BY: -

AMEET RAJ

I have prepared 80 SQL queries for analyzing an e-commerce database, covering various aspects such as sales performance, customer behaviour, product popularity, and payment methods. Additionally, I have created visualizations in Jupiter Notebook for 3 to 4 key questions to illustrate insights such as monthly sales trends and top-selling products. These visualizations enhance the understanding of the data and support data-driven decision-making.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import mysql.connector
import os
```

In [2]:
```
# Connect to the MySQL database
db = mysql.connector.connect(
    host='localhost',
    user='root',
    password='12345',
    database='ecommerce'
)

# Create a cursor object
cur = db.cursor()
```

1.List all unique cities where customers are located.

In [4]:
```
query = """SELECT DISTINCT customer_city FROM customers"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data,columns=["customer_city"])
df
```

Out[4]:

	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

2.Find the total number of customers.

In [6]:
```
query = """SELECT COUNT(*) AS total_customers FROM customers;
"""
cur.execute(query)
data = cur.fetchall()
data
"total number of customers are",data[0][0]
```

Out[6]:
```
('total number of customers are', 397764)
```

3.Count the number of customers in each state.

In [8]:
```
query = """
SELECT
     customer_state, COUNT(*) AS customer_count
FROM customers
GROUP BY
customer_state"""

cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["state","no of customer"])
df=df.sort_values(by="no of customer", ascending=False)
df
```

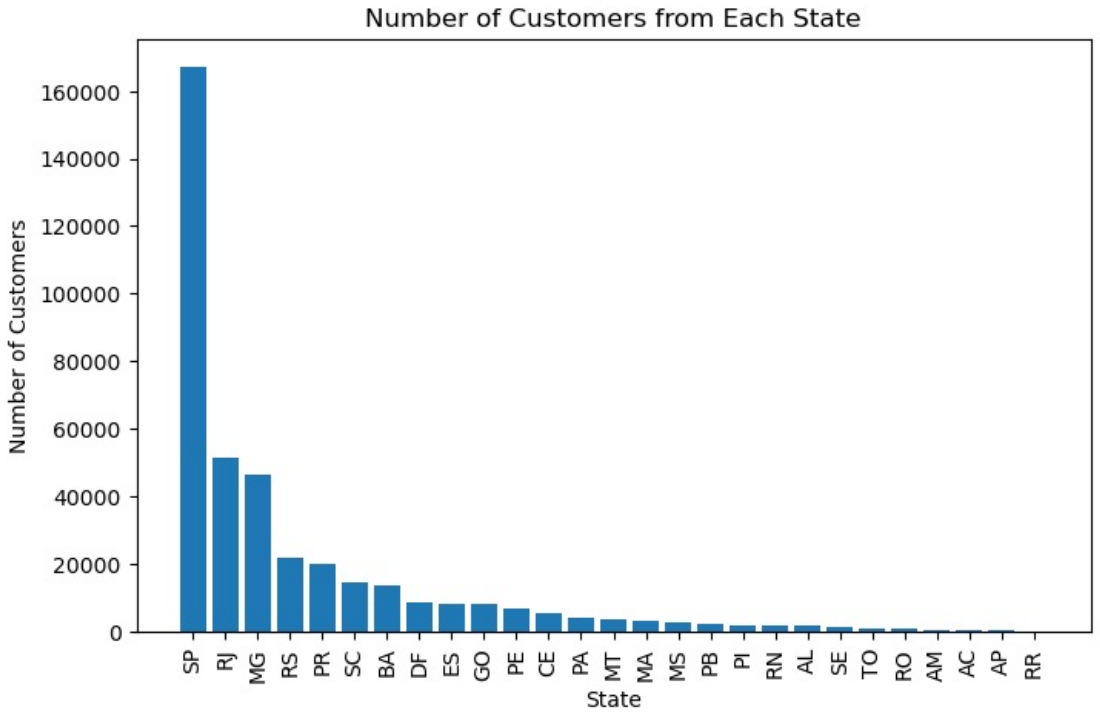
Out[8]:

	state	no of customer
0	SP	166984
4	RJ	51408
2	MG	46540
5	RS	21864
3	PR	20180
1	SC	14548
9	BA	13520
13	DF	8560
8	ES	8132
7	GO	8080
15	PE	6608
12	CE	5344
6	PA	3900
16	MT	3628
10	MA	2988
11	MS	2860
21	PB	2144
23	PI	1980
14	RN	1940
19	AL	1652
25	SE	1400
22	TO	1120
20	RO	1012
17	AM	592
24	AC	324
18	AP	272
26	RR	184

Ameet raj

In [9]:

