Navigating the Future of Online Shopping E-Commerce Data Analysis

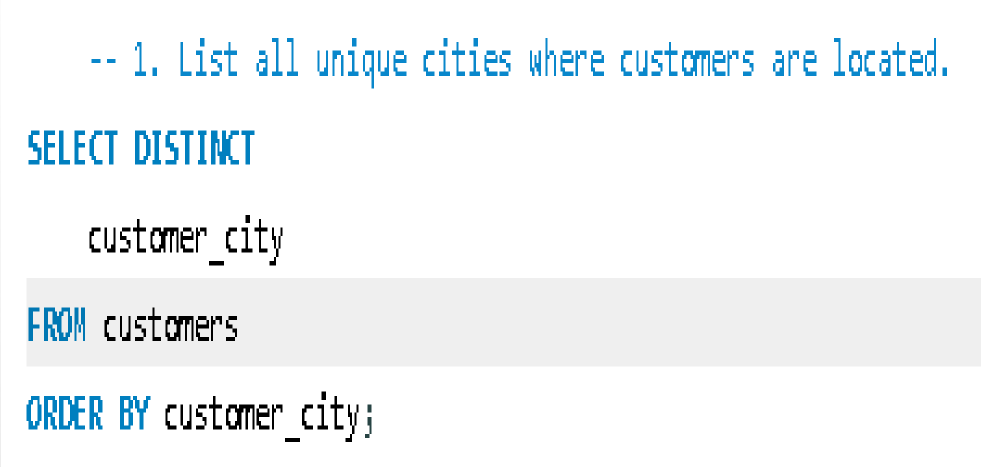
A Comparative Approach using SQL and Pytho

BASIC PROBLEMS

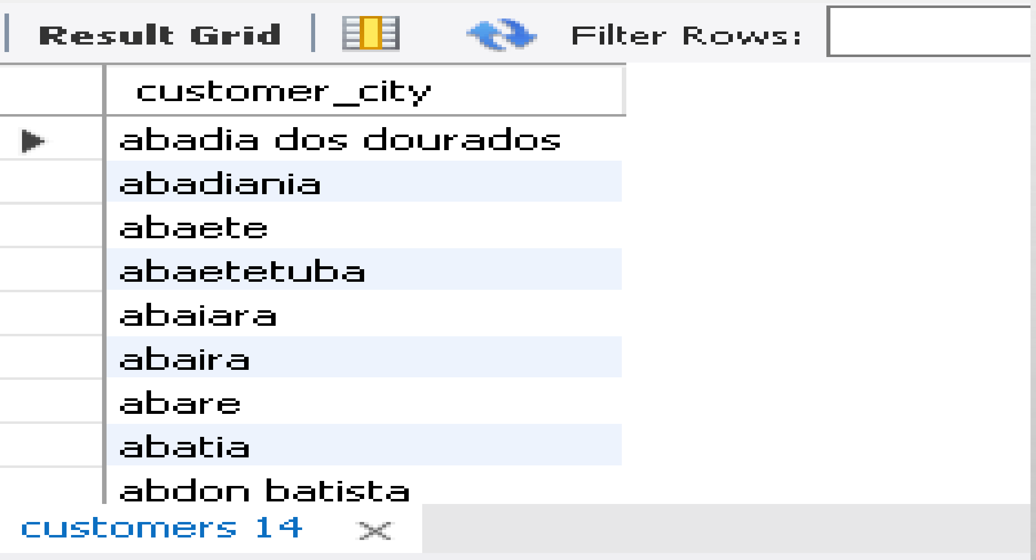
OBJECTIVE: EXTRACT FUNDAMENTAL INSIGHTS FROM THE DATASET

LIST ALL UNIQUE CITIES WHERE CUSTOMERS ARE LOCATED

SQL QUERY

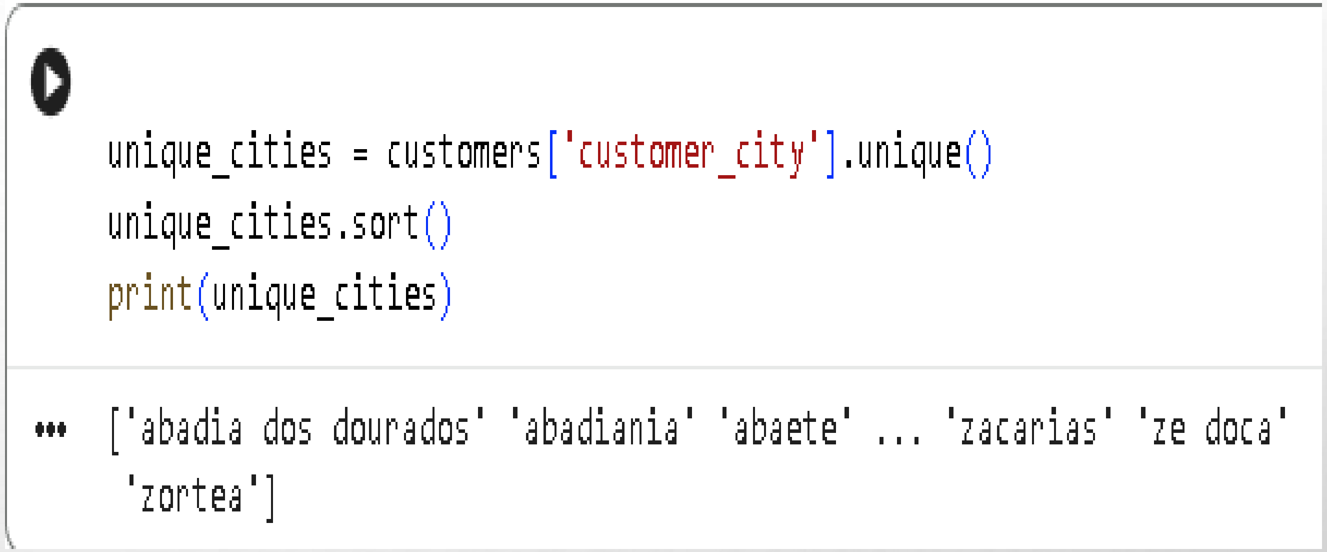


```
    -- 1. List all unique cities where customers are located.
SELECT DISTINCT
    customer_city
FROM customers
ORDER BY customer_city;
```



customer_city
abadia dos dourados
abadiania
abaete
abaetetuba
abaiara
abaira
abare
abatia
abdon batista

PYTHON CODE

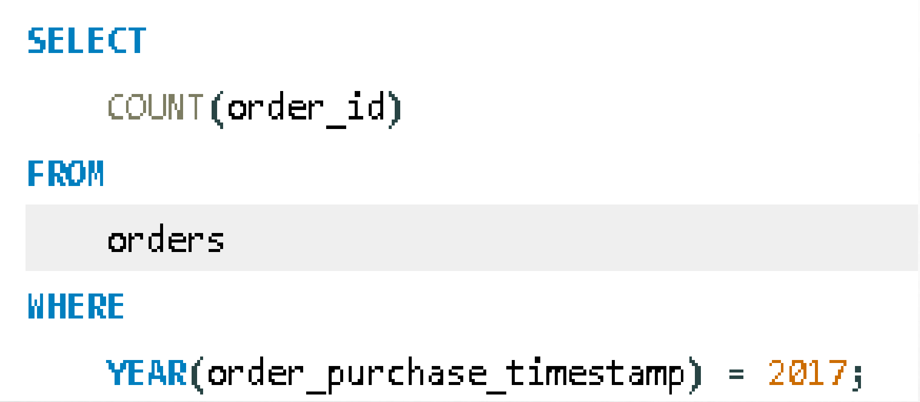


```
unique_cities = customers['customer_city'].unique()
unique_cities.sort()
print(unique_cities)
```

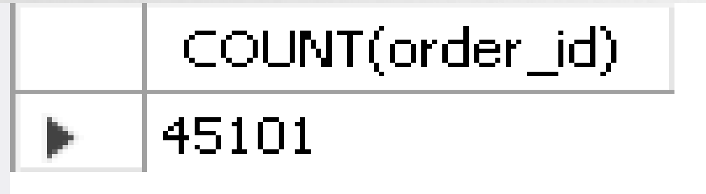
```
['abadia dos dourados' 'abadiania' 'abaete' ... 'zacarias' 'ze doca'
 'zortea']
```

COUNT THE NUMBER OF ORDERS PLACED IN 2017

SQL QUERY

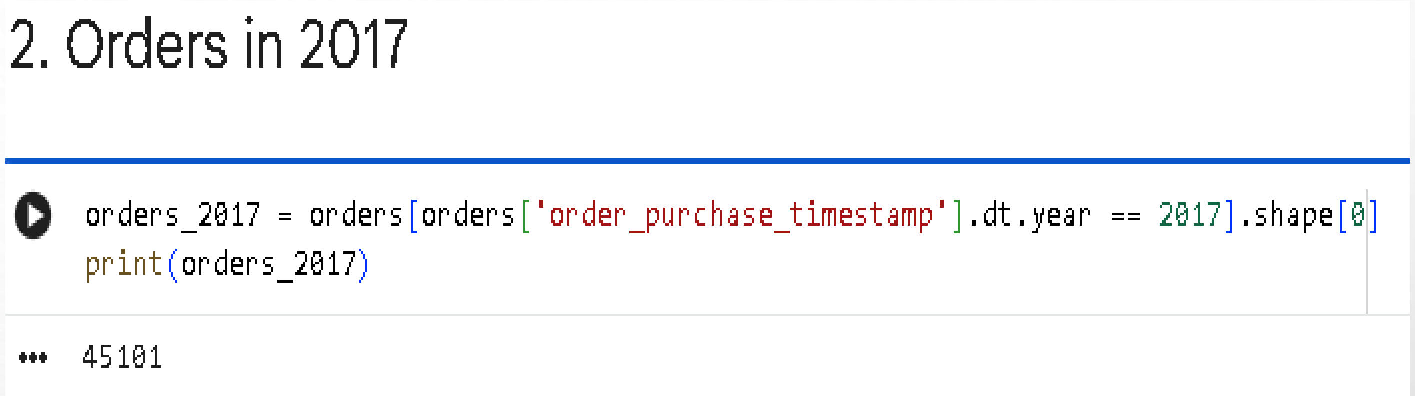


```sql
SELECT
    COUNT(order_id)
FROM
    orders
WHERE
    YEAR(order_purchase_timestamp) = 2017;
```



	COUNT(order_id)
▶	45101

PYTHON CODE



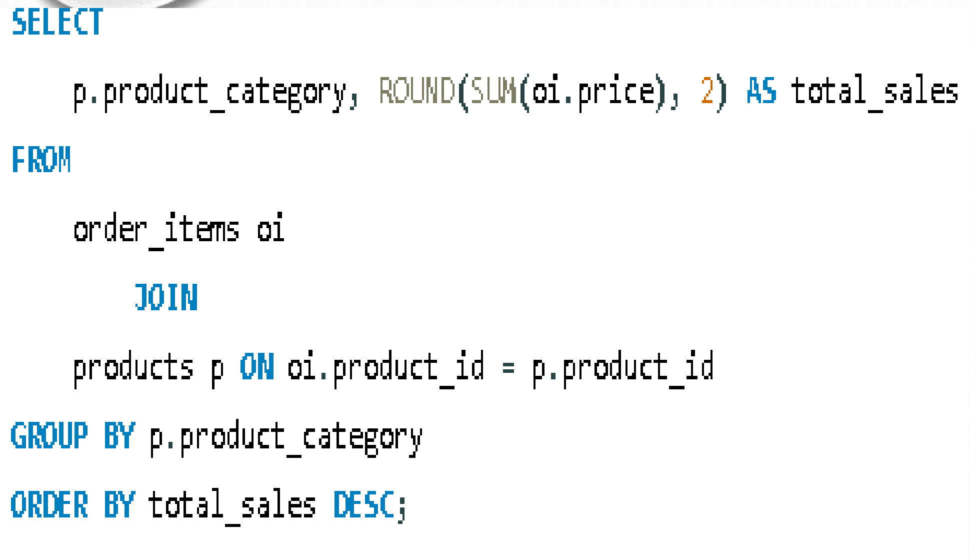
2. Orders in 2017

```python
orders_2017 = orders[orders['order_purchase_timestamp'].dt.year == 2017].shape[0]
print(orders_2017)
```

