Bright Coffee Shop Methodology

Python Version

Busile Ndhlovu

Table of Contents

Objective

Extract actionable insights from historical data to help grow the company's revenue and improve product performance.

Data Overview

- The dataset from Bright Coffee Shop contain information about the shops transactions and product details.
- The dataset was from the months January, February, March, April, May and June from the year 2023.
- There was a total of 149116 transactions.
- The dataset provided also revealed that the store opens at 6:00 am and closes at 21:00 pm.

Tools and Techniques

- Planning: Miro for Data Flow & Architecture Diagram.
- Data Analysis Tools: Python on Google Colab for statistical analysis and visualization.
- Visualization Tools: PowerPoint for visualization.

Understanding the dataset

1.Handle Missing Values:

Code used to find missing values in all columns of the user profile table.

Missing values in this case are considered as NULL values.

```
[ ]  # Check for missing values
     print(df.isnull())
```

Returned Output:

```
        transaction_id  transaction_date  transaction_time  transaction_qty  \
0                False             False             False            False
1                False             False             False            False
2                False             False             False            False
3                False             False             False            False
4                False             False             False            False
...                ...               ...               ...              ...
124156           False             False             False            False
124157           False             False             False            False
124158           False             False             False            False
124159           False             False             False            False
124160           False             False             False            False

        store_id  store_location  product_id  unit_price  product_category  \
0          False           False       False       False             False
1          False           False       False       False             False
2          False           False       False       False             False
3          False           False       False       False             False
4          False           False       False       False             False
...          ...             ...         ...         ...               ...
124156     False           False       False       False             False
124157     False           False       False       False             False
124158     False           False       False       False             False
```

Returns 0 Null values.

2.Checking for duplicates

```
[9]  #checking for duplicates
     df.duplicated().sum()

     np.int64(0)
```

Returned 0 Duplicates

3.Checking the DataType

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148029 entries, 0 to 148028
Data columns (total 14 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   STORE_LOCATION           148029 non-null  object
 1   PRODUCT_CATEGORY         148029 non-null  object
 2   PRODUCT_TYPE             148029 non-null  object
 3   PRODUCT_DETAIL           148029 non-null  object
 4   T_DATE                   148029 non-null  object
 5   DAYS_OF_THE_WEEK         148029 non-null  object
 6   MONTH_NAME               148029 non-null  object
 7   T_TIME                   148029 non-null  object
 8   TRANSACTION_TIME_BUCKET  148029 non-null  object
 9   TOTAL_TRANSACTIONS       148029 non-null  int64
 10  UNIQUE_PRODUCTS_SOLD     148029 non-null  int64
 11  TOTAL_SALES              148029 non-null  int64
 12  TOATAL_AMOUNT            148029 non-null  int64
 13  TIME_BUCKET              148029 non-null  object
dtypes: int64(4), object(10)
memory usage: 15.8+ MB
```

4.Checking the column names for the table

```
df.columns
```

```
Index(['STORE_LOCATION', 'PRODUCT_CATEGORY', 'PRODUCT_TYPE', 'PRODUCT_DETAIL',
       'T_DATE', 'DAYS_OF_THE_WEEK', 'MONTH_NAME', 'T_TIME',
       'TRANSACTION_TIME_BUCKET', 'TOTAL_TRANSACTIONS', 'UNIQUE_PRODUCTS_SOLD',
       'TOTAL_SALES', 'TOATAL_AMOUNT', 'TIME_BUCKET'],
      dtype='object')
```

5.Changing the time and date data type

```
[27] # Convert 'transaction_time' and 'transaction_date' to datetime format
     df['transaction_time'] = pd.to_datetime(df['transaction_time'])
     df['transaction_date'] = pd.to_datetime(df['transaction_date'])
```

6.Changing the format of the digit in unit price from , to .

This is done for easy data processing using a comma instead might cause errors or misinterpretations during calculations.

