CASE STUDY- TARGET SQL

- By Naman Agarwal

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 – orders

After analysis, the datatype of the columns in the table, orders are as follows-

For order_id, customer_id, order_status – the data type is VARCHAR string data type, means it can have variable length of characters.

For order_purchase_timestamp, order_approved_at, Level1order_delivered_carrier_date, order_delivered_customer_date, Level 1order_estimated_delivery_date - data type is timestamp, which means it has date and time in the format YYYY-MM-DD hh:mm:ss .

```
SELECT *
FROM `scaler-380812.target.orders`
LIMIT 10
```

Query results

SAVE RESULTS EXPLORE DATA

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	order_id	customer_id	order_stat	order_purchase_timesta	order_approved_at	order_delivered_carris	order_delivered	order_estimated_delivery_date
1	7a4df5d8cff4090...	725e9c75605414b21...	created	2017-11-25 11:10:3...	null	null	null	2017-12-12 00:00:00 UTC
2	35de4050331c6c...	4ee64f4bfc542546f4...	created	2017-12-05 01:07:5...	null	null	null	2018-01-08 00:00:00 UTC
3	b5359909123fa0...	438449d4af8980d10...	created	2017-12-05 01:07:5...	null	null	null	2018-01-11 00:00:00 UTC
4	dba5062fbda3af4...	964a6df3d9bdf60fe3...	created	2018-02-09 17:21:0...	null	null	null	2018-03-07 00:00:00 UTC
5	90ab3e7d52544e...	7d61b9f4f216052ba...	created	2017-11-06 13:12:3...	null	null	null	2017-12-01 00:00:00 UTC
6	fa65dad1b0e818...	9af2372a1e4934027...	shipped	2017-04-20 12:45:3...	2017-04-22 09:1...	2017-04-24 11:31:...	null	2017-05-18 00:00:00 UTC
7	1df2775799eecdf...	1240c2e65c4601dd8...	shipped	2017-07-13 11:03:0...	2017-07-13 11:1...	2017-07-18 18:17:...	null	2017-08-14 00:00:00 UTC
8	6190a94657e101...	5fc4c97dcb63903f99...	shipped	2017-07-11 13:36:3...	2017-07-11 13:4...	2017-07-13 17:55:...	null	2017-08-14 00:00:00 UTC
9	58ce513a55c740...	530d41b47b9dda9bc...	shipped	2017-07-29 18:05:0...	2017-07-29 18:1...	2017-07-31 16:41:...	null	2017-08-14 00:00:00 UTC
10	088683f795a3d3...	58d89fd1f863819ff9...	shipped	2017-07-13 10:02:4...	2017-07-14 02:2...	2017-07-20 20:02:...	null	2017-08-14 00:00:00 UTC

2. **Time period for which the data is given**

```
SELECT
  MIN(order_purchase_timestamp) AS starting_date_time,
  MAX(order_purchase_timestamp) AS ending_date_time
FROM `scaler-380812.target.orders`
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

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

By using min and max functions we can find the time period of the dataset given, it is giving first order date and last order date for which the dataset is given.

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

```
SELECT DISTINCT
  geolocation_state,
  geolocation_city
FROM `scaler-380812.target.geolocation`
ORDER BY geolocation_state ASC, geolocation_city ASC
```

Query results

SAVE RESULTS | EXPLO

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	geolocation_state	geolocation_city
1	AC	acrelandia
2	AC	acrelândia
3	AC	assis brasil
4	AC	brasileia
5	AC	brasiléia
6	AC	bujari
7	AC	campinas
8	AC	capixaba
9	AC	cruzeiro do sul
10	AC	epitaciolandia
11	AC	epitaciolândia
12	AC	feijo
13	AC	feijó

Results per page: 50 1 – 50 of 8463

By using distinct, we are finding unique city and state from where customers had ordered, so it is displaying output in such a way that for each state it is giving all the 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?**

For trend over years-

