Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster en Ciencia de Datos

Master in Data Science

Trabajo Fin de Máster
Master Thesis

Modelo de Propensión a Compra de Clientes

Customer Propensity to Buy Model

Autor / Author: Ignacio Regaña Muñoz

Madrid, Agosto, 2022

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid.

This Master Thesis has been deposited in ETSI Informáticos de la Universidad Politécnica de Madrid.

Trabajo Fin de Máster
Master Thesis
Máster en Ciencia de Datos
Master in Data Science

Título: Modelo de Propensión a Compra de Clientes

Title: Customer Propensity to Buy Model

Agosto, 2022

Autor / Author: Ignacio Regaña Muñoz

Entry-Supervisor:

Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Exit-supervisor:
Simon Malinowski

Institut de Recherche en Informatique et Systèmes Aléatoires
Université de Rennes 1

Abstract

Customer purchase forecasting aims to predict future customer purchases. The resulting data is of great importance to perform future business activities by designing strategies that match consumer preferences.

To obtain accurate predictions of customer purchases, this paper develops a machine learning framework based on historical data.

Machine learning techniques are applied to predict which customers have a higher propensity to purchase and ranking techniques are applied to prioritize which customers to act on first based on those that provide the highest value, based on RFM (Recency, Frequency, Monetary) analysis.

Due to the high competition and the numerous loyalty plans that exist in the market, there is a need to determine which are the most valuable consumers for the online store and to know how likely they are to make a repeat purchase.

To do this, consumers registered in the loyalty program of an online store are analyzed. The loyalty plan will measure purchase frequency, spending, how current the customer is, as well as different sociodemographic measures.

First, the data is explored to get an idea of what kind of information is available and how it can be used to predict purchases.

Then different machine learning algorithms are tested on the data and their performance is compared. After this, the parameters of the chosen algorithm are adjusted to further improve its performance.

Finally, a dataset is adopted to test the feasibility of the proposed prediction framework. Experimental results and comparative analysis verify the validity of the proposed model.

Keywords: Purchase prediction, segmentation, loyalty programs, machine learning, profitability, customers, hypermarket.

Index

Figures Index

Diagram Index

Table Index

Plot Index

1. Introduction

The propensity to buy from an online store is the likelihood that a consumer will purchase goods or services from a retailer. It is influenced by a wide variety of factors, including the consumer's income, the price of goods and services and their own availability.

There are several reasons why it is important to analyze customer propensity to purchase. First, it can help a company assess the potential demand for its goods and services. In addition, it can be used as a tool to identify potential customers and target marketing resources. Finally, it can help a company assess the potential profitability of its operations. The cost of keeping a customer is 5 to 6 times lower than the cost of adding a new one. Therefore, the economic impact it can have on a large consumer company would be substantial.

RFM Analysis is a data-driven customer behavior segmentation technique. It stands for Recency or Recency, Frequency and Monetary Value.

The idea is to segment customers based on when their last purchase was, how often they have shopped in the past, and how much they have spent in total. These three measures have proven effective in predicting a customer's willingness to engage with marketing messages and offers.

1.1. Context

Today, FMCG companies are using various marketing strategies to attract consumers. Their main objective is to increase sales of their products.

This is because today's market is extremely competitive, and companies are always looking for ways to stand out from the crowd. Because of this, FMCG companies are always looking for ways to build brand loyalty and increase sales. This is useful both to reach new customers and to expand their customer base and gain new information to better adapt to their needs.

The tool used to develop the code for this project was Jupyter Notebook, which offers a simple environment for the user to visualize data and graphics. As for the code, the language used has been Python in its entirety, this provides libraries such as Scikit-learn and XGBoost, which are very useful tools for predictive analytics through Machine Learning and Deep Learning.

1.2. Objectives

The proposed work is intended to identify which customers have a higher propensity to purchase and classify them in terms of profitability to end up sorting them according to the value for the company. In this way, with this information obtained through this study, it will be possible to take measures to reinforce the retention of the best customers of the online store.

2. Methodology

The methodology of the present work is CRISP, (Cross Industry Standard Process). It consists of 6 phases. The arrows in the model indicate the most important and frequent dependencies between the phases. Although the order of these phases is not strict, it is advisable to follow the structure indicated by the figure shown. Each stage of the process is described below:

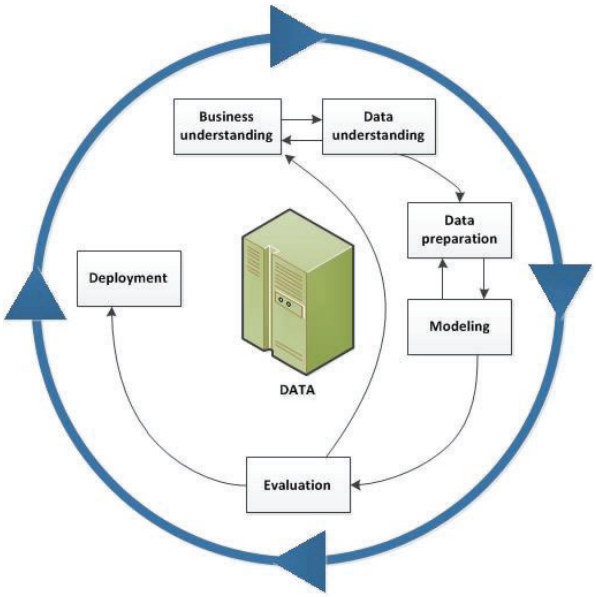


Diagram 1: CRISP diagram.

1. Business Understanding: The first phase focuses on understanding the objectives and work requirements from a business perspective and translates this knowledge into a problem definition to generate a preliminary plan to achieve those objectives.

2. Data Understanding: This phase begins with the initial data collection and processing. The objective is to be familiar with the data, determine its quality and discover unique attributes or an interesting subset.

3. Data Preparation: The data preparation phase involves the entire process of creating the final data set from the original data. This data will be the input value for the model to be generated later. This stage can be executed several times in any order. Tasks include variables, records, attribute selection, and data transformation and cleaning.

4. Modeling: At this point, different modeling technologies can be selected and applied, setting the model parameters to the optimal values.

5. Evaluation: At this stage of the work, we have already created a valid model in terms of data analysis. Before proceeding to the final implementation of the model, it is important to carefully evaluate the model, validate the steps to build the model and ensure that the desired purpose is achieved.

6. Deployment: Modeling consists of finding information in the data, reorganizing it, and displaying it in a user-friendly way. Conclusions and possible implementations are formulated.

3. Modeling Algorithms

There are a variety of different machine learning algorithms that can be used for predictive modeling. Some of the most popular algorithms used in this project are Support Vector Classification, Decision Trees, Random Forest, XGboost, Logistic Regression, Gaussian Naive Bayes and KNeighbours.

Each of these algorithms has its own advantages and disadvantages, there is no best algorithm that works for all data sets. It is important to experiment with different algorithms and adjust the parameters to find the best model for your data.

3.1. Support Vector Regression (SVR)

Support Vector Machines (SVM) are mainly proposed for binary classification problems, and SVR (Support Vector Regression) is an important application branch of SVM, where all the main properties that characterize the algorithm are maintained.

SVR uses the same principles as SVM for classification, however, it has the difference that in SVR the regression has only one sample point at the end, and the optimal hyperplane it seeks is not a maximum distance between two or more types of sample points like SVM but minimizes the deviation of all sample points from the hyperplane. An explanatory figure is shown below:

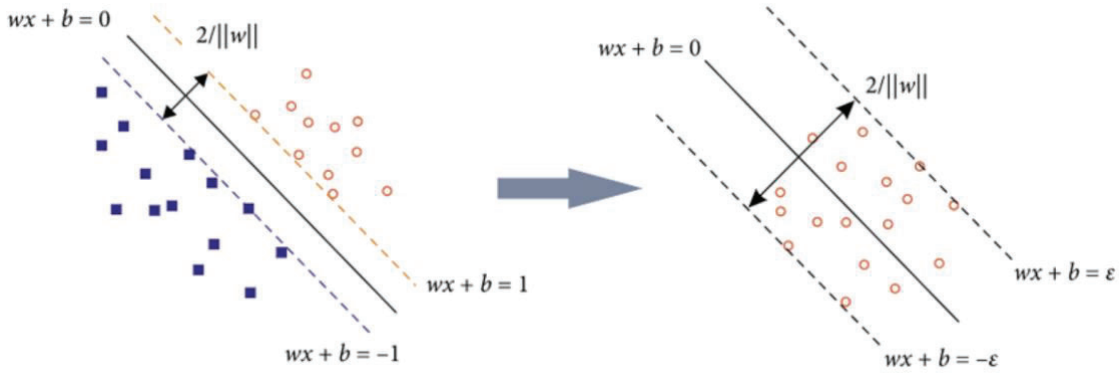


Diagram 2: SVM and SVR differences.

3.2. Decision Trees

To obtain the optimal tree and evaluate each subdivision among all the possible trees and to obtain the root node and the subsequent ones, the algorithm must somehow measure the predictions achieved and evaluate them to compare them all and obtain the best one. To measure and evaluate, it uses various functions, the best known and most used being the Gini index and information gain which uses the so-called entropy (the uncertainty that exists in classifying an experiment or random signal).

By obtaining the entropy measure of each attribute, we can calculate the information gain of the tree. We should maximize that gain.

The division of nodes will continue until we reach the maximum possible depth of the tree or limit the nodes to a minimum number of samples in each leaf.

3.3. Random Forest (RF)

Random Forest is a supervised learning algorithm, which belongs to the Bagging algorithm, and Bagging is an ensemble learning method (which is divided into two methods Boosting and Bagging). The general idea of Ensemble Learning is to train multiple simple models and package them to form a robust model, the performance is much better than that of a single 'simple' model, which has not been trained and compared with multiple variations by modifying its hyperparameters. Decision Trees are assembled in this algorithm.

The Random Forest algorithm can be applied to classification and regression problems by using multiple decision trees to train and predict samples.

To select the optimal segmentation point, the minimum value of the MSE (Mean Squared Error) or MAE (Mean Absolute Error) is used. This point divides the data into two parts and determines the corresponding output value. The predicted value of the random forest is the mean of the predicted values of all trees.

Random Forest is characterized by being simple, easy to implement, low computational overhead and low risk of overfitting.

3.4. Extreme Gradient Boosting (XGBoost)

XGBoost obtains really good predictions without the need of great computational efforts. It is based on an idea similar to Ensemble Learning. It consists of generating multiple sequential simple prediction models, and each new model takes the result of the previous model, making the results obtained in each sequence more and more robust, and, therefore, obtaining more accurate results in each iteration.

During training, the parameters of each weak model are iteratively adjusted trying to find the minimum of an objective function, which can be the classification error ratio, the area under the curve (AUC), the root mean square error (RMSE) or some other.

For the construction of each of these simple models, XGBoost uses decision trees, which are generally used to solve classification and regression problems.

