# PROJECT REPORT

## Data Pre-Processing and Visualization

Joana Neves & Mijail Naranjo


By Group E:

Afonso Cadete | 20211519
Daniel Kruk | 20211687
Marcelo Junior | 20211677
Rita Centeno | 20211579

# Index

# Introduction

The company *Il Mercato* extracts data on a daily basis from customers who make purchases on their various business platforms. However, despite having this data, the company cannot take advantage of the data stored in its information systems.

This ends up hurting the company as it does not allow it to maintain a sustainable level of continuous growth and does not bolster the business toward a successful direction.

To tackle this issue, *Il Mercato* hires a Data Preprocessing team whose primary focus will be to “monitor the business and segment the customers”.

# Project Objective

The hired DP team will have at hand a dataset including various information regarding each customer's transaction and the main objective is to present *Il Mercato* with an exploratory analysis and analytic-based table (an ABT).

These two deliverables will allow the company to answer some simple business questions and also get some descriptive insights regarding the customers who choose *Il Mercato.*

The creation of these deliverables will be made by preparing the data (which will also be useful for advanced analysis methods in the future) and also by giving some insights on the business. This will also tackle the company's problem of lacking information on their activity and mainly regarding their customers' behavior.

# Methodology

The execution of the project will involve several different stages which will also be divided into different phases:

- **Data Pre-Processing:** Initially the team went through some steps to preprocess the data. This initial stage included some first visualizations of the data, the exclusion of some clear and visible outliers, the treatment of data inconsistencies (divided into 2 steps), and then the preparation of the final version of the Transaction Table. These steps were divided into what we called phases, where we used different programs to complete the defined steps. These were:

    - **Phase 1:** Initial Visualizations & Outlier Exclusion which will be executed in SAS Enterprise Miner;
    - **Phase 2:** Treatment of initial data inconsistencies which will be executed in Python;
    - **Phase 3:** Imputation of values for the missing values existent in the dataset which will be executed (again) in SAS Enterprise Miner;
    - **Phase 4:** Final assessment of possible inconsistencies within the dataset and finalization of the transactional table which will be done using Python;

*After these phases are concluded the final version of the transactional table will be ready to be used and this version will be the one that will be utilized in the Data Visualization stage.*

- **Creation of the Analytical Base Table (ABT):** In this stage, the team will proceed to create the desired deliverable which is the Analytical Base Table. This creation will be made by adding up and organizing the data existent in the final transactional table, combining all the existent pieces of information regarding the company's customers into a single table in which each row will have the characteristics that characterize each customer. This process will be done using the SAS Studio software - which essentially uses SQL code to make the creation of the final ABT.

*After this stage is completed the final version of the Analytical Based Table will be finished.*

- **Data Visualization:** In this final stage, to be able to take some more insights, and in order to make the analysis of our dataset more intuitive, the team opted to create some Dashboards using PowerBI. This will allow the people of *Il Mercato,* will then be able to make conclusions regarding their customers and the transactions made in their stores more automatically than ever before.

*Having gone through all these stages, we will finally have all the deliverables done and ready for delivery (also accounting for this report of course).*

# Data Preprocessing

In the stage that involves preprocessing, the dataset will be divided into 4 steps where different tools will be used for each. The odd ones will be processed with SAS Enterprise Miner and the even ones with Python. Although at first glance it may not seem a logical line of reasoning, the methods used in Python forced an earlier treatment of outliers and, given this, the repetitive switching between programs was necessary. In addition, we tried to leverage our knowledge and perform the treatments based on our comfort zone.

## Phase 1: Initial Visualizations and Outlier Exclusion with SAS

The first phase of preprocessing (as it was mentioned before) consisted of making some initial visualizations of the dataset as well as the exclusion of some elements that were outliers to the general pattern of the dataset. These steps were implemented using the SAS Enterprise Miner program where the following diagram was created.

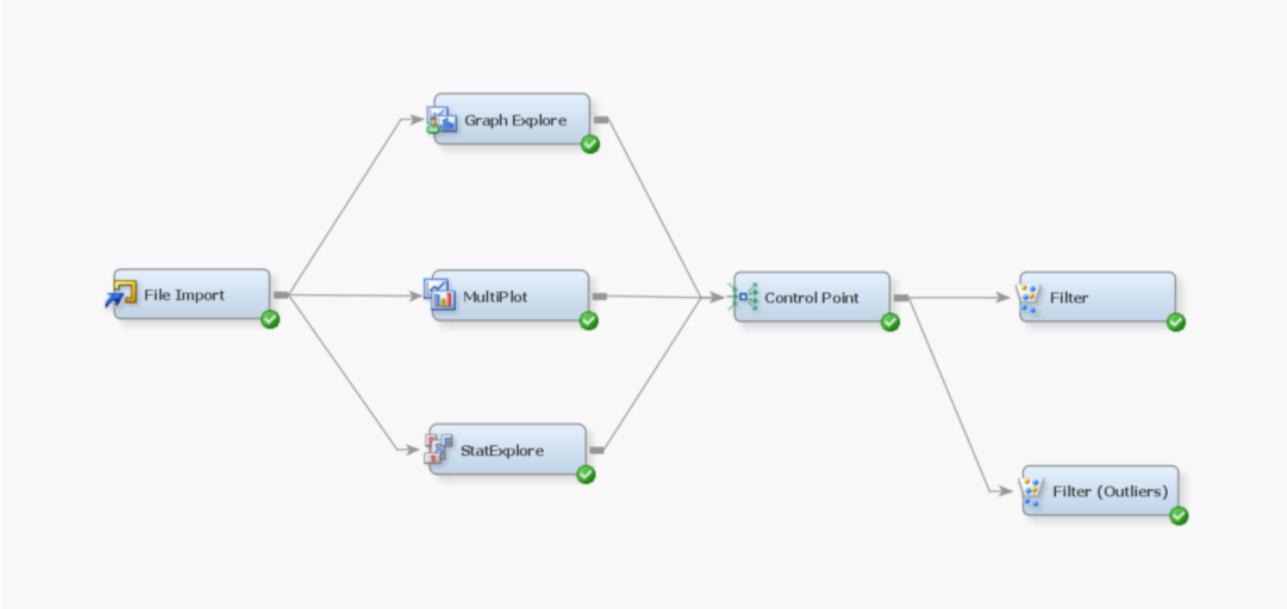


Fig.1 - SAS Miner Phase 1 Diagram

The first node that was used had the very simple utility (yet very important) of importing the dataset provided by the company. Furthermore, also in this node's definitions, in the left section of the interface, the team proceeded to change the automatic definitions of the variables and their roles. After making the changes to the automatic settings the program generated, the variables looked as follows:

| Name | Role | Level | Report | Order | Drop |
|---|---|---|---|---|---|
| city | Input | Nominal | No | | No |
| city_code | Input | Interval | No | | Yes |
| Customer_since | Input | Interval | No | | No |
| cust_id | ID | Nominal | No | | No |
| DOB | Input | Interval | No | | No |
| Gender | Input | Nominal | No | | No |
| Kids | Input | Nominal | No | | No |
| Nationality | Input | Nominal | No | | No |
| Payment_type | Input | Nominal | No | | No |
| prod_cat | Input | Nominal | No | | No |
| prod_cat_code | ID | Interval | No | | Yes |
| prod_subcat | Input | Nominal | No | | No |
| prod_subcat_co | ID | Interval | No | | Yes |
| Qty | Input | Interval | No | | No |
| Store_type | Input | Nominal | No | | No |
| Tax | Input | Interval | No | | No |
| total_amt | Input | Interval | No | | No |
| transaction_id | ID | Nominal | No | | No |
| tran_date | Input | Interval | No | | No |

Fig.2 - Variables' Level and Role Definition

In summary, the dataset has 2 variables that play the role of ID, and the rest are inputs. Regarding levels, 10 variables are nominal and 6 are intervals. For these accounts, the variables which end in *code* are not considered, because they are useless for this project. As the goal is not to create a conceptual data model, remaining the columns of this kind has no benefit, and the information is redundant with the corresponding nominal variables. For this reason, they are labeled with 'Yes' in the column called 'Drop' in the table presented above.

From the File Import node, three nodes branch out: GraphExplore; MultiPlot; StatExplore. We decided not to delve too deeply into the analysis of exploration nodes, because the information they offer is intrinsically related to the graphs presented in MultiPlot. This node provides us with graphs on the distribution of all input variables that allow us to have an initial notion of data tendencies. In the report's body, we will only present those whose peculiarities are of major importance, because including all of them would cause visual pollution. Those that give no reason to be singled out will be put in the annexes.

Regarding the nominal variables, in a first analysis, the two that need special attention are *Nationality* and *city*, both and their graphs are respectively presented in the next page. The first one seems much more problematic, since more than 50% of the observations are missing values. That said, it presents a great potential to be eliminated, because the proportion of lack of information is quite high. However, the final decision on this aspect will be postponed to the second phase. As for the *city* there is little to worry about, 9 is a very small number compared to the total number of rows.

In the case of the interval variables, there are 4 to mention: *Tax*, *total_amt*, *DOB* and *Customer_since*; which are represented by the figures 5, 6, 7 and 8, respectively. All of them have extreme outliers, besides the first two have few missing values.

