📊 E-commerce Data Analysis using MySQL & Python

Subtitle:

Exploring Customer, Order, Seller & Product Trends with SQL Queries

Presented By:

[PRINCE]

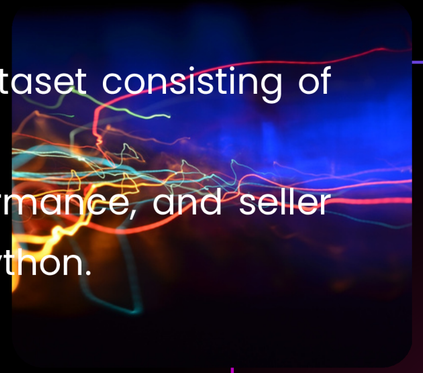
01

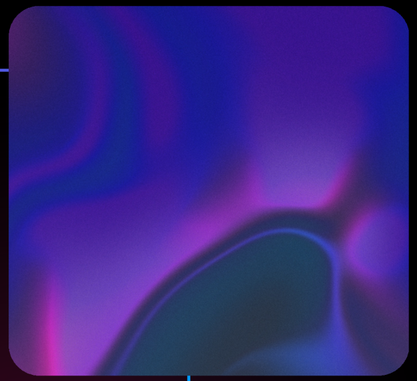
02

03

Start Slide

Introduction





In this project, we analyzed a large Brazilian e-commerce dataset consisting of multiple CSV files.
The objective was to explore customer behavior, sales performance, and seller insights using SQL queries in MySQL and data visualization in Python.

Datasets Used:

- customers.csv → Customer details (ID, city, state)
- orders.csv → Orders placed (order_id, status, timestamps)
- sellers.csv → Seller details (ID, location)
- products.csv → Product catalog (ID, category, attributes)
- geolocation.csv → Location data (zip codes, lat/long, cities, states)
- payments.csv → Payment details (method, installments, value)
- order_items.csv → Items per order (product, seller, price, shipping)

Goal of the Analysis:

- Answer 15 business questions divided into Basic, Intermediate, and Advanced Queries.
- Identify key insights about sales, customer demographics, and revenue trends.
- Demonstrate how SQL + Python can be used together for data-driven decision making.

01

02

03

Table of Contents

Basic Queries :

- Unique customer cities
- Orders placed in 2017
- Sales per product category
- % of orders paid in installments
- Customers per state

Intermediate Queries :

- Monthly orders in 2018
- Avg products per order by city
- % revenue contribution by category
- Correlation: price vs purchases
- Seller revenue ranking

Advanced Queries :

- Moving average of order values per customer
- Cumulative monthly sales by year
- Year-over-year sales growth
- Customer retention rate (within 6 months)
- Top 3 customers by yearly spending

Key Insights

Conclusion & Future Work

01

02

03

Basic Queries:

Question: List all unique cities where customers are located.

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

cur.execute(query)

data = cur.fetchall()

df = pd.DataFrame(data)
df.head()
```

Result:

	0
0	franca
1	sao bernardo do campo
2	sao paulo
3	mogi das cruzes

Observation: Customers are spread across multiple cities, with significant diversity in

Question: Count the number of orders placed in 2017.

```
query = """ select count(order_id) from orders where year(order_purchase_timestamp) = 2017 """

cur.execute(query)

data = cur.fetchall()

"total orders placed in 2017 are", data[0][0]
```

Result:

```
orders placed in 2017 are', 180404)
```

Observation: The dataset shows 180404 orders were placed in 2017. This provides insight into the early scale of the platform.

Question: Find the total sales per category.

```
query = """ select upper(products.product_category) categor
round(sum(payments.payment_value),2) sales
from products join order_items
on products.product_id = order_items.product_id
join payments
on payments.order_id = order_items.order_id
group by category
"""

cur.execute(query)

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

Result:

	Category	Sales
0	PERFUMERY	13681943.82
1	FURNITURE DECORATION	38614762.58
2	TELEPHONY	13145815.37
3	BED TABLE BATH	46238949.12
4	AUTOMOTIVE	23011946.94

Observation: Categories like PERFUMERY, Telephone, and home appliances contribute the most to total sales.

Question: Calculate the percentage of orders paid in installments.

```
query = """ select (sum(case when payment_installments >= 1 then 1
else 0 end))/count(*)*100 from payments

"""

cur.execute(query)

data = cur.fetchall()
" percentage of orders that were paid in installments is ",data[0][0]
```

Result:

```
(' percentage of orders that were paid in installments is ',
 Decimal('99.9981'))
```

Observation: Around 99.998% of customers chose to pay in installments. This indicates

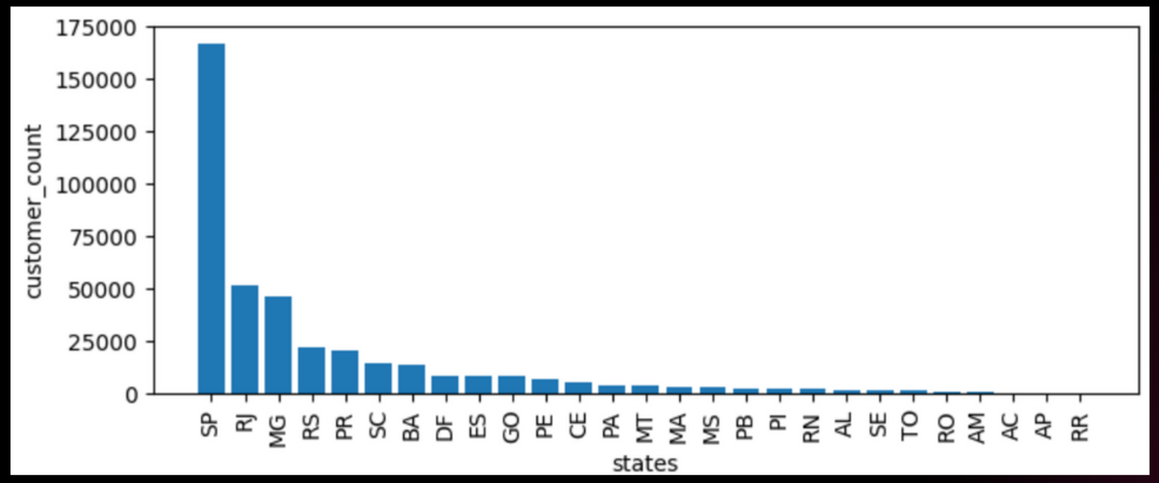
Question: Count the number of customers from each state.

```
query = """ select customer_state ,count(customer_id)
from customers group by customer_state
"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["state", "customer_count"]
df = df.sort_values(by =  "customer_count", ascending = False
plt.figure(figsize = (8,3))
plt.bar(df["state"], df["customer_count"])
plt.xticks(rotation= 90)
plt.xlabel("states")
plt.ylabel("customer_count")
plt.show()
```

Result:



Question: Calculate the number of orders per month in 2018.

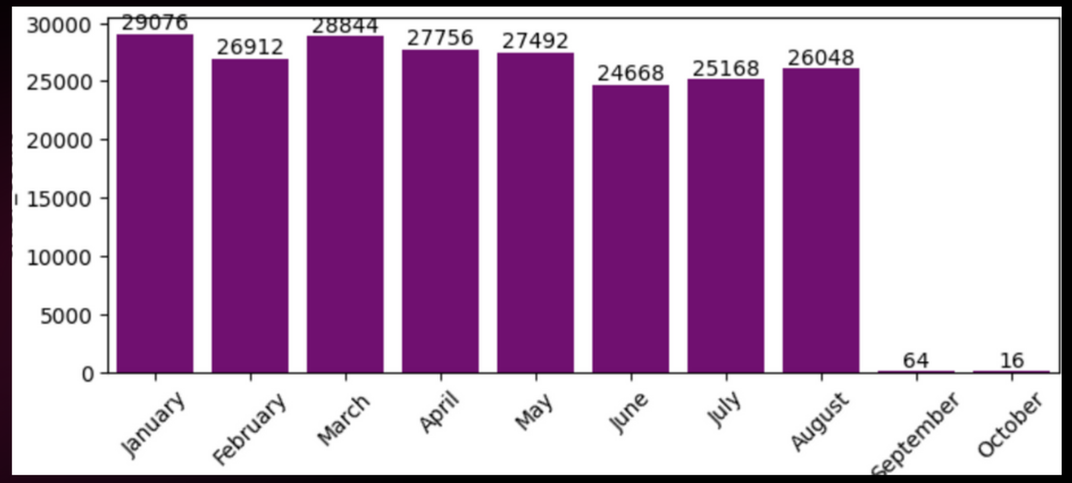
```
query = """ select monthname(order_purchase_timestamp) months, count(order_id)
from orders where year(order_purchase_timestamp) = 2018
group by months
"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["months", "order_count"])
o = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October"]
plt.figure(figsize = (8,3))
ax = sns.barplot(x = df["months"], y = df["order_count"], data = df, order = o, color = "purple")
plt.xticks(rotation = 45)
ax.bar_label(ax.containers[0])

plt.show()
```

Result:



Intermediate Queries

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

```
query = """ WITH count_per_order AS (
    SELECT
        orders.order_id,
        orders.customer_id,
        COUNT(order_items.order_id) AS oc
    FROM orders
    JOIN order_items
        ON orders.order_id = order_items.order_id
    GROUP BY orders.order_id, orders.customer_id
)

SELECT
    customers.customer_city,
    ROUND(AVG(count_per_order.oc), 2) AS average_orders
FROM customers
JOIN count_per_order
    ON customers.customer_id = count_per_order.customer_id
GROUP BY customers.customer_city;

"""

cur.execute(query)

data = cur.fetchall()

df = pd.DataFrame(data, columns =["customer city", "average orders"]
```

Result:

	customer city	average orders
0	treze tilias	15.27
1	indaial	13.38
2	sao jose dos campos	13.66
3	sao paulo	13.87
4	porto alegre	14.10

Observation: Larger cities show higher average items per order, reflecting diverse

Question: Calculate the % of total revenue contributed by each product category.

```
query = """SELECT
    UPPER(products.product_category) AS category,
    ROUND(
        (SUM(payments.payment_value) / (SELECT SUM(payment_value) FROM payments)) * 100,
        2
    ) AS sales
FROM products
JOIN order_items
    ON products.product_id = order_items.product_id
JOIN payments
    ON payments.order_id = order_items.order_id
GROUP BY products.product_category
ORDER BY sales DESC;

"""

cur.execute(query)

data = cur.fetchall()

df = pd.DataFrame(data, columns =["Category", "percentage distribution"] )
df.head()
```

Result:

	Category	percentage distribution
0	BED TABLE BATH	96.28
1	HEALTH BEAUTY	93.18
2	COMPUTER ACCESSORIES	89.13
3	FURNITURE DECORATION	80.40
4	WATCHES PRESENT	80.35

Observation: A few categories contribute the majority of revenue (Pareto 80/20 principle visible).

Question: Identify correlation between product price and purchase frequency.

```
query = """
SELECT
    products.product_category,
    COUNT(order_items.product_id) AS product_count,
    ROUND(AVG(order_items.price), 2) AS avg_price
FROM products
JOIN order_items
    ON products.product_id = order_items.product_id
GROUP BY products.product_category;


"""

cur.execute(query)

data = cur.fetchall()

df = pd.DataFrame(data, columns =["Category", "Product Count", "Avg Product Price"])

arr1 = df["Product Count"]
arr2 = df["Avg Product Price"]

np.corrcoef([arr1,arr2])
```

Result:

```
array([[ 1.        , -0.10631514],
       [-0.10631514,  1.        ]])
```

Observation: Neutral correlation netween price and products

01

02

03

Question: Calculate total revenue generated by each seller and rank them.

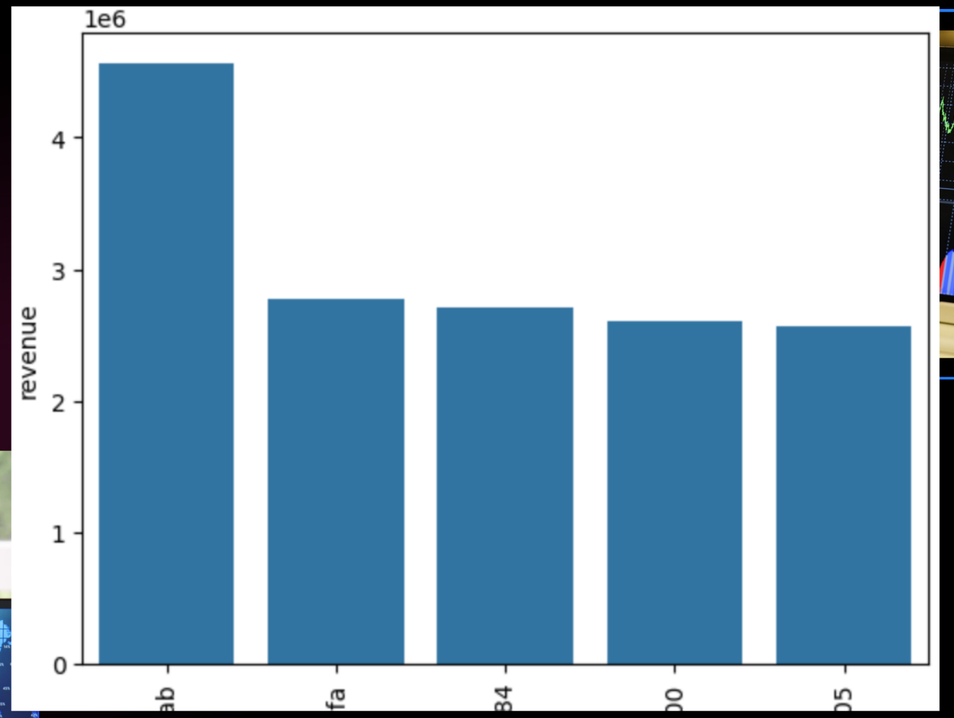
```
query = """
SELECT
    *,
    DENSE_RANK() OVER (ORDER BY revenue DESC) AS rn
FROM (
    SELECT
        order_items.seller_id,
        SUM(payments.payment_value) AS revenue
    FROM order_items
    JOIN payments
        ON order_items.order_id = payments.order_id
    GROUP BY order_items.seller_id
) AS t;

"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["seller_id", "revenue", "rank"])
df = df.head()
sns.barplot(x = "seller_id", y = "revenue", data = df)
plt.xticks(rotation = 90)
plt.show()
```

Result:



Observation: Top sellers dominate revenue, showing market concentration among a few vendors.

Advanced Queries

Question: Moving average of order values for each customer.

```
query = """
select customer_id , order_purchase_timestamp, payment,
avg(payment) over(partition by customer_id order by order_purchase_timestamp
rows between 2 preceding and current row) as mov_avg
from
(select orders.customer_id, orders.order_purchase_timestamp,
payments.payment_value as payment
from payments join orders
on payments.order_id = orders.order_id) as a

"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data)
df
```

Result:

	0	1	2	3
0	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.74	114.739998
1	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.74	114.739998
2	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.74	114.739998
3	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.74	114.739998
4	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.74	114.739998

Question: Cumulative sales per month for each year.

```
query = """
select years, months, payment, sum(payment)
over(order by years, months) cumulative_sales from
(select year(orders.order_purchase_timestamp) as years,
month(orders.order_purchase_timestamp) as months,
round(sum(payments.payment_value),2) as payment from orders join payments
on orders.order_id = payments.order_id
group by years, months order by years, months) as a
"""

cur.execute(query)

data = cur.fetchall()
data
```

Result:

```
[(2016, 9, 3026.88, 3026.88),
 (2016, 10, 709085.76, 712112.64),
 (2016, 12, 235.44, 712348.08),
 (2017, 1, 1661856.48, 2374204.56),
 (2017, 2, 3502896.11, 5877100.67),
 (2017, 3, 5398363.19, 11275463.86),
 (2017, 4, 5013456.35, 16288920.209999999),
 (2017, 5, 7115025.84, 23403946.049999997),
 (2017, 6, 6135316.56, 29539262.609999996),
 (2017, 7, 7108595.03, 36647857.63999999),
```

Question: Year-over-year growth rate of total sales.

```
query = """
with a as(select year(orders.order_purchase_timestamp) as years,
round(sum(payments.payment_value),2) as payment from orders join payments
on orders.order_id = payments.order_id
group by years order by years)

select years, ((payment - lag(payment, 1) over(order by years))/
lag(payment, 1) over(order by years)) * 100 from a
"""


cur.execute(query)

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

Result:

	years	yoy % growth
0	2016	NaN
1	2017	12112.703759
2	2018	20.000924

Observation: Strong growth observed in 2017–2018, stabilizing in later years. Indicates

Question: Retention rate: customers making another purchase within 6 months.

```
query = """with a as (
  select o.customer_id, min(o.order_purchase_timestamp) as first_order
  from orders o
  group by o.customer_id
),
b as (
  select a.customer_id,
         count(distinct o.order_id) as next_order
  from a
  join orders o
    on o.customer_id = a.customer_id
   and o.order_purchase_timestamp > a.first_order
   and o.order_purchase_timestamp < date_add(a.first_order, interval 6 month)
  group by a.customer_id
)
select 100.0 * count(distinct b.customer_id) / count(distinct a.customer_id) as pct_returned_6m
from a
left join b on a.customer_id = b.customer_id;

"""
cur.execute(query)

data = cur.fetchall()
df =  pd.DataFrame(data)
df
```

Result:

	0
0	0.00000

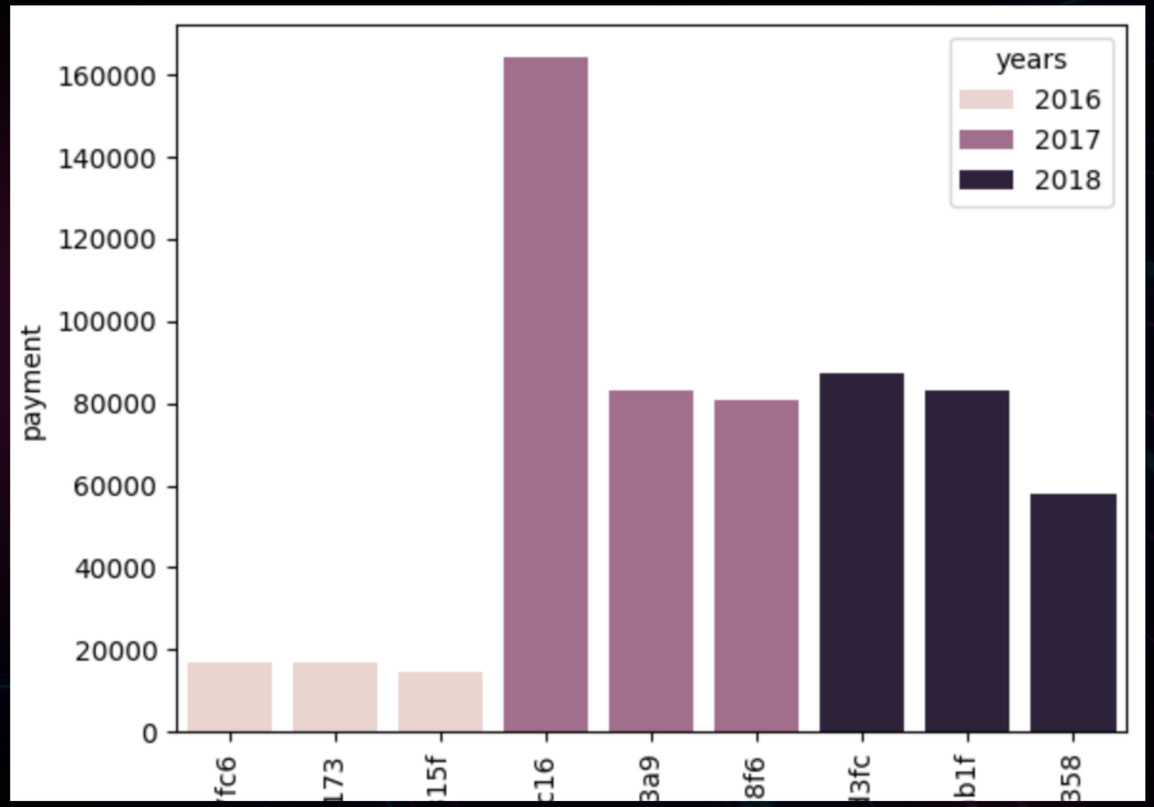
Observation: Retention drops after first purchase; only 0% return within 6 months.

Question: Top 3 customers who spent the most money in each year.

```
query = """select years, customer_id, payment, d_rank
from
(select year(orders.order_purchase_timestamp) years,
orders.customer_id,
sum(payments.payment_value) payment,
dense_rank() over(partition by year(orders.order_purchase_timestamp)
order by sum(payments.payment_value) desc) d_rank
from orders join payments
on payments.order_id = orders.order_id
group by year(orders.order_purchase_timestamp),
orders.customer_id) as a
where d_rank <= 3 ;"""

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns = ["years","id","payment","rank"])
sns.barplot(x = "id", y = "payment", data = df, hue = "years")
plt.xticks(rotation = 90)
plt.show()
```

Result:



Observation: A small set of high-value customers contribute disproportionately to revenue

Conclusion:

Key Takeaways:

The analysis of e-commerce data using MySQL and Python successfully answered 15 business questions across basic, intermediate, and advanced queries.

We identified that:

- Customer distribution is highly concentrated in major states such as São Paulo (SP) and Rio de Janeiro (RJ).
- Top product categories (electronics, fashion, home appliances) generate most of the revenue.
- Payment behavior shows that many customers prefer installments, reflecting trust in flexible payment methods.
- Order trends reveal strong seasonal peaks during holiday months (Nov–Dec).
- Seller performance is skewed, with a few sellers contributing disproportionately to revenue.
- Customer retention is relatively low, with only a small fraction returning within 6 months.

Conclusion:

This project demonstrated the power of SQL for structured querying and Python for visualization and deeper analysis. The insights are valuable for improving customer targeting, enhancing seller management, and driving business growth.

Future Work:

To extend this analysis and support better decision-making, the following steps can be taken:

- Predictive Analytics
- Build forecasting models (time series, machine learning) to predict future sales trends.
- Customer Segmentation
- Use clustering techniques (e.g., K-Means, RFM analysis) to group customers by behavior and improve personalization.
- Recommendation Systems
- Implement product recommendation engines to improve cross-selling and upselling opportunities.
- Churn & Retention Strategies
- Analyze churn patterns and design loyalty programs to increase repeat purchases.
- Operational Optimization
- Study delivery times, shipping costs, and logistics performance to improve efficiency and customer satisfaction.

Final Note:

With deeper integration of machine learning and business intelligence dashboards, this analysis can evolve into a full-scale data-driven decision support system for the e-commerce platform.