```
SELECT
x.year,
x.total_sales
FROM
(
SELECT
  EXTRACT(year FROM order_purchase_timestamp) AS year,
  ROUND(SUM(payment_value),3) AS total_sales
FROM `scaler-380812.target.orders` AS o
JOIN `scaler-380812.target.payments` AS p
ON o.order_id = p.order_id
GROUP BY EXTRACT(year FROM order_purchase_timestamp)
) AS x
ORDER BY x.year ASC
```

Query results

JOB INFORMATION | RESULTS | JSON | EXEC

Row	year	total_sales
1	2016	59362.34
2	2017	7249746.73
3	2018	8699763.05

We are using group by clause on the year, extracted from the purchase date and using aggregation on payment value to get the total sales of the year. From the above output it can be seen, there is an increasing trend in the sales, as sales increased from 2017 as compared to 2016 and sales also increased in 2018 as compared to 2017.

For seasonality on months across years-

```
SELECT
x.year,
x.month_number,
x.total_sales

FROM
(
SELECT
  EXTRACT(year FROM order_purchase_timestamp) AS year,
  EXTRACT(month FROM order_purchase_timestamp) AS month_number,
  ROUND(SUM(payment_value),3) AS total_sales
FROM `scaler-380812.target.orders` AS o
JOIN `scaler-380812.target.payments` AS p
ON o.order_id = p.order_id
GROUP BY EXTRACT(year FROM order_purchase_timestamp), EXTRACT(month
FROM order_purchase_timestamp)
) AS x
ORDER BY  x.month_number ASC , x.year ASC
```

Query results

SAVE RESULTS EXPL

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	year	month_number	total_sales
1	2017	1	138488.04
2	2018	1	1115004.18
3	2017	2	291908.01
4	2018	2	992463.34
5	2017	3	449863.6
6	2018	3	1159652.12
7	2017	4	417788.03
8	2018	4	1160785.48
9	2017	5	592918.82
10	2018	5	1153982.15
11	2017	6	511276.38
12	2018	6	1023880.5
13	2017	7	592382.92

Results per page: 50 1 – 25 of 25

As can be observed for the data, there is no specific seasonality in months over years where customer buying habits remains same. The sales does not have any specific trend in months over years.

So we can say customers buying habits are varying.

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

```
SELECT
  x.slots,
  COUNT(x.order_id) AS no_of_orders
FROM
(
SELECT
CASE
WHEN EXTRACT(hour from order_purchase_timestamp) BETWEEN 0 AND 6
THEN "DAWN(0-6)"
WHEN EXTRACT(hour from order_purchase_timestamp) BETWEEN 7 AND 12
THEN "Mor(7-12)"
WHEN EXTRACT(hour from order_purchase_timestamp) BETWEEN 13 AND 18
THEN "Eve(13-18)"
WHEN EXTRACT(hour from order_purchase_timestamp) BETWEEN 19 AND 23
THEN "Nig(19-23)"
END AS slots,
order_id,
customer_id
FROM `scaler-380812.target.orders`
) AS x
GROUP BY x.slots
ORDER BY no_of_orders
```

Query results

JOB INFORMATION | RESULTS | JSON | E

Row	slots	no_of_orders
1	DAWN(0-6)	5242
2	Mor(7-12)	27733
3	Nig(19-23)	28331
4	Eve(13-18)	38135

We are using bins to divide the time in 4 time slots and then using group by to group those slots as per count of orders.

As from the output, it can be observed, that in the evening (13 to 18 Hrs), most number of the orders were placed, hence we can say customers in Brazil prefer to buy in evening (13-18) Hrs.

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

1.Get month on month orders by states

```sql
SELECT
  g.geolocation_state,
  o.year,
  o.month_number,
  COUNT(o.order_id) AS no_of_orders
FROM
(
SELECT
  order_id,
  customer_id,
  EXTRACT(month FROM order_purchase_timestamp) AS month_number,
  EXTRACT(year FROM order_purchase_timestamp) AS year
FROM `scaler-380812.target.orders`
) AS o
JOIN `scaler-380812.target.customers` AS c
ON o.customer_id = c.customer_id
JOIN `scaler-380812.target.geolocation` AS g
ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
GROUP BY  g.geolocation_state , month_number , year
ORDER BY  g.geolocation_state ASC, year ASC, month_number ASC
```

Query results

SAVE RESULTS | EXPLORE DATA

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	geolocation_state	year	month_number	no_of_orders
1	AC	2017	1	45
2	AC	2017	2	179
3	AC	2017	3	329
4	AC	2017	4	362
5	AC	2017	5	886
6	AC	2017	6	432
7	AC	2017	7	605
8	AC	2017	8	657
9	AC	2017	9	161
10	AC	2017	10	535
11	AC	2017	11	368
12	AC	2017	12	389
13	AC	2018	1	649

Results per page: 50 | 1 – 50 of 565

Here, we grouped the data on basis of state, month and year and getting aggregation on count of orders placed in each month, every year. So, it can be observed from above data the number of orders placed in each state in every month and in each year.

2.Distribution of customers across the states in Brazil

