Experiment No. 5
Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset
Date of Performance: 21/08/2023
Date of Submission: 24/09/2023

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

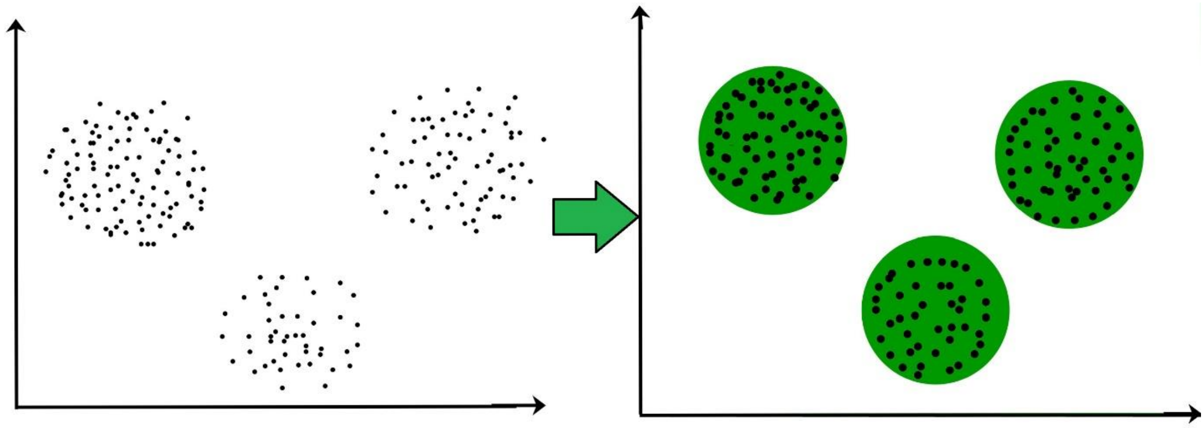
Objective: Able to perform various feature engineering tasks, apply Clustering Algorithm on the given dataset.

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:

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/content/customers.csv'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Import necessary libraries
import pandas as pd

# Define a function to load the data
def load_data(path):
    """
    This function reads a csv file from a specified path and returns a
pandas DataFrame.

    Parameters:
    path (str): The path to the csv file.

    Returns:
    DataFrame: A pandas DataFrame containing the data from the csv
file.
    """
    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/customers.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  \
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

   Delicatessen
0          1338
1          1776
2          7844
3          1788
4          5185
```

```python
print("Column names:")
print(df.columns)
```

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

```python
# 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
Delicatessen        int64
dtype: object
```

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

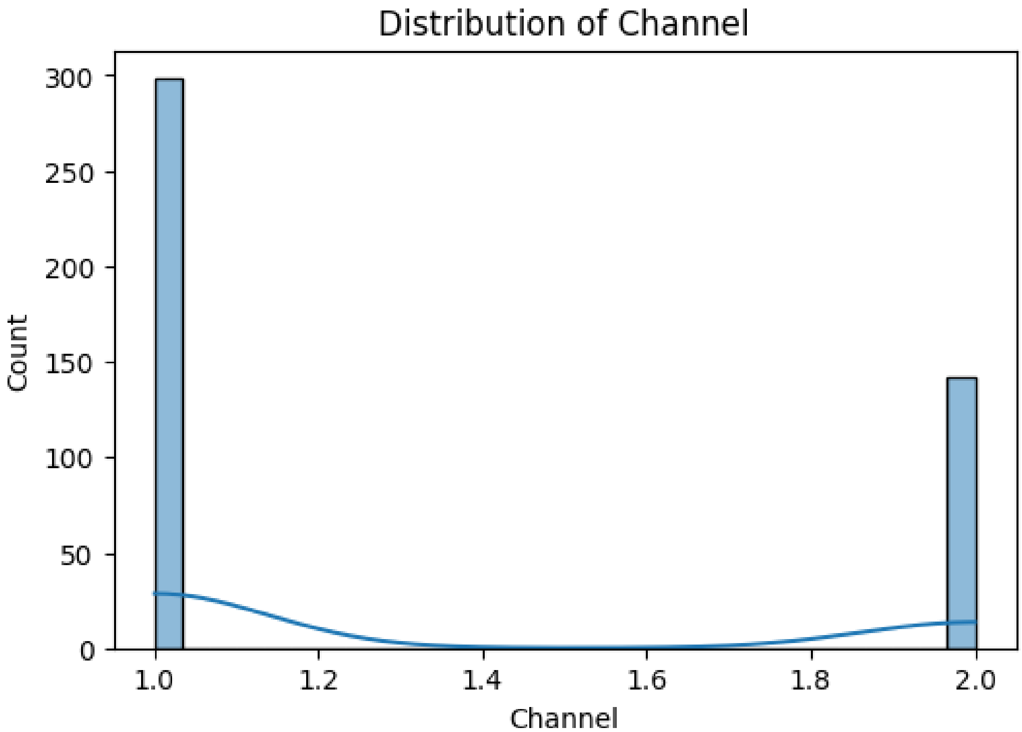
```
Missing values per column:
Channel             0
Region              0
Fresh               0
Milk                0
Grocery             0
Frozen              0
Detergents_Paper    0
Delicatessen        0
dtype: int64
```

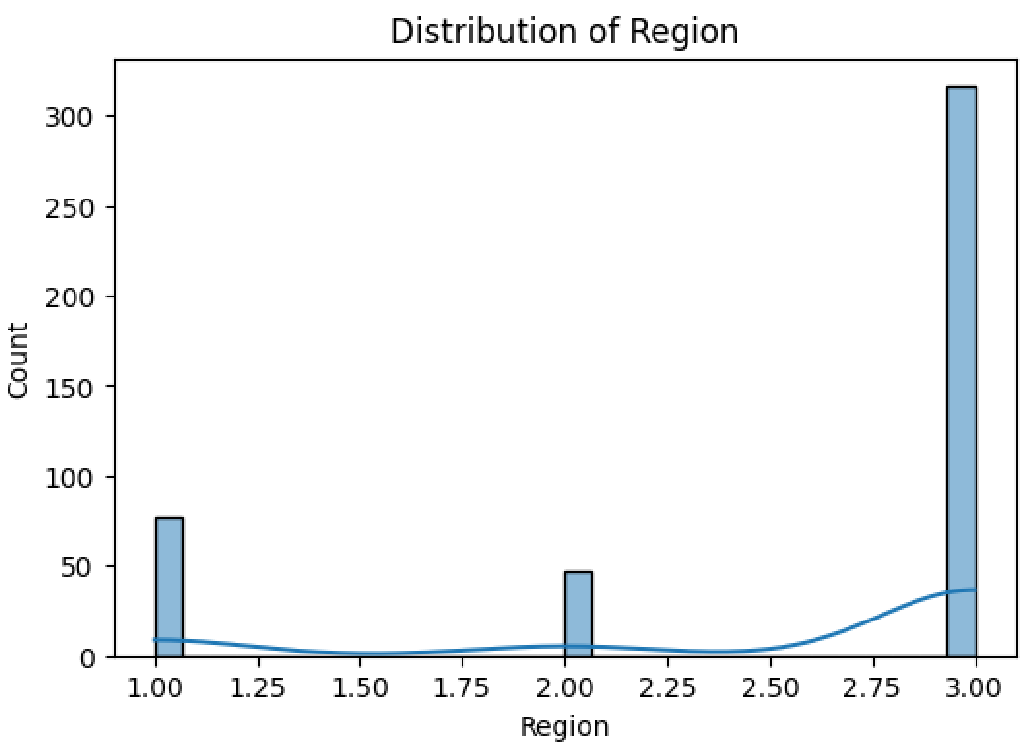
```python
# Import necessary libraries
```

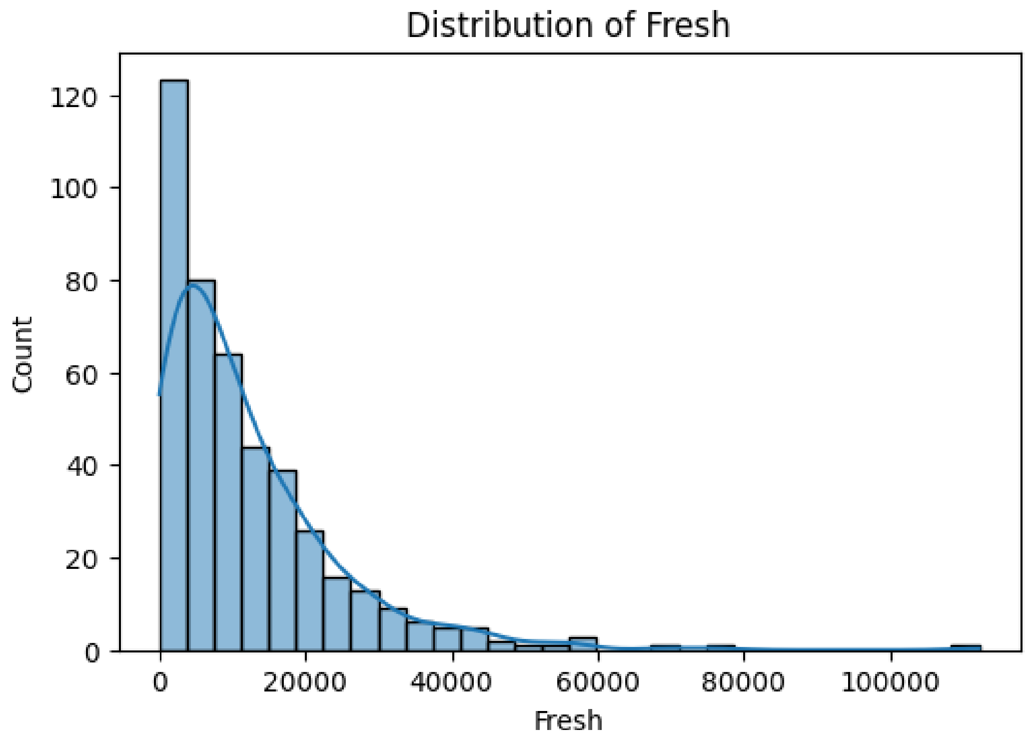
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Check descriptive statistics
print("Descriptive Statistics:")
print(df.describe())

# Check for duplicates
print("Number of duplicate rows: ", df.duplicated().sum())

# 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()
```

```
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
```

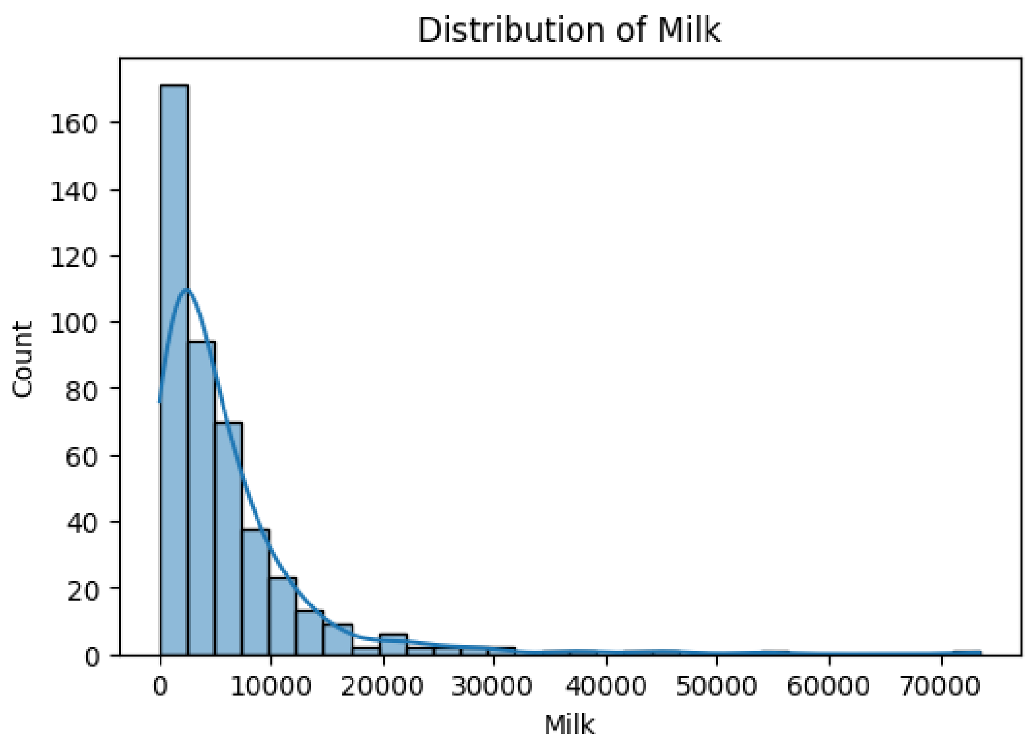
```
75%      3554.250000       3922.000000   1820.250000
max     60869.000000      40827.000000  47943.000000
Number of duplicate rows:  0
```



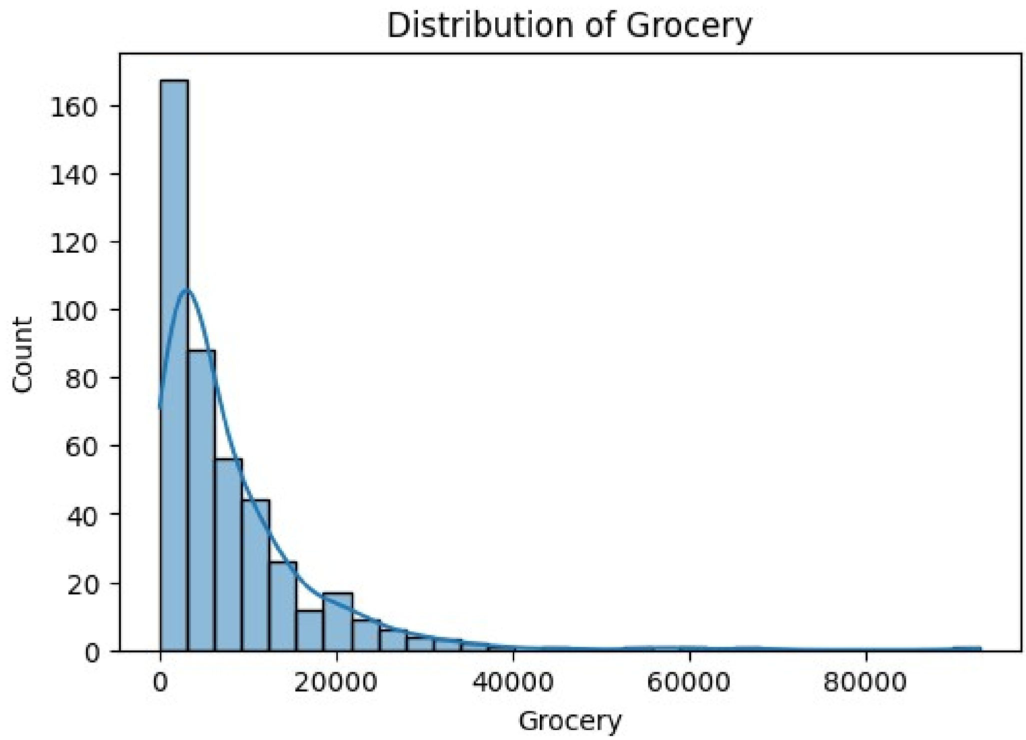


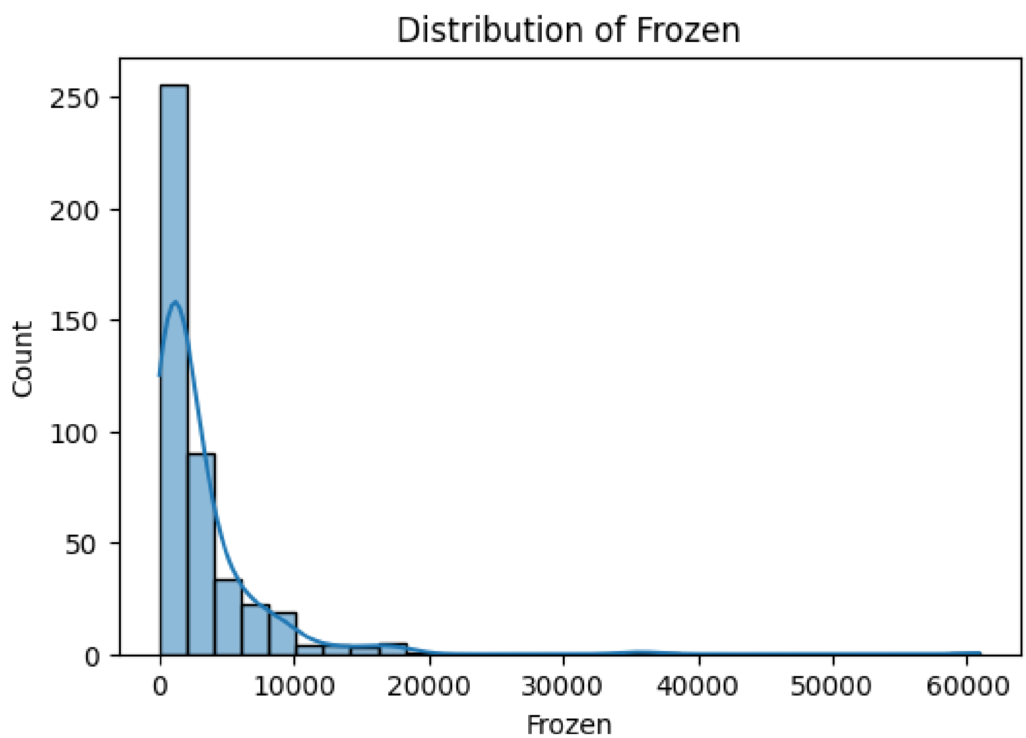


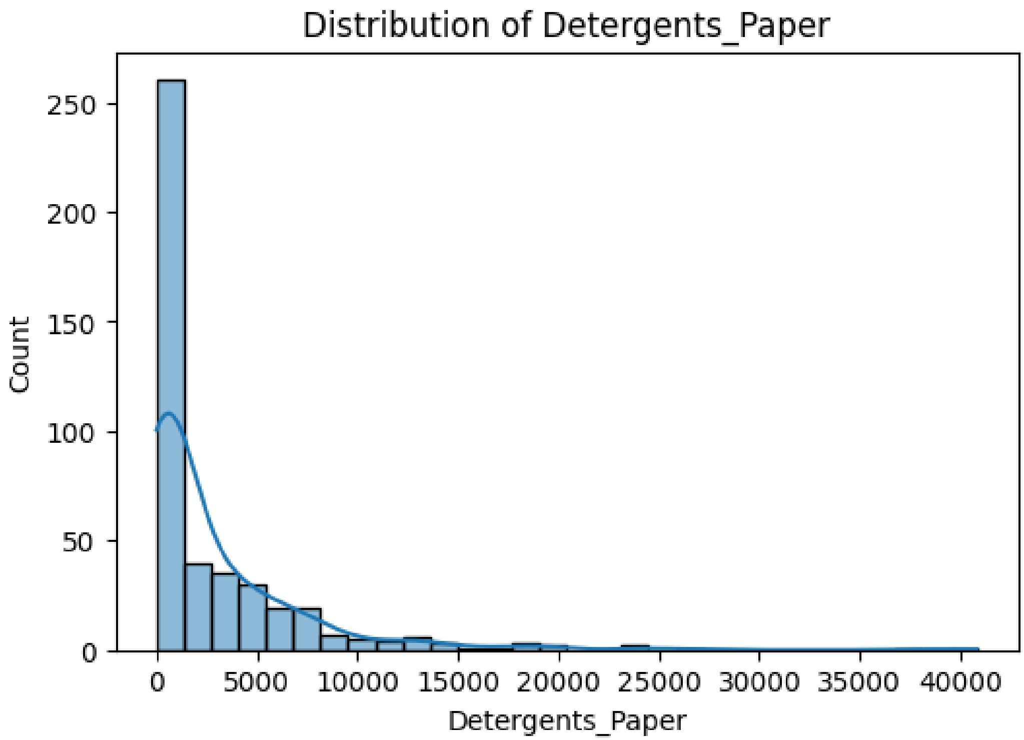
Distribution of Fresh
Count
120
100
80
60
40
20
0
0
20000
40000
60000
80000
100000
Fresh



