WIE3007 DATA MINING & WAREHOUSING

SEMESTER 1 2023/2024

INDIVIDUAL ASSIGNMENT 1:

LEVERAGING FEATURETOOLS TO PERFORM AUTOMATED FEATURE ENGINEERING

LECTURER:

PROF. DR. TEH YING WAH

NAME: JASMINE CHONG SEE YAN

MATRIC NUMBER: S2132419

Table of Contents

Dataset Examination

1. About the Dataset

The dataset used in this assignment is ‘Brazilian E-Commerce Public Dataset by Olist’ and it is obtained from Kaggle. Olist is the largest department store in Brazil and it acts as a connector between small Brazilian businesses to reach out to a wider customers. This enables businesses to sell their products via the Olist store and ship them directly to customers using Olist Logistics partners (similar to the e-commerce businesses in Malaysia such as Shopee and Lazada).

The dataset consists of more than 100 000 data between 2016 to 2018. Besides, this dataset provides many dimensions to choose from such as customers, products, orders, reviews and sellers.

2. Possible Entities

a. Entity 1: Customers

In [19]:
```
customers.head()
```

Out[19]:

	customer_id	customer_zip_code_prefix	customer_city	customer_state
0	06b8999e2fba1a1fbc88172c00ba8bc7	14409	franca	SP
1	18955e83d337fd6b2def6b18a428ac77	9790	sao bernardo do campo	SP
2	4e7b3e00288586ebd08712fdd0374a03	1151	sao paulo	SP
3	b2b6027bc5c5109e529d4dc6358b12c3	8775	mogi das cruzes	SP
4	4f2d8ab171c80ec8364f7c12e35b23ad	13056	campinas	SP

b. Entity 2: Products

In [20]:
```
products.head()
```

Out[20]:

	product_id	product_category_name	product_name_lenght	product_description_lenght	product_photos_qty	product_weight_g
0	1e9e8ef04dbcff4541ed26657ea517e5	perfumaria	40.0	287.0	1.0	225.0
1	3aa071139cb16b67ca9e5dea641aaa2f	artes	44.0	276.0	1.0	1000.0
2	96bd76ec8810374ed1b65e291975717f	esporte_lazer	46.0	250.0	1.0	154.0
3	cef67bcfe19066a932b7673e239eb23d	bebes	27.0	261.0	1.0	371.0
4	9dc1a7de274444849c219cff195d0b71	utilidades_domesticas	37.0	402.0	4.0	625.0

product_length_cm	product_height_cm	product_width_cm
16.0	10.0	14.0
30.0	18.0	20.0
18.0	9.0	15.0
26.0	4.0	26.0
20.0	17.0	13.0

c. Entity 3: Reviews

In [21]:
```
reviews.head()
```
Out[21]:

	review_id	review_score	review_creation_date	review_answer_timestamp
0	7bc2406110b926393aa56f80a40eba40	4	18/1/2018 00:00	18/1/2018 21:46
1	80e641a11e56f04c1ad469d5645fdfde	5	10/3/2018 00:00	11/3/2018 03:05
2	228ce5500dc1d8e020d8d1322874b6f0	5	17/2/2018 00:00	18/2/2018 14:36
3	e64fb393e7b32834bb789ff8bb30750e	5	21/4/2017 00:00	21/4/2017 22:02
4	f7c4243c7fe1938f181bec41a392bdeb	5	1/3/2018 00:00	2/3/2018 10:26

d. Entity 4: Sellers

In [22]:
```
sellers.head()
```
Out[22]:

	seller_id	seller_zip_code_prefix	seller_city	seller_state
0	3442f8959a84dea7ee197c632cb2df15	13023	campinas	SP
1	d1b65fc7debc3361ea86b5f14c68d2e2	13844	mogi guacu	SP
2	ce3ad9de960102d0677a81f5d0bb7b2d	20031	rio de janeiro	RJ
3	c0f3eea2e14555b6faeea3dd58c1b1c3	4195	sao paulo	SP
4	51a04a8a6bdcb23deccc82b0b80742cf	12914	braganca paulista	SP

e. Entity 5: Orders

In [23]:
```
orders.head()
```
Out[23]:

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at
0	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	2017-10-02 10:56:33	2017-10-02 11:07:15
1	53cdb2fc8bc7dce0b6741e2150273451	b0830fb4747a6c6d20dea0b8c802d7ef	delivered	2018-07-24 20:41:37	2018-07-26 03:24:27
2	47770eb9100c2d0c44946d9cf07ec65d	41ce2a54c0b03bf3443c3d931a367089	delivered	2018-08-08 08:38:49	2018-08-08 08:55:23
3	949d5b44dbf5de918fe9c16f97b45f8a	f88197465ea7920adcdbec7375364d82	delivered	2017-11-18 19:28:06	2017-11-18 19:45:59
4	ad21c59c0840e6cb83a9ceb5573f8159	8ab97904e6daea8866dbdbc4fb7aad2c	delivered	2018-02-13 21:18:39	2018-02-13 22:20:29

order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date	product_id
2017-10-04 19:55:00	2017-10-10 21:25:13	2017-10-18 00:00:00	87285b34884572647811a353c7ac498a
2018-07-26 14:31:00	2018-08-07 15:27:45	2018-08-13 00:00:00	595fac2a385ac33a80bd5114aec74eb8
2018-08-08 13:50:00	2018-08-17 18:06:29	2018-09-04 00:00:00	aa4383b373c6aca5d8797843e5594415
2017-11-22 13:39:59	2017-12-02 00:28:42	2017-12-15 00:00:00	d0b61bfb1de832b15ba9d266ca96e5b0
2018-02-14 19:46:34	2018-02-16 18:17:02	2018-02-26 00:00:00	65266b2da20d04dbe00c5c2d3bb7859e

seller_id	shipping_limit_date	price	freight_value	review_id
3504c0cb71d7fa48d967e0e4c94d59d9	2017-10-06 11:07:15	29.99	8.72	a54f0611adc9ed256b57ede6b6eb5114
289cdb325fb7e7f891c38608bf9e0962	2018-07-30 03:24:27	118.70	22.76	8d5266042046a06655c8db133d120ba5
4869f7a5dfa277a7dca6462dcf3b52b2	2018-08-13 08:55:23	159.90	19.22	e73b67b67587f7644d5bd1a52deb1b01
66922902710d126a0e7d26b0e3805106	2017-11-23 19:45:59	45.00	27.20	359d03e676b3c069f62cadba8dd3f6e8
2c9e548be18521d1c43cde1c582c6de8	2018-02-19 20:31:37	19.90	8.72	e50934924e227544ba8246aeb3770dd4

3. Relationships

a. Customers has one-to-many relationship with Orders.
   i. A single customer can have one or many orders.
   ii. A single order belongs to one and only one customer.

b. Products has one-to-many relationship with Orders.
   i. A single product can have zero or many orders.
   ii. A single order contains one and only one product.

c. Reviews has a one-to-one relationship with Orders.
   i. A single review belongs to one and only one order.
   ii. A single order has one and only one review.

d. Sellers has one-to-many relationship with Orders.
   i. A single seller can have one or many orders.
   ii. A single order belongs to one and only one seller.

e. Orders has many-to-one relationships with Customers.
   i. A single order belongs to one and only one customer.
   ii. A single customer can have one or many orders.

f. Orders has many-to-one relationships with Products.
   i. A single order contains one and only one product.
   ii. A single product can have zero or many orders.

g. Orders has one-to-one relationships with Reviews.
   i. A single order has one and only one review.
   ii. A single review belongs to one and only one order.

h. Orders has many-to-one relationships with Sellers.
   i. A single order belongs to one and only one seller.

ii. A single seller can have one or many orders.