```
df = pd.DataFrame(data, columns=["state", "customer_count"])
df = df.sort_values(by="customer_count", ascending=False)
plt.figure(figsize=(8, 5))
plt.bar(df["state"], df["customer_count"])
plt.xticks(rotation=90)
plt.xlabel('State')
plt.ylabel('Number of Customers')
plt.title('Number of Customers from Each State')
plt.show()
```



4.Predict the details of customers from a specific state of 'SP'.

In [11]:
```
query = """SELECT * FROM customers WHERE customer_state = 'SP';
"""
cur.execute(query)
data = cur.fetchall()
data
columns =["customer_id","unique_id","zip code","city","state"]
df= pd.DataFrame(data,columns=columns)
df
```

Out[11]:

	customer_id	unique_id	zip code	city	state
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	14409	franca	SP
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	9790	sao bernardo do campo	SP
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	1151	sao paulo	SP
3	b2b6027bc5c5109e529d4dc6358b12c3	259dac757896d24d7702b9acbbff3f3c	8775	mogi das cruzes	SP
4	4f2d8ab171c80ec8364f7c12e35b23ad	345ecd01c38d18a9036ed96c73b8d066	13056	campinas	SP
...	...	...	...	...	...
166979	f255d679c7c86c24ef4861320d5b7675	d111b06b6f3a2add0d2241325f65b5ca	13500	rio claro	SP
166980	f5a0b560f9e9427792a88bec97710212	b3e53d18a997f27a3ffd16da497eaf58	7790	cajamar	SP
166981	17ddf5dd5d51696bb3d7c6291687be6f	1a29b476fee25c95fbafc67c5ac95cf8	3937	sao paulo	SP
166982	e7b71a9017aa05c9a7fd292d714858e8	d52a67c98be1cf6a5c84435bd38d095d	6764	taboao da serra	SP
166983	274fa6071e5e17fe303b9748641082c8	84732c5050c01db9b23e19ba39899398	6703	cotia	SP

166984 rows × 5 columns

5.Retrieve the customer details with a specific zip code prefix "18112".

In [13]:
```
query = """SELECT * FROM customers WHERE customer_zip_code_prefix = 18112
"""
cur.execute(query)
data = cur.fetchall()
columns =["customer_id","unique_id","zip code","city","state"]
data
df= pd.DataFrame(data,columns=columns)
df
```

Out[13]:

	customer_id	unique_id	zip code	city	state
0	00a39528c677a55852f57235f988b837	9f67d6145893e41e7ada2e646b84f658	18112	votorantim	SP
1	7c970ed3afc2cc695e6b4ca38954bef8	58e51c87c3ec742b03d5cee5e49dd664	18112	votorantim	SP
2	07d18ca5eb0bb3e96f406d74edb2462e	8ca4019dfb727a43ec2d7671290be758	18112	votorantim	SP
3	6f5d085fa90e41ad46b2f5623426febe	32efa507b1a360412dee7ff30905ebdc	18112	votorantim	SP
4	23157f439f941d147f552ac180d68323	9ac370ba1e5aa108afc187fd57ce9442	18112	votorantim	SP
5	474090b9b8e1541d8ff3f614fb17398e	ea6ad96f2d4ea562128886eee252fc3e	18112	votorantim	SP
6	9b38c5e19f3e7f498ab36c0b095f39a6	0dc08717c1e08839d7c105878743f55e	18112	votorantim	SP
7	25ebf9efb39e9dc48d6c51a9896af9a5	2b229b2693dc783259f684b24fcd23a7	18112	votorantim	SP
8	74f0cc1d27884a102efa142ba1f540b3	2099f93ac5ea679cf2eb1de091ab87c9	18112	votorantim	SP
9	59699572fb53609e196978771c89a9dd	fce3cbf990e49bda6b74fa9c68640b06	18112	votorantim	SP
10	347a5aa5d381e31e52b84bfcae3e1eac	51fb62110fde8d22e80023732c46ef38	18112	votorantim	SP
11	00a39528c677a55852f57235f988b837	9f67d6145893e41e7ada2e646b84f658	18112	votorantim	SP
12	7c970ed3afc2cc695e6b4ca38954bef8	58e51c87c3ec742b03d5cee5e49dd664	18112	votorantim	SP
13	07d18ca5eb0bb3e96f406d74edb2462e	8ca4019dfb727a43ec2d7671290be758	18112	votorantim	SP
14	6f5d085fa90e41ad46b2f5623426febe	32efa507b1a360412dee7ff30905ebdc	18112	votorantim	SP
15	23157f439f941d147f552ac180d68323	9ac370ba1e5aa108afc187fd57ce9442	18112	votorantim	SP
16	474090b9b8e1541d8ff3f614fb17398e	ea6ad96f2d4ea562128886eee252fc3e	18112	votorantim	SP
17	9b38c5e19f3e7f498ab36c0b095f39a6	0dc08717c1e08839d7c105878743f55e	18112	votorantim	SP
18	25ebf9efb39e9dc48d6c51a9896af9a5	2b229b2693dc783259f684b24fcd23a7	18112	votorantim	SP
19	74f0cc1d27884a102efa142ba1f540b3	2099f93ac5ea679cf2eb1de091ab87c9	18112	votorantim	SP
20	59699572fb53609e196978771c89a9dd	fce3cbf990e49bda6b74fa9c68640b06	18112	votorantim	SP
21	347a5aa5d381e31e52b84bfcae3e1eac	51fb62110fde8d22e80023732c46ef38	18112	votorantim	SP
22	00a39528c677a55852f57235f988b837	9f67d6145893e41e7ada2e646b84f658	18112	votorantim	SP
23	7c970ed3afc2cc695e6b4ca38954bef8	58e51c87c3ec742b03d5cee5e49dd664	18112	votorantim	SP
24	07d18ca5eb0bb3e96f406d74edb2462e	8ca4019dfb727a43ec2d7671290be758	18112	votorantim	SP
25	6f5d085fa90e41ad46b2f5623426febe	32efa507b1a360412dee7ff30905ebdc	18112	votorantim	SP
26	23157f439f941d147f552ac180d68323	9ac370ba1e5aa108afc187fd57ce9442	18112	votorantim	SP
27	474090b9b8e1541d8ff3f614fb17398e	ea6ad96f2d4ea562128886eee252fc3e	18112	votorantim	SP
28	9b38c5e19f3e7f498ab36c0b095f39a6	0dc08717c1e08839d7c105878743f55e	18112	votorantim	SP
29	25ebf9efb39e9dc48d6c51a9896af9a5	2b229b2693dc783259f684b24fcd23a7	18112	votorantim	SP
30	74f0cc1d27884a102efa142ba1f540b3	2099f93ac5ea679cf2eb1de091ab87c9	18112	votorantim	SP
31	59699572fb53609e196978771c89a9dd	fce3cbf990e49bda6b74fa9c68640b06	18112	votorantim	SP
32	347a5aa5d381e31e52b84bfcae3e1eac	51fb62110fde8d22e80023732c46ef38	18112	votorantim	SP
33	00a39528c677a55852f57235f988b837	9f67d6145893e41e7ada2e646b84f658	18112	votorantim	SP
34	7c970ed3afc2cc695e6b4ca38954bef8	58e51c87c3ec742b03d5cee5e49dd664	18112	votorantim	SP
35	07d18ca5eb0bb3e96f406d74edb2462e	8ca4019dfb727a43ec2d7671290be758	18112	votorantim	SP
36	6f5d085fa90e41ad46b2f5623426febe	32efa507b1a360412dee7ff30905ebdc	18112	votorantim	SP
37	23157f439f941d147f552ac180d68323	9ac370ba1e5aa108afc187fd57ce9442	18112	votorantim	SP
38	474090b9b8e1541d8ff3f614fb17398e	ea6ad96f2d4ea562128886eee252fc3e	18112	votorantim	SP
39	9b38c5e19f3e7f498ab36c0b095f39a6	0dc08717c1e08839d7c105878743f55e	18112	votorantim	SP
40	25ebf9efb39e9dc48d6c51a9896af9a5	2b229b2693dc783259f684b24fcd23a7	18112	votorantim	SP
41	74f0cc1d27884a102efa142ba1f540b3	2099f93ac5ea679cf2eb1de091ab87c9	18112	votorantim	SP
42	59699572fb53609e196978771c89a9dd	fce3cbf990e49bda6b74fa9c68640b06	18112	votorantim	SP
43	347a5aa5d381e31e52b84bfcae3e1eac	51fb62110fde8d22e80023732c46ef38	18112	votorantim	SP

6.Find the top 5 cities with the most customers.

In [15]:

```
query = """SELECT customer_city, COUNT(*) AS customer_count
FROM customers
GROUP BY customer_city
ORDER BY customer_count DESC
LIMIT 5
```

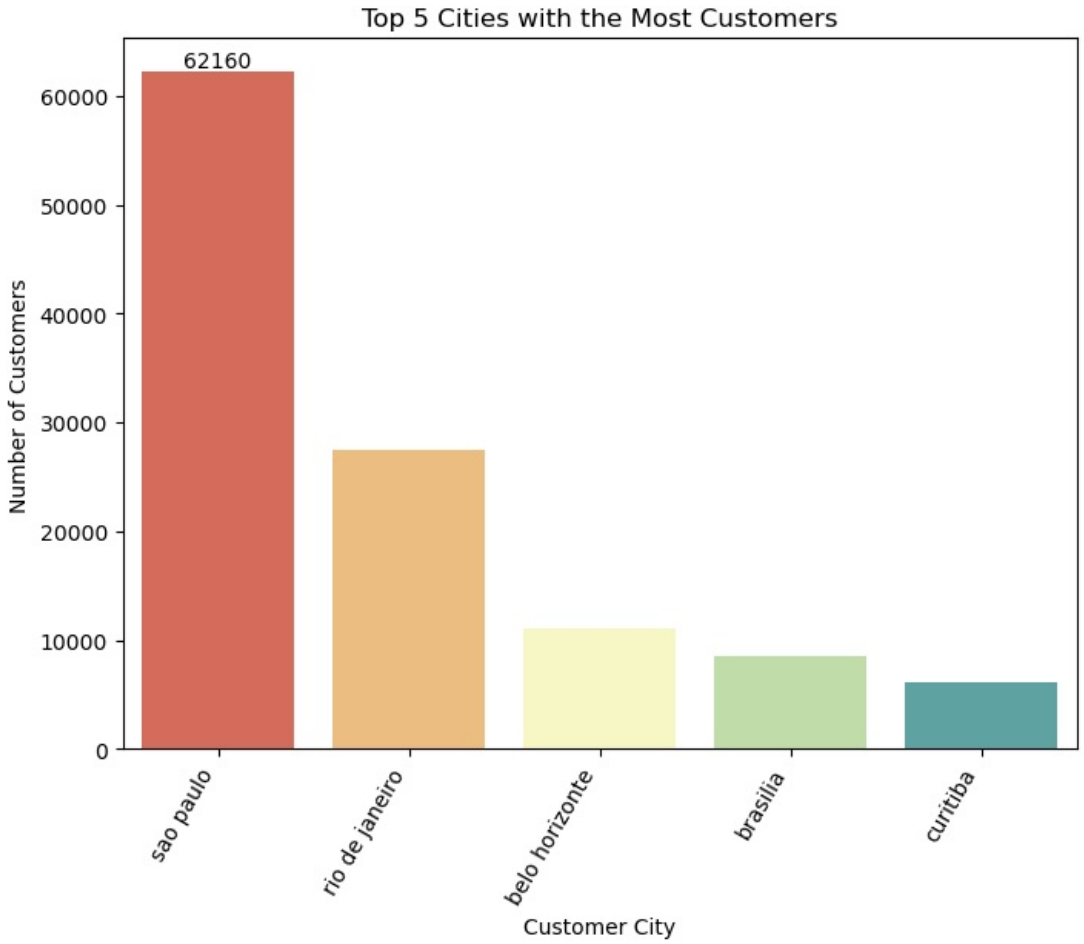
```
"""
cur.execute(query)
data = cur.fetchall()
data
df= pd.DataFrame(data,columns=["customer_city","customer_count"])
df
```

Out[15]:

	customer_city	customer_count
0	sao paulo	62160
1	rio de janeiro	27528
2	belo horizonte	11092
3	brasilia	8524
4	curitiba	6084

In [16]:

```
plt.figure(figsize=(8, 6))
ax = sns.barplot(x='customer_city', y='customer_count', data=df, palette="Spectral", hue='customer_city', dodge=
ax.bar_label(ax.containers[0])
plt.xlabel('Customer City')
plt.ylabel('Number of Customers')
plt.title('Top 5 Cities with the Most Customers')
plt.legend([],[], frameon=False)
plt.xticks(rotation=60, ha='right')
plt.show()
```



7.Count the number of unique customer IDs.

In [18]:

```
query = """SELECT COUNT(DISTINCT customer_unique_id) AS unique_customers FROM customers"""
cur.execute(query)
data = cur.fetchall()
data
"The no. of unique customer id is ",data[0][0]
```

Out[18]: ('The no. of unique customer id is ', 96096)

8.List the number of customers for each zip code prefix.

In [20]:

```
query = """SELECT customer_zip_code_prefix, COUNT(*) AS customer_count
FROM customers
GROUP BY customer_zip_code_prefix
ORDER BY customer_count DESC;
"""
```

```
cur.execute(query)
data = cur.fetchall()
data
df= pd.DataFrame(data,columns=["zip code","customer_count"])
df
```

Out[20]:

	zip code	customer_count
0	22790	568
1	24220	496
2	22793	484
3	24230	468
4	22775	440
...	...	...
14989	87145	4
14990	98860	4
14991	5538	4
14992	74980	4
14993	99043	4

14994 rows × 2 columns

9.Find customers who live in cities starting with 'A'.

In [22]:

```
query = """SELECT customer_id, customer_city FROM customers WHERE customer_city LIKE 'A%'"""
cur.execute(query)
data = cur.fetchall()
columns = ["customer_id", "customer_city"]
df = pd.DataFrame(data, columns=columns)
df.head(10)
```

Out[22]:

	customer_id	customer_city
0	690172ab319622688d3b4df42f676898	aparecida de goiania
1	1b2cb35b19b40b61f953d32ea157b337	araucaria
2	8392e3d4cfeec63f2a8bfea68bf1f91f	areia branca
3	fe6d73ac006153a398439253006e5adc	astolfo dutra
4	029ca19aa34f9311941ab931975cff1b	apucarana
5	af75138c42a3bc72157eaf45b6da592f	alta floresta
6	01509d555b8b6775f5847912d8c52d46	atibaia
7	7ee0952c8c02af4f93762c21ddb66607	americana
8	d5a5dcf7970697c1d38458beb2cb264f	arapongas
9	47c2b21c22de46f42a427db0ea223499	aripuana

10.Retrieve customers who have the same city and state.

In [24]:

```
query = """SELECT customer_city, customer_state, COUNT(*) AS customer_count
FROM customers
GROUP BY customer_city, customer_state
HAVING COUNT(*) > 1
"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["city","state","customers_id"])
df.head(10)
```

Out[24]:

	city	state	customers_id
0	franca	SP	644
1	sao bernardo do campo	SP	3752
2	sao paulo	SP	62160
3	mogi das cruzes	SP	1532
4	campinas	SP	5776
5	jaragua do sul	SC	356
6	timoteo	MG	216
7	curitiba	PR	6084
8	belo horizonte	MG	11092
9	montes claros	MG	844

11.Identify the cities where the number of customers is above the average number of customers per city.

In [26]:
```
query = """SELECT customer_city, COUNT(*) AS customer_count
FROM customers
GROUP BY customer_city
HAVING customer_count > (
    SELECT AVG(customer_count)
    FROM (
        SELECT COUNT(*) AS customer_count
        FROM customers
        GROUP BY customer_city
    ) AS subquery
)"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data,columns =["city","No of customers>avg.no"])
df.head(10)
```

Out[26]:

	city	No of customers>avg.no
0	franca	644
1	sao bernardo do campo	3752
2	sao paulo	62160
3	mogi das cruzes	1532
4	campinas	5776
5	jaragua do sul	356
6	timoteo	216
7	curitiba	6084
8	belo horizonte	11092
9	montes claros	844

12.Find the state with the highest number of unique customers.

In [28]:
```
query = """SELECT customer_state, COUNT(DISTINCT customer_unique_id) AS unique_customers
FROM customers
GROUP BY customer_state
ORDER BY unique_customers DESC
LIMIT 1"""
cur.execute(query)
data = cur.fetchall()
data
```

Out[28]:
```
[('SP', 40302)]
```

13.Find the state with the lowest number of unique customers.

In [30]:
```
query = """SELECT customer_state, COUNT(DISTINCT customer_unique_id) AS unique_customers
FROM customers
GROUP BY customer_state
ORDER BY unique_customers ASC
LIMIT 1"""
cur.execute(query)
data = cur.fetchall()
data
```

Out[30]:
```
[('RR', 45)]
```

14.Calculate the percentage of customers in each state relative to the total number of customers.

In [32]:
```
query = """
SELECT
    customer_state,
    COUNT(*) AS customer_count,
    ROUND((COUNT(*) * 100.0 / (SELECT COUNT(*) FROM customers)), 2) AS percentage
FROM
    customers
GROUP BY
    customer_state;
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns= ["customer_state", "customer_count", "percentage"])
df.head()
```

Out[32]:

	customer_state	customer_count	percentage
0	SP	166984	41.98
1	SC	14548	3.66
2	MG	46540	11.70
3	PR	20180	5.07
4	RJ	51408	12.92

15.Identify the most common zip code prefix in each state..

In [34]:
```
query = """SELECT customer_state, customer_zip_code_prefix, COUNT(*) AS customer_count
FROM customers
GROUP BY customer_state, customer_zip_code_prefix
ORDER BY customer_state, customer_count DESC"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["state","zip_code","customer_count"])
df.head()
```

Out[34]:

	state	zip_code	customer_count
0	AC	69900	56
1	AC	69918	52
2	AC	69915	36
3	AC	69901	32
4	AC	69919	28

16.List all unique payment types.

In [36]:
```
query = """SELECT DISTINCT payment_type FROM payments"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data,columns=["payment types"])
df
```

Out[36]:

	payment types
0	credit_card
1	UPI
2	voucher
3	debit_card
4	not_defined

17.Count the total number of payments.

In [38]:
```
query = """SELECT COUNT(*) AS total_payments FROM payments
"""
cur.execute(query)
data = cur.fetchall()
data
'Total no of payments are', data[0][0]
```

Out[38]: ('Total no of payments are', 311658)

18.Find the total payment value for each payment type.

In [40]:
```
query = """SELECT payment_type, SUM(payment_value) AS total_value
FROM payments
GROUP BY payment_type"""

cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["payment type","payment value"])
df
```

Out[40]:

	payment type	payment value
0	credit_card	3.762625e+07
1	UPI	8.608084e+06
2	voucher	1.138311e+06
3	debit_card	6.539694e+05
4	not_defined	0.000000e+00

19.Calculate the average payment value.

In [42]:
```
query = """SELECT AVG(payment_value) AS average_payment_value FROM payments;
"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[42]: 154.1003804175234

20.Count the number of installments for each payment type.

In [44]:
```
query = """SELECT payment_type, COUNT(payment_installments) AS total_installments
FROM payments
GROUP BY payment_type"""

cur.execute(query)
data = cur.fetchall()
data
df =pd.DataFrame(data,columns=["Payment type","No.of installments"])
df
```

Out[44]:

	Payment type	No.of installments
0	credit_card	230385
1	UPI	59352
2	voucher	17325
3	debit_card	4587
4	not_defined	9

21.Find the total payment value for each type of installment.

In [46]:
```
query = """SELECT payment_installments,sum(payment_value) AS total_Payment_value
FROM payments
GROUP BY payment_installments"""

cur.execute(query)
data = cur.fetchall()
data
df =pd.DataFrame(data,columns=["payment_installments","payment_value"])
df.head()
```

Out[46]:

	payment_installments	payment_value
0	8	3.940270e+06
1	1	1.772170e+07
2	2	4.737849e+06
3	3	4.473311e+06
4	6	2.467835e+06

22.List the top 5 payment types with the highest total payment value.