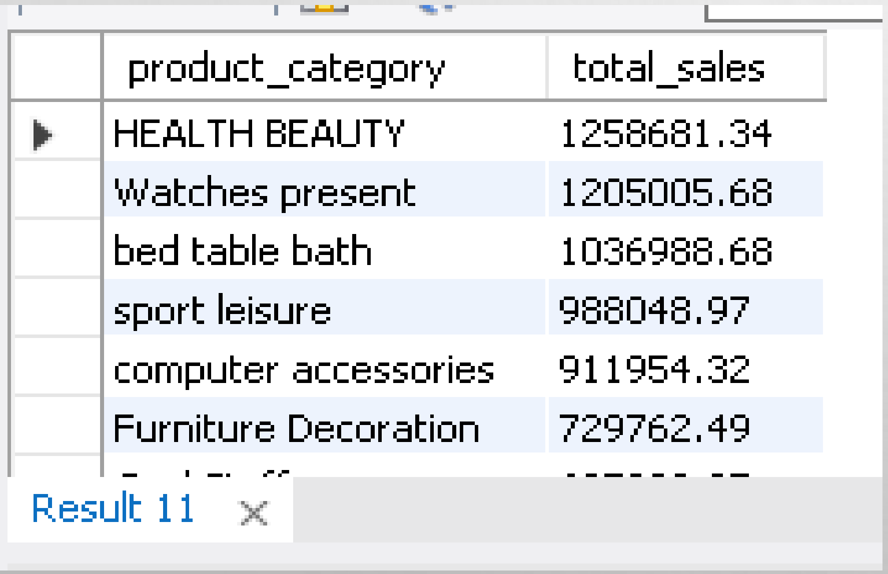
```
45101
```

FIND THE TOTAL SALES PER CATEGORY

SQL QUERY

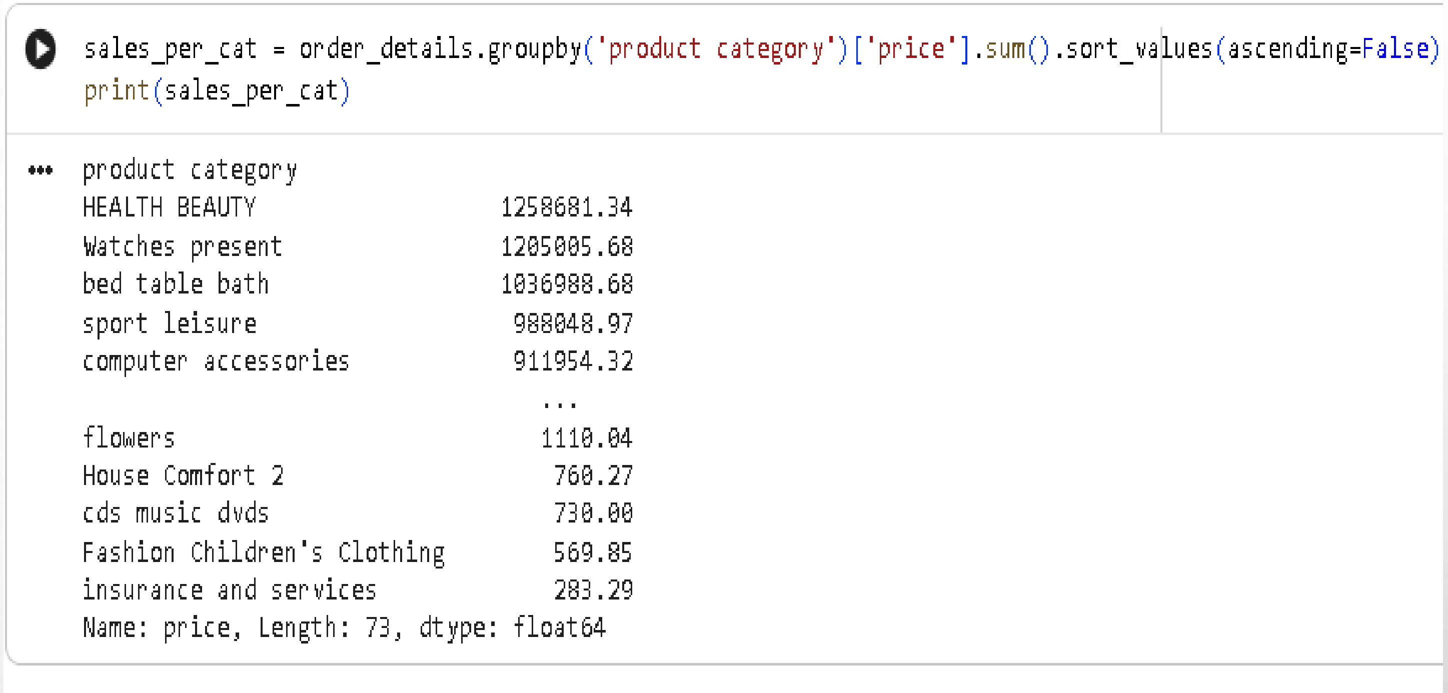


```sql
SELECT
    p.product_category, ROUND(SUM(oi.price), 2) AS total_sales
FROM
    order_items oi
        JOIN
    products p ON oi.product_id = p.product_id
GROUP BY p.product_category
ORDER BY total_sales DESC;
```



	product_category	total_sales
▶	HEALTH BEAUTY	1258681.34
	Watches present	1205005.68
	bed table bath	1036988.68
	sport leisure	988048.97
	computer accessories	911954.32
	Furniture Decoration	729762.49

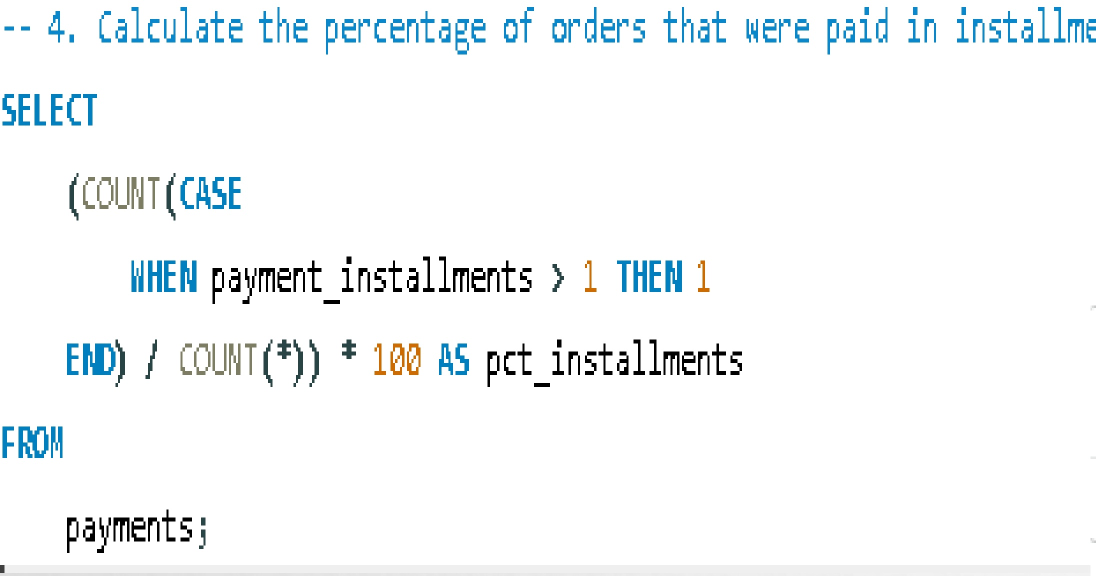
PYTHON CODE



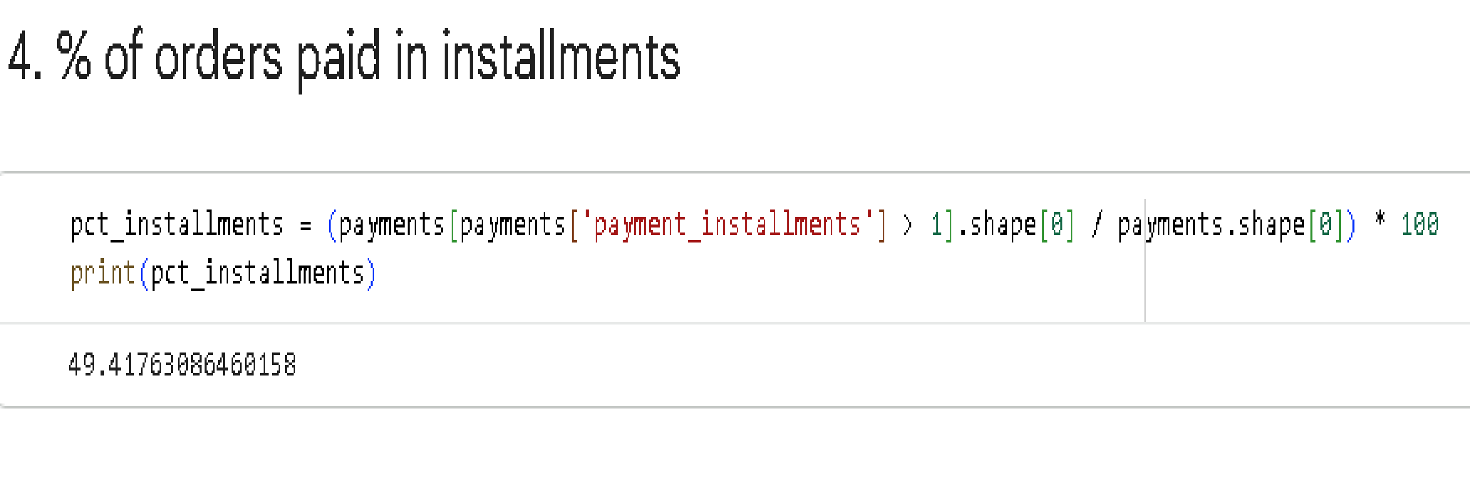
```python
sales_per_cat = order_details.groupby('product category')['price'].sum().sort_values(ascending=False)
print(sales_per_cat)
```

```
product category
HEALTH BEAUTY                 1258681.34
Watches present               1205005.68
bed table bath                1036988.68
sport leisure                  988048.97
computer accessories           911954.32
                                 ...
flowers                          1110.04
House Comfort 2                   760.27
cds music dvds                    730.00
Fashion Children's Clothing       569.85
insurance and services            283.29
Name: price, Length: 73, dtype: float64
```

CALCULATE THE PERCENTAGE OF ORDERS THAT WERE PAID IN INSTALLMENTS.



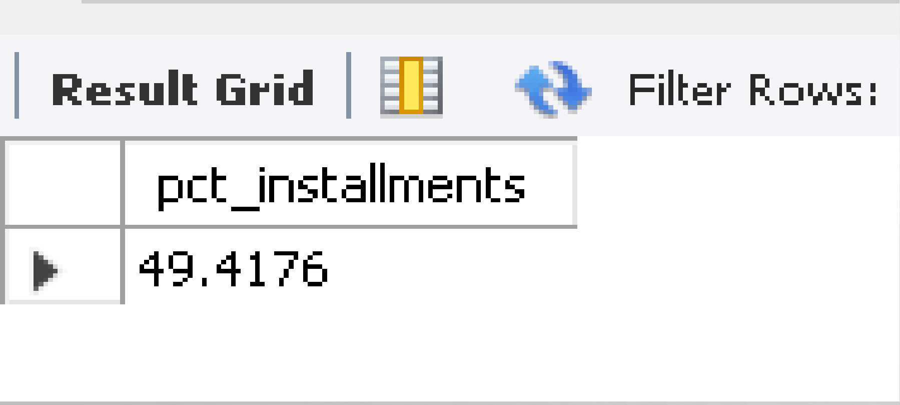
```sql
-- 4. Calculate the percentage of orders that were paid in installments.
SELECT
    (COUNT(CASE
        WHEN payment_installments > 1 THEN 1
    END) / COUNT(*)) * 100 AS pct_installments
FROM
    payments;
```



4. % of orders paid in installments

```python
pct_installments = (payments[payments['payment_installments'] > 1].shape[0] / payments.shape[0]) * 100
print(pct_installments)
```

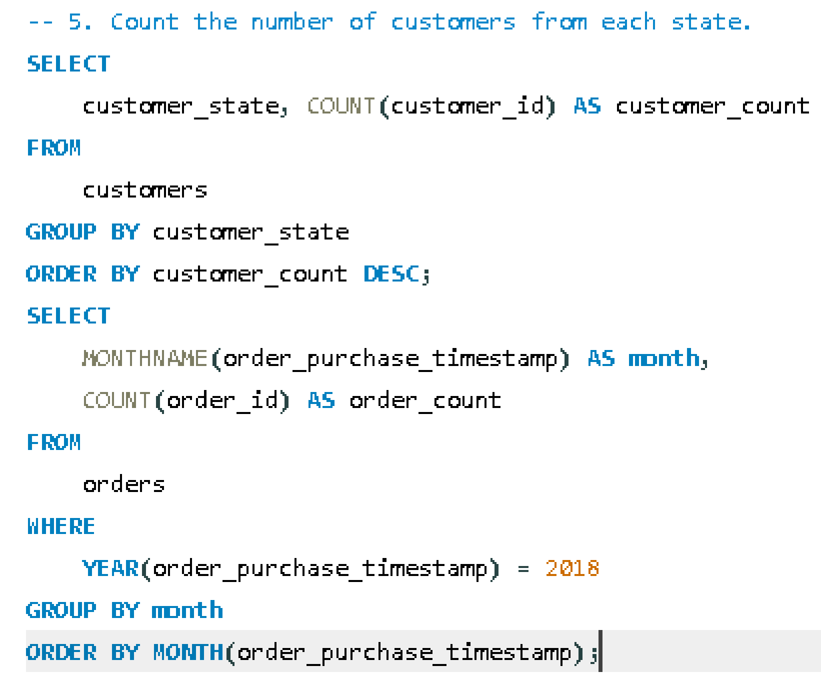
```
49.41763086460158
```



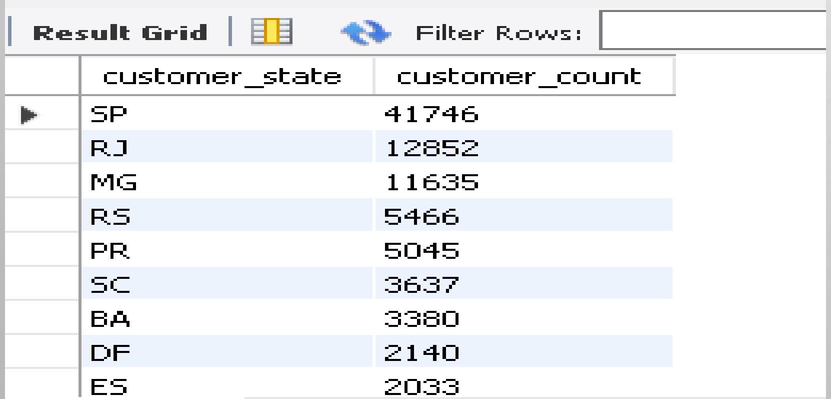
	pct_installments
▶	49.4176

COUNT THE NUMBER OF CUSTOMERS FROM EACH STATE

SQL QUERY

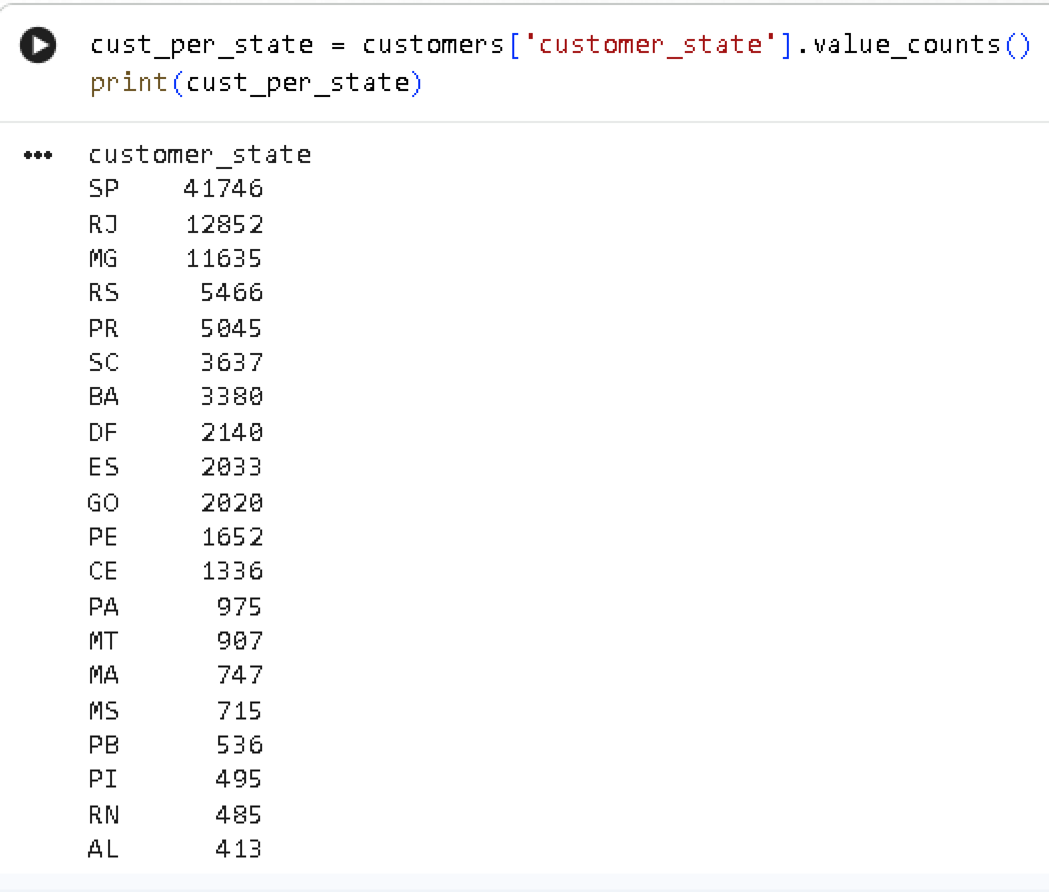


```
-- 5. Count the number of customers from each state.
SELECT
    customer_state, COUNT(customer_id) AS customer_count
FROM
    customers
GROUP BY customer_state
ORDER BY customer_count DESC;
SELECT
    MONTHNAME(order_purchase_timestamp) AS month,
    COUNT(order_id) AS order_count
FROM
    orders
WHERE
    YEAR(order_purchase_timestamp) = 2018
GROUP BY month
ORDER BY MONTH(order_purchase_timestamp);
```



customer_state	customer_count
SP	41746
RJ	12852
MG	11635
RS	5466
PR	5045
SC	3637
BA	3380
DF	2140
ES	2033

PYTHON CODE



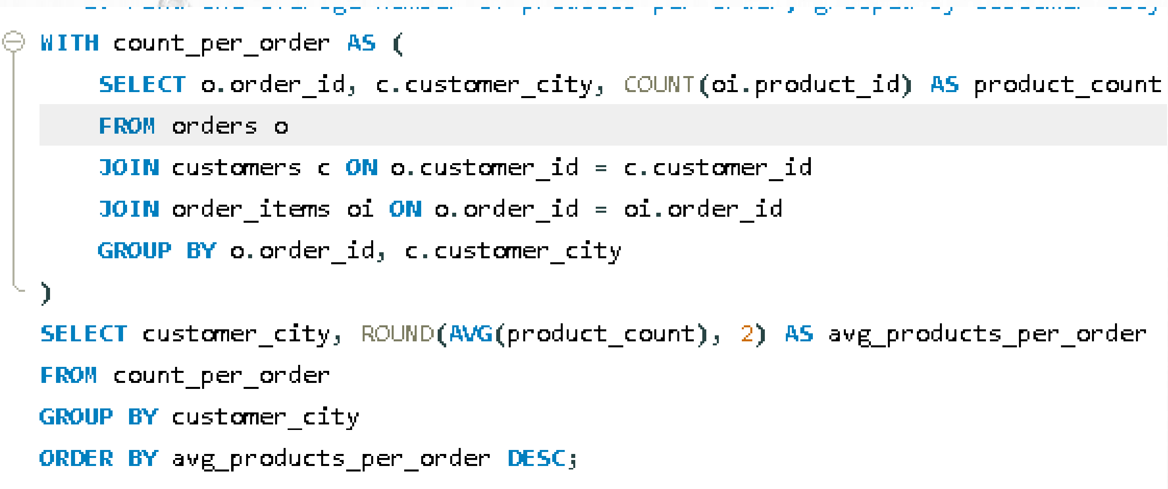
```
cust_per_state = customers['customer_state'].value_counts()
print(cust_per_state)
```

```
customer_state
SP    41746
RJ    12852
MG    11635
RS     5466
PR     5045
SC     3637
BA     3380
DF     2140
ES     2033
GO     2020
PE     1652
CE     1336
PA      975
MT      907
MA      747
MS      715
PB      536
PI      495
RN      485
AL      413
```

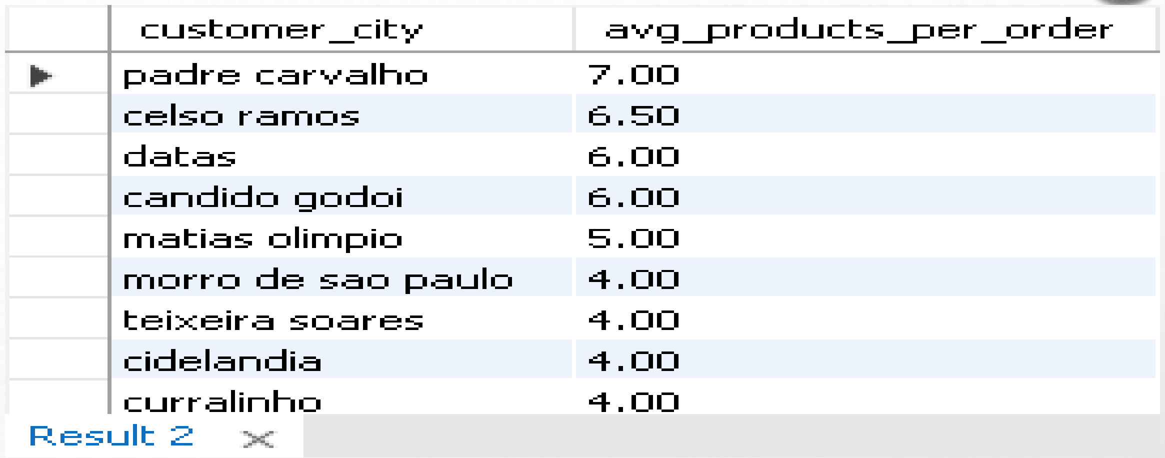
INTERMEDIATE PROBLEMS

OBJECTIVE: DIVE DEEPER INTO SALES AND ORDER TRENDS

FIND THE AVERAGE NUMBER OF PRODUCTS PER ORDER, GROUPED BY CUSTOMER CITY

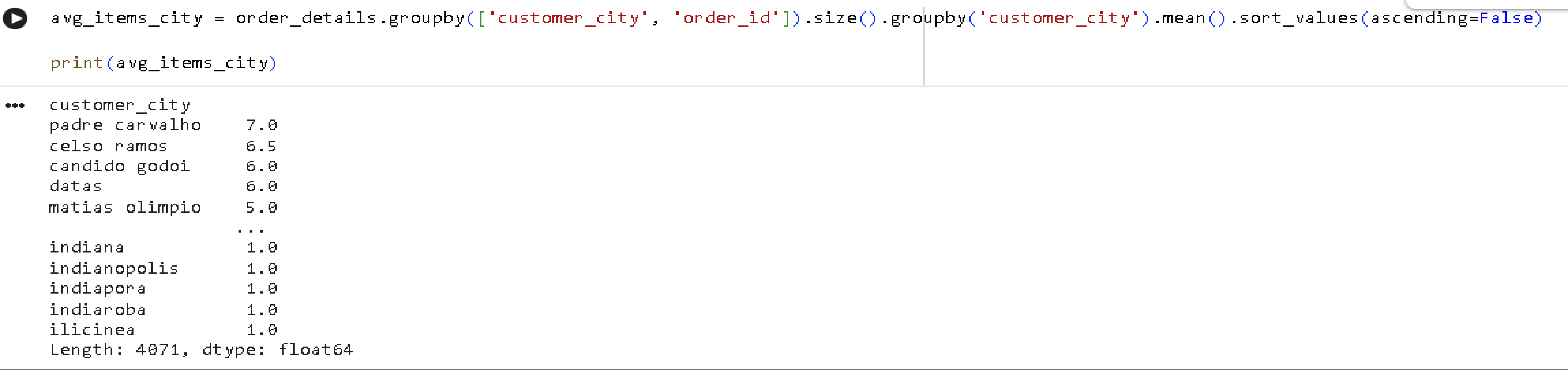


```sql
WITH count_per_order AS (
    SELECT o.order_id, c.customer_city, COUNT(oi.product_id) AS product_count
    FROM orders o
    JOIN customers c ON o.customer_id = c.customer_id
    JOIN order_items oi ON o.order_id = oi.order_id
    GROUP BY o.order_id, c.customer_city
)
SELECT customer_city, ROUND(AVG(product_count), 2) AS avg_products_per_order
FROM count_per_order
GROUP BY customer_city
ORDER BY avg_products_per_order DESC;
```



customer_city	avg_products_per_order
padre carvalho	7.00
celso ramos	6.50
datas	6.00
candido godoi	6.00
matias olimpio	5.00
morro de sao paulo	4.00
teixeira soares	4.00
cidelandia	4.00
curralinho	4.00

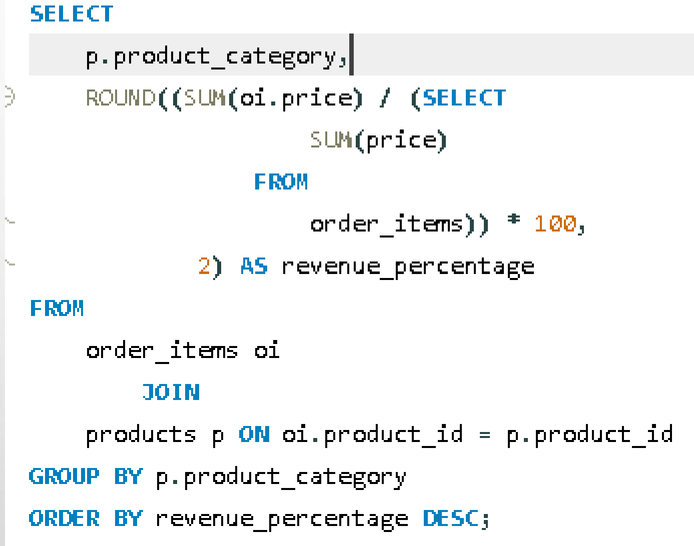
Python code



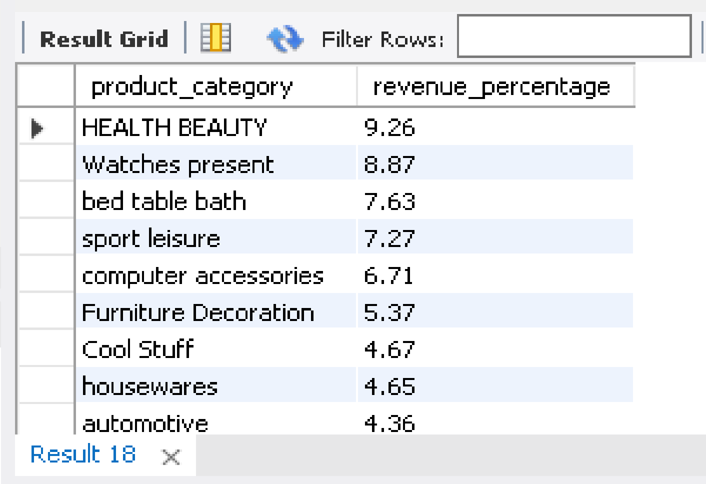
```python
avg_items_city = order_details.groupby(['customer_city', 'order_id']).size().groupby('customer_city').mean().sort_values(ascending=False)

print(avg_items_city)
```

```
customer_city
padre carvalho    7.0
celso ramos       6.5
candido godoi     6.0
datas             6.0
matias olimpio    5.0
                 ...
indiana           1.0
indianopolis      1.0
indiapora         1.0
indiaroba         1.0
ilicinea          1.0
Length: 4071, dtype: float64
```

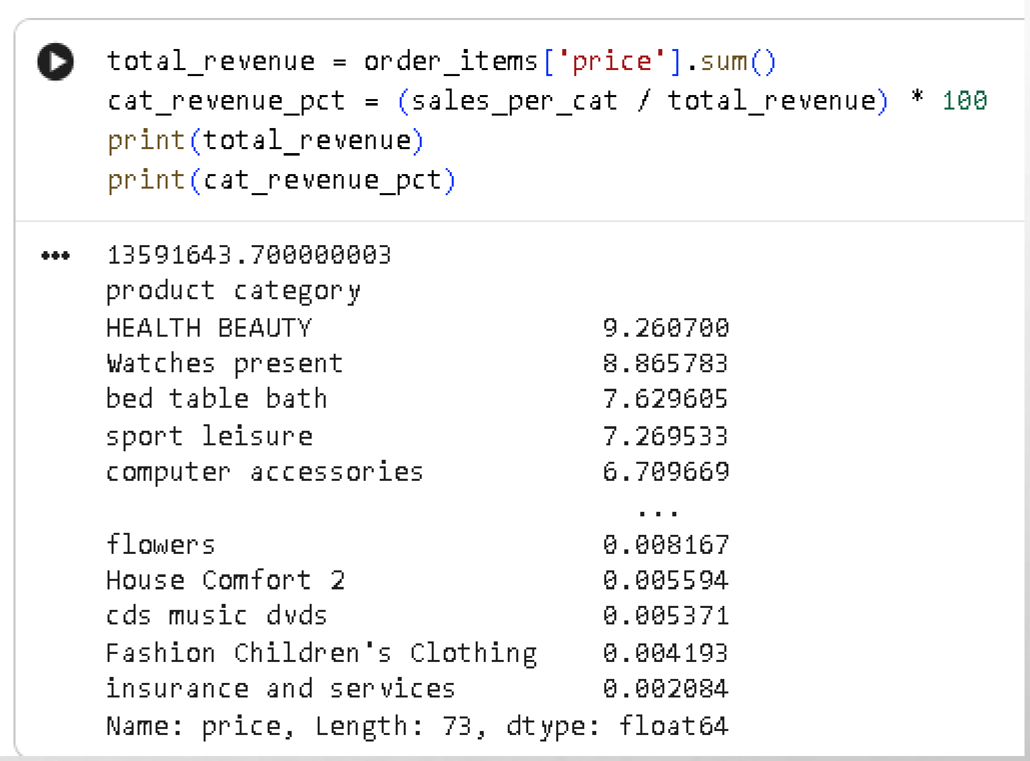
CALCULATE THE PERCENTAGE OF TOTAL REVENUE CONTRIBUTED BY EACH PRODUCT CATEGORY.



```sql
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
ORDER BY revenue_percentage DESC;
```



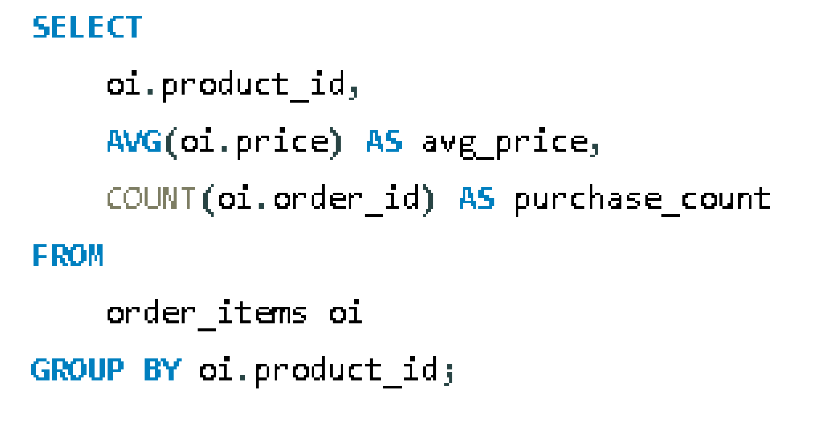
product_category	revenue_percentage
HEALTH BEAUTY	9.26
Watches present	8.87
bed table bath	7.63
sport leisure	7.27
computer accessories	6.71
Furniture Decoration	5.37
Cool Stuff	4.67
housewares	4.65
automotive	4.36



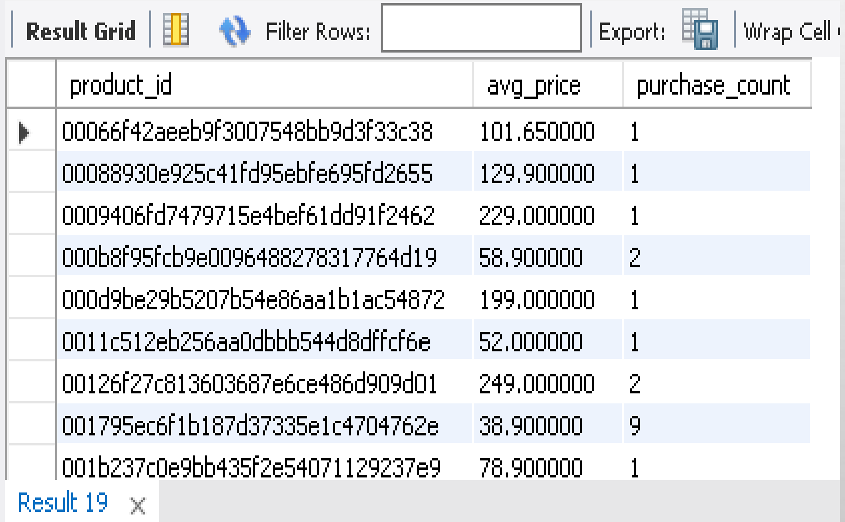
```python
total_revenue = order_items['price'].sum()
cat_revenue_pct = (sales_per_cat / total_revenue) * 100
print(total_revenue)
print(cat_revenue_pct)
```

```
13591643.700000003
product category
HEALTH BEAUTY                  9.260700
Watches present                8.865783
bed table bath                 7.629605
sport leisure                  7.269533
computer accessories           6.709669
                                 ...
flowers                        0.008167
House Comfort 2                0.005594
cds music dvds                 0.005371
Fashion Children's Clothing    0.004193
insurance and services         0.002084
Name: price, Length: 73, dtype: float64
```

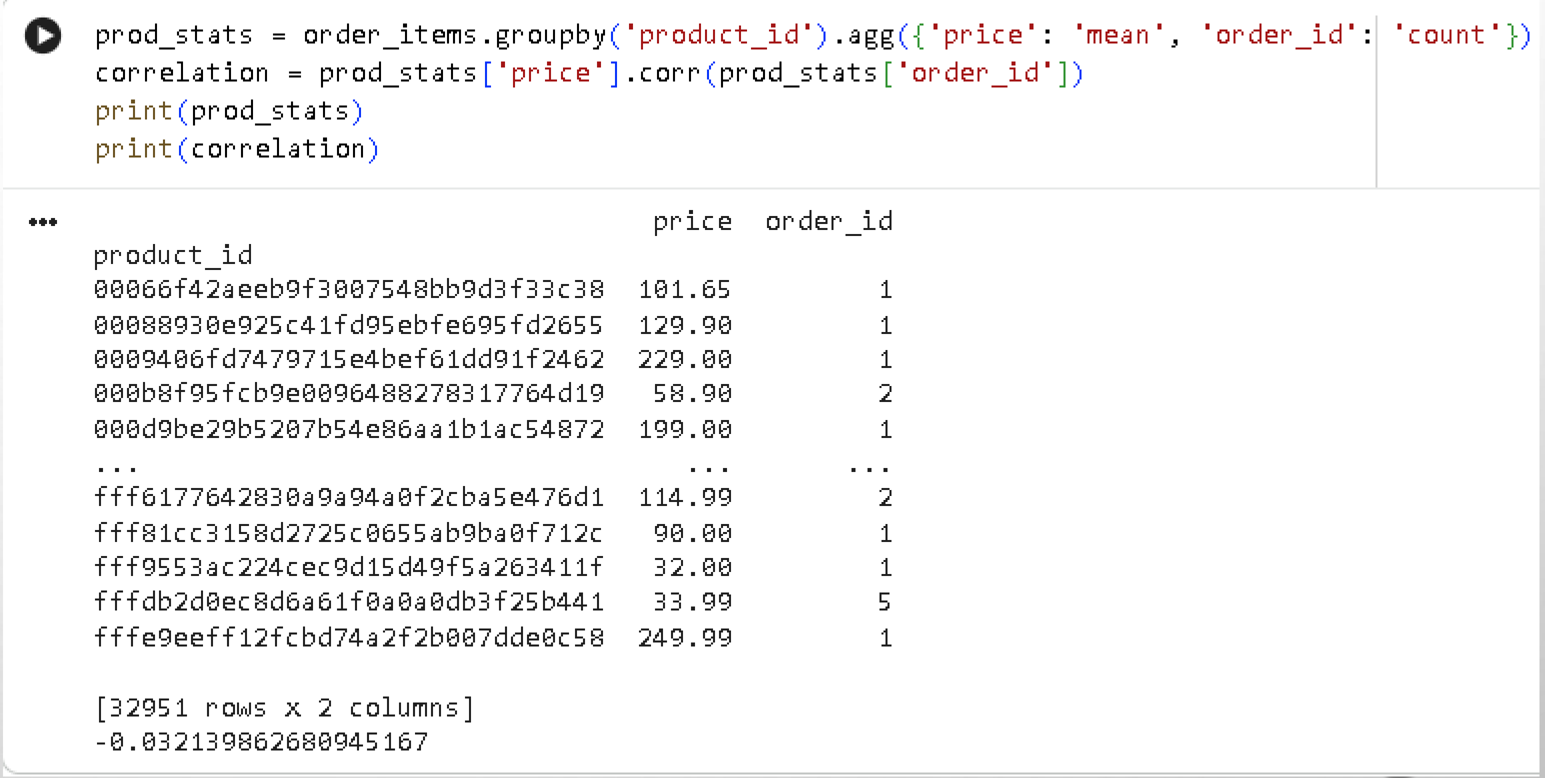
IDENTIFY THE CORRELATION BETWEEN PRODUCT PRICE AND THE NUMBER OF TIMES A PRODUCT HAS BEEN PURCHASED



```sql
SELECT
    oi.product_id,
    AVG(oi.price) AS avg_price,
    COUNT(oi.order_id) AS purchase_count
FROM
    order_items oi
GROUP BY oi.product_id;
```



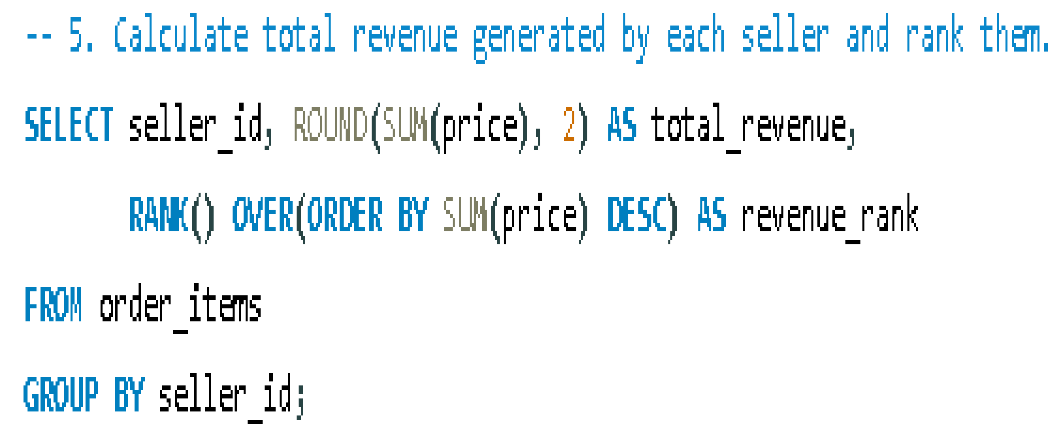
product_id	avg_price	purchase_count
00066f42aeeb9f3007548bb9d3f33c38	101.650000	1
00088930e925c41fd95ebfe695fd2655	129.900000	1
0009406fd7479715e4bef61dd91f2462	229.000000	1
000b8f95fcb9e0096488278317764d19	58.900000	2
000d9be29b5207b54e86aa1b1ac54872	199.000000	1
0011c512eb256aa0dbbb544d8dffcf6e	52.000000	1
00126f27c813603687e6ce486d909d01	249.000000	2
001795ec6f1b187d37335e1c4704762e	38.900000	9
001b237c0e9bb435f2e54071129237e9	78.900000	1



