

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

### 1.1 . Data type of columns in a table

We can check the datatypes by clicking on the respective table and go to schema, under **TYPE** column. As no particular table has been mentioned. For all the given table Data only the following Datatypes of the columns were included:

String, Integer, Float, Timestamp

Schema Snapshot for Customers Table (below).

Field name	Type	Mode	Key	Collation	Default value	Policy tags	Description
<input type="checkbox"/> customer_id	STRING	NULLABLE					
<input type="checkbox"/> customer_unique_id	STRING	NULLABLE					
<input type="checkbox"/> customer_zip_code_prefix	INTEGER	NULLABLE					
<input type="checkbox"/> customer_city	STRING	NULLABLE					
<input type="checkbox"/> customer_state	STRING	NULLABLE					

### 1.2 . Time period for which the data is given

As explained in the Context, the data contains information of 100k orders from 2016 to 2018. We can confirm this time period by sending the below query:

#### Query:

select

min (order\_purchase\_timestamp) as first\_order,

max (order\_purchase\_timestamp) as last\_order

from `Biz\_target.orders`

limit 5

The screenshot displays the Databricks SQL workspace. At the top, there's a toolbar with icons for home, close, download, save, share, schedule, and more settings. Below the toolbar, the query editor shows a SQL query:

```
select  
min (order_purchase_timestamp) as first_order,  
max (order_purchase_timestamp) as last_order  
from `Biz_target.orders`  
limit 5
```

Below the query editor, the "Query results" section is visible. It contains a tabbed interface with four tabs: "JOB INFORMATION", "RESULTS" (which is active), "JSON", "EXECUTION DETAILS", and "EXECUTION GRAPH". The "RESULTS" tab shows a table with two columns: "first\_order" and "last\_order". The table has one row of data.

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

### Insight:

The Orders table dataset contains the purchase timestamp of each order. We can find the time period by extracting the first order and last order from the orders table data by using aggregate functions MIN, MAX. MIN gives the first or least value and MAX gives the last or highest value from the dataset

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

Here we need to include the cities and states of customers for the above query which is:

**Query:**

```
select
c.customer_id,c.customer_city,c.customer_state,o.order_purchase_timestamp

from `Biz_target.customers` c left join `Biz_target.orders` o

ON c.customer_id = o.customer_id

order by o.order_purchase_timestamp
```

## Insight:

For this query we need **customers and orders** tables because the required data can be shown only by intercepting these two tables which in our case, we have used LEFT JOIN. The reason to use LEFT JOIN is because we only need to show the data from customers table only which holds the city and state columns as these are the required data.

From the above query you can see that we have joined these two tables with customer\_id as it is the common column from the two tables and selected the customer\_id, city, state and purchase timestamp which results with customers city and state at the particular time period.

## Snap of Query result:

Untitled					Pres
<pre>11 select c.customer_id,c.customer_city,c.customer_state,o.order_purchase_timestamp 12 from 'Biz_target.customers' c left join 'Biz_target.orders' o ON c.customer_id = o.customer_id 13 order by o.order_purchase_timestamp 14 15 16</pre>					
Query results					
JOB INFORMATION RESULTS JSON EXECUTION DETAILS EXECUTION GRAPH PREVIEW					
Row	customer_id	customer_city	customer_state	order_purchase_timestamp	
1	08c5351a6aca1c1589a38f244...	boa vista	RR	2016-09-04 21:15:19 UTC	
2	683c54fc24d40ee9f8a6fc179f...	passo fundo	RS	2016-09-05 00:15:34 UTC	
3	622e13439d6b5a0b486c4356...	sao jose dos campos	SP	2016-09-13 15:24:19 UTC	
4	86dc2f2ce2dfff336de2f386a78...	sao joaquim da barra	SP	2016-09-15 12:16:38 UTC	
5	b106b360fe2ef8849fbbd056f7...	sao paulo	SP	2016-10-02 22:07:52 UTC	
6	355077684019f7f60a031656b...	sao paulo	SP	2016-10-03 09:44:50 UTC	
7	7ec40b22510fdbea1b08921dd...	panambi	RS	2016-10-03 16:56:50 UTC	
8	70fc57eeae292675927697fe0...	rio de janeiro	RJ	2016-10-03 21:01:41 UTC	
9	6f989332712d3222b6571b1cf...	porto alegre	RS	2016-10-03 21:13:36 UTC	
10	b8cf418e97ae795672d326288...	hortolandia	SP	2016-10-03 22:06:03 UTC	
11	7812fcebfc5e8065d31e1bb5f0...	taubate	SP	2016-10-03 22:31:31 UTC	
12	e6f959bf384d1d53b6d688266...	mozarlandia	GO	2016-10-03 22:44:10 UTC	
13	dc607dc98d6a11d5d04d9f2a7...	ipatinga	MG	2016-10-03 22:51:30 UTC	
14	4f3f778022aefa22b9f9e52d2c...	maua	SP	2016-10-04 09:06:10 UTC	
15	b3a9bf200375f53cc5c699191...	sao paulo	SP	2016-10-04 09:16:33 UTC	
Results per page: 50 1 - 50 of 99441					

## 2. In-depth Exploration:

2.1. 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?

If strictly considering the data in that time period, then NO.

I can say that the e-commerce trend in Brazil has made a good track for a short period. But from the given data in the span of three years from 2016 -2018 the order purchase has grown drastically for quite some time and has faced instant breakdown at the last two months. Which makes us think that there may be a possibility of low purchase rate in the future.

If we consider the previous purchase rate there might be a chance to bring back the growing trend, if there are new products available which makes the people in Brazil happy. There should be particular products available only in the region of Brazil which are suitable for the people by making the lives better or release multiple discounts for products which have been purchased from the previous years.

Retrieving the seasonality with monthly peak purchases data

### Through Big Query:

```
select order_id, count(order_id) as order_count, order_purchase_timestamp,
extract (month from order_purchase_timestamp) as month,
from `Biz_target.orders`
group by order_id, order_purchase_timestamp
order by order_purchase_timestamp;
```

Query results SAVE RESULTS

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS		EXECUTION GRAPH	PREVIEW
Row	order_id	order_count	order_purchase_timestamp	month			
1	2e7a8482f6fb09756ca50c10d...	1	2016-09-04 21:15:19 UTC	9			
2	e5fa5a7210941f7d56d0208e4...	1	2016-09-05 00:15:34 UTC	9			
3	809a282bdd5dbcabb6f2f724fc...	1	2016-09-13 15:24:19 UTC	9			
4	bfbdf09bdef84302105ad712db...	1	2016-09-15 12:16:38 UTC	9			
5	71303d7e93b399f5bcd537d12...	1	2016-10-02 22:07:52 UTC	10			
6	3b697a20d9e427646d925679...	1	2016-10-03 09:44:50 UTC	10			
7	be5bc2f0da14d8071e2d45451...	1	2016-10-03 16:56:50 UTC	10			
8	65d1e226dfaeb8cdc42f66542...	1	2016-10-03 21:01:41 UTC	10			
9	a41c8759fbe7aab36ea07e038...	1	2016-10-03 21:13:36 UTC	10			
10	d207cc272675637bfd0062ed...	1	2016-10-03 22:06:03 UTC	10			
11	cd3b8574c82b42fc8129f6d50...	1	2016-10-03 22:31:31 UTC	10			
12	ae8a60e4b03c5a4ba9ca0672c...	1	2016-10-03 22:44:10 UTC	10			
13	ef1b29b591d31d57c0d733746...	1	2016-10-03 22:51:30 UTC	10			
14	0a0837a5eee9e7a9ce2b1fa83...	1	2016-10-04 09:06:10 UTC	10			
15	1ff217aa612f6cd7c4255c9bfe...	1	2016-10-04 09:16:33 UTC	10			
16	ed8c7b1b3eb256c70ce0c7423...	1	2016-10-04 09:59:03 UTC	10			
17	1aecd4362edaca7fa033e882...	1	2016-10-04 10:05:45 UTC	10			
18	...	...	...	...			

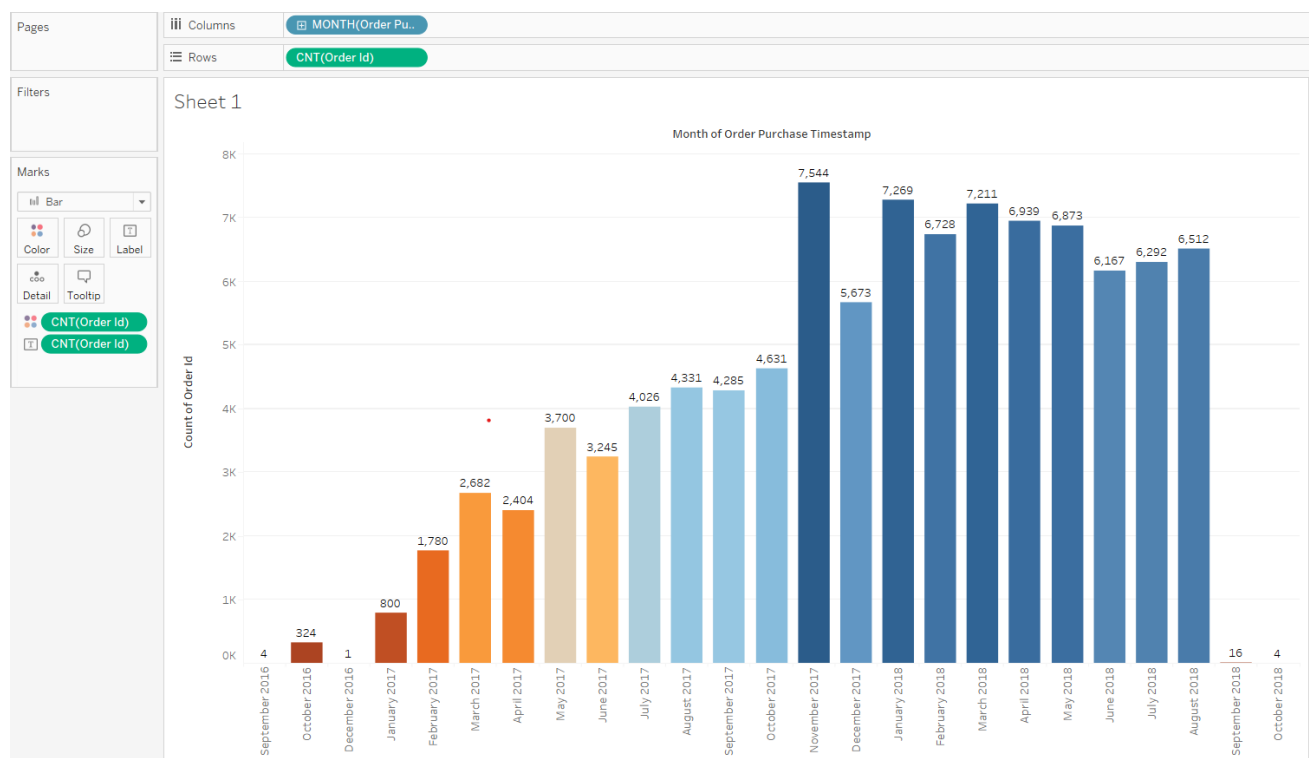
Results per page: 50 1 – 50 of 99441

## Insight:

Used the count of order\_id with purchase timestamp, extracted monthly count with respect to order\_count column. In my query under Month column the same number of months is equal to number orders purchased. I realised that my query is correct after using tableau visualization.

For example, 4 orders were purchased in the 9<sup>th</sup> month (September). We can find in the below tableau view the first month has 4 orders.

Through Tableau:



After visualizing in tableau, we can see that the peak monthly purchase in the given period is from Nov 2017 to Aug 2018 with November, 2017 as highest with 7544 orders.

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

At **Afternoon time** Brazilian customers tend to make more purchases.

Here I categorized the time as below:

Dawn: 5 – 6

Morning: 6 – 12

Afternoon: 12 – 17

Evening: 17 – 21

Night: 21 - 5

The SQL Query for customers tend to buy at the given time period is

#### QUERY:

```
select tbl.hour, count(hour) as hour_count
from
(select order_id,order_purchase_timestamp,
extract(hour from datetime(order_purchase_timestamp)) as hour
from `Biz_target.orders`
group by order_id, order_purchase_timestamp) tbl
where hour between 0 and 23
group by tbl.hour
order by hour_count desc;
```

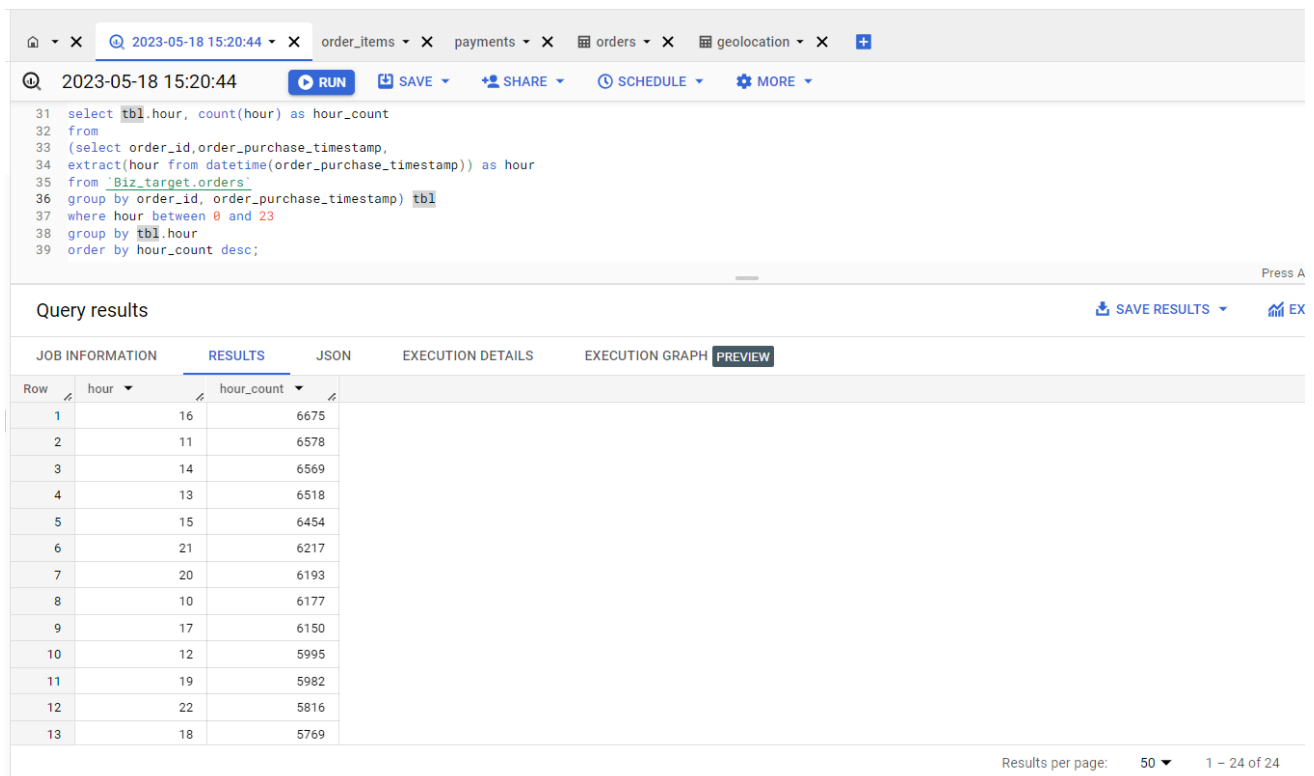
#### Insights:

From the above query we can see that I have extracted **hour** from the given timestamp and grouped it by order\_id and timestamp and completely made the query as a sub query (TBL). And from the sub query I have used the hour column to count the number of hours for each order\_id as hour\_count. Distinguished the hour from 0 to 23 and grouped it by itself and order by with count from high to low as we need the maximum purchases happened in particular hour.

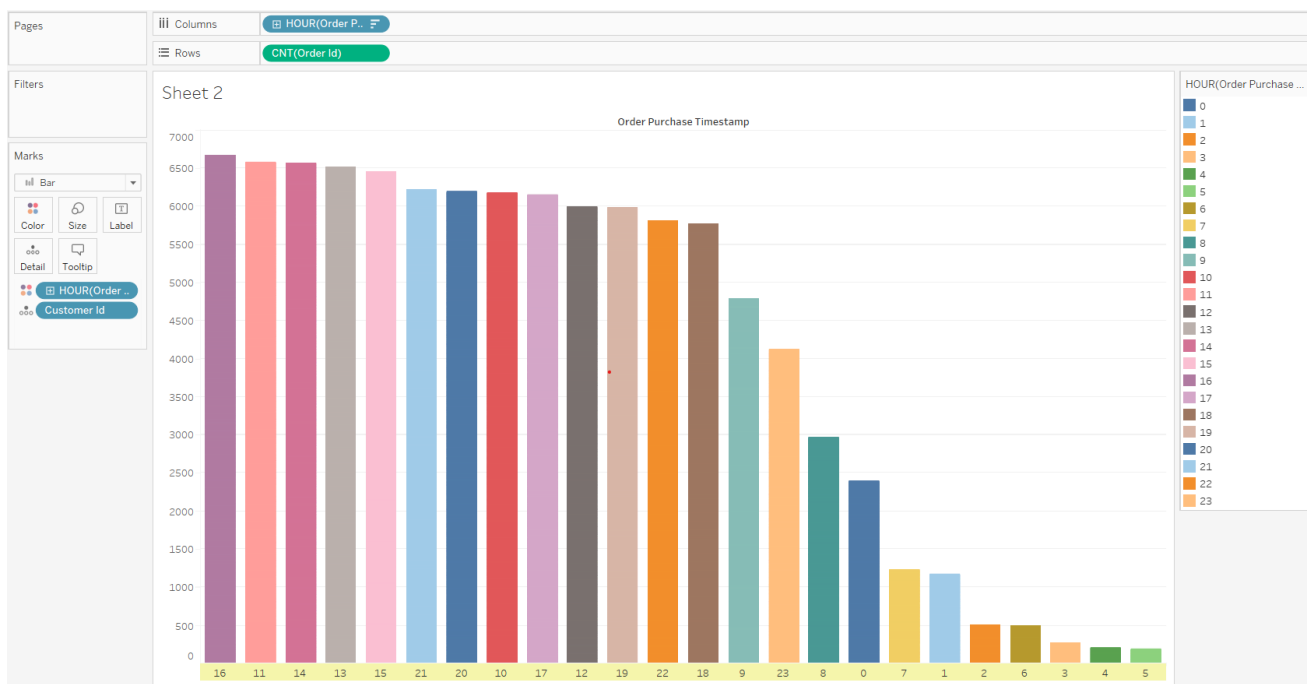
So, for the maximum purchases in a particular hour, we can finalise the hour with our categorized timeline of dawn to night, which is **Afternoon** at **16:00 hour** with purchase count of **6675**. **Through Tableau I have cross checked with timestamp(hour) and order (count) and resulted as Afternoon (16:00) as highest which is same.**

## QUERY RESULT:

From Big Query:



From Tableau:



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

#### 3.1. Get month on month orders by states

Got the data by using inner join for orders table and customers table with customer\_id as common column.

#### Query:

```
select order_id, customer_state, extract (month from
order_purchase_timestamp) as month
from `Biz_target.orders` o inner join `Biz_target.customers` c on
o.customer_id = c.customer_id
group by order_id, customer_state, order_purchase_timestamp ;
```

#### Query Result:

🏠

✕

🔍 2023-05-18 15:20:44

✕

📄 order\_items

✕

💳 payments

✕

📅 orders

✕

📍 geolocation

✕

👤 customers

✕

+

🔍 2023-05-18 15:20:44

▶ RUN

💾 SAVE

👤 SHARE

🕒 SCHEDULE

⚙️ MORE

✅ This script will p

41

42

43

44

45

```
select order_id, customer_state, extract (month from order_purchase_timestamp) as month
from `Biz_target.orders` o inner join `Biz_target.customers` c on o.customer_id = c.customer_id
group by order_id, customer_state, order_purchase_timestamp ;
```

Press ,

Query results

📄 SAVE RESULTS

📄 E

JOB INFORMATION

RESULTS

JSON

EXECUTION DETAILS

EXECUTION GRAPH

PREVIEW

Row	order_id	customer_state	month	
1	6190a94657e1012983a274b8...	AL	7	
2	52cb9b4d5ee3ce7d1e2a8d9c2...	SE	7	
3	274a7f7e4f1c17b7434a830e9...	SE	6	
4	d430c6c36d198f044555a51a5...	AL	7	
5	48a310c40917683b0b399849...	PI	2	
6	eaec582d6403b30186964f6df...	AL	3	
7	e05f1340b7ae14a83463bb498...	RN	6	
8	0538bda829ac14e64a201dc84...	PI	5	
9	7e63ea5a0a34684140ae0e51e...	PI	4	
10	5bd293c86326611e6604399e...	SE	6	
11	1769cdad44f0f8456b101d679f...	AM	3	
12	93c6619a4bcc406da04323380...	RN	1	
13	bf74f34eea55f16dd17b621231...	RN	1	
14	77123692722eeb90408b713bf...	RR	2	
15	08cc8c614786867bb0a845cd1...	SE	8	
16	1bc46ba43c22864d082f8b967...	SE	6	

Results per page: 501 - 50 of 99441

#### 3.2. Distribution of customers across the states in Brazil

Retrieved the data from customers table using count of customer\_id for total number of customers and distinct customer\_state. Finally used group by for the aligned data of state wise customer count.



SQL Query:

```
select distinct customer_state, count(customer_id) as
No_Of_Customers
from `Biz_target.customers`
group by customer_state
order by No_Of_Customers desc ;
```

Query Result:

2023-05-18 15:20:44

RUN

SAVE

SHARE

SCHEDULE

MORE

45

46

47 select distinct customer\_state, count(customer\_id) as No\_Of\_Customers

48 from `Biz\_target.customers`

49 group by customer\_state

50 order by No\_Of\_Customers desc ;

51

Query results

SAVE RESULTS

E

JOB INFORMATION

RESULTS

JSON

EXECUTION DETAILS

EXECUTION GRAPH

PREVIEW

Row	customer_state	No_Of_Customers
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
11	PE	1652
12	CE	1336
13	PA	975
14	MT	907

Results per page: 50 1 – 27 of 27

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

4.1 . 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

#### SQL Query:

```
select year, month, (payment_value/total_price)*100 as percent_increase
from
(
select
extract(year from order_purchase_timestamp) as year,
extract(month from order_purchase_timestamp) as month,
price + freight_value as total_price, payment_value
from `Biz_target.orders` o join `Biz_target.order_items` oi on o.order_id
= oi.order_id join `Biz_target.payments` p on oi.order_id = p.order_id
) tbl1
where tbl1.year in (2017,2018) and month between 1 and 8
group by year, month, payment_value, total_price
order by percent_increase
```

#### Insights:

For this question, I have used 3 tables which are orders table for extracting year and month and order\_items table for using price and freight\_value as total\_price and payments table for using payment\_value.

So, with all the data I have for providing the percentage in increase cost using payment\_value divided by total\_price multiplied by 100. And finally filtered using where for year and month as asked.

## Query Results:

<div> <div>2023-05-18 15:20:44</div> <div> <div>RUN</div> <div>SAVE</div> <div>SHARE</div> <div>SCHEDULE</div> <div>MORE</div> </div> </div>				
<pre> 52 53 select year,month,(payment_value/total_price)*100 as percent_increase 54 from 55 ( 56   select 57     extract(year from order_purchase_timestamp) as year, 58     extract(month from order_purchase_timestamp) as month, 59     price + freight_value as total_price, payment_value 60   from `Biz_target.orders` o join `Biz_target.order_items` oi on o.order_id = oi.order_id join `Biz_target.payments` p on oi.order_id = p.order_id 61   ) tbl1 62 where tbl1.year in (2017,2018) and month between 1 and 8 63 group by year,month,payment_value,total_price 64 order by percent_increase 65 </pre>				
Query results				<div> <div>SAVE RESULTS</div> <div>E</div> </div>
JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
				EXECUTION GRAPH <div>PREVIEW</div>
Row	year	month	percent_increase	
1	2017	4	0.0	
2	2018	1	0.0	
3	2017	5	0.0	
4	2017	6	0.0	
5	2017	5	0.006088650754...	
6	2018	4	0.014306151645...	
7	2018	7	0.018782870022...	
8	2017	8	0.024993751562...	
9	2018	6	0.025630510559...	
10	2017	4	0.069870521190...	

Results per page: 50 1 - 50 of 56971

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

### SQL Query:

```

select
customer_state,
tbl.price_sum,(price_sum/price_count) as mean_price,
tbl.freight_sum,(freight_sum/freight_count) as mean_frieght,
from
(
  select
    sum(price) as price_sum ,
    count(price) as price_count,
    sum(freight_value) as freight_sum,
    count(freight_value) as freight_count,customer_state
  from `Biz_target.order_items` oi join `Biz_target.orders` o on
oi.order_id = o.order_id
  join `Biz_target.customers` c on o.customer_id = c.customer_id
  group by customer_state
) tbl

```

## Insights:

For the mean and sum for price and freight value columns, I have taken 3 tables which are order\_items table for calculating the mean and sum for price and freight value columns and customers table for grouping by customer\_state column and orders table for intersecting the above two tables.

Here, I have calculated the mean by using sum of respective column with count of the rows for the same column.

## Query Result:

<

## 5. Analysis on sales, freight and delivery time

### 5.1 . Calculate days between purchasing, delivering and estimated delivery

#### SQL Query:

```
select order_id,
round(hours*0.041666667) as shipping_days,
round(hour_gap*0.041666667) as estimated_date_days
from
(
  select tbl.order_id,
  extract(hour from days_In_between) hours,
  extract(hour from remaining_Days) hour_gap
  from
  (
    select order_id,
    (order_delivered_carrier_date - order_purchase_timestamp) as days_In_between,
    (order_estimated_delivery_date - order_delivered_carrier_date)as
remaining_Days
    from `Biz_target.orders`
  )tbl
)tbl2
order by order_id
```

#### Insights:

For this problem I have used 3 columns which are carrier\_date, purchase\_timestamp and delivered\_carrier\_date. Simply subtracted the columns for getting the remaining days.

But got the data in hours (E.g.: 0-0-0 126:13:12), so extracted only hours from the first sub query(tbl) and converted those hours into days (1 hour = 0.041666667 days, multiply hours with the value) from the second sub query which gave us the required data in days.

## Query Results:

2023-05-18 15:20:44					products X orders X payments X order_reviews X order_items X				
2023-05-18 15:20:44					RUN SAVE SHARE SCHEDULE MORE				
<pre> 85 select order_id, 86 round(hours*0.041666667) as shipping_days, 87 round(hour_gap*0.041666667) as estimated_date_days 88 from 89 ( 90   select tbl.order_id, 91   extract(hour from days_in_between) hours, 92   extract(hour from remaining_days) hour_gap 93   from 94   ( 95     select order_id, 96     (order_delivered_customer_date - order_purchase_timestamp) as days_in_between </pre>					Press				
Query results					SAVE RESULTS E				
JOB INFORMATION					RESULTS JSON EXECUTION DETAILS EXECUTION GRAPH PREVIEW				
Row	order_id	shipping_days	estimated_date_days						
1	00010242fe8c5a6d1ba2dd792cb16214	6.0	9.0						
2	00018f77f2f0320c557190d7a144bdd3	8.0	10.0						
3	000229ec398224ef6ca0657da4fc703e	2.0	19.0						
4	00024acbcdf0a6daa1e931b038114c75	2.0	9.0						
5	00042b26cf59d7ce69dfabb4e55b4fd9	12.0	29.0						
6	00048cc3ae777c65dbb7d2a0634bc1ea	2.0	20.0						
7	00054e8431b9d7675808bcb819fb4a32	2.0	23.0						
8	000576fe39319847cbb9d288c5617fa6	1.0	19.0						
9	0005a1a1728c9d785b8e2b08b904576c	8.0	1.0						
10	0005f50442cb953dcd1d21e1fb923495	1.0	19.0						
11	00061f2a7bc09da83e415a52dc8a4af1	2.0	13.0						
12	00061f2a7bc09da83e415a52dc8a4af1	2.0	13.0						
					Results per page: 50 1 - 50 of 99441				

5.2. Find time\_to\_delivery & diff\_estimated\_delivery.

## SQL Query:

```

select tbl.order_id,
extract(hour from time_delivery) time_to_delivery,
extract(hour from estimated_delivery) diff_estimated_delivery
from
(
select order_id,
(order_delivered_customer_date - order_purchase_timestamp) as
time_delivery,
(order_estimated_delivery_date - order_delivered_customer_date) as
estimated_delivery
from `Biz_target.orders`
)tbl
order by order_id ;

```

Insights:

From the given formula, used orders table and retrieved data was giving the time in hours format from the first query. So created it as subquery and extracted only hours from the data. Finally ordered it by Order\_id.

Query Results:

2023-05-18 15:20:44

RUNSAVESHARESCHEDULEMORE

```
104 select tbl.order_id,
105       extract(hour from time_delivery) time_to_delivery,
106       extract(hour from estimated_delivery) diff_estimated_delivery
107 from
108 (
109   select order_id,
110         (order_delivered_customer_date - order_purchase_timestamp)::as time_delivery,
111         (order_estimated_delivery_date - order_delivered_customer_date)::as estimated_delivery
112   from "Biz_target.orders"
113 )tbl
114 order by order_id ;
115
```

Query results

SAVE RESULTS EX

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row	order_id	time_to_delivery	diff_estimated_delivery			
1	00010242fe8c5a6d1ba2dd792...	182	192			
2	00018f77f2f0320c557190d7a1...	389	55			
3	000229ec398224ef6ca0657da...	190	322			
4	00024acbcd0a6daa1e931b03...	147	130			
5	00042b26cf59d7ce69dfabb4e...	602	367			
6	00048cc3ae777c65dbb7d2a06...	160	346			
7	00054e8431b9d7675808bcb8...	202	385			
8	000576fe39319847cbb9d288c...	121	369			
9	0005a1a1728c9d785b8e2b08...	239	-18			
10	0005f50442cb953dcd1d21e1f...	51	438			
11	00061f2a7bc09da83e415a52d...	97	263			

Results per page: 50 1 - 50 of 99441



5.3. Group data by state, take mean of freight\_value, time\_to\_delivery, diff\_estimated\_delivery

#### SQL Query:

```
select customer_state,
(tbl3.delivery_time_sum/tbl3.delivery_time_count) as mean_delivery_time,
(tbl3.delivery_estimate_sum/tbl3.delivery_estimate_count) as mean_estimate_time,
(tbl3.freight_sum/tbl3.freight_count) as mean_freight_value
from
(
select customer_state, tbl2.freight_count, tbl2.freight_sum,
sum(time_to_delivery) as delivery_time_sum, count(time_to_delivery) as
delivery_time_count,
sum(diff_estimated_delivery) as delivery_estimate_sum,
count(diff_estimated_delivery) as delivery_estimate_count
from
(
select tbl1.customer_state, tbl1.freight_count, tbl1.freight_sum,
extract(hour from time_delivery) time_to_delivery,
extract(hour from estimated_delivery) diff_estimated_delivery
from
(
select customer_state,
sum(freight_value) as freight_sum,
count(freight_value) as freight_count,
(order_delivered_customer_date - order_purchase_timestamp) as
time_delivery,
(order_estimated_delivery_date - order_delivered_customer_date) as
estimated_delivery
from `Biz_target.order_items` oi join `Biz_target.orders` o on oi.order_id
= o.order_id
join `Biz_target.customers` c on o.customer_id = c.customer_id
group by customer_state, order_delivered_customer_date,
order_purchase_timestamp, order_estimated_delivery_date
) tbl1
) tbl2
group by customer_state, tbl2.freight_count, tbl2.freight_sum
) tbl3
order by customer_state ;
```

#### Insights:

For this problem I have considered previous queries from problems 4.2 & 5.2 and created 3<sup>rd</sup> sub query for calculating the mean for freight value, delivery time, estimated time. As we cannot calculate the newly created columns on the same query itself, we got 3 subqueries for these calculations.



## Query Results:

2023-05-18 15:20:44					
RUN SAVE SHARE SCHEDULE MORE					
select customer_state, (tbl2.delivery_time_sum/tbl2.delivery_time_count) as mean_delivery_time					
Query results					
SAVE RESULTS E					
JOB INFORMATION RESULTS JSON EXECUTION DETAILS EXECUTION GRAPH PREVIEW					
Row	customer_state	mean_delivery_time	mean_estimate_time	mean_freight_value	
1	AC	null	null	42.39	
2	AC	451.0	800.0	32.46	
3	AC	389.0	526.0	26.04	
4	AC	363.0	775.0	27.75	
5	AC	533.0	574.0	52.98	
6	AC	548.0	512.0	36.16	
7	AC	984.0	-25.0	93.08	
8	AC	1732.0	-767.0	27.58	
9	AC	720.0	249.0	26.61	
10	AC	413.0	842.0	77.19	
11	AC	429.0	413.6	25.38	
12	AC	299.0	793.0	53.72	
13	AC	481.0	393.0	33.24	
14	AC	695.0	193.0	35.85	
15	AC	414.0	484.0	27.4	
16	AC	336.0	481.0	45.78	
17	AC	360.0	385.0	30.3	
Results per page: 50 1 - 50 of 27726					

5.4. Sort the data to get the following:

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

### SQL Query:

#### Highest average

```
select
customer_state,
(freight_sum/freight_count) as mean_freight,
from
(
  select
    sum(freight_value) as freight_sum,
    count(freight_value) as freight_count, customer_state
  from `Biz_target.order_items` oi join `Biz_target.orders` o on
oi.order_id = o.order_id
  join `Biz_target.customers` c on o.customer_id = c.customer_id
  group by customer_state
) tbl
order by mean_freight desc
limit 5 ;
```

## SQL Query:

### Lowest average

```
select
customer_state,
(freight_sum/freight_count) as mean_freight,
from
(
  select
    sum(freight_value) as freight_sum,
    count(freight_value) as freight_count, customer_state
  from `Biz_target.order_items` oi join `Biz_target.orders` o on
oi.order_id = o.order_id
  join `Biz_target.customers` c on o.customer_id = c.customer_id
  group by customer_state
) tbl
order by mean_freight
limit 5 ;
```

## Query Results:

### Highest

The screenshot shows a SQL query editor with a toolbar at the top containing icons for home, search, and tabs for various database tables (products, orders, payments, order\_reviews, order\_items). Below the toolbar is a search bar with the text "2023-05-18 15:20:44" and buttons for "RUN", "SAVE", "SHARE", "SCHEDULE", and "MORE". The main area displays the SQL query, which is highlighted in blue. The query is identical to the one shown in the previous block. Below the query editor, there is a section titled "Query results" which contains a table with 5 rows and 2 columns: "customer\_state" and "mean\_freight".

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row	customer_state	mean_freight				
1	RR	42.98442307692...				
2	PB	42.72380398671...				
3	RO	41.06971223021...				
4	AC	40.07336956521...				
5	PI	39.14797047970...				

## Lowest

🏠

🔍 \*2023-05-18 15:20:44

📊 products

📊 orders

📊 payments

📊 order\_reviews

🔍 2023-05-18 15:20:44

▶ RUN

💾 SAVE

👤 SHARE

🕒 SCHEDULE

⚙️ MORE

```
147
148
149 select
150 customer_state,
151 (freight_sum/freight_count) as mean_freight,
152 from
153 (
154   select
155     sum(freight_value) as freight_sum,
156     count(freight_value) as freight_count, customer_state
157   from Biz_target.order_items oi join Biz_target.orders o on oi.order_id = o.order_id
158   join Biz_target.customers c on o.customer_id = c.customer_id
159   group by customer_state
160 ) tbl
161 order by mean_freight
162 limit 5;
```

## Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row	customer_state	mean_freight				
1	SP	15.14727539041...				
2	PR	20.53165156794...				
3	MG	20.63016680630...				
4	RJ	20.96092393168...				
5	DF	21.04135494596...				

## 2. 5.6. Top 5 states with highest/lowest average time to delivery

### SQL Query: highest

```
select customer_state,
(tbl3.delivery_time_sum/tbl3.delivery_time_count) as Average_delivery_time,
from
(
  select customer_state,
  sum(time_to_delivery) as delivery_time_sum, count(time_to_delivery) as
delivery_time_count
  from
    (
      select tbl1.customer_state,
      extract(hour from time_delivery) time_to_delivery
      from
        (
          select customer_state,
          (order_delivered_customer_date - order_purchase_timestamp) as
time_delivery,
          from `Biz_target.orders` o join `Biz_target.customers` c on o.customer_id =
c.customer_id
          group by customer_state, order_delivered_customer_date,
order_purchase_timestamp
        ) tbl1
      ) tbl2
    group by customer_state
  ) tbl3
order by Average_delivery_time desc
limit 5 ;
```

### SQL Query: lowest

```
select customer_state,
(tbl3.delivery_time_sum/tbl3.delivery_time_count) as Average_delivery_time,
from
(
  select customer_state,
  sum(time_to_delivery) as delivery_time_sum, count(time_to_delivery) as
delivery_time_count
  from
    (
      select tbl1.customer_state,
      extract(hour from time_delivery) time_to_delivery
      from
        (
          select customer_state,
          (order_delivered_customer_date - order_purchase_timestamp) as
time_delivery,
          from `Biz_target.orders` o join `Biz_target.customers` c on o.customer_id =
c.customer_id
```

```

        group by customer_state, order_delivered_customer_date,
order_purchase_timestamp
    ) tbl1
    ) tbl2
    group by customer_state
    ) tbl3
order by Average_delivery_time
limit 5 ;

```

## Query Results:

### Highest

🏠 ✕ 🔍 \*2023-05-18 15:20:44 ✕
📊 products ✕
📊 orders ✕
📊 payments ✕
📊 order\_reviews ✕

🕒 2023-05-18 15:20:44
▶ RUN
💾 SAVE
👤 SHARE
🕒 SCHEDULE
⚙️ MORE

```

165 select customer_state,
166 (tbl3.delivery_time_sum/tbl3.delivery_time_count) as Average_delivery_time,
167 from
168 (
169   select customer_state,
170   sum(time_to_delivery) as delivery_time_sum, count(time_to_delivery) as delivery_time_count
171   from
172   (
173     select tbl1.customer_state,
174     extract(hour from time_delivery) time_to_delivery
175     from
176     (
177       select customer_state,
178       (order_delivered_customer_date - order_purchase_timestamp) as time_delivery,
179       from Biz_target.orders o join Biz_target.customers c on o.customer_id = c.customer_id
180       group by customer_state, order_delivered_customer_date, order_purchase_timestamp
181     ) tbl1
182   ) tbl2
183   group by customer_state
184   ) tbl3
185 order by Average_delivery_time desc
186 limit 5 ;

```

### Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row	customer_state	Average_delivery_time				
1	RR	704.7317073170...				
2	AP	651.9701492537...				
3	AM	633.6965517241...				
4	AL	588.5415617128...				
5	PA	570.0507399577...				

## Lowest

2023-05-18 15:20:44

RUN

SAVE

SHARE

SCHEDULE

MORE

```
165 select customer_state,
166 (tbl3.delivery_time_sum/tbl3.delivery_time_count) as Average_delivery_time,
167 from
168 .. (
169 .. select customer_state,
170 .. sum(time_to_delivery) as delivery_time_sum, count(time_to_delivery) as delivery_time_count
171 .. from
172 .. .. (
173 .. .. select tbl1.customer_state,
174 .. .. extract(hour from time_delivery) time_to_delivery
175 .. .. from
176 .. .. .. (
177 .. .. .. select customer_state,
178 .. .. .. (order_delivered_customer_date - order_purchase_timestamp) as time_delivery,
179 .. .. .. from `Biz_target.orders` o join `Biz_target.customers` c on o.customer_id = c.customer_id
180 .. .. .. group by customer_state, order_delivered_customer_date, order_purchase_timestamp
181 .. .. .. ) tbl1
182 .. .. .. ) tbl2
183 .. .. group by customer_state
184 .. ) tbl3
185 order by Average_delivery_time
186 limit 5;
```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row	customer_state	Average_delivery_time				
1	SP	209.7709840721...				
2	PR	287.3026609790...				
3	MG	287.7454865697...				
4	DF	310.7235576923...				
5	SC	358.5167747392...				

3. 5.7. Top 5 states where delivery is really fast/ not so fast compared to estimated date

### SQL Query:

```
select order_id, case
when order_estimated_delivery_date > order_delivered_customer_date
then 'Really Fast'
when order_estimated_delivery_date < order_delivered_customer_date
then 'Not So Fast'
end as delivery_status
from `Biz_target.orders`
order by order_id
```

## Query Results:

2023-05-18 15:20:44

RUN

SAVE

SHARE

SCHEDULE

MORE

This script will pr

```
189 select order_id, case
190 when order_estimated_delivery_date > order_delivered_customer_date
191 then 'Really Fast'
192 when order_estimated_delivery_date < order_delivered_customer_date
193 then 'Not So Fast'
194 end as delivery_status
195 from `Biz_target.orders`
196 order by order_id ;
```

Query results

SAVE RESULTS

E

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
Row	order_id	delivery_status			
1	00010242fe8c5a6d1ba2dd792cb16214	Really Fast			
2	00018f77f2f0320c557190d7a144bdd3	Really Fast			
3	000229ec398224ef6ca0657da4fc703e	Really Fast			
4	00024acbcd0a6daa1e931b038114c75	Really Fast			
5	00042b26cf59d7ce69dfabb4e55b4fd9	Really Fast			
6	00048cc3ae777c65dbb7d2a0634bc1ea	Really Fast			
7	00054e8431b9d7675808bcb819fb4a32	Really Fast			
8	000576fe39319847cbb9d288c5617fa6	Really Fast			
9	0005a1a1728c9d785b8e2b08b904576c	Not So Fast			
10	0005f50442cb953dcd1d21e1fb923495	Really Fast			
11	00061f2a7bc09da83e415a52dc8a4af1	Really Fast			
12	00063b381e2406b52ad429470734ebd5	Not So Fast			
13	0006ec9db01a64e59a68b2c340bf65a7	Really Fast			
14	0008288aa423d2a3f0fcb17cd7d8719	Really Fast			

Results per page: 50 1 - 50 of 99441

## 6. Payment type analysis:

### 6.1 . Month over Month count of orders for different payment types

#### SQL Query:

```
select o.order_id, count(o.order_id) as order_count, extract (month from
order_purchase_timestamp) as month,payment_type
from `Biz_target.orders` o inner join `Biz_target.payments` p on o.order_id =
p.order_id
group by order_id,payment_type, order_purchase_timestamp ;
```

#### Insights:

Here I have extracted month and count of orders from orders table and payment\_ type column from payments table. Using join for orders and payments tables and grouping the columns respectively

## Query Results:

2023-05-18 15:20:44					
RUN SAVE SHARE SCHEDULE MORE					
This query will					
198 select o.order_id, count(o.order_id) as order_count, extract(month from order_purchase_timestamp) as month, payment_type 199 from `Biz_target.orders` o inner join `Biz_target.payments` p on o.order_id = p.order_id 200 group by order_id, payment_type, order_purchase_timestamp ; 201					
Query results					
SAVE RESULTS					
JOB INFORMATION RESULTS JSON EXECUTION DETAILS EXECUTION GRAPH PREVIEW					
Row	order_id	order_count	month	payment_type	
5	591083bc42b589c7052118aa...	6	1	voucher	
6	330534a2e7ad98d0c76ee20bf...	4	1	voucher	
7	11483c6a164cf839a3b0c945a...	3	1	voucher	
8	acc08ec914c1ab328cfbaf6a5...	1	1	voucher	
9	3f515b21edaaa117bbab2833c...	3	1	voucher	
10	91fb22794b4915b6ddddd8e747...	2	1	voucher	
11	193dce17420e5a1662145bdba...	2	1	voucher	
12	2122719f6fc72fba16b0b3001d...	1	1	voucher	
13	1dd4114d014ff38caa0b9a94b...	1	1	voucher	
14	563404de47cdc6763e7eced48...	1	1	voucher	
15	b8872794c438d699251b948e...	1	1	credit_card	
16	38c61d1bbf5143a8c347ebaa4...	3	1	voucher	
17	2d61f270bc350937faeb8bdc...	1	1	credit_card	
18	418f645ce19284efcb0faacc40...	1	1	credit_card	
19	57f3dce57b231e9b80186f99d...	1	1	voucher	
20	8ea33c8a1daf30c76ff35b6574...	1	1	credit_card	
Results per page: 50 1 - 50 of 101686					

## 6.2. Count of orders based on the no. of payment installments

### SQL Query:

```
select o.order_id, count(o.order_id) as order_count, payment_installments
from `Biz_target.orders` o inner join `Biz_target.payments` p on
o.order_id = p.order_id
group by order_id, payment_installments, order_purchase_timestamp ;
```



Query Results:

🏠

✕

🔍 +2023-05-18 15:20:44 ✕

📄 products ✕

📄 orders ✕

📄 payments ✕

📄 order\_reviews ✕

📄 order\_items ✕

+

🔍 2023-05-18 15:20:44

▶ RUN

💾 SAVE

👤 SHARE

🕒 SCHEDULE

⚙️ MORE

281

282

283

284 select o.order\_id, count(o.order\_id) as order\_count, payment\_installments

285 from `Biz\_target.orders` o inner join `Biz\_target.payments` p on o.order\_id = p.order\_id

286 group by order\_id, payment\_installments, order\_purchase\_timestamp;

287

288

Press ↵

Query results

📄 SAVE RESULTS

📊 E

JOB INFORMATION

RESULTS

JSON

EXECUTION DETAILS

EXECUTION GRAPH

PREVIEW

Row	order_id	order_count	payment_installment
1	7c259a397799aa0b7043c30ff...	1	12
2	9dd59f7ebb5a498a78c92b880...	1	20
3	80e3778eb13610b0026e877f2...	1	15
4	32510b081654e6d2e96fad5a6...	1	12
5	ba2821b49c92e278db491d3e...	1	11
6	1cd8d1567debd198821fe9d46...	1	13
7	10b21177ed3e0f82c350c7f59...	1	12
8	bfb8512783ccd2271d2e96d3b...	1	15
9	59dd30ed66bb5ab4c41d8333...	1	15
10	1a57108394169c0b47d8f876a...	1	0
11	78135b03c6d7a39f10996624b...	1	15
12	35153d02b70cad4eb36cc1782...	1	12
13	b76190a2c095fba255ab46987...	1	12
14	fa5f88521abe0927f354980a96...	1	18

Results per page: 50 1 - 50 of 100308