In [48]:
```
query = """SELECT payment_type, SUM(payment_value) AS total_value
FROM payments
GROUP BY payment_type
ORDER BY total_value DESC
LIMIT 5"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data,columns=["payment type","payment value"])
df
```

Out[48]:

	payment type	payment value
0	credit_card	3.762625e+07
1	UPI	8.608084e+06
2	voucher	1.138311e+06
3	debit_card	6.539694e+05
4	not_defined	0.000000e+00

23.Count the number of orders with more than one installment.

In [50]:
```
query = """SELECT COUNT(DISTINCT order_id) AS orders_with_installments
FROM payments
WHERE payment_installments > 1
"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[50]:
```
51170
```

24.Find the payment type with the highest average payment value.

In [52]:
```
query = """SELECT payment_type, AVG(payment_value) AS average_value
FROM payments
GROUP BY payment_type
ORDER BY average_value DESC
LIMIT 1"""
cur.execute(query)
data = cur.fetchall()
data[0]
```

Out[52]:
```
('credit_card', 163.31902064167556)
```

25.Calculate the percentage of payments made with credit card.

In [54]:
```
query = """SELECT ROUND(
    (SELECT COUNT(*) FROM payments WHERE payment_type = 'credit_card') /
    (SELECT COUNT(*) FROM payments) * 100, 2
) AS credit_card_percentage"""
cur.execute(query)
data = cur.fetchall()
data
"percentage of payments made with credit card", float(data[0][0])
```

Out[54]:
```
('percentage of payments made with credit card', 73.92)
```

26.Find the payment type with the lowest total payment value.

In [56]:
```
query = """SELECT payment_type, SUM(payment_value) AS total_value
FROM payments
GROUP BY payment_type
ORDER BY total_value ASC
LIMIT 1"""
cur.execute(query)
data = cur.fetchall()
data[0]
```

Out[56]:
```
('not_defined', 0.0)
```

27.Find the payment type with the highest total payment value.

In [58]:
```
query = """SELECT payment_type, SUM(payment_value) AS total_value
FROM payments
GROUP BY payment_type
```

```
ORDER BY total_value DESC
LIMIT 1"""
cur.execute(query)
data = cur.fetchall()
data[0]
```

Out[58]: ('credit_card', 37626252.570532426)

28.Fetch unique order statuses

In [60]:
```
query = """SELECT DISTINCT order_status FROM orders"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data,columns=["unique order"])
df
```

Out[60]:

	unique order
0	delivered
1	invoiced
2	shipped
3	processing
4	unavailable
5	canceled
6	created
7	approved

29.Count the total number of orders.

In [62]:
```
query = """SELECT COUNT(*) FROM orders"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[62]: 397764

30.Fetch all top 5 orders placed on a specific date "2017-05-26".

In [64]:
```
query = """SELECT * FROM orders WHERE DATE(order_purchase_timestamp) = '2017-05-26'"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data)
df.head()
```

Out[64]:

	0	1	2	3	4	5	6	7
0	2f246ee329fce6932458bc7a10625d2a	45f9053fc8b24a6d5afa05b6d1b330ab	delivered	2017-05-26 18:35:51	2017-05-26 18:45:10	2017-05-29 09:37:28	2017-06-09 08:26:30	2017-06-28 00:00:00
1	db88396cf9aab6635eb909ed423d6662	167db18e4fbd9ba3dec2aa5a7b79969b	delivered	2017-05-26 14:36:30	2017-05-26 14:50:14	2017-05-31 15:47:23	2017-06-13 13:52:05	2017-06-20 00:00:00
2	832e0a03685bfc20477ea18436e023a2	87722cfd2b0a53af8fdfee69e8c7feac	delivered	2017-05-26 16:43:10	2017-05-26 16:55:08	2017-05-29 15:20:53	2017-06-07 12:26:04	2017-06-20 00:00:00
3	c56bdcd0ef1249dcc18901b491b1b976	1358c61e82221ecc4d165d21123eb23d	delivered	2017-05-26 16:09:19	2017-05-27 16:15:17	2017-05-29 06:56:41	2017-06-05 13:33:08	2017-06-20 00:00:00
4	0e3fb5621c179dff735c7494afef624f	022aad8457b9fe76c22eaa3cbaf5b6e5	delivered	2017-05-26 02:28:47	2017-05-26 02:35:30	2017-06-05 17:38:33	2017-06-12 14:53:47	2017-06-27 00:00:00

31.Count the number of orders for each status.

In [66]:
```
query="""SELECT order_status, COUNT(*) AS order_count
FROM orders
GROUP BY order_status"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["status","No.of orders"])
```

```
df
```

Out[66]:

	status	No.of orders
0	delivered	385912
1	invoiced	1256
2	shipped	4428
3	processing	1204
4	unavailable	2436
5	canceled	2500
6	created	20
7	approved	8

32.Find the average approval time for orders (time between purchase and approval).

In [68]:

```
query="""SELECT AVG(TIMESTAMPDIFF(MINUTE, order_purchase_timestamp, order_approved_at)) AS avg_approval_time
FROM orders
WHERE order_approved_at IS NOT NULL;
"""
cur.execute(query)
data = cur.fetchall()
float(data[0][0])
```

Out[68]: 624.6608

33.Find the top 5 customers with the most orders.

In [70]:

```
query="""SELECT customer_id, COUNT(*) AS order_count
FROM orders
GROUP BY customer_id
ORDER BY order_count DESC
LIMIT 5
"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data,columns=["customer_id","order"])
df
```

Out[70]:

	customer_id	order
0	e04757bb7741d0781cda14c9be20ad2a	4
1	96c3d51816ee07cb78ebbfe0e8bdb889	4
2	f1913159750548eb81dca4b69fbd5e0c	4
3	6967af003c642a30a79242f2756a73d5	4
4	5eef8c5a24bd948fac341c6ebe3ce41e	4

34.Calculate the monthly order count for a specific year 2018.

In [72]:

```
query="""SELECT DATE_FORMAT(order_purchase_timestamp, '%Y-%m') AS month, COUNT(*) AS order_count
FROM orders
WHERE YEAR(order_purchase_timestamp) = 2018
GROUP BY month
ORDER BY month
"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data,columns=["specic year &month","order"])
df
```

Out[72]:

	specic year &month	order
0	2018-01	29076
1	2018-02	26912
2	2018-03	28844
3	2018-04	27756
4	2018-05	27492
5	2018-06	24668
6	2018-07	25168
7	2018-08	26048
8	2018-09	64
9	2018-10	16

35.Identify the correlation between order approval time and delivery time.

In [74]:
```
query="""SELECT
    TIMESTAMPDIFF(MINUTE, order_purchase_timestamp, order_approved_at) AS approval_time,
    TIMESTAMPDIFF(DAY, order_purchase_timestamp, order_delivered_customer_date) AS delivery_time
FROM orders
WHERE order_approved_at IS NOT NULL AND order_delivered_customer_date IS NOT NULL
"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["Approval_time","Delivery_time"])
df.head(10)
```

Out[74]:

	Approval_time	Delivery_time
0	10	8
1	1842	13
2	16	9
3	17	13
4	61	2
5	13	16
6	11	9
7	1941	9
8	10	18
9	9	12

36.Count the total number of products.

In [76]:
```
query="""SELECT COUNT(*) FROM products"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[76]: 131804

37.Fetch all products with a specific category,"Electronics".

In [78]:
```
query= """SELECT * FROM products WHERE product_category = 'Electronics'
"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data)
df.head()
```

Out[78]:

	0	1	2	3	4	5	6	7	8
0	750cf819d127191920eda79a4b6fb479	electronics	31.0	806.0	1.0	263.0	18.0	12.0	16.0
1	75b433ca888fe027b18dfac89a284667	electronics	37.0	374.0	1.0	1350.0	33.0	12.0	26.0
2	3fc3d637781e5d185455013606a6e2cd	electronics	32.0	1274.0	1.0	250.0	16.0	10.0	30.0
3	0e40b1ed4cfd3da1962ec91913e54ba8	electronics	52.0	424.0	1.0	175.0	22.0	11.0	15.0
4	98190f6a2c3253d07ca86525bb238162	electronics	25.0	813.0	1.0	1836.0	61.0	6.0	24.0

38.Count the number of products for each category in descending order.

In [80]:
```
query= """SELECT product_category, COUNT(*) AS product_count
FROM products
GROUP BY product_category"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["category","no.of products"])
df = df.sort_values(by="no.of products", ascending=False)
df
```

Out[80]:

	category	no.of products
10	bed table bath	12116
2	sport leisure	11468
7	Furniture Decoration	10628
13	HEALTH BEAUTY	9776
4	housewares	9340
...	...	...
61	Fashion Children's Clothing	20
69	House Comfort 2	20
64	PC Gamer	12
71	insurance and services	8
73	cds music dvds	4

74 rows × 2 columns

39.Calculate the average product weight.

In [82]:
```
query = """SELECT ROUND(AVG(product_weight_g), 2) AS avg_weight FROM products"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[82]: 2276.47

40.Find the product with the maximum weight.