```
[29] # Convert 'unit_price' to float, fixing any comma issues (e.g., '3,1' → 3.1)
     df['unit_price'] = df['unit_price'].replace(',', '.', regex=True).astype(float)
```

7. Creating a Transaction time bucket with 1 hour intervals

```
[30] # Create 'transaction_time_bucket' with 30-minute intervals (or modify for 1 hour)
     df['transaction_time_bucket'] = df['transaction_time'].dt.floor('1h')
```

8. Calculating the total amount

```
[ ]  # Compute 'total_amount'
     df['total_amount'] = df['unit_price'] * df['transaction_qty']
```

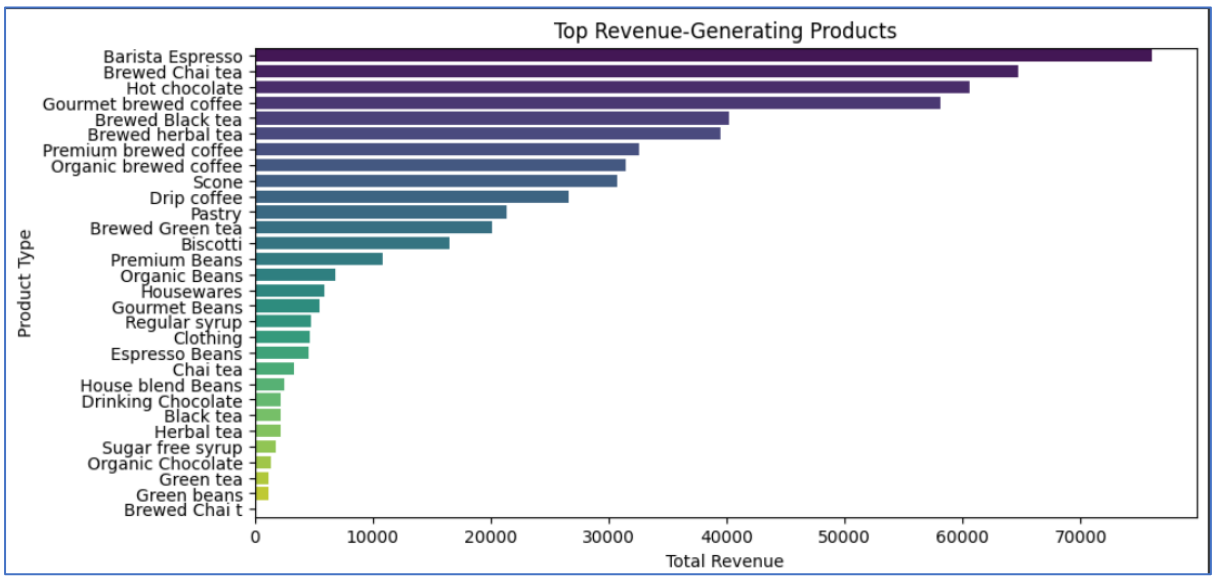
9.Generating a graph that displays the top Revenue generating product types

```
# Aggregate revenue by product type
product_revenue = df.groupby('product_type')['total_amount'].sum().reset_index()
product_revenue = product_revenue.sort_values(by='total_amount', ascending=False)

