ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ
SCHOOL OF BUSINESS

ΤΜΗΜΑ ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ & ΤΕΧΝΟΛΟΓΙΑΣ
DEPARTMENT OF MANAGEMENT SCIENCE & TECHNOLOGY

Μάθημα «Επιχειρηματική Ευφυΐα και Ανάλυση Μεγάλων Δεδομένων»
7ο Εξάμηνο

Data Warehouse and Data Mining Using Olist Dataset

Ομάδα Εργασίας (10):
Μάστορη Άννα (t8190105)
Νικόλαος Γιαννάτος (t8190025)

2023

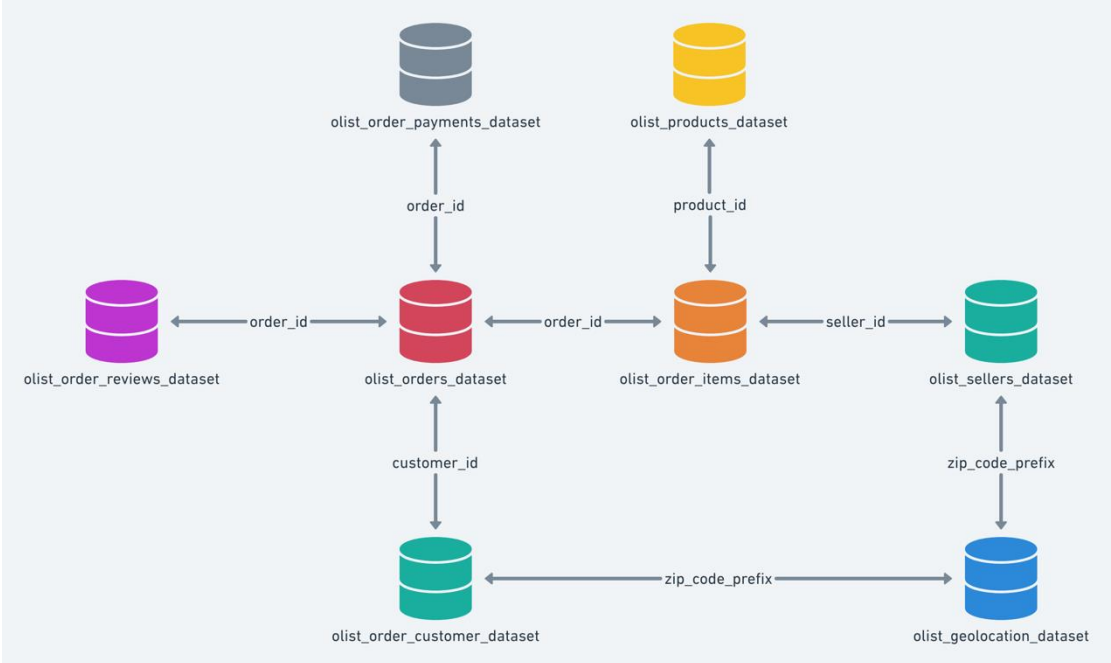
CONTENTS

1. What is Olist:

The Olist store is an e-commerce business headquartered in Sao Paulo, Brazil. This firm acts as a single point of contact between various small businesses and the customers who wish to buy their products. Olist connects small businesses from all over Brazil to channels without hassle and with a single contract. Those merchants are able to sell their products through the Olist Store and ship them directly to the customers using Olist logistics partners. Olist profit comes from a commission on products sold by the sellers.

2. About our Dataset:

This dataset was generously provided by Olist, the largest department store in Brazilian marketplaces. The Dataset provides data from 01/01/2017 to 06/01/2018. The dataset has information of over 100k orders from 2016 to 2018 made by multiple marketplaces in Brazil. The Dataset was separated into many different sub-dataset which we used to extract various information concerning the customer, the seller, the products, the payments, the reviews made by the customers ,the orders and the locations. We managed to use them all except the geolocation to create useful insights.
[Brazilian E-Commerce Public Dataset by Olist | Kaggle]



(Image taken from: Data Schema)

It's important to mention that this Dataset is extremely popular with over **165000** downloads and **0.17** download per view ratio

