International E-commerce Analytics 101

Analyst: Shuaib O.B
Email: shuaibosaze75@gmail.com
LinkedIn: Shuaib O.B
Date of Completion: 02-04-2022

Table of Contents

Platform Setup
- Installing the required Library
- Importing Packages
- Google Colab User Authentication for Drive, Cloud Storage
- GCP Integration: BigQuery - Colab
- Querying the BigQuery E-commerce Database

Section 1: Exploratory Analysis
- Query 1.1 : Information Schema
- Query 1.2 : Calculating Duration and Time Range of Orders
- Query 1.3 : Unique Order Statuses

Section 2: In-depth Exploration
- Query 2.1 : Order Value Trend
- Query 2.2 : Analyzing Order Size

Section 3: Evolution of E-commerce
- Query 3.1 : Month-on-Month Orders Placed By State Analysis
- Query 3.2 : Customers Distribution across all the States

Section 4: Impact on Economy
- Query 4.1 : Percentage Increase in Order Costs from 2017 to 2018 (January to August)
- Query 4.2 : Analyzing the Monos movement of E-Commerce w.r.t order prices, Freight, and Others

Section 5: Analysis based on Sales, Freight and Delivery time.
- Query 5.1 : Analyzing Order Delivery Time and Estimation Accuracy
- Query 5.2: Highest Avg Freight Value
- Query 5.3b: Lowest Avg Freight Value
- Query 5.3a: Highest Avg Delivery Time
- Query 5.4b: Lowest Avg Delivery Time
- Query 5.4: Identifying Top 5 States with Fastest Over-Estimated Order Deliveries

Section 6: Analysis based on Payments
- Query 6.1 : Month-on-Month Analysis of Orders Placed by Payment Types.
- Query 6.2: Analysis of Orders Based on Payment Installments Paid.

Section 7: RFM Analysis.
- Query 7.1 : RFM Score.
- Query 7.2 : Customer Segmentation.

Section 8: Strategizing the Opportunity For Market Capture
- Recommendations

Introduction

Target is a globally renowned brand, has expanded its operations to Brazil, offering a unique shopping experience to its customers. With an extensive dataset covering 100,000 orders placed between 2016 and 2018, there is an opportunity to gain valuable insights into Target's operations in Brazil. This analysis aims to provide actionable recommendations to enhance order processing, pricing strategies, payment set shipping efficiency, and overall customer satisfaction.

Objective

The primary objective of this analysis is to provide Target with actionable insights derived from the dataset, enabling informed decision-making to enhance operational efficiency, customer satisfaction, and overall business performance in the Brazilian market.

Tech Stack

- KDBMS BigQuery for storing and querying large datasets
- Colab Notebook for executing Google Cloud SDK commands in a Linux-based cloud environment.
- Data Manipulation Pandas for dataframe operations.
- Data Visualization
  - Excel Chart
  - Tableau Chart

Installing the required Library

```
# Kick start matplotlib_sheetsediting
!
# Apply install graphviz
!
```

Importing Packages

```
# [sort] pandas to gsf
!rm -r google-cloud-sdk/ Signature
!rm -r google-cloud-sdk/ Original
!apt-get install graphviz -qq
```

Google Colab User Authentication for Drive, Cloud Storage

This code authenticates a user in Google Colab, enabling access to Google services like Drive, Cloud Storage, and other resources.

```
# [sort] google-colab [sort] auth
auth.authenticate_user()
!gcloud init --console-only
!
```

GCP Integration : BigQuery - Colab

In Colab, users can take advantage of the **google-cloud-bigq_table** package to display large pandas dataframes interactively. Additionally, accessing the BigQuery database is made possible by using the `%%bigquery [[sort]]` magic method within the Google Cloud Platform environment. This enables users to retrieve results through BigQuery SQL queries directly from Colab notebook.

```
# [sort] google [sort] data_table
```

Querying the BigQuery Ecomerce Database

Section 1: Exploratory Analysis

Query 1.1 : Information Schema

INFORMATION_SCHEMA provides metadata about database objects and enables a-frame validation. It helps in understanding the database structure, particularly with notionally linked tables using primary and foreign keys. It also provides insights into each column, including its data type, "is_nullable", characteristics, and more.

```
# Setting up BigQuery Client.
!bq --project_id=target-ecommerce-analysis-339547 \
!
# Query to see all the data stored in the database
BASH \
bq query --nolog "data_schema, table_name, column_name, ordinal_position, is_nullable, data_type
FROM \`target-ecommerce-analysis-339547.INFORMATION_SCHEMA.COLUMNS\`
"
# Storing the query
query_job = client.query(sql)
results = query_job.result()
df = results.to_dataframe()
df.to_csv("/content/drive/MyDrive/Target/Target_Ecommerce_Analysis/Data_Analysis/Data_Analysis/Table_Schema.csv")
# Setting table to the dataframe
table = dataframe_to_table(df)
# Display the dataframe
table
```

index	table_catalog	table_schema	table_name	column_name	ordinal_position	is_nullable	data_type
0	sixteenth-420915	target	customers	customer_id	1	YES	STRING
1	sixteenth-420915	target	customers	customer_unique_id	2	YES	STRING
2	sixteenth-420915	target	customers	customer_zip_code_prefix	3	YES	STRING
3	sixteenth-420915	target	customers	customer_city	4	YES	STRING
4	sixteenth-420915	target	customers	customer_state	5	YES	STRING
5	sixteenth-420915	target	geolocation	geolocation_zip_code_prefix	1	YES	STRING
6	sixteenth-420915	target	geolocation	geolocation_lat	2	YES	FLOAT64
7	sixteenth-420915	target	geolocation	geolocation_lng	3	YES	FLOAT64
8	sixteenth-420915	target	geolocation	geolocation_city	4	YES	STRING
9	sixteenth-420915	target	geolocation	geolocation_state	5	YES	STRING
10	sixteenth-420915	target	order_items	order_id	1	YES	STRING
11	sixteenth-420915	target	order_items	order_item_id	2	YES	INT64
12	sixteenth-420915	target	order_items	product_id	3	YES	STRING
13	sixteenth-420915	target	order_items	seller_id	4	YES	STRING
14	sixteenth-420915	target	order_items	shipping_limit_date	5	YES	TIMESTAMP
15	sixteenth-420915	target	order_items	price	6	YES	FLOAT64
16	sixteenth-420915	target	order_items	freight_value	7	YES	FLOAT64
17	sixteenth-420915	target	order_payments	order_id	1	YES	STRING
18	sixteenth-420915	target	order_payments	sequential_id	2	YES	INT64
19	sixteenth-420915	target	order_payments	payment_type	3	YES	STRING
20	sixteenth-420915	target	order_payments	payment_installments	4	YES	INT64
21	sixteenth-420915	target	order_payments	payment_value	5	YES	FLOAT64

Rows: 25 Next rows Generate code with sql Show recommended plots

BigQuery Schema Visualization Script

An **Entity-Relationship Diagram (ERD)** visually represents entities (objects), attributes (properties), and relationships in a database, aiding in understanding data organization and connections between entities.

This code generates an Entity-Relationship Diagram (ERD) from a DataFrame representing table metadata. It groups the DataFrame by table name, creates nodes for tables and columns, and connects primary keys to foreign keys. The `graphviz` library (`Digraph` object) is used to visualize the relationships between tables and their columns. A data table is also created for lookup during relationship analysis.

```
# [ Using the DataFrame to table_name and aggregate column names and data types
df_grouped = df.groupby('table_name')['column_name', 'data_type'].apply(lambda x: [tuple(x[['column_name', '
# Initialize a Digraph object with improved styling
dot = Digraph(node_attr={'shape': 'record', 'fontsize': '12', 'fontname': 'Helvetica'},
              graph_attr={'rankdir': 'LR', 'splines': 'ispline'})
# Add nodes for each table and its columns
for table_name, columns in df_grouped.iteritems():
    label = f'{table_name}|{}'.format(*[f'{col_name}: {data_type}' for col_name, data_type in columns])
    dot.node(table_name, label=label)
    print(f"Added node for {table_name} with {len(columns)} columns")
# Add edges for relationships
for _, row in df.iterrows():
    if row['ordinal_position'] == 1:
        dot.edge(row['table_name'], f'{row[\'column_name\']}:e', dir='both')
# Render the graph to a file
dot.render('database_schema', format='png', cleanup=True)
# Display the rendered image
from IPIL import Image, Image
```


Query 1.2 : Calculating Duration and Time Range of Orders

This SQL code calculates the range of order times and duration between the earliest and latest orders, providing insights for business scheduling, inventory management, and advertising optimization based on order patterns.

```
# [sort]
!bq --project_id=target-ecommerce-analysis-339547 \
BASH \
bq query --nolog "SELECT
    MAX(TIMESTAMP(order_purchase_timestamp)) AS MostRecentTime,
    MIN(TIMESTAMP(order_purchase_timestamp)) AS MostRecentTime,
    TIME(order_purchase_timestamp) AS MostRecentTime
FROM \`target.orders\`
"
BASH \
    MostRecentTime AS EarliestOrderTime,
    MostRecentTime AS LatestOrderTime,
    DATEDIFF(MAX MostRecentTime, MostRecentTime) AS DurationDays
FROM \`target.orders\`
"
df = client.query(query).to_dataframe()
df
```

index	EarliestOrderTime	LatestOrderTime	DurationDays
0	00:00:00	23:59:59	0

Rows: 1 Generate code with df Show recommended plots

The hour part was observed as 23 hours (within a range of 23 hours 59 minutes), prompting an investigation into whether this result was influenced by outliers.

Analyzing Order Timing Distribution

Investigating the distribution pattern of order placement with respect to the **hours of the day** to check the possibility of outliers.

```
# [sort]
!bq --project_id=target-ecommerce-analysis-339547 \
BASH \
bq query --nolog "SELECT
    EXTRACT(HOUR FROM TIMESTAMP(order_purchase_timestamp)) AS order_hour,
    COUNT(DISTINCT order_purchase_timestamp) AS MostRecentTime,
    ROUND(COUNT(DISTINCT order_purchase_timestamp) / SUM(COUNT(DISTINCT order_purchase_timestamp)) OVER () * 100, 2) AS percentage_of_days
FROM \`target.orders\`
GROUP BY order_hour
ORDER BY order_hour;
"
df = client.query(query).to_dataframe()
df
```

index	order_hour	MostRecentTime	percentage_of_days
0	0	1	0.1
1	1	1	0.1
2	2	1	0.1
3	3	1	0.1
4	4	1	0.1
5	5	1	0.1
6	6	1	0.1
7	7	1	0.1
8	8	1	0.1
9	9	1	0.1
10	10	1	0.1
11	11	1	0.1
12	12	1	0.1
13	13	1	0.1
14	14	1	0.1
15	15	1	0.1
16	16	1	0.1
17	17	1	0.1
18	18	1	0.1
19	19	1	0.1
20	20	1	0.1
21	21	1	0.1
22	22	1	0.1
23	23	1	0.1

Rows: 24 Generate code with df Show recommended plots

The analysis results show that **seventy-eight percent** of the days within the dataset fall into the **23 hour range**. This indicates that it was not an outlier instead, **the majority of orders** approved for almost 23+ hours or 697 days.
Thus, it is conclusive to answer **The orders were placed** between 12 AM to 11:59 PM (23-Plus hours).

Query 1.3 : Unique Order Statuses

This query provides a list of distinct order statuses present in the dataset.

```
# [sort]
!bq --project_id=target-ecommerce-analysis-339547 \
BASH \
bq query --nolog "SELECT DISTINCT order_status
FROM \`target.orders\`
"
df = client.query(query).to_dataframe()
df
```

index	order_status
0	created
1	shipped
2	canceled
3	unavailable
4	invoiced
5	processing
6	delivered
7	approved

Rows: 8 Generate code with df Show recommended plots

This output provides insights into shipping-run that conditions.
It is underscored that we need to include order statuses such as "created", "processing", and "invoiced" as order statuses.

Customer City and State Counts Excluding insignificant Order Statuses:

```
# [sort]
!bq --project_id=target-ecommerce-analysis-339547 \
BASH \
bq query --nolog "SELECT
    COUNTDISTINCT customers.city) AS City_Count,
    COUNTDISTINCT customers.state) AS State_Count
FROM \`target.customers\` AS
JOIN \`target.orders\` AS
    ON customers.customer_id = orders.customer_id
WHERE
    order_status NOT IN ('canceled', 'processing', 'unavailable')
"
df = client.query(query).to_dataframe()
df
```

index	City_Count	State_Count
0	4119	27

Rows: 1 Generate code with df Show recommended plots

By applying this filter, we obtain 4119 unique cities across 27 states.

Section 2: In-depth Exploration

Exploring Order Trends

- **Yearly Order Growth:** Is there a trend of increasing orders over the years?
- **Monthly Seasonality:** Can we observe monthly patterns in order volumes?
- **Time of Day Ordering Habits:** When do Brazilian customers typically place orders?
  - Dawn (0-6 hrs)
  - Morning (7-12 hrs)
  - Afternoon (13-18 hrs)
  - Night (19-23 hrs)

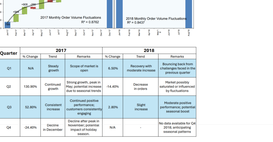
Query 2.1 : Order Value Trend

```
# [sort]
!bq --project_id=target-ecommerce-analysis-339547 \
BASH \
bq query --nolog "WITH
    order_purchase_month AS (SELECT order_purchase_timestamp AS order_year,
        EXTRACTYEAR FROM o.order_purchase_timestamp) AS order_month,
        SUMCASE WHEN o.order_status = 'canceled' THEN 1 ELSE 0 END) AS
        COUNTDISTINCT o.order_id) AS order_count,
        ROUND(
        SUMprice) AS revenue,
        ROUND(
        SUMprice) / COUNTDISTINCT o.order_id), 2) AS avg_order_value,
        ROUND(
        SUMprice) / COUNTDISTINCT o.order_id) - LAG(SUMprice) / COUNTDISTINCT o.order_id) OVER (ORDER BY order_year, order_month), 0) * 100, 2) AS percent_revenue_change,
        ROUND(
        COUNTDISTINCT o.order_id) - LAG(COUNTDISTINCT o.order_id) OVER (ORDER BY order_year, order_month), 0) * 100, 2) AS percent_num_orders_change
FROM
    order_purchase
    JOIN \`target.orders\` AS
    ON order_purchase.order_id = orders.order_id
GROUP BY
    order_year, order_month
ORDER BY
    order_year, order_month;
"
df = client.query(query).to_dataframe()
df
```

index	order_year	order_month	revenue	percent_revenue_change	num_orders	percent_num_orders_change
0	2017	1	138488.04	-	800	-
1	2017	2	291908.01	108.85	1780	122.5
2	2017	3	450480.94	54.35	2682	50.67
3	2017	4	379799.56	-15.7	2404	-10.37
4	2017	5	608296.29	60.17	3700	53.9
5	2017	6	499093.27	-17.96	3245	-12.3
6	2017	7	548811.36	9.97	3965	22.19
7	2017	8	548908.70	0.02	4193	5.75
8	2017	9	513371.17	-6.48	4331	3.29
9	2017	10	549774.71	7.1	4728	9.16
10	2017	11	1065748.29	93.83	7544	59.57
11	2017	12	843987.12	-20.79	5673	-24.79
12	2018	1	1157141.59	36.93	7269	28.14
13	2018	2	1092179.12	-5.61	6728	-7.45
14	2018	3	1159652.04	6.18	7211	7.18
15	2018	4	1123495.68	-3.12	6939	-3.77
16	2018	5	1013078.75	-9.83	6873	-0.95
17	2018	6	992547.94	-2.03	6167	-10.27
18	2018	7	1077261.06	8.53	6292	2.03
19	2018	8	1048314.17	-2.69	6512	3.5
20	2018	9	954.44	-99.91	16	-99.75

Rows: 21 Generate code with df Show recommended plots

Fig: Waterfall Chart for Order Volume Trend Analysis



Quarter	2017			2018		
	% Change	Trend	Remarks	% Change	Trend	Remarks
Q1	N/A	Steady growth	Scope of market is open	6.30%	Steady growth	Recovery with moderate increase
Q2	156.90%	Continued growth	Strong growth, peak in May, potential increase from April's seasonal trends	14.4%	Decrease in orders	Recovering backs from previous quarter's decline but lower than previous quarter
Q3	52.40%	Consistent increase	Continued positive momentum, customers consistently shopping	1.6%	Slight increase	Moderate positive performance, potential seasonal boost
Q4	-28.40%	Decline in December	Decline after peak in November, potential impact of holiday season	N/A	-	No data available for Q4 2018, extrapolating based on seasonal patterns

Inference of Result

January 2017 to December 2017:

The number of orders increased from January to November 2017 by a factor of 8 times.

There is a significant increase in November 2017 (up by 53.17%).

However, there is a sharp decrease in December 2017 (down by 24.52%).

January 2018 to August 2018:

The trend in January 2018 shows an increase compared to December 2017.

February 2018 shows a slight decrease (down by 7.29%).

March 2018 sees an increase again (up by 7.18%).

From April to August 2018, there's a gradual decrease in the number of orders.

Observations:

The data suggests a cyclic pattern with peaks in November 2017 and January 2018.