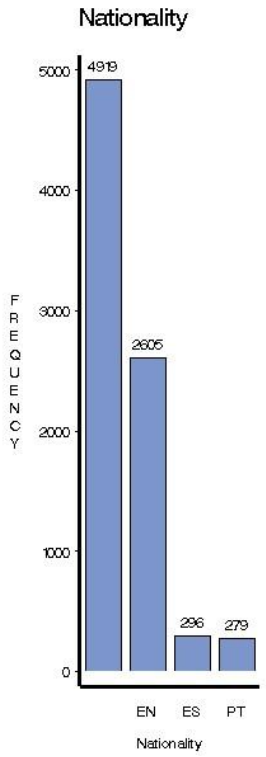


Fig.3 - Variable Nationality distribution

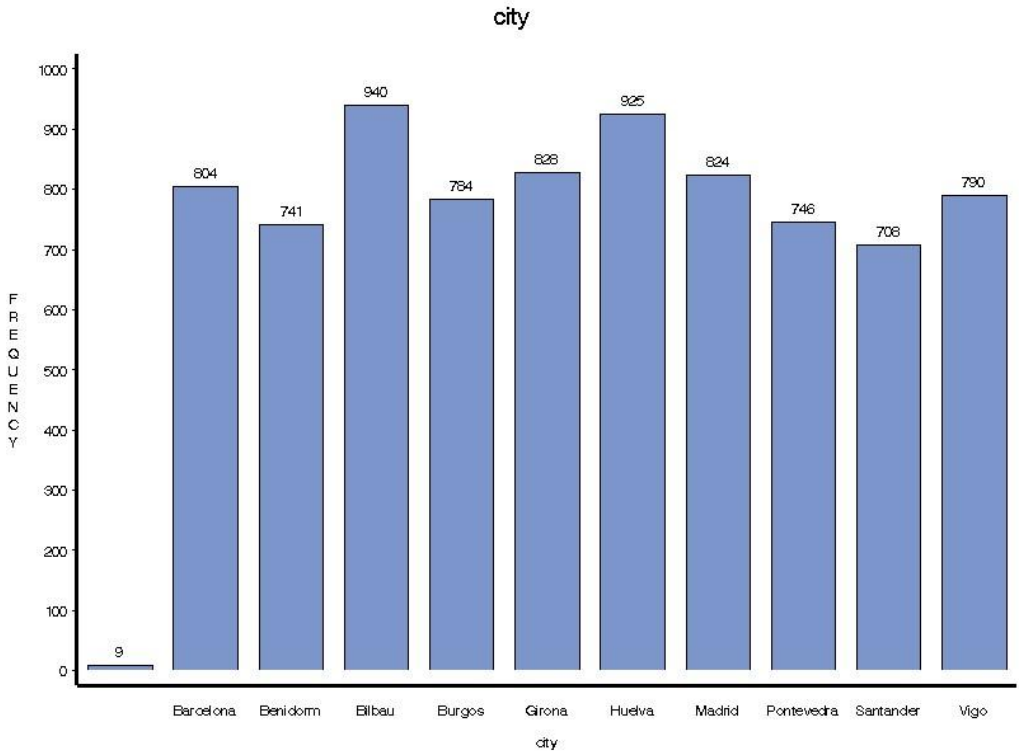


Fig.4 - Variable City Distribution

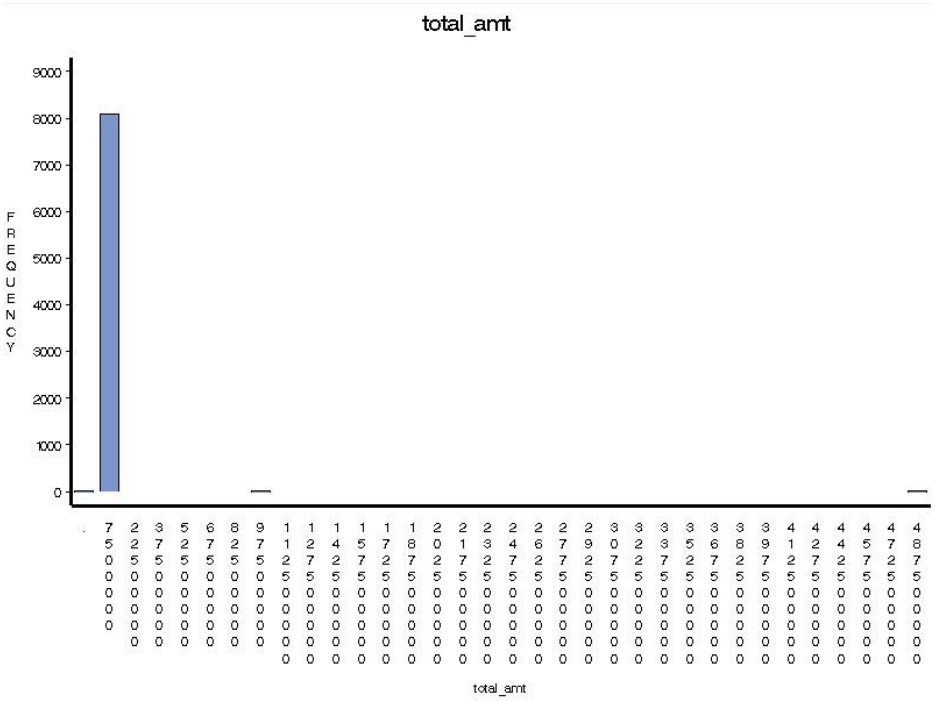


Fig.5 - Variable total_amt distribution

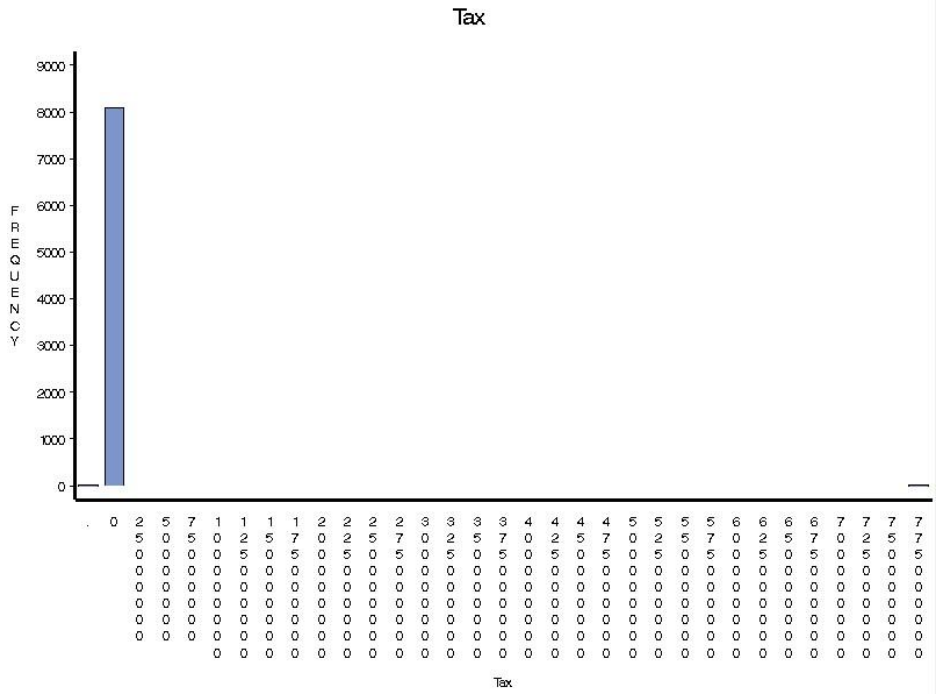


Fig.6 - Variable Tax distribution

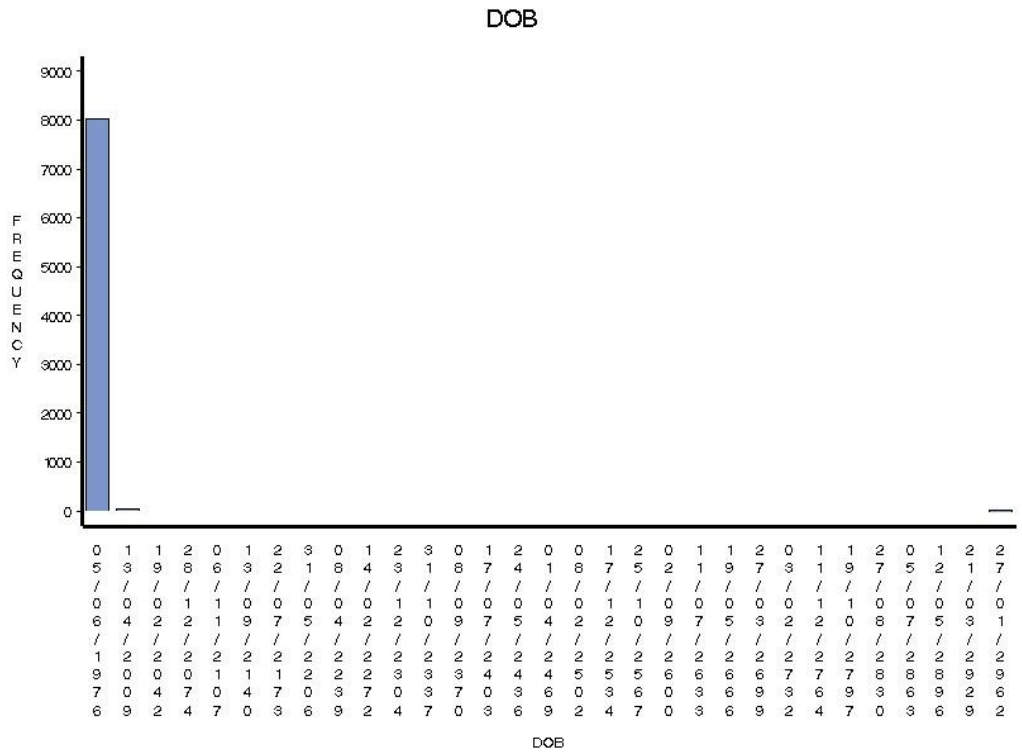


Fig.7 - Variable DOB distribution

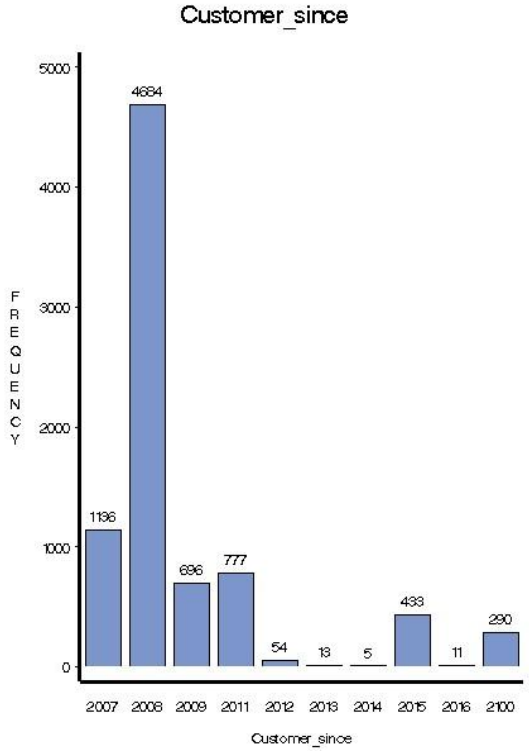


Fig.8 - Variable Customer_since distribution

To conclude this phase, we used the Filter node to be able to eliminate the outliers we visualized earlier. This action was possible through the option called 'Interval Variables' in the left section of the interface. Below is the final table for this step.

| Name | Report | Filtering Method | Keep Missing Values | Filter Lower Limit | Filter Upper Limit |
|---|---|---|---|---|---|
| Customer_sinc | No | User Specified | Default | 2000 | 2022 |
| DOB | No | User Specified | Default | 0 | 16803.94 |
| Qty | No | Default | Default | . | . |
| Tax | No | User Specified | Yes | 0 | 47649897 |
| city_code | No | Default | Default | . | . |
| total_amt | No | User Specified | Yes | 0 | 47649897 |
| tran_date | No | Default | Default | . | . |

Fig.9 - Interval variables outliers treatment

*Customer_since* was handled by writing the upper limit directly to the appropriate place. Although in the three intermediate nodes we verified that the last transaction in the dataset under study occurred in 2014, therefore having customers who registered in 2015 or 2016 makes no sense, we decided to treat these values as inconsistencies rather than pure outliers, deferring their treatment to the next phase. The remaining three were filtered using the slider at the top of the graphs, where we defined the acceptable region. As an example, below is the filtering of *total_amt*.

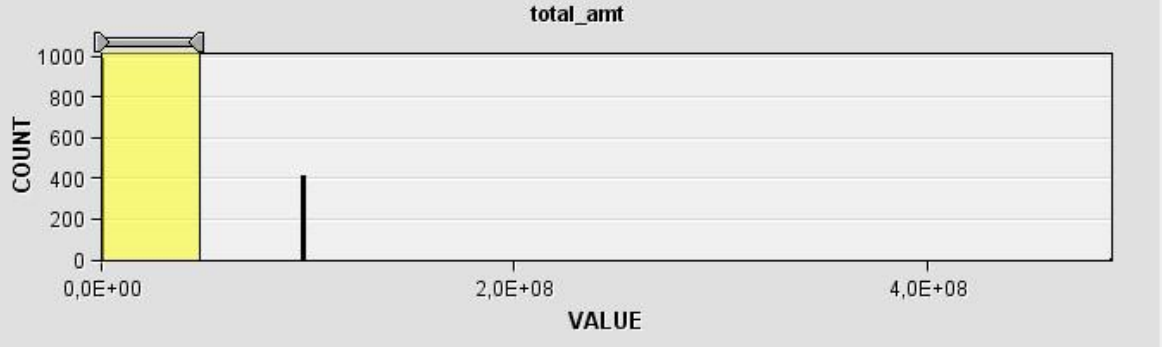


Fig10 - *total_amt* outliers treatment

Once the outliers were excluded, we exported the resulting table to an excel in order to proceed with the analysis.

# Phase 2 – Treating inconsistencies with Python

Extreme values were not the only factory errors in the dataset. There are also quite a few incorrect ones. Therefore, in this second phase, we removed inconsistencies from the Transactional Table with Python, supported by numpy and panda libraries. However, before solving the problem at hand, it is necessary to file some edges. Starting by deleting the columns ending in *code*, as justified in the previous phase.

In SAS Enterprise Miner, dates are converted into the number of days passed since the 1st of January of 1960, so as it is in our interest to keep the original format, it is necessary to import the dataset without changes. To reset the format of *tran_date* and *DOB*, we started by creating a dataframe that contained these variables and the two IDs. After that, we removed the duplicated values of each transaction, leaving only the last row associated. This step prevents the existence of one more *tran_date* or *DOB* per transaction, which would be an inconsistency. Then, we defined *cust_id* and *transaction_id* as indexes for the transaction table and dates dataset, in order to equalize both and date main data columns return to their original format. *reset_index*

The last adjustments we had to make are both related to the variables *Tax* and *total_amt*. As in the first phase missing values were not treated, when exported, the blank values of the interval variables became dots. In addition, in this process, both columns become object date types. Logically, it is important to solve these two anomalies.

At this point, it is safe to proceed to the treatment of inconsistencies. First, we checked that there were no duplicate lines, which was confirmed. The remaining discrepancies are particular to certain variables that in our analysis have contradictory or out-of-context values.

The simplest problem is the fact that there are values in both ***total_amt*** and ***Tax*** with more than two decimal places. As visualized in the MultiPlot node of phase 1, the stores presented in the data are all Spanish and the customers are split between native, English and Portuguese. Taking this into account, as neither the euro nor the pound is monetized to more than two decimal places. Thus, the existence of such values makes no sense.

