E-commerce analysis

> Project Introduction:

This project aims to analyze e-commerce data, which involves various aspects of online transactions. The datasets provided include information on customers, geolocations, order items, orders, products, sellers, and payments. Each of these datasets plays a critical role in understanding different dimensions of the e-commerce platform, from customer demographics and product details to order fulfillment and payment processing.

> Datasets Overview:

1. customers.csv: Contains data about the customers, such as unique customer identifiers, location information, and other demographic details.
2. geolocation.csv: Includes geographical information, like latitude and longitude, which can be linked to customer and seller addresses for spatial analysis.
3. order_items.csv: Holds details about the items included in each order, including product IDs, prices, and quantities.
4. orders.csv: Provides order-level data, including order status, purchase timestamps, and customer IDs, allowing for the analysis of order patterns and customer behavior.
5. products.csv: Contains information about the products available on the platform, such as product IDs, names, and categories.
6. seller.csv: Holds details about the sellers, including their location and unique seller identifiers.
7. payments.csv: Includes information on payment transactions, like payment types, amounts, and order associations.

> Motive of the Analysis:

This analysis is designed to unlock valuable insights into customer behavior, sales trends, and business performance within the e-commerce platform. By exploring the data through basic, intermediate, and advanced queries, we aim to uncover patterns and opportunities that can drive growth and optimize operations.

- This analysis is divided into three levels:

1. Basic Queries: These straightforward queries help us understand the essentials—where customers are located, how many orders were placed, and what products are popular. This level sets the stage by highlighting key metrics like total sales and unique customers.

2. Intermediate Queries: These queries dig deeper, revealing trends and relationships. We explore sales patterns over time, customer preferences by location, and revenue contributions by product category. This helps us identify what drives customer choices and how different factors impact sales.

3. Advanced Queries: These complex queries focus on long-term growth and customer retention. We analyze trends over time, identify top customers and products, and calculate growth rates. The insights here are crucial for strategic decisions, helping the business stay competitive and profitable.

"Importing main libraries used in this analysis."

In [1]:
```
# Object manipulation
import statistics
import numpy as np
import pandas as pd
from collections import defaultdict

# Plot
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.figure_factory as ff
import seaborn as sns

# SQL Connector
import mysql.connector
import os

# For controlling the display of warnings
import warnings
warnings.filterwarnings('ignore')
```

> Connection to MySQL database.

In [2]:
```
conn = mysql.connector.connect(host='localhost',
    user='root',
    password='sumit',
    database='ecommerce')
```

In [3]:
```
# Get tables
query = "SHOW TABLES;"
tables = pd.read_sql_query(query, conn)
```

In [4]:
```
tables
```

Out[4]:

	Tables_in_ecommerce
0	customers
1	geolocation
2	order_items
3	orders
4	payments
5	products
6	sellers

In [5]:
```
# Explore All Tables
```

In [6]:
```
# explore what type of data acailble in the tables
for table in tables['Tables_in_ecommerce']:
     querry = f"select * from {table} limit 2"
     display(f"{table}",pd.read_sql_query(querry,conn))
```

'customers'

	customer_id	customer_unique_id	customer_zip_code_
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	

'geolocation'

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city	geolocatior
0	1037	-23.5456	-46.6393	sao paulo	
1	1046	-23.5461	-46.6448	sao paulo	

'order_items'

	order_id	order_item_id	product_id	
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61	484
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	dd

'orders'

	order_id	customer_id	order_status	orde
0	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	
1	53cdb2fc8bc7dce0b6741e2150273451	b0830fb4747a6c6d20dea0b8c802d7ef	delivered	

'payments'

	order_id	payment_sequential	payment_type	payment_installmen
0	b81ef226f3fe1789b1e8b2acac839d17	1	credit_card	
1	a9810da82917af2d9aefd1278f1dcfa0	1	credit_card	

'products'

	product_id	product_category	product_name_length	product_descr
0	1e9e8ef04dbcff4541ed26657ea517e5	perfumery	40.0	
1	3aa071139cb16b67ca9e5dea641aaa2f	Art	44.0	

'sellers'

	seller_id	seller_zip_code_prefix	seller_city	seller_state
0	3442f8959a84dea7ee197c632cb2df15	13023	campinas	SP
1	d1b65fc7debc3361ea86b5f14c68d2e2	13844	mogi guacu	SP

Basic Queries Analysis

In [7]:
```
# 1.    List all unique cities where customers are located:
pd.read_sql_query("""select distinct customer_city from customers;""",conn)

# There are 4,118 unique cities where customers are located.
```

Out[7]:

	customer_city
0	franca
1	sao bernardo do campo
2	sao paulo
3	mogi das cruzes
4	campinas
...	...
4114	siriji
4115	natividade da serra
4116	monte bonito
4117	sao rafael
4118	eugenio de castro

4119 rows × 1 columns

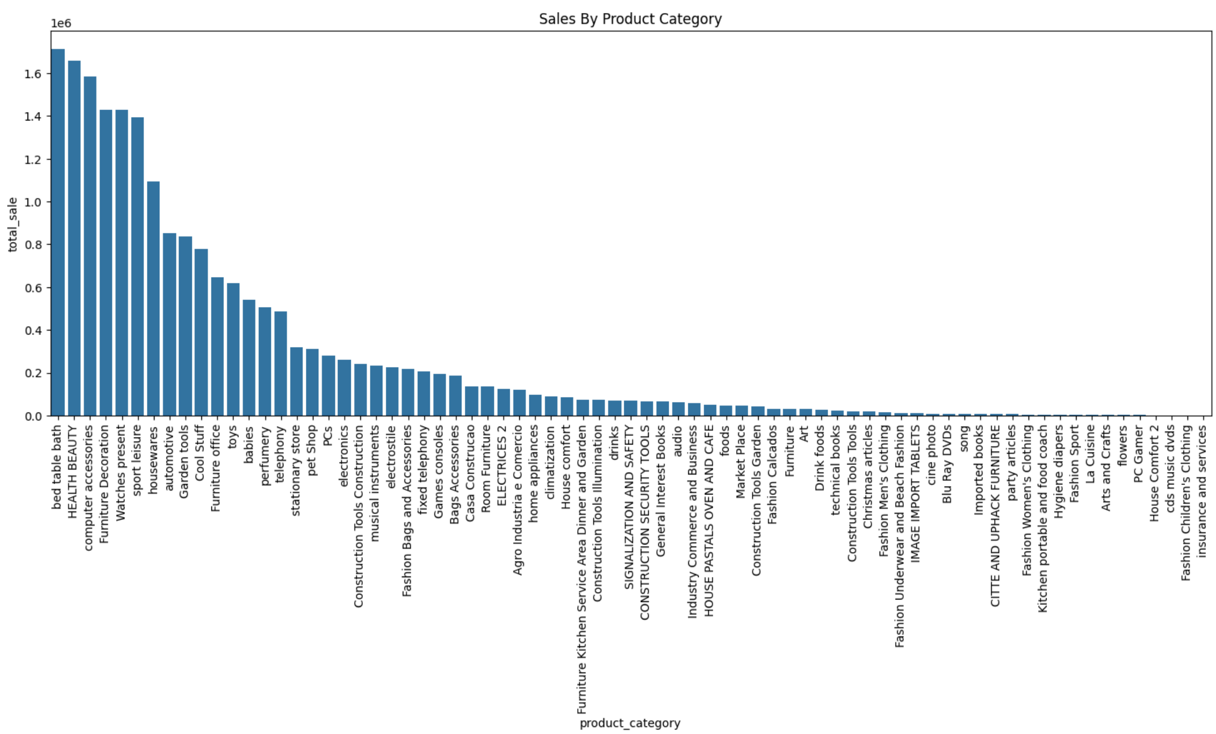
In [8]:
```python
# 2.    Count the number of orders placed in 2017:
pd.read_sql_query("""select count(order_id) as total_orders from orders
                  where year(order_purchase_timestamp)  = 2017 ;""",conn)
```

