PYTHON & SQL PROJECT

E-Commerce (Target) Sales Dataset ANALYSIS

By:

Name: Sagar Gupta

E-mail: krishanasagar406@gmail.com

Liked In: https://www.linkedin.com/in/sagar-gupta087/

Qualification: BCA Undergraduate (2021 -2024)

GitHub Link: https://github.com/Sagar-Gupta008/Python-and-SQL-Project/tree/main

AIM

1. Data Collection and Cleaning:

- Gather the Target E-Commerce sales dataset.
- Clean and preprocess the data to handle missing values, inconsistencies, and outliers.

2. Exploratory Data Analysis (EDA):

- Conduct descriptive statistics to summarize the dataset.
- Visualize key metrics using graphs and charts (e.g., sales trends over time, product categories performance, geographical sales distribution).
- Identify patterns and correlations within the data.

3. Sales Performance Analysis:

- Analyze sales performance across different product categories, regions, and time periods.
- Identify best-selling products and categories.

4. Customer Analysis:

- Segment customers based on their purchasing behavior and analyze Customer Retention Rates.

5. SQL Integration:

- Use SQL queries to extract and manipulate data from the database.
- Perform complex joins, aggregations, and subqueries to derive insights.
- Store and retrieve analysis results efficiently.

6. Reporting and Visualization:

- Create interactive visualizations using Python libraries (e.g., Matplotlib, Seaborn).
- Summarize key findings and present actionable insights.

TARGET

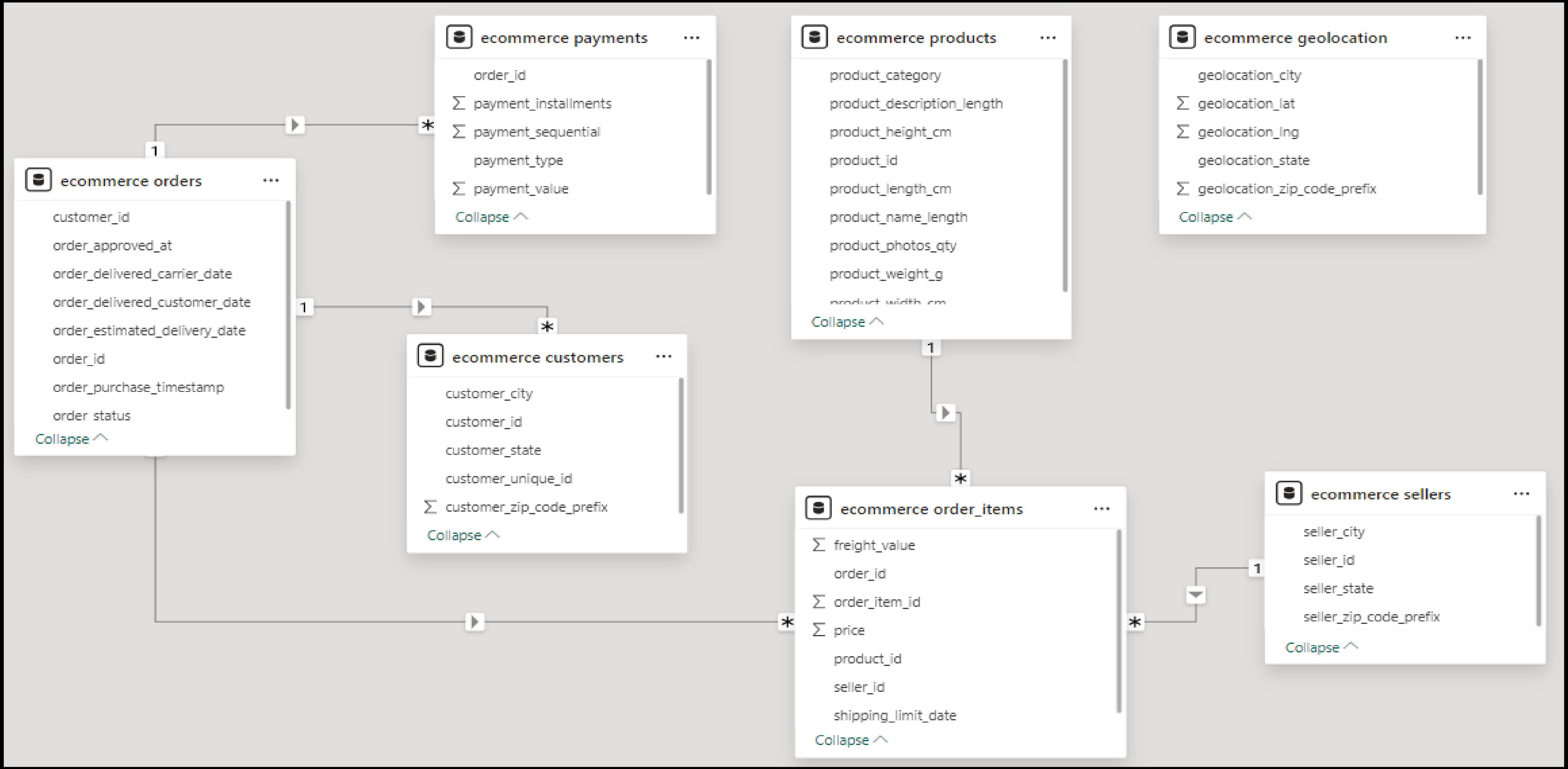
WHAT IS SQL?

