Technical Report for Brazilian E-Commerce Public Dataset Analysis

DA-NAT4 - Group 2

This report has been compiled by:
Alice Venables
Dawit Atreso
Abdulrahman Rajjoub
Hamza Ibrahim

Introduction

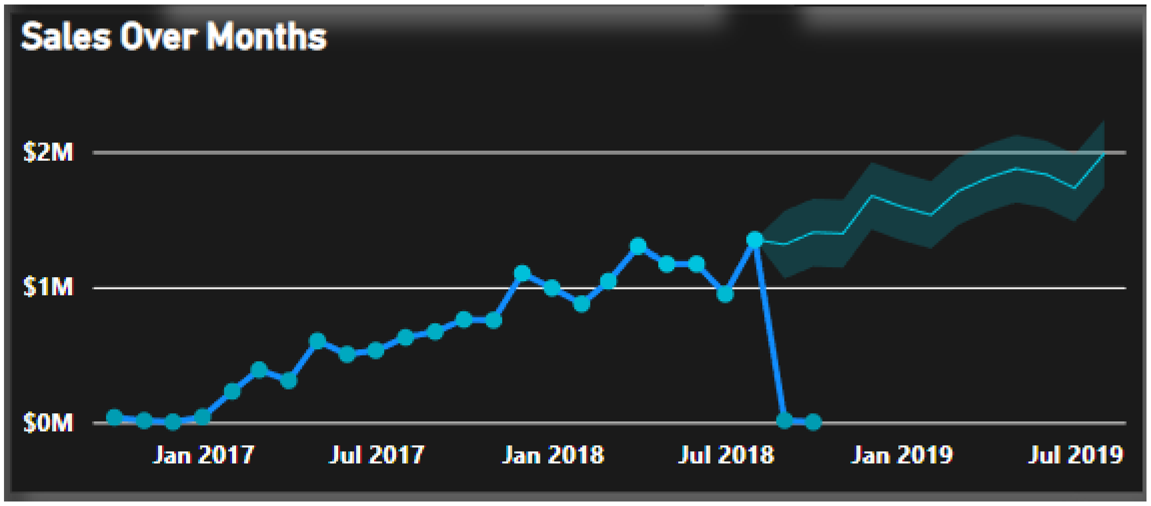
The purpose of this technical report is to provide in depth analysis of the dataset provided, and explanation for the code utilised in order to perform this analysis. The dataset is provided by Olist through Kaggle, and is publicly accessible. It features data from sellers from across Brazil, who utilise drop shipping to generate their profits.

The base dataset features a series of CSV files that allow the viewing of an order from a multitude of dimensions. These include the details of the order, anonymised customer information, details regarding payments, the reviews from the orders placed - with the reviews themselves being in Portuguese - and information regarding the types of items sold.

As this is a complex, multifaceted dataset, this data has been analysed from a multitude of angles. The primary ones that this report focuses on are sales over time and by item category, delivery performance, reviews and semantic analysis, and customer analysis. All of these are analysed with a lens towards improvement of the profits, along with the growth and longevity of the Olist company and the sellers who use its services.

Sales

The sales across multiple categories have been calculated primarily using the order payments dataset, specifically utilising the column “payment_value” in order to determine the amount that the sale was made for. This has then been utilised alongside the orders dataset in order to determine the date of the order, and alongside the product category name translation dataset in order to retrieve the english names for the products. From this, the following graphs were able to be generated, using the mentioned combinations of datasets respectively.

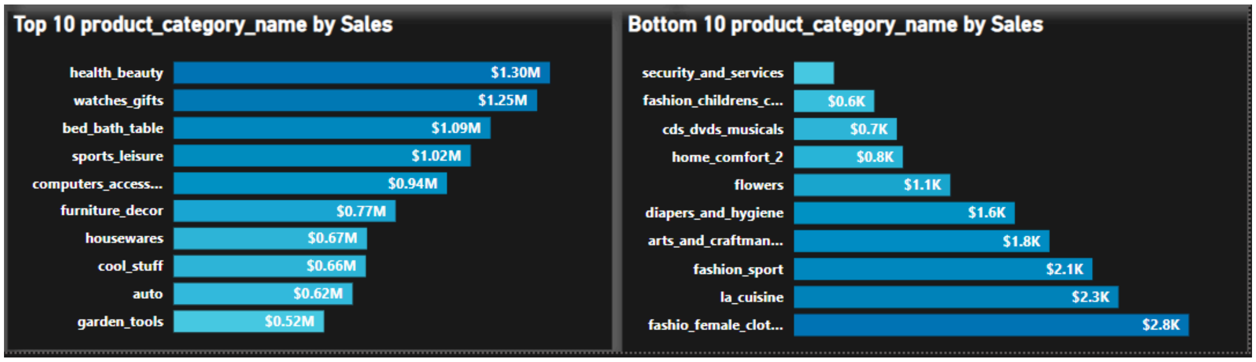


This first graph displays the sales over time, and utilises the forecast analysis function in Power BI in order to predict future sales trends. There are a number of notable insights from this graph, though perhaps the most notable is the seemingly anomalous data at the end of 2018. We felt it was pertinent to include this in the chart itself, as these are sales that have been reported - however, these were ignored for the purpose of forecast creation. Instead, we worked with the non-anomalous data, which allowed us to create the analysis. This clearly shows that the sales will increase over time, and on average will hit around two million dollars per month in sales in August 2019.

