Table of Contents

Introduction:

In today's highly competitive business landscape, understanding and catering to customer needs has become a key priority for organizations. To achieve this, businesses need to delve deep into their customer data and extract meaningful insights. One powerful approach to accomplishing this is by applying clustering, time series analysis, and classification techniques to a dataset of customer invoices. These techniques enable businesses to identify customer segments, uncover trends in customer behavior, and predict customer churn. By harnessing the power of these analytical tools, businesses can enhance their marketing campaigns, optimize inventory management, and streamline staffing, leading to improved customer satisfaction and overall business performance.

Abstract:

This project explores the use of clustering, time series analysis, and classification on a dataset of customer invoices. These techniques can be used to group similar customers together, identify trends in customer behaviour, and classify customers based on their characteristics. This information can be used to improve marketing campaigns, prevent customer churn, and make better decisions about inventory, staffing, and marketing. This would allow you to target different marketing campaigns to different groups of customers. Time series analysis could be used to identify trends in customer behaviour. This would allow you to predict when customers are likely to buy and how much they are likely to spend. Classification could be used to predict whether a customer is likely to churn.

The following are benefits of using clustering, time series analysis, and classification on a dataset of customer invoices:

- Improved marketing campaigns: By understanding the different types of customers that you have, you can create marketing campaigns that are more likely to appeal to each group.
- Better inventory management: By understanding the demand for different products, you can ensure that you have the right amount of inventory on hand.
- More efficient staffing: By understanding the peak times of demand, you can ensure that you have the right number of staff on hand to meet customer needs.

Overall, the use of clustering, time series analysis, and classification on a dataset of customer invoices can provide a number of benefits for businesses. By understanding their customers better, businesses can create more effective

marketing campaigns, reduce customer churn, improve inventory management, and more.

Objective:

The primary objective of this project is to explore the potential of clustering, time series analysis, and classification in the context of customer invoice data. By analyzing this dataset, we aim to achieve the following goals:

- Customer Segmentation: Utilize clustering techniques to group customers with similar purchasing patterns, preferences, and characteristics. This segmentation will enable businesses to tailor their marketing efforts, providing personalized campaigns that resonate with each customer segment. By understanding the unique needs of different customer groups, businesses can maximize their marketing ROI and enhance customer engagement.
- Trend Identification: Apply time series analysis to uncover temporal patterns and trends in customer behavior. By identifying recurring patterns, seasonal fluctuations, and changes in purchasing habits, businesses can anticipate demand fluctuations, optimize inventory levels, and make informed decisions regarding production and supply chain management.
- Enhanced Decision-Making: By gaining a comprehensive understanding of customer segments, trends, and churn predictions, businesses can make data-driven decisions related to inventory management, staffing, and resource allocation. This will lead to optimized operational efficiency, cost savings, and improved customer satisfaction.

Code & Output:

Importing necessary Libraries:

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings("ignore")
        from sklearn.impute import SimpleImputer
```

Dataset:

```
df = pd.read_excel("G:\DCS-SEM 4\Predicitive analysis\Dataset\Superstore.xls")
df_cat = df['Category']
df.head()
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code	Region	Product ID	Category	Sub-Category	Product Name
0	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-BO-10001798	Furniture	Bookcases	Bush Somerset Collection Bookcase
1	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-CH-10000454	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs,...
2	3	CA-2016-138688	2016-06-12	2016-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	90036	West	OFF-LA-10000240	Office Supplies	Labels	Self-Adhesive Address Labels for Typewriters b...
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	FUR-TA-10000577	Furniture	Tables	Bretford CR4500 Series Slim Rectangular Table
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	OFF-ST-10000760	Office Supplies	Storage	Eldon Fold 'N Roll Cart System

Dataset Preprocessing:

```
In [3]: df.dtypes

Out[3]: Row ID                    int64
        Order ID                 object
        Order Date       datetime64[ns]
        Ship Date        datetime64[ns]
        Ship Mode                object
        Customer ID              object
        Customer Name            object
        Segment                  object
        Country                  object
        City                     object
        State                    object
        Postal Code               int64
        Region                   object
        Product ID               object
        Category                 object
        Sub-Category             object
        Product Name             object
        Sales                   float64
        Quantity                  int64
        Discount                float64
        Profit                  float64
        dtype: object
```