Out[8]:

	total_orders
0	45101

In [9]:
```python
# 3.    Find the total sales per category:
sales_per_category=pd.read_sql_query("""select product_category,round(sum(payment_v
                  join order_items as ot on p.product_id = ot.product_id
                  join payments py on py.order_id = ot.order_id
                  group by product_category
                  order by total_sale desc; """,conn)
```

In [10]:
```python
# Visualize sales by category
plt.figure(figsize=(18,6))
plt.title('Sales By Product Category')

sns.barplot(data=sales_per_category , x='product_category',y='total_sale')
plt.xticks(rotation=90)

plt.show()


sales_per_category.head(10)
```



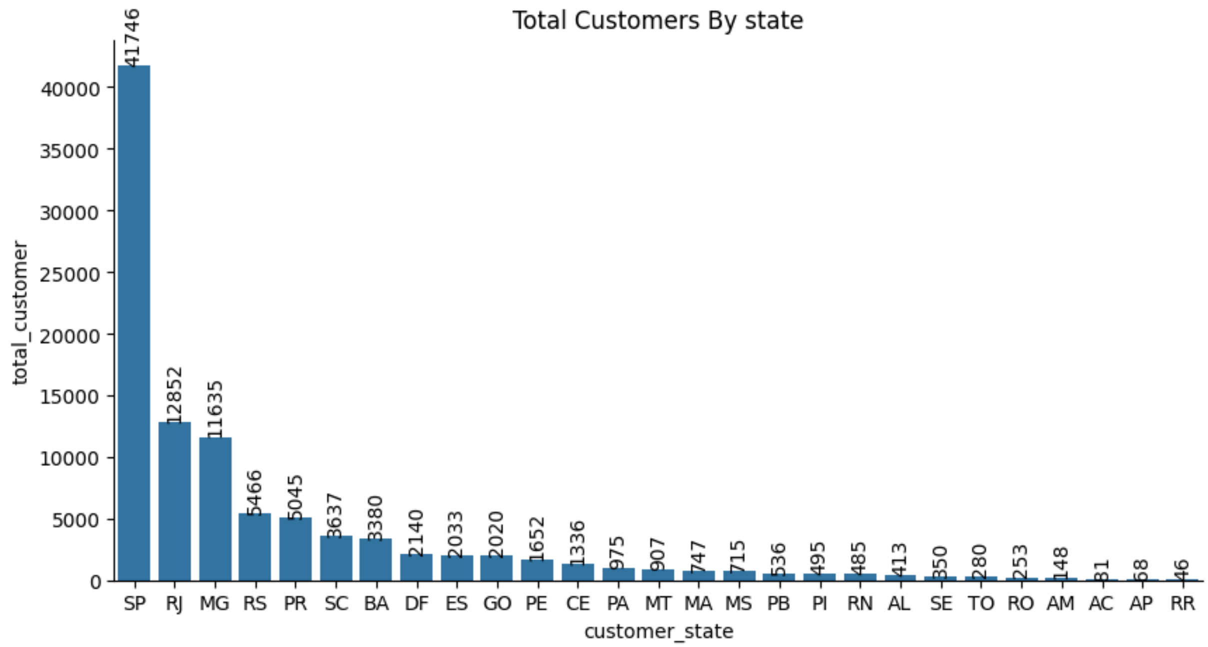
Out[10]:

	product_category	total_sale
0	bed table bath	1712553.67
1	HEALTH BEAUTY	1657373.12
2	computer accessories	1585330.45
3	Furniture Decoration	1430176.39
4	Watches present	1429216.68
5	sport leisure	1392127.56
6	housewares	1094758.13
7	automotive	852294.33
8	Garden tools	838280.75
9	Cool Stuff	779698.00

> The contribution of sales for bed bath products and health beauty products over the years had the highest sales overall.

In [11]:
```
# 5.    Count the number of customers from each state:
customers_by_state=pd.read_sql_query("""select customer_state,count(*) as total_cus
                    group by customer_state
                      order by total_customer desc""",conn)
```

In [12]:
```
plt.figure(figsize=(10,5))
ax=sns.barplot(data=customers_by_state,x='customer_state',y='total_customer')
plt.title('Total Customers By state')
ax.bar_label(ax.containers[0],rotation=90)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.show()
```



> São Paulo estado state having the higest num of customers;

In [13]:
```
# 6.    List all unique product categories available in the store:
pd.read_sql_query("""select distinct product_category from products""",conn)
```

Out[13]:

	product_category
0	perfumery
1	Art
2	sport leisure
3	babies
4	housewares
...	...
68	House Comfort 2
69	Kitchen portable and food coach
70	insurance and services
71	CITTE AND UPHACK FURNITURE
72	cds music dvds

73 rows × 1 columns

In [14]:
```
# 7.    Count the total number of unique customers:
pd.read_sql_query("""select count(distinct customer_id) as unique_customers from cu
```

Out[14]:

	unique_customers
0	99441

In [15]:
```
# 8.    Find the average order value across all orders:
pd.read_sql_query("""select avg(payment_value) as avg_order_value from payments""",
```

Out[15]:

	avg_order_value
0	154.10038

In [16]:
```
# 9.    List all unique payment methods used by customers:
pd.read_sql_query("""select distinct payment_type from payments;""",conn)
```

Out[16]:

	payment_type
0	credit_card
1	UPI
2	voucher
3	debit_card
4	not_defined

In [17]:
```
# 10.   Count the number of orders placed by each customer:

pd.read_sql_query("""select customer_id,count(order_id) as total_orders from orders
                  group by customer_id
                  order by total_orders desc""", conn)
```

Out[17]:

	customer_id	total_orders
0	e04757bb7741d0781cda14c9be20ad2a	1
1	96c3d51816ee07cb78ebbfe0e8bdb889	1
2	f1913159750548eb81dca4b69fbd5e0c	1
3	6967af003c642a30a79242f2756a73d5	1
4	5eef8c5a24bd948fac341c6ebe3ce41e	1
...	...	...
99436	69a6d39c85ef49a5896ac8a51eb09bb4	1
99437	fae68a5c34595ab2673163891d0c0aee	1
99438	690c457dfd68f1556cd9a6894e684c12	1
99439	1bd1508a2184ade648319b14e788bc6e	1
99440	6739c6c4a0a27db63e15c9c5a31d159a	1