```
SELECT
  g.geolocation_state,
  COUNT(DISTINCT c.customer_id) AS no_of_customers
FROM `scaler-380812.target.customers` AS c
JOIN `scaler-380812.target.geolocation` AS g
ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
GROUP BY  g.geolocation_state
ORDER BY  no_of_customers DESC
```

Query results

SAVE RESULTS | EXPLORE DATA

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	geolocation_state	no_of_customers
1	SP	41731
2	RJ	12839
3	MG	11624
4	RS	5473
5	PR	5034
6	SC	3651
7	BA	3371
8	ES	2027
9	GO	2011
10	DF	1974
11	PE	1648
12	CE	1332
13	PA	972

Results per page: 50 1 – 27 of 27

We are grouping states and getting aggregation on count of distinct customer IDs to get count of unique customers in each state. From the above, it can be observed that state SP has highest number of customers

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

```
SELECT
  g.geolocation_state,
  ROUND(SUM(oi.price),2) AS price_of_the_product,
  ROUND(SUM(p.payment_value),2) AS amount_paid_by_customer,
  ROUND(SUM(oi.freight_value),2) AS freight_value
FROM `scaler-380812.target.order_items` AS oi
JOIN  `scaler-380812.target.sellers` AS s
ON oi.seller_id = s.seller_id
JOIN `scaler-380812.target.geolocation` AS g
ON s.seller_zip_code_prefix = g.geolocation_zip_code_prefix
JOIN `scaler-380812.target.orders` AS o
ON o.order_id = oi.order_id
JOIN `scaler-380812.target.payments` AS p
ON p.order_id = o.order_id
GROUP BY g.geolocation_state
ORDER BY freight_value DESC, price_of_the_product DESC , amount_paid
_by_customer DESC
```

Query results

SAVE RESULTS EXPLORE

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	geolocation_state	price_of_the_product	amount_paid_by_customer	freight_value
1	SP	1192360215.42	1755181175.22	208552400....
2	MG	264465221.83	407498692.42	49188130.55
3	PR	169123423.96	232879689.08	27112053.14
4	RJ	163082254.65	200602052.29	19049333.42
5	SC	113499125.98	145113832.91	17773446.04
6	RS	58042693.59	81319545.55	9663408.98
7	BA	26533684.47	30858752.45	2117672.61
8	DF	4842728.72	7209242.0	1270439.18
9	MA	3629727.09	5146318.98	1210855.14
10	ES	3336166.44	4980036.81	750690.36
11	GO	4615429.14	8204875.86	739732.04
12	PE	2841362.83	3297543.82	275892.88
13	MT	976186.8	1346333.75	268753.7

Results per page: 50 1 – 22 of 22

So, from the above, it can be seen the monetary value which comprises of the price of the product, amount paid by the customer and the freight value generated in each state is depicted.

Further-

