

Python Script



BRIGHT COFFEE SHOP

Sales Analysis Presentation

1. Loading the libraries (pandas, numpy, metlablib, seaborn, plotly) and loading the data on Colab and displaying the table

```
[62] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from datetime import time
#Avoid to give unnecessary warnings or errors
import warnings
warnings.filterwarnings('ignore')

[63] df = pd.read_csv("/content/drive/MyDrive/Coffee Shop Sales (3) CSV.csv", delimiter=';')

[64] #Display the current table
display(df)
```

	transaction_id	transaction_date	transaction_time	transaction_qty	store_id	store_location	product_id	unit_price	product_category	product_type	product_de
0	1	2023/01/01	07:06:11	2	5	Lower Manhattan	32	3	Coffee	Gourmet brewed coffee	Ethiop
1	2	2023/01/01	07:08:56	2	5	Lower Manhattan	57	3,1	Tea	Brewed Chai tea	Spicy Eye Of Ch
2	3	2023/01/01	07:14:04	2	5	Lower Manhattan	59	4,5	Drinking Chocolate	Hot chocolate	Dark choc
3	4	2023/01/01	07:20:24	1	5	Lower Manhattan	22	2	Coffee	Drip coffee	Our Old Diner Blen
4	5	2023/01/01	07:22:41	2	5	Lower Manhattan	57	3,1	Tea	Brewed Chai tea	Spicy Eye Of Ch
...
149111	149452	2023/06/30	20:18:41	2	8	Hell's Kitchen	44	2,5	Tea	Brewed herbal tea	Peppermi
149112	149453	2023/06/30	20:25:10	2	8	Hell's Kitchen	49	3	Tea	Brewed Black tea	En Breakfa
149113	149454	2023/06/30	20:31:34	1	8	Hell's Kitchen	45	3	Tea	Brewed herbal tea	Peppermi
149114	149455	2023/06/30	20:57:19	1	8	Hell's Kitchen	40	3,75	Coffee	Barista Espresso	Cappuc
149115	149456	2023/06/30	20:57:19	2	8	Hell's Kitchen	64	0,8	Flavours	Regular syrup	Hazelnut s

2. Checking the number of rows and columns in the table and the statistics of the numerical columns

✓
0s

[70] #Show the number of columns and rows
df.shape

↔

(149116, 11)

✓
0s

[71] df.describe()

↔

	transaction_id	transaction_qty	store_id	product_id
count	149116.000000	149116.000000	149116.000000	149116.000000
mean	74737.371872	1.438276	5.342063	47.918607
std	43153.600016	0.542509	2.074241	17.930020
min	1.000000	1.000000	3.000000	1.000000
25%	37335.750000	1.000000	3.000000	33.000000
50%	74727.500000	1.000000	5.000000	47.000000
75%	112094.250000	2.000000	8.000000	60.000000
max	149456.000000	8.000000	8.000000	87.000000

3. Checking the format of each of the columns

✓ [72] df.dtypes



0

transaction_id	int64
transaction_date	object
transaction_time	object
transaction_qty	int64
store_id	int64
store_location	object
product_id	int64
unit_price	object
product_category	object
product_type	object
product_detail	object

dtype: object

4. Changing the format of the unit price column to be a numeric column

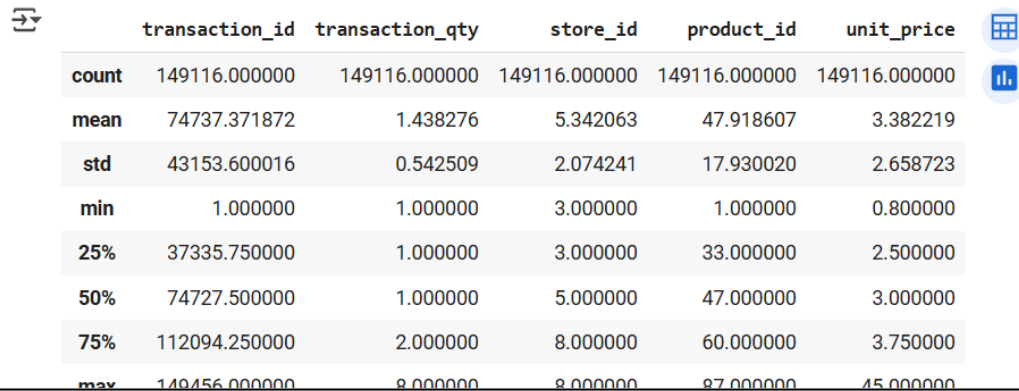
```
[74] #Remove any non-numeric characters (like currency symbols and spaces) in the unit price column  
df['unit_price'] = df['unit_price'].replace(r'[^\d.,]', '', regex=True)
```

```
[75] #Replace comma with dot if comma is used as decimal separator in the unit price column  
df['unit_price'] = df['unit_price'].str.replace(',', '.', regex=False)
```

```
[76] #Convert to float safely in the unit price column  
df['unit_price'] = pd.to_numeric(df['unit_price'], errors='coerce').fillna(0.0)
```

```
[77] #Round to 2 decimals (if needed) in the unit price column  
df['unit_price'] = df['unit_price'].round(2)
```

```
[78] df.describe()
```



	transaction_id	transaction_qty	store_id	product_id	unit_price
count	149116.000000	149116.000000	149116.000000	149116.000000	149116.000000
mean	74737.371872	1.438276	5.342063	47.918607	3.382219
std	43153.600016	0.542509	2.074241	17.930020	2.658723
min	1.000000	1.000000	3.000000	1.000000	0.800000
25%	37335.750000	1.000000	3.000000	33.000000	2.500000
50%	74727.500000	1.000000	5.000000	47.000000	3.000000
75%	112094.250000	2.000000	8.000000	60.000000	3.750000
max	149456.000000	8.000000	8.000000	87.000000	45.000000