99441 rows × 2 columns

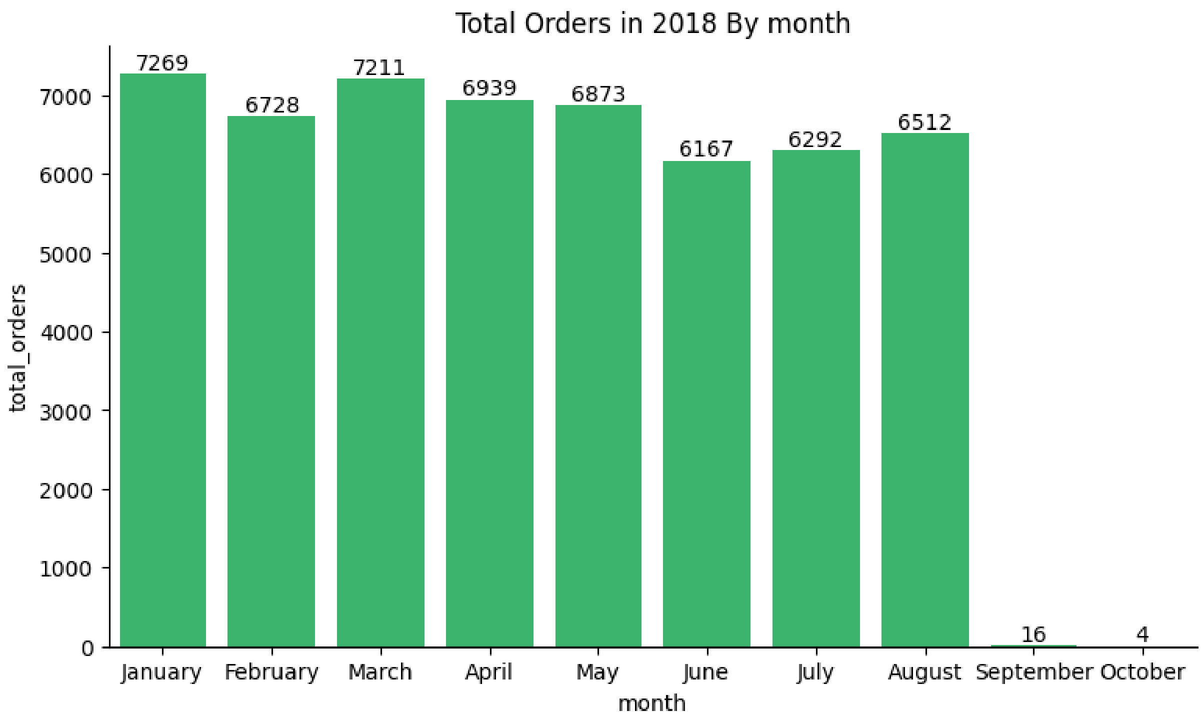
Intermediate Queries

In [18]:

```
# 1.    Calculate the number of orders per month in 2018:
orders_2018 = pd.read_sql_query(""" select monthname(order_purchase_timestamp) as m
                  where year(order_purchase_timestamp) = 2018
                  group by month(order_purchase_timestamp),monthname(order_purchase
                  order by month(order_purchase_timestamp);""",conn)
```

In [19]:

```
plt.figure(figsize=(9,5))
ax=sns.barplot(data= orders_2018,x='month',y='total_orders',color='#2cc86c')
plt.title('Total Orders in 2018 By month')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

ax.bar_label(ax.containers[0])
plt.show()
```



In [20]:
```
# 2.	Find the average number of products per order, grouped by customer city:
pd.read_sql_query("""select avg(num_of_product) as avg_products from (
                    select customer_city, count(product_id) as num_of_product
                    from customers c
                    join orders o on c.customer_id = o.customer_id
                    join order_items ot on ot.order_id = o.order_id
                    group by customer_city
                    order by num_of_product desc) a""",conn)
```

Out[20]:

	avg_products
0	27.4088

In [21]:
```
# 3.	Calculate the percentage of total revenue contributed by each product categ
pd.read_sql_query("""select product_category, concat(round(sum(payment_value)*100/(
                    from products p
                    join order_items ot on ot.product_id = p.product_id
                    join payments pm on pm.order_id = ot.order_id
                    group by product_category
                    """,conn)
```

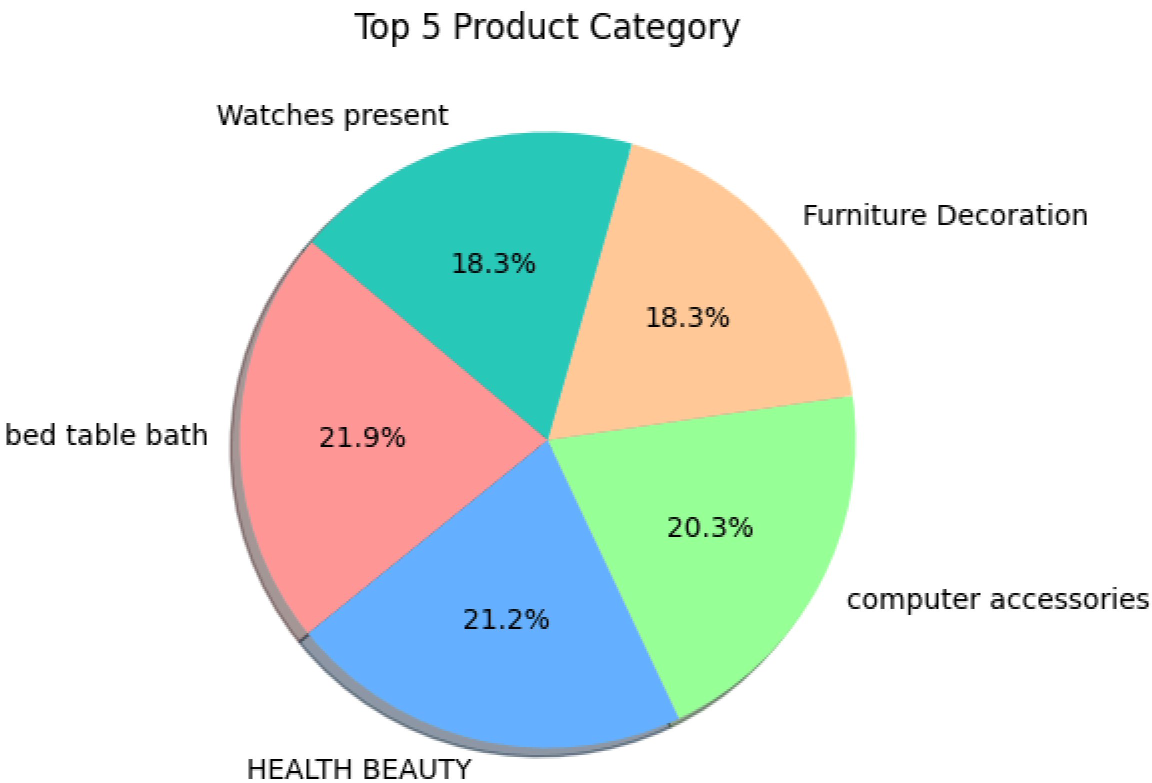
Out[21]:

	product_category	per_revenue
0	perfumery	3.17 %
1	Furniture Decoration	8.93 %
2	telephony	3.04 %
3	bed table bath	10.7 %
4	automotive	5.32 %
...	...	...
68	cds music dvds	0.01 %
69	La Cuisine	0.02 %
70	Fashion Children's Clothing	0 %
71	PC Gamer	0.01 %
72	insurance and services	0 %

