# Sales Data Analysis Documentation

## Introduction

In this Jupyter Notebook, we perform an analysis of sales data to derive insights and recommendations. The analysis includes data loading, exploration, cleaning, exploratory data analysis (EDA), and visualization.

## Loading the data

We start by loading the sales data from the provided CSV file using the pandas library. The data contains various columns related to sales transactions, including invoice details, product information, pricing, and customer feedback.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import plotly.express as px
import plotly.io as pio

data = pd.read_csv('supermarket_sales - Sheet1.csv')
data['Date'] = pd.to_datetime(data['Date'])
```

# Data Exploration

After loading the data, we examine its structure and data types. The dataset consists of both categorical and numerical features. We check for missing values and outliers, and explore the basic statistics of the dataset.

```
print(data.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Invoice ID               1000 non-null   object
 1   Branch                   1000 non-null   object
 2   City                     1000 non-null   object
```

```
 3   Customer type            1000 non-null   object
 4   Gender                   1000 non-null   object
 5   Product line             1000 non-null   object
 6   Unit price               1000 non-null   float64
 7   Quantity                 1000 non-null   int64
 8   Tax 5%                   1000 non-null   float64
 9   Total                    1000 non-null   float64
 10  Date                     1000 non-null   datetime64[ns]
 11  Time                     1000 non-null   object
 12  Payment                  1000 non-null   object
 13  cogs                     1000 non-null   float64
 14  gross margin percentage  1000 non-null   float64
 15  gross income             1000 non-null   float64
 16  Rating                   1000 non-null   float64
dtypes: datetime64[ns](1), float64(7), int64(1), object(8)
memory usage: 132.9+ KB
None
```

# Data Cleaning

We handle missing values, if any, by applying appropriate techniques such as filling with mean/median values or dropping rows/columns. Outliers are detected using box plots, and we consider their impact on the analysis.

```python
print(data.isnull().sum())
```

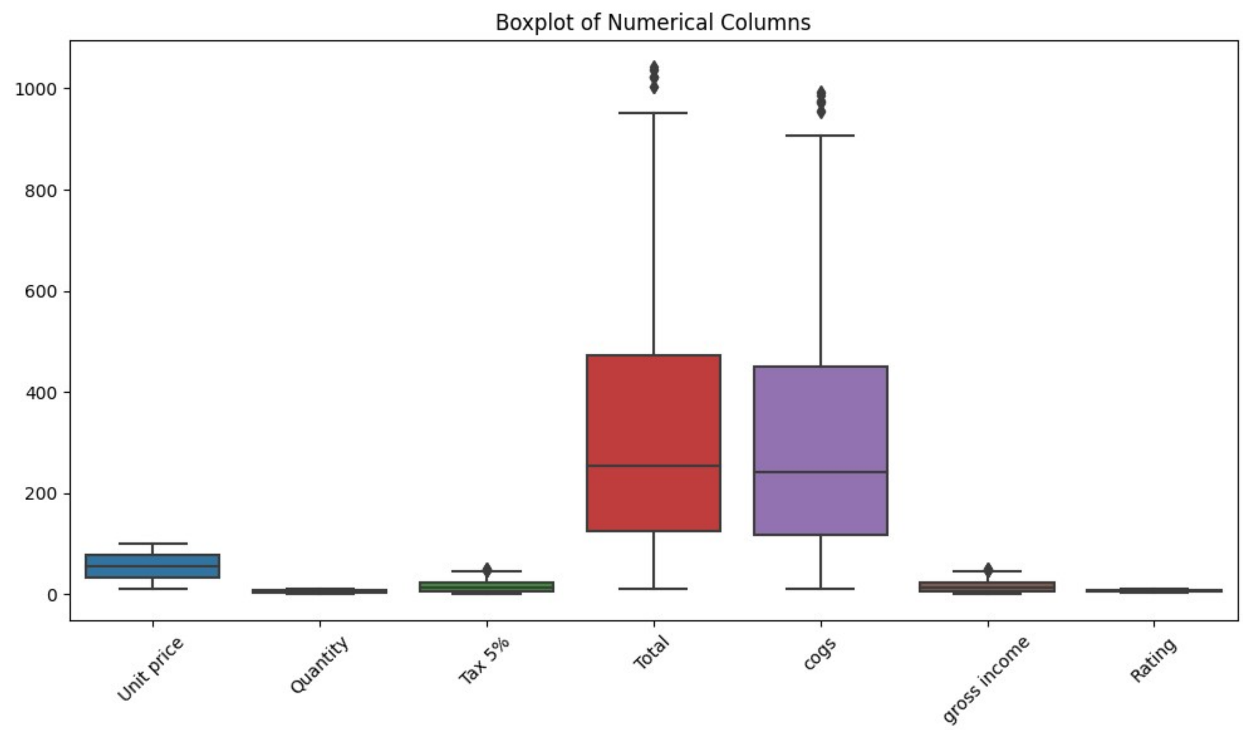
```
Invoice ID                 0
Branch                     0
City                       0
Customer type              0
Gender                     0
Product line               0
Unit price                 0
Quantity                   0
Tax 5%                     0
Total                      0
Date                       0
Time                       0
Payment                    0
cogs                       0
gross margin percentage    0
gross income               0
Rating                     0
dtype: int64
```

```python
data = data.drop(columns=['Invoice ID'])
```

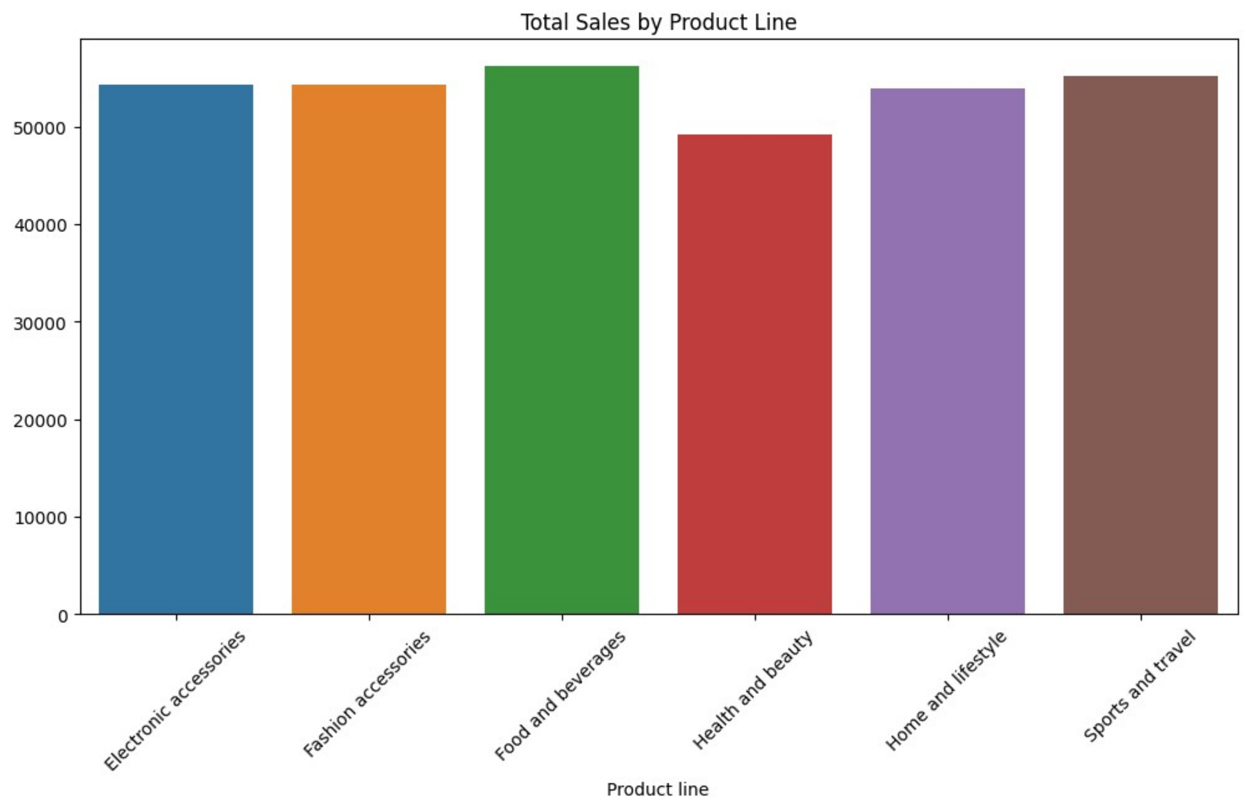
# Exploratory Data Analysis (EDA)

We perform EDA to uncover patterns, trends, and relationships within the sales data. We analyze total sales by product line, city-wise variations in sales, and correlations between key metrics.

