E-commerce

Sales Performance Analysis using PYTHON and SQL

Introduction

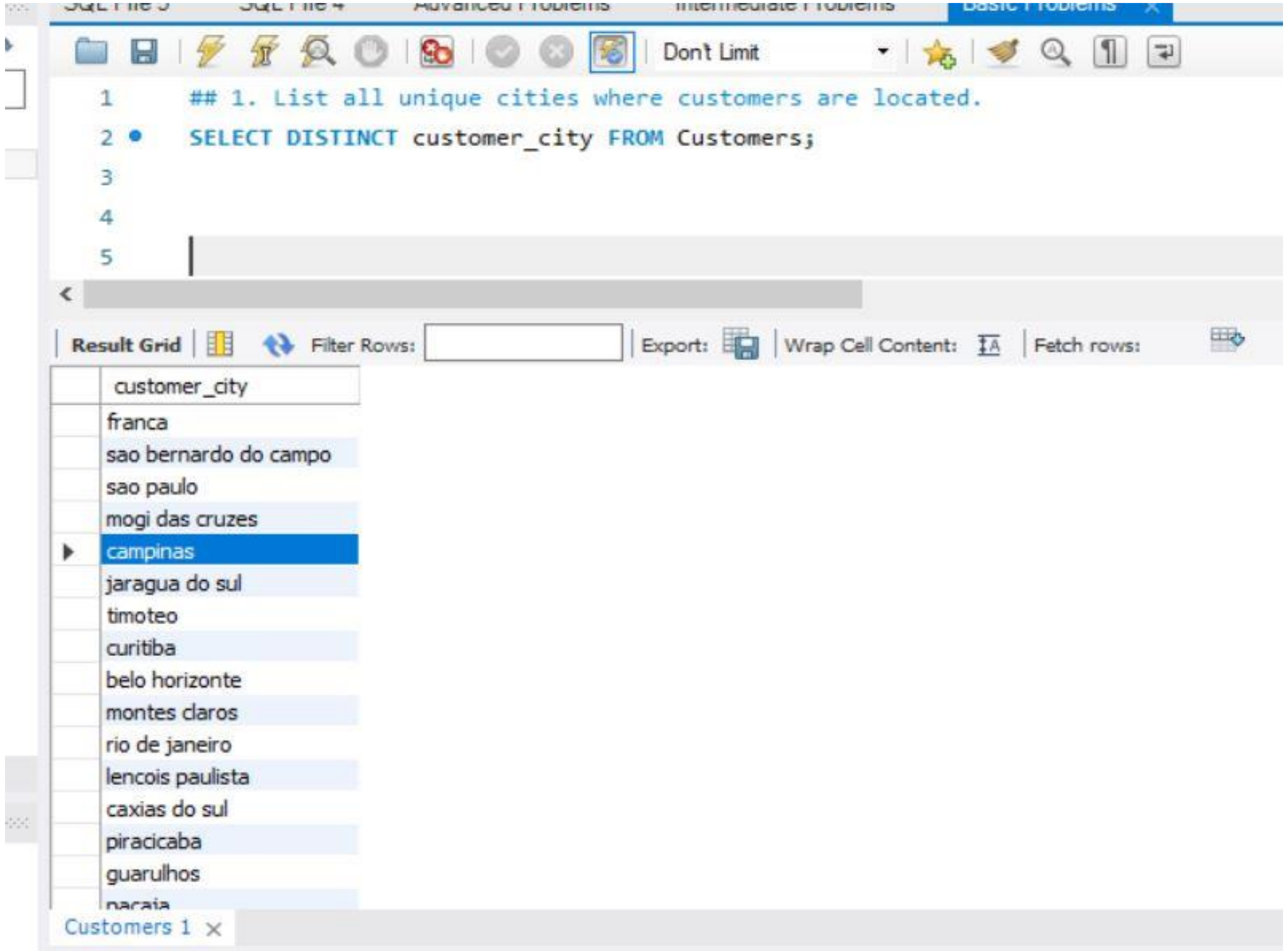
Our company is a rapidly growing e-commerce platform serving customers across Brazil, with a presence in over 4,000 cities and achieving 20% year-over-year growth in 2018. We offer a wide range of products—from Health & Beauty to Electronics—connecting customers with sellers through a user-friendly, installment-friendly shopping experience.

While we've successfully expanded our reach and built a strong customer acquisition engine, our next phase of growth is focused on deepening customer relationships and diversifying our operations. By investing in loyalty programs, geographic expansion beyond São Paulo, and a broader, more inclusive seller network, we aim to transform one-time buyers into repeat customers and create a more resilient, sustainable business model.

With strong market potential and a clear roadmap, we are positioned to become a leading player in Brazil's e-commerce ecosystem.

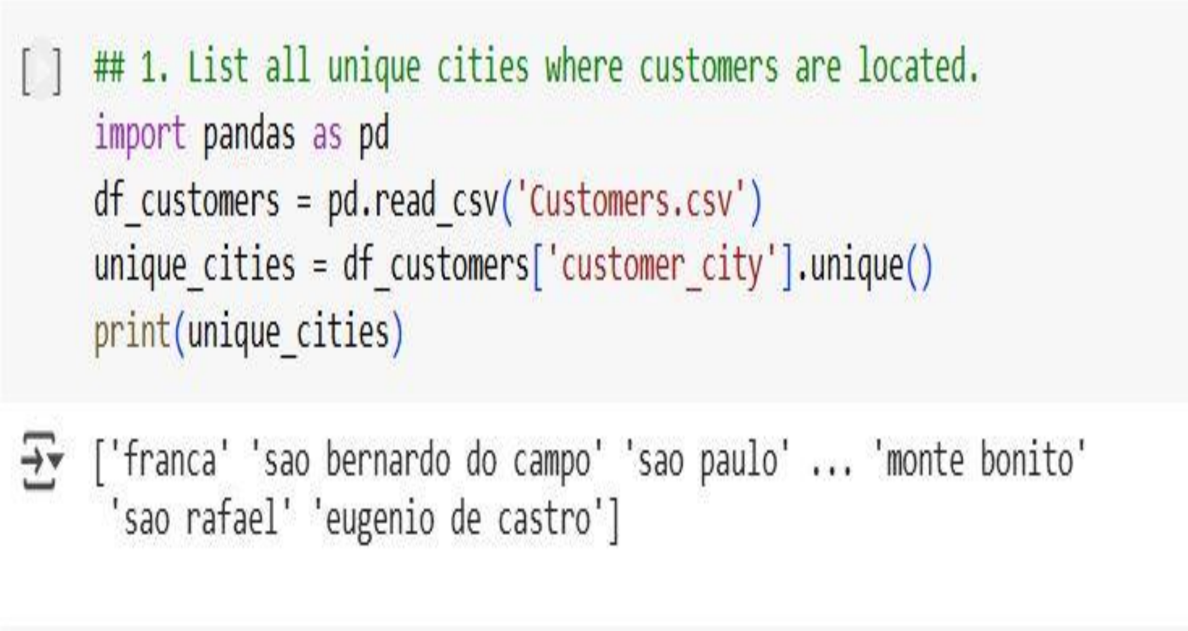
BASIC PROBLEMS

1. List all unique cities where customers are located.



```sql
## 1. List all unique cities where customers are located.
SELECT DISTINCT customer_city FROM Customers;
```

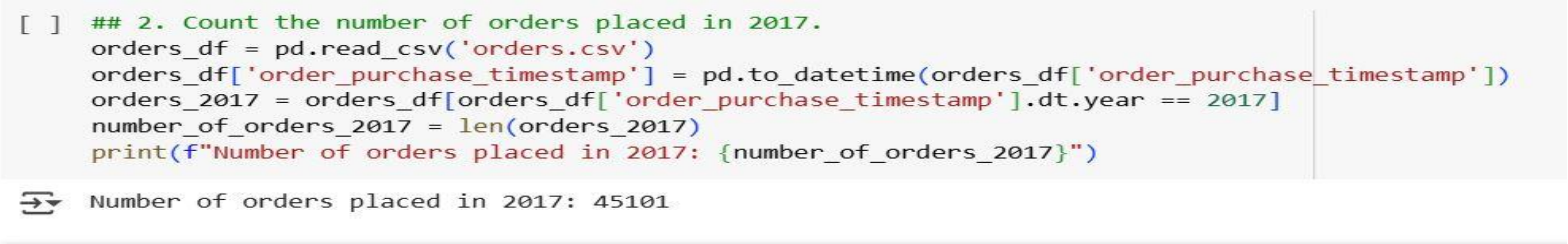
customer_city
franca
sao bernardo do campo
sao paulo
mogi das cruzes
campinas
jaragua do sul
timoteo
curitiba
belo horizonte
montes claros
rio de janeiro
lencois paulista
caxias do sul
piracicaba
guarulhos
pacaja



```python
## 1. List all unique cities where customers are located.
import pandas as pd
df_customers = pd.read_csv('Customers.csv')
unique_cities = df_customers['customer_city'].unique()
print(unique_cities)
```

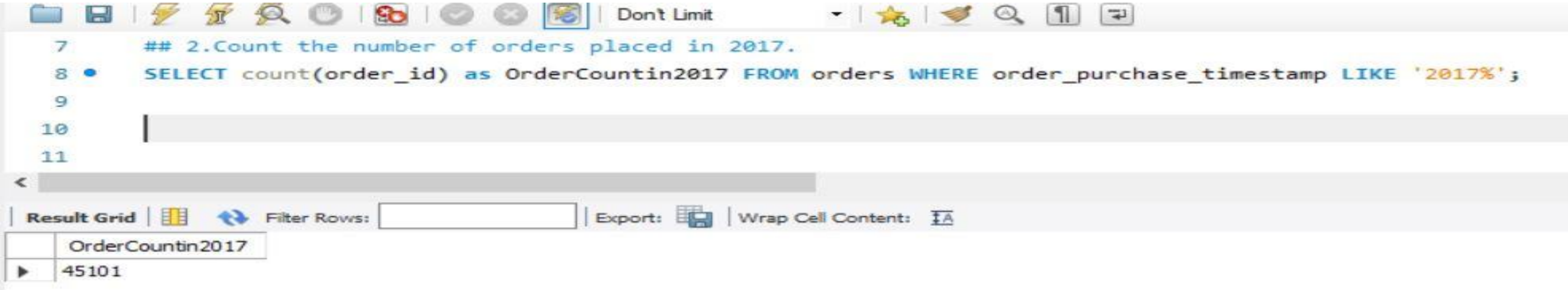
```
['franca' 'sao bernardo do campo' 'sao paulo' ... 'monte bonito'
 'sao rafael' 'eugenio de castro']
```

2. Count the number of orders placed in 2017.



```python
## 2. Count the number of orders placed in 2017.
orders_df = pd.read_csv('orders.csv')
orders_df['order_purchase_timestamp'] = pd.to_datetime(orders_df['order_purchase_timestamp'])
orders_2017 = orders_df[orders_df['order_purchase_timestamp'].dt.year == 2017]
number_of_orders_2017 = len(orders_2017)
print(f"Number of orders placed in 2017: {number_of_orders_2017}")
```

