Experiment No. 5
Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset
Date of Performance: 21-08-2023
Date of Submission: 08-10-2023

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

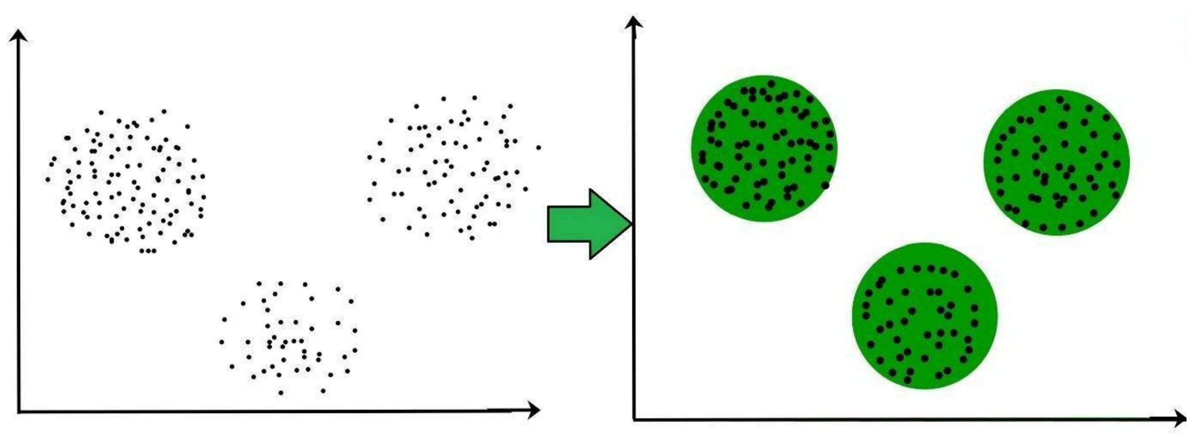
Objective: Able to perform various feature engineering tasks, apply Clustering Algorithm on the given dataset.

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)

ml-experiment-5

October 8, 2023

[1]:
```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
from IPython.display import display # Allows the use of display() for DataFrames
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

import seaborn as sns
import matplotlib.pyplot as plt
# Pretty display for notebooks
%matplotlib inline

# Load the wholesale customers dataset
try:
    data = pd.read_csv("customers.csv")
    data.drop(['Region', 'Channel'], axis = 1, inplace = True)
    print("Wholesale customers dataset has {} samples with {} features each.".
  ↪format(*data.shape))
except:
    print("Dataset could not be loaded. Is the dataset missing?")
```

```
Wholesale customers dataset has 440 samples with 6 features each.
```

[2]:
```
###########################################
# Suppress matplotlib user warnings
# Necessary for newer version of matplotlib
import warnings
warnings.filterwarnings("ignore", category = UserWarning, module = "matplotlib")
#
# Display inline matplotlib plots with IPython
from IPython import get_ipython
get_ipython().run_line_magic('matplotlib', 'inline')
###########################################

import matplotlib.pyplot as plt
```

```
import matplotlib.cm as cm
import pandas as pd
import numpy as np

def pca_results(good_data, pca):


        # Dimension indexing
        dimensions = dimensions = ['Dimension {}'.format(i) for i in␣
 ↪range(1,len(pca.components_)+1)]

        # PCA components
        components = pd.DataFrame(np.round(pca.components_, 4), columns =␣
 ↪list(good_data.keys()))
        components.index = dimensions

        # PCA explained variance
        ratios = pca.explained_variance_ratio_.reshape(len(pca.components_), 1)
        variance_ratios = pd.DataFrame(np.round(ratios, 4), columns =␣
 ↪['Explained Variance'])
        variance_ratios.index = dimensions

        # Create a bar plot visualization
        fig, ax = plt.subplots(figsize = (14,8))

        # Plot the feature weights as a function of the components
        components.plot(ax = ax, kind = 'bar');
        ax.set_ylabel("Feature Weights")
        ax.set_xticklabels(dimensions, rotation=0)


        # Display the explained variance ratios
        for i, ev in enumerate(pca.explained_variance_ratio_):
                ax.text(i-0.40, ax.get_ylim()[1] + 0.05, "Explained Variance\n ␣
 ↪        %.4f"%(ev))

        # Return a concatenated DataFrame
        return pd.concat([variance_ratios, components], axis = 1)

def cluster_results(reduced_data, preds, centers, pca_samples):


        predictions = pd.DataFrame(preds, columns = ['Cluster'])
        plot_data = pd.concat([predictions, reduced_data], axis = 1)

        # Generate the cluster plot
        fig, ax = plt.subplots(figsize = (14,8))
```

```
        # Color map
        cmap = cm.get_cmap('gist_rainbow')

        # Color the points based on assigned cluster
        for i, cluster in plot_data.groupby('Cluster'):
            cluster.plot(ax = ax, kind = 'scatter', x = 'Dimension 1', y =␣
 ↪'Dimension 2', \
                         color = cmap((i)*1.0/(len(centers)-1)), label =␣
 ↪'Cluster %i'%(i), s=30);

        # Plot centers with indicators
        for i, c in enumerate(centers):
            ax.scatter(x = c[0], y = c[1], color = 'white', edgecolors =␣
 ↪'black', \
                       alpha = 1, linewidth = 2, marker = 'o', s=200);
            ax.scatter(x = c[0], y = c[1], marker='$%d$'%(i), alpha = 1, s=100);

        # Plot transformed sample points
        ax.scatter(x = pca_samples[:,0], y = pca_samples[:,1], \
                   s = 150, linewidth = 4, color = 'black', marker = 'x');

        # Set plot title
        ax.set_title("Cluster Learning on PCA-Reduced Data - Centroids Marked␣
 ↪by Number\nTransformed Sample Data Marked by Black Cross");


def biplot(good_data, reduced_data, pca):

    fig, ax = plt.subplots(figsize = (14,8))
    # scatterplot of the reduced data
    ax.scatter(x=reduced_data.loc[:, 'Dimension 1'], y=reduced_data.loc[:,␣
 ↪'Dimension 2'],
        facecolors='b', edgecolors='b', s=70, alpha=0.5)

    feature_vectors = pca.components_.T

    # we use scaling factors to make the arrows easier to see
    arrow_size, text_pos = 7.0, 8.0,

    # projections of the original features
    for i, v in enumerate(feature_vectors):
        ax.arrow(0, 0, arrow_size*v[0], arrow_size*v[1],
                  head_width=0.2, head_length=0.2, linewidth=2, color='red')
        ax.text(v[0]*text_pos, v[1]*text_pos, good_data.columns[i],␣
 ↪color='black',
```

```
                 ha='center', va='center', fontsize=18)

    ax.set_xlabel("Dimension 1", fontsize=14)
    ax.set_ylabel("Dimension 2", fontsize=14)
    ax.set_title("PC plane with original feature projections.", fontsize=16);
    return ax


def channel_results(reduced_data, outliers, pca_samples):


        # Check that the dataset is loadable
        try:
            full_data = pd.read_csv("../input/customers.csv")
        except:
            print("Dataset could not be loaded. Is the file missing?")
            return False

        # Create the Channel DataFrame
        channel = pd.DataFrame(full_data['Channel'], columns = ['Channel'])
        channel = channel.drop(channel.index[outliers]).reset_index(drop = True)
        labeled = pd.concat([reduced_data, channel], axis = 1)

        # Generate the cluster plot
        fig, ax = plt.subplots(figsize = (14,8))

        # Color map
        cmap = cm.get_cmap('gist_rainbow')

        # Color the points based on assigned Channel
        labels = ['Hotel/Restaurant/Cafe', 'Retailer']
        grouped = labeled.groupby('Channel')
        for i, channel in grouped:
            channel.plot(ax = ax, kind = 'scatter', x = 'Dimension 1', y =␣
↪'Dimension 2', \
                         color = cmap((i-1)*1.0/2), label = labels[i-1], s=30);

        # Plot transformed sample points
        for i, sample in enumerate(pca_samples):
                ax.scatter(x = sample[0], y = sample[1], \
                   s = 200, linewidth = 3, color = 'black', marker = 'o',␣
↪facecolors = 'none');
                ax.scatter(x = sample[0]+0.25, y = sample[1]+0.3,␣
↪marker='$%d$'%(i), alpha = 1, s=125);

        # Set plot title
```

```
        ax.set_title("PCA-Reduced Data Labeled by 'Channel'\nTransformed Sample␣
↪Data Circled");
```