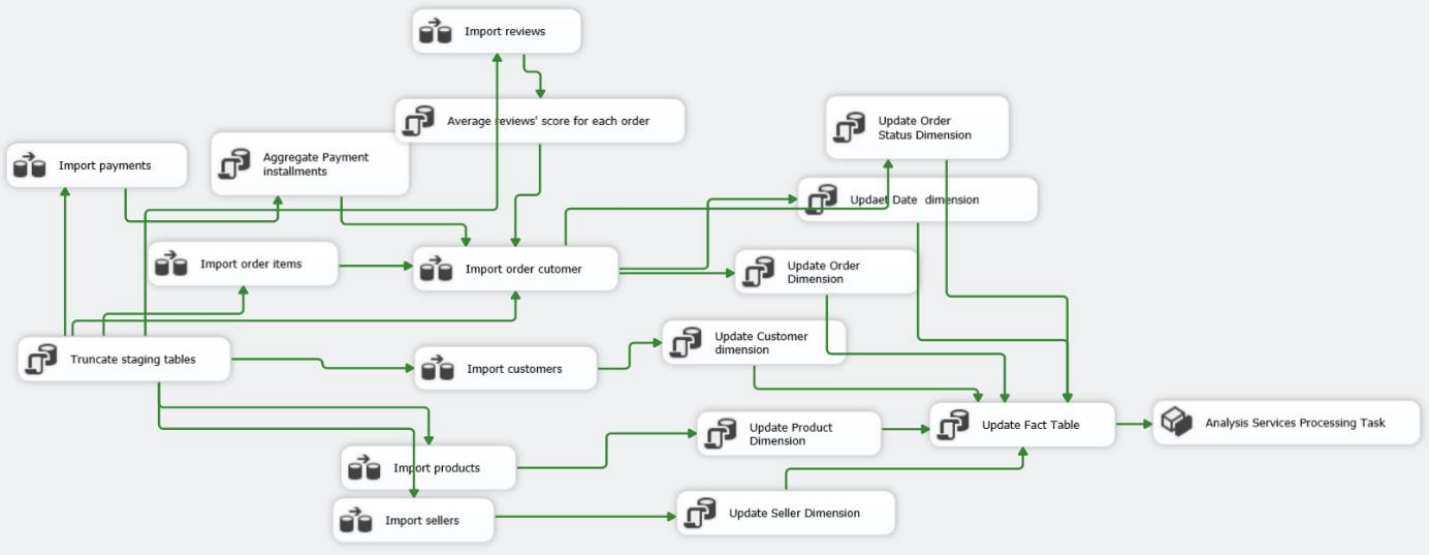
3. Extract – Transform – Load Process:

The dataset we used consisted of the following csv files : olist_orders_dataset.csv, olist_sellers_dataset.csv, olist_products_in_spanish_dataset.csv, product_category_name_translation.csv, olist_products_dataset.csv, olist_order_reviews_dataset.csv, olist_order_payments_dataset.csv, olist_order_items_dataset.csv, olist_geolocation_dataset.csv, olist_customers_dataset.csv
Use did not use the olist_geolocation_dataset.csv in our project.

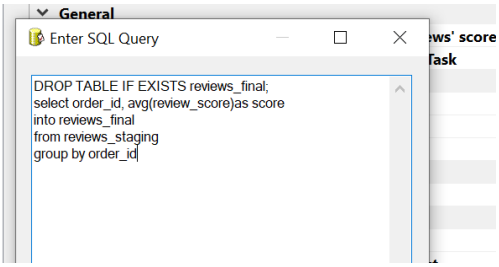
Firstly, with python we translated the product categories names' in the olist_products_dataset.csv file using product_category_name_translation.csv with the correct name of the categories in English which came along with the dataset.

```
import pandas as pd

trans= pd.read_csv('olist_products_dataset.csv')
prods= pd.read_csv('olist_products_in_spanish_dataset.csv')

trans.set_index(['product_category_name'],inplace=True)
dict=pd.Series(trans.product_category_name_english).to_dict()
prods=prods.replace(dict)
prods.fillna(0,inplace=True)
prods.astype({"product_width_cm": int, "product_height_cm": int,'product_length_cm':int,'product_weight_g':int},copy=False)
prods.to_csv('olist_products_dataset.csv',index=False)
```

After that we continued with process using only SSIS and SQL Server.

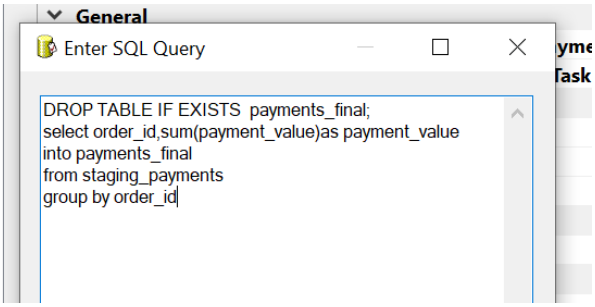


We imported in the sql server database each csv file as an sql table using a data flow from the SSIS toolbox. For each csv a staging table was created (reviews_staging, order_item_staging,



staging_customer, staging_products, staging_sellers, staging_payments, staging_order_customer).

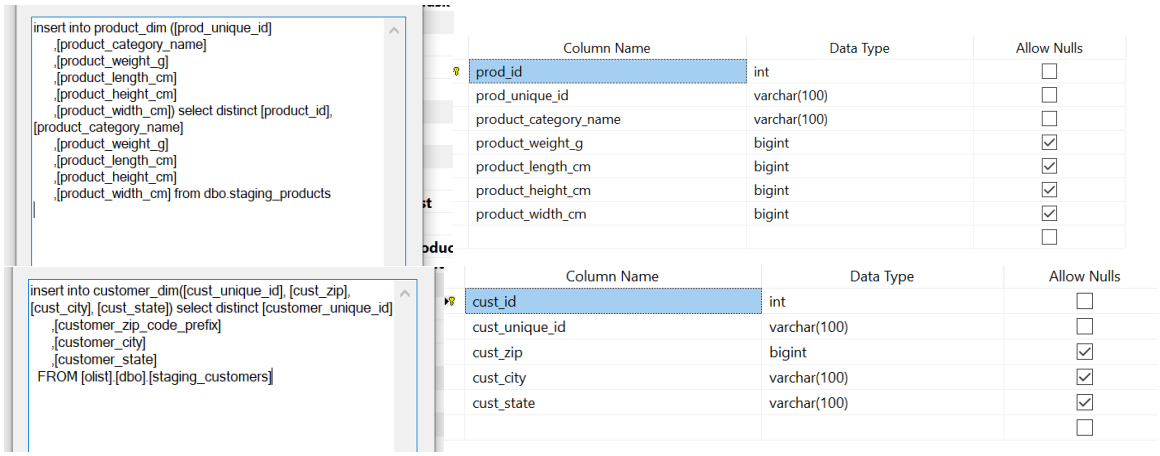
The reviews_staging table has for some orders more than one review and because we want one review per order we create a new table reviews_final in which we have the average of the reviews of each order.
The staging_payments table has one row for each installment and the corresponding payments amount. We need for our analysis the sum of all the payment amounts and for that reason we created a new table with the sum per order payments_final.



