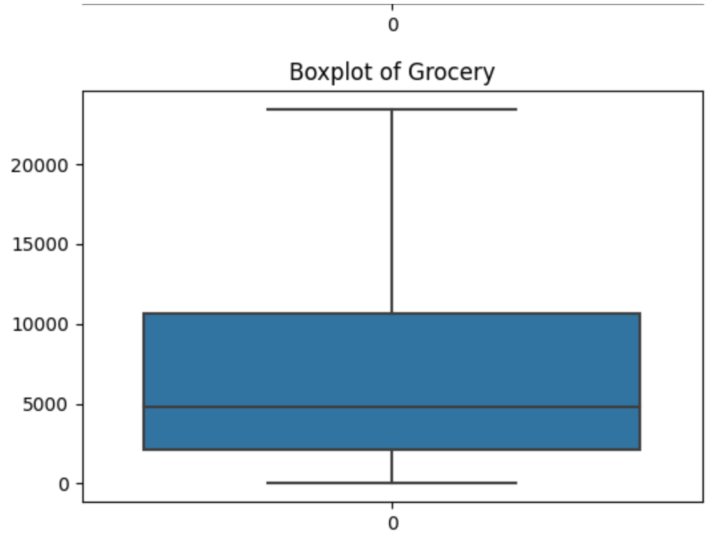
```
def handle_outliers(dataframe, column):
    Q1 = dataframe[column].quantile(0.25)
    Q3 = dataframe[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_limit = Q1 - 1.5*IQR
    upper_limit = Q3 + 1.5*IQR
    dataframe[column] = dataframe[column].apply(lambda x: upper_limit if x > upper_limit else lower_limit if x < lower_limit else x)

for column in df.columns:
    handle_outliers(df, column)


for column in df.columns:
    plt.figure(figsize=(6, 4))
    sns.boxplot(df[column])
    plt.title(f'Boxplot of {column}')
    plt.show()
```


Boxplot of Channel
2.0
1.8
1.6
1.4
1.2
1.0
0


Boxplot of Region
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0


Boxplot of Fresh
35000
30000
25000
20000
15000
10000
5000
0
0


Boxplot of Milk
16000
14000
12000
10000
8000
6000
4000
2000
0




```
def detect_outliers(dataframe, column):
    Q1 = dataframe[column].quantile(0.25)
    Q3 = dataframe[column].quantile(0.75)
    IQR = Q3 - Q1
    outliers = dataframe[(dataframe[column] < Q1 - 1.5*IQR) | (dataframe[column] > Q3 + 1.5*IQR)]
    return outliers

for column in df.columns:
    outliers = detect_outliers(df, column)
    print(f'Number of outliers in {column}: {len(outliers)}')
```

```
Number of outliers in Channel: 0
Number of outliers in Region: 0
Number of outliers in Fresh: 0
Number of outliers in Milk: 0
Number of outliers in Grocery: 0
Number of outliers in Frozen: 0
Number of outliers in Detergents_Paper: 0
Number of outliers in Delicatessen: 0
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
```

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Calculate WCSS for different number of clusters
wcss = []
max_clusters = 15
for i in range(1, max_clusters+1):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(df)
    wcss.append(kmeans.inertia_)

# Plot the WCSS values
plt.plot(range(1, max_clusters+1), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
```

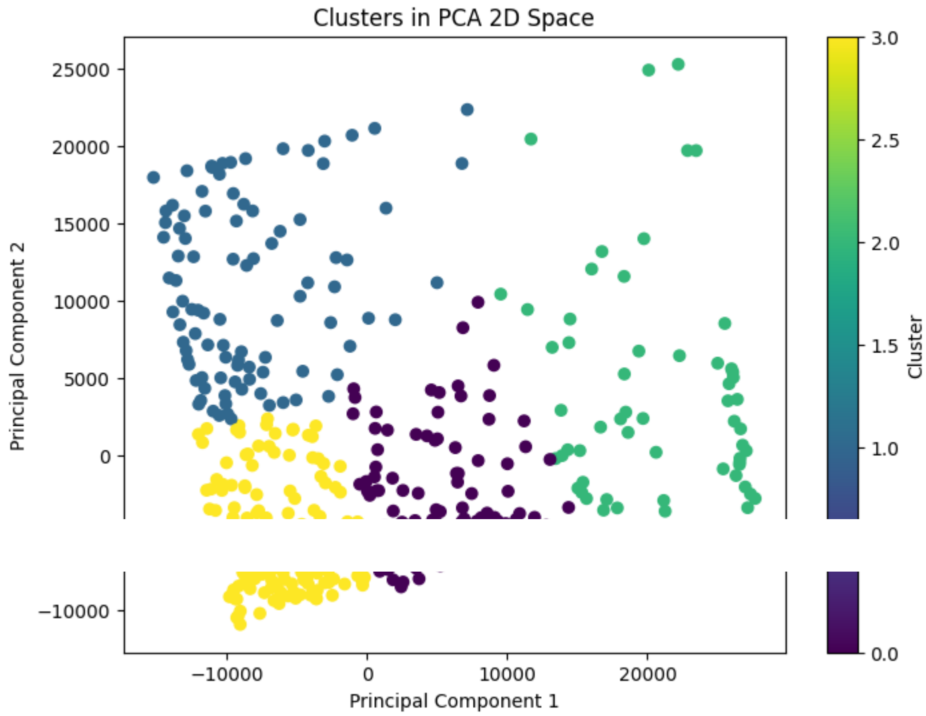

