Alexandria University

Faculty of Computers and data science

Department of Data Science

Course Title: Data Mining 2023 - 2024

Data Mining Project on

Supermarket sales data

In

Data Mining Course

02-24-00206

Members Names and roles

Name	ID	Role
أحمد حسين حسن دويدار	20225926030	Data Preprocessing
محمد ياسر القطب	22010234	Exploratory data Analysis (EDA)
ماثيو بهجت جريس عبيد منصور	22011965	K-Medoids Clustering
معاذ مصطفى عبدالحميد مصطفى	22010263	Hierarchal Clustering
حمزة حسين يوسف عمران	22011501	Evaluation and interpretation
خالد محمد جوده محمد شافعى	22011646	Documentation and reporting

• Introduction

- In today's dynamic retail landscape, supermarkets play a vital role in catering to the different needs of customers. With the available advanced technologies and increase of data, supermarkets are presented with unprecedented opportunities to optimize their processes, improve customer experience, and increase profit. Data mining can be a powerful analytical approach that could empower organizations to extract valuable insights from different datasets, resulting in informed decision-making and strategic planning.

- This report presents a comprehensive analysis using different data mining techniques on a supermarket dataset. The objectives of this project encompass data preprocessing, exploratory data analysis (EDA), and the application of K-Medoids and Hierarchical clustering algorithms. We aim to find underlying patterns, and segment customers based on their purchasing behaviors.

• Problem Statement

-In such a competitive supermarket industry, understanding customers preferences, identifying market segments, and optimizing inventory management are critical for continuous success. Traditional and old approaches usually fail to show full potential of available data. Therefore, the problem we aim to address is to **optimize supermarket operations** by leveraging data mining techniques. We aim to optimize supermarket operations by effectively segmenting customers, identifying purchasing patterns and seeking marketing strategies and product offerings to meet different needs of customers.

• Objectives

-*Data preprocessing*: Before starting the analysis, the dataset requires preprocessing to ensure its quality, completeness, and readiness for exploration and further modeling.

-*Exploratory Data Analysis (EDA)*: Through it, we aim to gain insights into the dataset's structure, identify trends, outlies, and correlations, and lay foundation for future analysis.

-*K-Medoids Clustering*: Utilizing such algorithm, we divide customers into distinct clusters based on similarities in their purchasing patterns. This enables targeted marketing strategies and personalized services.

-*Hierarchical Clustering*: providing a hierarchical decomposition of the dataset, allowing us to understand the structure and relationships among customers and products.

-*Evaluation and Interpretation*: finally, evaluating the efficiency of the clustering algorithms in effectively segmenting customers and interpreting the clusters to derive actionable insights for the supermarket.

(1) Data Preprocessing:

- Data Preprocessing is a crucial initial step in any data mining project, aiming to ensure the quality, consistency, and suitability of the dataset for analysis. This process involves several key tasks to prepare the data for further exploration and modeling:

• Data Cleaning where we address missing values, outliers, and inconsistencies in the dataset. Techniques such as imputation, removal of duplicates, and outlier detection.

• Feature Selection where we Identify and select relevant features that are most informative for the analysis. It might involve removing redundant or irrelevant attributes, or even changing new features through transformation.

• Dealing with Imbalanced data: We address imbalance issues if existing in the dataset.

• Data Transformation as We Transform variables to meet assumptions of statistical models or improve the distribution of data.

- In our python code, we first need to import our libraries that we will use throughout the project as follows:

```
import pandas as pd
import seaborn as sns
import numpy as np
import sklearn
import matplotlib.pyplot as plt
import warnings
import math
from datetime import datetime
warnings.filterwarnings("ignore")
%matplotlib inline
from sklearn_extra.cluster import KMedoids
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc
from sklearn.preprocessing import StandardScaler
```

- Now, we begin our data pre-processing phase by importing the dataset into our code using pd.read_csv from pandas library and then typing data.info() to get a brief about our data as follows:

```
data=pd.read_csv(r"C:\Users\LapCell\Downloads\supermarket_sales - Sheet1.csv")
data.dtypes
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   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
```

- In our dataset, we notice an attribute called Date that is not of type *(date)*. To change that, we first import datatime and use it to make that change. Then, we verify that by typing data.dtypes as follows:

```
print(data.head())
from datetime import datetime
data["Date"]=pd.to_datetime(data["Date"]);
data.dtypes
```

```
    Invoice ID Branch       City Customer type  Gender  \
0  750-67-8428      A     Yangon        Member  Female
1  226-31-3081      C  Naypyitaw        Normal  Female
2  631-41-3108      A     Yangon        Normal    Male
3  123-19-1176      A     Yangon        Member    Male
4  373-73-7910      A     Yangon        Normal    Male

             Product line  Unit price  Quantity   Tax 5%     Total       Date  \
0       Health and beauty       74.69         7  26.1415  548.9715   1/5/2019
1  Electronic accessories       15.28         5   3.8200   80.2200   3/8/2019
2      Home and lifestyle       46.33         7  16.2155  340.5255   3/3/2019
3       Health and beauty       58.22         8  23.2880  489.0480  1/27/2019
4       Sports and travel       86.31         7  30.2085  634.3785   2/8/2019

    Time      Payment    cogs  gross margin percentage  gross income  Rating
0  13:08      Ewallet  522.83                 4.761905       26.1415     9.1
1  10:29         Cash   76.40                 4.761905        3.8200     9.6
2  13:23  Credit card  324.31                 4.761905       16.2155     7.4
3  20:33      Ewallet  465.76                 4.761905       23.2880     8.4
4  10:37      Ewallet  604.17                 4.761905       30.2085     5.3
Invoice ID                       object
Branch                           object
City                             object
Customer type                    object
Gender                           object
Product line                     object
Unit price                      float64
Quantity                          int64
Tax 5%                          float64
Total                           float64
Date                     datetime64[ns]
Time                             object
Payment                          object
cogs                            float64
gross margin percentage         float64
gross income                    float64
Rating                          float64
dtype: object
```

- Next, we check if there's any null values using is.null() command:

```
cols=data.columns
data[cols].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
```

- Seeing that we don't have any null values, We then move to checking if we have any duplicates using .duplicated().sum() command:

finding the duplicated values

```
data.duplicated().sum()
```

```
0
```

- By also having 0 duplicates, it shows that our data was moderately clean and well-collected. A good practice is to check statistics about our dataset. Doing so can help us understand data distribution knowing the *Mean, Median mode and standard deviation* that provide insights into the central tendency and variability of the dataset as well as Identifying outliers. To do so, we can use the .describe() command:

finding the statistics about the data

```
data.describe()
```

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1.000000e+03	1000.000000	1000.00000
mean	55.672130	5.510000	15.379369	322.966749	307.58738	4.761905e+00	15.379369	6.97270
std	26.494628	2.923431	11.708825	245.885335	234.17651	6.131498e-14	11.708825	1.71858
min	10.080000	1.000000	0.508500	10.678500	10.17000	4.761905e+00	0.508500	4.00000
25%	32.875000	3.000000	5.924875	124.422375	118.49750	4.761905e+00	5.924875	5.50000
50%	55.230000	5.000000	12.088000	253.848000	241.76000	4.761905e+00	12.088000	7.00000
75%	77.935000	8.000000	22.445250	471.350250	448.90500	4.761905e+00	22.445250	8.50000
max	99.960000	10.000000	49.650000	1042.650000	993.00000	4.761905e+00	49.650000	10.00000

- and to find the correlation between quantitative values, we use .corr(). It's considered to be a crucial aspect of EDA, and we use it to identify relationships, predictive modeling data as features with high correlation are often strong predictors. And finally for imputation as if two variables are highly correlated, missing values in one of them can be estimated using values from the correlated one.

```
data.corr()
```

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
Unit price	1.000000	0.010778	0.633962	0.633962	0.633962	NaN	0.633962	-0.008778
Quantity	0.010778	1.000000	0.705510	0.705510	0.705510	NaN	0.705510	-0.015815
Tax 5%	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.036442
Total	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.036442
cogs	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.036442
gross margin percentage	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gross income	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.036442
Rating	-0.008778	-0.015815	-0.036442	-0.036442	-0.036442	NaN	-0.036442	1.000000

- We can make a good use out of the *correlation matrix* to get more insights of our data by defining a function called plotCorrelationMatrix to calculate the correlation matrix,

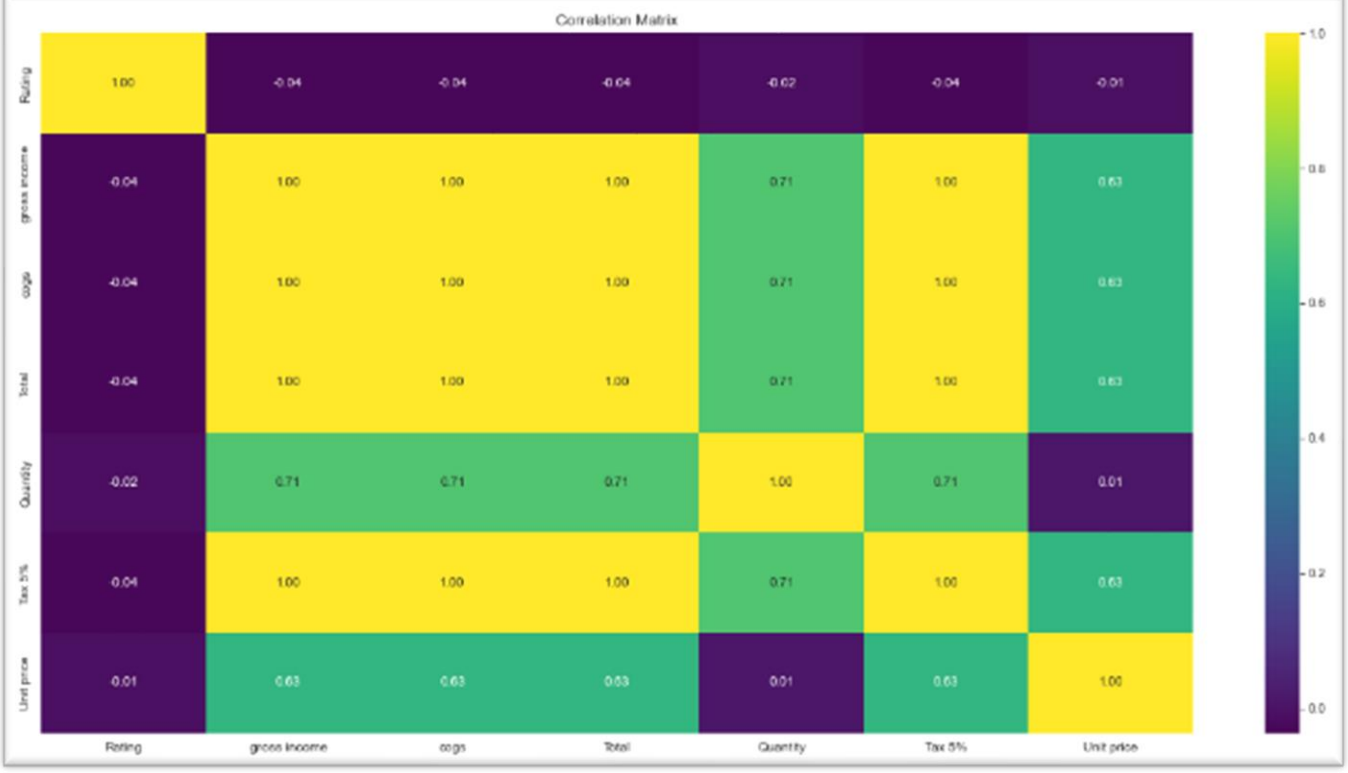
• there's approximately no correlation between ratings and other variables and between unit and quantity.

• the correlation between 'Total', 'Gross income', 'Cogs', and 'Tax 5%' is strong and positive.

• and the correlation is medium and positive between the parts where the value is 0.69.

