



Experiment No. 5
Apply appropriate Unsupervised Learning Technique on the Wholesale Customers Dataset
Date of Performance:14-09-23
Date of Submission:05-10-23



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

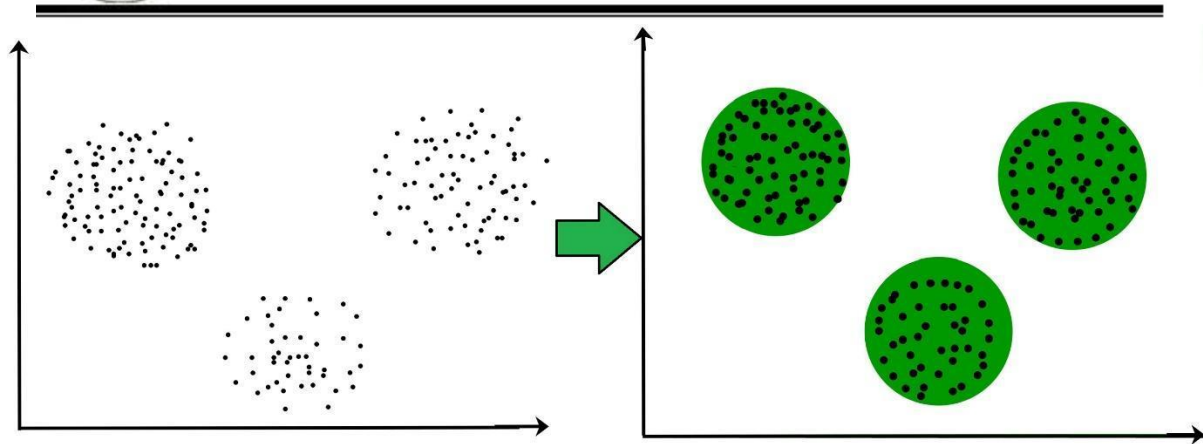
Objective: Able to perform various feature engineering tasks, apply Clustering Algorithm on the given dataset.

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 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 make use of the clustered data?

Customer Segmentation: Use clustered data to understand customer groups and tailor marketing campaigns to their preferences.

Personalized Recommendations: Identify products frequently purchased together within clusters for personalized product suggestions.

Inventory Optimization: Optimize inventory based on cluster preferences for efficient stock management.

Supply Chain Efficiency: Customize supply chain operations to meet each cluster's unique needs.

Customer Retention: Develop strategies based on cluster characteristics to boost customer loyalty.

Market Expansion: Discover new markets or similar customer segments through clustering for expansion opportunities.

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

Premium Delivery: High-value customers who prioritize convenience and are willing to pay more for faster delivery.

Budget Shoppers: Price-sensitive customers who prefer cost-effective or standard delivery options, including free choices.

Bulk Buyers: Customers who like purchasing in larger quantities and may benefit from bulk order discounts or specialized delivery options.


Frequent Shoppers: Customers who shop often and can enjoy subscription or loyalty-based delivery schemes to encourage repeat purchases and loyalty.



Vidyavardhini's College of Engineering & Technology
Department of Computer Engineering

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
for dirname, __, filenames in os.walk('/content/Wholesale customers data.csv'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import pandas as pd
# Define a function to load the data
def load_data(path):
    try:
        df = pd.read_csv(path)
        print("Data loaded successfully!")
        return df
    except Exception as e:
        print(f"An error occurred: {e}")
        return None
# Path to the data file
path = '/content/Wholesale customers data.csv'
# Load the data
df = load_data(path)
# Display the first few rows of the DataFrame
print(df.head())
```

 Data loaded successfully!

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

```
print("Column names:")
print(df.columns)

Column names:
Index(['Channel', 'Region', 'Fresh', 'Milk', 'Grocery', 'Frozen',
       'Detergents_Paper', 'Delicassen'],
      dtype='object')
```

```
# Print the data types of each column
print("Data types:")
print(df.dtypes)
```

```
Data types:
Channel          int64
Region           int64
Fresh            int64
Milk             int64
Grocery          int64
Frozen           int64
Detergents_Paper int64
Delicassen       int64
dtype: object
```

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

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