# Plot bar chart
plt.figure(figsize=(10, 5))
sns.barplot(x='total_amount', y='product_type', data=product_revenue, palette='viridis')
plt.xlabel("Total Revenue")
plt.ylabel("Product Type")
plt.title("Top Revenue-Generating Products")
plt.show()
```

Returned Output:

Which products generate the most revenue

The table shows the best-selling product type to the least, the top 6 products make up 60% of our total revenue while the 6 underperforming make up 2% of the total revenue.



10. Create a time bucket

Grouping people this way helps capture behaviours and needs unique to each time frame.

Also Created an time bracket to group individuals into meaningful, manageable categories for analysis and decision-making. Specifically, here's how I created an time bracket with differences like 4 hours years between categories:

Time Bracket	Category
6:00 – 9:00 am	Early Morning
10:00 – 13:00 pm	Mid-Morning
14:00 : 17:00 pm	Afternoon
18:00 -21:00 pm	Evening

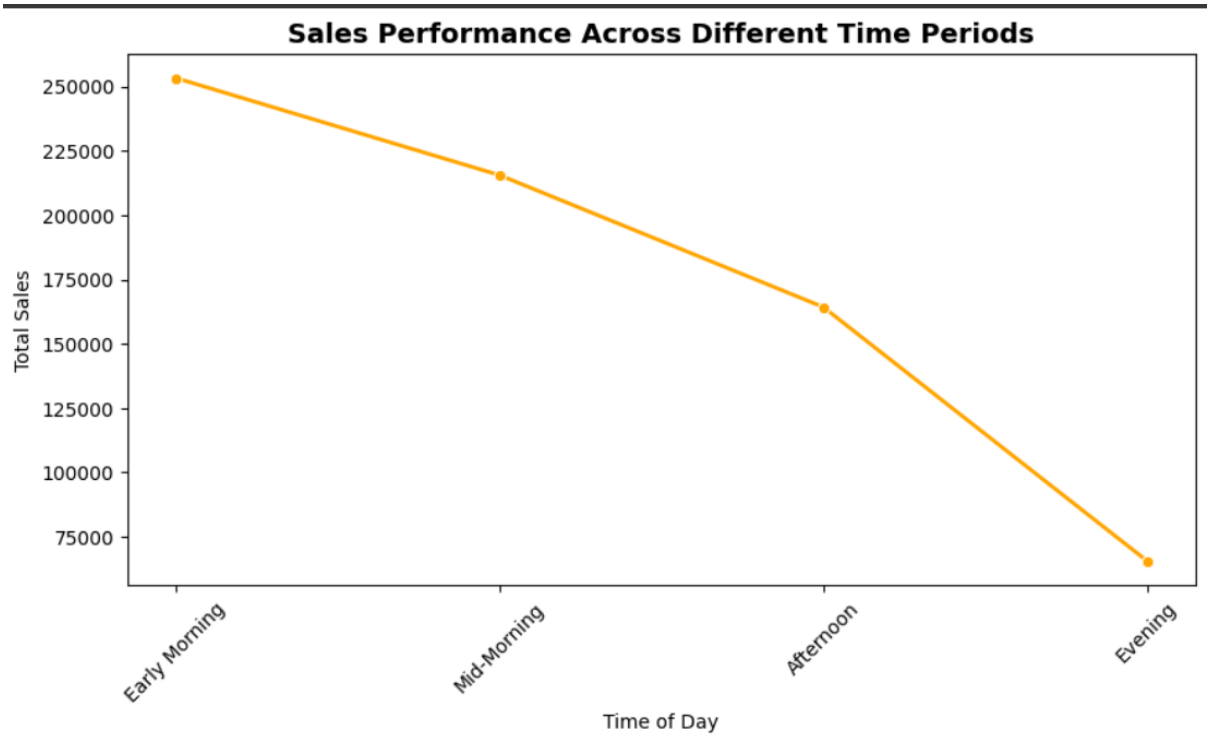
```python
#time bucket
def categorize_time_of_day(hour):
    if 6 <= hour <= 9:
        return 'Early Morning'
    elif 10 <= hour <= 13:
        return 'Mid-Morning'
    elif 14 <= hour <= 17:
        return 'Afternoon'
    elif 18 <= hour <= 20:
        return 'Evening'
    else:
        return 'Unknown'
```

11. Creating a line graph that displays Sales Performance over different Time periods

```python
[51] # Apply the function to create the 'time_of_day' column
    df['time_of_day'] = df['transaction_time'].dt.hour.apply(categorize_time_of_day)

    # Group sales by 'time_of_day' instead of 'transaction_time_bucket'
    time_bucket_sales = df.groupby('time_of_day')['total_amount'].sum().reset_index()

    # Sort categories in logical order
    time_bucket_sales['time_of_day'] = pd.Categorical(
        time_bucket_sales['time_of_day'],
        categories=["Early Morning", "Mid-Morning", "Afternoon", "Evening"],
        ordered=True
    )

    time_bucket_sales = time_bucket_sales.sort_values('time_of_day')
```

```python
    # Graph
    plt.figure(figsize=(10, 5))
    sns.lineplot(x='time_of_day', y='total_amount', data=time_bucket_sales, marker='o', color='

    plt.xlabel("Time of Day")
    plt.ylabel("Total Sales")
    plt.title("Sales Performance Across Different Time Periods" ,fontsize=14, fontweight='bold'

