

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

1.1 Data type of all columns in the "customers" table.

Code: `Select * FROM `Business_Case_Target_SQL.Customers` limit 100;`

The screenshot shows the Google Cloud Console interface. On the left, the 'Explorer' pane displays the project 'scaler-dsml-405916' and the dataset 'Business_Case_Target_SQL'. The 'customers' table is selected. The main pane shows the query results for the query `Select * FROM `Business_Case_Target_SQL.Customers` limit 100;`. The results are displayed in a table with columns: `customer_id`, `customer_unique_id`, `customer_zip_code_prefix`, `customer_city`, and `customer_state`. The table contains 100 rows of data. The 'customer_id' column contains long alphanumeric strings. The 'customer_unique_id' column contains shorter alphanumeric strings. The 'customer_zip_code_prefix' column contains numeric values. The 'customer_city' column contains city names. The 'customer_state' column contains state abbreviations.

The screenshot shows the Google Cloud Console interface. On the left, the 'Explorer' pane displays the project 'scaler-dsml-405916' and the dataset 'Business_Case_Target_SQL'. The 'customers' table is selected. The main pane shows the schema of the 'customers' table. The schema is displayed in a table with columns: `customer_id`, `customer_unique_id`, `customer_zip_code_prefix`, `customer_city`, and `customer_state`. The table contains 5 columns. The 'customer_id' column is of type `STRING` and is nullable. The 'customer_unique_id' column is of type `STRING` and is nullable. The 'customer_zip_code_prefix' column is of type `INTEGER` and is nullable. The 'customer_city' column is of type `STRING` and is nullable. The 'customer_state' column is of type `STRING` and is nullable.

1.2 Get the time range between which the orders were placed.

Code:

```
WITH OrderWithTimeRange AS (
SELECT
order_id,
order_purchase_timestamp,
LAG(order_purchase_timestamp) OVER (ORDER BY order_purchase_timestamp) AS previous_order_timestamp,
TIMESTAMP_DIFF(order_purchase_timestamp, LAG(order_purchase_timestamp) OVER (ORDER BY
order_purchase_timestamp), HOUR) AS time_range_between_orders
FROM
`Business_Case_Target_SQL.Orders`
)
SELECT
MIN(time_range_between_orders) AS min_range,
MAX(time_range_between_orders) AS max_range
FROM
OrderWithTimeRange;
```

The screenshot shows the Google Cloud BigQuery console interface. On the left, the Explorer pane displays the project 'scaler-dsml-405916' with various resources like Queries, Notebooks, and External connections. The main editor shows a SQL query titled 'Untitled 2' which is the same query provided in the text block. Below the editor, the 'Query results' section is visible, showing a table with two columns: 'min_range' and 'max_range'. The results table has one row with values 0 and 5410280. The bottom of the console shows the 'Job history' section with a 'REFRESH' button.

Query results

Row	min_range	max_range
1	0	5410280

Comment: Many orders were placed at the same time and hence the difference is zero at the lower end and 62 days is the difference at the higher end. The above query I have used 'Seconds' to calculate the time difference.

1.3 Count the Cities & States of customers who ordered during the given period.

Code:

SELECT

COUNT(DISTINCT C.customer_id) as customer_count,

COUNT(DISTINCT customer_city) as count_of_cities,

COUNT(DISTINCT customer_state) as count_of_states

FROM

`Business_Case_Target_SQL.Customers` AS C

JOIN

`Business_Case_Target_SQL.Orders` AS O ON C.customer_id = O.customer_id;

The screenshot shows the Google Cloud BigQuery console interface. The query editor displays the following SQL code:

```

1 SELECT
2   COUNT(DISTINCT C.customer_id) as customer_count,
3   COUNT(DISTINCT customer_city) as count_of_cities,
4   COUNT(DISTINCT customer_state) as count_of_states
5 FROM
6   `Business_Case_Target_SQL.Customers` AS C
7 JOIN
8   `Business_Case_Target_SQL.Orders` AS O ON C.customer_id = O.customer_id;
9

```

The query results are displayed in a table with the following columns: customer_count, count_of_cities, and count_of_states. The results show 99441 customers, 4119 cities, and 27 states.

Row	customer_count	count_of_cities	count_of_states
1	99441	4119	27

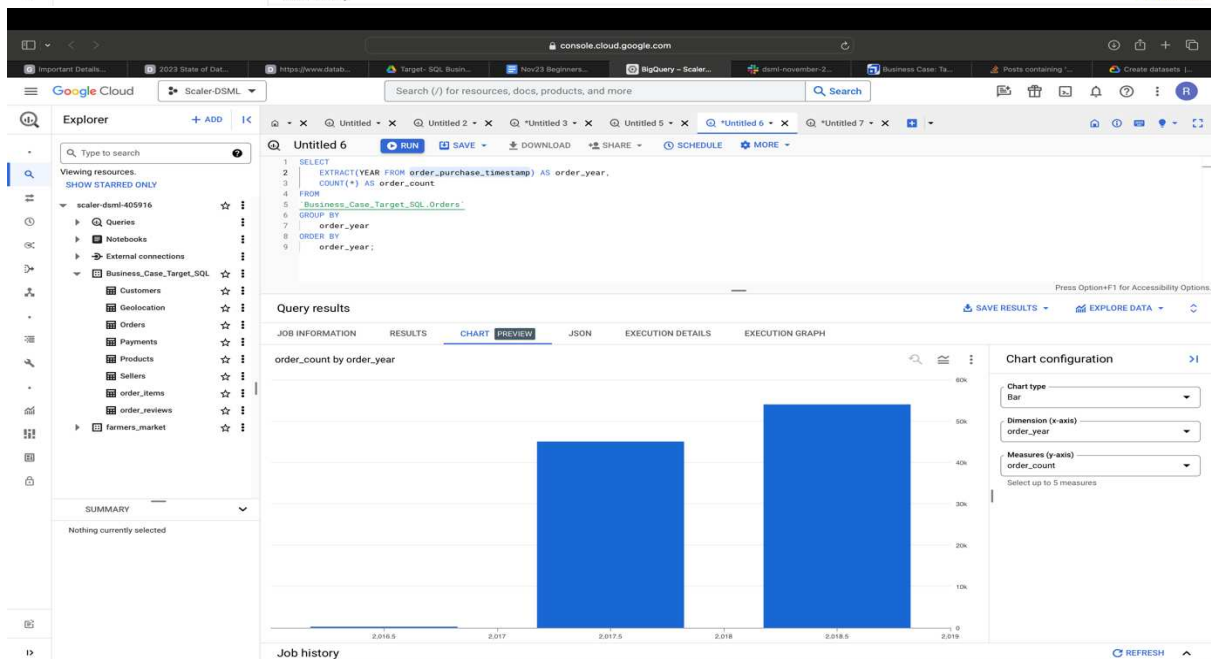
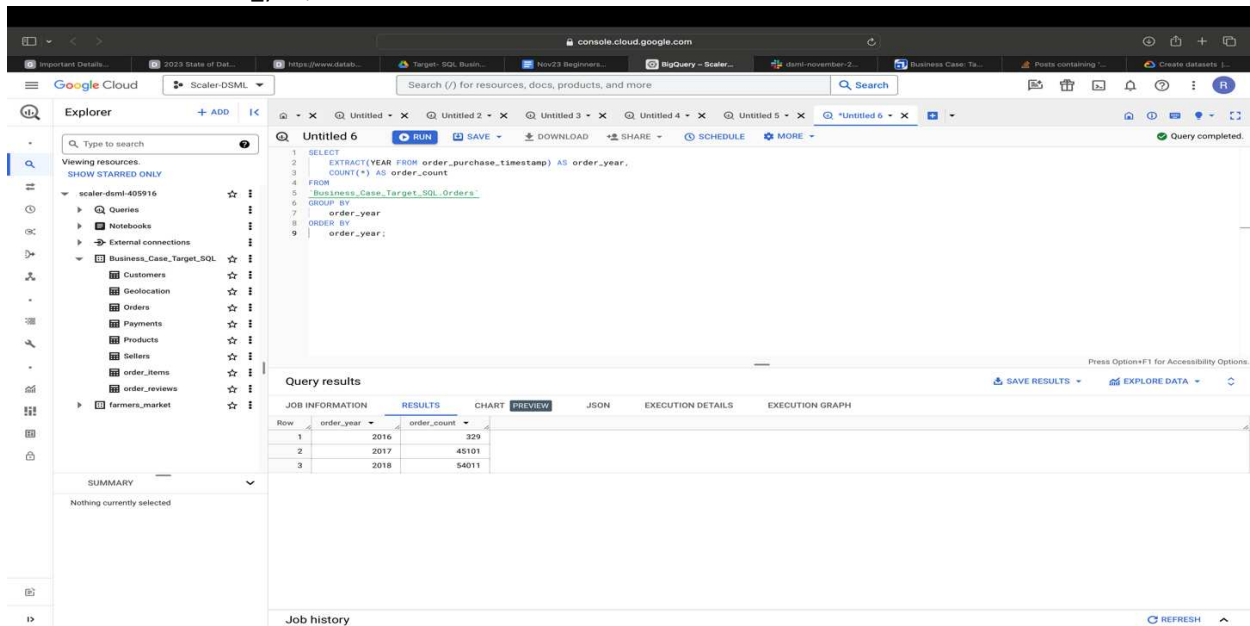
Comment: In total 99441 customers placed orders from 4119 cities which are located in 27 states in Brazil.

2. In-depth Exploration:

2.1 Is there a growing trend in the no. of orders placed over the past years?

Code:

```
SELECT
    EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
    COUNT(*) AS order_count
FROM
    `Business_Case_Target_SQL.Orders`
GROUP BY
    order_year
ORDER BY
    order_year;
```



Comment: There is a 19.7% increase in no. of orders from 2017 to 2018 in spite of a massive dip in orders for Sep and Oct of 2018. Also we have compared 12 months of 2017 data compared with 10 months of 2018 data. Not enough data available for 2016 to be compared.