4. Hierarchical Structures

a. Customers

No.	Category	Column Name
1	Customer Information	customer_id
2	Customer Location	customer_state
		customer_city
		customer_zip_code_prefix

b. Products

No.	Category	Column Name
1	Product Information	product_id
2	Product Details	product_category_name
		product_description_length
		product_photos_qty
3	Product Dimensions and Weight	product_weight_g
		product_length_cm
		product_height_cm
		product_width_cm

c. Reviews

No.	Category	Column Name
1	Review Information	review_id

2	Review Scores	review_score
3	Review Timestamps	review_creation_date
		review_answer_timestamp

d. Sellers

No.	Category	Column Name
1	Seller Information	seller_id
2	Seller Location	seller_state
		seller_city
		seller_zip_code_prefix

e. Orders

No.	Category	Column Name
1	Order Information	order_id
2	Customer Information	customer_id
3	Order Status and Timestamps	order_status
		order_purchase_timestamp
		order_approved_at
		order_delivered_carrier_date
		order_delivered_customer_date
		order_estimated_delivery_date
4	Product Information	product_id
5	Seller Information	seller_id

6	Shipping Information	shipping_limit_date
7	Order Cost	price
		freight_value
8	Review Information	review_id

Featuretools Implementation: Python Code and Explanation

Link to code: https://github.com/JasmineChong/feature-engineering

1. Data Cleaning and Preparation

a. Load the datasets.

```
customers = pd.read_csv('Datasets/olist_customers_dataset.csv')
orders = pd.read_csv('Datasets/olist_orders_dataset.csv')
orderDetails = pd.read_csv('Datasets/olist_order_items_dataset.csv')
products = pd.read_csv('Datasets/olist_products_dataset.csv')
sellers = pd.read_csv('Datasets/olist_sellers_dataset.csv')
reviews = pd.read_csv('Datasets/olist_order_reviews_dataset.csv')
```

b. Check for any duplicates and drop them if any.

Drop duplicates

```
# Check for duplicated values in the primary key column
print(customers['customer_id'].duplicated().any())
print(products['product_id'].duplicated().any())
print(orders['order_id'].duplicated().any())
print(sellers['seller_id'].duplicated().any())
print(reviews['review_id'].duplicated().any())
```

```
False
False
False
False
True
```

```
# Drop duplicates in primary key column
reviews = reviews.drop_duplicates(subset = ['review_id'])

print(reviews['review_id'].duplicated().any())
```

```
False
```

c. Merge orderDetails dataframe into orders dataframe.

```
# Drop order_item_id
orderDetails.drop('order_item_id', axis=1, inplace=True)

# combine dataframe of orders and orderItems
orders = pd.merge(orders, orderDetails, on='order_id', how='inner')
```

d. Add review_id into the orders dataframe and drop the order_id in reviews dataframe.

```
# Add foreign key (review_id) into orders df
orders = pd.merge(orders, reviews[['order_id', 'review_id']], on='order_id', how='inner')
```

```
# Drop order_id in reviews df
reviews.drop('order_id', axis=1, inplace=True)
```

e. Check for any missing values and drop them if any.

```
# Check for any missing values
df_list = {
    'Customers': customers,
    'Orders': orders,
    'Products': products,
    'Sellers': sellers,
    'Reviews': reviews
}
null = []

for name, df in df_list.items():
    # Count missing values in each column
    null.append(df.isna().sum())

# Print null content
for name, null_values in zip(df_list.keys(), null):
    print(f"Null values in {name} DataFrame:")
    print(null_values)
    print("\n")
```

```
# Drop rows containing missing values
orders = orders.dropna()
products = products.dropna()
```

2. Define EntitySet and Entities

a. Create entity set object called ‘ecommerce_data’ using ft.EntitySet().

```
# creating and entity set 'es'
es = ft.EntitySet(id = 'ecommerce_data')
```

b. Define all entities by adding all dataframes into the EntitySet and set the primary keys.

```
# Adding a DataFrame as an entity
es.add_dataframe(dataframe_name='customers_entity', dataframe=customers, index='customer_id')
es.add_dataframe(dataframe_name='products_entity', dataframe=products, index='product_id')
es.add_dataframe(dataframe_name='orders_entity', dataframe=orders, index='order_id')
es.add_dataframe(dataframe_name='sellers_entity', dataframe=sellers, index='seller_id')
es.add_dataframe(dataframe_name='reviews_entity', dataframe=reviews, index='review_id')
```

```
Out[247]: Entityset: ecommerce_data
  DataFrames:
    customers_entity [Rows: 99441, Columns: 4]
    products_entity [Rows: 32340, Columns: 9]
    orders_entity [Rows: 95992, Columns: 14]
    sellers_entity [Rows: 3095, Columns: 4]
    reviews_entity [Rows: 99173, Columns: 4]
  Relationships:
    No relationships
```

3. Establish relationships between entities.

```
es.add_relationship(parent_dataframe_name='customers_entity', parent_column_name='customer_id',
                    child_dataframe_name='orders_entity', child_column_name='customer_id')
es.add_relationship(parent_dataframe_name='products_entity', parent_column_name='product_id',
                    child_dataframe_name='orders_entity', child_column_name='product_id')
es.add_relationship(parent_dataframe_name='sellers_entity', parent_column_name='seller_id',
                    child_dataframe_name='orders_entity', child_column_name='seller_id')
es.add_relationship(parent_dataframe_name='reviews_entity', parent_column_name='review_id',
                    child_dataframe_name='orders_entity', child_column_name='review_id')
```

```
Out[248]: Entityset: ecommerce_data
  DataFrames:
    customers_entity [Rows: 99441, Columns: 4]
    products_entity [Rows: 32340, Columns: 9]
    orders_entity [Rows: 95992, Columns: 14]
    sellers_entity [Rows: 3095, Columns: 4]
    reviews_entity [Rows: 99173, Columns: 4]
  Relationships:
    orders_entity.customer_id -> customers_entity.customer_id
    orders_entity.product_id -> products_entity.product_id
    orders_entity.seller_id -> sellers_entity.seller_id
    orders_entity.review_id -> reviews_entity.review_id
```