```
SELECT
x.geolocation_state,
ROUND((x.price_of_the_product + x.amount_paid_by_customer + x.freigh
t_value),2) AS total_money_transaction
FROM
(
SELECT
  g.geolocation_state,
  ROUND(SUM(oi.price),2) AS price_of_the_product,
  ROUND(SUM(p.payment_value),2) AS amount_paid_by_customer,
  ROUND(SUM(oi.freight_value),2) AS freight_value
FROM `scaler-380812.target.order_items` AS oi
JOIN  `scaler-380812.target.sellers` AS s
ON oi.seller_id = s.seller_id
JOIN `scaler-380812.target.geolocation` AS g
ON s.seller_zip_code_prefix = g.geolocation_zip_code_prefix
JOIN `scaler-380812.target.orders` AS o
ON o.order_id = oi.order_id
JOIN `scaler-380812.target.payments` AS p
ON p.order_id = o.order_id
GROUP BY g.geolocation_state
ORDER BY freight_value DESC, price_of_the_product DESC , amount_paid
_by_customer DESC
) AS x
ORDER BY total_money_transaction DESC
```

Query results

SAVE RESULTS EXPLO

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	geolocation_state	total_money_transaction
1	SP	3156093790.81
2	MG	721152044.8
3	PR	429115166.18
4	RJ	382733640.36
5	SC	276386404.93
6	RS	149025648.12
7	BA	59510109.53
8	GO	13560037.04
9	DF	13322409.9
10	MA	9986901.21
11	ES	9066893.61
12	PE	6414799.53
13	MT	2591274.25

Results per page: 50 1 – 22 of 22

It can be observed,

State SP has the highest monetary transaction or highest monetary generation among all the states.

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 T1 AS
(
SELECT
EXTRACT(year FROM order_purchase_timestamp) AS year,
EXTRACT(month FROM order_purchase_timestamp) AS month,
order_id
FROM `scaler-380812.target.orders`
WHERE EXTRACT(year FROM order_purchase_timestamp) IN (2017,2018) AND
 EXTRACT(month FROM order_purchase_timestamp) IN(1,2,3,4,5,6,7,8)
)

SELECT
y.year,
y.total_sales,
y.prev_year_sales,
ROUND(((y.total_sales - y.prev_year_sales)/y.prev_year_sales)*100,2)
 AS percentage_increase_in_sales
FROM
(
SELECT
x.year,
x.total_sales,
LAG(x.total_sales,1) OVER(ORDER BY x.year ASC) AS prev_year_sales
FROM
(
SELECT
T1.year,
ROUND(SUM(p.payment_value),2) AS total_sales
FROM T1
JOIN `scaler-380812.target.payments` AS p
ON T1.order_id = p.order_id
GROUP BY T1.year
) AS x
) AS y
ORDER BY y.year ASC
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	year	total_sales	prev_year_sales	percentage_incr
1	2017	3669022.12	null	null
2	2018	8694733.84	3669022.12	136.98

AS from above data, it can be observed that there is growth in total sales in year 2018 as compared to year 2017 for the months from January to August. Now further to we had calculated the percentage increase in sales which is around 136.98 %.

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

```
SELECT
  g.geolocation_state,
  ROUND(SUM(oi.price),2) AS sum_of_price,
  ROUND(AVG(oi.price),2) AS mean_of_price,
  ROUND(SUM(oi.freight_value),2) AS sum_of_freight_value,
  ROUND(AVG(oi.freight_value),2) AS mean_of_freight_value
FROM `scaler-380812.target.geolocation` AS g
JOIN `scaler-380812.target.sellers` AS s
ON g.geolocation_zip_code_prefix = s.seller_zip_code_prefix
JOIN `scaler-380812.target.order_items` AS oi
ON s.seller_id = oi.seller_id
GROUP BY g.geolocation_state
ORDER BY g.geolocation_state ASC
```

Query results

SAVE RESULTS EXPLOR

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	geolocation_state	sum_of_price	mean_of_price	sum_of_freight_value	mean_of_freight_va
1	AC	43788.0	267.0	5385.76	32.84
2	AM	31779.0	392.33	2208.6	27.27
3	BA	23385841.45	351.61	1939324.41	29.16
4	CE	740073.63	246.12	163715.97	54.44
5	DF	4674257.66	72.56	1223546.71	18.99
6	ES	3211486.26	127.31	724107.32	28.7
7	GO	4444926.59	164.13	694619.52	25.65
8	MA	3604486.05	89.9	1201987.71	29.98
9	MG	253958733.43	122.97	47130618.12	22.82
10	MS	724755.24	165.43	113813.87	25.98
11	MT	961121.8	116.85	263738.15	32.07
12	PB	852601.9	352.31	83959.72	34.69
13	PE	2593455.74	277.52	265633.12	28.43

Results per page: 50 1 – 22 of 22

We are grouping the states and using aggregation function to calculate sum and mean of the price and freight value.

The above output is displaying the sum of price and average of the price for each state. It is also showing the total freight value and avg freight value for each state.

5.Analysis on sales, freight and delivery time

1. **Calculate days between purchasing, delivering and estimated delivery**