73 rows × 2 columns

In [22]:

```
# For the pie chart, use the top 5 categories that contribute the maximum percentag
df= pd.read_sql_query("""select product_category,round(sum(payment_value)*100/(sele
                  from products p
                  join order_items ot on ot.product_id = p.product_id
                  join payments pm on pm.order_id = ot.order_id
                  group by product_category
                  order by per_revenue desc
                  limit 5
                  """,conn)


# Custom colors
colors = ['#ff9999','#66b3ff','#99ff99','#ffcc99','#2cc8ba']
plt.figure(figsize=(5, 5))
plt.pie(df['per_revenue'], labels=df['product_category'], autopct='%1.1f%%', starta
# plt.legend(title="Product Categories", loc="upper left", bbox_to_anchor=(1, 1))

plt.title('Top 5 Product Category')

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



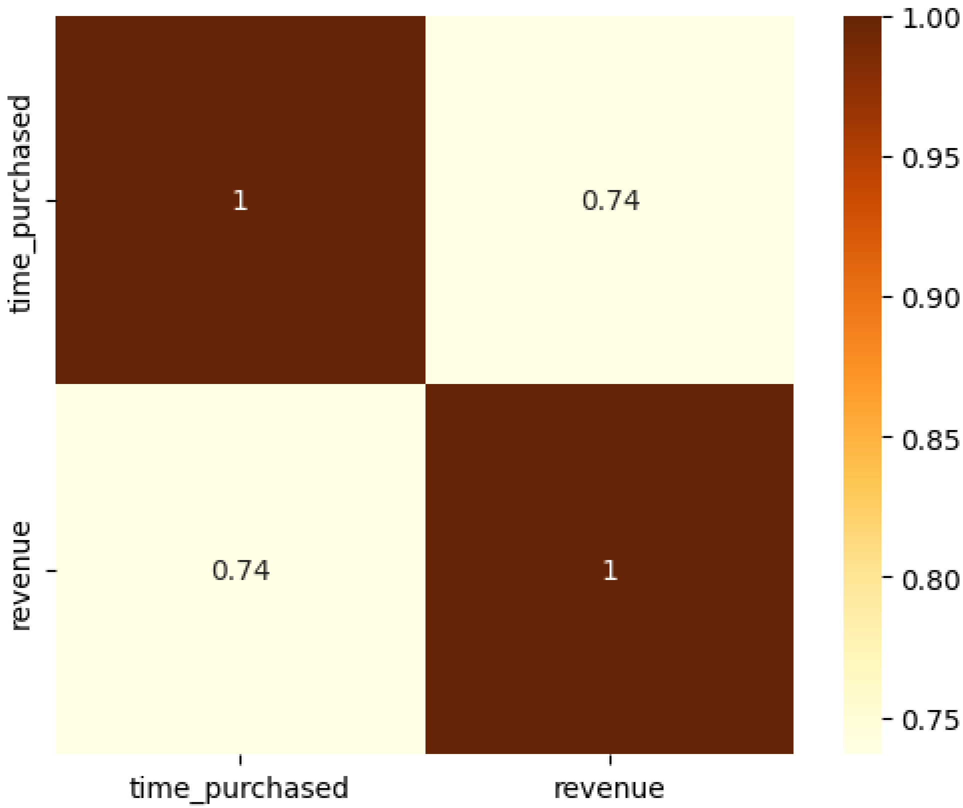
In [23]:
```
# 4.    Identify the correlation between product price and the number of times a pr
df=pd.read_sql_query("""select time_purchased, revenue from(
                    select p.product_id, product_category,count(ot.product_id) as
                  from products p
                  join order_items ot on ot.product_id = p.product_id
                  join payments pm on pm.order_id = ot.order_id
                  group by 1,2) a
                   """,conn)


corr= df.corr()
corr
```

Out[23]:

	time_purchased	revenue
time_purchased	1.000000	0.736903
revenue	0.736903	1.000000

In [24]:
```
sns.heatmap(corr,annot=True,cmap='YlOrBr',square=True)
```

Out[24]: <Axes: >



Insights from Correlation Heatmap

The heatmap shows a **strong positive correlation** (`0.74`) between `time_purchased` and `revenue`, indicating that as purchase time increases, revenue tends to increase as well.

In [25]:
```
# 5.    Calculate the total revenue generated by each seller, and rank them by reve
pd.read_sql_query("""select s.seller_id,round(sum(payment_value),2) as total_revenu
                  rank() over (order by sum(payment_value) DESC) as rank_r
                  from sellers s
                  join order_items ot on ot.seller_id = s.seller_id
                  join payments p on p.order_id = ot.order_id
                  group by 1""",conn)
```

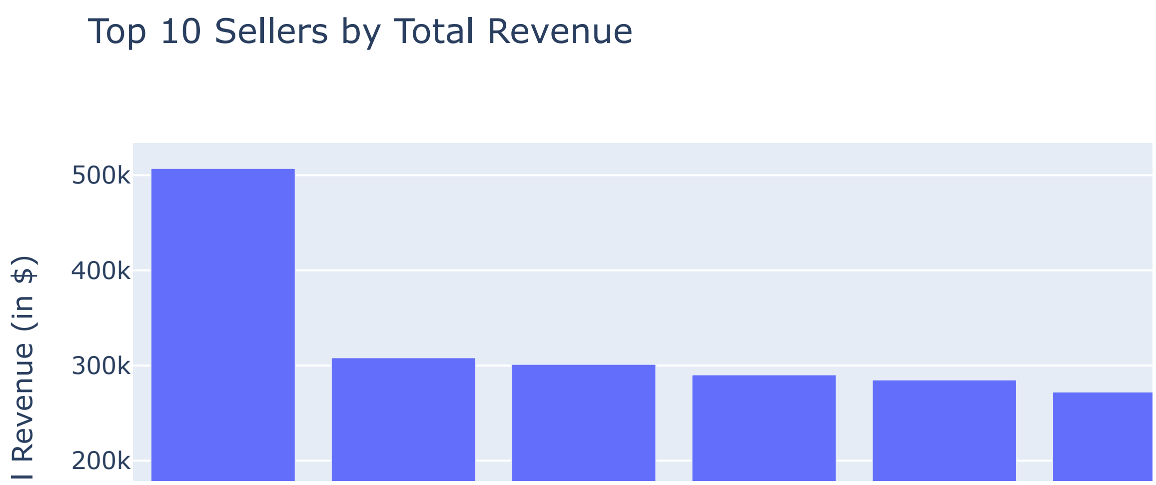
Out[25]:

	seller_id	total_revenue	rank_r
0	7c67e1448b00f6e969d365cea6b010ab	507166.91	1
1	1025f0e2d44d7041d6cf58b6550e0bfa	308222.04	2
2	4a3ca9315b744ce9f8e9374361493884	301245.27	3
3	1f50f920176fa81dab994f9023523100	290253.42	4
4	53243585a1d6dc2643021fd1853d8905	284903.08	5
...	...	...	...
3090	ad14615bdd492b01b0d97922e87cb87f	19.21	3091
3091	702835e4b785b67a084280efca355756	18.56	3092
3092	4965a7002cca77301c82d3f91b82e1a9	16.36	3093
3093	77128dec4bec4878c37ab7d6169d6f26	15.22	3094
3094	cf6f6bc4df3999b9c6440f124fb2f687	12.22	3095

3095 rows × 3 columns

In [26]:

```
# Top 10 sellers with the highest sales contribution.
df=pd.read_sql_query("""select s.seller_id,round(sum(payment_value),2) as total_rev
                  rank() over (order by sum(payment_value) DESC) as rank_r
                  from sellers s
                  join order_items ot on ot.seller_id = s.seller_id
                  join payments p on p.order_id = ot.order_id
                  group by 1
                  limit 10""",conn)


