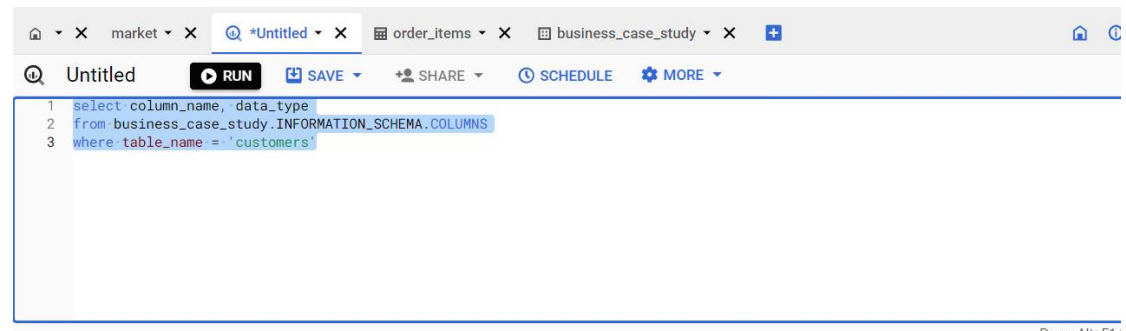


## BUSINESS CASE STUDY:

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

A. Data type of all columns in the “customers” table.

```
select column_name, data_type
from business_case_study.INFORMATION_SCHEMA.COLUMNS
where table_name = 'customers'
```

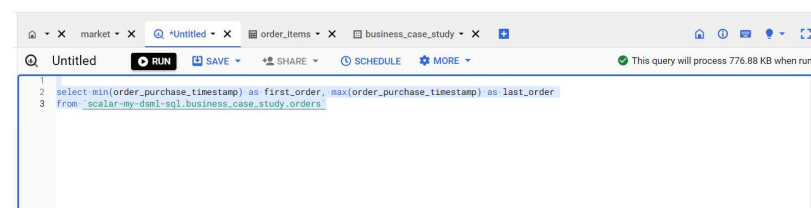


Query results

Row	column_name	data_type
1	customer_id	STRING
2	customer_unique_id	STRING
3	customer_zip_code_prefix	INT64
4	customer_city	STRING
5	customer_state	STRING

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

```
select min(order_purchase_timestamp) as first_order, max(order_purchase_timestamp)
as last_order
from `scalar-my-dsml-sql.business_case_study.orders`
```



Query results

Row	first_order	last_order
1	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC

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

```

select distinct ord.order_purchase_timestamp, concat(cus.customer_city, ' ', cus.customer_state) as
cities_state, count(ord.order_id)
from `scalar-my-dsml-sql.business_case_study.orders` ord join
`scalar-my-dsml-sql.business_case_study.customers` cus
on ord.customer_id = cus.customer_id
where order_purchase_timestamp between '2016-09-04 21:15:19 UTC' and '2018-10-17 17:30:18 UTC'
group by 1,2
order by 3 desc

```

The screenshot shows a SQL query editor with a toolbar at the top containing icons for home, search, market, and tabs for 'Untitled', 'order\_items', and 'business\_case\_study'. The 'Untitled' tab is active, showing the SQL query. Below the query, there are buttons for 'RUN', 'SAVE', 'SHARE', 'SCHEDULE', and 'MORE'. A green checkmark and the text 'Query completed' are visible in the top right corner of the editor area.

Query results SAVE RESULTS EXPLORE DATA

JOB INFORMATION	RESULTS	JSON	EXECUTION DETAILS	CHART	PREVIEW	EXECUTION GRAPH
Row	order_purchase_timestamp	cities_state	order_count			
1	2018-06-01 13:39:44 UTC	salvador BA	3			
2	2018-03-31 15:08:21 UTC	belo horizonte MG	3			
3	2017-12-01 14:38:00 UTC	francisco beltrao PR	2			
4	2017-01-08 19:27:22 UTC	rio de janeiro RJ	2			
5	2017-11-23 22:44:09 UTC	sao goncalo RJ	2			
6	2017-09-18 22:16:48 UTC	lajoticabal SP	2			

## II. In-depth Exploration:

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

```

select distinct format_datetime('%m', order_purchase_timestamp) as order_month,
count(order_id) as order_count
from `scalar-my-dsml-sql.business_case_study.orders`
group by 1
order by order_count, order_month

```

The screenshot shows a SQL query editor with a toolbar at the top. The 'Untitled' tab is active, showing the SQL query. Below the query, there are buttons for 'RUN', 'SAVE', 'SHARE', 'SCHEDULE', and 'MORE'. A green checkmark and the text 'Query completed' are visible in the top right corner of the editor area.

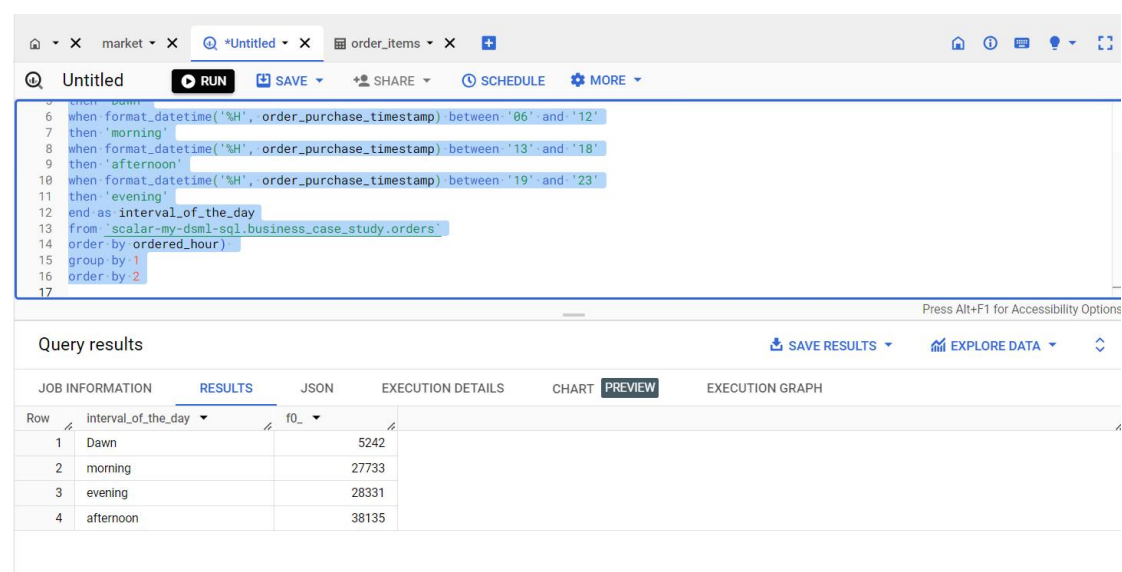
Query results SAVE RESULTS EXPLORE DATA

JOB INFORMATION	RESULTS	JSON	EXECUTION DETAILS	CHART	PREVIEW	EXECUTION GRAPH
Row	order_month	order_count				
1	09	4305				
2	10	4959				
3	12	5674				
4	11	7544				
5	01	8069				
6	02	8508				

C.)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

```
select interval_of_the_day, count(interval_of_the_day)
from (select order_purchase_timestamp, format_datetime('%H',
order_purchase_timestamp) ordered_hour,
case
when format_datetime('%H', order_purchase_timestamp) between '00' and '06'
then 'Dawn'
when format_datetime('%H', order_purchase_timestamp) between '06' and '12'
then 'morning'
when format_datetime('%H', order_purchase_timestamp) between '13' and '18'
then 'afternoon'
when format_datetime('%H', order_purchase_timestamp) between '19' and '23'
then 'evening'
end as interval_of_the_day
from `scalar-my-dsml-sql.business_case_study.orders`
order by ordered_hour)
group by 1
order by 2
```



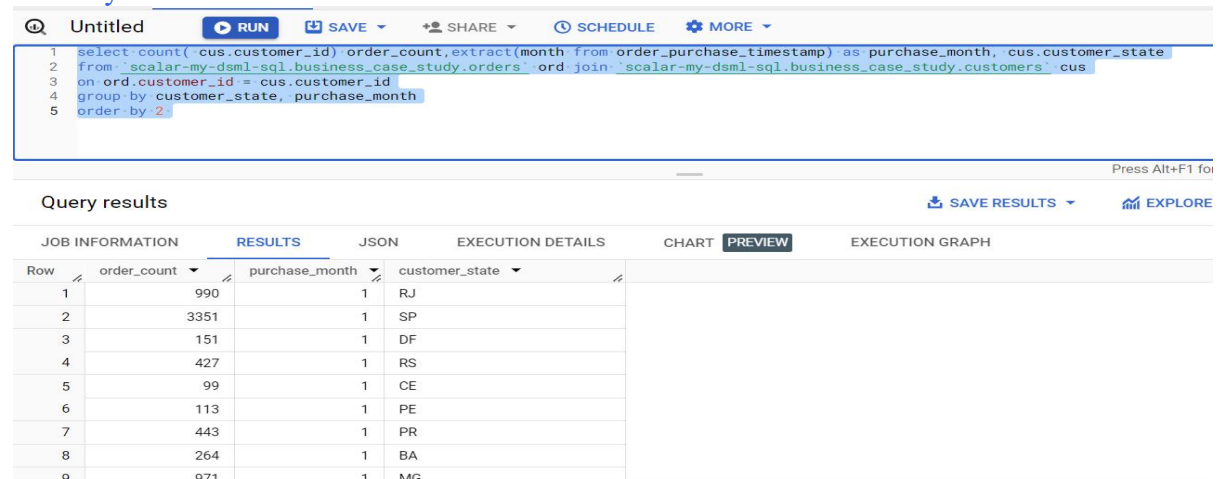
The screenshot shows a SQL query editor with a query that categorizes orders by time of day and counts them. Below the editor, the 'Query results' section displays a table with the following data:

Row	interval_of_the_day	count
1	Dawn	5242
2	morning	27733
3	evening	28331
4	afternoon	38135

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

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

```
select count( cus.customer_id) order_count,extract(month from
order_purchase_timestamp) as purchase_month, cus.customer_state
from `scalar-my-dsml-sql.business_case_study.orders` ord join `scalar-my-dsml-
sql.business_case_study.customers` cus
on ord.customer_id = cus.customer_id
group by customer_state, purchase_month
order by 2
```



The screenshot shows a SQL query editor with the following query:

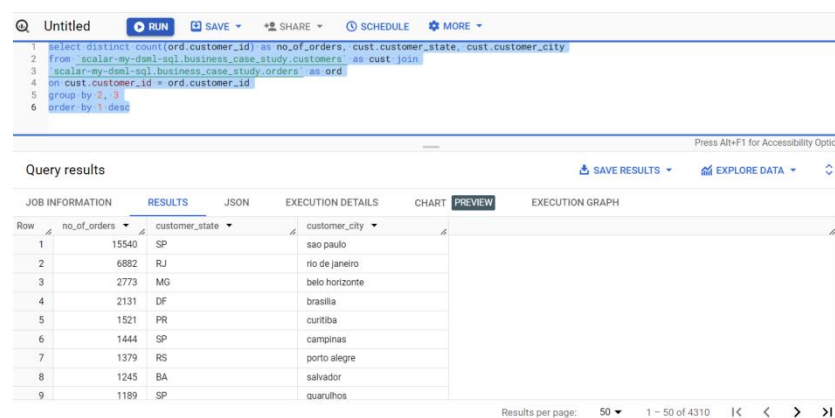
```
1 select count( cus.customer_id) order_count,extract(month from order_purchase_timestamp) as purchase_month, cus.customer_state
2 from `scalar-my-dsml-sql.business_case_study.orders` ord join `scalar-my-dsml-sql.business_case_study.customers` cus
3 on ord.customer_id = cus.customer_id
4 group by customer_state, purchase_month
5 order by 2
```

Below the editor, the 'Query results' section displays a table with the following data:

Row	order_count	purchase_month	customer_state
1	990	1	RJ
2	3351	1	SP
3	151	1	DF
4	427	1	RS
5	99	1	CE
6	113	1	PE
7	443	1	PR
8	264	1	BA
9	971	1	MG

--B.) How are the customers distributed across all the states?

```
select distinct count(ord.customer_id) as no_of_orders, cust.customer_state,
cust.customer_city
from `scalar-my-dsml-sql.business_case_study.customers` as cust join
`scalar-my-dsml-sql.business_case_study.orders` as ord
on cust.customer_id = ord.customer_id
group by 2, 3
order by 1 desc
```



The screenshot shows a SQL query editor with the following query:

```
1 select distinct count(ord.customer_id) as no_of_orders, cust.customer_state, cust.customer_city
2 from `scalar-my-dsml-sql.business_case_study.customers` as cust join
3 `scalar-my-dsml-sql.business_case_study.orders` as ord
4 on cust.customer_id = ord.customer_id
5 group by 2, 3
6 order by 1 desc
```

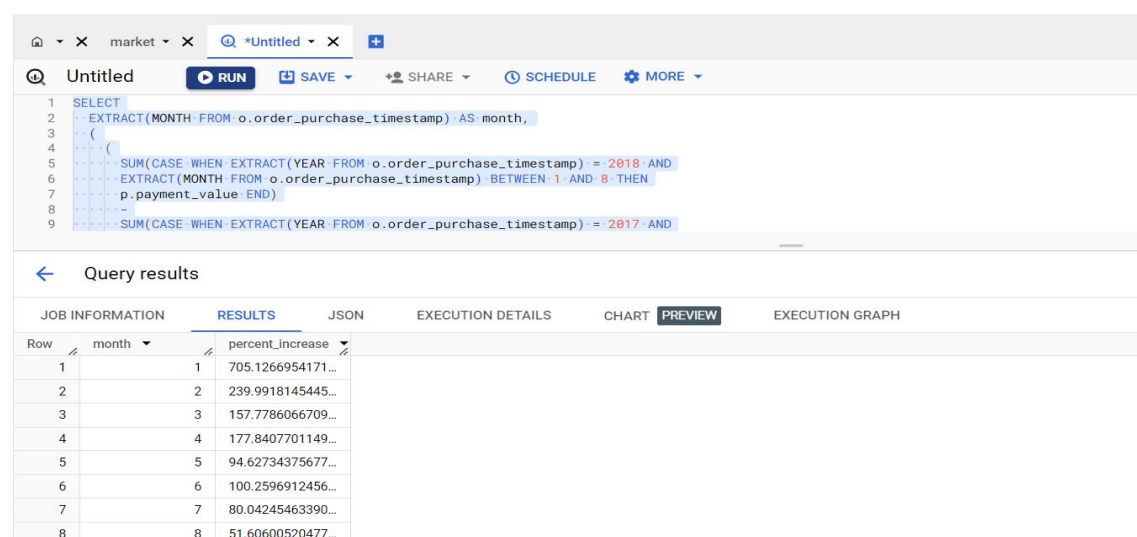
Below the editor, the 'Query results' section displays a table with the following data:

Row	no_of_orders	customer_state	customer_city
1	15540	SP	sao paulo
2	6882	RJ	rio de janeiro
3	2773	MG	belo horizonte
4	2131	DF	brasilia
5	1521	PR	curitiba
6	1444	SP	campinas
7	1379	RS	porto alegre
8	1245	BA	salvador
9	1189	SP	quarunhos

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

A. Get the % increase in the cost of orders from year 2017 to 2018 (include months between Jan to Aug only).

```
SELECT
  EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
  (
    (
      SUM(CASE WHEN EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
AND
      EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8 THEN
      p.payment_value END)
    -
      SUM(CASE WHEN EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2017
AND
      EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8 THEN
      p.payment_value END)
    )
    /
    SUM(CASE WHEN EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2017
AND
    EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8 THEN
    p.payment_value END)
  ) * 100 AS percent_increase
FROM
  `scalar-my-dsml-sql.business_case_study.orders` o
JOIN
  `scalar-my-dsml-sql.business_case_study.payments` p ON o.order_id = p.order_id
WHERE
  EXTRACT(YEAR FROM o.order_purchase_timestamp) IN (2017, 2018) AND
  EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8
GROUP BY 1
ORDER BY 1;
```

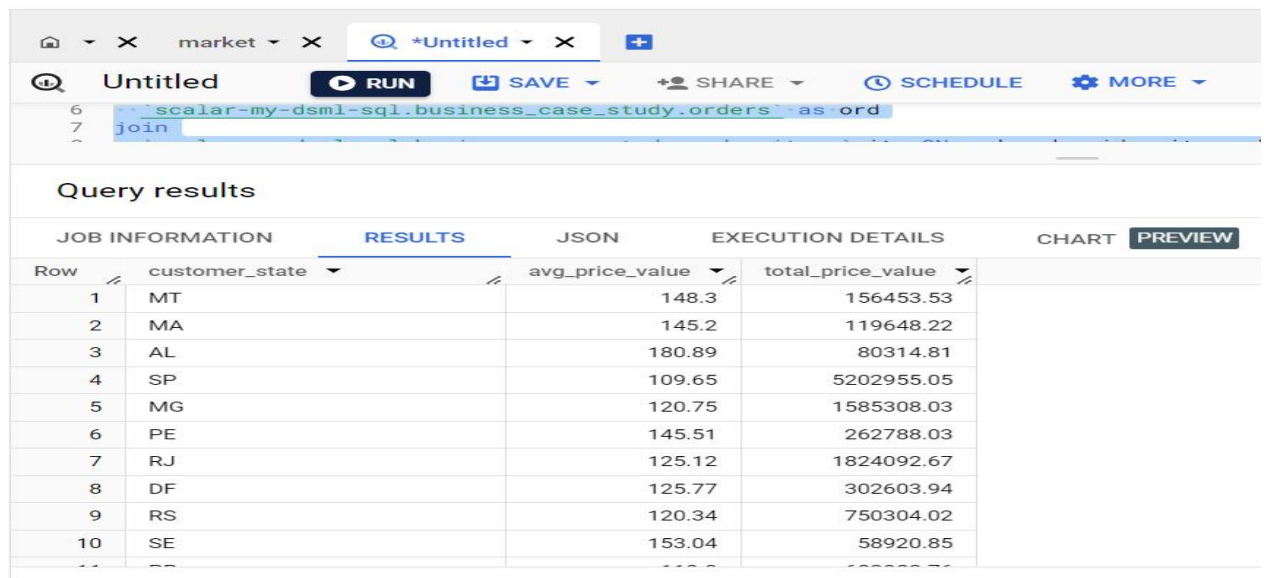


The screenshot shows a SQL query editor interface with a query window and a results window. The query window contains the SQL code for calculating the percentage increase in order costs from 2017 to 2018 for months January through August. The results window displays a table with three columns: Row, month, and percent\_increase. The data shows the percentage increase for each month from January to August.

Row	month	percent_increase
1	1	705.1266954171...
2	2	239.9918145445...
3	3	157.7786066709...
4	4	177.8407701149...
5	5	94.62734375677...
6	6	100.2596912456...
7	7	80.04245463390...
8	8	51.60600520477...

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

```
select
  cus.customer_state,
  round(avg(ite.price), 2) AS avg_price_value,
  round(sum(ite.price), 2) AS total_price_value
from
  `scalar-my-dsml-sql.business_case_study.orders` as ord
join
  `scalar-my-dsml-sql.business_case_study.order_items` ite ON ord.order_id =
ite.order_id
join
  `scalar-my-dsml-sql.business_case_study.customers` cus ON ord.customer_id =
cus.customer_id
group by
  cus.customer_state;
```



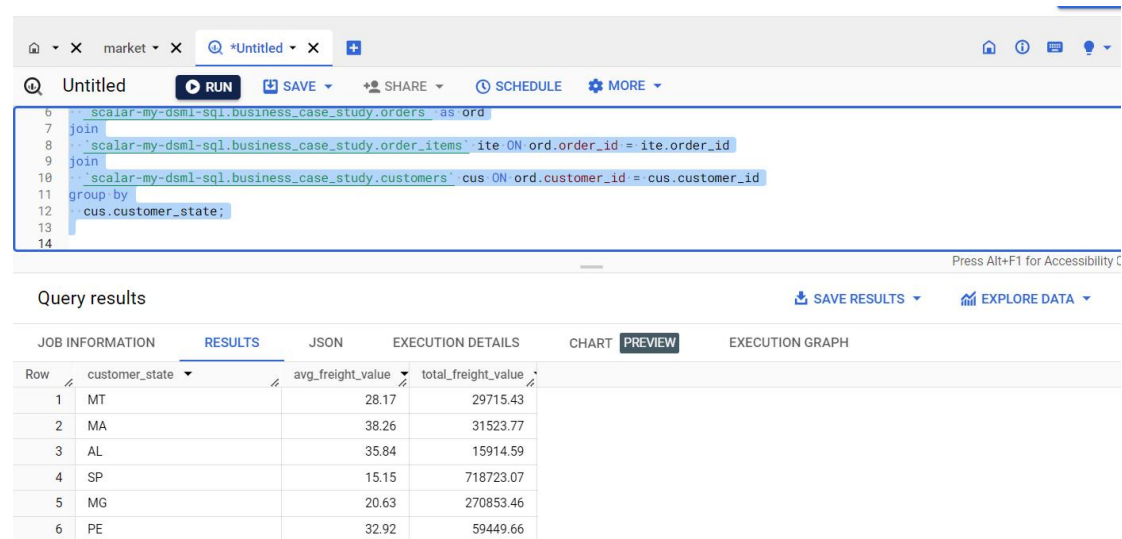
The screenshot shows a SQL query editor with a query window titled 'Untitled' containing the SQL code for task B. Below the query window, the 'Query results' section is displayed, showing a table with 4 columns: 'customer\_state', 'avg\_price\_value', and 'total\_price\_value'. The table contains 10 rows of data, one for each state (MT, MA, AL, SP, MG, PE, RJ, DF, RS, SE).

Row	customer_state	avg_price_value	total_price_value
1	MT	148.3	156453.53
2	MA	145.2	119648.22
3	AL	180.89	80314.81
4	SP	109.65	5202955.05
5	MG	120.75	1585308.03
6	PE	145.51	262788.03
7	RJ	125.12	1824092.67
8	DF	125.77	302603.94
9	RS	120.34	750304.02
10	SE	153.04	58920.85

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

```
select
  cus.customer_state,
  round(avg(ite.freight_value), 2) AS avg_freight_value,
  round(sum(ite.freight_value), 2) AS total_freight_value
from
  `scalar-my-dsml-sql.business_case_study.orders` as ord
join
  `scalar-my-dsml-sql.business_case_study.order_items` ite ON ord.order_id =
ite.order_id
join
  `scalar-my-dsml-sql.business_case_study.customers` cus ON ord.customer_id =
cus.customer_id
```

group by  
cus.customer\_state;



The screenshot shows a SQL query editor with a query that joins three tables: orders, order\_items, and customers. The query groups the results by customer\_state. Below the query editor, the 'Query results' section is displayed, showing a table with 6 rows and 4 columns: Row, customer\_state, avg\_freight\_value, and total\_freight\_value.

Row	customer_state	avg_freight_value	total_freight_value
1	MT	28.17	29715.43
2	MA	38.26	31523.77
3	AL	35.84	15914.59
4	SP	15.15	718723.07
5	MG	20.63	270853.46
6	PE	32.92	59449.66

V. Analysis based on sales, freight and delivery time.

A. Find the no. of days taken to deliver each order from the order's purchase date as delivery time.

Also, calculate the difference (in days) between the estimated & actual delivery date of an order.

Do this in a single query.

```
select
order_id,
date_diff(order_delivered_customer_date, order_purchase_timestamp, DAY)
as order_delivered_in_days,
date_diff(order_estimated_delivery_date, order_delivered_customer_date, DAY)
as diff_estimated_delivery
from `scalar-my-dsml-sql.business_case_study.orders`
where date_diff(order_estimated_delivery_date, order_delivered_customer_date,
DAY) is not NULL
order by 2
```



market x \*Untitled x

Untitled RUN SAVE SHARE SCHEDULE MORE

```

1 select
2   order_id,
3   date_diff(order_delivered_customer_date, order_purchase_timestamp, DAY)
4   as order_delivered_in_days,
5   date_diff(order_estimated_delivery_date, order_delivered_customer_date, DAY)
6   as diff_estimated_delivery
7 from `scalar-my-dsml-sql.business_case_study.orders`
8 where date_diff(order_estimated_delivery_date, order_delivered_customer_date, DAY) is not NULL

```

Query results SAVE RESULTS EXPLORE DATA

Row	order_id	order_delivered_in_d	diff_estimated_deliv
1	e65f1eeee1f52024ad1dcd034...	0	9
2	bb5a519e352b45b714192a02f...	0	25
3	434cecee7d1a65fc65358a632...	0	19
4	d3ca7b82c922817b06e5ca211...	0	11
5	1d893dd7ca5f77ebf5f9f0d20...	0	10
6	d5fbedc85190ba88580d6f82...	0	7
7	79e324907160caea526fd8b94...	0	8
8	38c1e3d4ed6a13cd0cf612d4c...	0	16

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

```

select cust.customer_state, round(avg(ord_it.freight_value), 2) as
top_5_avg_freight_value
from `scalar-my-dsml-sql.business_case_study.customers` cust join
`scalar-my-dsml-sql.business_case_study.orders` ord
on cust.customer_id = ord.customer_id
join
`scalar-my-dsml-sql.business_case_study.order_items` ord_it
on ord.order_id = ord_it.order_id
group by cust.customer_state
order by top_5_avg_freight_value desc
limit 5

```

market x \*Untitled x order\_items x

Untitled RUN SAVE SHARE SCHEDULE MORE Query completed

```

1 (select cust.customer_state, round(avg(ord_it.freight_value), 2) as top_5_avg_freight_value
2 from `scalar-my-dsml-sql.business_case_study.customers` cust join
3 `scalar-my-dsml-sql.business_case_study.orders` ord
4 on cust.customer_id = ord.customer_id
5 join
6 `scalar-my-dsml-sql.business_case_study.order_items` ord_it
7 on ord.order_id = ord_it.order_id
8 group by cust.customer_state
9 order by top 5 avg freight value desc

```

Query results SAVE RESULTS EXPLORE DATA

Row	customer_state	top_5_avg_freight_va
1	RR	42.98
2	PB	42.72
3	RO	41.07
4	AC	40.07
5	PI	39.15

```

select cust.customer_state, round(avg(ord_it.freight_value), 2) as
bottom_5_avg_freight_value
from `scalar-my-dsml-sql.business_case_study.customers` cust join
`scalar-my-dsml-sql.business_case_study.orders` ord
on cust.customer_id = ord.customer_id
join
`scalar-my-dsml-sql.business_case_study.order_items` ord_it

```



```

on ord.order_id = ord_it.order_id
group by cust.customer_state
order by bottom_5_avg_freight_value
limit 5

```

Query results

Row	customer_state	bottom_5_avg_freigh
1	SP	15.15
2	PR	20.53
3	MG	20.63
4	RJ	20.96
5	DF	21.04

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

```

SELECT
  cust.customer_state,
  ROUND(AVG(DATE_DIFF(order_delivered_customer_date,
order_purchase_timestamp, DAY)), 2)
  AS highest_avg_delivery_time,
FROM
  `scalar-my-dsml-sql.business_case_study.customers` cust
JOIN
  `scalar-my-dsml-sql.business_case_study.orders` ord ON ord.customer_id =
cust.customer_id
WHERE
  DATE_DIFF(order_purchase_timestamp, order_delivered_customer_date, DAY) IS
NOT NULL

GROUP BY cust.customer_state
ORDER BY 2 desc
limit 5

```

Untitled		<a href="#">RUN</a> <a href="#">SAVE</a> <a href="#">SHARE</a> <a href="#">SCHEDULE</a> <a href="#">MORE</a>
<pre> 4 AS highest_avg_delivery_time, 5 FROM 6 `scalar-my-dsml-sql.business_case_study.customers` cust 7 JOIN 8 `scalar-my-dsml-sql.business_case_study.orders` ord ON ord.customer_id = cust.customer_id 9 WHERE 10 DATE_DIFF(order_purchase_timestamp, order_delivered_customer_date, DAY) IS NOT NULL 11 12 GROUP BY cust.customer_state 13 ORDER BY 2 desc 14 limit 5 15 </pre>		Press Alt+F1 for Accessibility Options
Query results		
<a href="#">SAVE RESULTS</a> <a href="#">EXPLORE DATA</a>		
<a href="#">JOB INFORMATION</a> <a href="#">RESULTS</a> <a href="#">JSON</a> <a href="#">EXECUTION DETAILS</a> <a href="#">CHART</a> <a href="#">PREVIEW</a> <a href="#">EXECUTION GRAPH</a>		
Row	customer_state	highest_avg_delivery
1	RR	28.98
2	AP	26.73
3	AM	25.99
4	AL	24.04
5	PA	23.32

```

SELECT
    cust.customer_state,
    ROUND(AVG(DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY)), 2)
    AS lowest_avg_delivery_time,
FROM
    `scalar-my-dsml-sql.business_case_study.customers` cust
JOIN
    `scalar-my-dsml-sql.business_case_study.orders` ord ON ord.customer_id = cust.customer_id
WHERE
    DATE_DIFF(order_purchase_timestamp, order_delivered_customer_date, DAY) IS NOT NULL

GROUP BY cust.customer_state
ORDER BY 2
limit 5

```

Untitled		<a href="#">RUN</a> <a href="#">SAVE</a> <a href="#">SHARE</a> <a href="#">SCHEDULE</a> <a href="#">MORE</a>
<pre> 1 SELECT 2     cust.customer_state, 3     ROUND(AVG(DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY)), 2) 4     AS lowest_avg_delivery_time, 5 FROM 6     `scalar-my-dsml-sql.business_case_study.customers` cust 7 JOIN 8     `scalar-my-dsml-sql.business_case_study.orders` ord ON ord.customer_id = cust.customer_id 9 WHERE 10    DATE_DIFF(order_purchase_timestamp, order_delivered_customer_date, DAY) IS NOT NULL 11 12 GROUP BY cust.customer_state </pre>		Press Alt+F1 for Accessibility Options
Query results		
<a href="#">SAVE RESULTS</a> <a href="#">EXPLORE DATA</a>		
<a href="#">JOB INFORMATION</a> <a href="#">RESULTS</a> <a href="#">JSON</a> <a href="#">EXECUTION DETAILS</a> <a href="#">CHART</a> <a href="#">PREVIEW</a> <a href="#">EXECUTION GRAPH</a>		
Row	customer_state	lowest_avg_delivery
1	SP	8.3
2	PR	11.53
3	MG	11.54
4	DF	12.51
5	SC	14.48

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

```

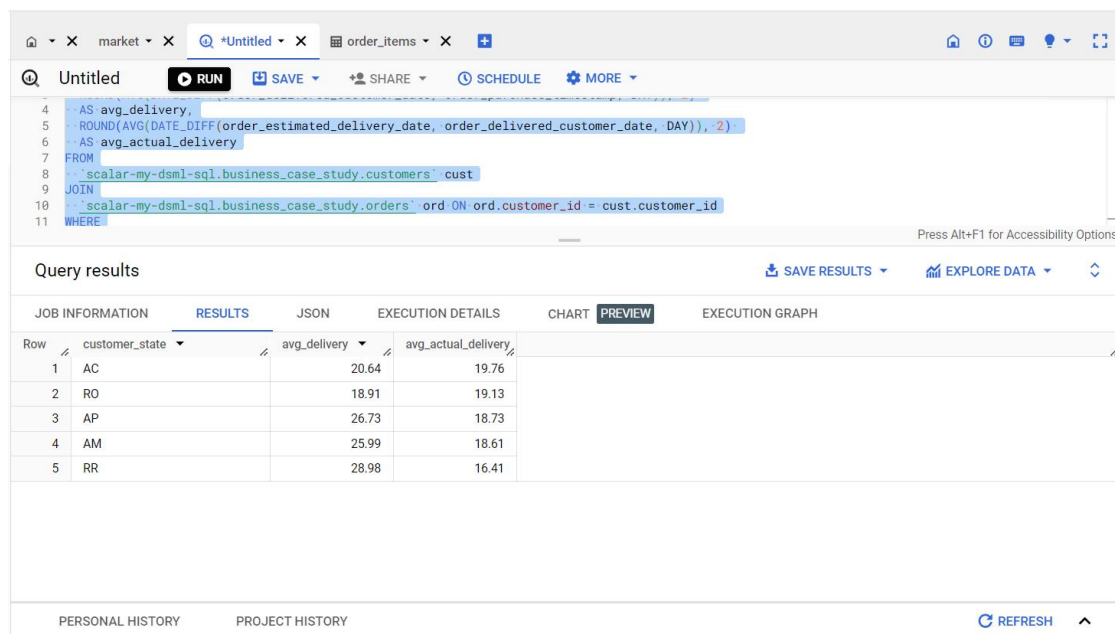
SELECT
    cust.customer_state,
    ROUND(AVG(DATE_DIFF(order_delivered_customer_date,
order_purchase_timestamp, DAY)), 2)
    AS avg_delivery,
    ROUND(AVG(DATE_DIFF(order_estimated_delivery_date,
order_delivered_customer_date, DAY)), 2)
    AS avg_actual_delivery

```

```

FROM
  `scalar-my-dsml-sql.business_case_study.customers` cust
JOIN
  `scalar-my-dsml-sql.business_case_study.orders` ord ON ord.customer_id =
cust.customer_id
WHERE
  DATE_DIFF(order_purchase_timestamp, order_delivered_customer_date, DAY) IS
NOT NULL
AND
  DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY)
IS NOT NULL
GROUP BY
  cust.customer_state
ORDER BY 3 desc
limit 5

```



The screenshot shows a SQL query editor with a query that calculates the average delivery time for different customer states. The query is as follows:

```

4 AS avg_delivery,
5 ROUND(AVG(DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY)), 2)
6 AS avg_actual_delivery
7 FROM
8 `scalar-my-dsml-sql.business_case_study.customers` cust
9 JOIN
10 `scalar-my-dsml-sql.business_case_study.orders` ord ON ord.customer_id = cust.customer_id
11 WHERE

```

Below the query editor, the 'Query results' section displays a table with the following data:

Row	customer_state	avg_delivery	avg_actual_delivery
1	AC	20.64	19.76
2	RO	18.91	19.13
3	AP	26.73	18.73
4	AM	25.99	18.61
5	RR	28.98	16.41

VI. Analysis based on the payments:

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

```

SELECT
  pay.payment_type,
  EXTRACT(MONTH FROM ord.order_purchase_timestamp) AS month,
  COUNT(DISTINCT ord.order_id) AS order_count
FROM
  `scalar-my-dsml-sql.business_case_study.orders` ord
JOIN
  `scalar-my-dsml-sql.business_case_study.payments` pay
ON
  ord.order_id = pay.order_id

```

GROUP BY

1, 2

ORDER BY

1, 2;

Untitled RUN SAVE SHARE SCHEDULE MORE

```
1 SELECT
2   pay.payment_type,
3   EXTRACT(MONTH FROM ord.order_purchase_timestamp) AS month,
4   COUNT(DISTINCT ord.order_id) AS order_count
5 FROM
6   `scalar-my-dsml-sql.business_case_study.orders` ord
7 JOIN
8   `scalar-my-dsml-sql.business_case_study.payments` pay
9 ON
10  ord.order_id = pay.order_id
11 GROUP BY
12   1, 2
13 ORDER BY
14   1, 2;
```

Query results SAVE RESULTS EXPLORE DATA

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	CHART	PREVIEW	EXECUTION GRAPH
Row	payment_type	month	order_count				
1	UPI		1	1715			
2	UPI		2	1723			
3	UPI		3	1942			
4	UPI		4	1783			
5	UPI		5	2035			

Results per page: 50 1 - 50 of 50

B. Find the no. of orders placed on the basis of the payment installments that have been paid.

SELECT

paye.payment\_installments,  
COUNT(ord.order\_id) AS order\_count

FROM

`scalar-my-dsml-sql.business\_case\_study.orders` ord

JOIN

`scalar-my-dsml-sql.business\_case\_study.payments` paye

ON

ord.order\_id = paye.order\_id

WHERE

ord.order\_status != 'paid' and paye.payment\_installments >= 1

GROUP BY

1

ORDER BY 1

market

Untitled

order\_items

RUN

SAVE

SHARE

SCHEDULE

MORE

```
1 SELECT
2   paye.payment_installments,
3   COUNT(ord.order_id) AS order_count
4 FROM
5   scalar-my-dsm1-sql.business_case_study.orders ord
6 JOIN
7   scalar-my-dsm1-sql.business_case_study.payments paye
8 ON
9   ord.order_id = paye.order_id
10 WHERE
```

Press Alt+F1 for Accessibility Options

Query results

SAVE RESULTS

EXPLORE DATA

JOB INFORMATION

RESULTS

JSON

EXECUTION DETAILS

CHART

PREVIEW

EXECUTION GRAPH

Row	payment_installment	order_count
1	1	52546
2	2	12413
3	3	10461
4	4	7098
5	5	5239
6	6	3920
7	7	1626

Results per page: 50 1 - 23 of 23