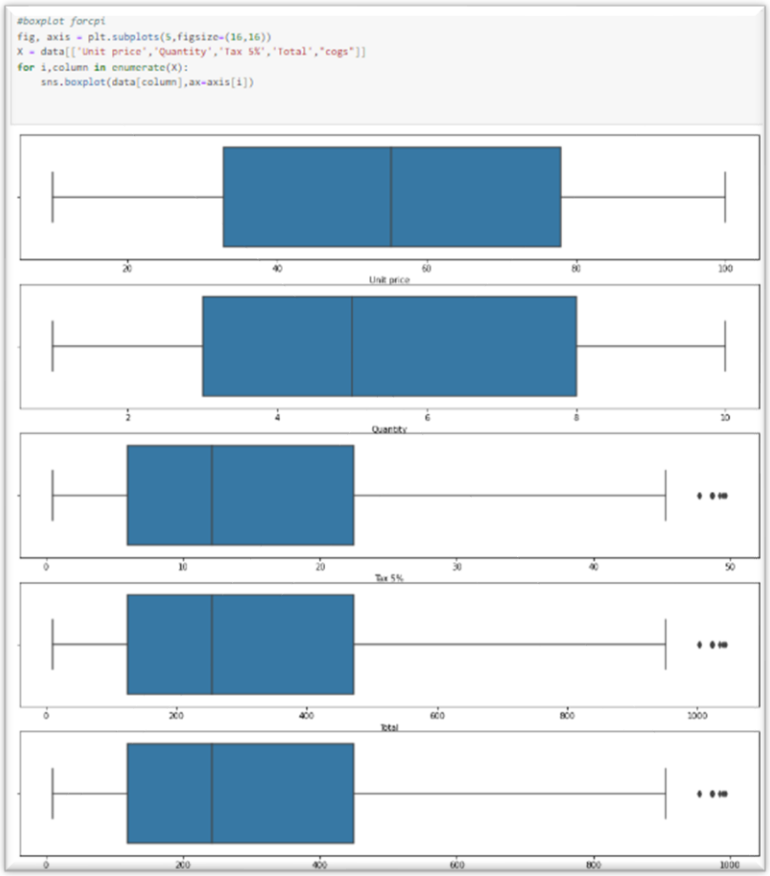
- so we can create the heatmap and then display the plot as follows:

```
df = data[
    [
        "Rating",
        "gross income",
        "cogs",
        "Total",
        "Quantity",
        "Tax 5%",
        "Unit price",
    ]
]


def plotCorrelationMatrix(df, figsize):
    # Calculate the correlation matrix
    corr_matrix = df.corr()

    # Set the figure size
    plt.figure(figsize=figsize)

    # Create the heatmap with a different colormap (viridis)
    sns.heatmap(corr_matrix, annot=True, cmap="viridis", fmt=".2f")

    # Add title
    plt.title("Correlation Matrix")

    # Display the plot
    plt.show()


# Replace df1 with your DataFrame name
plotCorrelationMatrix(df, (20, 10))
```



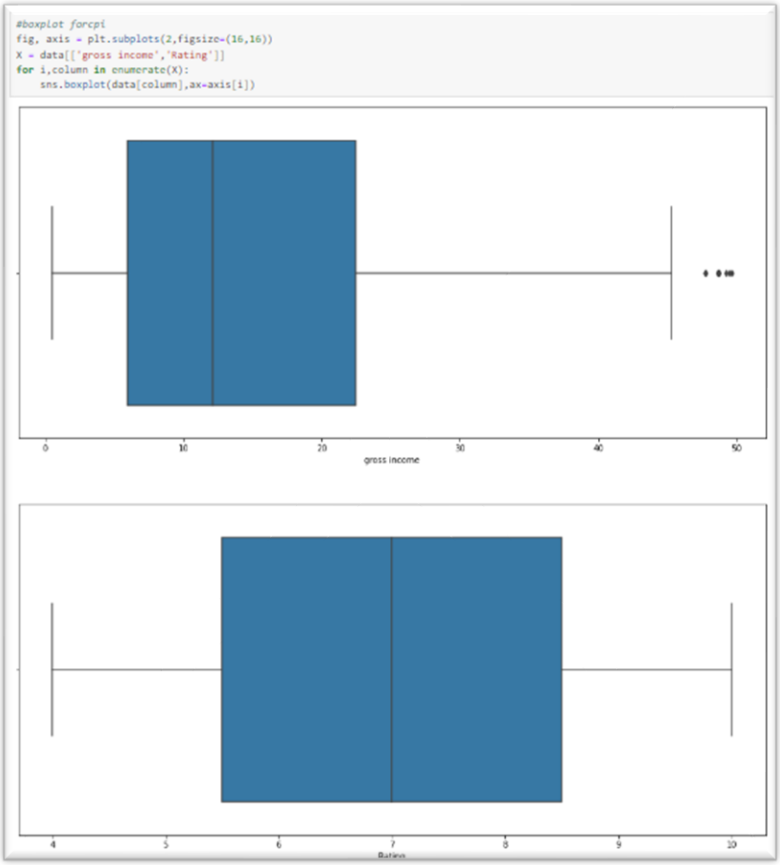
- We can also make boxplots for quantitative data. As a result, we found outliers in **total**, **cogsand**, and **Tax 5%**:

```
#boxplot forcpi
fig, axis = plt.subplots(5,figsize=(16,16))
X = data[['Unit price','Quantity','Tax 5%','Total',"cogs"]]
for i,column in enumerate(X):
    sns.boxplot(data[column],ax=axis[i])
```



-Same goes for **Gross income**:

```
#boxplot forcpi
fig, axis = plt.subplots(2,figsize=(16,16))
X = data[['gross income','Rating']]
for i,column in enumerate(X):
    sns.boxplot(data[column],ax=axis[i])
```



- We can perform some calculations on the quantitative data such as Median to use it in the removal of outliers using .median() as follows:

```
# Calculate the median value of each numeric column
medians = data.select_dtypes(include=['float64', 'int64']).median()

print(medians)
```

```
Unit price                 55.230000
Quantity                    5.000000
Tax 5%                     12.088000
Total                     253.848000
cogs                      241.760000
gross margin percentage     4.761905
gross income               12.088000
Rating                      7.000000
dtype: float64
```

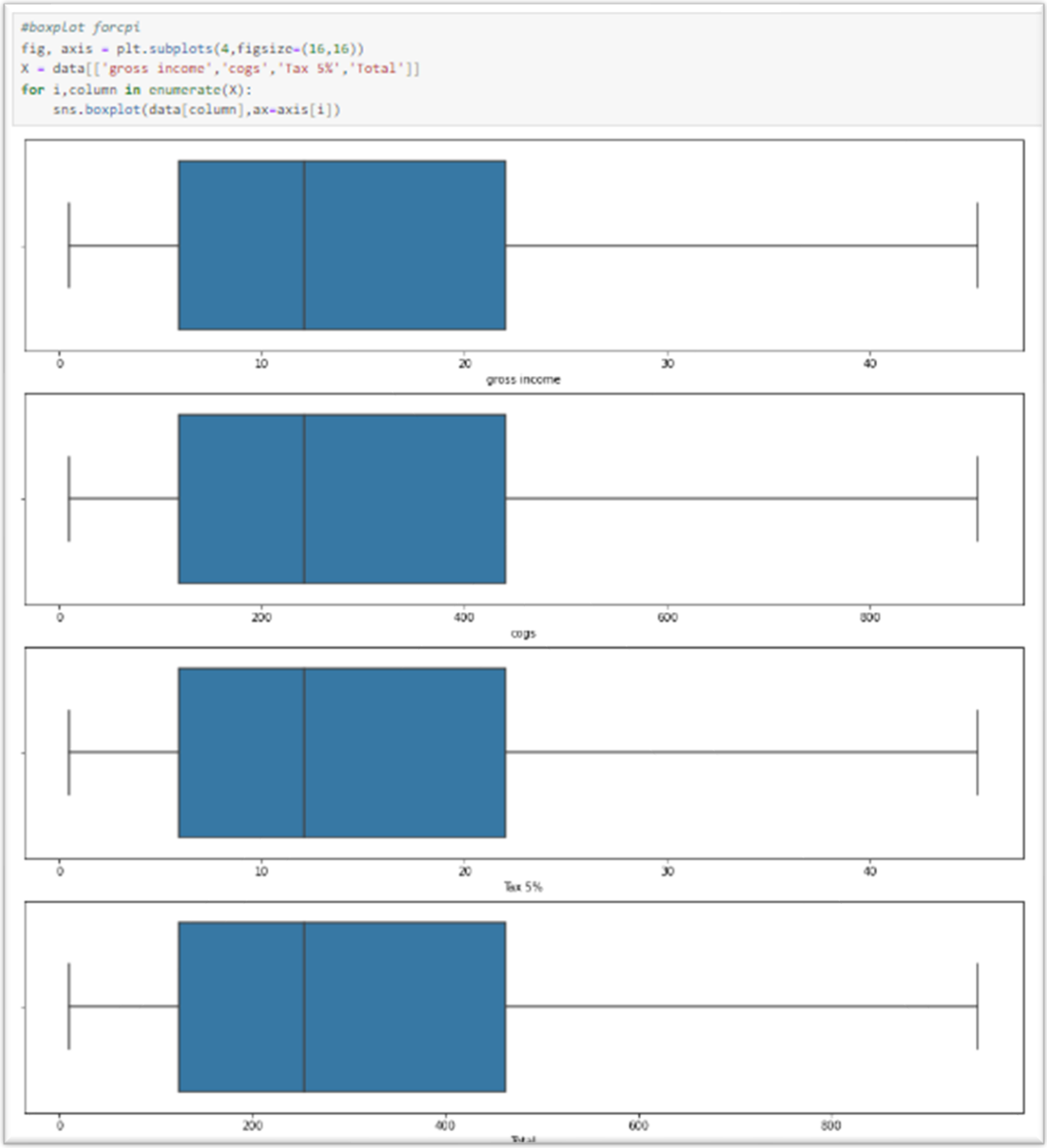
- Using the interquartile range method (IQR), we can identify outliers in each columns and then replace them with the median. After that, we can verify that the replacement was done using the .describe() command as follows:

```
# Identify outliers using the interquartile range (IQR) method for each column
for col in medians.index:
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    outliers = data[(data[col] < Q1 - 1.5*IQR) | (data[col] > Q3 + 1.5*IQR)]

    # Replace outliers with the median value of the column
    data.loc[outliers.index, col] = medians[col]

# Verify that outliers have been replaced
print(data.describe())
```

```
        Unit price     Quantity       Tax 5%        Total         cogs  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean     55.672130     5.510000    15.049521   316.039941   300.990420
std      26.494628     2.923431    11.271937   236.710686   225.438749
min      10.080000     1.000000     0.508500    10.678500    10.170000
25%      32.875000     3.000000     5.924875   124.422375   118.497500
50%      55.230000     5.000000    12.084000   253.764000   241.680000
75%      77.935000     8.000000    22.041000   462.861000   440.820000
max      99.960000    10.000000    45.325000   951.825000   906.500000

       gross margin percentage  gross income      Rating
count             1.000000e+03   1000.000000  1000.00000
mean              4.761905e+00     15.049521     6.97270
std               6.131498e-14     11.271937     1.71858
min               4.761905e+00      0.508500     4.00000
25%               4.761905e+00      5.924875     5.50000
50%               4.761905e+00     12.084000     7.00000
75%               4.761905e+00     22.041000     8.50000
max               4.761905e+00     45.325000    10.00000
```

- Going back to our Boxplots, we can verify that every outlier has been removed successfully and the data is now more consistent:

```
#boxplot forcpi
fig, axis = plt.subplots(4,figsize=(16,16))
X = data[['gross income','cogs','Tax 5%','Total']]
for i,column in enumerate(X):
    sns.boxplot(data[column],ax=axis[i])
```