[3]:
```
# Print some sample data
display(data.head())
```

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

Data Exploration

[4]:
```
# Display Data Info
display(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Fresh             440 non-null    int64
 1   Milk              440 non-null    int64
 2   Grocery           440 non-null    int64
 3   Frozen            440 non-null    int64
 4   Detergents_Paper  440 non-null    int64
 5   Delicatessen      440 non-null    int64
dtypes: int64(6)
memory usage: 20.8 KB
```

```
None
```

[5]:
```
# Display a description of the dataset
display(data.describe())
```

```
               Fresh          Milk       Grocery        Frozen  \
count     440.000000    440.000000    440.000000    440.000000
mean    12000.297727   5796.265909   7951.277273   3071.931818
std     12647.328865   7380.377175   9503.162829   4854.673333
min         3.000000     55.000000      3.000000     25.000000
25%      3127.750000   1533.000000   2153.000000    742.250000
50%      8504.000000   3627.000000   4755.500000   1526.000000
75%     16933.750000   7190.250000  10655.750000   3554.250000
max    112151.000000  73498.000000  92780.000000  60869.000000

       Detergents_Paper  Delicatessen
count        440.000000    440.000000
```

```
mean        2881.493182   1524.870455
std         4767.854448   2820.105937
min            3.000000      3.000000
25%          256.750000    408.250000
50%          816.500000    965.500000
75%         3922.000000   1820.250000
max        40827.000000  47943.000000
```

[6]:

```
# Select three indices of your choice you wish to sample from the dataset
np.random.seed(2018)
indices = np.random.randint(low = 0, high = 441, size = 3)
print("Indices of Samples => {}".format(indices))

# Create a DataFrame of the chosen samples
samples = pd.DataFrame(data.loc[indices], columns = data.keys()).
 ↪reset_index(drop = True)
print("\nChosen samples of wholesale customers dataset:")
display(samples)
```

```
Indices of Samples => [250 102 226]

Chosen samples of wholesale customers dataset:
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	3191	1993	1799	1730	234	710
1	2932	6459	7677	2561	4573	1386
2	20782	5921	9212	1759	2568	1553

[7]:

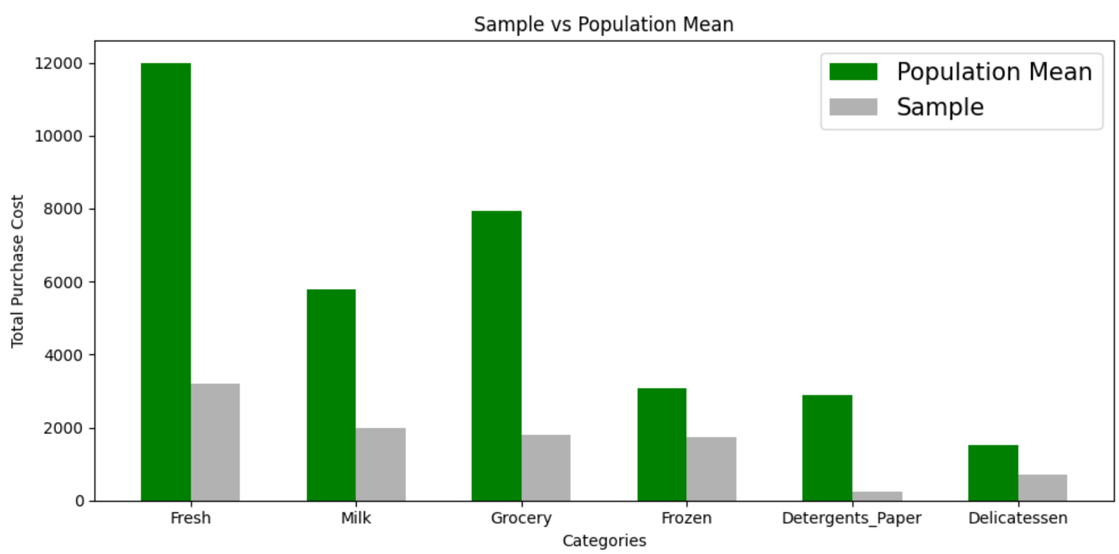
```
# Function to display the sample data vs the population mean for
# each of the categories
def sampl_pop_plotting(sample):
    fig, ax = plt.subplots(figsize=(10,5))

    index = np.arange(sample.count())
    bar_width = 0.3
    opacity_pop = 1
    opacity_sample = 0.3

    rect1 = ax.bar(index, data.mean(), bar_width,
                    alpha=opacity_pop, color='g',
                    label='Population Mean')

    rect2 = ax.bar(index + bar_width, sample, bar_width,
                    alpha=opacity_sample, color='k',
                    label='Sample')

    ax.set_xlabel('Categories')
    ax.set_ylabel('Total Purchase Cost')
```

```
    ax.set_title('Sample vs Population Mean')
    ax.set_xticks(index + bar_width / 2)
    ax.set_xticklabels(samples.columns)
    ax.legend(loc=0, prop={'size': 15})

    fig.tight_layout()
    plt.show()
```

[8]:
```
# Display data for the first sample wrt to the population mean
display(samples.iloc[0] - data.mean())
```

```
Fresh              -8809.297727
Milk               -3803.265909
Grocery            -6152.277273
Frozen             -1341.931818
Detergents_Paper   -2647.493182
Delicatessen        -814.870455
dtype: float64
```

[9]:
```
# Plot data for the first sample wrt to the population mean
sampl_pop_plotting(samples.iloc[0])
```

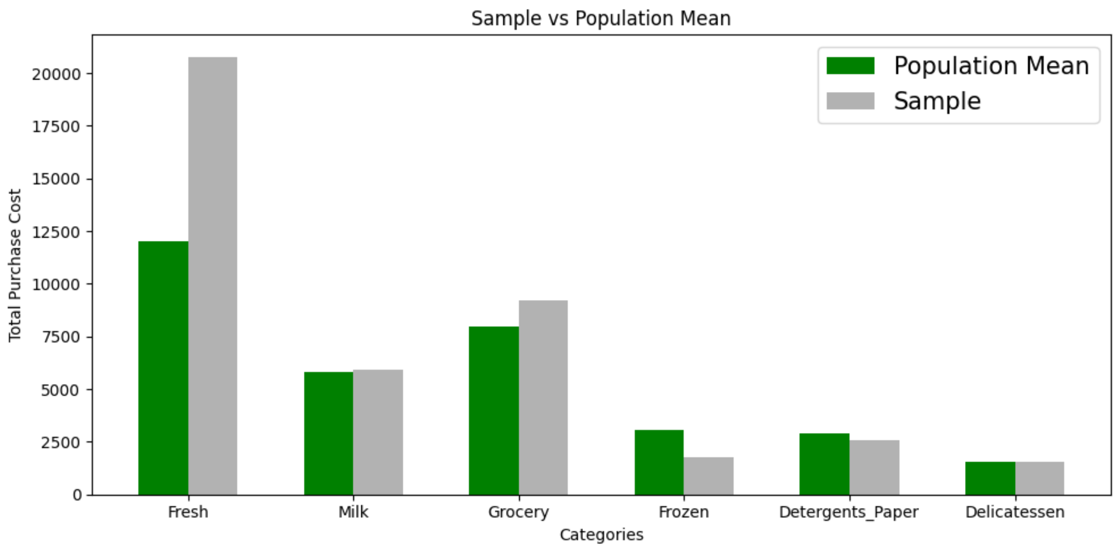


[10]:
```
# Display data for the second sample wrt to the population mean
display(samples.iloc[1] - data.mean())
```

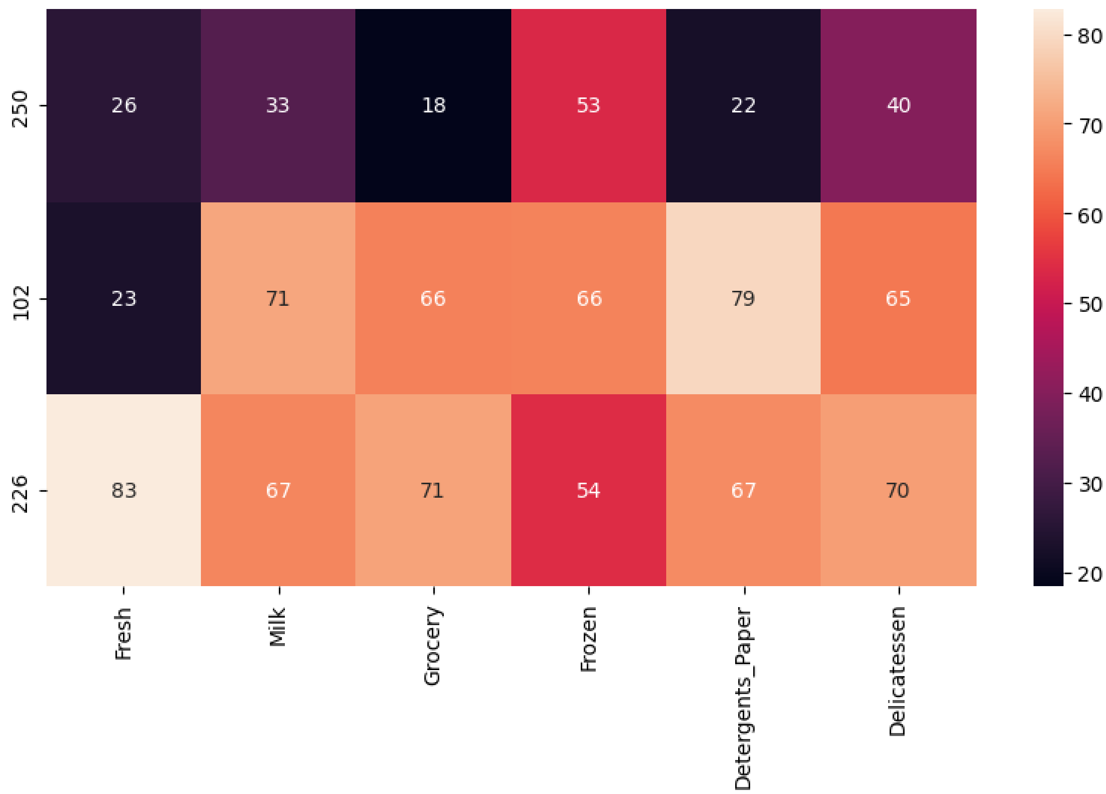
```
Fresh              -9068.297727
Milk                 662.734091
Grocery             -274.277273
Frozen              -510.931818
```

```
Detergents_Paper    1691.506818
Delicatessen        -138.870455
dtype: float64
```

[11]:
```
# Plot data for the third sample wrt to the population mean
sampl_pop_plotting(samples.iloc[2])
```



[12]:
```
# percentile heatmap for sample points
percentiles_data = 100*data.rank(pct=True)
percentiles_samples = percentiles_data.iloc[indices]
plt.subplots(figsize=(10,5))
_ = sns.heatmap(percentiles_samples, annot=True)
```



[13]:
```
def predict_one_feature(dropped_feature):
    # Make a copy of the DataFrame, using the 'drop' function to drop the given␣
 ↪feature
    print("Dropping feature -> {}".format(dropped_feature))
    new_data = data.drop([dropped_feature], axis = 1)

    # Split the data into training and testing sets(0.25) using the given␣
 ↪feature as the target
    # Set a random state.
    X_train, X_test, y_train, y_test = train_test_split(new_data,␣
 ↪data[dropped_feature], test_size=0.25, random_state=0)

    # Create a decision tree regressor and fit it to the training set
    regressor = DecisionTreeRegressor(random_state=0)
    regressor.fit(X_train, y_train)

    # Report the score of the prediction using the testing set
    score = regressor.score(X_test, y_test)
    print("Score for predicting '{}' using other features = {:.3f}\n".
 ↪format(dropped_feature, score))
```

[14]:
```
# Attempt to predict the score of 'Milk' using other features
predict_one_feature('Milk')
```

```
Dropping feature -> Milk
Score for predicting 'Milk' using other features = 0.366
```

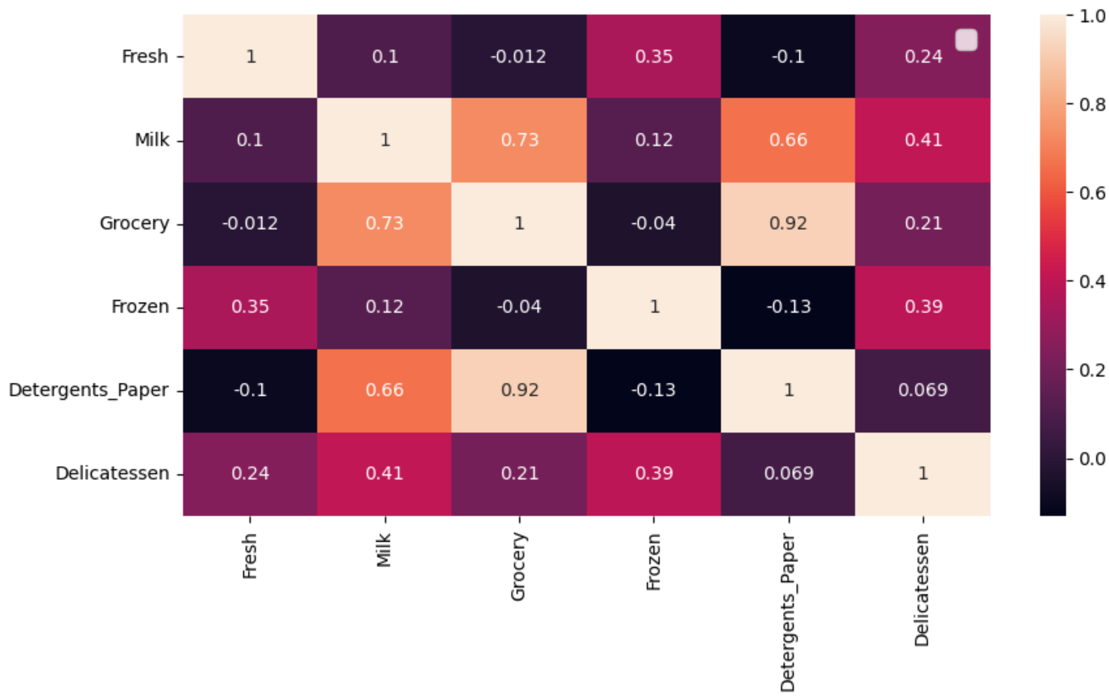
[15]:
```
print("Features in data -> {}\n".format(data.columns.values))

# Predict the score of each feature by dropping it and using other features
for cols in data.columns.values:
    predict_one_feature(cols)
```

```
Features in data -> ['Fresh' 'Milk' 'Grocery' 'Frozen' 'Detergents_Paper'
'Delicatessen']

Dropping feature -> Fresh
Score for predicting 'Fresh' using other features = -0.252

Dropping feature -> Milk
Score for predicting 'Milk' using other features = 0.366

Dropping feature -> Grocery
Score for predicting 'Grocery' using other features = 0.603

Dropping feature -> Frozen
Score for predicting 'Frozen' using other features = 0.254

Dropping feature -> Detergents_Paper
Score for predicting 'Detergents_Paper' using other features = 0.729