5. Displaying the table

✓ [79] display(df)

ransaction_id	transaction_date	transaction_time	transaction_qty	store_id	store_location	product_id	unit_price	product_category	product_type	product_detail
1	2023/01/01	07:06:11	2	5	Lower Manhattan	32	3.00	Coffee	Gourmet brewed coffee	Ethiopia Rg
2	2023/01/01	07:08:56	2	5	Lower Manhattan	57	3.10	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg
3	2023/01/01	07:14:04	2	5	Lower Manhattan	59	4.50	Drinking Chocolate	Hot chocolate	Dark chocolate Lg
4	2023/01/01	07:20:24	1	5	Lower Manhattan	22	2.00	Coffee	Drip coffee	Our Old Time Diner Blend Sm
5	2023/01/01	07:22:41	2	5	Lower Manhattan	57	3.10	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg
...
149452	2023/06/30	20:18:41	2	8	Hell's Kitchen	44	2.50	Tea	Brewed herbal tea	Peppermint Rg
149453	2023/06/30	20:25:10	2	8	Hell's Kitchen	49	3.00	Tea	Brewed Black tea	English Breakfast Lg
149454	2023/06/30	20:31:34	1	8	Hell's Kitchen	45	3.00	Tea	Brewed herbal tea	Peppermint Lg
149455	2023/06/30	20:57:19	1	8	Hell's Kitchen	40	3.75	Coffee	Barista Espresso	Cappuccino
149456	2023/06/30	20:57:19	2	8	Hell's Kitchen	64	0.80	Flavours	Regular syrup	Hazelnut syrup

6. Checking the sum of the unit price and calculating total sales in the table

```
[80] #Checking the sum of the unit price column
total_unit_price = df['unit_price'].sum()
print(total_unit_price)
```

→ 504343.02999999997

```
[81] #Calculation to obtain Total sales
      df['total_sales'] = df['transaction_qty'] * df['unit_price']
```

```
✓ [82] display(df)
```

ion_id	transaction_date	transaction_time	transaction_qty	store_id	store_location	product_id	unit_price	product_category	product_type	product_detail	total_sales
1	2023/01/01	07:06:11	2	5	Lower Manhattan	32	3.00	Coffee	Gourmet brewed coffee	Ethiopia Rg	6.00
2	2023/01/01	07:08:56	2	5	Lower Manhattan	57	3.10	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg	6.20
3	2023/01/01	07:14:04	2	5	Lower Manhattan	59	4.50	Drinking Chocolate	Hot chocolate	Dark chocolate Lg	9.00
4	2023/01/01	07:20:24	1	5	Lower Manhattan	22	2.00	Coffee	Drip coffee	Our Old Time Diner Blend Sm	2.00
5	2023/01/01	07:22:41	2	5	Lower Manhattan	57	3.10	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg	6.20
...
10175	2023/01/02	08:12:11	2	5	Lower Manhattan	11	2.50	Tea	Brewed herbal	Delectable D	5.00

7. Checking the columns in the table and for duplicates

```
✓ [83] #Check the columns of the table  
0s df.columns
```

```
↔ Index(['transaction_id', 'transaction_date', 'transaction_time',  
        'transaction_qty', 'store_id', 'store_location', 'product_id',  
        'unit_price', 'product_category', 'product_type', 'product_detail',  
        'total_sales'],  
       dtype='object')
```

```
✓ [84] #Checking for any duplicates in the table  
0s df.duplicated().sum()
```

```
↔ 0
```


8. Creating time buckets

```
✓ [85] #Creating time buckets
3s df["transaction_time"] = pd.to_datetime(df["transaction_time"], errors='coerce')
times = df["transaction_time"].dt.time
peakmorningrush = (times >= time(6, 0, 0)) & (times <= time(8, 59, 59))
midmorning = (times >= time(9, 0, 0)) & (times <= time(11, 59, 59))
afternoon = (times >= time(12, 0, 0)) & (times <= time(15, 59, 59))
peakafternoonrush = (times >= time(16, 0, 0)) & (times <= time(17, 59, 59))
evening = ~peakmorningrush & ~midmorning & ~afternoon & ~peakafternoonrush
conditions = [peakmorningrush, midmorning, afternoon, peakafternoonrush, evening]
choices = ['peak morning rush', 'mid morning', 'afternoon', 'peak afternoon rush', 'evening']
df['Time Buckets'] = np.select(conditions, choices, default='Unknown')
```

```
✓ [86] display(df)
0s
```



transaction_date	transaction_time	transaction_qty	store_id	store_location	product_id	unit_price	product_category	product_type	product_detail	total_sales	Time Buckets
2023/01/01	2025-05-11 07:06:11	2	5	Lower Manhattan	32	3.00	Coffee	Gourmet brewed coffee	Ethiopia Rg	6.00	peak morning rush
2023/01/01	2025-05-11 07:08:56	2	5	Lower Manhattan	57	3.10	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg	6.20	peak morning rush
2023/01/01	2025-05-11 07:14:04	2	5	Lower Manhattan	59	4.50	Drinking Chocolate	Hot chocolate	Dark chocolate Lg	9.00	peak morning rush
2023/01/01	2025-05-11 07:20:24	1	5	Lower Manhattan	22	2.00	Coffee	Drip coffee	Our Old Time Diner Blend Sm	2.00	peak morning rush

