Python+SQL Ecommerce Data Analysis Project

In [14]:

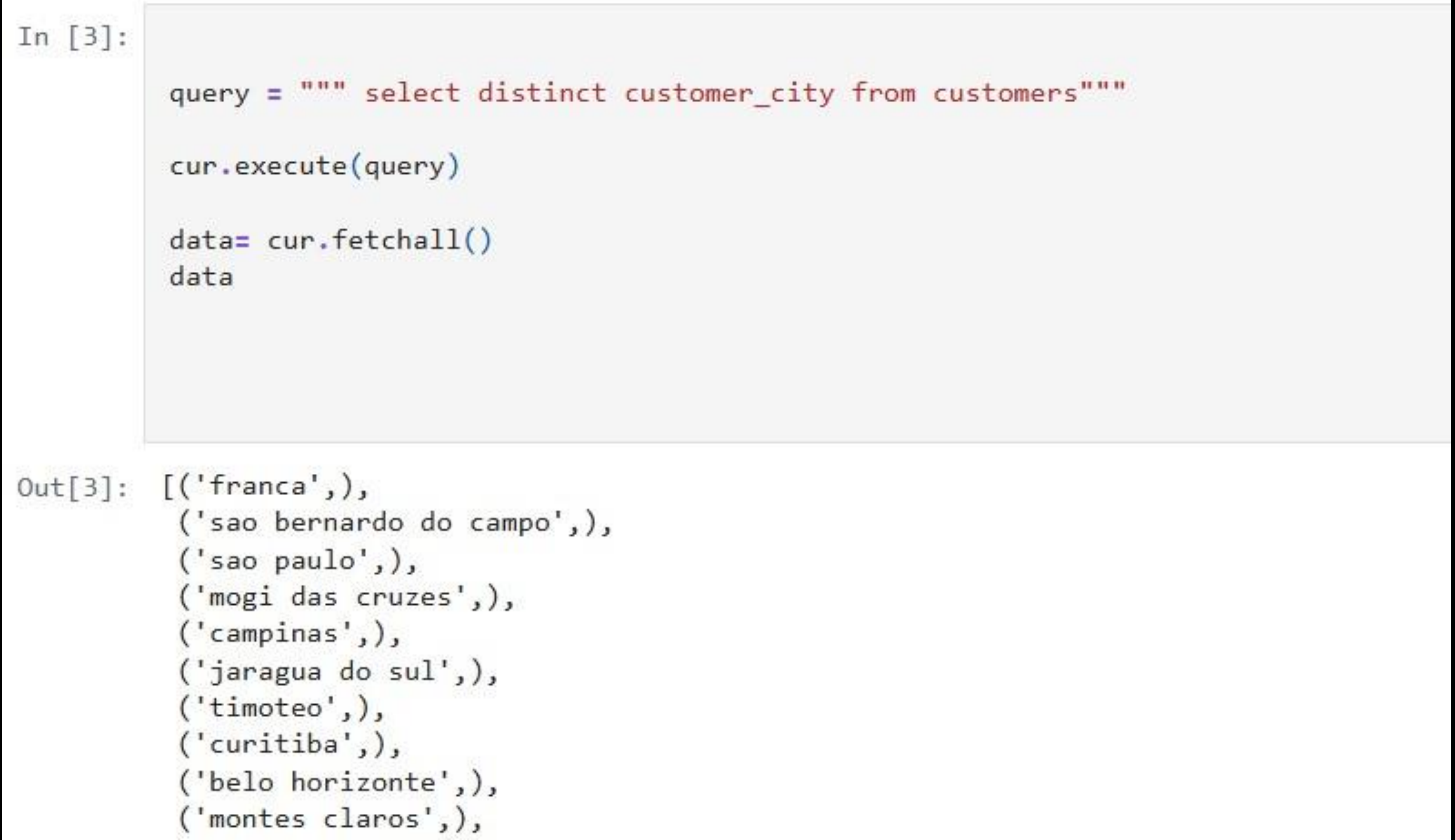
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import mysql.connector
import numpy as np

db = mysql.connector.connect(host ="localhost",
                             username ="root",
                             password ="1234",
                             database = "Ecommerce"


)

cur = db.cursor()
```

1. List all unique cities where customers are located.

In [14]:

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

db = mysql.connector.connect(host ="localhost",
                             username ="root",
                             password ="1234",
                             database = "Ecommerce"


)

cur = db.cursor()
```

1. List all unique cities where customers are located.



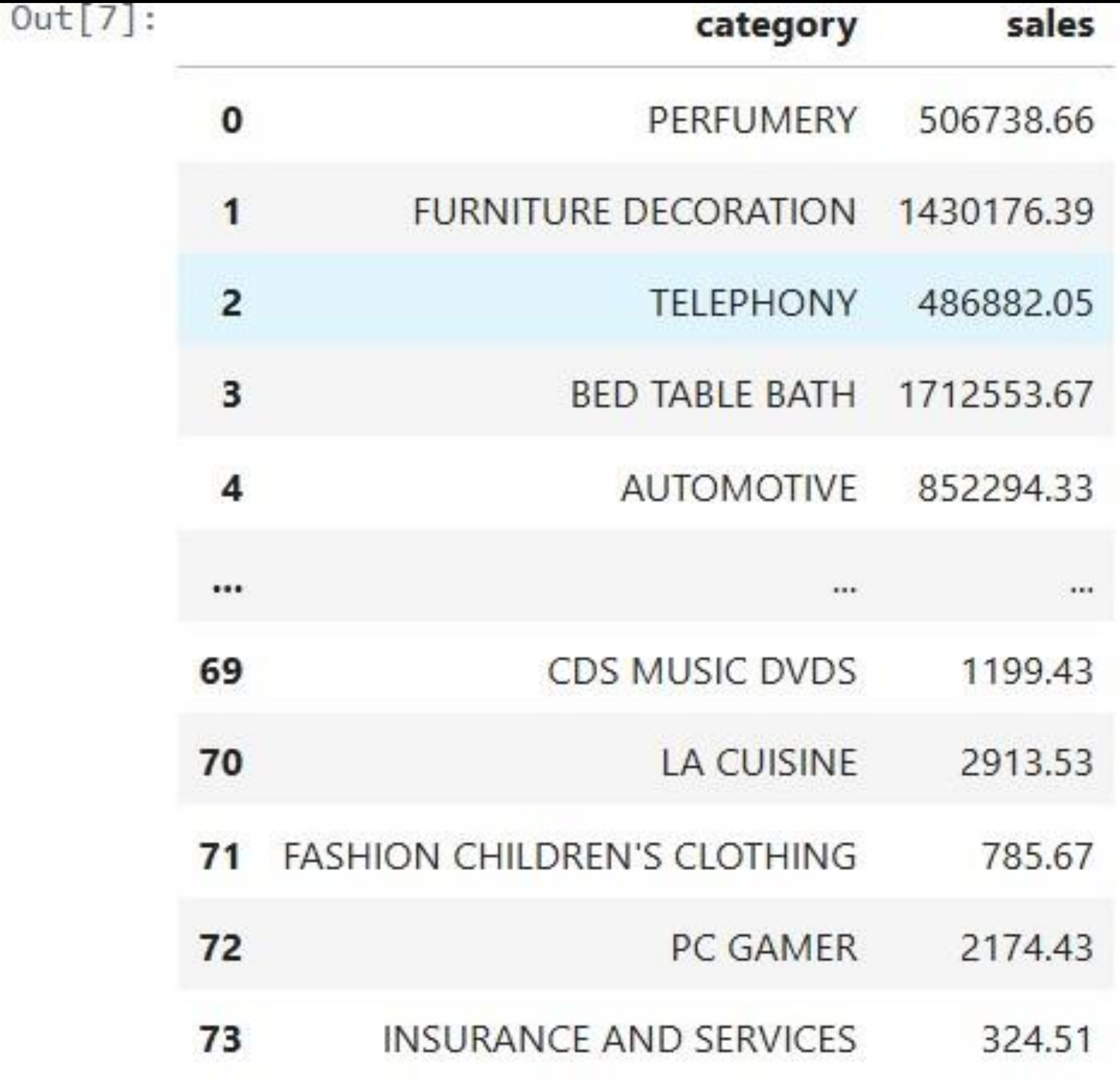
```
In [3]:

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

cur.execute(query)

data= cur.fetchall()
data
```

```
Out[3]: [('franca',),
         ('sao bernardo do campo',),
         ('sao paulo',),
         ('mogi das cruzes',),
         ('campinas',),
         ('jaragua do sul',),
         ('timoteo',),
         ('curitiba',),
         ('belo horizonte',),
         ('montes claros',),
```

3. Find the total sales per category.

In [7]:

```
query = """ select upper(products.product_category) category,
round(sum(payments.payment_value),2) 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
```



Out[7]:

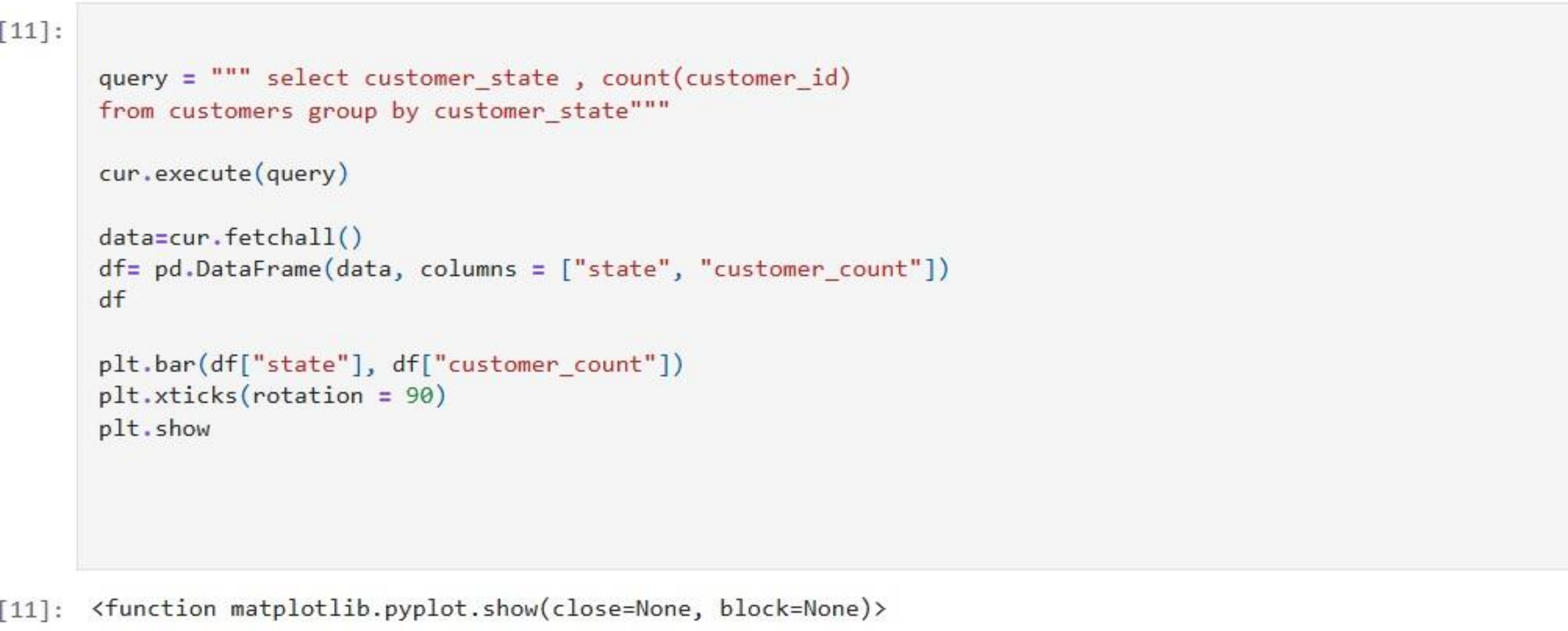
	category	sales
0	PERFUMERY	506738.66
1	FURNITURE DECORATION	1430176.39
2	TELEPHONY	486882.05
3	BED TABLE BATH	1712553.67
4	AUTOMOTIVE	852294.33
...	...	...
69	CDS MUSIC DVDS	1199.43
70	LA CUISINE	2913.53
71	FASHION CHILDREN'S CLOTHING	785.67
72	PC GAMER	2174.43
73	INSURANCE AND SERVICES	324.51

4. Calculate the percentage of orders that were paid installments.

```
[9]:

query = """ select (sum(case when payment_installments > 1 then 1
else 0 end ))/count(*)*100 from payments
"""
cur.execute(query)
data = cur.fetchall()
"The percentage of order that were paid in installments is ", data[0][0]
```

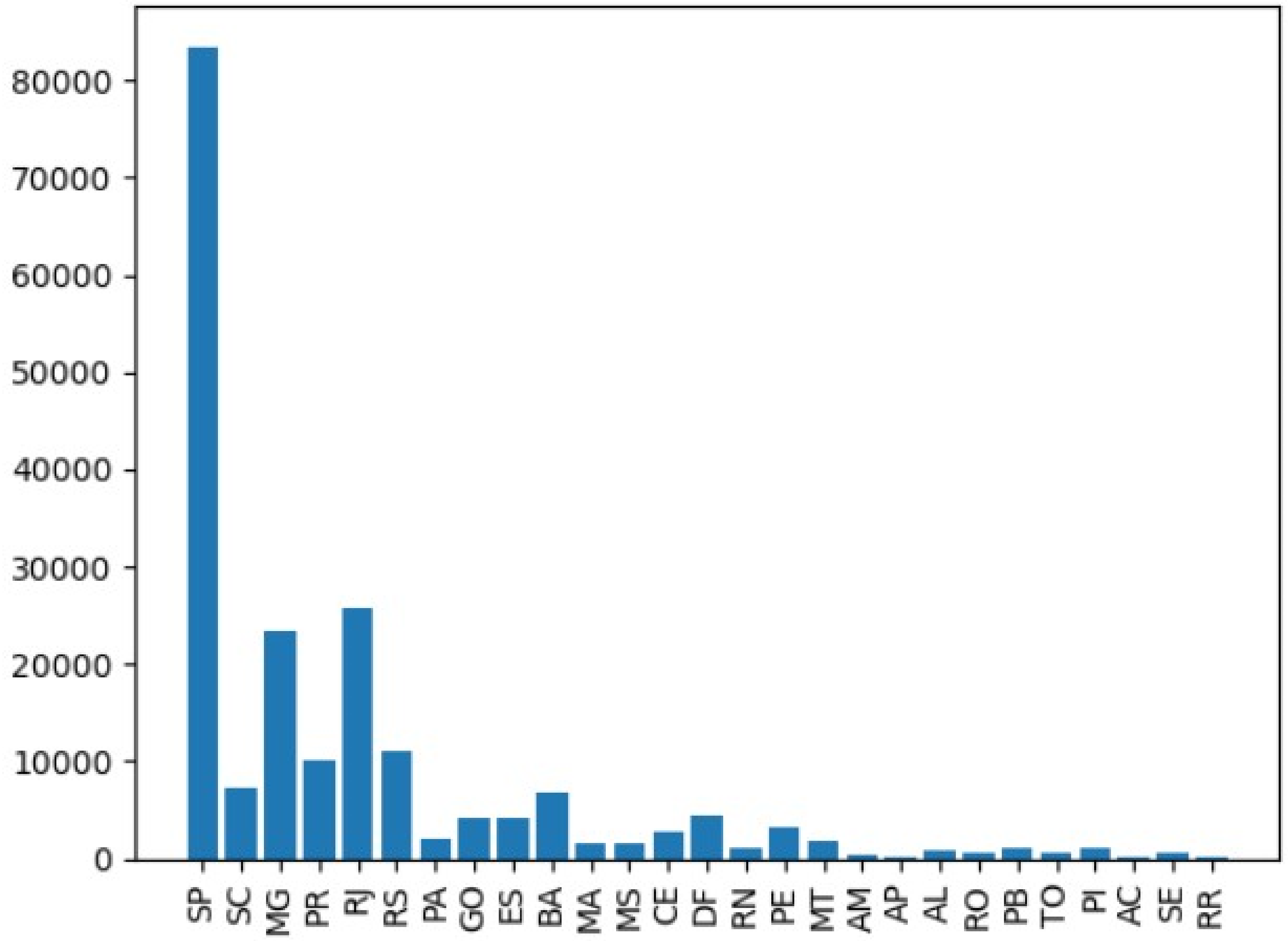
```
t[9]: ('The percentage of order that were paid in installments is ',
       Decimal('49.4176'))
```