In [4]:
```
df.shape
```
Out[4]:
```
(9994, 21)
```

In [5]:
```
obj = [x for x in df.columns if df[x].dtype == 'object']
```

In [6]:
```
from sklearn import preprocessing
encoder = preprocessing.LabelEncoder()
for i in df.columns:
    if i in obj:
        df[i] = encoder.fit_transform(df[i])

df.head()
```
Out[6]:

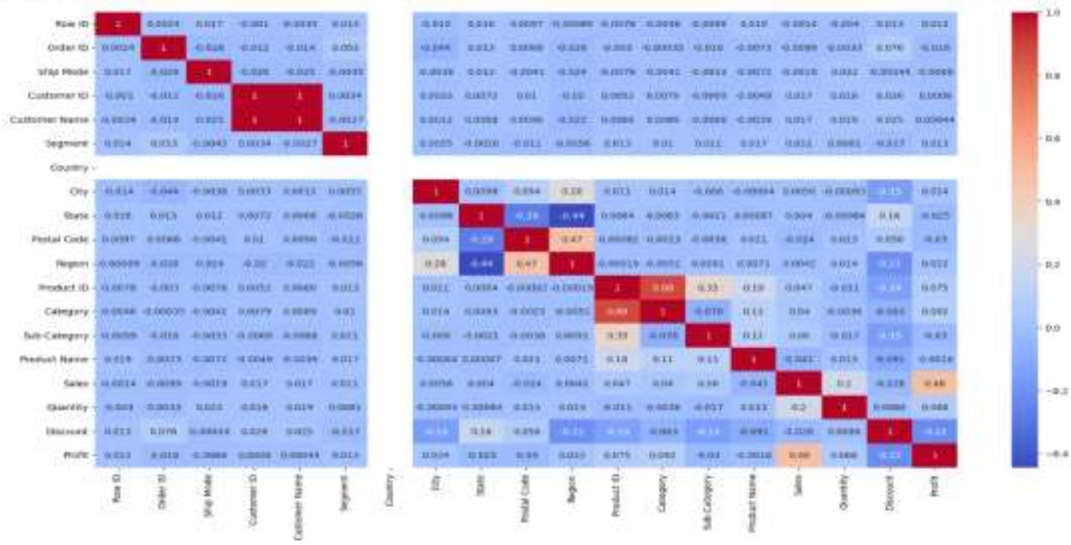
	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales	Quant
0	1	2500	2016-11-08	2016-11-11	2	143	166	0	0	194	...	42420	2	12	0	4	386	261.9600	
1	2	2500	2016-11-08	2016-11-11	2	143	166	0	0	194	...	42420	2	55	0	5	839	731.9400	
2	3	2296	2016-06-12	2016-06-16	2	237	201	1	0	266	...	90036	3	946	1	10	1433	14.6200	
3	4	4372	2015-10-11	2015-10-18	3	705	687	0	0	153	...	33311	2	319	0	16	366	957.5775	
4	5	4372	2015-10-11	2015-10-18	3	705	687	0	0	153	...	33311	2	1316	1	14	573	22.3680	

5 rows × 21 columns

In [7]:
```
df.isnull().sum()
```
Out[7]:
```
Row ID           0
Order ID         0
Order Date       0
Ship Date        0
Ship Mode        0
Customer ID      0
Customer Name    0
Segment          0
Country          0
City             0
State            0
Postal Code      0
Region           0
Product ID       0
Category         0
Sub-Category     0
Product Name     0
Sales            0
Quantity         0
Discount         0
Profit           0
dtype: int64
```

