Data Analysis and Data Visualization Using Python

About Project:

This project explores the power of Python for data analysis and visualization, transforming raw data into actionable insights. By leveraging Python’s robust libraries, such as Pandas, NumPy, Matplotlib, and Seaborn, the project demonstrates how to clean, analyse, and visualize data effectively. Whether it’s identifying trends, patterns, or anomalies, this project showcases how Python can be used to make data-driven decisions through clear and compelling visualizations. The goal is to provide a comprehensive understanding of data analysis processes while delivering meaningful visual representations that can inform strategic decisions.

- Importing and managing data from different sources using Python.
- Cleaning and preprocessing data to ensure accuracy and consistency.
- Performing descriptive statistics to summarize and understand data.
- Manipulating data with Pandas for in-depth analysis.
- Visualizing data trends and patterns using Matplotlib.
- Creating various plot types, including Pie charts, bar charts, and line graphs, Donut chart,etc.
- Identifying correlations and relationships within datasets.
- Making data-driven decisions based on analysis and visualizations.
- Enhancing Python programming skills through practical application.
- Understanding the importance of effective data presentation

Prepared By: Khushboo Makhijani

Date: 16th May,2024

Starting the Program

This program is designed to perform comprehensive data analysis and visualization on Supermarket Sales. The analysis aims to uncover patterns, trends, and insights that can inform strategic decisions. The program starts by importing essential libraries and loading the dataset for further processing.

Loading Libraries

To begin the data analysis and visualization process, we first need to import the necessary Python libraries. These libraries provide a wide range of functions and tools that facilitate data manipulation, statistical analysis, and the creation of visualizations.
-Pandas
-Matplotlib
l

Loading the Dataset

With the necessary libraries in place, the next step is to load the dataset into the environment. The dataset, named `Supermarket Sales, is stored in a CSV file and contains Sales data of Supermarket. Loading the dataset into a Pandas Data Frame allows us to easily manipulate, analyse, and visualize the data.

In [5]:

```
#Data Analysis and visualization with Python

#importing essential libraries for data analysis and visualization
import pandas as pd import matplotlib.pyplot as plt
```

In [6]:

```
#importing data of supermarket sales import pandas as pd
data=pd.read_csv("supermarket sales.csv") #importing excel sheet/data into pyth
```

o

In [7]:

```
data.head(10) #data.head is used to print top rows mentioned in ()
```

Out[7]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415
1	226-313081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200

2	631-413108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155
3	123-191176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880
4	373-737910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085
5	699-143026	C	Naypyitaw	Normal	Male	Electronic accessories	85.39	7	29.8865
6	355-535943	A	Yangon	Member	Female	Electronic accessories	68.84	6	20.6520
7	315-225665	C	Naypyitaw	Normal	Female	Home and lifestyle	73.56	10	36.7800
8	665-329167	A	Yangon	Member	Female	Health and beauty	36.26	2	3.6260
9	692-925582	B	Mandalay	Member	Female	Food and beverages	54.84	3	8.2260

To check the dimensions of the dataset

data.shape is used to check the dimensions of the dataset. It returns a tuple representing the number of rows and columns in the DataFrame. This is a quick way to understand the size of the dataset and is often one of the first steps in exploratory data analysis.

```
In [8]: data.shape #data.shape helps to identify how many rows and columns are there in
Out[8]: (1000, 17)
```

Missing Data Analysis with ‘data.isnull()’

Data.isnull() is used to detect missing values in the dataset. It returns a dataframe of the same shape as data, where each element is a boolean indicating whether the corresponding element in the original dataframe is nan (true) or not (false).

```
In [10]: data.isnull().sum() #it is used to identify null/empty values in our data set
```

```
Out[10]: 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
```

Descriptive Statistics with data.describe()

data.describe() is used to generate descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset’s distribution, excluding NaN values. This method is particularly useful for getting a quick overview of the numerical data in the dataset.

```
In [11]: data.describe() # data.decribe gives Summary statistics of numerical columns
```

Out[11]:

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage
mean	55.672130	5.510000	15.379369	322.966749	307.58738	
std	26.494628	2.923431	11.708825	245.885335	234.17651	0.000000
min	10.080000	1.000000	0.508500	10.678500	10.17000	4.761905
25%	32.875000	3.000000	5.924875	124.422375	118.49750	4.761905
50%	55.230000	5.000000	12.088000	253.848000	241.76000	4.761905
75%	77.935000	8.000000	22.445250	471.350250	448.90500	4.761905
						1000.000000 10
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	4.761905

max	99.960000	10.000000	49.650000	1042.650000	993.00000	4.761905

In [12]:

```
#describing specific columns
 data[['Unit price', 'Quantity', 'Tax 5%',
'Total']].describe()
```

Out[12]:

	Unit price	Quantity	Tax 5%	Total
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	55.672130	5.510000	15.379369	322.966749
std	26.494628	2.923431	11.708825	245.885335
min	10.080000	1.000000	0.508500	10.678500
25%	32.875000	3.000000	5.924875	124.422375
50%	55.230000	5.000000	12.088000	253.848000
75%	77.935000	8.000000	22.445250	471.350250
max	99.960000	10.000000	49.650000	1042.650000

Dataset Overview with data.info()

data.info() is a method used to obtain a concise summary of a DataFrame. It provides essential information about the dataset, including the number of entries, the number of non-null values in each column, the data types of the columns, and the memory usage. This method is particularly useful for quickly assessing the structure and completeness of the dataset.

```
 #data.info() is used to  provide information about the DataFrame's structure,
data.info()
```

In [13]:*i*

```
<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   object
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: float64(7), int64(1), object(9)
    memory usage: 132.9+ KB
```

Renaming columns with data.rename

‘data.rename’ is used to rename existing columns. This can be useful when you need to standardize column names, make them more descriptive, or remove any special characters that might interfere with data processing.

The inplace=True parameter in the data.rename() method is used to modify the original DataFrame directly, without creating a new copy. This is particularly useful when you want to apply changes to the DataFrame without needing to reassign it to a new variable. By default, inplace=False, meaning that data.rename() returns a new DataFrame with the changes, leaving the original DataFrame unchanged.

```
In [47]: #Renaming Column Names
Out[47]: import pandas as pd
```

```
         data.rename(columns={'Total':'Total Bill Amt'}, inplace=True) data.head()
```

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415
1	226-313081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200
2	631-413108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155
3	123-191176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880
4	373-737910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085

5 rows × 23 columns

```
#cleaning the Data
import pandas as pd

# Converting the date column to datetime structure with errors='coerce'
data['Date'] = pd.to_datetime(data['Date'], errors='coerce') #it will change al

# Print the DataFrame to see the changes data.head(10)
```

Out[14]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415
1	226-313081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200
2	631-413108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155
3	123-191176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880
4	373-737910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085
5	699-143026	C	Naypyitaw	Normal	Male	Electronic accessories	85.39	7	29.8865

7	315-225665	C	Naypyitaw	Normal	Female	Home and lifestyle	73.56	10	36.7800
8	665-329167	A	Yangon	Member	Female	Health and beauty	36.26	2	3.6260
9	692-925582	B	Mandalay	Member	Female	Food and beverages	54.84	3	8.2260

6	535943	A	Yangon	Member	Female	Electronic accessories	68.84	6 20.6520

Understanding Data Types with .dtype

.dtype is an attribute of a Pandas Series that returns the data type of the elements in the Series. It helps you understand the kind of data stored in a particular column, such as whether the data is numerical, categorical, or textual. This is critical for ensuring that operations performed on the data are appropriate for the data type.

```
In [17]: #printing datatype of Date column

print("Data type of 'Date' column:", data['Date'].dtype)
```

```
Data type of 'Date' column: datetime64[ns] In
```

Converting to Datetime with pd.to_datetime()

pd.to_datetime() is used to convert a string, integer, or list-like object (e.g., a column in a DataFrame) into a Pandas datetime object. This is particularly useful when you have date or time information stored as strings or other formats and need to perform operations such as sorting, filtering, or calculating time

```
[19]:
import pandas as pd

# Converting the time column to datetime structure with a specific format
data['Time'] = pd.to_datetime(data['Time'], format='%H:%M')

# Check the data types of the columns after conversion print("Data
type of 'Time' column:", data['Time'].dtype)
```

```
Data type of 'Time' column: datetime64[ns] In
```

```
[20]:  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   datetime64[ns]
 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](2), float64(7),
     int64(1), object(7) memory usage: 132.9+ KB
```

Counting Orders with ‘value_counts()’

value_counts() is a Pandas method used to count the occurrences of unique values in a Series. This method returns a Series containing counts of unique values, sorted in descending order by default. It’s an efficient way to summarize categorical data, such as the number of orders for different products, the frequency of customer visits, or the distribution of ratings.

```
In [21]: #Analysing the number of orders in each product line data['Product
         line'].value_counts()
```

```
Out[21]: Product line
          Fashion accessories       178
          Food and beverages        174
          Electronic accessories    170
          Sports and travel         166
          Home and lifestyle        160
          Health and beauty         152
          Name: count, dtype: int64
```

```
In [22]: #Analysing the number of orders by different customer type data['Customer
         type'].value_counts()
```

```
Out[22]: Customer type Member
          501
          Normal    499
          Name: count, dtype: int64
```

```
In [23]: #Analysing No. of orders by Gender:Male & Female
         data['Gender'].value_counts()
```

```
Out[23]: Gender
          Female    501
          Male      499
          Name: count, dtype: int64
```

```
In [24]: #City wise orders data['City'].value_counts()
```

```
Out[24]: City
          Yangon       340
          Mandalay     332
          Naypyitaw    328
          Name: count, dtype: int64
```