Deep Feature Synthesis (DFS) Implementation: Python Code and Explanation

1. Perform Deep Feature Synthesis (DFS) to generate additional new features on the fact table (orders_entity).

```
feature_matrix, feature_names = ft.dfs(entityset=es,
                                      target_dataframe_name = 'orders_entity')
```

```
In [251]: feature_matrix.columns

Out[251]: Index(['order_status', 'price', 'freight_value', 'DAY(order_approved_at)',
                 'DAY(order_delivered_carrier_date)',
                 'DAY(order_delivered_customer_date)',
                 'DAY(order_estimated_delivery_date)', 'DAY(order_purchase_timestamp)',
                 'DAY(shipping_limit_date)', 'MONTH(order_approved_at)',
                 ...
                 'reviews_entity.SUM(orders_entity.freight_value)',
                 'reviews_entity.SUM(orders_entity.price)',
                 'reviews_entity.DAY(review_answer_timestamp)',
                 'reviews_entity.DAY(review_creation_date)',
                 'reviews_entity.MONTH(review_answer_timestamp)',
                 'reviews_entity.MONTH(review_creation_date)',
                 'reviews_entity.WEEKDAY(review_answer_timestamp)',
                 'reviews_entity.WEEKDAY(review_creation_date)',
                 'reviews_entity.YEAR(review_answer_timestamp)',
                 'reviews_entity.YEAR(review_creation_date)'],
                dtype='object', length=110)
```

2. All the generated features are as follows:

```
order_status
price
freight_value
DAY(order_approved_at)
DAY(order_delivered_carrier_date)
DAY(order_delivered_customer_date)
DAY(order_estimated_delivery_date)
DAY(order_purchase_timestamp)
DAY(shipping_limit_date)
MONTH(order_approved_at)
MONTH(order_delivered_carrier_date)
MONTH(order_delivered_customer_date)
MONTH(order_estimated_delivery_date)
MONTH(order_purchase_timestamp)
MONTH(shipping_limit_date)
WEEKDAY(order_approved_at)
WEEKDAY(order_delivered_carrier_date)
WEEKDAY(order_delivered_customer_date)
WEEKDAY(order_estimated_delivery_date)
```

```
WEEKDAY(order_purchase_timestamp)
WEEKDAY(shipping_limit_date)
YEAR(order_approved_at)
YEAR(order_delivered_carrier_date)
YEAR(order_delivered_customer_date)
YEAR(order_estimated_delivery_date)
YEAR(order_purchase_timestamp)
YEAR(shipping_limit_date)
customers_entity.customer_zip_code_prefix
customers_entity.customer_city
customers_entity.customer_state
products_entity.product_category_name
products_entity.product_name_lenght
products_entity.product_description_lenght
products_entity.product_photos_qty
products_entity.product_weight_g
products_entity.product_length_cm
products_entity.product_height_cm
products_entity.product_width_cm
```

```
sellers_entity.seller_zip_code_prefix
sellers_entity.seller_city
sellers_entity.seller_state
reviews_entity.review_score
customers_entity.COUNT(orders_entity)
customers_entity.MAX(orders_entity.freight_value)
customers_entity.MAX(orders_entity.price)
customers_entity.MEAN(orders_entity.freight_value)
customers_entity.MEAN(orders_entity.price)
customers_entity.MIN(orders_entity.freight_value)
customers_entity.MIN(orders_entity.price)
customers_entity.MODE(orders_entity.order_status)
customers_entity.NUM_UNIQUE(orders_entity.order_status)
customers_entity.SKEW(orders_entity.freight_value)
customers_entity.SKEW(orders_entity.price)
customers_entity.STD(orders_entity.freight_value)
customers_entity.STD(orders_entity.price)
customers_entity.SUM(orders_entity.freight_value)
customers_entity.SUM(orders_entity.price)
```

```
products_entity.COUNT(orders_entity)
products_entity.MAX(orders_entity.freight_value)
products_entity.MAX(orders_entity.price)
products_entity.MEAN(orders_entity.freight_value)
products_entity.MEAN(orders_entity.price)
products_entity.MIN(orders_entity.freight_value)
products_entity.MIN(orders_entity.price)
products_entity.MODE(orders_entity.order_status)
products_entity.NUM_UNIQUE(orders_entity.order_status)
products_entity.SKEW(orders_entity.freight_value)
products_entity.SKEW(orders_entity.price)
products_entity.STD(orders_entity.freight_value)
products_entity.STD(orders_entity.price)
products_entity.SUM(orders_entity.freight_value)
products_entity.SUM(orders_entity.price)
sellers_entity.COUNT(orders_entity)
sellers_entity.MAX(orders_entity.freight_value)
sellers_entity.MAX(orders_entity.price)
```

```
sellers_entity.MEAN(orders_entity.freight_value)
sellers_entity.MEAN(orders_entity.price)
sellers_entity.MIN(orders_entity.freight_value)
sellers_entity.MIN(orders_entity.price)
sellers_entity.MODE(orders_entity.order_status)
sellers_entity.NUM_UNIQUE(orders_entity.order_status)
sellers_entity.SKEW(orders_entity.freight_value)
sellers_entity.SKEW(orders_entity.price)
sellers_entity.STD(orders_entity.freight_value)
sellers_entity.STD(orders_entity.price)
sellers_entity.SUM(orders_entity.freight_value)
sellers_entity.SUM(orders_entity.price)
reviews_entity.COUNT(orders_entity)
reviews_entity.MAX(orders_entity.freight_value)
reviews_entity.MAX(orders_entity.price)
reviews_entity.MEAN(orders_entity.freight_value)
reviews_entity.MEAN(orders_entity.price)
reviews_entity.MIN(orders_entity.freight_value)
reviews_entity.MIN(orders_entity.price)
```

```
reviews_entity.MODE(orders_entity.order_status)
reviews_entity.NUM_UNIQUE(orders_entity.order_status)
reviews_entity.SKEW(orders_entity.freight_value)
reviews_entity.SKEW(orders_entity.price)
reviews_entity.STD(orders_entity.freight_value)
reviews_entity.STD(orders_entity.price)
reviews_entity.SUM(orders_entity.freight_value)
reviews_entity.SUM(orders_entity.price)
reviews_entity.DAY(review_answer_timestamp)
reviews_entity.DAY(review_creation_date)
reviews_entity.MONTH(review_answer_timestamp)
reviews_entity.MONTH(review_creation_date)
reviews_entity.WEEKDAY(review_answer_timestamp)
reviews_entity.WEEKDAY(review_creation_date)
reviews_entity.YEAR(review_answer_timestamp)
reviews_entity.YEAR(review_creation_date)
```