✓ 0s completed at 7:27 PM

9. Creating a Month ID column

```
✓ [87] #Convert the transaction date column to datetime format  
0s df["transaction_date"] = pd.to_datetime(df["transaction_date"], errors='coerce')
```

```
✓ [88] #Extract year and month into a new Month ID Column  
0s df["Month_ID"] = df["transaction_date"].dt.strftime("%Y%m")
```

```
✓ [89] display(df)
```



_date	transaction_time	transaction_qty	store_id	store_location	product_id	unit_price	product_category	product_type	product_detail	total_sales	Time Buckets	Month_ID
-01-01	2025-05-11 07:06:11	2	5	Lower Manhattan	32	3.00	Coffee	Gourmet brewed coffee	Ethiopia Rg	6.00	peak morning rush	202301
-01-01	2025-05-11 07:08:56	2	5	Lower Manhattan	57	3.10	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg	6.20	peak morning rush	202301
-01-01	2025-05-11 07:14:04	2	5	Lower Manhattan	59	4.50	Drinking Chocolate	Hot chocolate	Dark chocolate Lg	9.00	peak morning rush	202301
-01-01	2025-05-11 07:20:24	1	5	Lower Manhattan	22	2.00	Coffee	Drip coffee	Our Old Time Diner Blend Sm	2.00	peak morning rush	202301
-01-01	2025-05-11 07:22:41	2	5	Lower Manhattan	57	3.10	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg	6.20	peak morning rush	202301


10. Creating a weekday column from the transaction_date column


```
[90] df['weekday'] = pd.to_datetime(df['transaction_date'], errors='coerce').dt.day_name()
```

```
[91] display(df)
```

transaction_time	transaction_qty	store_id	store_location	product_id	unit_price	product_category	product_type	product_detail	total_sales	Time Buckets	Month_ID	weekday
2025-05-11 07:06:11	2	5	Lower Manhattan	32	3.00	Coffee	Gourmet brewed coffee	Ethiopia Rg	6.00	peak morning rush	202301	Sunday
2025-05-11 07:08:56	2	5	Lower Manhattan	57	3.10	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg	6.20	peak morning rush	202301	Sunday
2025-05-11 07:14:04	2	5	Lower Manhattan	59	4.50	Drinking Chocolate	Hot chocolate	Dark chocolate Lg	9.00	peak morning rush	202301	Sunday
2025-05-11 07:20:24	1	5	Lower Manhattan	22	2.00	Coffee	Drip coffee	Our Old Time Diner Blend Sm	2.00	peak morning rush	202301	Sunday
2025-05-11 07:22:41	2	5	Lower Manhattan	57	3.10	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg	6.20	peak morning rush	202301	Sunday
...
2025-05-11 20:18:41	2	8	Hell's Kitchen	44	2.50	Tea	Brewed herbal tea	Peppermint Rg	5.00	evening	202306	Friday
2025-05-11 20:25:10	2	8	Hell's Kitchen	49	3.00	Tea	Brewed Black tea	English Breakfast Lg	6.00	evening	202306	Friday

11. Checking the sum of the Total_Sales column

```
 #Checking the sum of the total_Sales column  
total_sales = df['total_sales'].sum()  
print(total_sales)
```