In [8]:
```
corr = df.corr()
```

In [9]:
```
plt.figure(figsize = (20 ,10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```



Feature Selection

In [10]:
```
drop = ['Row ID' , 'Order ID' , 'Ship Mode' ,
         'Customer Name' , 'Segment' , 'Country' ,
         'City' , 'State' , 'Postal Code' , 'Region' , 'Sub-Category']
df = df.drop(drop , axis = 1)
df.head()
```

Out[10]:

	Order Date	Ship Date	Customer ID	Product ID	Category	Product Name	Sales	Quantity	Discount	Profit
0	2016-11-08	2016-11-11	143	12	0	386	261.9600	2	0.00	41.9136
1	2016-11-08	2016-11-11	143	55	0	839	731.9400	3	0.00	219.5820
2	2016-06-12	2016-06-16	237	946	1	1433	14.6200	2	0.00	6.8714
3	2015-10-11	2015-10-18	705	319	0	366	957.5775	5	0.45	-383.0310
4	2015-10-11	2015-10-18	705	1316	1	573	22.3680	2	0.20	2.5164

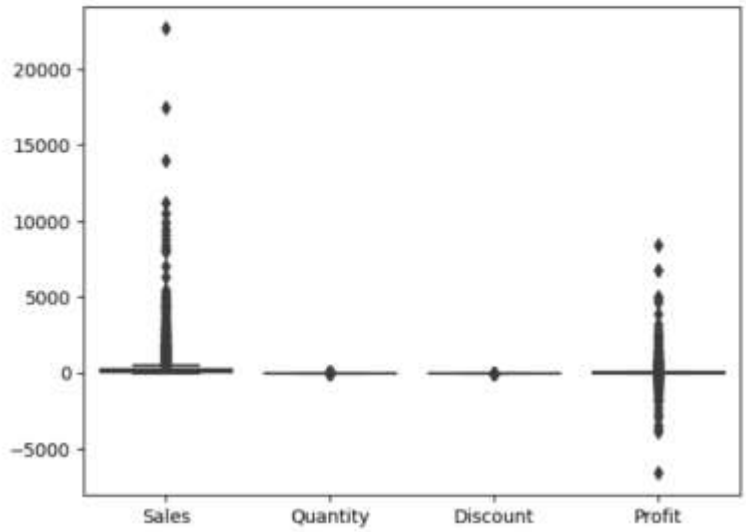
In [11]:
```
df.describe()
```

Out[11]:

	Customer ID	Product ID	Category	Product Name	Sales	Quantity	Discount	Profit
count	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000
mean	400.460376	898.472383	0.972584	922.324795	229.858001	3.789574	0.156203	28.656896
std	228.585576	526.708445	0.629544	531.515975	623.245101	2.225110	0.206452	234.260108
min	0.000000	0.000000	0.000000	0.000000	0.444000	1.000000	0.000000	-6599.978000
25%	205.250000	453.000000	1.000000	474.250000	17.280000	2.000000	0.000000	1.728750
50%	405.500000	863.000000	1.000000	907.000000	54.490000	3.000000	0.200000	8.666500
75%	602.000000	1347.000000	1.000000	1390.000000	209.940000	5.000000	0.200000	29.364000
max	792.000000	1861.000000	2.000000	1849.000000	22638.480000	14.000000	0.800000	8399.976000

Outlier Analysis:

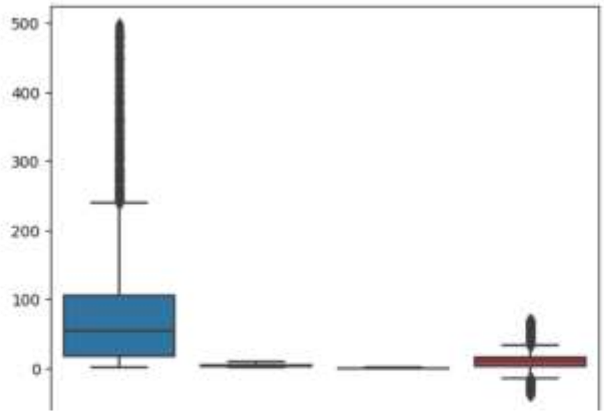
In [14]:
```
sns.boxplot(df[["Sales", "Quantity", "Discount", "Profit"]])
plt.show()
```



In [15]:
```
train = df[["Sales", "Quantity", "Discount", "Profit"]]
def impute_outliers(data, column, factor):
    q1 = data[column].quantile(0.25)
    q3 = data[column].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - factor * iqr
    upper_bound = q3 + factor * iqr

    data_copy = data.copy()
    data_copy[column] = np.where(data_copy[column] < lower_bound, np.nan, data_copy[column])
    data_copy[column] = np.where(data_copy[column] > upper_bound, np.nan, data_copy[column])
    imputer = SimpleImputer(strategy="mean")
    data_imputed = imputer.fit_transform(data_copy)