- SQL (Structured Query Language) is a standardized programming language used for managing and manipulating relational databases.
- It is designed for querying, updating, and managing data stored in relational database management systems (RDBMS).

WHAT IS PYTHON

- Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It was created by Guido van Rossum and first released in 1991.
- Python emphasizes code readability with its clear and concise syntax, which allows developers to write less code to accomplish tasks compared to many other programming languages.

DATASET SCHEMA



Loading the Dataset to Python in the form of Data Frames


```python
import pandas as pd
import mysql.connector
import os

# List of CSV files and their corresponding table names
csv_files = [
    ('customers.csv', 'customers'),
    ('orders.csv', 'orders'),
    ('sellers.csv', 'sellers'),
    ('products.csv', 'products'),
    ('geolocation.csv', 'geolocation'),
    ('payments.csv', 'payments'),
    ('order_items.csv', 'order_items')# Added payments.csv for specific handling
]
```

```python
# Connect to the MySQL database
conn = mysql.connector.connect(
    host='localhost',
    user='root',
    password='27104720A',
    database='ecommerce'
)
cursor = conn.cursor()
```

```python
# Folder containing the CSV files
folder_path = 'D:\Python and Sql Project'

def get_sql_type(dtype):
    if pd.api.types.is_integer_dtype(dtype):
        return 'INT'
    elif pd.api.types.is_float_dtype(dtype):
        return 'FLOAT'
    elif pd.api.types.is_bool_dtype(dtype):
        return 'BOOLEAN'
    elif pd.api.types.is_datetime64_any_dtype(dtype):
        return 'DATETIME'
    else:
        return 'TEXT'

for csv_file, table_name in csv_files:
    file_path = os.path.join(folder_path, csv_file)

    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv(file_path)

    # Replace NaN with None to handle SQL NULL
    df = df.where(pd.notnull(df), None)
```

```python
# Debugging: Check for NaN values
print(f"Processing {csv_file}")
print(f"NaN values before replacement:\n{df.isnull().sum()}\n")

# Clean column names
df.columns = [col.replace(' ', '_').replace('-', '_').replace('.', '_') for col in df.columns]

# Generate the CREATE TABLE statement with appropriate data types
columns = ', '.join([f'`{col}` {get_sql_type(df[col].dtype)}' for col in df.columns])
create_table_query = f'CREATE TABLE IF NOT EXISTS `{table_name}` ({columns})'
cursor.execute(create_table_query)
```

```python
    # Insert DataFrame data into the MySQL table
    for _, row in df.iterrows():
        # Convert row to tuple and handle NaN/None explicitly
        values = tuple(None if pd.isna(x) else x for x in row)
        sql = f"INSERT INTO `{table_name}` ({', '.join(['`' + col + '`' for col in df.columns])}) VALUES ({', '.join(['%s'] * len(row))})"
        cursor.execute(sql, values)

    # Commit the transaction for the current CSV file
    conn.commit()

# Close the connection
conn.close()
```

Output


```
Processing customers.csv
NaN values before replacement:
customer_id                 0
customer_unique_id          0
customer_zip_code_prefix    0
customer_city               0
customer_state              0
dtype: int64

Processing orders.csv
NaN values before replacement:
order_id                            0
customer_id                         0
order_status                        0
order_purchase_timestamp            0
order_approved_at                 160
order_delivered_carrier_date     1783
order_delivered_customer_date    2965
order_estimated_delivery_date       0
dtype: int64

Processing sellers.csv
NaN values before replacement:
seller_id                 0
seller_zip_code_prefix    0
seller_city               0
seller_state              0
dtype: int64

Processing products.csv
NaN values before replacement:
product_id                    0
product category            610
product_name_length         610
product_description_length  610
product_photos_qty          610
product_weight_g              2
product_length_cm             2
product_height_cm             2
product_width_cm              2
dtype: int64
```

```
Processing geolocation.csv
NaN values before replacement:
geolocation_zip_code_prefix    0
geolocation_lat                0
geolocation_lng                0
geolocation_city               0
geolocation_state              0
dtype: int64

Processing payments.csv
NaN values before replacement:
order_id                0
payment_sequential      0
payment_type            0
payment_installments    0
payment_value           0
dtype: int64

Processing order_items.csv
NaN values before replacement:
order_id               0
order_item_id          0
product_id             0
seller_id              0
shipping_limit_date    0
price                  0
freight_value          0
dtype: int64
```

Establishing Connection Between Python and SQL


```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import mysql.connector

db=mysql.connector.connect(host="localhost",
                           username="root",
                           password="27104720A",
                           database="ecommerce")


cur=db.cursor()
```