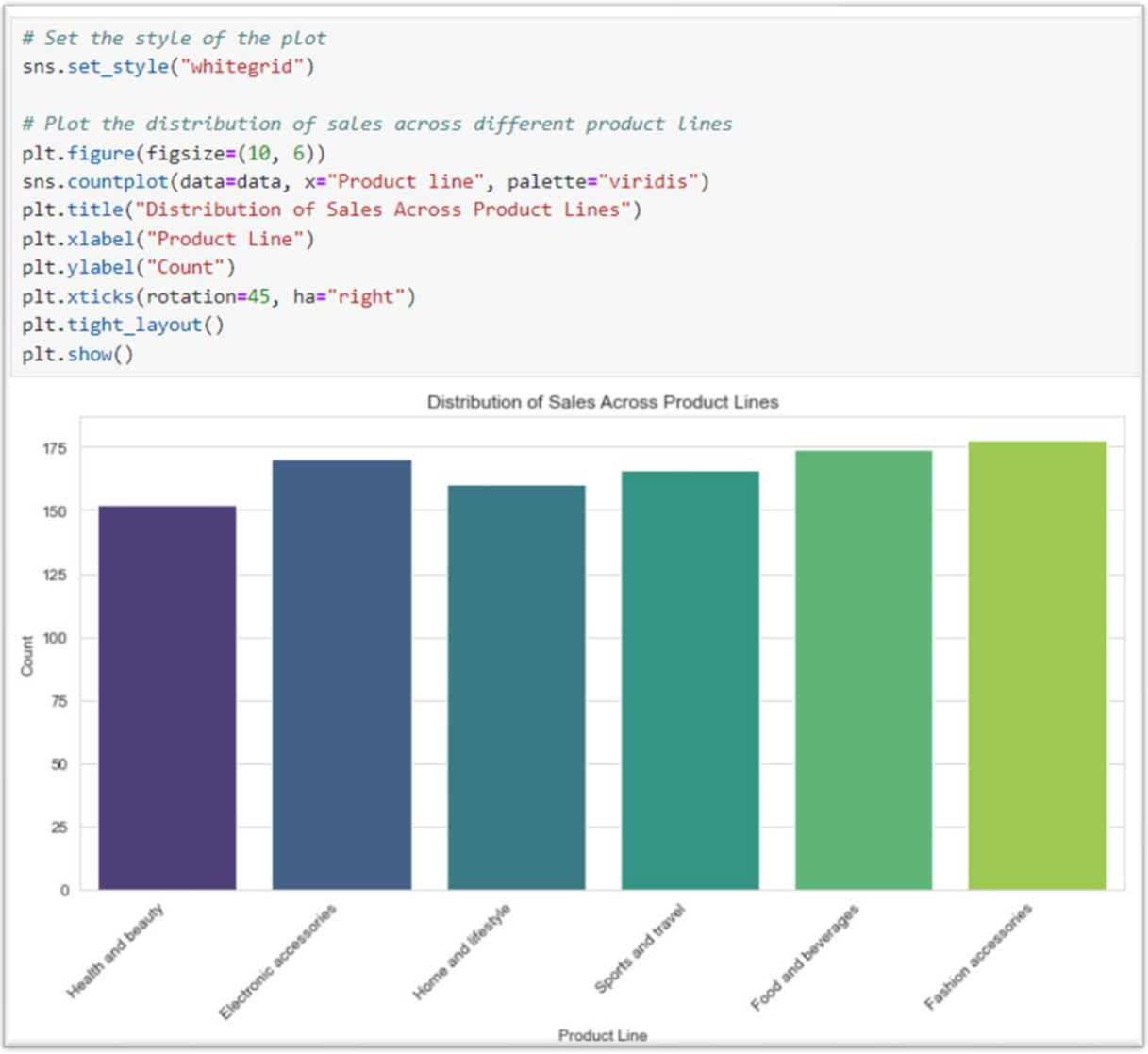


(2) Exploratory Data Analysis (EDA):

- EDA or Exploratory data analysis is another important phase in the data mining process aimed to gaining insights into the dataset's characteristics, identifying patterns, and formulating hypotheses for further analysis. It involves different techniques and visualizations to understand the structure, distribution, and relationships within the data. Some key components are:

- As a beginning, we could start by exploring the number of transactions by each branch in our data set using some commands from seaborn library as follows:

```
# Set the style of the plot
sns.set_style("whitegrid")

# Plot the distribution of sales across different product lines
plt.figure(figsize=(10, 6))
sns.countplot(data=data, x="Product line", palette="viridis")
plt.title("Distribution of Sales Across Product Lines")
plt.xlabel("Product Line")
plt.ylabel("Count")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



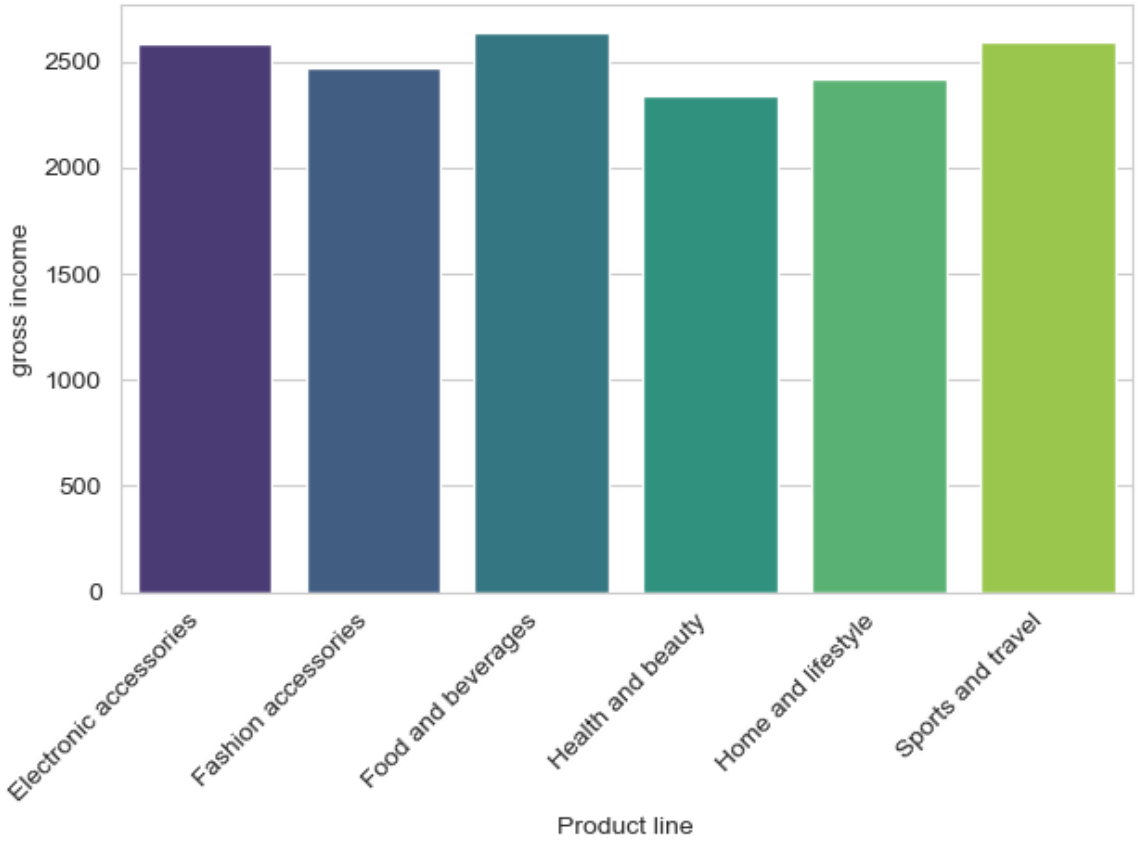
•The graph shows the distribution of sales across different product lines, each bar represents a product line, and the height of each bar represents the count of sales for that product line.

• from this graph, we can identify the max product line sales and the minimum one.

• and as it's clear, we can see that 'Fashion accessories' product line has the most times of sales with more than 175 times and the lowest is 'Health and Beauty' product line with 150 times. So, in general, the product lines times of purchase is between 150 and 175 or slightly more.

- Taking a look at which product line has the most total Gross income:

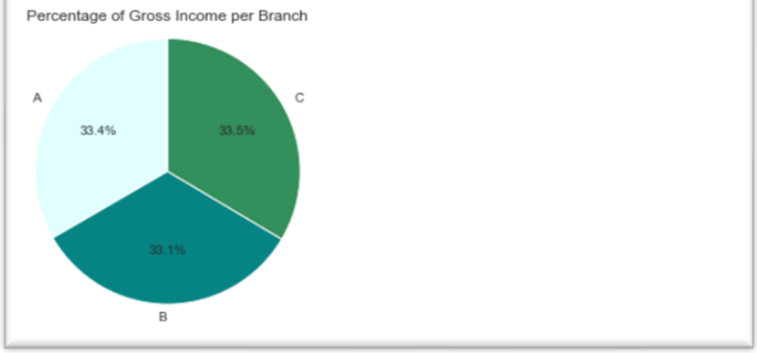
```
gross_income_by_product = (
    data.groupby("Product line")["gross income"].sum().reset_index()
)
sns.barplot(
    data=gross_income_by_product, x="Product line", y="gross income", palette="viridis"
)
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



- *this graph shows the total income for each product line, and that the highest in total sales is 'Food and Beverages' line with more than 2500. The lowest one is 'Health and Beauty' with more than 2300.*
- *as for the rest, they're close to the range of 2400 – 2500 which shows stability in Gross income.*

- A better look at it would be using a pie chart and making it by branch:

```
total_gross_income_per_branch = data.groupby("Branch")["gross income"].sum()

# Plotting a pie chart
sns.set_style("darkgrid")
plt.figure(figsize=(4, 4))  # Adjust figure size if needed
plt.pie(
    total_gross_income_per_branch,
    labels=total_gross_income_per_branch.index,
    autopct="%1.1f%%",
    startangle=90,
    colors=["lightcyan", "teal", "seagreen"],
)
plt.title("Percentage of Gross Income per Branch")
plt.axis("equal")  # Equal aspect ratio ensures that pie is drawn as a circle
plt.show()
```

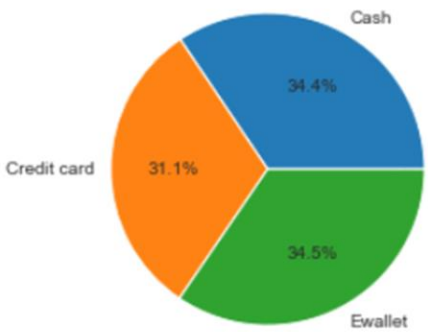


• *A is for Mandalay, B is for Naypyitaw, and C is for Yangon.*

• *the previous pie chart shows the three cities that are nearby and that they're equal in the total income.*

- More into transactions, taking a good look at each transaction per payment:

```
transaction_per_payment_type = data.groupby("Payment")["Invoice ID"].count()
plt.pie(
    transaction_per_payment_type,
    labels=transaction_per_payment_type.index,
    autopct="%1.1f%%",
)
plt.show()
```



→ *We can notice that payments by cash and e-wallets are slightly equal, Although e-wallets is more. And credit cards come in 3rd place as per usage in payments.*

→ *In general, all of the three types can be considered equal and important to track.*

- Showing quantities sold per product line in as well as average rating per product line in a table:

```
rating_per_product_line = (
    data.groupby("Product line")["Rating"].mean().reset_index()
)
rating_per_product_line
```