Then we created a view named orders that is the blueprint for the fact table we will need later. Into the view, we inner joined the tables payments_final, order_item_staging, staging_order_customer on their common column order_id . We did on the above joined tables an outer left join with reviews_final in order to keep the rows that had no review score with null as score.

```
SELECT dbo.order_item_staging.order_id, dbo.order_item_staging.product_id, dbo.order_item_staging.price, dbo.order_item_staging.seller_id, dbo.order_item_staging.freight_value
       dbo.staging_order_customer.order_purchase_timestamp, dbo.staging_order_customer.order_delivered_customer_date, dbo.staging_order_customer.order_estimated_deliv
FROM   dbo.order_item_staging INNER JOIN
       dbo.staging_order_customer ON dbo.order_item_staging.order_id = dbo.staging_order_customer.order_id INNER JOIN
       dbo.payments_final ON dbo.staging_order_customer.order_id = dbo.payments_final.order_id LEFT OUTER JOIN
       dbo.reviews_final ON dbo.staging_order_customer.order_id = dbo.reviews_final.order_id
```

Also, from the staging_products, staging_sellers, staging_customer we created the corresponding dimension tables.
Indicatively, we will so the steps we took to create 2 of the dimensiontables.



Product_dim has 32951 unique products, seller_dim contains 3095 sellers and customer_dim has 195537 customers.

Also, for the columns order_purchase_timestamp date, order_estimation_delivery date, order_delivered_customer date we created a date dimension.

Column Name	Data Type	Allow Nulls
date_id	int	☐
sqldate	date	☐
day	int	☑
day_of_week	varchar(100)	☑
month	int	☑
month_name	varchar(150)	☑
year	int	☑
quarter	int	☑

It contains all sql dates that an order was either purchased, delivered or estimated to be delivered. The dimension apart from the sql date contains columns for the day of the month, name of the day_of_week, month number and name, year and quarter. To fill the table with the derived columns we used the following query:

```
Select distinct CONVERT(date, dates) as dates
Into  #Temp
From  (SELECT orders.order_delivered_customer_date as dates FROM [olist].[dbo].[orders]
 UNION
 SELECT orders.order_estimated_delivery_date FROM [olist].[dbo].[orders]
 UNION
 SELECT orders.order_purchase_timestamp FROM [olist].[dbo].[orders]
) as all_dates


Declare @currentdate date

DECLARE
        @DayOfWeekInMonth INT,
        @DayOfWeekInYear INT,
        @DayOfQuarter INT,
        @WeekOfMonth INT,
        @CurrentYear INT,
        @CurrentMonth INT,
        @CurrentQuarter INT

/*Table Data type to store the day of week count for the month and year*/
DECLARE @DayOfWeek TABLE (DOW INT, MonthCount INT, QuarterCount INT, YearCount INT)

INSERT INTO @DayOfWeek VALUES (1, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (2, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (3, 0, 0, 0)
        UPDATE @DayOfWeek
        SET
                MonthCount = MonthCount + 1,
                QuarterCount = QuarterCount + 1,
                YearCount = YearCount + 1
        WHERE DOW = DATEPART(DW, @CurrentDate)

        SELECT
                @DayOfWeekInMonth = MonthCount,
                @DayOfQuarter = QuarterCount,
                @DayOfWeekInYear = YearCount
        FROM @DayOfWeek
        WHERE DOW = DATEPART(DW, @CurrentDate)

/*End day of week logic*/


/* Populate Your Dimension Table with values*/

        INSERT INTO olist.[dbo].[date_dim](sqldate, [day],day_of_week, [month],[month_name],[year],[quarter])
        SELECT


                @CurrentDate AS Date,

                DATEPART(DD, @CurrentDate) AS DayOfMonth,
                DATENAME(DW, @CurrentDate) AS DayName,

                DATEPART(MM, @CurrentDate) AS Month,
                DATENAME(MM, @CurrentDate) AS MonthName,
                DATEPART(YEAR, @CurrentDate) AS Year,

                DATEPART(QQ, @CurrentDate) AS Quarter

   Delete #Temp Where dates = @currentdate
        Select Top 1 @currentdate = dates From #Temp

End
```

```
 Select Top 1 @currentdate = dates From #Temp
SET @CurrentMonth = DATEPART(MM, @CurrentDate)
SET @CurrentYear = DATEPART(YY, @CurrentDate)
SET @CurrentQuarter = DATEPART(QQ, @CurrentDate)


WHILE EXISTS(SELECT * FROM #Temp)
Begin

  IF @CurrentMonth != DATEPART(MM, @CurrentDate)
        BEGIN
                UPDATE @DayOfWeek
                SET MonthCount = 0
                SET @CurrentMonth = DATEPART(MM, @CurrentDate)
        END

    /* Check for Change in Quarter of the Current date if Quarter changed then change
     Variable value*/

        IF @CurrentQuarter != DATEPART(QQ, @CurrentDate)
        BEGIN
                UPDATE @DayOfWeek
                SET QuarterCount = 0
                SET @CurrentQuarter = DATEPART(QQ, @CurrentDate)
        END

    /* Check for Change in Year of the Current date if Year changed then change
     Variable value*/
```