```
from sklearn.cluster import KMeans

# Build the model
kmeans = KMeans(n_clusters=4, init='k-means++', random_state=42)
kmeans.fit(df)

# Get cluster labels
cluster_labels = kmeans.labels_

# Add cluster labels to your original dataframe
df['Cluster'] = cluster_labels
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
```

```
df['Cluster'] = kmeans.labels_

# Check the size of each cluster
print("Cluster Sizes:\n", df['Cluster'].value_counts())

# Check the characteristics of each cluster
for i in range(4):
    print("\nCluster ", i)
    print(df[df['Cluster'] == i].describe())
```

```
count    94.000000        94.000000    94.000000     94.0
mean   1496.428191      6936.898936  1547.364362      1.0
std    1538.882840      2383.035957  1176.131062      0.0
min      25.000000       241.000000     3.000000      1.0
25%     438.500000      5274.250000   680.000000      1.0
50%     973.000000      6931.500000  1366.500000      1.0
75%    1900.000000      9419.875000  2157.750000      1.0
max    7772.250000      9419.875000  3938.250000      1.0

Cluster  2
         Channel     Region         Fresh          Milk       Grocery  \
count  58.000000  58.000000     58.000000     58.000000     58.000000
mean    1.172414   2.655172  32136.810345   5973.515086   7309.012931
std     0.381039   0.714554   5122.024937   4808.223223   5915.174661
min     1.000000   1.000000  22647.000000    286.000000    471.000000
25%     1.000000   3.000000  27207.500000   2393.000000   2726.250000
50%     1.000000   3.000000  31664.000000   4347.000000   5259.500000
75%     1.000000   3.000000  37642.750000   7829.500000   9344.000000
max     2.000000   3.000000  37642.750000  15676.125000  23409.875000

            Frozen  Detergents_Paper  Delicatessen  Cluster
count    58.000000         58.000000     58.000000     58.0
mean   4170.017241       1417.426724   1967.702586      2.0
std    2841.060439       2055.702539   1267.507352      0.0
min     127.000000         10.000000      3.000000      2.0
25%    1370.750000        250.250000   1037.250000      2.0
50%    3662.000000        617.500000   1821.500000      2.0
75%    7772.250000       1428.000000   2910.250000      2.0
max    7772.250000       9419.875000   3938.250000      2.0

Cluster  3
          Channel      Region         Fresh          Milk       Grocery  \
count  176.000000  176.000000    176.000000    176.000000    176.000000
mean     1.136364    2.539773   4741.261364   3073.790483   3817.880682
std      0.344153    0.777254   3072.006036   2492.137013   2790.348628
min      1.000000    1.000000      3.000000     55.000000    137.000000
25%      1.000000    2.000000   2116.000000   1109.000000   1739.250000
50%      1.000000    3.000000   4659.500000   2268.000000   2765.500000
75%      1.000000    3.000000   7369.250000   4394.250000   5494.500000
max      2.000000    3.000000  10290.000000  15676.125000  12400.000000

            Frozen  Detergents_Paper  Delicatessen  Cluster
count   176.000000        176.000000    176.000000    176.0
mean   2192.274148       1176.454545    909.451705      3.0
std    2210.017535       1473.393792    872.339683      0.0
min      47.000000          5.000000      3.000000      3.0
25%     587.750000        216.500000    308.250000      3.0
50%    1310.000000        472.500000    674.500000      3.0
75%    2964.250000       1545.000000   1154.750000      3.0
max    7772.250000       7271.000000   3938.250000      3.0
```

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Apply PCA and fit the features selected
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(df.drop('Cluster', axis=1))

# Create a DataFrame with the two components
PCA_components = pd.DataFrame(principalComponents, columns=['Principal Component 1', 'Principal Component 2'])

# Concatenate the clusters labels to the DataFrame
PCA_components['Cluster'] = df['Cluster']

# Plot the clustered dataset
plt.figure(figsize=(8,6))
plt.scatter(PCA_components['Principal Component 1'], PCA_components['Principal Component 2'], c=PCA_components['Cluster'])
plt.title('Clusters in PCA 2D Space')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.show()
```



Clusters in PCA 2D Space
Principal Component 1
Principal Component 2
Cluster
25000
20000
15000
10000
5000
0
−10000
0
10000
20000
3.0
2.5
2.0
1.5
1.0
0.0