Experiment No. 5
Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset
Date of Performance: 21/8/23
Date of Submission: 24/9/23

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

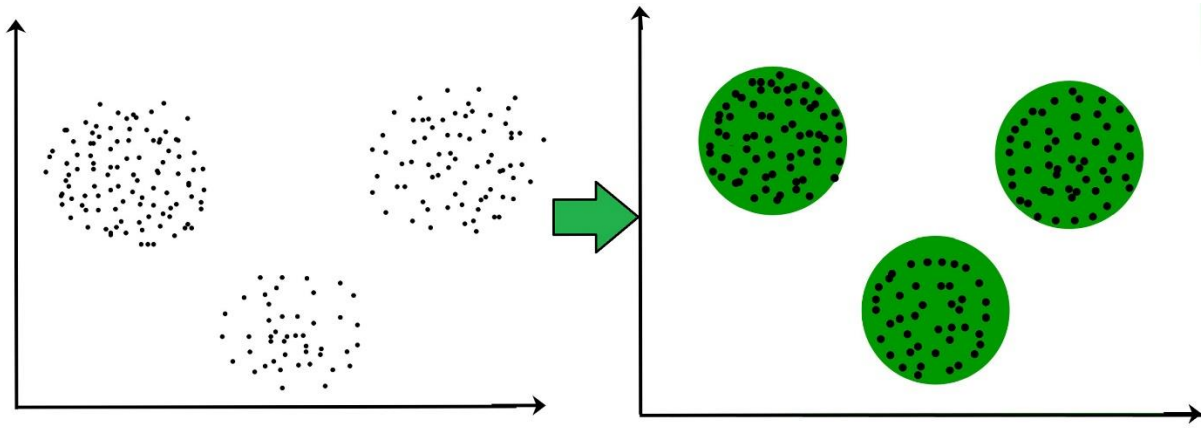
Objective: Able to perform various feature engineering tasks, apply Clustering Algorithm on the given dataset.

Theory:

It is basically a type of unsupervised learning method. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labeled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

For example: The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



Dataset:

This data set refers to clients of a wholesale distributor. It includes the annual spending in monetary units (m.u.) on diverse product categories. The wholesale distributor operating in different regions of Portugal has information on annual spending of several items in their stores across different regions and channels. The dataset consist of 440 large retailers annual spending on 6 different varieties of product in 3 different regions (lisbon , oporto, other) and across different sales channel (Hotel, channel)

Detailed overview of dataset

Records in the dataset = 440 ROWS

Columns in the dataset = 8 COLUMNS

FRESH: annual spending (m.u.) on fresh products (Continuous)

MILK:- annual spending (m.u.) on milk products (Continuous)

GROCERY:- annual spending (m.u.) on grocery products (Continuous)

FROZEN:- annual spending (m.u.) on frozen products (Continuous)

DETERGENTS_PAPER :- annual spending (m.u.) on detergents and paper products (Continuous)

DELICATESSEN:- annual spending (m.u.)on and delicatessen products (Continuous);

CHANNEL: - sales channel Hotel and Retailer

REGION:- three regions (Lisbon, Oporto, Other)

Code:

Conclusion:

Clustering wholesale customers is a strategic data analysis method that uncovers hidden spending patterns and behaviors within the customer base. Through this process, businesses can effectively categorize customers into high-value, regular, and low-value segments based on their distinctive spending habits. Once these customer segments are defined, companies can craft highly tailored marketing strategies. High-value customers can be rewarded and retained through loyalty programs, while regular customers can receive personalized product recommendations and promotions.
This personalized approach enhances customer engagement and drives revenue growth.

Furthermore, clustering assists in optimizing inventory management by aligning stock with the preferences of each customer segment.

For instance, businesses can ensure consistent supply of fresh products for clusters that prioritize them, reducing waste and improving resource allocation.

In conclusion, clustering wholesale customers not only provides valuable insights into customer behavior but also empowers businesses to deliver customized experiences, increase customer satisfaction, and streamline operations for long-term success.

Wholesale Customers | Segmentation

```
#Data handling Imports
import pandas as pd
import numpy as np

#Notebook arrange Imports
import warnings
warnings.filterwarnings('ignore')

#Calculation Imports
import math
import random

#Visualisation Imports
import seaborn as sns
import pylab
pylab.style.use('seaborn-pastel')
import matplotlib.pyplot as plt
%matplotlib inline

#Feature Selection Imports
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import mutual_info_classif
from sklearn.feature_selection import GenericUnivariateSelect

#Outlier Handling Imports
from scipy.stats.mstats import winsorize

#Normalization & Scaler Imports
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from scipy.stats import boxcox, probplot, norm, shapiro
from sklearn.decomposition import PCA

#Sampling Imports
from sklearn.model_selection import KFold

#Encoding Imports
from sklearn.preprocessing import LabelEncoder

#Modeling Imports
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

#Clustering Imports
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import Birch
from sklearn.cluster import MiniBatchKMeans
import scipy.cluster.hierarchy as shc

#Accuracy Validation Imports
from sklearn import metrics
from sklearn.metrics import auc, roc_curve, f1_score, accuracy_score,precision_recall_curve,\
confusion_matrix, classification_report
from sklearn.metrics import precision_recall_fscore_support
from sklearn.model_selection import cross_val_score

#Other required libraries
import time
start = time. time()
```

[1] Data preprocessing

```
df = pd.read_csv('/content/Wholesale customers data.csv')
#Taking copy for missing value handling purpose
```

```
df_MV = df.copy()
# df.head()
df_MV.sample(5)
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
102	2	3	2932	6459	7677	2561	4573	1386
245	2	1	3062	6154	13916	230	8933	2784
401	1	3	27167	2801	2128	13223	92	1902
140	1	3	17623	4280	7305	2279	960	2616
195	1	3	17023	5139	5230	7888	330	1755

```
#Checking the size of the dataset
df_MV.shape
```

```
(440, 8)
```

```
#Checking for missing values
df_MV.isnull().sum()
```

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

```
#As we don't have any missing values in dataset . checking structure to put sunthetic nulls
df_MV.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Channel           440 non-null    int64
 1   Region            440 non-null    int64
 2   Fresh             440 non-null    int64
 3   Milk              440 non-null    int64
 4   Grocery           440 non-null    int64
 5   Frozen            440 non-null    int64
 6   Detergents_Paper  440 non-null    int64
 7   Delicassen        440 non-null    int64
dtypes: int64(8)
memory usage: 27.6 KB
```