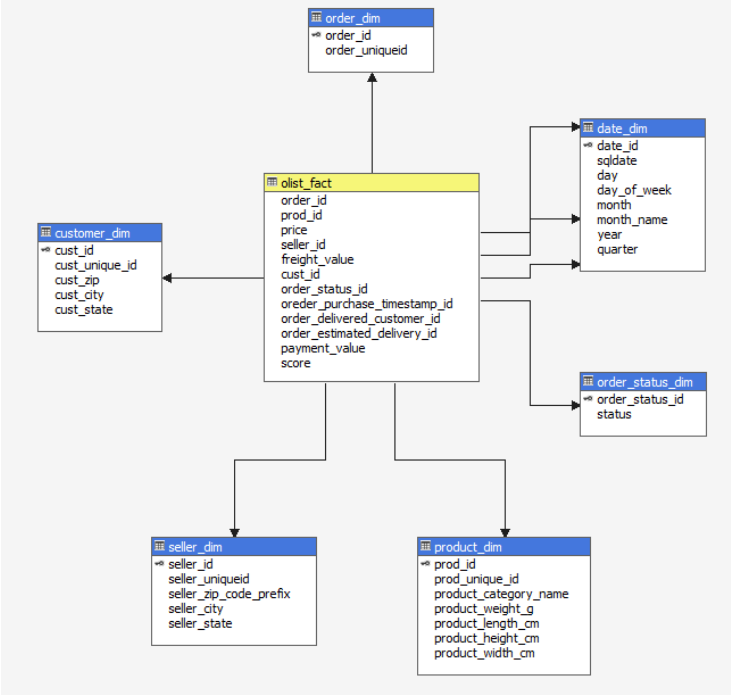
Also we created an order dimension and an order_status dimension for the distinct statuses that an order can have (approved, canceled, delivered, invoiced, processing, shipped unavailable). Finally, we created the fact table of our data warehouse named olist_fact. It contained foreign key to the dimensions order, customer, product, seller, date, order_status and measures: score, freight, price, payment_value.

Column Name	Data Type	Allow Nulls
order_id	int	☐
prod_id	int	☐
price	float	☑
seller_id	int	☐
freight_value	float	☑
cust_id	int	☐
order_status_id	int	☐
oreder_purchase_timestamp_id	int	☐
order_delivered_customer_id	int	☐
order_estimated_delivery_id	int	☐
payment_value	float	☑
score	bigint	☑

```
use olist
Insert into olist.dbo.olist_fact
select  order_dim.[order_id]
     ,product_dim.prod_id
    ,orders.[price]
              ,seller_dim.[seller_id]
     ,orders.[freight_value]
   ,[customer_dim].cust_id
     ,order_status_dim.[order_status_id]
     ,pur.date_id
              ,del.date_id
              ,est.date_id
              ,orders.payment_value
              ,orders.score
  --FROM (select * from [olist].[dbo].[orders] where [order_id] not in (select distinct order_id from olist_fact) )as orders
  FROM orders inner join [order_dim] on [order_dim].[order_uniqueid]= [orders].[order_id]
  Inner join [product_dim] on product_dim.prod_unique_id= orders.product_id
 Inner join [customer_dim] on customer_dim.cust_unique_id= orders.customer_id
inner join [seller_dim] on seller_dim.[seller_uniqueid]= orders.seller_id
  inner join [order_status_dim] on [order_status_dim].[status]=orders.order_status
  inner join [date_dim] as pur on pur.[sqldate]= CONVERT(date,orders.order_purchase_timestamp)
  inner join [date_dim] as del on del.[sqldate]= CONVERT(date, orders.order_delivered_customer_date )
  inner join [date_dim] as est on est.[sqldate]= CONVERT(date, orders.order_estimated_delivery_date )
```

As shown in the diagram, the E-T-L process we created truncates the staging tables, imports all the csv files we use, updates all the dimensions and the fact table and finally processes the cube in order for the changes to be incorporated in the olap server.