Concerning ***Customer_since***, the problem begins with the fact that there were clients with more than one year associated. It is not coherent that an individual started being a customer of the company in 2008 and in 2012, for example. Each one has only one year of entrance. Since the problem is per customer, to solve it, we needed to define 'cust_id' as an index. Then, we sorted the dataset by the column in question in an ascending order, i.e. from the oldest to the most recent customers, and, with group by function, we checked how many people are in this situation (783) (as can be seen in the image below).

```
cust_id
266816    2
266819    2
266822    2
266827    2
266829    2
          ..
275226    2
275227    2
275230    2
275246    2
275250    3
Name: Customer_since, Length: 783, dtype: int64
```

Fig.11 - Customers who have different ‘Customer_since’ years

Subsequently, we matched each client with its earliest year; a process that would not have been possible if we had not sorted the values. We chose to fix the first year, because, excluding 2015 inconsistency, the observations corresponding to customers with unique values were all from years prior the space-time under study (as can be concluded in figure 12).

```
2008    2378
2007     139
2009      60
2011      43
2015      22
Name: Customer_since, dtype: int64
```

Fig.12 - Observations with an unique ‘Customer_since’ value counts

After this change, the variable was still not 100% correct, because there are rows where the value of *Customer_since* is higher than the corresponding year of *tran_date*. In practical terms, this implies that someone made a transaction as a registered customer of the shop, but not yet being one, which is contradictory.

| cust_id | transaction_id | tran_date | Customer_since |
|---|---|---|---|
| **271854** | 94579687823 | 2013-01-29 | 2014 |
| **271854** | 94579687823 | 2013-01-29 | 2014 |
| **271854** | 95802108331 | 2013-12-24 | 2014 |
| **271854** | 95802108331 | 2013-12-24 | 2014 |
| **271377** | 95105057906 | 2013-09-23 | 2015 |
| **271377** | 95105057906 | 2013-09-23 | 2015 |
| **271377** | 95105057906 | 2013-09-23 | 2015 |
| **271377** | 95105057906 | 2013-09-23 | 2015 |
| **271377** | 96433587064 | 2013-11-17 | 2015 |
| **267466** | 99581788104 | 2014-02-20 | 2015 |
| **267466** | 99581788104 | 2014-02-20 | 2015 |
| **267466** | 99581788104 | 2014-02-20 | 2015 |
| **267466** | 99581788104 | 2014-02-20 | 2015 |
| **267466** | 99598973047 | 2013-08-06 | 2015 |
| **267466** | 99598973047 | 2013-08-06 | 2015 |
| **267466** | 99598973047 | 2013-08-06 | 2015 |
| **267466** | 99598973047 | 2013-08-06 | 2015 |
| **275204** | 97998106932 | 2013-08-04 | 2015 |
| **275204** | 97998106932 | 2013-08-04 | 2015 |
| **275204** | 99564160197 | 2014-01-28 | 2015 |
| **275204** | 99564160197 | 2014-01-28 | 2015 |
| **271243** | 94647669972 | 2013-04-29 | 2015 |
| **271243** | 98096058779 | 2013-04-05 | 2015 |
| **271243** | 98096058779 | 2013-04-05 | 2015 |
| **271243** | 98096058779 | 2013-04-05 | 2015 |
| **271243** | 98096058779 | 2013-04-05 | 2015 |

Fig.13 - Observations with a ‘tran_date’ older than ‘Customer_since’ year

The figure above allowed us to analyze some points that indicate the best approach to take. Initially, we considered transforming the value of *Customer_since* of these five problematic customers into the year of their oldest transaction. However, this line of reasoning went against the general picture of the dataset.

```
2008    4884
2007    2471
2009     298
2011      97
2015      22
2012      19
2014       4
Name: Customer_since, dtype: int64
```

Fig.14 - 'Customer_since' value counts

```
Int64Index([2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013,
            2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013],
           dtype='int64', name='tran_date')
```

Fig.15 - Years of the transactions of clients who become customers in 2012

As mentioned above, the data under analysis is in the space-time from December 2012 to February 2014. Figure 14 shows us that in the years in question, there are 19 for 2012, none for 2013 and 4 for 2014. The last mentioned are presented in figure 15; therefore, they are in the group of inconsistencies. The remaining image indicates that the 2012 customer transactions all occur in 2013. By cross-referencing these facts, the conclusion we have come to is that all the rows have different transaction years and values of *Customer_since*. Thus, we can assume that the individuals under study are long-standing and/or regular customers. That said, it would be illogical for us to treat an error with a trend that does not happen, so we decided that the most correct approach was to eliminate the 26 rows. Since there were not many, it did no't make a big impact to exclude them.

At an early point when we analyzed the raw ***DOB***, there did not appear to be any error in this variable. However, when comparing date of birth with *Customer_since*, there are some individuals who theoretically made registered purchases before the age of 16, and it is not legal for such people to have a customer account. Faced with this situation, we decided to simply delete the wrong rows. We considered this the best option because, although more than desired, the observations were relatively few and we did not want to jeopardize the veracity of yet another variable. Having already eliminated about 5.6%, from now on, no further observations have been discarded. We have slightly exceeded the 4% margin, but we believe this is best for the analysis. It is also important to note that the sequence of inconsistency treatment was not coincidental. If we had treated the *DOB* before the *Customer_since*, the work would be inconclusive, because the correction made in the second one generates new consumers with incorrect age. Furthermore, these two inconsistencies were dealt with first, because they are the only ones where we drop rows, so in the remaining processes the computer is spared of running unnecessary problems.

Another inconsistency regards ***Nationality***. Aside from the considerable amount of missing values found in phase 1 for this feature, we found two cases for these missing values. The first one were customers that had never disclosed their nationality (Figure *), whereas the second one were customers that once shared their nationality and did not do so for other

transactions (Figure *). In addition to these two cases, by creating a dataframe with the count of different nationalities grouped by customer, we found customers who had more than one associated nationality (Figure *).

| | cust_id | transaction_id | Nationality |
|---|---|---|---|
| **3743** | 266816 | 67944016762 | NaN |
| **3808** | 266816 | 67944016762 | NaN |
| **6538** | 266816 | 88316484015 | NaN |
| **6547** | 266816 | 88316484015 | NaN |

Fig. 16 - Nationality inconsistency case 1 example

| | cust_id | transaction_id | Nationality |
|---|---|---|---|
| **3642** | 266806 | 71028241442 | NaN |
| **3644** | 266806 | 71028241442 | NaN |
| **5044** | 266806 | 14141098432 | EN |
| **5046** | 266806 | 14141098432 | EN |
| **5047** | 266806 | 14141098432 | EN |
| **5062** | 266806 | 14141098432 | EN |
| **5066** | 266806 | 14141098432 | EN |
| **5067** | 266806 | 14141098432 | EN |

Fig. 17 - Nationality inconsistency case 2 example

| | cust_id | transaction_id | Nationality |
|---|---|---|---|
| **723** | 267067 | 9611958302 | EN |
| **724** | 267067 | 9611958302 | EN |
| **5794** | 267067 | 50139060955 | ES |

Fig. 18 - Nationality inconsistency case 3 example

The first case was left in charge of phase 3. Regarding the remaining ones, we decided that, for every customer in the second case, that nationality will stand for their missing values, and, in the third case, the nationality chosen would be the latter, as if to consider that the previous one would have been an imputation error. With just one process we solved both at the same time. For that, we created a dataframe with the last non-missing nationality used by every customer. With that, around 840 customers already had a nationality which could be imputed to other transactions. Then, we assigned existing nations dataframe values in place of the old ones based on the *cust_id* as index, by equating the *Nationality* column of both data frames based on it, leaving us now with, approximately, 30% of missing values which shall have a value imputed in the next phase. Note that, while lower than previously, this value is still relatively high

and this feature should be taken with a grain of salt, as it presents a relatively large margin of error. Nonetheless, we decided that, having already eliminated around half of the missing values, that it could be useful to keep it for others that may be interested to work with, even if not fully reliable. Finally, we confirmed with the same method used initially, if there were still customers with more than one nationality, which was not the case.

In regards to ***Kids***, we found out that, for reasons totally unknown to us, some customers strangely said they had and did not have kids within the same transaction.

| | cust_id | transaction_id | Kids |
|---|---|---|---|
| **4961** | 266806 | 14141098432 | 1 |
| **4963** | 266806 | 14141098432 | 1 |
| **4964** | 266806 | 14141098432 | 0 |
| **4979** | 266806 | 14141098432 | 0 |
| **4983** | 266806 | 14141098432 | 0 |
| **4984** | 266806 | 14141098432 | 0 |

Fig. 19 - Kids inconsistency example

Having taken note of this problem, we decided that we would fill all the values within each transaction with their respective mode. With that in mind, we aggregated the data by the mode of each *transaction_id* that each *cust_id* made within the data timeframe. As could be expected, more than 1700 transactions had an even distribution of values, resulting in them being bimodal, which are represented as NaNs, due to the condition of the mode being in the interval [0, 1], which these are not. With a data frame composed of only those transactions whose *Kids* are evident, we have access to every non-bimodal mode there is, which we, after appending *transaction_id* to the index of the original data, then used to, as done before, impute these values into the post phase 1 original dataframe, with these transactions now all presenting only one value for *Kids*.

For the bimodal transactions, more specifically for those of customers which have at least one valid mode, we decided to fill them, represented as missing values, with the most appropriate value, by the use of Forward Fill, which will impute the NaNs that happened before the first non-bimodal transaction of each customer with its value, alongside Backwards Fill, which does the same thing for the NaNs that happen after the last non-bimodal transaction, using the last valid value. They are both limited by the *cust_id* index, ensuring that no value is passed to other customers and all their values are consistent.

Example: a customer that has 5 ordered transactions with the modes (NaN, NaN, 0, 1, NaN) will have their modes changed to (0, 0, 0, 1, 1). A customer with no accurate mode will keep their NaNs.

Having done that, we then dropped every duplicate index (*cust_id* and *transaction_id*) and assigned its values to the ones within the original dataset, based on the previously mentioned indexes.

In addition to all the situations mentioned, we also found different values associated with the same customer in *DOB* and *Gender*.

To prepare the data to return to SAS, in the hopes of treating the remaining inconsistencies after this initial treatment, we reset its index and exported it to a new Excel file to be used.

## Phase 3 – Imputation of values for the remaining missing values in the dataset in SAS Enterprise Miner

Having a more complete dataset, we can now better impute the missing values that remain in the data, as more information is given about the customer base, which is the main objective of this phase. The diagram for this part is as follows:


Fig.20 - SAS Miner Phase 3 Diagram

We start off by importing the new data set, succeeded by a basic Graph Explore to visualize the table.

The cluster node has its Number of Clusters parameters set to ‘Automatic’ and uses the Ward criterion. This node will cluster the data which will help us understand if there are any multidimensional outliers. As we can see from figure 17, there are no significantly small clusters, which means that there are no multidimensional outliers for us to worry about. Therefore, we conclude that we can safely impute the values.

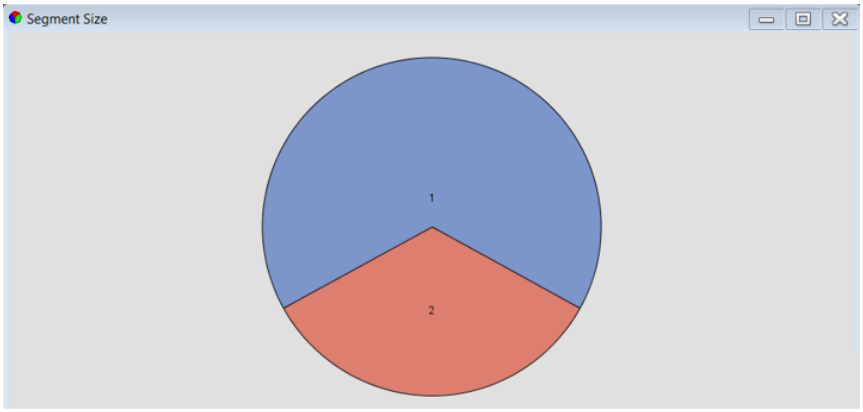


Fig.21 - Cluster node result

Then, the Impute node shall impute the remaining missing values. For its *imputation method*, the Decision Tree was chosen for both Class and Interval variables, with no target variable imputation needed. The normalization for Class variables was set to True, missing cutoff was set to 50%, and the *random seed* was set to 12345.

Finally, we exported the data from the Impute node for the next phase.

## Phase 4 – Final Adjustments with Python

The final preprocessing phase serves to organize the final product and check that all changes have been successful. For this purpose, we wrote the code based on phase 2 one. We imported the same libraries, the dataset resulting from phase 3 and the original one, in order to redo the step of reformatting the columns with dates. After that, we convert the values of the original columns into those imputed by SAS, and the values of the variables *total_amt* and *Tax* were rounded to two decimal places. Then, we deleted all the useless columns, including those starting with *Imputed*, since their values were already included.

The only very problematic variables from phase 2 that changed in phase 3 were *Nationality* and *Kids*, so we only needed to check their consistency. By running the code that indicates if there is any customer with more than one associated nationality, we conclude that no inconsistency in *Nationality* was imputed in phase 3, so its treatment is completed. Unfortunately the same cannot be said about the variable *Kids*, because there were still 235 customers with bi-modal values. By redoing the whole process, we reached the point where it was not possible to reach a conclusion about the correct value of 944 observations. Although the authenticity of the variable is rather doubtful, we decided not to eliminate it and convert the missing values into the number "-1". In this way, there is a chance that future Machine Learning engineers will use this variable if it shows potential.

Summarizing our preprocessing, we tried to keep as much as possible a balance between treating and eliminating; so that we both did not damage the data veracity and did not lose potentially important information. In terms of elimination, we prioritized keeping columns rather than the observations. There is a possibility that we discarded some relevant rows, but from our point of view it is more beneficial to keep all variables in order to get a more diverse and complete analysis, and in fact, in the end, we eliminated none.

To conclude, it is important to point out that one should be careful about relying on the variables *cust_id*, *Nationality* and *Kids*, especially the last one, because they have been treated a lot and their values are not undoubted.

# Creation of the Analytical Base Table (ABT)

In this part of the project, the team started by deciding on the variables to include in the Analytical Base Table. After some brainstorming, we were set on including the following variables in the ABT:

| Variable | Description |
|---|---|
| *cust_id* | Customer's ID |
| *age* | Customer's age |
| *Nationality* | Customer's nationality |
| *Gender* | Customer's gender |
| *Kids* | Customer has kids (1: yes; 0: no; -1: inconclusive) |
| *Customer_since* | Year in which the customer became a client |
| *first _purchase* | Date of the first purchase |
| *last_purchase* | Date of the most recent purchase |
| *total_purchases* | Number of purchases made |
| *monetary* | Total amount of money spent |
| *total_tax* | Total amount of taxes paid |
| *total_prods_purchased* | Total products bought |
| *max_mnt_p_purchase* | Maximum amount of money spent on a purchase |
| *min_mnt_p_purchase* | Minimum amount of money spent on a purchase |
| *avg_mnt_p_purchase* | Average amount of money spent on a purchase |
| *purchases_in_2012* | Number of purchases made in 2012 |
| *purchases_in_2013* | Number of purchases made in 2013 |
| *purchases_in_2014* | Number of purchases made in 2014 |
| *mnt_in_2012* | Total amount of  money spent in 2012 |

| | |
|---|---|
| *mnt_in_2013* | Total amount of  money spent in 2013 |
| *mnt_in_2014* | Total amount of  money spent in 2014 |
| *purchases_with_Credit Card* | Number of purchases made with Credit Card |
| *purchases_with_PayPal* | Number of purchases made with PayPal |
| *purchases_with_Cash* | Number of purchases made with Cash |
| *mnt_with_Credit Card* | Total amount of money spent using Credit Card |
| *mnt_with_PayPal* | Total amount of money spent using PayPal |
| *mnt_with_Cash* | Total amount of money spent using Cash |
| *n_prods_Books* | Number of products bought from Books category |
| *n_prods_Clothing* | Number of products bought from Clothing category |
| *n_prods_Footwear* | Number of products bought from Footwear category |
| *n_prods_Home and Kitchen* | Number of products bought from Home and Kitchen category |
| *n_prods_Bags* | Number of products bought from Bags category |
| *n_prods_Eletronics* | Number of products bought from Electronics category |
| *mnt_Books* | Total amount of money spent in Books category |
| *mnt_Clothing* | Total amount of money spent in Clothing category |
| *mnt_Footwear* | Total amount of money spent in Footwear category |
| *mnt_Home and Kitchen* | Total amount of money spent in Home and Kitchen category |
| *mnt_Bags* | Total amount of money spent in Bags category |
| *mnt_Eletronics* | Total amount of money spent in Electronics category |

The following step consisted on building the Analytical Base Table, which was done using Sas Studio. However, before going into SAS, the team decided to change the format of the date variables (*DOB* and *tran_date*). This change was done for a better understanding of these variables, and to make the calculations with these same variables simpler, when programming in SAS Studio. This was done directly in Excel, following these steps:

1. selecting both the date type variables;
2. selecting the option ‘Format cells’;
3. selecting the *DD/MM/YYYY* date type.

(as seen in the picture below)

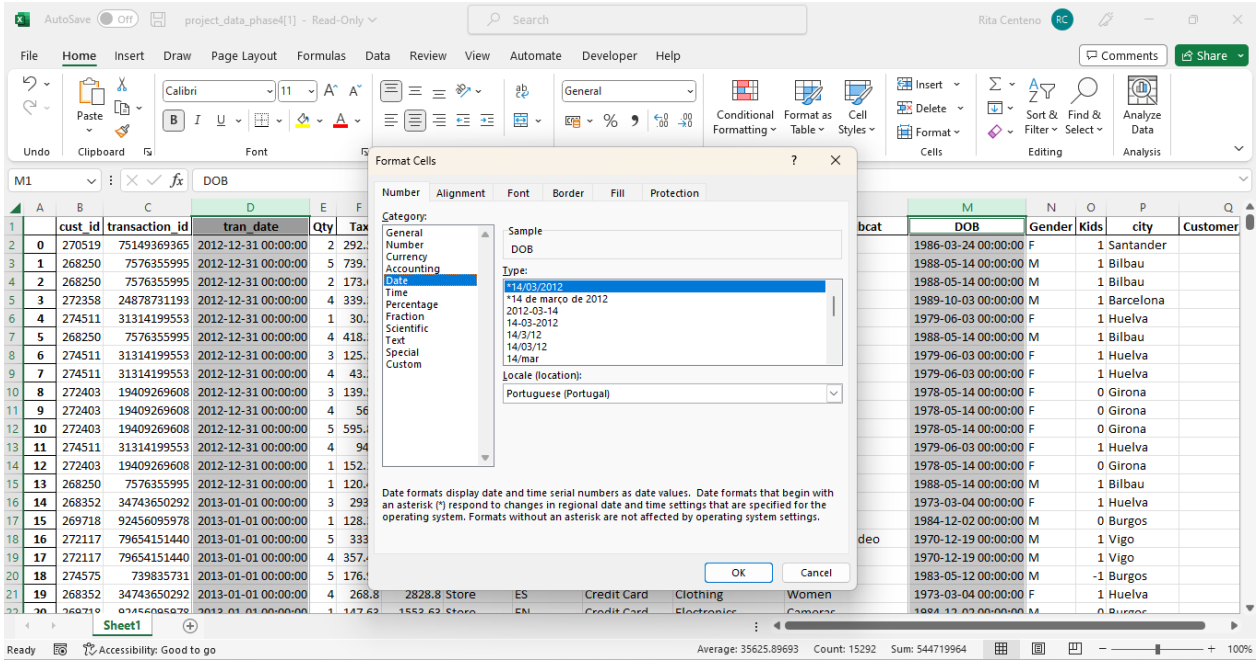


Fig.22 - Process of formating the date columns

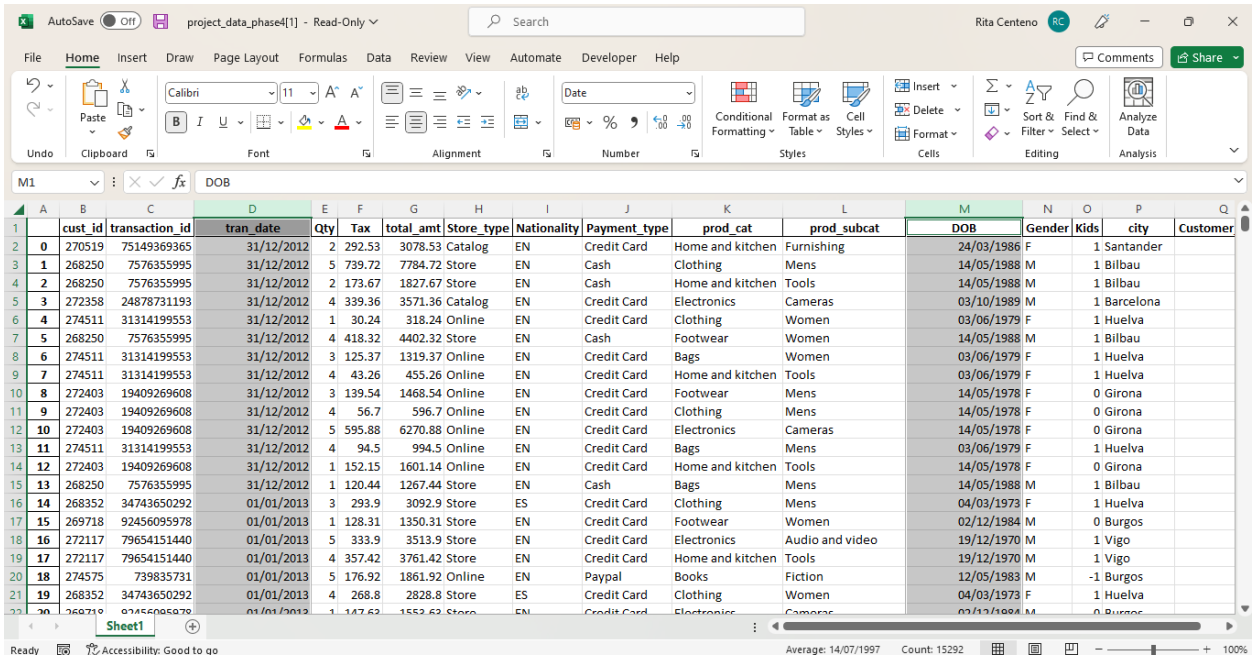

Fig.23 - Final transactional table with dates in the right format

Having the Transaction Table already treated, we proceeded to build the Analytical Base Table, which is one of the deliverables.

In the ABT we decided to keep six of the columns from the Transactional Table (*cust_id*, *age*, *Nationality*, *Gender*, *Kids* and *Customer_since*) as they hold representative information

about the customer. In addition to that, the variables *first_purchase* and *last_purchase* were created, in order to make clear the time frame in which the customer interacted with the company.

Moreover, variables which summarized the customers activity within the time frame of the three years which we are studying were created. This includes the total of products purchased (*total_prods_purchased*), the total amount of money spent (*monetary*), the total amount of taxes paid (*total_tax*) and the total number of products purchased (*total_prods_purchased*). Furthermore, to understand the customer's spending habits, we calculated the maximum, the minimum and the average money spent on a purchase (*max_mnt_p_product*, *min_mnt_p_product* and *avg_mnt_p_product*, respectively).

Additionally we thought it might be relevant to understand the customers habits in each individual year. To do this, we created variables which represented the number of products bought and the total of money spent for each customer by year. For trying to understand the customer's preferential payment type, a similar approach was taken. The same variables were created; however, instead of dividing the purchases by year, they were divided into payment types.

Finally, as we deemed important to understand the customer's preferences in what regards the type of products they buy, we decided to, in the same way as we did before, calculate the total of products bought and the amount of money spent in the different product's categories. The team decided to discard this study for the product's subcategories as it would mean creating 36 new columns, which would have sparse values. Instead, we created a column that tells us in which product subcategory the customer spent more money on.