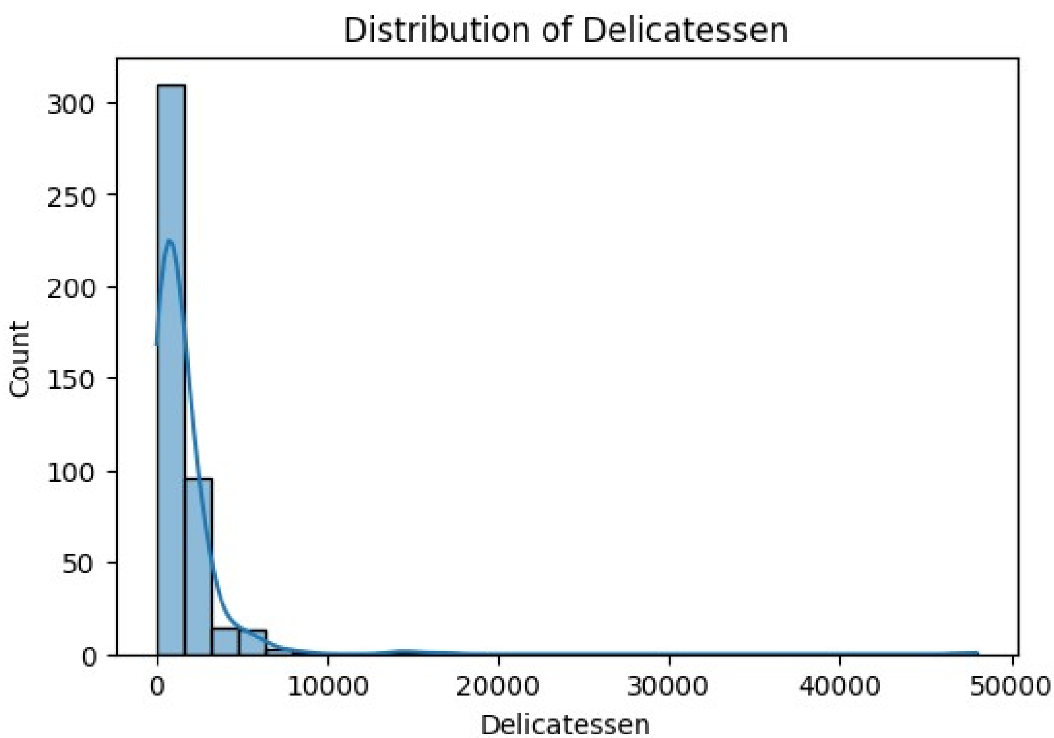
Distribution of Milk
Count
160
140
120
100
80
60
40
20
0
0
10000
20000
30000
40000
50000
60000
70000
Milk


Distribution of Grocery
Count
Grocery
0
20
40
60
80
100
120
140
160
0
20000
40000
60000
80000

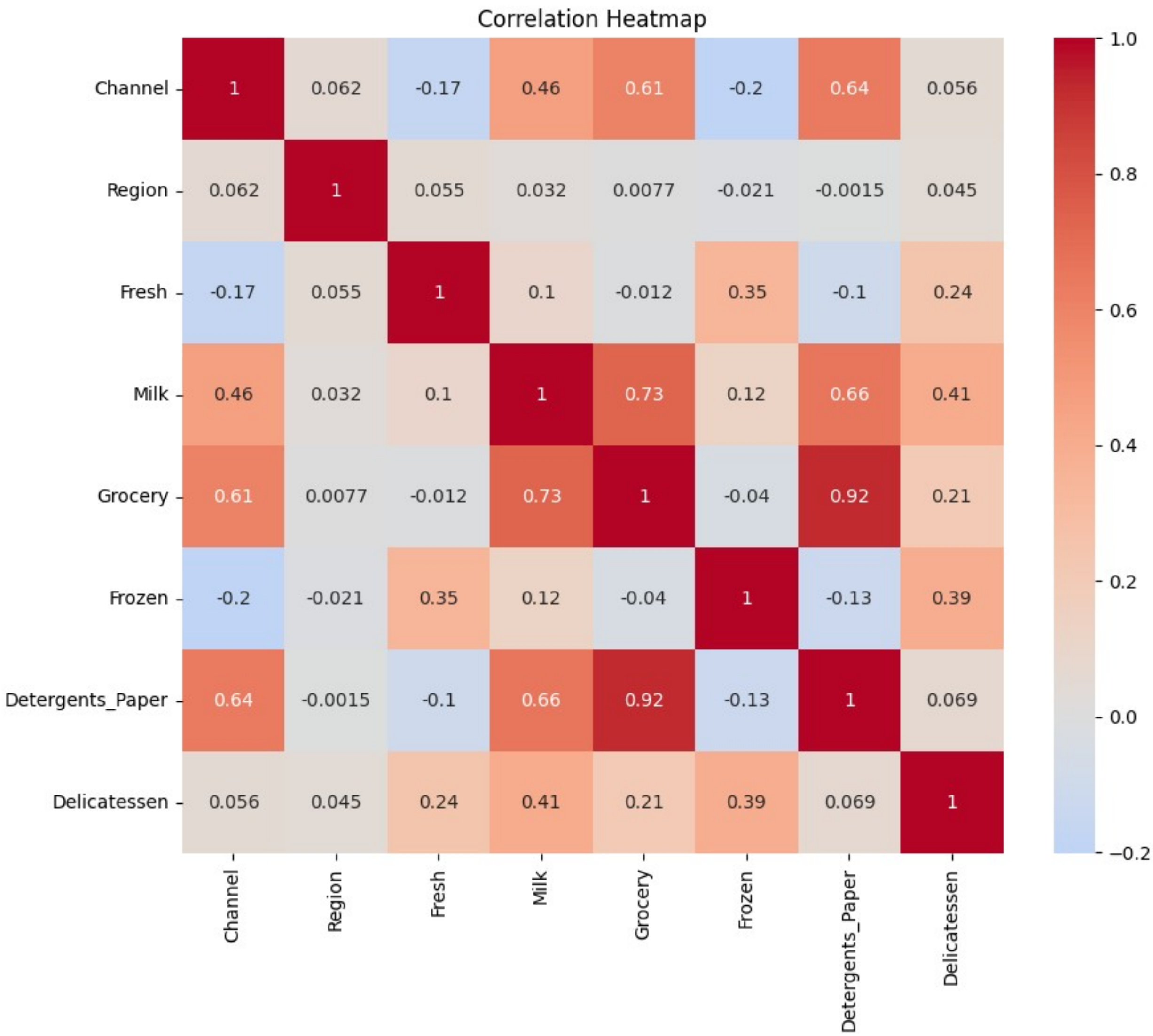

Distribution of Frozen
Count
Frozen
0
50
100
150
200
250
0
10000
20000
30000
40000
50000
60000



Distribution of Detergents_Paper
Count
250
200
150
100
50
0
0
5000
10000
15000
20000
25000
30000
35000
40000
Detergents_Paper

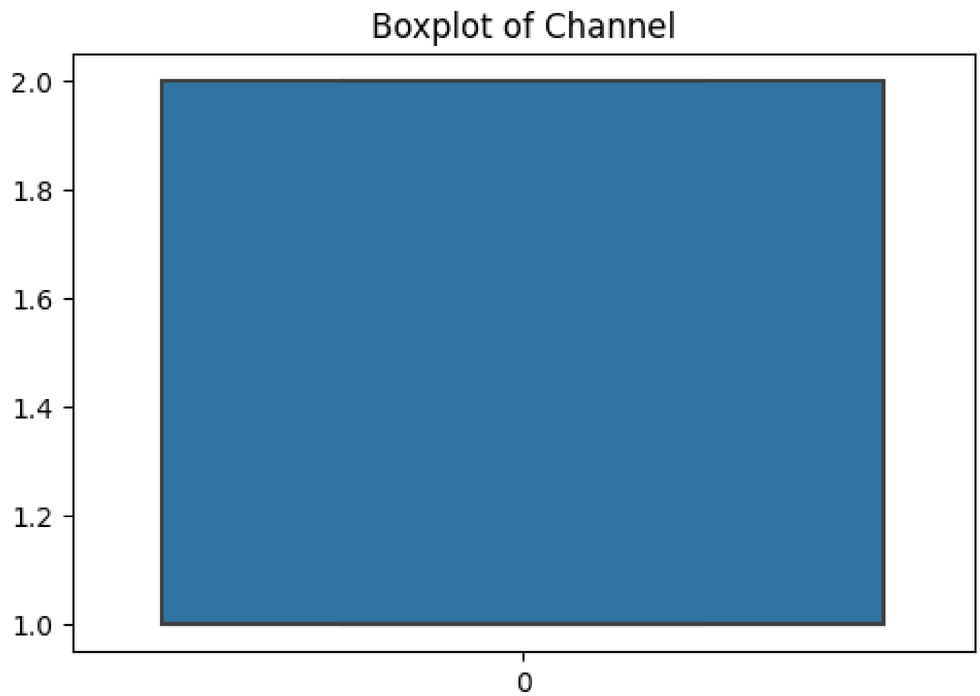


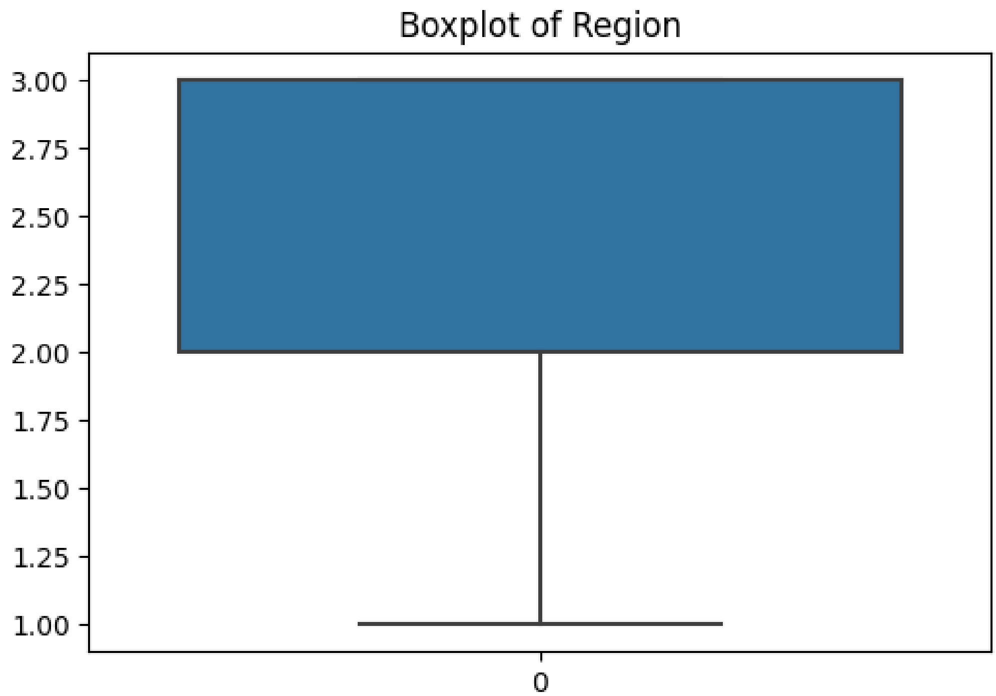
Distribution of Delicatessen
Count
300
250
200
150
100
50
0
0
10000
20000
30000
40000
50000
Delicatessen

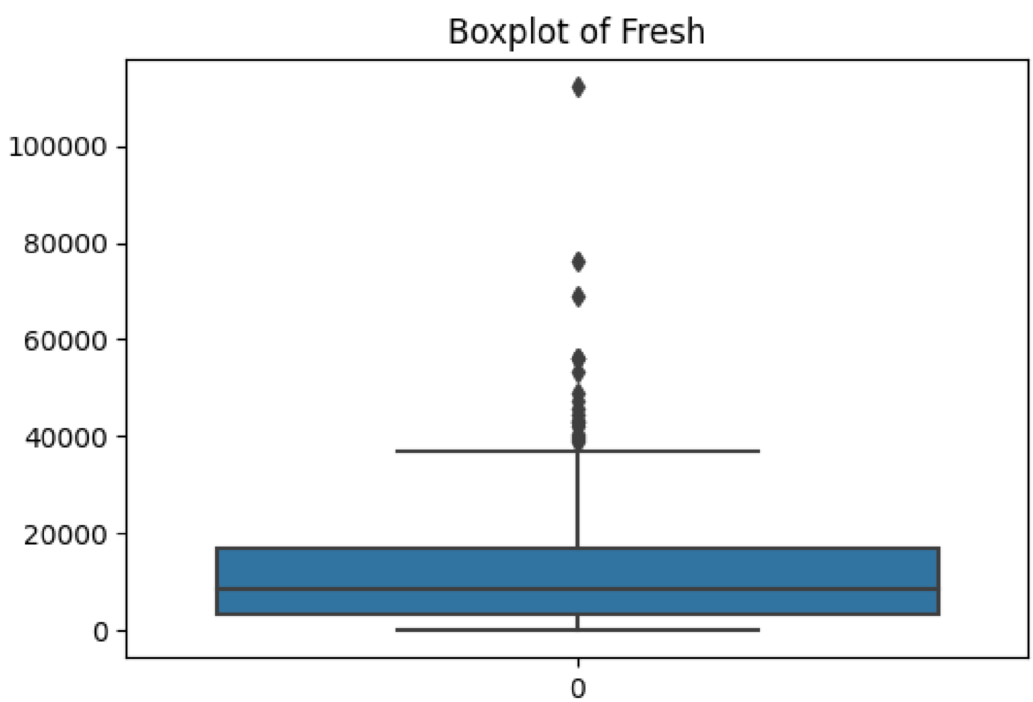


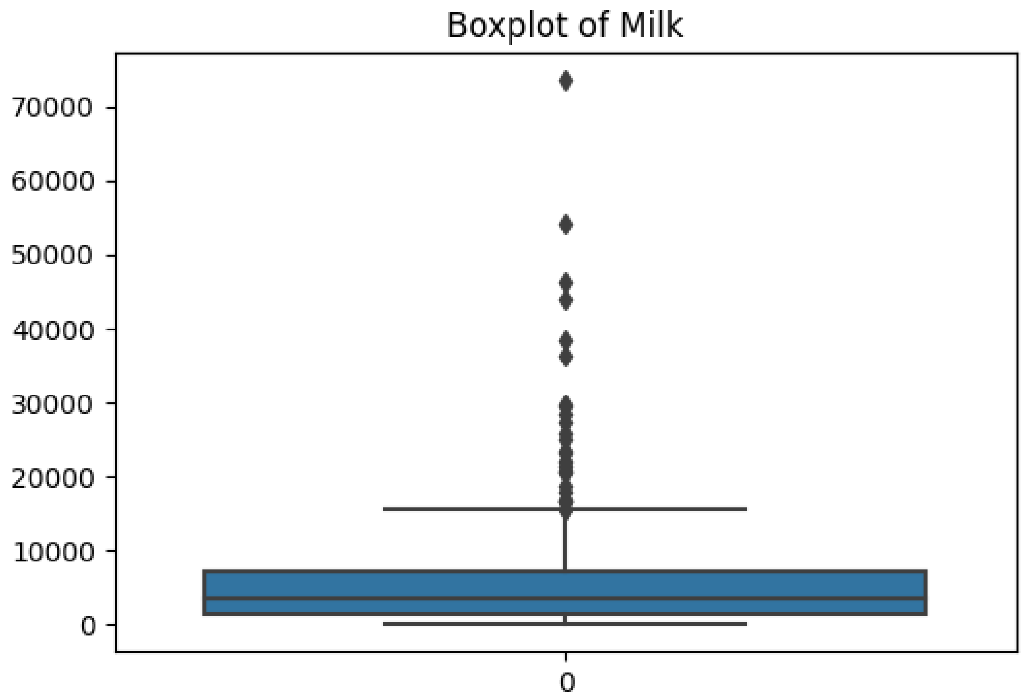

```python
# checking for outliers
import seaborn as sns
import matplotlib.pyplot as plt

# 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
```

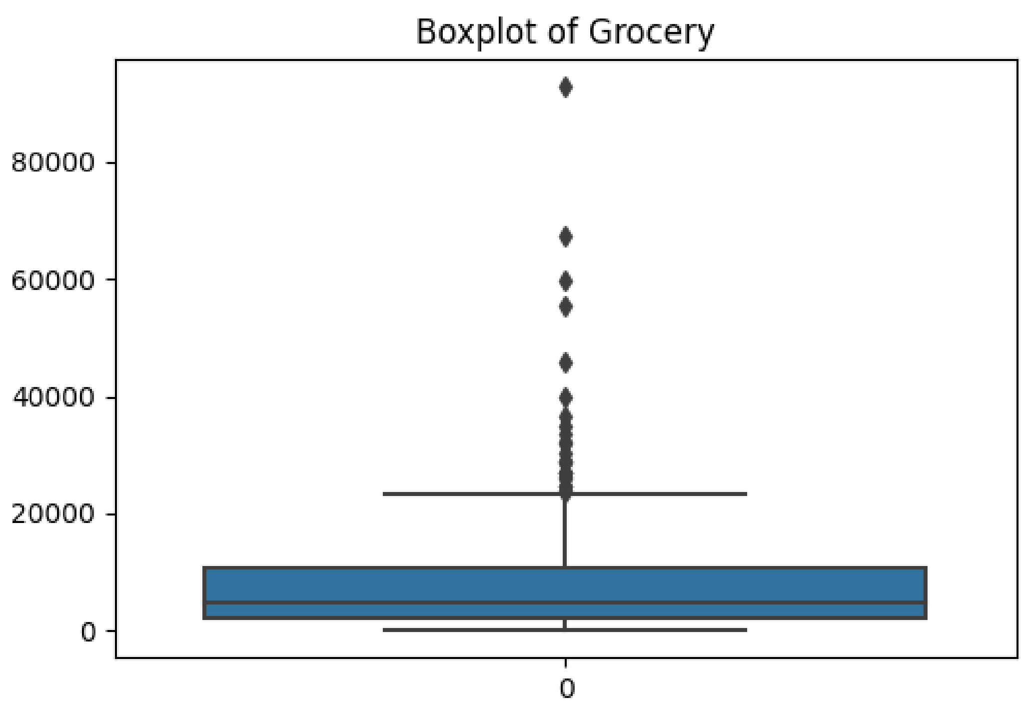
```
    outliers = dataframe[(dataframe[column] < Q1 - 1.5*IQR) |
(dataframe[column] > Q3 + 1.5*IQR)]
    return outliers

# 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)}')
```

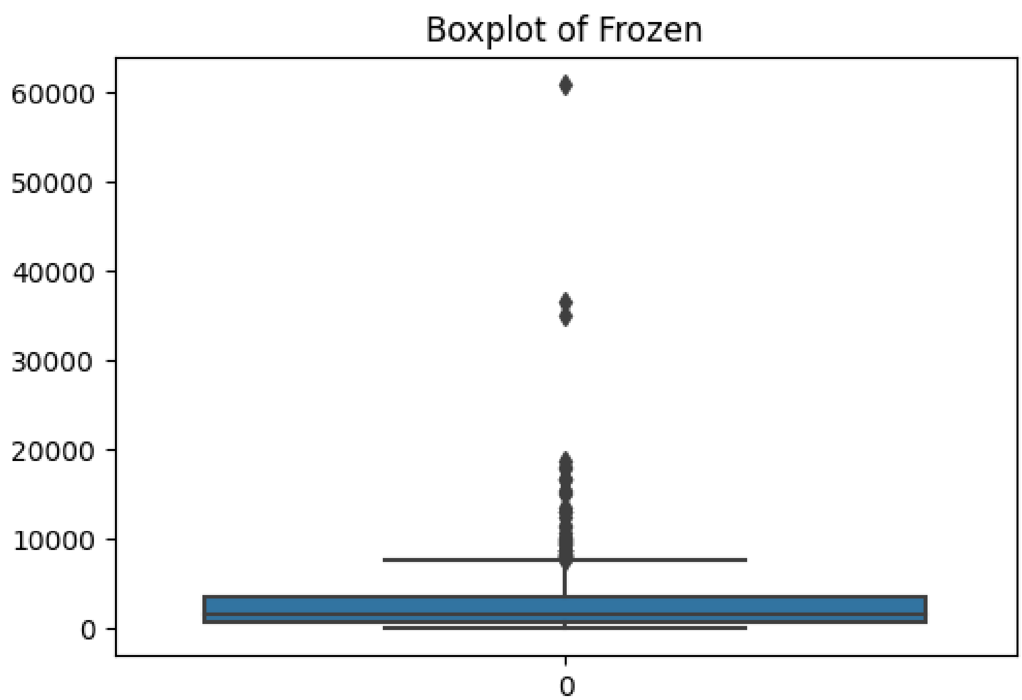



Boxplot of Region
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0

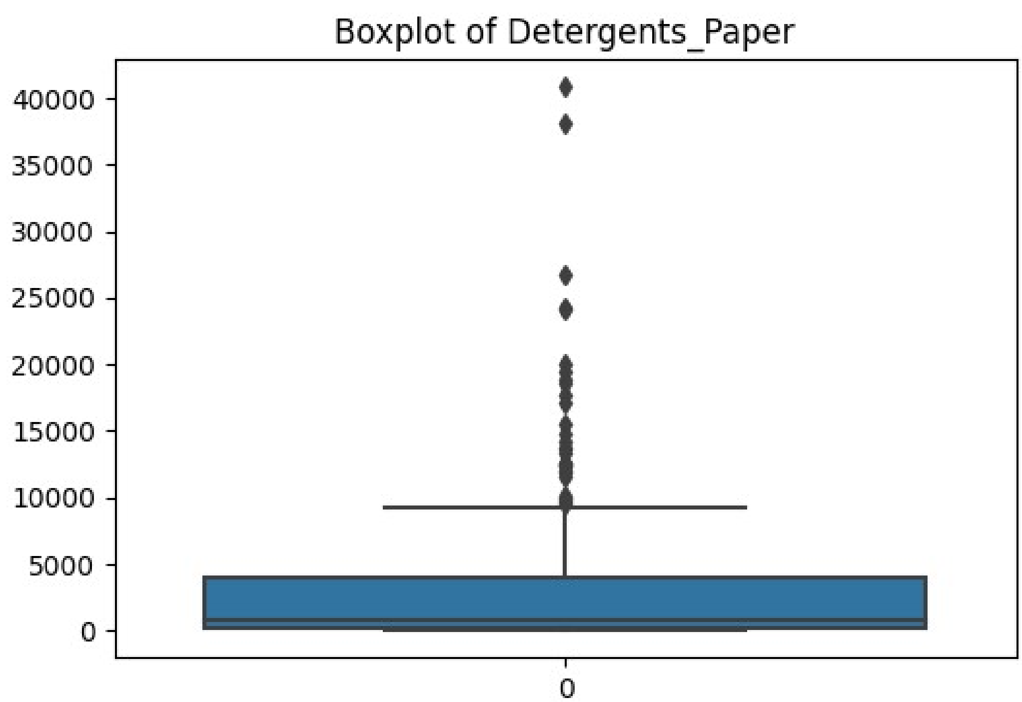

Boxplot of Fresh
100000
80000
60000
40000
20000
0
0


Boxplot of Milk
70000
60000
50000
40000
30000
20000
10000
0
0

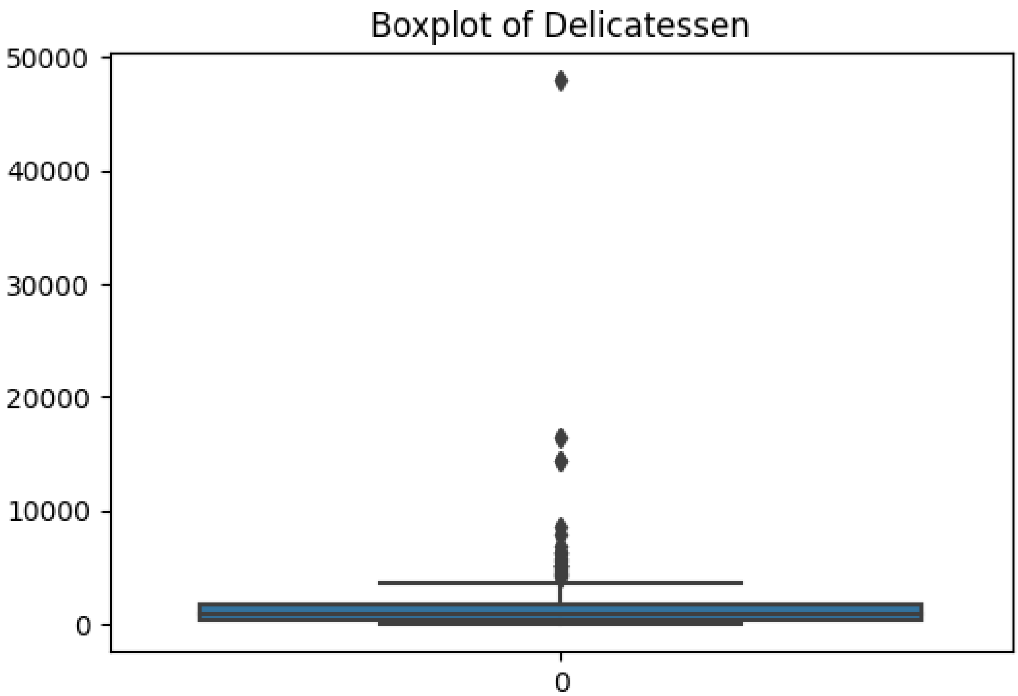

Boxplot of Grocery
80000
60000
40000
20000
0
0



Boxplot of Frozen
60000
50000
40000
30000
20000
10000
0
0

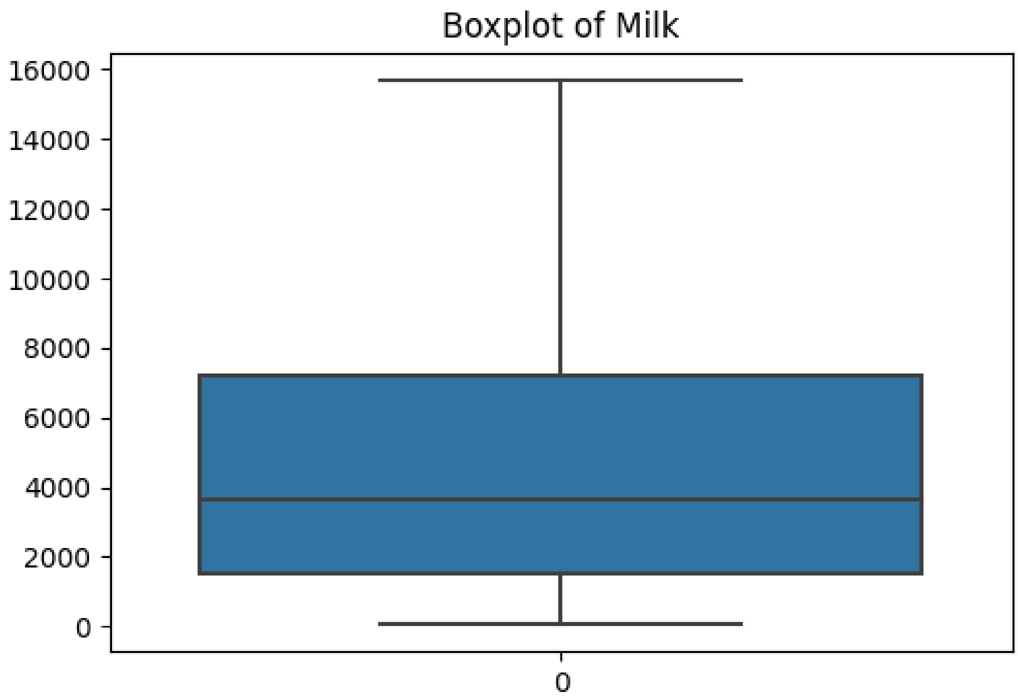


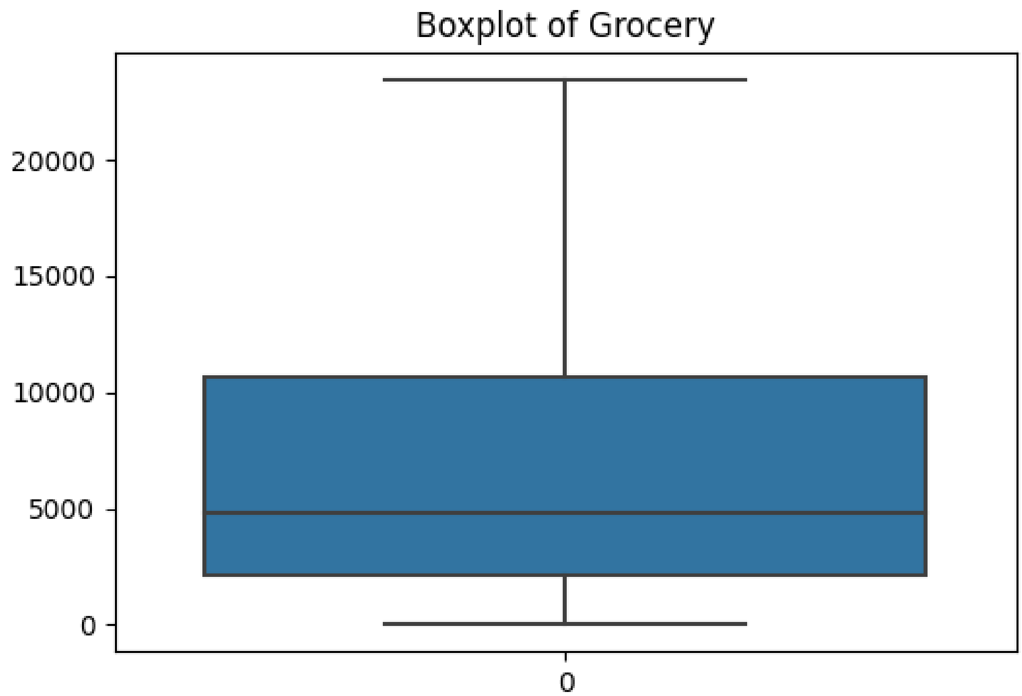
Boxplot of Detergents_Paper
40000
35000
30000
25000
20000
15000
10000
5000
0
0

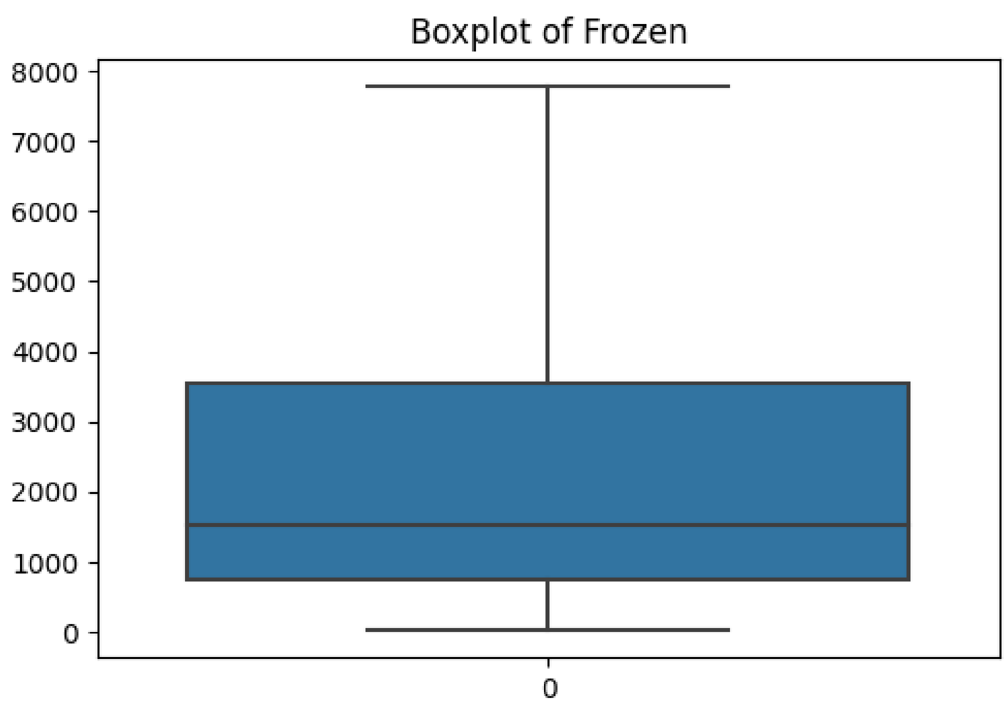


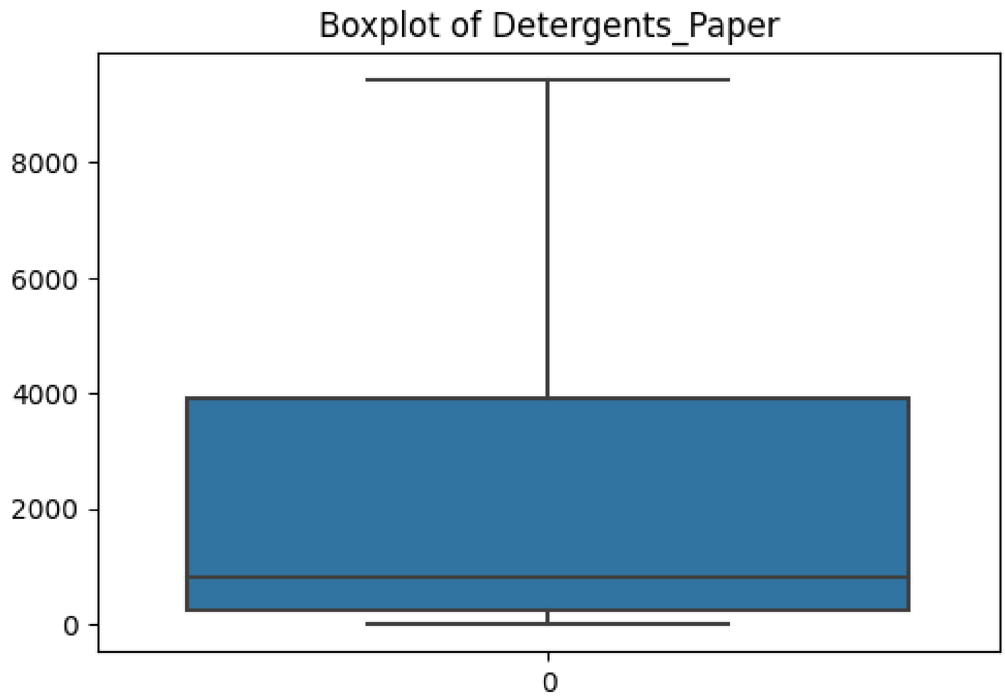

```
Number of outliers in Channel: 0
Number of outliers in Region: 0
Number of outliers in Fresh: 20
Number of outliers in Milk: 28
Number of outliers in Grocery: 24
Number of outliers in Frozen: 43
Number of outliers in Detergents_Paper: 30
Number of outliers in Delicatessen: 27

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:
```

