Unsupervised Learning - Project

In this Project, we are going to perform a full unsupervised learning machine learning project on a "Wholesale Data" dataset. The dataset refers to clients of a wholesale distributor. It includes the annual spending in monetary units (m.u.) on diverse product categories

Kaggle Link

Part I : EDA - Exploratory Data Analysis & Pre-processing

The given dataset seems to be a grocery sales dataset containing information about various products sold by a grocery store. To perform an exploratory data analysis (EDA) on this dataset, we can perform the following tasks:

- Data Import: Import the dataset into a statistical software tool such as Python or R.
- Data Cleaning: Check the dataset for any missing or incorrect data and clean the dataset accordingly. This may involve removing or imputing missing data or correcting any obvious errors. Data Description: Generate summary statistics such as mean, median, and standard deviation for each column of the dataset. This will help in understanding the distribution of data in each column.
- Data Visualization: Create various visualizations such as histograms, box plots, scatter plots, and heatmaps to understand the relationships and trends between the different variables in the dataset. For example, we can create a scatter plot between the "Fresh" and "Milk" variables to see if there is any correlation between them.
- Outlier Detection: Check for any outliers in the dataset and determine whether they are valid or erroneous data points.
- Correlation Analysis: Calculate the correlation between different variables in the dataset to determine which variables are highly correlated and which ones are not. For example, we can calculate the correlation between "Grocery" and "Detergents_Paper" to see if there is any relationship between these two variables.
- Data Transformation: If necessary, transform the data by standardizing or normalizing the variables to make them comparable across different scales.
- Feature Selection: Identify the most important features or variables that contribute the most to the overall variance in the dataset. This can be done using various feature selection techniques such as principal component analysis (PCA) or random forest regression.

In [80]:
```
pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\user\conda\lib\site-packages (1.5.
3)Note: you may need to restart the kernel to use updated packages.

Requirement already satisfied: pytz>=2020.1 in c:\users\user\conda\lib\site-packages
(from pandas) (2022.7)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\user\conda\lib\site
-packages (from pandas) (2.8.2)
Requirement already satisfied: numpy>=1.21.0 in c:\users\user\conda\lib\site-packages
(from pandas) (1.23.5)
Requirement already satisfied: six>=1.5 in c:\users\user\conda\lib\site-packages (fro
m python-dateutil>=2.8.1->pandas) (1.16.0)
```

In [81]:

```
import pandas as pd
```

In [82]:

```
# read CVS file
df = pd.read_csv('Wholesale_Data.csv')
```

In [83]:

```
df.head()
```

Out[83]:

	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

In [84]:

```
# Check for missing values in each column
missing_values = df.isnull().sum()

print(missing_values)
```

```
Channel             0
Region              0
Fresh               0
Milk                0
Grocery             0
Frozen              0
Detergents_Paper    0
Delicassen          0
dtype: int64
```

In [85]:

```
# Remove rows with any missing values
df = df.dropna()
```

In [86]:

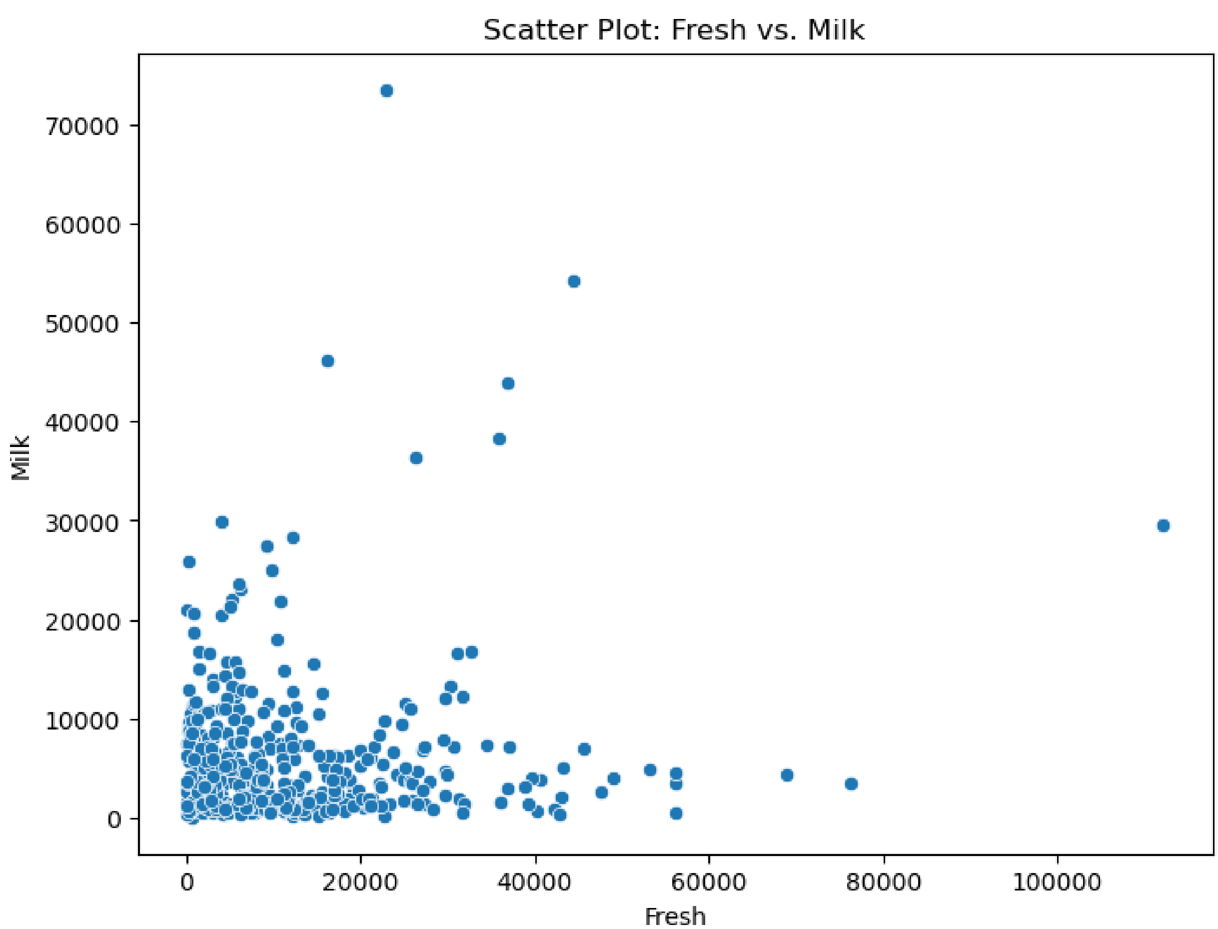
```
# Impute missing values using column means
df = df.fillna(df.mean())
```

