Data Science Project

Predicting Santander Bank Customers' Satisfaction in Real - Time

Fellipe Augusto Soares Silva

2020

Chapters

Figures

Tables

Summary

This project was developed with the purpose of forecasting Santander Bank Customers' Satisfaction in Real Time, providing metrics for the company to act more quickly in preventing customer losses and also to retain satisfied consumer.

Using Pyhton programming language together with Apache Spark (cluster computing platform) large amounts of data were collected in order to implement a Supervised Machine Learning Predictive Model in historical data.

For this project, the dataset provided by Santander Bank to Kaggle, an online community of data and machine learning scientists owned by Google LLC, was used as open data for the community.

With the data provided I developed: data loading, Feature engineering, cleaning and transformation, processing in memory ensuring speed, exploratory analyzes evidencing opportunities for the company, Machine Learning and a series of other practices in Spark.

After exploring all the information provided by the data, two predictive classification models were implemented in test data to train and analyze the model's accuracy.

After selecting the best machine learning model, I simulated a real environment with a batch of data acquisition (previously unknown) for a whole week (Monday to Friday) presenting the results on PowerBI visualization platform with interactive graphics showing unsatisfied consumers and generating action opportunities for decision makers.

This project's differential will also be the presentation of Spark Jobs being executed in real time through PySparkShell, a powerful tool for interactive analysis of data behavior.

Key words: Data Science, Machine Learning, Python, Big Data, Business Intelligence, BI, Spark, Pyspark, Pyspark Shell, Spark RDD, Spark SQL, Spark MLLib, Accuracy, Confusion Matrix, Santander, Customer Satisfaction, Kaggle.

1. Introduction

This project was developed using the Spark API[1] for Python[2], the Pyspark, and Jupyter Notebook[3] as a script development and testing platform[4], in addition to the Pyspark Shell for interactive analysis of the execution of Jobs.

Spark is a tool that belongs to the Apache Software Foundation, has a large capacity for processing Big Data and is one of the most implemented mechanisms by companies today. In addition to being open-source, Spark provides speed and ease in handling large amount of data through its high-level libraries.

This project used the concepts and practices of:

- Batch processing with Spark RDD[5];
- SQL Query with Spark SQL[6];
- Spark Session to use Pyspark Dataframes[7];
- Machine Learning with Spark MLLib[8].

Python is a programming language that has been gaining space in the job market due to its versatility and functionality when combined with more than 100,000 other libraries such as:

- NumPy for numerical computing;
- Pandas for data manipulation;
- SciPy for scientific computing;
- MatplotLib for visualization and plotting.

In this way, Pyspark is a Spark API for the Python programming language, allowing the combination of these two concepts previously presented, transforming the tool into an excellent option for use in distributed computing, Big Data and data streaming projects.

[1] https://spark.apache.org/docs/latest/api/python/index.html
[2] Python is an object-oriented programming language.
[3] https://jupyter.org/
[4] Script is a set of instructions in code written in computer language so that it performs different functions inside a program.
[5] https://spark.apache.org/docs/latest/api/python/pyspark.html?highlight=rdd#pyspark.RDD
[6] https://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=sql#module-pyspark.sql
[7]https://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=dataframe#pyspark.sql.SparkSession .createDataFrame
[8] https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html?highlight=mllib

Jupyter Notebook[9] , on the other hand, is an open-source project conceived in 2014 as a web computing environment for creating codes through different programming languages such as R[10], Scala[11], Python[12] among others. Jupyter is a tool that also allows you to document your work in different sources such as HTML, LaTeX[13], PNG, PDF, SVG, among others.

Finally, a tool available for analyzing the distribution and recovery of data within the cluster is Pyspark Shell[14]. With it, each job is presented in real time with its execution chain and the time it took to finish the job.

To complement the knowledge of these data and provide a better view of satisfied and unsatisfied customers, PowerBi[15] was also used to create a dashboard with interactive graphics for the Business Intelligence process.

Like any Data Science project, before starting the parameterization of the predictive model it is necessary to perform exploratory analysis and to know the dataset.

To conclude, I will present all script items, commented line by line and presenting graphical analysis to complement the understanding of the development of this Data Science project.

The beginning of business problem solving lies in understanding the problem itself, which will be presented in the following chapter.

During the presentation of this project I will expose the most important parts of the code. You will find in the last chapter the source code commented in full. I decided to do it this way so that the presentation of the project would be more dynamic and less heavy so that people from different business areas could understand the potential of data science.

[9] https://jupyter.org/
[10] https://www.r-project.org/
[11] https://www.scala-lang.org/
[12] https://www.python.org/
[13] https://www.latex-project.org/
[14] https://spark.apache.org/docs/latest/quick-start.html
[15] Developed and provided by Microsoft, Power BI is a set of Business Intelligence tools used to create cloud dashboards for data and business analysis.

2. Business Problem

Measuring customer satisfaction is important for companies to assess whether their business is flowing in a healthy way. A satisfied customer is one who has their expectations exceeded, whether during the purchase of services, after sales, or when handling tools on a daily basis, among other possibilities, but each consumer has his understanding of what makes him loyal to the brand.

Understanding your target audience helps to direct more assertive projects and lead company employees according to established goals, study the market and its evolution, study your product and how it is marketed, among many other sources.

We are in an era that just selling a product is not enough to build customer loyalty, it is necessary to look at what brings value to that consumer. Satisfaction is measured when the consumer has a product (or service) and begins to compare it with others available on the market. With globalization and easy access to information over the internet, it is possible to see the opinions of other consumers and assess whether what is being offered is worth maintaining or exchanging.

Listening to the customer is important because his unsatisfaction can be a good indicator of failures in the way the product / service is being offered and with that we generate opportunities for improvement. For this project, we are going to study ways to predict more quickly which customer is not satisfied by analyzing a large set of data and not just looking at the opinion of satisfaction or not, but at all the interactions he does with Santander Bank.

With this information in hand, decision makers can act more quickly in the prevention processes and not in the containment process because this may be too late and the customer has already made his decision. A satisfied customer is a customer who is loyal to the brand and in order to maintain loyalty, it is necessary to have relevant information in a timely manner.

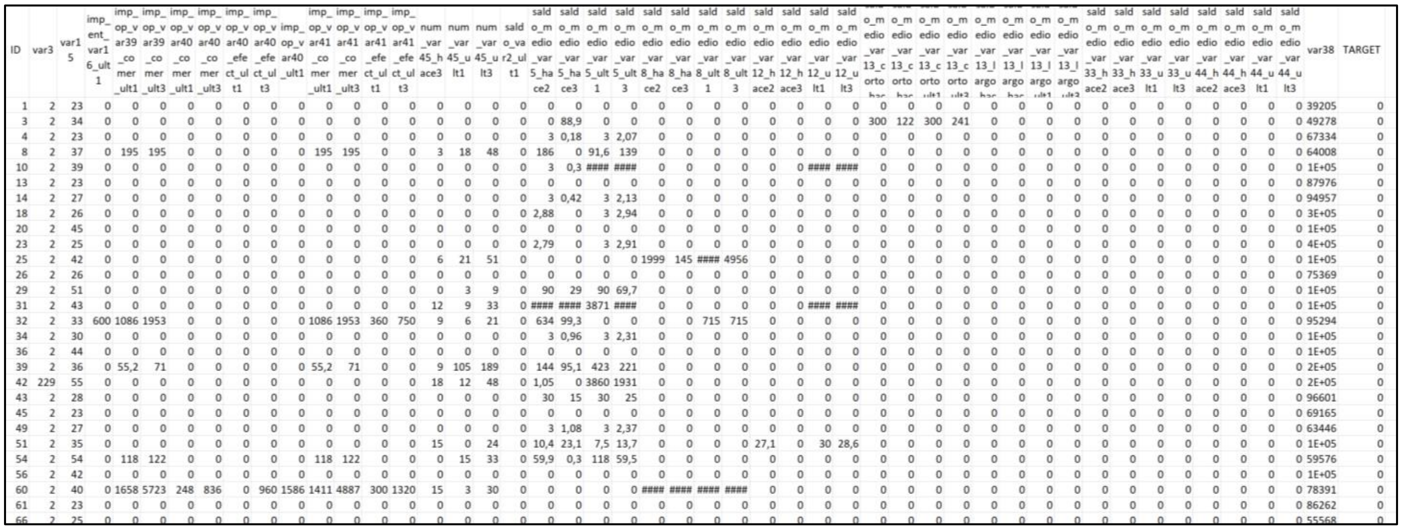
Aiming at this agility, Spark was chosen to process a large volume of information and deliver reliability to the company, employees and its stakeholders. With this project I sought to introduce intelligence to the business, generating opportunities and increasing profitability.

3. Dataset

For this business problem we have the following data set:

3.1 Training and testing data

This dataset is structured as follows:



ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3	imp_op_var40_comer_ult1	imp_op_var40_comer_ult3	imp_op_var40_efect_ult1	imp_op_var40_efect_ult3	imp_op_var40_ult1	imp_op_var41_comer_ult1	imp_op_var41_comer_ult3	imp_op_var41_efect_ult1	imp_op_var41_efect_ult3	num_var45_hace3	num_var45_ult1	num_var45_ult3	saldo_var2_ult1	saldo_medio_var5_hace2	saldo_medio_var5_hace3	saldo_medio_var5_ult1	saldo_medio_var5_ult3	saldo_medio_var8_hace2	saldo_medio_var8_hace3	saldo_medio_var8_ult1	saldo_medio_var8_ult3	saldo_medio_var12_hace2	saldo_medio_var12_hace3	saldo_medio_var12_ult1	saldo_medio_var12_ult3	saldo_medio_var13_corto_hac	saldo_medio_var13_corto_hac	saldo_medio_var13_corto_ult1	saldo_medio_var13_corto_ult3	saldo_medio_var13_largo_hac	saldo_medio_var13_largo_hac	saldo_medio_var13_largo_ult1	saldo_medio_var13_largo_ult3	saldo_medio_var33_hace2	saldo_medio_var33_hace3	saldo_medio_var33_ult1	saldo_medio_var33_ult3	saldo_medio_var44_hace2	saldo_medio_var44_hace3	saldo_medio_var44_ult1	saldo_medio_var44_ult3	var38	TARGET
1	2	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	39205	0
3	2	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	88,9	0	0	0	0	0	0	0	0	0	0	300	122	300	241	0	0	0	0	0	0	0	0	0	0	0	0	49278	0
4	2	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0,18	3	2,07	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	67334	0
8	2	37	0	195	195	0	0	0	0	0	195	195	0	0	3	18	48	0	186	0	91,6	139	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	64008	0
10	2	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0,3	####	####	0	0	0	0	0	0	####	####	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1E+05	0
13	2	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	87976	0
14	2	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0,42	3	2,13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	94957	0
18	2	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2,88	0	3	2,94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3E+05	0
20	2	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1E+05	0
23	2	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2,79	0	3	2,91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4E+05	0
25	2	42	0	0	0	0	0	0	0	0	0	0	0	0	6	21	51	0	0	0	0	0	1999	145	####	4956	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1E+05	0
26	2	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	75369	0
29	2	51	0	0	0	0	0	0	0	0	0	0	0	0	0	3	9	0	90	29	90	69,7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1E+05	0
31	2	43	0	0	0	0	0	0	0	0	0	0	0	0	12	9	33	0	####	####	3871	####	0	0	0	0	0	0	####	####	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1E+05	0
32	2	33	600	1086	1953	0	0	0	0	0	1086	1953	360	750	9	6	21	0	634	99,3	0	0	0	0	715	715	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	95294	0
34	2	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0,96	3	2,31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1E+05	0
36	2	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1E+05	0
39	2	36	0	55,2	71	0	0	0	0	0	55,2	71	0	0	9	105	189	0	144	95,1	423	221	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2E+05	0
42	229	55	0	0	0	0	0	0	0	0	0	0	0	0	18	12	48	0	1,05	0	3860	1931	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2E+05	0
43	2	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	15	30	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96601	0
45	2	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	69165	0
49	2	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1,08	3	2,37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	63446	0
51	2	35	0	0	0	0	0	0	0	0	0	0	0	0	15	0	24	0	10,4	23,1	7,5	13,7	0	0	0	0	27,1	0	30	28,6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1E+05	0
54	2	54	0	118	122	0	0	0	0	0	118	122	0	0	0	15	33	0	59,9	0,3	118	59,5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59576	0
56	2	42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1E+05	0
60	2	40	0	1658	5723	248	836	0	960	1586	1411	4887	300	1320	15	3	30	0	0	0	0	0	####	####	####	####	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	78391	0
61	2	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	86262	0
66	2	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	55568	0

Figure 1 - Train Dataset

Figure 1 illustrates only a piece of the entire dataset that has:

- Rows: 76.020
- Columns: 371
- Total Data: 28.203.420

We have 28,203,420 pieces of information to analyze and return the level of customer satisfaction, a very large amount of information that an ordinary machine cannot process. This is one of the big data concepts, the "v" of volume.

The best way to store and process this amount of data is through a cluster, that is, a set of machines working all together. Spark together with HDFS offers the best tool for the business problem because while HDFS[16] (Hadoop Distributed File System) acquires the data and distributes it into the cluster, spark acquires the data stored in the cluster by performing parallel mapping and processing.

[16] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

4. Data Dictionary

This is a different business problem as we do not have a clear description of each information within the dataset, in fact it is the exact opposite, we have 369 anonymous columns and only 2 known ones, ID and TARGET.

For this reason, it is not possible to describe each variable in detail using a data dictionary.

As we are working with Pyspark the reading mode is different from the conventional one provided by Python libraries. In this case, we have to create RDD's and load the data into them.

RDD[17] stands for Resilient Distributed Datasets and corresponds to a collection of objects partitioned in the cluster[18], distributed, immutable and with Spark structure.

Using a Spark Context - sc - (object that tells spark how to connect to the cluster) we read the dataset as follows:

```
santanderRDD = sc.textFile("train.csv")
```

textFile transforms a csv file into an RDD object. At this point, the santanderRDD object received the data, distributed it in the cluster and became immutable. To acquire any information we must perform actions on the RDD's as:

```
santanderRDD.take(2)
```

This action returns the first 2 rows of the dataset (note the large amount of information returned from the cluster):

```
['ID,var3,var15,imp_ent_var16_ult1,imp_op_var39_comer_ult1,imp_op_var39_comer_ult3,imp_op_var40_c
omer_ult1,imp_op_var40_comer_ult3,imp_op_var40_efect_ult1,imp_op_var40_efect_ult3,imp_op_var40_ul
t1,imp_op_var41_comer_ult1,imp_op_var41_comer_ult3,imp_op_var41_efect_ult1,imp_op_var41_efect_ult
3,imp_op_var41_ult1,imp_op_var39_efect_ult1,imp_op_var39_efect_ult3,imp_op_var39_ult1,imp_sal_var
16_ult1,ind_var1_0,ind_var1,ind_var2_0,ind_var2,ind_var5_0,ind_var5,ind_var6_0,ind_var6,ind_var8_
0,ind_var8,ind_var12_0,ind_var12,ind_var13_0,ind_var13_corto_0,ind_var13_corto,ind_var13_largo_0,
ind_var13_largo,ind_var13_medio_0,ind_var13_medio,ind_var13,ind_var14_0,ind_var14,ind_var17_0,ind
_var17,ind_var18_0,ind_var18,ind_var19,ind_var20_0,ind_var20,ind_var24_0,ind_var24,ind_var25_cte,
ind_var26_0,ind_var26_cte,ind_var26,ind_var25_0,ind_var25,ind_var27_0,ind_var28_0,ind_var28,ind_v
ar27,ind_var29_0,ind_var29,ind_var30_0,ind_var30,ind_var31_0,ind_var31,ind_var32_cte,ind_var32_0,
ind_var32,ind_var33_0,ind_var33,ind_var34_0,ind_var34,ind_var37_cte,ind_var37_0,ind_var37,ind_var
39_0,ind_var40_0,ind_var40,ind_var41_0,ind_var41,ind_var39,ind_var44_0,ind_var44,ind_var46_0,ind_
var46,num_var1_0,num_var1,num_var4,num_var5_0,num_var5,num_var6_0,num_var6,num_var8_0,num_var8,nu
m_var12_0,num_var12,num_var13_0,num_var13_corto_0,num_var13_corto,num_var13_largo_0,num_var13_lar
go,num_var13_medio_0,num_var13_medio,num_var13,num_var14_0,num_var14,num_var17_0,num_var17,num_va
r18_0,num_var18,num_var20_0,num_var20,num_var24_0,num_var24,num_var26_0,num_var26,num_var25_0,num
_var25,num_op_var40_hace2,num_op_var40_hace3,num_op_var40_ult1,num_op_var40_ult3,num_op_var41_hac
e2,num_op_var41_hace3,num_op_var41_ult1,num_op_var41_ult3,num_op_var39_hace2,num_op_var39_hace3,n
um_op_var39_ult1,num_op_var39_ult3,num_var27_0,num_var28_0,num_var28,num_var27,num_var29_0,num_va
r29,num_var30_0,num_var30,num_var31_0,num_var31,num_var32_0,num_var32,num_var33_0,num_var33,num_v
ar34_0,num_var34,num_var35,num_var37_med_ult2,num_var37_0,num_var37,num_var39_0,num_var40_0,num_v
ar40,num_var41_0,num_var41,num_var39,num_var42_0,num_var42,num_var44_0,num_var44,num_var46_0,num_
var46,saldo_var1,saldo_var5,saldo_var6,saldo_var8,saldo_var12,saldo_var13_corto,saldo_var13_largo
```

[17] https://spark.apache.org/docs/latest/rdd-programming-guide.html

[18] Set of interconnected computers that work as if they were one big system.

```
,saldo_var13_medio,saldo_var13,saldo_var14,saldo_var17,saldo_var18,saldo_var20,saldo_var24,saldo_
var26,saldo_var25,saldo_var28,saldo_var27,saldo_var29,saldo_var30,saldo_var31,saldo_var32,saldo_v
ar33,saldo_var34,saldo_var37,saldo_var40,saldo_var41,saldo_var42,saldo_var44,saldo_var46,var36,de
lta_imp_amort_var18_1y3,delta_imp_amort_var34_1y3,delta_imp_aport_var13_1y3,delta_imp_aport_var17
_1y3,delta_imp_aport_var33_1y3,delta_imp_compra_var44_1y3,delta_imp_reemb_var13_1y3,delta_imp_ree
mb_var17_1y3,delta_imp_reemb_var33_1y3,delta_imp_trasp_var17_in_1y3,delta_imp_trasp_var17_out_1y3
,delta_imp_trasp_var33_in_1y3,delta_imp_trasp_var33_out_1y3,delta_imp_venta_var44_1y3,delta_num_a
port_var13_1y3,delta_num_aport_var17_1y3,delta_num_aport_var33_1y3,delta_num_compra_var44_1y3,del
ta_num_reemb_var13_1y3,delta_num_reemb_var17_1y3,delta_num_reemb_var33_1y3,delta_num_trasp_var17_
in_1y3,delta_num_trasp_var17_out_1y3,delta_num_trasp_var33_in_1y3,delta_num_trasp_var33_out_1y3,d
elta_num_venta_var44_1y3,imp_amort_var18_hace3,imp_amort_var18_ult1,imp_amort_var34_hace3,imp_amo
rt_var34_ult1,imp_aport_var13_hace3,imp_aport_var13_ult1,imp_aport_var17_hace3,imp_aport_var17_ul
t1,imp_aport_var33_hace3,imp_aport_var33_ult1,imp_var7_emit_ult1,imp_var7_recib_ult1,imp_compra_v
ar44_hace3,imp_compra_var44_ult1,imp_reemb_var13_hace3,imp_reemb_var13_ult1,imp_reemb_var17_hace3
,imp_reemb_var17_ult1,imp_reemb_var33_hace3,imp_reemb_var33_ult1,imp_var43_emit_ult1,imp_trans_va
r37_ult1,imp_trasp_var17_in_hace3,imp_trasp_var17_in_ult1,imp_trasp_var17_out_hace3,imp_trasp_var
17_out_ult1,imp_trasp_var33_in_hace3,imp_trasp_var33_in_ult1,imp_trasp_var33_out_hace3,imp_trasp_
var33_out_ult1,imp_venta_var44_hace3,imp_venta_var44_ult1,ind_var7_emit_ult1,ind_var7_recib_ult1,
ind_var10_ult1,ind_var10cte_ult1,ind_var9_cte_ult1,ind_var9_ult1,ind_var43_emit_ult1,ind_var43_re
cib_ult1,var21,num_var2_0_ult1,num_var2_ult1,num_aport_var13_hace3,num_aport_var13_ult1,num_aport
_var17_hace3,num_aport_var17_ult1,num_aport_var33_hace3,num_aport_var33_ult1,num_var7_emit_ult1,n
um_var7_recib_ult1,num_compra_var44_hace3,num_compra_var44_ult1,num_ent_var16_ult1,num_var22_hace
2,num_var22_hace3,num_var22_ult1,num_var22_ult3,num_med_var22_ult3,num_med_var45_ult3,num_meses_v
ar5_ult3,num_meses_var8_ult3,num_meses_var12_ult3,num_meses_var13_corto_ult3,num_meses_var13_larg
o_ult3,num_meses_var13_medio_ult3,num_meses_var17_ult3,num_meses_var29_ult3,num_meses_var33_ult3,
num_meses_var39_vig_ult3,num_meses_var44_ult3,num_op_var39_comer_ult1,num_op_var39_comer_ult3,num
_op_var40_comer_ult1,num_op_var40_comer_ult3,num_op_var40_efect_ult1,num_op_var40_efect_ult3,num_
op_var41_comer_ult1,num_op_var41_comer_ult3,num_op_var41_efect_ult1,num_op_var41_efect_ult3,num_o
p_var39_efect_ult1,num_op_var39_efect_ult3,num_reemb_var13_hace3,num_reemb_var13_ult1,num_reemb_v
ar17_hace3,num_reemb_var17_ult1,num_reemb_var33_hace3,num_reemb_var33_ult1,num_sal_var16_ult1,num
_var43_emit_ult1,num_var43_recib_ult1,num_trasp_var11_ult1,num_trasp_var17_in_hace3,num_trasp_var
17_in_ult1,num_trasp_var17_out_hace3,num_trasp_var17_out_ult1,num_trasp_var33_in_hace3,num_trasp_
var33_in_ult1,num_trasp_var33_out_hace3,num_trasp_var33_out_ult1,num_venta_var44_hace3,num_venta_
var44_ult1,num_var45_hace2,num_var45_hace3,num_var45_ult1,num_var45_ult3,saldo_var2_ult1,saldo_me
dio_var5_hace2,saldo_medio_var5_hace3,saldo_medio_var5_ult1,saldo_medio_var5_ult3,saldo_medio_var
8_hace2,saldo_medio_var8_hace3,saldo_medio_var8_ult1,saldo_medio_var8_ult3,saldo_medio_var12_hace
2,saldo_medio_var12_hace3,saldo_medio_var12_ult1,saldo_medio_var12_ult3,saldo_medio_var13_corto_h
ace2,saldo_medio_var13_corto_hace3,saldo_medio_var13_corto_ult1,saldo_medio_var13_corto_ult3,sald
o_medio_var13_largo_hace2,saldo_medio_var13_largo_hace3,saldo_medio_var13_largo_ult1,saldo_medio_
var13_largo_ult3,saldo_medio_var13_medio_hace2,saldo_medio_var13_medio_hace3,saldo_medio_var13_me
dio_ult1,saldo_medio_var13_medio_ult3,saldo_medio_var17_hace2,saldo_medio_var17_hace3,saldo_medio
_var17_ult1,saldo_medio_var17_ult3,saldo_medio_var29_hace2,saldo_medio_var29_hace3,saldo_medio_va
r29_ult1,saldo_medio_var29_ult3,saldo_medio_var33_hace2,saldo_medio_var33_hace3,saldo_medio_var33
_ult1,saldo_medio_var33_ult3,saldo_medio_var44_hace2,saldo_medio_var44_hace3,saldo_medio_var44_ul
t1,saldo_medio_var44_ult3,var38,TARGET',
 '1,2,23,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0,
0,0,0,0,0,0,0,0,3,0,0,3,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,99,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39205.17,0']
```

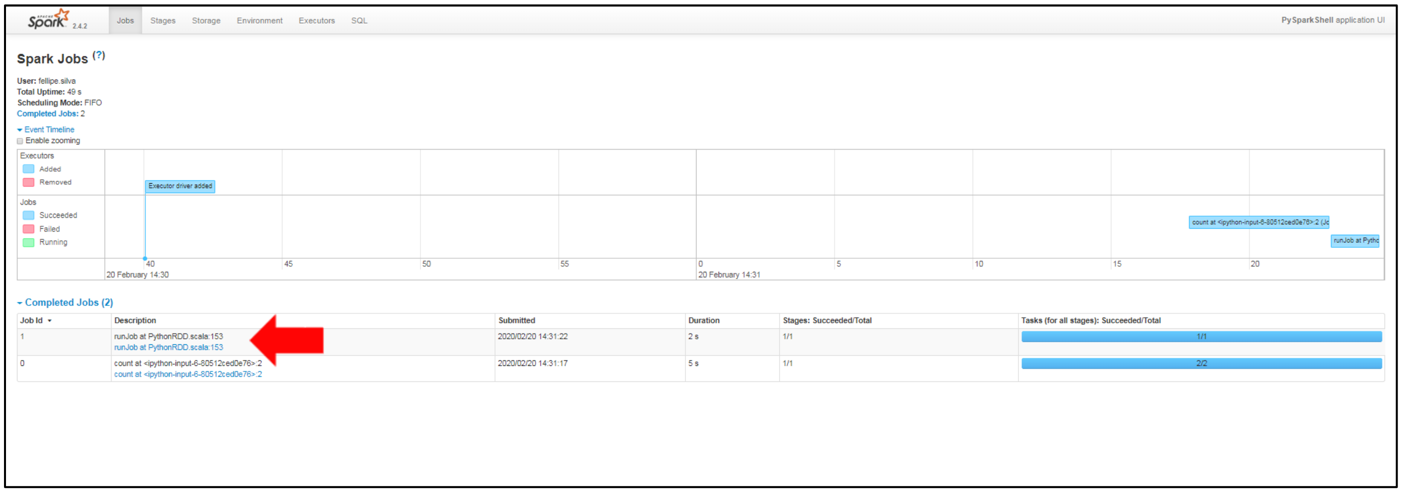
Note that each line has its data in single quotes (shown in red in the image above), indicating that they were recognized as a single string (text). It is necessary to transform this information so that each data is in its respective column, so, before the exploratory analysis we will carry out the feature engineering, treating and adding relevant variables.

Even before performing feature engineering, let's understand how PySpark Shell works.

5. PySpark Shell

As stated earlier in the introductory chapter from now on we will analyze the execution of the Jobs generated in Pyspark Shell. Not all will be presented as there is repetition in the code's actions, but the most relevant will be highlighted in the documentation of this project. Interesting fact will be when we enter the chapter of Machine Learning because it will be possible to see the large amount of Jobs generated sequentially.

The action of the previous chapter ("take") generated the following job:



Spark Jobs (?)

User: fellipe.silva
Total Uptime: 49 s
Scheduling Mode: FIFO
Completed Jobs: 2

Completed Jobs (2)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	runJob at PythonRDD.scala:153 runJob at PythonRDD.scala:153	2020/02/20 14:31:22	2 s	1/1	1/1
0	count at <ipython-input-6-80512ced0e76>:2 count at <ipython-input-6-80512ced0e76>:2	2020/02/20 14:31:17	5 s	1/1	2/2

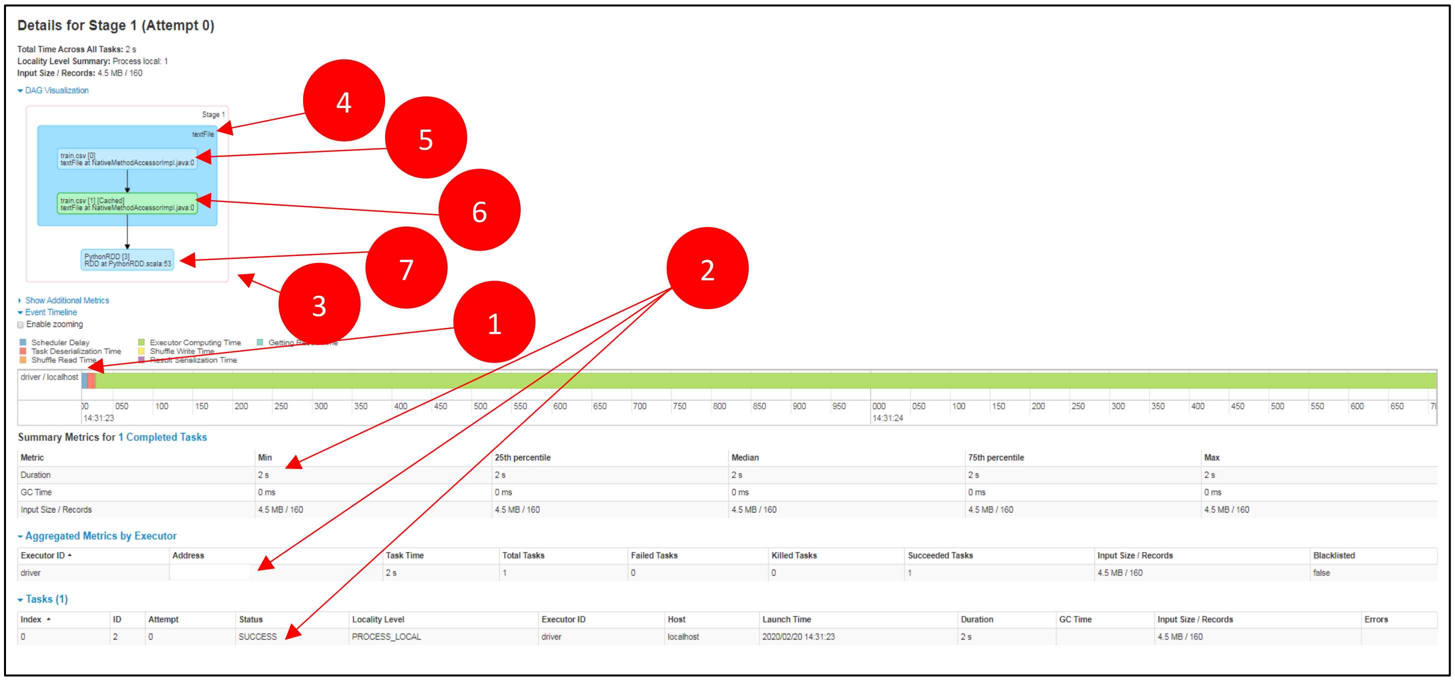
Figure 2 - Pyspark Shell I

As illustrated in the figure above, we have two Jobs executed, one from the test I performed earlier to retrieve from the cluster the number of rows we have in the data ("count") and the job with the red arrow ("take") that retrieves all the data of my santanderRDD.

With the action of Job id 0 it takes us four seconds to recover, and with Job id 1 it takes us a second, because in the action of the "take" I requested only the first 2 lines, while the action of the "count" requested everything there is in the cluster.

Note that we have a timeline where it shows in real time the execution of all Jobs. At the end of the project, we will have a series of operations filling this timeline.

Within each Job we can see the details of its execution:



Details for Stage 1 (Attempt 0)

Total Time Across All Tasks: 2 s
Locality Level Summary: Process local: 1
Input Size / Records: 4.5 MB / 160

DAG Visualization

Summary Metrics for 1 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	2 s	2 s	2 s	2 s	2 s
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Input Size / Records	4.5 MB / 160	4.5 MB / 160	4.5 MB / 160	4.5 MB / 160	4.5 MB / 160

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Blacklisted
driver		2 s	1	0	0	1	4.5 MB / 160	false

Tasks (1)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	2	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2020/02/20 14:31:23	2 s		4.5 MB / 160	

Figure 3 - Pyspark Shell II

In item 1 we can see the specific timeline of that Job within the cluster. Note that because we are not in a production environment we have only one machine in the cluster, localhost. If we had a configured cluster, Jobs' data and "tasks" would be distributed among the machines and we would see the execution on each of them.

In item 2 we have details of the execution time, metrics indicating the time spent in each quartile of execution, the address of the machine that requested the "task", how much memory was used and if the Job was successful or failed.

Item 3 concerns the visualization of the DAG (Direct Acyclic Graph) and refers to the chain of dependencies and execution of an RDD. In this case we can see that spark executes the concept of pipeline[19] in operations, passing from instruction to instruction until the final result.

In this DAG we have blue and green boxes. The box in item 4 represents an operation that I wrote in my code, in this case a read operation with the "textfile". This reading is from an input file on HDFS[20] (Hadoop Distributed File System). The box in item 5 represents the RDD generated by this reading operation. Each smaller box (items 5, 6, 7) is RDD's.

Then, in green (item 6) we have the same instruction as item 5, but with the command "cache" persisting the RDD in memory for faster access later, so that I don't need to access this data in HDFS.

Finally in item 7 we have the final RDD with the action "take" returning the result previously presented.

We move on to the next chapter with Feature Engineering.

[19] Data entry is processed and delivered for next processing following a queue.
[20] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

6. Feature Engineering

Feature Engineering is the process of treating, adding and removing variables. This process consists of finding out which columns of data create the most useful attributes to improve the accuracy of the machine learning model.

As our model does not read string, I removed the first line with the action "filter", that is, it filters everything that does not have the determined condition:

```
santanderRDD2 = santanderRDD.filter(lambda x: "ID" not in x)
```

Note that I did not attribute the result to the RDD already created (santanderRDD) as we would have an error message indicating its immutability. This way we have santanderRDD2 to perform cleaning and transformation.

```
['1,2,23,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0
,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0
,0,0,0,0,0,3,0,0,3,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,99,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,39205.17,0']
```

With the data without the header, we perform transformation actions, so I created a function facilitating the process of manipulating the variables where I initially divide the dataset using the comma separator (“,”) and later assigning each variable to an object converting to int, float , str according to its type.

Still in the function, after the process above, each variable will be inserted in an object "lines" through the function “Row”[21] that will prepare the lines to later create a Pyspark Dataframe assisting in the use of the "select" function of SparkSql for exploratory analysis.

An important piece of information is that the “Row” function accepts only 255 arguments, however I have 371, knowing that I performed the treatment twice with created function, each one for half data.

Not all variables will be used, but initially I had to perform the transformation and analyze it one by one.

[21] https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.Row

```
# Feature Engineering I - Function
def feateng1(data):

    # Dividing data into columns, separating by "," character
    dataList = data.split(",")

    TARGET = float(dataList[370])# 1 for unsatisfied customers and 0 for satisfied customers.
    VAR1 = float(dataList[1])
    VAR2 = float(dataList[2])
    VAR3 = float(dataList[3])
    VAR4 = float(dataList[4])
    VAR5 = float(dataList[5])
    VAR6 = float(dataList[6])
    VAR7 = float(dataList[7])
    VAR8 = float(dataList[8])
    VAR9 = float(dataList[9])
    VAR10 = float(dataList[10])
    VAR11 = float(dataList[11])
    VAR12 = float(dataList[12])
    …
    VAR184 = float(dataList[184])
    VAR185 = float(dataList[185])

    # Creating 'lines' using the 'Row' function, preparing to the dataframe a
nalysis, cleaning and converting the data from string to float
    lines1 = Row(TARGET = TARGET, VAR1 = VAR1, VAR2 = VAR2, VAR3 = VAR3, VAR4
= VAR4, VAR5 = VAR5, VAR6 = VAR6, VAR7 = VAR7 ... VAR184 = VAR184, VAR185 = V
AR185)

    return lines1
```

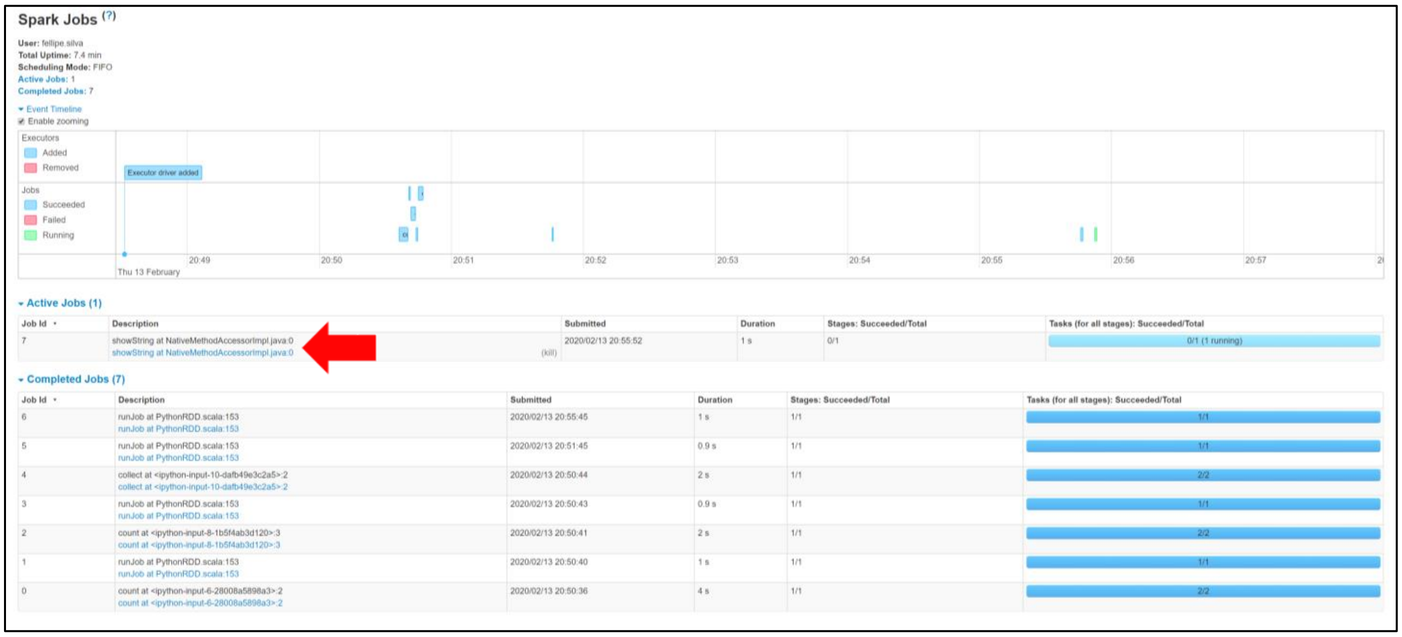
After mapping, we can create a PySpark SQL Dataframe to use a query to obtain data and analyze it.

```
        santanderDF_1 = spSession.createDataFrame(santanderRDD3_1)
```

I used santanderDF_1 object to structure the variables (source code at the end of this document) resulting in the following columns:

```
 |-- TARGET: double (nullable = true)
 |-- VAR1: double (nullable = true)
 |-- VAR10: double (nullable = true)
 |-- VAR100: double (nullable = true)
 |-- VAR101: double (nullable = true)
 |-- VAR102: double (nullable = true)
 |-- VAR103: double (nullable = true)
 |-- VAR104: double (nullable = true)
  ...
 |-- VAR97: double (nullable = true)
 |-- VAR98: double (nullable = true)
 |-- VAR99: double (nullable = true)
```

With the feature engineering completed, we can start the exploratory analysis process.



Spark Jobs (?)

User: fellipe.silva
Total Uptime: 7.4 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 7

▾ Event Timeline
☑ Enable zooming

▾ Active Jobs (1)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
7	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0 (kill)	2020/02/13 20:55:52	1 s	0/1	0/1 (1 running)

▾ Completed Jobs (7)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6	runJob at PythonRDD.scala:153 runJob at PythonRDD.scala:153	2020/02/13 20:55:45	1 s	1/1	1/1
5	runJob at PythonRDD.scala:153 runJob at PythonRDD.scala:153	2020/02/13 20:51:45	0.9 s	1/1	1/1
4	collect at <ipython-input-10-dafb49e3c2a5>:2 collect at <ipython-input-10-dafb49e3c2a5>:2	2020/02/13 20:50:44	2 s	1/1	2/2
3	runJob at PythonRDD.scala:153 runJob at PythonRDD.scala:153	2020/02/13 20:50:43	0.9 s	1/1	1/1
2	count at <ipython-input-8-1b5f4ab3d120>:3 count at <ipython-input-8-1b5f4ab3d120>:3	2020/02/13 20:50:41	2 s	1/1	2/2
1	runJob at PythonRDD.scala:153 runJob at PythonRDD.scala:153	2020/02/13 20:50:40	1 s	1/1	1/1
0	count at <ipython-input-6-28008a5898a3>:2 count at <ipython-input-6-28008a5898a3>:2	2020/02/13 20:50:36	4 s	1/1	2/2

Figure 4 – Processing Jobs

Notice on the red arrow in the Pyspark Shell that Jobs running also appear while they are being processed.

7. Exploratory Analysis

Exploratory analysis is essential to understand where we have better variables, where we have problems, where we have business opportunities, and in this way, we can generate a series of important conclusions to continue the machine learning process.

To perform exploratory analysis Pyspark SQL functions were initially used, which are minor modifications in the traditional SQL language. However, as the traditional SQL language is more widely used, I made a modification in Pyspark Dataframe so that I could use standard SQL language.

This is a Pyspark SQL statement:

```
    santanderDF_2.cube('TARGET').agg(count('TARGET').alias(' Target_Data'),
                                     grouping('TARGET')).orderBy('TARGET').show()
```

Resulting in:

```
+------+-----------+----------------+
|TARGET|Target_Data|grouping(TARGET)|
+------+-----------+----------------+
|  null|      76020|               1|
|   0.0|      73012|               0|
|   1.0|       3008|               0|
+------+-----------+----------------+
```

Table 1 - Spark SQL

For those who have never worked with Spark it is necessary to consult its extensive documentation[22] and search for the necessary commands to perform selections, aggregations, ordering and organization of data.

Due to the ease of data manipulation with SQL language I created a temporary table ("santanderTB_2") in memory, speeding up data recovery and facilitating the use of SQL commands.

This is the same command used in table 1 but in SQL language:

```
  spSession.sql("select TARGET, count(TARGET) as Target_Data \
                 from santanderTB_2 \
                 group by TARGET")
```

That returns as a result:

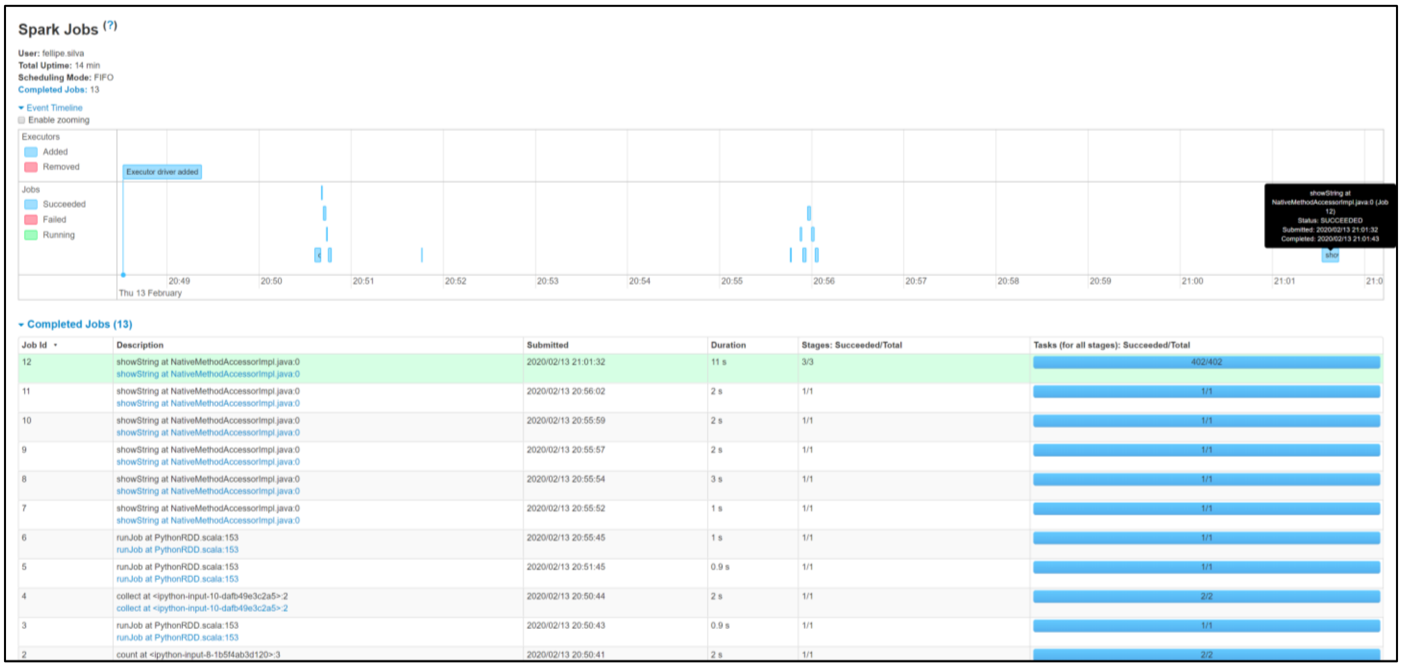
[22] https://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=sql#pyspark.sql.DataFrame

```
+------+---------------+
|TARGET|  Target_Data   |
+------+---------------+
|   0.0|          73012|
|   1.0|           3008|
+------+---------------+
```

Table 2 - SQL ANSII

Both commands return the number of customers in the dataset that have target 0 and target 1, that is, satisfied customers and unsatisfied customers, respectively. Note that we have many more satisfied customers.

Pyspark Shell in this case was processed with more instructions within the cluster:



Spark Jobs (?)

User: fellipe.silva
Total Uptime: 14 min
Scheduling Mode: FIFO
Completed Jobs: 13

▾ Event Timeline
☐ Enable zooming

showString at NativeMethodAccessorImpl.java:0 (Job 12)
Status: SUCCEEDED
Submitted: 2020/02/13 21:01:32
Completed: 2020/02/13 21:01:43

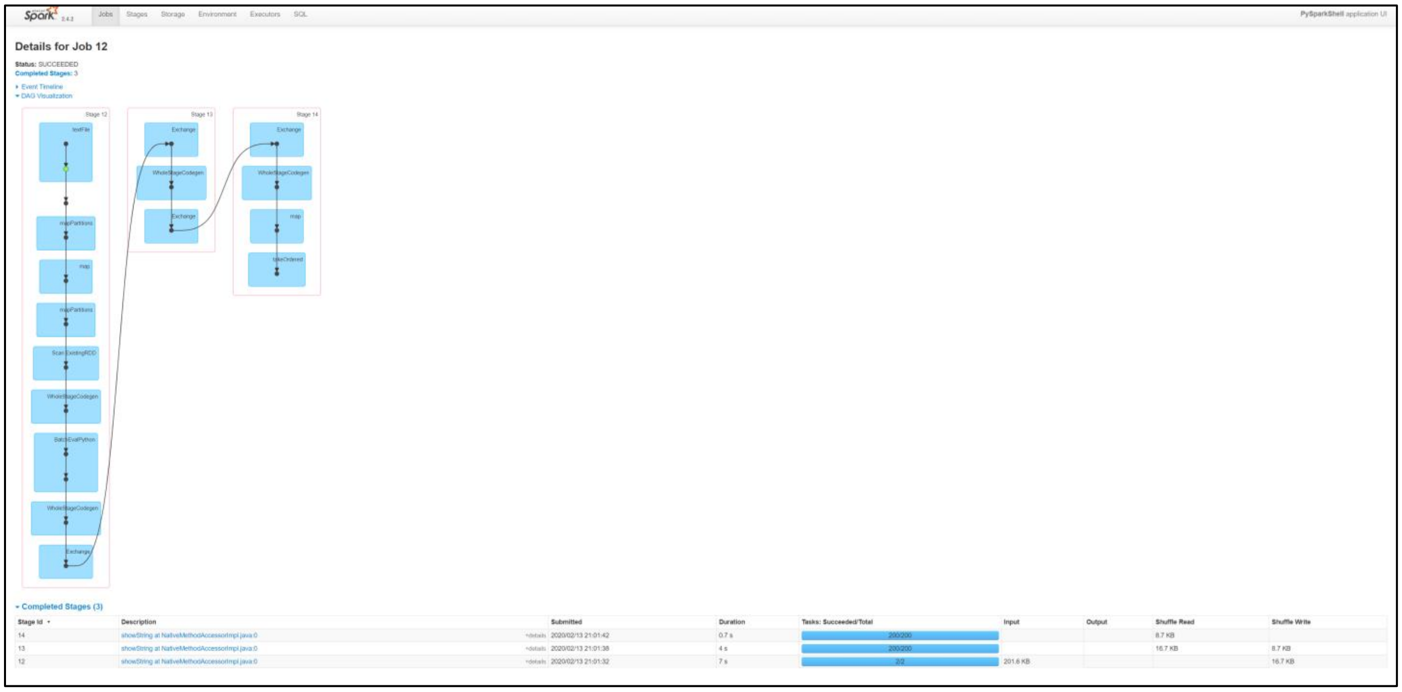
▾ **Completed Jobs (13)**

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
12	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/02/13 21:01:32	11 s	3/3	402/402
11	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/02/13 20:56:02	2 s	1/1	1/1
10	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/02/13 20:55:59	2 s	1/1	1/1
9	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/02/13 20:55:57	2 s	1/1	1/1
8	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/02/13 20:55:54	3 s	1/1	1/1
7	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2020/02/13 20:55:52	1 s	1/1	1/1
6	runJob at PythonRDD.scala:153 runJob at PythonRDD.scala:153	2020/02/13 20:55:45	1 s	1/1	1/1
5	runJob at PythonRDD.scala:153 runJob at PythonRDD.scala:153	2020/02/13 20:51:45	0.9 s	1/1	1/1
4	collect at <ipython-input-10-dafb49e3c2a5>:2 collect at <ipython-input-10-dafb49e3c2a5>:2	2020/02/13 20:50:44	2 s	1/1	2/2
3	runJob at PythonRDD.scala:153 runJob at PythonRDD.scala:153	2020/02/13 20:50:43	0.9 s	1/1	1/1
2	count at <ipython-input-8-1b5f4ab3d120>:3	2020/02/13 20:50:41	2 s	1/1	2/2

Figure 5 – Longer Jobs

Figure 5 shows the timeline and the last job executed by Spark, which in this case refers to the Spark SQL statement. Note that instead of 1 task we have 402 tasks to bring the final result. As it is a more extensive process, spark took 11 seconds to return the information, which is not much compared to the amount of data in the operation.

Within the job we can see that the DAG has more stages (stage 12, 13, 14), as shown in figure 6 below, and that they communicate, that is, the final result of one is the input of another and so on.



Details for Job 12

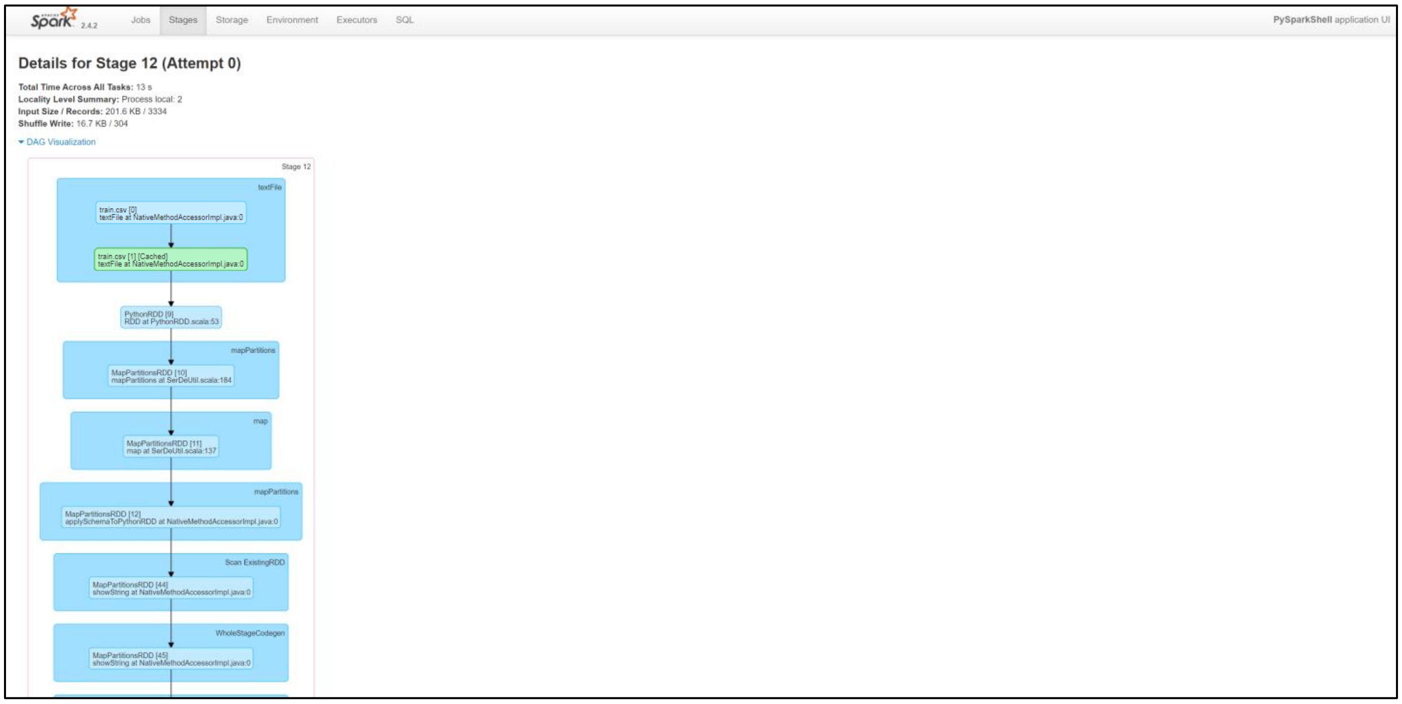
Status: SUCCEEDED
Completed Stages: 3

Completed Stages (3)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
14	showString at NativeMethodAccessorImpl.java:0	+details	2020/02/13 21:01:42	0.7 s	200/200			8.7 KB	
13	showString at NativeMethodAccessorImpl.java:0	+details	2020/02/13 21:01:38	4 s	200/200			16.7 KB	8.7 KB
12	showString at NativeMethodAccessorImpl.java:0	+details	2020/02/13 21:01:32	7 s	2/2	201.6 KB			16.7 KB

Figura 6 – DAG longer jobs

Pyspark shell provides an interactive analysis, that is, if you click on each stage it is possible to see in detail the execution of each RDD.

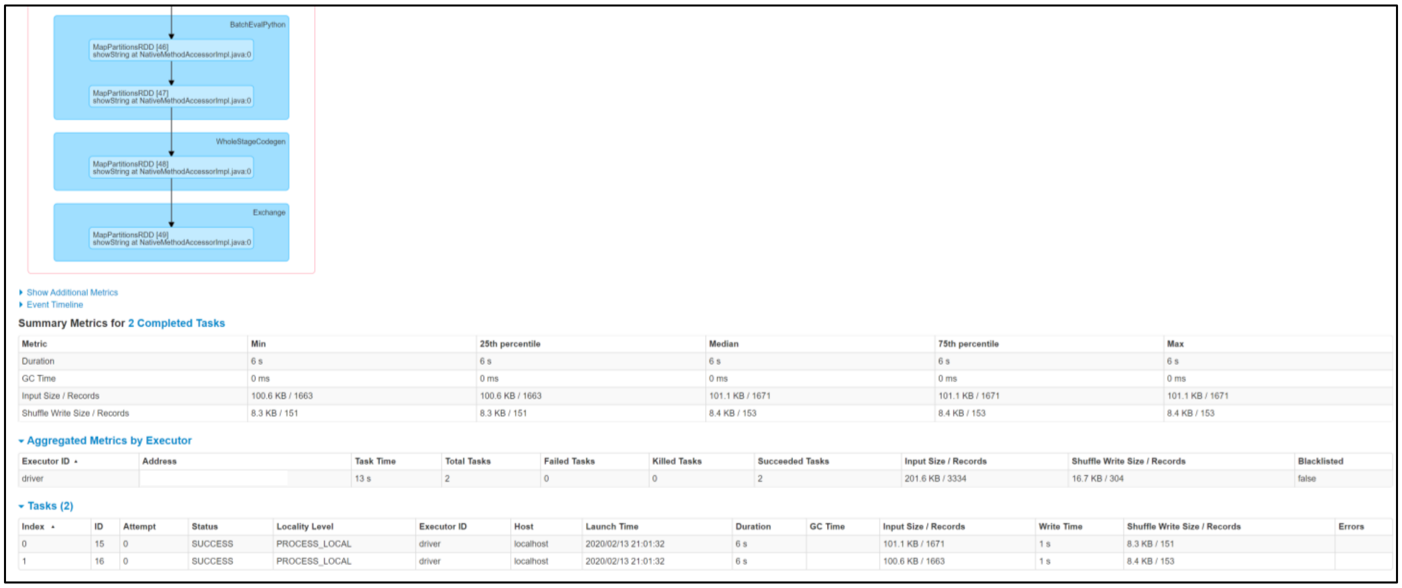

Spark 2.4.2
Jobs
Stages
Storage
Environment
Executors
SQL
PySparkShell application UI
DAG Visualization
Stage 12
textFile
train.csv [0] textFile at NativeMethodAccessorImpl.java:0
train.csv [1] [Cached] textFile at NativeMethodAccessorImpl.java:0
PythonRDD [9] RDD at PythonRDD.scala:53
mapPartitions
MapPartitionsRDD [10] mapPartitions at SerDeUtil.scala:184
map
MapPartitionsRDD [11] map at SerDeUtil.scala:137
mapPartitions
MapPartitionsRDD [12] applySchemaToPythonRDD at NativeMethodAccessorImpl.java:0
Scan ExistingRDD
MapPartitionsRDD [44] showString at NativeMethodAccessorImpl.java:0
WholeStageCodegen
MapPartitionsRDD [45] showString at NativeMethodAccessorImpl.java:0

Details for Stage 12 (Attempt 0)

Total Time Across All Tasks: 13 s
Locality Level Summary: Process local: 2
Input Size / Records: 201.6 KB / 3334
Shuffle Write: 16.7 KB / 304

Figure 7 - Stage 12

Continued in figure 8.


BatchEvalPython
MapPartitionsRDD [46]
showString at NativeMethodAccessorImpl.java:0
MapPartitionsRDD [47]
showString at NativeMethodAccessorImpl.java:0
WholeStageCodegen
MapPartitionsRDD [48]
showString at NativeMethodAccessorImpl.java:0
Exchange
MapPartitionsRDD [49]
showString at NativeMethodAccessorImpl.java:0
Show Additional Metrics
Event Timeline

Summary Metrics for 2 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	6 s	6 s	6 s	6 s	6 s
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Input Size / Records	100.6 KB / 1663	100.6 KB / 1663	101.1 KB / 1671	101.1 KB / 1671	101.1 KB / 1671
Shuffle Write Size / Records	8.3 KB / 151	8.3 KB / 151	8.4 KB / 153	8.4 KB / 153	8.4 KB / 153

Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Shuffle Write Size / Records	Blacklisted
driver		13 s	2	0	0	2	201.6 KB / 3334	16.7 KB / 304	false

Tasks (2)

Index	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	15	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2020/02/13 21:01:32	6 s		101.1 KB / 1671	1 s	8.3 KB / 151	
1	16	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2020/02/13 21:01:32	6 s		100.6 KB / 1663	1 s	8.4 KB / 153	

Figure 8 - Continuation Stage 12

Notice in figure 8 that the last instruction in the blue box is “takeOrdered” because what I wrote in my code was to order the final result in descending order, presenting on screen an easier way to view the result.

I believe that you have already understood the importance of Pyspark Shell for the Data Science process, with it I can carry out the execution of the code following how each instruction requested to the cluster was managed and in the event of a possible error it is faster to find where there are failures, analyze the problems and solve them to continue the project.

Unfortunately, it is not possible to deepen the studies during the exploratory analysis because all data are anonymous and do not make objective references to the values that we can interpret. The only known item is the “target” variable, which will be explored in the next chapter.

During exploratory analysis we work with SQL statements, but we will now enter an important library responsible for machine learning models, Spark MLlib[23].

[23] https://spark.apache.org/mllib/

8. Building the Machine Learning Model

In this chapter, we will build two different machine learning models in order to analyze which one we get the most accuracy from and then present new and unknown data and then evaluate if we have a reliable model.

Machine Learning is an area of artificial intelligence that studies learning techniques by applying programming and computing to build mathematical models with the ability to get knowledge automatically based on the data provided.

Through this data, the trained algorithm is able to make decisions when new data is presented to it, thus providing complex problem solving solutions that in many cases would be difficult for a person to perform.

Within this concept we have 2 types of learning:

- Supervised Learning
  In this type of learning, we have a set of input data and possible output data that should be used to train the model. In other words, by testing this model we can compare the predicted output data with that provided and evaluate the accuracy of the trained model. If unsatisfactory, it is possible to go back to the beginning of model creation and improve the predictor variables to perform new tests.

- Unsupervised Learning
  In this other type of learning, we have an input data set but the outputs are unknown, so I can't compare the predicted data with the expected outputs. For this learning modality other techniques are applied, for example, the data will be grouped and the results will change according to the variables.

Machine Learning can be used to solve a range of problems from internet search suggestions, spam message tracking, to digital marketing usage.

For our business problem, forecasting Santander Bank Customers' Satisfaction in Real Time, we will use supervised learning, but before we go into creating the predictive model, let's see how data is balanced.

8.1Data Balancing

After loading the variables I performed an analysis on the dataset and obtained an interesting result: data distribution of the predictor variable (target) is totally unbalanced. This can be a big problem because it generates a biased model prioritizing one variable over another.

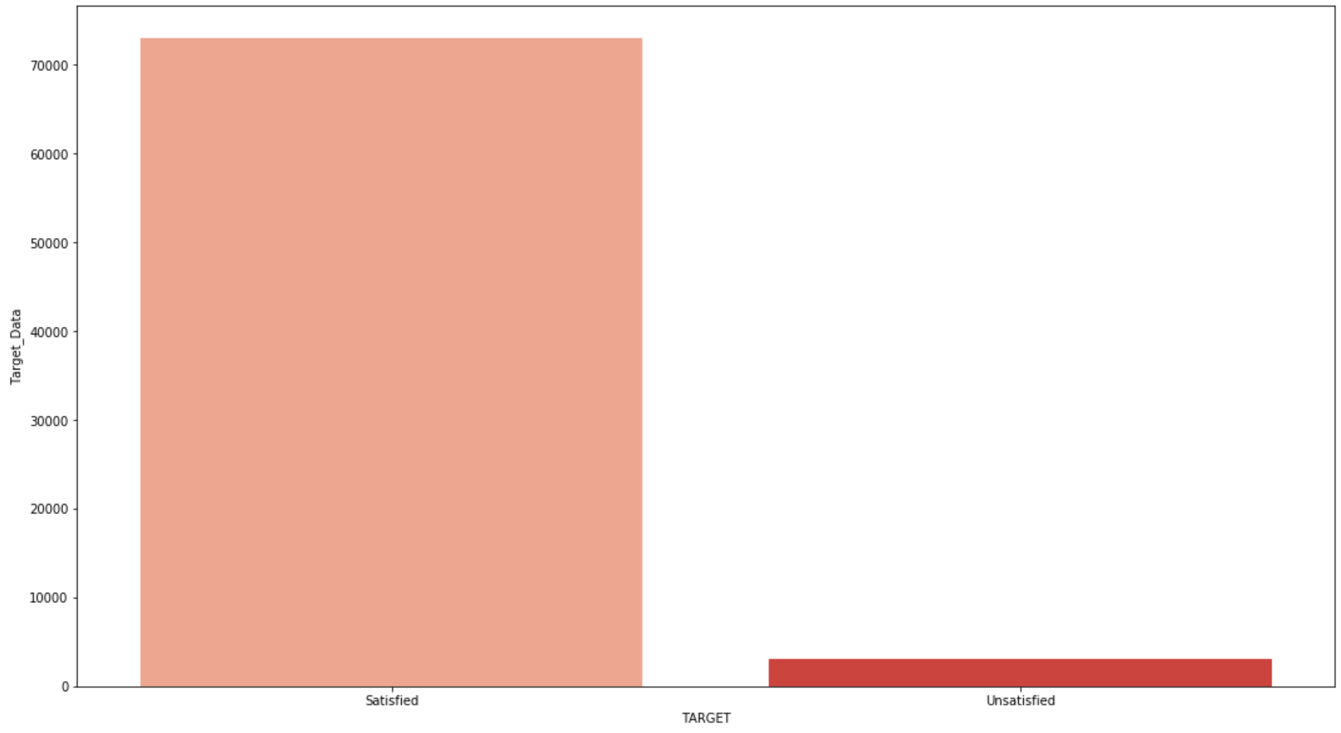


Figure 9 – Amount of Data (%)

We wold have no problem creating the predictive model but considering a real environment I recommend to acquire balanced data avoiding bias when creating the machine learning model.

Considering this problem, we will see the effect of having balanced data in a simulated environment with approximately 50% "Satisfied" and 50% "Unsatisfied", a machine learning model will be created, testing this model, assessing accuracy, presenting new data and finally plotting the confusion matrix.

However, it is known that we have many variables and machine learning models may not be able to convert results, but how can we choose the best variables for our model if they are all anonymous data? Let's look at this answer through the data by performing the correlation between the variables.

8.2 Correlation and Variables of Importance

Studying the correlation between variables is an important source for understanding a problem and finding possible solutions. Finding the relevant variables can help improve the predictive model and bring valuable sources of information to the analysis process.

The correlation coefficient varies from -1 to 1 indicating that two variables have a strong negative or positive correlation, respectively.

With the aid of a "loop for" I ran through my dataset comparing each variable to the Target variable using the following command:

```
for i in santanderDF_1.columns:
    # for each one I'll 'select(i)', and use the 'take' action from [0][0] combination as if it was
      a matrix
    if not(isinstance(santanderDF_1.select(i).take(1)[0][0], str)):
        print("Correlation between Target with: ", i, santanderDF_1.stat.corr('TARGET', i))
```

Resulting in:

```
Correlation between Target with:  TARGET 1.0
Correlation between Target with:  VAR1 0.004474798175536685
Correlation between Target with:  VAR10 0.0030872912147492
Correlation between Target with:  VAR100 -0.034431709419395266
Correlation between Target with:  VAR101 -0.01771812356515599
Correlation between Target with:  VAR102 -0.017694370368032397
Correlation between Target with:  VAR103 -0.0010411146833006317
...
Correlation between Target with:  VAR104 -0.0010411146833006317
Correlation between Target with:  VAR105 -0.038399938657465146
Correlation between Target with:  VAR106 -0.003607583627587863
Correlation between Target with:  VAR107 -0.007387144633475057
Correlation between Target with:  VAR108 -0.002183710218270266
Correlation between Target with:  VAR109 -0.0013340101850203155
Correlation between Target with:  VAR11 0.010082404008577805
Correlation between Target with:  VAR110 -0.0010411146833006377
```

Tabela 3 - Correlação e Variáveis de Importância

There are many variables as mentioned above, to improve the visualization of the result we will plot the correlation matrices:

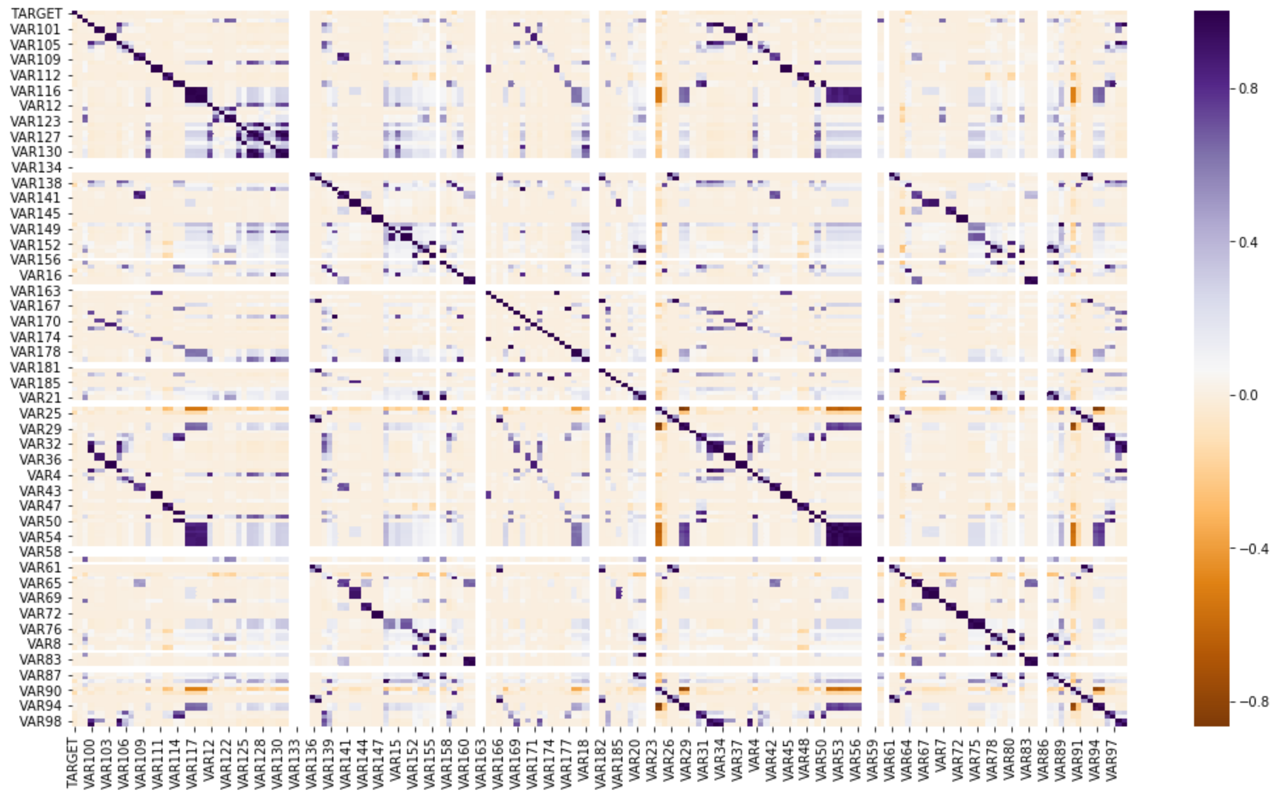


Figure 10 – Correlation Matrix I

I performed the same process for the other half of the dataset, resulting in:

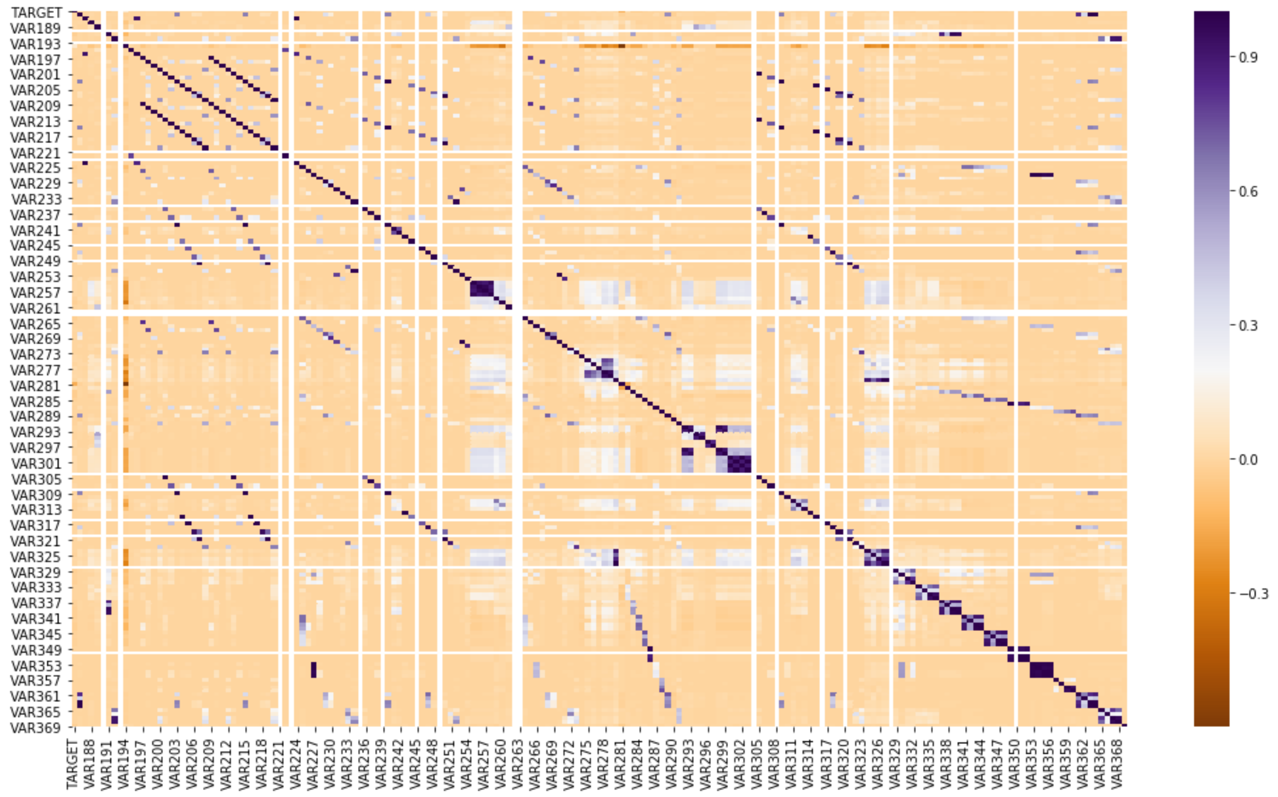


Figure 11 - Correlation Matrix II

Figures 10 and 11 represent the degree of correlation between the variables, that is, the interdependence between them. Variables with strong correlation provide better results when offered to the model.

Although it is possible to have a notion of which variables would perform better in the model, it is still difficult to say due to the number of items we have, so I will use the following criterion: Variables with a value of more than 0.03 of correlation with target variable will be the selected importance variables.

In this way I select:

- 24 variables with correlation equal to 0.03...
- 2 variables with correlation equal to 0.04...
- 1 variables with correlation equal to 0.07...
- 1 variables with correlation equal to 0.08...
- 8 variables with correlation equal to 0.10...

Totaling 36 variables to deliver to the machine learning model.

Using a new function for transformation (presented in the source code at the end of the project) I loaded the data into the respective variables, being able to carry out a new balancing of the dataset.

8.3 New Data Balancing and Correlation Matrix

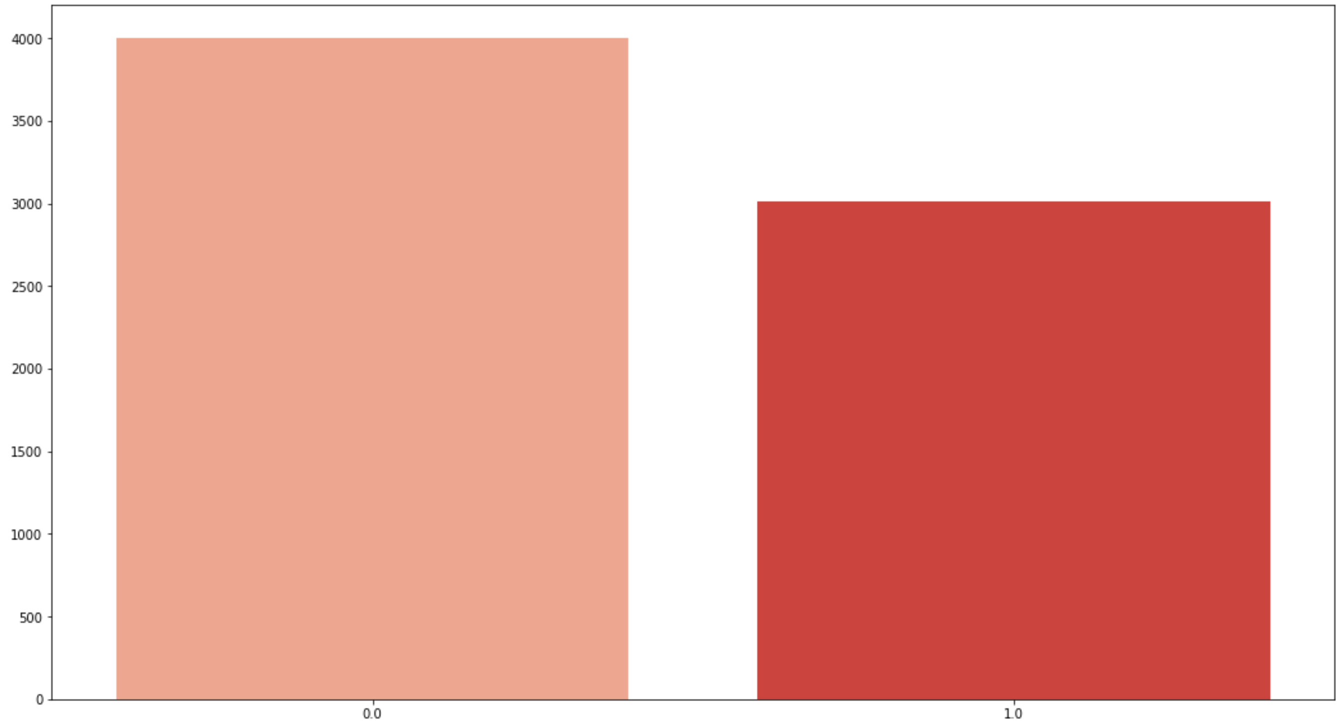


Figure 12 - New Data Balance

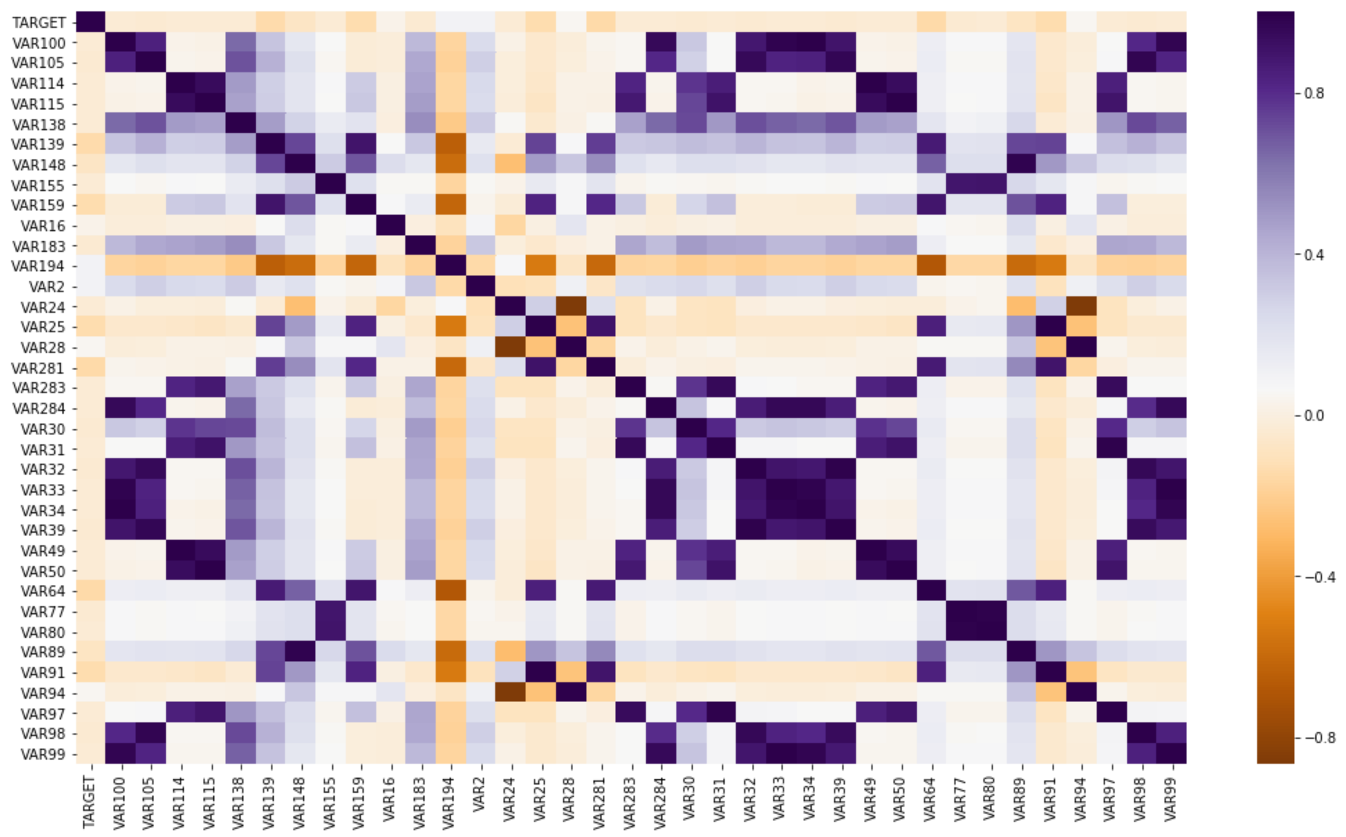


Figure 13 - Correlation Matrix III

Notice in figure 13 that we have a good part of the variables in violet color or close to it indicating a strong positive correlation between them, different from the previous correlation matrices where we had few variables in this configuration.

We now have a leaner correlation matrix and with the variables ready to start the process of delivering data to the machine learning model.

These correlation operations generated a series of different jobs to the cluster, each costing an average of 2 seconds to be processed, a job from the “take” action where we acquire the data for delivery to the second “corr” job that makes the correlation between the variables . Everything spelled out in Pyspark Shell.

8.4 Pre-Processing of the Dataset

To load these most important variables into the machine learning model, it is necessary to first understand an important concept of data delivery to Spark MLlib (MLlib is the abbreviation for Machine Learning Library), the concept of Dense Vector and Sparse Vector.

The need for some algorithms (mainly regression) for apache spark machine learning is that the data must be in a specific format to be able to train the algorithm.

For that, I need to pass the data in vector format, specifically using a dense or sparse vector, as shown in the image below, because Apache spark works with a cluster and then it will distribute the data between the machines.

Dense Vector:	9	4	0	0	0	2	0	0	9

Sparse Vector:	size:	9			
	indices:	0	1	5	8
	values:	9	4	2	9

Table 4 - Vector Dense x Vector Sparse

Conceptually they are the same object, just a vector. However sparse vectors are vectors that have many values like zero. While a dense vector is when most of the values in the vector are nonzero.

To build the vector I need to import the following library:

```
from pyspark.ml.linalg import Vectors
```

With the function "Vectors" I created the following function using the importance variables previously selected and the dense vector:

```
def transformFeatures(row):
    obj = (row['TARGET'], Vectors.dense(row['VAR100'], row['VAR105'], row['VAR114'
      ], row['VAR115'], row['VAR138'], row['VAR155'], row['VAR16'], row['VAR183'],
      row['VAR24'], row['VAR30'], row['VAR31'], row['VAR32'], row['VAR33'], row['V
      AR34'], row['VAR39'], row['VAR49'], row['VAR50'], row['VAR77'], row['VAR80']
      , row['VAR97'], row['VAR98'], row['VAR99'], row['VAR283'], row['VAR284'], ro
      w['VAR28'], row['VAR94'], row['VAR148'], row['VAR89'], row['VAR139'], row['V
      AR159'], row['VAR2'], row['VAR25'], row['VAR64'], row['VAR91'], row['VAR194'
      ], row['VAR281']) )

    return obj
```

Resulting in the following structure, the first column being the Target value:

```
[(0.0, DenseVector([0.0, 0.0, 3.0, 3.0, 6.0, 3.0, 0.0, 93041.85, 1.0, 1.0,
1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 3.0, 0.0, 0.0, 3.0, 0.0, 0.0,
0.0, 3.0, 1.0, 3.0, 3.0, 84.0, 0.0, 1.0, 0.0, 1.0, 0.0])),
 (0.0, DenseVector([0.0, 0.0, 0.0, 0.0, 3.0, 3.0, 0.0, 3.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
3.0, 1.0, 3.0, 3.0, 24.0, 1.0, 1.0, 3.0, 3.0, 3.0]))]
```

Best viewed in table format:

```
+-------+--------------------+
|TARGET |            FEATURES|
+-------+--------------------+
|    0.0| [0.0,0.0,3.0,3.0,...|
|    0.0| [0.0,0.0,0.0,0.0,...|
|    0.0| [0.0,0.0,0.0,0.0,...|
|    0.0| [0.0,0.0,0.0,0.0,...|
|    0.0| [0.0,0.0,0.0,0.0,...|
|    0.0| [0.0,0.0,0.0,0.0,...|
|    0.0| [0.0,0.0,0.0,0.0,...|
|    0.0| [0.0,0.0,0.0,0.0,...|
|    0.0| [0.0,0.0,0.0,0.0,...|
|    0.0| [0.0,0.0,0.0,0.0,...|
+-------+--------------------+
```

Table 5 - Target x Features

Now we are ready for Spark Machine Learning processes.

8.5 Machine Learning

8.6 Train/Test Split

Although the data is ready for delivery to the machine learning model, I will separate the dataset into training and test data following the proportion of 70% and 30% respectively.

Initially I will use the training data so that the model performs its mathematical transformation by learning patterns and delivering a final model, after this process I analyze with the test data if it has acceptable accuracy. Remembering that the test data is different and unknown to the model.

```
+------+----------+        +------+----------+
|TARGET|Qt of Data|        |TARGET|Qt of Data|
+------+----------+        +------+----------+
|  null|      4892|        |  null|      2116|
|   0.0|      2803|        |   0.0|      1197|
|   1.0|      2089|        |   1.0|       919|
+------+----------+        +------+----------+
```

Table 6 - train x test

8.7Chosen Machine Learning Models

Two classification models were chosen for training and evaluation:

- DecisionTreeClassifier;
- RandomForestClassifier.

Let's study how these machine learning models work.

8.7.1 DecisionTreeClassifier X RandomForestClassifier

With the selected variables we can train the predictive model, but a key point is to try to identify when the model enters the underfitting zone, when it encounters the smallest error (ideal value) and when it arrives in the overfitting zone. However, first let's understand some concepts about Decision Tree and randomForest.

The machine learning model chosen for our classification problem will be defined by the best accuracy provided by the model after presented test data. As the name suggests, randomForest means Random Forest, which in Data Science we can make analogy to Decision Trees where each tree has a depth and decides between its 'leaves' which is the best path to travel.

Imagine an inverted tree:

Figura 14 - DecisionTree

End nodes (or leaves) are at the bottom of the decision tree. This means that the decision trees are drawn upside down. Thus, the leaves are the bottom and the roots are the tops (figure above).

A Decision Tree works with both categorical and continuous variables and works by dividing the population (or sample) into subpopulations (two or more sets) based on the most significant divisors of the input variables. For this and many other reasons, decision trees are used in classification and regression problems where the supervised learning algorithm has a predefined target variable.

8.7.2 RandomForest Predictive Model x Underfitting x Overfitting

In randomForest, or Random Forest, we grow multiple trees instead of a single tree. But how does the classification process work? Initially for classifying a new attribute-based object, a tree generates a classification for that object (which is as if the tree gives votes for this class). This process goes on for each tree in the forest and finally, the forest chooses the classification with the most votes (from all trees in the forest). In case of regression, the average of the exits by different trees is considered.

To illustrate the process performed by randomForest, follow figure below:

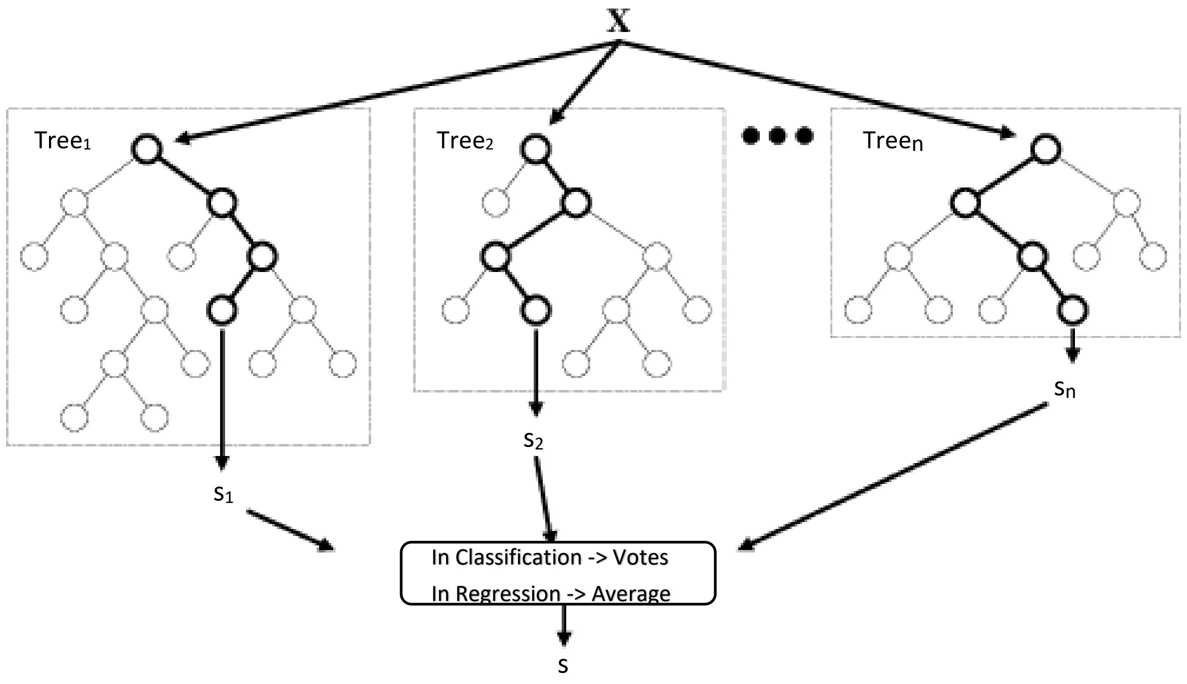


Figura 15 – RandomForest

As shown in figure 15, we can have n trees and each tree can have as many leaves as it wants. This is where we have a problem, because a shallow tree that has been trained to classify an object may not be accurate because it has learned little, or in other words underfitting. At the other extreme we have overfitting, that is, if no limit is set the model will give 100% accuracy in the training set because it ends up making a leaf for each observation.

I imagine the question now would be, "But isn't offering 100% accuracy good? " The answer is yes and no, because it is good to have accuracy, but here the accuracy is only in the training data and my goal is to generate an unbiased machine learning model where any data can be predicted. In case of overfitting when I present new data (which is the test data) the model will fail and return poor accuracy.

The following image illustrates the problem:

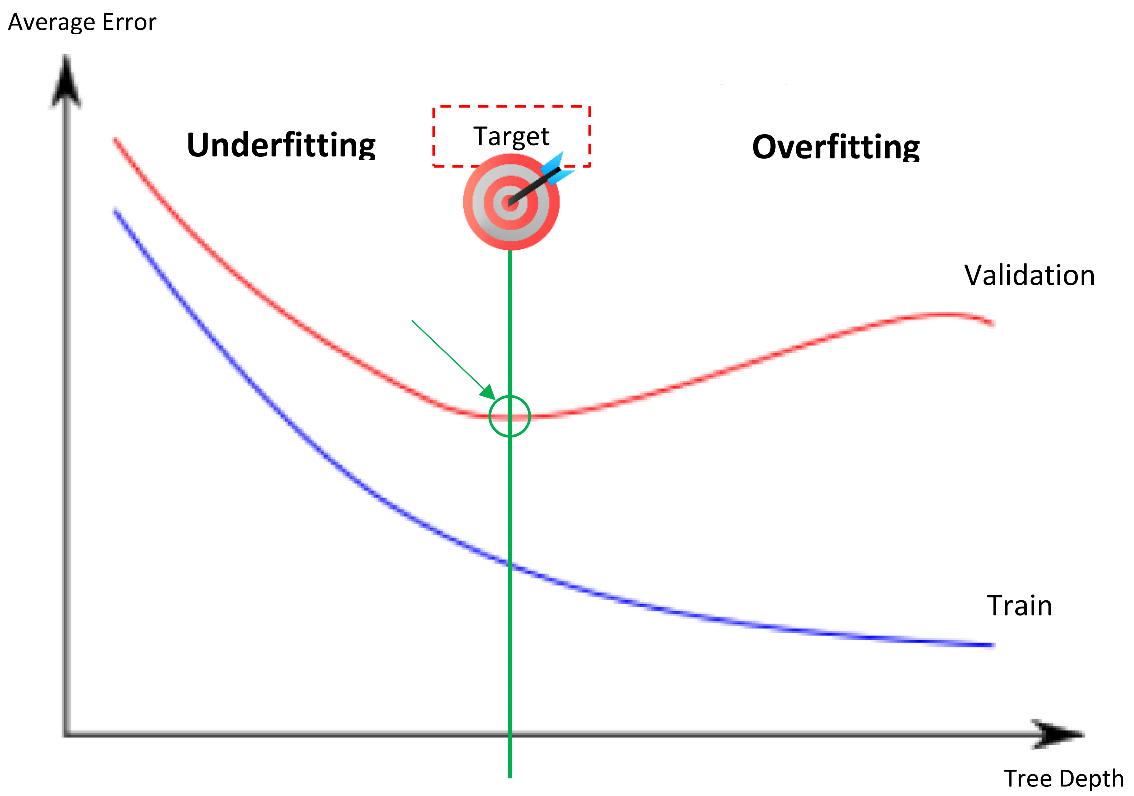


Figura 16 - Underfitting x Overfitting

In figure 16 we have 3 lines with different colors and each one represents important information:

- Blue Line: represents the average error that training data gives according to tree depth. The deeper the tree, the smaller the error as the training data was learned almost entirely.
- Red Line: represents the error in the test data (validation). Note that the error starts high, decreases, and then increases again. This is one of the key challenges faced when modeling decision trees, finding the optimal point where the error is as small as possible.
- Green Line: As you can see, the green line crosses at the ideal point, where the error in the test data (validation) is as little as possible, giving the model better accuracy.

Consolidating in the following figure the goal in performing predictive modeling controlling underfitting and overfitting:

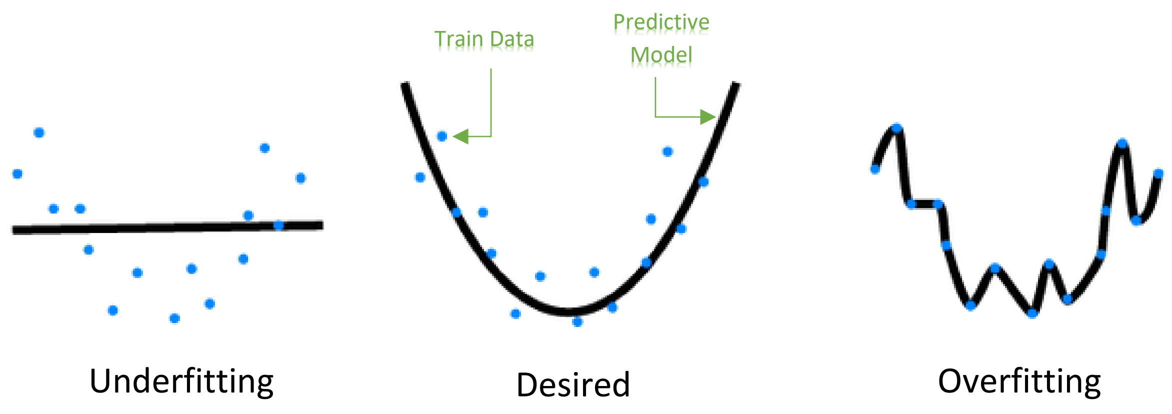


Figura 17 - Underfitting x Ideal x Overfitting

As shown above, what we want is that the model has an ideal curve avoiding poor accuracy, but also does not memorize 100% of the training data failing with new and unknown data.

8.8Predictive model (DecisionTree) and the business problem

Back to the business problem of this project, so that the model presented does not suffer from underfitting or overfitting, a loop was created to train the different depths of the tree.

The line of code below will carry out the complete training, create the model, carry out the forecast and finally return the precision at each depth. This process will be repeated n times, where n indicates the depth range of the tree (1 - 15).

Follow code:

```
# Avoiding Underfitting and Overfitting in Decision Trees
for depth in range(1,15):
    model_v1 = DecisionTreeClassifier(maxDepth=depth, labelCol= 'TARGET', features
    Col= 'FEATURES')
    model_v1_train = model_v1.fit(train_data)
    model_v1_prediction = model_v1_train.transform(test_data)
    evaluator = MulticlassClassificationEvaluator(predictionCol= 'prediction',
                                                  labelCol = 'TARGET',
                                                  metricName = 'accuracy')
    acc = evaluator.evaluate(model_v1_prediction)
    print('In Depth %.0f this model is %.4f%% Accurate.' %(depth, acc*100))
```

Resulting in:

```
In Depth 1 this model is 66.1626% Accurate.
In Depth 2 this model is 75.0473% Accurate.
In Depth 3 this model is 75.0473% Accurate.
In Depth 4 this model is 76.7958% Accurate.
In Depth 5 this model is 76.7013% Accurate.
In Depth 6 this model is 75.8979% Accurate.
In Depth 7 this model is 76.5595% Accurate.
In Depth 8 this model is 75.8507% Accurate.
In Depth 9 this model is 76.0397% Accurate.
In Depth 10 this model is 74.9527% Accurate.
In Depth 11 this model is 75.0945% Accurate.
In Depth 12 this model is 74.4329% Accurate.
In Depth 13 this model is 74.1493% Accurate.
In Depth 14 this model is 72.6371% Accurate.
```

According to the model's response, depth equal to 4 offers better accuracy and avoids underfitting and overfitting.

Model 01 (DecisionTree) -> Precision of 76.70%

We will use this depth to continue carrying out the forecasts and evaluation.

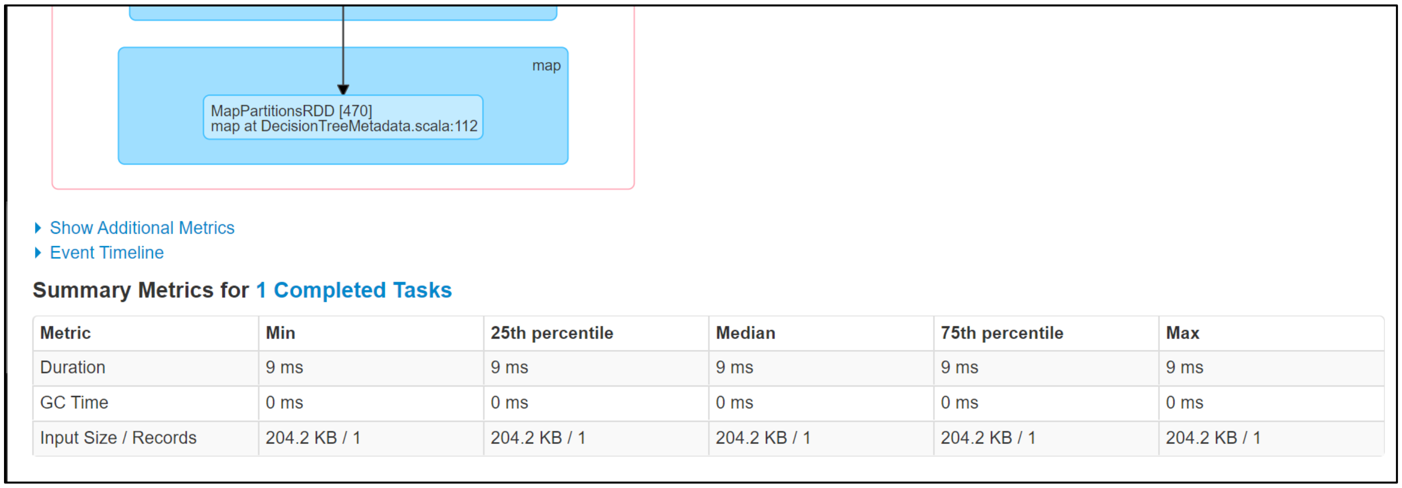
8.8.1 Forecasting and Evaluating

With the model parameterized to offer the best accuracy, I presented unknown data to predict whether a customer is satisfied or unsatisfied with the service. Here is a part of this prediction made by the machine learning model:

```
[Row(prediction=0.0, TARGET=0.0),
 Row(prediction=0.0, TARGET=0.0),
 Row(prediction=0.0, TARGET=0.0),
 Row(prediction=0.0, TARGET=0.0),
 Row(prediction=0.0, TARGET=0.0),
 Row(prediction=1.0, TARGET=0.0),
 Row(prediction=1.0, TARGET=0.0),
 Row(prediction=1.0, TARGET=0.0),
 Row(prediction=1.0, TARGET=0.0),
 Row(prediction=1.0, TARGET=0.0),
 Row(prediction=1.0, TARGET=0.0),
 Row(prediction=0.0, TARGET=0.0),
 Row(prediction=0.0, TARGET=0.0)]
```

Note that at times the model should predict "0" and predicted "1" indicating that the customer was unsatisfied when in fact he was satisfied. Let's build a Confusion Matrix and analyze all predictions.

In this case, Pyspark Shell presented more than 190 operations that were very fast in execution, lasting around 50 ms each. Here we can see how advantageous it is to bring spark to a big data project, as it performed machine learning processes in less than a second, that is, it created the model, delivered the trained model, performed tests with new data and presented the evaluation metric so that neither underfitting nor overfitting occurs. All in a few milliseconds of execution on all data in the cluster.



Summary Metrics for 1 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	9 ms	9 ms	9 ms	9 ms	9 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Input Size / Records	204.2 KB / 1	204.2 KB / 1	204.2 KB / 1	204.2 KB / 1	204.2 KB / 1

Figure 18 - DecisionTree Job

In figure 18 we can see that the last action of this job is the mapping in the machine learning model DecisionTree. The job was started and then ended 9 milliseconds later.

8.8.2 Confusion Matrix

Confusion Matrix is one of many performance gauges for evaluating data predicted by a machine learning model. We already know that the accuracy is close to 77% but what about the data that was not classified correctly? A Confusion Matrix can tell us what happened to all classified data, following image for a better understanding of its operation:

	0 (Predicted Data)	1 (Predicted Data)
0 (Original Data)	TruePositive	FalseNegative
1 (Original Data)	FalsePositive	TrueNegative

Figure 19 - Confusion Matrix Concept

Let's interpret what the Confusion Matrix is telling us:

- TruePositive(TP): It means that my model predicted from the data provided that the class would be 0 and it really is correct, it was supposed to be 0.
- FalseNegative(FN): It means that my model predicted from the data provided that the class would be 1 but was supposed to be 0.
- FalsePositive(FP): It means that my model predicted from the data provided that the class would be 0 but also missed because it was supposed to be 1.
- TrueNegative(TN): It means that my model predicted from the data provided that the class would be 1 and this time it was right because it was supposed to be 1.

Our goal is to predict and succeed so we seek to maximize the values present in TP and TN, and on the other hand mitigate values of FN and FP.

Our machine learning model returned the following results:

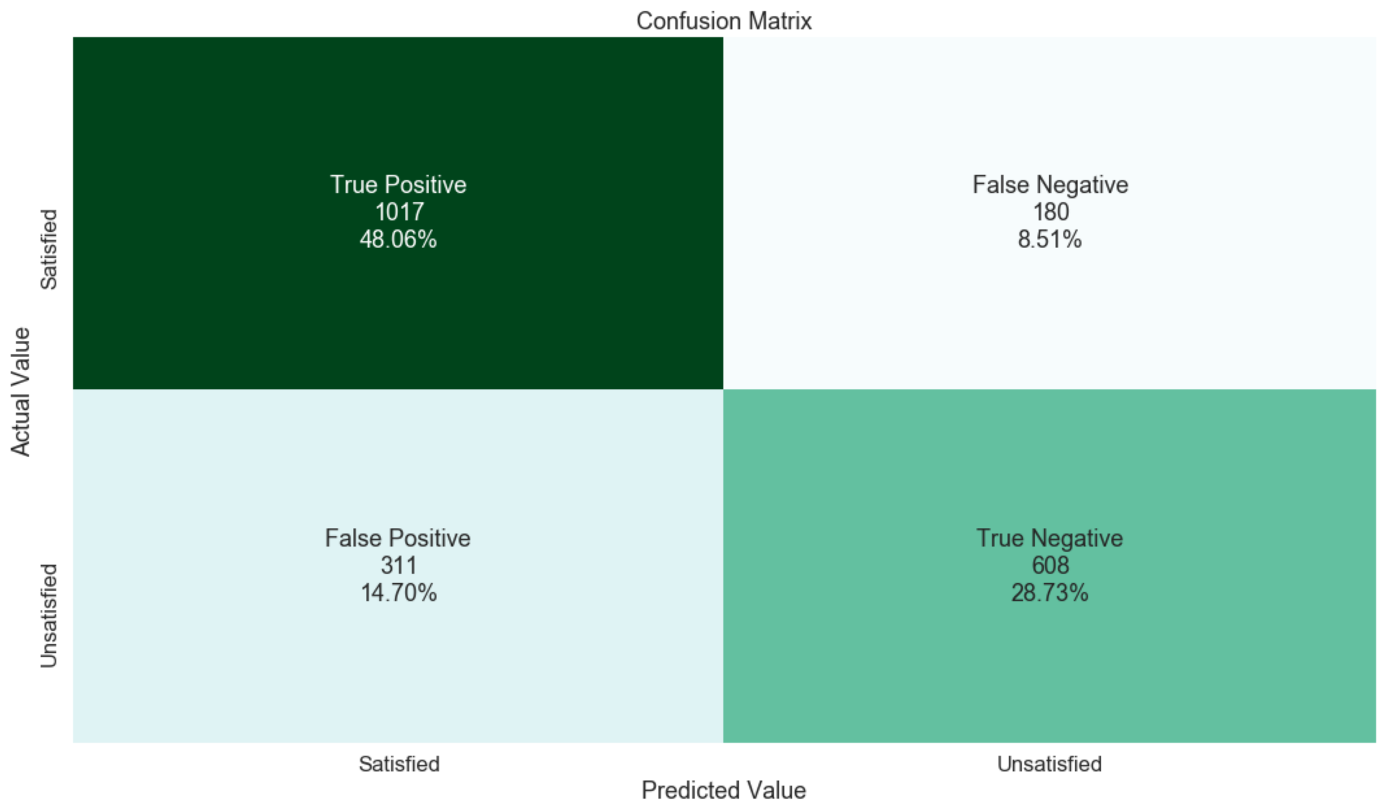


Figure 20 - Confusion Matrix I

Great results because TP and TN are the highest values among all the predicted data. It is worth mentioning that TP is much higher since, as explained in chapter 8 (item 8.1), we have unbalanced data and prone to "Satisfied".

8.9 Predictive model (RandomForest) and the business problem

Using the same concept and processes as the previous model, that is, complete training, create the model, perform the forecast and finally return the precision we obtain:

Model 02 (RandomForest) -> Precision of 75.33%

8.9.1 Forecasting and Evaluating

```
[Row(prediction=0.0, TARGET=0.0),
 Row(prediction=0.0, TARGET=0.0),
 Row(prediction=0.0, TARGET=0.0),
 Row(prediction=0.0, TARGET=0.0),
 Row(prediction=1.0, TARGET=0.0),
 Row(prediction=1.0, TARGET=0.0),
 Row(prediction=1.0, TARGET=0.0)]
```

8.9.2 Confusion Matrix

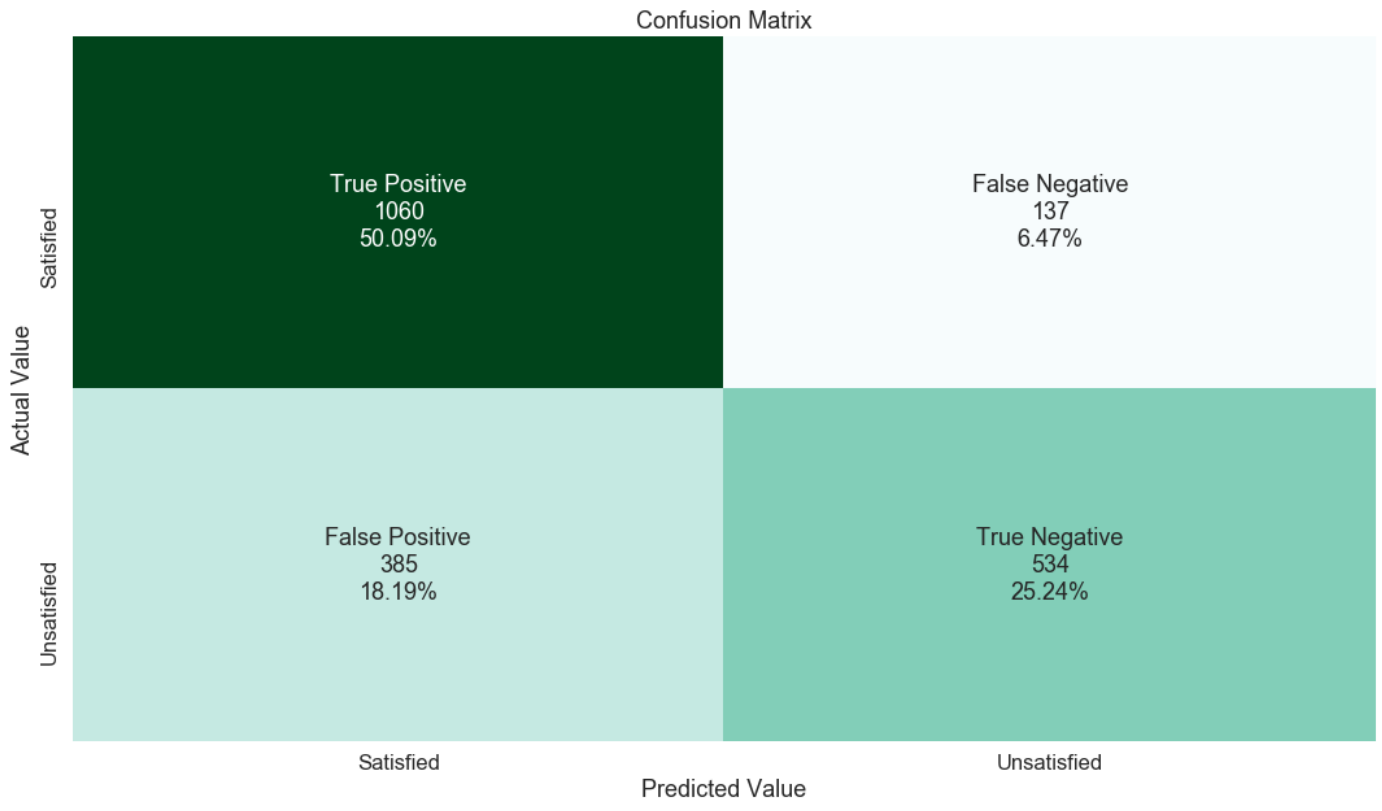


Figure 21 - Confusion Matrix II

Based on the results of the two models, we can choose the one that presented the best performance. The first model had better accuracy and delivered better True Negative, better predicting unsatisfied users, while the second model was better with True Positive, better predicting satisfied users.

As we want to predict the level of unsatisfaction and act on top of customers likely to leave the bank, our choice would be the first model, the Decision Tree model.

8.10 Saving Chosen Model

```
model_v1_train.write().overwrite().save('santander_ML_Model.model')
```

So, we have a model trained and ready to receive new, unknown data to predict the level of satisfaction of Santander customers, how to present this information to decision makers?

I chose to build a dashboard in PowerBi simulating a real environment with data collected from Monday to Friday in a week.

9. Simulated Environment and Results Visualization

After selecting the best machine learning model, I simulated a real environment with the acquisition of unknown data batches for an entire week (Monday to Friday) presenting the results on the PowerBI visualization platform with interactive graphics highlighting unsatisfied consumers and generating action opportunities for decision makers.

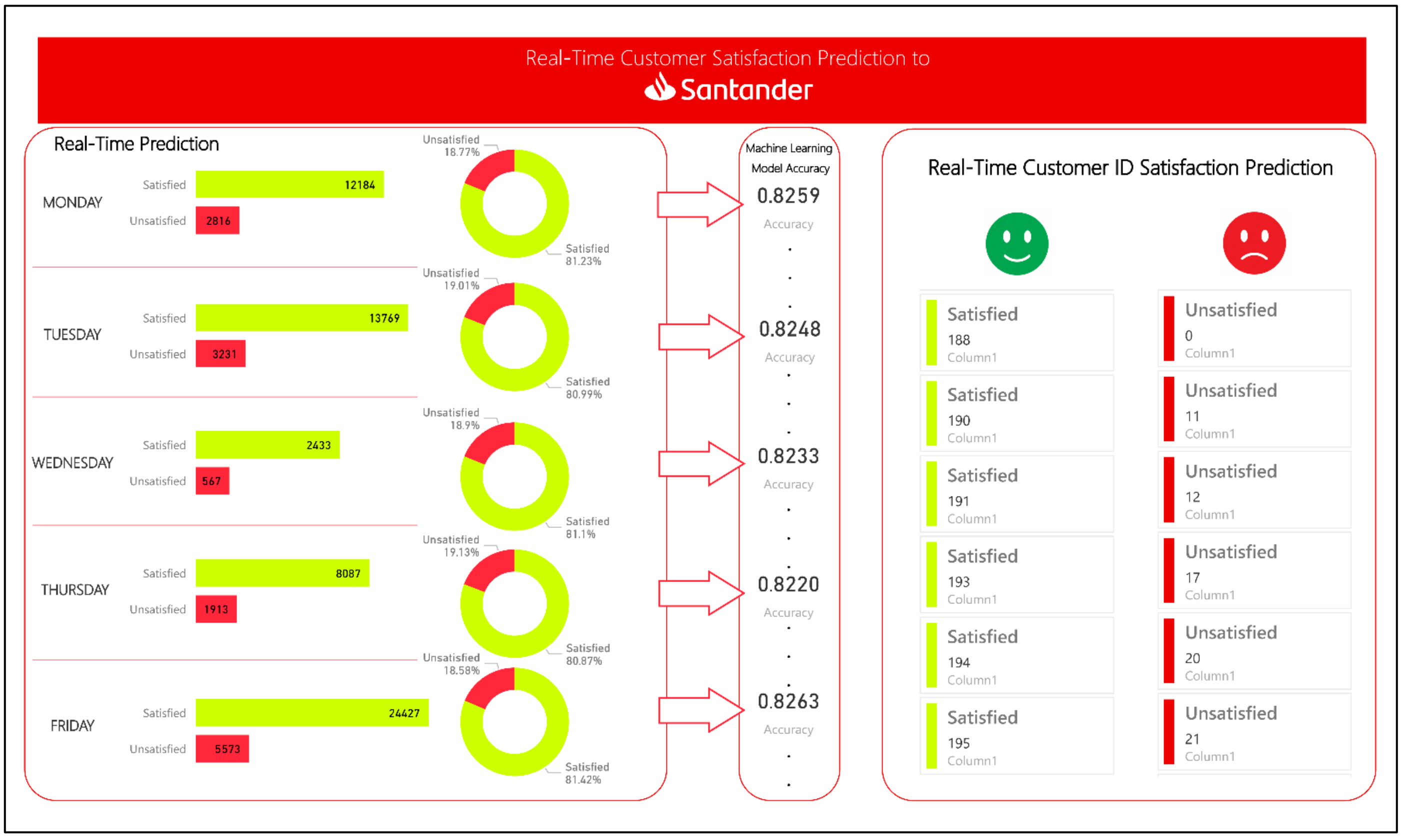
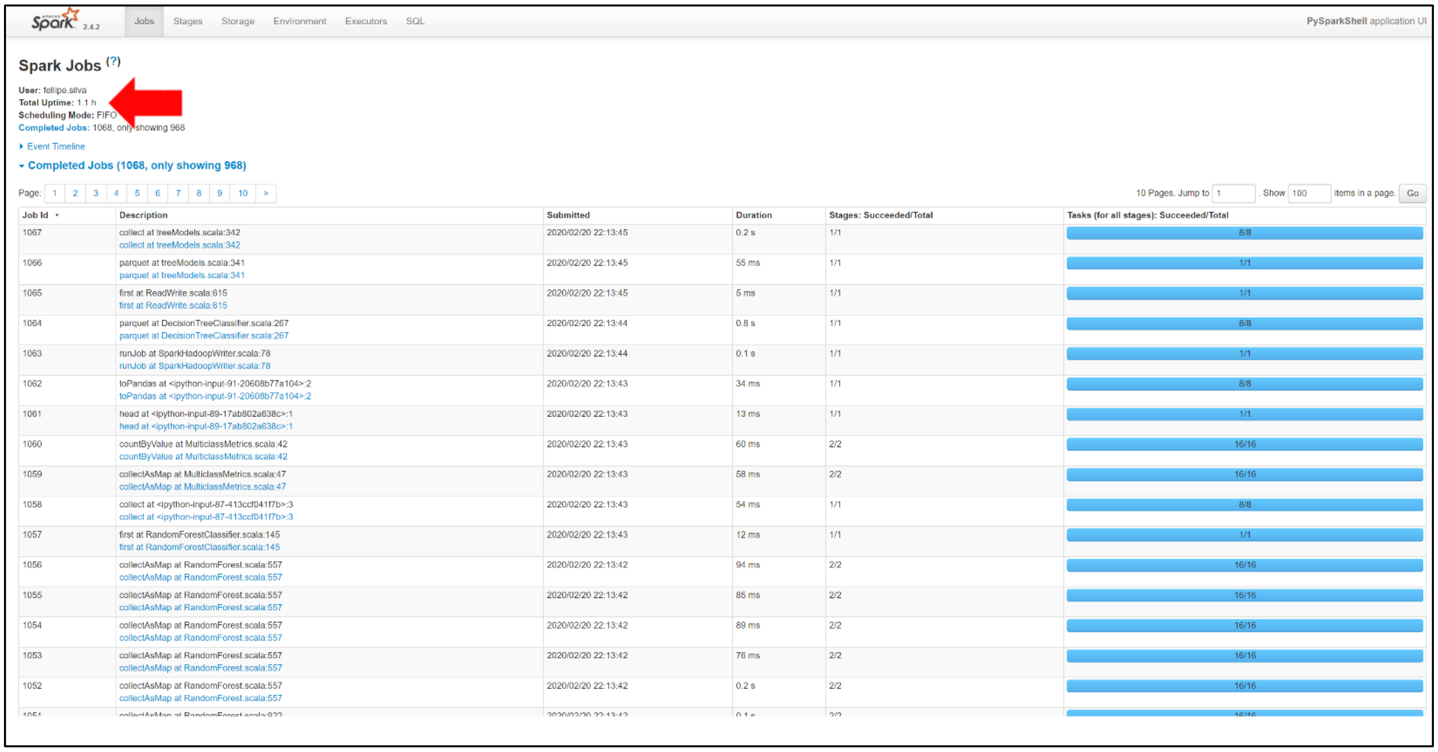


Figure 22 - Dashboard Santander

This dashboard illustrates an environment with real-time data collection for analysis where we have the following information:

- Real – Time Prediction: Real-time forecast by day of the week with donut chart indicating the percentage of customer satisfaction level.
- Machine – Learning Model Accuracy: Reliability indicator of the machine learning model based on predictions made.
- Real – Time Customer ID Satisfaction Prediction: Detailed real-time forecast by consumer.

With the above dashboard, the team responsible for customer satisfaction has relevant information to outline the best strategies for improving the services provided, progressing in metrics and achieving results through a large amount of data.



Spark Jobs (?)

User: fellipe.silva
Total Uptime: 1.1 h
Scheduling Mode: FIFO
Completed Jobs: 1068, only showing 968

▸ Event Timeline

▾ **Completed Jobs (1068, only showing 968)**

Page: 1 2 3 4 5 6 7 8 9 10 >   10 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1067	collect at treeModels.scala:342 collect at treeModels.scala:342	2020/02/20 22:13:45	0.2 s	1/1	8/8
1066	parquet at treeModels.scala:341 parquet at treeModels.scala:341	2020/02/20 22:13:45	55 ms	1/1	1/1
1065	first at ReadWrite.scala:615 first at ReadWrite.scala:615	2020/02/20 22:13:45	5 ms	1/1	1/1
1064	parquet at DecisionTreeClassifier.scala:267 parquet at DecisionTreeClassifier.scala:267	2020/02/20 22:13:44	0.8 s	1/1	8/8
1063	runJob at SparkHadoopWriter.scala:78 runJob at SparkHadoopWriter.scala:78	2020/02/20 22:13:44	0.1 s	1/1	1/1
1062	toPandas at <ipython-input-91-20608b77a104>:2 toPandas at <ipython-input-91-20608b77a104>:2	2020/02/20 22:13:43	34 ms	1/1	8/8
1061	head at <ipython-input-89-17ab802a638c>:1 head at <ipython-input-89-17ab802a638c>:1	2020/02/20 22:13:43	13 ms	1/1	1/1
1060	countByValue at MulticlassMetrics.scala:42 countByValue at MulticlassMetrics.scala:42	2020/02/20 22:13:43	60 ms	2/2	16/16
1059	collectAsMap at MulticlassMetrics.scala:47 collectAsMap at MulticlassMetrics.scala:47	2020/02/20 22:13:43	58 ms	2/2	16/16
1058	collect at <ipython-input-87-413ccf041f7b>:3 collect at <ipython-input-87-413ccf041f7b>:3	2020/02/20 22:13:43	54 ms	1/1	8/8
1057	first at RandomForestClassifier.scala:145 first at RandomForestClassifier.scala:145	2020/02/20 22:13:43	12 ms	1/1	1/1
1056	collectAsMap at RandomForest.scala:557 collectAsMap at RandomForest.scala:557	2020/02/20 22:13:42	94 ms	2/2	16/16
1055	collectAsMap at RandomForest.scala:557 collectAsMap at RandomForest.scala:557	2020/02/20 22:13:42	85 ms	2/2	16/16
1054	collectAsMap at RandomForest.scala:557 collectAsMap at RandomForest.scala:557	2020/02/20 22:13:42	89 ms	2/2	16/16
1053	collectAsMap at RandomForest.scala:557 collectAsMap at RandomForest.scala:557	2020/02/20 22:13:42	76 ms	2/2	16/16
1052	collectAsMap at RandomForest.scala:557 collectAsMap at RandomForest.scala:557	2020/02/20 22:13:42	0.2 s	2/2	16/16
1051	collectAsMap at RandomForest.scala:922	2020/02/20 22:13:42	0.1 s	2/2	16/16

Figure 23 – Final PySpark Shell

Figure 23 illustrates the end of Jobs running on Spark. Note that 1067 Jobs were generated and the total time to run the entire script is approximately 1 hour.

On a production scale, that is, in a real environment this time would certainly decrease because:

1. Not all the script would be executed as it is at the moment because only the chosen predictive model would be used;
2. Balancing data would already be acquired in the initial collection, even before entering the script, in other words, the ETL process (Extract - Transform - Load) could feed a datalake with the standards defined in the project;
3. Third and most importantly, we would have an entire cluster with more machines at our disposal for distributed data storage with HDFS and consequently distributed processing with spark increasing our computational capacity.

10. Final Considerations

The use of Spark in this project was essential for the distributed processing of large amount of data. It was possible to identify that the agility and flexibility in data manipulation with Spark SQL, Spark MLlib, Pyspark libraries, among others presented, brings a competitive advantage to the project of forecasting Santander Bank Customers' Satisfaction.

With Pyspark Shell we observed several Jobs in operation, another advantage for the Data Science process because with it, if a possible error in the execution of the code occurs, we can find more quickly what the failure was and the causes of the problem, in addition to following each instruction in detail.

Fundamental concepts were also presented in the data science process, such as feature engineering, cleaning and transforming data, generating better opportunities for machine learning models; exploratory analysis delivering a little more value with the information implicit in the data; identification of the most relevant variables for the predictive model; building the model and assessing its accuracy.

During the beginning of the exploratory phase, we observed that we have more than 28 million anonymous data, which makes it difficult to carry out an analysis and generate business opportunities through the data. A set of unordered numbers does not indicate much, but a set of structured data generates content and consequently value for the company.

The statistical analysis of the data balance brought interesting information: unbalanced data and prone to “Satisfied”. Training the model in this way could generate a biased model where the predictions would converge to this variable, however our business problem aims to predict which users are Unsatisfied.

Knowing this, we entered a simulated environment where 50% of the data was "Satisfied" and 50% of the data was "Unsatisfied". We created 2 predictive models (DecisionTree and RandomForest), trained, tested and obtained accuracy of 76.70% and 75.33% respectively.

In order to choose the best model, I decided to analyze more resources besides the accuracy, so I analyzed the variables individually through the Confusion Matrix. The first model had better accuracy and delivered better True Negative, better predicting unsatisfied users, while the second was better with True Positive, better predicting satisfied users.

As we want to predict the level of unsatisfaction and act on top of customers likely to leave the bank, our choice would be the first model, the Decision Tree model. After choosing the model that presented the best overall performance, I saved it for future use.

Thinking about simulating a real environment, I built a dashboard for decision makers where batches of unknown data were collected daily in real time for a whole week.

Inside the dashboard we have interactive charts with some relevant information such as: Real - Time Prediction presenting the real-time forecast of the satisfaction level; Machine - Learning Model Accuracy presented the model's reliability for that day's predictions, that is, its accuracy is always being evaluated because in the event of a drop in this value we would have to model and train the predictive model again; and Real - Time Customer ID Satisfaction Prediction presenting in real time the IDs of consumers identified by the model as unsatisfied generating opportunities for action of decision makers.

This dashboard is of great value to the customer satisfaction team, as we reduce the identification time of unsatisfied consumers and as we are forecasting a large amount of data in real time, Santander Bank’s team will have more reliability in the prevention and action with customers likely to leave the customer portfolio.

To further improve this result, it would be possible to focus more intensively on improving the model parameters, or also add new variables at the time of feature engineering, or even collect more data focusing on the analyzed resources to train the model.

Source code

```
## *** Attention: ***
## Use Java JDK 11 and Apache Spark 2.4.2

## If an error message "name 'sc' is not defined" appears, stop pyspark and d
elete the folder metastore_db at the same folder where Jupyter notebook is se
t.

## Access http://localhost:4040 whenever you want to follow the jobs excecuti
on

## Predicting Santander Customer Satisfaction -----

## Kaggle ---------------------------------------------------------------------
-------------------
# https://www.kaggle.com/c/santander-customer-satisfaction/overview

## Data Description -----------------------------------------------------------
-----------------------------
# - train.csv - the training set
# - test.csv - the test set

## Evaluation -----------------------------------------------------------------
-----------------------
#The evaluation is based on area under the ROC curve between the predicted pr
obability and the observed target.
#The file should contain a header and have the following format:
# ID,TARGET
#  2, 0
#  5, 0
#  6, 0
#  etc.

## Data Dictionary ------------------------------------------------------------
----------------------------
# 1.ID
# 2.var3
# ...
# (meaningless features)
# ...
# 371.Target -> 1 for unsatisfied customers and 0 for satisfied customers.

# Note that most information are encoded.
```

Important Libraries

```
## Imports - Libraries ----------------------------------------------------
--------------------------------

# IMPORTING NECESSARY LIBRARIES

from pyspark.sql import SparkSession

from pyspark.sql import SQLContext

from pyspark.sql import Row

from pyspark.sql.types import *

import pandas as pd

#import warnings

#warnings.filterwarnings('ignore')
```

Creating a Spark Session to work with Dataframes

```
# Spark Session - used to work with Dataframes on Spark

spSession = SparkSession.builder.master("local").appName("Fellipe-Silva-
DS").getOrCreate()
```

Loading Data

```
## Datasets ----------------------------------------------------------------
---------------------

# Loading data and getting an RDD

# textFile CREATES AN RDD

santanderRDD = sc.textFile("train.csv")


# Optimizing performance using cache on RDD

santanderRDD.cache()


# Making an Action

santanderRDD.count()


# Making another Action

santanderRDD.take(4)
```

```
# Note that the data is between simple aspects '' (each line), indicating
they are strings, so I'll convert it further using a function I'll create
```

Feature Engineering

```
# Cleaning data - Removing first line

santanderRDD2 = santanderRDD.filter(lambda x: "ID" not in x)

santanderRDD2.count()


# Looking Again at my data

santanderRDD2.take(3)


# Looking to ALL data and trying to figure out if there is missing values (?,
NaN, "", ...)

santanderRDD2.collect()

# No Missing Values
```

Preparing Data to Dataframe

```
# Feature Engineering I - Function

def feateng1(data):


    # Dividing data into columns, separating by "," character

    dataList = data.split(",")


    TARGET = float(dataList[370])       # 1 for unsatisfied customers and 0
for satisfied customers.

    #ID = (dataList[0])

    VAR1 = float(dataList[1])

    VAR2 = float(dataList[2])

    VAR3 = float(dataList[3])

    VAR4 = float(dataList[4])

    VAR5 = float(dataList[5])

    VAR6 = float(dataList[6])

    VAR7 = float(dataList[7])

    VAR8 = float(dataList[8])
```

```
VAR9 = float(dataList[9])

VAR10 = float(dataList[10])

VAR11 = float(dataList[11])

VAR12 = float(dataList[12])

VAR13 = float(dataList[13])

VAR14 = float(dataList[14])

VAR15 = float(dataList[15])

VAR16 = float(dataList[16])

VAR17 = float(dataList[17])

VAR18 = float(dataList[18])

VAR19 = float(dataList[19])

VAR20 = float(dataList[20])

VAR21 = float(dataList[21])

VAR22 = float(dataList[22])

VAR23 = float(dataList[23])

VAR24 = float(dataList[24])

VAR25 = float(dataList[25])

VAR26 = float(dataList[26])

VAR27 = float(dataList[27])

VAR28 = float(dataList[28])

VAR29 = float(dataList[29])

VAR30 = float(dataList[30])

VAR31 = float(dataList[31])

VAR32 = float(dataList[32])

VAR33 = float(dataList[33])

VAR34 = float(dataList[34])

VAR35 = float(dataList[35])

VAR36 = float(dataList[36])

VAR37 = float(dataList[37])

VAR38 = float(dataList[38])

VAR39 = float(dataList[39])

VAR40 = float(dataList[40])
```

```
VAR41 = float(dataList[41])
VAR42 = float(dataList[42])
VAR43 = float(dataList[43])
VAR44 = float(dataList[44])
VAR45 = float(dataList[45])
VAR46 = float(dataList[46])
VAR47 = float(dataList[47])
VAR48 = float(dataList[48])
VAR49 = float(dataList[49])
VAR50 = float(dataList[50])
VAR51 = float(dataList[51])
VAR52 = float(dataList[52])
VAR53 = float(dataList[53])
VAR54 = float(dataList[54])
VAR55 = float(dataList[55])
VAR56 = float(dataList[56])
VAR57 = float(dataList[57])
VAR58 = float(dataList[58])
VAR59 = float(dataList[59])
VAR60 = float(dataList[60])
VAR61 = float(dataList[61])
VAR62 = float(dataList[62])
VAR63 = float(dataList[63])
VAR64 = float(dataList[64])
VAR65 = float(dataList[65])
VAR66 = float(dataList[66])
VAR67 = float(dataList[67])
VAR68 = float(dataList[68])
VAR69 = float(dataList[69])
VAR70 = float(dataList[70])
VAR71 = float(dataList[71])
VAR72 = float(dataList[72])
```

```
VAR73 = float(dataList[73])
VAR74 = float(dataList[74])
VAR75 = float(dataList[75])
VAR76 = float(dataList[76])
VAR77 = float(dataList[77])
VAR78 = float(dataList[78])
VAR79 = float(dataList[79])
VAR80 = float(dataList[80])
VAR81 = float(dataList[81])
VAR82 = float(dataList[82])
VAR83 = float(dataList[83])
VAR84 = float(dataList[84])
VAR85 = float(dataList[85])
VAR86 = float(dataList[86])
VAR87 = float(dataList[87])
VAR88 = float(dataList[88])
VAR89 = float(dataList[89])
VAR90 = float(dataList[90])
VAR91 = float(dataList[91])
VAR92 = float(dataList[92])
VAR93 = float(dataList[93])
VAR94 = float(dataList[94])
VAR95 = float(dataList[95])
VAR96 = float(dataList[96])
VAR97 = float(dataList[97])
VAR98 = float(dataList[98])
VAR99 = float(dataList[99])
VAR100 = float(dataList[100])
VAR101 = float(dataList[101])
VAR102 = float(dataList[102])
VAR103 = float(dataList[103])
VAR104 = float(dataList[104])
```

```
VAR105 = float(dataList[105])
VAR106 = float(dataList[106])
VAR107 = float(dataList[107])
VAR108 = float(dataList[108])
VAR109 = float(dataList[109])
VAR110 = float(dataList[110])
VAR111 = float(dataList[111])
VAR112 = float(dataList[112])
VAR113 = float(dataList[113])
VAR114 = float(dataList[114])
VAR115 = float(dataList[115])
VAR116 = float(dataList[116])
VAR117 = float(dataList[117])
VAR118 = float(dataList[118])
VAR119 = float(dataList[119])
VAR120 = float(dataList[120])
VAR121 = float(dataList[121])
VAR122 = float(dataList[122])
VAR123 = float(dataList[123])
VAR124 = float(dataList[124])
VAR125 = float(dataList[125])
VAR126 = float(dataList[126])
VAR127 = float(dataList[127])
VAR128 = float(dataList[128])
VAR129 = float(dataList[129])
VAR130 = float(dataList[130])
VAR131 = float(dataList[131])
VAR132 = float(dataList[132])
VAR133 = float(dataList[133])
VAR134 = float(dataList[134])
VAR135 = float(dataList[135])
VAR136 = float(dataList[136])
```

```
VAR137 = float(dataList[137])
VAR138 = float(dataList[138])
VAR139 = float(dataList[139])
VAR140 = float(dataList[140])
VAR141 = float(dataList[141])
VAR142 = float(dataList[142])
VAR143 = float(dataList[143])
VAR144 = float(dataList[144])
VAR145 = float(dataList[145])
VAR146 = float(dataList[146])
VAR147 = float(dataList[147])
VAR148 = float(dataList[148])
VAR149 = float(dataList[149])
VAR150 = float(dataList[150])
VAR151 = float(dataList[151])
VAR152 = float(dataList[152])
VAR153 = float(dataList[153])
VAR154 = float(dataList[154])
VAR155 = float(dataList[155])
VAR156 = float(dataList[156])
VAR157 = float(dataList[157])
VAR158 = float(dataList[158])
VAR159 = float(dataList[159])
VAR160 = float(dataList[160])
VAR161 = float(dataList[161])
VAR162 = float(dataList[162])
VAR163 = float(dataList[163])
VAR164 = float(dataList[164])
VAR165 = float(dataList[165])
VAR166 = float(dataList[166])
VAR167 = float(dataList[167])
VAR168 = float(dataList[168])
```

```
    VAR169 = float(dataList[169])

    VAR170 = float(dataList[170])

    VAR171 = float(dataList[171])

    VAR172 = float(dataList[172])

    VAR173 = float(dataList[173])

    VAR174 = float(dataList[174])

    VAR175 = float(dataList[175])

    VAR176 = float(dataList[176])

    VAR177 = float(dataList[177])

    VAR178 = float(dataList[178])

    VAR179 = float(dataList[179])

    VAR180 = float(dataList[180])

    VAR181 = float(dataList[181])

    VAR182 = float(dataList[182])

    VAR183 = float(dataList[183])

    VAR184 = float(dataList[184])

    VAR185 = float(dataList[185])


    # Creating 'lines' using the 'Row' function, preparing to the dataframe
analysis, cleaning and converting the data from string to float

    lines1 = Row(TARGET = TARGET, VAR1 = VAR1, VAR2 = VAR2, VAR3 = VAR3, VAR4
= VAR4, VAR5 = VAR5, VAR6 = VAR6, VAR7 = VAR7,

                 VAR8 = VAR8, VAR9 = VAR9, VAR10 = VAR10, VAR11 = VAR11, VAR12
= VAR12, VAR13 = VAR13, VAR14 = VAR14,

                 VAR15 = VAR15, VAR16 = VAR16, VAR17 = VAR17, VAR18 = VAR18,
VAR19 = VAR19, VAR20 = VAR20, VAR21 = VAR21,

                 VAR22 = VAR22, VAR23 = VAR23, VAR24 = VAR24, VAR25 = VAR25,
VAR26 = VAR26, VAR27 = VAR27, VAR28 = VAR28,

                 VAR29 = VAR29, VAR30 = VAR30, VAR31 = VAR31, VAR32 = VAR32,
VAR33 = VAR33, VAR34 = VAR34, VAR35 = VAR35,

                 VAR36 = VAR36, VAR37 = VAR37, VAR38 = VAR38, VAR39 = VAR39,
VAR40 = VAR40, VAR41 = VAR41, VAR42 = VAR42,

                 VAR43 = VAR43, VAR44 = VAR44, VAR45 = VAR45, VAR46 = VAR46,
VAR47 = VAR47, VAR48 = VAR48, VAR49 = VAR49,

                 VAR50 = VAR50, VAR51 = VAR51, VAR52 = VAR52, VAR53 = VAR53,
VAR54 = VAR54, VAR55 = VAR55, VAR56 = VAR56,
                VAR57 = VAR57, VAR58 = VAR58, VAR59 = VAR59, VAR60 = VAR60,
VAR61 = VAR61, VAR62 = VAR62, VAR63 = VAR63,

                VAR64 = VAR64, VAR65 = VAR65, VAR66 = VAR66, VAR67 = VAR67,
VAR68 = VAR68, VAR69 = VAR69, VAR70 = VAR70,

                VAR71 = VAR71, VAR72 = VAR72, VAR73 = VAR73, VAR74 = VAR74,
VAR75 = VAR75, VAR76 = VAR76, VAR77 = VAR77,

                VAR78 = VAR78, VAR79 = VAR79, VAR80 = VAR80, VAR81 = VAR81,
VAR82 = VAR82, VAR83 = VAR83, VAR84 = VAR84,

                VAR85 = VAR85, VAR86 = VAR86, VAR87 = VAR87, VAR88 = VAR88,
VAR89 = VAR89, VAR90 = VAR90, VAR91 = VAR91,

                VAR92 = VAR92, VAR93 = VAR93, VAR94 = VAR94, VAR95 = VAR95,
VAR96 = VAR96, VAR97 = VAR97, VAR98 = VAR98,

                VAR99 = VAR99, VAR100 = VAR100, VAR101 = VAR101, VAR102 =
VAR102, VAR103 = VAR103, VAR104 = VAR104,

                VAR105 = VAR105, VAR106 = VAR106, VAR107 = VAR107, VAR108 =
VAR108, VAR109 = VAR109, VAR110 = VAR110,

                VAR111 = VAR111, VAR112 = VAR112, VAR113 = VAR113, VAR114 =
VAR114, VAR115 = VAR115, VAR116 = VAR116,

                VAR117 = VAR117, VAR118 = VAR118, VAR119 = VAR119, VAR120 =
VAR120, VAR121 = VAR121, VAR122 = VAR122,

                VAR123 = VAR123, VAR124 = VAR124, VAR125 = VAR125, VAR126 =
VAR126, VAR127 = VAR127, VAR128 = VAR128,

                VAR129 = VAR129, VAR130 = VAR130, VAR131 = VAR131, VAR132 =
VAR132, VAR133 = VAR133, VAR134 = VAR134,

                VAR135 = VAR135, VAR136 = VAR136, VAR137 = VAR137, VAR138 =
VAR138, VAR139 = VAR139, VAR140 = VAR140,

                VAR141 = VAR141, VAR142 = VAR142, VAR143 = VAR143, VAR144 =
VAR144, VAR145 = VAR145, VAR146 = VAR146,

                VAR147 = VAR147, VAR148 = VAR148, VAR149 = VAR149, VAR150 =
VAR150, VAR151 = VAR151, VAR152 = VAR152,

                VAR153 = VAR153, VAR154 = VAR154, VAR155 = VAR155, VAR156 =
VAR156, VAR157 = VAR157, VAR158 = VAR158,

                VAR159 = VAR159, VAR160 = VAR160, VAR161 = VAR161, VAR162 =
VAR162, VAR163 = VAR163, VAR164 = VAR164,

                VAR165 = VAR165, VAR166 = VAR166, VAR167 = VAR167, VAR168 =
VAR168, VAR169 = VAR169, VAR170 = VAR170,

                VAR171 = VAR171, VAR172 = VAR172, VAR173 = VAR173, VAR174 =
VAR174, VAR175 = VAR175, VAR176 = VAR176,

                VAR177 = VAR177, VAR178 = VAR178, VAR179 = VAR179, VAR180 =
VAR180, VAR181 = VAR181, VAR182 = VAR182,
                  VAR183 = VAR183, VAR184 = VAR184, VAR185 = VAR185)

    return lines1

# Feature Engineering II - Function
def feateng2(data):

    # Dividing data into columns, separating by "," character
    dataList = data.split(",")

    VAR186 = float(dataList[186])
    VAR187 = float(dataList[187])
    VAR188 = float(dataList[188])
    VAR189 = float(dataList[189])
    VAR190 = float(dataList[190])
    VAR191 = float(dataList[191])
    VAR192 = float(dataList[192])
    VAR193 = float(dataList[193])
    VAR194 = float(dataList[194])
    VAR195 = float(dataList[195])
    VAR196 = float(dataList[196])
    VAR197 = float(dataList[197])
    VAR198 = float(dataList[198])
    VAR199 = float(dataList[199])
    VAR200 = float(dataList[200])
    VAR201 = float(dataList[201])
    VAR202 = float(dataList[202])
    VAR203 = float(dataList[203])
    VAR204 = float(dataList[204])
    VAR205 = float(dataList[205])
    VAR206 = float(dataList[206])
    VAR207 = float(dataList[207])
```

```
VAR208 = float(dataList[208])
VAR209 = float(dataList[209])
VAR210 = float(dataList[210])
VAR211 = float(dataList[211])
VAR212 = float(dataList[212])
VAR213 = float(dataList[213])
VAR214 = float(dataList[214])
VAR215 = float(dataList[215])
VAR216 = float(dataList[216])
VAR217 = float(dataList[217])
VAR218 = float(dataList[218])
VAR219 = float(dataList[219])
VAR220 = float(dataList[220])
VAR221 = float(dataList[221])
VAR222 = float(dataList[222])
VAR223 = float(dataList[223])
VAR224 = float(dataList[224])
VAR225 = float(dataList[225])
VAR226 = float(dataList[226])
VAR227 = float(dataList[227])
VAR228 = float(dataList[228])
VAR229 = float(dataList[229])
VAR230 = float(dataList[230])
VAR231 = float(dataList[231])
VAR232 = float(dataList[232])
VAR233 = float(dataList[233])
VAR234 = float(dataList[234])
VAR235 = float(dataList[235])
VAR236 = float(dataList[236])
VAR237 = float(dataList[237])
VAR238 = float(dataList[238])
VAR239 = float(dataList[239])
```

```
VAR240 = float(dataList[240])
VAR241 = float(dataList[241])
VAR242 = float(dataList[242])
VAR243 = float(dataList[243])
VAR244 = float(dataList[244])
VAR245 = float(dataList[245])
VAR246 = float(dataList[246])
VAR247 = float(dataList[247])
VAR248 = float(dataList[248])
VAR249 = float(dataList[249])
VAR250 = float(dataList[250])
VAR251 = float(dataList[251])
VAR252 = float(dataList[252])
VAR253 = float(dataList[253])
VAR254 = float(dataList[254])
VAR255 = float(dataList[255])
VAR256 = float(dataList[256])
VAR257 = float(dataList[257])
VAR258 = float(dataList[258])
VAR259 = float(dataList[259])
VAR260 = float(dataList[260])
VAR261 = float(dataList[261])
VAR262 = float(dataList[262])
VAR263 = float(dataList[263])
VAR264 = float(dataList[264])
VAR265 = float(dataList[265])
VAR266 = float(dataList[266])
VAR267 = float(dataList[267])
VAR268 = float(dataList[268])
VAR269 = float(dataList[269])
VAR270 = float(dataList[270])
VAR271 = float(dataList[271])
```

```
VAR272 = float(dataList[272])
VAR273 = float(dataList[273])
VAR274 = float(dataList[274])
VAR275 = float(dataList[275])
VAR276 = float(dataList[276])
VAR277 = float(dataList[277])
VAR278 = float(dataList[278])
VAR279 = float(dataList[279])
VAR280 = float(dataList[280])
VAR281 = float(dataList[281])
VAR282 = float(dataList[282])
VAR283 = float(dataList[283])
VAR284 = float(dataList[284])
VAR285 = float(dataList[285])
VAR286 = float(dataList[286])
VAR287 = float(dataList[287])
VAR288 = float(dataList[288])
VAR289 = float(dataList[289])
VAR290 = float(dataList[290])
VAR291 = float(dataList[291])
VAR292 = float(dataList[292])
VAR293 = float(dataList[293])
VAR294 = float(dataList[294])
VAR295 = float(dataList[295])
VAR296 = float(dataList[296])
VAR297 = float(dataList[297])
VAR298 = float(dataList[298])
VAR299 = float(dataList[299])
VAR300 = float(dataList[300])
VAR301 = float(dataList[301])
VAR302 = float(dataList[302])
VAR303 = float(dataList[303])
```

```
VAR304 = float(dataList[304])

VAR305 = float(dataList[305])

VAR306 = float(dataList[306])

VAR307 = float(dataList[307])

VAR308 = float(dataList[308])

VAR309 = float(dataList[309])

VAR310 = float(dataList[310])

VAR311 = float(dataList[311])

VAR312 = float(dataList[312])

VAR313 = float(dataList[313])

VAR314 = float(dataList[314])

VAR315 = float(dataList[315])

VAR316 = float(dataList[316])

VAR317 = float(dataList[317])

VAR318 = float(dataList[318])

VAR319 = float(dataList[319])

VAR320 = float(dataList[320])

VAR321 = float(dataList[321])

VAR322 = float(dataList[322])

VAR323 = float(dataList[323])

VAR324 = float(dataList[324])

VAR325 = float(dataList[325])

VAR326 = float(dataList[326])

VAR327 = float(dataList[327])

VAR328 = float(dataList[328])

VAR329 = float(dataList[329])

VAR330 = float(dataList[330])

VAR331 = float(dataList[331])

VAR332 = float(dataList[332])

VAR333 = float(dataList[333])

VAR334 = float(dataList[334])

VAR335 = float(dataList[335])
```

```
VAR336 = float(dataList[336])
VAR337 = float(dataList[337])
VAR338 = float(dataList[338])
VAR339 = float(dataList[339])
VAR340 = float(dataList[340])
VAR341 = float(dataList[341])
VAR342 = float(dataList[342])
VAR343 = float(dataList[343])
VAR344 = float(dataList[344])
VAR345 = float(dataList[345])
VAR346 = float(dataList[346])
VAR347 = float(dataList[347])
VAR348 = float(dataList[348])
VAR349 = float(dataList[349])
VAR350 = float(dataList[350])
VAR351 = float(dataList[351])
VAR352 = float(dataList[352])
VAR353 = float(dataList[353])
VAR354 = float(dataList[354])
VAR355 = float(dataList[355])
VAR356 = float(dataList[356])
VAR357 = float(dataList[357])
VAR358 = float(dataList[358])
VAR359 = float(dataList[359])
VAR360 = float(dataList[360])
VAR361 = float(dataList[361])
VAR362 = float(dataList[362])
VAR363 = float(dataList[363])
VAR364 = float(dataList[364])
VAR365 = float(dataList[365])
VAR366 = float(dataList[366])
VAR367 = float(dataList[367])
```

```
    VAR368 = float(dataList[368])

    VAR369 = float(dataList[369])

    TARGET = float(dataList[370])      # 1 for unsatisfied customers and 0
for satisfied customers.


    # Creating 'lines' using the 'Row' function, preparing to the dataframe
analysis, cleaning and converting the data from string to float

    lines2 = Row(VAR186 = VAR186, VAR187 = VAR187, VAR188 = VAR188,

                VAR189 = VAR189, VAR190 = VAR190, VAR191 = VAR191, VAR192 =
VAR192, VAR193 = VAR193, VAR194 = VAR194,

                VAR195 = VAR195, VAR196 = VAR196, VAR197 = VAR197, VAR198 =
VAR198, VAR199 = VAR199, VAR200 = VAR200,

                VAR201 = VAR201, VAR202 = VAR202, VAR203 = VAR203, VAR204 =
VAR204, VAR205 = VAR205, VAR206 = VAR206,

                VAR207 = VAR207, VAR208 = VAR208, VAR209 = VAR209, VAR210 =
VAR210, VAR211 = VAR211, VAR212 = VAR212,

                VAR213 = VAR213, VAR214 = VAR214, VAR215 = VAR215, VAR216 =
VAR216, VAR217 = VAR217, VAR218 = VAR218,

                VAR219 = VAR219, VAR220 = VAR220, VAR221 = VAR221, VAR222 =
VAR222, VAR223 = VAR223, VAR224 = VAR224,

                VAR225 = VAR225, VAR226 = VAR226, VAR227 = VAR227, VAR228 =
VAR228, VAR229 = VAR229, VAR230 = VAR230,

                VAR231 = VAR231, VAR232 = VAR232, VAR233 = VAR233, VAR234 =
VAR234, VAR235 = VAR235, VAR236 = VAR236,

                VAR237 = VAR237, VAR238 = VAR238, VAR239 = VAR239, VAR240 =
VAR240, VAR241 = VAR241, VAR242 = VAR242,

                VAR243 = VAR243, VAR244 = VAR244, VAR245 = VAR245, VAR246 =
VAR246, VAR247 = VAR247, VAR248 = VAR248,

                VAR249 = VAR249, VAR250 = VAR250, VAR251 = VAR251, VAR252 =
VAR252, VAR253 = VAR253, VAR254 = VAR254,

                VAR255 = VAR255, VAR256 = VAR256, VAR257 = VAR257, VAR258 =
VAR258, VAR259 = VAR259, VAR260 = VAR260,

                VAR261 = VAR261, VAR262 = VAR262, VAR263 = VAR263, VAR264 =
VAR264, VAR265 = VAR265, VAR266 = VAR266,

                VAR267 = VAR267, VAR268 = VAR268, VAR269 = VAR269, VAR270 =
VAR270, VAR271 = VAR271, VAR272 = VAR272,

                VAR273 = VAR273, VAR274 = VAR274, VAR275 = VAR275, VAR276 =
VAR276, VAR277 = VAR277, VAR278 = VAR278,
                VAR279 = VAR279, VAR280 = VAR280, VAR281 = VAR281, VAR282 =
VAR282, VAR283 = VAR283, VAR284 = VAR284,

                VAR285 = VAR285, VAR286 = VAR286, VAR287 = VAR287, VAR288 =
VAR288, VAR289 = VAR289, VAR290 = VAR290,

                VAR291 = VAR291, VAR292 = VAR292, VAR293 = VAR293, VAR294 =
VAR294, VAR295 = VAR295, VAR296 = VAR296,

                VAR297 = VAR297, VAR298 = VAR298, VAR299 = VAR299, VAR300 =
VAR300, VAR301 = VAR301, VAR302 = VAR302,

                VAR303 = VAR303, VAR304 = VAR304, VAR305 = VAR305, VAR306 =
VAR306, VAR307 = VAR307, VAR308 = VAR308,

                VAR309 = VAR309, VAR310 = VAR310, VAR311 = VAR311, VAR312 =
VAR312, VAR313 = VAR313, VAR314 = VAR314,

                VAR315 = VAR315, VAR316 = VAR316, VAR317 = VAR317, VAR318 =
VAR318, VAR319 = VAR319, VAR320 = VAR320,

                VAR321 = VAR321, VAR322 = VAR322, VAR323 = VAR323, VAR324 =
VAR324, VAR325 = VAR325, VAR326 = VAR326,

                VAR327 = VAR327, VAR328 = VAR328, VAR329 = VAR329, VAR330 =
VAR330, VAR331 = VAR331, VAR332 = VAR332,

                VAR333 = VAR333, VAR334 = VAR334, VAR335 = VAR335, VAR336 =
VAR336, VAR337 = VAR337, VAR338 = VAR338,

                VAR339 = VAR339, VAR340 = VAR340, VAR341 = VAR341, VAR342 =
VAR342, VAR343 = VAR343, VAR344 = VAR344,

                VAR345 = VAR345, VAR346 = VAR346, VAR347 = VAR347, VAR348 =
VAR348, VAR349 = VAR349, VAR350 = VAR350,

                VAR351 = VAR351, VAR352 = VAR352, VAR353 = VAR353, VAR354 =
VAR354, VAR355 = VAR355, VAR356 = VAR356,

                VAR357 = VAR357, VAR358 = VAR358, VAR359 = VAR359, VAR360 =
VAR360, VAR361 = VAR361, VAR362 = VAR362,

                VAR363 = VAR363, VAR364 = VAR364, VAR365 = VAR365, VAR366 =
VAR366, VAR367 = VAR367, VAR368 = VAR368,

                VAR369 = VAR369, TARGET = TARGET)


    return lines2


# Applying function above to RDD without headers

santanderRDD3_1 = santanderRDD2.map(feateng1)

santanderRDD3_2 = santanderRDD2.map(feateng2)
```

```
# Looking at the result
santanderRDD3_1.cache
santanderRDD3_1.take(1)


# Looking at the result
santanderRDD3_2.cache
santanderRDD3_2.take(1)
```

Creating a Pyspark Dataframe

```
# Creating a Dataframe SO I'M ABLE TO USE THE select FUNCTION FROM SparkSQL
santanderDF_1 = spSession.createDataFrame(santanderRDD3_1)
santanderDF_2 = spSession.createDataFrame(santanderRDD3_2)


type(santanderDF_1)


# Printing santanderDF_1 Object Type and each Row Type
santanderDF_1.printSchema()


# Printing santanderDF_2 Object Type and each Row Type
santanderDF_2.printSchema()


santanderDF_1.show(5)
```

Exploratory Analysis

```
# Exploratory Analysis
# Here I'll compare in some cases both Pyspark SQL Functions and SQL
Functions


# 01.Pyspark SQL Functions ->
https://spark.apache.org/docs/2.1.0/api/python/pyspark.sql.html
# 02.SQL Functions -> http://www-
db.deis.unibo.it/courses/TW/DOCS/w3schools/sql/sql_func_count.asp.html
```

```
# Attention: standard queries SQL ANSI just work on JAVA 8


!java -version


# Registering the dataframe as a Temp Table (so i'll be able to use queries
SQL ANSI)

# HERE I'M CREATING A REAL TABLE, IT WILL EXIST IN MEMORY, NOT ONLY THE
DATAFRAME

santanderDF_1.createOrReplaceTempView("santanderTB_1")

santanderDF_2.createOrReplaceTempView("santanderTB_2")
```

Data Balance

```
# Plotting some analysis using Seaborn Library

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline


# 01.Pyspark SQL Functions

from pyspark.sql.functions import *


santanderDF_2.cube('TARGET').agg(count('TARGET').alias('Target_Data'),

                                 grouping('TARGET')).orderBy('TARGET').show()


# 02.SQL Functions

# Acquiring Data from santanderTB_2 to plot the Data Balance we have in this
dataset

pysparkSqlDF = spSession.sql("select TARGET, count(TARGET) as Target_Data \

                              from santanderTB_2 \

                              group by TARGET")

#pysparkSqlDF.show()


# Transforming to Pandas so I'll plot some analysis using Seaborn

pandasDF_2 = pysparkSqlDF.toPandas()
```

```
# Looking at the result
pandasDF_2


# Transforming values of Feature 'TARGET'
for x in pandasDF_2.TARGET:
    if x == 0:
        pandasDF_2['TARGET'][0] = 'Satisfied'
    else:
        pandasDF_2['TARGET'][1] = 'Unsatisfied'


# Setting width and height of the plot-figure
plt.figure(figsize=(18,10))


# Barplot of 'Satisfied' and 'Unsatisfied' Customers
sns.barplot(x=pandasDF_2.TARGET, y=pandasDF_2.Target_Data, palette='Reds')


# Data not balanced, maybe it's important to get almost the same amount of
data for both to have a fair ML Model.
# Before that I'll get the most important features and then resize the
dataset to almost 50/50 of Satisfied and Unsatisfied Customers.
```

Correlation

Correlation Matrix - santanderTB_1

```
# Now I'll compare all features with TARGET


# Acquiring Data from santanderTB_1 to plot a Correlation Matrix
pysparkSqlDF = spSession.sql("select * \
                              from santanderTB_1 ")
#pysparkSqlDF.show()


# Transforming to Pandas so I'll plot some analysis using Seaborn
```

```
pandasDF_1 = pysparkSqlDF.toPandas()


corrMatrix1 = pandasDF_1.corr()


# Setting width and height of the plot-figure

plt.figure(figsize=(18,10))


sns.heatmap(corrMatrix1, cmap='PuOr')
```

Correlation of Target and all other Features - santanderTB_1

```
# Correlation between features


# First I'll get each column of telecomDF4

for i in santanderDF_1.columns:

    # for each one I'll 'select(i)', and use the 'take' action from [0][0]
combination as if it was a matrix

    if not(isinstance(santanderDF_1.select(i).take(1)[0][0], str)):

        print("Correlation between Target with: ", i,
santanderDF_1.stat.corr('TARGET', i))
```

Correlation Matrix - santanderTB_2

```
# Acquiring Data from santanderTB_2 to plot a Correlation Matrix

pysparkSqlDF = spSession.sql("select * \

                              from santanderTB_2 ")

#pysparkSqlDF.show()


# Transforming to Pandas so I'll plot some analysis using Seaborn

pandasDF_2 = pysparkSqlDF.toPandas()


corrMatrix2 = pandasDF_2.corr()


# Setting width and height of the plot-figure
```

```
plt.figure(figsize=(18,10))

sns.heatmap(corrMatrix2, cmap='PuOr')
```

Correlation of Target and all other Features - santanderTB_2

```
# Correlation between features

# First I'll get each column of telecomDF4
for i in santanderDF_2.columns:
    # for each one I'll 'select(i)', and use the 'take' action from [0][0]
combination as if it was a matrix
    if not(isinstance(santanderDF_2.select(i).take(1)[0][0], str)):
        print("Correlation between Target with: ", i,
santanderDF_2.stat.corr('TARGET', i))
```

Chosen Features from - santanderTB_1 and santanderTB_2

```
# Feature Engineering III - Function
def feateng3(data):

    # Dividing data into columns, separating by "," character
    dataList = data.split(",")

    # Features with correlation -> 0.01......
    VAR101 = float(dataList[101])
    VAR102 = float(dataList[102])
    VAR11 = float(dataList[11])
    VAR112 = float(dataList[112])
    VAR113 = float(dataList[113])
    VAR116 = float(dataList[116])
    VAR117 = float(dataList[117])
    VAR118 = float(dataList[118])
    VAR119 = float(dataList[119])
```

```
VAR126 = float(dataList[126])

VAR127 = float(dataList[127])

VAR130 = float(dataList[130])

VAR131 = float(dataList[131])

VAR158 = float(dataList[158])

VAR165 = float(dataList[165])

VAR170 = float(dataList[170])

VAR35 = float(dataList[35])

VAR36 = float(dataList[36])

VAR4 = float(dataList[4])

VAR47 = float(dataList[47])

VAR48 = float(dataList[48])

VAR52 = float(dataList[52])

VAR54 = float(dataList[54])

VAR55 = float(dataList[55])

VAR56 = float(dataList[56])

VAR8 = float(dataList[8])

VAR9 = float(dataList[9])

VAR189 = float(dataList[189])

VAR242 = float(dataList[242])

VAR255 = float(dataList[255])

VAR278 = float(dataList[278])

VAR279 = float(dataList[279])

VAR285 = float(dataList[285])

VAR290 = float(dataList[290])

VAR296 = float(dataList[296])

VAR297 = float(dataList[297])

VAR301 = float(dataList[301])

VAR307 = float(dataList[307])

VAR312 = float(dataList[312])

VAR330 = float(dataList[330])

VAR331 = float(dataList[331])
```

```
VAR332 = float(dataList[332])

VAR337 = float(dataList[337])

VAR338 = float(dataList[338])

VAR342 = float(dataList[342])

VAR345 = float(dataList[345])

VAR347 = float(dataList[347])

VAR348 = float(dataList[348])

VAR369 = float(dataList[369])


# Features with correlation -> 0.02......

VAR13 = float(dataList[13])

VAR14 = float(dataList[14])

VAR15 = float(dataList[15])

VAR152 = float(dataList[152])

VAR168 = float(dataList[168])

VAR169 = float(dataList[169])

VAR17 = float(dataList[17])

VAR172 = float(dataList[172])

VAR177 = float(dataList[177])

VAR18 = float(dataList[18])

VAR29 = float(dataList[29])

VAR40 = float(dataList[40])

VAR51 = float(dataList[51])

VAR53 = float(dataList[53])

VAR90 = float(dataList[90])

VAR95 = float(dataList[95])

VAR96 = float(dataList[96])

VAR191 = float(dataList[191])

VAR225 = float(dataList[225])

VAR260 = float(dataList[260])

VAR264 = float(dataList[264])

VAR277 = float(dataList[277])
```

```
VAR282 = float(dataList[282])
VAR300 = float(dataList[300])
VAR302 = float(dataList[302])
VAR303 = float(dataList[303])
VAR329 = float(dataList[329])
VAR339 = float(dataList[339])
VAR340 = float(dataList[340])
VAR341 = float(dataList[341])
VAR343 = float(dataList[343])
VAR344 = float(dataList[344])

# Features with correlation -> 0.03......
VAR100 = float(dataList[100])
VAR105 = float(dataList[105])
VAR114 = float(dataList[114])
VAR115 = float(dataList[115])
VAR138 = float(dataList[138])
VAR155 = float(dataList[155])
VAR16 = float(dataList[16])
VAR183 = float(dataList[183])
VAR24 = float(dataList[24])
VAR30 = float(dataList[30])
VAR31 = float(dataList[31])
VAR32 = float(dataList[32])
VAR33 = float(dataList[33])
VAR34 = float(dataList[34])
VAR39 = float(dataList[39])
VAR49 = float(dataList[49])
VAR50 = float(dataList[50])
VAR77 = float(dataList[77])
VAR80 = float(dataList[80])
VAR97 = float(dataList[97])
```

```
    VAR98 = float(dataList[98])

    VAR99 = float(dataList[99])

    VAR283 = float(dataList[283])

    VAR284 = float(dataList[284])


    # Features with correlation -> 0.04......

    VAR28 = float(dataList[28])

    VAR94 = float(dataList[94])


    # Features with correlation -> 0.07......

    VAR148 = float(dataList[148])


    # Features with correlation -> 0.08......

    VAR89 = float(dataList[89])


    # Features with correlation -> 0.1......

    VAR139 = float(dataList[139])

    VAR159 = float(dataList[159])

    VAR2 = float(dataList[2])

    VAR25 = float(dataList[25])

    VAR64 = float(dataList[64])

    VAR91 = float(dataList[91])

    VAR194 = float(dataList[194])

    VAR281 = float(dataList[281])


    # Target Feature

    TARGET = float(dataList[370])      # 1 for unsatisfied customers and 0 
for satisfied customers.


    # Creating 'lines' using the 'Row' function, preparing to the dataframe 
analysis, cleaning and converting the data from string to float

    #lines3 = Row(VAR101 = VAR101, VAR102 = VAR102, VAR11 = VAR11, VAR112 = 
VAR112, VAR113 = VAR113, VAR116 = VAR116, VAR117 = VAR117, VAR118 = VAR118, 
VAR119 = VAR119, VAR126 = VAR126, VAR127 = VAR127, VAR130 = VAR130, VAR131 =
VAR131, VAR158 = VAR158, VAR165 = VAR165, VAR170 = VAR170, VAR35 = VAR35,
VAR36 = VAR36, VAR4 = VAR4, VAR47 = VAR47, VAR48 = VAR48, VAR52 = VAR52,
VAR54 = VAR54, VAR55 = VAR55, VAR56 = VAR56, VAR8 = VAR8, VAR9 = VAR9, VAR189
= VAR189, VAR242 = VAR242, VAR255 = VAR255, VAR278 = VAR278, VAR279 = VAR279,
VAR285 = VAR285, VAR290 = VAR290, VAR296 = VAR296, VAR297 = VAR297, VAR301 =
VAR301, VAR307 = VAR307, VAR312 = VAR312, VAR330 = VAR330, VAR331 = VAR331,
VAR332 = VAR332, VAR337 = VAR337, VAR338 = VAR338, VAR342 = VAR342, VAR345 =
VAR345, VAR347 = VAR347, VAR348 = VAR348, VAR369 = VAR369,

    #              VAR13 = VAR13, VAR14 = VAR14, VAR15 = VAR15, VAR152 =
VAR152, VAR168 = VAR168, VAR169 = VAR169, VAR17 = VAR17, VAR172 = VAR172,
VAR177 = VAR177, VAR18 = VAR18, VAR29 = VAR29, VAR40 = VAR40, VAR51 = VAR51,
VAR53 = VAR53, VAR90 = VAR90, VAR95 = VAR95, VAR96 = VAR96, VAR191 = VAR191,
VAR225 = VAR225, VAR260 = VAR260, VAR264 = VAR264, VAR277 = VAR277, VAR282 =
VAR282, VAR300 = VAR300, VAR302 = VAR302, VAR303 = VAR303, VAR329 = VAR329,
VAR339 = VAR339, VAR340 = VAR340, VAR341 = VAR341, VAR343 = VAR343, VAR344 =
VAR344,

    #              VAR100 = VAR100, VAR105 = VAR105, VAR114 = VAR114, VAR115 =
VAR115, VAR138 = VAR138, VAR155 = VAR155, VAR16 = VAR16, VAR183 = VAR183,
VAR24 = VAR24, VAR30 = VAR30, VAR31 = VAR31, VAR32 = VAR32, VAR33 = VAR33,
VAR34 = VAR34, VAR39 = VAR39, VAR49 = VAR49, VAR50 = VAR50, VAR77 = VAR77,
VAR80 = VAR80, VAR97 = VAR97, VAR98 = VAR98, VAR99 = VAR99, VAR283 = VAR283,
VAR284 = VAR284,

    #              VAR28 = VAR28, VAR94 = VAR94,

    #              VAR148 = VAR148,

    #              VAR89 = VAR89,

    #              VAR139 = VAR139, VAR159 = VAR159, VAR2 = VAR2, VAR25 =
VAR25, VAR64 = VAR64, VAR91 = VAR91, VAR194 = VAR194, VAR281 = VAR281,

    #              TARGET = TARGET)


    lines3 = Row(VAR100 = VAR100, VAR105 = VAR105, VAR114 = VAR114, VAR115 =
VAR115, VAR138 = VAR138, VAR155 = VAR155, VAR16 = VAR16, VAR183 = VAR183,
VAR24 = VAR24, VAR30 = VAR30, VAR31 = VAR31, VAR32 = VAR32, VAR33 = VAR33,
VAR34 = VAR34, VAR39 = VAR39, VAR49 = VAR49, VAR50 = VAR50, VAR77 = VAR77,
VAR80 = VAR80, VAR97 = VAR97, VAR98 = VAR98, VAR99 = VAR99, VAR283 = VAR283,
VAR284 = VAR284,

                 VAR28 = VAR28, VAR94 = VAR94,

                 VAR148 = VAR148,

                 VAR89 = VAR89,

                 VAR139 = VAR139, VAR159 = VAR159, VAR2 = VAR2, VAR25 =
VAR25, VAR64 = VAR64, VAR91 = VAR91, VAR194 = VAR194, VAR281 = VAR281,

                 TARGET = TARGET)
    return lines3


# Applying function above to RDD without headers

santanderRDD4 = santanderRDD2.map(feateng3)


# Looking at the result

santanderRDD4.cache

santanderRDD4.take(1)


# Creating a PYSPARK Dataframe SO I'M ABLE TO USE THE select FUNCTION FROM SparkSQL

santanderDF = spSession.createDataFrame(santanderRDD4)


type(santanderDF)


# Printing santanderDF_1 Object Type and each Row Type

santanderDF.printSchema()
```

New Data Balance

```
# Registering the dataframe as a Temp Table (so i'll be able to use queries SQL ANSI)

# HERE I'M CREATING A REAL TABLE, IT WILL EXIST IN MEMORY, NOT ONLY THE DATAFRAME

santanderDF.createOrReplaceTempView("santanderTB")


# Acquiring 'Satisfied' Data from santanderTB to plot the Data Balance we have in this dataset

pysparkSqlDF = spSession.sql("select * \

                              from santanderTB \

                              where TARGET = 0")

#pysparkSqlDF.show()


# Transforming to Pandas so I'll plot some analysis using Seaborn
```

```
pandasDF = pysparkSqlDF.toPandas()


len(pandasDF)


# Sampling 4.000 'Satisfied' Data

import random


random.seed(69)

pandasDF0 = pandasDF.sample(4000)


len(pandasDF0)


# Storing not used dataset to use at the final of the project

pandasDFN = pandasDF.drop(index=pandasDF0.index)


len(pandasDFN)


# Acquiring 'Unsatisfied' Data from santanderTB to plot the Data Balance we
have in this dataset

pysparkSqlDF = spSession.sql("select * \

                              from santanderTB \

                              where TARGET = 1")

#pysparkSqlDF.show()


# Transforming to Pandas so I'll plot some analysis using Seaborn

pandasDF1 = pysparkSqlDF.toPandas()


len(pandasDF1)


# Joining both Dataframes in one

pandasDF = pandasDF0.append(pandasDF1, ignore_index=True)

pandasDF
```

```
df = pandasDF['TARGET'].value_counts().sort_values()
df


# Setting width and height of the plot-figure
plt.figure(figsize=(18,10))


# Barplot of 'Satisfied' and 'Unsatisfied' Customers
sns.barplot(x=df.index, y=df.values, palette='Reds')


type(pandasDF)


# Transforming the pandas Dataframe to a PYSPARK Dataframe
santanderDF2 = spSession.createDataFrame(pandasDF)


type(santanderDF2)


# Correlation between features


# First I'll get each column of telecomDF4
for i in santanderDF2.columns:
    # for each one I'll 'select(i)', and use the 'take' action from [0][0]
combination as if it was a matrix
    if not(isinstance(santanderDF2.select(i).take(1)[0][0], str)):
        print("Correlation between Target with: ", i,
santanderDF2.stat.corr('TARGET', i))


# Acquiring Data from santanderTB_2 to plot a Correlation Matrix
pysparkSqlDF = spSession.sql("select * \
                              from santanderTB ")
#pysparkSqlDF.show()


# Transforming to Pandas so I'll plot some analysis using Seaborn
```

```
pandasDF_Corr = pysparkSqlDF.toPandas()


corrMatrix2 = pandasDF_Corr.corr()


# Setting width and height of the plot-figure

plt.figure(figsize=(18,10))


sns.heatmap(corrMatrix2, cmap='PuOr')


# Now that we have better correlations between features, let's start the
Machine Learning Process.
```

Pre-Processing the Dataset

```
# Pre-Processing my Dataset


# Creating an LabeledPoint (target, Vector[features])

# Removing not relevant columns or with low correlation to the model, this
way I choose the columns I want the model to have (just observe the selection
done in Vectors.dense(....))


# ATTENTION TO THIS VERY IMPORTANT CONCEPT!!!!!!!!!

# SOME APACHE SPARK MACHINE LEARNING ALGORITHMS (MAINLY THE REGRESSION TYPE)
NEED DATA TO BE

# IN A SPECIFIC FORMAT TO PERFORM THE TRAINING(FIT).

# TO THIS HAPPEN I NEED TO DELIVER TO THE MODEL THE DATA IN VECTOR FORMAT,
SPECIFICALLY USING A

# DENSE OR SPARSE VECTOR. THIS BECAUSE APACHE SPARK WORKS WITH CLUSTER AND
THEN IT WILL

# DISTRIBUTE THESE DATA BETWEEN MACHINES.

from pyspark.ml.linalg import Vectors


def transformFeatures(row):

    #obj = (row['TARGET'], Vectors.dense(row['VAR101'], row['VAR102'],
row['VAR11'], row['VAR112'], row['VAR113'], row['VAR116'], row['VAR117'],
row['VAR118'], row['VAR119'], row['VAR126'], row['VAR127'], row['VAR130'],
row['VAR131'], row['VAR158'], row['VAR165'], row['VAR170'], row['VAR35'],
row['VAR36'], row['VAR4'], row['VAR47'], row['VAR48'], row['VAR52'],
row['VAR54'], row['VAR55'], row['VAR56'], row['VAR8'], row['VAR9'],
row['VAR189'], row['VAR242'], row['VAR255'], row['VAR278'], row['VAR279'],
row['VAR285'], row['VAR290'], row['VAR296'], row['VAR297'], row['VAR301'],
row['VAR307'], row['VAR312'], row['VAR330'], row['VAR331'], row['VAR332'],
row['VAR337'], row['VAR338'], row['VAR342'], row['VAR345'], row['VAR347'],
row['VAR348'], row['VAR369'], row['VAR13'], row['VAR14'], row['VAR15'],
row['VAR152'], row['VAR168'], row['VAR169'], row['VAR17'], row['VAR172'],
row['VAR177'], row['VAR18'], row['VAR29'], row['VAR40'], row['VAR51'],
row['VAR53'], row['VAR90'], row['VAR95'], row['VAR96'], row['VAR191'],
row['VAR225'], row['VAR260'], row['VAR264'], row['VAR277'], row['VAR282'],
row['VAR300'], row['VAR302'], row['VAR303'], row['VAR329'], row['VAR339'],
row['VAR340'], row['VAR341'], row['VAR343'], row['VAR344'], row['VAR100'],
row['VAR105'], row['VAR114'], row['VAR115'], row['VAR138'], row['VAR155'],
row['VAR16'], row['VAR183'], row['VAR24'], row['VAR30'], row['VAR31'],
row['VAR32'], row['VAR33'], row['VAR34'], row['VAR39'], row['VAR49'],
row['VAR50'], row['VAR77'], row['VAR80'], row['VAR97'], row['VAR98'],
row['VAR99'], row['VAR283'], row['VAR284'], row['VAR28'], row['VAR94'],
row['VAR148'], row['VAR89'], row['VAR139'], row['VAR159'], row['VAR2'],
row['VAR25'], row['VAR64'], row['VAR91'], row['VAR194'], row['VAR281']]) )

    obj = (row['TARGET'], Vectors.dense(row['VAR100'], row['VAR105'],
row['VAR114'], row['VAR115'], row['VAR138'], row['VAR155'], row['VAR16'],
row['VAR183'], row['VAR24'], row['VAR30'], row['VAR31'], row['VAR32'],
row['VAR33'], row['VAR34'], row['VAR39'], row['VAR49'], row['VAR50'],
row['VAR77'], row['VAR80'], row['VAR97'], row['VAR98'], row['VAR99'],
row['VAR283'], row['VAR284'], row['VAR28'], row['VAR94'], row['VAR148'],
row['VAR89'], row['VAR139'], row['VAR159'], row['VAR2'], row['VAR25'],
row['VAR64'], row['VAR91'], row['VAR194'], row['VAR281']]) )


    return obj


# HERE santanderDF2 IS A PYSPARK DATAFRAME, BUT IS BEING CONVERTED TO AN rdd
AND THEN I'M ABLE TO USE THE map FUNCTION

santanderRDD5 = santanderDF2.rdd.map(transformFeatures)


santanderRDD5.take(2)


# CONVERTING TO DATAFRAME AGAIN

santanderDF3 = spSession.createDataFrame(santanderRDD5, ['TARGET',
'FEATURES'])

santanderDF3.show(10)

santanderDF3.cache()
```

Machine Learning Modeling

Train/Test Split

```
# Now entering the Machine Learning Process

random.seed(6)

(train_data, test_data) = santanderDF3.randomSplit([0.7, 0.3])

print(train_data.count(), "data to train my model")

print(test_data.count(), "data to test my model")

#4924

#2084


# Looking for how much data of 'Satisfied'(0) and 'Unsatisfied'(1) we have in
the train_data

# 01.Pyspark SQL Functions

from pyspark.sql.functions import *


train_data.cube('TARGET').agg(count('TARGET').alias('Qt of
Data')).orderBy('TARGET').show()


# Looking for how much data of 'Satisfied'(0) and 'Unsatisfied'(1) we have in
the test_data

test_data.cube('TARGET').agg(count('TARGET').alias('Qt of
Data')).orderBy('TARGET').show()
```

Creating and Training Model - I

```
# Decision Tree ML Model

from pyspark.ml.classification import DecisionTreeClassifier

from pyspark.ml.evaluation import MulticlassClassificationEvaluator


# Avoiding Underfitting and Overfitting in Decision Trees

for depth in range(1,15):

    model_v1 = DecisionTreeClassifier(maxDepth=depth, labelCol= 'TARGET',
featuresCol= 'FEATURES')
```

```
    model_v1_train = model_v1.fit(train_data)

    model_v1_prediction = model_v1_train.transform(test_data)

    evaluator = MulticlassClassificationEvaluator(predictionCol=
'prediction',

                                                  labelCol = 'TARGET',

                                                  metricName = 'accuracy')

    acc = evaluator.evaluate(model_v1_prediction)

    print('In Depth %.0f this model is %.4f%% Accurate.' %(depth, acc*100))


# Better Depth is 4 -> 76.87%

# Building the model

model_v1 = DecisionTreeClassifier(maxDepth=4, labelCol= 'TARGET',
featuresCol= 'FEATURES')


# Training my Model

model_v1_train = model_v1.fit(train_data)


print(model_v1_train.numNodes, "Nodes")

print(model_v1_train.depth, "Depth")
```

Predictions and Evaluation

```
# Making Predictions with test_data

model_v1_prediction = model_v1_train.transform(test_data)

model_v1_prediction.select('prediction', 'TARGET').collect()


# Evaluating my model

evaluator = MulticlassClassificationEvaluator(predictionCol= 'prediction',

                                              labelCol = 'TARGET',

                                              metricName = 'accuracy')

acc = evaluator.evaluate(model_v1_prediction)

print('This model is %.2f%% Accurate.' %(acc*100))


# Taking the Confusion Matrix
```

```
model_v1_prediction.groupBy('TARGET', 'prediction').count().show()
```

Plotting Confusion Matrix

```
model_v1_prediction.head(2)

type(model_v1_prediction)

# Transforming to Pandas so I'll plot some analysis using Seaborn
pandasDF = model_v1_prediction.toPandas()

# Getting Necessarry Data from model_v1_prediction
Actual = pandasDF.ix[:, 'TARGET']
Predic = pandasDF.ix[:, 'prediction']

data = {'Actual': Actual,
        'Predicted': Predic
        }

confMatDF = pd.DataFrame(data, columns=['Actual', 'Predicted'])
confMatDF.head(5)

# Renaming Target (0-> Satisfied, 1-> Unsatisfied)
def targetTransformation(lst):
    if lst == 1:
        return 'Unsatisfied'
    else:
        return 'Satisfied'

auxList1 = list(confMatDF['Actual'])
auxList2 = list(map(targetTransformation, auxList1))
confMatDF['Actual'] = auxList2
```

```
auxList1 = list(confMatDF['Predicted'])

auxList2 = list(map(targetTransformation, auxList1))

confMatDF['Predicted'] = auxList2


confusionMatrix = pd.crosstab(confMatDF.Actual, confMatDF.Predicted,
rownames=['Actual Value'], colnames=['Predicted Value'] )

confusionMatrix


confusionMatrix.values.sum()


# Looking as Percentage of Churn and No Churn

import numpy as np


# Setting width and height of the plot-figure

plt.figure(figsize=(18,10))


status = ['True Positive', 'False Negative', 'False Positive', 'True
Negative']

val    = [x for x in confusionMatrix.values.flatten()]

percent= ['{:.2%}'.format(x) for x in
(confusionMatrix.values.flatten()/confusionMatrix.values.sum())]


labels = [f'{q1}\n{q2}\n{q3}' for q1, q2, q3 in zip(status, val, percent)]

labels = np.asarray(labels).reshape(2,2)


sns.heatmap((confusionMatrix/confusionMatrix.values.sum()), annot=labels,
fmt='', cmap='BuGn', cbar=False)

sns.set(font_scale=1.5)

plt.title('Confusion Matrix')


# We get pretty good results let's try another ML Models
```

Creating and Training Model - II

```
# Random Forest ML Model
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator


# Building new model
model_v2 = RandomForestClassifier(labelCol = "TARGET", featuresCol =
"FEATURES")
model_v2_train = model_v2.fit(train_data)


# With the trained model I show it new data to make predictions with
test_data
model_v2_prediction = model_v2_train.transform(test_data)
model_v2_prediction.select('prediction', 'TARGET').collect()


# Evaluating my model
evaluator = MulticlassClassificationEvaluator(predictionCol = "prediction",
                                              labelCol = "TARGET",
                                              metricName = "accuracy")
acc = evaluator.evaluate(model_v2_prediction)
print('This model is %.2f%% Accurate.' %(acc*100))
```

Plotting Confusion Matrix

```
model_v2_prediction.head(2)


type(model_v2_prediction)


# Transforming to Pandas so I'll plot some analysis using Seaborn
pandasDF = model_v2_prediction.toPandas()


# Getting Necessarry Data from model_v1_prediction
Actual = pandasDF.ix[:, 'TARGET']
Predic = pandasDF.ix[:, 'prediction']
```

```
data = {'Actual': Actual,
        'Predicted': Predic
        }


confMatDF = pd.DataFrame(data, columns=['Actual', 'Predicted'])

confMatDF.head(5)


# Renaming Target (0-> Satisfied, 1-> Unsatisfied)

def targetTransformation(lst):

    if lst == 1:

        return 'Unsatisfied'

    else:

        return 'Satisfied'


auxList1 = list(confMatDF['Actual'])

auxList2 = list(map(targetTransformation, auxList1))

confMatDF['Actual'] = auxList2


auxList1 = list(confMatDF['Predicted'])

auxList2 = list(map(targetTransformation, auxList1))

confMatDF['Predicted'] = auxList2


confusionMatrix = pd.crosstab(confMatDF.Actual, confMatDF.Predicted,
rownames=['Actual Value'], colnames=['Predicted Value'] )

confusionMatrix


confusionMatrix.values.sum()


# Looking as Percentage of Churn and No Churn

import numpy as np


# Setting width and height of the plot-figure
```

```
plt.figure(figsize=(18,10))


status = ['True Positive', 'False Negative', 'False Positive', 'True
Negative']

val    = [x for x in confusionMatrix.values.flatten()]

percent= ['{:.2%}'.format(x)  for x in
(confusionMatrix.values.flatten()/confusionMatrix.values.sum())]


labels = [f'{q1}\n{q2}\n{q3}' for q1, q2, q3 in zip(status, val, percent)]

labels = np.asarray(labels).reshape(2,2)


sns.heatmap((confusionMatrix/confusionMatrix.values.sum()), annot=labels,
fmt='', cmap='BuGn', cbar=False)

sns.set(font_scale=1.5)

plt.title('Confusion Matrix')


# The first model leads to a better Accuracy and TP and TN higher.

# The second one is giving a FP higher, this means that Unsatisfied Customers
are considered Satisfied leading to a problem because these customers could
leave the company since they aren't being contacted.

# My choice is the first model.
```

Saving My Chosen Model

```
type(model_v1_train)

# As we are in PYSPARK ML MODEL, the following instructions doesn't work:

#import pickle

# Saving my model with the name: train_model_v1.sav

#filename = 'train_model_v1.sav'

#pickle.dump(model_v1, open(filename, 'wb'))


# These instructions are to python Ml Models, so let's see how to save a
PYSPARK ML MODEL:

# Documentation:
https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=sav
e#pyspark.ml.classification.DecisionTreeClassificationModel.save
```

```
# First: save my model (model_v1) in a file ('santander_ML_Model.model')

model_v1_train.write().overwrite().save('santander_ML_Model.model')


# Then: load it using the model type (DecisionTreeClassifier) into a new
variable (model_trained_load)

from pyspark.ml.classification import DecisionTreeClassificationModel

model_trained_load =
DecisionTreeClassificationModel.load('santander_ML_Model.model')


type(model_trained_load)


type(model_v1_train)


# Note they are the same!
```

New and Unseen Data

Simulating a Real World

```
# Now that I have the best model, let's simulate an Environment where new
data is giving to this Machine Learning Model


# Joining both Dataframes in one

pandasDFN = pandasDFN.append(pandasDF1, ignore_index=True)


# Retrieving Not Seen Data as a Real World Environment

type(pandasDFN)


# Dividing pandas Dataframe in 7 Days to acquire 7 batchs

len(pandasDFN)/5


# Simulating as if we have Real-Time Data

pandasDFNbatch = pandasDFN.sample(17000, replace=False)
```

```
#pandasDFNbatch


# Transforming the pandas Dataframe to a PYSPARK Dataframe

santanderDFNbatch = spSession.createDataFrame(pandasDFNbatch)


type(santanderDFNbatch)


# Pre-Processing my Dataset


# Creating an LabeledPoint (target, Vector[features])

# Removing not relevant columns or with low correlation to the model, this
way I choose the columns I want the model to have (just observe the selection
done in Vectors.dense(....))


# ATTENTION TO THIS VERY IMPORTANT CONCEPT!!!!!!!!!

# SOME APACHE SPARK MACHINE LEARNING ALGORITHMS (MAINLY THE REGRESSION TYPE)
NEED DATA TO BE

# IN A SPECIFIC FORMAT TO PERFORM THE TRAINING(FIT).

# TO THIS HAPPEN I NEED TO DELIVER TO THE MODEL THE DATA IN VECTOR FORMAT,
SPECIFICALLY USING A

# DENSE OR SPARSE VECTOR. THIS BECAUSE APACHE SPARK WORKS WITH CLUSTER AND
THEN IT WILL

# DISTRIBUTE THESE DATA BETWEEN MACHINES.

from pyspark.ml.linalg import Vectors


def transformFeatures(row):

    #obj = (row['TARGET'], Vectors.dense(row['VAR101'], row['VAR102'],
row['VAR11'], row['VAR112'], row['VAR113'], row['VAR116'], row['VAR117'],
row['VAR118'], row['VAR119'], row['VAR126'], row['VAR127'], row['VAR130'],
row['VAR131'], row['VAR158'], row['VAR165'], row['VAR170'], row['VAR35'],
row['VAR36'], row['VAR4'], row['VAR47'], row['VAR48'], row['VAR52'],
row['VAR54'], row['VAR55'], row['VAR56'], row['VAR8'], row['VAR9'],
row['VAR189'], row['VAR242'], row['VAR255'], row['VAR278'], row['VAR279'],
row['VAR285'], row['VAR290'], row['VAR296'], row['VAR297'], row['VAR301'],
row['VAR307'], row['VAR312'], row['VAR330'], row['VAR331'], row['VAR332'],
row['VAR337'], row['VAR338'], row['VAR342'], row['VAR345'], row['VAR347'],
row['VAR348'], row['VAR369'], row['VAR13'], row['VAR14'], row['VAR15'],
row['VAR152'], row['VAR168'], row['VAR169'], row['VAR17'], row['VAR172'],
row['VAR177'], row['VAR18'], row['VAR29'], row['VAR40'], row['VAR51'],
row['VAR53'], row['VAR90'], row['VAR95'], row['VAR96'], row['VAR191'],
row['VAR225'], row['VAR260'], row['VAR264'], row['VAR277'], row['VAR282'],
row['VAR300'], row['VAR302'], row['VAR303'], row['VAR329'], row['VAR339'],
row['VAR340'], row['VAR341'], row['VAR343'], row['VAR344'], row['VAR100'],
row['VAR105'], row['VAR114'], row['VAR115'], row['VAR138'], row['VAR155'],
row['VAR16'], row['VAR183'], row['VAR24'], row['VAR30'], row['VAR31'],
row['VAR32'], row['VAR33'], row['VAR34'], row['VAR39'], row['VAR49'],
row['VAR50'], row['VAR77'], row['VAR80'], row['VAR97'], row['VAR98'],
row['VAR99'], row['VAR283'], row['VAR284'], row['VAR28'], row['VAR94'],
row['VAR148'], row['VAR89'], row['VAR139'], row['VAR159'], row['VAR2'],
row['VAR25'], row['VAR64'], row['VAR91'], row['VAR194'], row['VAR281']]) )

    obj = (row['TARGET'], Vectors.dense(row['VAR100'], row['VAR105'],
row['VAR114'], row['VAR115'], row['VAR138'], row['VAR155'], row['VAR16'],
row['VAR183'], row['VAR24'], row['VAR30'], row['VAR31'], row['VAR32'],
row['VAR33'], row['VAR34'], row['VAR39'], row['VAR49'], row['VAR50'],
row['VAR77'], row['VAR80'], row['VAR97'], row['VAR98'], row['VAR99'],
row['VAR283'], row['VAR284'], row['VAR28'], row['VAR94'], row['VAR148'],
row['VAR89'], row['VAR139'], row['VAR159'], row['VAR2'], row['VAR25'],
row['VAR64'], row['VAR91'], row['VAR194'], row['VAR281']]) )


    return obj


# HERE santanderDFN IS A PYSPARK DATAFRAME, BUT IS BEING CONVERTED TO AN rdd
AND THEN I'M ABLE TO USE THE map FUNCTION

santanderRDDNbatch = santanderDFNbatch.rdd.map(transformFeatures)


santanderRDDNbatch.take(2)


# CONVERTING TO DATAFRAME AGAIN

santanderDFNbatch = spSession.createDataFrame(santanderRDDNbatch, ['TARGET',
'FEATURES'])

santanderDFNbatch.show(10)

santanderDFNbatch.cache()


# Making Predictions with New Batch Data

model_trained_load_prediction =
model_trained_load.transform(santanderDFNbatch)

model_trained_load_prediction.select('prediction', 'TARGET').collect()


# Evaluating my model
```

```
evaluator = MulticlassClassificationEvaluator(predictionCol= 'prediction',
                                              labelCol = 'TARGET',
                                              metricName = 'accuracy')
acc = evaluator.evaluate(model_trained_load_prediction)
print('This model is %.2f%% Accurate.' %(acc*100))

# Taking the Confusion Matrix
model_trained_load_prediction.groupBy('TARGET', 'prediction').count().show()
```

Plotting Confusion Matrix

```
model_trained_load_prediction.head(2)

type(model_trained_load_prediction)

# Transforming to Pandas so I'll plot some analysis using Seaborn
pandasDF = model_trained_load_prediction.toPandas()

# Getting Necessarry Data from model_v1_prediction
Actual = pandasDF.ix[:, 'TARGET']
Predic = pandasDF.ix[:, 'prediction']

data = {'Actual': Actual,
        'Predicted': Predic
        }

confMatDF = pd.DataFrame(data, columns=['Actual', 'Predicted'])
confMatDF.head(5)

# Renaming Target (0-> Satisfied, 1-> Unsatisfied)
def targetTransformation(lst):
    if lst == 1:
        return 'Unsatisfied'
    else:

        return 'Satisfied'


auxList1 = list(confMatDF['Actual'])

auxList2 = list(map(targetTransformation, auxList1))

confMatDF['Actual'] = auxList2


auxList1 = list(confMatDF['Predicted'])

auxList2 = list(map(targetTransformation, auxList1))

confMatDF['Predicted'] = auxList2


confusionMatrix = pd.crosstab(confMatDF.Actual, confMatDF.Predicted,
rownames=['Actual Value'], colnames=['Predicted Value'] )

confusionMatrix


confusionMatrix.values.sum()


type(confMatDF)


# Saving confusionMatrix to CSV so I'll plot some results in PowerBi

confMatDF.to_csv('PBI_Analysis1.csv')

acc = pd.DataFrame({'Accuracy': [acc]}, columns=['Accuracy'])

acc.to_csv('PBI_Accuracy1.csv')
```

The End