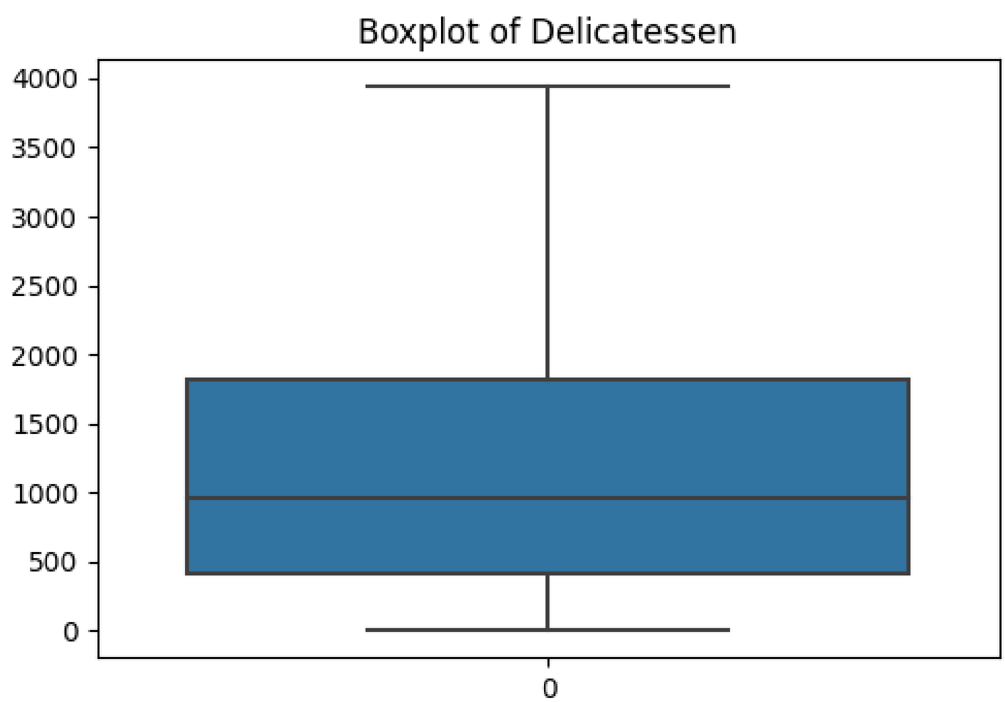
```
    plt.figure(figsize=(6, 4))
    sns.boxplot(df[column])
    plt.title(f'Boxplot of {column}')
    plt.show()

# 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()
```

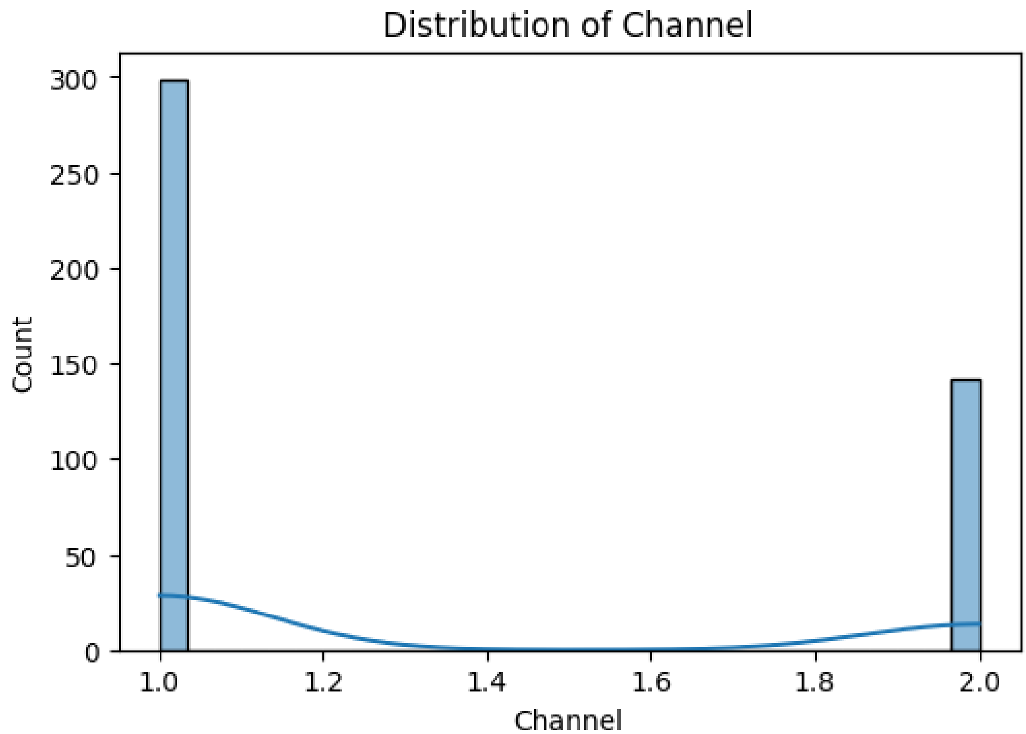


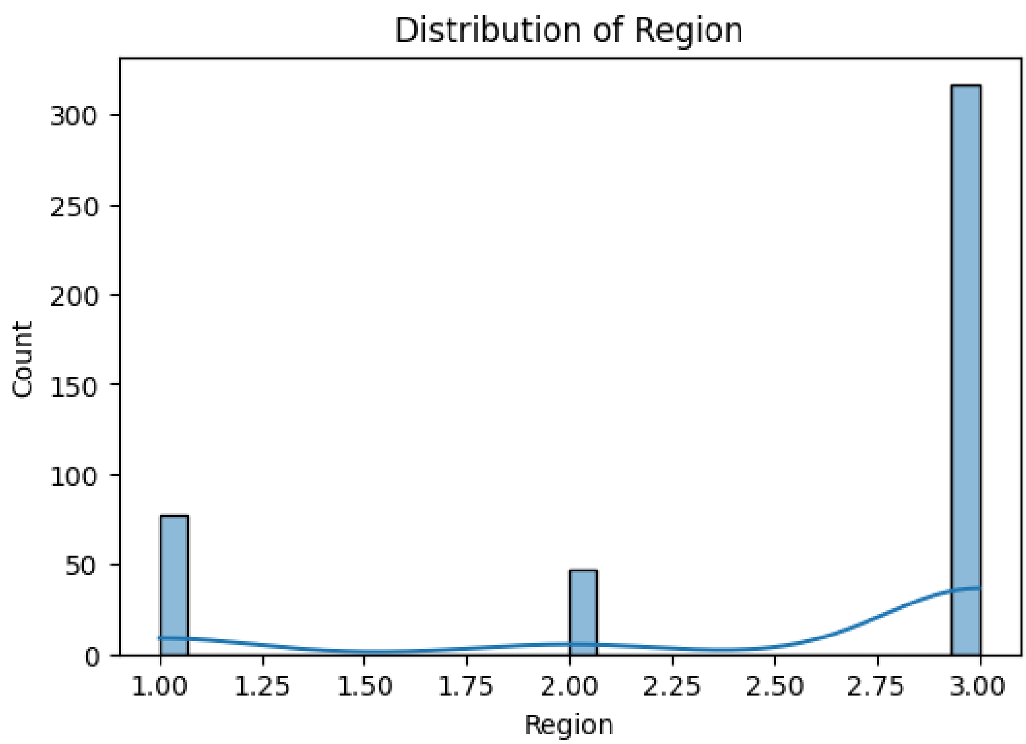

Boxplot of Grocery
20000
15000
10000
5000
0
0

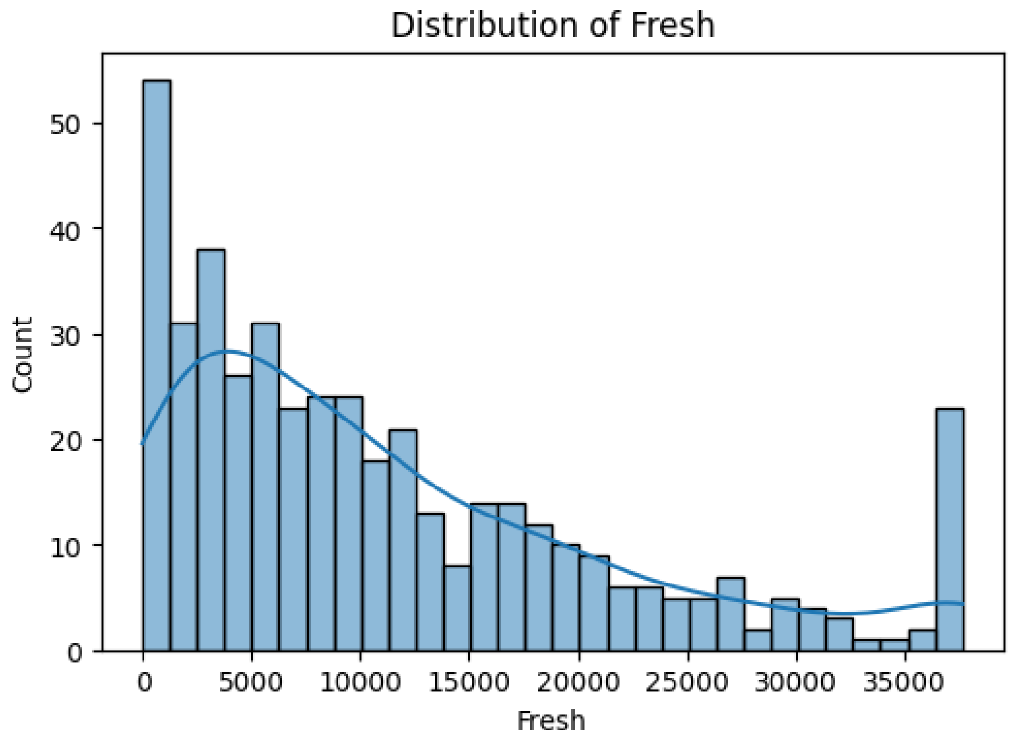

Boxplot of Frozen
8000
7000
6000
5000
4000
3000
2000
1000
0
0

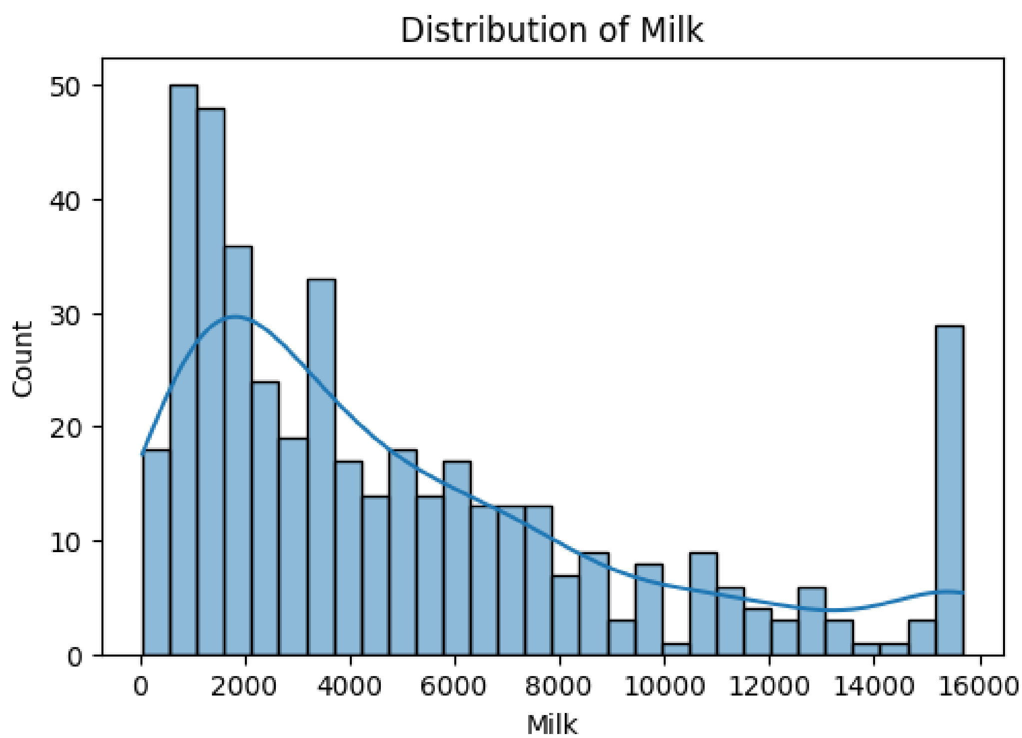

Boxplot of Detergents_Paper
8000
6000
4000
2000
0
0

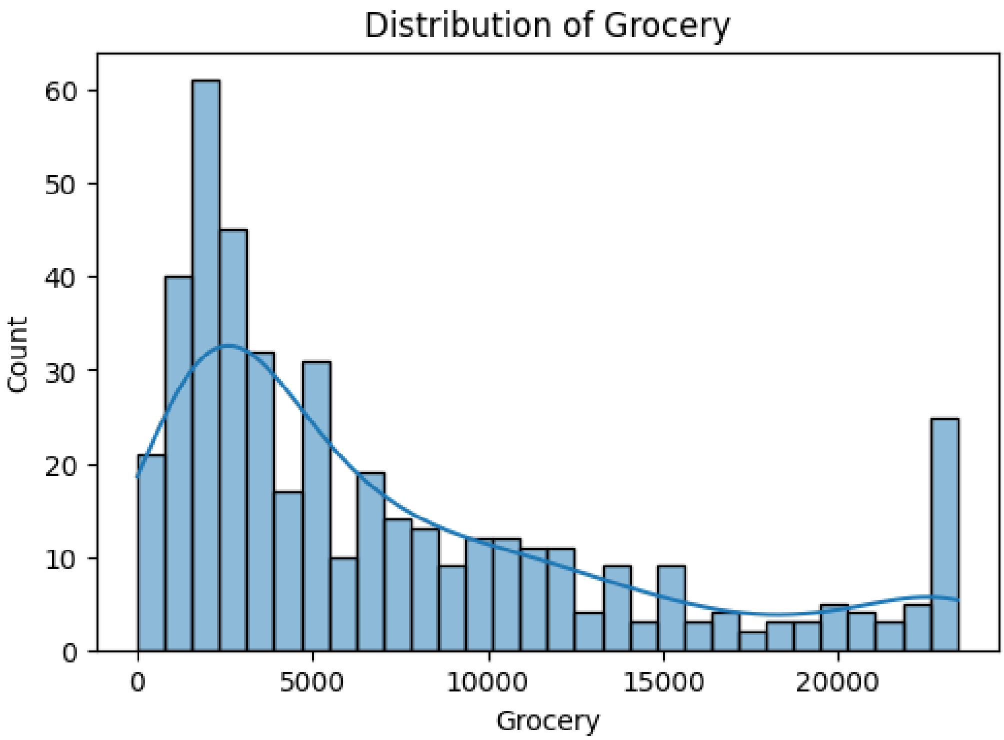

Boxplot of Delicatessen
4000
3500
3000
2500
2000
1500
1000
500
0
0


Distribution of Channel
300
250
200
150
100
50
0
Count
1.0
1.2
1.4
1.6
1.8
2.0
Channel

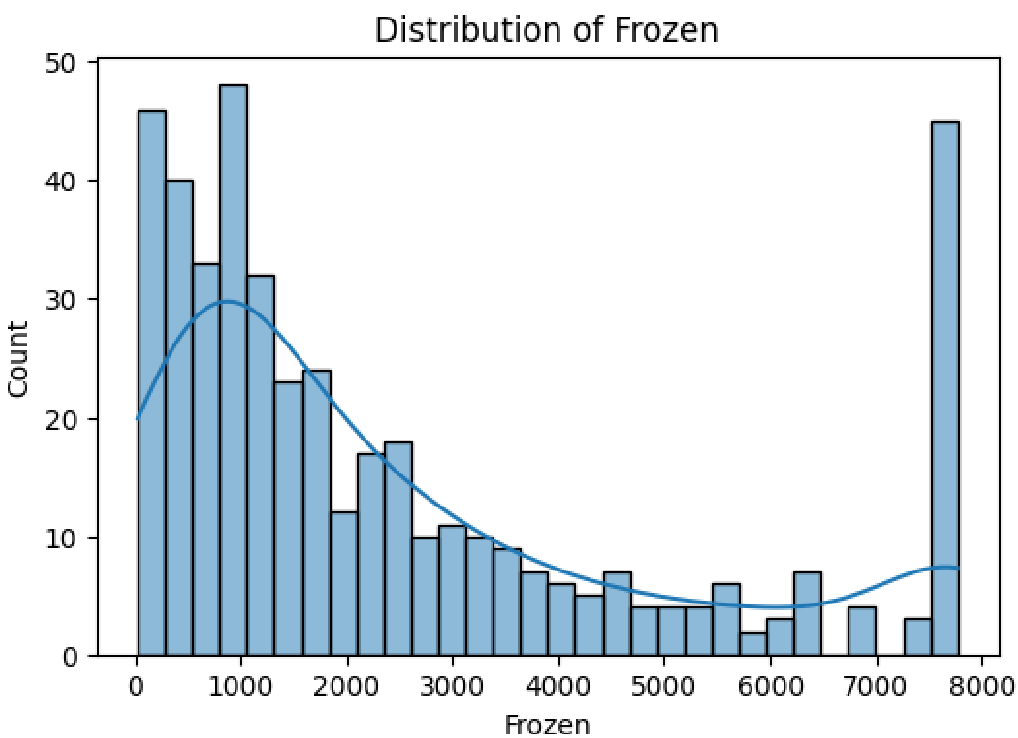

Distribution of Region
300
250
200
150
100
50
0
Count
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Region


Distribution of Fresh
Count
50
40
30
20
10
0
0
5000
10000
15000
20000
25000
30000
35000
Fresh