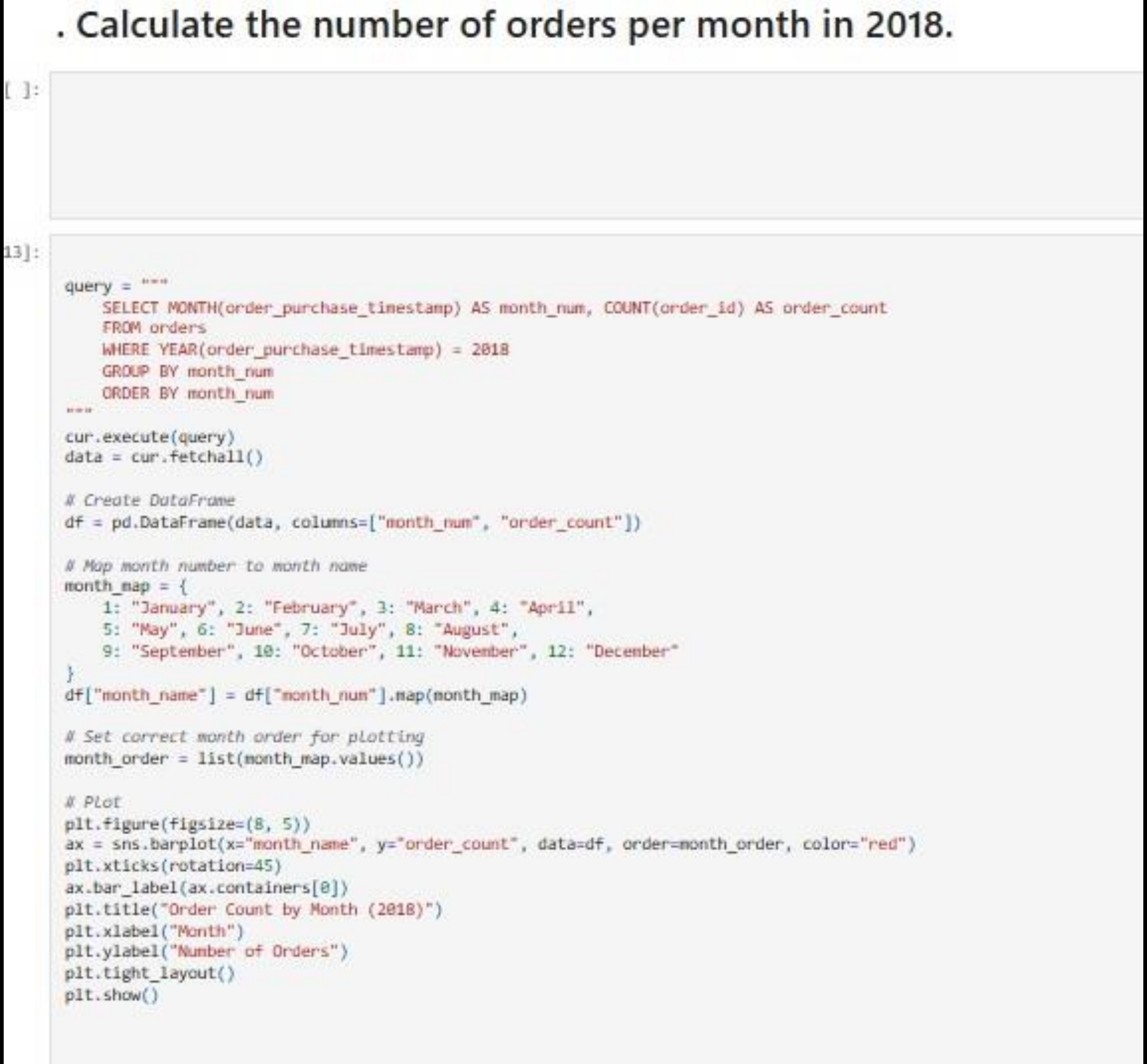
5. Count the number of customers from each state.



```
[11]:

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", "customer_count"])
df

plt.bar(df["state"], df["customer_count"])
plt.xticks(rotation = 90)
plt.show
```

```
[11]: <function matplotlib.pyplot.show(close=None, block=None)>
```


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

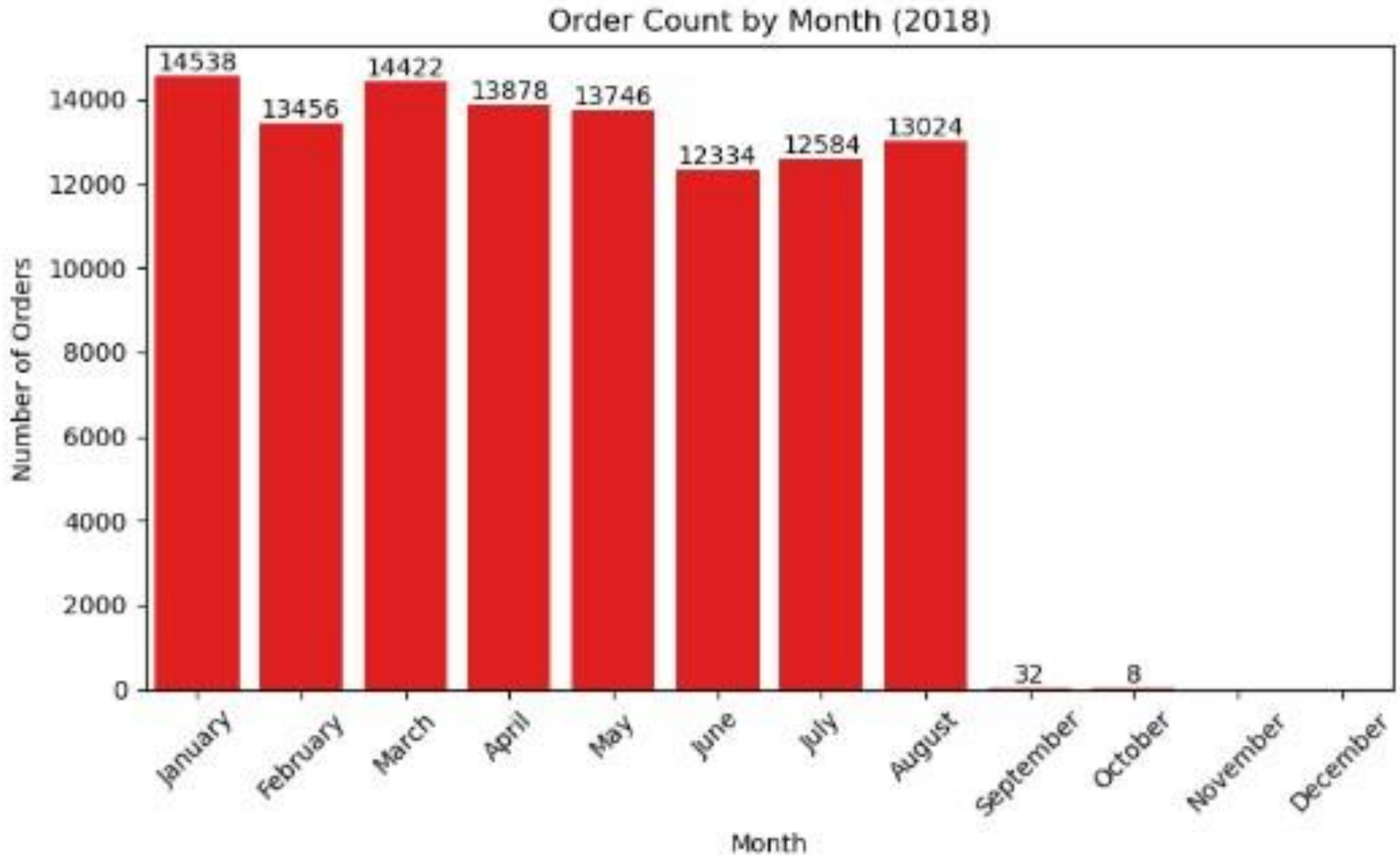


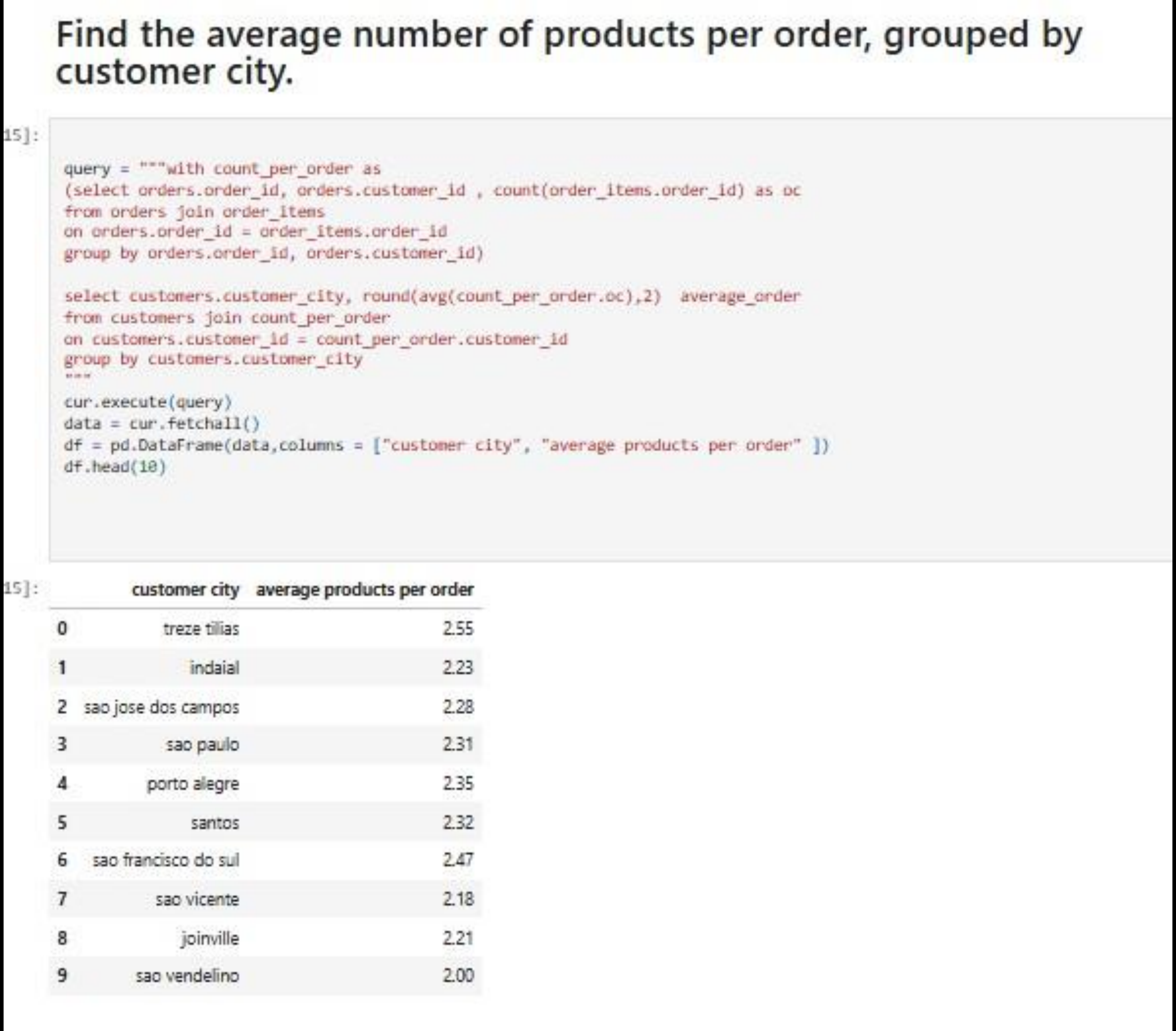
. Calculate the number of orders per month in 2018.