3. Save the results into a new csv to be used to perform further analysis and gain insights.

Save feature_matrix as a csv file

```
feature_matrix.to_csv('feature_matrix.csv', index = False)
```

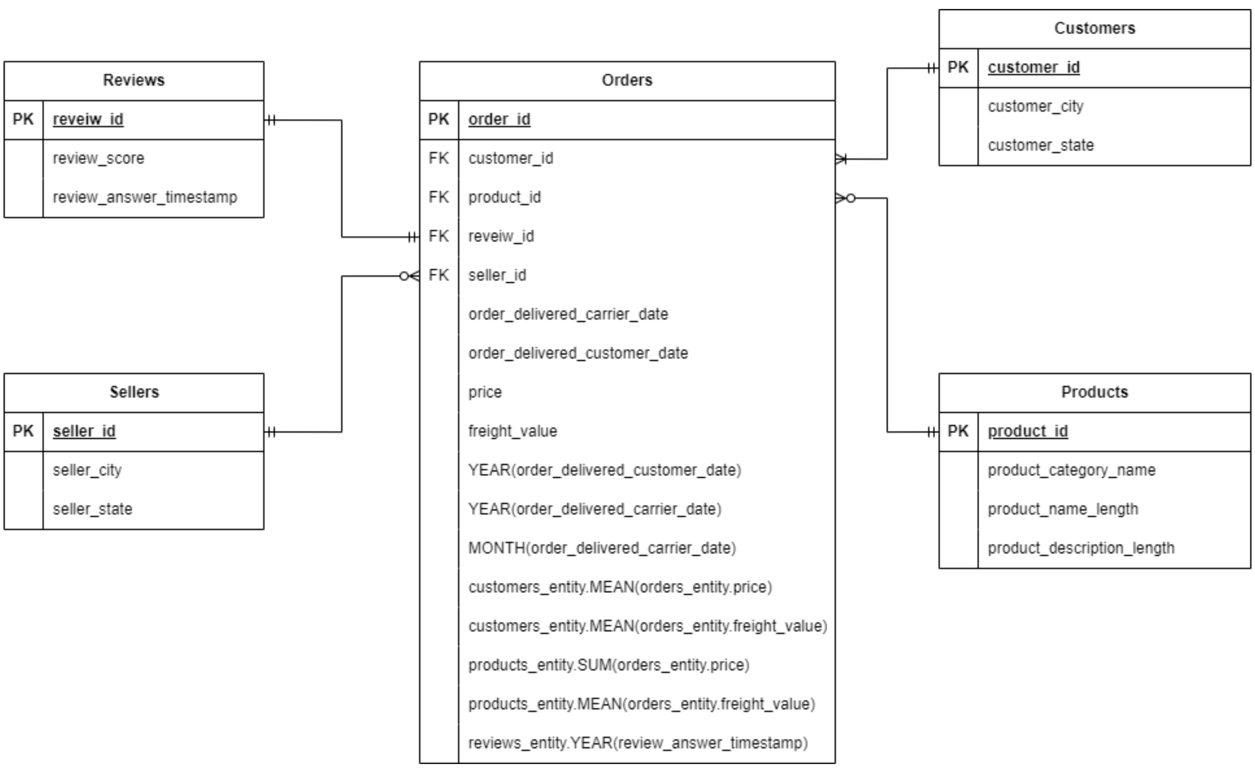
Objectives

- To identify the top 5 states with the highest percentage of customers for an efficient marketing strategy.
- To identify the top 5 states with the highest percentage of sellers for efficient, optimise and cost-effective logistics and delivery planning.
- To determine the average annual customers’ spendings for each state for customer segmentation.
- To identify the top 5 best-selling product categories for efficient inventory and marketing planning.
- To identify the annual monthly average freight cost to analyze and understand the shipping expenses over time.
- To determine the top 10 highest average product category's freight cost for pricing strategies and cost optimisation.
- To determine the number of reviews answered by the customer after their order has been delivered to gauge customers engagement.

Data Modeling

1. Optimised Star Schema

The fact table is the Order table while Customers, Products, Reviews and Sellers are the dimension tables. The presence of foreign keys in the fact table such as customer_id, product_id, review_id and seller_id connects the dimension tables, thus, establishing relationships between them. This allows for various analysis across dimensions.



Data Dictionary

1. Dimension Tables

a. Customers

Attribute	Data type	Constraints	Description
customer_id	String	• Primary key	A unique identifier for each customer
customer_city	String	• Not null	The city where the customer is located
customer_state	String	• Not null	The state where the customer is located

b. Products

Attribute	Data type	Constraints	Description
product_id	String	• Primary key	A unique identifier for each product
product_category_name	String	• Not null	The name of the category to which the product belongs
product_name_length	Integer	• Not null	The length of the product name in characters
product_description_length	String	• Not null	The length of the product description

c. Reviews

Attribute	Data type	Constraints	Description
review_id	String	• Primary key	A unique identifier for each review
review_score	Integer	• Not null	The score of the review
review_answer_timestamp	Datetime	• Not null	The timestamp of when the review was answered by the customer

d. Sellers

Attribute	Data type	Constraints	Description
seller_id	String	• Primary key	A unique identifier for each customer
seller_state	String	• Not null	The city where the customer is located
seller_city	String	• Not null	The state where the customer is located

2. Fact Tables

a. Orders

Attribute	Data type	Constraints	Description
order_id	String	• Primary key	A unique identifier for each order
customer_id	String	• Foreign key	A reference to the customer associated with the order
product_id	String	• Foreign key	A reference to the product associated with the order
reveiw_id	String	• Foreign key	A reference to the review associated with the order
seller_id	String	• Foreign key	A reference to the seller associated with the order
order_delivered_carrier_date	Datetime	• Not null	The date and time when the order was handed to the logistics partner
order_delivered_customer_date	Datetime	• Not null	The date and time when the order was delivered to the customer
price	Float	• Not null	The price of the order
freight_value	Float	• Not null	The cost of delivering the order to the customer
YEAR(order_delivered	Integer	• Not null	The year in which the order was

_customer_date)			delivered to the customer
YEAR(order_delivered_carrier_date)	Integer	• Not null	The year in which the order was delivered to the logistics partner
MONTH(order_delivered_carrier_date)	Integer	• Not null	The month in which the order was delivered to the logistics partner
customers_entity.MEAN(orders_entity.price)	Float	• Not null	The average order price for each customer
customers_entity.MEAN(orders_entity.freight_value)	Float	• Not null	The average freight cost (shipping cost) for each customer
products_entity.SUM(orders_entity.price)	Float	• Not null	The total sales for each product
products_entity.MEAN(orders_entity.freight_value)	Float	• Not null	The average freight cost (shipping cost) for each product
reviews_entity.YEAR(review_answer_timestamp)	Integer	• Not null	The year in which the review was answered by the customer

Insights Report Summary

Insights 1: Top 5 States with the Highest Percentage of Customers

Goal:

- To identify the top 5 states with the highest percentage of customers for an efficient marketing strategy.