4. Star Schema

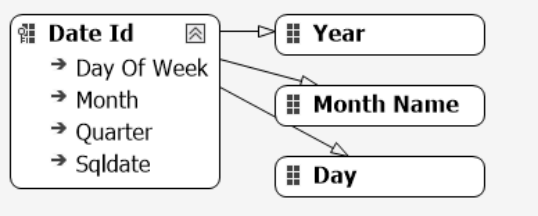


In the previous step when we created the dimension tables and the fact table in essence we had created our data warehouse modeled as a star schema. Our star schema consists of 1 fact table and 6 dimensions. The dimensions are the following: customer dimension, product dimension, seller dimension, order status dimension, date dimension, order dimension and product dimension. Date dimension has 3 foreign keys attached for the 3 different dates (order_purchase_timestamp date, order_estimation_delivery date, order_delivered_customer date).
Our fact table has a row for each item purchased, approximately 100.000 rows. It contains the following measures :freight value of each item, price of each item, score for each order(it is the same for all items of an order).

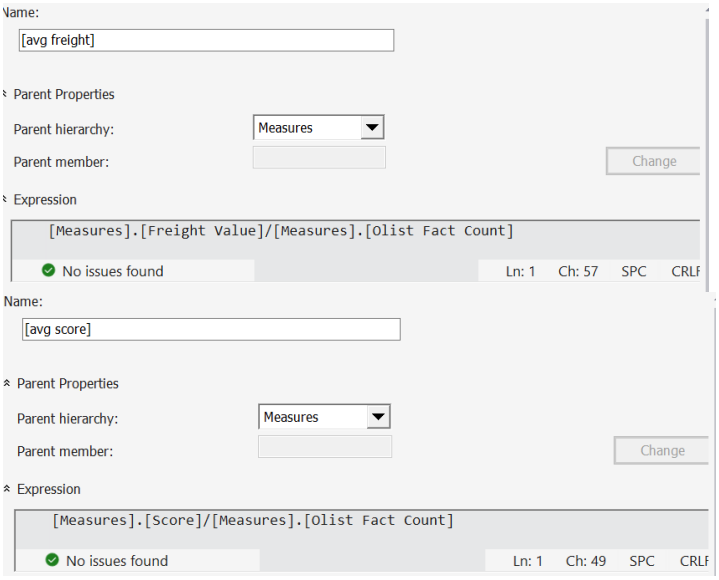
5. Cube

We used as data source the SQL Server database olist we previously created. Then, for the data source view we chose from all the tables on the sql server only the fact table and the dimension tables. Finally, with the cube wizard we created a cube using as data source the above mentioned data source.
Our multidimentional cube and consists of 1 fact table and 6 dimensions.

We created a hierarchy of the date attributes for the Date dimension.



We also created 2 Calculated Members in the cube, average score and average freight value because SSAS does not have an AggregatedFunction for average.

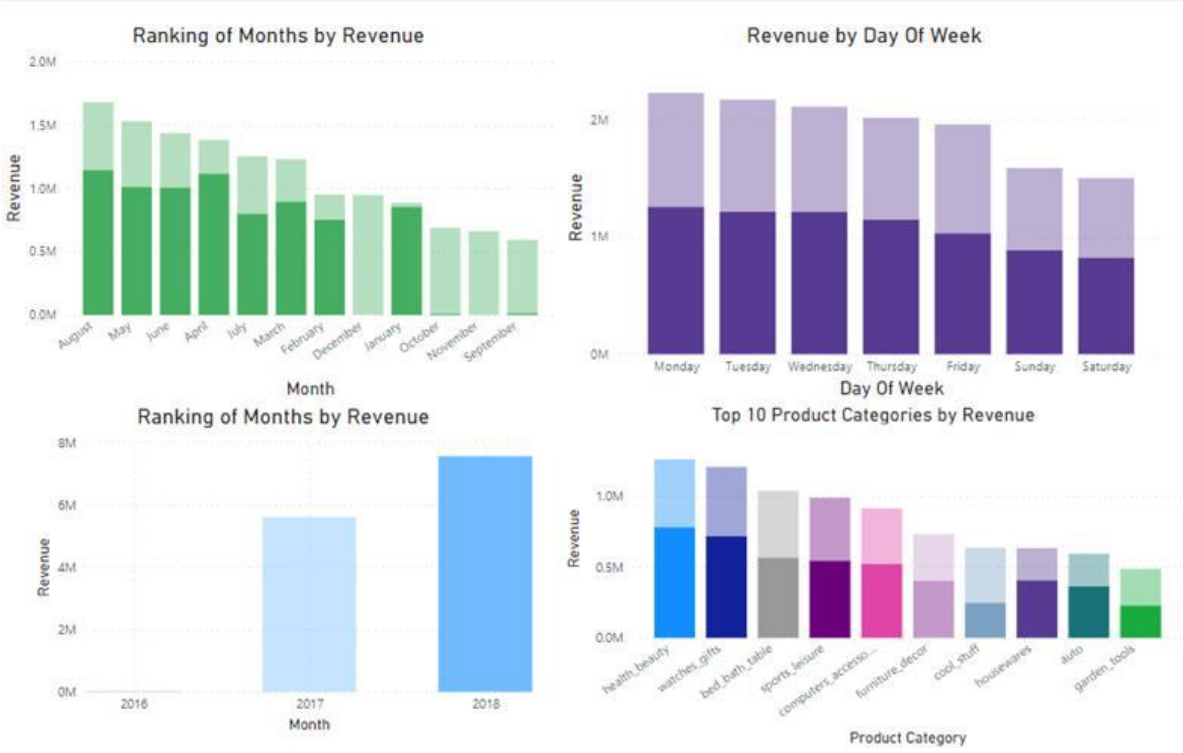


We connected this olap server with Power BI in order to produce various reports and dashboards. We also used the browser view of the cube to open a connection with excel and export there all the data of the warehouse in order to use them for data mining with python.

