Experiment No. 5
Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset
Date of Performance:
Date of Submission:

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

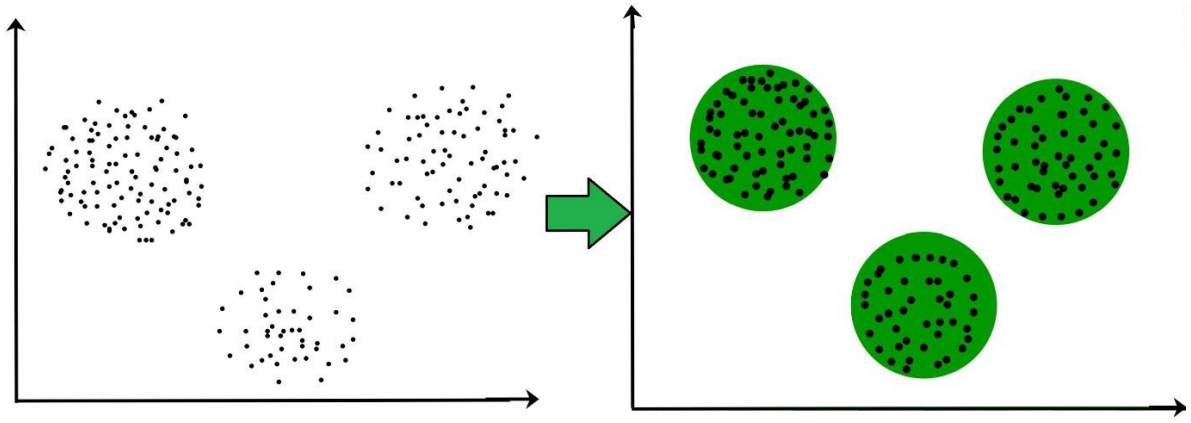
Objective: Able to perform various feature engineering tasks, apply Clustering Algorithm on the given dataset.

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:

Based on the visualization, comment on following:

1. How can you can make use of the clustered data?

   By examining the clusters, we can reach conclusions and take actions, such as Knowing Your Customers' Segments The development of more relevant marketing efforts that are tailored to the interests of each group is made possible by the use of clustered data, which makes it simpler to identify various client segments based on their purchase behaviors. Clustering supports inventory optimization by ensuring that the appropriate products are stocked in the appropriate quantities to suit the preferences of each cluster. Looking for New Markets Clustering can also discover these potential new markets or consumer segments by contrasting current clusters with potential new markets or segments. Delivery schedules and routes can be customized to the unique needs of each cluster to enhance supply chain operations.

2. How the different groups of customers, the ***customer segments***, may be affected differently by a specific delivery scheme?

   High-value customers are more likely to be sensitive to shipping options because they buy things frequently. They are inclined to esteem and be willing to pay for top-notch delivery services. Contrarily, low-value clients seek cost savings, are typically less affected by delivery speed, and may not demand expedited options. Customers may encounter variable levels of service quality and delivery speed depending on their location. Remote or underserved locations may experience longer delivery delays and may be particularly impacted by any changes to distribution strategies. Loyal consumers who have built a relationship of trust with a business could be excused from delivery issues on occasion. New consumers might be more sensitive to delivery experiences, and a bad one might discourage them from making more purchases in the future.

```
import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings('ignore')

import math
import random

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

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

from scipy.stats.mstats import winsorize

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from scipy.stats import boxcox, probplot, norm, shapiro
from sklearn.decomposition import PCA

from sklearn.model_selection import KFold

from sklearn.preprocessing import LabelEncoder

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

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

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

import time
start = time. time()


df = pd.read_csv('/content/Wholesale customers data.csv')
df_MV = df.copy()
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

```
df_MV.shape
```

```
(440, 8)
```

```
df_MV.isnull().sum()
```

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

```
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
```

```
column_list = ['Channel','Region']
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]
```

```
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
```

```
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
```

```
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
```

```
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

```
df_MVal = df_MV.drop(['Channel','Region'], axis=1)
df_MV.groupby(['Channel', 'Region']).agg(['median','mean']).round(1)
```

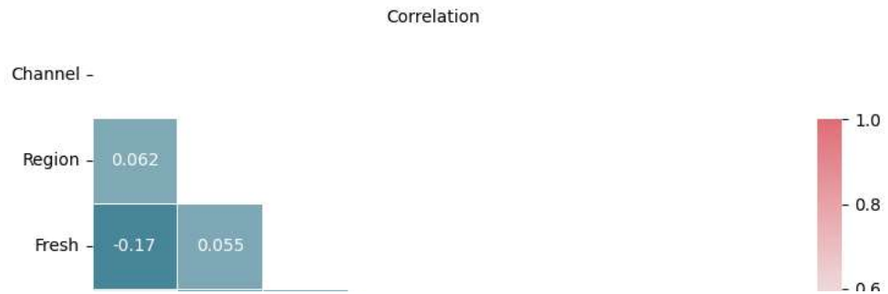
		Fresh		Milk		Grocery		Frozen		Detergents_Paper		Delicassen	
		median	mean	median	mean	median	mean	median	mean	median	mean	median	mean
Channel	Region												
1.0	1.0	8885.0	13059.7	2179.0	3638.7	2501.0	3855.4	2077.0	2951.2	405.5	813.0	693.0	1092.1
	2.0	9790.0	11892.7	1560.5	2334.5	3315.0	4388.6	2679.0	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

```
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
```

```
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)
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')
```




```
X = df.drop('Channel', axis=1)
y = df['Channel']

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

```
mutual_info = mutual_info_classif(X, y)
mutual_data = pd.Series(mutual_info, index = X.columns)
mutual_data.sort_values(ascending=False)
```

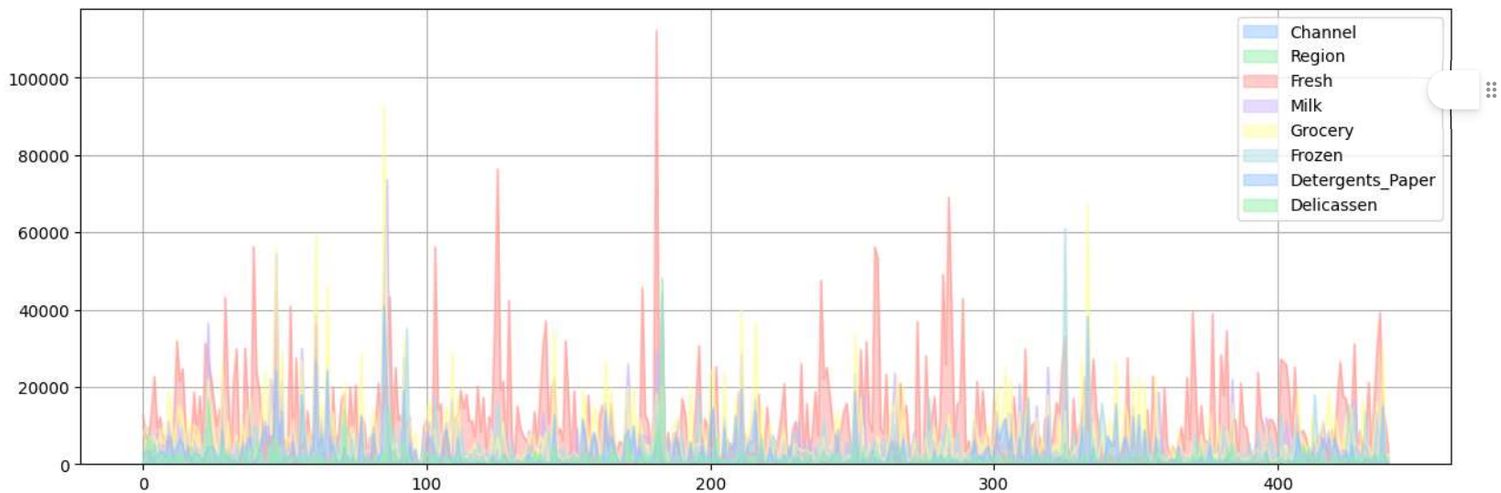
```
Grocery             0.237157
Detergents_Paper    0.224623
Milk                0.115816
Frozen              0.069639
Region              0.004907
Fresh               0.000816
Delicassen          0.000000
dtype: float64
```

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

```
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 = pd.DataFrame(trans_feat, columns=columns_)

Y_label = Y

X_feature.columns
```

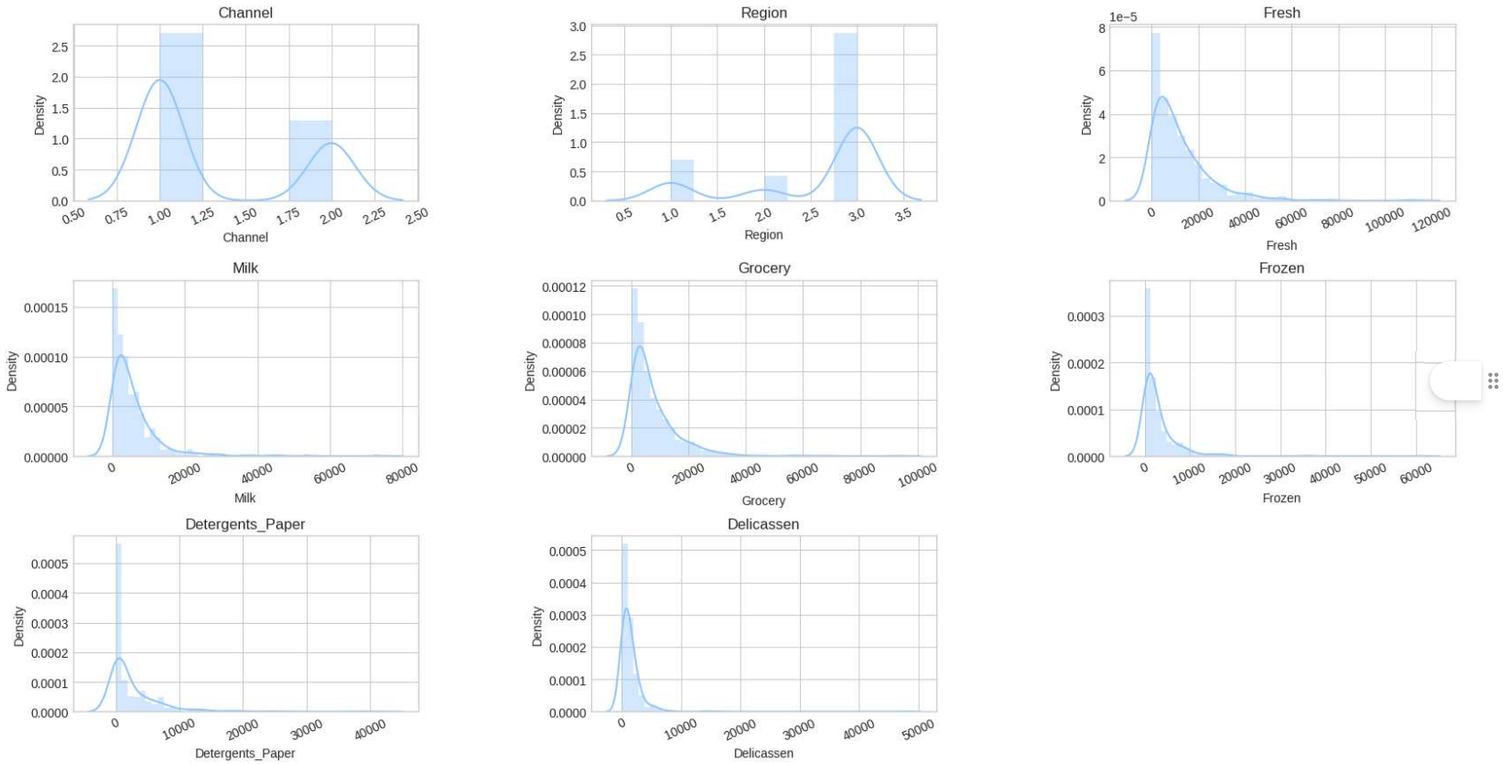
```
Index(['Region', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper'], dtype='object')
```

