

What does 'good' look like?

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

```
SELECT * FROM SCALER_SQL_PROJECT_1.INFORMATION_SCHEMA.COLUMNS;
```

The screenshot shows the Google Cloud BigQuery interface. The query editor displays the query: `SELECT * FROM SCALER_SQL_PROJECT_1.INFORMATION_SCHEMA.COLUMNS;`. The query results are shown in a table with columns: Row, table_schema, table_name, column_name, ordinal_position, is_nullable, and data_type. The results list 16 columns from the SCALER_SQL_PROJECT_1 database.

Row	table_schema	table_name	column_name	ordinal_position	is_nullable	data_type
1	SCALER_SQL_PROJECT_1	order_items	order_id	1	YES	STRING
2	SCALER_SQL_PROJECT_1	order_items	order_item_id	2	YES	INT64
3	SCALER_SQL_PROJECT_1	order_items	product_id	3	YES	STRING
4	SCALER_SQL_PROJECT_1	order_items	seller_id	4	YES	STRING
5	SCALER_SQL_PROJECT_1	order_items	shipping_limit_date	5	YES	TIMESTAMP
6	SCALER_SQL_PROJECT_1	order_items	price	6	YES	FLOAT64
7	SCALER_SQL_PROJECT_1	order_items	freight_value	7	YES	FLOAT64
8	SCALER_SQL_PROJECT_1	sellers	seller_id	1	YES	STRING
9	SCALER_SQL_PROJECT_1	sellers	seller_zip_code_prefix	2	YES	INT64
10	SCALER_SQL_PROJECT_1	sellers	seller_city	3	YES	STRING
11	SCALER_SQL_PROJECT_1	sellers	seller_state	4	YES	STRING
12	SCALER_SQL_PROJECT_1	geolocation	geolocation_zip_code_prefix	1	YES	INT64
13	SCALER_SQL_PROJECT_1	geolocation	geolocation_lat	2	YES	FLOAT64
14	SCALER_SQL_PROJECT_1	geolocation	geolocation_lng	3	YES	FLOAT64
15	SCALER_SQL_PROJECT_1	geolocation	geolocation_city	4	YES	STRING
16	SCALER_SQL_PROJECT_1	geolocation	geolocation_state	5	YES	STRING

2. Time period for which the data is given

```
SELECT MIN(order_purchase_timestamp) AS Starting_time,  
       MAX(order_purchase_timestamp) AS Ending_time  
FROM `SCALER_SQL_PROJECT_1.orders`;
```

The screenshot shows the Google Cloud BigQuery interface. The query editor displays the query: `SELECT MIN(order_purchase_timestamp) AS Starting_time, MAX(order_purchase_timestamp) AS Ending_time FROM `SCALER_SQL_PROJECT_1.orders`;`. The query results are shown in a table with columns: Row, Starting_time, and Ending_time. The results show the minimum and maximum order purchase timestamps.

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

3. Cities and States of customers ordered during the given period

```
SELECT customer_city, customer_state FROM `SCALER_SQL_PROJECT_1.customers`  
GROUP BY customer_city, customer_state  
ORDER BY customer_city, customer_state;
```

The screenshot shows the Google Cloud BigQuery console. On the left is the Explorer pane with a project tree containing 'SCALER_SQL_PROJECT_1' and its tables: customers, geolocation, order_items, order_reviews, orders, payments, products, and sellers. The main editor shows a query in 'Untitled 5' with the following SQL:

```
1 SELECT customer_city, customer_state FROM `SCALER_SQL_PROJECT_1.customers`
2 GROUP BY customer_city, customer_state
3 ORDER BY customer_city, customer_state;
```

The query results are displayed in a table with 17 rows. The columns are 'customer_city' and 'customer_state'. The results show a list of cities and their corresponding states in Brazil.

Row	customer_city	customer_state
1	abadia dos dourados	MG
2	abadiania	GO
3	abaete	MG
4	abaetetuba	PA
5	abaiara	CE
6	abaira	BA
7	abare	BA
8	abatia	PR
9	abdon batista	SC
10	abelardo luz	SC
11	abrantes	BA
12	abre campo	MG
13	abreu e lima	PE
14	acaiaca	MG
15	acailandia	MA
16	acajutiba	BA
17	acaru	CE

Business Insights, Analysis and Recommendation:

- The given data contains data types: Integer, Float, String, Timestamp and Geolocation.
- All the columns are nullable which means they allow nulls.
- The data contains details of e-commerce business in Brazil which have order, customer, payments and sellers' information.
- The data have Orders between 4th September 2016 to 17th October 2018.
- The data have customers from 27 states and 4119 cities.