3.5. Logistic Regression

The Logistic Regression method is a statistical method used to solve binary classification problems, where the result can only be dichotomous in nature, that is, it can only take two possible values.

It is a widely used technique due to its effectiveness and simplicity. It does not require large computational resources, both in training and execution.

Logistic regression measures the relationship between the dependent variable, the statement to be predicted, with one or more independent variables, the set of characteristics available to the model. To do so, it uses a logistic function that determines the probability of the dependent variable.

3.6. Gaussian Naive Bayes

It is widely used in NLP (Natural Language Processing) both in the typical example of detecting spam or not and in more complex tasks such as recognizing a language or detecting the appropriate category of a text article. It can also be used for detecting intrusions or anomalies in computer networks and for medical diagnostics given observed symptoms.

Gaussian Naive Bayes is fast, simple to implement, works well with small datasets, does well with many features and can give good results even if it is naive without all the necessary distribution conditions being met in the data.

On the other hand, it requires removing correlated dimensions and for good results the inputs should meet the 2 assumptions of normal distribution and independence from each other (very difficult to do so or we should do transformations on the input data).

3.7. KNeighbours

It is a method that simply looks at the observations closest to the one you are trying to predict and classifies the point of interest based on most of the surrounding data.

It is an instance-based algorithm, meaning that it does not explicitly learn a model (as for example in Logistic Regression or Decision Trees). Instead, it memorizes training instances that are used as a knowledge base for the prediction phase.

It uses the entire dataset to train every point and therefore requires a lot of memory and CPU resources. For these reasons KNeighbours tends to work best on small datasets and without a huge number of features (the columns).

4. Model Evaluation

To objectively measure model performance, metrics such as accuracy and precision are analyzed. This allows us to determine whether the model is useful and what fit is best for the specific problem being explored.

One way to analyze this information is through the confusion matrix. It consists of a matrix representation of the prediction results of any binary test that is often used to describe the performance of the classification model on a set of test data whose true values are known. To understand the metrics analyzed in the confusion matrix, it is important to know some concepts first:

- The Accuracy of the model is basically the total number of correct predictions divided by the total number of predictions.

- The Precision of a class defines how reliable a model is in answering if a point belongs to that class.

- The Recall of a class expresses how well the model can detect that class.

- The F1-Score of a class is given by the harmonic mean of precision and recall.

5. Data Preparation

The quality and quantity of information obtained is very important, as it will have a direct impact on how our model works. We will have to manipulate and convert the data in a way that produces the best results.

The goal is to make the data as clean and usable as possible, and it is important to do it well so that the model can be trained on the most accurate data. It is a critical step in any research project. The data collected will be used to answer research questions, test hypotheses, and build models.

5.1. Data Source

Open data refers to a type of data that has been selected and licensed. This type of information is not restricted by copyright, patent rights and other management mechanisms, and can be open to the public, anyone can publish and use it freely, regardless of whether it is used for publication or other use.

The data used in this project comes from an E-commerce company located in Brazil, it contains information from 2016 to 2018 with 113.425 rows.

These are real commercial data, which were anonymized. Its features allow viewing an order from multiple dimensions: from order status, price, payment and freight performance to customer location, product attributes and, finally, reviews written by customers. However, we will focus on analyzing variables only those variables that allow us to predict customer purchase.

5.2. Definition of Variables

We analyze the data from the dataset which are relevant to the problem in hand. Some variables will be transformed and adapted to the RFM analysis in the information processing section. The types of variables are:

- Continuous variables: These are numerical variables that can take any value within a certain range. Examples of continuous variables are age, height, and weight.

- Categorical variables: These are variables that can take on one of a limited number of values. Examples of categorical variables are gender and nationality.

- Binary variables: Is a type of categorical variable that can take on only two values. The two values must be mutually exclusive and exhaustive, meaning that they must cover all possible values of the variable and there can be no overlap between them. Examples of binary variables are person's gender (male or female) and employment (yes or no).

VARIABLE	DESCRIPTION	VARIABLE TYPE
Order_id	Each order has a unique customer_id.	Categorical
Customer_id	Unique identifier of a customer.	Categorical
Order_date	Shows the purchase timestamp.	Continuous
Quantity	Number of items included in the same order.	Categorical
Product_id	Product unique identifier.	Categorical
Price	Item price.	Continuous
Customer_region	Stores the region of a customer.	Categorical

Table 1: Description of variables.

5.3. Exploratory Data Analysis (EDA)

In statistics, EDA is an approach to analyzing data sets and summarizing key features, often using visualization techniques. Statistical models may or may not be used, but EDA is primarily aimed at seeing what the data are trying to say beyond formal modeling or hypothesis testing tasks.

EDA is used to check the dimensions of the dataframe and possible duplicates. In addition, the variables are described by means of statistics (quartiles, mean, median...) and graphs (bar or column charts, boxplot...). We analyze the distribution and dispersion of continuous variables. Finally, we look for outliers for each variable.

STATISTICS	PRICE
COUNT	112.650
MEAN	120,65
STD	183,63
MIN	0,85
25%	39,9
50%	74,99
75%	134,9
MAX	6.735

Table 3: Basic statistics of price.

STATISTICS	ORDER_DATE
COUNT	113.425
UNIQUE	98.875
TOP	16/7/17 18:19
FREQ	21
FIRST	4/9/16 21:15
LAST	17/10/18 17:30

Table 2: Basic statistics of order_date.

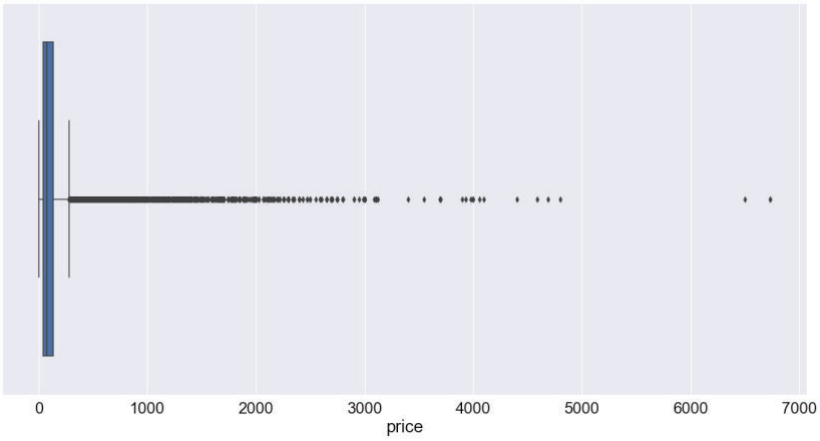
In Table 2 we can see that the variable Price contains most of its values between 39.90 and 134.90. It can be seen that the number of records is lower than the total number of values, which would indicate the presence of null values. It is also interesting to consider that the currency in Brazil is the Brazilian real and is equivalent to 0.19 euros, when studying the value of sales.

On the other hand, Table 3 shows sales records from September 4, 2016, to October 17, 2018. For this project, the time will be disregarded.

STATISTICS	ORDER_ID	CUSTOMER_ID	QUANTITY	PRODUCT_ID	CUSTOMER_REGION
COUNT	113.425	112.650	112.650	112.650	113.425
UNIQUE	99.441	3.095	21	32.951	27
TOP	827…ef	656…c0	1	Aca…af	SP
FREQ	21	2.033	98.666	527	47.820

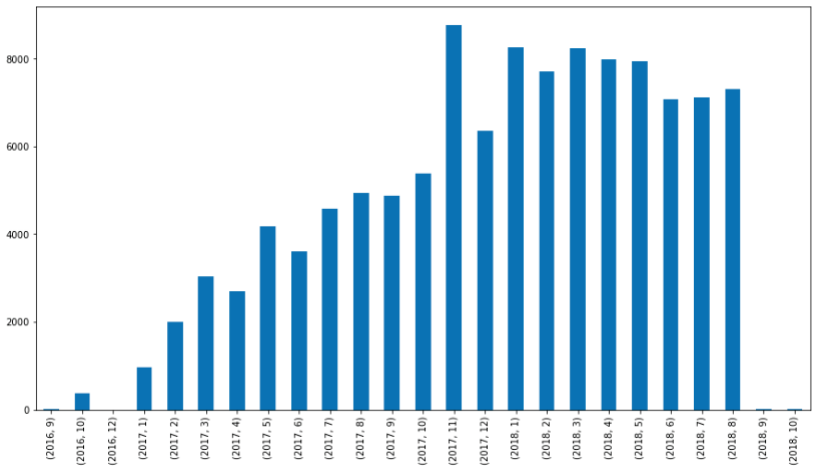
Table 4: Basic statistics of categorical variables.

Through this table we can see the basic statistics of the categorical variables, highlighting the count for customer_id, quantity, and product_id, since they have fewer records of the total. We can observe through the order_id variable that 99.441 sales were made, through customer_id we know that there were 3.095 unique customers, observing the quantity variable we know that in the same order up to 21 items were sent, although the most common was that only one was sent, as for product_id we know that 32.951 unique products were sold, finally, we know that products were sold to 27 different regions of Brazil, however, the region with more sales corresponds to SP, with approximately 42% of total sales.



Plot 1: Boxplot distribution of price.

Through this graph we can observe what was previously mentioned about the distribution of price, most of its records are in values lower than 150.



Plot 2: Barplot distribution of order_date.

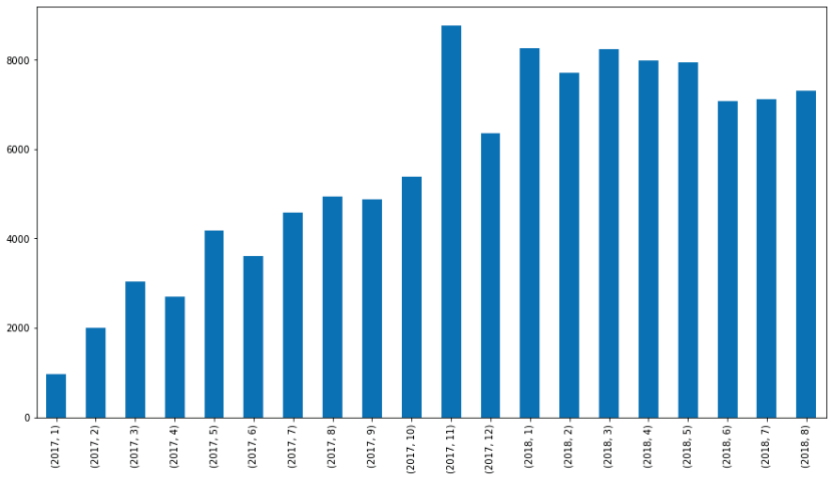
In this graph We can observe that sales were increasing since January 2017 and the highest concentration of sales occurred in November 2017. The latest records for September and October 2018 do not present much data, which may mean that they were not updated in the database.

5.4. Data Cleaning

Through the treatment of missing and outliers we will be able to impute the values of the variables in which they have been detected based on the univariate analysis. Duplicates will also be analyzed.

To do so, the structure of the dataset and the relationship between the variables must be considered.

First, the order_date variable was treated for outliers. Since, as we have been able to verify in the EDA, it contained some records that could confuse the prediction model. Through this adjustment, the values of order_date are bounded between January 2017 and August 2018, as can be seen in the following graph.



Plot 3: Barplot distribution of order_date without outliers.

As we have seen above, the variables customer_id, order_date, product_id and price contain null values.

VARIABLES	NULL VALUES
ORDER_ID	0
CUSTOMER_ID	739
ORDER_DATE	0
QUANTITY	739
PRODUCT_ID	739
PRICE	739
CUSTOMER_REGION	0

Table 5: Null values of each variable.

After analyzing their position in the dataset, it was determined that they all coincided in the same rows. As these were less than 1% of the total number of records, these rows were deleted. After these changes the dataset contained 112.279 records, about 99% of the original data.

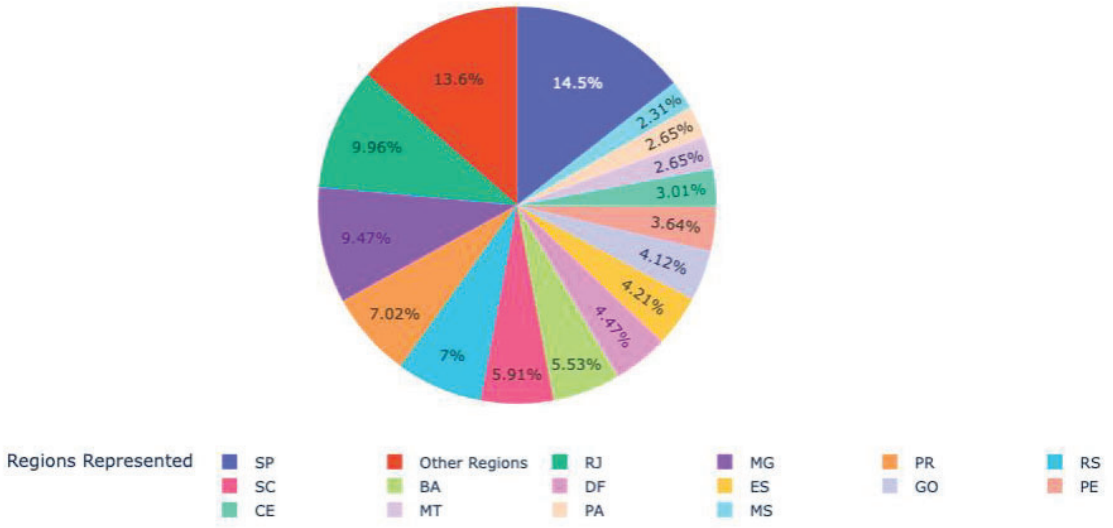
6. Data Transformation

We apply transformations to the data to optimize them and thus achieve greater accuracy in the machine learning model. In addition, the data must meet specific conditions in order to generalize the model.

6.1. Multivariate Analysis

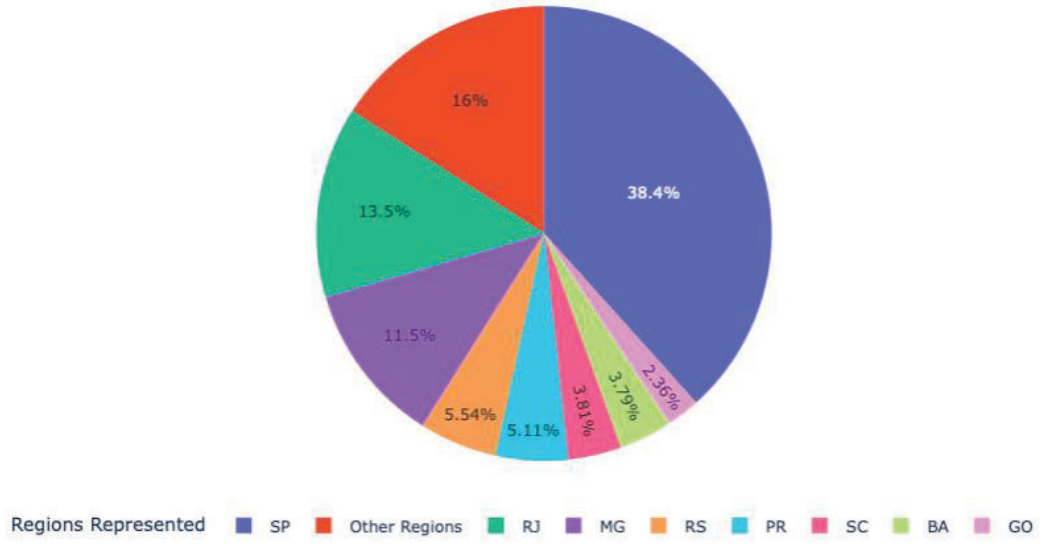
Through multivariate analysis we take different combinations of variables that may be interesting for the project's objective, and we study them with different metrics and generate graphs with them.

First, we wanted to analyze the representation of unique customers by region, in order to have an estimate of the representation of sales depending on the geographic area. In some regions the representation was low (less) than 2.2%, so we proceeded to put them together in a joint category, which indicates through the graph, that many small regions make online purchases.



Plot 4: Pieplot customer count in percentage by region.

Continuing with the analysis by regions, we check the profit generated in each of them. The main regions are placed in the same order, although it stands out that the volume of profits is considerably higher for the main region, this may be since much of the company's advertising or other sales strategies were focused on this particular region. In addition, it should be noted that the representation in terms of profit is much lower for other smaller regions. This can be seen in the graph below:



Plot 5: Pieplot revenue in percentage by region.

6.2. Data Engineering

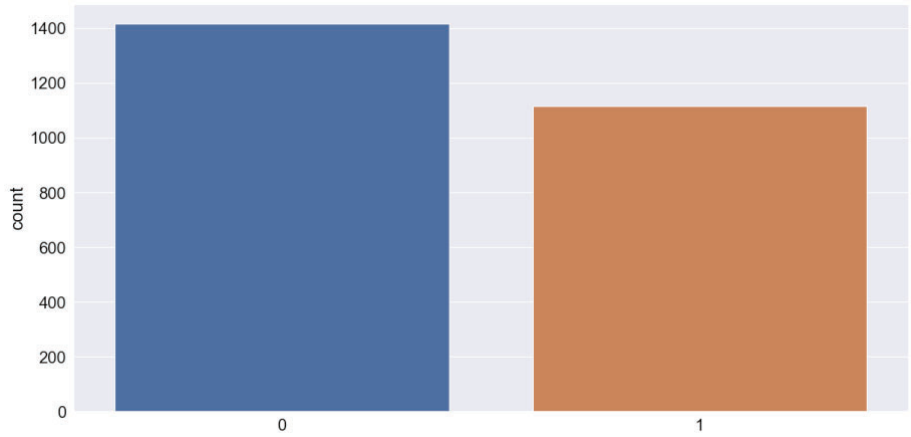
RFM analysis is a data engineering technique that can be used to understand the behavior of customers. It can be used to segment customers based on their past behavior, and to understand how likely they are to respond to future offers. For example, customers who have made a purchase recently, and who spend a lot of money per purchase, are more likely to respond to a future offer than customers who have not made a purchase recently, and who spend very little per purchase.

In this part we focus on performing a series of transformations that allow us to subsequently predict whether a customer will make a purchase after 90 days from their last purchase.

We will start by setting up two different dataframes. The first one will contain data from January 1, 2017, to May 31, 2018, it will serve us to study customer behavior through RFM analysis. On the other hand, we will have a second dataframe with data from June 1, 2018, to August 31, 2018, approximately 90 days, with which we will check if any purchase was made based on the first dataframe.

This allows us to create a variable that indicates in days the time it took a customer to make a purchase based on the last purchase made in the first dataframe and the first purchase of the same customer in the second dataframe. This variable will be used to generate the target variable, which would have a value of 1 if the customers made another purchase in a period of less than 90 days and 0 otherwise. In cases where customers only made one purchase in the first dataframe, and none in the second, they are considered in category 0.

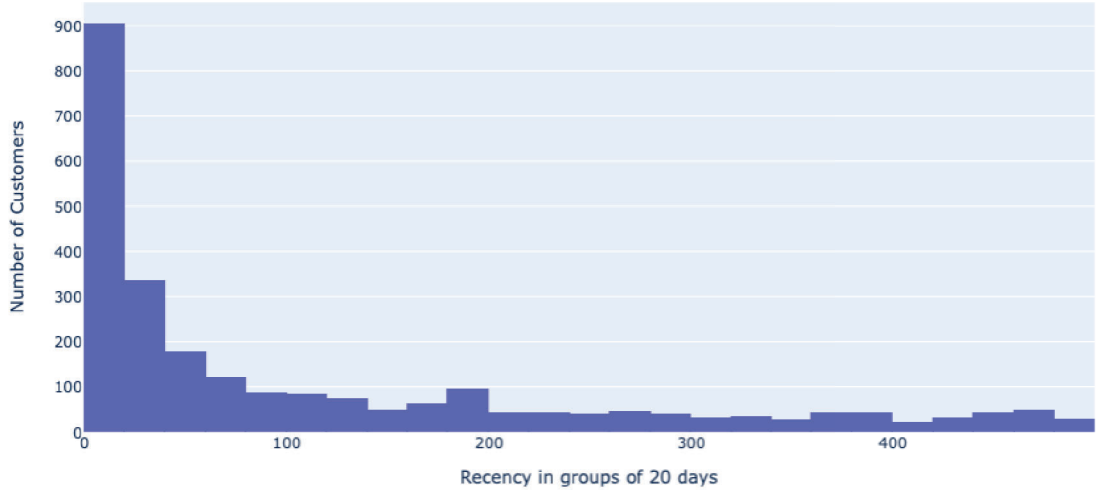
The distribution of the target variable, time_diff_range, is quite balanced, containing around 55% of the cases in category 0 (no purchase in a period longer than 90 days) and in category 1, the rest with 1115 cases, as can be seen in the graph below:



Plot 6: Countplot distribution of time_diff_range.

The most important factor in identifying customers who are most likely to respond to a new offer is timeliness. Customers who have recently made purchases are more likely to purchase new products again than customers who have purchased products in the past.

We calculate the recency as the difference between the first and the last purchase of the first dataframe, in order to find out when customers usually make a second purchase.

