Z-FINDER PROJECT v0.0.1

“An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism.”

D. Hawkins. Identification of Outliers, Chapman and Hall, 1980.

Tabla de contenido

[WHAT IS Z-FINDER PROJECT? 4](#_Toc492985088)

[QUICK START 5](#_Toc492985089)

[INSTALLATION REQUIREMENTS 5](#_Toc492985090)

[LIBRARY REQUERIMENTS 5](#_Toc492985091)

[HOW TO RUN 5](#_Toc492985092)

[MOTIVATION 6](#_Toc492985093)

[DATA USAGE 8](#_Toc492985094)

[CHALLENGES PRESENTED 9](#_Toc492985095)

[UNNORMALIZED DATA – FUZZY RULES 9](#_Toc492985096)

[MISSING VALUES 10](#_Toc492985097)

[UNSTRUCTURED DATA 10](#_Toc492985098)

[CATEGORICAL DATA 10](#_Toc492985099)

[OUTLIERS 11](#_Toc492985100)

[SMIRNOV-GRUBBS 11](#_Toc492985101)

[PERCENTILE BASED APPROACH 12](#_Toc492985102)

[MEDIAN ABSOLUTE DEVIATION 12](#_Toc492985103)

[OUTLIERS FOR NOVELTY DATA 15](#_Toc492985104)

[HIGH DIMENSIONALITY 17](#_Toc492985105)

[DATA UNDERSTANDING 18](#_Toc492985106)

[FILES PROCESSING 25](#_Toc492985107)

[IDS BOTTLE 25](#_Toc492985108)

[CUSTOMER 27](#_Toc492985109)

[CUSTOMER\_PROPERTY 35](#_Toc492985110)

[DATES 37](#_Toc492985111)

[GUARANTEES 46](#_Toc492985112)

[ASSISTANCE 46](#_Toc492985113)

[PROPERTY 46](#_Toc492985114)

[PAYMENTS 53](#_Toc492985115)

[POLICY 57](#_Toc492985116)

[LOSS ADJUSTER 67](#_Toc492985117)

[SINISTER 72](#_Toc492985118)

[INTERMEDIARY 75](#_Toc492985119)

[CUSTOMER\_OBJECT\_RESERVE 85](#_Toc492985120)

[HISTORICAL\_SINISTER 86](#_Toc492985121)

[HISTORICAL\_POLICY 87](#_Toc492985122)

[HISTORICAL\_OTHER\_POLICIES 87](#_Toc492985123)

[HISTORICAL\_OTHER \_SINISTER 87](#_Toc492985124)

[HISTORICO\_OTHER\_POL\_SINISTER 87](#_Toc492985125)

[BLACKLIST 87](#_Toc492985126)

[AUXILIAR FILES 87](#_Toc492985127)

[DAILY AND WEEKLY PRE PROCESSING 88](#_Toc492985128)

[WEEKLY PRE-PROCESSING 88](#_Toc492985129)

[DAILY PRE-PROCESSING 88](#_Toc492985130)

[PROCESSING TABLE 90](#_Toc492985131)

[1. VARIANCE THRESHOLD 90](#_Toc492985132)

[2. APPEND DATAFRAME 91](#_Toc492985133)

[3. FILL THE NAN VALUES IN THE NEW BOTTLE 92](#_Toc492985134)

[4. DELETE COLUMN BY A SUBSTRING NAME 93](#_Toc492985135)

[5. MULTIOUTPUT FILL NAN STRATEGY 94](#_Toc492985136)

[11. APPEND AND MARK TEST 97](#_Toc492985137)

[6. ROBUST SCALE 98](#_Toc492985138)

[7. PRINCIPAL COMPONENT ANALYSIS 99](#_Toc492985139)

[8. APPEND THE BLACKLIST 100](#_Toc492985140)

[9. CORRELATIONS (OPTIONAL) 101](#_Toc492985141)

[10. NORMAL/ANORMAL/NEW DATASET 102](#_Toc492985142)

[RUNNING THE MODEL 104](#_Toc492985143)

[TRAIN-VALID-TEST 104](#_Toc492985144)

[MODEL APPLICATION 110](#_Toc492985145)

[UNSUPERVISED MODEL 112](#_Toc492985146)

[FRAUD SCORE 112](#_Toc492985147)

[SUPERVISED MODEL 120](#_Toc492985148)

[MACHINE LEARNING IMPLEMENTATION 120](#_Toc492985149)

[CHECKLISTS 121](#_Toc492985150)

[CHECKLIST\_OBLIGATORIAS CLASS 122](#_Toc492985151)

[CHECKLIST 1 122](#_Toc492985152)

[CHECKLIST 2 122](#_Toc492985153)

[CHECKLIST 3 123](#_Toc492985154)

[CHECKLIST 4 124](#_Toc492985155)

[CHECKLIST 5 125](#_Toc492985156)

[CHECKLIST 6a 127](#_Toc492985157)

[CHECKLIST 6b 136](#_Toc492985158)

[CHECKLIST 7 138](#_Toc492985159)

[CHECKLIST\_VALORACION CLASS 140](#_Toc492985160)

[CHECKLIST 14 140](#_Toc492985161)

[UTILS 144](#_Toc492985162)

[FUZZY RULES 144](#_Toc492985163)

[DATAFRAME UTILS 144](#_Toc492985164)

[FRAUD SCORE 147](#_Toc492985165)

[GRAPH NETWORKS 147](#_Toc492985166)

[MODEL UTILS 147](#_Toc492985167)

[OUTLIERS 149](#_Toc492985168)

[READ\_CSV 152](#_Toc492985169)

[TO\_DOCX 153](#_Toc492985170)

[UNIVARIATE ANALYSIS 153](#_Toc492985171)

[PROCESS\_UTILS 153](#_Toc492985172)

[TRAIN\_TEST\_UTILS 153](#_Toc492985173)

[WRITE CSV 153](#_Toc492985174)

[CHECKLISTS 153](#_Toc492985175)

[EUROPA 154](#_Toc492985176)

[STRING 155](#_Toc492985177)

[FUTURE CHALLENGES 156](#_Toc492985178)

[GRAPH NETWORKS 156](#_Toc492985179)

[TEXT MINING 156](#_Toc492985180)

[ANNEX 157](#_Toc492985181)

[OTHER RESULTS 157](#_Toc492985182)

# WHAT IS Z-FINDER PROJECT?

Z-FINDER (Fraudulent Insurance-Claims Detection Research) is a tool created to predict fraudulent insurance Property claims. Basically, it uses Machine-Learning Semi-Supervised algorithms to give a fraud probability to each opened sinister. It is programmed in Python language.

# QUICK START

All the information needed is included in the README.md, setup.py, setupy.cfg and LICENSE files. Please, refers to them before start.

## INSTALLATION REQUIREMENTS

It is required to have installed Python 3.x or superior.

## LIBRARY REQUERIMENTS

You can check the setup.py file. It is mandatory to have installed pandas 0.19 or superior, scikit-learn 0.18 or superior, numpy, matplotlib and seaborn libraries.

## HOW TO RUN

From cmd (in the program path):

*python z\_finder\main.py*

It will run the whole model, using the saved processed bottles.

# MOTIVATION

The problem to be solved refers to fraud detection in property insurance claims. In general terms, insurance fraud can be interpreted in two ways: providing untruthful or partial information in the policy, or submitting claims based on misleading or untruthful circumstances (including exaggerating). Fraud can be committed by the policyholder or by a third party claiming against the insurer about a sinister.

But, what is the cost of insurance fraud? Detected and undetected fraud is estimated to represent up to 10% of all claims in Europe (The Impact of Insurance Fraud, 2013). It accounts around 10%-19% of the paid bill. However, fraud varies between countries. For example in UK, ABI statistics released in 2016 estimated that fraud on average, adds an extra £50 to the annual payment. In Italy, according to the Instituto per la Vigilanza Sulle Assicuarazioni, the claims expenditure has increased 21% in the period from 2004 till 2014. In France, the Agence pour la Lutte Contre la Fraude à l’Assurance estimates that fraud represents 15% of claims paid. In Germany, the German Insurance Association calculated the fraud cost in €4bn per year. The ICEA (Investigación Cooperativa entre Entidades Aseguradoras y Fondos de Pensiones) calculated €545 million in claimed costs in 2016, with an increased number of claims of 16% respect to 2014.

The main insurance services are automobile and property insurances. The insured European paid €133 billion in automobile premiums and €93 billion in property premiums in 2015 out of a total of more than €344 billion. €103 billion and €53 billion out of €204 billion were the cost claim of both, respectively, i.e., a 76% of the claimed costs.

While there are many studies in literature on automobile fraud detection (Artís, et al, 1999; Artís, et al 2002; Viane, et al, 2007; Wilson, J.H., 2009; Nian, et al, 2016, etc.) property fraud is undeveloped. Even more, property insurances are not comparable because of the particular characteristics (it is a fixed object with a variety and different typology of sinister). In addition, the detection of fraud is more difficult because witnesses are infrequent or are usually cohabitants.

For this reason, our main objective will be to present a variety of Semi-supervised Machine Learning models applied to property insurance claims. Our goal is being capable of understanding the property fraud behavior using the best suitable techniques. With this in mind, we will compare the results with Supervised and Unsupervised models to determine the relative accuracy gain in terms of prediction error. As an additional point, we will be applying several innovative techniques of advanced analytics programming to manage a variety of misleading data, unstructured data and high-dimensionality problems.

But why do we think Semi-Supervised models are better in this case? Statistically speaking, fraud is a special case of outliers. Outliers are points in the dataset that are significantly different from the remaining data. That is, they are abnormal data. Fraud claims are sequences of multiple data points, rather than individual data points. This type of anomalies are referred to as collective anomalies, because they can only be inferred collectively from a set of data point sequences. Such collective anomalies are often a result of unusual events that generate anomalous patterns of activity. This implies a fundamental problem. We do not just have to separate data between inliers and outliers, but we also have to separate outliers from noise.

Suppose we use unsupervised models. It would be subjective what we define as outlier or noise. However, we only want to focus on the significantly interesting deviations. If we use unsupervised models we have to represent the noise as a boundary between normal data and true anomalies. In definitive, noise is a weak form of outliers that does not meet the strong criteria for been an outlier. This means that we are setting noise as semantics. We really do not know which ones are noise or which one are outliers.

But why not use Supervised Models? Because in general, in fraud detection there exists a great problem of misclassification claims (Artís et al, 2002). In practice, the process of fraud detection is done in two steps. The first step is deciding if a claim is suspicious or not (Viane, et al, 2007). In the second step the Investigation Fraud Area analyzes only the suspicious case and determines if there is fraud or not. This process implies that unsuspicious cases are never studied, which is reasonable in terms of efficiency if we cannot automatize the process. The adjuster has no time to perform an extensive investigation. But it gives us partial information, because we only have labels for a small and partial sample. Using a Supervised model in this case will return bias into the confusion matrix. Mainly, it will have a severe bias in false negatives, and therefore we will be predicting many cases as not fraudulent, while in reality they are. If we use supervised algorithms we will be assuming that the first process is capable to discern perfectly when a claim is a fraud or not, something that infrequently happen. This is referred to in literature as omission error (Bollinger and David, 1997; Poterba and Summers, 1995).

However, the information provided in the suspicious cases is more likely to be specified correctly once it has passed the first stage. It provides us with useful information about a part of the distribution and which is why it is very important to take this information into account. With this in mind, it is notorious that fraud detection can be considered as a Semi-Supervised problem because the ground-truth labeling of data is partially known.

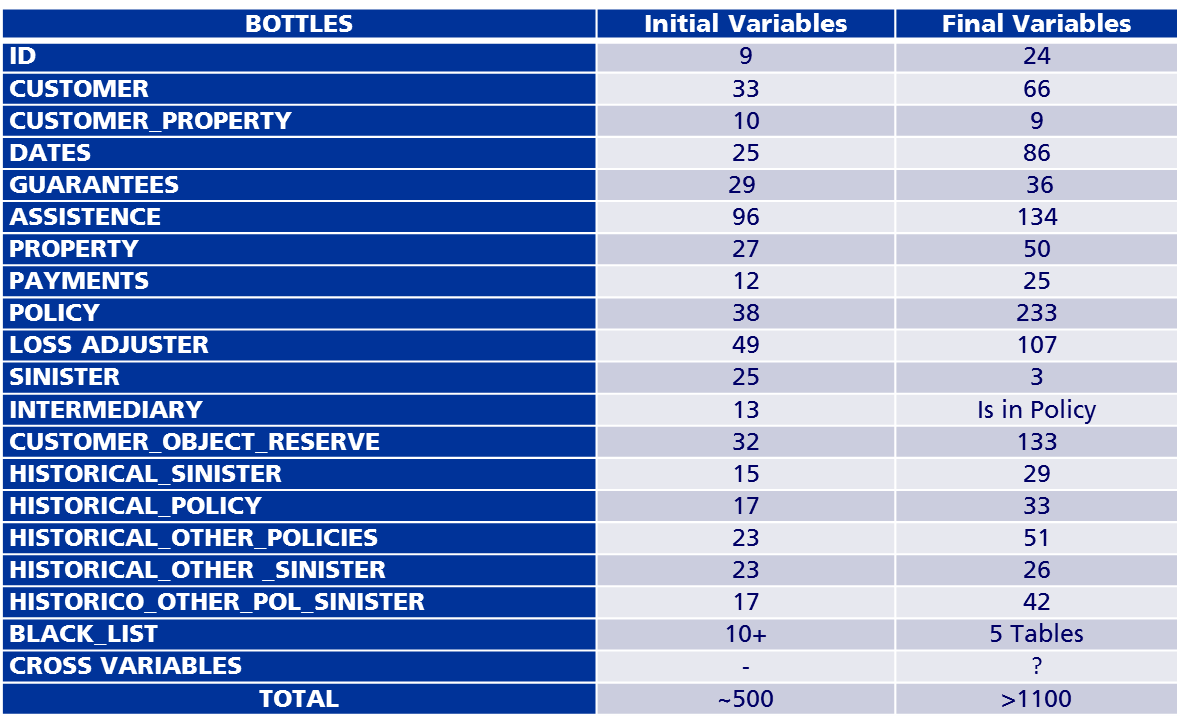
Based on the previous, our contribution can be divided in two key results. First, we are applying innovative Semi-Supervised techniques on fraud detection for the first time, and also applying a model for Property Insurance fraud for the first time. Second, we show the potential of applying Semi-Supervised Models by comparing it to Supervised and Unsupervised models, together with the construction of an automatized algorithm to predict fraud.

# DATA USAGE

Our analysis will be focused on Property Insurance Fraud using data provided by a leader insurer company in Spain during the period 2015-2016. Our main sample consists of more than 300 thousand property sinister for which some of them have been analyzed as possible fraud.

From these cases, the Investigation Office analyzed 1936 claims as suspicious, resolving 1561 during the period. These positive and negative cases will be the base of transforming an unsupervised model into a semi-supervised.

On this version, we collected 18 bottles of interrelated data and one blacklist for property sinister. Following you can find the bottles and the variables generated.



# CHALLENGES PRESENTED

This variables have several problems that we have to treat before any analysis (or at least to keep it in mind to treat later). Basically they can be summarized as follows:

1- Unnormalized data

2- Missing Values

3- Unstructured Data

4- Categorical Data

5- Existence of outliers

6- High Dimensionality

## UNNORMALIZED DATA – FUZZY RULES

When we refer to normalization we have to distinguish the semantic of the concept. For one side, we talk about normalization as the process to apply to quantitative attributes in order to eliminate any scale effect.

But the real problem emerge with the other type of normalization and it is referred to bad typo on Number of Identification, Addresses and Names of customer. It is essential to have a well identification of the fraudster and the object related. If we cannot recognize his patterns over the time or we cannot relate him with other sinister, it is probably that the prediction would be erroneous.

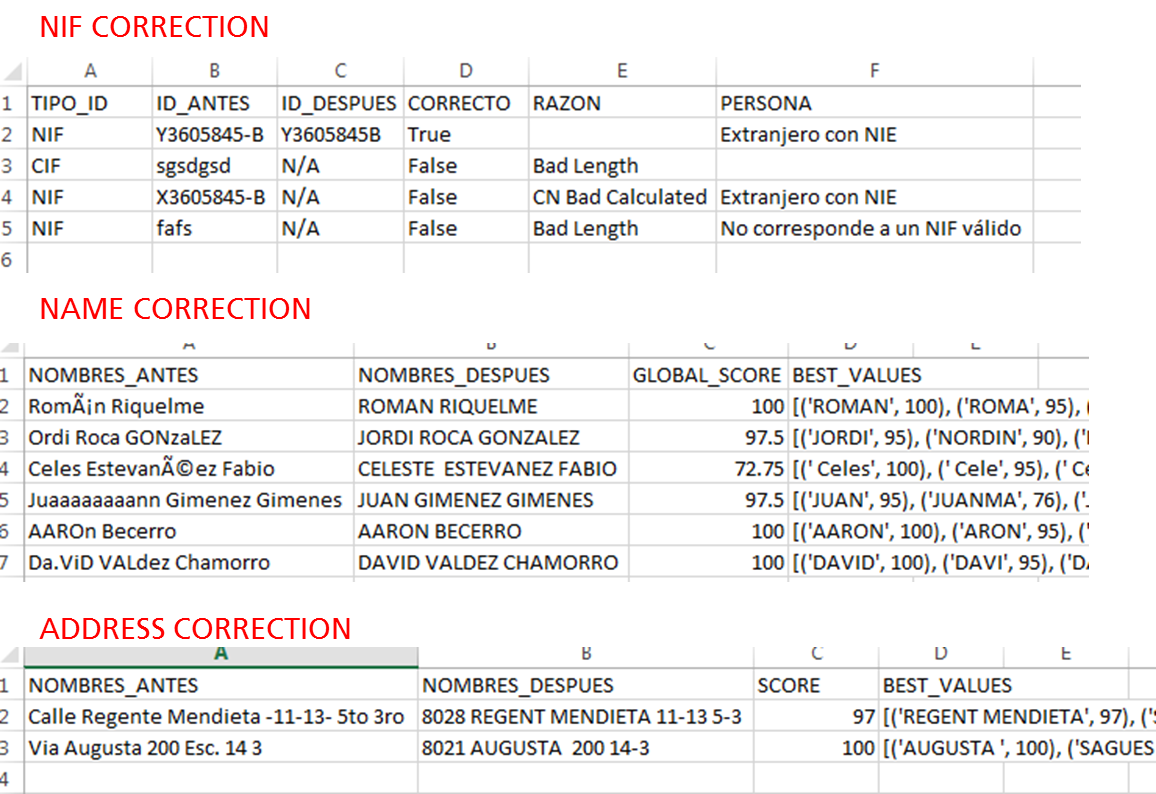
This problematic is consequence of several data entry errors. Bad typos of IDs, text field that are free text and not control over what the policy holder declared.

What we did here, it was to create a Library using Python that at this moment is in process of testing. Basically, using fuzzy rules algorithms (Chiu, S.L., 1994) based on cluster estimations it corrects the three types of variables.

-IDs: Using the calculation rules of NIFs and CIFs stipulated by law.

-Addresses: Using the database of Callejero Censo Electoral from Spain it search the correspondent property by Postal Code.

-Names and Surnames: Using the database of Names Frequencies by the INE we approximate bad typo names by geographical region.



You can also find the library integrated to the Z-FINDER as an additional class, and it is applied to several bottles during the process.

## MISSING VALUES

During the pre-processing stage we are not going to treat this relevant problem. We want to keep it as pure as possible. We are going to solve this in the [Multioutput Strategy](#multioutput) section.

## UNSTRUCTURED DATA

At the moment, we do not present any solution to how to work with unstructured data. Please, refer to the [Future Challenges](#futurechallanges) section.

## CATEGORICAL DATA

As we are going to use Principal Component Analysis (which in definitive it is a linear approach), we need to give sense to the data. Every categorical data has to be transformed in some way.

As you will see later, in general we follow the same approach. Basically we create dummies to each category if they do not have any order sense. It is generated as a one to one category-dummy or it is grouped by certain ranges.

Also, for certain variables, we create a null value dummy to identify some missed categories.

## OUTLIERS

The outlier detection problem is one of the most important challenges here. Basically because fraud detection problem is an outlier problem.

Therefore, we construct a special module in the Z-FINDER project. This module permits calculate up to three different approaches of outlier detection:

-Smirnov-Grubbs Approach.

-Percentile Based Approach.

-Median Absolute Deviation.

### SMIRNOV-GRUBBS

Find the upper critical value of the t-distribution with n-2 degrees of freedom and a significance level of alpha/2n (for two-sided tests) or alpha/n (for one-sided tests). Use this t value to find the score with the following formula:

((n-1) / sqrt(n)) \* (sqrt(t\*\*2 / (n-2 + t\*\*2)))

In simple words, it is assuming a normal distribution, and basically it is trying to find the maximum residuals from the normal distribution.

max\_smirnov = smirnov\_grubbs.max\_test\_outliers(file\_df\_col, alpha=0.10)  
min\_smirnov = smirnov\_grubbs.min\_test\_outliers(file\_df\_col, alpha=0.10)  
  
**if** max\_smirnov:  
 max\_thresold = min(max\_smirnov)  
 name\_max = str(col\_name) + '\_max\_smirnov'  
 file\_df[name\_max] = pd.Series(0, index=file\_df.index)  
 file\_df[name\_max] = file\_df.apply(  
 **lambda** x: 1  
 **if** x[col\_name] < max\_thresold  
 **else** 0, axis=1)  
**if** min\_smirnov:  
 min\_thresold = max(min\_smirnov)  
 name\_min = str(col\_name) + '\_min\_smirnov'  
 file\_df[name\_min] = pd.Series(0, index=file\_df.index)  
 file\_df[name\_min] = file\_df.apply(  
 **lambda** x: 1  
 **if** x[col\_name] < min\_thresold  
 **else** 0, axis=1)

The subjacent problem here is that we have several variables that cannot be approximated to a Normal Distribution, so it does not work very well.

### PERCENTILE BASED APPROACH

This is probably the simplest approach. It consists in check the left and right tails of the distribution (keeping 95% of it), and consider everything outside as an outlier. But also the problem is that we are assuming some kind of normal distribution. With skewed distributions, you'll tend to expect more observations being marked at one end than the other.

**def percentile\_based\_outlier**(data, threshold=95):  
 diff = (100 - threshold) / 2.0  
 minval, maxval = np.percentile(data, [diff, 100 - diff])  
 **return** (data < minval) | (data > maxval)

### MEDIAN ABSOLUTE DEVIATION

This metodologhy is based in the paper “Detecting outliers: Do not use standard deviation around the mean, use absolutedeviation around the median (Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, Laurent Licata) 2013.

It is also a simple approach but it is a very robust measure. Theoretically, it seems to be the best alternative among the three proposed. It is more resilient to outliers in a data set than the standard deviation. In the standard deviation, the distances from the mean are squared, so large deviations are weighted more heavily, and thus outliers can heavily influence it. In the MAD, the deviations of a small number of outliers are irrelevant.

Because the MAD is a more robust estimator of scale than the sample variance or standard deviation, it works better with distributions without a mean or variance.

Also, empirically, when we have tested our variables it seemed to be the most robust.

**def mad\_based\_outlier**(points, thresh=3.5):  
 **if** len(points.shape) == 1:  
 points = points[:, **None**]  
 median = np.median(points, axis=0)  
 diff = np.sum((points - median) \*\* 2, axis=-1)  
 diff = np.sqrt(diff)  
 med\_abs\_deviation = np.median(diff)  
  
 modified\_z\_score = 0.6745 \* diff / med\_abs\_deviation  
  
 **return** modified\_z\_score > thresh

As you can see, we are taking the absolute distance between the points and the median value of the univariate distribution. Also, in the formula above we are using the probable error notion. The probable error defines the half-range of an interval about a central point for the distribution, such that half of the values from the distribution will lie within the interval and half outside.

Let me show an example.

Suppose we have an array of points like this:

Points = [[1], [5], [8], [3], [4], [2], [2], [3]]

Note, that this has to be a nested array, so we can take each point specifically.

In the next step we take the median of the points:

Median = 3.0

Then, we calculate the quadratic distance between each point and the median.

Diff = [4, 4, 25, 0, 1, 1, 1, 0]

We take the squared:

Diff = [2, 2, 5, 0, 1, 1, 1, 0]

Now, diff is the absolute median distance for each point.

Therefore, the Median value for the absolute distances (the median of Diff). This is = 1.

We want now to establish which is the absolute distance between the point and the median value that has to be consider as outlier. MAD can be defined as:

MAD = b \* Median Value Absolute Distances

Where b is a constant linked to the assumption of normality of the data, but disregarding the abnormality induced by outliers. This value tends to be 1.4826 which represents 1/sigma. Where sigma, using the probable error notion, is 0.6745[[1]](#footnote-1).

The rejection criteria is a subjective aspect, depending on the stringency of the researcher.

We can rewrite the MAD as follows to be understandable in our code:

MAD =

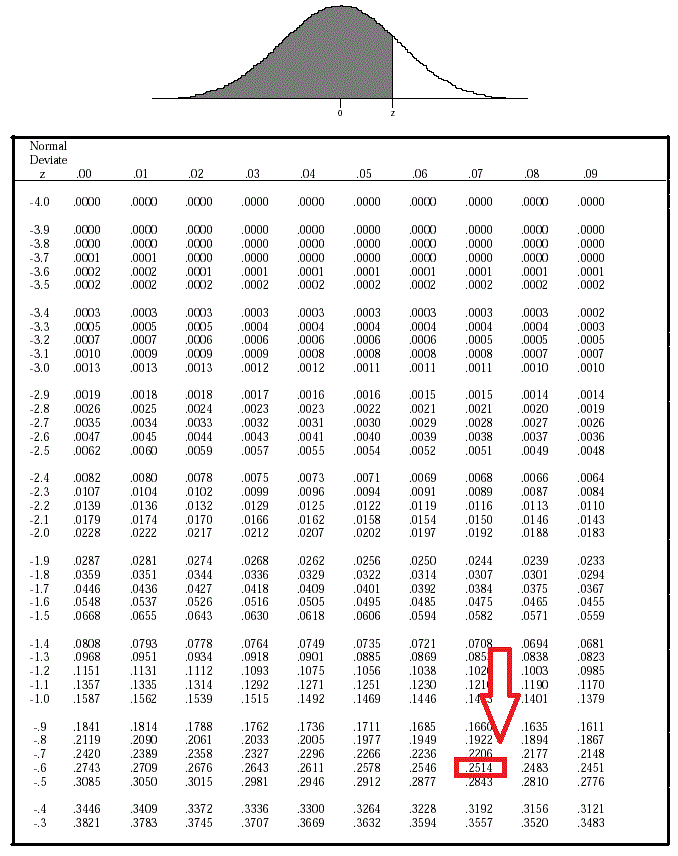
What it is the same as:

MAD = 1.4826 \* 1

This MAD, basically represents that the 50% of the distribution to both sides of the Median is between (-0.6745; 0.6745).

Therefore, our threshold has to be a multiplicative value of this MAD.

It will depend in which stringent we want to be, and then use a normal distribution table:



As you can see, with 0.67 we are leaving 25% of the distribution in both sides. If we want to set another threshold we have to increase the MAD value (that just set us in the 50% of the distribution). This is:

Median – threshold\*MAD < Point < Median – threshold\*MAD

This is the same as:

|Point – Median|/ MAD > threshold

What is the same as our code:

Diff \* 0.674/ med\_abs\_deviation > threshold

Basically we are saying that with a threshold of 3.5 (a moderately conservative and recommended value) we are an in the 2.345 (3.5\*0.67) z-value, and therefore we consider as outlier each 1% of both tails.

### OUTLIERS FOR NOVELTY DATA

As you will see later, when we receive the new bottles (bottles which come from the daily process) we really do not know if they are Outliers. We cannot compare only with the new sinister, we have to retrieve the historical data from the original database to compare if they are really outliers.

For that, we have created an addition method called outliers\_test\_values and auxiliar method called mad\_based\_outlier\_parameters.

The auxiliary method is as follows:

**def mad\_based\_outlier\_parameters**(points):  
 **if** len(points.shape) == 1:  
 points = points[:, **None**]  
  
 median = np.median(points, axis=0)  
  
 diff = np.sum((points - median) \*\* 2, axis=-1)  
  
 diff = np.sqrt(diff)  
  
 med\_abs\_deviation = np.median(diff)  
  
 **return** median, med\_abs\_deviation

Essentially, it gets the median of a column and the mean absolute deviation from that column.

The outliers\_test\_values method is as follows:

**def outliers\_test\_values**(file\_df, base\_df, col\_name, not\_count\_zero=**True**, just\_count\_zero=**False**):  
  
 # base\_df  
 base\_df\_col = base\_df[col\_name].dropna()  
 base\_df\_col = pd.to\_numeric(base\_df\_col, errors='coerce')  
  
 base\_df\_col = base\_df\_col.fillna(base\_df\_col.median())  
  
 **if** not\_count\_zero:  
 base\_df\_col = base\_df\_col[base\_df\_col > 0]  
 **if** just\_count\_zero:  
 base\_df\_col = base\_df\_col[base\_df\_col >= 0]  
  
 base\_df[col\_name] = pd.to\_numeric(base\_df[col\_name], errors='coerce')  
 base\_df[col\_name] = base\_df[col\_name].fillna(base\_df[col\_name].median())  
  
 # test df  
 file\_df\_col = file\_df[col\_name].dropna()  
 file\_df\_col = pd.to\_numeric(file\_df\_col, errors='coerce')  
 file\_df\_col = file\_df\_col[col\_name].dropna()  
  
 **if** not\_count\_zero:  
 file\_df\_col = file\_df\_col[file\_df\_col > 0]  
 **if** just\_count\_zero:  
 file\_df\_col = file\_df\_col[file\_df\_col >= 0]  
  
 # MAD  
 median, med\_abs\_deviation = Outliers.mad\_based\_outlier\_parameters(base\_df\_col)  
 **if** len(file\_df\_col.shape) == 1:  
 points = file\_df\_col[:, **None**]  
  
 diff = np.sum((points - median) \*\* 2, axis=-1)  
 diff = np.sqrt(diff)  
 modified\_z\_score = 0.6745 \* diff / med\_abs\_deviation  
  
 outliers\_mad = modified\_z\_score > 3.5  
  
 list\_outlier = []  
 **for** ax, func **in** zip(file\_df\_col, outliers\_mad):  
 **if** func: # True is outlier  
 list\_outlier.append(ax)  
 list\_outlier = set(list\_outlier)  
 name = str(col\_name) + '\_mad\_outlier'  
 file\_df[name] = pd.Series(0, index=file\_df.index)  
 file\_df[name] = file\_df.apply(  
 **lambda** x: 1  
 **if** x[col\_name] **in** list\_outlier  
 **else** 0, axis=1)  
  
 **return** file\_df

It is similar to the Median Absolute Deviation method viewed in the last section. The basic difference is that first it gets the median and the absolute median deviation from a base dataframe (our bottle from weekly process) and then calculate the modified score using that values in the new column (the new data column). This will tell us if the new sinister is an outlier compared to the history of that sinister.

## HIGH DIMENSIONALITY

You can see the dimensionality reduction strategy in the [Variance Threshold](#variancethreshold) and the [PCA](#pca) section.

# DATA UNDERSTANDING

This was the first part of the Project. We have to analyze the univariate distribution of each variable to get the implicit main insights.

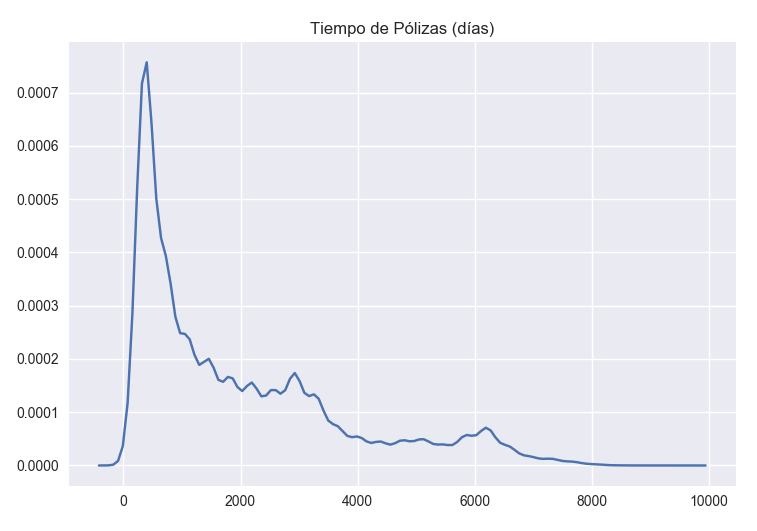
To do that, we constructed two modules in the Z-FINDER project. They are almost the same, but they differ in the output format.

-univariate\_analysis.py contains an extensive class that analysis every dimension of a specific variable (or a range of variables). Its output is shown in the console.

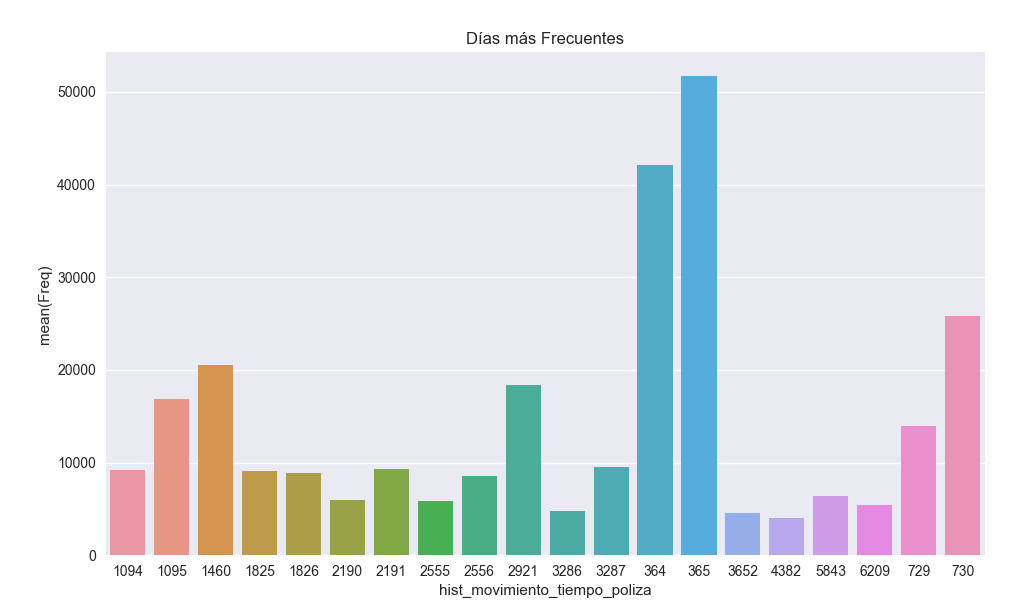
-to\_docx.py contains the same information. But it uses Python docx library to also make a .doc output file.

We are going to show with examples the output generated for each option of the methods inside of both modules.

a) **graph:** It makes a plot of the variable distribution using Seaborn Library.



b) **string\_col:** If it is a categorical variable, it constructs a frequency plot and its corresponding table.





c) **id\_variable + name\_id:** With this option it is possible to get a sort and a bivariate representation of the key variable with other columns.

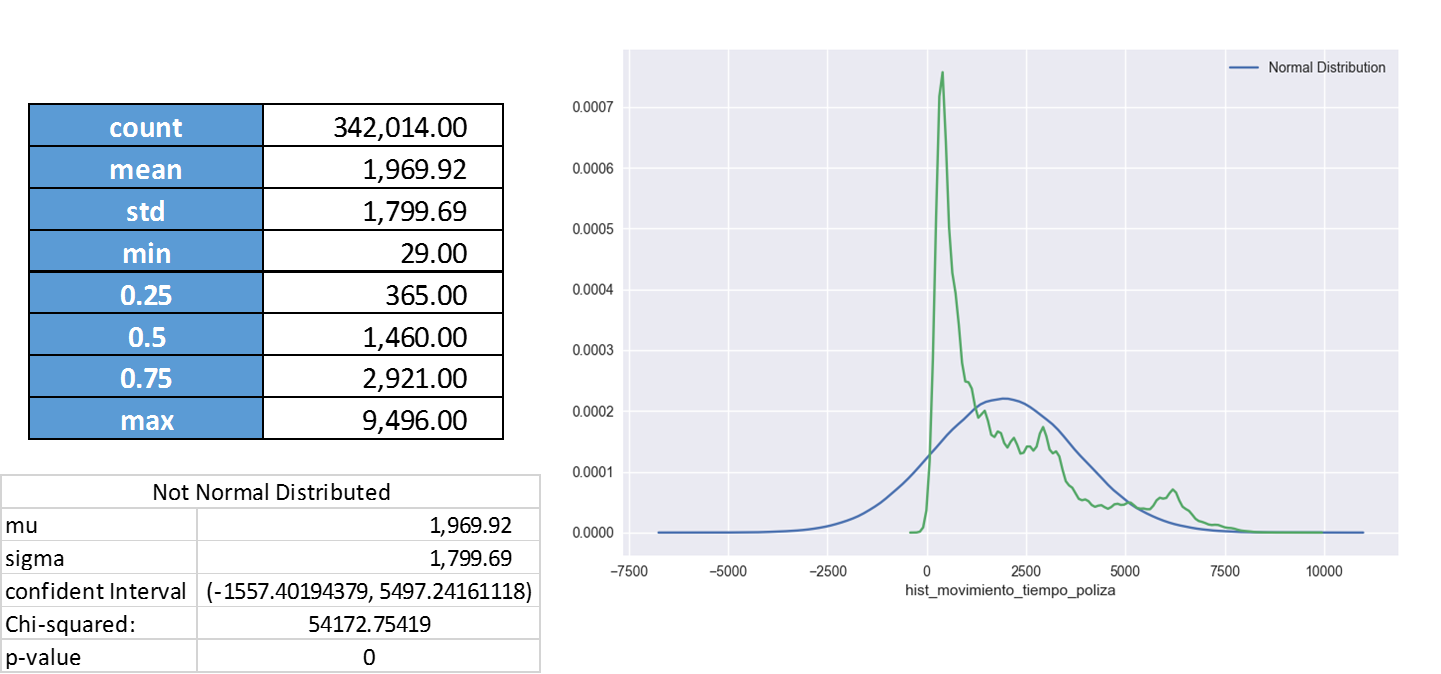




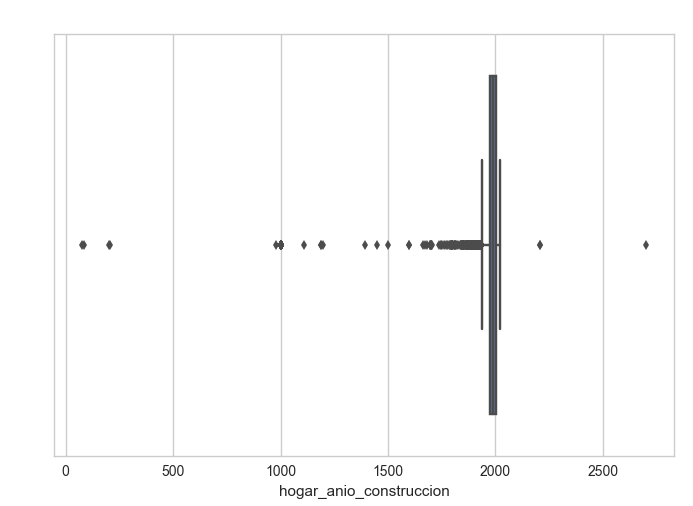
d) **null\_analyisis:** We get the null distribution of each variable and we can get the specific cases to analyze the data consistency.

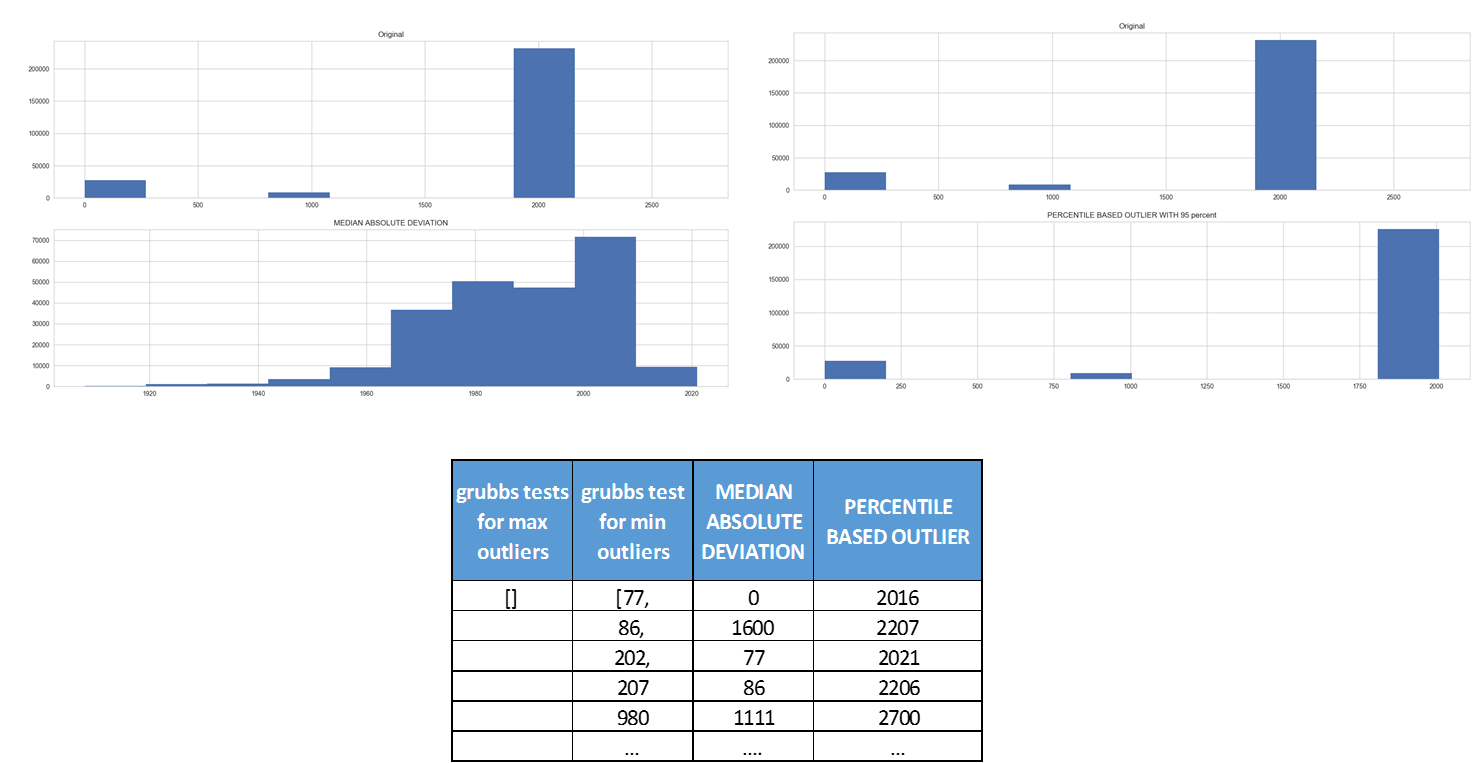


e) **graph\_numerical\_values:** It cleans not numerical data and get basic statistics from the original distribution (mean, std, quartiles, etc.). Also, it makes a Chi-Squared test to evaluate the proximity to a normal distribution.