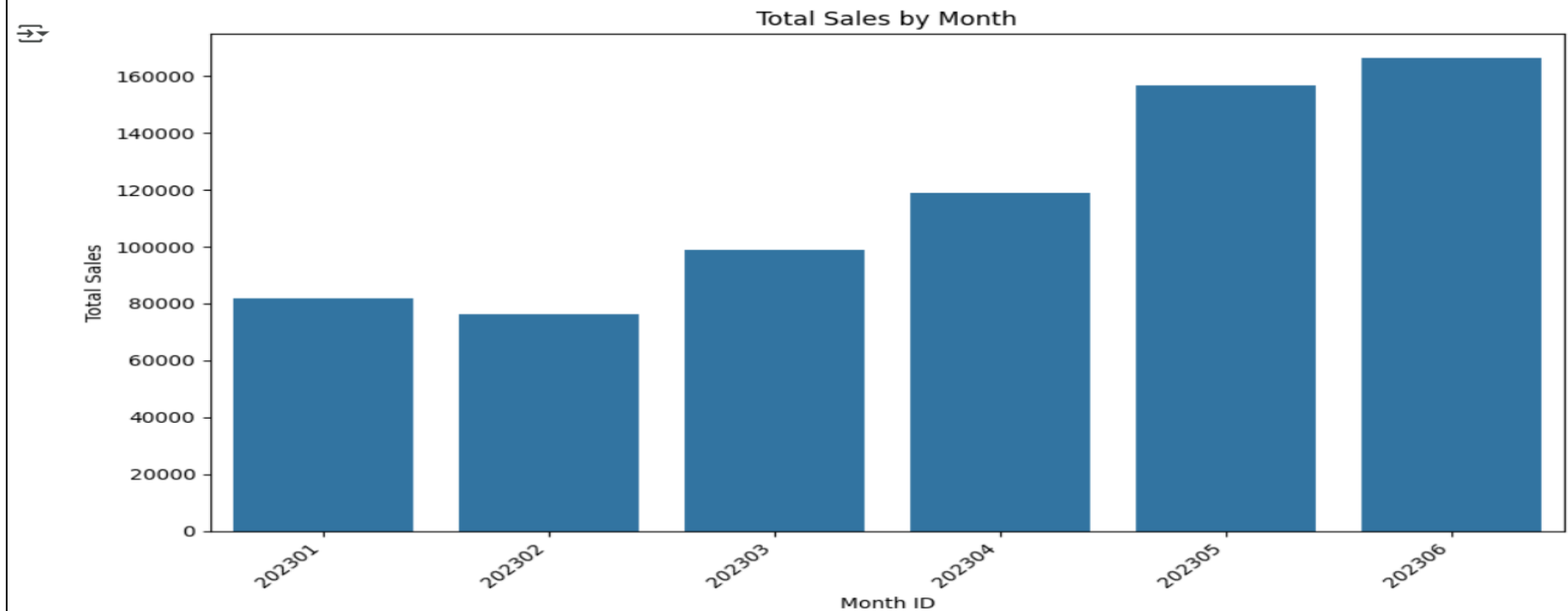
```
 698812.3300000002
```

12. Total sales by month bar graph

```
✓ [120] import matplotlib.pyplot as plt
0s

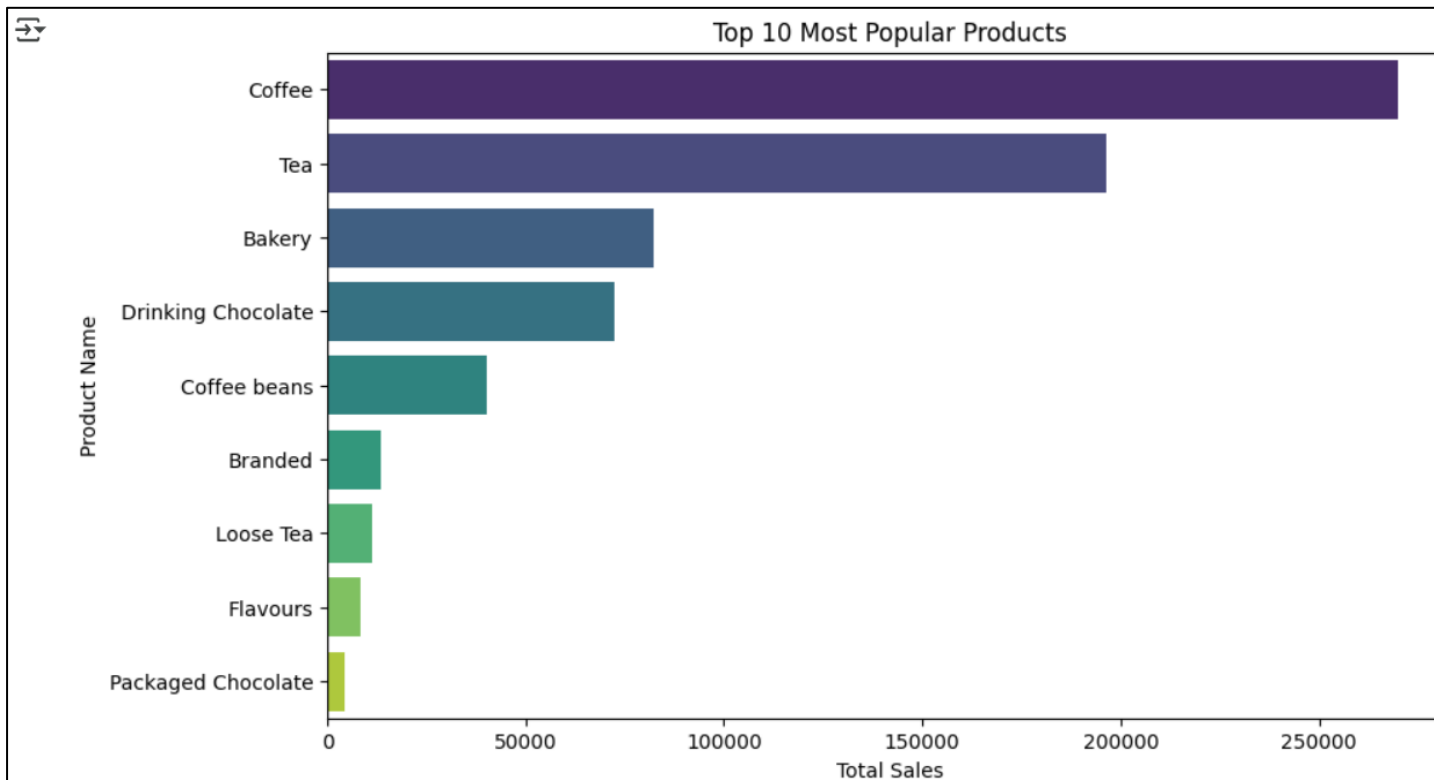
# Group by Month_ID and sum total sales
monthly_sales = df.groupby('Month_ID')['total_sales'].sum().reset_index()

# Plotting using seaborn (alternative)
plt.figure(figsize=(10, 6))
sns.barplot(data=monthly_sales, x='Month_ID', y='total_sales')
plt.xlabel("Month ID")
plt.ylabel("Total Sales")
plt.title("Total Sales by Month")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



13. Top 10 most popular products bar graph

```
✓ [93] # Product Popularity Visualization  
0s top_products = df.groupby('product_category')['total_sales'].sum().sort_values(ascending=False).head(10)  
plt.figure(figsize=(10, 6))  
sns.barplot(x=top_products.values, y=top_products.index, palette='viridis')  
plt.xlabel('Total Sales')  
plt.ylabel('Product Name')  
plt.title('Top 10 Most Popular Products')  
plt.show()
```



14. Total sales by store location bar graph