In [84]:
```
query = """SELECT * FROM products ORDER BY product_weight_g DESC LIMIT 1"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[84]: '26644690fde745fc4654719c3904e1db'

41.Find the product with the maximum weight.

In [86]:
```
query = """SELECT * FROM products ORDER BY product_weight_g ASC LIMIT 1"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[86]: '5eb564652db742ff8f28759cd8d2652a'

42.Fetch products where the description length is greater than 100 characters.

In [88]:
```
query = """SELECT * FROM products WHERE product_description_length > 100;
"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[88]: '1e9e8ef04dbcff4541ed26657ea517e5'

43.Calculate the total number of photos for all products.

In [90]:
```
query = """SELECT SUM(product_photos_qty) AS total_photos FROM products"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[90]: 283176.0

44.Find products with a width greater than the average width of all products:

In [92]:
```
query = """SELECT * FROM products
WHERE product_width_cm > (SELECT AVG(product_width_cm) FROM products);
"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[92]: 'cef67bcfe19066a932b7673e239eb23d'

45.Find the top 5 heaviest products by category.

In [94]:
```
query = """
SELECT product_category, product_weight_g
FROM products
ORDER BY product_weight_g DESC
LIMIT 5
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns=["product_category", "product_weight_g"])
df
```

Out[94]:

	product_category	product_weight_g
0	bed table bath	40425.0
1	bed table bath	40425.0
2	bed table bath	40425.0
3	bed table bath	40425.0
4	Games consoles	30000.0

46.Find the bottom 5 heaviest products by category.

In [96]:
```
query = """
SELECT product_category, product_weight_g
FROM products
ORDER BY product_weight_g ASC
LIMIT 5
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns=["product_category", "product_weight_g"])
df
```

Out[96]:

	product_category	product_weight_g
0	None	None
1	babies	None
2	babies	None
3	None	None
4	babies	None

47.Calculate the percentage of products in each category relative to the total number of products.

In [98]:
```
query = """SELECT product_category,
       COUNT(*) AS product_count,
       ROUND((COUNT(*) / (SELECT COUNT(*) FROM products)) * 100, 2) AS percentage
FROM products
GROUP BY product_category;
"""
cur.execute(query)
data = cur.fetchall()
```

```
data[0]
df = pd.DataFrame(data,columns=["Product_category","no.of products","percentage"])
df
```

Out[98]:

	Product_category	no.of products	percentage
0	perfumery	3472	2.63
1	Art	220	0.17
2	sport leisure	11468	8.70
3	babies	3676	2.79
4	housewares	9340	7.09
...	...	...	...
69	House Comfort 2	20	0.02
70	Kitchen portable and food coach	40	0.03
71	insurance and services	8	0.01
72	CITTE AND UPHACK FURNITURE	40	0.03
73	cds music dvds	4	0.00

74 rows × 3 columns

48.Find the products that have the highest photo quantity in each category

In [100…

```
query = """
SELECT p1.product_category, p1.product_photos_qty
FROM products p1
JOIN (
    SELECT product_category, MAX(product_photos_qty) AS max_photos
    FROM products
    GROUP BY product_category
) p2 ON p1.product_category = p2.product_category AND p1.product_photos_qty = p2.max_photos
"""

cur.execute(query)
data = cur.fetchall()
new_df = pd.DataFrame(data, columns=["product_category", "photo_quantity"])
new_df
```

Out[100…

	product_category	photo_quantity
0	Fashion Calcados	12.0
1	stationary store	10.0
2	home appliances	10.0
3	Imported books	7.0
4	Construction Tools Garden	6.0
...	...	...
671	Bags Accessories	10.0
672	climatization	7.0
673	Construction Tools Garden	6.0
674	home appliances	10.0
675	musical instruments	10.0

676 rows × 2 columns

49.Calculate the average dimensions (length, height, width) of products for each category.

In [102…

```
query = """
SELECT product_category,
       AVG(product_length_cm) AS avg_length,
       AVG(product_height_cm) AS avg_height,
       AVG(product_width_cm) AS avg_width
FROM products
GROUP BY product_category"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data,columns=["product category","length","height","width"])
df
```

Out[102…

	product category	length	height	width
0	perfumery	20.322581	13.059908	16.668203
1	Art	35.927273	11.800000	23.363636
2	sport leisure	31.280084	18.109871	20.824206
3	babies	37.147059	21.617647	28.717865
4	housewares	31.859957	22.337901	24.821413
...	...	...	...	...
69	House Comfort 2	53.200000	15.200000	26.200000
70	Kitchen portable and food coach	28.500000	25.300000	24.600000
71	insurance and services	26.500000	28.500000	13.000000
72	CITTE AND UPHACK FURNITURE	46.300000	34.400000	41.300000
73	cds music dvds	35.000000	15.000000	25.000000

74 rows × 4 columns

50.Find the product with the smallest volume (length × width × height).

In [104…

```
query = """
SELECT product_category,
       min(product_length_cm) AS smallest_length,
       min(product_height_cm) AS smallest_height,
       min(product_width_cm) AS smallest_width
FROM products
GROUP BY product_category"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data,columns=["product category","length","height","width"])
df
```

Out[104…

	product category	length	height	width
0	perfumery	11.0	2.0	8.0
1	Art	16.0	2.0	11.0
2	sport leisure	12.0	2.0	10.0
3	babies	14.0	2.0	8.0
4	housewares	10.0	2.0	11.0
...	...	...	...	...
69	House Comfort 2	20.0	8.0	20.0
70	Kitchen portable and food coach	20.0	6.0	14.0
71	insurance and services	26.0	22.0	11.0
72	CITTE AND UPHACK FURNITURE	25.0	10.0	21.0
73	cds music dvds	35.0	15.0	25.0

74 rows × 4 columns

51.Find the product with the largest volume (length × width × height).

In [106…

```
query = """
SELECT product_category,
       max(product_length_cm) AS largest_length,
       max(product_height_cm) AS largest_height,
       max(product_width_cm) AS largest_width
FROM products
GROUP BY product_category"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data,columns=["product category","length","height","width"])
df
```

Out[106…

	product category	length	height	width
0	perfumery	70.0	102.0	35.0
1	Art	100.0	62.0	70.0
2	sport leisure	105.0	105.0	97.0
3	babies	102.0	97.0	84.0
4	housewares	105.0	105.0	90.0
...	...	...	...	...
69	House Comfort 2	105.0	20.0	33.0
70	Kitchen portable and food coach	40.0	33.0	38.0
71	insurance and services	27.0	35.0	15.0
72	CITTE AND UPHACK FURNITURE	70.0	73.0	100.0
73	cds music dvds	35.0	15.0	25.0

74 rows × 4 columns

52.Fetch the top 3 categories with the highest average product weight.

In [108…

```
query = """SELECT product_category, AVG(product_weight_g) AS avg_weight
FROM products
GROUP BY product_category
ORDER BY avg_weight DESC
LIMIT 3"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns=["product_category", "avg.product_weight_g"])
df
```

Out[108…

	product_category	avg.product_weight_g
0	CITTE AND UPHACK FURNITURE	13190.000000
1	Furniture office	12740.867314
2	Furniture Kitchen Service Area Dinner and Garden	11598.563830

53.List all unique seller cities.

In [110…

```
query = """SELECT DISTINCT seller_city FROM sellers"""
cur.execute(query)
data = cur.fetchall()
data
df = pd.DataFrame(data, columns=["all unique seller cities "])
df
```

Out[110…

	all unique seller cities
0	campinas
1	mogi guacu
2	rio de janeiro
3	sao paulo
4	braganca paulista
...	...
605	aparecida de goiania
606	bandeirantes
607	vitoria de santo antao
608	palotina
609	leme

610 rows × 1 columns

54.Count the number of sellers in each state.

In [112…

```
query = """SELECT seller_state, COUNT(seller_id) AS seller_count
FROM sellers
GROUP BY seller_state;
"""
cur.execute(query)
```

```
data = cur.fetchall()
data
df = pd.DataFrame(data, columns=["state","no of sellers"])
df
```

Out[112…

	state	no of sellers
0	SP	7396
1	RJ	684
2	PE	36
3	PR	1396
4	GO	160
5	SC	760
6	BA	76
7	DF	120
8	RS	516
9	MG	976
10	RN	20
11	MT	16
12	CE	52
13	PB	24
14	AC	4
15	ES	92
16	RO	8
17	PI	4
18	MS	20
19	SE	8
20	MA	4
21	AM	4
22	PA	4

55.Find the number of sellers in a specific city (e.g., 'sao paulo').

In [114…

```
query = """SELECT COUNT(*) AS seller_count
FROM sellers
WHERE seller_city = 'sao paulo'"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[114… 2780

56.Retrieve all sellers in a specific state (e.g., 'MT').