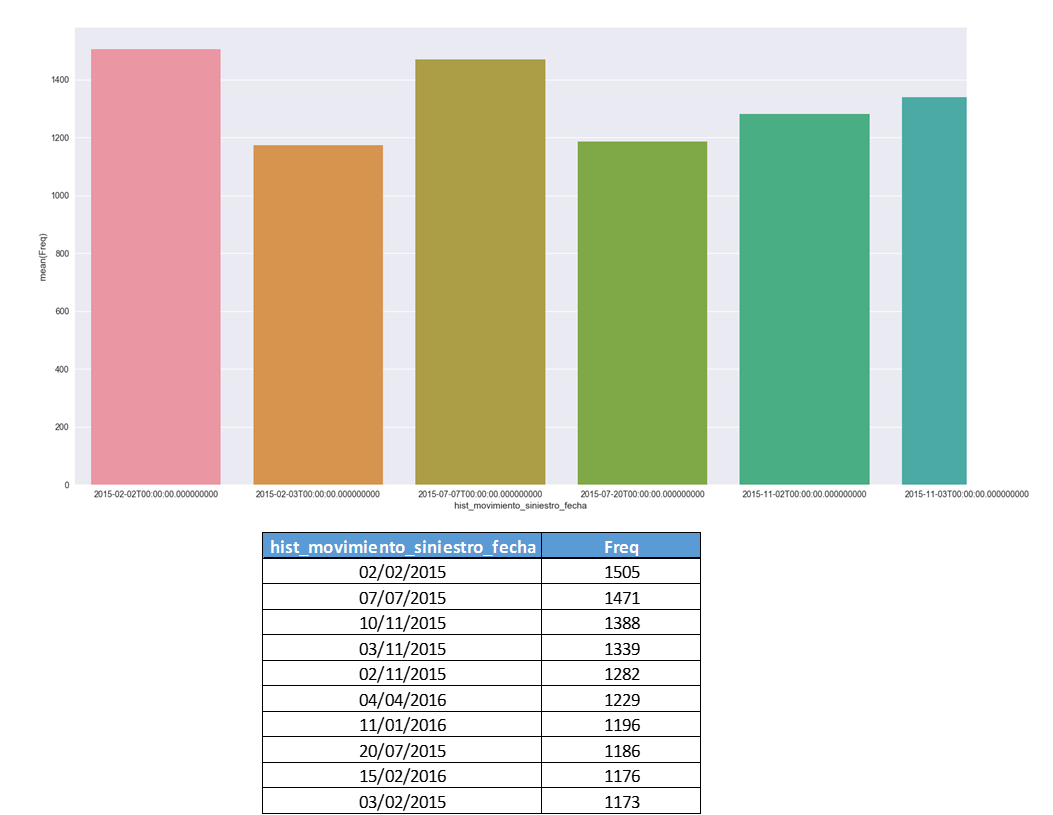


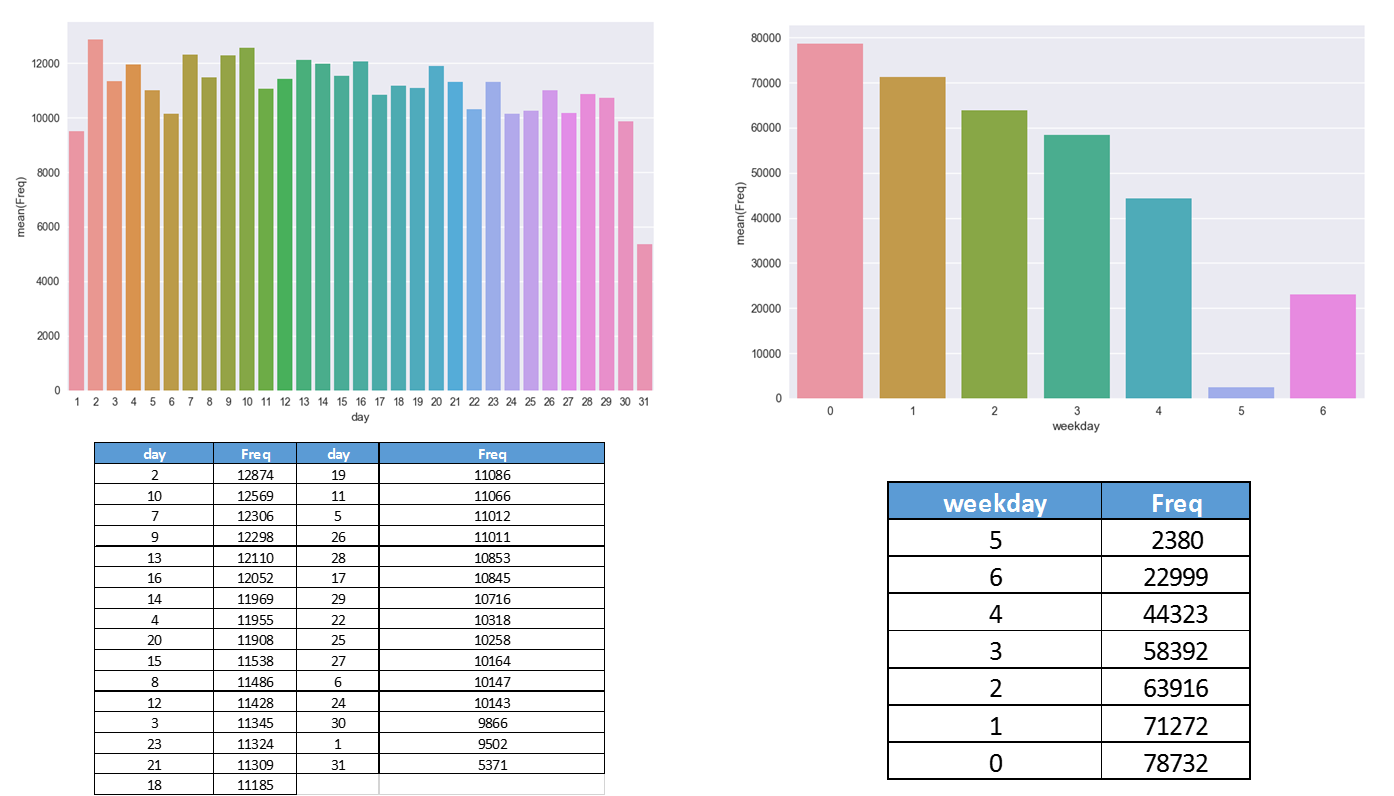
f) **outiler\_analysis:** First a boxplot is drafted. Then, it evaluates using the outliers methods presented before (see [Outliers Section](#outlier)). Then, it plots the original distribution versus the not-null distribution.

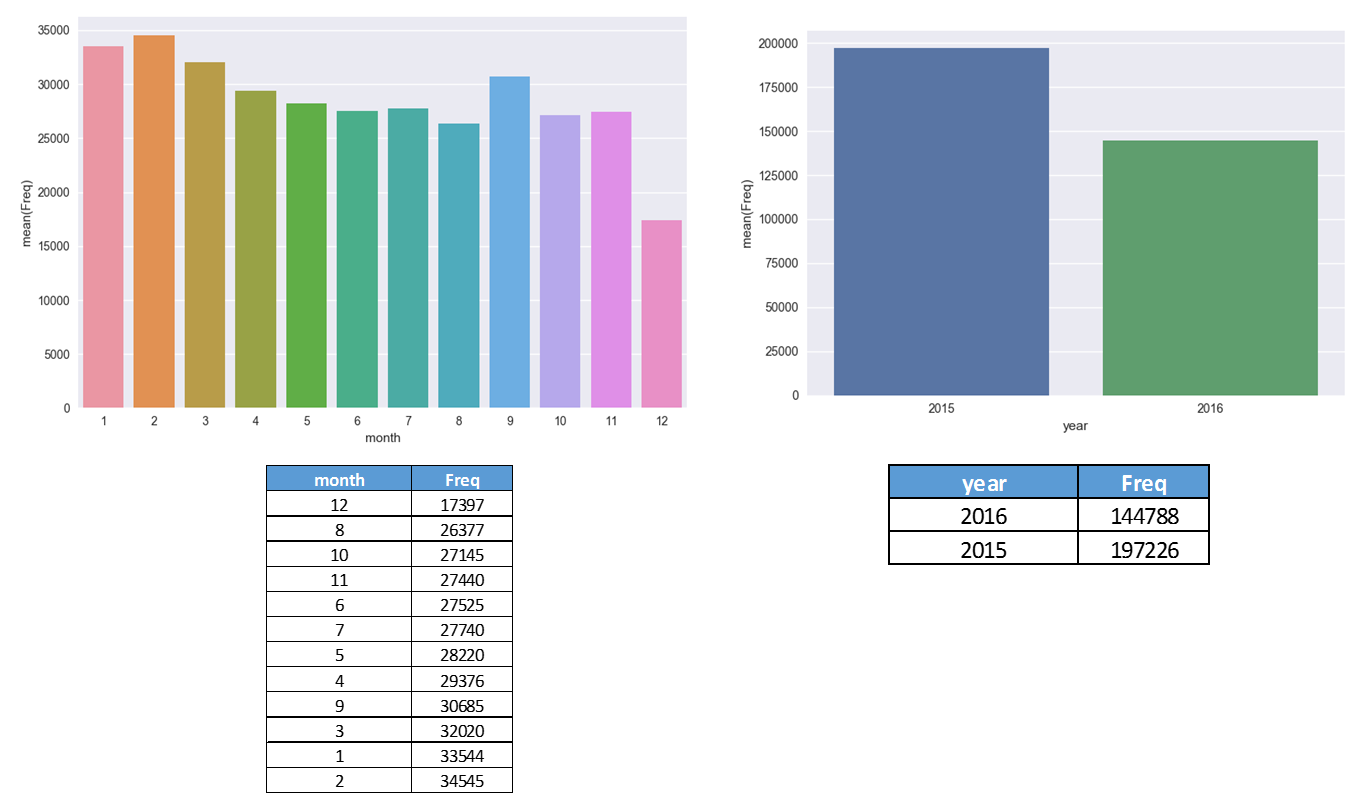




g) **date\_analyisis**: If it is a daytime variable, this option get different groupings of the reference column: By day, by month, by year, by month-year, by weekday, by weekday-month.

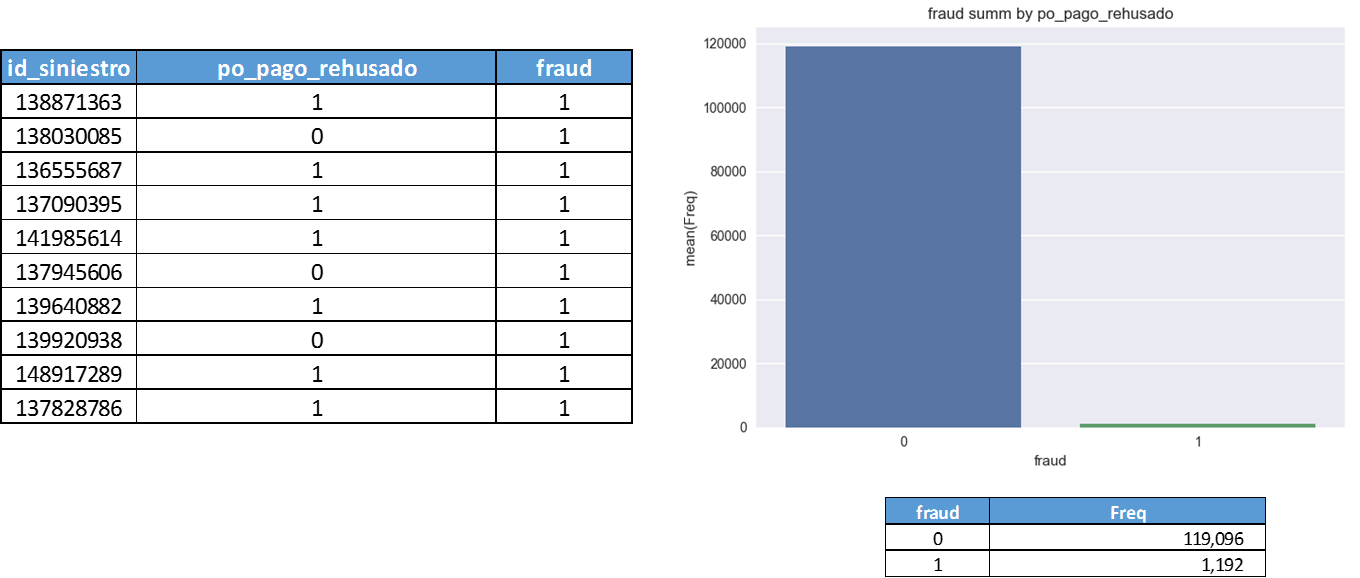


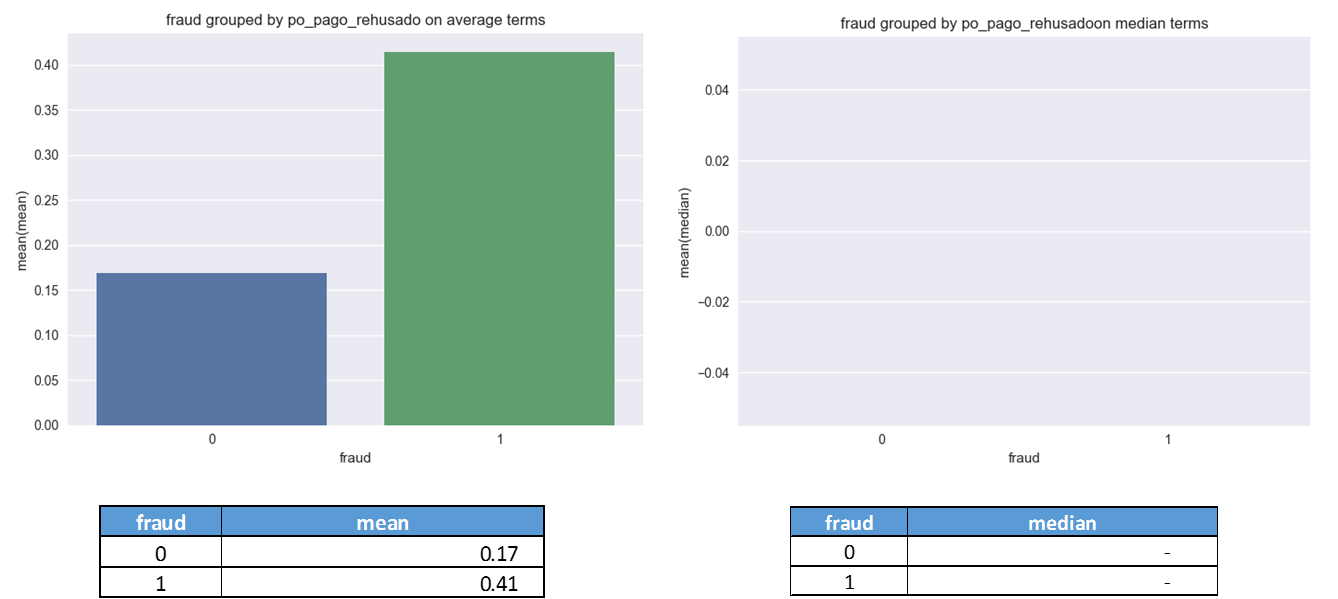


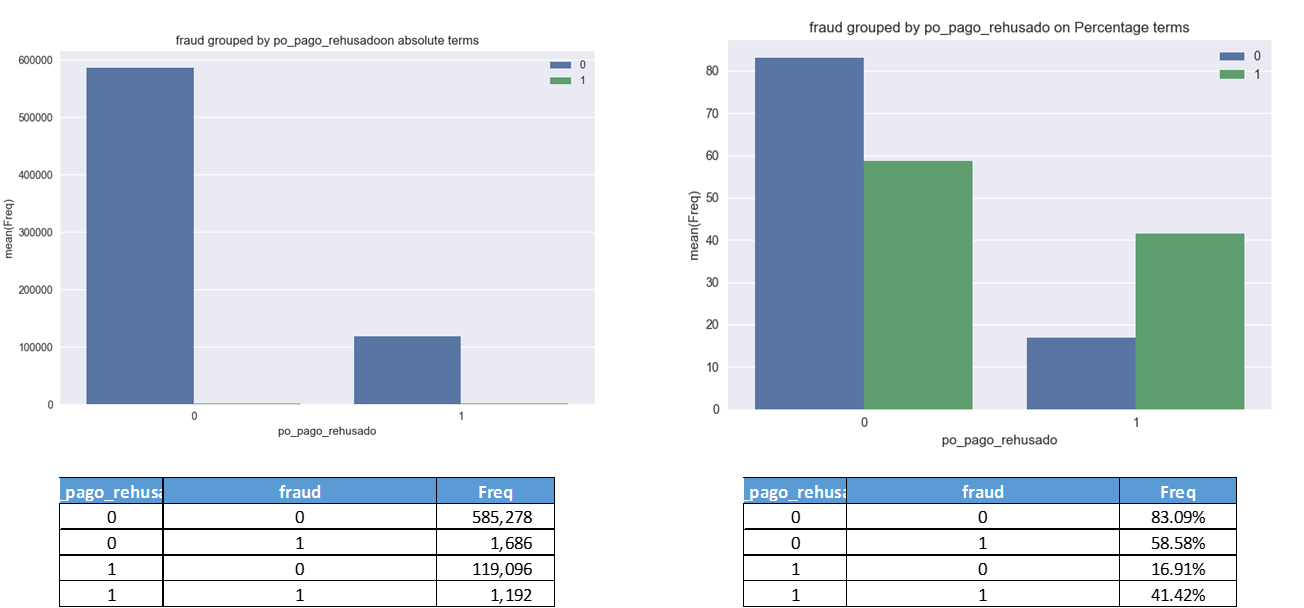




h) **fraud\_analysis:** This is the most complex. Using the Blacklist, we cross the column with the fraud cases. It finds the correlations between them (weighted and in absolute terms) and make several plots.







# FILES PROCESSING

In this section, we are going to do a briefly review of each bottle involved in the final process.

## IDS BOTTLE

In ID we obtain variables that will work as keys in the other Tables. Basically they are ids associated to the client and the sinister.

We start with the following variables:

|  |  |
| --- | --- |
| **Nombre** | **Descripción** |
| id\_siniestro | Número de siniestro |
| id\_fiscal | Identificador fiscal del tomador de la póliza asociada al siniestro |
| id\_poliza | Número de póliza del siniestro |
| id\_producto | Código de producto técnico de la póliza asociada al siniestro |
| id\_dossier | Código de dossier de Europa |
| poliza\_entidad\_legal | Entidad legal de la póliza |
| cliente\_clase\_persona\_codigo | Tipo de persona (F,J) |
| cliente\_tipo\_documento | Tipo de Documento |
| cliente\_codfiliacion | Código filiación del tomador |

The reference module that process IDs is called ids.py using the method ids() explained below:

1- First we load the file and transform it as a Dataframe:

# 1) Load File  
file = ReadCsv.load\_csv(STRING.id\_file)  
  
# 2) Transform file to DF  
file\_df = DfUtils.processing\_file(file)

2- We map the key variables so they have a homogeneous format during the whole process:

# Map Important Variables  
file\_df['id\_siniestro'] = file\_df['id\_siniestro'].map(int)  
file\_df['id\_fiscal'] = file\_df['id\_fiscal'].map(str)  
file\_df['id\_poliza'] = file\_df['id\_poliza'].map(str)

3- We create a dummy to distinguish between juridical or physic person:

# PERSON TYPE: If it is a juridic person, dummy\_fisica = 1  
file\_df['dummy\_fisica'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['cliente\_clase\_persona\_codigo'] == 'F', 'dummy\_fisica'] = 1  
**del** file\_df['cliente\_clase\_persona\_codigo']

4- We generate dummies for each type of existing product type:

# PRODUCT TYPE: We create a dummy by product  
dummy\_producto = pd.get\_dummies(file\_df['id\_producto'], prefix='d\_producto')  
file\_df = pd.concat([file\_df, dummy\_producto], axis=1)  
**del** dummy\_producto  
**del** file\_df['id\_producto']

5- We do the same for distinguish between entities:

# ENTITY TPYE: We create a dummy by entity  
dummy\_entidad = pd.get\_dummies(file\_df['poliza\_entidad\_legal'], prefix='d\_entidad')  
file\_df = pd.concat([file\_df, dummy\_entidad], axis=1)  
**del** file\_df['poliza\_entidad\_legal']

6- Also we distinguish among IDs types:

# DOC. TYPE: We create a dummy by doc. type.  
dummy\_tipodoc = pd.get\_dummies(file\_df['cliente\_tipo\_documento'], prefix='d\_tipodoc')  
file\_df = pd.concat([file\_df, dummy\_tipodoc], axis=1)  
**del** dummy\_tipodoc

7- Finally we apply Fuzzy Rules to detect that IDs that are not well specified:

# BAD ID: We use fuzzy rules to detect if an ID is bad specified  
file\_df['bad\_id'] = pd.Series(0, index=file\_df.index)  
**for** index, row **in** file\_df.iterrows():  
 tipo\_doc = row['cliente\_tipo\_documento']  
 nif = row['id\_fiscal']  
 **try**:  
 sys.stdout = open(os.devnull, 'w')  
 value = id\_conversor(tipo\_doc, nif)  
 sys.stdout = sys.\_\_stdout\_\_  
 **except**:  
 value = 1  
 file\_df.loc[index, 'bad\_id'] = value

8- As a last step, we delete variables that are useless:

# DELETE VARIABLES  
delete\_var = ['cliente\_tipo\_documento', 'ID\_DOSSIER']  
**for** i **in** delete\_var:  
 **del** file\_df[i]

Therefore, the final bottle is configurated as follows:

|  |  |
| --- | --- |
| **Name** | **Description** |
| id\_siniestro | Sinister Id |
| id\_fiscal | Fiscal code |
| id\_poliza | Policy code |
| cliente\_codfiliacion | Filiation code |
| dummy\_fisica | Type of person |
| d\_producto\_00530 | Property product 530 |
| d\_producto\_00535 | Property product 535 |
| d\_producto\_00536 | Property product 536 |
| d\_producto\_00537 | Property product 537 |
| d\_producto\_00555 | Property product 555 |
| d\_producto\_00592 | Property product 592 |
| d\_producto\_01500 | Property product 1500 |
| d\_producto\_72530 | Property product 72530 |
| d\_producto\_72531 | Property product 72531 |
| d\_producto\_72532 | Property product 72532 |
| d\_entidad\_B1 | Banc Sabadell entity |
| d\_entidad\_Z | Zurich entity |
| d\_tipodoc\_C | Tipo de Documento C |
| d\_tipodoc\_N | Tipo de Documento N |
| d\_tipodoc\_P | Tipo de Documento P |
| d\_tipodoc\_R | Tipo de Documento R |
| d\_tipodoc\_S | Tipo de Documento S |
| d\_tipodoc\_X | Tipo de Documento X |
| bad\_id | Bad specified ID |

## CUSTOMER

This is the table of policyholder’s attributes variables that are embodied in insurance policies. It contains names, sex, age, addresses, etc.

They are relevant in terms of fraud because this is the base information of the policy holder. Also it contains the main contact data which is useful because we can cross it with other contact information sources. Intuitively, it is important to notice the importance of this type of information. Mainly, the contact information. This will permit us to cross with other contact information from another sources. In Fraud, it is important to reveal relationships between individuals, principally for treating “hard fraud” as organized crime.

We start with the following variables:

|  |  |
| --- | --- |
| **Name** | **Description** |
| id\_fiscal | Id. Fiscal |
| id\_poliza | Número de póliza |
| version\_poliza | Número de versión de la póliza |
| cliente\_sexo | Sexo del tomador de la póliza seleccionada |
| cliente\_edad | Edad del tomador de la póliza |
| cliente\_antiguedad | Número años de antigüedad en la compañía del tomador de la póliza seleccionada |
| cliente\_cp | Código postal |
| cliente\_morosidad | Presencia en tablas de morosidad |
| cliente\_tipo\_doc | Tipo doc. |
| cliente\_apellido1 | Apellido 1 |
| cliente\_apellido2 | Apellido 2 |
| cliente\_nombre | Nombre |
| cliente\_fecha\_nacimiento | F. Nacimiento |
| cliente\_forma\_contacto | Forma de contacto |
| cliente\_nacionalidad | Nacionalidad persona |
| cliente\_pais\_residencia | País de residencia |
| cliente\_tipo\_via | Tipo de vía |
| cliente\_nombre\_via | Nombre de la vía |
| cliente\_numero\_hogar | Nº |
| cliente\_puerta | Piso/Puerta/Escalera |
| cliente\_poblacion | Población |
| cliente\_provincia | Provincia |
| cliente\_telefono\_tipo | Tipo com. |
| cliente\_telefono\_pais | País |
| cliente\_telefono\_numero | Número |
| cliente\_domicilio\_principal | Domicilio principal |
| cliente\_domicilio\_bancario\_IBAN | Cuenta IBAN |
| cliente\_domicilio\_bancario\_titular | Titulares |
| cliente\_numero\_siniestros\_anterior | Númeo de Siniestros Anteriores de Cliente |
| cliente\_numero\_siniestros\_anterior\_hogar | Númeo de Siniestros Anteriores de Cliente en el ramo HOGAR |
| cliente\_email | Dirección de email |
| auditCodigoSiniestroReferencia | Codigo del siniestro de referencia |
| auditFechaAperturaSiniestroReferencia | Fecha de apertura del siniestro de referencia |

Using cliente\_info.py module we process this variables.

1- As always we start loading the file, transforming it to a Dataframe and mapping the key variables:

# Load File  
file = ReadCsv.load\_csv(STRING.cliente\_info)  
  
# Transform file to DF  
file\_df = DfUtils.processing\_file(file)  
file\_df = file\_df.sort\_values(by=['auditCodigoSiniestroReferencia'],  
 ascending=[**True**])  
file\_df = file\_df.rename(columns={'auditCodigoSiniestroReferencia': 'id\_siniestro'})  
  
# Define Principal ID dtypes  
file\_df['id\_siniestro'] = file\_df['id\_siniestro'].map(int)  
file\_df['id\_fiscal'] = file\_df['id\_fiscal'].map(str)

2- Now we search if the ‘Id\_Fiscal’ variable matches with the Id bottle. For that we need to do a merge function and create a Boolean variable:

# ID COINCIDE: We compare with ID's from ID processed if the ID has not coincidence  
file\_id = ReadCsv.load\_csv(STRING.processed\_id)  
df\_id = DfUtils.processing\_file(file\_id, delimiter=';')  
df\_id = df\_id[['id\_siniestro', 'id\_fiscal']]  
df\_id['id\_siniestro'] = df\_id['id\_siniestro'].map(int)  
file\_df = pd.merge(file\_df, df\_id, how='left', on='id\_siniestro', suffixes=('', '\_id'))  
df\_id['id\_fiscal'] = df\_id['id\_fiscal'].map(str)  
file\_df['cliente\_id\_fiscal\_no\_coincide'] = pd.Series(1, index=file\_df.index)  
file\_df.loc[file\_df['id\_fiscal'] == file\_df['id\_fiscal\_id'], 'cliente\_id\_fiscal\_no\_coincide'] = 0  
**del** df\_id  
**del** file\_df['id\_fiscal\_id']

3- We try to find the ‘Id\_Fiscal’ in the Blacklist, so we need to bring temporarily the Blacklist that holds the fraudulent insured (Tomador Blacklist).

# ID BLACKLIST: We compare with Tomador Blacklist if the customer appears in a Fraud Sinister  
file\_blacklist\_tomador = ReadCsv.load\_csv(STRING.processed\_blacklist\_tomador)  
df\_bl\_tomador = DfUtils.processing\_file(file\_blacklist\_tomador, delimiter=';')  
df\_bl\_tomador = df\_bl\_tomador.drop\_duplicates(subset=['NIF'], keep='last')  
df\_bl\_tomador['NIF'] = df\_bl\_tomador['NIF'].map(str)  
file\_df = pd.merge(file\_df, df\_bl\_tomador, how='left', left\_on='id\_fiscal', right\_on='NIF')  
file\_df['cliente\_id\_fiscal\_blacklist'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['NIF'].notnull(), 'cliente\_id\_fiscal\_blacklist'] = 1  
**del** df\_bl\_tomador  
**del** file\_df['NIF']

4- Customer Age: We have created several dummy ranges using the traditional market view segmentation. It consists in five ranges: 18-29, 30-39, 40-49, 50-59, >60. Also we add a new segment that catch NULL values or inconsistent values (>100 or <18).

# CLIENTE EDAD: We create 4 age groups and an inconsistency group  
file\_df['cliente\_edad'] = file\_df['cliente\_edad'].fillna(0)  
  
file\_df['cliente\_e18\_29'] = pd.Series(0, index=file\_df.index)  
file\_df['cliente\_e30\_39'] = pd.Series(0, index=file\_df.index)  
file\_df['cliente\_e40\_49'] = pd.Series(0, index=file\_df.index)  
file\_df['cliente\_e50\_59'] = pd.Series(0, index=file\_df.index)  
file\_df['cliente\_e60'] = pd.Series(0, index=file\_df.index)  
file\_df['cliente\_edad'] = pd.to\_numeric(file\_df['cliente\_edad'], errors='coerce')  
file\_df['cliente\_edad\_incosistente'] = pd.Series(0, index=file\_df.index)  
  
file\_df.loc[file\_df['cliente\_edad'].between(18, 29, inclusive=**True**), 'cliente\_e18\_29'] = 1  
file\_df.loc[file\_df['cliente\_edad'].between(30, 39, inclusive=**True**), 'cliente\_e30\_39'] = 1  
file\_df.loc[file\_df['cliente\_edad'].between(40, 49, inclusive=**True**), 'cliente\_e40\_49'] = 1  
file\_df.loc[file\_df['cliente\_edad'].between(50, 59, inclusive=**True**), 'cliente\_e50\_59'] = 1  
file\_df.loc[60 <= file\_df['cliente\_edad'], 'cliente\_e60'] = 1  
file\_df.loc[(18 > file\_df['cliente\_edad']) | (file\_df['cliente\_edad'] > 100), 'cliente\_edad\_incosistente'] = 1

5- We correct the customer seniority bad calculation values. We have identify just one error persistent type. They are the cases with seniority greater than 100. Surely, they are associated with the value 01/01/1900. Implicit here there is a NULL value. Therefore we replace values > 100 with NaN values.

# CLIENTE ANTIGUEDAD: We replace bad values with NAN. We consider > 100 because bad values are  
# associated with the bad inputation 01/01/1900  
file\_df['cliente\_antiguedad'] = pd.to\_numeric(file\_df['cliente\_antiguedad'], errors='coerce')  
file\_df.loc[file\_df['cliente\_antiguedad'] > 100, 'cliente\_antiguedad'] = np.nan

6- Categorical Variables: We create dummies for each of them because they do not have an order sense.

# CATEGORICAL VARIABLES: We create dummy variables to the next categorical values  
cat\_variables = ['cliente\_forma\_contacto', 'cliente\_telefono\_tipo']  
**for** i **in** cat\_variables:  
 name = str(i)  
 file\_df[name] = file\_df[name].replace('',np.nan)  
 dummy = pd.get\_dummies(file\_df[i], dummy\_na=**True**, prefix=name)  
 file\_df = pd.concat([file\_df, dummy], axis=1)  
 **del** dummy  
 **del** file\_df[name]

7-IBAN informed: If the IBAN is not null we assign one as value. Otherwise zero.

# IBAN INFORMADO: We check if the IBAN is informed  
file\_df['iban\_informado'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['cliente\_domicilio\_bancario\_IBAN'].notnull(), 'iban\_informado'] = 1

8- Using the IBAN blacklist, we search if the IBAN registered is associated with a past fraud sinister. For that, we have to bring temporarily the IBAN blacklist table.

# IBAN FRAUDE: We check if the IBAN is associated with a previous Fraud Sinister  
file\_blacklist\_IBAN = ReadCsv.load\_csv(STRING.processed\_IBAN)  
df\_bl\_IBAN = DfUtils.processing\_file(file\_blacklist\_IBAN, delimiter=';')  
df\_bl\_IBAN = df\_bl\_IBAN.drop\_duplicates(subset=['IBAN'], keep='last')  
file\_df['cliente\_domicilio\_bancario\_IBAN'] = file\_df['cliente\_domicilio\_bancario\_IBAN'].map(str)  
df\_bl\_IBAN['IBAN'] = df\_bl\_IBAN['IBAN'].map(str)  
file\_df = pd.merge(file\_df, df\_bl\_IBAN, how='left', left\_on='cliente\_domicilio\_bancario\_IBAN', right\_on='IBAN')  
file\_df['cliente\_iban\_blacklist'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['IBAN'].notnull(), 'cliente\_iban\_blacklist'] = 1  
**del** df\_bl\_IBAN  
**del** file\_df['IBAN']

9- We construct to Nationality variable types. One dummy for Spanish customers and one dummy for the country region.

We have to construct a table with every possible nationality and the belonging grouping region. Then, we makee a merge between the bottle and this table.

# CLIENTE NACIONALIDAD: Using a List of country regions we group the nationalities. Also we create a variable if  
# the customer is Spanish or not  
file\_df['cliente\_d\_español'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['cliente\_nacionalidad'] == 'ESPAÃ‘A', 'cliente\_d\_español'] = 1  
country\_file = pd.read\_csv('docs\_input\\country\_list.csv', sep=';', encoding='latin1')  
file\_df = pd.merge(file\_df, country\_file, how='left', left\_on='cliente\_nacionalidad',  
 right\_on='COUNTRY')  
dummy\_region = pd.get\_dummies(file\_df['REGION'], prefix='cliente\_region', dummy\_na=**True**)  
file\_df = pd.concat([file\_df, dummy\_region], axis=1)  
**del** file\_df['REGION']  
**del** file\_df['COUNTRY']  
**del** dummy\_region

10- We do the same but for the residence variable:

# PAIS DE RESIDENCIA: We do the same as Nationality but with the Residence Country  
file\_df['cliente\_d\_residencia\_espania'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['cliente\_pais\_residencia'] == 'ESPAÃ‘A', 'cliente\_d\_residencia\_espania'] = 1  
country\_file = pd.read\_csv('docs\_input\\country\_list.csv', sep=';', encoding='latin1')  
file\_df = pd.merge(file\_df, country\_file, how='left', left\_on='cliente\_pais\_residencia',  
 right\_on='COUNTRY')  
dummy\_region = pd.get\_dummies(file\_df['REGION'], prefix='cliente\_residencia\_region', dummy\_na=**True**)  
file\_df = pd.concat([file\_df, dummy\_region], axis=1)  
**del** dummy\_region

11- We have two types of Float Variables. Total number of sinister and property number of sinister. First, we ensure that the values are numeric (coercing to error every kind of mistakes). Second, we get the proportion between property sinister and total sinister. Finally, we append to an outlier list that contains the column names of the variables we want to search for outliers.

# FLOAT VARIABLES:  
# 1-First, we generate the proportion between Property Sinisters and Total Sinisters that the customer had.  
file\_df['cliente\_numero\_siniestros\_anterior\_hogar'] = pd.to\_numeric(  
 file\_df['cliente\_numero\_siniestros\_anterior\_hogar'],  
 errors='coerce')  
  
file\_df['cliente\_numero\_siniestros\_anterior'] = pd.to\_numeric(file\_df['cliente\_numero\_siniestros\_anterior'],  
 errors='coerce')  
  
file\_df['cliente\_siniestro\_hogar\_porc'] = pd.Series(  
 file\_df['cliente\_numero\_siniestros\_anterior\_hogar'] / file\_df['cliente\_numero\_siniestros\_anterior'],  
 index=file\_df.index)  
  
# 2-Second, we save each float variable to apply Outliers MAD algorithm.  
float\_variables = ['cliente\_numero\_siniestros\_anterior', 'cliente\_numero\_siniestros\_anterior\_hogar',  
 'cliente\_siniestro\_hogar\_porc']  
  
**for** i **in** float\_variables:  
 outliers.append(i)

12- We check if emails and phone numbers are the same used in the contact with the Assistance Center. For that, we bring temporarily the EUROPA table.

# EMAIL Y TELEFONO COINCIDE: We merge with EUROPA database and check if phone and email has any coincidence.  
file\_europa = ReadCsv.load\_csv(STRING.processed\_europa)  
df\_europa = DfUtils.processing\_file(file\_europa, delimiter=';')  
df\_europa = df\_europa[['id\_fiscal', 'ASIST\_EUROPA\_LLAMANTE\_MAIL', 'ASIST\_EUROPA\_LLAMANT\_TELEFONO1'  
 ]]  
  
df\_europa['id\_fiscal'] = df\_europa['id\_fiscal'].map(**lambda** x: x.strip())  
df\_europa['id\_fiscal'] = df\_europa['id\_fiscal'].map(str)  
df\_europa = df\_europa.dropna(how='any', subset=['id\_fiscal'])  
df\_europa = df\_europa.dropna(how='any', subset=['ASIST\_EUROPA\_LLAMANTE\_MAIL',  
 'ASIST\_EUROPA\_LLAMANT\_TELEFONO1'])  
  
df\_europa = df\_europa.drop\_duplicates(subset=['id\_fiscal'], keep='last')  
  
file\_df = pd.merge(left=file\_df, right=df\_europa, how='left',  
 on='id\_fiscal')  
  
file\_df['cliente\_email'] = file\_df['cliente\_email'].map(str)  
df\_europa['ASIST\_EUROPA\_LLAMANTE\_MAIL'] = df\_europa['ASIST\_EUROPA\_LLAMANTE\_MAIL'].map(str)  
file\_df['cliente\_email'] = file\_df['cliente\_email'].fillna(1) #Esto es para que no coincida si ambos son nulos  
file\_df['cliente\_europa\_email\_no\_coincide'] = pd.Series(1, index=file\_df.index)  
file\_df.loc[file\_df['cliente\_email'] == file\_df['ASIST\_EUROPA\_LLAMANTE\_MAIL'], 'cliente\_europa\_email\_no\_coincide'] = 0  
  
file\_df['cliente\_telefono\_numero'] = file\_df['cliente\_telefono\_numero'].map(str)  
df\_europa['ASIST\_EUROPA\_LLAMANT\_TELEFONO1'] = df\_europa['ASIST\_EUROPA\_LLAMANT\_TELEFONO1'].map(str)  
file\_df['cliente\_telefono\_numero'] = file\_df['cliente\_telefono\_numero'].fillna(1) # Esto es para que no coincida si ambos son nulos  
file\_df['cliente\_europa\_telefono\_no\_coincide'] = pd.Series(1, index=file\_df.index)  
file\_df.loc[file\_df['cliente\_telefono\_numero'] == file\_df['ASIST\_EUROPA\_LLAMANT\_TELEFONO1'], 'cliente\_europa\_telefono\_no\_coincide'] = 0  
  
**del** df\_europa

13- We apply MAD outlier formula to the outlier list generated:

# OUTLIERS: We generate new variables that indicates if the value is an outlier or not, using MAD algorithm.  
**for** i **in** outliers:  
 Outliers.outliers\_mad(file\_df, i, not\_count\_zero=**False**)

14- Finally we delete the useless variables:

delete\_var = ['id\_fiscal', 'id\_poliza', 'version\_poliza', 'cliente\_sexo',  
 'cliente\_fecha\_nacimiento',  
 'cliente\_edad',  
 'cliente\_morosidad', 'cliente\_tipo\_doc', 'cliente\_apellido1', 'cliente\_apellido2',  
 'cliente\_nombre',  
 'cliente\_pais\_residencia', 'cliente\_tipo\_via', 'cliente\_nombre\_via',  
 'cliente\_numero\_hogar', 'cliente\_puerta', 'cliente\_poblacion',  
 'cliente\_provincia', 'cliente\_telefono\_pais', 'cliente\_domicilio\_principal',  
 'cliente\_domicilio\_bancario\_titular', 'cliente\_domicilio\_bancario\_IBAN',  
 'cliente\_email', 'ASIST\_EUROPA\_LLAMANTE\_MAIL', 'ASIST\_EUROPA\_LLAMANT\_TELEFONO1',  
 'cliente\_nacionalidad',  
 'auditFechaAperturaSiniestroReferencia', 'cliente\_telefono\_numero',  
 'COUNTRY', 'REGION']  
  
**for** i **in** delete\_var:  
 **del** file\_df[i]

|  |  |
| --- | --- |
| **Name** | **Description** |
| cliente\_antiguedad | How many days as a costumer |
| cliente\_cp | Costumer postal code |
| cliente\_numero\_siniestros\_anterior | Previous sinister total quantity |
| cliente\_numero\_siniestros\_anterior\_hogar | Previous sinister property quantity |
| id\_siniestro | Sinister Id |
| cliente\_id\_fiscal\_no\_coincide | Customer ID does not match with Bottle ID |
| cliente\_id\_fiscal\_blacklist | Customer is in Blacklist |
| cliente\_e18\_29 | Age 18-29 |
| cliente\_e30\_39 | Age 30-39 |
| cliente\_e40\_49 | Age 40-49 |
| cliente\_e50\_59 | Age 50-59 |
| cliente\_e60 | Age > 60 |
| cliente\_edad\_incosistente | Nan Value or <18 or > 100 |
| cliente\_forma\_contacto\_A | Contact way A |
| cliente\_forma\_contacto\_C | Contact way C |
| cliente\_forma\_contacto\_D | Contact way D |
| cliente\_forma\_contacto\_L | Contact way L |
| cliente\_forma\_contacto\_M | Contact way M |
| cliente\_forma\_contacto\_S | Contact way S |
| cliente\_forma\_contacto\_W | Contact way W |
| cliente\_forma\_contacto\_nan | Contact Way NaN |
| cliente\_telefono\_tipo\_FI | Phone type FI |
| cliente\_telefono\_tipo\_MO | Phone type MO |
| cliente\_telefono\_tipo\_TL | Phone type TL |
| cliente\_telefono\_tipo\_nan | Phone Type NaN |
| iban\_informado | = 1 If the IBAN is not null |
| cliente\_iban\_blacklist | IBAN is present in the Blacklist |
| cliente\_d\_español | Spanish customer |
| cliente\_region\_AFRICA ARABE | AFRICA ARABIC customer |
| cliente\_region\_AFRICA SUBSHARIANA | AFRIC SUBSHARIAN customer |
| cliente\_region\_AMERICA CENTRAL | CENTRAL AMERICA customer |
| cliente\_region\_AMERICA DEL NORTE | NORTH AMERICA customer |
| cliente\_region\_AMERICA DEL SUR | SOUTH AMIERCA customer |
| cliente\_region\_ASIA CENTRAL | ASIA CENTRAL customer |
| cliente\_region\_ASIA OCCIDENTAL | ASIA OCCIDENTAL customer |
| cliente\_region\_ASIA ORIENTAL | ASIA ORIENTAL customer |
| cliente\_region\_EUROPA CENTRAL | EUROPA CENTRAL customer |
| cliente\_region\_EUROPA DEL NORTE | NORTH EUROPE customer |
| cliente\_region\_EUROPA DEL SUR | SOUTH EUROPE customer |
| cliente\_region\_EUROPA OCCIDENTAL | OCCIDENTAL EUROPE customer |
| cliente\_region\_EUROPA ORIENTAL | ORIENTAL EUROPE customer |
| cliente\_region\_MAR CARIBE | MAR CARIBE customer |
| cliente\_region\_OCEANIA | OCEANIA customer |
| cliente\_region\_nan | NAN region customer |
| cliente\_d\_residencia\_espania | Spain residence |
| cliente\_residencia\_region\_AFRICA ARABE | AFRICA ARABIC residence |
| cliente\_residencia\_region\_AFRICA SUBSHARIANA | AFRICA SUBSHARIANA residence |
| cliente\_residencia\_region\_AMERICA CENTRAL | AMERICA CENTRAL residence |
| cliente\_residencia\_region\_AMERICA DEL NORTE | AMERICA DEL NORTE residence |
| cliente\_residencia\_region\_AMERICA DEL SUR | AMERICA DEL SUR residence |
| cliente\_residencia\_region\_ASIA OCCIDENTAL | ASIA OCCIDENTAL residence |
| cliente\_residencia\_region\_ASIA ORIENTAL | ASIA ORIENTAL residence |
| cliente\_residencia\_region\_EUROPA CENTRAL | EUROPA CENTRAL residence |
| cliente\_residencia\_region\_EUROPA DEL NORTE | NORTH EUROPE residence |
| cliente\_residencia\_region\_EUROPA DEL SUR | SOUTH EUROPE residence |
| cliente\_residencia\_region\_EUROPA OCCIDENTAL | OCCIDENTAL EUROPE residence |
| cliente\_residencia\_region\_EUROPA ORIENTAL | ORIENTAL EUROPE residence |
| cliente\_residencia\_region\_MAR CARIBE | MAR CARIBE residence |
| cliente\_residencia\_region\_OCEANIA | OCEANIA residence |
| cliente\_residencia\_region\_nan | NAN region residence |
| cliente\_siniestro\_hogar\_porc | % property sinister / total sinister |
| cliente\_europa\_email\_no\_coincide | Email does not match in EUROPA |
| cliente\_europa\_telefono\_no\_coincide | Phone does not match in EUROPA |
| cliente\_numero\_siniestros\_anterior\_mad\_outlier | Previous sinister total quantity Outliers |
| cliente\_numero\_siniestros\_anterior\_hogar\_mad\_outlier | Previous sinister property quantity Outliers |
| cliente\_siniestro\_hogar\_porc\_mad\_outlier | % property sinister / total sinister Outliers |

## CUSTOMER\_PROPERTY

Crossing data from policyholders and the specific property of the policy analyzed, we construct several variables about this relationship. What we are doing here is not only has separated dimensions about property and client but a third dimension that interrelates both.

|  |  |
| --- | --- |
| **Name** | **Description** |
| id\_fiscal | Id. Fiscal |
| id\_poliza | Número de póliza |
| version\_poliza | Número de versión de la póliza |
| cliente\_hogar\_carga\_siniestral | Coste total de los siniestros tanto abiertos como cerrados de tipo hogar por id\_fiscal Pólizas en vigor y/o anuladas  Siniestros tanto abiertos como cerrados |
| cliente\_hogar\_numero\_siniestros\_anterior | Númeo de Siniestros Anteriores de Cliente asociado al Hogar específico |
| cliente\_hogar\_mediador\_tomador | NIF Mediador y Tomador o Pagador coinciden |
| auditCodigoSiniestroReferencia | Codigo del siniestro de referencia |
| auditFechaAperturaSiniestroReferencia | Fecha de apertura del siniestro de referencia |

1- As always we start loading the file, transforming it to a Dataframe and mapping the key variables:

# Load File  
file = ReadCsv.load\_csv(name\_file)  
  
# Transform file to DF  
file\_df = DfUtils.processing\_file(file)  
file\_df = file\_df.sort\_values(by=['auditCodigoSiniestroReferencia'],  
 ascending=[**True**])  
file\_df = file\_df.rename(columns={'auditCodigoSiniestroReferencia':  
 'id\_siniestro'})

2- We calculate outliers for several variables:

# OUTLIERS: Get the outliers using MAD from the next variables  
outliers = ['cliente\_hogar\_numero\_siniestros\_anterior',  
 'cliente\_hogar\_carga\_siniestral']  
  
**if** test == **False**:  
 **for** i **in** outliers:  
 Outliers.outliers\_mad(file\_df, i, not\_count\_zero=**False**)  
**else**:  
 base\_df = ReadCsv.load\_csv(STRING.cliente\_hogar)  
 base\_df = DfUtils.processing\_file(base\_df)  
 base\_df = base\_df[outliers]  
 **for** i **in** outliers:  
 Outliers.outliers\_test\_values(file\_df, base\_df, i, not\_count\_zero=**False**)  
 **del** base\_df

3- We delete useless variables:

# DELETE VARIABLES  
delete\_var = ['id\_fiscal', 'id\_poliza', 'version\_poliza',  
 'cliente\_hogar\_mediador\_tomador', 'auditFechaAperturaSiniestroReferencia'  
 ]  
  
**for** i **in** delete\_var:  
 **del** file\_df[i]

The output is:

|  |  |
| --- | --- |
| **Name** | **Description** |
| cliente\_hogar\_carga\_siniestral | Sum of total sinister cost by HOGAR |
| cliente\_hogar\_numero\_siniestros\_anterior | HOGAR number of sinister |
| id\_siniestro | Sinister id |
| cliente\_hogar\_numero\_siniestros\_anterior\_mad\_outlier | Outlier number of sinister |
| cliente\_hogar\_carga\_siniestral\_mad\_outlier | Outlier sinister cost |

## DATES

This table contains one of the most important spectrum of whole analysis. It includes variables related to dates. There are up to thirty relevant dates that can give us an insight of fraudulent behaviors.