In [10]:

```
import pandas as pd data=pd.read_csv("supermarket

sales.csv")

# Group by 'Payment' and count the occurrences payment_counts
= data['Payment'].value_counts()

# Finding the payment method with the highest count
 highest_payment_count = payment_counts.max()
 highest_payment_method = payment_counts.idxmax()
print(f"The payment method used the highest is '{highest_payment_method}' with
```

```
{ The payment method used the highest is 'Ewallet' with 345 occurrences.
```

Grouping Data with data.groupby()

data.groupby() is a powerful function in Pandas used for grouping data based on one or more columns, and then performing operations on each group separately. This is especially useful in data analysis when you need to aggregate data, apply functions, or analyze subsets of your dataset based on categorical variables.

Counting Grouped Data with .size() and .reset_index()

- **.size()**: This function in Pandas is used after a groupby() operation to count the number of elements in each group. Unlike count(), which counts non-null values for specific columns, .size() counts the total number of entries in each group, including those with missing values.
- **.reset_index()**: After using .size(), the result is a Series with a multi-index (based on the grouping columns). .reset_index() is used to convert this Series into a DataFrame by resetting the index and making the grouping columns into regular columns.

In [9]:

```
#Analysing which Payment method is used prominently by which customer

import pandas as pd

e # Group by 'Customer type' and 'Payment', and counting the number of occurrences
customer_analysis = data.groupby(['Customer type', 'Payment']).size().reset_ind

# Printing the result print(customer_analysis)
```

```
Customer type      Payment   Count 0
Member          Cash    168
        1   Member   Credit card    172
        2   Member       Ewallet    161
        3   Normal          Cash    176
        4   Normal   Credit card    139
        5   Normal       Ewallet    184 In [36]:
```

```
#Analysing in  which year the sales was highest import
pandas as pd

# Convert 'Date' column to datetime data['Date']
= pd.to_datetime(data['Date'])

# Extract year from 'Date' column data['Year'] =
data['Date'].dt.year

# Group by year and calculate total sales amount sales_by_year
= data.groupby('Year')['Total'].sum()

# Find the year with the highest sales highest_sales_year
= sales_by_year.idxmax() highest_sales_amount =
sales_by_year.max()

print("Year with the highest sales:", highest_sales_year) print("Total sales
amount in the highest sales year:", highest_sales_amount)
```

```
Year with the highest sales: 2019
```

In [35]:

```
#Analysing at which date the sales was highest import
pandas as pd

# Grouping sales by date and calculating total sales amount sales_by_date
= data.groupby('Date')['Total'].sum()

# Finding the date with the highest sales highest_sales_date
= sales_by_date.idxmax() highest_sales_amount =
sales_by_date.max()

#Printing the result
print("Date with the highest sales:", highest_sales_date) print("Total sales
amount on the highest sales date:", highest_sales_amount)
```

```
 Total sales amount in the highest sales year: 322966.749
Date with the highest sales: 2019-03-09 00:00:00 Total sales
amount on the highest sales date: 7474.0470000000005 In
```

[45]:

```
#CIty Wise Sales Analysis
import pandas as pd

# Group by city and calculate total sales amount city_wise_sales =
data.groupby('City')['Total'].sum().reset_index()

# Display city-wise sales print("City-wise
Sales Analysis:") print(city_wise_sales)
```

```
City-wise Sales Analysis:
City        Total 0   Mandalay
106197.6720
1      Naypyitaw  110568.7065
2      Yangon  106200.3705
```

Analysis of sales in cities with Product line

In [71]:

```
import pandas as pd

# Read the data data =
pd.read_csv("Supermarket sales.csv")

# Step 1: Calculate total sales for each city city_sales
= data.groupby('City')['Total'].sum()

# Step 2: Identify city with highest sales
max_sales_city = city_sales.idxmax() min_sales_city
= city_sales.idxmin()

# Step 3: Determining product line with highest and lowest sales in the city wit
max_sales_city_data = data[data['City'] == max_sales_city] product_sales_max_city
= max_sales_city_data.groupby('Product line')['Total'].s max_sales_product_line =
u product_sales_max_city.idxmax() min_sales_product_line =
product_sales_max_city.idxmin()

#minimum min_sales_city_data = data[data['City'] == min_sales_city]
product_sales_min_city
= min_sales_city_data.groupby('Product line')['Total'].s
maximum_sales_product_line = product_sales_min_city.idxmax()
u minimum_sales_product_line = product_sales_min_city.idxmin()

# Display results



print("City with the highest sales:", max_sales_city) print("Product line with the
highest sales in", max_sales_city, ":", max_sales_p print("Product line with the
lowest sales in", max_sales_city, ":", min_sales_pr print("City with the lowest
sales:", min_sales_city) print("Product line with the highest sales in",
min_sales_city, ":", maximum_sal print("Product line with the lowest sales in",
min_sales_city, ":", minimum_sale
```

```
City with the highest sales: Naypyitaw
Product line with the highest sales in Naypyitaw : Food and beverages
Product line with the lowest sales in Naypyitaw : Home and lifestyle
City with the lowest sales: Mandalay
Product line with the highest sales in Mandalay : Sports and travel
Product line with the lowest sales in Mandalay : Food and beverages
```