```
✓ [130] import matplotlib.pyplot as plt
0s      import seaborn as sns

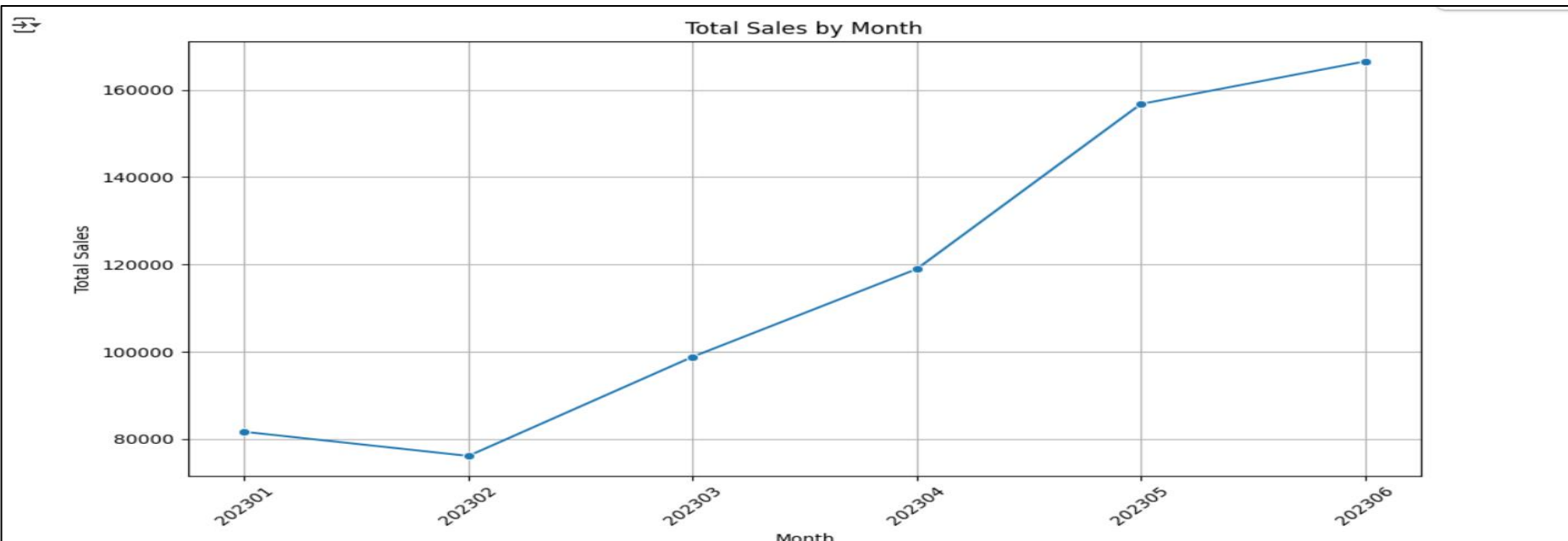
# Group by store_location and sum total sales
sales_by_location = df.groupby('store_location')['total_sales'].sum()

# Plotting using seaborn with custom color
plt.figure(figsize=(10, 6))
sns.barplot(x=sales_by_location.index, y=sales_by_location.values, color='#FFD700') # Change color here
plt.xlabel("Store Location")
plt.ylabel("Total Sales")
plt.title("Total Sales by Store Location")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



15. Total sales by month by line graph

```
✓ [103] monthly_sales = df.groupby('Month_ID')['total_sales'].sum().reset_index()  
0s  
  
✓ [104] plt.figure(figsize=(10, 6))  
0s      sns.lineplot(data=monthly_sales, x='Month_ID', y='total_sales', marker='o')  
  
      plt.title('Total Sales by Month')  
      plt.xlabel('Month')  
      plt.ylabel('Total Sales')  
      plt.xticks(rotation=45)  
      plt.grid(True)  
      plt.tight_layout()  
      plt.show()
```