# Plot
fig = go.Figure(data=[go.Bar(x=df['seller_id'], y=df['total_revenue'])])


fig.update_layout(title="Top 10 Sellers by Total Revenue",
                  xaxis_title="Seller ID",
                  yaxis_title="Total Revenue (in $)",
                  xaxis=dict(tickmode='linear'))
fig.show()
df
```



Out[26]:

	seller_id	total_revenue	rank_r
0	7c67e1448b00f6e969d365cea6b010ab	507166.91	1
1	1025f0e2d44d7041d6cf58b6550e0bfa	308222.04	2
2	4a3ca9315b744ce9f8e9374361493884	301245.27	3
3	1f50f920176fa81dab994f9023523100	290253.42	4
4	53243585a1d6dc2643021fd1853d8905	284903.08	5
5	da8622b14eb17ae2831f4ac5b9dab84a	272219.32	6
6	4869f7a5dfa277a7dca6462dcf3b52b2	264166.12	7
7	955fee9216a65b617aa5c0531780ce60	236322.30	8
8	fa1c13f2614d7b5c4749cbc52fecda94	206513.23	9
9	7e93a43ef30c4f03f38b393420bc753a	185134.21	10

In [27]:

```
# 6.    Calculate the total number of products sold per category in 2018:
pd.read_sql_query("""select product_category, count(o.order_id) total_sold
                  from products p
                  join order_items ot on ot.product_id=p.product_id
```

```
                    join orders o on o.order_id = ot.order_id
                    where year(order_purchase_timestamp) = 2018
                    group by product_category
                    order by total_sold desc; """,conn)
```

Out[27]:

	product_category	total_sold
0	HEALTH BEAUTY	5951
1	bed table bath	5884
2	computer accessories	4708
3	sport leisure	4527
4	Furniture Decoration	4118
...	...	...
67	Fashion Sport	5
68	PC Gamer	5
69	La Cuisine	4
70	Fashion Children's Clothing	3
71	cds music dvds	1

72 rows × 2 columns

In [28]:

```
# .Total revenue and total number of orders by year for each category:
pd.read_sql_query("""select product_category,year(order_purchase_timestamp) as year
                     round(sum(payment_value),2) as total_revenue,count(o.order_id) t
                    from products p
                    join order_items ot on ot.product_id=p.product_id
                    join orders o on o.order_id = ot.order_id
                    join payments pm on pm.order_id = o.order_id
                    group by 1,2
                    order by year desc, total_sold desc;""",conn)
```

Out[28]:

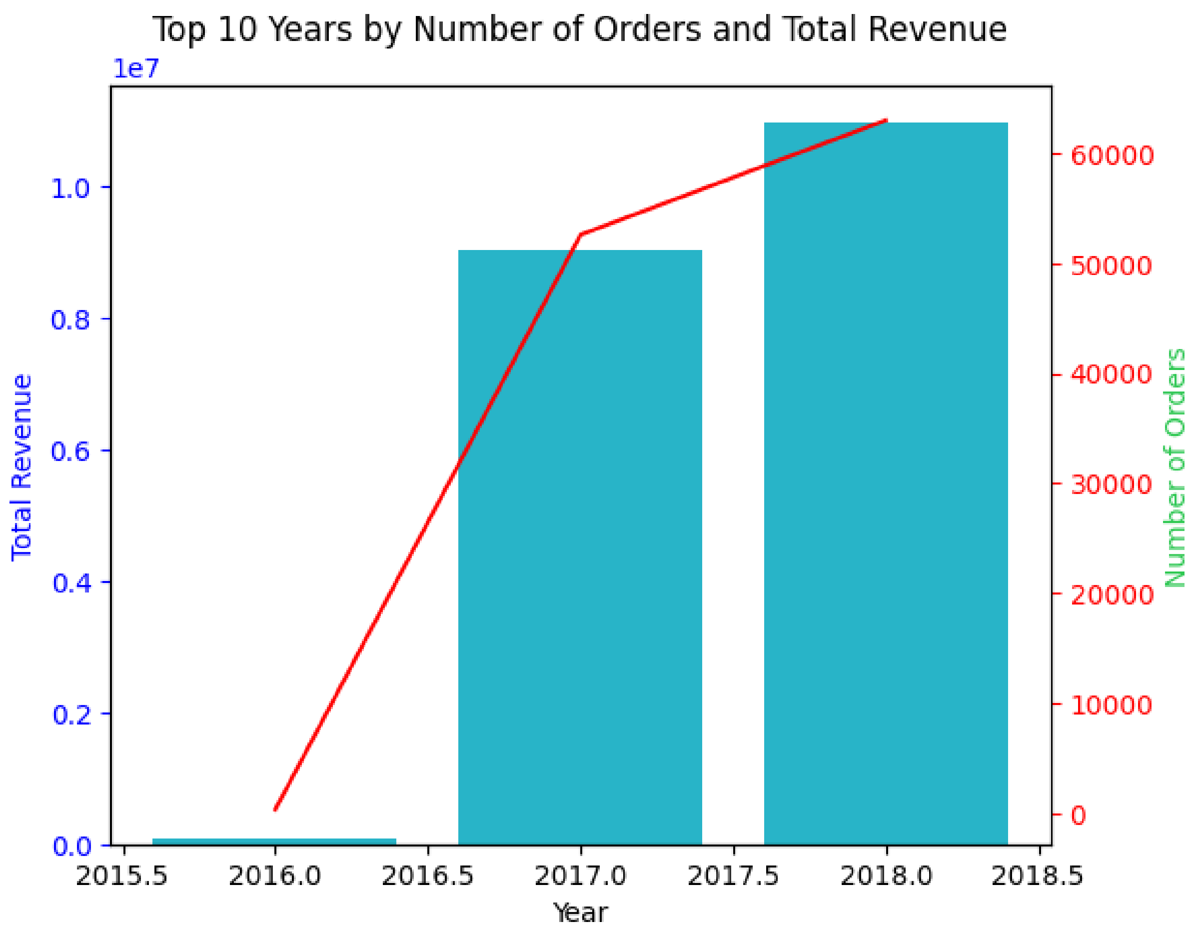
	product_category	year	total_revenue	total_sold
0	bed table bath	2018	912947.74	6169
1	HEALTH BEAUTY	2018	1033604.09	6117
2	computer accessories	2018	878236.82	4834
3	sport leisure	2018	746186.14	4663
4	Furniture Decoration	2018	744045.60	4289
...	...	...	...	...
170	technical books	2016	299.84	1
171	Fashion Calcados	2016	40.95	1
172	General Interest Books	2016	144.54	1
173	Hygiene diapers	2016	150.99	1
174	Fashion Men's Clothing	2016	35.86	1

175 rows × 4 columns

In [29]:

```
# Top 10 .Total revenue and total number of orders by year for each category:
top_10 =pd.read_sql_query("""select year(order_purchase_timestamp) as year,
                   round(sum(payment_value),2) as total_revenue,count(o.order_id) n
                  from products p
                  join order_items ot on ot.product_id=p.product_id
                  join orders o on o.order_id = ot.order_id
                  join payments pm on pm.order_id = o.order_id
                  group by 1
                  order by num_orders desc
                  limit 10;""",conn)

# Create a figure and axis object
fig, ax1 = plt.subplots()

# Plot the bar chart for total_revenue
ax1.bar(top_10['year'], top_10['total_revenue'], color='#2cb5c8')
ax1.set_xlabel('Year')
ax1.set_ylabel('Total Revenue', color='b')
ax1.tick_params('y', colors='b')