Dropping feature -> Delicatessen
Score for predicting 'Delicatessen' using other features = -11.664
```

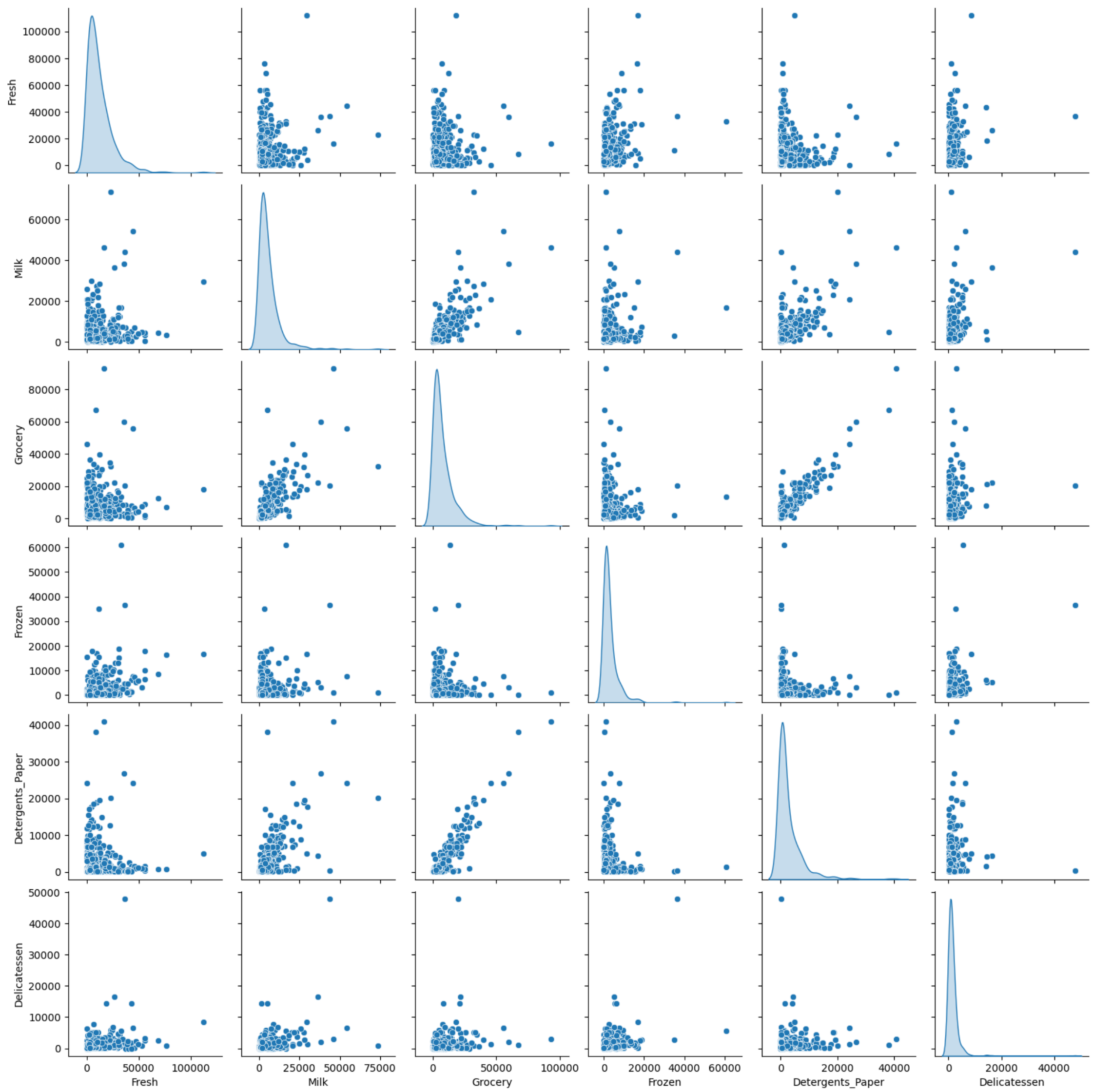
[16]:
```
# Display the correlation heatmap
corr = data.corr()

plt.figure(figsize = (10,5))
ax = sns.heatmap(corr, annot=True)
ax.legend(loc=0, prop={'size': 15})
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
that artists whose label start with an underscore are ignored when legend() is
called with no argument.
```

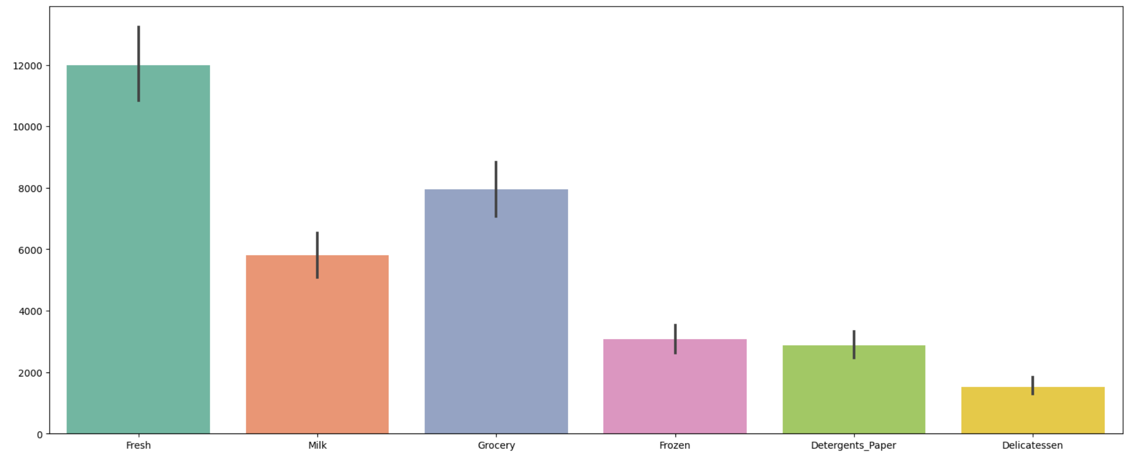
[16]:
```
<matplotlib.legend.Legend at 0x7c917cf22f20>
```



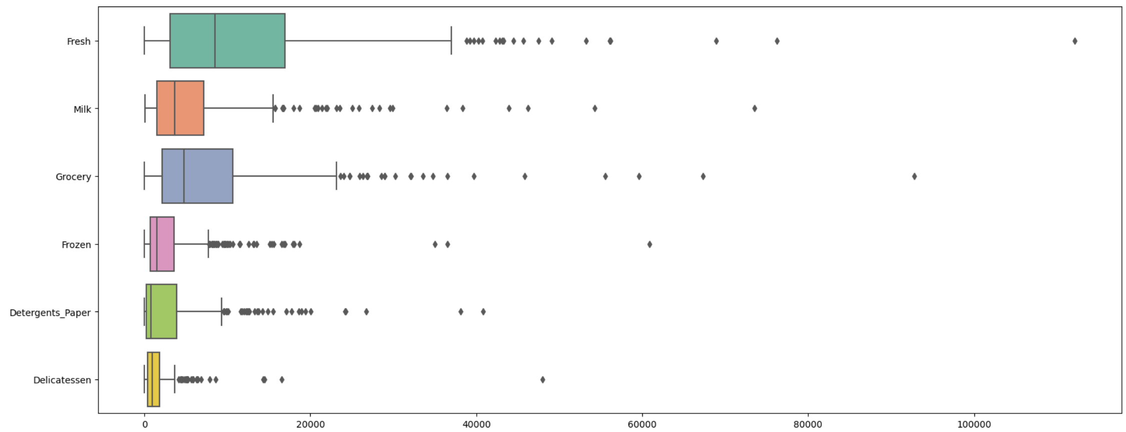
[17]:
```python
# Produce a scatter matrix for each pair of features in the data
_ = sns.pairplot(data, diag_kind = 'kde')
```



[18]:
```
plt.figure(figsize = (20,8))
_ = sns.barplot(data=data, palette="Set2")
```



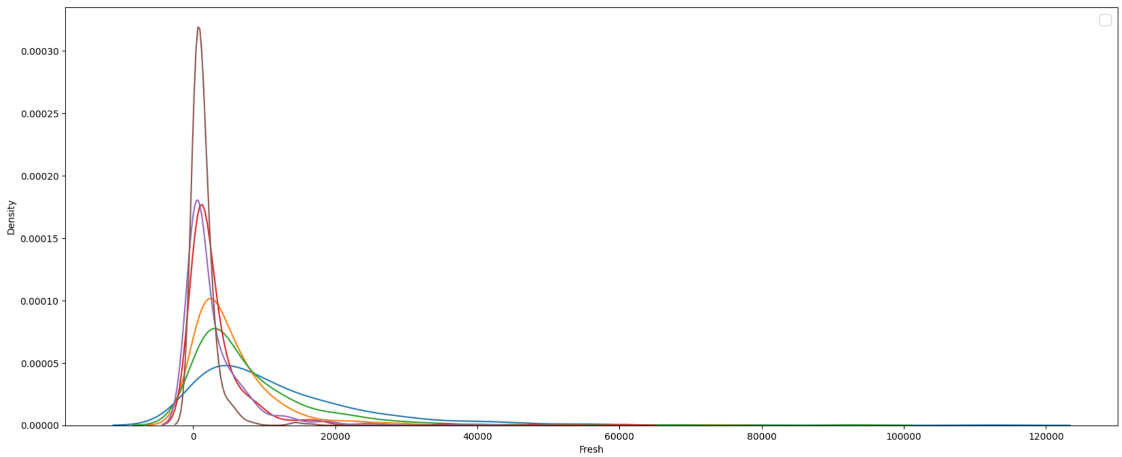
[19]:
```
plt.figure(figsize = (20,8))
_ = sns.boxplot(data=data, orient='h', palette="Set2")
```



[20]:
```
plt.figure(figsize = (20,8))

for cols in data.columns.values:
    ax = sns.kdeplot(data[cols])
    ax.legend(loc=0, prop={'size': 15})
```

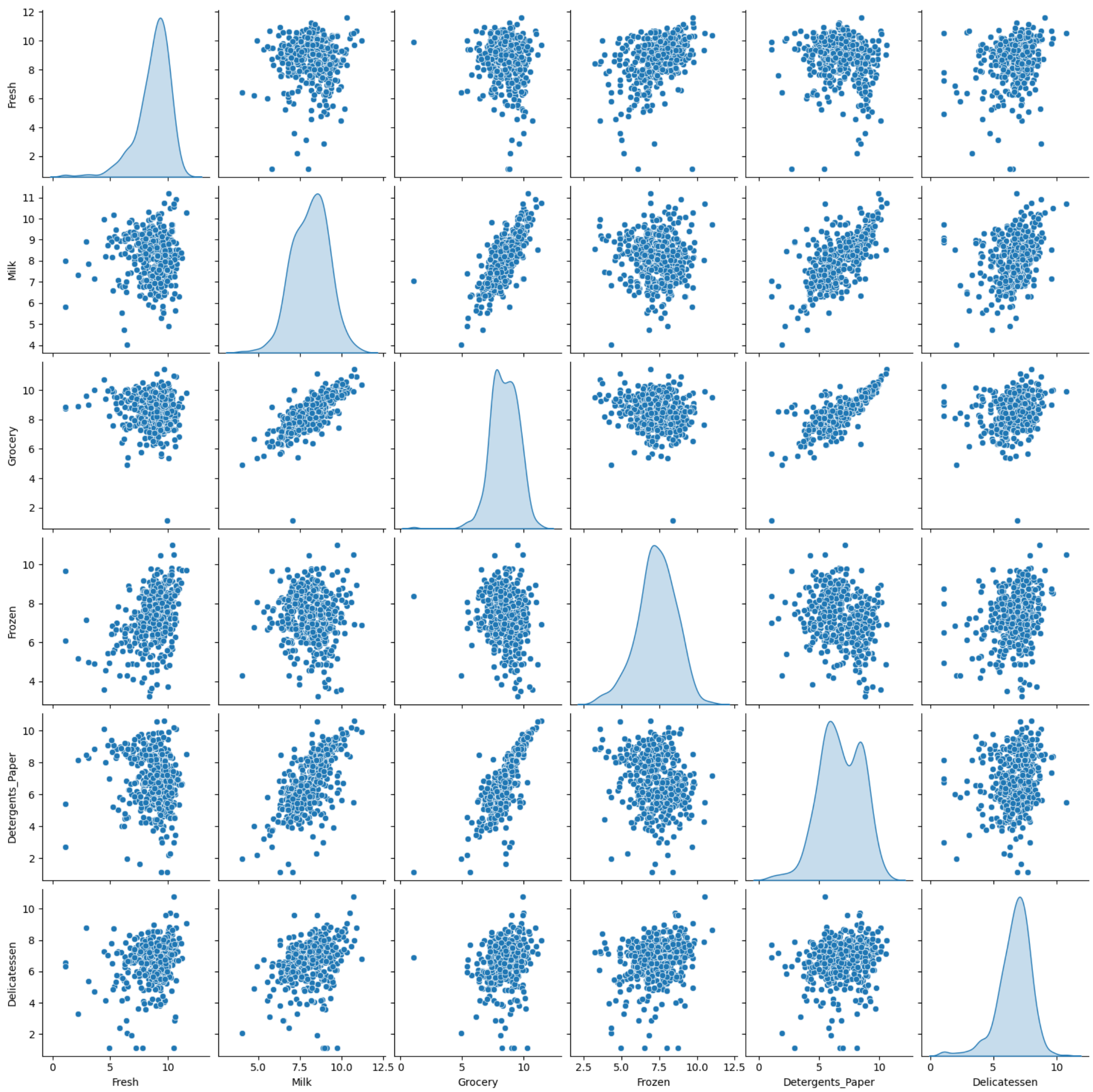
```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
that artists whose label start with an underscore are ignored when legend() is
called with no argument.
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
that artists whose label start with an underscore are ignored when legend() is
called with no argument.
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
that artists whose label start with an underscore are ignored when legend() is
```

```
called with no argument.
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
that artists whose label start with an underscore are ignored when legend() is
called with no argument.
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
that artists whose label start with an underscore are ignored when legend() is
called with no argument.
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
that artists whose label start with an underscore are ignored when legend() is
called with no argument.
```



Data Preprocessing

[21]:
```python
# Scale the data using the natural logarithm
log_data = np.log(data)

# Scale the sample data using the natural logarithm
log_samples = np.log(samples)

# Produce a scatter matrix for each pair of newly-transformed features
_ = sns.pairplot(log_data, diag_kind = 'kde')
```

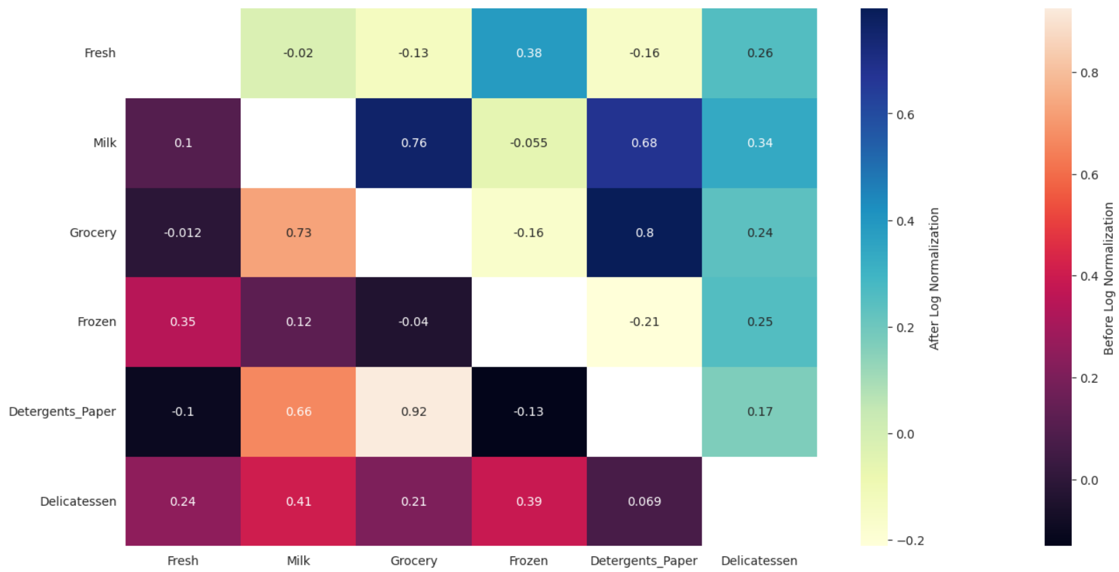


[22]:
```
# Display the log-transformed sample data
display(log_samples)
```

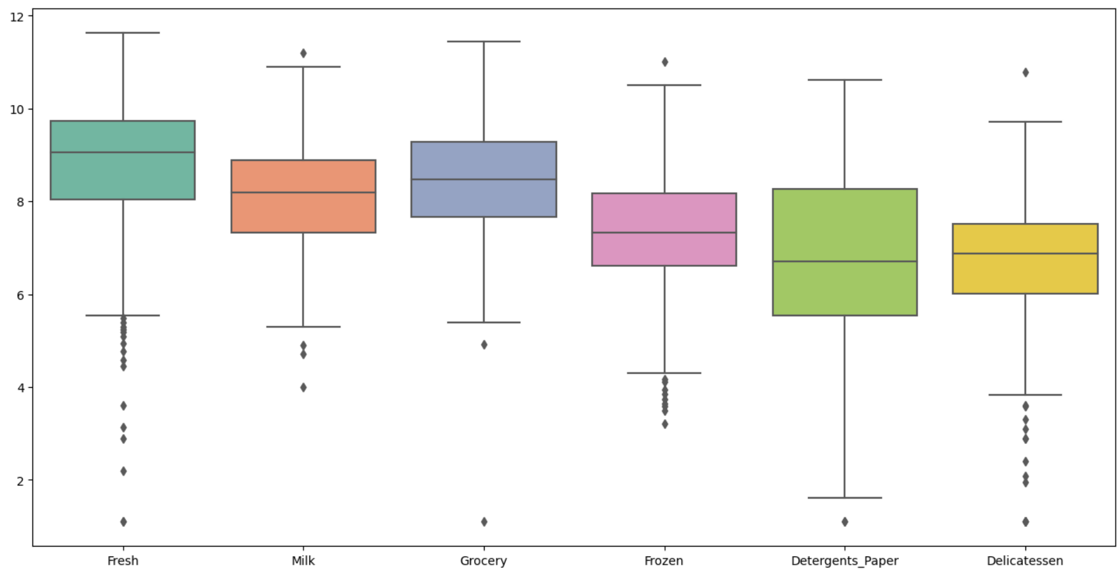
	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	8.068090	7.597396	7.494986	7.455877	5.455321	6.565265
1	7.983440	8.773230	8.945984	7.848153	8.427925	7.234177
2	9.941843	8.686261	9.128262	7.472501	7.850883	7.347944

[23]:
```
# Display the correlation heatmap
log_corr = log_data.corr()

f = plt.figure(figsize = (16,8))
mask = np.zeros_like(corr)
```

```
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
    ax1 = sns.heatmap(corr, annot=True, mask=mask, cbar_kws={'label': 'Before␣
 ↪Log Normalization'})

mask2 = np.zeros_like(corr)
mask2[np.tril_indices_from(mask2)] = True
with sns.axes_style("white"):
    ax2 = sns.heatmap(log_corr, annot=True, mask=mask2, cmap="YlGnBu",␣
 ↪cbar_kws={'label': 'After Log Normalization'})
```



[24]:
```
# boxplot on the logdata
plt.figure(figsize = (16,8))
_ = sns.boxplot(data=log_data, palette="Set2")
```



[25]:

```
outliers_list = []
# For each feature find the data points with extreme high or low values
for feature in log_data.keys():

    # Calculate Q1 (25th percentile of the data) for the given feature
    Q1 = np.percentile(log_data[feature], 25)

    # Calculate Q3 (75th percentile of the data) for the given feature
    Q3 = np.percentile(log_data[feature], 75)

    # Use the interquartile range to calculate an outlier step (1.5 times the␣
↪interquartile range)
    step = (Q3 - Q1) * 1.5

    # Display the outliers
    print("Data points considered outliers for the feature '{}':".
↪format(feature))
    outliers = list(log_data[~((log_data[feature] >= Q1 - step) &␣
↪(log_data[feature] <= Q3 + step))].index.values)
    display(log_data[~((log_data[feature] >= Q1 - step) & (log_data[feature] <=␣
↪Q3 + step))])
    outliers_list.extend(outliers)

print("List of Outliers -> {}".format(outliers_list))
duplicate_outliers_list = list(set([x for x in outliers_list if outliers_list.
↪count(x) >= 2]))
duplicate_outliers_list.sort()
print("\nList of Common Outliers -> {}".format(duplicate_outliers_list))
```

```python
# Select the indices for data points you wish to remove
outliers  = duplicate_outliers_list

# Remove the outliers, if any were specified
good_data = log_data.drop(log_data.index[outliers]).reset_index(drop = True)
```

Data points considered outliers for the feature 'Fresh':

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
65	4.442651	9.950323	10.732651	3.583519	10.095388	7.260523
66	2.197225	7.335634	8.911530	5.164786	8.151333	3.295837
81	5.389072	9.163249	9.575192	5.645447	8.964184	5.049856
95	1.098612	7.979339	8.740657	6.086775	5.407172	6.563856
96	3.135494	7.869402	9.001839	4.976734	8.262043	5.379897
128	4.941642	9.087834	8.248791	4.955827	6.967909	1.098612
171	5.298317	10.160530	9.894245	6.478510	9.079434	8.740337
193	5.192957	8.156223	9.917982	6.865891	8.633731	6.501290
218	2.890372	8.923191	9.629380	7.158514	8.475746	8.759669
304	5.081404	8.917311	10.117510	6.424869	9.374413	7.787382
305	5.493061	9.468001	9.088399	6.683361	8.271037	5.351858
338	1.098612	5.808142	8.856661	9.655090	2.708050	6.309918
353	4.762174	8.742574	9.961898	5.429346	9.069007	7.013016
355	5.247024	6.588926	7.606885	5.501258	5.214936	4.844187
357	3.610918	7.150701	10.011086	4.919981	8.816853	4.700480
412	4.574711	8.190077	9.425452	4.584967	7.996317	4.127134

Data points considered outliers for the feature 'Milk':

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
86	10.039983	11.205013	10.377047	6.894670	9.906981	6.805723
98	6.220590	4.718499	6.656727	6.796824	4.025352	4.882802
154	6.432940	4.007333	4.919981	4.317488	1.945910	2.079442
356	10.029503	4.897840	5.384495	8.057377	2.197225	6.306275

Data points considered outliers for the feature 'Grocery':

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
75	9.923192	7.036148	1.098612	8.390949	1.098612	6.882437
154	6.432940	4.007333	4.919981	4.317488	1.945910	2.079442

Data points considered outliers for the feature 'Frozen':

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
38	8.431853	9.663261	9.723703	3.496508	8.847360	6.070738
57	8.597297	9.203618	9.257892	3.637586	8.932213	7.156177
65	4.442651	9.950323	10.732651	3.583519	10.095388	7.260523
145	10.000569	9.034080	10.457143	3.737670	9.440738	8.396155
175	7.759187	8.967632	9.382106	3.951244	8.341887	7.436617
264	6.978214	9.177714	9.645041	4.110874	8.696176	7.142827
325	10.395650	9.728181	9.519735	11.016479	7.148346	8.632128

```
420   8.402007  8.569026   9.490015   3.218876          8.827321      7.239215
429   9.060331  7.467371   8.183118   3.850148          4.430817      7.824446
439   7.932721  7.437206   7.828038   4.174387          6.167516      3.951244
```

```
Data points considered outliers for the feature 'Detergents_Paper':
```

```
        Fresh      Milk   Grocery    Frozen  Detergents_Paper  Delicatessen
75   9.923192  7.036148  1.098612  8.390949          1.098612      6.882437
161  9.428190  6.291569  5.645447  6.995766          1.098612      7.711101
```

```
Data points considered outliers for the feature 'Delicatessen':
```

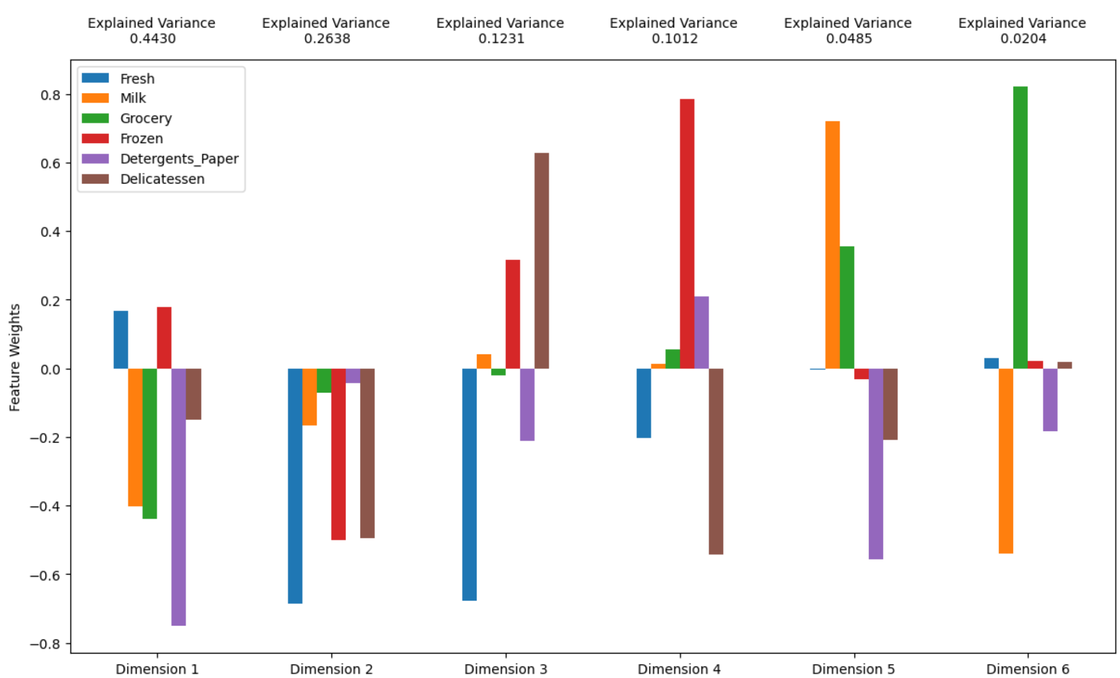
```
         Fresh       Milk    Grocery     Frozen  Detergents_Paper  \
66    2.197225   7.335634   8.911530   5.164786          8.151333
109   7.248504   9.724899  10.274568   6.511745          6.728629
128   4.941642   9.087834   8.248791   4.955827          6.967909
137   8.034955   8.997147   9.021840   6.493754          6.580639
142  10.519646   8.875147   9.018332   8.004700          2.995732
154   6.432940   4.007333   4.919981   4.317488          1.945910
183  10.514529  10.690808   9.911952  10.505999          5.476464
184   5.789960   6.822197   8.457443   4.304065          5.811141
187   7.798933   8.987447   9.192075   8.743372          8.148735
203   6.368187   6.529419   7.703459   6.150603          6.860664
233   6.871091   8.513988   8.106515   6.842683          6.013715
285  10.602965   6.461468   8.188689   6.948897          6.077642
289  10.663966   5.655992   6.154858   7.235619          3.465736
343   7.431892   8.848509  10.177932   7.283448          9.646593

     Delicatessen
66       3.295837
109      1.098612
128      1.098612
137      3.583519
142      1.098612
154      2.079442
183     10.777768
184      2.397895
187      1.098612
203      2.890372
233      1.945910
285      2.890372
289      3.091042
343      3.610918
```

```
List of Outliers -> [65, 66, 81, 95, 96, 128, 171, 193, 218, 304, 305, 338, 353,
355, 357, 412, 86, 98, 154, 356, 75, 154, 38, 57, 65, 145, 175, 264, 325, 420,
429, 439, 75, 161, 66, 109, 128, 137, 142, 154, 183, 184, 187, 203, 233, 285,
289, 343]

List of Common Outliers -> [65, 66, 75, 128, 154]
```

Feature Transformation

[26]:
```
# Apply PCA by fitting the good data with the same number of dimensions as␣
↪features
pca = PCA(n_components = 6, random_state=0)
pca.fit(good_data)

# Transform log_samples using the PCA fit above
pca_samples = pca.transform(log_samples)
print("Explained Variance Ratio => {}\n".format(pca.explained_variance_ratio_))
print("Explained Variance Ratio(csum) => {}\n".format(pca.
↪explained_variance_ratio_.cumsum()))

# Generate PCA results plot
pca_results = pca_results(good_data, pca)
```

```
Explained Variance Ratio => [0.44302505 0.26379218 0.1230638  0.10120908
0.04850196 0.02040793]

Explained Variance Ratio(csum) => [0.44302505 0.70681723 0.82988103 0.93109011
0.97959207 1.        ]
```



[27]:
```
# Display sample log-data after having a PCA transformation applied
display(pd.DataFrame(np.round(pca_samples, 4), columns = pca_results.index.
↪values))
```

```
   Dimension 1  Dimension 2  Dimension 3  Dimension 4  Dimension 5  \
0       1.5715       0.6914       0.7154      -0.0264       0.0495
1      -1.8145      -0.2029       0.7064       0.6552      -0.4010
2      -1.1818      -1.3883      -0.5519      -0.2136      -0.0931

   Dimension 6
0      -0.2803
1      -0.2483
2       0.1051
```

Dimensionality Reduction

[28]:
```
# Apply PCA by fitting the good data with only two dimensions
pca = PCA(n_components = 2, random_state=0)
pca.fit(good_data)

# Transform the good data using the PCA fit above
reduced_data = pca.transform(good_data)

# Transform log_samples using the PCA fit above
pca_samples = pca.transform(log_samples)

# Create a DataFrame for the reduced data
reduced_data = pd.DataFrame(reduced_data, columns = ['Dimension 1', 'Dimension␣
↪2'])
```

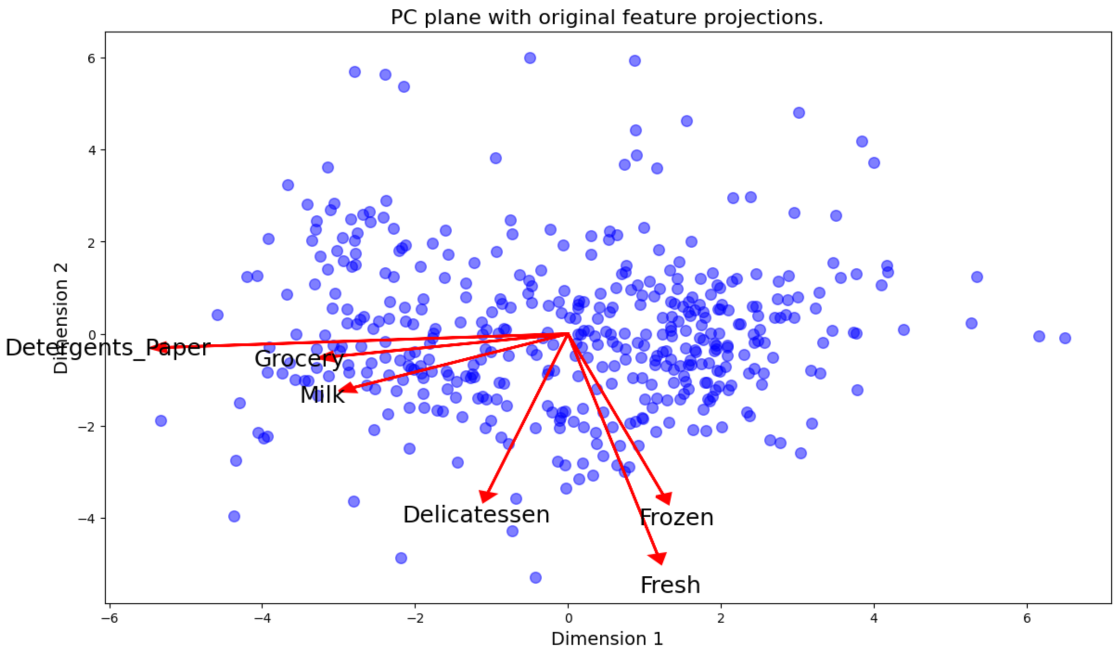
[29]:
```
# Display sample log-data after applying PCA transformation in two dimensions
display(pd.DataFrame(np.round(pca_samples, 4), columns = ['Dimension 1',␣
↪'Dimension 2']))
```

```
   Dimension 1  Dimension 2
0       1.5715       0.6914
1      -1.8145      -0.2029
2      -1.1818      -1.3883
```

Visualizing a Biplot

[30]:
```
# Create a biplot
biplot(good_data, reduced_data, pca)
```

[30]:
```
<Axes: title={'center': 'PC plane with original feature projections.'},
xlabel='Dimension 1', ylabel='Dimension 2'>
```



[31]:
```
def sil_coeff(no_clusters):
    # Apply your clustering algorithm of choice to the reduced data
    clusterer_1 = KMeans(n_clusters=no_clusters, random_state=0 )
    clusterer_1.fit(reduced_data)