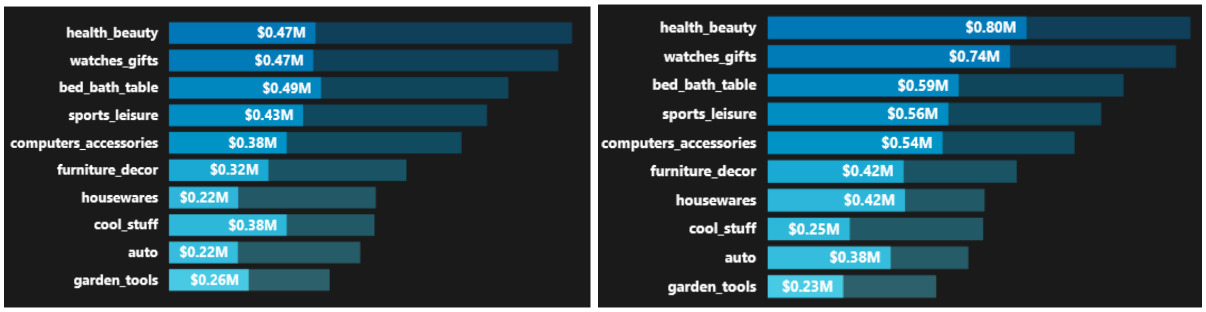
We also see that there are notable dips yearly in both January and July, from which sales often bounce back - however, these dips do eat into profits, and are a potential area for investigation. January is to be expected, as it is just after Christmas, and sales across all industries tend to see a decrease in this month - especially since people tend to be paid at the end of the month. However, July is not within the usual trends across the industry, and warrants further insights.

Potential explanations for this do include sellers taking holidays during this time, especially if they have children, as this is around the time at which schools break up for the summer holidays - as such, sellers may not be as active during this time. Regardless, this is an area that consistently eats into our sales, and as such it is pertinent to investigate and address, if possible.

Next, we will analyse the top and bottom products in terms of sales.



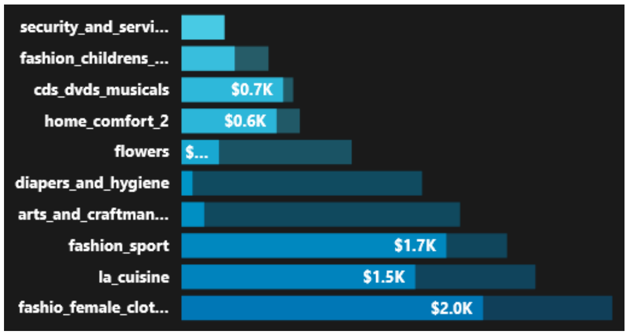
From these charts, we can see some very interesting trends. Health and beauty, along with watches and gifts, tend to be our best selling categories. Some amount of this can be attributed to these being rather large categories, however far more is likely due to customers having greater interest in buying these items. However, we would be remiss to talk about how these figures without also addressing how they have changed from 2017 to 2018.

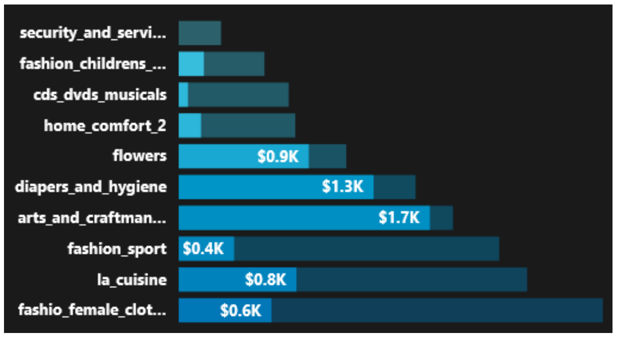


Left: 2017   Right: 2018

As these charts demonstrate, both health and beauty and watches and gifts have shown solid growth over these years. However, bed, bath and table has remained stable without much growth, and the cool stuff category has shown a decrease in value, despite being within the top 10. We also see that auto has shown a monumental amount of growth, as has housewares, indicating that these areas are potentially significant places for investment for further company growth.

In terms of the bottom 10 items, we see that clothes based areas tend to sell poorly, with both female clothes and sports clothes ranking within the bottom ten, and children's clothes being the second worst selling item in general. CDs, DVDs and musicals are also scoring rather low, which isn't all that surprising, given that streaming and on demand services are becoming increasingly popular, driving physical media out of the market. As such, it feels pertinent to cut back funding in these areas, to instead refocus it into other areas. However, it is also important to once again examine how the sales have changed from 2017 to 2018.





Left: 2017   Right: 2018

Here we can once again see a dramatic shift from 2017 to 2018, especially in terms of the sales of some of the categories we called out before. Fashion sales have dropped dramatically, and physical media seems to have been entirely driven out of the market. Food and cooking is another area that has experienced a dramatic drop in sales compared to previous years, potentially indicating that it is being driven out of the market as well.

However, we also see some rather surprising insights. Arts and crafts has seen a huge boom in terms of sales from 2017 to 2018, indicating a potentially emerging market to invest more into, and support. We see a similar thing in flowers, as well as diapers and hygiene, indicating that more and more people are turning to Olist in order to purchase these items. Further research into more effective shipping methods for these could be pertinent, especially given that flowers tend to be rather fragile, and could become damaged in shipping - this could indicate a market that could be cornered and entirely capitalised by Olist, allowing for large amounts of profit to be generated by potentially removing any competitors before they can get onto the scene.

Delivery Performance

To analyse delivery performance, the orders dataset and the customers dataset has been used. Deliveries have been broken down into a number of different categories - early, on time, and delayed deliveries. Furthermore, the average amount of time each delivery takes to ship has been calculated via the use of DAX, along with the amount of successful deliveries that have been made. DAX was used for this processing as it rendered it possible to do it in Power BI, and allowed for easier transforming of the data. Finally, DAX was used to determine if an order arrived on time, early, or after the estimated delivery time. The code for this, along with explanations of what each part does, is below.

```
DelayedDeliveries =
COUNTROWS(
        FILTER(
                Olist_orders_dataset,
                NOT(ISBLANK(olist_orders_dataset[order_delivered_customer_date]))
&&
                olist_orders_dataset[order_delivered_customer_date] >
olist_orders_dataset[order_estimated_delivery_date]
        )
)
```

This first DAX function counts the number of orders delivered after the delivery date.
Next, we work out the number of orders with the status ‘Delivered’.

```
DeliveredCount =
COUNTROWS(
        FILTER
                olist_orders_dataset,
                LOWER(TRIM(olist_orders_dataset[order_status])) = "delivered"
        )
)
```

With this calculated, we now need to define our annual delivery target and our monthly delivery target, in order to create KPIs.

```
Annual Delivery Target = 33333
Monthly Target = 2777
```

These functions are created by taking the count of total deliveries, and then dividing them by either the total number of years for the annual delivery target - in this case, three - or by twelve for the monthly target. The end result allows us to create KPIs, which we will later be using to analyse deliveries and search for areas for improvement.

Next, we calculate the amount of successful deliveries, once again using a DAX function.

```
DeliverySuccessRate =
DIVIDE(
        COUNTROWS(
                FILTER(
                        olist_orders_dataset,
                        olist_orders_dataset[order_status] = "delivered"
                )
        ),
        COUNTROWS(olist_orders_dataset),
        0
)
```

This calculates the ratio of delivered orders, by dividing the amount of delivered orders by the total order count. We have also built a function into the DAX code to make it so that if the denominator is zero, the result will be zero, rather than an error.

```
EarlyDeliveries =
COUNTROWS(
        FILTER(
                olist_orders_dataset,NOT(ISBLANK(olist_orders_dataset[order_delivered
                _customer_date])) &&
                olist_orders_dataset[order_delivered_customer_date] <
                olist_orders_dataset[order_estimated_delivery_date]
        ))
```

This code filters for the rows with a delivery date earlier than the estimated delivery date, and then provides a count for those rows, allowing us to work out how many were delivered earlier than expected. The next piece of code performs a similar function, but instead calculates for rows that arrived before or on the estimated delivery date, allowing us to see all deliveries that were not late.

```
OnTimeDeliveries =
COUNTROWS(
        FILTER(
                olist_orders_dataset,NOT(ISBLANK(olist_orders_dataset[order_delivered
                _customer_date])) &&
                olist_orders_dataset[order_delivered_customer_date] <=
                olist_orders_dataset[order_estimated_delivery_date]
        ))
```

With this calculated, we are then able to work out the proportion of deliveries that weren't late, using a similar piece of code as to previous calculations. Once again, we set the count to zero if it attempts to divide by zero.

Next, we calculate the total number of orders made.

```
Total order = COUNT(olist_orders_dataset[order_id])
```

Here we simply use a count function in order to calculate the amount of orders made, and given the order_id column is an index column, we know all the values will be distinct. This gives us our total amount of orders, which is useful for calculations later.

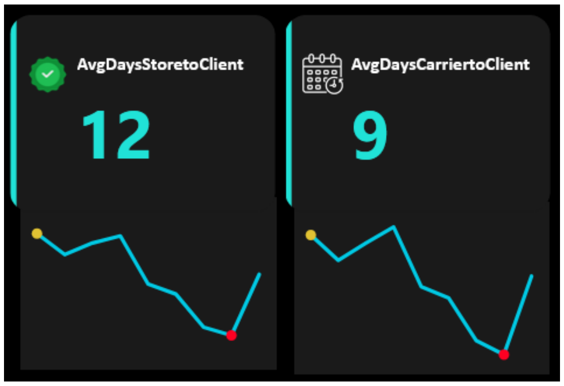
Finally, we calculate the amount of time it takes for an order to be delivered to a customer from when they place the order, and from when the carrier receives it. The codes below calculate each one respectively.

```
AvgDaysStoretoClient = AVERAGEX(olist_orders_dataset,
DATEDIFF(olist_orders_dataset[order_purchase_timestamp],olist_orders_dataset[order_
delivered_customer_date],DAY))
```

```
AvgDaysCarriertoClient = AVERAGEX(olist_orders_dataset,
DATEDIFF(olist_orders_dataset[order_delivered_carrier_date],olist_orders_dataset[orde
r_delivered_customer_date],DAY))
```

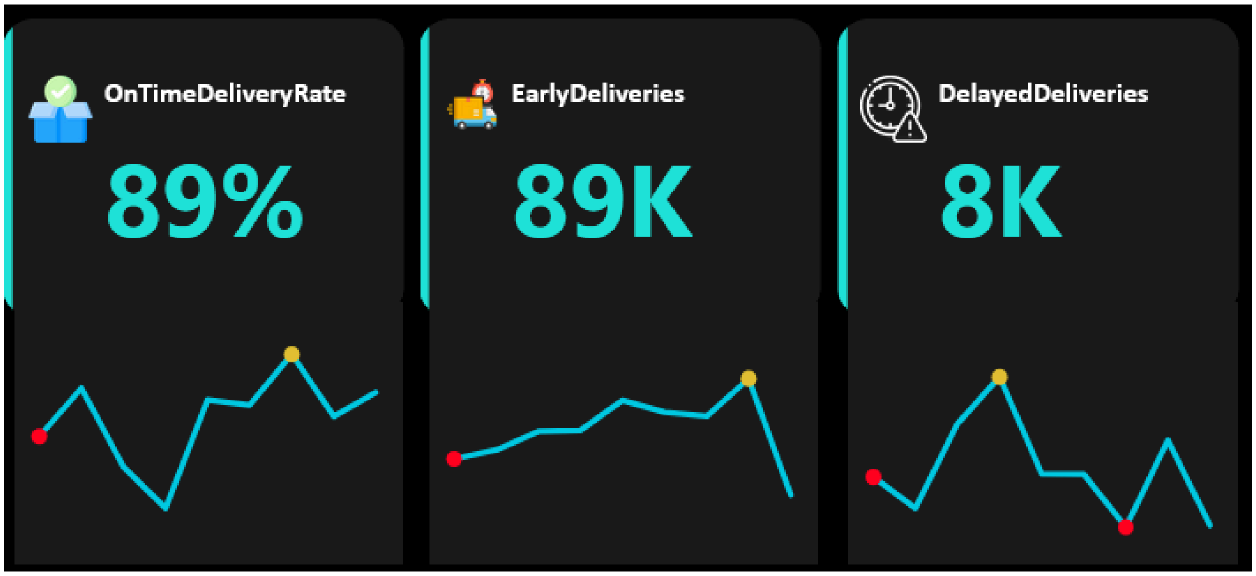
For both of these pieces of code, we utilise the DATEDIFF function in order to calculate the difference between two datetime values, and then we calculate the average of this difference in order to find the average amount of time each one takes.