In this section we have date variables linked to policies movements, sinister and sinister reports.

|  |  |
| --- | --- |
| **Name** | **Description** |
| id\_poliza | Número de póliza |
| version\_poliza | Número de versión de la póliza |
| id\_siniestro | Código de Siniestro |
| id\_dossier | Código de Dossier |
| fecha\_poliza\_emision | Fecha de emisión de la póliza. La fecha de emisión es la fecha de inicio de póliza |
| fecha\_poliza\_efecto\_natural | Efecto Natural |
| fecha\_poliza\_efecto\_mvto | Efecto Movto. |
| fecha\_poliza\_vto\_movimiento | F. vto. Mvto |
| fecha\_poliza\_vto\_natural | F. vto. Natural |
| fecha\_poliza\_cambio\_estado | Fecha en la que se registra el último cambio de estado de la póliza |
| fecha\_pago\_recibo\_vencimiento | Fecha vencimiento pago recibo |
| fecha\_pago\_recibo | Fecha Pago de Recibo |
| fecha\_siniestro\_ocurrencia | F. Ocurrencia |
| fecha\_siniestro\_reclamación | F. Ocurrencia |
| fecha\_siniestro\_comunicación | F. Ocurrencia |
| fecha\_siniestro\_manifestacion | F. Ocurrencia |
| fecha\_siniestro\_situacion | F. Situación |
| fecha\_europa\_apertura | Fecha de Apertura de Dossier |
| fecha\_europa\_declaracion | Fecha declaración de Siniestro |
| fecha\_europa\_denuncia | Fecha de Denuncia de Siniestro |
| fecha\_europa\_ocurrencia | Fecha de Ocurrencia de Siniestro |
| fecha\_encargo\_peritaje |  |
| fecha\_primera\_visita\_peritaje | IT: fecha primer peritaje |
| fecha\_ultima\_visita\_peritaje | IT: fecha último peritaje |
| fecha\_informe\_peritaje |  |

Using fecha.py module we process this variables.

1- As always we start loading the file, transforming it to a Dataframe and mapping the key variables:

# Load File  
file = ReadCsv.load\_csv(name\_file)  
  
# Transform file to DF  
file\_df = DfUtils.processing\_file(file)

2- We get the year, month, day, weekday for each of the next variables and we put them in dummy variables. Also, this process is very important because normalize date variables to datetime type.

# MESES-AÑOS-DIAS: We transform every date variable to dummies, YEAR, MONTH, DAY, WEEKDAY  
var\_fecha = ["fecha\_poliza\_emision", "fecha\_poliza\_efecto\_natural",  
 "fecha\_poliza\_efecto\_mvto", "fecha\_poliza\_vto\_movimiento",  
 "fecha\_poliza\_vto\_natural",  
 "fecha\_siniestro\_ocurrencia",  
 'fecha\_primera\_visita\_peritaje',  
 'fecha\_ultima\_visita\_peritaje', 'fecha\_siniestro\_comunicación'  
 ]  
  
**for** i **in** var\_fecha:  
 year = str(i) + '\_year'  
 month = str(i) + '\_month'  
 day = str(i) + '\_day'  
 weekday = str(i) + '\_weekday'  
  
 file\_df[i] = pd.to\_datetime(file\_df[i], format='%Y-%m-%d', errors='coerce')  
  
 file\_df[year] = pd.DatetimeIndex(file\_df[i]).year  
 file\_df[month] = pd.DatetimeIndex(file\_df[i]).month  
 file\_df[day] = pd.DatetimeIndex(file\_df[i]).day  
 file\_df[weekday] = pd.Series(file\_df[i].dt.weekday, index=file\_df.index)  
  
file\_df = file\_df[file\_df['fecha\_siniestro\_ocurrencia\_year'] >= 2014]  
  
file\_df = file\_df.sort\_values(['id\_siniestro', 'fecha\_primera\_visita\_peritaje'], ascending=[**True**, **True**])

3- We define two types of dates: Complex and Non-Complex. Complex are referred to that date variables we want to get more information as it is possible. Less complex are more grouped variables than Complex variables.

Complex variables:

Complex date variables are the variables most important in terms of fraud detection. For example fecha\_siniestro\_ocurrencia. We generate dummies for year, month, day and weekday. Also we get dummies between day ranges (1-10, 10-20, 20-30).

**for** i **in** var\_fecha\_complex:  
 year = str(i) + '\_year'  
 month = str(i) + '\_month'  
 day = str(i) + '\_day'  
 weekday = str(i) + '\_weekday'  
  
 dummy\_year = pd.get\_dummies(file\_df[year], prefix='d\_' + year)  
 file\_df = pd.concat([file\_df, dummy\_year], axis=1)  
 **del** dummy\_year  
  
 dummy\_month = pd.get\_dummies(file\_df[month], prefix='d' + month)  
 file\_df = pd.concat([file\_df, dummy\_month], axis=1)  
 **del** dummy\_month  
  
 day\_1\_10 = day + '\_day\_1\_10'  
 day\_10\_20 = day + '\_day\_10\_20'  
 day\_20\_30 = day + '\_day\_20\_30'  
 file\_df[day\_1\_10] = pd.Series(0, index=file\_df.index)  
 file\_df[day\_10\_20] = pd.Series(0, index=file\_df.index)  
 file\_df[day\_20\_30] = pd.Series(0, index=file\_df.index)  
 file\_df.loc[file\_df[day].between(1, 10, **True**), day\_1\_10] = 1  
 file\_df.loc[file\_df[day].between(11, 20, **True**), day\_10\_20] = 1  
 file\_df.loc[file\_df[day].between(21, 31, **True**), day\_20\_30] = 1  
  
 dummy\_weekday = pd.get\_dummies(file\_df[weekday], prefix='d\_' + weekday)  
 file\_df = pd.concat([file\_df, dummy\_weekday], axis=1)  
 **del** dummy\_weekday

Less Complex:

**for** i **in** var\_fecha\_less\_complex:  
 year = str(i) + '\_year'  
 month = str(i) + '\_month'  
 day = str(i) + '\_day'  
 weekday = str(i) + '\_weekday'  
  
 dummy\_year = pd.get\_dummies(file\_df[year], prefix='d\_' + year)  
 file\_df = pd.concat([file\_df, dummy\_year], axis=1)  
 **del** dummy\_year  
  
 month\_holiday = month + '\_holiday'  
 file\_df[month\_holiday] = pd.Series(0, index=file\_df.index)  
 file\_df.loc[file\_df[month].isin([1, 8, 12]), month\_holiday] = 1  
  
 day\_1\_10 = day + '\_day\_1\_10'  
 day\_10\_20 = day + '\_day\_10\_20'  
 day\_20\_30 = day + '\_day\_20\_30'  
 file\_df[day\_1\_10] = pd.Series(0, index=file\_df.index)  
 file\_df[day\_10\_20] = pd.Series(0, index=file\_df.index)  
 file\_df[day\_20\_30] = pd.Series(0, index=file\_df.index)  
 file\_df.loc[file\_df[day].between(1, 10, **True**), day\_1\_10] = 1  
 file\_df.loc[file\_df[day].between(11, 20, **True**), day\_10\_20] = 1  
 file\_df.loc[file\_df[day].between(21, 31, **True**), day\_20\_30] = 1  
  
 weekday\_weekend = weekday + '\_weekend'  
 file\_df[weekday\_weekend] = pd.Series(0, index=file\_df.index)  
 file\_df.loc[file\_df[weekday].isin([6, 7]), weekday\_weekend] = 1  
  
 weekday\_monday = weekday + '\_monday'  
 file\_df[weekday\_monday] = pd.Series(0, index=file\_df.index)  
 file\_df.loc[file\_df[weekday] == 0, weekday\_monday] = 1

Also we get year, month, day, weekday and range of days. However, weekday will be simplified to weekend and Monday. Month will be simplified to Holiday months. And also we will delete year and day information because they will appear as time duration variables.

4- Logical Dates: This section covers different alternatives of relevant date’s transformations. Basically they can be summarized as follows:

a) Diff between first policy emission and expiration date:

file\_df['fecha\_diferencia\_vto\_emision'] = pd.Series(file\_df['fecha\_poliza\_vto\_natural'] -  
 file\_df['fecha\_poliza\_emision'], index=file\_df.index).dt.days

b) If Date Effect < Date Emission, then d = 1

file\_df['fecha\_indicador\_efecto\_emision'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['fecha\_poliza\_emision'] > file\_df[  
 'fecha\_poliza\_efecto\_natural'], 'fecha\_indicador\_efecto\_emision'] = 1

c) Diff between policy effect and the sinister (Also add 5, 15, 30 dummies).

file\_df['fecha\_diferencia\_siniestro\_efecto'] = pd.Series(  
 file\_df['fecha\_siniestro\_ocurrencia'] - file\_df['fecha\_poliza\_efecto\_natural'], index=file\_df.index).dt.days  
file\_df['fecha\_diferencia\_siniestro\_efecto\_5dias'] = pd.Series(0, index=file\_df.index)  
file\_df['fecha\_diferencia\_siniestro\_efecto\_15dias'] = pd.Series(0, index=file\_df.index)  
file\_df['fecha\_diferencia\_siniestro\_efecto\_30dias'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['fecha\_diferencia\_siniestro\_efecto'] <= 5, 'fecha\_diferencia\_siniestro\_efecto\_5dias'] = 1  
file\_df.loc[file\_df['fecha\_diferencia\_siniestro\_efecto'] <= 15, 'fecha\_diferencia\_siniestro\_efecto\_15dias'] = 1  
file\_df.loc[file\_df['fecha\_diferencia\_siniestro\_efecto'] <= 30, 'fecha\_diferencia\_siniestro\_efecto\_30dias'] = 1

d) Diff between policy emission and the sinister (Also add 5, 15, 30 dummies).

file\_df['fecha\_diferencia\_siniestro\_emision'] = pd.Series(  
 file\_df['fecha\_siniestro\_ocurrencia'] - file\_df['fecha\_poliza\_emision'], index=file\_df.index).dt.days  
file\_df['fecha\_diferencia\_siniestro\_emision\_5dias'] = pd.Series(0, index=file\_df.index)  
file\_df['fecha\_diferencia\_siniestro\_emision\_15dias'] = pd.Series(0, index=file\_df.index)  
file\_df['fecha\_diferencia\_siniestro\_emision\_30dias'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['fecha\_diferencia\_siniestro\_emision'] <= 5, 'fecha\_diferencia\_siniestro\_emision\_5dias'] = 1  
file\_df.loc[file\_df['fecha\_diferencia\_siniestro\_emision'] <= 15, 'fecha\_diferencia\_siniestro\_emision\_15dias'] = 1  
file\_df.loc[file\_df['fecha\_diferencia\_siniestro\_emision'] <= 30, 'fecha\_diferencia\_siniestro\_emision\_30dias'] = 1

e) Diff between policy expiration and the sinister (Also add 5, 15, 30 dummies).

# diferencia entre siniestro y vencimiento 5, 15, 30 días  
file\_df['fecha\_diferencia\_siniestro\_vto\_natural'] = pd.Series(  
 file\_df['fecha\_poliza\_vto\_natural']-file\_df['fecha\_siniestro\_ocurrencia'], index=file\_df.index).dt.days  
file\_df['fecha\_diferencia\_siniestro\_vto\_natural\_5dias'] = pd.Series(0, index=file\_df.index)  
file\_df['fecha\_diferencia\_siniestro\_vto\_natural\_15dias'] = pd.Series(0, index=file\_df.index)  
file\_df['fecha\_diferencia\_siniestro\_vto\_natural\_30dias'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['fecha\_diferencia\_siniestro\_vto\_natural'] <= 5,  
 'fecha\_diferencia\_siniestro\_vto\_natural\_5dias'] = 1  
file\_df.loc[  
 file\_df['fecha\_diferencia\_siniestro\_vto\_natural'] <= 15, 'fecha\_diferencia\_siniestro\_vto\_natural\_15dias'] = 1  
file\_df.loc[  
 file\_df['fecha\_diferencia\_siniestro\_vto\_natural'] <= 30, 'fecha\_diferencia\_siniestro\_vto\_natural\_30dias'] = 1

f) If comunication date – sinister date >= 7, then d = 1.

file\_df['fecha\_diferencia\_siniestro\_comunicación'] = pd.Series(  
 file\_df['fecha\_siniestro\_comunicación'] - file\_df['fecha\_siniestro\_ocurrencia'],  
 index=file\_df.index).dt.days  
file\_df['fecha\_diferencia\_comunicación\_outlier'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[  
 file\_df['fecha\_diferencia\_siniestro\_comunicación'] >= 7, 'fecha\_diferencia\_comunicación\_outlier'] = 1  
**del** file\_df['fecha\_siniestro\_comunicación']

g) If loss adjuster visit does not exist (first visit and last visit are null values), d = 1:

file\_df['fecha\_peritaje\_no\_realizado'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[(file\_df['fecha\_primera\_visita\_peritaje'].isnull()) &  
 (file\_df['fecha\_ultima\_visita\_peritaje'].isnull()), 'fecha\_peritaje\_no\_realizado'] = 1

h) If there is more than one visit, d = 1:

file\_df['fecha\_peritaje\_no\_unico'] = pd.Series(1, index=file\_df.index)  
file\_df.loc[(file\_df['fecha\_primera\_visita\_peritaje'].isnull() |  
 file\_df['fecha\_ultima\_visita\_peritaje'].isnull()), 'fecha\_peritaje\_no\_unico'] = 0  
file\_df.loc[(file\_df['fecha\_primera\_visita\_peritaje'] ==  
 file\_df['fecha\_ultima\_visita\_peritaje']), 'fecha\_peritaje\_no\_unico'] = 0

i) If the last visit is NaN, we fill the last visit with the first visit. This will be helpful to calculate date differences.

file\_df['fecha\_primera\_visita\_peritaje'] = file\_df['fecha\_primera\_visita\_peritaje'].fillna(  
 file\_df['fecha\_ultima\_visita\_peritaje'])  
  
file\_df['fecha\_ultima\_visita\_peritaje'] = file\_df['fecha\_ultima\_visita\_peritaje'].fillna(  
 file\_df['fecha\_primera\_visita\_peritaje'])  
  
file\_df['fecha\_primera\_visita\_peritaje'] = pd.to\_datetime(file\_df['fecha\_primera\_visita\_peritaje'],  
 format='%Y-%m-%d', errors='coerce')  
  
file\_df['fecha\_ultima\_visita\_peritaje'] = pd.to\_datetime(file\_df['fecha\_ultima\_visita\_peritaje'],  
 format='%Y-%m-%d', errors='coerce')

j) Diff between first visit and sinister.

file\_df['fecha\_diferencia\_siniestro\_peritaje\_primero'] = pd.Series(  
 file\_df['fecha\_primera\_visita\_peritaje'] - file\_df['fecha\_siniestro\_ocurrencia'],  
 index=file\_df.index).dt.days

k) Diff between last visit and sinister.

file\_df['fecha\_diferencia\_siniestro\_peritaje\_ultimo'] = pd.Series(  
 file\_df['fecha\_ultima\_visita\_peritaje'] - file\_df['fecha\_siniestro\_ocurrencia'],  
 index=file\_df.index).dt.days

l) Diff between last visit and first visit.

file\_df['fecha\_diferencia\_peritajes'] = pd.Series(  
 file\_df['fecha\_ultima\_visita\_peritaje'] - file\_df['fecha\_primera\_visita\_peritaje'],  
 index=file\_df.index).dt.days

5) Calculate Outliers. This outliers will not follow the logic of MAD outliers. We are going to calculate the outliers based in rules which have some intuition (suggested by the Investigation Unity).

file\_df['fecha\_diferencia\_siniestro\_primerperitaje\_outlier'] = pd.Series(0, index=file\_df.index)  
file\_df['fecha\_diferencia\_siniestro\_ultimoperitaje\_outlier'] = pd.Series(0, index=file\_df.index)  
file\_df['fecha\_diferencia\_peritaje\_primerultimo\_outlier'] = pd.Series(0, index=file\_df.index)  
  
file\_df.loc[file\_df[  
 'fecha\_diferencia\_siniestro\_peritaje\_primero'] >= 15,  
 'fecha\_diferencia\_siniestro\_primerperitaje\_outlier'] = 1  
file\_df.loc[file\_df[  
 'fecha\_diferencia\_siniestro\_peritaje\_primero'] >= 30,  
 'fecha\_diferencia\_siniestro\_ultimoperitaje\_outlier'] = 1  
file\_df.loc[file\_df['fecha\_diferencia\_siniestro\_peritaje\_ultimo'] - file\_df[  
 'fecha\_diferencia\_siniestro\_peritaje\_primero'] >= 30,  
 'fecha\_diferencia\_peritaje\_primerultimo\_outlier'] = 1  
  
file\_df.loc[file\_df[  
 'fecha\_diferencia\_siniestro\_peritaje\_primero'] < 0,  
 'fecha\_diferencia\_siniestro\_peritaje\_primero'] = 0  
file\_df.loc[  
 file\_df['fecha\_diferencia\_siniestro\_peritaje\_ultimo'] < 0, 'fecha\_diferencia\_siniestro\_peritaje\_ultimo'] = 0

We create three types only for loss-adjuster visits (the other are implict in the logical dates).

6) We fill the NaN visits with -1 to be not associated to 0 values.

file\_df['fecha\_diferencia\_siniestro\_peritaje\_primero'] = file\_df[  
 'fecha\_diferencia\_siniestro\_peritaje\_primero'].fillna(-1)  
file\_df['fecha\_diferencia\_siniestro\_peritaje\_ultimo'] = file\_df[  
 'fecha\_diferencia\_siniestro\_peritaje\_ultimo'].fillna(-1)  
file\_df['fecha\_diferencia\_peritajes'] = file\_df[  
 'fecha\_diferencia\_peritajes'].fillna(-1)

7) We delete useless variables or variables that are too specific:

del\_variables = ['fecha\_poliza\_emision\_year',  
 'fecha\_poliza\_emision\_month',  
 'fecha\_poliza\_emision\_day',  
 'fecha\_poliza\_emision\_weekday',  
 'fecha\_poliza\_efecto\_natural\_year',  
 'fecha\_poliza\_efecto\_natural\_month',  
 'fecha\_poliza\_efecto\_natural\_day',  
 'fecha\_poliza\_efecto\_natural\_weekday',  
 'fecha\_poliza\_efecto\_mvto\_year',  
 'fecha\_poliza\_efecto\_mvto\_month',  
 'fecha\_poliza\_efecto\_mvto\_day',  
 'fecha\_poliza\_efecto\_mvto\_weekday',  
 'fecha\_poliza\_vto\_movimiento\_year',  
 'fecha\_poliza\_vto\_movimiento\_month',  
 'fecha\_poliza\_vto\_movimiento\_day',  
 'fecha\_poliza\_vto\_movimiento\_weekday',  
 'fecha\_poliza\_vto\_natural\_year',  
 'fecha\_poliza\_vto\_natural\_month',  
 'fecha\_poliza\_vto\_natural\_day',  
 'fecha\_poliza\_vto\_natural\_weekday',  
 'fecha\_siniestro\_ocurrencia\_year',  
 'fecha\_siniestro\_ocurrencia\_month',  
 'fecha\_siniestro\_ocurrencia\_day',  
 'fecha\_siniestro\_ocurrencia\_weekday',  
 'fecha\_primera\_visita\_peritaje\_year',  
 'fecha\_primera\_visita\_peritaje\_month',  
 'fecha\_primera\_visita\_peritaje\_day',  
 'fecha\_primera\_visita\_peritaje\_weekday',  
 'fecha\_ultima\_visita\_peritaje\_year',  
 'fecha\_ultima\_visita\_peritaje\_month',  
 'fecha\_ultima\_visita\_peritaje\_day',  
 'fecha\_ultima\_visita\_peritaje\_weekday',  
 'fecha\_siniestro\_comunicación\_year',  
 'fecha\_siniestro\_comunicación\_month',  
 'fecha\_siniestro\_comunicación\_day',  
 'fecha\_siniestro\_comunicación\_weekday'  
 ]  
  
**for** i **in** del\_variables:  
 **del** file\_df[i]

The input variables are:

|  |  |
| --- | --- |
| **Names** | **Description** |
| id\_siniestro | Sinister Id |
| version\_poliza | Policy version |
| fecha\_poliza\_emision | Emission policy date |
| fecha\_poliza\_efecto\_natural | Effect policy date |
| fecha\_poliza\_efecto\_mvto | Movement Effect date |
| fecha\_poliza\_vto\_movimiento | Movement Expiration Date |
| fecha\_poliza\_vto\_natural | Policy expiration date |
| fecha\_siniestro\_ocurrencia | Sinister date |
| fecha\_primera\_visita\_peritaje | Loss adjuster first visit date |
| fecha\_ultima\_visita\_peritaje | Loss adjuster last visit date |
| d\_fecha\_siniestro\_ocurrencia\_year\_2014 | Dummy Sinister 2014 |
| d\_fecha\_siniestro\_ocurrencia\_year\_2015 | Dummy Sinister 2015 |
| d\_fecha\_siniestro\_ocurrencia\_year\_2016 | Dummy Sinister 2016 |
| dfecha\_siniestro\_ocurrencia\_month\_1 | Dummy sinister Month 1 |
| dfecha\_siniestro\_ocurrencia\_month\_2 | Dummy sinister Month 2 |
| dfecha\_siniestro\_ocurrencia\_month\_3 | Dummy sinister Month 3 |
| dfecha\_siniestro\_ocurrencia\_month\_4 | Dummy sinister Month 4 |
| dfecha\_siniestro\_ocurrencia\_month\_5 | Dummy sinister Month 5 |
| dfecha\_siniestro\_ocurrencia\_month\_6 | Dummy sinister Month 6 |
| dfecha\_siniestro\_ocurrencia\_month\_7 | Dummy sinister Month 7 |
| dfecha\_siniestro\_ocurrencia\_month\_8 | Dummy sinister Month 8 |
| dfecha\_siniestro\_ocurrencia\_month\_9 | Dummy sinister Month 9 |
| dfecha\_siniestro\_ocurrencia\_month\_10 | Dummy sinister Month 10 |
| dfecha\_siniestro\_ocurrencia\_month\_11 | Dummy sinister Month 11 |
| dfecha\_siniestro\_ocurrencia\_month\_12 | Dummy sinister Month 12 |
| fecha\_siniestro\_ocurrencia\_day\_day\_1\_10 | Sinister month day 1-10 |
| fecha\_siniestro\_ocurrencia\_day\_day\_10\_20 | Sinister month day 10-20 |
| fecha\_siniestro\_ocurrencia\_day\_day\_20\_30 | Sinister month day 20-30 |
| d\_fecha\_siniestro\_ocurrencia\_weekday\_0 | Dummy Sinister weekday 0 |
| d\_fecha\_siniestro\_ocurrencia\_weekday\_1 | Dummy Sinister weekday 1 |
| d\_fecha\_siniestro\_ocurrencia\_weekday\_2 | Dummy Sinister weekday 2 |
| d\_fecha\_siniestro\_ocurrencia\_weekday\_3 | Dummy Sinister weekday 3 |
| d\_fecha\_siniestro\_ocurrencia\_weekday\_4 | Dummy Sinister weekday 4 |
| d\_fecha\_siniestro\_ocurrencia\_weekday\_5 | Dummy Sinister weekday 5 |
| d\_fecha\_siniestro\_ocurrencia\_weekday\_6 | Dummy Sinister weekday 6 |
| d\_fecha\_poliza\_efecto\_natural\_year\_2002 | Dummy Policy Effect 2002 |
| d\_fecha\_poliza\_efecto\_natural\_year\_2008 | Dummy Policy Effect 2008 |
| d\_fecha\_poliza\_efecto\_natural\_year\_2011 | Dummy Policy Effect 2011 |
| d\_fecha\_poliza\_efecto\_natural\_year\_2012 | Dummy Policy Effect 2012 |
| d\_fecha\_poliza\_efecto\_natural\_year\_2013 | Dummy Policy Effect 2013 |
| d\_fecha\_poliza\_efecto\_natural\_year\_2014 | Dummy Policy Effect 2014 |
| d\_fecha\_poliza\_efecto\_natural\_year\_2015 | Dummy Policy Effect 2015 |
| d\_fecha\_poliza\_efecto\_natural\_year\_2016 | Dummy Policy Effect 2016 |
| fecha\_poliza\_efecto\_natural\_month\_holiday | Dummy Policy Effect Holiday Months |
| fecha\_poliza\_efecto\_natural\_day\_day\_1\_10 | Dummy Policy effect day 1-10 |
| fecha\_poliza\_efecto\_natural\_day\_day\_10\_20 | Dummy Policy effect day 10-20 |
| fecha\_poliza\_efecto\_natural\_day\_day\_20\_30 | Dummy Policy effect day 20-30 |
| fecha\_poliza\_efecto\_natural\_weekday\_weekend | Dummy Policy natural effect weekend |
| fecha\_poliza\_efecto\_natural\_weekday\_monday | Dummy Policy natural effect on Monday |
| d\_fecha\_poliza\_vto\_natural\_year\_2009 | Dummy Expiration 2009 |
| d\_fecha\_poliza\_vto\_natural\_year\_2012 | Dummy Expiration 2012 |
| d\_fecha\_poliza\_vto\_natural\_year\_2013 | Dummy Expiration 2013 |
| d\_fecha\_poliza\_vto\_natural\_year\_2014 | Dummy Expiration 2014 |
| d\_fecha\_poliza\_vto\_natural\_year\_2015 | Dummy Expiration 2015 |
| d\_fecha\_poliza\_vto\_natural\_year\_2016 | Dummy Expiration 2016 |
| d\_fecha\_poliza\_vto\_natural\_year\_2017 | Dummy Expiration 2017 |
| fecha\_poliza\_vto\_natural\_month\_holiday | Dummy Expiration Holiday |
| fecha\_poliza\_vto\_natural\_day\_day\_1\_10 | Dummy Expiration day 1-10 |
| fecha\_poliza\_vto\_natural\_day\_day\_10\_20 | Dummy Expiration day 10-20 |
| fecha\_poliza\_vto\_natural\_day\_day\_20\_30 | Dummy Expiration day 20-30 |
| fecha\_poliza\_vto\_natural\_weekday\_weekend | Dummy Expiration Weekend |
| fecha\_poliza\_vto\_natural\_weekday\_monday | Dummy Expiration on Monday |
| fecha\_diferencia\_vto\_emision | Diff Emission-Expiration |
| fecha\_indicador\_efecto\_emision | Diff Emission - Effect |
| fecha\_diferencia\_siniestro\_efecto | Diff Sinister-Effect |
| fecha\_diferencia\_siniestro\_efecto\_5dias | Sinister - Effect < 5 days |
| fecha\_diferencia\_siniestro\_efecto\_15dias | Sinister - Effect < 15 days |
| fecha\_diferencia\_siniestro\_efecto\_30dias | Sinister - Effect <30 days |
| fecha\_diferencia\_siniestro\_emision | Diff Sinister-Emission |
| fecha\_diferencia\_siniestro\_emision\_5dias | Sinister - Emission < 5 days |
| fecha\_diferencia\_siniestro\_emision\_15dias | Sinister - Emission < 15 days |
| fecha\_diferencia\_siniestro\_emision\_30dias | Sinister - Emission < 30 days |
| fecha\_diferencia\_siniestro\_vto\_natural | Diff Sinister - Expiration |
| fecha\_diferencia\_siniestro\_vto\_natural\_5dias | Sinister - Expiration < 5 days |
| fecha\_diferencia\_siniestro\_vto\_natural\_15dias | Sinister - Expiration < 15 days |
| fecha\_diferencia\_siniestro\_vto\_natural\_30dias | Sinister - Expiration < 30 days |
| fecha\_diferencia\_siniestro\_comunicación | Diff Sinister - Communication |
| fecha\_diferencia\_comunicación\_outlier | Diff Sinister - Communication > 7 days |
| fecha\_peritaje\_no\_realizado | Empty loss-adjuster visit |
| fecha\_peritaje\_no\_unico | More than one visit |
| fecha\_diferencia\_siniestro\_peritaje\_primero | Diff Sinister - First visit |
| fecha\_diferencia\_siniestro\_peritaje\_ultimo | Diff Sinister - Last visit |
| fecha\_diferencia\_peritajes | Diff Last Visit - First Visit |
| fecha\_diferencia\_siniestro\_primerperitaje\_outlier | Diff Sinister - First visit > 15 days |
| fecha\_diferencia\_siniestro\_ultimoperitaje\_outlier | Diff Sinister - Last visit > 30 days |
| fecha\_diferencia\_peritaje\_primerultimo\_outlier | Diff Last Visit - First Visit > 30 days |

## GUARANTEES

In this bottle we collect information about the coverage and guarantees of the subscribed policy.

TODAVIA NO ESTA

## ASSISTANCE

Assistance Table is a key aspect of the entire analysis. This information arise from Europa, a system that collect the information from the claims call center. It gives us the first contact with the insured person. As we emphasize in Date Table, it is important to remark the timing of fraudulent behavior. First contact implies the first moment of revealing information by the policyholder.

Furthermore, it has the subsequent communications between the company and the affected parts, not only the covered person but also repairer, experts and third parties involved. Moreover, we have descriptions and places of the incident.

One of the problem with this information will be explained later (See Integration Problems section). To sum up, at the moment, the integration with the main database is not possible.

## PROPERTY

Property table supply variables about the insured object. Relevant information about location is the principal aspect of this table.

As we will explain later, here there is an important issue. It is the fact that we cannot uniquely identify the address, because the fields that compound this information is free text type. See Normalized Methodology Applied section.

The input variables are:

|  |  |
| --- | --- |
| **Names** | **Description** |
| id\_fiscal | Id. Fiscal |
| id\_poliza | Número de póliza |
| version\_poliza | Número de versión de la póliza |
| hogar\_tipo\_via | Tipo de vía |
| hogar\_nombre\_via | Nombre de la vía |
| hogar\_numero\_via | Núm. |
| hogar\_info\_adicional | Información adicional |
| hogar\_direccion\_completa | Dirección completa del hogar |
| hogar\_codigo\_unico | Código Único de Hogar |
| hogar\_codigo\_unico\_repite | El código único se repite en otros siniestros que no son de la póliza |
| hogar\_tipo\_vivienda | Tipo de vivienda |
| hogar\_capital\_continente | Es el capital por el que se encuentra asegurado el continente |
| hogar\_capital\_contenido | Es el capital por el que se encuentra asegurado el contenido de la vivienda |
| hogar\_m2 | Metros cuadrados de la vivienda |
| hogar\_carga\_siniestral | Coste total ( Solo Pagos) de los siniestros abiertos y cerrados de la póliza |
| hogar\_cp | CP |
| hogar\_cod\_poblacion | Cód. Población |
| hogar\_poblacion | Población |
| hogar\_provincia | Provinica |
| hogar\_anio\_construccion | Año cons. |
| hogar\_ubicacion | Ubicación |
| hogar\_caracter | Carácter |
| hogar\_uso | Uso de la vivienda |
| hogar\_numero\_seguridad | Hogar cuenta con medidas de Seguridad |
| poliza\_producto\_tecnico | producto técnico |
| poliza\_producto\_comercial | producto comercial |
| auditCodigoSiniestroReferencia | Codigo del siniestro de referencia |

We use the module hogar.py to process the variables.

1- As always we start loading the file, transforming it to a Dataframe and mapping the key variables:

# Load File  
file = ReadCsv.load\_csv(name\_file)  
  
# Transform file to DF  
file\_df = DfUtils.processing\_file(file)  
  
# Map relevant variables  
file\_df = file\_df.rename(columns={'auditCodigoSiniestroReferencia': 'id\_siniestro'})  
file\_df['id\_siniestro'] = file\_df['id\_siniestro'].map(int)

2- We generat dummies by tipo\_hogar. We have to take into account that some categories are named differently (although they are the same). This is because they are different products from different entities. We group them by 5 types:

-Unifamiliar (UF)

-Ático (AT)

-Planta Baja (PB)

-Piso Intermedio (PI)

-Desconocido

file\_df['hogar\_tipo\_vivienda'] = file\_df['hogar\_tipo\_vivienda'].fillna('No Identificado')  
file\_df['hogar\_tipo\_vivienda'] = file\_df['hogar\_tipo\_vivienda'].map(str)  
'''  
001 UNIFAMILIAR ADOSADA   
002 UNIFAMILIAR AISLADA   
003 ÚLTIMO PISO (ÁTICO)   
004 PLANTA BAJA   
005 PISO INTERMEDIO   
006 DESCONOCIDO   
'''  
  
file\_df['hogar\_tipo\_vivienda'] = file\_df.apply(  
 **lambda** x: x['hogar\_tipo\_vivienda'][2:]  
 **if** x['hogar\_tipo\_vivienda'].startswith('00')  
 **else** x['hogar\_tipo\_vivienda']  
 , axis=1)  
  
file\_df['hogar\_tipo\_vivienda'] = file\_df.apply(  
 **lambda** x: x['hogar\_tipo\_vivienda'][1:]  
 **if** x['hogar\_tipo\_vivienda'].startswith('0')  
 **else** x['hogar\_tipo\_vivienda']  
 , axis=1)  
  
file\_df.loc[file\_df['hogar\_tipo\_vivienda'].isin(['1', '2']), 'hogar\_tipo\_vivienda'] = 'UF'  
file\_df.loc[file\_df['hogar\_tipo\_vivienda'] == '3', 'hogar\_tipo\_vivienda'] = 'AT'  
file\_df.loc[file\_df['hogar\_tipo\_vivienda'] == '4', 'hogar\_tipo\_vivienda'] = 'PB'  
file\_df.loc[file\_df['hogar\_tipo\_vivienda'] == '5', 'hogar\_tipo\_vivienda'] = 'PI'  
  
dummy\_hogar = pd.get\_dummies(file\_df['hogar\_tipo\_vivienda'], prefix='d\_tipo\_hogar')  
file\_df = pd.concat([file\_df, dummy\_hogar], axis=1)  
**del** dummy\_hogar

3- Capital Continente and Hogar Captial Contenido. We have many null values (and probably bad inputation). Therefore, we generate dummies identifying empty values.

# 2) Capital continente  
# Creamos una dummy para los no identificados  
file\_df['hogar\_capital\_continente'] = file\_df['hogar\_capital\_continente'].fillna('No\_Identificado')  
file\_df.loc[file\_df['hogar\_capital\_continente'] == 0, 'hogar\_capital\_continente'] = 'No Identificado'  
file\_df['d\_capital\_continente\_no\_identificado'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['hogar\_capital\_continente'] == 'No\_Identificado', 'd\_capital\_continente\_no\_identificado'] = 1  
  
# 3) Hogar capital contenido  
  
# Creamos una dummy para los no identificados  
file\_df['hogar\_capital\_contenido'] = file\_df['hogar\_capital\_contenido'].fillna('No\_Identificado')  
file\_df.loc[file\_df['hogar\_capital\_contenido'] == 0, 'hogar\_capital\_contenido'] = 'No Identificado'  
file\_df['d\_capital\_contenido\_no\_identificado'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['hogar\_capital\_contenido'] == 'No\_Identificado', 'd\_capital\_contenido\_no\_identificado'] = 1

4- M2: We do the same for M2. It highly probable they are not well inputed. After, we will identify the bad values.

# 4) M2  
# Creamos una dummy para los no identificados  
file\_df['hogar\_m2'] = file\_df['hogar\_m2'].fillna('No\_Identificado')  
file\_df.loc[file\_df['hogar\_m2'] == 0, 'hogar\_m2'] = 'No Identificado'  
file\_df['d\_hogar\_m2\_no\_identificado'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['hogar\_m2'] == 'No\_Identificado', 'd\_hogar\_m2\_no\_identificado'] = 1

5- Año de Construcción: The same problem as before. But here is more obviously. We calculate automatically the current year (therefore it will update every time the program is running) and replace bad values with it. Just the values that are greater than the current year.

year\_today = date.today().year  
file\_df['hogar\_anio\_construccion'] = file\_df['hogar\_anio\_construccion'].convert\_objects(convert\_numeric=**True**)  
file\_df.loc[file\_df['hogar\_anio\_construccion'] > year\_today, 'hogar\_anio\_construccion'] = np.nan

6-Hogar ubicación: We fill the NaN values with an additional category (‘No Identificado’). Then, we get dummies.

file\_df['hogar\_ubicacion'] = file\_df['hogar\_ubicacion'].fillna('No\_Identificado')  
dummy\_hogar = pd.get\_dummies(file\_df['hogar\_ubicacion'], prefix='d\_hogar\_ubicacion')  
file\_df = pd.concat([file\_df, dummy\_hogar], axis=1)  
**del** dummy\_hogar

7- Carácter Hogar: We have the same mismatch problem that in tipo\_hogar. We group the categories as follows:

-Propietario (P)

-Inquilino (I)

-No Identificado

file\_df['hogar\_caracter'] = file\_df['hogar\_caracter'].fillna('No Identificado')  
file\_df['hogar\_caracter'] = file\_df['hogar\_caracter'].map(str)  
  
file\_df['hogar\_caracter'] = file\_df.apply(  
 **lambda** x: x['hogar\_caracter'][2:]  
 **if** x['hogar\_caracter'].startswith('00')  
 **else** x['hogar\_caracter']  
 , axis=1)  
  
file\_df['hogar\_caracter'] = file\_df.apply(  
 **lambda** x: x['hogar\_caracter'][1:]  
 **if** x['hogar\_caracter'].startswith('0')  
 **else** x['hogar\_caracter']  
 , axis=1)  
  
file\_df.loc[file\_df['hogar\_caracter'].isin(['1', '2', '5']), 'hogar\_caracter'] = 'P'  
file\_df.loc[file\_df['hogar\_caracter'].isin(['3', '4']), 'hogar\_caracter'] = 'I'  
file\_df.loc[file\_df['hogar\_caracter'] == '6', 'hogar\_caracter'] = 'No Identificado'  
  
dummy\_hogar = pd.get\_dummies(file\_df['hogar\_caracter'], prefix='d\_hogar\_caracter')  
file\_df = pd.concat([file\_df, dummy\_hogar], axis=1)  
**del** dummy\_hogar

8- Hogar uso: Same problem as previous step. We group into:

-Principal

-Secundario

-No Identificado

file\_df['hogar\_uso'] = file\_df['hogar\_uso'].fillna('No Identificado')  
file\_df['hogar\_uso'] = file\_df['hogar\_uso'].map(str)  
  
file\_df['hogar\_uso'] = file\_df.apply(  
 **lambda** x: x['hogar\_uso'][2:]  
 **if** x['hogar\_uso'].startswith('00')  
 **else** x['hogar\_uso']  
 , axis=1)  
  
file\_df['hogar\_uso'] = file\_df.apply(  
 **lambda** x: x['hogar\_uso'][1:]  
 **if** x['hogar\_uso'].startswith('0')  
 **else** x['hogar\_uso']  
 , axis=1)  
  
file\_df.loc[file\_df['hogar\_uso'] == '1', 'hogar\_uso'] = 'P'  
file\_df.loc[file\_df['hogar\_uso'] == '2', 'hogar\_uso'] = 'S'  
file\_df.loc[file\_df['hogar\_uso'] == '3', 'hogar\_uso'] = 'No Identificado'  
  
dummy\_hogar = pd.get\_dummies(file\_df['hogar\_uso'], prefix='d\_hogar\_uso')  
file\_df = pd.concat([file\_df, dummy\_hogar], axis=1)  
**del** dummy\_hogar

9-Número de seguridades: We group the securities into 4 groups as follows:

|  |  |
| --- | --- |
| Número de Seguridades | Categorías |
| 0 | Null |
| 1 | Baja |
| 2, 3, 4 | Media |
| 5, 6 | Alta |

file\_df['hogar\_seguridad\_null'] = pd.Series(0, index=file\_df.index)  
file\_df['hogar\_seguridad\_alta'] = pd.Series(0, index=file\_df.index)  
file\_df['hogar\_seguridad\_baja'] = pd.Series(0, index=file\_df.index)  
file\_df['hogar\_seguridad\_media'] = pd.Series(0, index=file\_df.index)  
  
file\_df.loc[file\_df['hogar\_numero\_seguridad'] == 0, 'hogar\_seguridad\_null'] = 1  
file\_df.loc[file\_df['hogar\_numero\_seguridad'] == 1, 'hogar\_seguridad\_baja'] = 1  
file\_df.loc[file\_df['hogar\_numero\_seguridad'].isin([5, 6]), 'hogar\_seguridad\_alta'] = 1  
file\_df.loc[file\_df['hogar\_numero\_seguridad'].isin([2, 3, 4]), 'hogar\_seguridad\_media'] = 1

10- We have an unapplied method that correct the addresses using Fuzzy Rules and Censo Callejero. The problem is that it takes too long time, and we use it just for creating two variables: If the property is involved in another sinister, and if the address is bad specified.

'''  
query\_file = file\_df[['hogar\_tipo\_via', 'hogar\_nombre\_via','hogar\_numero\_via',  
 'hogar\_info\_adicional']]  
  
query\_file = query\_file[1:]  
query\_file = query\_file.fillna('')  
query\_file = query\_file.values.tolist()  
sys.stdout = open(os.devnull, 'w')  
output, mean\_score, best\_score, coincidence\_name\_list = frules\_direccion.frule\_direction(query\_file,range\_acceptance=80)  
sys.stdout = sys.\_\_stdout\_\_  
coincidence\_name\_list = pd.DataFrame(coincidence\_name\_list, columns = ['hogar\_corrected\_name'])  
file\_df = pd.concat([file\_df, coincidence\_name\_list], axis = 1)  
del coincidence\_name\_list  
  
file\_df['hogar\_corrected\_repetido'] = file\_df.groupby('hogar\_corrected\_name')['hogar\_corrected\_name'].transform('count')  
  
file\_df['hogar\_address\_bad\_id'] = pd.Series(0, index = file\_df.index)  
file\_df.loc[file\_df['hogar\_corrected\_name'].str.contains('N/A'), 'hogar\_address\_bad\_id'] = 1  
'''

11-Outliers: Here, the main objective is identified the bad imputed values. Therefore, we apply our MAD algorithm for capital\_continente, capital\_contenido, hogar\_m2, hogar\_anio\_construccion. If we identify an outlier, we change the value with a null. Later, we will estimate them using Multioutput Regression. Once we finish this process, we recalculate the outliers without this NaN values.

outliers = ['hogar\_capital\_continente', 'hogar\_capital\_contenido', 'hogar\_m2', 'hogar\_anio\_construccion']  
**if** test == **False**:  
 **for** i **in** outliers:  
 Outliers.outliers\_mad(file\_df, i, just\_count\_zero=**True**)  
 outlier\_name = str(i) + '\_mad\_outlier'  
 file\_df.loc[file\_df[outlier\_name] == 1, i] = np.NaN  
 **del** file\_df[outlier\_name]  
 Outliers.outliers\_mad(file\_df, i, just\_count\_zero=**True**)  
  
  
**else**:  
 base\_df = ReadCsv.load\_csv(STRING.hogar\_file)  
 base\_df = DfUtils.processing\_file(base\_df)  
 base\_df = base\_df[outliers]  
 **for** i **in** outliers:  
 Outliers.outliers\_test\_values(file\_df, base\_df, i, just\_count\_zero=**True**)  
 outlier\_name = str(i) + '\_mad\_outlier'  
 file\_df.loc[file\_df[outlier\_name] == 1, i] = np.NaN  
 **del** file\_df[outlier\_name]  
 Outliers.outliers\_test\_values(file\_df, i, just\_count\_zero=**True**)  
 **del** base\_df

12- Finally we delete the useless variables.

del\_variables = ['id\_poliza', 'id\_fiscal', 'hogar\_tipo\_via', 'hogar\_nombre\_via', 'hogar\_numero\_via', 'hogar\_info\_adicional',  
 'hogar\_direccion\_completa', 'hogar\_codigo\_unico',  
 'hogar\_carga\_siniestral', 'hogar\_cod\_poblacion', 'hogar\_anio\_construccion',  
 'poliza\_producto\_tecnico', 'poliza\_producto\_comercial', 'hogar\_tipo\_vivienda',  
 'hogar\_poblacion', 'hogar\_provincia', 'hogar\_ubicacion', 'hogar\_caracter',  
 'hogar\_uso'  
 ]  
  
**for** i **in** del\_variables:  
 **del** file\_df[i]

The output is:

FALTA LA NUEVA TABLA

## PAYMENTS

It is referred to the policy payments made by the insured. Chiefly we are trying to reflect some insight about late payment and a possible relation with fraud claims.

The input variables are:

|  |  |
| --- | --- |
| **Name** | **Description** |
| id\_fiscal | Id. Fiscal |
| id\_poliza | Número de póliza |
| version\_poliza | Número de versión de la póliza |
| pago\_canal\_cobro\_1er\_recibo | Canal cobro 1er. Recibo |
| pago\_IBAN | CUENTA IBAN |
| pago\_situacion\_recibo | Situación del Recibo |
| pago\_pendientes\_recibo | Recibos pendientes de pago |
| pago\_morosidad | Indice de morosidad |
| pago\_forma\_curso | Forma de Pago de la Anualidad en Curso |
| pago\_forma\_sucesivas | Forma de Pago de la Anualidad sucesivas |
| auditCodigoSiniestroReferencia | Codigo del siniestro de referencia |
| auditFechaOcurrenciaSiniestroReferencia | Fecha de ocurrencia del siniestro de referencia |

We use the module pago\_info.py to process the variables.

1- As always we start loading the file, transforming it to a Dataframe and mapping the key variables:

# Load File  
file = ReadCsv.load\_csv(name\_file)  
  
# Transform file to DF  
file\_df = DfUtils.processing\_file(file)  
file\_df = file\_df.sort\_values(by=['auditCodigoSiniestroReferencia'],  
 ascending=[**True**])  
file\_df = file\_df.rename(columns={'auditCodigoSiniestroReferencia': 'id\_siniestro'})  
  
# Define Principal ID dtypes  
file\_df['id\_siniestro'] = file\_df['id\_siniestro'].map(int)

2-Pago forma sucesiva no coincide: We compare the first mean payment with the sucesives paid methods. If they differ, this variables is equal to one.

file\_df['pago\_forma\_curso\_sucesiva\_no\_coincide'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['pago\_forma\_sucesivas'] != file\_df['pago\_forma\_curso'],  
 'pago\_forma\_curso\_sucesiva\_no\_coincide'] = 1

3-Pago canal cobro 1er Recibo: We have the cod for each type of means. We use a dictionary to map the key with the values.

primerer\_recibo = {'IN': 'Intermediario', 'BC': 'Banco', 'CD': 'Ventanilla', 'CO': 'Negocio Aceptado',  
 'CC': 'Tarjeta Credito'}

**for** key, value **in** primerer\_recibo.items():  
 file\_df.loc[file\_df['pago\_canal\_cobro\_1er\_recibo'] == key, 'pago\_canal\_cobro\_1er\_recibo'] = value

4-Situacion recibo: The same as previous.

situacion\_recibo = {'A': 'Anulado', 'E': 'Emitido', 'L': 'Liquidado', 'P': 'Pendiente',  
 'A|L': 'Anulado-Liquidado', 'L|P': 'Liquidado-Pendiente'}

**for** key, value **in** situacion\_recibo.items():  
 file\_df.loc[file\_df['pago\_situacion\_recibo'] == key, 'pago\_situacion\_recibo'] = value

5-Forma Pago: The same as previous.

forma\_pago = {'A': 'Anual', 'B': 'Bimestral', 'C': 'Cuatrimestral', 'E': 'Extraordinaria',  
 'F': 'Fraccionada', 'M': 'Mensual', 'P': 'Plan Pagos', 'S': 'Semestral',  
 'T': 'Trimestral', 'U': 'Unica', 'Z': 'Aperiodica'}

**for** key, value **in** forma\_pago.items():  
 file\_df.loc[file\_df['pago\_forma\_curso'] == key, 'pago\_forma\_curso'] = value

6- Pago Morosidad: This is a created variable using the debt pays. It takes a range between 0-10 using the proportion of (debt pays / Total pays). Here we have some aspects to clarify.

-If a policy has just one version, and not valid bills, it was inputed a 1.

-There are policies with several versions but with all not valid bills. This policies are represented as null values. We cannot put 1, because it is improbable that it has only invalid bills.

Due to this problems, we have to construct categories:

|  |  |
| --- | --- |
| **Default Rate** | **Category** |
| 0 | Never Paid |
| 1-5 | Bad |
| 6-8 | Regular |
| 9-10 | Good |

7- Categorical Variables: We create dummy variables for each category.

# CATEGORICAL: We convert the categorical variables to dummies  
categorical\_var = ['pago\_canal\_cobro\_1er\_recibo', 'pago\_situacion\_recibo', 'pago\_morosidad',  
 'pago\_forma\_curso']  
  
**for** i **in** categorical\_var:  
 prefijo = 'd\_' + i  
 dummy\_i = pd.get\_dummies(file\_df[i], prefix=prefijo, dummy\_na=**True**)  
 file\_df = pd.concat([file\_df, dummy\_i], axis=1)  
 **del** file\_df[i]  
 **del** dummy\_i

Also we add a variable that identify if there is a mix of receipts:

file\_df['pago\_cambio\_situacion\_recibo'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[(file\_df['d\_pago\_situacion\_recibo\_Anulado-Liquidado'] == 1) |  
 (file\_df['d\_pago\_situacion\_recibo\_Liquidado-Pendiente'] == 1),  
 'pago\_cambio\_situacion\_recibo'] = 1

8- IBAN: We create two essential variables. The first indicates if the IBAN is informed. The second is more complex. It invocates the Blacklist and check if the IBAN is in the table and how many times.

# IBAN INFORMADO: We check if the IBAN is informed  
file\_df['pago\_iban\_informado'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['pago\_IBAN'].notnull(), 'pago\_iban\_informado'] = 1  
  
# IBAN FRAUDE: We check if the IBAN is associated with a previous Fraud Sinister  
file\_blacklist\_IBAN = ReadCsv.load\_csv(STRING.processed\_IBAN)  
df\_bl\_IBAN = DfUtils.processing\_file(file\_blacklist\_IBAN, delimiter=';')  
df\_bl\_IBAN = df\_bl\_IBAN.drop\_duplicates(subset=['IBAN'], keep='last')  
file\_df['pago\_IBAN'] = file\_df['pago\_IBAN'].map(str)  
df\_bl\_IBAN['IBAN'] = df\_bl\_IBAN['IBAN'].map(str)  
file\_df = pd.merge(file\_df, df\_bl\_IBAN, how='left', left\_on='pago\_IBAN', right\_on='IBAN')  
file\_df['pago\_iban\_blacklist'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['IBAN'].notnull(), 'pago\_iban\_blacklist'] = 1  
**del** df\_bl\_IBAN  
**del** file\_df['IBAN']

9-We delete the useless variables:

# DELETE VARIABLES: Delete useless or transformed variables  
delete\_var = ['id\_fiscal', 'id\_poliza', 'version\_poliza',  
 'auditFechaOcurrenciaSiniestroReferencia', 'pago\_forma\_sucesivas', 'pago\_IBAN']

The output is:

|  |  |
| --- | --- |
| **Name** | **Description** |
| pago\_pendientes\_recibo | Payment pending receipt |
| id\_siniestro | Sinister id |
| pago\_forma\_curso\_sucesiva\_no\_coincide | Sucesive paid methods and first paid method differs |
| d\_pago\_canal\_cobro\_1er\_recibo\_Banco | First payment Bank |
| d\_pago\_canal\_cobro\_1er\_recibo\_Intermediario | First payment Intermediary |
| d\_pago\_canal\_cobro\_1er\_recibo\_Ventanilla | First payment Cash |
| d\_pago\_canal\_cobro\_1er\_recibo\_nan | First payment non-identified |
| d\_pago\_situacion\_recibo\_Anulado | Canceled receipt |
| d\_pago\_situacion\_recibo\_Anulado-Liquidado | Canceled and liquidated receipts |
| d\_pago\_situacion\_recibo\_Liquidado | Liquidated receipts |
| d\_pago\_situacion\_recibo\_Liquidado-Pendiente | Pending and liquidated receipts |
| d\_pago\_situacion\_recibo\_Pendiente | Pending receipts |
| d\_pago\_situacion\_recibo\_nan | non-identified receipts |
| d\_pago\_morosidad\_Bueno | Default rate good |
| d\_pago\_morosidad\_Malo | Default Rate bad |
| d\_pago\_morosidad\_Regular | Default Rate regular |
| d\_pago\_morosidad\_nan | Default rate non-identified |
| d\_pago\_forma\_curso\_Anual | Anual receipts |
| d\_pago\_forma\_curso\_Mensual | Mensual receipts |
| d\_pago\_forma\_curso\_Semestral | Semestral receipts |
| d\_pago\_forma\_curso\_Trimestral | Trimestral receipts |
| d\_pago\_forma\_curso\_nan | Non identified time receipts |
| pago\_cambio\_situacion\_recibo | Recepit State |
| pago\_iban\_informado | IBAN informed |
| pago\_iban\_blacklist | How many times the IBAN is in the blacklist |

## POLICY

Policy summarize main variables of the policy contract. Product contracted, changes, duration, etc.

The input variables are:

|  |  |
| --- | --- |
| **Name** | **Description** |
| id\_fiscal | Id. Fiscal |
| id\_poliza | Número de póliza |
| version\_poliza | Número de versión de la póliza |
| id\_producto | Código de Producto |
| poliza\_cod\_comercial | Código Comercial |
| poliza\_entidad\_legal | Legal entity |
| poliza\_descripcion\_producto | Descripción del producto de la póliza seleccionada |
| poliza\_cod\_estructura | Código de la Estructura |
| poliza\_desc\_estructura | Descripción de la Estructura |
| poliza\_canal | Canal de la Póliza |
| poliza\_codigo\_negocio | Código negocio de la póliza seleccionada |
| poliza\_nombre\_negocio | Nombre del Negocio de la Póliza seleccionada |
| poliza\_cod\_intermediario | Código del Intermediario Productor Priimero |
| poliza\_denominacion\_intermediario | Denominación del intermediario Productor Primero |
| poliza\_nif\_intermediario | NIF/ CIF del intermediario |
| poliza\_id\_mediador\_productor\_secundario | Cód. interm. |
| poliza\_mediador\_denominacion\_productor\_secundario | Denominación intermediario |
| poliza\_nif\_intermediario\_secundario | NIF/ CIF del intermediario |
| poliza\_id\_mediador\_productor\_tercero | Cód. interm. |
| poliza\_mediador\_denominacion\_productor\_tercero | Denominación intermediario |
| poliza\_nif\_intermediario\_tercero | NIF/ CIF del intermediario |
| poliza\_id\_mediador\_gestor\_primer\_recibo | Cód. interm. |
| poliza\_mediador\_denominacion\_gestor\_primer\_recibo | Denominación intermediario |
| poliza\_nif\_intermediario\_primer\_recibo | NIF/ CIF del intermediario |
| poliza\_id\_mediador\_gestor\_recibo\_sucesivo | Cód. interm. |
| poliza\_mediador\_denominacion\_gestor\_recibo\_sucesivo | Denominación intermediario |
| poliza\_nif\_intermediario\_recibo\_sucesivos | NIF/ CIF del intermediario |
| poliza\_cambios\_cobertura | Cambio paquete / coberturas  \*Se basa en el movimiento de la póliza |
| poliza\_cambio\_datos | Modificación de datos \*Se basa en el movimiento de la póliza |
| poliza\_suplementos | Suplementos, suplementos al vencimiento |
| poliza\_ultimo\_movimiento | Indica el último movimiento que ha sufrido la póliza (En INFO es el tipo de movimiento) |
| poliza\_motivo\_ultimo\_movimiento | Indica los motivos del último movimiento que ha sufrido la póliza |
| poliza\_franquicia | Franquicia general |
| poliza\_duracion | Duración |
| poliza\_cesion\_derechos | Titular cesión de derechos |
| poliza\_tipo\_movimiento\_sospechoso | Motivo de Movimiento Último Sospechoso |
| poliza\_credit\_scoring | Credit Scoring |
| audit\_siniestro\_referencia | Siniestro de referencia |
| poliza\_subcanal | Subcanal de la Póliza |

We use the module poliza\_info.py to process the variables. It is important to remark that we are also using the intermediary variables.

1- As always we start loading the file, transforming it to a Dataframe and mapping the key variables:

# Load File  
file = ReadCsv.load\_csv(name\_file)  
  
# Transform file to DF  
file\_df = DfUtils.processing\_file(file, delimiter=',')  
  
file\_df = file\_df.rename(columns={'audit\_siniestro\_referencia': 'id\_siniestro'})  
file\_df = file\_df.dropna(subset=['id\_siniestro'])  
file\_df = file\_df.dropna(subset=['id\_poliza'])  
  
# Map Important Variables  
file\_df['id\_siniestro'] = file\_df['id\_siniestro'].map(int)  
file\_df['poliza\_cod\_intermediario'] = file\_df['poliza\_cod\_intermediario'].map(int)  
file\_df['poliza\_id\_mediador\_gestor\_primer\_recibo'] = file\_df['poliza\_id\_mediador\_gestor\_primer\_recibo'].fillna(-1)  
file\_df['poliza\_id\_mediador\_gestor\_primer\_recibo'] = file\_df['poliza\_id\_mediador\_gestor\_primer\_recibo'].map(int)  
file\_df['poliza\_id\_mediador\_gestor\_recibo\_sucesivo'] = \  
 file\_df['poliza\_id\_mediador\_gestor\_recibo\_sucesivo'].map(int)

2 – We check if the intermediary is the same, both on the first bill and on succesives bills:

# MEDIADOR PRIMER RECIBO and SUCESIVOS do not match  
file\_df['poliza\_mediador\_primero\_sucesivo\_no\_coincidencia'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['poliza\_id\_mediador\_gestor\_primer\_recibo'] != file\_df[  
 'poliza\_id\_mediador\_gestor\_recibo\_sucesivo'], 'poliza\_mediador\_primero\_sucesivo\_no\_coincidencia'] = 1

3 – We cross the policy table with the intermediary table. First we do that for the intermediary code of the current policy:

# MATCH WITH MEDIADOR FILE  
file\_mediador = ReadCsv.load\_csv(STRING.processed\_mediador\_info)  
df\_mediador = DfUtils.processing\_file(file\_mediador, delimiter=';')  
df\_mediador['mediador\_cod\_intermediario'] = df\_mediador['mediador\_cod\_intermediario'].map(int)  
  
# cod\_intermediario  
file\_df = file\_df.sort\_values(ascending=[**True**], by=['poliza\_cod\_intermediario'])  
file\_df = pd.merge(file\_df, df\_mediador, left\_on='poliza\_cod\_intermediario',  
 right\_on='mediador\_cod\_intermediario', how='left', suffixes=('', ''))  
mediador\_cols = df\_mediador.columns.values.tolist()  
**for** i **in** mediador\_cols:  
 file\_df[i] = file\_df[i].fillna(-1)  
**del** file\_df['mediador\_cod\_intermediario']

4 – But then, we made for intermediary present in the first bill and for the intermediary associated in the next bills. However, here we reduce the amount of variables to that variables that seem more relevant:

# For the remaining codes we select only variables we think are really relevant.  
key\_values = ['mediador\_cod\_intermediario', 'mediador\_numero\_polizas\_count',  
 'mediador\_numero\_polizas\_vigor\_count',  
 'mediador\_numero\_siniestros\_count',  
 'mediador\_numero\_siniestros\_declarados\_count',  
 'mediador\_numero\_siniestros\_fraude\_count',  
 'mediador\_numero\_siniestros\_pagados\_count',  
 'mediador\_numero\_polizas\_hogar',  
 'mediador\_numero\_polizas\_vigor\_hogar',  
 'mediador\_numero\_siniestros\_hogar',  
 'mediador\_numero\_siniestros\_declarados\_hogar',  
 'mediador\_numero\_siniestros\_fraude\_hogar',  
 'mediador\_numero\_siniestros\_pagados\_hogar',  
 'mediador\_poliza\_vigor\_total',  
 'mediador\_siniestros\_declarado\_total',  
 'mediador\_siniestros\_fraude\_total',  
 'mediador\_siniestros\_pagados\_total',  
 'mediador\_numero\_siniestros/poliza',  
 'mediador\_numero\_siniestros\_declarados/poliza',  
 'mediador\_numero\_siniestros\_fraude/poliza',  
 'mediador\_numero\_siniestros\_pagados/poliza',  
 'mediador\_poliza\_vigor\_total\_hogar',  
 'mediador\_siniestros\_declarado\_total\_hogar',  
 'mediador\_siniestros\_fraude\_total\_hogar',  
 'mediador\_siniestros\_pagados\_total\_hogar',  
 'mediador\_numero\_siniestros/poliza\_hogar',  
 'mediador\_numero\_siniestros\_declarados/poliza\_hogar',  
 'mediador\_numero\_siniestros\_fraude/poliza\_hogar',  
 'mediador\_numero\_siniestros\_pagados/poliza\_hogar',  
 'mediador\_numero\_polizashogar/total',  
 'mediador\_numero\_polizas\_vigorhogar/total',  
 'mediador\_numero\_siniestroshogar/total',  
 'mediador\_numero\_siniestros\_declaradoshogar/total',  
 'mediador\_numero\_siniestros\_fraudehogar/total',  
 'mediador\_numero\_siniestros\_pagadoshogar/total'  
 ]  
df\_mediador = df\_mediador[key\_values]  
  
# primer recibo  
file\_df = file\_df.sort\_values(ascending=[**True**], by=['poliza\_id\_mediador\_gestor\_primer\_recibo'])  
file\_df = pd.merge(file\_df, df\_mediador, left\_on='poliza\_id\_mediador\_gestor\_primer\_recibo',  
 right\_on='mediador\_cod\_intermediario', how='left', suffixes=('', '\_primer\_recibo'))  
**del** file\_df['poliza\_id\_mediador\_gestor\_primer\_recibo']  
  
# recibos sucesivos  
file\_df = file\_df.sort\_values(ascending=[**True**], by=['poliza\_id\_mediador\_gestor\_recibo\_sucesivo'])  
file\_df = pd.merge(file\_df, df\_mediador, left\_on='poliza\_id\_mediador\_gestor\_recibo\_sucesivo',  
 right\_on='mediador\_cod\_intermediario', how='left',  
 suffixes=('', '\_sucesivos\_recibo'))  
**del** file\_df['poliza\_id\_mediador\_gestor\_recibo\_sucesivo']  
mediador\_cols = df\_mediador.columns.values.tolist()  
**for** i **in** mediador\_cols:  
 file\_df[i] = file\_df[i].fillna(-1)  
**del** df\_mediador

5 – Using a dict, we change the code values for POLIZA CANAL.

# POLIZA CANAL: We match with the description  
file\_df['poliza\_canal'] = file\_df['poliza\_canal'].map(int)  
canal = {1: 'Mediador', 5: 'Colectivo', 6: 'Grandes\_Dist', 7: 'Deutsche', 9: 'Vida', 10: 'CAS'}  
**for** key, value **in** canal.items():  
 file\_df.loc[file\_df['poliza\_canal'] == key, 'poliza\_canal'] = value

6- We fill the empty Credit Scoring values with ‘No Informado’:

# POLIZA CREDIT SCORING: We match with the description  
file\_df['poliza\_credit\_scoring'] = file\_df['poliza\_credit\_scoring'].fillna('No Informado')

7- As always, categorical variables are transformed as dummies.

# CATEGORICAL VAR: We transform categorical values to DUMMIES  
categorical\_var = ['poliza\_desc\_estructura', 'poliza\_canal', 'poliza\_duracion', 'poliza\_credit\_scoring']  
**for** i **in** categorical\_var:  
 dummy = pd.get\_dummies(file\_df[i], prefix='d\_'+i)  
 file\_df = pd.concat([file\_df, dummy], axis=1)  
 **del** dummy  
 **del** file\_df[i]

8- Dummy for the last movement:

# TIPOS MOVIMENTO: Creamos Dummies para los últimos tipos de movimiento  
dummy\_movimiento = pd.get\_dummies(file\_df['poliza\_ultimo\_movimiento'],  
 prefix='d\_poliza\_ultimo\_movimiento')  
file\_df = pd.concat([file\_df, dummy\_movimiento], axis=1)  
**del** dummy\_movimiento  
**del** file\_df['poliza\_ultimo\_movimiento']

9- We redefine the variable ‘cesion\_derechos’. This is a check that open a text field if ‘yes’ it is selected. So we need to identify ‘No’ cases or the remainer cases (Whatever text that is different to ‘No’).

# CESION DE DERECHOS: Existe cesión de derechos?  
file\_df['poliza\_cesion\_derechos'] = file\_df['poliza\_cesion\_derechos'].map(str)  
file\_df.loc[file\_df['poliza\_cesion\_derechos'] == 'No', 'poliza\_cesion\_derechos'] = 0  
file\_df.loc[file\_df['poliza\_cesion\_derechos'] != 'No', 'poliza\_cesion\_derechos'] = 1  
**del** file\_df['poliza\_cesion\_derechos']

10- We delete useless variables:

# DELETE USELESS VAR  
delete\_var = ['id\_fiscal', 'id\_poliza', 'id\_producto', 'poliza\_cod\_comercial', 'poliza\_entidad\_legal',  
 'poliza\_descripcion\_producto', 'poliza\_nif\_intermediario',  
 'poliza\_id\_mediador\_productor\_secundario', 'poliza\_mediador\_denominacion\_productor\_secundario',  
 'poliza\_nif\_intermediario\_secundario', 'poliza\_id\_mediador\_productor\_tercero',  
 'poliza\_mediador\_denominacion\_productor\_tercero', 'poliza\_nif\_intermediario\_tercero',  
 'poliza\_mediador\_denominacion\_gestor\_primer\_recibo', 'poliza\_nif\_intermediario\_primer\_recibo',  
 'poliza\_mediador\_denominacion\_gestor\_recibo\_sucesivo', 'poliza\_nif\_intermediario\_recibo\_sucesivos',  
 'poliza\_cod\_estructura', 'poliza\_codigo\_negocio', 'poliza\_nombre\_negocio',  
 'poliza\_cod\_intermediario', 'poliza\_denominacion\_intermediario', 'mediador\_cod\_intermediario',  
 'mediador\_cod\_intermediario\_sucesivos\_recibo', 'poliza\_motivo\_ultimo\_movimiento'  
 ]  
**for** i **in** delete\_var:  
 **del** file\_df[i]

The output is:

|  |  |
| --- | --- |
| **Names** | **Description** |
| version\_poliza | Policy Version |
| poliza\_cambios\_cobertura | If exists coverage changes |
| poliza\_cambio\_datos | If exists personal data changes |
| poliza\_suplementos | If exist suplement changes |
| poliza\_franquicia | If the policy has a deductible type |
| poliza\_tipo\_movimiento\_sospechoso | If the last movement was suspicios |
| id\_siniestro | Sinister code |
| poliza\_mediador\_primero\_sucesivo\_no\_coincidencia | Intermediary differs on first and succesive bills |
| mediador\_clase\_intermediario | Intermediary class |
| mediador\_estado |
| mediador\_producto\_count |
| mediador\_cod\_count\_blacklist |
| mediador\_cod\_blacklist |
| mediador\_nif\_count\_blacklist |
| mediador\_nif\_blacklist |
| mediador\_clase\_intermediario\_AC |
| mediador\_clase\_intermediario\_AE |
| mediador\_clase\_intermediario\_AF |
| mediador\_clase\_intermediario\_AV |
| mediador\_clase\_intermediario\_BA |
| mediador\_clase\_intermediario\_BC |
| mediador\_clase\_intermediario\_BM |
| mediador\_clase\_intermediario\_CB |
| mediador\_clase\_intermediario\_CC |
| mediador\_clase\_intermediario\_CD |
| mediador\_clase\_intermediario\_CE |
| mediador\_clase\_intermediario\_CF |
| mediador\_clase\_intermediario\_CO |
| mediador\_clase\_intermediario\_CR |
| mediador\_clase\_intermediario\_EC |
| mediador\_clase\_intermediario\_EM |
| mediador\_clase\_intermediario\_I |
| mediador\_clase\_intermediario\_LI |
| mediador\_clase\_intermediario\_LP |
| mediador\_clase\_intermediario\_OC |
| mediador\_clase\_intermediario\_OE |
| mediador\_clase\_intermediario\_OP |
| mediador\_clase\_intermediario\_OS |
| mediador\_clase\_intermediario\_OV |
| mediador\_clase\_intermediario\_PL |
| mediador\_clase\_intermediario\_SU |
| mediador\_clase\_intermediario\_VD |
| mediador\_estado\_Activo |
| mediador\_estado\_Inactivo |
| mediador\_estado\_Pendiente |
| mediador\_estado\_Tramite |
| mediador\_agrup\_producto\_ACCIDENTES |
| mediador\_agrup\_producto\_AUTOS |
| mediador\_agrup\_producto\_COLECTIVOAHORROPERIODICAS |
| mediador\_agrup\_producto\_COLECTIVOAHORROUNICAS |
| mediador\_agrup\_producto\_COLECTIVORIESGOPERIODICAS |
| mediador\_agrup\_producto\_COMERCIOS |
| mediador\_agrup\_producto\_DECESOS |
| mediador\_agrup\_producto\_EMBARCACIONES |
| mediador\_agrup\_producto\_EPSV |
| mediador\_agrup\_producto\_FLOTAS |
| mediador\_agrup\_producto\_HOGAR |
| mediador\_agrup\_producto\_INDIVIDUALAHORROPERIODICAS |
| mediador\_agrup\_producto\_INDIVIDUALAHORROUNICAS |
| mediador\_agrup\_producto\_INDIVIDUALPPA |
| mediador\_agrup\_producto\_INDIVIDUALRIESGOPERIODICAS |
| mediador\_agrup\_producto\_INDIVIDUALRIESGOUNICAS |
| mediador\_agrup\_producto\_INDIVIDUALUNITLINKPERIODICAS |
| mediador\_agrup\_producto\_INDIVIDUALUNITLINKUNICAS |
| mediador\_agrup\_producto\_INMUEBLES |
| mediador\_agrup\_producto\_OTROS |
| mediador\_agrup\_producto\_PATRIMONIALES |
| mediador\_agrup\_producto\_PLANESPENSIONES |
| mediador\_agrup\_producto\_PROT.PAGOS |
| mediador\_agrup\_producto\_RESP.CIVIL |
| mediador\_agrup\_producto\_TECNICOS |
| mediador\_agrup\_producto\_TRANSPORTES |
| mediador\_antiguedad |
| mediador\_numero\_polizas\_count |
| mediador\_numero\_polizas\_vigor\_count |
| mediador\_numero\_siniestros\_count |
| mediador\_numero\_siniestros\_declarados\_count |
| mediador\_numero\_siniestros\_fraude\_count |
| mediador\_numero\_siniestros\_pagados\_count |
| mediador\_numero\_polizas\_hogar |
| mediador\_numero\_polizas\_vigor\_hogar |
| mediador\_numero\_siniestros\_hogar |
| mediador\_numero\_siniestros\_declarados\_hogar |
| mediador\_numero\_siniestros\_fraude\_hogar |
| mediador\_numero\_siniestros\_pagados\_hogar |
| mediador\_poliza\_vigor\_total |
| mediador\_siniestros\_declarado\_total |
| mediador\_siniestros\_fraude\_total |
| mediador\_siniestros\_pagados\_total |
| mediador\_numero\_siniestros/poliza |
| mediador\_numero\_siniestros\_declarados/poliza |
| mediador\_numero\_siniestros\_fraude/poliza |
| mediador\_numero\_siniestros\_pagados/poliza |
| mediador\_poliza\_vigor\_total\_hogar |
| mediador\_siniestros\_declarado\_total\_hogar |
| mediador\_siniestros\_fraude\_total\_hogar |
| mediador\_siniestros\_pagados\_total\_hogar |
| mediador\_numero\_siniestros/poliza\_hogar |
| mediador\_numero\_siniestros\_declarados/poliza\_hogar |
| mediador\_numero\_siniestros\_fraude/poliza\_hogar |
| mediador\_numero\_siniestros\_pagados/poliza\_hogar |
| mediador\_numero\_polizas\_weight |
| mediador\_numero\_polizas\_vigor\_weight |
| mediador\_numero\_siniestros\_weight |
| mediador\_numero\_siniestros\_declarados\_weight |
| mediador\_numero\_siniestros\_fraude\_weight |
| mediador\_numero\_siniestros\_pagados\_weight |
| mediador\_numero\_siniestros\_hogar\_weight |
| mediador\_numero\_siniestros\_declarados\_hogar\_weight |
| mediador\_numero\_siniestros\_fraude\_hogar\_weight |
| mediador\_numero\_siniestros\_pagados\_hogar\_weight |
| mediador\_numero\_polizashogar/total |
| mediador\_numero\_polizas\_vigorhogar/total |
| mediador\_numero\_siniestroshogar/total |
| mediador\_numero\_siniestros\_declaradoshogar/total |
| mediador\_numero\_siniestros\_fraudehogar/total |
| mediador\_numero\_siniestros\_pagadoshogar/total |
| mediador\_numero\_polizas\_count\_mad\_outlier |
| mediador\_numero\_polizas\_hogar\_mad\_outlier |
| mediador\_numero\_polizas\_vigor\_count\_mad\_outlier |
| mediador\_numero\_polizas\_vigor\_hogar\_mad\_outlier |
| mediador\_numero\_siniestros\_count\_mad\_outlier |
| mediador\_numero\_siniestros\_hogar\_mad\_outlier |
| mediador\_numero\_siniestros\_declarados\_count\_mad\_outlier |
| mediador\_numero\_siniestros\_declarados\_hogar\_mad\_outlier |
| mediador\_numero\_siniestros\_fraude\_count\_mad\_outlier |
| mediador\_numero\_siniestros\_fraude\_hogar\_mad\_outlier |
| mediador\_numero\_siniestros\_pagados\_count\_mad\_outlier |
| mediador\_numero\_siniestros\_pagados\_hogar\_mad\_outlier |
| mediador\_cod\_count\_blacklist\_mad\_outlier |
| mediador\_nif\_count\_blacklist\_mad\_outlier |
| mediador\_numero\_polizas\_count\_primer\_recibo |
| mediador\_numero\_polizas\_vigor\_count\_primer\_recibo |
| mediador\_numero\_siniestros\_count\_primer\_recibo |
| mediador\_numero\_siniestros\_declarados\_count\_primer\_recibo |
| mediador\_numero\_siniestros\_fraude\_count\_primer\_recibo |
| mediador\_numero\_siniestros\_pagados\_count\_primer\_recibo |
| mediador\_numero\_polizas\_hogar\_primer\_recibo |
| mediador\_numero\_polizas\_vigor\_hogar\_primer\_recibo |
| mediador\_numero\_siniestros\_hogar\_primer\_recibo |
| mediador\_numero\_siniestros\_declarados\_hogar\_primer\_recibo |
| mediador\_numero\_siniestros\_fraude\_hogar\_primer\_recibo |
| mediador\_numero\_siniestros\_pagados\_hogar\_primer\_recibo |
| mediador\_poliza\_vigor\_total\_primer\_recibo |
| mediador\_siniestros\_declarado\_total\_primer\_recibo |
| mediador\_siniestros\_fraude\_total\_primer\_recibo |
| mediador\_siniestros\_pagados\_total\_primer\_recibo |
| mediador\_numero\_siniestros/poliza\_primer\_recibo |
| mediador\_numero\_siniestros\_declarados/poliza\_primer\_recibo |
| mediador\_numero\_siniestros\_fraude/poliza\_primer\_recibo |
| mediador\_numero\_siniestros\_pagados/poliza\_primer\_recibo |
| mediador\_poliza\_vigor\_total\_hogar\_primer\_recibo |
| mediador\_siniestros\_declarado\_total\_hogar\_primer\_recibo |
| mediador\_siniestros\_fraude\_total\_hogar\_primer\_recibo |
| mediador\_siniestros\_pagados\_total\_hogar\_primer\_recibo |
| mediador\_numero\_siniestros/poliza\_hogar\_primer\_recibo |
| mediador\_numero\_siniestros\_declarados/poliza\_hogar\_primer\_recibo |
| mediador\_numero\_siniestros\_fraude/poliza\_hogar\_primer\_recibo |
| mediador\_numero\_siniestros\_pagados/poliza\_hogar\_primer\_recibo |
| mediador\_numero\_polizashogar/total\_primer\_recibo |
| mediador\_numero\_polizas\_vigorhogar/total\_primer\_recibo |
| mediador\_numero\_siniestroshogar/total\_primer\_recibo |
| mediador\_numero\_siniestros\_declaradoshogar/total\_primer\_recibo |
| mediador\_numero\_siniestros\_fraudehogar/total\_primer\_recibo |
| mediador\_numero\_siniestros\_pagadoshogar/total\_primer\_recibo |
| mediador\_numero\_polizas\_count\_sucesivos\_recibo |
| mediador\_numero\_polizas\_vigor\_count\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_count\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_declarados\_count\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_fraude\_count\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_pagados\_count\_sucesivos\_recibo |
| mediador\_numero\_polizas\_hogar\_sucesivos\_recibo |
| mediador\_numero\_polizas\_vigor\_hogar\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_hogar\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_declarados\_hogar\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_fraude\_hogar\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_pagados\_hogar\_sucesivos\_recibo |
| mediador\_poliza\_vigor\_total\_sucesivos\_recibo |
| mediador\_siniestros\_declarado\_total\_sucesivos\_recibo |
| mediador\_siniestros\_fraude\_total\_sucesivos\_recibo |
| mediador\_siniestros\_pagados\_total\_sucesivos\_recibo |
| mediador\_numero\_siniestros/poliza\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_declarados/poliza\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_fraude/poliza\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_pagados/poliza\_sucesivos\_recibo |
| mediador\_poliza\_vigor\_total\_hogar\_sucesivos\_recibo |
| mediador\_siniestros\_declarado\_total\_hogar\_sucesivos\_recibo |
| mediador\_siniestros\_fraude\_total\_hogar\_sucesivos\_recibo |
| mediador\_siniestros\_pagados\_total\_hogar\_sucesivos\_recibo |
| mediador\_numero\_siniestros/poliza\_hogar\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_declarados/poliza\_hogar\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_fraude/poliza\_hogar\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_pagados/poliza\_hogar\_sucesivos\_recibo |
| mediador\_numero\_polizashogar/total\_sucesivos\_recibo |
| mediador\_numero\_polizas\_vigorhogar/total\_sucesivos\_recibo |
| mediador\_numero\_siniestroshogar/total\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_declaradoshogar/total\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_fraudehogar/total\_sucesivos\_recibo |
| mediador\_numero\_siniestros\_pagadoshogar/total\_sucesivos\_recibo |
| D\_poliza\_desc\_estructura\_BANC SABADELL SEGUROS GENERALES | Policy Structure |
| D\_poliza\_desc\_estructura\_INDIVIDUAL COLECTIVOS | Policy Structure |
| D\_poliza\_desc\_estructura\_INDIVIDUAL DEUTSCHE BANK | Policy Structure |
| D\_poliza\_desc\_estructura\_INDIVIDUAL GRANDES DISTRIBUIDORES | Policy Structure |
| D\_poliza\_desc\_estructura\_INDIVIDUAL MEDIACION | Policy Structure |
| D\_poliza\_desc\_estructura\_PYMES MEDIACION | Policy Structure |
| D\_poliza\_desc\_estructura\_SMALL BUSINESS COLECTIVOS | Policy Structure |
| D\_poliza\_desc\_estructura\_SMALL BUSINESS MEDIACION | Policy Structure |
| D\_poliza\_canal\_Colectivo | Policy channel |
| D\_poliza\_canal\_Deutsche | Policy channel |
| D\_poliza\_canal\_Grandes\_Dist | Policy channel |
| D\_poliza\_canal\_Mediador | Policy channel |
| D\_poliza\_duracion\_R | Policy term |
| D\_poliza\_credit\_scoring\_1 | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_2 | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_3 | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_4 | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_5 | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_6 | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_A | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_B | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_C | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_D | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_E | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_F | Policy Credit Scoring |
| D\_poliza\_credit\_scoring\_No Informado | Policy Credit Scoring |
| d\_poliza\_ultimo\_movimiento\_1 | Policy last movement |
| d\_poliza\_ultimo\_movimiento\_2 | Policy last movement |
| d\_poliza\_ultimo\_movimiento\_3 | Policy last movement |
| d\_poliza\_ultimo\_movimiento\_4 | Policy last movement |
| d\_poliza\_ultimo\_movimiento\_5 | Policy last movement |
| d\_poliza\_ultimo\_movimiento\_8 | Policy last movement |
| d\_poliza\_ultimo\_movimiento\_A | Policy last movement |
| d\_poliza\_ultimo\_movimiento\_S | Policy last movement |

## LOSS ADJUSTER

Loss adjuster information is other of the principal legs of fraud detection. It contains information about the process of the investigation task but also we gather information about the loss adjuster, and calculate some statistics of his past performance.