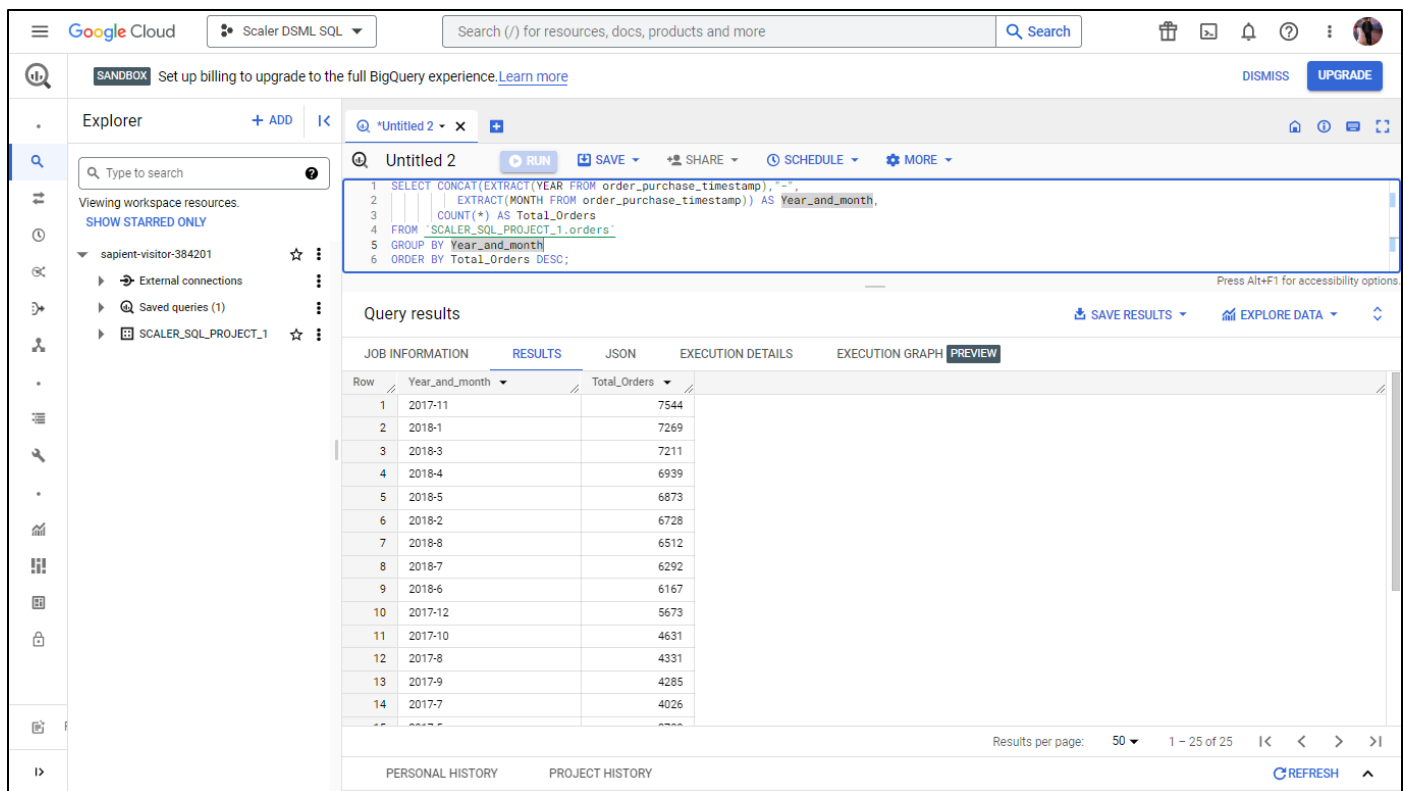
2. In-depth Exploration:

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?

```
SELECT CONCAT(EXTRACT(YEAR FROM order_purchase_timestamp), "-",
              EXTRACT(MONTH FROM order_purchase_timestamp)) AS Year_and_month,
COUNT(*) AS Total_Orders
FROM `SCALER_SQL_PROJECT_1.orders`
GROUP BY Year_and_month
ORDER BY Total_Orders DESC;
```

Business Insights, Analysis and Recommendation:

- ❖ As we can see there is a growing trend in Brazil as the count of Orders is increasing month by month.
- ❖ Maximum order was received in November 2017.
- ❖ As per the given data and extracted result we cannot say about seasonality peaks in specific month. However, there is November 2017 data where the maximum orders were received and 2018 data is not available for November. There could be a probability of receiving maximum no of orders in 2018 as well.



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

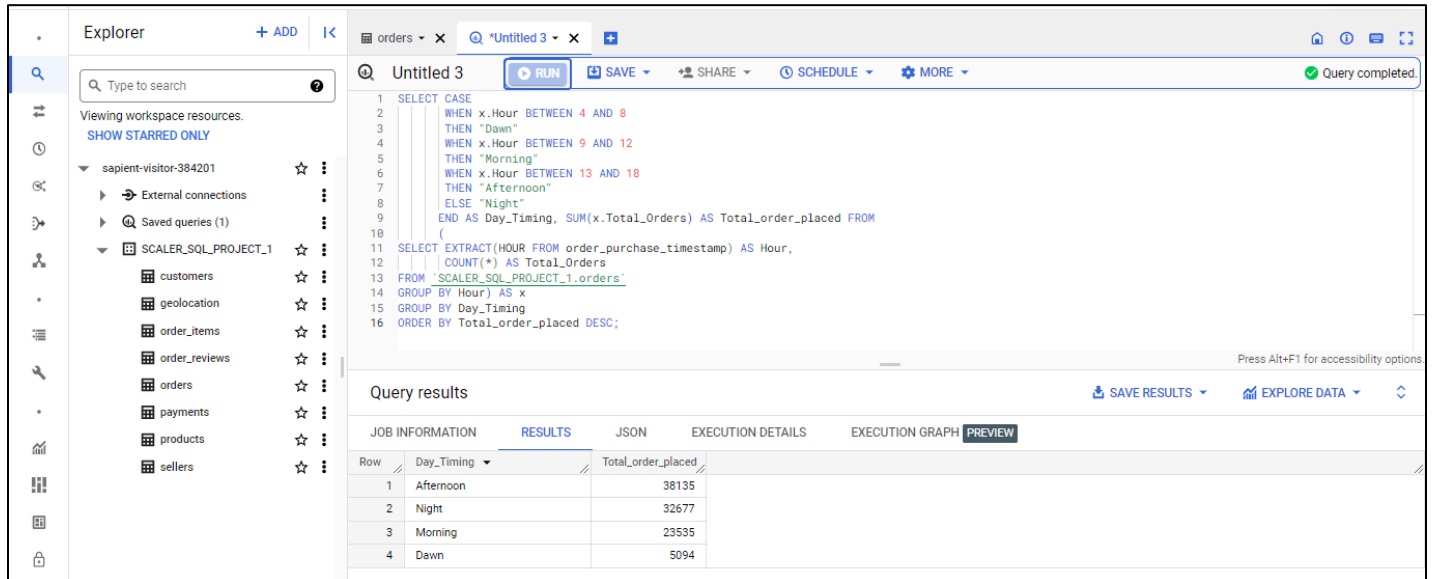
```

SELECT CASE
  WHEN x.Hour BETWEEN 4 AND 8
  THEN "Dawn"
  WHEN x.Hour BETWEEN 9 AND 12
  THEN "Morning"
  WHEN x.Hour BETWEEN 13 AND 18
  THEN "Afternoon"
  ELSE "Night"
END AS Day_Timing, SUM(x.Total_Orders) AS Total_order_placed FROM
(SELECT EXTRACT(HOUR FROM order_purchase_timestamp) AS Hour,
  COUNT(*) AS Total_Orders
FROM `SCALER_SQL_PROJECT_1.orders`
GROUP BY Hour) AS x
GROUP BY Day_Timing
ORDER BY Total_order_placed DESC;