```
#Count of unique values and categories featurewise for Channel & Region Column
column_list = ['Channel','Region']
#print(column_list)
for col in column_list:
    print('Feature: {:<9s} | Unique-Count: {:<3} | Categories: {:}'.format(col,df_MV[col].nunique(),df_MV[col].unique()))
```

```
Feature: Channel   | Unique-Count: 2   | Categories: [2 1]
Feature: Region    | Unique-Count: 3   | Categories: [3 1 2]
```

```
#As dataset is not having any null values imputing null values to explain importance of preprocessing of null value handling
#Inserting 3% of each column's values as null... [Synthetic]
#As we have only 440 records.. doesn't want to distort..
ix = [(row, col) for row in range(df_MV.shape[0]) for col in range(df_MV.shape[1])]
for row, col in random.sample(ix, int(round(.03*len(ix)))):
    df_MV.iat[row, col] = None
```

```
#Checking null values after synthetic null insert
df_MV.isnull().sum().sort_values(ascending=False)
```

```
Detergents_Paper    16
Fresh               15
Grocery             14
Channel             13
Frozen              13
```

```
Milk               12
Delicassen         12
Region             11
dtype: int64
```

```
#Assigning unique category to region & Channel feature as '3' & '4' recpectively. For null handling
df_MV['Channel'] = df_MV['Channel'].fillna(3)
df_MV['Region'] = df_MV['Region'].fillna(4)
df_MV.isnull().sum().sort_values(ascending=False)
```

```
Detergents_Paper    16
Fresh               15
Grocery             14
Frozen              13
Milk                12
Delicassen          12
Channel              0
Region               0
dtype: int64
```

```
#For rest null value imputation let's check insight of data
df_MV.describe()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	425.000000	428.000000	426.000000	427.000000	424.000000	428.000000
mean	1.377273	2.572727	12135.272941	5821.287383	8019.948357	3063.262295	2906.207547	1524.011682
std	0.542862	0.801014	12818.230597	7459.208623	9535.928012	4891.107261	4823.133559	2849.943010
min	1.000000	1.000000	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	1.000000	2.000000	3097.000000	1506.250000	2183.000000	744.000000	256.000000	405.750000
50%	1.000000	3.000000	8590.000000	3616.500000	4903.500000	1535.000000	823.000000	960.500000
75%	2.000000	3.000000	17023.000000	7217.500000	10675.250000	3512.500000	3922.000000	1824.750000
max	3.000000	4.000000	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	47943.000000

```
#Median and Mean by grouping cannel & region
df_MVal = df_MV.drop(['Channel','Region'], axis=1)
df_MV.groupby(['Channel', 'Region']).agg(['median','mean']).round(1)
```

		Fresh		Milk		Grocery		Frozen		Detergents_Paper		Delicassen	
		median	mean	median	mean	median	mean	median	mean	median	mean	median	mean
Channel	**Region**												
1.0	**1.0**	8885.0	13059.7	2179.0	3638.7	2501.0	3855.4	2077.0	2951.2	405.5	813.0	693.0	1092.1
	2.0	9790.0	11892.7	1560.5	2334.5	3315.0	4388.6	2679.0	5889.4	294.5	473.6	898.0	1162.0
	3.0	10253.0	14374.3	2096.0	3379.9	2625.5	3952.4	1870.5	3642.4	355.0	782.5	819.0	1580.0
	4.0	6211.0	9618.8	2761.0	4128.7	2918.5	3435.5	1089.0	3996.8	721.0	641.8	665.0	886.0
2.0	**1.0**	2926.0	5200.0	8866.0	10788.5	16106.0	18471.9	1522.0	2584.1	6374.0	8396.4	1414.0	1871.9
	2.0	6468.0	7289.8	7138.5	9596.0	12469.0	16326.3	934.0	1540.6	6094.0	8318.1	1037.0	1239.0
	3.0	6705.0	9858.9	8390.5	11296.8	12188.0	16162.3	1028.5	1430.9	5141.0	7108.0	1423.0	1844.8
	4.0	14044.5	14044.5	5837.5	5837.5	9253.5	9253.5	4708.5	4708.5	1332.5	1332.5	1214.0	1214.0
3.0	**1.0**	3061.5	3061.5	8683.5	8683.5	11199.5	11199.5	1351.5	1351.5	4894.5	4894.5	1801.5	1801.5
	2.0	5113.0	5113.0	1486.0	1486.0	4583.0	4583.0	5127.0	5127.0	492.0	492.0	739.0	739.0
	3.0	8258.0	11954.8	4888.0	5059.7	4387.5	5105.2	4186.5	4004.3	549.0	1637.4	864.0	1100.9

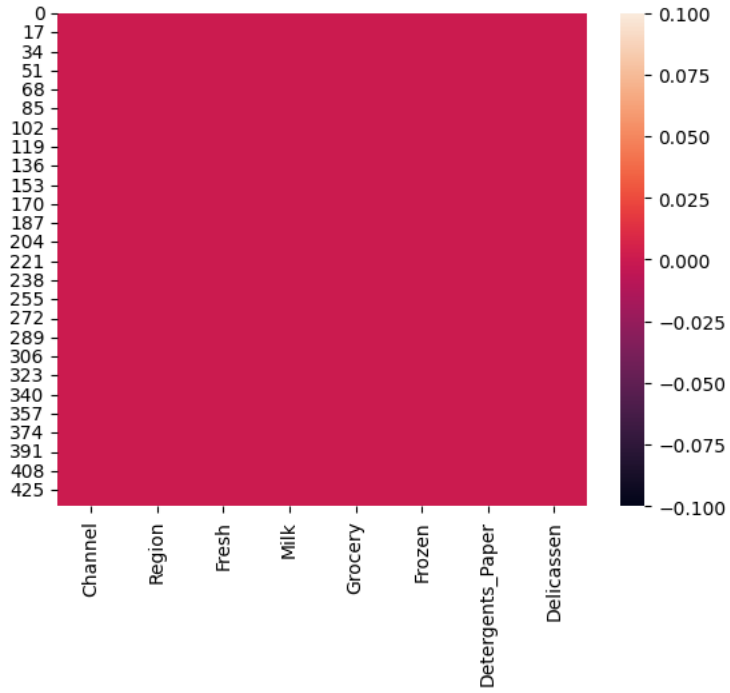
```
#As per above we can easily understood filling by mean is optimal as grouping with Channel & Region won't help here.
#Vast difference between median and mean
df_MV.fillna(df_MV.mean(), inplace=True)
df_MV.isnull().sum().sort_values(ascending=False)
```

```
Channel            0
Region             0
Fresh              0
Milk               0
```

```
Grocery            0
Frozen             0
Detergents_Paper   0
Delicassen         0
dtype: int64
```

```
#Null heatmap to visualize the dataset
sns.heatmap(df.isnull())
```

```
<Axes: >
```