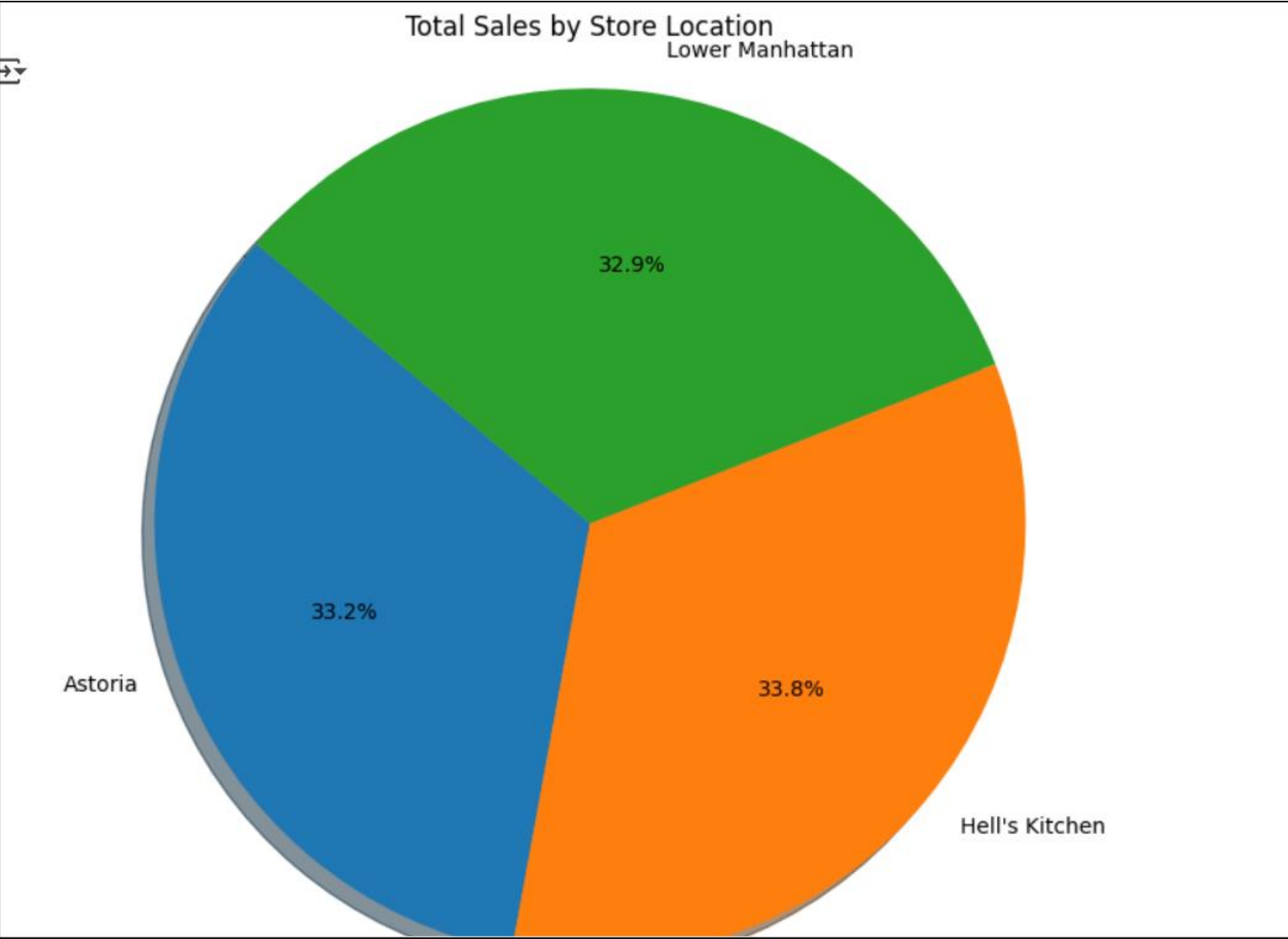
16. Total sales by store location pie chart

```
✓ [105] import matplotlib.pyplot as plt
0s

# Group by store_location and sum total sales
sales_by_location = df.groupby('store_location')['total_sales'].sum()

✓ [106] # Plot pie chart
0s
plt.figure(figsize=(8, 8))
plt.pie(
    sales_by_location,
    labels=sales_by_location.index,
    autopct='%1.1f%%',          # show percentages
    startangle=140,
    shadow=True
)

plt.title('Total Sales by Store Location')
plt.axis('equal') # Equal aspect ratio ensures pie is a circle
plt.show()
```