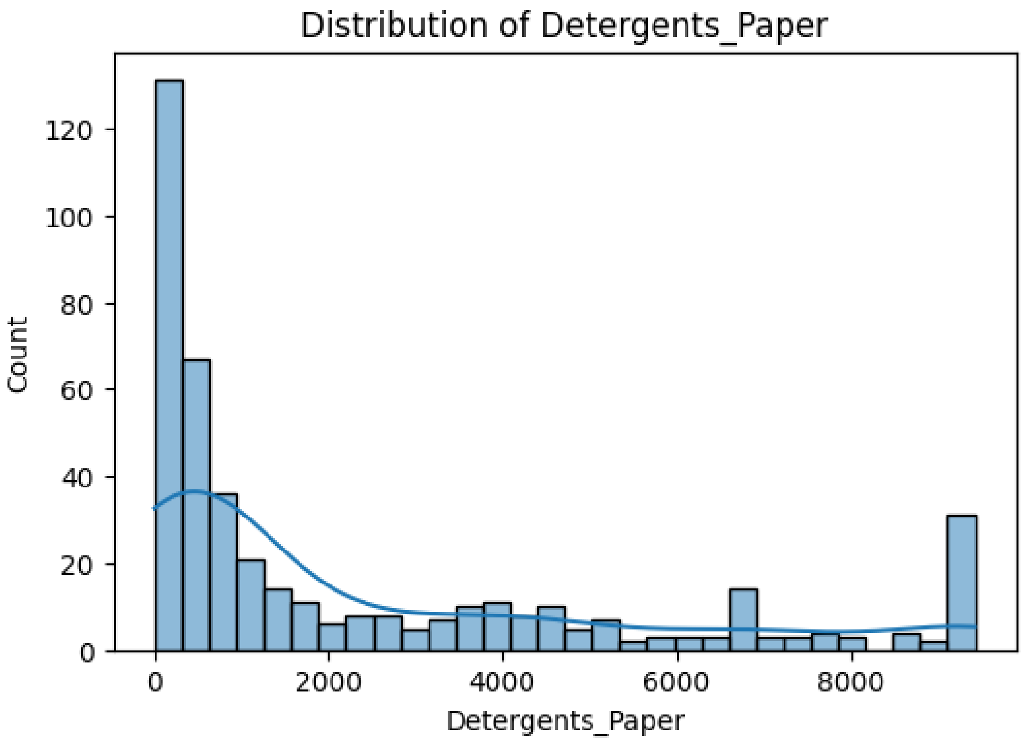

Distribution of Milk
Count
50
40
30
20
10
0
0
2000
4000
6000
8000
10000
12000
14000
16000
Milk

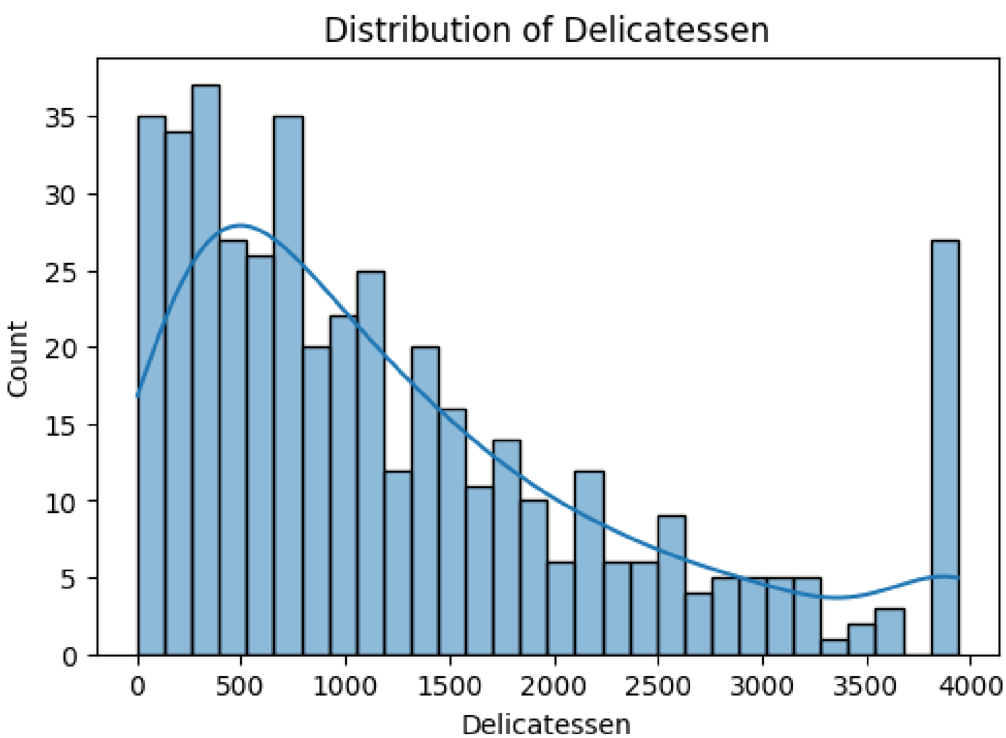


Distribution of Grocery
Count
Grocery
0
10
20
30
40
50
60
0
5000
10000
15000
20000



Distribution of Frozen
Count
Frozen
0
10
20
30
40
50
0
1000
2000
3000
4000
5000
6000
7000
8000






```
# Function to detect outliers
def detect_outliers(dataframe, column):
    Q1 = dataframe[column].quantile(0.25)
```

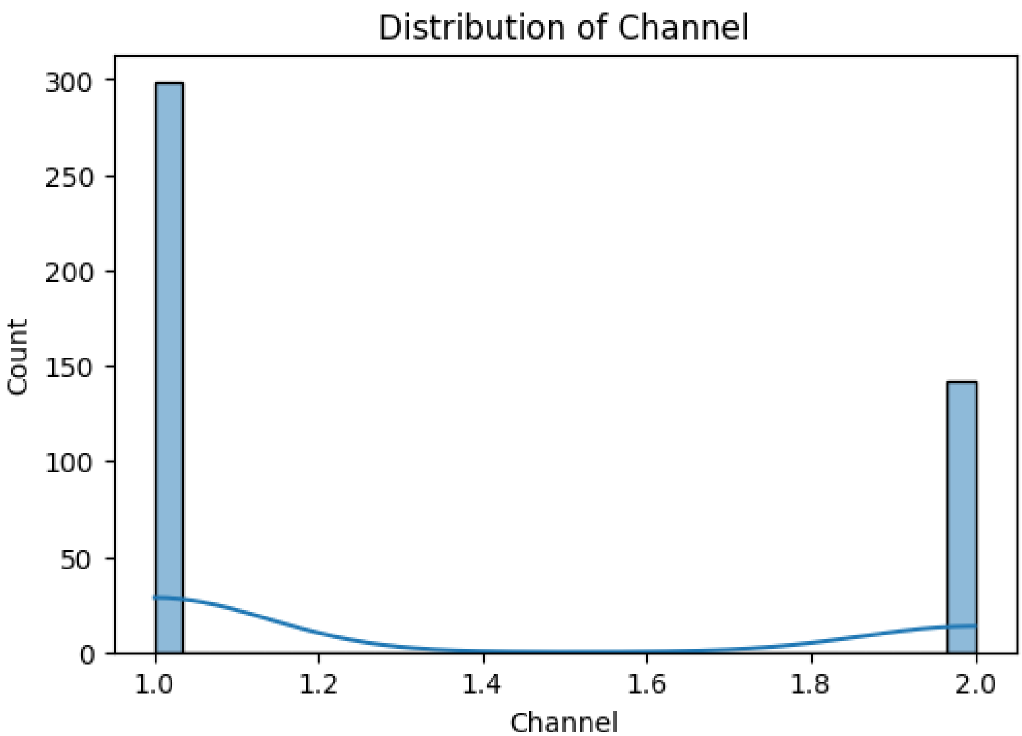
```
    Q3 = dataframe[column].quantile(0.75)
    IQR = Q3 - Q1
    outliers = dataframe[(dataframe[column] < Q1 - 1.5*IQR) |
(dataframe[column] > Q3 + 1.5*IQR)]
    return outliers

# 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)}')
```

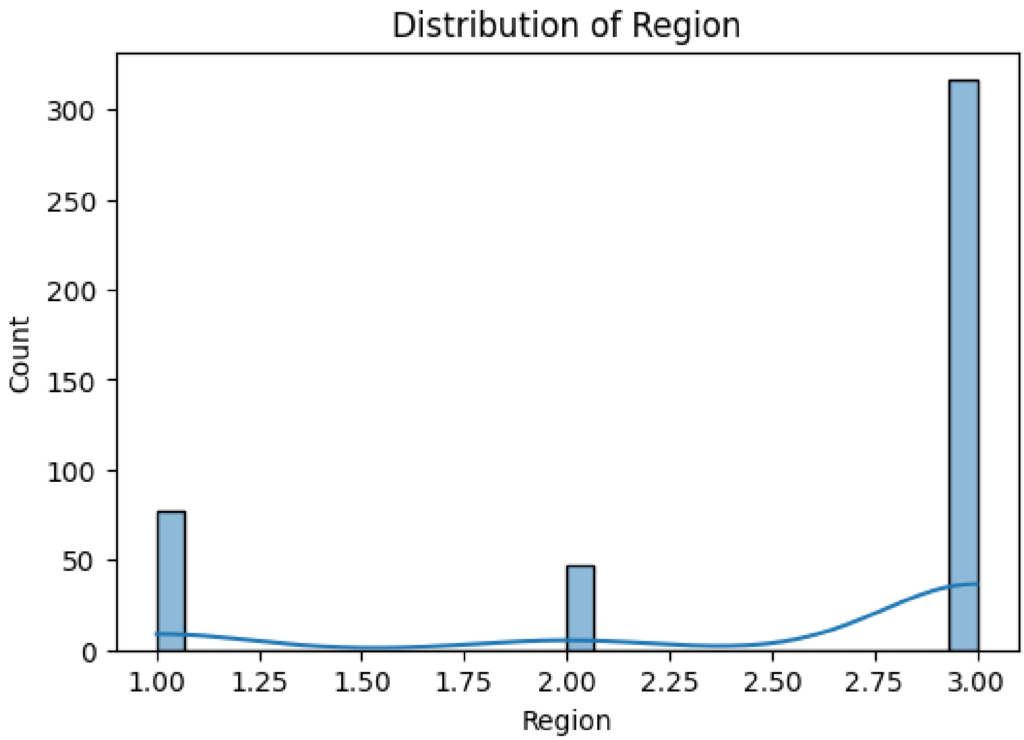
```
Number of outliers in Channel: 0
Number of outliers in Region: 0
Number of outliers in Fresh: 0
Number of outliers in Milk: 0
Number of outliers in Grocery: 0
Number of outliers in Frozen: 0
Number of outliers in Detergents_Paper: 0
Number of outliers in Delicatessen: 0
```

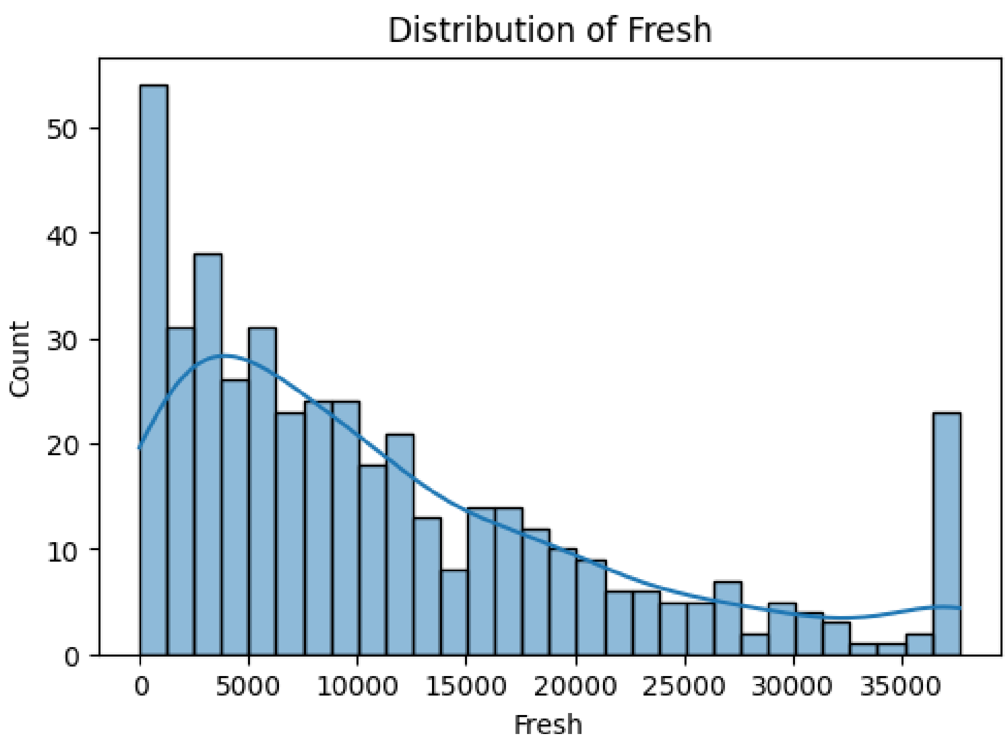
```
# Check descriptive statistics
print("Descriptive Statistics:")
print(df.describe())

# Check for duplicates
print("Number of duplicate rows: ", df.duplicated().sum())

# 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()
```

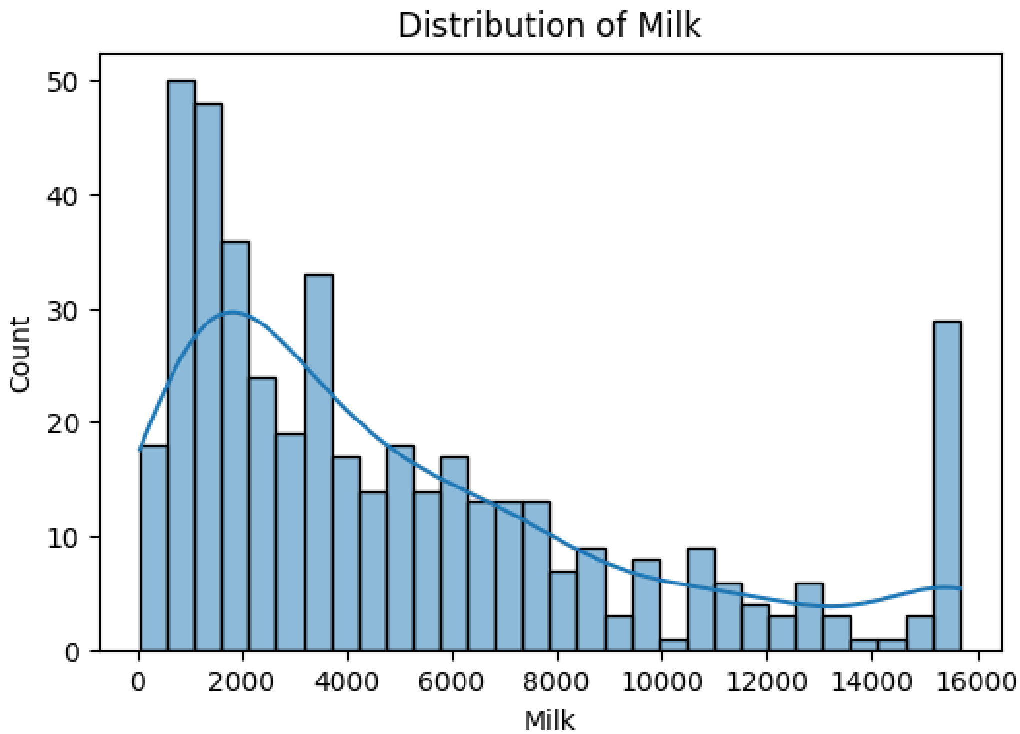
```
Descriptive Statistics:
          Channel      Region         Fresh          Milk      Grocery
\
count  440.000000  440.000000    440.000000    440.000000    440.00000

mean     1.322727    2.543182  11357.568182   5048.592045   7236.37500

std      0.468052    0.774272  10211.542235   4386.377073   6596.53308

min      1.000000    1.000000      3.000000     55.000000      3.00000
```

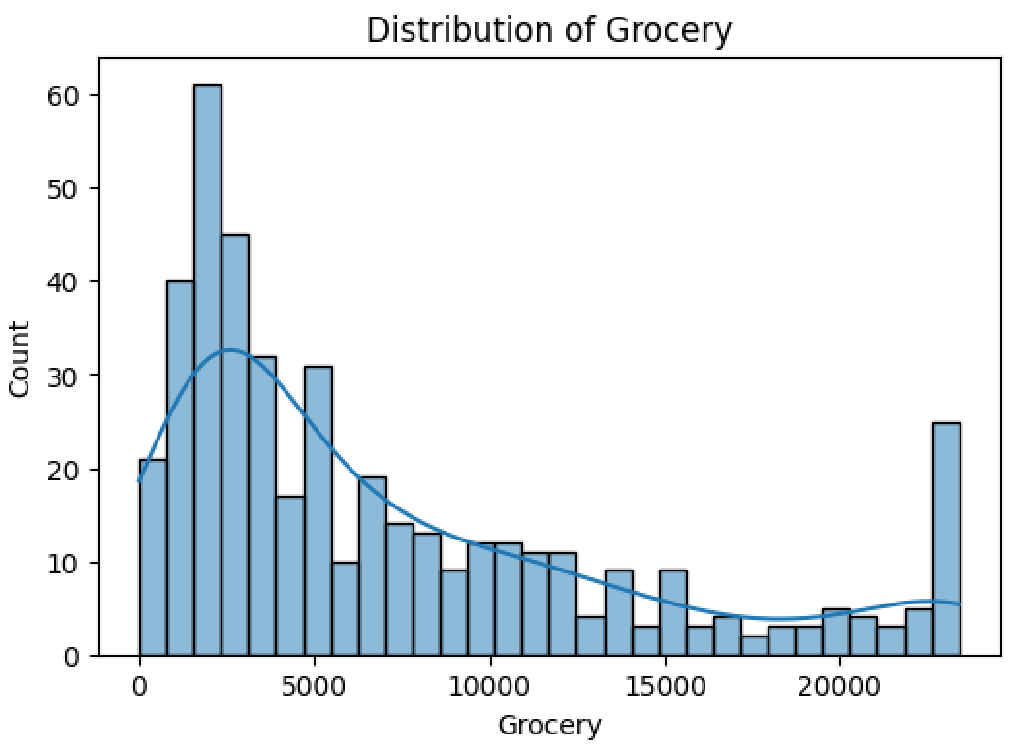
```
25%       1.000000    2.000000   3127.750000   1533.000000   2153.00000

50%       1.000000    3.000000   8504.000000   3627.000000   4755.50000

75%       2.000000    3.000000  16933.750000   7190.250000  10655.75000

max       2.000000    3.000000  37642.750000  15676.125000  23409.87500


            Frozen  Detergents_Paper  Delicatessen
count   440.000000        440.000000    440.000000
mean   2507.085795       2392.616477   1266.715341
std    2408.297738       2940.794090   1083.069792
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    7772.250000       9419.875000   3938.250000
Number of duplicate rows:  0
```