```sql
SELECT
x.order_id,
x.order_status,
DATE_DIFF(x.delivered_date, x.purchase_date , day) AS day_diff_purch
ase_delivered,
DATE_DIFF(x.estimated_delivery_date, x.purchase_date, day) AS day_di
ff_purchase_est_delivery
FROM
(
SELECT
order_id,
order_status,
EXTRACT(date FROM order_purchase_timestamp) AS purchase_date,
EXTRACT(date FROM order_delivered_customer_date) AS delivered_date,
EXTRACT(date FROM order_estimated_delivery_date) AS estimated_delive
ry_date
FROM `scaler-380812.target.orders`
) AS x
WHERE x.order_status = "delivered"
```

Query results

SAVE RESULTS EXPLORE DAT

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	order_id	order_status	day_diff_purchase_delivered	day_diff_purchase_est_delivery
1	635c894d068ac37e6e03dc54eccb6189	delivered	31	33
2	3b97562c3aee8bdedcb5c2e45a50d5e1	delivered	33	34
3	68f47f50f04c4cb6774570cfde3a9aa7	delivered	30	32
4	276e9ec344d3bf029ff83a161c6b3ce9	delivered	44	40
5	54e1a3c2b97fb0809da548a59f64c813	delivered	41	37
6	fd04fa4105ee8045f6a0139ca5b49f27	delivered	37	36
7	302bb8109d097a9fc6e9cefc5917d1f3	delivered	34	29
8	66057d37308e787052a32828cd007e58	delivered	39	33
9	19135c945c554eebfd7576c733d5ebdd	delivered	36	34
10	4493e45e7ca1084efcd38ddebf174dda	delivered	34	34
11	70c77e51e0f179d75a64a614135afb6a	delivered	43	32
12	d7918e406132d7c81f1b845276b03a3b	delivered	35	32
13	43f6604e77ce6433e7d68dd86db73b45	delivered	33	26

Results per page: 50 1 – 50 of 96478

First, we are extracting date part from all the three date columns, then using datediff function, and calculating date difference in form of days between the dates. We are filtering the data to get order_id for the delivered order only.

The above output displays the difference between delivered and purchase date and estimated delivery date and purchase date for each order.

2.Find time_to_delivery & diff_estimated_delivery. Formula for the same given below:

time_to_delivery = order_purchase_timestamp-order_delivered_customer_date

diff_estimated_delivery = order_estimated_delivery_date-order_delivered_customer_date

```
SELECT
x.order_id,
x.order_status,
DATE_DIFF(delivered_date, purchase_date , day) AS time_to_deliver
y ,
DATE_DIFF(delivered_date, estimated_delivery_date, day) AS diff_e
stimated_delivery