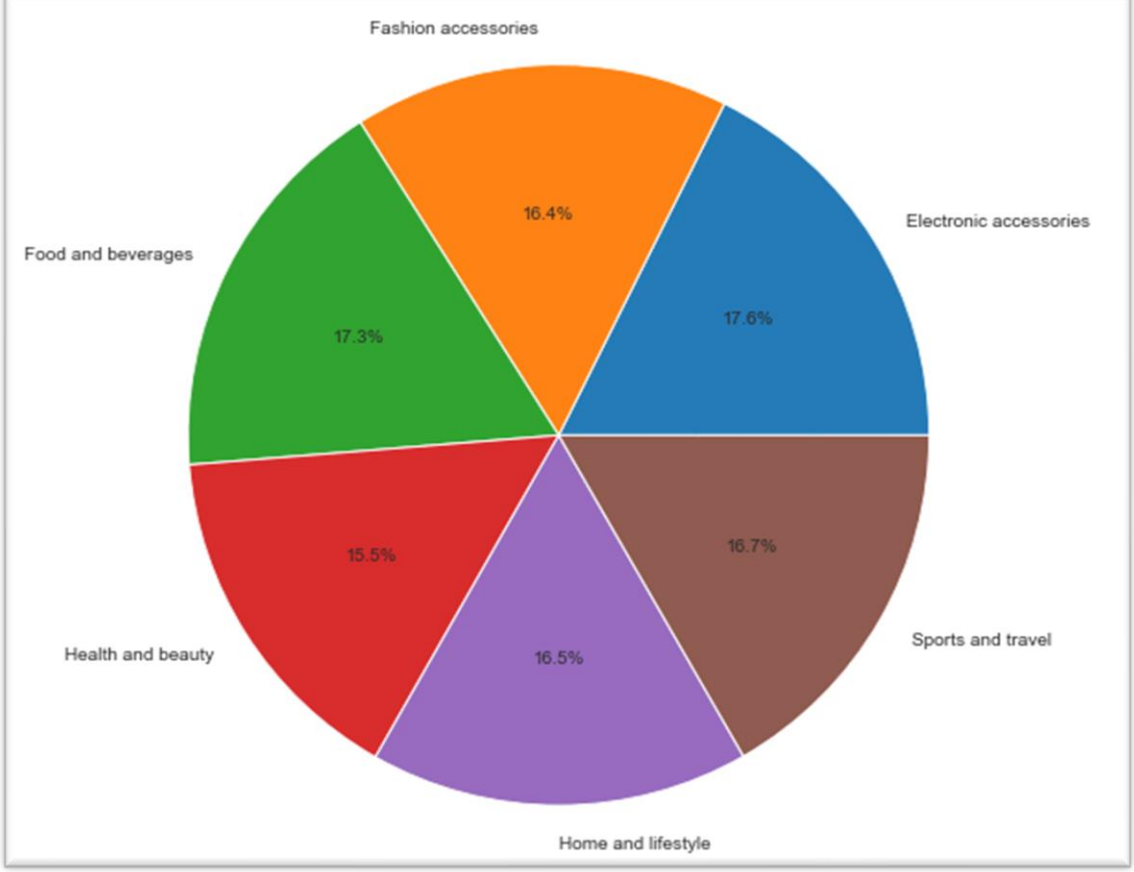
	Product line	Rating
0	Electronic accessories	6.924706
1	Fashion accessories	7.029213
2	Food and beverages	7.113218
3	Health and beauty	7.003289
4	Home and lifestyle	6.837500
5	Sports and travel	6.916265

```
quantities_sold_per_product = (
    data.groupby("Product line")["Quantity"].sum().reset_index()
)
quantities_sold_per_product
```

	Product line	Quantity
0	Electronic accessories	971
1	Fashion accessories	902
2	Food and beverages	952
3	Health and beauty	854
4	Home and lifestyle	911
5	Sports and travel	920

- Before looking at the most common attribute in our dataset which is sales, we can use a pie chart to see quantity distribution by product line to know the most influential ones from the least one as follows:

```
product_line_quantity = (
    data.groupby("Product line")["Quantity"].sum().reset_index()
)
product_line_quantity

plt.figure(figsize=(8, 8))
plt.pie(
    product_line_quantity["Quantity"],
    labels=product_line_quantity["Product line"],
    autopct="%1.1f%%",
)
plt.axis("equal")
plt.show()
```

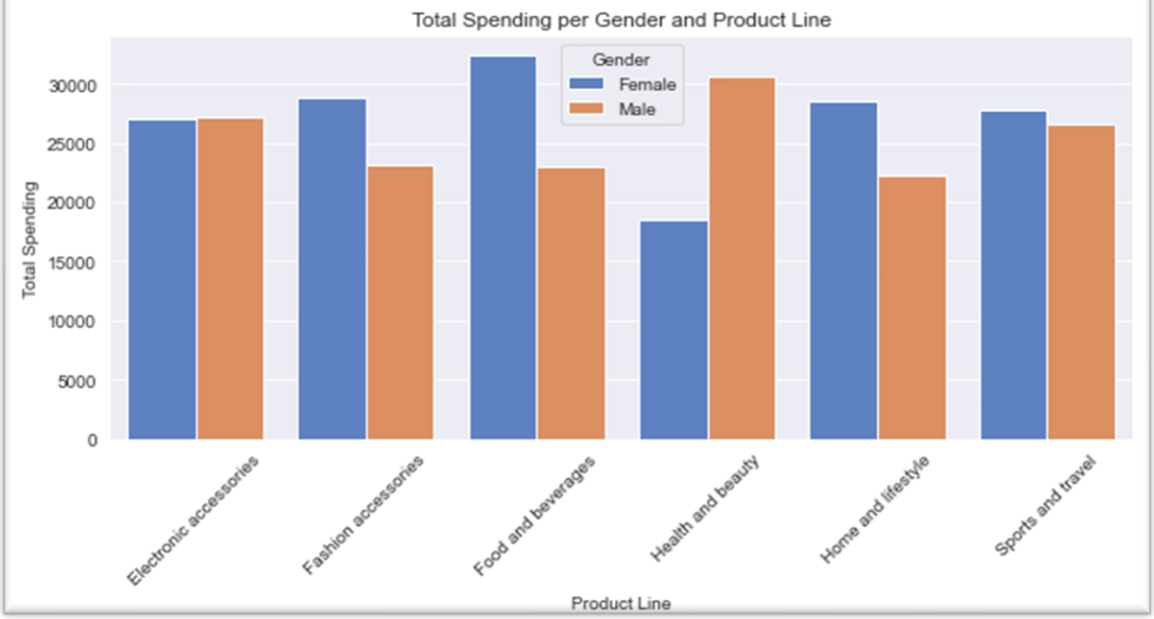


→ This graph shows the percentage of quantities purchased from each product line in general and that they're between 16.4 and 17.6.

→ We can also say that they're approximately equal and the highest line by quantity is 'Electronic accessories' and the lowest one is 'Health and Beauty'.

- looking at our customers, we can see the difference between male and female customers who spent money based on different product categories:

```
total_spending_per_gender = (
    data.groupby(["Gender", "Product line"])["Total"].sum().reset_index()
)


plt.figure(figsize=(9, 5))
sns.barplot(
    data=total_spending_per_gender,
    x="Product line",
    y="Total",
    hue="Gender",
    palette="muted",
)
plt.xlabel("Product Line")
plt.ylabel("Total Spending")
plt.title("Total Spending per Gender and Product Line")
plt.xticks(rotation=45)
plt.legend(title="Gender")
plt.tight_layout()
plt.show()
```



→ *Doing so can lead to information such as female customers are more interested "Home and life style", "Food and beverages", and "Fashion accessories" where Male ones are more interested in "Sports and travel" and "Health & beauty" but mostly equal at "Electronic accessories" which could be a good product line to focus on for more sales from both genders.*

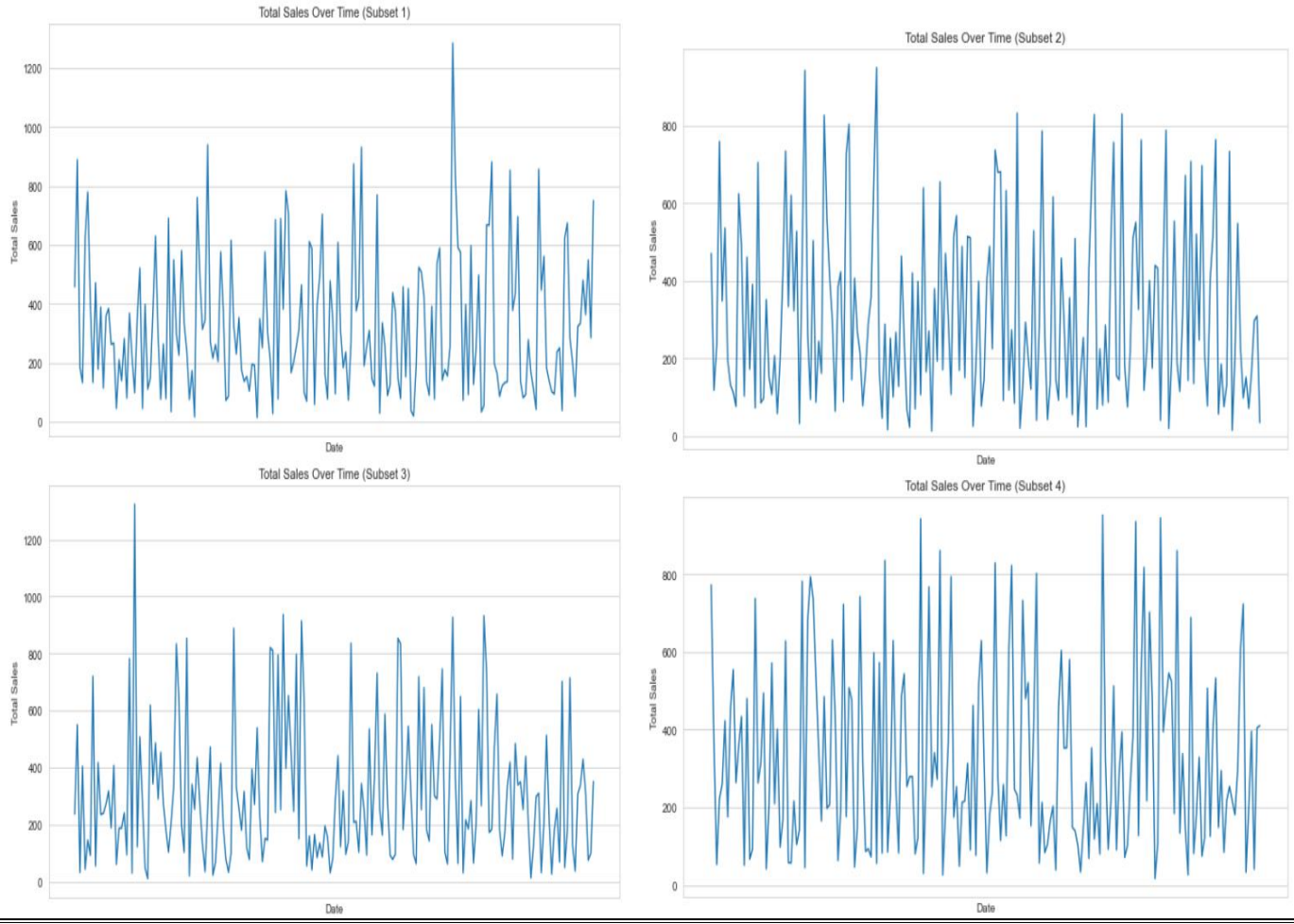
→ *looking at numerical values, both genders are spending on 'Electronic accessories' slightly equal with a value of 26500, and the same in 'Sports and Travel' with about 27000. But, in 'Fashion and Beauty' it's 23000 for males, and 29000 for females.*

- Now moving on to sales. We first want to group data by "DateTime" to calculate the total sales. Then, we can use a line chart to represent total sales for each subset from the four subsets we have in our data:

→ *In the following code, we got the total income per data and divided the data into 4 sections so it can be easier to see.*

→ *In general, the total income is about 800 and in some cases it becomes over 800.*

```python
# Group the data by "DateTime" and calculate the total sales
sales_per_day = data.groupby("DateTime")["Total"].sum().reset_index()

# Set the style of the plot
sns.set_style("whitegrid")

# Define the number of rows to plot in each iteration
plot_interval = 200

# Calculate the total number of iterations
num_iterations = len(sales_per_day) // plot_interval

# Calculate the number of rows and columns for subplots
nrows = (num_iterations + 1) // 2
ncols = 2

# Create subplots
fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(20, 10))

# Plot each subset of data
for i in range(num_iterations):
    start_index = i * plot_interval
    end_index = (i + 1) * plot_interval
    subset = sales_per_day[start_index:end_index]

    row = i // ncols
    col = i % ncols

    sns.lineplot(data=subset, x="DateTime", y="Total", ax=axes[row, col])
    axes[row, col].set_title(f"Total Sales Over Time (Subset {i+1})")
    axes[row, col].set_xlabel("Date")
    axes[row, col].set_ylabel("Total Sales")
    axes[row, col].set_xticks([])  # Remove x-axis ticks
    axes[row, col].set_xticklabels([])  # Remove x-axis labels

# Adjust layout
plt.tight_layout()
plt.show()
```