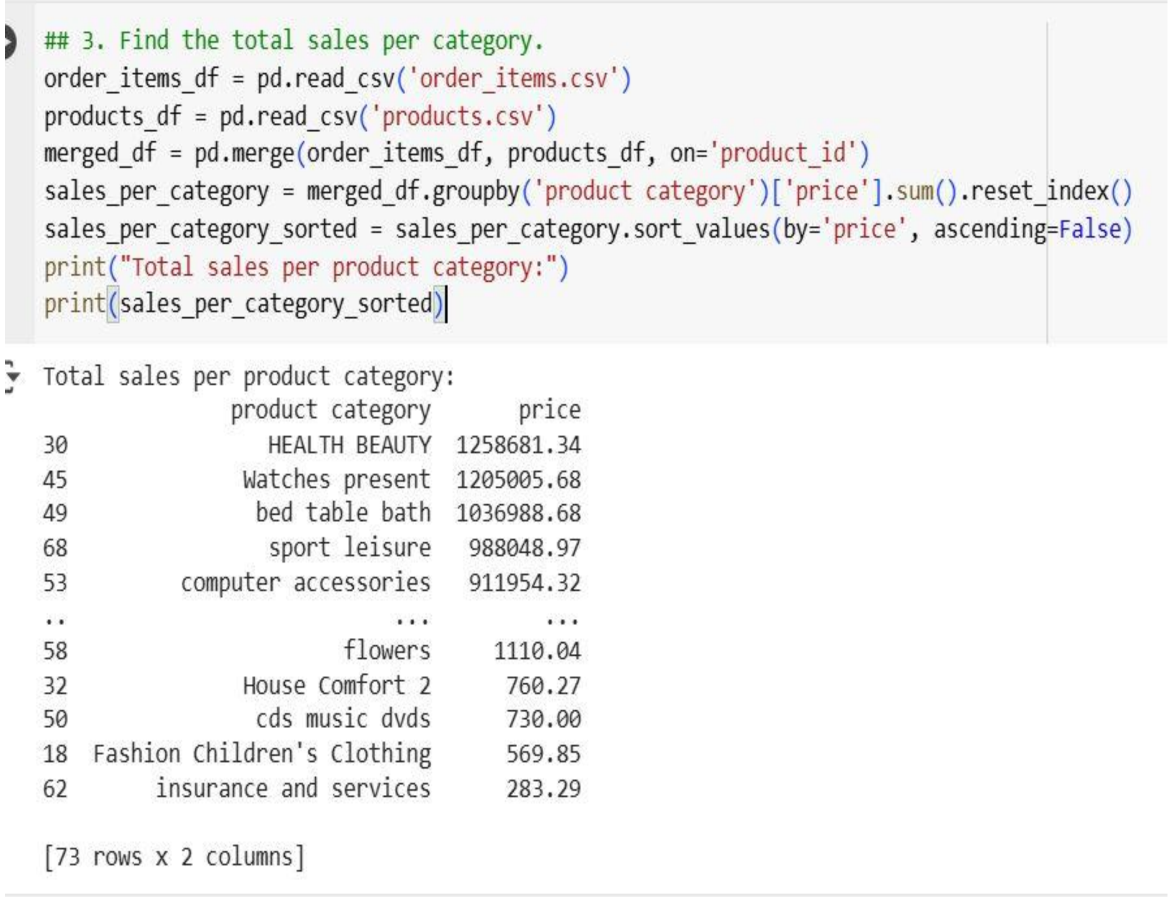
```
Number of orders placed in 2017: 45101
```



```sql
## 2.Count the number of orders placed in 2017.
SELECT count(order_id) as OrderCountin2017 FROM orders WHERE order_purchase_timestamp LIKE '2017%';
```

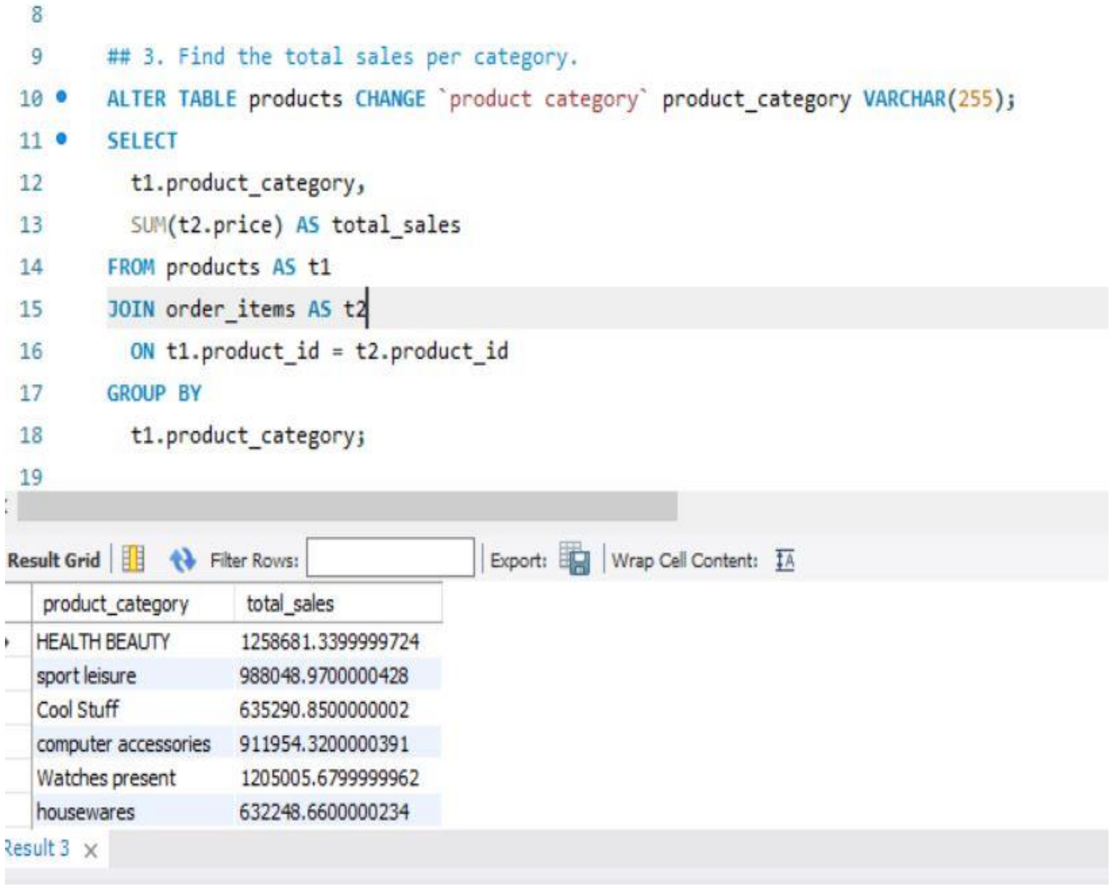
OrderCountin2017
45101

3. Find the total sales per category.



```python
## 3. Find the total sales per category.
order_items_df = pd.read_csv('order_items.csv')
products_df = pd.read_csv('products.csv')
merged_df = pd.merge(order_items_df, products_df, on='product_id')
sales_per_category = merged_df.groupby('product category')['price'].sum().reset_index()
sales_per_category_sorted = sales_per_category.sort_values(by='price', ascending=False)
print("Total sales per product category:")
print(sales_per_category_sorted)
```

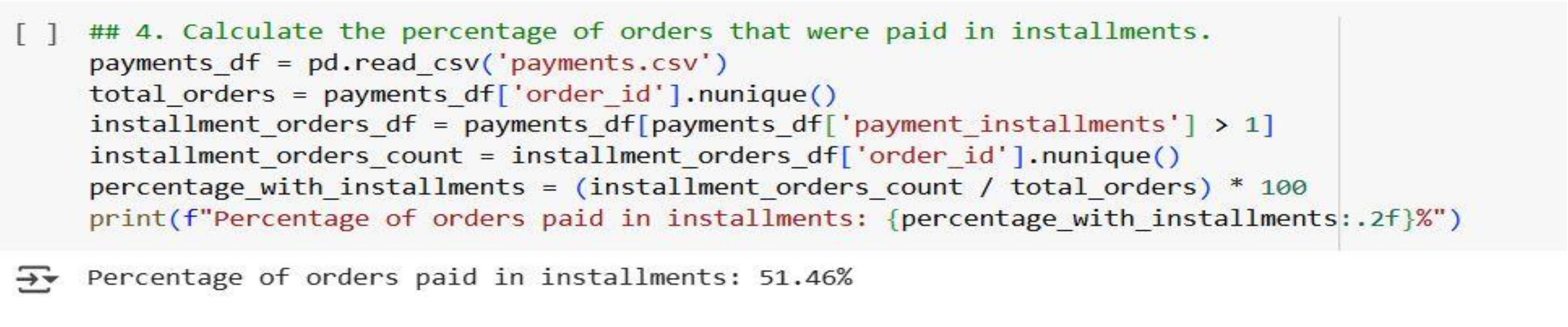
```
Total sales per product category:
                 product category       price
30                  HEALTH BEAUTY  1258681.34
45                Watches present  1205005.68
49                 bed table bath  1036988.68
68                  sport leisure   988048.97
53           computer accessories   911954.32
..                            ...         ...
58                        flowers     1110.04
32                House Comfort 2      760.27
50                 cds music dvds      730.00
18    Fashion Children's Clothing      569.85
62          insurance and services      283.29

[73 rows x 2 columns]
```



```sql

## 3. Find the total sales per category.
ALTER TABLE products CHANGE `product category` product_category VARCHAR(255);
SELECT
  t1.product_category,
  SUM(t2.price) AS total_sales
FROM products AS t1
JOIN order_items AS t2
  ON t1.product_id = t2.product_id
GROUP BY
  t1.product_category;

```

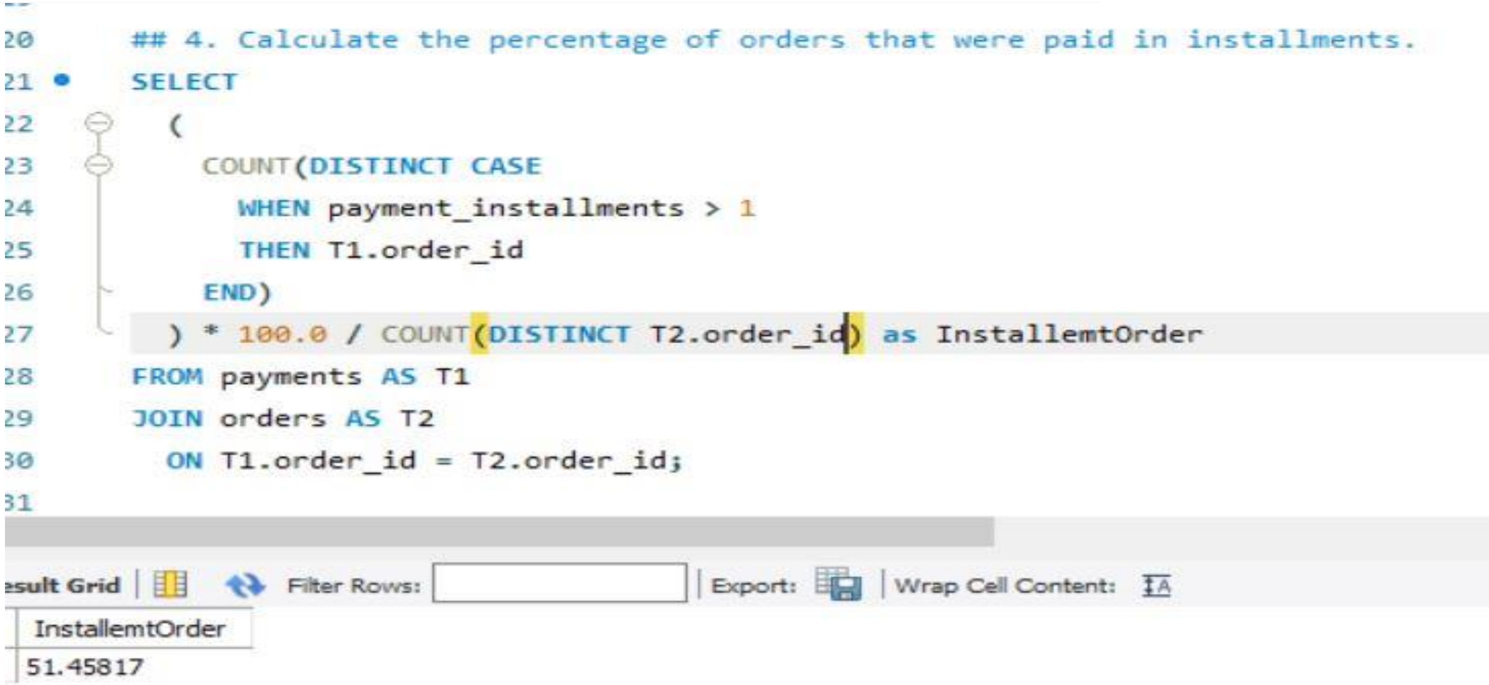
product_category	total_sales
HEALTH BEAUTY	1258681.3399999724
sport leisure	988048.9700000428
Cool Stuff	635290.8500000002
computer accessories	911954.3200000391
Watches present	1205005.6799999962
housewares	632248.6600000234

