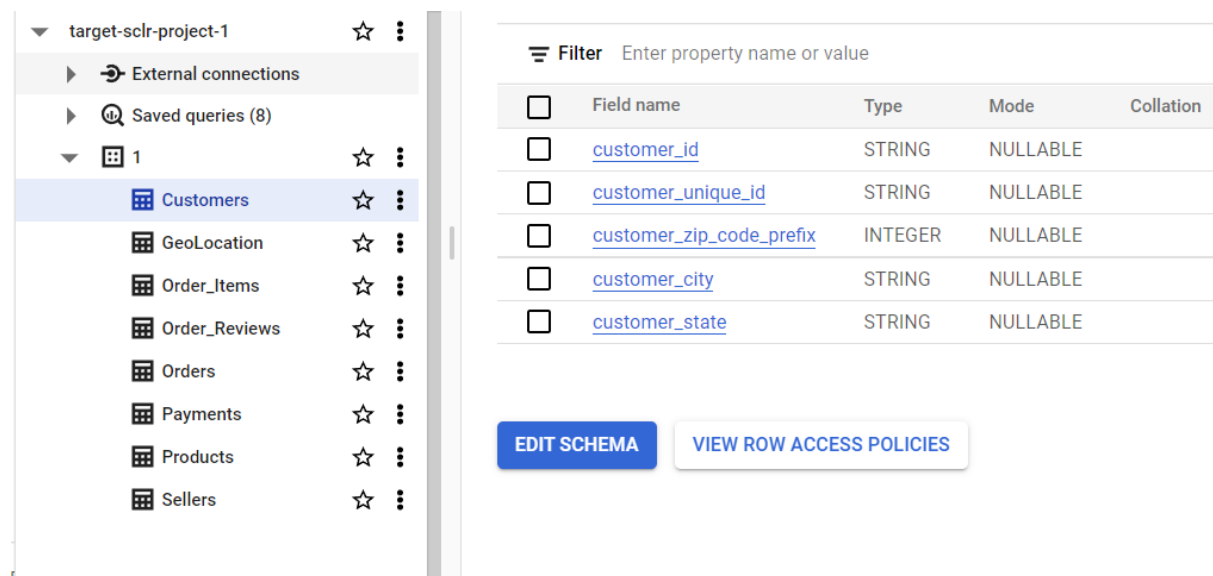


Name : Preetham Tiwari K
Target SQL Project Submission
Submission Date : 26-10-2022

Submission for the Project Target SQL which contains data of orders from Target store in Brazil .There were a total of 8 csv files provided for the project .The csv files were imported in google big query and analysis was performed based on the requirements .

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

For this question the data type of columns was available upon importing the data into bigquery .



The screenshot shows the Google BigQuery interface. On the left, a sidebar displays the project hierarchy: 'target-sqlr-project-1' > 'External connections' > 'Saved queries (8)' > '1' > 'Customers' (selected). Below 'Customers' are other tables: 'GeoLocation', 'Order_Items', 'Order_Reviews', 'Orders', 'Payments', 'Products', and 'Sellers'. On the right, the 'Customers' table schema is displayed. It has a 'Filter' bar at the top with the text 'Enter property name or value'. Below it is a table with columns: 'Field name', 'Type', 'Mode', and 'Collation'. The rows are: 'customer_id' (STRING, NULLABLE), 'customer_unique_id' (STRING, NULLABLE), 'customer_zip_code_prefix' (INTEGER, NULLABLE), 'customer_city' (STRING, NULLABLE), and 'customer_state' (STRING, NULLABLE). At the bottom of the schema view are two buttons: 'EDIT SCHEMA' and 'VIEW ROW ACCESS POLICIES'.

Field name	Type	Mode	Collation
customer_id	STRING	NULLABLE	
customer_unique_id	STRING	NULLABLE	
customer_zip_code_prefix	INTEGER	NULLABLE	
customer_city	STRING	NULLABLE	
customer_state	STRING	NULLABLE	

The data type of the columns in the Customers table was mostly of string data type except the customer_zip_code_prefix column which had the data type as integer.

The same analysis could be performed on all the tables to understand the data type in each column

2. Time period for which the data is given

To understand the time period of the given data the minimum and maximum date from the order_purchase_timestamp column in the orders table was used .

The screenshot shows the Google BigQuery interface. On the left, the Explorer pane shows a project named 'target-sclr-project-1' with a table 'Orders' selected. The main editor displays a SQL query to find the minimum and maximum purchase timestamps and calculate time differences. The query results are shown in a table with columns for start_date, end_date, and three time range calculations (Days, Month, Year).

```
1 SELECT min(order_purchase_timestamp) as start_date,
2 max(order_purchase_timestamp) as end_date ,
3 DATE_DIFF(max(order_purchase_timestamp),min(order_purchase_timestamp),DAY) as Time_Range_Days,
4 DATE_DIFF(DATE '2018-10-17',DATE '2016-09-04' ,Month) as Time_Range_Month,
5 DATE_DIFF(DATE '2018-10-17',DATE '2016-09-04' ,YEAR) as Time_Range_Year
6 FROM `target-sclr-project-1.1.Orders`
7
```

Row	start_date	end_date	Time_Range...	Time_Range...	Time_Range...
1	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC	772	25	2

Query:

```
SELECT min(order_purchase_timestamp) as start_date,

max(order_purchase_timestamp) as end_date ,

DATE_DIFF(max(order_purchase_timestamp),min(order_purchase_timestamp),
DAY) as Time_Range_Days,

DATE_DIFF(DATE '2018-10-17',DATE '2016-09-04' ,Month) as
Time_Range_Month,

DATE_DIFF(DATE '2018-10-17',DATE '2016-09-04' ,YEAR) as
Time_Range_Year

FROM `target-sclr-project-1.1.Orders`
```

Upon selecting the maximum and minimum dates from the order_purchase_timestamp column the dates that were displayed are

Minimum date = 2016-09-04

Maximum date = 2018-10-17

The maximum date column was renamed as start_date and minimum date column was renamed as end_date .Then the DATE_DIFF function was used to calculate the difference between the maximum and minimum dates to get the range of days ,months and year for the given data and the columns were named as Time_Range_Days for range of days ,Time_Range_Months for the range of months and Time_Range_Year for the number of years .From the result the following time ranges were displayed .

Start date of data = 2016-09-04

End date of data = 2018-10-17

Number of days = 772

Number of months = 25

Number of years = 2

So it can be inferred from the result that the date range of the given data is between 2016-09-04 and 2018-10-17 and number of days in the date range is 772 number of months is 25 and number of years is 2

3. Cities and States covered in the dataset

To understand the cities and states covered in the dataset the category had to be split into 3 based on the available tables .

- Customer cities and states
- States and cities from the geolocation
- States and cities from the seller table.

Customer cities and states.

To get the information about the customer cities the following query was run .

The screenshot shows a data exploration interface with a sidebar on the left and a main workspace on the right. The sidebar contains an 'Explorer' panel with a search bar and a list of resources under 'target-sclr-project-1'. The main workspace displays a SQL query in a text editor and its results in a table.

SQL Query:

```
SELECT DISTINCT(customer_city) FROM `target-sclr-project-1.1.Customers`
```

Query results:

Row	customer_city
1	acu
2	ico
3	ipe
4	ipu
5	ita

The interface also shows a 'Query results' section with tabs for 'JOB INFORMATION', 'RESULTS', 'JSON', and 'EXECUTION DETAILS'. The 'RESULTS' tab is active, displaying the table above. At the bottom, there are links for 'PERSONAL HISTORY' and 'PROJECT HISTORY'.

SQL Query :

```
SELECT DISTINCT(customer_city) FROM  
`target-sclr-project-1.1.Customers`
```

This query gave a result indicating that the customers were distributed in a total of 4119 cities in Brazil.

To get the information about the customer states the following query was run .

The screenshot shows a data analytics interface with a sidebar on the left containing a search bar and a list of resources. The main area displays a SQL query in a text editor and its results in a table.

SQL Query:

```
1 SELECT DISTINCT(customer_state)
2 FROM `target-sclr-project-1.1.Customers`
```

Query results:

Row	customer_state
1	RN
2	CE
3	RS
4	SC
5	SP

The interface also includes tabs for 'JOB INFORMATION', 'RESULTS', 'JSON', and 'EXECUTION DETAILS'. The 'RESULTS' tab is currently selected, showing the query results table. At the bottom, there are tabs for 'PERSONAL HISTORY' and 'PROJECT HISTORY'.

SQL Query :

```
SELECT DISTINCT(customer_state)

FROM `target-sclr-project-1.1.Customers`
```

From the result it can be seen that the customers are distributed in 27 states across Brazil.

- States and cities from the geolocation

To get the information about the cities from the geolocation table the following queries were run.

The screenshot shows a data analytics interface with a sidebar on the left containing a search bar and a list of resources. The main area displays a SQL query in a text editor and its results in a table.

SQL Query:

```
1 SELECT DISTINCT(geolocation_city) FROM `target-sclr-project-1.1.GeoLocation`
2
```

Query results:

Row	geolocation_city
1	aracaju
2	riachuelo
3	nossa senhora do socorro
4	barra dos coqueiros
5	Itanoranna r'Alairia

The interface also includes tabs for 'JOB INFORMATION', 'RESULTS', 'JSON', and 'EXECUTION DETAILS'. The 'RESULTS' tab is currently selected, showing the query results table. At the bottom, there are tabs for 'PERSONAL HISTORY' and 'PROJECT HISTORY'.

SQL Query :

```
SELECT DISTINCT(geolocation_city)
FROM `target-sclr-project-1.1.GeoLocation`
```

From the result it can be seen that there are a total of 8011 cities listed in the geolocation table .

To get the information about the states from the geolocation table the following queries were run.

The screenshot shows a data exploration interface. On the left is an 'Explorer' pane with a search bar and a list of saved queries. The main area displays a SQL query in a text editor, and below it, the 'Query results' section shows a table of data.

SQL Query:

```
1 SELECT DISTINCT(geolocation_state)
2 FROM `target-sclr-project-1.1.GeoLocation`
3
4
```

Query results:

Row	geolocation_state
1	SE
2	AL
3	PI
4	AP
5	AM

Results per page: 50 1 - 27 of 27

SQL Query:

```
SELECT DISTINCT(geolocation_state)
FROM `target-sclr-project-1.1.GeoLocation`
```

The results from the query indicate that there were a total of 27 states listed in the geolocation table.

States and cities from the seller table.

To get the information about the seller cities from the sellers table the following query was run .

The screenshot shows a data exploration interface. On the left, the 'Explorer' pane lists resources under 'target-sclr-project-1', with 'Seller City' selected. The main query editor displays the following SQL query:

```
1 SELECT DISTINCT(seller_city)
2 FROM `target-sclr-project-1.1.Sellers`
3
```

The 'Query results' pane shows a table with 5 rows of distinct seller cities:

Row	seller_city
1	rio branco
2	manaus
3	bahia
4	ipira
5	irene

The interface also shows tabs for 'PERSONAL HISTORY' and 'PROJECT HISTORY'.

SQL Query :

```
SELECT DISTINCT(seller_city)
FROM `target-sclr-project-1.1.Sellers`
```

The results from this query were interesting as the sellers seemed to be distributed across only 611 cities in comparison to the customers who were distributed across 4119 cities.

To get the information about the seller states from the sellers table the following query was run .

The screenshot shows the same data exploration interface, but with 'Seller state' selected in the Explorer pane. The SQL query editor displays the following query:

```
1 SELECT DISTINCT(seller_state)
2 FROM `target-sclr-project-1.1.Sellers`
```

The 'Query results' pane shows a table with 5 rows of distinct seller states:

Row	seller_state
1	AC
2	AM
3	BA
4	CE
5	DF

The interface also shows tabs for 'PERSONAL HISTORY' and 'PROJECT HISTORY'.

SQL Query

```
SELECT DISTINCT(seller_state)
FROM `target-sclr-project-1.1.Sellers`
```

From the results it can be seen that the sellers are distributed across 23 states which is also less in comparison to the distribution of customers across 27 states.

Summary of results .

Count of cities of customer distribution : 4119

Count of states of customer distribution : 27

Count of cities from geolocation table : 8011

Count of states from geolocation table : 27

Count of cities of seller distribution : 611

Count of states of seller distribution : 23

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?

For this question a query extracting the year and month from the column `order_purchase_timestamp` was executed by considering order status of only delivered orders and the count of order id as `OrderCount`.

The screenshot displays the Google BigQuery interface. On the left, the 'Explorer' panel shows a list of saved queries, with 'Seller state' selected. The main editor shows a SQL query:

```
1 SELECT EXTRACT(Year FROM order_purchase_timestamp) as Year,
2 EXTRACT(Month FROM order_purchase_timestamp) as Month,
3 COUNT(order_id) as OrderCount FROM `target-sclr-project-1.1.Orders`
4 WHERE order_status='delivered'
5 GROUP BY Year,Month ORDER BY Year,Month
```

 Below the query editor, the 'Query results' section shows a table with 5 rows of data. The table has columns: Row, Year, Month, and OrderCount. The results are: Row 1 (2016, 9, 1), Row 2 (2016, 10, 265), Row 3 (2016, 12, 1), Row 4 (2017, 1, 750), and Row 5 (2017, 2, 1653). The interface also includes a top bar with 'Sandbox' and 'Set up billing' options, and a bottom bar with 'PERSONAL HISTORY' and 'PROJECT HISTORY' tabs.

Row	Year	Month	OrderCount
1	2016	9	1
2	2016	10	265
3	2016	12	1
4	2017	1	750
5	2017	2	1653

SQL Query:

```
SELECT EXTRACT(Year FROM order_purchase_timestamp) as Year,

EXTRACT(Month FROM order_purchase_timestamp) as Month,

COUNT(order_id) as OrderCount

FROM `target-sclr-project-1.1.Orders`

WHERE order_status='delivered'

GROUP BY Year,Month ORDER BY Year,Month
```

From the result it can be seen that the trend has been on a consistent growth the time period of 2016 september to the last time period of august 2018 .

There were less than 300 orders places in the three months available for analysis from the year 2016 and in comparison to this the number of orders placed in the year 2017 were 43428 and in just the 8 months available for analysis in the year 2018 the total order count was 52783 which is greater than the total number of orders placed in the entire year of 2017. This can be seen from the query below.

The screenshot shows the Google BigQuery web interface. On the left is the Explorer panel with a search bar and a list of saved queries. The main editor displays a SQL query that calculates the sum of order counts by year. Below the editor, the 'Query results' section shows a table with 3 rows of data. The table has columns for Row, Sum_Of_OrderCount, and Year. The results show 267 orders for 2016, 43428 for 2017, and 52783 for 2018. The interface also includes a top navigation bar with options like 'RUN', 'SAVE', and 'SHARE', and a bottom status bar with 'PERSONAL HISTORY' and 'PROJECT HISTORY'.

Row	Sum_Of_OrderCount	Year
1	267	2016
2	43428	2017
3	52783	2018

SQL Query:

```
SELECT SUM(OrderCount) as Sum_Of_OrderCount,

Year

FROM

(SELECT EXTRACT(Year FROM order_purchase_timestamp) as Year,

COUNT(order_id) as OrderCount FROM `target-sclr-project-1.1.Orders`
```

```
WHERE order_status='delivered'
GROUP BY Year ORDER BY Year)
GROUP BY Year ORDER BY Sum_Of_OrderCount
```

Such numbers are a clear indication of a growing trend in e-commerce in Brazil.

The month during which the maximum orders were placed is the month of November from the year 2017 with the order count being 7289 .

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer pane with a search bar and a list of saved queries. The main area displays a SQL query and its results. The query is as follows:

```
1 SELECT EXTRACT(YEAR FROM order_purchase_timestamp) as Year,
2 EXTRACT(MONTH FROM order_purchase_timestamp) as Month,
3 COUNT(order_id) as OrderCount
4 FROM `target-sclr-project-1.1.Orders`
5 WHERE order_status='delivered'
6 GROUP BY Year,Month ORDER BY OrderCount desc LIMIT 5
```

The query results are displayed in a table with the following data:

Row	Year	Month	OrderCount
1	2017	11	7289
2	2018	1	7069
3	2018	3	7003
4	2018	4	6798
5	2018	5	6749

SQL Query :

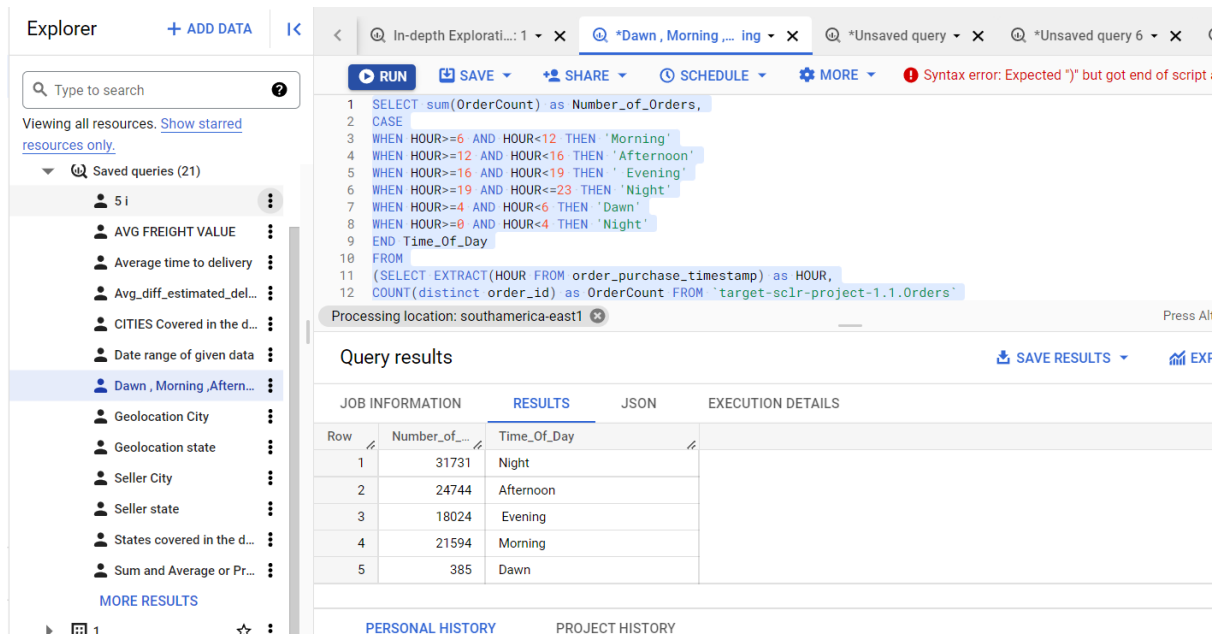
```
SELECT EXTRACT(YEAR FROM order_purchase_timestamp) as Year,
EXTRACT(MONTH FROM order_purchase_timestamp) as Month,
COUNT(order_id) as OrderCount
FROM `target-sclr-project-1.1.Orders`
WHERE order_status='delivered'
GROUP BY Year,Month ORDER BY OrderCount desc LIMIT 5
```

This could be because of 2 major festivals that are celebrated in Brazil in the month of November which are Day of the Dead (November 2) and Proclamation of the Republic (November 15) .

But the other interesting factor to be observed is that in the above query which displays the top 5 months where maximum orders were placed apart from the 1st result which is from the year 2017 all the rest of the results are from the year 2018 . This is also a clear indicator that the e commerce business is on a stark rise in Brazil.

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

To understand the purchasing pattern of customers in Brazil based on the time of the day the column order_purchase_timestamp from the Orders table was analysed and the following query was executed .



The screenshot shows a SQL query editor with a query that categorizes orders by time of day. The query uses a CASE statement to map time ranges to categories: Morning (6-12), Afternoon (12-16), Evening (16-19), Night (19-23), Dawn (4-6), and Night (0-4). The results table shows the count of orders for each category.

```
1 SELECT sum(OrderCount) as Number_of_Orders,
2 CASE
3 WHEN HOUR>=6 AND HOUR<12 THEN 'Morning'
4 WHEN HOUR>=12 AND HOUR<16 THEN 'Afternoon'
5 WHEN HOUR>=16 AND HOUR<19 THEN 'Evening'
6 WHEN HOUR>=19 AND HOUR<=23 THEN 'Night'
7 WHEN HOUR>=4 AND HOUR<6 THEN 'Dawn'
8 WHEN HOUR>=0 AND HOUR<4 THEN 'Night'
9 END Time_Of_Day
10 FROM
11 (SELECT EXTRACT(HOUR FROM order_purchase_timestamp) as HOUR,
12 COUNT(distinct order_id) as OrderCount FROM `target-solr-project-1.1.Orders`
13 )
```

Row	Number_of_Orders	Time_Of_Day
1	31731	Night
2	24744	Afternoon
3	18024	Evening
4	21594	Morning
5	385	Dawn

SQL Query :

```
SELECT sum(OrderCount) as Number_of_Orders,

CASE

WHEN HOUR>=6 AND HOUR<12 THEN 'Morning'

WHEN HOUR>=12 AND HOUR<16 THEN 'Afternoon'

WHEN HOUR>=16 AND HOUR<19 THEN 'Evening'

WHEN HOUR>=19 AND HOUR<=23 THEN 'Night'

WHEN HOUR>=4 AND HOUR<6 THEN 'Dawn'

WHEN HOUR>=0 AND HOUR<4 THEN 'Night'

END Time_Of_Day

FROM
```

```

(SELECT EXTRACT(HOUR FROM order_purchase_timestamp) as HOUR,

COUNT(distinct order_id) as OrderCount FROM `
target-sclr-project-1.1.Orders`

where order_status = 'delivered'

group by HOUR

) group by Time_Of_Day

```

The above query splits of the 24 hour time period in a day into four categories as below .

4am - 6am - Dawn
 6am - 12pm -Morning
 12pm - 4pm - Afternoon
 4pm - 7pm - Evening
 7pm-4am -Night

These different time categories are obtained from the order_purchase_timestamp column in the orders table .The count of order id is considered for order count and from the results it can be seen that the maximum number of orders are placed during the night with the count being 31731 orders .The night time ranges from 7pm -4am .further analysis can be performed to understand this trait in detail so as to maximise the orders that be sold to the customers.

The screenshot displays a data analytics tool interface. On the left is an 'Explorer' panel with a search bar and a list of saved queries. The main area shows a SQL query editor with a query that categorizes orders by time of day. Below the editor, the 'Query results' section is active, showing a table with 6 rows of data. The table has columns for 'Row', 'Number_of_Orders', and 'Time_Of_Day'. The results show that the 'Night1' category has the highest number of orders (27522), followed by 'Afternoon' (24744), 'Evening' (18024), 'Night2' (4209), 'Morning' (21594), and 'Dawn' (385).

Row	Number_of_Orders	Time_Of_Day
1	27522	Night1
2	24744	Afternoon
3	18024	Evening
4	4209	Night2
5	21594	Morning
6	385	Dawn

SQL Query:

```
SELECT sum(OrderCount) as Number_of_Orders,

CASE

WHEN HOUR>=6 AND HOUR<12 THEN 'Morning'

WHEN HOUR>=12 AND HOUR<16 THEN 'Afternoon'

WHEN HOUR>=16 AND HOUR<19 THEN 'Evening'

WHEN HOUR>=19 AND HOUR<=23 THEN 'Night1'

WHEN HOUR>=4 AND HOUR<6 THEN 'Dawn'

WHEN HOUR>=0 AND HOUR<4 THEN 'Night2'

END Time_Of_Day

FROM

(SELECT EXTRACT(HOUR FROM order_purchase_timestamp) as HOUR,

COUNT(distinct order_id) as OrderCount FROM `target-sclr-project-1.1.Orders`

where order_status = 'delivered'

group by HOUR

) group by Time_Of_Day
```

Upon performing further analysis by splitting the night time into parts as Night 1 which is for the time period between 7 pm - 12 am and Night 2 which is for the time period between 12am - 4am it can be seen that the maximum orders are placed between 7pm - 12 pm with the count being 27522 .

From these results it can be seen that the time period between 7pm -12 am is the most preferable time to launch new products and provide attractive offers and discounts to the customers as the traffic of customers is at the peak during these hours .

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

1. Get month on month orders by region, states

The following query provides the data for the count of month on month orders by city and state.

The screenshot shows a data analytics tool interface. On the left is an 'Explorer' panel with a search bar and a list of saved queries. The main area displays a SQL query in a text editor. Below the editor, the 'Query results' section shows a table with columns: Row, Year, Month, OrderCount, customer_city, and customer_state. The table contains 5 rows of data. At the bottom right, it indicates 'Results per page: 50' and '1 - 50 of 21333'.

```
1 SELECT EXTRACT(YEAR FROM o.order_purchase_timestamp) as Year,
2 EXTRACT(MONTH FROM o.order_purchase_timestamp) as Month,
3 COUNT(o.order_id) as OrderCount,
4 c.customer_city,
5 c.customer_state
6 FROM `target-sclr-project-1.1.Orders` o
7 JOIN `target-sclr-project-1.1.Customers` c
8 ON o.customer_id = c.customer_id
9 WHERE order_status='delivered'
10 GROUP BY Year,Month,c.customer_state,c.customer_city ORDER BY OrderCount desc
```

Row	Year	Month	OrderCount	customer_city	customer_state
1	2018	8	1269	sao paulo	SP
2	2018	5	1191	sao paulo	SP
3	2018	4	1135	sao paulo	SP
4	2018	3	1127	sao paulo	SP
5	2017	11	1081	sao paulo	SP

SQL Query :

```
SELECT EXTRACT(YEAR FROM o.order_purchase_timestamp) as Year,

EXTRACT(MONTH FROM o.order_purchase_timestamp) as Month,

COUNT(o.order_id) as OrderCount,

c.customer_city,

c.customer_state

FROM `target-sclr-project-1.1.Orders` o

JOIN `target-sclr-project-1.1.Customers` c

ON o.customer_id = c.customer_id

WHERE order_status='delivered'

GROUP BY Year,Month,c.customer_state,c.customer_city ORDER BY

OrderCount desc
```

We can see that the city of Sao Paulo is the region with the highest orders placed in a month .Sao Paulo is the only city in the list of top 5 cities for orders placed in a month .

The screenshot shows a SQL query editor with a query that extracts the year and month from the order purchase timestamp, counts the number of orders per city, and orders the results by count descending, limited to 5 rows. The query results table shows the following data:

Row	Year	Month	OrderCount	customer_city	customer_state
1	2018	8	1269	sao paulo	SP
2	2018	5	1191	sao paulo	SP
3	2018	4	1135	sao paulo	SP
4	2018	3	1127	sao paulo	SP
5	2017	11	1081	sao paulo	SP

SQL Query :

```
SELECT EXTRACT(Year FROM o.order_purchase_timestamp) as Year,
EXTRACT(Month FROM o.order_purchase_timestamp) as Month,
COUNT(o.order_id) as OrderCount,
C.customer_city,
C.customer_state
FROM `target-sclr-project-1.1.Orders` o
JOIN `target-sclr-project-1.1.Customers` c
ON o.customer_id = c.customer_id
WHERE order_status='delivered'
GROUP BY Year,Month,c.customer_state,c.customer_city ORDER BY
OrderCount desc LIMIT 5
```

Special offers and features focused for the city of Sao Paulo would bring in more customers and orders .

The 5 cities with least number of orders placed in a month can be seen by executing the following query

The screenshot shows the Google BigQuery interface. On the left is the Explorer panel with a search bar and a list of saved queries. The main editor displays a SQL query: `SELECT EXTRACT(YEAR FROM o.order_purchase_timestamp) as Year, EXTRACT(MONTH FROM o.order_purchase_timestamp) as Month, COUNT(o.order_id) as OrderCount, c.customer_city, c.customer_state FROM `target-sqlr-project-1.1.Orders` o JOIN `target-sqlr-project-1.1.Customers` c ON o.customer_id = c.customer_id WHERE order_status='delivered' GROUP BY Year, Month, c.customer_state, c.customer_city ORDER BY OrderCount asc LIMIT 5`. The query results are shown in a table with columns: Row, Year, Month, OrderCount, customer_city, and customer_state. The results list 5 cities: jacaraci (BA), sao jose do cedro (SC), divisa nova (MG), almas (TO), and entre-ijos (RS), all with an OrderCount of 4.

Row	Year	Month	OrderCount	customer_city	customer_state
1	2017		4	jacaraci	BA
2	2017		4	sao jose do cedro	SC
3	2017		4	divisa nova	MG
4	2017		4	almas	TO
5	2017		4	entre-ijos	RS

These cities have a very low order count and do not provide any value to business.

2. How are customers distributed in Brazil

To get the data about the top 5 states with the most customer count the following query was executed .

The screenshot shows the Google BigQuery interface. The main editor displays a SQL query: `SELECT COUNT(customer_unique_id) as CustomerCount, customer_state FROM `target-sqlr-project-1.1.Customers` GROUP BY customer_state ORDER BY CustomerCount Desc Limit 5`. The query results are shown in a table with columns: Row, CustomerCount, and customer_state. The results list the top 5 states: SP (41746), RJ (12852), MG (11635), RS (5466), and PR (5045).

Row	CustomerCount	customer_state
1	41746	SP
2	12852	RJ
3	11635	MG
4	5466	RS
5	5045	PR

SQL Query :

```
SELECT COUNT(customer_unique_id) as CustomerCount ,  
customer_state  
FROM `target-sclr-project-1.1.Customers`  
GROUP BY customer_state ORDER BY CustomerCount Desc Limit 5
```

The results indicate that the state with the most customers is Sao Paulo with a count of 41746 customers .

To get the data for the states with the least number of customers the following query was executed .

The screenshot shows the Google BigQuery interface. On the left is the Explorer panel with a search bar and a list of saved queries. The main editor displays a SQL query to count customers by state, ordered by count in descending order with a limit of 5. Below the query editor, the 'Query results' section is visible, showing a table with 5 rows of data. The table has columns for Row number, CustomerCount, and customer_state. The results are: Row 1 (RR, 46), Row 2 (AP, 68), Row 3 (AC, 81), Row 4 (AM, 148), and Row 5 (RO, 253).

Row	CustomerC...	customer_state
1	46	RR
2	68	AP
3	81	AC
4	148	AM
5	253	RO

SQL Query :

```
SELECT COUNT(customer_unique_id) as CustomerCount ,  
customer_state  
FROM `target-sclr-project-1.1.Customers`  
GROUP BY customer_state ORDER BY CustomerCount asc Limit 5
```

The query indicates that the state Roraima indicated as RR has the least number of customers with the count being just 46.

To understand the maximum count of customers across cities the following query was executed .

The screenshot shows the Google Cloud BigQuery console interface. On the left is the Explorer sidebar with a search bar and a list of saved queries. The main panel displays a SQL query in the editor:

```
1 SELECT COUNT(customer_unique_id) as CustomerCount ,
2 customer_city
3 FROM `target-sclr-project-1.1.Customers`
4 GROUP BY customer_city ORDER BY CustomerCount Desc Limit 5
```

Below the editor, the 'Query results' section is visible, showing a table with 5 rows of data:

Row	CustomerC...	customer_city
1	15540	sao paulo
2	6882	rio de janeiro
3	2773	belo horizonte
4	2131	brasilia
5	1521	curitiba

SQL Query :

```
SELECT COUNT(customer_unique_id) as CustomerCount ,
customer_city
FROM `target-sclr-project-1.1.Customers`
GROUP BY customer_city ORDER BY CustomerCount Desc Limit 5
```

Sao Paulo is the city with the most number of customers with the count being 15540

To understand the least count of customers across cities the following query was executed .

The screenshot shows the Google Cloud BigQuery console interface. The SQL query in the editor is:

```
1 SELECT COUNT(customer_unique_id) as CustomerCount ,
2 customer_city
3 FROM `target-sclr-project-1.1.Customers`
4 GROUP BY customer_city ORDER BY CustomerCount asc Limit 5
```

The 'Query results' section shows a table with 5 rows of data:

Row	CustomerC...	customer_city
1	1	caem
2	1	aval
3	1	bodo
4	1	cipo
5	1	bora

All the 5 cities in the result have a customer count of just 1 and are the cities with the least count of customers .

1. 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)

To calculate the percent increase of cost of orders from the year 2017 - 2018 for the first 8 months the following query was executed.

The screenshot shows the Google BigQuery interface. On the left is the Explorer panel with a search bar and a list of saved queries. The main editor displays a SQL query. Below the editor, the 'Query results' section is visible, showing a table with one row and two columns: 'Row' and 'Percentage...'. The result shows a percentage increase of 141.0.

```

8 CTE2 AS ( SELECT
9   SUM (OI.price + OI.freight_value) as cost_of_order
10  FROM `target-sclr-project-1.1.Orders` as O
11  JOIN `target-sclr-project-1.1.Order_Items` OI
12    on O.order_id = OI.order_id
13  WHERE order_purchase_timestamp between '2018-01-01' and '2018-08-31'
14 )
15
16 SELECT ROUND(((CTE2.cost_of_order-CTE1.cost_of_order)/CTE1.cost_of_order)*100,2)
17 AS Percentage_increase_cost_of_orders
18 FROM CTE1,CTE2
19

```

Row	Percentage...
1	141.0

SQL Query :

```

WITH CTE1 AS ( SELECT

SUM (OI.price + OI.freight_value) as cost_of_order

FROM `target-sclr-project-1.1.Orders` as O

JOIN `target-sclr-project-1.1.Order_Items` OI

on O.order_id = OI.order_id

WHERE order_purchase_timestamp between '2017-01-01' and '2017-08-31'

) ,

CTE2 AS ( SELECT

SUM (OI.price + OI.freight_value) as cost_of_order

FROM `target-sclr-project-1.1.Orders` as O

```

```

JOIN `target-sclr-project-1.1.Order_Items` OI

on O.order_id = OI.order_id

WHERE order_purchase_timestamp between '2018-01-01' and '2018-08-31'

)

SELECT
ROUND(((CTE2.cost_of_order-CTE1.cost_of_order)/CTE1.cost_of_order)*100
,2)

AS Percentage_increase_cost_of_orders

FROM CTE1,CTE2

```

The query is executed across 2 tables which are orders and order items tables. From the order items table the columns price and freight_value are summed and from the table orders the column order_purchase_timestamp is considered to choose the time period between January and August. The sum of price and freight value for the period between January and August for the year 2017 and the same is calculated for the year 2018 and the difference of the result between the year 2018 and 2017 is dividing by the value obtained for 2017 and the result is rounded to 2 to get the percentage increase in cost of orders.

Percentage increase in cost of orders = 141.0

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

The following query provides the mean and sum of price and freight value by customer state .

The screenshot displays the Google Cloud BigQuery interface. On the left, the 'Explorer' panel shows a list of saved queries, with 'Sum and Average or Pr...' selected. The main editor shows a SQL query that calculates the sum and average of price and freight values, grouped by customer state and ordered by customer state in descending order. The query results are displayed in a table with columns for Row, Sum_Of_Pri..., Sum_Of_Fre..., Average_Pri..., Average_Fre..., customer_state, and customer_unique_id. The results show data for five rows, with customer states TO, TO, TO, TO, and TO. The 'Results per page' is set to 50, and the total number of results is 50 of 95458.

```
1 SELECT SUM(price) AS Sum_Of_Price,
2 sum(freight_value) AS Sum_Of_Freight_Value,
3 avg(price) AS Average_Price,
4 avg(freight_value) AS Average_Freight_Value,
5 c.customer_state,
6 c.customer_unique_id
7 FROM `target-sclr-project-1.1.Order_Items` OI
8 JOIN `target-sclr-project-1.1.Orders` O
9 ON OI.order_id = O.order_id
10 JOIN `target-sclr-project-1.1.Customers` C
11 ON C.customer_id = O.customer_id
12 GROUP BY c.customer_state, c.customer_unique_id ORDER BY c.customer_state desc
```

Row	Sum_Of_Pri...	Sum_Of_Fre...	Average_Pri...	Average_Fre...	customer_state	customer_unique_id
1	526.0	28.96	526.0	28.96	TO	0465f164e35aba70f70e911b2...
2	50.9	19.06	50.9	19.06	TO	c6557ef8aee5f76221cc8e12ed...
3	349.58	25.0	349.58	25.0	TO	f849b8ea39f8ae164ab09d6d5...
4	213.75	36.82	213.75	36.82	TO	f04fb267036654af73a83e7c4d...
5	78.0	35.56	78.0	35.56	TO	35hhh96d4850n5f9ahR1302h4d...

SQL Query:

```
SELECT SUM(price) AS Sum_Of_Price,
sum(freight_value) AS Sum_Of_Freight_Value,
avg(price) AS Average_Price,
avg(freight_value) AS Average_Freight_Value,
c.customer_state,
c.customer_unique_id
FROM `target-sclr-project-1.1.Order_Items` OI
JOIN `target-sclr-project-1.1.Orders` O
ON OI.order_id = O.order_id
JOIN `target-sclr-project-1.1.Customers` C
ON C.customer_id = O.customer_id
GROUP BY c.customer_state, c.customer_unique_id
ORDER BY c.customer_state desc
```

5. Analysis on sales, freight and delivery time

1. Calculate days between purchasing, delivering and estimated delivery

The following query provides the required information .

The screenshot shows the Google BigQuery interface. On the left is the Explorer panel with a search bar and a list of saved queries. The main area displays a SQL query in the editor, and below it, the 'Query results' section shows a table with 5 rows and 4 columns (d1, d2, d3, and an unlabeled column). The results are sorted by d1 in descending order.

```
1 SELECT DATE_DIFF(order_estimated_delivery_date,order_purchase_timestamp,DAY) as d1,
2 DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY) as d2,
3 DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date,DAY) as d3
4 FROM `target-sclr-project-1.1.Orders`
5 WHERE order_delivered_customer_date IS NOT NULL
6 ORDER BY d1 desc
7
```

Row	d1	d2	d3
1	155	20	134
2	149	3	146
3	146	6	139
4	140	16	123
5	116	7	108

SQL Query :

```
SELECT DATE_DIFF(order_estimated_delivery_date,order_purchase_timestamp,DAY)
as d1,
```

```
DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY) as d2,
```

```
DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date,DAY)
as d3
```

```
FROM `target-sclr-project-1.1.Orders`
```

```
WHERE order_delivered_customer_date IS NOT NULL
```

```
ORDER BY d1 desc
```

In the above query d1,d2 and d3 indicate the following.

d1 = Difference between order_estimated_delivery_date and order_purchase_timestamp

d2 = Difference between order_delivered_customer_date and order_purchase_timestamp

d3 = Difference between order_estimated_delivery_date and order_delivered_customer_date

2. Create columns:

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

To create the 2 columns time_to_delivery and diff_estimated_delivery the following query is executed.

The screenshot shows the Google BigQuery interface. On the left is the Explorer panel with a search bar and a list of saved queries. The main editor displays a SQL query. Below the editor, the 'Query results' section is visible, showing a table with 5 rows and 2 columns: 'time_to_delivery' and 'diff_estimated_delivery'. The results are as follows:

Row	time_to_delivery	diff_estimated_delivery
1	30	12
2	30	-28
3	35	-16
4	30	-1
5	32	0

At the bottom of the results section, it indicates 'Results per page: 50' and '1 - 50 of 99441'.

SQL Query:

SELECT

DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY)
AS time_to_delivery,

DATE_DIFF(order_delivered_customer_date,order_estimated_delivery_date,
DAY) **AS** diff_estimated_delivery

FROM `target-sclr-project-1.1.Orders`

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

Mean of freight grouped by state .

The screenshot shows the Google BigQuery interface. On the left is the Explorer panel with a search bar and a list of saved queries. The query 'AVG FREIGHT VALUE' is selected. The main editor shows a SQL query that calculates the average freight value by customer state. The query results are displayed in a table with columns 'Row', 'MEAN_FREIGHT_VALUE', and 'customer_state'. The results are ordered by the mean freight value in descending order.

```
1 SELECT
2   AVG (oi.freight_value) as MEAN_FREIGHT_VALUE,
3   c.customer_state
4 FROM `target-sclr-project-1.1.Order_Items` oi
5 JOIN `target-sclr-project-1.1.Orders` o
6   ON oi.order_id=o.order_id
7 JOIN `target-sclr-project-1.1.Customers` c
8   ON c.customer_id = o.customer_id
9 GROUP BY c.customer_state ORDER BY Mean_Freight_Value DESC
10
```

Row	MEAN_FREIGHT_VALUE	customer_state
1	42.9844230...	RR
2	42.7238039...	PB
3	41.0697122...	RO
4	40.0733695...	AC
5	39.1479704...	PI

SQL Query :

```
SELECT

AVG (oi.freight_value) as MEAN_FREIGHT_VALUE,

c.customer_state

FROM `target-sclr-project-1.1.Order_Items` oi

JOIN `target-sclr-project-1.1.Orders` o

on oi.order_id=o.order_id

JOIN `target-sclr-project-1.1.Customers` c

ON c.customer_id = o.customer_id

GROUP BY c.customer_state ORDER BY Mean_Freight_Value DESC
```


Mean of time to delivery grouped by state .

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer panel with a search bar and a list of saved queries. The main panel displays a SQL query and its results. The query calculates the average time to delivery for each customer state, ordered by descending average time.

SQL Query:

```
1 SELECT
2   AVG (DATE_DIFF(o.order_delivered_customer_date,o.order_purchase_timestamp,DAY)) AS Avg_time_to_delivery,
3   c.customer_state
4 FROM `target-sclr-project-1.1.Orders` o
5 JOIN `target-sclr-project-1.1.Customers` c
6 ON o.customer_id = c.customer_id
7 GROUP BY c.customer_state ORDER BY Avg_time_to_delivery desc
8
9
```

Query results:

Row	Avg_time_to...	customer_state
1	28.9756097...	RR
2	26.7313432...	AP
3	25.9862068...	AM
4	24.0403022...	AL
5	23.3160676	PA

Results per page: 50 1 - 27 of 27

SQL Query :

SELECT

AVG

(DATE_DIFF(o.order_delivered_customer_date,o.order_purchase_timestamp, DAY)) AS Avg_time_to_delivery,

c.customer_state

FROM `target-sclr-project-1.1.Orders` o

JOIN `target-sclr-project-1.1.Customers` c

ON o.customer_id = c.customer_id

GROUP BY c.customer_state ORDER BY Avg_time_to_delivery desc

Mean of difference in estimated delivery grouped by state.

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

```
1 SELECT
2 AVG (DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date,DAY)) AS Avg_diff_estimated_del
3 c.customer_state
4 FROM `target-sclr-project-1.1.Orders` o
5 JOIN `target-sclr-project-1.1.Customers` c
6 ON o.customer_id = c.customer_id
7 GROUP BY c.customer_state ORDER BY Avg_diff_estimated_delivery desc
```

The query results table is displayed below the query editor:

Row	Avg_diff_est...	customer_state
1	19.7625	AC
2	19.1316872...	RO
3	18.7313432...	AP
4	18.6068965...	AM
5	16.4146341	RR

SQL Query :

SELECT

AVG

(DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date,DAY))
AS Avg_diff_estimated_delivery,

c.customer_state

FROM `target-sclr-project-1.1.Orders` o

JOIN `target-sclr-project-1.1.Customers` c

ON o.customer_id = c.customer_id

GROUP BY c.customer_state ORDER BY Avg_diff_estimated_delivery desc

4.Sort the data to get the following:

1. Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5

Top 5 states with highest average freight value.

The screenshot shows the Google Cloud BigQuery interface. On the left, the 'Explorers' sidebar lists saved queries, with 'AVG FREIGHT VALUE' selected. The main editor displays a SQL query to find the top 5 states by average freight value. The query results are shown in a table with columns for Row, MEAN_FREIGHT_VALUE, and customer_state. The results show that Roraima (RR) has the highest average freight value, followed by Pernambuco (PB), Rio de Janeiro (RJ), Acre (AC), and Piauí (PI).

```
1 SELECT
2   AVG (oi.freight_value) as MEAN_FREIGHT_VALUE,
3   c.customer_state
4 FROM `target-sclr-project-1.1.Order_Items` oi
5 JOIN `target-sclr-project-1.1.Orders` o
6   ON oi.order_id=o.order_id
7 JOIN `target-sclr-project-1.1.Customers` c
8   ON c.customer_id = o.customer_id
9 GROUP BY c.customer_state ORDER BY Mean_Freight_Value DESC LIMIT 5
```

Row	MEAN_FREIGHT_VALUE	customer_state
1	42.9844230...	RR
2	42.7238039...	PB
3	41.0697122...	RJ
4	40.0733695...	AC
5	39.1479704...	PI

SQL Query :

```
SELECT
```

```
AVG (oi.freight_value) as MEAN_FREIGHT_VALUE,
```

```
c.customer_state
```

```
FROM `target-sclr-project-1.1.Order_Items` oi
```

```
JOIN `target-sclr-project-1.1.Orders` o
```

```
on oi.order_id=o.order_id
```

```
JOIN `target-sclr-project-1.1.Customers` c
```

```
ON c.customer_id = o.customer_id
```

```
GROUP BY c.customer_state ORDER BY Mean_Freight_Value DESC LIMIT 5
```

From the results it can be seen that the state Roraima has the highest average freight value .

Top 5 states with lowest average freight value.

The screenshot shows a data analytics interface with a sidebar on the left containing a search bar and a list of saved queries. The main panel displays a SQL query and its results. The query calculates the average freight value for each customer state, joining data from 'target-sclr-project-1.1.Order_Items', 'target-sclr-project-1.1.Orders', and 'target-sclr-project-1.1.Customers'. The results table shows the top 5 states with the lowest average freight value.

```
1 SELECT
2 AVG (oi.freight_value) as MEAN_FREIGHT_VALUE,
3 c.customer_state
4 FROM `target-sclr-project-1.1.Order_Items` oi
5 JOIN `target-sclr-project-1.1.Orders` o
6 ON oi.order_id=o.order_id
7 JOIN `target-sclr-project-1.1.Customers` c
8 ON c.customer_id = o.customer_id
9 GROUP BY c.customer_state ORDER BY Mean_Freight_Value asc LIMIT 5
10
```

Row	MEAN_FREI...	customer_state
1	15.1472753...	SP
2	20.5316515...	PR
3	20.6301668...	MG
4	20.9609239...	RJ
5	21.0413549...	DF

The state with the lowest average freight value is the state of Sao Paulo.

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

Top 5 states with highest average time to delivery

The following query provides the top 5 states with highest average time to delivery .

The screenshot shows the same data analytics interface, but with a different query. This query calculates the average time to delivery for each customer state, using the 'DATE_DIFF' function to find the difference between the order delivery date and the purchase timestamp. The results table shows the top 5 states with the highest average time to delivery.

```
1 SELECT
2 AVG (DATE_DIFF(o.order_delivered_customer_date,o.order_purchase_timestamp,DAY)) AS Avg_time_to_deliver
3 c.customer_state
4 FROM `target-sclr-project-1.1.Orders` o
5 JOIN `target-sclr-project-1.1.Customers` c
6 ON o.customer_id = c.customer_id
7 GROUP BY c.customer_state ORDER BY Avg_time_to_delivery DESC LIMIT 5
8
9
```

Row	Avg_time_to...	customer_state
1	28.9756097...	RR
2	26.7313432...	AP
3	25.9862068...	AM
4	24.0403022...	AL
5	23.3160676...	PA

SQL Query :

```
SELECT

    AVG

    (DATE_DIFF(o.order_delivered_customer_date,o.order_purchase_timestamp,DAY))
    AS Avg_time_to_delivery,

    c.customer_state

FROM `target-sclr-project-1.1.Orders` o

JOIN `target-sclr-project-1.1.Customers` c

ON o.customer_id = c.customer_id

GROUP BY c.customer_state ORDER BY Avg_time_to_delivery DESC LIMIT 5
```

From the results it can be seen that the state Roraima has the highest average time to delivery .

Top 5 states with lowest average time to delivery

The screenshot displays a data analytics application interface. On the left is an 'Explorer' panel with a search bar and a list of saved queries. The main area shows a SQL query editor with the following code:

```
1 SELECT
2   AVG (DATE_DIFF(o.order_delivered_customer_date,o.order_purchase_timestamp,DAY)) AS Avg_time_to_delivery,
3   c.customer_state
4 FROM `target-sclr-project-1.1.Orders` o
5 JOIN `target-sclr-project-1.1.Customers` c
6 ON o.customer_id = c.customer_id
7 GROUP BY c.customer_state ORDER BY Avg_time_to_delivery ASC LIMIT 5
8
9
```

Below the query editor, the 'Query results' section is visible, showing a table with 5 rows of data. The table has columns for 'Row', 'Avg_time_to_delivery', and 'customer_state'. The results are ordered by 'Avg_time_to_delivery' in ascending order.

Row	Avg_time_to_delivery	customer_state
1	8.29806148...	SP
2	11.5267113...	PR
3	11.5438132...	MG
4	12.5091346...	DF
5	14.4795601...	SC

The interface also includes a 'PERSONAL HISTORY' and 'PROJECT HISTORY' section at the bottom.

SQL Query :

```
SELECT

AVG

(DATE_DIFF(o.order_delivered_customer_date,o.order_purchase_timestamp,DAY))
AS Avg_time_to_delivery,

c.customer_state

FROM `target-sclr-project-1.1.Orders` o

JOIN `target-sclr-project-1.1.Customers` c

ON o.customer_id = c.customer_id

GROUP BY c.customer_state ORDER BY Avg_time_to_delivery ASC LIMIT 5
```

From the results it can be seen that the state with the lowest average time to delivery is the state of Sao Paulo.

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

Top 5 states where delivery is really fast compared to estimated date

The screenshot displays a data analytics tool interface. On the left, the 'Explorer' panel lists various saved queries, with 'Avg_diff_estimated_del...' selected. The main panel shows a SQL query being executed. The query calculates the average difference between the estimated delivery date and the actual delivered date for the top 5 states. The results are shown in a table with columns 'Avg_diff_est...' and 'customer_state'.

Query results table:

Row	Avg_diff_est...	customer_state
1	19.7625000...	AC
2	19.1316872...	RO
3	18.7313432...	AP
4	18.6068965...	AM
5	16.4146341...	RR

SQL Query :

SELECT

AVG

(DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date,DAY))

AS Avg_diff_estimated_delivery,

c.customer_state

FROM `target-sclr-project-1.1.Orders` o

JOIN `target-sclr-project-1.1.Customers` c

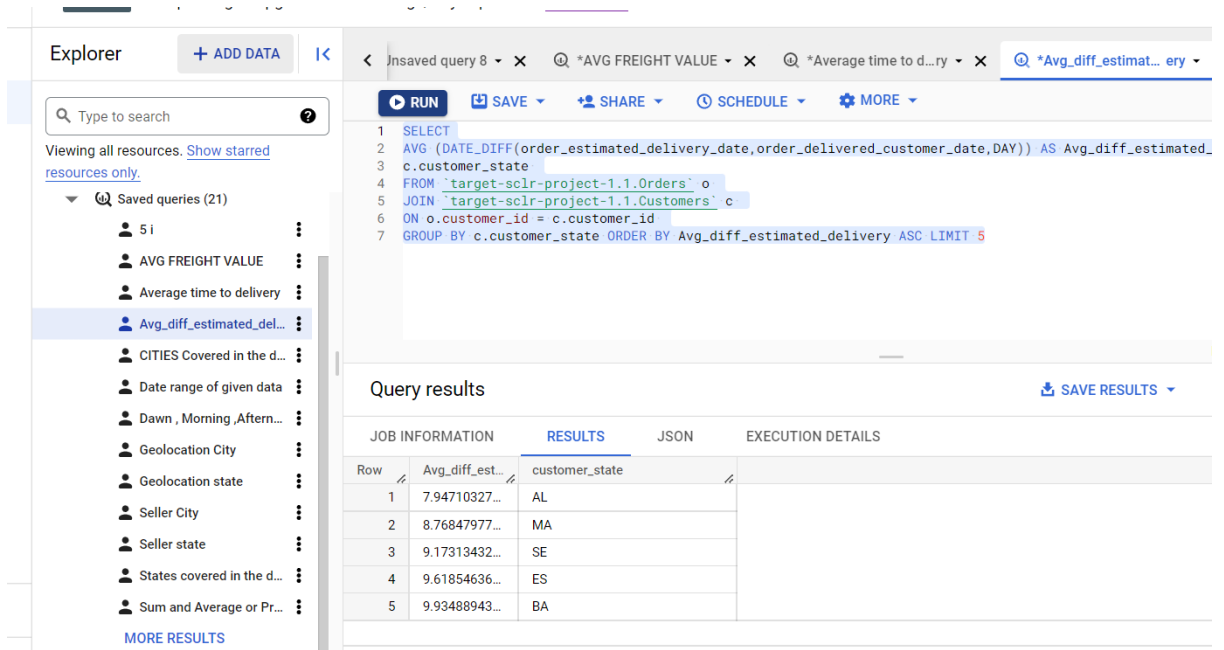
ON o.customer_id = c.customer_id

GROUP BY c.customer_state ORDER BY Avg_diff_estimated_delivery DESC LIMIT 5

From the result it can be seen that the state where delivery is really fast compared to estimated date is Acre .

Top 5 states where delivery is really slow compared to estimated date

The following query provides the list of top 5 states where delivery is really slow.



The screenshot shows a SQL query editor interface. On the left is an 'Explorer' panel with a search bar and a list of saved queries. The main editor displays the following SQL query:

```
1 SELECT
2   AVG (DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date,DAY)) AS Avg_diff_estimated_
3   c.customer_state
4 FROM `target-sclr-project-1.1.Orders` o
5 JOIN `target-sclr-project-1.1.Customers` c
6 ON o.customer_id = c.customer_id
7 GROUP BY c.customer_state ORDER BY Avg_diff_estimated_delivery ASC LIMIT 5
```

Below the query editor, the 'Query results' section is visible, showing a table with the following data:

Row	Avg_diff_est...	customer_state
1	7.94710327...	AL
2	8.76847977...	MA
3	9.17313432...	SE
4	9.61854636...	ES
5	9.93488943...	BA

SQL Query :

SELECT

AVG

(DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date,DAY))

AS Avg_diff_estimated_delivery,

c.customer_state

FROM `target-sclr-project-1.1.Orders` o

JOIN `target-sclr-project-1.1.Customers` c

ON o.customer_id = c.customer_id

GROUP BY c.customer_state ORDER BY Avg_diff_estimated_delivery ASC LIMIT 5

From the result it can be seen that the state Alagoas is the state where the delivery is the slowest compared to estimated date

6.Payment type analysis:

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

The following query provides the month over month count of orders for different payment types .

The screenshot shows a SQL query editor interface. On the left is an 'Explorer' pane with a search bar and a list of saved queries. The main area displays a SQL query that extracts year and month from the order purchase timestamp, counts orders by month and payment type, and filters for 'delivered' orders. Below the query editor, the 'Query results' section shows a table with columns for Row, Year, Month, OrderCount, and payment_type. The results show data for the year 2016 across five rows, representing different months and payment types.

Row	Year	Month	OrderCount	payment_type
1	2016	10	209	credit_card
2	2016	10	20	voucher
3	2016	10	2	debit_card
4	2016	10	51	UPI
5	2016	12	1	credit_card

SQL Query :

```
SELECT EXTRACT(Year FROM o.order_purchase_timestamp) as Year,
EXTRACT(Month FROM o.order_purchase_timestamp) as Month,
COUNT(o.order_id) as OrderCount ,
p.payment_type
FROM `target-sclr-project-1.1.Orders` o
JOIN `target-sclr-project-1.1.Payments` p
ON p.order_id = o.order_id
WHERE order_status='delivered'
GROUP BY Year,Month,p.payment_type ORDER BY Year,Month
```

By the running the same query and ordering the result by OrderCount in descending order and limiting the result to 10 it can be seen that the most used payment type is credit card as all the payment type for the top 10 results belong to the category of credit card .

The screenshot displays a data analytics platform interface. On the left is the 'Explorer' panel with a search bar and a list of saved queries. The main area shows a SQL query editor with a query that filters for 'delivered' orders and groups by year, month, and payment type, ordered by OrderCount. Below the editor, the 'Query results' section shows a table with 6 rows of data. The table has columns for Row, Year, Month, OrderCount, and payment_type. All payment types listed are 'credit_card'. At the bottom, there are tabs for 'PERSONAL HISTORY' and 'PROJECT HISTORY'.

Explorer

Viewing all resources. [Show starred resources only.](#)

Saved queries (21)

- 5 i
- AVG FREIGHT VALUE
- Average time to delivery
- Avg_diff_estimated_del...
- CITIES Covered in the d...
- Date range of given data
- Dawn , Morning ,Aftern...
- Geolocation City
- Geolocation state
- Seller City
- Seller state
- States covered in the d...
- Sum and Average or Pr...

MORE RESULTS

1

PERSONAL HISTORY PROJECT HISTORY

Query results

Row	Year	Month	OrderCount	payment_type
1	2017	11	5716	credit_card
2	2018	3	5526	credit_card
3	2018	5	5398	credit_card
4	2018	1	5368	credit_card
5	2018	4	5341	credit_card
6	2018	2	5114	credit_card

SQL Query :

```
SELECT EXTRACT(Year FROM o.order_purchase_timestamp) as Year,

EXTRACT(Month FROM o.order_purchase_timestamp) as Month,

COUNT(o.order_id) as OrderCount ,

p.payment_type

FROM `target-sclr-project-1.1.Orders` o

JOIN `target-sclr-project-1.1.Payments` p

ON p.order_id = o.order_id

WHERE order_status='delivered'

GROUP BY Year,Month,p.payment_type ORDER BY OrderCount DESC LIMIT 10
```

2. Distribution of payment instalments and count of orders

The following query provides the distribution of payment instalments and the respective count of orders .

The screenshot displays a data analytics platform interface. On the left, the 'Explorer' panel shows a list of saved queries, with 'Avg_diff_estimated_del...' selected. The main panel shows a SQL query editor with the following code:

```
1 SELECT
2 COUNT(o.order_id) as OrderCount ,
3 p.payment_installments
4 FROM `target-sclr-project-1.1.Orders` o
5 JOIN `target-sclr-project-1.1.Payments` p
6 ON p.order_id = o.order_id
7 WHERE order_status='delivered'
8 GROUP BY p.payment_installments ORDER BY OrderCount DESC
```

Below the query editor, the 'Query results' section is visible, showing a table with the following data:

Row	OrderCount	payment_in...
1	50929	1
2	12075	2
3	10164	3
4	6891	4
5	5150	10

The interface also includes a 'PERSONAL HISTORY' and 'PROJECT HISTORY' section at the bottom.

SQL Query :

```
SELECT  
  
COUNT(o.order_id) as OrderCount ,  
  
p.payment_installments  
  
FROM `target-sclr-project-1.1.Orders` o  
  
JOIN `target-sclr-project-1.1.Payments` p  
  
ON p.order_id = o.order_id  
  
WHERE order_status='delivered'  
  
GROUP BY p.payment_installments ORDER BY OrderCount DESC
```

From the results it can be seen that the instalment type of 1 has the highest orders with the count of orders as 50929 followed by instalments of 2 , 3 , 4 and 10 in the top 5 category .

This concludes the submission of the project .