# Create a second axis object that shares the x-axis with ax1
ax2 = ax1.twinx()

# Plot the line chart for num_orders
ax2.plot(top_10['year'], top_10['num_orders'], color='r')
ax2.set_ylabel('Number of Orders', color='#2cc852')
ax2.tick_params('y', colors='r')

# Set the title and layout
fig.tight_layout()
plt.title('Top 10 Years by Number of Orders and Total Revenue')
```

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

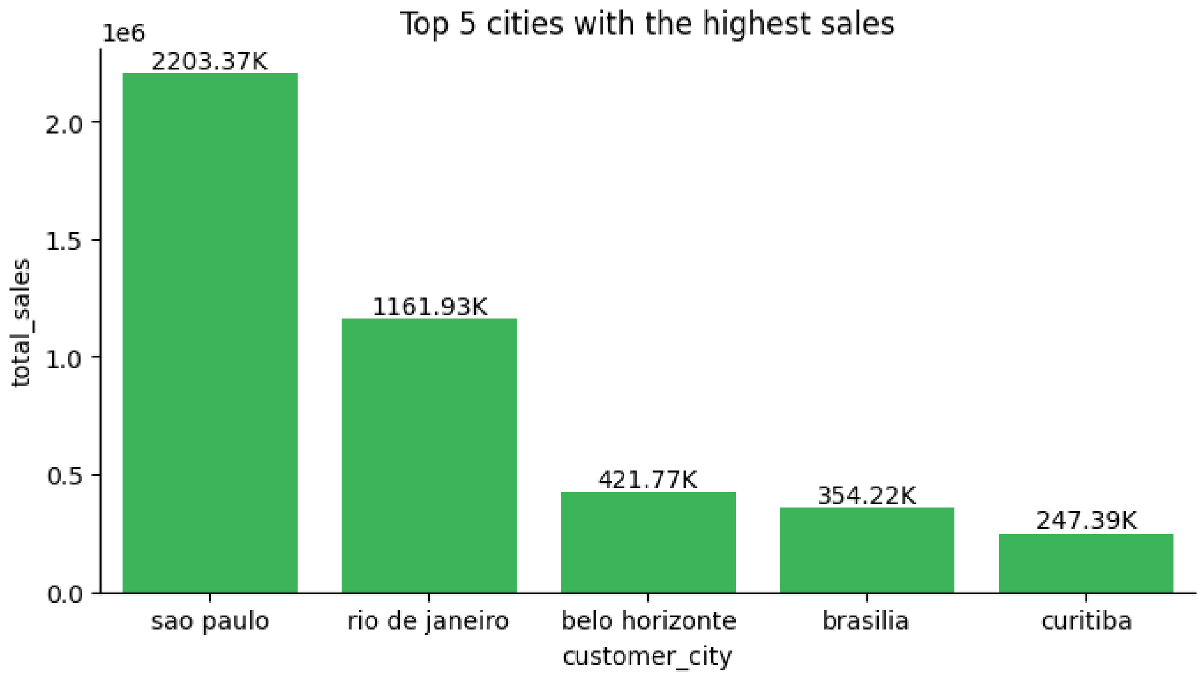


Insights from Orders and Revenue Over the Years

The chart shows a significant increase in both the number of orders and total revenue from 2016 to 2018, with the highest figures recorded in 2018. The steep rise from 2016 to 2017 highlights rapid growth in the business during this period.

In [30]:
```python
# 7.    Find the top 5 cities by total sales:

top_5_city=pd.read_sql_query("""select customer_city, round(sum(payment_value),2) a
                  join  orders o on c.customer_id = o.customer_id
                  join payments p on p.order_id = o.order_id
                  group by 1
                  order by total_sales desc
                  limit 5""", conn)

plt.figure(figsize=(8,4))
ax=sns.barplot(data=top_5_city,x='customer_city', y='total_sales', color='#2cc852')
plt.title('Top 5 cities with the highest sales')


# Format and add the bar labels
```

```
labels = [f'{val/1e3:.2f}K' for val in top_5_city['total_sales']]
ax.bar_label(ax.containers[0], labels=labels)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.show()

top_5_city
```



Out[30]:

	customer_city	total_sales
0	sao paulo	2203373.09
1	rio de janeiro	1161927.36
2	belo horizonte	421765.12
3	brasilia	354216.78
4	curitiba	247392.48

Insights from Top 5 Cities with the Highest Sales

São Paulo leads with the highest total sales (2.2M), followed by Rio de Janeiro (1.16M), while other cities like Belo Horizonte, Brasília, and Curitiba contribute significantly less, highlighting a concentration of sales in major metropolitan areas.

In [31]:

```
# 8.    Identify the top 10 products by total revenue:
sales_per_category=pd.read_sql_query("""select product_category,round(sum(payment_v
                  join order_items as ot on p.product_id = ot.product_id
                  join payments py on py.order_id = ot.order_id
                  join orders o on o.order_id = ot.order_id
                  group by product_category
                  order by total_sales desc
                  limit 10; """,conn)
```

```
fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(
    go.Bar(x=sales_per_category['product_category'],y=sales_per_category['total_sal
    secondary_y=False,)

fig.add_trace(go.Scatter(x=sales_per_category['product_category'],y=sales_per_categ
    secondary_y=True,)

# Set x-axis title
fig.update_xaxes(title_text="Product Category")

# Set y-axes titles
fig.update_yaxes(title_text="<b>Total Sales</b> (in USD)", secondary_y=False)
fig.update_yaxes(title_text="<b>Total Orders</b>", secondary_y=True)

# Here we modify the tickangle of the xaxis, resulting in rotated labels.
fig.update_layout(barmode='group',xaxis_tickangle=-45,title_text='Total Sales and O

fig.show()
```

Bed table bath has the highest sales among all product categories, followed by HEALTH BEAUTY and computer accessories.

The total orders decreases with the decrease in total sales. The total orders for the Cool Stuff product category are lowest among all product categories.

In [32]:
```
# 10.   Calculate the average time (in days) between a customer’s first and last or
pd.read_sql_query("""select customer_id , avg(time_diff) as avg_time from (
                  select customer_id,datediff(max(order_purchase_timestamp),min(ord
                  group by customer_id) a
                  group by customer_id;""", conn)


# The average time is zero because no customers are repeated; all customer IDs are
```

Out[32]:

	customer_id	avg_time
0	9ef432eb6251297304e76186b10a928d	0.0
1	b0830fb4747a6c6d20dea0b8c802d7ef	0.0
2	41ce2a54c0b03bf3443c3d931a367089	0.0
3	f88197465ea7920adcdbec7375364d82	0.0
4	8ab97904e6daea8866dbdbc4fb7aad2c	0.0
...	...	...
99436	39bd1228ee8140590ac3aca26f2dfe00	0.0
99437	1fca14ff2861355f6e5f14306ff977a7	0.0
99438	1aa71eb042121263aafbe80c1b562c9c	0.0
99439	b331b74b18dc79bcdf6532d51e1637c1	0.0
99440	edb027a75a1449115f6b43211ae02a24	0.0

99441 rows × 2 columns

Advanced Queries

In [33]:
```
# 1.    Calculate the moving average of order values for each customer over their o
pd.read_sql_query("""with customer_revenue as (
                  select c.customer_id , order_purchase_timestamp as order_date,sum
                  join orders o on o.customer_id = c.customer_id
                  join payments p on p.order_id = o.order_id
                  group by 1,2)
```

```
                  select customer_id,order_date,
                  round(avg(revenue) over (order by order_date ROWS BETWEEN UNBOUND
                                  from customer_revenue
                  group by 1,2
                  order by order_date""",conn)
```

Out[33]:

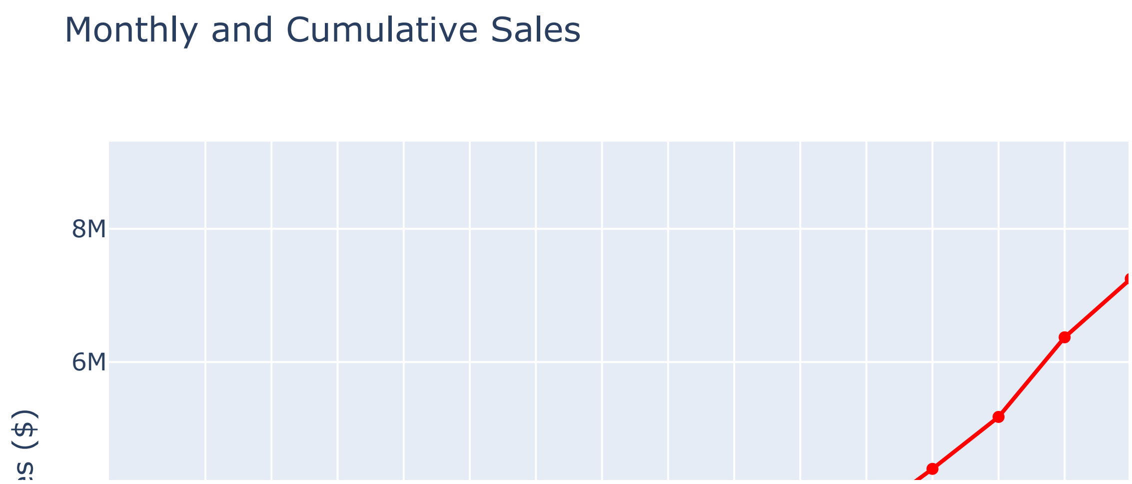
	customer_id	order_date	cumulative_avg
0	08c5351a6aca1c1589a38f244edeee9d	2016-09-04 21:15:19	136.23
1	683c54fc24d40ee9f8a6fc179fd9856c	2016-09-05 00:15:34	105.64
2	622e13439d6b5a0b486c435618b2679e	2016-09-13 15:24:19	84.08
3	b106b360fe2ef8849fbbd056f777b4d5	2016-10-02 22:07:52	90.39
4	355077684019f7f60a031656bd7262b8	2016-10-03 09:44:50	81.41
...	...	...	...
99435	2823ffda607a2316375088e0d00005ec	2018-09-29 09:13:03	160.99
99436	bf6181a85bbb4115736c0a8db1a53be3	2018-10-01 15:30:09	160.99
99437	4c2ec60c29d10c34bd49cb88aa85cfc4	2018-10-03 18:55:29	160.99
99438	856336203359aa6a61bf3826f7d84c49	2018-10-16 20:16:02	160.99
99439	a4b417188addbc05b26b72d5e44837a1	2018-10-17 17:30:18	160.99

99440 rows × 3 columns

In [34]:

```
# 2.    Calculate the cumulative sales per month for each year:
comm_sales=pd.read_sql_query("""with year_revenue as (select year(order_purchase_ti
                  from orders o
                  join payments p on p.order_id = o.order_id
                  group by 1,2,month(order_purchase_timestamp)
                  order by year,month(order_purchase_timestamp) asc)

                  select year,month,revenue,
                  sum(revenue) over (partition by year order by year ROWS BETWEEN U
                  from year_revenue
                  order by year""",conn)


fig = go.Figure()

# Plot the revenue data
fig.add_trace(go.Scatter(x=comm_sales['month'] + '-' + comm_sales['year'].astype(st
                         y=comm_sales['revenue'],
                         mode='lines+markers',
                         name='Monthly Revenue'))

# Plot the cumulative sales data
fig.add_trace(go.Scatter(x=comm_sales['month'] + '-' + comm_sales['year'].astype(st
                         y=comm_sales['cumulative_sales'],
```

```
                         mode='lines+markers',
                         name='Cumulative Sales',
                         line=dict(color='red')))

fig.update_layout(title='Monthly and Cumulative Sales',
                  xaxis_title='Month-Year',
                  yaxis_title='Sales ($)',
                  xaxis_tickangle=-45)

fig.show()

comm_sales
```



Out[34]:

	year	month	revenue	cumulative_sales
0	2016	SEP	252.24	252.24
1	2016	OCT	59090.48	59342.72
2	2016	DEC	19.62	59362.34
3	2017	JAN	138488.04	138488.04
4	2017	FEB	291908.01	430396.05
5	2017	MAR	449863.60	880259.65
6	2017	APR	417788.03	1298047.68
7	2017	MAY	592918.82	1890966.50
8	2017	JUN	511276.38	2402242.88
9	2017	JUL	592382.92	2994625.80
10	2017	AUG	674396.32	3669022.12
11	2017	SEP	727762.45	4396784.57
12	2017	OCT	779677.88	5176462.45
13	2017	NOV	1194882.80	6371345.25
14	2017	DEC	878401.48	7249746.73
15	2018	JAN	1115004.18	1115004.18
16	2018	FEB	992463.34	2107467.52
17	2018	MAR	1159652.12	3267119.64
18	2018	APR	1160785.48	4427905.12
19	2018	MAY	1153982.15	5581887.27
20	2018	JUN	1023880.50	6605767.77
21	2018	JUL	1066540.75	7672308.52
22	2018	AUG	1022425.32	8694733.84
23	2018	SEP	4439.54	8699173.38
24	2018	OCT	589.67	8699763.05

In [35]:

```
# 3.    Calculate the year-over-year growth rate of total sales:
pd.read_sql_query("""with yoy_sales as (select year(order_purchase_timestamp) as ye
                   ifnull(round(lag(sum(payment_value)) over (order by  year(order_p
                   from orders o
                   join payments p on p.order_id=o.order_id
                   group by 1)
```

```
                    select year,ifnull(round(((total_sales-privious_year_sales)/privi
                    """,conn)
```

Out[35]:

	year	yoy_growth
0	2016	0.0
1	2017	12112.7
2	2018	20.0

In [36]:

```
# 4.Calculate the retention rate of customers, defined as the percentage of custome
pd.read_sql_query("""WITH first_orders AS (
                SELECT c.customer_id, MIN(o.order_purchase_timestamp) AS first_orde
                FROM customers c
                JOIN orders o  ON o.customer_id = c.customer_id
                GROUP BY c.customer_id),
            next_orders AS (SELECT a.customer_id, COUNT(DISTINCT o.order_purchase_t
                FROM first_orders a
                JOIN orders o ON o.customer_id = a.customer_id
                    AND o.order_purchase_timestamp > a.first_order
                    AND o.order_purchase_timestamp < DATE_ADD(a.first_order, INTERV
                GROUP BY a.customer_id)
            SELECT COUNT(DISTINCT next_orders.customer_id) / COUNT(DISTINCT first_o
            FROM first_orders
            LEFT JOIN next_orders
                ON first_orders.customer_id = next_orders.customer_id """
                  ,conn)

# The retention rate is zero because all customer IDs are unique in this data.
```

Out[36]:

	retention_rate
0	0.0

In [37]:

```
# 5.    Identify the top 3 customers who spent the most money in each year:
pd.read_sql_query("""select year,customer_id,payment from (
                    select year(order_purchase_timestamp) as year,
                    o.order_id,o.customer_id,round(sum(payment_value),2) as payment
                    dense_rank() over (partition by year(order_purchase_timestamp)
                    from orders o
                    join payments p on p.order_id = o.order_id
                    group by 1,2,3) a
                    where c_rank <= 3;"""
                  ,conn)
```

Out[37]:

	year	customer_id	payment
0	2016	a9dc96b027d1252bbac0a9b72d837fc6	1423.55
1	2016	1d34ed25963d5aae4cf3d7f3a4cda173	1400.74
2	2016	4a06381959b6670756de02e07b83815f	1227.78
3	2017	1617b1357756262bfa56ab541c47bc16	13664.08
4	2017	c6e2731c5b391845f6800c97401a43a9	6929.31
5	2017	3fd6777bbce08a352fddd04e4a7cc8f6	6726.66
6	2018	ec5b2ba62e574342386871631fafd3fc	7274.88
7	2018	f48d464a0baaea338cb25f816991ab1f	6922.21
8	2018	e0a2412720e9ea4f26c1ac985f6a7358	4809.44

In [38]:

```
# 6.    Determine the customer lifetime value (CLTV) for each customer:
pd.read_sql_query("""select c.customer_id , round(sum(payment_value),2) as CLTV
                  from customers c
                  join orders o on o.customer_id = c.customer_id
                  join payments p on p.order_id = o.order_id
                  group by c.customer_id
                  order by CLTV Desc
                  """,conn)
```

Out[38]:

	customer_id	CLTV
0	1617b1357756262bfa56ab541c47bc16	13664.08
1	ec5b2ba62e574342386871631fafd3fc	7274.88
2	c6e2731c5b391845f6800c97401a43a9	6929.31
3	f48d464a0baaea338cb25f816991ab1f	6922.21
4	3fd6777bbce08a352fddd04e4a7cc8f6	6726.66
...	...	...
99435	184e8e8e48937145eb96c721ef1f0747	10.07
99436	a790343ca6f3fee08112d678b43aa7c5	9.59
99437	a73c1f73f5772cf801434bf984b0b1a7	0.00
99438	3532ba38a3fd242259a514ac2b6ae6b6	0.00
99439	197a2a6a77da93f678ea0d379f21da0a	0.00

99440 rows × 2 columns

In [39]:

```
# 7.    Calculate the average order value (AOV) growth rate over time:
pd.read_sql_query("""with yoy_aov as (select year(order_purchase_timestamp) as year
                  ifnull(round(lag(avg(payment_value)) over (order by  year(order_p
```

```
                 from orders o
                 join payments p on p.order_id=o.order_id
                 group by 1)

                 select year,ifnull(round(((avg_sales-privious_year_sales)/priviou
                 """,conn)
```

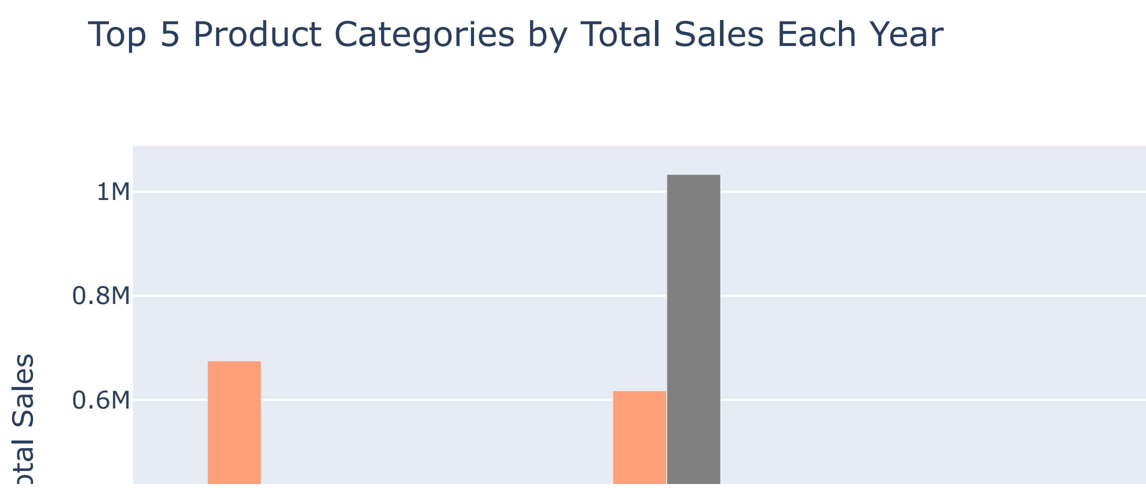
Out[39]:

	year	aov_growth_rate
0	2016	0.00
1	2017	-11.09
2	2018	1.81

In [40]:

```
# 8.    Identify the top 5 products contributing to the highest sales margin each y

yoy_high_sales=pd.read_sql_query("""with sales_rank as (SELECT
                product_category,
                YEAR(order_purchase_timestamp) AS year,
                SUM(payment_value) AS total_sales,
                RANK() OVER (PARTITION BY YEAR(order_purchase_timestamp) ORDER BY S
            FROM products p
                  join order_items ot on ot.product_id = p.product_id
                  join orders o on o.order_id = ot.order_id
                  join payments pm on pm.order_id = o.order_id
            GROUP BY product_category, YEAR(order_purchase_timestamp))

                  select product_category, year,round(total_sales,2) as total_sales
                  where sales_rank <=5;
                  """,conn)



color_map = {
    2016: 'indianred',
    2017: 'lightsalmon',
    2028: 'seagreen'}

# Create the figure
fig = go.Figure()

# Add trace for each year with distinct colors
for year in yoy_high_sales['year'].unique():
    data = yoy_high_sales[yoy_high_sales['year'] == year]
    fig.add_trace(go.Bar(
        x=data['product_category'],
        y=data['total_sales'],
        name=f'Sales in {year}',
        marker_color=color_map.get(year, 'grey')  # Default to 'grey' if year not i
    ))

# Update layout for grouped bars and x-axis label rotation
fig.update_layout(
    barmode='group',
    xaxis_tickangle=-45,
```

```
    title_text='Top 5 Product Categories by Total Sales Each Year',
    xaxis_title='Product Category',
    yaxis_title='Total Sales',
    legend_title='Year'
)

# Show the plot
fig.show()


# print dataframe
yoy_high_sales
```



Out[40]:

	product_category	year	total_sales
0	Furniture Decoration	2016	11125.00
1	perfumery	2016	7201.84
2	HEALTH BEAUTY	2016	6062.16
3	toys	2016	5670.74
4	Market Place	2016	4955.87
5	bed table bath	2017	797314.22
6	computer accessories	2017	704696.93
7	Furniture Decoration	2017	675005.79
8	sport leisure	2017	642014.31
9	HEALTH BEAUTY	2017	617706.87
10	HEALTH BEAUTY	2018	1033604.09
11	bed table bath	2018	912947.74
12	computer accessories	2018	878236.82
13	Watches present	2018	854349.04
14	sport leisure	2018	746186.14

The Health & Beauty category saw the most significant increase in sales between 2017 and 2018.

Furniture Decoration saw the highest total sales in 2017, but sales have been declining since.

Conclusion

This e-commerce analysis project provided valuable insights into customer behavior, sales trends, and product performance. By leveraging Python and SQL, we were able to efficiently process large datasets, perform complex queries, and visualize key metrics.

Thank you!