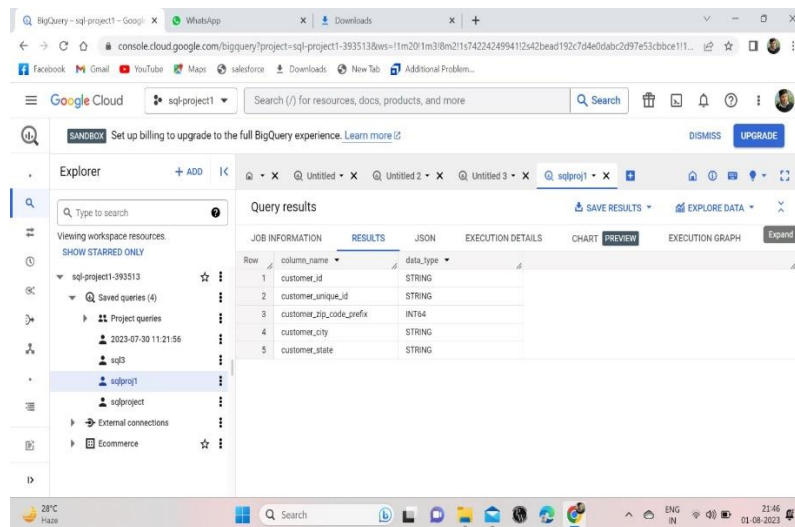


- I. Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset.

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

Ans: select column_name,data_type

```
from `sql-project1-393513.Ecommerce.INFORMATION_SCHEMA.COLUMNS`  
where table_name="customers";
```



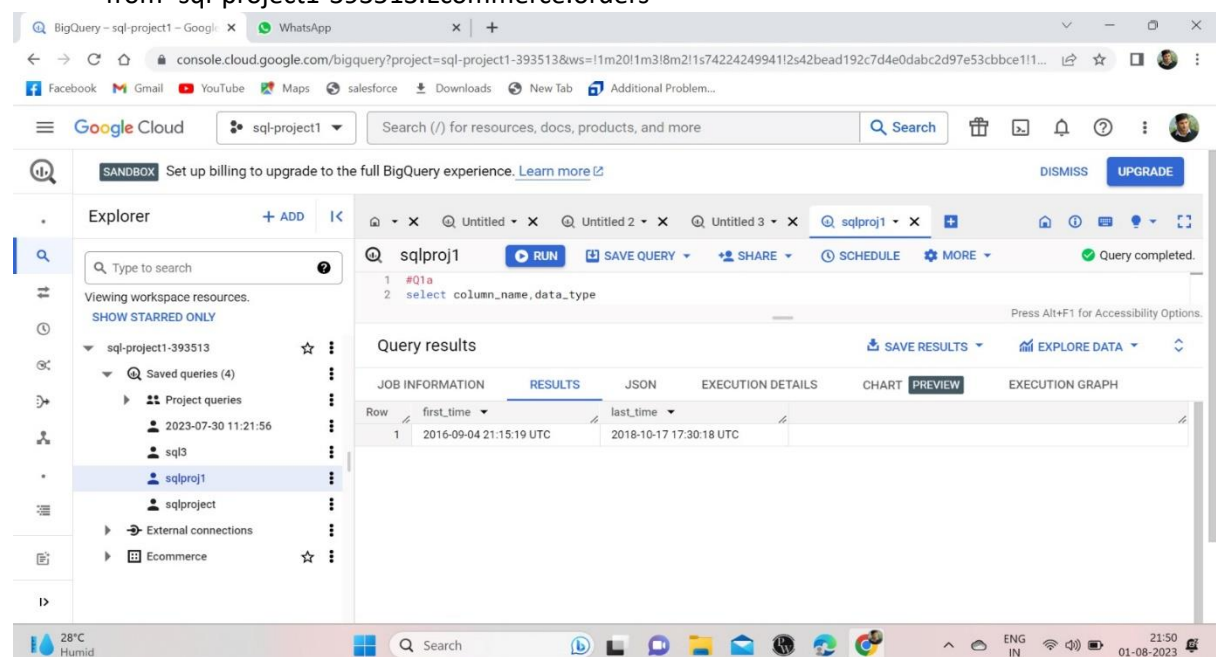
Inference : Here we use information schema to display data types of the customer table by selecting data type column to display result. Mostly seen data type in the table is int and char.

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

ANS: select min(order_purchase_timestamp) as first_time,

max(order_purchase_timestamp) as last_time

```
from `sql-project1-393513.Ecommerce.orders`
```

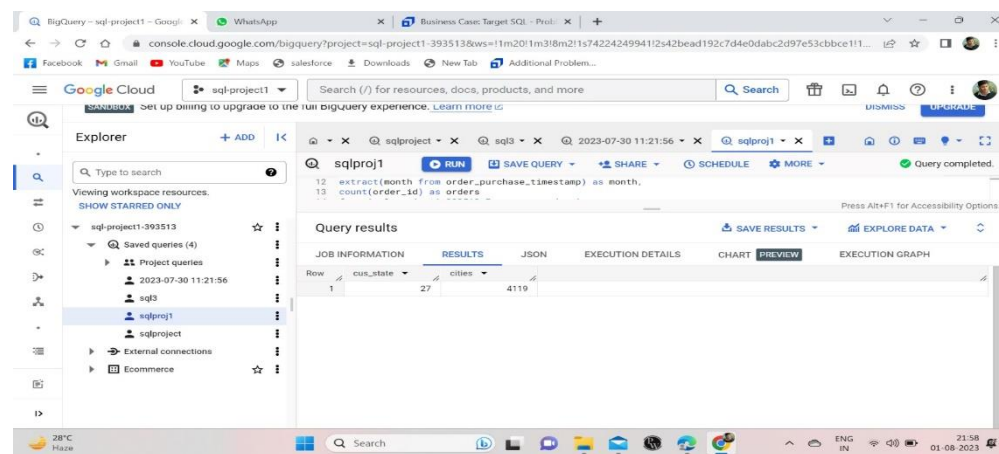


Inference: To find first and last time we use min and max key word of the order_purchase_timestamp column in the orders table.

Min key word gives the least result of the selected column in the table ,max gives the highest result of the selected column in the table

C. Count the number of Cities and States in our dataset.

Ans: select count(distinct customer_state) as cus_state,
count(distinct customer_city) as cities
from
`sql-project1-393513.Ecommerce.orders` o inner join `sql-project1-393513.Ecommerce.customers` c on o.customer_id=c.customer_id;



Inference: To count number of states and cities in the table we use count as a key word and here we use distinct key word to remove duplicates over state and cities. And we need to join orders and customers table because we cannot get the data from single table.

II. In-depth Exploration:

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

Ans select

extract(year from order_purchase_timestamp) as year,
extract(month from order_purchase_timestamp) as month,
count(order_id) as orders
from `sql-project1-393513.Ecommerce.orders`
group by 1,2
order by 1,2 asc;

inference: Here we need to extract year and month from order_purchase_timestamp in order to show the growing trend over past years. We need to group by the year and month column and count number of orders placed.

The screenshot shows the Google Cloud BigQuery console. The Explorer panel on the left shows the project structure. The Query results panel on the right displays a table with the following data:

Row	year	month	orders
1	2016	9	4
2	2016	10	324
3	2016	12	1
4	2017	1	800
5	2017	2	1780
6	2017	3	2682
7	2017	4	2404
8	2017	5	3700
9	2017	6	3245
10	2017	7	4026
11	2017	8	4331
12	2017	9	4285
13	2017	10	4631

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

Ans: select

count(*) as no_of_orders,

extract(month from order_purchase_timestamp) as month

from `sql-project1-393513.Ecommerce.orders`

group by month

order by month;

The screenshot shows the Google Cloud BigQuery console. The Explorer panel on the left shows the project structure. The Query results panel on the right displays a table with the following data:

Row	no_of_orders	month
1	8069	1
2	8308	2
3	9893	3
4	9343	4
5	10573	5
6	9412	6
7	10318	7
8	10843	8
9	4305	9

C. Inference :To find monthly seasonality in terms of the no. of orders being placed we need to extract month only and count number of orders placed using count(*).we need to group by month to find the data monthly

c. 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

Ans: 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'

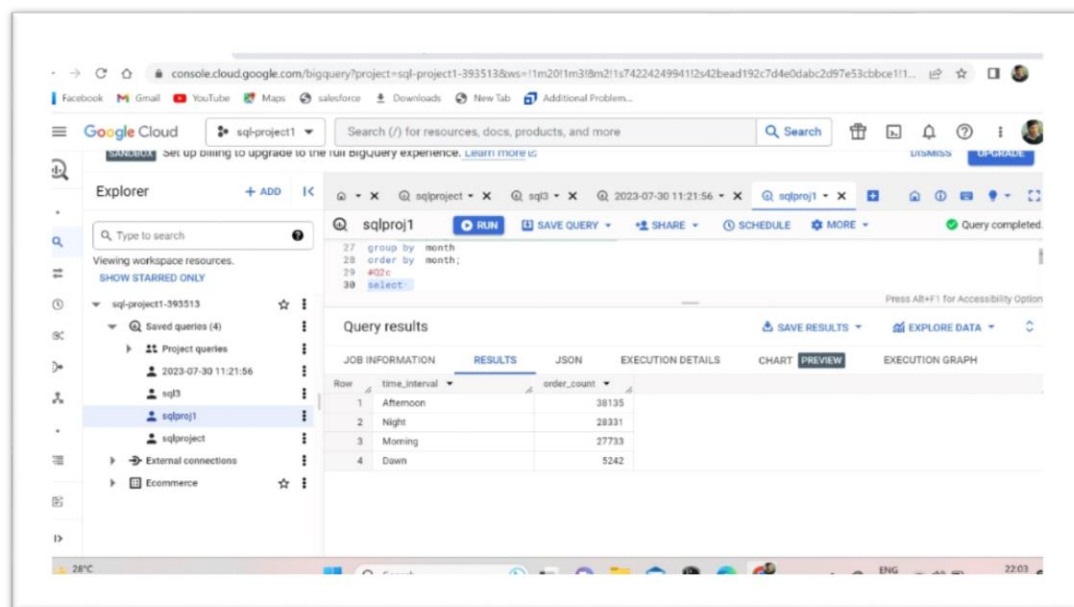
end as time_interval,

count(*) as order_count

from `sql-project1-393513.Ecommerce.orders`

group by time_interval

order by order_count desc;



Inference: Here we use case study it acts like if else statement in the query between the during hours the particular day can we mention such as morning, evening, night and group by the time_intervals finally we need to do order by in desc order.

III. Evolution of E-commerce orders in the Brazil region

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

Ans:

```

select extract( month from o.order_purchase_timestamp) as month,
       c.customer_state,
       count(o.order_id) as no_of_order
from `sql-project1-393513.Ecommerce.orders` o join `sql-project1-393513.Ecommerce.customers` c on o.customer_id=c.customer_id
group by month,c.customer_state
order by month,c.customer_state;

```

Row	month	customer_state	no_of_order
12	1	MS	71
13	1	MT	96
14	1	PA	82
15	1	PB	33
16	1	PE	113
17	1	PI	55
18	1	PR	443
19	1	RJ	990
20	1	RN	51
21	1	RO	23
22	1	RR	2
23	1	RS	427
24	1	SC	345
25	1	SE	74

Inference : Here month on month means jan to dec and count number of orders places in each state so need to do group by state and month.And need to do join operation between orders and customers table

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

```

Ans: select customer_state,
       count(customer_unique_id) as unique_count
from `sql-project1-393513.Ecommerce.customers`
group by customer_state;

```

The screenshot shows the Google Cloud BigQuery console. On the left, the 'Explorer' pane displays the project hierarchy: sql-project1-393513 > Saved queries (4) > Project queries > 2023-07-30 11:21:56 > sqlproj1. The main area shows the 'Query results' for the selected query. The results are displayed in a table with two columns: 'customer_state' and 'unique_count'. The table contains 14 rows of data, representing different US states and their corresponding unique customer counts.

Row	customer_state	unique_count
1	RN	485
2	CE	1336
3	RS	5466
4	SC	3637
5	SP	41746
6	MG	11635
7	BA	3380
8	RJ	12852
9	GO	2020
10	MA	747
11	PE	1652
12	PB	536
13	ES	2033
14	PR	5045

Inference : The customers distributed across all the states here we should select customer state and unique count to display in the result and need to group by customer state column .

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

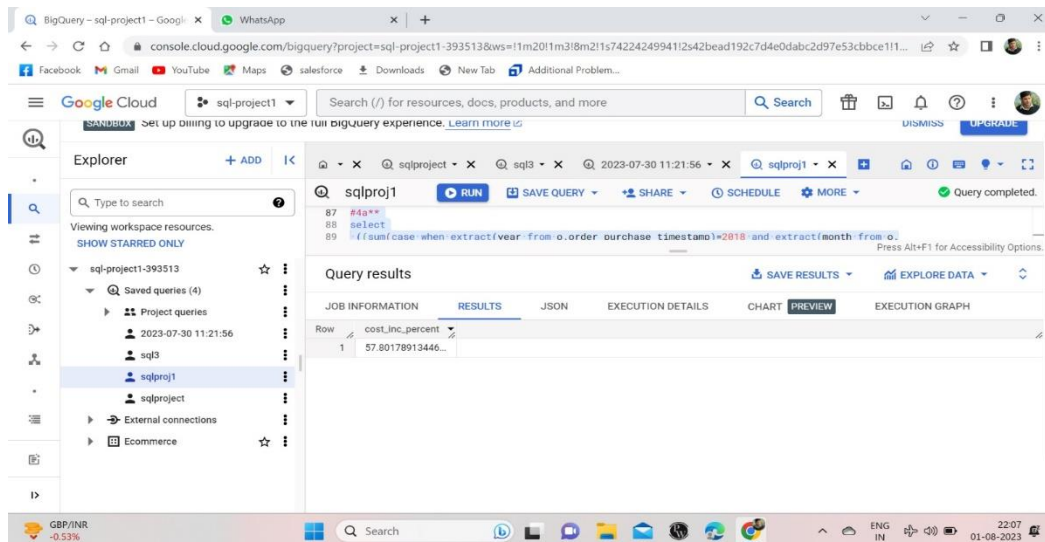
A. Get the % increase in the cost of orders from year 2017 to 2018 (include months between Jan to Aug only).

Ans: select

((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 else 0 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 else 0 end))/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 else 0 end))*100 as cost_inc_percent

from `sql-project1-393513.Ecommerce.payments` p inner join `sql-project1-393513.Ecommerce.orders` o on p.order_id=o.order_id

where extract(year from o.order_purchase_timestamp) between 2017 and 2018 and extract(month from o.order_purchase_timestamp)between 1 and 8;



Inference: To get the percentage increase in the year 2017-2018, here we need to extract year and month where month must lie between 1 and 8 and need to use percentage formula over the filtered data. And we need the help of the payments table over to get the data

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

Ans: select customer_state ,

round(sum(total),2) as total_price,

round(avg(total),2) as avg_price

from (

select c.customer_state, o.order_id, sum(p.payment_value) as total

from `sql-project1-393513.Ecommerce.payments` p join `sql-project1-393513.Ecommerce.orders` o on p.order_id=o.

order_id join `sql-project1-393513.Ecommerce.customers` c on c.customer_id=o.customer_id

group by c.customer_state, o.order_id

) as order_total

group by customer_state;

Row	customer_state	total_price	avg_price
1	BA	616645.82	182.44
2	SP	5998226.96	143.69
3	RJ	2144379.69	166.85
4	MT	187029.29	206.21
5	GO	350092.31	173.31
6	ES	325967.55	160.34
7	RS	890898.54	162.99
8	MG	1872257.26	160.92
9	MA	152523.02	204.18
10	SC	623086.43	171.32
11	PR	811156.38	160.78
12	SE	75246.25	214.99
13	AM	27956.93	188.97

Inference: To get avg and sum of the prices we have aggregation function which were used near the select clause mostly after need to round 2 decimals of obtained output. In this case we need to do join operation of 3 tables customers orders and payments table. finally need to group by the customer state.

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

Ans: select c.customer_state,

round(sum(od.freight_value),2) as totalfreight,

round(avg(od.freight_value),2) as avgfreight

from `sql-project1-393513.Ecommerce.order_items` od join `sql-project1-393513.Ecommerce.orders` o on o.order_id=od.order_id join `sql-project1-393513.Ecommerce.customers` c on c.customer_id=o.customer_id

group by c.customer_state;

Row	customer_state	totalfreight	avgfreight
1	SP	718723.07	15.15
2	RJ	305589.31	20.96
3	PR	117851.68	20.53
4	SC	89660.26	21.47
5	DF	50625.5	21.04
6	MG	270853.46	20.63
7	PA	38699.3	35.83
8	BA	100156.68	26.36
9	GO	53114.98	22.77
10	RS	135522.74	21.74
11	TO	11732.68	37.25
12	AM	5478.89	33.21
13	MA	31523.77	38.26

Inference: similarly like above query we need to use aggregation function for the question but here differs the table name order_items where added above query in place of payments and need to group by for customer_state

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.

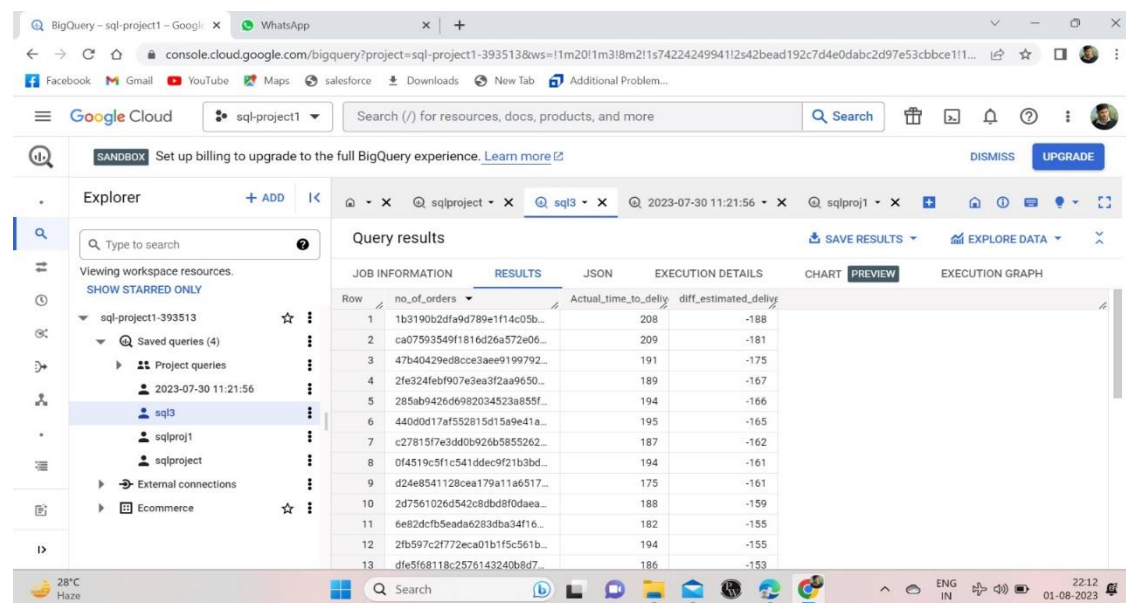
Ans: Select (order_id) as
no_of_orders,date_diff(order_delivered_customer_date,order_purchase_timestamp,DAY) as
Actual_time_to_deliver,

date_diff(order_estimated_delivery_date,order_delivered_customer_date,DAY) as
diff_estimated_delivery

from `sql-project1-393513.Ecommerce.orders`

where date_diff(order_delivered_customer_date,order_purchase_timestamp,DAY) is not null

order by date_diff(order_estimated_delivery_date,order_delivered_customer_date,DAY) asc



Row	no_of_orders	Actual_time_to_deliv	diff_estimated_deliv
1	1b3190b2dfe9d789e1f14c05b...	208	-188
2	ca07593549f1816d26a572e06...	209	-181
3	47b40429ed8cce3aee9199792...	191	-175
4	2fe324feb907e9ea3f2aa9650...	189	-167
5	285ab9426d6982034523a855f...	194	-166
6	440d0d17af552815d15a9e41a...	195	-165
7	c27815f7e3dd0b926b5855262...	187	-162
8	0f4519c5f1c541ddec9f21b3bd...	194	-161
9	d24e8541128cea179a11a6517...	175	-161
10	2d7561026d542c8dbd8f0daea...	188	-159
11	6e82dcfb5eada6283dba34f16...	182	-155
12	2fb597c2f772eca01b1f5c561b...	194	-155
13	dfe5f68118c2576143240b8d7...	186	-153

Inference: Here we use date diff between customer date and purchase time to, calculate the difference (in days) between the estimated & actual delivery date taken the help of orders table and need to filter the data which where not null and order by in asc

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

Ans: ((select c.customer_state,

round(avg(od.freight_value),2) as avg

from `sql-project1-393513.Ecommerce.order_items` od join `sql-project1-393513.Ecommerce.orders` o on o.order_id=od.order_id join `sql-project1-393513.Ecommerce.customers` c on c.customer_id=o.customer_id

group by c.customer_state

order by avg asc

limit 5)

union all

(select (c.customer_state),

round(avg(od.freight_value),2) as avg

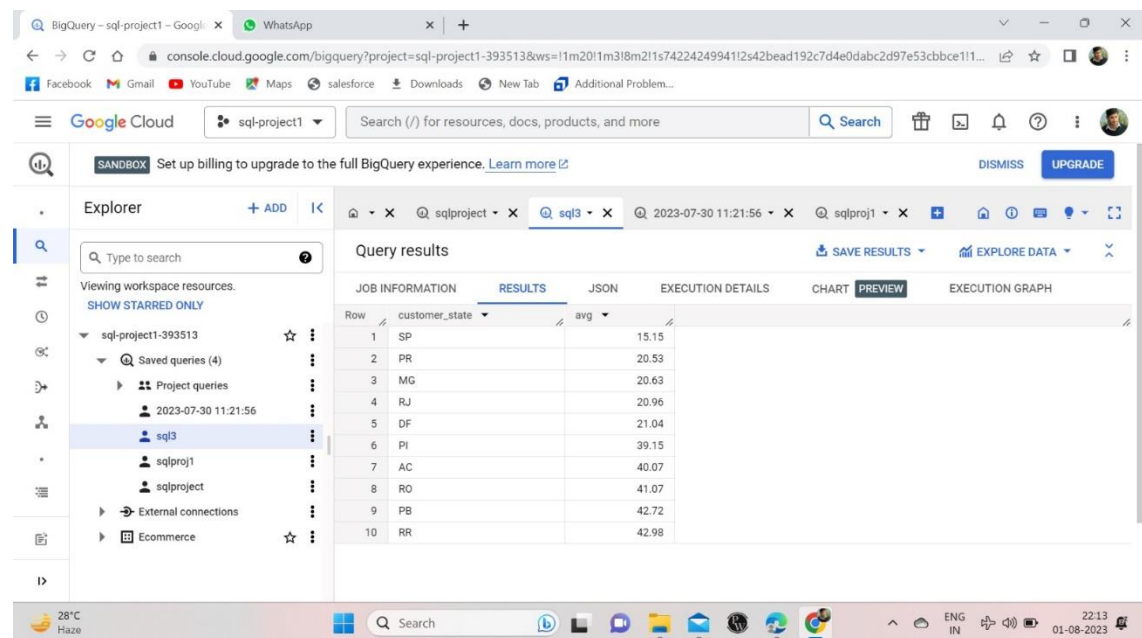
from `sql-project1-393513.Ecommerce.order_items` od join `sql-project1-393513.Ecommerce.orders` o on o.order_id=od.order_id join `sql-project1-393513.Ecommerce.customers` c on c.customer_id=o.customer_id

group by c.customer_state

order by avg desc

limit 5))

order by avg asc;



The screenshot shows the Google Cloud BigQuery console interface. The left sidebar displays the Explorer view with a search bar and a list of workspace resources, including 'sql-project1-393513' and 'Saved queries (4)'. The main panel shows the 'Query results' tab for a query named 'sql3'. The query results are displayed in a table with columns 'customer_state' and 'avg'. The table contains 10 rows of data, ordered by the 'avg' column in descending order. The bottom of the screenshot shows the Windows taskbar with the date and time '22:13 01-08-2023'.

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

Inference: First we need to take the avg freight value in the desc order with the limit 5 means top 5 elements similarly take the avg freight value in asc order with limit 5 means bottom 5 elements now combine both of them using union operator between .

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

Ans: (select c.customer_state,

round(avg(date_diff(o.order_estimated_delivery_date,o.order_delivered_customer_date,day)),2) as avg_del_time

from `sql-project1-393513.Ecommerce.orders` o join `sql-project1-393513.Ecommerce.customers` c on o.customer_id=c.customer_id

where o.order_delivered_customer_date is not null

group by c.customer_state

order by avg_del_time asc

limit 5)

union all

(select c.customer_state,

round(avg(date_diff(o.order_estimated_delivery_date,o.order_delivered_customer_date
,day)),2) as avg_del_time

from `sql-project1-393513.Ecommerce.orders` o join `sql-project1-
393513.Ecommerce.customers` c on o.customer_id=c.customer_id

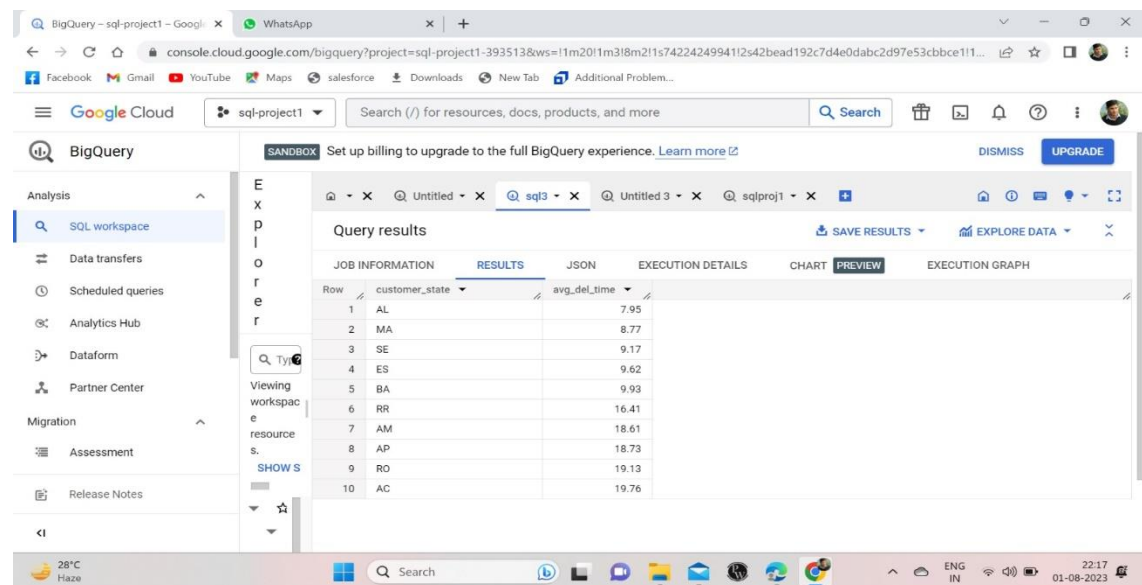
where o.order_delivered_customer_date is not null

group by c.customer_state

order by avg_del_time desc

limit 5)

order by avg_del_time asc;



The screenshot shows the Google Cloud BigQuery console interface. The left sidebar contains navigation options like 'SQL workspace', 'Data transfers', 'Scheduled queries', 'Analytics Hub', 'Dataform', 'Partner Center', 'Migration', 'Assessment', and 'Release Notes'. The main area displays 'Query results' for a query named 'sql3'. The results are shown in a table with columns 'customer_state' and 'avg_del_time'. The table is sorted by 'avg_del_time' in descending order, with the top 5 rows highlighted. The bottom 5 rows are also visible, showing the results of the 'union all' operation. The status bar at the bottom indicates the system is in 'SANDBOX' mode and provides options to 'DISMISS' or 'UPGRADE'.

Row	customer_state	avg_del_time
1	AL	7.95
2	MA	8.77
3	SE	9.17
4	ES	9.62
5	BA	9.93
6	RR	16.41
7	AM	18.61
8	AP	18.73
9	RO	19.13
10	AC	19.76

Inference: First we need to take the avg delivery time value in the desc order with the limit 5 means top 5 elements similarly take the avg delivery time value in asc order with limit 5 means bottom 5 elements now combine both of them using union operator between. Finally arrange the resultant output in the asc order

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

Ans: select c.customer_state,

round(avg(date_diff(o.order_estimated_delivery_date,o.order_delivered_customer_date
,day)),2) as avg_del_time

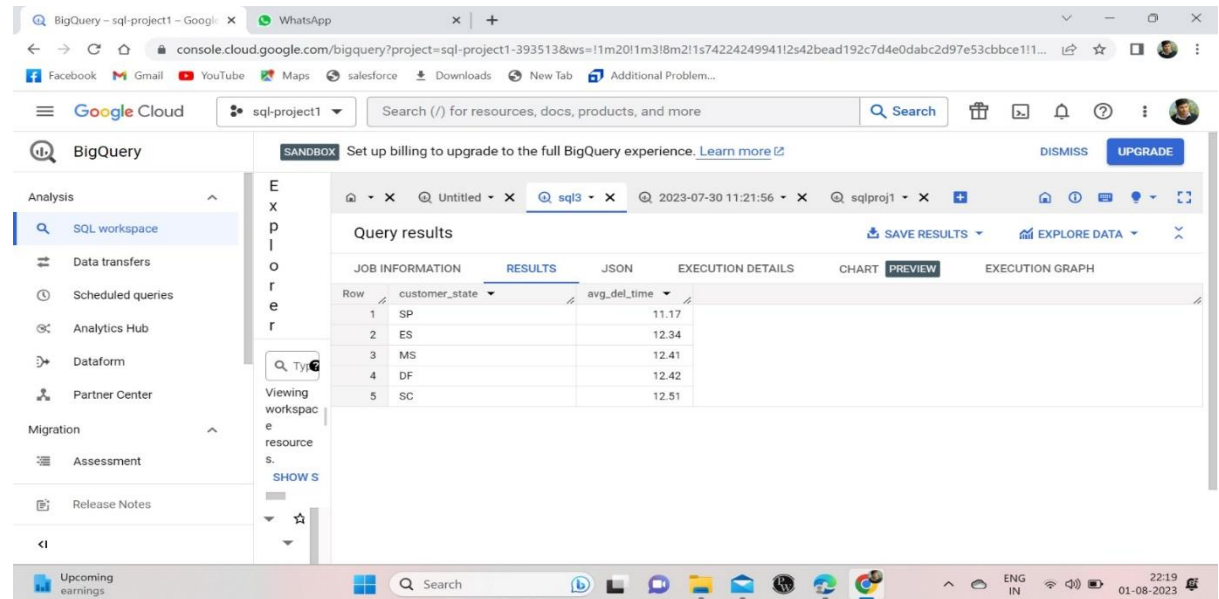
```
from `sql-project1-393513.Ecommerce.orders` o join `sql-project1-393513.Ecommerce.customers` c on o.customer_id=c.customer_id
```

```
where o.order_delivered_customer_date<=o.order_estimated_delivery_date and  
o.order_delivered_customer_date is not null
```

```
group by c.customer_state
```

```
order by avg_del_time
```

```
limit 5;
```



The screenshot shows the Google Cloud BigQuery console interface. On the left is a navigation menu with options like 'SQL workspace', 'Data transfers', 'Scheduled queries', 'Analytics Hub', 'Dataform', 'Partner Center', 'Migration', 'Assessment', and 'Release Notes'. The main area displays 'Query results' for a query named 'sql3'. The results are shown in a table with columns 'customer_state' and 'avg_del_time'. The table contains 5 rows of data.

Row	customer_state	avg_del_time
1	SP	11.17
2	ES	12.34
3	MS	12.41
4	DF	12.42
5	SC	12.51

Inference: Here we need top 5 states whose delivery is too fast than the estimated delivery time here we need to use the where condition in order to filter the data whose delivery time less than the estimated delivery and order delivery date must not equal to null

VI. Analysis based on the payments:

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

Ans: select extract(month from o.order_purchase_timestamp) as month,

p.payment_type,

count(*) as no_of_order

```
from `sql-project1-393513.Ecommerce.orders` o join `sql-project1-393513.Ecommerce.payments` p on o.order_id=p.order_id
```

```
group by extract(month from o.order_purchase_timestamp) ,
```

```
p.payment_type;
```

The screenshot shows the Google Cloud BigQuery console interface. The left sidebar contains navigation options like Analysis, SQL workspace, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Center, Migration, Assessment, and Release Notes. The main area displays a query result table with the following data:

Row	month	payment_type	no_of_order
1	11	UPI	1509
2	12	credit_card	4378
3	2	UPI	1723
4	11	credit_card	5897
5	4	voucher	572
6	7	credit_card	7841
7	7	UPI	2074
8	5	credit_card	8350
9	10	credit_card	3778
10	1	credit_card	6103
11	6	credit_card	7276
12	9	credit_card	3286
13	2	credit_card	6609

Inference: Here we need to find no_of_orders with different payments need to join payments and orders table, for no_of_orders we use count(*) and group by month and payment_type

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

Ans: select count(distinct(order_id)) as no_of_orders,

payment_installments

from `sql-project1-393513.Ecommerce.payments`

where payment_installments > 0

group by payment_installments

order by payment_installments ;

The screenshot shows the Google Cloud BigQuery console interface. The left sidebar contains navigation options like Analysis, SQL workspace, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Center, Migration, Assessment, and Release Notes. The main area displays a query result table with the following data:

Row	no_of_orders	payment_installment
1	49060	1
2	12389	2
3	10443	3
4	7088	4
5	5234	5
6	3916	6
7	1623	7
8	4253	8
9	644	9
10	5315	10
11	23	11
12	133	12
13	16	13

Inference: For no_of_orders we use count(*) and choose payment_installment column where payment_installment must be greater than 0 mean positive value only needed and group by the payment_installment finally order by payment_installment by default order by is in asc order no need to mention.