With this calculated, we are now able to create our visuals, and perform our analysis.



This is the first visual we are going to utilise in our analysis. This visual uses the DAX code outlined above in order to calculate the average amount of time it takes for a delivery to go from the store to the client, and from the carrier to the client. Immediately, two major insights are clear. Firstly, a vast majority of the delivery time is occupied in the 'carrier to client' phase. This could very strongly indicate that delayed deliveries are largely in the hands of the carriers - a notion further reinforced by the fact that the time from store to carrier is a consistent three days.

Secondly, it becomes immediately clear that the summer months are far, far more reliable in terms of carriers and average delivery times overall, with deliveries taking far less time. There are a number of possible reasons for this, but with the largest dip being in August, with the rise beginning again as September begins, it would strongly indicate that the summer weather tends to be a reason - as September tends to bring with it colder weather. It could also be that less people are ordering - as we saw previously, around July orders are far lower, so it is possible that carrier services are less overwhelmed. This is an area for further investigation, but if carrier services are overwhelmed, we could see an increase in profits by further investing into more carrier services.



Next, we are going to look at the rates of deliveries by month. Here we see a comforting statistic - on time deliveries are generally high, resting at just below 90% - not quite at the ideal levels of above 90%, but significant growth given the two years of operation. Furthermore, we once again see that over time, the early deliveries tend to be significantly higher, with the winter months being the major outlier here. However, this data is likely skewed by the fact that in 2016 Olist was only operating during the winter months, resulting in overall lower than expected average deliveries - especially when paired with the lack of operation in the winter of 2018 thus far.

Overall, the deliveries indicate a consistent level of growth throughout the company, and indicate that the company is at healthy levels. Investigation into if the carrier services are overwhelmed with the increased amount of orders is the only major line of advice - otherwise, deliveries are progressing comfortably, according to this analysis.

Reviews and Semantic Analysis

In order to perform analysis of the reviews, and the overall sentiments present in these reviews, we utilised a mix of python and google sheets, alongside the order reviews dataset, and two new datasets that were created via python - english reviews, and RFM Scores. The first part that we will touch upon is google sheets.

As google sheets is owned and operated by google, we were able to upload the order reviews CSV onto the site, and then make use of it’s native translation function in order to translate the review text to english. In order to do this, a new column was created on the spreadsheet, titled “English Reviews”. Into the column beneath this, in cell H2, the following formula was entered.

```
=GOOGLETRANSLATE(E:E,"pt","en")
```

This code was then applied vertically to all cells below in that column, making use of google sheet’s native GOOGLETRANSLATE function to translate the reviews from Portuguese to English. With this done, the CSV was then redownloaded, and analysis of the reviews, and the sentiments within, was able to begin.

The following code is what was used to prepare the reviews for the semantic analysis.

First, we import the relevant libraries in order to perform the analysis of the text. We utilise the stopwords module from nltk.corpus in order to filter out the most common stopwords in the english language. Next, we import pandas in order to read the CSV and load it into the dataframe. We are then also going to use re in order to remove emojis and other special characters in order to make the word tokenisation work better, and allow the subsequent visualisations to function better - this will also allow the stopwords to catch words which would otherwise be missed, due to special characters. Finally, we use word_tokenize from nltk.tokenize in order to split the words themselves, which will allow the visualisation to work better.

```python
from nltk.corpus import stopwords
import pandas as pd
import re
from nltk.tokenize import word_tokenize
```

Next we set up the filters that we are going to utilise to remove certain key words from the reviews, in order to allow the visualisations to work better. Our first filter will be the stopwords we imported earlier, whilst the second is a list of common words that arise in the reviews that aren’t relevant to the actual analysis we’re going to perform.

```python
filler = stopwords.words('english')
filler1 = ['product', 'delivery', 'delivered', 'arrived']
```

Next, we import the dataframe itself utilising pandas, and view the head of the dataframe in order to ensure the data has imported correctly. With this done, we now move onto the next step.

```
dataframe = pd.read_csv('reviews_order_dataset_trans.csv')
dataframe.head()
```

Here, we drop all of the rows that have #VALUE! in any of their columns. We do this because the google sheets translate function puts #VALUE! when attempting to translate empty cells, and including these in the analysis would cause further issues - especially given that these rows didn’t have any review in the first place.

```
dataframe.drop(dataframe[dataframe['Review Comment English'] ==
'#VALUE!'].index, inplace=True)
```

Next, we are going to define the code to remove emojis and other special characters, by using the re function we imported earlier. re is able to take input for certain unicode characters, which allow us to filter out all of these characters for future analysis.
Then, with this function defined, we simply apply it across all the rows of our dataframe, stripping any emojis, symbols, and other special characters from the reviews.

```
def emoji_gone(string):
    emoji = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols &
pictographs
                           u"\U0001F680-\U0001F6FF"  # transport and
map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji.sub(r'', string)

dataframe['Review Comment English'] = dataframe['Review Comment
English'].apply(lambda text: emoji_gone(str(text)))
```

Now with the emojis and other special characters removed, we can begin to clean the text up further, once again using re. First, we replace any characters that aren’t from the latin alphanumeric alphabet, and then we convert all of the text into lowercase. Finally, we tokenise the words using word_tokenize, and finally filter the text to ensure that it removes any of the words that are present in either the premade stoplist or the custom stoplist we created. Finally, we apply this new code to the dataframe, resulting in a cleaned up and tokenised dataset.

```
def textprep(text):
  text = re.sub('[\W ]+',' ',text)
  text = text.lower()
  words = word_tokenize(text)
  return " ".join([word for word in words if word not in filler and
word not in filler1])
dataframe['Review Comment English'] = dataframe['Review Comment
English'].apply(textprep)
```

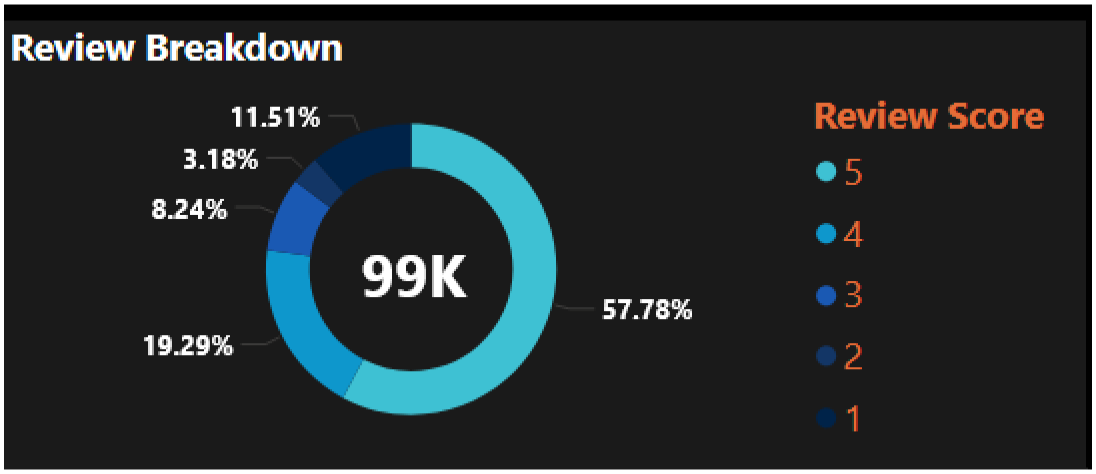
```
dataframe
```

Next we display the dataframe to ensure that all of the prior work has gone through smoothly, and provided it has, we then export this to a new CSV, which we can add to Power BI.

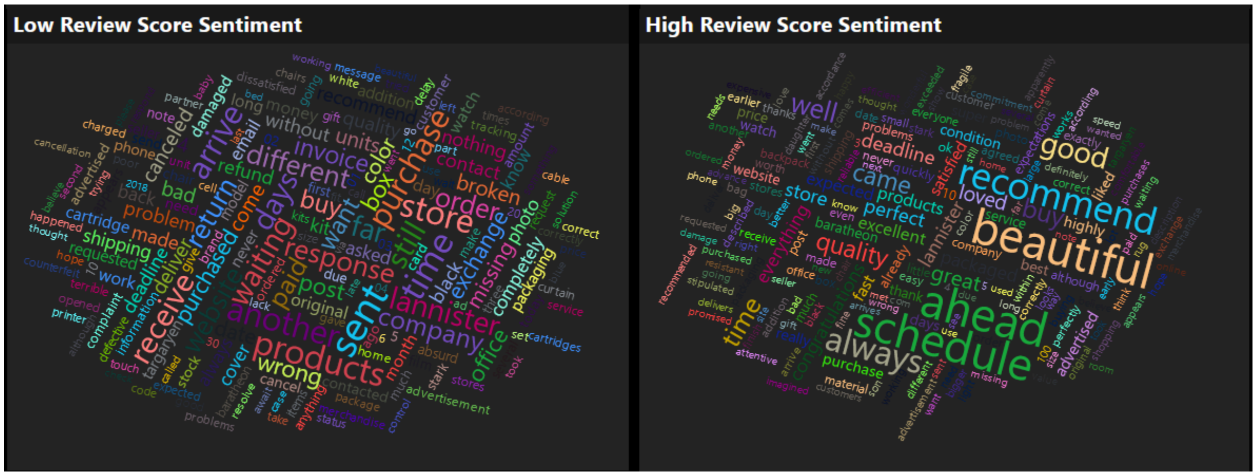
```
dataframe.to_csv('review_english.csv')
```

Exporting it with the .to_csv method is simple and effective, and allows us to easily upload it onto Power BI.

With this preparation for the reviews performed, we can now begin to analyse the reviews in full, and see what insights they can bring.



First, we analyse the general review score, out of our total reviews. Out of nearly one hundred thousand reviews, 57.78% were at five stars, and less than 20% of them were at two stars or lower. However, of these lower review scores, 11.51% were 1 star reviews, strongly indicating that when people were upset, they were very upset indeed. Still, this strongly indicates that overall our customers are happy with our services, and that we are performing well. In terms of what customers are saying we did well and what we need to improve upon, we created word clouds using the tokenised reviews in order to provide clearer analysis.



Here we have the review sentiment word clouds for both high (4 and above) and low (2 and below) review scores. Our first focus will be on the high review score sentiments to see where we are doing well, and then we will shift our focus to the low review score sentiment to see where we can improve.

In the high review score sentiment, we can see that people often mention that things came as advertised, that they recommend our services, that the products are of high quality and beautiful. However, even more prominently than this, we see that the products often come ahead of schedule, with words such as earlier and time also jumping out, indicating that - as we saw before - our delivery speed is one of our strong points.

In terms of our lower review score sentiment, however, a number of things jump out. First and foremost is that people seem to be receiving the wrong products, or that the products they do receive are broken or otherwise damaged. Furthermore, there are complaints about customers waiting, and a lack of response from our team, which are very much potential issue zones.

The other major thing that appears in these word clouds is the name of one of our sellers - Lannister. Lannister has consistently appeared in low scoring reviews, and seems to be a consistent issue for our customers. This seems like a major area in which we could investigate further and improve, potentially cutting off these sellers if they can't improve their services, as they are consistently letting our customers down.

Customer Analysis

For the customer analysis, we utilise the order items dataset, the geolocation dataset, and the customer and seller datasets to perform geolocational analysis on our customer and seller locations, as well as the order payments and orders dataset, in order to perform RFM analysis upon our customers.

First, we are going to utilise python code to perform the RFM analysis. For this, we are going to use pandas to read the CSVs and perform transformations upon the resulting dataframes, datetime to analyse the recency of purchases.

```
# importing necessary libraries
import pandas as pd
import datetime as dt
```

With the relevant modules imported, we then import the dataframes we need, and view the heads of these dataframes in order to ensure they have been imported smoothly.

```
# importing the data
dfprice = pd.read_csv('olist_order_payments_dataset.csv')
# We then check the columns present, and make sure the dataframe is
correct
dfprice.head()
```

```
# We repeat this process for the other dataset we need as well
dforder = pd.read_csv('olist_orders_dataset.csv')
dforder.head()
```

Once we have the two datasets imported, we then merge the dataframes on the column order_id, which will allow us to better perform the RFM analysis we need to. As usual, we view the head of the dataframe to ensure that the merging has gone smoothly.

```
# as the two datasets we needed weren't previously merged, we have
to merge them together into one dataset
dfmerged = dforder.merge(dfprice[['order_id', 'payment_value']],
on='order_id')
dfmerged.head()
```

Next, we enforce the datatime format onto the merged dataframe, which will allow us to accurately calculate the recency score for RFM - without doing this, the code will result in an error when trying to calculate this value.

```
#We then enforce date-time on the merged dataframe, to make sure
that future calculations are performed correctly
dfmerged['order_purchase_timestamp'] =
pd.to_datetime(dfmerged['order_purchase_timestamp'])
```

Next, we are going to perform the calculation to find the recency of a given order. The following code performs this calculation, first creating a new dataframe for recency, which will group the previous dataframe by customer_id. With this done, it will assign each one their latest order purchase timestamp, and assign these columns the titles “CustomerID”, and “LastPurchaseDate”.

With this done, we create the variable recent_date, which will simply take the latest date from our LastPurchaseDate column, and store it as the value. Finally, we create a new column called ‘Recency’, and populate this column with the difference between the most recent purchase date and the values from LastPurchaseDate. By doing this, we are able to calculate a value for how recently any given customer placed an order.

```
# Now we can begin to calculate the recency
# we first group all the customer IDs, and grab the latest order
purchase timestamp
df_recency = dfmerged.groupby(by='customer_id',
as_index=False)['order_purchase_timestamp'].max()

# then we rename the columns to be something more appropriate
df_recency.columns = ['CustomerID', 'LastPurchaseDate']

# We grab the most recent data from the LastPurchaseDate in order to
help calculate our recency score (rather than using the current
date)
recent_date = df_recency['LastPurchaseDate'].max()

# finally, we calculate the amount of days between the most recent
date and the date of the purchase
df_recency['Recency'] = df_recency['LastPurchaseDate'].apply(lambda
x: (recent_date - x).days)

# we can print the head of the dataframe to make sure all is well
df_recency.head()
```

Now, we are able to calculate the frequency of the customers orders. This is a simply matter of taking the customer ID, and counting the number of purchases made by that customer ID. We are once again creating a new dataframe, this time called frequency_df, in order to store this data for later merging.

```
# here we calculate the frequency of the orders

# we calculate it by grabbing the customer ID, and counting the
number of order purchases made by that customer
frequency_df = dfmerged.groupby(by=['customer_id'],
as_index=False)['order_purchase_timestamp'].count()

# then we rename the column titles as usual
frequency_df.columns = ['CustomerID', 'Frequency']
frequency_df.head()
```

Finally, we have to calculate how much each customer is spending. This is once again a simple process, where-in we calculate the sum of the customer's payment value, grouped by their customer ID. With this done, we load it into another dataframe, this time called monetary_df, which we will also be using in our RFM analysis.

```
# finally, we have to calculate how much they are spending

# here we're going to simply calculate the sum of the payment values
per customer
monetary_df = dfmerged.groupby(by='customer_id',
as_index=False)['payment_value'].sum()

# Once again, change the column titles
monetary_df.columns = ['CustomerID', 'Monetary']
monetary_df.head()
```

Now we are going to merge the dataframes we have created together, in order to create an RFM dataframe, from which we will be able to calculate each customer's RFM score, their normalised RFM rank, alongside other factors.

```
# Now we can begin to calculate our RFM!

# first we merge recency and frequency
rf_df = df_recency.merge(frequency_df, on='CustomerID')

# then we merge this table to monetary to get our RFM table!
rfm_df = rf_df.merge(monetary_df,
on='CustomerID').drop(columns='LastPurchaseDate')
```

```
rfm_df.head()
```

Here we normalise the data. In order to do so, we assign each customer a rank within the dataframe, creating new columns for each rank, and scoring them against the Recency, Frequency, and Monetary columns. Then, we normalise the ranks, by dividing them by the highest rank, and multiplying it by 100 to calculate it as a percentage. Then, we remove the original rank columns, in order to avoid confusion and wasted space on the dataframe.

```
# Now we normalise the data

# first, we grant each customer a rank within their columns
rfm_df['R_rank'] = rfm_df['Recency'].rank(ascending=False)
rfm_df['F_rank'] = rfm_df['Frequency'].rank(ascending=True)
rfm_df['M_rank'] = rfm_df['Monetary'].rank(ascending=True)

# normalizing the rank of the customers
rfm_df['R_rank_norm'] =
(rfm_df['R_rank']/rfm_df['R_rank'].max())*100
rfm_df['F_rank_norm'] =
(rfm_df['F_rank']/rfm_df['F_rank'].max())*100
rfm_df['M_rank_norm'] =
(rfm_df['F_rank']/rfm_df['M_rank'].max())*100

# finally, we drop the original rank columns, to avoid confusion!

rfm_df.drop(columns=['R_rank', 'F_rank', 'M_rank'], inplace=True)

rfm_df.head()
```

Next, we calculate RFM. For RFM, we use a predefined formula, which uses our normalised ranks in order to grant each of our customers an RFM score. Then we round this score down to two decimal places in order for better storage. Finally, we print the head, to once again ensure that everything has gone smoothly. Since we have applied a complex step, we print ten values for the head rather than five, to better check for issues.

```
# now we calculate rfm

# we use a preset formula for this, using the normalised rankings
rfm_df['RFM_Score'] = 0.15*rfm_df['R_rank_norm']+0.28 * \
  rfm_df['F_rank_norm']+0.57*rfm_df['M_rank_norm']
rfm_df['RFM_Score'] *= 0.05
```

```
# once this is calculated, we round it to two decimal places
rfm_df = rfm_df.round(2)

# and print the head to check all is well!
rfm_df[['CustomerID', 'RFM_Score']].head(10)
```

Finally, we're going to assign each of our customers a category, based on their RFM score. This will allow us to better analyse the customers in Power BI, as it can help us tell how many of our customers are high value, and if we need to work on keeping more customers on board.

```
# Here we're going to assign each of our customers a category, based
on their RFM score.
for index, row in rfm_df.iterrows():
  if row['RFM_Score'] > 3.5:
    rfm_df.at[index, 'Customer_Segment'] = 'Top Customer'
  elif row['RFM_Score'] > 2.5:
    rfm_df.at[index, 'Customer_Segment'] = 'High Value Customer'
  elif row['RFM_Score'] > 1.5:
    rfm_df.at[index, 'Customer_Segment'] = 'Low Value Customer'
  else:
    rfm_df.at[index, 'Customer_Segment'] = 'Lost Customer'
```

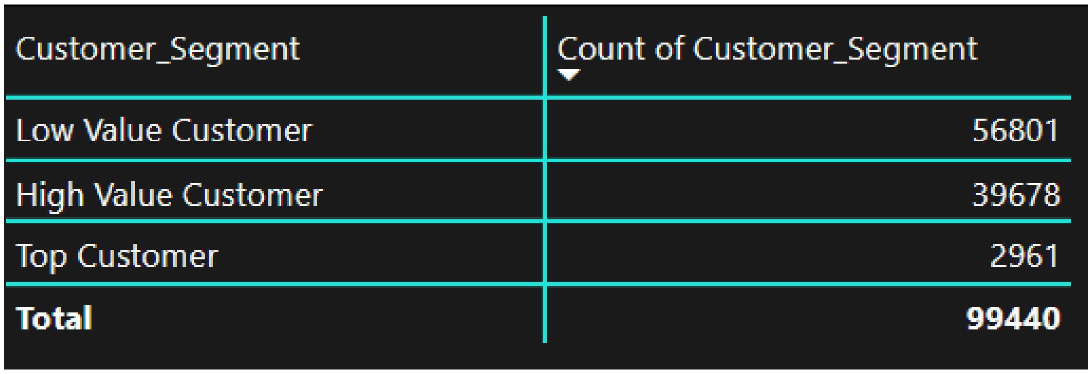
Once more we view the dataframe, in order to ensure everything has gone smoothly before we export it to a CSV.

```
# We display the dataframe just to ensure that everything has gone
smoothly
rfm_df
```

We create a new CSV to store this analysed data, to better be able to import it into Power BI.

```
# Finally, we export it to a CSV in order to load it into Power BI!
rfm_df.to_csv('RFM_Scores')
```

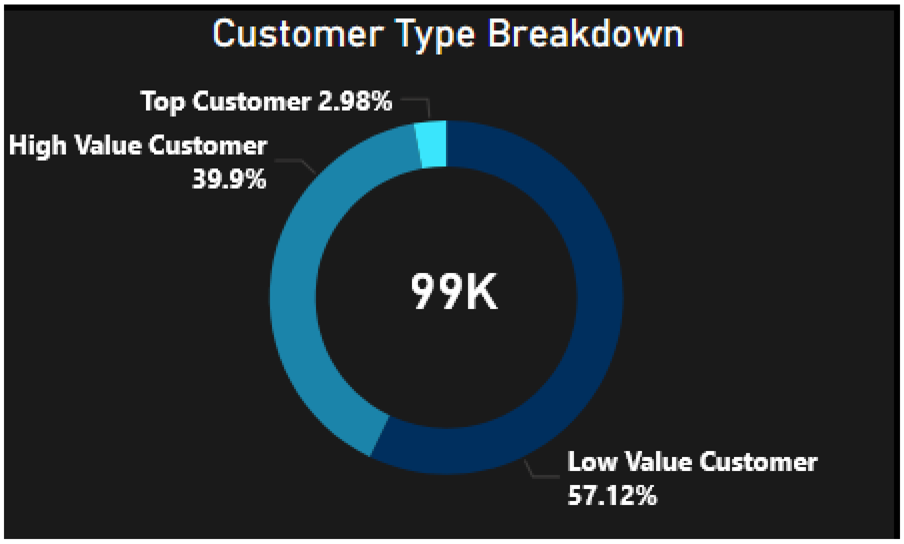
With this new CSV created, we are now able to analyse how many of our customers are top customers, high value customers, low value customers, and lost customers. The following table displays the numbers for our customers.



Customer_Segment	Count of Customer_Segment
Low Value Customer	56801
High Value Customer	39678
Top Customer	2961
Total	**99440**

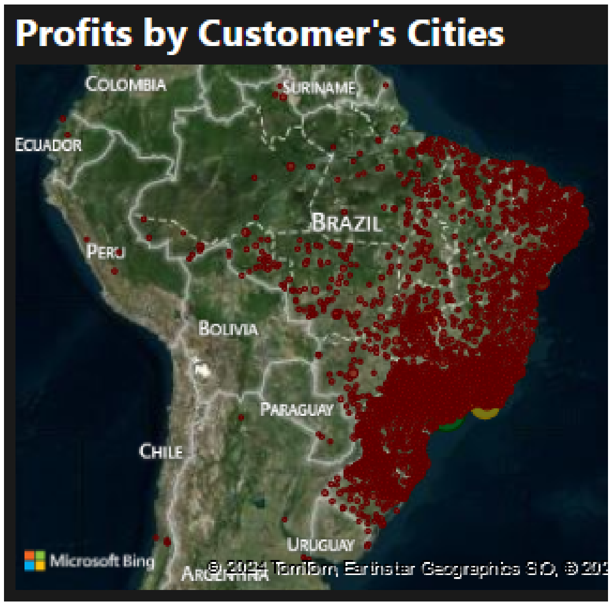
This table gives us some insight into our customer count - it is first immediately clear that we have a high amount of low value customers, with the count of low value customers numbering over 50 thousand. However, we also have a good amount of high value customers, and no customers that are classed as lost - indicating that those who do shop with us have a tendency to return. This is overall a very positive thing, however this high amount of low value customers is a worrying factor. It is likely worth looking into how we can convert these low value customers into high value customers - especially by looking at the reviews, as analysed earlier in this report, to see what customers are upset about.

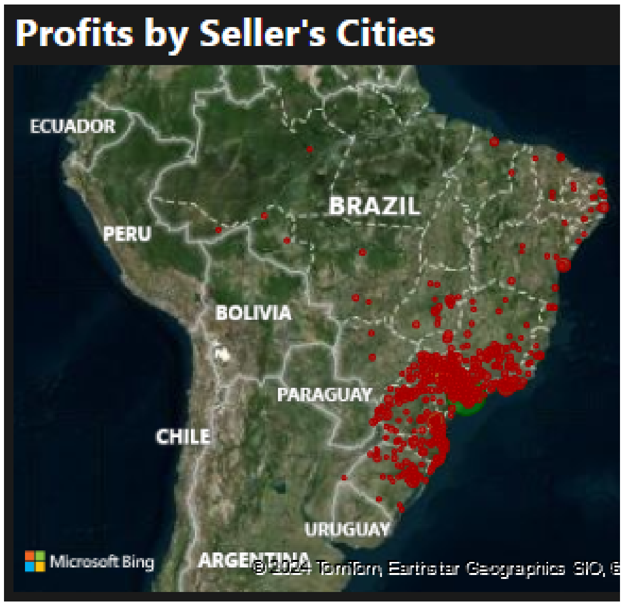
We also have a donut chart which allows us to see the exact percentages of our customer breakdown, as seen below.



This allows us to clearly see that low value customers make up almost 60% of our customer base - whilst this isn't surprising for a drop shipping company, especially one that has started as recently as Olist, this is something to be aware of and work towards improving. However, the 40% high value customers and the complete lack of lost customers is an incredibly encouraging sign, as it very strongly indicates that customers who do come tend to continue to order, even if it is infrequently, or for low monetary amounts.

Next, we have an analysis of sellers and customers by city, which provides fascinating insights into large potential areas for improvement within sales and shipping - especially important, given that shipping was one of the major factors that was reported in the low rank semantic analysis.




Profits by Seller's Cities
ECUADOR
BRAZIL
PERU
BOLIVIA
PARAGUAY
CHILE
URUGUAY
ARGENTINA
Microsoft Bing
© 2024 TomTom, Earthstar Geographics SIO, ©

Here we can clearly see maps that display the customer and seller locations, with bubble size dictated by the profits generated from given cities. From these two maps, we can see there is a dramatic difference between the seller locations and the buyer locations, with a majority of the sellers being located in the south, whilst there is a very, very large seller market in the north east. Further analysis also reveals that there are a number of high value sellers in the central north as well as the north east.

What these charts clearly indicate is a potential area for expansion, as adding more sellers in the north would allow for better logistics in terms of shipping, and better ability to utilise the clear interest that is present in these northern areas. Furthermore, it would help to address key issues brought up by the semantic analysis of the reviews, where in a majority of the complaints were about the delivery times. As such, it feels pertinent to begin expansion into the northern areas of Brazil immediately, as this will allow for far better capitalising upon this market, and allow for vastly increased profits, by making this investment.

Bibliography:

In the course of this analysis, we have utilised the Brazilian E-Commerce Public Dataset by Olist. This data was sourced from Kaggle, who anonymised the data by removing customer names, and replacing any reference to sellers with the names of the Great Houses from Game of Thrones.

This analysis has been performed using a mix of Power BI and Python. The python libraries used will be listed below, alongside the reason for their use.

- nltk.corpus and nltk.tokenize has been used in order to access a library of common stopwords in the english language, and in order to tokenise the words of the review, to split them from each other and allow for the word cloud function in Power BI to better work. NLTK's documentation can be found here.
- pandas has been utilised to read the CSVs provided by Kaggle, as well as load them into dataframes and transform them. It has also been used to write dataframes to CSVs, in order to import them into Power BI. Pandas' documentation can be found here.
- re has been used to use regex to remove emojis and other special characters, including flags. It has then also been used to remove non-latin characters from the reviews, in order to clean them and better apply tokenisation and word cloud analysis. re is a core python library, its documentation can be found here.
- Finally, datetime has been used in order to calculate the difference between certain timestamps using the .days function, and allow us to work with datetime formats in order to calculate values for recency in RFM analysis. datetime is a core python library, its documentation can be found here.