Lastly, during this process of variable creation, we noticed that, when the variables *Kids* and *prod_subcat* were created, the table returned had one extra row. In what concerns the variable Kids, this happens because a customer made two purchases in the same day and the number of kids assigned to each of the transactions was different, as can be seen in Figure 23. Contrary to what one may think, this is not an inconsistency because, considering the case in which the first child of this customer has been born in between transactions, these values for the variable can be accepted.

| | cust_id | age | Nationality | Gender | Kids | Customer_since |
|---|---|---|---|---|---|---|
| 2 | 266799 | 44 | EN | F | 1 | 2008 |
| 3 | 266806 | 23 | EN | F | 0 | 2008 |
| 4 | 266814 | 32 | EN | F | 1 | 2008 |
| 5 | 266816 | 40 | EN | M | 1 | 2008 |
| 6 | 266819 | 32 | ES | M | 0 | 2008 |
| 7 | 266822 | 44 | EN | F | 1 | 2008 |
| 8 | 266823 | 28 | EN | M | 1 | 2008 |
| 9 | 266827 | 37 | EN | F | 1 | 2007 |
| 10 | 266829 | 38 | EN | F | 1 | 2008 |
| 11 | 266841 | 23 | EN | M | -1 | 2008 |
| 422 | 269766 | 23 | EN | F | 0 | 2007 |
| 423 | 269767 | 34 | EN | F | 0 | 2008 |
| 424 | 269767 | 34 | EN | F | 1 | 2008 |
| 425 | 269772 | 33 | EN | F | 0 | 2008 |
| 426 | 269776 | 35 | ES | F | 0 | 2008 |

Fig.24 - Duplicate customer row because of Kids variable

In what regards the variable *prod_subcat*, what happens is that the client has the same amount of money spent for two different product subcategories, as can be seen in Figure 24.

| | cust_id | age | Nationality | Gender | Kids | Customer_since | first_purchase | | mnt_Clothing | mnt_Footwea | mnt_Home and kitchen | mnt_Bags | mnt_Electronics | prod_subcat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 266799 | 44 | EN | F | 1 | 2008 | 24/set/13 | .24 | 3228.81 | 0 | 0 | 0 | 0 | Comics |
| 3 | 266806 | 23 | EN | F | 0 | 2008 | 21/fev/13 | 1.9 | 3664.18 | 4299.55 | 7813.46 | 0 | 0 | Furnishing |
| 4 | 266814 | 32 | EN | F | 1 | 2008 | 06/mar/13 | .63 | 0 | 0 | 8321.75 | 1055.28 | 689.86 | Tools |
| 5 | 266816 | 40 | EN | M | 1 | 2008 | 29/mar/13 | .96 | 3419.98 | 81.77 | 0 | 0 | 1521.58 | Children |
| 6 | 266819 | 32 | ES | M | 0 | 2008 | 21/mar/13 | .88 | 0 | 1823.25 | 0 | 3769.16 | 6885.25 | Women |
| 7 | 266822 | 44 | EN | F | 1 | 2008 | 01/jun/13 | 0 | 1214.4 | 0 | 658.58 | 0 | 5760.36 | Personal Appliances |
| 8 | 266823 | 28 | EN | M | 1 | 2008 | 09/jan/13 | .38 | 0 | 0 | 6060.92 | 0 | 1152.52 | Kitchen |
| 9 | 266827 | 37 | EN | F | 1 | 2007 | 02/set/13 | .92 | 1331.52 | 0 | 0 | 0 | 1188.98 | Children |
| 10 | 266829 | 38 | EN | F | 1 | 2008 | 25/mai/13 | 9.3 | 211.06 | 517.14 | 4455.36 | 375.91 | 3259.75 | Fiction |
| 11 | 266841 | 23 | EN | M | -1 | 2008 | 03/jan/14 | .46 | 5047.64 | 0 | 4131.6 | 0 | 320.45 | Comics |
| 546 | 270641 | 23 | EN | M | 1 | 2009 | 25/mai/13 | 0 | 3474.12 | 0 | 4853.16 | 0 | 3436.55 | Bath |
| 547 | 270643 | 26 | EN | M | 1 | 2007 | 23/ago/13 | .42 | 1319.37 | 0 | 0 | 5478.59 | 6921.72 | Children |
| 548 | 270643 | 26 | EN | M | 1 | 2007 | 23/ago/13 | .42 | 1319.37 | 0 | 0 | 5478.59 | 6921.72 | DIY |
| 549 | 270644 | 30 | EN | F | 1 | 2008 | 03/mai/13 | 7.2 | 0 | 9291.94 | 4112.8 | 6232.2 | 1363.57 | Women |

Fig.25 - Duplicate customer row because of prod_subcat variable

In the end, we decided to only keep the first row that appears. Having done this, the final Analytical Base Table is complete.

# Data Visualization

## Step 1: Data Importation & Transformation

Having concluded the transformation and correction of the original Transactional Table provided by the company and the creation of the Analytical Based Table, the team proceeded to create some data visualizations using the Microsoft PowerBI platform. These same visualizations will be helpful to the employees of *Il Mercato,* as they will allow them to make some conclusions regarding the dataset that was provided.

Before starting to create the visualizations, the team imported both the analytical based and transactional tables into the program as queries and made some changes to their initial outlook using the *Power Query Editor* (which appeared by clicking on the Transform Data ribbon) .

On the Transactional Table, the changes applied were just the removal of the index column that was automatically generated after making the exportation of the final transactional table in phase 4, and the change of data type of the variable *tran_date* from Date/Time to just Date. On the other hand, in the imported ABT no changes were made.

Because the information of the initially imported tables was displayed in a way that was way too limited for the future creation of visualizations, we decided to divide the already imported datasets into three different queries (other than the original two): a **Customer Table**, a **Products Table** and also a **Transactions Table**.

### Customer Table

The **Customer Table** was created with the main objective of gathering in a single place (without duplicates) all the personal information regarding every customer. In this table there is no information about the transactions made, the products acquired or even the money each customer spent. This query will allow the company (just as it allowed the DP team) to take conclusions solely about the demographic characteristics of every customer.

The variables that are present within the table are:

- The Customer's ID;
- The Customer's Date of Birth;
- The Customer's age;
- The year in which these customers joined the company;
- The age with which these customers joined the company;
- The Customer's nationality;

- If the Customers have kids or not;
- The Customer's gender;
- The Customer's city which was created (or in this case brought into this table) by using the LOOKUPVALUE function of PowerBI making a cross reference between the index column of this table (*cust_id*) and the index column of the imported Transactional table (*cust_id*) to bring the corresponding value of the variable *City* to every customer

# Products Table

With the creation of the **Products Table,** the main goal was to create a sort of a dictionary that would simplify the analysis of the products bought.

The variables included in this table were:

- Every *transaction_id*;
- The category of every product bought;
- The subcategory of every product that was bought;
- The total quantity of each product bought in each transaction;
- The partial cost of every total quantity of products;

The creation of the table was executed in the Power Query Editor using the GROUP BY function, where the variables *Qty* and *total_amt* were grouped by each *transaction_id* and by the *prod_cat* and *prod_subcat* variables in a triple junction. This way this group by was performed as is shown in the figure below.

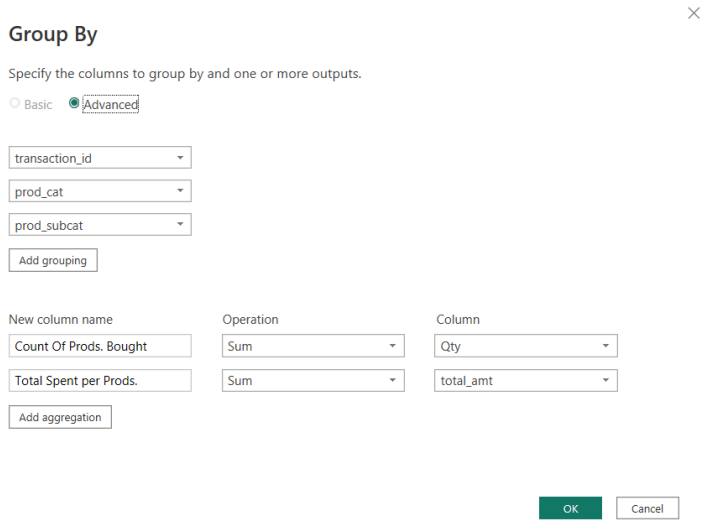


Fig. 26 - Group By of the variables included in the Products Table

# Transactions Table

Last but not least, the creation of the **Transactions Table** was performed. This happened as the team found the need of creating a table that would group by the information regarding every transaction as a whole having as the centerpiece of the table not every product that was bought (like in the transactional table resulting from the initial 4 phases of transformation) but every transaction. With this in mind, in the table the variables that are present are:

- Every *transaction_id* (that is presented in a unique manner - meaning without duplicates);
- The date in which every one of these transactions occurred;
- The payment type with which every transaction was made;
- The type of store in which these transactions were made;
- The total quantity of products that were bought in every transaction;
- The total amount of money that was spent in every purchase;
- The total amount of money that was paid in taxes per transaction;

The table was created from the original transaction table, with the team starting by removing the *cust_id* row and then proceeding to change the data type of the variable *tran_date* from Date/Time to Date. The following step which was the last one made using the Power Query Editor was to create the other columns of the table. For this, the team used once again the Group By ribbon and the grouping by was defined as the image below shows:

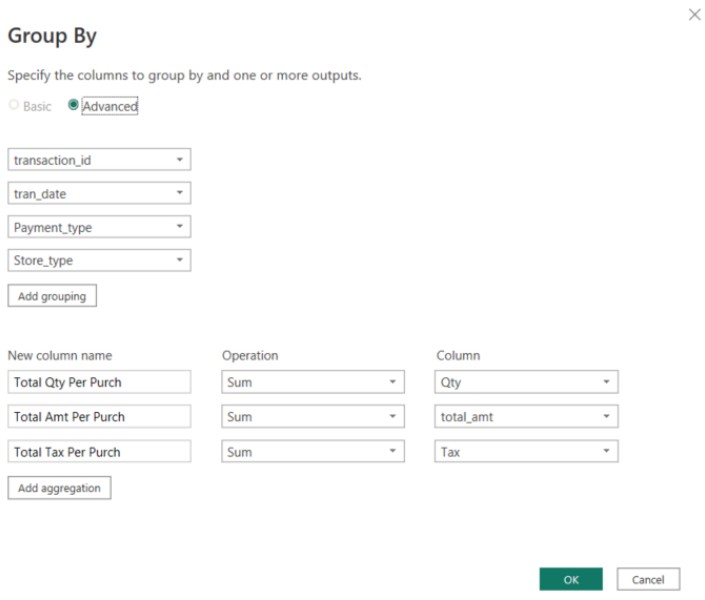


Fig. 27 - Group By of the variables included in the Products Table

To finalize the creation of tables, the team proceeded to create a calendar table so that all the dates existent within the data provided by the company as well as those that are present in the ABT could be ordered and divided properly further along the way (if needed).

# Step 2 - Data Modelling (Creation of the Relationship Data Model)

Having all these tables created, in order for them to inherit the information from the originally imported tables and for the tables to be connected between themselves the team used the model view to generate a data model within the PowerBI platform.

The creation of these connections will be very helpful for when the team uses slicers within the dashboards to filter the visualizations, as this will make all the visualizations be filtered according to the criteria that the users define in the filters.

These relationships are displayed in the figure below:

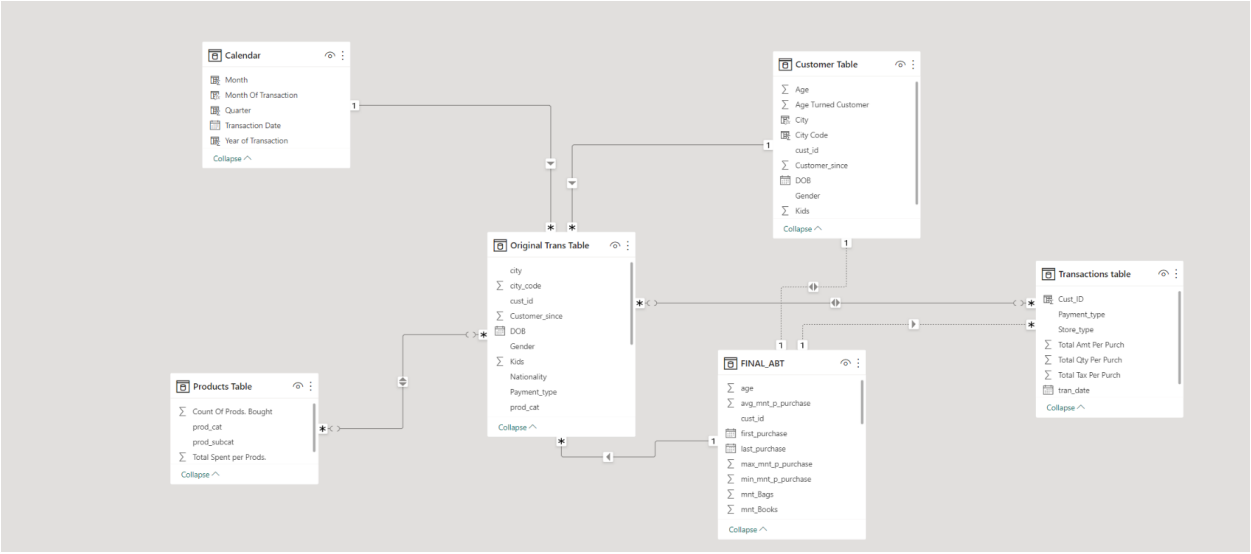


Fig. 28 - Data Model Containing All Relationships between the different tables that were created

In a somewhat summarized manner, the relationships shown above can be explained as:

- **From Calendar to Original Trans Table** (One-To-Many Relationship): which was established to interconnect the *tran_date* from the Original Transactional Table to the column Transaction Date of the Calendar Table;

- **From Products Table to Original Trans Table** (Many-To-Many Relationship): which was defined in order to pass the various instances of each existent *transaction_id* from one table to the other (without removing duplicate values);

- **From Customer Table to Original Trans Table** (One-To-Many Relationship): which was defined to pass the various instances of *cust_id* into the customer table without the duplicate instances;

- **From Transactions Table to Original Trans Table** (Many-To-Many Relationship): which was used to pass the various instances of *transaction_id* from the original Transactional table into the transactions Table;

- **From FINAL_ABT to Original Trans Table** (One-To-Many Relationship): which was used to connect the *cust_id* columns of both tables, with the relationship being established as a One-To-Many because the IDs of our customers appear without being duplicated;

- **From Transactions Table to FINAL_ABT** (Many-To-One Relationship): which was created to connect the cust_id values of both tables with it being a Many-To-One as the values in the Transactions Table appear duplicated (since a customer is allowed to make more than one transaction);

# Step 3 - Data Visualization

In this last step of the Data Visualization section of our project, we will be addressing the way in which the team opted to divide the charts and graphics created. The first two pages of the PowerBI file are a Cover Page and an Index which contains a button associated with every topic that will redirect the user towards the dashboard it selects. For the sake of organization, we opted to split the visualizations by the themes they addressed and the result of this division was a total of three dashboards:

- **Demographic Customer Data:** in this dashboard, the team inserted visualizations that address the personal information of our customers, as well as slicers that will allow the employees at *Il Mercato* to filter the customers whose information will be displayed in the visualizations;

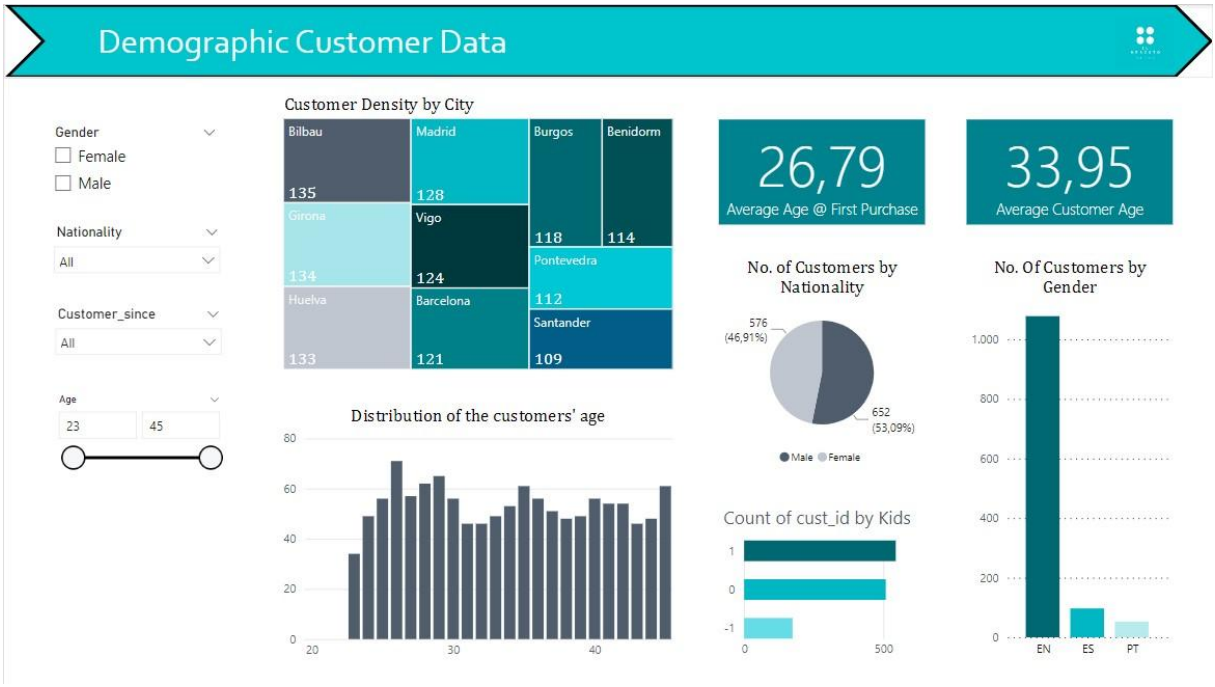


Fig. 29 - Demographic dashboard

- **Transactional Data:** in this dashboard, the existing visualizations relate to all the transactions that exist within our dataset, with the existing slicers allowing the viewer of the dashboard to filter by period of time, to select a certain time gap or even to filter according to some customer related characteristics;

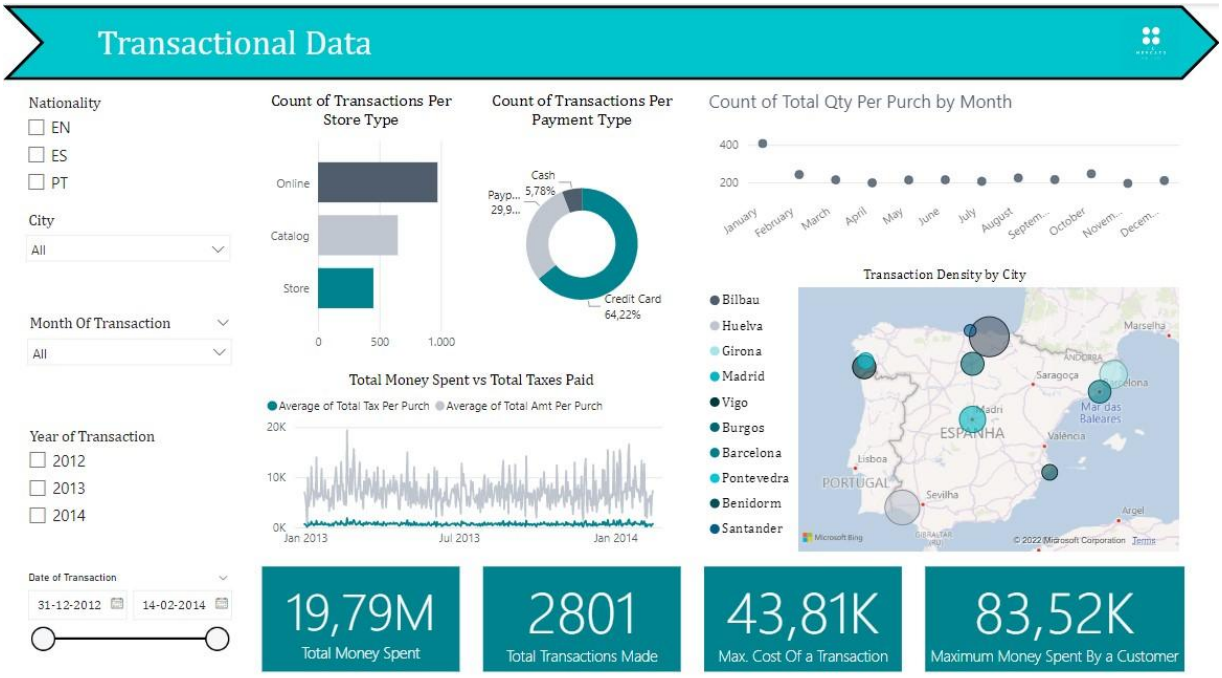


Fig. 30 - Transactional dashboard

- **Product data:** in this final dashboard, the charts inserted all regard information about the products sold by *Il Mercato* with the possibility of filtering by product category, subcategory, type of store or even the nationality of the company's customers.

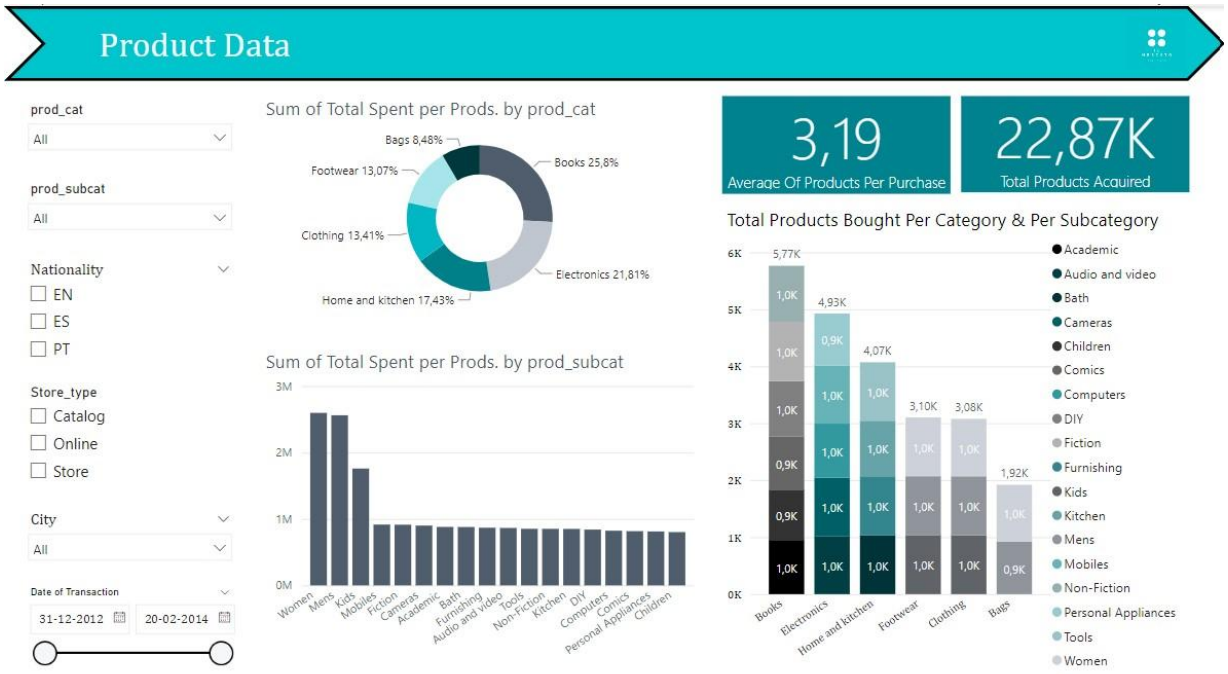


Fig. 31 - Product dashboard

It is to be noted that every dashboard has the logo of the company in the top right corner, and in every case (other than the index) this logo is also a button that serves as a bookmark that will return the user to the Dashboard Index

# Conclusion

In the end, it can be concluded that the team was able to treat every inconsistency and eliminate the outliers existent in the original dataset provided by the company.

These treatments gave way to the creation of dashboards that provided abundant and accurate information about *Il Mercato*'s customers, transactions and products.

We hope that the tools that will be delivered alongside this report will provide great value to the company and aid it in its future decisions.