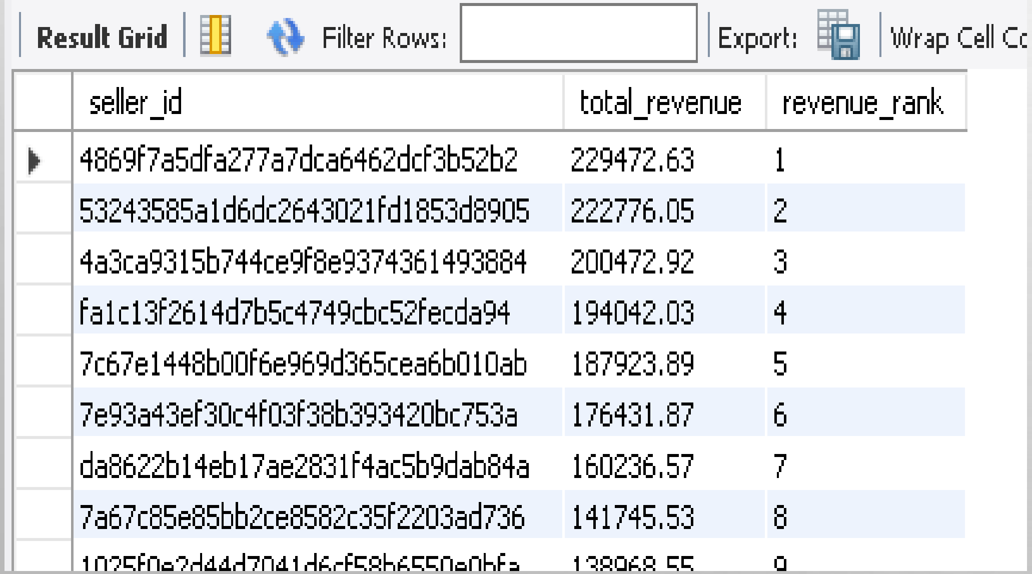
```python
prod_stats = order_items.groupby('product_id').agg({'price': 'mean', 'order_id': 'count'})
correlation = prod_stats['price'].corr(prod_stats['order_id'])
print(prod_stats)
print(correlation)
```

```
                                   price  order_id
product_id
00066f42aeeb9f3007548bb9d3f33c38  101.65         1
00088930e925c41fd95ebfe695fd2655  129.90         1
0009406fd7479715e4bef61dd91f2462  229.00         1
000b8f95fcb9e0096488278317764d19   58.90         2
000d9be29b5207b54e86aa1b1ac54872  199.00         1
...                                  ...       ...
fff6177642830a9a94a0f2cba5e476d1  114.99         2
fff81cc3158d2725c0655ab9ba0f712c   90.00         1
fff9553ac224cec9d15d49f5a263411f   32.00         1
fffdb2d0ec8d6a61f0a0a0db3f25b441   33.99         5
fffe9eeff12fcbd74a2f2b007dde0c58  249.99         1

[32951 rows x 2 columns]
-0.032139862680945167
```

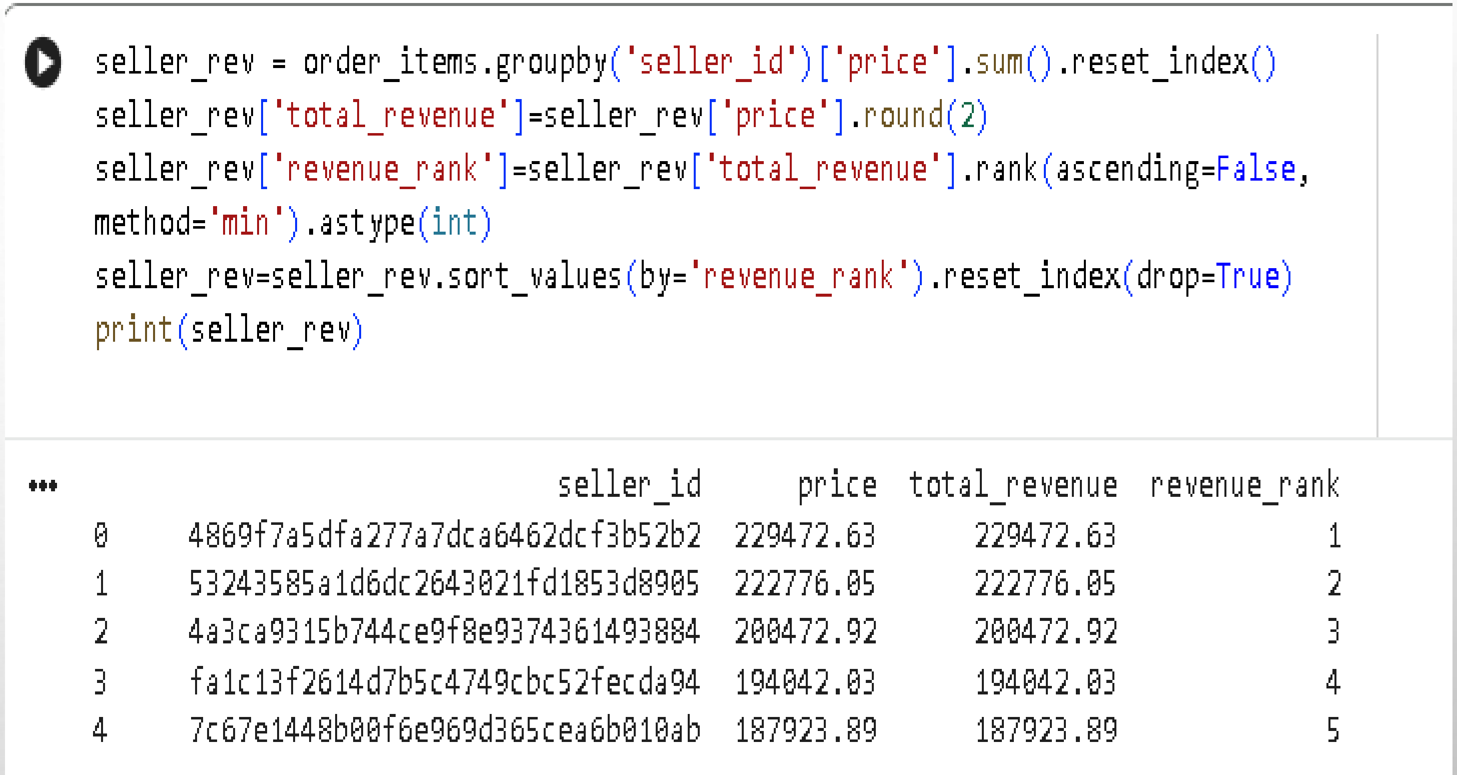
. CALCULATE THE TOTAL REVENUE GENERATED BY EACH SELLER, AND RANK THEM BY REVENUE.



```sql
-- 5. Calculate total revenue generated by each seller and rank them.

SELECT seller_id, ROUND(SUM(price), 2) AS total_revenue,

       RANK() OVER(ORDER BY SUM(price) DESC) AS revenue_rank

FROM order_items

GROUP BY seller_id;
```



seller_id	total_revenue	revenue_rank
4869f7a5dfa277a7dca6462dcf3b52b2	229472.63	1
53243585a1d6dc2643021fd1853d8905	222776.05	2
4a3ca9315b744ce9f8e9374361493884	200472.92	3
fa1c13f2614d7b5c4749cbc52fecda94	194042.03	4
7c67e1448b00f6e969d365cea6b010ab	187923.89	5
7e93a43ef30c4f03f38b393420bc753a	176431.87	6
da8622b14eb17ae2831f4ac5b9dab84a	160236.57	7
7a67c85e85bb2ce8582c35f2203ad736	141745.53	8
1025f0e2d44d7041d6cf58b6550e0bfa	138968.55	9



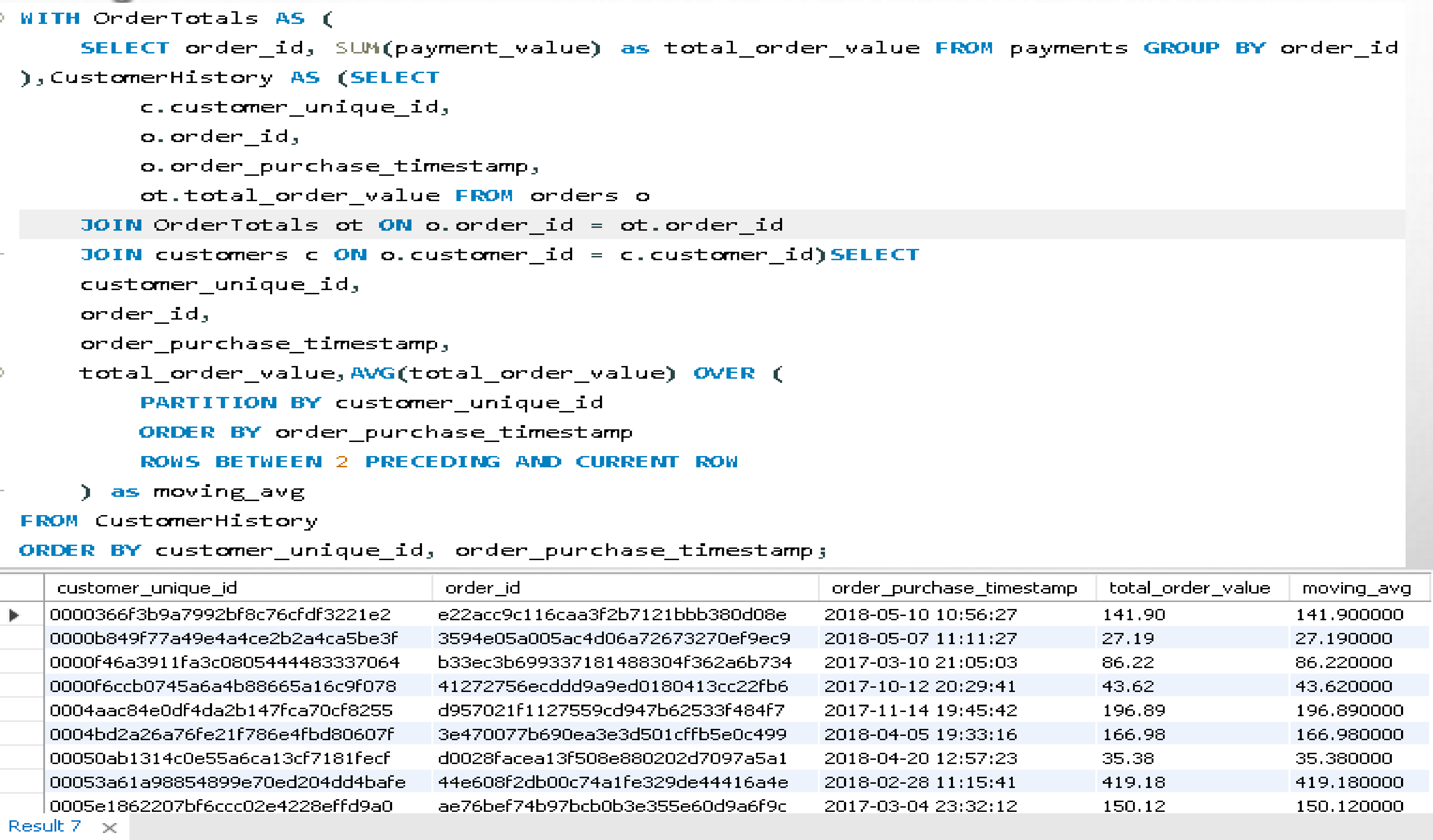
```python
seller_rev = order_items.groupby('seller_id')['price'].sum().reset_index()
seller_rev['total_revenue']=seller_rev['price'].round(2)
seller_rev['revenue_rank']=seller_rev['total_revenue'].rank(ascending=False,
method='min').astype(int)
seller_rev=seller_rev.sort_values(by='revenue_rank').reset_index(drop=True)
print(seller_rev)
```

```
                          seller_id      price  total_revenue  revenue_rank
0  4869f7a5dfa277a7dca6462dcf3b52b2  229472.63      229472.63             1
1  53243585a1d6dc2643021fd1853d8905  222776.05      222776.05             2
2  4a3ca9315b744ce9f8e9374361493884  200472.92      200472.92             3
3  fa1c13f2614d7b5c4749cbc52fecda94  194042.03      194042.03             4
4  7c67e1448b00f6e969d365cea6b010ab  187923.89      187923.89             5
```

ADVANCED PROBLEMS

OBJECTIVE: GENERATE STRATEGIC AND CUSTOMER-CENTRIC INSIGHTS

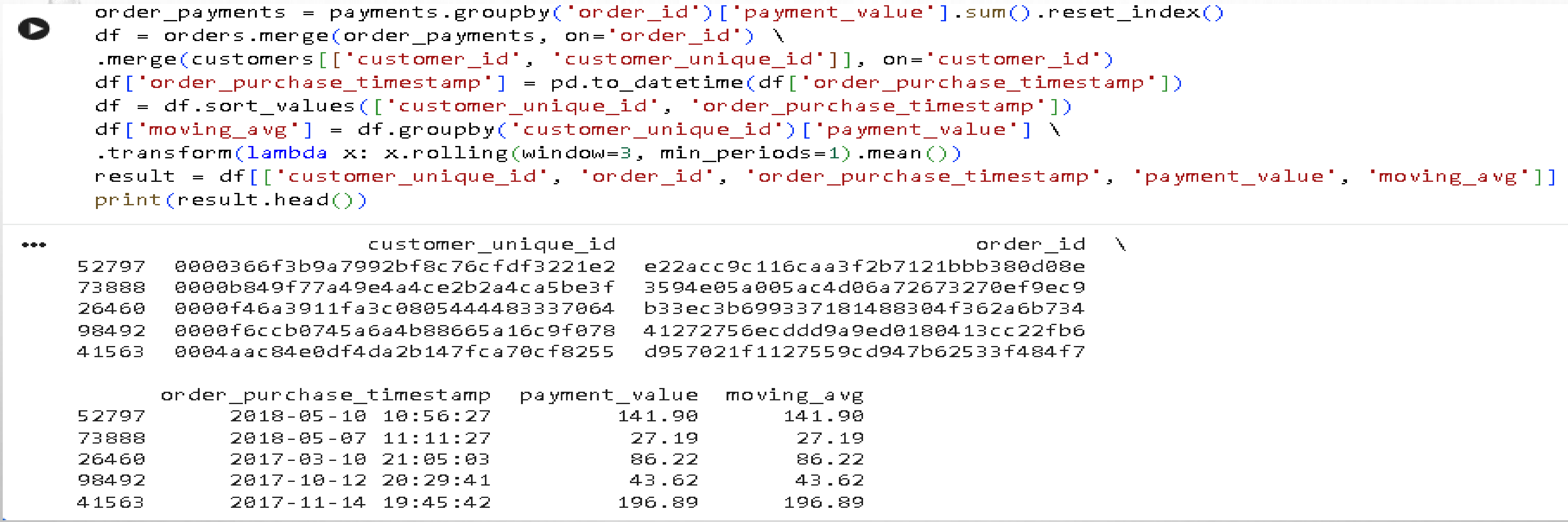
. CALCULATE THE MOVING AVERAGE OF ORDER VALUES FOR EACH CUSTOMER OVER THEIR ORDER HISTORY.



```
WITH OrderTotals AS (
    SELECT order_id, SUM(payment_value) as total_order_value FROM payments GROUP BY order_id
),CustomerHistory AS (SELECT
        c.customer_unique_id,
        o.order_id,
        o.order_purchase_timestamp,
        ot.total_order_value FROM orders o
    JOIN OrderTotals ot ON o.order_id = ot.order_id
    JOIN customers c ON o.customer_id = c.customer_id)SELECT
    customer_unique_id,
    order_id,
    order_purchase_timestamp,
    total_order_value,AVG(total_order_value) OVER (
        PARTITION BY customer_unique_id
        ORDER BY order_purchase_timestamp
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ) as moving_avg
FROM CustomerHistory
ORDER BY customer_unique_id, order_purchase_timestamp;
```

	customer_unique_id	order_id	order_purchase_timestamp	total_order_value	moving_avg
▶	0000366f3b9a7992bf8c76cfdf3221e2	e22acc9c116caa3f2b7121bbb380d08e	2018-05-10 10:56:27	141.90	141.900000
	0000b849f77a49e4a4ce2b2a4ca5be3f	3594e05a005ac4d06a72673270ef9ec9	2018-05-07 11:11:27	27.19	27.190000
	0000f46a3911fa3c0805444483337064	b33ec3b699337181488304f362a6b734	2017-03-10 21:05:03	86.22	86.220000
	0000f6ccb0745a6a4b88665a16c9f078	41272756ecddd9a9ed0180413cc22fb6	2017-10-12 20:29:41	43.62	43.620000
	0004aac84e0df4da2b147fca70cf8255	d957021f1127559cd947b62533f484f7	2017-11-14 19:45:42	196.89	196.890000
	0004bd2a26a76fe21f786e4fbd80607f	3e470077b690ea3e3d501cffb5e0c499	2018-04-05 19:33:16	166.98	166.980000
	00050ab1314c0e55a6ca13cf7181fecf	d0028facea13f508e880202d7097a5a1	2018-04-20 12:57:23	35.38	35.380000
	00053a61a98854899e70ed204dd4bafe	44e608f2db00c74a1fe329de44416a4e	2018-02-28 11:15:41	419.18	419.180000
	0005e1862207bf6ccc02e4228effd9a0	ae76bef74b97bcb0b3e355e60d9a6f9c	2017-03-04 23:32:12	150.12	150.120000

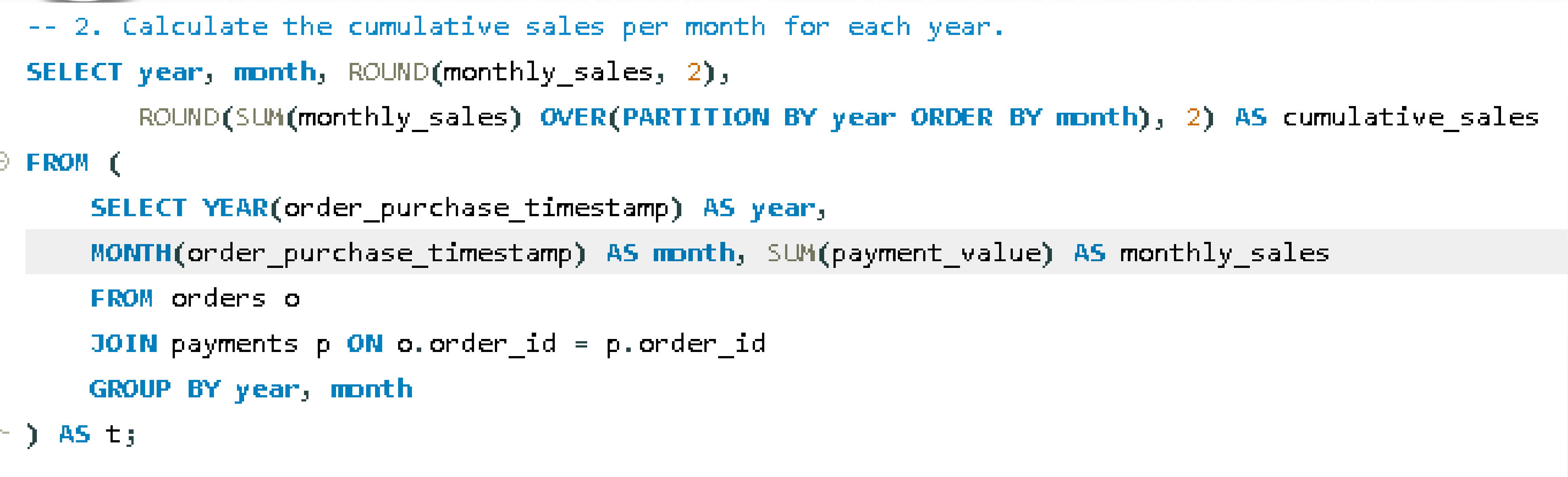
PYTHON CODE



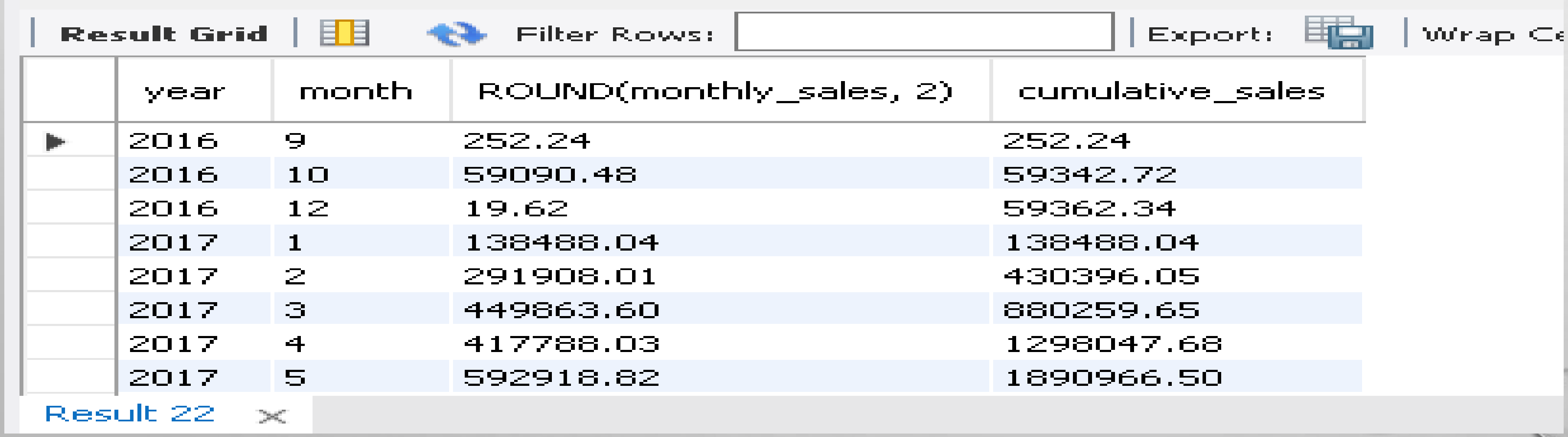
```
order_payments = payments.groupby('order_id')['payment_value'].sum().reset_index()
df = orders.merge(order_payments, on='order_id') \
.merge(customers[['customer_id', 'customer_unique_id']], on='customer_id')
df['order_purchase_timestamp'] = pd.to_datetime(df['order_purchase_timestamp'])
df = df.sort_values(['customer_unique_id', 'order_purchase_timestamp'])
df['moving_avg'] = df.groupby('customer_unique_id')['payment_value'] \
.transform(lambda x: x.rolling(window=3, min_periods=1).mean())
result = df[['customer_unique_id', 'order_id', 'order_purchase_timestamp', 'payment_value', 'moving_avg']]
print(result.head())
```

```
                     customer_unique_id                          order_id  \
52797  0000366f3b9a7992bf8c76cfdf3221e2  e22acc9c116caa3f2b7121bbb380d08e
73888  0000b849f77a49e4a4ce2b2a4ca5be3f  3594e05a005ac4d06a72673270ef9ec9
26460  0000f46a3911fa3c0805444483337064  b33ec3b699337181488304f362a6b734
98492  0000f6ccb0745a6a4b88665a16c9f078  41272756ecddd9a9ed0180413cc22fb6
41563  0004aac84e0df4da2b147fca70cf8255  d957021f1127559cd947b62533f484f7

      order_purchase_timestamp  payment_value  moving_avg
52797      2018-05-10 10:56:27         141.90      141.90
73888      2018-05-07 11:11:27          27.19       27.19
26460      2017-03-10 21:05:03          86.22       86.22
98492      2017-10-12 20:29:41          43.62       43.62
41563      2017-11-14 19:45:42         196.89      196.89
```

. CALCULATE THE CUMULATIVE SALES PER MONTH FOR EACH YEAR

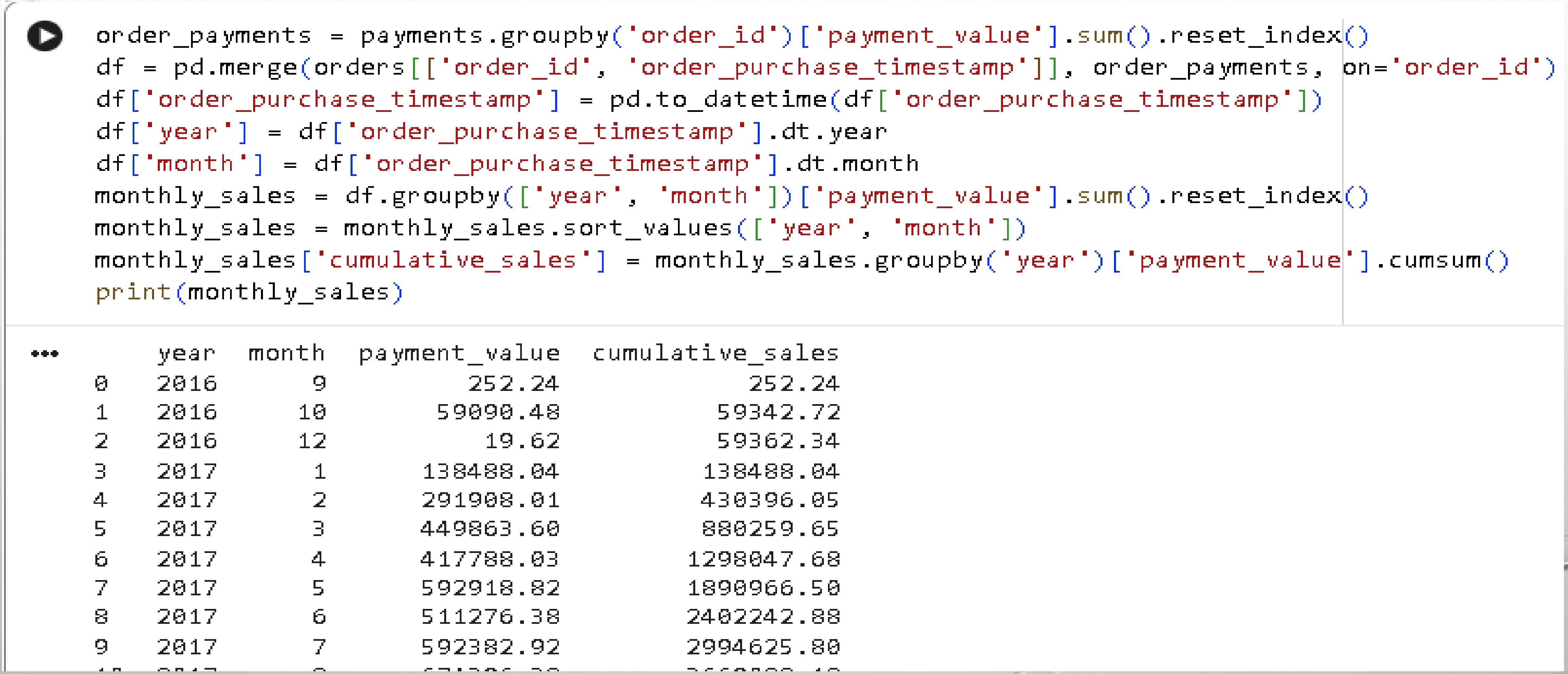


```
-- 2. Calculate the cumulative sales per month for each year.
SELECT year, month, ROUND(monthly_sales, 2),
       ROUND(SUM(monthly_sales) OVER(PARTITION BY year ORDER BY month), 2) AS cumulative_sales
FROM (
    SELECT YEAR(order_purchase_timestamp) AS year,
    MONTH(order_purchase_timestamp) AS month, SUM(payment_value) AS monthly_sales
    FROM orders o
    JOIN payments p ON o.order_id = p.order_id
    GROUP BY year, month
) AS t;
```



year	month	ROUND(monthly_sales, 2)	cumulative_sales
2016	9	252.24	252.24
2016	10	59090.48	59342.72
2016	12	19.62	59362.34
2017	1	138488.04	138488.04
2017	2	291908.01	430396.05
2017	3	449863.60	880259.65
2017	4	417788.03	1298047.68
2017	5	592918.82	1890966.50

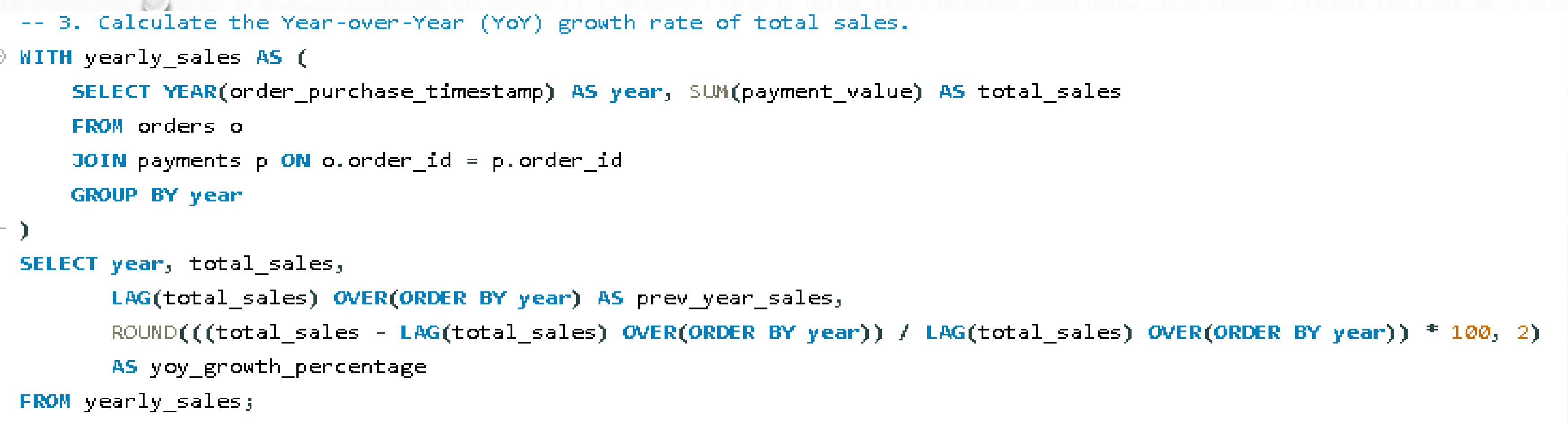
PYTHON CODE



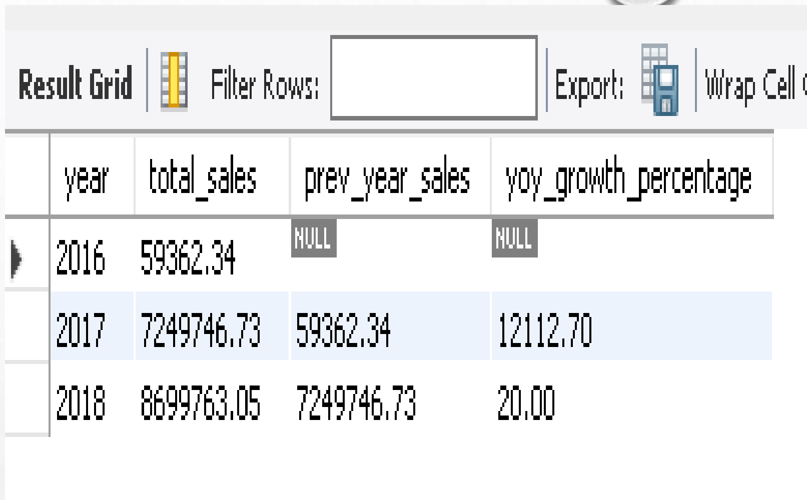
```python
order_payments = payments.groupby('order_id')['payment_value'].sum().reset_index()
df = pd.merge(orders[['order_id', 'order_purchase_timestamp']], order_payments, on='order_id')
df['order_purchase_timestamp'] = pd.to_datetime(df['order_purchase_timestamp'])
df['year'] = df['order_purchase_timestamp'].dt.year
df['month'] = df['order_purchase_timestamp'].dt.month
monthly_sales = df.groupby(['year', 'month'])['payment_value'].sum().reset_index()
monthly_sales = monthly_sales.sort_values(['year', 'month'])
monthly_sales['cumulative_sales'] = monthly_sales.groupby('year')['payment_value'].cumsum()
print(monthly_sales)
```

```
    year  month  payment_value  cumulative_sales
0   2016      9         252.24            252.24
1   2016     10       59090.48          59342.72
2   2016     12          19.62          59362.34
3   2017      1      138488.04         138488.04
4   2017      2      291908.01         430396.05
5   2017      3      449863.60         880259.65
6   2017      4      417788.03        1298047.68
7   2017      5      592918.82        1890966.50
8   2017      6      511276.38        2402242.88
9   2017      7      592382.92        2994625.80
```

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

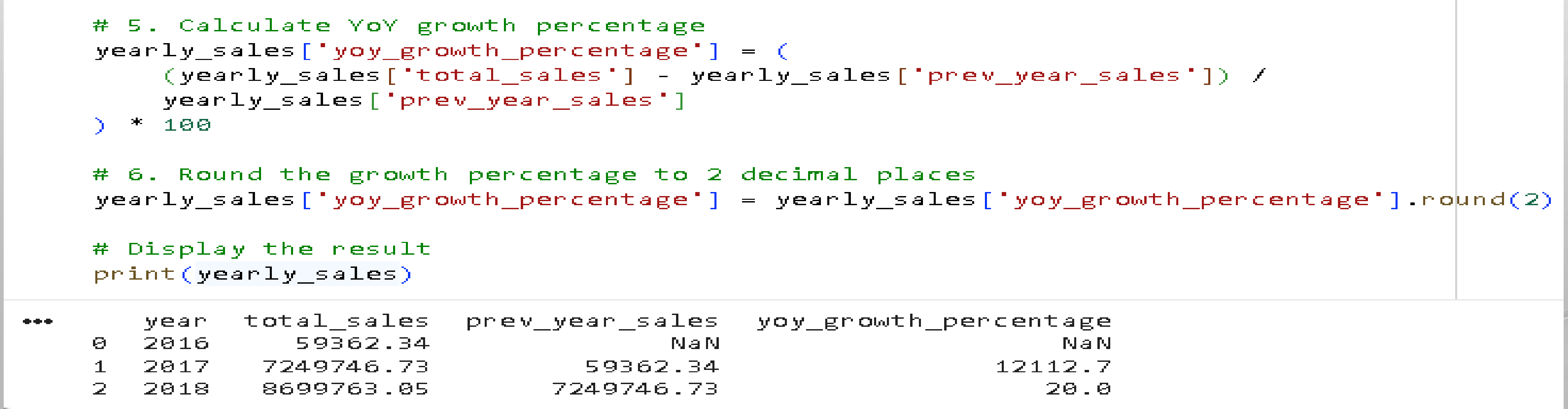


```sql
-- 3. Calculate the Year-over-Year (YoY) growth rate of total sales.
WITH yearly_sales AS (
    SELECT YEAR(order_purchase_timestamp) AS year, SUM(payment_value) AS total_sales
    FROM orders o
    JOIN payments p ON o.order_id = p.order_id
    GROUP BY year
)
SELECT year, total_sales,
       LAG(total_sales) OVER(ORDER BY year) AS prev_year_sales,
       ROUND(((total_sales - LAG(total_sales) OVER(ORDER BY year)) / LAG(total_sales) OVER(ORDER BY year)) * 100, 2)
       AS yoy_growth_percentage
FROM yearly_sales;
```



year	total_sales	prev_year_sales	yoy_growth_percentage
2016	59362.34	NULL	NULL
2017	7249746.73	59362.34	12112.70
2018	8699763.05	7249746.73	20.00

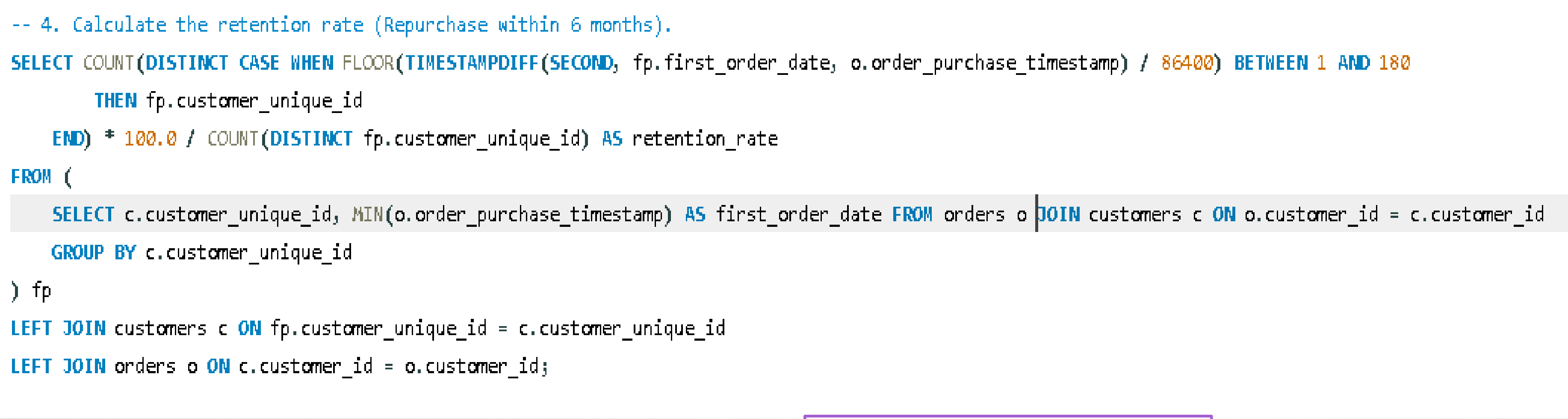
Python code



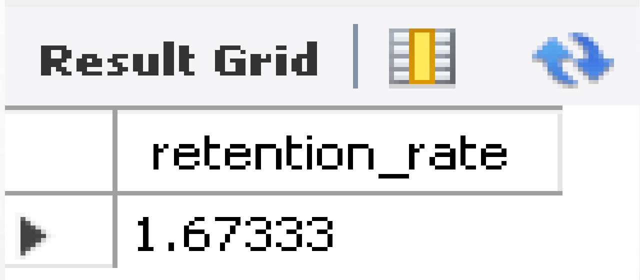
```python
# 5. Calculate YoY growth percentage
yearly_sales['yoy_growth_percentage'] = (
    (yearly_sales['total_sales'] - yearly_sales['prev_year_sales']) /
    yearly_sales['prev_year_sales']
) * 100

# 6. Round the growth percentage to 2 decimal places
yearly_sales['yoy_growth_percentage'] = yearly_sales['yoy_growth_percentage'].round(2)

# Display the result
print(yearly_sales)
```

```
...    year   total_sales   prev_year_sales   yoy_growth_percentage
0      2016      59362.34               NaN                     NaN
1      2017    7249746.73          59362.34                 12112.7
2      2018    8699763.05        7249746.73                    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.

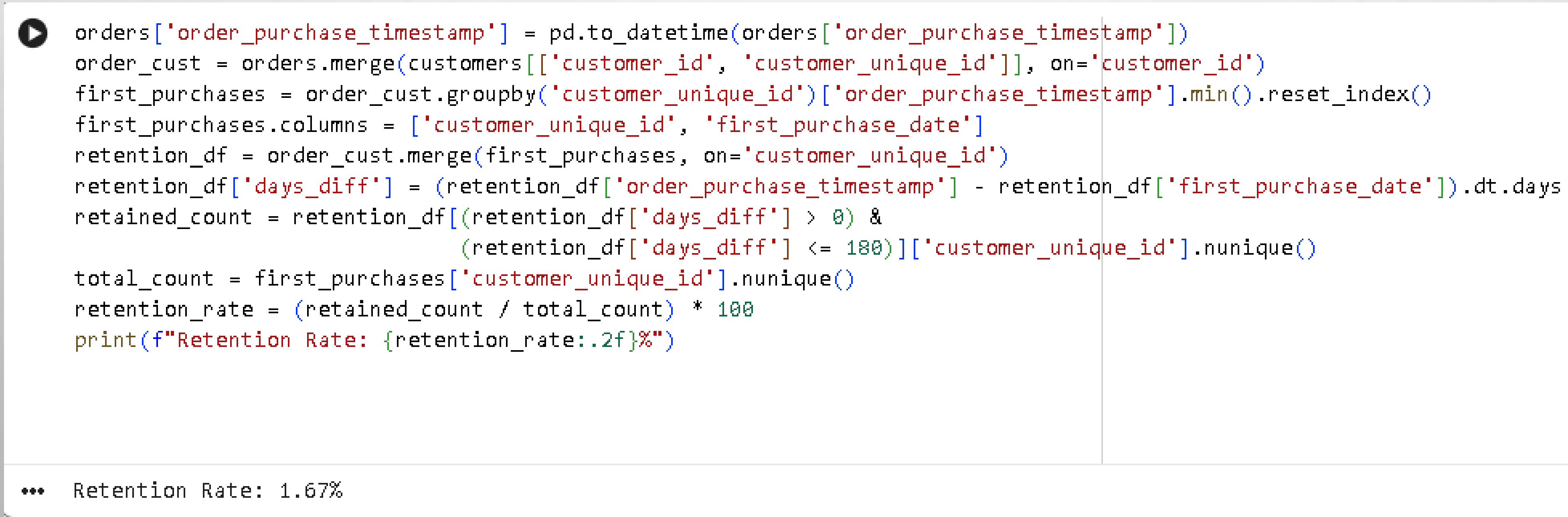


```sql
-- 4. Calculate the retention rate (Repurchase within 6 months).
SELECT COUNT(DISTINCT CASE WHEN FLOOR(TIMESTAMPDIFF(SECOND, fp.first_order_date, o.order_purchase_timestamp) / 86400) BETWEEN 1 AND 180
        THEN fp.customer_unique_id
    END) * 100.0 / COUNT(DISTINCT fp.customer_unique_id) AS retention_rate
FROM (
    SELECT c.customer_unique_id, MIN(o.order_purchase_timestamp) AS first_order_date FROM orders o JOIN customers c ON o.customer_id = c.customer_id
    GROUP BY c.customer_unique_id
) fp
LEFT JOIN customers c ON fp.customer_unique_id = c.customer_unique_id
LEFT JOIN orders o ON c.customer_id = o.customer_id;
```



	retention_rate
▶	1.67333

PYTHON CODE

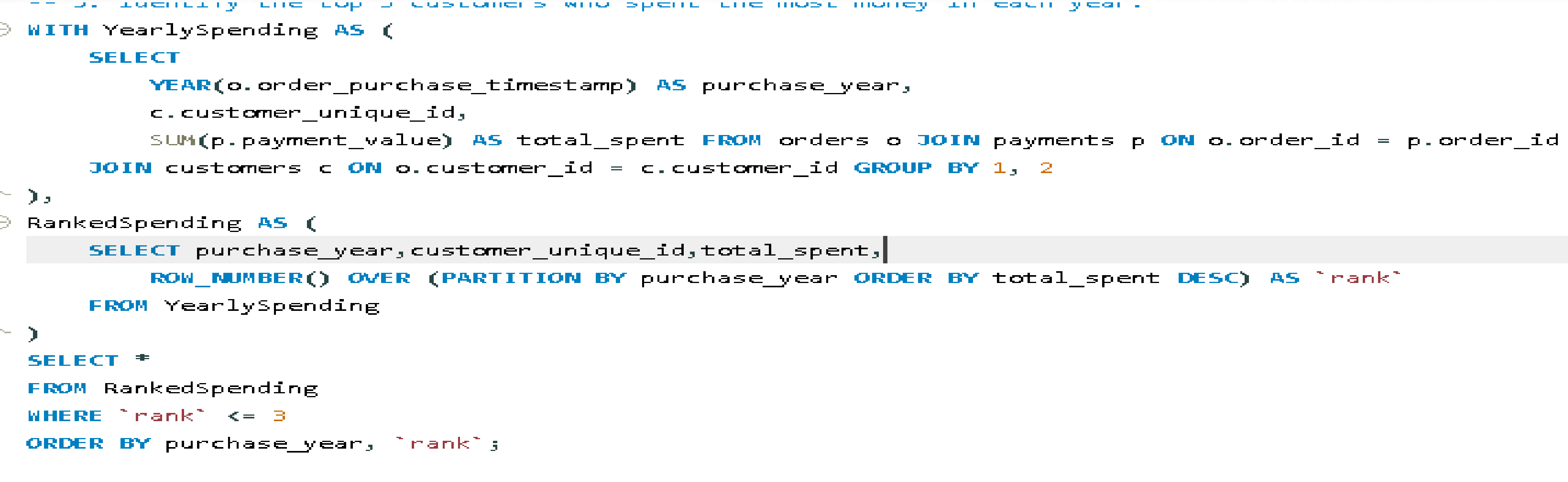


```python
orders['order_purchase_timestamp'] = pd.to_datetime(orders['order_purchase_timestamp'])
order_cust = orders.merge(customers[['customer_id', 'customer_unique_id']], on='customer_id')
first_purchases = order_cust.groupby('customer_unique_id')['order_purchase_timestamp'].min().reset_index()
first_purchases.columns = ['customer_unique_id', 'first_purchase_date']
retention_df = order_cust.merge(first_purchases, on='customer_unique_id')
retention_df['days_diff'] = (retention_df['order_purchase_timestamp'] - retention_df['first_purchase_date']).dt.days
retained_count = retention_df[(retention_df['days_diff'] > 0) &
                              (retention_df['days_diff'] <= 180)]['customer_unique_id'].nunique()
total_count = first_purchases['customer_unique_id'].nunique()
retention_rate = (retained_count / total_count) * 100
print(f"Retention Rate: {retention_rate:.2f}%")
```

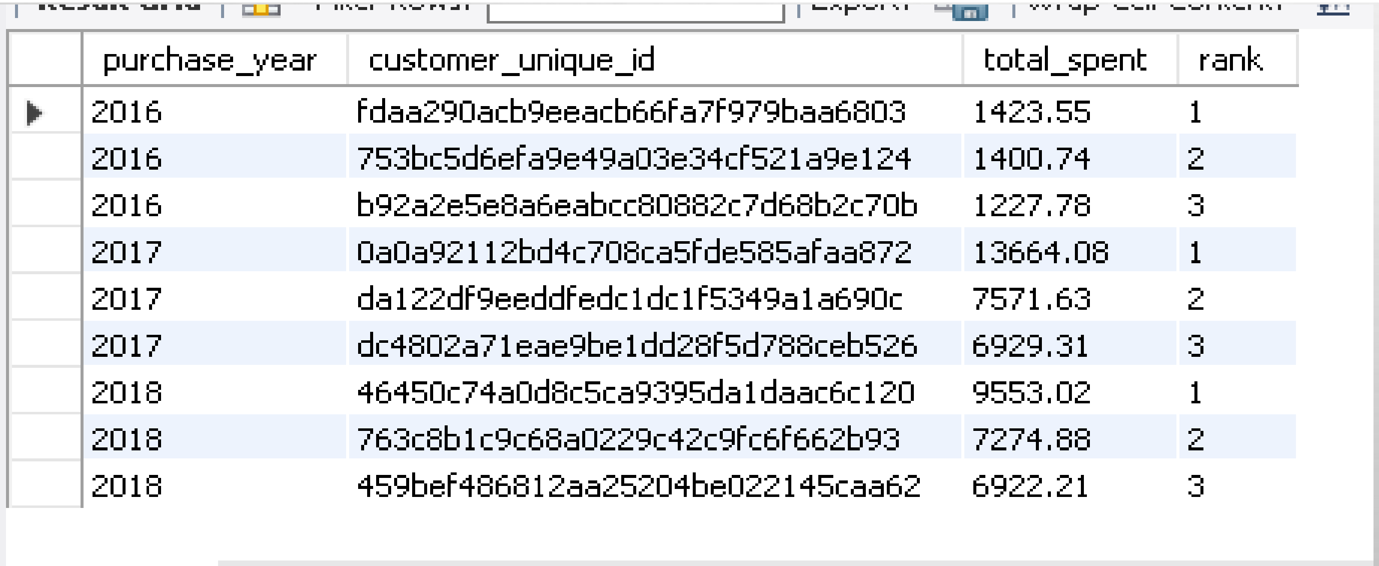
```
Retention Rate: 1.67%
```

IDENTIFY THE TOP 3 CUSTOMERS WHO SPENT THE MOST MONEY IN EACH YEAR.

SQL query

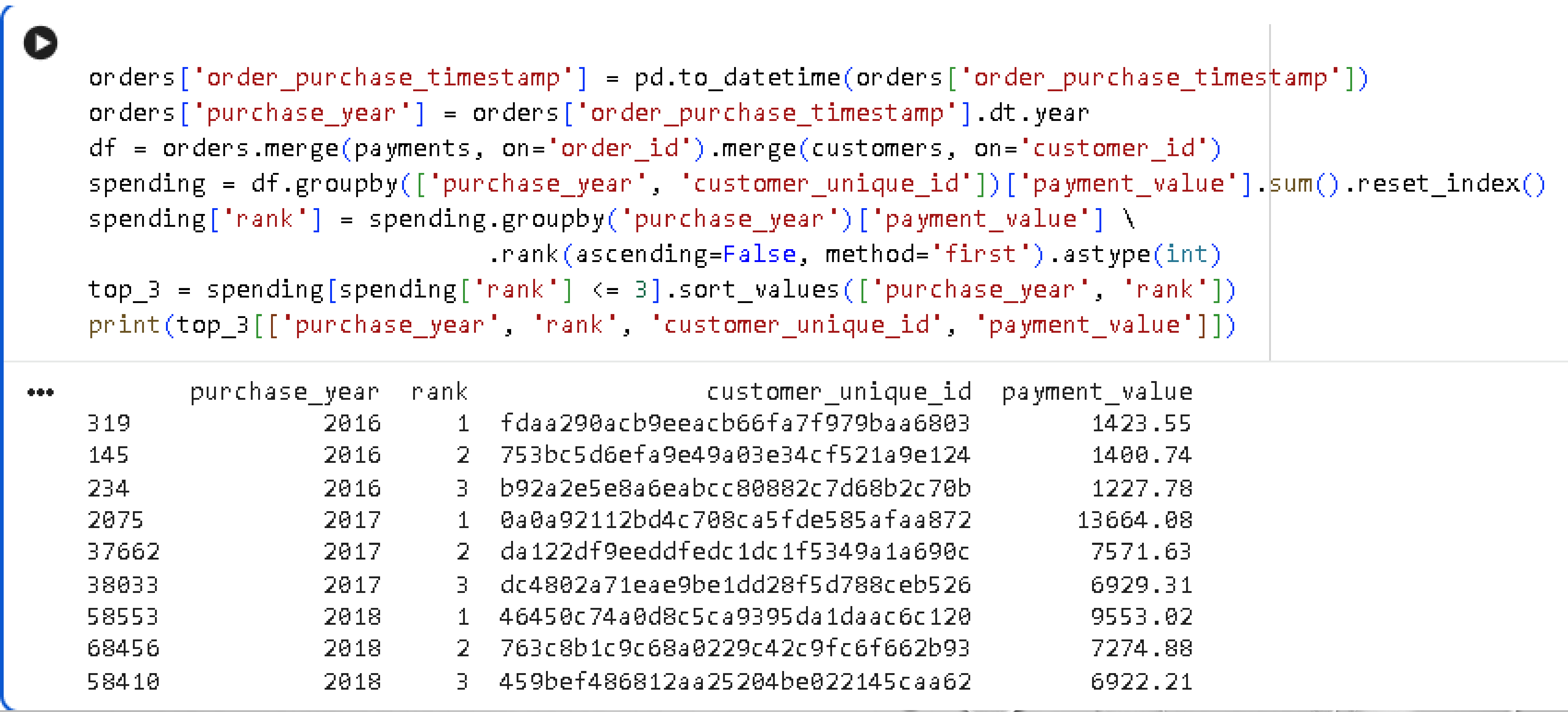


```sql
-- 3. Identify the top 3 customers who spent the most money in each year.
WITH YearlySpending AS (
    SELECT
        YEAR(o.order_purchase_timestamp) AS purchase_year,
        c.customer_unique_id,
        SUM(p.payment_value) AS total_spent FROM orders o JOIN payments p ON o.order_id = p.order_id
    JOIN customers c ON o.customer_id = c.customer_id GROUP BY 1, 2
),
RankedSpending AS (
    SELECT purchase_year,customer_unique_id,total_spent,
        ROW_NUMBER() OVER (PARTITION BY purchase_year ORDER BY total_spent DESC) AS `rank`
    FROM YearlySpending
)
SELECT *
FROM RankedSpending
WHERE `rank` <= 3
ORDER BY purchase_year, `rank`;
```



purchase_year	customer_unique_id	total_spent	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

PYTHON CODE



```python
orders['order_purchase_timestamp'] = pd.to_datetime(orders['order_purchase_timestamp'])
orders['purchase_year'] = orders['order_purchase_timestamp'].dt.year
df = orders.merge(payments, on='order_id').merge(customers, on='customer_id')
spending = df.groupby(['purchase_year', 'customer_unique_id'])['payment_value'].sum().reset_index()
spending['rank'] = spending.groupby('purchase_year')['payment_value'] \
                            .rank(ascending=False, method='first').astype(int)
top_3 = spending[spending['rank'] <= 3].sort_values(['purchase_year', 'rank'])
print(top_3[['purchase_year', 'rank', 'customer_unique_id', 'payment_value']])
```

```
       purchase_year  rank                customer_unique_id  payment_value
319             2016     1  fdaa290acb9eeacb66fa7f979baa6803        1423.55
145             2016     2  753bc5d6efa9e49a03e34cf521a9e124        1400.74
234             2016     3  b92a2e5e8a6eabcc80882c7d68b2c70b        1227.78
2075            2017     1  0a0a92112bd4c708ca5fde585afaa872       13664.08
37662           2017     2  da122df9eeddfedc1dc1f5349a1a690c        7571.63
38033           2017     3  dc4802a71eae9be1dd28f5d788ceb526        6929.31
58553           2018     1  46450c74a0d8c5ca9395da1daac6c120        9553.02
68456           2018     2  763c8b1c9c68a0229c42c9fc6f662b93        7274.88
58410           2018     3  459bef486812aa25204be022145caa62        6922.21
```