FROM
(
SELECT
order_id,
order_status,
EXTRACT(date FROM order_purchase_timestamp) AS purchase_date,
EXTRACT(date FROM order_delivered_customer_date) AS delivered_dat
e,
EXTRACT(date FROM order_estimated_delivery_date) AS estimated_del
ivery_date
FROM `scaler-380812.target.orders`
)AS x
WHERE x.order_status = "delivered"
```

Query results

SAVE RESULTS | EXPLO

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	order_id	order_status	time_to_delivery	diff_estimated_delivery
1	635c894d068ac37e6e03dc54eccb6189	delivered	31	-2
2	3b97562c3aee8bdedcb5c2e45a50d5e1	delivered	33	-1
3	68f47f50f04c4cb6774570cfde3a9aa7	delivered	30	-2
4	276e9ec344d3bf029ff83a161c6b3ce9	delivered	44	4
5	54e1a3c2b97fb0809da548a59f64c813	delivered	41	4
6	fd04fa4105ee8045f6a0139ca5b49f27	delivered	37	1
7	302bb8109d097a9fc6e9cefc5917d1f3	delivered	34	5
8	66057d37308e787052a32828cd007e58	delivered	39	6
9	19135c945c554eebfd7576c733d5ebdd	delivered	36	2
10	4493e45e7ca1084efcd38ddebf174dda	delivered	34	0
11	70c77e51e0f179d75a64a614135afb6a	delivered	43	11
12	d7918e406132d7c81f1b845276b03a3b	delivered	35	3
13	43f6604e77ce6433e7d68dd86db73b45	delivered	33	7

Results per page: 50 1 – 50 of 96478

We are calculating time difference between purchase date and delivered date which is represented by time_to_delivery.
Then, we are also calculating diff_estimated_delivery, which is difference between estimated delivery date and delivered date.

negative diff_estimated_delivery means that the order is delivered before the estimated delivery date, while positive diff_estimated_delivery means order is delivered even after the estimated delivery date.

Further we are filtering the orders only for delivered orders, since we are interested in delivered orders.

The above output displays the time to deliver which is difference between delivered date and purchase date and estimated delivery which is the difference between delivered date and estimated delivery date for each order.

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

```
WITH T1 AS
(
SELECT
order_id,
order_status,
DATE_DIFF(delivered_date, purchase_date , day) AS time_to_delivery ,
DATE_DIFF(delivered_date, estimated_delivery_date, day) AS diff_esti
mated_delivery
FROM
(
SELECT
order_id,
order_status,
EXTRACT(date FROM order_purchase_timestamp) AS purchase_date,
EXTRACT(date FROM order_delivered_customer_date) AS delivered_date,
EXTRACT(date FROM order_estimated_delivery_date) AS estimated_delive
ry_date
FROM `scaler-380812.target.orders`
)
)
SELECT
g.geolocation_state,
ROUND(AVG(oi.freight_value),2) AS mean_freight_value,
ROUND(AVG(T1.time_to_delivery),2) AS mean_time_to_delivery,
ROUND(AVG(T1.diff_estimated_delivery),2) AS mean_diff_estimated_deli
very
FROM `scaler-380812.target.geolocation` AS g
JOIN `scaler-380812.target.sellers` AS s
ON g.geolocation_zip_code_prefix = s.seller_zip_code_prefix
JOIN `scaler-380812.target.order_items` AS oi
ON oi.seller_id = s.seller_id
JOIN T1
ON T1.order_id = oi.order_id
WHERE T1.order_status = "delivered"
GROUP BY g.geolocation_state
ORDER BY g.geolocation_state
```

Query results

SAVE RESULTS EXPLO

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	geolocation_state	mean_freight_value	mean_time_to_delivery	mean_diff_estimated_delivery
1	AM	27.27	48.0	9.0
2	BA	29.37	14.06	-15.13
3	CE	55.75	16.66	-13.83
4	DF	18.94	11.98	-13.2
5	ES	29.08	12.61	-12.77
6	GO	25.73	12.73	-14.93
7	MA	30.03	17.65	-11.26
8	MG	22.76	12.57	-13.26
9	MS	25.98	14.04	-15.19
10	MT	32.19	14.51	-15.77
11	PB	34.58	12.16	-21.06
12	PE	28.47	13.1	-16.06
13	PI	37.14	13.73	-15.0

Results per page: 50 1 – 21 of 21

We are first using CTE to get the time_to_delivery and diff_estimated_delivery Then using joins to join all the tables to get the required output.

Finally we are using group by on states and using aggregation function AVG() to calculate mean of freight_value, time_to_delivery, diff_estimated_delivery.

So for each state the average freight value, average time to deliver and average estimated delivery time is displayed.

Further we are filtering the data only for delivered orders since only for those orders above parameters are applicable.

Negative average estimated delivery means the order is delivered before the estimated delivery date and positive means order is delivered even after the estimated delivery date.

The above output displays the average freight value, average time to deliver and average estimated time to deliver for each state.

4. Sort the data to get the following:

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

