PYTHON PROJECT COMPREHENSIVE REPORT

Project Folder Setup

To ensure proper organization and easy reproducibility, a dedicated project folder named **“Python Project”** was created.

- The **Jupyter Notebook** used for the analysis was saved inside this folder.
- The **dataset (CSV file)** was kept in the parent directory, and accessed using a **relative path**.
- This folder structure helped to keep raw data separate from the analysis files while making paths easy to manage.

Importing Libraries

The analysis was carried out using **Python**, with the help of the **pandas** library, which is widely used for handling structured datasets.

```
import pandas as pd
```

Loading the Dataset

The dataset was read into a **pandas DataFrame** using the command:

```
df = pd.read_csv("dataset.csv")
```

This made the data available for cleaning and analysis.

Initial Dataset Inspection

To understand the structure of the dataset, the following exploratory commands were applied:

```
df.shape       # Returns number of rows and columns
 df.columns     # Displays list of column names
 df.dtypes      # Shows data types of each column
 df.head()     # Displays first 5 rows
```

These checks revealed formatting issues such as **unwanted rows** and **extra unnamed columns**.

Correcting Column Headers

The first row of the dataset contained irrelevant information. To fix this, the import was adjusted to skip the first row:

```
df = pd.read_csv("dataset.csv", skiprows=1)
```

This ensured that the correct row was used as the header.

Removing Unwanted Columns

Extra unnamed columns were dropped, leaving only relevant data:

```
df = df.loc[:, ~df.columns.str.contains('^Unnamed')]
```

Rechecking Dataset Structure

The dataset was inspected again to confirm corrections:

```
df.shape
 df.columns
 df.head()
```

Viewing Sample Data

A sample of the dataset was displayed to examine its structure:

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

Checking Dataset Information

Metadata on column names, non-null counts, and data types was obtained using:

```
df.info()
```

Handling Missing Values

Missing values were checked with:

```
df.isnull().sum()
```

This showed whether any columns had null entries.

Removing Duplicate Records

Duplicate records were removed to maintain data integrity:

```
df.drop_duplicates(inplace=True)
 df.duplicated().sum()
```

Standardizing Column Names

Column names were standardized by removing unwanted spaces:

```
df.columns = df.columns.str.strip()
```

Cleaning Extra Spaces in Data

Extra spaces within cell values were also removed:

```
df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)
```

This ensured that text-based data was uniform and consistent.

Converting Date Column

The date column was initially stored as **object type**. It was converted to proper **datetime format** to allow time-based operations:

```
df['date'] = pd.to_datetime(df['date'], dayfirst=True)
```

The `dayfirst=True` argument was used since the dataset followed the **dd/mm/yyyy** format.

Verifying Conversion

The conversion was confirmed using:

```
df.dtypes
 df.head()
```

This showed that the date column had successfully changed to datetime64[ns].

Summary of the Cleaning Steps

The dataset was cleaned and prepared by performing the following steps:

- Imported the dataset into **pandas**.
- Skipped unnecessary rows and corrected headers.
- Removed unwanted **Unnamed** columns.
- Inspected structure, shape, and column details.
- Displayed sample rows and metadata.
- Checked and handled missing values.
- Removed duplicate records.
- Standardized column names and stripped extra spaces.
- Converted the date column from **object** to **datetime**.
- Verified corrections with dataset checks.

The dataset is now **clean, standardized, and ready for statistical analysis.**

Descriptive Statistics

To understand the dataset more deeply, descriptive statistics were carried out on both numerical and categorical variables, alongside exploratory summaries of sales across different dimensions such as date, city, product, and manager.

Numerical Summary

The descriptive statistics for numerical variables (*Price* and *Quantity*) provided key measures such as the mean, standard deviation, minimum, maximum, and quartiles. These values helped in identifying the spread, central tendency, and variability of sales prices and product quantities in the dataset

Categorical Summary

For categorical variables (*Product*, *Purchase Type*, *Payment Method*, *Manager*, and *City*), the summary generated information such as the number of unique values, the most common category, and the frequency of its occurrence. This provided insight into the most frequent sales practices, including preferred payment methods, commonly sold products, and leading sales managers.

Total Sales Column

A new column (*Total_Sale*) was created by multiplying *Price* and *Quantity*. This allowed for the calculation of overall revenue generated per transaction, serving as a foundation for further analysis.

Sales Trend Over Time

The dataset was grouped by transaction date to identify the top three highest revenue-generating days. This analysis revealed the peak sales periods and highlighted specific dates when sales were at their maximum.

Sales by City

Sales were aggregated by city to determine which locations contributed the most to overall revenue. The top three cities with the highest sales were identified, showing geographic areas with the strongest market performance.

Sales by Product

The analysis of total sales by product highlighted the top three products with the highest revenue contribution. This helped to identify which products were most profitable and in greatest demand.

Sales by Manager

Total sales were also grouped by manager, allowing the identification of the top three managers who generated the highest revenue. This provided insights into individual performance and leadership impact on sales.

PROJECT QUESTIONS

1. Most Preferred Payment Method

Code:

```
preferred_payment = df['Payment_Method'].value_counts().head(1)
 print("Most preferred Payment Method:", preferred_payment)
```

Result Interpretation:

The most preferred payment method was **Credit Card**, indicating customers largely favored this method for convenience and accessibility.

