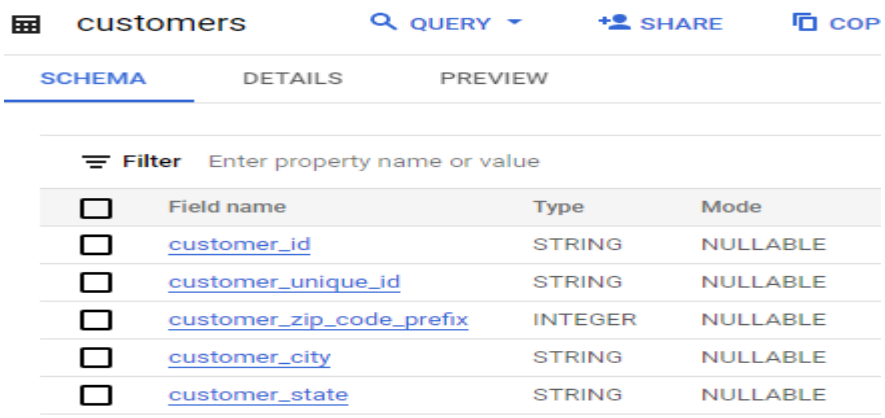
SQL Target case study (Shubham Patel)

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.

Create the Database in BigQuery.

Table: **customers**



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

Comments:

Here each **customer_unique_id** can be considered as Primary Key, as there can be multiple **customer_id** corresponding to each **customer_unique_id.**

Table: **sellers**

Field name	Type	Mode
seller_id	STRING	NULLABLE
seller_zip_code_prefix	INTEGER	NULLABLE
seller_city	STRING	NULLABLE
seller_state	STRING	NULLABLE

Table: **order_items**

order_items QUERY SHARE

SCHEMA DETAILS PREVIEW

Field name	Type	Mode
order_id	STRING	NULLABLE
order_item_id	INTEGER	NULLABLE
product_id	STRING	NULLABLE
seller_id	STRING	NULLABLE
shipping_limit_date	TIMESTAMP	NULLABLE
price	FLOAT	NULLABLE
freight_value	FLOAT	NULLABLE

Comments:

Order_id should be a Primary key, as each order should be uniquely identified.

Table: **geolocations**

geolocation QUERY SHARE COPY

SCHEMA DETAILS PREVIEW

Filter Enter property name or value

Field name	Type	Mode	Co
geolocation_zip_code_prefix	INTEGER	NULLABLE	
geolocation_lat	FLOAT	NULLABLE	
geolocation_lng	FLOAT	NULLABLE	
geolocation_city	STRING	NULLABLE	
geolocation_state	STRING	NULLABLE	

Table: **payments**

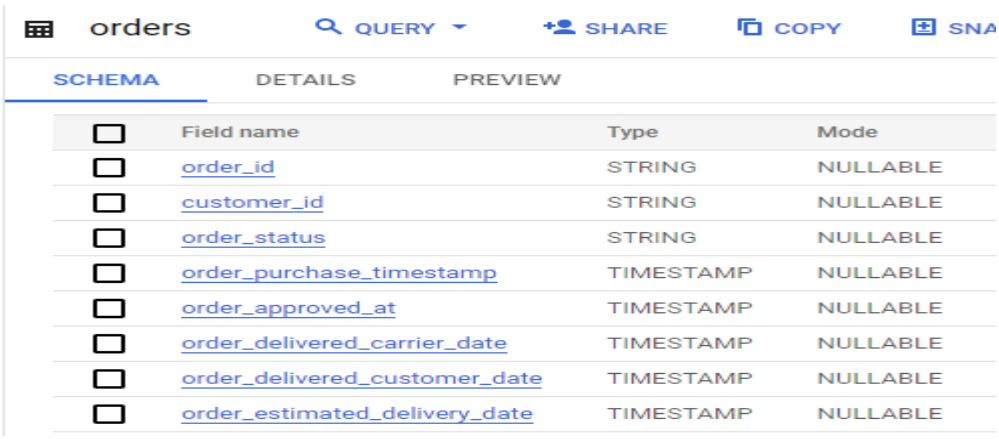
payments QUERY SHARE

SCHEMA DETAILS PREVIEW

Filter Enter property name or value

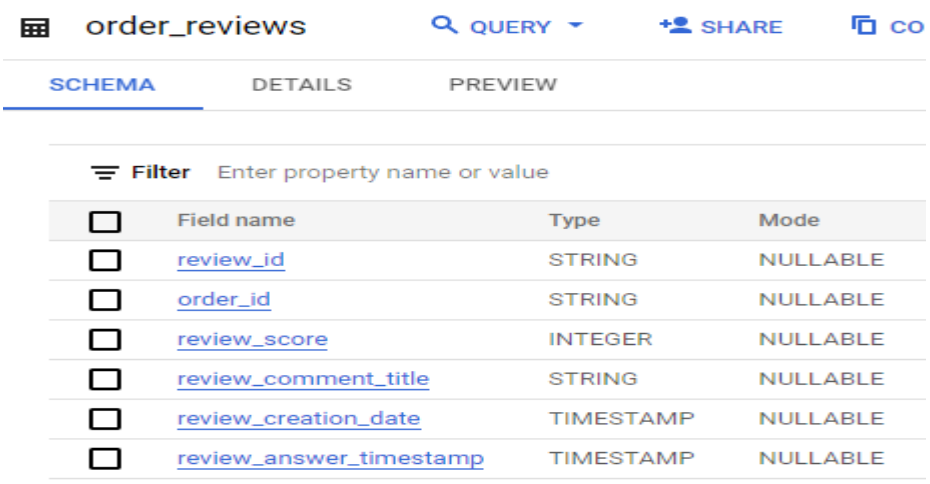
Field name	Type	Mode
order_id	STRING	NULLABLE
payment_sequential	INTEGER	NULLABLE
payment_type	STRING	NULLABLE
payment_installments	INTEGER	NULLABLE
payment_value	FLOAT	NULLABLE

Table: **orders**



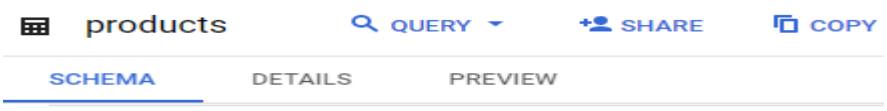
Field name	Type	Mode
order_id	STRING	NULLABLE
customer_id	STRING	NULLABLE
order_status	STRING	NULLABLE
order_purchase_timestamp	TIMESTAMP	NULLABLE
order_approved_at	TIMESTAMP	NULLABLE
order_delivered_carrier_date	TIMESTAMP	NULLABLE
order_delivered_customer_date	TIMESTAMP	NULLABLE
order_estimated_delivery_date	TIMESTAMP	NULLABLE

Table: **order_reviews**



Field name	Type	Mode
review_id	STRING	NULLABLE
order_id	STRING	NULLABLE
review_score	INTEGER	NULLABLE
review_comment_title	STRING	NULLABLE
review_creation_date	TIMESTAMP	NULLABLE
review_answer_timestamp	TIMESTAMP	NULLABLE

Table: **products**



product_id	STRING	NULLABLE
product_category	STRING	NULLABLE
product_name_length	INTEGER	NULLABLE
product_description_length	INTEGER	NULLABLE
product_photos_qty	INTEGER	NULLABLE
product_weight_g	INTEGER	NULLABLE
product_length_cm	INTEGER	NULLABLE
product_height_cm	INTEGER	NULLABLE
product_width_cm	INTEGER	NULLABLE

1.2 Time period for which the data is given.

Code:

```
1. -- Time period for which the data is given
2.
3. select min(o.order_purchase_timestamp) as First_order_date,
4.        max(o.order_purchase_timestamp) as last_order_date
5. from `sql-case-study-360620.case_study.orders` o;
6.
```

Output:

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DET

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

Comments:

The first order date is 4th Sep 2016 to 17th Oct 2018.

1.3 Cities and States covered in the dataset.

Code:

```
1. -- States in dataset
2.
3. select distinct g.geolocation_state
4. from `sql-case-study-360620.case_study.geolocation` g;
```

Query results

SAVE RESULTS

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

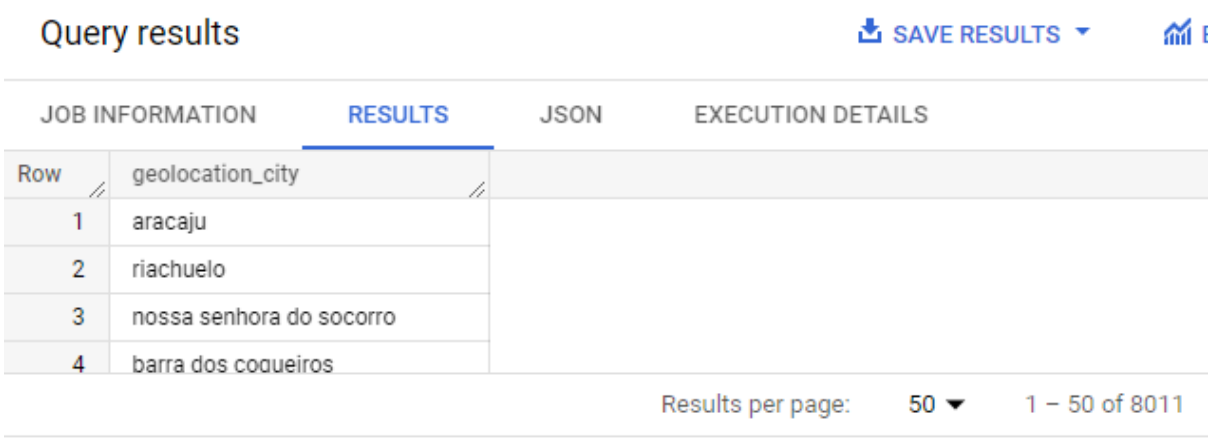
Row	geolocation_state
1	SE
2	AL
3	PI
4	AP
5	AM

Results per page: 50 1 – 27 of 27

Geoloactions maps to both Customers and sellers, we have 27 states in total.

Cities in dataset:

```
1.    select distinct g.geolocation_city
2.    from `sql-case-study-360620.case_study.geolocation` g;
3.
```



We have data across 8011 cities across Brazil.

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?

Code:

```
1.  -- order_status = delivered for successful order purchases.
2.  -- Not considering the freight charges, as only concerned about the growing trend.
3.
4.  select
5.    ROUND(sum(price)/1000000, 2) as Price_in_milions,
6.    EXTRACT(YEAR from order_purchase_timestamp) as YEAR
7.  from `sql-case-study-360620.case_study.orders` ord
8.  left join `sql-case-study-360620.case_study.order_items` ord_item
9.  ON ord.order_id = ord_item.order_id
10. where order_status = 'delivered'
11. group by YEAR
12. order by YEAR;
13.
14. -- Now getting more insights, we try to extract the growth over months
15. select
16.   ROUND(sum(price)/1000000, 2) as Price_in_milions,
17.   FORMAT_DATE('%Y%m',order_purchase_timestamp) as MONTH
18. from `sql-case-study-360620.case_study.orders` ord
19. left join `sql-case-study-360620.case_study.order_items` ord_item
20. ON ord.order_id = ord_item.order_id
21. where order_status = 'delivered'
22. group by MONTH
23. order by MONTH;
24.
```

Query results:

Query results

JOB INFORMATION | RESULTS

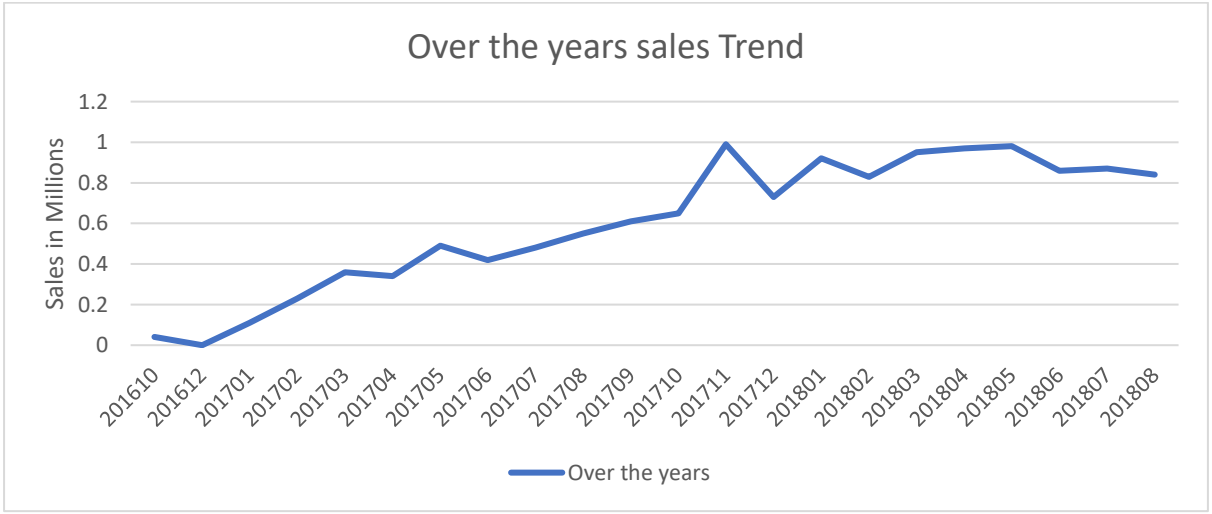
Row	Price_in_mil...	YEAR
1	0.04	2016
2	5.96	2017
3	7.22	2018

Query results

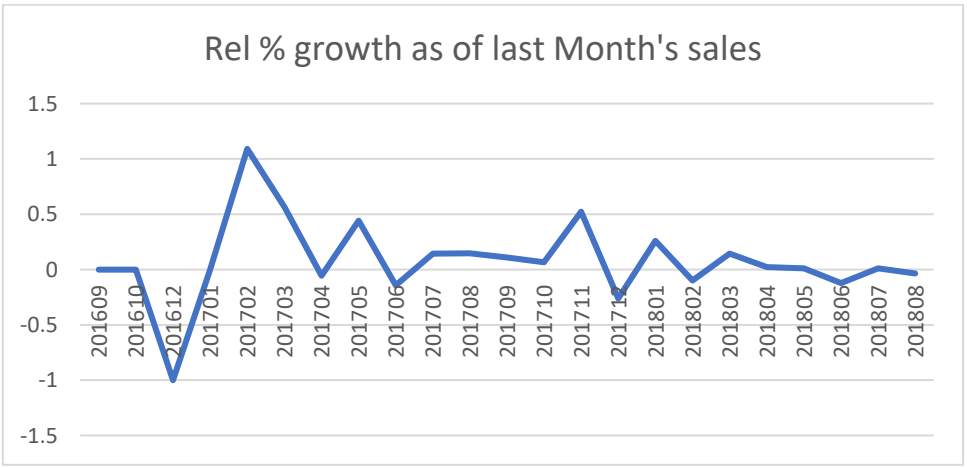
JOB INFORMATION | RESULTS | JSON

Row	Price_in_mil...	MONTH
7	0.34	201704
8	0.49	201705
9	0.42	201706
10	0.48	201707
11	0.55	201708
12	0.61	201709
13	0.65	201710
14	0.99	201711
15	0.73	201712

Analysing the data using Excel, we see a 21.14% of growth over year from 2017 to 2018. Since we only have 8 months of Data in 2018, so expected yearly growth from 2017 to 2018 becomes 31.71%, which clearly shows adoption of online sales by customers in Brazil.



Analysing more on the seasonality, sales dependency over months.



There's not much data to comment on Seasonality as we have growth starting on 2017 Jan so there's a spike in sales, but there's a inc. to sales in Jan 2018 as well, which leads us to conclude that sales increases during New year festive seasons.

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

   We categorise the time ranges:

Dawn	4:00 am to 05.30 am
Morning	5:30 am to 11:30 am
Afternoon	11:30 am to 05:30 pm
Evening	05:30 pm to 7:30 pm
Night	07:30 pm to 11:00 pm
Late night	11:00 pm to 4:00 am

Code:

```
1.     -- categorisation wrt Time ranges during the Day.
2.  select
3.    ROUND(sum(price)/1000000, 2) as Price_in_milions,
4.    CASE WHEN TIME(order_purchase_timestamp) BETWEEN TIME '04:00:00' AND TIME '05:29:59'
5.  THEN 'Dawn'
6.         WHEN TIME(order_purchase_timestamp) BETWEEN TIME '05:30:00' AND TIME '11:29:59'
7.  THEN 'Morning'
8.         WHEN TIME(order_purchase_timestamp) BETWEEN TIME '11:30:00' AND TIME '17:29:59'
9.  THEN 'Afternoon'
10.        WHEN TIME(order_purchase_timestamp) BETWEEN TIME '17:30:00' AND TIME '19:29:59'
11. THEN 'Evening'
12.        WHEN TIME(order_purchase_timestamp) BETWEEN TIME '19:30:00' AND TIME '22:59:59'
13. THEN 'Night'
14.        WHEN (TIME(order_purchase_timestamp) BETWEEN TIME '23:00:00' AND TIME '23:59:59'
    OR
15.              TIME(order_purchase_timestamp) BETWEEN TIME '00:00:00' AND TIME '03:59:59')
     THEN 'Late_night'
16.        ELSE 'WRONG_TIME'
17.        END AS TIME_GROUPS
18. from `sql-case-study-360620.case_study.orders` ord
19. left join `sql-case-study-360620.case_study.order_items` ord_item
20. ON ord.order_id = ord_item.order_id
21. where order_status = 'delivered'
22. group by TIME_GROUPS;
23.
```

Query results:

Query results

JOB INFORMATION | RESULTS | JSON

Row	Price_in_mil...	TIME_GROUPS
1	2.81	Night
2	5.22	Afternoon
3	1.6	Evening
4	1.06	Late_night
5	2.51	Morning
6	0.03	Dawn

Price_in_milions	TIME_GROUPS	% share Total	Hours btw range	Rel sale per hour (Millions)
2.81	Night	21.24%	3.5	0.80
5.22	Afternoon	39.46%	6	0.87
1.6	Evening	12.09%	2	0.80
1.06	Late_night	8.01%	5	0.21
2.51	Morning	18.97%	6	0.42
0.03	Dawn	0.23%	1.5	0.02

Analysing the results, we can observe that Afternoon time period brings more sale, almost 40% of total sales among the Time groups, also since Afternoon we categorized 6 hours, it was important to scale the results. So we have Relative sales per hour, which is also higher for Afternoon.

Overall, we could target to promote offers during 11:30 am to 23:00 pm.

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

1. Get month on month orders by region, states.

Code:

```
-- Month on month orders by region/ city.

select
  cust.customer_city,
  count(*) as Total_orders
from `sql-case-study-360620.case_study.orders` ord
left join `sql-case-study-360620.case_study.customers` cust
ON ord.customer_id = cust.customer_id
where order_status = 'delivered'
group by cust.customer_city
order by 2 desc;

 -- Total_orders
select
  count(*) as Total_orders                              -- # 96,478 delivered orders.
from `sql-case-study-360620.case_study.orders` ord
where order_status = 'delivered';

-- No. of cities across region, for delievered orders.
select
  count(distinct cust.customer_city) as Total_cities
from `sql-case-study-360620.case_study.orders` ord
left join `sql-case-study-360620.case_study.customers` cust
ON ord.customer_id = cust.customer_id
where order_status = 'delivered';

```

Query results:

Query results

JOB INFORMATION | RESULTS | JSON

Row	customer_city	Total_orders
1	sao paulo	15045
2	rio de janeiro	6601
3	belo horizonte	2697
4	brasilia	2071
5	curitiba	1489

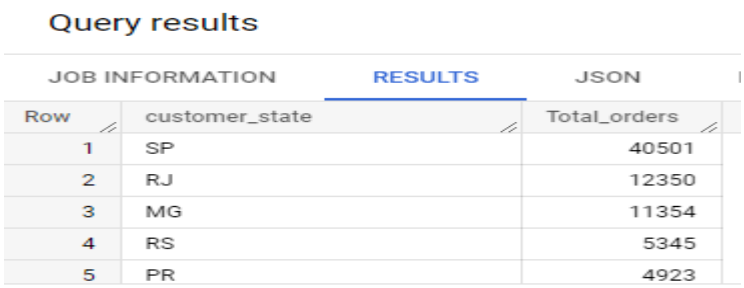
Code:

```sql
-- Month on month orders by state.
select
  cust.customer_state,
  count(*) as Total_orders
from `sql-case-study-360620.case_study.orders` ord
left join `sql-case-study-360620.case_study.customers` cust
ON ord.customer_id = cust.customer_id
where order_status = 'delivered'
group by cust.customer_state
order by 2 desc;

-- No. of states across region, for delievered orders.
select
  count(distinct cust.customer_state) as Total_states
from `sql-case-study-360620.case_study.orders` ord
left join `sql-case-study-360620.case_study.customers` cust
ON ord.customer_id = cust.customer_id
where order_status = 'delivered';

```

Query results:



Query results

JOB INFORMATION | RESULTS | JSON

Row	customer_state	Total_orders
1	SP	40501
2	RJ	12350
3	MG	11354
4	RS	5345
5	PR	4923

2. How are customers distributed in Brazil

We have total 96,478 orders successfully delivered across 4,085 cities in 27 States of Brazil.

Top 5 cities accounts for 29% of overall sales, while top 5 states accounts for 71% of overall sales.

These regions can be targeted for promotional offers and have potentially better penetration for e-commerce.

Results analysed in Excel:

customer_city	Total_ord	% total
sao paulo	15045	15.59%
rio de janeiro	6601	6.84%
belo horizonte	2697	2.80%
brasilia	2071	2.15%
curitiba	1489	1.54%

Total_orders
96478

customer_state	Total_ord	% total
SP	40501	41.98%
RJ	12350	12.80%
MG	11354	11.77%
RS	5345	5.54%
PR	4923	5.10%

4. **Impact on Economy: Analyse 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)**

        Cost of orders = Price – Freight_value, is the true measure of revenue generated for e-commerce, where Freight_value is the cost on company for carrier charges.

Code:

```
1.  -- Get % increase in cost of orders from 2017 to 2018
2.  -- (include months between Jan to Aug only)
3.
4.  select
5.    ROUND(sum(price - freight_value)/1000000, 2) as Actual_rev_in_mil,
6.    FORMAT_DATE('%Y%m',order_purchase_timestamp) as MONTH
7.  from `sql-case-study-360620.case_study.orders` ord
8.  left join `sql-case-study-360620.case_study.order_items` ord_item
9.  ON ord.order_id = ord_item.order_id
10. where order_status = 'delivered'
11. group by MONTH
12. order by MONTH;
13.
```

Query results:

Query results

JOB INFORMATION | RESULTS

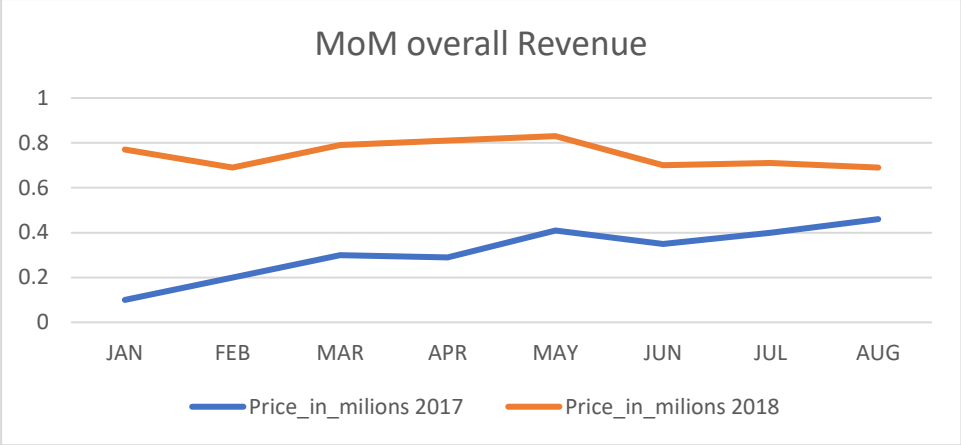
Row	Actual_rev_i...	MONTH
5	0.2	201702
6	0.3	201703
7	0.29	201704
8	0.41	201705
9	0.35	201706

Comments:

Using this data, there is healthy growth over Months on Months from 2017 to 2018.

		JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	Overall
Price_in_milions	2017	0.1	0.2	0.3	0.29	0.41	0.35	0.4	0.46	2.51
	2018	0.77	0.69	0.79	0.81	0.83	0.7	0.71	0.69	5.99
	% Inc MOM	670%	245%	163%	179%	102%	100%	78%	50%	139%

There is overall an increase of 139% in total revenue over time.



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

Code:

```
1.  -- Mean & Sum of price and freight value by customer state
2.  select
3.    cust.customer_state,
4.    ROUND(sum(ord_items.price), 2) as Total_ord_price,
5.    ROUND(AVG(ord_items.price), 2) as avg_ord_price,
6.    ROUND(sum(ord_items.freight_value), 2) as Total_freight_value,
7.    ROUND(AVG(ord_items.freight_value), 2) as avg_freight_value,
8.  from `sql-case-study-360620.case_study.customers` cust
9.  join `sql-case-study-360620.case_study.orders` ord
10. ON cust.customer_id = ord.customer_id
11. join `sql-case-study-360620.case_study.order_items` ord_items
12. ON ord.order_id = ord_items.order_id
13. where order_status = 'delivered'
14. group by cust.customer_state
15. order by 2 desc;
16.
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	customer_state	Total_ord_pr...	avg_ord_price	Total_freigh...	avg_freight_...
1	SP	5067633.16	109.1	702069.99	15.12
2	RJ	1759651.13	124.42	295750.44	20.91
3	MG	1552481.83	120.2	266409.84	20.63
4	RS	728897.47	118.83	132575.32	21.61
5	PR	666063.51	117.91	115645.29	20.47

5. Analysis on sales, freight and delivery time

1. Calculate days between purchasing, delivering and estimated delivery

```
1.  select
2.    date_diff(DATE(ord.order_delivered_customer_date), DATE(ord.order_purchase_timestamp),
     DAY) as Days_to_delivery,
3.    date_diff(DATE(ord.order_estimated_delivery_date), DATE(ord.order_delivered_customer_d
    ate), DAY) as Delivery_delta
4.  -- Delivery_delta, measure of delay/pre delivery,
5.  -- + value Days -> Item delivered before date, vise versa.
6.  FROM
7.    `sql-case-study-360620.case_study.orders` ord
8.    where ord.order_delivered_customer_date is not NULL;
9.
```

We find **days_to_delivery** which is the actual days to delivery, and **Delivery Delta** which is a measure to check if the delivery happened before Delivery date or not.

Basically, **Delivery Delta** (`estimated_delivery_date - order_delivered_customer_date`), which is + ve when Delivery happened before expected Date and vice-versa.

2. Create columns:

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

Above code calculated the similar metrices.

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

Code:

```
1.  -- Group data by state, take mean of freight_value,
2.  -- time_to_delivery, diff_estimated_delivery
3.
4.  select
5.    cust.customer_state,
6.    ROUND(AVG(ord_items.freight_value), 2) as avg_freight_value,
7.    ROUND(AVG(date_diff(DATE(ord.order_delivered_customer_date), DATE(ord.order_purchase_t
    imestamp), DAY))) as time_to_delivery,
8.    ROUND(AVG(date_diff(DATE(ord.order_estimated_delivery_date), DATE(ord.order_delivered_
    customer_date), DAY))) as diff_estimated_delivery
9.  from `sql-case-study-360620.case_study.customers` cust
10. join `sql-case-study-360620.case_study.orders` ord
11. ON cust.customer_id = ord.customer_id
12. join `sql-case-study-360620.case_study.order_items` ord_items
13. ON ord.order_id = ord_items.order_id
14. where order_status = 'delivered'
15. group by cust.customer_state
16. order by 2 desc;
17.
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	customer_state	avg_freight_...	time_to_deli...	diff_estimat...
22	SC	21.51	15.0	12.0
23	DF	21.07	13.0	12.0
24	RJ	20.91	15.0	12.0
25	MG	20.63	12.0	13.0
26	PR	20.47	12.0	13.0
27	SP	15.12	9.0	11.0

3. Sort the data to get the following:

    1. Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5
    2. Top 5 states with highest/lowest average time to delivery
    3. Top 5 states where delivery is really fast/ not so fast compared to estimated date

States w.r.t highest and lowest freight average freight value:

```
1.   -- 1. Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5
2.  select
3.    cust.customer_state,
4.    ROUND(AVG(ord_items.freight_value), 2) as avg_freight_value
5.  from `sql-case-study-360620.case_study.customers` cust
6.  join `sql-case-study-360620.case_study.orders` ord
7.  ON cust.customer_id = ord.customer_id
8.  join `sql-case-study-360620.case_study.order_items` ord_items
9.  ON ord.order_id = ord_items.order_id
10. where order_status = 'delivered'
11. group by cust.customer_state
12. order by 2 desc     -- change ASC, sort in ASC order.
13. limit 5;
14.
```

Query results

JOB INFORMATION | RESULTS | JSON

Row	customer_state	avg_freight_...
1	PB	43.09
2	RR	43.09
3	RO	41.33
4	AC	40.05
5	PI	39.12

Query results

JOB INFORMATION | RESULTS | JSON

Row	customer_state	avg_freight_...
1	SP	15.12
2	PR	20.47
3	MG	20.63
4	RJ	20.91
5	DF	21.07

States w.r.t highest and lowest freight average time to delivery :

```
1.  -- Top 5 states with highest/lowest average time to delivery
2.  select
3.    cust.customer_state,
4.    ROUND(AVG(date_diff(DATE(ord.order_delivered_customer_date), DATE(ord.order_purchase_t
    imestamp), DAY))) as time_to_delivery
5.  from `sql-case-study-360620.case_study.customers` cust
6.  join `sql-case-study-360620.case_study.orders` ord
7.  ON cust.customer_id = ord.customer_id
8.  where order_status = 'delivered'
9.  group by cust.customer_state
10. order by 2 desc       -- change ASC, sort in ASC order.
11. limit 5;
```

Query results

JOB INFORMATION | RESULTS | JSON

Row	customer_state	time_to_deli...
1	RR	29.0
2	AP	27.0
3	AM	26.0
4	AL	25.0
5	PA	24.0

Query results

JOB INFORMATION | RESULTS | JSON

Row	customer_state	time_to_deli...
1	SP	9.0
2	MG	12.0
3	PR	12.0
4	DF	13.0
5	RS	15.0

States w.r.t faster delivery or slow/delayed delivery of orders :

Here if the orders are delivered before estimated delivery date, we can say that state has faster delivery or vice-versa.

```
1.  -- Top 5 states where delivery is really fast/ not so fast compared to estimated date
2.  select
3.    cust.customer_state,
4.    ROUND(AVG(date_diff(DATE(ord.order_estimated_delivery_date), DATE(ord.order_delivered_
    customer_date), DAY))) as diff_estimated_delivery
5.    -- diff_estimated_delivery, + ve value date diff -> order del. before estimated date -
    > faster delivery or vice-versa.
6.  from `sql-case-study-360620.case_study.customers` cust
7.  join `sql-case-study-360620.case_study.orders` ord
8.  ON cust.customer_id = ord.customer_id
9.  where order_status = 'delivered'
10. group by cust.customer_state
11. order by 2 desc       -- change ASC, sort in ASC order.
12. limit 5;
13.
```

States with faster delivery:

Query results

JOB INFORMATION | RESULTS | JSON | EXECU

Row	customer_state	diff_estimated_deli...
1	AC	21.0
2	AM	20.0
3	RO	20.0
4	AP	20.0
5	RR	17.0

It states that for State, AC on average orders is delivered 21 days before the estimated delivery Date, thus very fast delivery.

States with slow delivery:

Query results

JOB INFORMATION | RESULTS | JSON

Row	customer_state	diff_estimat...
1	AL	9.0
2	ES	10.0
3	MA	10.0
4	SE	10.0
5	SP	11.0

6. **Payment type analysis:**

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

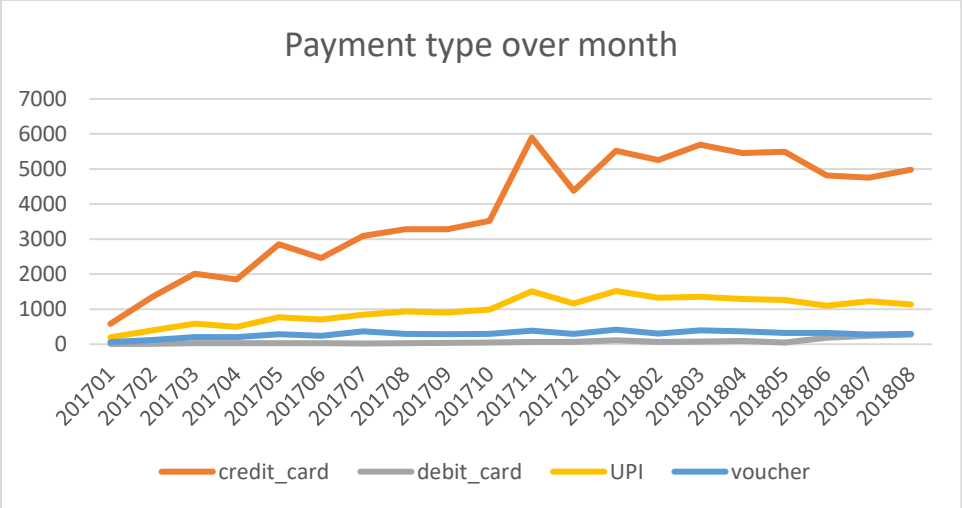
```
1.  -- Month over Month count of orders for different payment types
2.
3.  select
4.    FORMAT_DATE('%Y%m',order_purchase_timestamp) as MONTH,
5.    pay.payment_type, count(*) as Total_orders
6.  from `sql-case-study-360620.case_study.payments` pay
7.  join `sql-case-study-360620.case_study.orders` ord
8.  ON ord.order_id = pay.order_id
9.  group by MONTH, pay.payment_type
10. order by MONTH, Total_orders desc;
11.
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	MONTH	payment_type	Total_orders
51	201712	credit_card	4377
52	201712	UPI	1160
53	201712	voucher	294
54	201712	debit_card	64
55	201801	credit_card	5520

Analysing Data, Credit card is the preferred option for payment, followed by UPI.



2. Distribution of payment installments and count of orders:

```
1.  -- Distribution of payment installments and count of orders
2.
3.  select
4.    pay.payment_installments, count(*) as Total_orders
5.  from `sql-case-study-360620.case_study.payments` pay
6.  group by pay.payment_installments;
7.
```

Query results

JOB INFORMATION | RESULTS | JSON

Row	payment_installments	Total_orders
1	0	2
2	1	52546
3	2	12413
4	3	10461
5	4	7098
6	5	5239
7	6	3920