| |
|---|
| Experiment No. 5 |
| Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset |
| Date of Performance: |
| Date of Submission: |

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

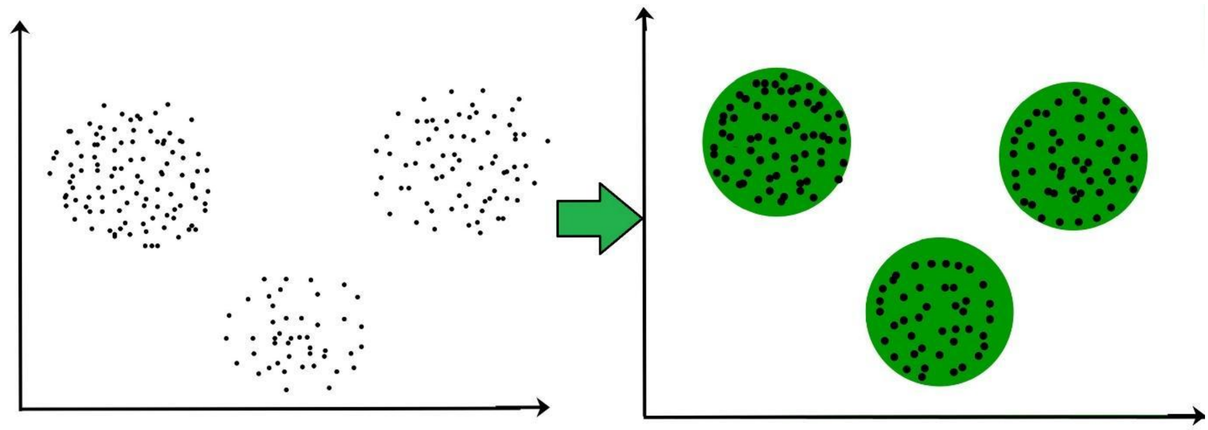
**Objective:** Able to perform various feature engineering tasks, apply Clustering Algorithm on the given dataset.

**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?

Clustered data can be highly valuable for various data analysis and business purposes:

**Customer Segmentation**: Clustering helps identify distinct customer segments based on their purchasing behavior. This information can be used for targeted marketing and product recommendations, tailoring strategies to suit the preferences of each group.

**Inventory Management**: Understanding cluster preferences can optimize inventory management. If a cluster predominantly buys certain products, stock levels for those items can be adjusted accordingly.

**Supply Chain Optimization**: Efficient supply chain management can be achieved by recognizing which clusters demand certain products and when they typically order them.

**Customer Service**: Customer service strategies can be customized based on cluster behavior. Some clusters might require more assistance or have different communication preferences.

Overall, leveraging clustered data allows businesses to make informed decisions, enhance customer experiences, and improve operational efficiency by tailoring their strategies to the distinct needs and behaviors of each cluster.

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

Different customer segments can indeed be affected differently by a specific delivery scheme. Here's how it can impact various customer segments:

**Cluster 0 (e.g., Smaller Buyers):**

Positive Impact: A more efficient and cost-effective delivery scheme may attract these customers to purchase more frequently due to reduced delivery costs. It can lead to increased customer loyalty.

**Cluster 1 (e.g., Medium-Sized Buyers):**

Positive Impact: This segment may appreciate a more flexible and faster delivery scheme. It could encourage them to place larger orders or buy more frequently, particularly for perishable goods.

**Cluster 2 (e.g., Large Buyers):**

Positive Impact: Efficient delivery could be crucial for this group, given their substantial purchasing volume. A reliable and timely delivery scheme can ensure the uninterrupted supply of goods, fostering a strong and long-lasting business relationship.