    plt.xticks(rotation=45)  # Rotate labels for better readability
    plt.show()
```

Retuned Output:



The peak sale time is from 6:00 – 9:00 am

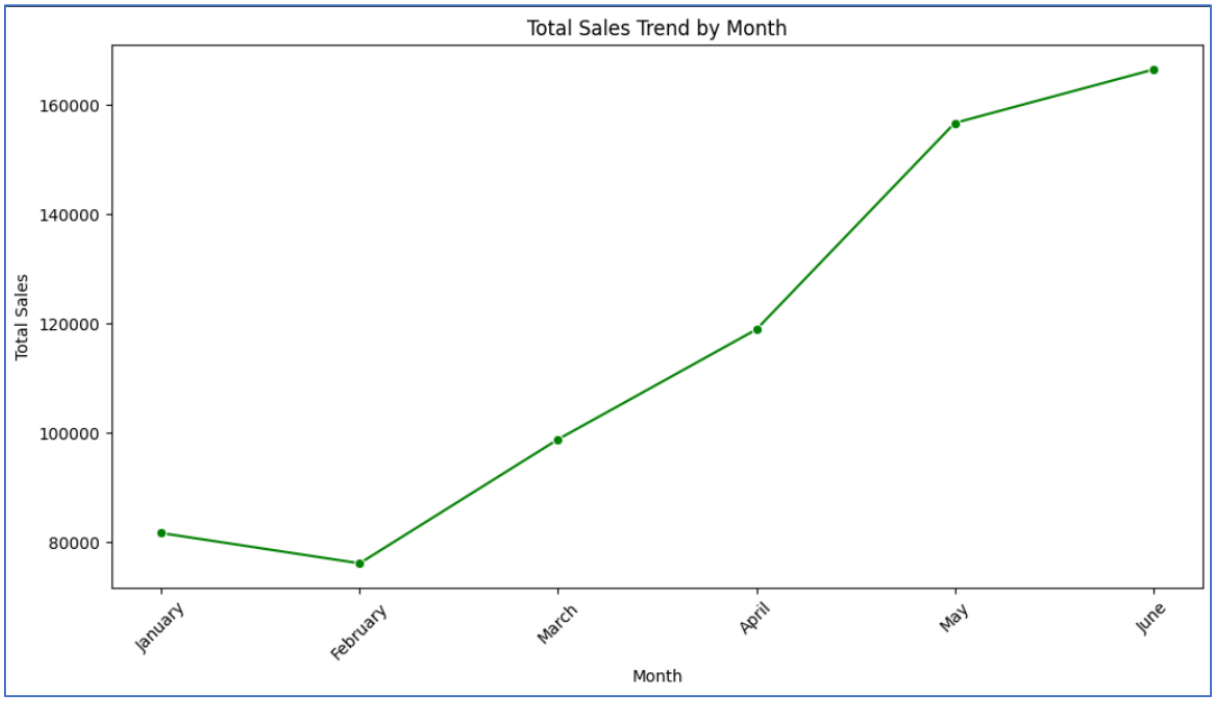
12.Total Sales Trend

From this graph we will be able see the stores peak consumption seasons.

```python
import calendar  # Import the calendar module

# Convert 'transaction_date' to datetime format if not already done
df['transaction_date'] = pd.to_datetime(df['transaction_date'])

# Extract full month name
df['month'] = df['transaction_date'].dt.month.apply(lambda x: calendar.month_name[x])


monthly_sales = df.groupby('month')['total_amount'].sum().reset_index()

# Convert 'month' to categorical type for correct chronological ordering
monthly_sales['month'] = pd.Categorical(
    monthly_sales['month'],
    categories=[
        "January", "February", "March", "April", "May", "June",
        "July", "August", "September", "October", "November", "December"
    ],
    ordered=True
)
```

```
# Sort properly
monthly_sales = monthly_sales.sort_values(by="month")


plt.figure(figsize=(12, 6))
sns.lineplot(x='month', y='total_amount', data=monthly_sales, marker='o', color="green")

plt.xlabel("Month")
plt.ylabel("Total Sales")
plt.title("Total Sales Trend by Month")
plt.xticks(rotation=45)
plt.show()
```

Returned Output:



14.Total Sales By Store Location

The distribution of sales on the different store locations.