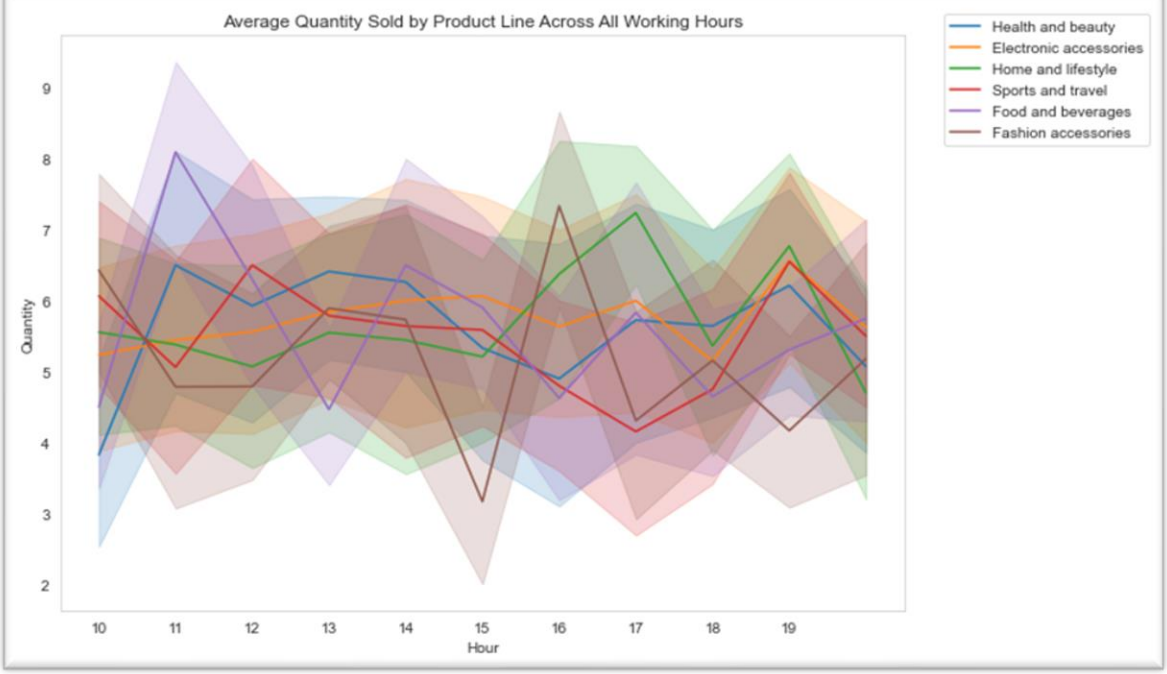
- We can now convert on date time attribute as follows:

```
# Define a function to extract hour from datetime string
def extract_hour(datetime_str):
    datetime_obj = datetime.strptime(datetime_str, "%Y-%m-%d %H:%M")
    return datetime_obj.hour


# Apply the function to the "DateTime" column to create the "Hour" column
data["Hour"] = data["DateTime"].apply(extract_hour)
data.head()
```

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Payment	cogs	gross margin percentage	gross income	Rating	DateTime	Hour
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	Ewallet	522.83	4.761905	26.1415	9.1	2019-01-05 13:08	13
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	Cash	76.40	4.761905	3.8200	9.6	2019-03-08 10:29	10
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	Credit card	324.31	4.761905	16.2155	7.4	2019-03-03 13:23	13
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	Ewallet	465.76	4.761905	23.2880	8.4	2019-01-27 20:33	20
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	Ewallet	604.17	4.761905	30.2085	5.3	2019-02-08 10:37	10

- Then, we can look at the average Quantity sold by product line across all working hours:

```
# Plot the data
plt.figure(figsize=(10, 7))
sns.lineplot(
    x="Hour",
    y="Quantity",
    data=data,
    hue="Product line",  # Corrected: changed "Product_line" to "Product line"
    estimator=np.mean,
)

plt.xticks(range(10, 20))  # Set x-axis ticks for each hour
plt.legend(bbox_to_anchor=(1.3, 1.05))
plt.title("Average Quantity Sold by Product Line Across All Working Hours")
plt.grid()
plt.show()
```

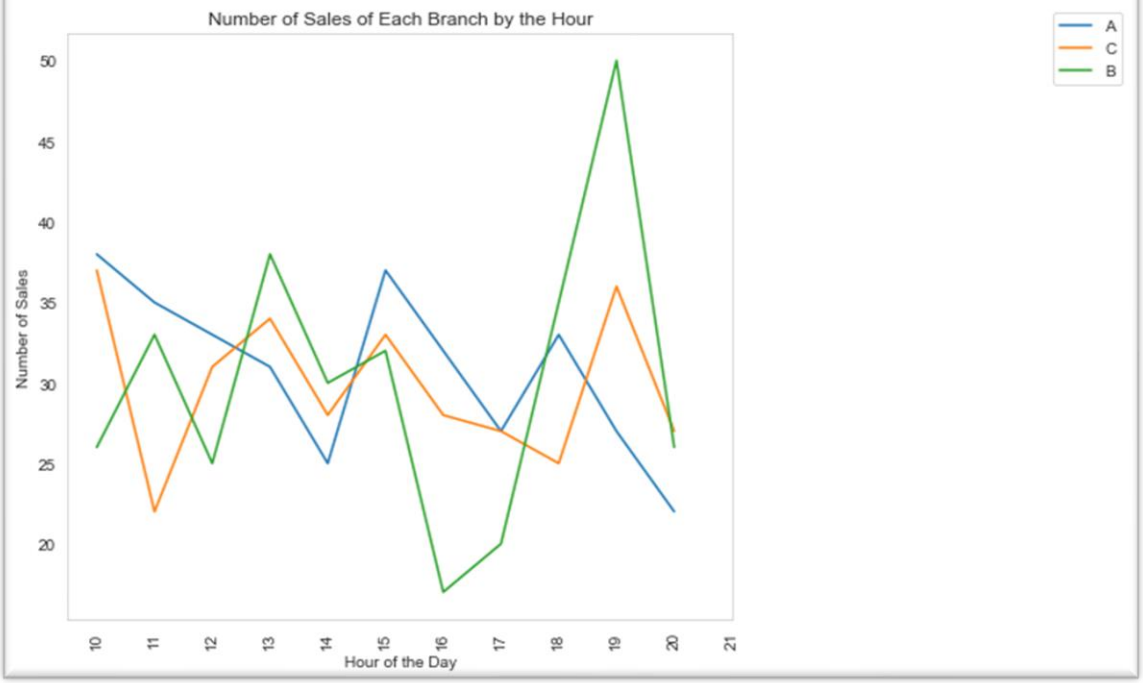


→ *The previous graph shows some important points as follows:*

- *'Food and Beverages' are purchased mostly in 11 and 15 hours.*
- *'Health and Beauty' is considered consistent except between 14:30 and 17.*
- *'Electronic accessories' increases slightly between 18 and 19, but it gets back to normal.*
- *'Home and Lifestyle' increases in the range of 15 to 17 & 18 to 19.*
- *'Sports and Travel' becomes less in 16 & 18, but it increases back in 18 and 19.*
- *'Fashion accessories' changes a lot between 14 to 17.*

- Another good use of the line chart would be for showing the number of sales of each branch per hour:

```
sales = data.groupby(["Branch", "Hour"])["Invoice ID"].count()

# Get unique branch names
unique_branches = data["Branch"].unique()

# Set the figure size
plt.figure(figsize=(12, 6))

# Plot sales data for each branch
for branch in unique_branches:
    sales[branch].plot(label=branch)

plt.xticks(range(10, 22), rotation="vertical")  # Adjust the range for all hours
plt.legend(bbox_to_anchor=(1.6, 1.05))
plt.grid()
plt.title("Number of Sales of Each Branch by the Hour")
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Sales")
plt.tight_layout()
plt.show()
```



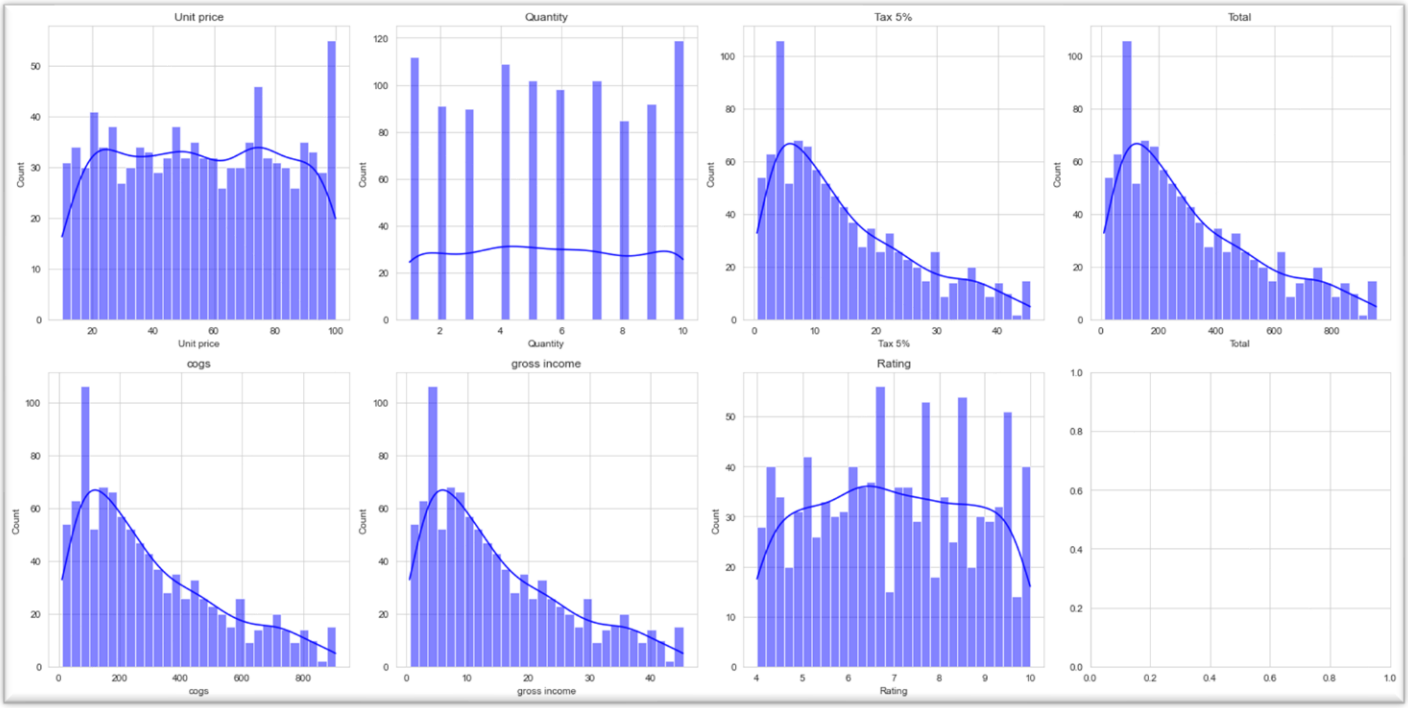
→ *In A (Yangon), it increases between 14 to 15 to get to its peak hours.*

→ *in B (Mandalay), it increases between 17 to 18 to get to its peak hours.*

→ *in C (Naypyitaw), it increases between 18 to 19:30 to get to its peak hours.*

- And the following code shows the distribution for each numerical column:

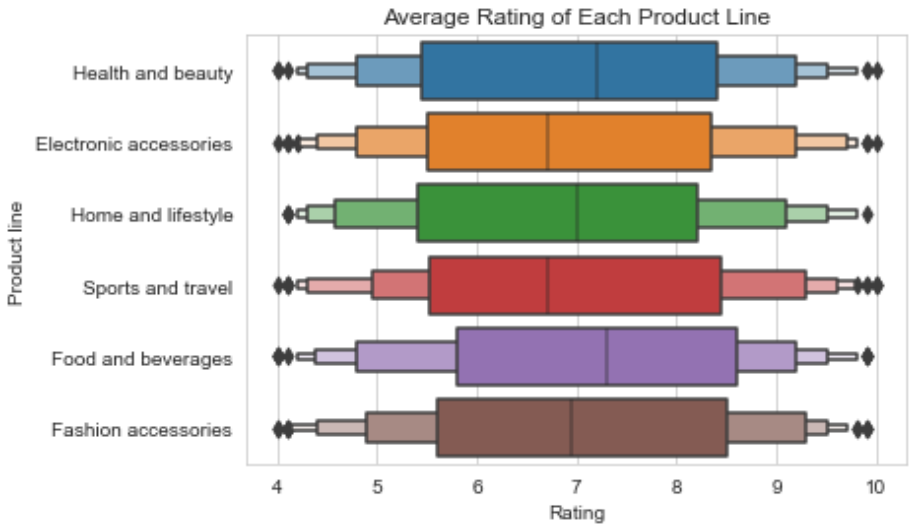
```
# Select only numerical columns except 'gross margin percentage'
numerical_cols = data.select_dtypes(include='number').columns
numerical_cols = [col for col in numerical_cols if col != 'gross margin percentage']

# Calculate the number of rows needed based on the number of columns
num_cols = 4
num_rows = 2

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 5*num_rows))

# Flatten the axes array
axes = axes.flatten()

# Plot each numerical column
for i, col in enumerate(numerical_cols):
    if col=='Cluster' or col=='K-Medoids' or col=='Hour' or col=='Month':
        continue
    sns.histplot(data=data, x=col, kde=True, bins=30, color="blue", ax=axes[i])
    axes[i].set_title(col)  # Set the title for each subplot

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```



→ *In the 'Cogs', 'Tax 5%', 'Total', and 'Gross income', since they're strongly and positively correlated, they have the same distribution which tend to be normal or chi-squared.*

- Showing Average Rating of each product line using boxplots as follows:

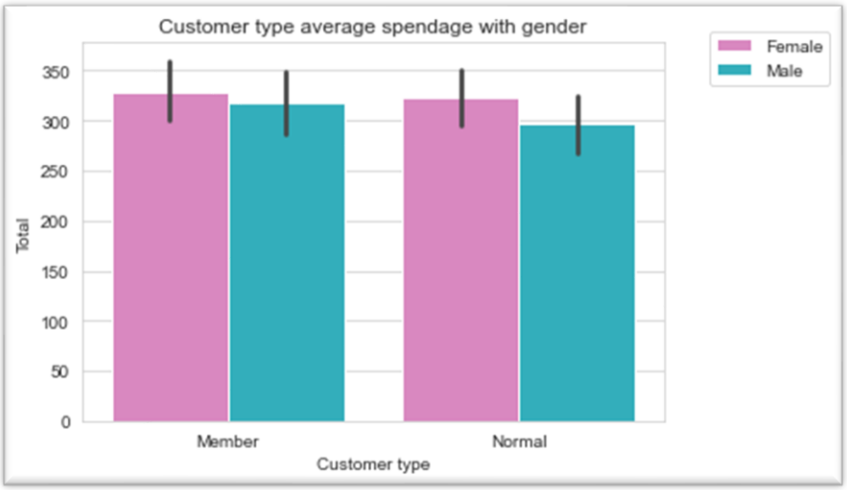
```
sns.boxenplot(y="Product line", x="Rating", data=data)
plt.title("Average Rating of Each Product Line")
plt.show()
```



→ *in the 'Food and Beverages', the mean rating is the highest and in 'Sports and Travel' is the lowest among other mean rating for the product lines. But in general, they seem to be near in the rating ranges.*

- Customer type (Member/Normal) average spending with gender:

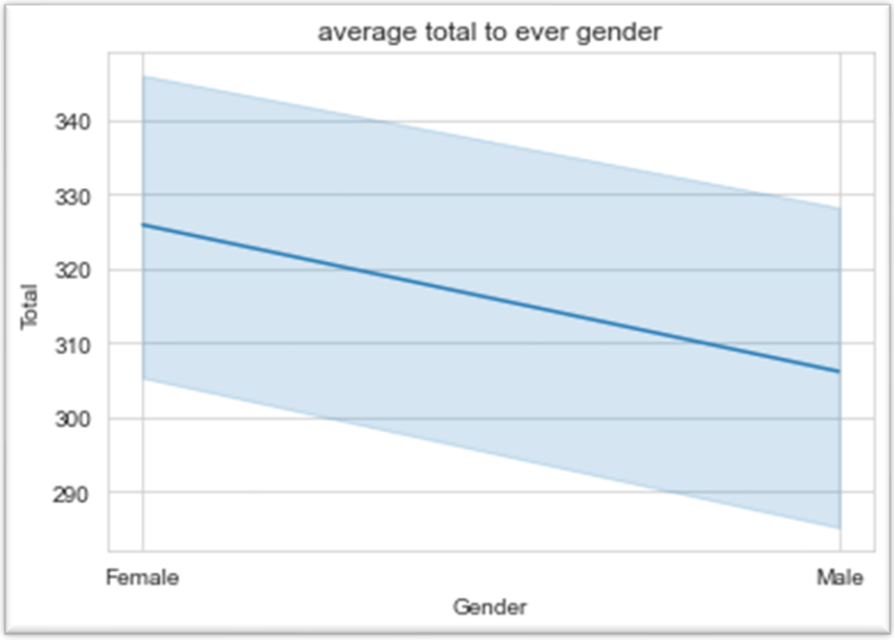
```
sns.barplot(
    x="Customer type",
    y="Total",
    data=data,
    estimator=np.average,
    hue="Gender",
    palette=["tab:pink", "tab:cyan"],
)
plt.legend(bbox_to_anchor=(1.3, 1.05))
plt.title("Customer type average spendage with gender")
plt.show()
```



→ *As the bar plot shows, it's clear that the female in members and normal customers have the highest average spending with something close to 325, and males in members are close to 310, and in normal customers are under 300.*

- And total sales using a .lineplot as follows:

```
genderCount = sns.lineplot(x="Gender", y="Total", data=data).set_title(
    "average total to ever gender"
)
```



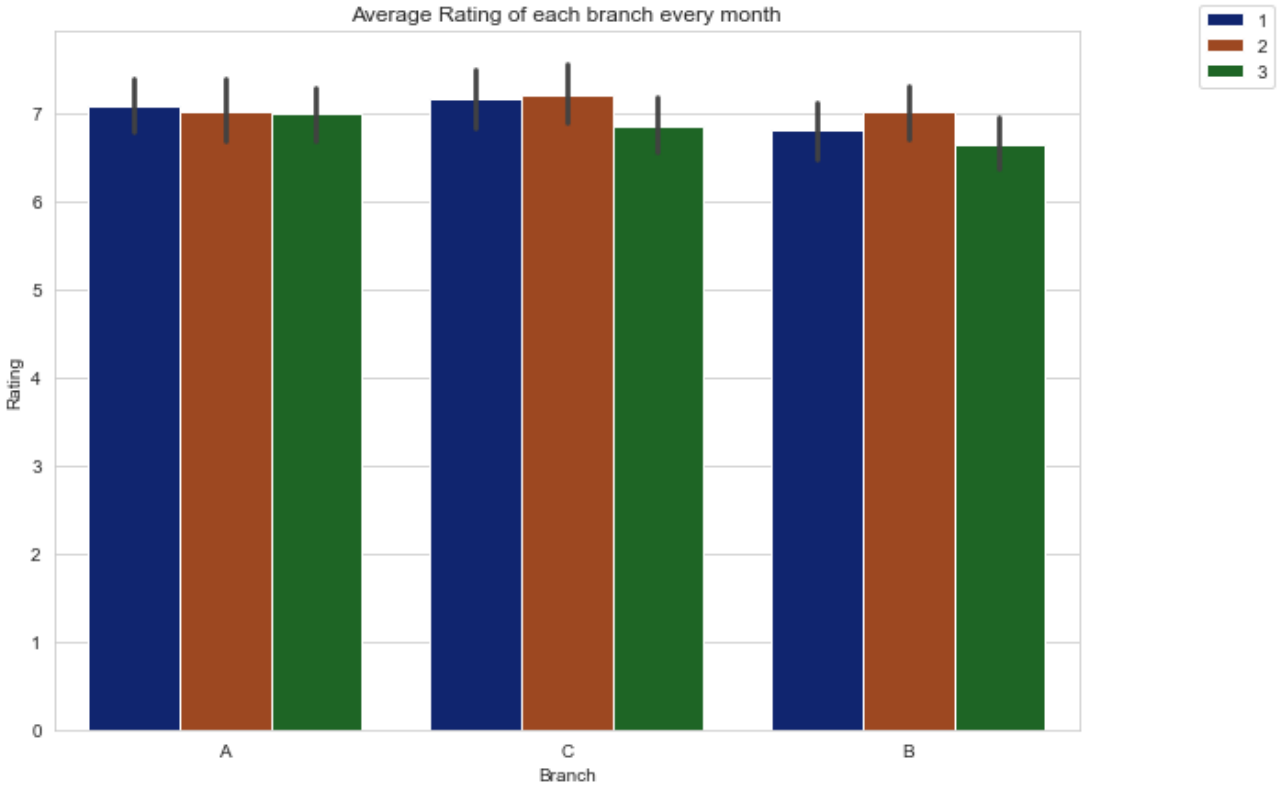
- Distribution of Customer type level codes:

```
Quantity_CT = pd.crosstab(data["Quantity"], data["Customer type"])
plt.figure(figsize=(8, 6))
Quantity_CT.plot(kind="bar", color=["lightgreen","skyblue"])
plt.title("Distribution of Customer type Level Codes")
plt.xlabel("Quantity Code")
plt.ylabel("Count")
plt.show()
```

→ *we can see that the member type buys 10, 4, or 1 more than other quantities in frequency.*

- Average rating of each branch every month:

```
palette_color = sns.color_palette("dark")
plt.figure(figsize=(10, 7))
sns.barplot(
    x="Branch",
    y="Rating",
    data=data,
    estimator=np.mean,
    hue="Month",
    palette=palette_color,
)

plt.title("Average Rating of each branch every month")
plt.legend(bbox_to_anchor=(1.2, 1.05))
plt.show()
```



- We can calculate the count of data points per hour as follows:

```
sum_hour = data.groupby("Hour").count()
sum_hour
```

Hour	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Payment	cogs	gross margin percentage	gross income	Rating	DateTime	Month
10	101	101	101	101	101	101	101	101	101	101	101	101	101	101	101	101	101
11	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
12	89	89	89	89	89	89	89	89	89	89	89	89	89	89	89	89	89
13	103	103	103	103	103	103	103	103	103	103	103	103	103	103	103	103	103
14	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83
15	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102
16	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77
17	74	74	74	74	74	74	74	74	74	74	74	74	74	74	74	74	74
18	93	93	93	93	93	93	93	93	93	93	93	93	93	93	93	93	93
19	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113
20	75	75	75	75	75	75	75	75	75	75	75	75	75	75	75	75	75

- as well as the average rating per hour:

```
avg_rating = data.groupby("Hour")["Rating"].mean()
avg_rating
```

```
Hour
10    7.098020
11    6.806667
12    7.300000
13    7.030097
14    6.934940
15    6.876471
16    6.859740
17    6.939189
18    7.187097
19    6.716814
20    6.977333
Name: Rating, dtype: float64
```

- doing so can allow us to see the count of sales per hour in a line chart:

```
plt.plot(sum_hour.index, sum_hour.values, color="blue")
plt.title("Count of Sales by Hour")
plt.xlabel("Hour")
plt.ylabel("Count")
```