**Cluster 3 (Empty Cluster):**

Since this cluster contains no data, it's not applicable in this context.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
for dirname, _, filenames in os.walk('/content/Wholesale customers data.csv'):
     for filename in filenames:
          print(os.path.join(dirname, filename))
```

```
import pandas as pd
# Define a function to load the data
def load_data(path):
     try:
         df = pd.read_csv(path)
         print("Data loaded successfully!")
         return df
     except Exception as e:
           print(f"An error occurred: {e}")
           return None
# Path to the data file
path = '/content/Wholesale customers data.csv'
# Load the data
df = load_data(path)
# Display the first few rows of the DataFrame
print(df.head())
```

```
Data loaded successfully!
   Channel  Region  Fresh  Milk  Grocery  Frozen  Detergents_Paper  Delicassen
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("Column names:")
print(df.columns)
```

```
Column names:
Index(['Channel', 'Region', 'Fresh', 'Milk', 'Grocery', 'Frozen',
       'Detergents_Paper', 'Delicassen'],
      dtype='object')
```

```
# Print the data types of each column
print("Data types:")
print(df.dtypes)
```

```
Data types:
Channel             int64
Region              int64
Fresh               int64
Milk                int64
Grocery             int64
Frozen              int64
Detergents_Paper    int64
Delicassen          int64
dtype: object
```

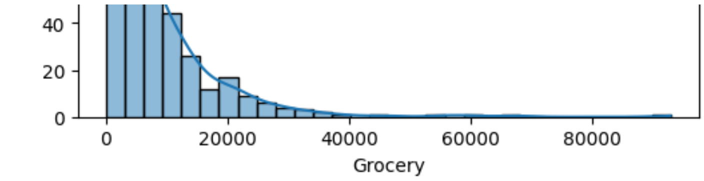
```
# Check for missing values
print("Missing values per column:")
print(df.isnull().sum())
```

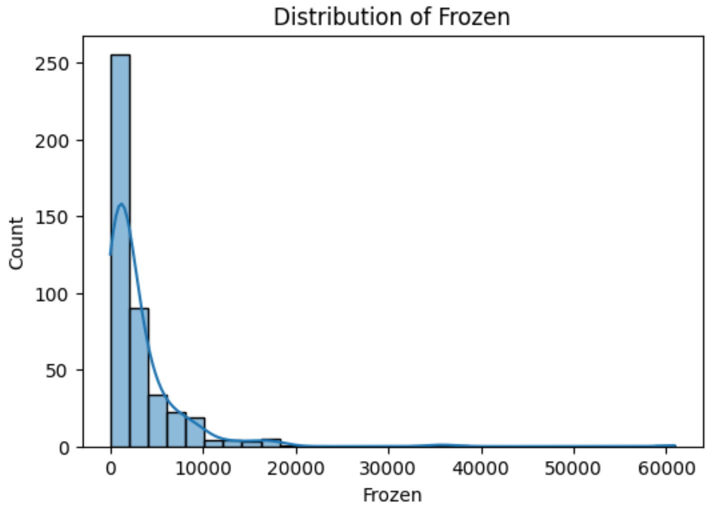
```
Missing values per column:
Channel             0
Region              0
Fresh               0
Milk                0
Grocery             0
Frozen              0
Detergents_Paper    0
Delicassen          0
dtype: int64
```

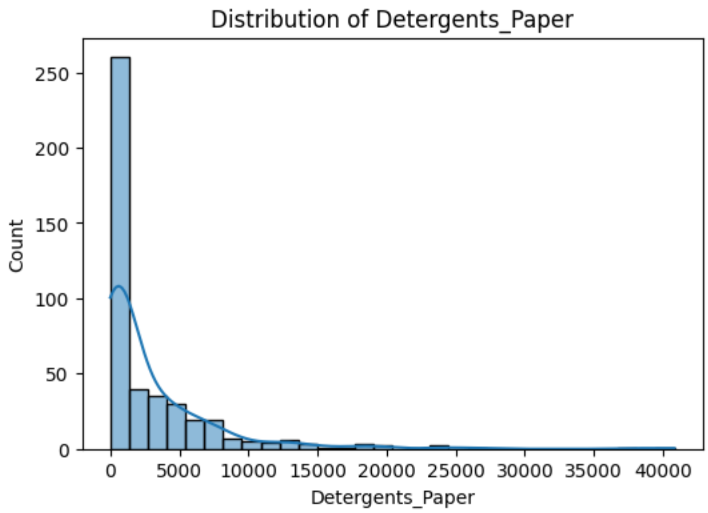
```
import matplotlib.pyplot as plt
import seaborn as sns
# Check descriptive statistics
print("Descriptive Statistics:")
print(df.describe())
# Check for duplicates
print("Number of duplicate rows: ", df.duplicated().sum())
```

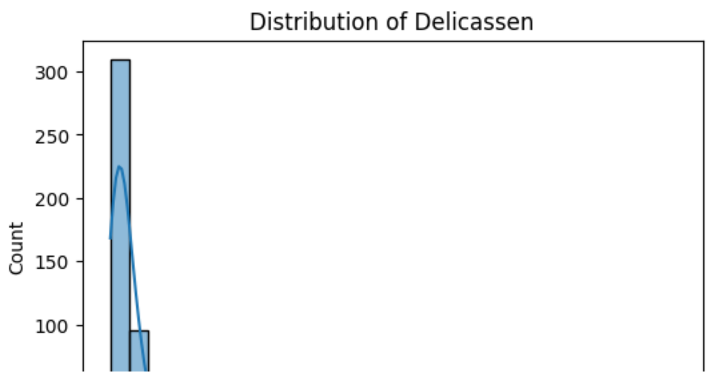
```
Descriptive Statistics:
          Channel      Region          Fresh          Milk       Grocery  \
count  440.000000  440.000000     440.000000    440.000000    440.000000
mean     1.322727    2.543182   12000.297727   5796.265909   7951.277273
std      0.468052    0.774272   12647.328865   7380.377175   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    Delicassen
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
```