It is remarkably important to have precise and standardized information. This is an important challenge we have encounter. As explained in Integration Problems section, we have could not get through the main database several aspects of the investigation process.

|  |  |
| --- | --- |
| **Names** | **Description** |
| id\_fiscal | Id. Fiscal |
| id\_poliza | Número de póliza |
| version\_poliza | Número de versión de la póliza |
| id\_siniestro | Código de Siniestro |
| peritaje\_codigo | Código de Peritaje |
| fecha\_encargo\_peritaje | Fecha de Encargo de Peritaje |
| peritaje\_numero\_informe | Número de informe |
| peritaje\_fecha\_informe | Fecha de Encargo de Peritaje |
| fecha\_informe\_encargo\_extendida | Si la fecha del ultimo informe y del encargo es muy amplia |
| peritaje\_siniestro\_causa | Causa objetiva del siniestro |
| peritaje\_nombre | Perito |
| peritaje\_informe\_previo\_origen\_siniestro | Explicación causa origen siniestro |
| peritaje\_informe\_previo\_reserva\_estimada | Reserva Estimada Global |
| peritaje\_aide\_nombre\_reparador | Nombre del reparador |
| peritaje\_pregunta\_1 | Situación riesgo coincide c/ declarada póliza |
| peritaje\_pregunta\_2 | Actividad coincide c/ declarada póliza |
| peritaje\_pregunta\_3 | Riesgo Agravado |
| peritaje\_pregunta\_4 | Siniestro consorciable |
| peritaje\_pregunta\_5 | Intervienen siniestros especiales |
| peritaje\_pregunta\_6a | Existen terceros |
| peritaje\_pregunta\_6b | Perjudicado o causante (nombre, domicilio, cía.seg, pól, stro) |
| peritaje\_pregunta\_7 | Interviene Zurich Asistencia |
| peritaje\_pregunta\_8a | Siniestro con cobertura en póliza |
| peritaje\_pregunta\_8b | Motivo, ref,cond,nºapart. En que apoya decisión |
| peritaje\_pregunta\_9 | Capital 1 coincide con la realidad |
| peritaje\_pregunta\_10 | Capital 2 coincide con la realidad |
| peritaje\_pregunta\_11 | Capital 3 coinicide con la realidad |
| peritaje\_pregunta\_12 | Capital 4 coincide con la realidad |
| peritaje\_pregunta\_13 | Capital 5 coincide con la realidad |
| peritaje\_pregunta\_14 | Capital 6 coinicide con la realidad |
| peritaje\_pregunta\_15 | Capital 7 coincide con la realidad |
| peritaje\_pregunta\_16 | Capital 8 coincide con la realidad |
| peritaje\_pregunta\_17 | Medidas protección del riesgo activas |
| peritaje\_pregunta\_18 | Posible incidencia |
| peritaje\_pregunta\_19 | Concurrencia de Seguros |
| peritaje\_pregunta\_20 | Medidas protección distintas declaradas en póliza |
| peritaje\_posible\_fraude | Si las respuestas consideradas de posible fraude no indican fraude |
| peritaje\_negativo | = 1 si el peritaje es negativo |
| peritaje\_negativos\_perito | Cuenta en función de la regla peritaje\_negativo sobre el total de casos que fueron encargados |
| peritaje\_obs\_documentacion | Observaciones |
| peritaje\_obs\_fotos | Fotos |
| peritaje\_transferencia\_IBAN | Núm Cuenta |
| peritaje\_transferencia\_nif | Id. Fiscal |
| peritaje\_indem\_informe\_previo | Reserva inicial Estimada por el perito |
| peritaje\_indem\_informe\_definitivo | Indemnización Final Estimada por el perito |
| peritaje\_coberturas\_indemnizar\_previo | Garantías a Indemnizar inicial Estimada por el perito |
| peritaje\_coberturas\_indemnizar\_definitivo | Garantías a Indemnizar final Estimada por el perito |
| peritaje\_indem\_final\_inicial | Indemnización Final - Indemnización Inicial |
| peritaje\_garantia\_final\_inicial | Cantidad de Garantías Final - Cantidad de Garantías Inicial |

1- As always we start loading the file, transforming it to a Dataframe and mapping the key variables:

# Load File  
file = ReadCsv.load\_csv(name\_file)  
  
# Transform file to DF  
file\_df = DfUtils.processing\_file(file, delimiter=',')  
  
# Map Important Variables  
file\_df['id\_siniestro'] = file\_df['id\_siniestro'].map(int)  
file\_df['peritaje\_transferencia\_IBAN'] = file\_df['peritaje\_transferencia\_IBAN'].str.replace(' ', '')  
file\_df['peritaje\_transferencia\_IBAN'] = file\_df['peritaje\_transferencia\_IBAN'].fillna(0)  
file\_df['peritaje\_transferencia\_IBAN'] = file\_df['peritaje\_transferencia\_IBAN'].map(int)  
file\_df["peritaje\_transferencia\_nif"] = file\_df["peritaje\_transferencia\_nif"].map(str)

2- We reformat the existing date variables:

# VARIABLES DE FECHA  
fecha\_variables = ['fecha\_encargo\_peritaje', 'peritaje\_fecha\_informe']  
  
**for** i **in** fecha\_variables:  
 file\_df[i] = pd.to\_datetime(file\_df[i], format='%Y-%m-%d', errors='coerce')  
 file\_df.loc[file\_df[i] > STRING.Parameters.end\_date] = np.NaN

3- We have the information organized as follows. We have a sinister with several loss-adjuster visit codes. Each sinister has the total visits a loss-adjuster realized. Therefore, it will imply an extra work to aggregate this information. Finally we should have a unique sinister row.

file\_df = file\_df.dropna(subset=['id\_siniestro'])  
file\_df = file\_df.sort\_values(by=['id\_siniestro', 'peritaje\_codigo'],  
 ascending=[**True**, **True**])

4- Sinister has two types of expert reports. One is the previous report which is a basic version of the second type. The second type is the definitive report, which is basically the last visit. This last visit collects information from the previous one. Its information is richer. It is important to note this distinction from now on. The first variable generated is relative to the definitive report. Previous report always exist in this bottle, because it is associated with a visit code. And always, the first visit is a previous visit. But not always exists a definitive report. It is important to to recognize this cases, because it is probable that we are getting less information than the other cases.

file\_df['peritaje\_no\_informe\_definitivo'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['peritaje\_fecha\_informe'].isnull(), 'peritaje\_no\_informe\_definitivo'] = 1  
file\_df['peritaje\_no\_informe\_definitivo'] = file\_df.groupby(  
 ['id\_siniestro'])['peritaje\_no\_informe\_definitivo'].apply(**lambda** x: x.cumsum())  
  
file\_df.loc[file\_df['peritaje\_no\_informe\_definitivo'] > 0, 'peritaje\_no\_informe\_definitivo'] = 1

As we are going to notice, we are using the cumsum function in most variables. That is because we want to get the aggregation by sinister. Therefore, at the end, we are just going to keep the last row.

5- As explained before, we are using a cumsum function. Therefore, at the end, we need to weight by how many visits we had in each sinister. That is because we are using several measures related also to average terms.

Here we construct a weight variable called ‘peritaje\_count’. This kind of variable will be common in another atomized bottles.

file\_df['peritaje\_pondera'] = pd.Series(1, index=file\_df.index)  
file\_df['peritaje\_count'] = file\_df.groupby(  
 ['id\_siniestro'])['peritaje\_pondera'].apply(**lambda** x: x.cumsum())

6- Now we want to keep the first visit and the last visit. With both, we can calculate how long taked the total visits. We have several dates associated to each sinister. So we have to make some tricks. Firs of all, if we need to take the first date associated to the column ‘fecha\_encargo\_peritaje’, which are the previous reports. From them, we have to take the first value of them. That is the first previous report.

# First Date: We create a table that only keep the sinister and the first date. Then we remarge keeping the  
# first value of each sinister  
fecha\_primer\_df = file\_df[['id\_siniestro', 'fecha\_encargo\_peritaje']]  
fecha\_primer\_df = fecha\_primer\_df.drop\_duplicates(subset=['id\_siniestro'], keep ='first')  
fecha\_primer\_df = fecha\_primer\_df.rename(columns={'fecha\_encargo\_peritaje': 'fecha\_primer\_peritaje'})  
file\_df = pd.merge(file\_df, fecha\_primer\_df, how='left', on='id\_siniestro')  
**del** fecha\_primer\_df

Then we have to make the inverse process but with the variable ‘peritaje\_fecha’ which is related tho the definitive report.

# Last Date: We create an additional table with peritaje\_fecha\_informe and we keep only the last value for  
# each sinister.  
fecha\_ultimo\_df = file\_df[['id\_siniestro', 'peritaje\_fecha\_informe']]  
fecha\_ultimo\_df = fecha\_ultimo\_df.drop\_duplicates(subset=['id\_siniestro'], keep='last')  
fecha\_ultimo\_df = fecha\_ultimo\_df.rename(columns={'peritaje\_fecha\_informe': 'fecha\_ultimo\_informe'})  
file\_df = pd.merge(file\_df, fecha\_ultimo\_df, how='left', on='id\_siniestro')  
**del** fecha\_ultimo\_df

Finally we merge these auxiliary tables in one, and take the difference between them.

# TIEMPO ENTRE PERITAJES: fecha\_ultimo\_informe - fecha\_primer\_peritaje  
file\_df['peritaje\_duracion'] = pd.Series(file\_df['fecha\_ultimo\_informe'] - file\_df['fecha\_primer\_peritaje'],  
 index=file\_df.index).dt.days

7- The average of each visit will be simple the total duration divided by total number of visits.

# DURACION PROMEDIO PERITAJE: We calculate the duration time average by number of visits in each sinister  
file\_df['peritaje\_duracion\_promedio'] = pd.Series(file\_df['peritaje\_duracion'] / file\_df['peritaje\_count'],  
 index=file\_df.index)

8- We reconvert the format of FECHA\_INFORME\_EXTENDIDA:

# FECHA\_INFORME EXTENDIDA: We convert fecha\_informe\_encargo\_extendida into two values S = 1, N = 0  
file\_df.loc[file\_df['fecha\_informe\_encargo\_extendida'] == 'S', 'fecha\_informe\_encargo\_extendida'] = 1  
file\_df.loc[file\_df['fecha\_informe\_encargo\_extendida'] == 'N', 'fecha\_informe\_encargo\_extendida'] = 0  
file\_df['fecha\_informe\_encargo\_extendida'] = file\_df.groupby(  
 ['id\_siniestro'])['fecha\_informe\_encargo\_extendida'].apply(**lambda** x: x.cumsum())

9 – CATEGORICAL VARIABLES: Here we apply transformations as always. But we have several hide subprocesses in each iteration. We are going to take some time to explain it.

We iterate through the next variables:

var\_dummies = ['peritaje\_pregunta\_1', 'peritaje\_pregunta\_2',  
 'peritaje\_pregunta\_3', 'peritaje\_pregunta\_4',  
 'peritaje\_pregunta\_5', 'peritaje\_pregunta\_6a',  
 'peritaje\_pregunta\_7', 'peritaje\_pregunta\_8a',  
 'peritaje\_pregunta\_9', 'peritaje\_pregunta\_10',  
 'peritaje\_pregunta\_11', 'peritaje\_pregunta\_12',  
 'peritaje\_pregunta\_13', 'peritaje\_pregunta\_14',  
 'peritaje\_pregunta\_15', 'peritaje\_pregunta\_16',  
 'peritaje\_pregunta\_17', 'peritaje\_pregunta\_18',  
 'peritaje\_pregunta\_19', 'peritaje\_pregunta\_20',  
 'peritaje\_negativo', 'peritaje\_posible\_fraude',  
 'peritaje\_negativos\_perito'  
 ]

For each variable we generate an auxliary table that contains the variable plus ‘id\_siniestro’.

dummies\_sum = file\_df[['id\_siniestro'] + [i]]

We are going to make some operations so we need to drop NaN values.

# We drop the nan values in each loss adjuster visit  
dummies\_sum = dummies\_sum.dropna(how='any', subset=[i])

## SINISTER

Sinister table not give us much information about the incident (the core is collected in Assistance) which is also an integration problem. Here we have briefly and partial sinister information like some dates, location and not much more.

|  |  |
| --- | --- |
| **Name** | **Description** |
| id\_fiscal | Id. Fiscal |
| id\_poliza | Número de póliza |
| version\_poliza | Número de versión de la póliza |
| id\_siniestro | Código de Siniestro |
| siniestro\_oficina\_apertura | Canal de Apertura del Siniestro |
| siniestro\_nombre\_usuario\_apertura | Canal de Apertura del Siniestro |
| siniestro\_situacion | Sit. |
| siniestro\_numeros\_anteriores | Núm. |
| siniestro\_descripcion | Descripción del Siniestro |
| siniestro\_lugar | Lugar |
| siniestro\_cp | CP |
| siniestro\_poblacion | Población |
| siniestro\_provincia | Provincia |
| siniestro\_indicador\_consorcio | Indicador de Consorcio |
| siniestro\_indicador\_denuncia | Indicador de Denuncia |
| siniestro\_indicador\_subrogacion | Indicador de Subrogación |
| siniestro\_indicador\_via\_judicial | Vía judicial |
| siniestro\_indicador\_intervencion\_policial | Intervención policial |
| siniestro\_indicador\_posible\_graciable | Posible snstro. Graciable |
| siniestro\_indicador\_concurrencia | Concurrencia |

1- As always we start loading the file, transforming it to a Dataframe and mapping the key variables:

# Load File  
file = ReadCsv.load\_csv(name\_file)  
  
# Transform file to DF  
file\_df = DfUtils.processing\_file(file)  
  
# Define Principal ID dtypes  
file\_df['id\_siniestro'] = file\_df['id\_siniestro'].map(int)

2- Second, we check the user who opened the sinister. We verify if it was a manual user and try to clean batch processes and migration sinister.

# USUARIO: If it is a manual user (if it is not AIDE, MIGRACION or BATCH = 1)  
file\_df['siniestro\_usuario\_manual'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[-file\_df['siniestro\_nombre\_usuario\_apertura'].isin(['USUARAIDE', 'MSD MIGRA', 'BATCH1',  
 'BATCH2', 'SGR']),  
 'siniestro\_usuario\_manual'] = 1

3- We also check if the person is a costumer, a Zurich employee, a center or a web service.

# USUARIO DE APERTURA: What kind of user is it. TCE, Zurich or Web Service.  
file\_df['siniestro\_usuario\_cliente'] = pd.Series(0, index=file\_df.index)  
file\_df['siniestro\_usuario\_Z'] = pd.Series(0, index=file\_df.index)  
file\_df['siniestro\_usuario\_WS'] = pd.Series(0, index=file\_df.index)  
  
file\_df.loc[file\_df["siniestro\_nombre\_usuario\_apertura"].str.startswith('TCE'),  
 'siniestro\_usuario\_cliente'] = 1  
  
file\_df.loc[file\_df["siniestro\_nombre\_usuario\_apertura"].str.startswith('Z'),  
 'siniestro\_usuario\_Z'] = 1  
  
file\_df.loc[file\_df["siniestro\_nombre\_usuario\_apertura"].str.startswith('WS'),  
 'siniestro\_usuario\_WS'] = 1

4- We identify each opening office with a dummy:

# OFICINA APERTURA: We get dummies for each OFICINA APERUTRA  
file\_df['siniestro\_oficina\_apertura'] = file\_df['siniestro\_oficina\_apertura'].fillna(  
 'No Informado')  
  
file\_df['siniestro\_oficina\_apertura'] = file\_df[  
 'siniestro\_oficina\_apertura'].str.upper()  
  
file\_df.loc[file\_df['siniestro\_oficina\_apertura'].str.startswith('ZURICH'),  
 'siniestro\_oficina\_apertura'] = 'ZURICH'  
  
file\_df.loc[file\_df['siniestro\_oficina\_apertura'].str.startswith('CTD LINEAS PERSONALES'),  
 'siniestro\_oficina\_apertura'] = 'CTD LINEAS PERSONALES'  
  
file\_df.loc[file\_df['siniestro\_oficina\_apertura'].str.contains('MIGRACION'),  
 'siniestro\_oficina\_apertura'] = 'MIGRACION'  
  
dummies = pd.get\_dummies(file\_df['siniestro\_oficina\_apertura'],  
 prefix='siniestro\_oficina', dummy\_na=**False**)  
  
file\_df = pd.concat([file\_df, dummies], axis=1)  
  
**del** file\_df['hist\_siniestro\_actual\_oficina\_apertura']  
**del** dummies

5- We create dummies for the sinister state:

# SITUACION  
dummies = pd.get\_dummies(file\_df['siniestro\_situacion'], prefix='siniestro\_situacion')  
file\_df = pd.concat([file\_df, dummies], axis=1)  
**del** dummies  
**del** file\_df['siniestro\_situacion']

6- We delete useless variables:

# DELETE VARIABLES  
delete\_var = ['id\_fiscal', 'id\_poliza', 'version\_poliza', 'siniestro\_descripcion', 'siniestro\_factor\_culpa',  
 'siniestro\_hora\_ocurrencia', 'siniestro\_lugar', 'siniestro\_cp', 'siniestro\_poblacion',  
 'siniestro\_provincia', 'siniestro\_indicador\_consorcio', 'siniestro\_indicador\_denuncia',  
 'siniestro\_indicador\_subrogacion', 'siniestro\_indicador\_via\_judicial',  
 'siniestro\_indicador\_intervencion\_policial', 'siniestro\_indicador\_posible\_graciable',  
 'siniestro\_indicador\_concurrencia', 'siniestro\_posible\_fraude', 'siniestro\_numero\_perjudicados',  
 'siniestro\_datos\_testigos']  
  
**for** i **in** delete\_var:  
 **del** file\_df[i]  
  
**return** file\_df

The output is:

|  |  |
| --- | --- |
| **Name** | **Description** |
| id\_siniestro | Sinister Id |
| siniestro\_numeros\_anteriores | Number of past sinister |
| siniestro\_usuario\_manual | Manual User |
| siniestro\_usuario\_cliente | Costumer user |
| siniestro\_usuario\_Z | Zurich user |
| siniestro\_usuario\_WS | Web Services user |
| siniestro\_oficina\_CSS | Office type |
| siniestro\_oficina\_CT RC | Office type |
| siniestro\_oficina\_CTA | Office type |
| siniestro\_oficina\_CTD | Office type |
| siniestro\_oficina\_MIGRACION | Office type |
| siniestro\_oficina\_NO INFORMADO | Office type |
| siniestro\_oficina\_ZURICH | Office type |
| siniestro\_situacion\_T | Sinister state |

## INTERMEDIARY

The information about intermediaries is relevant because he is the person who has the business knowledge and generally a closer relationship with the customer.

Gathering this data, we try to identify uniquely the whole spectrum of intermediaries, and calculate several statistics for each of them. With this information we can capture the behavior of the subject analyzed that can give us insights about a possible fraudulent claim.

|  |  |
| --- | --- |
| **Name** | **Description** |
| mediador\_cod\_intermediario | Código del Intermediario Productor Priimero |
| mediador\_denominacion\_intermediario | Denominación del intermediario Productor Primero |
| mediador\_nif\_intermediario | NIF/ CIF del intermediario |
| mediador\_clase\_intermediario | Clase de Intermediario de mediador |
| mediador\_fecha\_alta | Fecha de Alta del Mediador |
| mediador\_estado | Estado del mediador |
| id\_agrup\_producto | Agrupación de producto |
| mediador\_numero\_polizas | Número de pólizas asociadas al mediador |
| mediador\_numero\_polizas\_vigor | Número de pólizas en vigor del mediador |
| mediador\_numero\_siniestros | Número de Siniestros asociados a las pólizas que están con el mediador |
| mediador\_numero\_siniestros\_declarados | Histórico de siniestros declarados del mediador |
| mediador\_numero\_siniestros\_fraude | Histórico de siniestros fraudulentos del mediador |
| mediador\_numero\_siniestros\_pagados | Histórico de siniestros del mediador que han sido pagados |

1- As always we start loading the file, transforming it to a Dataframe and mapping the key variables:

# Load File  
file = ReadCsv.load\_csv(name\_file)  
  
# Transform file to DF  
file\_df = DfUtils.processing\_file(file)  
  
# Define Principal ID dtypes  
file\_df['mediador\_cod\_intermediario'] = file\_df['mediador\_cod\_intermediario'].map(int)  
file\_df['mediador\_nif\_intermediario'] = file\_df['mediador\_nif\_intermediario'].map(str)

2- We have to set the variables at intermediary level. Therefore, we have to do some grouping variable process. The raw data is ordered by intermediary-product. Therefore, we have to weight using intermediary’s products:

# COUNT PRODUCTOS QUE POSEE  
file\_df['pondera\_producto'] = pd.Series(1, index=file\_df.index)  
file\_df['mediador\_producto\_count'] = file\_df.groupby(  
 ['mediador\_cod\_intermediario'])['pondera\_producto'].apply(**lambda** x: x.cumsum())

3- First of all, we are going to retrieve the blacklist. Using the intermediary blacklist we compare the intermediary codes. We count how many times it appears.

# ID BLACKLIST: We compare with Tomador Blacklist if the customer appears in a Fraud Sinister  
file\_blacklist\_intermediario = ReadCsv.load\_csv(STRING.processed\_blacklist\_intermediario)  
df\_bl\_intermediario = DfUtils.processing\_file(file\_blacklist\_intermediario, delimiter=';')  
df\_bl\_intermediario = pd.DataFrame(  
 df\_bl\_intermediario.groupby('NIF').size().rename('mediador\_cod\_count\_blacklist').reset\_index())  
df\_bl\_intermediario['NIF'] = pd.to\_numeric(df\_bl\_intermediario['NIF'], errors='coerce')  
df\_bl\_intermediario = df\_bl\_intermediario.dropna(subset=['NIF'], how='any', axis=0)  
df\_bl\_intermediario['NIF'] = df\_bl\_intermediario['NIF'].map(int)  
file\_df = pd.merge(file\_df, df\_bl\_intermediario, how='left', left\_on='mediador\_cod\_intermediario',  
 right\_on='NIF')  
file\_df['mediador\_cod\_blacklist'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['NIF'].notnull(), 'mediador\_cod\_blacklist'] = 1  
**del** file\_df['NIF']  
**del** df\_bl\_intermediario

4- But also, we compare NIF, because we can be losing intermediaries that have changed their code. Or also, intermediary that are fraudulants but not as intermediaries.

# Si aparece como NIF  
file\_blacklist\_intermediario = ReadCsv.load\_csv(STRING.processed\_blacklist\_intermediario)  
df\_bl\_intermediario = DfUtils.processing\_file(file\_blacklist\_intermediario, delimiter=';')  
df\_bl\_intermediario = pd.DataFrame(  
 df\_bl\_intermediario.groupby('NIF').size().rename('mediador\_nif\_count\_blacklist').reset\_index())  
df\_bl\_intermediario['NIF'] = df\_bl\_intermediario['NIF'].map(str)  
file\_df = pd.merge(file\_df, df\_bl\_intermediario, how='left', left\_on='mediador\_nif\_intermediario',  
 right\_on='NIF')  
file\_df['mediador\_nif\_blacklist'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['NIF'].notnull(), 'mediador\_nif\_blacklist'] = 1  
**del** file\_df['NIF']  
**del** df\_bl\_intermediario

5- We check the intermediary status and change the values using a dictionary:

# ESTADO DEL MEDIADOR: We map the categoric values that correspond to mediador\_estado  
file\_df['mediador\_estado'] = file\_df['mediador\_estado'].map(int)  
estado = {1: 'Activo', 2: 'Activo', 3: 'Inactivo', 4: 'Pendiente', 5: 'Tramite'}  
  
**for** key, value **in** estado.items():  
 file\_df.loc[file\_df['mediador\_estado'] == key, 'mediador\_estado'] = value

6- We create dummy varibles of categorical variables. We separate between variables that need a cumsum and variables that not. Also we need an auxiliary tab to retrieve the name of products code.

# DUMMIES: This dummies do not require a cumsum because the are a unique value  
dummy\_var = ['mediador\_clase\_intermediario', 'mediador\_estado']  
**for** i **in** dummy\_var:  
 name = str(i)  
 dummy = pd.get\_dummies(file\_df[i], prefix=name)  
 file\_df = pd.concat([file\_df, dummy], axis=1)  
 **del** dummy  
 **del** file\_df[i]  
  
# AGRUPACION DE PRODUCTO: It needs a cumsum  
file\_agrupacion = ReadCsv.load\_csv(STRING.auxiliar\_mediador\_producto)  
df\_agrupacion = DfUtils.processing\_file(file\_agrupacion, delimiter=';')  
df\_agrupacion['Code'] = df\_agrupacion['Code'].map(str)  
file\_df['id\_agrup\_producto'] = file\_df['id\_agrup\_producto'].map(str)  
file\_df = pd.merge(file\_df, df\_agrupacion, how='left', left\_on='id\_agrup\_producto', right\_on='Code')  
dummy = pd.get\_dummies(file\_df['Description'], prefix='mediador\_agrup\_producto', dummy\_na=**True**)  
file\_df = pd.concat([file\_df, dummy], axis=1)  
dummy\_names = dummy.columns.values.tolist()  
**del** dummy  
**del** file\_df['id\_agrup\_producto']  
**del** file\_df['Code']  
**for** i **in** dummy\_names:  
 file\_df[i] = file\_df.groupby(  
 ['mediador\_cod\_intermediario'])[i].apply(**lambda** x: x.cumsum())

7- Using the start date and the parameter value associated with today, we get the time in the company.

# ANTIGUEDAD DE MEDIADOR  
file\_df['mediador\_fecha\_alta'] = pd.to\_datetime(  
 file\_df['mediador\_fecha\_alta'], format='%Y%m%d', errors='coerce')  
today = datetime.datetime.today().strftime("%Y-%m-%d")  
today = datetime.datetime.strptime(today, "%Y-%m-%d")  
file\_df['mediador\_antiguedad'] = pd.Series(today - file\_df['mediador\_fecha\_alta'], index=file\_df.index).dt.days

8- Now, we start with the statistics associated to the intermediary.

a) First we do a simple cumsum for each variable:

# FLOAT VARIABLES: These need to be cumsum  
mediador\_nan = file\_df[pd.isnull(file\_df['mediador\_numero\_polizas'])]  
file\_df = file\_df.dropna(subset=['mediador\_numero\_polizas'], axis=0)  
  
float\_variables = ['mediador\_numero\_polizas', 'mediador\_numero\_polizas\_vigor', 'mediador\_numero\_siniestros',  
 'mediador\_numero\_siniestros\_fraude',  
 'mediador\_numero\_siniestros\_pagados']  
  
**for** i **in** float\_variables:  
 name\_count = str(i) + '\_count'  
 file\_df[i] = file\_df[i].map(int)  
 file\_df[name\_count] = file\_df.groupby(  
 ['mediador\_cod\_intermediario'])[i].apply(**lambda** x: x.cumsum())

As you can see, we delete rows with Null policy values.

b) We do the same, but now just for HOGAR products. For that we have to create an auxiliary dataframe that just contains HOGAR segmentation.

# COUNT BY HOGAR: We now create an additional table to get the statistics for PROPERTY and then remarge  
file\_df\_hogar = file\_df[file\_df['Description'] == 'HOGAR']  
keep\_hogar = ['mediador\_cod\_intermediario'] + float\_variables  
file\_df\_hogar = file\_df\_hogar[keep\_hogar]  
**for** i, s **in** enumerate(keep\_hogar):  
 **if** i != 0: # so it not modify mediador\_cod\_intermediario  
 keep\_hogar[i] = str(s)+'\_hogar'  
file\_df\_hogar.columns = keep\_hogar  
  
file\_df = pd.merge(file\_df, file\_df\_hogar, how='left', on='mediador\_cod\_intermediario')

# KEEP THE LAST VALUE OF THE CUMSUM:  
file\_df = file\_df.drop\_duplicates(subset=['mediador\_cod\_intermediario'], keep='last')

c) Now, we start to interrelate policy variables with policy variables, sinister variables with sinister variables and policy variables with sinister variables. We call this SET BY ROW statistics. And it is comprended by GLOBAL statistics (associated to the whole products) and HOGAR statistics (associated to HOGAR products).

# 1) STATISTICS SAME SET BY ROW: Here we compare the same set (GLOBAL or HOGAR) respecto to their  
# respectives columns  
poliza\_var = ['mediador\_numero\_polizas', 'mediador\_numero\_polizas\_vigor']  
siniestro\_var = ['mediador\_numero\_siniestros',  
 'mediador\_numero\_siniestros\_fraude',  
 'mediador\_numero\_siniestros\_pagados']  
  
# a) Global: Need to use '\_count'  
  
# polizas\_vigor / polizas\_total  
file\_df['mediador\_poliza\_vigor\_total'] = pd.Series(file\_df['mediador\_numero\_polizas\_vigor\_count'] /  
 file\_df['mediador\_numero\_polizas\_count'], index=file\_df.index)  
  
# siniestros / siniestros total  
file\_df['mediador\_siniestros\_fraude\_total'] = pd.Series(  
 file\_df['mediador\_numero\_siniestros\_fraude\_count'] /  
 file\_df['mediador\_numero\_siniestros\_count'],  
 index=file\_df.index)  
  
file\_df['mediador\_siniestros\_pagados\_total'] = pd.Series(  
 file\_df['mediador\_numero\_siniestros\_pagados\_count'] /  
 file\_df['mediador\_numero\_siniestros\_count'],  
 index=file\_df.index)  
  
# siniestros / poliza  
**for** sin\_str **in** siniestro\_var:  
 name = sin\_str + '/poliza'  
 file\_df[name] = pd.Series(file\_df[sin\_str + '\_count'] / file\_df['mediador\_numero\_polizas\_count'],  
 index= file\_df.index)  
  
# b) Hogar: Need to use '\_hogar'  
  
# polizas\_vigor / polizas\_total  
file\_df['mediador\_poliza\_vigor\_total\_hogar'] = pd.Series(file\_df['mediador\_numero\_polizas\_vigor\_hogar'] /  
 file\_df['mediador\_numero\_polizas\_hogar'],  
 index=file\_df.index)  
  
# siniestros / siniestros total  
file\_df['mediador\_siniestros\_fraude\_total\_hogar'] = pd.Series(  
 file\_df['mediador\_numero\_siniestros\_fraude\_hogar'] /  
 file\_df['mediador\_numero\_siniestros\_hogar'],  
 index=file\_df.index)  
  
file\_df['mediador\_siniestros\_pagados\_total\_hogar'] = pd.Series(  
 file\_df['mediador\_numero\_siniestros\_pagados\_hogar'] /  
 file\_df['mediador\_numero\_siniestros\_hogar'],  
 index=file\_df.index)  
  
# siniestros / poliza  
**for** sin\_str **in** siniestro\_var:  
 name = sin\_str + '/poliza\_hogar'  
 file\_df[name] = pd.Series(file\_df[sin\_str + '\_hogar'] / file\_df['mediador\_numero\_polizas\_hogar'],  
 index=file\_df.index)

d) Now we compare the relative weigth of each row respect to its column distribution. This is called statistics SAME SET BY COLUMN. Also with GLOBAL and HOGAR products.

# 2) STATISTICS SAME SET BY COLUMN: We compare the relative weight to the specific column.  
# a) Global:  
**for** i **in** poliza\_var:  
 var = i + '\_count'  
 name = i + '\_weight'  
 total = file\_df[var].sum()  
 file\_df[name] = file\_df[var] / total  
  
**for** i **in** siniestro\_var:  
 var = i + '\_count'  
 name = i + '\_weight'  
 total = file\_df[var].sum()  
 file\_df[name] = file\_df[var] / total  
  
# b) Hogar:  
**for** i **in** poliza\_var:  
 var = i + '\_hogar'  
 name = i + '\_weight'  
 total = file\_df[var].sum()  
 file\_df[name] = file\_df[var] / total  
  
**for** i **in** siniestro\_var:  
 var = i + '\_hogar'  
 name = var + '\_weight'  
 total = file\_df[var].sum()  
 file\_df[name] = file\_df[var] / total

e) Finally we compare the interrelation between HOGAR and total products. This is called DIFFERENT SETS BY COLUMN:

# STATISTICS DIFFERENT SETS BY COLUMN: Here we compare the relatives between HOGAR / GLOBAL  
**for** i **in** poliza\_var:  
 global\_var = i + '\_count'  
 hogar\_var = i + '\_hogar'  
 name = i + 'hogar/total'  
 file\_df[name] = pd.Series(file\_df[hogar\_var] / file\_df[global\_var], index=file\_df.index)  
  
**for** i **in** siniestro\_var:  
 global\_var = i + '\_count'  
 hogar\_var = i + '\_hogar'  
 name = i + 'hogar/total'  
 file\_df[name] = pd.Series(file\_df[hogar\_var] / file\_df[global\_var], index=file\_df.index)

9) We get outliers variables for the fraud sinister cases sum:

Outliers.outliers\_mad(file\_df, 'mediador\_cod\_count\_blacklist', just\_count\_zero=**True**)  
Outliers.outliers\_mad(file\_df, 'mediador\_nif\_count\_blacklist', just\_count\_zero=**True**)

10) We delete useless variables:

# DELETE VARIABLES  
del\_variables = float\_variables + ['mediador\_denominacion\_intermediario', 'mediador\_nif\_intermediario',  
 'mediador\_fecha\_alta', 'Description', 'pondera\_producto']  
  
**for** i **in** del\_variables:  
 **del** file\_df[i]

11) We fill the NaN variables with zero, because they are the result of processes where the intermediary actually has not value. Therefore, the zero is representative.

file\_df = file\_df.fillna(0)

The outcome is:

|  |  |
| --- | --- |
| **Name** | **Description** |
| mediador\_agrup\_producto\_ACCIDENTES | Intermediary product type |
| mediador\_agrup\_producto\_AUTOS | Intermediary product type |
| mediador\_agrup\_producto\_COMERCIOS | Intermediary product type |
| mediador\_agrup\_producto\_DECESOS | Intermediary product type |
| mediador\_agrup\_producto\_EMBARCACIONES | Intermediary product type |
| mediador\_agrup\_producto\_EPSV | Intermediary product type |
| mediador\_agrup\_producto\_FLOTAS | Intermediary product type |
| mediador\_agrup\_producto\_HOGAR | Intermediary product type |
| mediador\_agrup\_producto\_INDIVIDUAL AHORRO PERIODICAS | Intermediary product type |
| mediador\_agrup\_producto\_INDIVIDUAL AHORRO UNICAS | Intermediary product type |
| mediador\_agrup\_producto\_INDIVIDUAL PPA | Intermediary product type |
| mediador\_agrup\_producto\_INDIVIDUAL RIESGO PERIODICAS | Intermediary product type |
| mediador\_agrup\_producto\_INDIVIDUAL RIESGO UNICAS | Intermediary product type |
| mediador\_agrup\_producto\_INDIVIDUAL UNIT LINK PERIODICAS | Intermediary product type |
| mediador\_agrup\_producto\_INDIVIDUAL UNIT LINK UNICAS | Intermediary product type |
| mediador\_agrup\_producto\_INMUEBLES | Intermediary product type |
| mediador\_agrup\_producto\_OTROS | Intermediary product type |
| mediador\_agrup\_producto\_PATRIMONIALES | Intermediary product type |
| mediador\_agrup\_producto\_PLANES PENSIONES | Intermediary product type |
| mediador\_agrup\_producto\_PROT. PAGOS | Intermediary product type |
| mediador\_agrup\_producto\_RESP. CIVIL | Intermediary product type |
| mediador\_agrup\_producto\_TECNICOS | Intermediary product type |
| mediador\_agrup\_producto\_TRANSPORTES | Intermediary product type |
| mediador\_agrup\_producto\_nan | Intermediary product type |
| mediador\_antiguedad | Intermediary time |
| mediador\_clase\_intermediario\_AC | Intermediary class |
| mediador\_clase\_intermediario\_AE | Intermediary class |
| mediador\_clase\_intermediario\_AF | Intermediary class |
| mediador\_clase\_intermediario\_AV | Intermediary class |
| mediador\_clase\_intermediario\_BA | Intermediary class |
| mediador\_clase\_intermediario\_BC | Intermediary class |
| mediador\_clase\_intermediario\_BD | Intermediary class |
| mediador\_clase\_intermediario\_BM | Intermediary class |
| mediador\_clase\_intermediario\_CB | Intermediary class |
| mediador\_clase\_intermediario\_CC | Intermediary class |
| mediador\_clase\_intermediario\_CD | Intermediary class |
| mediador\_clase\_intermediario\_CE | Intermediary class |
| mediador\_clase\_intermediario\_CF | Intermediary class |
| mediador\_clase\_intermediario\_CO | Intermediary class |
| mediador\_clase\_intermediario\_CR | Intermediary class |
| mediador\_clase\_intermediario\_EC | Intermediary class |
| mediador\_clase\_intermediario\_EM | Intermediary class |
| mediador\_clase\_intermediario\_FR | Intermediary class |
| mediador\_clase\_intermediario\_I | Intermediary class |
| mediador\_clase\_intermediario\_LI | Intermediary class |
| mediador\_clase\_intermediario\_LP | Intermediary class |
| mediador\_clase\_intermediario\_OC | Intermediary class |
| mediador\_clase\_intermediario\_OE | Intermediary class |
| mediador\_clase\_intermediario\_OP | Intermediary class |
| mediador\_clase\_intermediario\_OS | Intermediary class |
| mediador\_clase\_intermediario\_OV | Intermediary class |
| mediador\_clase\_intermediario\_PL | Intermediary class |
| mediador\_clase\_intermediario\_PR | Intermediary class |
| mediador\_clase\_intermediario\_SU | Intermediary class |
| mediador\_clase\_intermediario\_TC | Intermediary class |
| mediador\_clase\_intermediario\_VD | Intermediary class |
| mediador\_cod\_blacklist | =1, If it appears in the Blacklist by id |
| mediador\_cod\_count\_blacklist | How many fraud sinister has by id |
| mediador\_cod\_intermediario | Intermediary Id |
| mediador\_estado\_Activo | Intermediary state |
| mediador\_estado\_Inactivo | Intermediary state |
| mediador\_estado\_Pendiente | Intermediary state |
| mediador\_estado\_Tramite | Intermediary state |
| mediador\_nif\_blacklist | Intermediary NIF |
| mediador\_nif\_count\_blacklist | How many fraud sinister has by NIF |
| mediador\_numero\_polizas\_count | Policy quantities |
| mediador\_numero\_polizas\_count\_mad\_outlier | Policy outlier |
| mediador\_numero\_polizas\_hogar | HOGAR policy quantities |
| mediador\_numero\_polizas\_hogar\_mad\_outlier | HOGAR policy outlier |
| mediador\_numero\_polizas\_vigor\_count | Current policy quantities |
| mediador\_numero\_polizas\_vigor\_count\_mad\_outlier | Outlier current policies |
| mediador\_numero\_polizas\_vigor\_hogar | HOGAR current policies |
| mediador\_numero\_polizas\_vigor\_hogar\_mad\_outlier | Outlier HOGAR current policies |
| mediador\_numero\_polizas\_vigor\_weight | Weigthed current policies |
| mediador\_numero\_polizas\_vigorhogar/total | Hogar current Policies / total Policies |
| mediador\_numero\_polizas\_weight | Weigthed policies |
| mediador\_numero\_polizashogar/total | Hogar policies / Total policies |
| mediador\_numero\_siniestros/poliza | Siniester / Policy |
| mediador\_numero\_siniestros/poliza\_hogar | Siniester / Hogar policy |
| mediador\_numero\_siniestros\_count | Sinister number |
| mediador\_numero\_siniestros\_count\_mad\_outlier | Outlier sinister number |
| mediador\_numero\_siniestros\_fraude/poliza | Fraud sinister / policy |
| mediador\_numero\_siniestros\_fraude/poliza\_hogar | Fraud sinister / HOGAR policy |
| mediador\_numero\_siniestros\_fraude\_count | Fraud sinister count |
| mediador\_numero\_siniestros\_fraude\_count\_mad\_outlier | Outlier fraud sinister |
| mediador\_numero\_siniestros\_fraude\_hogar | HOGAR fraud sinister |
| mediador\_numero\_siniestros\_fraude\_hogar\_mad\_outlier | Outlier fraud sinister |
| mediador\_numero\_siniestros\_fraude\_hogar\_weight | Weighted HOGAR fraud sinister |
| mediador\_numero\_siniestros\_fraude\_weight | Weighted fraud sinister |
| mediador\_numero\_siniestros\_fraudehogar/total | Fraud HOGAR / Total fraud |
| mediador\_numero\_siniestros\_hogar | HOGAR sinister |
| mediador\_numero\_siniestros\_hogar\_mad\_outlier | Outlier HOGAR sinister |
| mediador\_numero\_siniestros\_hogar\_weight | Weighted HOGAR sinister |
| mediador\_numero\_siniestros\_pagados/poliza | Paid sinister / Policy |
| mediador\_numero\_siniestros\_pagados/poliza\_hogar | Paid sinister / HOGAR policy |
| mediador\_numero\_siniestros\_pagados\_count | Paid sinister |
| mediador\_numero\_siniestros\_pagados\_count\_mad\_outlier | Outlier paid sinister |
| mediador\_numero\_siniestros\_pagados\_hogar | Hogar paid sinister |
| mediador\_numero\_siniestros\_pagados\_hogar\_mad\_outlier | Outlier HOGAR paid sinister |
| mediador\_numero\_siniestros\_pagados\_hogar\_weight | Weighted HOGAR paid sinister |
| mediador\_numero\_siniestros\_pagados\_weight | Weighted paid sinister |
| mediador\_numero\_siniestros\_pagadoshogar/total | HOGAR paid sinister / total sinister |
| mediador\_numero\_siniestros\_weight | Weighted sinister |
| mediador\_numero\_siniestroshogar/total | Hogar sinister / Total sinister |
| mediador\_poliza\_vigor\_total | Total current policies |
| mediador\_poliza\_vigor\_total\_hogar | Total current HOGAR policies |
| mediador\_producto\_count | Count products |
| mediador\_siniestros\_fraude\_total | Total fraud sinister |
| mediador\_siniestros\_fraude\_total\_hogar | Total hogar fraud sinister |
| mediador\_siniestros\_pagados\_total | Total paid sinister |
| mediador\_siniestros\_pagados\_total\_hogar | Intermediary paid HOGAR sinister |
| mediador\_cod\_count\_blacklist\_mad\_outlier | intermediary blacklist outlier by id |
| mediador\_nif\_count\_blacklist\_mad\_outlier | intermediary blacklist outlier by NIF |

## CUSTOMER\_OBJECT\_RESERVE

Here we analyze the coverture and guarantees involved in the sinister. Also, we have information about every payment or expenditure incurred by the company (not only to the insured).

|  |  |
| --- | --- |
| **Name** | **Description** |
| id\_poliza | Número de póliza |
| version\_poliza | Número de versión de la póliza |
| id\_siniestro | Código de Siniestro |
| id\_persona\_objeto\_reservable | Id de la persona / objeto reservable dentro del siniestro |
| po\_res\_garantia\_id | Id de la garantía |
| po\_res\_garantia | Garantías |
| po\_res\_cobertura\_id | ID de la cobertura |
| po\_res\_cobertura | Coberturas |
| po\_res\_limite | Límite de la Cobertura  IT: Datos de la cobertura a nivel póliza |
| po\_res\_indem | R i/m indem |
| po\_res\_gasto | R i/m gasto |
| po\_res\_situacion | Situación de la POR |
| po\_pago\_indicador\_indem | = 1 si existe pago por indemnización |
| po\_pago\_rehusado | Si no existe pago por indemnización y la situación está Terminada, entonces = 0 |
| po\_pago\_gasto\_codigo | Indemnización - Códiog de pago |
| po\_pago\_gasto\_codigo\_detective | Existe un código de Pago que corresponde a = "Gdetective/investigador" |
| po\_pago\_perceptor | Es el perceptor del pago de las indemnizaciones (Asegurados y Reparadores) |
| po\_pago\_medio | Indemnización - Medio de Pago |
| po\_pago\_IBAN | Indemnización - Cuenta IBAN |
| po\_pago\_emision | Indemnización - Fecha emisión |
| po\_pago\_factura\_fecha\_emision | Indemnización - Fecha emisión Factura |
| po\_pago\_factura\_importe\_neto | Indemnización - Importe Neto |
| po\_pago\_indemnizacion\_importe\_neto | Pago en caso de indemnizacion (caso en que no hay factura) |
| po\_gasto\_perceptor | Gastos - Perceptor |
| po\_gasto\_IBAN | Gastos -Cuenta IBAN |
| po\_gasto\_emision | Gastos -Fecha emisión |
| po\_gasto\_factura\_fecha\_emision | Gastos -Fecha emisión Factura |
| po\_gasto\_factura\_importe\_neto | Gastos -Importe Neto |
| po\_gasto\_indemnizacion\_importe\_neto | Gasto en caso de indemnizacion (caso en que no hay factura) |
| persona\_objeto\_asegurado | Indicador si la persona / objeto interviniente es el asegurado |
| po\_pago\_es\_anulacion | 0/1, 1 si un pago anula a otro (los 2 registros tendrán 1) |

This is one of the most complex bottles. So, we have to be very careful.