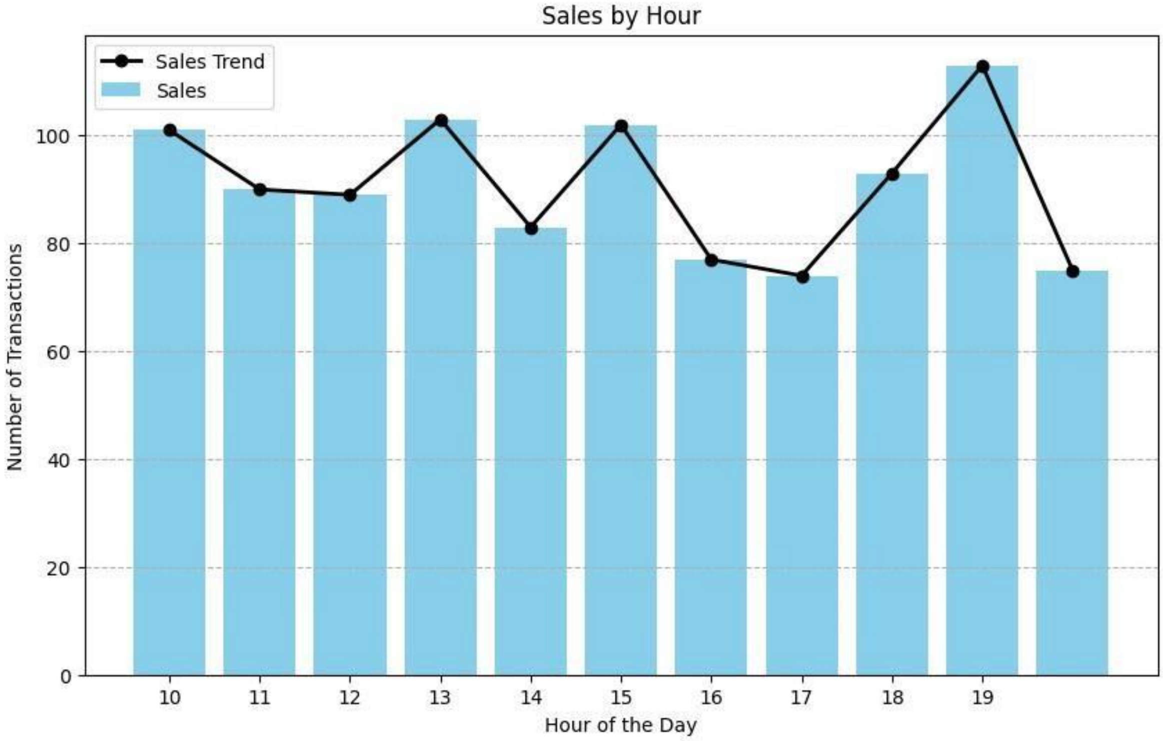
Data

Visualization

Now, lets start by creating Visulaizations:
Here is a step-by-step guide to creating a bar chart using Matplotlib, starting from loading the dataset to customizing the chart:
1.Load the Dataset: Import the necessary libraries and load your dataset.
2.Prepare the Data: Extract the data you need for the plot.
3.Create a Bar Chart: Use plt.bar() to create the bar chart.
4.Label the Axes: Add labels to the x and y axes using plt.xlabel() and plt.ylabel().
5.Add a Title: Use plt.title() to add a title to the chart.
6.Customize Markers: Customize markers for the bars (if needed) using the marker parameter marker='o'
7.Add a Legend: Use plt.legend() to add the legend to the chart.
8.Add a Grid: Use plt.grid() to add a grid to the plot for better readability.
9.Show the Plot: Display the plot using plt.show().
These steps are standard for creating most charts using Matplotlib. The primary differences among charts lie in the specific plotting functions used (e.g., plt.bar() for bar charts, plt.plot() for line charts, etc.).