BASIC QUESTIONS

Q1 List all unique cities where customers are located.

```
query="""select distinct customer_city from customers"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data)
df.head(10)
```

	0
0	franca
1	sao bernardo do campo
2	sao paulo
3	mogi das cruzes
4	campinas
5	jaragua do sul
6	timoteo
7	curitiba
8	belo horizonte
9	montes claros

Q2 Count the number of orders placed in 2017.

```
query="""select count(order_id) from orders where year(order_purchase_timestamp)=2017"""
cur.execute(query)
data=cur.fetchall()
print('The total number of orders placed in 2017 are:',data[0][0])
```

```
The total number of orders placed in 2017 are: 45101
```

Q3 Find the total sales per category.


```
query="""select upper(products.product_category) as category,round(sum(payments.payment_value),2) as sales
from products join order_items
on products.product_id=order_items.product_id
join payments
on payments.order_id=order_items.order_id
group by category"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data,columns=['Category','Sales'])
df
```

	Category	Sales
0	PERFUMERY	506738.66
1	FURNITURE DECORATION	1430176.39
2	TELEPHONY	486882.05
3	BED TABLE BATH	1712553.67
4	AUTOMOTIVE	852294.33
...	...	...
69	CDS MUSIC DVDS	1199.43
70	LA CUISINE	2913.53
71	FASHION CHILDREN'S CLOTHING	785.67
72	PC GAMER	2174.43
73	INSURANCE AND SERVICES	324.51

74 rows × 2 columns

Q4 Calculate the percentage of orders that were paid in installments.

```
query="""select round((sum(case when payment_installments>=1 then 1
else 0 end)) / count(*)*100,3) from payments;"""

cur.execute(query)
data=cur.fetchall()
print("The percentage of orders that were paid in installments is:",data[0][0],'%')
```

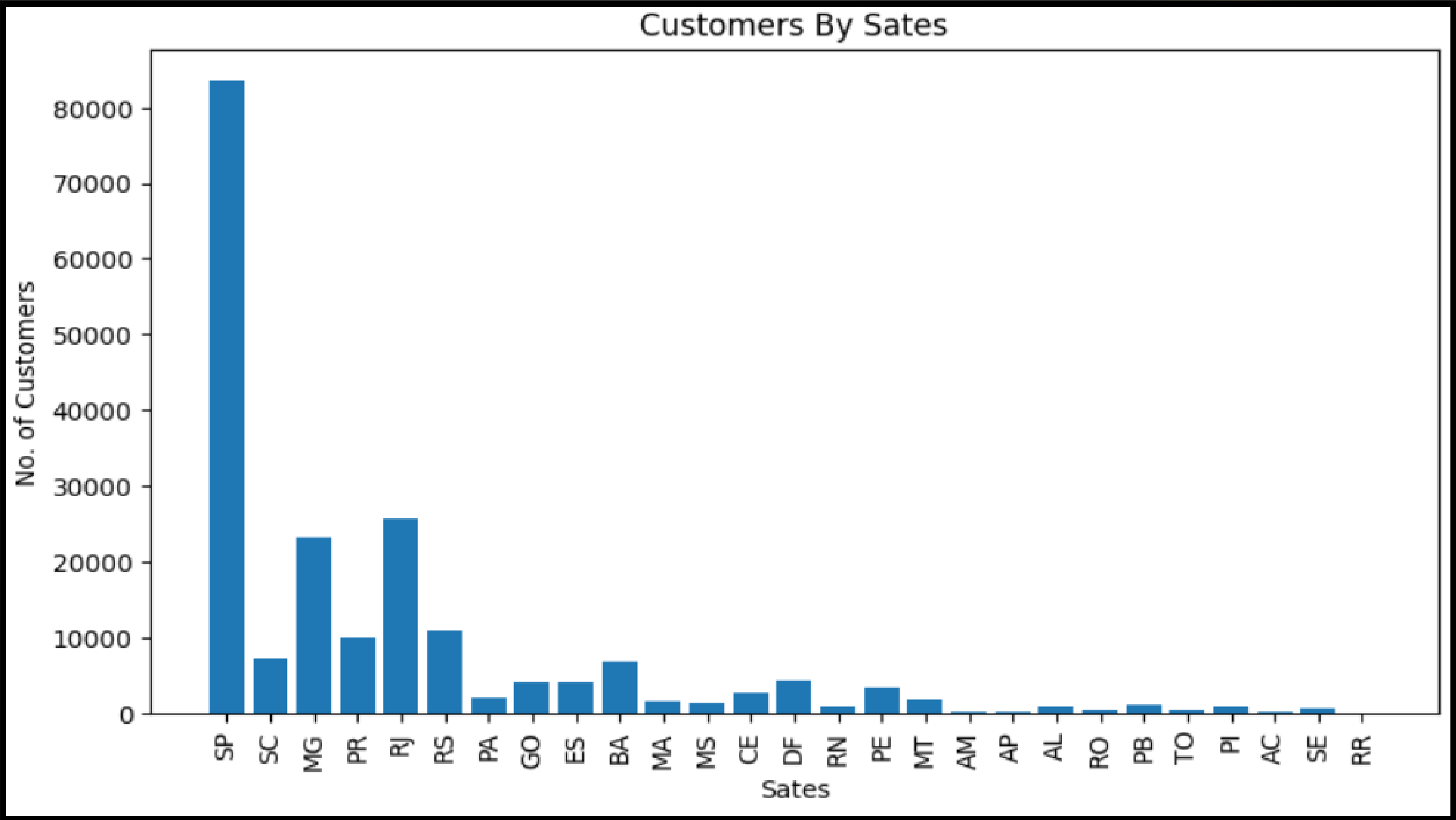
```
The percentage of orders that were paid in installments is: 99.998 %
```

