Analysis of *Customer Supermarket Dataset*

Data Mining
AY 2020/2021

Elia Piccoli 621332
Nicola Gugole 619625
Alex Pasquali 626378

Index

1. Introduction

The aim of this report is exploring the various phases of a Market Basket Analysis conducted on a given dataset. In particular the objective is to extract the customer behaviour by working on a derived dataset which is constructed from the original one.
The analysis is composed of:

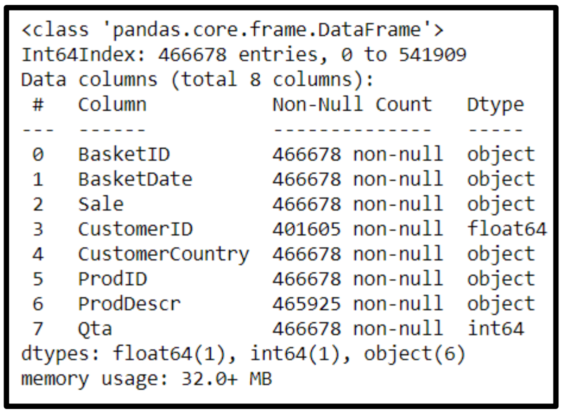
- *Data Understanding and Preparation*: original dataset has to be cleaned. Looking at data distribution, missing values, correlation between attributes is fundamental to the semantical understanding of the data and the following cleaning. It is also extremely important for preparing and correctly computing the customer's statistics needed for the derived dataset construction.
- *Customer Dataset Construction*: once the original dataset is cleaned it is possible to create the various indexes which compose the derived dataset.
- *Clustering*: the use of 5 different algorithms (K-Means, DBSCAN, Hierarchical Clustering, Fuzzy C-Means, Genetic Algorithm) leads to finding groups of customers with similar properties and therefore defining some behavioural description of customer groups.
- *Classification*: exploit various models - such as Decision Tree, Random Forest, etc. - to generalize and classify the customers relying on their behaviours.
- *Sequential Pattern Mining*: analyse common patterns in customers transaction to derive observations on people and correlations between products purchases.

2. Data Understanding and Preparation

The original dataset is composed of **466678** entries representing part of a basket transaction.

2.1 Data Semantics

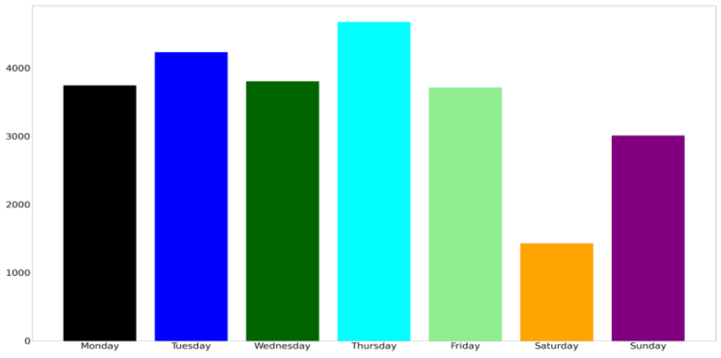
Taking a closer look at the 8 attributes:



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 466678 entries, 0 to 541909
Data columns (total 8 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   BasketID         466678 non-null  object
 1   BasketDate       466678 non-null  object
 2   Sale             466678 non-null  object
 3   CustomerID       401605 non-null  float64
 4   CustomerCountry  466678 non-null  object
 5   ProdID           466678 non-null  object
 6   ProdDescr        465925 non-null  object
 7   Qta              466678 non-null  int64
dtypes: float64(1), int64(1), object(6)
memory usage: 32.0+ MB
```

BasketID:
Basket identifier composed of a total of **24627** unique values. Object attribute since not all basketIDs are numerical, more specifically they are either:

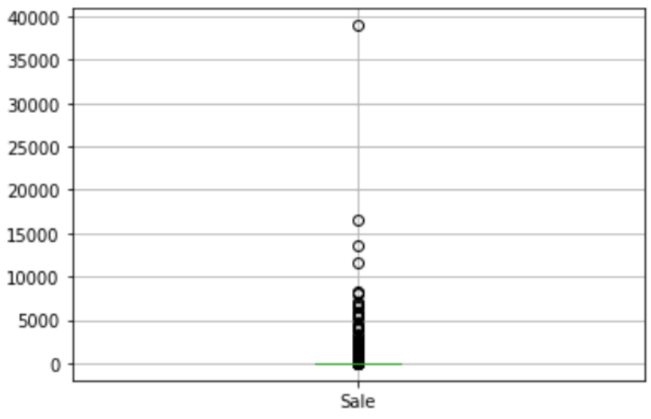
- Fully numerical: positive Qta
- Starting with C: negativa Qta
- Starting with A: only 2 entries, both having negative Qta and ProdDescr *Adjust bad debt*



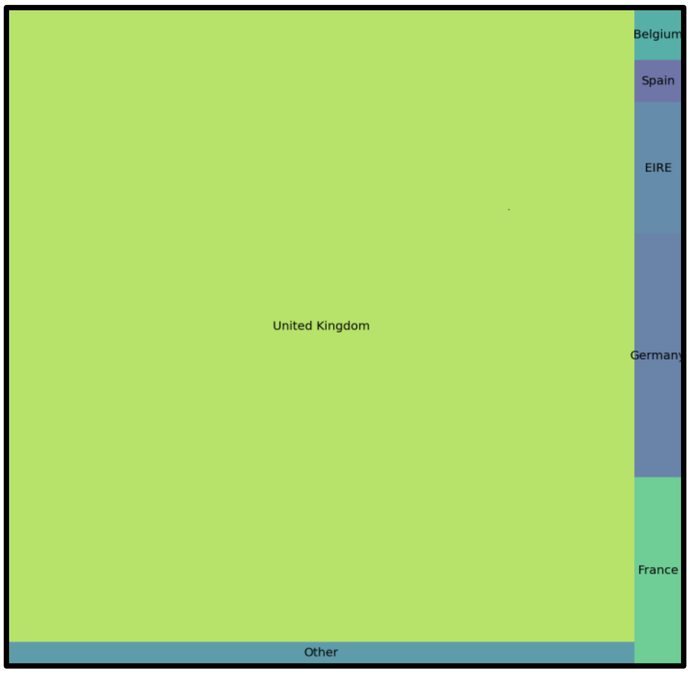
BasketDate:
Timestamp of the basket transaction entry. Object attribute defining the period of observation (01/12/10 08:26 - 09/12/11 12:50).

On the left: number of orders per weekday.

***Sale**:*
Cost per single product.
Object attribute (not float because decimal separator in the dataset is **comma** instead of **dot**).



CustomerID:
Customer identifier, composed of **4372** unique values, will be the index of the derived dataset.
Float64 attribute since for some arcane reason each customer terminates with the suffix '.0'

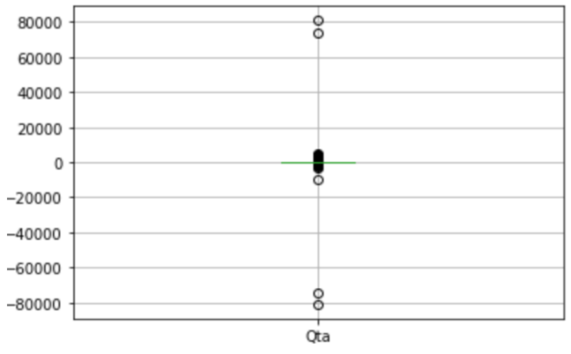


***CustomerCountry**:*
Customer country composed of **37** possible countries.
Object attribute, a string which is the name of the country.

On the left a tree map representing the number of orders per country, showing the predominance of the UK in the market.

***ProdID**:*
Product identifier composed of **3880** unique values.
Object attribute since these identifiers are alphanumeric strings.

***ProdDescr**:*
Brief product description composed of **4007** unique values.
Object attribute, together with *CustomerID* is the only attribute having missing values.



***Qta**:*
Product quantity, ideally it can be either:
- positive when buying the product
- negative when returning the product

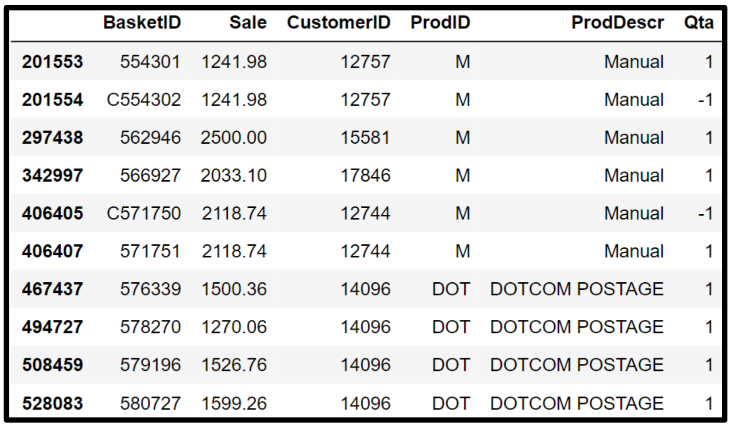
Object attribute, boxplot shows it needs some analysis.

2.2 Data Cleaning and Data Transformation

Data semantics already showed that there are errors and impurities inside the dataset: missing values, data with wrong types, data which needs transformation. Before proceeding to the customer dataset creation, the data quality needs to be assured.

In order to do so, various filters and transformations were applied:

- *Sale* attribute has been transformed to floating point by converting the decimal separator from **comma** to **dot**.
- Entries with *Sale* values too low to be relevant (e.g. Sale < 0.01) were removed. As a side effect the resulting dataset had no more missing values regarding *Prod-Descr*.
- Missing *CustomerIDs* were removed from the dataset since without this attribute there is not an index for the customer dataset records.



	BasketID	Sale	CustomerID	ProdID	ProdDescr	Qta
201553	554301	1241.98	12757	M	Manual	1
201554	C554302	1241.98	12757	M	Manual	-1
297438	562946	2500.00	15581	M	Manual	1
342997	566927	2033.10	17846	M	Manual	1
406405	C571750	2118.74	12744	M	Manual	-1
406407	571751	2118.74	12744	M	Manual	1
467437	576339	1500.36	14096	DOT	DOTCOM POSTAGE	1
494727	578270	1270.06	14096	DOT	DOTCOM POSTAGE	1
508459	579196	1526.76	14096	DOT	DOTCOM POSTAGE	1
528083	580727	1599.26	14096	DOT	DOTCOM POSTAGE	1

- Arcane '.0' suffix was also removed from *CustomerID* after making sure that all customers had the same strange suffix.
- *Qta* attribute is the most delicate attribute since it can be negative. An important decision was made to decide when the attribute negativity was to consider legal, either:
  - *Qta* value is referring to a discounted product
  - *Qta* negative value is matched (for that same product and that same customer!) by a positive *Qta* value which is at least of the same magnitude. If the magnitude of negative *Qta* is higher than the corresponding positive there is no plausible meaning to give to that product: the returned quantity would be higher than the bought one, impossible in the dataset period of observation!

  Therefore, each record not satisfying the above conditions was removed from the dataset.
- Further outliers in *Qta* not removed by the previous filtering were dropped.
- Outlier analysis in *Sale* highlighted some particular *ProdID* values which required closer studies: ['POST' 'D' 'M' 'BANK CHARGES' 'DOT']
  - *BANK CHARGES:* by looking at these entries statistics it is plausible they represent some kind of bank payment that may not be strictly correlated to the customer sale analysis. Hence they were removed.
  - *D:* by looking at these entries *ProdDescr* ('Discount'), it is clear they regard some shopping-related discount on a particular customer, surely relevant for the analysis. Therefore they were left in the dataset.
  - *DOT/POST:* treated together because of their similar *ProdDescr* (both containing 'POSTAGE'), it is plausible these entries have something to do with postage payments, not necessarily correlated to the customer sale analysis. Therefore they were removed.
  - *M:* by looking at these entries *ProdDescr* ('Manual'), it is possible these records are referring to payments due/received (some of these are positive, some are negative) about manual works of some kind. Without any further information it is impossible to understand the true nature of these records and even more impossible to assure the relevancy for the analysis. Therefore they were removed.

By the end of this process the resulting dataset is cleaner and more meaningful, ready to be used for creating the customer dataset.

3. Customer Dataset

The derived customer dataset is composed of **4333** entries, one per customer. As a result of data cleaning some customers were deleted (at the beginning there were **4372** unique customers).

3.1 Indexes creation

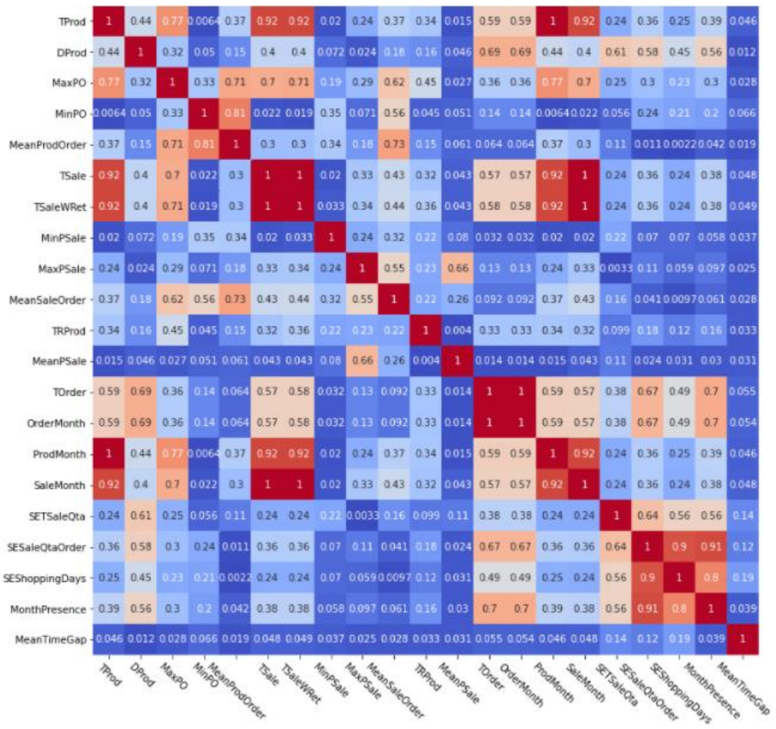
Several indexes were computed and added to the customer dataset, in particular the aim was to create a diversified set of dimensions, including statistical indexes such as mean, max, min of various original attributes, but also temporal indexes and even a trio exploiting Shannon entropy.

TProd: total number of items purchased by a customer during the period of observation.
DProd: total number of distinct items purchased by a customer during the period of observation.
TRProd: total number of products returned by a customer in the period of observation.
TSale: total amount spent during the period of observation.
TSaleWRet: total amount spent (without returned products) during the period of observation.
TOrder: total number of orders made by a customer during the period of observation.