The months of December 2017 and September 2018 show significant drops in the number of orders.

Conclusion:

While there was growth in the number of orders during certain months, there were also periods of decline. It's important to consider external factors such as holidays, promotions, or marketing campaigns that may have influenced these trends.

Analysis of Order Times by Time Category

```
# [sort]
!bq --project_id=target-ecommerce-analysis-339547 \
BASH \
bq query --nolog "WITH
    SELECT
    CASE
    WHEN EXTRACTHOUR FROM order_purchase_timestamp) BETWEEN 0 AND 6 THEN 'Dawn'
    WHEN EXTRACTHOUR FROM order_purchase_timestamp) BETWEEN 7 AND 12 THEN 'Morning'
    WHEN EXTRACTHOUR FROM order_purchase_timestamp) BETWEEN 13 AND 18 THEN 'Afternoon'
    WHEN EXTRACTHOUR FROM order_purchase_timestamp) BETWEEN 19 AND 23 THEN 'Night'
    END AS TimeCategory,
    COUNT(*) AS
    FROM
    TimeCategory,
    COUNTDISTINCT DE.NumberOfOrders),
    ROUND(COUNT(*) / SUMCOUNT(*) OVER () * 100, 2) AS Percentage
FROM
    order_purchase_timestamp
GROUP BY TimeCategory;
ORDER BY TimeCategory;
"
df = client.query(query).to_dataframe()
df
```

index	TimeCategory	NumberOfOrders	Percentage
0	Afternoon	30142	30.1
1	Dawn	5943	5.93
2	Morning	27733	27.68
3	Night	36067	36.01

Rows: 4 Generate code with df Show recommended plots

For the Chart for Order Time Segmentation

```
# [sort]
!bq --project_id=target-ecommerce-analysis-339547 \
BASH \
bq query --nolog "WITH
    SELECT
    CASE
    WHEN EXTRACTHOUR FROM order_purchase_timestamp) BETWEEN 0 AND 6 THEN 'Dawn'
    WHEN EXTRACTHOUR FROM order_purchase_timestamp) BETWEEN 7 AND 12 THEN 'Morning'
    WHEN EXTRACTHOUR FROM order_purchase_timestamp) BETWEEN 13 AND 18 THEN 'Afternoon'
    WHEN EXTRACTHOUR FROM order_purchase_timestamp) BETWEEN 19 AND 23 THEN 'Night'
    END AS TimeCategory,
    COUNT(*) AS
    FROM
    TimeCategory,
    COUNTDISTINCT DE.NumberOfOrders),
    ROUND(COUNT(*) / SUMCOUNT(*) OVER () * 100, 2) AS Percentage
FROM
    order_purchase_timestamp
GROUP BY TimeCategory;
ORDER BY TimeCategory;
"
df = client.query(query).to_dataframe()
df
```

index	TimeCategory	NumberOfOrders	Percentage
0	Afternoon	30142	30.1
1	Dawn	5943	5.93
2	Morning	27733	27.68
3	Night	36067	36.01

Rows: 4 Generate code with df Show recommended plots

Query 2.2 : Analysing Order Time

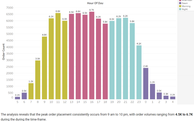
```
[ GO: query * ]
SELECT
    order_id IS 1
    ,CASE
        WHEN
        EXTRACTOR(YEAR order_purchase_timestamp) IS NULL
        THEN length(order)
        ELSE
        1
    END AS
    "Number_of_Orders"
    ,EXTRACTOR(HOUR FROM order_purchase_timestamp) AS hour_of_day
    FROM olist_orders
    GROUP BY hour_of_day
    ORDER BY hour_of_day
;
df = client.query(query).to_dataframe()
df
```

index	hour_of_day	order_count
0	0	2452
1	1	1548
2	2	978
3	3	679
4	4	569
5	5	1264
6	6	3773
7	7	6843
8	8	10843
9	9	12137
10	10	13659
11	11	13929
12	12	13777
13	13	13876
14	14	14284
15	15	14054
16	16	13381
17	17	13148
18	18	14403
19	19	14454
20	20	14090
21	21	13277
22	22	11162
23	23	8105

Rows: 24, Job complete

Next steps: Generate code with v7 | View recommended plans

Viz. Bar Chart for Hourly Order Volume Distribution



The analysis reveals that the peak order placement consistently occurs from 8 am to 16 pm, with order volumes ranging from **4.3K to 6.7K** during this during this time frame.

Inference: The distribution illustrates the variations in order placements throughout the day, providing insights into customer behaviour and potential peak times for various activities or transactions. The morning and Afternoon periods appear to be the busiest, with the Dawn and Night categories experiencing lower order volumes.

Section 3: Evolution of E-commerce

Monthly Orders Placed in Each State To understand the evolution of e-commerce orders in Brazil, we first examine the month-on-month number of orders placed in each state. This analysis provides insights into the ordering trends and potential variations across different regions.

Query 3.1 : Month-on-Month Orders Placed by State Analysis

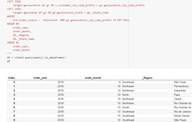
```
[ GO: query * ]
    WITH
    SELECT
        EXTRACTOR(YEAR order_purchase_timestamp) AS order_year,
        EXTRACTOR(MONTH order_purchase_timestamp) AS order_month,
        customer_state AS region,
        COUNT(DISTINCT order_id) AS Orderincustomers
    FROM
        olist_customers JOIN olist_orders USING(customer_id)
    WHERE
        order_purchase_timestamp IS NOT NULL
    GROUP BY
        1, 2, 3
    ORDER BY
        order_year ASC, order_month ASC, Orderincustomers DESC
    LIMIT 1000
    SELECT
        order_year,
        order_month,
        region,
        Orderincustomers,
        LAG(Orderincustomers, 1) OVER (PARTITION BY region ORDER BY order_year, order_month) AS Prev_Month_Orders,
        ROUND(
            SAFE_DIVIDE(
                (Orderincustomers - LAG(Orderincustomers, 1) OVER (PARTITION BY region ORDER BY order_year, order_month)),
                LAG(Orderincustomers, 1) OVER (PARTITION BY region ORDER BY order_year, order_month)
            ) * 100, 2
        ) AS Growth_Percentage
    FROM
        Month_Wise_Orders
    ORDER BY
        order_year ASC, order_month ASC, Orderincustomers DESC
;
df = client.query(query).to_dataframe()
df
```

index	order_year	order_month	region	Orderincustomers	Prev_Month_Orders	Growth_Percentage
0	2016	9	SE - Southeast	1	NULL	NULL
1	2016	10	SE - Southeast	258	1	25800.0
2	2016	12	SE - Northeast	1	258	-99.61
3	2016	12	SE - North	1	NULL	NULL
4	2016	12	SE - Southeast	4	1	300.0
5	2016	12	SE - Southeast	1	4	-75.0
6	2017	1	SE - Southeast	283	1	28200.0
7	2017	1	SE - Southeast	1	283	-99.65
8	2017	1	SE - Southeast	1	1	0.0
9	2017	1	SE - Southeast	1	1	0.0
10	2017	1	SE - Southeast	1	1	0.0
11	2017	1	SE - Southeast	1	1	0.0
12	2017	1	SE - Center West	1	1	0.0
13	2017	1	SE - Northeast	1	1	0.0
14	2017	1	SE - South	1	1	0.0
15	2017	1	SE - Northeast	1	1	0.0
16	2017	1	SE - Northeast	1	1	0.0
17	2017	1	SE - Southeast	1	1	0.0
18	2017	1	SE - Southeast	1	1	0.0
19	2017	1	SE - Southeast	1	1	0.0
20	2017	1	SE - Southeast	1	1	0.0
21	2017	1	SE - Southeast	1	1	0.0
22	2017	1	SE - Southeast	1	1	0.0
23	2017	1	SE - Southeast	1	1	0.0
24	2017	1	SE - Southeast	1	1	0.0

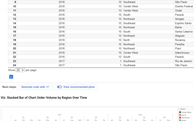
Rows: 25, Job complete

Next steps: Generate code with v7 | View recommended plans

Viz. Stacked Bar of Chart Order Volume by Region Over Time



Viz. Map Chart Order Volume across Brazilian States



Row	Region	OrderVolumeSum	OrderPercentage
1	Southeast	60738	68.73
2	South	15810	14.36
3	Northeast	9016	9.37
4	Center West	5492	5.56
5	North	1655	1.91

Inference

- **The South-East region** dominates with Sao Paulo, Rio de Janeiro, and Minas Gerais contributing to 48.7% of the order volume.
- **Followed by the South-region**, led by Rio Grande do Sul, Parana, and Santa Catarina, accounts for 14.35%.
- **Other regions**, including North-East, Center West & North, contribute proportionally, while Northern states constitute the remaining small market share.

