1️⃣ Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
plt.style.use("seaborn-v0_8")
```

Purpose:
You load essential Python libraries for data analysis and visualization.

- **pandas** → for data reading, cleaning, and manipulation (DataFrames).
- **numpy** → for numerical operations (used for arrays and NaN handling).
- **`matplotlib.pyplot`** → for creating charts and plots.
- **`seaborn`** → for stylish and statistical visualizations built on top of matplotlib.
- **`plotly.express`** → for interactive visualizations (optional but useful).

`plt.style.use("seaborn-v0_8")` sets the global theme for all plots, making them look cleaner and more modern.

✅ *No output produced yet; this just prepares the environment.*

2️⃣ Load the Dataset

```
df = pd.read_csv("/kaggle/input/supermarket-sales-dataset/SuperMarket Analysis.csv")
```

Purpose:
Loads your CSV file from Kaggle's input folder into a pandas **DataFrame** (a 2D table).

Result:

- Variable `df` now holds the entire dataset.
- Each column becomes a `Series` (like "City", "Gender", "Total", etc.)

- Each row represents a sales record.

✅ *No print output yet, but you've now imported your dataset.*

🟫 3️⃣ Clean and Standardize Column Names

```
df.columns = df.columns.str.strip().str.title().str.replace(" ", "_")
```

Purpose:
This line cleans your column names to make them easier to use in code.

What it does in sequence:

1. `.str.strip()` → removes spaces before or after column names.
2. `.str.title()` → capitalizes the first letter of each word (e.g. "customer type" → "Customer Type").
3. `.str.replace(" ", "_")` → replaces spaces with underscores (e.g. "Customer Type" → "Customer_Type").

Why?
Because in Python, column names like "Customer Type" can cause syntax problems. This makes them clean and consistent.

✅ *No visual output — but column names are now standardized.*

🟫 4️⃣ Display Dataset Columns

```
print("🧾 Columns in dataset:")
print(df.columns.tolist())
```

Purpose:
Displays all current column names after cleaning.
Useful to confirm that columns like `"Gender"` or `"Customer_Type"` exist.

Result Example:

```
🧾 Columns in dataset:
['Invoice_Id', 'Branch', 'City', 'Customer_Type', 'Gender',
'Product_Line', 'Unit_Price', 'Quantity', 'Tax_5%', 'Total', 'Date',
'Time', 'Payment', 'Cogs', 'Gross_Income', 'Rating']
```

✅ *This helps avoid "column not found" errors later.*

🟦 5️⃣ Preview the Data

```
display(df.head(10))
```

Purpose:
Shows the first 10 rows of your dataset as a table (like Excel preview).

Result:
You can see sample transactions — each row showing:

- Customer gender
- City
- Product line
- Unit price, Quantity, Total
- Payment method, Rating, etc.

✅ *You visually confirm the dataset is read correctly.*

🟦 6️⃣ Show Dataset Info

```
df.info()
```

Purpose:
Summarizes the structure of your dataset.

Result Example:

```
<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
 ...
```

Explanation:

- `Non-Null Count` → shows missing data (if any).
- `Dtype` → shows data types (object = text, float64 = numbers).
- Confirms data is clean (no NaN values).

✅ *Gives technical overview of your dataset.*

🟦 7️⃣ Basic Statistics

```
print("\n 🔹 Dataset shape:", df.shape)
display(df.describe())
```

Purpose:

- `.shape` → shows the number of rows and columns (e.g., `(1000, 17)`).
- `.describe()` → gives statistics like mean, min, max, standard deviation for numeric columns.

Result Example:

	Unit_Price	Quantity	Tax_5%	Total	Rating

count	1000	1000	1000	1000	1000
mean	55.67	5.51	15.38	322.97	6.97
min	10.08	1	0.53	10.97	4.0
max	99.96	10	49.65	1042.65	10.0

✅ *This helps understand overall sales and prices.*

🟦 8️⃣ Unique Values in Categorical Columns

```
for col in df.select_dtypes(include='object').columns:
    print(f"{col}: {df[col].nunique()}")
```

Purpose:
Shows how many unique categories exist per column (e.g., how many cities, product types, etc.).

Result Example:

```
City: 3
Customer_Type: 2
Gender: 2
Product_Line: 6
Payment: 3
```

✅ *Helps identify what dimensions you can analyze (like comparing 3 cities).*

🟦 9️⃣ Safe Countplot Function

```
def safe_countplot(column, title):
    if column in df.columns:
        plt.figure(figsize=(5,4))
        sns.countplot(data=df, x=column)
        plt.title(title)
        plt.show()
```

```
    else:
        print(f"⚠️ Column '{column}' not found in dataset.")
```

Purpose:
Creates a *safe plotting function* that won't crash if a column name is wrong.

Why useful?
You can reuse this to quickly visualize how data is distributed (e.g., how many males vs females).

✅ *No result yet — this just defines the function.*

📊 🔹 Plot Examples

```
safe_countplot("Gender", "Gender Distribution")
safe_countplot("Customer_Type", "Customer Type Counts")
safe_countplot("Branch", "Number of Sales per Branch")
safe_countplot("City", "Number of Sales per City")
```

Purpose:
Each of these calls draws a bar chart (countplot) for the given column.

Result Example:

- *Gender Distribution:* Shows ratio of Male vs Female customers.
- *Customer Type:* Compares Members vs Normal customers.
- *Branch:* Displays sales count in each branch (A, B, C).
- *City:* Shows transactions by city.

✅ *Visual patterns become visible (e.g., "Branch C has most sales").*

🟦 ◇ Bivariate Analysis

```
branch_sales =
df.groupby("Branch")["Total"].sum().sort_values(ascending=False)
```

Purpose:
Groups data by branch and sums up the "Total" sales.
Sorts branches from highest to lowest total.

Result Example:

```
C     106200
B     101787
A      99685
```

✅ *Shows which branch earns the most.*

Similar logic applies to:

```
city_transactions = df["City"].value_counts()
product_revenue =
df.groupby("Product_Line")["Total"].sum().sort_values(ascending=False)
spending_by_type = df.groupby("Customer_Type")["Total"].mean()
```

They produce:

- **City Transactions:** most active cities.
- **Product Revenue:** which product lines are top earners.
- **Spending by Type:** members vs normal spending average.

✅ *All these produce printed tables + bar charts.*

🟦 ◇ Correlation Heatmap

```
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm',
fmt=".2f")
```

Purpose:
Computes and displays numeric correlations between columns.
E.g. how "Quantity" relates to "Total" or "Tax 5%".

Result Example:

	Unit_Price	Quantity	Tax_5%	Total
Unit_Price	1.00	0.02	0.88	0.88
Quantity	0.02	1.00	0.90	0.90

✅ *You can visually see which variables are strongly linked (bright red/blue cells).*

🟦 ◇ Scatterplots

```
sns.scatterplot(data=df, x="Quantity", y="Total", hue="Gender")
sns.scatterplot(data=df, x="Unit_Price", y="Total")
```

Purpose:
Visualize relationships between variables:

- Quantity vs Total: do higher quantities mean higher totals?
- Unit Price vs Total: do expensive items increase total value?

Result:
A scatterplot with each point representing a transaction.

✅ *Helps understand data trends.*

🟦 ◇ Dashboard KPIs

```
total_sales = df["Total"].sum()
avg_basket = df["Total"].mean()
num_transactions = df.shape[0]
avg_rating = df["Rating"].mean()
```

Purpose:
Calculates key business performance metrics.

Result Example:

```
📊 Quick Dashboard KPIs:
Total Sales: $322,000.00
Average Basket Size: $322.97
Number of Transactions: 1000
Average Rating: 7.00
```

✅ *Gives an instant summary for business insight.*

☑ Final Line

```
print("\n✅ Analysis Complete — Proceed to Markdown cells for
insights.")
```

Purpose:
Simply prints a completion message.
Tells you to interpret your findings in markdown cells.

Result:

```
✅ Analysis Complete — Proceed to Markdown cells for insights.
```

✅ *End of the notebook execution.*