```
SELECT
g.geolocation_state,
ROUND(AVG(freight_value),2) AS avg_freight_value
FROM `scaler-380812.target.geolocation` AS g
JOIN `scaler-380812.target.sellers` AS s
ON g.geolocation_zip_code_prefix = s.seller_zip_code_prefix
JOIN `scaler-380812.target.order_items` AS oi
ON oi.seller_id = s.seller_id
GROUP BY g.geolocation_state
ORDER BY avg_freight_value ASC
LIMIT 5
```

Query results

JOB INFORMATION | RESULTS | JSON

Row	geolocation_state	avg_freight_value
1	RN	15.93
2	SP	18.44
3	RJ	18.93
4	DF	18.99
5	PR	22.11

We are first joining tables then using group by to group the data on basis of state and using aggregating function AVG to get average freight value.
From the above data, it can be seen the state RN has the lowest average freight value, means in state RN customers had to pay least amount to get the order delivered at their location.
Further, it also shows the top 5 states where average freight value is least.

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

```
WITH T1 AS
(
SELECT
Order_id,
order_status,
DATE_DIFF(delivered_date, purchase_date , day) AS time_to_deliver
y
FROM
(
SELECT
EXTRACT(date FROM order_purchase_timestamp) AS purchase_date,
EXTRACT(date FROM order_delivered_customer_date) AS delivered_dat
e,
order_id,
order_status
FROM `scaler-380812.target.orders`
)
)

SELECT
g.geolocation_state,
ROUND(AVG(T1.time_to_delivery),2) AS avg_time_to_deliver
FROM `scaler-380812.target.geolocation` AS g
JOIN `scaler-380812.target.sellers` AS s
ON g.geolocation_zip_code_prefix = s.seller_zip_code_prefix
JOIN `scaler-380812.target.order_items` AS oi
ON oi.seller_id = s.seller_id
JOIN T1
ON T1.order_id = oi.order_id
WHERE T1.order_status = "delivered"
GROUP BY g.geolocation_state
ORDER BY avg_time_to_deliver ASC
LIMIT 5
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION

Row	geolocation_state	avg_time_to_deliver
1	RN	7.49
2	RS	11.11
3	RJ	11.62
4	DF	11.98
5	PB	12.16

First, we are using CTE to get time_to_deliver, then using joins to join the tables.
Then we are grouping the data by state and using AVG function to get average time to deliver the order.
From the above data, it can be seen that state RN has least average time to deliver the order. It also shows the top 5 states where the average time to deliver the order is least.

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

```
WITH T1 AS
(
SELECT
Order_id,
DATE_DIFF(delivered_date, purchase_date , day) AS time_to_deliver,
DATE_DIFF(est_delivery_date , purchase_date, day) AS est_time_to_del
iver
FROM
(
SELECT
EXTRACT(date FROM order_purchase_timestamp) AS purchase_date,
EXTRACT(date FROM order_delivered_customer_date) AS delivered_date,
EXTRACT(date FROM order_estimated_delivery_date) AS est_delivery_dat
e,
order_id
FROM `scaler-380812.target.orders`
)
)

SELECT
geolocation_state,
ROUND(avg_time_to_deliver,2) AS avg_time_to_deliver,
ROUND(avg_est_time_to_deliver,2) AS avg_est_time_to_deliver
FROM
(
SELECT
g.geolocation_state,
AVG(time_to_deliver) AS avg_time_to_deliver,
AVG(est_time_to_deliver) AS avg_est_time_to_deliver
FROM `scaler-380812.target.geolocation` AS g
JOIN `scaler-380812.target.sellers` AS s
ON g.geolocation_zip_code_prefix = s.seller_zip_code_prefix
JOIN `scaler-380812.target.order_items` AS oi
ON oi.seller_id = s.seller_id
JOIN T1
ON T1.order_id = oi.order_id
GROUP BY g.geolocation_state
)
WHERE NOT avg_time_to_deliver IS NULL
ORDER BY avg_time_to_deliver ASC
LIMIT 5
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | E

Row	geolocation_state	avg_time_to_deliver	avg_est_time_to_deliver
1	RN	7.49	24.2
2	RS	11.11	28.0
3	RJ	11.62	24.2
4	DF	11.98	25.31
5	PB	12.16	33.31

We are using CTE first to extract date part from the columns and then calculating date difference so as to calculate the average time to deliver and average estimated time to deliver.

From the above result, this can be verified that average time to deliver in the mentioned states is much less than the estimated time to deliver. From all the states, state RN has least average time to deliver.

Above result shows the top 5 states where time to deliver the product is least.

6. Payment type analysis:

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

