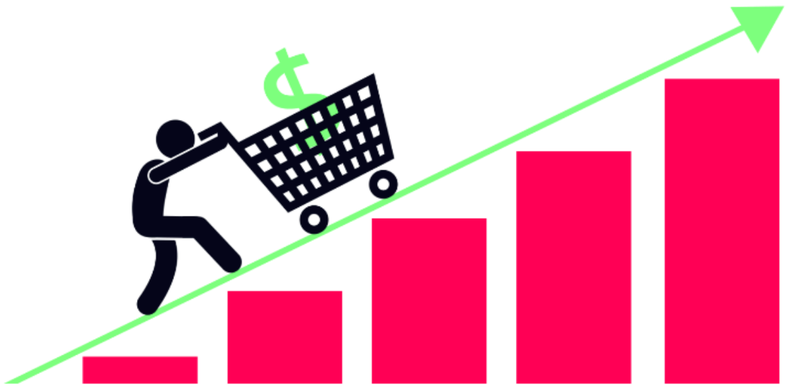
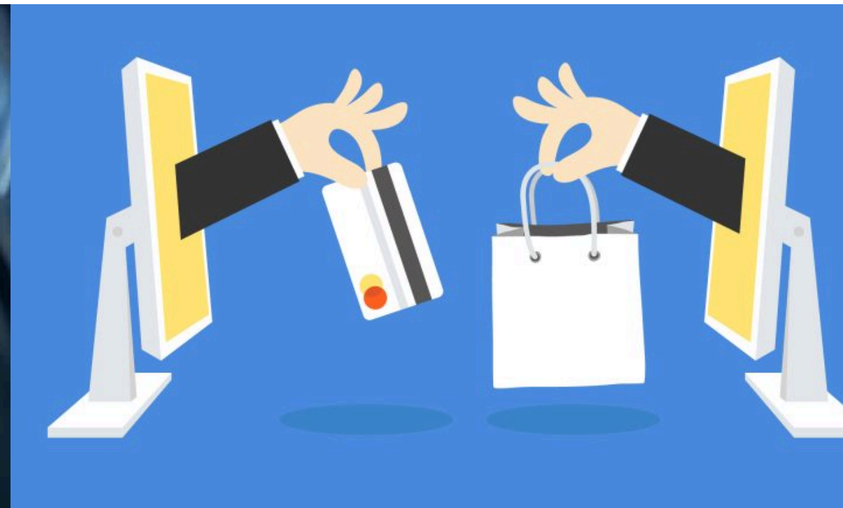
E-commerce Sales Analysis





About Dataset

Target is a globally recognized brand and a leading retailer in the United States, known for offering exceptional value, inspiration, innovation, and a unique shopping experience.

This dataset focuses on Target's operations in Brazil, covering 100,000 orders placed between 2016 and 2018. It includes detailed information on order status, pricing, payment and shipping performance, customer locations, product attributes, and customer reviews.

- **Orders:**The dataset includes details about customer orders, including order dates, statuses, total amounts, and associated products.
- **Payments:** Information on various payment methods used by customers, including credit cards, debit cards, UPI, and vouchers, along with their transaction statuses.
- **Products:** A comprehensive list of products available for purchase, including product names, categories, prices, and stock levels.
- **Sellers:** Provides information about sellers, including seller IDs, names, locations, and their respective product offerings.
- **Geolocation:** Data capturing the geographical locations of customers and sellers, which can be used for analyzing regional sales patterns.
- **Order Items:** A breakdown of items included in each order, detailing quantities and product IDs for effective inventory management.
- **Customers:** Information about customers, including demographics, order history, and preferences, which can be used for targeted marketing strategies.

```sql
-- 1. Find the number of unique states and their names from the database.
SELECT DISTINCT customer_state
FROM customers;

SELECT COUNT(DISTINCT customer_state) AS state_count
FROM customers;
```

Result Grid

	state_count
▶	27

Result Grid | Filter Row

	customer_state
▶	SP
	SC
	MG
	PR
	RJ
	RS
	PA
	GO
	ES
	BA
	MA
	MS

```sql
-- 2. Find the number of unique cities and their names from the database.
SELECT DISTINCT customer_city
FROM customers;

SELECT COUNT(DISTINCT customer_city) AS city_count
FROM customers;
```

Result Grid

	city_count
▶	4119

Result Grid | Filter Rows:

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

```sql
-- 3. List all unique cities where customers are located.
SELECT DISTINCT customer_city
FROM customers;
```

Result Grid | Filter R

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

```sql
-- 4. Count the number of orders placed in 2017.?
SELECT COUNT(order_id)
FROM orders
WHERE YEAR(order_purchase_timestamp) = 2017;
```

Result Grid | Filter Rows:

	COUNT(order_id)
▶	90202

```sql
-- 5. Find the total sales per category.?
SELECT UPPER(products.product_category) AS category,
       ROUND(SUM(payments.payment_value), 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 category;
```

Result Grid | Filter Rows: | Export:

category	sales
PERFUMERY	4053909.28
FURNITURE DECORATION	11441411.13
TELEPHONY	3895056.41
FASHION BAGS AND ACCESSORIES	1745266.24
BED TABLE BATH	13700429.37
AUTOMOTIVE	6818354.65
COMPUTER ACCESSORIES	12682643.57
HOUSEWARES	8758065.04
BABIES	4318765.28
TOYS	4952301.52
FURNITURE OFFICE	5174611.9
COOL STUFF	6237584

```sql
-- 6. Calculate the percentage of orders that were paid in installments.?
SELECT ((SUM(CASE WHEN payment_installments >= 1 THEN 1 ELSE 0 END) / COUNT(*)) * 100) AS percentage
FROM payments;
```

Result Grid | Filter Rows:

	percentage
▶	99.9981

```sql
-- 7. 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;
```

Result Grid | Filter Rows:

customer_state	Customer_Count
SP	83492
RJ	25704
MG	23270
RS	10932
PR	10090
SC	7274
BA	6760
DF	4280
ES	4066
GO	4040
PE	3304
CE	2672

```sql
-- 8. Count customers by city and state.
SELECT customer_city, customer_state, COUNT(DISTINCT customer_unique_id) AS unique_customers
FROM customers
GROUP BY customer_city, customer_state
ORDER BY unique_customers DESC;
```

Result Grid | Filter Rows: | Export:

customer_city	customer_state	unique_customers
sao paulo	SP	14984
rio de janeiro	RJ	6620
belo horizonte	MG	2672
brasilia	DF	2069
curitiba	PR	1465
campinas	SP	1398
porto alegre	RS	1326
salvador	BA	1209
guarulhos	SP	1153
sao bernardo do campo	SP	908
niteroi	RJ	811

```sql
-- 9. What is the total number of orders placed in each month?
SELECT
    MONTH(order_purchase_timestamp) AS month,
    COUNT(order_id) AS total_orders
FROM
    orders
GROUP BY
    MONTH(order_purchase_timestamp)
ORDER BY month ;
```

Result Grid | Filter Rows:

month	total_orders
1	16138
2	17016
3	19786
4	18686
5	21146
6	18824
7	20636
8	21686
9	8610
10	9918
11	15088
12	11348

```sql
-- 10. Identify the order with the highest total value.
SELECT
    orders.order_id,
    SUM(payments.payment_value) AS total_value
FROM
    orders
JOIN
    payments ON orders.order_id = payments.order_id
GROUP BY
    orders.order_id
ORDER BY
    total_value DESC
LIMIT 1;
```

Result Grid | Filter Rows: | Export: | W

order_id	total_value
03caa2c082116e1d31e67e9ae3700499	54656.3203125

```sql
-- 11. How many orders were placed by each customer?
SELECT
    customer_id,
    COUNT(order_id) AS total_orders
FROM
    orders
GROUP BY
    customer_id
ORDER BY total_orders;
```

Result Grid | Filter Rows: | Export:

customer_id	total_orders
e04757bb7741d0781cda14c9be20ad2a	2
96c3d51816ee07cb78ebbfe0e8bdb889	2
f1913159750548eb81dca4b69fbd5e0c	2
6967af003c642a30a79242f2756a73d5	2
5eef8c5a24bd948fac341c6ebe3ce41e	2
2dc86eb45d955da9a9afa6f431ac00f4	2
92f1d4d2f1a2a03f858f7afea11ff0b9	2
03ee2343040a3871cb2cf313cc795c72	2
9573415dd691e34a2b92c9e39558ef22	2
bc64988c743d4ea05faa42f945a49396	2
7b25189538bd6c13bf9440f0d3214a02	2
e3ecfba8964e0757587b1b1913499c1a	2

```sql
-- 12. What percentage of orders use each payment type?
DESCRIBE payments;

SELECT payment_type,
       COUNT(*) * 100.0 / (SELECT COUNT(*) FROM payments) AS percentage
FROM payments
GROUP BY payment_type;
```

Result Grid | Filter Rows:

payment_type	percentage
credit_card	73.92238
UPI	19.04395
voucher	5.55898
debit_card	1.47181
not_defined	0.00289

```sql
-- 13.What is the average order value for each payment type?
SELECT payment_type,
       AVG(payments.payment_value) AS avg_order_value
FROM payments
JOIN orders ON payments.order_id = orders.order_id
GROUP BY payment_type;
```

Result Grid | Filter Rows:

payment_type	avg_order_value
credit_card	163.3190206416755
voucher	65.70335418108189
UPI	145.03443537677197
debit_card	142.57017003764818
not_defined	0

```sql
-- 14. Which payment type contributes the most to the total revenue
SELECT payment_type,
       SUM(payments.payment_value) AS total_revenue
FROM payments
GROUP BY payment_type
ORDER BY total_revenue DESC;
```

Result Grid | Filter Rows:

payment_type	total_revenue
credit_card	25084168.38035495
UPI	5738722.538988113
voucher	758873.7407914959
debit_card	435979.5799751282
not_defined	0

```sql
-- 15. How does the frequency of each payment type change over time (monthly/yearly)?
SELECT YEAR(orders.order_purchase_timestamp) AS year,
       payment_type,
       COUNT(*) AS order_count
FROM payments
JOIN orders ON payments.order_id = orders.order_id
GROUP BY year, payment_type
ORDER BY year, payment_type;
```

Result Grid | Filter Rows:

year	payment_type	order_count
2016	credit_card	1032
2016	debit_card	8
2016	UPI	252
2016	voucher	92
2017	credit_card	138272
2017	debit_card	1688
2017	UPI	38032
2017	voucher	12108
2018	credit_card	167876
2018	debit_card	4420
2018	not_defined	12
2018	UPI	40852

```sql
-- 16. What percentage of orders are made using undefined payment methods (not_defined)?
SELECT COUNT(*) * 100.0 / (SELECT COUNT(*) FROM payments) AS undefined_percentage
FROM payments
WHERE payment_type = 'not_defined';
```

Result Grid | Filter Rows:

	undefined_percentage
▶	0.00289

```sql
-- 17. which city and state customers use specific payment methods such as credit_card, UPI, voucher, debit_card, and not_defined to purchase products
SELECT
    customers.customer_city AS city,
    customers.customer_state AS state,
    payments.payment_type,
    COUNT(*) AS order_count
FROM customers
JOIN orders
    ON customers.customer_id = orders.customer_id
JOIN payments
    ON orders.order_id = payments.order_id
WHERE payments.payment_type IN ('credit_card', 'UPI', 'voucher', 'debit_card', 'not_defined')
GROUP BY customers.customer_city, customers.customer_state, payments.payment_type
ORDER BY order_count DESC;
```

Result Grid | Filter Rows: | Export

city	state	payment_type	order_count
sao paulo	SP	credit_card	97808
rio de janeiro	RJ	credit_card	44200
sao paulo	SP	UPI	22264
belo horizonte	MG	credit_card	17864
brasilia	DF	credit_card	13544
curitiba	PR	credit_card	9088
rio de janeiro	RJ	UPI	8984
campinas	SP	credit_card	8624
porto alegre	RS	credit_card	8432
salvador	BA	credit_card	8144

```sql
-- 18. which state customers use each payment method (like credit_card, UPI, voucher, debit_card, and not_defined)
SELECT
    customers.customer_state AS state,
    payments.payment_type,
    COUNT(*) AS order_count
FROM customers
JOIN orders
    ON customers.customer_id = orders.customer_id
JOIN payments
    ON orders.order_id = payments.order_id
WHERE payments.payment_type IN ('credit_card', 'UPI', 'voucher', 'debit_card', 'not_defined')
GROUP BY customers.customer_state, payments.payment_type
ORDER BY order_count DESC;
```

Result Grid | Filter Rows:

state	payment_type	order_count
SP	credit_card	257344
RJ	credit_card	82304
MG	credit_card	72560
SP	UPI	65640
RS	credit_card	31880
PR	credit_card	30288
SC	credit_card	21704
BA	credit_card	21296
SP	voucher	19896
MG	UPI	18432
RJ	UPI	17394

```sql
-- 19. Which cities have the highest number of orders?
SELECT customers.customer_city, COUNT(orders.order_id) AS total_orders
FROM orders
JOIN customers ON orders.customer_id = customers.customer_id
GROUP BY customers.customer_city
ORDER BY total_orders DESC
LIMIT 10;
```

Result Grid | Filter Rows:

customer_city	total_orders
sao paulo	62160
rio de janeiro	27528
belo horizonte	11092
brasilia	8524
curitiba	6084
campinas	5776
porto alegre	5516
salvador	4980
guarulhos	4756
sao bernardo do campo	3752

```sql
-- 20. What is the Average Payment Amount per Transaction?
SELECT AVG(payment_value) AS average_payment
FROM payments;
```

Result Grid | Filter Rows:

	average_payment
▶	154.10038041752344

```sql
-- 21. List the Top 5 Products Based on Total Sales Revenue
SELECT product_id, SUM(payment_value) AS total_sales
FROM order_items
JOIN payments ON order_items.order_id = payments.order_id
GROUP BY product_id
ORDER BY total_sales DESC
LIMIT 5;
```

Result Grid | Filter Rows: | Export:

product_id	total_sales
5769ef0a239114ac3a854af00df129e4	437250.5625
bb50f2e236e5eea0100680137654686c	327549.6779460907
422879e10f46682990de24d770e7f83d	318048.88179826736
d1c427060a0f73f6b889a5c7c61f2ac4	282231.59934043884
6cdd53843498f92890544667809f1595	259302.68002319336

```sql
-- 22. How Many Different Categories of Products Are Available?
DESCRIBE products;

SELECT COUNT(DISTINCT product_category) AS total_categories
FROM products;
```

Result Grid

	total_categories
▶	73

```sql
-- 23. Who are the Top 3 Sellers Based on Total Revenue Generated?
SELECT seller_id, SUM(payment_value) AS total_revenue
FROM orders
JOIN order_items ON orders.order_id = order_items.order_id
JOIN payments ON orders.order_id = payments.order_id
GROUP BY seller_id
ORDER BY total_revenue DESC
LIMIT 3;
```

Result Grid | Filter Rows: | Export:

seller_id	total_revenue
7c67e1448b00f6e969d365cea6b010ab	4057335.2584171295
1025f0e2d44d7041d6cf58b6550e0bfa	2465776.3187217712
4a3ca9315b744ce9f8e9374361493884	2409962.158122301

```sql
-- 24. How many products does each seller offer?
DESCRIBE sellers;
DESCRIBE sellers;

SELECT
    sellers.seller_id,
    sellers.seller_city,
    COUNT(products.product_id) AS product_count
FROM
    sellers
JOIN
    order_items ON sellers.seller_id = order_items.seller_id
JOIN
    products ON order_items.product_id = products.product_id
GROUP BY
    sellers.seller_id, sellers.seller_city
LIMIT 0, 500;
```

Result Grid | Filter Rows: | Export: | Wrap

seller_id	seller_city	product_count
7b07b3c7487f0ea825fc6df75abd658b	sao paulo	784
834f3294fba9f932f56edc879193f925	araraquara	528
391fc6631aebcf3004804e51b40bcf1e	ibitinga	4904
a420f60ff1aa9acc80d0e42959f2b313	sao paulo	552
3785b653b1b82de85ab47dd139938091	ampere	192
a6fe7de3d16f6149ffe280349a8535a0	franca	416
6560211a19b47992c3666cc44a7e94c0	sao paulo	16264
827f8f69dfa529c561901c4f2e0f332f	curitiba	832
da8622b14eb17ae2831f4ac5b9dab84a	piracicaba	12408
cc419e0650a3c5ba77189a1882b7556a	santo andre	14200
323ce52b5b81df2cd804b017b7f09aa7	sao paulo	240

```sql
-- INTERMEDIATE Queries
-- 25. Calculate the number of orders per month in 2018.?
SELECT MONTHNAME(order_purchase_timestamp) AS months,
       COUNT(order_id) AS order_count
FROM orders
WHERE YEAR(order_purchase_timestamp) = 2018
GROUP BY months;
```

Result Grid | Filter Rows:

months	order_count
July	12584
August	13024
February	13456
June	12334
March	14422
January	14538
May	13746
April	13878
September	32
October	8

```sql
-- 26. Find the average number of products per order, grouped by customer city.?
SELECT * FROM ecommerce.orders;
SELECT * FROM ecommerce.order_items;

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
ORDER BY average_orders DESC;
```

Result Grid | Filter Rows:

customer_city	average_orders
padre carvalho	28.00
celso ramos	26.00
datas	24.00
candido godoi	24.00
matias olimpio	20.00
cidelandia	16.00
picarra	16.00
morro de sao paulo	16.00
teixeira soares	16.00
curralinho	16.00
inconfidentes	14.00
inua	13.00

```sql
-- 27. Calculate the percentage of total revenue contributed by each product category.?
SELECT
    UPPER(products.product_category) AS category,
    ROUND((SUM(payments.payment_value) / (SELECT SUM(payment_value) FROM payments)) * 100,2) AS sales_percentage
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
ORDER BY sales_percentage DESC;
```

Result Grid | Filter Rows:

category	sales_percentage
BED TABLE BATH	42.79
HEALTH BEAUTY	41.41
COMPUTER ACCESSORIES	39.61
FURNITURE DECORATION	35.73
WATCHES PRESENT	35.71
SPORT LEISURE	34.78
HOUSEWARES	27.35
AUTOMOTIVE	21.3
GARDEN TOOLS	20.95
COOL STUFF	19.48
FURNITURE OFFICE	16.16

```sql
-- 28. Identify the correlation between product price and the number of times a product has been purchased.?
WITH category_data AS (
    SELECT
        products.product_category,
        COUNT(order_items.product_id) AS order_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 ),
correlation_data AS (
    SELECT
        COUNT(*) AS n,
        SUM(order_count) AS sum_order_count,
        SUM(avg_price) AS sum_avg_price,
        SUM(order_count * avg_price) AS sum_product,
        SUM(order_count * order_count) AS sum_order_count_sq,
        SUM(avg_price * avg_price) AS sum_avg_price_sq
    FROM category_data )

SELECT
    (n * sum_product - sum_order_count * sum_avg_price) /
    (SQRT(n * sum_order_count_sq - sum_order_count * sum_order_count) * SQRT(n * sum_avg_price_sq - sum_avg_price * sum_avg_price)) AS correlation
FROM
    correlation_data;
```

Result Grid | Filter Rows:

	correlation
▶	-0.106315141671576

```sql
-- 29. Calculate the total revenue generated by each seller, and rank them by revenue ?
SELECT * FROM ecommerce.sellers;
SELECT * FROM ecommerce.order_items;
SELECT * FROM ecommerce.payments;

SELECT *,
       DENSE_RANK() OVER (ORDER BY revenue DESC) AS ranks
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 a;
```

Result Grid | Filter Rows: | Export: | Wrap

seller_id	revenue	ranks
7c67e1448b00f6e969d365cea6b010ab	2028667.6292085648	1
1025f0e2d44d7041d6cf58b6550e0bfa	1232888.1593608856	2
4a3ca9315b744ce9f8e9374361493884	1204981.0790611506	3
1f50f920176fa81dab994f9023523100	1161013.6805104613	4
53243585a1d6dc2643021fd1853d8905	1139612.3219909668	5
da8622b14eb17ae2831f4ac5b9dab84a	1088877.2772586346	6
4869f7a5dfa277a7dca6462dcf3b52b2	1056664.4837551117	7
955fee9216a65b617aa5c0531780ce60	945289.2020090595	8
fa1c13f2614d7b5c4749cbc52fecda94	826052.9194793701	9
7e93a43ef30c4f03f38b393420bc753a	740536.8388252258	10

```sql
-- ADVANCED Queries
-- 30. Calculate the moving average of order values for each customer over their order history.?
SELECT * FROM ecommerce.orders;

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
    ORDER BY mov_avg DESC;
```

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

customer_id	order_purchase_timestamp	payment	mov_avg
1617b1357756262bfa56ab541c47bc16	2017-09-29 15:24:52	13664.1	13664.080078125
1617b1357756262bfa56ab541c47bc16	2017-09-29 15:24:52	13664.1	13664.080078125
1617b1357756262bfa56ab541c47bc16	2017-09-29 15:24:52	13664.1	13664.080078125
1617b1357756262bfa56ab541c47bc16	2017-09-29 15:24:52	13664.1	13664.080078125
ec5b2ba62e574342386871631fafd3fc	2018-07-15 14:49:44	7274.88	7274.8798828125
ec5b2ba62e574342386871631fafd3fc	2018-07-15 14:49:44	7274.88	7274.8798828125
ec5b2ba62e574342386871631fafd3fc	2018-07-15 14:49:44	7274.88	7274.8798828125
ec5b2ba62e574342386871631fafd3fc	2018-07-15 14:49:44	7274.88	7274.8798828125
c6e2731c5b391845f6800c97401a43a9	2017-02-12 20:37:36	6929.31	6929.31005859375
c6e2731c5b391845f6800c97401a43a9	2017-02-12 20:37:36	6929.31	6929.31005859375

```sql
-- 31. Calculate the cumulative sales per month for each year.?
SELECT * FROM ecommerce.orders;

SELECT
    years, months, payment, SUM(payment) OVER (ORDER BY years, months) AS 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;
```

Result Grid | Filter Rows: | Export:

years	months	payment	cumulative_sales
2016	9	1008.96	1008.96
2016	10	236361.92	237370.88
2016	12	78.48	237449.36000000002
2017	1	553952.16	791401.52
2017	2	1167632.04	1959033.56
2017	3	1799454.4	3758487.96
2017	4	1671152.12	5429640.08
2017	5	2371675.28	7801315.359999999
2017	6	2045105.52	9846420.879999999
2017	7	2369531.68	12215952.559999999

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

```sql
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 AS "yoy % growth"
FROM a;
```

Result Grid | Filter Rows:

years	yoy % growth
2016	NULL
2017	12112.703757129522
2018	20.000923887447627

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

```sql
WITH a AS (
SELECT
    customers.customer_id,
    MIN(orders.order_purchase_timestamp) AS first_order
FROM customers
JOIN orders
ON customers.customer_id = orders.customer_id
GROUP BY customers.customer_id  ),

b AS (
SELECT
    a.customer_id,
    COUNT(DISTINCT orders.order_purchase_timestamp) AS next_order
FROM a
JOIN orders
ON orders.customer_id = a.customer_id
AND orders.order_purchase_timestamp > first_order
AND orders.order_purchase_timestamp < DATE_ADD(first_order, INTERVAL 6 MONTH)
GROUP BY a.customer_id )

SELECT  100 * (COUNT(DISTINCT a.customer_id) / COUNT(DISTINCT b.customer_id))  AS Retention_Rate_Of_customer
FROM a
LEFT JOIN b
ON a.customer_id = b.customer_id;
```

Result Grid | Filter Rows:

Retention_Rate_Of_customer
NULL

-- 34. Identify the top 3 customers who spent the most money in each year.?

```
SELECT years, customer_id, payment, d_rank
FROM
    (SELECT
        YEAR(orders.order_purchase_timestamp) AS years,
        orders.customer_id,
        SUM(payments.payment_value) AS payment,
        DENSE_RANK() OVER (PARTITION BY YEAR(orders.order_purchase_timestamp)
                           ORDER BY SUM(payments.payment_value) DESC) AS 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;
```

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

years	customer_id	payment	d_rank
2016	a9dc96b027d1252bbac0a9b72d837fc6	5694.2001953125	1
2016	1d34ed25963d5aae4cf3d7f3a4cda173	5602.9599609375	2
2016	4a06381959b6670756de02e07b83815f	4911.1201171875	3
2017	1617b1357756262bfa56ab541c47bc16	54656.3203125	1
2017	c6e2731c5b391845f6800c97401a43a9	27717.240234375	2
2017	3fd6777bbce08a352fddd04e4a7cc8f6	26906.640625	3
2018	ec5b2ba62e574342386871631fafd3fc	29099.51953125	1
2018	f48d464a0baaea338cb25f816991ab1f	27688.83984375	2
2018	e0a2412720e9ea4f26c1ac985f6a7358	19237.759765625	3

Project Learning

MySQL and Python Integration for EDA

- **Establishing Database Connection:** Learned to connect MySQL with Python using libraries such as `mysql-connector` to facilitate seamless data retrieval and manipulation.
- **Data Retrieval:** Developed skills in executing SQL queries to extract relevant datasets from MySQL, enabling efficient data analysis directly within Python.
- **Exploratory Data Analysis (EDA):** Conducted thorough EDA using Python libraries like `pandas`, `numpy`, `seaborn`, and `matplotlib` to analyze data distributions, identify patterns, and uncover insights.
- **Data Visualization:** Created visualizations to represent data trends, distributions, and relationships, enhancing the interpretability of findings.
- **Statistical Analysis:** Applied statistical techniques to analyze data characteristics, performing tasks such as correlation analysis .
- **Report Generation:** Compiled findings into comprehensive reports using Jupyter Notebook, showcasing the analysis process and key insights derived from the data.

Advantages of E-commerce Analysis

This analysis project provides valuable insights into e-commerce operations, enabling businesses to enhance their strategies and drive growth. By calculating average order values and revenue contributions by payment type, companies can optimize their payment strategies to improve the customer experience. Additionally, assessing product performance, seller revenue, and customer retention rates enhances marketing strategies.

Counting orders and total sales per category allows businesses to optimize inventory and tailor promotions effectively. Understanding payment types and customer behavior helps refine payment strategies. Analyzing order trends and average values provides actionable insights for revenue growth. Identifying top products and sellers aids in effective inventory management and strategic partnerships.

Furthermore, examining customer behavior, including retention rates and top spenders, enables businesses to refine marketing strategies and optimize product offerings. By leveraging these insights, companies can drive higher revenue, improve operational efficiency, and foster sustainable growth in the e-commerce sector.

Thank you !