2.2 Can we see some kind of monthly seasonality in terms of the no. of orders being placed?

Code:

SELECT

```
EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
EXTRACT(MONTH FROM order_purchase_timestamp) AS order_month,
COUNT(*) AS order_count
```

FROM

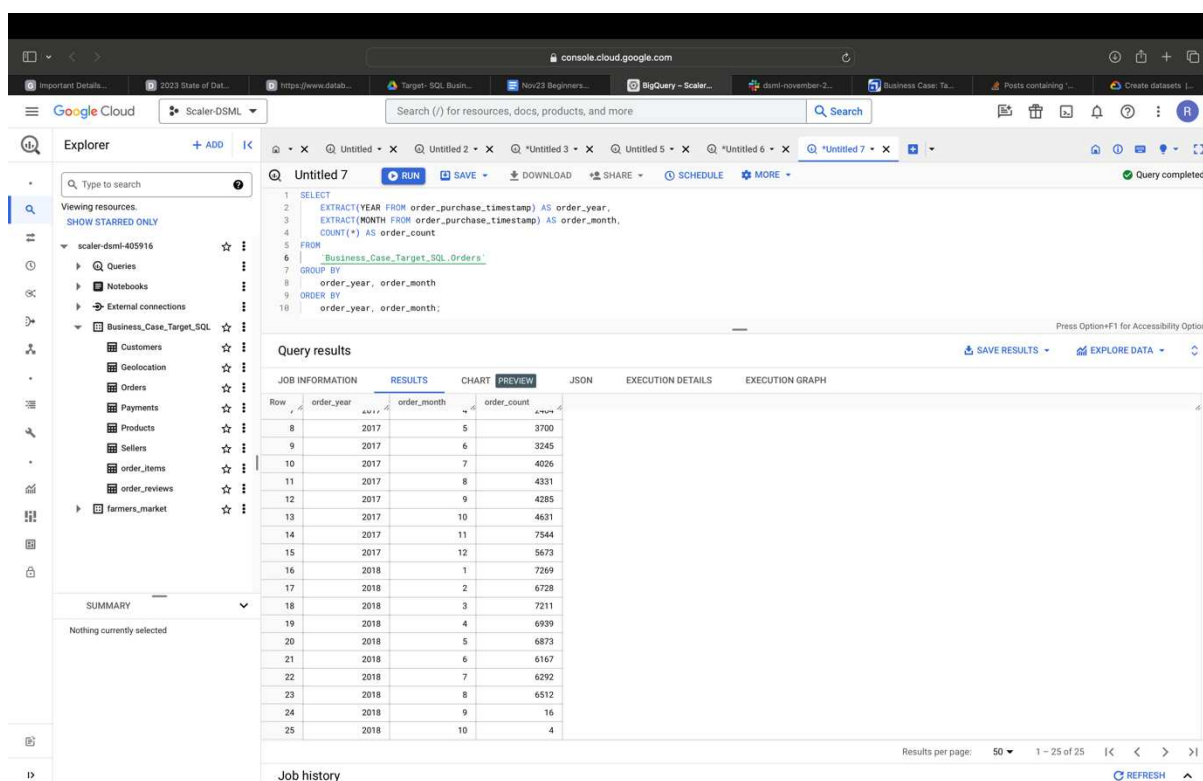
```
`Business_Case_Target_SQL.Orders`
```

GROUP BY

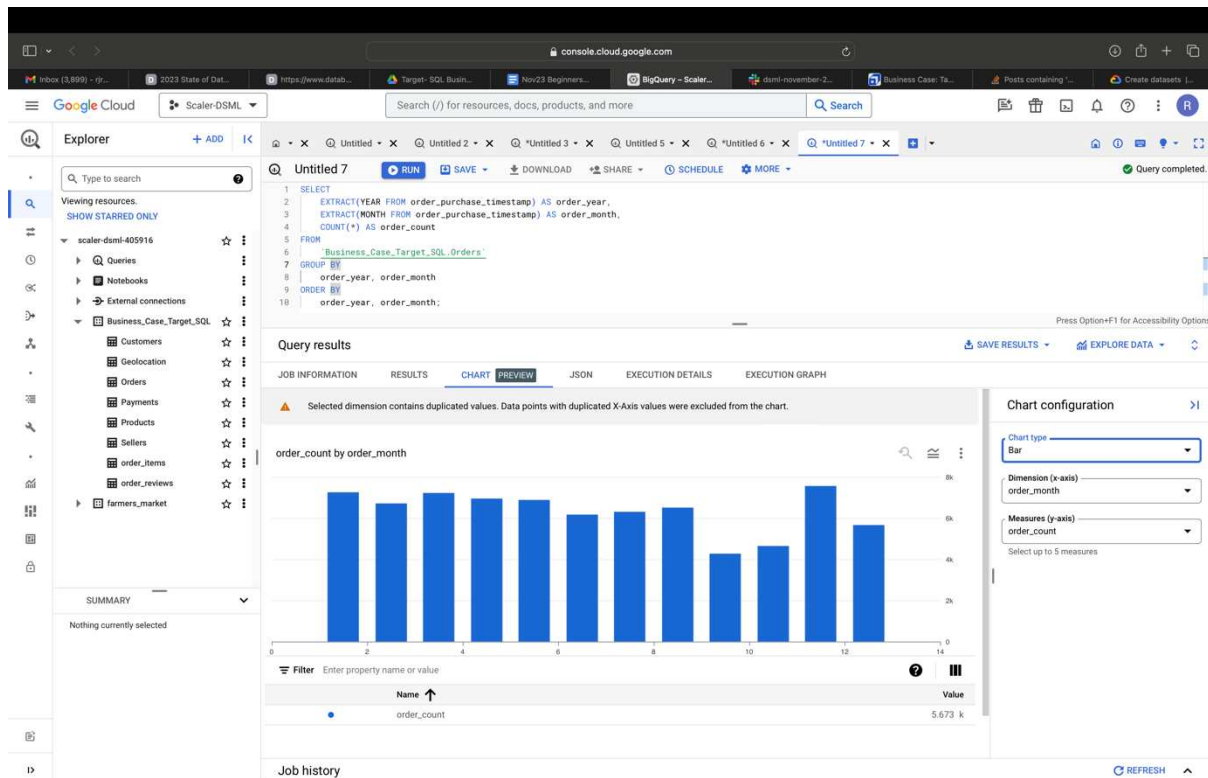
```
order_year, order_month
```

ORDER BY

```
order_year, order_month;
```



Comment: From Feb 2017 there was a steady increase in the no. of orders placed moving from 2000 orders per month to 7500 in the month of Nov 2017 and after a slight fall in Dec 2017 the orders steadied at 6000+ range till August 2018. Not sure about the double digit and single digit order recorded for the month of Sep and Oct 2018 which also resembles the trend during the Sep, Oct and Dec data of 2016.



2.3 During what time of the day, do the Brazilian customers mostly place their orders? (Dawn, Morning, Afternoon or Night)

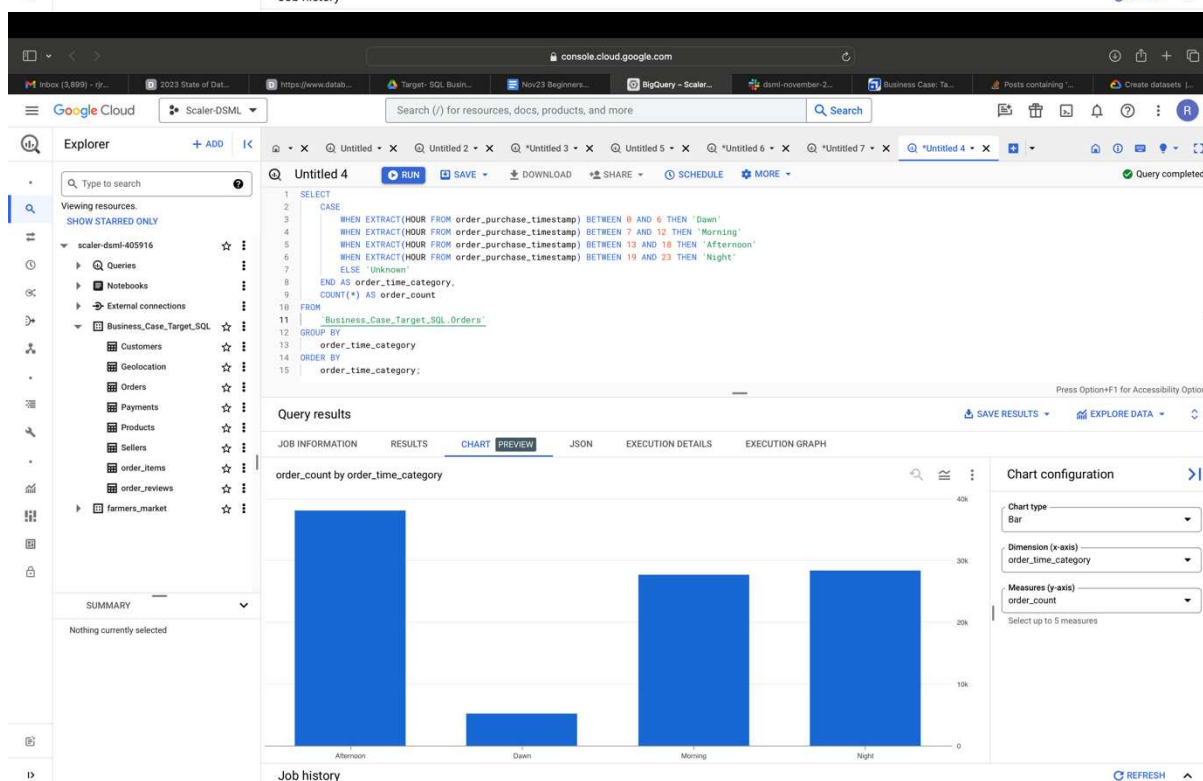
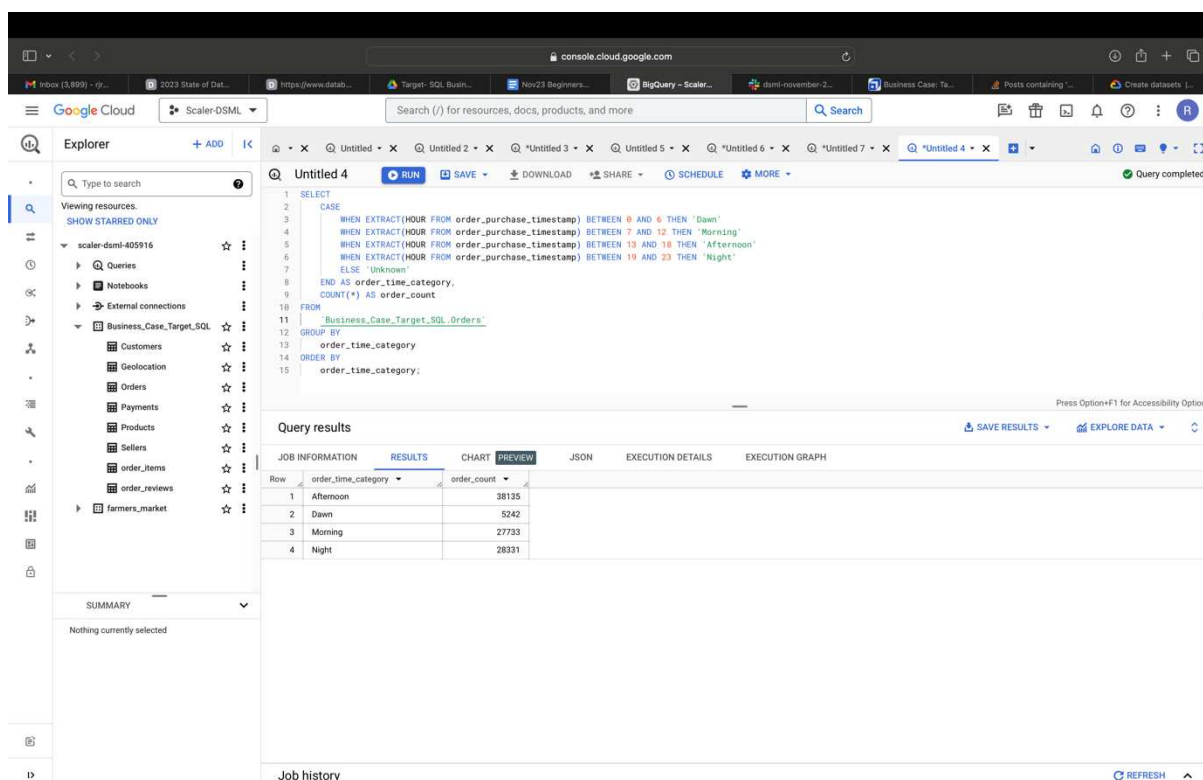
- 0-6 hrs : Dawn
- 7-12 hrs : Mornings
- 13-18 hrs : Afternoon
- 19-23 hrs : Night

Code:

```

SELECT
CASE
  WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 0 AND 6 THEN 'Dawn'
  WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 7 AND 12 THEN 'Morning'
  WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 13 AND 18 THEN 'Afternoon'
  WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 19 AND 23 THEN 'Night'
  ELSE 'Unknown'
END AS order_time_category,
COUNT(*) AS order_count
FROM
  `Business_Case_Target_SQL.Orders`
GROUP BY
  order_time_category
ORDER BY
  order_time_category;

```

Comment: 95% of the orders were placed during Afternoon and Night and Morning time.
Around 38.5% of the orders were placed during Afternoon

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

3.1 Get the month on month no. of orders placed in each state.

Code:

```
SELECT
    EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
    EXTRACT(MONTH FROM order_purchase_timestamp) AS order_month,
    C.customer_state,
    COUNT(*) AS order_count
FROM
    `Business_Case_Target_SQL.Orders` as O
JOIN
    `Business_Case_Target_SQL.Customers` C ON O.customer_id =
C.customer_id
GROUP BY
    order_year, order_month, C.customer_state
ORDER BY
    order_year, order_month, C.customer_state;
```

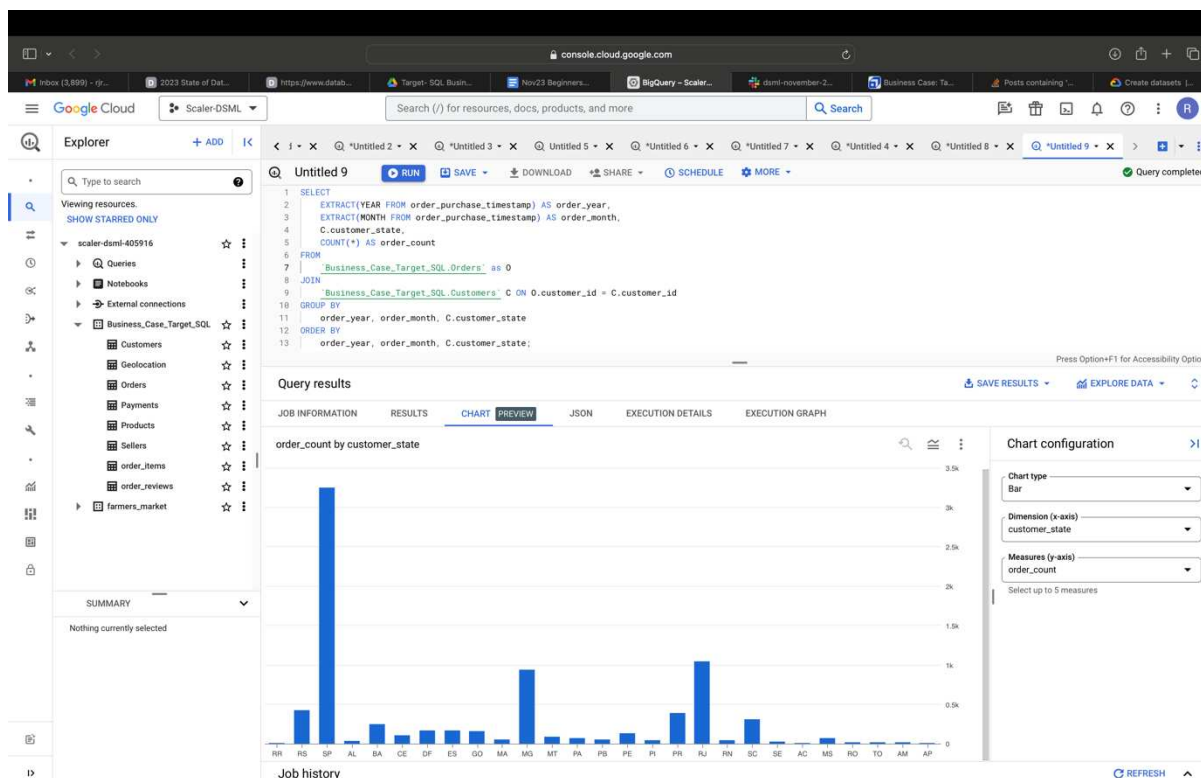
The screenshot displays the Google Cloud BigQuery console interface. On the left, the 'Explorer' sidebar shows a project named 'scaler-dsml-405916' with various resources like Queries, Notebooks, and External connections. The main area shows a SQL query in 'Untitled 9' with the following code:

```
1 SELECT
2   EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
3   EXTRACT(MONTH FROM order_purchase_timestamp) AS order_month,
4   C.customer_state,
5   COUNT(*) AS order_count
6 FROM
7   `Business_Case_Target_SQL.Orders` as O
8 JOIN
9   `Business_Case_Target_SQL.Customers` C ON O.customer_id = C.customer_id
10 GROUP BY
11   order_year, order_month, C.customer_state
12 ORDER BY
13   order_year, order_month, C.customer_state;
```

Below the query editor, the 'Query results' section is visible, showing a table with 17 rows and 4 columns: order_year, order_month, customer_state, and order_count. The results are as follows:

Row	order_year	order_month	customer_state	order_count
1	2016	9	RR	1
2	2016	9	RS	1
3	2016	9	SP	2
4	2016	10	AL	2
5	2016	10	BA	4
6	2016	10	CE	8
7	2016	10	DF	6
8	2016	10	ES	4
9	2016	10	GO	9
10	2016	10	MA	4
11	2016	10	MG	40
12	2016	10	MT	3
13	2016	10	PA	4
14	2016	10	PB	1
15	2016	10	PE	7
16	2016	10	PI	1
17	2016	10	PP	10

At the bottom, the 'Job history' section is visible with a 'REFRESH' button.



Comment: Sao Paulo(SP) state being the most populated and having the highest economic activity in Brazil is clearly in the no. 1 spot by having the most no. of orders. The next state Rio De Janeiro(RJ) and Minas Gerais(MG) which are 2nd and 3rd spot put together still fall short of total orders placed in SP. The key factor here is the population as RJ & MG have less than 40% & 50% population of SP respectively. Though RJ has less population than MG the orders placed from there are higher.

3.2 How are the customers distributed across all the states?

Code:

SELECT

customer_state,

COUNT(DISTINCT customer_id) AS customer_count

FROM

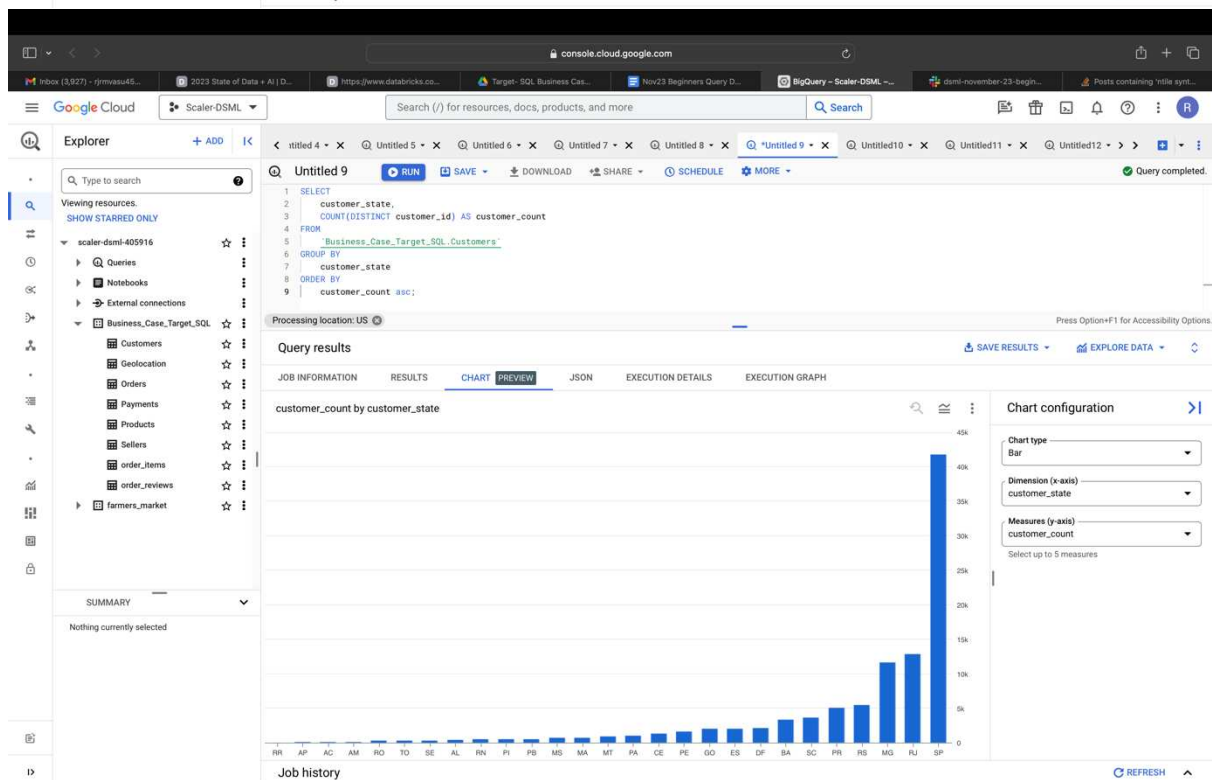
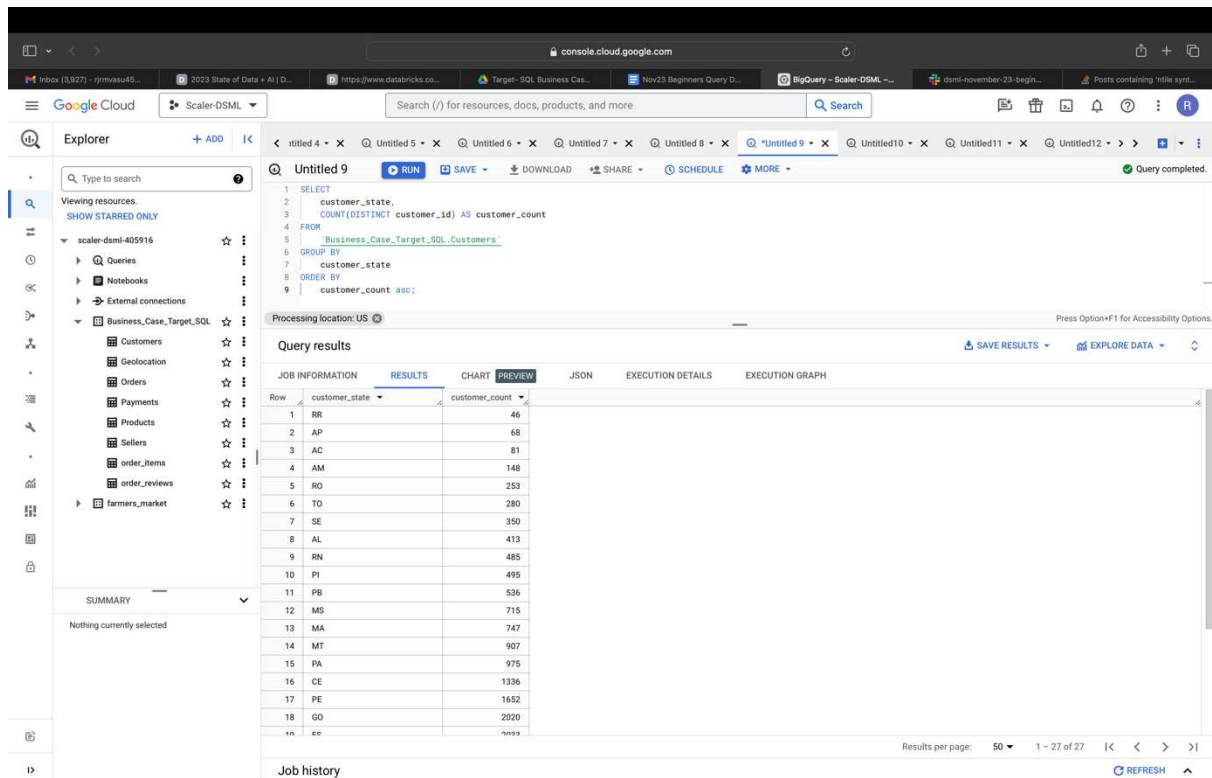
`Business_Case_Target_SQL.Customers`

GROUP BY

customer_state

ORDER BY

customer_state;



Comment: As mentioned in the previous graph, the no. of customers from each state clearly shows the population density and economic activity of each state.

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

4.1 Get the % increase in the cost of orders from year 2017 to 2018 (include months between Jan to Aug only). You can use the "payment_value" column in the payments table to get the cost of orders.

Code:

```
WITH OrderCosts AS (  
  SELECT  
    EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,  
    SUM(payment_value) AS total_cost  
  FROM  
    `Business_Case_Target_SQL.Orders` as O  
  JOIN  
    `Business_Case_Target_SQL.Payments` P ON P.order_id = O.order_id  
  
  WHERE  
    EXTRACT(YEAR FROM order_purchase_timestamp) IN (2017, 2018)  
    AND EXTRACT(MONTH FROM order_purchase_timestamp) BETWEEN 1 AND 8  
  GROUP BY  
    order_year  
)  
SELECT  
  2017 AS base_year,  
  2018 AS target_year,  
  COALESCE(  
    ((OC_2018.total_cost - OC_2017.total_cost) / OC_2017.total_cost) * 100,0) AS percentage_increase  
FROM  
  OrderCosts OC_2017  
JOIN  
  OrderCosts OC_2018 ON OC_2017.order_year = OC_2018.order_year  
WHERE  
  OC_2017.order_year = 2017;
```

The screenshot shows the Google Cloud BigQuery console. On the left is the Explorer pane with a search bar and a list of resources under 'scaler-dsml-405916', including Queries, Notebooks, External connections, and various tables like Customers, Orders, Payments, etc. The main editor shows a SQL query in 'Untitled 11'. The query calculates the percentage increase in total order cost from 2017 to 2018 for a specific customer state. The query results pane at the bottom shows a single row of data.

```

1 WITH OrderCosts AS (
2   SELECT
3     EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
4     SUM(payment_value) AS total_cost
5   FROM
6     `Business_Case_Target_SQL.Orders` AS O
7   JOIN
8     `Business_Case_Target_SQL.Payments` P ON P.order_id = O.order_id
9   WHERE
10    EXTRACT(YEAR FROM order_purchase_timestamp) IN (2017, 2018)
11    AND EXTRACT(MONTH FROM order_purchase_timestamp) BETWEEN 1 AND 8
12   GROUP BY
13     order_year
14 )
15 SELECT
16   2017 AS base_year,
17   2018 AS target_year,
18   COALESCE(
19     ((OC_2018.total_cost - OC_2017.total_cost) / OC_2017.total_cost) * 100, 0) AS percentage_increase
20 FROM
21   OrderCosts OC_2017
22 JOIN
23   OrderCosts OC_2018 ON OC_2017.order_year = OC_2018.order_year
24 WHERE
25   OC_2017.order_year = 2017;

```

Query results

Row	base_year	target_year	percentage_increase
1	2017	2018	-2.42411473593...

Job history

Comment: As per the data there is a slight decrease in the cost of orders in 2018 when compared to 2017.

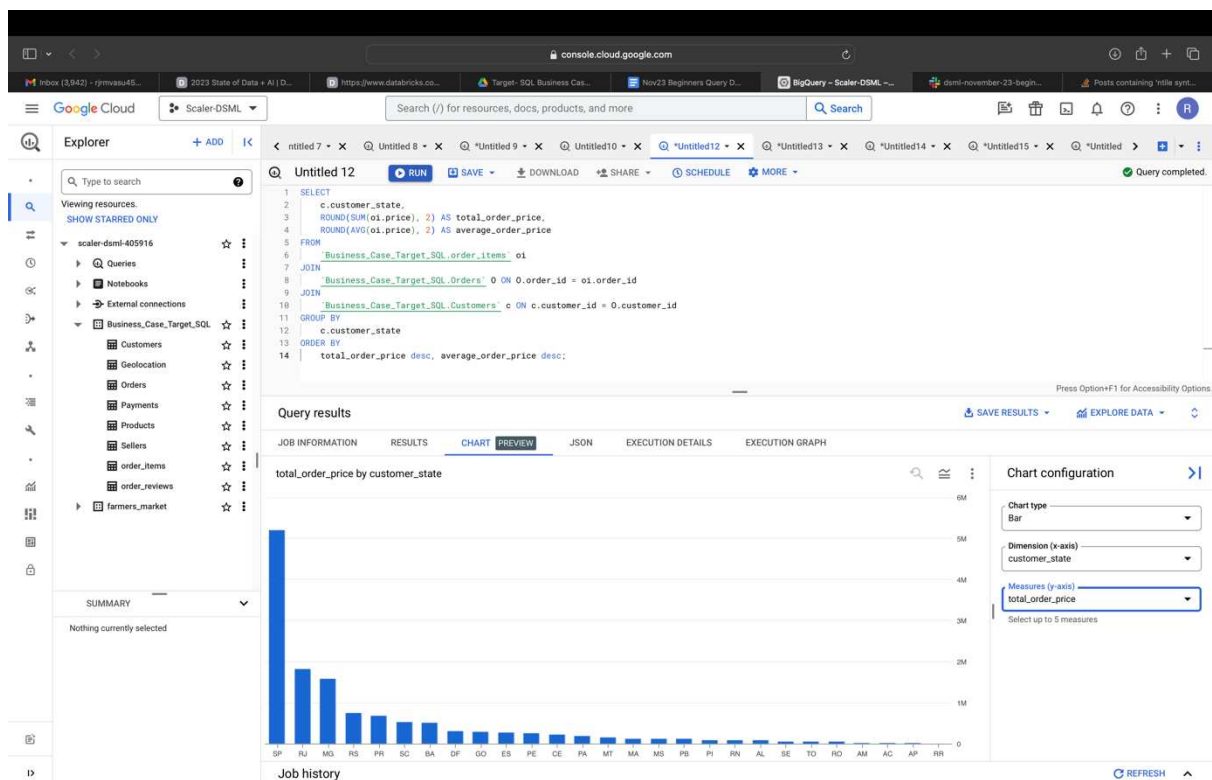
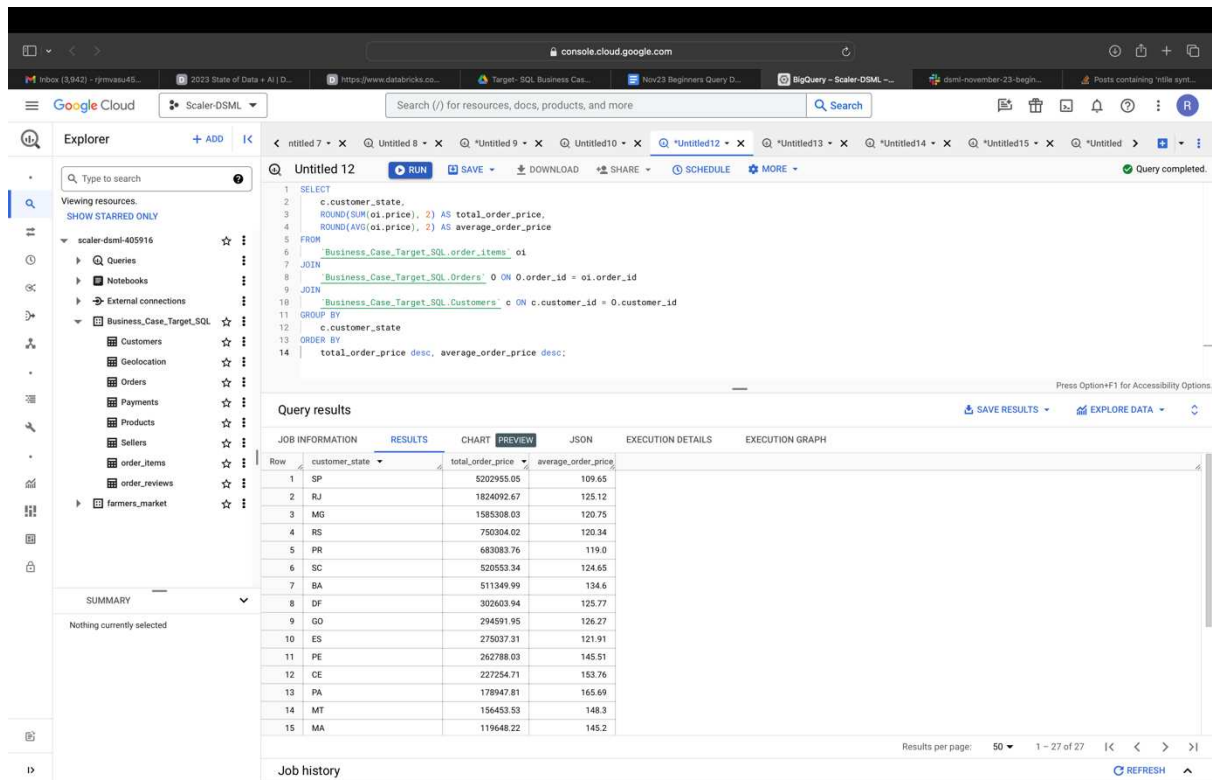
4.2 Calculate the Total & Average value of order price for each state.

Code:

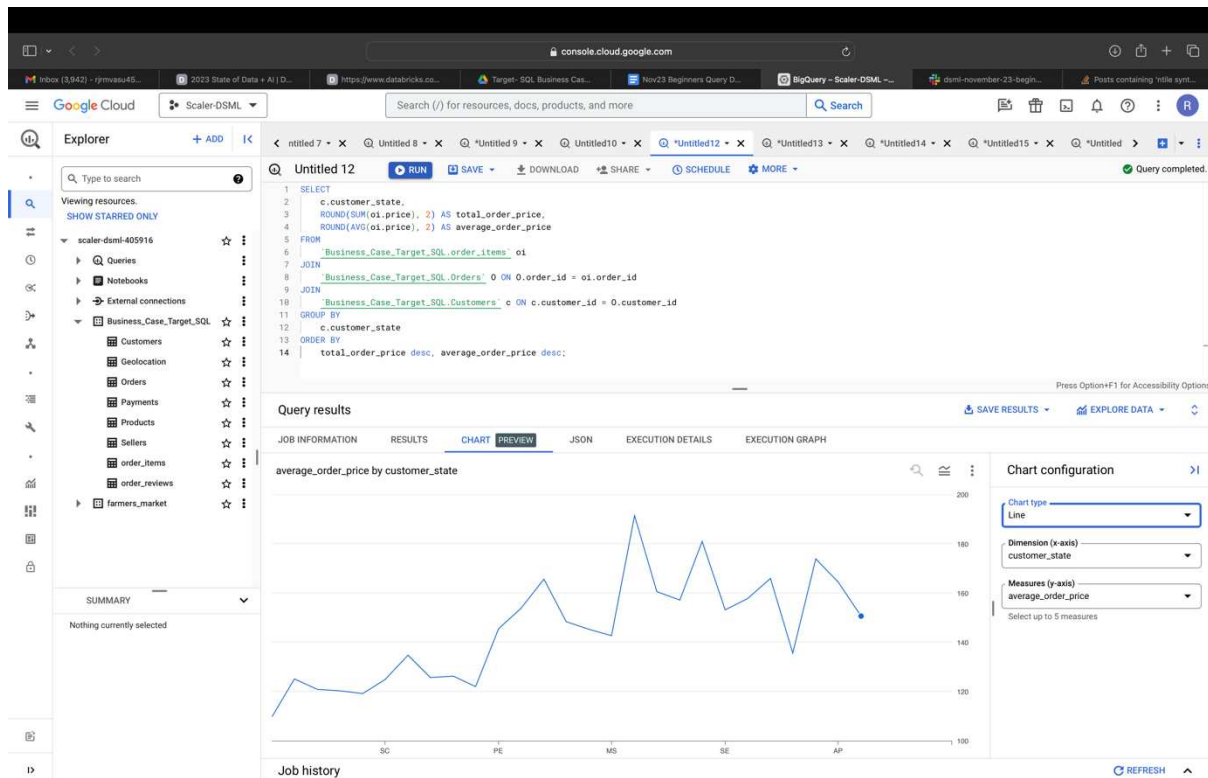
```

SELECT
  c.customer_state,
  ROUND(SUM(oi.price), 2) AS total_order_price,
  ROUND(AVG(oi.price), 2) AS average_order_price
FROM
  `Business_Case_Target_SQL.order_items` oi
JOIN
  `Business_Case_Target_SQL.Orders` O ON O.order_id = oi.order_id
JOIN
  `Business_Case_Target_SQL.Customers` c ON c.customer_id = O.customer_id
GROUP BY
  c.customer_state
ORDER BY
  c.customer_state;

```



Comment: The top 5 states were most orders placed contribute to the total value of the orders.



Comment: Though the top 5 states were most orders placed have the lesser average order price but it is not consistent with where lower the orders, higher the average is the story for the rest of the states. Need deeper analysis to find the other contributing factors.

4.3 Calculate the Total & Average value of order freight for each state.

Code:

```

SELECT
  c.customer_state,
  ROUND(SUM(oi.price), 2) AS total_freight_value,
  ROUND(AVG(oi.price), 2) AS average_freight_value
FROM
  `Business_Case_Target_SQL.order_items` oi
JOIN
  `Business_Case_Target_SQL.Orders` O ON O.order_id = oi.order_id
JOIN
  `Business_Case_Target_SQL.Customers` c ON c.customer_id = O.customer_id
GROUP BY
  c.customer_state
ORDER BY
  c.customer_state;

```

console.cloud.google.com

Google Cloud Scaler-DSML

Search (/) for resources, docs, products, and more

Explorer

Type to search

Viewing resources. SHOW STARRED ONLY

scaler-dsml-405916

- Queries
- Notebooks
- External connections
- Business_Case_Target_SQL
 - Customers
 - Geolocation
 - Orders
 - Payments
 - Products
 - Sellers
 - order_items
 - order_reviews
 - farmers_market

SUMMARY

Nothing currently selected

Untitled 11

```

1 SELECT
2   c.customer_state,
3   ROUND(SUM(o1.freight_value), 2) AS total_freight_value,
4   ROUND(AVG(o1.freight_value), 2) AS average_freight_value
5 FROM
6   `Business_Case_Target_SQL_order_items` o1
7 JOIN
8   `Business_Case_Target_SQL_Orders` o ON o.order_id = o1.order_id
9 JOIN
10  `Business_Case_Target_SQL_Customers` c ON c.customer_id = o.customer_id
11 GROUP BY
12   c.customer_state
13 ORDER BY
14   total_freight_value desc, average_freight_value desc;

```

Query results

JOB INFORMATION RESULTS CHART PREVIEW JSON EXECUTION DETAILS EXECUTION GRAPH

Row	customer_state	total_freight_value	average_freight_value
1	SP	718723.07	15.15
2	RJ	305589.31	20.96
3	MG	270853.46	20.63
4	RS	135522.74	21.74
5	PR	117851.68	20.53
6	BA	100156.68	26.36
7	SC	89660.26	21.47
8	PE	59449.66	32.92
9	GO	53114.98	22.77
10	DF	50625.5	21.04
11	ES	49764.6	22.06
12	CE	48351.59	32.71
13	PA	38699.3	35.83
14	MA	31523.77	38.26
15	MT	29715.43	28.17

Results per page: 50 1 - 27 of 27

Job history

console.cloud.google.com

Google Cloud Scaler-DSML

Search (/) for resources, docs, products, and more

Explorer

Type to search

Viewing resources. SHOW STARRED ONLY

scaler-dsml-405916

- Queries
- Notebooks
- External connections
- Business_Case_Target_SQL
 - Customers
 - Geolocation
 - Orders
 - Payments
 - Products
 - Sellers
 - order_items
 - order_reviews
 - farmers_market

SUMMARY

Nothing currently selected

Untitled 11

```

1 SELECT
2   c.customer_state,
3   ROUND(SUM(o1.freight_value), 2) AS total_freight_value,
4   ROUND(AVG(o1.freight_value), 2) AS average_freight_value
5 FROM
6   `Business_Case_Target_SQL_order_items` o1
7 JOIN
8   `Business_Case_Target_SQL_Orders` o ON o.order_id = o1.order_id
9 JOIN
10  `Business_Case_Target_SQL_Customers` c ON c.customer_id = o.customer_id
11 GROUP BY
12   c.customer_state
13 ORDER BY
14   total_freight_value desc, average_freight_value desc;

```

Query results

JOB INFORMATION RESULTS CHART PREVIEW JSON EXECUTION DETAILS EXECUTION GRAPH

average_freight_value, total_freight_value by customer_state

Chart configuration

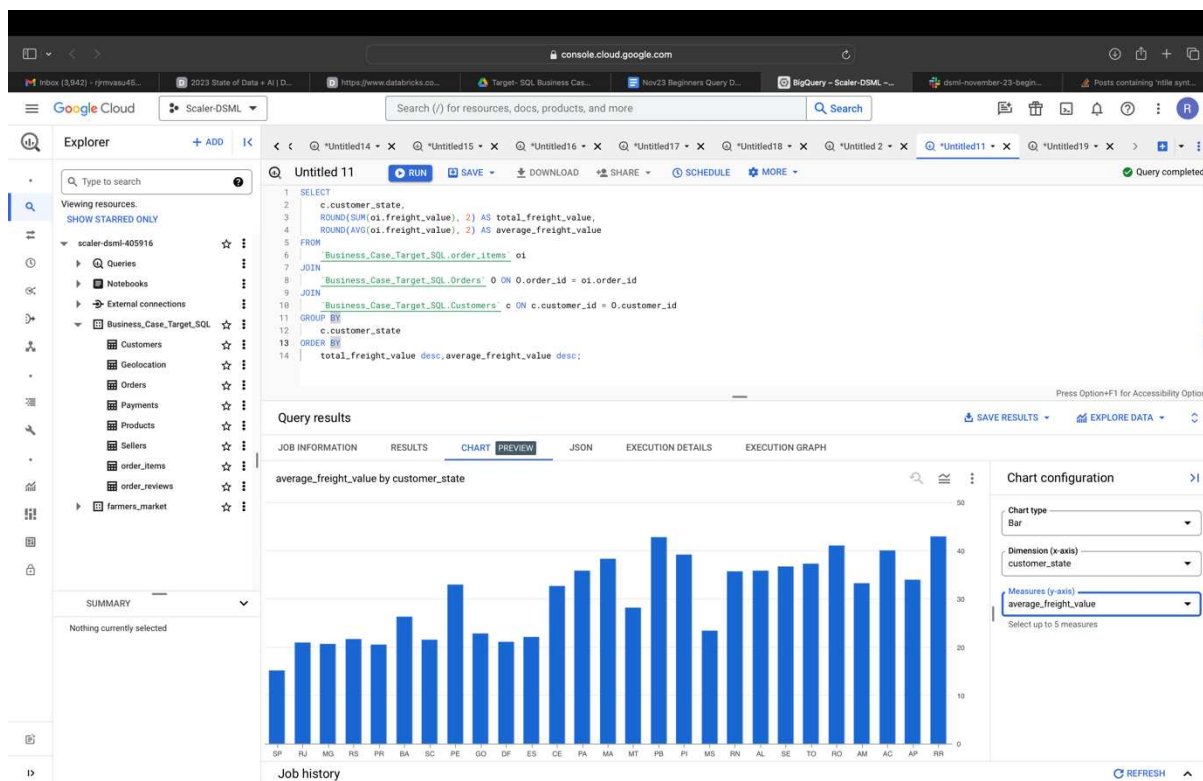
Chart type: Bar

Dimension (x-axis): customer_state

Measures (y-axis): average_freight_value and total_freight_v...

Select up to 5 measures

Job history



The screenshot shows the Google Cloud Console interface. On the left, the 'Explorer' sidebar displays the project 'scaler-dsml-405916'. The main area shows a BigQuery query in 'Untitled13' with the following SQL code:

```

1 SELECT
2   order_id,
3   order_purchase_timestamp,
4   order_delivered_customer_date,
5   order_estimated_delivery_date,
6   TIMESTAMP_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS delivery_time,
7   TIMESTAMP_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY) AS diff_estimated_delivery
8 FROM
9   `Business_Case_Target_SQL.Orders`
10 WHERE
11   order_delivered_customer_date IS NOT NULL and order_estimated_delivery_date IS NOT NULL
12 ORDER BY delivery_time ASC, diff_estimated_delivery ASC;

```

Below the query editor, the 'Query results' section shows a table with 16 rows and 7 columns. The columns are: Row, order_id, order_purchase_timestamp, order_delivered_customer_date, order_estimated_delivery_date, delivery_time, and diff_estimated_delivery. The data shows various orders with their timestamps and delivery times.

5.2 Find out the top 5 states with the highest & lowest average freight value.

Code:

```

WITH Top5States AS (
  SELECT
    C.customer_state,
    ROUND(AVG(oi.freight_value), 2) AS avg_freight_value
  FROM
    `Business_Case_Target_SQL.order_items` AS oi
  JOIN
    `Business_Case_Target_SQL.Orders` AS O ON O.order_id = oi.order_id
  JOIN
    `Business_Case_Target_SQL.Customers` AS C ON C.customer_id = O.customer_id
  GROUP BY
    C.customer_state
  ORDER BY
    avg_freight_value DESC
  LIMIT 5
),
Bottom5States AS (
  SELECT
    C.customer_state,
    ROUND(AVG(oi.freight_value), 2) AS avg_freight_value
  FROM
    `Business_Case_Target_SQL.order_items` AS oi
  JOIN
    `Business_Case_Target_SQL.Orders` AS O ON O.order_id = oi.order_id
  JOIN
    `Business_Case_Target_SQL.Customers` AS C ON C.customer_id = O.customer_id
  GROUP BY
    C.customer_state
  ORDER BY
    avg_freight_value ASC
  LIMIT 5
)

```

```

`Business_Case_Target_SQL.Customers` AS C ON C.customer_id = O.customer_id
GROUP BY
    C.customer_state
ORDER BY
    avg_freight_value ASC
LIMIT 5
)

SELECT * FROM Top5States
UNION ALL
SELECT * FROM Bottom5States;

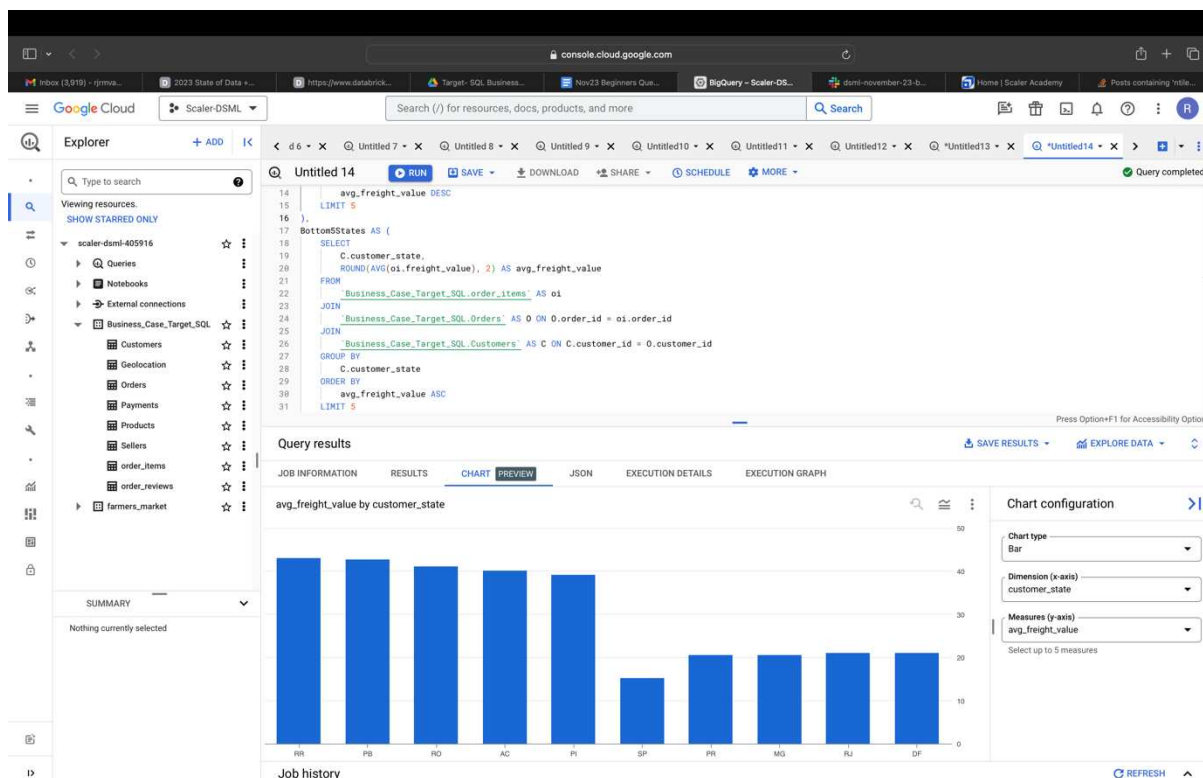
```

Query results

Row	customer_state	avg_freight_value
1	RR	42.98
2	PB	42.72
3	RO	41.07
4	AC	40.07
5	PI	39.15
6	SP	15.15
7	PR	20.53
8	MG	20.63
9	RJ	20.96
10	DF	21.04

Query results

Row	customer_state	avg_freight_value
1	RR	42.98
2	PB	42.72
3	RO	41.07
4	AC	40.07
5	PI	39.15
6	SP	15.15
7	PR	20.53
8	MG	20.63
9	RJ	20.96
10	DF	21.04



Comment: The lesser the orders more the average of the freight value which can be seen with states like AC, RR PB, RO & AC, the state which placed most orders having the lowest average of the freight value like SP, MG, RJ, etc.

5.3 Find out the top 5 states with the highest & lowest average delivery time.

Code:

```

WITH Top5States AS (
SELECT
  C.customer_state,
  O.order_purchase_timestamp,
  O.order_delivered_customer_date,
  TIMESTAMP_DIFF(O.order_delivered_customer_date,
O.order_purchase_timestamp, DAY) AS delivery_time
FROM
  `Business_Case_Target_SQL.order_items` AS oi
JOIN
  `Business_Case_Target_SQL.Orders` AS O ON O.order_id =
oi.order_id
JOIN
  `Business_Case_Target_SQL.Customers` AS C ON C.customer_id =
O.customer_id
WHERE
  O.order_delivered_customer_date IS NOT NULL and
O.order_purchase_timestamp IS NOT NULL
GROUP BY

```

```

        C.customer_state,
        O.order_purchase_timestamp,O.order_delivered_customer_date
        ORDER BY
            delivery_time DESC
        LIMIT 5
    ),
    Bottom5States AS (
        SELECT
            C.customer_state,
            O.order_purchase_timestamp,
            O.order_delivered_customer_date,
            TIMESTAMP_DIFF(O.order_delivered_customer_date,
        O.order_purchase_timestamp, DAY) AS delivery_time
        FROM
            `Business_Case_Target_SQL.order_items` AS oi

        JOIN
            `Business_Case_Target_SQL.Orders` AS O ON O.order_id =
        oi.order_id
        JOIN
            `Business_Case_Target_SQL.Customers` AS C ON C.customer_id =
        O.customer_id
        WHERE
            O.order_delivered_customer_date IS NOT NULL and
        O.order_purchase_timestamp IS NOT NULL
        GROUP BY
            C.customer_state,
        O.order_purchase_timestamp,O.order_delivered_customer_date
        ORDER BY
            delivery_time
        LIMIT 5
    )

SELECT * FROM Top5States
UNION ALL
SELECT * FROM Bottom5States;

```

The screenshot shows the Google Cloud BigQuery console interface. The left sidebar displays the Explorer view with a tree structure of resources, including 'Business_Case_Target_SQL' and its sub-items like 'Customers', 'Geolocation', 'Orders', 'Payments', 'Products', 'Sellers', 'order_items', 'order_reviews', and 'farmers_market'. The main panel shows a SQL query titled 'Untitled 15' with the following code:

```

1 WITH TopStates AS (
2   SELECT
3     C.customer_state,
4     0.order_purchase_timestamp,
5     0.order_delivered_customer_date,
6     TIMESTAMP_DIFF(0.order_delivered_customer_date, 0.order_purchase_timestamp, DAY) AS delivery_time
7   FROM
8     'Business_Case_Target_SQL.order_items' AS oi
9   JOIN
10    'Business_Case_Target_SQL.Orders' AS O ON O.order_id = oi.order_id
11   JOIN
12    'Business_Case_Target_SQL.Customers' AS C ON C.customer_id = 0.customer_id
13   WHERE
14     0.order_delivered_customer_date IS NOT NULL and 0.order_purchase_timestamp IS NOT NULL
15   GROUP BY
16     C.customer_state, 0.order_purchase_timestamp, 0.order_delivered_customer_date
17   ORDER BY
18     delivery_time DESC
19   LIMIT 5
20 );

```

The 'Query results' section displays a table with 10 rows and 5 columns: 'customer_state', 'order_purchase_timestamp', 'order_delivered_customer_date', and 'delivery_time'. The data is as follows:

Row	customer_state	order_purchase_timestamp	order_delivered_customer_date	delivery_time
1	SP	2017-11-16 13:54:08 UTC	2017-11-17 13:49:40 UTC	0
2	SP	2018-05-18 15:03:19 UTC	2018-05-19 12:28:30 UTC	0
3	SP	2017-05-31 11:11:55 UTC	2017-06-01 08:34:36 UTC	0
4	RJ	2017-06-19 08:19:45 UTC	2017-06-19 21:07:52 UTC	0
5	SP	2017-05-29 13:21:46 UTC	2017-05-30 08:06:56 UTC	0
6	ES	2017-02-21 23:31:27 UTC	2017-09-19 14:36:39 UTC	209
7	RJ	2018-02-23 14:57:35 UTC	2018-09-19 23:24:07 UTC	208
8	PA	2017-03-07 23:59:51 UTC	2017-09-19 15:12:50 UTC	195
9	PI	2017-03-09 13:26:57 UTC	2017-09-19 14:38:21 UTC	194
10	SE	2017-03-08 22:47:40 UTC	2017-09-19 14:00:04 UTC	194

The 'Job history' section is visible at the bottom of the results panel.

The screenshot shows the Google Cloud BigQuery console interface. The left sidebar displays the Explorer view with a tree structure of resources, including 'Business_Case_Target_SQL' and its sub-items like 'Customers', 'Geolocation', 'Orders', 'Payments', 'Products', 'Sellers', 'order_items', 'order_reviews', and 'farmers_market'. The main panel shows a SQL query titled 'Untitled 15' with the following code:

```

21 //
22 BottomStates AS (
23   SELECT
24     C.customer_state,
25     0.order_purchase_timestamp,
26     0.order_delivered_customer_date,
27     TIMESTAMP_DIFF(0.order_delivered_customer_date, 0.order_purchase_timestamp, DAY) AS delivery_time
28   FROM
29     'Business_Case_Target_SQL.order_items' AS oi
30   JOIN
31    'Business_Case_Target_SQL.Orders' AS O ON O.order_id = oi.order_id
32   JOIN
33    'Business_Case_Target_SQL.Customers' AS C ON C.customer_id = 0.customer_id
34   WHERE
35     0.order_delivered_customer_date IS NOT NULL and 0.order_purchase_timestamp IS NOT NULL
36   GROUP BY
37     C.customer_state, 0.order_purchase_timestamp, 0.order_delivered_customer_date
38   ORDER BY
39     delivery_time
40   LIMIT 5
41 )
42
43 SELECT * FROM TopStates
44 UNION ALL
45 SELECT * FROM BottomStates;

```

The 'Query results' section displays a table with 10 rows and 5 columns: 'customer_state', 'order_purchase_timestamp', 'order_delivered_customer_date', and 'delivery_time'. The data is as follows:

Row	customer_state	order_purchase_timestamp	order_delivered_customer_date	delivery_time
1	SP	2017-11-16 13:54:08 UTC	2017-11-17 13:49:40 UTC	0
2	SP	2018-05-18 15:03:19 UTC	2018-05-19 12:28:30 UTC	0
3	SP	2017-05-31 11:11:55 UTC	2017-06-01 08:34:36 UTC	0
4	RJ	2017-06-19 08:19:45 UTC	2017-06-19 21:07:52 UTC	0
5	SP	2017-05-29 13:21:46 UTC	2017-05-30 08:06:56 UTC	0
6	ES	2017-02-21 23:31:27 UTC	2017-09-19 14:36:39 UTC	209
7	RJ	2018-02-23 14:57:35 UTC	2018-09-19 23:24:07 UTC	208
8	PA	2017-03-07 23:59:51 UTC	2017-09-19 15:12:50 UTC	195
9	PI	2017-03-09 13:26:57 UTC	2017-09-19 14:38:21 UTC	194
10	SE	2017-03-08 22:47:40 UTC	2017-09-19 14:00:04 UTC	194

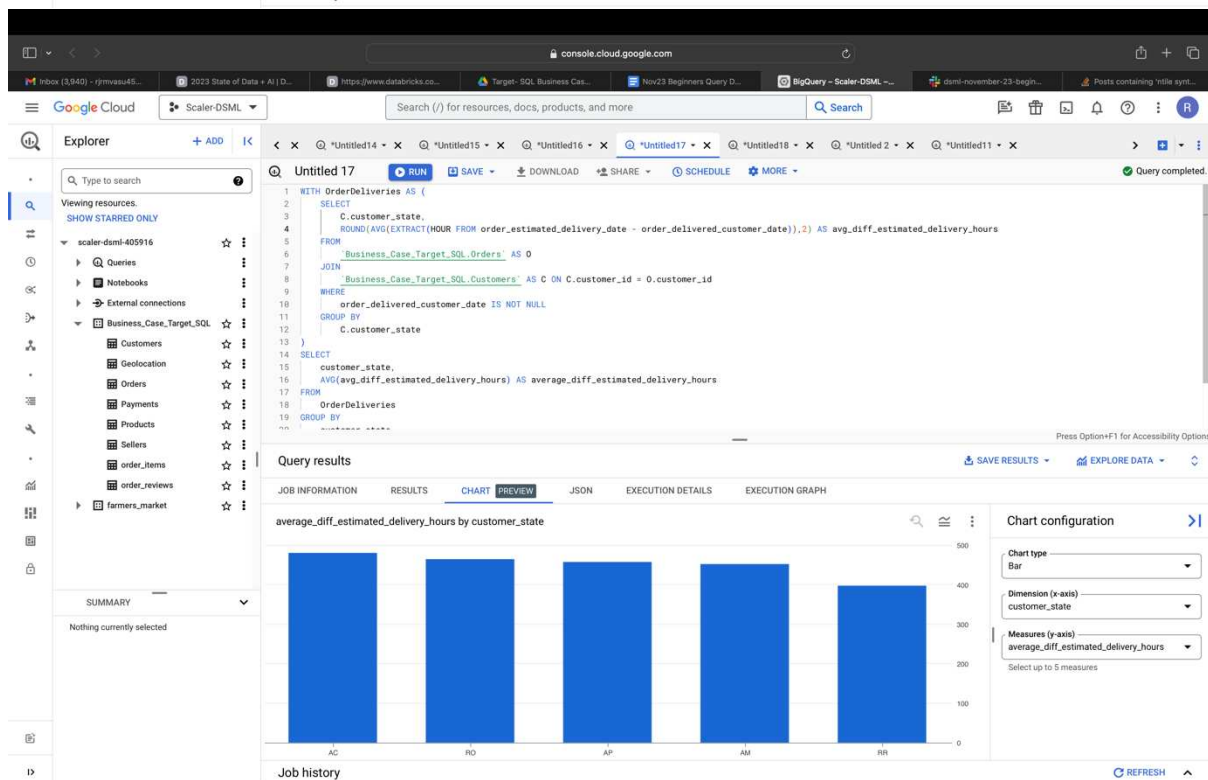
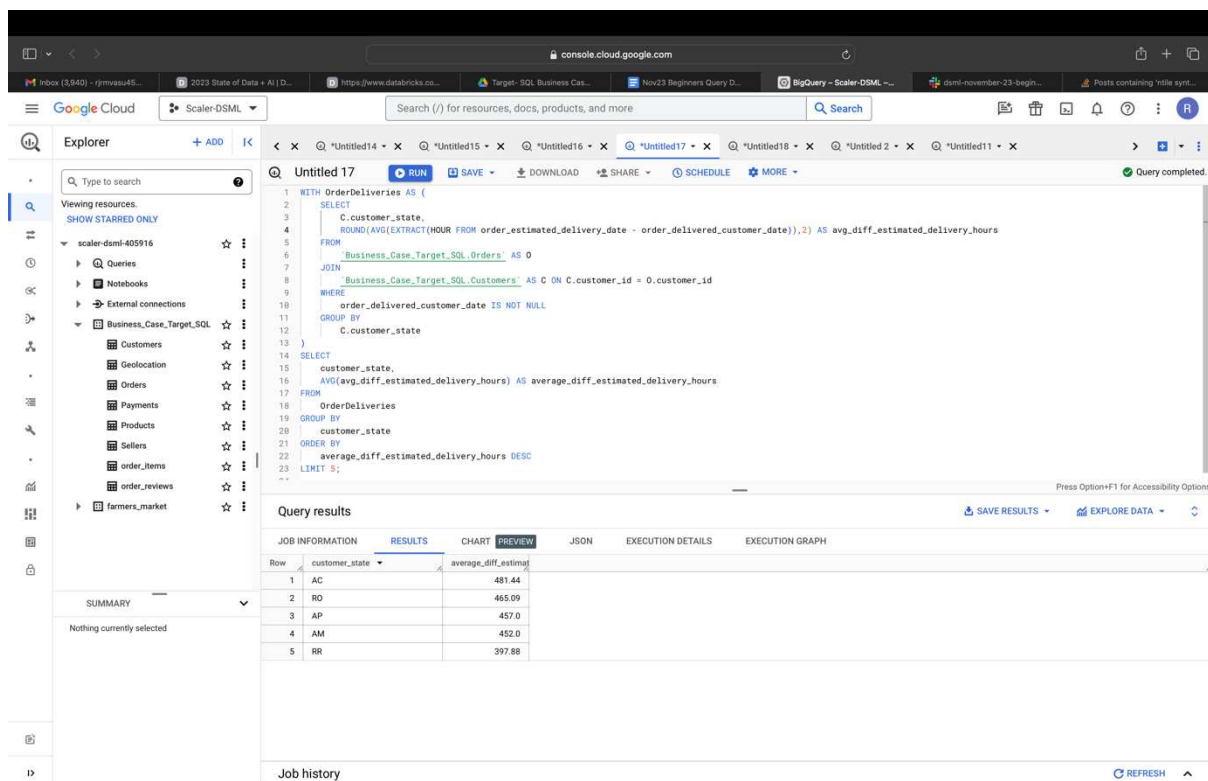
The 'Job history' section is visible at the bottom of the results panel.

Comment: Again SP and RJ with the better infrastructure and connectivity among other states has the lowest(fastest) delivery time.

5.4 Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.

Code:

```
WITH OrderDeliveries AS (  
  SELECT  
    C.customer_state,  
    AVG(EXTRACT(HOUR FROM order_delivered_customer_date - order_purchase_timestamp)) AS  
    avg_delivery_time_hours,  
    AVG(EXTRACT(HOUR FROM order_estimated_delivery_date - order_delivered_customer_date)) AS  
    avg_diff_estimated_delivery_hours  
  FROM  
    `Business_Case_Target_SQL.Orders` AS O  
  JOIN  
    `Business_Case_Target_SQL.Customers` AS C ON C.customer_id = O.customer_id  
  WHERE  
    order_delivered_customer_date IS NOT NULL  
  GROUP BY  
    C.customer_state  
)  
SELECT  
  customer_state,  
  AVG(avg_delivery_time_hours) AS average_delivery_time_hours,  
  AVG(avg_diff_estimated_delivery_hours) AS average_diff_estimated_delivery_hours  
FROM  
  OrderDeliveries  
GROUP BY  
  customer_state  
ORDER BY  
  avg_diff_estimated_delivery_hours DESC  
LIMIT 5;
```

Comment: Top 5 states where the actual delivery time is much ahead of the estimated delivery date/time.

6. Analysis based on the payments:

6.1 Find the month on month no. of orders placed using different payment types.

Code:

SELECT

EXTRACT(MONTH FROM order_purchase_timestamp) AS month,

P.payment_type,

COUNT(*) AS order_count

FROM

`Business_Case_Target_SQL.Payments` as P

JOIN

`Business_Case_Target_SQL.Orders` as O on O.order_id = P.order_id

GROUP BY

month,

P.payment_type

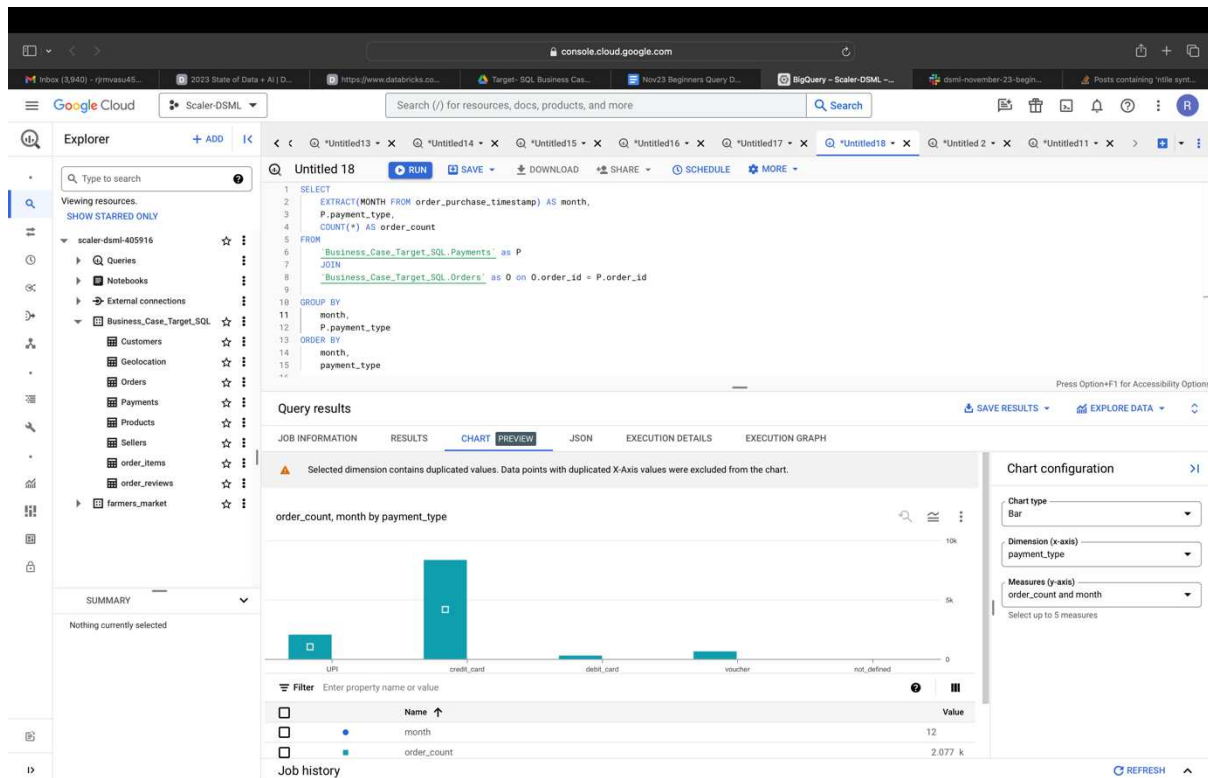
ORDER BY

month,

payment_type

The screenshot shows the Google Cloud Console interface. On the left, the Explorer pane displays the project structure, including a dataset named 'Business_Case_Target_SQL' with tables like 'Orders', 'Payments', 'Customers', etc. The main editor shows a SQL query titled 'Untitled 18' that extracts the month from the purchase timestamp, joins the Payments and Orders tables, and counts the number of orders for each month and payment type. Below the query editor, the 'Query results' section is visible, showing a table with 14 rows of data. The table has columns for 'month', 'payment_type', and 'order_count'. The results show data for months 1 through 4, with various payment types like UPI, credit_card, debit_card, and voucher. At the bottom, there is a 'Job history' section with a 'REFRESH' button.

Row	month	payment_type	order_count
1	1	UPI	1715
2	1	credit_card	6103
3	1	debit_card	118
4	1	voucher	477
5	2	UPI	1723
6	2	credit_card	6609
7	2	debit_card	82
8	2	voucher	424
9	3	UPI	1942
10	3	credit_card	7707
11	3	debit_card	109
12	3	voucher	591
13	4	UPI	1783
14	4	credit_card	7301



COMMENT: Overall Credit Card payment type is the most used payment method by customers to place orders.

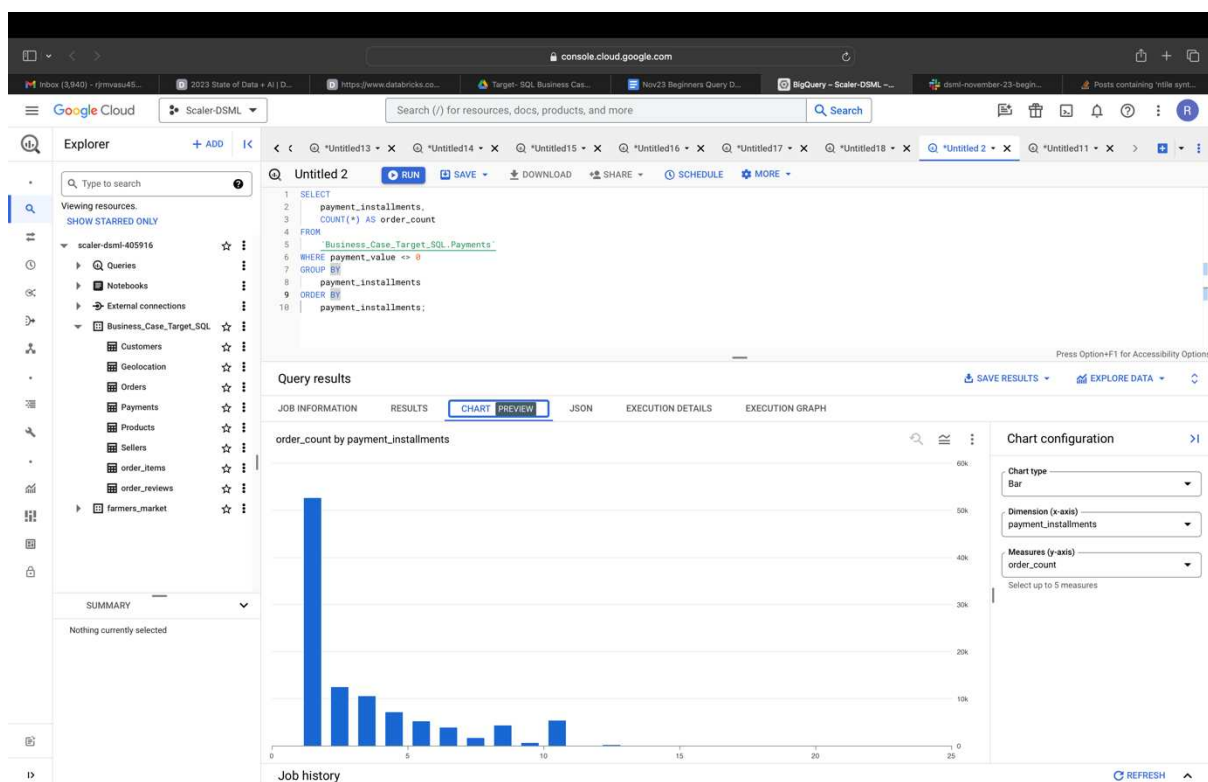
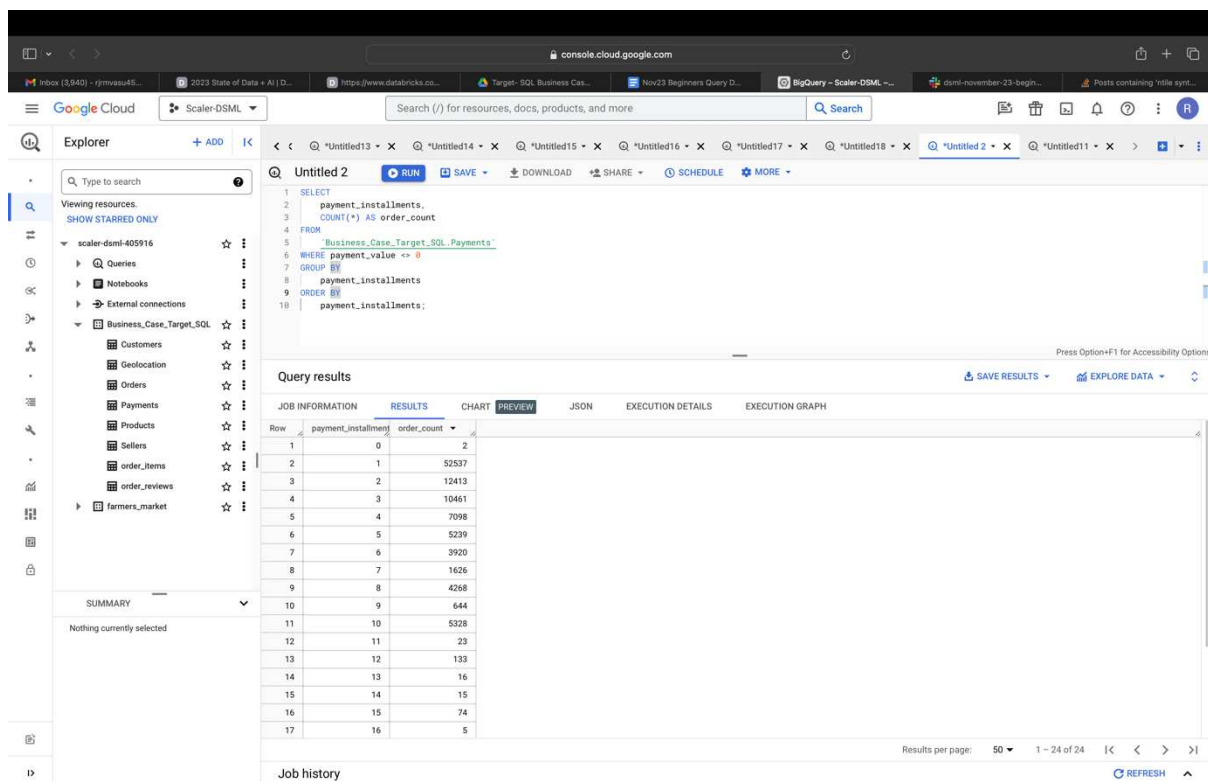
6.2 Find the no. of orders placed on the basis of the payment instalments that have been paid.

Code:

```

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

```



Comment: More than 50% of the orders the payments were made in a single payment.