Q5 Count the number of customers from each state.

```
query="""select customer_state,count(customer_id)
from customers group by customer_state"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data,columns=['State','No. of customers'])

#plotting the data
plt.figure(figsize=(9,5))
x=df['State']
y=df['No. of customers']
plt.bar(x,y)
plt.xticks(rotation='vertical')
plt.xlabel('Sates')
plt.ylabel('No. of Customers')
plt.title('Customers By Sates')
plt.show()
```


Customers By Sates
No. of Customers
80000
70000
60000
50000
40000
30000
20000
10000
0
SP
SC
MG
PR
RJ
RS
PA
GO
ES
BA
MA
MS
CE
DF
RN
PE
MT
AM
AP
AL
RO
PB
TO
PI
AC
SE
RR
Sates

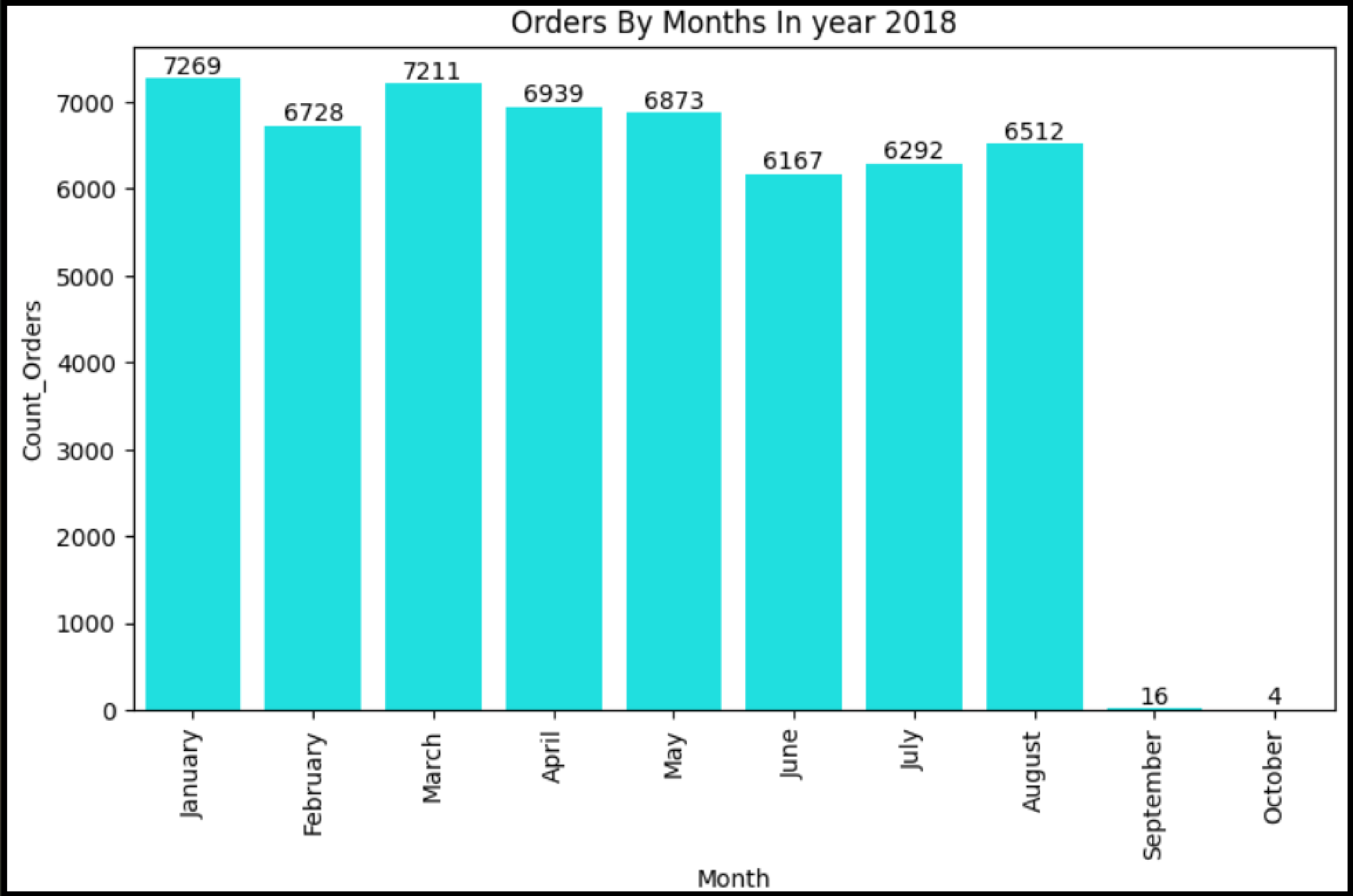
INTERMEDIATE QUESTIONS

Q1 Calculate the number of orders per month in 2018.

```
query="""select monthname(order_purchase_timestamp),count(order_id)
from orders where year(order_purchase_timestamp)=2018
group by monthname(order_purchase_timestamp)"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data,columns=['Month','Count_Orders'])
df

#plotting the data
plt.figure(figsize=(9,5))
o=["January", "February", "March", "April", "May", "June", "July", "August", "September", "October"]
ax=sns.barplot(x=df['Month'],y=df['Count_Orders'],data=df,order=o,color='cyan')
ax.bar_label(ax.containers[0])
plt.xticks(rotation='vertical')
plt.title('Orders By Months In year 2018')
plt.show()
```


Orders By Months In year 2018
Count_Orders
Month
7269
6728
7211
6939
6873
6167
6292
6512
16
4
January
February
March
April
May
June
July
August
September
October
0
1000
2000
3000
4000
5000
6000
7000

Q2 Find the average number of products per order, grouped by customer city.

```
query="""with count_per_order as
(select orders.order_id,orders.customer_id,
count(order_items.order_id) as OC
from orders join order_items
on orders.order_id=order_items.order_id
group by orders.order_id,orders.customer_id)
select customers.customer_city,
round(avg(count_per_order.OC),2) as Avg_Orders from
customers join count_per_order
on customers.customer_id=count_per_order.customer_id
group by customers.customer_city order by Avg_Orders desc"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data,columns=['Customer_city','Avg_Orders'])
df.head(10)
```

	Customer_city	Avg_Orders
0	padre carvalho	7.00
1	celso ramos	6.50
2	datas	6.00
3	candido godoi	6.00
4	matias olimpio	5.00
5	cidelandia	4.00
6	curralinho	4.00
7	picarra	4.00
8	morro de sao paulo	4.00
9	teixeira soares	4.00

Q3 Calculate the percentage of total revenue contributed by each product category.

```
query="""select upper(products.product_category) as category,
round(sum(payments.payment_value)/(select sum(payment_value) from payments)*100,2)
as Percentage_sales
from products join order_items
on products.product_id=order_items.product_id
join payments
on payments.order_id=order_items.order_id
group by category
order by Percentage_sales desc;"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data,columns=['Category','Percentage_Sales'])
df.head()
```

	Category	Percentage_Sales
0	BED TABLE BATH	10.70
1	HEALTH BEAUTY	10.35
2	COMPUTER ACCESSORIES	9.90
3	FURNITURE DECORATION	8.93
4	WATCHES PRESENT	8.93

Q4 Identify the correlation between product price and the number of times a product has been purchased.

```
import numpy as np
query="""SELECT products.product_category,
count(order_items.product_id) as Count_orders,
round(avg(order_items.price),2) as Avg_Price
from products join order_items
on products.product_id=order_items.product_id
group by products.product_category;"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data,columns=['Category','Count_orders','Avg_Price'])

#calculating the correlation between Count_orders and Avg_Price
arr1=df['Count_orders']
arr2=df['Avg_Price']
corr=np.corrcoef([arr1,arr2])
print('The correlation between product price and \nthe number of times a product has been purchased is:\n',corr[0][1])
```

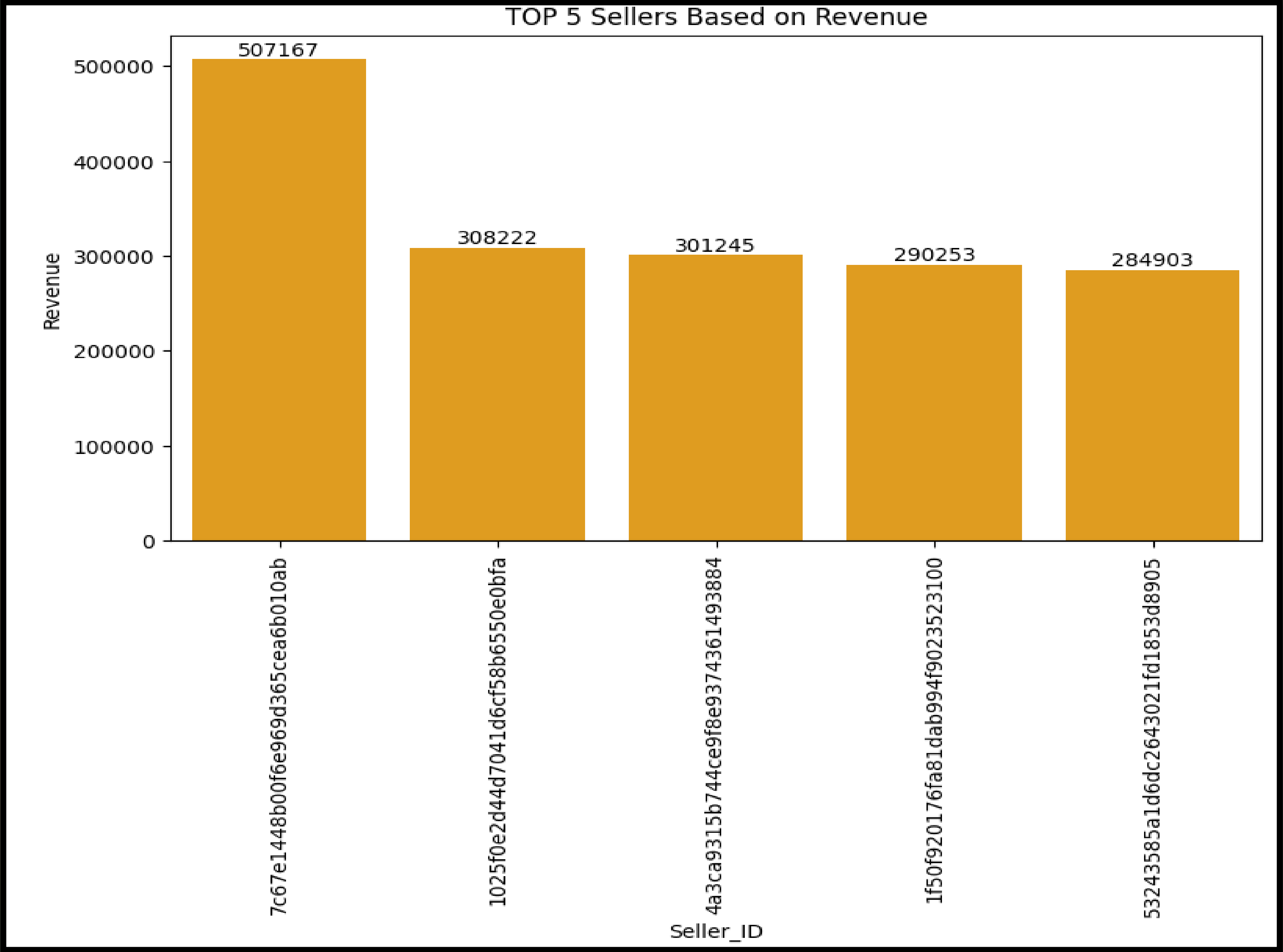
```
The correlation between product price and
the number of times a product has been purchased is:
 -0.10631514167157562
```

Q5 Calculate the total revenue generated by each seller, and rank them by revenue.

```
query="""select *, dense_rank() over(order by Revenue desc) as rnk from
(select order_items.seller_id,
round(sum(payments.payment_value),2) as Revenue from
order_items join payments
on payments.order_id=order_items.order_id
group by order_items.seller_id) as A;"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data,columns=['Seller_ID','Revenue','Rank'])
df=df.head()

#plotting the data
plt.figure(figsize=(9,5))
ax=sns.barplot(x=df['Seller_ID'],y=df['Revenue'],data=df,color='orange')
ax.bar_label(ax.containers[0])
plt.xticks(rotation='vertical')
plt.title('TOP 5 Sellers Based on Revenue')
plt.show()
```


TOP 5 Sellers Based on Revenue
507167
308222
301245
290253
284903
500000
400000
300000
200000
100000
0
Revenue
7c67e1448b00f6e969d365cea6b010ab
1025f0e2d44d7041d6cf58b6550e0bfa
4a3ca9315b744ce9f8e9374361493884
1f50f920176fa81dab994f9023523100
53243585a1d6dc2643021fd1853d8905
Seller_ID

ADVANCED QUESTIONS

Q1 Calculate the moving average of order values for each customer over their order history.

```
query="""select customer_id,order_purchase_timestamp,payment,
avg(payment) over(partition by customer_id order by order_purchase_timestamp
rows between 2 preceding and current row) as Moving_Avg
from
(select orders.customer_id, orders.order_purchase_timestamp,
payments.payment_value as payment from
payments join orders
on payments.order_id=orders.order_id) as A;"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data,columns=['Customer_id','Order_purchase_timestamp','Payments','Moving_Avg'])
df
```

	Customer_id	Order_purchase_timestamp	Payments	Moving_Avg
0	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.74	114.739998
1	000161a058600d5901f007fab4c27140	2017-07-16 09:40:32	67.41	67.410004
2	0001fd6190edaaf884bcaf3d49edf079	2017-02-28 11:06:43	195.42	195.419998
3	0002414f95344307404f0ace7a26f1d5	2017-08-16 13:09:20	179.35	179.350006
4	000379cdec625522490c315e70c7a9fb	2018-04-02 13:42:17	107.01	107.010002
...	...	...	...	...
103881	fffecc9f79fd8c764f843e9951b11341	2018-03-29 16:59:26	71.23	27.120001
103882	fffeda5b6d849fbd39689bb92087f431	2018-05-22 13:36:02	63.13	63.130001
103883	ffff42319e9b2d713724ae527742af25	2018-06-13 16:57:05	214.13	214.130005
103884	ffffa3172527f765de70084a7e53aae8	2017-09-02 11:53:32	45.50	45.500000
103885	ffffe8b65bbe3087b653a978c870db99	2017-09-29 14:07:03	18.37	18.370001

103886 rows × 4 columns

Q2 Calculate the cumulative sales per month for each year.

```
query="""select *, round(sum(Sales) over(order by Years,Months),2)
as Cumulative_Sales from
(select year(orders.order_purchase_timestamp) as Years,
month(orders.order_purchase_timestamp) as Months,
round(sum(payments.payment_value),2) as Sales from
orders join payments on
orders.order_id=payments.order_id
group by Years,Months
order by Years,Months) as a;"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data,columns=['Years','Months','Sales','Cumulative Sales'])
df
```

	Years	Months	Sales	Cumulative Sales
0	2016	9	252.24	252.24
1	2016	10	59090.48	59342.72
2	2016	12	19.62	59362.34
3	2017	1	138488.04	197850.38
4	2017	2	291908.01	489758.39
5	2017	3	449863.60	939621.99
6	2017	4	417788.03	1357410.02
7	2017	5	592918.82	1950328.84
8	2017	6	511276.38	2461605.22
9	2017	7	592382.92	3053988.14
10	2017	8	674396.32	3728384.46

Q3 Calculate the year-over-year growth rate of total sales.

```
query="""with b as (select *,lag(Current_Year_Sales,1) over(order by years) as
Previous_Year_Sales from
(select year(orders.order_purchase_timestamp) as Years,
round(sum(payments.payment_value),2) as Current_Year_Sales from
orders join payments on
orders.order_id=payments.order_id
group by Years
order by Years) as a)
select Years,
round(((Current_Year_Sales-Previous_Year_Sales)/Previous_Year_Sales)*100,2)
as Year_over_Year_Growth from b;"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data,columns=['Years','Year_over_Year_Growth'])
df
```

	Years	Year_over_Year_Growth
0	2016	NaN
1	2017	12112.7
2	2018	20.0

Q4 Calculate the retention rate of customers, defined as the percentage of customers who make another purchase within 6 months of their first purchase


```
query="""with a as(select customers.customer_id,
min(orders.order_purchase_timestamp) as first_order from
customers join orders on
customers.customer_id=orders.customer_id
group by customers.customer_id),

b as(select a.customer_id,
count(distinct orders.order_purchase_timestamp) as next_order
from a join orders on
a.customer_id=orders.customer_id
and first_order<orders.order_purchase_timestamp
and orders.order_purchase_timestamp<date_add(first_order, interval 6 month)
group by a.customer_id)

select 100*(count(distinct a.customer_id)/count(distinct b.customer_id))
as Customer_Retention_Rate
from a left join b on
a.customer_id=b.customer_id;"""

cur.execute(query)
data=cur.fetchall()
print('The retention rate of customers is:',data[0])
```

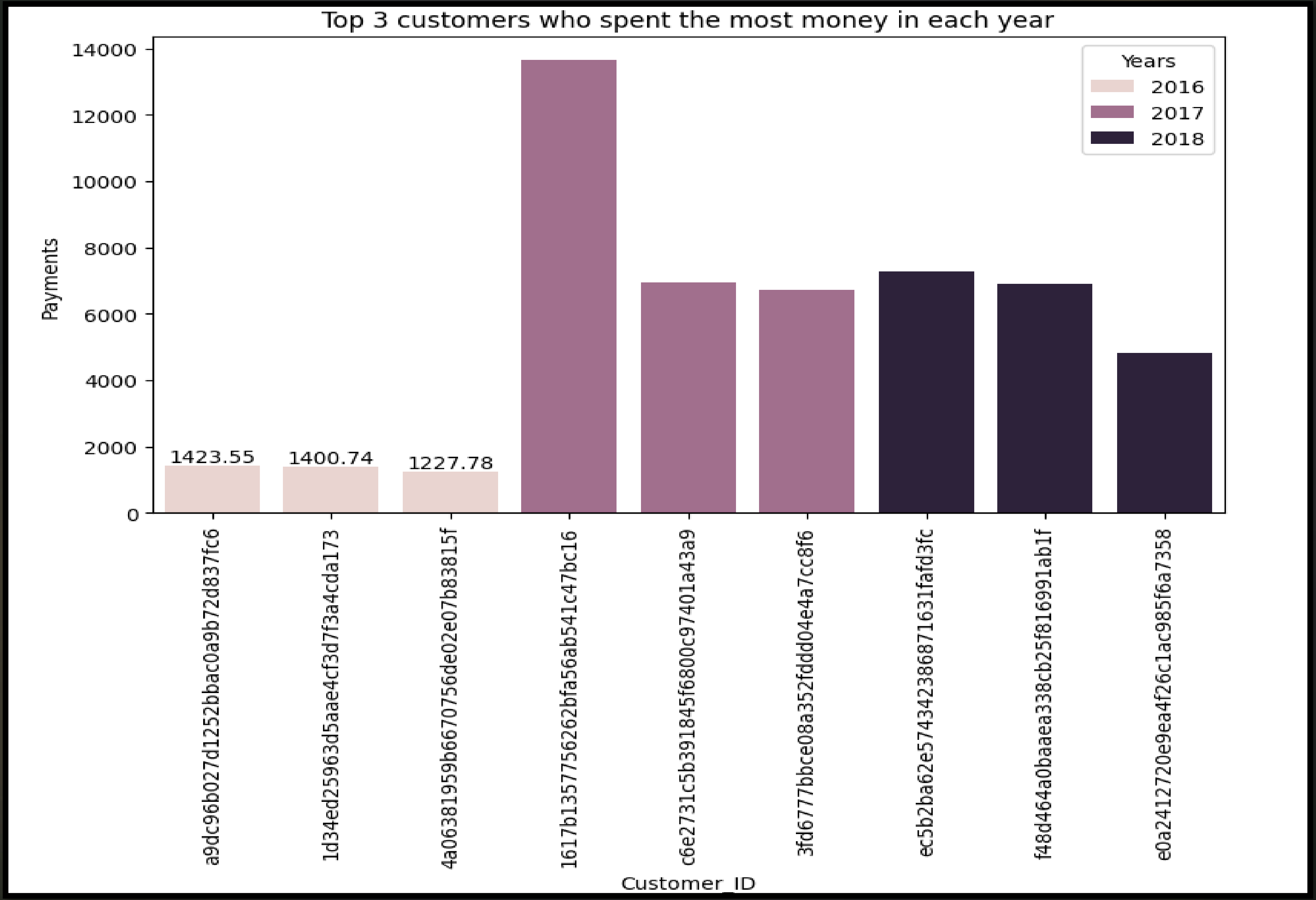
```
The retention rate of customers is: (None,)
```

Q5 Identify the top 3 customers who spent the most money in each year.


```
query="""select Years,customer_id,Payment,rnk
from
(select year(orders.order_purchase_timestamp) as Years,
orders.order_id,
orders.customer_id,
round(sum(payments.payment_value),2) as Payment,
dense_rank() over(partition by year(orders.order_purchase_timestamp)
order by sum(payments.payment_value) desc)
as rnk from orders join payments on
orders.order_id=payments.order_id
group by year(orders.order_purchase_timestamp),
orders.customer_id,orders.order_id) as a
where rnk<=3;"""

cur.execute(query)
data=cur.fetchall()
df=pd.DataFrame(data,columns=['Years','Customer_ID','Payments','Rank'])

#plotting the data
plt.figure(figsize=(9,5))
ax=sns.barplot(x=df['Customer_ID'],y=df['Payments'],data=df,hue=df['Years'])
ax.bar_label(ax.containers[0])
plt.xticks(rotation='vertical')
plt.title('Top 3 customers who spent the most money in each year')
plt.show()
```


Top 3 customers who spent the most money in each year
Payments
14000
12000
10000
8000
6000
4000
2000
0
Years
2016
2017
2018
1423.55
1400.74
1227.78
a9dc96b027d1252bbac0a9b72d837fc6
1d34ed25963d5aae4cf3d7f3a4cda173
4a06381959b6670756de02e07b83815f
1617b1357756262bfa56ab541c47bc16
c6e2731c5b391845f6800c97401a43a9
3fd6777bbce08a352fddd04e4a7cc8f6
ec5b2ba62e574342386871631fafd3fc
f48d464a0baaea338cb25f816991ab1f
e0a2412720e9ea4f26c1ac985f6a7358
Customer_ID

FUTURE SCOPE OF PROJECT

1. Advanced Predictive Modeling:

- Develop more sophisticated machine learning models to predict future sales, customer behavior, and inventory needs.

2. Real-Time Data Analysis:

- Implement real-time data processing and analysis pipelines to monitor sales and customer behavior in real-time.

3. Integration with Business Intelligence Tools:

- Integrate the analysis with BI tools such as Tableau or Power BI for more interactive and dynamic visualizations.

4. Enhanced Customer Segmentation:

- Use clustering algorithms (e.g., K-means, DBSCAN) and advanced segmentation techniques to create more granular customer segments.

5. Recommendation Systems:

- Develop personalized recommendation systems using collaborative filtering, content-based filtering, or hybrid methods.

6. Sentiment Analysis and Text Mining:

- Analyze customer reviews, feedback, and social media mentions to understand customer sentiment and preferences.


Thank you!