6. Reports/ Dashboards

1. Charts about revenues. We see that 2018 is the year with the most revenues. August was the month with highest revenue and September the month with the least throughout the years. Monday was the day with the highest revenue and the weekend had the lowest sales. The categories that were sold the most were health and beauty, watches gifts and bed and bath products, so we assume that the buyers are mostly women.

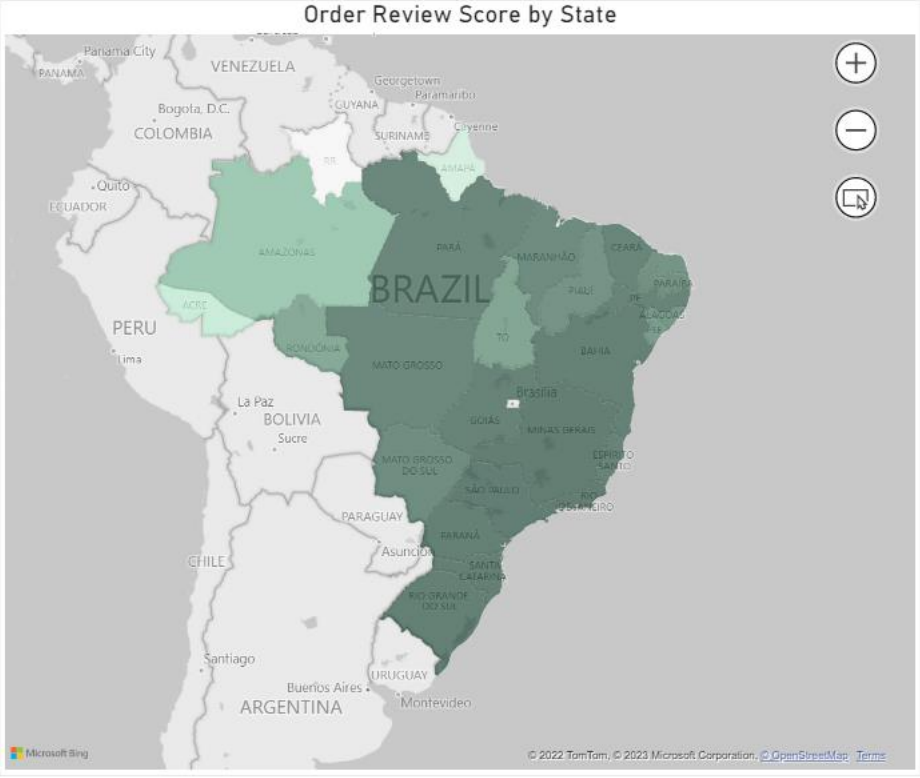
2. We filtered the above report for the year 2018. We observed that the revenue for the last quarter of the year is the lowest.



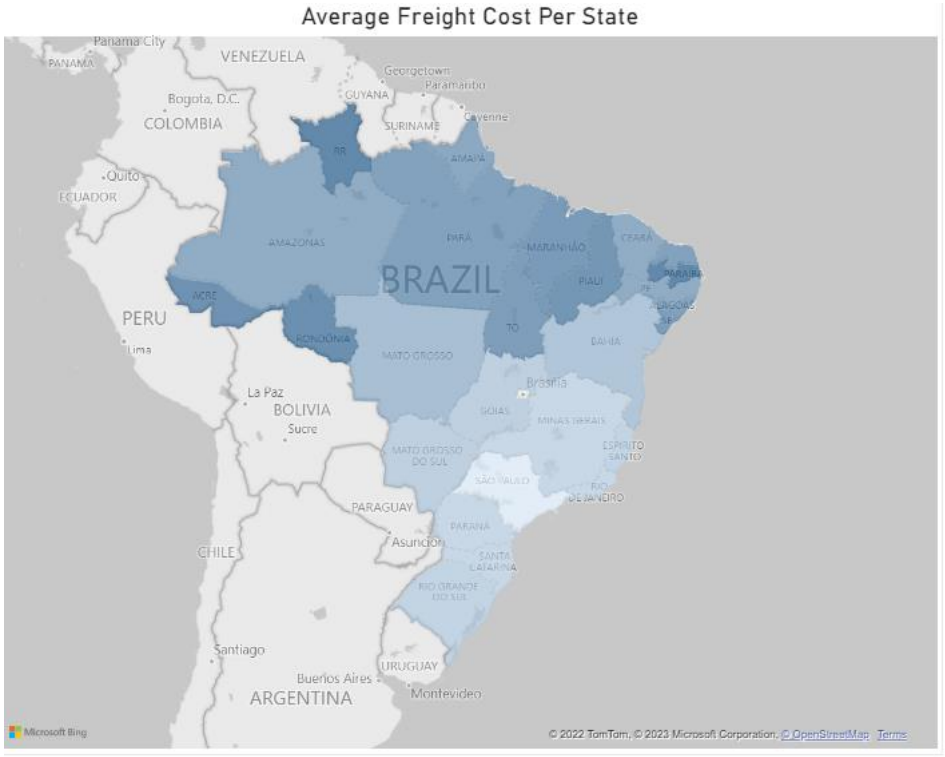
3. In this report we examine the days needed for delivery each month and the delivery error of estimation for each month. We observed that during the summer the