1- As always we start loading the file, transforming it to a Dataframe and mapping the key variables:

# Load File  
file = ReadCsv.load\_csv(name\_file)  
df\_id = ReadCsv.load\_csv(df\_id)  
  
# 2) Transform file to DF  
file\_df = DfUtils.processing\_file(file)  
df\_id = DfUtils.processing\_file(df\_id)  
  
file\_df = file\_df.sort\_values(by=['id\_siniestro'],  
 ascending=[**True**])

# Map Important Variables  
file\_df['id\_siniestro'] = file\_df['id\_siniestro'].map(int)

The bottle is organized by id\_siniestro – pay movements. Therefore, we have to reach the sinister, taking into account that we have several payment types and several recipient types.

2- We apply the checklist 5 to jewelry sinister.

# Checklist 5  
checklist5 = checklists\_obligatorias.checklist5(file\_df, df\_reserva\_test=**None**, df\_id=df\_id, df\_id\_test=**None**)  
file\_df = pd.merge(file\_df, checklist5, how='left', on='id\_siniestro')  
file\_df['checklist5\_poliza'] = file\_df['checklist5\_poliza'].fillna(0)  
file\_df['checklist5\_nif'] = file\_df['checklist5\_nif'].fillna(0)  
**del** checklist5

3- Now, we are going to calculate which is the initial compensation Reserve by sinister. Why? Because the initial reserve is associated with the pure costumer claim. For that, we have to group by the sinister by coverage. And then, we have to take the initial reserve value of each of them. Finally, we sum by sinister.

# RESERVA INICIAL INDEMNIZACION  
reserva\_indem = file\_df[['id\_siniestro', 'po\_res\_cobertura\_id', 'po\_res\_indem']]  
  
reserva\_indem = reserva\_indem.drop\_duplicates(subset=['id\_siniestro', 'po\_res\_cobertura\_id', 'po\_res\_indem'],  
 keep='first')  
**del** reserva\_indem['po\_res\_cobertura\_id']  
reserva\_indem['po\_res\_indem'] = reserva\_indem['po\_res\_indem'].map(float)  
reserva\_indem = reserva\_indem.groupby(['id\_siniestro'])['po\_res\_indem'].sum().reset\_index()

Also, we generate a dummy variable indicating when the cost is upper 5000 euros (following the checklist 4 strategy).

reserva\_indem['po\_res\_indem\_mayor\_5000'] = pd.Series(0, index=reserva\_indem.index)  
reserva\_indem.loc[reserva\_indem['po\_res\_indem'] >= 5000, 'po\_res\_indem\_mayor\_5000'] = 1  
**del** file\_df['po\_res\_indem']  
file\_df = pd.merge(file\_df, reserva\_indem, how='left', on='id\_siniestro')  
**del** reserva\_indem

4- We make the same procedure but using the expenditure payments (but limited to 1000 euros).

# RESERVA INICIAL GASTO  
reserva\_gasto = file\_df[['id\_siniestro', 'po\_res\_cobertura\_id', 'po\_res\_gasto']]  
  
reserva\_gasto = reserva\_gasto.drop\_duplicates(subset=['id\_siniestro', 'po\_res\_cobertura\_id', 'po\_res\_gasto'],  
 keep='first')  
**del** reserva\_gasto['po\_res\_cobertura\_id']  
reserva\_gasto['po\_res\_gasto'] = reserva\_gasto['po\_res\_gasto'].map(float)  
reserva\_gasto = reserva\_gasto.groupby(['id\_siniestro'])['po\_res\_gasto'].sum().reset\_index()  
  
reserva\_gasto['po\_res\_gasto\_mayor\_1000'] = pd.Series(0, index=reserva\_gasto.index)  
reserva\_gasto.loc[reserva\_gasto['po\_res\_gasto'] >= 1000, 'po\_res\_gasto\_mayor\_1000'] = 1  
**del** file\_df['po\_res\_gasto']  
file\_df = pd.merge(file\_df, reserva\_gasto, how='left', on='id\_siniestro')  
**del** reserva\_gasto

5- Now, we are going to weight the sinister by pay movements.

# COUNT POLIZAS POR SINIESTRO  
file\_df['pondera\_siniestro'] = pd.Series(1, index=file\_df.index)  
file\_df['po\_reserva\_pagoxsiniestro\_count'] = file\_df.groupby(  
 ['id\_siniestro'])['pondera\_siniestro'].apply(**lambda** x: x.cumsum())

6- We have to identify which compensation payments are cancelled. They have to be zero, because they do not take part of the real cost. The same with expenditures.

# PAGO INDEM ANULADOS: Cuando la anulación es == 1 marca tanto el pago como su anulación  
file\_df.loc[file\_df['po\_pago\_es\_anulacion'] == 1, 'po\_pago\_indemnizacion\_importe\_neto'] = 0  
  
# GASTO\_INDEM\_ANULADOS: Cuando el gasto es == 1 marca tanto el pago como su anulación  
file\_df.loc[file\_df['po\_gasto\_es\_anulacion'] == 1, 'po\_gasto\_indemnizacion\_importe\_neto'] = 0

7- We sum the total payments made. That is, bill payments and transfer payments. We make the same for expenditures.

# PAGOS: Sumamos el importe neto de factura + los pagos netos por indemnizaciòn  
file\_df['po\_pago\_factura\_importe\_neto'] = file\_df['po\_pago\_factura\_importe\_neto'].map(float)  
file\_df['po\_pago\_indemnizacion\_importe\_neto'] = file\_df['po\_pago\_indemnizacion\_importe\_neto'].map(float)  
  
file\_df['po\_pago\_importe\_neto'] = file\_df['po\_pago\_factura\_importe\_neto'] + \  
 file\_df['po\_pago\_indemnizacion\_importe\_neto']  
  
# GASTOS:  
file\_df['po\_gasto\_factura\_importe\_neto'] = file\_df['po\_gasto\_factura\_importe\_neto'].map(float)  
file\_df['po\_gasto\_indemnizacion\_importe\_neto'] = file\_df['po\_gasto\_indemnizacion\_importe\_neto'].map(float)  
  
file\_df['po\_gasto\_importe\_neto'] = file\_df['po\_gasto\_factura\_importe\_neto'] + \  
 file\_df['po\_gasto\_indemnizacion\_importe\_neto']

8- Now, we filter the values that are associated to the insured. Because we want to know exactly how much has received the insured. Therefore, if the payment is not make to the insured, then the payment is equal to zero.

# PAGO ASEGURADO: Si la persona no es el asegurado ponemos los importes del Asegurado en 0  
file\_df['persona\_objeto\_asegurado'] = file\_df['persona\_objeto\_asegurado'].map(int)  
file\_df['po\_pago\_importe\_neto\_ASEGURADO'] = file\_df['po\_pago\_importe\_neto']  
file\_df.loc[file\_df['persona\_objeto\_asegurado'] == 0, 'po\_pago\_importe\_neto\_ASEGURADO'] = 0

9- Porcentual Value: We calculate how much received the insured respect to the total amounts paid.

# IMPORTE PORCENTUAL QUE EFECTIVAMENTE COBRA EL ASEGURADO: importe\_neto\_asegurado/importe\_total  
file\_df['po\_pago\_importe\_porcentual\_ASEGURADO'] = pd.Series(file\_df['po\_pago\_importe\_neto\_ASEGURADO']  
 / file\_df['po\_pago\_importe\_neto'],  
 index=file\_df.index)

10- We check the IBAN involved in the payment. We chech if it belongs to the IBAN blacklist pregenerated. We check first for the compensations:

# IBAN FRAUDE: We check if the IBAN is associated with a previous Fraud Sinister  
file\_blacklist\_IBAN = ReadCsv.load\_csv(STRING.processed\_IBAN)  
df\_bl\_IBAN = DfUtils.processing\_file(file\_blacklist\_IBAN, delimiter=';')  
df\_bl\_IBAN = df\_bl\_IBAN.drop\_duplicates(subset=['IBAN'], keep='last')  
file\_df['po\_pago\_IBAN'] = file\_df['po\_pago\_IBAN'].map(str)  
df\_bl\_IBAN['IBAN'] = df\_bl\_IBAN['IBAN'].map(str)  
file\_df = pd.merge(file\_df, df\_bl\_IBAN, how='left', left\_on='po\_pago\_IBAN', right\_on='IBAN')  
file\_df['peritaje\_pago\_iban\_blacklist'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['IBAN'].notnull(), 'peritaje\_pago\_iban\_blacklist'] = 1  
**del** file\_df['IBAN']

Second, for the expenditures:

file\_df['po\_gasto\_IBAN'] = file\_df['po\_gasto\_IBAN'].map(str)  
file\_df = pd.merge(file\_df, df\_bl\_IBAN, how='left', left\_on='po\_gasto\_IBAN', right\_on='IBAN')  
file\_df['peritaje\_gasto\_iban\_blacklist'] = pd.Series(0, index=file\_df.index)  
file\_df.loc[file\_df['IBAN'].notnull(), 'peritaje\_gasto\_iban\_blacklist'] = 1

11- Integer Variables: We group by sinister the integer variables:

# INT VARIABLES  
# Agrupamos los valores INT por Siniestro y lo guardamos en la lista de INT's  
int\_var = ['peritaje\_pago\_iban\_blacklist', 'peritaje\_gasto\_iban\_blacklist']  
int\_outliers = []  
  
**for** i **in** int\_var:  
 count = i + '\_count'  
 file\_df[i] = file\_df[i].map(int)  
 file\_df[count] = file\_df.groupby(['id\_siniestro'])[i].apply(**lambda** x: x.cumsum())  
 int\_outliers.append(count)  
 **del** file\_df[i]

12 – We compute the max of person/object that participates in each sinister:

# PERSONA OBJETO RESERVABLE  
file\_df['id\_persona\_objeto\_reservable'] = file\_df['id\_persona\_objeto\_reservable'].map(int)  
file\_df['id\_persona\_objeto\_reservable\_max'] = file\_df.groupby(['id\_siniestro'])['id\_persona\_objeto\_reservable'  
 ].transform(max)

13- Now, we aggrupate some variables that are categorical. This is to reduce the scope, but trying to be consistent within the group.

a) For coding participants:

Asegurado\_Beneficiario\_Perjudicado = ['770', '778', '779', '601', '715', '740', '850', '851', '852', '363']  
Profesional\_Legal = ['774', '779', '780', '101', '102', '103', '104', '109', '113', '303', '723',  
 '724', '741', '743', '853', '861', '350', '351', '352', '354', '357', '801', '802', '803']  
Detective = ['105', '353', '804']  
Perito = ['106', '107', '108', '356', '360']  
Reparador = ['705']  
  
todos = Asegurado\_Beneficiario\_Perjudicado + Profesional\_Legal + Detective + Perito + Reparador  
  
  
  
file\_df.loc[file\_df['po\_pago\_gasto\_codigo'].isin(Asegurado\_Beneficiario\_Perjudicado),'po\_pago\_gasto\_codigo'] = 'Asegurado\_Beneficiario\_Perjudicado'  
file\_df.loc[file\_df['po\_pago\_gasto\_codigo'].isin(Profesional\_Legal),'po\_pago\_gasto\_codigo'] = 'Profesional\_Legal'  
file\_df.loc[file\_df['po\_pago\_gasto\_codigo'].isin(Detective),'po\_pago\_gasto\_codigo'] = 'Detective'  
file\_df.loc[file\_df['po\_pago\_gasto\_codigo'].isin(Perito),'po\_pago\_gasto\_codigo'] = 'Perito'  
file\_df.loc[file\_df['po\_pago\_gasto\_codigo'].isin(Reparador),'po\_pago\_gasto\_codigo'] = 'Reparador'  
file\_df.loc[-file\_df['po\_pago\_gasto\_codigo'].isin(todos),'po\_pago\_gasto\_codigo'] = 'Otros'

b) For guarantees:

# GARANTIA  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('RC'), 'po\_res\_garantia'] = 'RC'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('CIVIL'), 'po\_res\_garantia'] = 'RC'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('ROBO'), 'po\_res\_garantia'] = 'ROBO'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('HURTO'), 'po\_res\_garantia'] = 'ROBO'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('EXPO'), 'po\_res\_garantia'] = 'ROBO'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('CRIST'), 'po\_res\_garantia'] = 'CRISTALES'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('AGUA'), 'po\_res\_garantia'] = 'AGUA'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('DAG'), 'po\_res\_garantia'] = 'AGUA'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('ELECTR'), 'po\_res\_garantia'] = 'ELECTRICIDAD'  
file\_df.loc[file\_df['po\_res\_garantia'] == 'VV\_DE', 'po\_res\_garantia'] = 'ELECTRICIDAD'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('ATM'), 'po\_res\_garantia'] = 'ATMOSFERICO'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('INCENDIO'), 'po\_res\_garantia'] = 'INCENDIO'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('ESTETICA'), 'po\_res\_garantia'] = 'ESTETICA'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('JUR'), 'po\_res\_garantia'] = 'DEF\_JURIDICA'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('EXTENSION'), 'po\_res\_garantia'] = 'EXTENSION'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('CONTINENTE'), 'po\_res\_garantia'] = 'CONTINENTE'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('CONTENIDO'), 'po\_res\_garantia'] = 'CONTENIDO'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('CTE'), 'po\_res\_garantia'] = 'CONTINENTE'  
file\_df.loc[file\_df['po\_res\_garantia'].str.contains('CDO'), 'po\_res\_garantia'] = 'CONTENIDO'

c) For Coverages:

# COBERTURA  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('PORCALOR'), 'po\_res\_cobertura'] = 'CALOR'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('VALLAS'), 'po\_res\_cobertura'] = 'VALLAS'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('OTR'), 'po\_res\_cobertura'] = 'OTRO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('RC'), 'po\_res\_cobertura'] = 'RC'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('ROBO'), 'po\_res\_cobertura'] = 'ROBO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('HURTO'), 'po\_res\_cobertura'] = 'ROBO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('ATR'), 'po\_res\_cobertura'] = 'ROBO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('EXPO'), 'po\_res\_cobertura'] = 'ROBO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('CRIS'), 'po\_res\_cobertura'] = 'CRISTALES'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('INUNDA'), 'po\_res\_cobertura'] = 'INUNDACION'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('ELECT'), 'po\_res\_cobertura'] = 'ELECTRICIDAD'  
file\_df.loc[file\_df['po\_res\_cobertura'] == 'DE', 'po\_res\_cobertura'] = 'ELECTRICIDAD'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('CHOQUE'), 'po\_res\_cobertura'] = 'CHOQUE'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('INC'), 'po\_res\_cobertura'] = 'INCENDIO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('AGUA'), 'po\_res\_cobertura'] = 'AGUA'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('RAYO'), 'po\_res\_cobertura'] = 'ATMOSFERICO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('VIENTO'), 'po\_res\_cobertura'] = 'ATMOSFERICO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('LLUVIA'), 'po\_res\_cobertura'] = 'ATMOSFERICO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('PEDRISCO'), 'po\_res\_cobertura'] = 'ATMOSFERICO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('JUR'), 'po\_res\_cobertura'] = 'DEF\_JURIDICA'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('LLAVE'), 'po\_res\_cobertura'] = 'LLAVES'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('VANDAL'), 'po\_res\_cobertura'] = 'VANDALISMO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('ALIM'), 'po\_res\_cobertura'] = 'ALIMENTOS'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('EXC.'), 'po\_res\_cobertura'] = 'CONTENIDO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('JOY'), 'po\_res\_cobertura'] = 'JOYAS'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('METALICO'), 'po\_res\_cobertura'] = 'METALICO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('VITRO'), 'po\_res\_cobertura'] = 'VITROCERAMICA'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('ACC'), 'po\_res\_cobertura'] = 'ACCIDENTE'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('VV\_EXT'), 'po\_res\_cobertura'] = 'EXTERIOR'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('ESTET'), 'po\_res\_cobertura'] = 'ESTETICA'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('DAEST'), 'po\_res\_cobertura'] = 'ESTETICA'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('DEST'), 'po\_res\_cobertura'] = 'ESTETICA'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('RL'), 'po\_res\_cobertura'] = 'RL'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('INMB'), 'po\_res\_cobertura'] = 'INMUEBLE'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('MAT'), 'po\_res\_cobertura'] = 'MATERIAL'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('ALQ'), 'po\_res\_cobertura'] = 'ALQUILER'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('SAN'), 'po\_res\_cobertura'] = 'SANITARIO'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('FRAUD'), 'po\_res\_cobertura'] = 'FRAUDE\_TARJETAS'  
file\_df.loc[file\_df['po\_res\_cobertura'].str.contains('CDO'), 'po\_res\_cobertura'] = 'CONTENIDO'

14- We get dummies for each categorical variable:

categorical\_var = ['po\_res\_garantia', 'po\_res\_cobertura', 'po\_res\_situacion',  
 'po\_pago\_medio', 'po\_pago\_gasto\_codigo']  
variable\_dummy = []  
**for** i **in** categorical\_var:  
 file\_df[i] = file\_df[i].map(str)  
 prefijo = 'd\_' + i  
 dummy\_i = pd.get\_dummies(file\_df[i], prefix=prefijo)  
 file\_df = pd.concat([file\_df, dummy\_i], axis=1)  
 name\_var = dummy\_i.columns.tolist()  
 variable\_dummy += name\_var  
 **del** dummy\_i  
 **del** file\_df[i]

We group by sinister those categorical variables:

# Agrupamos las dummies por siniestro  
variable\_dummy += ['po\_pago\_indicador\_indem', 'po\_pago\_rehusado', 'po\_pago\_gasto\_codigo\_detective',  
 'persona\_objeto\_asegurado']  
  
**for** i **in** variable\_dummy:  
 name = i + '\_count'  
 file\_df[i] = file\_df[i].fillna(0)  
 file\_df[i] = file\_df[i].map(int)  
 file\_df[name] = file\_df.groupby(  
 ['id\_siniestro'])[i].apply(**lambda** x: x.cumsum())  
 **del** file\_df[i]

We calculate the percent value in each sinister by the situation in that sinister. That is because we want to separate finished sinister respect to open sinister. The last case will apply when we have new income sinister. Also with ‘indicador\_indem’.

# Ajustamos las situaciones para representarlo porcentualmente:  
var\_sit = file\_df.columns.values.tolist()  
var\_sit = [col **for** col **in** var\_sit **if** col.startswith('d\_po\_res\_situacion')]  
**for** i **in** var\_sit:  
 file\_df[i] = file\_df[i] / file\_df['po\_reserva\_pagoxsiniestro\_count']  
**del** var\_sit  
  
# Ajustamos po\_pago\_indicador\_indem para verlo porcentualmente  
file\_df['po\_pago\_indicador\_indem\_count'] = file\_df['po\_pago\_indicador\_indem\_count'] / file\_df[  
 'po\_reserva\_pagoxsiniestro\_count']

15- Float Variables: We take the relevant float variables and group them by sinister. Also, we save them in the Outlier list.

# FLOAT VARIABLES  
# Agrupamos las FLOAT por siniestro y las ponemos para analizar como Outliers  
variable\_float\_perceptor = ['po\_pago\_factura\_importe\_neto', 'po\_pago\_indemnizacion\_importe\_neto',  
 'po\_gasto\_factura\_importe\_neto', 'po\_gasto\_indemnizacion\_importe\_neto',  
 'po\_pago\_importe\_neto', 'po\_gasto\_importe\_neto',  
 'po\_pago\_importe\_neto\_ASEGURADO',  
 'po\_pago\_importe\_porcentual\_ASEGURADO',  
 ]  
float\_outliers = []  
**for** i **in** variable\_float\_perceptor:  
 name = i + '\_count'  
 file\_df[i] = file\_df[i].fillna(0)  
 file\_df[i] = file\_df[i].map(float)  
 file\_df[name] = file\_df.groupby(  
 ['id\_siniestro'])[i].apply(**lambda** x: x.cumsum())  
 float\_outliers.append(name)

16- We get the percent of average insured payments accumulated by sinister.

# Porcentual\_Asegurado: Lo ajustamos para que refleje el valor porcentual  
file\_df['po\_pago\_importe\_porcentual\_ASEGURADO\_count'] = file\_df['po\_pago\_importe\_porcentual\_ASEGURADO\_count'] \  
 / file\_df['po\_reserva\_pagoxsiniestro\_count']

17- Now we are going to get several statistics by recipient:

a) First we standardize the AIDE ASISTENCIA recipient.

# TABLA DE PERCEPTOR  
file\_df.loc[file\_df['po\_pago\_perceptor'].str.startswith('AIDE ASISTENCIA'), 'po\_pago\_perceptor'] = 'AIDE ASISTENCIA'

b) We create an auxiliary dataframe with the relevant variables. This dataframe only contains payments to a recipient.

# Aquí hacemos análisis por perceptor de la indemnización en términos globales  
perceptor = file\_df[['po\_pago\_perceptor', 'id\_siniestro',  
 'po\_pago\_factura\_importe\_neto',  
 "po\_pago\_indemnizacion\_importe\_neto", "po\_pago\_es\_anulacion",  
 'pondera\_siniestro'  
 ]]  
  
perceptor.loc[perceptor['po\_pago\_perceptor'] == '0', 'po\_pago\_perceptor'] = np.NaN  
perceptor = perceptor.dropna(subset=['po\_pago\_perceptor'], how='any')

c) We get total payments count by recipient and total cancelled payments:

# Por Pagos  
perceptor['po\_pagos\_total\_countxperceptor'] = perceptor.groupby(  
 ['po\_pago\_perceptor'])['pondera\_siniestro'].apply(**lambda** x: x.cumsum())

variable\_dummy\_perceptor = ['po\_pago\_es\_anulacion']  
**for** i **in** variable\_dummy\_perceptor:  
 name = i + '\_countxperceptor'  
 perceptor[i] = perceptor[i].fillna(0)  
 perceptor[i] = perceptor[i].map(int)  
 perceptor[name] = perceptor.groupby(  
 ['po\_pago\_perceptor'])[i].apply(**lambda** x: x.cumsum())  
 **del** perceptor[i]

d) Also we get total payments by bill and by transferences:

variable\_float\_perceptor = ['po\_pago\_factura\_importe\_neto', 'po\_pago\_indemnizacion\_importe\_neto']  
# Nota: está bien que haya valores negativos porque son los recobros a otras empresas  
**for** i **in** variable\_float\_perceptor:  
 name = i + '\_countxperceptor'  
 perceptor[i] = perceptor[i].map(float)  
 perceptor[name] = perceptor.groupby(  
 ['po\_pago\_perceptor'])[i].apply(**lambda** x: x.cumsum())  
 **del** perceptor[i]

e) We transform everything to sinister level:

perceptor = perceptor.drop\_duplicates(subset = ['po\_pago\_perceptor', 'id\_siniestro'], keep='last')  
perceptor['po\_pagoxsiniestro\_countxperceptor'] = perceptor.groupby(  
 ['po\_pago\_perceptor'])['pondera\_siniestro'].apply(**lambda** x: x.cumsum())  
**del** perceptor['pondera\_siniestro']

f) We get the average values by recipient:

# Obtenemos los niveles promedio por Perceptor  
promedio\_var = ['po\_pago\_factura\_importe\_neto\_countxperceptor',  
 'po\_pago\_indemnizacion\_importe\_neto\_countxperceptor']  
  
**for** i **in** promedio\_var:  
 name\_promedio = i + 'xpromediosiniestro'  
 perceptor[name\_promedio] = pd.Series(perceptor[i]/perceptor['po\_pagoxsiniestro\_countxperceptor'],  
 index=perceptor.index)

e) We check in the blacklist if the recipient is present and how many times (and also in percent values):

# Aparece en blacklist y cuántas  
# We merge with the known cases of Fraud  
file\_blacklist\_resume = ReadCsv.load\_csv(STRING.blacklist\_processed\_resume)  
df\_bl\_resume = DfUtils.processing\_file(file\_blacklist\_resume, delimiter=';')  
df\_bl\_resume['ID\_SINIESTRO'] = df\_bl\_resume['ID\_SINIESTRO'].map(int)  
perceptor = pd.merge(perceptor, df\_bl\_resume, how='left', left\_on='id\_siniestro', right\_on='ID\_SINIESTRO')  
perceptor['po\_reserva\_perceptor\_fraud'] = pd.Series(0, index=perceptor.index)  
perceptor.loc[perceptor['ID\_SINIESTRO'].notnull(), 'po\_reserva\_perceptor\_fraud'] = 1  
**del** df\_bl\_resume  
**del** perceptor['ID\_SINIESTRO']  
  
perceptor['po\_fraude\_countxperceptor'] = perceptor.groupby(  
 ['po\_pago\_perceptor'])['po\_reserva\_perceptor\_fraud'].apply(**lambda** x: x.cumsum())  
**del** perceptor['po\_reserva\_perceptor\_fraud']  
  
perceptor['po\_fraude\_porcentaje\_perceptor'] = pd.Series(  
 perceptor['po\_fraude\_countxperceptor']/perceptor['po\_pagoxsiniestro\_countxperceptor'],  
 index=perceptor.index)

f) Finally, we transform everything to the recipient level and send a file to a CSV. This will be necessary when we have the new sinister. It will be easier to call this auxiliary table. We merge with the original dataframe.

perceptor = perceptor.drop\_duplicates(subset=['po\_pago\_perceptor'], keep='last')  
  
**del** perceptor['id\_siniestro']  
  
perceptor.to\_csv(STRING.auxiliar\_po\_perceptor, sep=';', index=**False**)  
  
file\_df = pd.merge(file\_df, perceptor, how='left', on='po\_pago\_perceptor')  
perceptor\_cols = perceptor.columns.values.tolist()  
perceptor\_cols.remove('po\_pago\_perceptor')  
**for** i **in** perceptor\_cols:  
 file\_df[i] = file\_df[i].fillna(0)  
**del** perceptor  
**del** perceptor\_cols

18- We do a similar analysis apply to the services supplied.

a) We create a reduced dataframe using ‘po\_gasto\_perceptor’ as the key variable.

servicios = file\_df[['po\_gasto\_perceptor', 'id\_siniestro',  
 'po\_gasto\_factura\_importe\_neto', 'po\_gasto\_indemnizacion\_importe\_neto',  
 'pondera\_siniestro'  
 ]]  
  
servicios.loc[servicios['po\_gasto\_perceptor'] == '0', 'po\_gasto\_perceptor'] = np.NaN  
servicios = servicios.dropna(subset=['po\_gasto\_perceptor'], how='any')

b) We get total payments count by service:

# por Gastos  
servicios['po\_pagos\_total\_countxservicios'] = servicios.groupby(  
 ['po\_gasto\_perceptor'])['pondera\_siniestro'].apply(**lambda** x: x.cumsum())

c) Also we get total payments by bill and by transferences:

variable\_float\_servicios = ['po\_gasto\_factura\_importe\_neto', 'po\_gasto\_indemnizacion\_importe\_neto']  
  
**for** i **in** variable\_float\_servicios:  
 name = i + '\_countxservicios'  
 servicios[i] = servicios[i].fillna(0)  
 servicios[i] = servicios[i].map(float)  
 servicios[name] = servicios.groupby(  
 ['po\_gasto\_perceptor'])[i].apply(**lambda** x: x.cumsum())  
 **del** servicios[i]

d) We transform everything to sinister level:

# Ponemos *todo a nivel siniestro*servicios = servicios.drop\_duplicates(subset=['po\_gasto\_perceptor', 'id\_siniestro'], keep='last')  
  
servicios['po\_pagoxsiniestro\_countxservicios'] = servicios.groupby(  
 ['po\_gasto\_perceptor'])['pondera\_siniestro'].apply(**lambda** x: x.cumsum())  
**del** servicios['pondera\_siniestro']

e) We get the averages values by service:

**for** i **in** promedio\_var:  
 name\_promedio = i + 'xpromediosiniestro'  
 servicios[name\_promedio] = pd.Series(servicios[i] / servicios['po\_pagoxsiniestro\_countxservicios'],  
 index=servicios.index)

f) We check if the service supplier appears in the blacklist and how many times:

# Aparece en blacklist y cuántas  
file\_blacklist\_resume = ReadCsv.load\_csv(STRING.blacklist\_processed\_resume)  
df\_bl\_resume = DfUtils.processing\_file(file\_blacklist\_resume, delimiter=';')  
df\_bl\_resume['ID\_SINIESTRO'] = df\_bl\_resume['ID\_SINIESTRO'].map(int)  
servicios = pd.merge(servicios, df\_bl\_resume, how='left', left\_on='id\_siniestro', right\_on='ID\_SINIESTRO')  
servicios['po\_reserva\_servicios\_fraud'] = pd.Series(0, index=servicios.index)  
servicios.loc[servicios['ID\_SINIESTRO'].notnull(), 'po\_reserva\_servicios\_fraud'] = 1  
**del** df\_bl\_resume  
**del** servicios['ID\_SINIESTRO']  
  
servicios['po\_fraude\_countxservicios'] = servicios.groupby(  
 ['po\_gasto\_perceptor'])['po\_reserva\_servicios\_fraud'].apply(**lambda** x: x.cumsum())  
**del** servicios['po\_reserva\_servicios\_fraud']  
  
servicios['po\_fraude\_porcentaje\_servicios'] = pd.Series(  
 servicios['po\_fraude\_countxservicios'] / servicios['po\_pagoxsiniestro\_countxservicios'],  
 index=servicios.index)

g) We transform everything to the service supplier level and send a file to a CSV.

servicios = servicios.drop\_duplicates(subset=['po\_gasto\_perceptor'], keep='last')  
  
servicios.to\_csv(STRING.auxiliar\_po\_servicios, sep=';', index=**False**)  
  
**del** servicios['id\_siniestro']  
  
file\_df = pd.merge(file\_df, servicios, how='left', on='po\_gasto\_perceptor')  
  
servicios\_cols = servicios.columns.values.tolist()  
servicios\_cols.remove('po\_gasto\_perceptor')  
**for** i **in** servicios\_cols:  
 file\_df[i] = file\_df[i].fillna(0)  
**del** servicios  
**del** servicios\_cols

19 – Outliers: We were collecting several variables in the outlier list. Using this, we calculate outliers for each of them.

**for** i **in** int\_outliers:  
 Outliers.outliers\_mad(file\_df, i, not\_count\_zero=**True**)  
  
**for** i **in** float\_outliers:  
 Outliers.outliers\_mad(file\_df, i, not\_count\_zero=**True**)

20- We delete useless variables:

del\_variables = ['id\_poliza', 'version\_poliza', "po\_res\_garantia\_id",  
 "po\_res\_cobertura\_id", 'po\_res\_limite',  
 "po\_pago\_IBAN", "po\_pago\_emision", "po\_pago\_factura\_fecha\_emision",  
 'po\_pago\_factura\_importe\_neto', 'po\_pago\_indemnizacion\_importe\_neto',  
 'po\_gasto\_IBAN', 'po\_gasto\_emision', 'po\_gasto\_factura\_fecha\_emision',  
 'po\_pago\_es\_anulacion', 'po\_gasto\_es\_anulacion', 'pondera\_siniestro',  
 'po\_gasto\_perceptor', 'po\_gasto\_factura\_importe\_neto', 'po\_gasto\_indemnizacion\_importe\_neto',  
 'po\_pago\_importe\_neto', 'po\_gasto\_importe\_neto',  
 'po\_pago\_importe\_neto\_ASEGURADO', 'po\_pago\_importe\_porcentual\_ASEGURADO'  
 ]  
  
**for** i **in** del\_variables:  
 **del** file\_df[i]

## HISTORICAL\_SINISTER

Historical movements associated to the reference sinister.

## HISTORICAL\_POLICY

Historical movements associated to the reference policy (the policy involved in the sinister).

## HISTORICAL\_OTHER\_POLICIES

Historical movements of any other policy (property or not) related to the reference policy.

## HISTORICAL\_OTHER \_SINISTER

Historical sinister associated to the reference policy (excluding the reference sinister).

## HISTORICO\_OTHER\_POL\_SINISTER

Other sinister associated to other policies that are not in the reference policy (but are related by the customer).

## BLACKLIST

Originally, this is a list with all the fraud sinister that were discovered by the company (since 2010). This list, is expanded with information associated to IBANs and every participants involved.

## AUXILIAR FILES

Here there is a compile of auxiliary files that are helpful for both processes: Daily and Weekly processes.

# DAILY AND WEEKLY PRE PROCESSING

We create basically two modules. One that will be executed once time per week (semanal.py) and the other that will be executed every day (diario.py).

## WEEKLY PRE-PROCESSING

The weekly module processes a huge amount of historical information. First, it has to be processed by IT. Then, it has to be processed by Z-Finder program. Therefore, we think it is optimal to be processed not every day. We will not lost too much information (only the closed sinister in that week). But we are going to sharply improve in processing speed.

Essentially, this will apply [files processing section](#filesprocessing) for each bottle.

## DAILY PRE-PROCESSING

The daily files are the new sinister. We have to process 18 bottles for a few new income sinister. Therefore, it is supposed it should not take too much time in the processing step.

In this module, we calculate each test bottle involved, with some exceptions.

1) Intermediary Bottle: It is not calculated because it reflects global statistics from the intermediary behavior. Therefore, it has not sense only calculate this for the new sinister. Thus, when we calculate policy bottle, we append the whole intermediary dataset and not just the new sinister.

2) Customer Property Bottle: This uses crossed information with EUROPA and ID bottles, so we need to take not the original dataframe. It needs to use the new sinister bottles.

3) EUROPA: The same as before. But using Date and ID bottles.

The remaining bottles are calculated as in weekly pre-processing section with the next exceptions:

1) We have some variables that are function of the total column set. Mainly, we are speaking about the historical bottles. This implies that we have to append the daily information to the base bottle column and calculate the whole bottle again.

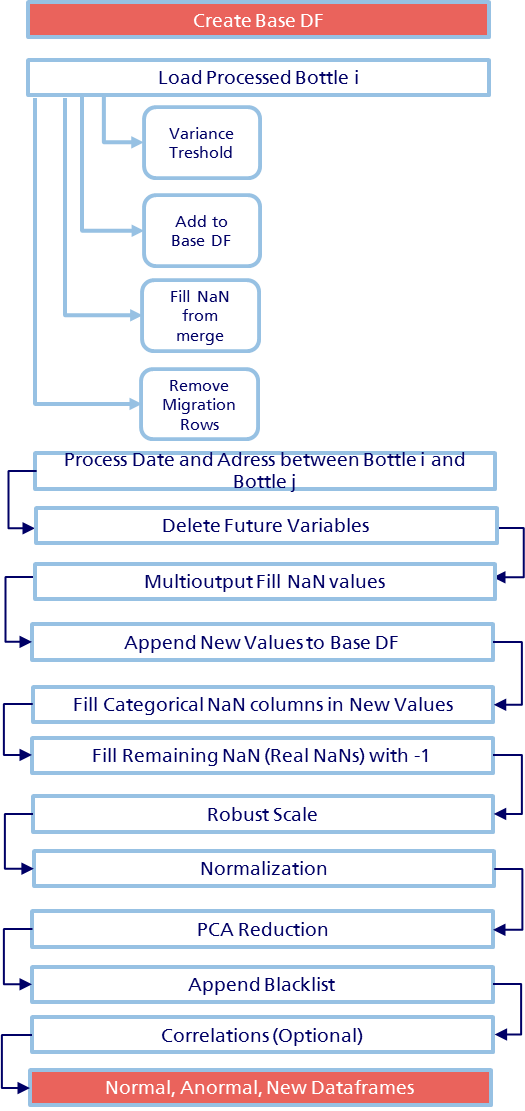
2) Also we have several group calculations (loss adjuster information, intermediary information, etc.). In this cases we have to retrieve the information from the original bottles. For that, we have created auxiliary tables that are generated in the weekly process. The daily process take the information from there.

3) Outliers are calculated as information from the whole base bottle column. Therefore, we have to create an additional method (see the [Outlier for Novelty Data Section](#_OUTLIERS_FOR_NOVELTY)), to get information from the original bottle columns to calculate the new outliers.

4) Also we have to fill that columns that are present in the weekly process but are not present in the new data. We are talking about that categories that are automatically generated based on data. They are not NaN values. It happens because some categories are not present in the new bottle data.

# PROCESSING TABLE

Once we have every file processed we continue to process the data as a whole. We divide in several steps each part of the process. We briefly summarize it as follows:



First of all we get from our output\_file path every document that is saved. This documents are passed to a list, with the exception of the blacklist files (we do not take into account files with ‘\_bl\_’).

path = os.getcwd() + '\\' + STRING.path\_processing  
os.chdir(path)  
files = set([f **for** f **in** os.listdir(path)])  
files = [f **for** f **in** files **if** '\_bl\_' **not in** f]

Next, we create an empty dataframe that will be our main object.

df = pd.DataFrame()

And also, we bring our new sinister to keep them aside (for now):

df\_test = diario.processing\_test\_diario()

Now, we create a loop for each document involved in files object. And using our utils ReadCsv and DFUtils we read the file with extension .csv and then we convert it in a dataframe.

In the next points, we will using exclusively the process\_utils module. You can take a look in [Utils Section](#utils).

## 1. VARIANCE THRESHOLD

From process\_utils we import the method variance threshold.

**def variance\_threshold**(self: pd.DataFrame, threshold=0.0):  
 *"""   
 VarianceThreshold is a simple baseline approach to feature selection. It removes all features whose variance  
 doesn’t meet some threshold. By default, it removes all zero-variance features, i.e.  
 features that have the same value in all samples.  
 As an example, suppose that we have a dataset with boolean features,  
 and we want to remove all features that are either one or zero (on or off) in more than 80% of the samples.  
 """* column\_names = self.columns.values.tolist()  
 key\_variables = ['id\_siniestro', 'id\_poliza', 'cod\_filiacion']  
 removed\_var = []  
 **for** i **in** key\_variables:  
 **try**:  
 column\_names.remove(i)  
 removed\_var.append(i)  
 **except**:  
 **pass** append\_names = []  
 **for** i **in** column\_names:  
 self\_i = self[[i]]  
 self\_i = self\_i.apply(pd.to\_numeric, errors='coerce')  
 self\_i = self\_i.dropna(how='any', axis=0)  
 selection = VarianceThreshold(threshold=threshold)  
 **try**:  
 selection.fit(self\_i)  
 features = selection.get\_support(indices=**True**)  
 features = [column **for** column **in** self\_i[features]]  
 selection = pd.DataFrame(selection.transform(self\_i), index=self\_i.index)  
 selection.columns = features  
 append\_names.append(selection.columns.values.tolist())  
 **except**:  
 **pass** append\_names = [item **for** sublist **in** append\_names **for** item **in** sublist]  
 self = self[removed\_var + append\_names]  
 **return** self

Basically, it is a univariant method that permits make a feature selection to remove low-variance features. Because it is univariant it can be used for unsupervised learning.

In our code we stipulate a threshold = 0.0. Therefore, only if we have a constant column, it will be removed. Why we do not use a less flexible threshold? Basically because we have an outlier problem thus we could be deleting relevant points in terms of fraud detection.

## 2. APPEND DATAFRAME

Here we call a very simple method from process\_utils but that will be relevant for the next step. Essentially it does the next actions:

* Cast the key variable of the original Dataframe.
* Cast the key variable of the new Dataframe.
* Join Left between both on the key variable.
* It saves the column names of the new Dataframe without the key variable.

**def append\_df**(df\_base: pd.DataFrame, df\_add: pd.DataFrame, on\_var:str='id\_siniestro', on\_var\_type=int):  
 *"""  
 It appends a dataframe based on 'id\_siniestro' using join left. Also it returns the column names of the new  
 dataframe so in the next step we can evaluate missing values* ***:param*** *df\_add: The new DataFrame* ***:param*** *on\_var: The key column* ***:param*** *on\_var\_type: The type of the key column* ***:return****: df\_base + df\_add, df\_add column names  
 """* print('addding... ')  
 print(df\_add.columns.values.tolist())  
 print('initial shape ', df\_base.shape)  
 df\_base[on\_var] = df\_base[on\_var].map(on\_var\_type)  
 df\_add[on\_var] = df\_add[on\_var].map(on\_var\_type)  
 df\_base = pd.merge(df\_base, df\_add, how='left', on=on\_var)  
 df\_add\_cols = df\_add.columns.values.tolist()  
 df\_add\_cols.remove(on\_var)  
 print('final shape ', df\_base.shape)  
 **return** df\_base, df\_add\_cols

## 3. FILL THE NAN VALUES IN THE NEW BOTTLE

At this stage, we are going to fill the null values in the new bottle. But it is important to remark that we will not fill every empty cell. Why?

At this moment, we have two types of null values:

1) Null values that were present before we append the new bottle to the total Dataframe.

2) Null values originated in the join left between the new bottle and the total Dataframe.

The first type are null derivated from the original distribution of the variables. So for now, we want to keep them unchanged.

The second type are originated because the key variable could not be crossed with the values in the new bottle. This means that we do not have any value in the new bottle row. This case emerges for example when we cross the HISTORICAL\_OTHER\_POLICIES bottle with the whole Dataframe. If the customer has not had another policy, his row will be empty. And that is an important information for us. Therefore we cannot consider it as a simple NULL value.

In conclusion, we use the next method us a way to identify this patterns.

**def fillna\_by\_bottle**(df\_base: pd.DataFrame, df\_add\_cols: list, fill\_value=-1):  
 *"""  
 Using append\_df, we get the column names added. Then we will fillna only if the whole columns are NaN  
 after they have been appended.* ***:param*** *df\_add\_cols: the column names just added* ***:param*** *fill\_value: The fillna value we choose* ***:return****: Dataframe with the fillna process  
 """* condition\_nan\_values = len(df\_add\_cols)  
 # We create a variable that count how many NAN values are in the selected columns df\_add\_cols  
 df\_base['count\_NAN'] = df\_base[df\_add\_cols].apply(**lambda** x: x.isnull().count(), axis=1)  
  
 # We make a fillna only if 'count\_NAN' is exactly the number of new columns  
 df\_base[df\_add\_cols].apply(**lambda** x: x.fillna(fill\_value) **if** x['count\_NAN'] == condition\_nan\_values  
 **else** x, axis=0)  
  
 **del** df\_base['count\_NAN']  
  
 **return** df\_base

As you can see, first we create a new column that counts the null values for each row in the new columns aggregated (using the list created in the previous step).

Second, using the numbers of new columns as a restriction, we fill every row that has nulls in every column.

## 4. DELETE COLUMN BY A SUBSTRING NAME

It may be possible we want to delete several bad imputation rows. For that, we create a method that check for a column that identifies this type of rows. For example, we now that every row that was created by a MIGRATION process maybe was not correctly specified. Because of we do not want observations that can distortion the real distribution of the variables, it is possible we want to delete these rows.

**def delete\_row\_df\_by\_name**(df\_base: pd.DataFrame, del\_name:str = 'MIGRA'):  
 *"""  
 If the del\_name is contained in a column name it will delete the row which is == 1 from the dataframe* ***:param*** *del\_name: String we want to search for deleting porpose.* ***:return****: df\_base without the rows found  
 """* col\_names = df\_base.columns.values.tolist()  
 **for** i **in** col\_names:  
 **if** del\_name **not in** i:  
 col\_names.remove(i)  
  
 **for** i **in** col\_names:  
 rows\_0 = len(df\_base.index)  
 df\_base = df\_base[df\_base[i] != 1]  
 **del** df\_base[i]  
 print(i + ' was Deleted')  
  
 rows\_1 = len(df\_base.index)  
 print(rows\_0 - rows\_1, ' rows were deleted')  
  
 **return** df\_base

Basically, the method will check every column that we identify with a substring (by default it is ‘MIGRA’) and delete every row which value is equal to one. Then, it returns the original Dataframe without the rows and the columns involved.

Once we develop the processing task by bottle, we continue processing the table as a whole.

## 5. MULTIOUTPUT FILL NAN STRATEGY

This is one of the most complex part of the pre-processing analysis. Consequently, we are going to take some time to explain which the empirical strategy is.

You can see the method fillna\_multioutput used. We need to pass two parameters: Our final Dataframe and a list of variables that are redundant as ‘id\_siniestro’.

As we explained before we had two types of NULL values. Here, we are trying to solve the first type. Our real NULL values present in the variables’ distribution. As different from the other types, NULL could represent whatever.

Hence, we first determines which columns have null values and which not:

jcols = df.columns[df.isnull().any()].tolist()  
icols = df.columns.values.tolist()  
  
icols = [icols.remove(i) **for** i **in** jcols]

Our jcols are the name of the columns with NULL, and icols represents the not-null columns.

Now, we want to know which specific rows possess NULL values. This is because we need to estimate this and only this rows. Considering that, we create an auxiliary dataframe. This dataframe contains the columns derivated from jcols (the null columns) and the cells are just expressed as Boolean values. If the cell is not null it will return True (False instead).

notnans = df[jcols].notnull().all(axis=1)

We show this schematically below:

Original Dataframe:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INDEX | icols\_1 | Icols\_2 | Jcols\_1 | Jcols\_2 |
| 1 | 1 | 2 | 1 | NaN |
| 2 | 2 | 3 | NaN | NaN |
| 3 | 4 | 4 | 2 | 4 |

Notnans Dataframe:

|  |  |  |
| --- | --- | --- |
| INDEX | Jcols\_1 | Jcols\_2 |
| 1 | True | False |
| 2 | False | False |
| 3 | True | True |

Using notnans dataframe, we create a dataframe with the total columns but without the rows with NAN values. What we want is to use this dataframe as the Training Set. Therefore, it contains the total variables without NaN values.

Df\_notnans Dataframe:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INDEX | icols\_1 | Icols\_2 | Jcols\_1 | Jcols\_2 |
| 3 | 4 | 4 | 2 | 4 |

Now, it is clearer. Basically, we will estimate Jcols (the original NULL column variables) using the Icols:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_notnans[icols], df\_notnans[jcols], train\_size=0.70, random\_state=42)

Where:

Df\_notnans[icols] = X =

|  |  |  |
| --- | --- | --- |
| INDEX | icols\_1 | Icols\_2 |
| 3 | 4 | 4 |

And:

Df\_notnans[jcols] = Y =

|  |  |  |
| --- | --- | --- |
| INDEX | jcols\_1 | jcols\_2 |
| **3** | **2** | **4** |

Then, we estimate using a Random Forest. We use Random Forest because we have too many variables and not a great amount of observations.

regr\_multirf = MultiOutputRegressor(RandomForestRegressor(n\_estimators=n\_estimator, max\_depth=max\_depth, random\_state=42, verbose=1, max\_features=max\_features, min\_samples\_split=min\_samples\_split, min\_samples\_leaf=min\_samples\_leaf))

We need to use a Multioutput Regression because we have several target variables (all the icols). This strategy consists of fitting one regressor per target.

Following, we fit the model and get the R2 score of the model.

regr\_multirf.fit(X\_train.drop(not\_consider, axis=1), y\_train)  
  
score = regr\_multirf.score(X\_test.drop(not\_consider, axis=1), y\_test)

Once we have tried the performance of our model, we have to bring our dataframe with the rows that only have null values, because this is our final goal.

df\_nans = df.loc[~notnans].copy()

df\_nans represents the Dataframe containing the whole columns but with only the rows which have at least one NaN value.

Df\_nans Dataframe

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INDEX | icols\_1 | Icols\_2 | Jcols\_1 | Jcols\_2 |
| 1 | 1 | 2 | 1 | NaN |
| 2 | 2 | 3 | NaN | NaN |

Using icols we are going to estimate jcols, with the model generated before.

df\_nans[jcols] = regr\_multirf.predict(df\_nans[icols].drop(not\_consider, axis=1))

|  |  |  |
| --- | --- | --- |
| INDEX | icols\_1 | Icols\_2 |
| 1 | 1 | 2 |
| 2 | 2 | 3 |

Where:

Df\_nans[icols] = X =

For estimating df\_nans[jcols]:

|  |  |  |
| --- | --- | --- |
| INDEX | Jcols\_1 | Jcols\_2 |
| 1 | 1 | NaN |
| 2 | NaN | NaN |

Df\_nans[jcols] = Y =

Our final result was a **R2 of 89%**.

## 11. APPEND AND MARK TEST

This is a parallel step really. Here, we bring the new sinister to be analyzed. Exactly before we start to rescale and to reduce our dataset.

# 11) APPEND AND MARK TEST  
df['TEST'] = pd.Series(0, index=df.index)  
df\_test['TEST'] = pd.Series(1, index=df.index)  
df = df.append(df\_test, ignore\_index=**True**)  
print('Nan Values with test ', df.isnull().sum())  
# First, we fill the categorical variables because they are not NaN  
**for** i **in** STRING.fillna\_vars:  
 df[i] = df[i].fillna(0)  
print('Nan Values after categoric fill ', df.isnull().sum())  
# Second, we fill the remaining NULL values  
df = df.fillna(-1)

As you can see, we have a new dataframe df\_test that we brought at the beginning of the processing table section.

We created a new variable ‘TEST’ that will reveal to which sample belongs each row. Also, we need to make some imputation assumption in NaN values to apply rescale and PCA methods.

We have three types of NaN values here. The first type is associated with categories that are in the base dataframe but are not included in the new dataframe. This is because several categories are generated automatically. Therefore, in this case, we have to fill that categories with 0 because they are represented by another category (This categories are identified in STRING.fillna\_vars).

The possible second type of null values are the real NaN values. As we are marking with -1, that NaN values that are strictly NULL values, we continue with the same metodhology (refer to the [NaN values section](#_3._FILL_THE)).Finally, we append to the Base Dataframe.

Exists a third type of null values (but less probable). They are new categories in the new data that are not present in the base dataframe. In that case, they will take the value zero because they are contained in the automatized process. Anyway, it is a minor noise that will be automatically corrected in the weekly process.

## 6. ROBUST SCALE

In machine-learning it is a must to standardize datasets. Several algorithms are based in Gaussian distributions assumptions. Others are very sensitive to scale effects. Anyway, it is highly recommended to use some kind of standardization.

In practice, we tend to use standard normalization. However, we are trying to have a clean view of possible outliers. In this case, it is possible a mean approach is not the best option. Outliers, can often influence the sample mean/variance in a negative way. In consequence, we are going to use a Robust Scaler approach.

This scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

Centering and scaling happen independently on each feature (or each sample, depending on the axis argument) by computing the relevant statistics on the samples in the training set. Median and interquartile range are then stored to be used on later data using the transform method.

**def robust\_scale**(df:pd.DataFrame, quantile\_range=(25.0, 75.0)):  
 *"""  
 Scale features using statistics that are robust to outliers.  
 This Scaler removes the median and scales the data according to the quantile range   
 (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile)   
 and the 3rd quartile (75th quantile).* ***:return****: scaled df  
 """* **from** sklearn.preprocessing **import** RobustScaler  
  
 robust\_scaler = RobustScaler(quantile\_range=quantile\_range)  
  
 df\_cols = df.columns.values.tolist()  
 df\_cols.remove('id\_siniestro')  
  
 params = []  
  
 **for** i **in** df\_cols:  
 df[i] = robust\_scaler.fit\_transform(df[i])  
 center = robust\_scaler.center\_  
 scale = robust\_scaler.scale\_  
 params.append([str[i], center, scale])  
  
 params = pd.DataFrame(params, columns=['column\_name', 'center', 'scale'])  
  
 **return** df, params

We also get the parameters used in each feature. Basically, we are saving the center (that will be the median value) and the scale (that is the max value – median value).

It will permit us to compute the scale to the new incoming values. This is made in main.py module as follows:

# First we separate TEST values  
df\_base = df[df['TEST'] == 0]  
df\_test = df[df['TEST'] == 1]  
  
# then compute the robust scale for TEST=0  
df\_base, params = process\_utils.robust\_scale(df\_base, quantile\_range=(10.0, 90.0))  
  
# Finally we get the center and the scale from TEST=0 and apply to TEST=1  
columns\_to\_scale = params['column\_name'].tolist()  
**for** i **in** columns\_to\_scale:  
 center = params.loc[params['column\_name'] == i, 'center']  
 scale = params.loc[params['column\_name'] == i, 'scale']  
 df\_test[i] = (df\_test[i] - center) / scale  
  
df = pd.concat([df\_base, df\_test], axis=0)

We get the parameters and apply the scale formulation using:

## 7. PRINCIPAL COMPONENT ANALYSIS

PCA is used to decompose a multivariate dataset in a set of successive orthogonal components that explain a maximum amount of the variance.

We need to use PCA basically because we have too many dimensions. This could lead two major problems. First, several algorithms tend to be confused in high-dimensionality environments. Second, more dimensions are time-cost consuming (Third, with linear models is probably we have collinearity problems).

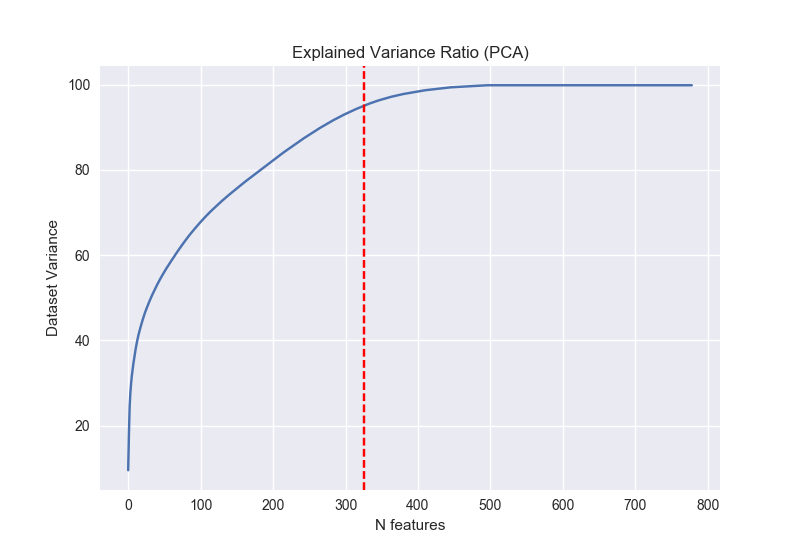
In conclusion, PCA is a linear dimensionality reduction process that use Singular Value Decomposition of the data to project it to a lower dimensional space.

Below, you can appreciate the code:

**def pca\_reduction**(df: pd.DataFrame, show\_plot=**False**, variance=95.00):  
 *"""  
 This automatically calcualte a PCA to df taking into account the 95% of the dataset explained variance* ***:param*** *show\_plot: Threshold Variance Plot* ***:param*** *variance: Dataset variance limit to consider in the PCA.* ***:return****: PCA df  
 """* **from** pre\_processing.feature\_selection **import** \*  
  
 siniestro\_df = df[['id\_siniestro']]  
 **del** df['id\_siniestro']  
  
 pca = PCA(whiten=**True**, svd\_solver='randomized')  
  
 pca.fit(df, axis=1)  
  
 cumsum = np.cumsum(np.round(pca.explained\_variance\_ratio\_, decimals=4) \* 100)  
  
 **if** show\_plot:  
 plot.plot(cumsum)  
 plot.show()  
 cumsum = list(cumsum)  
 var = [value **for** value **in** cumsum **if** value <= variance]  
 pca\_components = len(var)  
 print('PCA Components ', pca\_components)  
  
 pca = PCA(n\_components=pca\_components, whiten=**True**, svd\_solver='randomized')  
  
 pca.fit(df)  
 df = pca.fit\_transform(df)  
 df = pd.DataFrame(df)  
  
 df = pd.concat([df, siniestro\_df], axis=1)  
  
 **return** df

Our basic approach is to fit our dataset to the PCA transformation without assigning the Principal Components. What we do is to input a threshold of the variance dataset. Therefore the PCA will calculate how many components we need to get the 95% variance threshold limit. It is recommended between 95%-99% of the total variance. It is a trade-off between getting the total dataset variation and overfitting.

Finally we reduce the dataset to Components that explain the 95% of the variance. You can appreciate in the next Figure:



As you can see, we reach the threshold with 324 Components.

## 8. APPEND THE BLACKLIST

The following method is a simple one. It just append a column which match the sinister number with the sinister in the blacklist. If it exists, the new variable ‘FRAUDE’ is equal to one.

**def append\_blacklist**(df: pd.DataFrame):  
 *"""  
 It append the variable FRAUDE = 1 in the dataframe passed.* ***:return****: df + FRAUDE  
 """* **from** utils.read\_csv **import** ReadCsv  
 **from** utils.dataframe\_utils **import** DfUtils  
 **import** STRING  
  
 file\_blacklist\_resume = ReadCsv.load\_csv(STRING.blacklist\_processed\_resume)  
 df\_bl\_resume = DfUtils.processing\_file(file\_blacklist\_resume, delimiter=';')  
 df\_bl\_resume['ID\_SINIESTRO'] = df\_bl\_resume['ID\_SINIESTRO'].map(int)  
 df['id\_siniestro'] = df['id\_siniestro'].map(int)  
 df = pd.merge(df, df\_bl\_resume, how='left', left\_on='id\_siniestro', right\_on='ID\_SINIESTRO')  
 df['FRAUDE'] = pd.Series(0, index=df.index)  
 df.loc[df['ID\_SINIESTRO'].notnull(), 'FRAUDE'] = 1  
 **del** df\_bl\_resume  
 **del** df['ID\_SINIESTRO']  
  
 **return** df

**However, it is important to remark that Robust Scale does not correct the Scale Effects. Therefore, PCA can be very sensible to the data. We have to do a double standardization. First, with Robust Scale, and then with a traditional Standard Normalization.**

The previous code will be useful to compare the performance of the Machine Learning algorithms.

## 9. CORRELATIONS (OPTIONAL)

This step is divided into two methods. They work in a similar way, the difference is that the first take a list of variables and analyze the correlation between each of them. The second, return the correlations between every variable in the dataset or one specific column against every column in the dataset.

**def correlation\_get\_column**(df: pd.DataFrame, columns: list =[], output\_file=**False**, show\_plot=**False**):  
 *"""  
 We get correlations for specific columns selected in 'columns' list.* ***:param*** *columns: Columns of the df we want to correlate.* ***:param*** *output\_file: If we want a csv with the output.* ***:param*** *show\_plot: If we want a Heat Map.* ***:return****: df subset correlations.  
 """* **import** seaborn **as** sns  
 index = []  
 **for** i **in** columns:  
 index\_i = df.columns.get\_loc(i)  
 index.append(index\_i)  
 corrmat = df.corr().iloc[:, index]  
 print(corrmat)  
  
 **if** output\_file:  
 corrmat.to\_csv('corrmat\_subset.csv', sep=';')  
  
 **if** show\_plot:  
 f, ax = plot.subplots(figsize=(12, 9))  
 sns.heatmap(corrmat, vmax=.8, square=**True**)  
 networks = corrmat.columns.values.tolist()  
 **for** i, network **in** enumerate(networks):  
 **if** i **and** network != networks[i - 1]:  
 ax.axhline(len(networks) - i, c="w")  
 ax.axvline(i, c="w")  
 f.tight\_layout()  
 plot.show()  
  
**def correlation\_get\_all**(df: pd.DataFrame, get\_all=**False**, get\_specific='FRAUDE', output\_file=**False**, show\_plot=**False**):  
 *"""  
 We get correlations for a specific column or the whole dataframe.* ***:param*** *get\_all: True if we want the whole dataframe correlation.* ***:param*** *get\_specific: Column of the df we want to correlate.* ***:param*** *output\_file: If we want a csv with the output.* ***:param*** *show\_plot: If we want a Heat Map.* ***:return****: df correlations.  
 """* **import** seaborn **as** sns  
 **if** get\_all:  
 corrmat = df.corr()  
 **else**:  
 index\_i = df.columns.get\_loc(get\_specific)  
 corrmat = df.corr().iloc[:, index\_i]  
 print(corrmat)  
  
 **if** output\_file:  
 corrmat.to\_csv('corrmat\_whole.csv', sep=';')  
  
 **if** show\_plot:  
 f, ax = plot.subplots(figsize=(12, 9))  
 sns.heatmap(corrmat, vmax=.8, square=**True**)  
 networks = corrmat.columns.values.tolist()  
 **for** i, network **in** enumerate(networks):  
 **if** i **and** network != networks[i - 1]:  
 ax.axhline(len(networks) - i, c="w")  
 ax.axvline(i, c="w")  
 f.tight\_layout()  
 plot.show()

As you can appreciate, both of them can optionally create a .csv with the correlations and produce a heatmap using Seaborn Library.

## 10. NORMAL/ANORMAL/NEW DATASET

The last step in the processing table section corresponds to the dataset split.

**def output\_normal\_anormal\_new**(df: pd.DataFrame, output\_file=**True**, input\_name\_file='raw\_file'):  
 *"""  
 It split the dataframe into three dataframes (normal, anormal, new) based on FRAUDE = (0,1) and New Sinister   
 Bottle. Also, if 'output\_file' = True, it creates a new version of the final table.* ***:param*** *output\_file: Boolean if it is necessary an output file* ***:param*** *input\_name\_file: The name of the output file.* ***:return****: Two dataframes based on normally anormally.  
 """* **import** STRING  
  
 # First we separete New sinister  
 new = df[df['TEST'] == 1]  
 df = df[df['TEST'] == 0]  
  
 **del** new['TEST']  
 **del** df['TEST']  
  
 df['FRAUDE'] = df['FRAUDE'].map(int)  
  
 anomaly = df[df['FRAUDE'] == 1]  
 normal = df[df['FRAUDE'] == 0]  
  
 print('anomaly shape ', anomaly.shape)  
 print('normal shape ', normal.shape)  
 string\_anomaly = 'anomaly\_' + input\_name\_file  
 print('output file anomaly ', string\_anomaly)  
 string\_normal = 'normal\_' + input\_name\_file  
 print('output file normal ', string\_normal)  
  
 **if** output\_file:  
 **import** os  
 parent\_dir = os.path.dirname(os.getcwd())  
 os.chdir(parent\_dir)  
 path = STRING.path\_final\_files  
  
 normal\_file = path + string\_normal + '.csv'  
 anormal\_file = path + string\_anomaly + '.csv'  
 new\_file = path + 'new\_sinister.csv'  
  
 anomaly.to\_csv(normal\_file, sep=';', index=**False**)  
 normal.to\_csv(anormal\_file, sep=';', index=**False**)  
 new\_file.to\_csv(new\_file, sep=';', index=**False**)  
  
 **return** normal, anomaly, new

Here we separate into three dataframes/files. We are going to have a normal Dataframe which contains uniquely the sinister that are not in the Blacklist database. And we are going to have an abnormal dataframe, which contains only the sinister involved in the Blacklist. This step will be important when we construct our Training, Valid, Test datasets in terms of balanced samples.

Also, we will have a new Dataframe which contains the processed new sinister. They will be the sinister we are going to calculate the final probabilities.

# RUNNING THE MODEL

In the following section, we are going to evaluate, to tune and to output the final probabilities of the model chosen.

## TRAIN-VALID-TEST

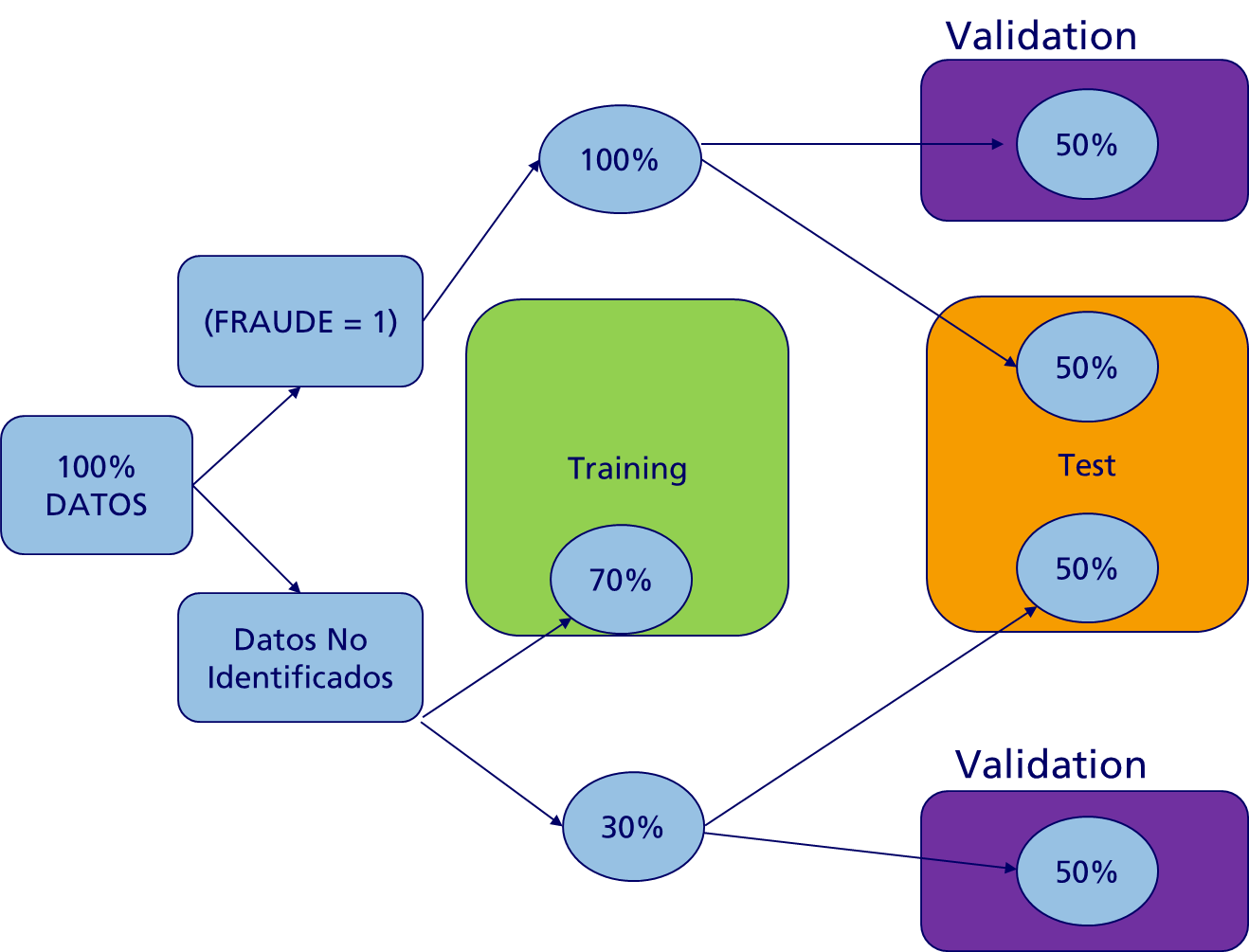
In this subsection, we are analyzing the created module train\_test\_utils. This module has several ways to split the dataset. However, in practice we are going to use just two methods:

* Training\_test\_valid
* Training\_test

Following we explain briefly everyone.

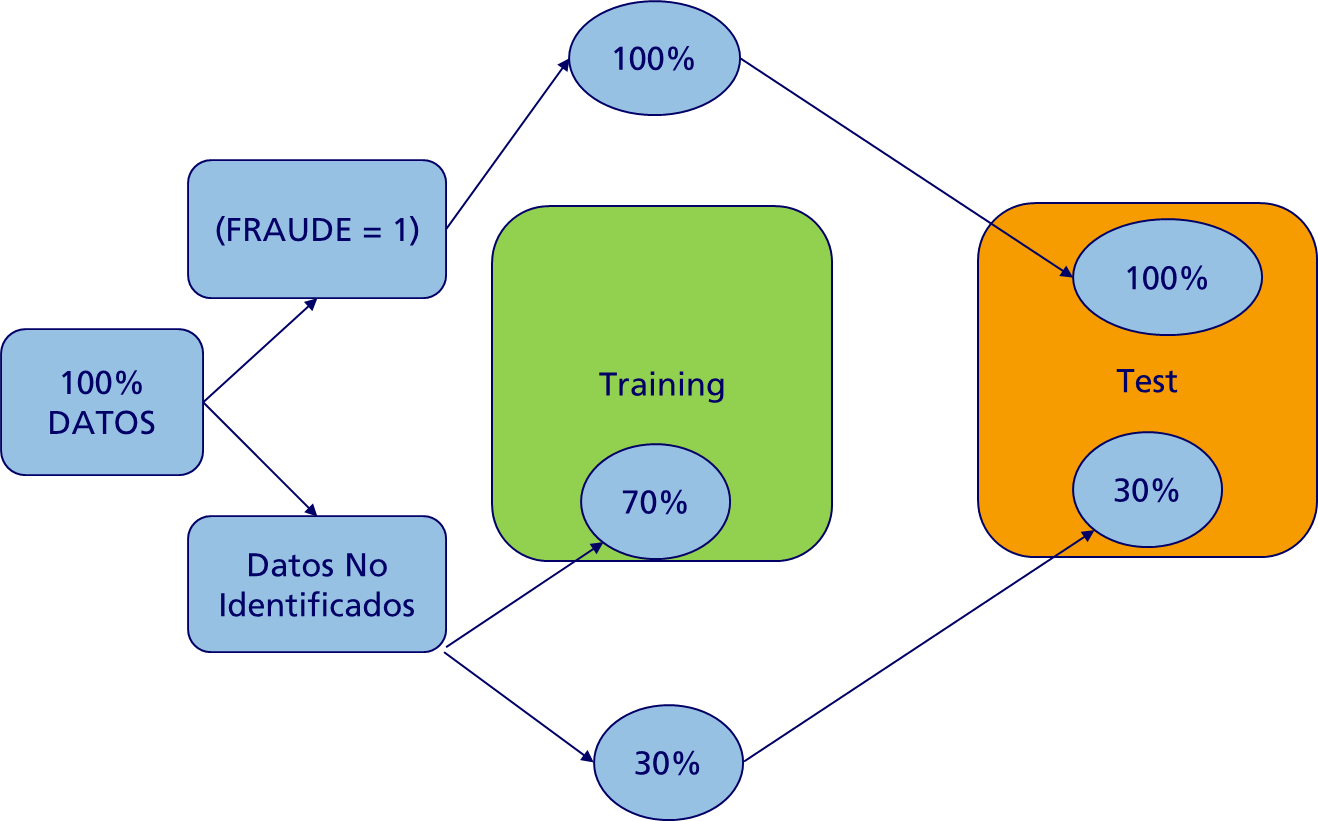
**def training\_test\_valid\_unbalanced**(normal\_file, anormal\_file):  
 *"""  
 Separate between training, test and valid using the next proportions:  
 Training 70%  
 Test 15%  
 Valid 15%  
 Also it keeps the same proportion between Fraud class inside Test an Valid.  
 However, it excludes every fraud claim in the Train Set.  
 """* normal = pd.read\_csv(normal\_file, delimiter=';')  
 anomaly = pd.read\_csv(anormal\_file, delimiter=';')  
  
 train, normal\_test, \_, \_ = train\_test\_split(normal, normal, test\_size=.3, random\_state=42)  
  
 normal\_valid, normal\_test, \_, \_ = train\_test\_split(normal\_test, normal\_test, test\_size=.5, random\_state=42)  
 anormal\_valid, anormal\_test, \_, \_ = train\_test\_split(anomaly, anomaly, test\_size=.5, random\_state=42)  
  
 train = train.reset\_index(drop=**True**)  
 valid = normal\_valid.append(anormal\_valid).sample(frac=1).reset\_index(drop=**True**)  
 test = normal\_test.append(anormal\_test).sample(frac=1).reset\_index(drop=**True**)  
  
 print('Train shape: ', train.shape)  
 print('Proportion os anomaly in training set: %.2f\n', train['FRAUDE'].mean())  
 print('Valid shape: ', valid.shape)  
 print('Proportion os anomaly in validation set: %.2f\n', valid['FRAUDE'].mean())  
 print('Test shape:, ', test.shape)  
 print('Proportion os anomaly in test set: %.2f\n', test['FRAUDE'].mean())  
  
 **return** train, valid, test

This method permits to separate between Train-Test-Valid. But, in the Training Set we do not have Fraud cases. We created this because another Semi-Supervised model approach was tried. Basically, the sketch is as follows:



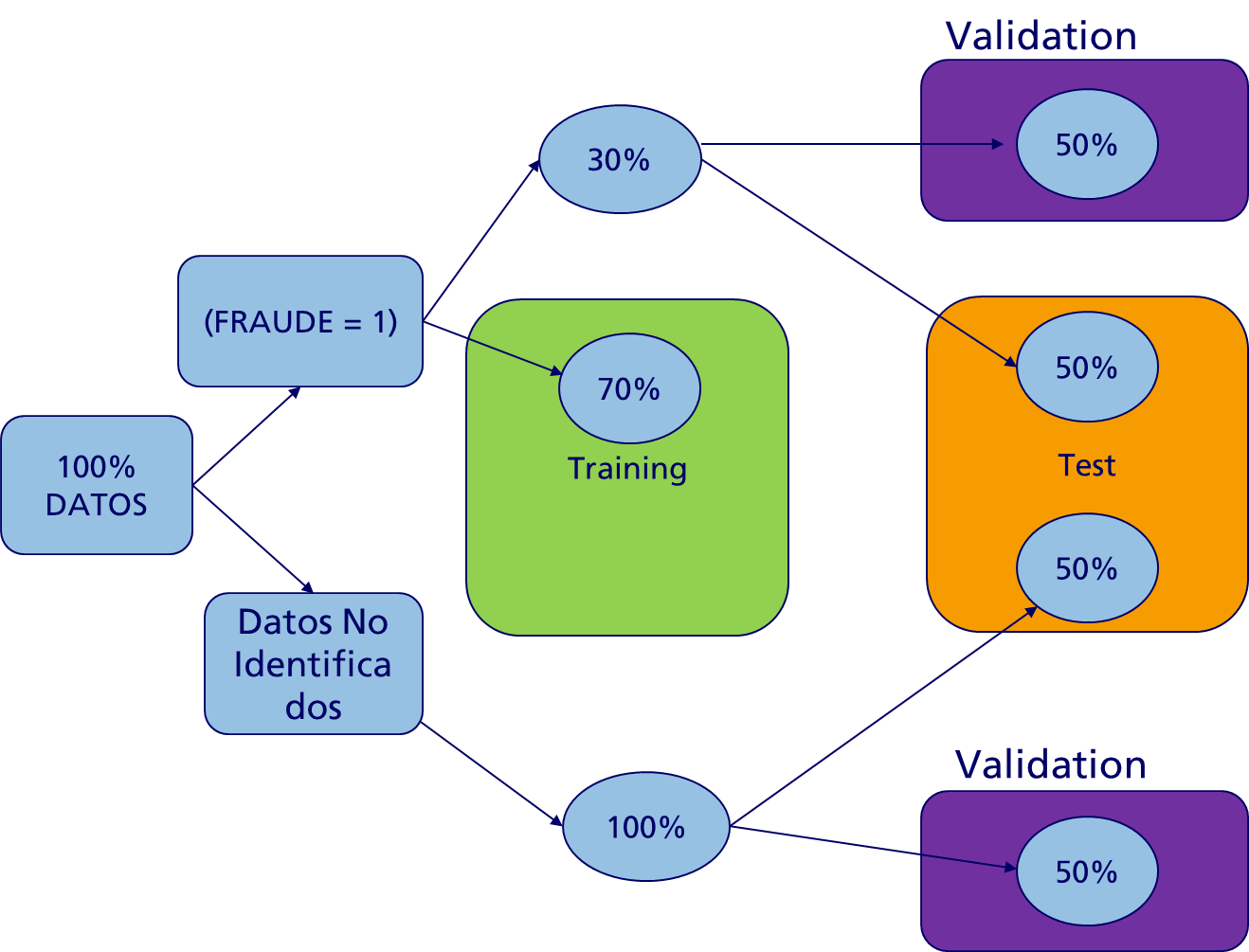
**def training\_test\_unbalanced**(normal\_file, anormal\_file):  
 *"""  
 Separate between training and Test using the next proportions:  
 Training 70%  
 Test 30%  
 Also it keeps the same proportion between Fraud class.  
 However, it excludes every fraud claim in the Train Set.  
 """* normal = pd.read\_csv(normal\_file, delimiter=';')  
 anormal = pd.read\_csv(anormal\_file, delimiter=';')  
  
 train\_normal, test\_normal, \_, \_ = train\_test\_split(normal, normal, test\_size=.3, random\_state=42)  
 train\_anormal, test\_anormal, \_, \_ = train\_test\_split(anormal, anormal, test\_size=.3, random\_state=42)  
  
 train = train\_normal.append(train\_anormal).sample(frac=1).reset\_index(drop=**True**)  
 test = test\_normal.append(test\_anormal).sample(frac=1).reset\_index(drop=**True**)  
  
 print('Train shape: ', train.shape)  
 print('Proportion os anomaly in training set: %.2f\n', train['FRAUDE'].mean())  
 print('Valid shape: ', test.shape)  
 print('Proportion os anomaly in validation set: %.2f\n', test['FRAUDE'].mean())  
  
 **return** train, test

The previous method is similar to the former one. The difference is that it produces only test and training samples:



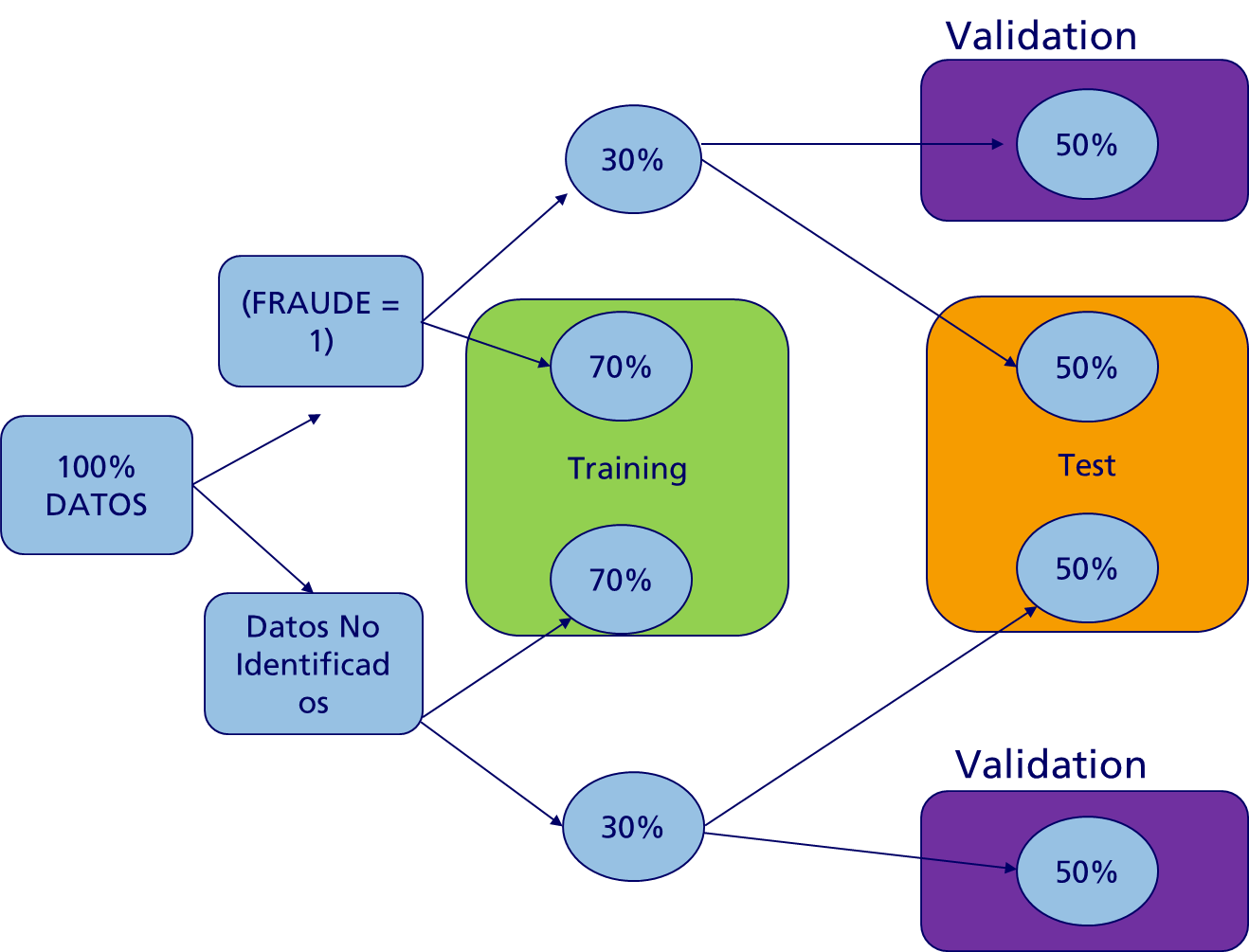
**def inverse\_training\_test\_valid**(normal\_file, anormal\_file):  
 *"""  
 Separate between training, test and valid using the next proportions:  
 Training 70%  
 Test 15%  
 Valid 15%  
 The difference is that it only includes anomaly cases inside the Train Set.  
 """* normal = pd.read\_csv(normal\_file, delimiter=';')  
 anomaly = pd.read\_csv(anormal\_file, delimiter=';')  
  
 train, anormal\_test, \_, \_ = train\_test\_split(anomaly, anomaly, test\_size=.5, random\_state=42)  
  
 anormal\_valid, anormal\_test, \_, \_ = train\_test\_split(anormal\_test, anormal\_test, test\_size=.5, random\_state=42)  
 normal\_valid, normal\_test, \_, \_ = train\_test\_split(normal, normal, test\_size=.5, random\_state=42)  
  
 train = train.reset\_index(drop=**True**)  
 valid = anormal\_valid.append(normal\_valid).sample(frac=1).reset\_index(drop=**True**)  
 test = anormal\_test.append(normal\_test).sample(frac=1).reset\_index(drop=**True**)  
  
 print('Train shape: ', train.shape)  
 print('Proportion os anomaly in training set: %.2f\n', train['FRAUDE'].mean())  
 print('Valid shape: ', valid.shape)  
 print('Proportion os anomaly in validation set: %.2f\n', valid['FRAUDE'].mean())  
 print('Test shape:, ', test.shape)  
 print('Proportion os anomaly in test set: %.2f\n', test['FRAUDE'].mean())  
  
 **return** train, valid, test

This method follow the same way as the former ones. It also produces training-test-valid samples but in the Training Set we only includes the Fraud class.