Query 3.2 : Customers Distribution across all the States

Customer Distribution Across States Additionally, we explore how customers are distributed across all the states in Brazil. This analysis helps in understanding the geographical spread of customers and identifying potential areas for targeted marketing and expansion efforts.

By analyzing both the month-on-month orders and customer distribution, we can gain valuable insights into the e-commerce landscape in Brazil and make informed decisions to optimize business strategies.

```
[ GO: query * ]
    WITH CTE AS (
    SELECT
        c.customer_state,
        g.city_name,
        g.state_name,
        COUNT(DISTINCT customer_id) AS Number_of_customers,
        SUM(COUNT(DISTINCT customer_id)) OVER(ORDER BY COUNT(DISTINCT customer_id) DESC) AS Total_Customers_Till_RunningPercentage
    FROM
        olist_customers AS c
    LEFT JOIN
        olist_geolocation AS g ON c.customer_state = g.state_code
    GROUP BY
        1, 2, 3
    ORDER BY
        Number_of_customers DESC
    )
    SELECT
        *,
        SUM(CTE.Number_of_customers) OVER(ORDER BY CTE.Number_of_customers DESC) AS RunningTotal,
        round(SAFE_DIVIDE(CTE.Number_of_customers, (SELECT SUM(CTE.Number_of_customers) FROM CTE)),2)*100 AS RunningPercentage
    FROM
        CTE
    ORDER BY
        1, 2, 3, 4, 5, 6, 7
    LIMIT 1000
    FROM
        CTE Number_of_customers DESC;
df = client.query(query).to_dataframe()
df
```

index	customer_state	city_name	state_name	Number_of_customers	Total_Customers_Till_RunningPercentage	RunningTotal	RunningPercentage
0	AP	São Paulo	Southeast	67142	38441	67142	0.12
1	AP	Rio de Janeiro	Southeast	67142	38441	134284	0.24
2	AP	Belo Horizonte	Minas Gerais	67142	38441	201426	0.36
3	AP	Rio Grande do Sul	South	3495	38441	71989	0.13
4	AP	Curitiba	South	3495	38441	75484	0.13
5	AP	Santa Catarina	South	3495	38441	78979	0.14
6	AP	Goiânia	Northeast	3495	38441	82474	0.15
7	AP	Distrito Federal	Northeast	3495	38441	85969	0.15
8	AP	Espírito Santo	Southeast	3495	38441	89464	0.16
9	AP	Belém	Northeast	3495	38441	92959	0.16
10	AP	Fortaleza	Northeast	3495	38441	96454	0.17
11	AP	Manaus	Northeast	3495	38441	99949	0.18
12	AP	Rio Grande do Norte	Northeast	3495	38441	103444	0.18
13	AP	Teresina	Northeast	3495	38441	106939	0.19
14	AP	Maceió	Northeast	3495	38441	110434	0.19
15	AP	João Pessoa	Northeast	3495	38441	113929	0.20
16	AP	Aracaju	Northeast	3495	38441	117424	0.21
17	AP	Campo Grande	Center West	3495	38441	120919	0.21
18	AP	Cuiabá	Center West	3495	38441	124414	0.22
19	AP	Goiânia	Center West	3495	38441	127909	0.23
20	AP	Belém	Northeast	3495	38441	131404	0.23
21	AP	Fortaleza	Northeast	3495	38441	134899	0.24
22	AP	Manaus	Northeast	3495	38441	138394	0.24
23	AP	Rio Grande do Norte	Northeast	3495	38441	141889	0.25
24	AP	Teresina	Northeast	3495	38441	145384	0.25
25	AP	Maceió	Northeast	3495	38441	148879	0.26
26	AP	João Pessoa	Northeast	3495	38441	152374	0.26
27	AP	Aracaju	Northeast	3495	38441	155869	0.27
28	AP	Campo Grande	Center West	3495	38441	159364	0.28
29	AP	Cuiabá	Center West	3495	38441	162859	0.28
30	AP	Goiânia	Center West	3495	38441	166354	0.29
31	AP	Belém	Northeast	3495	38441	169849	0.29
32	AP	Fortaleza	Northeast	3495	38441	173344	0.30
33	AP	Manaus	Northeast	3495	38441	176839	0.30
34	AP	Rio Grande do Norte	Northeast	3495	38441	180334	0.31
35	AP	Teresina	Northeast	3495	38441	183829	0.32
36	AP	Maceió	Northeast	3495	38441	187324	0.32
37	AP	João Pessoa	Northeast	3495	38441	190819	0.33
38	AP	Aracaju	Northeast	3495	38441	194314	0.33
39	AP	Campo Grande	Center West	3495	38441	197809	0.34
40	AP	Cuiabá	Center West	3495	38441	201304	0.35

Rows: 41, Job complete

Next steps: Generate code with v7 | View recommended plans

Viz. 6\|28 Rate Visualized in Map Chart Order Volume across Brazilian States

Inference

The top 6 states in the **Southern and Southeastern** regions contribute around **80%** of the customer base, while the remaining **20%** comes from the other 21 states in the North and Midwest regions. This highlights the significance of focusing marketing and sales efforts on these key states to effectively reach their customer base.

Section 4: Impact on Economy

Analyze the money movement by e-commerce by looking at order prices, freight, and others.

- 1. Get the **% increase in the cost of orders** from year **2017 to 2018** (include months between Jan to Aug only).
  - Use the **'payment_value'** column in the **payments** table to get the cost of orders.

Query 4.1 : Percentage Increase in Order Costs from 2017 to 2018 (January to August)

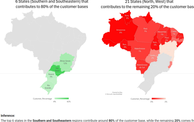
```
[ GO: query * ]
    WITH Percentageincrease AS (
        SELECT
            EXTRACTOR(YEAR o.order_purchase_ts) AS order_month,
            SUM(p.payment_value) AS total_order_cost,
            SAFECAST(SUM(CASE EXTRACTOR(YEAR o.order_purchase_ts) = 2017 THEN p.payment_value ELSE 0 END) AS total_order_cost_2017,
            SAFECAST(SUM(CASE EXTRACTOR(YEAR o.order_purchase_ts) = 2018 THEN p.payment_value ELSE 0 END) AS total_order_cost_2018
            FROM
            olist_orders o
            JOIN
            olist_payments AS p ON o.order_id = p.order_id
            GROUP BY
            1
            ORDER BY
            1
            SELECT
            order_month, total_order_cost, total_order_cost_2017, total_order_cost_2018,
            ROUND((total_order_cost_2018) - (total_order_cost_2017)) / (total_order_cost_2017) * 100, 1) AS percentage_difference
    FROM
    Percentageincrease
    GROUP BY
    1, 2, 3, 4, 5
    ORDER BY
    1
;
df = client.query(query).to_dataframe()
df
```

index	order_month	total_order_cost	total_order_cost_2017	total_order_cost_2018	percentage_difference
0	1	1123432.2	1123432.2	NULL	NULL
1	2	2218368.2	2218368.2	NULL	197.4
2	3	3543416.8	3543416.8	NULL	159.8
3	4	4728613.3	4728613.3	NULL	133.4
4	5	5953162.9	5953162.9	NULL	125.9
5	6	6984168.3	6984168.3	NULL	117.3
6	7	8067803.3	8067803.3	NULL	115.5
7	8	9737212.6	9737212.6	NULL	120.7

Rows: 8, Job complete

Next steps: Generate code with v7 | View recommended plans

Viz. Bar Chart 2017-2018 Order Value in Millions



Viz. 2017 2018 Grouped Bar Chart for Cost of Order Value in Millions

The cost of orders in January 2018 significantly increased by 197% compared to January 2017. This suggests a substantial spike in spending during this month.

The YOY percentage difference has decreased from 197% in 18% to 0-9months Span rapidly. Also suggests a significant reduction in the rate of growth. Market Saturation, or change in customer behaviour are some potential reasons for this decline.

Query 4.2 : Analysing the Money movement of E-commerce w.r.t order prices, Freight, and Others.

- a. Calculate the **Total & Average value of order price** for each state.
- b. Calculate the **Total & Average value of order freight** for each state.

```
[ GO: query * ]
    WITH
    SELECT
        c.customer_state,
        g.state_name,
        SUM(price) AS TotalPrice,
        ROUND(SAFECAST(AVG(price) AS INT64),2) AS AvgPrice,
        ROUND(SAFECAST(SUM(freight_value) AS INT64),2) AS TotalFreightValue,
        ROUND(SAFECAST(AVG(freight_value) AS INT64),2) AS AvgFreightValue
    FROM
        olist_order_items JOIN olist_orders USING(order_id)
        JOIN
        olist_customers AS c ON c.customer_id = o.customer_id
        LEFT JOIN
        olist_geolocation AS g ON c.customer_state = g.geolocation_city_code_prefix
    GROUP BY
        1, 2
    ORDER BY
        TotalPrice DESC, TotalFreightValue DESC
    LIMIT 1000
    SELECT
        customer_state, state_name, TotalPrice, TotalFreightValue, AvgPrice, AvgFreightValue,
        ROUND(SAFECAST(TotalPrice AS INT64) / SAFECAST(TotalFreightValue AS INT64), 2) AS Price_Freight_Ratio
    FROM
        Price_Freight
    ORDER BY
        Price_Freight_Ratio DESC
;
df = client.query(query).to_dataframe()
df
```

index	customer_state	state_name	TotalPrice	TotalFreightValue	AvgPrice	AvgFreightValue
0	SP	São Paulo	5571955.0	718843.0	113.0	15.0
1	RJ	Rio de Janeiro	2055690.0	305489.0	125.0	19.0
2	MG	Minas Gerais	1820820.0	270799.0	120.0	18.0
3	RS	Rio Grande do Sul	1078799.0	135832.0	119.0	15.0
4	PR	Parana	819160.0	101065.0	119.0	15.0
5	SC	Santa Catarina	619860.0	75696.0	121.0	15.0
6	BA	Bahia	611548.0	100156.0	134.0	22.0
7	GO	Goiás	338090.0	50360.0	132.0	20.0
8	ES	Espírito Santo	325967.0	49764.0	121.0	18.0
9	PE	Pernambuco	321744.0	59088.0	125.0	23.0
10	CE	Ceará	277437.0	48351.0	136.0	24.0
11	PB	Paraíba	238332.0	42723.0	154.0	27.0
12	MT	Mato Grosso	230128.0	35544.0	148.0	23.0
13	DF	Distrito Federal	225965.0	34693.0	144.0	22.0
14	MA	Maranhão	152523.0	38257.0	145.0	36.0
15	AL	Alagoas	150828.0	23494.0	151.0	23.0
16	SE	Sergipe	142776.0	23436.0	154.0	25.0
17	PI	Piauí	107086.0	32071.0	173.0	52.0
18	RN	Rio Grande do Norte	96475.0	19860.0	166.0	34.0
19	TO	Tocantins	49068.0	16628.0	177.0	60.0
20	AC	Acre	14693.0	3120.0	166.0	35.0
21	RR	Roraima	9064.0	2073.0	181.0	41.0

Rows: 27, Job complete

Next steps: Generate code with v7 | View recommended plans

Section 5: Analysis based on Sales, Freight and Delivery time.

1. First the no. of days taken to deliver each order from the order's purchase date as delivery time.
Also, calculate the difference (in days) between the estimated & actual delivery date of an order.
Do this on a single query. You can calculate the delivery time and the difference between the estimated & actual delivery date using the given formula:
Formula:
- time_to_delivery = order_delivered_customer_date - order_purchase_timestamp
- diff_estimated_delivery = order_delivered_customer_date - order_estimated_delivery_date

- Find top 5 **Cities with the highest average delivery time.**
- Find top 5 **Cities with the highest & lowest average delivery time.**
- Find top 5 **Cities where the order delivery is really fast** as compared to the estimated date of delivery.
You can see the difference between the averages of actual & estimated delivery date to figure out how fast the delivery was for each state.

Query 5.1 : Analyzing Order Delivery Time and Estimation Accuracy

```
WITH query AS (
    select *,
    order_purchase_timestamp,
    order_delivered_customer_date,
    order_estimated_delivery_date,
    DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS Actual_Delivery_Time,
    DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) AS Estimated_Delivery_Time,
    DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) AS Estimated_Delivery_Time_
    FROM
    (select *
    FROM
    (select *
    FROM `olist.orders`
    ) as orders_1
    ) as orders_2
    ) SELECT *
    FROM query
    ORDER BY
    order_estimated_delivery_date
    LIMIT 5;

Warning: Total number of rows (99278) exceeds max rows (5000). Falling back to pandas display.
```

	order_id	order_purchase_timestamp	order_delivered_customer_date	order_estimated_delivery_date	Actual_Delivery_Time	Estimated_Delivery_Time	Estimated_Delivery_Time_
0	cc285f7f7d7d4d4d4d0ee0ee7744	2017-05-17 15:08:07+00:00	2017-06-07 14:44:00+00:00	2017-06-20 00:00:00+00:00	20	43	43
1	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	2017-05-22 12:01:17+00:00	2017-06-22 12:14:00+00:00	2017-06-30 00:00:00+00:00	30	38	38
2	9d7a1d34a5052090a0a0a0a0a0a0a0a0	2017-05-23 10:47:44+00:00	2017-06-23 10:47:44+00:00	2017-06-30 00:00:00+00:00	30	37	37
3	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	2017-05-26 16:16:38+00:00	2017-06-05 16:16:38+00:00	2017-06-30 00:00:00+00:00	10	35	35
4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	2018-05-10 15:16:34+00:00	2018-05-22 15:16:34+00:00	2018-06-04 00:00:00+00:00	12	25	25

96475	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	2017-06-19 03:37:04+00:00	2017-06-26 21:57:04+00:00	2017-06-29 00:00:00+00:00	10	29	40
96476	9f1f5e7f5e7f5e7f5e7f5e7f5e7f5e7f	2017-06-20 11:18:44+00:00	2017-06-28 11:18:44+00:00	2017-06-29 00:00:00+00:00	8	29	40
96476	0a0b1b1f0a0b1b1f0a0b1b1f0a0b1b1f	2017-06-20 18:44:00+00:00	2017-06-27 18:44:00+00:00	2017-06-29 00:00:00+00:00	7	29	40
96477	085d5a091f2f085d5a091f2f085d5a09	2017-06-23 10:19:00+00:00	2017-06-30 10:19:00+00:00	2017-06-30 00:00:00+00:00	7	29	37
96478	1f5e6a7f1f5e6a7f1f5e6a7f1f5e6a7f	2018-05-10 10:48:00+00:00	2018-05-18 10:48:00+00:00	2018-06-04 00:00:00+00:00	8	25	25

Next steps: Generate code with c7 View recommended plots

Query 5.2a : Highest Avg Freight Value.

```
WITH query AS (
    select *,
    FROM
    (select *
    FROM
    (select *
    FROM `olist.orders`
    ) as orders_1
    ) as orders_2
    ) SELECT *
    FROM query
    ORDER BY
    order_estimated_delivery_date
    LIMIT 5;

Warning: Total number of rows (99278) exceeds max rows (5000). Falling back to pandas display.
```

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
0	cc285f7f7d7d4d4d4d0ee0ee7744	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	delivered	2017-05-17 15:08:07+00:00	2017-05-18 03:00:00+00:00	2017-05-19 08:00:00+00:00	2017-06-07 14:44:00+00:00	2017-06-20 00:00:00+00:00
1	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	9d7a1d34a5052090a0a0a0a0a0a0a0a0	delivered	2017-05-22 12:01:17+00:00	2017-05-22 12:15:10+00:00	2017-05-24 09:00:00+00:00	2017-06-22 12:14:00+00:00	2017-06-30 00:00:00+00:00
2	9d7a1d34a5052090a0a0a0a0a0a0a0a0	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	delivered	2017-05-23 10:47:44+00:00	2017-05-23 10:47:44+00:00	2017-05-24 09:00:00+00:00	2017-06-23 10:47:44+00:00	2017-06-30 00:00:00+00:00
3	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	delivered	2017-05-26 16:16:38+00:00	2017-05-26 16:30:38+00:00	2017-05-29 11:00:00+00:00	2017-06-05 16:16:38+00:00	2017-06-30 00:00:00+00:00
4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	delivered	2018-05-10 15:16:34+00:00	2018-05-10 15:35:31+00:00	2018-05-14 15:00:00+00:00	2018-05-22 15:16:34+00:00	2018-06-04 00:00:00+00:00

96475	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	2017-06-19 03:37:04+00:00	2017-06-26 21:57:04+00:00	2017-06-29 00:00:00+00:00	10	29	40
96476	9f1f5e7f5e7f5e7f5e7f5e7f5e7f5e7f	2017-06-20 11:18:44+00:00	2017-06-28 11:18:44+00:00	2017-06-29 00:00:00+00:00	8	29	40
96476	0a0b1b1f0a0b1b1f0a0b1b1f0a0b1b1f	2017-06-20 18:44:00+00:00	2017-06-27 18:44:00+00:00	2017-06-29 00:00:00+00:00	7	29	40
96477	085d5a091f2f085d5a091f2f085d5a09	2017-06-23 10:19:00+00:00	2017-06-30 10:19:00+00:00	2017-06-30 00:00:00+00:00	7	29	37
96478	1f5e6a7f1f5e6a7f1f5e6a7f1f5e6a7f	2018-05-10 10:48:00+00:00	2018-05-18 10:48:00+00:00	2018-06-04 00:00:00+00:00	8	25	25

Next steps: Generate code with c7 View recommended plots

Inference
The breakdown provides insights into the distribution of delivery.
The majority of deliveries are **Early Delivery** while **Late Delivery 1 On-time Delivery** make up a smaller proportions.
The Value is Percent category is relatively supporting a well-managed delivery process.

Query 5.2c : Highest Avg Delivery Time.

```
WITH query AS (
    select *,
    FROM
    (select *
    FROM
    (select *
    FROM `olist.orders`
    ) as orders_1
    ) as orders_2
    ) SELECT *
    FROM query
    ORDER BY
    order_estimated_delivery_date
    LIMIT 5;

Warning: Total number of rows (99278) exceeds max rows (5000). Falling back to pandas display.
```

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
0	cc285f7f7d7d4d4d4d0ee0ee7744	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	delivered	2017-05-17 15:08:07+00:00	2017-05-18 03:00:00+00:00	2017-05-19 08:00:00+00:00	2017-06-07 14:44:00+00:00	2017-06-20 00:00:00+00:00
1	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	9d7a1d34a5052090a0a0a0a0a0a0a0a0	delivered	2017-05-22 12:01:17+00:00	2017-05-22 12:15:10+00:00	2017-05-24 09:00:00+00:00	2017-06-22 12:14:00+00:00	2017-06-30 00:00:00+00:00
2	9d7a1d34a5052090a0a0a0a0a0a0a0a0	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	delivered	2017-05-23 10:47:44+00:00	2017-05-23 10:47:44+00:00	2017-05-24 09:00:00+00:00	2017-06-23 10:47:44+00:00	2017-06-30 00:00:00+00:00
3	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	delivered	2017-05-26 16:16:38+00:00	2017-05-26 16:30:38+00:00	2017-05-29 11:00:00+00:00	2017-06-05 16:16:38+00:00	2017-06-30 00:00:00+00:00
4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	delivered	2018-05-10 15:16:34+00:00	2018-05-10 15:35:31+00:00	2018-05-14 15:00:00+00:00	2018-05-22 15:16:34+00:00	2018-06-04 00:00:00+00:00

96475	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	2017-06-19 03:37:04+00:00	2017-06-26 21:57:04+00:00	2017-06-29 00:00:00+00:00	10	29	40
96476	9f1f5e7f5e7f5e7f5e7f5e7f5e7f5e7f	2017-06-20 11:18:44+00:00	2017-06-28 11:18:44+00:00	2017-06-29 00:00:00+00:00	8	29	40
96476	0a0b1b1f0a0b1b1f0a0b1b1f0a0b1b1f	2017-06-20 18:44:00+00:00	2017-06-27 18:44:00+00:00	2017-06-29 00:00:00+00:00	7	29	40
96477	085d5a091f2f085d5a091f2f085d5a09	2017-06-23 10:19:00+00:00	2017-06-30 10:19:00+00:00	2017-06-30 00:00:00+00:00	7	29	37
96478	1f5e6a7f1f5e6a7f1f5e6a7f1f5e6a7f	2018-05-10 10:48:00+00:00	2018-05-18 10:48:00+00:00	2018-06-04 00:00:00+00:00	8	25	25

Next steps: Generate code with c7 View recommended plots

Query 5.2b : Lowest Avg Freight Value.

```
WITH query AS (
    select *,
    FROM
    (select *
    FROM
    (select *
    FROM `olist.orders`
    ) as orders_1
    ) as orders_2
    ) SELECT *
    FROM query
    ORDER BY
    order_estimated_delivery_date
    LIMIT 5;

Warning: Total number of rows (99278) exceeds max rows (5000). Falling back to pandas display.
```

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
0	cc285f7f7d7d4d4d4d0ee0ee7744	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	delivered	2017-05-17 15:08:07+00:00	2017-05-18 03:00:00+00:00	2017-05-19 08:00:00+00:00	2017-06-07 14:44:00+00:00	2017-06-20 00:00:00+00:00
1	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	9d7a1d34a5052090a0a0a0a0a0a0a0a0	delivered	2017-05-22 12:01:17+00:00	2017-05-22 12:15:10+00:00	2017-05-24 09:00:00+00:00	2017-06-22 12:14:00+00:00	2017-06-30 00:00:00+00:00
2	9d7a1d34a5052090a0a0a0a0a0a0a0a0	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	delivered	2017-05-23 10:47:44+00:00	2017-05-23 10:47:44+00:00	2017-05-24 09:00:00+00:00	2017-06-23 10:47:44+00:00	2017-06-30 00:00:00+00:00
3	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	delivered	2017-05-26 16:16:38+00:00	2017-05-26 16:30:38+00:00	2017-05-29 11:00:00+00:00	2017-06-05 16:16:38+00:00	2017-06-30 00:00:00+00:00
4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	delivered	2018-05-10 15:16:34+00:00	2018-05-10 15:35:31+00:00	2018-05-14 15:00:00+00:00	2018-05-22 15:16:34+00:00	2018-06-04 00:00:00+00:00

96475	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	2017-06-19 03:37:04+00:00	2017-06-26 21:57:04+00:00	2017-06-29 00:00:00+00:00	10	29	40
96476	9f1f5e7f5e7f5e7f5e7f5e7f5e7f5e7f	2017-06-20 11:18:44+00:00	2017-06-28 11:18:44+00:00	2017-06-29 00:00:00+00:00	8	29	40
96476	0a0b1b1f0a0b1b1f0a0b1b1f0a0b1b1f	2017-06-20 18:44:00+00:00	2017-06-27 18:44:00+00:00	2017-06-29 00:00:00+00:00	7	29	40
96477	085d5a091f2f085d5a091f2f085d5a09	2017-06-23 10:19:00+00:00	2017-06-30 10:19:00+00:00	2017-06-30 00:00:00+00:00	7	29	37
96478	1f5e6a7f1f5e6a7f1f5e6a7f1f5e6a7f	2018-05-10 10:48:00+00:00	2018-05-18 10:48:00+00:00	2018-06-04 00:00:00+00:00	8	25	25

Next steps: Generate code with c7 View recommended plots

Query 5.3a : Highest Avg Delivery Time.

```
WITH query AS (
    select *,
    FROM
    (select *
    FROM
    (select *
    FROM `olist.orders`
    ) as orders_1
    ) as orders_2
    ) SELECT *
    FROM query
    ORDER BY
    order_estimated_delivery_date
    LIMIT 5;

Warning: Total number of rows (99278) exceeds max rows (5000). Falling back to pandas display.
```

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
0	cc285f7f7d7d4d4d4d0ee0ee7744	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	delivered	2017-05-17 15:08:07+00:00	2017-05-18 03:00:00+00:00	2017-05-19 08:00:00+00:00	2017-06-07 14:44:00+00:00	2017-06-20 00:00:00+00:00
1	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	9d7a1d34a5052090a0a0a0a0a0a0a0a0	delivered	2017-05-22 12:01:17+00:00	2017-05-22 12:15:10+00:00	2017-05-24 09:00:00+00:00	2017-06-22 12:14:00+00:00	2017-06-30 00:00:00+00:00
2	9d7a1d34a5052090a0a0a0a0a0a0a0a0	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	delivered	2017-05-23 10:47:44+00:00	2017-05-23 10:47:44+00:00	2017-05-24 09:00:00+00:00	2017-06-23 10:47:44+00:00	2017-06-30 00:00:00+00:00
3	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	delivered	2017-05-26 16:16:38+00:00	2017-05-26 16:30:38+00:00	2017-05-29 11:00:00+00:00	2017-06-05 16:16:38+00:00	2017-06-30 00:00:00+00:00
4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	delivered	2018-05-10 15:16:34+00:00	2018-05-10 15:35:31+00:00	2018-05-14 15:00:00+00:00	2018-05-22 15:16:34+00:00	2018-06-04 00:00:00+00:00

96475	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	2017-06-19 03:37:04+00:00	2017-06-26 21:57:04+00:00	2017-06-29 00:00:00+00:00	10	29	40
96476	9f1f5e7f5e7f5e7f5e7f5e7f5e7f5e7f	2017-06-20 11:18:44+00:00	2017-06-28 11:18:44+00:00	2017-06-29 00:00:00+00:00	8	29	40
96476	0a0b1b1f0a0b1b1f0a0b1b1f0a0b1b1f	2017-06-20 18:44:00+00:00	2017-06-27 18:44:00+00:00	2017-06-29 00:00:00+00:00	7	29	40
96477	085d5a091f2f085d5a091f2f085d5a09	2017-06-23 10:19:00+00:00	2017-06-30 10:19:00+00:00	2017-06-30 00:00:00+00:00	7	29	37
96478	1f5e6a7f1f5e6a7f1f5e6a7f1f5e6a7f	2018-05-10 10:48:00+00:00	2018-05-18 10:48:00+00:00	2018-06-04 00:00:00+00:00	8	25	25

Next steps: Generate code with c7 View recommended plots

Query 5.3b : Lowest Avg Delivery Time.

```
WITH query AS (
    select *,
    FROM
    (select *
    FROM
    (select *
    FROM `olist.orders`
    ) as orders_1
    ) as orders_2
    ) SELECT *
    FROM query
    ORDER BY
    order_estimated_delivery_date
    LIMIT 5;

Warning: Total number of rows (99278) exceeds max rows (5000). Falling back to pandas display.
```

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
0	cc285f7f7d7d4d4d4d0ee0ee7744	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	delivered	2017-05-17 15:08:07+00:00	2017-05-18 03:00:00+00:00	2017-05-19 08:00:00+00:00	2017-06-07 14:44:00+00:00	2017-06-20 00:00:00+00:00
1	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	9d7a1d34a5052090a0a0a0a0a0a0a0a0	delivered	2017-05-22 12:01:17+00:00	2017-05-22 12:15:10+00:00	2017-05-24 09:00:00+00:00	2017-06-22 12:14:00+00:00	2017-06-30 00:00:00+00:00
2	9d7a1d34a5052090a0a0a0a0a0a0a0a0	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	delivered	2017-05-23 10:47:44+00:00	2017-05-23 10:47:44+00:00	2017-05-24 09:00:00+00:00	2017-06-23 10:47:44+00:00	2017-06-30 00:00:00+00:00
3	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	delivered	2017-05-26 16:16:38+00:00	2017-05-26 16:30:38+00:00	2017-05-29 11:00:00+00:00	2017-06-05 16:16:38+00:00	2017-06-30 00:00:00+00:00
4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	delivered	2018-05-10 15:16:34+00:00	2018-05-10 15:35:31+00:00	2018-05-14 15:00:00+00:00	2018-05-22 15:16:34+00:00	2018-06-04 00:00:00+00:00

96475	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	2017-06-19 03:37:04+00:00	2017-06-26 21:57:04+00:00	2017-06-29 00:00:00+00:00	10	29	40
96476	9f1f5e7f5e7f5e7f5e7f5e7f5e7f5e7f	2017-06-20 11:18:44+00:00	2017-06-28 11:18:44+00:00	2017-06-29 00:00:00+00:00	8	29	40
96476	0a0b1b1f0a0b1b1f0a0b1b1f0a0b1b1f	2017-06-20 18:44:00+00:00	2017-06-27 18:44:00+00:00	2017-06-29 00:00:00+00:00	7	29	40
96477	085d5a091f2f085d5a091f2f085d5a09	2017-06-23 10:19:00+00:00	2017-06-30 10:19:00+00:00	2017-06-30 00:00:00+00:00	7	29	37
96478	1f5e6a7f1f5e6a7f1f5e6a7f1f5e6a7f	2018-05-10 10:48:00+00:00	2018-05-18 10:48:00+00:00	2018-06-04 00:00:00+00:00	8	25	25

Next steps: Generate code with c7 View recommended plots

Query 5.4 : Identifying Top 5 States with Faster-than-Estimated Order Deliveries

```
WITH query AS (
    select *,
    FROM
    (select *
    FROM
    (select *
    FROM `olist.orders`
    ) as orders_1
    ) as orders_2
    ) SELECT *
    FROM query
    ORDER BY
    order_estimated_delivery_date
    LIMIT 5;

Warning: Total number of rows (99278) exceeds max rows (5000). Falling back to pandas display.
```

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
0	cc285f7f7d7d4d4d4d0ee0ee7744	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	delivered	2017-05-17 15:08:07+00:00	2017-05-18 03:00:00+00:00	2017-05-19 08:00:00+00:00	2017-06-07 14:44:00+00:00	2017-06-20 00:00:00+00:00
1	3d4e7ae13f41c5ee4ee4ee4ee1a01b1c	9d7a1d34a5052090a0a0a0a0a0a0a0a0	delivered	2017-05-22 12:01:17+00:00	2017-05-22 12:15:10+00:00	2017-05-24 09:00:00+00:00	2017-06-22 12:14:00+00:00	2017-06-30 00:00:00+00:00
2	9d7a1d34a5052090a0a0a0a0a0a0a0a0	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	delivered	2017-05-23 10:47:44+00:00	2017-05-23 10:47:44+00:00	2017-05-24 09:00:00+00:00	2017-06-23 10:47:44+00:00	2017-06-30 00:00:00+00:00
3	91f4f8f4f4f4f4f4f4f4f4f4f4f4f4f4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	delivered	2017-05-26 16:16:38+00:00	2017-05-26 16:30:38+00:00	2017-05-29 11:00:00+00:00	2017-06-05 16:16:38+00:00	2017-06-30 00:00:00+00:00
4	d7f5d1e2a1ee7d7d8d7d7d8d7d7d8d7d	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	delivered	2018-05-10 15:16:34+00:00	2018-05-10 15:35:31+00:00	2018-05-14 15:00:00+00:00	2018-05-22 15:16:34+00:00	2018-06-04 00:00:00+00:00

96475	5ccc0d1f5ca7f7d3a8f8c7f7d3a8f8c7	2017-06-19 03:37:04+00:00	2017-06-26 21:57:04+00:00	2017-06-29 00:00:00+00:00	10	29	40
96476	9f1f5e7f5e7f5e7f5e7f5e7f5e7f5e7f	2017-06-20 11:18:44+00:00	2017-06-28 11:18:44+00:00	2017-06-29 00:00:00+00:00	8	29	40
96476	0a0b1b1f0a0b1b1f0a0b1b1f0a0b1b1f	2017-06-20 18:44:00+00:00	2017-06-27 18:44:00+00:00	2017-06-29 00:00:00+00:00	7	29	40
96477	085d5a091f2f085d5a091f2f085d5a09	2017-06-23 10:19:00+00:00	2017-06-30 10:19:00+00:00	2017-06-30 00:00:00+00:00	7	29	37
96478	1f5e6a7f1f5e6a7f1f5e6a7f1f5e6a7f	2018-05-10 10:48:00+00:00	2018-05-18 10:48:00+00:00	2018-06-04 00:00:00+00:00	8	25	25

Next steps: Generate code with c7 View recommended plots

Section 6 : Analysis based on the Payments

Query 6.1: Month-on-Month Analysis of Orders Placed by Payment Types.

```
WITH query AS (
    SUM(payment_value) AS payment_value,
    SUM(CASE WHEN payment_type = 'credit_card' THEN payment_value ELSE 0 END) AS credit_card_Volume_of_Users,
    SUM(CASE WHEN payment_type = 'voucher' THEN payment_value ELSE 0 END) AS voucher_Volume_of_Users,
    SUM(CASE WHEN payment_type = 'debit_card' THEN payment_value ELSE 0 END) AS debit_card_Volume_of_Users,
    SUM(CASE WHEN payment_type = 'UPI' THEN payment_value ELSE 0 END) AS UPI_Volume_of_Users,
    SUM(CASE WHEN payment_type = 'not_defined' THEN payment_value ELSE 0 END) AS not_defined_Volume_of_Users
    FROM
    (select *,
    FROM `olist.payments`
    ) as payments_1
    ) SELECT *
    FROM query
    ORDER BY
    Year,
    Month
    LIMIT 5;
    ERROR: Error: Unrecognized name: EXTRACT; Maybe you meant to call function EXTRACT(DATE|DATETIME|TIMESTAMP|INTERVAL FROM ...) [at 4:5]
```

	Year	Month	credit_card_Volume_of_Users	voucher_Volume_of_Users	debit_card_Volume_of_Users	UPI_Volume_of_Users	not_defined_Volume_of_Users
0	2016	9	252	27	0	65	0
1	2016	10	19962	1707	155	4074	0
2	2017	1	16560	1848	143	8035	0
3	2017	2	33544	3274	291	12389	0
4	2017	3	47253	4828	404	20000	0

5	2017	4	37773	3305	277	15705	0
6	2017	5	54965	5269	411	23549	0
7	2017	6	43779	4268	338	19123	0
8	2017	7	54904	5277	411	24412	0
9	2017	8	62543	5904	470	27799	0
10	2017	9	69541	6545	509	30919	0
11	2017	10	68020	6546	521	30318	0
12	2017	11	98043	9493	748	43369	0
13	2017	12	78137	7243	567	34234	0
14	2018	1	94886	8971	707	41799	0
15	2018	2	81114	7563	596	35682	0
16	2018	3	90980	8456	666	39906	0
17	2018	4	89039	8376	659	39090	0
18	2018	5	89660	8376	659	39090	0