# Annexes

## Remaining figures

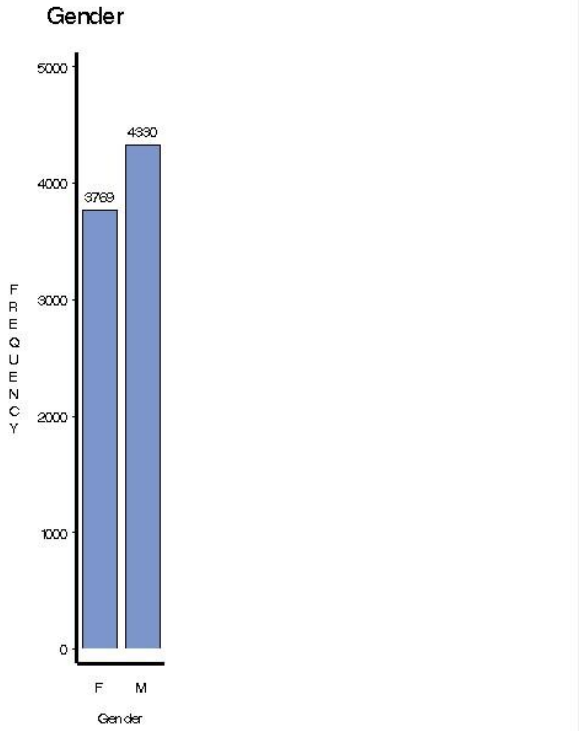


Fig. 32 - *Gender* distribution

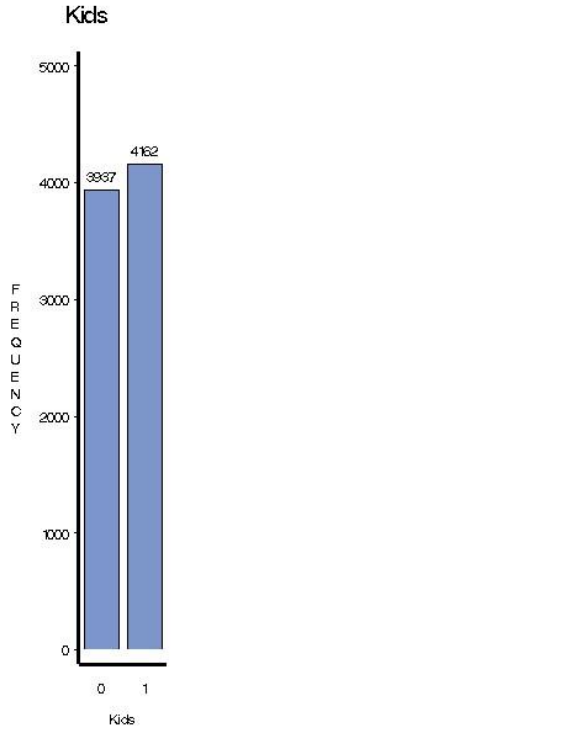


Fig. 33 - *Kids* distribution

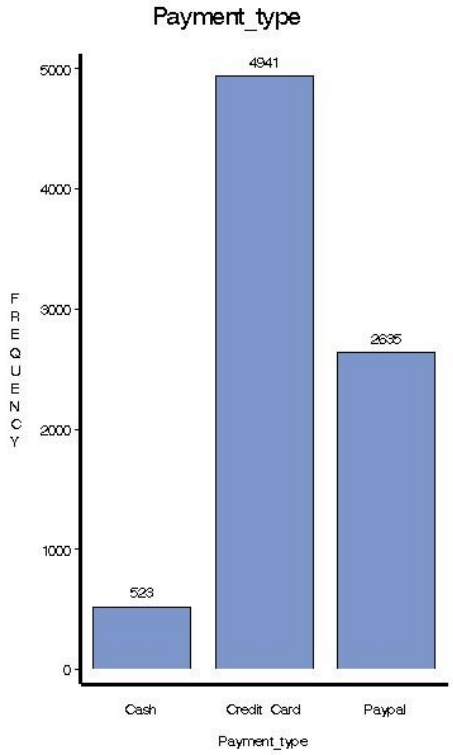


Fig. 34 - *Payment_type* distribution

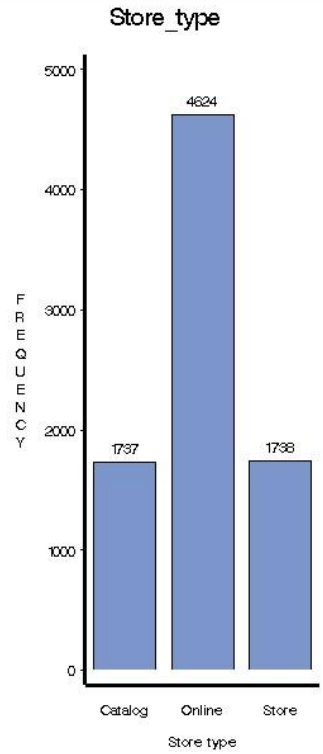


Fig. 35 - *Store_type* distribution

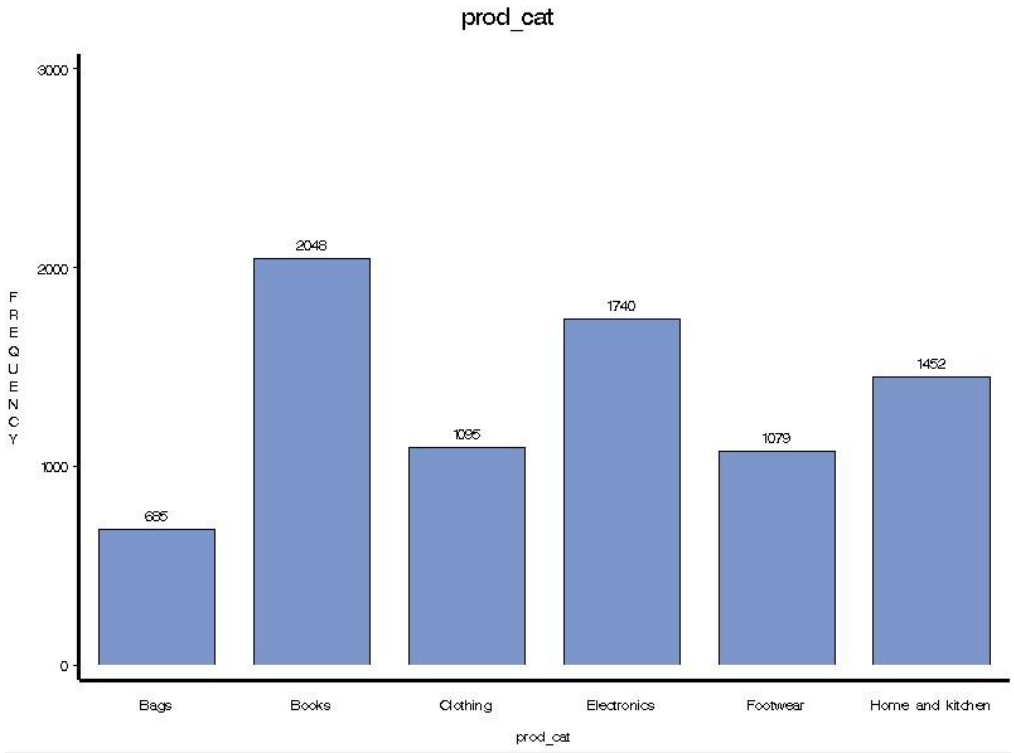


Fig. 36 - *prod_cat* distribution

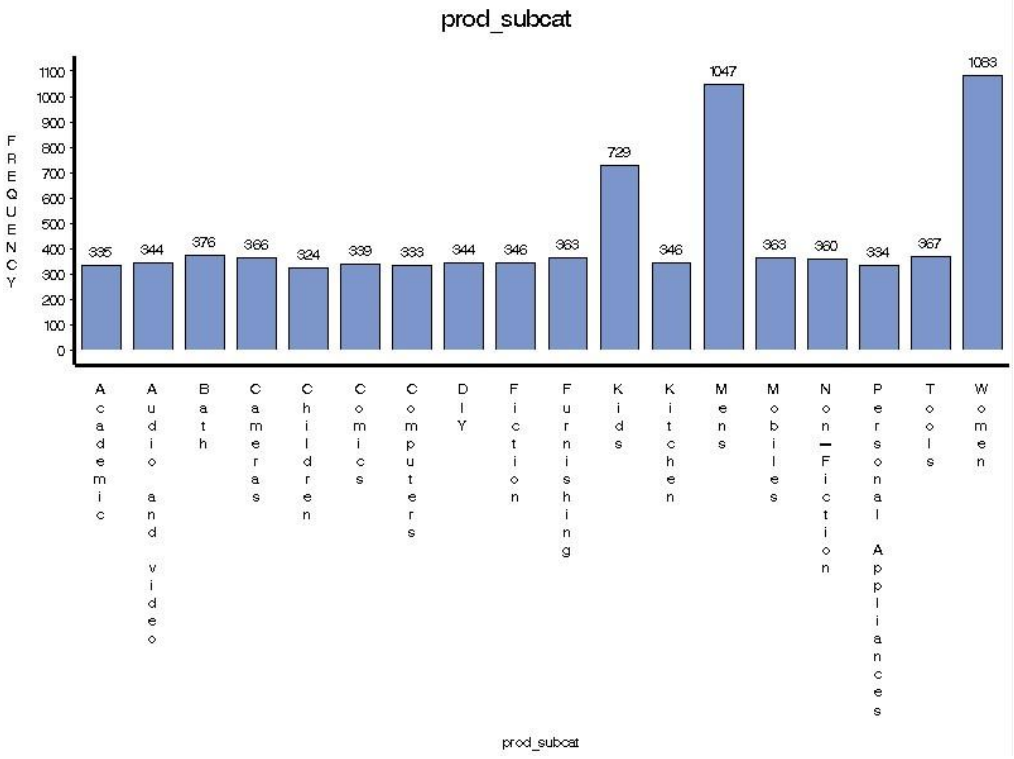


Fig. 37 - *prod_subcat* distribution

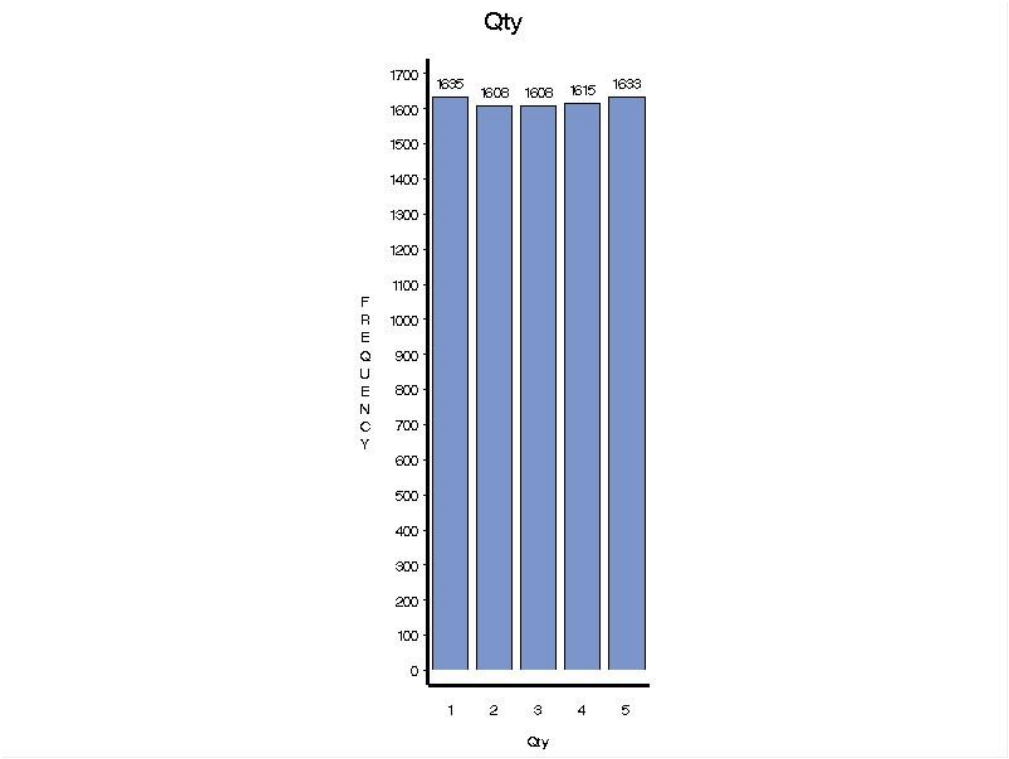


Fig. 38 - *Qty* distribution

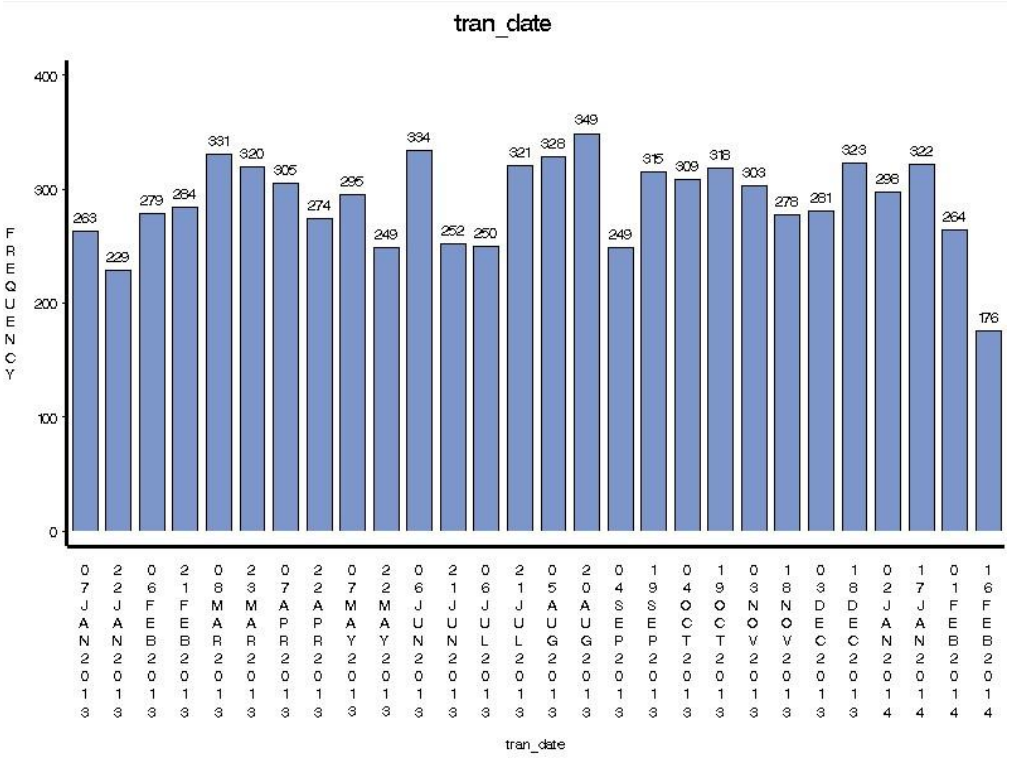


Fig. 39 - *tran_date* distribution


Fig. 40 - *Customer_since* outliers treatment

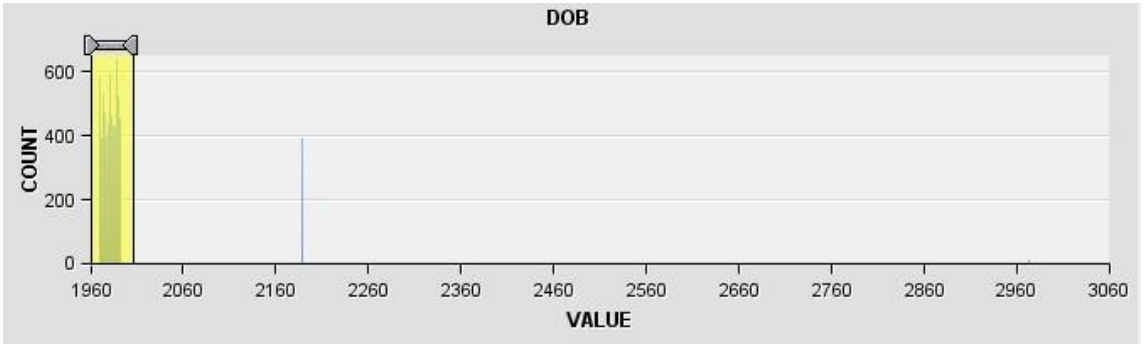


Fig. 41 - *DOB* outliers treatment

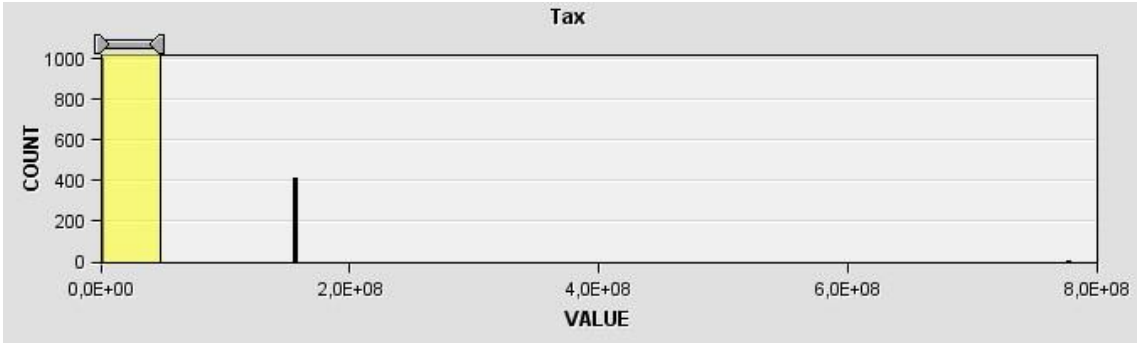


Fig. 42 - *Tax* outliers treatment

# Python code

```
##### phase 2

### import libraries
import numpy as np
import pandas as pd


### import datasets
path = "C:\\Users\\Afonso Cadete\\O meu disco\\Universidade\\2nd year\\1st semester\\Data Preprocessing\\Project\\"


# import phase 1 data
data = pd.read_excel(path + "project_data_phase1.xlsx")
data.drop(['prod_subcat_code', 'prod_cat_code', 'city_code'], axis = 1, inplace = True) #unnecessary

# import original data
data_original = pd.read_excel(path + "project_data.xlsx")


### reset formats
## date variables
# create dataframe of indexes and date variables
dates = data_original[['cust_id', 'transaction_id', 'DOB', 'tran_date']].copy()

# singularize transactions
dates = dates.drop_duplicates(['cust_id', 'transaction_id'], keep = 'last')

# set 'cust_id' and 'transaction_id' as index for both datasets
dates = dates.set_index(['cust_id', 'transaction_id'])
data = data.set_index(['cust_id', 'transaction_id'])

# transfer the original format to the current dataset
data[['tran_date', 'DOB']] = dates[['tran_date', 'DOB']].copy()

# sort values from oldest to most recent transaction
data = data.sort_values('tran_date')

# reset index
data.reset_index(inplace = True)

## dots to missing values
data['Tax'][data['Tax'] == '.'] = np.nan
data['total_amt'][data['total_amt'] == '.'] = np.nan

## 'Tax' and 'total_amt' data type
data['Tax'] = pd.to_numeric(data['Tax'], errors = 'coerce')
data['total_amt'] = pd.to_numeric(data['total_amt'], errors = 'coerce')

### inconsistencies treatment
## duplicates
print(data.duplicated().value_counts()) #0
```

```
## ´Tax' and 'total_amt'
data['Tax'] = data['Tax'].round(2)
data['total_amt'] = data['total_amt'].round(2)

## 'Customer_since'
# set 'cust_id' as index
data.set_index('cust_id', inplace = True)

# sort values from oldest to most recent customer acquisition
data.sort_values('Customer_since', inplace = True)

# check how many customers has more than one entry year
since_agg = data.groupby('cust_id')['Customer_since'].nunique()
display(since_agg[since_agg > 1]) #length:783
print(data.loc[since_agg == 1]['Customer_since'].value_counts())

# set 'Customer_since' with the oldest year associated with each customer
fill_dates = data.reset_index().drop_duplicates(['cust_id'], keep='first').set_index(['cust_id'])['Customer_since'].copy()
data['Customer_since'] = fill_dates.copy()

# dataframe of the years of the transactions
tran_year = pd.DatetimeIndex(data['tran_date']).year

# transactions which its year is older than 'Customer_since'
outliers_index = data[data['Customer_since'] > tran_year].index.drop_duplicates()
display(data.loc[outliers_index][['transaction_id', 'tran_date', 'Customer_since']]) #length:26

# verifying 'Customer_since' value frequencies
print(data['Customer_since'].value_counts()) #2013->0 2012->19

# verifying 'tran_date' of customers who entry in 2012
print(tran_year[(data['Customer_since'] == 2012)]) #2013(all)

# remove customers with outliers in 'Customer_since'
data.drop(data.loc[outliers_index].index, axis = 0, inplace = True)

# reset index
data.reset_index(inplace = True)

## DOB
# eliminate customers who shopped before the age of 16
print(len(data[(data['Customer_since'] - pd.DatetimeIndex(data['DOB']).year) < 16])) #124
data.drop(data[(data['Customer_since'] - pd.DatetimeIndex(data['DOB']).year) < 16].index, axis = 0, inplace = True)

# sort values from oldest to most recent transaction
data = data.sort_values('tran_date')

## 'Nationality'
# case 1 example
display(data[data['cust_id'] == 266816][['cust_id', 'transaction_id', 'Nationality']])

# case2 example
display(data[data['cust_id'] == 266806][['cust_id', 'transaction_id', 'Nationality']])
```

```
# case3 example
display(data[data['cust_id'] == 267067][['cust_id', 'transaction_id', 'Nationality']])

# check how many customers has more than one nationality
nation_count = pd.DataFrame(data.groupby(["cust_id"])["Nationality"].nunique())
display(nation_count[nation_count['Nationality'] > 1]) #length:81

# set 'Nationality' with the last one of all customers
# customers who only have missing values are not counted
existing_nations = data[data['Nationality'].isna() == False].drop_duplicates('cust_id', keep = 'last')[['cust_id',
'Nationality']].copy()
display(existing_nations) #length:822
data = data.set_index('cust_id')