In [87]:

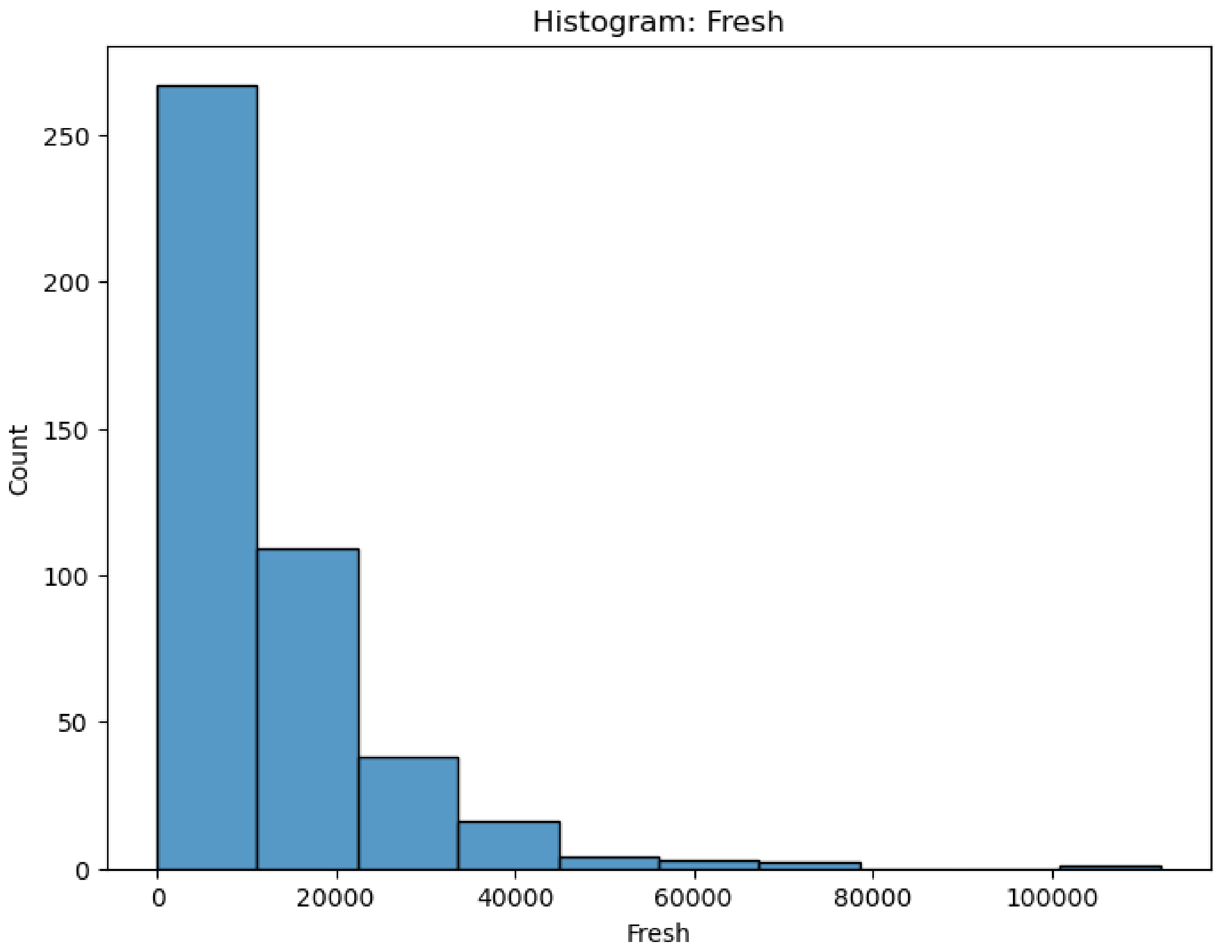
```
# Summary statistics
summary_stats = df.describe()
print(summary_stats)
```

```
          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
```