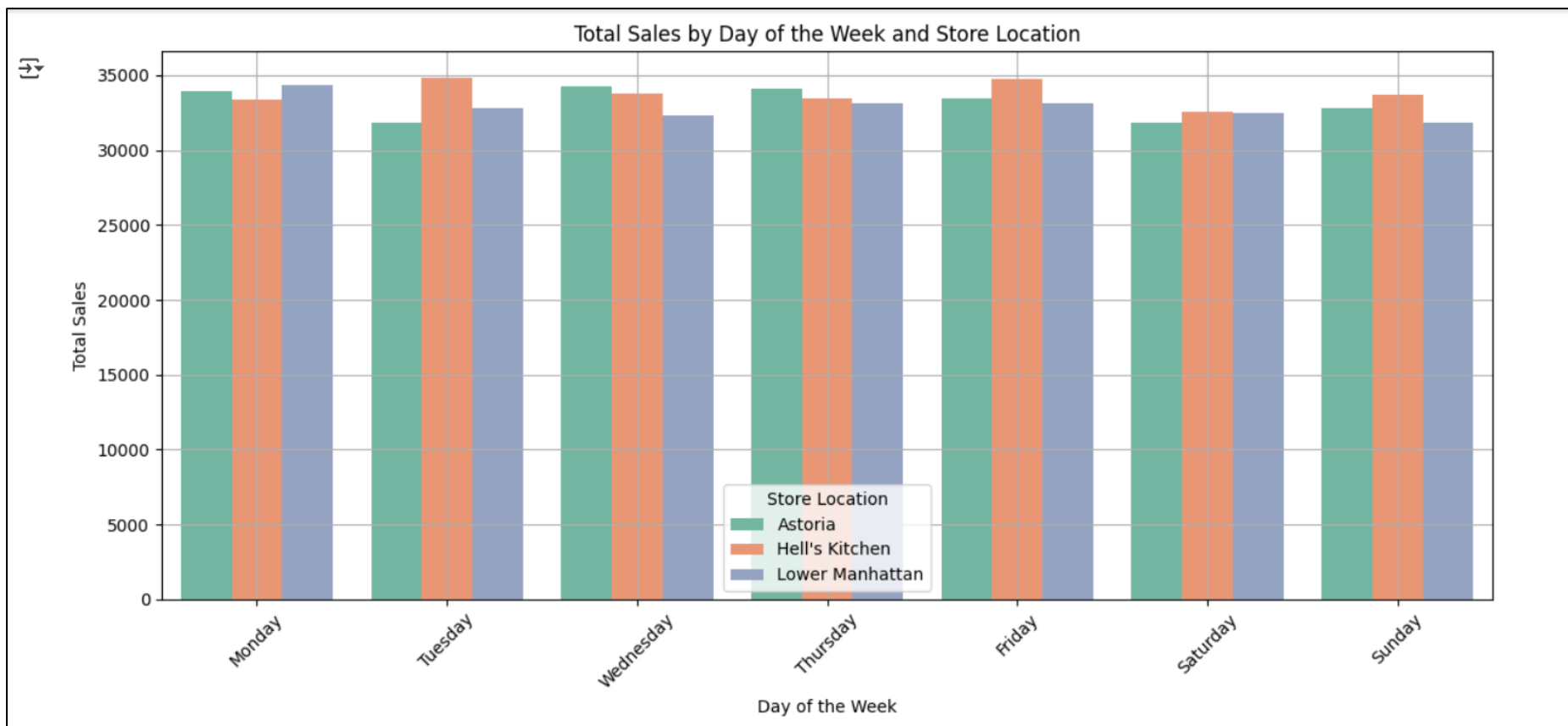


17. Total sales by day of the week and store location

```
✓ [110] sales_by_weekday_location = (  
0s      df.groupby(['weekday', 'store_location'])['total_sales']  
        .sum()  
        .reset_index()  
    )
```

```
▶ #Reorder weekdays for proper sorting  
weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']  
sales_by_weekday_location['weekday'] = pd.Categorical(sales_by_weekday_location['weekday'], categories=weekday_order, ordered=True)  
sales_by_weekday_location = sales_by_weekday_location.sort_values('weekday')
```

```
✓ [112] #Plotting the graph  
0s      plt.figure(figsize=(12, 6))  
      sns.barplot(  
          data=sales_by_weekday_location,  
          x='weekday',  
          y='total_sales',  
          hue='store_location',  
          palette='Set2'  
      )  
  
      plt.title('Total Sales by Day of the Week and Store Location')  
      plt.xlabel('Day of the Week')  
      plt.ylabel('Total Sales')  
      plt.xticks(rotation=45)  
      plt.legend(title='Store Location')  
      plt.tight_layout()  
      plt.grid(True)  
      plt.show()
```



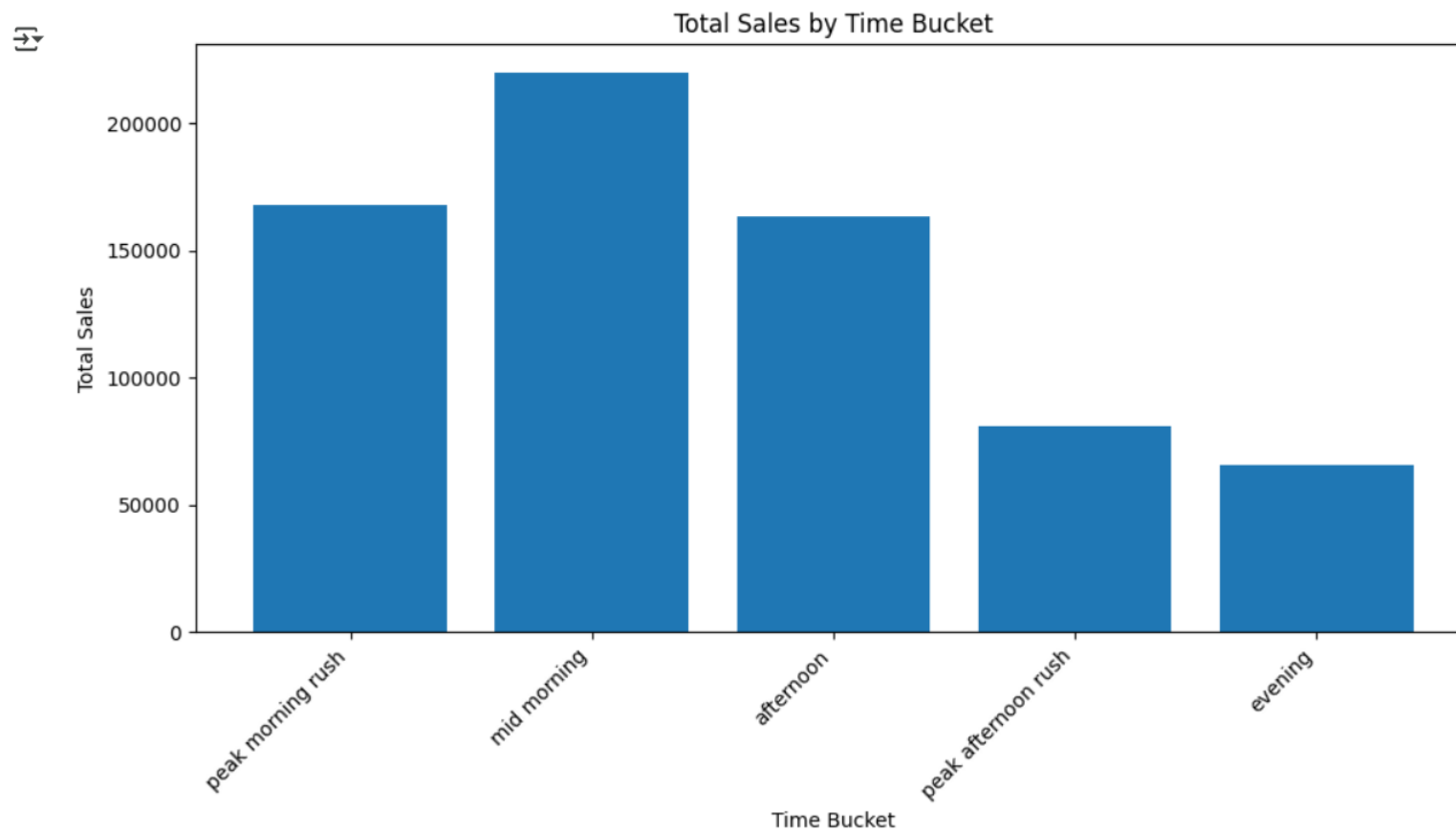
18. Total sales by time buckets

```
✓ [115] # Calculate total sales by time bucket
0s      sales_by_time_bucket = df.groupby('Time Buckets')['total_sales'].sum()

✓ [118] # Plotting using matplotlib
0s      plt.figure(figsize=(10, 6)) # Adjust figure size if needed
      plt.bar(sales_by_time_bucket.index, sales_by_time_bucket.values)
      plt.xlabel("Time Bucket")
      plt.ylabel("Total Sales")
      plt.title("Total Sales by Time Bucket")
      plt.xticks(rotation=45, ha="right") # Rotate x-axis labels for better readability
      plt.tight_layout() # Adjust layout to prevent labels from overlapping

      # Define the desired order of time buckets
      desired_order = ['peak morning rush', 'mid morning', 'afternoon', 'peak afternoon rush', 'evening']

      # Plotting using matplotlib
      plt.figure(figsize=(10, 6))
      plt.bar(desired_order, sales_by_time_bucket[desired_order])
      plt.xlabel("Time Bucket")
      plt.ylabel("Total Sales")
      plt.title("Total Sales by Time Bucket")
      plt.xticks(rotation=45, ha="right")
      plt.tight_layout()
      plt.show()
```