[ ]:

13]:

```
query = """
    SELECT MONTH(order_purchase_timestamp) AS month_num, COUNT(order_id) AS order_count
    FROM orders
    WHERE YEAR(order_purchase_timestamp) = 2018
    GROUP BY month_num
    ORDER BY month_num
"""
cur.execute(query)
data = cur.fetchall()

# Create DataFrame
df = pd.DataFrame(data, columns=["month_num", "order_count"])

# Map month number to month name
month_map = {
    1: "January", 2: "February", 3: "March", 4: "April",
    5: "May", 6: "June", 7: "July", 8: "August",
    9: "September", 10: "October", 11: "November", 12: "December"
}
df["month_name"] = df["month_num"].map(month_map)

# Set correct month order for plotting
month_order = list(month_map.values())

# Plot
plt.figure(figsize=(8, 5))
ax = sns.barplot(x="month_name", y="order_count", data=df, order=month_order, color="red")
plt.xticks(rotation=45)
ax.bar_label(ax.containers[0])
plt.title("Order Count by Month (2018)")
plt.xlabel("Month")
plt.ylabel("Number of Orders")
plt.tight_layout()
plt.show()
```


Order Count by Month (2018)
Number of Orders
Month
0
2000
4000
6000
8000
10000
12000
14000
14538
13456
14422
13878
13746
12334
12584
13024
32
8
January
February
March
April
May
June
July
August
September
October
November
December



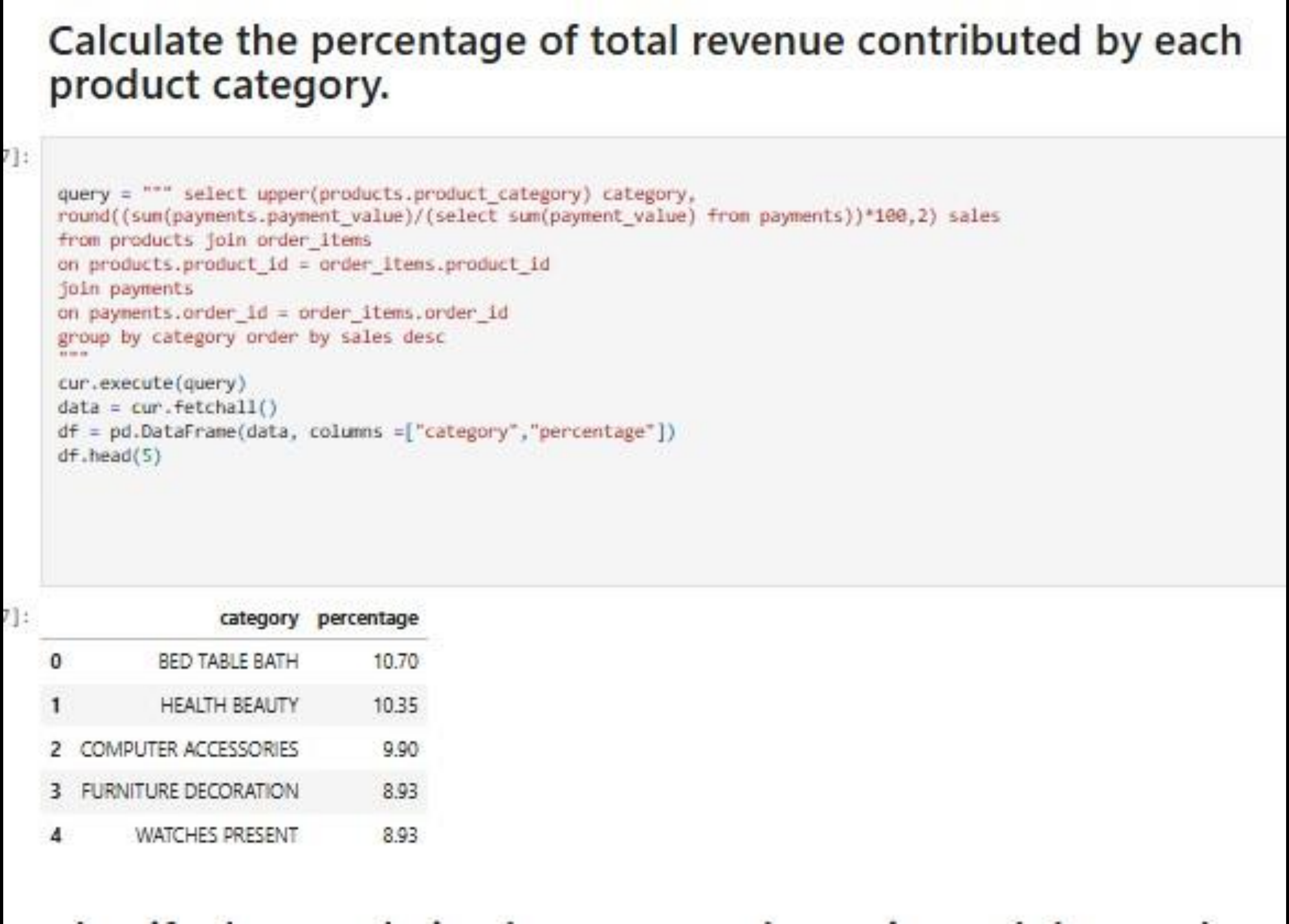
Find the average number of products per order, grouped by customer city.

15]:

```
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)  average_order
from customers join count_per_order
on customers.customer_id = count_per_order.customer_id
group by customers.customer_city
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data,columns = ["customer city", "average products per order" ])
df.head(10)
```

15]:

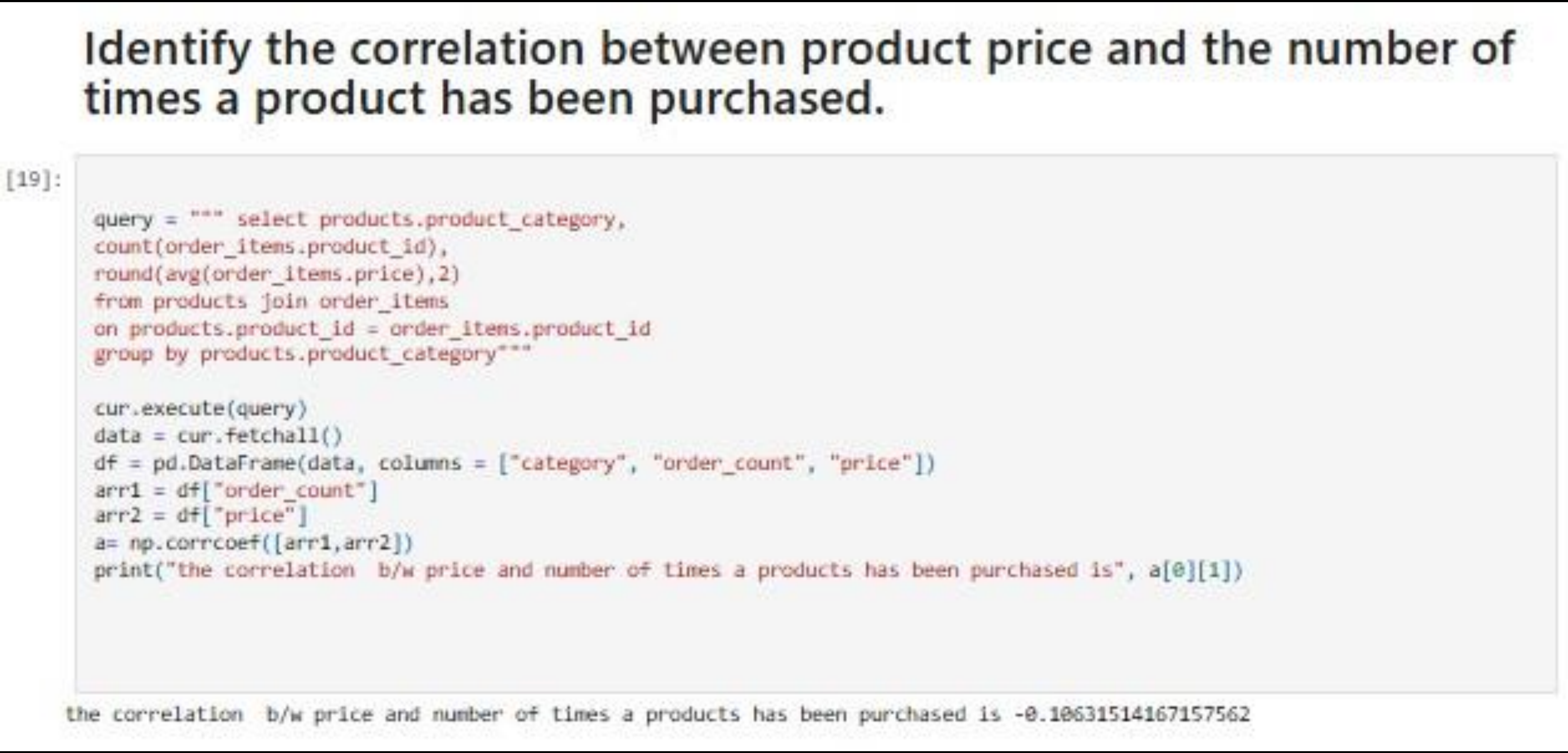
	customer city	average products per order
0	treze tilias	2.55
1	indaial	2.23
2	sao jose dos campos	2.28
3	sao paulo	2.31
4	porto alegre	2.35
5	santos	2.32
6	sao francisco do sul	2.47
7	sao vicente	2.18
8	joinville	2.21
9	sao vendelino	2.00