```
import matplotlib.pyplot as plt
import seaborn as sns
# Check descriptive statistics
print("Descriptive Statistics:")
print(df.describe())
# Check for duplicates
print("Number of duplicate rows: ", df.duplicated().sum())
```

Descriptive Statistics:

	Channel	Region	Fresh	Milk	Grocery \
count	440.000000	440.000000	440.000000	440.000000	440.000000
mean	1.322727	2.543182	12000.297727	5796.265909	7951.277273
std	0.468052	0.774272	12647.328865	7380.377175	9503.162829
min	1.000000	1.000000	3.000000	55.000000	3.000000
25%	1.000000	2.000000	3127.750000	1533.000000	2153.000000
50%	1.000000	3.000000	8504.000000	3627.000000	4755.500000
75%	2.000000	3.000000	16933.750000	7190.250000	10655.750000
max	2.000000	3.000000	112151.000000	73498.000000	92780.000000

	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000
mean	3071.931818	2881.493182	1524.870455
std	4854.673333	4767.854448	2820.105937
min	25.000000	3.000000	3.000000
25%	742.250000	256.750000	408.250000
50%	1526.000000	816.500000	965.500000
75%	3554.250000	3922.000000	1820.250000
max	60869.000000	40827.000000	47943.000000

Number of duplicate rows: 0

Distribution plots for each feature

for column in df.columns:

plt.figure(figsize=(6, 4))

sns.histplot(df[column], bins=30, kde=True)

plt.title(f'Distribution of {column}')

plt.show()

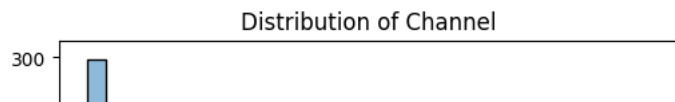
Heatmap for correlation between variables

plt.figure(figsize=(10, 8))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm', center=0)

plt.title('Correlation Heatmap')

plt.show()



```
# checking for outliers
import seaborn as sns
import matplotlib.pyplot as plt
# Draw boxplots for all features
for column in df.columns:
    plt.figure(figsize=(6, 4))
    sns.boxplot(df[column])
    plt.title(f'Boxplot of {column}')
    plt.show()

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

# Detect and print number of outliers for each feature
for column in df.columns:
    outliers = detect_outliers(df, column)
    print(f'Number of outliers in {column}: {len(outliers)}')
```

```
def handle_outliers(dataframe, column):
    Q1 = dataframe[column].quantile(0.25)
```



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

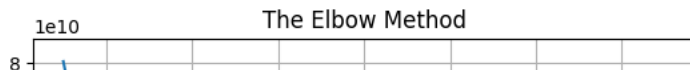
# Detect and print number of outliers for each feature
for column in df.columns:
    outliers = detect_outliers(df, column)
    print(f'Number of outliers in {column}: {len(outliers)}')

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

# Check descriptive statistics
print("Descriptive Statistics:")
print(df.describe())
# Check for duplicates
print("Number of duplicate rows: ", df.duplicated().sum())
# Distribution plots for each feature
for column in df.columns:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[column], bins=30, kde=True)
    plt.title(f'Distribution of {column}')
    plt.show()

# Heatmap for correlation between variables
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Heatmap')
plt.show()
```

<https://colab.research.google.com/drive/1iQs5xNOqwMM5F1OGNQXM3CIZ1VWnn1cl?authuser=2#printMode=true> 6/9



```
from sklearn.cluster import KMeans
# Build the model
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
kmeans.fit(df)
# Get cluster labels
cluster_labels = kmeans.labels_
# Add cluster labels to your original dataframe
df['Cluster'] = cluster_labels
print(df.head())
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 1 in the future. You should set it to the number of clusters you expect in the data.
warnings.warn(
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	\
0	2	3	12669.0	9656.0	7561.0	214.0	2674.0	
1	2	3	7057.0	9810.0	9568.0	1762.0	3293.0	
2	2	3	6353.0	8808.0	7684.0	2405.0	3516.0	
3	1	3	13265.0	1196.0	4221.0	6404.0	507.0	
4	2	3	22615.0	5410.0	7198.0	3915.0	1777.0	

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

```
# Add cluster labels to the DataFrame
df['Cluster'] = kmeans.labels_
# Check the size of each cluster
print("Cluster Sizes:\n", df['Cluster'].value_counts())
# Check the characteristics of each cluster
for i in range(4):
    print("\nCluster ", i)
    print(df[df['Cluster'] == i].describe())
```

Cluster Sizes:

```
0    227
1    112
2    101
```

Name: Cluster, dtype: int64

Cluster	0	
count	227.000000	227.000000
mean	1.132159	2.528634
std	0.339412	0.788647
min	1.000000	1.000000
25%	1.000000	2.000000
50%	1.000000	3.000000
75%	1.000000	3.000000
max	2.000000	3.000000

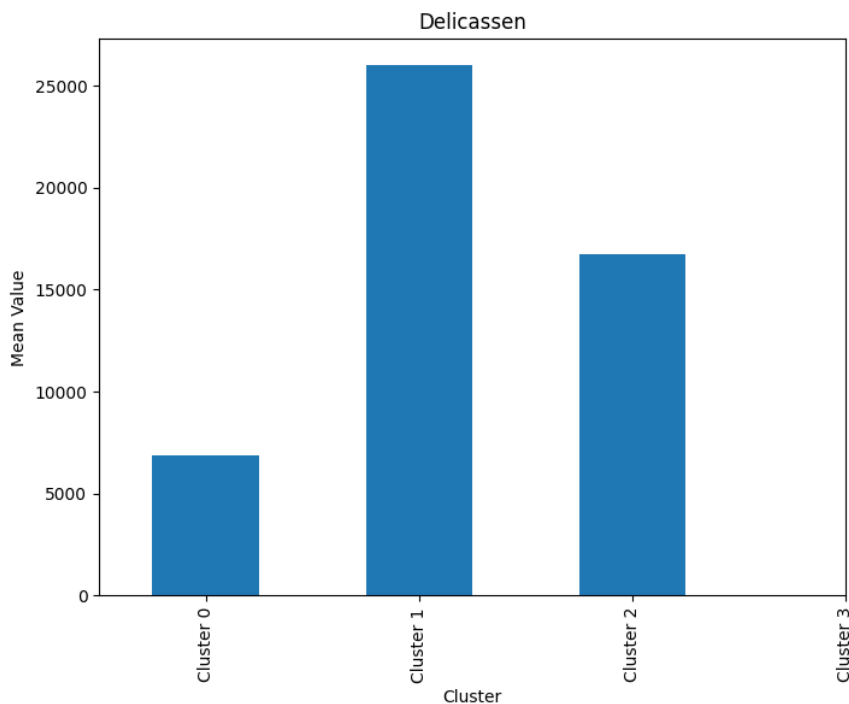
	Frozen	Detergents_Paper	Delicassen	Cluster
count	227.000000	227.000000	227.000000	227.0
mean	2326.412996	984.233480	963.896476	0.0
std	2264.692928	1235.547191	893.981219	0.0
min	47.000000	3.000000	3.000000	0.0
25%	663.500000	194.500000	320.500000	0.0
50%	1439.000000	402.000000	686.000000	0.0
75%	3283.500000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

Cluster	1	
count	112.000000	112.000000
mean	1.214286	2.598214
std	0.412170	0.740828
min	1.000000	1.000000
25%	1.000000	2.750000
50%	1.000000	3.000000
75%	1.000000	3.000000
max	2.000000	3.000000

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

Cluster	2	Channel	Region	Fresh	Milk	Grocery
count	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000
mean	1.871287	2.514851	5190.811881	10106.875000	16743.814356	
std	0.336552	0.782481	5053.693043	4022.429078	5021.119664	
min	1.000000	1.000000	18.000000	1266.000000	8852.000000	
25%	2.000000	2.000000	1210.000000	7097.000000	11924.000000	
50%	2.000000	3.000000	3830.000000	9933.000000	15541.000000	
75%	2.000000	3.000000	7362.000000	13316.000000	22182.000000	
max	2.000000	3.000000	22925.000000	15676.000000	23409.000000	

```
# Calculate the mean values for each feature per cluster
cluster_means = df.groupby('Cluster').mean()
# Transpose the DataFrame so that the features are the rows (this will make plotting easier)
cluster_means = cluster_means.transpose()
# Create bar plot for each feature
for feature in cluster_means.index:
    cluster_means.loc[feature].plot(kind='bar', figsize=(8,6))
    plt.title(feature)
    plt.ylabel('Mean Value')
    plt.xticks(ticks=range(4), labels=['Cluster 0', 'Cluster 1', 'Cluster 2', 'Cluster 3'])
```



```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
# Apply PCA and fit the features selected
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(df.drop('Cluster', axis=1))
# Create a DataFrame with the two components
PCA_components = pd.DataFrame(principalComponents, columns=['Principal Component 1', 'Principal Component 2'])
# Concatenate the clusters labels to the DataFrame
PCA_components['Cluster'] = df['Cluster']
# Plot the clustered dataset
plt.figure(figsize=(8,6))
plt.scatter(PCA_components['Principal Component 1'], PCA_components['Principal Component 2'], c=PCA_components['Cluster'])
```