MaxPO: maximum number of items purchased by a customer during a single shopping session.
MinPO: analogous to MaxPO.
MinPSale: minimum amount spent for a single product during the period of observation. Calculated by finding minimum *Qta*Sale*.
MaxPSale: analogous to MinPSale.
MeanPSale: mean amount spent for a single generic product during the period of observation. Calculated by finding mean *Qta*Sale*.
MeanProdOrder: mean number of items purchased by a customer during a shopping session.
MeanSaleOrder: mean amount spent for a generic order during the period of observation.
OrderMonth: mean monthly orders made by a customer during the period of observation.
ProdMonth: mean monthly products purchased by a customer during the period of observation.
SaleMonth: mean monthly amount spent by a customer during the period of observation.
SETSaleQta: Shannon Entropy for the variability of *Qta*Sale* in the period of observation.
SESaleQtaOrder: Shannon Entropy for the variability of *Qta*Sale* (accumulated per order) for a customer during the period of observation.
SEShoppingDays: Shannon Entropy to express the regularity of the weekday on which a customer goes shopping. (e.g. always on Monday).
MeanTimeGap: expresses the mean lapse of time (in days) in between 2 consecutive orders.
MonthPresence: percentage expressing how present the customer is in the period of observation based on month. Calculated as #months where customer is present / # total months.
MaxOrderMonth: categorical attribute, containing the month (as string of 3 characters) where the customer made the maximum amount of orders in the period of observation.
MaxOrderDay: categorical attribute, containing the weekday (as string of 3 characters) where the customer made the maximum amount of orders in the period of observation.

3.2 Dataset Correlation and Redundancy Analysis

Analyzing the correlation matrix (excluding *MaxOrderMonth* and *MaxOrderDay*, the only categorical attributes) it is clear there are some attributes too correlated one to the other. These attributes need to be dropped to avoid redundancy and bias in clustering.

Running a little script, the attributes over the correlation threshold (decided to be 0.75) are 13 over the total of 22. In particular, the attributes are: [*'TProd', 'MaxPO', 'MinPO', 'MeanProdOrder', 'TSale', 'TSaleWRet', 'TOrder', 'OrderMonth', 'ProdMonth', 'SaleMonth','SESaleQtaOrder', 'SEShoppingDays', 'MonthPresence*']



The final decision was to drop all these attributes apart from *TProd, TSale, TOrder* and *SESaleQtaOrder*. Motivations:

- *TProd* and *TSale* are indeed highly correlated but they have a different meaning which is interesting for both for the clustering and the post-clustering analysis phase.
- The others were highly correlated to the dropped attributes. Once the chosen ones were dropped their correlation array was below the selected correlation threshold.

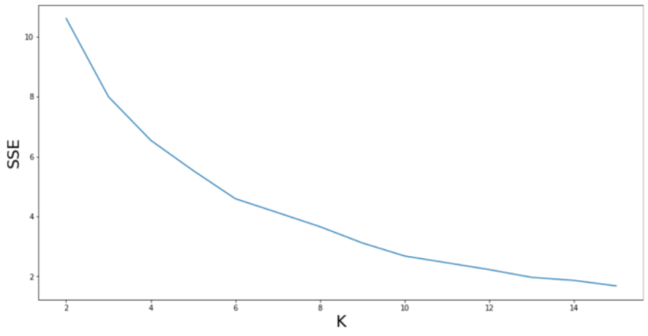
4. Clustering

4.1 K-Means

4.1.1 Choosing Attributes for Algorithm Run

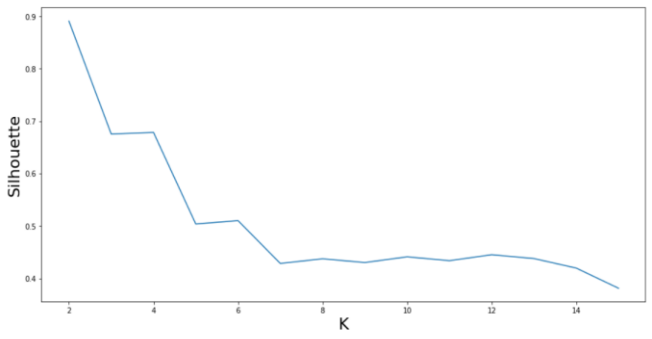
Only a few of the **13** remaining attributes had to be kept for the clustering (to avoid the curse of dimensionality and to keep out non characterizing attributes), and so a subset had to be decided. The decision was driven using analysis tools, logic and also a lot of trial and error.
After using the **PCA** (taking into consideration the most predominant subset of attributes in the PCA generation) as an analysis tool, the results were in, and they were not so good.
Following this first phase, a systematic trial and error produced a subset selection which was also a logic choice: the chosen subset contains the most diversifying (with also a bias to choosing the most robust) attributes in the customer dataset:
[*'TProd', 'MeanSaleOrder', 'MeanProdOrder', 'MeanPSale'*]

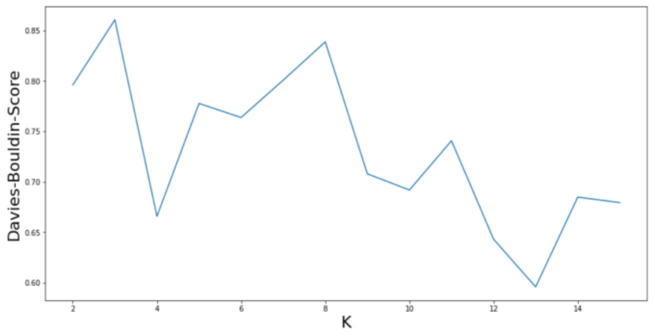
4.1.2 Choosing K Value



K is a fundamental parameter in K-Means clustering, it defines how many clusters will be the result of the algorithm. It is important to try many different runs to see the resulting metrics over a plethora of different K values.
After analyzing the metrics resulting from producing a number of clusters between **2** and **15**, the plots were strongly hinting to choose K = **4.**

This is due to the crossing of 3 parameters optimization: the informal elbow rule for the SSE metric (the lower this value is, the denser are the clusters), the need to keep as high as possible the Silhouette metric (the higher this value is, the farther are the clusters from each other), the need to keep as low as possible the Davies-Bouldin index (metric composed of a particular ratio of SSE and Silhouette).

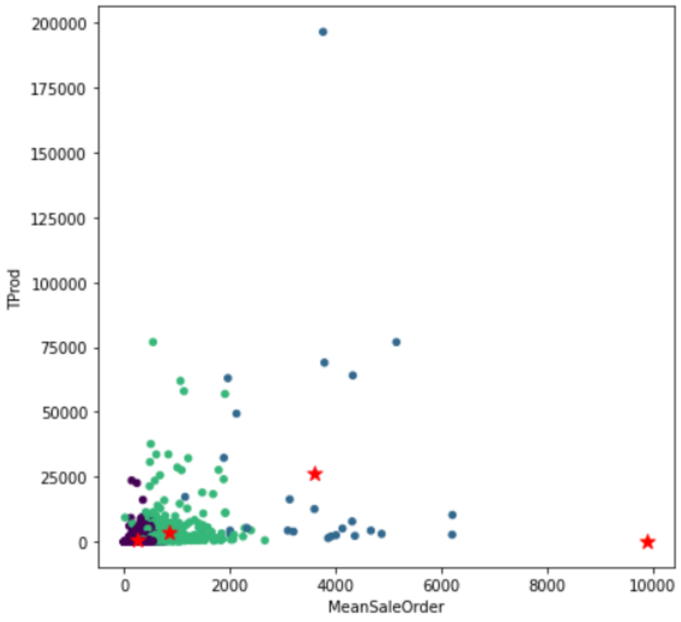




And in fact, the final decision was to consider K = 4, *SSE* = 6.54 and *Silhouette* = 0.68

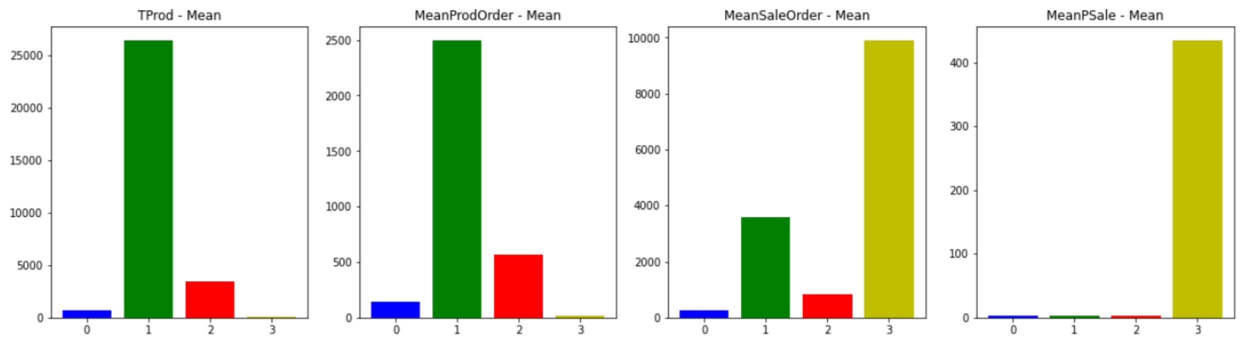
4.1.3 Analyzing Results

A scatter plot of the clusters over the attributes *Tprod* and *MeanSaleOrder* can quickly give an overview of the results, showing how **2** clusters are located near the origin and seem well defined, a **third** cluster seems to be capturing all the entries farther from the origin apart from one single value (positioned exactly below the rightmost red star) which by itself composes a **fourth** cluster.
By taking a look at the number of customers per cluster we can see that the four clusters include {0: **3857**, 1: **25**, 2: **450**, 3: **1**} for a total of **4333** customers.



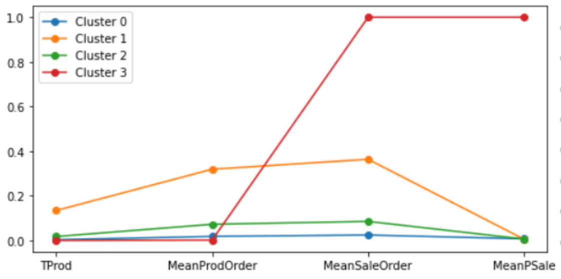
The presence of a cluster composed of only 1 customer is strange to say the least. More insight on the situation is given by the barplot below, which shows how the anomalous cluster has a peculiar behaviour, distinct from the other clusters which seem to follow a certain

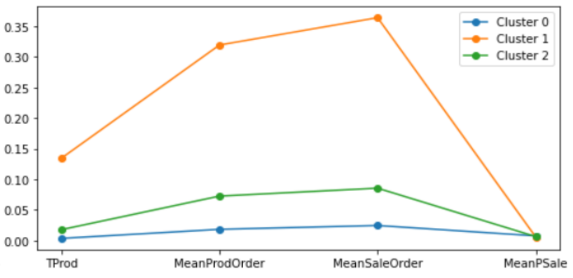
```
{0: 3857, 1: 25, 2: 450, 3: 1}
```



pattern in every attribute plot.

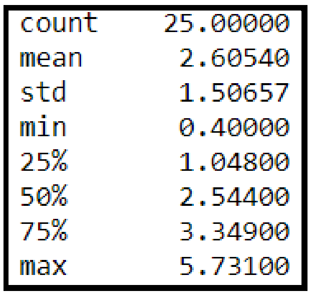
The suspect of cluster 3 being an anomaly is reinforced by looking at the centroids with and without the presence of said cluster. The trend of the non-anomalous clusters clearly follows a trapezoid shape while the anomalous one follows the trend dictated by its only customer, which is obviously not as robust and as meaningful as the other clusters trend.



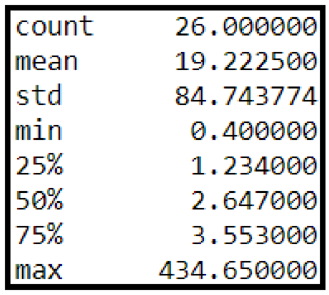


Some **postprocessing** is needed, there are 2 possible alternatives:

- *Merging* the anomalous cluster with the one most similar to it (in this case would be **cluster 1**). This leads to a deep variation in **cluster 1** statistics (e.g in **mean, std** and **max value**), as can be seen in the **before** and **after** below, which is about the *MeanPSale* attribute.
- *Discarding* the anomalous cluster, which was the definitive decision after the negative results brought by attempting merging.

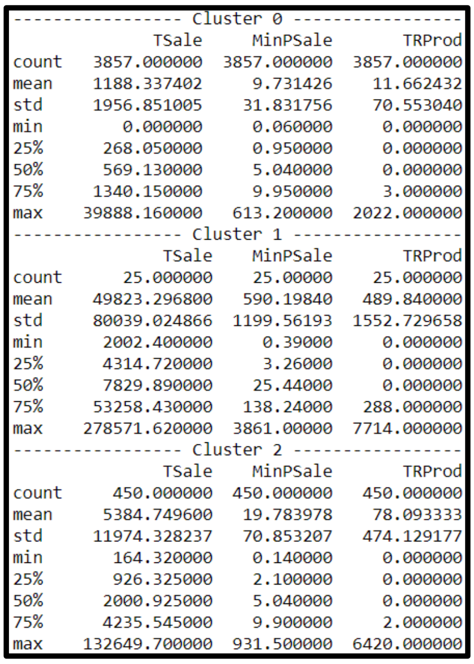