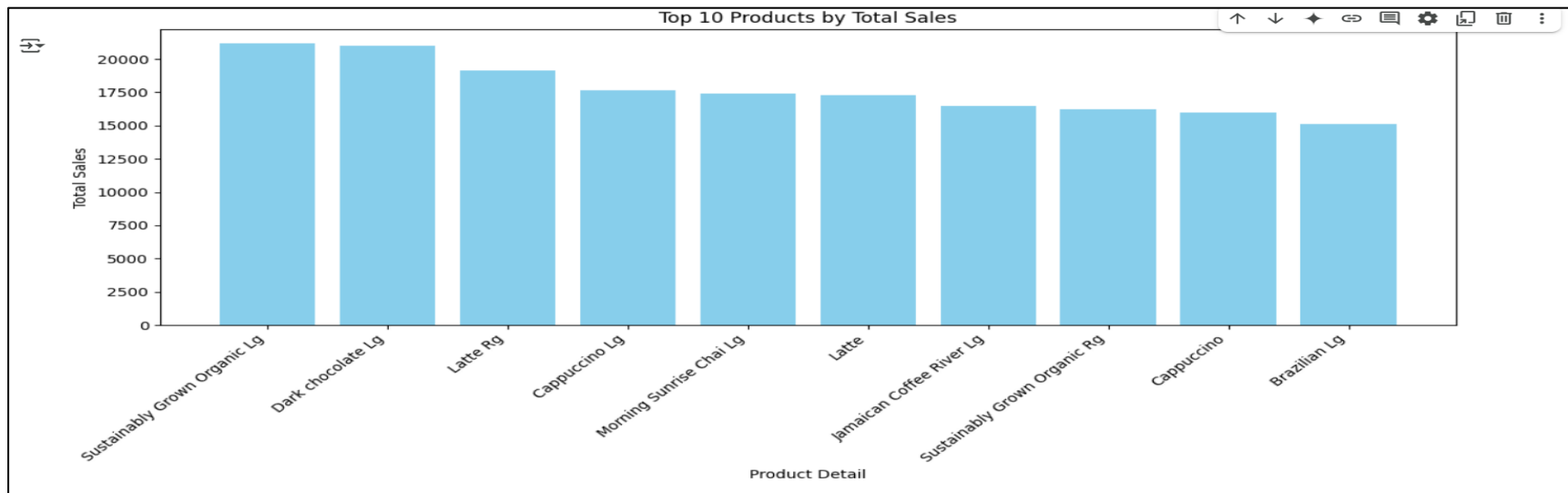
19. Top 10 products by total sales

```
✓ [125] import matplotlib.pyplot as plt
0s

# Group by product_detail and sum total sales
sales_by_product = df.groupby('product_detail')['total_sales'].sum().reset_index()

# Sort by total sales in descending order and select top 10
top_10_products = sales_by_product.sort_values(by='total_sales', ascending=False).head(10)

# Plotting using matplotlib with custom color
plt.figure(figsize=(12, 6))
plt.bar(top_10_products['product_detail'], top_10_products['total_sales'], color='skyblue')
plt.xlabel("Product Detail")
plt.ylabel("Total Sales")
plt.title("Top 10 Products by Total Sales")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



20. Total sales count by hour and weekday

```
✓ [131] # Extract hour of the day and weekday
0s      df['hour'] = df['transaction_time'].dt.hour

# Group by hour and weekday and count sales
sales_by_hour_weekday = df.groupby(['hour', 'weekday'])['total_sales'].count().reset_index()

# Pivot the data for heatmap
sales_heatmap = sales_by_hour_weekday.pivot(index='weekday', columns='hour', values='total_sales')

# Reorder weekdays for proper sorting
weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
sales_heatmap = sales_heatmap.reindex(weekday_order)

# Plot the heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(sales_heatmap, cmap='viridis', annot=True, fmt='d')
plt.title('Total Sales Count by Hour and Weekday')
plt.xlabel('Hour of the Day')
plt.ylabel('Weekday')
plt.show()
```




Total Sales Count by Hour and Weekday

