

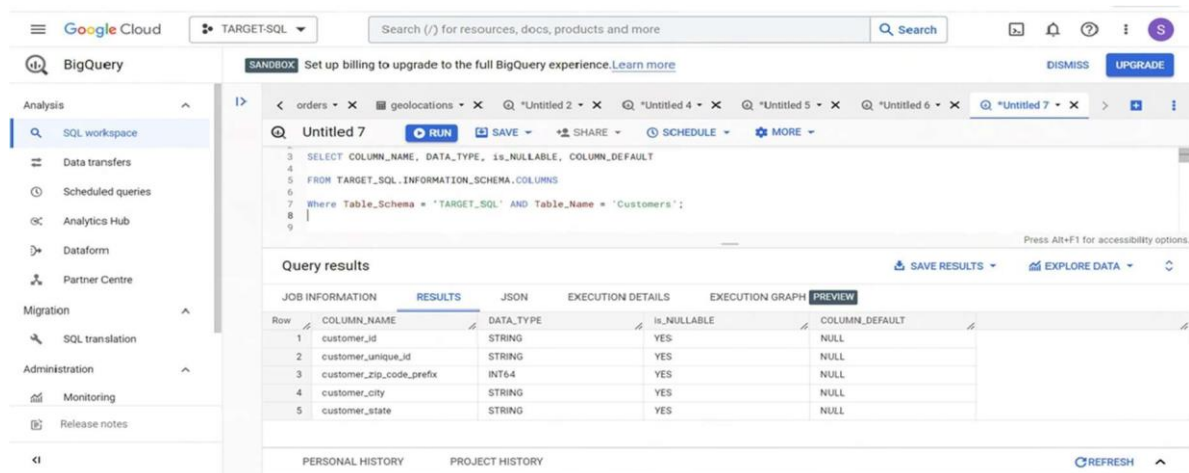
Business case: Target SQL

1(a)-Data type of columns in a table

- Datatype of column in a customer table

Query-

```
SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'Customers';
```



The screenshot shows the Google Cloud BigQuery console. A query is executed in the 'Sandbox' environment. The query results are displayed in a table with 5 columns: COLUMN_NAME, DATA_TYPE, is_NULLABLE, and COLUMN_DEFAULT. The results show 5 rows of data for the 'Customers' table.

Row	COLUMN_NAME	DATA_TYPE	is_NULLABLE	COLUMN_DEFAULT
1	customer_id	STRING	YES	NULL
2	customer_unique_id	STRING	YES	NULL
3	customer_zip_code_prefix	INT64	YES	NULL
4	customer_city	STRING	YES	NULL
5	customer_state	STRING	YES	NULL

Insights - Customer table has the information about the customer and it has 5 column and all the column stores the different type of information about the customers.

- Customer_id** - Data type of this column is **STRING** and it stores the id of customer who purchase. It can be null or default value of this column is null.
- Customer_unique_id** - Data type of this column is **STRING** and it stores id of the customer created an account on Target so this is the **UNIQUE** value column because every customer unique_id.
- Customer_zip_code_prefix** - Data type of this column is **INTEGER(64)** and it stores the pin code of customer.
- Customer_city** - Data type of this column is **STRING** and it stores the city from where order placed.
- Customer_state** - Data type of this column is **STRING** and it stores the state code where order has been placed.

Recommendations - After analysing the structure of this table my suggestion is that change the column customer_zip_code_prefix data type int(64) to int because customer pin code is not more than 5 digits and int(64) is use for very large number if you use int(64) then it creates performance issue

- Data type of column in a Geolocation table

Query-

```
SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'geolocations';
```

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

```
-- Data type of column in a geolocations
SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'geolocations';
-- Data type of column in a order table
```

The query results are displayed in a table with the following columns: COLUMN_NAME, DATA_TYPE, is_NULLABLE, and COLUMN_DEFAULT. The results show 5 rows of data for the 'geolocations' table.

Row	COLUMN_NAME	DATA_TYPE	is_NULLABLE	COLUMN_DEFAULT
1	geolocation_zip_code_prefix	INT64	YES	NULL
2	geolocation_lat	FLOAT64	YES	NULL
3	geolocation_lng	FLOAT64	YES	NULL
4	geolocation_city	STRING	YES	NULL
5	geolocation_state	STRING	YES	NULL

Insights- Geolocation table stores the information about the customer exact location like longitude, latitude, city and state and it has 5 columns and every column stores different type of information.

1. **geolocation_zip_code_prefix** - Data type of this column is **INTEGER(64)** and it stores the pin code of customer.
2. **geolocation_lat** - Data type of this column is **FLOAT(64)** and it stores the latitude of customer location.
3. **geolocation_lng** - Data type of this column is **FLOAT(64)** and it stores the longitude of the customer location.
4. **geolocation_city** - Data type of this column is **STRING** and it stores the city of customer.
5. **geolocation_state** - Data type of this location is **STRING** and it stores the state code of the customer.

Recommendations - After analysing the structure of this table my suggestion is that change the column customer_zip_code_prefix data type int(64) to int because customer pin code is not more than 5 digits and int(64) is use for very large number if you use int(64) then it creates performance issue in reading and writing both.

- Datatype of column in an orders table

Query-

```
SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
```

```
FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
```

```
Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'orders';
```

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

```
1 SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
2 FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
3
4 Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'orders';
```

The query results are displayed in a table with the following columns: COLUMN_NAME, DATA_TYPE, is_NULLABLE, and COLUMN_DEFAULT. The results show 8 rows of data for the 'orders' table.

Row	COLUMN_NAME	DATA_TYPE	is_NULLABLE	COLUMN_DEFAULT
1	order_id	STRING	YES	NULL
2	customer_id	STRING	YES	NULL
3	order_status	STRING	YES	NULL
4	order_purchase_timestamp	TIMESTAMP	YES	NULL
5	order_approved_at	TIMESTAMP	YES	NULL
6	order_delivered_carrier_date	TIMESTAMP	YES	NULL
7	order_delivered_customer_date	TIMESTAMP	YES	NULL
8	order_estimated_delivery_date	TIMESTAMP	YES	NULL

Insights- orders table stores the delivery related information, it has 8 column and all the column stores different type of information about order delivery.

1. **order_id** - Data type of this column is **STRING** and every order has unique id so this is the unique id column.
2. **customer_id** - Data type of this column is **STRING** and it stores the customer_id of customer who made purchase, it can be duplicate.
3. **order_status** - Data type of this column is **STRING** and it stores the status of the order made i.e., delivered, shipped etc.
4. **order_purchase_timestamp** - Data type of this column is **TIMESTAMP** and it stores the timestamp of that when order get placed.
5. **order_approved_at** - Data type of this column is **TIMESTAMP** and it stores the timestamp when order gets approved from sellerend.
6. **order_delivered_carrier_date** - Data type of this column is **TIMESTAMP** and it stores the timestamp when shipped orders get delivered to the customer.
7. **order_delivered_customer_date**- Data type of this column is **TIMESTAMP** and it stores the timestamp of that when order got received by customer.
8. **order_estimated_delivery_date**- Data type of this column is **TIMESTAMP** and it stores the timestamp of when it will delivered on or before.

- Data type of column in an order items table

Query-

```
SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'order-items';
```

The screenshot shows the Google Cloud BigQuery interface. On the left is the navigation menu with options like SQL workspace, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Centre, Migration, SQL translation, Administration, Monitoring, and Release notes. The main area is divided into an Explorer on the left and a query editor on the right. The Explorer shows a project named 'target-sql-382919' with a folder 'TARGET_SQL' containing tables like Customers, geolocations, order-items, order-reviews, orders, payments, products, and sellers. The query editor on the right shows a query named 'Untitled 7' with the following SQL code:

```
1 SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
2 FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
3
4 Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'order-items';
```

Below the query editor, the 'Query results' section is visible, showing a table with 7 columns: JOB INFORMATION, RESULTS, JSON, EXECUTION DETAILS, EXECUTION GRAPH, and PREVIEW. The 'RESULTS' tab is selected, displaying a table with 7 rows of data:

Row	COLUMN_NAME	DATA_TYPE	is_NULLABLE	COLUMN_DEFAULT
1	order_id	STRING	YES	NULL
2	order_item_id	INT64	YES	NULL
3	product_id	STRING	YES	NULL
4	seller_id	STRING	YES	NULL
5	shipping_limit_date	TIMESTAMP	YES	NULL
6	price	FLOAT64	YES	NULL
7	freight_value	FLOAT64	YES	NULL

Insights – order items table stores the information like customer_id of customer who placed order, id of order items, id of product, id of seller who supply the product, shipping limit date, price of item, delivery charge it has 7 columns and all the column stores different type of information about order items.

1. **order_id** - Data type of this column is **STRING** and every order has unique id so this is a unique id column, it can't be duplicate.
2. **order_item_id**- Data type of this column is **INT(64)** and every item has unique id so this is the unique_id column, it can't be duplicate.
3. **product_id** - Data type of this column is **STRING** and every product listed on target has unique id so this is also a unique id column, it can't be duplicate.
4. **seller_id** - Data type of this column is **STRING** and it is a unique id column because every seller listed on target who supplies the product has unique id, it can't be duplicate.
5. **shiping_limit_date**- Data type of this column is **TIMESTAMP** and it stores the timestamp of when product is shipped on or before.
6. **price** - Data type of this column is **FLOAT(64)** and it stores the price of the product.
7. **freight_value** - Data type of this column is **FLOAT(64)** and it stores the delievery charge of the product.

Recommendations - After analysing the structure of this table my suggestion is mention below-

1. change the column order_item_id data type int(64) to int because order item is not more is a very small number and int(64) is use for very large number if you use int(64) then itvcreates performance issue in reading and writing both.
 2. change the column price data type float(64) to float because order item has value up to 3 to 4 decimal point and float(64) is use for very huge precision if you use float(64) then it creates performance issue in reading and writing both.
 3. change the column freight price data type float(64) to float because order item has value up to 3 to 4 decimal point and float(64) is use for very huge precision if you use float(64) then it creates performance issue in reading and writing both.
- Data type of column in an order reviews table

Query-

```
SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'order-reviews';
```

The screenshot shows the Google Cloud BigQuery interface. The query editor on the right contains the following SQL query:

```
SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'order-reviews';
```

The query results are displayed in a table with the following columns: COLUMN_NAME, DATA_TYPE, is_NULLABLE, and COLUMN_DEFAULT. The results show 6 rows of data for the 'order-reviews' table.

Row	COLUMN_NAME	DATA_TYPE	is_NULLABLE	COLUMN_DEFAULT
1	review_id	STRING	YES	NULL
2	order_id	STRING	YES	NULL
3	review_score	INT64	YES	NULL
4	review_comment_title	STRING	YES	NULL
5	review_creation_date	TIMESTAMP	YES	NULL
6	review_answer_timestamp	TIMESTAMP	YES	NULL

Insights- order reviews table stores the information of reviews which has been given by customer, it has 6 column and all the column stores different type of information about order review.

1. **review_id** - Data type of this column is **STRING** and it stores the id of the review given on product by customer.
2. **order_id** - Data type of this column is **STRING** and it stores the unique id of order made by customer.
3. **review_score** - Data type of this column is **INT(64)** and it stores the review score given by the customer for each order on the scale of 1–5.
4. **review_comment_title** - Data type of this column is **STRING** and it stores the comment given on every review.

5. **review_creation_date** - Data type of this column is **TIMESTAMP** and it stores the timestamp of when review has been given.
6. **review_answer_timestamp** - Data type of this column is **TIMESTAMP** and it stores the timestamp of when the review gets answered.

Recommendations - After analysing the structure of this table my suggestion is mention below

1. change the column review score data type int(64) to int because review score range is 1 to 5 which is very small number and int(64) is use for very large number if you use int(64) then it creates performance issue in reading and writing both.
- Data type of column in an order sellers table

Query-

```
SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'sellers';
```

The screenshot shows the Google Cloud BigQuery interface. The query editor on the right contains the following SQL query:

```
1 SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
2 FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
3 Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'sellers';
```

The query results are displayed in a table with the following columns: JOB INFORMATION, RESULTS, JSON, EXECUTION DETAILS, and EXECUTION GRAPH. The RESULTS tab is active, showing a table with 4 columns: COLUMN_NAME, DATA_TYPE, is_NULLABLE, and COLUMN_DEFAULT. The data is as follows:

Row	COLUMN_NAME	DATA_TYPE	is_NULLABLE	COLUMN_DEFAULT
1	seller_id	STRING	YES	NULL
2	seller_zip_code_prefix	INT64	YES	NULL
3	seller_city	STRING	YES	NULL
4	seller_state	STRING	YES	NULL

Insights - sellers table stores the information about sellers who is listed on Target site, it has 4 column and every store different type of information.

1. **seller_id** - Data type of this column is **STRING** and it stores the id of seller who listed on target site and every seller has unique id so it is the unique id column, it can't be duplicate.
2. **seller_zip_code_prefix** - Data type of this column is **INT(64)** and it stores the pin code of seller address.
3. **seller_city** - Data type of this column is **STRING** and it stores the city of the seller address.
4. **seller_state** - Data type of this column is **STRING** and it stores the state code of seller address.

Recommendations - After analysing the structure of this table my suggestion is mention below-

1. change the column seller_zip_code_prefix data type int(64) to int because customer pin code is not more than 5 digits and int(64) is use for very large number if you use int(64) then it creates performance issue.

- Datatype of column in an order payments table

Query-

```
SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'payments';
```

The screenshot shows the Google Cloud BigQuery interface. On the left is a navigation menu with options like Analysis, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Centre, Migration, SQL translation, Administration, Monitoring, and Release notes. The main area is divided into an Explorer pane on the left showing a project named 'target-sql-382919' with various tables like Customers, geolocations, order-items, order-reviews, orders, payments, products, and sellers. The central pane shows a query editor with the following SQL code:

```
1 SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
2 FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
3 WHERE Table_Schema = 'TARGET_SQL' AND Table_Name = 'payments';
```

The query results are displayed in a table with the following columns: JOB INFORMATION, RESULTS, JSON, EXECUTION DETAILS, EXECUTION GRAPH, and PREVIEW. The RESULTS tab is active, showing a table with 5 rows and 5 columns: COLUMN_NAME, DATA_TYPE, is_NULLABLE, and COLUMN_DEFAULT.

Row	COLUMN_NAME	DATA_TYPE	is_NULLABLE	COLUMN_DEFAULT
1	order_id	STRING	YES	NULL
2	payment_sequential	INT64	YES	NULL
3	payment_type	STRING	YES	NULL
4	payment_installments	INT64	YES	NULL
5	payment_value	FLOAT64	YES	NULL

Insights - payments table stores the information about payments, it has 4 column and every column store different type of information about payments.

1. **order_id** - Data type of this column is **STRING** and it stores the id of order and every order has unique id so it is a unique id column, it can't be null.
2. **payment_sequential** - Data type of this column is **INT(64)** and it store the payment sequence in case of EMI.
3. **payment_type** - Data type of this column is **STRING** and it store the information about payment mode.
4. **payment_installment** - Data type of this column is **INT(64)** and it stores number of instalments in case of EMI.
5. **payment_value** - Data type of this column is **FLOAT(64)** and it store the total payment of every order.

Recommendations- After analysing the structure of this table my suggestion is mention below.

1. change the column payment_sequential data type int(64) to int because order item is not more is a very small number and int(64) is use for very large number if you use int(64) then it creates performance issue in reading and writing both.

2. change the column payment_installment data type int(64) to int because order item is not more is a very small number and int(64) is use for very large number if you use int(64) then it creates performance issue in reading and writing both.
 3. change the column payment value data type float(64) to float because order item has value up to 3 to 4 decimal point and float(64) is use for very huge precision if you use float(64) then it creates performance issue in reading and writing both.
- Data type of column in an order products table

Query-

```
SELECT COLUMN_NAME, DATA_TYPE, is_NULLABLE, COLUMN_DEFAULT
FROM TARGET_SQL.INFORMATION_SCHEMA.COLUMNS
Where Table_Schema = 'TARGET_SQL' AND Table_Name = 'products';
```

Row	COLUMN_NAME	DATA_TYPE	is_NULLABLE	COLUMN_DEFAULT
1	product_id	STRING	YES	NULL
2	product_category	STRING	YES	NULL
3	product_name_length	INT64	YES	NULL
4	product_description_length	INT64	YES	NULL
5	product_photos_qty	INT64	YES	NULL
6	product_weight_g	INT64	YES	NULL
7	product_length_cm	INT64	YES	NULL
8	product_height_cm	INT64	YES	NULL
9	product_width_cm	INT64	YES	NULL

Insights - products table stores the information about product who is listed on Target site, it has 9 column and every column store different type of information about product.

1. **product_id** - Data type of this column is **STRING** and it stores the id of product which is listed on Target site and every product has unique id so this the unique id column.
2. **product_category** - Data type of this column is **STRING** and it stores the category of product which is listed on Target site.
3. **product_name_length** - Data type of this column is **INT(64)** and it stores the length of the product name.
4. **product_description_length** - Data type of this column is **INT(64)** and it store the length of description given about product.
5. **product_photos_qty** - Data type of this column is **INT(64)** and it store the count of photos available for particular product.
6. **product_weight_g** - Data type of this column is **INT(64)** and it store the weight of the product in gram.

7. **product_length_cm** - Data type of this column is **INT(64)** and it store the length of the product in centimetre.
8. **product_height_cm** - Data type of this column is **INT(64)** and it store the height of the product in centimetre.
9. **product_width_cm** - Data type of this column is **INT(64)** and it store the width of the product in centimetre.

Recommendations- After analysing the structure of this table my suggestion is mention below.

1. Data type of column **product_name_length**, **product_description_length**, **product_photos_qty**, **product_weight_g**, **product_length_cm**, **product_height_cm**, **product_height_cm** and **product_width_cm** should be **INT** because in this column value are small and **INT(64)** is use for very big value.

1 (b)-Time period for which the data is given.

Query-

```
SELECT MIN(order_purchase_timestamp) as first_date,
MAX(order_purchase_timestamp) as last_date
FROM target-sql-382919.TARGET_SQL.orders;
```

The screenshot shows the Google Cloud BigQuery console. The query editor on the right contains the following SQL query:

```
SELECT MIN(order_purchase_timestamp) as first_date, MAX(order_purchase_timestamp) as last_date FROM
target-sql-382919.TARGET_SQL.orders;
```

The query has been executed, and the results are displayed in a table with two columns: **first_date** and **last_date**. The results show the time period from 2016-09-04 21:15:19 UTC to 2018-10-17 17:30:18 UTC.

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

Insight- Time period of given date is 4th September 2016 to 17th October 2018

1 (c)-Cities and state of customer order during given period.

Query-

```
SELECT c.customer_city, c.customer_state
FROM target-sql-382919.TARGET_SQL.Customers as c
INNER JOIN
target-sql-382919.TARGET_SQL.orders as o ON c.customer_id = o.customer_id
GROUP BY c.customer_city, c.customer_state limit 10;
```

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

```
1 select c.customer_city,c.customer_state from target-sql-382919.TARGET_SQL.Customers as c
2 inner join
3 target-sql-382919.TARGET_SQL.orders as o
4 on c.customer_id = o.customer_id
5 group by c.customer_city,c.customer_state limit 10;
```

The query results table shows the following data:

Row	customer_city	customer_state
1	acu	RN
2	ico	CE
3	ipe	RS
4	ipu	CE
5	ita	SC
6	itu	SP
7	jau	SP
8	luz	MG
9	poa	SP
10	uba	MG

Insights- After analysing the customer state and customer city wise record from the data I find that the total no of city and state from where customer placed order is respectively 4119 and 27. One more thing I observe that the maximum order has given by state SP and minimum order has given by state RR.

2 (a) 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?

Query-

```
select extract(month from order_purchase_timestamp) as month,
extract(year from order_purchase_timestamp) as year,
count(order_id) as count_order from target-sql-382919.TARGET_SQL.orders
group by month, year order by year, month;
```

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

```
1 select extract(month from order_purchase_timestamp) as month,
2 extract(year from order_purchase_timestamp) as year,
3 count(order_id) as count_order from target-sql-382919.TARGET_SQL.orders
4 group by month, year order by year, month;
```

The query results table shows the following data:

Row	month	year	count_order
1	9	2016	4
2	10	2016	324
3	12	2016	1
4	1	2017	800
5	2	2017	1780
6	3	2017	2862
7	4	2017	2404
8	5	2017	3700
9	6	2017	3245
10	7	2017	4026
11	8	2017	4331
12	9	2017	4285
13	10	2017	4631

Query results:

Row	month	year	count_order
13	10	2017	4031
14	11	2017	7544
15	12	2017	5673
16	1	2018	7269
17	2	2018	6728
18	3	2018	7211
19	4	2018	6939
20	5	2018	6873
21	6	2018	6167
22	7	2018	6292
23	8	2018	6512
24	9	2018	16
25	10	2018	4

Insights - Yes there is a growing trend on E-commerce in Brazil and it is between January 2017 to August 2018. I can describe this scenario after calculating the month-to-month order count. After analysing there is no peaks at specific month but we can say that in summer and spring seasons number of order is slightly greater than other season.

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

Query-

```
select count(order_id) as count_order,
case
when extract(time from order_purchase_timestamp)
between "00:00:00" and "06:00:00" then "dawn (0-6)"

when extract(time from order_purchase_timestamp)
between "06:00:01" and "12:00:00" then "morning (6-12)"

when extract(time from order_purchase_timestamp)
between "12:00:01" and "16:00:00" then "afternoon (12-16)"

when extract(time from order_purchase_timestamp)
between "16:00:01" and "23:59:59" then "night (16-23:59:59)"

END as time_slot
from target-sql-382919.TARGET_SQL.orders
group by time_slot
order by count_order desc;
```

The screenshot shows the Google Cloud BigQuery console. On the left is a navigation menu with options like SQL workspace, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Centre, Migration, SQL translation, Administration, Monitoring, Capacity management, BI Engine, Policy tags, and Release notes. The main area is divided into an Explorer on the left and a query editor on the right. The Explorer shows a project named 'target-sql-382919' with a folder 'TARGET_SQL' containing subfolders like Customers, geolocations, order items, order reviews, orders, payments, products, and sellers. The query editor shows a query named 'Untitled 8' with the following SQL code:

```

1 select count(order_id) as count_order,
2 case
3 when extract(time from order_purchase_timestamp) between "00:00:00" and "06:00:00" then "dawn(0-6)"
4 when extract(time from order_purchase_timestamp) between "06:00:00" and "12:00:00" then "morning(6-12)"
5 when extract(time from order_purchase_timestamp) between "12:00:00" and "16:00:00" then "afternoon(12-16)"
6 when extract(time from order_purchase_timestamp) between "16:00:00" and "23:59:59" then "night(16-23:59:59)"
7 end as time_slot
8 from target-sql-382919.TARGET_SQL.orders
9 group by time_slot
10 order by count_order

```

Below the query editor, the 'Query results' section is visible, showing a table with 4 rows and 2 columns: 'count_order' and 'time_slot'.

Row	count_order	time_slot
1	4740	dawn(0-6)
2	22240	morning(6-12)
3	25537	afternoon(12-16)
4	46924	night(16-23:59:59)

Assumption- I assume that the time slot is

Dawn – (0-6)

Morning – (6-12)

Afternoon – (12-16)

Night or Evening – (16-23:59:59)

Insight - After analysing the data I find that in the Night timeslot(16-23:59:59) customer tends to buy more as compare to another timeslot.

3 (a)- Get month on month orders by states.

Query-

```

(select c.customer_state as state,
extract(year from o.order_purchase_timestamp) as year,
extract(month from o.order_purchase_timestamp) as month,
count(order_id) as count_order
from target-sql-382919.TARGET_SQL.Customers as c
inner join
target-sql-382919.TARGET_SQL.orders as o on
c.customer_id = o.customer_id
group by state, extract(year from o.order_purchase_timestamp),
extract(month from o.order_purchase_timestamp))
order by year, month, count_order desc limit 10;

```

Query results

Row	state	year	month	count_order
1	SP	2019	9	2
2	RS	2019	9	1
3	RR	2019	9	1
4	SP	2019	10	113
5	RJ	2019	10	56
6	MG	2019	10	40
7	RS	2019	10	24
8	PR	2019	10	19
9	SC	2019	10	11
10	GO	2019	10	9

Insight- After analysing the data I find that every month state **SP** gives more order as compare to other state.

3 (b) Distribution of customers across the states in Brazil.

Query-

```
select customer_state, count(customer_unique_id) as count_customer
from target-sql-382919.TARGET_SQL.Customers
group by customer_state
order by count_customer desc;
```

Query results

Row	customer_state	count_customer
1	SP	41746
2	RJ	12852
3	MG	11635
4	RS	5466
5	PR	5045
6	SC	3637
7	BA	3380
8	DF	2140
9	ES	2033
10	GO	2020

Insight-After analysing the data I find that the maximum and minimum number of customers who sign up in Target is belong from state **SP** and **RR** respectively.

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

Query-

```
select *, round(((Total_cost -  
  
lag(Total_cost) over (order by year)) / lag(Total_cost) over (order by year)) * 100, 2)  
  
percentage_increase from  
  
(select A.year, sum(p.payment_value) as Total_cost from TARGET_SQL.payments as p  
  
join  
  
(select order_id, extract (year from order_purchase_timestamp) as year,  
  
extract (month from order_purchase_timestamp) as month  
  
from TARGET_SQL.orders  
  
where extract(year from order_purchase_timestamp) in (2017, 2018) and  
  
extract(month from order_purchase_timestamp) between 01 and 08) as A  
  
on A.order_id = p.order_id  
  
group by A.year  
  
order by A.year)  
  
order by year;
```

The screenshot displays the Google Cloud BigQuery interface. On the left, the 'Explorer' pane shows the project 'target-sql-382919' with various tables like 'customers', 'geolocations', 'order-items', 'order-reviews', 'orders', 'payments', 'products', and 'sellers'. The main editor shows a SQL query that calculates the percentage increase in total cost from 2017 to 2018, filtered by months 01 to 08. Below the query editor, the 'Query results' section shows a table with 2 rows of data.

Row	year	Total_cost	percentage_incr
1	2017	3669022.12	null
2	2018	8694733.83	136.98

Insight - After analyzing the data I found that % increase in cost of order from 2017 to 2018 (include months between Jan to Aug) is increased by 136.98%

4 (b) Mean & Sum of price and freight value by customer state.

Query-

```
select c.customer_state as state,
round(avg(ot.freight_value),2) as avg_freight_value,
round(sum(ot.freight_value),2) as sum_freight_value,
round(sum(ot.price),2) as sum_price,
round(avg(ot.price),2) as avg_price,
round((sum(ot.freight_value)/sum(ot.price))*100,2)
as percentage_contribution_freight
from target-sql-382919.TARGET_SQL.Customers as c
inner join
target-sql-382919.TARGET_SQL.orders as o
on c.customer_id = o.customer_id
inner join
`target-sql-382919.TARGET_SQL.order-items` as ot
on o.order_id = ot.order_id group by c.customer_state
order by avg_freight_value;
```

The screenshot shows the Google Cloud BigQuery console. The query editor on the right contains the SQL query from the previous block. Below the editor, the 'Query results' section displays a table with 8 rows of data. The table has columns for state, avg_freight_value, sum_freight_value, sum_price, avg_price, and percentage_contribution_freight. The results are sorted by avg_freight_value in descending order.

Row	state	avg_freight_value	sum_freight_value	sum_price	avg_price	percentage_contribution_freight
1	SP	15.15	718723.07	5202955.05	109.65	13.81
2	PR	20.53	117891.68	480983.76	119.0	17.25
3	MS	20.43	270653.46	1585308.03	120.75	17.09
4	RJ	20.96	305589.31	1824092.67	125.12	16.75
5	DF	21.04	50425.5	302603.94	125.77	16.73
6	SC	21.47	89640.26	520553.34	134.65	17.22
7	RS	21.74	135522.74	790304.02	120.34	18.06
8	ES	22.06	48764.6	275037.31	121.91	18.09

Insight -

1. After analysing the data I find that the which state generating more revenue and which state generating less revenue for company.
2. After calculating the freight value is how much percentage of product price then I observe that in the 50% of state freight value is more than 20% of product value.

5 (a)- Calculate days between purchasing, delivering and estimated delivery.

Query-

```
select order_id, customer_id, order_purchase_timestamp,  
order_delivered_carrier_date, order_estimated_delivery_date,  
date_diff(order_delivered_carrier_date, order_purchase_timestamp, day)  
as time_to_delivery ,  
date_diff(order_estimated_delivery_date, order_purchase_timestamp, day)  
as estimate_time_to_delivery from  
target-sql-382919.TARGET_SQL.orders;
```

The screenshot shows the Google Cloud BigQuery console. A query is executed in the 'Untitled 8' workspace. The query is: `select order_id, customer_id, order_purchase_timestamp, order_delivered_carrier_date, order_estimated_delivery_date, date_diff(order_delivered_carrier_date, order_purchase_timestamp, day) as time_to_delivery, date_diff(order_estimated_delivery_date, order_purchase_timestamp, day) as estimate_time_to_delivery from target-sql-382919.TARGET_SQL.orders;` The query results are displayed in a table with 12 columns: `row`, `order_id`, `customer_id`, `order_purchase_timestamp`, `order_delivered_carrier_date`, `order_estimated_delivery_date`, `time_to_delivery`, and `estimate_time`. The table shows 12 rows of data. The 'time_to_delivery' column represents the difference in days between the order purchase timestamp and the order delivered carrier date. The 'estimate_time' column represents the difference in days between the order purchase timestamp and the order estimated delivery date.

row	order_id	customer_id	order_purchase_timestamp	order_delivered_carrier_date	order_estimated_delivery_date	time_to_delivery	estimate_time
1	888aac7ebcc37119725a0753	b50a774cd941f6d114eaf6	2017-12-09 10:16:45 UTC	2017-12-18 17:43:38 UTC	2018-01-29 00:00:00 UTC	9	50
2	790cd376891930ca05002f6b	53a76d42c239c712d9a2f67	2018-08-10 15:14:50 UTC	2018-08-13 13:44:00 UTC	2018-08-17 00:00:00 UTC	2	6
3	49db79430605805c341f547	9c9f8557e02418f69923f9e	2017-05-13 21:23:34 UTC	2017-05-20 07:43:42 UTC	2017-06-27 00:00:00 UTC	6	44
4	063b573089f80e516ab87df	285195a5b585842c23c1ef90	2016-10-07 19:17:00 UTC	2016-10-30 10:23:36 UTC	2016-12-01 00:00:00 UTC	22	54
5	a680e168d536ca72b23ad4	d7bed5fac093a413a216072ab	2016-10-05 01:47:40 UTC	2016-11-07 16:37:37 UTC	2016-12-01 00:00:00 UTC	33	56
6	45973912a49086800c3aeaf	912108a702f25999240a54c	2016-10-07 22:45:28 UTC	2016-10-26 13:18:16 UTC	2016-12-01 00:00:00 UTC	18	54
7	cd8f73529ca7a071f67f5e1	76c74aa02f735546a9818a	2016-10-05 16:57:30 UTC	2016-11-14 11:34:39 UTC	2016-12-01 00:00:00 UTC	39	56
8	ea020687125d8f5899831b0	b296edf58ac2180a4575bcb	2018-03-08 07:06:35 UTC	2018-03-09 17:01:37 UTC	2018-04-19 00:00:00 UTC	1	41
9	6f028cc570812af251aa42a1f	3a0e5f04a4a5d0a6030a3	2018-08-05 07:21:56 UTC	2018-08-06 13:21:00 UTC	2018-08-09 00:00:00 UTC	1	3
10	8733c86440c173e5242e6b80	c581230659c12a017b0b3a60	2018-08-05 17:00:00 UTC	2018-08-06 15:18:00 UTC	2018-08-09 00:00:00 UTC	0	3
11	98a0f05411c05a659f024b0d	0008a3af1b00c1380c89255	2018-07-03 19:59:42 UTC	2018-07-04 14:15:00 UTC	2018-08-20 00:00:00 UTC	0	47
12	34a981c2f20303a9a6603f42	d377598d2c347c09d1ef52ec4	2018-06-04 19:34:00 UTC	2018-06-06 15:47:00 UTC	2018-07-19 00:00:00 UTC	1	44

Insight- After analysing the data, I found the information about the order delivery of how much time order is taking to delivered and what is the estimated time to delivered.

5 (b)- Find time to delivery & diff estimated delivery.

Query-

```
select order_id, customer_id,  
date_diff(order_delivered_carrier_date, order_purchase_timestamp, day)  
as time_to_delivery,  
date_diff(order_estimated_delivery_date, order_delivered_carrier_date, day)  
as diff_estimated_delivery from `target-sql-382919.TARGET_SQL.orders`  
limit 10;
```

The screenshot shows the Google Cloud BigQuery interface. A query is executed, and the results are displayed in a table. The query is: `select order_id, customer_id, date_diff(order_delivered_carrier_date, order_purchase_timestamp, day) as time_to_delivery, date_diff(order_estimated_delivery_date, order_delivered_carrier_date, day) as diff_estimated_delivery from `target-sql-382919.TARGET_SQL.orders` limit 10;`

Row	order_id	customer_id	time_to_delivery	diff_estimated_delivery
1	7a4f75b8c7f409e541401a209...	7256c75605414021f8c8d5a...	3	23
2	350e4050331cc644c5d36f4...	4ee4f4bfc542546f422ca0eb...	1	13
3	85359091123f603c50b50cfe...	43844954f9980d107b54571...	1	13
4	dba502f0d3af4f6c33b1e04...	964a4d309b0c0c9e798b0d9...	1	13
5	90ab3e7d52544ec7cc3363c82...	705109f4216052b46472d9c...	1	13
6	f6e5d8102e8184cc5c0e3...	9af2372a1e49340278e7c1ef8...	3	23
7	102775799ecd908502425...	12402d65c4601689043a367...	5	26
8	6190e94857e1012983a27408...	5fca0763e3903996714524...	2	31
9	58ce513a35c740a811e8b07...	530641b4796d6a68c0f310856...	1	13
10	088683795a30505091152c4f...	58d9f01f8a3810f90540734f...	7	24

Insight- After analysing the data, I find the information about order delivery of how much time order is taking to delivered and what days difference between order delivered date and estimated delivery date. One more thing I seen in result which is first 5 order time_to_delivery and diff_estimated_delivery is null it shows that the after placing the order they get cancelled.

5(c) Group data by state, take mean of freight value, time to delivery, diff estimated delivery.

Query-

```
select c.customer_state, round(avg(ot.freight_value),2)
as mean_freight_value,
round(avg(date_diff(o.order_delivered_carrier_date,
o.order_purchase_timestamp,day)),2)
as mean_time_to_delivery,
round(avg(date_diff(o.order_estimated_delivery_date,
o.order_delivered_carrier_date, day)),2)
as mean_diff_estimated_delivery
from target-sql-382919.TARGET_SQL.Customers as c
join target-sql-382919.TARGET_SQL.orders as o on
c.customer_id = o.customer_id
join `target-sql-382919.TARGET_SQL.order-items` as ot
on ot.order_id = o.order_id group by c.customer_state;
```

Row	customer_state	mean_freight_value	mean_time_to_delivery	mean_diff_estimated_delivery_date
1	RN	35.65	3.2	28.56
2	CE	32.71	2.93	27.6
3	RS	21.74	2.8	24.98
4	SC	21.47	2.92	22.1
5	SP	15.15	2.72	15.68
6	MO	20.63	2.79	21.0
7	BA	26.56	2.89	25.77
8	RJ	20.96	2.9	22.72
9	GO	22.77	2.62	23.53
10	MA	38.26	3.4	26.61

Insight - After analysing the data, I find that the mean of freight value approx. range is between 20 to 40 in every state and time to delivery is approx. 3 days in every state. one more thing I observe that the estimated delivery date is too high as compare to order delivery date.

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

Query-

Top 5 states with highest average freight value.

```
select c.customer_state as state, round(avg(ot.freight_value),2)
as avg_freight_value
from target-sql-382919.TARGET_SQL.Customers as c
inner join
target-sql-382919.TARGET_SQL.orders as o
on c.customer_id = o.customer_id
inner join
`target-sql-382919.TARGET_SQL.order-items` as ot
on o.order_id = ot.order_id
group by c.customer_state
order by avg_freight_value desc limit 5;
```

Start your free trial with \$300 in credit. Don't worry - you won't be charged if you run out of credit. [Learn more](#)

Google Cloud TARGET-SQL Search (/) for resources, docs, products and more Search

BigQuery SANDBOX Set up billing to upgrade to the full BigQuery experience [Learn more](#) DISMISS UPGRADE

Analysis

- SQL workspace
- Data transfers
- Scheduled queries
- Analytics Hub
- Dataform
- Partner Centre

Migration

- SQL translation

Administration

- Monitoring
- Capacity management
- Release notes

Registration FORM...pdf

Untitled 8

```

1 select c.customer_state as state, round(avg(ot.freight_value),2) as avg_freight_value
2 from target-sql-382919.TARGET_SQL.Customers as c
3 inner join
4 target-sql-382919.TARGET_SQL.orders as o
5 on c.customer_id = o.customer_id
6 inner join
7 target-sql-382919.TARGET_SQL.order-items as ot
8 on o.order_id = ot.order_id
9 group by c.customer_state
10 order by avg_freight_value desc limit 5 ;
11

```

Query results

JOB INFORMATION RESULTS JSON EXECUTION DETAILS EXECUTION GRAPH PREVIEW

Row	state	avg_freight_value
1	RR	42.98
2	PR	42.72
3	RO	41.07
4	AC	40.07
5	PI	39.15

PERSONAL HISTORY PROJECT HISTORY REFRESH

Show all

Query-

Top 5 state with lowest average freight value

```

select c.customer_state as state, round(avg(ot.freight_value),2)
as avg_freight_value
from target-sql-382919.TARGET_SQL.Customers as c
inner join
target-sql-382919.TARGET_SQL.orders as o
on c.customer_id = o.customer_id
inner join
`target-sql-382919.TARGET_SQL.order-items` as ot
on o.order_id = ot.order_id
group by c.customer_state
order by avg_freight_value limit 5;

```

The screenshot shows the Google Cloud BigQuery console. A query is executed in a workspace named 'Untitled 8'. The query filters for the top 5 states based on average freight value. The results table shows the following data:

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

Insight- After analysing the data, I find that the top and bottom average freight value state is RR and SP.

5(e)- Top 5 states with highest/lowest average time to delivery.

Query-

Top 5 states with highest average time to delivery.

```
select c.customer_state as state,
round(avg(date_diff(o.order_delivered_carrier_date,
o.order_purchase_timestamp,day)),2)
as avg_time_to_delivery
from target-sql-382919.TARGET_SQL.Customers as c
inner join
target-sql-382919.TARGET_SQL.orders as o
on c.customer_id = o.customer_id
group by c.customer_state
order by avg_time_to_delivery desc limit 5;
```


Start your free trial with \$300 in credit. Don't worry - you won't be charged if you run out of credit. [Learn more](#)

Google Cloud TARGET-SQL Search (/) for resources, docs, products and more Search

BigQuery SANDBOX Set up billing to upgrade to the full BigQuery experience. [Learn more](#)

Analysis SQL workspace Data transfers Scheduled queries Analytics Hub Dataform Partner Centre Migration SQL translation Administration Monitoring Capacity management Release notes

Untitled 8 RUN SAVE SHARE SCHEDULE MORE

```

1 select c.customer_state as state,
2 round(avg(date_diff(o.order_delivered_carrier_date,o.order_purchase_timestamp,day)),2)
3 as avg_time_to_delivery
4 from target-sql-382919.TARGET_SQL.Customers as c
5 inner join
6 target-sql-382919.TARGET_SQL.orders as o
7 on c.customer_id = o.customer_id
8 group by c.customer_state
9 order by avg_time_to_delivery desc limit 5;

```

Query results

Row	state	avg_time_to_delivery
1	RR	4.02
2	SE	3.15
3	MA	3.13
4	RN	3.11
5	AP	3.04

PERSONAL HISTORY PROJECT HISTORY REFRESH

Registration FORM...pdf Show all

Query-

Top 5 states with lowest average time to delivery.

```

select c.customer_state as state,
round(avg(date_diff(o.order_delivered_carrier_date,
o.order_purchase_timestamp,day)),2)
as avg_time_to_delivery
from target-sql-382919.TARGET_SQL.Customers as c
inner join
target-sql-382919.TARGET_SQL.orders as o
on c.customer_id = o.customer_id
group by c.customer_state
order by avg_time_to_delivery limit 5;

```

Start your free trial with \$300 in credit. Don't worry - you won't be charged if you run out of credit. [Learn more](#)

Google Cloud TARGET-SQL Search (/) for resources, docs, products and more Search

BigQuery SANDBOX Set up billing to upgrade to the full BigQuery experience. [Learn more](#)

Analysis SQL workspace Data transfers Scheduled queries Analytics Hub Dataform Partner Centre Migration SQL translation Administration Monitoring Capacity management Release notes

Untitled 8 RUN SAVE SHARE SCHEDULE MORE

```

1 select c.customer_state as state,
2 round(avg(date_diff(o.order_delivered_carrier_date,o.order_purchase_timestamp,day)),2)
3 as avg_time_to_delivery
4 from target-sql-382919.TARGET_SQL.Customers as c
5 inner join
6 target-sql-382919.TARGET_SQL.orders as o
7 on c.customer_id = o.customer_id
8 group by c.customer_state
9 order by avg_time_to_delivery limit 5;

```

Query results

Row	state	avg_time_to_delivery
1	AM	2.31
2	RO	2.33
3	GO	2.68
4	MS	2.68
5	SP	2.68

PERSONAL HISTORY PROJECT HISTORY REFRESH

Registration FORM...pdf Show all

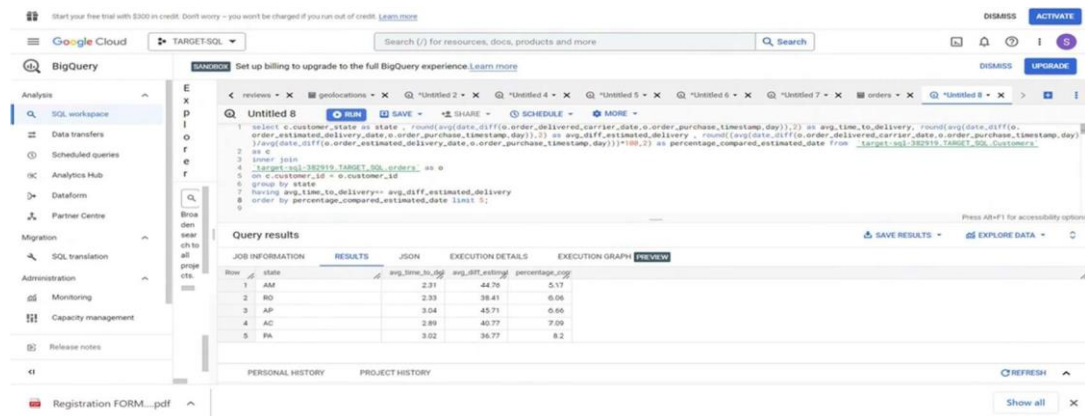
Insight - After analysing the problem statement, I found that the avg time to delivery is approx lies between 2 to 4 days.

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

Query-

Top 5 states where delivery is so fast as compared to estimated date.

```
select c.customer_state as state,
round(avg(date_diff(o.order_delivered_carrier_date,
o.order_purchase_timestamp, day)),2) as avg_time_to_delivery,
round(avg(date_diff(o.order_estimated_delivery_date,
o.order_purchase_timestamp, day)),2) as avg_diff_estimated_delivery,
round((avg(date_diff(o.order_delivered_carrier_date,
o.order_purchase_timestamp, day))/
avg(date_diff(o.order_estimated_delivery_date,
o.order_purchase_timestamp, day))) *100,2)
as percentage_compared_estimated_date
from `target-sql-382919.TARGET_SQL.Customers` as c
inner join
`target-sql-382919.TARGET_SQL.orders` as o
on c.customer_id = o.customer_id
group by state
having avg_time_to_delivery<= avg_diff_estimated_delivery
order by percentage_compared_estimated_date limit 5;
```



The screenshot shows the Google Cloud BigQuery interface. The SQL query is displayed in the editor, and the results are shown in a table below. The table has columns: state, avg_time_to_delivery, avg_diff_estimated_delivery, and percentage_avg. The results are sorted by percentage_avg in descending order, showing the top 5 states.

state	avg_time_to_delivery	avg_diff_estimated_delivery	percentage_avg
AM	2.31	44.76	5.17
RD	2.33	38.41	6.06
AP	3.04	45.71	6.66
AC	2.89	40.72	7.09
Pa	3.02	36.77	8.2

Query-

Top 5 states where delivery is not so fast as compared to estimated date.

```
select c.customer_state as state,
round(avg(date_diff(o.order_delivered_carrier_date,
o.order_purchase_timestamp, day)),2) as avg_time_to_delivery,
round(avg(date_diff(o.order_estimated_delivery_date,
o.order_purchase_timestamp, day)),2) as avg_diff_estimated_delivery,
round((avg(date_diff(o.order_delivered_carrier_date,
o.order_purchase_timestamp, day))/
avg(date_diff(o.order_estimated_delivery_date,
o.order_purchase_timestamp, day))) *100,2)
as percentage_compared_estimated_date
from `target-sql-382919.TARGET_SQL.Customers` as c
inner join
`target-sql-382919.TARGET_SQL.orders` as o
on c.customer_id = o.customer_id
group by state
having avg_time_to_delivery<= avg_diff_estimated_delivery
order by percentage_compared_estimated_date limit 5;
```

Insight- After analysing the data, I found that the estimated days for delivery is too long as compared to delivery date it is 5 to 15 percentage of estimated day for every state.

6 (a) Month over Month count of orders for different payment types.

Query-

```
select extract(month from o.order_purchase_timestamp) as month,
extract(year from order_purchase_timestamp) as year, p.payment_type
as payment_type, count(o.order_id) as count_order
from `target-sql-382919.TARGET_SQL.orders` as o
inner join
`target-sql-382919.TARGET_SQL.payments` as p
on o.order_id = p.order_id
group by year, month, p.payment_type
order by year, month;
```

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

```

count_order from `target-sql-382919.TARGET_SQL.orders` as o
inner join
`target-sql-382919.TARGET_SQL.payments` as p
on o.order_id = p.order_id
group by year,month,p.payment_type order by year,month limit 10;

```

The query results are displayed in a table with the following columns: Row, month, year, payment_type, and count_order. The results show the top 10 rows of data, grouped by year and month, ordered by payment type.

Row	month	year	payment_type	count_order
1	9	2016	credit_card	3
2	10	2016	debit_card	2
3	10	2016	voucher	23
4	10	2016	credit_card	254
5	10	2016	UPI	63
6	12	2016	credit_card	1
7	1	2017	UPI	197
8	1	2017	voucher	61
9	1	2017	credit_card	583
10	1	2017	debit_card	9

Insight- After calculating the month-to-month count of order for different payment types, I found that every month maximum payment done by credit card.

6 (b) Count of orders based on the no. of payment instalments.

Query-

```

select extract(month from o.order_purchase_timestamp) as month,
extract(year from order_purchase_timestamp) as year,
p.payment_installments as payment_installments,
count(o.order_id) as count_order
from `target-sql-382919.TARGET_SQL.orders` as o
inner join
`target-sql-382919.TARGET_SQL.payments` as p
on o.order_id = p.order_id
group by year,month,p.payment_installments order by year,month;

```

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

```

select extract(month from o.order_purchase_timestamp) as month, extract(year from order_purchase_timestamp) as year, p.payment_installments as payment_installments, count(o.order_id) as count_order from `target-sql-382919.TARGET_SQL.orders` as o
inner join
`target-sql-382919.TARGET_SQL.payments` as p
on o.order_id = p.order_id
group by year,month,p.payment_installments order by year,month,p.payment_installments limit 10;

```

The query results are displayed in a table with the following columns: Row, month, year, payment_installments, and count_order. The results show the top 10 rows of data, grouped by year, month, and payment installments, ordered by year, month, and payment installments.

Row	month	year	payment_installments	count_order
1	9	2016	1	1
2	9	2016	2	1
3	9	2016	3	1
4	10	2016	1	144
5	10	2016	2	30
6	10	2016	3	43
7	10	2016	4	26
8	10	2016	5	20
9	10	2016	6	18
10	10	2016	7	13

Insight - After calculating the month-to-month count of order based on payment instalments then I found that every month maximum order done by onetime payment.

Recommendations- After analysing all the problem statements I want to give some suggestion to the company so that company will grow more are following below.

1. Give more exciting offer like reward point, coupon code to the customer who gives more order so that customer will attract more and placed more orders.
2. Customer who gives lesser order, give them offers like Easy EMI payment, no cost EMI, pay later so that they will place more orders.
3. I observe in the season of winter and Autumn order is less as compared to other season so give special offers like buy 2 get 1 free, offers on payment on product for this season so that sales will grow more on this season.
4. Try to minimise the difference between estimated delivery date and delivered date because estimated delivery time was very high as compared to delivered date.
5. Try to minimise the freight value as much as possible.