In [86]:

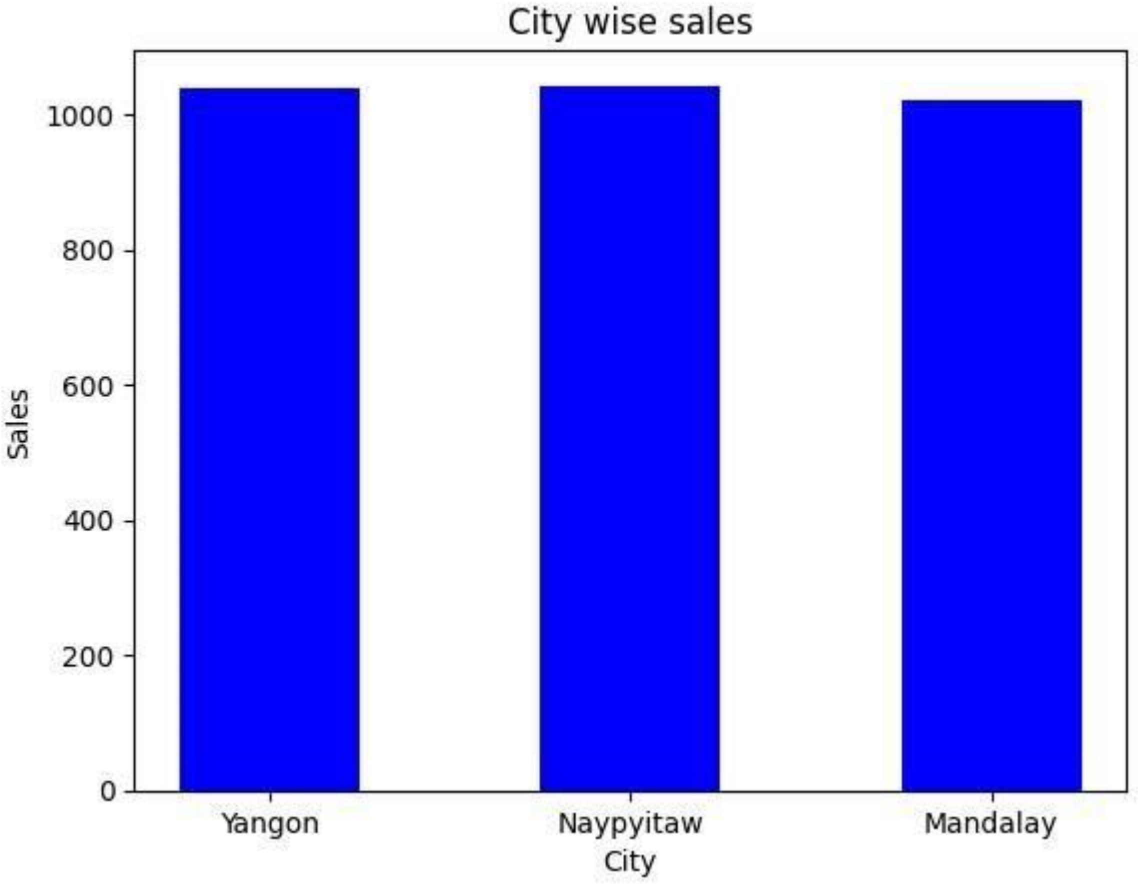
```
import pandas as pd
import matplotlib.pyplot as plt data =

pd.read_csv("supermarket sales.csv")

# Convert 'Time' column to datetime data['Time']
= pd.to_datetime(data['Time'])

# Extract hour from 'Time' column data['Hour'] =
data['Time'].dt.hour

# Group by hour and calculate total sales amount or number of transactions
sales_by_hour = data.groupby('Hour').size()  # You can also use sum() if you ha
v
# Plotting the graph plt.figure(figsize=(10, 6))
 plt.bar(sales_by_hour.index, sales_by_hour, color='skyblue', label='Sales')
plt.plot(sales_by_hour.index, sales_by_hour, color='black', marker='o', label='
plt.title('Sales by Hour')
S
plt.xlabel('Hour of the Day')
 plt.ylabel('Number of Transactions')
# Update ylabel accordingly if using sale plt.xticks(range(10, 20))  # Set x-
axis ticks from 10 to 23
s
 plt.legend() plt.grid(axis='y', linestyle='--')
 plt.show()
```



In [80]:

```
#City wise sales
import pandas as pd
import matplotlib.pyplot as plt city_wise_sales =

data.groupby('City')['Total'].sum().reset_index() print("City-

wise Sales Analysis:") print(city_wise_sales)

plt.bar(data['City'], data['Total'], width=0.5,color='Blue')

plt.title("City wise sales")
plt.xlabel("City")
plt.ylabel("Sales")
plt.show()
```

```
City-wise Sales Analysis:
City        Total 0
Mandalay  106197.6720
      1     Naypyitaw
               110568.7065
      2     Yangon  106200.3705
```

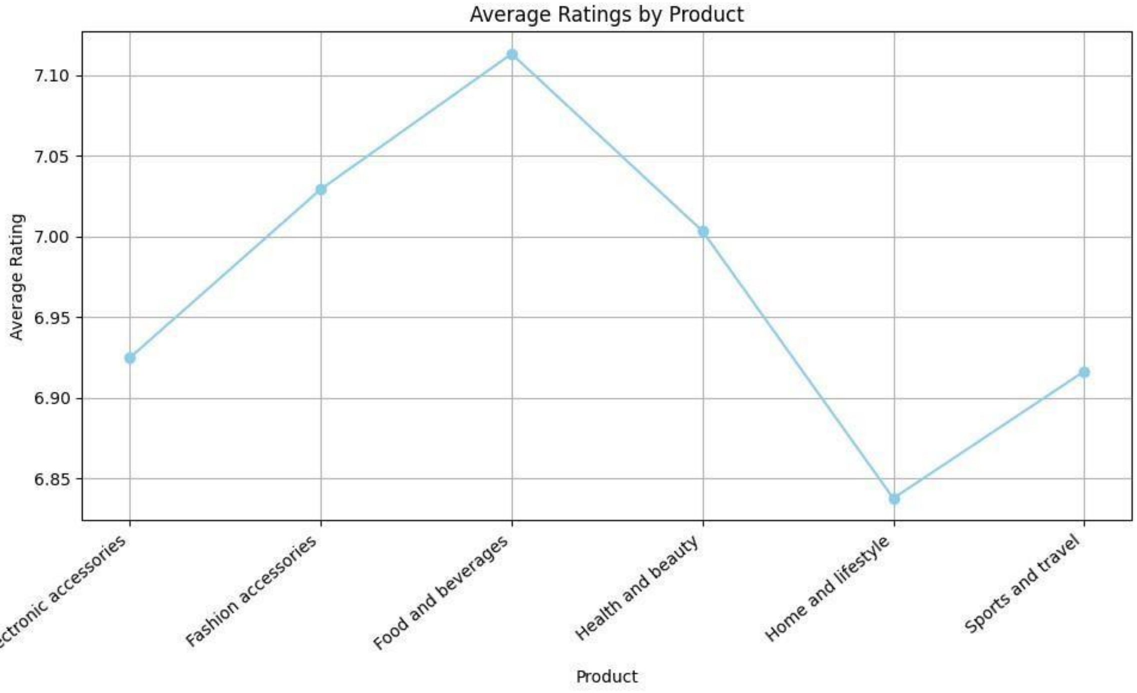


Customer Satisfaction Analysis

```
In [81]:
```

```
#Analysing customer ratings for each product line
i
import pandas as pd data=pd.read_csv("Supermarket
sales.csv")

# Group by 'Product' and calculate the average rating average_ratings
= data.groupby('Product line')['Rating'].mean()
t
# Print the average ratings for each product print(average_ratings)

plt.figure(figsize=(10, 6))
plt.plot(average_ratings.index, average_ratings, marker='o', color='skyblue', l

plt.title('Average Ratings by Product') plt.xlabel('Product') plt.ylabel('Average
Rating') plt.xticks(rotation=40, ha='right')  # Rotate x-axis labels for better
visibili plt.grid(True) plt.tight_layout() plt.show()
```

```
Product line
Electronic accessories    6.924706
Fashion accessories       7.029213
Food and beverages        7.113218
Health and beauty         7.003289
Home and lifestyle        6.837500
Sports and travel         6.916265
Name: Rating, dtype: float64
```