In [116…

```
query = """SELECT * FROM sellers
WHERE seller_state = 'MT'
"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["seller_id","zip_code","city","state"])
df
```

Out[116…

	seller_id	zip_code	city	state
0	99002261c568a84cce14d43fcffb43ea	78095	cuiaba	MT
1	1e9d5a33694bddb76316fd1f54734d20	78820	jaciara	MT
2	2dee2ce60de9709b1a24083217181a1f	78552	sinop	MT
3	abcd2cb37d46c2c8fb1bf071c859fc5b	78020	cuiaba	MT
4	99002261c568a84cce14d43fcffb43ea	78095	cuiaba	MT
5	1e9d5a33694bddb76316fd1f54734d20	78820	jaciara	MT
6	2dee2ce60de9709b1a24083217181a1f	78552	sinop	MT
7	abcd2cb37d46c2c8fb1bf071c859fc5b	78020	cuiaba	MT
8	99002261c568a84cce14d43fcffb43ea	78095	cuiaba	MT
9	1e9d5a33694bddb76316fd1f54734d20	78820	jaciara	MT
10	2dee2ce60de9709b1a24083217181a1f	78552	sinop	MT
11	abcd2cb37d46c2c8fb1bf071c859fc5b	78020	cuiaba	MT
12	99002261c568a84cce14d43fcffb43ea	78095	cuiaba	MT
13	1e9d5a33694bddb76316fd1f54734d20	78820	jaciara	MT
14	2dee2ce60de9709b1a24083217181a1f	78552	sinop	MT
15	abcd2cb37d46c2c8fb1bf071c859fc5b	78020	cuiaba	MT

57. Find the top 5 states with the most sellers.

In [118…

```
query = """SELECT seller_state, COUNT(seller_id) AS seller_count
FROM sellers
GROUP BY seller_state
ORDER BY seller_count DESC
LIMIT 5
"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["state","no.of seller"])
df
```

Out[118…

	state	no.of seller
0	SP	7396
1	PR	1396
2	MG	976
3	SC	760
4	RJ	684

58.Count the number of unique zip codes sellers are located in.

In [120…

```
query = """SELECT COUNT(DISTINCT seller_zip_code_prefix) AS unique_zip_codes
FROM sellers
"""
cur.execute(query)
data = cur.fetchall()
data[0][0]
```

Out[120… 2246

59.List all sellers whose zip code prefix starts with '2'.

In [122…

```
query = """SELECT * FROM sellers
WHERE seller_zip_code_prefix LIKE '2%';
"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["seller_id","zip_code","city","state"])
df
```

Out[122…

	seller_id	zip_code	city	state
0	ce3ad9de960102d0677a81f5d0bb7b2d	20031	rio de janeiro	RJ
1	c240c4061717ac1806ae6ee72be3533b	20920	rio de janeiro	RJ
2	e9e446d01bd10a97a8ffcfc4a3a20cb2	2261	sao paulo	SP
3	0747d5bb69f0586cc869d8af4c50f93e	28990	saquarema	RJ
4	c89cf7c468a48af70aada384e722f9e2	25730	petropolis	RJ
...	...	...	...	...
1211	30c7f28fd3a5897b2c82d152bb760c17	24020	niteroi	RJ
1212	55c96925041a14097b6a7825554f4ad5	2122	sao paulo	SP
1213	d66c11a9572221d92fbb8c4897db5f9b	20770	rio de janeiro	RJ
1214	db6a4d4b5f1f5f98820ce6ce2619e2de	2968	sao paulo	SP
1215	dde698c6d0bd24834c586e5111c2bba7	25755	petropolis	RJ

1216 rows × 4 columns

60.Find the average number of sellers per city.

In [124…

```
query = """SELECT AVG(seller_count) AS avg_sellers_per_city
FROM (
    SELECT seller_city, COUNT(seller_id) AS seller_count
    FROM sellers
    GROUP BY seller_city
) AS city_seller_counts
"""
cur.execute(query)
data = cur.fetchall()
float(data[0][0])
```

Out[124… 20.2951

61.Identify the cities with the highest number of sellers in each state.

In [126…

```
query = """SELECT seller_state, seller_city, MAX(seller_count) AS max_sellers
FROM (
    SELECT seller_state, seller_city, COUNT(seller_id) AS seller_count
    FROM sellers
    GROUP BY seller_state, seller_city
) AS state_city_sellers
GROUP BY seller_state, seller_city
ORDER BY seller_count DESC """
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["state","city","no.of sellers"])
df
```

Out[126…

	state	city	no.of sellers
0	SP	sao paulo	2780
1	PR	curitiba	496
2	RJ	rio de janeiro	372
3	MG	belo horizonte	264
4	SP	ribeirao preto	208
...	...	...	...
630	GO	aparecida de goiania	4
631	PR	bandeirantes	4
632	PE	vitoria de santo antao	4
633	PR	palotina	4
634	SP	leme	4

635 rows × 3 columns

62.Calculate the percentage of sellers in each state relative to the total number of sellers.

In [128…

```
query = """SELECT seller_state,
       COUNT(seller_id) AS seller_count,
```

```
       ROUND((COUNT(seller_id) / (SELECT COUNT(*) FROM sellers)) * 100, 2) AS percentage
FROM sellers
GROUP BY seller_state
ORDER BY seller_count DESC"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["state","Total sellers","percentage of sellers"])
df
```

Out[128…

	state	Total sellers	percentage of sellers
0	SP	7396	59.74
1	PR	1396	11.28
2	MG	976	7.88
3	SC	760	6.14
4	RJ	684	5.53
5	RS	516	4.17
6	GO	160	1.29
7	DF	120	0.97
8	ES	92	0.74
9	BA	76	0.61
10	CE	52	0.42
11	PE	36	0.29
12	PB	24	0.19
13	RN	20	0.16
14	MS	20	0.16
15	MT	16	0.13
16	RO	8	0.06
17	SE	8	0.06
18	AC	4	0.03
19	PI	4	0.03
20	MA	4	0.03
21	AM	4	0.03
22	PA	4	0.03

63.Find the seller with the maximum and minimum zip code prefix in each state.

In [130…

```
query = """SELECT seller_state,
       MAX(seller_zip_code_prefix) AS max_zip_code,
       MIN(seller_zip_code_prefix) AS min_zip_code
FROM sellers
GROUP BY seller_state;
"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["state","Max zip code","Min zip code"])
df
```

Out[130…

	state	Max zip code	Min zip code
0	SP	95076	1001
1	RJ	28990	20020
2	PE	55602	50751
3	PR	87900	80010
4	GO	76500	72801
5	SC	89900	88010
6	BA	48602	40130
7	DF	73020	70070
8	RS	99730	90010
9	MG	39801	30111
10	RN	59775	21210
11	MT	78820	78020
12	CE	63540	60025
13	PB	58865	58030
14	AC	69900	69900
15	ES	29704	29010
16	RO	76900	76804
17	PI	64033	64033
18	MS	79400	79090
19	SE	49980	49055
20	MA	65072	65072
21	AM	69005	69005
22	PA	85960	85960

64.List the states that have more than 100 sellers.

In [132…

```
query = """SELECT seller_state, COUNT(seller_id) AS seller_count
FROM sellers
GROUP BY seller_state
HAVING seller_count > 100
"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["state","seller_count > 100"])
df
```

Out[132…

	state	seller_count > 100
0	SP	7396
1	RJ	684
2	PR	1396
3	GO	160
4	SC	760
5	DF	120
6	RS	516
7	MG	976

65.Find the average zip code prefix of sellers in each city.

In [134…

```
query = """SELECT seller_city, AVG(seller_zip_code_prefix) AS avg_zip_code
FROM sellers
GROUP BY seller_city
ORDER BY AVG(seller_zip_code_prefix) DESC
"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["city","avg.zip code"])
df
```

Out[134…

	city	avg.zip code
0	erechim	99700.0000
1	ronda alta	99670.0000
2	carazinho	99500.0000
3	soledade	99300.0000
4	campina das missoes	98975.0000
...	...	...
605	sao paulop	3581.0000
606	sao paulo / sao paulo	3407.0000
607	sp / sp	3363.0000
608	sao pauo	2051.0000
609	sao paulo sp	1207.0000

610 rows × 2 columns

66.Identify the seller with the highest freight cost per order item.

In [136…

```
query = """SELECT seller_id, AVG(freight_value) AS avg_freight_cost
FROM order_items
GROUP BY seller_id
ORDER BY avg_freight_cost DESC
LIMIT 1;
"""
cur.execute(query)
data = cur.fetchall()
data[0]
```

Out[136… `('6fa9202c10491e472dffd59a3e82b2a3', 308.336664835612)`

67.Find the top 5 total number of items sold for each product.

In [138…

```
query = """SELECT product_id, COUNT(order_item_id) AS total_items_sold
FROM order_items
GROUP BY product_id
ORDER BY total_items_sold DESC
limit 5;
"""
cur.execute(query)
data = cur.fetchall()
data
df= pd.DataFrame(data,columns=["product","total no of items"])
df
```

Out[138…

	product	total no of items
0	aca2eb7d00ea1a7b8ebd4e68314663af	1581
1	99a4788cb24856965c36a24e339b6058	1464
2	422879e10f46682990de24d770e7f83d	1452
3	389d119b48cf3043d311335e499d9c6b	1176
4	368c6c730842d78016ad823897a372db	1164

68.Find the product and seller combination that generated the highest revenue.

In [140…

```
query = """SELECT product_id, seller_id, SUM(price) AS total_revenue
FROM order_items
GROUP BY product_id, seller_id
ORDER BY total_revenue DESC
LIMIT 1
"""
cur.execute(query)
data = cur.fetchall()
data[0]
```

Out[140…

```
('bb50f2e236e5eea0100680137654686c',
 'f7ba60f8c3f99e7ee4042fdef03b70c4',
 191655.0)
```

69.Find the order with the maximum number of items.

In [142…

```
query = """SELECT order_id, COUNT(order_item_id) AS item_count
```

```
FROM order_items
GROUP BY order_id
ORDER BY item_count DESC
LIMIT 1"""
cur.execute(query)
data = cur.fetchall()
data[0]
```

Out[142… ('8272b63d03f5f79c56e9e4120aec44ef', 63)

70.Determine the average price and total number of items sold for each shipping limit date.

In [144…
```
query = """SELECT shipping_limit_date, AVG(price) AS avg_price, COUNT(order_item_id) AS total_items_sold
FROM order_items
GROUP BY shipping_limit_date
ORDER BY shipping_limit_date DESC
limit 10
"""
cur.execute(query)
data = cur.fetchall()
data
df=pd.DataFrame(data,columns=["Shipping limit date","Avg price","Total sold items"])
df
```

Out[144…

	Shipping limit date	Avg price	Total sold items
0	2020-04-09 22:35:08	99.989998	6
1	2020-02-05 03:30:51	69.989998	3
2	2020-02-03 20:23:22	75.989998	3
3	2018-09-18 21:10:15	999.989990	3
4	2018-09-14 12:30:56	20.000000	3
5	2018-09-14 02:09:37	599.989990	3
6	2018-09-13 14:55:28	299.000000	3
7	2018-09-12 13:24:27	16.900000	3
8	2018-09-12 03:15:36	49.990002	3
9	2018-09-11 22:43:50	39.990002	6

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

In [146…
```
query = """
SELECT
    (SUM(CASE WHEN payment_installments >= 1 THEN 1 ELSE 0 END) / COUNT(*)) * 100
AS percentage_installments
FROM payments;
"""
cur.execute(query)
data = cur.fetchall()
float(data[0][0])
```

Out[146… 99.9981

72.Calculate the number of orders per month in 2018.

In [148…
```
query = """SELECT
    MONTHNAME(order_purchase_timestamp) AS month,
    COUNT(*) AS num_orders
FROM
    orders
WHERE
    YEAR(order_purchase_timestamp) = 2018
GROUP BY
    month
ORDER BY
    month """
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns=["months", "order_counts"])
df
```

Out[148…

	months	order_counts
0	April	27756
1	August	26048
2	February	26912
3	January	29076
4	July	25168
5	June	24668
6	March	28844
7	May	27492
8	October	16
9	September	64

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

In [150…

```
query = """
SELECT c.customer_city, COUNT(oi.order_item_id) / COUNT(DISTINCT o.order_id) AS avg_products_per_order
FROM order_items oi
JOIN orders o ON oi.order_id = o.order_id
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY c.customer_city
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns=["customer_city", "avg_products_per_order"])
df
```

Out[150…

	customer_city	avg_products_per_order
0	abadia dos dourados	48.0000
1	abadiania	48.0000
2	abaete	48.0000
3	abaetetuba	61.0909
4	abaiara	48.0000
...	...	...
4105	xinguara	53.3333
4106	xique-xique	48.0000
4107	zacarias	48.0000
4108	ze doca	48.0000
4109	zortea	48.0000

4110 rows × 2 columns

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

In [152…

```
query = """SELECT p.product_category,
       ROUND(SUM(oi.price) / (SELECT SUM(price) FROM order_items) * 100, 2) AS revenue_percentage
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
GROUP BY p.product_category"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns=["product_category", "total revenue"])
df
```

Out[152…

	product_category	total revenue
0	HEALTH BEAUTY	37.04
1	sport leisure	29.08
2	Cool Stuff	18.70
3	computer accessories	26.84
4	Watches present	35.46
...	...	...
69	Kitchen portable and food coach	0.12
70	House Comfort 2	0.02
71	CITTE AND UPHACK FURNITURE	0.13
72	insurance and services	0.01
73	cds music dvds	0.02

74 rows × 2 columns

75.dentify the correlation between product price and the number of times a product has been purchased.

In [154…

```
query = """
SELECT p.product_id, COUNT(oi.order_item_id) AS purchase_count
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
GROUP BY p.product_id
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns=["product_id", "purchase_count"])
df
```

Out[154…

	product_id	purchase_count
0	ef92defde845ab8450f9d70c526ef70f	60
1	310ae3c140ff94b03219ad0adc3c778f	24
2	8cab8abac59158715e0d70a36c807415	24
3	b50c950aba0dcead2c48032a690ce817	48
4	5ed9eaf534f6936b51d0b6c5e4d5c2e9	252
...	...	...
32946	e7597059b9e4eca21a96eed55693b31b	12
32947	32284a34d19761adc56333216faff3da	12
32948	d426c52017387e708aa9e72eb90797b2	12
32949	bd421826916d3e1d445cb860cea3c0fb	12
32950	5c8d8bd9771a2391092e1fbba43debeb	12

32951 rows × 2 columns

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

In [156…

```
query = """SELECT oi.seller_id, SUM(oi.price) AS total_revenue
FROM order_items oi
GROUP BY oi.seller_id
ORDER BY total_revenue DESC"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data,columns=["sellers","revenue"])
df
```

Out[156…

	sellers	revenue
0	4869f7a5dfa277a7dca6462dcf3b52b2	688417.885048
1	53243585a1d6dc2643021fd1853d8905	668328.148636
2	4a3ca9315b744ce9f8e9374361493884	601418.764378
3	fa1c13f2614d7b5c4749cbc52fecda94	582126.088188
4	7c67e1448b00f6e969d365cea6b010ab	563771.675817
...	...	...
3090	34aefe746cd81b7f3b23253ea28bef39	24.000000
3091	702835e4b785b67a084280efca355756	22.800000
3092	1fa2d3def6adfa70e58c276bb64fe5bb	20.700000
3093	77128dec4bec4878c37ab7d6169d6f26	19.500000
3094	cf6f6bc4df3999b9c6440f124fb2f687	10.500000

3095 rows × 2 columns

77Calculate the moving average of order values for each customer over their order history.