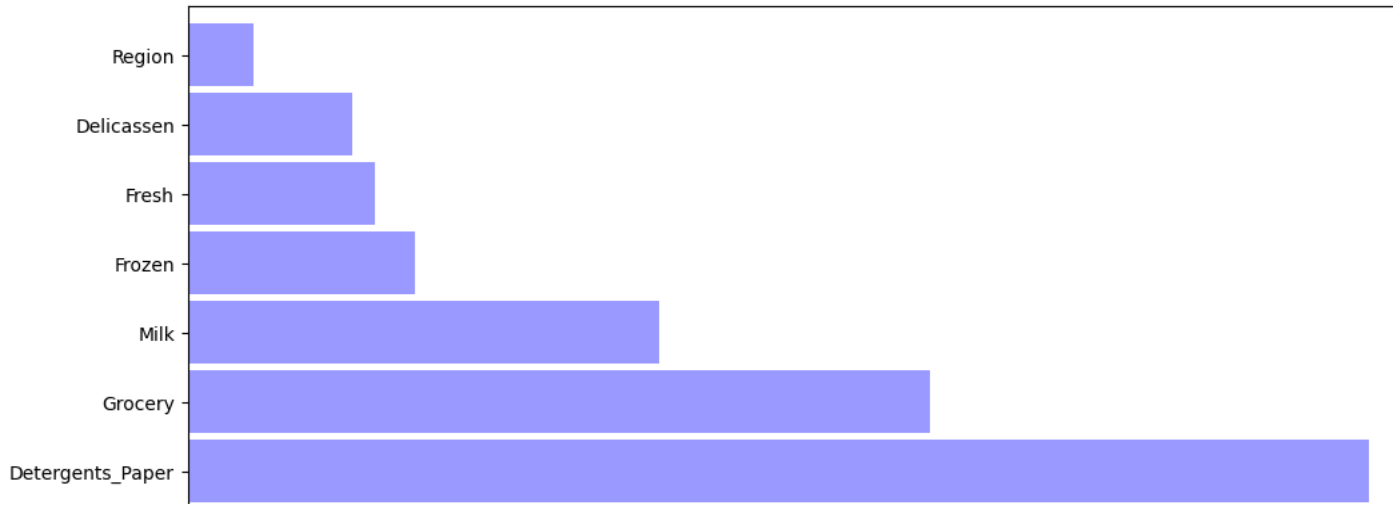

```
#Checking the correlation
corr = df.corr()
mask = np.triu(np.ones_like(corr, dtype=np.bool))
f, ax = plt.subplots(figsize=(9, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
#Drawing heatmap
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1, center=0.5, square=True, linewidths=.5, cbar_kws={"shrink": .6},annot=True)
plt.title("Correlation", fontsize =10)
```

```
Text(0.5, 1.0, 'Correlation')
```


```
#Target variable 'Channel'. let's see other feature selection techniques.
X = df.drop('Channel', axis=1)
y = df['Channel']

# Apply SelectKBest Algorithm using chi2 score function
kbest_features = SelectKBest(score_func=chi2, k=6)
ord_features = kbest_features.fit(X, y)
df_scores = pd.DataFrame(ord_features.scores_, columns=["Score"])
df_columns = pd.DataFrame(X.columns)
k_features = pd.concat([df_columns, df_scores], axis=1)
k_features.columns=['Features','Score']
k_features
```

	Features	Score
0	Region	3.981484e-01
1	Fresh	1.674662e+05
2	Milk	8.756852e+05
3	Grocery	1.848001e+06
4	Frozen	1.374907e+05
5	Detergents_Paper	1.401016e+06
6	Delicassen	7.183162e+03

```
# Finding the top ranked fetures and plotting
ranked_features = pd.Series(model.feature_importances_, index = X.columns)
plt.figure(figsize=(12, 5))
ranked_features.nlargest(15).plot(kind='barh', color='blue',  width =0.9 , linewidth = 0.05, alpha=0.4)
plt.show()
```




```
#the sorted values of mean higer score
mutual_info = mutual_info_classif(X, y)
mutual_data = pd.Series(mutual_info, index = X.columns)
mutual_data.sort_values(ascending=False)
```

```
Grocery            0.237157
Detergents_Paper   0.224623
Milk               0.115816
Frozen             0.069639
Region             0.004907
Fresh              0.000816
```