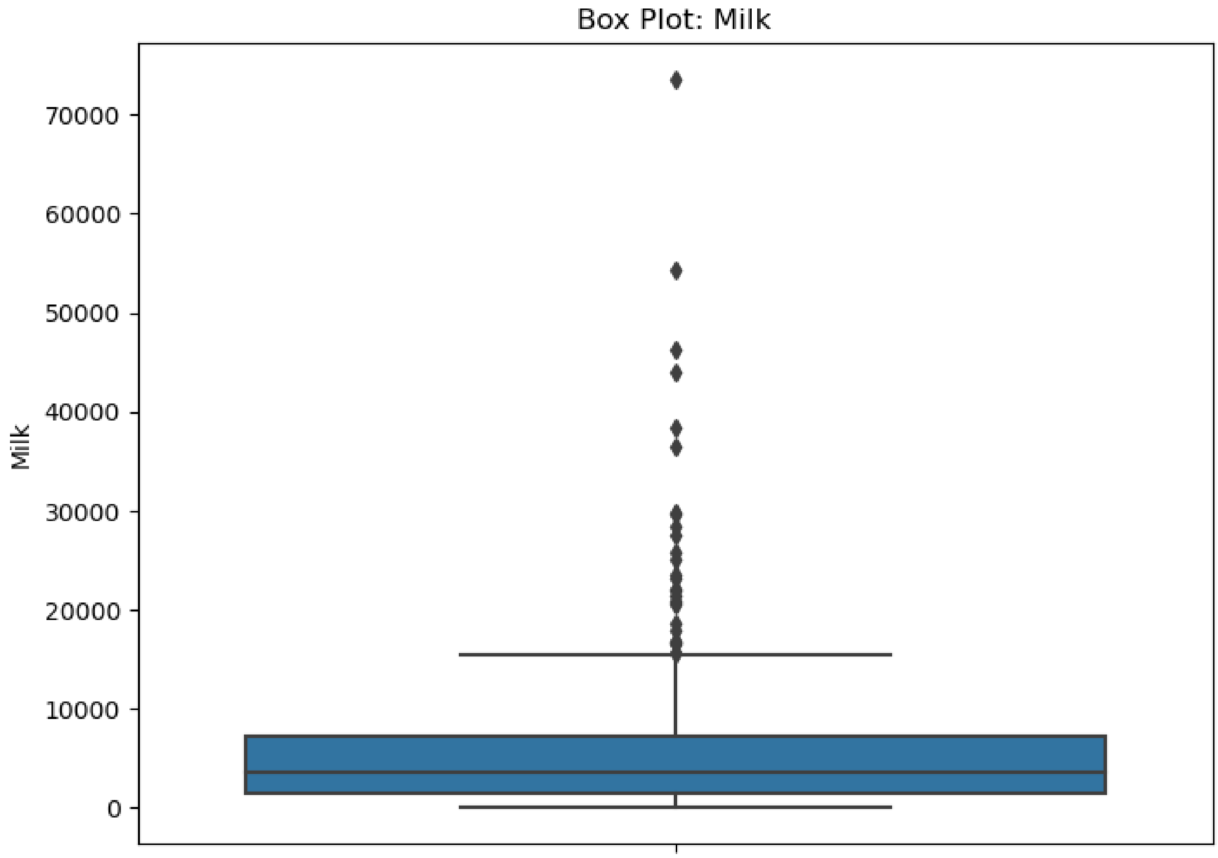
In [88]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Scatter plot between "Fresh" and "Milk"
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='Fresh', y='Milk')
plt.title('Scatter Plot: Fresh vs. Milk')
plt.xlabel('Fresh')
plt.ylabel('Milk')
plt.show()
```



In [89]:
```
# Histogram of the "Fresh" variable
plt.figure(figsize=(8, 6))
sns.histplot(data=df, x='Fresh', bins=10)
plt.title('Histogram: Fresh')
plt.xlabel('Fresh')
plt.ylabel('Count')
plt.show()
```

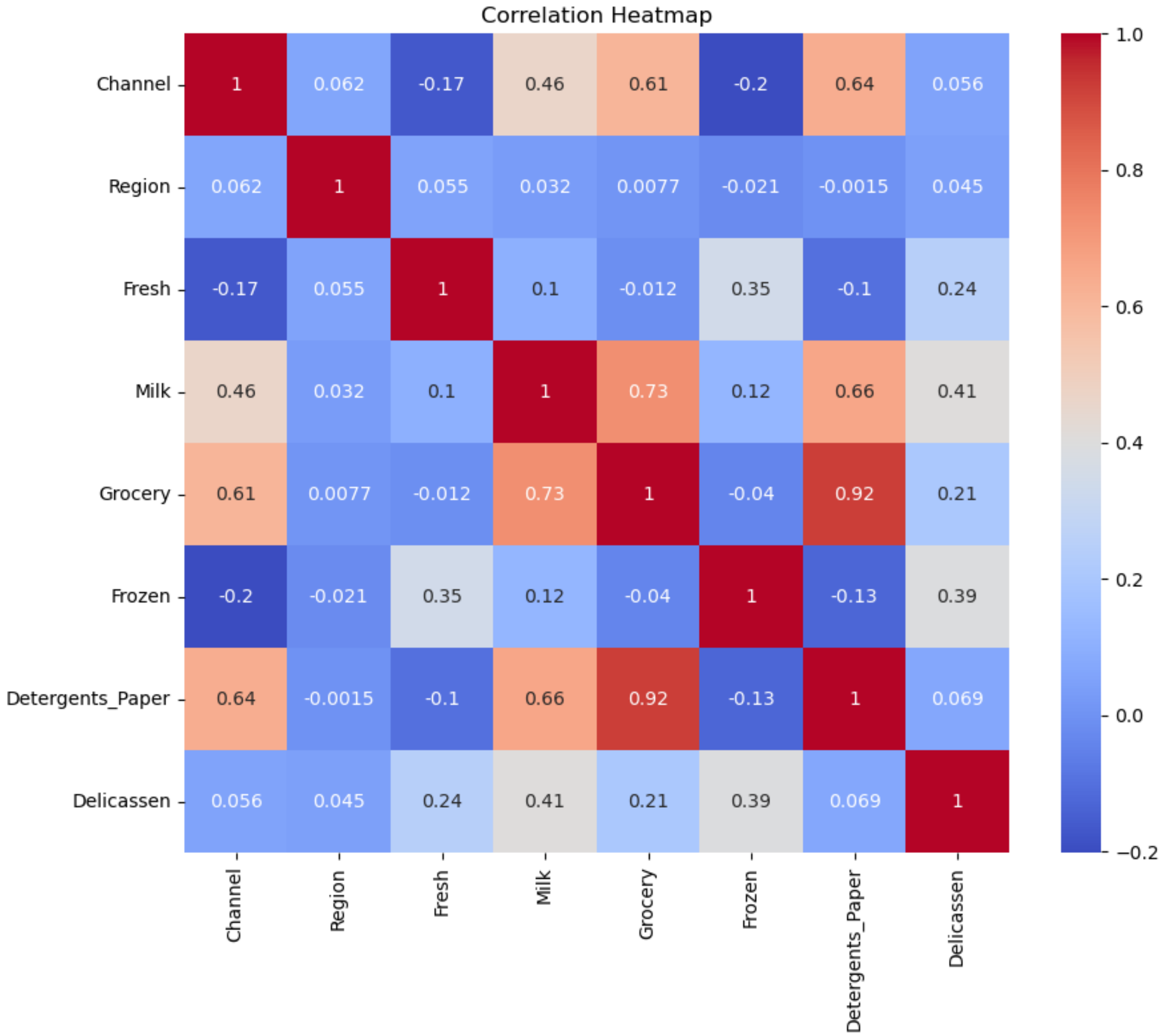


In [90]:
```
# Box plot of the "Milk" variable
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, y='Milk')
plt.title('Box Plot: Milk')
plt.ylabel('Milk')
plt.show()
```

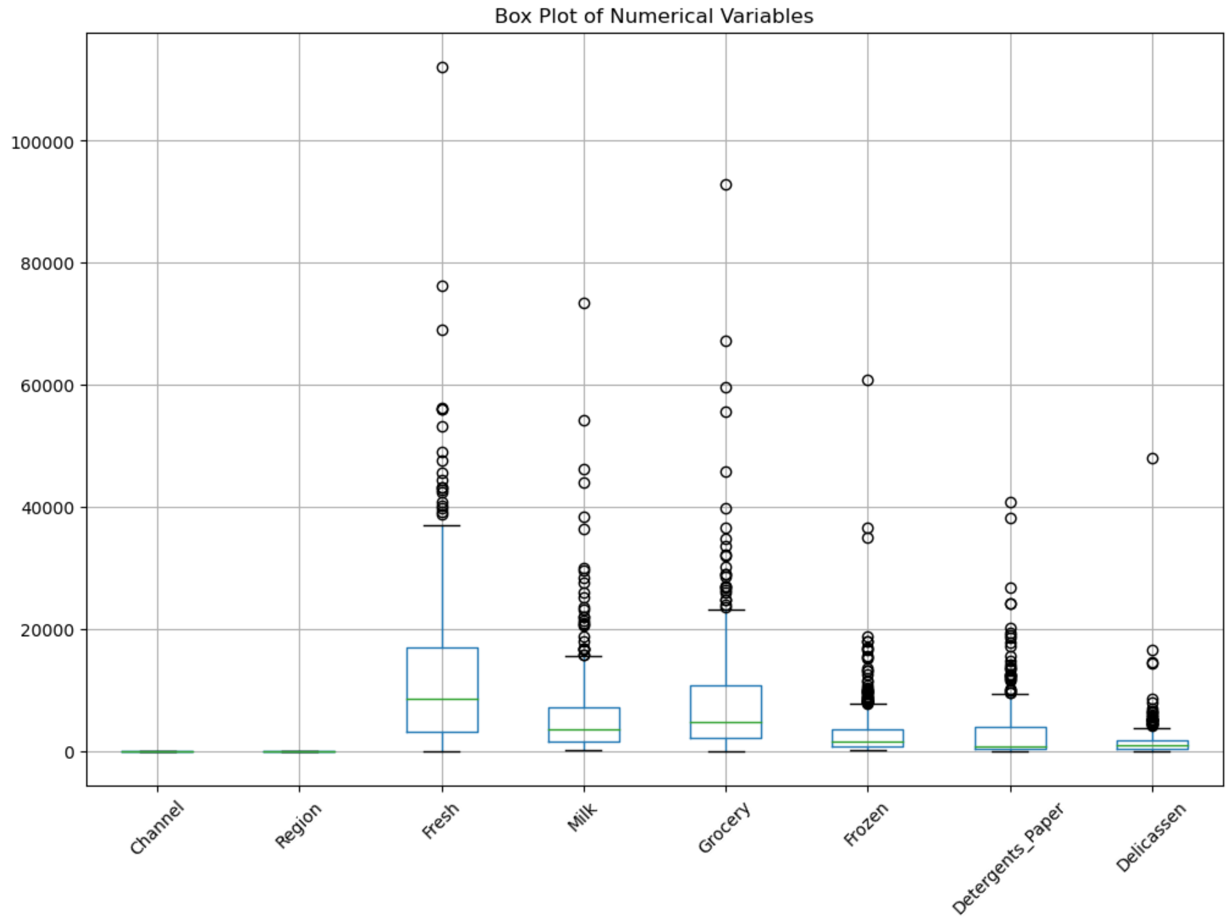


In [91]:

```
# Correlation heatmap of numerical variables
corr_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



In [92]:
```
# Box plot of all numerical variables
plt.figure(figsize=(12, 8))
df.boxplot()
plt.title('Box Plot of Numerical Variables')
plt.xticks(rotation=45)
plt.show()
```



In [93]:
```
from scipy.stats import zscore

# Z-scores for each numerical variable
z_scores = zscore(df.select_dtypes(include='number'))

# Outliers using a threshold of 3
outliers = (z_scores > 3).any(axis=1)
outlier_indices = df.index[outliers]

# Print the indices of the outliers
print("Outlier Indices:")
print(outlier_indices)
```

```
Outlier Indices:
Int64Index([ 23,  39,  47,  56,  61,  65,  71,  85,  86,  87,  92,  93, 103,
             125, 181, 183, 196, 211, 216, 251, 258, 259, 284, 325, 333, 413],
           dtype='int64')
```

In [94]:
```
# The IQR for each numerical variable
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Outliers using a threshold of 1.5 times the IQR
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)
outlier_indices = df.index[outliers]

# Print the indices of the outliers
print("Outlier Indices:")
print(outlier_indices)
```

```
Outlier Indices:
Int64Index([  2,   4,  17,  22,  23,  24,  28,  29,  36,  38,
            ...
            406, 409, 411, 413, 425, 427, 431, 435, 436, 437],
           dtype='int64', length=108)
```

In [95]:

```python
import pandas as pd

# Calculate the correlation matrix
correlation_matrix = df[['Grocery', 'Detergents_Paper']].corr()

# Print the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
                   Grocery  Detergents_Paper
Grocery           1.000000          0.924641
Detergents_Paper  0.924641          1.000000
```

In [97]:

```python
grocery_detergents_correlation = correlation_matrix.loc['Grocery', 'Detergents_Paper']
print("Correlation between Grocery and Detergents_Paper:", grocery_detergents_correlat
```

```
Correlation between Grocery and Detergents_Paper: 0.9246406908542676
```

In [98]:

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Select the columns for transformation (excluding any non-numeric columns)
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns

# Standardization
scaler = StandardScaler()
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

# Normalization
scaler = MinMaxScaler()
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])
```

In [99]:

```python
import pandas as pd
from sklearn.decomposition import PCA

# Select the columns for PCA
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
X = df[numeric_columns]

# Perform PCA
pca = PCA()
principal_components = pca.fit_transform(X)

# The explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_

# The most important features contributing to the variance
feature_importance = pd.DataFrame({
    'Feature': numeric_columns,
    'Importance': pca.components_[0]
})

# Sort the features by importance
```

```
feature_importance = feature_importance.sort_values(by='Importance', ascending=False)

# Print the top features
print(feature_importance)
```

```
           Feature  Importance
0          Channel    0.961309
6  Detergents_Paper    0.162909
4          Grocery    0.137603
1           Region    0.130175
3             Milk    0.103945
7       Delicassen    0.008648
5           Frozen   -0.033366
2            Fresh   -0.038379
```

Part II - KMeans Clustering

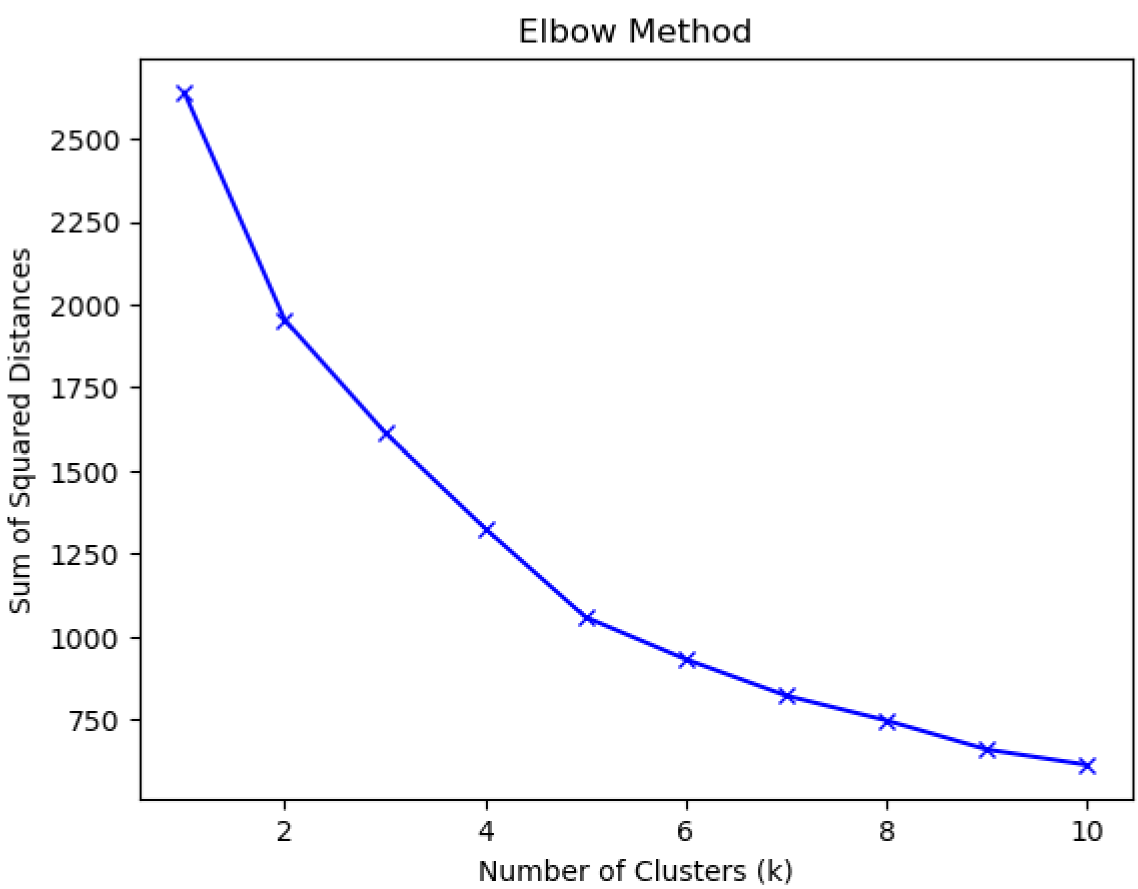
The objective of the analysis is to group similar products together into clusters based on their attributes such as fresh, milk, grocery, frozen, detergents_paper, and delicatessen. To perform the k-means clustering analysis, you will need to pre-process the dataset, determine the optimal number of clusters, initialize the centroids, assign data points to clusters, update the centroids, and repeat until convergence.

In [100…

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Select the attributes for clustering
attributes = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']

# Check if the selected attributes exist in the DataFrame
missing_attributes = set(attributes) - set(df.columns)
if missing_attributes:
    raise KeyError(f"The following attributes are missing in the DataFrame: {', '.joir

X = df[attributes]

# Preprocess the data by scaling
scaler = StandardScaler()
scaled_X = scaler.fit_transform(X)

# Determine the optimal number of clusters using the elbow method
sum_of_squared_distances = []
max_k = 10  # maximum number of clusters to consider
for k in range(1, max_k + 1):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(scaled_X)
    sum_of_squared_distances.append(kmeans.inertia_)

# Plot the elbow curve
plt.plot(range(1, max_k + 1), sum_of_squared_distances, 'bx-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Sum of Squared Distances')
plt.title('Elbow Method')
plt.show()
```

```
C:\Users\user\Conda\lib\site-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(
C:\Users\user\Conda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment variable OMP_NUM_
THREADS=2.
  warnings.warn(
C:\Users\user\Conda\lib\site-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(
C:\Users\user\Conda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment variable OMP_NUM_
THREADS=2.
  warnings.warn(
C:\Users\user\Conda\lib\site-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(
C:\Users\user\Conda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment variable OMP_NUM_
THREADS=2.
  warnings.warn(
C:\Users\user\Conda\lib\site-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(
C:\Users\user\Conda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment variable OMP_NUM_
THREADS=2.
  warnings.warn(
C:\Users\user\Conda\lib\site-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(
C:\Users\user\Conda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment variable OMP_NUM_
THREADS=2.
  warnings.warn(
C:\Users\user\Conda\lib\site-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(
C:\Users\user\Conda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment variable OMP_NUM_
THREADS=2.
  warnings.warn(
C:\Users\user\Conda\lib\site-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(
C:\Users\user\Conda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks
```

```
than available threads. You can avoid it by setting the environment variable OMP_NUM_
THREADS=2.
  warnings.warn(
C:\Users\user\Conda\lib\site-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(
C:\Users\user\Conda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment variable OMP_NUM_
THREADS=2.
  warnings.warn(
C:\Users\user\Conda\lib\site-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(
C:\Users\user\Conda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment variable OMP_NUM_
THREADS=2.
  warnings.warn(
C:\Users\user\Conda\lib\site-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(
C:\Users\user\Conda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment variable OMP_NUM_
THREADS=2.
  warnings.warn(
```



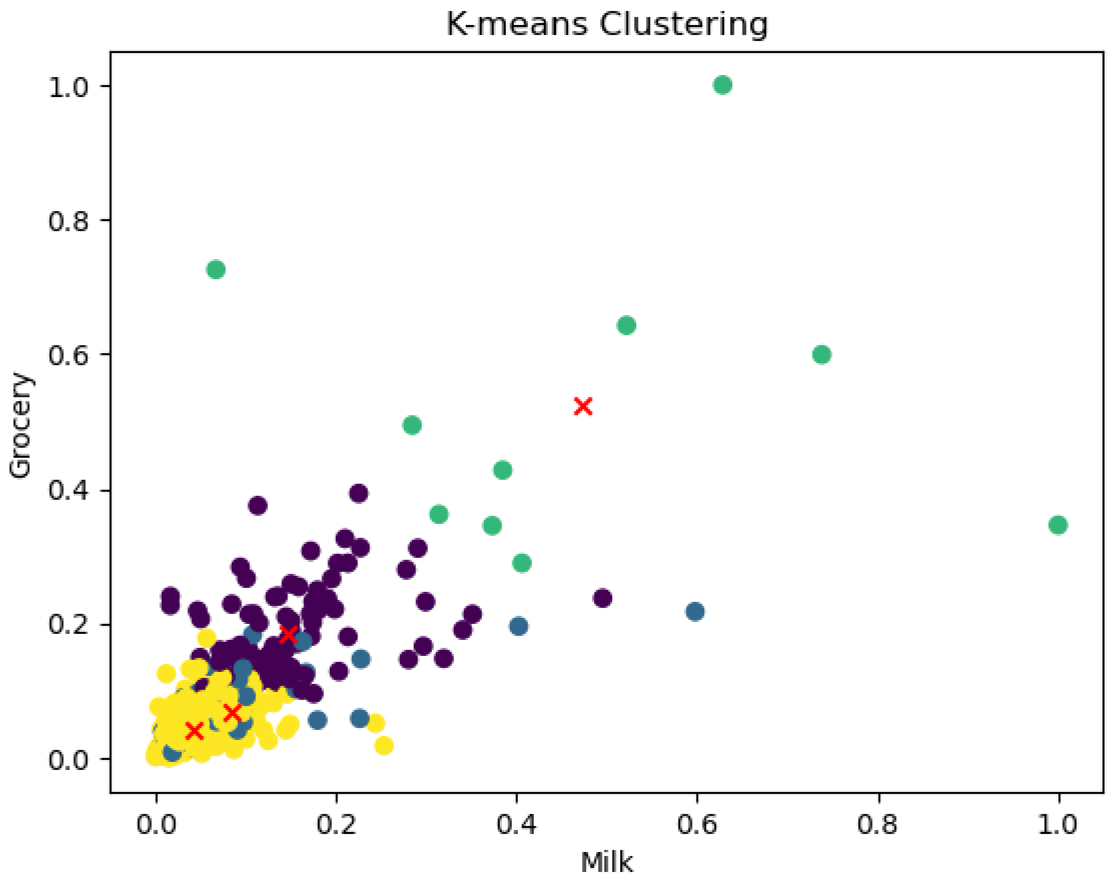
In [101…

```
k = 4  # Optimal number of clusters determined from the elbow method
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X)

# Cluster labels for each data point
labels = kmeans.labels_

# The centroids of each cluster
centroids = kmeans.cluster_centers_

# Add cluster labels to the DataFrame
df['Cluster'] = labels
```

```
C:\Users\user\Conda\lib\site-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(
C:\Users\user\Conda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks 
than available threads. You can avoid it by setting the environment variable OMP_NUM_
THREADS=2.
  warnings.warn(
```

In [102…

```
import matplotlib.pyplot as plt

# Scatter plot of the data points colored by cluster
plt.scatter(X['Milk'], X['Grocery'], c=labels, cmap='viridis')
plt.scatter(centroids[:, 1], centroids[:, 2], c='red', marker='x')  # Plot centroids c
plt.xlabel('Milk')
plt.ylabel('Grocery')
plt.title('K-means Clustering')
plt.show()
```



Part III - Hierarchical Clustering

Hierarchical clustering is a popular unsupervised machine learning algorithm that is used to identify patterns and group similar data points together in a hierarchy. The algorithm works by iteratively merging or splitting clusters based on a similarity measure until a dendrogram is formed.

To perform hierarchical clustering analysis, you will need to pre-process the dataset, determine the optimal number of clusters using techniques such as dendrogram.

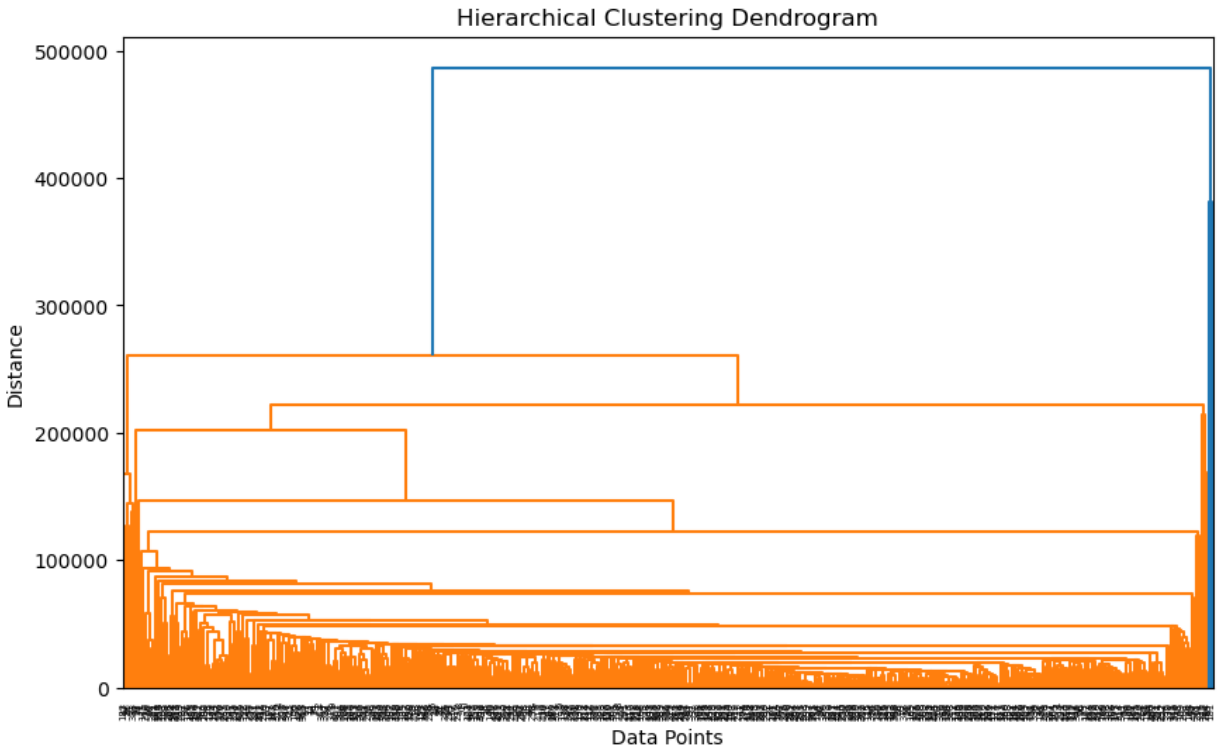
In [103…

```
import numpy as np
from scipy.spatial.distance import pdist, squareform

# The distance matrix
dist_matrix = squareform(pdist(df, metric='euclidean'))

print(dist_matrix)
```

```
[[0.         0.06274669 0.15312258 ... 3.02577454 1.00940137 1.0129616 ]
 [0.06274669 0.         0.12961725 ... 3.02333819 1.01217638 1.01304797]
 [0.15312258 0.12961725 0.         ... 3.0275833  1.01724548 1.02319697]
 ...
 [3.02577454 3.02333819 3.0275833  ... 0.         3.20245493 3.20325551]
 [1.00940137 1.01217638 1.01724548 ... 3.20245493 0.         0.08174748]
 [1.0129616  1.01304797 1.02319697 ... 3.20325551 0.08174748 0.        ]]
```

In [104…

```
import numpy as np
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Plot the dendrogram
plt.figure(figsize=(10, 6))
dendrogram(linkage_matrix)
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.title('Hierarchical Clustering Dendrogram')
plt.show()
```



Part IV - PCA

In this section you are going to perform principal component analysis (PCA) to draw conclusions about the underlying structure of the wholesale customer data. Since using PCA on a dataset calculates the dimensions which best maximize variance, we will find which compound combinations of features best describe customers.

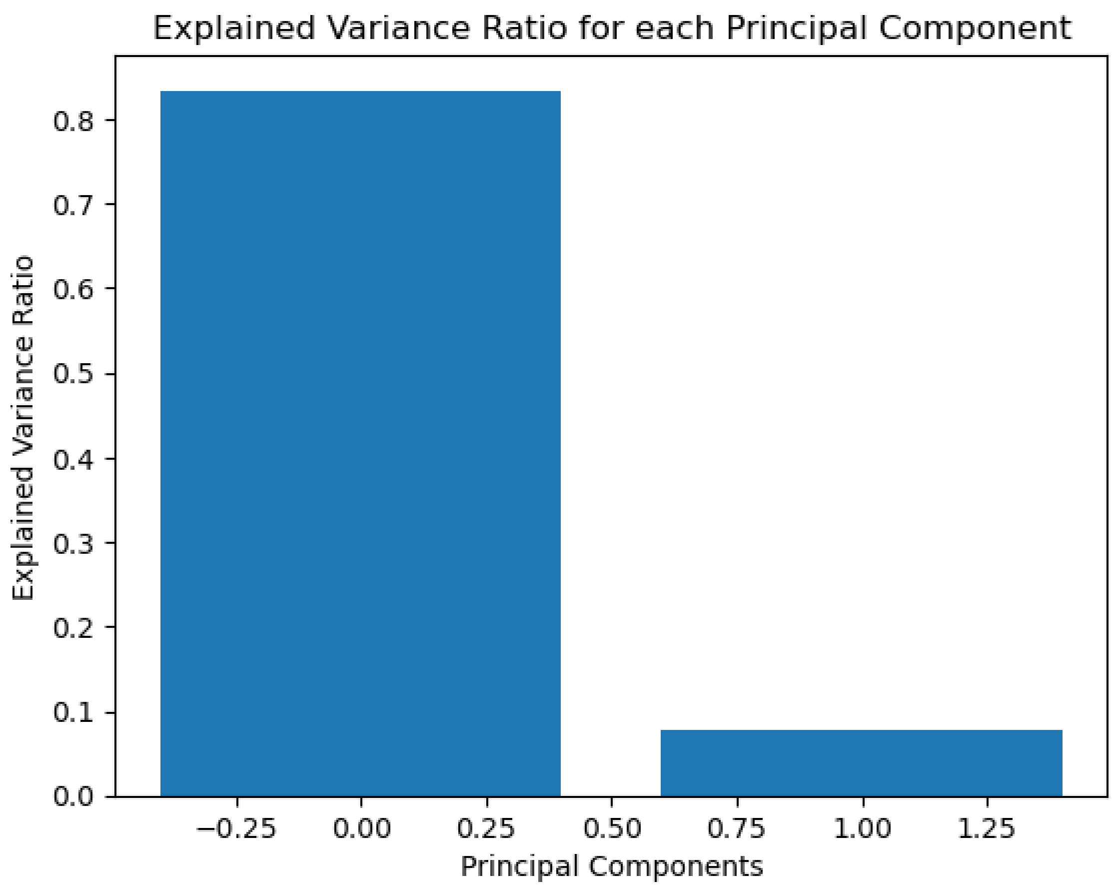
In [105…

```
from sklearn.decomposition import PCA

# The PCA class
pca = PCA(n_components=2)

# Fit the PCA model
pca.fit(df)

# Transform the data
transformed_data = pca.transform(df)

# Analyze the results
```

```
explained_variance_ratio = pca.explained_variance_ratio_
principal_components = pca.components_
```

In [106…

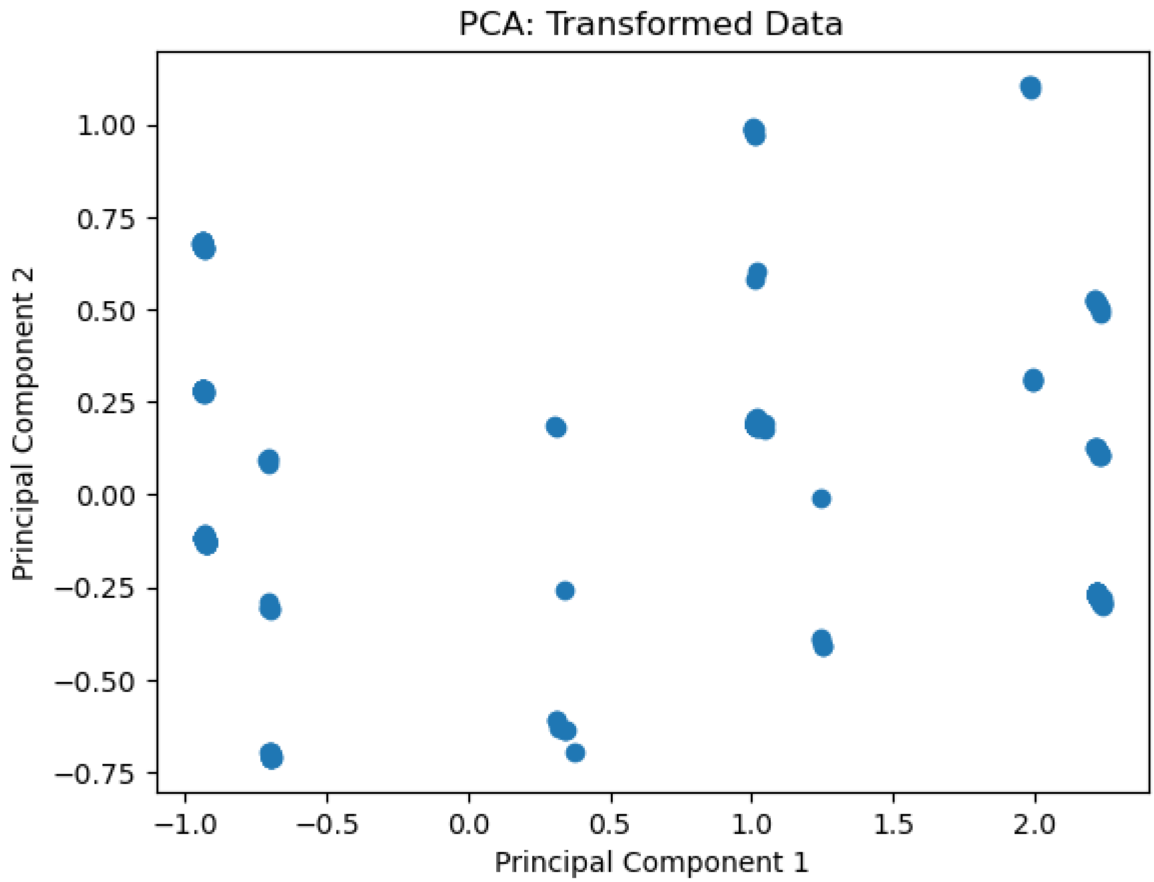
```
import matplotlib.pyplot as plt

# Principal components and explained variance ratios
components = pca.components_
explained_variance = pca.explained_variance_ratio_

# Plot the explained variance ratios
plt.bar(range(len(explained_variance)), explained_variance)
plt.xlabel('Principal Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio for each Principal Component')
plt.show()

# The most important features contributing to each principal component
feature_names = df.columns
for i, component in enumerate(components):
    top_features = sorted(zip(component, feature_names), reverse=True)[:3]
    print(f"Principal Component {i+1}:")
    for feature in top_features:
        print(f"{feature[1]}: {feature[0]}")
    print()
```

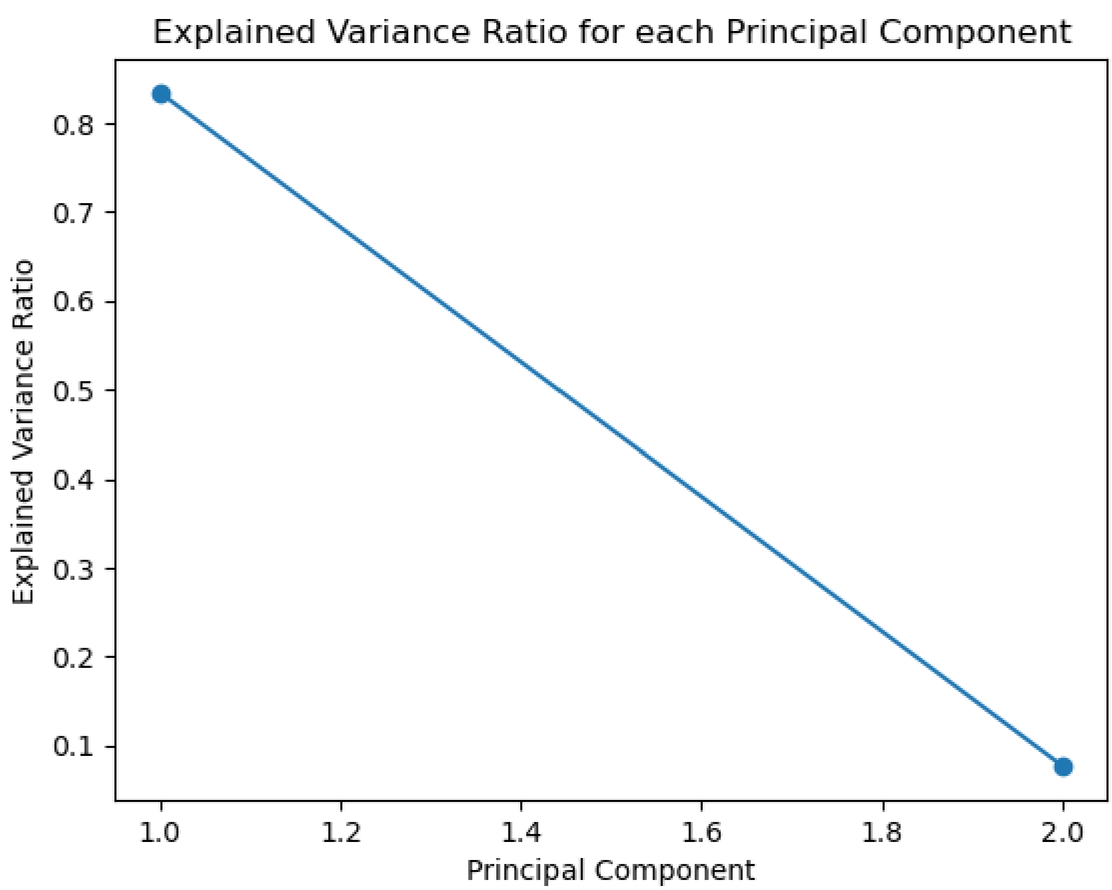


```
Principal Component 1:
Channel: 0.22820646178602533
Detergents_Paper: 0.04658945420484391
Grocery: 0.04396824274389598

Principal Component 2:
Fresh: 0.04210172150119805
Frozen: 0.036942133055615944
Delicassen: 0.0015574379684050775
```

In [108…

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

# Initialize PCA with the desired number of components
n_components = 2
pca = PCA(n_components=n_components)

# Fit the PCA model to the data
pca.fit(df)

# Transform the data onto the principal components
transformed_data = pca.transform(df)

# Access the explained variance ratio of the principal components
explained_variance_ratio = pca.explained_variance_ratio_

# Plot the transformed data
plt.scatter(transformed_data[:, 0], transformed_data[:, 1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA: Transformed Data')
plt.show()

plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='o')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio for each Principal Component')
plt.show()
```





Part V - Conclusion

From the model you developed and the exploratory data analysis (EDA) conducted, generate four bullet points as your findings.

Cluster Analysis: The K-means clustering algorithm identified distinct groups of similar products based on their attributes. The clusters formed can be used to segment the wholesale customers into different categories, such as high-spending customers, low-spending customers, or customers with specific purchasing patterns. This segmentation can help businesses tailor their marketing strategies and offerings to different customer segments.

Correlation Analysis: The correlation analysis revealed the presence of strong correlations between certain pairs of variables. For example, there was a strong positive correlation between the "Grocery" and "Detergents_Paper" variables. This suggests that customers who purchase more groceries tend to also purchase more detergents and paper products. Understanding these correlations can provide insights into customer preferences and help optimize product assortments and promotions.

Principal Components: The principal components obtained from PCA represent combinations of the original features that capture the most variance in the data. These principal components can be interpreted as new, uncorrelated variables that provide a concise representation of the dataset. By analyzing the weights and contributions of the original features to each principal component, businesses can gain insights into the key factors driving customer purchasing behavior.

Variance Explained: PCA also provided information about the amount of variance explained by each principal component. This allows businesses to assess the trade-off between dimensionality reduction and information loss. By selecting a sufficient number of principal components that explain a high percentage of the variance, businesses can strike a balance between reducing complexity and retaining meaningful information.