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



```
## 4. Calculate the percentage of orders that were paid in installments.
payments_df = pd.read_csv('payments.csv')
total_orders = payments_df['order_id'].nunique()
installment_orders_df = payments_df[payments_df['payment_installments'] > 1]
installment_orders_count = installment_orders_df['order_id'].nunique()
percentage_with_installments = (installment_orders_count / total_orders) * 100
print(f"Percentage of orders paid in installments: {percentage_with_installments:.2f}%")
```

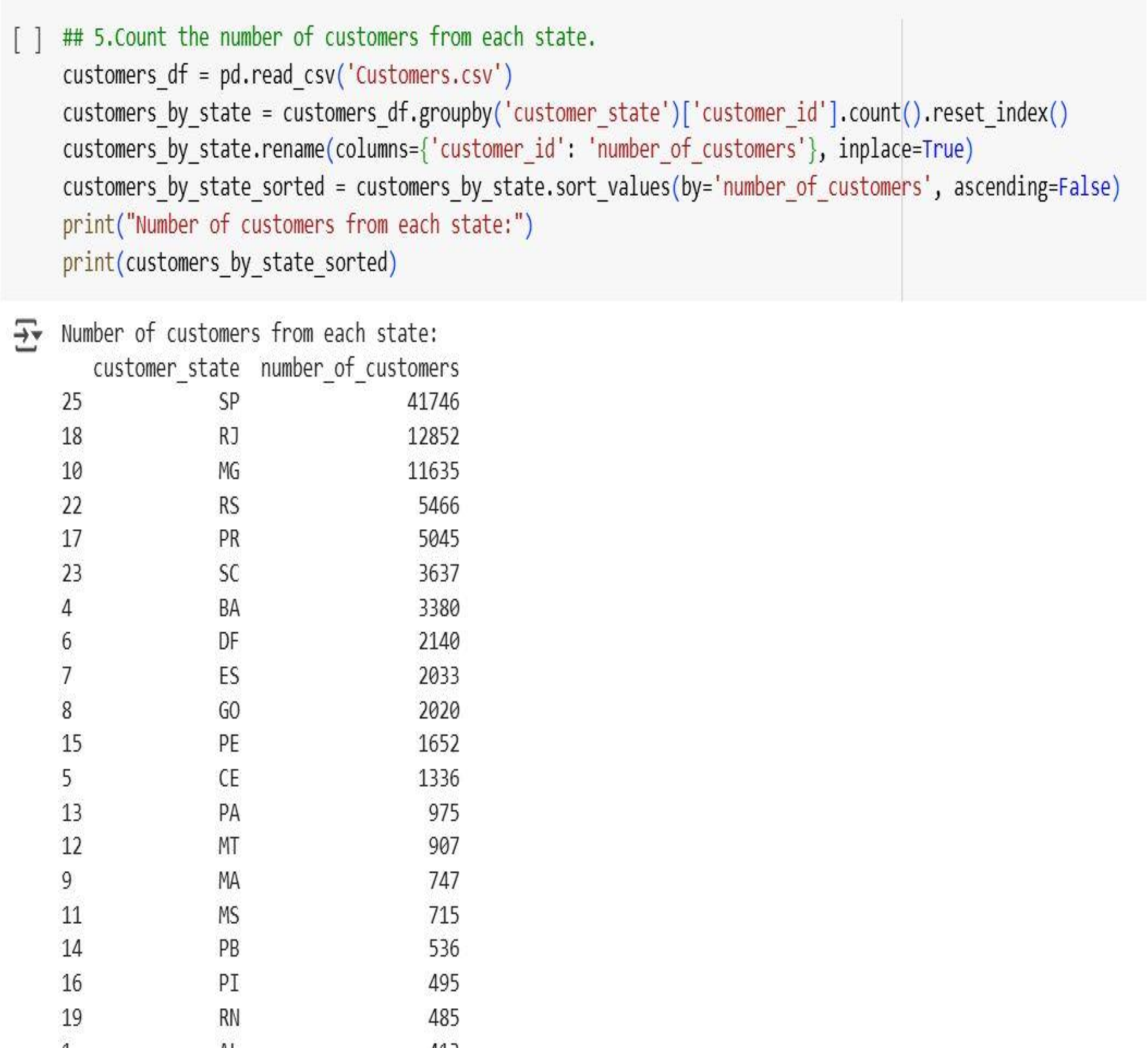
```
Percentage of orders paid in installments: 51.46%
```



```
## 4. Calculate the percentage of orders that were paid in installments.
SELECT
  (
    COUNT(DISTINCT CASE
      WHEN payment_installments > 1
      THEN T1.order_id
    END)
  ) * 100.0 / COUNT(DISTINCT T2.order_id) as InstallemtOrder
FROM payments AS T1
JOIN orders AS T2
  ON T1.order_id = T2.order_id;
```

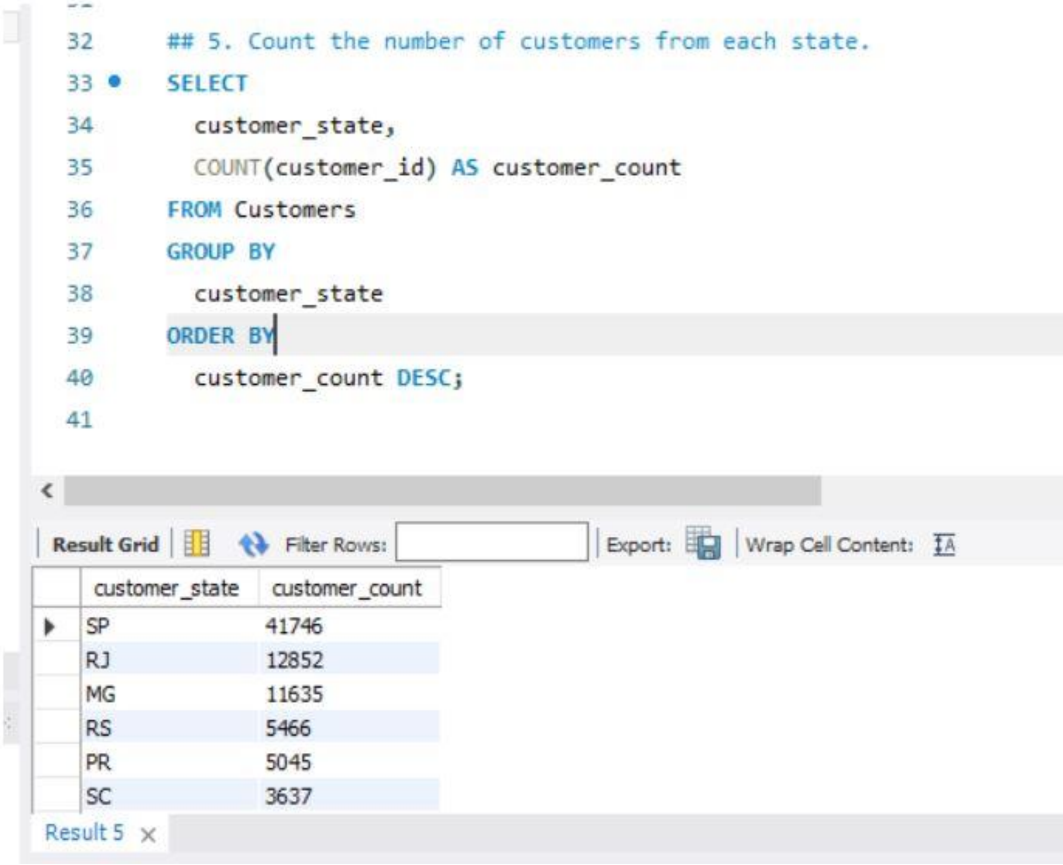
InstallemtOrder
51.45817

5. Count the number of customers from each state.



```
## 5.Count the number of customers from each state.
customers_df = pd.read_csv('Customers.csv')
customers_by_state = customers_df.groupby('customer_state')['customer_id'].count().reset_index()
customers_by_state.rename(columns={'customer_id': 'number_of_customers'}, inplace=True)
customers_by_state_sorted = customers_by_state.sort_values(by='number_of_customers', ascending=False)
print("Number of customers from each state:")
print(customers_by_state_sorted)
```

```
Number of customers from each state:
   customer_state  number_of_customers
25             SP                41746
18             RJ                12852
10             MG                11635
22             RS                 5466
17             PR                 5045
23             SC                 3637
4              BA                 3380
6              DF                 2140
7              ES                 2033
8              GO                 2020
15             PE                 1652
5              CE                 1336
13             PA                  975
12             MT                  907
9              MA                  747
11             MS                  715
14             PB                  536
16             PI                  495
19             RN                  485
```

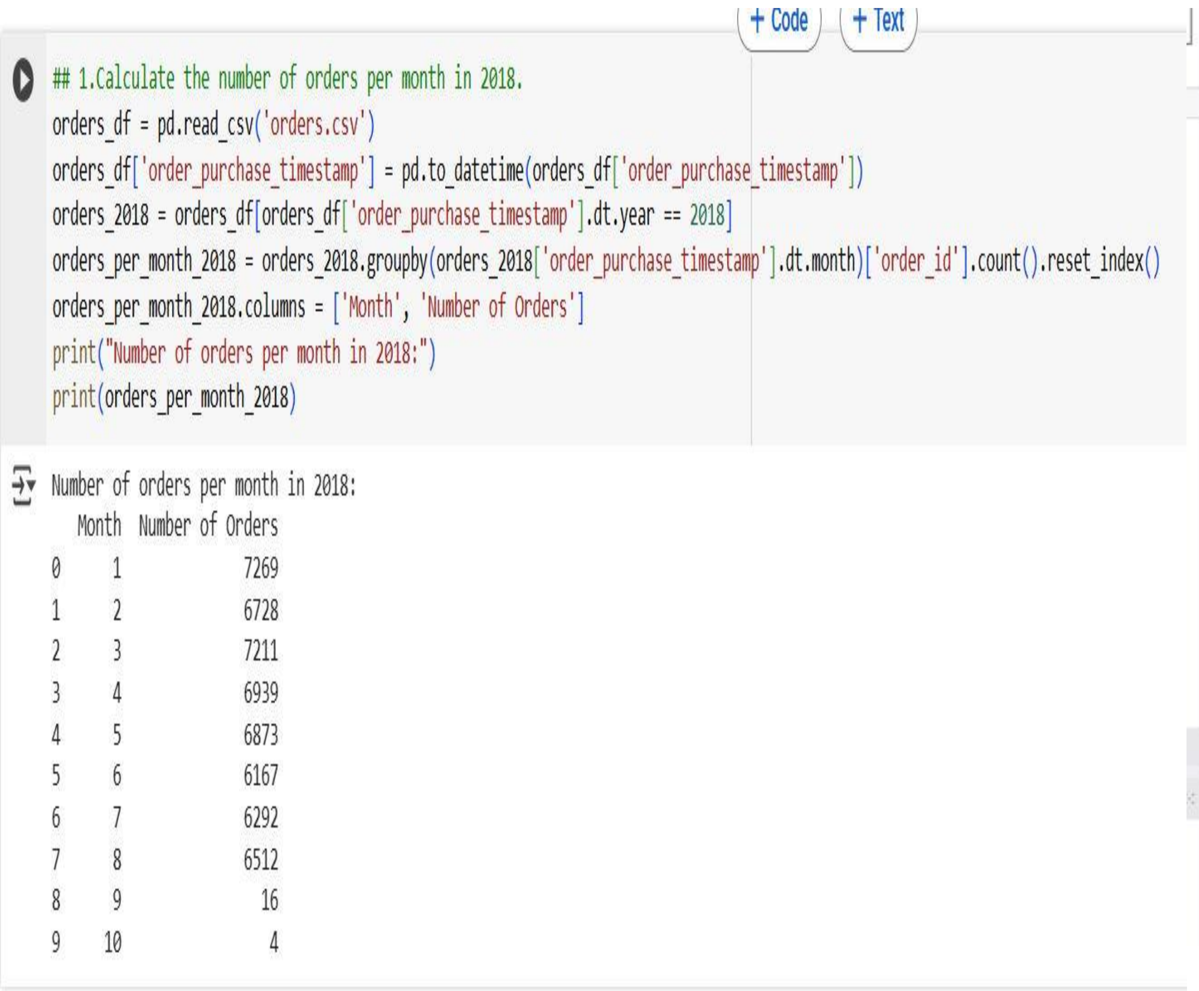


```sql
## 5. Count the number of customers from each state.
SELECT
  customer_state,
  COUNT(customer_id) AS customer_count
FROM Customers
GROUP BY
  customer_state
ORDER BY
  customer_count DESC;
```

customer_state	customer_count
SP	41746
RJ	12852
MG	11635
RS	5466
PR	5045
SC	3637

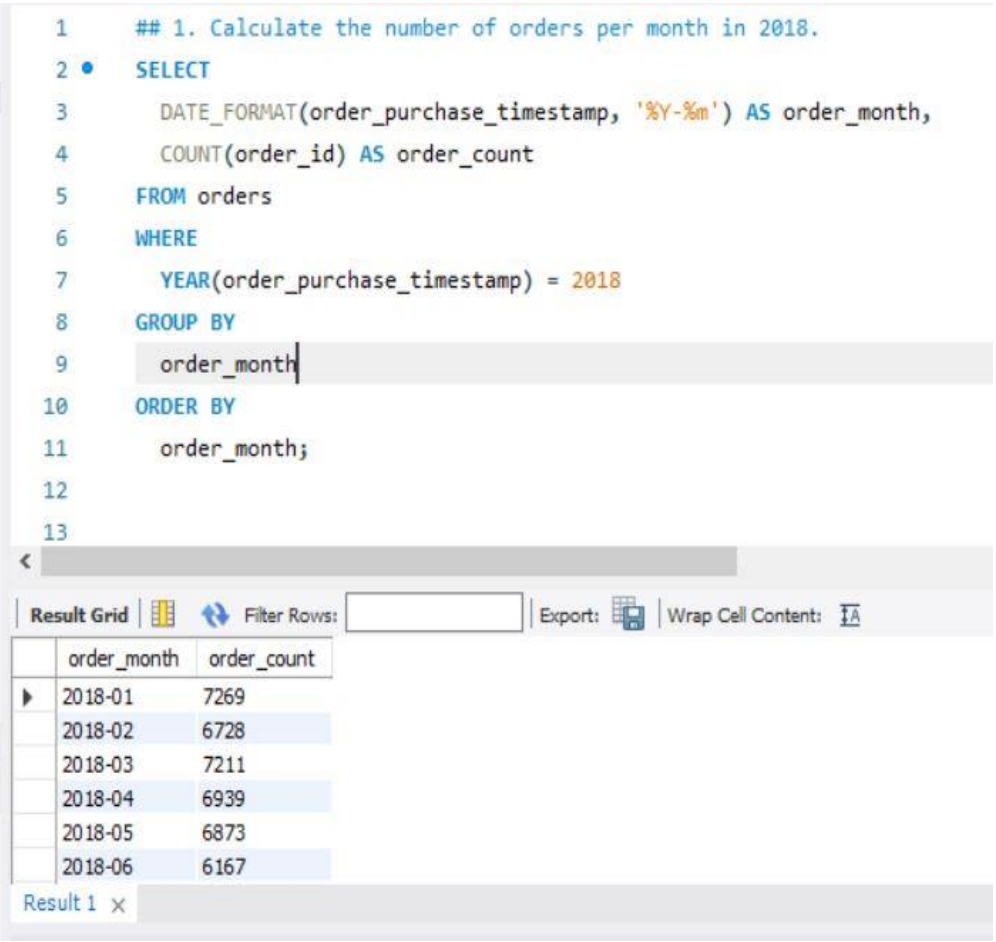
INTERMEDIATE PROBLEMS

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



```python
## 1.Calculate the number of orders per month in 2018.
orders_df = pd.read_csv('orders.csv')
orders_df['order_purchase_timestamp'] = pd.to_datetime(orders_df['order_purchase_timestamp'])
orders_2018 = orders_df[orders_df['order_purchase_timestamp'].dt.year == 2018]
orders_per_month_2018 = orders_2018.groupby(orders_2018['order_purchase_timestamp'].dt.month)['order_id'].count().reset_index()
orders_per_month_2018.columns = ['Month', 'Number of Orders']
print("Number of orders per month in 2018:")
print(orders_per_month_2018)
```

```
Number of orders per month in 2018:
   Month  Number of Orders
0      1              7269
1      2              6728
2      3              7211
3      4              6939
4      5              6873
5      6              6167
6      7              6292
7      8              6512
8      9                16
9     10                 4
```



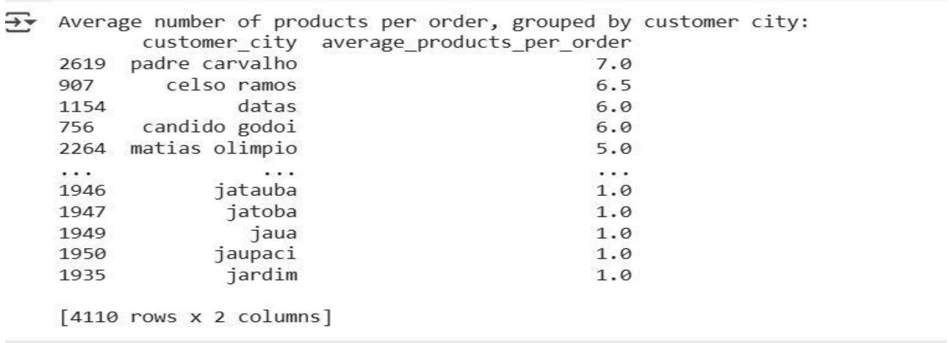
```sql
## 1. Calculate the number of orders per month in 2018.
SELECT
  DATE_FORMAT(order_purchase_timestamp, '%Y-%m') AS order_month,
  COUNT(order_id) AS order_count
FROM orders
WHERE
  YEAR(order_purchase_timestamp) = 2018
GROUP BY
  order_month
ORDER BY
  order_month;
```

order_month	order_count
2018-01	7269
2018-02	6728
2018-03	7211
2018-04	6939
2018-05	6873
2018-06	6167

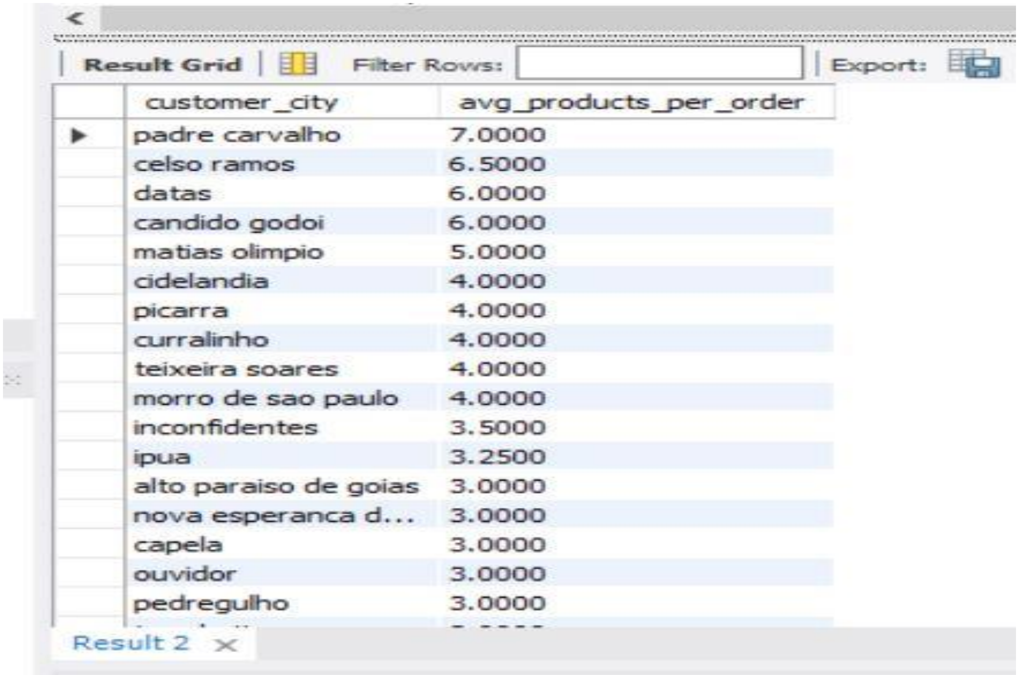
2. Find the average number of products per order. grouped by customer city.

Python code:

```
customers_df = pd.read_csv('Customers.csv')

orders_df = pd.read_csv('orders.csv')

order_items_df = pd.read_csv('order_items.csv')

orders_with_customers = pd.merge(orders_df, customers_df, on='customer_id')

merged_df = pd.merge(orders_with_customers, order_items_df, on='order_id')

products_per_order = merged_df.groupby(['customer_city', 'order_id']).size().reset_index(name='number_of_products')

average_products_per_city = products_per_order.groupby('customer_city')['number_of_products'].mean().reset_index()

average_products_per_city.rename(columns={'number_of_products': 'average_products_per_order'}, inplace=True)

average_products_per_city_sorted = average_products_per_city.sort_values(by='average_products_per_order', ascending=False)

print("Average number of products per order, grouped by customer city:")

print(average_products_per_city_sorted)
```



```
Average number of products per order, grouped by customer city:
        customer_city  average_products_per_order
2619  padre carvalho                         7.0
907      celso ramos                         6.5
1154           datas                         6.0
756    candido godoi                         6.0
2264  matias olimpio                         5.0
...              ...                         ...
1946         jatauba                         1.0
1947          jatoba                         1.0
1949            jaua                         1.0
1950         jaupaci                         1.0
1935          jardim                         1.0

[4110 rows x 2 columns]
```



customer_city	avg_products_per_order
padre carvalho	7.0000
celso ramos	6.5000
datas	6.0000
candido godoi	6.0000
matias olimpio	5.0000
cidelandia	4.0000
picarra	4.0000
curralinho	4.0000
teixeira soares	4.0000
morro de sao paulo	4.0000
inconfidentes	3.5000
ipua	3.2500
alto paraiso de goias	3.0000
nova esperanca d...	3.0000
capela	3.0000
ouvidor	3.0000
pedregulho	3.0000

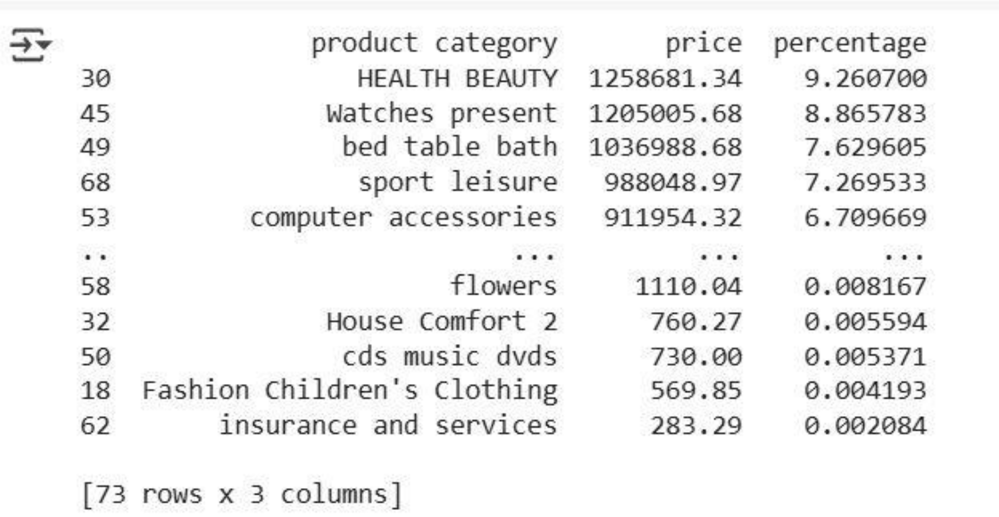
SQL Code :

```
WITH ProductsPerOrder AS ( SELECT order_id, COUNT(product_id) AS product_count
 FROM order_items GROUP BY order_id ) SELECT T3.customer_city,
 AVG(T1.product_count) AS avg_products_per_order FROM ProductsPerOrder AS T1
 JOIN orders AS T2 ON T1.order_id = T2.order_id JOIN Customers AS T3 ON
 T2.customer_id = T3.customer_id GROUP BY T3.customer_city ORDER BY
 avg_products_per_order DESC;
```

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

Python code :

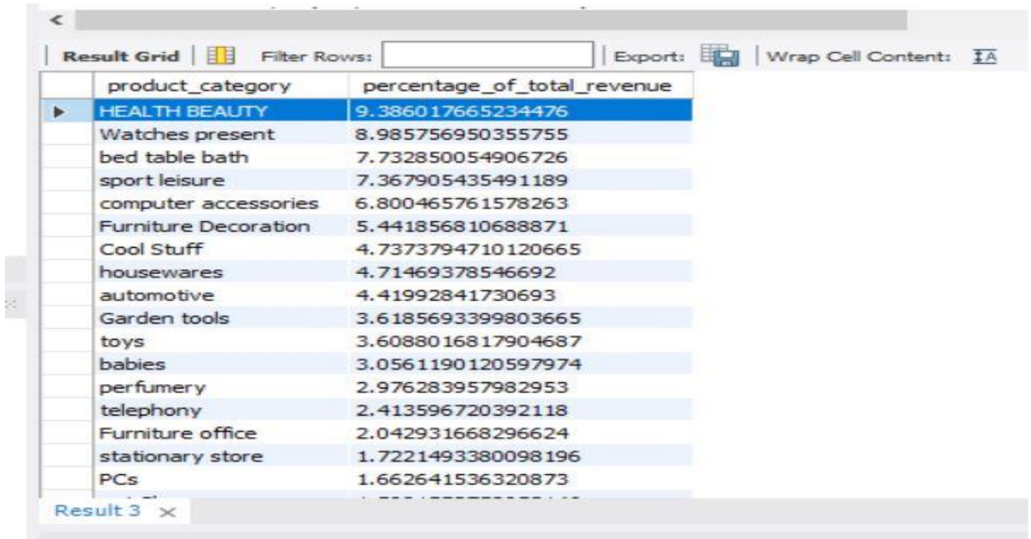
```
order_items_df = pd.read_csv('order_items.csv')

products_df = pd.read_csv('products.csv')

merged_df = pd.merge(order_items_df, products_df, on='product_id')

total_revenue = merged_df['price'].sum()

revenue_per_category = merged_df.groupby('product
category')['price'].sum().reset_index()

revenue_per_category['percentage'] = (revenue_per_category['price'] / total_revenue)
* 100

revenue_percentage_sorted = revenue_per_category.sort_values(by='percentage',
ascending=False)

print(revenue_percentage_sorted)
```



```
               product category       price  percentage
30                 HEALTH BEAUTY  1258681.34    9.260700
45               Watches present  1205005.68    8.865783
49                bed table bath  1036988.68    7.629605
68                 sport leisure   988048.97    7.269533
53          computer accessories   911954.32    6.709669
..                           ...         ...         ...
58                       flowers     1110.04    0.008167
32              House Comfort 2       760.27    0.005594
50                cds music dvds      730.00    0.005371
18  Fashion Children's Clothing      569.85    0.004193
62        insurance and services      283.29    0.002084

[73 rows x 3 columns]
```

SQL code:

```
category. WITH CategoryRevenue AS ( SELECT T1.product_category, SUM(T2.price) AS
 revenue FROM products AS T1 JOIN order_items AS T2 ON T1.product_id =
 T2.product_id GROUP BY T1.product_category ) SELECT product_category, (revenue *
 100.0) / ( SELECT SUM(revenue) FROM CategoryRevenue ) AS
 percentage_of_total_revenue FROM CategoryRevenue ORDER BY
 percentage_of_total_revenue DESC;
```



product_category	percentage_of_total_revenue
HEALTH BEAUTY	9.386017665234476
Watches present	8.985756950355755
bed table bath	7.732850054906726
sport leisure	7.367905435491189
computer accessories	6.800465761578263
Furniture Decoration	5.441856810688871
Cool Stuff	4.7373794710120665
housewares	4.71469378546692
automotive	4.41992841730693
Garden tools	3.6185693399803665
toys	3.6088016817904687
babies	3.0561190120597974
perfumery	2.976283957982953
telephony	2.413596720392118
Furniture office	2.042931668296624
stationary store	1.7221493380098196
PCs	1.662641536320873

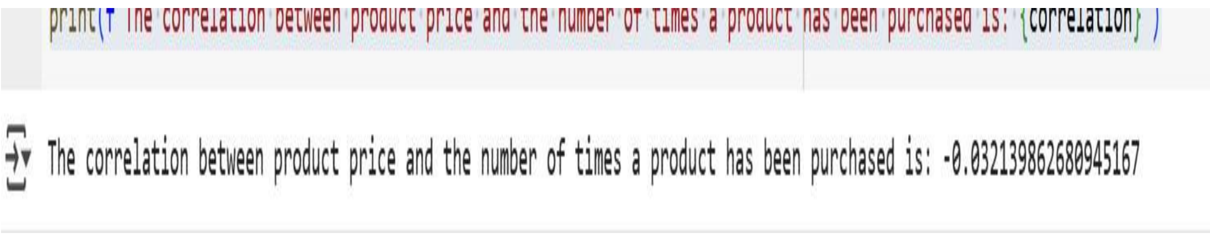
4. Identify the correlation between product price and the number of times a product has been purchased.

Python Code:

```
import pandas as pd
order_items_df = pd.read_csv('order_items.csv')
products_df = pd.read_csv('products.csv')
merged_df = pd.merge(order_items_df, products_df, on='product_id', how='inner')
product_summary =
 merged_df.groupby('product_id').agg(number_of_purchases=('order_id’,
 'count'),average_price=('price', 'mean')).reset_index()
correlation =
 product_summary['number_of_purchases'].corr(product_summary['average_price'])
print(f"The correlation between product price and the number of times a product has been
 purchased is: {correlation}")
```

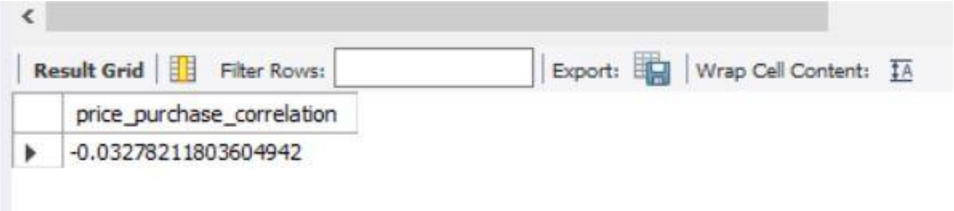
SQL Code:

```
WITH ProductsPerOrder AS (  SELECT    order_id,    COUNT(product_id) AS
 product_count  FROM order_items  GROUP BY    order_id)SELECT  T3.customer_city,
 AVG(T1.product_count) AS avg_products_per_orderFROM ProductsPerOrder AS
 T1JOIN orders AS T2  ON T1.order_id = T2.order_idJOIN Customers AS T3  ON
 T2.customer_id = T3.customer_idGROUP BY  T3.customer_cityORDER BY
 avg_products_per_order DESC;
```



```
print(f"The correlation between product price and the number of times a product has been purchased is: {correlation}")
```

```
The correlation between product price and the number of times a product has been purchased is: -0.032139862680945167
```

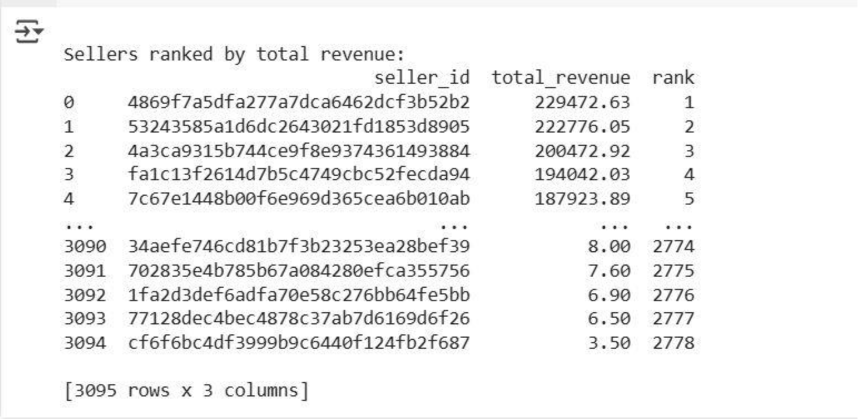


	price_purchase_correlation
▶	-0.03278211803604942

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

Python Code:

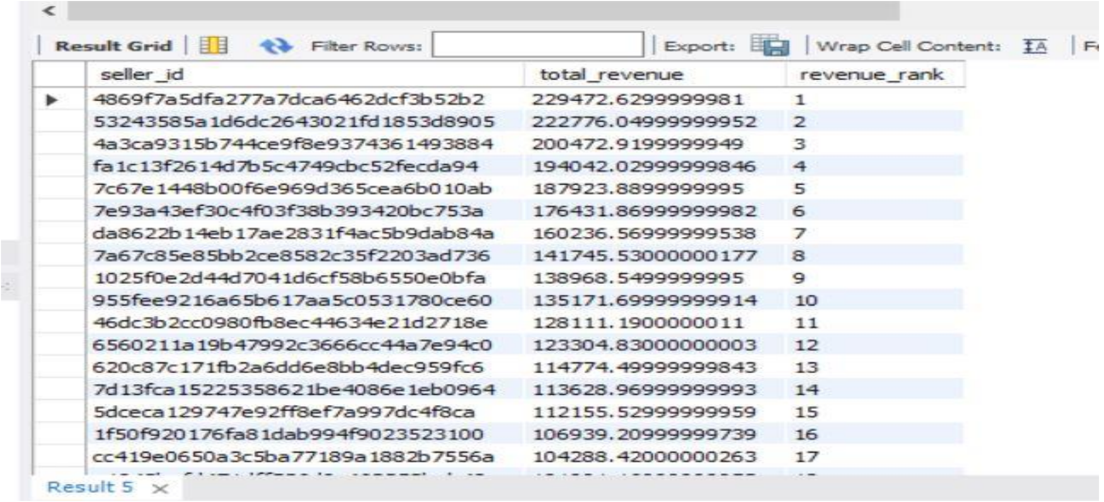
```
order_items_df = pd.read_csv('order_items.csv')
revenue_per_seller =
 order_items_df.groupby('seller_id')['price'].sum().reset_index()
revenue_per_seller.columns = ['seller_id', 'total_revenue']
ranked_sellers = revenue_per_seller.sort_values(by='total_revenue',
ascending=False)
ranked_sellers['rank’] =
 ranked_sellers['total_revenue'].rank(method='dense’,
 ascending=False).astype(int)
ranked_sellers = ranked_sellers.reset_index(drop=True)
print("\nSellers ranked by total revenue:")
print(ranked_sellers)
```



```
Sellers ranked by total revenue:
                             seller_id  total_revenue  rank
0     4869f7a5dfa277a7dca6462dcf3b52b2      229472.63     1
1     53243585a1d6dc2643021fd1853d8905      222776.05     2
2     4a3ca9315b744ce9f8e9374361493884      200472.92     3
3     fa1c13f2614d7b5c4749cbc52fecda94      194042.03     4
4     7c67e1448b00f6e969d365cea6b010ab      187923.89     5
...                                ...            ...   ...
3090  34aefe746cd81b7f3b23253ea28bef39           8.00  2774
3091  702835e4b785b67a084280efca355756           7.60  2775
3092  1fa2d3def6adfa70e58c276bb64fe5bb           6.90  2776
3093  77128dec4bec4878c37ab7d6169d6f26           6.50  2777
3094  cf6f6bc4df3999b9c6440f124fb2f687           3.50  2778

[3095 rows x 3 columns]
```

SQL Code:

```
SELECT T1.seller_id, SUM(T2.price) AS total_revenue, RANK() OVER (
ORDER BY SUM(T2.price)
 DESC ) AS revenue_rank FROM sellers AS T1 JOIN order_items AS T2
ON T1.seller_id =
 T2.seller_id GROUP BY T1.seller_id ORDER BY revenue_rank;
```



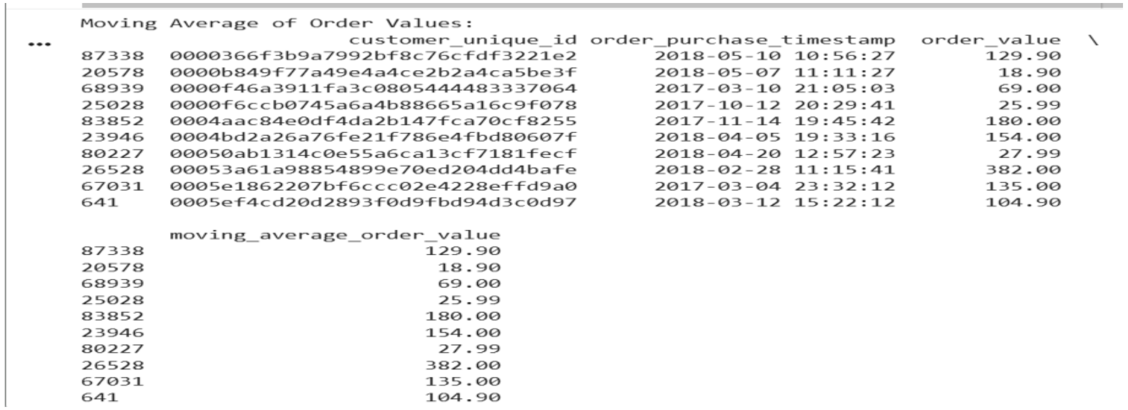
seller_id	total_revenue	revenue_rank
4869f7a5dfa277a7dca6462dcf3b52b2	229472.6299999981	1
53243585a1d6dc2643021fd1853d8905	222776.04999999952	2
4a3ca9315b744ce9f8e9374361493884	200472.9199999949	3
fa1c13f2614d7b5c4749cbc52fecda94	194042.02999999846	4
7c67e1448b00f6e969d365cea6b010ab	187923.8899999995	5
7e93a43ef30c4f03f38b393420bc753a	176431.86999999982	6
da8622b14eb17ae2831f4ac5b9dab84a	160236.56999999538	7
7a67c85e85bb2ce8582c35f2203ad736	141745.53000000177	8
1025f0e2d44d7041d6cf58b6550e0bfa	138968.5499999995	9
955fee9216a65b617aa5c0531780ce60	135171.69999999914	10
46dc3b2cc0980fb8ec44634e21d2718e	128111.1900000011	11
6560211a19b47992c3666cc44a7e94c0	123304.83000000003	12
620c87c171fb2a6dd6e8bb4dec959fc6	114774.49999999843	13
7d13fca15225358621be4086e1eb0964	113628.96999999993	14
5dceca129747e92ff8ef7a997dc4f8ca	112155.52999999959	15
1f50f920176fa81dab994f9023523100	106939.20999999739	16
cc419e0650a3c5ba77189a1882b7556a	104288.42000000263	17

ADVANCED PROBLEMS

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

Python Code:

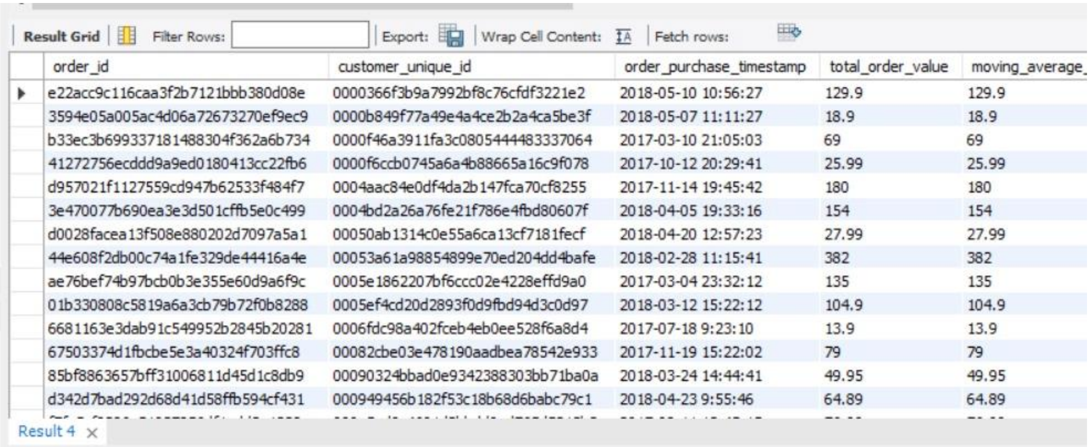
```
import pandas as pd

orders_df = pd.read_csv('orders.csv')

order_items_df = pd.read_csv('order_items.csv')

customers_df = pd.read_csv('Customers.csv')

order_value_df = pd.merge(order_items_df, orders_df, on='order_id', how='left')

order_value_df = order_value_df.groupby('order_id').agg(order_value=('price',
'sum'),order_purchase_timestamp=('order_purchase_timestamp', 'first')).reset_index()

order_value_df = pd.merge(order_value_df, orders_df[['order_id', 'customer_id']], on='order_id', how='left')

order_value_df = pd.merge(order_value_df, customers_df, on='customer_id', how='left')

order_value_df.dropna(subset=['customer_unique_id'], inplace=True)

order_value_df['order_purchase_timestamp'] = pd.to_datetime(order_value_df['order_purchase_timestamp'])

order_value_df.sort_values(by=['customer_unique_id', 'order_purchase_timestamp'], inplace=True)

order_value_df['moving_average_order_value'] =
order_value_df.groupby('customer_unique_id')['order_value'].transform(lambda x: x.rolling(window=2,
min_periods=1).mean())

print("Moving Average of Order Values:")

print(order_value_df[['customer_unique_id', 'order_purchase_timestamp', 'order_value',
'moving_average_order_value']].head(10))
```



```
Moving Average of Order Values:
                     customer_unique_id order_purchase_timestamp  order_value  \
87338  0000366f3b9a7992bf8c76cfdf3221e2      2018-05-10 10:56:27       129.90
20578  0000b849f77a49e4a4ce2b2a4ca5be3f      2018-05-07 11:11:27        18.90
68939  0000f46a3911fa3c0805444483337064      2017-03-10 21:05:03        69.00
25028  0000f6ccb0745a6a4b88665a16c9f078      2017-10-12 20:29:41        25.99
83852  0004aac84e0df4da2b147fca70cf8255      2017-11-14 19:45:42       180.00
23946  0004bd2a26a76fe21f786e4fbd80607f      2018-04-05 19:33:16       154.00
80227  00050ab1314c0e55a6ca13cf7181fecf      2018-04-20 12:57:23        27.99
26528  00053a61a98854899e70ed204dd4bafe      2018-02-28 11:15:41       382.00
67031  0005e1862207bf6ccc02e4228effd9a0      2017-03-04 23:32:12       135.00
641    0005ef4cd20d2893f0d9fbd94d3c0d97      2018-03-12 15:22:12       104.90

       moving_average_order_value
87338                      129.90
20578                       18.90
68939                       69.00
25028                       25.99
83852                      180.00
23946                      154.00
80227                       27.99
26528                      382.00
67031                      135.00
641                        104.90
```

SQL Code:

```
WITH order_values AS (    SELECT        t1.order_id,        t1.order_purchase_timestamp,
 t2.customer_unique_id,        SUM(t3.price) AS total_order_value    FROM        orders
 AS t1    JOIN        Customers AS t2 ON t1.customer_id = t2.customer_id    JOIN
 order_items AS t3 ON t1.order_id = t3.order_id    GROUP BY        t1.order_id,
 t1.order_purchase_timestamp,        t2.customer_unique_id)SELECT    order_id,
 customer_unique_id,    order_purchase_timestamp,    total_order_value,
 AVG(total_order_value) OVER (        PARTITION BY customer_unique_id        ORDER BY
 order_purchase_timestamp        ROWS BETWEEN 1 PRECEDING AND CURRENT ROW    )
 AS moving_average_order_valueFROM    order_valuesORDER BY
  customer_unique_id,    order_purchase_timestamp;
```

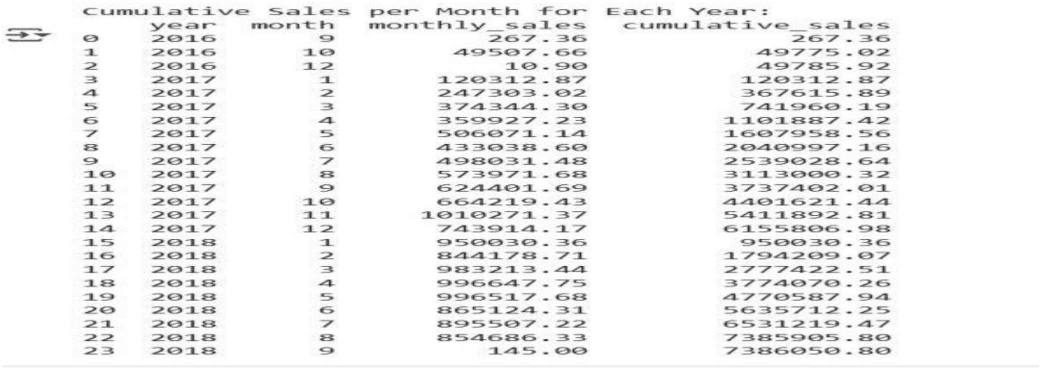


order_id	customer_unique_id	order_purchase_timestamp	total_order_value	moving_average_
e22acc9c116caa3f2b7121bbb380d08e	0000366f3b9a7992bf8c76cfdf3221e2	2018-05-10 10:56:27	129.9	129.9
3594e05a005ac4d06a72673270ef9ec9	0000b849f77a49e4a4ce2b2a4ca5be3f	2018-05-07 11:11:27	18.9	18.9
b33ec3b699337181488304f362a6b734	0000f46a3911fa3c0805444483337064	2017-03-10 21:05:03	69	69
41272756ecddd9a9ed0180413cc22fb6	0000f6ccb0745a6a4b88665a16c9f078	2017-10-12 20:29:41	25.99	25.99
d957021f1127559cd947b62533f484f7	0004aac84e0df4da2b147fca70cf8255	2017-11-14 19:45:42	180	180
3e470077b690ea3e3d501cffb5e0c499	0004bd2a26a76fe21f786e4fbd80607f	2018-04-05 19:33:16	154	154
d0028facea13f508e880202d7097a5a1	00050ab1314c0e55a6ca13cf7181fecf	2018-04-20 12:57:23	27.99	27.99
44e608f2db00c74a1fe329de44416a4e	00053a61a98854899e70ed204dd4bafe	2018-02-28 11:15:41	382	382
ae76bef74b97bcb0b3e355e60d9a6f9c	0005e1862207bf6ccc02e4228effd9a0	2017-03-04 23:32:12	135	135
01b330808c5819a6a3cb79b72f0b8288	0005ef4cd20d2893f0d9fbd94d3c0d97	2018-03-12 15:22:12	104.9	104.9
6681163e3dab91c549952b2845b20281	0006fdc98a402fceb4eb0ee528f6a8d4	2017-07-18 9:23:10	13.9	13.9
67503374d1fbcbe5e3a40324f703ffc8	00082cbe03e478190aadbea78542e933	2017-11-19 15:22:02	79	79
85bf8863657bff31006811d45d1c8db9	00090324bbad0e9342388303bb71ba0a	2018-03-24 14:44:41	49.95	49.95
d342d7bad292d68d41d58ffb594cf431	000949456b182f53c18b68d6babc79c1	2018-04-23 9:55:46	64.89	64.89

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

Python Code:

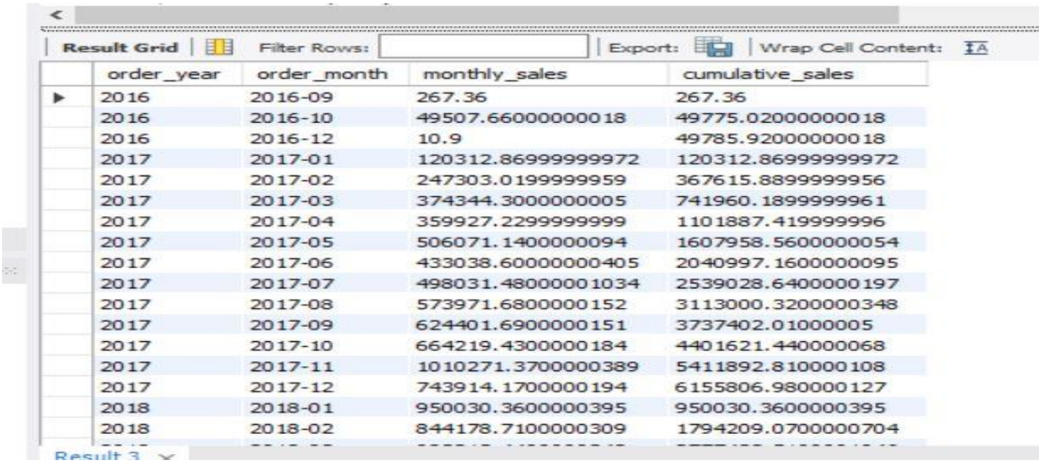
```
orders_df = pd.read_csv('orders.csv')
order_items_df = pd.read_csv('order_items.csv')
order_sales = order_items_df.groupby('order_id').agg(total_sales=('price', 'sum')).reset_index()
order_sales_with_date = pd.merge(orders_df[['order_id', 'order_purchase_timestamp']],order_sales,
 on='order_id')
order_sales_with_date['order_purchase_timestamp'] =
pd.to_datetime(order_sales_with_date['order_purchase_timestamp'])
order_sales_with_date['year'] = order_sales_with_date['order_purchase_timestamp'].dt.year
order_sales_with_date['month'] = order_sales_with_date['order_purchase_timestamp'].dt.month
monthly_sales = order_sales_with_date.groupby(['year', 'month']).agg(monthly_sales=('total_sales',
 'sum')).reset_index()
monthly_sales = monthly_sales.sort_values(by=['year', 'month'])
monthly_sales['cumulative_sales'] = monthly_sales.groupby('year')['monthly_sales'].cumsum()
print("\nCumulative Sales per Month for Each Year:")
```



```
Cumulative Sales per Month for Each Year:
    year  month  monthly_sales  cumulative_sales
0   2016      9         267.36            267.36
1   2016     10       49507.66          49775.02
2   2016     12          10.90          49785.92
3   2017      1      120312.87         120312.87
4   2017      2      247303.02         367615.89
5   2017      3      374344.30         741960.19
6   2017      4      359927.23        1101887.42
7   2017      5      506071.14        1607958.56
8   2017      6      433038.60        2040997.16
9   2017      7      498031.48        2539028.64
10  2017      8      573971.68        3113000.32
11  2017      9      624401.69        3737402.01
12  2017     10      664219.43        4401621.44
13  2017     11     1010271.37        5411892.81
14  2017     12      743914.17        6155806.98
15  2018      1      950030.36         950030.36
16  2018      2      844178.71        1794209.07
17  2018      3      983213.44        2777422.51
18  2018      4      996647.75        3774070.26
19  2018      5      996517.68        4770587.94
20  2018      6      865124.31        5635712.25
21  2018      7      895507.22        6531219.47
22  2018      8      854686.33        7385905.80
23  2018      9         145.00        7386050.80
```

SQL Code:

```
WITH MonthlySales AS ( SELECT
 DATE_FORMAT(T1.order_purchase_timestamp, '%Y') AS order_year,
 DATE_FORMAT(T1.order_purchase_timestamp, '%Y-%m') AS order_month,
 SUM(T2.price) AS monthly_sales FROM orders AS T1 JOIN order_items AS T2
 ON T1.order_id = T2.order_id GROUP BY order_year, order_month ) SELECT
 order_year, order_month, monthly_sales, SUM(monthly_sales) OVER (
 PARTITION BY order_year ORDER BY order_month ) AS cumulative_sales FROM
 MonthlySales ORDER BY order_year, order_month;
```



	order_year	order_month	monthly_sales	cumulative_sales
▶	2016	2016-09	267.36	267.36
	2016	2016-10	49507.6600000018	49775.0200000018
	2016	2016-12	10.9	49785.9200000018
	2017	2017-01	120312.86999999972	120312.86999999972
	2017	2017-02	247303.0199999959	367615.8899999956
	2017	2017-03	374344.3000000005	741960.1899999961
	2017	2017-04	359927.2299999999	1101887.419999996
	2017	2017-05	506071.1400000094	1607958.5600000054
	2017	2017-06	433038.60000000405	2040997.1600000095
	2017	2017-07	498031.48000001034	2539028.6400000197
	2017	2017-08	573971.6800000152	3113000.3200000348
	2017	2017-09	624401.6900000151	3737402.01000005
	2017	2017-10	664219.4300000184	4401621.440000068
	2017	2017-11	1010271.3700000389	5411892.810000108
	2017	2017-12	743914.1700000194	6155806.980000127
	2018	2018-01	950030.3600000395	950030.3600000395
	2018	2018-02	844178.7100000309	1794209.0700000704

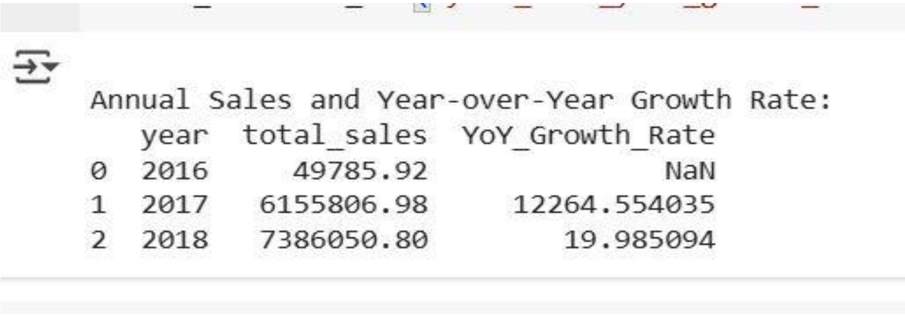
3. Calculate the year-over-year growth rate of total sales.

Python Code:

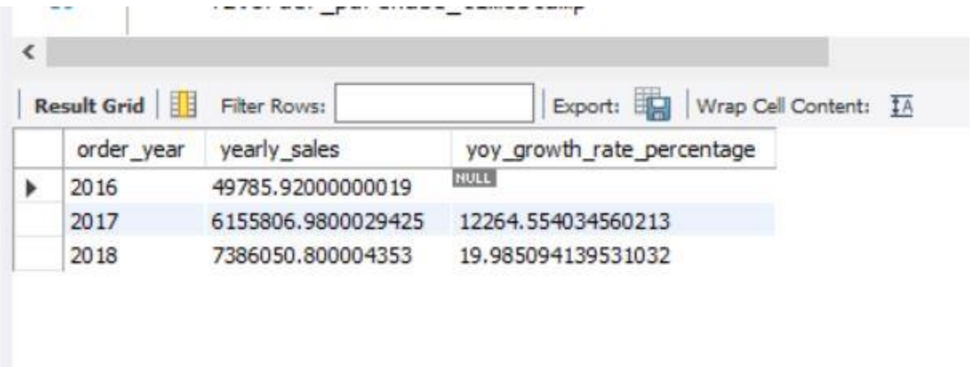
```
orders_df = pd.read_csv('orders.csv')

order_items_df = pd.read_csv('order_items.csv')

order_sales = order_items_df.groupby('order_id').agg(total_sales=('price',
'sum')).reset_index()

order_sales_with_date = pd.merge(orders_df[['order_id',
'order_purchase_timestamp']], order_sales, on='order_id')

order_sales_with_date['order_purchase_timestamp'] =
pd.to_datetime(order_sales_with_date['order_purchase_timestamp'])

order_sales_with_date['year'] =
order_sales_with_date['order_purchase_timestamp'].dt.year

annual_sales = order_sales_with_date.groupby('year')['total_sales'].sum().reset_index()

annual_sales['YoY_Growth_Rate'] = annual_sales['total_sales'].pct_change() * 100

print("\nAnnual Sales and Year-over-Year Growth Rate:")

print(annual_sales)

annual_sales.to_csv('year_over_year_growth_rate.csv', index=False)
```

SQL Code:

```
SELECT YEAR(T1.order_purchase_timestamp) AS order_year, SUM(T2.price) AS
 yearly_sales FROM orders AS T1 JOIN order_items AS T2 ON T1.order_id =
 T2.order_id GROUP BY order_year ) SELECT order_year, yearly_sales, (
 (yearly_sales - LAG(yearly_sales) OVER ( ORDER BY order_year )) /
 LAG(yearly_sales) OVER ( ORDER BY order_year ) ) * 100 AS
 yoy_growth_rate_percentage FROM AnnualSales ORDER BY order_year;
```



```
Annual Sales and Year-over-Year Growth Rate:
   year  total_sales  YoY_Growth_Rate
0  2016     49785.92              NaN
1  2017   6155806.98     12264.554035
2  2018   7386050.80        19.985094
```