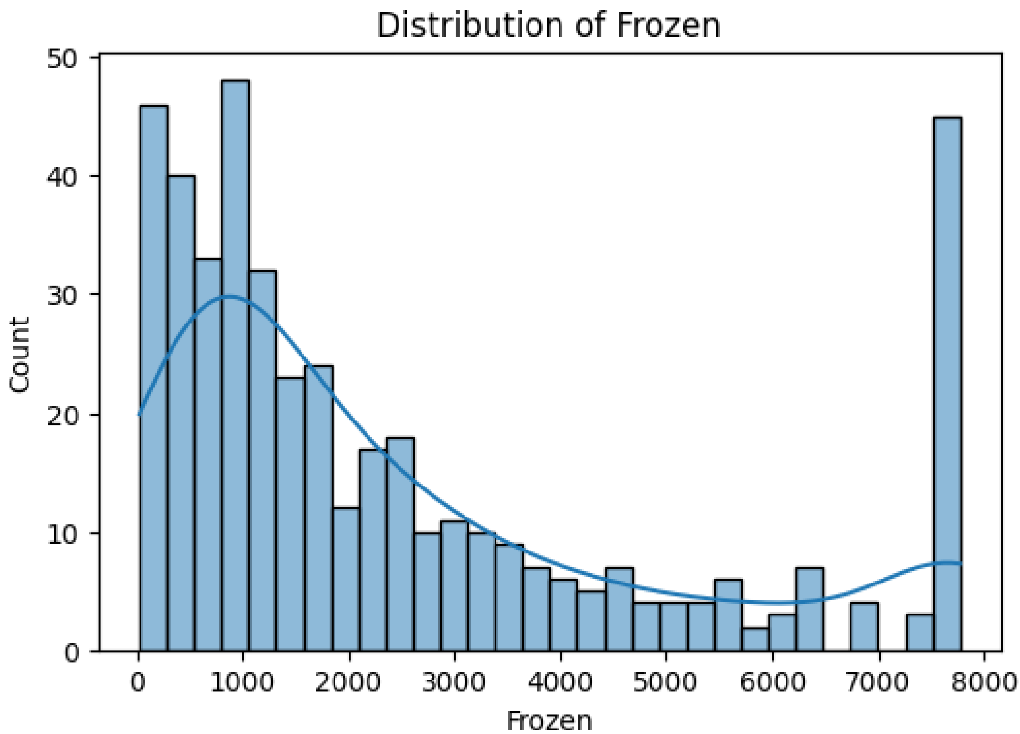



Distribution of Region
Count
Region
0
50
100
150
200
250
300
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

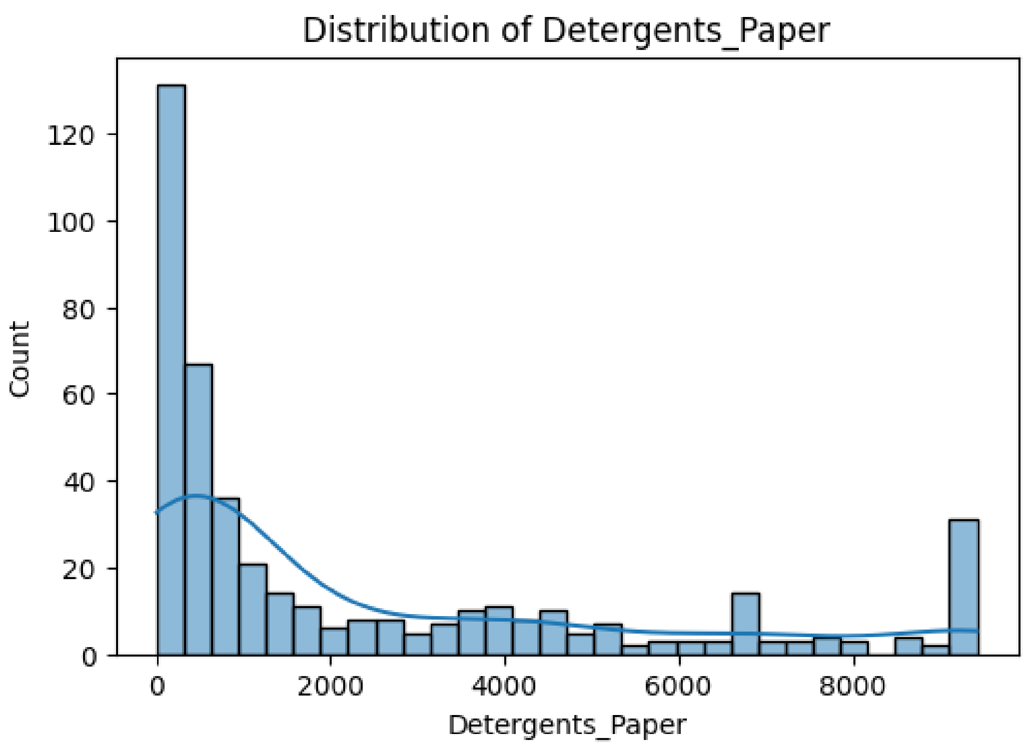

Distribution of Fresh
Count
Fresh
0
10
20
30
40
50
0
5000
10000
15000
20000
25000
30000
35000


Distribution of Milk
Count
Milk
0
10
20
30
40
50
0
2000
4000
6000
8000
10000
12000
14000
16000

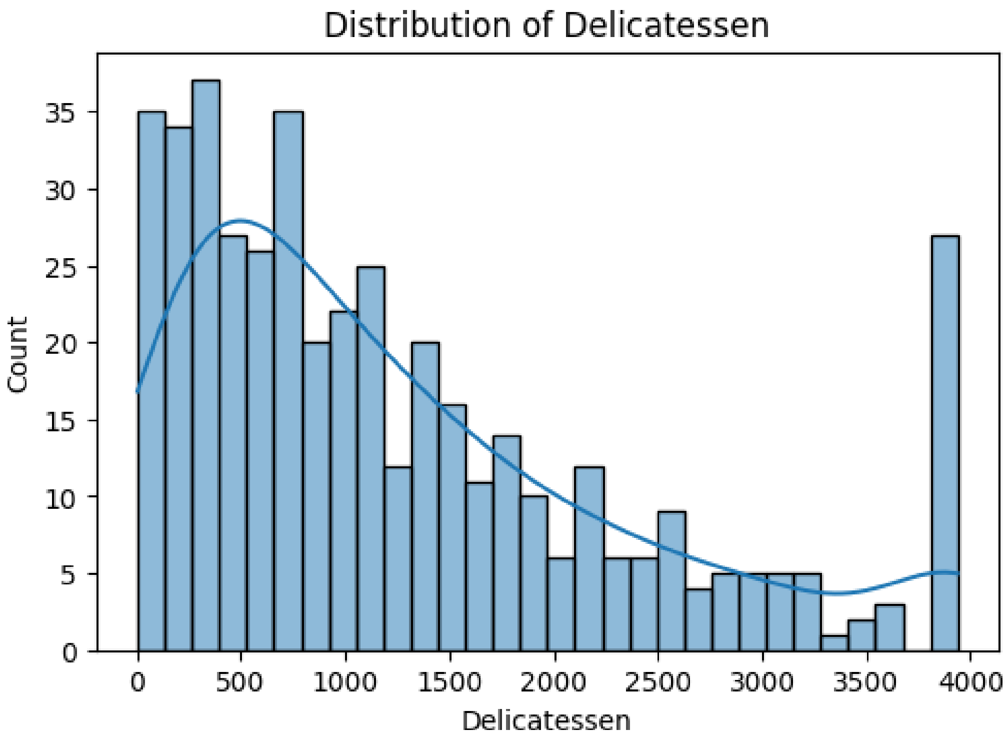

Distribution of Grocery
Count
Grocery
0
10
20
30
40
50
60
0
5000
10000
15000
20000

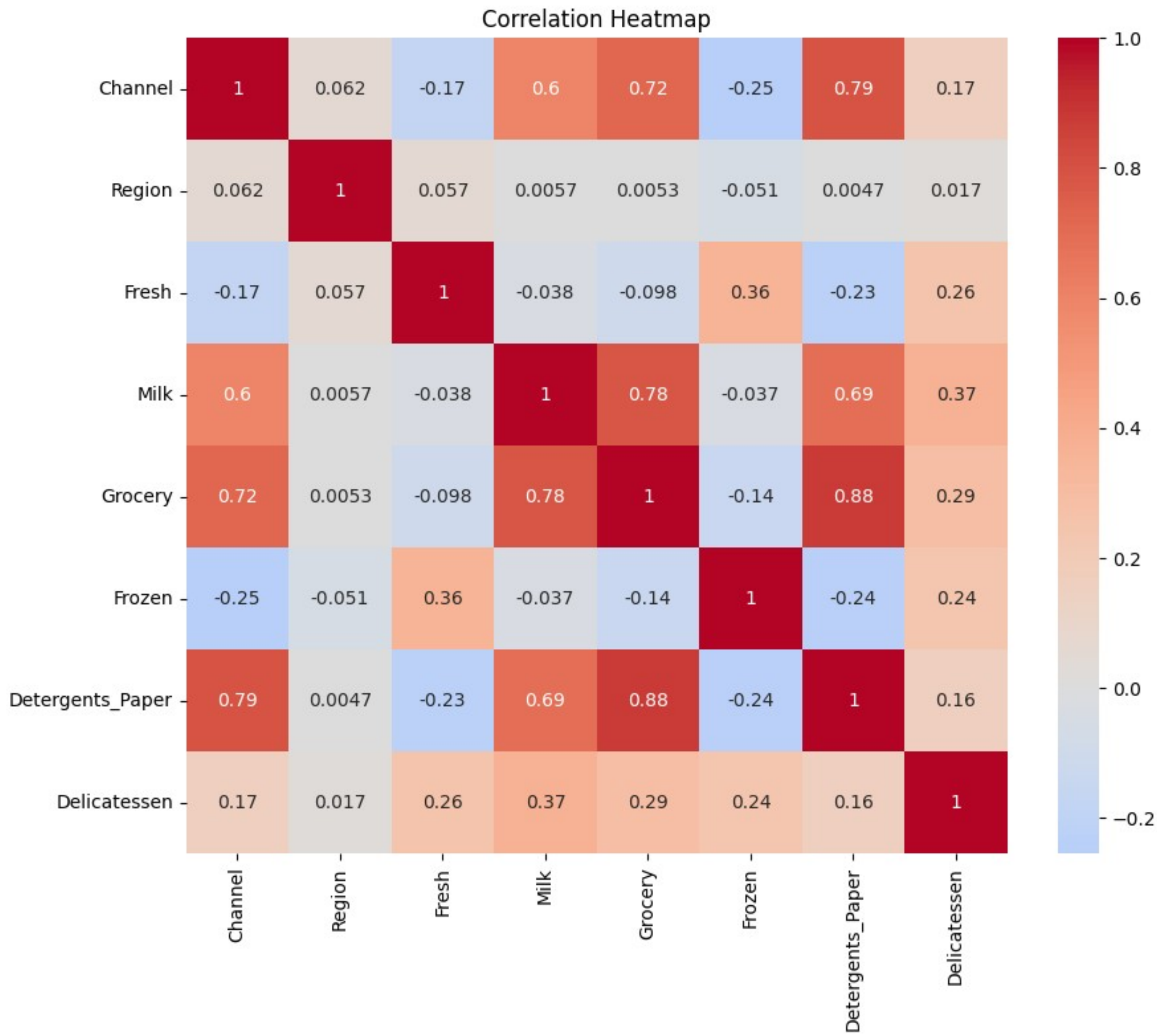


Distribution of Frozen
Count
Frozen
0
10
20
30
40
50
0
1000
2000
3000
4000
5000
6000
7000
8000



Distribution of Detergents_Paper
Count
Detergents_Paper
0
20
40
60
80
100
120
0
2000
4000
6000
8000


Distribution of Delicatessen
Count
Delicatessen
0
5
10
15
20
25
30
35
0
500
1000
1500
2000
2500
3000
3500
4000

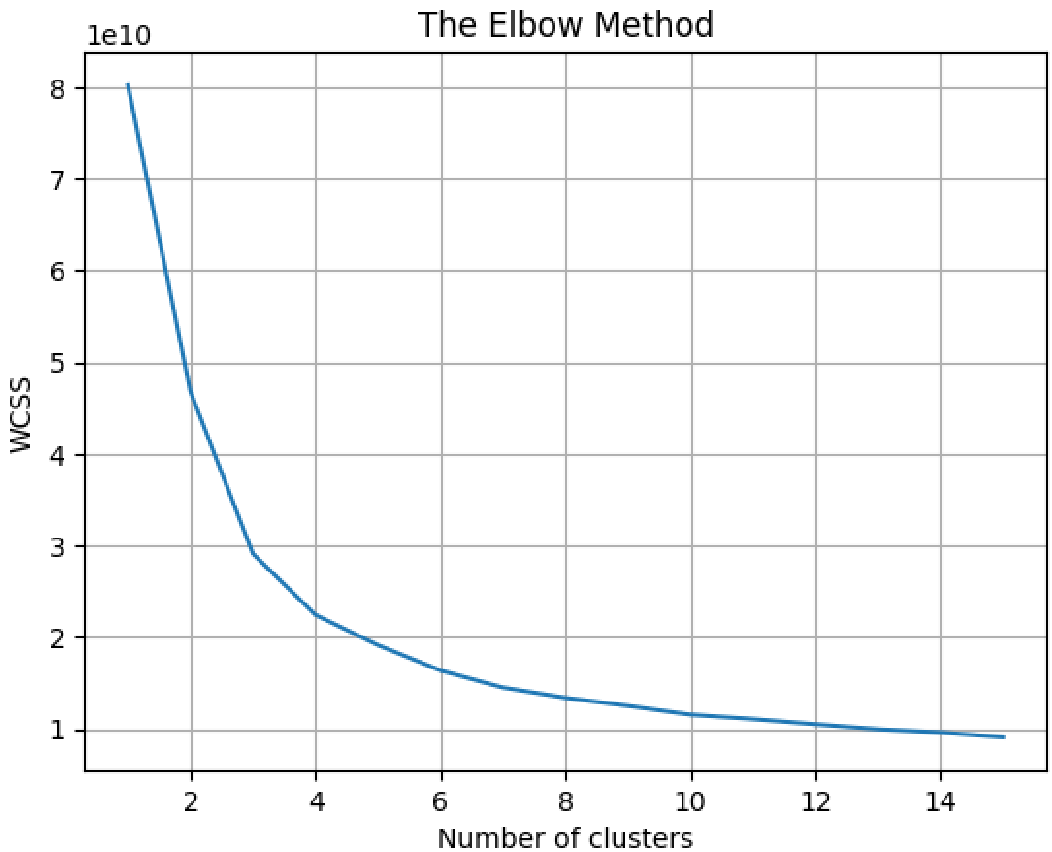



```
from sklearn.preprocessing import StandardScaler

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 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
```

```
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  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

print(df.head())
```

```
   Channel  Region    Fresh    Milk  Grocery  Frozen  Detergents_Paper  \
0        2       3  12669.0  9656.0   7561.0   214.0            2674.0

1        2       3   7057.0  9810.0   9568.0  1762.0            3293.0

2        2       3   6353.0  8808.0   7684.0  2405.0            3516.0

3        1       3  13265.0  1196.0   4221.0  6404.0             507.0
```

```
4        2       3  22615.0  5410.0   7198.0  3915.0            1777.0


   Delicatessen  Cluster
0       1338.00        0
1       1776.00        1
2       3938.25        3
3       1788.00        0
4       3938.25        0

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```

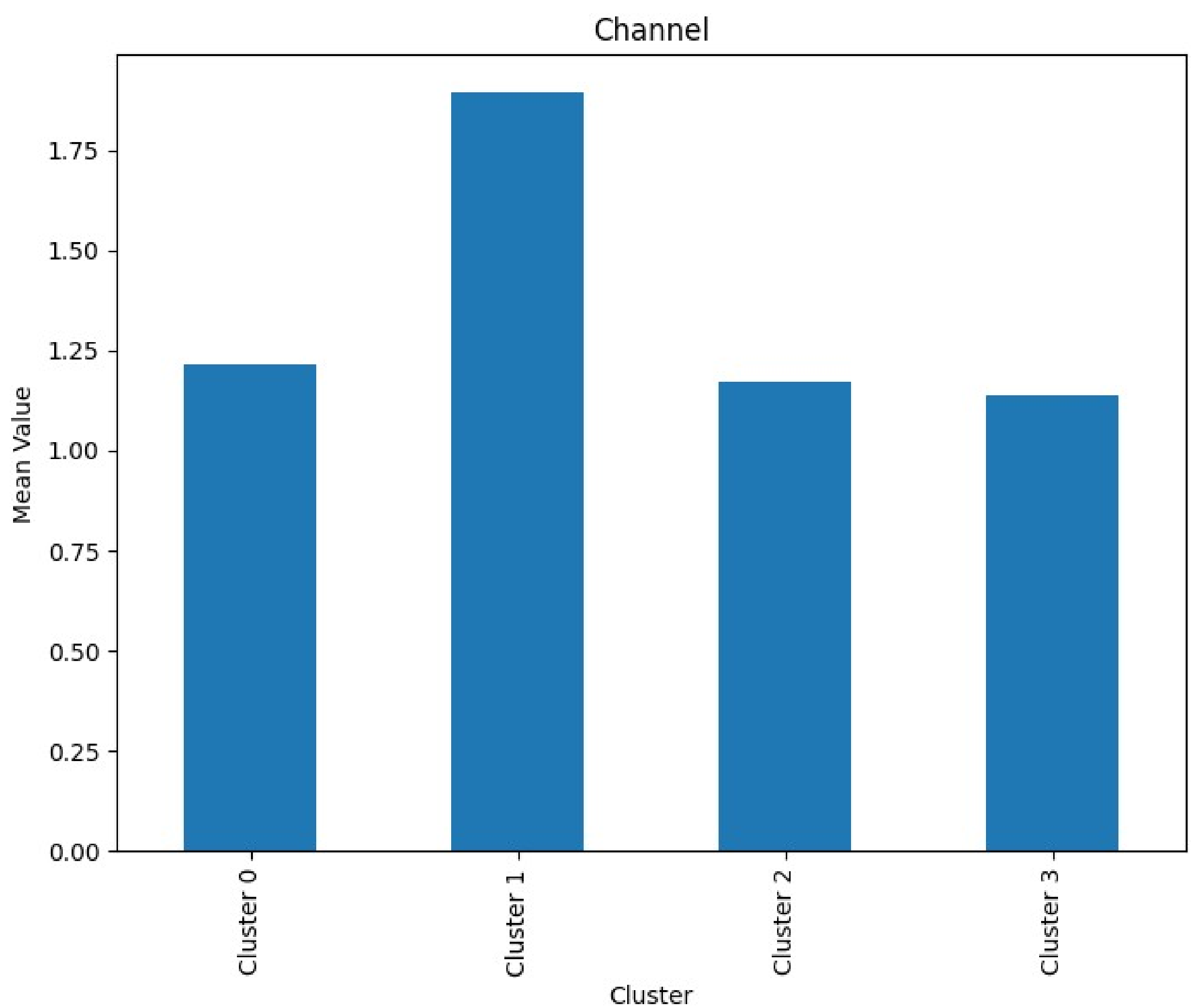
```python
# Add cluster labels to the DataFrame
df['Cluster'] = kmeans.labels_

# 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())
```

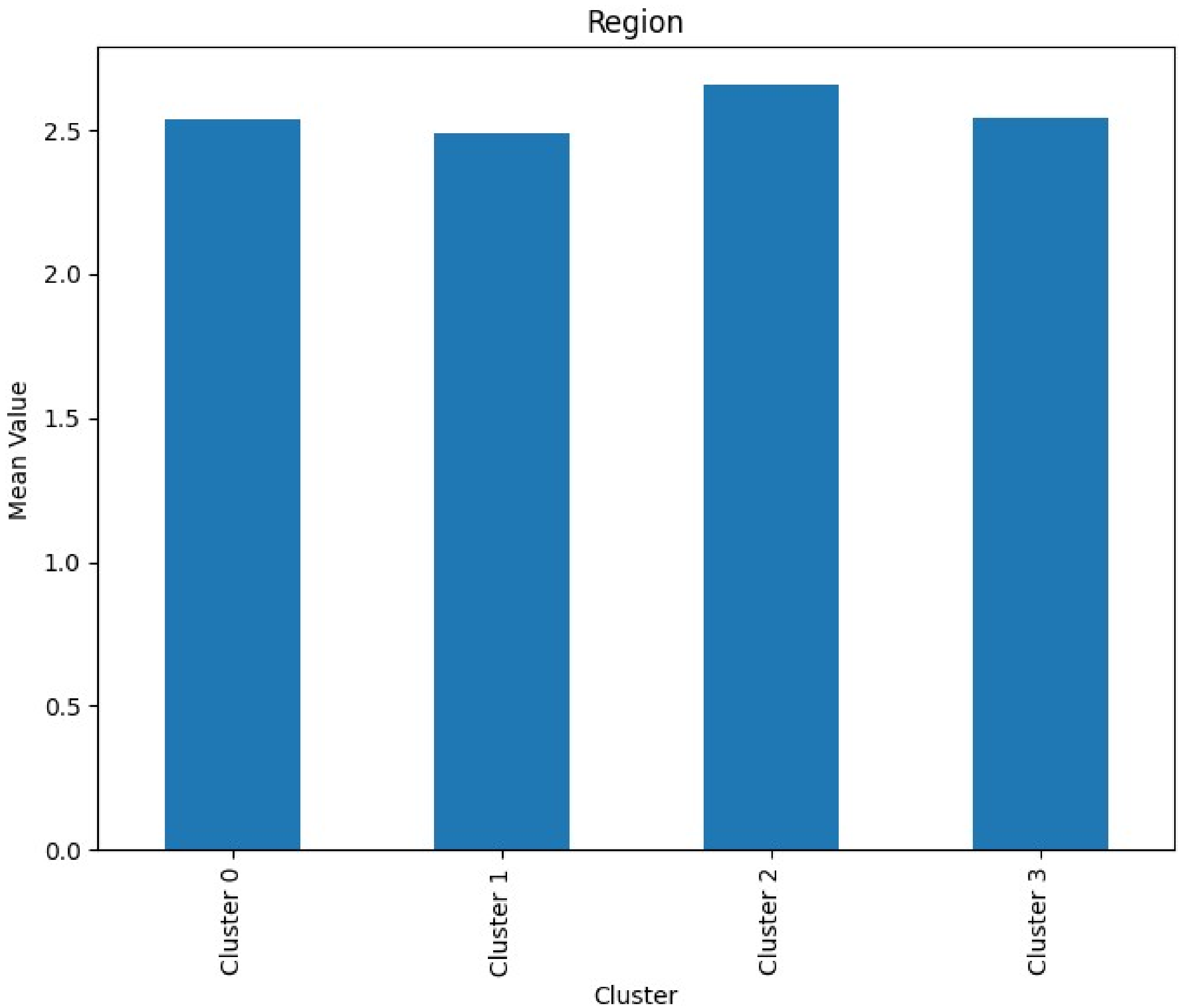
```
Cluster Sizes:
 3    176
0    112
1     94
2     58
Name: Cluster, dtype: int64

Cluster  0
          Channel      Region         Fresh          Milk
Grocery  \
count  112.000000  112.000000    112.000000    112.000000
112.000000
mean     1.214286    2.535714  16051.205357   3135.813616
4211.589286
std      0.412170    0.781873   3763.633078   2524.464860
3150.441587
min      1.000000    1.000000  10379.000000    134.000000
3.000000
25%      1.000000    2.000000  12419.750000   1283.500000
1970.500000
50%      1.000000    3.000000  16195.000000   2252.000000
3203.000000
75%      1.000000    3.000000  18830.250000   4537.000000
```

```
5700.250000
max      2.000000    3.000000  24929.000000  15676.125000
14982.000000

            Frozen  Detergents_Paper  Delicatessen  Cluster
count   112.000000        112.000000    112.000000    112.0
mean   2988.859375        994.785714   1229.573661      0.0
std    2531.352938       1245.589613    963.527882      0.0
min     118.000000          3.000000     51.000000      0.0
25%    1018.750000        188.500000    514.250000      0.0
50%    2157.500000        456.500000    879.000000      0.0
75%    4276.000000       1404.000000   1804.500000      0.0
max    7772.250000       6707.000000   3938.250000      0.0

Cluster  1
         Channel     Region         Fresh          Milk       Grocery
\
count  94.000000  94.000000     94.000000     94.000000     94.000000

mean    1.893617   2.489362   5331.893617  10454.450798  17196.140957

std     0.309980   0.799794   5111.448153   3937.245330   4905.345002

min     1.000000   1.000000     18.000000   1266.000000   8852.000000

25%     2.000000   2.000000   1409.500000   7576.000000  12563.250000

50%     2.000000   3.000000   4047.000000  10601.000000  16596.000000

75%     2.000000   3.000000   7870.500000  14316.500000  22288.500000

max     2.000000   3.000000  22925.000000  15676.125000  23409.875000


            Frozen  Detergents_Paper  Delicatessen  Cluster
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
```

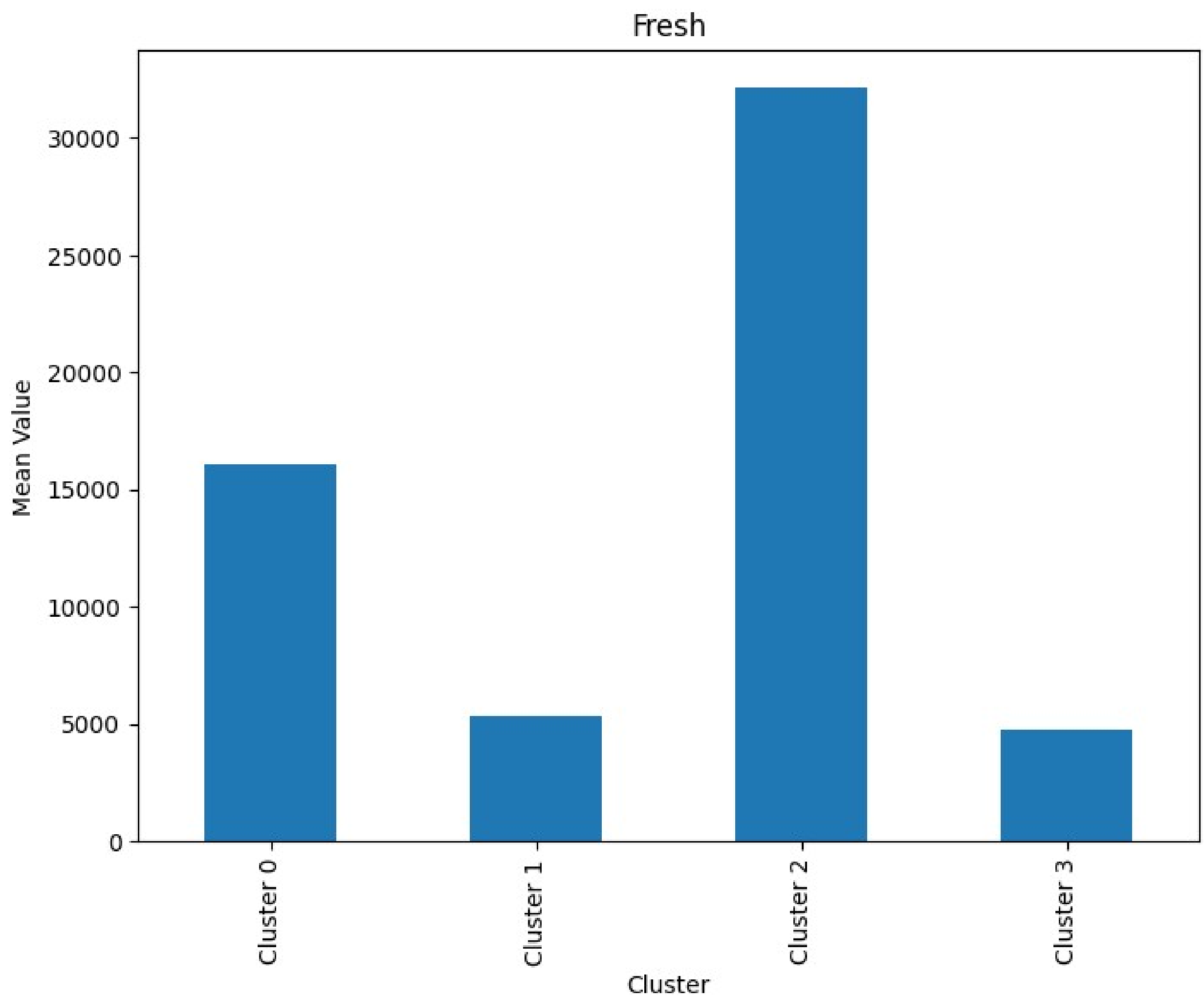
```
std      0.381039   0.714554   5122.024937   4808.223223   5915.174661

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

# Calculate the mean values for each feature per cluster
cluster_means = df.groupby('Cluster').mean()

# Transpose the DataFrame so that the features are the rows (this will
make plotting easier)
cluster_means = cluster_means.transpose()

# Create bar plot for each feature
for feature in cluster_means.index:
    cluster_means.loc[feature].plot(kind='bar', figsize=(8,6))
    plt.title(feature)
    plt.ylabel('Mean Value')
    plt.xticks(ticks=range(4), labels=['Cluster 0', 'Cluster 1',
'Cluster 2', 'Cluster 3'])
    plt.show()
```

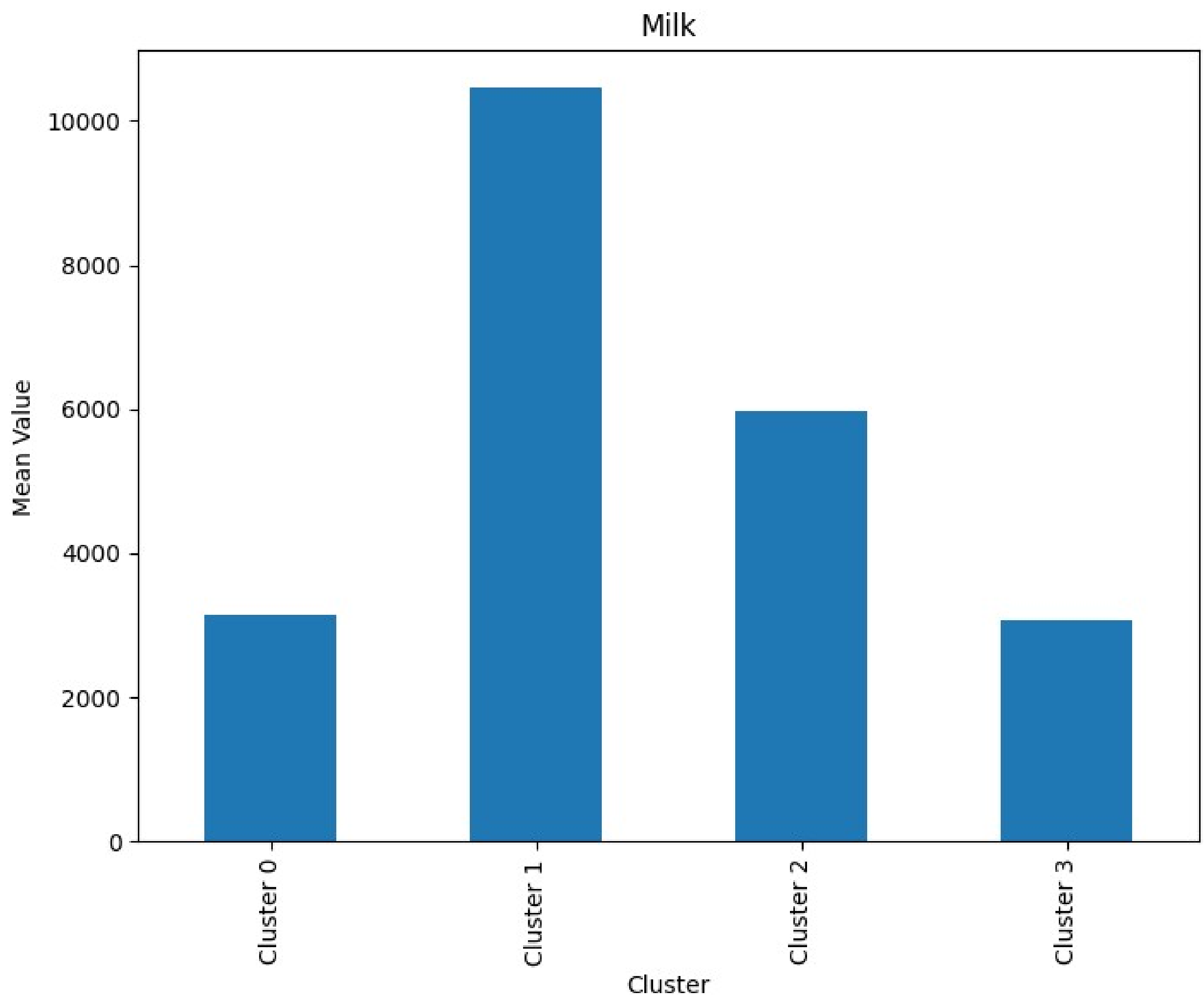


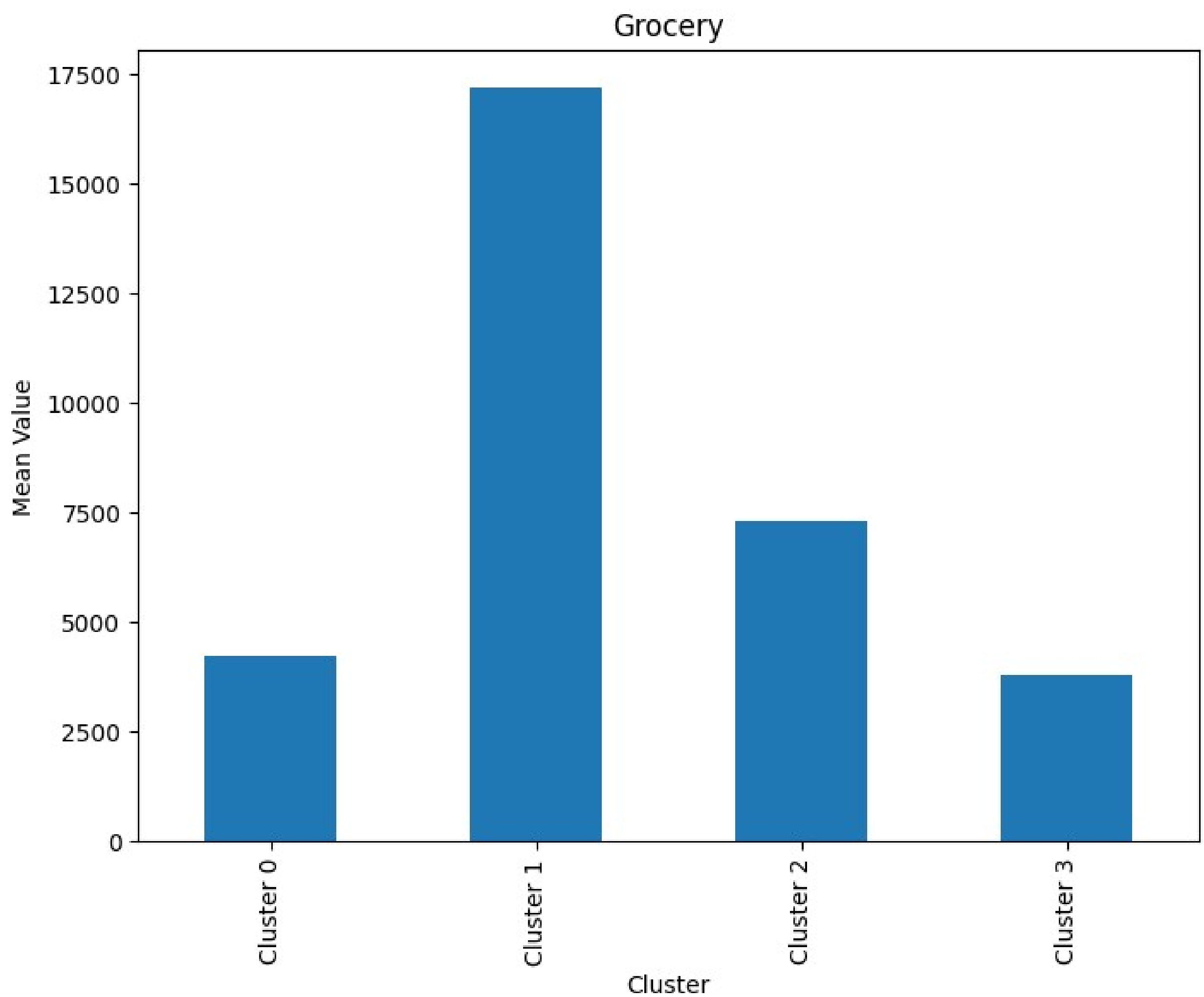


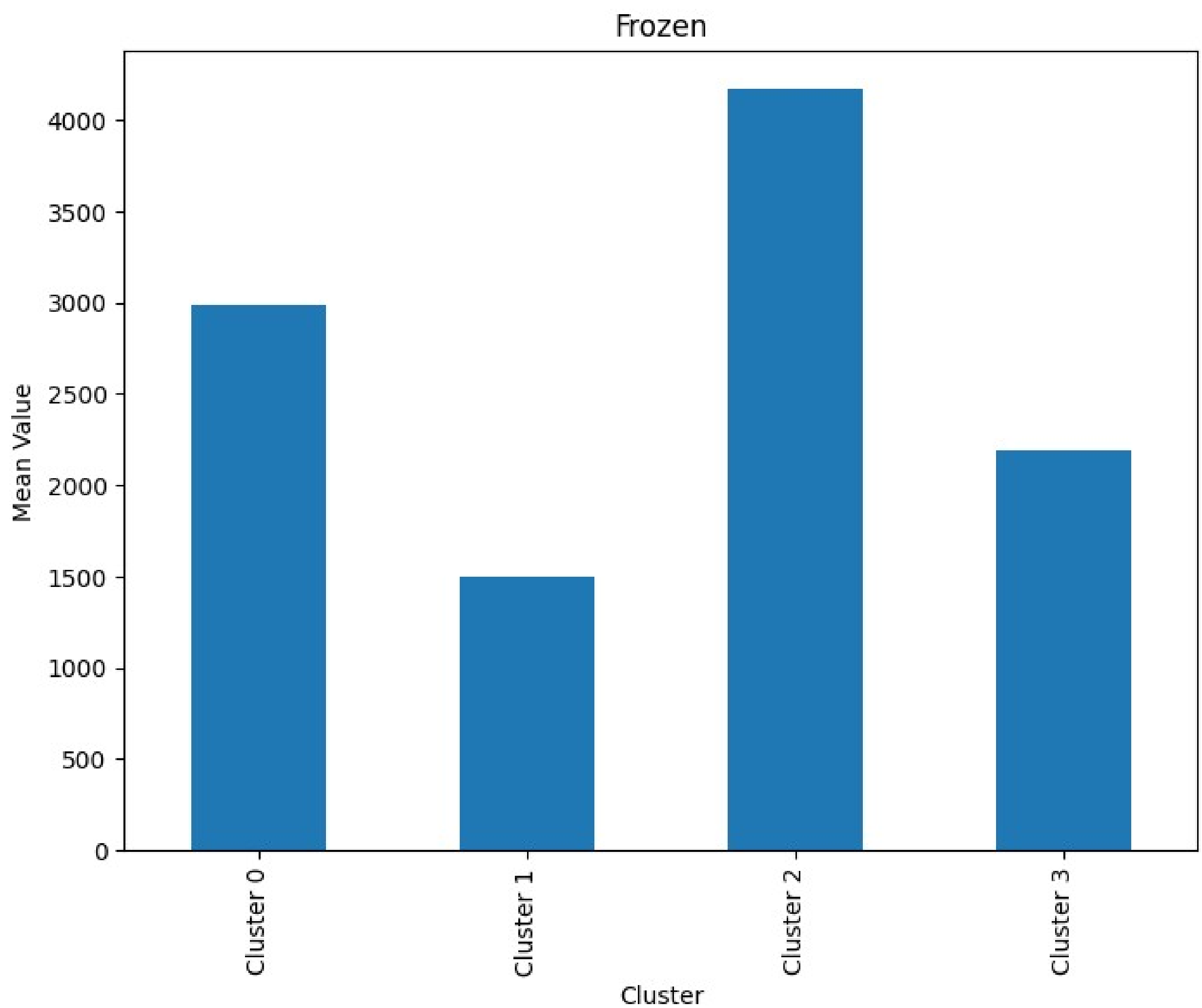
Region
2.5
2.0
1.5
1.0
0.5
0.0
Mean Value
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster



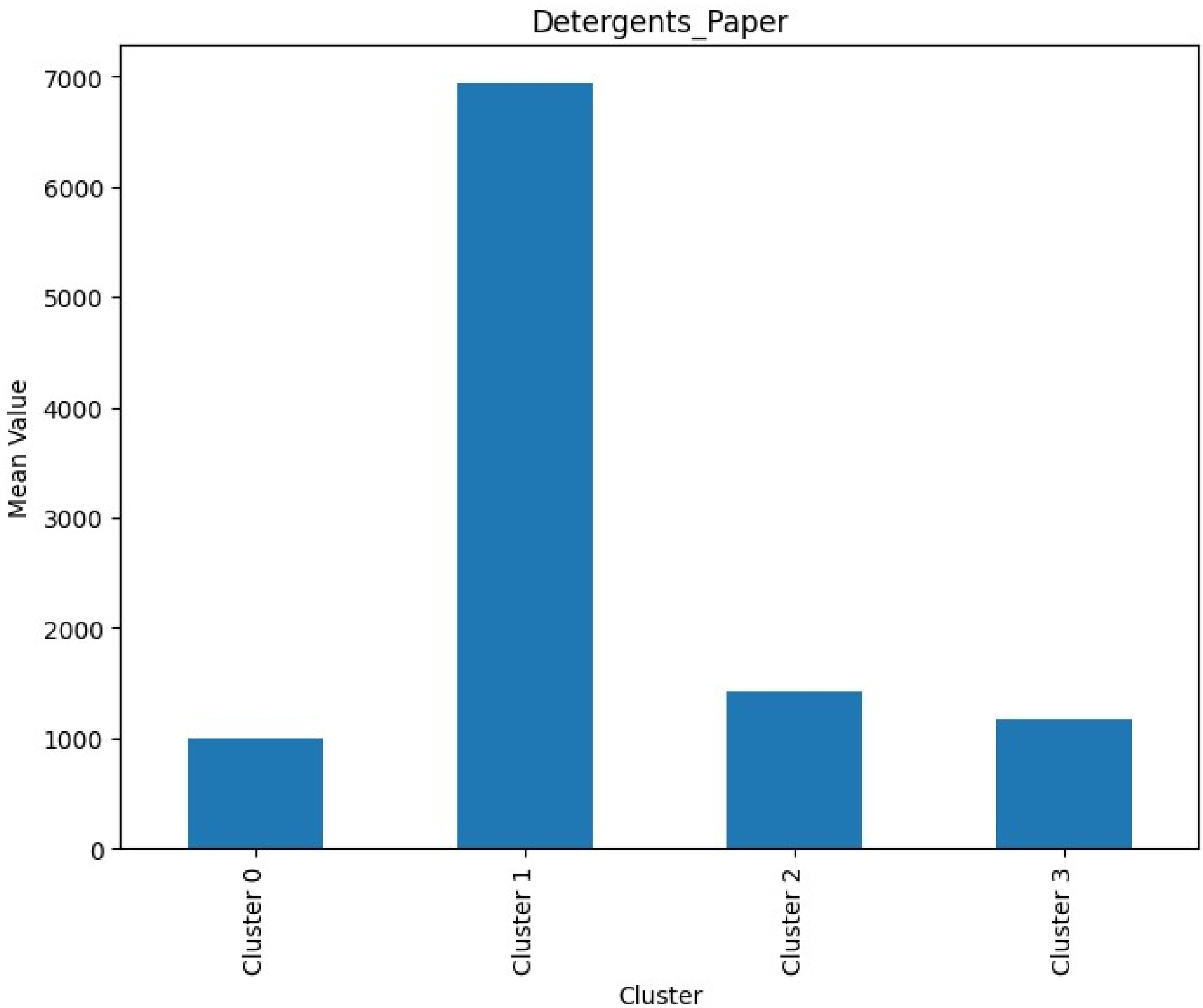
Fresh
Mean Value
30000
25000
20000
15000
10000
5000
0
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster

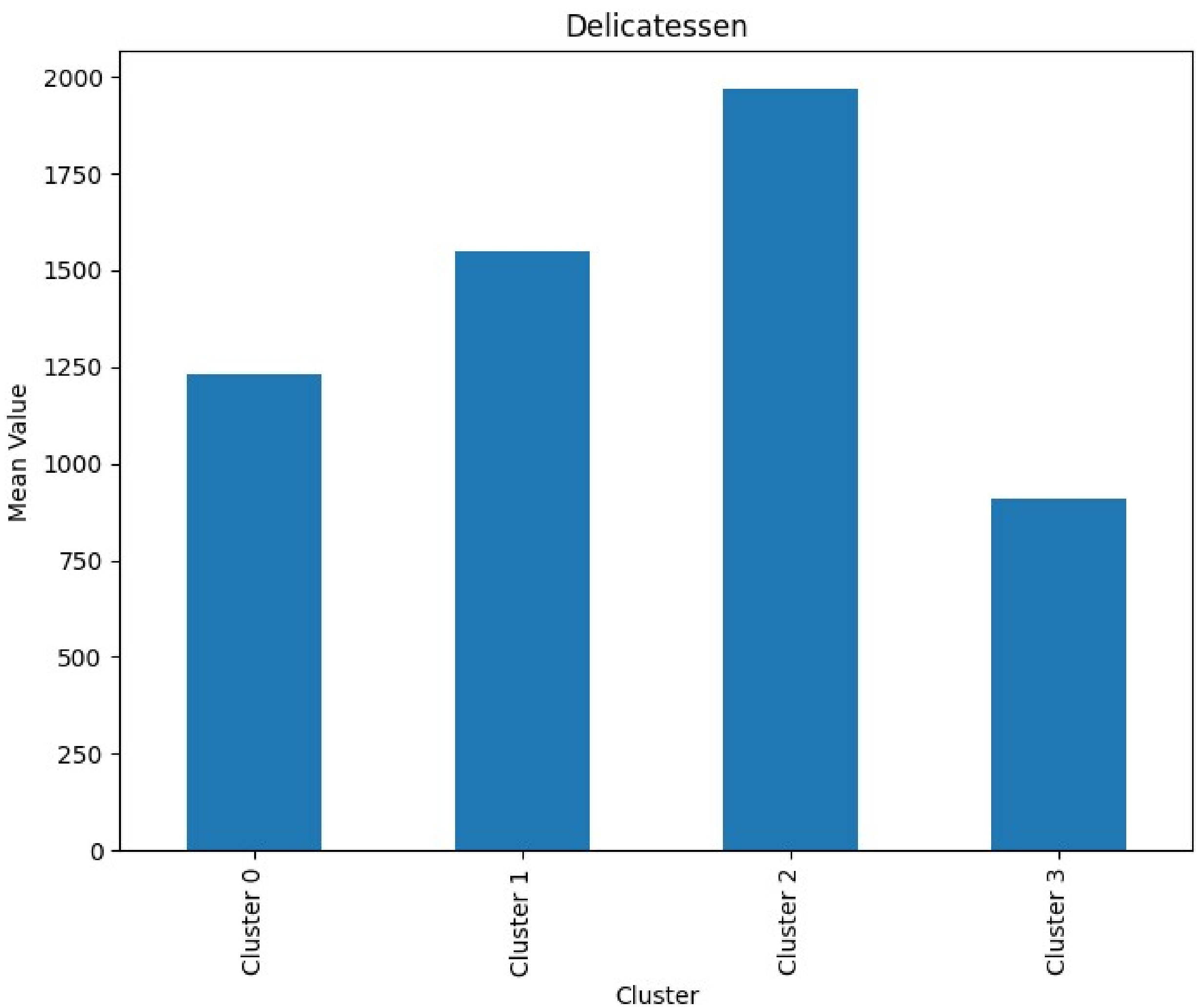

Milk
Mean Value
10000
8000
6000
4000
2000
0
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster


Grocery
Mean Value
17500
15000
12500
10000
7500
5000
2500
0
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster



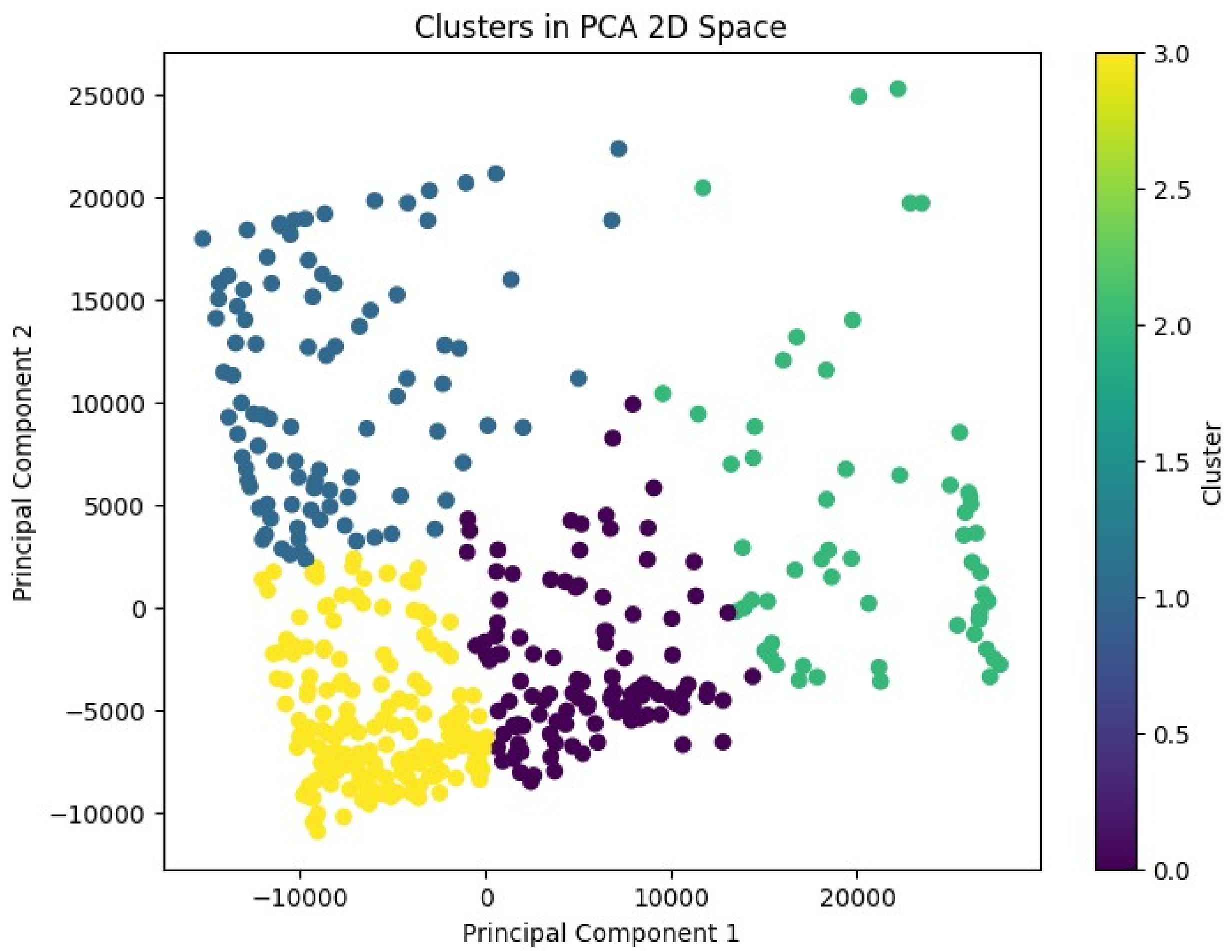
Frozen
Mean Value
0
500
1000
1500
2000
2500
3000
3500
4000
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster


Detergents_Paper
7000
6000
5000
4000
3000
2000
1000
0
Mean Value
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster




```python
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()
```



Conclusion:

1.Clustering the wholesale customers into different groups allows us to understand the underlying patterns and behaviors within the data. This can be useful for various purposes:

Customer Segmentation: By clustering customers based on their spending patterns, we can create customer segments. For example, we can identify high-value customers who spend more on certain product categories, regular customers who have balanced spending across categories, and low-value customers who spend less overall.

Marketing Strategies: Each customer segment may require a different marketing approach. For instance, high-value customers might respond well to loyalty programs, while regular customers may benefit from personalized recommendations.

Inventory Management: Retailers can optimize their inventory based on the preferences of different customer segments. If a cluster of customers prefers fresh products, the retailer can ensure sufficient stock of fresh items for that segment.

2.The impact of a specific delivery scheme can vary among customer segments:

High-Value Customers: High-value customers who spend more on certain categories may prioritize faster and more reliable deliveries. They might be willing to pay extra for premium delivery services to ensure product freshness and availability

Regular Customers: Regular customers with balanced spending patterns may appreciate flexible delivery options. They may not require as frequent deliveries as high-value customers but still prefer on-time deliveries to maintain their inventory.

Low-Value Customers: Low-value customers who spend less overall may prioritize cost-effectiveness. They might be more inclined to accept longer delivery times or consolidated deliveries to reduce shipping costs.

Segment-Specific Preferences: Some customer segments may have specific preferences, such as hotels requiring more frequent deliveries of fresh products. Tailoring delivery schemes to these preferences can lead to higher customer satisfaction.

Feedback and Adaptation: Regularly collecting feedback from different customer segments is essential. Adjusting the delivery scheme based on customer feedback can help meet the unique needs of each segment and improve customer loyalty.