```
df.plot.area(stacked=False,figsize=(15,5))
pylab.grid(); pylab.show()
```




```
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)
```




```
df.agg(['median','mean','std']).round(2)
```

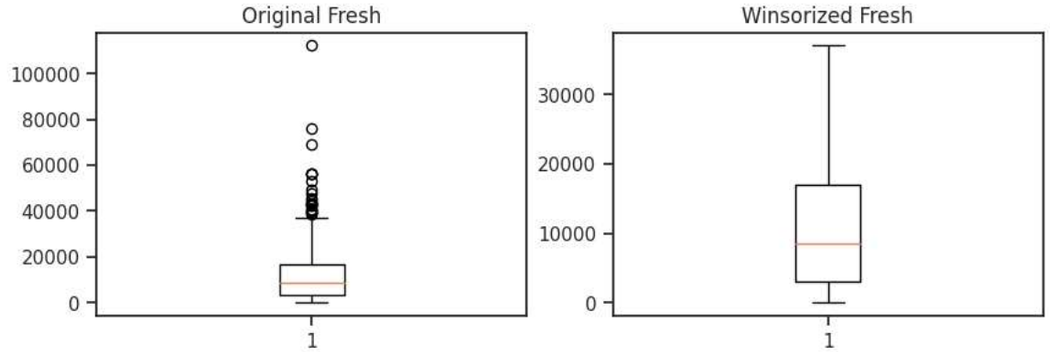
```
             Channel  Region     Fresh     Milk  Grocery  Frozen  Detergents_Paper  Delicassen
```

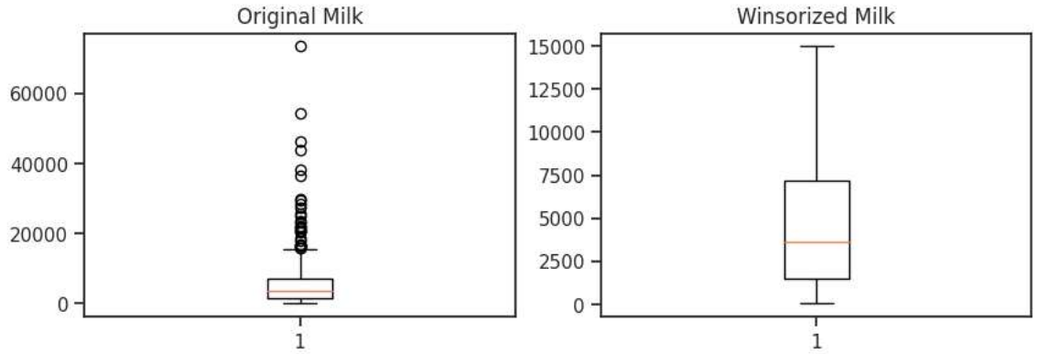
```
num_col = df.columns.tolist()
def outlier_count(col, data=df):
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    min_val = Q1 - (IQR*1.5)
    max_val = Q3 + (IQR*1.5)
    outlier_count = len(np.where((data[col] > max_val) | (data[col] < min_val))[0])
    outlier_percent = round(outlier_count/len(data[col])*100, 2)
    print('{:<20} {:<20} {:.2f}%'.format(col,outlier_count,outlier_percent))

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)
```

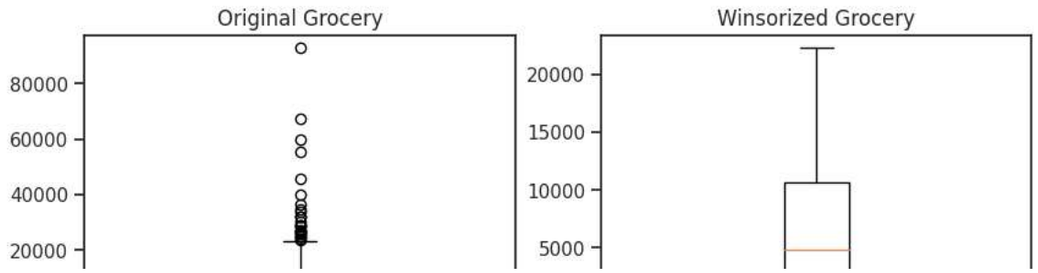
```
 ******************** Outliers ********************

 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%
```

```
def winsor(col, lower_limit=0, upper_limit=0, show_plot=True):

    winsor_data = winsorize(df[col], limits=(lower_limit, upper_limit))

    winsor_dict[col] = winsor_data

    if show_plot == True:
        plt.figure(figsize=(10,3))

        plt.subplot(121)
        plt.boxplot(df[col])
        plt.title('Original {}'.format(col))

        plt.subplot(122)
        plt.boxplot(winsor_data)

        plt.title('Winsorized {}'.format(col))
        plt.show()


winsor_dict = {}

winsor(num_col[2], upper_limit = 0.0455, show_plot=True)

winsor(num_col[3], upper_limit = 0.067, show_plot=True)

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

```
winsor(num_col[4], upper_limit = 0.0977, show_plot=True)

winsor(num_col[4], upper_limit = 0.0682, show_plot=True)

winsor(num_col[4], upper_limit = 0.0614, show_plot=True)
```



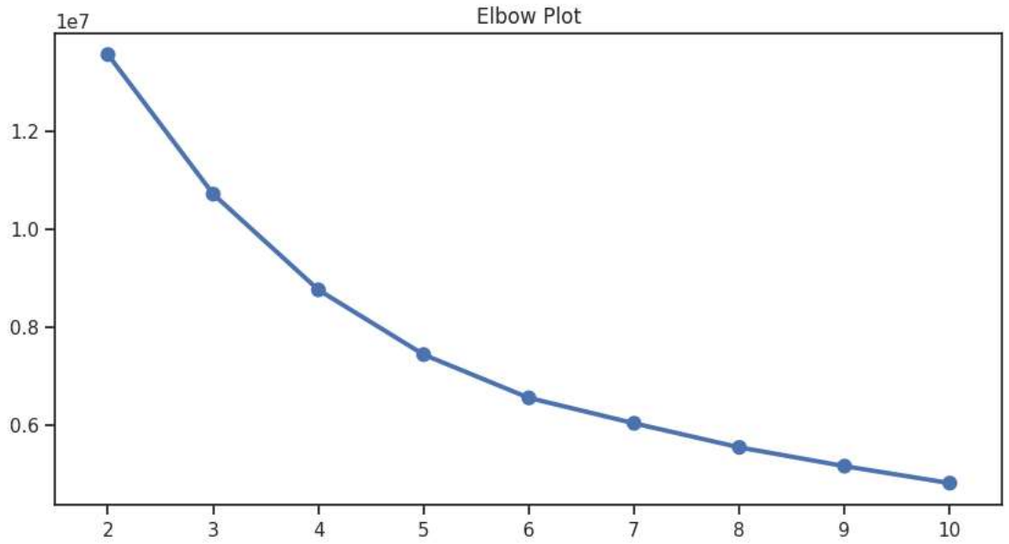





```
sc=StandardScaler()
scaled_data=sc.fit_transform(df)

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

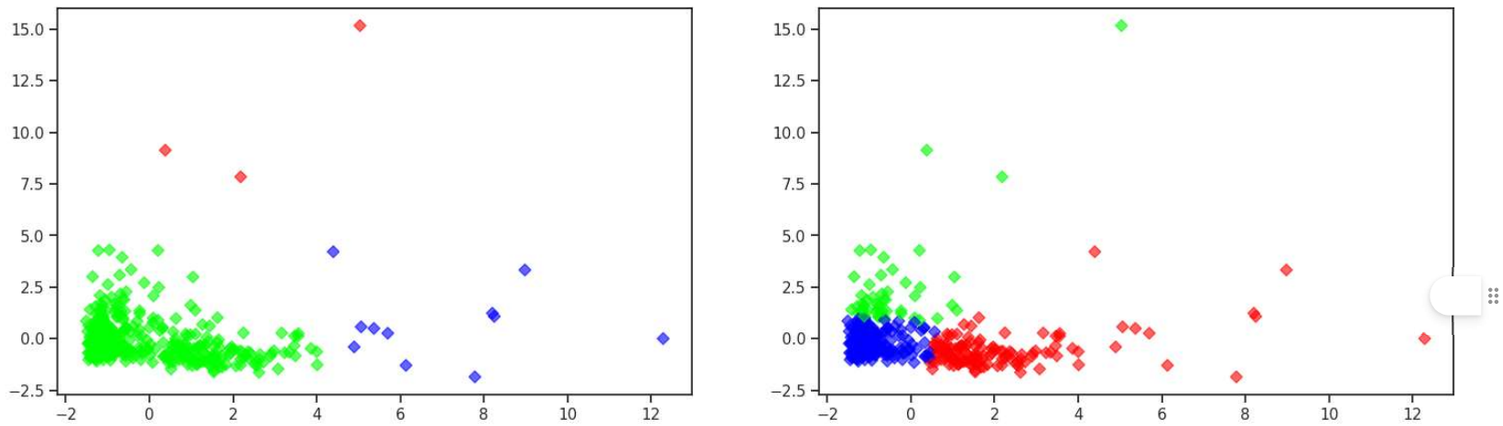
```
PCA_train = PCA(2).fit_transform(scaled_data)
ps = pd.DataFrame(PCA_train)

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()
```




```
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')

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)
```