

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

Assumption:

Data Type of columns in a table can be viewed by following the steps given below.
Under Target_SQL_Business_Case Dataset → Click any table. I have clicked on orders table. The table schema contains details such as datatypes, column names/field names and Mode etc.

Insight:

The datatypes of customers table are string and integer.

The datatypes of geolocation table are integer, string and float.

The datatypes of order_items table are string, integer, timestamp and float.

The datatypes of order_reviews are string, integer and timestamp.

The datatypes of order table are string and timestamp.

The datatypes of payments table are string, integer and float.

The datatypes of products table are string and integer.

The datatypes of seller table are string and integer.

The screenshot displays a data management interface. On the left, an 'Explorer' sidebar shows a tree view of workspace resources. Under the 'Target_SQL_Business_Case' dataset, the 'customers' table is selected. The main panel shows the 'customers' table schema. At the top, there are tabs for 'SCHEMA', 'DETAILS', 'PREVIEW', and 'LINEAGE', with 'SCHEMA' being the active tab. Below the tabs, there are buttons for 'QUERY', 'SHARE', 'COPY', 'SNAPSHOT', 'DELETE', and 'EXPORT'. A 'Filter' section allows searching for properties. The schema table lists columns with checkboxes, field names, types, and modes.

	Field name	Type	Mode	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				

At the bottom of the schema view, there are two buttons: 'EDIT SCHEMA' and 'VIEW ROW ACCESS POLICIES'.

The screenshot shows the Databricks Explorer interface. On the left, the 'Target_SQL_Business_Case' database is expanded, showing various tables including 'orders'. The 'orders' table is selected, and its schema is displayed on the right. The schema table lists columns such as 'order_id', 'customer_id', 'order_status', 'order_purchase_timestamp', 'order_approved_at', 'order_delivered_carrier_date', 'order_delivered_customer_date', and 'order_estimated_delivery_date'.

Field name	Type	Mode	Collation	Default Value	Policy Tags	Description
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				

1.2 Time period for which the data is given.

Assumption: order_purchase_timestamp column is used to find the first order and the last order as it more granular and updated more frequently.

The screenshot shows the Databricks Explorer interface with a SQL query executed. The query is: `SELECT MIN(order_purchase_timestamp) AS Start_date, MAX(order_purchase_timestamp) AS End_date FROM `Target_SQL_Business_Case.orders`;`. The results are displayed in a table with two columns: 'Start_date' and 'End_date'. The 'Start_date' is '2016-09-04 21:15:19 UTC' and the 'End_date' is '2018-10-17 17:30:18 UTC'.

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

Query:
 SELECT
 MIN(order_purchase_timestamp) AS Start_date,
 MAX(order_purchase_timestamp) AS End_date
 FROM
 `Target_SQL_Business_Case.orders`;

Insight: The earliest order timestamp from the given data is captured by the query as 2016-09-04 21:15:19 UTC and the last order timestamp is captured by the query as 2018-10-17 17:30:18 UTC.

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

Assumption: Customer city and state can be found using the customer table and by joining the customer table with orders table we can get the data for customers who placed orders during the given period.

Query:

```
SELECT
  DISTINCT c.customer_city,
  c.customer_state
FROM
  `Target_SQL_Business_Case.customers` AS c
JOIN
  `Target_SQL_Business_Case.orders` AS o
ON
  c.customer_id=o.customer_id;
```

The screenshot shows a SQL IDE interface. On the left is an 'Explorer' pane with a search bar and a tree view of workspace resources. The tree view shows a project named 'scaler-dsml-sql-381001' containing several tables: 'customers', 'geolocation', 'order_items', 'order_reviews', 'orders', 'payments', 'products', 'sellers', 'farmers_market', and 'hr'. The 'Target_SQL_Business_Case' folder is expanded, showing the 'customers' table. The main editor pane displays a SQL query in a text area, with a 'RUN' button and other options like 'SAVE', 'SHARE', 'SCHEDULE', and 'MORE'. Below the query editor is a 'Query results' section with tabs for 'JOB INFORMATION', 'RESULTS', 'JSON', 'EXECUTION DETAILS', and 'EXECUTION GRAPH'. The 'RESULTS' tab is active, showing a table with 10 rows of data. The table has two columns: 'customer_city' and 'customer_state'.

Row	customer_city	customer_state
1	rio de janeiro	RJ
2	sao leopoldo	RS
3	general salgado	SP
4	brasilia	DF
5	paranavai	PR
6	cuiaba	MT
7	sao luis	MA
8	maceio	AL
9	hortolandia	SP
10	varzea grande	MT

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?

Assumption: Number of orders placed or number of products purchased from orders table is analyzed find out about the growing trend on e-commerce in Brazil. To

extract this information order_purchase_timestamp column is used to extract individual time attributes such as year, month, day etc.

Query:

```
SELECT
  EXTRACT(year
FROM
  order_purchase_timestamp) AS year,
  EXTRACT(month
FROM
  order_purchase_timestamp) AS month,
  COUNT(DISTINCT order_id) AS total_orders,
FROM
  `Target_SQL_Business_Case.orders`
WHERE
  order_status = 'delivered'
GROUP BY
  year, month
ORDER BY
  year, month
```

The screenshot displays a data analytics platform interface. On the left, a sidebar shows a tree view of workspace resources under 'Target_SQL_Business_Case', including tables like customers, geolocation, order_items, order_reviews, orders, payments, products, sellers, farmers_market, and hr. The main area shows a SQL query editor with the following query:

```
1 SELECT
2   EXTRACT(year
3 FROM
4   order_purchase_timestamp) AS year,
5   EXTRACT(month
6 FROM
7   order_purchase_timestamp) AS month.
```

Below the query editor, the 'Query results' section is visible, showing a table with columns: Row, year, month, and total_orders. The table contains 10 rows of data, showing a growing trend in total orders over time.

Row	year	month	total_orders
1	2016	9	1
2	2016	10	265
3	2016	12	1
4	2017	1	750
5	2017	2	1653
6	2017	3	2546
7	2017	4	2303
8	2017	5	3546
9	2017	6	3135
10	2017	7	3872

Insight: Yes, there is a growing trend on e-commerce in Brazil. This is understood by analyzing the result generated from the above query. During specific months there is

a peak where customers have purchased more but in other months customers are inclined to purchase products online than before.

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

Assumption: Number of orders placed or number of products purchased from orders table is analyzed find out about the growing trend on e-commerce in Brazil. To extract this information order_purchase_timestamp column is used to extract individual time attributes such as year, month, hour etc. Dawn is assumed between 00:00H to 06:00H, Morning is assumed between 06:00H to 12:00H, Afternoon is assumed between 12:00H to 18:00H and the remaining period is considered as Night.

Query:

```
SELECT SUM(case when hour between 0 and 6 then orders else 0 end) as dawn,  
SUM(case when hour between 7 and 12 then orders else 0 end) as morning,  
SUM(case when hour between 13 and 18 then orders else 0 end) as afternoon,  
SUM(case when hour between 19 and 23 then orders else 0 end) as night  
from  
(SELECT extract(hour from order_purchase_timestamp) as hour,  
count (distinct order_id) as orders  
from `Target_SQL_Business_Case.orders`  
group by hour  
order by hour)
```

The screenshot shows a SQL query editor 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 with line numbers 1 through 10. Below the query is a 'Query results' section with tabs for 'JOB INFORMATION', 'RESULTS', 'JSON', 'EXECUTION DETAILS', and 'EXECUTION GRAPH'. The 'RESULTS' tab is active, showing a table with 5 columns: 'dawn', 'morning', 'afternoon', 'night', and an empty column. The first row of data shows values 5242, 27733, 38135, and 28331.

Row	dawn	morning	afternoon	night	
1	5242	27733	38135	28331	

Insight: Brazilian customers tend to buy more during afternoons.

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

3.1 Get month on month orders by states

Query:

SELECT

```
EXTRACT(Year FROM order_purchase_timestamp) AS year,  
EXTRACT(Month FROM order_purchase_timestamp) AS month,  
c.customer_state AS state,  
COUNT(distinct order_id) AS orders
```

FROM `Target_SQL_Business_Case.orders` AS o

JOIN `Target_SQL_Business_Case.customers` AS c

ON o.customer_id=c.customer_id

GROUP BY year, month, state

ORDER BY year, month, state;

The screenshot shows a data analytics interface with a sidebar on the left containing a tree view of workspace resources. The main area displays a SQL query and its results. The query is a SELECT statement that extracts the year and month from the order_purchase_timestamp, joins the orders table with the customers table, and counts distinct order IDs grouped by year, month, and state. The results are shown in a table with columns for Row, year, month, state, and orders. The table contains 13 rows of data for the year 2016, showing the number of orders for each month and state combination.

Row	year	month	state	orders
1	2016	9	RR	1
2	2016	9	RS	1
3	2016	9	SP	2
4	2016	10	AL	2
5	2016	10	BA	4
6	2016	10	CE	8
7	2016	10	DF	6
8	2016	10	ES	4
9	2016	10	GO	9
10	2016	10	MA	4
11	2016	10	MG	40
12	2016	10	MT	3
13	2016	10	PA	4

3.2 Distribution of customers across the states in Brazil

Query:

SELECT

customer_state AS state,

COUNT(DISTINCT customer_id) AS customers

FROM

`Target_SQL_Business_Case.customers`

GROUP BY

state

ORDER BY
customers DESC;

The screenshot shows a data analytics interface. On the left is the 'Explorer' panel with a search bar and a tree view of workspace resources. The main panel displays a SQL query and its results.

Explorer Panel:

- Search bar: Type to search
- Viewing workspace resources. [SHOW STARRED ONLY](#)
- scaler-dsml-sql-381001
 - External connections
 - Saved queries (5)
 - Target_SQL_Business_Case
 - customers
 - geolocation
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellers
 - [SHOW MORE](#)
 - farmers_market

SQL Query:

```
1 SELECT
2   customer_state AS state,
3   COUNT(DISTINCT customer_id) AS customers
4 FROM
5   `Target_SQL_Business_Case.customers`
6 GROUP BY
7   state
```

Query results:

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EX
Row	state	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			

Insights:

The query gives the distribution of customers across states in Brazil. The highest number of customers are 41,746 in Sao Paulo, southeast state of Brazil and the least number of customers are 46 in Roraima, northern most state of Brazil.

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.

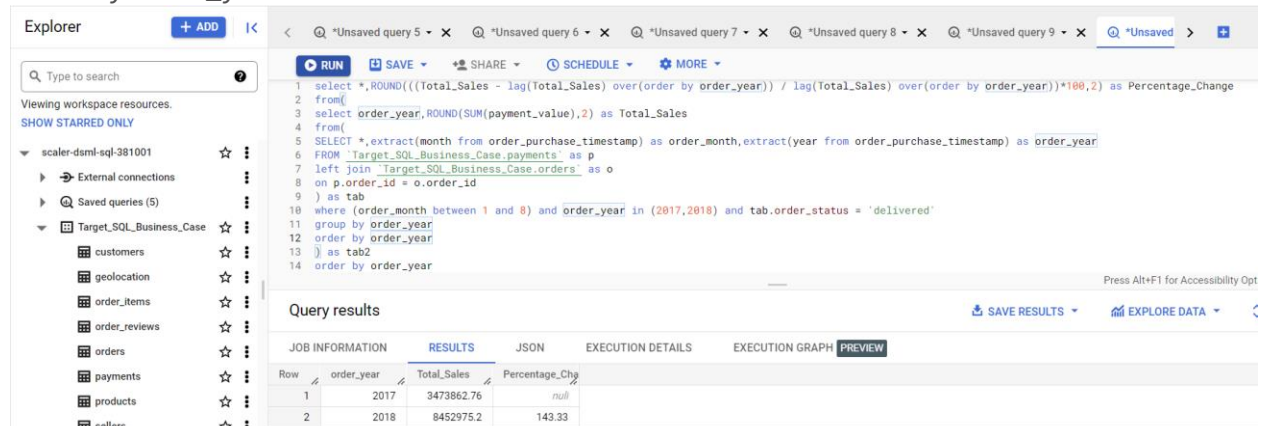
Query:

```
select *,ROUND((((Total_Sales - lag(Total_Sales) over(order by order_year)) /
lag(Total_Sales) over(order by order_year))*100,2) as Percentage_Change
from(
select order_year,ROUND(SUM(payment_value),2) as Total_Sales
from(
SELECT *,extract(month from order_purchase_timestamp) as
order_month,extract(year from order_purchase_timestamp) as order_year
FROM `Target_SQL_Business_Case.payments` as p
left join `Target_SQL_Business_Case.orders` as o
```

```

on p.order_id = o.order_id
) as tab
where (order_month between 1 and 8) and order_year in (2017,2018) and
tab.order_status = 'delivered'
group by order_year
order by order_year
) as tab2
order by order_year

```



The screenshot shows a SQL IDE interface. On the left is an Explorer pane with a search bar and a tree view of workspace resources. The main editor displays a SQL query. Below the editor, the 'Query results' section is active, showing a table with 4 columns: Row, order_year, Total_Sales, and Percentage_Change. The results show two rows for the years 2017 and 2018.

Row	order_year	Total_Sales	Percentage_Change
1	2017	3473862.76	null
2	2018	8452975.2	143.33

Insight: Total sales percentage increase of 143 is seen from 2017 to 2018.

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

Query:

```

SELECT
  c.customer_state AS state,
  AVG(o_i.price) AS avg_price,
  SUM(o_i.price) AS sum_price,
  AVG(o_i.freight_value) AS avg_freight_value,
  SUM(o_i.freight_value) AS sum_freight_value
FROM
  `Target_SQL_Business_Case.customers` as c
inner JOIN
  `Target_SQL_Business_Case.orders` as o
ON
  c.customer_id = o.customer_id
INNER JOIN
  `Target_SQL_Business_Case.order_items` as o_i
ON o.order_id = o_i.order_id
WHERE
  o.order_status = 'delivered'
GROUP BY

```


state
ORDER BY
sum_price DESC;

Query results

Row	state	avg_price	sum_price	avg_freight_val	sum_freight_val
1	SP	109.103366...	5067633.16...	15.1151823...	702069.989...
2	RJ	124.418520...	1759651.12...	20.9114360...	295750.440...
3	MG	120.198345...	1552481.82...	20.6263425...	266409.840...
4	RS	118.829062...	728897.470...	21.6131920...	132575.320...
5	PR	117.908215...	666063.510...	20.4718162...	115645.290...
6	SC	123.752045...	507012.130...	21.5073590...	88115.6500...
7	BA	134.016872...	493584.140...	26.4875563...	97553.6699...
8	DF	125.901660...	296498.409...	21.0721613...	49624.9399...
9	GO	124.214624...	282836.699...	22.5628678...	51375.6499...
10	ES	120.738629...	268643.449...	22.0289797...	49014.4799...
11	PE	144.266603...	251889.489...	32.6933333...	57082.5599...
12	CE	154.107559...	219757.379...	32.7344950...	46679.3899...
13	PA	165.531869...	174470.589...	35.6290132...	37552.9800...

Insight: Sao Paulo is the best performing state because of less freight value and price.

5. Analysis on sales, freight and delivery time

5.1 Calculate days between purchasing, delivering and estimated delivery.

Query:

```
SELECT
  order_id,
  DATE_DIFF(date_trunc(order_delivered_customer_date, DAY),
    date_trunc(order_purchase_timestamp, DAY), DAY) AS days_to_delivery,
  DATE_DIFF(date_trunc(order_estimated_delivery_date, DAY),
    date_trunc(order_delivered_customer_date, DAY), DAY) AS estimated_delivery_diff
FROM
  `Target_SQL_Business_Case.orders`;
```

Explorer + ADD IK

Q *Unsaved query 5 - X Q *Unsaved query 6 - X Q *Unsaved query 7 - X Q *Unsaved query 8 - X Q *Unsaved query 9 - X +

RUN SAVE SHARE SCHEDULE MORE

1 SELECT
2 order_id,
3 DATE_DIFF(date_trunc(order_delivered_customer_date, DAY), date_trunc(order_purchase_timestamp, DAY), DAY) AS days_to_delivery,
4 DATE_DIFF(date_trunc(order_estimated_delivery_date, DAY), date_trunc(order_delivered_customer_date, DAY), DAY) AS estimated_delivery_diff

Query results SAVE RESULTS EXPLORE DATA

Row	order_id	days_to_delivery	estimated_delivery_diff
1	1950d777989f6a877539f5379...	30	-12
2	2c45c33d2f9cb8ff8b1c86cc28...	31	29
3	65d1e226dfaeb8cdc42f66542...	36	17
4	635c894d068ac37e6e03dc54e...	31	2
5	3b97562c3aee8bdedcb5c2e45...	33	1
6	68f47f50f04c4cb6774570cfde...	30	2
7	276e9ec344d3bf029ff83a161c...	44	-4
8	54e1a3c2b97fb0809da548a59...	41	-4
9	fd04fa4105ee8045f6a0139ca5...	37	-1
10	302bb8109d097a9fcb6e9cfc5...	34	-5
11	66057d37308e787052a32828...	39	-6
12	19135c945c554eebfdf7576c73...	36	-2

Results per page: 50 1 - 50 of 99441

5.2 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

Query:

SELECT

AVG(DATE_DIFF(date_trunc(order_delivered_customer_date, DAY), date_trunc(order_purchase_timestamp, DAY), DAY)) AS avg_time_to_delivery,
AVG(DATE_DIFF(date_trunc(order_estimated_delivery_date, DAY), date_trunc(order_delivered_customer_date, DAY), DAY)) AS avg_diff_estimated_delivery

FROM

`Target_SQL_Business_Case.orders`

WHERE

order_status = 'delivered';

< Q *Unsaved query 5 - X Q *Unsaved query 6 - X Q *Unsaved query 7 - X Q *Unsaved query 8 - X Q *Unsaved query 9 - X Q *Unsaved > +

RUN SAVE SHARE SCHEDULE MORE This query will process 3.3 MB when run

1 SELECT
2 AVG(DATE_DIFF(date_trunc(order_delivered_customer_date, DAY), date_trunc(order_purchase_timestamp, DAY), DAY)) AS avg_time_to_delivery,
3 AVG(DATE_DIFF(date_trunc(order_estimated_delivery_date, DAY), date_trunc(order_delivered_customer_date, DAY), DAY)) AS avg_diff_estimated_delivery
4 FROM
5 `Target_SQL_Business_Case.orders`
6 WHERE
7 order_status = 'delivered';

Query results SAVE RESULTS EXPLORE DATA

Row	avg_time_to_delivery	avg_diff_estimated_delivery
1	12.4968487...	11.8758888...

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

Query:

```
SELECT
c.customer_state as state,
AVG(o_i.freight_value) AS avg_freight_value,
AVG(DATE_DIFF(date_trunc(o.order_delivered_customer_date, DAY),
date_trunc(o.order_purchase_timestamp, DAY), DAY)) AS avg_time_to_delivery,
AVG(DATE_DIFF(date_trunc(o.order_estimated_delivery_date, DAY),
date_trunc(o.order_delivered_customer_date, DAY), DAY)) AS
avg_diff_estimated_delivery
FROM
`Target_SQL_Business_Case.order_items` as o_i
join
`Target_SQL_Business_Case.orders` as o
on o.order_id=o_i.order_id
join `Target_SQL_Business_Case.customers` as c
on o.customer_id=c.customer_id
GROUP BY
state;
```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row	state	avg_freight_value	avg_time_to_delivery	avg_diff_estimated_delivery		
1	MT	28.1662843601896	17.907425265188...	14.571841851494709		
2	MA	38.25700242718446	21.5899999999999...	9.9062499999999929		
3	AL	35.843671171171152	24.447306791569...	8.73536299765808		
4	SP	15.147275390419248	8.66225265379071	11.207910772344571		
5	MG	20.630166806306541	11.920724626461...	13.342649221955588		
6	PE	32.917862679955796	18.224513172966...	13.450171821305863		
7	RJ	20.96092393168248	15.074791460483...	12.014774494556768		
8	DF	21.041354945968383	12.893842887473...	12.200424628450103		
9	RS	21.735804330392945	15.134518180335...	14.1341920756563		
10	SE	36.653168831168855	21.4186666666666...	10.002666666666677		
11	PR	20.531651567944248	11.893078420959...	13.486103735174341		

Insights: Roraima has the highest freight payment cost.

Recommendations: Freight value can be reduced for customers to purchase more online.

5.4 Sort the data to get the following:

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

Query:

```
SELECT
  c.customer_state as state ,
  AVG(o_i.freight_value) AS avg_freight_value
FROM
  `Target_SQL_Business_Case.order_items` as o_i
join `Target_SQL_Business_Case.orders` as o
on o_i.order_id=o.order_id
join `Target_SQL_Business_Case.customers` as c
on o.customer_id=c.customer_id
GROUP BY
  state
ORDER BY
  avg_freight_value desc
LIMIT
  5;
```

Explorer

+ ADD

<

Type to search

?

Viewing workspace resources.

SHOW STARRED ONLY

▼ scaler-dsml-sql-381001

▶ External connections

▶ Saved queries (5)

▼ Target_SQL_Business_Case

customers

geolocation

order_items

order_reviews

orders

payments

products

sellers

SHOW MORE

▶ farmers_market

▶ hr

Query 5

✕

🔍 *Unsaved query 6

✕

🔍 *Unsaved query 7

✕

🔍 *

▶ RUN

📄 SAVE

👤 SHARE

🕒 SCHEDULE

⚙️ MORE

```

1 SELECT
2   c.customer_state as state ,
3   AVG(o.i.freight_value) AS avg_freight_value
4 FROM
5   `Target_SQL_Business_Case.order_items` as o_i
6   join `Target_SQL_Business_Case.orders` as o
7   on o_i.order_id=o.order_id
8   join `Target_SQL_Business_Case.customers` as c
9   on o.customer_id=c.customer_id
10 GROUP BY
11   state
12 ORDER BY
13   avg_freight_value desc
14 LIMIT
15   5;

```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	state	avg_freight_value		
1	RR	42.984423076923072		
2	PB	42.723803986710969		
3	RO	41.069712230215814		
4	AC	40.073369565217362		
5	PI	39.147970479704838		

Recommendations: Freight value of RR and PB can be reduced for customers to purchase more online.

Query:

SELECT

```
c.customer_state as state ,
AVG(o_i.freight_value) AS avg_freight_value
```

FROM

```
`Target_SQL_Business_Case.order_items` as o_i
join `Target_SQL_Business_Case.orders` as o
on o_i.order_id=o.order_id
join `Target_SQL_Business_Case.customers` as c
on o.customer_id=c.customer_id
```

GROUP BY

state

ORDER BY

avg_freight_value ASC

LIMIT

5;

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 and its results.

Explorer Pane:

- Search: Type to search
- Viewing workspace resources. [SHOW STARRED ONLY](#)
- scaler-dsml-sql-381001
 - External connections
 - Saved queries (5)
 - Target_SQL_Business_Case
 - customers
 - geolocation
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellers
 - [SHOW MORE](#)
 - farmers_market
 - hr

SQL Query:

```

1 SELECT
2   c.customer_state as state ,
3   AVG(o.i.freight_value) AS avg_freight_value
4 FROM
5   `Target_SQL_Business_Case.order_items` as o_i
6   join `Target_SQL_Business_Case.orders` as o
7   on o_i.order_id=o.order_id
8   join `Target_SQL_Business_Case.customers` as c
9   on o.customer_id=c.customer_id
10 GROUP BY
11   state
12 ORDER BY
13   avg_freight_value ASC
14 LIMIT
15   5;

```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
Row	state	avg_freight_valu			
1	SP	15.1472753...			
2	PR	20.5316515...			
3	MG	20.6301668...			
4	RJ	20.9609239...			
5	DF	21.0413549...			

Insights: State codes of SP and PR are best states to do online shopping since they less freight value.

5.5 Top 5 states with highest/lowest average time to delivery

Query:

SELECT

```

c.customer_state as state ,
AVG(DATE_DIFF(date_trunc(o.order_delivered_customer_date, DAY) ,
date_trunc(o.order_purchase_timestamp, DAY), DAY)) AS avg_time_to_delivery,
FROM

```

```

`Target_SQL_Business_Case.order_items` as o_i
join `Target_SQL_Business_Case.orders` as o
on o_i.order_id=o.order_id
join `Target_SQL_Business_Case.customers` as c
on o.customer_id=c.customer_id

```

GROUP BY

state

ORDER BY

avg_time_to_delivery DESC

LIMIT

5;

Explorer + ADD IK

Q Type to search

Viewing workspace resources.
[SHOW STARRED ONLY](#)

- ▼ scaler-dsml-sql-381001
 - ▶ External connections
 - ▶ Saved queries (5)
 - ▼ Target_SQL_Business_Case
 - customers
 - geolocation
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellers
 - [SHOW MORE](#)
 - ▶ farmers_market
 - ▶ hr

Q *Unsaved query 6 X Q *Unsaved query 7 X Q *Unsaved query 8 X Q *Unsaved query 9 X Q *Unsaved query 10 X

RUN SAVE SHARE SCHEDULE MORE

```

1 SELECT
2   c.customer_state as state ,
3   AVG(DATE_DIFF(date_trunc(o.order_delivered_customer_date, DAY) , date_trunc(o.order_purchase_timestamp, DAY), DAY)) AS avg_time_to_delivery,
4 FROM
5   `Target_SQL_Business_Case.order_items` as o_i
6   join `Target_SQL_Business_Case.orders` as o
7   on o_i.order_id=o.order_id
8   join `Target_SQL_Business_Case.customers` as c
9   on o.customer_id=c.customer_id
10 GROUP BY
11   state
12 ORDER BY
13   avg_time_to_delivery DESC
14 LIMIT
15   5;

```

Press Alt+F1 for Accessibility

Query results SAVE RESULTS EXPLORE DATA

JOB INFORMATION RESULTS JSON EXECUTION DETAILS EXECUTION GRAPH PREVIEW

Row	state	avg_time_to_delivery
1	AP	28.22222222222218
2	RR	28.173913043478258
3	AM	26.337423312883427
4	AL	24.447306791569098
5	PA	23.702087286527469

Recommendations: Delivery time of AP and RR can be reduced for customers to purchase more online.

Query:

SELECT

c.customer_state as state ,
 AVG(DATE_DIFF(date_trunc(o.order_delivered_customer_date, DAY) ,
 date_trunc(o.order_purchase_timestamp, DAY), DAY)) AS avg_time_to_delivery,
 FROM

`Target_SQL_Business_Case.order_items` as o_i
 join `Target_SQL_Business_Case.orders` as o
 on o_i.order_id=o.order_id
 join `Target_SQL_Business_Case.customers` as c
 on o.customer_id=c.customer_id

GROUP BY

state

ORDER BY

avg_time_to_delivery ASC

LIMIT

5;

The screenshot shows a data analytics tool interface. On the left is an 'Explorer' pane with a search bar and a tree view of workspace resources. The tree view includes 'External connections', 'Saved queries (5)', and a folder 'Target_SQL_Business_Case' containing tables like 'customers', 'geolocation', 'order_items', 'order_reviews', 'orders', 'payments', 'products', 'sellers', 'farmers_market', and 'hr'. The main pane displays a SQL query in a text editor with a toolbar above it containing 'RUN', 'SAVE', 'SHARE', 'SCHEDULE', and 'MORE'. Below the editor, the 'Query results' section is active, showing a table with 5 rows and 3 columns: 'Row', 'state', and 'avg_time_to_delivery'. The results are as follows:

Row	state	avg_time_to_delivery
1	SP	8.66225265...
2	PR	11.8930784...
3	MG	11.9207246...
4	DF	12.8938428...
5	SC	14.9502196...

Insights: State codes of SP and PR are best states to do online shopping since they less delivery time.

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

SELECT

c.customer_state as state ,
 AVG(DATE_DIFF(date_trunc(o.order_estimated_delivery_date, DAY),
 date_trunc(o.order_delivered_customer_date, DAY), DAY)) AS
 avg_diff_estimated_delivery

FROM

`Target_SQL_Business_Case.order_items` as o_i
 join `Target_SQL_Business_Case.orders` as o
 on o_i.order_id=o.order_id
 join `Target_SQL_Business_Case.customers` as c
 on o.customer_id=c.customer_id

GROUP BY

state

ORDER BY

avg_diff_estimated_delivery DESC

LIMIT

5;

Explorer

+ ADD

⌵

Type to search

?

Viewing workspace resources.

SHOW STARRED ONLY

▼ scaler-dsml-sql-381001

External connections

Saved queries (5)

Target_SQL_Business_Case

customers

geolocation

order_items

order_reviews

orders

payments

products

sellors

SHOW MORE

farmers_market

hr

⌵

query 5

×

⌵

"Unsaved query 6"

×

⌵

"Unsaved query 7"

×

⌵

"Unsaved query 8"

×

⌵

"Unsaved query 9"

×

⌵

"Unsaved query 10"

×

⌵

>

+

⬮ RUN

⌵ SAVE

⬮ SHARE

⌵ SCHEDULE

⬮ MORE

```

1 SELECT
2   c.customer_state as state ,
3   AVG(DATE_DIFF(date_trunc(o.order_estimated_delivery_date, DAY), date_trunc(o.order_delivered_customer_date, DAY))) AS avg_diff_estimated_delivery
4 FROM
5   `Target_SQL_Business_Case.order_items` as o_i
6   join `Target_SQL_Business_Case.orders` as o
7   on o_i.order_id=o.order_id
8   join `Target_SQL_Business_Case.customers` as c
9   on o.customer_id=c.customer_id
10  GROUP BY
11    state
12  ORDER BY
13    avg_diff_estimated_delivery DESC
14  LIMIT
15    5;

```

Press Alt+F1 for Accessibility Op

Query results

⬮ SAVE RESULTS

⬮ EXPLORE DATA

JOB INFORMATION

RESULTS

JSON

EXECUTION DETAILS

EXECUTION GRAPH

PREVIEW

Row	state	avg_diff_estimated_delivery
1	AC	20.978021978021971
2	RO	20.040293040293058
3	AM	19.932515337423315
4	AP	18.395061728395063
5	RR	18.326086956521742

Query:

SELECT

c.customer_state as state ,
 AVG(DATE_DIFF(date_trunc(o.order_estimated_delivery_date, DAY),
 date_trunc(o.order_delivered_customer_date, DAY), DAY)) AS
 avg_diff_estimated_delivery

FROM

`Target_SQL_Business_Case.order_items` as o_i
 join `Target_SQL_Business_Case.orders` as o
 on o_i.order_id=o.order_id
 join `Target_SQL_Business_Case.customers` as c
 on o.customer_id=c.customer_id

GROUP BY

state

ORDER BY

avg_diff_estimated_delivery ASC

LIMIT

5;

Explorer + ADD IK

Q Type to search

Viewing workspace resources.
[SHOW STARRED ONLY](#)

- ▼ scaler-dsml-sql-381001
 - External connections
 - Saved queries (5)
 - Target_SQL_Business_Case
 - customers
 - geolocation
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellers
 - [SHOW MORE](#)
 - farmers_market
 - hr

1 SELECT
2 c.customer_state as state ,
3 |AVG(DATE_DIFF(date_trunc(o.order_estimated_delivery_date, DAY), date_trunc(o.order_delivered_customer_date, DAY))) AS avg_diff_estimated_delivery
4 FROM
5 |`Target_SQL_Business_Case.order_items` as o_i
6 |join `Target_SQL_Business_Case.orders` as o
7 |on o_i.order_id=o.order_id
8 |join `Target_SQL_Business_Case.customers` as c
9 |on o.customer_id=c.customer_id
10 GROUP BY
11 state
12 ORDER BY
13 avg_diff_estimated_delivery ASC
14 LIMIT
15 5;

Query results SAVE RESULTS EXPLORE DATA

Row	state	avg_diff_estimated_delivery
1	AL	8.73536299765808
2	MA	9.90624999999999929
3	SE	10.0026666666666677
4	ES	10.646292134831446
5	BA	10.98262286179745

6. Payment type analysis:

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

Query:

SELECT

extract(Year from order_purchase_timestamp) AS year,
extract(Month from order_purchase_timestamp) AS month,
p.payment_type,
COUNT(o.order_id) AS order_count

FROM

`Target_SQL_Business_Case.orders` as o
join

`Target_SQL_Business_Case.payments` as p
on o.order_id=p.order_id

GROUP BY

year,
month, payment_type

ORDER BY

year, month ASC;

Explorer + ADD <

Viewing workspace resources.

SHOW STARRED ONLY

- scaler-dsml-sql-381001
 - External connections
 - Saved queries (5)
 - Target_SQL_Business_Case
 - customers
 - geolocation
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellers
 - SHOW MORE
 - farmers_market
 - hr

Query results SAVE RESULTS EXI

JOB INFORMATION RESULTS JSON EXECUTION DETAILS EXECUTION GRAPH PREVIEW

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

Results per page: 50 1 - 50 of 90

Insight: Payments made by credit cards take the first place but there is slight downfall in credit card payment from April 2018.

Recommendation: Debit card and UPI payments can be promoted to generate more sales by providing ease of payment for customers.

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

Query:

```

SELECT
  p.payment_installments as installments,
  COUNT(*) AS order_count
FROM
  `Target_SQL_Business_Case.orders` as o
  join
  `Target_SQL_Business_Case.payments` as p
  on o.order_id=p.order_id
GROUP BY
  installments;

```

Explorer

+ ADD

IK

Type to search

?

Viewing workspace resources.

SHOW STARRED ONLY

▼ scaler-dsml-sql-381001

▶ External connections

▶ Saved queries (5)

▼ Target_SQL_Business_Case

customers

geolocation

order_items

order_reviews

orders

payments

products

sellers

SHOW MORE

▶ farmers_market

▶ hr

1 SELECT

2 p.payment_installments as installments,

3 COUNT(*) as order_count

Query results

SAVE RESULTS

E

JOB INFORMATION

RESULTS

JSON

EXECUTION DETAILS

EXECUTION GRAPH

PREVIEW

Row	installments	order_count
1	1	52546
2	7	1626
3	10	5328
4	6	3920
5	2	12413
6	4	7098
7	3	10461
8	8	4268
9	9	644
10	5	5239
11	12	133
12	20	17
13	15	74

Results per page: 50 1 - 24 of 24

Insight: Instalments with single payment is popular among customers.