existing_nations = existing_nations.set_index('cust_id')
data['Nationality'] = existing_nations['Nationality'].copy()

# check the number of rows with missing values and its percentage of the dataset
print(data['Nationality'].isna().sum()) #length:2285
print(f'Percentage of Missing Values: {round(2285 / data.shape[0] * 100, 2)}%') #29.89%

# verifying if changes result
nation_count = pd.DataFrame(data.groupby(["cust_id"])["Nationality"].nunique())
display(nation_count[nation_count['Nationality'] > 1]) #length:0

# 'Nationality' distribution
data['Nationality'].value_counts() #EN:4425 ES:551 PT:508

# reset_index
data.reset_index(inplace = True)

## 'Kids'
# example
display(data[(data['cust_id'] == 266806) & (data['transaction_id'] == 14141098432)][['cust_id', 'transaction_id', 'Kids']])

# define 'Kids' mode per transaction
kids_per_transaction = pd.DataFrame(data.groupby(["cust_id", 'transaction_id'])["Kids"].agg(lambda x:
pd.Series.mode(x)))

# append 'transaction_id' to index
data.set_index(['cust_id', 'transaction_id'], inplace = True)

# dataframe of 'kids_per_transaction' with clear mode
kids_clear = kids_per_transaction[(kids_per_transaction['Kids'].isin([0,1]) == True)]
display(kids_clear) #length:1712

# set 'Kids' column with clear values
data['Kids'] = kids_clear.copy()

# fill the missing values of each customer by the appropriate non-missing transaction mode
full_kids = data.groupby('cust_id')['Kids'].apply(lambda x : x.ffill().bfill())

fill_kids = full_kids.reset_index().drop_duplicates(['cust_id', 'transaction_id'], keep='first').set_index(['cust_id',
'transaction_id']).copy()
```

```
data['Kids'] = fill_kids.copy()

# percentage of missing values
print(f'Percentage of Missing Values: {round(int(full_kids.isna().sum()) / data.shape[0] * 100, 2)}%') #12.4%

# reset index
data.reset_index(inplace = True)


### checking data
display(data.info())

### export dataset to excel
data.to_excel(path + "project_data_phase2.xlsx")




### phase 4

### import libraries
import numpy as np
import pandas as pd


### import datasets
path = "/Users/marcelojunior/Downloads/Phase2_PPDV/Final Tests/"


# import phase 3 data
data = pd.read_excel(path + "project_data_phase3.xlsx")

# import original data
data_original = pd.read_excel(path + "project_data.xlsx")


### reset formats
## date variables
# create dataframe of indexes and date variables
dates = data_original[['cust_id', 'transaction_id', 'DOB', 'tran_date']].copy()

# singularize transactions
dates = dates.drop_duplicates(['cust_id', 'transaction_id'], keep = 'last')

# set 'cust_id' and 'transaction_id' as index for both datasets
dates = dates.set_index(['cust_id', 'transaction_id'])
data = data.set_index(['cust_id', 'transaction_id'])

# transfer the original format to the current dataset
data[['tran_date', 'DOB']] = dates[['tran_date', 'DOB']].copy()

# sort values from oldest to most recent transaction
data = data.sort_values('tran_date')
```

```
# reset index
data.reset_index(inplace = True)


### eliminating missing values
data['Tax'] = data['Imputed: Tax'].round(2).copy()
data['total_amt'] = data['Imputed: total_amt'].round(2).copy()
data['Nationality'] = data['Imputed: Nationality'].copy()
data['Kids'] = data['Imputed: Kids'].copy()
data['city'] = data['Imputed: city'].copy()


### discard unnecessary columns
data.drop(['A', 'Segment Id', 'Distance', 'Segment Description', 'Warnings', 'Imputed: Tax', 'Imputed: total_amt',
'Imputed: Nationality', 'Imputed: Kids', 'Imputed: city'], axis = 1, inplace = True)


### checking inconsistencies
## ‘Nationality’
nation_count = pd.DataFrame(data.groupby(["cust_id"])["Nationality"].nunique())
print(len(nation_count[nation_count['Nationality'] > 1])) #0

## 'Kids'
# define 'Kids' mode per transaction
kids_per_transaction = pd.DataFrame(data.groupby(["cust_id", 'transaction_id'])["Kids"].agg(lambda x:
pd.Series.mode(x)))
kids_test = kids_per_transaction[(kids_per_transaction['Kids'].isin([0,1]) == False)]
print(len(kids_test)) #235

# append 'transaction_id' to index
data.set_index(['cust_id', 'transaction_id'], inplace = True)

# dataframe of 'kids_per_transaction' with clear mode
kids_clear = kids_per_transaction[(kids_per_transaction['Kids'].isin([0,1]) == True)]
len(kids_clear) #2459

# set 'Kids' column with clear values
data['Kids'] = kids_clear.copy()

# fill the missing values of each customer by the appropriate non-missing transaction mode
# set -1 in case it is inconclusive
full_kids = data.groupby('cust_id')['Kids'].apply(lambda x : x.ffill().bfill().fillna(-1))
fill_kids = full_kids.reset_index().drop_duplicates(['cust_id', 'transaction_id'], keep='first').set_index(['cust_id',
'transaction_id']).copy()
data['Kids'] = fill_kids.copy()

# verifying the distribution
full_kids = pd.DataFrame(full_kids)
print(full_kids.value_counts()) #1-> 3504 0-> 3197 -1-> 944

# percentage of missing values
print(f'Percentage of Missing Values: {round(int(full_kids.isna().sum()) / data.shape[0] * 100, 2)}%') #12.4%

# reset index
```

```
data.reset_index(inplace = True)


### export dataset
data.to_excel(path + 'project_data_phase4.xlsx')
print('\nRun Completed!')
```