```
count    25.00000
mean      2.60540
std       1.50657
min       0.40000
25%       1.04800
50%       2.54400
75%       3.34900
max       5.73100
```



```
count     26.000000
mean      19.222500
std       84.743774
min        0.400000
25%        1.234000
50%        2.647000
75%        3.553000
max      434.650000
```

Statistical values change while merging the clusters

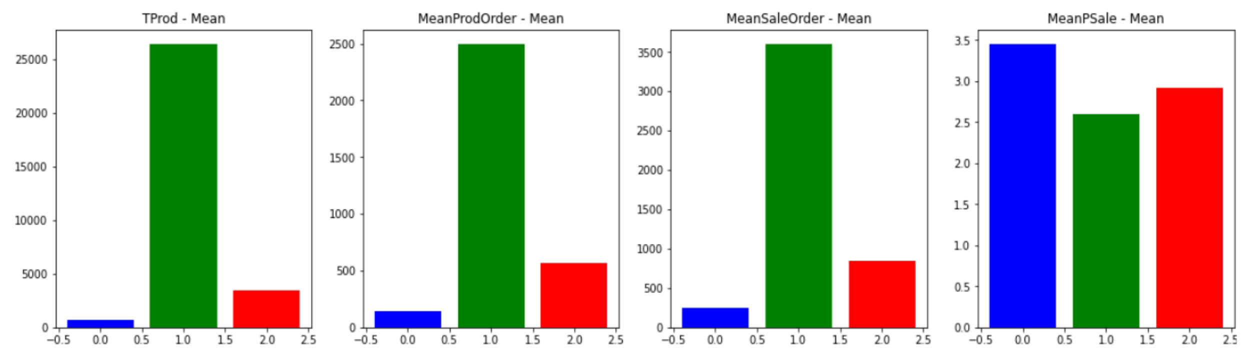


```
----------------- Cluster 0 -----------------
              TSale     MinPSale        TRProd
count   3857.000000  3857.000000  3857.000000
mean    1188.337402     9.731426    11.662432
std     1956.851005    31.831756    70.553040
min        0.000000     0.060000     0.000000
25%      268.050000     0.950000     0.000000
50%      569.130000     5.040000     0.000000
75%     1340.150000     9.950000     3.000000
max    39888.160000   613.200000  2022.000000
----------------- Cluster 1 -----------------
               TSale    MinPSale        TRProd
count      25.000000    25.00000     25.000000
mean    49823.296800   590.19840    489.840000
std     80039.024866  1199.56193   1552.729658
min      2002.400000     0.39000      0.000000
25%      4314.720000     3.26000      0.000000
50%      7829.890000    25.44000      0.000000
75%     53258.430000   138.24000    288.000000
max    278571.620000  3861.00000   7714.000000
----------------- Cluster 2 -----------------
               TSale    MinPSale        TRProd
count     450.000000  450.000000    450.000000
mean     5384.749600   19.783978     78.093333
std     11974.328237   70.853207    474.129177
min       164.320000    0.140000      0.000000
25%       926.325000    2.100000      0.000000
50%      2000.925000    5.040000      0.000000
75%      4235.545000    9.900000      2.000000
max    132649.700000  931.500000   6420.000000
```

After discarding the anomalous cluster, the result is composed of **3** clusters, each of them describing a customer behaviour:

- **Cluster 0:** containing the highest number of customers (**3857**, **89%** of the whole population), this cluster can easily define the group of **low-spending** customers, having lowest average total sale (**1188**) and lowest average number of products per order (**143**).
- **Cluster 1:** containing the lowest number of customers (**25**, **0.58%** of the whole population), this cluster can very well define the group of **high-spending** customers, having by far the highest average total sale (**49823**) and highest average number of products per order (**2497**). Interestingly, it is also the cluster with the highest average number of returned products (**490** per customer).
- **Cluster 2:** containing the value in the middle regarding number of customers (**450**, **10%** of the whole population), this cluster can define the group of **medium-spending** customers, having average total sale (**5385**) and average number of products per order (**568**).

```
{0: 3857, 1: 25, 2: 450}
```

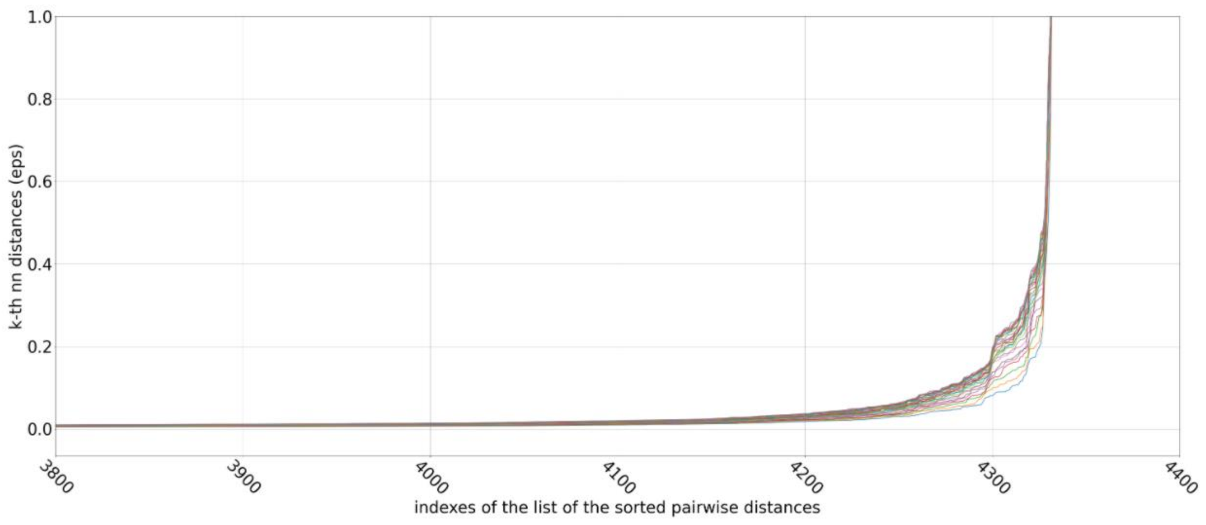


4.2 DBSCAN

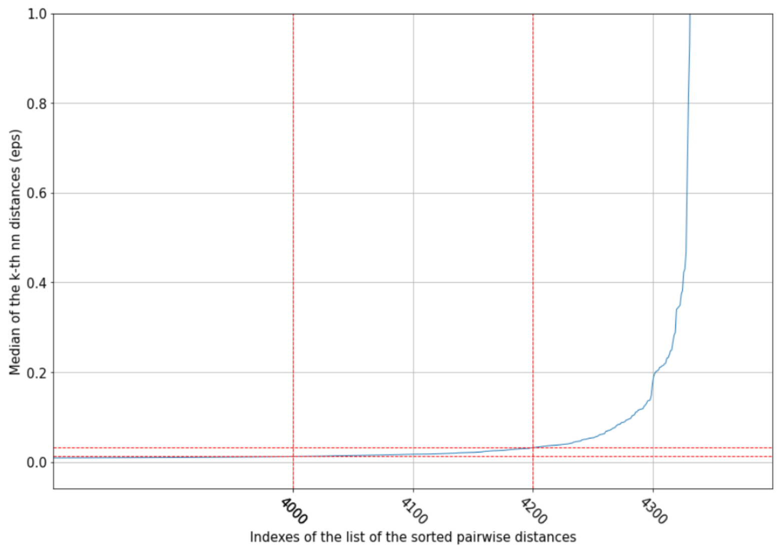
The dataframe used for DBSCAN is the same used for K-Means, with the same selected subset of attributes. The distance between data points is computed using the *Euclidean* metric, resulting in a square matrix containing the pairwise distance between all pairs of points.
kth_distances is a dictionary having as keys all the integers between *Kmin* and *Kmax* (3 and 30) and as values it has lists containing the distance of the k-th nearest neighbour of each point.
For example:

- k = 3 → list containing the distance of each point from its 3rd nearest neighbor
- k = 4 → list containing the distance of each point from its 4th nearest neighbor
- and so on...

Each of these lists is sorted and plotted, making possible to notice that the curve of the distance behaves almost in the same way regardless of the value of K.



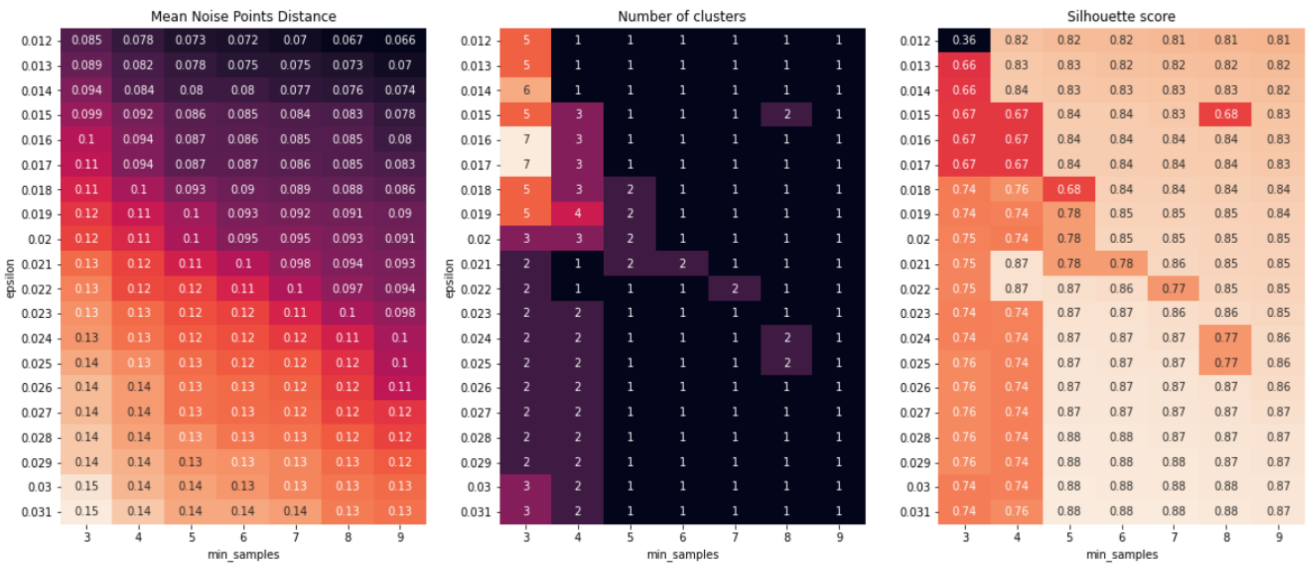
In order to visualize it as a single curve, there is a plot where, to each value in the x axis, is associated the median of the values in x of the *Kmin-Kmax* curves. The elbow is located between x=**4000** and x=**4200** with the corresponding values of eps between **0.0118** and **0.0313**.



At this point the grid search method is exploited to find the best values for the two parameters of the DBSCAN algorithm: *epsilon* and *min-samples*.
To optimize the search:

- it is applied only on those values of eps that are within the *elbow window*
- Since for *min-samples* greater than **8**, only a single cluster is detected, the grid search is restricted on the values between 3 and 9

As result, the following heatmaps are produced:



The best combination of parameters seems to be: *eps* = **0.02** and *min-samples* = **4**.
This will produce 3 clusters distributed as follows:

- Cluster 0: **4223** points
- Cluster 1: **4** points
- Cluster 2: **9** points
- There remain **97** noise points

Silhouette score: **0.74**
The result of this clustering algorithm applied to this dataset is very unbalanced, and different choices of the parameters do not improve the situation.

4.3 Hierarchical Clustering

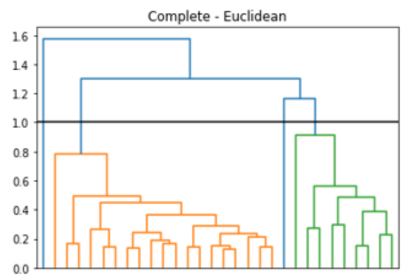
4.3.1 Attribute choice and distance methods

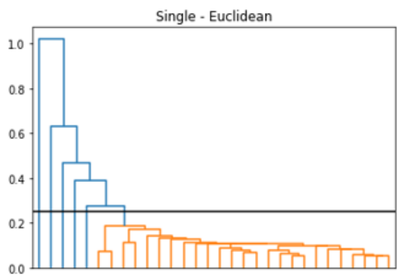
For the clustering via Hierarchical Clustering the same set of attributes of the previous algorithms were used, in order to get results comparable in terms of indicators and properties among the clusters. For all the different runs of the algorithm the metric was the euclidean one since it better fits our data. The analysis goes through different methods like *complete*, *single*, *average* and *ward*, each of them changes the way in which the distance between two clusters is calculated. The first three are based on mathematical functions while the last one has a more statistical approach.
Through the study of the dendrograms the best value for the threshold was chosen with the objective of getting a balanced clustering result. In order to evaluate the goodness of the clustering the silhouette coefficient was calculated.

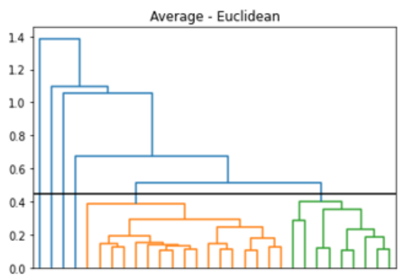
Method	**Clusters**	**Silhouette**
Complete	4311, 20, 1, 1	0.922
Single	4328, 1, 1, 1, 1, 1	0.935
Average	4312, 17, 1, 1, 1, 1	0.92
Ward	3853, 458, 22	0.672
Ward	3858, 458, 21, 1	0.672

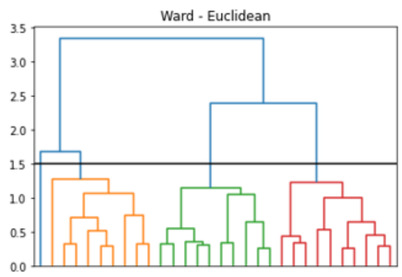
Even if the silhouette is very high, the first three methods' results are pretty bad since the clusters are unbalanced, collapsing all the elements in one single cluster. Best results are achieved with *ward method* both from the point of view of the number of elements in the different clusters and coherence of values among clusters too.

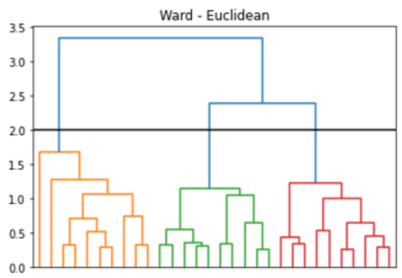
4.3.2 Analysis of dendrograms using different methods











As expected from the theoretical formulation of the different methods, *complete* joins clusters at a higher value than *single* and *average* since it evaluates the distance - between clusters - as the maximum distance between points of two different clusters. From the dendrograms and the results previously shown, the first three methods are not able to create a good and balanced clustering, all the items are collapsed in a singular big cluster. With Ward's method, instead, the result is more balanced, and it is possible to study a characterization of the elements inside the three main clusters. The results are similar to the ones obtained from K-Means, the three clusters seem to represent the three different types of customers: low, medium and high spending (keep in mind the last one can include outlier elements, which are customers who don't carry enough information for a good understanding).

4.4 Fuzzy C-Means

4.4.1 Introduction to the algorithm

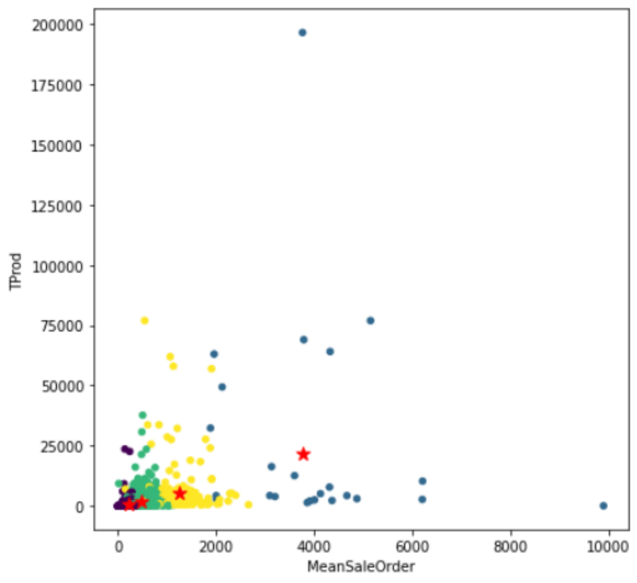
This alternative clustering approach is part of the family of so-called fuzzy clustering algorithms, which are based on the idea that an element can belong to more than one cluster (soft clustering). This idea is exploited using various coefficients, where each of them represents the probability of the element to be part of a certain cluster. The set of coefficients (one per cluster) is also known as membership matrix.
The algorithm in its implementation is very similar to K-Means:

- Choose C (K) number of clusters
- Initialize centroids, either randomly or by some criteria (**K-Means++** in this case)
- Compute membership function for each element and calculate new centroids
- Repeat the previous step until stopping criteria is reached

Once the algorithm terminates each element will be assigned to a cluster by looking at its highest membership values.

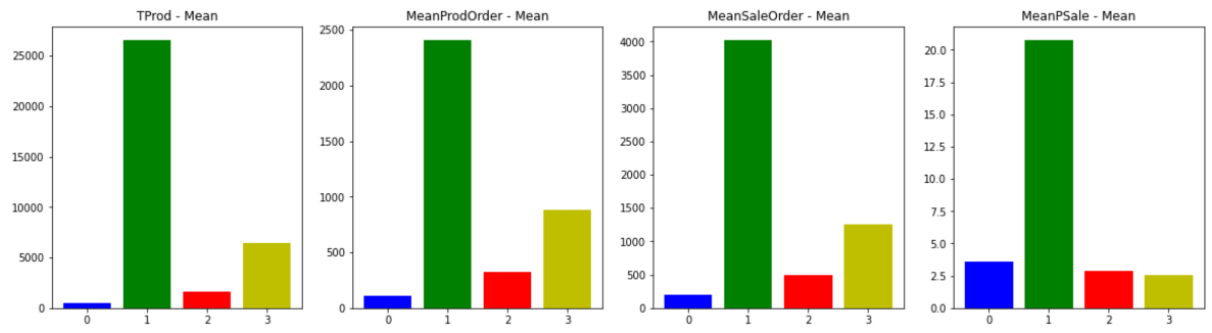
4.4.2 Analysis of results

It is true that the algorithm is different with respect to K-Means (soft vs hard clustering), but the underlying idea is pretty much the same. With this bias the decision regarding the number of clusters value (K) was straight forward: either 3 or 4.



Choosing the number of clusters to be equal to 4 brings a new clustering result. The resulting clusters are composed respectively of **3086, 24, 1086** and **137** customers**,** giving a substantially different result from the ones reported by K-Means and Hierarchical Clustering. It is a more smoothed out result, as to be expected by such a *soft* approach. Furthermore, by taking a look at the bar plots, it is plausible to semantically differentiate the clusters by individuating four possible behaviours: **low-spending (cluster 0), medium-spending (cluster 2), high-spending (cluster 3)** and **extremely high-spending (cluster 4).**
Talking about metrics this clustering has a Silhouette of roughly **0.5** and a Davies-Bouldin Index of **0.98.**

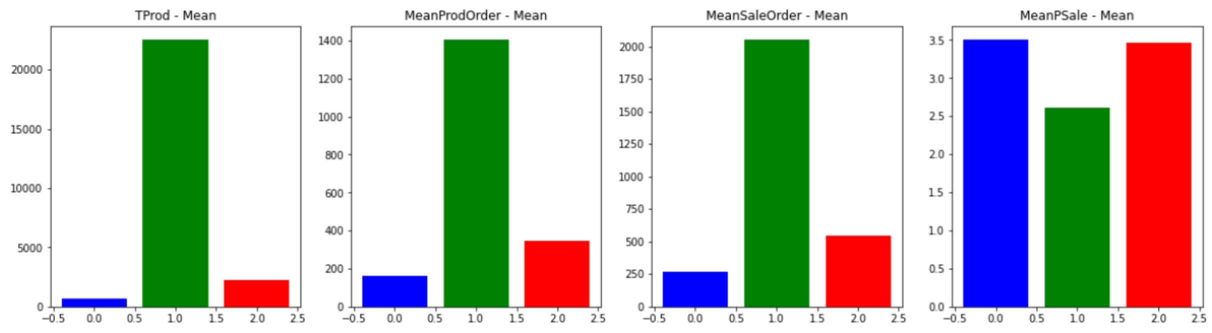
```
[0:3086] [1:24] [2:1086] [3:137]
```



Choosing the number of clusters to be equal to 3 (and deleting the **anomalous value** already found using K-Means) brings a clustering result very similar to the ones already portrayed by K-Means and Hierarchical Clustering. This means that the results suggest a distinction between the different clusters due to the kind of customers they contain (once again **low, medium** or **high spending**). One difference that can be easily highlighted between the Fuzzy and the K-Means result stays in the number of elements per cluster, which are different due to the algorithm being *soft*, tending to include elements which would not be included in the same cluster if the clustering algorithm was a *hard* kind.

Talking about metrics this clustering has a Silhouette of roughly **0.62** and a Davies-Bouldin Index of **0.86.** Confronting the two clustering (K = 3 and K = 4) the better metrics lies in the 3-cluster result.

```
[0:3681] [1:41] [2:610]
```



4.5 Clustering Genetic Algorithms

To go even further beyond in the analysis of the data a completely different approach was considered. This alternative method tries to combine GA (genetic algorithms) to clustering in order to create a hybrid technique that exploits the ability of exploration of GA to find the best possible cluster that fits the data. The problem of clustering is easily reinterpreted in a genetic point of view through the following mapping of the elements:

- *Population*: represents a set of L different subdivision of the elements in K clusters
- *Mutation rate*: probability of an element to be assign to a random cluster
- *Fitting function*: metric to evaluate and find the best representation, in this case will be based on statistics, such as variance, among the elements in the cluster
- *Evolution step*: choose the best 2 models from the population, crossover them to get the new model for the new generation

Given this formalization the algorithm will start with a random population and after several steps it will have reached the best possible clustering of the data. Unfortunately, this type of approach suffers from the fact that it is not possible to estimate if it will converge to a solution, and, if yes, after how many iterations. Even if different configurations were analyzed no good results were obtained.
This may be due to various reasons:

1. The number of iterations is too low, given this type of approach it is highly probable that a higher number of generations is needed in order to reach good results
2. The data is a bit noisy so this may cause bad evaluation of best model
3. Better initialization of the population since it is generated randomly but the result should follow a specific pattern. An initialization that meets the objective would be better in terms of results and convergence

4.6 Final consideration over clustering

In conclusion many clustering algorithms have been applied to the constructed customer dataset, some brought results, some brought interesting results, some unfortunately were not useful in this case.
More specifically:

- *K-Means* needed some postprocessing to delete a singular anomalous cluster but apart from that it produced a concrete and meaningful clustering, differentiating three possible customer behaviours.
- *Hierarchical Clustering* provided a plethora of possible solutions, proving itself not useful in this case when using *complete, single* and *average* methods. Nevertheless, the solutions provided using *Ward* method were incredibly similar to K-Means, pointing and hinting to the same possible customer behaviours.
- *Fuzzy C-Means* looked at the problem in a similar way compared to K-Means, but at the same time it turned the table using a *soft clustering* approach which resulted in smoother results. Concrete and meaningful clustering were its outcomes, outputting both a 3-cluster solution (highly similar to the K-Means/Hierarchical Clustering) and a 4-cluster solution. By comparing the metrics, the 3-cluster result is better than the 4-cluster one, leading once again to the approximate same solution as K-Means.
- *DBSCAN* unfortunately gives vastly unbalanced results, hinting that the dataset at hand may not be perfect for this kind of algorithm.
- *Clustering Genetic Algorithms* as highlighted before fail to get an appreciable result due to both the dataset values topology and to the tuning of the algorithm parameters.

5. Classification

5.1 Customer labelling and general introduction

Labelled data was collected from 2 different origins:

1. *KMeans Clustering*
2. *Fuzzy C-Means Clustering*

Both these results effectively gave 3 classes (with respective labels) which semantically correspond to *High_Spending, Med_Spending, Low_Spending* customers. On these datasets various common classification methods were therefore applied: *Decision Tree, Random Forest Classifier, Naïve Bayes Classifier, KNN, SVM* and eventually the big guns, *Neural Networks*.

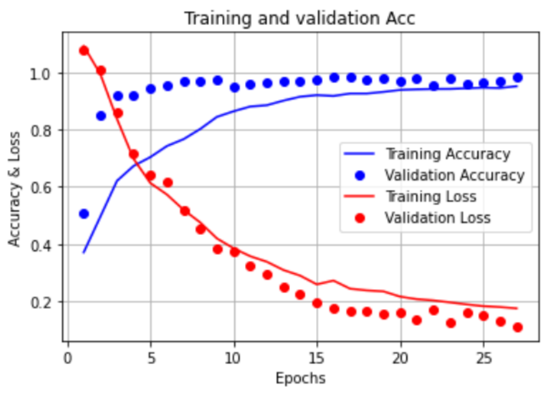
5.2 Working on KMeans computed labels

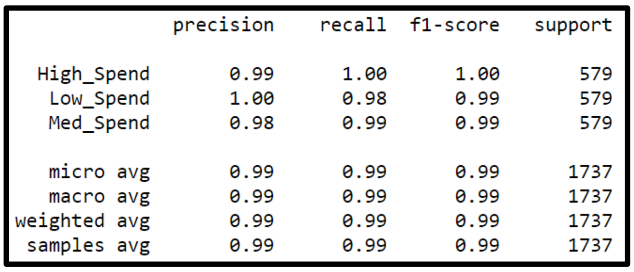
At a first glance an important fact can already be noticed: *High_Spend* class is incredibly unbalanced with respect to the other two (**26** elements while others have **449** and **3868**). This led the classification process to a deeper analysis through the useful tool of *stratification*. Three different classification analysis were carried on the dataset applying (or not) a particularly robust *oversampling* method. The best one - after looking in literature - was *SMOTE*, which makes use of original data and perturbation to create actual new data.
Looking at what the three analysis consist of:

- Classification on original dataset (*test set 30% at split*)
- Classification on oversampled dataset (*test set 30% at split, training set oversampled with SMOTE after splitting*)
- Classification on oversampled dataset (*test set 30% at split, training set and test set independently oversampled with SMOTE after splitting*)

Aware of the fact that oversampling a test set is not ideal since it leads to artificially vary otherwise “real” data, this decision was taken to try and suppress extreme variability of classification results on *High_Spend*, while in the meantime creating new random data. This was due to having a test cardinality regarding that class in between 5 and 10 samples (if the run was lucky!). All classification methods proved to be more or less susceptible to the low cardinality in *High_Spend* class, therefore having this kind of analysis where amount of data gets bigger and bigger brought out some interesting considerations on the various methods.

Decision Trees proved to be the most explainable of methods, and together with *Random Forest Classifier* showed robustness in each of the three analysis.
This means that over almost all the various attempts and runs these two methods were able to correctly identify the classes meaning, hence they were able to not break down on testing *High_Spend* class (other methods, e.g. *SVM*, showed no such ability in identifying low cardinality training data classes, giving **0** predictions on test set on said classes).
Naïve Bayes also showed incredibly high robustness by never breaking down, even when *High_Spend* class had extremely low cardinality. This has theoretical proof in being a probabilistic approach, so it is surely robust but does not obtain as good results as other methods.
Eventually *Neural Networks* proved to be able to obtain best results over all the approaches by applying oversampling on training set and test set (but not on validation set!).





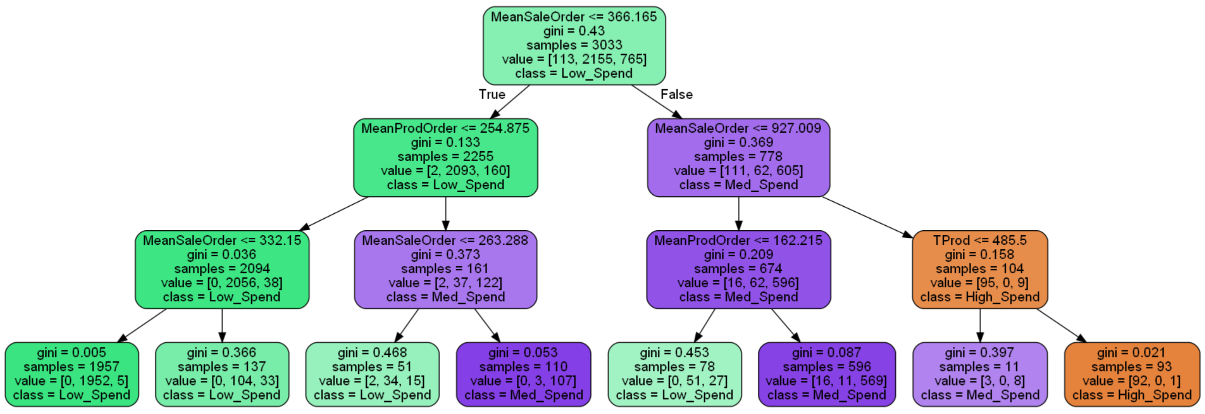
```
              precision    recall  f1-score   support

  High_Spend       0.99      1.00      1.00       579
   Low_Spend       1.00      0.98      0.99       579
   Med_Spend       0.98      0.99      0.99       579

   micro avg       0.99      0.99      0.99      1737
   macro avg       0.99      0.99      0.99      1737
weighted avg       0.99      0.99      0.99      1737
 samples avg       0.99      0.99      0.99      1737
```

5.3 Working on Fuzzy C-Means computed labels

Fuzzy C-Means output classes do not have the issues previously commented, meaning the *High_Spend* class is more populated. Obviously, this does not imply all three classes have almost same cardinality, but at the same time oversampling is surely not needed.
Considerations on classification methods do not change with respect to the ones previously made and it can be noticed that *SVM* method has a lower tendency to breaking down, confirming that the more the data, the merrier *SVM* is. Once again, *Neural Networks* bring the most stable and also best results, but again, this does not come as a shocker.
Right below a sample *Decision Tree* is shown as it can help to visualize how the classifier can take its decisions in this dataset.



5.4 Explainability of the models

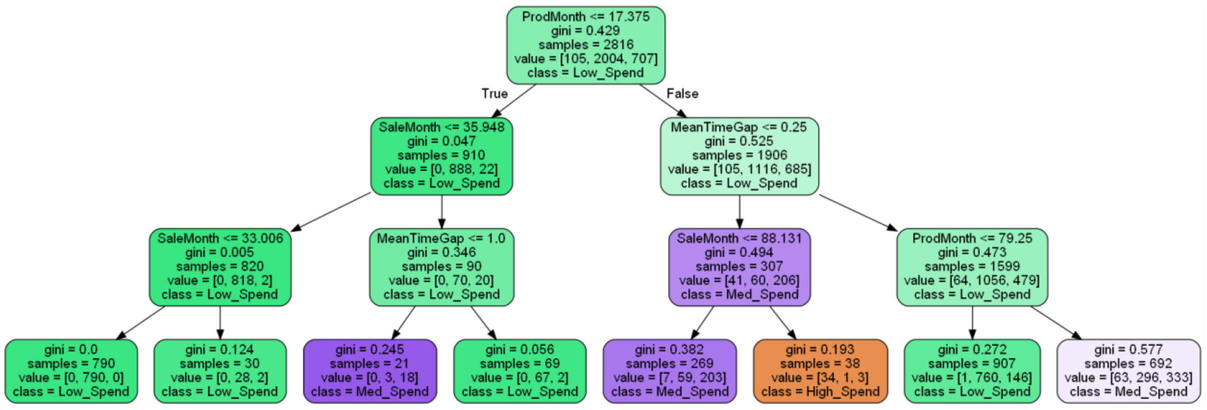
After the previous analysis of the models from the point of view of quantitative results, now the focus will shift over qualitative results. The discussion will be about the explainability of the model, so how well it describes the characterization of the customer with respect to his behaviour.
Two different results will be considered:

- *Decision Tree* built over Fuzzy C-Means (previous section)
- *Decision Tree* built over time dependent features not considered for clustering

Considering the first decision tree, it can be observed that the set of features over which the model bases its decisions are a subset of the ones that characterized the clustering process: *'TProd'*, *'MeanSaleOrder'*, *'MeanProdOrder'*.
To assure that the decision tree captured the correct generalization of the customers, various path through the tree can be followed and semantically analyzed.

Low Spend customer: Following the rules of the decision tree, customers are classified as **low** spending if they have an average amount of products and average sale per order which are in the leftmost part of the data distribution.
Medium Spend customer: Following the rules of the decision tree, customers are classified as **medium** spending if they have an average amount of products or average sale per order which are in the middle part of the data distribution, hence representing data having values not too low and not too high. As expected, this class has some elements behaving similarly to the other two classes up to a certain point. (e.g., left subtree mainly classifies low spending, but has a leaf which classifies the edge elements of medium spending)
High Spend customer: Following the rules of the decision tree, customers are classified as **high** spending if they have an average amount of products, an average sale per order and a total number of products which are in the rightmost part of the data distribution.

An alternative approach was taken into consideration to understand if a different explanation was possible on the customer classes. The second result focuses on this goal by removing the features that characterized the clustering process and maintaining only time dependent features.



Considering the resulting decision tree above it can be observed that the set of features over which the model bases its decisions are: *'ProdMonth, 'SaleMonth, 'MeanTimeGap'*.
Low Spend customer: customers are classified as **low** spending if they have a monthly number of products and monthly sale per order which are in the leftmost part of the data distribution and a frequency of shopping relatively low (*MeanTimeGap* high). For the other two classes we can derive analogous considerations as the previous model, changing the features upon which the classification is performed.

In both cases we obtain a model that portrays in a human understandable way how a certain class is represented. From a qualitative aspect the results are comparable, the real difference stands in the quantitative results. Since the two models are built upon the same labeling and with similar parameters, it is immediate to see how the first one returns a better classification for all the classes. Instead, the second one is less precise - especially in detecting the *high* spending class - introducing a lot of false negative samples (→ low recall).

6. Sequential Pattern Mining

6.1 General Introduction

Dataset used is the cleaned data coming out of the Data Preparation phase, avoiding outliers or transactions with no customers. Said dataset has been further filtered by pruning away all the customers that make a non-significant amount of orders, avoiding the presence of customers with low values, which would just lower the support percentage of interesting patterns. Therefore, after pruning, the dataset was composed of 1101 customers.
The choice of support values was influenced by the computational time of *gsp.py* and therefore, starting from a high minimum support value of 200 (~18%), the other support values thresholds were heuristically found until the computational time became too long (using the intermediate version).
Final minimum support values for the analysis are: 200, 150, 120, 90, 60, ranging in percentage from ~18% to ~5.5%.

6.2 Analyzing interesting patterns

As one could expect, the higher the minimum support, the smaller the set of interesting patterns.
To get a complete analysis it was decided to take a look into interesting patterns with various support percentages:

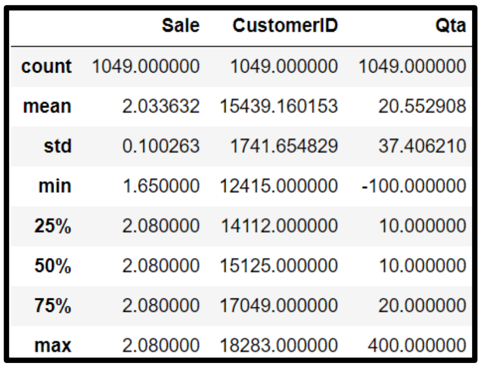
- Medium support patterns (support > 90)
- High support patterns (support > 200)
- Low support patterns (support > 60) adding the complications of having time constraints

6.2.1 Medium support patterns

A fairly interesting pattern mined using the algorithm is the following (with ~8%):

[JUMBO BAG RED RETROSPOT], [JUMBO BAG DOILEY PATTERNS], [JUMBO BAG RED RETROSPOT], [JUMBO BAG DOILEY PATTERNS]

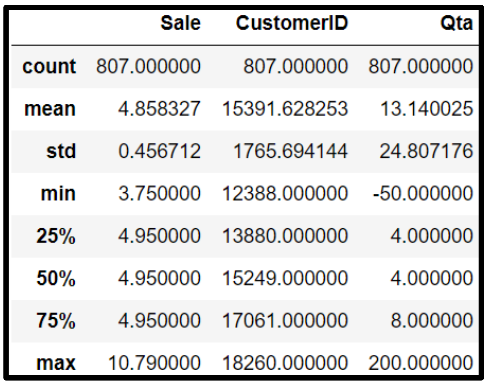
On the right is the description on *Sale* and *Qta* on the dataset composed only of customers having the above pattern and only on the transactions containing the products of the sequence. One can immediately appreciate how the quantity of purchase does not change much up to 75% of the transactions and how the product price stays almost identical in all the dataset (low *std*).
A short research on the nature of *JUMBO BAG* gives a quick explanation to the presence of this pattern, in fact this product is usually a container for other products, useful in many and various circumstances, from shopping to present wrapping. Furthermore, it is surely highly needed by shops, which will need to continuously refill this kind of product.



	Sale	CustomerID	Qta
count	1049.000000	1049.000000	1049.000000
mean	2.033632	15439.160153	20.552908
std	0.100263	1741.654829	37.406210
min	1.650000	12415.000000	-100.000000
25%	2.080000	14112.000000	10.000000
50%	2.080000	15125.000000	10.000000
75%	2.080000	17049.000000	20.000000
max	2.080000	18283.000000	400.000000

6.2.2 High support patterns

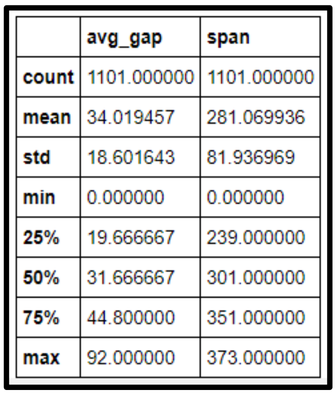
Looking at higher supported patterns means looking at the most present patterns in the whole dataset. The first interesting sequence is (with ~19%): [PARTY BUNTING], [PARTY BUNTING]



	Sale	CustomerID	Qta
count	807.000000	807.000000	807.000000
mean	4.858327	15391.628253	13.140025
std	0.456712	1765.694144	24.807176
min	3.750000	12388.000000	-50.000000
25%	4.950000	13880.000000	4.000000
50%	4.950000	15249.000000	4.000000
75%	4.950000	17061.000000	8.000000
max	10.790000	18260.000000	200.000000

Constructing the description as commented for the previous case, a fact that catches the eye is the quantity of bought items, which for the most part stands between 4 and 8. This quantity of *PARTY BUNTING* can be easily described as bought for the decoration of a single generic party (New Year's Eve, birthday party, etc). Therefore, a plausible and simple explanation for the existence of this pattern can be inferred by assuming that customers which contain this sequence had a couple of parties throughout the period of observation.

6.2.3 Low support patterns including time constraints



	avg_gap	span
count	1101.000000	1101.000000
mean	34.019457	281.069936
std	18.601643	81.936969
min	0.000000	0.000000
25%	19.666667	239.000000
50%	31.666667	301.000000
75%	44.800000	351.000000
max	92.000000	373.000000

After the first part of analysis that focused on the entire history of customers' transactions, it was time to introduce some constraints over the sequences. The next step was to introduce *Time Constraints* over the structure of patterns, more specifically, 3 conditions were introduced:

- *Minimum Gap:* minimum difference between the shopping day of two consecutive orders
- *Maximum Gap:* maximum difference between the shopping day of two consecutive orders
- *Maximum Span:* maximum time-length of the entire sequence

Statistical analysis over the distribution of the data were taken into consideration in order to find the optimal values for the three parameters. On the right are reported the results that guided the decision.

Introducing the time constraints check in *gsp.py* led to different and interesting results, in fact only the lower supported sequences, such as **60-90**, retrieved patterns containing sub-sequences of *2 or more* elements, while the ones with higher support only found patterns of single items.
Given this result, the focus of the analysis shifted to the patterns obtained with support **60**, especially two different patterns:

- ['REGENCY CAKESTAND 3 TIER'] - ['REGENCY CAKESTAND 3 TIER']
- [['JUMBO BAG DOILEY PATTERNS', 'LUNCH BAG DOILEY PATTERN '] - ['JUMBO BAG DOILEY PATTERNS']]

The support of the first one is 111 which corresponds to ~10% of the customers and was taken into consideration as it is the more frequent sequence (→ highest support).
The pattern is composed of a repetition of the same product, which means that the product must appear at least twice throughout the customers' orders. A more accurate study over the behavior of the customers that contain this pattern helps to understand how and to which kind of customer it applies. In fact, the results highlighted 3 distinguishable categories of customers in the dataset: warehouses, resellers and "normal" people. The pattern can be found in all the categories and what differentiates one group from the other is the number of items bought in each order.

The second pattern that was analyzed was chosen because it contains a couple of items in the same sub-sequence so we can try to analyze possible correlations between them. This pattern is less present among the customers, in fact it has a support of 64 (~6%). This can be seen in the dataset because most of the warehouses and resellers still appear but there are way less “normal” people buying these items. As far as concerns possible correlation between *JUMBO BAG* and *LUNCH BAG* there are various aspects that can be analyzed. The item pair is popular, in fact among customers’ order it is present on average 3 times. Another interesting correlation between the two items can be found by noticing that *LUNCH BAG* is present in almost half the orders in which *JUMBO BAG* appears, but, although this is true, it is also biased by the fact that *JUMBO BAG* is frequent by itself.

6.3 Optimization on GSP.py

During the development of this task the most important results were computed using the file *gsp.py*. Unfortunately, the algorithm was not perfectly optimized for our dataset which has many items for each customer and considering small supports implies that way more items are considered. The file had an intermediate version which was obtained by writing an iterative version of *isSubsequence* and an optimized code for *countSupport.* With these changes the time to compute gsp.py with support **60** was *~15530s* (~4.15h). Then a more accurate analysis of the code and implementation of the language itself guided towards the final version that to compute the same results as above “only” took *~2080s* (~30min).

6.4 Conclusion

Wrapping up, a smart pruning on the transaction dataset combined with an increasingly optimized code implementation for *gsp* allowed for an analysis on less supported patterns, bringing up intriguing sequences which were interpretable and explainable.
It is important to highlight the importance of going to less supported patterns since, in the case of this dataset, the vast majority of mined patterns were not of interest since they were composed of only a subsequence of a single element (not going over level 1 of *gsp* once support reached ~25% and of course much less results at even less support percentage once time constraints were introduced).
Furthermore, even the *interesting* patterns (composed of more than one subsequence) contain the same object over and over, and just a handful of these patterns contain a subsequence composed of more than one item. Unfortunately, the items of these *interesting* patterns are the most popular in the whole dataset, giving even less relevance to lots of these sequences.
All this is due to the dataset itself, to the handful of distinguished market fields inside the dataset and to the prevalence of a particular subset of products.
Nevertheless, this report has shown possible explanations for some of these patterns, giving insight into different possible sets of customers.

Full repository with all the code here.