```sql
SELECT
payment_type,
year,
month_number,
COUNT(o.order_id) AS no_of_orders
FROM
  (
  SELECT
  EXTRACT(month FROM order_purchase_timestamp) AS month_number
,
  EXTRACT(year FROM order_purchase_timestamp) AS year,
  order_id
  FROM `scaler-380812.target.orders`
  ) AS o
JOIN `scaler-380812.target.payments` AS p
ON o.order_id = p.order_id
GROUP BY payment_type, month_number, year
ORDER BY year ASC,month_number ASC
```

Query results

SAVE RESULTS EXPLOR

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	payment_type	year	month_number	no_of_orders
1	credit_card	2016	9	3
2	credit_card	2016	10	254
3	voucher	2016	10	23
4	debit_card	2016	10	2
5	UPI	2016	10	63
6	credit_card	2016	12	1
7	voucher	2017	1	61
8	UPI	2017	1	197
9	credit_card	2017	1	583
10	debit_card	2017	1	9
11	credit_card	2017	2	1356
12	voucher	2017	2	119
13	UPI	2017	2	398

Results per page: 50 1 – 50 of 90

Here, the data displays the various mode of payment used by customers to buy the products. It also shows the number of orders placed in each month and in each year using different types of the payment methods.

Further,

```sql
SELECT
payment_type,
COUNT(o.order_id) AS no_of_orders
FROM
  (
  SELECT
  EXTRACT(month FROM order_purchase_timestamp) AS month_number
,
  EXTRACT(year FROM order_purchase_timestamp) AS year,
  order_id
  FROM `scaler-380812.target.orders`
  ) AS o
JOIN `scaler-380812.target.payments` AS p
ON o.order_id = p.order_id
GROUP BY payment_type
ORDER BY no_of_orders DESC
```

Query results

JOB INFORMATION | RESULTS | JSON

Row	payment_type	no_of_orders
1	credit_card	76795
2	UPI	19784
3	voucher	5775
4	debit_card	1529
5	not_defined	3

This output shows that most used payment mode is credit card by the customers to place an order. So, in additional we can provide some discount on the other payment modes so as to promote them if required.

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

```
SELECT
x.payment_installments,
x.count_of_orders

FROM
(
SELECT
p.payment_installments,
COUNT(o.order_id) AS count_of_orders
FROM `scaler-380812.target.orders` AS o
JOIN `scaler-380812.target.payments` p
ON o.order_id = p.order_id
GROUP BY p.payment_installments
) AS x
ORDER BY x.payment_installments ASC
```

Query results

SAVE RESULTS EXPLOR

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	payment_installments	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

Results per page: 50 1 – 24 of 24

This output shows the number of installments opted by the customers to pay for the product. And also showing the numbers of orders placed by customers using different number of installments.

Further-

```sql
SELECT
x.payment_installments,
x.count_of_orders
FROM
(
SELECT
p.payment_installments,
COUNT(o.order_id) AS count_of_orders
FROM `scaler-380812.target.orders` AS o
JOIN `scaler-380812.target.payments` p
ON o.order_id = p.order_id
GROUP BY p.payment_installments
) AS x
ORDER BY x.count_of_orders DESC
```

Query results

SAVE RESULTS EXPLOR

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	payment_installments	count_of_orders
1	1	52546
2	2	12413
3	3	10461
4	4	7098
5	10	5328
6	5	5239
7	8	4268
8	6	3920
9	7	1626
10	9	644
11	12	133
12	15	74
13	18	27

Results per page: 50 1 – 24 of 24

This output shows the maximum number of orders are places using only 1 payment installment, which is good for business, in additional it can be seen even more than 10 payment installments are used by the customers to place an order, which means company has to wait for a long time to receive full payment.

We can sort this by providing some additional discounts or reducing the additional fee on the installments in order to promote less installments method.