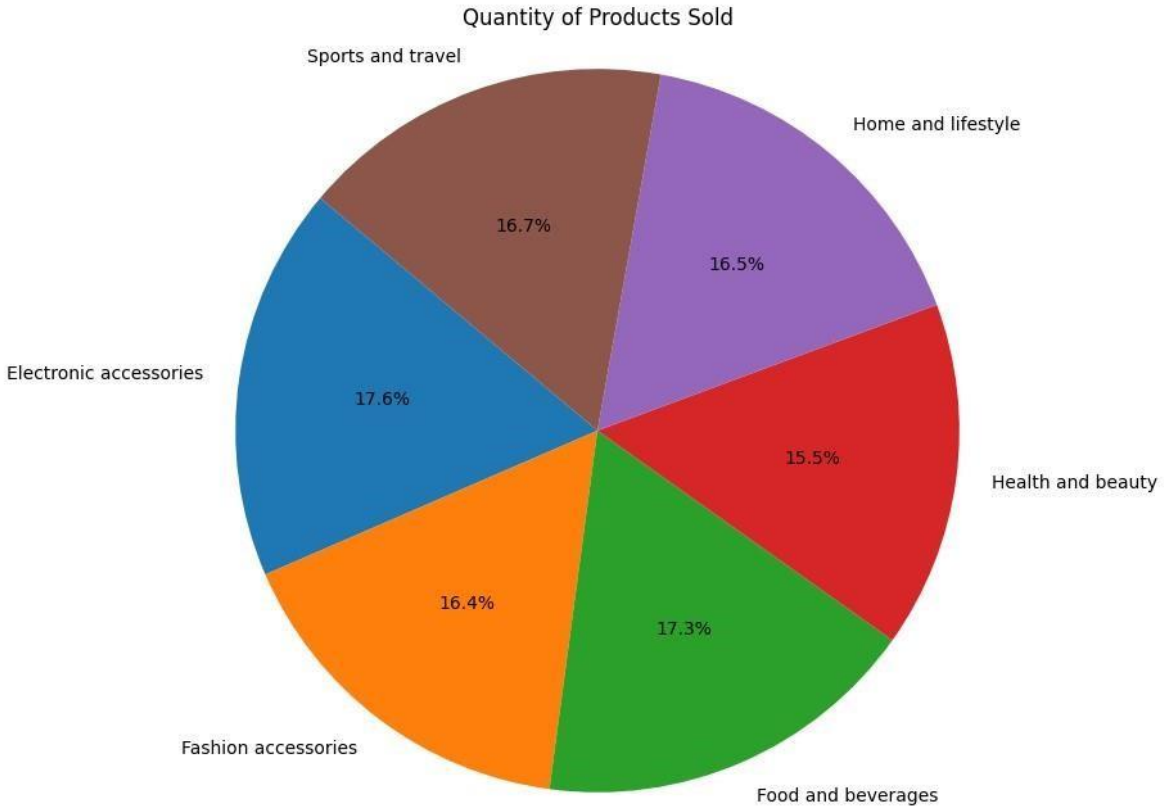


Comparative Analysis of Product Line :Quantity Sold

```
In [57]:
r
```

```
import pandas as pd data=pd.read_csv("Supermarket sales.csv")

product_quantity= data.groupby('Product line')['Quantity'].sum()

print(product_quantity)

# Plotting the pie chart plt.figure(figsize=(8, 8)) plt.pie(product_quantity,
labels=product_quantity.index, autopct='%1.1f%%', sta plt.title('Quantity of
Products Sold') plt.axis('equal')  # Equal aspect ratio ensures that pie is
drawn as a circle. plt.show()
```

```
Product line
Electronic accessories    971
Fashion accessories       902
Food and beverages        952
Health and beauty         854
Home and lifestyle        911
Sports and travel         920
Name: Quantity, dtype: int64
```

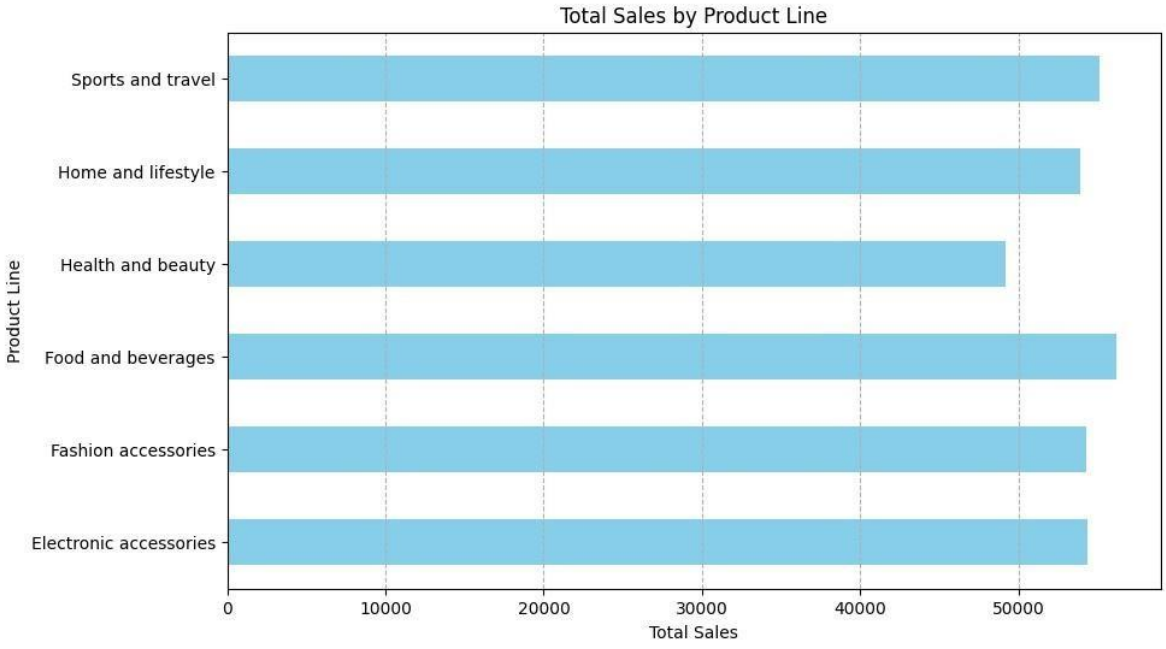


Analysing Sales of each product Line

In [64]:

```
import pandas as pd data=pd.read_csv("Supermarket

sales.csv") product_sales= data.groupby('Product

line')['Total'].sum() print(product_sales)

# Plotting the horizontal bar chart
plt.figure(figsize=(10, 6))
product_sales.plot(kind='barh', color='skyblue')
plt.title('Total Sales by Product Line')
plt.xlabel('Total Sales') plt.ylabel('Product
Line') plt.grid(axis='x', linestyle='--')
plt.show()
```

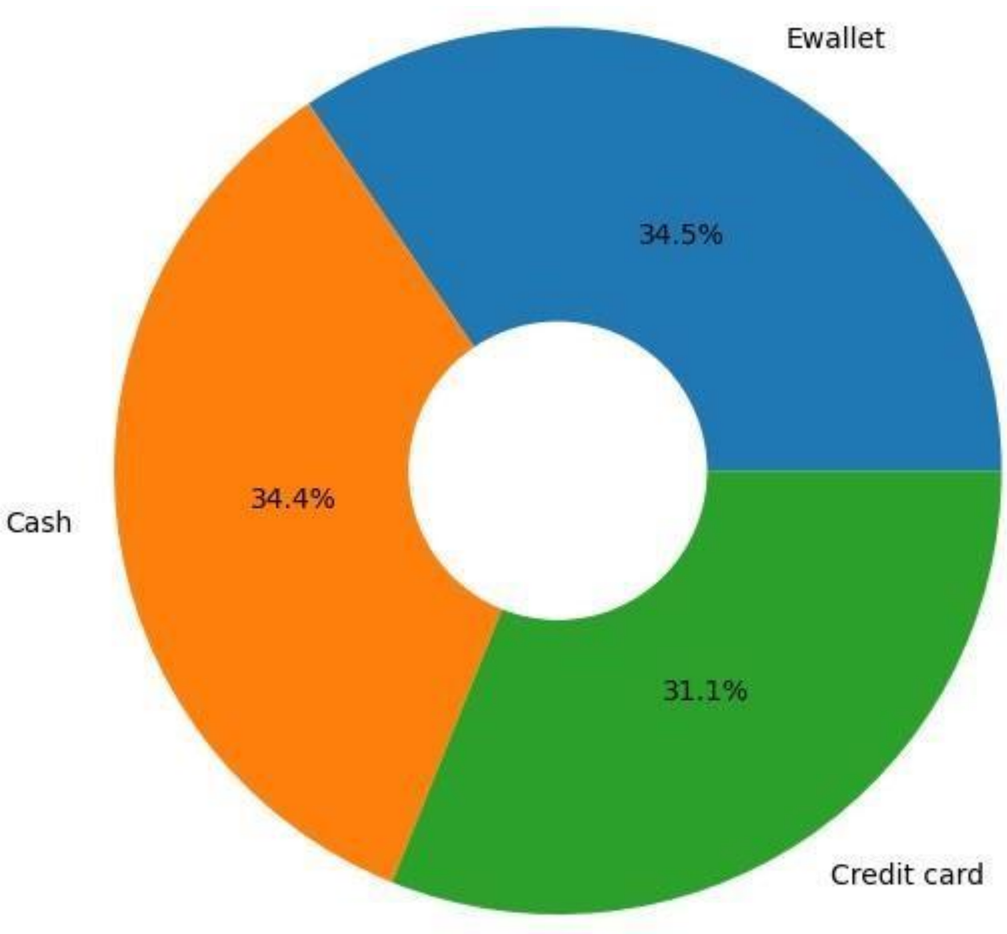
```
Product line
Electronic accessories    54337.5315
Fashion accessories       54305.8950
Food and beverages        56144.8440
Health and beauty         49193.7390
Home and lifestyle        53861.9130
Sports and travel         55122.8265
Name: Total, dtype: float64
```



Analysis of Payment mode being used by customers

In [84]:

```
#donut pie chart
plt.pie(data.Payment.value_counts(),
autopct="%.1f%%", radius=1.5,
labels=['Ewallet','Cash','Credit card']) circle = plt.Circle((0,0),
0.5, color='white') plot=plt.gcf() plot.gca().add_artist(circle)
plt.show()
```


Ewallet
Cash
Credit card