"NAVIGATING THE FUTURE ONLINE SHOPPING"

Presented by Mr. Hitesh Shetye

PROBLEM STATEMENT

•Online shopping creates a lot of data about customers, orders, and products.
•**Goal:** Use SQL and Python to study this data and find useful insights.

Focus Areas

1. Customer Behavior
•Who buys the most
•Who buys again and again (repeat customers)

2. Sales Performance
•Monthly and yearly sales
•Total sales growth

3. Operations
•Delivery speed and delays
•Popular payment methods

OBJECTIVES

1. Understand customer, order, product, and seller behavior.
2. Use SQL to write and run different types of queries.
3. Use Python (Pandas + charts) for deeper data analysis.
4. Perform basic, intermediate, and advanced analysis.
5. Create clear and meaningful visualizations.
6. Find important KPIs like growth rate, retention, and top customers.
7. Build a smooth end-to-end data workflow from raw data to insights.

DATASETS USED

7 key datasets analyzed (100k+ rows total):

customers.csv – customer IDs, locations (state, city).
orders.csv – order details (timestamps, status, delivery dates). **order_items.csv** – product-level order info (price, freight).
products.csv – product data (category, dimensions, description length). **sellers.csv** – seller details and location.
payments.csv – payment methods, installments, transaction values. **geolocation.csv** – mapping of ZIP codes to city/state.

TOOLS & TECHNOLOGIES

1. **SQL (MySQL Workbench)** – Querying relational data, joins, aggregations.
2. **Python (Google Colab)** – Data manipulation (Pandas), advanced analysis.

   Matplotlib & Seaborn – Trend analysis, visual storytelling.
3. **CSV files** – Primary source of structured e-commerce data.
4. **Integrated workflow:** SQL for raw query execution, Python for insights + charts.

WORKFLOW

Data Import & Cleaning

- Managed missing values & encoding issues (latin1).
- Standardized schema across 7 datasets.

Basic Queries

- Order counts, popular categories, order statuses.

Intermediate Queries

- Monthly sales, payment preferences, delivery delays.

Advanced Queries

- Moving averages, cumulative sales, YoY growth, retention, top customers.

Visualization

- Trend lines, bar charts, cumulative plots.

BASIC PROBLEMS

1. Total customers, sellers, orders extracted from database.
2. Most popular product categories identified.
3. Order statuses (delivered, shipped, canceled) distribution analyzed.
4. Insights: majority of orders delivered successfully; cancellations <5%.

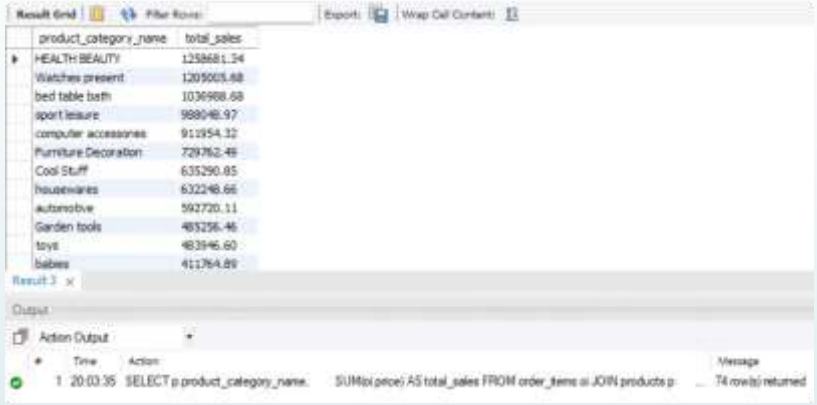
SQL QUERIES:

```sql
-- ----------------------------
-- BASIC QUERIES
-- ----------------------------

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

-- 2. Count the number of orders placed in 2017
SELECT COUNT(*) AS total_orders_2017
FROM orders
WHERE YEAR(order_purchase_timestamp) = 2017;

-- 3. Find the total sales per category
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;
```

Q1. Output

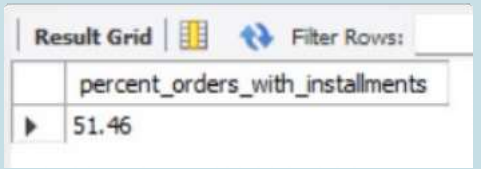
Q2. Output

Q3. Output




```sql
-- 4. Calculate the percentage of orders that were paid in installments
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;

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

Q4. Output



Q5. Output

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

Result 6

Output

Action Output

#	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

Q3. Total sales per category

```
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

Q4. Percentage of orders paid in installments

```
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 %
```

PYTHON CODE:

Q1. List all unique cities where customers are located

```
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)
```

Q2. Count the number of orders placed in 2017

```
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
```

Q5. Count customers from each state

```
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. Monthly sales trend analysis (2016–2018).
2. Delivery performance measured 3 several late deliveries detected.
3. Payment preferences 3 credit card (dominant), boleto (secondary).
4. Freight charges impacted final order value significantly in some categories.

SQL QUERIES:

```
-- ----------------------------
-- INTERMEDIATE QUERIES
-- ----------------------------

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

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

Q1. Output

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

Q2. Output

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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

Result 9

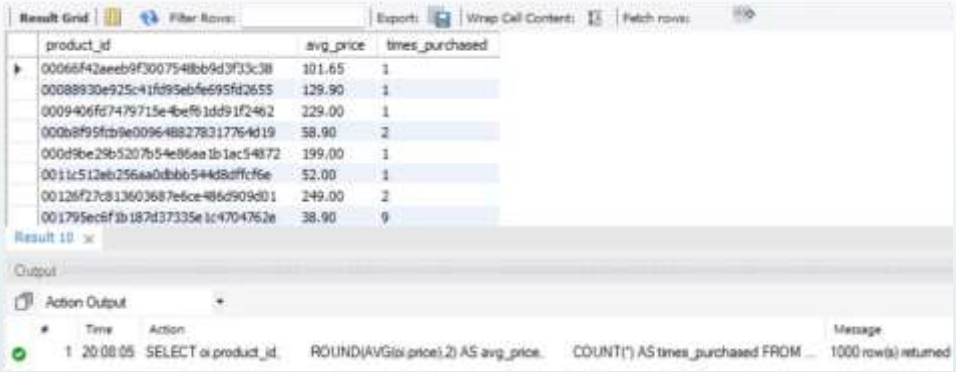
Output

Action Output

#	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

```sql
-- 3. Calculate the percentage of total revenue contributed by each product category
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;

-- 4. Identify the correlation between product price and the number of times a product has been purchased
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;
#Actual Correlation will be found using python

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

Q4. Output



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

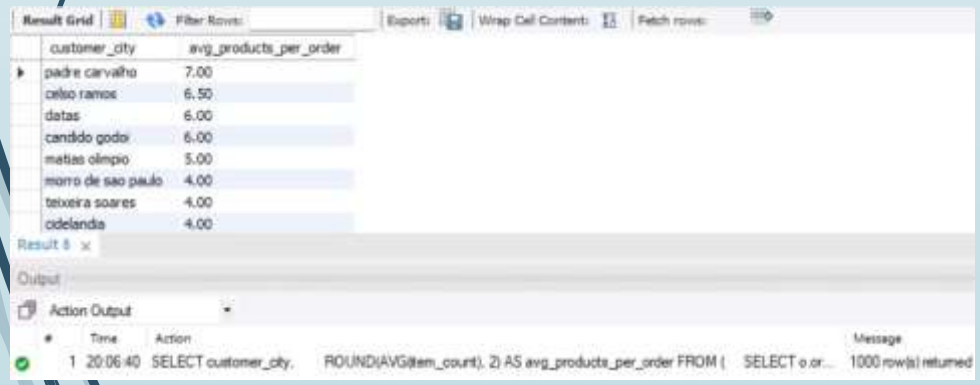
Result 10

Output

Action Output

#	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

Q3. Output



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

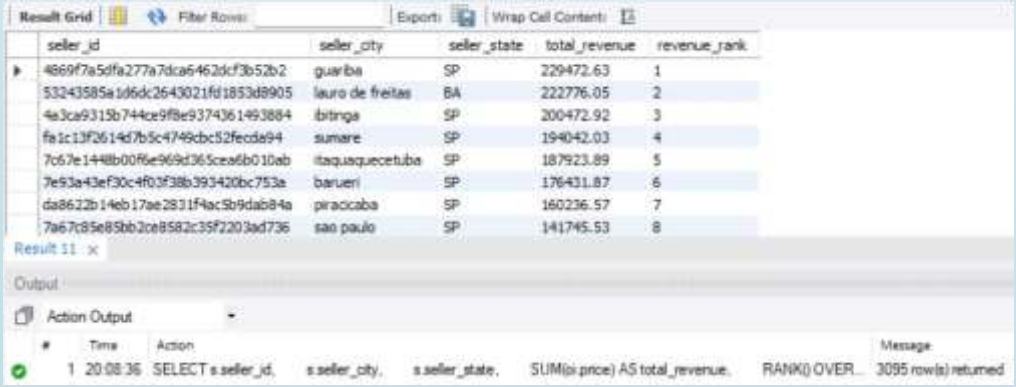
Result 8

Output

Action Output

#	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

Q5. Output



seller_id	seller_city	seller_state	total_revenue	revenue_rank
4869f7a5dfa277a7dca6462dcf3b52b2	guariba	SP	229472.63	1
53243585a1d6dc2643021fd1853d8905	lauro de freitas	BA	222776.05	2
4a3ca9315b744ce9f8e937436149388 4	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

Result 11

Output

Action Output

#	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 CODE:

Q1. Calculate the number of orders per month in 2018

```
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

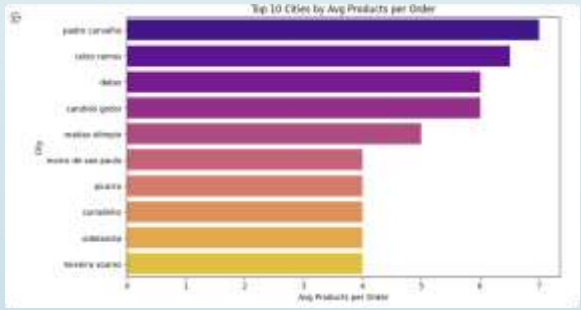
```
plt.figure(figsize=(12,6))
sns.lineplot(data=monthly_orders, x="order_purchase_timestamp", y="total_orders", marker="o")
plt.title("Monthly Orders in 2018")
plt.xlabel("Month")
plt.ylabel("Total Orders")
plt.xticks(rotation=45)
plt.show()
```

Q2. Find the average number of products per order, grouped by customer city

```
order_item_count = order_items.groupby("order_id").size().reset_index(name="item_count")
orders_with_customers = orders.merge(customers, on="customer_id", how="left").merge(order_item_count, on="order_id", how="left")

avg_products_city = orders_with_customers.groupby("customer_city")["item_count"].mean().reset_index()
avg_products_city = avg_products_city.sort_values("item_count", ascending=False)
avg_products_city.head()
```

	customer_city	item_count
2829	padre carvalho	7.0
988	celso ramos	6.5
1196	datas	6.5
758	candido godoi	6.5
2269	matias olimpio	5.5

```
plt.figure(figsize=(10,5))
sns.barplot(data=avg_products_city.head(10), x="item_count", y="customer_city", palette="viridis")
plt.title("Top 10 Cities by Avg Products per Order")
plt.xlabel("Avg Products per Order")
plt.ylabel("City")
plt.show()
```

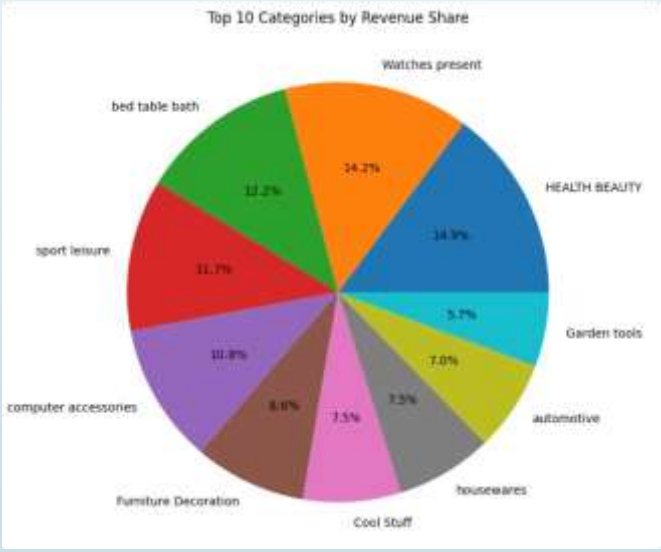


Q3. Calculate the percentage of total revenue contributed by each product category

```
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()
```

	product category	price	percent_revenue
36	HEALTH BEAUTY	1258681.34	9.994804
45	Watches present	1205005.68	9.564491
40	bed table bath	1036988.68	7.731738
66	sport leisure	988048.97	7.365843
53	computer accessories	911954.32	6.794468

```
plt.figure(figsize=(8,8))
plt.pie(revenue_per_cat.head(10)["percent_revenue"], labels=revenue_per_cat.head(10)["product category"], autopct='%1.1f%%')
plt.title("Top 10 Categories by Revenue Share")
plt.show()
```

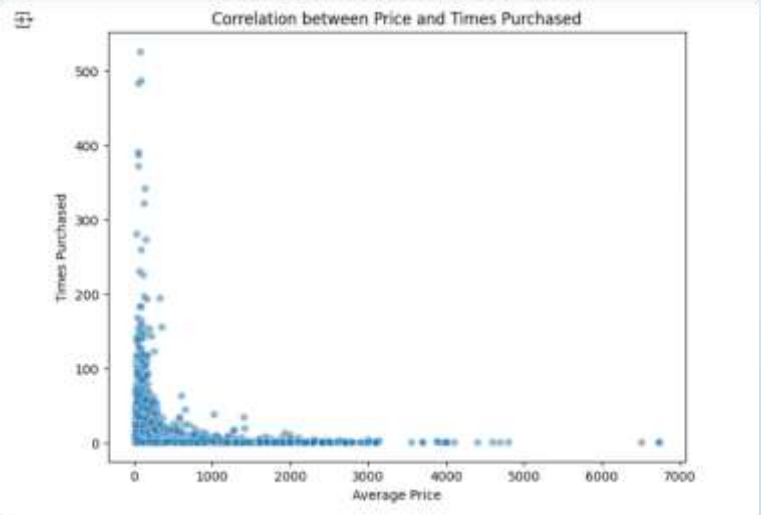


Q4. Correlation between product price and number of times purchased

```
product_stats = order_items.groupby("product_id").agg(
    avg_price=("price","mean"),
    times_purchased=("order_id","count")
).reset_index()

correlation = product_stats[["avg_price","times_purchased"]].corr()
print(correlation)
```

```
                avg_price  times_purchased
avg_price         1.00000         -0.03214
times_purchased  -0.03214          1.00000
```

```
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()
```

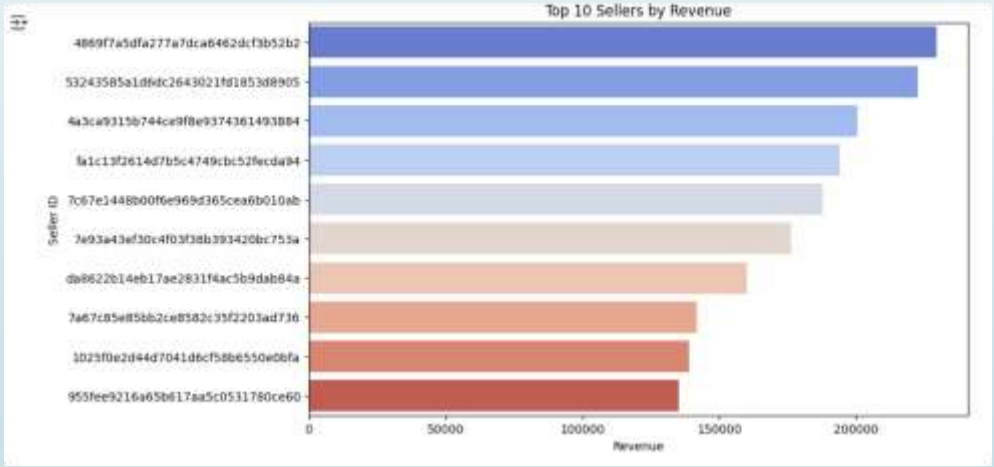


Q5. Total revenue generated by each seller, ranked

```
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

```
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()
```



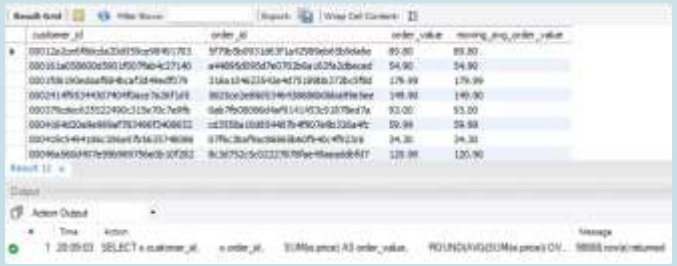
ADVANCED PROBLEMS

1. Rolling 3-order window used to track average spend patterns.
2. High-value customers show stable average spend over time.
3. Monthly sales aggregated and cumulative totals calculated.
4. Yearly sales acceleration clearly visible.
5. Shows opportunity for loyalty programs.
6. Revenue concentrated in top customers annually.
7. Top 3 customers contributed disproportionately higher spend.

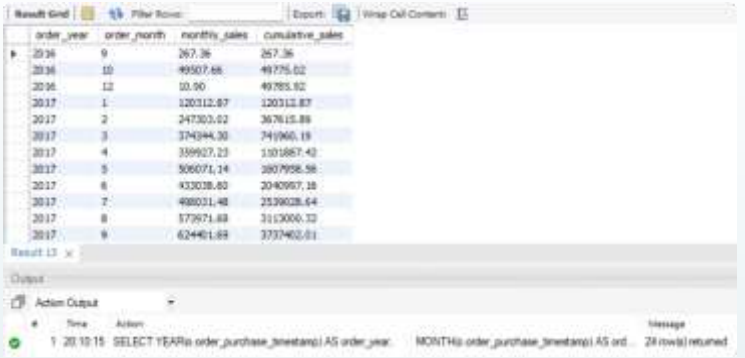
SQL QUERIES:

```sql
-- ============================
-- ADVANCED QUERIES
-- ============================

-- 1. Moving average of order values for each customer
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;

-- 2. Cumulative sales per month per year
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;
```

Q1. Output

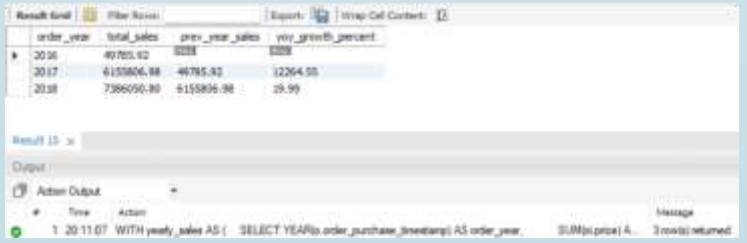


Q2. Output



```sql
-- 3. Year-over-year growth rate of sales
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;
```

Q3. Output



```sql
-- 4. Retention rate (customers returning within 6 months)
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;
```

Q4. Output

Result Grid | Filter Rows:

retention_rate_percent
0.00

```sql
-- 5. Top 3 customers by spend per year
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_;
```

Q5. Output

Result Grid | Filter Rows: | Export: | Wrap Cell Contents:

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

Result 17

Output

Action Output

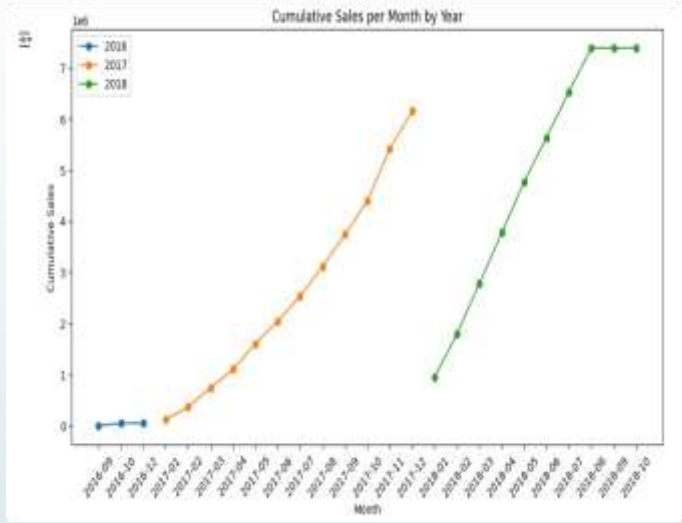
#	Time	Action	Message
1	20:12:36	WITH yearly_customer_sales AS (SELECT YEAR(o.order_purchase_timestamp) AS order_year, o.cust...	9 row(s) returned

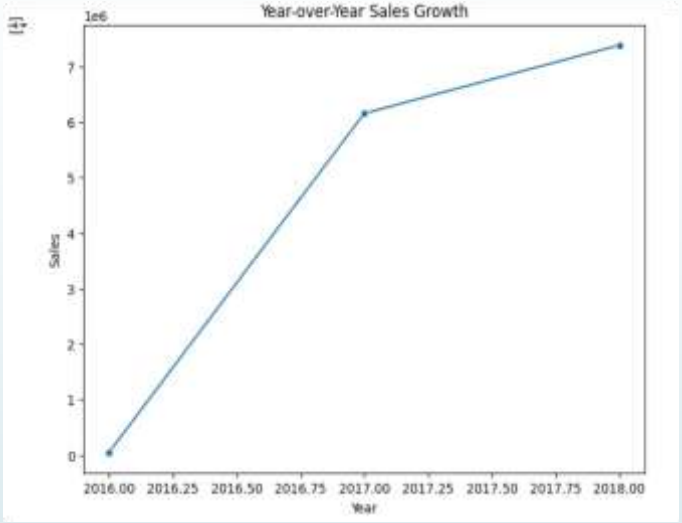
PYTHON CODE:

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)
```

	order_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
0		delivered	2017-11-14 16:08:26	2017-11-14 16:20:32	2017-11-17 15:32:08	2017-11-28 10:41:30	2017-12-04 0:00:00	89.80	89.80
1		delivered	2017-07-16 09:40:32	2017-07-16 9:55:12	2017-07-19 10:09:37	2017-07-25 19:57:33	2017-08-04 0:00:00	54.90	54.90
2		delivered	2017-02-28 11:06:43	2017-02-28 11:15:28	2017-03-01 15:24:20	2017-03-08 9:57:48	2017-03-22 0:00:00	179.99	179.99
3		delivered	2017-08-16 13:09:20	2017-08-17 3:10:27	2017-08-19 11:34:20	2017-09-13 20:09:02	2017-09-14 0:00:00	149.90	149.90
4		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
5		delivered	2017-04-12 08:35:12	2017-04-12 8:50:12	2017-04-12 17:09:42	2017-04-20 16:12:28	2017-05-04 0:00:00	59.99	59.99
6		delivered	2018-03-02 17:47:40	2018-03-02 14:10:38	2018-03-07 21:07:51	2018-04-17 17:17:54	2018-03-22 0:00:00	34.90	34.90
7		delivered	2017-12-18 11:08:30	2017-12-18 12:45:31	2017-12-19 20:09:54	2017-12-28 20:08:33	2018-01-12 0:00:00	120.90	120.90
8		delivered	2017-09-17 18:04:44	2017-09-17 18:15:13	2017-09-18 21:32:46	2017-10-02 21:14:31	2017-10-13 0:00:00	89.99	89.99





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)
```

	order_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
0		delivered	2017-11-14 16:08:26	2017-11-14 16:20:32	2017-11-17 15:32:08	2017-11-28 10:41:30	2017-12-04 0:00:00	89.80	89.80
1		delivered	2017-07-16 09:40:32	2017-07-16 9:55:12	2017-07-19 10:09:37	2017-07-25 19:57:33	2017-08-04 0:00:00	54.90	54.90
2		delivered	2017-02-28 11:06:43	2017-02-28 11:15:28	2017-03-01 15:24:20	2017-03-08 9:57:48	2017-03-22 0:00:00	179.99	179.99
3		delivered	2017-08-16 13:09:20	2017-08-17 3:10:27	2017-08-19 11:34:20	2017-09-13 20:09:02	2017-09-14 0:00:00	149.90	149.90
4		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
5		delivered	2017-04-12 08:35:12	2017-04-12 8:50:12	2017-04-12 17:09:42	2017-04-20 16:12:28	2017-05-04 0:00:00	59.99	59.99
6		delivered	2018-03-02 17:47:40	2018-03-02 14:10:38	2018-03-07 21:07:51	2018-04-17 17:17:54	2018-03-22 0:00:00	34.90	34.90
7		delivered	2017-12-18 11:08:30	2017-12-18 12:45:31	2017-12-19 20:09:54	2017-12-28 20:08:33	2018-01-12 0:00:00	120.90	120.90
8		delivered	2017-09-17 18:04:44	2017-09-17 18:15:13	2017-09-18 21:32:46	2017-10-02 21:14:31	2017-10-13 0:00:00	89.99	89.99

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	6155908.08	12264.554035
2	2018	7388050.80	19.985094

```
plt.figure(figsize=(8,4))
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()
```

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}%")
```

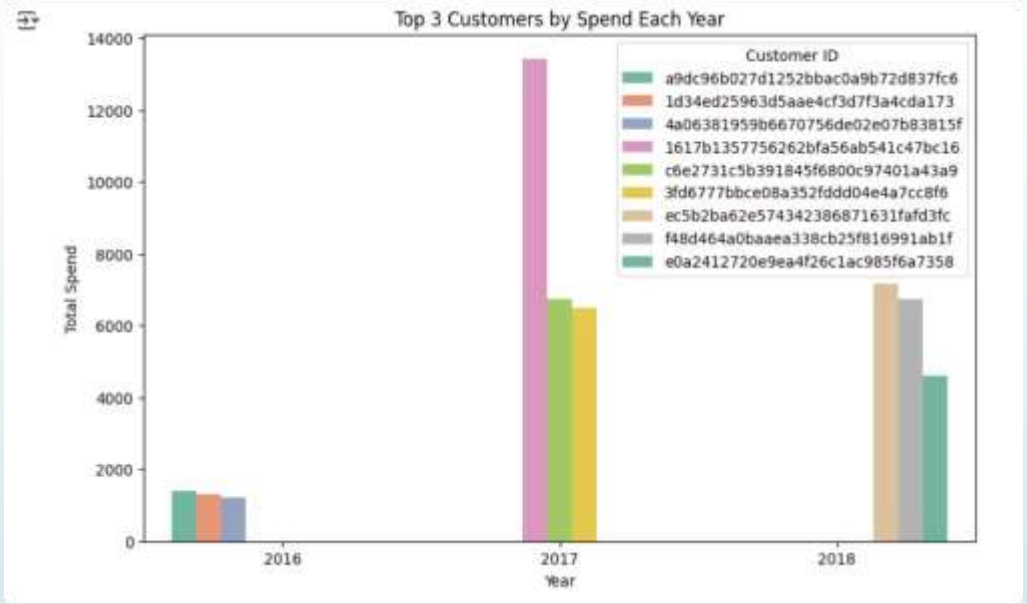
```
6-Month Repeat Purchase Retention Rate: 0.00%
```

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

```
[ ] 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()
```



BUSINESS IMPACT

1. Retention Insights ³ can drive Loyalty campaigns to improve repeat rate.
2. Payment Insights ³ optimize checkout experience, expand popular payment methods. Delivery delays ³ logistics optimization needed to improve customer satisfaction.
3. Top Customers ³ VIP strategy for high-spend customers.
4. Category Trends ³ guide Inventory and marketing focus for high-demand products.

CONCLUSION

1. The project successfully combined **SQL** and **Python** to analyze e-commerce data.
2. **SQL** helped extract and organize the data.
3. **Python** added deeper analysis and clear visualizations.
4. Together, they created a complete **end-to-end data workflow**:
5. **Raw Data → Cleaned Data → Queries → Insights → Business Decisions**
6. This project shows how using multiple tools together can solve real-world data problems effectively.

THANK YOU

LinkedIn

GitHub