    # Predict the cluster for each data point
    preds_1 = clusterer_1.predict(reduced_data)

    # Find the cluster centers
    centers_1 = clusterer_1.cluster_centers_

    # Predict the cluster for each transformed sample data point
    sample_preds_1 = clusterer_1.predict(pca_samples)

    # Calculate the mean silhouette coefficient for the number of clusters␣
 ↪chosen
    score = silhouette_score(reduced_data, preds_1)

    print("silhouette coefficient for `{}` clusters => {:.4f}".
 ↪format(no_clusters, score))

clusters_range = range(2,15)
for i in clusters_range:
    sil_coeff(i)
```

```
/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/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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(

silhouette coefficient for `2` clusters => 0.4263
silhouette coefficient for `3` clusters => 0.3969
silhouette coefficient for `4` clusters => 0.3320

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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(

silhouette coefficient for `5` clusters => 0.3510
silhouette coefficient for `6` clusters => 0.3666

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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(
```

```
silhouette coefficient for `7` clusters => 0.3633
silhouette coefficient for `8` clusters => 0.3510
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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(
```

```
silhouette coefficient for `9` clusters => 0.3541
silhouette coefficient for `10` clusters => 0.3510
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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(
```

```
silhouette coefficient for `11` clusters => 0.3519
silhouette coefficient for `12` clusters => 0.3509
```

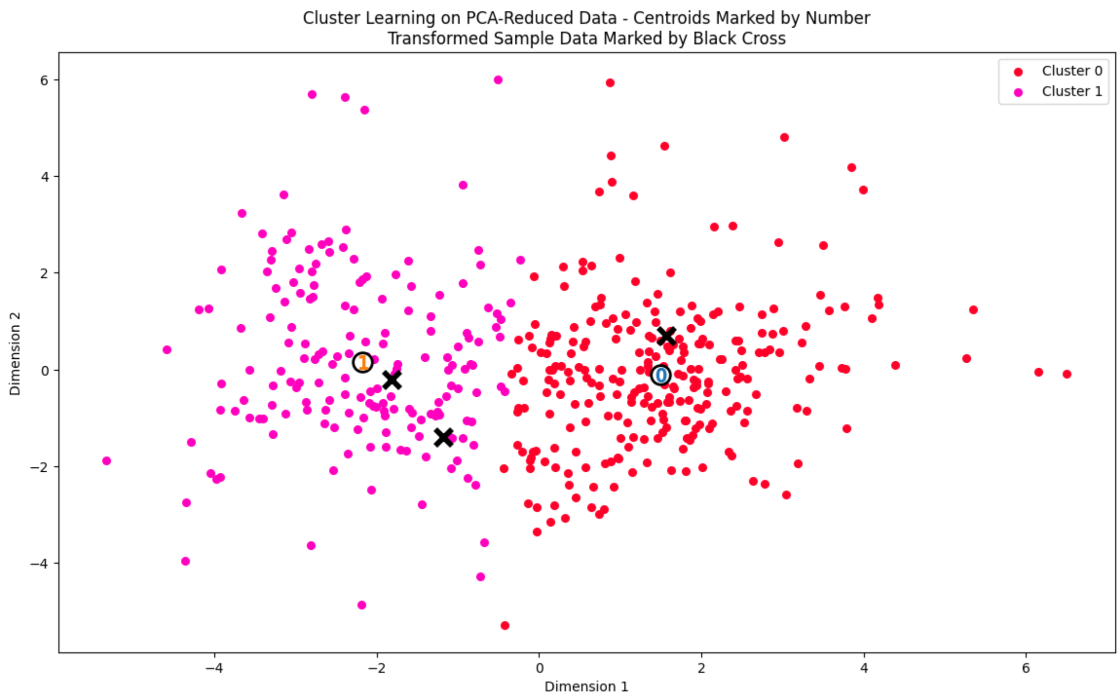
```
silhouette coefficient for `13` clusters => 0.3596
silhouette coefficient for `14` clusters => 0.3611
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  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/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  warnings.warn(
```

[32]:

```
# Display the results of the clustering from implementation for 2 clusters
clusterer = KMeans(n_clusters = 2)
clusterer.fit(reduced_data)
preds = clusterer.predict(reduced_data)
centers = clusterer.cluster_centers_
sample_preds = clusterer.predict(pca_samples)

cluster_results(reduced_data, preds, centers, pca_samples)
```

```
/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/base.py:439: UserWarning: X does
not have valid feature names, but KMeans was fitted with feature names
  warnings.warn(
<ipython-input-2-4dd6c4a109ef>:58: MatplotlibDeprecationWarning: The get_cmap
function was deprecated in Matplotlib 3.7 and will be removed two minor releases
later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  cmap = cm.get_cmap('gist_rainbow')
```



Data Recovery

[33]:
```
# Inverse transform the centers
log_centers = pca.inverse_transform(centers)

# Exponentiate the centers
true_centers = np.exp(log_centers)

# Display the true centers
segments = ['Segment {}'.format(i) for i in range(0,len(centers))]
true_centers = pd.DataFrame(np.round(true_centers), columns = data.keys())
true_centers.index = segments
display(true_centers)
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
Segment 0	8867.0	1897.0	2477.0	2088.0	294.0	681.0
Segment 1	4005.0	7900.0	12104.0	952.0	4561.0	1036.0

[34]:
```
display(data.mean(axis=0))
```

```
Fresh               12000.297727
Milk                 5796.265909
Grocery              7951.277273
Frozen               3071.931818
Detergents_Paper     2881.493182
```

```
Delicatessen         1524.870455
dtype: float64
```

[35]:
```
display(samples)
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	3191	1993	1799	1730	234	710
1	2932	6459	7677	2561	4573	1386
2	20782	5921	9212	1759	2568	1553

[36]:
```
# Display the predictions
for i, pred in enumerate(sample_preds):
    print("Sample point", i, "predicted to be in Cluster", pred)
```

```
Sample point 0 predicted to be in Cluster 0
Sample point 1 predicted to be in Cluster 1
Sample point 2 predicted to be in Cluster 1
```

[37]:
```
# Display the clustering results based on 'Channel' data
channel_results(reduced_data, outliers, pca_samples)
```

```
Dataset could not be loaded. Is the file missing?
```

[37]: False

Conclusion:

Use of the clustered data

- Image and Video Processing: In computer vision, clustering can be used to segment images or videos into regions of interest. For example, in medical imaging, clustering can help identify and separate different tissues or structures within an image.
- Anomaly Detection: Clustering can be used to detect anomalies or outliers in data. By identifying clusters of normal data points, any data points that fall outside these clusters can be flagged as potential anomalies.
- Document Classification: In natural language processing (NLP), text documents can be clustered based on their content or topics. This is useful for organizing large document collections, such as news articles, research papers, or customer reviews.
- Recommendation Systems: Clustering can be used in recommendation systems to group users or items with similar preferences. By identifying clusters of users who like similar products, recommendations can be more personalized.
- Genomic Analysis: In biology, clustering is used to group genes or proteins with similar functions or expression patterns. This helps researchers better understand biological processes and identify potential drug targets.
- Market Basket Analysis: In retail, clustering can be used to identify patterns in customer purchasing behavior. This information can be used to optimize product placement, promotions, and inventory management.

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

- Urban vs. Rural Customers: Urban customers might prefer and benefit from same-day or next-day delivery options due to proximity to distribution centers.Rural customers may require longer delivery windows and might appreciate cost-effective, slower shipping methods.
- Frequent Shoppers vs. Occasional Shoppers: Frequent shoppers may opt for subscription-based or premium delivery services, while occasional shoppers might prefer standard delivery.
- Price-Sensitive vs. Convenience-Seeking Customers: Price-sensitive customers may prefer slower, cost-effective delivery options, while convenience-seeking customers might be willing to pay more for faster delivery.
- B2B vs. B2C Customers: Business-to-business (B2B) customers might prioritize reliability and consistency in delivery times, whereas business-to-consumer (B2C) customers may have varied preferences.