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

```
query = """ select upper(products.product_category) category,
round((sum(payments.payment_value)/(select sum(payment_value) from payments))*100,2) 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 sales desc
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns =["category","percentage"])
df.head(5)
```

```
               category  percentage
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
```

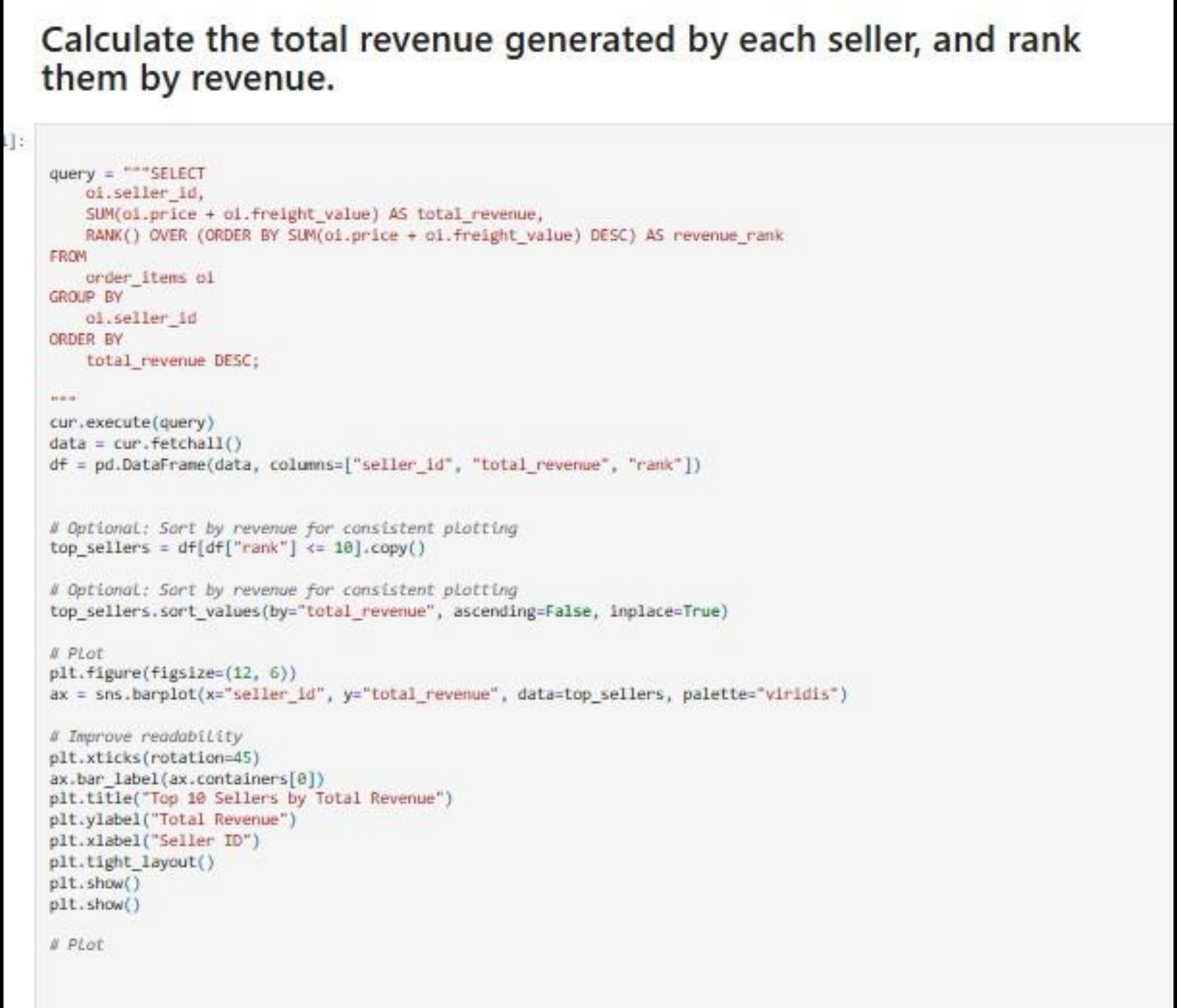


Identify the correlation between product price and the number of times a product has been purchased.

[19]:

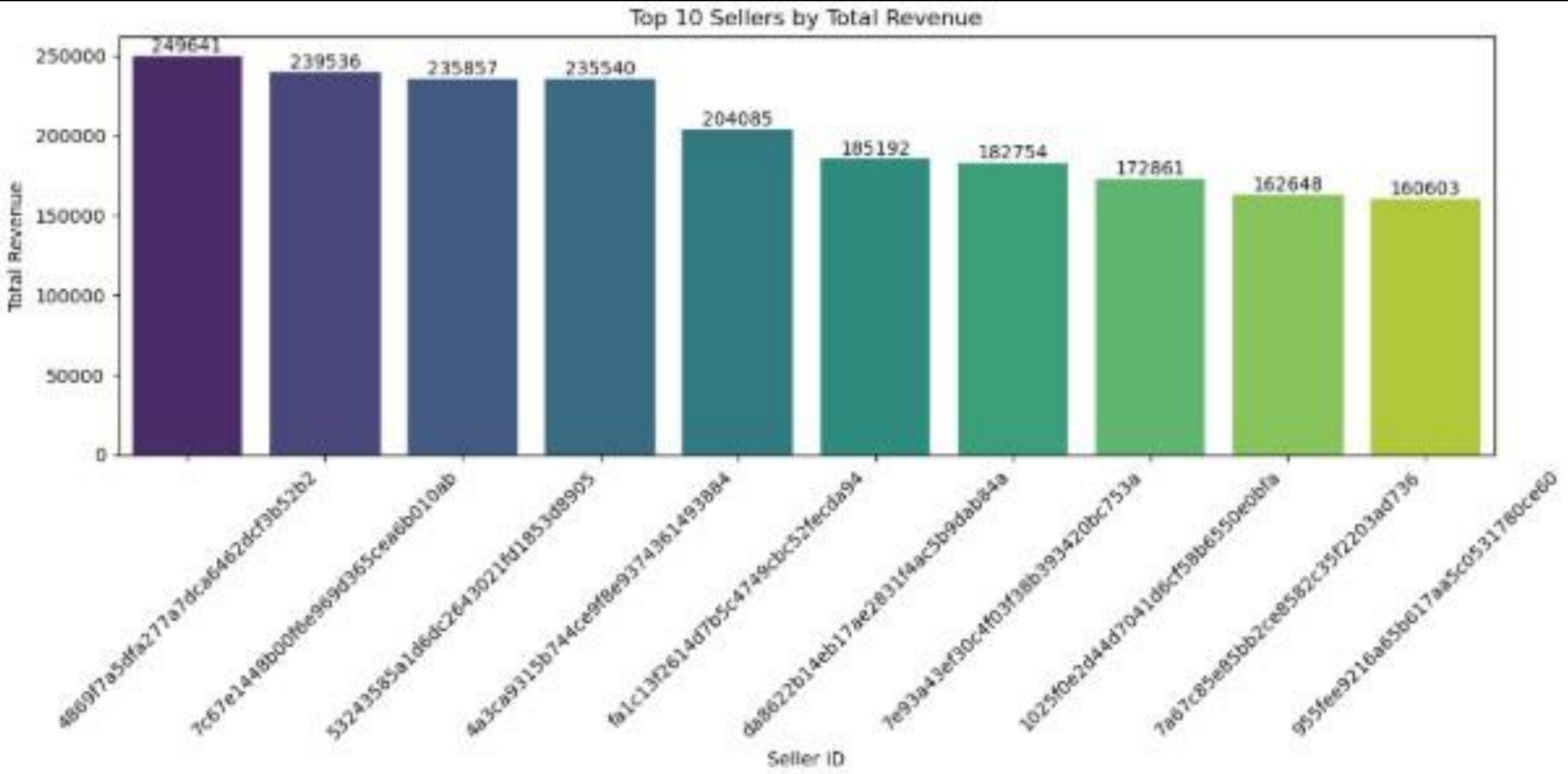
```
query = """ select products.product_category,
count(order_items.product_id),
round(avg(order_items.price),2)
from products join order_items
on products.product_id = order_items.product_id
group by products.product_category"""

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns = ["category", "order_count", "price"])
arr1 = df["order_count"]
arr2 = df["price"]
a= np.corrcoef([arr1,arr2])
print("the correlation  b/w price and number of times a products has been purchased is", a[0][1])
```

```
the correlation  b/w price and number of times a products has been purchased is -0.10631514167157562
```



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

```
query = """SELECT
    oi.seller_id,
    SUM(oi.price + oi.freight_value) AS total_revenue,
    RANK() OVER (ORDER BY SUM(oi.price + oi.freight_value) DESC) AS revenue_rank
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=["seller_id", "total_revenue", "rank"])


# Optional: Sort by revenue for consistent plotting
top_sellers = df[df["rank"] <= 10].copy()

# Optional: Sort by revenue for consistent plotting
top_sellers.sort_values(by="total_revenue", ascending=False, inplace=True)

# Plot
plt.figure(figsize=(12, 6))
ax = sns.barplot(x="seller_id", y="total_revenue", data=top_sellers, palette="viridis")

# Improve readability
plt.xticks(rotation=45)
ax.bar_label(ax.containers[0])
plt.title("Top 10 Sellers by Total Revenue")
plt.ylabel("Total Revenue")
plt.xlabel("Seller ID")
plt.tight_layout()
plt.show()
plt.show()

# Plot
```

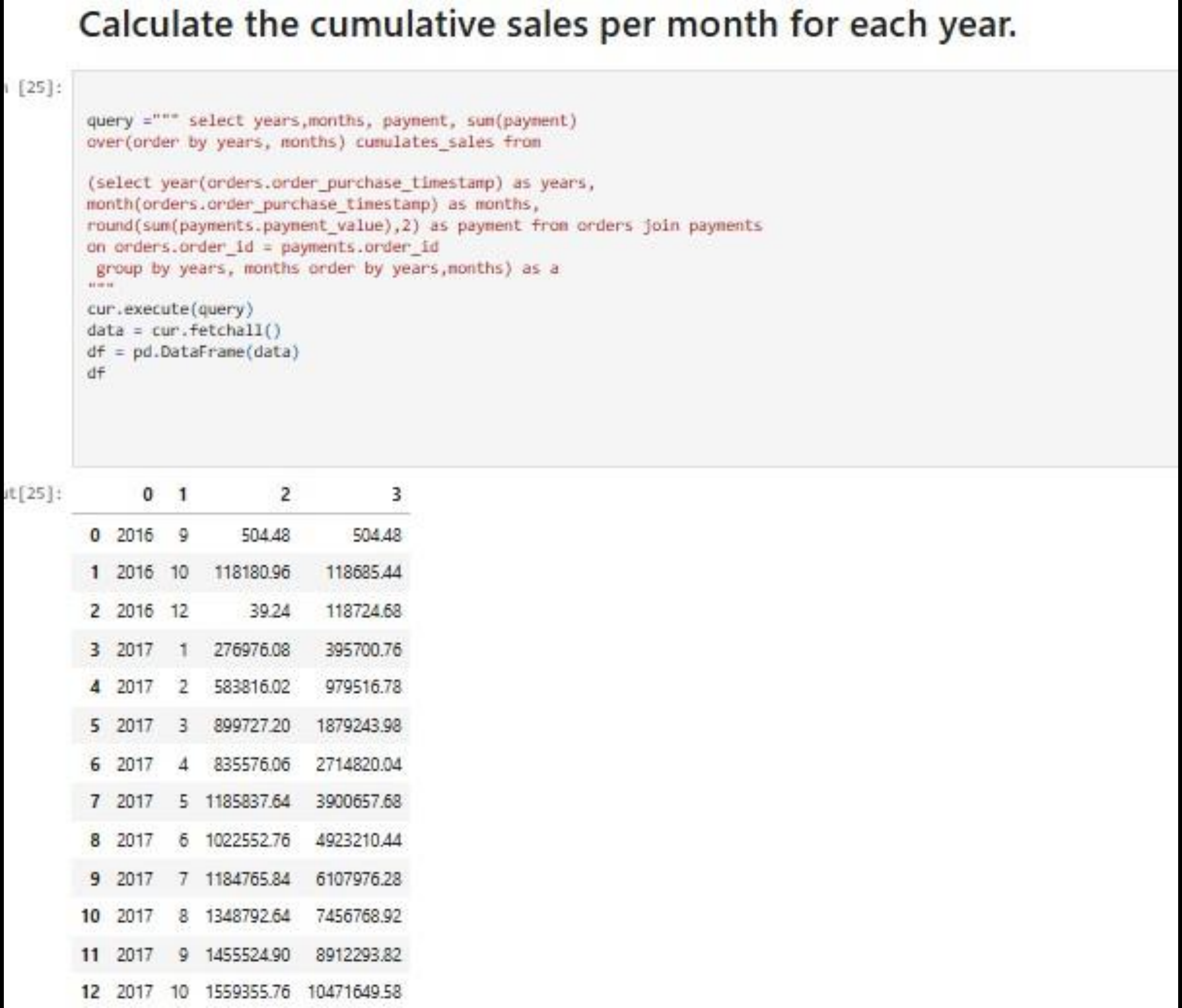

Top 10 Sellers by Total Revenue
Total Revenue
250000
200000
150000
100000
50000
0
249641
239536
235857
235540
204085
185192
182754
172861
162648
160603
4869f7a5dfa277a7dca6462dcf3b52b2
7c67e1448b00f6e969d365cea6b010ab
53243585a1d6dc2643021fd1853d8905
4a3ca9315b744ce9f8e9374361493884
fa1c13f2614d7b5c4749cbc52fecda94
da8622b14eb17ae2831f4ac5b9dab84a
7e93a43ef30c4f03f38b393420bc753a
1025f0e2d44d7041d6cf58b6550e0bfa
7a67c85e85bb2ce8582c35f2203ad736
955fee9216a65b617aa5c0531780ce60
Seller ID

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

23]:

```
query ="""SELECT
    o.customer_id,
    o.order_id,
    o.order_purchase_timestamp,
    oi.price + oi.freight_value AS order_value
FROM
    orders o
JOIN
    order_items oi ON o.order_id = oi.order_id
ORDER BY
    o.customer_id, o.order_purchase_timestamp;
"""
cur.execute(query)
data = cur.fetchall()


df = pd.DataFrame(data, columns=['customer_id', 'order_id', 'order_purchase_timestamp', 'order_value'])

# Ensure the timestamp is a datetime object
df['order_purchase_timestamp'] = pd.to_datetime(df['order_purchase_timestamp'])

# Sort by customer_id and order_purchase_timestamp to maintain chronological order
df = df.sort_values(by=['customer_id', 'order_purchase_timestamp'])

# Calculate the moving average (e.g., using a window size of 3)
df['moving_avg'] = df.groupby('customer_id')['order_value'].rolling(window=3, min_periods=1).mean().reset_index(level=0, dr

# View the result
print(df.head(5))
```

```
                        customer_id                          order_id  \
0  00012a2ce6f8dcda20d059ce98491703  5f79b5b0931d63f1a42989eb65b9da6e
1  00012a2ce6f8dcda20d059ce98491703  5f79b5b0931d63f1a42989eb65b9da6e
2  000161a058600d5901f007fab4c27140  a44895d095d7e0702b6a162fa2dbeced
3  000161a058600d5901f007fab4c27140  a44895d095d7e0702b6a162fa2dbeced
4  0001fd6190edaaf884bcaf3d49edf079  316a104623542e4d75189bb372bc5f8d
```



Calculate the cumulative sales per month for each year.

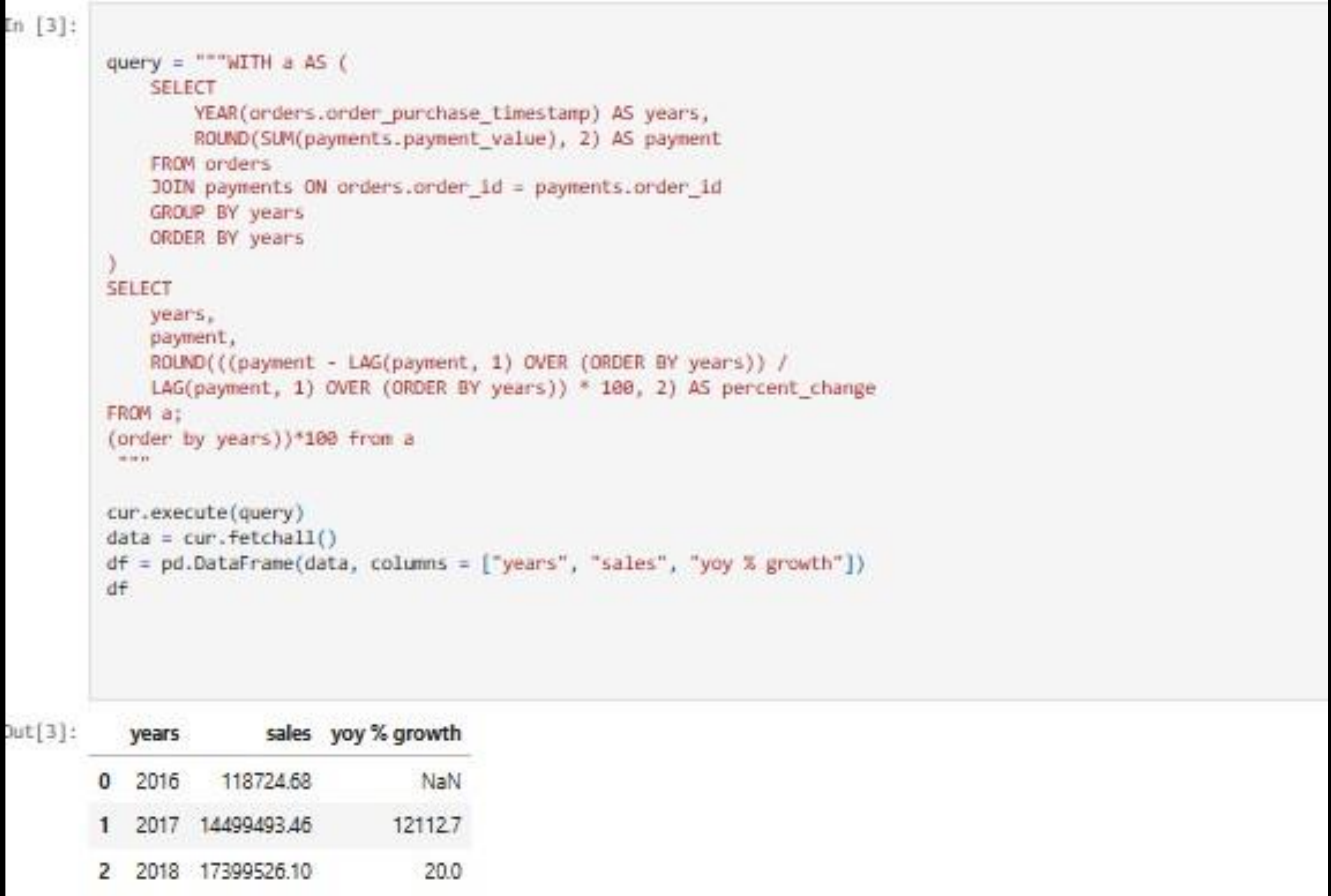
[25]:

```
query =""" select years,months, payment, sum(payment)
over(order by years, months) cumulates_sales from

(select year(orders.order_purchase_timestamp) as years,
month(orders.order_purchase_timestamp) as months,
round(sum(payments.payment_value),2) as payment 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)
df
```

t[25]:

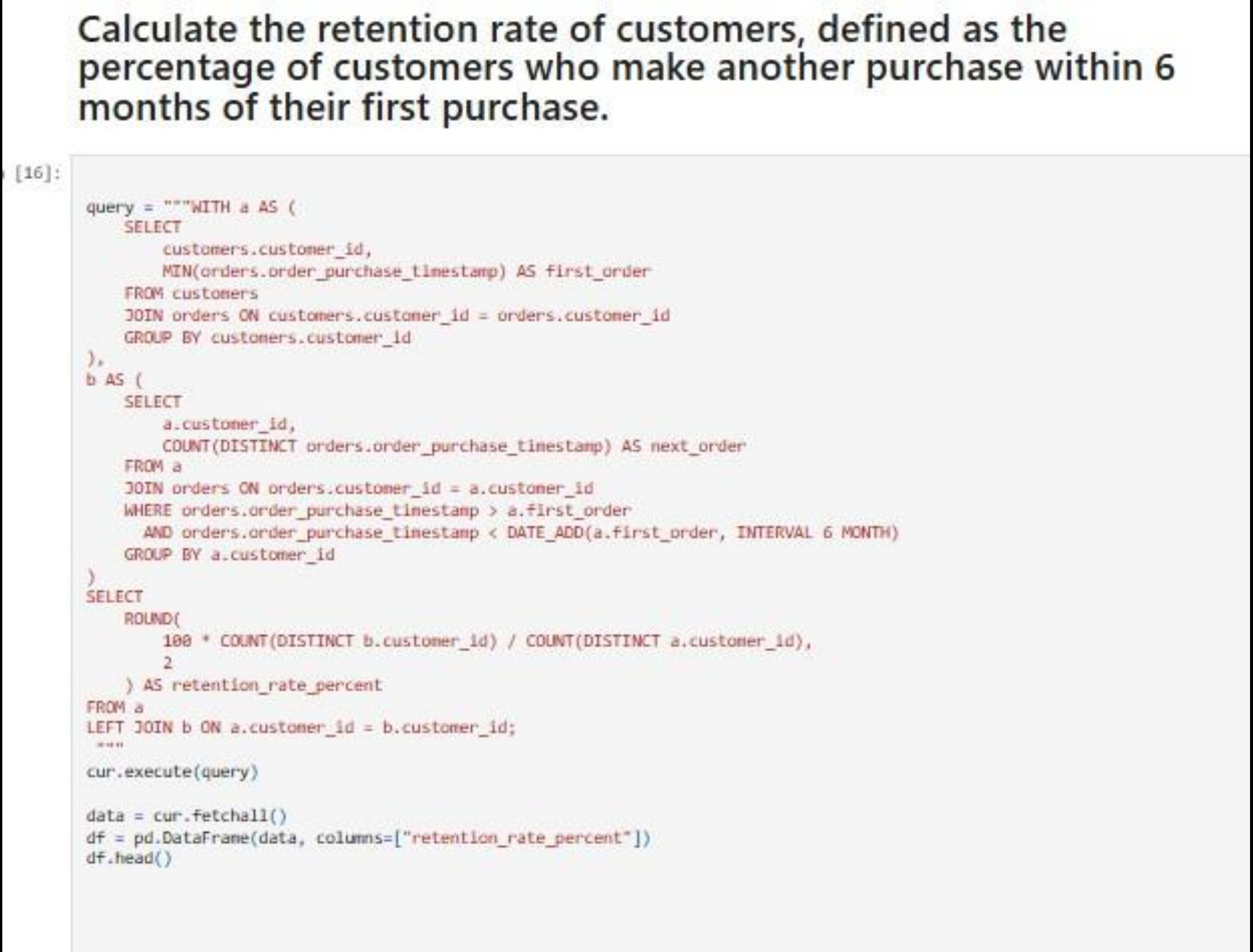
```
       0   1           2            3
 0  2016   9      504.48       504.48
 1  2016  10   118180.96    118685.44
 2  2016  12       39.24    118724.68
 3  2017   1   276976.08    395700.76
 4  2017   2   583816.02    979516.78
 5  2017   3   899727.20   1879243.98
 6  2017   4   835576.06   2714820.04
 7  2017   5  1185837.64   3900657.68
 8  2017   6  1022552.76   4923210.44
 9  2017   7  1184765.84   6107976.28
10  2017   8  1348792.64   7456768.92
11  2017   9  1455524.90   8912293.82
12  2017  10  1559355.76  10471649.58
```

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



```
In [3]:

query = """WITH a AS (
    SELECT
        YEAR(orders.order_purchase_timestamp) AS years,
        ROUND(SUM(payments.payment_value), 2) AS payment
    FROM orders
    JOIN payments ON orders.order_id = payments.order_id
    GROUP BY years
    ORDER BY years
)
SELECT
    years,
    payment,
    ROUND(((payment - LAG(payment, 1) OVER (ORDER BY years)) /
    LAG(payment, 1) OVER (ORDER BY years)) * 100, 2) AS percent_change
FROM a;
(order by years))*100 from a
 """

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns = ["years", "sales", "yoy % growth"])
df
```

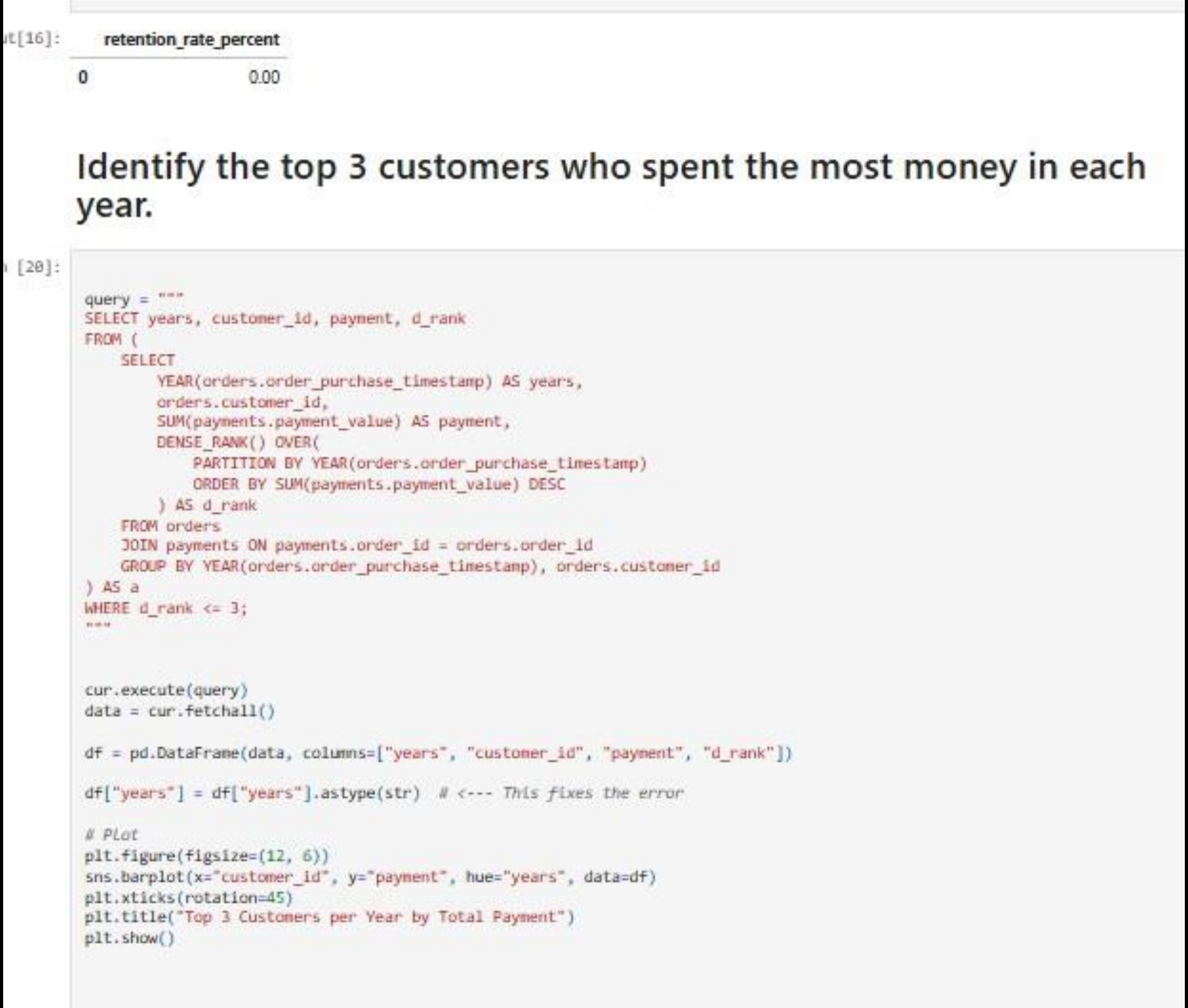
```
Out[3]:
   years        sales  yoy % growth
0   2016    118724.68           NaN
1   2017  14499493.46       12112.7
2   2018  17399526.10          20.0
```



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

[16]:

```
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 orders.customer_id = a.customer_id
    WHERE orders.order_purchase_timestamp > a.first_order
      AND orders.order_purchase_timestamp < DATE_ADD(a.first_order, INTERVAL 6 MONTH)
    GROUP BY a.customer_id
)
SELECT
    ROUND(
        100 * COUNT(DISTINCT b.customer_id) / COUNT(DISTINCT a.customer_id),
        2
    ) AS retention_rate_percent
FROM a
LEFT JOIN b ON a.customer_id = b.customer_id;
 """
cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns=["retention_rate_percent"])
df.head()
```



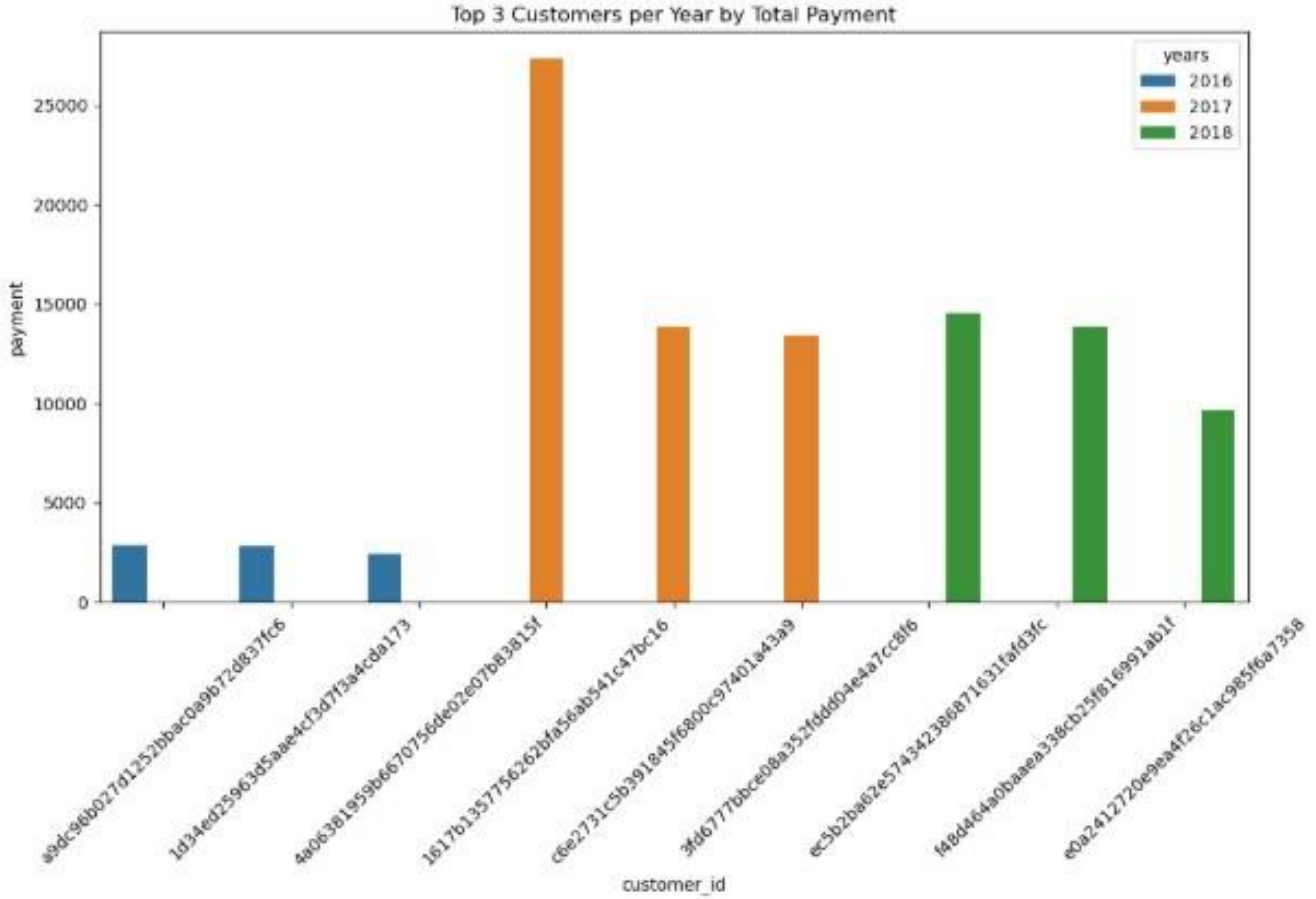
ut[16]:

	retention_rate_percent
0	0.00

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

[20]:

```
query = """
SELECT years, customer_id, payment, d_rank
FROM (
    SELECT
        YEAR(orders.order_purchase_timestamp) AS years,
        orders.customer_id,
        SUM(payments.payment_value) AS payment,
        DENSE_RANK() OVER(
            PARTITION BY YEAR(orders.order_purchase_timestamp)
            ORDER BY SUM(payments.payment_value) DESC
        ) AS d_rank
    FROM orders
    JOIN payments ON payments.order_id = orders.order_id
    GROUP BY YEAR(orders.order_purchase_timestamp), orders.customer_id
) AS a
WHERE d_rank <= 3;
"""


cur.execute(query)
data = cur.fetchall()

df = pd.DataFrame(data, columns=["years", "customer_id", "payment", "d_rank"])

df["years"] = df["years"].astype(str)  # <--- This fixes the error

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(x="customer_id", y="payment", hue="years", data=df)
plt.xticks(rotation=45)
plt.title("Top 3 Customers per Year by Total Payment")
plt.show()
```


Top 3 Customers per Year by Total Payment
years
2016
2017
2018
payment
25000
20000
15000
10000
5000
0
a9dc96b027d1252bbac0a9b72d837fc6
1d34ed25963d5aae4cf3d7f3a4cda173
4a06381959b6670756de02e07b83815f
1617b1357756262bfa56ab541c47bc16
c6e2731c5b391845f6800c97401a43a9
3fd6777bbce08a352fddd04e4a7cc8f6
ec5b2ba62e574342386871631fafd3fc
f48d464a0baaea338cb25f816991ab1f
e0a2412720e9ea4f26c1ac985f6a7358
customer_id

n [ ]: