Project Report

Scalable & Cloud Programming

MapReduce Customer Segmentation

Daniele Filippini - 953737
2020/2021
daniele.filippini2@studio.unibo.it

August 5, 2021

Alma Mater Studiorum - University of Bologna
Master's Degree in Computer Science

Introduction

Customer segmentation is the practice of dividing a company's customers into groups that reflect similarity among customers in each group [1]. The goal of segmenting customers is to decide how to relate to customers in each segment in order to maximize the value of each customer to the business.

Customer segmentation has the potential to allow marketers to address each customer in the most effective way.

Since the marketer's goal is usually to maximize the value (revenue and/or profit) from each customer, it is critical to know in advance how any particular marketing action will influence the customer. Ideally, such "action-centric" customer segmentation will not focus on the short-term value of a marketing action, but rather the long-term customer lifetime value (CLV) impact that such a marketing action will have. Thus, it is necessary to group, or segment, customers effectively.

Machine learning customer segmentation allows advanced algorithms to surface insights and groupings that marketers might find difficulty discovering on their own.

However, as the size of the datasets is extremely large in many real cases, parallel processing of complex data analysis becomes indispensable.

MapReduce is a desirable distributed parallel programming paradigm for data intensive applications, based on *shared-nothing* architectures. Due to its simplicity, MapReduce can effectively handle failures and thereby can be scaled to thousands of nodes, making it feasible to compute on large scale datasets.

The input data are partitioned and stored in a distributed file system that is shared across the nodes. The job is divided into small tasks and intermediate results are computed locally by each node. This makes the whole process very efficient.

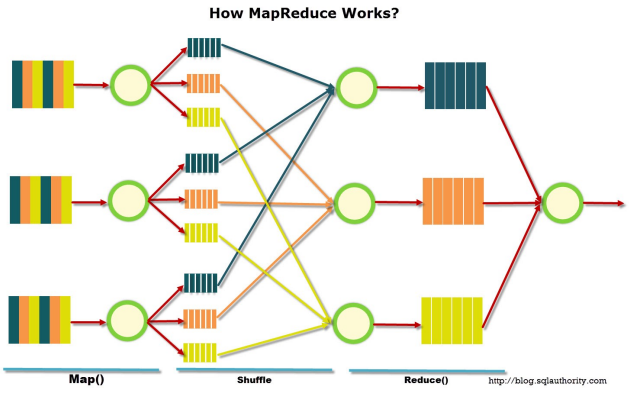


Figure 1: MapReduce framework

The objectives of the project are: build, test and analyse some of the most common machine learning clustering algorithms with their distributed implementation in the field of customer segmentation

and compare their results.

The remainder of the project is organised as follows:

1. Section I Clustering: theoretical explanation of the algorithms used in the experimental studies;
2. Section II RFM: brief review of the RFM approach that is used in the experimental studies to define the features used for clustering;
3. Section III Datasets and Data Exploration: visual exploration of the datasets used in the experiments;
4. Section IV Data Preprocessing: presents and motivates the preprocessing techniques applied to the data;
5. Section V Experimental Settings: summary of the tools and libraries used in the experimental phase;
6. Section VI Experimental Studies and Results: shows and analyses the results obtained in different experiments;
7. Section VII Conclusions: concludes the report.

Contents

1 Clustering

Clustering is an *unsupervised* machine learning task. It involves automatically discovering natural grouping in data. Unlike supervised learning, clustering algorithms only interpret the input data and find natural groups or clusters in feature space. Therefore, clustering techniques apply when there is no class to be predicted, but rather when the instances are to be divided into natural groups [3].

A **cluster** is often an area of density in the feature space where examples from the domain are closer to the cluster than other clusters. The cluster may have a center, called *centroid*, that is a sample or a point and may have a boundary or extent.

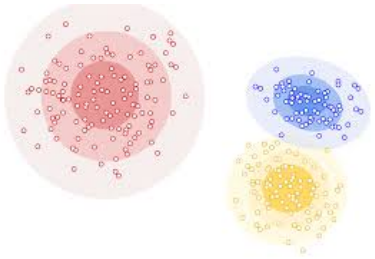


Figure 2: Example of clustering

Clustering can be helpful as a data analysis activity in order to learn more about the problem domain, so-called *pattern discovery*.

Evaluation of identified clusters is subjective and may require a domain expert, although many clustering-specific quantitative measures do exist. Typically, clustering algorithms are compared academically on synthetic datasets with pre-defined clusters, which an algorithm is expected to discover.

Clustering is an unsupervised learning technique, so it is hard to evaluate the quality of the output of any given method.

There are many types of clustering algorithms, but many of them use *similarity measures* between examples in the feature space in an effort to discover dense regions of observations. As such, it is often good practice to **scale data** prior to using clustering algorithms.

Central to all of the goals of cluster analysis is the notion of the **degree of similarity** (or dissimilarity) between the individual objects being clustered. A clustering method attempts to group the objects based on the definition of similarity supplied to it.

There are different clustering algorithms, each of them offers a different approach to the challenge of discovering natural groups in data. There is no best clustering algorithm and no easy way to find the best algorithm for the data without using controlled experiments.

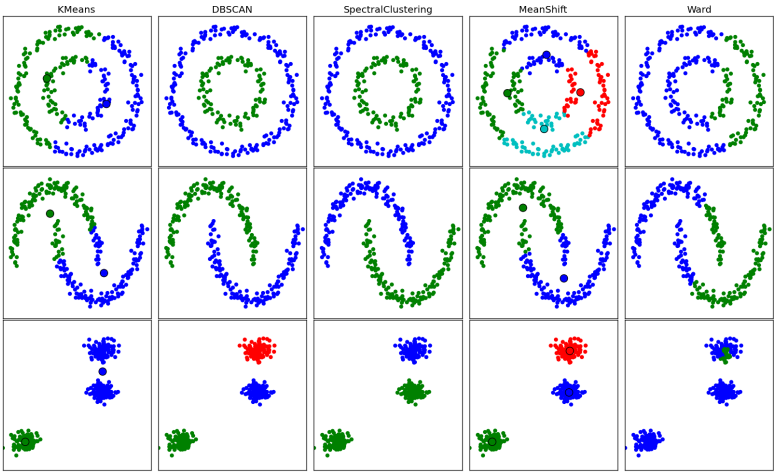


Figure 3: Different clustering algorithms

1.1 K-Means

One of the most well-known and widely used clustering algorithms is Lloyd's algorithm, commonly referred to simply as *K-Means*. The popularity of K-Means is due in part to its speed and in part to its simplicity, the only hyperparameter to choose is K (the number of clusters) [2].

Let $X = \{x_1, ..., x_n\}$ be a set of n data points, each with dimension d, the K-Means problem seek to find a set of k ($\leq n$) means $M = \{\mu_1, ..., \mu n\}$ which minimizes the function

$$f(M) = \sum_{x \in X} min_{\mu \in M} ||x - \mu||_2^2$$

In other words, the aim is to choose k means (cluster centres or cluster centroids) so as to minimize the sum of the squared distances between each point in the dataset and the mean closest to that point.

Find an exact solution to this point is an NP-hard problem, however, there are a number of heuristic algorithms which yield a good approximate solution.

The standard algorithm for solving K-means uses an iterative process which guarantees a decrease in total error (value of the objective function $f(M)$) on each step. The algorithm is as follows:

1. Choose k initial "means" $\mu_1, ..., \mu_k$ uniformly at random from the set X
2. For each point $x \in X$ find the closest mean μ_i and add x to the set S_i
3. For $i = 1..k$ set μ_i to be the centroid of the points in S_i
4. Repeat steps 2 and 3 until the means have converged

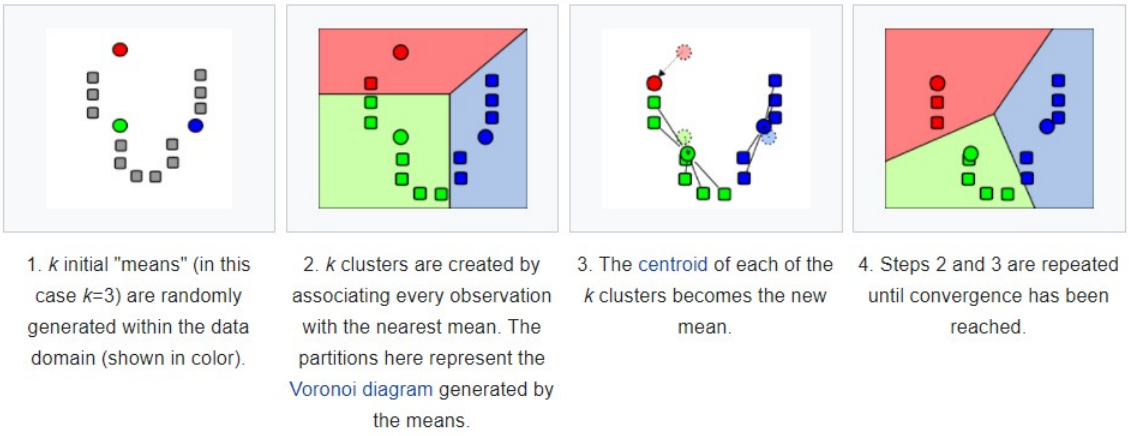


Figure 4: K-Means steps example

The convergence criterion for step 4 is typically when the total error stops changing between steps, in which case a local optimum of the objective function has been reached. However, some implementations terminate the search when the change in error between iterations drops below a certain threshold.

1.1.1 Distributed K-Means

Each iteration of the standard K-Means can be divided into two phases, the first of which computes the sets S_i of points closest to mean μ_i, so we assign each data point to a cluster, and the second of which computes new means as the centroids of these sets, this is the *recenter phase* that takes all the data points assigned to a cluster and use that to update the cluster center [4].

These two phases correspond to the *Map* and *Reduce* phases of the *MapReduce* approach. The Map phase operates on each point x in the dataset. For a given x, the squared distance between x and each mean is computed and the centroid μ_i which minimizes the distance is found.

Then a key-value pair with this mean's index i as key and the value $(x, 1)$ is emitted.

$$map(x) = (argmin_i ||x - \mu_i||_2^2, (x, 1))$$

The Reduce phase is just a straightforward pairwise summation over the values for each key. That is, given two values pairs for a particular key, these are combined by adding each corresponding element in the pairs.

$$reduce((i, [(x, s), (y, t)])) = (i, (x + y, s + t))$$

The MapReduce characterized by these two functions produces a set of k values of the form

$$(i, \sum_{x \in S_i} x, |S_i|)$$

where S_i denotes the set of points closest to mean μ_i. We can then compute the new means (the centroids of the sets S_i) as

$$\mu_i \leftarrow \frac{1}{|S_i|} \sum_{x \in S_i} x$$

Note that in order for the Map function to compute the distance between the point x and each of the centroids, each machine in the distributed cluster must have the current set of means (centroids). Therefore the new means must be broadcasted across the current cluster at the end of each iteration.

1. Choose k initial means $\mu_1, ..., \mu_k$ uniformly at random from the set X.
2. Apply the MapReduce on X.
3. Compute the new means $\mu_1, ..., \mu_k$ from the results of the MapReduce.
4. Broadcast the new means to each machine on the cluster.
5. Repeat steps 2, 3 and 4 until the means have converged.

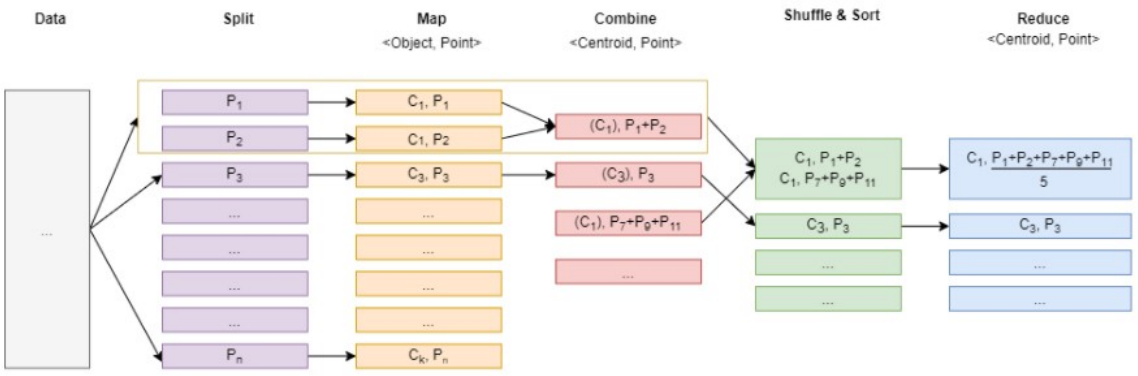


Figure 5: Distributed K-Means steps example

1.1.2 K-Means Analysis

This algorithm has advantages and disadvantages:

Pros:

- It's easy to implement
- It's an efficient algorithm

Cons:

- It requires to specify the initial number of k clusters in advance
- Choosing the initial centroids is important to the result, randomly selected initial centroids might be hard to separate them afterwards and the clustering results will not be ideal
- It is not suitable for all types of data, if the clusters have different sizes or densities or also have irregular shapes, K-Means will not be able to do the clustering very well.

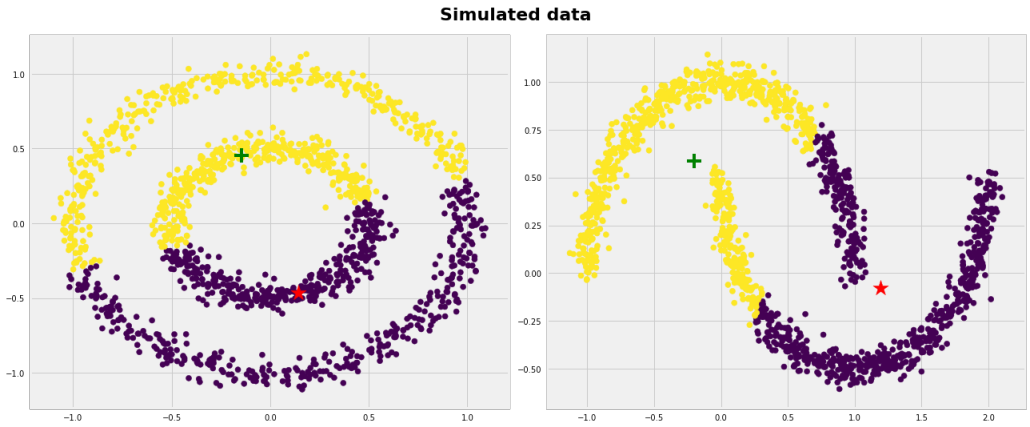


Figure 6: K-Means with clusters of irregular shape

1.1.3 Choosing parameters of K-Means algorithm

Two different methods commonly used to determine the optimal number of clusters for the K-Means algorithm are:

- *Elbow method*: it looks to the total WSS (within-cluster sum of squares) as a function of the number of clusters. One should choose a number of clusters so that adding another cluster doesn't improve much better the total WSS.

  So the idea is to compute the K-Means algorithm for different values of k and for each result to calculate the WSS.

  Then we can plot a curve of these WSS and the location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

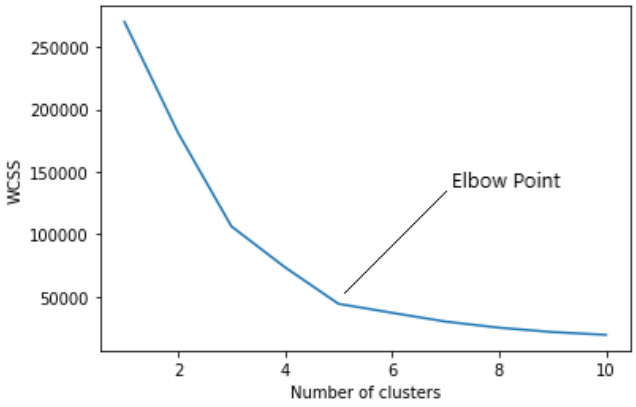


Figure 7: Elbow method

- *Average silhouette method*: it can be used to determine the degree of separation between the clusters.

Average silhouette method computes the average silhouette of observations for different values of k. The optimal number of clusters k is the one that maximizes the average silhouette over a range of possible values for k.

$$silhouettescore = \frac{p - q}{max(p, q)}$$

- p is the mean distance to the points in the nearest cluster that the data point is not a part of
- q is the mean intra-cluster distance to all the points in its own cluster
- The value of the silhouette score range lies between -1 and 1
- A score closer to 1 indicates that the data point is very similar to other data points in the cluster
- A score closer to -1 indicates that the data point is not similar to the data points in the cluster

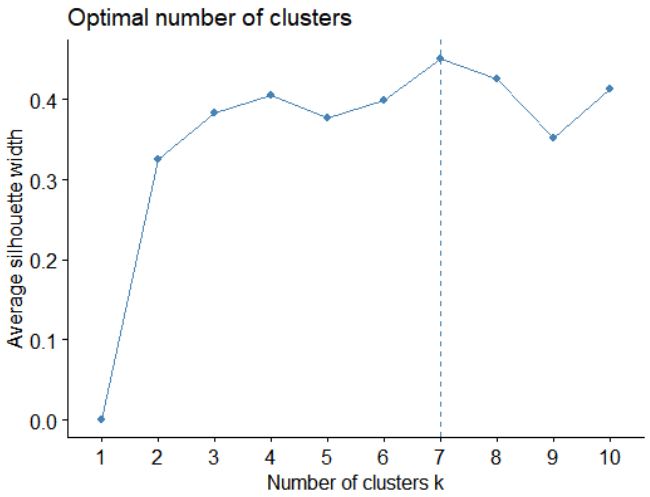


Figure 8: Average silhouette method

1.2 K-Means||

The K-Means algorithm has maintained its popularity even as datasets have grown in size. Scaling K-Means to massive data is relatively easy due to its simple iterative nature. Given a set of cluster centers, each point can independently decide which center is closest to it and, given an assignment of points to clusters, computing the optimum center can be done by simply averaging the points [7].

From a theoretical standpoint, K-Means is not a good clustering algorithm in terms of efficiency or quality: the running time can be exponential in the worst case and even though the final solution is locally optimal, it can be very far away from the global optimum. However, in practice, the speed and simplicity of K-Means cannot be beat.

Therefore, recent work has focused on improving the **initialization procedure**: deciding on a better way to initialize the clustering dramatically changes the performance of the Lloyd's iteration, both in terms of quality and convergence properties.

Ostrovsky et al. propose a new algorithm called **K-Means++**. This algorithm selects only the first center uniformly at random from the data, each subsequent center is selected with a probability proportional to its contribution to the overall error given the previous selections.

Intuitively, the initialization algorithm exploits the fact that a good clustering is relatively spread out, thus when selecting a new cluster center, preference should be given to those further away from the previously selected centers. In practice on a variety of datasets K-Means++ obtained order of magnitude improvements over the random initialization.

Steps of K-Means++ initialization:

1. Pick the first centroid C_1 randomly
2. Compute the distance of all points in the dataset from the selected centroid
3. Make the point x_i that is having maximum probability proportional to the distance d_i as the new centroid
4. Repeat the above two steps until k centroids have been found

The downside of the K-Means++ initialization is its inherently *sequential* nature, it is not apparently parallelizable because the probability which a point is chosen to be the i-th center depends critically on the realization of the previous $i-1$ centers. A naive implementation of K-Means++ initialization will make k passes over the data in order to produce the initial centers.

This slowdown is even more detrimental when the rest of the algorithm (i.e. Lloyd's iterations) can be implemented in a parallel environment like MapReduce.

To address these problems a new initialization method has been proposed in the **K-Means||** algorithm. This is composed of the following steps:

1. Sample a point uniformly at random, that becomes the first centroid in the set C
2. for each point in X find the distance to the closest point in C and compute the sum of all these distances, $\sigma = \sum_{i=1}^{n} d_C^2(x_i)$, that is the initial cost of clustering
3. proceeds in $log(\sigma)$ iterations, where in each iteration, given the current set C of centers, it samples in each x with probability $p_x = l d_C^2/\sigma$

   l is the oversampling factor and is a hyperparameter of the model
4. the sampled point $C^{'}$ are added to C, so $C \leftarrow C \cup C^{'}$ and the iteration continues.

   The expected number of points chosen in each iteration is l and at the end, the expected number of points in C is $l \cdot log(\sigma)$, which is typically more than k.
5. to reduce the number of centers, weights are assigned to the points in C, this is done by computing the number of points for which c_i is the closest centroid
6. Recluster the weighted points in C into k clusters. The size of C is significantly smaller than the input size, so the reclustering can be done quickly. Therefore K-Means++ algorithm is followed to select only k points as starting centroids

This type of initialization technique can be easily implemented in the MapReduce model of computation. To compute σ each mapper works on an input partition $X^{'} \subseteq X$ and computes $\sigma^{'}$, then the reducer can simply add these values from all the mappers to obtain σ. Also the step 3 is very simple, indeed each mapper can sample independently of the others; equally simple is the step 5, each mapper for a subset $X^{'} \subseteq X$ count the values associated to each centroid, by looking to which c_j is the closest for x_i. Then in the reduce phase this values are added finding the final values.

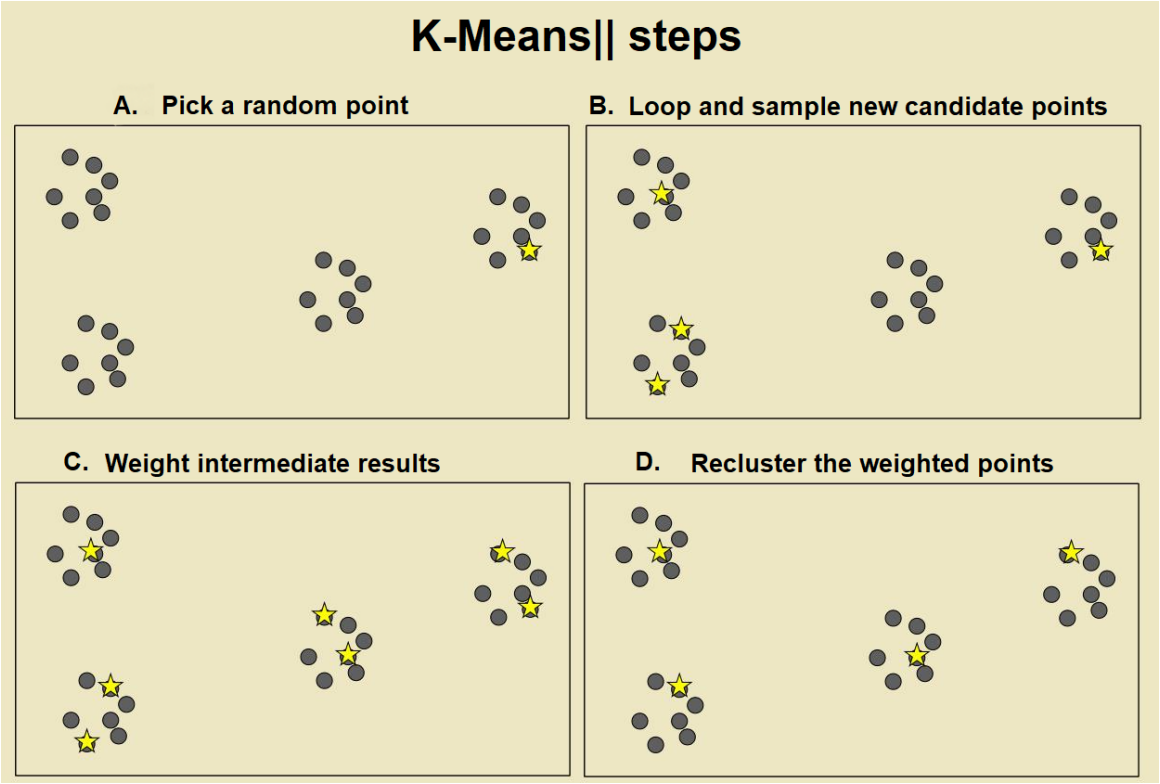


Figure 9: Scalable K-Means++ steps

1.3 DBSCAN: Density-Based Spatial Clustering of Applications with Noise

Density based clustering is a technique that separate regions with high density by regions of low density. DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature [2].

Considering a set of points in some space to be clustered, for the purpose of DBSCAN the points are classified in three categories: core points, border points and noise points.

Given a parameter ϵ, that specifies the radius of neighbourhood with respect to some point, and the parameter *minPts*, which is the minimum number of points for a cluster, we can define:

- a point p as a core point if at least minPts are within distance ϵ of it
- a point q is reachable from p if point q is within distance ϵ from core point p. Point q is directly reachable from point p.
- a point q is reachable from p if there is a path $p_1 ... p_n$ with $p_1 = q$ and $p_n = q$ where each p_{i+1} is directly reachable from p_i. Note that this implies that the initial point and all points on the path must be core points, with the possible exception of q
- all points not reachable from any other point are outliers or noise points

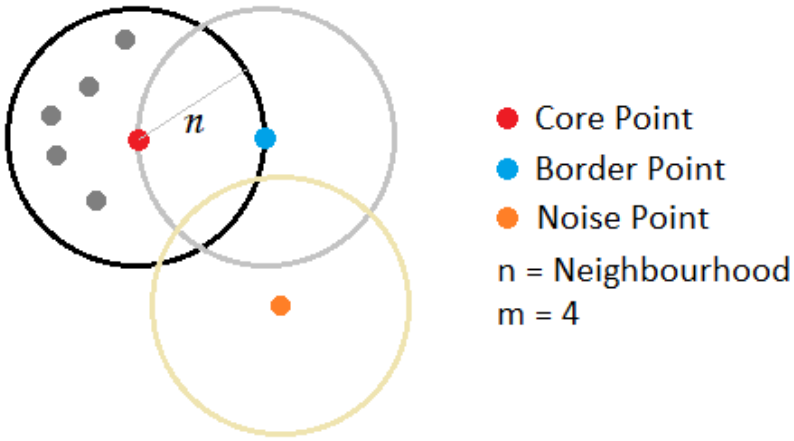


Figure 10: DBSCAN types of points

If p is a core point, then it forms a cluster together with all points (core and non-core) that are reachable from it. Each cluster contains at least one core point; non-core points can be part of a cluster, but they form its "edge", since they cannot be used to reach more points.

The DBSCAN algorithm can be represented with the following steps:

1. The algorithm starts with an arbitrary point which has not been visited and its neighbourhood information is retrieved from the ϵ parameter.
2. If this point contains minPts within ϵ neighbourhood, cluster formation starts. Otherwise the point is labelled as noise. This point can be later found within the ϵ neighbourhood of a different point and, thus can be made a part of the cluster.
3. If a point is found to be a core point then the points within the ϵ neighbourhood is also part of the cluster. So all the points found within ϵ neighbourhood are added, along with their own ϵ neighbourhood, if they are also core points.
4. The above process continues until the density-connected cluster is completely found and then restarts with a new point.

1.3.1 Distributed DBSCAN

The distributed version of DBSCAN is called MR-DBSCAN (MR means MapReduce) and consists of three stages: data partitioning, local clustering and global merging [5] [6].

Figure 11: DBSCAN example

The first step consist into split the data space into non-overlapped boxes that contain roughly the same amount of data points. DBSCAN on Spark uses the number of data points as a cost estimator for the algorithm [5].

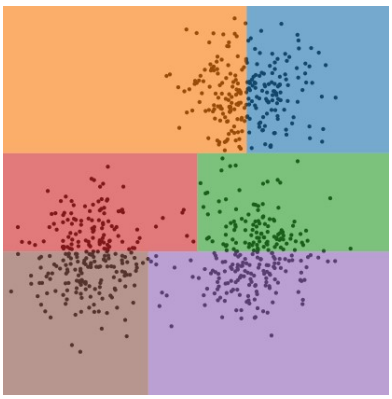


Figure 12: DBSCAN step 1

After all the data has had a box assigned to it, each box is grown to include the points that are within one *Eps* of it. The side effect of including these points is that each box, or partition, will have a copy of them. However, this step is necessary to join the global clusters later on.

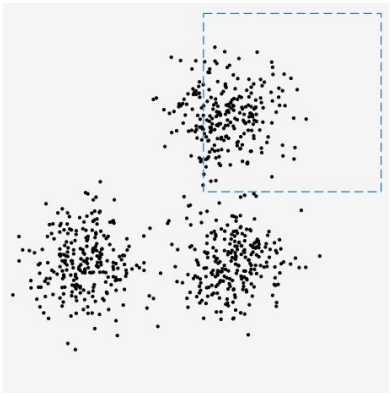


Figure 13: DBSCAN step 2

Therefore the points in the center of the box (also called cell) are processed together on one node and each surrounding cell, with its points, is processed by a different node (see figure 14). To make the point A in cell 1 "see" the point B in cell 2 in a share-nothing platform, we need to make point A part of the cell 2 and point B part of cell 1. So, all points in a cell that are ϵ away from the edge are emitted to the neighbouring cells, in such that now, a per node local DBSCAN can be performed. This double processing of the same edge points by two nodes is actually a blessing, as we can now relate and merge two neighbouring local clusters into one global one.

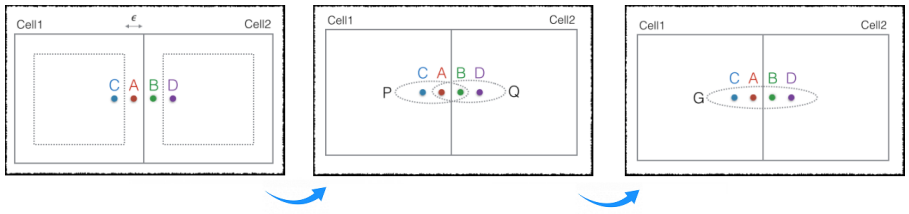


Figure 14: DBSCAN step 2

Then in the local clustering stage the sequential DBSCAN is applied to all the points contained within the boxes' borders. This is done in parallel for each different data partition. Points in different boxes get assigned different clusters (colors) just because they have been assigned to a different box.

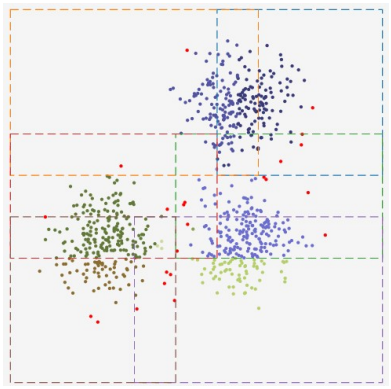


Figure 15: DBSCAN step 3

After local DBSCAN is done, all the points that fall within any two boxes' borders are examined. If the same point has been labelled as part of a cluster in two different boxes, it means that those two clusters are actually one single cluster and they should be merged. All the data points inside the border area get the same color (figure 16): they have been merged.

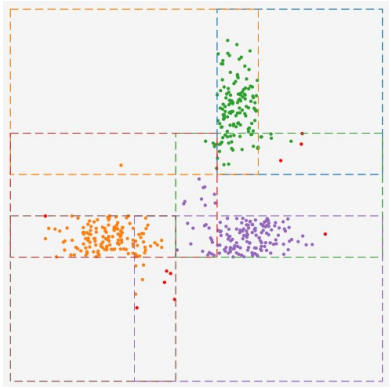


Figure 16: DBSCAN step 4

Finally the remaining points are re-labelled with the globally identified clusters and distributed DBSCAN is done. The red points that are visible in figure 17 are noise points.

Figure 17: DBSCAN step 5

In MR-DBSCAN, the second stage is the dominant part of the whole process in terms of computational time. Since the performance of this stage is decided by the slowest local clustering task, distribute the computation as evenly as possible is a fundamental part. For this reason a cost-based partitioning is performed, i.e. partitions are created based on the estimated computational cost.

1.3.2 DBSCAN Analysis

This algorithm has advantages and disadvantages:

Pros:

- It can handle clusters with different sizes and irregular shapes
- It does not requires to specify the number of clusters in the data a priori
- It is robust to outliers

Cons:

- DBSCAN cannot cluster well data sets with large differences in densities, since the MinPts-ϵ combination cannot then be chosen appropriately for all clusters
- If the data and scale are not well understood, choosing a meaningful distance threshold ϵ can be difficult
- Higher computational cost if compared with other clustering algorithms, e.g. K-Means

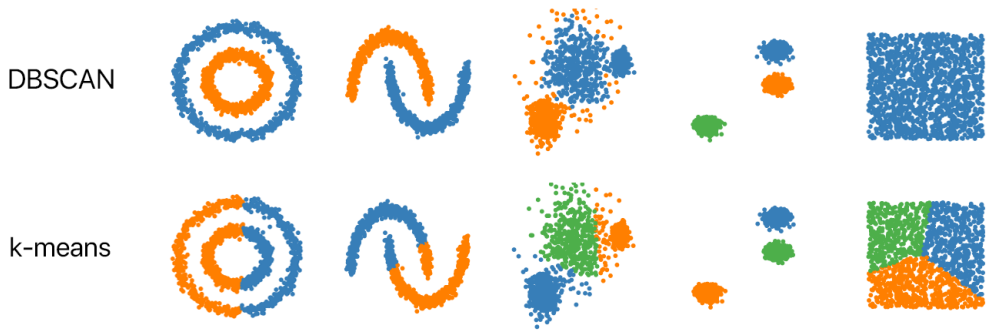


Figure 18: DBSCAN vs K-Means

1.3.3 Choosing parameters of DBSCAN algorithm

DBSCAN algorithm requires 2 parameters - *epsilon*, which specifies how close points should be to each other to be considered a part of a cluster; and *minPts*, which specifies how many neighbours a point should have to be included into a cluster. However, you may not know these values in advance.

There is not a standard method to define these two parameters and typically a domain knowledge is needed to choose them appropriately. However, some method can be used to try to understand which value is good to define clusters in the data.

To define a value for ϵ we can calculate the distance from each point to its nearest neighbour within the same partition, so, for a small fraction of points this distance will not be accurate. However, this inaccuracy doesn't affect the overall distribution of distances too much. Then we can plot these values and choose the most frequent one as ϵ.

After we have chosen the value for epsilon, we can counts each point's neighbours and build a histogram to have some insight on useful values. After that we can try different ranges of values and analyse the results by looking for example to the silhouette score.

Another common rule of thumb is to have $MinPts = NumFeatures \times 2$.

2 RFM

A typical way to approach customer segmentation is to conduct **RFM analysis** [11] [12].

RFM stands for Recency, Frequency, and Monetary value, each corresponding to some key customer trait. These RFM metrics are important indicators of the customer's behaviour because frequency and monetary value affects the customer's lifetime value, and recency affects retention, a measure of engagement.

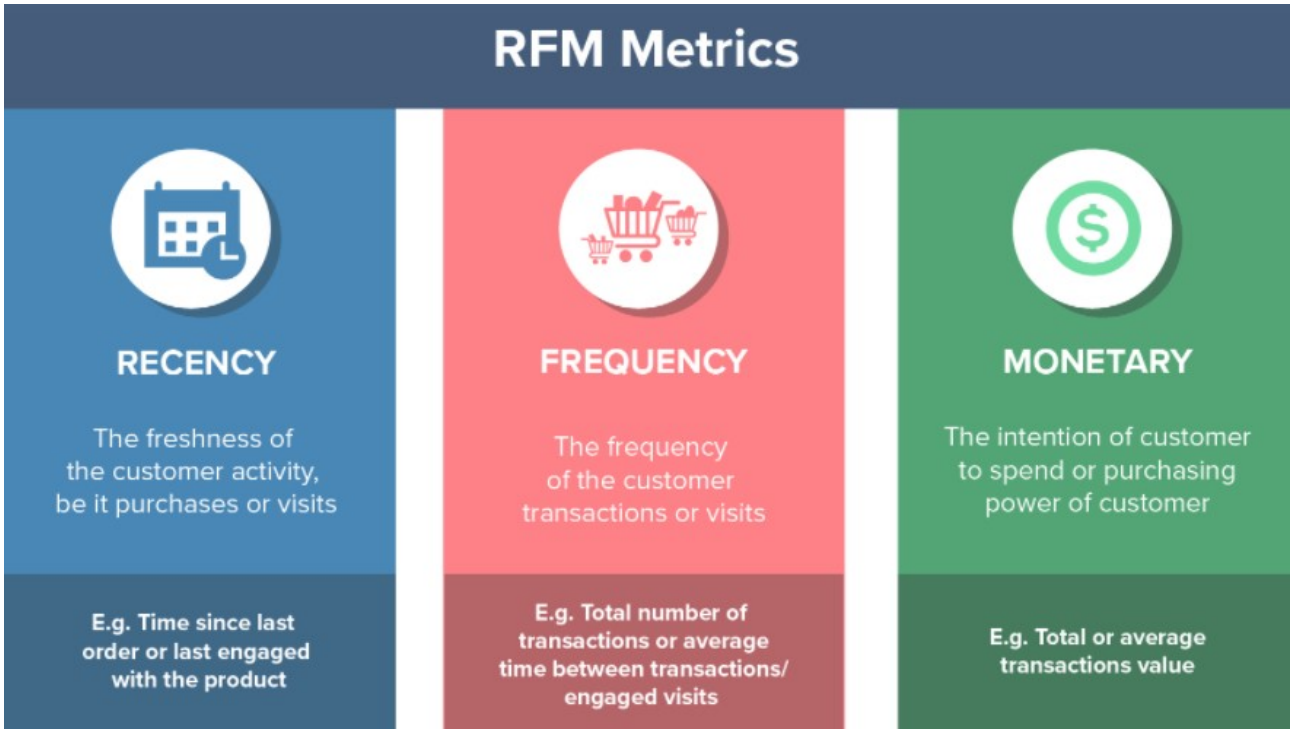


Figure 19: RFM Analysis metrics

Instead of simply using an overall RFM average value to identify the best customers, business analysts can use RFM method to identify clusters of customers with similar values, called customer segmentation. At this point machine learning techniques can be used to figure out the best number of clusters to use, taking a lot of the guesswork out of the clustering process.

3 Datasets and Data Exploration

Unfortunately there is no a benchmark dataset for customer segmentation available online, maybe due to the valuable value that such data have for companies and to privacy policy restrictions.

For the experimental studies three different dataset have been used, these datasets contains orders from customers to e-commerce sites, without any personal information about the users.

3.1 Online Retail dataset

This Online Retail II dataset [8] contains all the transactions occurring for a UK-based and registered, non-store online retail between 01/12/2009 and 09/12/2011. The company mainly sells unique all-occasion gift-ware. Many customers of the company are wholesalers.

The dataset contains the following information:

- a record for every transaction, including the day and time when each transaction was generated;
- a product unique code for every different item with a product description and also the quantity ordered with the unit price;
- a customerId and the country where he resides;

3.1.1 Explanatory Data Analysis

The dataset contains 1.067.371 of samples, that represents different product purchases. However, in 243.007 cases the CustomerID field contains Null values, therefore we cannot use those entries in our experiments and we have eliminated them.

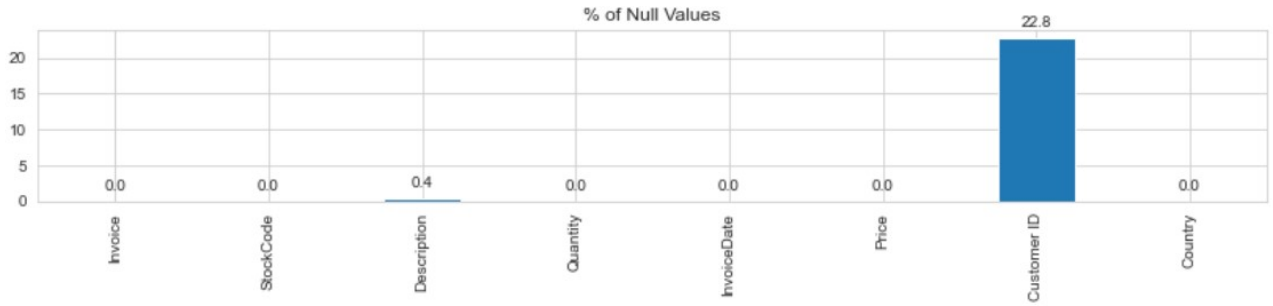


Figure 20: Online Retail dataset percentage of null values

The dataset contains also rows with erroneous values and duplicate entries, therefore after the cleaning process remains only 779.495 samples in the dataset.

However, the number of different customers is much much lower and is equal to 5.885, this is due to the fact that many made a great number of purchases, the maximum for a single customer is greater than 12.000 transactions.

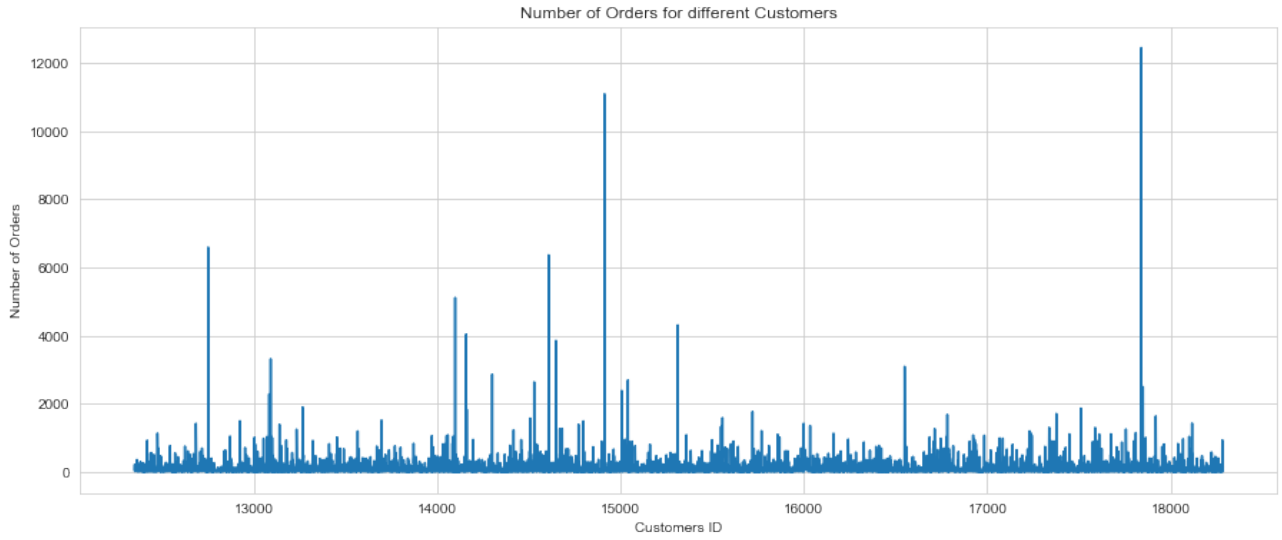


Figure 21: Online Retail dataset number of orders per customer

The distribution of the number of transactions made by the different customers shows that only few of them bought frequently from the shop, the great majority made a single purchase. A single customer made 395 transaction, but the 75% percent has a number of transaction less than 7.

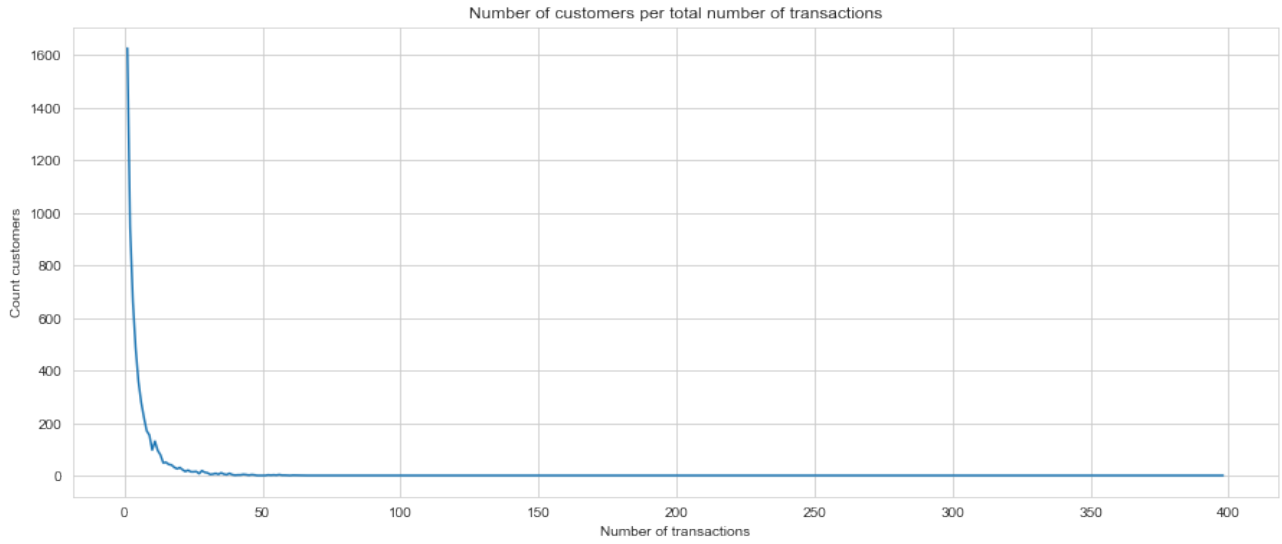


Figure 22: Online Retail dataset number of customer per number of transactions

Also the amount spent is very different among the customers, with a maximum of 580.987,00 and a standard deviation of 14.421,50.

Already these initial data suggest that there are customers with very different behaviours who buy from the site.

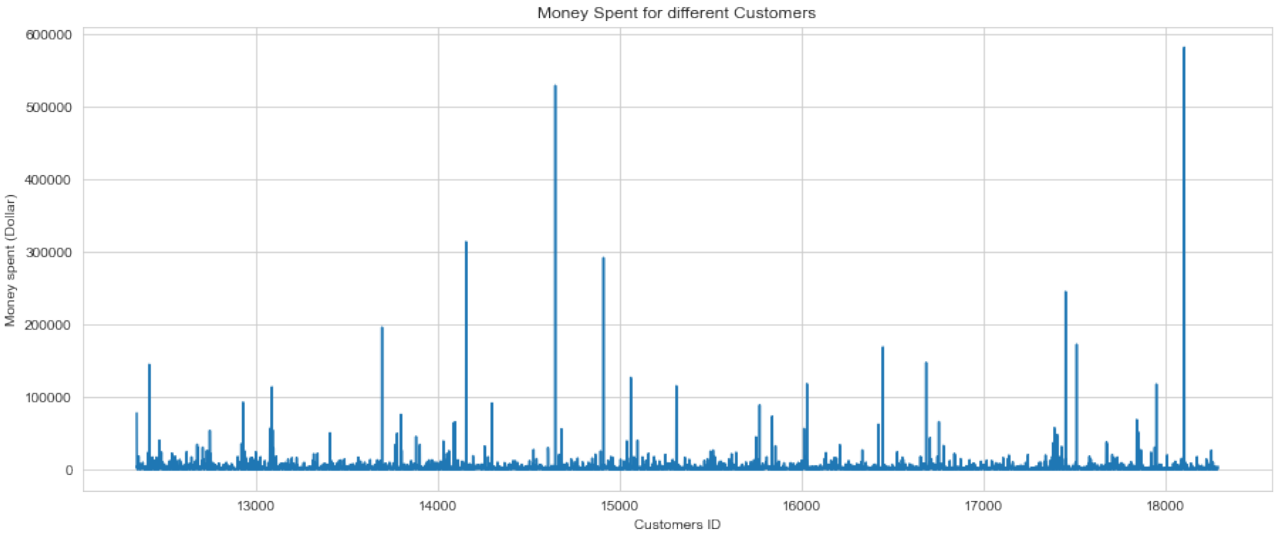


Figure 23: Online Retail dataset total amount spent per customer

The products have different unit price, the maximum is 10.953,50 and the minimum is 0,00 because there are free items given to customers from time to time. The average unit price is small and is about 3,00.

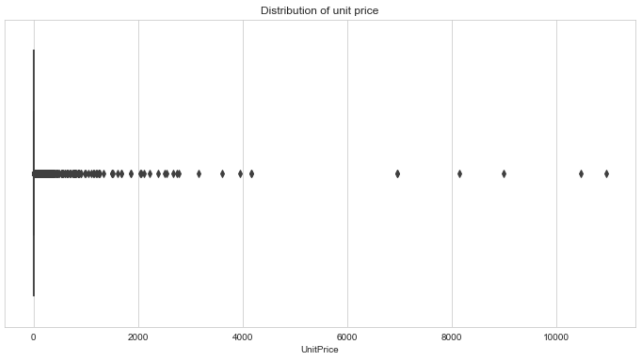


Figure 24: Online Retail dataset unit price distribution of different items

The only information that we have about the customer is their country. By analysing it we know that customers come from 41 different countries, the majority of them are from UK (about 85%), since the site is an English site, and almost all the countries served are Western countries, ∼95% of all the transactions occurring are from Europe.

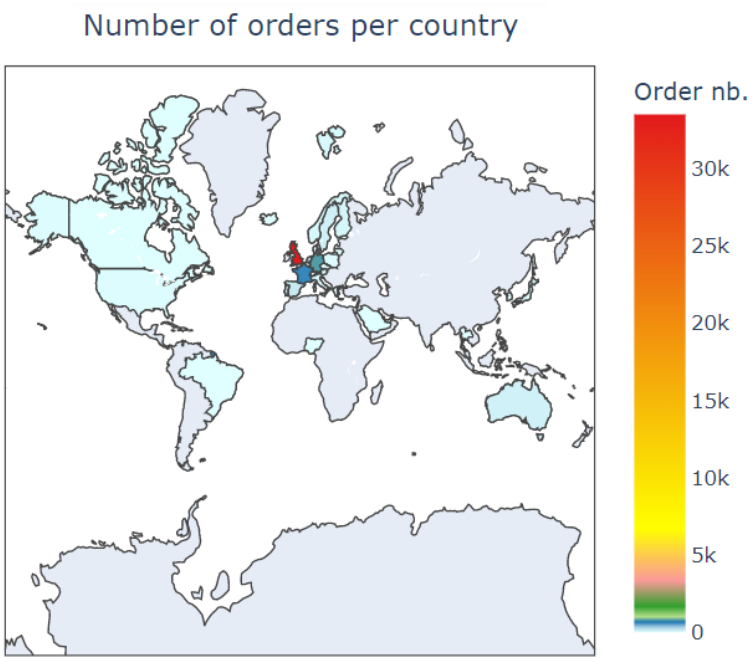


Figure 25: Online Retail dataset orders map

Also the number of items per order varies a lot, the great majority has a single item, but there are also orders with more than 500 objects.

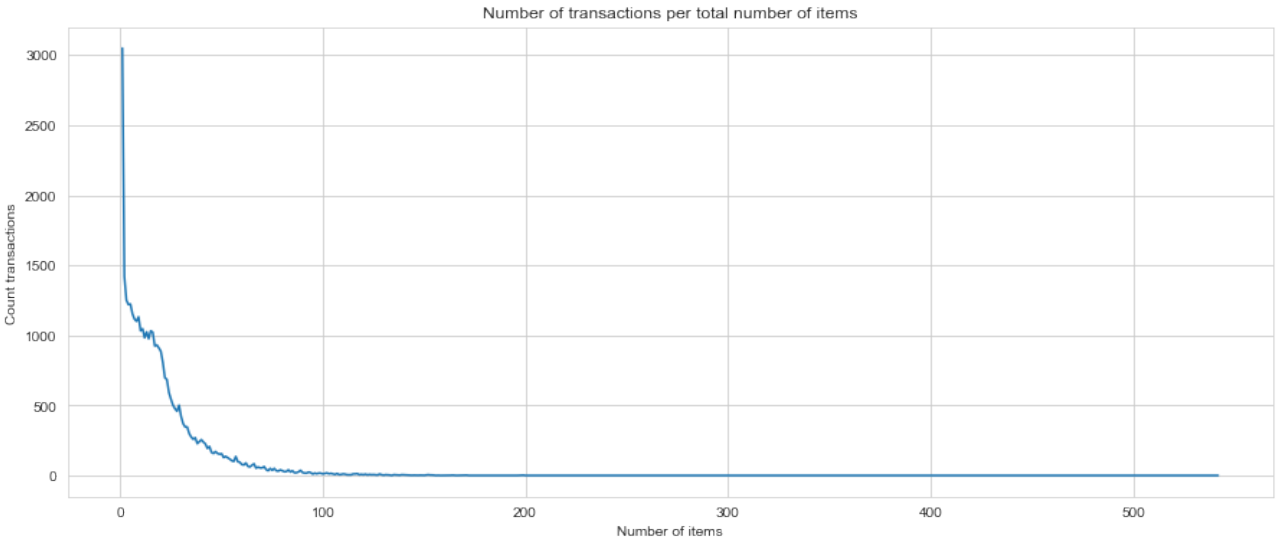


Figure 26: Online Retail dataset number of items per transaction

Finally by looking to the frequency of the transaction, by computing the number of days since the last transaction (considering as benchmark the last day in the dataset), we can see that many customers have recently made a purchase, more than 50% of them made a purchase in the last 100 days. There are also customers who made a purchase only once, a long time ago, and never came back.

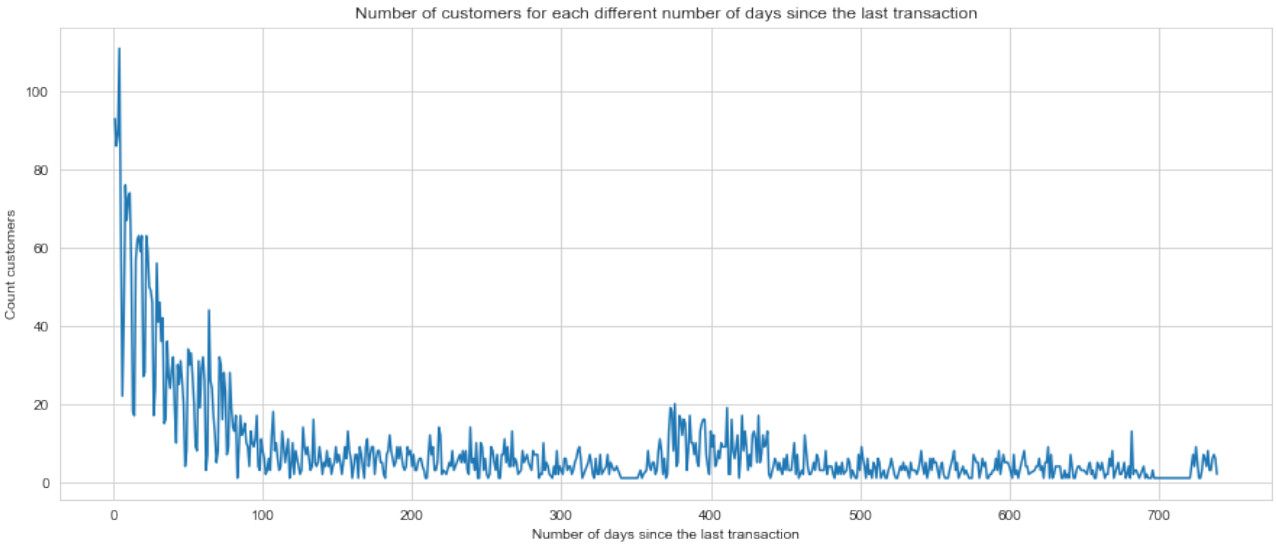


Figure 27: Online Retail dataset frequency of customer purchases

3.2 Instacart dataset

Instacart is a grocery shopping service. Users order their groceries through an app, and just as with other gig-economy companies, a freelance "shopper" takes responsibility for fulfilling user orders.

The Instacart dataset [9] contains the following informations:

- a record for every order placed, including the day of week and hour of day (but no actual timestamp);
- a record of every product in every order, along with the sequence in which each item was added to a given order, and an indication of whether the item had been ordered previously by the same customer;
- the name, aisle and department of every product.

The data has been thoroughly anonymized, so there is no information about users other than user ID and order history — no location data, actual order dates, or monetary values of orders.

This is important to note because those missing types of information are some of the most important for business analytics. Companies very much want to know whether a user has been active recently, how active they have been over the past day/week/month/quarter, and what their monetary value is to the company.

For this reason we will need to base our analysis on other parameters and synthesize new features starting from the available one in the dataset.

3.2.1 Explanatory Data Analysis

The dataset contains a sample of over 3 million grocery orders from more than 200.000 Instacart users. For each user, from 4 to 100 of their orders are given, with the sequence of products purchased in each order, there are the data of the previous purchases of each user.

In this second dataset there are no missing data for the user ID, but only for the field "days since prior order".

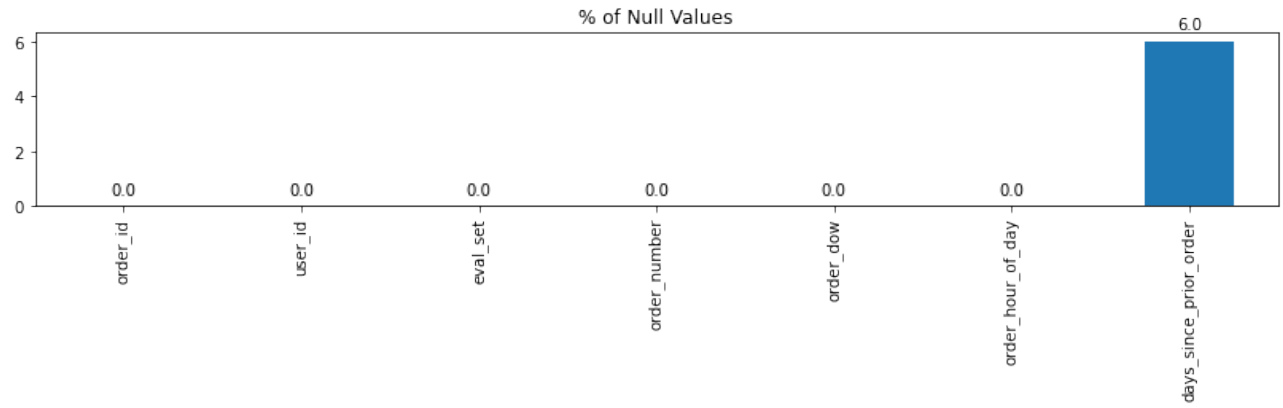


Figure 28: Instacart dataset percentage of null values

By analysing the data we can see how the number of orders per customer actually goes from 4 to 100, however over than 75% of customers made less than 20 orders.

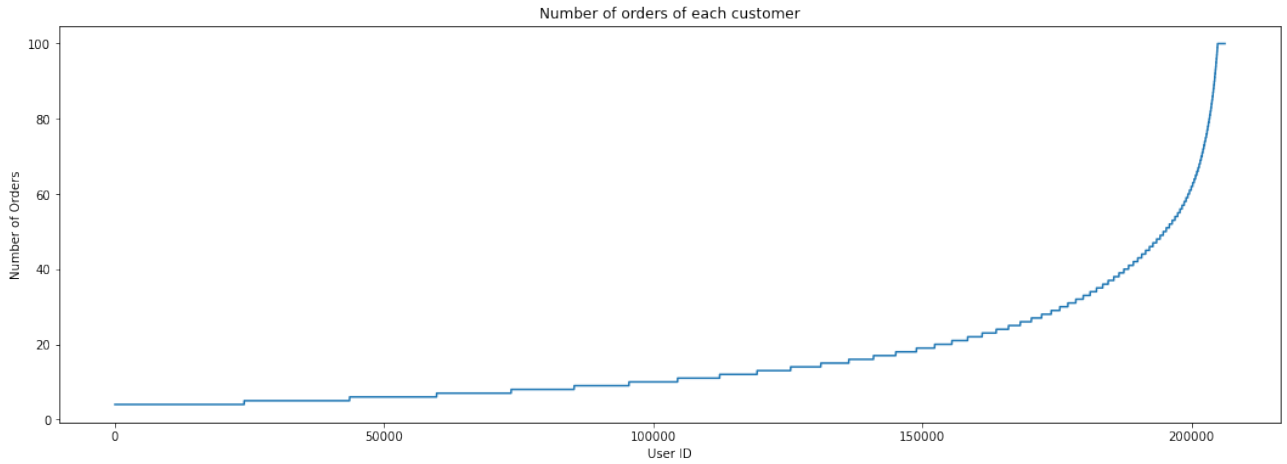


Figure 29: Instacart dataset number of orders per customer

The distribution is heavily skewed, the majority of the customers made only 4 orders.

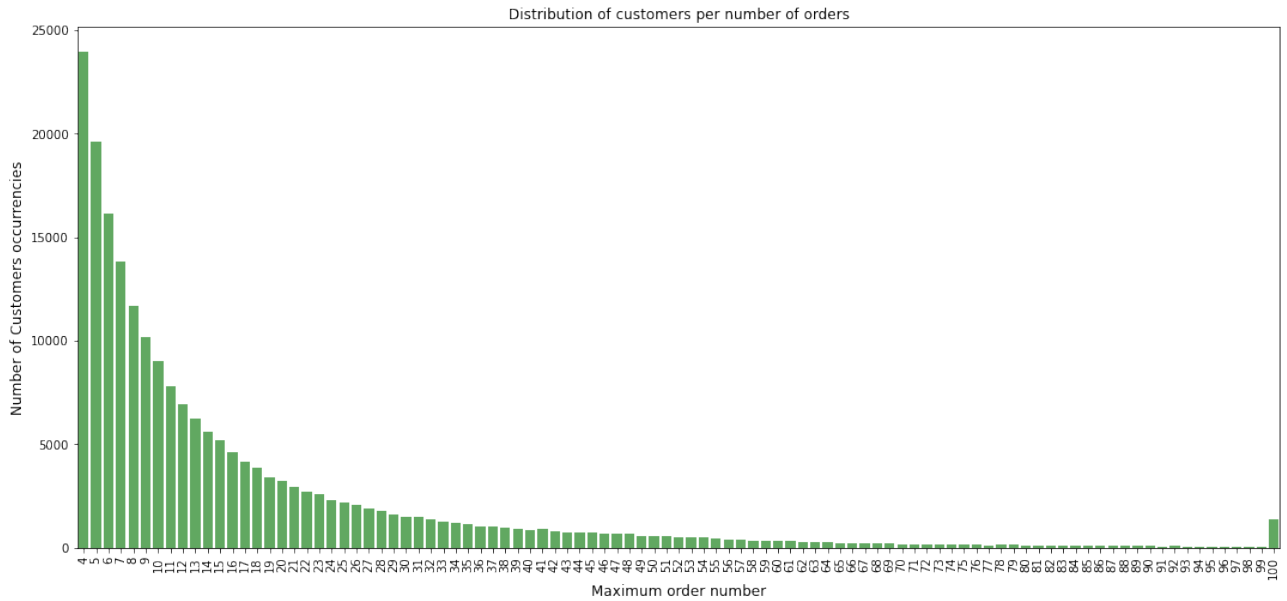


Figure 30: Instacart dataset distribution of customers per number of orders

By analysing the days since the prior order of the customers, it seems that customers order once in every week (check the peak at 7 days) or once in a month (peak at 30 days). We could also see smaller peaks at 14, 21 and 28 days (weekly intervals).

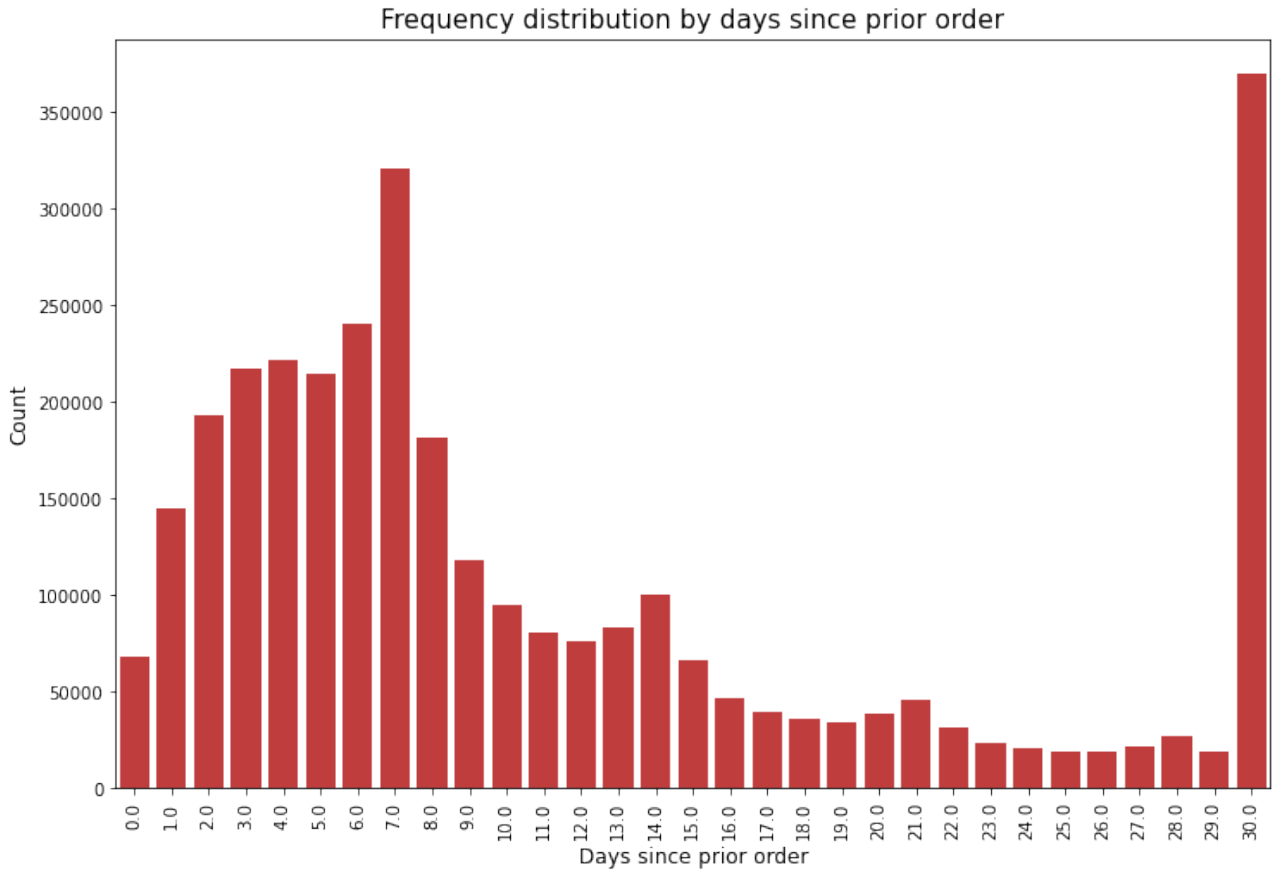


Figure 31: Instacart dataset orders frequency distribution

The majority of the orders contains few items, the higher frequency is 5, more than 75% of the orders contain less than 15 elements. The higher number is 80 in a single case.

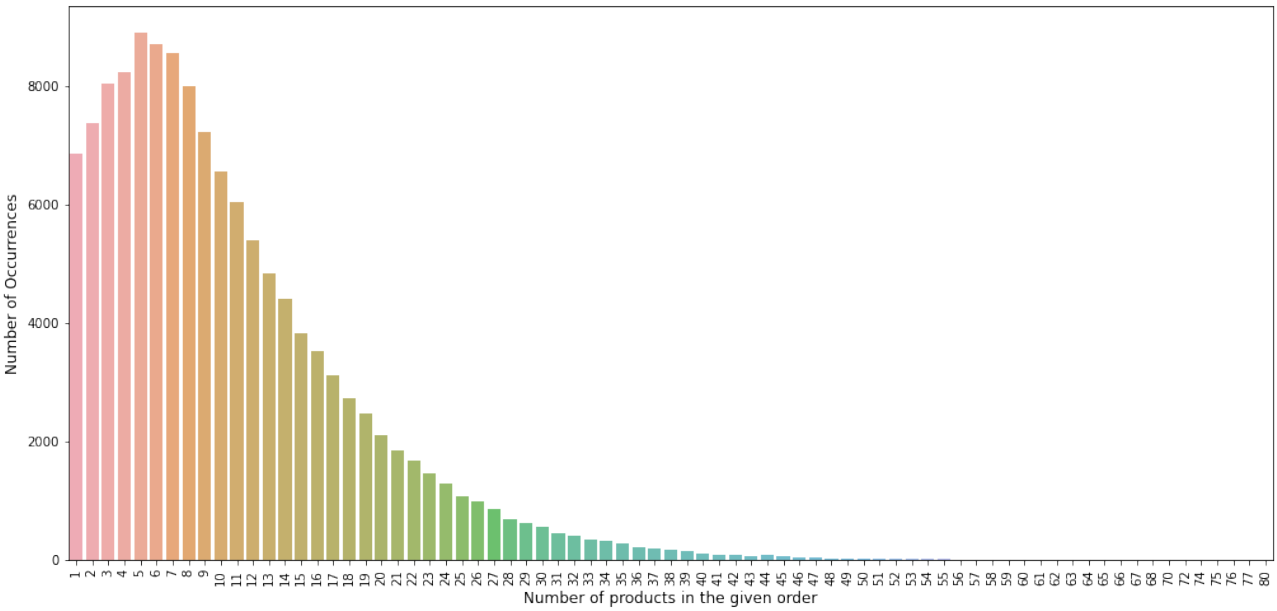


Figure 32: Instacart dataset count orders for different number of items

3.3 Multi Category Store dataset

This dataset [10] contains behaviour data from 7 months (from October 2019 to April 2020) from a large multi-category online store.

Each sample of the dataset represents an event that is related to a product and a user.

There are different types of events: view, cart, remove_from_cart and purchase, we are interested only in the last of them.

The dataset contains the following information:

- event type and time;
- a product unique code for every different item with a product description and the price;
- a user ID to identify the different customers;

3.3.1 Explanatory Data Analysis

The dataset contains a sample of over 6 million orders from more than 2 million of customers. In this dataset there aren't missing data, so we can leverage all off them.

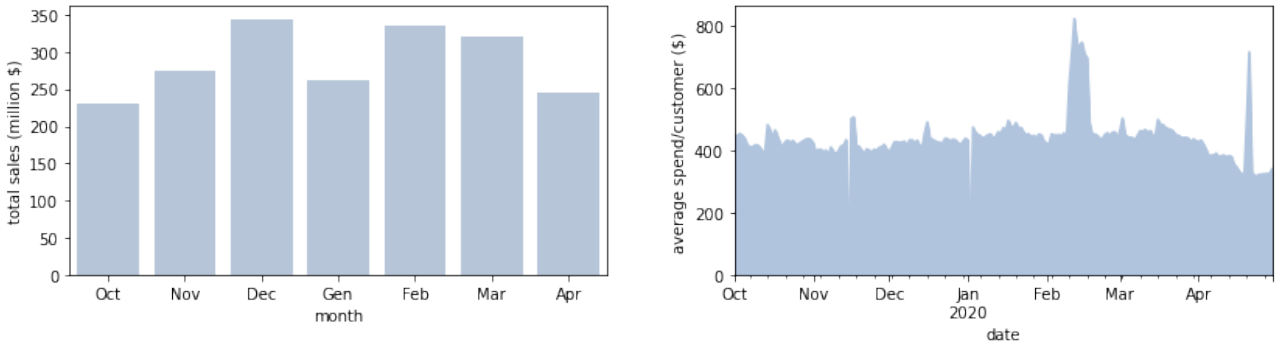


Figure 33: Multi Category Shop dataset sales per month

December is the month with the highest amount of sales, but we can also see that there is a peak between February and March where the average amount spent by the customer is very higher.

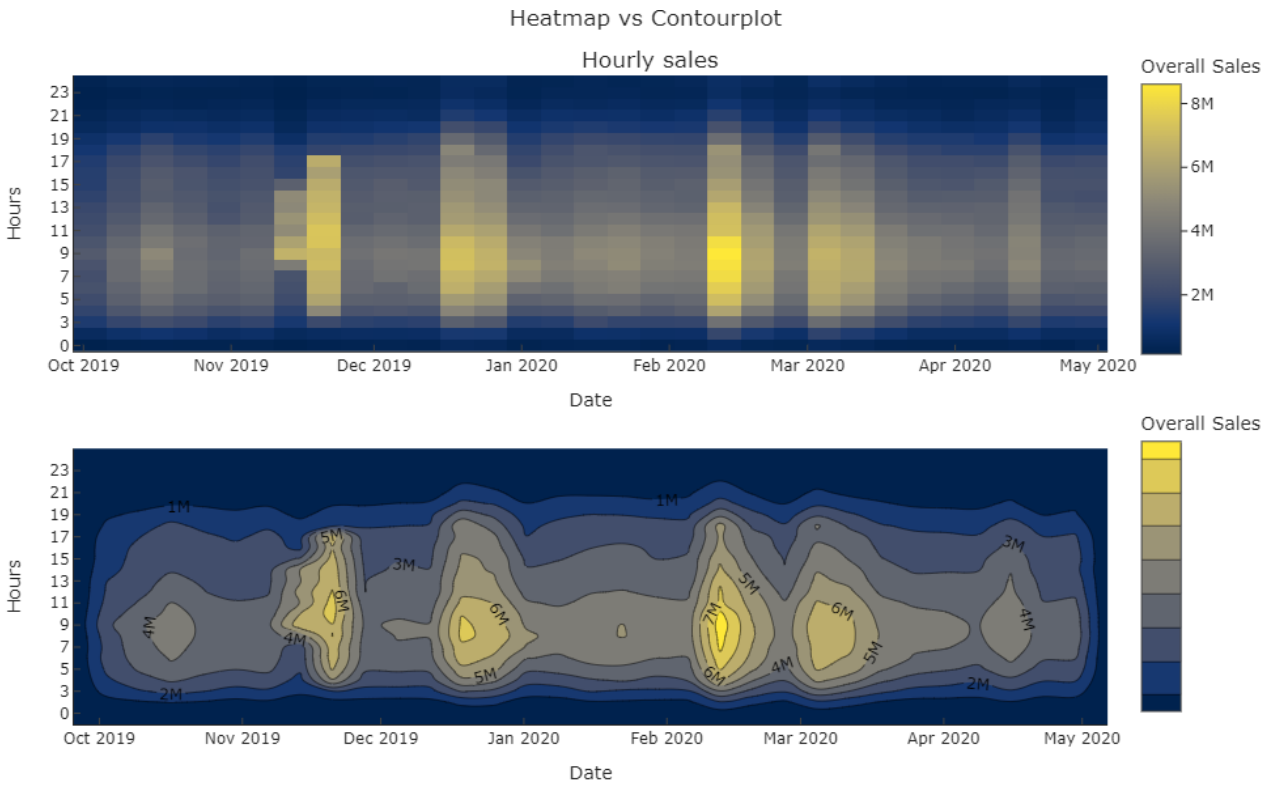


Figure 34: Multi Category Shop dataset order rate per hour and month

We can also see that the customers tend to make their orders between 5 a.m. and 7 p.m., with a higher rate around 9 a.m.

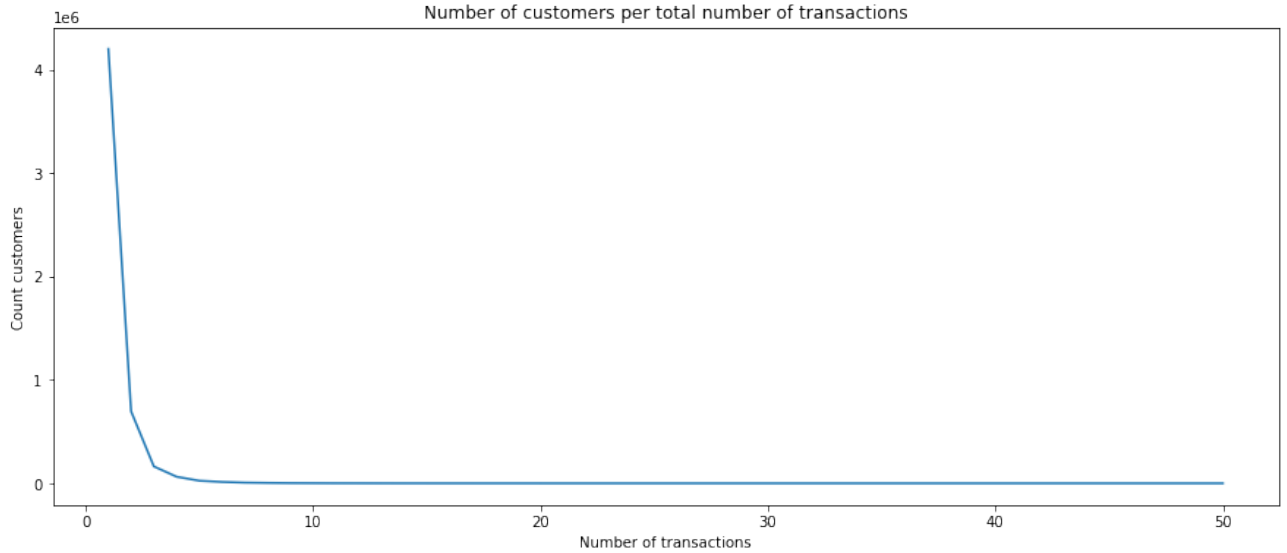


Figure 35: Multi Category Shop dataset customer x number of orders

Many users made a small number of purchases, with the minimum equal to 1, only a small fraction made more than 10 transactions, where the maximum is 50. The mean number of orders is equal to 1.

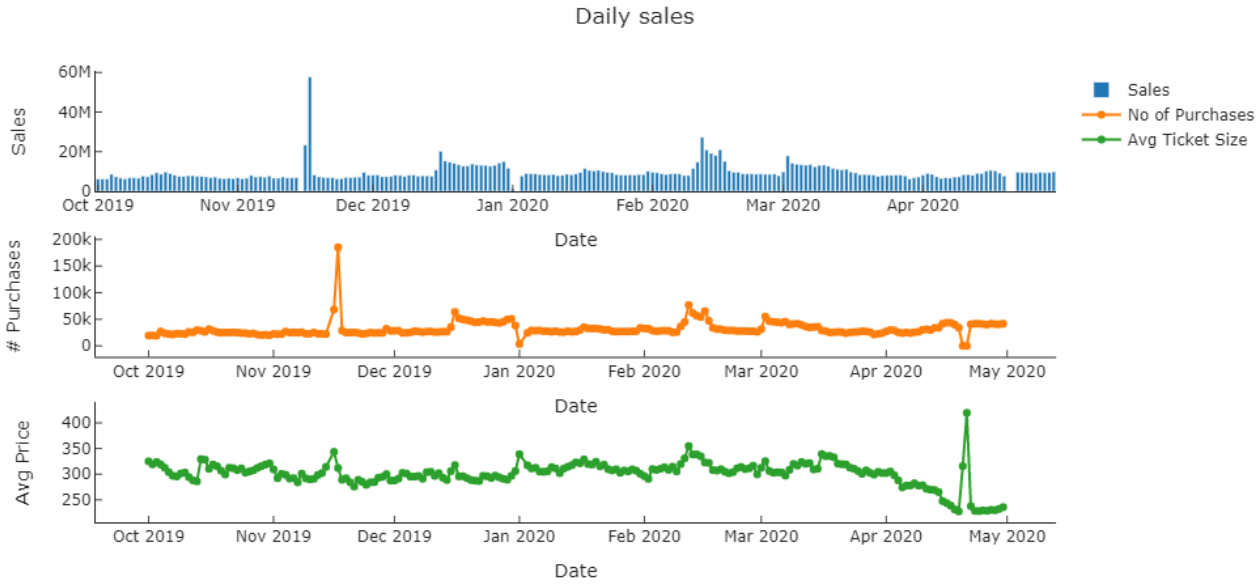


Figure 36: Multi Category Shop dataset daily sales

As we can see from above graph, there is a peak where sales were high. This happened on Nov 16th and Nov 17th, 2019. This might be due to some specific promotional campaigns. The number of sales is also high on these days but the average price is not that low by which we can infer that, even if there is any promotional campaigns running on those days, the discount is not that low affecting the average price.

Or also this could be due to some error in registration of data, as we can see for the Nov 15th we do not have any sale, maybe due to some error during data registering.

The sales are constant over the these months, the average price has more oscillations, we can see a slow descent between March and April, but with a peak on 20th April.

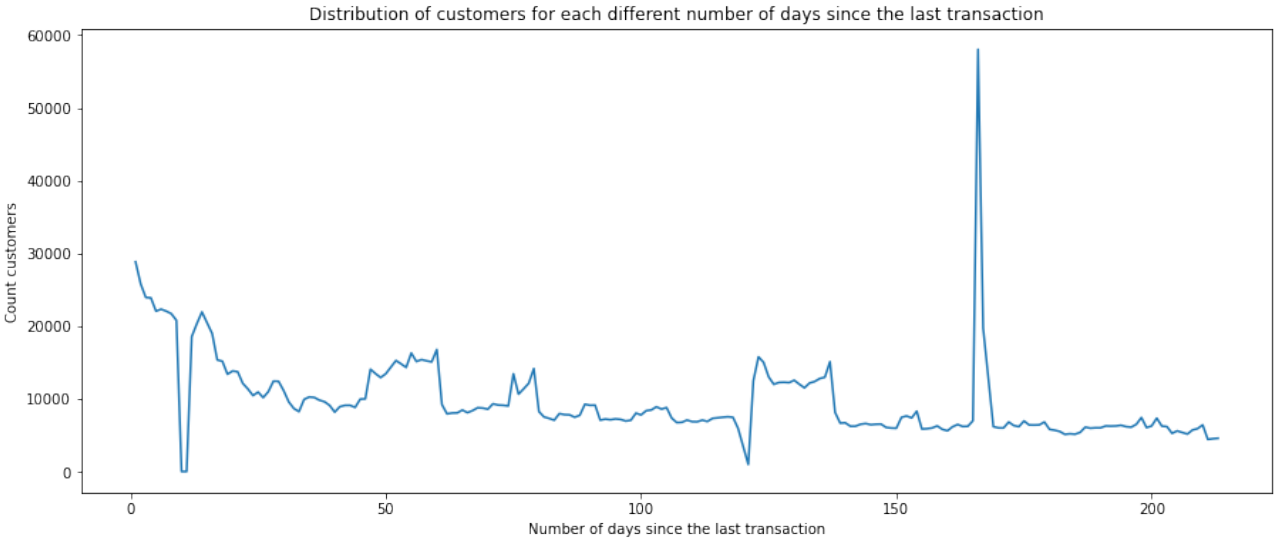


Figure 37: Multi Category Shop days since last order

This last plot shows the days since the last order, the minimum is 1 and the maximum is 213, more than 20% of users made an order in the last 30 days, but also more than 25% made the last order more than 120 days ago.

4 Data Preprocessing

The datasets are not directly built to do customer segmentation, but are available to deal with different tasks, such as multivariate analysis, time-series analysis to predict the next purchase, etc.

To obtain data that can be used to accomplish the customer segmentation task, some preprocessing steps are needed to synthesize a useful dataset.

Since all the presented dataset are limited to sales records and did not include other information about the customers, we will use RFM based model (for details see section 2) to find the customer segments.

4.1 Online Retail dataset preprocessing

The preprocessing steps applied in this case are those needed to obtain the three important information of the RFM model:

- **Recency**: it's the value of how recently a customer purchased on the site. To create the recency feature variable we take the last transaction date plus one day (i.e. 10th Dec 2011) as reference data and we computed the number of days before that reference date when a customer made its last purchase.

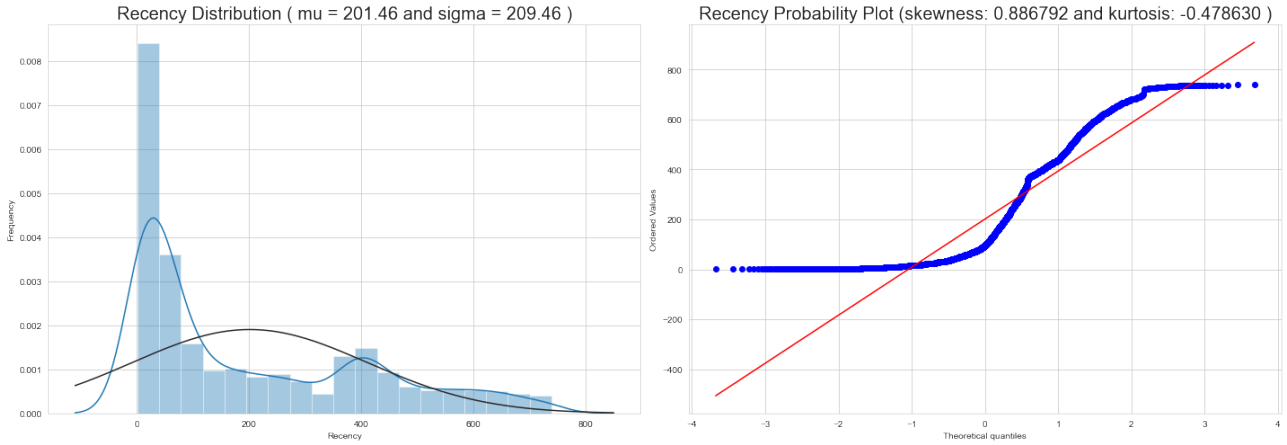


Figure 38: Online Retail dataset recency distribution

From the first graph above we can see that sales recency distribution is skewed, has a peak on the left and a long tail to the right. It deviates from normal distribution and is positively biased.

From the Probability Plot, we could see that sales recency also does not align with the diagonal red line which represent normal distribution. The form of its distribution confirm that is a skewed right.

With skewness positive of 1.25, we confirm the lack of symmetry and indicate that sales recency are skewed right, as we can see too at the Sales Distribution plot, skewed right means that the right tail is long relative to the left tail.

- **Frequency**: represents how frequently a customer make an order. To create the frequency feature we counted the number of different transactions per customer.

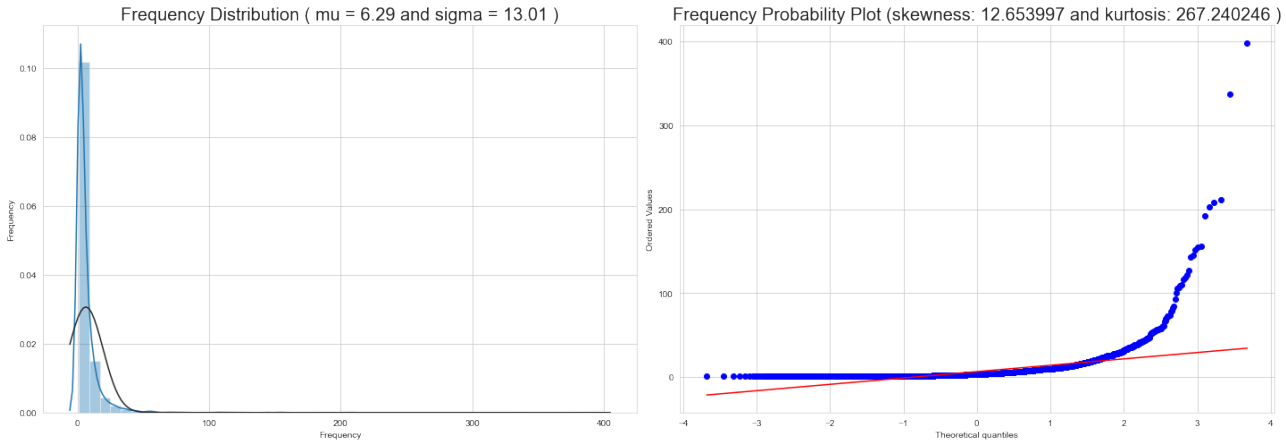


Figure 39: Online Retail dataset frequency distribution

From the first graph above we can see that sales frequency distribution is skewed, has a peak on the left and a long tail to the right. It deviates from normal distribution and is positively biased.

From the Probability Plot, we could see that sales frequency also does not align with the diagonal and confirm that is a skewed right.

- **Monetary Value**: it represents the monetary value of all the transactions that the customer made. This amount is simply given by the sum of all the pounds spent in each transaction made by an user.

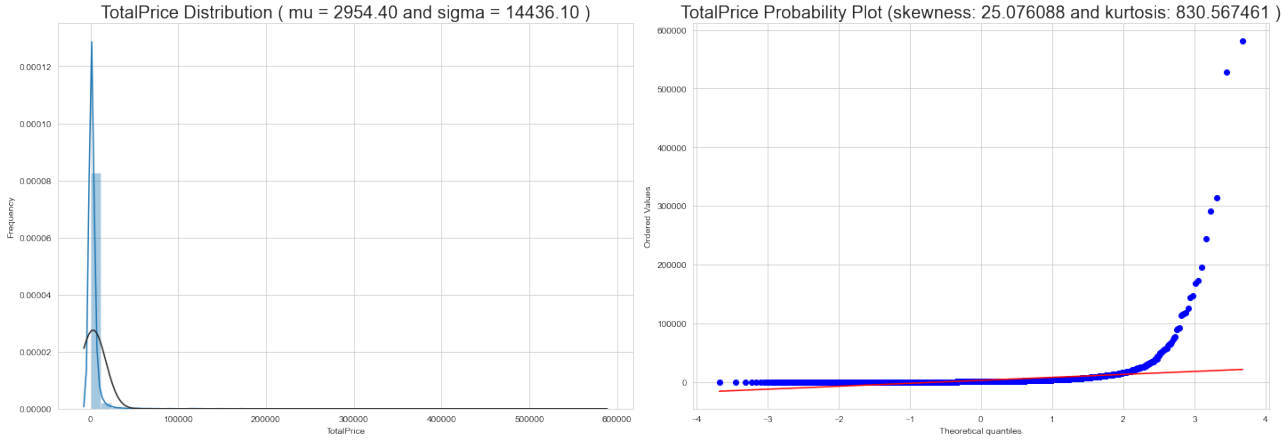


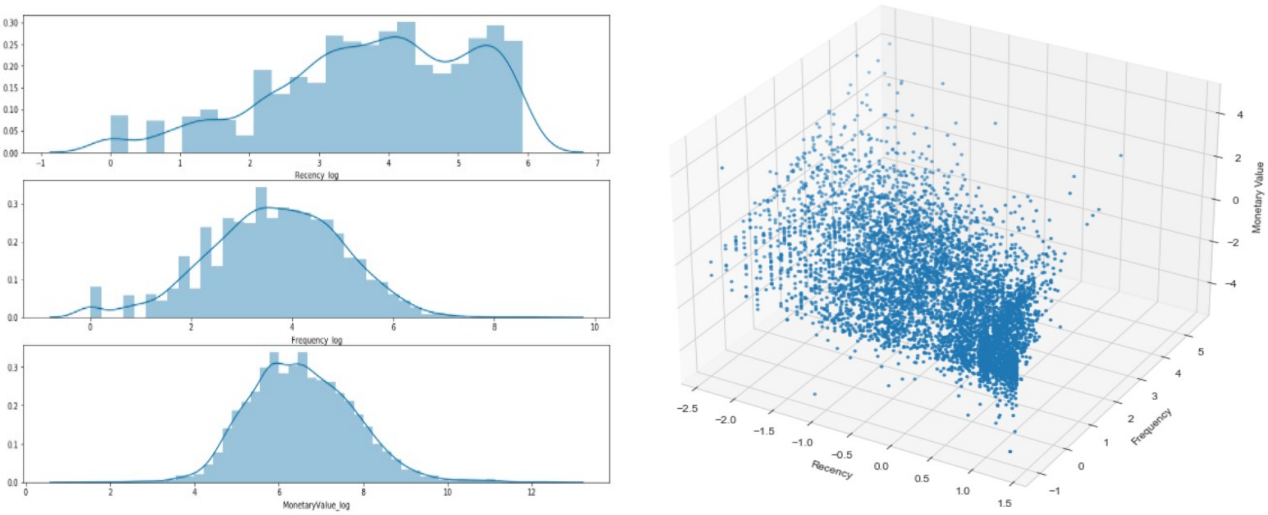
Figure 40: Online Retail dataset monetary value distribution

Also the sales amount has a skewed distribution with a peak on the left and a long tail to the right. It deviates from normal distribution and is positively biased.

Once we have created our **customer value dataset**, with the three features required in the RFM analysis, we performed some preprocessing on the data.

For our clustering task, we will be using the K-Means and other clustering algorithms. For all the algorithms that work with distances is better to *standardize* the data, i.e. mean centring of the variable values with a unit variance. This ensures that all the variables are in the same range and the differences in ranges of values doesn't cause the algorithms to not perform well. This is akin to **feature scaling**.

Another problem is the huge range of values each variable can take. This problem is particularly noticeable for the monetary amount variable. To take care of this problem, we will transform all the variables on the *log scale*. This transformation, along with the standardization, will ensure that the input to our algorithm is a homogeneous set of scaled and transformed values.



(a) Single features distributions after log scaling

(b) Data points

Figure 41: Online Retail data distribution after preprocessing

4.2 Instacart dataset preprocessing

Also in the case of Instacart dataset the preprocessing steps are needed to obtain the features used in the RFM approach.

In this case we do not have data about the price of the items bought by the customers, so the idea was to create the three features basing them on different aspects that would capture similar and useful properties of user behaviour.

After some experimentation, the three features of RFM model were based on:

- **Recency**: average lag (in days) between orders per customer;
- **Frequency**: total number of orders per customer;
- **Monetary Value**: average size of orders (in products) per customer.

With this features we are able to capture how much the customer uses Instacart (although in this case, that usage is spread over an undefined period) and the average size of orders per customer is kind of a proxy for monetary value (the general assumption is that the size of an order might have something to do with its monetary value).

Even if the features don't map perfectly onto RFM model, they still capture a lot of important information about how customers are using Instacart.

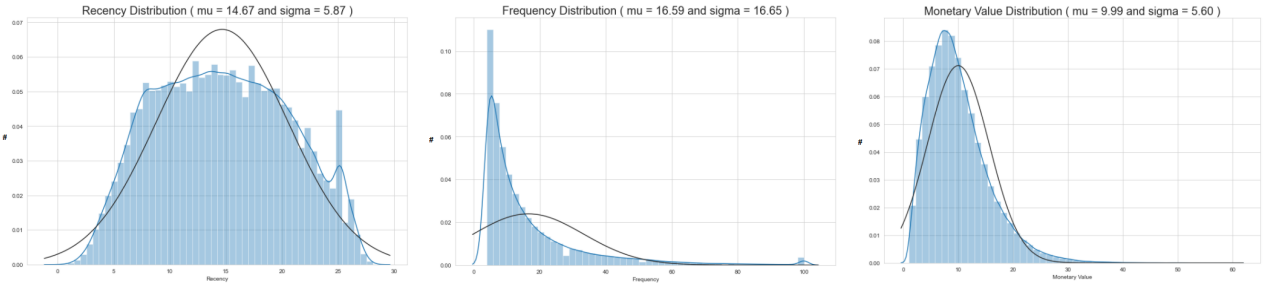


Figure 42: Instacart dataset, recency, frequency and monetary value distributions

As before log transformation and standardization are applied to the data with the aim of making the models less influenced by outliers and the different scales of the different features.

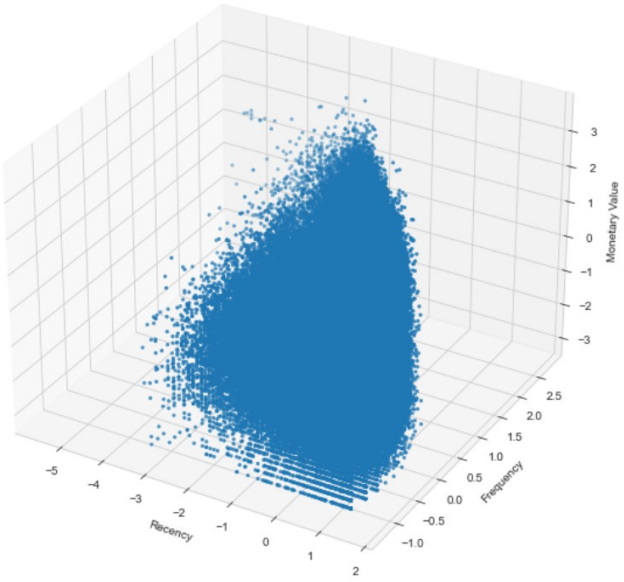


Figure 43: Instacart data distribution

4.3 Multi Category Shop dataset preprocessing

Also in this dataset the preprocessing steps are needed to obtain the feature used in the RFM model.

The first step needed is to extract purchase data from the whole dataset, then duplicate values and entries with negative price have been removed.

At this point, Recency (how many days had it been since the customer's last purchase), Frequency (how often had the customer made a purchase), and Monetary Value (how much did the customer spend) have been computed.

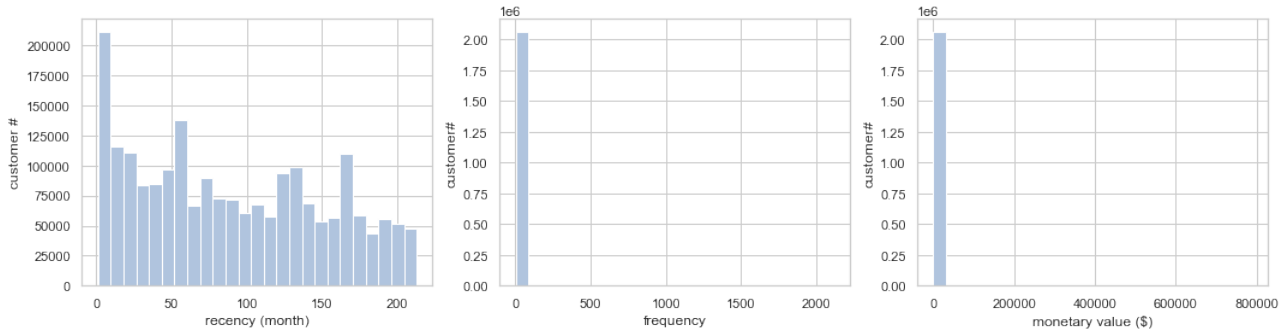


Figure 44: Multi Category Shop dataset, recency, frequency and monetary value distributions

Recency has a mean of 89 and a standard deviation of 62, it's the only feature without a skewed distribution. Frequency has a mean of 3, but a maximum of 2.119, however more than 50% of the customer made less than 2 orders. Monetary value has a minimum of 0,31 and a maximum of 788.338,60. We can see that there are lot of differences between the customers.

5 Experimental Settings

For this work Scala programming language (version 2.12.8) and Spark framework (version 3.0.1) has been used. Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs for Scala and other programming languages and an optimized engine that supports general execution graphs. Some Spark higher-level tools, including Spark SQL for SQL and structured data processing, MLlib for machine learning, has been used during the development of the project.

Jupyter notebook and Python have been used to do some data visualization because of the fact that are not available great data visualization libraries in Scala, such as Python's Seaborn and Matplotlib. However, some plot have been made using Scala Evilplot library [16].

Aws EMR has been used to experiment the deployment of the application on a cloud cluster. Amazon Elastic MapReduce (EMR) is an Amazon Web Services (AWS) platform for big data processing and analysis using famous open source tools like Apache Spark.

Moreover, Aws S3 has been used, it's a service for persistence of the datasets, code and generated output.

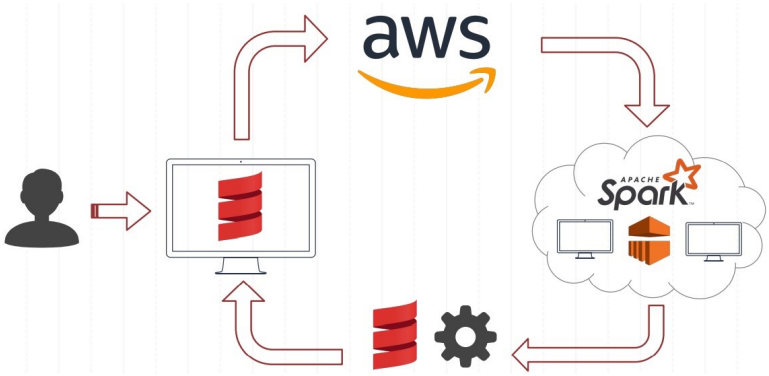


Figure 45: Working process

In the experimental studies different algorithms and different implementations have been tested:

- Standard K-Means implemented "by-hand" using different Scala language built-in functions, to compare the efficiency of them;
- Standard K-Means implemented in MLlib;
- K-Means|| also implemented in MLlib;
- Distributed DBSCAN implemented using available library on github [15];

6 Experimental Studies and Results

In the first part of the experimental studies we will analyse the computational cost of the algorithms and the pros/cons of using a cluster of nodes, here we will use both real datasets and synthetic ones to test the models. In the second part we will show the results obtained for the customer segmentation task using real datasets.

To compare the algorithms and the different runs of the same algorithm we will use time (in milliseconds, seconds or minutes), but we consider only the time spent in computing the clusters (e.g. computing centroids and associate each point to the nearest centroid). Loading of files, plotting clusters and save the results in csv files are not considered in time computation.

6.1 K-Means implementations local test

In this first experimental study we analysed the computational cost of different implementations of the K-Means algorithm using a synthetic file with 100.000 samples, created starting from 4 initial 2D points: (50, 50), (-50, 50), (50, -50), (-50, -50). The test has been made using Spark in **local mode** with an arbitrary number of threads.

To get the final time needed by the algorithms, an average of 10 runs has been made, so the results are not influenced by the starting random points that are chosen at the beginning of the algorithm.

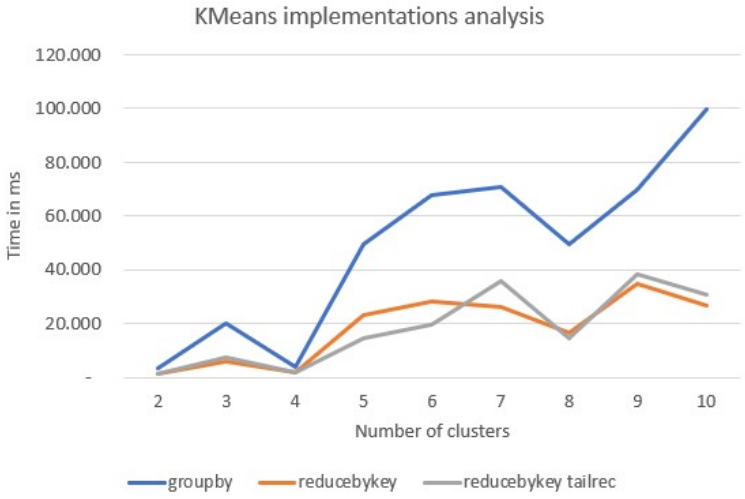


Figure 46: Analysis of "by-hand" implementations of K-Means algorithm

This first plot (figure 46) show three different implementations of the algorithm:

- the first uses the Scala built-in function *groupBy* inside a while loop to compute the centroids. This is clearly the most expensive solution because causes a great amount of shuffling and it's needed more computational time in combining the results in the main worker.
- the second uses Scala built-in function *reduceByKey* inside a while loop to compute the centroids. This is a less expensive solutions because data are combined in each partition and only one output for one key at each partition needs to be send over the network, reducing the shuffling and augmenting the parallel computation. The difference w.r.t. the first solution is huge, especially when the number of cluster is higher. The "groupBy model" seems to have an exponential cost as the number of cluster increment.
- the third solution uses, as before, the built-in *reduceByKey*, but in this case following the functional approach a recursive implementation replaces the while loop. From the line plot we can see that the computational time is similar to the previous solution, for this reason we can say that "looping" and "recursive" solutions are equivalent.

Another thing that is clearly visible is that computing four clusters in case of synthetic points generated by 4 initial centroids requires less iterations and for this reason less computational time.

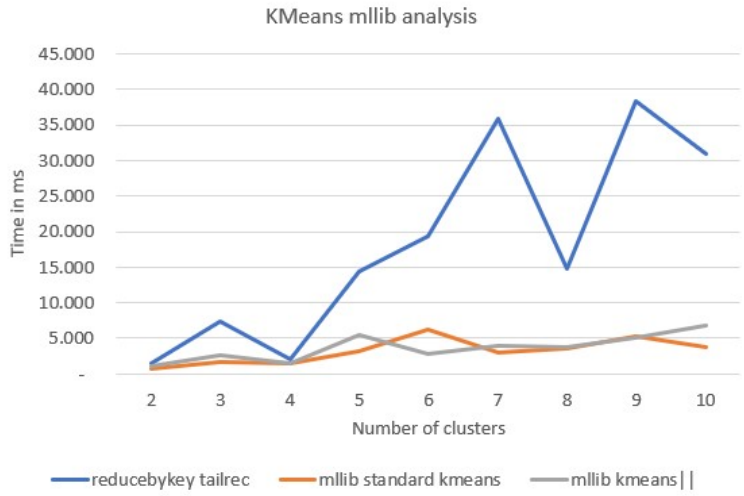


Figure 47: Analysis of MLlib implementations of K-Means algorithm

After the analysis of K-Means algorithm implemented "by-hand", we compared the results of the "reduceByKey model" with those obtained by using the Spark library MLlib.

The difference is great especially in the case of high number of clusters, the MLlib implementations are more efficient. The two models of standard K-Means and K-Means|| need almost the same time in all the cases. This could be due to points distributions, the better initialization given by the K-Means|| is not necessary in case the clusters are clearly separable.

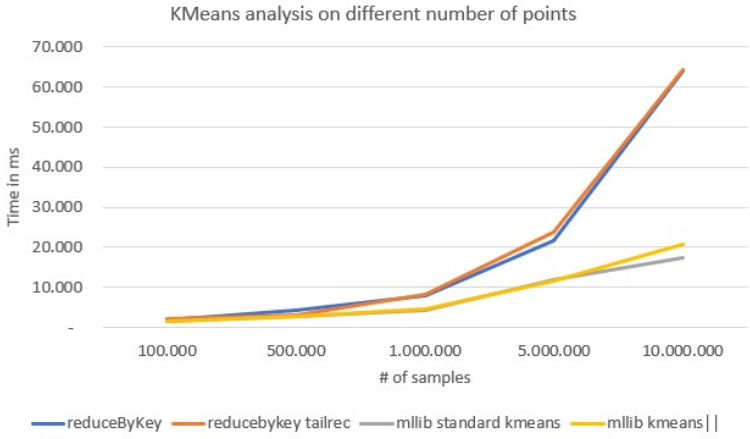


Figure 48: Analysis of K-Means algorithms with different numbers of samples

From this third analysis of the K-Means algorithms we can see that increasing the number of samples, not surprisingly, means higher computational time needed to find the cluster centres. The difference between the implementations of Spark MLlib and the other two increases as the number of observations became higher and higher. Also from here, we can see that the difference between standard K-Means and K-Means|| is not high.

Now we can analyse the performances of the algorithms by testing them on points with more than only two dimensions, these are generated as before by starting from 4 initial points clearly separated.

# features	reduceByKey	mllib standard kmeans	mllib kmeans||
2	1.915	1.563	1.507
4	1.255	1.876	2.460
6	1.477	2.341	3.926
8	2.169	2.081	3.011

Figure 49: Analysis of K-Means performances on 100.000 samples with different number of features

The table shows that with 100.000 samples the time needed to compute the clusters does not varies a lot, we can only observe a little increment for K-Means||, possibly due to the higher cost of initialization procedure with more dimensions in the vectorial space.

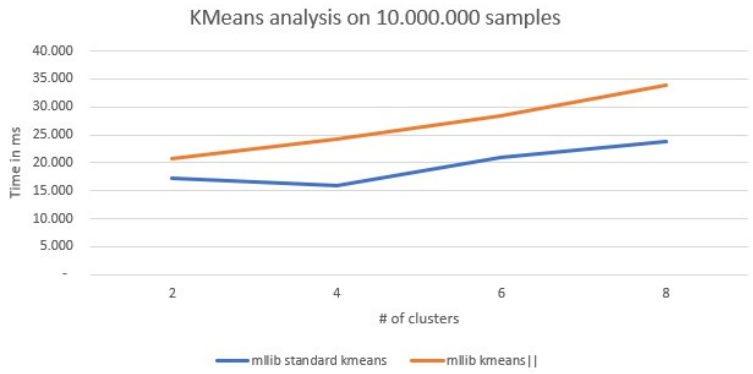
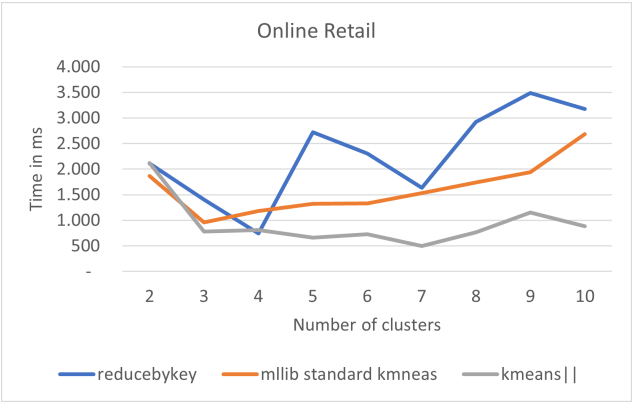


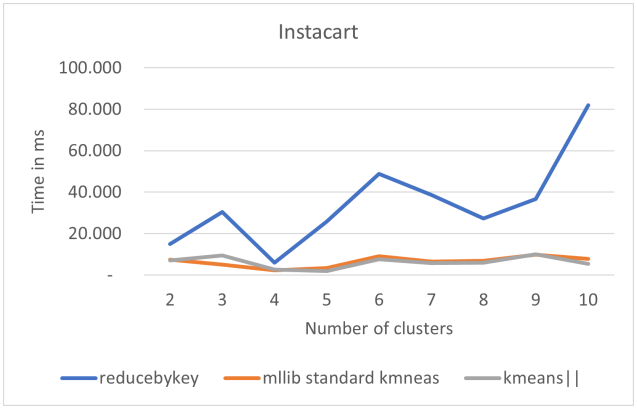
Figure 50: Analysis of K-Means performances on 10.000.000 samples with different number of features

In figure 50 is easy to note that increasing the number of dimensions when the number of points is high means more computational time needed to compute the cluster centres. The standard K-Means require less time than the other algorithm.

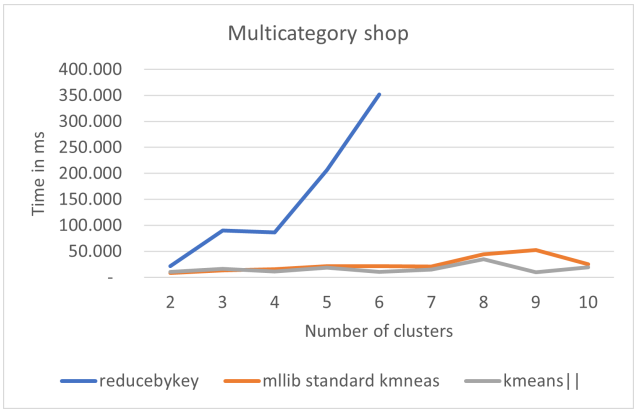
The "by-hand" implementations is not showed because of the poor performances, it requires ∼100.000 millisec in the case of 4 dimensions and ∼1.000.000 millisec with 6 dimensions, i.e. a huge computational cost if compared to MLlib algorithms.



(a) Online Retail



(b) Instacart



(c) Multi Category Shop

Figure 51: Analysis of K-Means performances on real datasets

The analysis of the algorithm implementations on real data shows similar behaviour, with the MLlib implementations much more efficient. However, in this case, the K-Means|| gets better results w.r.t. standard K-Means, especially in the first and in the last cases. This could mean that in case that the points do not form clearly distinct clusters, the better initialization obtained in the first case makes the algorithm converge faster.

6.2 DBSCAN local test

In this second experimental study we analysed the performances of DBSCAN algorithm and we compared them with those obtained with K-Means algorithm.

The hyperparameter of DBSCAN have been chosen in such a way to obtain 4 clusters with the correct number of observations, this is easily verifiable because the clusters are separated from each other.

Number of points	MLlib K-Means||	DBSCAN
100.000	1.507	11.583
500.000	2.802	67.060
1.000.000	4.460	173.879
5.000.000	11.588	1.749.885
10.000.000	20.819	7.424.733

Table 1: DBSCAN computational time (in millisec) needed with different number of samples

By looking to the table is clear that DBSCAN require much more time w.r.t. the K-Means|| algorithm. This is not a surprise, we know that K-Means is very popular due to its simplicity and its speed.

What we can point out is that the difference increases dramatically as the number of samples increases, if with 100.000 observations K-Means require approximately 1 second and DBSCAN 11, then with 1.000.000 of samples the former needs 4 seconds and the latter 3 minutes. Until this point the difference is high but not sensational. However, in the case of 5.000.000 of samples K-Means requires 11 seconds and DBSCAN needs 30 minutes, that is a very big difference. In the last case the distance is huge with the former that requires 20 seconds and the latter approximately 2 hours.

The computational cost of density-based clustering is exponential, while the cost of K-Means|| seems more linear.

Number of features	MLlib K-Means||	DBSCAN
2	1.507	11.583
4	2.460	36.987
6	3.926	186.101
8	3.011	694.173

Table 2: DBSCAN computational time (in millisec) needed with different number of features

This second table shows that, for DBSCAN, increasing the number of dimensions causes a considerable increase in computational time, instead this is not the case for the K-Means algorithm.

This higher cost is due to the computation of the different non-overlapping boxes that require more resources when there are more dimensions and to the cost associated to the merging of clusters due to points that fall within two boxes borders.

At the end we tested the algorithm on real dataset:

Dataset	DBSCAN time	epsilon	minPts
Online Retail	1.261	0,30	10
Instacart	75.739	0,18	20
Multi Cat. Shop	18.578.214	1,45	250

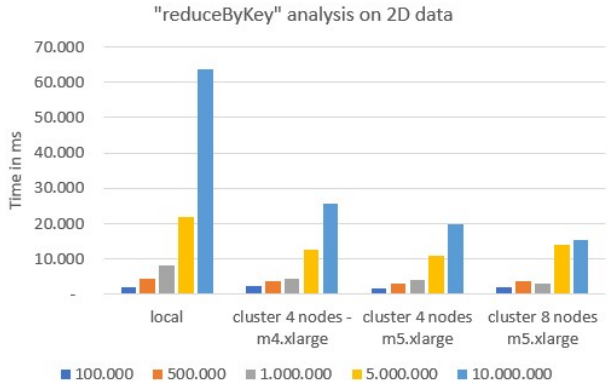
Table 3: DBSCAN computational time (in millisec) on real datasets

The results do not differ too much to what we have already seen on synthetic datasets, the cost is huge if compared to K-Means, especially for the third dataset that has 2 million of samples. Moreover, for the last dataset the cost is huge also if compared to the synthetic dataset with 10 million of observations, this could be due to the fact that the data are less separable than those in the synthetic ones and this causes more costs, but also could be due to the fact that the real datasets have 3 dimensions instead of 2.

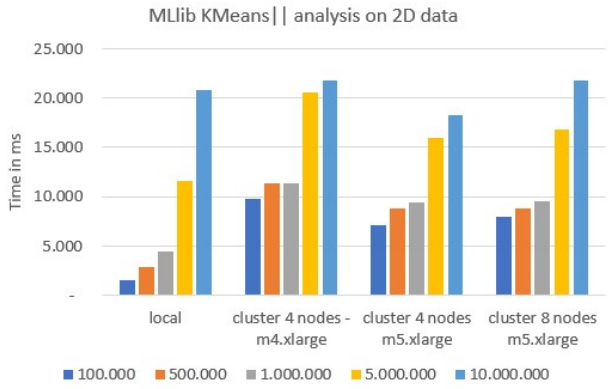
6.3 K-Means cluster test

In this experiment we analysed the variation of computational time when the application is deployed on clusters of different dimensions and with different computational power.

We started testing different clusters on synthetic datasets of 2D points and subsequently we used the real dataset. Only results for K-Means|| will be shown, those for MLlib standard K-Means are omitted as they are very similar to the former.



(a) "reduceByKey" model



(b) K-Means||

Figure 52: Analysis of performances for 2D sets of points on different types of cloud clusters

The analysis on synthetic datasets reveal that for small dataset, not surprisingly, is better to work in local mode, the time needed is slightly higher in case of "reduceByKey model" (e.g. 1.915 ms in local mode when there are 100.000 points, but 2.278 ms with a cluster of 4 nodes) and much higher in case of K-Means||, for 100.000 samples in local mode are needed 1.507 ms, but with a cluster of 4 nodes (m4.xlarge) the time increases to 9.717 ms.

For the "reduceBykey model" the time starts improving significantly with 1 million of points, it goes from 7.976 in local mode to 4.308 in cluster 4 nodes - m4.xlarge and even better in the next two cases.

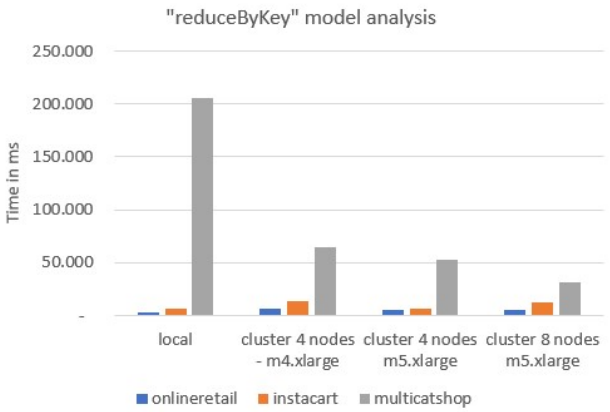
In this case increasing the computational power by using the more powerful m5.xlarge hardware produces a significant decreases in total time, especially when the number of samples is very high. In case of 10 million of points the time goes from 25.534 ms to 19.636 ms.

Also incrementing the number of nodes of the cluster leads to better results in case of high number of samples, in case of 10 million of points the time goes from 19.636 ms to 15.342 ms.

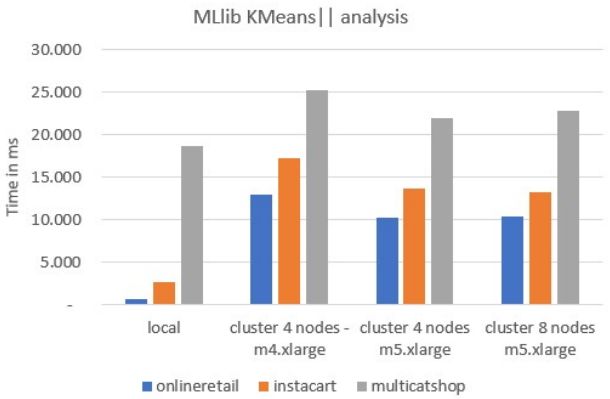
Instead in the case of K-Means|| the performances are always better in local mode over those obtained with the cluster 4 nodes - m4.xlarge. However, we can see a small improvement in case of the cluster 4 nodes - m5.xlarge hardware, but only in the case of 10 million of points.

In this second case increasing the number of nodes to 8 does not improve the results, this could be due to the higher shuffling that is present with a greater number of nodes.

The "reduceByKey model" with a cluster of 8 nodes and m5.xlarge hardware gets the better results in the case of 10 million samples, so it seems to scale better than MLlib model.



(a) "reduceByKey" model



(b) K-Means||

Figure 53: Analysis of performances for real datasets on different types of cloud clusters

The test of the clusters on real dataset seems to show the same pattern highlighted before. In the "reduceByKey model" we have an improvement only for the Multi Category Shop dataset, but this is

very important, the models goes from the 206.114 ms of the local execution to the 30.633 in the case of cluster with 8 nodes.

For the datasets with less samples, the local execution always produces better results.

In the case of K-Means|| the local mode is always better, the cluster of 4 nodes is the worst one, but also adding more node to the cluster does not improve the results.

What we can see in all the cases is that increasing the hardware's computational power (going from m4.xlarge to m5.xlarge) means decreasing the time needed to clustering the points.

6.4 DBSCAN cluster test

In this experiment we analysed the variation of computational time when the DBSCAN algorithm is executed on clusters of different dimensions and with different computational power.

We started testing it on synthetic datasets and subsequently we used the real ones.

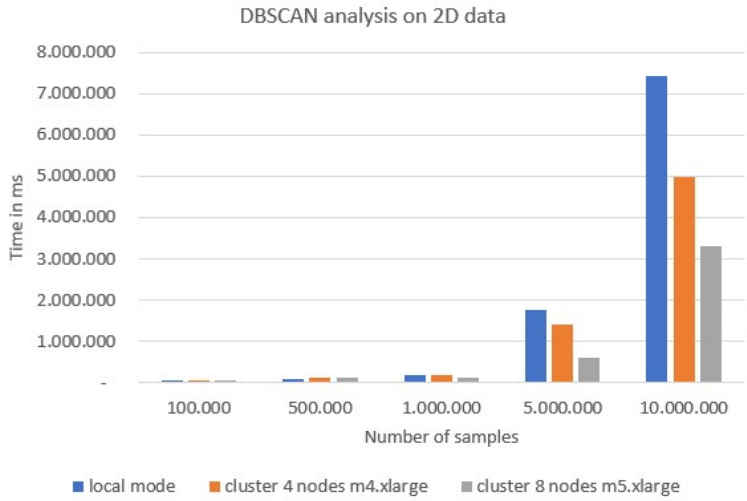


Figure 54: Analysis of DBSCAN algorithm with different numbers of samples on different types of cloud clusters

The test on 2D data shows that the time needed to segment the points by DBSCAN decreases considerably when the number of samples is very high, between the local mode and the cluster of 4 nodes there are almost 40 minutes of difference and other 25 if we consider the cluster of 8 nodes.

The time starts decreasing with 1 million of samples, but the difference is not noticeable, instead with less points the time needed is higher. This shows that is useful to work in a cloud only if the amount of data is sufficiently high.

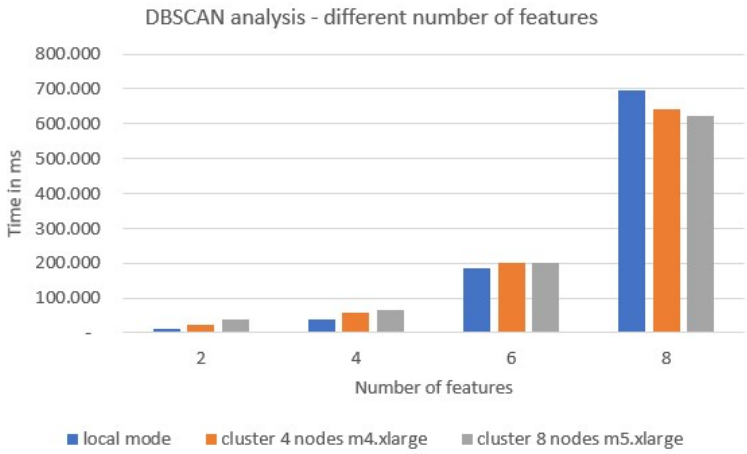


Figure 55: Analysis of DBSCAN algorithm on 100.000 samples with different number of dimensions (features) on different types of cloud clusters

The test on 100.000 points of different dimensions reveals that only in the last case working in a cloud brings benefits, in the other cases it's better to work in local mode.

We can see a common pattern in our experiments, when the time needed to solve the task is low,there are no benefit from running on the cloud, this is probably due to network latency that affect the final result, even when we are able to make more parallel computation.

Dataset	local mode	cluster 4 nodes m4.xlarge	cluster 8 nodes m5.xlarge
Online Retail	1.261	13.030	17.807
Instacart	75.739	85.320	90.994
Multi Cat. Shop	18.578.214	16.987.579	15.058.977

Table 4: DBSCAN computational time (in millisec) on real datasets on different types of cloud clusters

The test on real datasets shows the same pattern seen on synthetic ones, the computational time is higher, using a cloud cluster, when the number of samples is low, but is much more lower when there are many samples w.r.t. local mode, such as in Multi Category Shop dataset.

6.5 Customer Segmentation Results

By analysing the results obtained with the clustering algorithms we were able to define the different cluster segments and the marketing strategies that can be applied to them.

To identify the optimal number of clusters in K-Means we have used the *elbow method* (as described in section 1.1.3). An analysis of the statistical relevant measures (e.g. average, percentiles, min, max, ecc.) and the visualization of the results through different kind of plot have been used to define the characteristics of each single cluster. Only the K-Means|| has been used to compute the results.

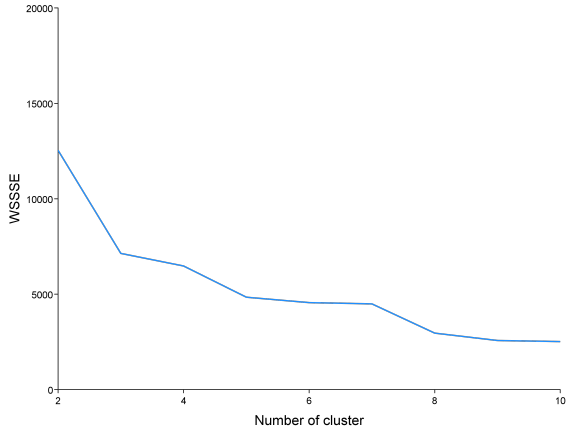
In the case of DBSCAN to choose the parameters of the model we made the two plot, 1. Distance to the nearest neighbour and 2. Number of nearest points given a specified epsilon, to get insights of which values can be valuables ones. Then by analysing the results though plots and comparing them with those obtained using the K-Means algorithm, we decided whether to modify these parameters and re-run the algorithm or not.

We have decided to not use the *silhouette score* because this is not very scalable, since it requires measuring the average distance from each point to every other point in its cluster, as well as the average distance from each point to every other point in all the other clusters. This is clearly an $O(n^2)$ operation and not likely to scale for large N, so we preferred to avoid it in our experimental studies.

Types of customers that we have considered in our analysis of results and the appropriate marketing strategies are:

- **Best/Core Customers**: Bought most recently, most often and spend the most. Marketing → No price incentives, new products or loyalty programs. Focus on value added offers through product recommendations based on previous purchases.
- **Loyal Customers**: Buy frequently and spend a good amount of money. Marketing → Advocacy programs and reviews are common strategies. Lastly, consider rewarding these customers with Free Shipping or other like benefits.
- **Big Senders**: They do not buy frequently, but they spend the most. Marketing → Market the most expensive products, premium offers, subscription tiers, luxury products, or value add cross/up-sells to increase revenues. Don't waste margin on discounts.
- **Promising Customers**: Customers who are in some way new (low recency and frequency), they didn't spend much. Marketing → Start building relationships with these customers by providing onboarding support and special offers to increase their visits.
- **Almost Lost**: Customers that haven't purchased for some time, but purchased frequently and spend the most in the past. Marketing → Aggressive price incentives and new products launches.
- **Lost Cheap Customers**: Last purchased long ago, purchased few and spent little. Marketing → Don't spend too much trying to re-acquire them.

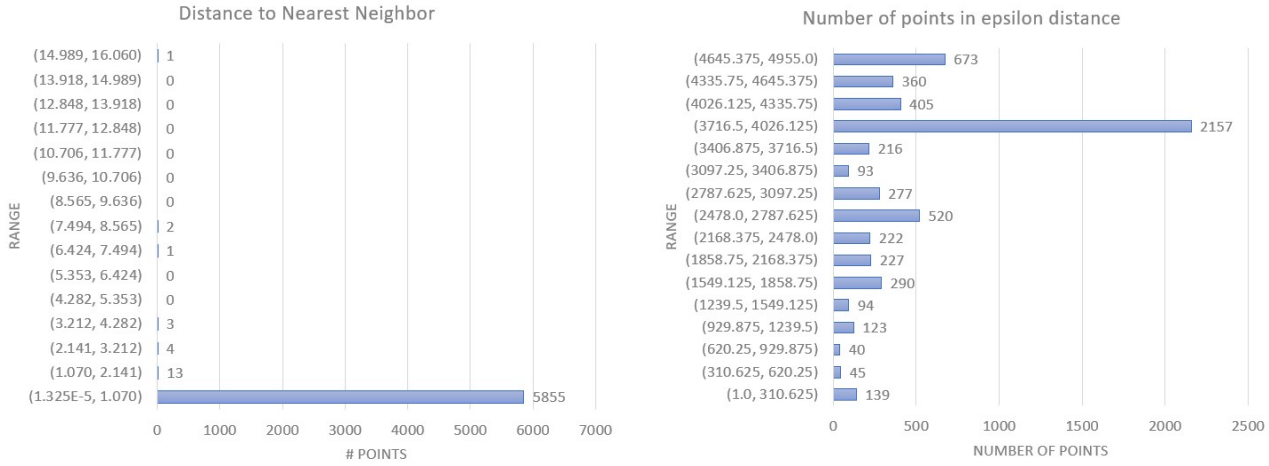
6.5.1 Online Retail Segmentation



By looking to the graph we have chosen to define 5 different clusters, this is the point where there is the "elbow" inside the graph and this is a reasonable number of cluster for segmenting customers.

Instead, to choose the parameters of DBSCAN, we used the two plots showed in figure 57 and we setted $\epsilon = 0.3$ and $minPts = 10$.

Figure 56: Online Retail dataset WSSSE per number of clusters



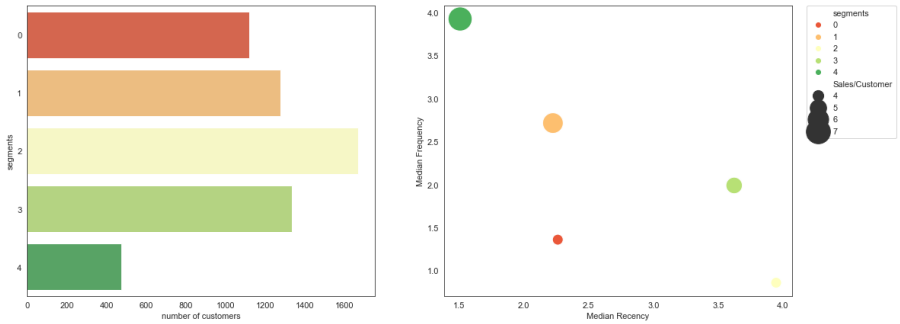
(a) Distance to the nearest neighbour

(b) Number of neighbour points in ϵ distance

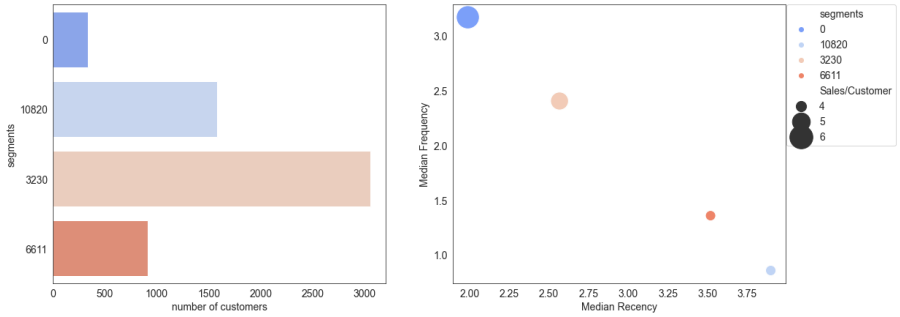
Figure 57: Online Retail data analysis to choose parameters for DBSCAN

By looking to the two plots in figure 58 we can see that the results obtained with the two models are quite different. In the fist case 5 clusters have been created with one of them that has a less number of samples associated to it, in the second case 3 clusters were defined by the model with the second much more numerous than the other two. In the second case the segment with cluster Id equal to 0 are the samples treated as outliers by the model, to include this in one clusters the ϵ parameter needs to be increased, but increasing that param causes the merging of the other clusters, resulting in one single big cluster that is useless in our analysis.

In the case of K-Means the cluster number 4 is the cluster with higher values for Frequency and Monetary Value, it also has the smallest values for Recency. In the case of DBSCAN is the cluster 0 that shows a similar tendency, so the extreme values for these features make that these samples are treated as outliers.



(a) K-Means



(b) DBSCAN

Figure 58: Online Retail segments summary

We can see that the extreme values, very high or very low, for all the three features make the samples classified as outliers also by looking to the figure 59.

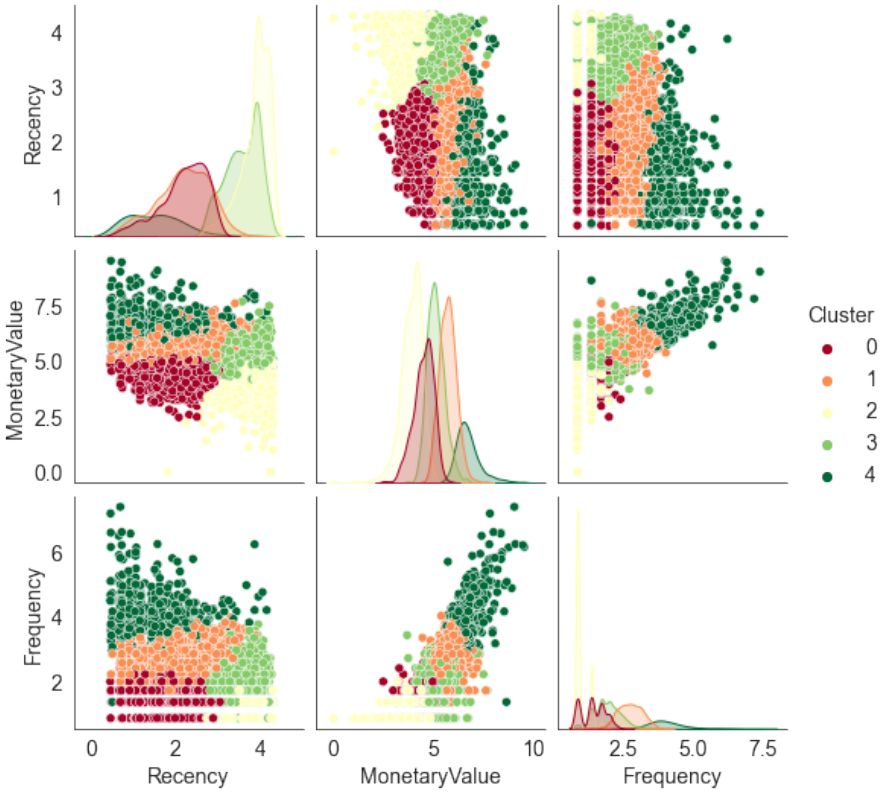
A first thing that we can notice is that Frequency and Monetary Value are positively correlated, higher frequency means higher monetary value.

The results in the two cases are very different from each other, but it's clear that K-Means produces results more useful for the task of customer segmentation.

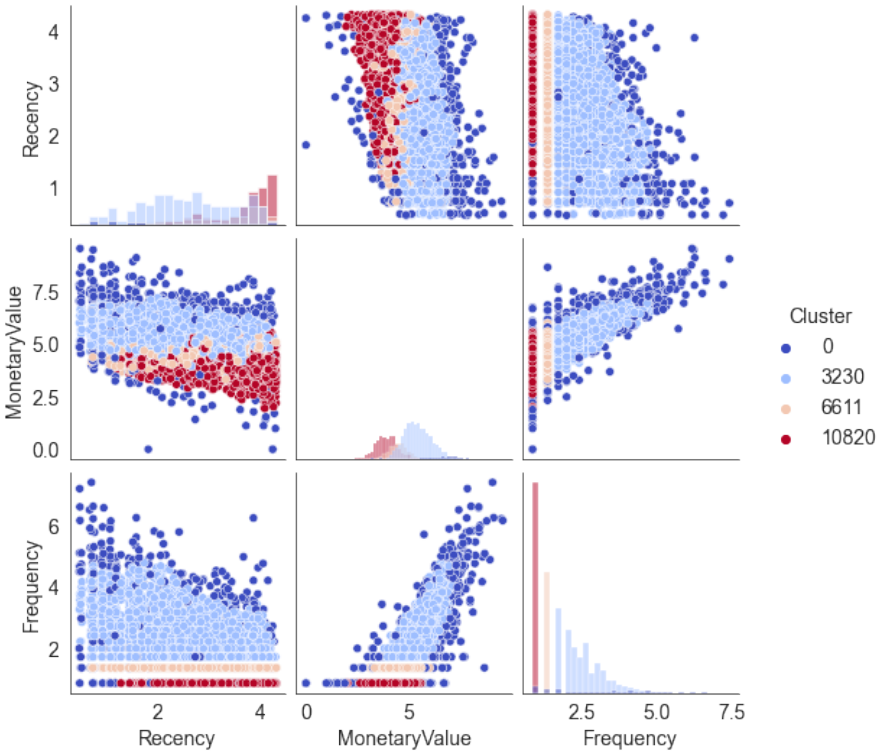
The difference among the clusters is clear if we look to the box-plots in figure 60, in the case of DBSCAN the cluster with id 10820 is that with higher Recency, that is not good because means that is much time that the customers do not buy from the shop, but with low Frequency and Monetary Value. Therefore, this cluster represents the less interesting customers for the shop. Between the other two clusters is clear that those with color light blue contains the more loyal users, that buy more frequently and spend more.

However, it's more interesting to analyse the results obtained with K-Means:

- Cluster with id 0 (Red) contains customers that make low-cost purchases, with a relatively low frequency, but above 1, and made their last purchase more recently, so they can be newest users. We can categorize them in *Promising*.
- Cluster with id 1 (Orange) represents users with a decent spend, but not as frequent as the cluster 4. We can categorize them in *Loyal customers*.
- Cluster with id 2 (Yellow) shows the customers with lower frequency and monetary value and at the same time with higher recency. These are *Lost cheap customers*.
- Cluster with id 3 (Light Green) represents users that purchase medium amounts, with a relatively low frequency and not very recent. We can categorize them in *Almost lost*.
- Cluster with id 4 (Green) is the segment of customers who shop often and with high amount. These are surely the *Best/Core customers*.

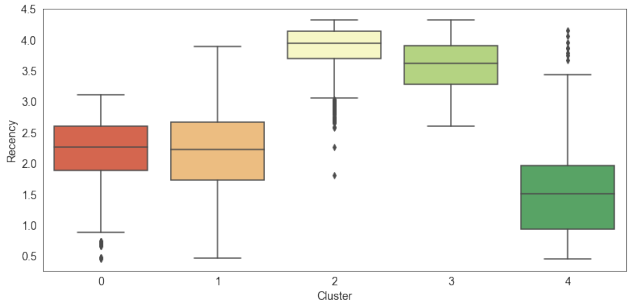


(a) K-Means

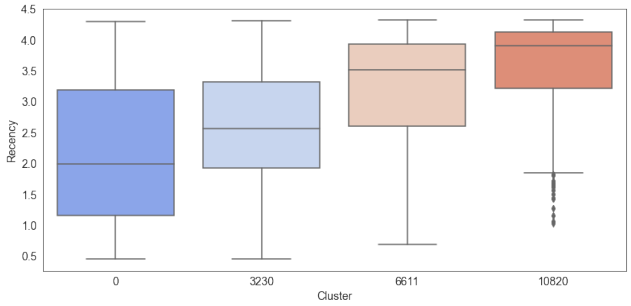


(b) DBSCAN

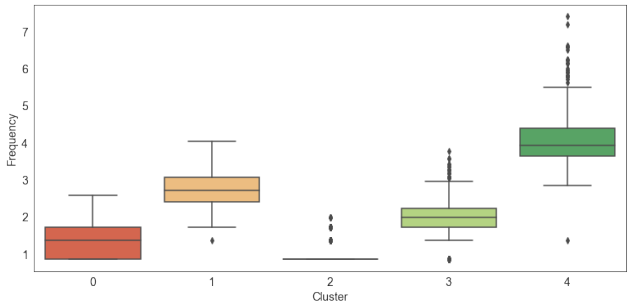
Figure 59: Online Retail customers distribution



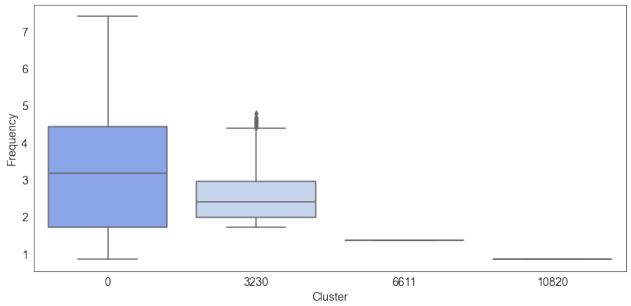
(a) K-Means Recency



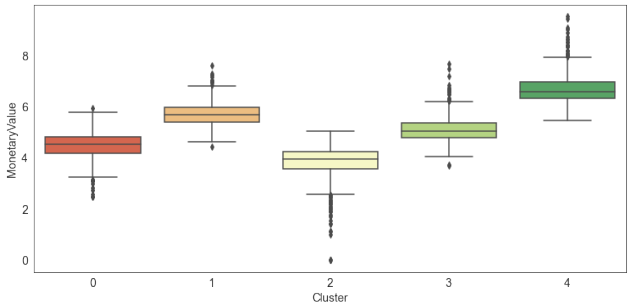
(b) DBSCAN Recency



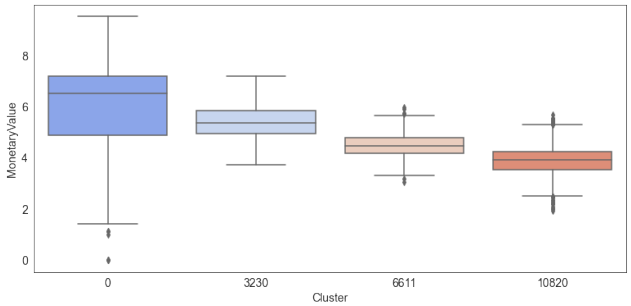
(c) K-Means Frequency



(d) DBSCAN Frequency



(e) K-Means Monetary Value



(f) K-Means Monetary Value

Figure 60: Online Reatil Rececny, Frequency and Monetary value distribution per cluster

6.5.2 Instacart Segmentation

For this and the last dataset we will not show the results obtained using the DBSCAN model as they are not useful for the definition of the segments of customers.

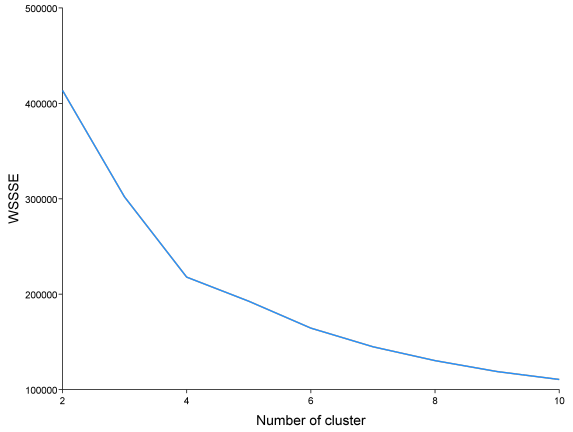


Figure 61: Instacart dataset WSSSE per number of clusters

By looking to the graph is clear that the "elbow" is where there are 4 clusters. So, in this second case the number of cluster is lower than those for Online Retail dataset.
From the two plot in figure 62, we can see that, as before, the last cluster is those with less number of samples, but also it's the one with high value (small recency, high frequency and monetary value). From this first plot the two clusters Yellow and Light Green seems to be quite similar from each other.

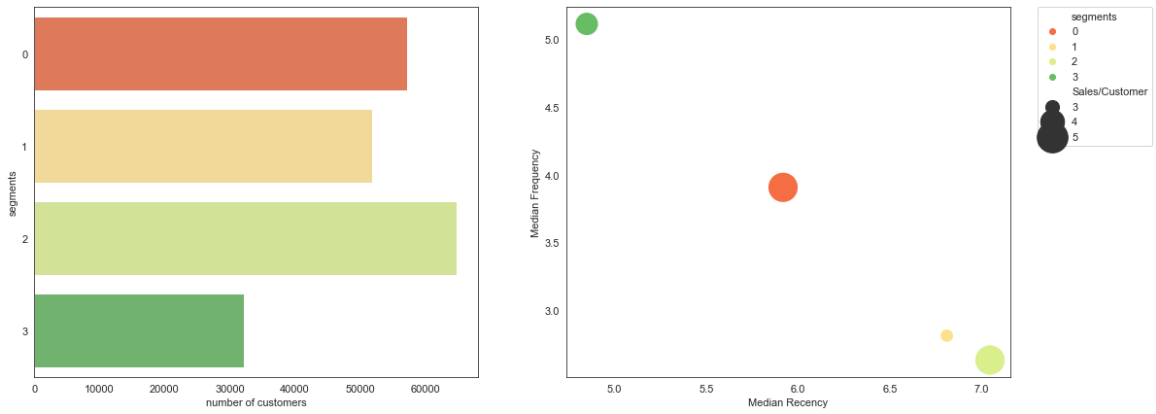


Figure 62: Instacart segments summary

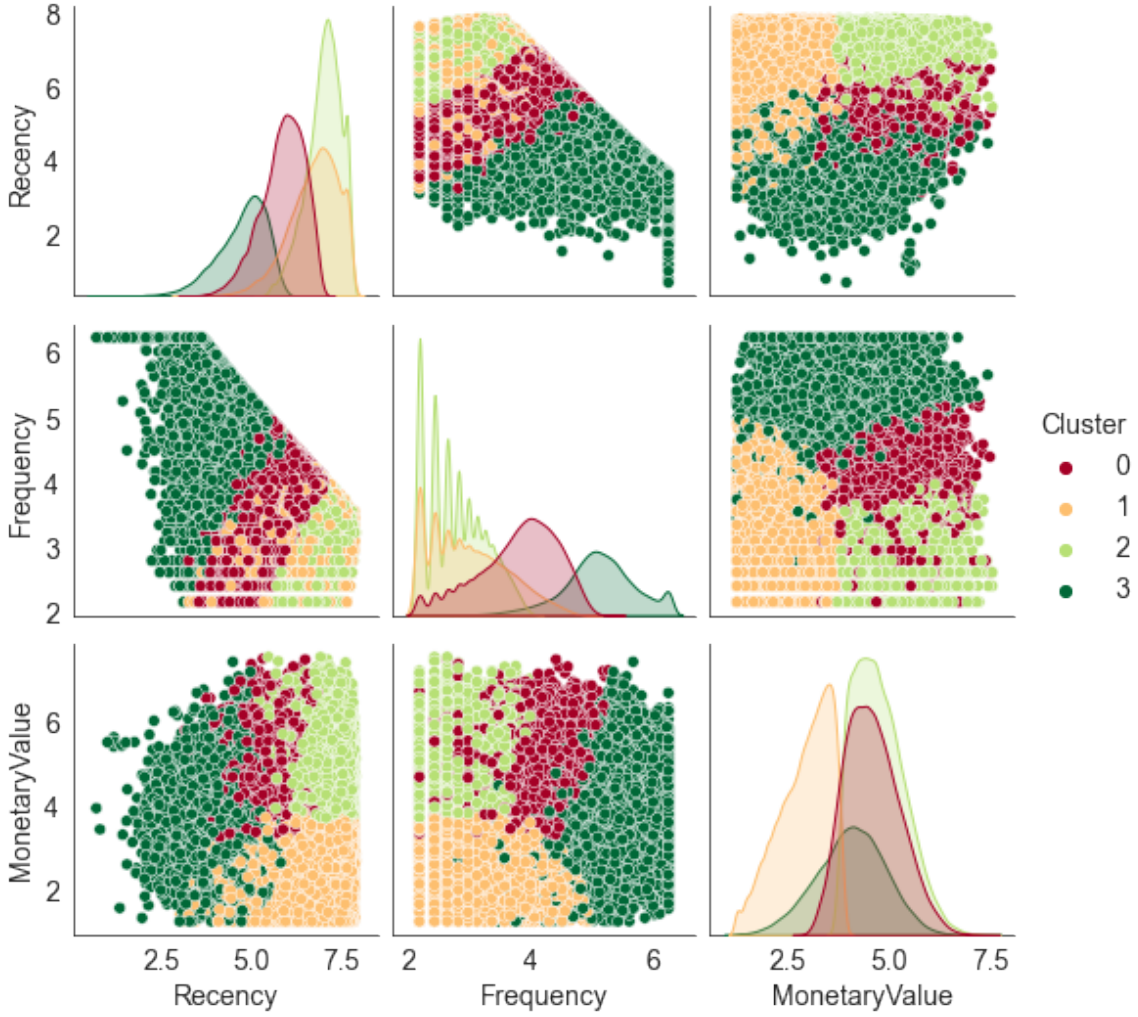
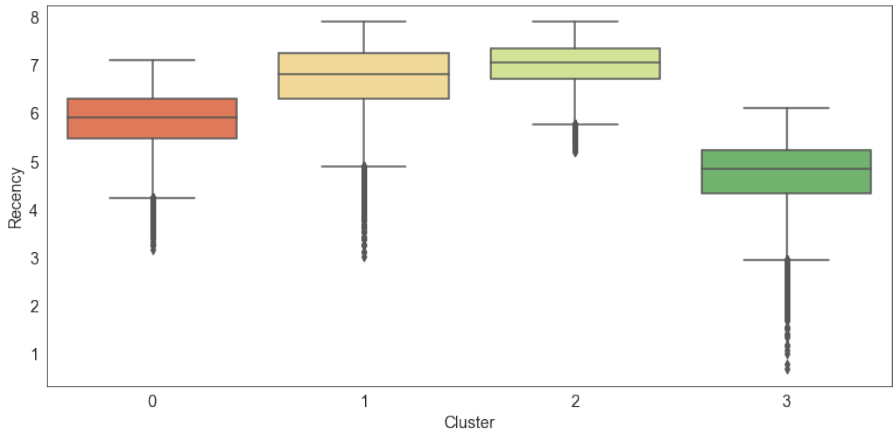


Figure 63: Instacart customers distribution

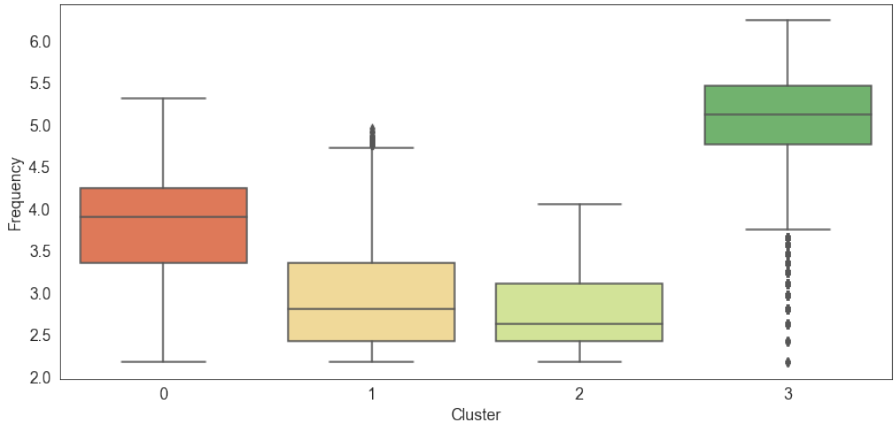
By looking to the scatter plots (figure 63) and to the box plots is clear the difference between the four clusters:

- Cluster with id 0 (Red) contains samples with medium recency and frequency, but also a high monetary values. We can classify this segment of customers as *Loyal customers*.
- Cluster with id 1 (Yellow) represents users with high recency, low frequency and especially low monetary value. So, these are customers that bought few times in the past and do not spend a lot, we can classify them as *Lost cheap customers*.

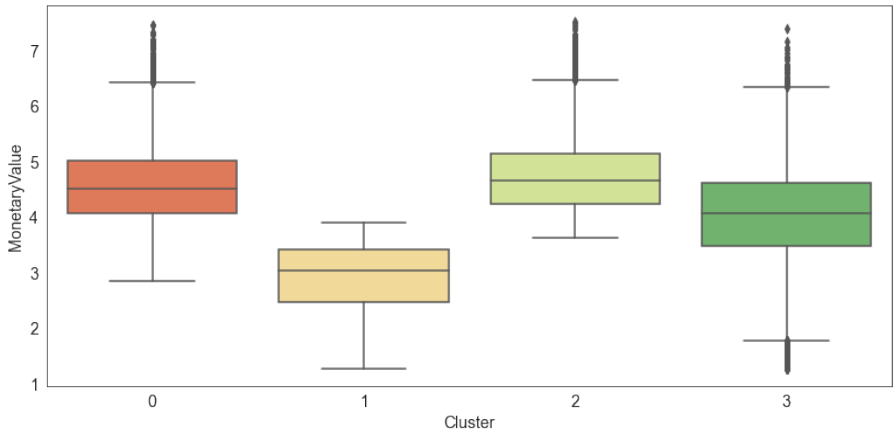
- Cluster with id 2 (Light Green) shows similar metrics to those of cluster 1, but it contains users with higher monetary value, therefore this customers are more important for the business. We can categorize them as *Great spenders*.
- Cluster with id 3 (Green) contains the customers that bought more recently and with clearly the higher frequency, but also this segment have a decent spend. We can categorize them as *Best/Core customers*.



(a) Recency



(b) Frequency



(c) Monetary Value

Figure 64: Instacart Rececny, Frequency and Monetary value distribution per cluster

6.5.3 Multi Category Shop Segmentation

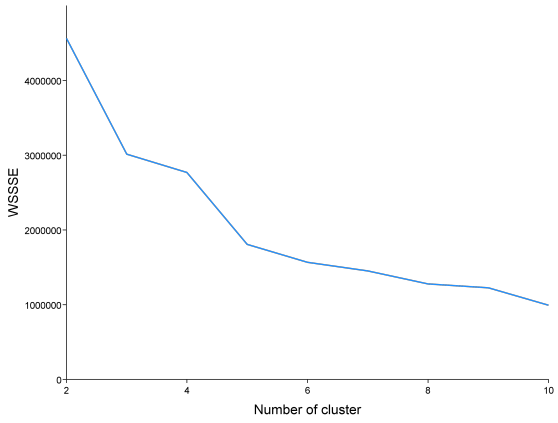


Figure 65: Multi Category Shop dataset WSSSE per number of clusters

Last dataset tested is Multi Category Shop dataset, the one with greater number of samples. As in the case of Online Retail a good number for clustering seems to be 5 by looking to the line plot.

The distribution of users among the 5 cluster is not uniform, as shown in figure 66, the cluster 0 is that with clearly the highest number of customers, more than 700.000, and the cluster number 1 has the lowest with only ≈180.000 observations.
Moreover, by looking to this first plot the two clusters 0 and 4 (Red and Green) seems to be very similar from each other, indeed the two dot are overlapped in the second plot.

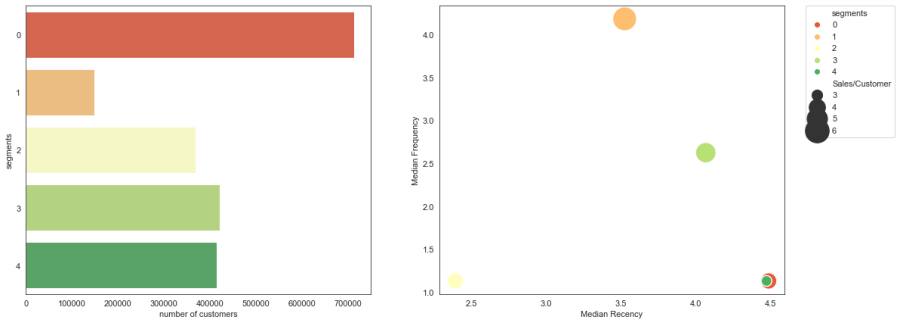


Figure 66: Multi Category Shop segments summary

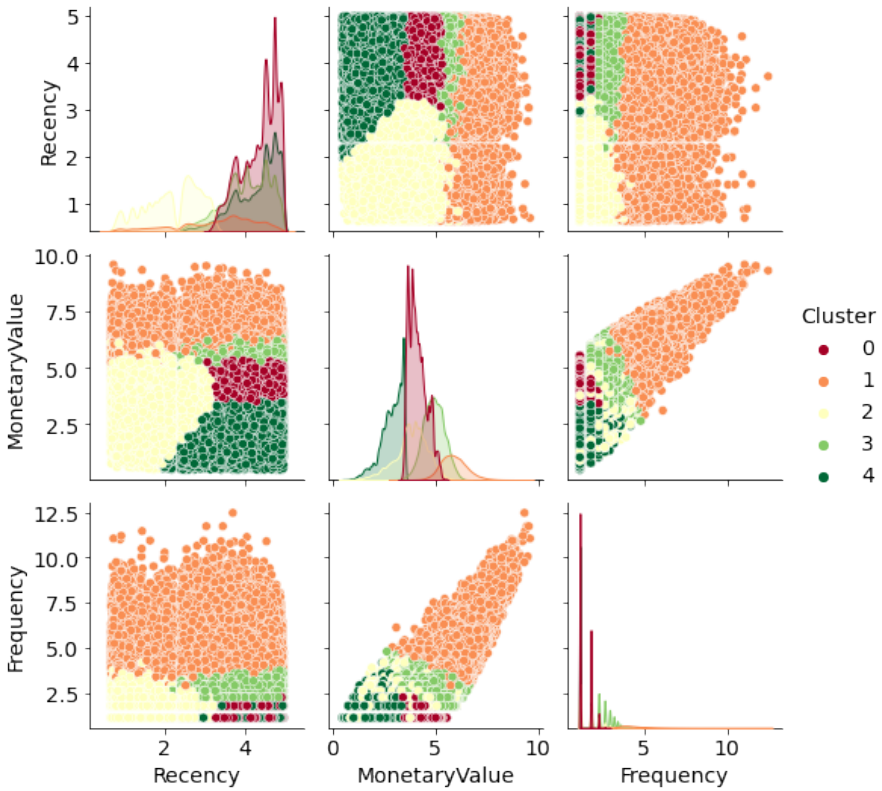
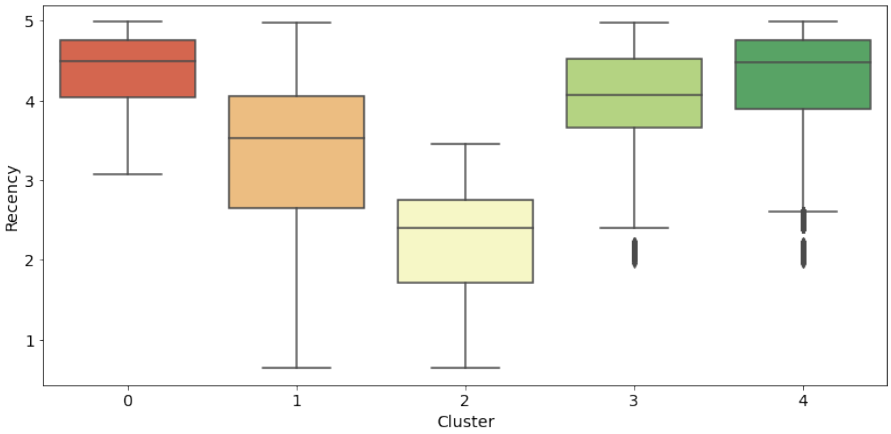


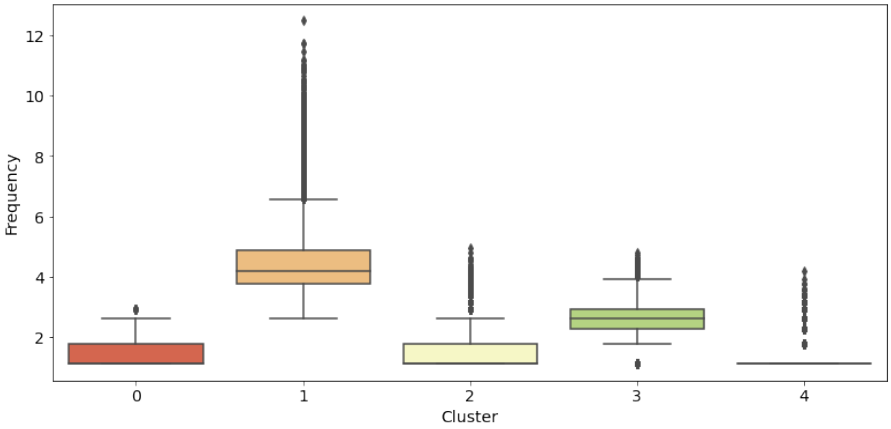
Figure 67: Multi Category Shop customers distribution

By looking to the scatter plots and the box plots we can define this classification:

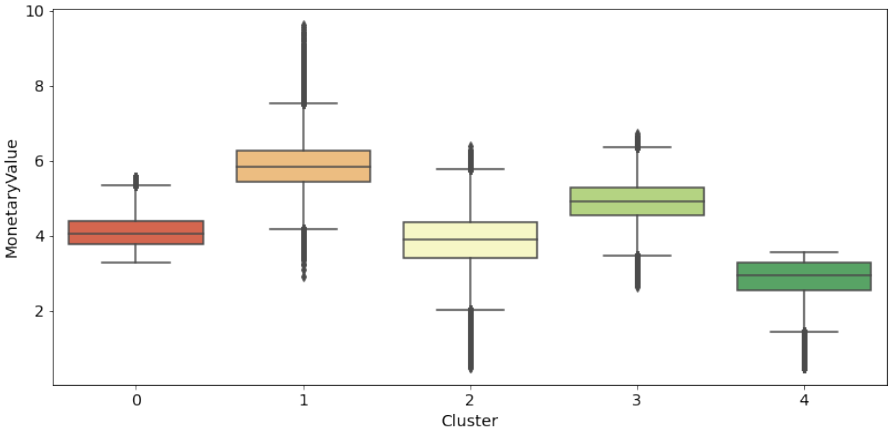
- Cluster with id 0 (Red) contains customers with high recency, low frequency and a medium-low monetary value. These can be categorized as *Almost lost customers*.
- Cluster with id 1 (Orange) represents the customers with clearly the higher frequency and monetary value. These are the *Best/Core customers* for the business.
- Cluster with id 2 (Yellow) shows a low frequency and a low monetary value, but also the recency is small, this means that the clusters contains customers that are new and for this reason they have low frequency. We can categorize this segment as *Promising*.
- Cluster with id 3 (Light Green) contains samples with high recency, medium frequency and good monetary value, so these are customers that bought a lot in the past and the business needs to try to re-acquire them. We can classify these users as *Almost lost*.
- Cluster with id 4 (Green) as said before has similar patterns to cluster 0, but the difference is that the monetary value is lower in this case. Therefore, we can categorize this group as *Lost cheap customers*.



(a) Recency



(b) Frequency



(c) Monetary Value

Figure 68: Multi Category Shop Rececny, Frequency and Monetary value distribution per cluster

7 Conclusions

We tested and analysed different algorithms, what our experiments show is:

- MLlib implementations of K-Means are very efficient, also when working in local mode. However there is no clear benefit in run it on a cluster, also when the amount of data is great. This could be due to the fact that the time needed by the algorithms is very low and the latency of the network could lead to worst performance
- Moreover, the difference between MLlib's Standard K-Means and K-Means|| is not so great in terms of time needed to compute the clusters. The time is quite the same in all the cases, the better initialization given by the K-Means|| requires more computation and does not lead to improvements in terms of total time, even if it is worth pointing out that the main advantage of this implementation is the better quality of the final solution
- MLlib's implementations also show that more machines doesn't mean always faster computations, indeed in the case of 8 nodes the performance are worst that those obtained with 4 nodes
- "by-hand" implementation of K-Means is clearly the less efficient if we look to the local executions, but it seems more scalable. Indeed, with this solution we obtain the best results on the synthetic dataset with 10 million of points. But also, the time needed to compute the clusters relative to the Multi Category Shop dataset in the case of 8 nodes is not so distance to the best performance of K-Means||
- DBSCAN is a much more complex algorithm w.r.t. K-Means, it requires an exponential time both when the number of samples or the number of dimensions (features) increases
- A more complex algorithm, such as DBSCAN (it requires more complex calculations), has more benefits from the distributed computation through a cluster of nodes. The difference become huge when the number of samples increases, the time needed to solve the task requires a few hours less
- Therefore, with complex algorithm is very useful to scale the computation on a cluster of node, with more simple and fast algorithms, like K-Means, it seems to be not so helpful unless you have hundreds of millions of data

What we want also to highlight is that DBSCAN seems not to be a suitable algorithm for customer segmentation, it's better to use other algorithms, such as K-Means, that do not classify samples as noise and produce better results when the data are not clearly separable, as in our experiments.

These kind of advanced algorithms are very useful to get insights about grouping in an easy and fast way. Groups made in an automated ways can be useful to discover pattern that marketers might find difficulty discovering on their own.

RFM model seems a valuable tool, each group of customers identified has precise characteristics and we are able to define precise marketing strategies to address them in the most effective way.

However, when developing or creating a marketing plan, marketing experts or decision makers should not rely on this analysis alone. Different options and analysis should be considered. Additionally, insights from customer feedback and results of previous marketing plans should be combined in order to decide the best way to understand and communicate with customers.

Customer Segmentation based on RFM model can be considered as a simple and fast way to get additional information useful to the business decisions.

References

[1] Customer Segmentation. Optimove. (2020).

[2] Jason Brownlee. 10 Clustering Algorithms With Python. Machine Learning Magistery. (2020).

[3] Geeksforgeeks. Clustering in Machine Learning. (2020).

[4] BODOIA, Max. MapReduce Algorithms for k-means Clustering. Stanford education. 2016.

[5] HE, Yaobin, et al. Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce. In: 2011 IEEE 17th International Conference on Parallel and Distributed Systems. IEEE, 2011. p. 473-480.

[6] HE, Yaobin, et al. MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data. Frontiers of Computer Science, 2014, 8.1: 83-99.

[7] BAHMANI, Bahman, et al. Scalable k-means++. arXiv preprint arXiv:1203.6402, 2012.

[8] Online Retail II Dataset from UCI machine learning repository. Kaggle. (2019).

[9] Instacart Market Basket Analysis Dataset. Kaggle. (2017).

[10] eCommerce behaviour data from multi category store. Kaggle. (2020).

[11] Pushpa Makhija. RFM analysis for Customer Segmentation. CleverTap. (2021).

[12] Gavin Wright. RFM analysis (recency, frequency, monetary). TechTarget Network. (2021).

[13] Tern Poh Lim. The Most Important Data Science Tool for Market and Customer Segmentation. Towards Data Science. (2019).

[14] JR Kreiger. Customer segmentation using the Instacart dataset. Towards Data Science. (2020).

[15] Alitouka. Spark_DBSCAN. Available on GitHub. (2018).

[16] Cibotech. Evilplot. Available on GitHub. (2021).