Feature used:

- customers_entity.customer_state

Insights gained:

- The top 5 states with the highest percentage of customers are Sao Paulo, Rio de Janeiro, Minas Gerais, Rio Grande de Sul and Parana with 70.9%, 7.9%, 7.7%, 4.3% and 3.7% respectively whereas the Others which comprises of a total percentage of customers in 22 other states in Brazil has a cumulative score of 5.5% only. *(Refer to Figure 1)*
- The results allow the marketing team to strategise their marketing plan better, catering to each state's needs and suitability.

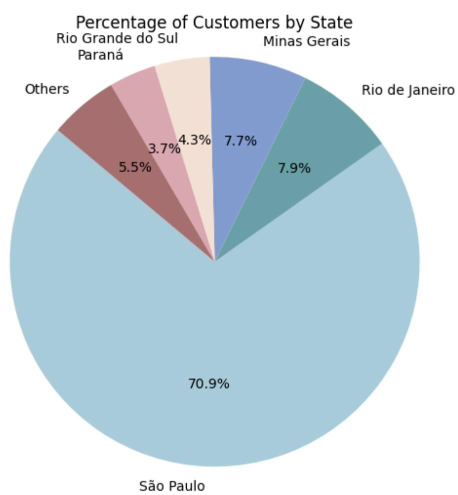


Figure 1: Customers Percentage by State

Python code:

1. Calculate number of customers for each unique state in the 'customers_entity.customer_state' column from the dataset using value_count().

```
# Count number of customers for each state
customers_state_counts = df['customers_entity.customer_state'].value_counts()
```

2. Calculate the total number of customers from the states not in the top 5 and store the results in the 'other_states_counts' variable.

```
# Get the index of the top 5 states with the most customers
top_5_states = customers_state_counts.head(5).index

# Calculate the total number of customers for states not in the top 5
other_states_counts = customers_state_counts[~customers_state_counts.index.isin(top_5_states)].sum()
```

3. Get the top 5 states using .head(5) and store the results as a dictionary using to_dict() and add the other states ('Others') to the same dictionary.

```
# Save the counts for the top 5 states and other states
states_count_dict = customers_state_counts.head(5).to_dict()
states_count_dict['Others'] = other_states_counts
```

4. Convert the dictionary to a Series.

```
# Convert the dictionary to a Series
states_counts = pd.Series(states_count_dict)
```

5. Plot the results into a pie chart.

```
# Plot the graph in a pie chart
plt.figure(figsize=(10, 6))

# Define the colors for the pie chart
# Link to color pallete: https://colorkit.co/palette/085578-538085-faf1e2-e3baaa-e47e8c-ffaa6a/
colours = [ '#a7cbd9', '#68a0a6', '#809bce', '#f2e0d5', '#d9a7b0', '#a66f6f']


labels = ['São Paulo', 'Rio de Janeiro', 'Minas Gerais', 'Rio Grande do Sul', 'Paraná', 'Others']
plt.pie(states_counts, labels=labels, autopct='%1.1f%%', startangle=140, colors=colours)

plt.axis('equal')

plt.title('Percentage of Customers by State')
plt.show()
```

Insights 2: Top 5 States with the Highest Percentage of Sellers

Goal:

- To identify the top 5 states with the highest percentage of sellers for efficient, optimise and cost-effective logistics and delivery planning.

Feature used:

- sellers_entity.seller_state

Insights gained:

- The top 5 states with the highest percentage of sellers are Sao Paulo, Minas Gerais, Parana, Rio de Janeiro and Santa Caterina with 70.9%, 7.9%, 7.7%, 4.3% and 3.7% respectively whereas the Others which comprises of a total percentage of customers in 22 other states in Brazil has a cumulative score of 5.5% only. *(Refer to Figure 2)*
- The results enable the ecommerce business to strategically locate distribution centers or warehouses and offer faster delivery times to customers living in the states with a higher concentration of sellers, leading to an increase customer satisfaction and trust.

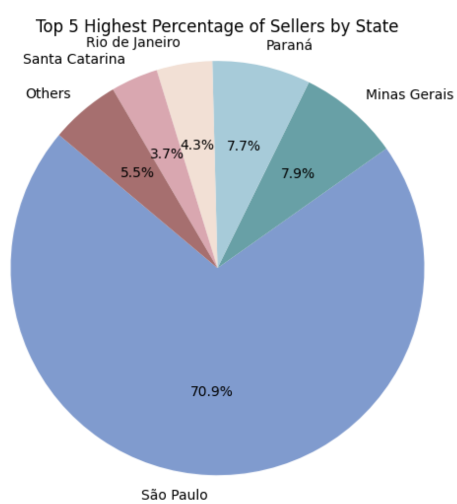


Figure 2: Sellers Percentage by State

Python code:

1. Calculate number of sellers for each unique state in the 'sellers_entity.seller_state' column from the dataset using value_count().

```
# Count number of sellers for each state
seller_state_counts = df['sellers_entity.seller_state'].value_counts()
```

2. Calculate the total number of sellers from the states not in the top 5 and store the results in the 'other_states_counts' variable.

```
# Select the top 5 state with the most number of sellers
top_5_states = seller_state_counts.head(5).index

# Calculate the total number of sellers for states not in the top 5
other_states_counts = seller_state_counts[~seller_state_counts.index.isin(top_5_states)].sum()
```

3. Get the top 5 states using .head(5) and store the results as a dictionary using to_dict() and add the other states ('Others') to the same dictionary.

```
# Save the counts for the top 5 states and other states
states_count_dict = customers_state_counts.head(5).to_dict()
states_count_dict['Others'] = other_states_counts
```

4. Convert the dictionary to a Series.

```
# Convert the dictionary to a Series
states_counts = pd.Series(states_count_dict)
```

5. Plot the results using a pie chart.

```
# Plot the graph in a pie chart
plt.figure(figsize=(10, 6))

colours = ['#809bce', '#68a0a6', '#a7cbd9', '#f2e0d5', '#d9a7b0', '#a66f6f']

labels = ['São Paulo', 'Minas Gerais', 'Paraná', 'Rio de Janeiro', 'Santa Catarina', 'Others']
plt.pie(states_counts, labels=labels, autopct='%1.1f%%', startangle=140, colors=colours )

plt.axis('equal')

plt.title('Top 5 Highest Percentage of Sellers by State')
plt.show()
```

Insights 3: Average Annual Customers' Spendings by State

Goal:

- To determine the average annual customers' spendings for each state for customer segmentation.

Feature used:

- customers_entity.customer_state
- customers_entity.MEAN(orders_entity.price)

Insights gained:

- The majority of the customers in all states spent more than R$100 yearly. Notably customers in Paraíba (PB) spent the most, that is at least R$200 on a yearly basis. *(Refer to Figure 3)*
- The results obtained allow the business to identify high-value and low-value customer segments. This in turn allows for more personalized marketing and product recommendations. Customers can also receive offers and content that are relevant to their spending habits and preferences.

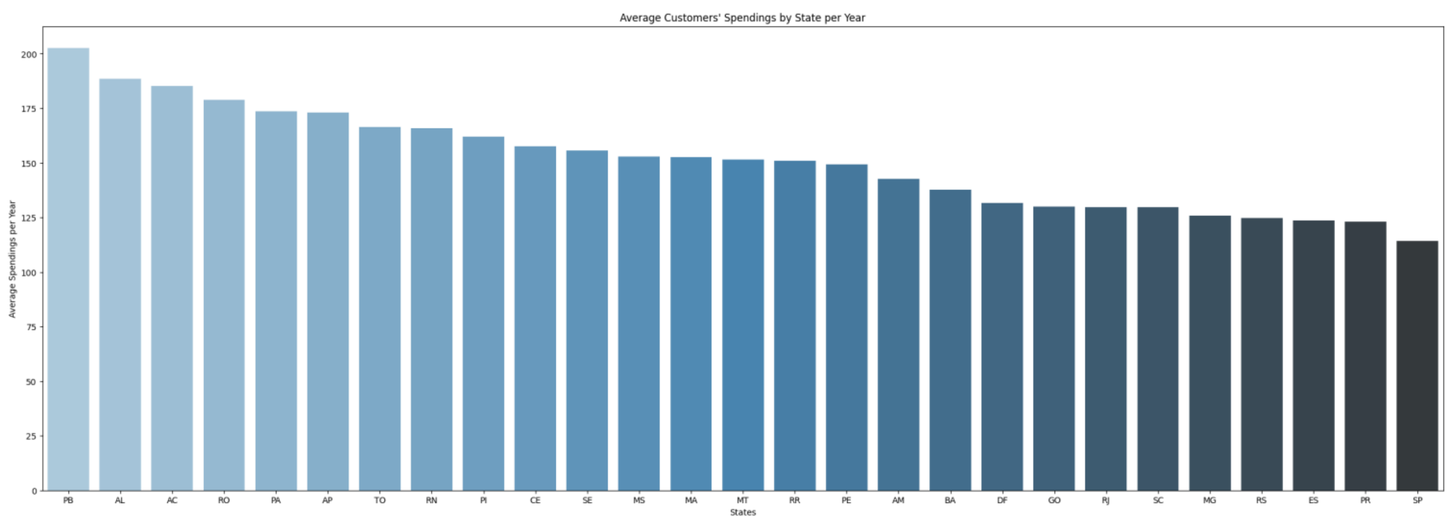


Figure 3: Average Annual Customers' Spendings by State

Python code:

1. Calculate the average order price of customers by state and store the results into 'customer_avg_spendings'.

```
# Group average order price by customer's state
customer_avg_spendings = df.groupby('customers_entity.customer_state')['customers_entity.MEAN(orders_entity.price)'].mean().reset_index()
```

2. Plot results in a bar plot.

```
# Plot graph
plt.figure(figsize=(30, 10))

sns.barplot(x='customers_entity.customer_state', y='customers_entity.MEAN(orders_entity.price)',
            data=customer_avg_spendings, palette='Blues_d')
plt.xlabel('States')
plt.ylabel('Average Spendings per Year')
plt.title('Average Customers\' Spendings by State per Year')
plt.show()
```

Insights 4: Top 5 Best-Selling Product Category

Goal:

- To identify the top 5 best-selling product categories for efficient inventory and marketing planning.

Feature used:

- products_entity.product_category_name
- products_entity.SUM(orders_entity.price)

Insights gained:

- The Top 5 best-selling products are beleza saude (Beauty & Health), relogios presents (Watches), informatica accessories (Computer Accessories), cama mesa banho (Bed & Bathroom) and ferramentas jardim (Gardening). *(Refer to Figure 4)*
- Knowing which product category are the best-sellers helps in optimizing inventory management. It ensures that high-demand products are adequately stocked.
- Business can also plan for targeted marketing campaigns and promotions for their best-selling products. Special offers, discounts, and advertising efforts can be focused on these products to boost sales further.

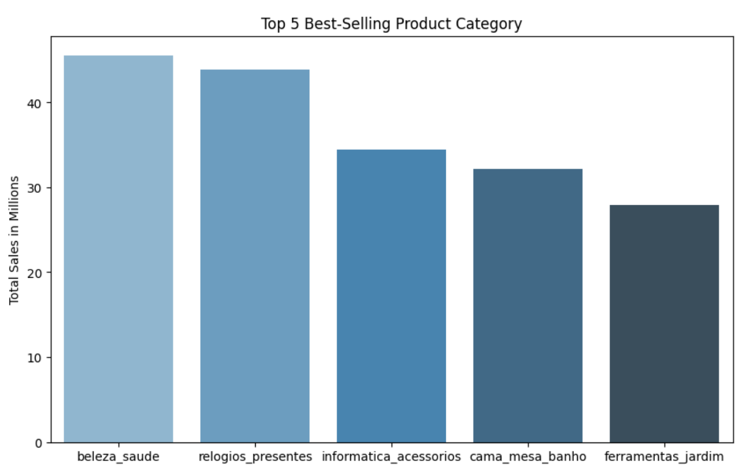


Figure 4: Top 5 Best-Selling Product Category

Python code:

1. Calculate the total sales for each product category and store the results into 'product_sales'.

```
# Group by product category and calculate the total sales for each product category
product_sales = df.groupby('products_entity.product_category_name')['products_entity.SUM(orders_entity.price)'].sum() .reset_index()
```

2. Sort the results in descending order.

```
# Sort data descendingly
product_sales = product_sales.sort_values(by='products_entity.SUM(orders_entity.price)', ascending=False)
```

3. As the total sales value is quite large, for simplicity, convert the values into smaller units by dividing them with 1 million such that the values in 'product_sales' are expressed in millions.

```
# Reduce the values by dividing by 1 000 000
product_sales['products_entity.SUM(orders_entity.price)_millions'] = product_sales['products_entity.SUM(orders_entity.price)']  / 1000000
```

4. Obtain the top 5 product category using .head(5).

```
# Get the top 5 product category
top_5_product = product_sales.head(5)
```

5. Plot results in a bar plot.

```
plt.figure(figsize=(10, 6))

sns.barplot(x='products_entity.product_category_name', y='products_entity.SUM(orders_entity.price)_millions',
            data=top_5_product, palette='Blues_d')
plt.xlabel('Product Category')
plt.ylabel('Total Sales in Millions')
plt.title('Top 5 Best-Selling Product Category')
plt.show()
```

Insights 5: Average Freight Cost per Month Per Year

Goal:

- To identify the annual monthly average freight cost to analyze and understand the shipping expenses over time.