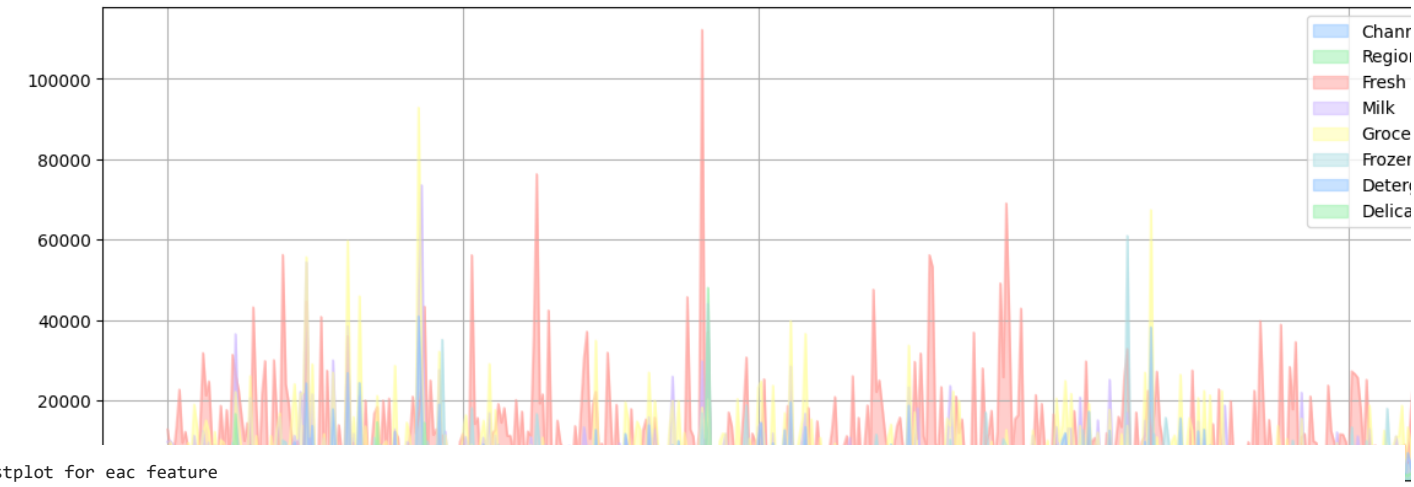
```
Delicassen          0.000000
dtype: float64
```

```
#Let's encode and select the best features

label_encoder = LabelEncoder()
df_1 = df.apply(label_encoder.fit_transform)
# X_feature = df.drop('Channel', axis=1)
# Y_label = df['Channel']
X = df_1.drop('Channel', axis=1)
Y = df_1['Channel']


#trans = GenericUnivariateSelect(score_func=mutual_info_classif, mode='k_best', param=15)
trans = GenericUnivariateSelect(score_func = mutual_info_classif, mode='percentile', param = 70)
trans_feat = trans.fit_transform(X, Y)
columns_ = df_1.iloc[:, 1:].columns[trans.get_support()].values

# X_feature as tranformed top feature variables
X_feature = pd.DataFrame(trans_feat, columns=columns_)

# Y_label with only target variable
Y_label = Y

X_feature.columns
```

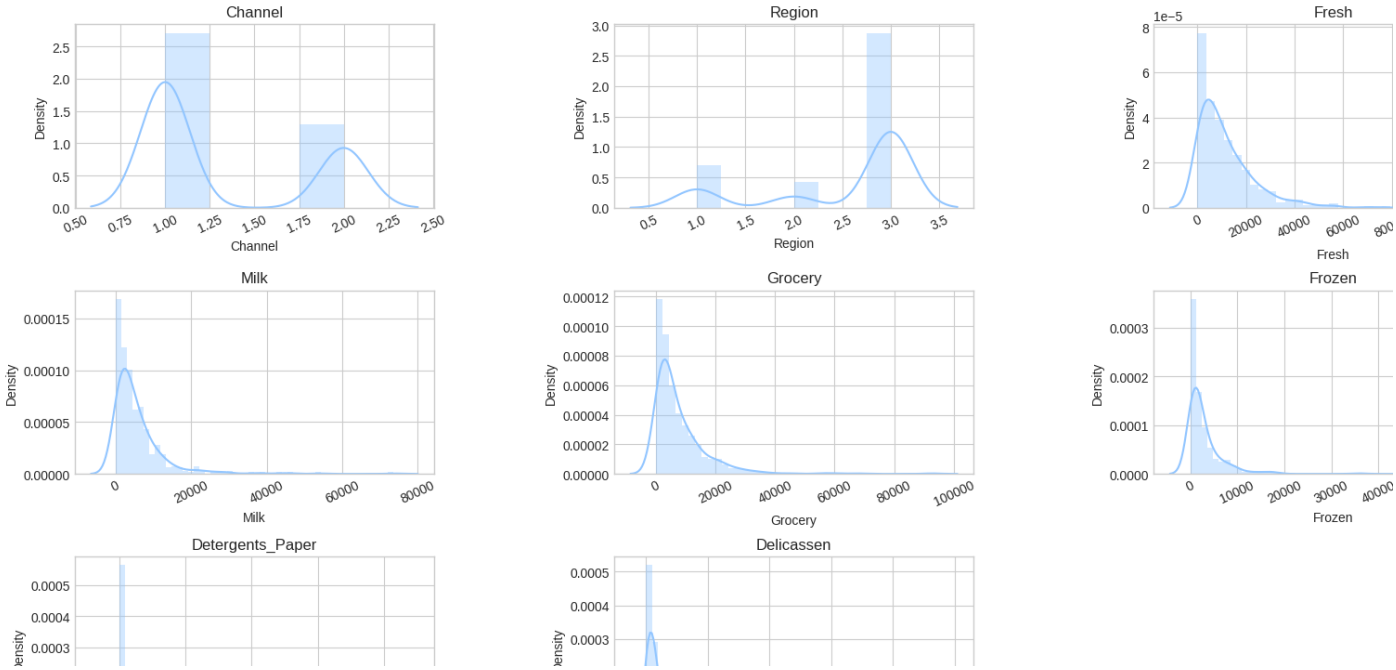
```
Index(['Region', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper'], dtype='object')
```

[2] Data visualization

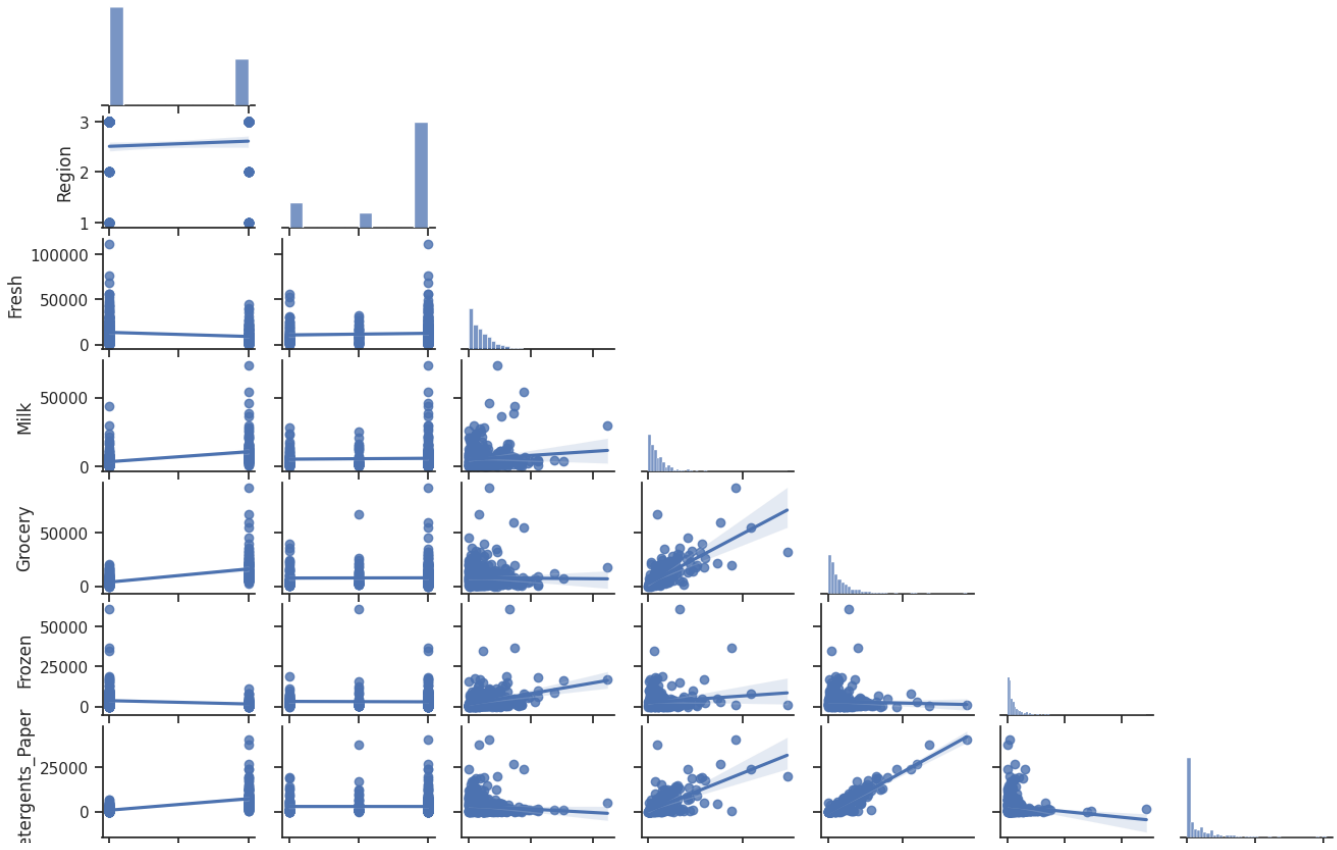
```
#1. Univariate analysis [Each feature individually]
#Plotting all features stacked
df.plot.area(stacked=False,figsize=(15,5))
pylab.grid(); pylab.show()
```




```
#Histplot for eac feature
def plot_draw(df, cols=5, width=10, height=10, hspace=0.2, wspace=0.5):
    """Ploting the individual feature histplot"""
    plt.style.use('seaborn-whitegrid')
    fig = plt.figure(figsize=(width,height))
    fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=wspace, hspace=hspace)
    rows = math.ceil(float(df.shape[1]) / cols)
    for i, column in enumerate(df.columns):
        ax = fig.add_subplot(rows, cols, i + 1)
        ax.set_title(column)
        if df.dtypes[column] == np.object:
            g = sns.countplot(y=column, data=df)
            substrings = [s.get_text()[:18] for s in g.get_yticklabels()]
            g.set(yticklabels=substrings)
            plt.xticks(rotation=25)
        else:
            g = sns.distplot(df[column])
            plt.xticks(rotation=25)

plot_draw(df, cols=3, width=20, height=10, hspace=0.45, wspace=0.5)
```




```
#2. Bivariate analysis [Pairwise]
#This is not needed for our dataset perspective but for visulization purpose let's plot
sns.set(style="ticks")
graph = sns.pairplot(df,corner=True,kind='reg')
graph.fig.set_size_inches(15,10)
```



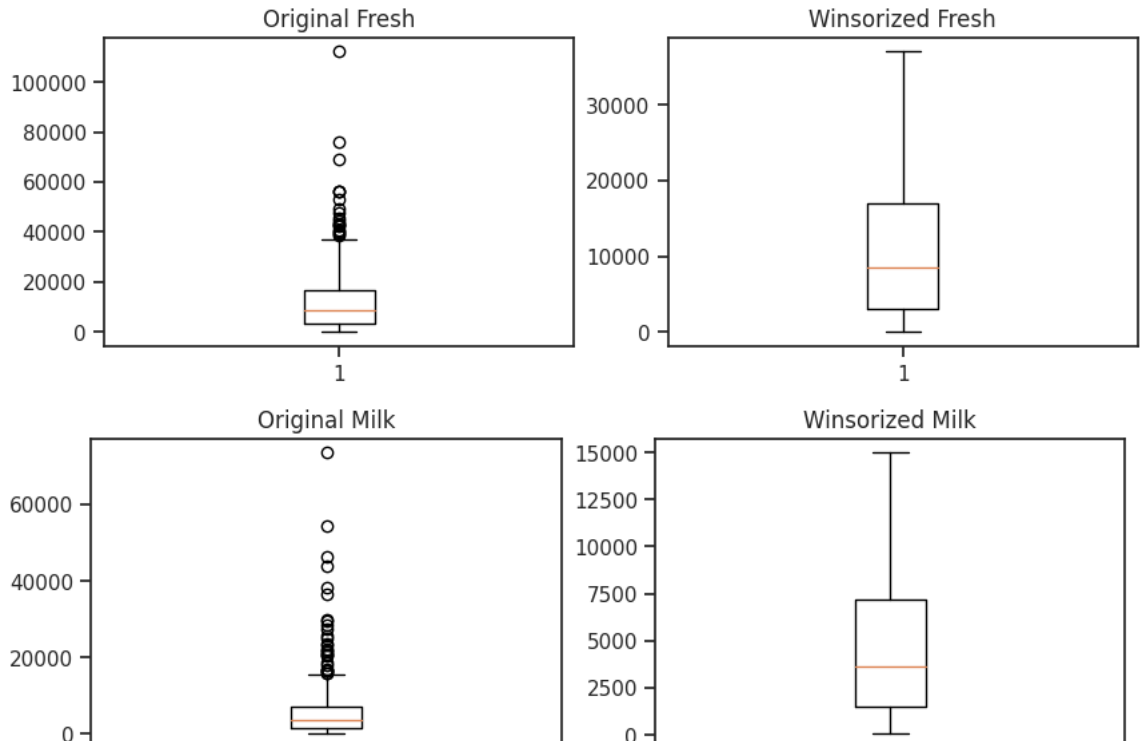

```
#Let's check central tendency for each feature
df.agg(['median','mean','std']).round(2)
```

```
#Outlier detection, measure in percentage
num_col = df.columns.tolist()
#Function to detect the outliers using IQR
def outlier_count(col, data=df):
    #q75, q25 = np.percentile(data[col], [25, 75])
    # calculate the interquartile range(Q1,Q3)
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    min_val = Q1 - (IQR*1.5)
    max_val = Q3 + (IQR*1.5)
    #Finding the length of data that is more than max threshold and lesser than min threshold
    outlier_count = len(np.where((data[col] > max_val) | (data[col] < min_val))[0])
    outlier_percent = round(outlier_count/len(data[col])*100, 2)
    print('{:<20} {:<20} {:.2f}%'.format(col,outlier_count,outlier_percent))

#Looping over all the numerical columns to outlier count function to find the total count of outliers in data.
print("\n"+20*'*' + ' Outliers ' + 20*'*'+"\n")
print('{:<20} {:<20} {:<20}'.format('Variable Name','Number Of Outlier','Outlier(%)'))
for col in num_col:
    outlier_count(col)
```

```
 ******************** Outliers ********************

 Variable Name        Number Of Outlier    Outlier(%)
 Channel              0                    0.00%
 Region               0                    0.00%
 Fresh                20                   4.55%
 Milk                 28                   6.36%
 Grocery              24                   5.45%
 Frozen               43                   9.77%
 Detergents_Paper     30                   6.82%
 Delicassen           27                   6.14%
```