In [158…

```
query = """SELECT customer_id, order_purchase_timestamp, AVG(price) OVER (PARTITION BY customer_id
ORDER BY order_purchase_timestamp ROWS BETWEEN 4 PRECEDING AND CURRENT ROW) AS moving_avg_order_value
FROM orders
JOIN order_items ON orders.order_id = order_items.order_id;
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data,columns=["customer_id","order_timestamp","avg.order_value"])
df
```

Out[158…

	customer_id	order_timestamp	avg.order_value
0	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	89.800003
1	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	89.800003
2	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	89.800003
3	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	89.800003
4	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	89.800003
...	...	...	...
1351795	ffffa3172527f765de70084a7e53aae8	2017-09-02 11:53:32	10.900000
1351796	ffffa3172527f765de70084a7e53aae8	2017-09-02 11:53:32	10.900000
1351797	ffffa3172527f765de70084a7e53aae8	2017-09-02 11:53:32	10.900000
1351798	ffffa3172527f765de70084a7e53aae8	2017-09-02 11:53:32	10.900000
1351799	ffffa3172527f765de70084a7e53aae8	2017-09-02 11:53:32	10.900000

1351800 rows × 3 columns

78.Calculate the cumulative sales per month for each year.

In [160…

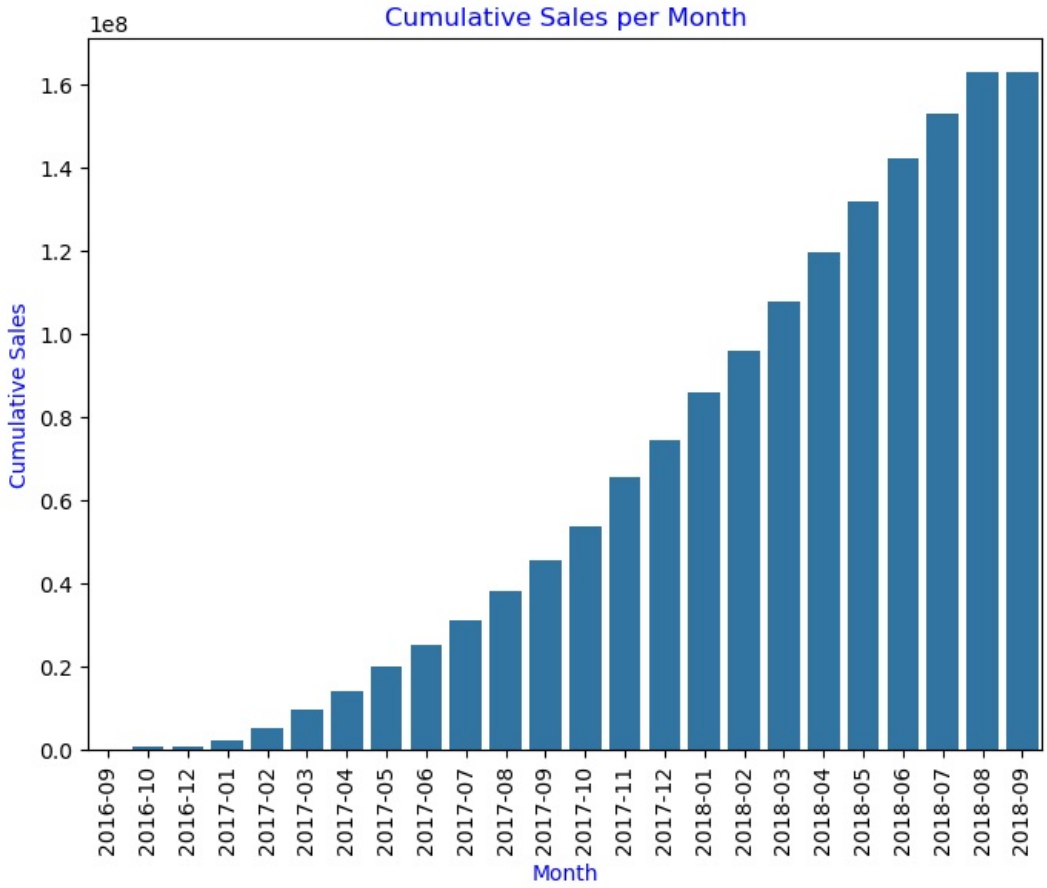
```
query = """
WITH monthly_sales AS (
    SELECT DATE_FORMAT(order_purchase_timestamp, '%Y-%m') AS month,
           SUM(price) AS total_sales
    FROM orders
    JOIN order_items ON orders.order_id = order_items.order_id
    GROUP BY month
)
SELECT month,
       total_sales,
       SUM(total_sales) OVER (ORDER BY month) AS cumulative_sales
FROM monthly_sales;
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns=["month", "total_sales", "cumulative_sales"])
df
```

Out[160…

	month	total_sales	cumulative_sales
0	2016-09	3.208320e+03	3.208320e+03
1	2016-10	5.940919e+05	5.973002e+05
2	2016-12	1.308000e+02	5.974310e+05
3	2017-01	1.443754e+06	2.041185e+06
4	2017-02	2.967636e+06	5.008822e+06
5	2017-03	4.492132e+06	9.500953e+06
6	2017-04	4.319127e+06	1.382008e+07
7	2017-05	6.072854e+06	1.989293e+07
8	2017-06	5.196463e+06	2.508940e+07
9	2017-07	5.976378e+06	3.106577e+07
10	2017-08	6.887660e+06	3.795343e+07
11	2017-09	7.492820e+06	4.544626e+07
12	2017-10	7.970633e+06	5.341689e+07
13	2017-11	1.212326e+07	6.554014e+07
14	2017-12	8.926970e+06	7.446711e+07
15	2018-01	1.140036e+07	8.586748e+07
16	2018-02	1.013014e+07	9.599762e+07
17	2018-03	1.179856e+07	1.077962e+08
18	2018-04	1.195977e+07	1.197560e+08
19	2018-05	1.195821e+07	1.317142e+08
20	2018-06	1.038149e+07	1.420957e+08
21	2018-07	1.074609e+07	1.528417e+08
22	2018-08	1.025624e+07	1.630980e+08
23	2018-09	1.740000e+03	1.630997e+08

In [161…

```
# Visualize the cumulative sales
plt.figure(figsize=(8, 6))
ax = sns.barplot(x='month', y='cumulative_sales', data=df)
plt.xlabel('Month',color="blue")
plt.ylabel('Cumulative Sales',color="blue")
plt.title('Cumulative Sales per Month',color="blue")
plt.xticks(rotation=90)
plt.show()
```



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

In [163…
```
query = """
SELECT YEAR(order_purchase_timestamp) AS year,
       SUM(price) AS total_sales,
       LAG(SUM(price), 1) OVER (ORDER BY YEAR(order_purchase_timestamp)) AS prev_year_sales,
       (SUM(price) - LAG(SUM(price), 1) OVER (ORDER BY YEAR(order_purchase_timestamp))) /
       LAG(SUM(price), 1) OVER (ORDER BY YEAR(order_purchase_timestamp)) * 100 AS yoy_growth
FROM orders
JOIN order_items ON orders.order_id = order_items.order_id
GROUP BY year
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns=["year", "total_sales", "prev_year_sales", "yoy_growth"])
df
```

Out[163…

	year	total_sales	prev_year_sales	yoy_growth
0	2016	5.974310e+05	NaN	NaN
1	2017	7.386968e+07	5.974310e+05	12264.554087
2	2018	8.863261e+07	7.386968e+07	19.985094

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

In [165…
```
query = """SELECT customer_id, YEAR(order_purchase_timestamp) AS year, SUM(price) AS total_spent
FROM orders
JOIN order_items ON orders.order_id = order_items.order_id
GROUP BY customer_id, year
ORDER BY year, total_spent DESC
LIMIT 3"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data,columns=["customer_id","year","total spent"])
df
```

Out[165…

	customer_id	year	total spent
0	a9dc96b027d1252bbac0a9b72d837fc6	2016	16788.000000
1	1d34ed25963d5aae4cf3d7f3a4cda173	2016	15599.879883
2	4a06381959b6670756de02e07b83815f	2016	14388.000000

In [ ]:

Ameet raj

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: