INDEX

Solution
Bonus [EDA, Skewness, Statical Test,Data]

All questions are marked in yellow. All sub questions marked in light yellow. Their corresponding answers are marked in **green**.

1. Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset

1. Data type of columns in a table

SELECT TABLE_NAME,COLUMN_NAME,IS_NULLABLE,DATA_TYPE
FROM information_schema.columns
WHERE table_schema = 'Target';

```
SELECT TABLE_NAME,COLUMN_NAME,IS_NULLABLE,DATA_TYPE
FROM information_schema.columns
WHERE table_schema = 'Target';
```

100% 1:1

Result Grid Filter Rows: Search Export:

TABLE_NAME	COLUMN_NAME	IS_NULLABLE	DATA_TYPE
customers	customer_city	YES	varchar
customers	customer_id	YES	varchar
customers	customer_state	YES	varchar
customers	customer_unique_id	YES	varchar
customers	customer_zip_code_prefix	YES	varchar
geolocation	geolocation_city	YES	varchar
geolocation	geolocation_lat	YES	varchar
geolocation	geolocation_lng	YES	varchar
geolocation	geolocation_state	YES	varchar
geolocation	geolocation_zip_code_prefix	YES	varchar
order_items	freight_value	YES	varchar
order_items	order_id	YES	varchar
order_items	order_item_id	YES	varchar
order_items	price	YES	varchar
order_items	product_id	YES	varchar
order_items	seller_id	YES	varchar
order_items	shipping_limit_date	YES	varchar
order_reviews	order_id	YES	varchar
order_reviews	review_answer_timestamp	YES	varchar
order_reviews	review_comment_title	YES	varchar
order_reviews	review_creation_date	YES	varchar

2. Time period for which the data is given

SELECT MIN(order_purchase_timestamp) as start_date,MAX(order_purchase_timestamp) as end_date FROM orders;

```
use Target;
SELECT MIN(order_purchase_timestamp) as start_date, MAX(order_purchase_timestamp) as end_date
FROM orders;
```

% 1:4

sult Grid Filter Rows: Search Export:

start_date	end_date
2016-09-04 21:15:19	2018-10-17 17:30:18

3. Cities and States of customers ordered during the given period

Select DISTINCT customer_city, customer_state FROM orders o JOIN customer c ON o.customer_id = c.customer_id where o.order_purchase_timestamp BETWEEN ‘2016-09-04 21:15:19’ AND ‘2018-10-17 17:30:18’

```

use Target;
SELECT DISTINCT customer_city, customer_state
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
WHERE o.order_purchase_timestamp BETWEEN '2016-09-04 21:15:19' AND '2018-10-17 17:30:18'

```

00% 1:1

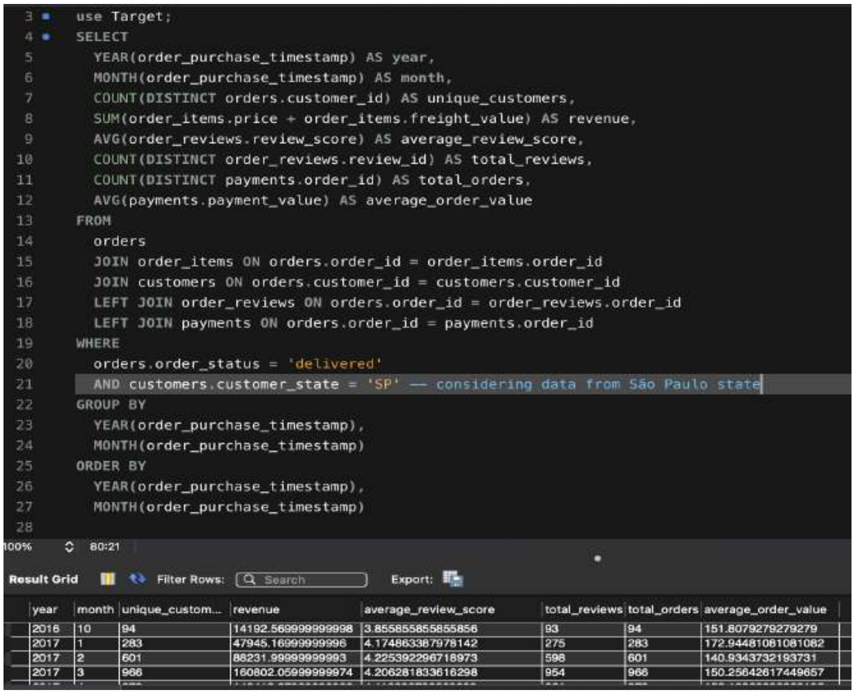
Result Grid Filter Rows: Search Export: Fetch rows:

customer_city	customer_state
sao paulo	SP
sumare	SP
sao jose dos pinhais	PR
porto alegre	RS
contagem	MG
guaruja	SP
serrinha	BA
rio de janeiro	RJ
brasilia	DF
belford roxo	RJ
santa rosa de viterbo	SP
juiz de fora	MG
valinhos	SP
sao jose dos campos	SP
vianopolis	GO
ribeirao pires	SP
gravatai	RS

2. In-depth Exploration:

1. Is there a growing trend on e-commerce in Brazil? How can we describe a complete scenario? Can we see some seasonality with peaks at specific months?

```sql
use Target;
SELECT
  YEAR(order_purchase_timestamp) AS year,
  MONTH(order_purchase_timestamp) AS month,
  COUNT(DISTINCT orders.customer_id) AS unique_customers,
  SUM(order_items.price + order_items.freight_value) AS revenue,
  AVG(order_reviews.review_score) AS average_review_score,
  COUNT(DISTINCT order_reviews.review_id) AS total_reviews,
  COUNT(DISTINCT payments.order_id) AS total_orders,
  AVG(payments.payment_value) AS average_order_value
FROM
  orders
  JOIN order_items ON orders.order_id = order_items.order_id
  JOIN customers ON orders.customer_id = customers.customer_id
  LEFT JOIN order_reviews ON orders.order_id = order_reviews.order_id
  LEFT JOIN payments ON orders.order_id = payments.order_id
WHERE
  orders.order_status = 'delivered'
  AND customers.customer_state = 'SP' -- considering data from São Paulo state
GROUP BY
  YEAR(order_purchase_timestamp),
  MONTH(order_purchase_timestamp)
ORDER BY
  YEAR(order_purchase_timestamp),
  MONTH(order_purchase_timestamp)
```



```sql
use Target;
SELECT
  YEAR(order_purchase_timestamp) AS year,
  MONTH(order_purchase_timestamp) AS month,
  COUNT(DISTINCT orders.customer_id) AS unique_customers,
  SUM(order_items.price + order_items.freight_value) AS revenue,
  AVG(order_reviews.review_score) AS average_review_score,
  COUNT(DISTINCT order_reviews.review_id) AS total_reviews,
  COUNT(DISTINCT payments.order_id) AS total_orders,
  AVG(payments.payment_value) AS average_order_value
FROM
  orders
  JOIN order_items ON orders.order_id = order_items.order_id
  JOIN customers ON orders.customer_id = customers.customer_id
  LEFT JOIN order_reviews ON orders.order_id = order_reviews.order_id
  LEFT JOIN payments ON orders.order_id = payments.order_id
WHERE
  orders.order_status = 'delivered'
  AND customers.customer_state = 'SP' -- considering data from São Paulo state
GROUP BY
  YEAR(order_purchase_timestamp),
  MONTH(order_purchase_timestamp)
ORDER BY
  YEAR(order_purchase_timestamp),
  MONTH(order_purchase_timestamp)
```

100% 80:21

Result Grid Filter Rows: Search Export:

year	month	unique_custom...	revenue	average_review_score	total_reviews	total_orders	average_order_value
2016	10	94	14192.569999999998	3.855855855855856	93	94	151.8079279279279
2017	1	283	47945.16999999996	4.174863387978142	275	283	172.94481081081082
2017	2	601	88231.99999999993	4.225392296718973	598	601	140.9343732193731
2017	3	966	160802.05999999974	4.206281833616298	954	966	150.25642617449657

2. What time do Brazilian customers tend to buy (Dawn, Morning, Afternoon or Night)?

use Target;
SELECT
 CASE
  WHEN HOUR(o.order_purchase_timestamp) >= 0 AND HOUR(o.order_purchase_timestamp) < 6 THEN 'Dawn'
  WHEN HOUR(o.order_purchase_timestamp) >= 6 AND HOUR(o.order_purchase_timestamp) < 12 THEN 'Morning'
  WHEN HOUR(o.order_purchase_timestamp) >= 12 AND HOUR(o.order_purchase_timestamp) < 18 THEN 'Afternoon'
  ELSE 'Night'
 END AS time_of_day,
 COUNT(*) AS num_orders
FROM
 orders o
 JOIN customers c ON o.customer_id = c.customer_id
GROUP BY
 time_of_day
ORDER BY
 num_orders DESC;

```sql
use Target;
SELECT
  CASE
    WHEN HOUR(o.order_purchase_timestamp) >= 0 AND HOUR(o.order_purchase_timestamp) < 6 THEN 'Dawn'
    WHEN HOUR(o.order_purchase_timestamp) >= 6 AND HOUR(o.order_purchase_timestamp) < 12 THEN 'Morning'
    WHEN HOUR(o.order_purchase_timestamp) >= 12 AND HOUR(o.order_purchase_timestamp) < 18 THEN 'Afternoon'
    ELSE 'Night'
  END AS time_of_day,
  COUNT(*) AS num_orders
FROM
  orders o
  JOIN customers c ON o.customer_id = c.customer_id
GROUP BY
  time_of_day
ORDER BY
  num_orders DESC;
```

1:17

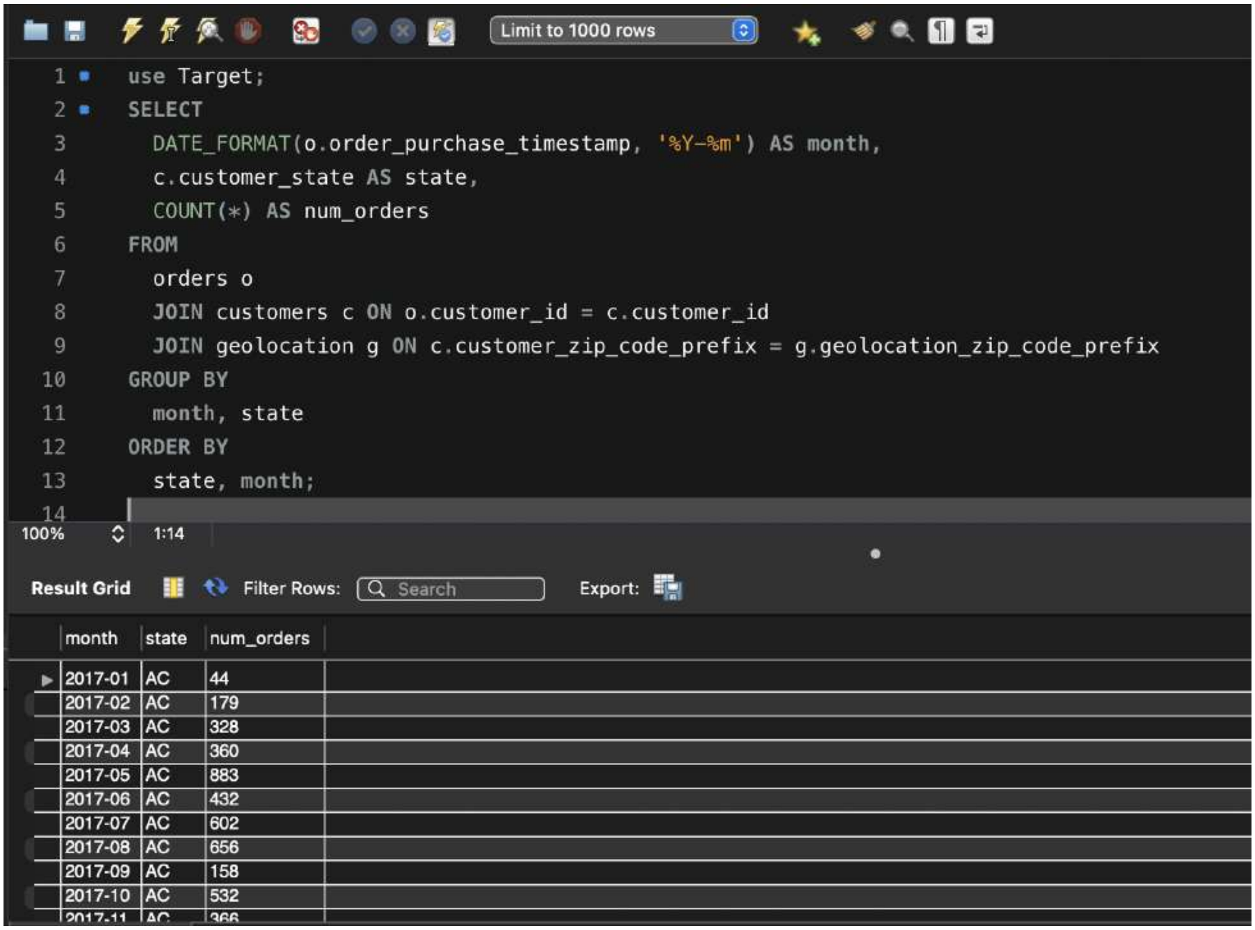
sult Grid Filter Rows: Search Export:

time_of_day	num_orders
Afternoon	38361
Night	34100
Morning	22240
Dawn	4740

3.Evolution of E-commerce orders in the Brazil region:

1. Get month on month orders by states

```sql
use Target;
SELECT
 DATE_FORMAT(o.order_purchase_timestamp, '%Y-%m') AS month,
 c.customer_state AS state,
 COUNT(*) AS num_orders
FROM
 orders o
 JOIN customers c ON o.customer_id = c.customer_id
 JOIN geolocation g ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
GROUP BY
 month, state
ORDER BY
 state, month;
```

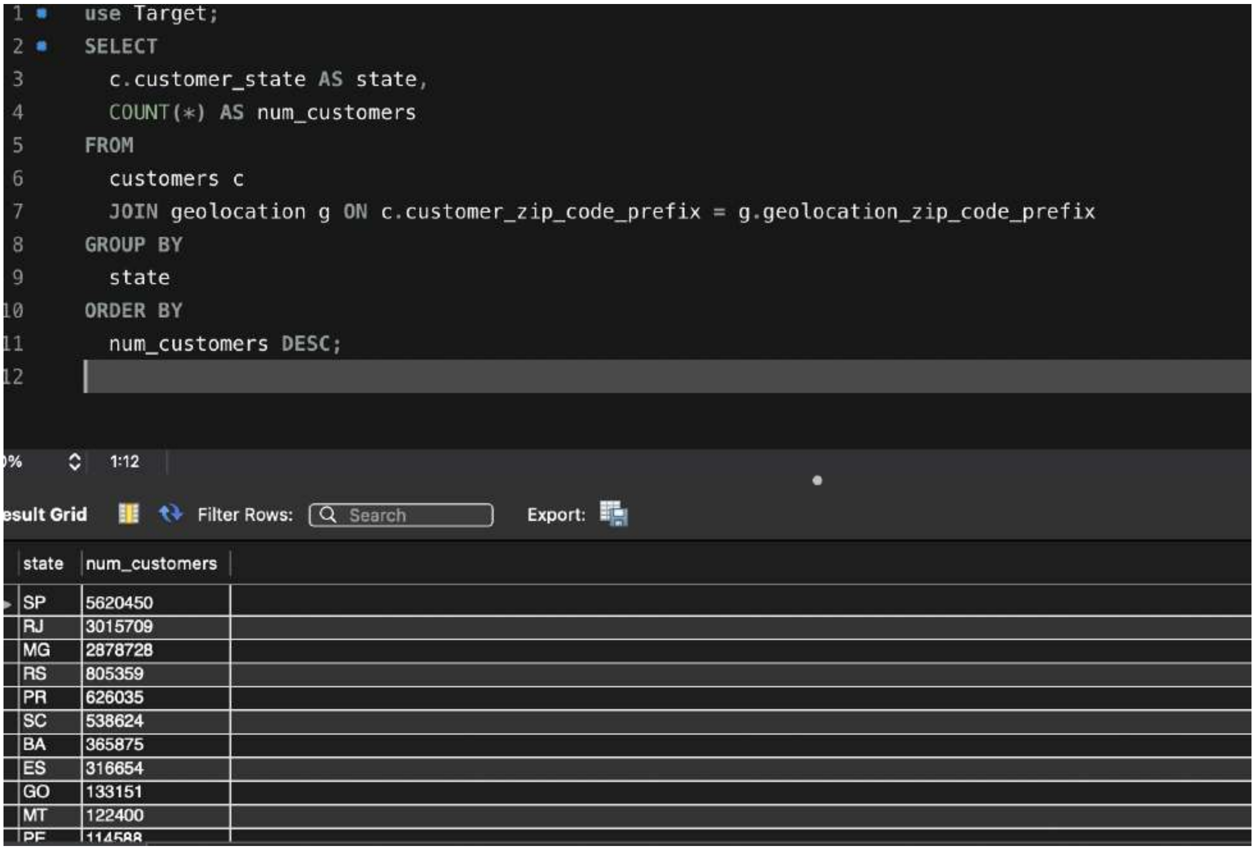


```
use Target;
SELECT
  DATE_FORMAT(o.order_purchase_timestamp, '%Y-%m') AS month,
  c.customer_state AS state,
  COUNT(*) AS num_orders
FROM
  orders o
  JOIN customers c ON o.customer_id = c.customer_id
  JOIN geolocation g ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
GROUP BY
  month, state
ORDER BY
  state, month;
```


month	state	num_orders
2017-01	AC	44
2017-02	AC	179
2017-03	AC	328
2017-04	AC	360
2017-05	AC	883
2017-06	AC	432
2017-07	AC	602
2017-08	AC	656
2017-09	AC	158
2017-10	AC	532
2017-11	AC	366

2. Distribution of customers across the states in Brazil

```
use Target;
SELECT
  c.customer_state AS state,
  COUNT(*) AS num_customers
FROM
  customers c
  JOIN geolocation g ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
GROUP BY
  state
ORDER BY
  num_customers DESC;
```

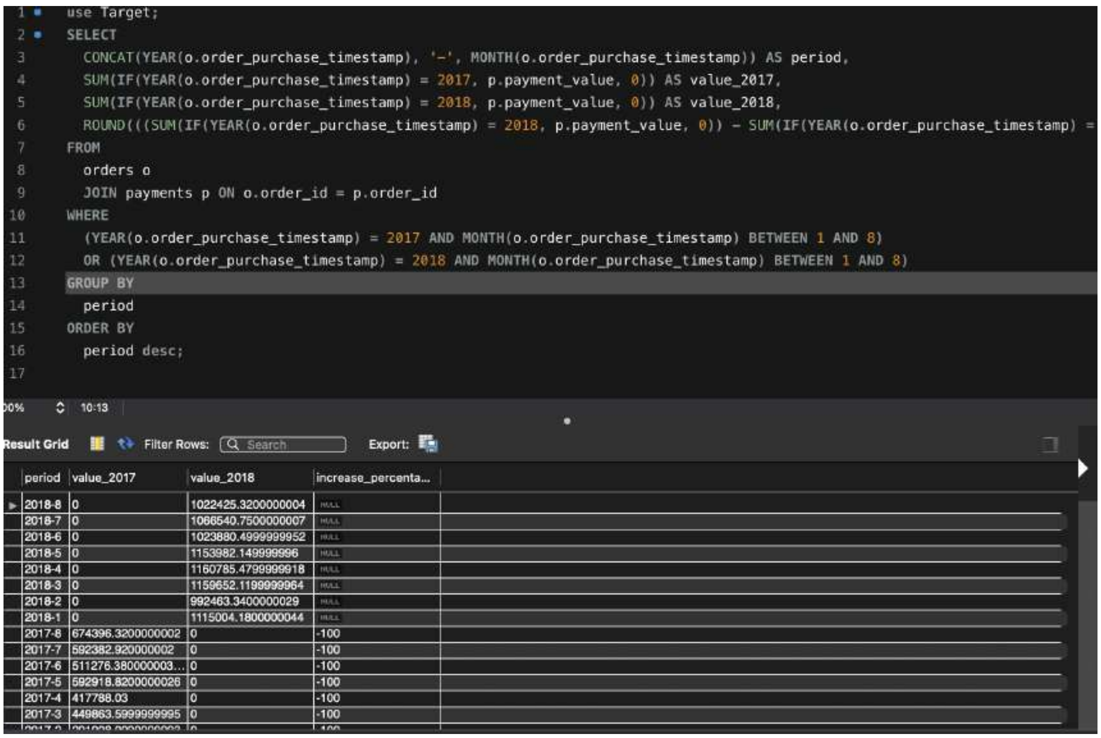


4. Impact on Economy: Analyze the money movement by e-commerce by looking at order prices, freight and others.

1. Get % increase in cost of orders from 2017 to 2018 (include months between Jan to Aug only) - You can use “payment_value” column in payments table

```
use Target;
SELECT
 CONCAT(YEAR(o.order_purchase_timestamp), '-', MONTH(o.order_purchase_timestamp)) AS period,
 SUM(IF(YEAR(o.order_purchase_timestamp) = 2017, p.payment_value, 0)) AS value_2017,
 SUM(IF(YEAR(o.order_purchase_timestamp) = 2018, p.payment_value, 0)) AS value_2018,
 ROUND(((SUM(IF(YEAR(o.order_purchase_timestamp) = 2018, p.payment_value, 0)) -
SUM(IF(YEAR(o.order_purchase_timestamp) = 2017, p.payment_value, 0))) /
SUM(IF(YEAR(o.order_purchase_timestamp) = 2017, p.payment_value, 0))) * 100, 2) AS
increase_percentage
FROM
 orders o
```

JOIN payments p ON o.order_id = p.order_id
WHERE
(YEAR(o.order_purchase_timestamp) = 2017 AND MONTH(o.order_purchase_timestamp) BETWEEN 1 AND 8)
OR (YEAR(o.order_purchase_timestamp) = 2018 AND MONTH(o.order_purchase_timestamp) BETWEEN 1 AND 8)
GROUP BY
period
ORDER BY
period desc;



2.Mean & Sum of price and freight value by customer state

```
use Target;
SELECT
 customers.customer_state,
 AVG(order_items.price) AS avg_price,
 SUM(order_items.price) AS total_price,
 AVG(order_items.freight_value) AS avg_freight,
 SUM(order_items.freight_value) AS total_freight
FROM
 customers
 JOIN orders ON customers.customer_id = orders.customer_id
 JOIN order_items ON orders.order_id = order_items.order_id
GROUP BY
 customers.customer_state;
```

```sql
use Target;
SELECT
  customers.customer_state,
  AVG(order_items.price) AS avg_price,
  SUM(order_items.price) AS total_price,
  AVG(order_items.freight_value) AS avg_freight,
  SUM(order_items.freight_value) AS total_freight
FROM
  customers
  JOIN orders ON customers.customer_id = orders.customer_id
  JOIN order_items ON orders.order_id = order_items.order_id
GROUP BY
  customers.customer_state;
```

100% 1:14

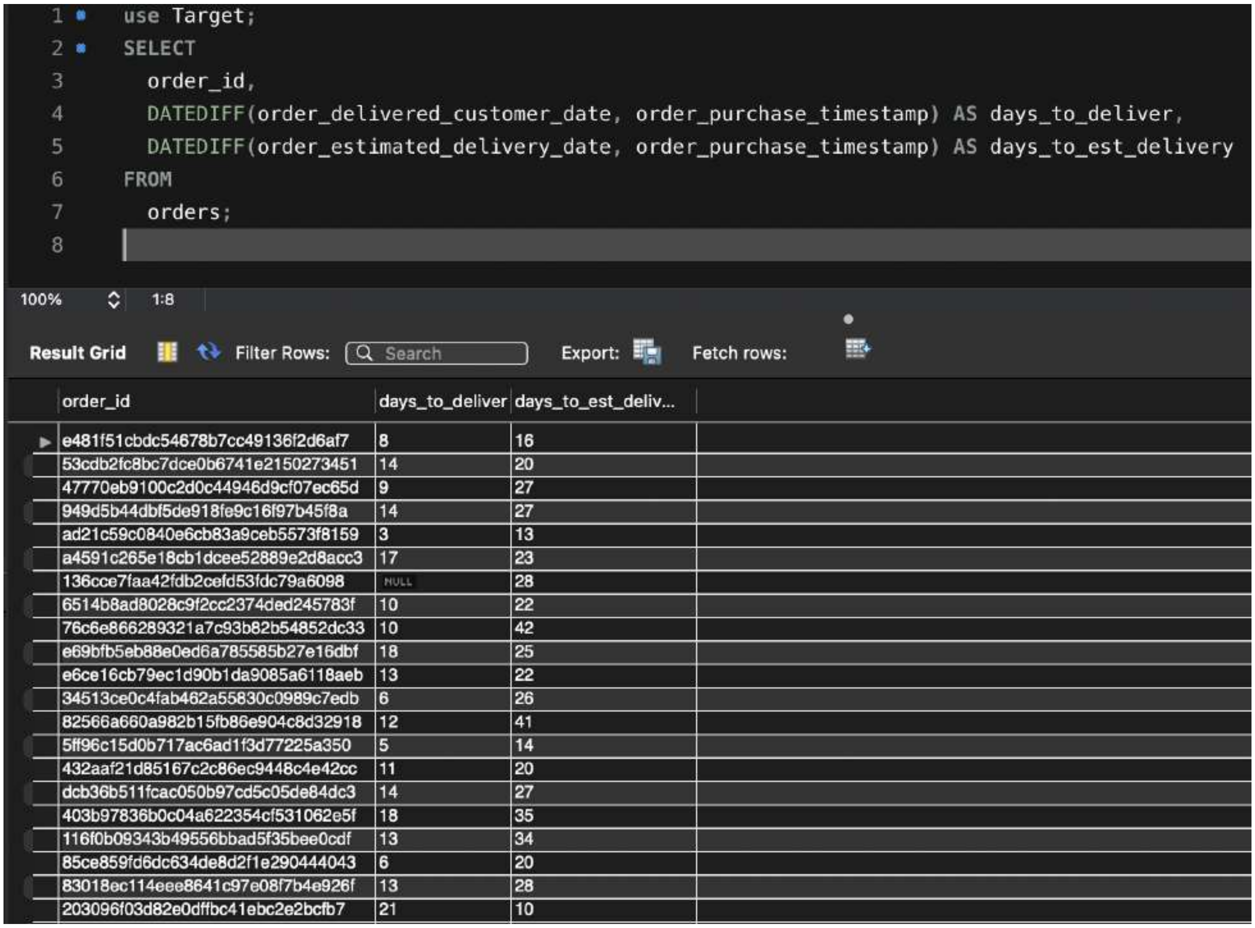
Result Grid Filter Rows: Search Export:

customer_state	avg_price	total_price	avg_freight	total_freight
SP	109.65362915976213	5202955.050001553	15.14727539041888	718723.0699999854
RS	120.33745308741331	750304.020000022	21.735804330393318	135522.7400000235
SC	124.65357758620901	520553.3400000088	21.470368773946397	89660.26000000015
BA	134.60120821268953	511349.9900000075	26.363958936562007	100156.67999999906
MS	142.6283760683758	116812.63999999977	23.374884004884	19144.029999999995
RJ	125.11781809450689	1824092.669999816	20.960923931682412	305589.3099999979
PI	160.35808118081195	86914.08000000007	39.14797047970483	21218.20000000002
MG	120.74857414882179	1585308.0299998813	20.630166806306867	270853.4600000029
ES	121.91370124113331	275037.30999999674	22.05877659574459	49764.599999999795
RO	165.97352517985644	46140.64000000009	41.06971223021583	11417.38
PR	119.00413937282565	683083.7600000192	20.53165156794443	117851.68000000103
DF	125.77054862842789	302603.9399999975	21.041354945968347	50625.49999999985
MT	148.29718483412253	156453.52999999927	28.166284360189632	29715.430000000062
GO	126.2717316759522	294591.94999999646	22.766815259322733	53114.97999999994

5. Analysis on sales, freight and delivery time

1.Calculate days between purchasing, delivering and estimated delivery

```sql
use Target;
SELECT
 order_id,
 DATEDIFF(order_delivered_customer_date, order_purchase_timestamp) AS days_to_deliver,
 DATEDIFF(order_estimated_delivery_date, order_purchase_timestamp) AS days_to_est_delivery
FROM
 orders;
```



```sql
use Target;
SELECT
  order_id,
  DATEDIFF(order_delivered_customer_date, order_purchase_timestamp) AS days_to_deliver,
  DATEDIFF(order_estimated_delivery_date, order_purchase_timestamp) AS days_to_est_delivery
FROM
  orders;
```

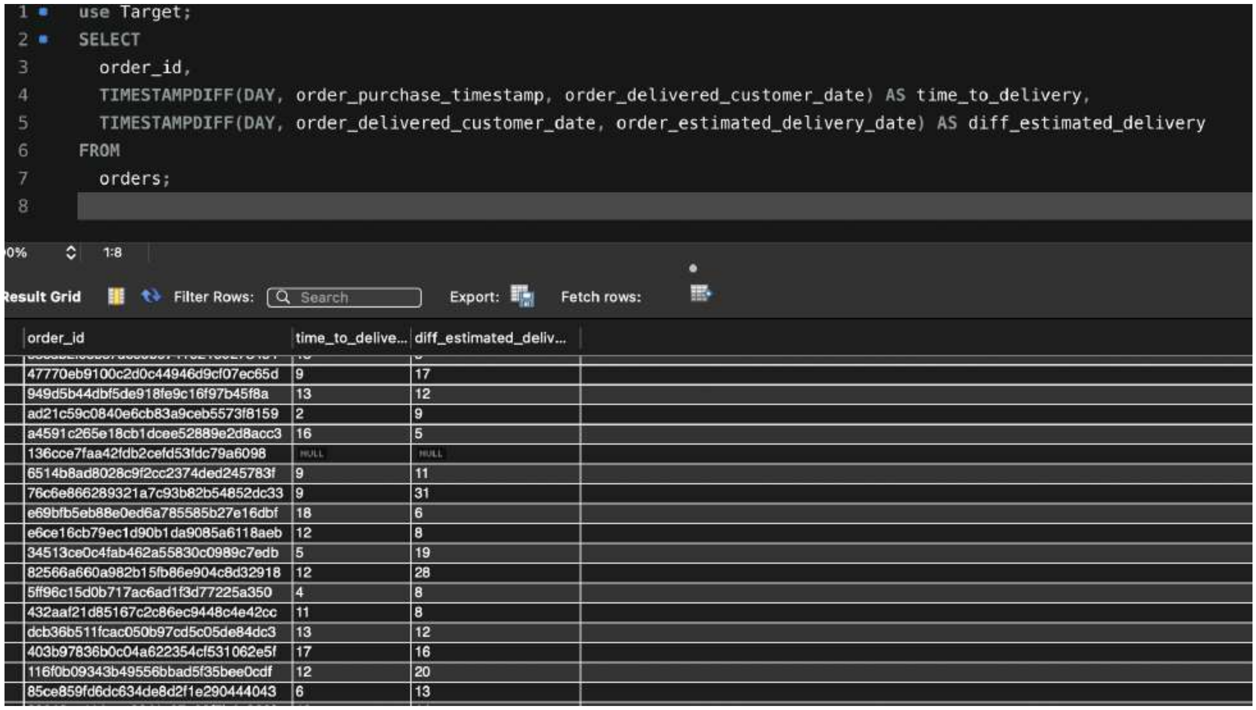

order_id	days_to_deliver	days_to_est_deliv...
e481f51cbdc54678b7cc49136f2d6af7	8	16
53cdb2fc8bc7dce0b6741e2150273451	14	20
47770eb9100c2d0c44946d9cf07ec65d	9	27
949d5b44dbf5de918fe9c16f97b45f8a	14	27
ad21c59c0840e6cb83a9ceb5573f8159	3	13
a4591c265e18cb1dcee52889e2d8acc3	17	23
136cce7faa42fdb2cefd53fdc79a6098	NULL	28
6514b8ad8028c9f2cc2374ded245783f	10	22
76c6e866289321a7c93b82b54852dc33	10	42
e69bfb5eb88e0ed6a785585b27e16dbf	18	25
e6ce16cb79ec1d90b1da9085a6118aeb	13	22
34513ce0c4fab462a55830c0989c7edb	6	26
82566a660a982b15fb86e904c8d32918	12	41
5ff96c15d0b717ac6ad1f3d77225a350	5	14
432aaf21d85167c2c86ec9448c4e42cc	11	20
dcb36b511fcac050b97cd5c05de84dc3	14	27
403b97836b0c04a622354cf531062e5f	18	35
116f0b09343b49556bbad5f35bee0cdf	13	34
85ce859fd6dc634de8d2f1e290444043	6	20
83018ec114eee8641c97e08f7b4e926f	13	28
203096f03d82e0dffbc41ebc2e2bcfb7	21	10

1.Find time_to_delivery & diff_estimated_delivery. Formula for the same given below:

- time_to_delivery = order_delivered_customer_date-order_purchase_timestamp
- diff_estimated_delivery = order_estimated_delivery_date-order_delivered_customer_date

use Target;
SELECT
order_id,
TIMESTAMPDIFF(DAY, order_purchase_timestamp, order_delivered_customer_date) AS time_to_delivery,
TIMESTAMPDIFF(DAY, order_delivered_customer_date, order_estimated_delivery_date) AS diff_estimated_delivery
FROM

orders;



```
use Target;
SELECT
  order_id,
  TIMESTAMPDIFF(DAY, order_purchase_timestamp, order_delivered_customer_date) AS time_to_delivery,
  TIMESTAMPDIFF(DAY, order_delivered_customer_date, order_estimated_delivery_date) AS diff_estimated_delivery
FROM
  orders;
```


order_id	time_to_delive...	diff_estimated_deliv...
47770eb9100c2d0c44946d9cf07ec65d	9	17
949d5b44dbf5de918fe9c16f97b45f8a	13	12
ad21c59c0840e6cb83a9ceb5573f8159	2	9
a4591c265e18cb1dcee52889e2d8acc3	16	5
136cce7faa42fdb2cefd53fdc79a6098	NULL	NULL
6514b8ad8028c9f2cc2374ded245783f	9	11
76c6e866289321a7c93b82b54852dc33	9	31
e69bfb5eb88e0ed6a785585b27e16dbf	18	6
e6ce16cb79ec1d90b1da9085a6118aeb	12	8
34513ce0c4fab462a55830c0989c7edb	5	19
82566a660a982b15fb86e904c8d32918	12	28
5ff96c15d0b717ac6ad1f3d77225a350	4	8
432aaf21d85167c2c86ec9448c4e42cc	11	8
dcb36b511fcac050b97cd5c05de84dc3	13	12
403b97836b0c04a622354cf531062e5f	17	16
116f0b09343b49556bbad5f35bee0cdf	12	20
85ce859fd6dc634de8d2f1e290444043	6	13

2. Group data by state, take mean of freight_value, time_to_delivery, diff_estimated_delivery

```
SELECT
  customers.customer_state AS state,
  AVG(order_items.freight_value) AS mean_freight_value,
  AVG(TIMESTAMPDIFF(DAY, orders.order_purchase_timestamp,
orders.order_delivered_customer_date)) AS mean_time_to_delivery,
  AVG(TIMESTAMPDIFF(DAY, orders.order_delivered_customer_date,
orders.order_estimated_delivery_date)) AS mean_diff_estimated_delivery
FROM
  orders
  JOIN order_items ON orders.order_id = order_items.order_id
  JOIN customers ON orders.customer_id = customers.customer_id
GROUP BY
```

state;

```
SELECT
    customers.customer_state AS state,
    AVG(order_items.freight_value) AS mean_freight_value,
    AVG(TIMESTAMPDIFF(DAY, orders.order_purchase_timestamp, orders.order_delivered_customer_date)) AS mean_time_to_delivery,
    AVG(TIMESTAMPDIFF(DAY, orders.order_delivered_customer_date, orders.order_estimated_delivery_date)) AS mean_diff_estimated_delivery
FROM
    orders
    JOIN order_items ON orders.order_id = order_items.order_id
    JOIN customers ON orders.customer_id = customers.customer_id
GROUP BY
    state
```

29:9

sult Grid Filter Rows: Search Export:

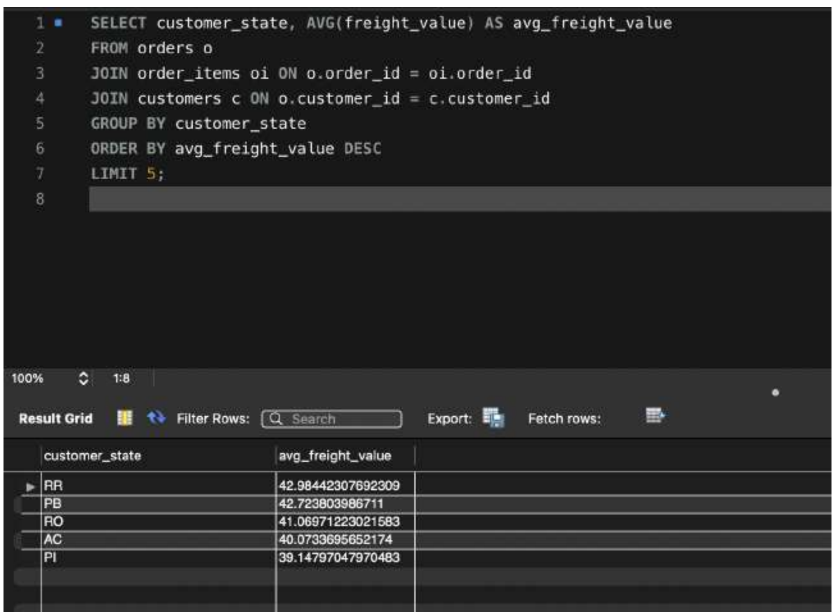
state	mean_freight_value	mean_time_to_deliv...	mean_diff_estimated_deliv...
SP	15.147275390418883	8.2596	10.2656
RS	21.735804330393314	14.7083	13.2030
SC	21.470368773946394	14.5210	10.6689
MS	23.374884004884	15.1073	10.3379
MG	20.630166806306867	11.5155	12.3972
RJ	20.960923931682416	14.6894	11.1445
RO	41.06971223021583	19.2821	19.0806
PR	20.531651567944433	11.4808	12.5339
MT	28.166284360189625	17.5082	13.6393
GO	22.766815259322733	14.9482	11.3729
DF	21.04135494596834	12.5015	11.2747
RN	35.6523629489603	18.8733	13.0557
BA	26.363958936562003	18.7746	10.1195
PE	32.91786267995564	17.7921	12.5521

2. Sort the data to get the following:
3. Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5

SELECT customer_state, AVG(freight_value) AS avg_freight_value
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY customer_state
ORDER BY avg_freight_value ASC
LIMIT 5;

```sql
SELECT customer_state, AVG(freight_value) AS avg_freight_value
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY customer_state
ORDER BY avg_freight_value ASC
LIMIT 5;
```

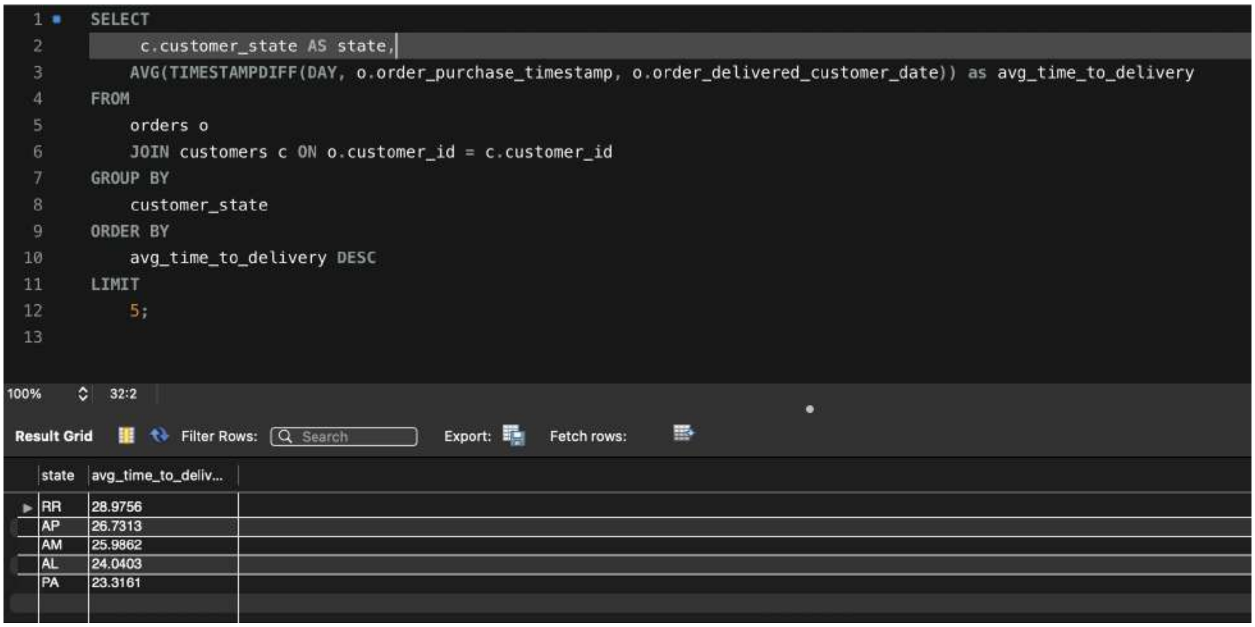
SELECT customer_state, AVG(freight_value) AS avg_freight_value
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY customer_state
ORDER BY avg_freight_value DESC
LIMIT 5;



4. Top 5 states with highest/lowest average time to delivery

SELECT
c.customer_state AS state,
AVG(TIMESTAMPDIFF(DAY, o.order_purchase_timestamp, o.order_delivered_customer_date)) as avg_time_to_delivery
FROM
orders o
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY

```
  customer_state
ORDER BY
  avg_time_to_delivery DESC
LIMIT
  5;
```




```
SELECT
   c.customer_state AS state,
  AVG(TIMESTAMPDIFF(DAY, o.order_purchase_timestamp, o.order_delivered_customer_date)) as
avg_time_to_delivery
FROM
  orders o
  JOIN customers c ON o.customer_id = c.customer_id
GROUP BY
  customer_state
ORDER BY
  avg_time_to_delivery ASC
LIMIT 5;
```

```sql
SELECT
     c.customer_state AS state,
    AVG(TIMESTAMPDIFF(DAY, o.order_purchase_timestamp, o.order_delivered_customer_date)) as avg_time_to_delivery
FROM
    orders o
    JOIN customers c ON o.customer_id = c.customer_id
GROUP BY
    customer_state
ORDER BY
    avg_time_to_delivery ASC
LIMIT
    5;
```

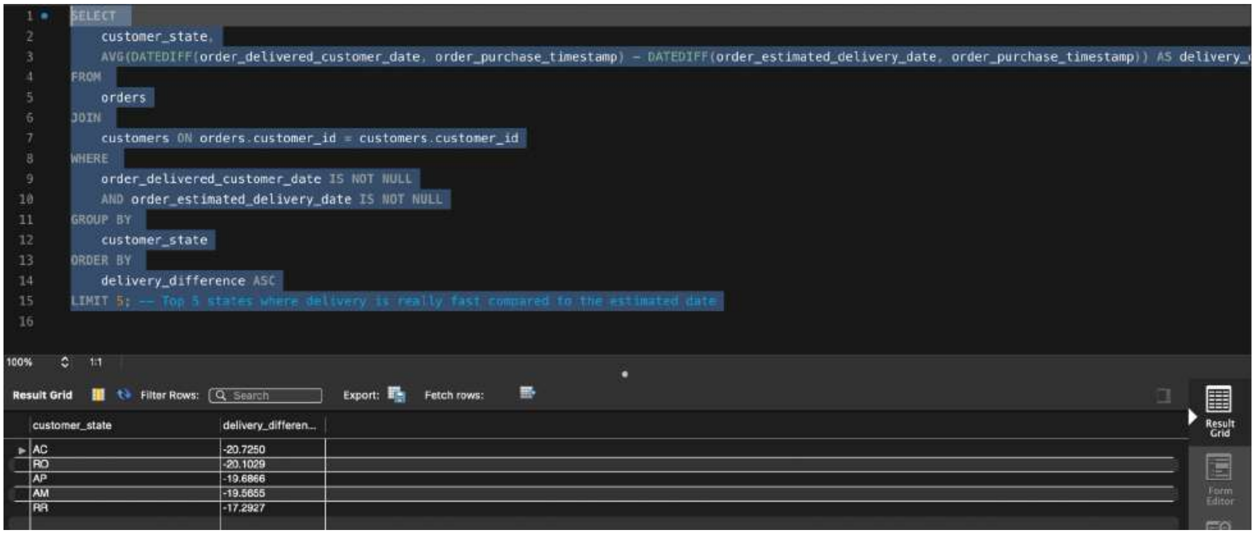
100% 58:3

Result Grid Filter Rows: Search Export: Fetch rows:

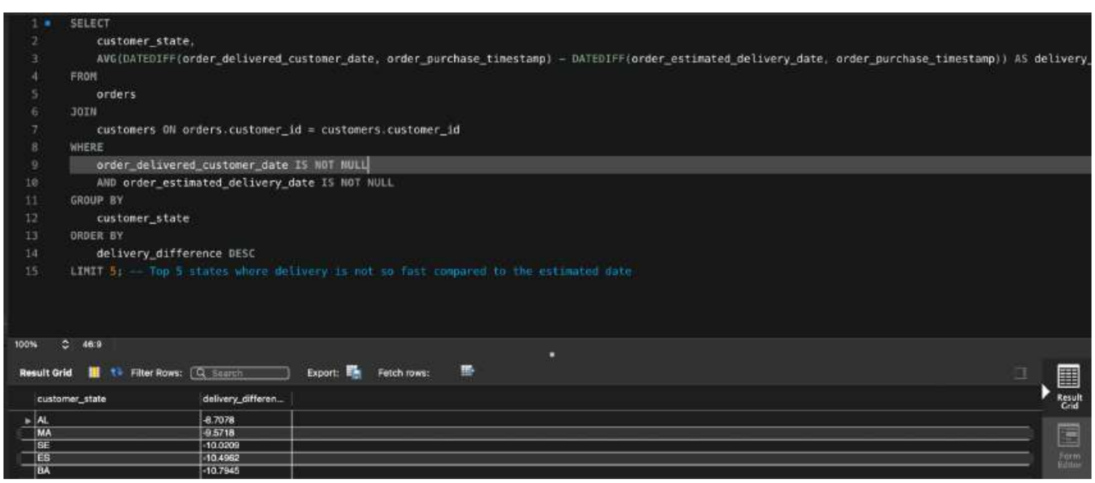
state	avg_time_to_deliv...
SP	8.2981
PR	11.5267
MG	11.5438
DF	12.5091
SC	14.4796

4. Top 5 states where delivery is really fast/ not so fast compared to estimated date

```sql
SELECT
    customer_state,
    AVG(DATEDIFF(order_delivered_customer_date, order_purchase_timestamp) -
DATEDIFF(order_estimated_delivery_date, order_purchase_timestamp)) AS delivery_difference
FROM
    orders
JOIN
    customers ON orders.customer_id = customers.customer_id
WHERE
    order_delivered_customer_date IS NOT NULL
    AND order_estimated_delivery_date IS NOT NULL
GROUP BY
    customer_state
ORDER BY
    delivery_difference ASC
LIMIT 5; -- Top 5 states where delivery is really fast compared to the estimated date
```



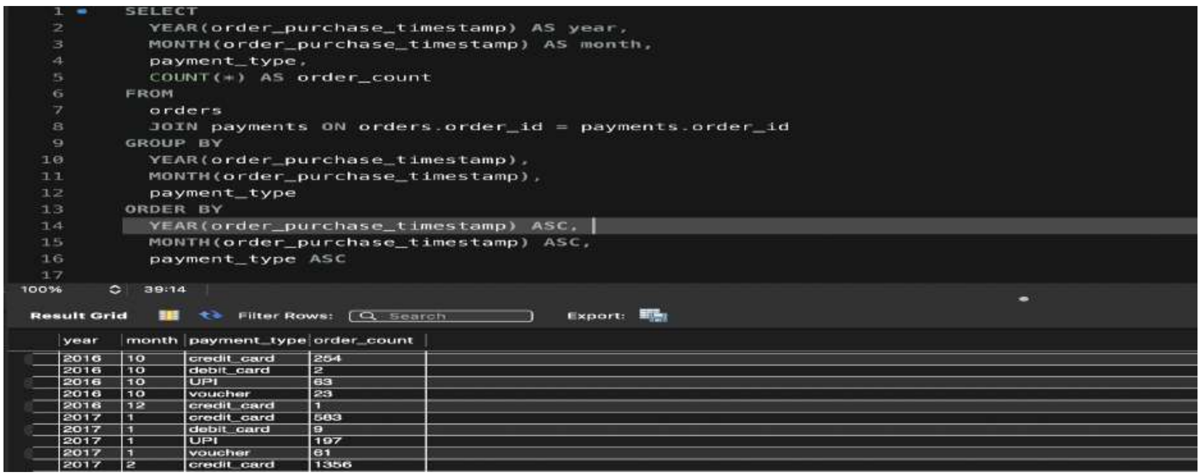

```
SELECT
    customer_state,
    AVG(DATEDIFF(order_delivered_customer_date, order_purchase_timestamp) -
DATEDIFF(order_estimated_delivery_date, order_purchase_timestamp)) AS delivery_difference
FROM
    orders
JOIN
    customers ON orders.customer_id = customers.customer_id
WHERE
    order_delivered_customer_date IS NOT NULL
    AND order_estimated_delivery_date IS NOT NULL
GROUP BY
    customer_state
ORDER BY
    delivery_difference DESC
LIMIT 5; -- Top 5 states where delivery is not so fast compared to the estimated date
```



6. Payment type analysis:

1.Month over Month count of orders for different payment types

SELECT
YEAR(order_purchase_timestamp) AS year,
MONTH(order_purchase_timestamp) AS month,
payment_type,
COUNT(*) AS order_count
FROM
orders
JOIN payments ON orders.order_id = payments.order_id
GROUP BY
YEAR(order_purchase_timestamp),
MONTH(order_purchase_timestamp),
payment_type
ORDER BY
YEAR(order_purchase_timestamp) ASC,
MONTH(order_purchase_timestamp) ASC,
payment_type ASC



```
SELECT
  YEAR(order_purchase_timestamp) AS year,
  MONTH(order_purchase_timestamp) AS month,
  payment_type,
  COUNT(*) AS order_count
FROM
  orders
  JOIN payments ON orders.order_id = payments.order_id
GROUP BY
  YEAR(order_purchase_timestamp),
  MONTH(order_purchase_timestamp),
  payment_type
ORDER BY
  YEAR(order_purchase_timestamp) ASC,
  MONTH(order_purchase_timestamp) ASC,
  payment_type ASC
```

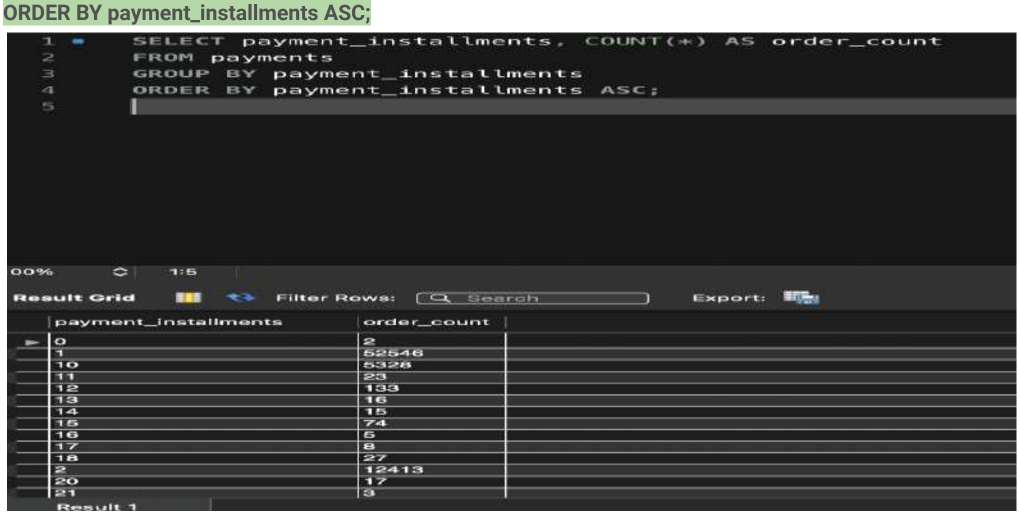
100% 39:14

Result Grid Filter Rows: Search Export:

year	month	payment_type	order_count
2016	10	credit_card	254
2016	10	debit_card	2
2016	10	UPI	63
2016	10	voucher	23
2016	12	credit_card	1
2017	1	credit_card	583
2017	1	debit_card	9
2017	1	UPI	197
2017	1	voucher	61
2017	2	credit_card	1356

2. Count of orders based on the no. of payment installments

SELECT payment_installments, COUNT(*) AS order_count
FROM payments
GROUP BY payment_installments
ORDER BY payment_installments ASC;



```
SELECT payment_installments, COUNT(*) AS order_count
FROM payments
GROUP BY payment_installments
ORDER BY payment_installments ASC;
```

00% 1:5

Result Grid Filter Rows: Search Export:

payment_installments	order_count
0	2
1	52546
10	5328
11	23
12	133
13	16
14	15
15	74
16	5
17	8
18	27
2	12413
20	17
21	3

Result 1

BONUS

1- EDA

order table:
order_id: A unique identifier for each order.
customer_id: A unique identifier for each customer who placed the order.
order_status: The status of the order, such as "delivered" or "shipped."
order_purchase_timestamp: The timestamp when the order was placed.
order_approved_at: The timestamp when the order was approved.
order_delivered_carrier_date: The timestamp when the order was picked up by the carrier.
order_delivered_customer_date: The timestamp when the order was delivered to the customer.
order_estimated_delivery_date: The estimated delivery date for the order.
order_items table:

order_id: A unique identifier for the order that the item belongs to.
order_item_id: A unique identifier for the item within the order.
product_id: A unique identifier for the product that was ordered.
seller_id: A unique identifier for the seller that fulfilled the order.
shipping_limit_date: The timestamp by which the order should be shipped.
price: The price of the item.
freight_value: The cost of shipping the item.

geolocation table:
geolocation_zip_code_prefix: The first 5 digits of the zip code for a location.
geolocation_lat: The latitude of the location.
geolocation_lng: The longitude of the location.
geolocation_city: The city of the location.
geolocation_state: The state of the location.

customers table:
customer_id: A unique identifier for each customer.
customer_unique_id: A unique identifier for each customer that does not change across orders.
customer_zip_code_prefix: The first 5 digits of the zip code for the customer's location.
customer_city: The city of the customer's location.
customer_state: The state of the customer's location.

order_reviews table:
review_id: A unique identifier for each review.
order_id: A unique identifier for the order that the review pertains to.
review_score: A score from 1-5 given by the customer to rate their satisfaction with the order.
review_comment_title: The title of the review.
review_creation_date: The timestamp when the review was created.
review_answer_timestamp: The timestamp when the review was answered.

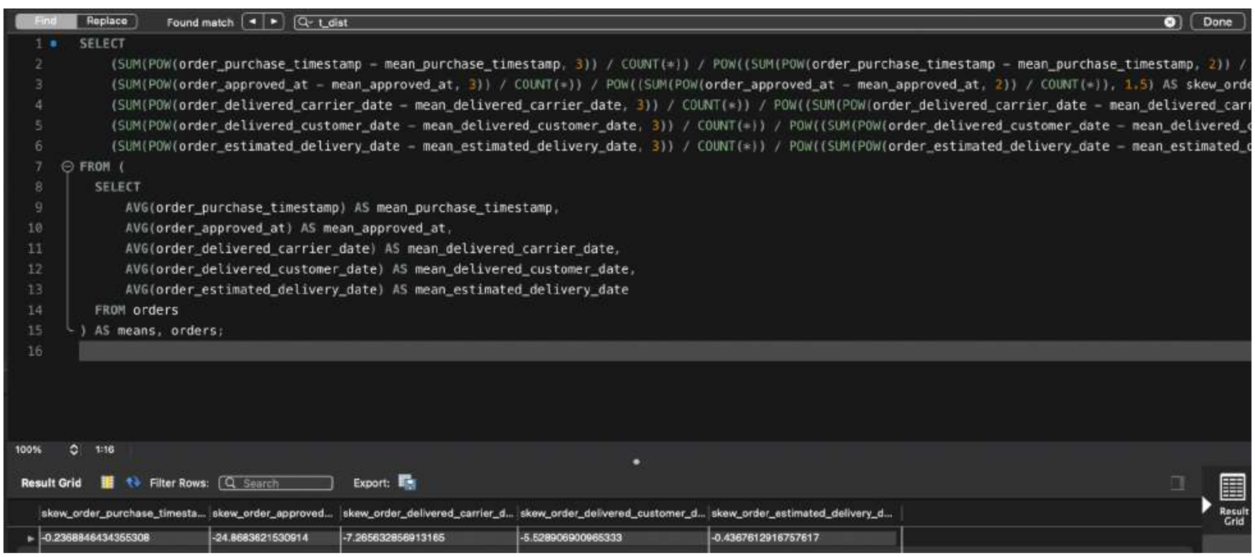
payments table:
order_id: A unique identifier for the order that the payment is associated with.
payment_sequential: A sequential number assigned to each payment associated with an order.
payment_type: The type of payment used to make the payment, such as credit card or debit card.
payment_installments: The number of installments the payment was split into.
payment_value: The total value of the payment.

sellers table:
seller_id: A unique identifier for each seller.
seller_zip_code_prefix: The first 5 digits of the zip code for the seller's location.
seller_city: The city of the seller's location.
seller_state: The state of the seller's location.

products table:
product_id: A unique identifier for each product.
product_category: The category that the product belongs to.
product_name_length: The length of the name of the product.
product_description_length: The length of the description of the product.
product_photos_qty: The number of photos associated with the product.
product_weight_g: The weight of the

2- Skewness

```
SELECT
    (SUM(POW(order_purchase_timestamp - mean_purchase_timestamp, 3)) / COUNT(*)) / POW((SUM(POW(order_purchase_timestamp - mean_purchase_timestamp, 2)) / COUNT(*)), 1.5) AS skew_order_purchase_timestamp,
    (SUM(POW(order_approved_at - mean_approved_at, 3)) / COUNT(*)) / POW((SUM(POW(order_approved_at - mean_approved_at, 2)) / COUNT(*)), 1.5) AS skew_order_approved_at,
    (SUM(POW(order_delivered_carrier_date - mean_delivered_carrier_date, 3)) / COUNT(*)) / POW((SUM(POW(order_delivered_carrier_date - mean_delivered_carrier_date, 2)) / COUNT(*)), 1.5) AS skew_order_delivered_carrier_date,
    (SUM(POW(order_delivered_customer_date - mean_delivered_customer_date, 3)) / COUNT(*)) / POW((SUM(POW(order_delivered_customer_date - mean_delivered_customer_date, 2)) / COUNT(*)), 1.5) AS skew_order_delivered_customer_date,
    (SUM(POW(order_estimated_delivery_date - mean_estimated_delivery_date, 3)) / COUNT(*)) / POW((SUM(POW(order_estimated_delivery_date - mean_estimated_delivery_date, 2)) / COUNT(*)), 1.5) AS skew_order_estimated_delivery_date
FROM (
  SELECT
     AVG(order_purchase_timestamp) AS mean_purchase_timestamp,
     AVG(order_approved_at) AS mean_approved_at,
     AVG(order_delivered_carrier_date) AS mean_delivered_carrier_date,
     AVG(order_delivered_customer_date) AS mean_delivered_customer_date,
     AVG(order_estimated_delivery_date) AS mean_estimated_delivery_date
  FROM orders
) AS means, orders;
```



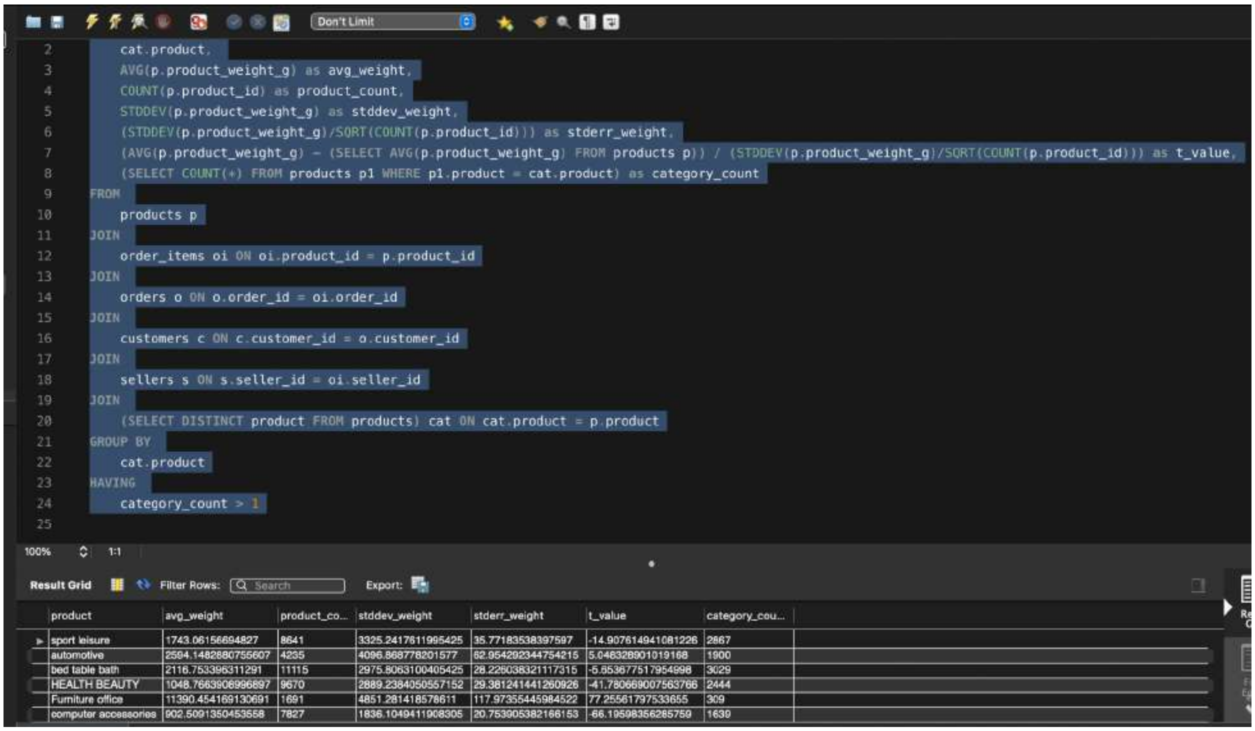
skew_order_purchase_timestamp: This value (-0.2369) represents the skewness of the order_purchase_timestamp column in the orders table. Skewness is a measure of the asymmetry of a distribution, and this value indicates that the distribution of purchase timestamps is slightly negatively skewed, i.e., it is skewed to the left.
skew_order_approved_at: This value (-24.8684) represents the skewness of the order_approved_at column in the orders table. This value is very large and negative, which indicates that the distribution of approval timestamps is heavily skewed to the left.
skew_order_delivered_carrier_date: This value (-7.2656) represents the skewness of the order_delivered_carrier_date column in the orders table. This value indicates that the distribution of carrier delivery timestamps is also skewed to the left, although not as heavily as the approval timestamps.
skew_order_delivered_customer_date: This value (-5.5289) represents the skewness of the order_delivered_customer_date column in the orders table. This value indicates that the distribution of customer delivery timestamps is also skewed to the left, although not as heavily as the approval timestamps or the carrier delivery timestamps.
skew_order_estimated_delivery_date: This value (-0.4368) represents the skewness of the order_estimated_delivery_date column in the orders table. This value indicates that the distribution of estimated delivery dates is slightly negatively skewed, i.e., it is skewed to the left.

3- Statistical test

H0: There is no significant difference in the average weight of products in different categories.
H1: There is a significant difference in the average weight of products in different categories.

```
SELECT
  cat.product,
  AVG(p.product_weight_g) as avg_weight,
  COUNT(p.product_id) as product_count,
  STDDEV(p.product_weight_g) as stddev_weight,
  (STDDEV(p.product_weight_g)/SQRT(COUNT(p.product_id))) as stderr_weight,
  (AVG(p.product_weight_g) - (SELECT AVG(p.product_weight_g) FROM products p)) /
(STDDEV(p.product_weight_g)/SQRT(COUNT(p.product_id))) as t_value,
  (SELECT COUNT(*) FROM products p1 WHERE p1.product = cat.product) as category_count
FROM
  products p
JOIN
```

```
    order_items oi ON oi.product_id = p.product_id
JOIN
    orders o ON o.order_id = oi.order_id
JOIN
    customers c ON c.customer_id = o.customer_id
JOIN
    sellers s ON s.seller_id = oi.seller_id
JOIN
    (SELECT DISTINCT product FROM products) cat ON cat.product = p.product
GROUP BY
    cat.product
HAVING
    category_count > 1
```



```
    cat.product,
    AVG(p.product_weight_g) as avg_weight,
    COUNT(p.product_id) as product_count,
    STDDEV(p.product_weight_g) as stddev_weight,
    (STDDEV(p.product_weight_g)/SQRT(COUNT(p.product_id))) as stderr_weight,
    (AVG(p.product_weight_g) - (SELECT AVG(p.product_weight_g) FROM products p)) / (STDDEV(p.product_weight_g)/SQRT(COUNT(p.product_id))) as t_value,
    (SELECT COUNT(*) FROM products p1 WHERE p1.product = cat.product) as category_count
FROM
    products p
JOIN
    order_items oi ON oi.product_id = p.product_id
JOIN
    orders o ON o.order_id = oi.order_id
JOIN
    customers c ON c.customer_id = o.customer_id
JOIN
    sellers s ON s.seller_id = oi.seller_id
JOIN
    (SELECT DISTINCT product FROM products) cat ON cat.product = p.product
GROUP BY
    cat.product
HAVING
    category_count > 1
```

product	avg_weight	product_co...	stddev_weight	stderr_weight	t_value	category_cou...
sport leisure	1743.06156694827	8641	3325.2417611995425	35.77183538397597	-14.907614941081226	2867
automotive	2594.1482880755607	4235	4096.868778201577	62.954292344754215	5.046328901019168	1900
bed table bath	2116.753396311291	11115	2975.8063100405425	28.226038321117316	-5.653677517954998	3029
HEALTH BEAUTY	1048.7663906996897	9670	2889.2384050557152	29.381241441260926	-41.780669007563766	2444
Furniture office	11390.454169130691	1691	4851.281418578611	117.97355445984522	77.25561797533655	309
computer accessories	902.5091350453558	7827	1836.1049411908305	20.753905382166153	-66.19598356285759	1639

The results show the t-value for each category, along with the corresponding number of products, average weight, standard deviation of the weight, standard error of the mean, and the number of categories. The t-value indicates the difference between the sample mean and the reference value in units of the standard error, with higher (in absolute value) t-values indicating stronger evidence against the null hypothesis.

It seems that for some categories, there is strong evidence against the null hypothesis, while for others, there is not. For example, the category 'PCs' has a t-value of 12.8, which is very high, indicating strong evidence against the null hypothesis, while the category 'party articles' has a t-value of -0.45, which is low, indicating weak evidence against the null hypothesis.

It is not possible to draw any overall conclusion or make generalizations without more context and a clear research question, but the output could be used as a starting point for further analysis or investigation.

4- Data Load

```
import json
import mysql.connector as mysql
import csv
from mysql.connector import Error

class DataLoader:

   def __init__(self, config_path):
       with open(config_path) as f:
           config = json.load(f)['config']
       self.host = config['host']
       self.port = config['port']
       self.user = config['user']
       self.password = config['password']
       self.database_name = config['database_name']
       self.conn = mysql.connect(
           host=self.host,
           port=self.port,
           user=self.user,
           password=self.password,
           database=self.database_name
       )

       self.cursor = self.conn.cursor()


   def __enter__(self):
           self.conn = mysql.connect(
               user=self.user,
               password=self.password,
               host=self.host,
               database=self.database
           )
           return self

   def __exit__(self, exc_type, exc_val, exc_tb):
       if self.conn:
           self.conn.close()

   @staticmethod
   def load_data_from_csv_to_mysql(csv_file_path, table_name):
       # Connect to database
       try:
```

```python
            conn = mysql.connect(user=loader.user, password=loader.password,
host=loader.host, database=loader.database_name)
            cursor = conn.cursor()
        except Error as e:
            print(f"Error connecting to MySQL: {e}")
            return

        # Open the CSV file
        with open(csv_file_path, mode='r') as csv_file:
            # Read the CSV data using reader class
            csv_reader = csv.reader(csv_file)
            # Skip the header row
            next(csv_reader)

            # Read the CSV row-wise and insert into table
            for row in csv_reader:
                sql = f"INSERT INTO {table_name}
(seller_id,seller_zip_code_prefix,seller_city,seller_state) VALUES
(%s,%s,%s,%s)"
                cursor.execute(sql, tuple(row))
                print("Record inserted")

        # Commit the changes and close the cursor
        conn.commit()
        cursor.close()

loader = DataLoader(config_path='config.json')
loader.load_data_from_csv_to_mysql('sellers.csv', 'sellers2')
```

config.json

```json
{
   "config": {
       "host": "localhost",
       "port": 3306,
       "user": "root",
       "password": "Hima@2023",
       "database_name": "Target"
   }
}
```