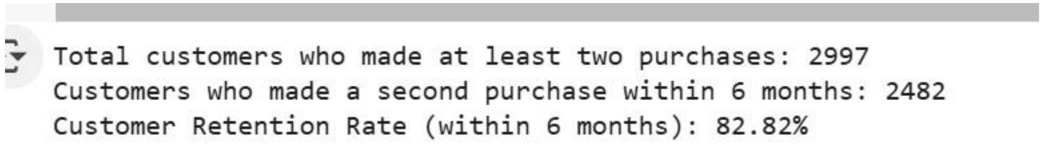


order_year	yearly_sales	yoy_growth_rate_percentage
2016	49785.92000000019	NULL
2017	6155806.9800029425	12264.554034560213
2018	7386050.800004353	19.985094139531032

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

Python code:

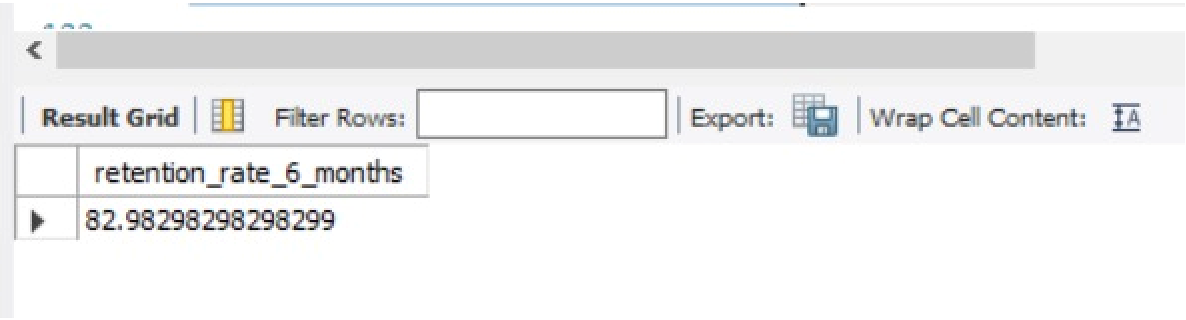
```
import pandas as pd

from datetime import timedelta

orders_df = pd.read_csv('orders.csv')

customers_df = pd.read_csv('Customers.csv')

merged_df = pd.merge(orders_df, customers_df, on='customer_id', how='inner')

merged_df['order_purchase_timestamp'] = pd.to_datetime(merged_df['order_purchase_timestamp'])

merged_df.sort_values(by=['customer_unique_id', 'order_purchase_timestamp'], inplace=True)

purchase_dates = merged_df.groupby('customer_unique_id')['order_purchase_timestamp'].apply(list)

retained_customers_data = purchase_dates[purchase_dates.apply(len) > 1].reset_index()

retained_customers_data['first_purchase'] =
retained_customers_data['order_purchase_timestamp'].apply(lambda x: x[0])

retained_customers_data['second_purchase'] =
retained_customers_data['order_purchase_timestamp'].apply(lambda x: x[1])

retained_customers_data['time_to_second_purchase'] = retained_customers_data['second_purchase'] -
retained_customers_data['first_purchase']

six_months = timedelta(days=180)

retained_count = (retained_customers_data['time_to_second_purchase'] <= six_months).sum()

total_retained_customers = len(retained_customers_data)

retention_rate = (retained_count / total_retained_customers) * 100

print(f"Total customers who made at least two purchases: {total_retained_customers}")

print(f"Customers who made a second purchase within 6 months: {retained_count}")

print(f"Customer Retention Rate (within 6 months): {retention_rate:.2f}%")
```



```
Total customers who made at least two purchases: 2997
Customers who made a second purchase within 6 months: 2482
Customer Retention Rate (within 6 months): 82.82%
```

SQL Code:

```
WITH CustomerOrders AS ( SELECT T1.customer_unique_id,
 T2.order_purchase_timestamp FROM Customers AS T1 JOIN orders AS T2 ON
 T1.customer_id = T2.customer_id ), FirstPurchaseDates AS ( SELECT
 customer_unique_id, MIN(order_purchase_timestamp) AS first_purchase_date
 FROM CustomerOrders GROUP BY customer_unique_id ), RetainedCustomers AS (
 SELECT DISTINCT T1.customer_unique_id FROM FirstPurchaseDates AS T1 JOIN
 CustomerOrders AS T2 ON T1.customer_unique_id = T2.customer_unique_id
 WHERE T2.order_purchase_timestamp > T1.first_purchase_date AND
 T2.order_purchase_timestamp <= DATE_ADD(T1.first_purchase_date, INTERVAL 6
 MONTH) ) SELECT ( ( SELECT COUNT(*) FROM RetainedCustomers ) * 100.0 ) / (
 SELECT COUNT(*) FROM FirstPurchaseDates ) AS retention_rate_percentage;
```

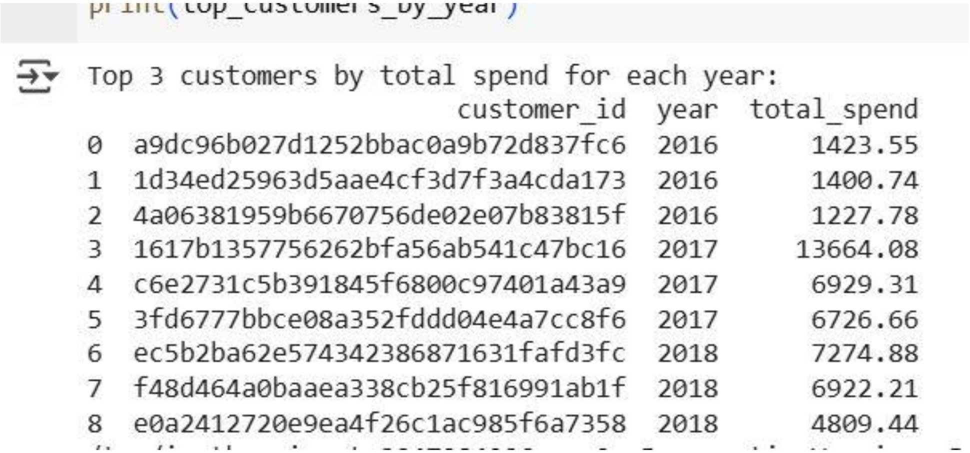


	retention_rate_6_months
▶	82.98298298298299

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

Python Code :

```
orders_df = pd.read_csv('orders.csv')

payments_df = pd.read_csv('payments.csv')

df_merged = pd.merge(orders_df, payments_df, on='order_id')

df_merged['order_purchase_timestamp'] =
pd.to_datetime(df_merged['order_purchase_timestamp'])

df_merged['year'] = df_merged['order_purchase_timestamp'].dt.year

customer_yearly_spend = df_merged.groupby(['customer_id',
'year'])['payment_value'].sum().reset_index()

customer_yearly_spend.columns = ['customer_id', 'year', 'total_spend']

top_customers_by_year = customer_yearly_spend.groupby('year').apply(lambda x:

 x.sort_values(by='total_spend', ascending=False).head(3)).reset_index(drop=True)

print("Top 3 customers by total spend for each year:")

print(top_customers_by_year)
```

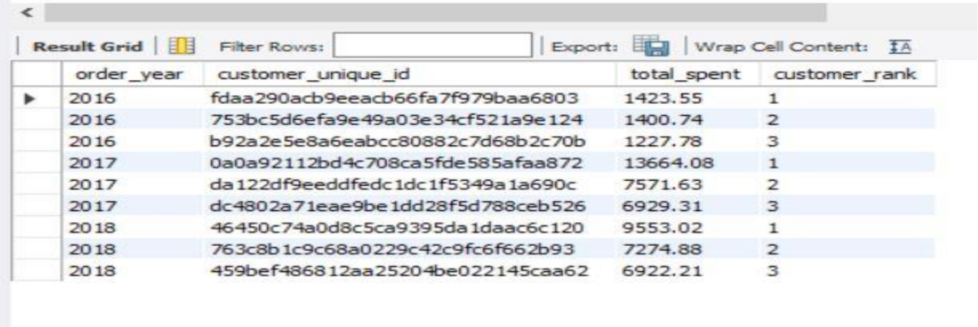


```
print(top_customers_by_year)
```

```
Top 3 customers by total spend for each year:
                        customer_id  year  total_spend
0  a9dc96b027d1252bbac0a9b72d837fc6  2016      1423.55
1  1d34ed25963d5aae4cf3d7f3a4cda173  2016      1400.74
2  4a06381959b6670756de02e07b83815f  2016      1227.78
3  1617b1357756262bfa56ab541c47bc16  2017     13664.08
4  c6e2731c5b391845f6800c97401a43a9  2017      6929.31
5  3fd6777bbce08a352fddd04e4a7cc8f6  2017      6726.66
6  ec5b2ba62e574342386871631fafd3fc  2018      7274.88
7  f48d464a0baaea338cb25f816991ab1f  2018      6922.21
8  e0a2412720e9ea4f26c1ac985f6a7358  2018      4809.44
```

SQL Code :

```
WITH CustomerOrderSpending AS ( SELECT T1.order_id, SUM(T1.payment_value) AS
 order_value FROM payments AS T1 GROUP BY T1.order_id ), CustomerAnnualSpending
 AS ( SELECT YEAR(T2.order_purchase_timestamp) AS order_year,
 T3.customer_unique_id, SUM(T1.order_value) AS total_spent FROM
 CustomerOrderSpending AS T1 JOIN orders AS T2 ON T1.order_id = T2.order_id JOIN
 Customers AS T3 ON T2.customer_id = T3.customer_id GROUP BY order_year,
 T3.customer_unique_id ), RankedCustomers AS ( SELECT order_year,
 customer_unique_id, total_spent, RANK() OVER ( PARTITION BY order_year ORDER BY
 total_spent DESC ) AS customer_rank FROM CustomerAnnualSpending ) SELECT
 order_year, customer_unique_id, total_spent, customer_rank FROM RankedCustomers
 WHERE customer_rank <= 3 ORDER BY order_year, customer_rank
```



order_year	customer_unique_id	total_spent	customer_rank
2016	fdaa290acb9eeacb66fa7f979baa6803	1423.55	1
2016	753bc5d6efa9e49a03e34cf521a9e124	1400.74	2
2016	b92a2e5e8a6eabcc80882c7d68b2c70b	1227.78	3
2017	0a0a92112bd4c708ca5fde585afaa872	13664.08	1
2017	da122df9eeddfedc1dc1f5349a1a690c	7571.63	2
2017	dc4802a71eae9be1dd28f5d788ceb526	6929.31	3
2018	46450c74a0d8c5ca9395da1daac6c120	9553.02	1
2018	763c8b1c9c68a0229c42c9fc6f662b93	7274.88	2
2018	459bef486812aa25204be022145caa62	6922.21	3

Recommendations

Customer Retention:

Introduce loyalty programs, subscription discounts, or cashback to improve repeat purchase rate.

Personalized product recommendations (using past purchase data) could increase retention.

Geographic Expansion:

Focus marketing on SP, RJ, MG since they contribute the bulk of sales.

But also design growth campaigns in RS, PR, SC where customer base is decent, but sales are relatively lower → untapped potential.

Product Strategy:

Push top categories (Health & Beauty, Watches, Bed/Bath) with targeted ads.

Bundle high-margin but less frequent categories (like electronics, computer accessories) with popular categories.

Payment Optimization:

Since ~50% of payments are installment-based, offering more flexible EMI options could drive higher order values.

Partnerships with banks/fintech's can reduce payment friction.

Seller Network:

Dependency is high on a few sellers. Encourage new seller onboarding from underrepresented states (e.g., Northern Brazil) to diversify.

Provide seller analytics dashboards to improve their sales performance.

Conclusion

The business shows strong YoY growth (20% in 2018) and a wide customer reach (4,119 cities) but suffers from very low customer retention (0%). Sales are highly concentrated in SP state and a few top sellers, which creates risk. To ensure long-term sustainability, the company should:

Build loyalty programs

Expand beyond SP into secondary states

Diversify seller & product portfolio.

With these steps, the platform can convert its large new customer base into repeat loyal buyers, ensuring sustainable revenue growth.