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.

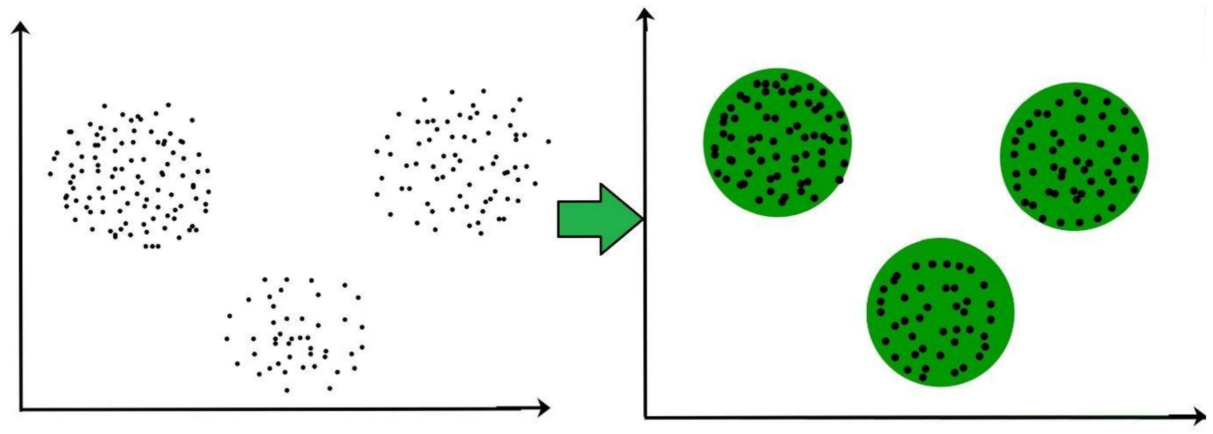
Objective: Able to perform various feature engineering tasks, apply Clustering Algorithm on the given dataset.

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)

12-aditi-sawant-ml-exp5

September 5, 2023

[ ]:
```
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('/kaggle/input'):
    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
```

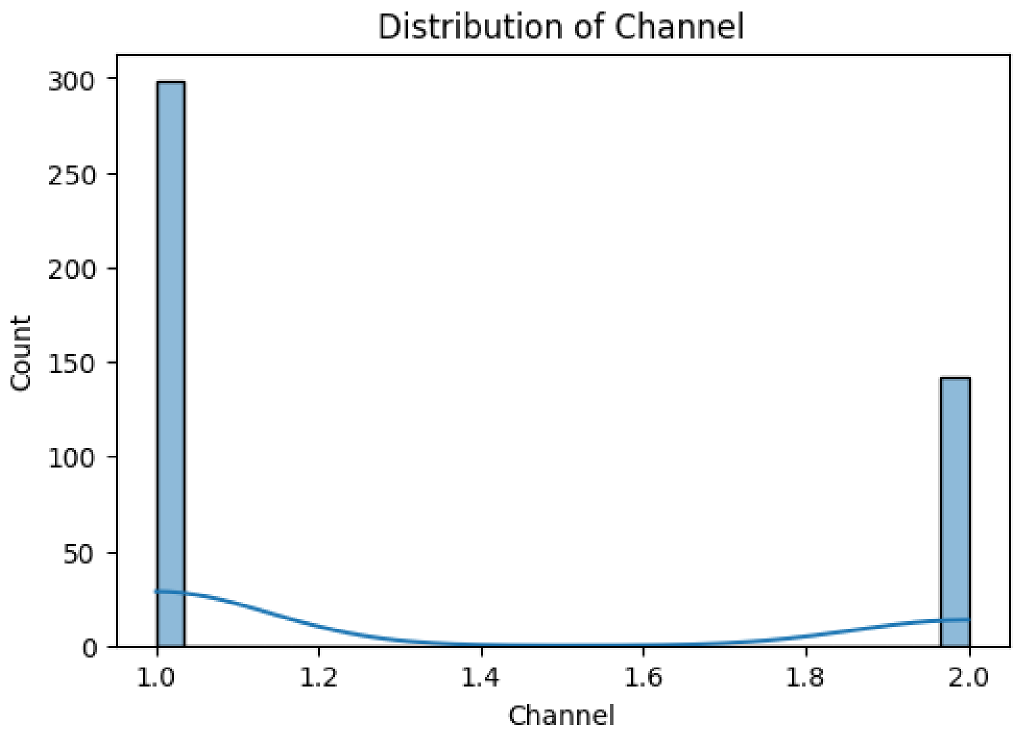
```
[ ]: # Check for missing values
     print("Missing values per column:")
     print(df.isnull().sum())
```

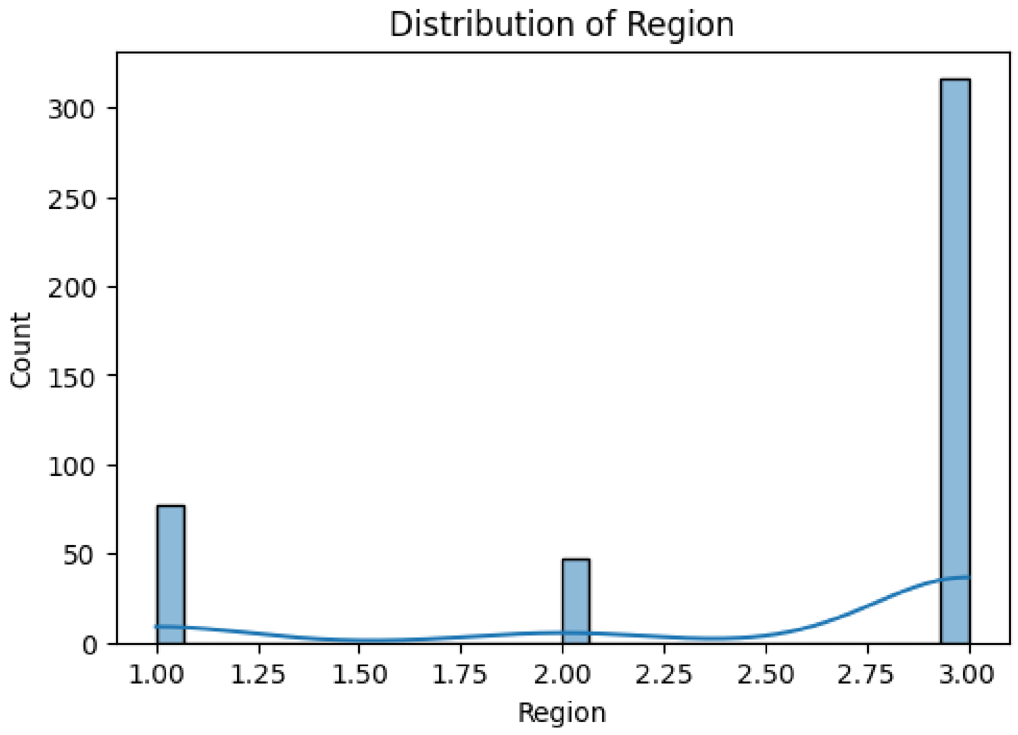
```
Missing values per column:
Channel             0
Region              0
Fresh               0
Milk                0
Grocery             0
Frozen              0
Detergents_Paper    0
Delicassen          0
dtype: int64
```

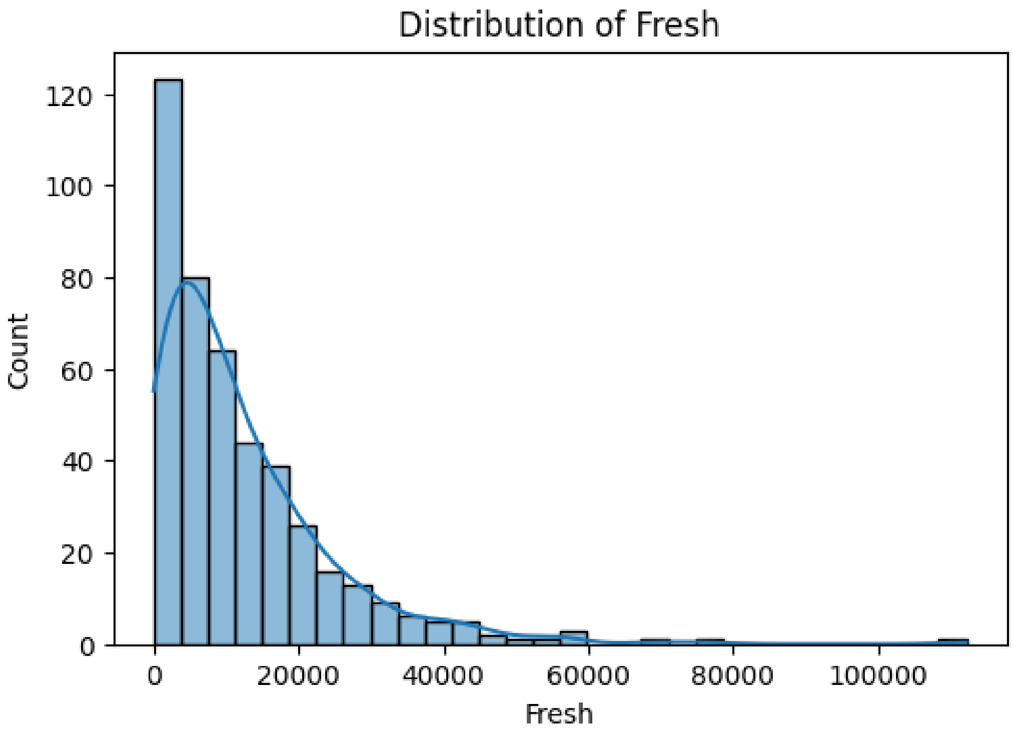
```
[ ]: import matplotlib.pyplot as plt
     import seaborn as sns

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

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

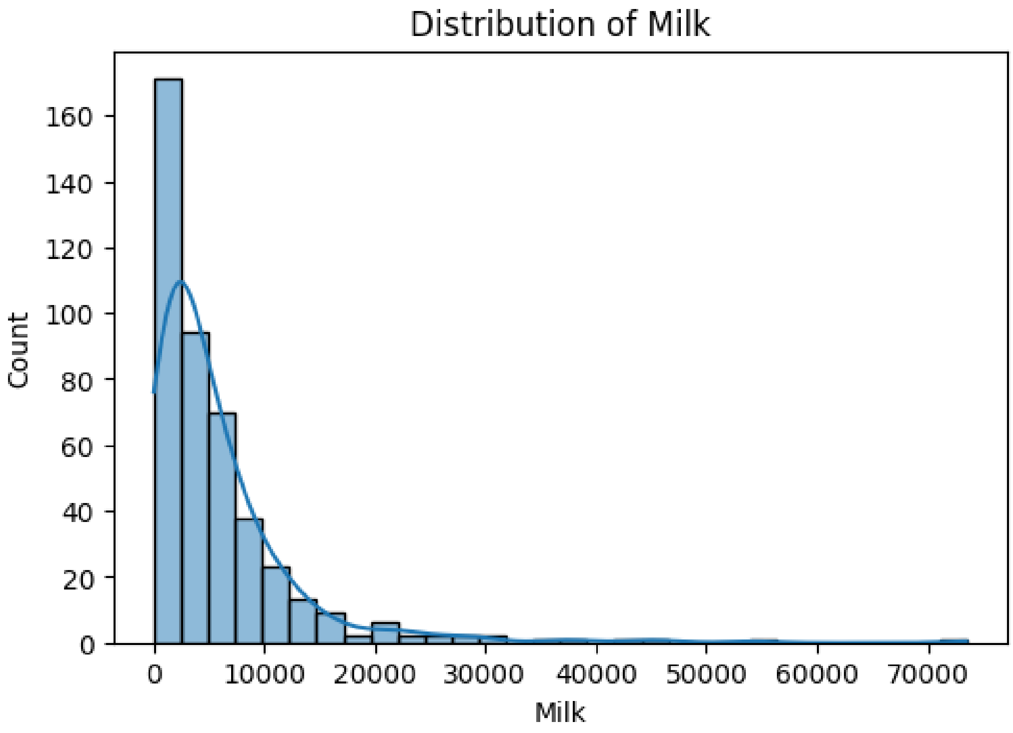
```
# Distribution plots for each feature
for column in df.columns:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[column], bins=30, kde=True)
    plt.title(f'Distribution of {column}')
    plt.show()

# 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()
```

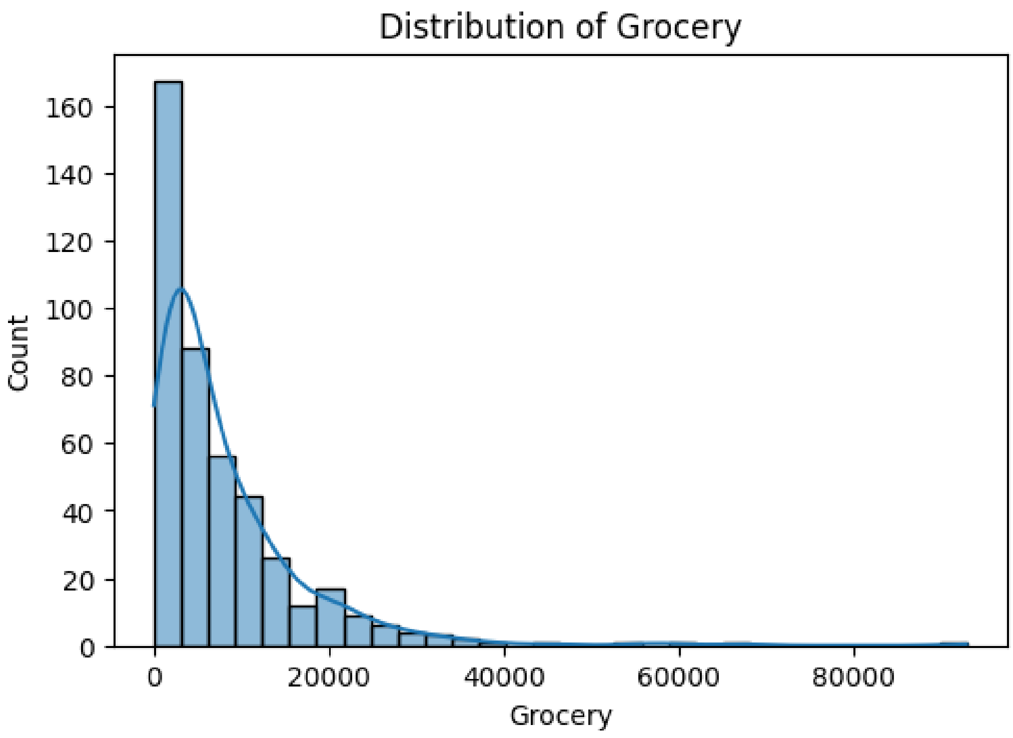
```
Descriptive Statistics:
          Channel      Region          Fresh          Milk       Grocery  \
count  440.000000  440.000000     440.000000    440.000000    440.000000
mean     1.322727    2.543182   12000.297727   5796.265909   7951.277273
std      0.468052    0.774272   12647.328865   7380.377175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

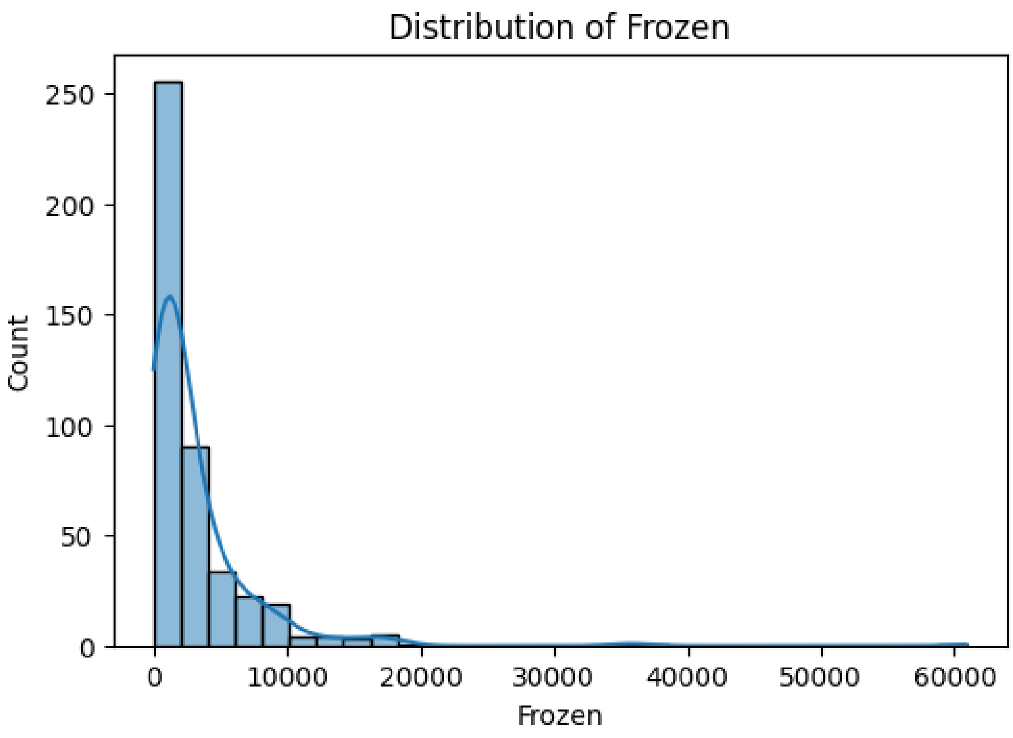

Distribution of Channel
Count
Channel
300
250
200
150
100
50
0
1.0
1.2
1.4
1.6
1.8
2.0

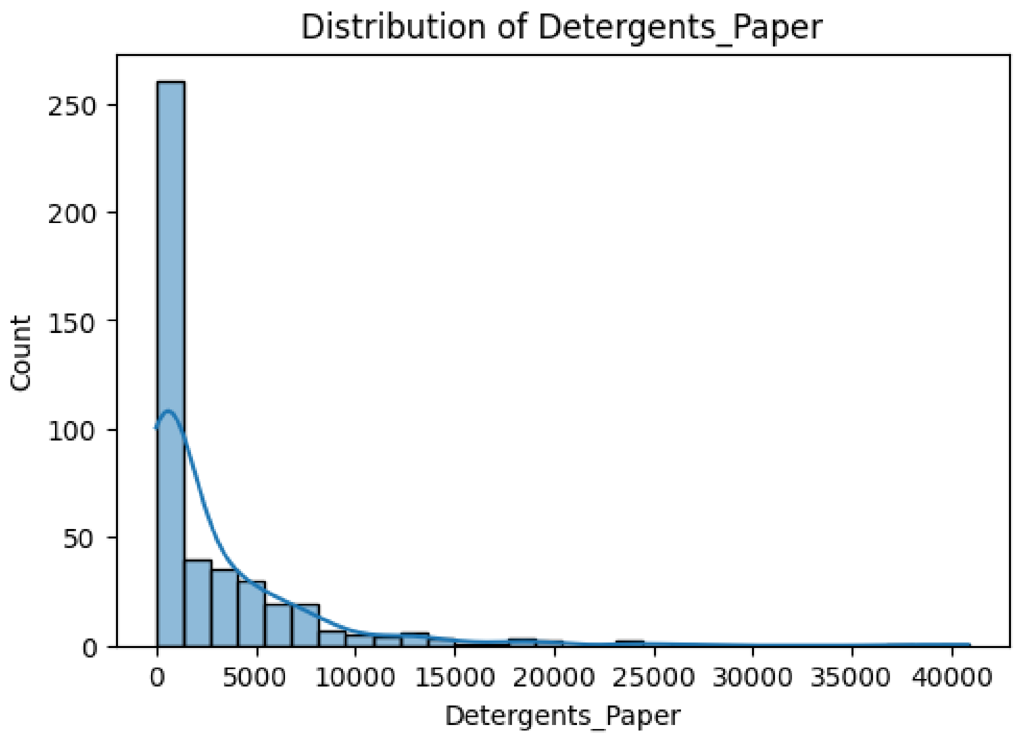

Distribution of Region
Count
Region
300
250
200
150
100
50
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

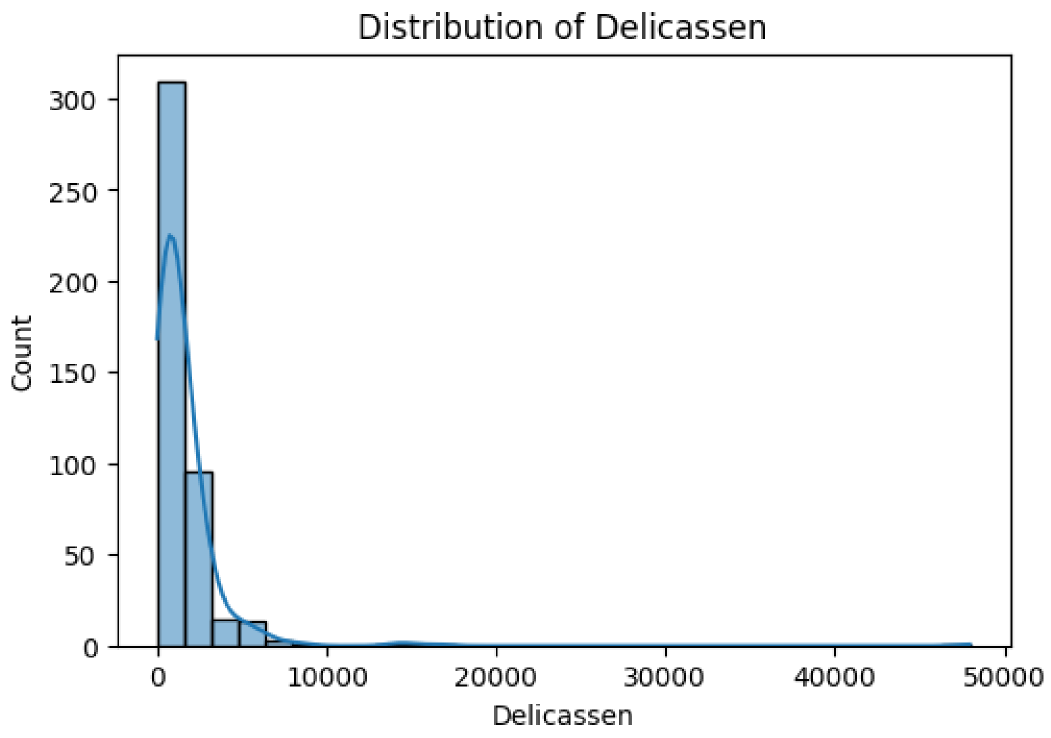

Distribution of Fresh
Count
Fresh
0
20
40
60
80
100
120
0
20000
40000
60000
80000
100000

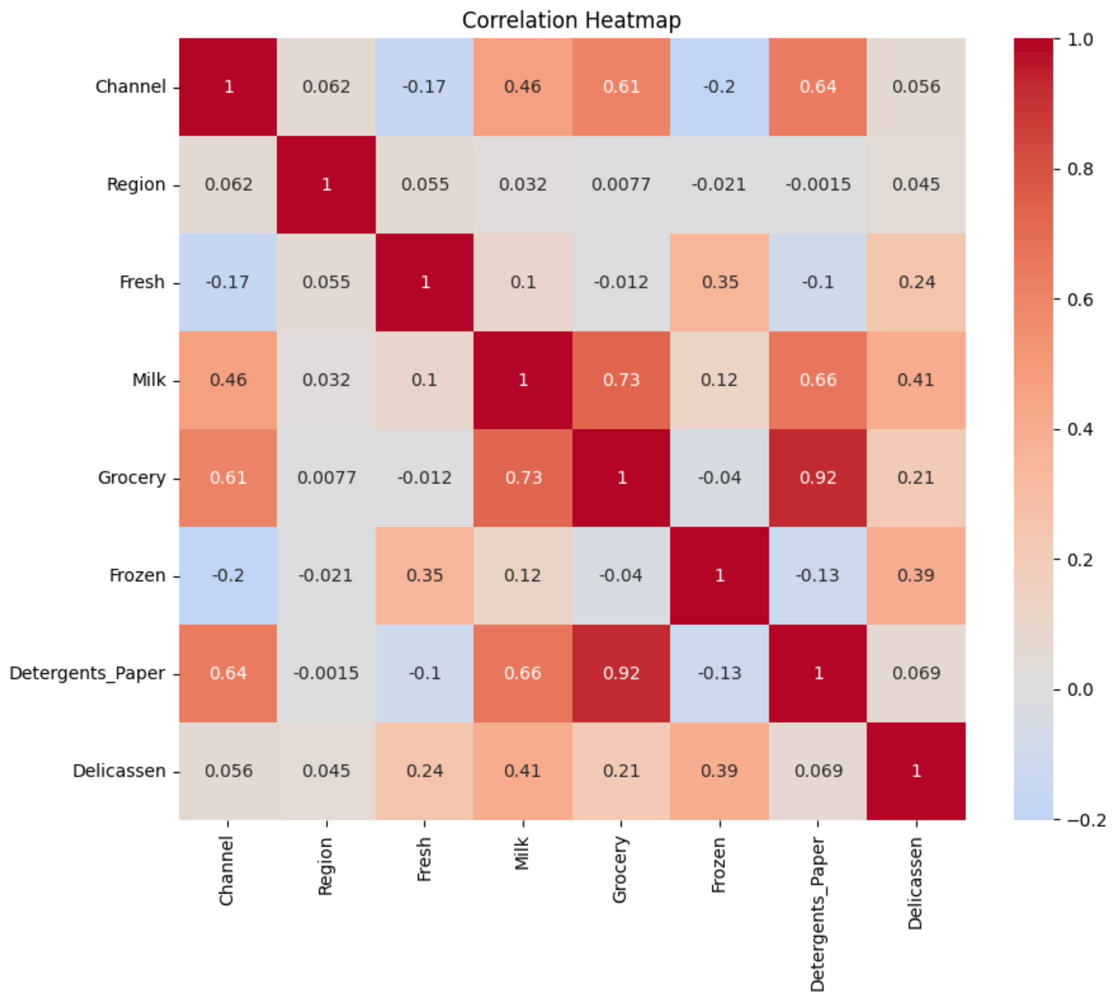

Distribution of Milk
Count
Milk
0
20
40
60
80
100
120
140
160
0
10000
20000
30000
40000
50000
60000
70000


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


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

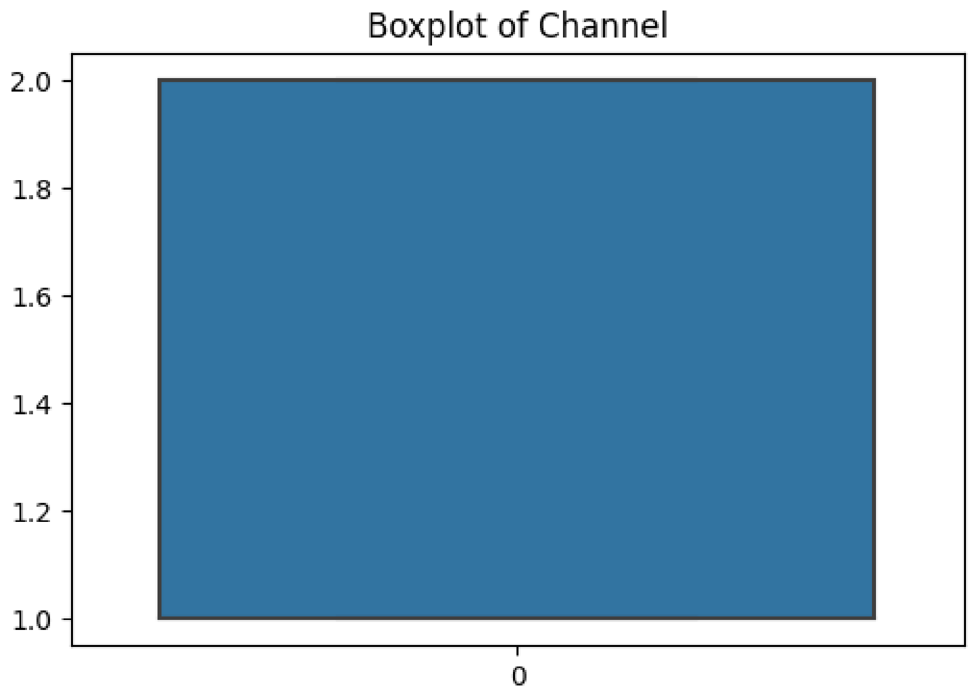

Distribution of Detergents_Paper
Count
250
200
150
100
50
0
0
5000
10000
15000
20000
25000
30000
35000
40000
Detergents_Paper

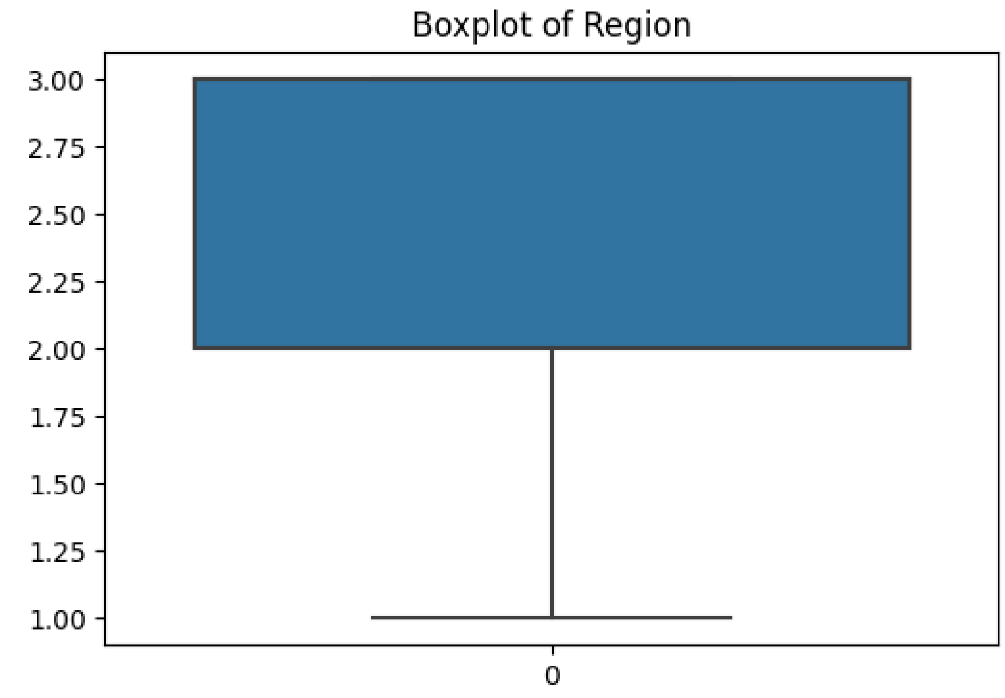

Distribution of Delicassen
Count
300
250
200
150
100
50
0
0
10000
20000
30000
40000
50000
Delicassen

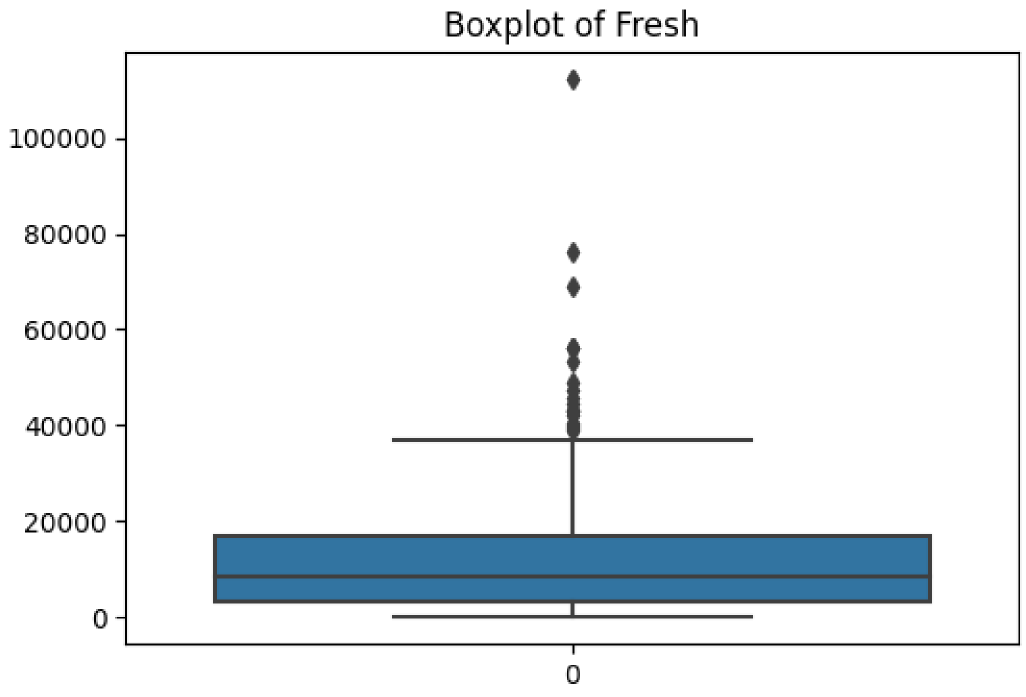


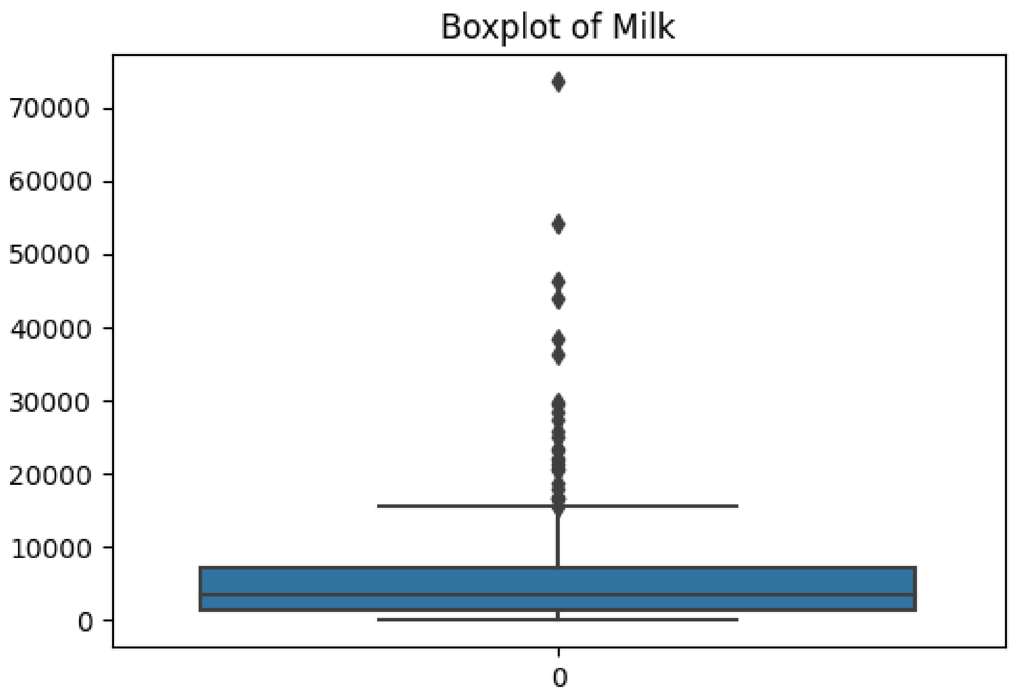
	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Channel	1	0.062	-0.17	0.46	0.61	-0.2	0.64	0.056
Region	0.062	1	0.055	0.032	0.0077	-0.021	-0.0015	0.045
Fresh	-0.17	0.055	1	0.1	-0.012	0.35	-0.1	0.24
Milk	0.46	0.032	0.1	1	0.73	0.12	0.66	0.41
Grocery	0.61	0.0077	-0.012	0.73	1	-0.04	0.92	0.21
Frozen	-0.2	-0.021	0.35	0.12	-0.04	1	-0.13	0.39
Detergents_Paper	0.64	-0.0015	-0.1	0.66	0.92	-0.13	1	0.069
Delicassen	0.056	0.045	0.24	0.41	0.21	0.39	0.069	1

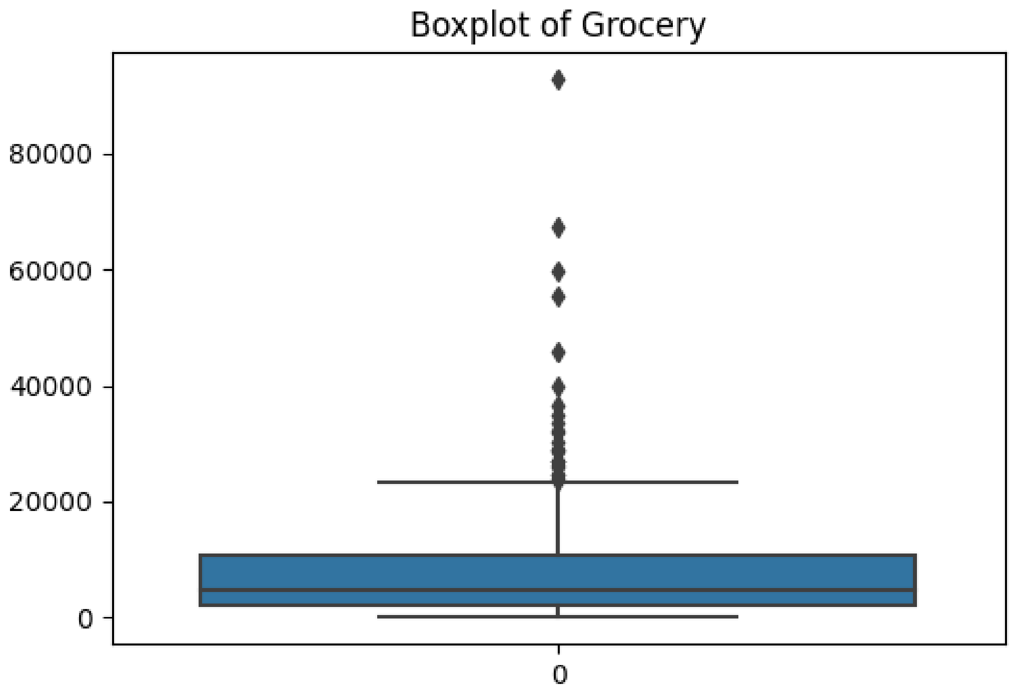
[ ]:
```
# checking for outliers
import seaborn as sns
import matplotlib.pyplot as plt

# 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)}')
```

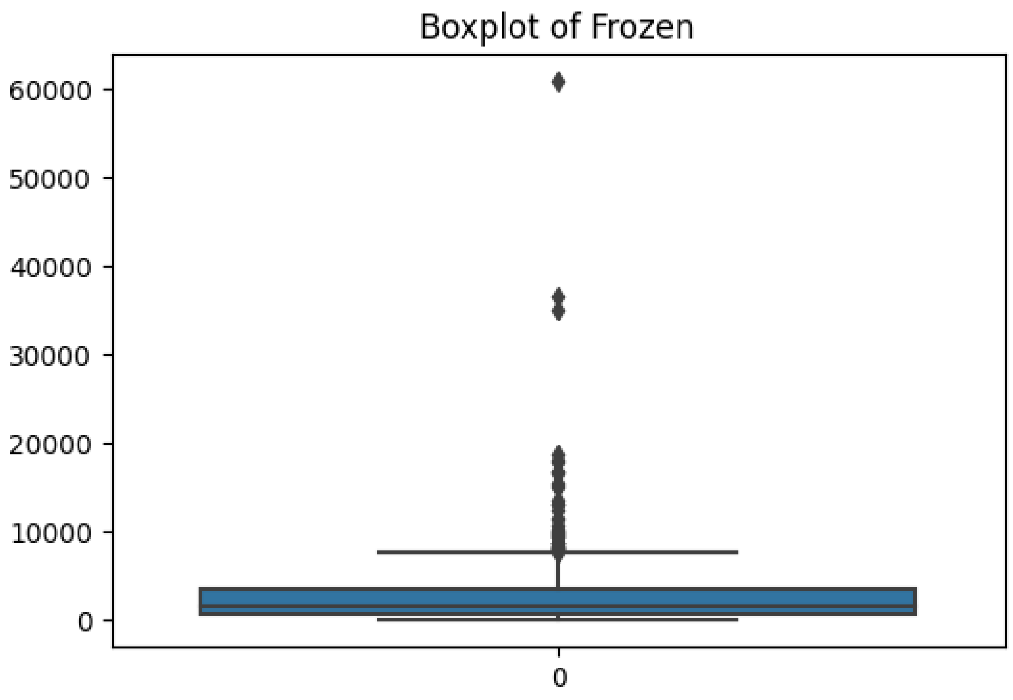


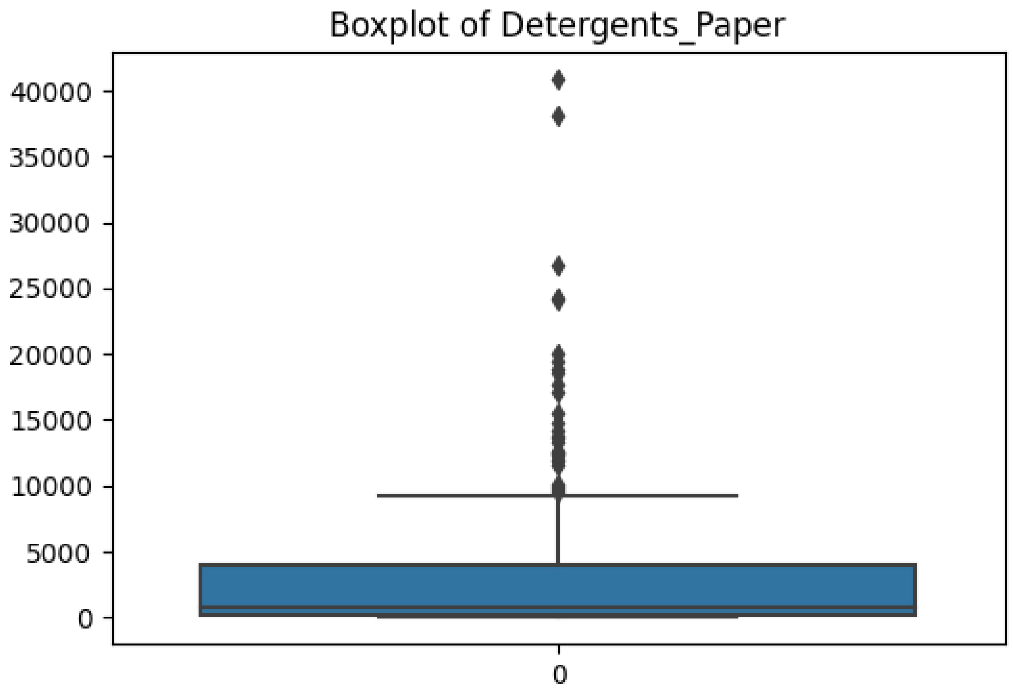

Boxplot of Region
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0

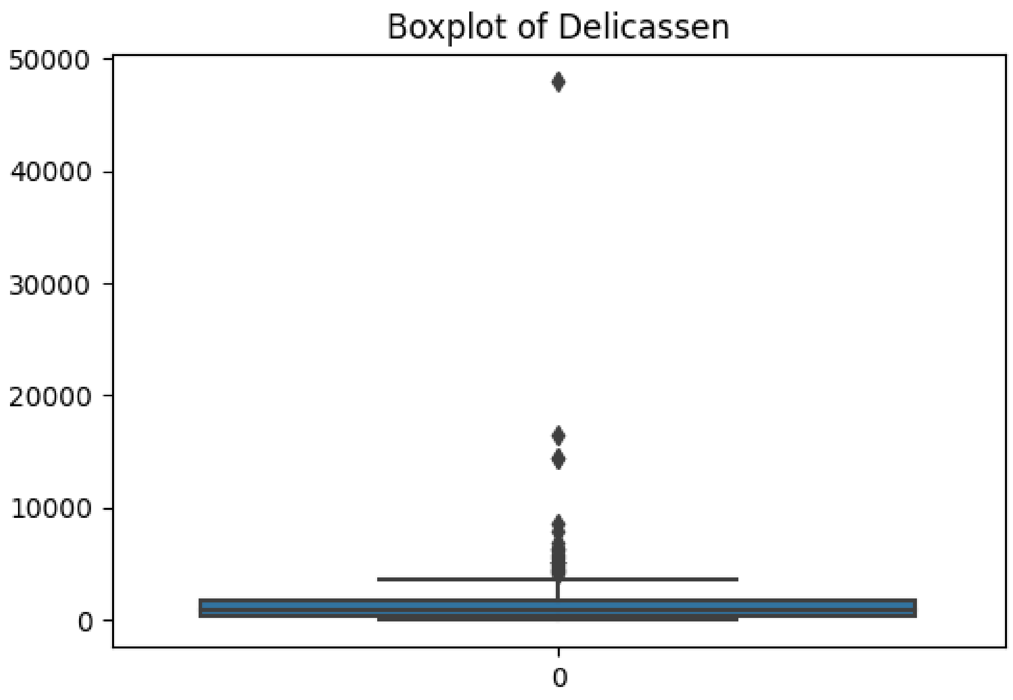

Boxplot of Fresh
100000
80000
60000
40000
20000
0
0


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

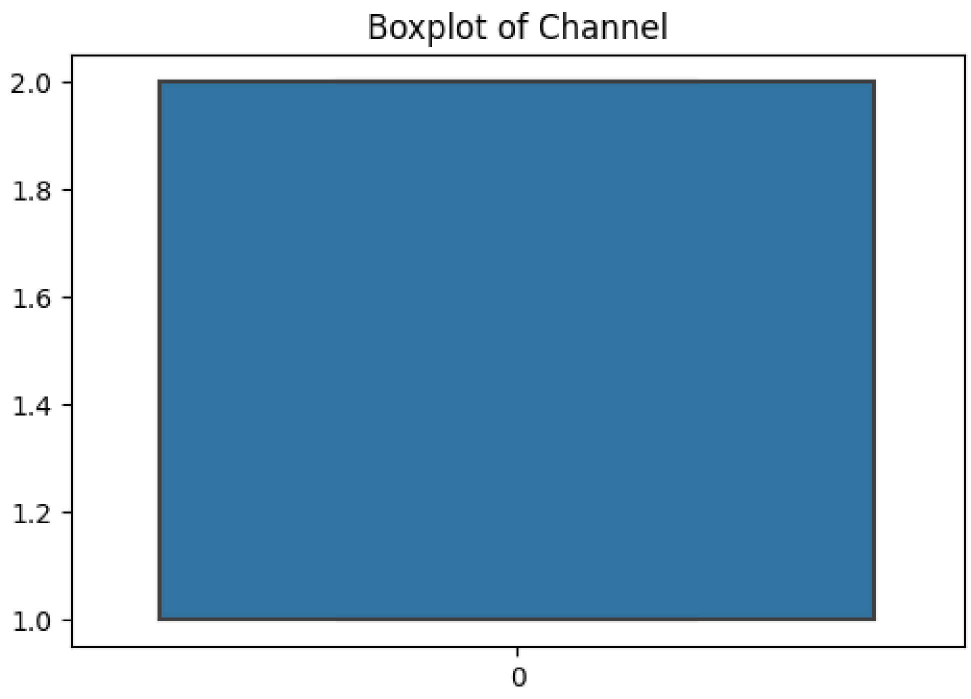


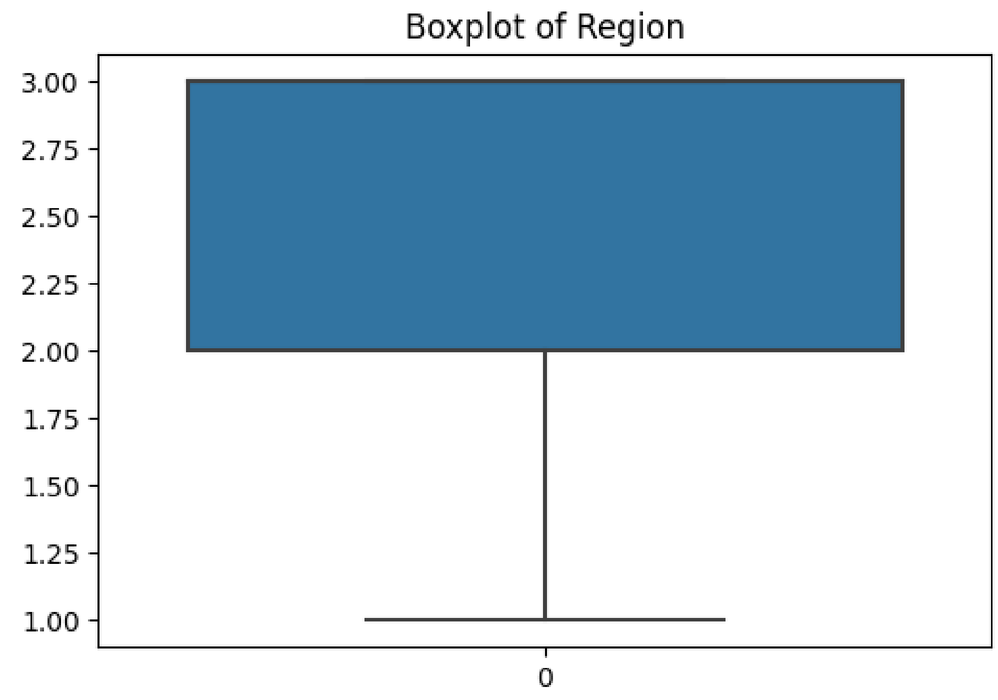
```
Number of outliers in Channel: 0
Number of outliers in Region: 0
Number of outliers in Fresh: 20
Number of outliers in Milk: 28
Number of outliers in Grocery: 24
Number of outliers in Frozen: 43
Number of outliers in Detergents_Paper: 30
Number of outliers in Delicassen: 27
```

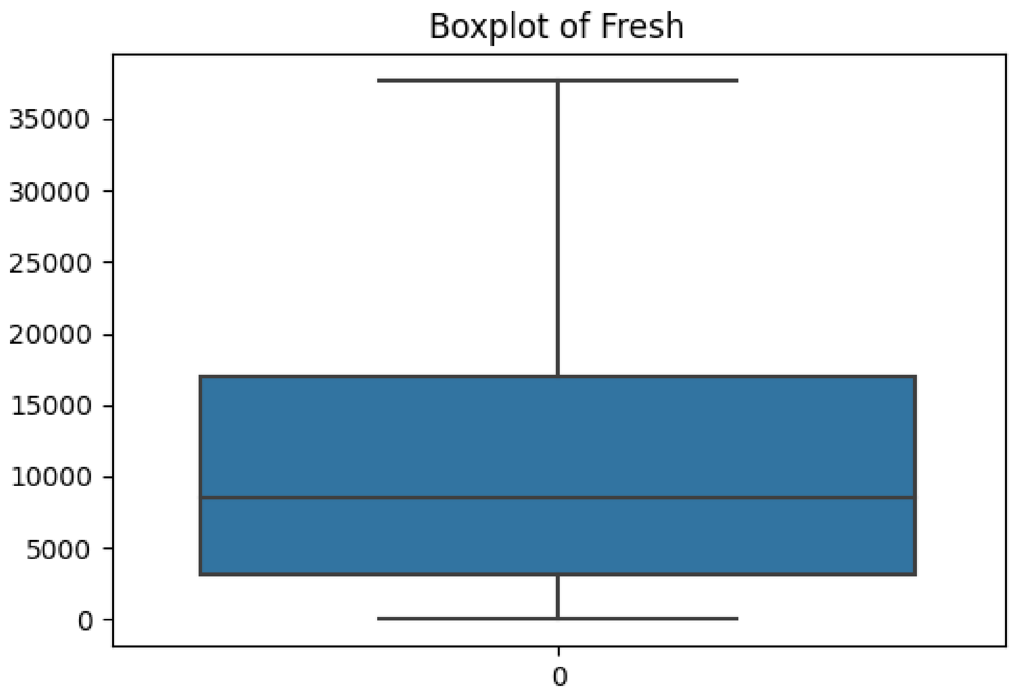
[ ]:
```
def handle_outliers(dataframe, column):
    Q1 = dataframe[column].quantile(0.25)
    Q3 = dataframe[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_limit = Q1 - 1.5*IQR
    upper_limit = Q3 + 1.5*IQR
    dataframe[column] = dataframe[column].apply(lambda x: upper_limit if x > 
 ↪upper_limit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

# Handle outliers for each feature
for column in df.columns:
    handle_outliers(df, column)
```

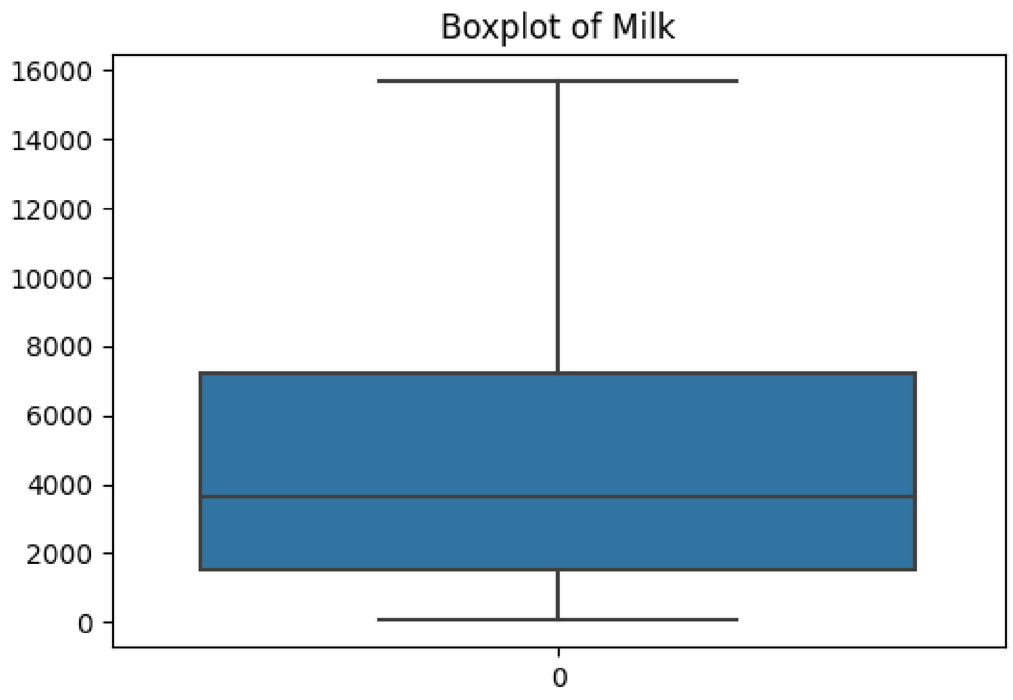
[ ]:
```
# Import necessary libraries
import seaborn as sns
```

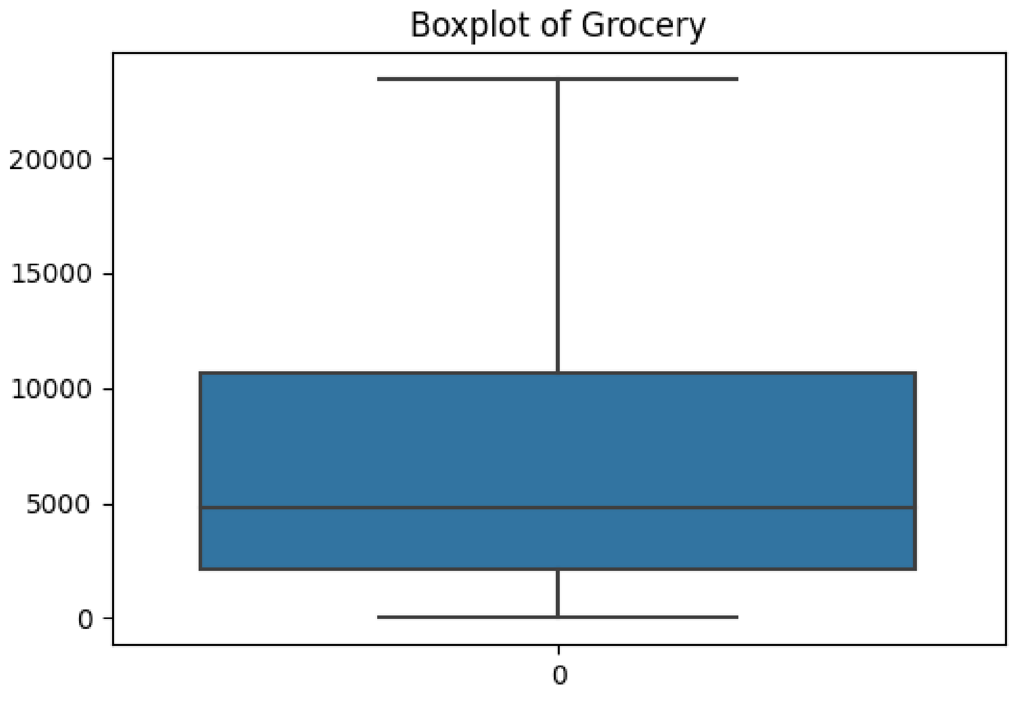
```python
import matplotlib.pyplot as plt

# 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()
```

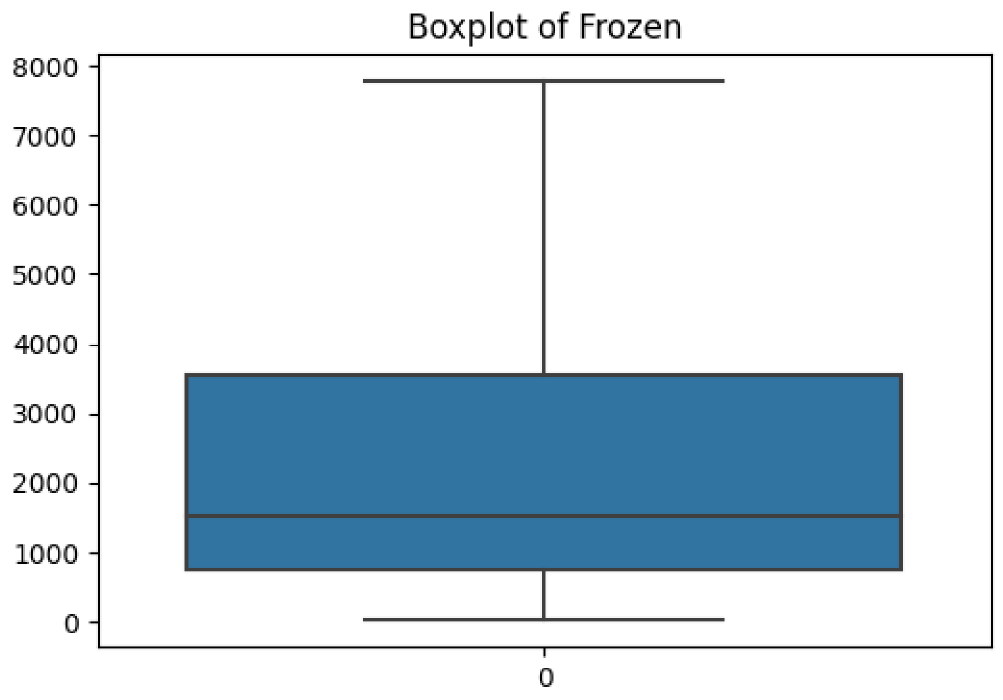


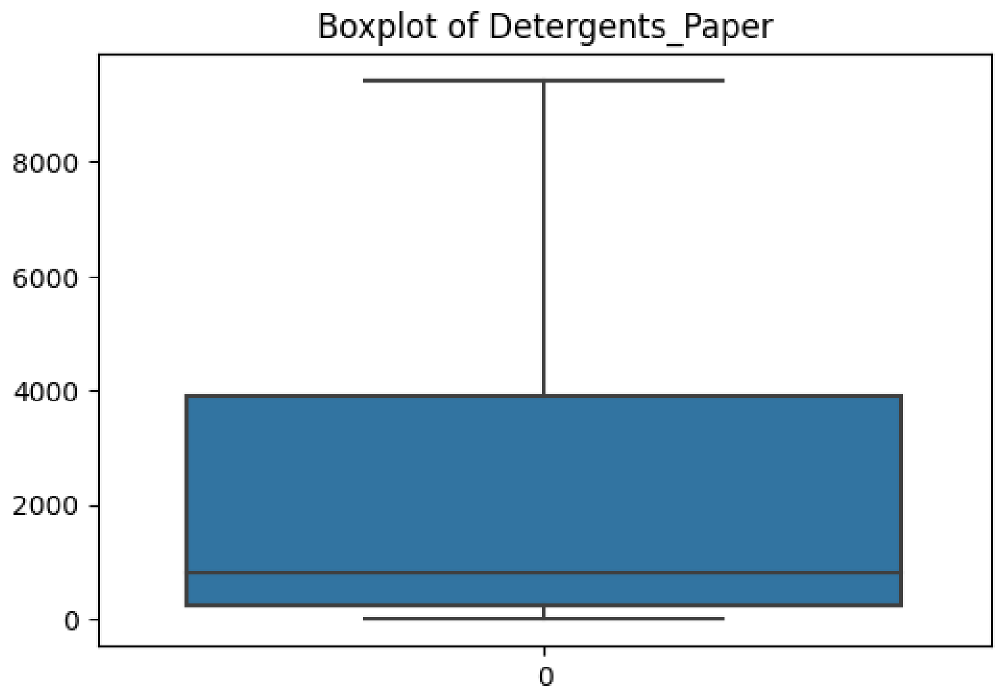

Boxplot of Region
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0

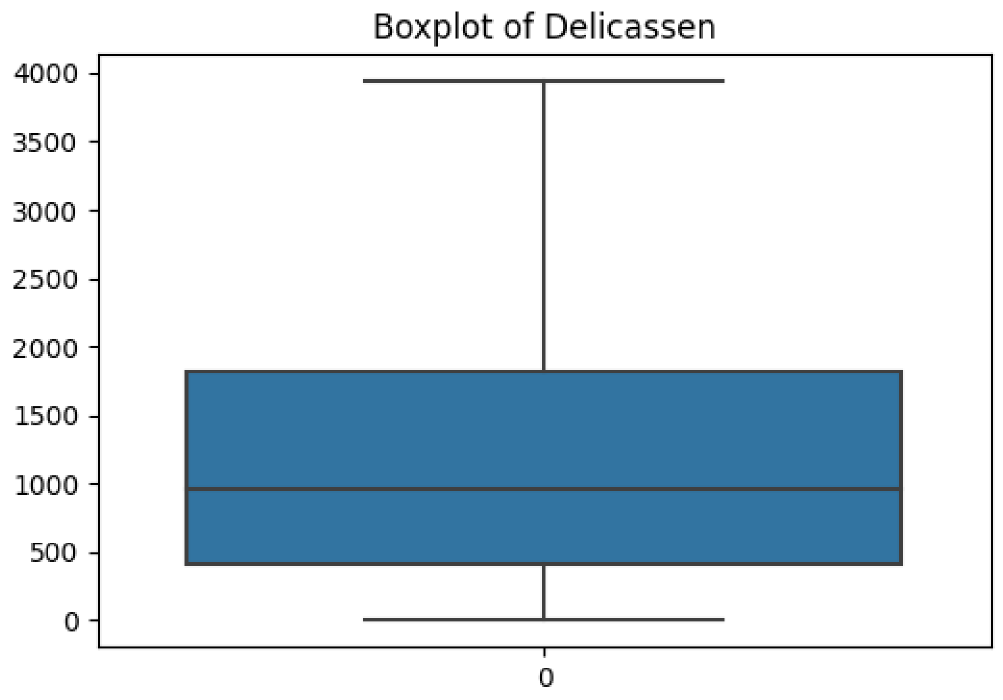

Boxplot of Fresh
35000
30000
25000
20000
15000
10000
5000
0
0

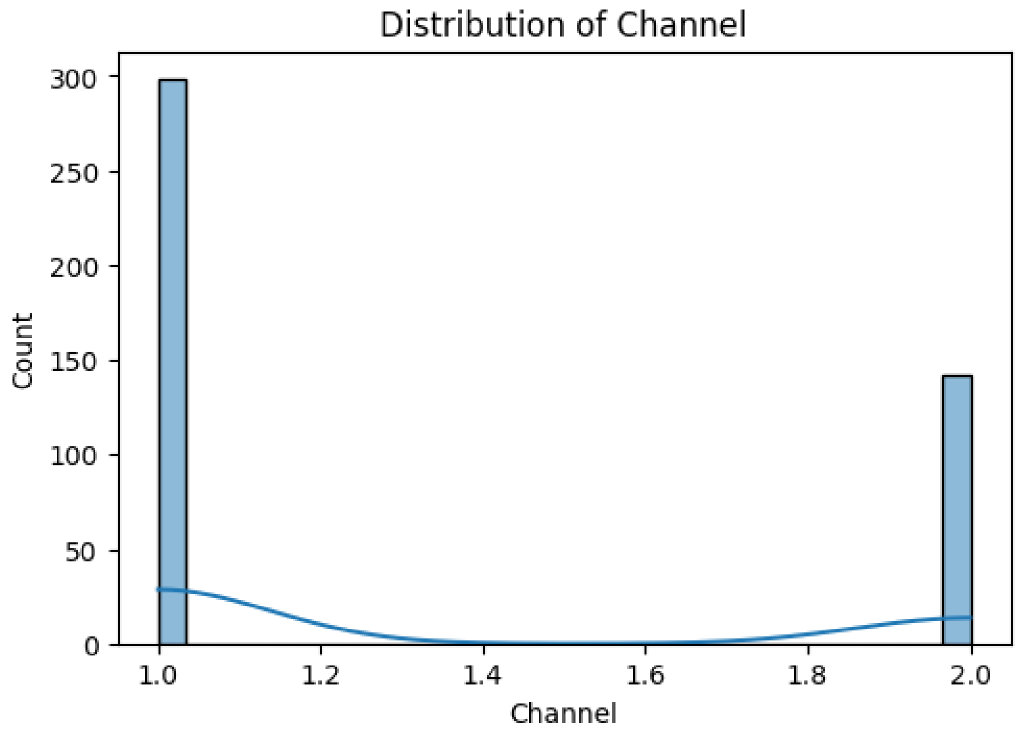

Boxplot of Milk
16000
14000
12000
10000
8000
6000
4000
2000
0
0

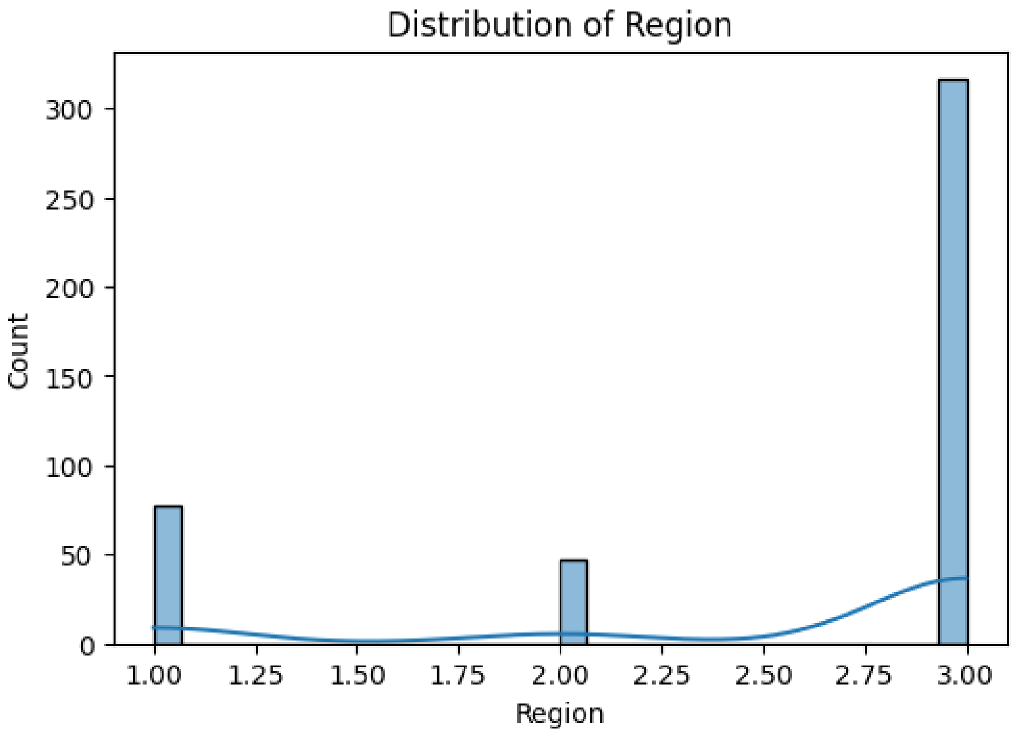

Boxplot of Grocery
20000
15000
10000
5000
0
0

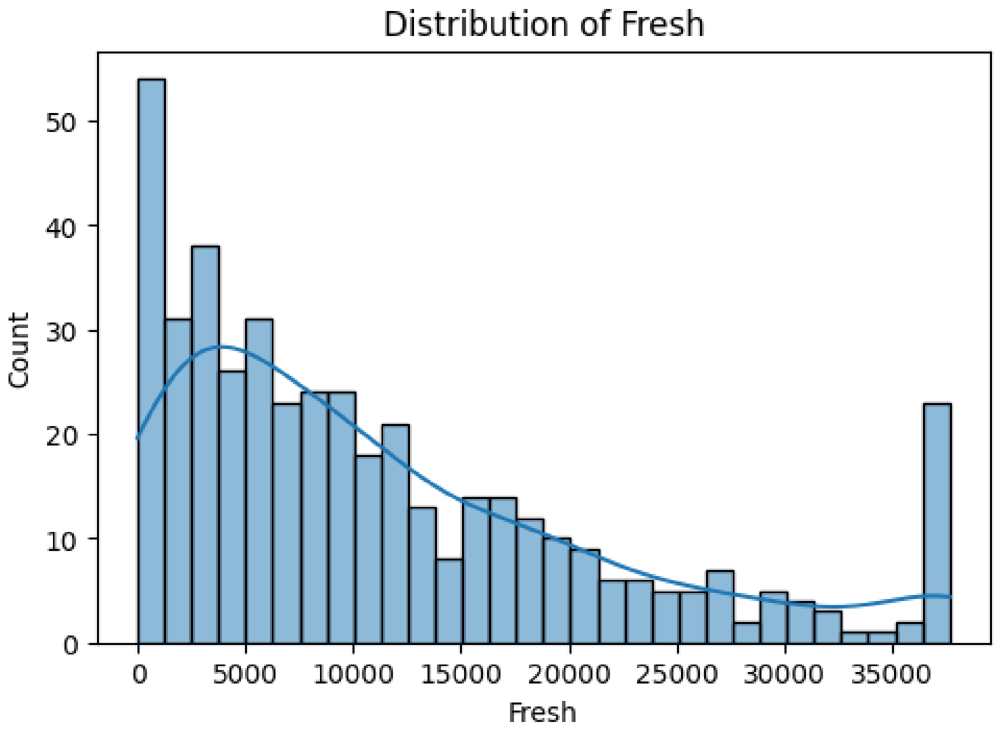

Boxplot of Frozen
8000
7000
6000
5000
4000
3000
2000
1000
0
0

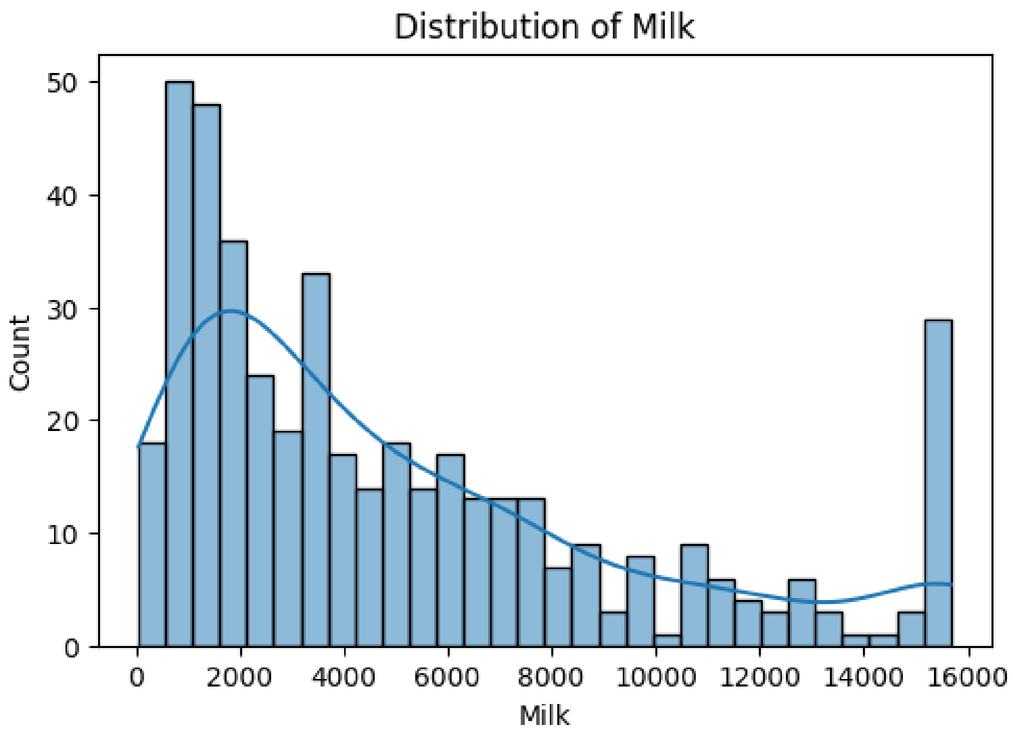

Boxplot of Detergents_Paper
8000
6000
4000
2000
0
0

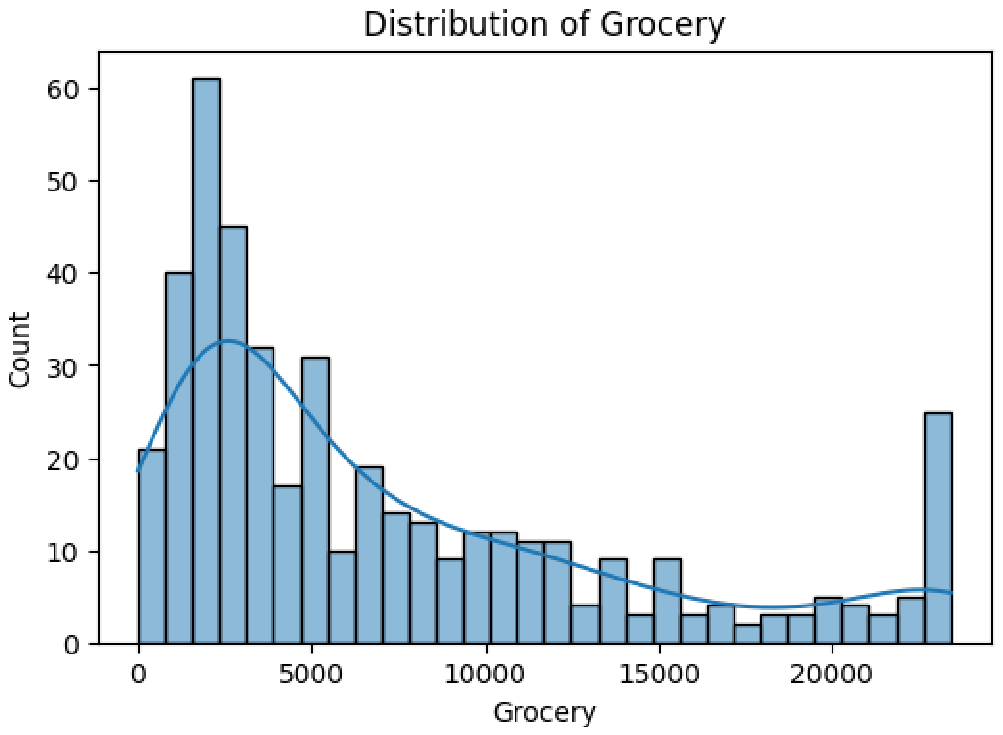

Boxplot of Delicassen
4000
3500
3000
2500
2000
1500
1000
500
0
0

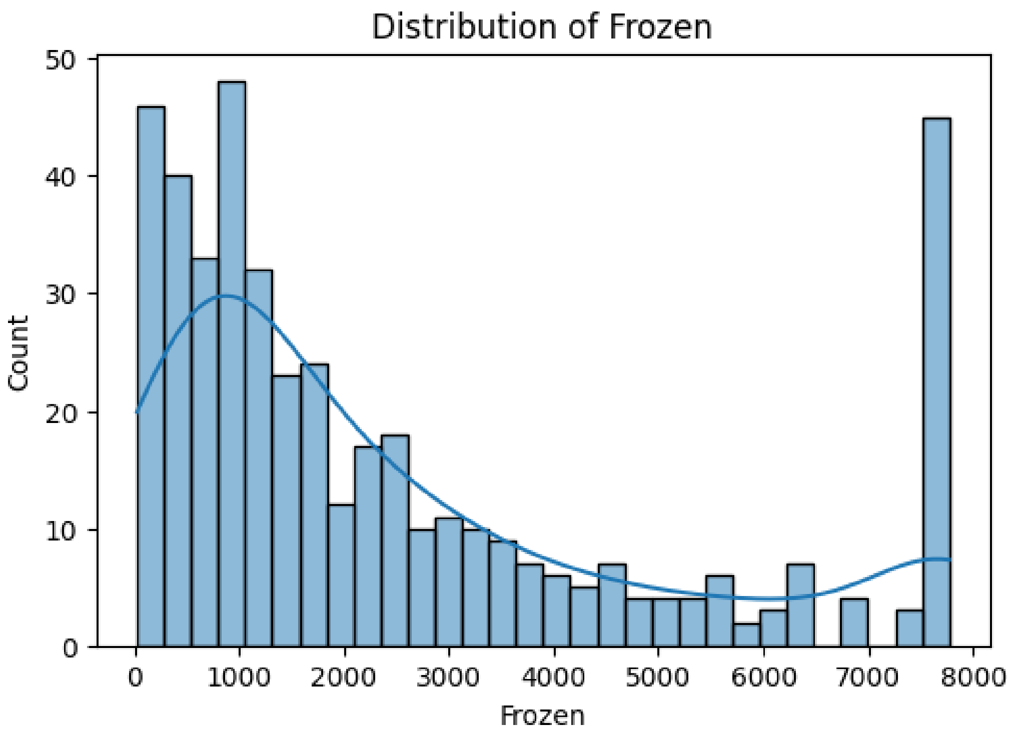

Distribution of Channel
300
250
200
150
100
50
0
Count
1.0
1.2
1.4
1.6
1.8
2.0
Channel

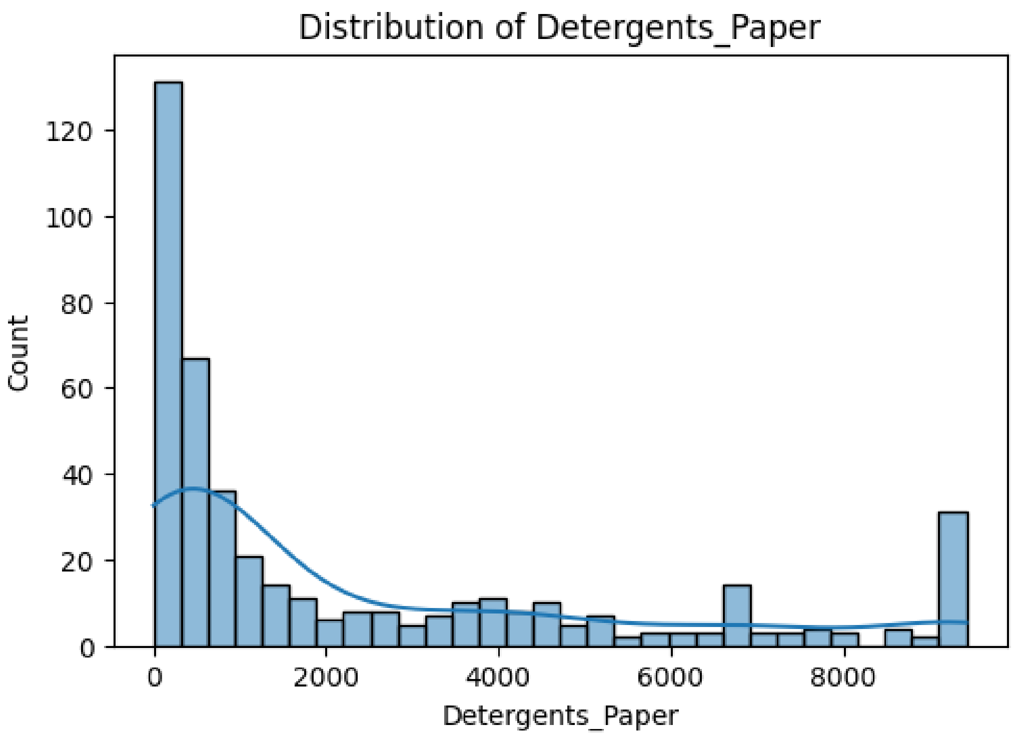

Distribution of Region
Count
Region
300
250
200
150
100
50
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

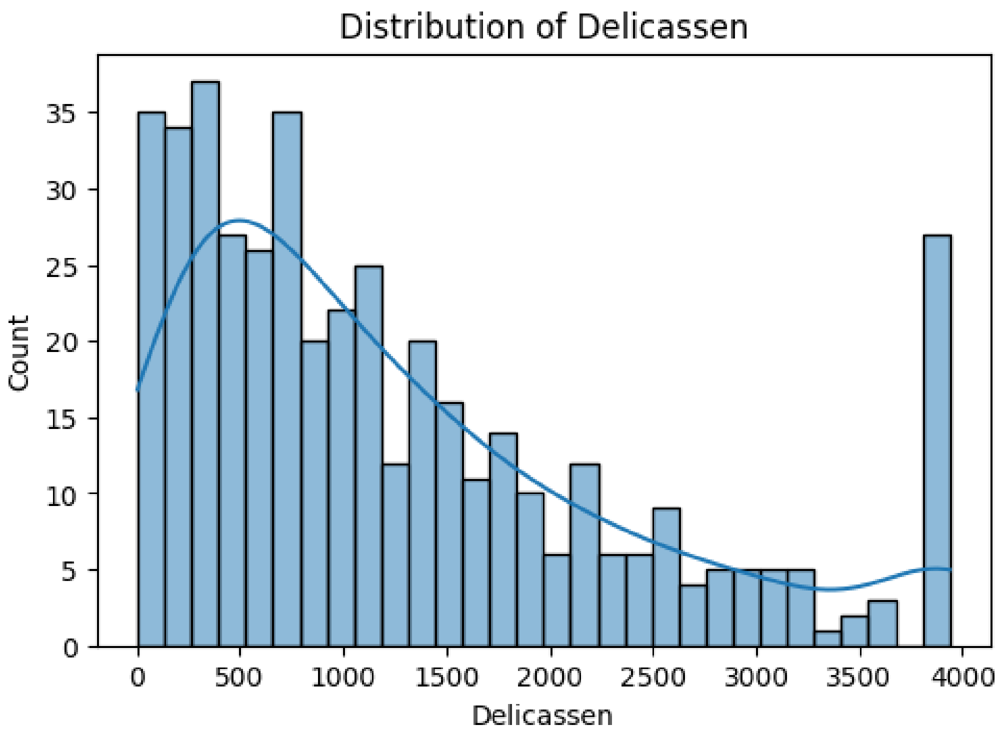

Distribution of Fresh
Count
Fresh
50
40
30
20
10
0
0
5000
10000
15000
20000
25000
30000
35000


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

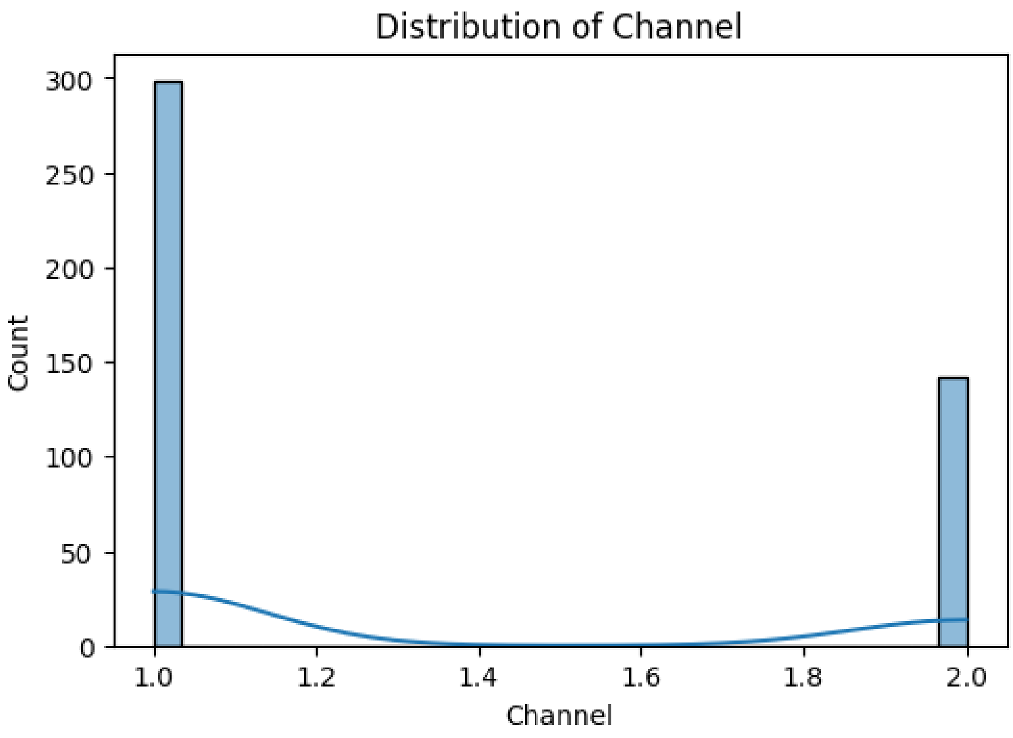


[ ]:
```
# Function to detect outliers
def detect_outliers(dataframe, column):
    Q1 = dataframe[column].quantile(0.25)
    Q3 = dataframe[column].quantile(0.75)
    IQR = Q3 - Q1
    outliers = dataframe[(dataframe[column] < Q1 - 1.5*IQR) |␣
 ↪(dataframe[column] > Q3 + 1.5*IQR)]
    return outliers

# 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)}')
```

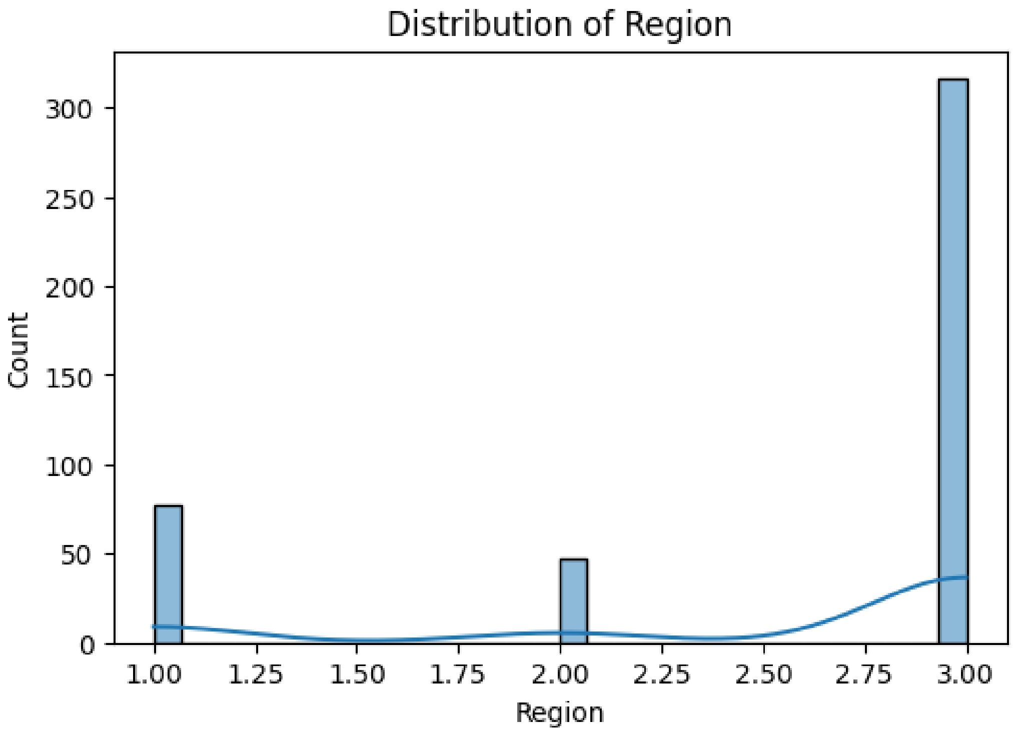
```
Number of outliers in Channel: 0
Number of outliers in Region: 0
Number of outliers in Fresh: 0
Number of outliers in Milk: 0
Number of outliers in Grocery: 0
Number of outliers in Frozen: 0
Number of outliers in Detergents_Paper: 0
Number of outliers in Delicassen: 0
```

```
[ ]: # 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()
```

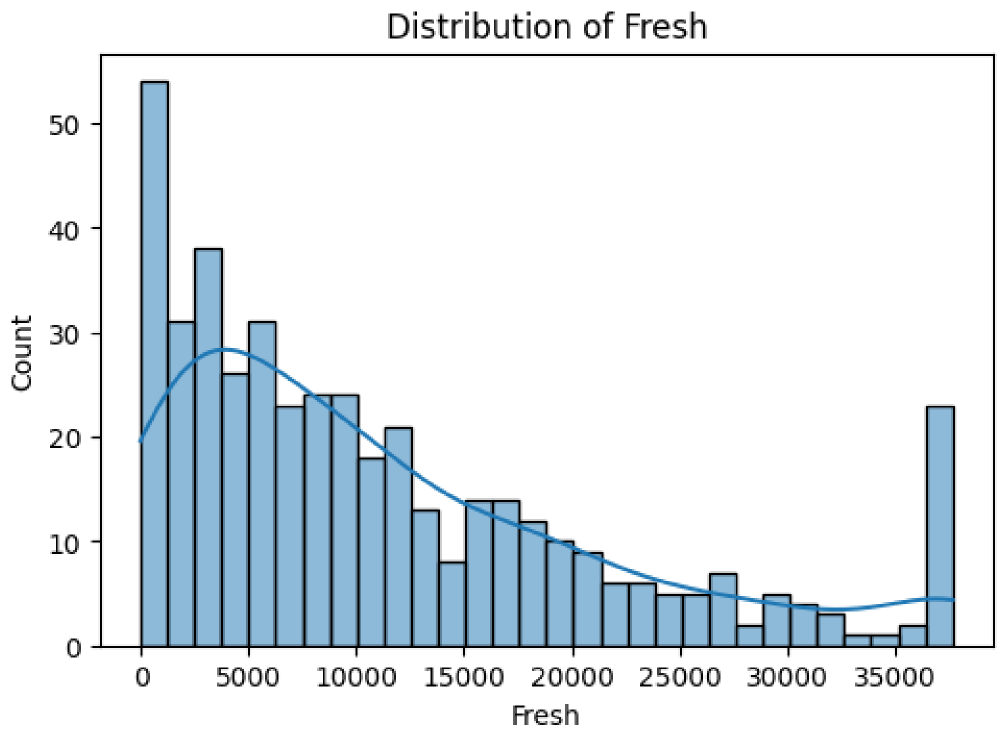
```
Descriptive Statistics:
          Channel      Region         Fresh          Milk       Grocery  \
count  440.000000  440.000000    440.000000    440.000000    440.00000
mean     1.322727    2.543182  11357.568182   5048.592045   7236.37500
std      0.468052    0.774272  10211.542235   4386.377073   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   Delicassen
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
```

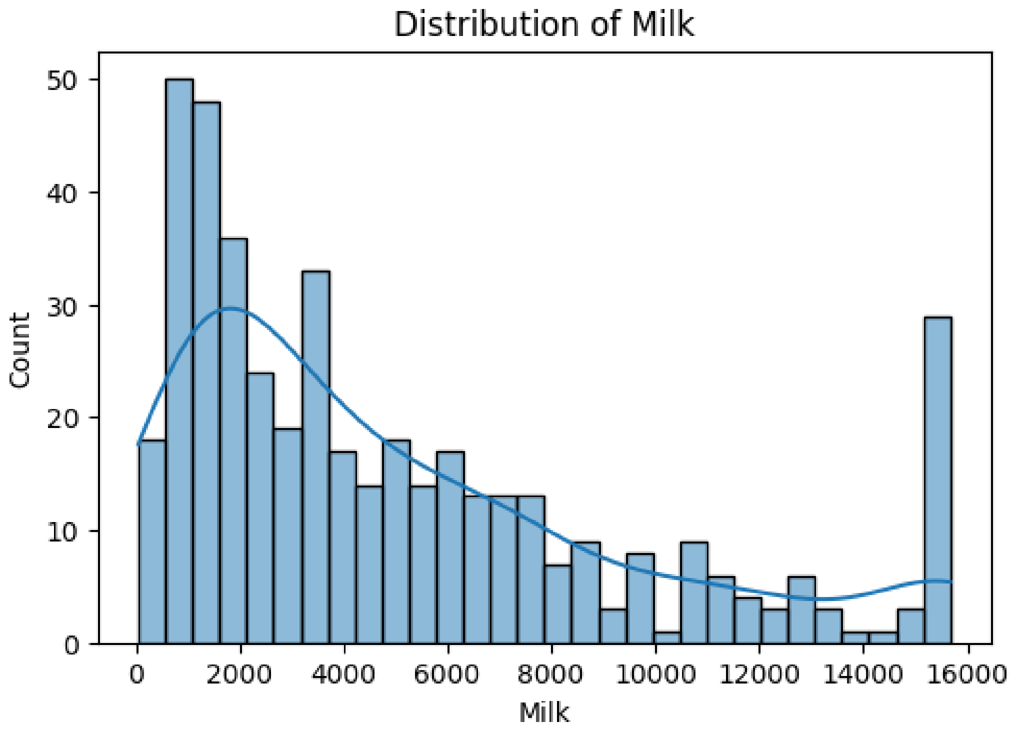


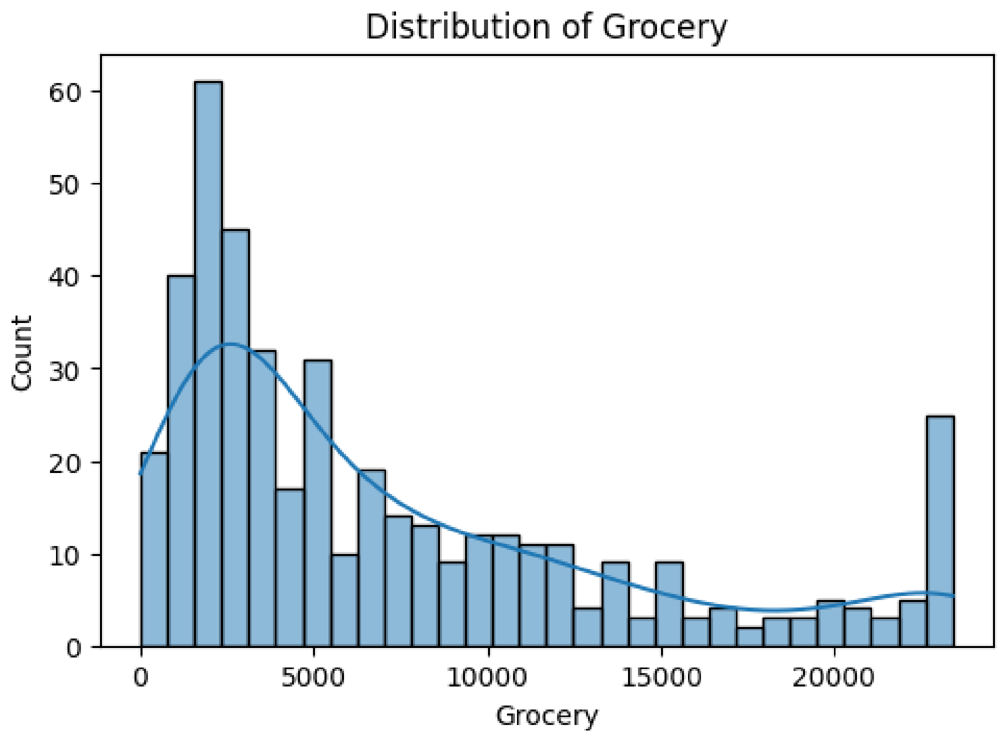
Distribution of Channel
Count
Channel
0
50
100
150
200
250
300
1.0
1.2
1.4
1.6
1.8
2.0

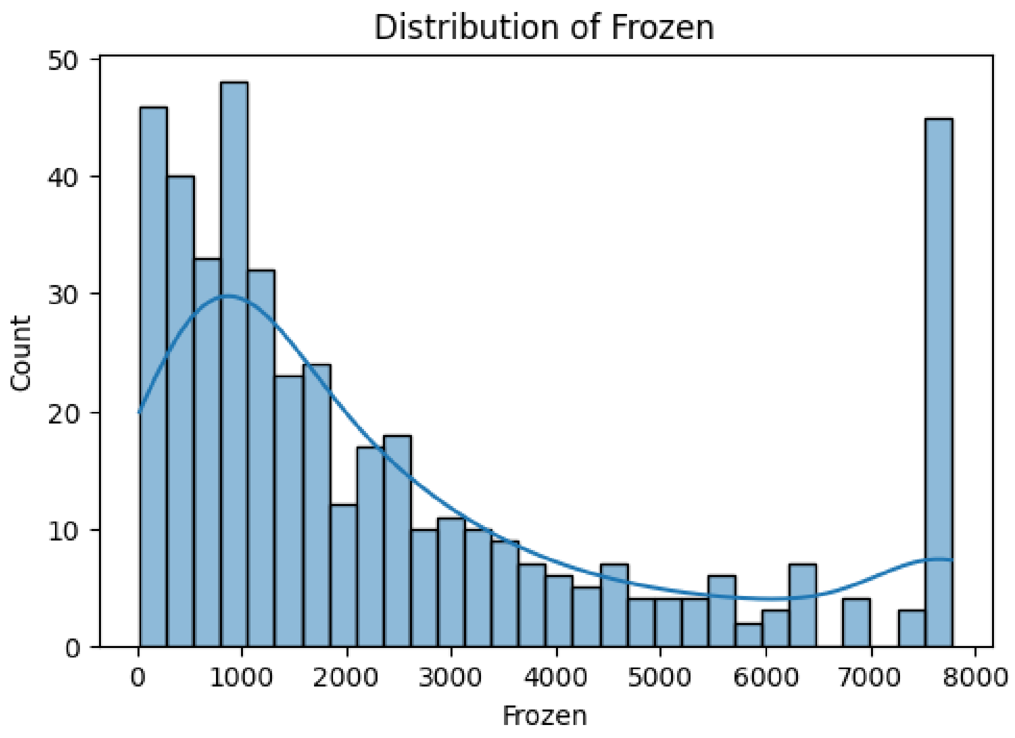


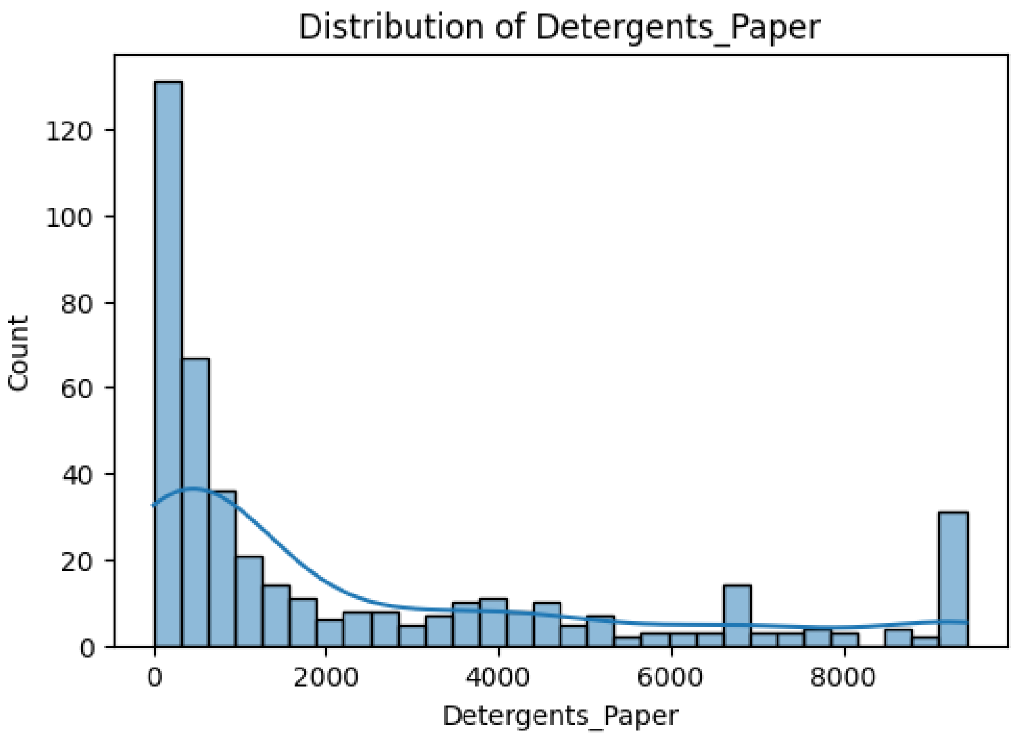
Distribution of Region
Count
Region
0
50
100
150
200
250
300
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

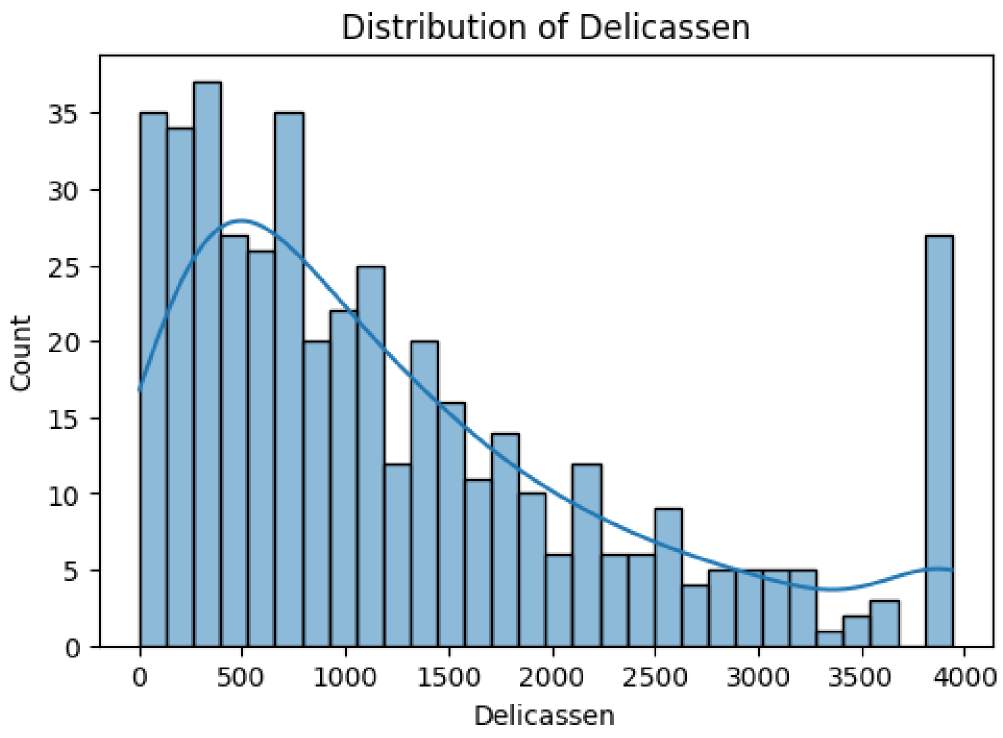

Distribution of Fresh
Count
0
10
20
30
40
50
0
5000
10000
15000
20000
25000
30000
35000
Fresh

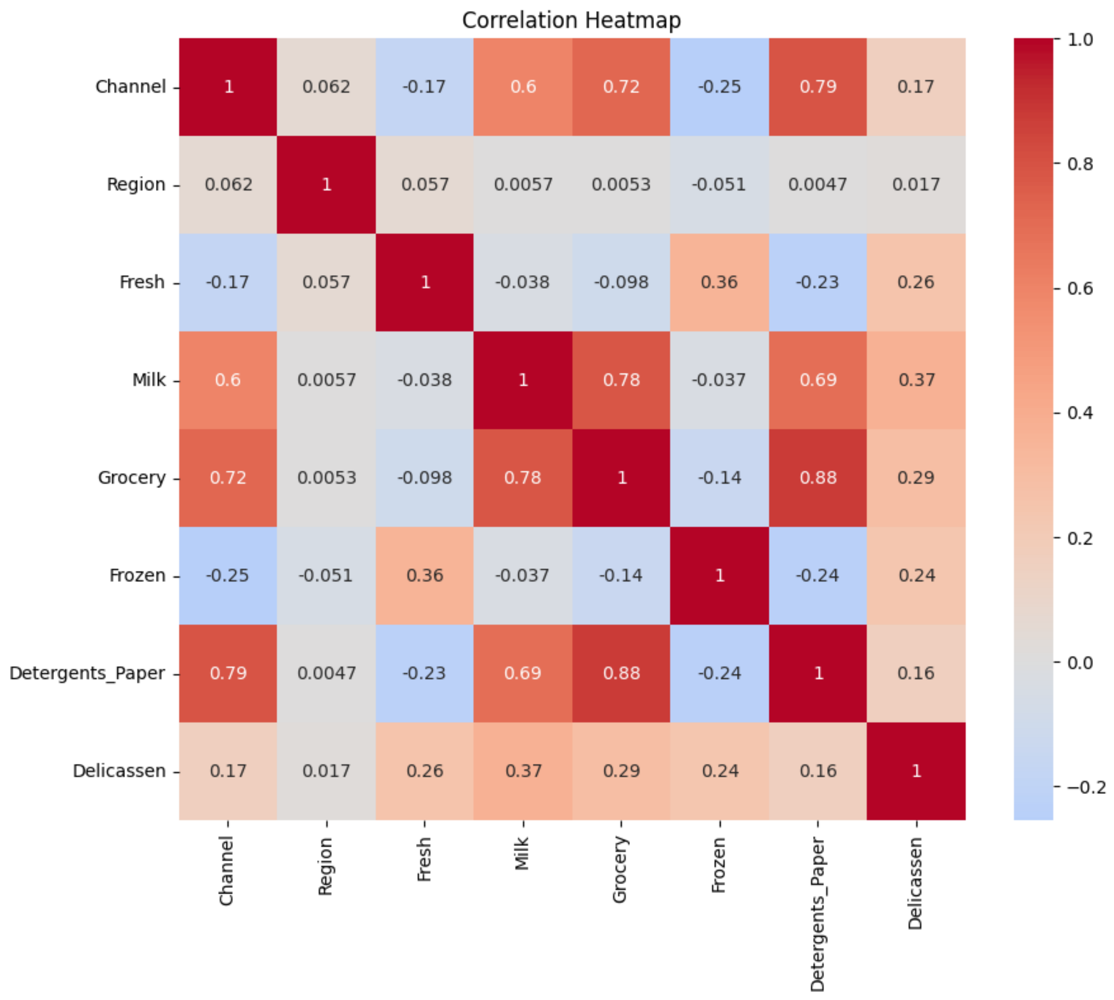

Distribution of Milk
Count
0
10
20
30
40
50
0
2000
4000
6000
8000
10000
12000
14000
16000
Milk


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


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


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



	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Channel	1	0.062	-0.17	0.6	0.72	-0.25	0.79	0.17
Region	0.062	1	0.057	0.0057	0.0053	-0.051	0.0047	0.017
Fresh	-0.17	0.057	1	-0.038	-0.098	0.36	-0.23	0.26
Milk	0.6	0.0057	-0.038	1	0.78	-0.037	0.69	0.37
Grocery	0.72	0.0053	-0.098	0.78	1	-0.14	0.88	0.29
Frozen	-0.25	-0.051	0.36	-0.037	-0.14	1	-0.24	0.24
Detergents_Paper	0.79	0.0047	-0.23	0.69	0.88	-0.24	1	0.16
Delicassen	0.17	0.017	0.26	0.37	0.29	0.24	0.16	1

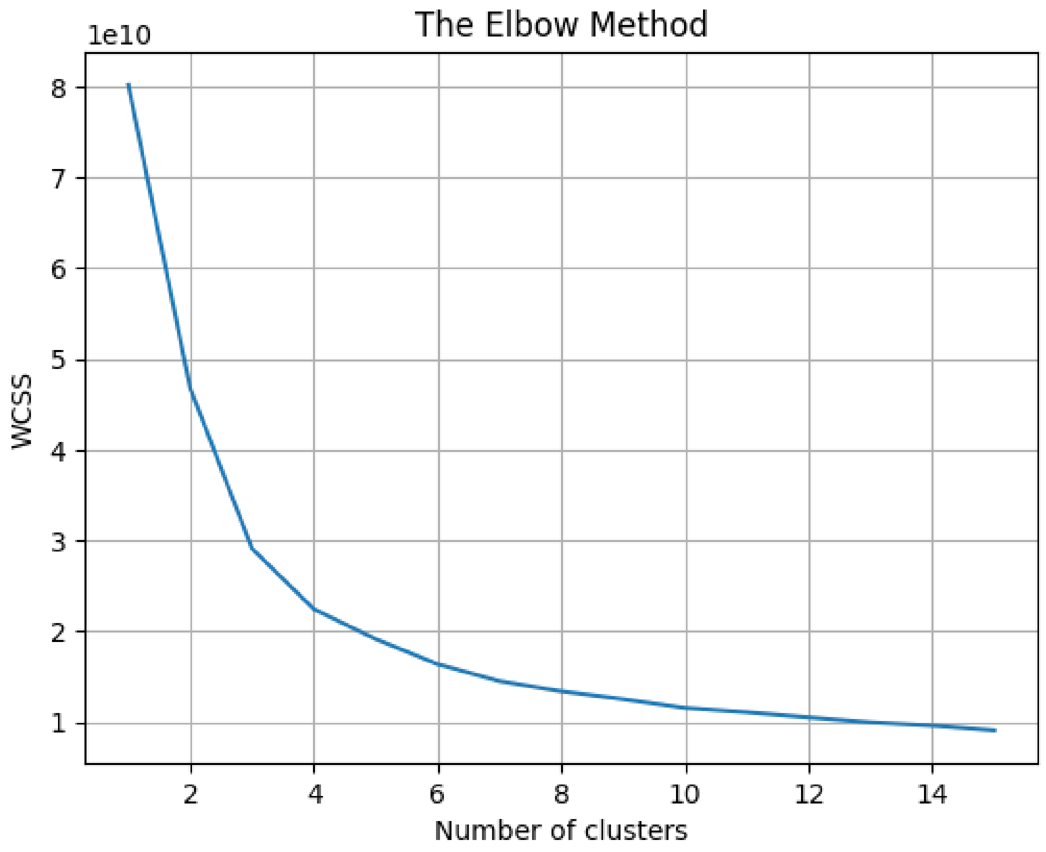
```
[ ]: from sklearn.preprocessing import StandardScaler

     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_)
```

```python
# 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(
```



[ ]:
```python
from sklearn.cluster import KMeans

# Build the model
kmeans = KMeans(n_clusters=3, 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

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

/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())
```

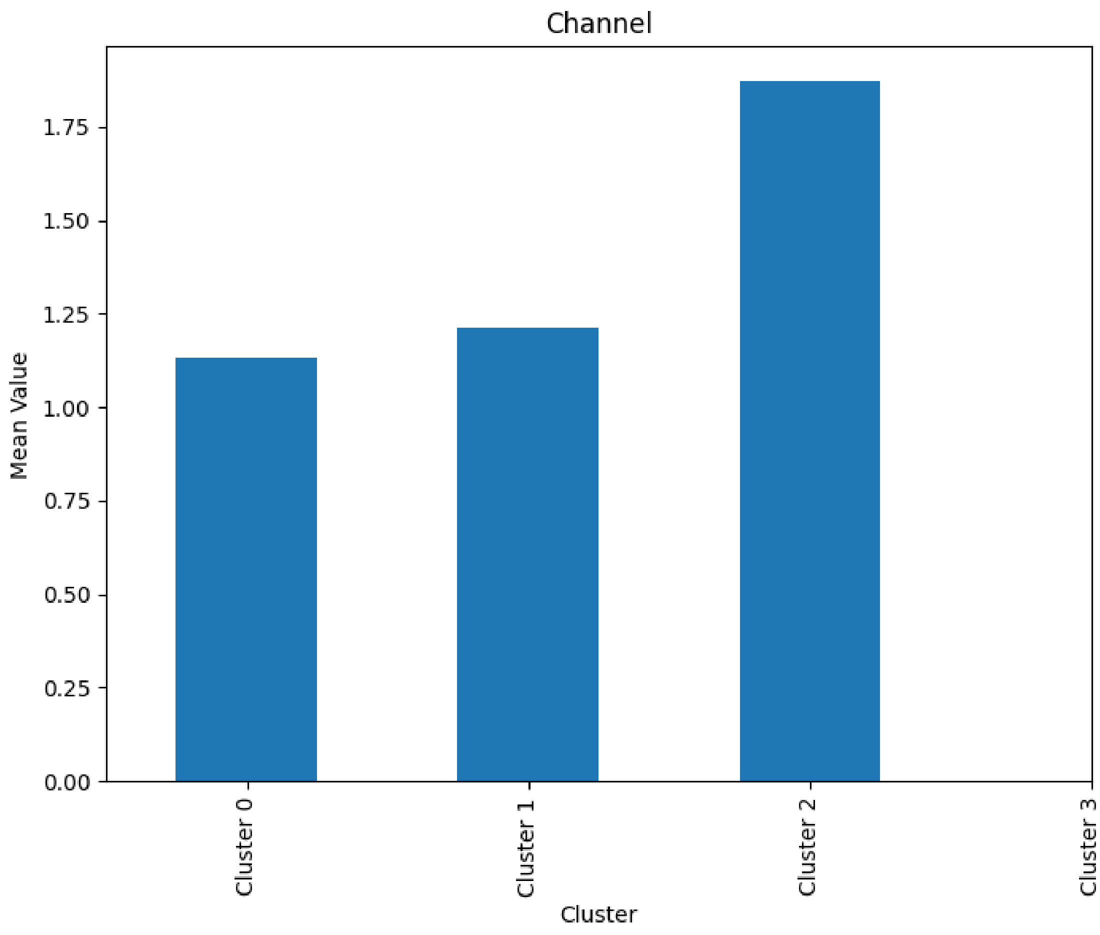
```
Cluster Sizes:
 0    227
1    112
```

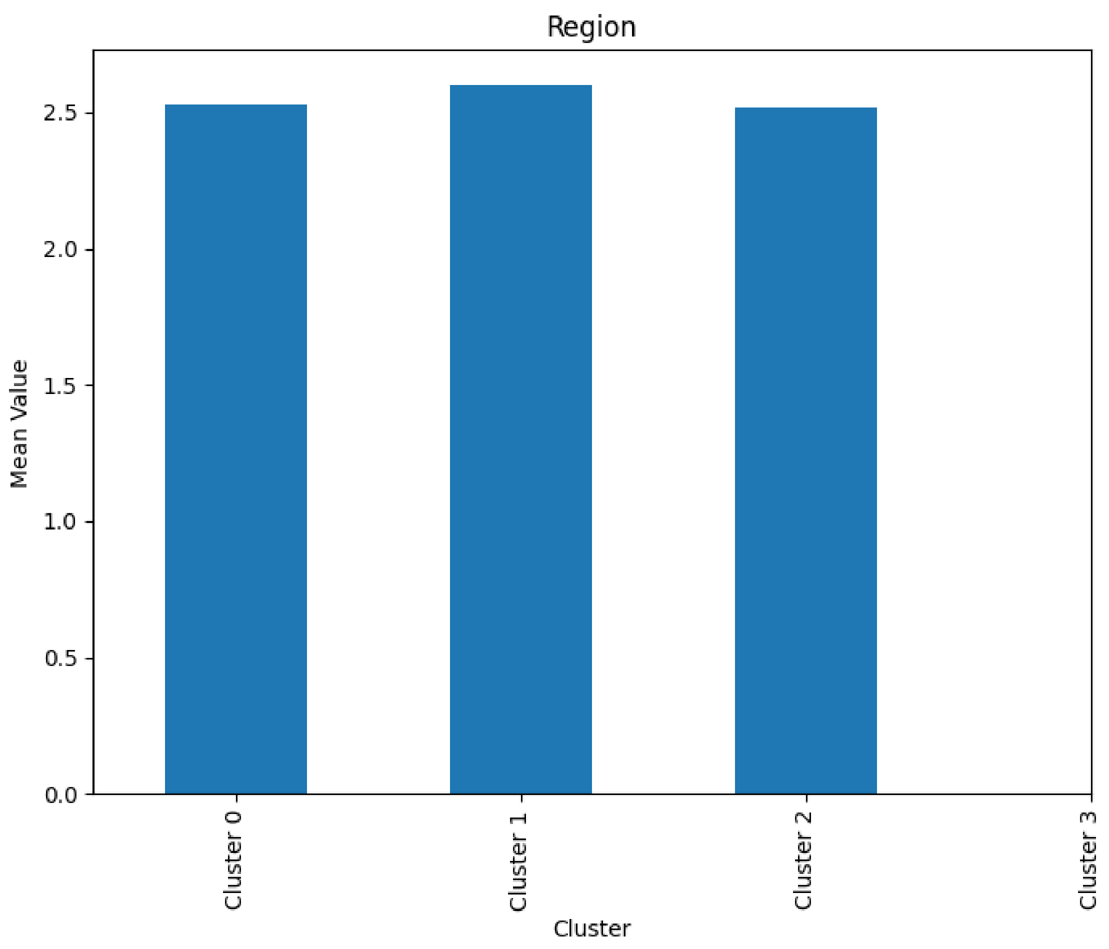
```
2    101
Name: Cluster, dtype: int64

Cluster  0
          Channel      Region         Fresh          Milk       Grocery  \
count  227.000000  227.000000    227.000000    227.000000    227.000000
mean     1.132159    2.528634   6880.828194   3004.604626   3603.237885
std      0.339412    0.788647   4497.653118   2608.249620   2498.211340
min      1.000000    1.000000      3.000000     55.000000    137.000000
25%      1.000000    2.000000   2929.000000   1070.500000   1666.000000
50%      1.000000    3.000000   6758.000000   2160.000000   2824.000000
75%      1.000000    3.000000  10334.500000   3965.500000   5163.500000
max      2.000000    3.000000  16260.000000  15676.125000  11593.000000

            Frozen  Detergents_Paper   Delicassen  Cluster
count   227.000000        227.000000   227.000000    227.0
mean   2326.412996        984.233480   963.896476      0.0
std    2264.692928       1235.547191   893.981219      0.0
min      47.000000          3.000000     3.000000      0.0
25%     663.500000        194.500000   320.500000      0.0
50%    1439.000000        402.000000   686.000000      0.0
75%    3283.500000       1236.500000  1333.000000      0.0
max    7772.250000       5316.000000  3938.250000      0.0

Cluster  1
          Channel      Region         Fresh          Milk       Grocery  \
count  112.000000  112.000000    112.000000    112.000000    112.000000
mean     1.214286    2.598214  25992.053571   4629.829241   6026.292411
std      0.412170    0.740828   7518.249908   3957.886679   5094.821164
min      1.000000    1.000000  16448.000000    134.000000      3.000000
25%      1.000000    2.750000  19076.750000   1795.750000   2308.000000
50%      1.000000    3.000000  24778.500000   3645.000000   4603.000000
75%      1.000000    3.000000  31738.500000   6202.000000   8259.750000
max      2.000000    3.000000  37642.750000  15676.125000  23409.875000

            Frozen  Detergents_Paper   Delicassen  Cluster
count   112.000000        112.000000   112.000000    112.0
mean   3798.729911       1290.006696  1679.750000      1.0
std    2745.000953       1759.882080  1177.995942      0.0
min     118.000000          3.000000     3.000000      1.0
25%    1283.750000        245.500000   785.500000      1.0
50%    3028.500000        593.000000  1374.500000      1.0
75%    7341.000000       1543.750000  2518.250000      1.0
max    7772.250000       9419.875000  3938.250000      1.0

Cluster  2
          Channel      Region         Fresh          Milk       Grocery  \
count  101.000000  101.000000    101.000000    101.000000    101.000000
```

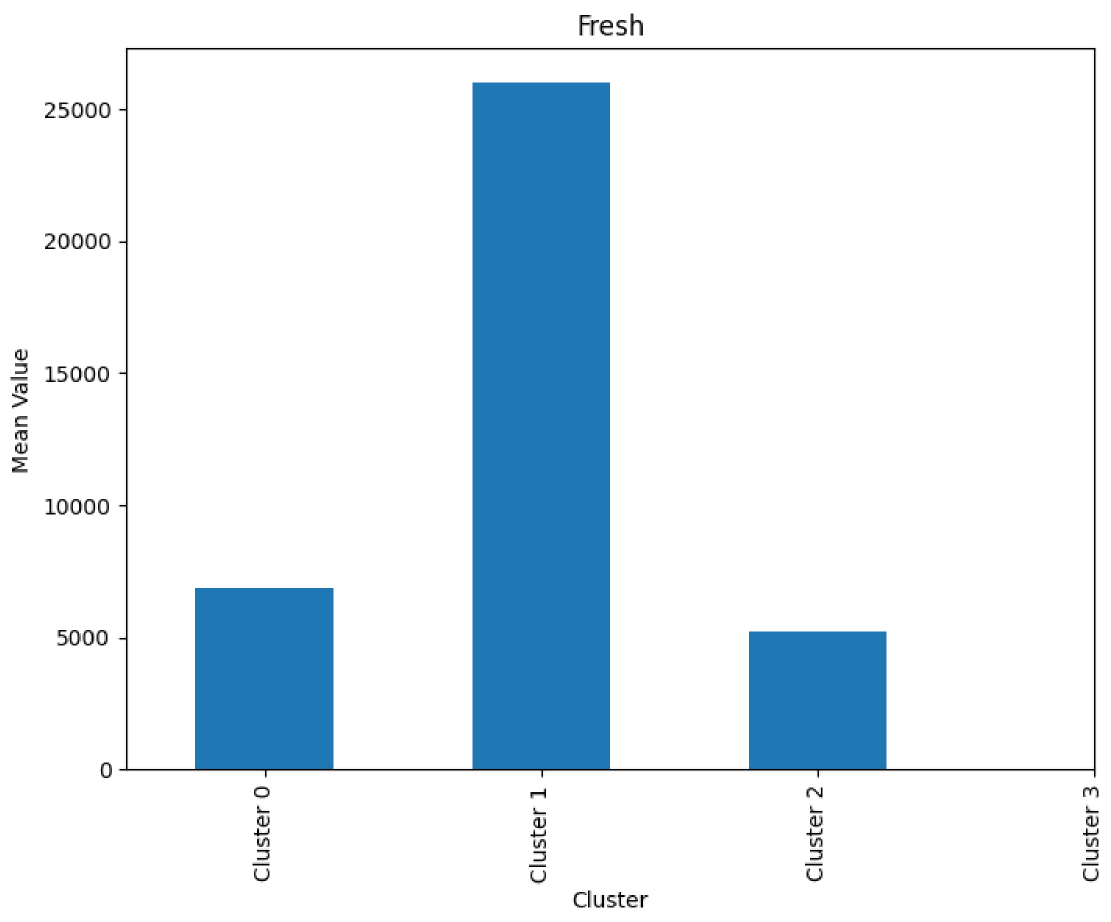
```
mean      1.871287    2.514851   5190.811881  10106.875000  16743.814356
std       0.336552    0.782481   5053.693043   4022.429078   5021.119664
min       1.000000    1.000000     18.000000   1266.000000   8852.000000
25%       2.000000    2.000000   1210.000000   7097.000000  11924.000000
50%       2.000000    3.000000   3830.000000   9933.000000  15541.000000
75%       2.000000    3.000000   7362.000000  13316.000000  22182.000000
max       2.000000    3.000000  22925.000000  15676.125000  23409.875000

            Frozen  Detergents_Paper   Delicassen  Cluster
count   101.000000        101.000000   101.000000    101.0
mean   1480.834158       6780.688119  1489.289604      2.0
std    1581.181399       2401.241628  1163.558720      0.0
min      25.000000        241.000000     3.000000      2.0
25%     388.000000       5038.000000   555.000000      2.0
50%     937.000000       6846.000000  1295.000000      2.0
75%    1840.000000       9419.875000  2153.000000      2.0
max    7772.250000       9419.875000  3938.250000      2.0

Cluster  3
       Channel  Region  Fresh  Milk  Grocery  Frozen  Detergents_Paper  \
count      0.0     0.0    0.0   0.0      0.0     0.0               0.0
mean       NaN     NaN    NaN   NaN      NaN     NaN               NaN
std        NaN     NaN    NaN   NaN      NaN     NaN               NaN
min        NaN     NaN    NaN   NaN      NaN     NaN               NaN
25%        NaN     NaN    NaN   NaN      NaN     NaN               NaN
50%        NaN     NaN    NaN   NaN      NaN     NaN               NaN
75%        NaN     NaN    NaN   NaN      NaN     NaN               NaN
max        NaN     NaN    NaN   NaN      NaN     NaN               NaN

       Delicassen  Cluster
count         0.0      0.0
mean          NaN      NaN
std           NaN      NaN
min           NaN      NaN
25%           NaN      NaN
50%           NaN      NaN
75%           NaN      NaN
max           NaN      NaN
```

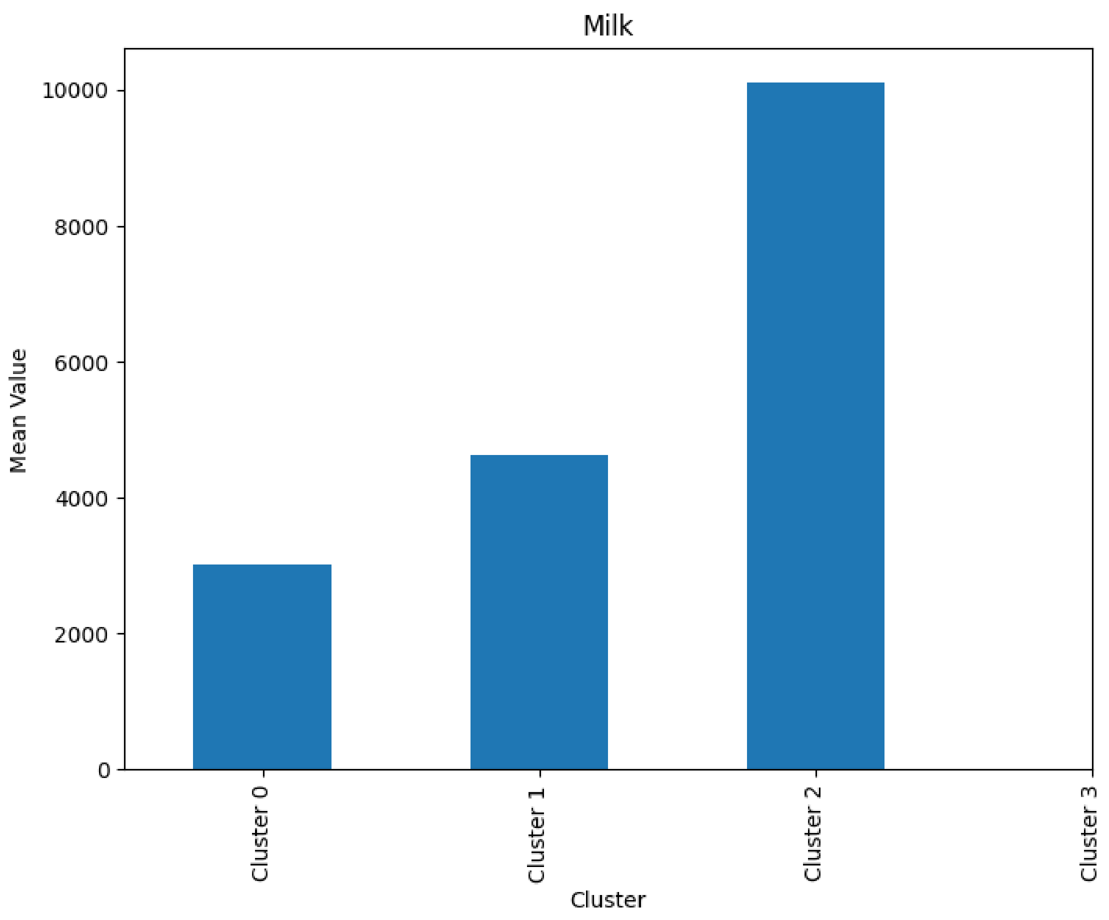
[ ]:
```
# Calculate the mean values for each feature per cluster
cluster_means = df.groupby('Cluster').mean()

# 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
```

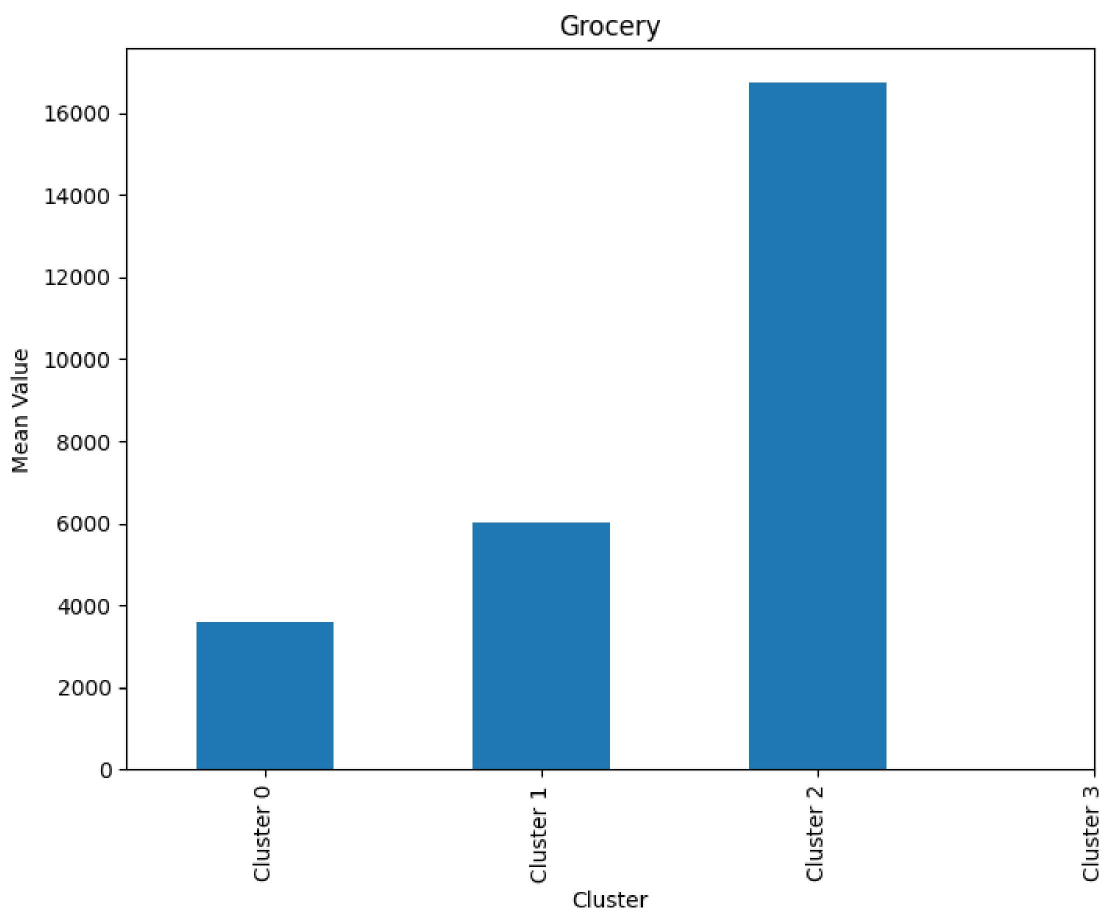
```
for feature in cluster_means.index:
    cluster_means.loc[feature].plot(kind='bar', figsize=(8,6))
    plt.title(feature)
    plt.ylabel('Mean Value')
    plt.xticks(ticks=range(4), labels=['Cluster 0', 'Cluster 1', 'Cluster 2',␣
 ↪'Cluster 3'])
    plt.show()
```

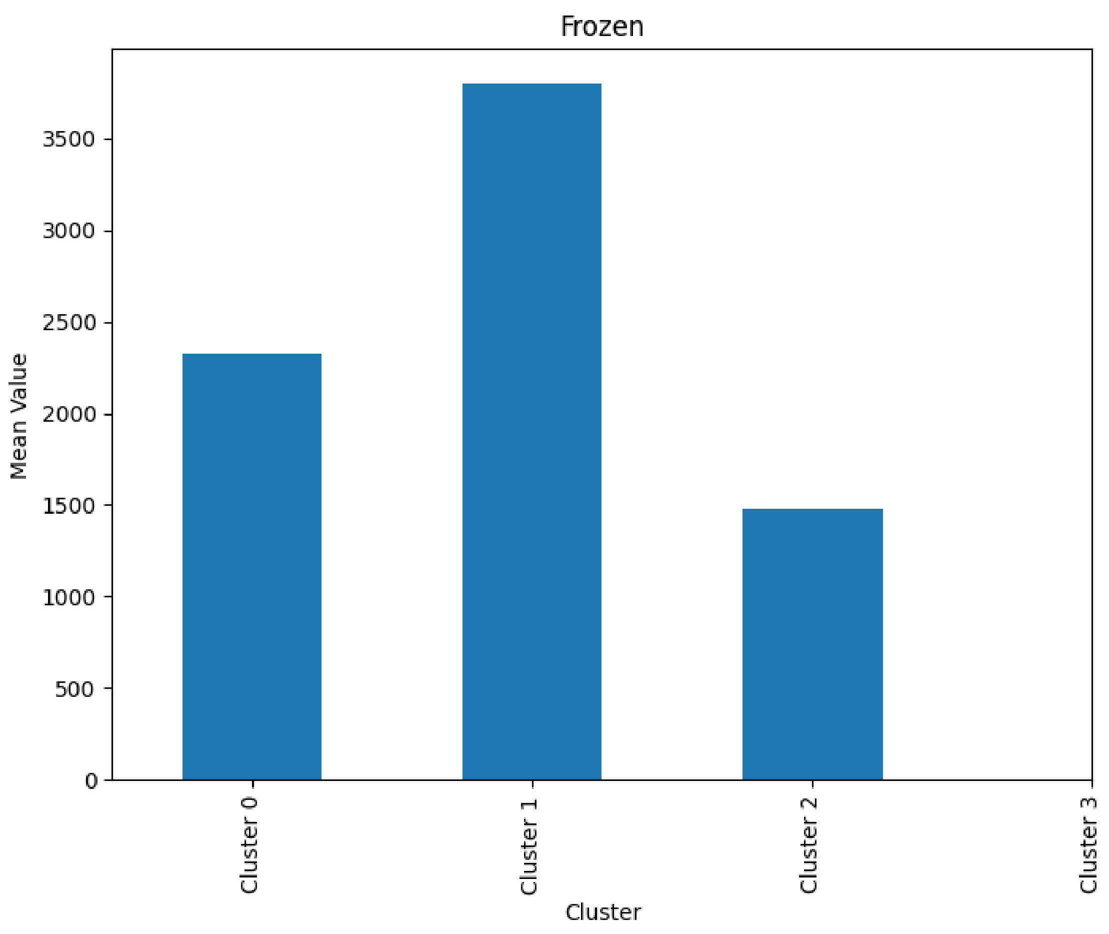


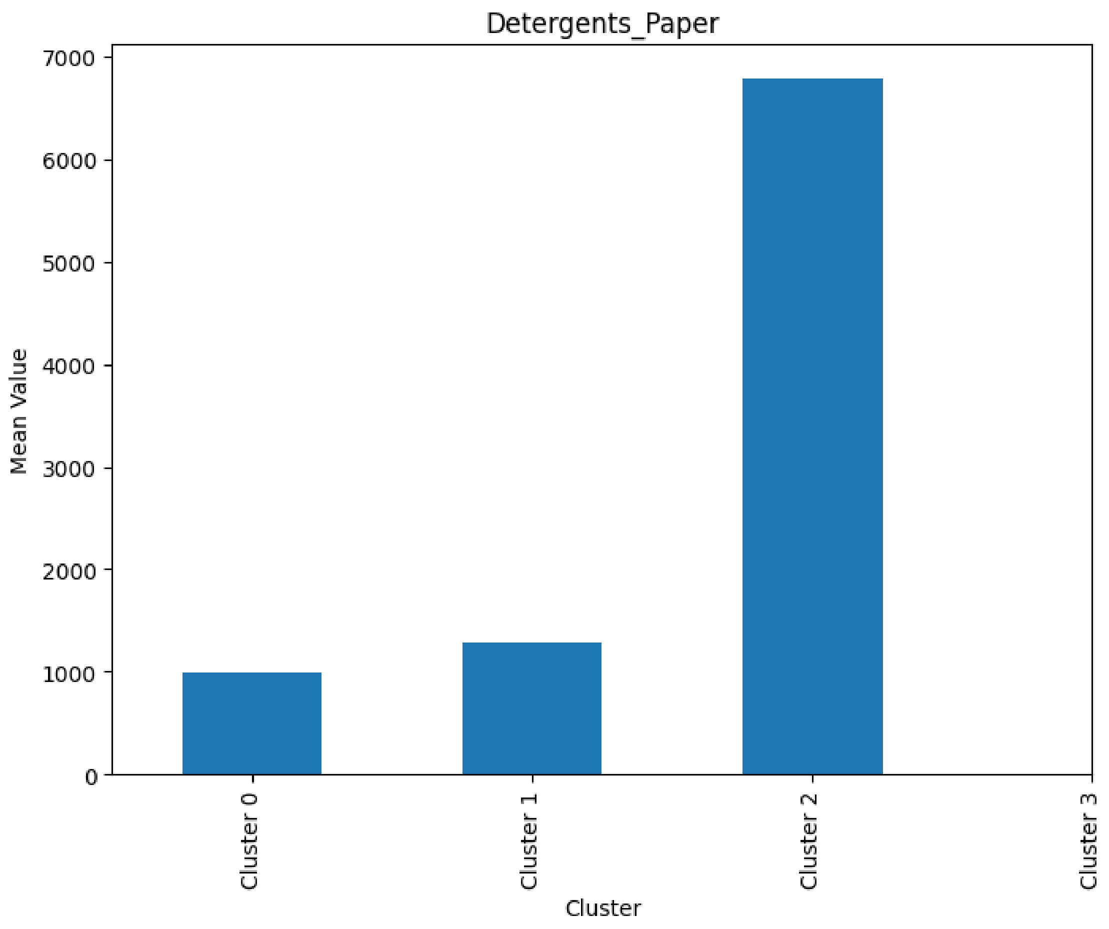

Region
Mean Value
0.0
0.5
1.0
1.5
2.0
2.5
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster

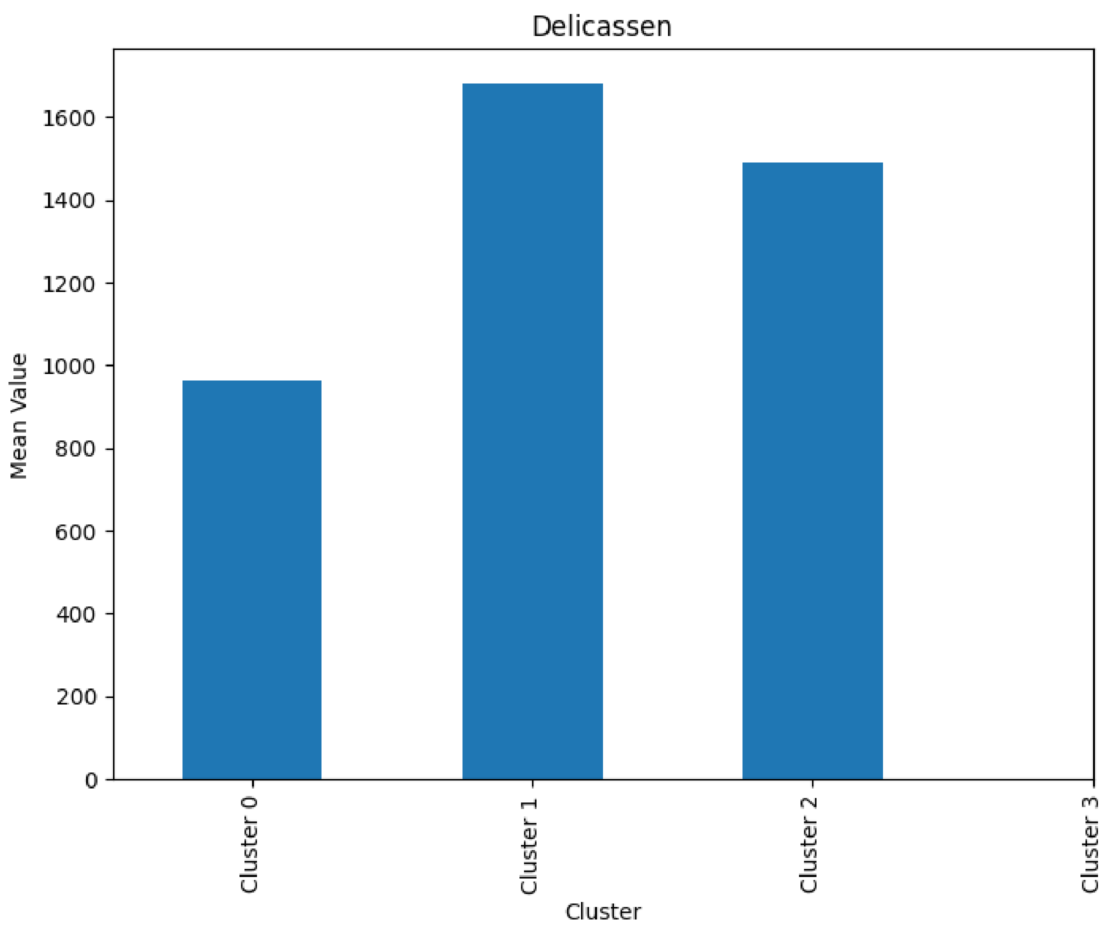

Fresh
Mean Value
25000
20000
15000
10000
5000
0
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster


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


Grocery
Mean Value
0
2000
4000
6000
8000
10000
12000
14000
16000
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster

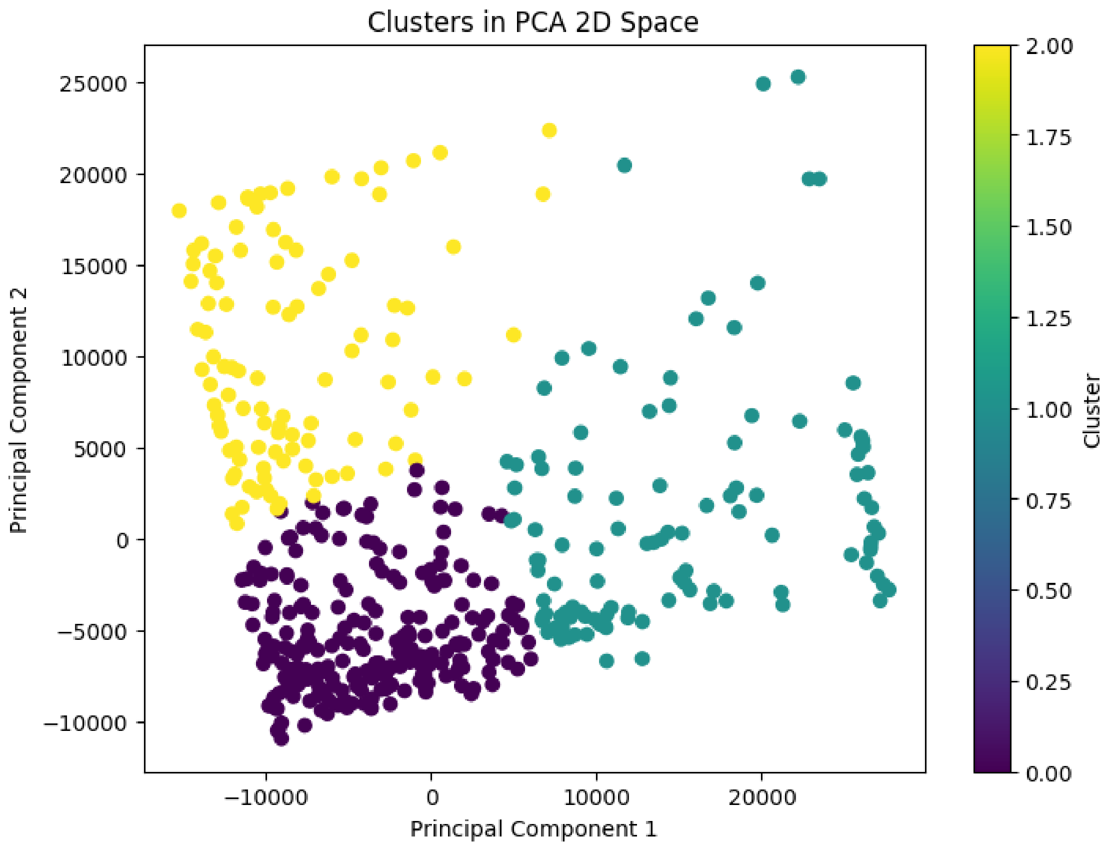

Frozen
Mean Value
0
500
1000
1500
2000
2500
3000
3500
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster


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



```
[ ]: 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:

Use of the clustered data

- Customer Segmentation: Clustered data helps you understand different customer segments based on their purchasing behavior. We can design marketing campaigns that are more relevant to each cluster's preferences.
- Product Recommendations: By knowing which products are frequently purchased together within each cluster, we can make personalized product recommendations to customers.
- Inventory Management: Clustering can help optimize inventory management by ensuring that the right products are stocked in the right quantities to meet the preferences of each cluster.
- Supply Chain Optimization: We can optimize our supply chain operations by tailoring delivery schedules and routes to each cluster's needs.
- Customer Retention: By understanding the characteristics of each cluster, we can develop retention strategies that are more likely to resonate with specific customer groups.
- Market Expansion: Clustering can also help identify potential new markets or customer segments that are similar to existing clusters.

Different groups of customers, the customer segments, may be affected differently by a specific delivery scheme

- High-Value Customers: If a delivery scheme offers premium or expedited delivery options, high-value customers who value convenience and are willing to pay more may respond positively.
- Budget-Conscious Customers: Customers in this segment may be more price-sensitive and may prefer a cost-effective or free standard delivery scheme.
- Bulk Buyers: If there is a segment of customers who prefer bulk purchases, they might benefit from delivery schemes that offer discounts for larger orders or specialized bulk delivery options.
- Frequent Shoppers: Customers who shop frequently may benefit from subscription-based or loyalty-based delivery schemes. These schemes can encourage repeat purchases and loyalty.