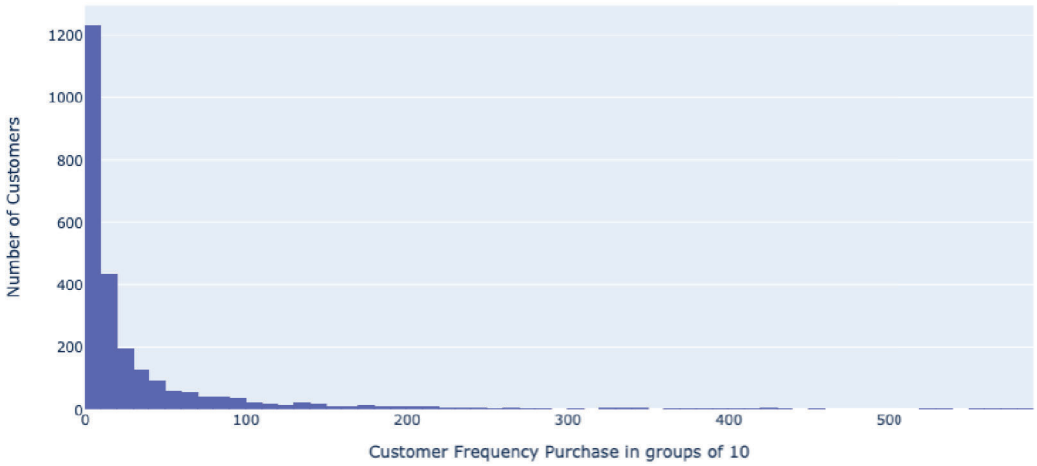


Plot 7: Barplot customer recency in days.

We can see that customers tend to make a second purchase before the first 100 days, specifically, we can highlight that in the first 20 days there is a large volume of repeat customers.

The second most important factor is frequency. Customers who have purchased more products are more likely to respond than those who have purchased fewer products. It indicates how often a customer purchases from a company. The more frequently a customer purchases, the more loyal they are likely to be.

We calculate frequency as the count of times a customer has purchased a product.



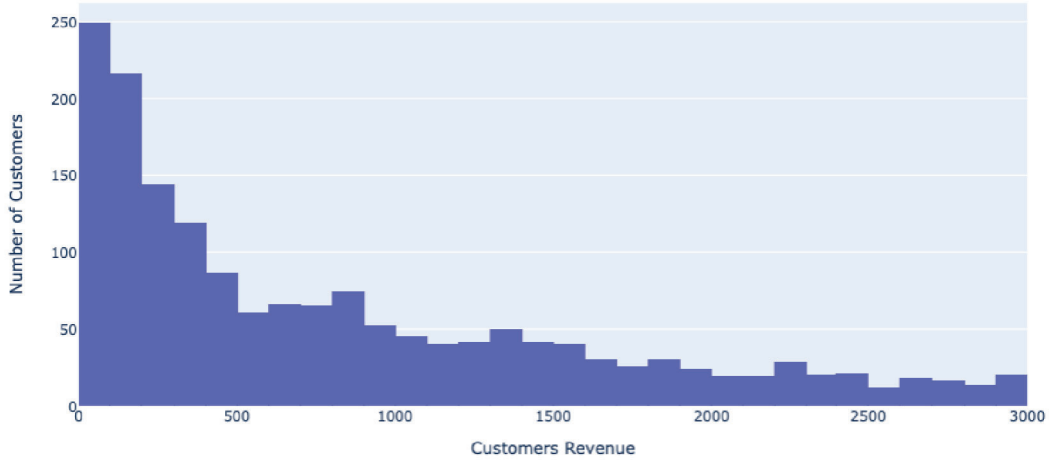
Plot 8: Barplot customers with purchase frequency less than 600.

In this case we can see how most customers tend to buy less than 100 times. However, it is worth noting that many customers (more than 400) buy at least another 10 times.

From customers who buy more than 200 times there is little difference from those who buy more than 500 times, the volume of high-frequency buyers is low.

The third most important factor is the total amount invested, which is referred to as the monetary value. Customers who have invested more (in total across all purchases) in the past are more likely to respond than those who have invested less.

We calculate the monetary value variable by multiplying the number of orders by the price per item.



Plot 9: Barplot customers with monetary value below 3000.

In this graph we can see that most customers tend to spend below a value of 500. It is also worth noting that the slope is less steep than in the graphs seen previously, so it could be studied how many customers would be willing to spend more money on products of this online company.

In order to analyze customer behavior in detail and reach more precise conclusions in the RFM analysis, it is necessary to group consumers.

6.2.1. Kmeans

K-Means is an unsupervised Clustering algorithm. It is used when we have a lot of unlabeled data. The objective of this algorithm is to find K groups (clusters) among the raw data.

The algorithm works iteratively to assign to each point (the rows of our input set form a coordinate) one of the K groups based on their features. They are grouped based on the similarity of their features (the columns). As a result of running the algorithm we will have:

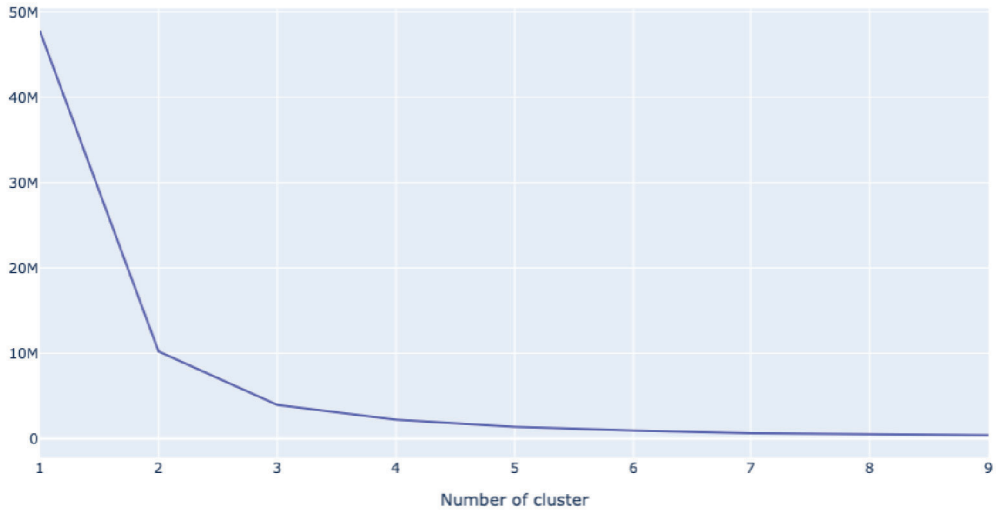
- The centroids of each group that will be a coordinate of each of the K sets that will be used to be able to label new samples.

- Labels for the training data set. Each label belonging to one of the K groups formed.

The groups are defined organically, i.e., their position is adjusted in each iteration of the process, until the algorithm converges.

This algorithm works by pre-selecting a value of K. To find the number of clusters in the data, we should run the algorithm for a range of K values, view the results and compare characteristics of the clusters obtained. In general, there is no exact way to determine the K value, but it can be estimated with acceptable accuracy using the elbow method.

One of the metrics used to compare results is the mean distance between data points and their centroid. Since the value of the mean will decrease as we increase the value of K, we should use the mean distance to the centroid as a function of K and find the elbow point, where the rate of decline sharpens. The graph obtained for this analysis is shown below:



Plot 10: Elbow method.

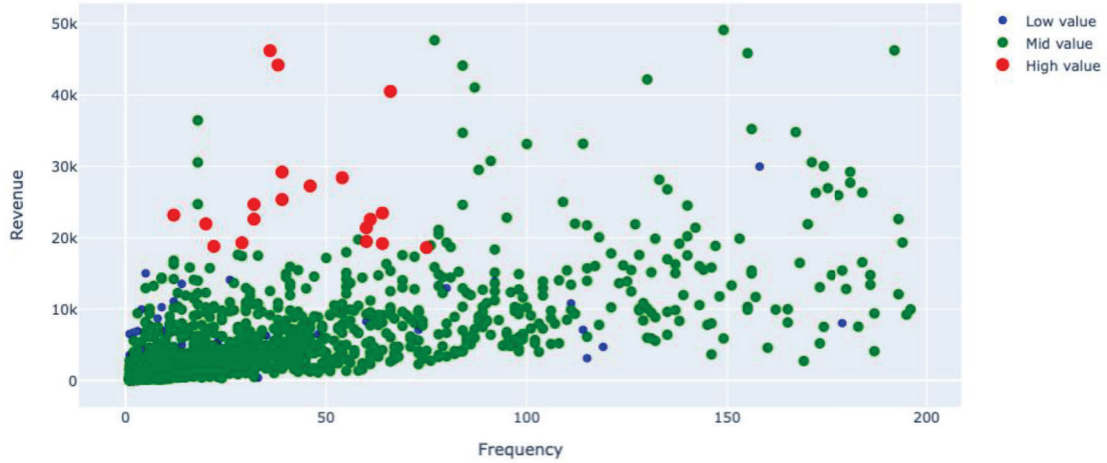
In this case we select 4 clusters for higher classification accuracy. After this, we can indicate the desired number of clusters in the Kmeans algorithm.

We apply the Kmeans algorithm together with the RFM analysis, this will allow us to classify by means of clusters the different groups of customers according to Recency, Frequency and Monetary Value, in order to generate a scoring that will allow us to select the most convenient target for the company.

Score	Recency	Frequency	Monetary
2	392	80	12.918,29
3	426,54	5,15	747,61
4	265,59	28,5	1.978,58
5	100,84	70,59	6.651,96
6	25,01	38,27	6.154,07
7	20,35	106,12	56.086,50

Table 6: Scoring and average RFM values.

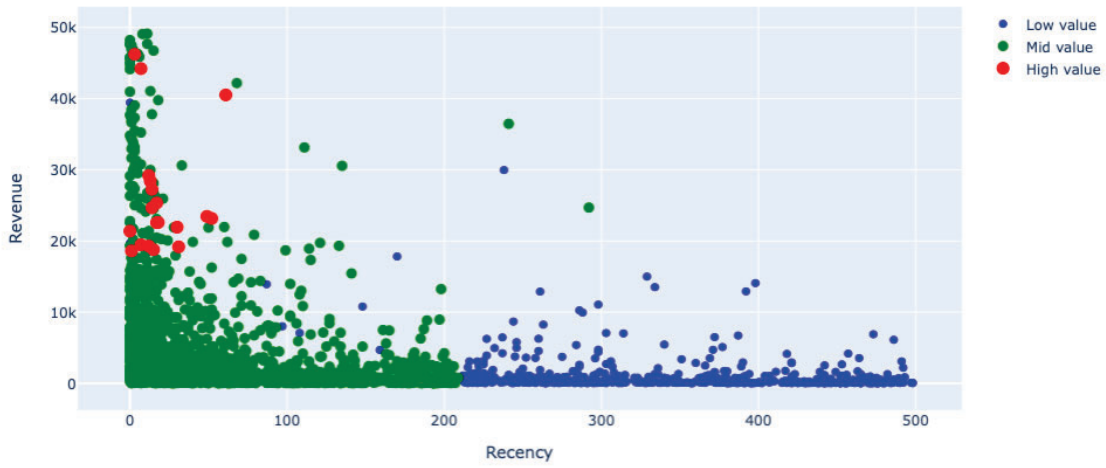
With this information we can conclude that the best target is located with scoring 6 and 7, since the recency is low, so orders will be placed in a short period of time, the frequency is high, so more orders will be placed, and the monetary value is high, so the customer will provide greater benefits to the company. On the other hand, the worst results are found in scoring 2 and 3. After defining these intervals, we proceed to analyze the clustered variables.



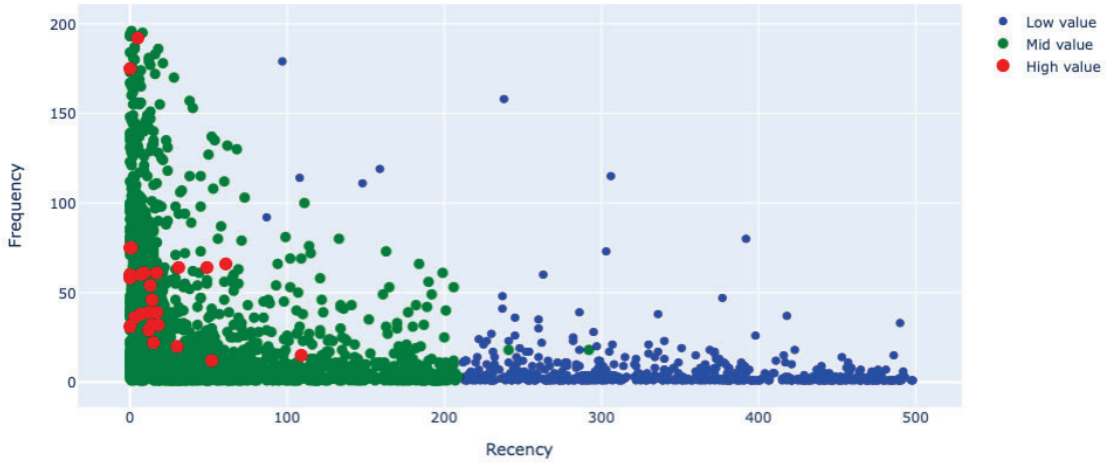
Plot 11: Scatterplot of monetary value and frequency grouped by clusters.

We can graphically observe the results shown in the table above. Customers with monetary value between 20000 and 40000, and with frequency between 10 and 80 are placed as high value customers. It is also worth noting that there is a large majority of medium value customers, this is relevant because depending on the interests of the company they could be classified more accurately. As the frequency increases, so does the dispersion of the points, which could make them more difficult to classify.

For the analysis of the recency variable, the classification of customers is more evident. Below are the graphs combining recency with revenue and frequency:



Plot 12: Scatterplot of monetary value and recency grouped by clusters.

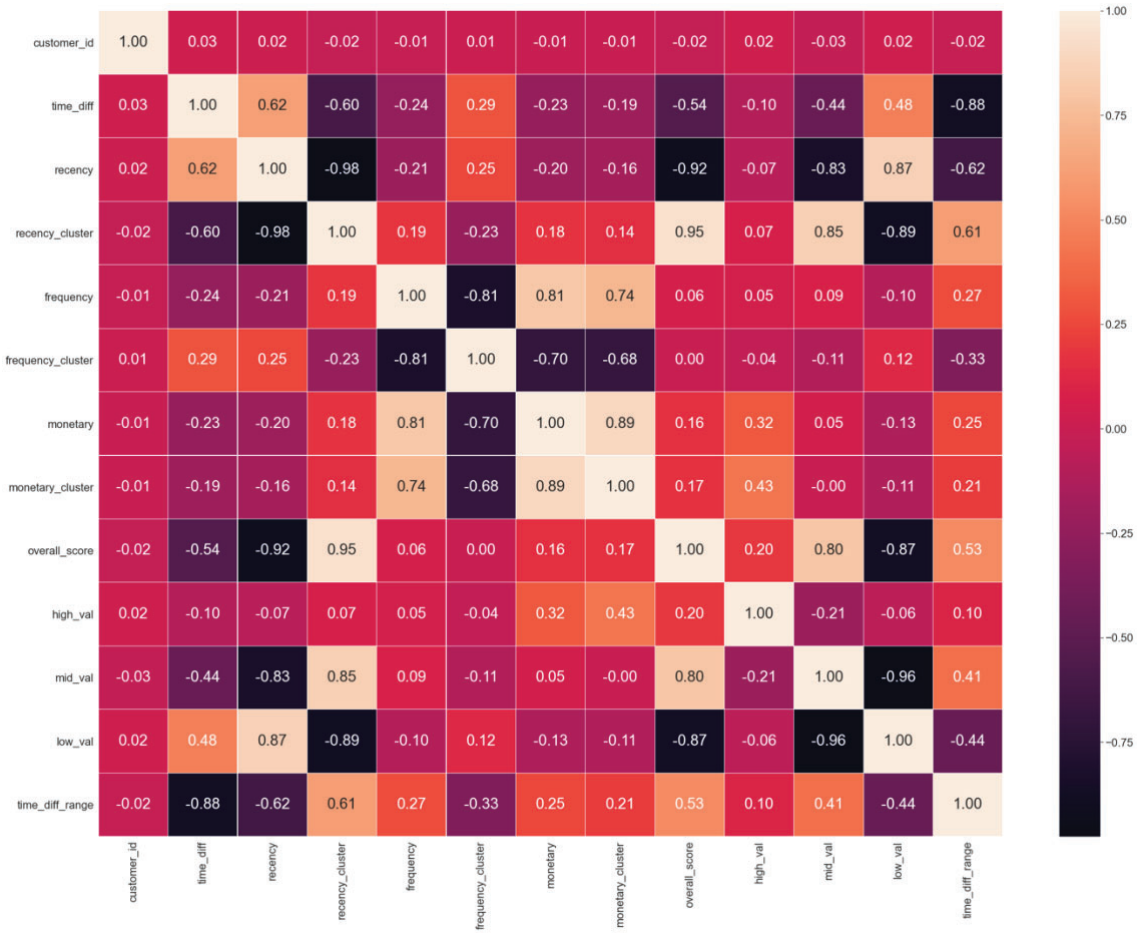


Plot 13: Scatterplot of frequency and recency grouped by clusters.

A clear distinction of medium and low value clients can be observed for the recency variable starting at about 220, this change coincides with the decrease in monetary value and frequency respectively. The higher value class also matches the values shown in the table above. In these two cases there is little dispersion, the points are very close together, which can also make classification difficult, especially in the case of high-value clients.

Finally, we analyze the correlation between all the variables of the dataframe and show the correlation matrix, the darker or lighter a cell is, the more correlation there is, that is, there is a strong correlation between both variables, and it can be a positive or negative correlation.

And the closer it is to 0, the lower the correlation between the pair of variables, i.e., the greater the independence, therefore, there is no linear relationship between the two variables.



Plot 14: Correlation matrix.

As the main objective is to predict customer purchases, the time_diff_range variable is the target of the project, so we will be especially interested in the correlation between this variable and the other variables.

According to the representation of Plot 14, it is observed that the recency_cluster variable (a value of 0.61) and the overall_score variable (a value of 0.53) have a high positive correlation with the target of interest, while the recency (-0.62) and low_val (-0.44) variables have a high negative correlation. The variable time_diff is not considered since it is where the target variable comes from and will be discarded in the generation of the machine learning model.

7. Creation of the Model

Machine learning model training refers to the process of defining the parameters and hyperparameters of an algorithm so that it fits the data well.

In the previous section, we conducted a more detailed exploration of consumer trends. Therefore, the next step is to model and evaluate the buying trends using different algorithms to achieve the purpose of prediction.

7.1. Train/Test Selection

To train and evaluate the machine learning algorithms, the sample is divided into train and test sets. It is important that the splitting is random, to ensure that both data sets come from the same distribution.

The train set is the data we use to train a model. The quality of our machine learning model will be directly proportional to the quality of the data.

The test set is the data we reserve to check if the model we have generated from the train set works. That is, if the answers predicted by the model for a totally new case are correct or not.

The train and test sets contain 80% and 20% of the data, respectively and has the variables customer_id, recency, recency_cluster, frequency, frequency_cluster, monetary, monetary_cluster, overall_score, high_val, mid_val, low_val and the target variable, time_diff_range.

7.2. Training the Model

For model training we make use of cross-validation. It is a technique with which we can prevent the existence of different problems such as the appearance of overfitting. This allows more stable models to be obtained.

In cross-validation the training data set is divided into groups of equal size. Once the partitioning is done, the model is trained for each of the groups except the test group to validate the results. An important difference with the out-of-sample validation is that this time the model is trained and validated with all the data, changing the set used for validation in each iteration.

Kfold cross-validation divides the data set into K groups of k folds each.

This process is shown schematically in the following figure. In it, a data set has been represented, which has been divided into five, although in the project the partition has been of two splits.

	Dataset				
	Training set				Test set
Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

Diagram 3: Cross validation algorithm with Kfold.

It uses the first split in the first iteration to test the model. It uses the remaining data sets to train the model. The second fold helps to test the data set and the remaining data sets are used for the training process. This procedure is repeated until the test set uses all the folds in all five splits.

We take advantage of this process to combine it with the Multiscorer function, which will provide us with a quick and easy visualization of the statistics, being able to adjust some parameters.

We set the Multiscorer function with the parameter average, this will define the way in which the metrics are calculated. In this case we set it to be with macro-average. This will calculate the metric independently for each class and then perform the average (therefore, it will treat all classes equally).

Micro-average is preferable if it is suspected that there may be an imbalance of classes, as it will aggregate the weights of all classes to calculate the average metric.

Multiscorer also allows us to calculate multiple metrics for the same classifier without retraining it. This allows us to obtain any number of metrics in cross validation. As we can see in the table below:

model_name	*accuracy*	*f1_score*	*recall*	*precision*	*time*
RandomForestClassifier	0,8731	0,8725	0,8775	0,8726	0,3525
LogisticRegression	0,8728	0,8720	0,8765	0,8727	0,0562
GaussianNB	0,8694	0,8689	0,8748	0,8703	0,0173
xgb.XGBClassifier	0,8431	0,8390	0,8413	0,8446	0,1513
DecisionTreeClassifier	0,8388	0,8341	0,8368	0,8410	0,0176
SVC	0,8354	0,8300	0,8332	0,8388	0,0814
KNeighborsClassifier	0,8294	0,8256	0,8281	0,8305	0,0791

Table 7: Metrics of the different models.

The data shown indicate that we have a good fit of the model, since, in general terms, both recall and precision have high values.

In terms of accuracy, we can see that Random Forest and Logistic Regression would be the best models to predict customer purchase.

In this model evaluation we have included the variable time, which indicates the execution time from the time the model is created until the model results are stored. It can be observed that the processing time is slightly higher in Random Forest and in XGBoosting. In larger projects this would be a particularly important variable.

7.3. Improvement of the Model

During the training we obtained good predictions, however our accuracy is not the minimum desired so we will test a new parameter setting of our model. We can increase the number of times we iterate our training data (EPOCHs).

Another important parameter is known as Learning Rate. It is not the same to increase our values by 0.1 units than by 0.001 units, this can significantly affect the model execution time. We can also indicate the maximum allowed error of our model.

We can go from taking a few minutes to hours to train our machine. These parameters are known as hyperparameters. There are usually many parameters to be adjusted and by combining them all our options can be triggered. Each algorithm has its own parameters to adjust. In this case, we will use the GridSearch algorithm together with the XGBoost classification algorithm to find the best hyperparameters for the model.

The GridSearch algorithm is an effective method for tuning parameters in supervised learning and improving the generalization performance of a model. With GridSearch we test all possible combinations of the parameters of interest, in this case we study:

- min_child_weight: this is the minimum number of samples that a node can represent to be able to keep splitting. If there are less min_child_weight samples in that node, the node becomes a leaf and no longer splits. This can help reduce the complexity of the model and avoid overfitting. We tested with values from 1 to 10 in jumps of 2.

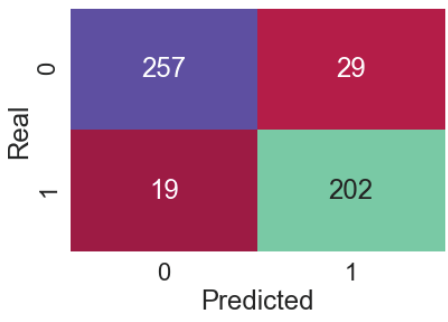
- max_depth: This is the depth of each decision tree formed by the XGBoost algorithm, i.e., the number of levels generated per tree. Each new branch implies an increase in depth. It is usually a very low value, thus making that each tree can only learn a small part of the relationship between predictors and response variable. We tested with values from 3 to 15 in jumps of 3.

Finally, the best result provided by the GridSearch algorithm suggested values of 9 and 3 for min_child_weight and max_depth respectively.

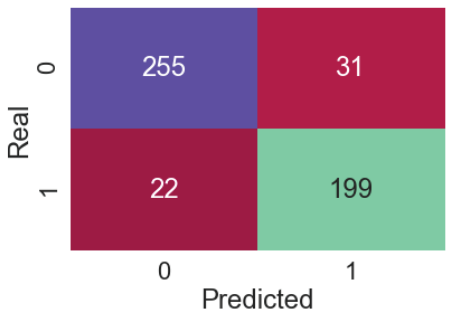
8. Select the Best Model

After obtaining an improved XGBoost model, we compared its performance with that of the best performing unmodified model, Random Forest.

Using confusion matrices, we can analyze the results of how a supervised learning algorithm works. The confusion matrices obtained are shown below:



Plot 15: XGBoost confusion matrix.



Plot 16: Random Forest confusion matrix.

model_name	*accuracy*	*f1_score*	*recall*	*precision*
xgb.XGBClassifier	0,9053	0,8938	0,9140	0,8745
RandomForestClassifier	0,8955	0,8825	0,9005	0,8652

Table 8: Metrics of the improved models.

We can see that the number of values predicted by both models is very similar, although slightly higher in XGBoost. These data indicate that the model is capable of detecting nearly all of the values present in a sample with a 90% of accuracy.

The presence of more false negatives than false positives mean that the model is more efficient at detecting the absence of a value than its presence. This is due to the fact that the model was trained with a more balanced dataset, meaning that it learned to be more sensitive to the values that were absent.

9. Conclusions and Future Applications

As mentioned in the introduction and in the objectives, this work is not only a prediction of the propensity to buy of the customers of the analyzed company, but also a study of the positioning of the customer groups and to know the different departments in relation to the customer groups in order to better understand their business situation. This study will provide companies with an analysis of the information that will enable them to develop a customer acquisition and retention strategy.

As we have seen, the study of customer propensity to buy allows a company to manage its resources correctly and efficiently, with the aim of designing strategies for the customers it wishes to keep and learn even more from them.

As for the different algorithms that have been studied, it should be said that although the differences between them are minimal, any of them being valid enough to carry out our purpose. The performance of the XGBoost model stands out, with which we have tried to improve the predictions with the GridSearch method, a technique that improves quantitatively any algorithm on its own.

In this study it has been shown that RFM analysis has the ability to provide business information to connect with your customers, and thus create an individual response that makes them feel special because you know what they need and because you solve their needs.

From the analysis of the data, different customer segments or groups can be created to reach each one with a personalized message and offer that is relevant to them.

Having information on customer behavior and other customer data makes it possible to design different offers that are better adapted to what they need.

A possible extension of this work would be to have more consumer variables, customer relationship variables, and customer satisfaction variables, in order to study the effect of the customer-company relationship. It would be interesting to combine this study with one on customer abandonment in order to find those factors that make the customer want to buy again or choose another company for their purchases.

10. Bibliography

- Villaécija, Raquel. «¿Somos fieles a nuestro "súper"? | Economia Home | EL MUNDO», 9 de febrero de 2017. https://www.elmundo.es/economia/2017/02/09/5898bbede5fdeab2538b4682.html.

- Gil, Daniel Álvarez. «Metodología CRISP-DM - Adictos al trabajo Tutoriales». *Adictos al trabajo* (blog), 14 de enero de 2021. https://www.adictosaltrabajo.com/2021/01/14/metodologia-crisp-dm/.

- Amat Rodrigo, Joaquín. «Máquinas de Vector Soporte (Support Vector Machines, SVMs)». Accedido 10 de junio de 2022. https://www.cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines.

- Kumar, Aditya. «Random Forest for Prediction». Medium, 22 de junio de 2020. https://towardsdatascience.com/random-forest-ca80e56224c1.

- Vega, Juan Bosco Mendoza. «XGBoost En R». *Medium* (blog), 8 de diciembre de 2019. https://medium.com/@jboscomendoza/xgboost-en-r-398e7c84998e.

- Rodríguez, Daniel. «La regresión logística». Analytics Lane, 23 de julio de 2018. https://www.analyticslane.com/2018/07/23/la-regresion-logistica/.

- Hojas, Ignacio Moreno. «Evaluación del modelo de clasificación». *StatDeveloper* (blog), 17 de enero de 2020. https://www.statdeveloper.com/evaluacion-del-modelo-de-clasificacion/.

- Brazilian E-Commerce Public Dataset by Olist. https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce. Accessed 21 Aug. 2022.

- ¿Comprar Casa o Alquilar? Naive Bayes Usando Python | Aprende Machine Learning. https://www.aprendemachinelearning.com/comprar-casa-o-alquilar-naive-bayes-usando-python/.

- Crea Un Arbol de Decisión En Python | Aprende Machine Learning. https://www.aprendemachinelearning.com/arbol-de-decision-en-python-clasificacion-y-prediccion/.

- Filvà, Daniel Amo. Entropía en la recolección de datos – Edulíticas | Analítica del Aprendizaje – Learning Analytics. https://eduliticas.com/2017/09/divulgacion/entropia-en-la-recoleccion-de-datos/.

- 'Guía de análisis RFM 2021 - Ejemplos de segmentación predictiva'. Barilliance, 18 Aug. 2021, https://www.barilliance.com/es/guia-de-analisis-por-rfm-6-segmentos-clave-para-el-rfm-basado-en-marketing/.

- Karaman, Barış. 'Predicting Next Purchase Day'. Medium, 15 Sept. 2019, https://towardsdatascience.com/predicting-next-purchase-day-15fae5548027.

- Leung, Kenneth. 'Micro, Macro & Weighted Averages of F1 Score, Clearly Explained'. Medium, 20 June 2022, https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f.

- Stylianopoulos, Kyriakos. Multiscorer. 2017. 14 Aug. 2022. GitHub, https://github.com/StKyr/multiscorer.

- 'Elbow Method for Optimal Value of k in KMeans'. *GeeksforGeeks*, 6 June 2019, https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/.

- *K-Fold Cross-Validation in Python Using SKLearn - AskPython*. 12 Nov. 2020, https://www.askpython.com/python/examples/k-fold-cross-validation.

- rédac, Team. 'Cross-Validation : definición e importancia en Machine Learning'. *Formation Data Science | DataScientest.com*, 13 May 2022, https://datascientest.com/es/cross-validation-definicion-e-importancia.

11. Code

```
# # Imports and functions


# importing necessary Python libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
import plotly.express as px
import plotly.graph_objs as go


#import machine learning related libraries
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, cross_val_score, train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from multiscorer import MultiScorer
import xgboost as xgb
```

```
# Return the given dataset with the input column and its corresponding cluster column sorted in
ascending or descending order.

def sort_cluster(data, column, cluster_column, ascending):

    # Rename cluster_column adding "tmp"
    new_cluster_column = "tmp_" + cluster_column

    # New dataframe by grouping data by cluster_column, getting the mean of column
    data_new = data.groupby(cluster_column)[column].mean().reset_index()

    # Order data_new depending on the ascending variable
    data_new = data_new.sort_values(by = column, ascending = ascending).reset_index(drop = True)

    # Add the index to data_new as a new column
    data_new["index"] = data_new.index

    # Merge data with cluster_column and index in data_new by cluster_column
    data_final = pd.merge(data, data_new[[cluster_column, "index"]], on = cluster_column)

    # Drop cluster_column and rename index with its name
    data_final = data_final.drop([cluster_column], axis = 1)

    data_final = data_final.rename(columns = {"index": cluster_column})

    return data_final


# Print a **confusion matrix** with the real and the predicted values.

def plot_conf_mtx(y_test, y_pred):

    data = {'y_real': y_test, 'y_predicted': y_pred}

    df = pd.DataFrame(data, columns = ['y_real','y_predicted'])
    conf_mtx = pd.crosstab(df['y_real'], df['y_predicted'],
                           rownames = ['Real'],
                           colnames = ['Predicted'])
```

```
    sns.heatmap(conf_mtx, annot = True, fmt = "d", cmap = "Spectral", cbar=False)
    plt.show()


# # Load data

df_customers = pd.read_csv('olist_customers_dataset.csv')
df_item = pd.read_csv('olist_order_items_dataset.csv')
df_orders = pd.read_csv('olist_orders_dataset.csv')


df321 = df_orders.merge(df_item, on='order_id', how='left')
df32 = df321.merge(df_customers, on='customer_id', how='outer')


display(df32)


df = df32[['order_id', 'seller_id', 'order_purchase_timestamp', 'order_item_id', 'product_id', 'price',
'customer_state']]


display(df)


# # EDA

df.rename(columns={'order_purchase_timestamp':'order_date', 'seller_id':'customer_id',
                   'order_item_id':'quantity', 'customer_state':'customer_region'}, inplace=True)
df.head()


df.describe()


plt.figure(figsize=(15,8))
sns.countplot(df['order_id'])


plt.figure(figsize=(15,8))
```

```
sns.countplot(df['customer_id'])


df['order_date'] = pd.to_datetime(df['order_date'])
plt.figure(figsize=(15,8))
df['order_date'].groupby([df['order_date'].dt.year, df['order_date'].dt.month]).count().plot(kind="bar")


plt.figure(figsize=(15,8))
sns.countplot(df['quantity'])


plt.figure(figsize=(15,8))
sns.countplot(df['product_id'])


plt.figure(figsize=(15,8))
sns.distplot(df['price'])


plt.figure(figsize=(15,8))
sns.boxplot(df['price'])


plt.figure(figsize=(15,8))
sns.countplot(df['customer_region'])


# # Missing values and outliers treatment

df.info()

df.isnull().sum()

# Delete outliers of date
df['order_date'] = pd.to_datetime(df['order_date'])

df = df[(df['order_date'] < pd.Timestamp(2018,9,1)) &
    (df['order_date'] >= pd.Timestamp(2017,1,1))].reset_index(drop=True)
```

```
df['order_date'].describe()


plt.figure(figsize=(15,8))
df['order_date'].groupby([df['order_date'].dt.year, df['order_date'].dt.month]).count().plot(kind="bar")


# 0.65% of the values for customer_id, quantity, product_id and price are missing
(739/113018)*100

# Delete the missing values
df.dropna(axis=0, inplace=True)


df.isnull().sum()


df.head()


# # Multivariant analysis

# Customer representation by region

# Get the percentage of customers grouping by region

ctm_rgn_df = df.groupby(['customer_id', 'customer_region']).count().reset_index()

ctm_rgn_df = ctm_rgn_df.groupby('customer_region')['customer_id'].count().reset_index().sort_values(
    by=['customer_id'], ascending=False)

# Calculate the customer representation in percentage
ctm_rgn_df['pct_customer']= np.round(ctm_rgn_df['customer_id'] / ctm_rgn_df['customer_id'].sum() * 100,
2)

ctm_rgn_df.head()


# Group countries with customer percentage value less than 2.2
```

```
percent_margin = 2.2

ctm_rgn_df['region_category'] = ctm_rgn_df['customer_region']

# region_category contains the values of customer_region with the countries that represent
## less than 2.2% as Other Regions
ctm_rgn_df.loc[ctm_rgn_df.pct_customer <= percent_margin, 'region_category'] = 'Other Regions'

ctm_rgn_df.head()


# plot pie chart

pie_fig = px.pie(ctm_rgn_df,
                 names="region_category",
                 values="pct_customer",
                 title="Customer Count in Percentage by Region"
                )

pie_fig.update_layout(title_x=0,
                      legend_title="Regions Represented",
                      legend=dict(orientation="h")
                     )

pie_fig.show(config={'displaylogo': False})


# Revenue made each month

# Change date format to year-month

df['order_date'] = pd.to_datetime(df['order_date'])

df['order_date_ym'] = df['order_date'].map(lambda date: 100*date.year + date.month)
df.head()


# Get the variable monetary

df['monetary'] = df['price'] * df['quantity']
```

```
df.head()


# Group by year and month getting the sum of the revenue

ctm_monetary = df.groupby('order_date_ym')['monetary'].sum().reset_index()
ctm_monetary.head()


pd.DataFrame(ctm_monetary['monetary'].describe())


# line plot

line_fig = px.line(ctm_monetary,
                   x = "order_date_ym",
                   y = "monetary",
                   title = "Monthly revenue from Jan. 2017 to Aug. 2018"
                  )

line_fig.update_layout(title_x=0.5,
                       showlegend=False,
                       xaxis={"type": "category"},
                       xaxis_title="Year-Month",
                       yaxis_title="Monthly Revenue"
                      )

line_fig.show(config={'displaylogo': False})


# Percentage of revenue based on customer regions

# Get the percentage of revenue grouping by region

region_monetary_df =
df.groupby(['customer_region']).monetary.sum().reset_index().sort_values(by=['monetary'],
                                                                         ascending=False)

region_monetary_df['pct_monetary_region'] = np.round(region_monetary_df.monetary /
region_monetary_df.monetary.sum() * 100, 2)
```

```
region_monetary_df.head()


# Group regions with a representation smaller than the 2.2%

percent_margin = 2.2

region_monetary_df['region_category'] = region_monetary_df['customer_region']

region_monetary_df.loc[region_monetary_df.pct_monetary_region <= percent_margin, 'region_category']
= 'Other Regions'

region_monetary_df.head()


# plot pie chart
pie_fig = px.pie(region_monetary_df,
                 names = "region_category",
                 values = "pct_monetary_region",
                 title = "Region Revenue in Percentage"
                )

pie_fig.update_layout(title_x = 0,
                      legend_title = "Regions Represented",
                      legend = dict(orientation = "h")
                     )

pie_fig.show(config = {'displaylogo': False})


#
# # Data Engineering

# Substitute numeric ID based on the original

df['customer_id'] = df.customer_id.astype('category').cat.rename_categories(range(1,
df.customer_id.nunique() + 1))

df['customer_id'] = df['customer_id'].astype('int')
```

```
df.head()


# Split the dates in rng_date and eval_rng_date

rng_date = df[(df['order_date'] < pd.Timestamp(2018,6,1)) &
     (df['order_date'] >= pd.Timestamp(2017,1,1))].reset_index(drop=True)


eval_rng_date = df[(df['order_date'] < pd.Timestamp(2018,9,1)) &
     (df['order_date'] >= pd.Timestamp(2018,6,1))].reset_index(drop=True)


# Get the distinct customers in the dataframe

ctm_df = pd.DataFrame(rng_date['customer_id'].unique())

ctm_df.columns = ['customer_id']

ctm_df.head()


# Create a dataframe with customers first purchase date in eval_rng_date

eval_rng_1st_date = eval_rng_date.groupby('customer_id')['order_date'].min().reset_index()
eval_rng_1st_date.columns = ['customer_id','1st_prchs_date']
eval_rng_1st_date.head()


# Create a dataframe with customers last purchase date in rng_date
eval_rng_last_date = rng_date.groupby('customer_id')['order_date'].max().reset_index()
eval_rng_last_date.columns = ['customer_id','last_prchs_date']
eval_rng_last_date.head()


# Merge the two dataframes

rng_purchase_dates = pd.merge(eval_rng_last_date, eval_rng_1st_date, on = 'customer_id', how = 'left')
```

```
rng_purchase_dates.head()


# Time difference in days between last and first purchases

rng_purchase_dates['time_diff'] = (rng_purchase_dates['1st_prchs_date'] -
rng_purchase_dates['last_prchs_date']).dt.days

rng_purchase_dates.head()


# merge with ctm_df

ctm_df = pd.merge(ctm_df, rng_purchase_dates[['customer_id','time_diff']], on='customer_id', how='left')
ctm_df.head()


# Fill missing values with 9999

ctm_df = ctm_df.fillna(9999)
ctm_df.head()


# RFM: Recency
# Activity of the customers based on the customers last purchase

eval_rng_last_date = rng_date.groupby('customer_id').order_date.max().reset_index()
eval_rng_last_date.columns = ['customer_id','last_prchs_date']
eval_rng_last_date.head()


# Get the recency in days
eval_rng_last_date['recency'] = (eval_rng_last_date['last_prchs_date'].max() -
eval_rng_last_date['last_prchs_date']).dt.days

ctm_df = pd.merge(ctm_df, eval_rng_last_date[['customer_id', 'recency']], on = 'customer_id')
ctm_df.head()


pd.DataFrame(ctm_df.recency.describe())
```

```
# plot histogram
hist_fig = px.histogram(ctm_df,
                        x = "recency",
                        title = "Customers Recency in Days"
                       )

hist_fig.update_layout(title_x = 0.5,
                       xaxis_title = "Recency in groups of 20 days",
                       yaxis_title = "Number of Customers"
                      )

hist_fig.show(config = {'displaylogo': False})


# Elbow Method to determine how many clusters are needed in order to apply K-means

tmp_dct = {}
ctm_recency = ctm_df[['recency']]
for idx in range(1, 10):
    kmeans = KMeans(n_clusters = idx, max_iter = 1000).fit(ctm_recency)
    ctm_recency["clusters"] = kmeans.labels_
    tmp_dct[idx] = kmeans.inertia_

line_fig = px.line(x = list(tmp_dct.keys()),
                   y = list(tmp_dct.values())
                  )

line_fig.update_layout(title_x = 0,
                       xaxis_title = "Number of cluster",
                       yaxis_title = ""
                      )

line_fig.show(config = {'displaylogo': False})


# Generate 4 clusters

nb_clusters = 4
```

```
kmeans = KMeans(n_clusters=nb_clusters)
kmeans.fit(ctm_df[['recency']])
ctm_df['recency_cluster'] = kmeans.predict(ctm_df[['recency']])
ctm_df.head()


ctm_df = sort_cluster(ctm_df, 'recency', 'recency_cluster', False)
ctm_df.head()


ctm_df.groupby('recency_cluster')['recency'].describe()


# RFM: Frequency
# Total number of orders by each customer on time

ctm_frequency = df.groupby('customer_id').order_date.count().reset_index()
ctm_frequency.columns = ['customer_id','frequency']


ctm_df = pd.merge(ctm_df, ctm_frequency, on = 'customer_id')

ctm_df.head()


pd.DataFrame(ctm_df.frequency.describe())


# plot histogram
hist_fig = px.histogram(x=ctm_df.query('frequency < 600')['frequency'],
                        title="Customers with Purchase Frequency less than 600"
                       )

hist_fig.update_layout(title_x=0.5,
                       xaxis_title="Customer Frequency Purchase in groups of 10",
                       yaxis_title="Number of Customers"
                      )

hist_fig.show(config={'displaylogo': False})
```

```
kmeans = KMeans(n_clusters=nb_clusters)
kmeans.fit(ctm_df[['frequency']])
ctm_df['frequency_cluster'] = kmeans.predict(ctm_df[['frequency']])


ctm_df = sort_cluster(ctm_df, 'frequency', 'frequency_cluster', False)
ctm_df.head()


#see details of each cluster
ctm_df.groupby('frequency_cluster')['frequency'].describe()


ctm_df.head()


# RFM: Monetary
# Customers spent money

ctm_monetary = df.groupby('customer_id').monetary.sum().reset_index()


#merge it with our ctm_dt
ctm_df = pd.merge(ctm_df, ctm_monetary, on = 'customer_id')
ctm_df.head()


# plot histogram
hist_fig = px.histogram(x = ctm_df.query('monetary < 3000')['monetary'],
                        title = "Customers with Monetary Value below 3000"
                       )

hist_fig.update_layout(title_x = 0.5,
                       xaxis_title = "Customers Revenue",
                       yaxis_title = "Number of Customers"
                      )

hist_fig.show(config = {'displaylogo': False})
```

```
#apply clustering
kmeans = KMeans(n_clusters=nb_clusters)
kmeans.fit(ctm_df[['monetary']])
ctm_df['monetary_cluster'] = kmeans.predict(ctm_df[['monetary']])


#order the cluster numbers
ctm_df = sort_cluster(ctm_df, 'monetary', 'monetary_cluster', True)
ctm_df.head()


ctm_df.groupby('monetary_cluster')['monetary'].describe()


#calculate overall score and use mean() to see details
ctm_df['overall_score'] = ctm_df['recency_cluster'] + ctm_df['frequency_cluster'] +
ctm_df['monetary_cluster']
ctm_df.groupby('overall_score')['recency','frequency','monetary'].mean()


ctm_df['segment'] = 'low_val'
ctm_df.loc[ctm_df['overall_score'] > 4, 'segment'] = 'mid_val'
ctm_df.loc[ctm_df['overall_score'] > 6, 'segment'] = 'high_val'


ctm_df.head()


# scatter plot monetary vs frequency

ctm_graph = ctm_df.query("monetary < 50000 and frequency < 200")


plot_data = [
    go.Scatter(
        x=ctm_graph.query("segment == 'low_val'")['frequency'],
        y=ctm_graph.query("segment == 'low_val'")['monetary'],
        mode='markers',
        name='Low value',
```

```
        marker=dict(
            size=7,
            color='blue'
           )
    ),

    go.Scatter(
        x=ctm_graph.query("segment == 'mid_val'")['frequency'],
        y=ctm_graph.query("segment == 'mid_val'")['monetary'],
        mode='markers',
        name='Mid value',
        marker=dict(
            size=9,
            color='green'
           )
    ),

    go.Scatter(
        x=ctm_graph.query("segment == 'high_val'")['frequency'],
        y=ctm_graph.query("segment == 'high_val'")['monetary'],
        mode='markers',
        name='High value',
        marker=dict(
            size=11,
            color='red'
           )
    ),
]

plot_layout = go.Layout(
        yaxis= {'title': "Revenue"},
        xaxis= {'title': "Frequency"},
        title='Segments',
        title_x = 0.5
    )

fig = go.Figure(
        data=plot_data,
        layout=plot_layout
    )
```

```
#pyoff.iplot(fig)
fig.show(config={'displaylogo': False})


# Scatter Plot of Revenue verses Recency

ctm_graph = ctm_df.query("monetary < 50000 and recency < 500")

plot_data = [
    go.Scatter(
        x=ctm_graph.query("segment == 'low_val'")['recency'],
        y=ctm_graph.query("segment == 'low_val'")['monetary'],
        mode='markers',
        name='Low value',
        marker=dict(
            size=7,
            color='blue'
           )
    ),

    go.Scatter(
        x=ctm_graph.query("segment == 'mid_val'")['recency'],
        y=ctm_graph.query("segment == 'mid_val'")['monetary'],
        mode='markers',
        name='Mid value',
        marker=dict(
            size=9,
            color='green'
           )
    ),

    go.Scatter(
        x=ctm_graph.query("segment == 'high_val'")['recency'],
        y=ctm_graph.query("segment == 'high_val'")['monetary'],
        mode='markers',
        name='High value',
        marker=dict(
            size=11,
            color='red'
```

```
        )
    ),
]

plot_layout = go.Layout(
        yaxis= {'title': "Revenue"},
        xaxis= {'title': "Recency"},
        title='Segments',
        title_x = 0.5
    )

fig = go.Figure(
        data=plot_data,
        layout=plot_layout
    )

#pyoff.iplot(fig)
fig.show(config={'displaylogo': False})


# Scatter Plot of Revenue verses Frequency

ctm_graph = ctm_df.query("recency < 500 and frequency < 200")

plot_data = [
    go.Scatter(
        x=ctm_graph.query("segment == 'low_val'")['recency'],
        y=ctm_graph.query("segment == 'low_val'")['frequency'],
        mode='markers',
        name='Low value',
        marker= dict(
            size=7,
            color='blue'
          )
    ),

    go.Scatter(
        x=ctm_graph.query("segment == 'mid_val'")['recency'],
        y=ctm_graph.query("segment == 'mid_val'")['frequency'],
        mode='markers',
```

```
        name='Mid value',
        marker=dict(
            size=9,
            color='green'
           )
    ),

    go.Scatter(
        x=ctm_graph.query("segment == 'high_val'")['recency'],
        y=ctm_graph.query("segment == 'high_val'")['frequency'],
        mode='markers',
        name='High value',
        marker=dict(
            size=11,
            color='red'
           )
    ),
]

plot_layout = go.Layout(
        yaxis= {'title': "Frequency"},
        xaxis= {'title': "Recency"},
        title='Segments: Frequency vs Recency',
        title_x = 0.5
    )

fig = go.Figure(
        data=plot_data,
        layout=plot_layout
    )

#pyoff.iplot(fig)
fig.show(config={'displaylogo': False})


# Generate dummies using segment's categories

segment = ctm_df['segment']
segment = pd.get_dummies(segment)
segment.head()
```

```
# Save the generated dummies in ctm_df_dummies

ctm_df_dummies = ctm_df.copy()
ctm_df_dummies = ctm_df_dummies.drop('segment', axis=1)
ctm_df_dummies['high_val'] = segment['high_val']
ctm_df_dummies['mid_val'] = segment['mid_val']
ctm_df_dummies['low_val'] = segment['low_val']
ctm_df_dummies.head()


ctm_df_dummies['time_diff_range'] = 1  ## less than 3 months
ctm_df_dummies.loc[ctm_df_dummies.time_diff>90,'time_diff_range'] = 0 # more than 3 months
ctm_df_dummies.head()


# Correlation between features

corr_mtx = ctm_df_dummies.corr()
corr_coef = pd.DataFrame(corr_mtx.min())
corr_coef.columns = ['corr_coef_min']
corr_coef['corr_coef_max'] = corr_mtx[corr_mtx < 1].max()
corr_coef


plt.figure(figsize = (40, 30))
sns.heatmap(corr_mtx, annot = True, linewidths=0.2, fmt=".2f",annot_kws={"fontsize":30})
sns.set(font_scale=2);


ctm_df_dummies.head()


# # Generation of the model

ctm_df_dummies_model = ctm_df_dummies.drop('time_diff', axis = 1)


X = ctm_df_dummies_model.drop('time_diff_range', axis = 1)
```

```
y = ctm_df_dummies_model.time_diff_range

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = None,
shuffle=True)


# Save models in an array

models = []
models.append(("LogisticRegression", LogisticRegression()))
models.append(("GaussianNB", GaussianNB()))
models.append(("RandomForestClassifier", RandomForestClassifier()))
models.append(("SVC", SVC()))
models.append(("DecisionTreeClassifier", DecisionTreeClassifier()))
models.append(("xgb.XGBClassifier", xgb.XGBClassifier(eval_metric = 'mlogloss')))
models.append(("KNeighborsClassifier", KNeighborsClassifier()))


# Measuring the metrics of the different models

scorer = MultiScorer({'accuracy'  : (accuracy_score , {}),
                      'f1_score'  : (f1_score       , {'average':'macro'}),
                      'recall'    : (recall_score   , {'average':'macro'}),
                      'precision' : (precision_score, {'average':'macro'})
                     })

# A dictionary for all the distinct models and their respective metrics

model_scores_dict = {'model_name' : [],
                     'accuracy'   : [],
                     'f1_score'   : [],
                     'recall'     : [],
                     'precision'  : [],
                     'time'       : []
                    }

# For each model name and model in models
for model_name, model in models:

    # Add model_name to model_scores_dict
```

```
    model_scores_dict['model_name'].append(model_name)

    #print(model_name)
    kfold = KFold(n_splits = 2, random_state = 24, shuffle = True)

    start = time.time()

    cross_val_score(model, X_train, y_train, cv = kfold, scoring = scorer)

    cv_result = scorer.get_results()

    # For each metric in cv_result.keys()
    for metric_name in cv_result.keys():

        # Get the average of cv_result[metric_name]
        average_score = np.average(cv_result[metric_name])

        # Update model_scores_dict with average_score for model_name
        model_scores_dict[metric_name].append(average_score)

        #print('%s : %f' %(metric_name, average_score))

    model_scores_dict['time'].append((time.time() - start))
    #print('time : ', time.time() - start, '\n\n')

model_score_df = pd.DataFrame(model_scores_dict).set_index("model_name")

model_score_df.sort_values(by = ["accuracy", "f1_score", "time"], ascending = False)


parameter = {
    'max_depth':range(3,15,2),
    'min_child_weight':range(1,10,2)
    }

p_grid_search = GridSearchCV(estimator = xgb.XGBClassifier(eval_metric='mlogloss'),
                             param_grid = parameter,
                             scoring='accuracy',
                             n_jobs=-1,
                             cv=2
```

```
                        )

p_grid_search.fit(X_train, y_train)


p_grid_search.best_params_, p_grid_search.best_score_


refined_xgb_model = xgb.XGBClassifier(eval_metric='logloss',
                                      max_depth=list(p_grid_search.best_params_.values())[0]-2,
                                      min_child_weight=list(p_grid_search.best_params_.values())[1]+3
                                     ).fit(X_train, y_train)

print('Accuracy of XGB classifier on training set: {:.2f}'.format(refined_xgb_model.score(X_train,
y_train)))
print('Accuracy of XGB classifier on test set:
{:.2f}'.format(refined_xgb_model.score(X_test[X_train.columns], y_test)))


ref_xgb_pred_y = refined_xgb_model.predict(X_test)


rf_pred_y = RandomForestClassifier().fit(X_train, y_train).predict(X_test)


plot_conf_mtx(np.array(y_test), ref_xgb_pred_y);


plot_conf_mtx(np.array(y_test), rf_pred_y);


# A dictionary of model names with the various metrics
ref_xgb_rf_dict = {"model_name" : ["xgb.XGBClassifier", "RandomForestClassifier"],
                   "accuracy"   : [accuracy_score(y_test, ref_xgb_pred_y), accuracy_score(y_test,
rf_pred_y)],
                   "f1_score"   : [f1_score(y_test, ref_xgb_pred_y), f1_score(y_test, rf_pred_y)],
                   "recall"     : [recall_score(y_test, ref_xgb_pred_y), recall_score(y_test, rf_pred_y)],
                   "precision"  : [precision_score(y_test, ref_xgb_pred_y), precision_score(y_test,
rf_pred_y)]
                  }
```

```
# Create a dataframe with ref_xgb_rf_dict
ref_xgb_rf_df = pd.DataFrame(ref_xgb_rf_dict).set_index("model_name")

# Order the dataframe ref_xgb_log_reg_df by the metric values in increasing order

ref_xgb_rf_df.sort_values(by=["accuracy", "f1_score", "recall", "precision"], ascending=False)
```