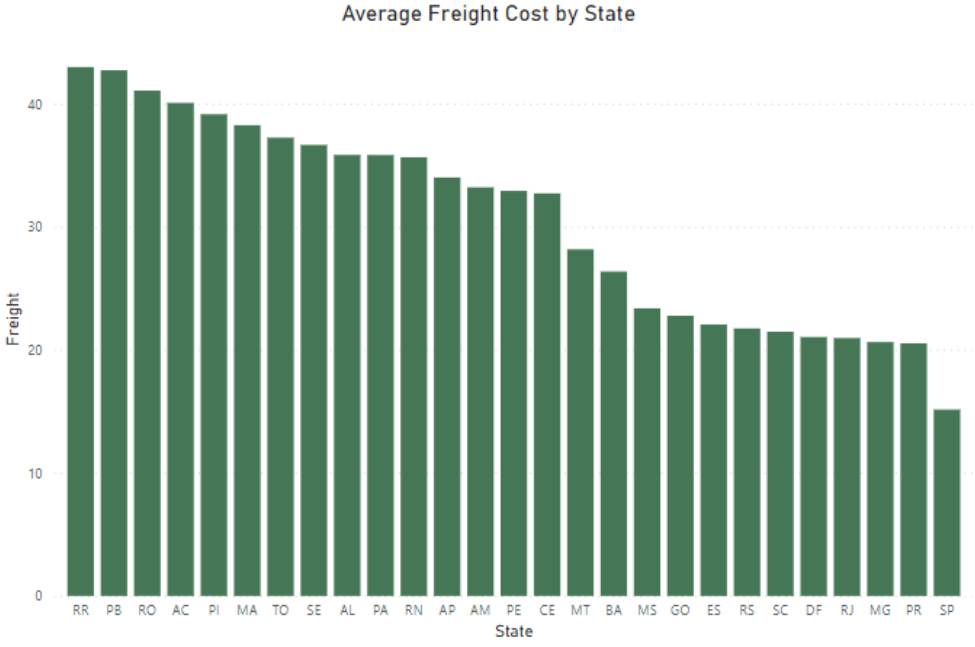
absolute estimation error is the highest but the time needed for delivery is the lowest.

4. This report shows the satisfaction rate of the customers. We notice that the customers that live on the cost lines are the most satisfied.



5. The following map and bar chart show the average freight cost per orders in the states of Brazil. The highest cost are in the north of Brazil and the lowest are in the coast cities. That is expected because in the cost line are the biggest cities and in the north the states are rural. It is logical for the distribution cost of products to be lower in the big cities.





7. Data Mining

a. Sellers Classification

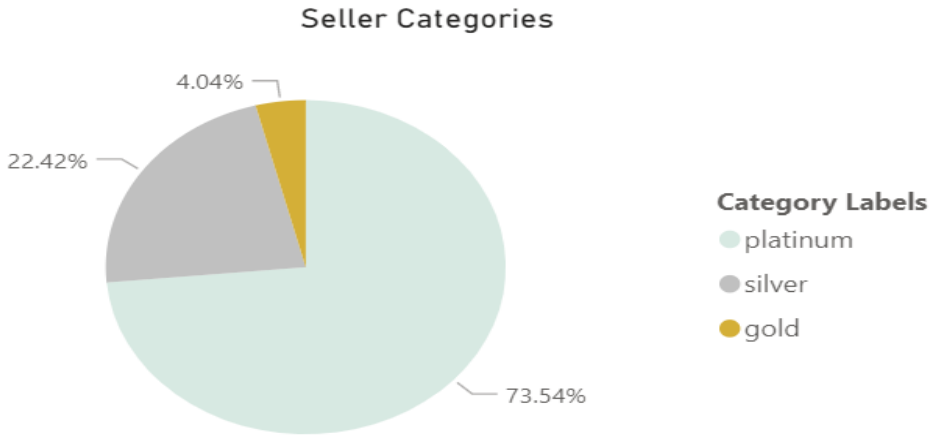
We wanted to differentiate the sellers to see which ones are more valuable to the Company. So we came up with a plan to categorize them into three categorize silver, gold and platinum. By doing that we hope to help the company figure out which sellers are important and must keep because they bring the most revenue. There are a few steps we followed to do this task. Firstly, in order to preform the classification we had to label the first 600 sellers by hand depending on three parameters 1) the number of customers of each seller, 2) the revenue of each seller and 3) the number of products of each seller. After taking into consideration all the three above mentioned parameters, we filled the ranking of each seller from the 1st one to the 600th one. By doing this we had a set of sellers that we could use to train and test the prediction model we were about to make. After doing this we ran some metrics to see if the prediction model we made was good.

The classification report for the model:

	Precision	**Recall**
Silver	0.67	0.67
Gold	1.00	0.60
Platinum	0.91	0.98

From the classification report we conclude that it was good enough to preform this kind of classification we wanted. Then we ran the prediction to the rest of the data and created the labels needed to categorize the sellers.

The distribution of the categories is shown in the following chart:



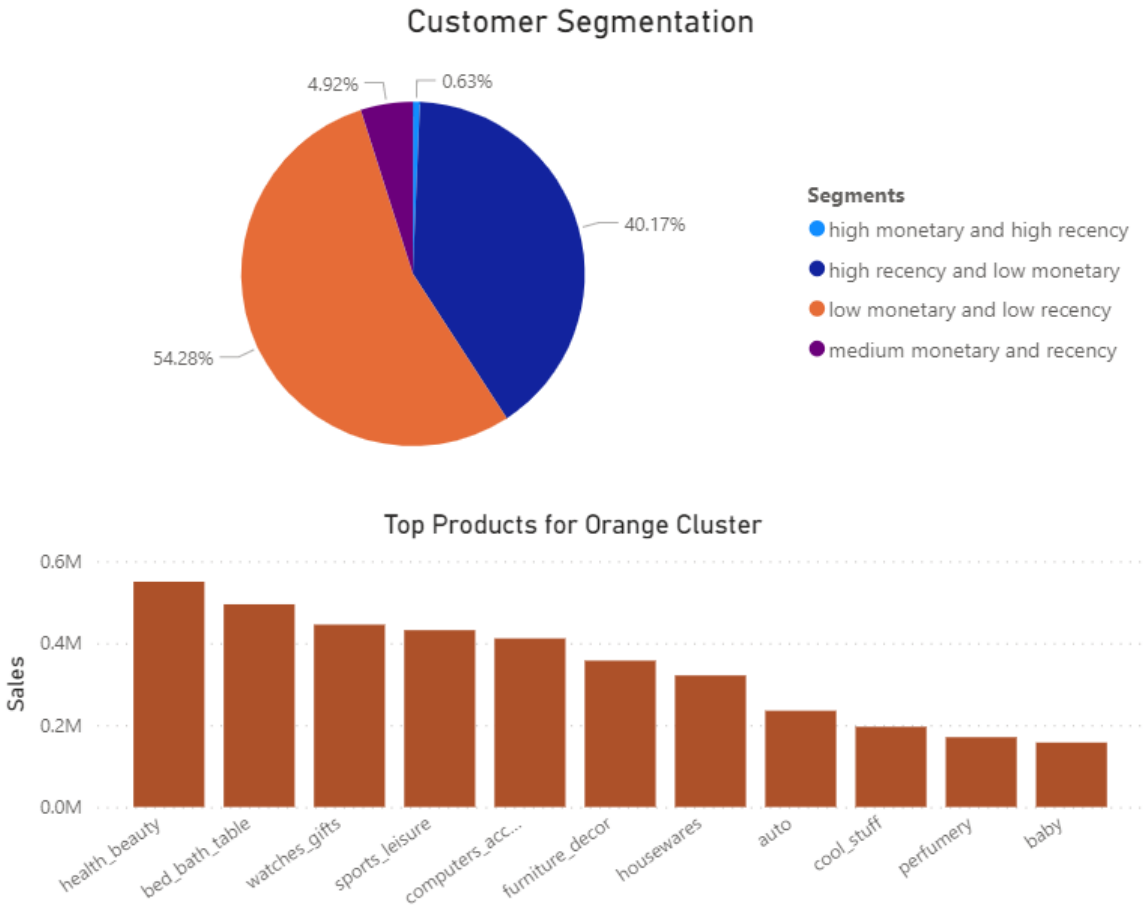
b. Clustering

We performed clustering on the customers based on the RFM- Recency Frequency Monetary model. Grouping clients with similar behavior can help to orientate commercial and marketing efforts to each group. We used the sum of money that each customer spend on products as monetary, the difference of the date of purchase from today as recency and the number of orders that each one made as the frequency. We used the sklearn python lib for the algorithms. Firstly, we scaled the data and ran the elbow method to determine which would be the best number of clusters and found that 4 was the optimal number. We used the K Means algorithm to perform the clustering with k set to 4. Our results were the following:

cluster_kmeans	mean_frequency	mean_recency	mean_monetary	cluster_size
0	1	1726	101	53552
1	1	1987	102	39637
2	1	1842	1879	623
3	1	1830	617	4853

Interpretation of Clusters :

cluster 0: low monetary and low recency
cluster 1: high recency and low monetary
cluster 2: high monetary and high recency
cluster 3: medium monetary and recency



In this powerbi report we have a pie chart that shows the distribution of the customers in the segments. The orange segment has the largest population therefore we are interested in examining its behaviour. We created a bar chart that shows only for the orange cluster the categories of the products that had high revenue. This chart shows that the category of baby products has higher revenue in this segment compared to the general population of customers. We conclude that the customers of this segment have a connection to children and therefore in order to increase their spending Olist should promote sellers with family products in general.