    return pd.DataFrame(data_imputed, columns=data.columns)
for column in ["Sales", "Quantity", "Discount", "Profit"]:
    train = impute_outliers(train, column, 1.5)
df[["Sales", "Quantity", "Discount", "Profit"]] = train
sns.boxplot(train)
plt.show()
```



Time series:

ARIMA Forecasting

In [17]:
```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error, mean_squared_error
import warnings
warnings.filterwarnings("ignore")
```

Model Fitting and evaluation

In [18]:
```
arima_model = ARIMA(sales_by_year["Sales"], order=(1, 1, 1))
arima_model_fit = arima_model.fit()
```

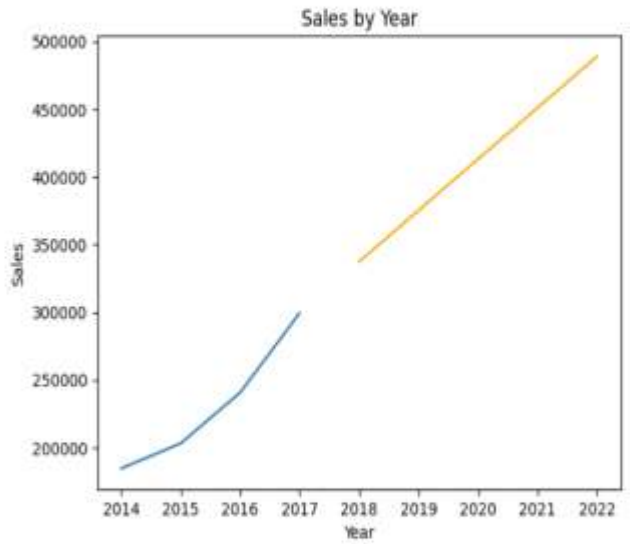
Forecasting

In [19]:
```
forecast = arima_model_fit.forecast(steps = 5).round(2)
print(forecast)
```

```
4    337297.66
5    375207.10
6    413114.25
7    451019.13
8    488921.73
Name: predicted_mean, dtype: float64
```

Plotting of Actual data with Forecasted data

In [20]:
```
plt.plot(sales_by_year["Order Year"], sales_by_year["Sales"])
plt.plot([2018 , 2019 , 2020 , 2021 , 2022],forecast, color = "orange")
plt.xlabel("Year")
plt.ylabel("Sales")
plt.title("Sales by Year")
plt.show()
```



Forecasting for each Category

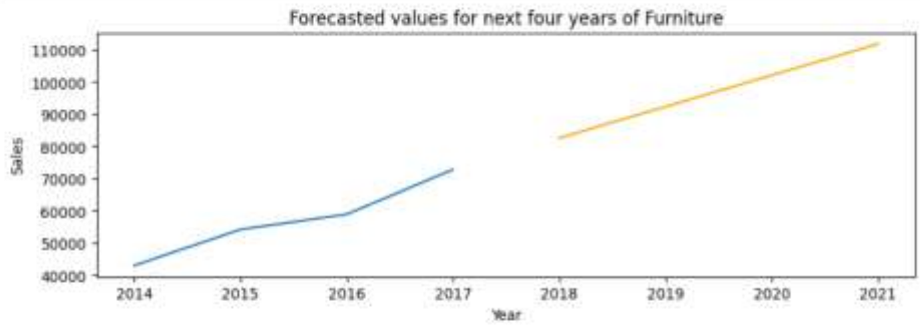
In [21]:
```
x = df.groupby([ 'Category' , 'Order Year'])['Sales'].sum().reset_index().pivot(index = 'Order Year' ,
                                                                              columns = 'Category' , values = "Sales")
x.head()
```

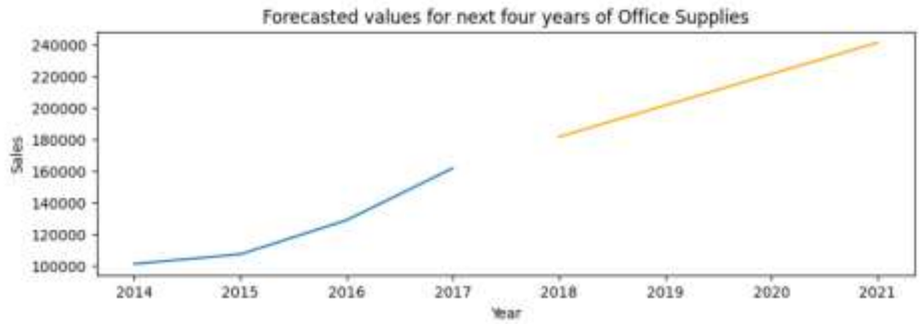
Out[21]:

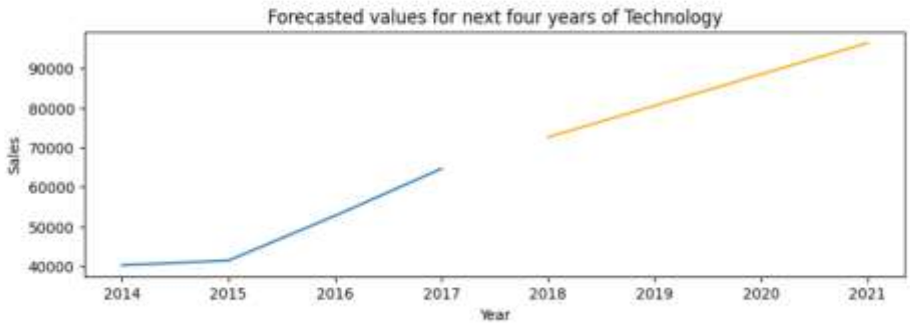
Category / Order Year	Furniture	Office Supplies	Technology
2014	43028.099412	101398.051450	40193.214155
2015	54214.385590	107482.871647	41380.777012
2016	58929.683552	129057.724561	52706.806171
2017	72848.195671	161924.353365	64615.391020

In [22]:
```
def forecasting(x):
    arima_model = ARIMA(x , order = (1,1,1))
    arima_model_fit= arima_model.fit()
    forecast_values = arima_model_fit.forecast(steps = 4)
    return forecast_values.round(2)

for i in x.columns:
    values = forecasting(x[i])
    plt.figure(figsize = (10, 3))
    plt.plot(x.index,x[i])
    plt.plot([2018 , 2019 , 2020 , 2021] , values , color = 'orange')
    plt.xlabel('Year')
    plt.ylabel('Sales')
    plt.title(f'Forecasted values for next four years of {i}')
```







Clustering:

Clustering

In [23]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

In [24]:
```
df_cluster = df.groupby("Customer ID").sum()
train = df_cluster[["Sales", "Quantity", "Discount", "Profit"]]
```

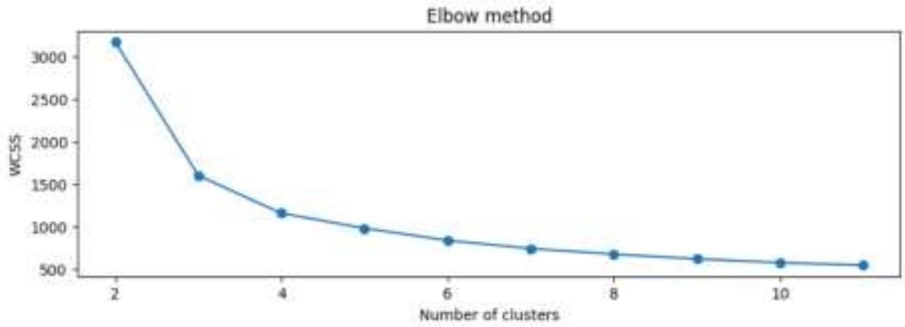
Optimal K size

Elbow Method

In [25]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(train)

#Elbow Method
wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42 , n_init = 10)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

plt.figure(figsize = (10 ,3))
plt.plot(range(2, 12), wcss, marker="o")
plt.xlabel("Number of clusters")
plt.ylabel("WCSS")
plt.title("Elbow method")
plt.show()
```



Inference:

We can see that from 2 - 3 the value of WCSS decreasing fast and in a straight but after 2 there is a slow drop and decreasing exponentially so K = 3

K - Means Algorithim

In [26]:
```python
k_opt = 3
kmeans_opt = KMeans(n_clusters=k_opt, random_state=42 , n_init = 10)
kmeans_opt.fit(train)

# Assign the cluster labels to the data set
df_cluster["Cluster"] = kmeans_opt.labels_
df_cluster.head()
```

Out[26]:

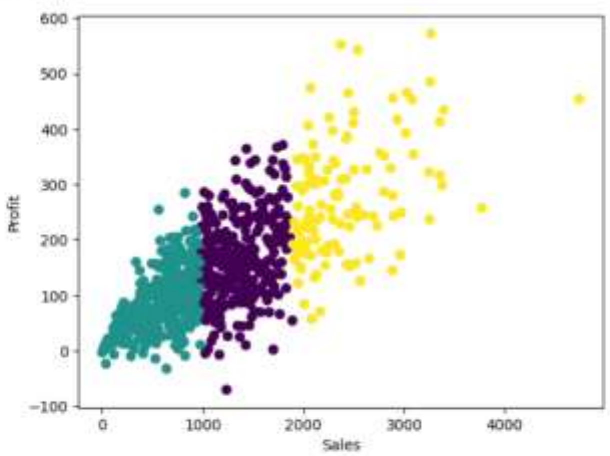
Customer ID	Product ID	Product Name	Sales	Quantity	Discount	Profit	Order Month	Order Year	Cluster
0	10675	8722	1145.629496	35.000000	0.903468	121.282406	58	22169	0
1	16095	13133	914.829496	45.000000	1.000000	217.662303	128	30237	1
2	12696	14127	1790.632000	36.000000	0.200000	195.867206	84	24191	0
3	13339	17334	1488.231473	70.654077	1.600000	240.832309	162	38274	0
4	5041	5259	831.264000	21.000000	0.103468	82.141903	33	12088	1

In [27]:
```python
df_cluster[["Sales", "Quantity", "Discount", "Profit" , "Cluster"]].groupby("Cluster").mean(numeric_only = True).round(2)
```

Out[27]:

Cluster	Sales	Quantity	Discount	Profit
0	1367.79	51.86	1.50	164.51
1	594.49	29.44	0.83	87.28
2	2399.70	81.18	2.20	277.66

In [28]:
```python
import matplotlib.pyplot as plt
plt.scatter(df_cluster['Sales'] , df_cluster['Profit'] , c=df_cluster['Cluster'])
plt.xlabel('Sales')
plt.ylabel('Profit')
plt.show()
```



Inference:

- Cluster 0 has an average sales value of 1367.79 , an average quantity of 51.86, an average discount of 1.50 and an average profit of 164.51.
- Cluster 1 has an average sales value of 594.49, an average quantity of 29.44 , an average discount of 29.44 and an average profit of 87.26.
- Cluster 2 has an average sales value of 2399.70 , an average quantity of 81.18, an average discount of 2.20 and an average profit of 277.66.

Conclusion:

Based on these values, Cluster 2 has the highest mean values for sales, quantity, discount and profit. This suggests that customers in Cluster 2 are generating more sales and profit for the supermarket compared to customers in Clusters 0 and 1.

Overall, this analysis suggests that customers in Cluster 2 may be more valuable to the supermarket in terms of generating sales and profit. It may be worth exploring further to understand what differentiates customers in Cluster 0 from those in Clusters 0 and 1 and how the supermarket can attract more customers like those in Cluster 2.

Classification:

Classification

In [29]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
```

Fitting and Evaluation

In [30]:
```
X = df_cluster[["Sales" ,  "Quantity", "Discount", "Profit"]]
y = df_cluster["Cluster"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
log_reg = LogisticRegression(max_iter = 150)
log_reg.fit(X_train, y_train)

#Predictions
y_pred = log_reg.predict(X_test)

#Evaluation
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[57  2  3]
 [ 0 73  0]
 [ 4  0 20]]
              precision    recall  f1-score   support

           0       0.93      0.92      0.93        62
           1       0.97      1.00      0.99        73
           2       0.87      0.83      0.85        24

    accuracy                           0.94       159
   macro avg       0.93      0.92      0.92       159
weighted avg       0.94      0.94      0.94       159
```

Classifiying new rows

```
In [31]: def predict(test):
    test = scaler.transform(test)
    return log_reg.predict(test)

name          = input('Name:')
sales         = int(input("Sales:"))
quantity      = int(input("Quantity:"))
discount_rate = float(input("Discount:"))
profit        = int(input("Profit:"))

cluster = predict(pd.DataFrame({'Sales' :[sales], 'Quantity' :[quantity] , 'Discount' : [discount_rate] , 'Profit':[profit]}))
print(f'Cusomer {name} belongs to cluster :{cluster[0]}')
```

```
Name:Harish
Sales:100000000000
Quantity:2123
Discount:0.1
Profit:100020
Cusomer Harish belongs to cluster :2
```

Conclusion:

The value of clustering, time series analysis, and classification techniques in extracting actionable insights from customer invoice data. The project's findings provide businesses with the knowledge and tools to improve marketing effectiveness, optimize inventory management, reduce customer churn, and make informed decisions for sustainable growth and success in the market.