2. Most Selling Product (by Quantity & Revenue)

Code (By Quantity):

```
most_selling_quantity =
df.groupby('Product')['Quantity'].sum().sort_values(ascending=False).h
ead(1)
 print("Most selling product by Quantity:")
 print(most_selling_quantity)
```

Code (By Revenue):

```
most_selling_revenue =
df.groupby('Product')['Total_Sale'].sum().sort_values(ascending=False)
.head(1)
 print("Most selling product by Revenue:")
 print(most_selling_revenue)
```

Result Interpretation:

- By Quantity → **Beverages** sold the most units.
- By Revenue → **Burger** generated the highest revenue.

This shows that the best-selling product by unit is not always the highest earner in terms of revenue.

3. City & Manager with Maximum Revenue

Code (City):

```
max_city =
df.groupby('City')['Total_Sale'].sum().sort_values(ascending=False).he
ad(1)
 print("City with maximum Revenue:")
 print(max_city)
```

Code (Manager):

```
max_manager =
df.groupby('Manager')['Total_Sale'].sum().sort_values(ascending=False)
.head(1)
 print("Manager with Maximum Revenue:")
 print(max_manager)
```

Result Interpretation:

- The city with maximum revenue was **Lisbon**.
- The manager with maximum revenue was **Joao Silva**.

This highlights regional and managerial performance differences.

4. Average Revenue

Code:

```
avg_revenue = df['Total_Sale'].mean()
 print("Average Revenue:", avg_revenue)
```

Result Interpretation:

The average revenue per transaction was **₦3029.59**, giving an idea of the typical sale size.

5. Average Revenue in November & December

Code:

```
nov_dec = df[df['Date'].dt.month.isin([11, 12])]
 avg_revenue_nov_dec = nov_dec['Total_Sale'].mean()
 print("Average Revenue (Nov & Dec):", avg_revenue_nov_dec)
```

Result Interpretation:

The average revenue in November and December was **₦3029.59**, reflecting **holiday/seasonal sales performance**.

6. Standard Deviation of Revenue & Quantity

Code:

```
std_revenue = df['Total_Sale'].std()
 std_quantity = df['Quantity'].std()

print("Standard Deviation of Revenue:", std_revenue)
 print("Standard Deviation of Quantity:", std_quantity)
```

Result Interpretation:

- Standard deviation of revenue = **2420.12**
- Standard deviation of quantity = **214.89**

This shows how spread out sales and quantities are around the average.

7. Variance of Revenue & Quantity

Code:

```
print(df[['Total_Sale','Quantity']].var())
```

Result Interpretation:

- Variance of revenue = **5,856,972.96**
- Variance of quantity = **46,177.15**

A higher variance means more fluctuation in sales/quantities.

8. Revenue Trend Over Time

Code:

```
revenue_trend = df.groupby('Date')['Total_Sale'].sum()
 print(revenue_trend)

import matplotlib.pyplot as plt
 revenue_trend.plot(figsize=(8,5), marker='o', title="Revenue Trend
Over Time")
 plt.xlabel("Date")
 plt.ylabel("Revenue")
 plt.show()
```

Result Interpretation:

The revenue trend fluctuated at first, then stabilized, and increased towards the end of December, suggesting a **positive upward trend overall**.

9. Average Quantity Sold & Revenue per Product

Code:

```
avg_product_stats = df.groupby('Product').agg({
     'Quantity': 'mean',
     'Total_Sale': 'mean'
 })
 print(avg_product_stats)
```

Result Interpretation:

Each product shows different average sales and revenue.

- Some products sold **fewer units but had high revenue**.
- Others sold **many units but with lower revenue per sale**.

10. Total Number of Orders

Code:

```
total_orders = df.shape[0]
 print("Total Number of Orders:", total_orders)
```

Result Interpretation:

The dataset recorded **254 orders** in total.

This represents the number of sales transactions within the time frame.

Discussions

- **Payment Preference** – Most customers used Credit Card. This means the business should keep credit card payments reliable and secure.
- **Best-Selling Products** – Beverages were bought the most in number, but Burgers brought in the highest money. This shows that some products sell more by volume, while others sell fewer but bring in more revenue.
- **City & Manager Performance** – Lisbon made the most sales, and Joao Silva was the manager with the highest revenue. This shows strong performance in that city and by that manager.

- **Revenue Behavior** – On average, each transaction made about ₦3029.59. However, revenue was not steady; it went up during holiday periods.
- **Sales Variability** – Sales and revenue were not consistent. The wide difference in values shows that some days or products performed much better than others.
- **Trend Analysis** – Sales and revenue increased towards December, likely because of holiday shopping.
- **Product Revenue Balance** – Some products earned more money even with fewer sales, while others relied on selling in large quantities.

Recommendations

- Keep and improve Credit Card payment options since most customers prefer it.
- Focus marketing on **Beverages** (high volume) and **Burgers** (high revenue).
- Strengthen business in **Lisbon**, while checking why other cities are not doing as well.
- Use **Joao Silva** as a performance example for other managers.
- Get ready for festive seasons by stocking more products and running promotions.
- Work on strategies to make sales more stable and reduce big ups and downs.