DATASET EXPLORATION: AN SQL AND PYTHON APPROACH

Presented by Hitesh Shetye

BASIC PROBLEMS

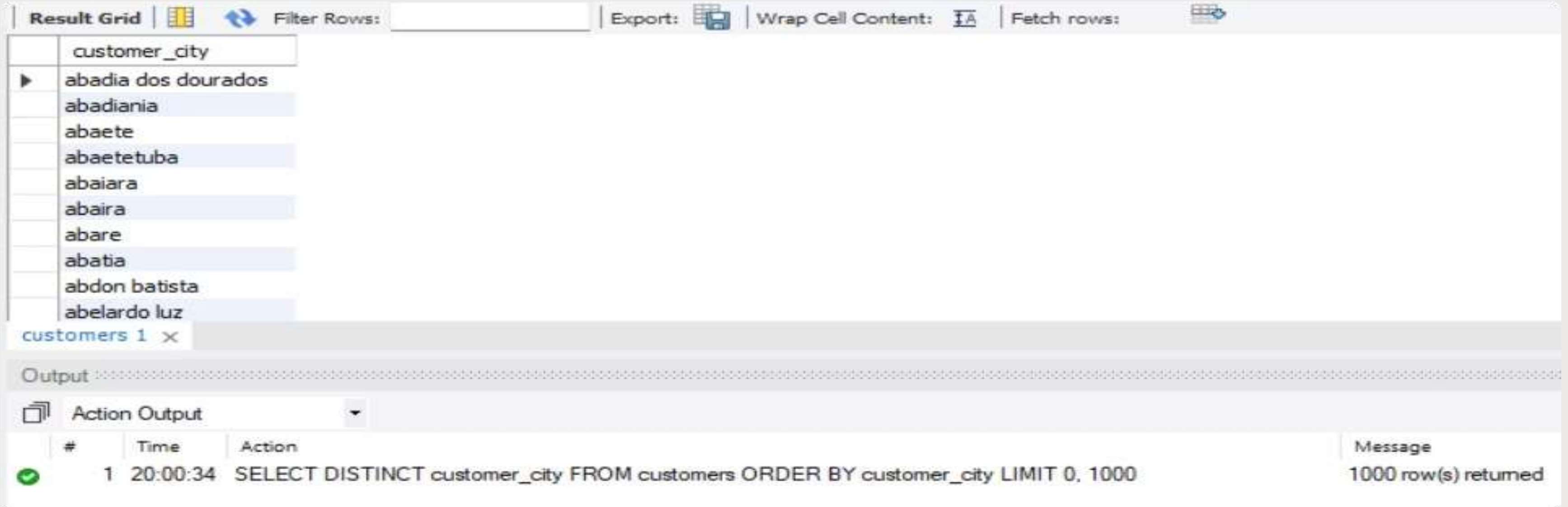
1. List all unique cities where customers are located

SQL

SELECT DISTINCT customer_city

FROM customers

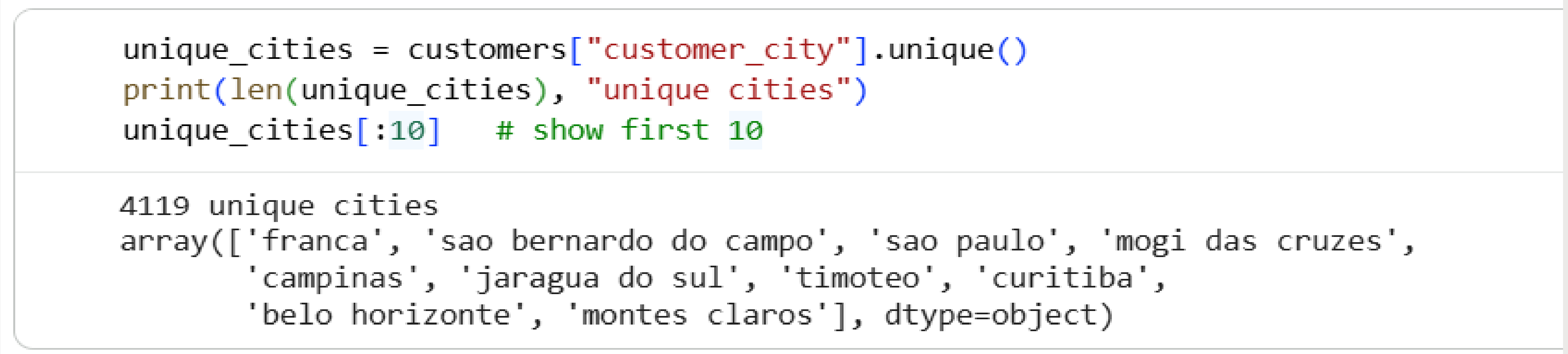
ORDER BY customer_city;



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

```
#   Time      Action                                                              Message
1   20:00:34  SELECT DISTINCT customer_city FROM customers ORDER BY customer_city LIMIT 0, 1000   1000 row(s) returned
```

PYTHON



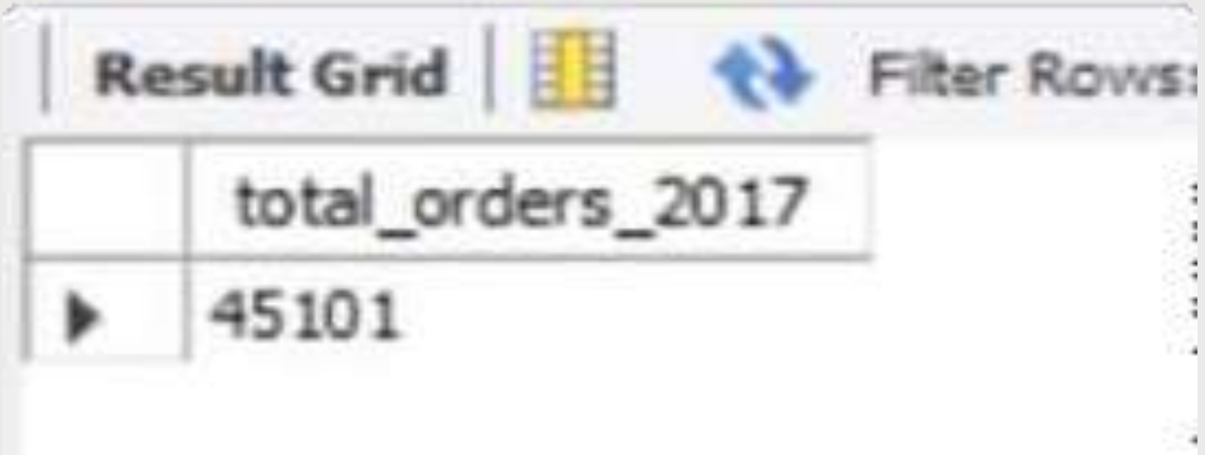
```
unique_cities = customers["customer_city"].unique()
print(len(unique_cities), "unique cities")
unique_cities[:10]   # show first 10
```

```
4119 unique cities
array(['franca', 'sao bernardo do campo', 'sao paulo', 'mogi das cruzes',
       'campinas', 'jaragua do sul', 'timoteo', 'curitiba',
       'belo horizonte', 'montes claros'], dtype=object)
```

2. Count the number of orders placed in 2017

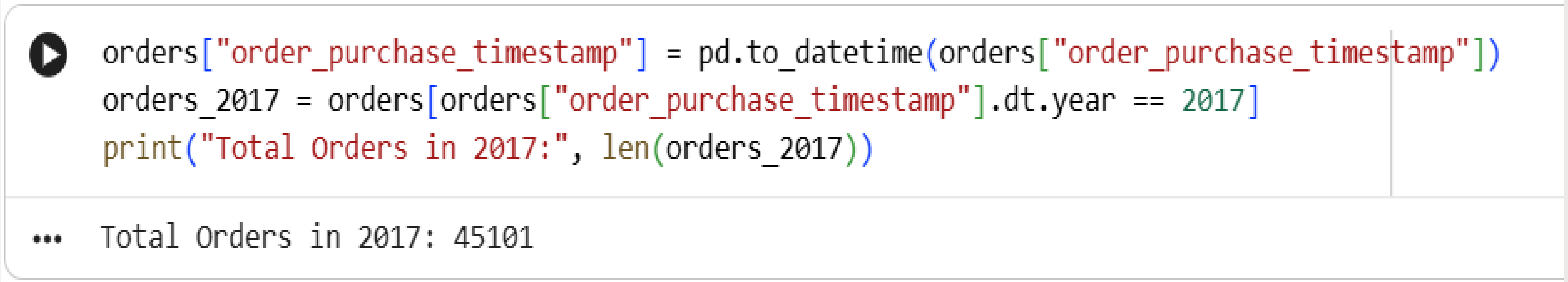
SQL

```
SELECT COUNT(*) AS total_orders_2017FROM orders

WHEREYEAR(order_purchase_timestamp) = 2017;
```



	total_orders_2017
▶	45101

PYTHON



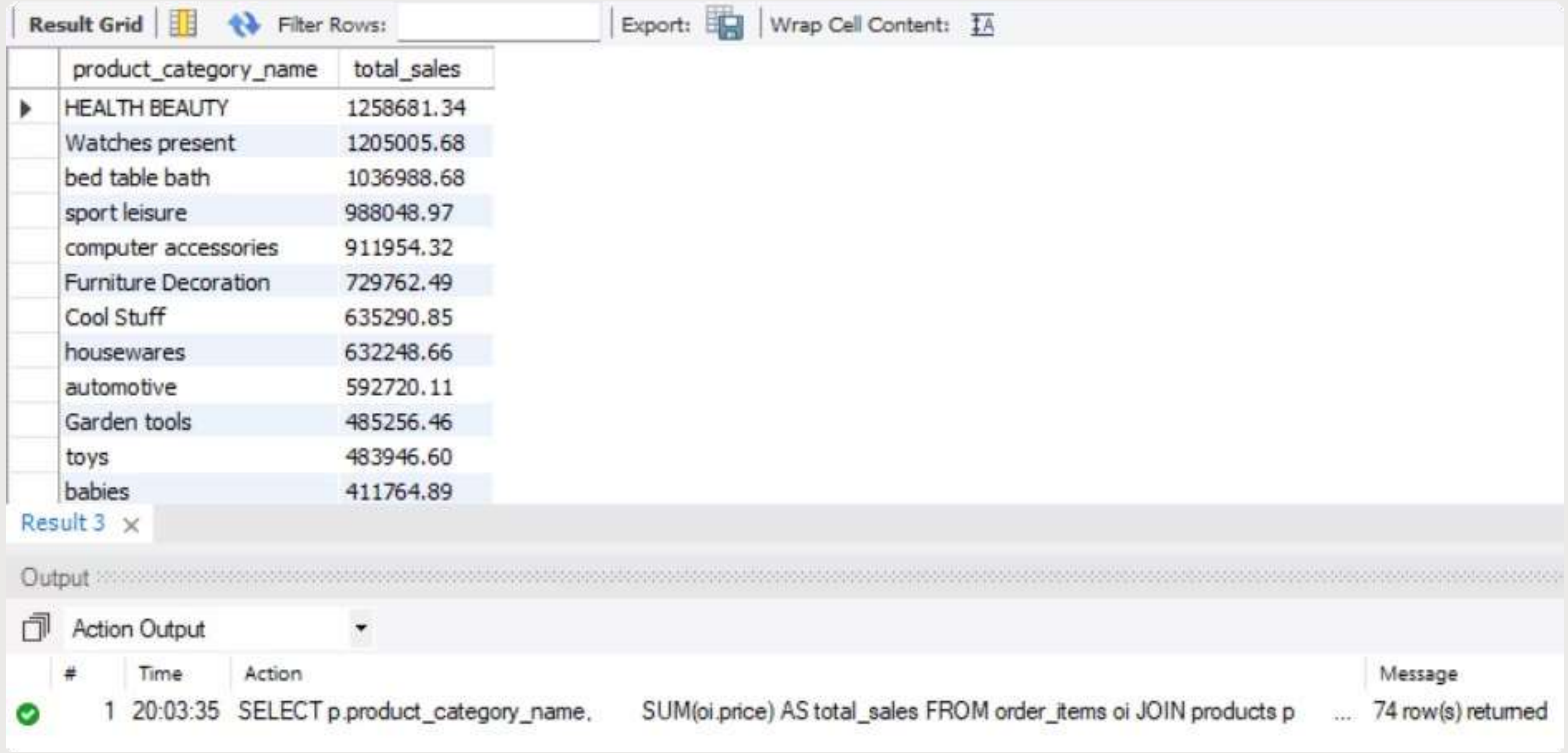
```
orders["order_purchase_timestamp"] = pd.to_datetime(orders["order_purchase_timestamp"])
orders_2017 = orders[orders["order_purchase_timestamp"].dt.year == 2017]
print("Total Orders in 2017:", len(orders_2017))
```

```
Total Orders in 2017: 45101
```

3. Find the total sales per category

SQL

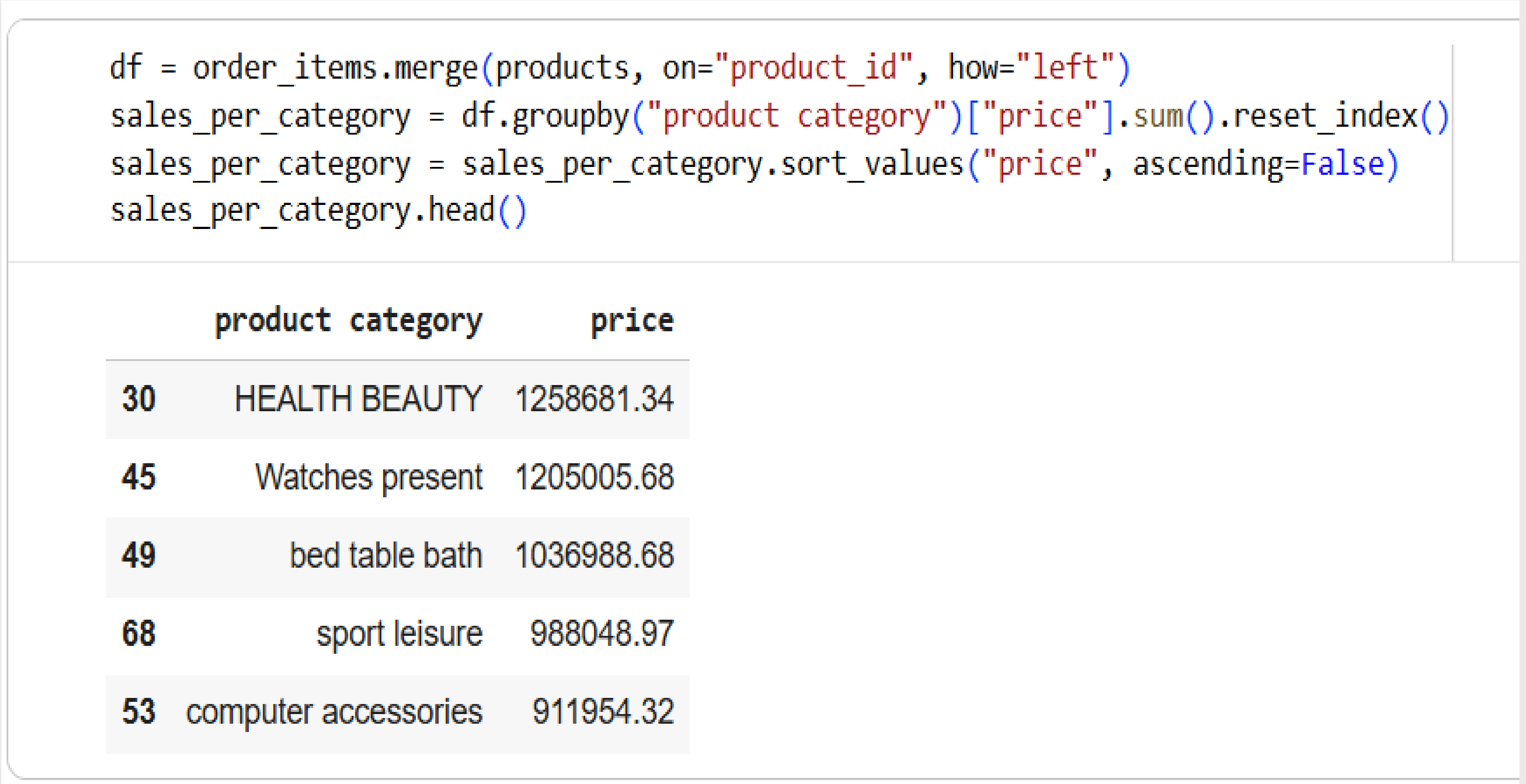
```
SELECT p.product_category_name,
       SUM(oi.price) AS total_sales
FROM order_items oi
JOIN products p
     ON oi.product_id = p.product_id
GROUP BY p.product_category_name
ORDER BY total_sales DESC;
```



product_category_name	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
Cool Stuff	635290.85
housewares	632248.66
automotive	592720.11
Garden tools	485256.46
toys	483946.60
babies	411764.89

```
#   Time      Action                                                                                        Message
1   20:03:35  SELECT p.product_category_name,    SUM(oi.price) AS total_sales FROM order_items oi JOIN products p ...   74 row(s) returned
```

PYTHON



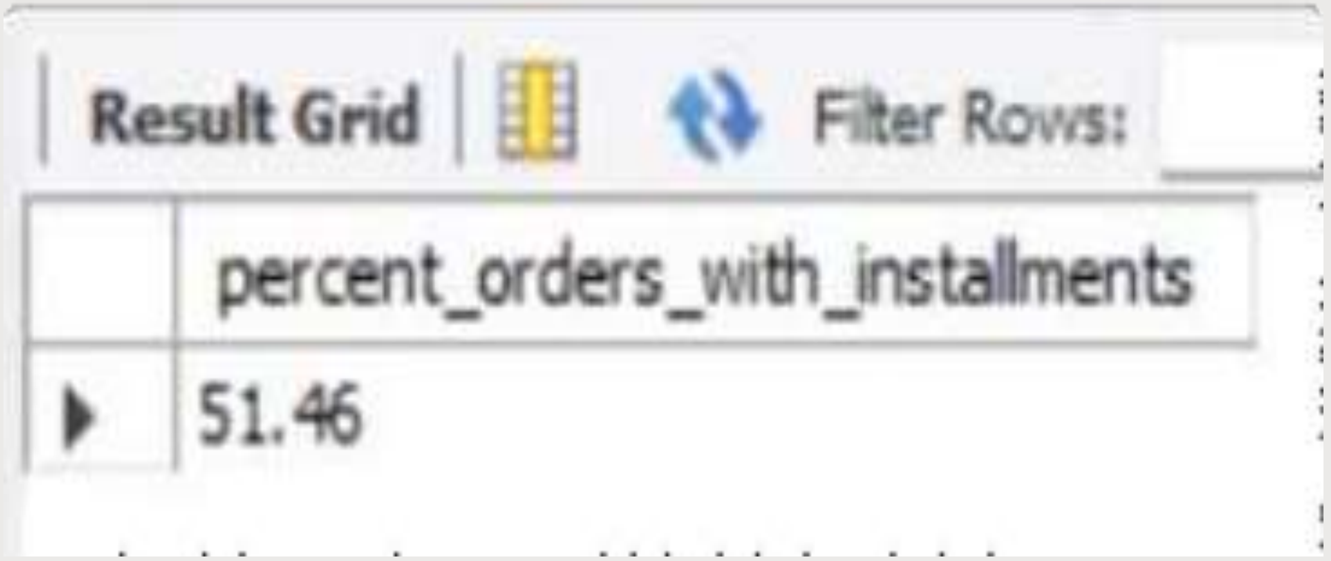
```python
df = order_items.merge(products, on="product_id", how="left")
sales_per_category = df.groupby("product category")["price"].sum().reset_index()
sales_per_category = sales_per_category.sort_values("price", ascending=False)
sales_per_category.head()
```

	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

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

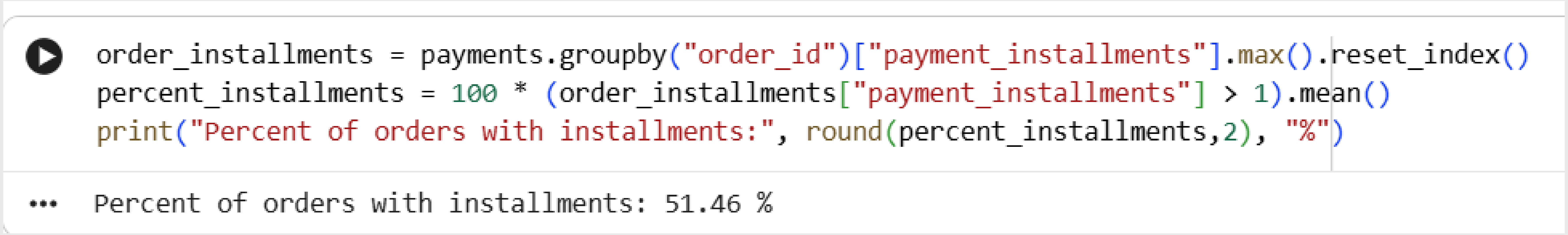
SQL

```
SELECT
 ROUND( 100.0 * COUNT(DISTINCT CASE WHEN payment_installments > 1 THEN order_id END)
  / COUNT(DISTINCT order_id)
 , 2) AS percent_orders_with_installments
FROM payments;
```



	percent_orders_with_installments
▶	51.46

PYTHON



```
order_installments = payments.groupby("order_id")["payment_installments"].max().reset_index()
percent_installments = 100 * (order_installments["payment_installments"] > 1).mean()
print("Percent of orders with installments:", round(percent_installments,2), "%")
```

```
Percent of orders with installments: 51.46 %
```

5. Count the number of customers from each state.

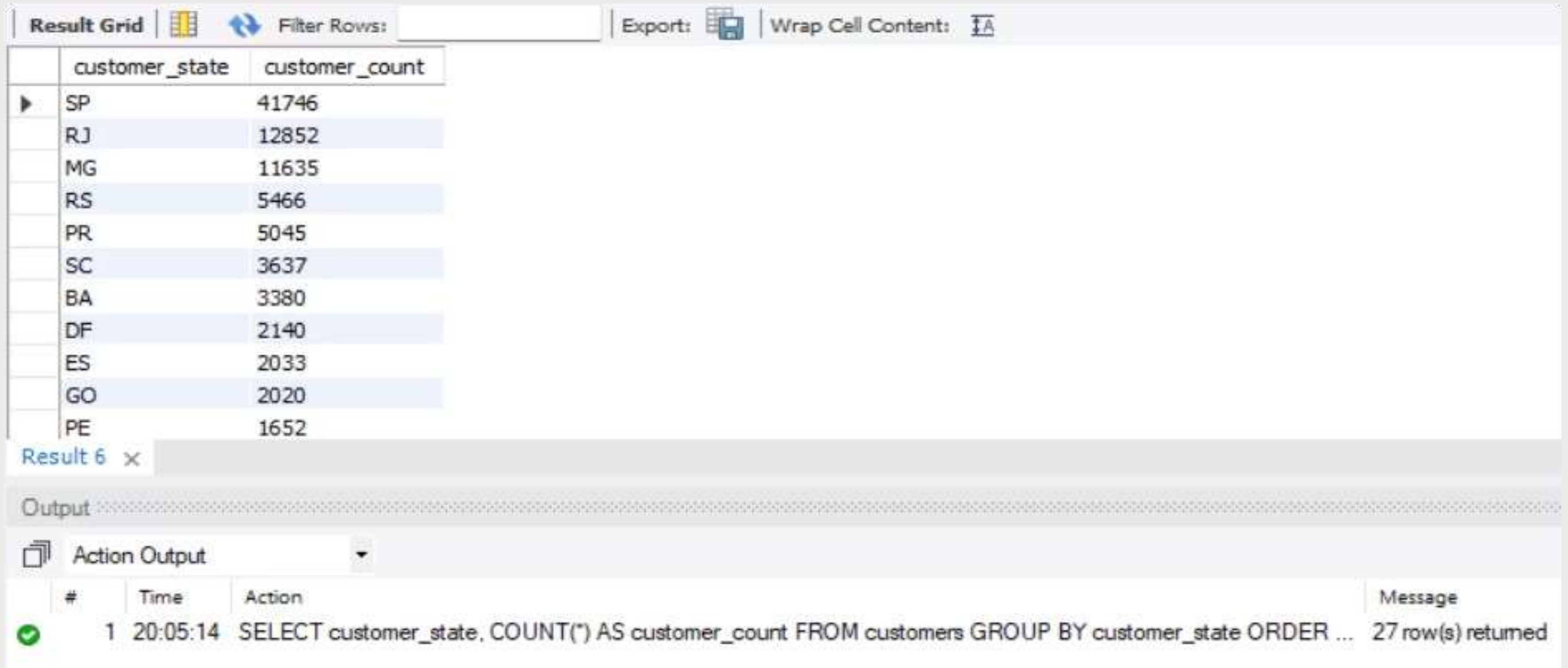
SQL

SELECT customer_state, COUNT(*) AS customer_count

FROM customers

GROUP BY customer_state

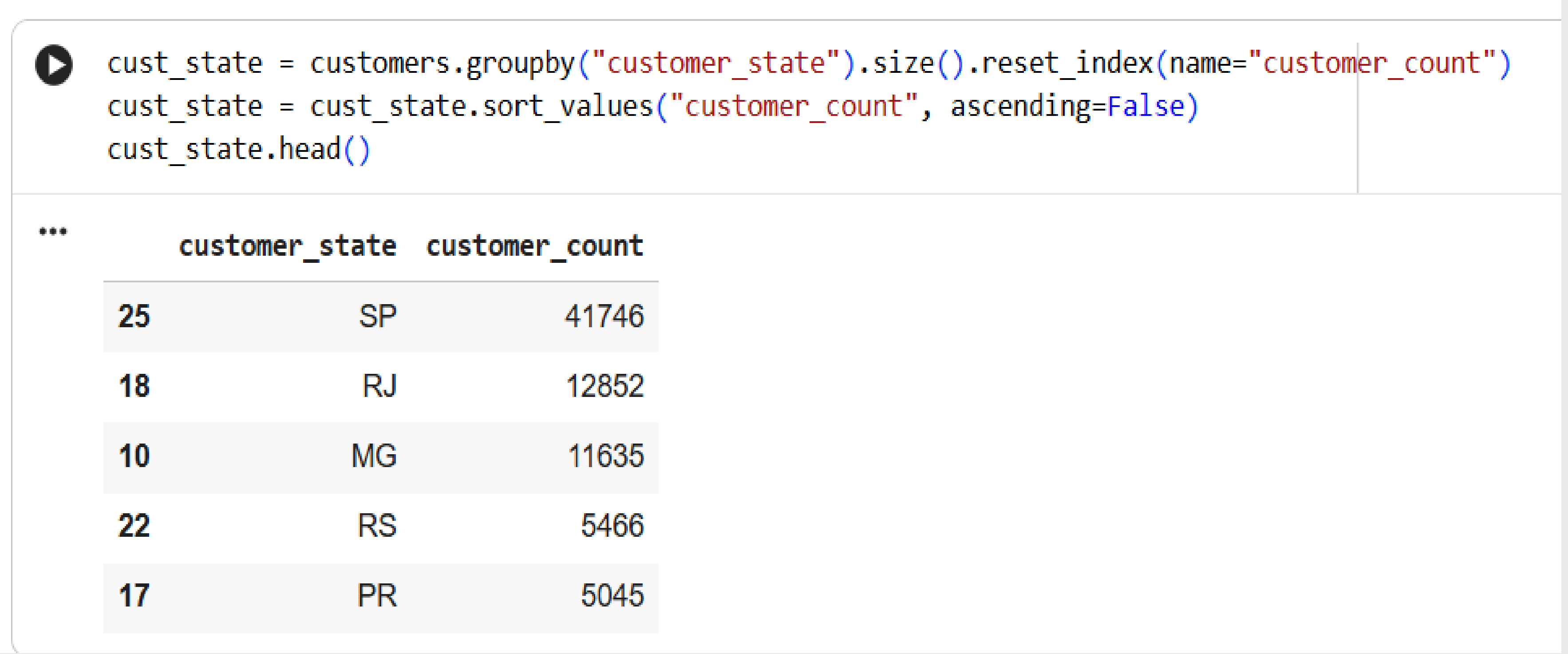
ORDER BY customer_count DESC;



customer_state	customer_count
SP	41746
RJ	12852
MG	11635
RS	5466
PR	5045
SC	3637
BA	3380
DF	2140
ES	2033
GO	2020
PE	1652

```
#   Time      Action                                                                                        Message
1   20:05:14  SELECT customer_state, COUNT(*) AS customer_count FROM customers GROUP BY customer_state ORDER ...  27 row(s) returned
```

PYTHON



```
cust_state = customers.groupby("customer_state").size().reset_index(name="customer_count")
cust_state = cust_state.sort_values("customer_count", ascending=False)
cust_state.head()
```

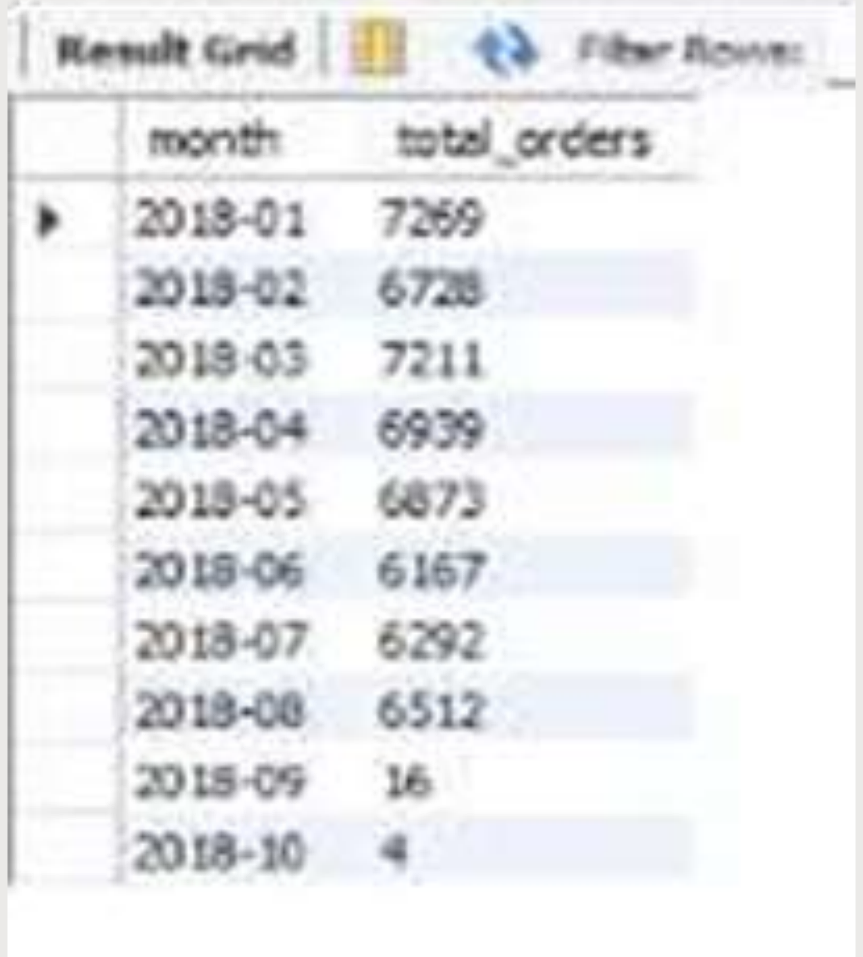
	customer_state	customer_count
25	SP	41746
18	RJ	12852
10	MG	11635
22	RS	5466
17	PR	5045

INTERMEDIATE PROBLEMS

1.Calculate the number of orders per month in 2018

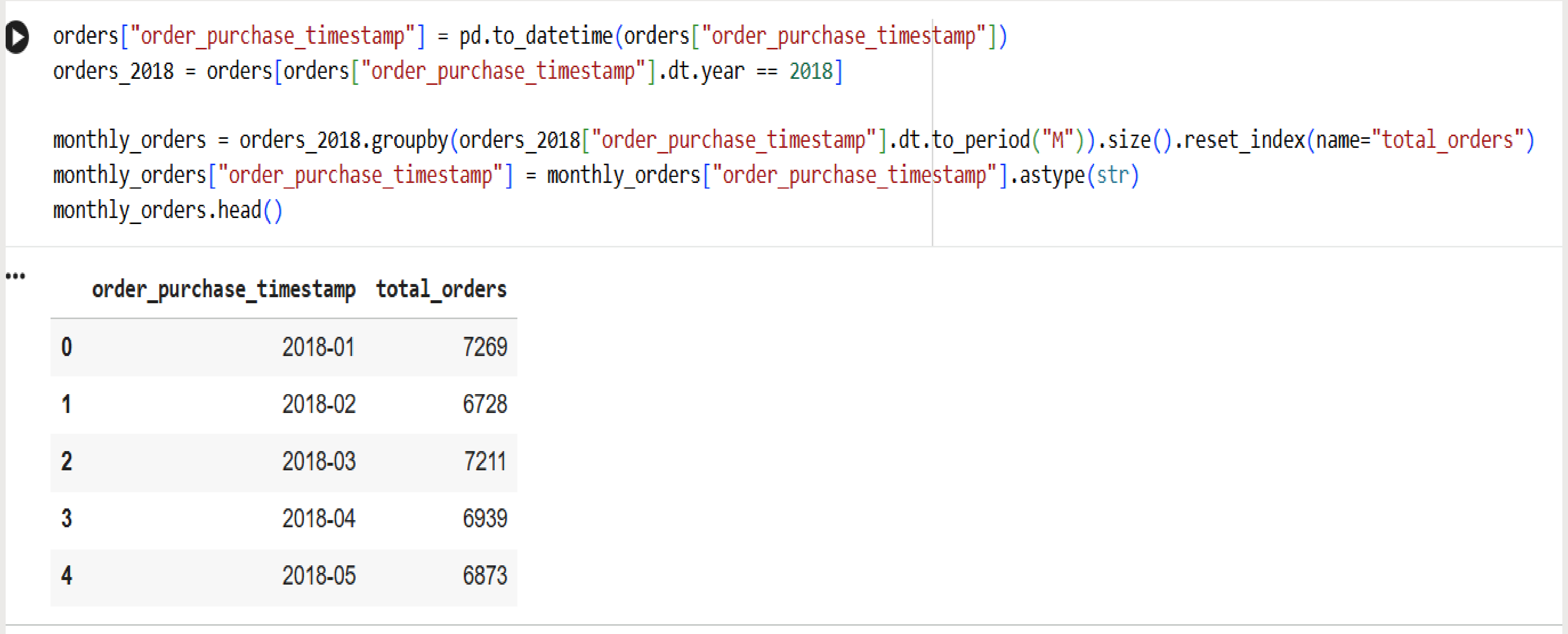
SQL

```
SELECT DATE_FORMAT(order_purchase_timestamp, '%Y-%m') AS month,
   COUNT(*) AS total_orders
FROM orders
WHERE YEAR(order_purchase_timestamp) = 2018
GROUP BY month
ORDER BY month;
```


Result Grid
Filter Rows:

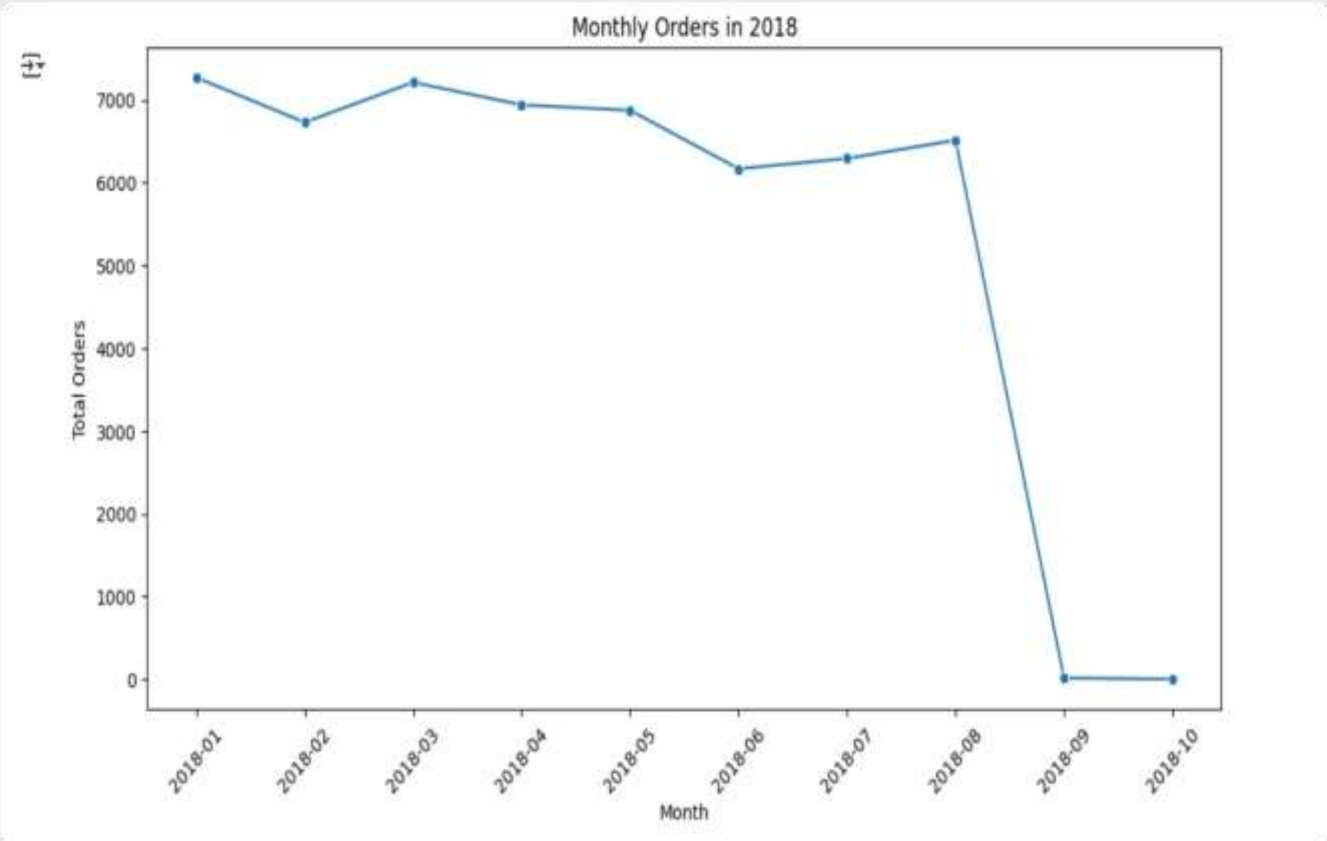
month	total_orders
2018-01	7269
2018-02	6728
2018-03	7211
2018-04	6939
2018-05	6873
2018-06	6167
2018-07	6292
2018-08	6512
2018-09	16
2018-10	4

PYTHON



```
orders["order_purchase_timestamp"] = pd.to_datetime(orders["order_purchase_timestamp"])
orders_2018 = orders[orders["order_purchase_timestamp"].dt.year == 2018]

monthly_orders = orders_2018.groupby(orders_2018["order_purchase_timestamp"].dt.to_period("M")).size().reset_index(name="total_orders")
monthly_orders["order_purchase_timestamp"] = monthly_orders["order_purchase_timestamp"].astype(str)
monthly_orders.head()
```

```
   order_purchase_timestamp  total_orders
0                   2018-01          7269
1                   2018-02          6728
2                   2018-03          7211
3                   2018-04          6939
4                   2018-05          6873
```



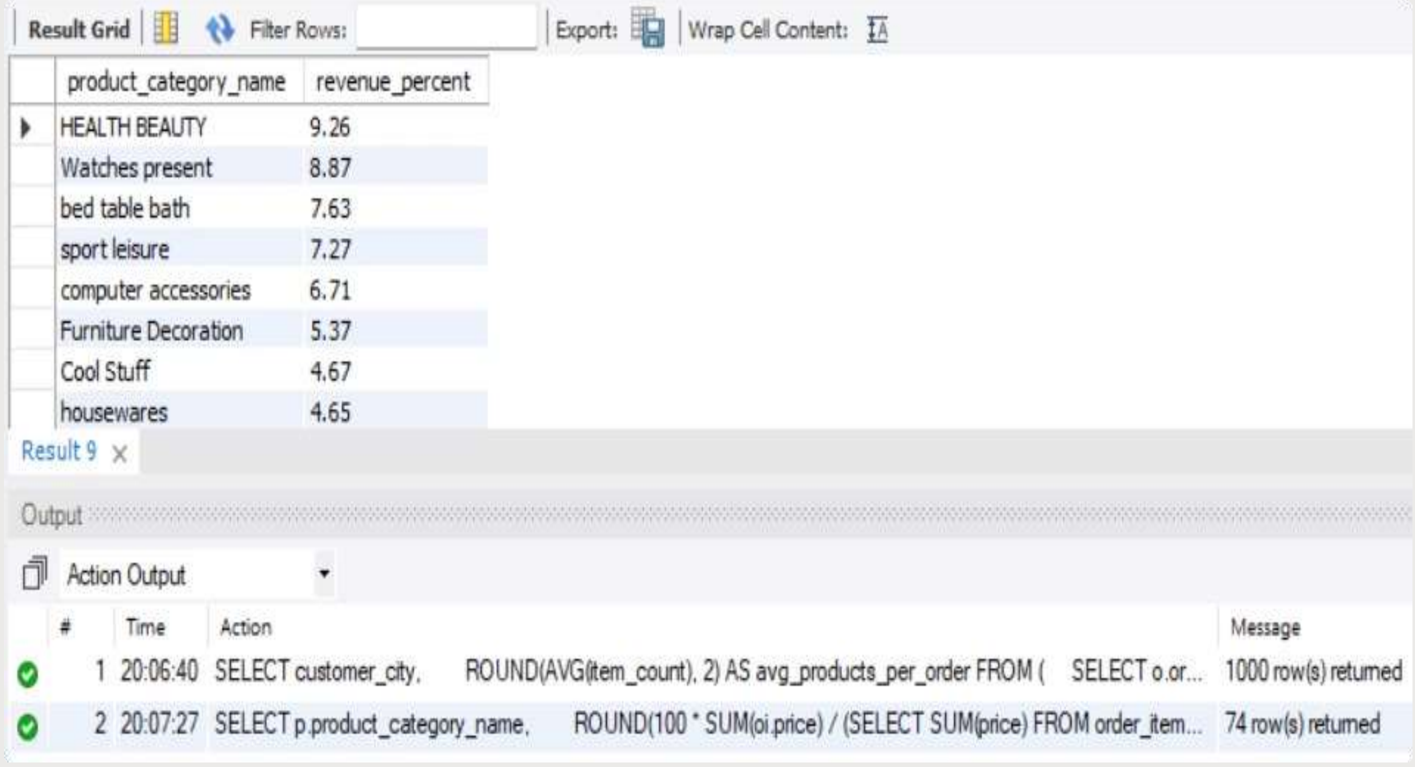
2. Find the average number of products per order, grouped by customer city

SQL

```
SELECT customer_city,
   ROUND(AVG(item_count), 2) AS avg_products_per_order
FROM (
  SELECT o.order_id, c.customer_city, COUNT(oi.product_id) AS item_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
) AS order_summary
GROUP BY customer_city
ORDER BY avg_products_per_order DESC;
```

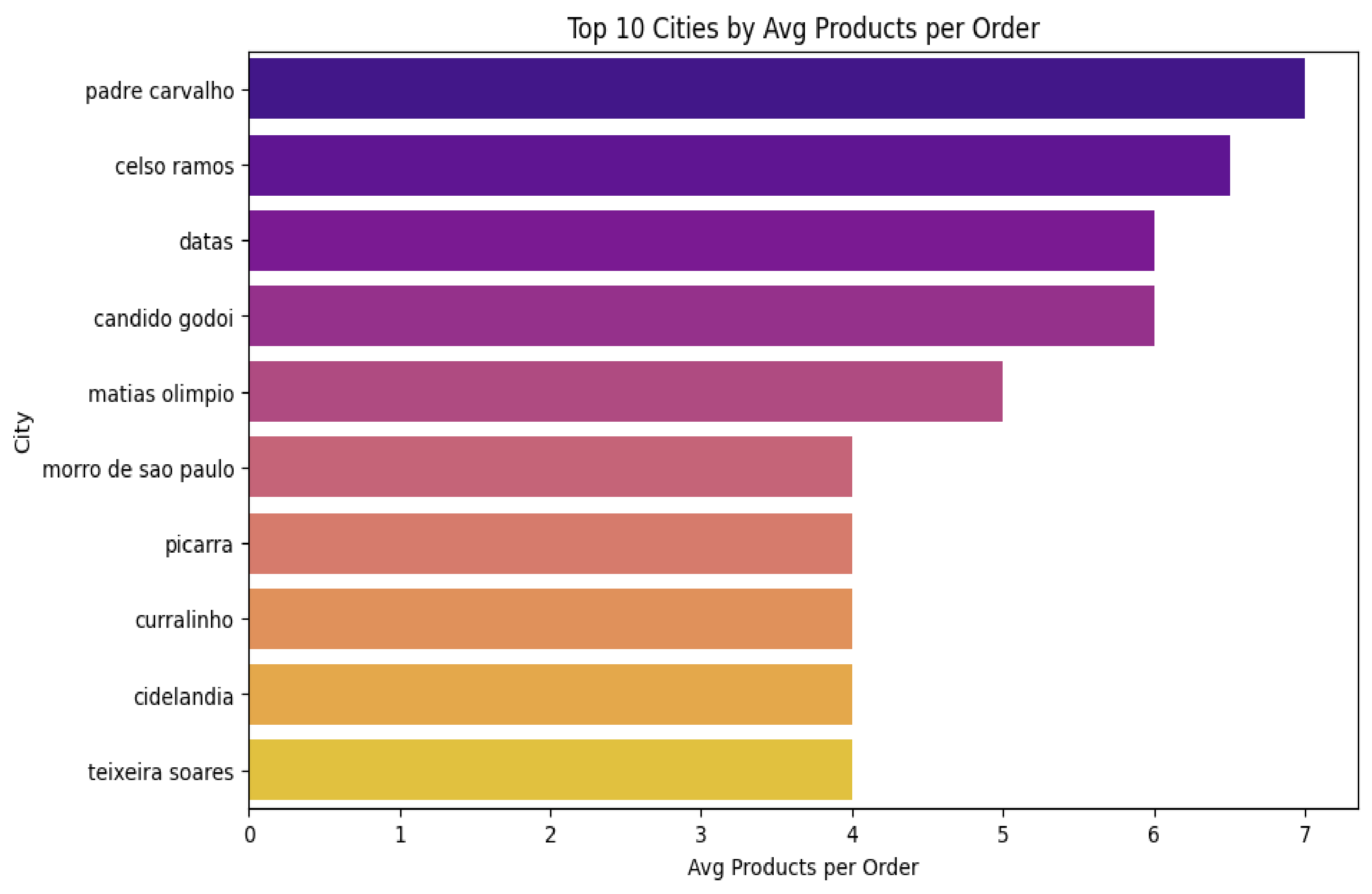
PYTHON

```
plt.figure(figsize=(10,6))
sns.barplot(data=avg_products_city.head(10), x="item_count", y="customer_city",
palette="plasma")
plt.title("Top 10 Cities by Avg Products per Order")
plt.xlabel("Avg Products per Order")
plt.ylabel("City")
plt.show()
```



product_category_name	revenue_percent
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

```
#  Time      Action                                                                                   Message
1  20:06:40  SELECT customer_city,     ROUND(AVG(item_count), 2) AS avg_products_per_order FROM (    SELECT o.or...   1000 row(s) returned
2  20:07:27  SELECT p.product_category_name,     ROUND(100 * SUM(oi.price) / (SELECT SUM(price) FROM order_item...   74 row(s) returned
```



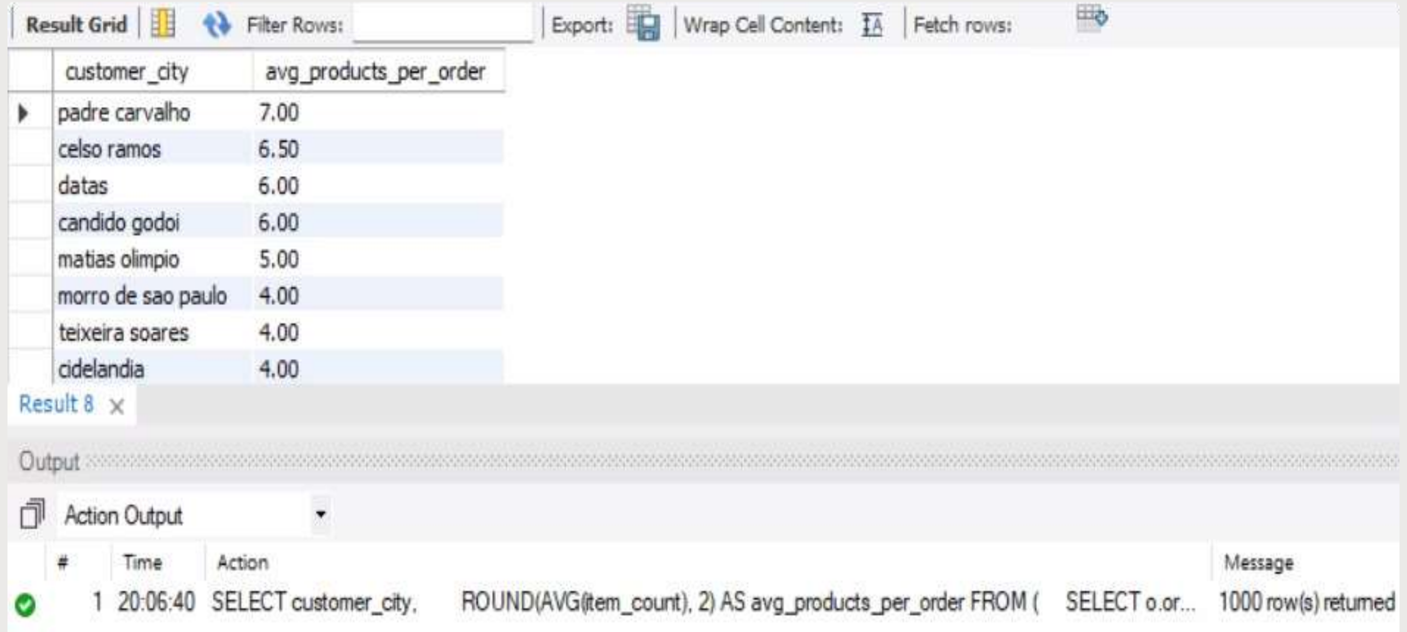
3. Calculate the percentage of total revenue contributed by each product category

SQL

```
SELECT p.product_category_name,
   ROUND(100 * SUM(oi.price) / (SELECT SUM(price) FROM order_items),2) AS revenue_percent
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
GROUP BY p.product_category_name
ORDER BY revenue_percent DESC;
```

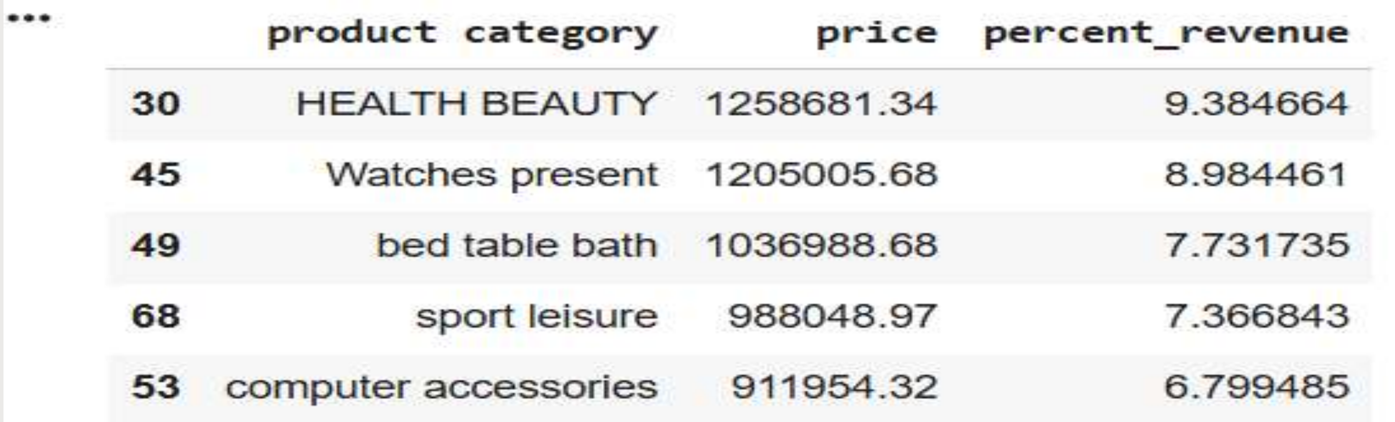
PYTHON

```
df = order_items.merge(products, on="product_id", how="left")
revenue_per_cat = df.groupby("product category")["price"].sum().reset_index()
total_revenue = revenue_per_cat["price"].sum()
revenue_per_cat["percent_revenue"] = 100 * revenue_per_cat["price"] / total_revenue
revenue_per_cat = revenue_per_cat.sort_values("percent_revenue", ascending=False)
revenue_per_cat.head()
```

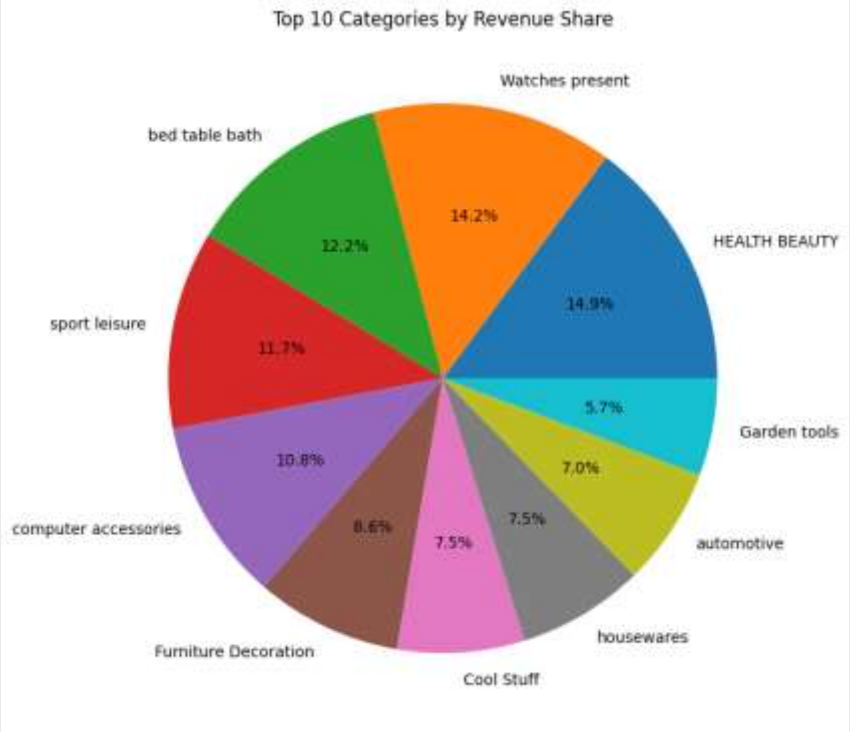


customer_city	avg_products_per_order
padre carvalho	7.00
celso ramos	6.50
datas	6.00
candido godoi	6.00
matias olimpio	5.00
morro de sao paulo	4.00
teixeira soares	4.00
cidelandia	4.00

```
#  Time      Action                                                                                         Message
1  20:06:40  SELECT customer_city,     ROUND(AVG(item_count), 2) AS avg_products_per_order FROM (     SELECT o.or...  1000 row(s) returned
```



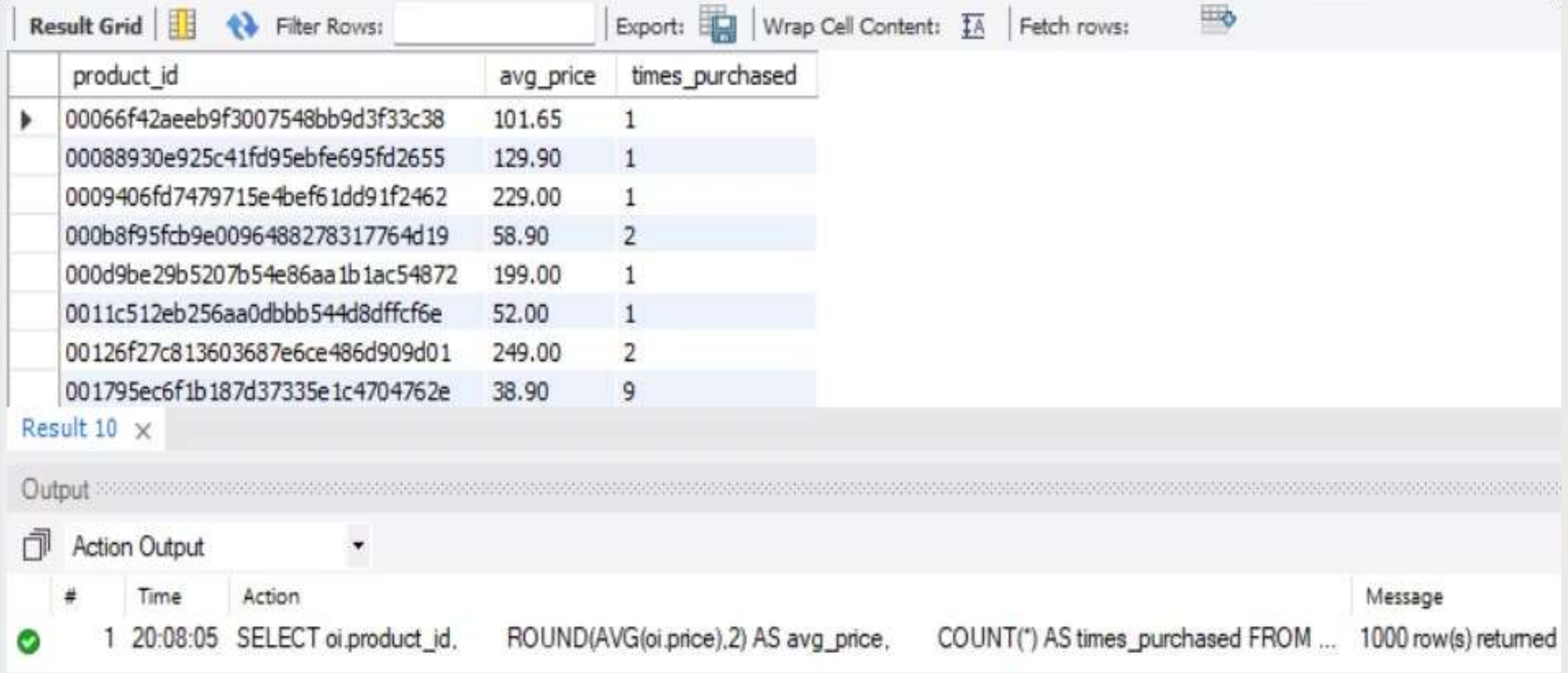
	product category	price	percent_revenue
30	HEALTH BEAUTY	1258681.34	9.384664
45	Watches present	1205005.68	8.984461
49	bed table bath	1036988.68	7.731735
68	sport leisure	988048.97	7.366843
53	computer accessories	911954.32	6.799485



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

SQL

```
SELECT oi.product_id,
    ROUND(AVG(oi.price),2) AS avg_price,
    COUNT(*) AS times_purchased
FROM order_items oi
GROUP BY oi.product_id;
```

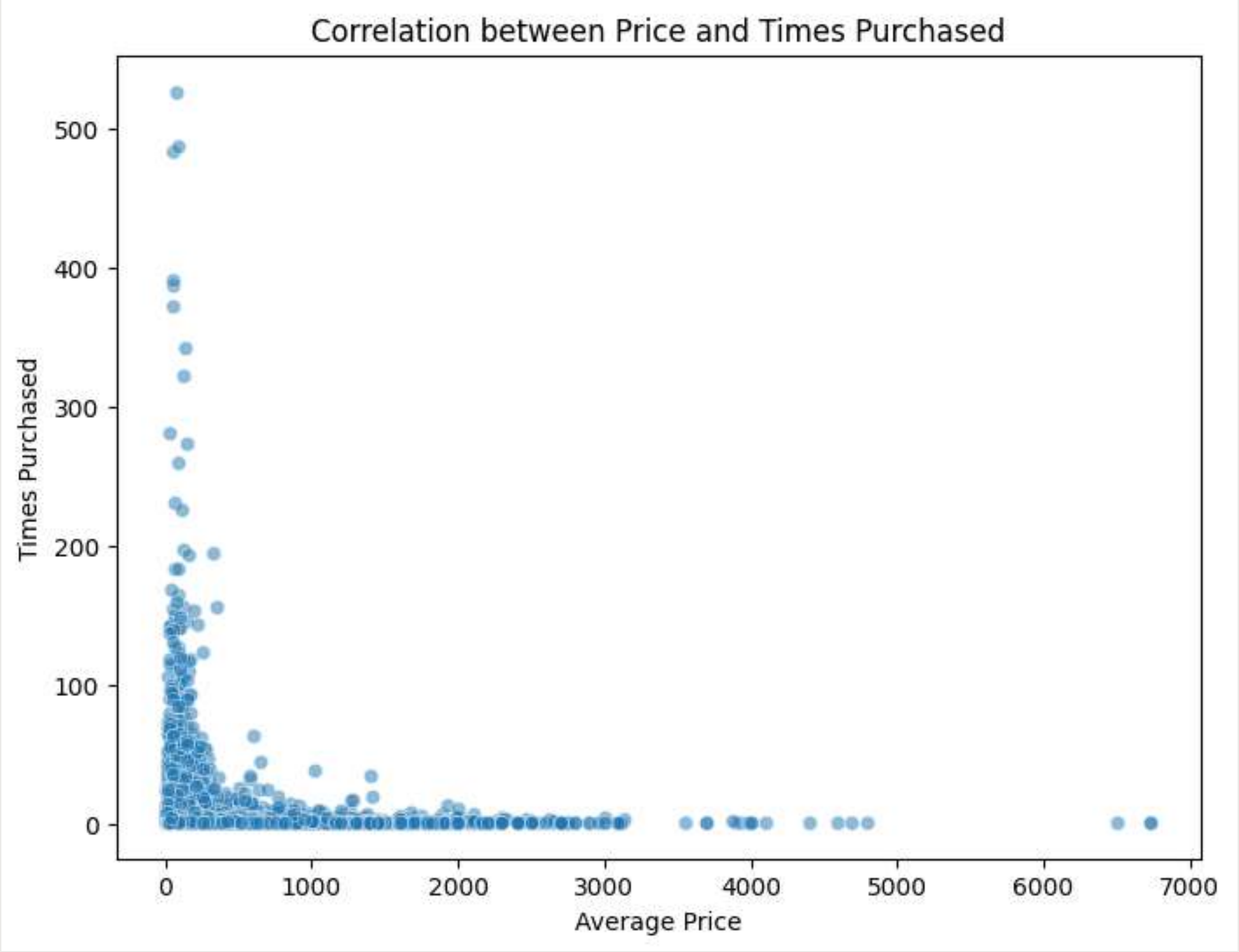


product_id	avg_price	times_purchased
00066f42aeeb9f3007548bb9d3f33c38	101.65	1
00088930e925c41fd95ebfe695fd2655	129.90	1
0009406fd7479715e4bef61dd91f2462	229.00	1
000b8f95fcb9e0096488278317764d19	58.90	2
000d9be29b5207b54e86aa1b1ac54872	199.00	1
0011c512eb256aa0dbbb544d8dffcf6e	52.00	1
00126f27c813603687e6ce486d909d01	249.00	2
001795ec6f1b187d37335e1c4704762e	38.90	9

```
#  Time      Action                                                                                          Message
1  20:08:05  SELECT oi.product_id,    ROUND(AVG(oi.price),2) AS avg_price,    COUNT(*) AS times_purchased FROM ...  1000 row(s) returned
```

PYTHON

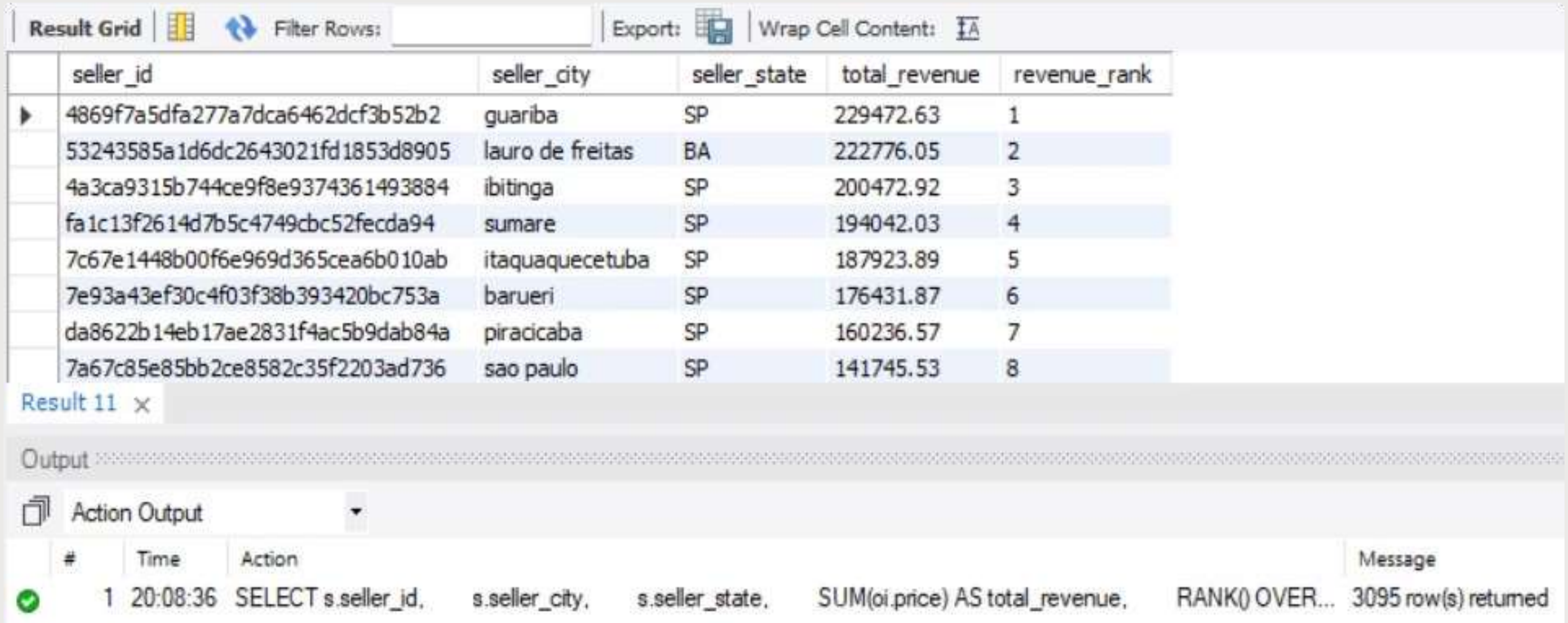
```
plt.figure(figsize=(8,6))
sns.scatterplot(data=product_stats, x="avg_price", y="times_purchased", alpha=0.5)
plt.title("Correlation between Price and Times Purchased")
plt.xlabel("Average Price")
plt.ylabel("Times Purchased")
plt.show()
```



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

SQL

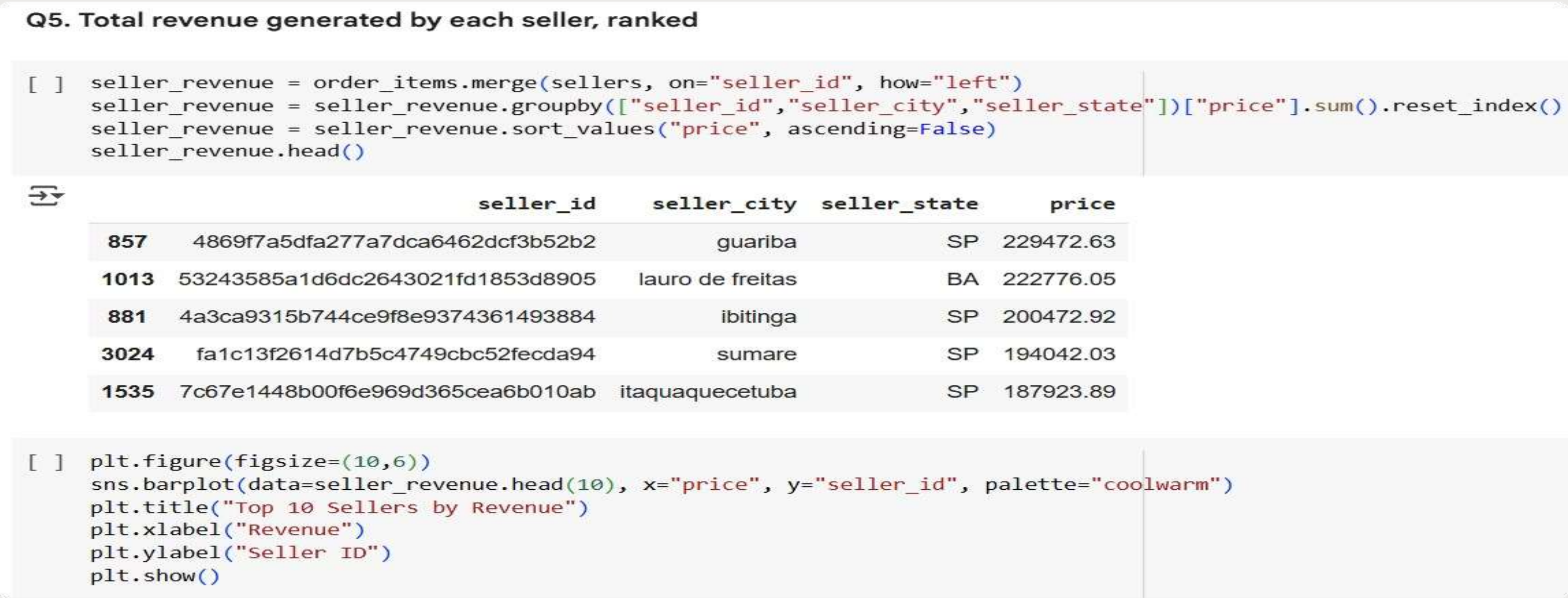
```sql
SELECT s.seller_id,
   s.seller_city,
   s.seller_state,
   SUM(oi.price) AS total_revenue,
   RANK() OVER (ORDER BY SUM(oi.price) DESC) AS revenue_rank
FROM order_items oi
JOIN sellers s ON oi.seller_id = s.seller_id
GROUP BY s.seller_id, s.seller_city, s.seller_state
ORDER BY total_revenue DESC;
```



seller_id	seller_city	seller_state	total_revenue	revenue_rank
4869f7a5dfa277a7dca6462dcf3b52b2	guariba	SP	229472.63	1
53243585a1d6dc2643021fd1853d8905	lauro de freitas	BA	222776.05	2
4a3ca9315b744ce9f8e9374361493884	ibitinga	SP	200472.92	3
fa1c13f2614d7b5c4749cbc52fecda94	sumare	SP	194042.03	4
7c67e1448b00f6e969d365cea6b010ab	itaquaquecetuba	SP	187923.89	5
7e93a43ef30c4f03f38b393420bc753a	barueri	SP	176431.87	6
da8622b14eb17ae2831f4ac5b9dab84a	piracicaba	SP	160236.57	7
7a67c85e85bb2ce8582c35f2203ad736	sao paulo	SP	141745.53	8

#	Time	Action	Message
1	20:08:36	SELECT s.seller_id, s.seller_city, s.seller_state, SUM(oi.price) AS total_revenue, RANK() OVER...	3095 row(s) returned

PYTHON

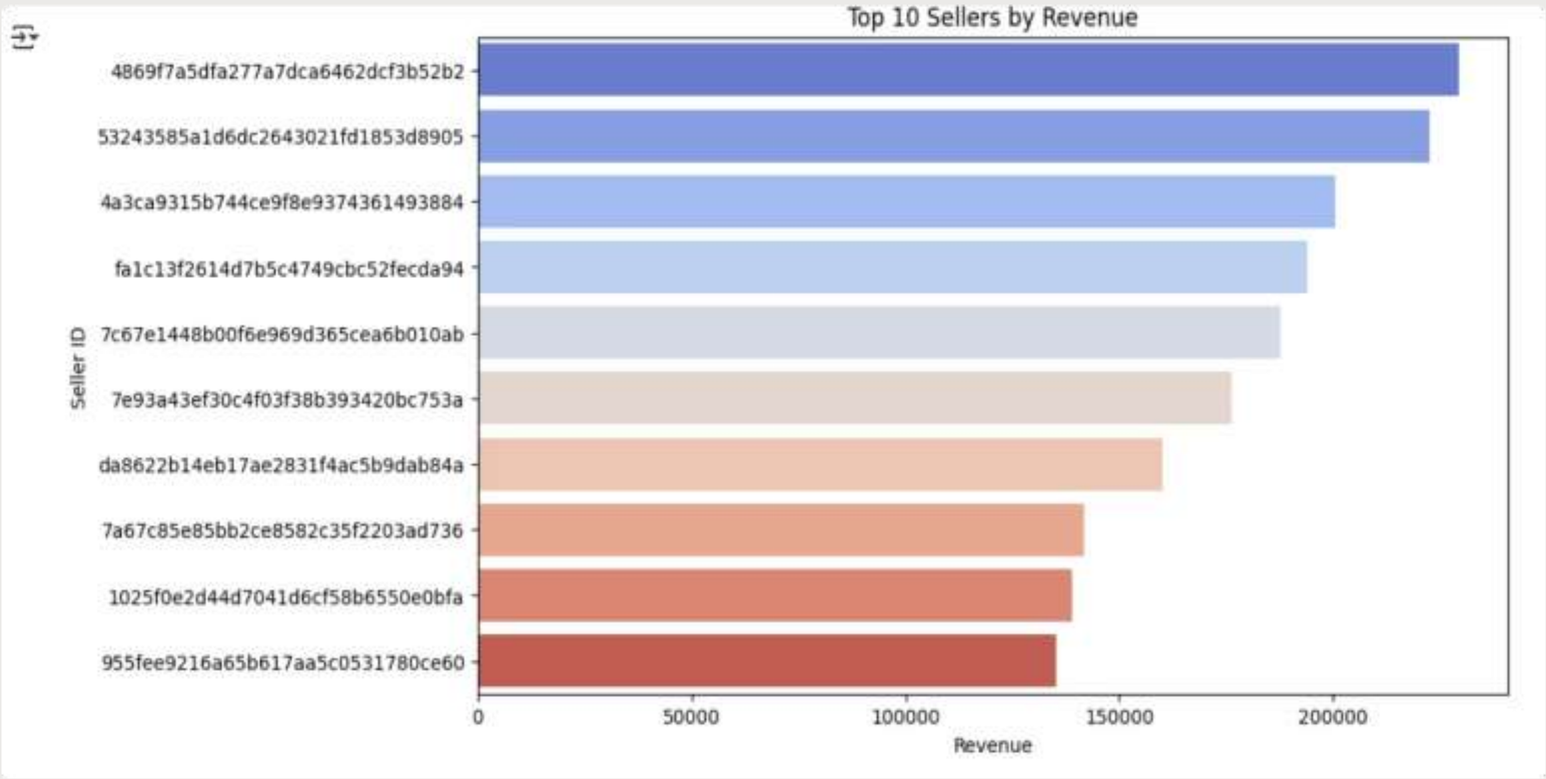


Q5. Total revenue generated by each seller, ranked

```python
[ ]  seller_revenue = order_items.merge(sellers, on="seller_id", how="left")
     seller_revenue = seller_revenue.groupby(["seller_id","seller_city","seller_state"])["price"].sum().reset_index()
     seller_revenue = seller_revenue.sort_values("price", ascending=False)
     seller_revenue.head()
```

	seller_id	seller_city	seller_state	price
857	4869f7a5dfa277a7dca6462dcf3b52b2	guariba	SP	229472.63
1013	53243585a1d6dc2643021fd1853d8905	lauro de freitas	BA	222776.05
881	4a3ca9315b744ce9f8e9374361493884	ibitinga	SP	200472.92
3024	fa1c13f2614d7b5c4749cbc52fecda94	sumare	SP	194042.03
1535	7c67e1448b00f6e969d365cea6b010ab	itaquaquecetuba	SP	187923.89

```python
[ ]  plt.figure(figsize=(10,6))
     sns.barplot(data=seller_revenue.head(10), x="price", y="seller_id", palette="coolwarm")
     plt.title("Top 10 Sellers by Revenue")
     plt.xlabel("Revenue")
     plt.ylabel("Seller ID")
     plt.show()
```

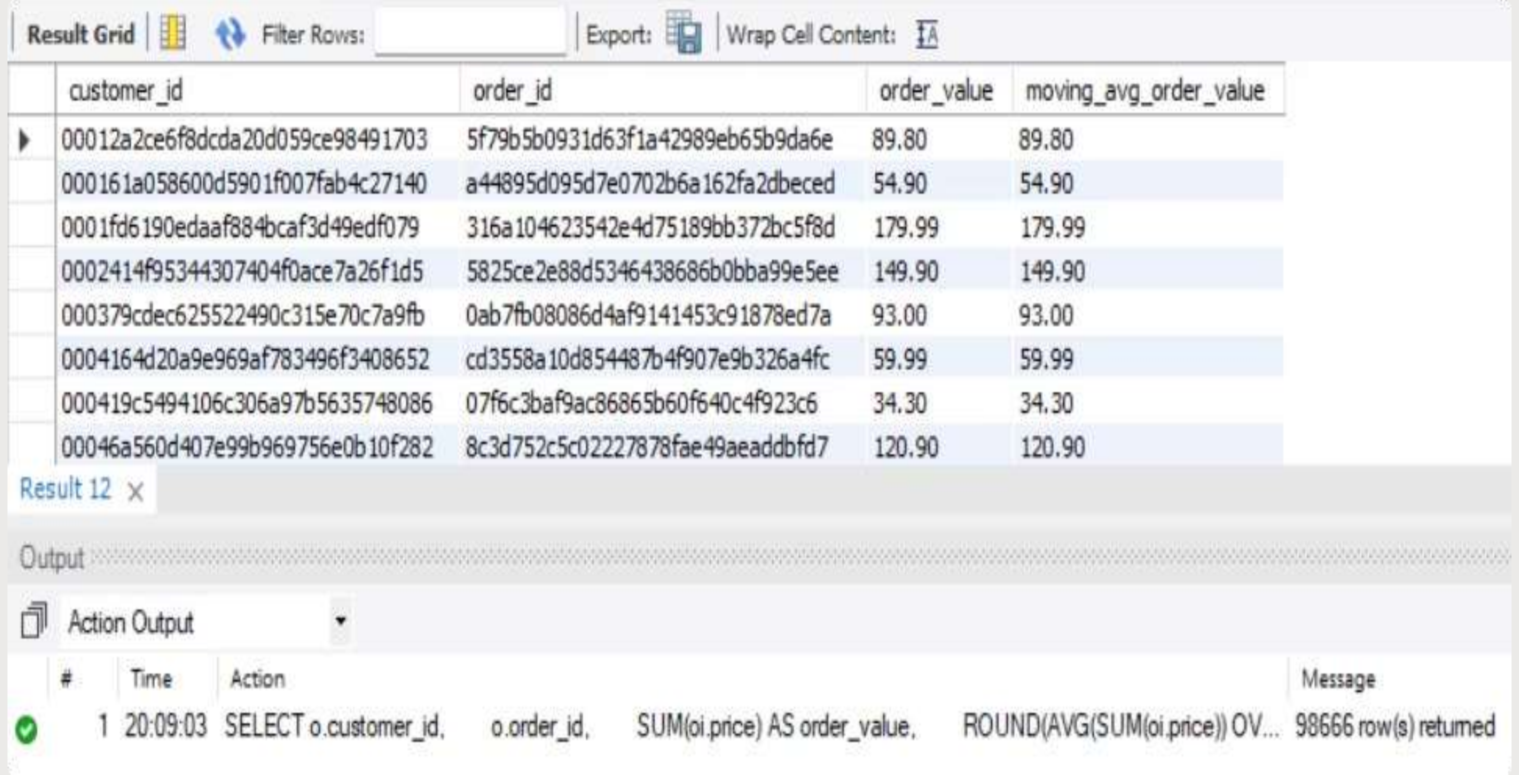


ADVANCED PROBLEMS

1. Moving average of order values for each customer

SQL

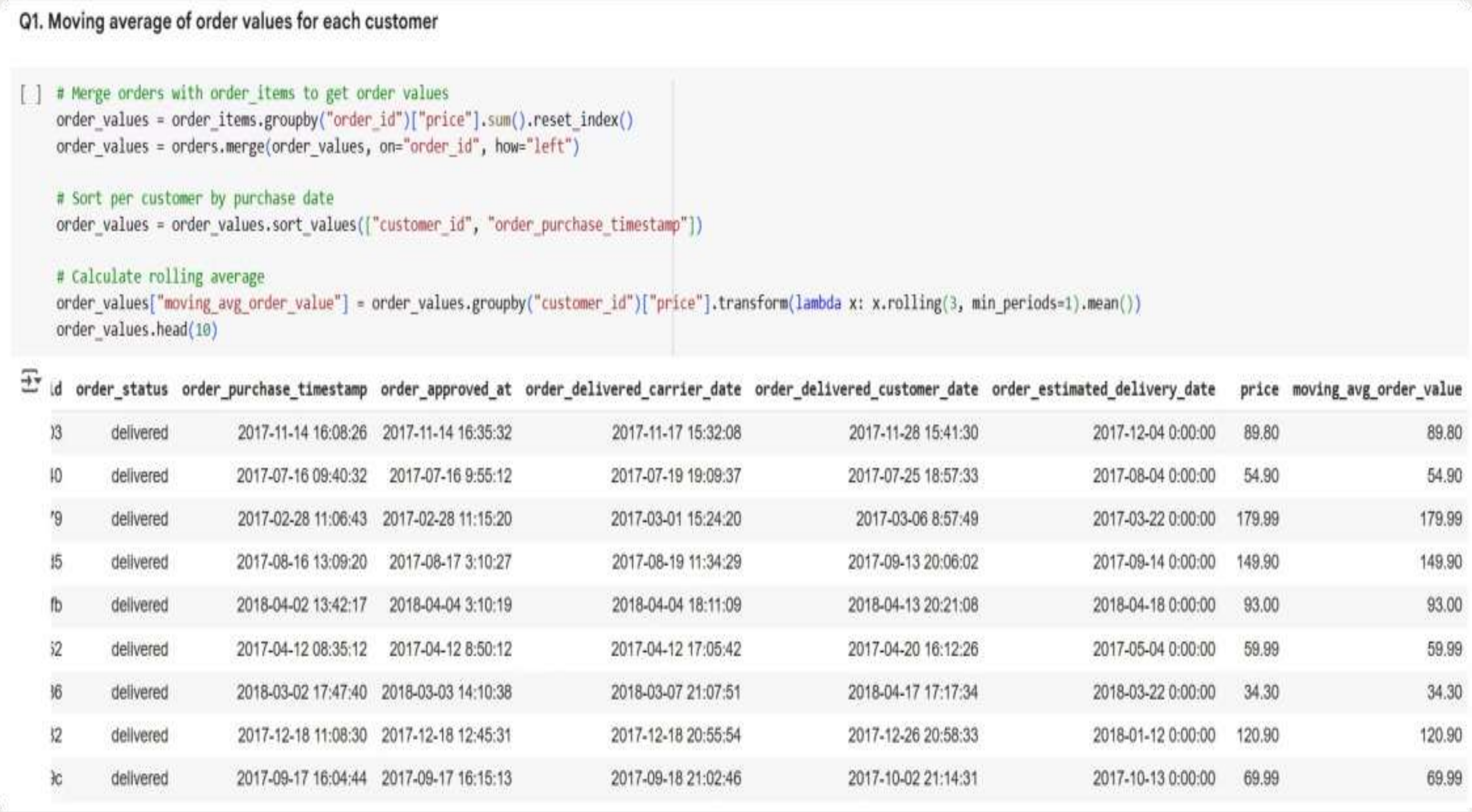
```
SELECT o.customer_id,
       o.order_id,
       SUM(oi.price) AS order_value,
       ROUND(AVG(SUM(oi.price)) OVER (
           PARTITION BY o.customer_id
           ORDER BY o.order_purchase_timestamp
           ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
       ), 2) AS moving_avg_order_value
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY o.customer_id, o.order_id, o.order_purchase_timestamp;
```



customer_id	order_id	order_value	moving_avg_order_value
00012a2ce6f8dcda20d059ce98491703	5f79b5b0931d63f1a42989eb65b9da6e	89.80	89.80
000161a058600d5901f007fab4c27140	a44895d095d7e0702b6a162fa2dbeced	54.90	54.90
0001fd6190edaaf884bcaf3d49edf079	316a104623542e4d75189bb372bc5f8d	179.99	179.99
0002414f95344307404f0ace7a26f1d5	5825ce2e88d5346438686b0bba99e5ee	149.90	149.90
000379cdec625522490c315e70c7a9fb	0ab7fb08086d4af9141453c91878ed7a	93.00	93.00
0004164d20a9e969af783496f3408652	cd3558a10d854487b4f907e9b326a4fc	59.99	59.99
000419c5494106c306a97b5635748086	07f6c3baf9ac86865b60f640c4f923c6	34.30	34.30
00046a560d407e99b969756e0b10f282	8c3d752c5c02227878fae49aeaddbfd7	120.90	120.90

#	Time	Action	Message
1	20:09:03	SELECT o.customer_id, o.order_id, SUM(oi.price) AS order_value, ROUND(AVG(SUM(oi.price)) OV...	98666 row(s) returned

PYTHON


Q1. Moving average of order values for each customer

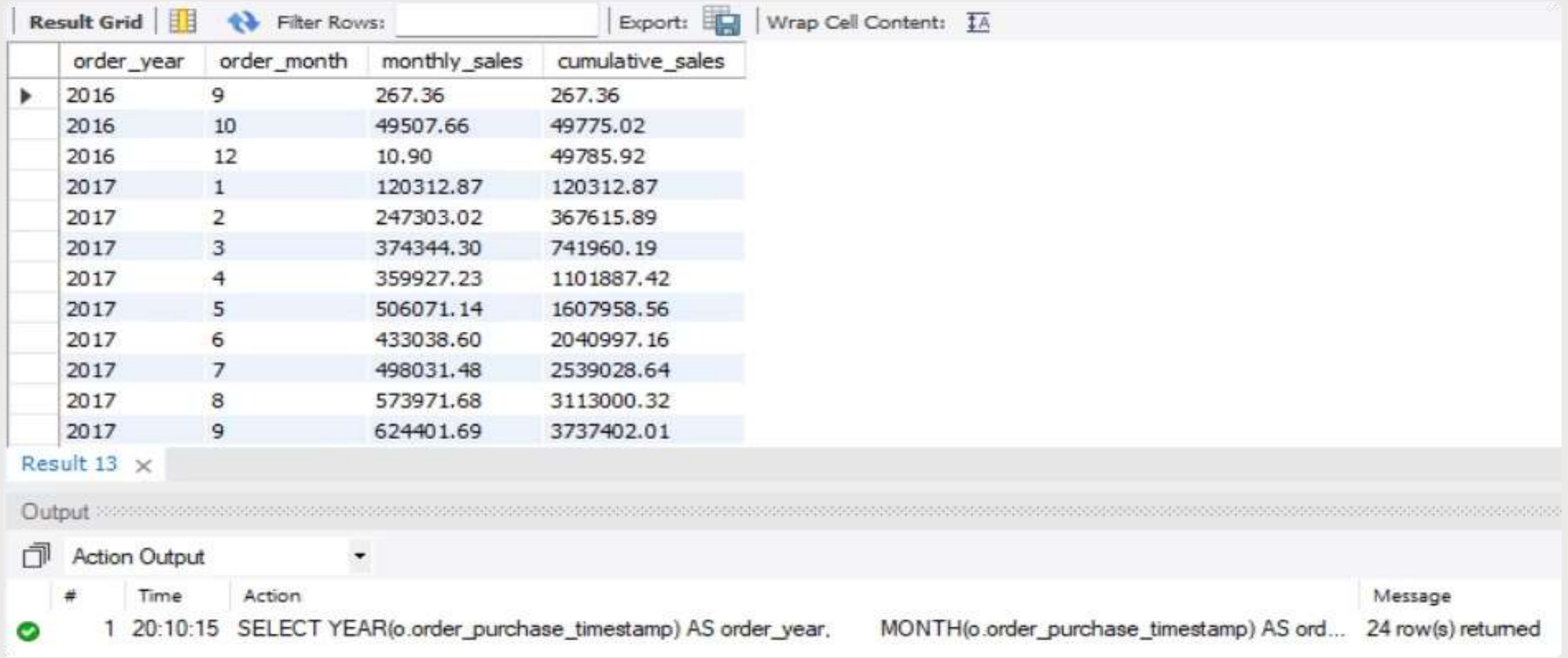
```
[ ] # Merge orders with order_items to get order values
    order_values = order_items.groupby("order_id")["price"].sum().reset_index()
    order_values = orders.merge(order_values, on="order_id", how="left")

    # Sort per customer by purchase date
    order_values = order_values.sort_values(["customer_id", "order_purchase_timestamp"])

    # Calculate rolling average
    order_values["moving_avg_order_value"] = order_values.groupby("customer_id")["price"].transform(lambda x: x.rolling(3, min_periods=1).mean())
    order_values.head(10)
```

id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date	price	moving_avg_order_value
)3	delivered	2017-11-14 16:08:26	2017-11-14 16:35:32	2017-11-17 15:32:08	2017-11-28 15:41:30	2017-12-04 0:00:00	89.80	89.80
40	delivered	2017-07-16 09:40:32	2017-07-16 9:55:12	2017-07-19 19:09:37	2017-07-25 18:57:33	2017-08-04 0:00:00	54.90	54.90
'9	delivered	2017-02-28 11:06:43	2017-02-28 11:15:20	2017-03-01 15:24:20	2017-03-06 8:57:49	2017-03-22 0:00:00	179.99	179.99
15	delivered	2017-08-16 13:09:20	2017-08-17 3:10:27	2017-08-19 11:34:29	2017-09-13 20:06:02	2017-09-14 0:00:00	149.90	149.90
fb	delivered	2018-04-02 13:42:17	2018-04-04 3:10:19	2018-04-04 18:11:09	2018-04-13 20:21:08	2018-04-18 0:00:00	93.00	93.00
52	delivered	2017-04-12 08:35:12	2017-04-12 8:50:12	2017-04-12 17:05:42	2017-04-20 16:12:26	2017-05-04 0:00:00	59.99	59.99
36	delivered	2018-03-02 17:47:40	2018-03-03 14:10:38	2018-03-07 21:07:51	2018-04-17 17:17:34	2018-03-22 0:00:00	34.30	34.30
32	delivered	2017-12-18 11:08:30	2017-12-18 12:45:31	2017-12-18 20:55:54	2017-12-26 20:58:33	2018-01-12 0:00:00	120.90	120.90
3c	delivered	2017-09-17 16:04:44	2017-09-17 16:15:13	2017-09-18 21:02:46	2017-10-02 21:14:31	2017-10-13 0:00:00	69.99	69.99

2. Cumulative sales per month per year

SQL

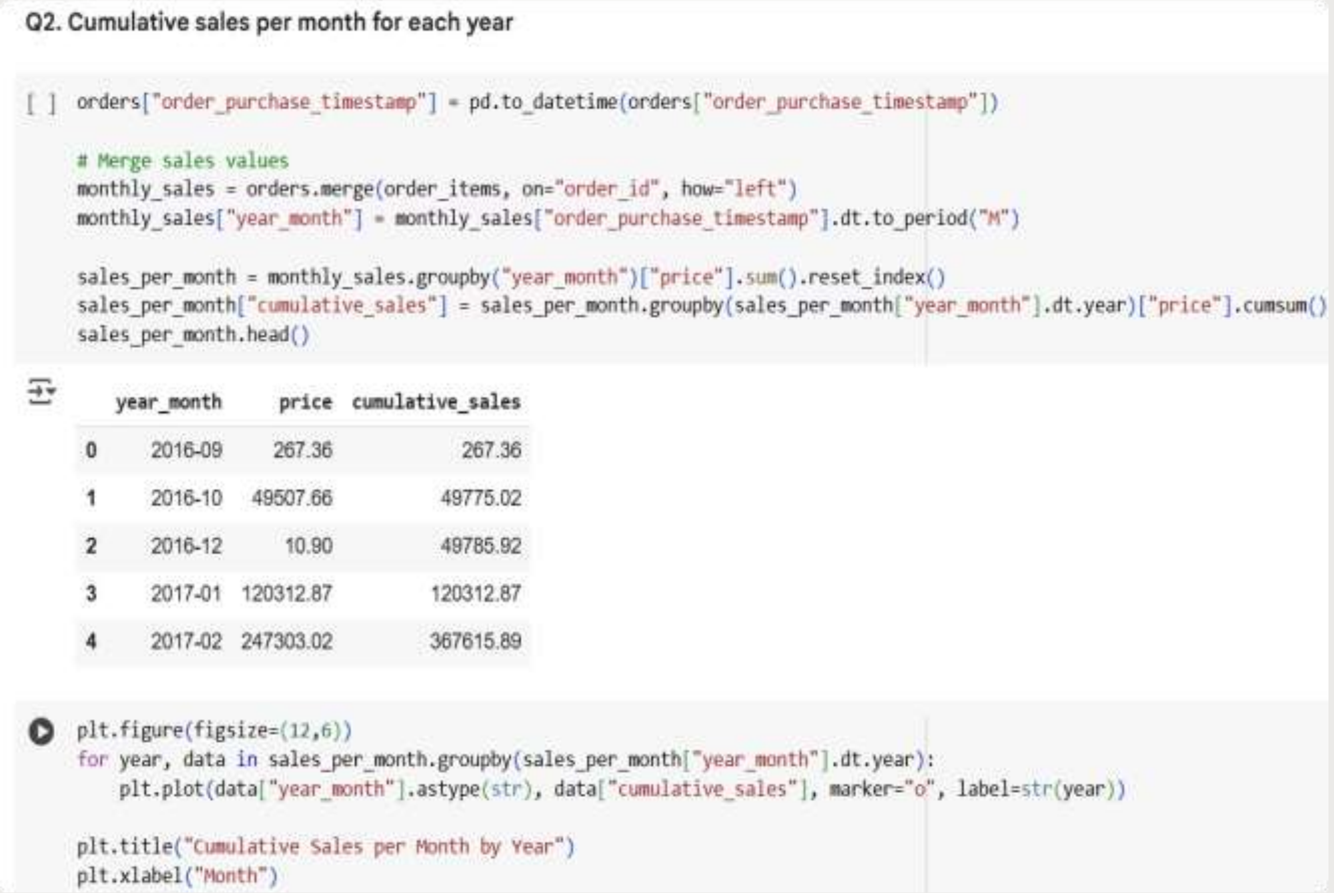
```
SELECT YEAR(o.order_purchase_timestamp) AS order_year,
    MONTH(o.order_purchase_timestamp) AS order_month,
    SUM(oi.price) AS monthly_sales,
    SUM(SUM(oi.price)) OVER (
      PARTITION BY YEAR(o.order_purchase_timestamp)
      ORDER BY MONTH(o.order_purchase_timestamp)
    ) AS cumulative_sales
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY YEAR(o.order_purchase_timestamp), MONTH(o.order_purchase_timestamp)
ORDER BY order_year, order_month;
```



order_year	order_month	monthly_sales	cumulative_sales
2016	9	267.36	267.36
2016	10	49507.66	49775.02
2016	12	10.90	49785.92
2017	1	120312.87	120312.87
2017	2	247303.02	367615.89
2017	3	374344.30	741960.19
2017	4	359927.23	1101887.42
2017	5	506071.14	1607958.56
2017	6	433038.60	2040997.16
2017	7	498031.48	2539028.64
2017	8	573971.68	3113000.32
2017	9	624401.69	3737402.01

#	Time	Action	Message
1	20:10:15	SELECT YEAR(o.order_purchase_timestamp) AS order_year, MONTH(o.order_purchase_timestamp) AS ord...	24 row(s) returned

PYTHON

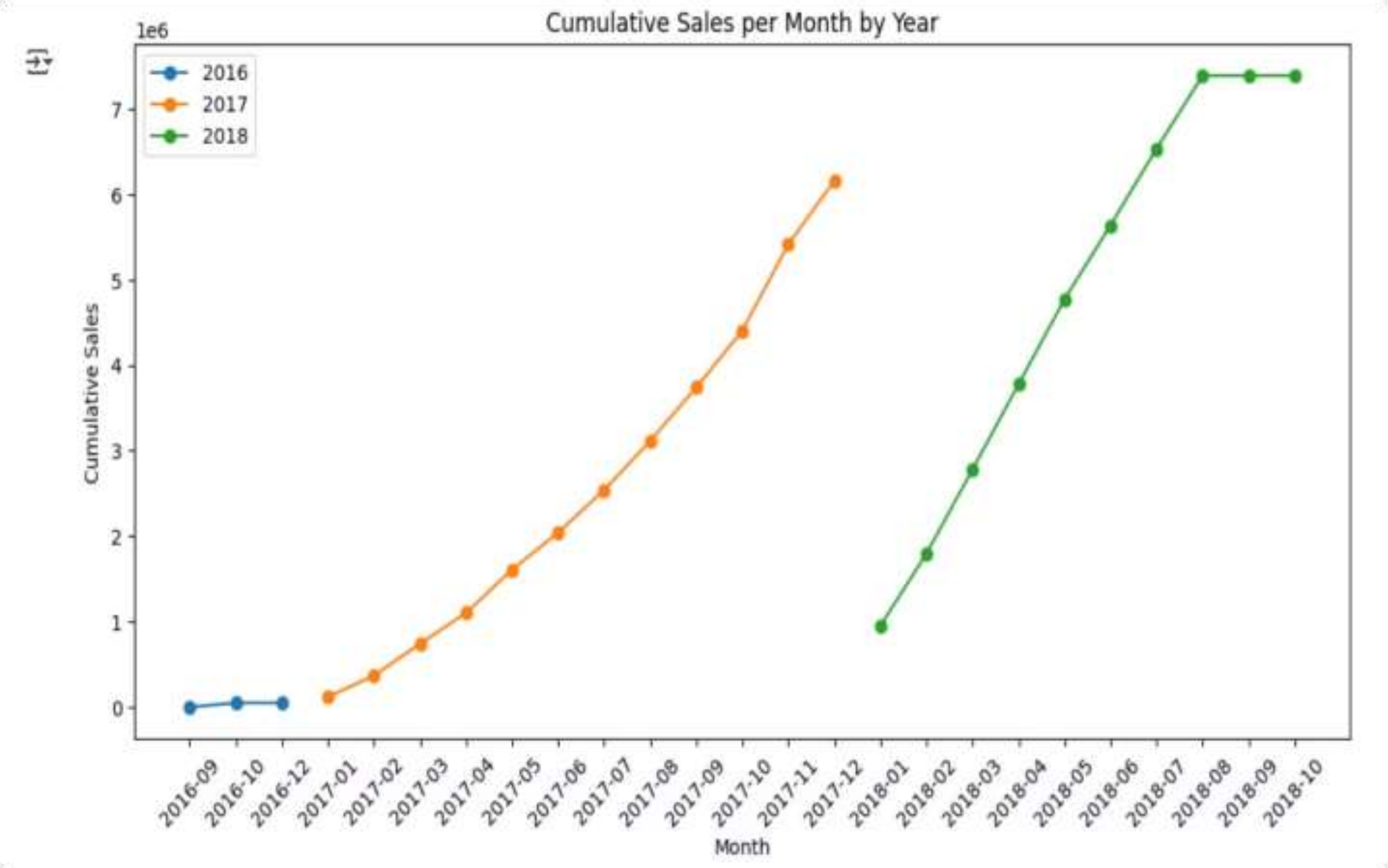


Q2. Cumulative sales per month for each year

```
orders["order_purchase_timestamp"] = pd.to_datetime(orders["order_purchase_timestamp"])

# Merge sales values
monthly_sales = orders.merge(order_items, on="order_id", how="left")
monthly_sales["year_month"] = monthly_sales["order_purchase_timestamp"].dt.to_period("M")

sales_per_month = monthly_sales.groupby("year_month")["price"].sum().reset_index()
sales_per_month["cumulative_sales"] = sales_per_month.groupby(sales_per_month["year_month"].dt.year)["price"].cumsum()
sales_per_month.head()
```

```
   year_month      price  cumulative_sales
0     2016-09     267.36            267.36
1     2016-10   49507.66          49775.02
2     2016-12      10.90          49785.92
3     2017-01  120312.87         120312.87
4     2017-02  247303.02         367615.89
```

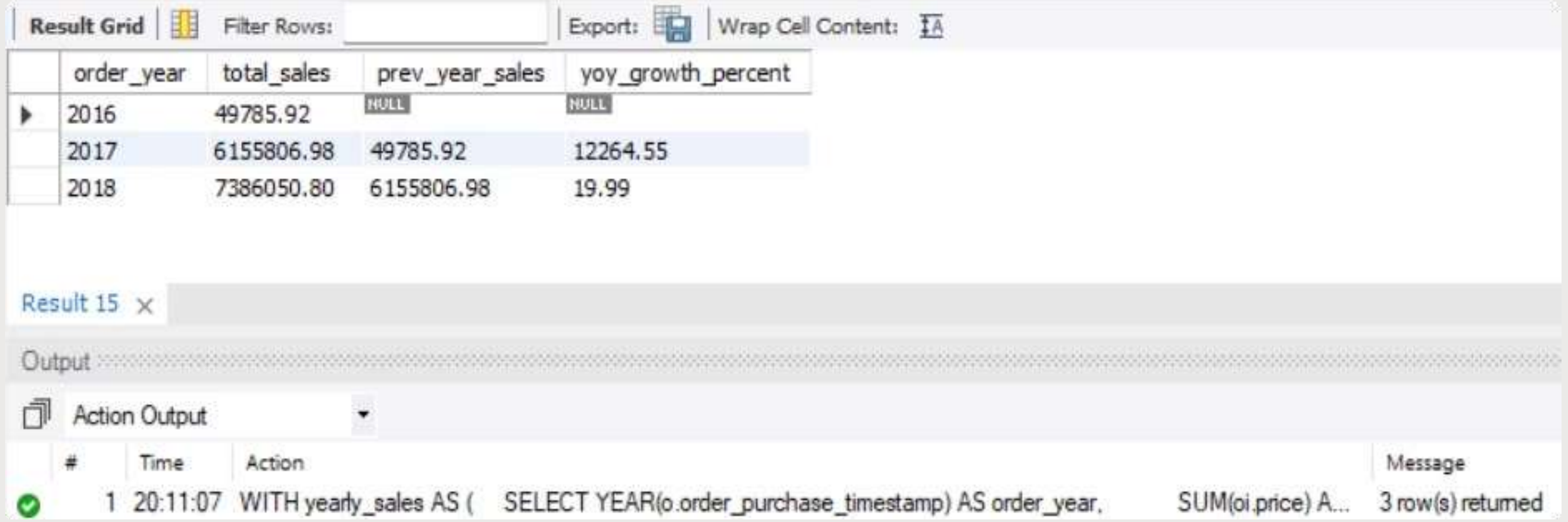
```
plt.figure(figsize=(12,6))
for year, data in sales_per_month.groupby(sales_per_month["year_month"].dt.year):
    plt.plot(data["year_month"].astype(str), data["cumulative_sales"], marker="o", label=str(year))

plt.title("Cumulative Sales per Month by Year")
plt.xlabel("Month")
```



3. Year-over-year growth rate of sales

SQL

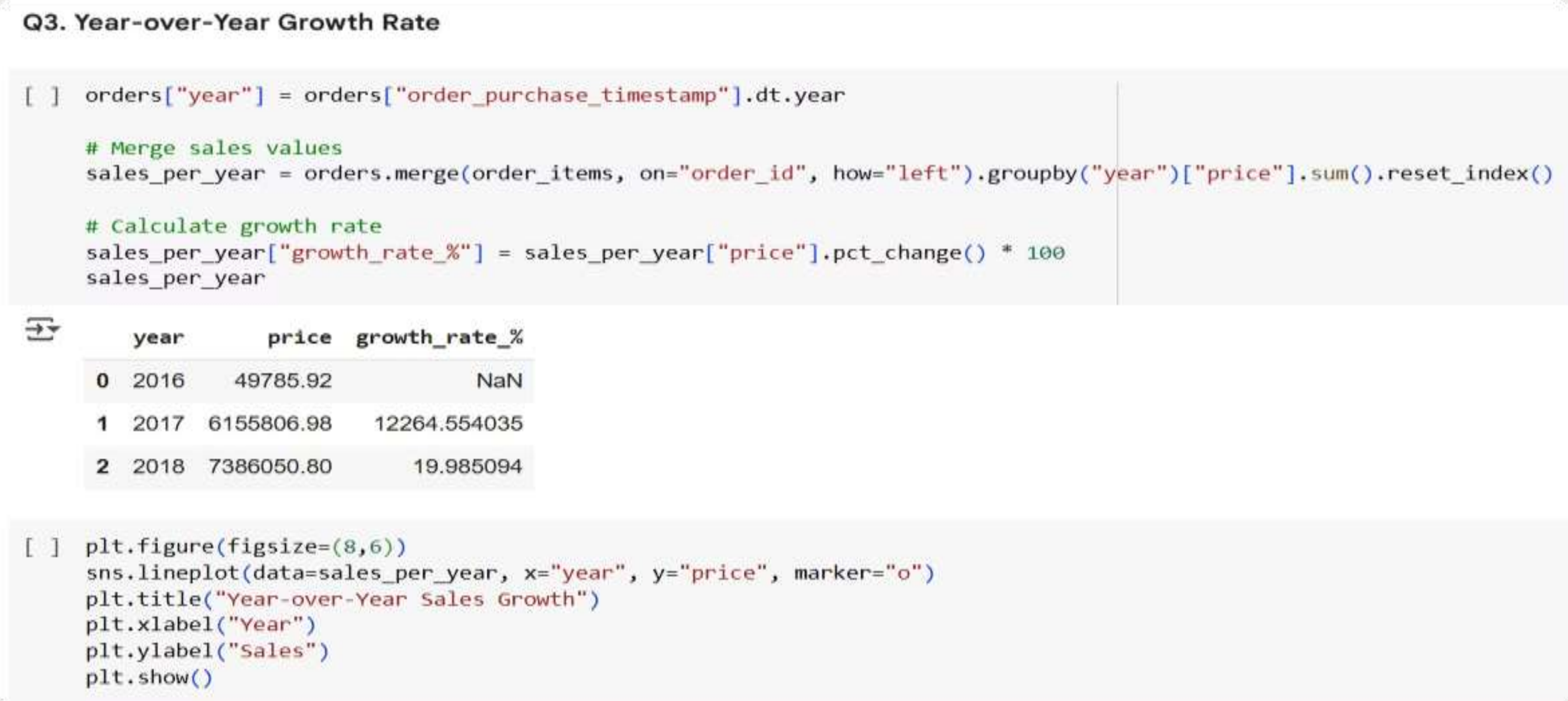
```
WITH yearly_sales AS (
   SELECT YEAR(o.order_purchase_timestamp) AS order_year,
        SUM(oi.price) AS total_sales
   FROM orders o
   JOIN order_items oi ON o.order_id = oi.order_id
   GROUP BY YEAR(o.order_purchase_timestamp)
)
SELECT order_year,
     total_sales,
     LAG(total_sales) OVER (ORDER BY order_year) AS prev_year_sales,
     ROUND((total_sales - LAG(total_sales) OVER (ORDER BY order_year))
         / LAG(total_sales) OVER (ORDER BY order_year) * 100, 2)
         AS yoy_growth_percent
FROM yearly_sales;
```



```
  order_year  total_sales  prev_year_sales  yoy_growth_percent
▶ 2016        49785.92     NULL             NULL
  2017        6155806.98   49785.92         12264.55
  2018        7386050.80   6155806.98       19.99
```

```
   #  Time      Action                                                                                          Message
✓  1  20:11:07  WITH yearly_sales AS (    SELECT YEAR(o.order_purchase_timestamp) AS order_year,    SUM(oi.price) A...  3 row(s) returned
```

PYTHON

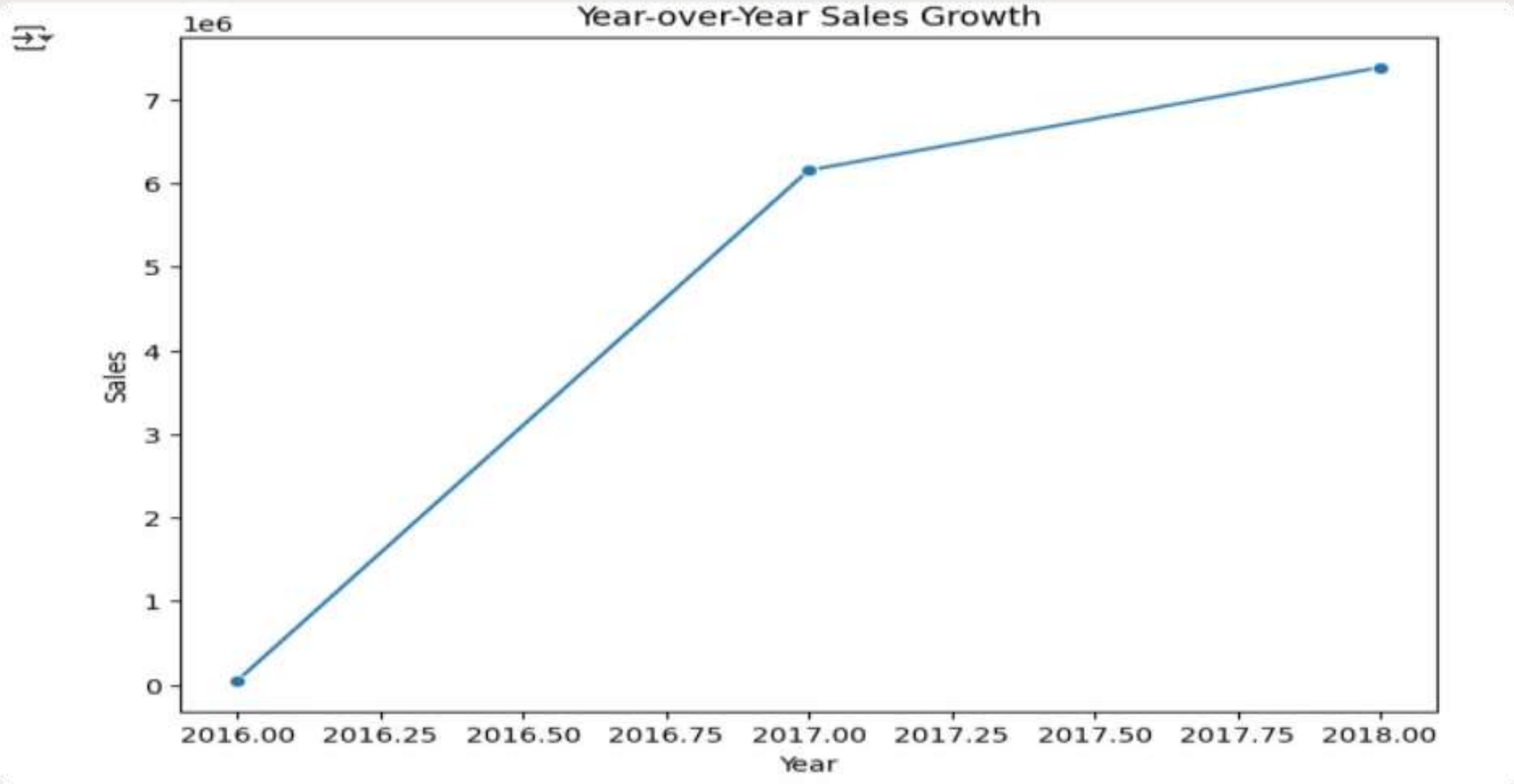


Q3. Year-over-Year Growth Rate

```
[ ] orders["year"] = orders["order_purchase_timestamp"].dt.year

    # Merge sales values
    sales_per_year = orders.merge(order_items, on="order_id", how="left").groupby("year")["price"].sum().reset_index()

    # Calculate growth rate
    sales_per_year["growth_rate_%"] = sales_per_year["price"].pct_change() * 100
    sales_per_year
```

```
   year       price  growth_rate_%
0  2016    49785.92            NaN
1  2017  6155806.98   12264.554035
2  2018  7386050.80      19.985094
```

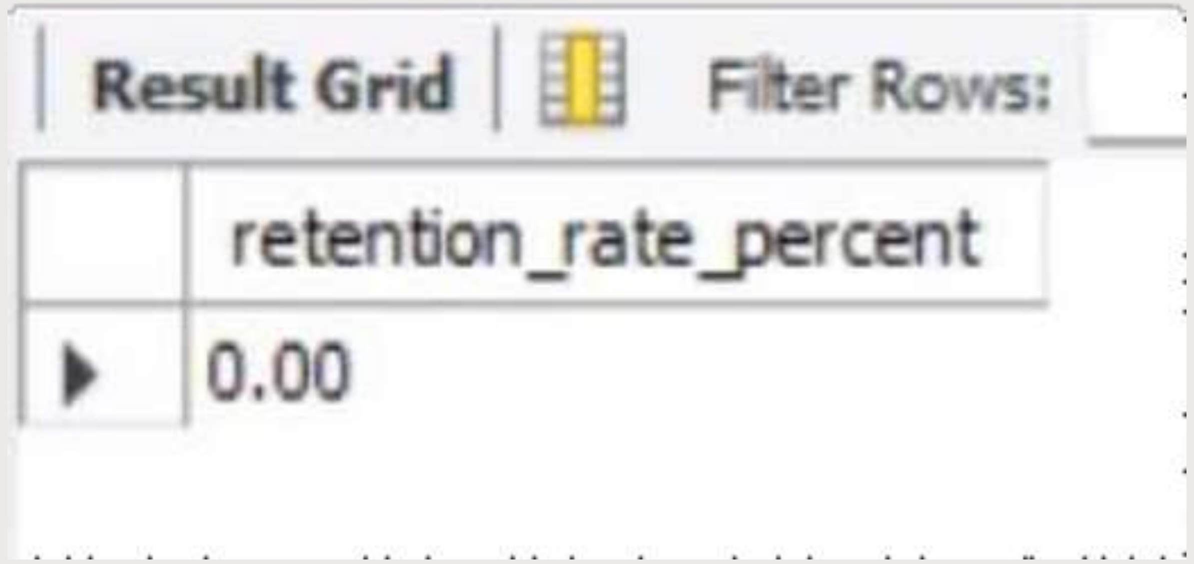
```
[ ] plt.figure(figsize=(8,6))
    sns.lineplot(data=sales_per_year, x="year", y="price", marker="o")
    plt.title("Year-over-Year Sales Growth")
    plt.xlabel("Year")
    plt.ylabel("Sales")
    plt.show()
```



4. Retention rate (customers returning within 6 months)

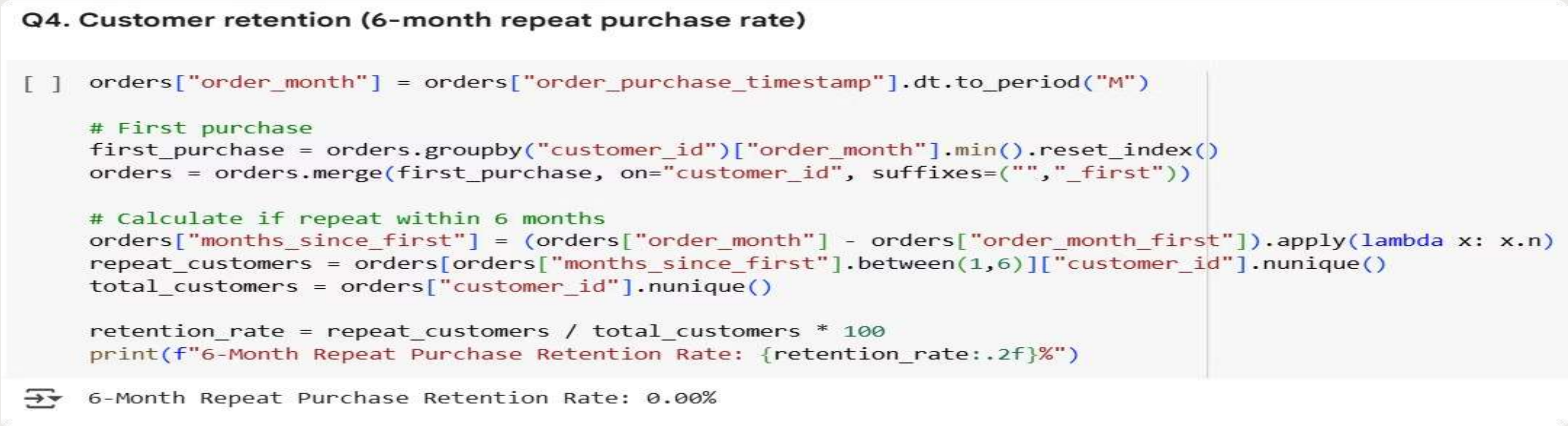
SQL

```
WITH first_orders AS (
   SELECT customer_id,
         MIN(order_purchase_timestamp) AS first_order_date
   FROM orders
   GROUP BY customer_id
),
next_orders AS (
   SELECT o.customer_id,
         o.order_purchase_timestamp
   FROM orders o
   JOIN first_orders f ON o.customer_id = f.customer_id
   WHERE o.order_purchase_timestamp > f.first_order_date
     AND TIMESTAMPDIFF(MONTH, f.first_order_date, o.order_purchase_timestamp) <= 6
)
SELECT ROUND(COUNT(DISTINCT next_orders.customer_id)
           / COUNT(DISTINCT first_orders.customer_id) * 100, 2) AS retention_rate_percent
FROM first_orders
LEFT JOIN next_orders ON first_orders.customer_id = next_orders.customer_id;
```



	retention_rate_percent
▶	0.00

PYTHON



Q4. Customer retention (6-month repeat purchase rate)

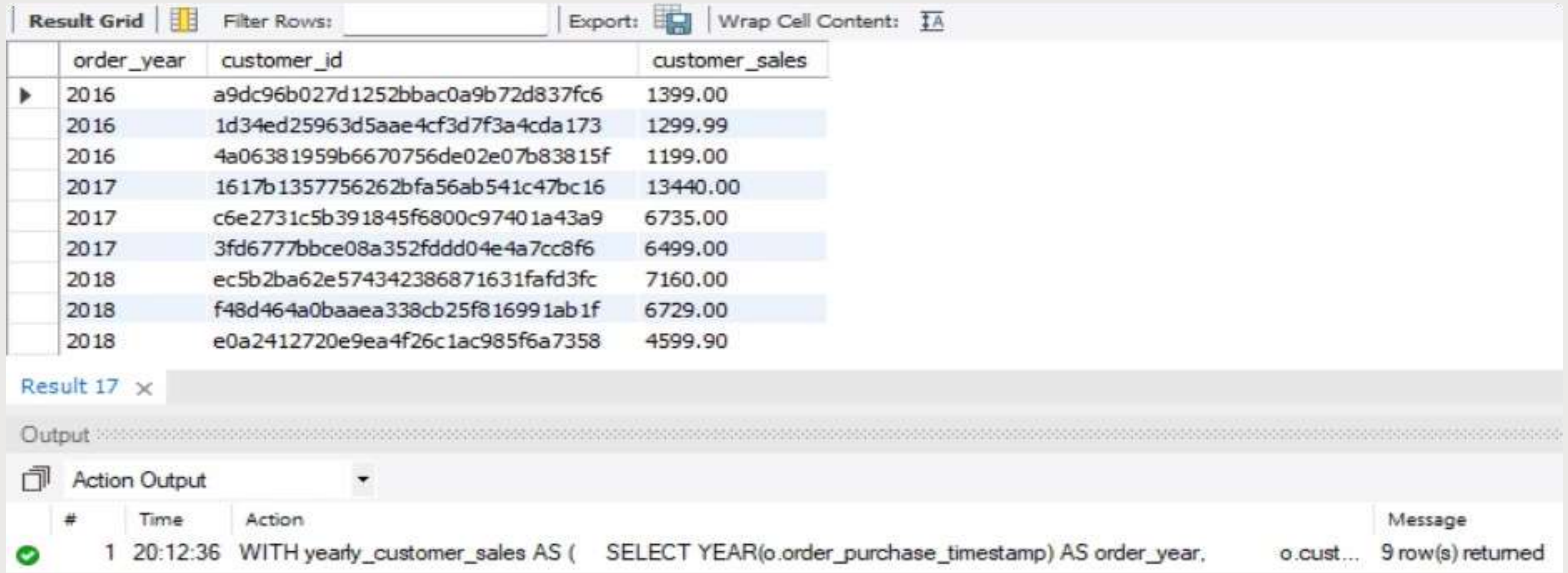
```
[ ]  orders["order_month"] = orders["order_purchase_timestamp"].dt.to_period("M")

     # First purchase
     first_purchase = orders.groupby("customer_id")["order_month"].min().reset_index()
     orders = orders.merge(first_purchase, on="customer_id", suffixes=("","_first"))

     # Calculate if repeat within 6 months
     orders["months_since_first"] = (orders["order_month"] - orders["order_month_first"]).apply(lambda x: x.n)
     repeat_customers = orders[orders["months_since_first"].between(1,6)]["customer_id"].nunique()
     total_customers = orders["customer_id"].nunique()

     retention_rate = repeat_customers / total_customers * 100
     print(f"6-Month Repeat Purchase Retention Rate: {retention_rate:.2f}%")
```

```
6-Month Repeat Purchase Retention Rate: 0.00%
```

5. Top 3 customers by spend per year

SQL

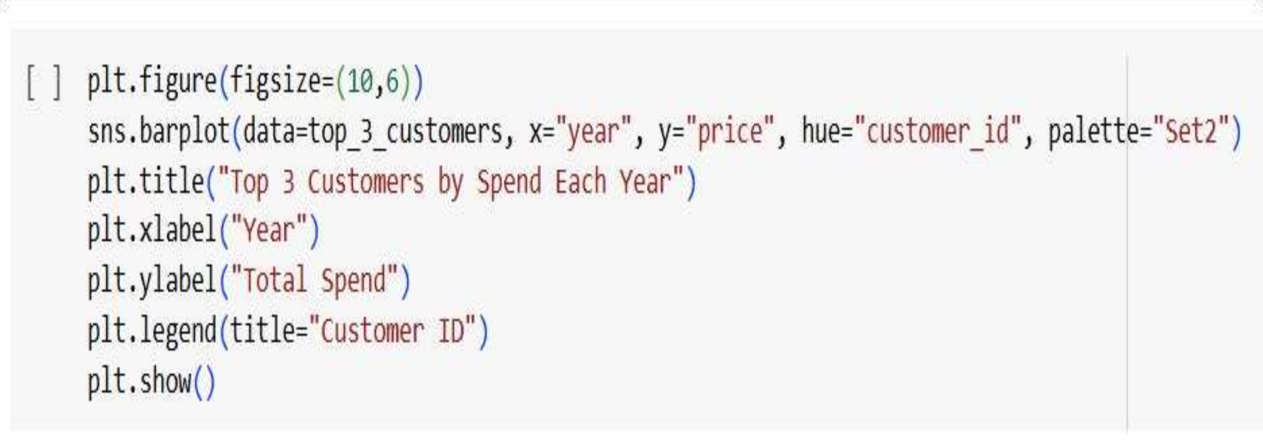
```
WITH yearly_customer_sales AS (
   SELECT YEAR(o.order_purchase_timestamp) AS order_year,
         o.customer_id,
         SUM(oi.price) AS customer_sales
   FROM orders o
   JOIN order_items oi ON o.order_id = oi.order_id
   GROUP BY YEAR(o.order_purchase_timestamp), o.customer_id
)
SELECT order_year, customer_id, customer_sales
FROM (
   SELECT order_year, customer_id, customer_sales,
         ROW_NUMBER() OVER (PARTITION BY order_year ORDER BY customer_sales DESC) AS rank_
   FROM yearly_customer_sales
) ranked
WHERE rank_ <= 3
ORDER BY order_year, rank_;
```



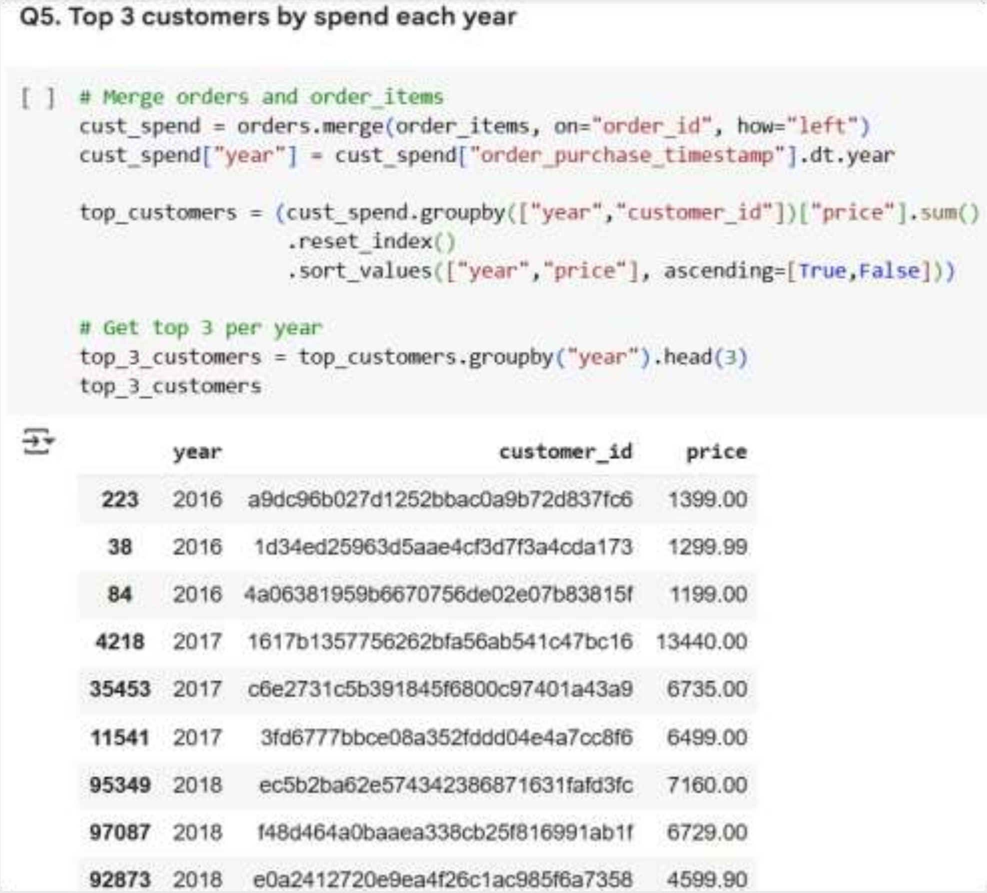
order_year	customer_id	customer_sales
2016	a9dc96b027d1252bbac0a9b72d837fc6	1399.00
2016	1d34ed25963d5aae4cf3d7f3a4cda173	1299.99
2016	4a06381959b6670756de02e07b83815f	1199.00
2017	1617b1357756262bfa56ab541c47bc16	13440.00
2017	c6e2731c5b391845f6800c97401a43a9	6735.00
2017	3fd6777bbce08a352fddd04e4a7cc8f6	6499.00
2018	ec5b2ba62e574342386871631fafd3fc	7160.00
2018	f48d464a0baaea338cb25f816991ab1f	6729.00
2018	e0a2412720e9ea4f26c1ac985f6a7358	4599.90

#	Time	Action	Message
1	20:12:36	WITH yearly_customer_sales AS (SELECT YEAR(o.order_purchase_timestamp) AS order_year, o.cust...	9 row(s) returned

PYTHON



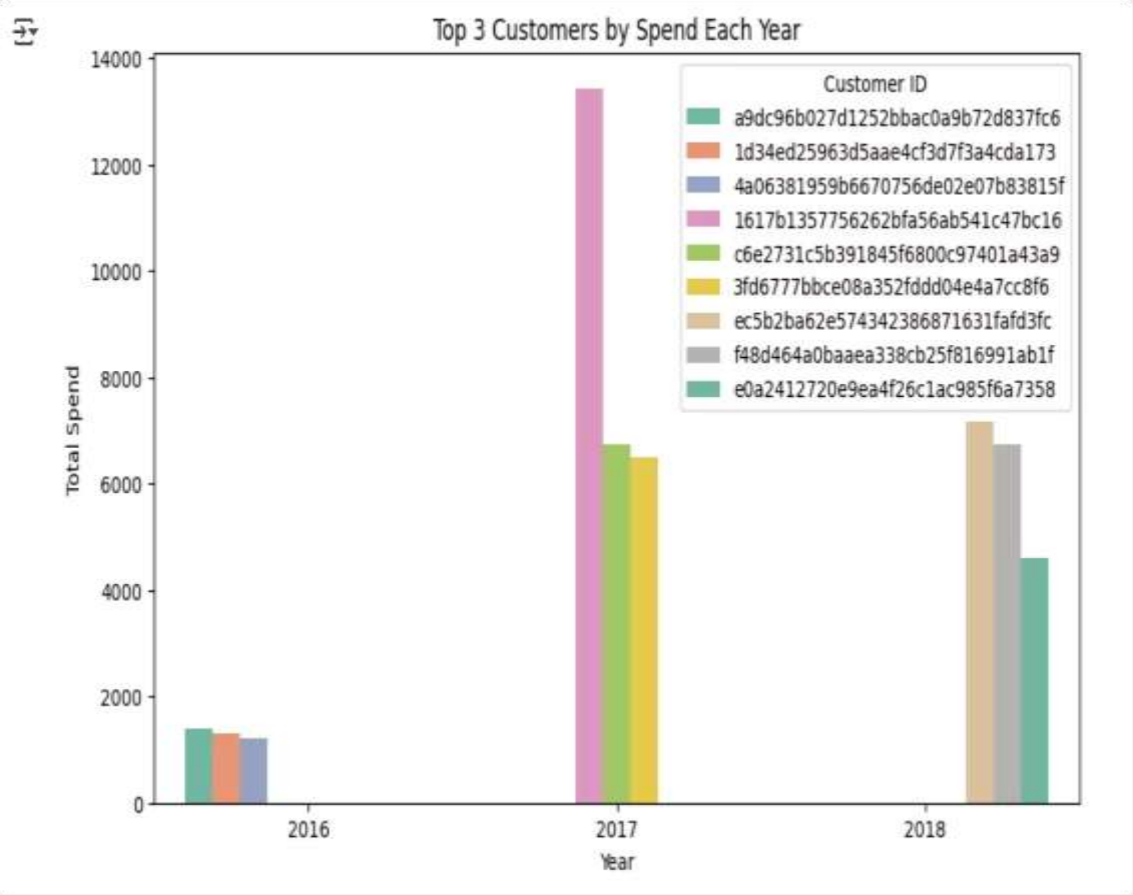
```
[ ] plt.figure(figsize=(10,6))
    sns.barplot(data=top_3_customers, x="year", y="price", hue="customer_id", palette="Set2")
    plt.title("Top 3 Customers by Spend Each Year")
    plt.xlabel("Year")
    plt.ylabel("Total Spend")
    plt.legend(title="Customer ID")
    plt.show()
```



Q5. Top 3 customers by spend each year

```
[ ] # Merge orders and order_items
    cust_spend = orders.merge(order_items, on="order_id", how="left")
    cust_spend["year"] = cust_spend["order_purchase_timestamp"].dt.year

    top_customers = (cust_spend.groupby(["year","customer_id"])["price"].sum()
                     .reset_index()
                     .sort_values(["year","price"], ascending=[True,False]))

    # Get top 3 per year
    top_3_customers = top_customers.groupby("year").head(3)
    top_3_customers
```

	year	customer_id	price
223	2016	a9dc96b027d1252bbac0a9b72d837fc6	1399.00
38	2016	1d34ed25963d5aae4cf3d7f3a4cda173	1299.99
84	2016	4a06381959b6670756de02e07b83815f	1199.00
4218	2017	1617b1357756262bfa56ab541c47bc16	13440.00
35453	2017	c6e2731c5b391845f6800c97401a43a9	6735.00
11541	2017	3fd6777bbce08a352fddd04e4a7cc8f6	6499.00
95349	2018	ec5b2ba62e574342386871631fafd3fc	7160.00
97087	2018	f48d464a0baaea338cb25f816991ab1f	6729.00
92873	2018	e0a2412720e9ea4f26c1ac985f6a7358	4599.90



THANK YOU

LinkedIn

GitHub