```
Text(0, 0.5, 'Count')
```

→ *From 15 to 18, it's the least sales hours.*
→ *from 18 to 19, it's the highest sales hours.*

(3) K-Medoids Clustering

- K-Medoids clustering is our next step in this project and it's a partitioning method used to group similar data points into distinct clusters based on their attributes or features. It's similar to K-means but instead of using centroids to represent cluster centers, K-Medoids utilizes actual data points, known as *Medoids* as cluster representatives. The algorithm aims to minimize the dissimilarity (distance) between data points within the same cluster while maximizing the dissimilarity between clusters.

- K-Medoids process could be summarized into these 4 steps:

• *Initialization*: We first have to select *K* medoids randomly or based on some heuristic.
• *Assignment*: as we assign each data point to the nearest medoid, forming initial clusters.
• *Update Medoids*: which happens for each cluster. We select the data point that minimizes the total dissimilarity using the sum of distances to other points within the cluster as the new medoid.
• *Repeat*: as we iterate between the last two steps until we converge to the highest accuracy that we can reach, where the assignments no longer change significantly.

- Applying K-Medoids into our project is a significantly important step, and here's why:

(1) *Robustness to Outliers*: K-Medoids is more robust to outliers that K-Means, which makes it suitable for datasets with noisy or sparse data. In our case, we're dealing with supermarket sales/transactions data where outliers may represent unique customer behaviors.

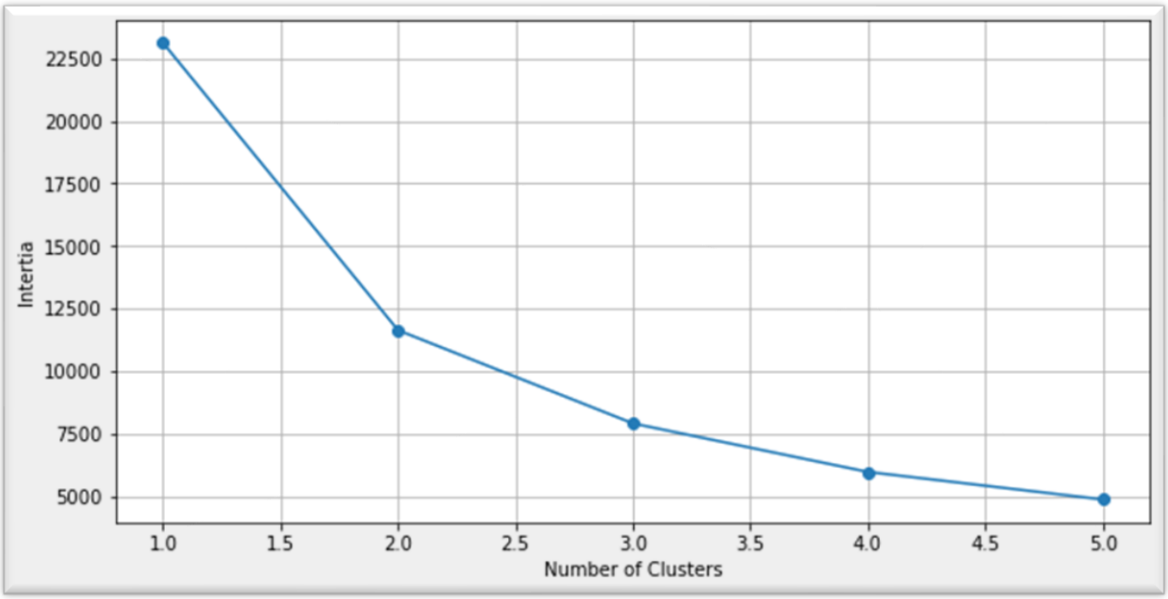
(2) *Interpretability*: As we use medoids, we enhance the interpretability of the clusters. Each cluster is represented by a real data point, making it easier to understand and interpret the characteristics of each cluster, especially in our case of customer segmentation or product categorization in a supermarket dataset.

(3) *Flexibility in Distance Measures*: it allows the use of various distance measures, like Euclidean distance, Manhattan distance, and cosine similarity, to quantify dissimilarity between data points. In our case, this flexibility enables customization of the clustering process to suit the specific characteristics and requirements of the dataset.

(4) *Scalability*: K-Medoids is computationally efficient and scalable, making it suitable for analyzing large datasets commonly encountered in data mining projects. It can handle datasets with thousands or even millions of data points efficiently, making it easier to scale our analysis in the context of a supermarket dataset with numerous transactions and customers like our case here.

- As a beginning in our code, we could make a function to decide how many clusters to use in such algorithm instead of randomly selecting it.

```
def optimise_kmedoids(data, max_k):
    means=[]
    intertias=[]

    for k in range(1, max_k):
        kmedoids=KMedoids(n_clusters=k)
        kmedoids.fit(data)

        means.append(k)
        intertias.append(kmedoids.inertia_)
    fig=plt.subplots(figsize=(10,5))
    plt.plot(means,intertias,'o-')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Intertia')
    plt.grid(True)
    plt.show()
```



- We landed on 3 clusters to use, we then go ahead and perform K-Medoids clustering on the ‘Unit Price’ and ‘Rating’

```
kmedoids=KMedoids(n_clusters=3)
```

```
kmedoids.fit(data[['Unit price','Rating']])
```

```
KMedoids(n_clusters=3)
```

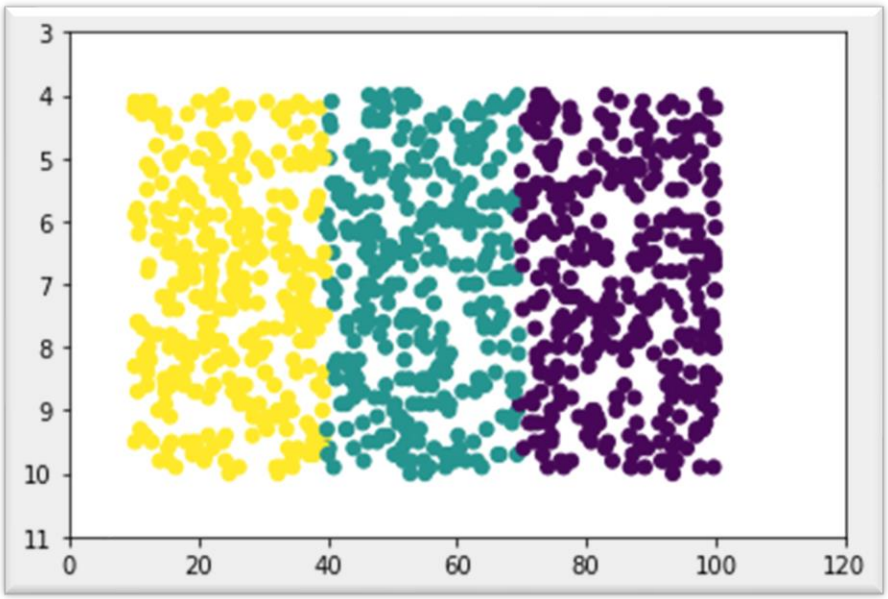
- We could then start adding clusters column into the dataset as follows:

```
data['K-Medoids']=kmedoids.predict(data[['Unit price','Rating']])
data
```

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Payment	cogs	gross margin percentage	gross income	Rating	DateTime	K-Medoids
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	Ewallet	522.83	4.761905	26.1415	9.1	2019-01-05 13:08	0
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	Cash	76.40	4.761905	3.8200	9.6	2019-03-08 10:29	2
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	Credit card	324.31	4.761905	16.2155	7.4	2019-03-03 13:23	1
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	Ewallet	465.76	4.761905	23.2880	8.4	2019-01-27 20:33	1
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	Ewallet	604.17	4.761905	30.2085	5.3	2019-02-08 10:37	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.0175	42.3675	Ewallet	40.35	4.761905	2.0175	6.2	2019-01-29 13:46	1
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.6900	1022.4900	Ewallet	973.80	4.761905	48.6900	4.4	2019-03-02 17:16	0
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	31.84	1	1.5920	33.4320	Cash	31.84	4.761905	1.5920	7.7	2019-02-09 13:22	2
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1	3.2910	69.1110	Cash	65.82	4.761905	3.2910	4.1	2019-02-22 15:33	1
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7	30.9190	649.2990	Cash	618.38	4.761905	30.9190	6.6	2019-02-18 13:28	0

- After that, we can visualize the results in a scatter plot showing each cluster with a different color as follows:

```
plt.scatter(x=data['Unit price'],y=data['Rating'],c=data['K-Medoids'])
plt.xlim(0,120)
plt.ylim(11,3)
plt.show()
```



→ *The "0" cluster, it's sales are between 10-40*
→ *the "1" cluster, it's sales are between 40-70*
→ *the "2" cluster, it's sales are between 70-100*

(4) Hierarchal Clustering

- Hierarchical Clustering is a method used to group similar data points into clusters by creating a hierarchy of clusters. Unlike partitioning methods such as K-Means or K-Medoids that we've just applied, it does not require an initial guess/value of clusters to be predefined. Instead, it creates a tree-like structure which we call **Dendrogram** where clusters are merged or split recursively based on their similarity.

- Here's why we should apply Hierarchical clustering in our project:

• *Hierarchy of Clusters*: it provides a hierarchical decomposition of the dataset, allowing for more accurate understanding of the data structure. And this representation is useful in exploring nested relationships and identifying clusters at different granularity levels, which can be valuable in analyzing complex datasets like the one we're working on.

• *No Need for predefined number of clusters*: As said before, hierarchical clustering doesn't require the specification of the number of clusters beforehand, and such flexibility is really helpful when the optimal cluster count is unknown or when exploring different segmentation levels in a dataset.

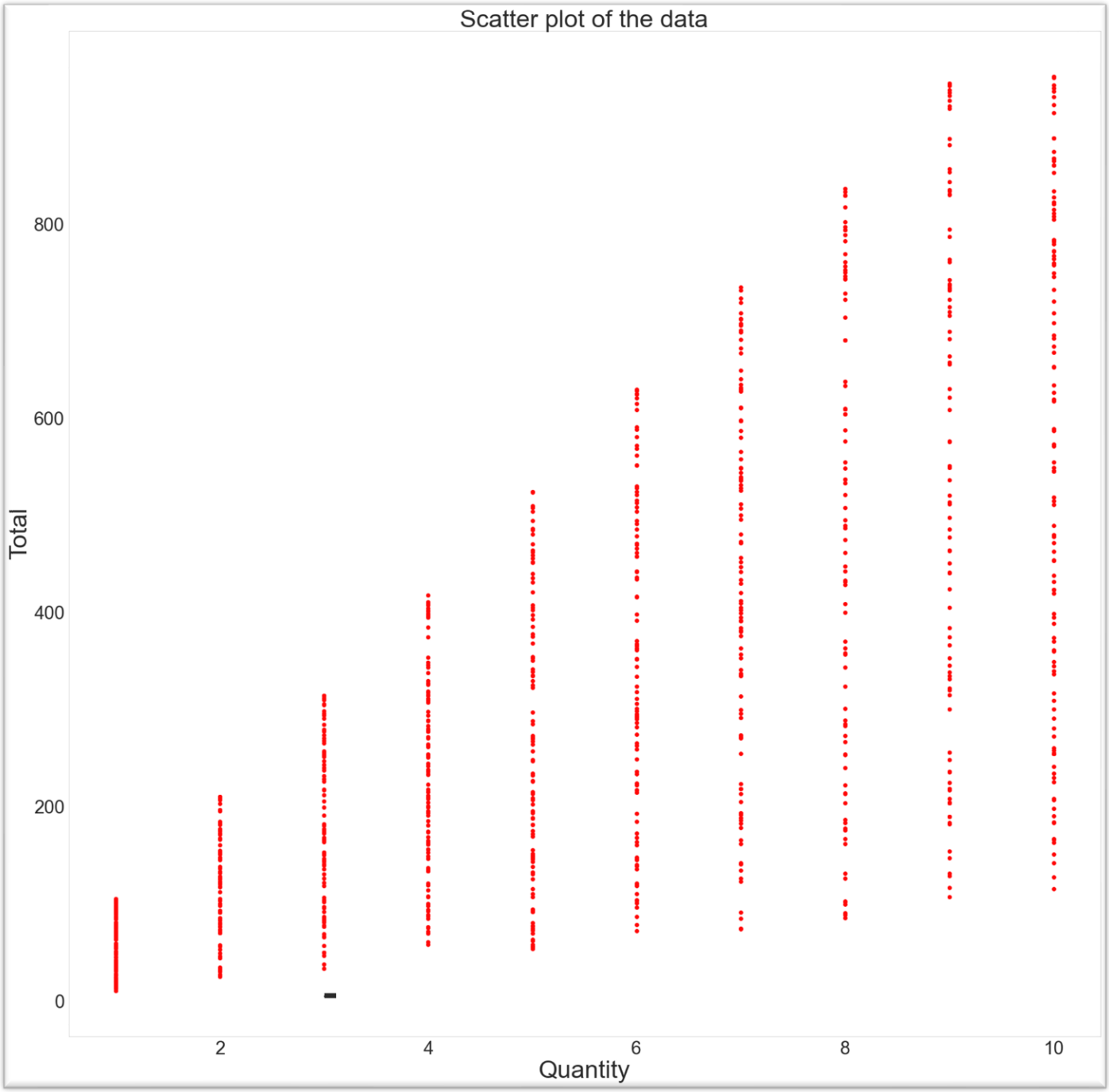
• *Interpretability*: The dendrogram that is produced as an output and is visually represented, shows the clustering structure that we have, making it easy to interpret and understand the relationships between clusters. We then can visually inspect the dendrogram to identify meaningful clusters and determine their characteristics based on the merging/splitting process.

• *Robustness to Initialization*: as it's less sensitive to initialization than some partitioning methods, such as K-Means. The hierarchical nature of the algorithm ensures that clusters are merged or split based on the overall similarity structure of the data, rather than initial cluster assignments.

- So, by applying Hierarchical clustering in our project here, we can uncover hierarchical relationships, explore clustering structures at different granularity levels, and derive actionable insights to optimize supermarket operations and enhance customer experiences.

- First, we can create a scatter plot of ‘Quantity’ vs ‘Total’ as the scatter plot shows when the quantity increases as sales increases as well:

```
plt.figure(figsize=(30, 30))
plt.scatter(data['Quantity'], data['Total'], c="r")
for i in range(data.shape[0]):
    plt.annotate(str(i), (data.loc[i, 'Quantity'], data.loc[i, 'Total']), xytext=(3,3), fontsize=12)
plt.xlabel("Quantity", fontsize=40)
plt.ylabel("Total", fontsize=40)
plt.title("Scatter plot of the data", fontsize=40)
plt.xticks(fontsize=30)
plt.yticks(fontsize=30)
plt.grid()
plt.show()
```

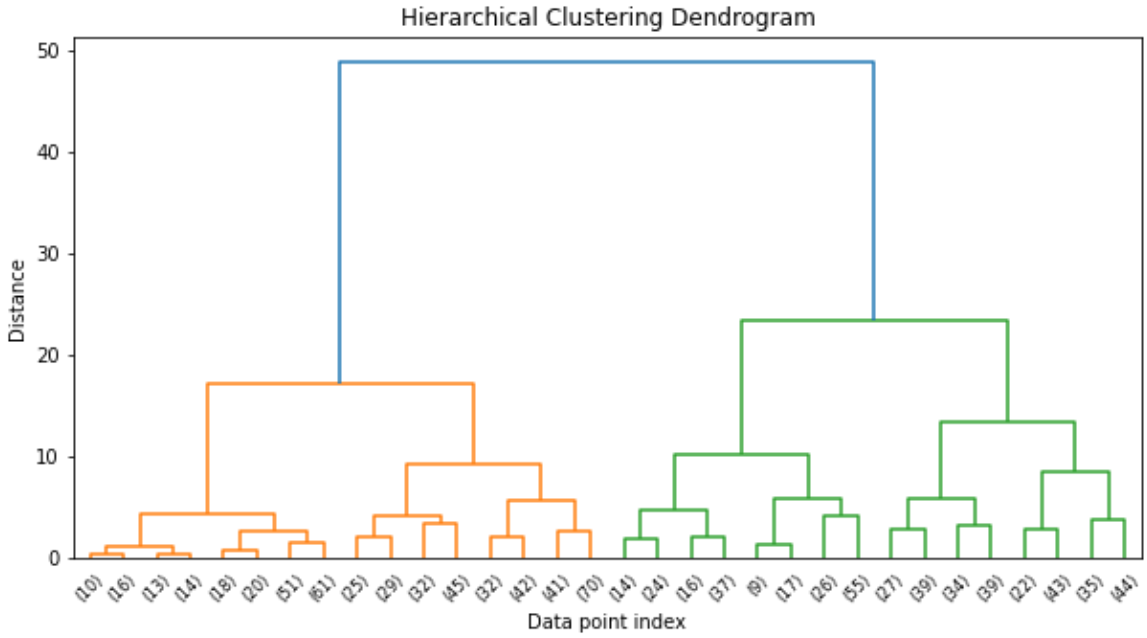


→ *As the scatter plot shows when the quantity increases and sales increases as well.*

- Now, we can perform hierarchical clustering between ‘Total’ and ‘Quantity’ as well as the dendrogram, and we can add truncation at a specific level (truncate_mode = ‘leve’) and a maximum of four level (p=4)

```
# Select relevant columns for clustering
X = data[['Quantity', 'Total']]  # DataFrame slicing

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform hierarchical clustering
clustering = AgglomerativeClustering(n_clusters=4)
clusters = clustering.fit_predict(X_scaled)

# Add cluster labels to DataFrame
data['Cluster'] = clusters

print(data.head())

# Perform hierarchical clustering for dendrogram
Z = linkage(X_scaled, "ward")

# Plot dendrogram
plt.figure(figsize=(10, 5))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data point index')
plt.ylabel('Distance')
dendrogram(Z, truncate_mode='level', p=4)
plt.show()
```

```
    Invoice ID Branch       City Customer type  Gender  \
0  750-67-8428      A     Yangon        Member  Female
1  226-31-3081      C  Naypyitaw        Normal  Female
2  631-41-3108      A     Yangon        Normal    Male
3  123-19-1176      A     Yangon        Member    Male
4  373-73-7910      A     Yangon        Normal    Male

             Product line  Unit price  Quantity   Tax 5%     Total  \
0       Health and beauty       74.69         7  26.1415  548.9715
1  Electronic accessories       15.28         5   3.8200   80.2200
2      Home and lifestyle       46.33         7  16.2155  340.5255
3       Health and beauty       58.22         8  23.2880  489.0480
4       Sports and travel       86.31         7  30.2085  634.3785

       Payment    cogs  gross margin percentage  gross income  Rating  \
0      Ewallet  522.83                 4.761905       26.1415     9.1
1         Cash   76.40                 4.761905        3.8200     9.6
2  Credit card  324.31                 4.761905       16.2155     7.4
3      Ewallet  465.76                 4.761905       23.2880     8.4
4      Ewallet  604.17                 4.761905       30.2085     5.3

          DateTime  Cluster
0  2019-01-05 13:08        2
1  2019-03-08 10:29        1
2  2019-03-03 13:23        0
3  2019-01-27 20:33        0
4  2019-02-08 10:37        2
```

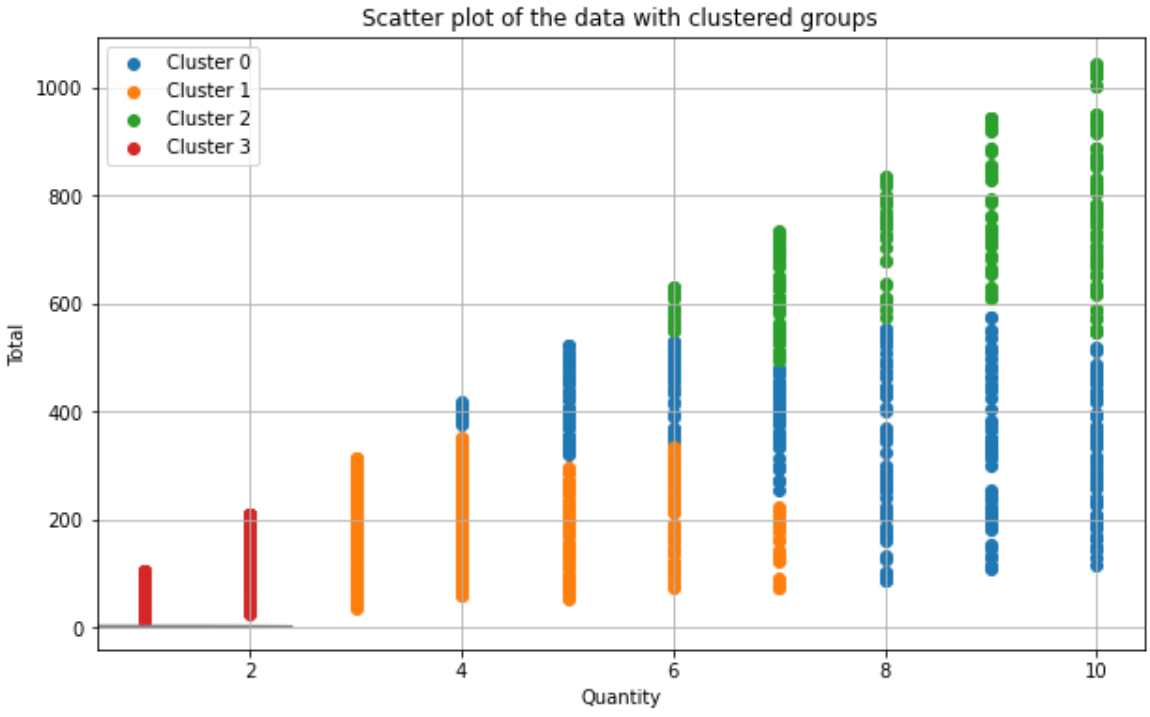


→ *we have used minimum or single metric to get the clusters with the smallest distance in one cluster.*

→ *the K-Medoids method is better than hierarchical method to use it on the data.*

- we found that the best number of clusters is 4, we can then create a scatterplot with clustered groups on the 'Quantity' and 'Total' columns as follows:

```
# Plot scatter plot
plt.figure(figsize=(10, 6))
for cluster in np.unique(clusters):
    plt.scatter(data.loc[clusters == cluster, 'Quantity'], data.loc[clusters == cluster, 'Total'], label=f'Cluster {cluster}')

# Draw circles around clusters
for cluster in np.unique(clusters):
    cluster_center = np.mean(X_scaled[clusters == cluster], axis=0)
    radius = max(np.linalg.norm(X_scaled[clusters == cluster] - cluster_center, axis=1))
    circle = plt.Circle(cluster_center, radius, color='gray', fill=False)
    plt.gca().add_artist(circle)

plt.xlabel("Quantity")
plt.ylabel("Total")
plt.title("Scatter plot of the data with clustered groups")
plt.legend()
plt.grid()
plt.show()
```



Summary

Throughout this data mining project, we focused on making a comprehensive analysis of a Supermarket dataset, employing various techniques to extract valuable insights and enhance decision-making.

The project encompassed key phases including data *preprocessing*, *exploratory analysis (EDA*), and the application of clustering algorithms such as *K-Medoids* and *Hierarchical Clustering*. Each phase contributed to a deeper understanding of the dataset's characteristics and made the identification of meaningful patterns and relationships easier.

Data preprocessing involved cleaning, transforming, and preparing the dataset for analysis, ensuring its quality and suitability for further exploration. Through *Exploratory Analysis (EDA)*, we gained insights into the distribution, relationships, and outliers within the dataset, laying the foundation for subsequent analysis.

The application of *K-Medoids Clustering* enabled us to segment customers based on heir purchasing behaviors, facilitating targeted marketing strategies and personalized services. Meanwhile, *Hierarchical Clustering* provided a hierarchical decomposition of the dataset, offering insights into the underlying structure and relationships among customers and products.

Conclusion

This data mining project on the Supermarket dataset has demonstrated the efficacy of leveraging advanced analytical techniques to extract actionable insights and drive business value. By preprocessing the data and conducting exploratory analysis, we gained a comprehensive understanding of the dataset's characteristics, paving the way for more focused analysis.

The application of clustering algorithms, specifically K-Medoids and Hierarchical Clustering, allowed us to uncover meaningful patterns segment customers effectively. These insights can inform targeted marketing campaigns, optimize inventory management, and enhance overall supermarket operations.

Moving forward, the insights derived from this project can serve as a foundation for further analysis and strategic decision-making within the supermarket industry. By leveraging data mining techniques, organizations continue to innovate, adapt to changing consumer preferences, and drive sustainable growth in an increasingly competitive market landscape.