```

Business Insights, Analysis and Recommendation:

- ❖ Day-Timing have 24 Hour Format which segregated in day timing as per below:
 - ✓ Dawn timing is from 4 o'clock to 8 o'clock.
 - ✓ Morning timing start from 9 o'clock and ends at 12 o'clock.
 - ✓ Afternoon timing starts from 13 o'clock and ends at 18 o'clock.
 - ✓ Night timing start from 19 o'clock and ends at 3 o'clock
- ❖ As we can see we have received maximum number of orders in Afternoon timing and need to focus on Night timings where we have received very less orders.



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

1. Get month on month orders by states

```

SELECT x.customer_state,x.Year_and_month, SUM(x.Total_orders) AS Total_order_placed FROM
(
SELECT  CONCAT(EXTRACT(YEAR FROM order_purchase_timestamp),"-",
              EXTRACT(MONTH FROM order_purchase_timestamp)) AS Year_and_month,
        COUNT(*) AS Total_Orders, c.customer_state
FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state, Year_and_month
) AS x
GROUP BY x.customer_state, x.Year_and_month
ORDER BY Total_order_placed DESC;
  
```


Business Insights, Analysis and Recommendation:

- SP state is dominating in terms of receiving maximum orders compare to order states.
- Maximum orders were placed in August 2018 in SP state.
- 41746 orders placed in SP state out of 99441 orders.
- SP state have placed 42% of total order placed.
- RJ state have placed 2nd highest orders followed by MG state which placed 11635 orders.
- RR, AP & AC received very few orders. 46, 68 & 81 is the count of order placed in these states respectively.
- As SP state have maximum numbers of customers 15540, which can be a reason to placed maximum number of orders as well, followed by RJ 6882 & MG 2773.
- As per extracted data we can see which state have maximum customers placed maximum orders, hence business should invest to build customers from different states which will help to have more orders from different states.

Google Cloud | Scaler DSML SQL | Search (/) for resources, docs, products and more

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

Explorer + ADD | customers x Q 'Untitled 3' x Q 'Untitled 4' x Q 'Untitled 5' x

Q 'Untitled 5' RUN SAVE SHARE SCHEDULE MORE

```

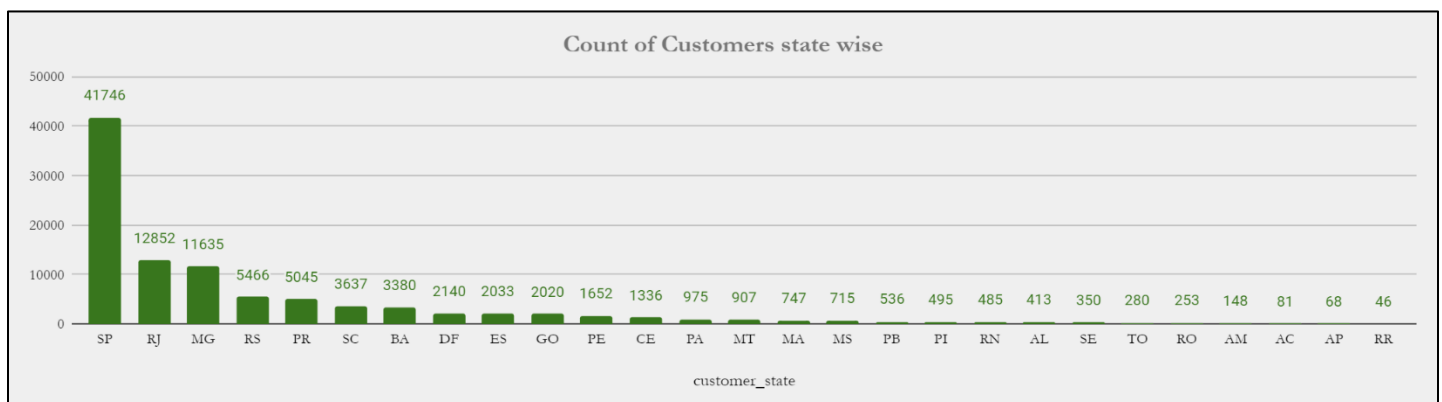
1 SELECT COUNT(*) AS customer_count,
2       customer_state,
3       customer_city
4 FROM `SCALER_SQL_PROJECT_1.customers`
5 GROUP BY customer_state, customer_city
6 ORDER BY customer_count DESC;

```

Query results

Row	customer_count	customer_state	customer_city
1	15540	SP	sao paulo
2	6882	RJ	rio de janeiro
3	2773	MG	belo horizonte
4	2131	DF	brasilia
5	1521	PR	curitiba
6	1444	SP	campinas
7	1379	RS	porto alegre
8	1245	BA	salvador
9	1189	SP	guarulhos
10	938	SP	sao bernardo do campo
11	849	RJ	niteroi
12	796	SP	santo andre
13	746	SP	osasco
14	713	SP	santos

Results per page: 50 | 1 - 50 of 4310 | REFRESH



4. **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) - You can use "payment_value" column in payments table**