Feature used:

- YEAR(order_delivered_carrier_date)
- MONTH(order_delivered_carrier_date)
- customers_entity.MEAN(orders_entity.freight_value)

Insights gained:

- The annual monthly average freight cost dropped at the end of year 2016 and increases slightly in early 2017. The drop is possibly due to the end-of-year season where there are special shipping rates and discounts whereas the increase means the shipping rate returns to normal. *(Refer to Figure 5)*
- The average freight cost remains constant throughout 2017 until the middle of the year 2018, indicating that the business has efficient logistics management. *(Refer to Figure 5)*
- The business experiences a sudden spike in their average freight cost from the month of August until September in the year 2018. The business might have experienced a surge in orders, thus requiring additional resources and leading to higher costs or the shipping fees might have increased during this period, impacting on the business's expenses. *(Refer to Figure 5)*
- The insights highlight the importance of monitoring and analyzing freight costs regularly. Businesses can use this data to identify cost anomalies, make timely adjustments, and potentially implement cost-saving measures.

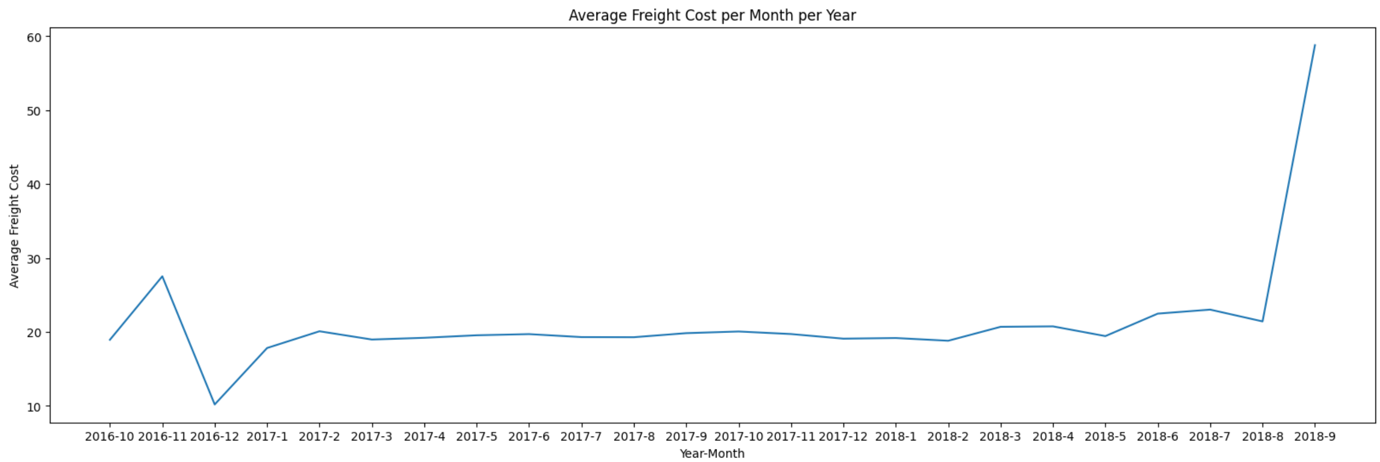


Figure 5: Average Annual Monthly Freight Cost

Python code:

1. Calculate the annual monthly average freight cost for every order and store the results into 'freight_cost'.

```
# Group transport fees by month & year & calcualte the average freight cost for each order
freight_cost = df.groupby(['YEAR(order_delivered_carrier_date)', 'MONTH(order_delivered_carrier_date)'])

['customers_entity.MEAN(orders_entity.freight_value)'].mean().reset_index()
```

2. As the year and month for each order are separated into 2 different column, combine the 2 columns such that the year and month are in the same column so that it can be used as the x-axis label.

```
# Create a new column that combines 'year' and 'month' for the x-axis labels
freight_cost['year_month'] = freight_cost['YEAR(order_delivered_carrier_date)'].astype(str) + '-' +

 freight_cost['MONTH(order_delivered_carrier_date)'].astype(str)
```

3. Plot results in a line plot.

```
# Create a line plot
plt.figure(figsize=(20, 6))
plt.plot(freight_cost['year_month'], freight_cost['customers_entity.MEAN(orders_entity.freight_value)'])
plt.xlabel('Year-Month')
plt.ylabel('Average Freight Cost')
plt.title('Average Freight Cost per Month per Year')
plt.xticks(rotation=0)
plt.show()
```

Insights 6: Top 10 Highest Average Product Category's Freight Cost

Goal:

- To determine the top 10 highest average product category's freight cost for pricing strategies and cost optimisation.

Feature used:

- products_entity.product_category_name
- products_entity.MEAN(orders_entity.freight_value)

Insights gained:

- The top 10 highest average product category’s freight cost are PCs, electrodomestica (Household Appliaces), moveis colchao e estofado (Upholstered Furniture), móveis quarto (Bedroom Furniture), moveis cozinha area de servico jantar e jardim (Kitchen & Dining Furniture), moveis escritorio (Office Furniture), portateis casa forno e café (Oven & Coffee Machines), moveis sala (Living Room Furniture), industria comercio e negocios (Industry, Commerce And Business) and malas acessórios (Bags Accessories). *(Refer to Figure 6)*
- Understanding the product categories with high freight costs encourages business to focus on cost optimization. They can explore ways to reduce shipping expenses within these categories, such as improving packaging efficiency or negotiating better shipping rates.
- The results also prompt the business to assess supply chain efficiency. It encourages them to streamline logistics operations and investigate more efficient shipping methods.