Next steps: Generate code with c7 View recommended plots

Query 6.2: Analysis of Orders Based on Payment Installments Paid.

```
WITH query AS (
    payment_installments,
    COUNT(DISTINCT order_id) AS num_orders,
    ROUND(100*SAFE_DIVIDE(COUNT(DISTINCT order_id),SUM(COUNT(DISTINCT order_id)) OVER ()), 2) AS order_density_percentage
    FROM
    (select *,
    FROM `olist.payments`
    ) as payments_1
    ) SELECT *
    FROM query
    ORDER BY
    payment_installments
    LIMIT 5;
```

	payment_installments	num_orders	order_density_percentage
0	0	47262	0.48
1	1	52546	53.12
2	2	12413	12.7
3	3	10571	10.81
4	4	7076	7.24
5	5	5328	5.45
6	6	4239	4.33
7	7	1626	1.66
8	8	4268	4.36
9	9	644	0.66
10	10	5475	5.6
11	11	23	0.02
12	12	133	0.14
13	13	16	0.02
14	14	15	0.02
15	15	74	0.08
16	16	5	0.01
17	17	8	0.01
18	18	5	0.01
19	19	1	0.0
20	20	3	0.0
21	21	1	0.0
22	22	1	0.0
23	23	1	0.0
24	24	10	0.01

Next steps: Generate code with c7 View recommended plots

Section 7 : RFM Analysis.

Query 7.1 : RFM Score.

```
WITH query AS (
    customer_id,
    Recency,
    Frequency,
    Monetary,
    NTILE(5) OVER (ORDER BY Recency DESC) R,
    NTILE(5) OVER (ORDER BY Frequency) F,
    NTILE(5) OVER (ORDER BY Monetary) M,
    CONCAT(R,F,M)
    FROM
    (SELECT customer_id, DATE_DIFF(CURRENT_DATE(),MAX(order_purchase_timestamp),DAY) AS Recency,
    COUNT(order_id) AS Frequency,
    SUM(payment_value) AS Monetary
    FROM
    (select *,
    FROM `olist.orders`
    ) as orders_1
    ) as orders_2
    GROUP BY
    customer_id
    ) SELECT *
    FROM query
    ORDER BY
    customer_id
    LIMIT 5;
```

	customer_id	Recency	Frequency	Monetary	R	F	M	RFM
0	00012a2ce6f8dcda20d059ce98491703	352	1	114.74	1	3	3	1-3-3 Stone
1	000161a058600d5901f007fab4c09062	351	1	67.41	1	2	2	1-2-2 Stone
2	0004c5d4bbf6d68a286e6f7f425ef551	444	1	195.42	1	3	3	1-3-3 Stone
3	00090f0b67f67f0d9f627f48f1f5a760	537	1	179.99	1	3	3	1-3-3 Stone
4	00096d1a1d9b17f6d8d4f4d5f0d4d4d4	435	1	107.78	1	3	3	1-3-3 Stone
5	000a229112604f727404d493948ff2bf	571	1	118.19	1	3	3	1-3-3 Stone
6	000b8f0a9f109f193d947d02f0f137f3	548	1	159.27	1	3	3	1-3-3 Stone
7	000bfa593b8a331fa332f4d8436e15f6	293	1	135.28	2	3	3	2-3-3 Stone
8	000c8231d2e5ef8ba2d74bc8a4a6d0c0	554	1	78.69	1	2	2	1-2-2 Stone
9	000d9be29b5207b54e86a137d1a50b65	136	1	25.99	3	2	1	3-2-1 Wooden
10	000e3561310a274bb48f4f5f6f0f4f4f	351	1	145.95	1	3	3	1-3-3 Stone

Next steps: Generate code with c7 View recommended plots

Query 7.1 : Customer Segmentation.

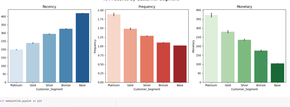
```
WITH query AS (
    customer_id,
    Recency,
    Frequency,
    Monetary,
    R,
    F,
    M,
    RFM,
    CASE
    WHEN RFM = '5-5-5' THEN 'Champions'
    WHEN RFM = '4-5-5' THEN 'Loyal'
    WHEN RFM = '5-4-5' THEN 'Potential'
    WHEN RFM = '5-5-4' THEN 'Recent'
    WHEN RFM = '5-3-5' THEN 'Promising'
    WHEN RFM = '5-2-5' THEN 'Need Attention'
    ELSE 'Other'
    END AS Customer_Segment
    FROM
    (SELECT customer_id, DATE_DIFF(CURRENT_DATE(),MAX(order_purchase_timestamp),DAY) AS Recency,
    COUNT(order_id) AS Frequency,
    SUM(payment_value) AS Monetary
    FROM
    (select *,
    FROM `olist.orders`
    ) as orders_1
    ) as orders_2
    GROUP BY
    customer_id
    ) SELECT *
    FROM query
    ORDER BY
    customer_id
    LIMIT 5;
```

	customer_id	Recency	Frequency	Monetary	R	F	M	RFM	Customer_Segment
0	00012

Bronze	2.00e+07	0.00e+00	0.00e+00	0.00e+00	0.00e+00	2.00 Bronze
Silver	1.00e+07	0.00e+00	0.00e+00	0.00e+00	0.00e+00	1.00 Silver
Gold	1.00e+07	0.00e+00	0.00e+00	0.00e+00	0.00e+00	1.00 Platinum
Platinum	1.00e+07	0.00e+00	0.00e+00	0.00e+00	0.00e+00	1.00

```
[36]: import matplotlib.pyplot as plt
      import seaborn as sns

      fig, axes = plt.subplots(1, 3, figsize=(15, 5))

      sns.boxplot(x='Customer_Segment', y='Recency', data=df, order=['Platinum', 'Gold', 'Silver', 'Bronze'],
      axes[0].set_title('Recency')

      sns.boxplot(x='Customer_Segment', y='Frequency', data=df, order=['Platinum', 'Gold', 'Silver', 'Bronze'],
      axes[1].set_title('Frequency')

      sns.boxplot(x='Customer_Segment', y='Monetary', data=df, order=['Platinum', 'Gold', 'Silver', 'Bronze'],
      axes[2].set_title('Monetary')

      plt.suptitle('RFM Scores by Customer Segment', fontsize=15)

      plt.tight_layout()
      plt.show()
```

```
/tmp/ipykernel_30/3621570611.py:5: FutureWarning:

Passing 'palette' without assigning 'hue' is deprecated and will be removed in v0.14.0. Assign the 'x' variable to 'hue' and set 'legend=False' for the same effect.

  sns.boxplot(x='Customer_Segment', y='Recency', data=df, order=['Platinum', 'Gold', 'Silver', 'Bronze'],
/tmp/ipykernel_30/3621570611.py:6: FutureWarning:

Passing 'palette' without assigning 'hue' is deprecated and will be removed in v0.14.0. Assign the 'x' variable to 'hue' and set 'legend=False' for the same effect.

  sns.boxplot(x='Customer_Segment', y='Frequency', data=df, order=['Platinum', 'Gold', 'Silver', 'Bronze'],
/tmp/ipykernel_30/3621570611.py:7: FutureWarning:

Passing 'palette' without assigning 'hue' is deprecated and will be removed in v0.14.0. Assign the 'x' variable to 'hue' and set 'legend=False' for the same effect.

  sns.boxplot(x='Customer_Segment', y='Monetary', data=df, order=['Platinum', 'Gold', 'Silver', 'Bronze'],
/tmp/ipykernel_30/3621570611.py:8: FutureWarning:

Passing 'palette' without assigning 'hue' is deprecated and will be removed in v0.14.0. Assign the 'x' variable to 'hue' and set 'legend=False' for the same effect.

  sns.boxplot(x='Customer_Segment', y='Monetary', data=df, order=['Platinum', 'Gold', 'Silver', 'Bronze'],
```




```
[37]: import matplotlib.pyplot as plt

      segment_counts = df['Customer_Segment'].value_counts()

      plt.figure(figsize=(8, 8))
      plt.pie(segment_counts, labels=segment_counts.index, autopct='%1.1f%%', startangle=140, colors=['skyblue', 'gold', 'silver', 'lightcoral'])
      plt.title('Customer Segments Distribution')
      plt.axis('equal')
      plt.show()
```

- Showing that the **low customers** comprise 57% of the customer base, the E-commerce store should position their marketing campaign to focus on transitioning base customers into the other segments, as base customers are most susceptible to churn.

RFM Analysis And Customer Segmentation

Platinum
Gold
Silver
Bronze
Base
Platinum
Gold
Silver
Bronze
Base
Platinum
Gold
Silver
Bronze
Base