```
numerical_columns = ['Unit price', 'Quantity', 'Tax 5%', 'Total',
'cogs', 'gross income', 'Rating']
plt.figure(figsize=(12, 6))
sns.boxplot(data=data[numerical_columns])
plt.title('Boxplot of Numerical Columns')
plt.xticks(rotation=45)
plt.show()
```

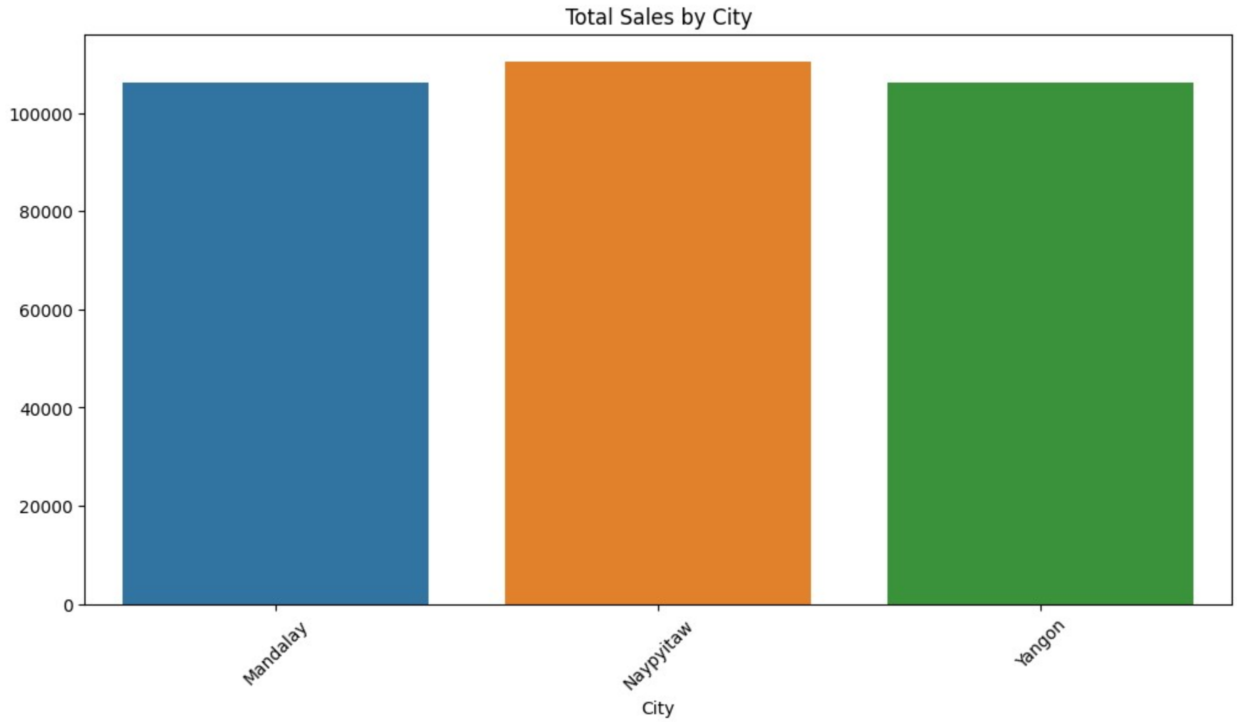



```
total_sales = data['Total'].sum()
average_sales = data['Total'].mean()
sales_by_product_line = data.groupby('Product line')['Total'].sum()
sales_by_city = data.groupby('City')['Total'].sum()

plt.figure(figsize=(12, 6))
sns.barplot(x=sales_by_product_line.index,
y=sales_by_product_line.values)
plt.title('Total Sales by Product Line')
plt.xticks(rotation=45)
plt.show()
```




# As we can see the most sales are done by Foods n beverages

```
plt.figure(figsize=(12, 6))
sns.barplot(x=sales_by_city.index, y=sales_by_city.values)
plt.title('Total Sales by City')
plt.xticks(rotation=45)
plt.show()
```




# Naypyitaw has done the most amount of sales

# Label encoding

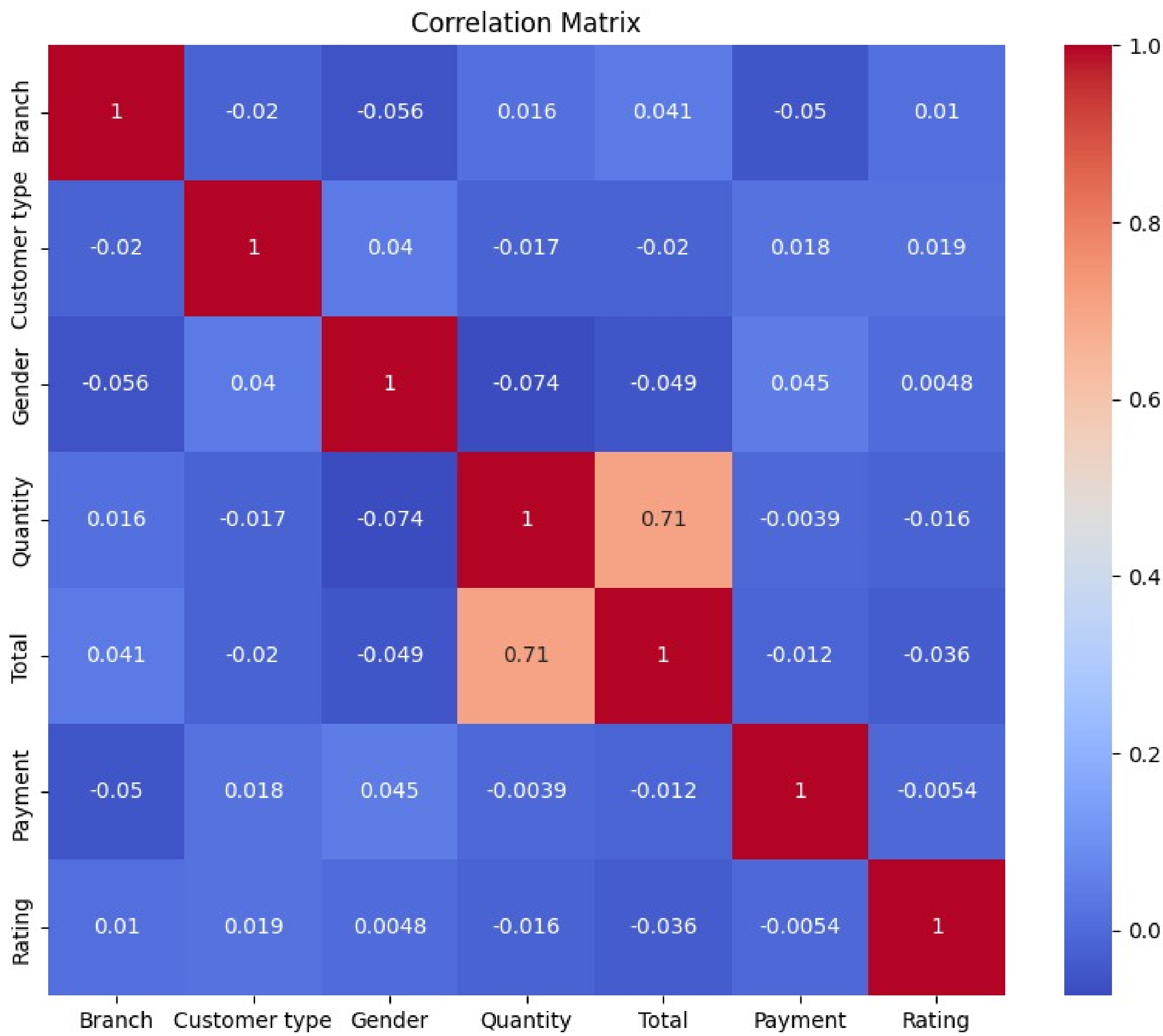
We assign values to categorical for better calculation

```
# Initializing LabelEncoder
label_encoder = LabelEncoder()

# Labelling encode "Branch," "Customer type," and "Gender" columns
columns_to_encode = ['Branch', 'Customer type', 'Gender','Payment']
for column in columns_to_encode:
    data[column] = label_encoder.fit_transform(data[column])

print(data.head())
```

```
   Branch       City  Customer type  Gender            Product line  \
0       0     Yangon              0       0       Health and beauty
1       2  Naypyitaw              1       0  Electronic accessories
2       0     Yangon              1       1      Home and lifestyle
3       0     Yangon              0       1       Health and beauty
4       0     Yangon              1       1       Sports and travel

   Unit price  Quantity   Tax 5%     Total       Date    Time  Payment
```

```
cogs  \
0        74.69          7  26.1415  548.9715 2019-01-05  13:08          2
522.83
1        15.28          5   3.8200   80.2200 2019-03-08  10:29          0
76.40
2        46.33          7  16.2155  340.5255 2019-03-03  13:23          1
324.31
3        58.22          8  23.2880  489.0480 2019-01-27  20:33          2
465.76
4        86.31          7  30.2085  634.3785 2019-02-08  10:37          2
604.17

   gross margin percentage  gross income  Rating
0                 4.761905       26.1415     9.1
1                 4.761905        3.8200     9.6
2                 4.761905       16.2155     7.4
3                 4.761905       23.2880     8.4
4                 4.761905       30.2085     5.3
```

```
columns_of_interest = ['Branch', 'Customer type', 'Gender',
'Quantity', 'Total', 'Payment', 'Rating']

# Calculating the correlation matrix
correlation_matrix = data[columns_of_interest].corr()

# Creating a heatmap using Seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```




This shows the co relation between all the feature and negative co relation means that they are inversely propotioanl and a positive one means that they are propotional

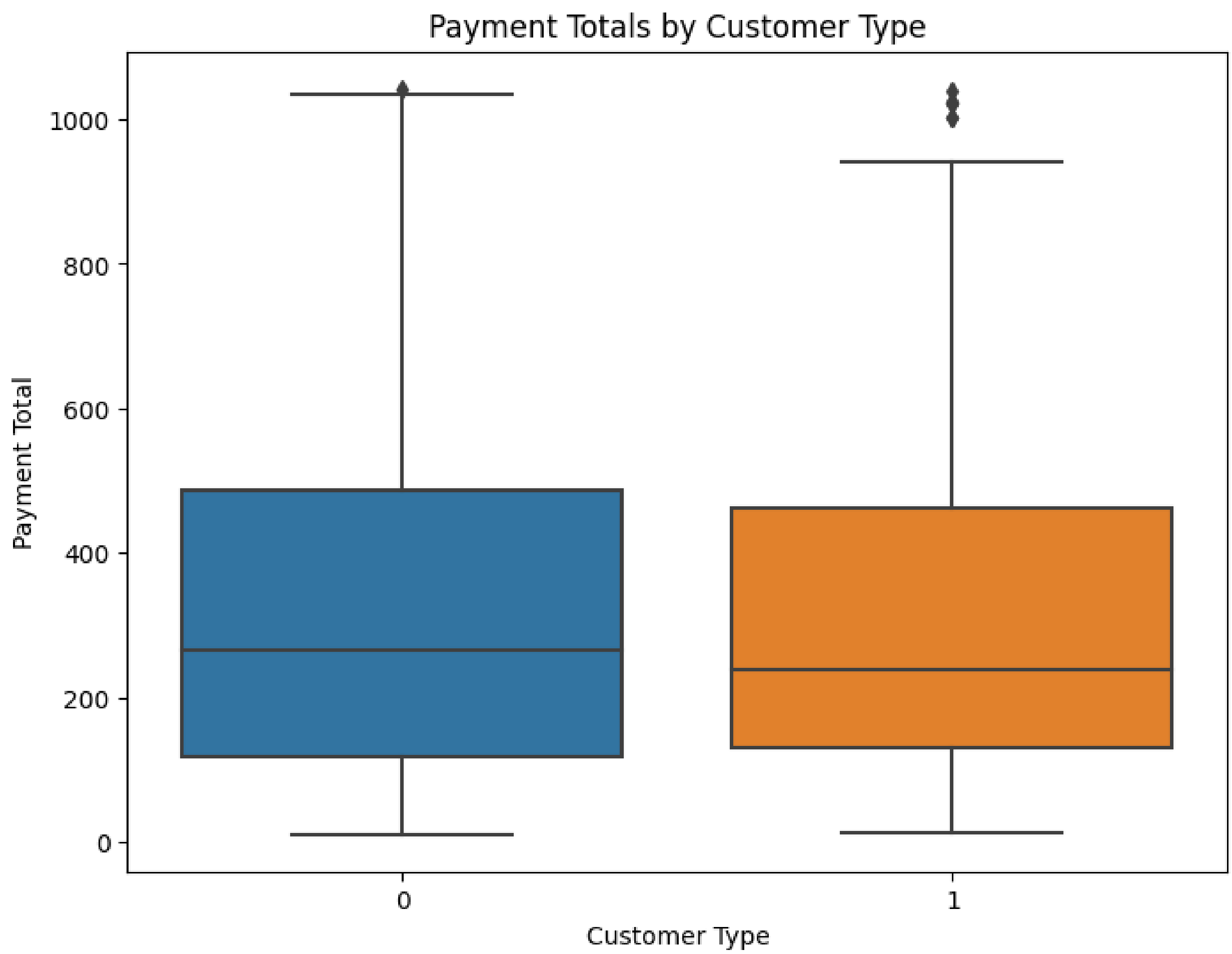
```
# Calculating the correlation between "Customer type" and "Payment total"
correlation_payment = data['Customer type'].corr(data['Total'])

# Calculating the correlation between "Customer type" and "Rating"
correlation_rating = data['Customer type'].corr(data['Rating'])
```

```
print("Correlation between Customer type and Payment total:",
correlation_payment)
print("Correlation between Customer type and Rating:",
correlation_rating)
```

```
Correlation between Customer type and Payment total: -
0.019670282859210724
Correlation between Customer type and Rating: 0.018888672182968986
```

```
payment_stats = data.groupby('Customer type')['Total'].agg(['mean',
'median'])

print(payment_stats)

# Creating  a box plot to visualize the distribution of payment totals
by customer type
plt.figure(figsize=(8, 6))
sns.boxplot(data=data, x='Customer type', y='Total')
plt.title('Payment Totals by Customer Type')
plt.ylabel('Payment Total')
plt.xlabel('Customer Type')
plt.show()
```

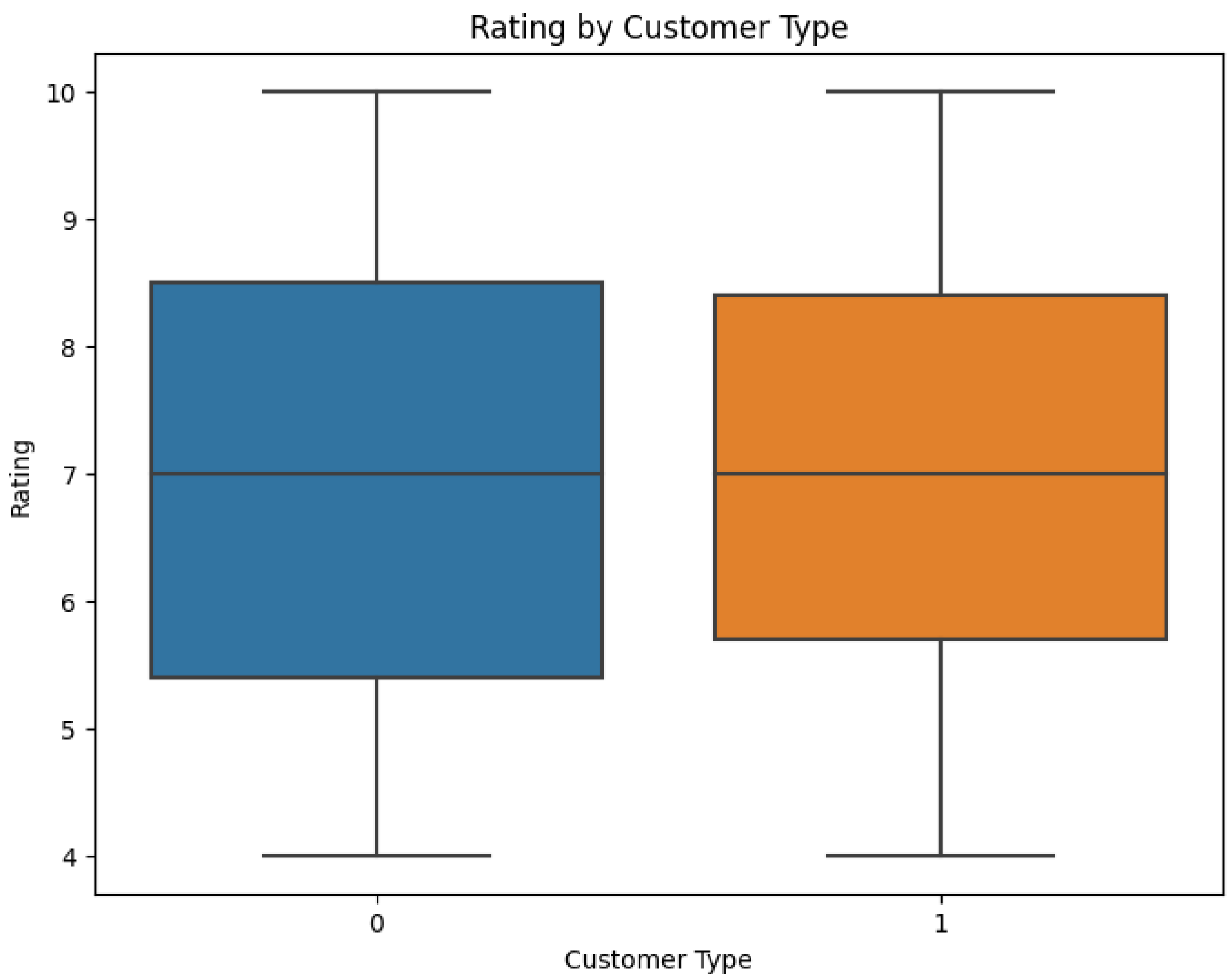
```
                     mean   median
Customer type
0              327.791305  266.028
1              318.122856  237.426
```




# We can see that non members pay more than members

```
payment_stats = data.groupby('Customer type')['Rating'].agg(['mean',
'median'])

print(payment_stats)

# Creating a box plot to visualize the distribution of payment totals
by customer type
plt.figure(figsize=(8, 6))
sns.boxplot(data=data, x='Customer type', y='Rating')
plt.title('Rating by Customer Type')
plt.ylabel('Rating')
plt.xlabel('Customer Type')
plt.show()
```

```
               mean  median
Customer type
0          6.940319     7.0
1          7.005210     7.0
```




We can see that Non members give better ratings as compared to members but again the non members count is more than the members

```
print(data.head())

   Branch       City  Customer type  Gender             Product line  \
0       0     Yangon              0       0        Health and beauty
1       2  Naypyitaw              1       0  Electronic accessories
2       0     Yangon              1       1       Home and lifestyle
3       0     Yangon              0       1        Health and beauty
```

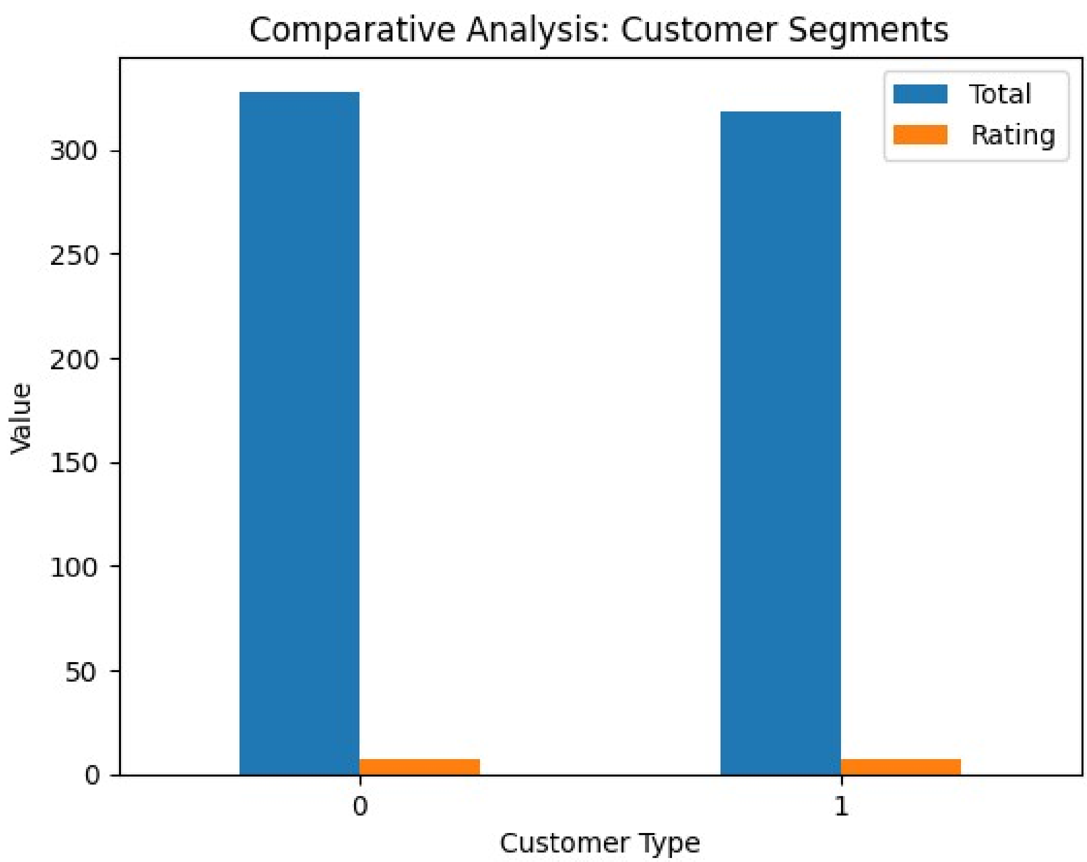
```
4       0     Yangon              1       1       Sports and travel

   Unit price  Quantity   Tax 5%     Total       Date   Time  Payment
cogs  \
0       74.69         7  26.1415  548.9715 2019-01-05  13:08        2
522.83
1       15.28         5   3.8200   80.2200 2019-03-08  10:29        0
76.40
2       46.33         7  16.2155  340.5255 2019-03-03  13:23        1
324.31
3       58.22         8  23.2880  489.0480 2019-01-27  20:33        2
465.76
4       86.31         7  30.2085  634.3785 2019-02-08  10:37        2
604.17

   gross margin percentage  gross income  Rating
0                 4.761905       26.1415     9.1
1                 4.761905        3.8200     9.6
2                 4.761905       16.2155     7.4
3                 4.761905       23.2880     8.4
4                 4.761905       30.2085     5.3
```

# Here we have a Interactive dashboard

```
pio.renderers.default = 'notebook'
fig = px.bar(data, x='Total', y='Quantity', color='Product line',
hover_data=['City', 'Rating'])

# Creating layout for the dashboard
layout = {
    'title': 'Sales Analysis Dashboard',
    'xaxis': {'title': 'Total Sales'},
    'yaxis': {'title': 'Quantity'},
}

# Adding any additional visualizations, like bar charts or line plots,
using px.bar() or px.line()

# Display the dashboard
fig.update_layout()
fig.show()

customer_type_stats = data.groupby('Customer type').agg({'Total':
'mean', 'Rating': 'mean'})

print(customer_type_stats)

# bar plot to compare mean payment totals and ratings by customer type
```

```
plt.figure(figsize=(10, 6))
customer_type_stats.plot(kind='bar')
plt.title('Comparative Analysis: Customer Segments')
plt.ylabel('Value')
plt.xlabel('Customer Type')
plt.xticks(rotation=0)
plt.show()
```

```
                    Total    Rating
Customer type
0              327.791305  6.940319
1              318.122856  7.005210

<Figure size 1000x600 with 0 Axes>
```



# As we can see that using bar plots we found that Non members pay more than members , but there is a slight difference in the ratings part as we can see that members rate highly than non members

```
city_stats = data.groupby('City').agg({'Total': 'sum', 'Rating':
'mean'})

# Sorting cities by total spending and average ratings
sorted_cities_by_spending = city_stats.sort_values(by='Total',
ascending=False)
sorted_cities_by_ratings = city_stats.sort_values(by='Rating',
ascending=False)

print("Cities with the highest total spending:")
print(sorted_cities_by_spending)

print("\nCities with the highest average ratings:")
print(sorted_cities_by_ratings)
```

```
Cities with the highest total spending:
                 Total    Rating
City
Naypyitaw  110568.7065  7.072866
Yangon     106200.3705  7.027059
Mandalay   106197.6720  6.818072

Cities with the highest average ratings:
                 Total    Rating
City
Naypyitaw  110568.7065  7.072866
Yangon     106200.3705  7.027059
Mandalay   106197.6720  6.818072
```

# This is statiscally correct so we need to normalise it for better understanding of the differences

```
city_stats = data.groupby('City').agg({'Total': 'sum', 'Rating':
'mean', 'Total': 'count'})
```

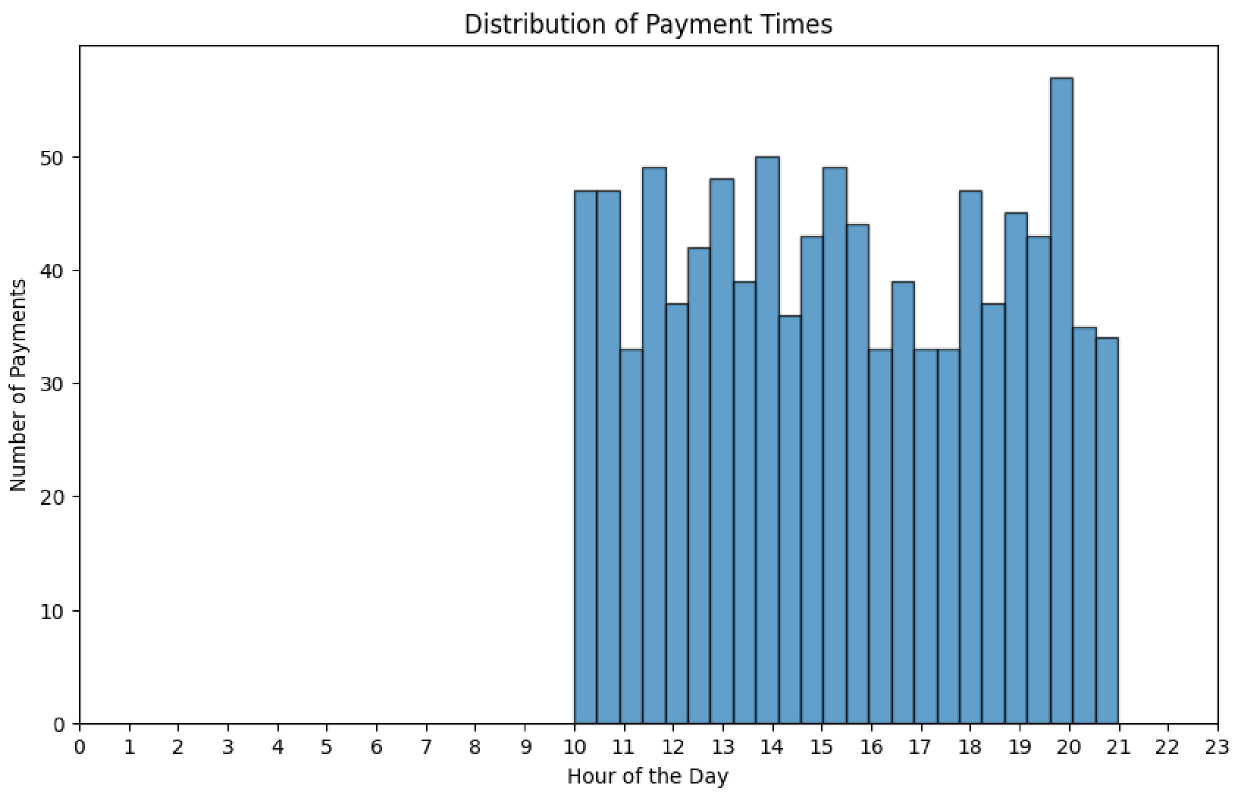
```python
# Normalizing total spending and average ratings by total number of people
city_stats['Spending per Person'] = city_stats['Total'] / city_stats['Total']
city_stats['Normalized Rating'] = city_stats['Rating'] * city_stats['Total']

# Sorting cities by normalized total spending and normalized average ratings
sorted_cities_by_spending = city_stats.sort_values(by='Spending per Person', ascending=True)
sorted_cities_by_ratings = city_stats.sort_values(by='Normalized Rating', ascending=True)

print("Cities sorted by normalized spending per person:")
print(sorted_cities_by_spending)

print("\nCities sorted by normalized average ratings:")
print(sorted_cities_by_ratings)
```

```
Cities sorted by normalized spending per person:
           Total    Rating  Spending per Person  Normalized Rating
City
Mandalay     332  6.818072                  1.0             2263.6
Naypyitaw    328  7.072866                  1.0             2319.9
Yangon       340  7.027059                  1.0             2389.2

Cities sorted by normalized average ratings:
           Total    Rating  Spending per Person  Normalized Rating
City
Mandalay     332  6.818072                  1.0             2263.6
Naypyitaw    328  7.072866                  1.0             2319.9
Yangon       340  7.027059                  1.0             2389.2
```

# This is the normalised ratings

```python
data['Numeric Time'] = pd.to_datetime(data['Time']).dt.hour + pd.to_datetime(data['Time']).dt.minute / 60

# Analyzing the distribution of payment times
plt.figure(figsize=(10, 6))
plt.hist(data['Numeric Time'], bins=24, edgecolor='k', alpha=0.7)
plt.title('Distribution of Payment Times')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Payments')
plt.xticks(range(0, 24))
plt.show()
```

```
/var/folders/cb/5j26d32n0p31wf68cqjpgr3h0000gn/T/
ipykernel_63405/1969741099.py:1: UserWarning:

Could not infer format, so each element will be parsed individually,
falling back to `dateutil`. To ensure parsing is consistent and as-
expected, please specify a format.

/var/folders/cb/5j26d32n0p31wf68cqjpgr3h0000gn/T/ipykernel_63405/19697
41099.py:1: UserWarning:

Could not infer format, so each element will be parsed individually,
falling back to `dateutil`. To ensure parsing is consistent and as-
expected, please specify a format.
```

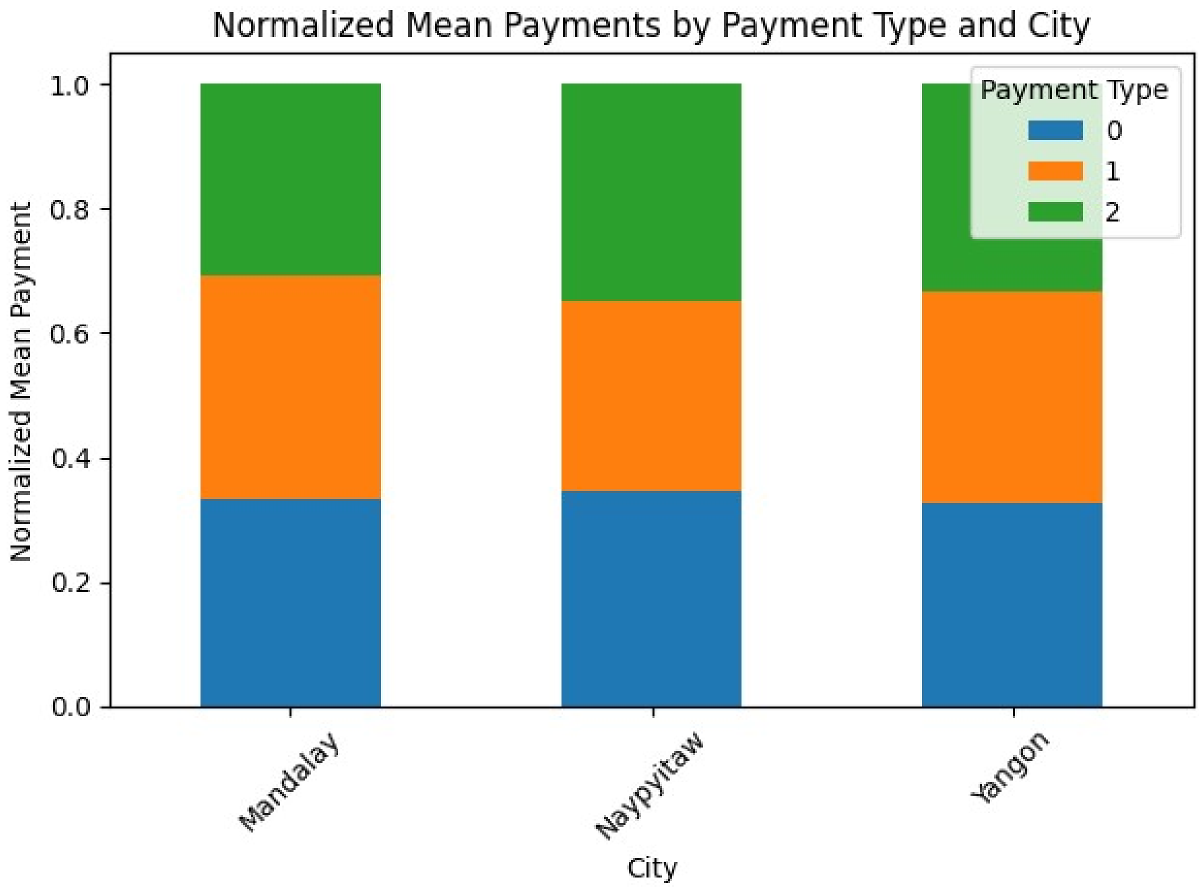


As we can see that the busiest time of the day is during 20:00hrs and the supermarket might use some more extra staff at that point of time in the day

```
city_payment_stats = data.groupby(['City', 'Payment'])
['Total'].mean().reset_index()

# Pivoting the data for visualization
pivot_table = city_payment_stats.pivot(index='City',
columns='Payment', values='Total')

# Normalizing the pivot table by row (city)
normalized_pivot = pivot_table.div(pivot_table.sum(axis=1), axis=0)
print(normalized_pivot)

# Visualizing the normalized data using a stacked bar plot
plt.figure(figsize=(10, 6))
normalized_pivot.plot(kind='bar', stacked=True)
plt.title('Normalized Mean Payments by Payment Type and City')
plt.xlabel('City')
plt.ylabel('Normalized Mean Payment')
plt.xticks(rotation=45)
plt.legend(title='Payment Type')
plt.tight_layout()
plt.show()
```

```
Payment           0         1         2
City
Mandalay   0.334494  0.356718  0.308788
Naypyitaw  0.344896  0.307175  0.347929
Yangon     0.327604  0.339463  0.332933
```

```
<Figure size 1000x600 with 0 Axes>
```



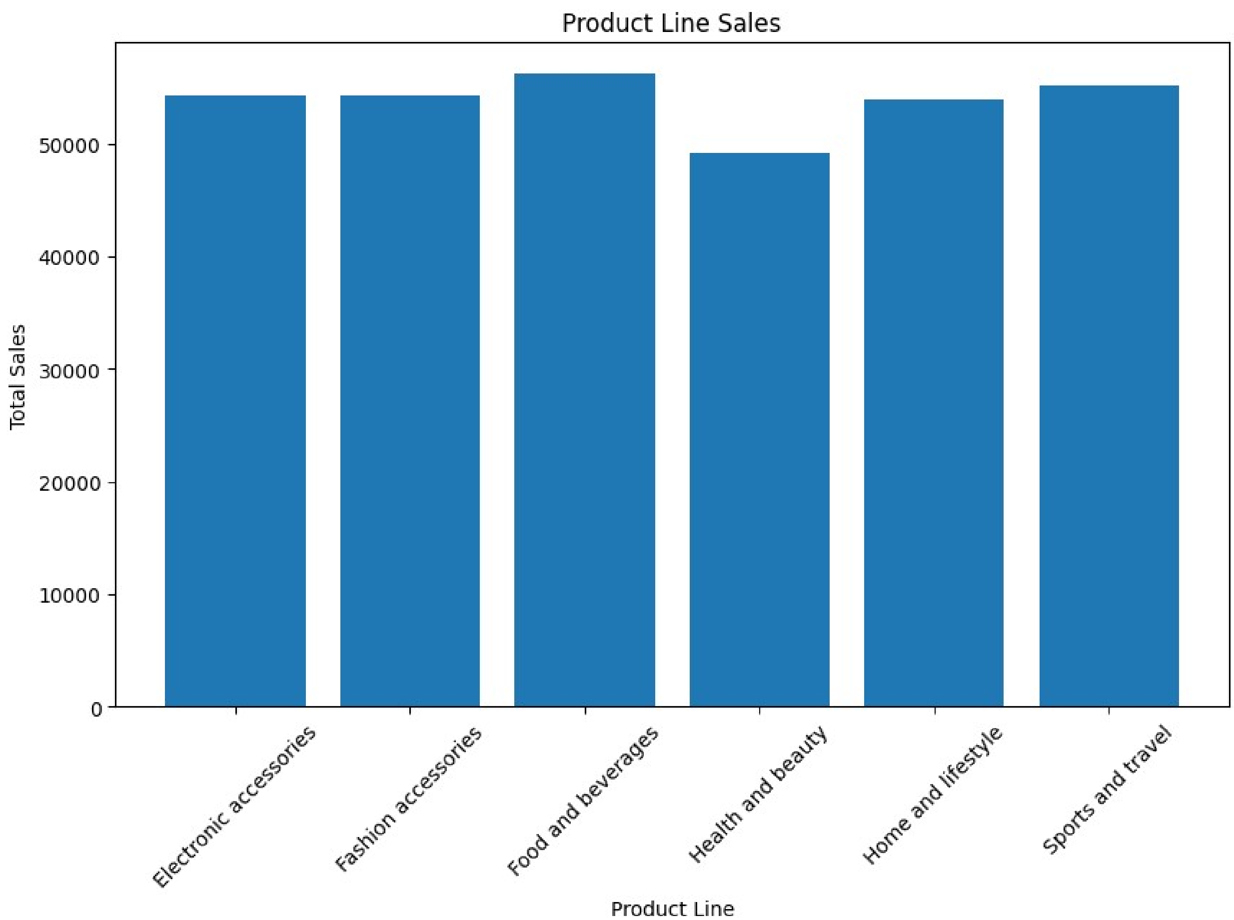

# Here we find out that

Mandalay prefers credit card Naypyitaw prefers E wallet Yango prefers Cash

```
product_line_sales = data.groupby('Product line')['Total'].sum()

#  product line with the highest sales
highest_sales_product_line = product_line_sales.idxmax()
highest_sales_amount = product_line_sales.max()

print("Product line with the highest sales:",
highest_sales_product_line)
print("Highest sales amount:", highest_sales_amount)

# Create a bar plot to visualize product line sales
plt.figure(figsize=(10, 6))
plt.bar(product_line_sales.index, product_line_sales.values)
plt.xlabel('Product Line')
plt.ylabel('Total Sales')
plt.title('Product Line Sales')
```

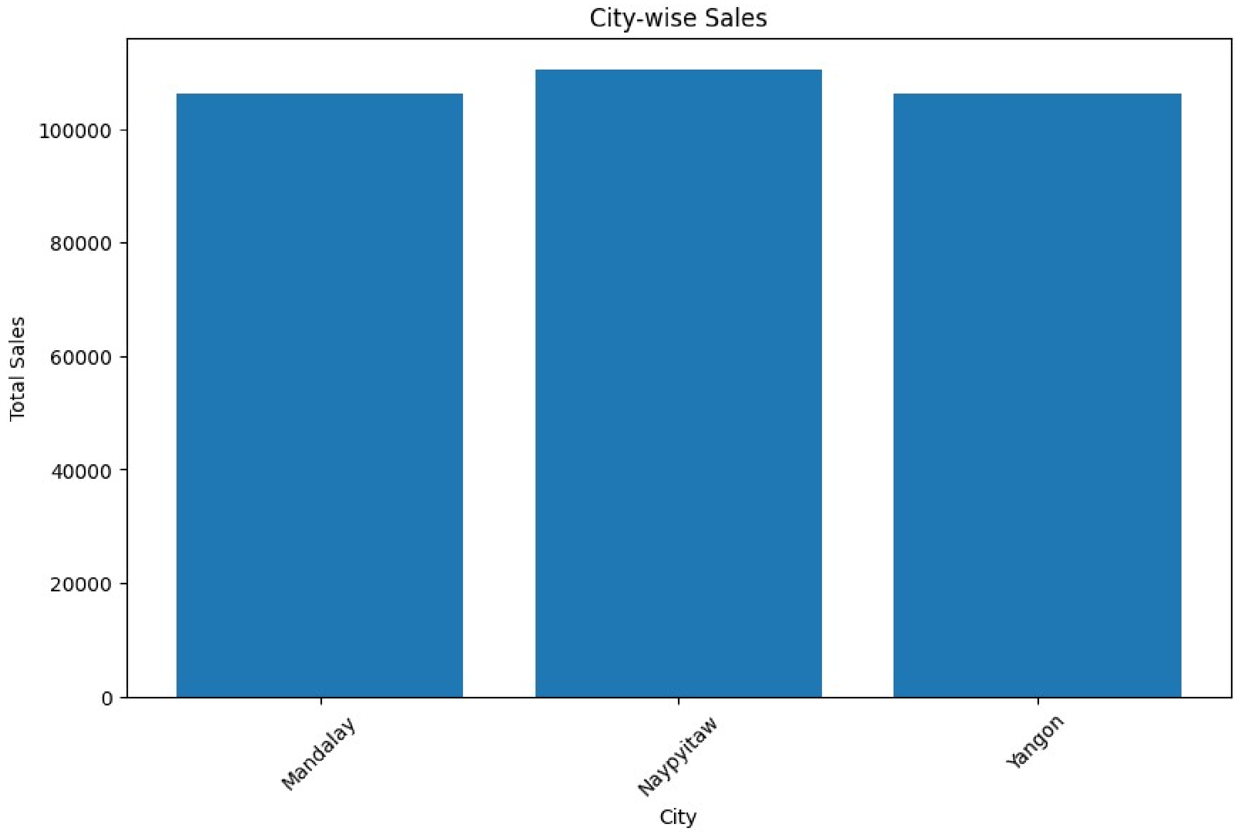
```
plt.xticks(rotation=45)
plt.show()
```

```
Product line with the highest sales: Food and beverages
Highest sales amount: 56144.844
```




```
city_sales = data.groupby('City')['Total'].sum()

# Finding the city with the highest sales
highest_sales_city = city_sales.idxmax()
highest_sales_amount = city_sales.max()

print("City with the highest sales:", highest_sales_city)
print("Highest sales amount:", highest_sales_amount)

# Creating a bar plot to visualize city-wise sales
plt.figure(figsize=(10, 6))
plt.bar(city_sales.index, city_sales.values)
plt.xlabel('City')
plt.ylabel('Total Sales')
plt.title('City-wise Sales')
```

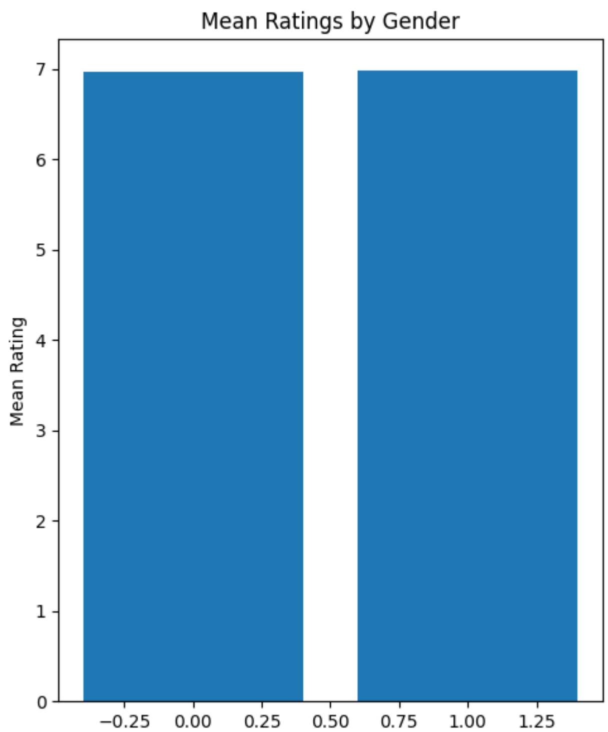
```
plt.xticks(rotation=45)
plt.show()

City with the highest sales: Naypyitaw
Highest sales amount: 110568.7065
```

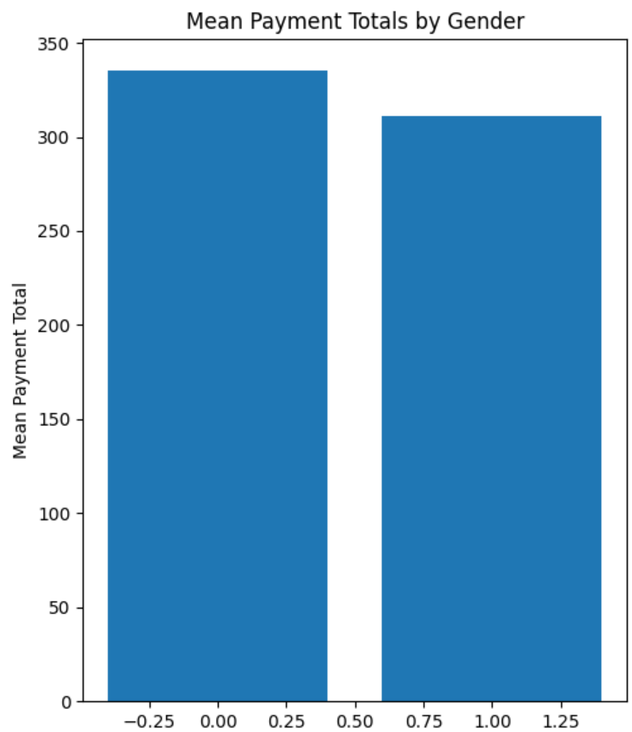



```
# Grouping data by "Gender" and  mean ratings and mean payment totals
gender_stats = data.groupby('Gender').agg({'Rating': 'mean', 'Total':
'mean'})

# Displaying the gender statistics
print("Mean Ratings and Mean Payment Totals by Gender:")
print(gender_stats)

# Creating bar plots for mean ratings and mean payment totals by
gender
plt.figure(figsize=(10, 6))

plt.subplot(1, 2, 1)
plt.bar(gender_stats.index, gender_stats['Rating'])
plt.title('Mean Ratings by Gender')
plt.ylabel('Mean Rating')

plt.subplot(1, 2, 2)
```

```
plt.bar(gender_stats.index, gender_stats['Total'])
plt.title('Mean Payment Totals by Gender')
plt.ylabel('Mean Payment Total')

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

```
Mean Ratings and Mean Payment Totals by Gender:
          Rating       Total
Gender
0       6.964471  335.095659
1       6.980962  310.789226
```



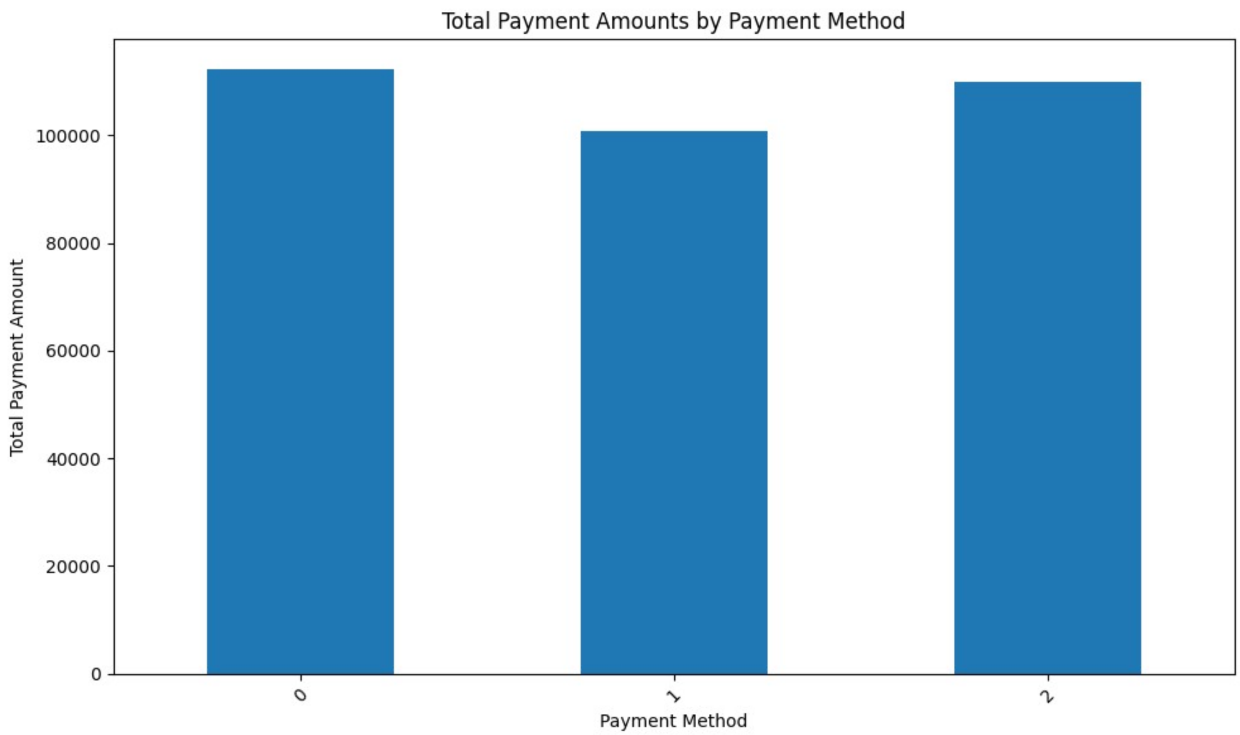




# We can see that Women spend more but men rate better

```
# Grouping data by "Payment" type and  total payment amounts
payment_stats = data.groupby('Payment')['Total'].sum()

# Displaying payment statistics
print("Total Payment Amounts by Payment Method:")
print(payment_stats)

# bar plot to visualize total payment amounts by payment method
```

```
plt.figure(figsize=(10, 6))
payment_stats.plot(kind='bar')
plt.title('Total Payment Amounts by Payment Method')
plt.xlabel('Payment Method')
plt.ylabel('Total Payment Amount')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
Total Payment Amounts by Payment Method:
Payment
0    112206.570
1    100767.072
2    109993.107
Name: Total, dtype: float64
```

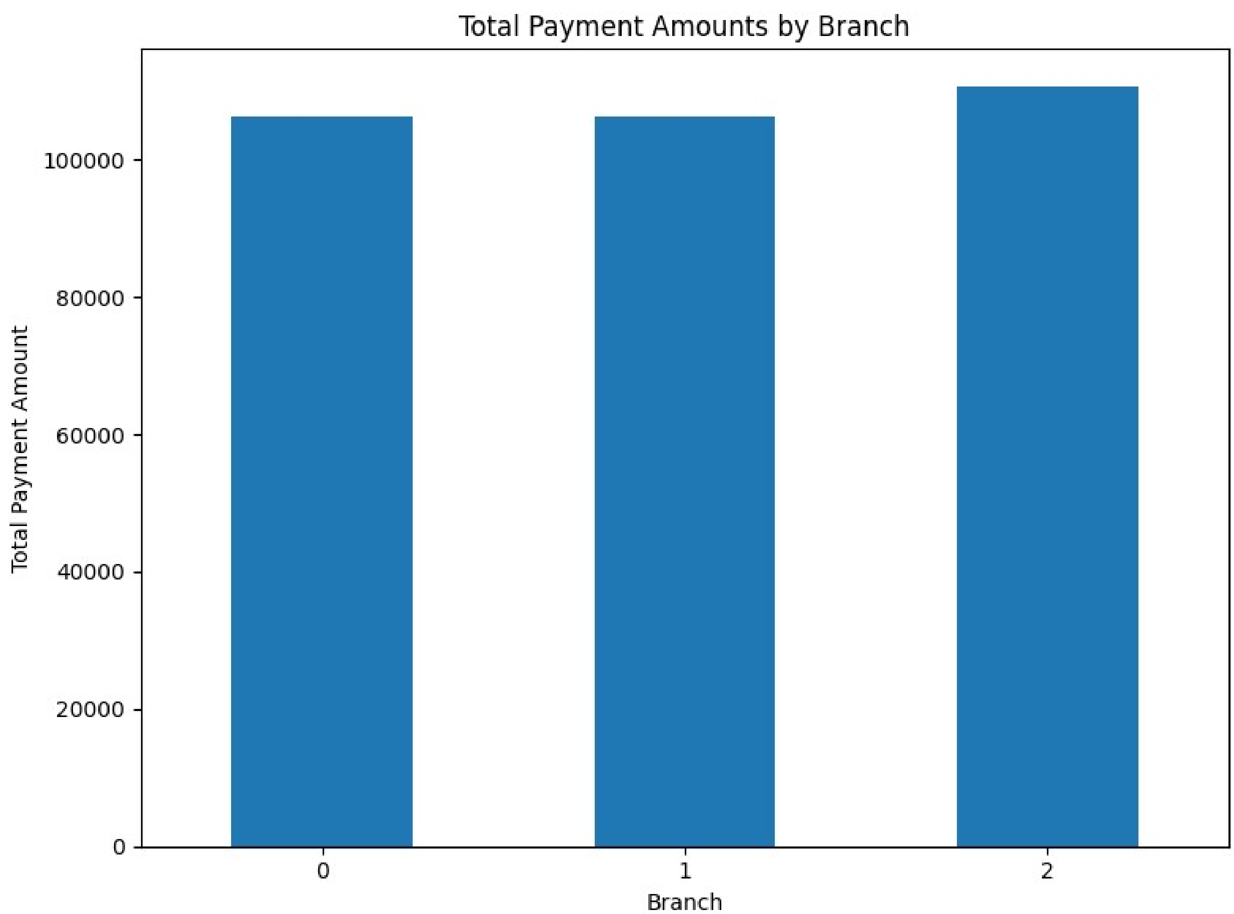



# Although there might be new modes of payments but people still prefer cash as their prefered mode of payment

```
branch_stats = data.groupby('Branch')['Total'].sum()

# Displaying branch statistics
```

```
print("Total Payment Amounts by Branch:")
print(branch_stats)

# Creating a bar plot to visualize total payment amounts by branch
plt.figure(figsize=(8, 6))
branch_stats.plot(kind='bar')
plt.title('Total Payment Amounts by Branch')
plt.xlabel('Branch')
plt.ylabel('Total Payment Amount')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
Total Payment Amounts by Branch:
Branch
0    106200.3705
1    106197.6720
2    110568.7065
Name: Total, dtype: float64
```



# the most amount of sales are done in Yangon and Supermarket should focus more on that branch

```
scenario_stats = data.groupby(['Time', 'Branch', 'Gender', 'Payment'])
['Total'].sum()

# Finding the scenario with the highest total profit
best_scenario = scenario_stats.idxmax()
highest_profit = scenario_stats.max()

print("Best Scenario for Highest Profit:")
print("Time:", best_scenario[0])
print("Branch:", best_scenario[1])
print("Gender:", best_scenario[2])
print("Payment Method:", best_scenario[3])
print("Highest Profit:", highest_profit)
```

```
Best Scenario for Highest Profit:
Time: 13:00
Branch: 2
Gender: 0
Payment Method: 1
Highest Profit: 1963.605
```

# This is the best case scenario for maximising profits

Time: 13:00 Branch: 2 Gender: 0 Payment Method: 1 Highest Profit: 1963.605

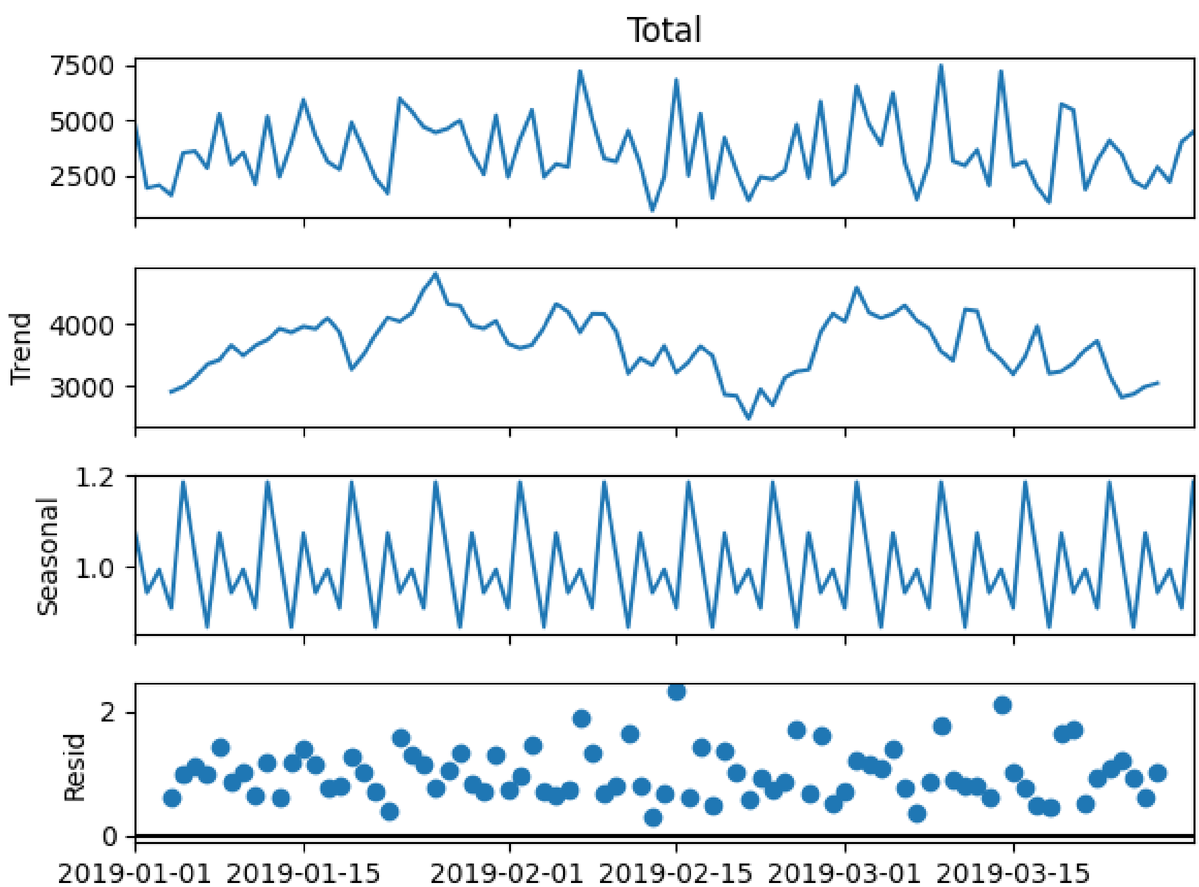
# Advanced Techniques

We perform advanced analysis techniques, such as Time forecasting, clustering, and elbow method

# Time series forecasting

```
# Creating a time series DataFrame
time_series_data = data.groupby('Date')['Total'].sum()

# Decomposing the time series data to identify trends and seasonality
```

```
result = seasonal_decompose(time_series_data, model='multiplicative')
result.plot()
plt.show()

# Fiting an Exponential Smoothing model for forecasting
model = ExponentialSmoothing(time_series_data, trend='add',
seasonal='add', seasonal_periods=12)
model_fit = model.fit()
forecast = model_fit.forecast(steps=12)  # Forecast for the next 12
months
print(forecast)
```




```
2019-03-31    2685.277342
2019-04-01    3504.847717
2019-04-02    2753.204956
2019-04-03    2461.195151
2019-04-04    3833.237702
2019-04-05    2683.920840
2019-04-06    3930.519992
2019-04-07    2627.871598
2019-04-08    4178.274214
```

```
2019-04-09    3804.675929
2019-04-10    2908.789480
2019-04-11    2375.527051
Freq: D, dtype: float64

/Users/karmukilan/miniconda3/lib/python3.11/site-packages/
statsmodels/tsa/base/tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency D will be
used.

/Users/karmukilan/miniconda3/lib/python3.11/site-packages/statsmodels/
tsa/holtwinters/model.py:917: ConvergenceWarning:

Optimization failed to converge. Check mle_retvals.
```

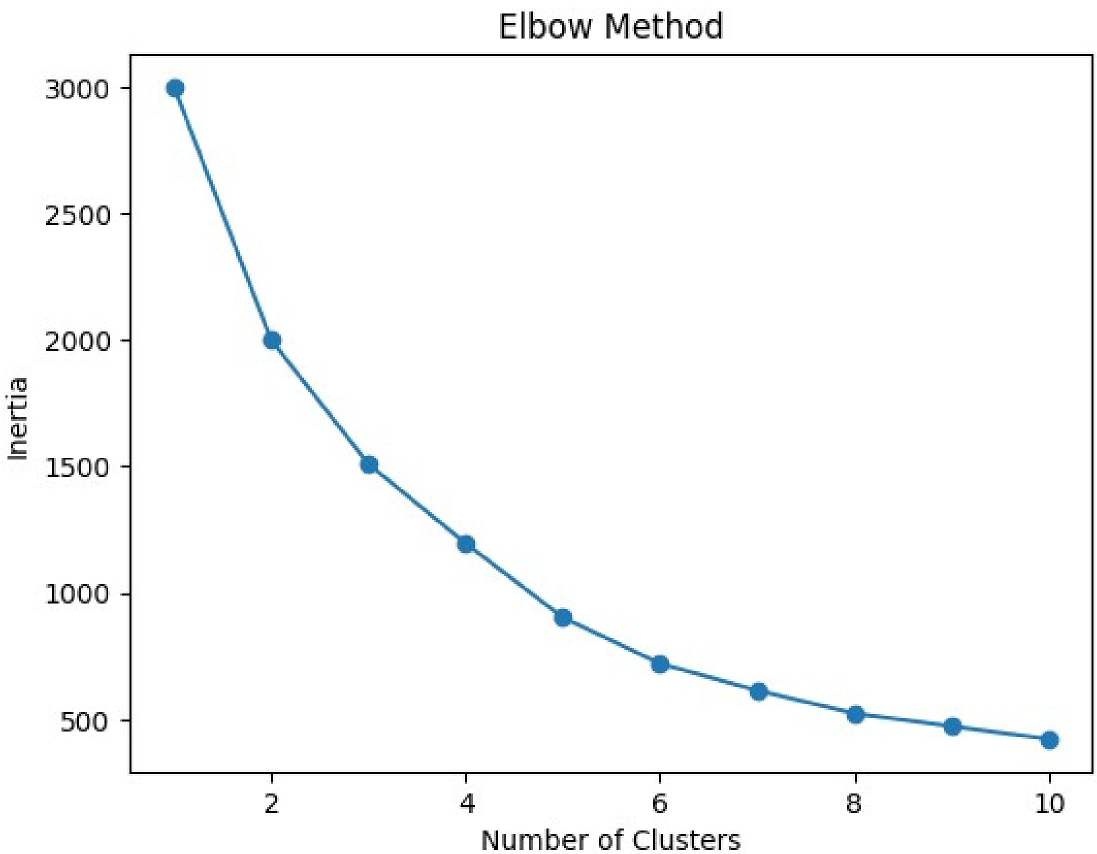
# Elbow method

```
features = data[['Customer type','Total','Rating']]

# Standardizing the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Determining the optimal number of clusters using the Elbow Method
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=0)
    kmeans.fit(scaled_features)
    inertia.append(kmeans.inertia_)

plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()

# Based on the elbow method, choose the optimal number of clusters and
fit KMeans
n_clusters = 3  # Example: 3 clusters
kmeans = KMeans(n_clusters=n_clusters, random_state=0)
kmeans.fit(scaled_features)

# Adddding cluster labels to the DataFrame
data['Cluster'] = kmeans.labels_

# Visualizing the clusters
sns.scatterplot(data=data, x='Customer type', y='Total',
```

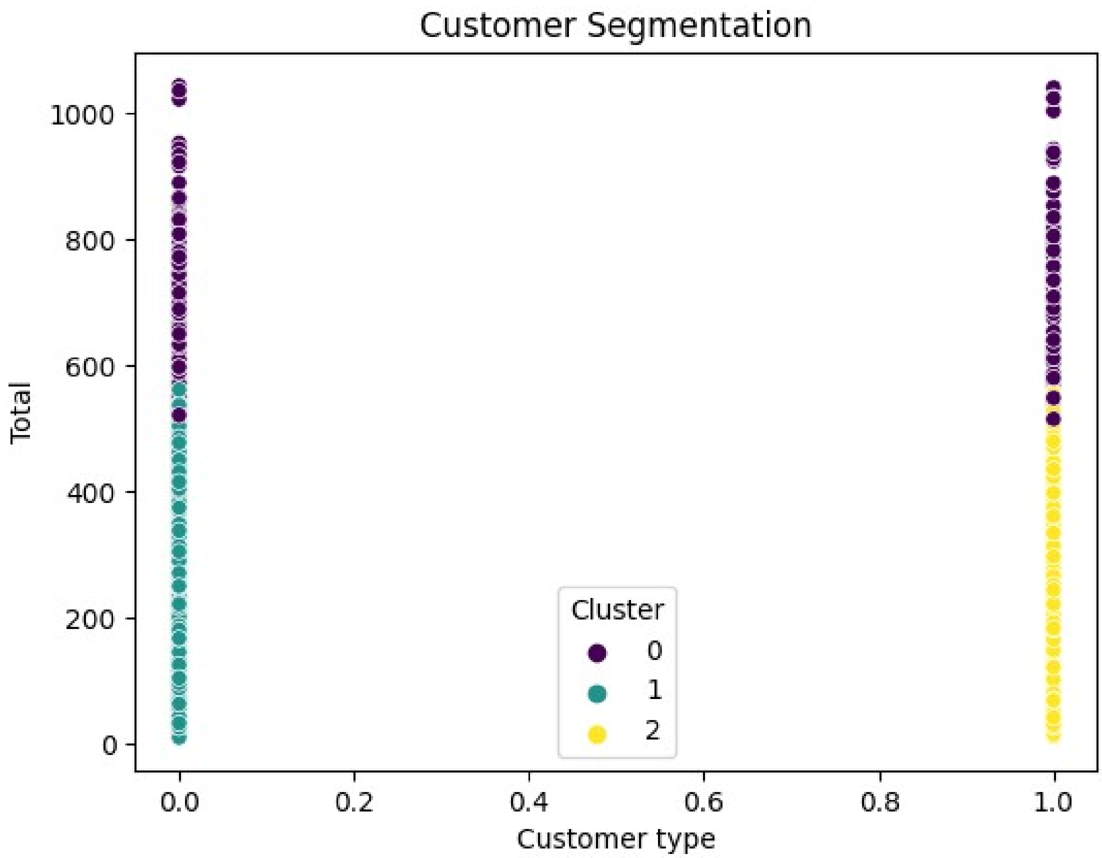
```
hue='Cluster', palette='viridis')
plt.title('Customer Segmentation')
plt.show()
```

```
/Users/karmukilan/miniconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: 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

/Users/karmukilan/miniconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: 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

/Users/karmukilan/miniconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: 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

/Users/karmukilan/miniconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: 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

/Users/karmukilan/miniconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: 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

/Users/karmukilan/miniconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: 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

/Users/karmukilan/miniconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: 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

/Users/karmukilan/miniconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: 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
```

```
/Users/karmukilan/miniconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: 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

/Users/karmukilan/miniconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: 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
```




```
/Users/karmukilan/miniconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: 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
```


Customer Segmentation
Total
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0
Customer type
Cluster
0
1
2