```
# Distribution plots for each feature
for column in df.columns:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[column], bins=30, kde=True)
    plt.title(f'Distribution of {column}')
    plt.show()
# Heatmap for correlation between variables
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Heatmap')
plt.show()
```



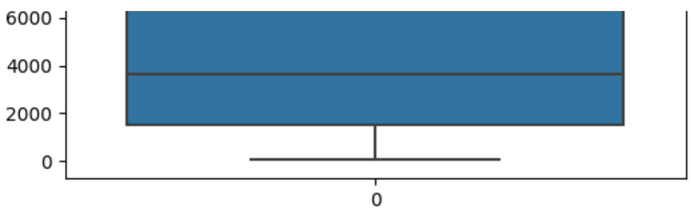



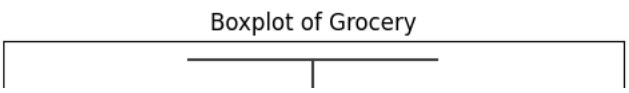

```
# checking for outliers
import seaborn as sns
import matplotlib.pyplot as plt
# Draw boxplots for all features
```

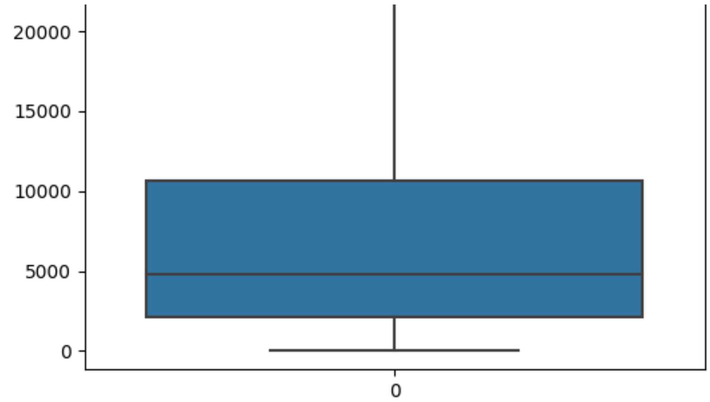
```
for column in df.columns:
    plt.figure(figsize=(6, 4))
    sns.boxplot(df[column])
    plt.title(f'Boxplot of {column}')
    plt.show()
# Function to detect outliers
def detect_outliers(dataframe, column):
     Q1 = dataframe[column].quantile(0.25)
     Q3 = dataframe[column].quantile(0.75)
     IQR = Q3 - Q1
     outliers = dataframe[(dataframe[column] < Q1 - 1.5*IQR)|(dataframe[column] > Q3 + 1.5*IQR)]
     return outliers
# Detect and print 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)}')
```

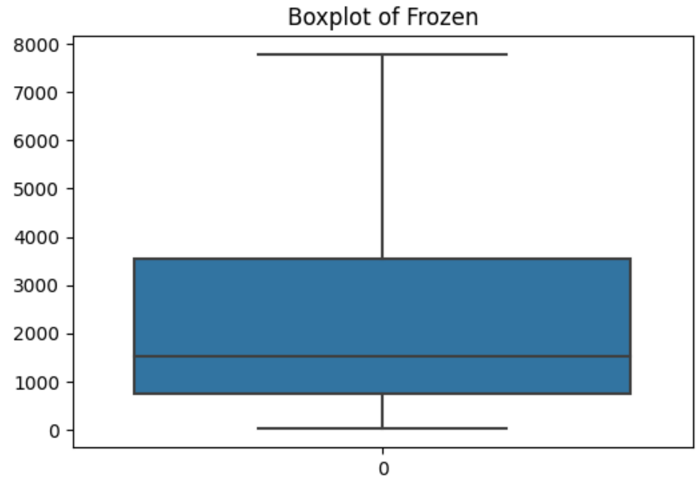
```
def handle_outliers(dataframe, column):
   Q1 = dataframe[column].quantile(0.25)
   Q3 = dataframe[column].quantile(0.75)
   IQR = Q3 - Q1
   lower_limit = Q1 - 1.5*IQR
   upper_limit = Q3 + 1.5*IQR
   dataframe[column] = dataframe[column].apply(lambda x: upper_limit if x > upper_limit else lower_limit if x < lower_limit else x)
# Handle outliers for each feature
for column in df.columns:
    handle_outliers(df, column)
```

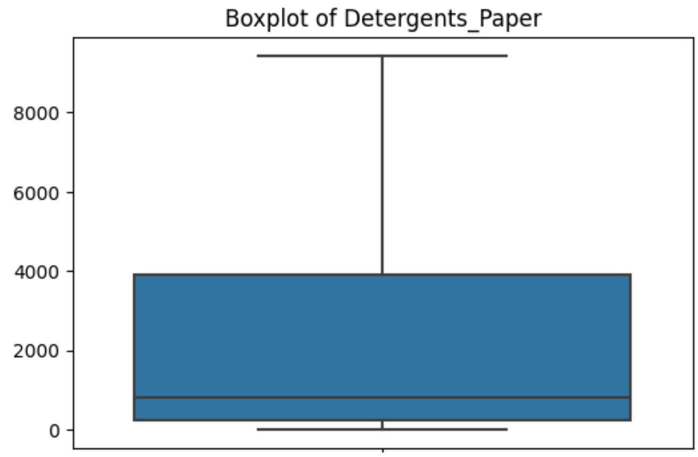
```
 # Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
# Draw boxplots for all features
for column in df.columns:
    plt.figure(figsize=(6, 4))
    sns.boxplot(df[column])
    plt.title(f'Boxplot of {column}')
    plt.show()
# Draw distribution plots for all features
for column in df.columns:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[column], bins=30, kde=True)
    plt.title(f'Distribution of {column}')
    plt.show()
```


6000
4000
2000
0
0


Boxplot of Grocery


20000
15000
10000
5000
0
0


Boxplot of Frozen
8000
7000
6000
5000
4000
3000
2000
1000
0
0


Boxplot of Detergents_Paper
8000
6000
4000
2000
0