WITH cte AS

```
(
SELECT SUM(p.payment_value) AS payment_of_2017,
      EXTRACT(MONTH FROM o.order_purchase_timestamp) AS Month,
      EXTRACT(YEAR FROM o.order_purchase_timestamp) AS Year
FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.payments` AS p
ON o.order_id = p.order_id
GROUP BY Year, Month
HAVING Month BETWEEN 1 AND 8 AND
      Year = 2017 ),
```

cte_1 AS

```
(SELECT SUM(p.payment_value) AS payment_of_2018,
      EXTRACT(MONTH FROM o.order_purchase_timestamp) AS Month,
      EXTRACT(YEAR FROM o.order_purchase_timestamp) AS Year
FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.payments` AS p
ON o.order_id = p.order_id
GROUP BY Year, Month
HAVING Month BETWEEN 1 AND 8 AND
      Year = 2018 )
```

```
SELECT ROUND((((SUM(cte_1.payment_of_2018) -
SUM(cte.payment_of_2017))*100)/SUM(cte_1.payment_of_2018)),2) AS
percentage_increase_in_cost_of_order_from_2017_to_2018
FROM cte JOIN cte_1 ON cte.Month = cte_1.Month
```

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer panel with a tree view of workspace resources. The main area displays a SQL query in the editor, and below it, the Query results are shown in a table format. The query calculates the percentage increase in the cost of orders from 2017 to 2018, focusing on months between January and August.

Query results

Row	percentage_increase
1	57.8

The interface also includes a top navigation bar with Google Cloud logo, a search bar, and various utility icons. The bottom of the screen shows tabs for Job Information, Results, JSON, Execution Details, Execution Graph, and Preview, with the Results tab currently active.

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

```
SELECT c.customer_state,
       ROUND(AVG(ot.price),2) AS Mean_price,
       ROUND(SUM(ot.price),2) AS Sum_price,
       ROUND(AVG(ot.freight_value),2) AS Mean_freight_value,
       ROUND(SUM(ot.freight_value),2) AS Sum_freight_value
FROM `SCALER_SQL_PROJECT_1.order_items` AS ot INNER JOIN `SCALER_SQL_PROJECT_1.orders` AS o
ON ot.order_id = o.order_id INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY c.customer_state;
```

Google Cloud | Scaler DSML SQL | Search (/) for resources, docs, products and more | Search

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

Explorer | + ADD | I<

Viewing workspace resources. SHOW STARRED ONLY

- sapient-visitor-384201
 - External connections
 - Saved queries (1)
 - SCALER_SQL_PROJECT_1
 - customers
 - geolocation
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellors

Untitled 7 | RUN | SAVE | SHARE | SCHEDULE | MORE

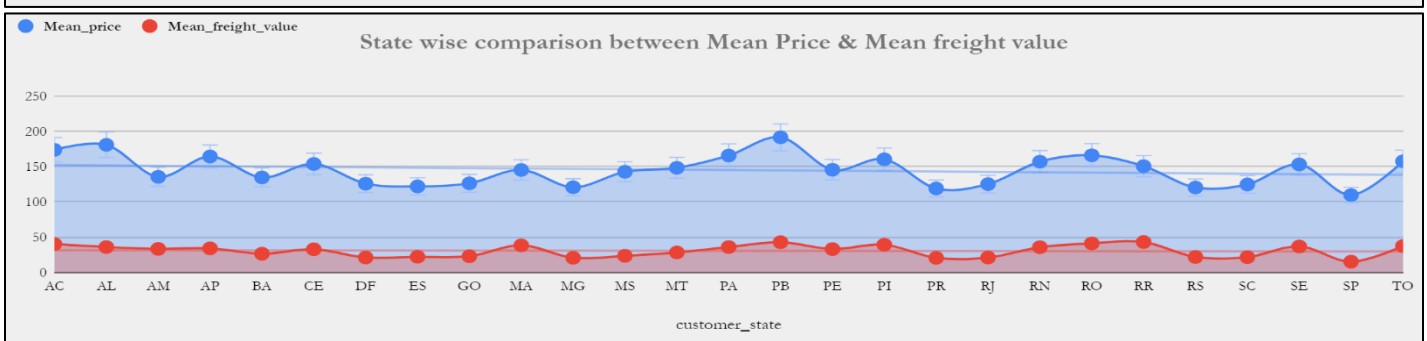
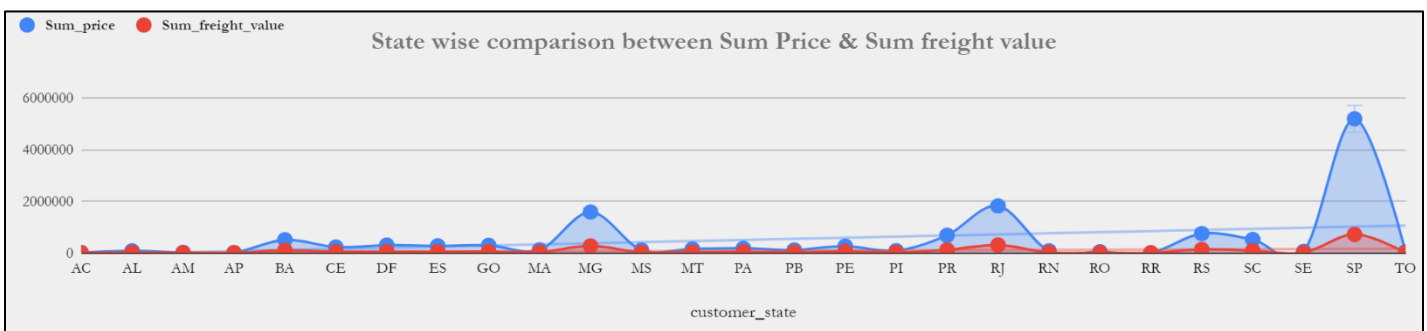
This query will process 15.42 MB when run.

```
1 SELECT c.customer_state,
2       ROUND(AVG(ot.price),2) AS Mean_price,
3       ROUND(SUM(ot.price),2) AS Sum_price,
4       ROUND(AVG(ot.freight_value),2) AS Mean_freight_value,
5       ROUND(SUM(ot.freight_value),2) AS Sum_freight_value
6 FROM `SCALER_SQL_PROJECT_1.order_items` AS ot INNER JOIN `SCALER_SQL_PROJECT_1.orders` AS o
7 ON ot.order_id = o.order_id INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
8 ON o.customer_id = c.customer_id
9 GROUP BY c.customer_state
10 ORDER BY c.customer_state;
```

Query results | SAVE RESULTS | EXPLORE DATA

Row	customer_state	Mean_price	Sum_price	Mean_freight_value	Sum_freight_value
1	AC	173.73	15982.95	40.07	3686.75
2	AL	180.89	80314.81	35.84	15914.59
3	AM	135.5	22356.84	33.21	5478.89
4	AP	164.32	13474.3	34.01	2788.5
5	BA	134.6	511349.99	26.36	100156.68
6	CE	153.76	227254.71	32.71	48351.59
7	DF	125.77	302603.94	21.04	50625.5
8	ES	121.91	275037.31	22.06	49764.6
9	GO	126.27	294591.95	22.77	53114.98
10	MA	145.2	119648.22	38.26	31523.77
11	MG	120.75	1585308.03	20.63	270853.46
12	MS	142.63	116812.64	23.37	19144.03

Results per page: 50 | 1 - 27 of 27 | < > | REFRESH



Business Insights, Analysis and Recommendation:

- There is growing trend in Brazil get 57.8% increase in cost of orders from 2017 to 2018 (include months between Jan to Aug only)
- SP state have dominance in terms of order received, customer count and now we can see there is highest revenue generated with SP state with high freight cost.
- Freight value is highest due to maximum order placed from the state to deliver the product to customer.
- Mean freight cost is high in RR, PB, RO, AC, PI in these places providing a free delivery above a certain order value could increase the orders from that region along with revenue

5. Analysis on sales, freight and delivery time

1. Calculate days between purchasing, delivering and estimated delivery

```
SELECT order_id, DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY)
      AS Purchase_delivered,
      DATE_DIFF(order_delivered_customer_date,order_estimated_delivery_date,DAY)
      AS Estimated_received,
      DATE_DIFF(order_estimated_delivery_date,order_purchase_timestamp,DAY)
      AS Estimated_purchase
FROM `SCALER_SQL_PROJECT_1.orders`
ORDER BY Purchase_delivered DESC, Estimated_received, Estimated_purchase;
```

The screenshot shows a data analytics tool interface. On the left is an 'Explorer' pane with a search bar and a list of workspace resources. The main area displays a SQL query in a text editor, titled 'Untitled 8'. Below the query editor, the 'Query results' section is visible, showing a table with 13 rows of data. The table has columns for 'order_id', 'Purchase_delivered', 'Estimated_received', and 'Estimated_purchase'. The results are sorted by 'Purchase_delivered' in descending order. At the bottom right, there are controls for 'Results per page' (set to 50) and a pagination indicator '1 - 50 of 99441'.

Row	order_id	Purchase_delivered	Estimated_received	Estimated_purchase
1	ca07593549f1816d26a572e06...	209	181	28
2	1b3190b2dfe9d789e1f14c05b...	208	188	19
3	440d0d17af552815d15a9e41a...	195	165	30
4	2fb597c2f772eca01b1f5c561b...	194	155	39
5	0f4519c5f1c541ddec9f21b3bd...	194	161	32
6	285ab9426d6982034523a855f...	194	166	28
7	47b40429ed8cce3aee9199792...	191	175	15
8	2fe324feb907e3ea3f2aa9650...	189	167	22
9	2d7561026d542c8dbd8f0deaa...	188	159	28
10	437222e3fd1b07396f1d9ba8c...	187	144	42
11	c27815f7e3dd0b926b5855262...	187	162	25
12	dfe5f68118c2576143240b8d7...	186	153	32
13	6e82dcfb5eada6283dba34f16...	182	155	27

2. Find time_to_delivery & diff_estimated_delivery. Formula for the same given below:
 - time_to_delivery = order_delivered_customer_date-order_purchase_timestamp
 - diff_estimated_delivery = order_estimated_delivery_date-order_delivered_customer_date

```
SELECT order_id, DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY)
    AS Time_to_delivery,
    DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date,DAY)
    AS Diff_estimated_delivery
FROM `SCALER_SQL_PROJECT_1.orders`
WHERE DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY) IS NOT NULL
    AND DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date,DAY) IS NOT NULL
ORDER BY Time_to_delivery, Diff_estimated_delivery;
```

The screenshot shows a data analytics tool interface. On the left is an 'Explorer' panel with a search bar and a tree view of workspace resources. The main area displays a SQL query in a text editor, with tabs for 'orders' and several 'Untitled' files. Below the editor, the 'Query results' section is active, showing a table with 13 rows of data. The table has columns for 'order_id', 'Time_to_delivery', and 'Diff_estimated_delivery'. The 'Time_to_delivery' column contains values from 0 to 27, and the 'Diff_estimated_delivery' column contains values from 7 to 27. At the bottom, there are tabs for 'PERSONAL HISTORY' and 'PROJECT HISTORY', and a 'REFRESH' button.

Row	order_id	Time_to_delivery	Diff_estimated_delivery
1	d5fbedc85190ba88580d6f82...	0	7
2	79e324907160caea526d8b94...	0	8
3	e65f1eeef1f52024ad1dcd034...	0	9
4	b70a8d75313560b4acf607739...	0	9
5	1d893dd7ca5f77ebf5f9fd20...	0	10
6	d3ca7b82c922817b06e5ca211...	0	11
7	f3c6775ba3d2d9fe2826f93b71...	0	11
8	21a8ffca665bc7a1087d31751...	0	11
9	f349c0b62f69c3fae5c4d7d3f3...	0	12
10	38c1e3d4ed6a13cd0cf612d4c...	0	16
11	434cecee7d1a65fc65358a632...	0	19
12	bb5a519e352b45b714192a02f...	0	25
13	8339b608be0d84fca9d8da68b...	0	27

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

```
SELECT customer_state, ROUND(AVG(freight_value),2) AS Mean_freight_value,
    ROUND(AVG(DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY)))
    AS Time_to_delivery,
    ROUND(AVG(DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date,DAY)))
    AS Diff_estimated_delivery
FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
ON o.customer_id = c.customer_id INNER JOIN `SCALER_SQL_PROJECT_1.order_items` AS oi
ON o.order_id = oi.order_id
GROUP BY customer_state
ORDER BY Time_to_delivery;
```

Explorer + ADD

Viewing workspace resources.
SHOW STARRED ONLY

sapient-visitor-384201

- External connections
- Saved queries (1)
- SCALER_SQL_PROJECT_1
 - customers
 - geolocation
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellers

Untitled 8

```

1 SELECT customer_state, ROUND(AVG(freight_value),2) AS Mean_freight_value,
2        ROUND(AVG(DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY)))
3        AS Time_to_delivery,
4        ROUND(AVG(DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date,DAY)))
5        AS Diff_estimated_delivery
6 FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
7 ON o.customer_id = c.customer_id INNER JOIN `SCALER_SQL_PROJECT_1.order_items` AS oi
8 ON o.order_id = oi.order_id
9 GROUP BY customer_state
10 ORDER BY Time_to_delivery;
```

Query results

JOB INFORMATION RESULTS JSON EXECUTION DETAILS EXECUTION GRAPH PREVIEW

Row	customer_state	Mean_freight_value	Time_to_delivery	Diff_estimated_delivery
1	SP	15.15	8.0	10.0
2	PR	20.53	11.0	13.0
3	MG	20.63	12.0	12.0
4	DF	21.04	13.0	11.0
5	RJ	20.96	15.0	11.0
6	RS	21.74	15.0	13.0
7	GO	22.77	15.0	11.0
8	ES	22.06	15.0	10.0
9	SC	21.47	15.0	11.0
10	MS	23.37	15.0	10.0
11	TO	37.25	17.0	11.0
12	MT	28.17	18.0	14.0

Results per page: 50 1 - 27 of 27

- Sort the data to get the following:
- Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5

Top 5 states which have lowest average freight value:

```

SELECT customer_state, ROUND(AVG(freight_value),2) AS Mean_freight_value
FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
ON o.customer_id = c.customer_id INNER JOIN `SCALER_SQL_PROJECT_1.order_items` AS oi
ON o.order_id = oi.order_id
GROUP BY customer_state
ORDER BY Mean_freight_value
LIMIT 5;
```

Explorer + ADD

Viewing workspace resources.
SHOW STARRED ONLY

sapient-visitor-384201

- External connections
- Saved queries (1)
- SCALER_SQL_PROJECT_1
 - customers
 - geolocation
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellers

Untitled 9

```

1 SELECT customer_state, ROUND(AVG(freight_value),2) AS Mean_freight_value
2 FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
3 ON o.customer_id = c.customer_id INNER JOIN `SCALER_SQL_PROJECT_1.order_items` AS oi
4 ON o.order_id = oi.order_id
5 GROUP BY customer_state
6 ORDER BY Mean_freight_value
7 LIMIT 5;
```

Query results

JOB INFORMATION RESULTS JSON EXECUTION DETAILS EXECUTION GRAPH PREVIEW

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

Top 5 states which have highest average freight value:

```
SELECT customer_state, ROUND(AVG(freight_value),2) AS Mean_freight_value
FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
ON o.customer_id = c.customer_id INNER JOIN `SCALER_SQL_PROJECT_1.order_items` AS oi
ON o.order_id = oi.order_id
GROUP BY customer_state
ORDER BY Mean_freight_value DESC
LIMIT 5;
```

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

```
1 SELECT customer_state, ROUND(AVG(freight_value),2) AS Mean_freight_value
2 FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
3 ON o.customer_id = c.customer_id INNER JOIN `SCALER_SQL_PROJECT_1.order_items` AS oi
4 ON o.order_id = oi.order_id
5 GROUP BY customer_state
6 ORDER BY Mean_freight_value DESC
7 LIMIT 5;
```

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

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

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

Top 5 states with lowest average time to delivery

```
SELECT customer_state,
ROUND(AVG(DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY)))
AS Time_to_delivery
FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY customer_state
ORDER BY Time_to_delivery
LIMIT 5;
```

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

```
1 SELECT customer_state,
2 ROUND(AVG(DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY)))
3 AS Time_to_delivery
4 FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
5 ON o.customer_id = c.customer_id
6 GROUP BY customer_state
7 ORDER BY Time_to_delivery
8 LIMIT 5;
```

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

Row	customer_state	Time_to_delivery
1	SP	8.0
2	MG	12.0
3	PR	12.0
4	DF	13.0
5	SC	14.0

Top 5 states with highest average time to delivery

```
SELECT customer_state,  
       ROUND(AVG(DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY)))  
       AS Time_to_delivery  
FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c  
ON o.customer_id = c.customer_id  
GROUP BY customer_state  
ORDER BY Time_to_delivery DESC  
LIMIT 5;
```

The screenshot shows the Google Cloud BigQuery interface. The query editor on the right contains the SQL query for finding the top 5 states with the highest average time to delivery. The query results are displayed in a table below the editor.

Row	customer_state	Time_to_delivery
1	RR	29.0
2	AP	27.0
3	AM	26.0
4	AL	24.0
5	PA	23.0

7. 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:

```
SELECT customer_state,  
       ROUND(AVG(DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date,DAY)))  
       AS Delivery_timings  
FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c  
ON o.customer_id = c.customer_id  
GROUP BY customer_state  
ORDER BY Delivery_timings  
LIMIT 5;
```

The screenshot shows the Google Cloud BigQuery interface. The query editor on the right contains the SQL query for finding the top 5 states where delivery is really fast compared to the estimated date. The query results are displayed in a table below the editor.

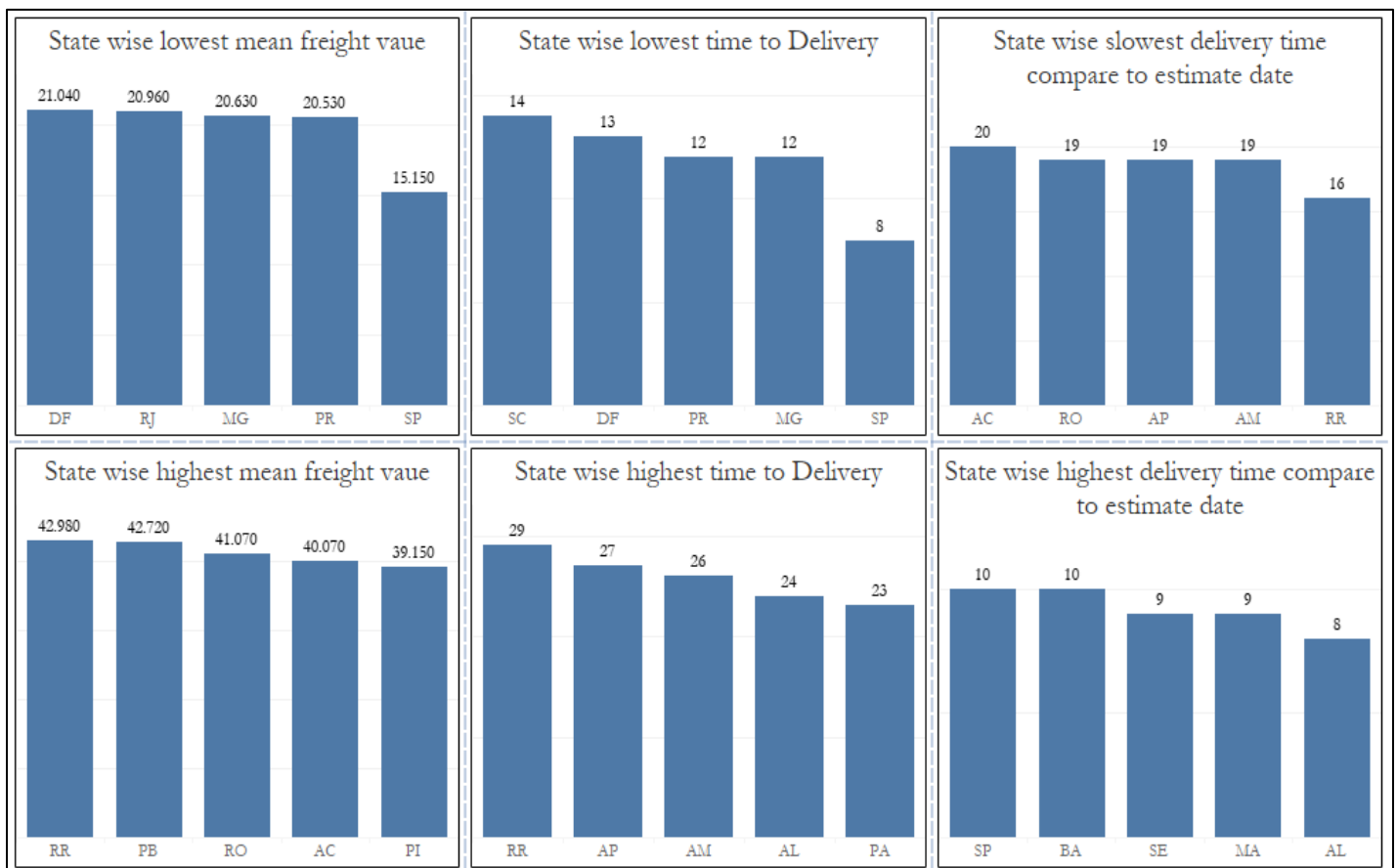
Row	customer_state	Delivery_timings
1	AL	8.0
2	MA	9.0
3	SE	9.0
4	SP	10.0
5	BA	10.0

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

```
SELECT customer_state,
       ROUND(AVG(DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY)))
       AS Delivery_timings
FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY customer_state
ORDER BY Delivery_timings DESC
LIMIT 5;
```

The screenshot shows a SQL IDE interface. On the left is an Explorer pane with a search bar and a list of workspace resources. The main editor displays the SQL query. Below the editor, the 'Query results' section is active, showing a table with 5 rows of data.

Row	customer_state	Delivery_timings
1	AC	20.0
2	RO	19.0
3	AM	19.0
4	AP	19.0
5	RR	16.0



Business Insights, Analysis and Recommendation:

- If the difference is in positive, then it shows the number of days order delayed by those days, if incase the difference is in Negative, then it shows the number of days order received early.
- States SP, PR, MG, DF have the lowest Freight cost and delivery time which due to warehouse and sellers might be in these regions.
- Increment in number of warehouses across the country might increase the number of orders and help to increase the revenue as well.
- Business should invest to have more sellers in across the country which can help to deliver the item earliest to customer which will somehow help to build customer relation and we can see increment in number of orders.

6. Payment type analysis:

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

```
SELECT CONCAT(EXTRACT(YEAR FROM order_purchase_timestamp), "-",  
              EXTRACT(MONTH FROM order_purchase_timestamp)) AS Year_and_month,  
        payment_type, COUNT(*) AS Count_of_orders  
FROM `SCALER_SQL_PROJECT_1.orders` AS o INNER JOIN `SCALER_SQL_PROJECT_1.payments` AS p  
ON o.order_id = p.order_id  
GROUP BY Year_and_month, payment_type  
ORDER BY Year_and_month;
```

The screenshot displays the Google Cloud BigQuery console. The left sidebar shows the Explorer view with a project named 'SCALER_SQL_PROJECT_1' containing tables like 'customers', 'geolocation', 'order_items', 'order_reviews', 'orders', 'payments', 'products', and 'sellers'. The main area shows a SQL query in the editor, which is the same query provided in the text block. Below the editor, the 'Query results' section is visible, showing a table with 14 rows of data. The table has four columns: 'Row', 'Year_and_month', 'payment_type', and 'Count_of_orders'. The data shows the count of orders for different payment types across various months and years.

Row	Year_and_month	payment_type	Count_of_orders
1	2016-10	credit_card	254
2	2016-10	voucher	23
3	2016-10	debit_card	2
4	2016-10	UPI	63
5	2016-12	credit_card	1
6	2016-9	credit_card	3
7	2017-1	voucher	61
8	2017-1	UPI	197
9	2017-1	credit_card	583
10	2017-1	debit_card	9
11	2017-10	voucher	291
12	2017-10	credit_card	3524
13	2017-10	UPI	993
14	2017-10	debit_card	52

Month over Month count of orders for different payment types					
F1	credit_card	debit_card	F2	UPI	voucher
2016-9	3				
2016-10	254	2		63	23
2016-12	1				
2017-1	583	9		197	61
2017-2	1,356	13		398	119
2017-3	2,016	31		590	200
2017-4	1,846	27		496	202
2017-5	2,853	30		772	289
2017-6	2,463	27		707	239
2017-7	3,086	22		845	364
2017-8	3,284	34		938	294
2017-9	3,283	43		903	287
2017-10	3,524	52		993	291
2017-11	5,897	70		1,509	387
2017-12	4,377	64		1,160	294
2018-1	5,520	109		1,518	416
2018-2	5,253	69		1,325	305
2018-3	5,691	78		1,352	391
2018-4	5,455	97		1,287	370
2018-5	5,497	51		1,263	324
2018-6	4,813	182		1,100	324
2018-7	4,755	242		1,229	281
2018-8	4,985	277		1,139	295
2018-9			2		15
2018-10			1		4

2. Count of orders based on the no. of payment installments

```
SELECT payment_installments,
       COUNT(*) As Count_of_orders
FROM `SCALER_SQL_PROJECT_1.payments`
GROUP BY payment_installments
ORDER BY payment_installments;
```

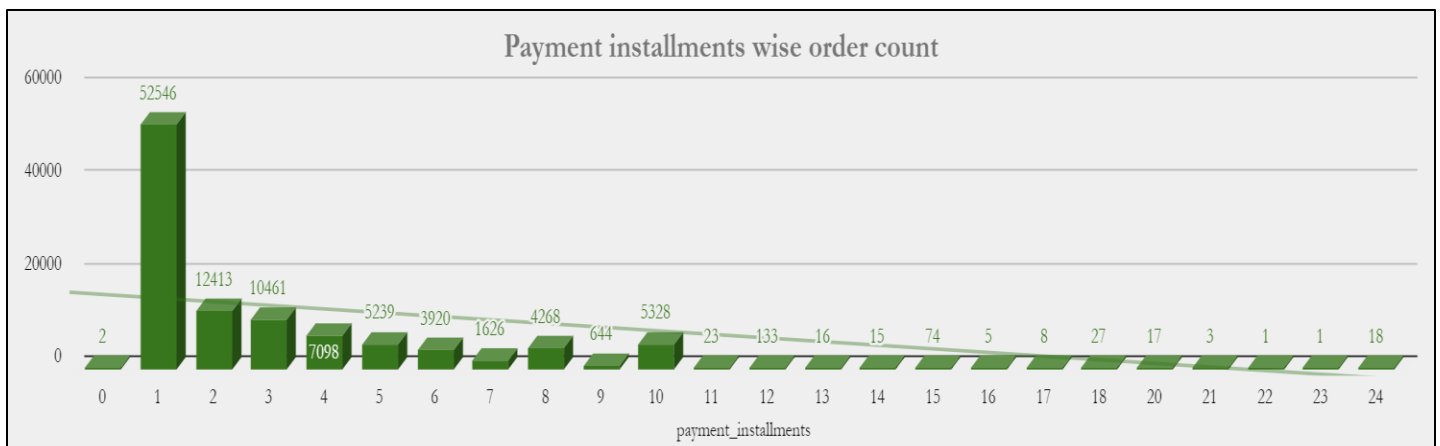
The screenshot displays the Google Cloud BigQuery interface. On the left, the Explorer pane shows the project hierarchy for 'SCALER_SQL_PROJECT_1', including tables like 'customers', 'geolocation', 'order_items', 'order_reviews', 'orders', 'payments', 'products', and 'sellers'. The main editor shows a SQL query titled 'Untitled 11' with the following code:

```
1 SELECT payment_installments,
2       COUNT(*) As Count_of_orders
3 FROM `SCALER_SQL_PROJECT_1.payments`
4 GROUP BY payment_installments
5 ORDER BY payment_installments;
```

The 'Query results' pane shows the output of the query. It includes a table with the following data:

Row	payment_installment	Count_of_orders
1	0	2
2	1	52546
3	2	12413
4	3	10461
5	4	7098
6	5	5239
7	6	3920
8	7	1626
9	8	4268
10	9	644
11	10	5328
12	11	23
13	12	133
14	13	16
15	14	15

At the bottom, the interface shows 'Results per page: 50' and '1 - 24 of 24' results, along with a 'REFRESH' button.



Business Insights, Analysis and Recommendation:

- Credit card is the most preferred payment type using payments to placed order by customer.
- Business can provide offers while using credit card by customer to attract them to increase value of orders for example we can provide cashback offer if customer is placing order for more than 2999. Which will attract customer to add more items and which will increase are sale.
- UPI method should also be in place while placing order as these days UPI is in trend for paying the amount direct from bank account.
- Most of the orders placed with full payment while placing orders, business should provide offers on EMI option so those customers would be attractive who cannot pay the complete amount at a time. So, they would also place order and net revenue of business would be increase.