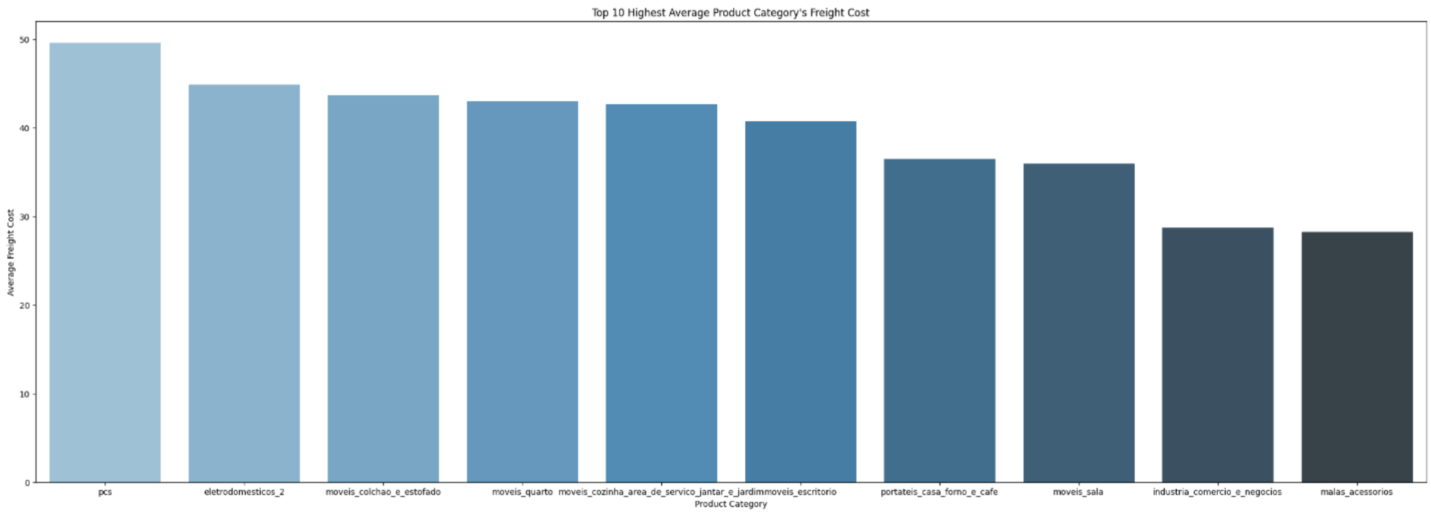


Figure 6: Top 10 Highest Average Freight Cost by Product Category

Python code:

1. Calculate the average freight cost for each product category for every product order and store the results into 'freight_cost_by_product'.

```
# Group freight cost by product category and calculate the average freight cost for each product order
freight_cost_by_product = df.groupby('products_entity.product_category_name')['products_entity.MEAN(orders_entity.freight_value)' ].mean().reset_index()
```

2. Sort the results in descending order.

```
# Sort data descendingly
freight_cost_by_product = freight_cost_by_product.sort_values(by='products_entity.MEAN(orders_entity.freight_value)', ascending=False)
```

3. Obtain the top 10 product category using .head(10).

```
# Get the top 10 product category for freight cost
top_10_freight_cost = freight_cost_by_product.head(10)
```

4. Plot results in a bar plot.

```
# Plot the graph
plt.figure(figsize=(30, 10))

sns.barplot(x='products_entity.product_category_name', y='products_entity.MEAN(orders_entity.freight_value)',
            data=top_10_freight_cost, palette='Blues_d')
plt.xlabel('Product Category')
plt.ylabel('Average Freight Cost')
plt.title('Top 10 Highest Average Product Category\'s Freight Cost')
plt.show()
```

Insights 7: The Number of Reviews Answered After Customer's Order is Delivered

Goal:

- To determine the number of reviews answered by the customer after their order has been delivered to gauge customers engagement.

Feature used:

- YEAR(order_delivered_customer_date)
- reviews_entity.YEAR(review_answer_timestamp)

Insights gained:

- It can be observed that almost all of the customers answered the reviews after their order has been delivered. *(Refer to Figure 7)*
- The number of reviews answered after the order is delivered indicates the level of customer engagement and satisfaction. Higher numbers may suggest that customers are actively providing feedback to the business, which could be a positive sign.
- Reviews answered after delivery often contain information about the received product's quality. Analyzing these reviews helps in evaluating and improving the overall quality of the products offered and gauge customer satisfaction.

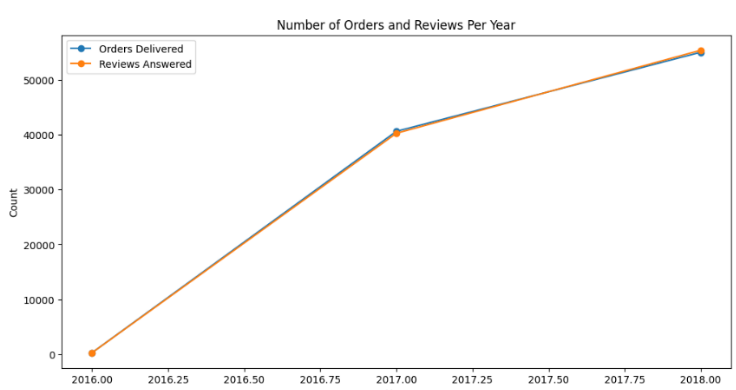


Figure 7: Number of Orders and Reviews Answered Per Year

Python code:

1. Calculate the number of orders and reviews per year using the .value_counts().

```
# Count the number of orders and reviews per year
orders_per_year = df['YEAR(order_delivered_customer_date)'].value_counts().sort_index()
reviews_per_year = df['reviews_entity.YEAR(review_answer_timestamp)'].value_counts().sort_index()
```

2. Create a data frame to display the results easily.

```
# Create a dataframe to display the results
result_df = pd.DataFrame({
    'Year': orders_per_year.index,
    'Orders Delivered': orders_per_year.values,
    'Reviews Answered': reviews_per_year.values
})
```

3. Plot the results.

```
# Plot the results
plt.figure(figsize=(12, 6))

plt.plot(result_df['Year'], result_df['Orders Delivered'], label='Orders Delivered', marker='o')
plt.plot(result_df['Year'], result_df['Reviews Answered'], label='Reviews Answered', marker='o')

plt.title('Number of Orders and Reviews Per Year')
plt.xlabel('Year')
plt.ylabel('Count')
plt.legend()

plt.show()
```

Reflection

Featuretools is an open-source library for performing automated feature engineering. Automated feature engineering tools have a big impact in designing data models for data warehouses. These tools help find new details and relationships in the data, which makes data models understand and predict things better.

These tools are like detectives that uncover hidden patterns and relationships in the data. They connect the dots between different parts of the data, helping to see how things are related. This is like finding clues that we might have missed if we were looking at the data manually.

Using these tools not only makes data models more accurate, but it also saves time. They do a lot of the work, so that more focus can be placed on the important stuff like making decisions based on the data.

In today's data-driven world, these tools are essential for getting the most out of data and staying competitive. They help create better models that give valuable insights, thus making better decisions.

References

Brazilian E-Commerce Public Dataset by OLIST. (2018, November 29). Kaggle. https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce/versions/7?resource=download

Joshi, P. (2022). *A Hands-On Guide to Automated Feature Engineering using Featuretools in Python*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2018/08/guide-automated-feature-engineering-featuretools-python/

Singh, R. (2023). Feature Engineering using Featuretools with code - Rana singh - Medium. *Medium*. https://ranasinghiitkgp.medium.com/feature-engineering-using-featuretools-with-code-10f8c83e5f68