```
[75]

    # Define a nude & coffee color palette
    nude_coffee_colors = ['#E3C8A8', '#D2B49A', '#A67B5B', '#8C624A', '#603E31']

    # Pie chart visualization
    plt.figure(figsize=(6, 6))
    plt.pie(store_sales['total_amount'], labels=store_sales['store_location'], autopct='%1.1f%%',
            colors=nude_coffee_colors, startangle=140)

    plt.title("🍩 Total Sales by Store Location")
    plt.show()
```

Returned Output:

This could mean the stores follow the same pricing strategy, inventory stocking, and promotions, leading to even revenue shares.

Recommendations for Growth

Revenue Growth Strategies

1. Enhance Best-Selling Products

- Introduce **premium versions** or seasonal variations of the top 6 products to boost sales.
- Offer **subscription plans** for regular customers to ensure repeat purchases.

2.Revamp Underperforming Items

- **Bundle slower-selling items** with popular products to increase visibility.
- Launch **limited-time promotions** to test demand before deciding whether to phase them out.

3.Optimize Peak Hours Sales

- Introduce **breakfast bundles** during **Early Morning** (6–9 AM) and **Mid-Morning** (10 AM–1 PM).
- Implement **express service options** during peak hours to reduce waiting times.

4.Customer Engagement Initiatives

Loyalty Program & Subscription Model

- Reward frequent customers with **discounts, freebies, or exclusive early access** to new menu items.
- Introduce a **coffee subscription** for regular customers (e.g., unlimited coffee for a fixed monthly fee).

5.Personalized Marketing

- Use **customer purchase history** to send targeted promotions (e.g., a discount on a customer's favourite drink).
- Leverage **SMS or app notifications** to remind customers about special offers based on peak sales times.
-

Store & Sales Optimization

1.Expand Popular Store Locations

- Consider **opening new branches** in high-performing areas.
- Optimize **staffing levels** to ensure enough employees during peak hours.

2.Weekend Promotional Campaigns

- Since weekdays perform better than weekends, introduce **weekend-exclusive promotions** (e.g., “Saturday Family Coffee Discount”).
- Organize **community events or coffee-tasting experiences** to draw in more foot traffic.

3.Seasonal and Monthly Promotions

- Based on monthly sales data, create **seasonal offers** aligned with high-demand months.
- Offer **holiday-themed products** to drive sales during festive seasons.

Initiatives for Growth

1. Product Strategy

- Enhance best-selling items → Introduce limited-edition flavors or seasonal versions.
- Revamp slow-selling products → Adjust pricing, reposition or bundle them with popular items.

2. Peak Time Optimization

- Improve early morning efficiency → Train staff to handle high-demand hours.
- Create express menu options → Faster service during peak rush to increase sales volume.

3. Operational Improvements

- Adjust staffing during peak hours → Ensure high sales periods have sufficient employee coverage.
- Optimize inventory stocking → Align product availability with high-demand periods.

4. Data-Driven Decision Making

- Continuously monitor trends → Implement ongoing data analysis to refine strategy.
- Leverage customer preferences → Use purchase history insights for targeted marketing campaigns.

Conclusion

The analysis of Bright Coffee Shop's sales data has provided valuable insights into the business's revenue drivers, customer behaviour, and product performance. By identifying **best-selling products**, **peak consumption hours**, and **high-performing days**, we can strategically refine operations and marketing efforts to maximize growth.

A few key takeaways:

- **Best-selling products** contribute **60%** of total sales, requiring further optimization for maximum revenue.
- **Underperforming products** account for only **2%**, needing adjustments in pricing, marketing, or removal.
- **Early morning (6–9 AM) and mid-morning (10 AM–1 PM)** drive the highest revenue, highlighting prime business hours.
- **Seasonal patterns** suggest targeted promotions should align with high-performing months.

By implementing **strategic growth initiatives**, Bright Coffee Shop can **increase revenue**, **enhance customer engagement**, and **boost overall business performance**.

This analysis equips **Bright Coffee Shop** with a clear path for **revenue growth and product performance improvements**. By implementing **data-driven strategies**, the company can strengthen its competitive position and build long-term customer.