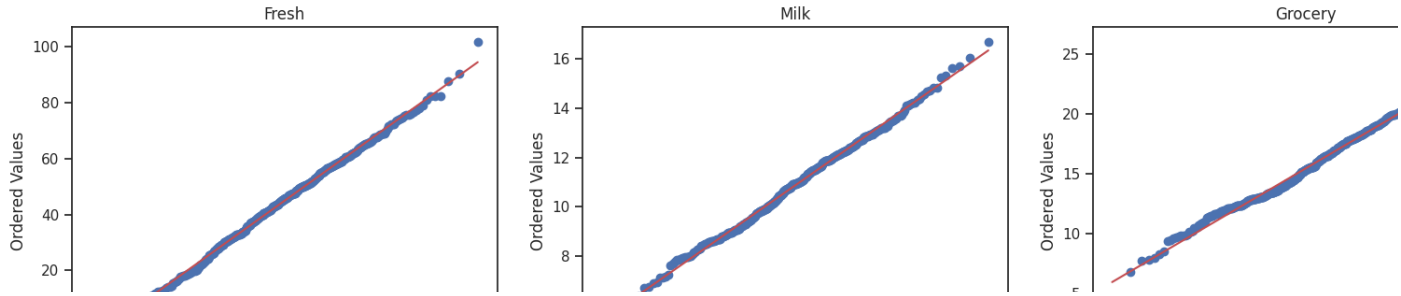
```
#Using function applying winsorize technique to cap the outliers and adding the new winsorized column to winsor_dict
# which can be used for futher implementation.
def winsor(col, lower_limit=0, upper_limit=0, show_plot=True):

    #Using scipy.stats.mstats.winsorize to each column
    winsor_data = winsorize(df[col], limits=(lower_limit, upper_limit))

    #Assigning the winsorized data from each column to  dict
    winsor_dict[col] = winsor_data

    #Using box plot, visializing the data to check the outliers before and after winsorizing
    if show_plot == True:
        plt.figure(figsize=(10,3))

        #draw plot with original dataset
        plt.subplot(121)
        plt.boxplot(df[col])
        plt.title('Original {}'.format(col))

        #draw plot with winsorized dataset
        plt.subplot(122)
        plt.boxplot(winsor_data)

        #assigning titile to the plot
        plt.title('Winsorized {}'.format(col))
        plt.show()


#Creating an empty dict to load all the winsorised data
winsor_dict = {}

#From the analysis found from the box plot, based on the outliers position,
#various limit has been experimented to limit the outlier count.

#In boxplot 2 ['Fresh'], It is seen that the outliers are in the upper boundanday of the plot,
winsor(num_col[2], upper_limit = 0.0455, show_plot=True)

#In boxplot 3 ['Milk'], It is seen that the outliers are in the upper boundanday of the plot,
winsor(num_col[3], upper_limit = 0.067, show_plot=True)

#In boxplot 4 ['grocery'], It is seen that the outliers are in the upper boundanday of the plot,
```

```
winsor(num_col[4], upper_limit = 0.06, show_plot=True)

#In boxplot 5 ['Frozen'], It is seen that the outliers are in the upper boundanday of the plot,
winsor(num_col[4], upper_limit = 0.0977, show_plot=True)

#In boxplot 6 ['Detergents_Paper'], It is seen that the outliers are in the upper boundanday of the plot,
winsor(num_col[4], upper_limit = 0.0682, show_plot=True)

#In boxplot 7 ['Delicassen'], It is seen that the outliers are in the upper boundanday of the plot,
winsor(num_col[4], upper_limit = 0.0614, show_plot=True)
```



Feature Transformation & Normalization

```
#All the variable are statistically non normally distributed.Let's try BoxCox transformation
shapiro_test = {}
lambdas = {}
j=2
plt.figure(figsize=(20, 10))
for i in range(6):
    ax = plt.subplot(2,3,i+1)
    x, lbd = boxcox(df[df.columns[j]])
    probplot(x = x, dist=norm, plot=ax)
    plt.title(df.columns[j])
    shapiro_test[df.columns[j]] = shapiro(x)
    lambdas[df.columns[j]] = lbd
    j+=1

plt.show()

pd.DataFrame(shapiro_test, index=['Statistic', 'p-value']).transpose()
```




```
#Using standard scalaer let's Transform & Normalize the data and visulize it.
sc=StandardScaler()
scaled_data=sc.fit_transform(df)

norm_data=normalize(df)

df=pd.DataFrame(scaled_data,columns=df.columns)
df_SN=pd.DataFrame(norm_data,columns=df.columns)
df_SN.head()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	0.000112	0.000168	0.708333	0.539874	0.422741	0.011965	0.149505	0.074809
1	0.000125	0.000188	0.442198	0.614704	0.599540	0.110409	0.206342	0.111286
2	0.000125	0.000187	0.396552	0.549792	0.479632	0.150119	0.219467	0.489619
3	0.000065	0.000194	0.856837	0.077254	0.272650	0.413659	0.032749	0.115494
4	0.000079	0.000119	0.895416	0.214203	0.284997	0.155010	0.070358	0.205294

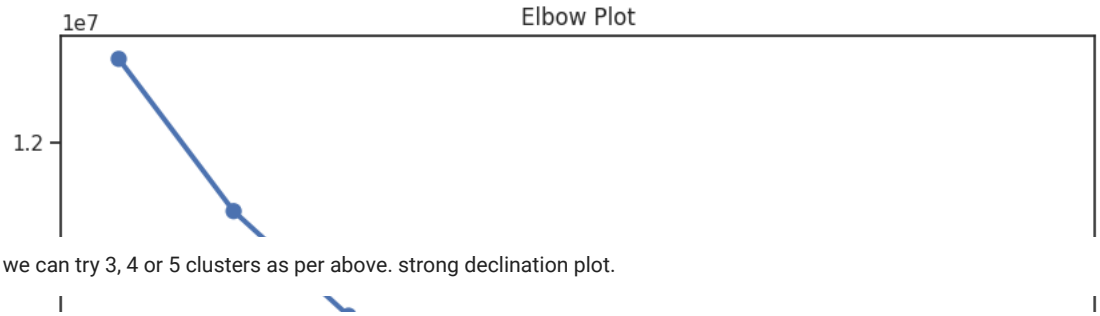
	Statistic	p-value
Milk	0.996949	0.584163

We have normalized the dataset. but, here, for modeling purpose for our dataset, it would be better to go without normalization.

[4] Unsupervised learning

For using K-Means algorithm. Let's determine the optimal value of clusters here.

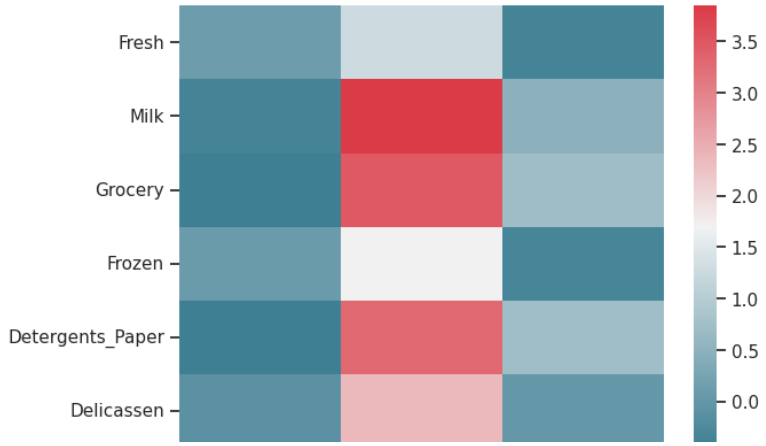
```
#As we already have our scaled data ready, lets do principle component analysis
#and print elbow plot to determine the optimal number of clusters.

PCA_train = PCA(2).fit_transform(scaled_data)
ps = pd.DataFrame(PCA_train)

le = {}
for k in range(2,11):
    kmeans = KMeans(n_clusters = k, random_state=123)
    Y_label = kmeans.fit_predict(X_feature)
    le[k] = kmeans.inertia_

plt.figure(figsize=(10,5))
plt.title('Elbow Plot')
sns.pointplot(x = list(le.keys()), y = list(le.values()))

plt.show()
```

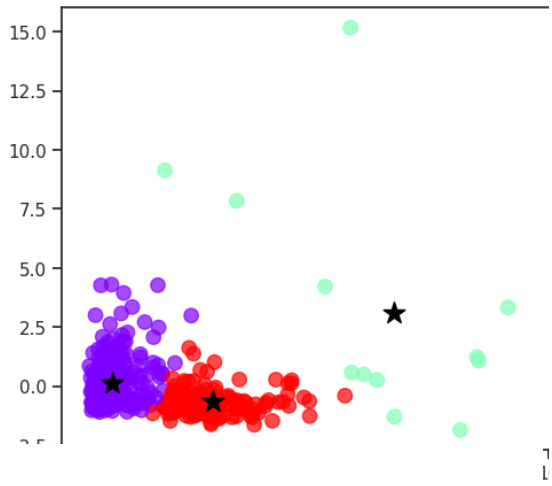


Here, we can try 3, 4 or 5 clusters as per above. strong declination plot.

```
#Let's parallely plot the heatmap and scatter plot to see the segmentation
#Cluster=3
kmeans = KMeans(n_clusters=3, random_state=123).fit(ps)
y_kmeans = kmeans.predict(ps)
df = df.assign(segment = kmeans.labels_)
kmeans_3_means = df.drop(['Channel','Region'], axis=1).groupby('segment').mean()

lab = kmeans.labels_

plt.figure(figsize=(15,5))

plt.subplot(1,2,1)
sns.heatmap(kmeans_3_means.T, cmap=cmap)

plt.subplot(1,2,2)
plt.scatter(ps[0], ps[1],c = y_kmeans, s=80, cmap='rainbow',alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], marker = '*', color='black', s=200)
```

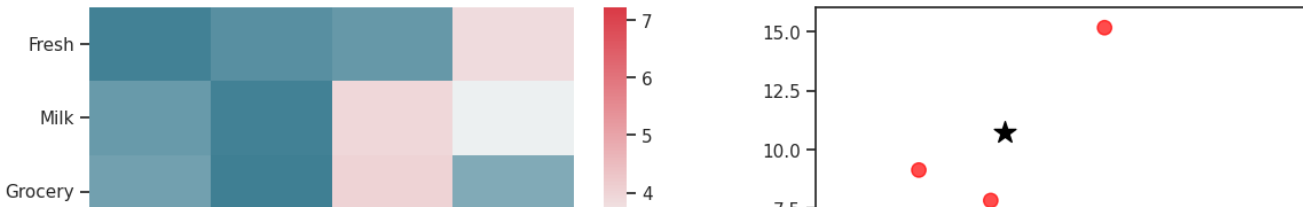
```
<matplotlib.collections.PathCollection at 0x7da469140310>
```





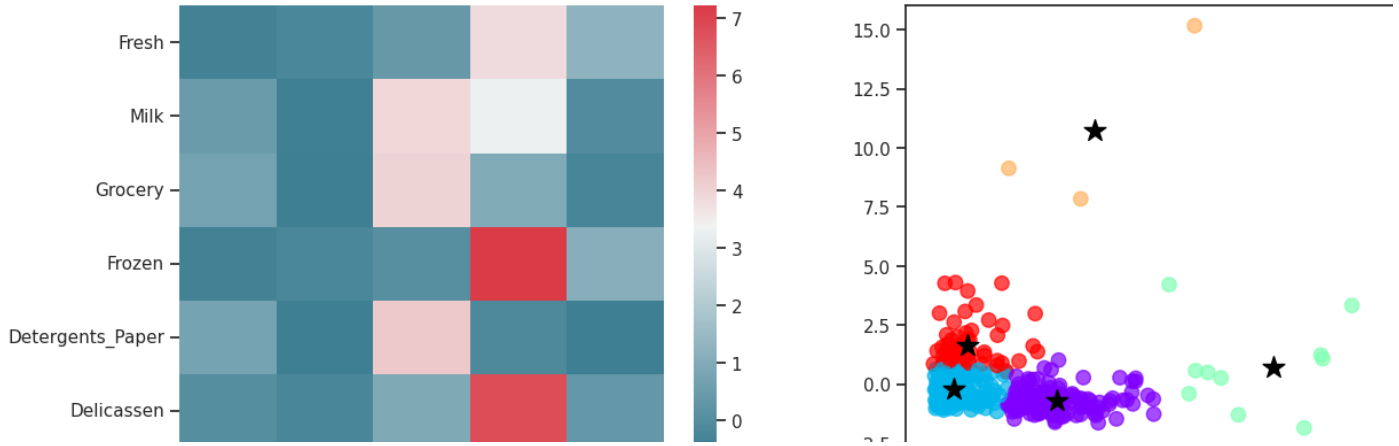

```
#Let's parallely plot the heatmap and scatter plot to see the segmentation
#Cluster=4
kmeans = KMeans(n_clusters=4, random_state=123).fit(ps)
y_kmeans = kmeans.predict(ps)
df = df.assign(segment = kmeans.labels_)
kmeans_3_means = df.drop(['Channel','Region'], axis=1).groupby('segment').mean()

lab = kmeans.labels_

plt.figure(figsize=(15,5))

plt.subplot(1,2,1)
sns.heatmap(kmeans_3_means.T, cmap=cmap)

plt.subplot(1,2,2)
plt.scatter(ps[0], ps[1],c = y_kmeans, s=80, cmap='rainbow',alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], marker = '*', color='black', s=200)
```

```
<matplotlib.collections.PathCollection at 0x7da46bdbb190>
```




```
#Let's parallely plot the heatmap and scatter plot to see the segmentation
#Cluster=5
kmeans = KMeans(n_clusters=5, random_state=123).fit(ps)
y_kmeans = kmeans.predict(ps)
df = df.assign(segment = kmeans.labels_)
kmeans_3_means = df.drop(['Channel','Region'], axis=1).groupby('segment').mean()

lab = kmeans.labels_

plt.figure(figsize=(15,5))

plt.subplot(1,2,1)
sns.heatmap(kmeans_3_means.T, cmap=cmap)

plt.subplot(1,2,2)
plt.scatter(ps[0], ps[1],c = y_kmeans, s=80, cmap='rainbow',alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], marker = '*', color='black', s=200)
```

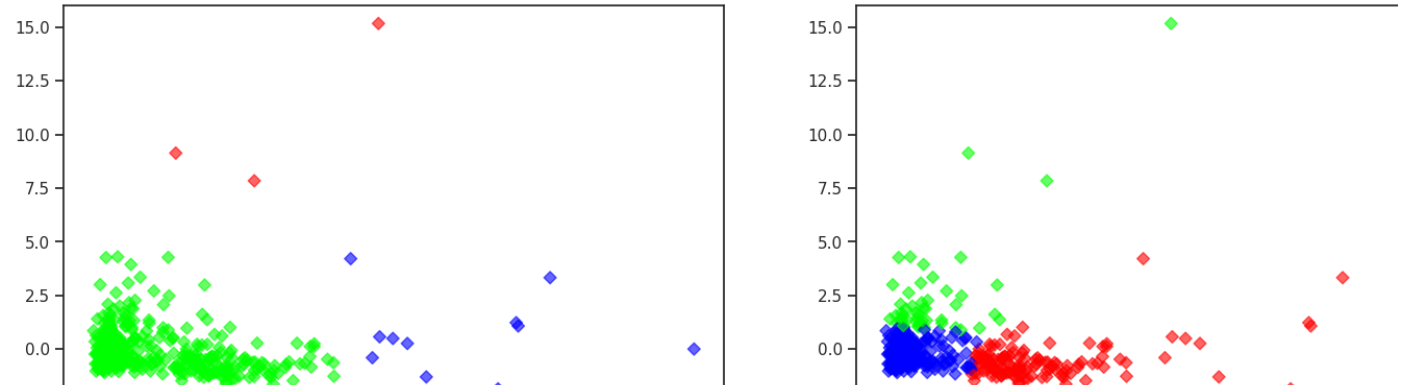
```
<matplotlib.collections.PathCollection at 0x7da47042c9a0>
```



Let's try other kind of clustering.

```
#Agglomerative Clustering
agc = AgglomerativeClustering(n_clusters=3,affinity='euclidean',linkage='ward')
y_agc_pred = agc.fit_predict(ps)
plt.figure(figsize =(18,5))

plt.subplot(1,2,1)
plt.scatter(ps[0], ps[1],c = y_agc_pred, s=80, cmap=cmap,alpha=0.6,marker='s')

plt.subplot(1,2,2)
dend=shc.dendrogram(shc.linkage(ps,method='ward') ,truncate_mode='level', p=3)
plt.show()
```

```
#Birch clustering
brc = Birch(branching_factor=500, n_clusters=3, threshold=1.5)
brc.fit(ps)
labels = brc.predict(ps)
plt.figure(figsize =(18,5))

plt.subplot(1,2,1)
plt.scatter(ps[0], ps[1], c=labels, cmap='brg',alpha=0.6,marker='D')

#MiniBatchKMeans
mb = MiniBatchKMeans(n_clusters=3, random_state=0)
mb.fit(ps)
labels = mb.predict(ps)

plt.subplot(1,2,2)
plt.scatter(ps[0], ps[1], c=labels, cmap='brg',alpha=0.6,marker='D')


plt.show()
```




```
end = time. time()
sec = (end - start)
print(f'Total time taken to complete the execution :{sec} seconds(s)')
```

```
Total time taken to complete the execution :337.60855317115784 seconds(s)
```