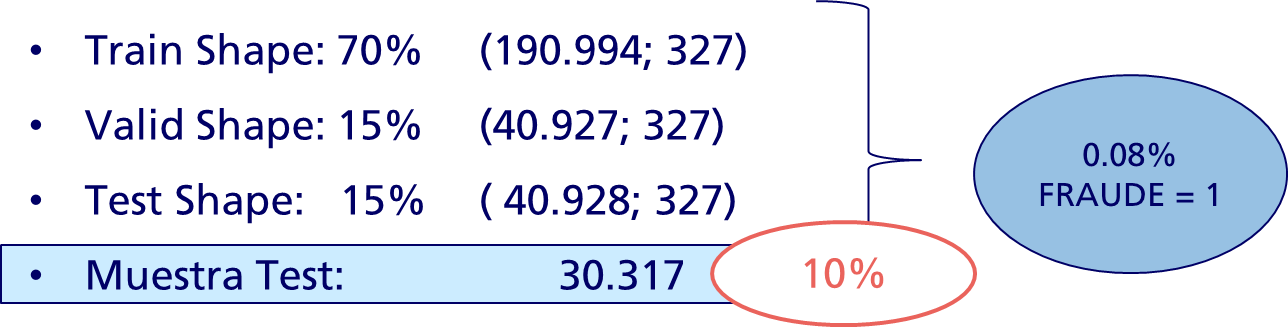


**def training\_test\_valid**(normal, anormal):  
 *"""  
 Separate between training, test and valid using the next proportions:  
 Training 70%  
 Test 15%  
 Valid 15%  
 Here, we include in the Training Set either normal cases and anormal cases using the proportions  
 derivated from the original distribution.  
 Then we split between Test and Valid using the same original proportions.  
 """* normal = pd.read\_csv(normal, delimiter=';')  
 anomaly = pd.read\_csv(anormal, delimiter=';')  
  
 normal\_train, normal\_test, \_, \_ = train\_test\_split(normal, normal, test\_size=.3, random\_state=42)  
 anormal\_train, anormal\_test, \_, \_ = train\_test\_split(anomaly, anomaly, test\_size=.3, random\_state=42)  
 normal\_valid, normal\_test, \_, \_ = train\_test\_split(normal\_test, normal\_test, test\_size=.5, random\_state=42)  
 anormal\_valid, anormal\_test, \_, \_ = train\_test\_split(anormal\_test, anormal\_test, test\_size=.5, random\_state=42)  
  
 train = normal\_train.append(anormal\_train).sample(frac=1).reset\_index(drop=**True**)  
 valid = normal\_valid.append(anormal\_valid).sample(frac=1).reset\_index(drop=**True**)  
 test = normal\_test.append(anormal\_test).sample(frac=1).reset\_index(drop=**True**)  
  
 print('Train shape: ', train.shape)  
 print('Proportion os anomaly in training set: %.2f\n', train['FRAUDE'].mean())  
 print('Valid shape: ', valid.shape)  
 print('Proportion os anomaly in validation set: %.2f\n', valid['FRAUDE'].mean())  
 print('Test shape:, ', test.shape)  
 print('Proportion os anomaly in test set: %.2f\n', test['FRAUDE'].mean())  
  
 **return** train, valid, test

training\_test\_valid method is one of our main methods used in the estimations. We divide between a train-test-valid samples trying to keep the real fraud class distribution.



This keep the same proportion of fraud in the Training, Test and Valid datasets.



But, why is important a Valid Sample? Because we need it to tune our probabilistic threshold.

**def training\_test**(normal, anormal):  
 *"""  
 Separate between training, test and valid using the next proportions:  
 Training 70%  
 Test 15%  
 Valid 15%  
 Here, we include in the Training Set either normal cases and anormal cases using the proportions  
 derivated from the original distribution.  
 Then we split between Test and Valid using the same original proportions.  
 """* normal = pd.read\_csv(normal, delimiter=';')  
 anomaly = pd.read\_csv(anormal, delimiter=';')  
  
 normal\_train, normal\_test, \_, \_ = train\_test\_split(normal, anormal, test\_size=.3, random\_state=42)  
 anormal\_train, anormal\_test, \_, \_ = train\_test\_split(anomaly, anomaly, test\_size=.3, random\_state=42)  
  
 train = normal\_train.append(anormal\_train).sample(frac=1).reset\_index(drop=**True**)  
 test = normal\_test.append(anormal\_test).sample(frac=1).reset\_index(drop=**True**)  
  
 print('Train shape: ', train.shape)  
 print('Proportion os anomaly in training set: %.2f\n', train['FRAUDE'].mean())  
  
 print('Test shape:, ', test.shape)  
 print('Proportion os anomaly in test set: %.2f\n', test['FRAUDE'].mean())  
  
 **return** train, test

The last code is very similar. The unique difference is that it splits between Train and Test. Once we have our parameters, it is important to return to the standard train-test approach to keep the maximum possible information.

## MODEL APPLICATION

When we are looking for outliers, what we are looking is the existence of clusters. That is, try to separate normal, noise and abnormal data. This implies not only constructing a model, also a process to analyze the existence of clusters and its number, and then the internal and external validation of the clusters.

In Outlier detection we try to separate our observations between regulars and outliers. Yet, we do not have a clean regular data representing the population.

One common way is to assume that regular data come from a known distribution (as a Gaussian). From this assumption we can define the shape of data, and define outliers as observations which stand far enough from the fit shape.

In that case, we can use covariance.EllipticEnvelope from sklearn. This fits a robust covariance estimate to the data, and thus fits an ellipse to the central data points, ignoring points outside the central mode.

Assuming that the inliers are Gaussian, it will estimate the inlier location and covariance in a robust way. The Mahalanobis distances obtained from this estimate is used to derive a measure of outlyingness.

Elliptic Envelope is an object for detecting outliers in a Gaussian distributed dataset. The contamination option, is the amount of contamination of the data set (the proportion of outliers in the data set). It needs the definition of a covariance, because, as explained before, it is part of the definition of distance. We will use the Empirical Covariance that is highly sensitive to outliers and a robust covariance (MCD) that guarantee that the estimation is resistant to erroneous observations.

If we have a high-dimensional datasets we can use random forests to detect outliers. The Isolation Forest algorithm (Liu, et al, 2008) isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximium and minimum values of the selected feature.

Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.

This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.

Random partitioning produces noticeably shorter paths for anomalies. Hence, when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies.

Basically, we apply the same logic of Decision Trees. A tree structure is constructed to isolate every minority. That values that are more susceptible to be isolated (important anomalies) are discarded more near the root. Normal points tend to be isolated to the deeper on the tree. With this behavior we can construct a score. Isolation Forest, build finally an esamble of i Trees. In conclusion, anomalies will be that data points that have short AVERAGE path lengths on the iTrees.

In practice, it can be used as a supervised learning (training data has labels 1 and -1) or as unsupervised. In this case, we will apply as unsupervised, so the algorithm will define if it is a 1 (normal data) or -1 (abnormal data).

But why is semi-supervised? Empirically because in general, in fraud detection there exists a great problem of misclassification claims.

Technically, because first we apply an unsupervised model, and then we test using a Supervised Model. This permit us to obtain a gain in terms of accuracy and the possibility to use partial information. Trivedi, Pardos and Hefferman (2014) has shown that the use a predictor in conjunction with clustering improved the prediction accuracy in several datasets. They conclude that this methodology provides a novel and useful source of variance in the prediction process.

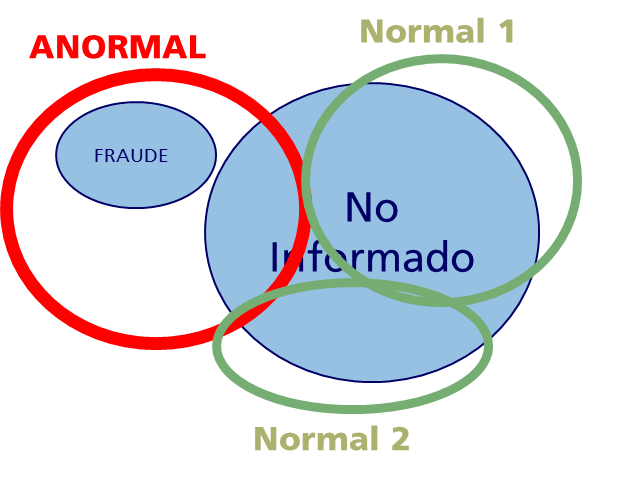
The Semi-supervised algorithm permits to reduce data in clusters leading with high-dimensionality problems. After the clusters are formed, they are giving to a supervised training algorithm.

We tried several combination of unsupervised-supervised techinques. You can see some of this results in the Annex section.

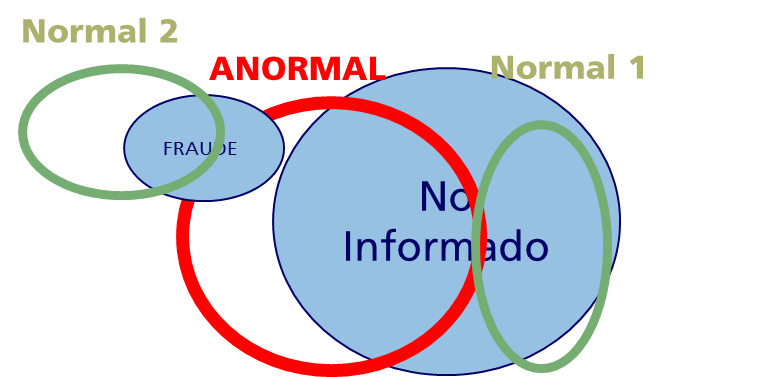
## UNSUPERVISED MODEL SELECTION

### FRAUD SCORE

The main important problem in the unsupervised model is that we actually do not know which the real classes of the points are. We just have partial information about fraud cases. But, which is the acceptable threshold for not informed cases? When we calculate unsupervised models we are reducing the dimensions to clusters. Some of them will return several categories where is probable we have both types of classes inside (fraud and not informed). Intuitevely, we want that the revealed fraud group is concentrated in a cluster. And some non revealed cases are with them. Something like this:

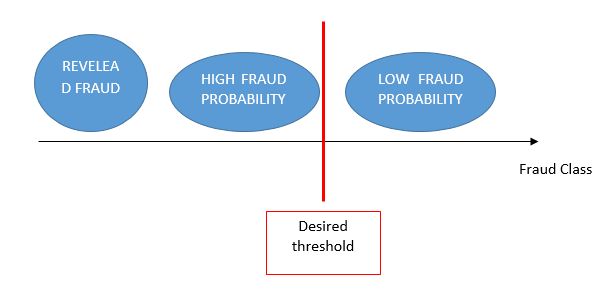


And we want to avoid cases where anormal cases and normal cases are uniformely distributed between groups. Something like this:



However, at some extent, we have to put a limit. Which is the quantity of ‘normal’ cases we want to accept as fraud class?

We can draw a line on tunning parameters between accept only cases of revealed fraud as fraud and to accept every case as fraud. We are basically in the middle. But the intuition says that we have to be more approximated to the low threshold. That is, we prefer to get first the revealed fraud as fraud class. It is something like this:



But again, we do not know which the correct limit is. Therefore, we created an experimental method that will assign a score to help us to take the decision.

This method is included in fraud\_score.py module in the utils package. The Fraud Score is calculated as a combination between two interrelated scores. We call them F1 and F2.

#### F1 SCORE

F1 Score calculates which is the probability of revealed fraud point to belong to the group J. And this probability is weighted by the total amount of fraud points in that group over the total cases of revealed fraud.

Mathematically this is:

Where is the fraud revealed point in the cluster group j. is the total amount of points in the group J. And is the total amount of revealed fraud cases.

Basically, we are calculating the probability of a point to belong to a specific group j () and we are weighting this values by the participation of that points in revealed fraud group.

Our objective will be to maximize this function. Basically, that is to get assigned revealed fraud in the same groups. In the limit F1=1 implies that in the j groups we have only revealed fraud points. However, that is something we do not want. Therefore, we have to equilibrate this function with another function.

#### F2 SCORE

F2 is the inverse case of F1. Here we calculate the probability of non revealed fraud to belong to group J.

And the objective is the same as F1. To cluster this group without assigning revealed fraud to that clusters.

#### FRAUD SCORE

Individually, maximizing F1 and F2 take us to unwanted situation. Basically, they are trying to be mutually splitted.

However, you can notice that . Therefore, when maximizing one we are minimizing the other. Individually, this leads to uniquely clusterization. But if we maximize both together, this implies a trade-off between them, a trade-off where we can choose. Moreover, as we pointed before, we actually want to maximize F1 subject to F2. Consequently, the Fraud Score is constructed as follows:

With β≥1. If β = 1, F1 and F2 will have the same weight. But if we assing β>1, this will reduce the charge of F2.

In conclusion, with this FS we have an objective parameter to tunne the unsupervised model. Basically, we are going to maximize FS. The only decision that remains for us is determine the relevance of β.

#### PRACTICAL EXAMPLE

Imagine we have the following output from an unsupervised model:

|  |  |
| --- | --- |
| **class** | **label** |
| 0 | 1 |
| 0 | 2 |
| 0 | 3 |
| 0 | 1 |
| 1 | 2 |
| 1 | 2 |
| 1 | 3 |
| 0 | 3 |
| 0 | 3 |
| 0 | 2 |
| 0 | 1 |
| 0 | 3 |
| 1 | 2 |
| 0 | 1 |
| 1 | 2 |

Class represent fraud (=1) and non-identified (=0). The output label is the clustering labels. As you can see we have just 33% of detected fraud. If we group the class by the clusters:

|  |  |  |  |
| --- | --- | --- | --- |
| **label** | **class** | **subtotal\_class** | **total** |
| 1 | 0 | 4 | 4 |
| 1 | 1 | 0 | 4 |
| 2 | 0 | 2 | 6 |
| 2 | 1 | 4 | 6 |
| 3 | 0 | 4 | 5 |
| 3 | 1 | 1 | 5 |

As you can see, the fraud class tend to be assigned to the second cluster.

First we calculate F1 using the following method:

**def f1**(df: pd.DataFrame, class\_column:str, label\_column:str):  
  
 df = df.groupby([label\_column, class\_column]).size().reset\_index(drop=**False**)  
 df.columns = [label\_column, class\_column, 'subtotal']  
  
 df\_total = df.groupby([label\_column], as\_index=**False**).sum()  
 df\_total = df\_total[[label\_column, 'subtotal']]  
 df\_total.columns = [label\_column, 'total']  
  
 df = pd.merge(df, df\_total, on = label\_column, how='left')   
 f1\_df = df[df['class'] == 1]  
  
 f1\_df['weight'] = pd.Series(df['subtotal'] / df['total'], index=f1\_df.index)  
  
 f1\_df['weight\*x'] = pd.Series(f1\_df['weight']\*f1\_df['subtotal'], index=f1\_df.index)  
  
 f1 = sum(f1\_df['weight\*x'].values) / sum(f1\_df['subtotal'].values)  
  
 **return** f1

Basically it implies the next formulation:

Then we calculate F2 using a similar formulation:

**def f2**(df: pd.DataFrame, class\_column: str, label\_column: str):  
 df = df.groupby([label\_column, class\_column]).size().reset\_index(drop=**False**)  
 df.columns = [label\_column, class\_column, 'subtotal']  
  
 df\_total = df.groupby([label\_column], as\_index=**False**).sum()  
 df\_total = df\_total[[label\_column, 'subtotal']]  
 df\_total.columns = [label\_column, 'total']  
  
 df = pd.merge(df, df\_total, on=label\_column, how='left')  
  
 f2\_df = df[df['class'] == 0]  
  
 f2\_df['weight'] = pd.Series(df['subtotal'] / df['total'], index=f2\_df.index)  
  
 f2\_df['weight\*x'] = pd.Series(f2\_df['weight'] \* f2\_df['subtotal'], index=f2\_df.index)  
  
 f2 = sum(f2\_df['weight\*x'].values) / sum(f2\_df['subtotal'].values)  
  
 **return** f2

Which it implies:

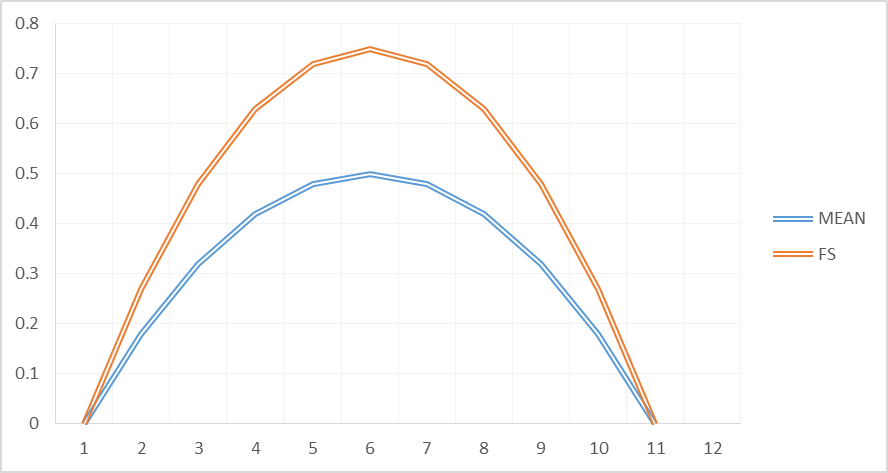
As you can appreciate, F1 gives worst results. This is because its core group (group 2) it is quite contaminated (66% of observations are actually fraud). This effect represents 93% of the total effect. The effect to mismatch the core group (1/5) is negligible. This reveals that the important is to have constructed a strong core group.

This conclusion is notorious in F2. Non-identified classes are very robust in two groups (1 and 3).

If we calculate the FS with a β=2 (balanced F1 and F2) we obtain:

Which is a value near 0.5733. This not implies that we are paying attention to the F1. This formula permits to balance the results, given more importance to the lower score. Indicating that we want both good and balanced scores. It is not the same to have F1= 0, F2=1 that F1=0.5, F2=0.5. Former will return an FS=0. Let compares the mean versus FS:

|  |  |  |  |
| --- | --- | --- | --- |
| **f1** | **f2** | **MEAN** | **FS** |
| 0 | 1 | 0.5 | 0 |
| 0.1 | 0.9 | 0.5 | 0.18 |
| 0.2 | 0.8 | 0.5 | 0.32 |
| 0.3 | 0.7 | 0.5 | 0.42 |
| 0.4 | 0.6 | 0.5 | 0.48 |
| 0.5 | 0.5 | 0.5 | 0.5 |
| 0.6 | 0.4 | 0.5 | 0.48 |
| 0.7 | 0.3 | 0.5 | 0.42 |
| 0.8 | 0.2 | 0.5 | 0.32 |
| 0.9 | 0.1 | 0.5 | 0.18 |
| 1 | 0 | 0.5 | 0 |

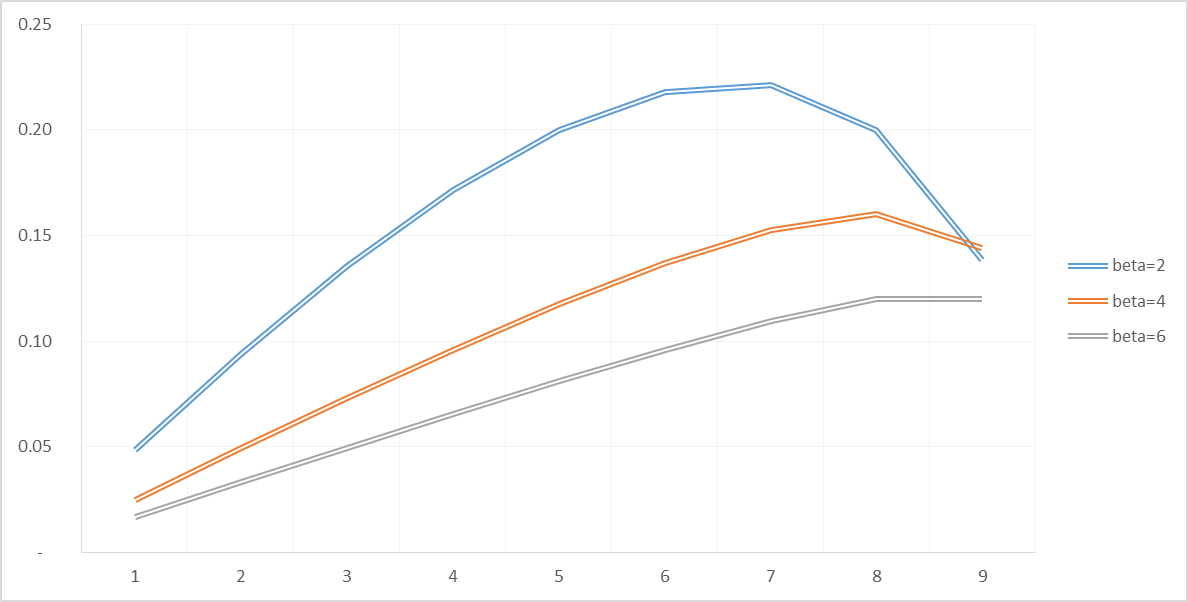


As you can see, it is the same unbalanced scores as balanced for the mean score. FS penalized unbalanced. That is why we obtain different results with the same proportions.

However, we also can adjust which is the relevance we give to each group. If we put higher β, we are going to penalize the results to F2, and viceversa.

What happen if we choose β>2?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **f1** | **f2** | **beta=2** | **beta=4** | **beta=6** |
| 0.1 | 0.9 | 0.05 | 0.02 | 0.02 |
| 0.2 | 0.8 | 0.09 | 0.05 | 0.03 |
| 0.3 | 0.7 | 0.14 | 0.07 | 0.05 |
| 0.4 | 0.6 | 0.17 | 0.10 | 0.07 |
| 0.5 | 0.5 | 0.20 | 0.12 | 0.08 |
| 0.6 | 0.4 | 0.22 | 0.14 | 0.10 |
| 0.7 | 0.3 | 0.22 | 0.15 | 0.11 |
| 0.8 | 0.2 | 0.20 | 0.16 | 0.12 |
| 0.9 | 0.1 | 0.14 | 0.14 | 0.12 |



As you can see, we have two effects. First, while F1 is increasing, FS also is increasing (although F2 is decreasing to the same speed). But this effect present in the balanced case, now it extends further. Just when we are on F1=0.7 the balanced effect tends to revert the situation. The second effect is that the score curve is shifted downward. FS now is demanding stringlier that F1 has to be higher. And while beta higher, more stringler.

#### MACHINE LEARNING IMPORTANCE

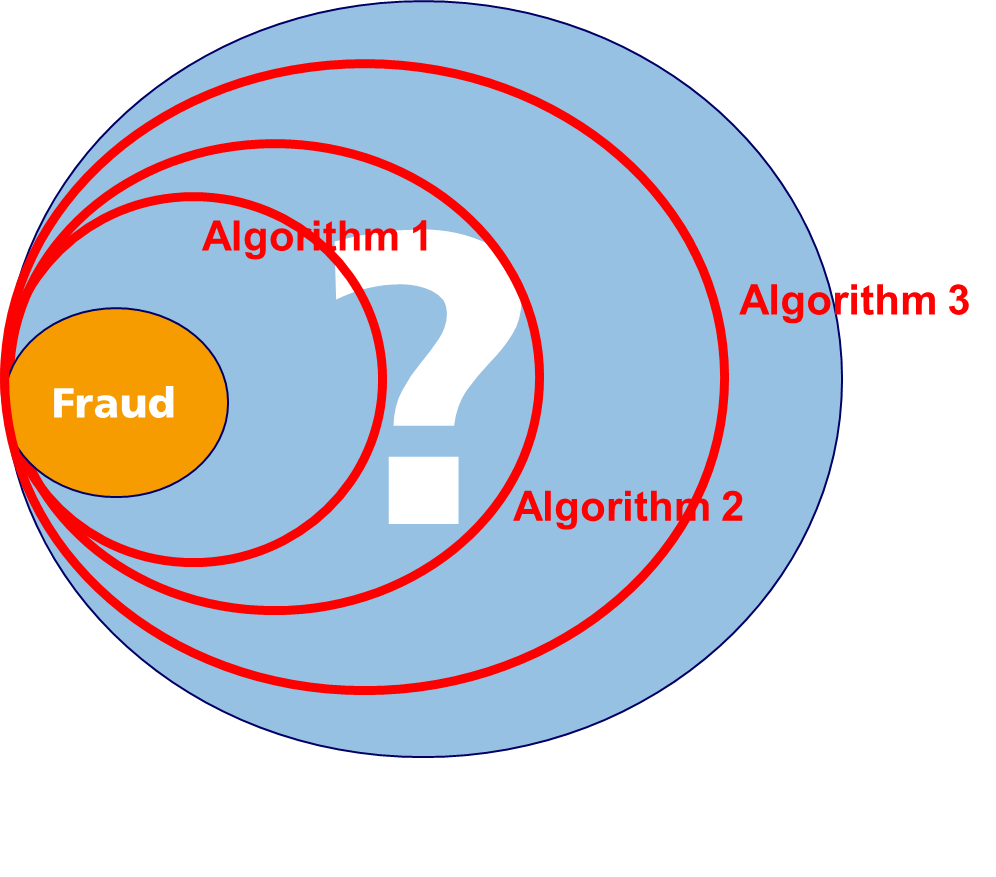
**It is important to remark, that everytime we have more information about Fraud Cases, this threshold will improve. Therefore, this is the core where the retroalimentative process of Machine Learning works.**

What that means?

We are going to explain this schematically. Basically, we start with an unknown distribution where we know some datapoints:

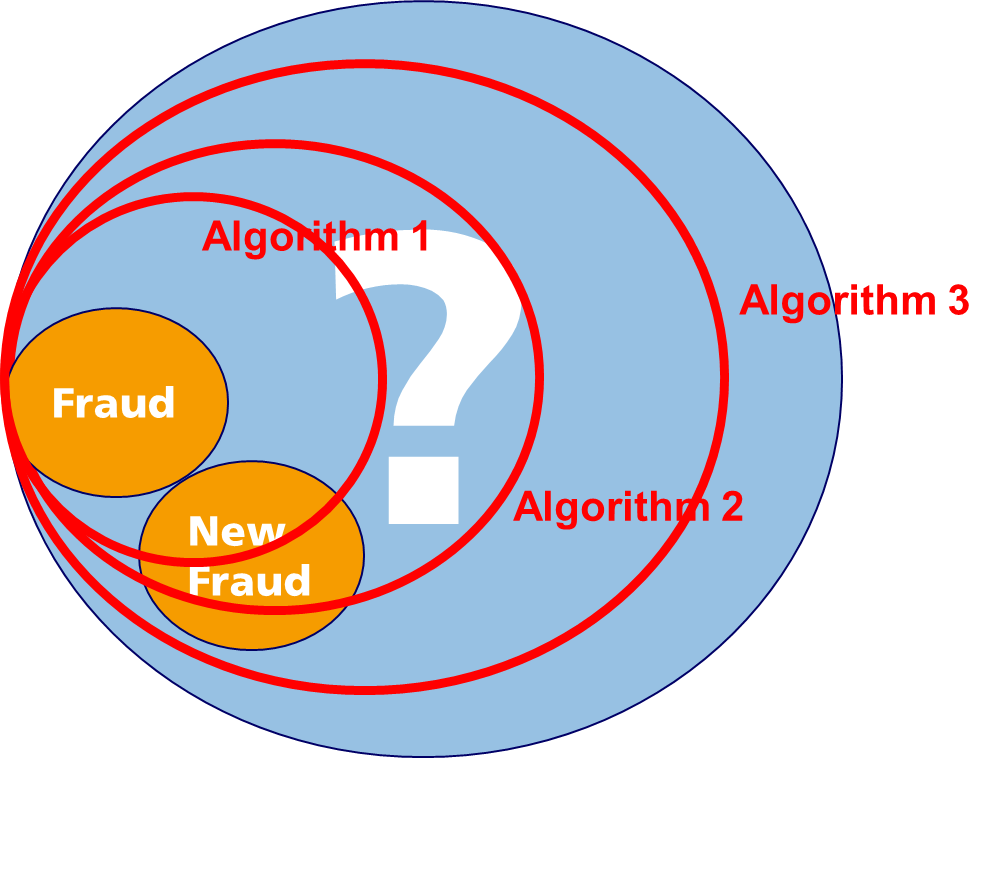


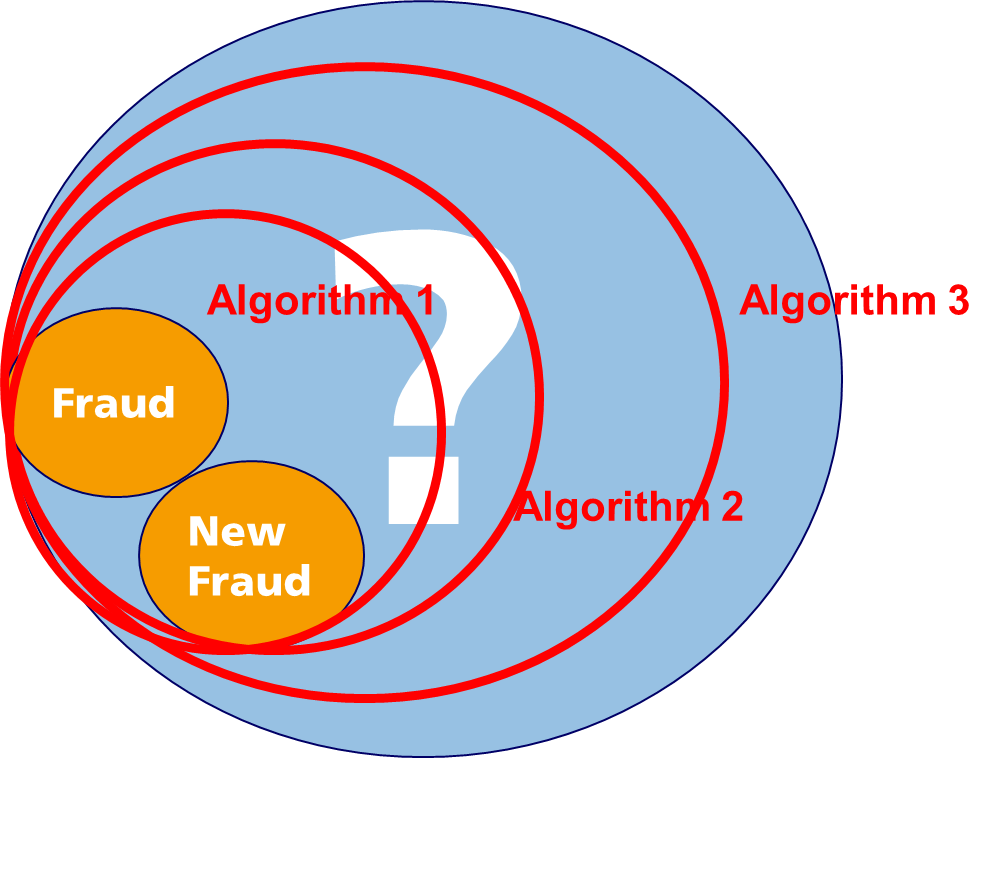
Our algorithms, using the propose Fraud Score, will be approximating the best they can to the fraud cases. Actually, they will try to fit to those, keeping a margin for not discovered cases. Like this:



Each algorithm can be a different model or can be the same algorithm with different parameters. On average, we are going to try almost 2.000-40.000 variation for each of the four models proposed. Therefore, this is very schematic.

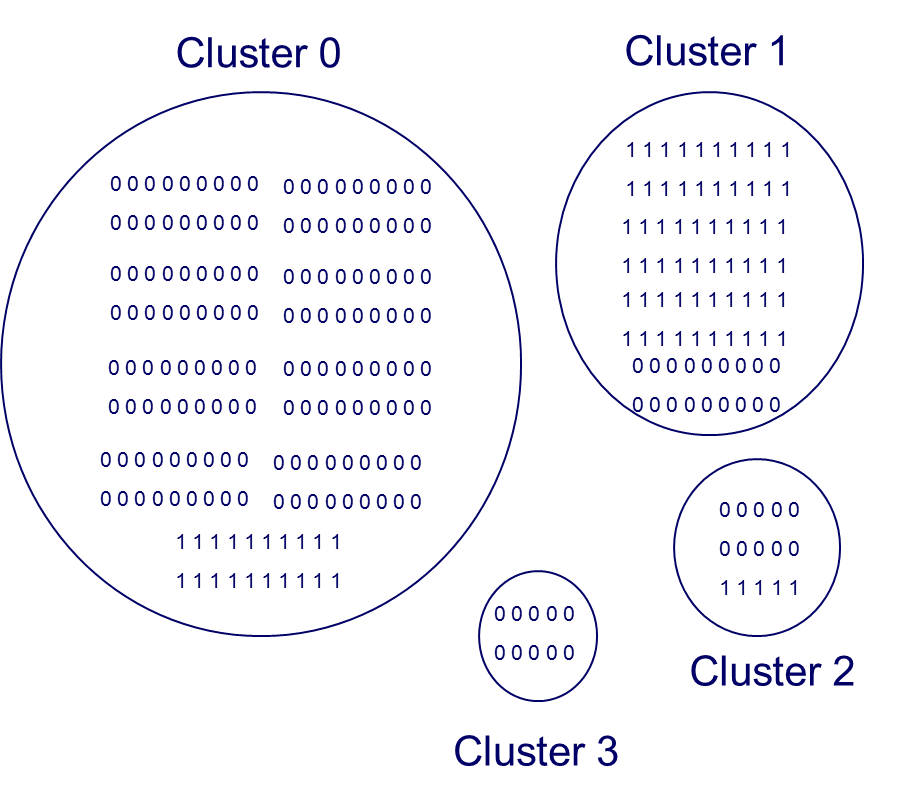
Now, if we get new information about the fraud cases, this algorithms will adjust to get the max Fraud Score. They will try to get a specific quantity of fraud cases. That is because they work with notions based on density or contamination. Therefore, they will change their shapes to adjust to this new information.



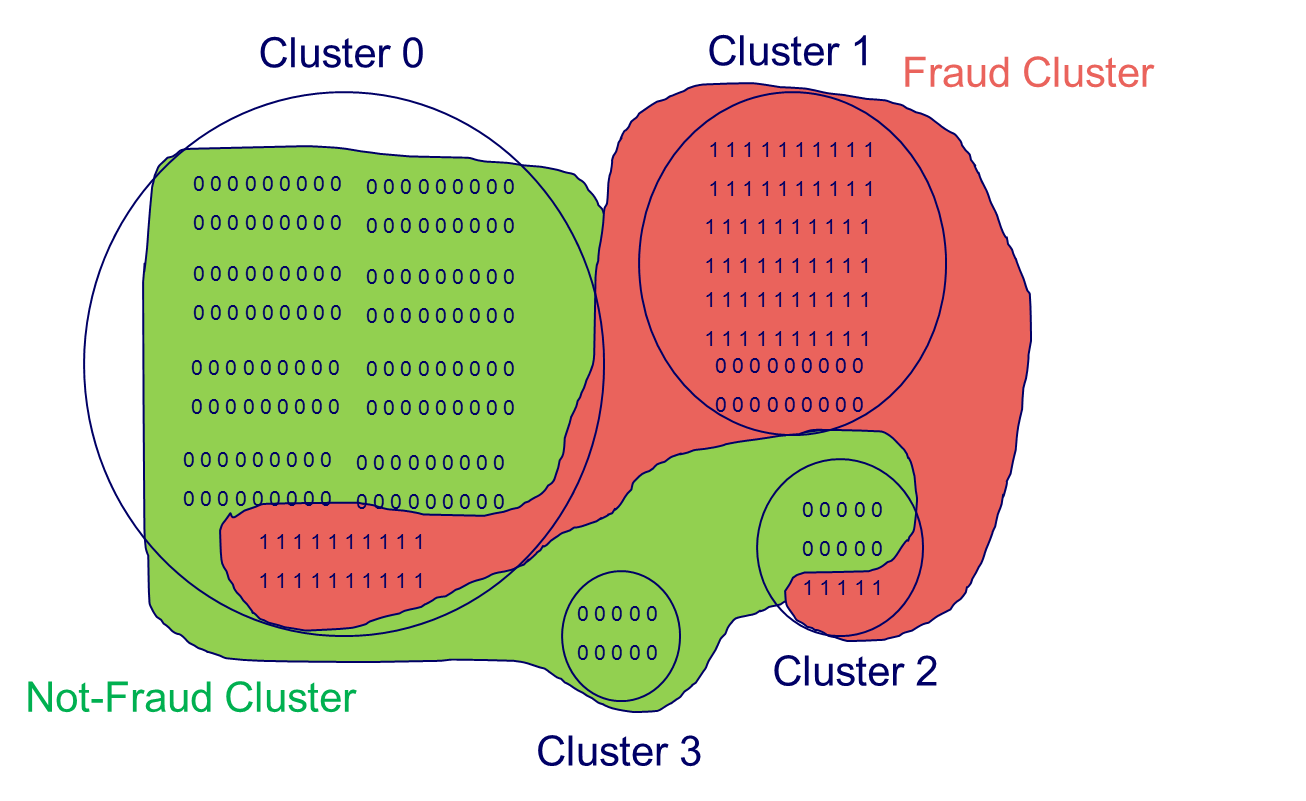


Once we have the max Fraud Score, we have to decide about the clusters generated. The problem is that is probably we will have several clusters. And also, they will have mixed type: fraud cases and not identified cases. Therefore, we have to decide which a Fraud Cluster is and which not.

As we have an unbalanced distribution, it is probably we have a cluster distribution like this:



Where 0 are not-identified cases and 1 are Fraud cases. We will use a parameter threshold that is initialized at 0.5. That is, if a cluster has more than 50% of Fraud cases, this cluster is a Fraud Cluster, otherwise, it is a Not-Fraud Cluster. Notice the difference. The Not-Fraud Cluster is not a non-identified cluster now. We are assuming that they are actually not fraud cases. That is the way in which the algorithm will evolve to semi-supervised. In the example above, we will separate the clusters between Fraud and not Fraud using the threshold suggested above.



Known fraud cases are obviously part of the Fraud cluster. But also, as cluster 1 has more than 50% of Fraud cases, we consider the whole cluster as Fraud cases. Remaining cases that do not belong to a danse fraud cluster are considered as not-fraud cases.

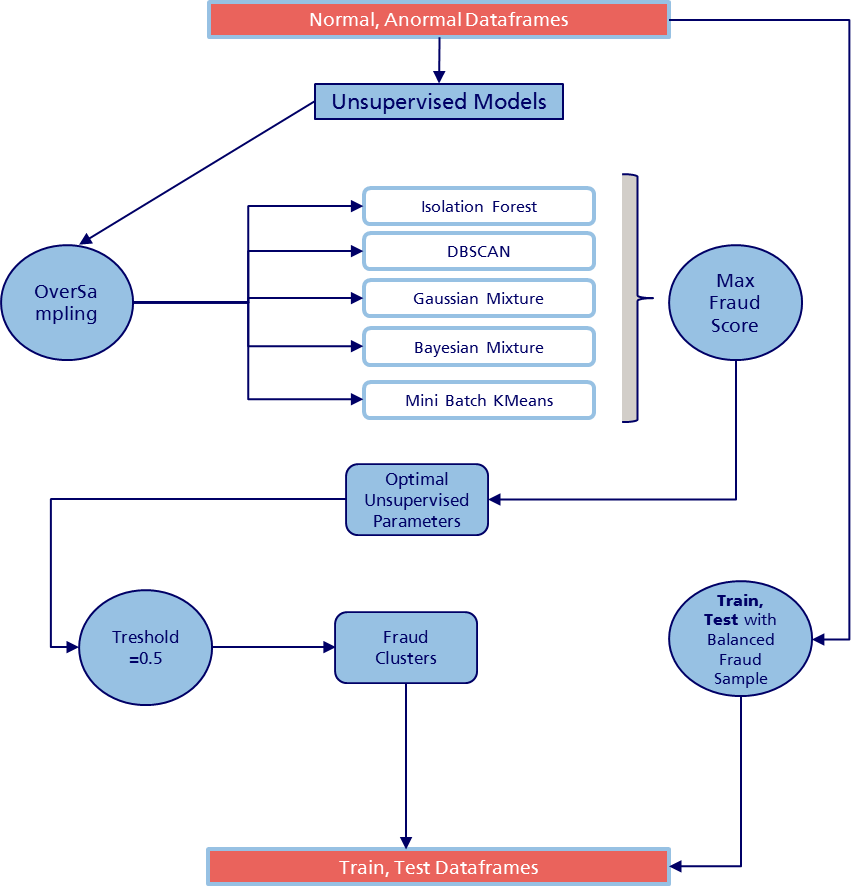
## UNSUPERVISED MODELS

We are using model\_evaluation.py module. This basically will apply different unsupervised models. This module calls other modules where several algorithms are developed. Therefore, we call those algorithms here and then, we iterate through their respective parameters. Basically, we are going to get the parameters that maximize our Fraud Score.

When we are looking for outliers, what we are looking is the existence of clusters. That is, try to separate normal, noise and abnormal data. This implies not only constructing a model, also a process to analyze the existence of clusters and its number, and then the internal and external validation of the clusters.

Next, we have to propose a model that can identify Outliers as a cluster.

We are going to follow the next sketch:



### OVERSAMPLING STRATEGY

We are trying to get separate clusters for outliers. But as we have highly dense data, and a highly dimension dataset, clusters methods can not understand the existence of outliers. Therefore, we have to balance the samples where we have some information, with the objective of give them representativity. We cannot use a forecast oversampling method, after all, we need to label by id\_sinister at which cluster belongs each sinister. For that reason what we do is to overweight our known fraud cases exactly the times where our distribution is 50/50.

Once we have our dataset, we iterate thousands of time trhough different models trying to optime our parameters (using the Fraud Score). Actually we have tried the following models: Isolation Forest, One-Class SVM, DBSCAN, HDBSCAN, Mean Shift, Mini Batch Kmeans, Agglomerative Clustering, Gaussian Mixture and Bayesian Mixture. However, finally we have to take only Isolation Forest, DBSCAN, Gaussian Mixture, Bayesian Mixture and Mini Batch Kmeans. This is because the other algoritmhs has a high computational cost. Regardless we have to iterate them several times. Althouth, we briefly explain them.

### MEAN SHIFT

MeanShift clustering aims to discover blobs in a smooth density of samples. It is a centroid based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region. These candidates are then filtered in a post-processing stage to eliminate near-duplicates to form the final set of centroids.

### DBSCAN

The DBSCAN algorithm (Ester et al, 1996) views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be any shape, as opposed to k-means which assumes that clusters are convex shaped.

The central component to the DBSCAN is the concept of core samples, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples).

Once the model are applied we have to validate the clusters as threefold: First, Internal Evaluation will permit to know if the clusters have good properties as a whole. Then, External Validation will analyze if each cluster specifically has good properties. Finally, we will determine if the number of clusters chosen is well defined.

It is a bit slow actually. It takes between 15 and 80 minutes each iteration. As the results does not change modifying the **eps** and the **min\_samples** parameters (in theory which size and shape the clusters).

Basically it gives as a F1 = 50.2% and F2 = 49.8% with a FS = 50.1%. It is clearly a bad result. Basically it says that the Fraud Cluster and the non-Fraud Cluster have the same proportions of both classes.

### LABEL PROPAGATION

Label propagation denotes a few variations of semi-supervised graph inference algorithms.

[Label Propagation](http://scikit-learn.org/stable/modules/generated/sklearn.semi_supervised.LabelPropagation.html#sklearn.semi_supervised.LabelPropagation) and [Label Spreading](http://scikit-learn.org/stable/modules/generated/sklearn.semi_supervised.LabelSpreading.html#sklearn.semi_supervised.LabelSpreading) differ in modifications to the similarity matrix that graph and the clamping effect on the label distributions. Clamping allows the algorithm to change the weight of the true ground labeled data to some degree. The [LabelPropagation](http://scikit-learn.org/stable/modules/generated/sklearn.semi_supervised.LabelPropagation.html#sklearn.semi_supervised.LabelPropagation) algorithm performs hard clamping of input labels, which means \alpha=0. This clamping factor can be relaxed, to say \alpha=0.2, which means that we will always retain 80 percent of our original label distribution, but the algorithm gets to change its confidence of the distribution within 20 percent.

[LabelPropagation](http://scikit-learn.org/stable/modules/generated/sklearn.semi_supervised.LabelPropagation.html#sklearn.semi_supervised.LabelPropagation) uses the raw similarity matrix constructed from the data with no modifications. In contrast, [LabelSpreading](http://scikit-learn.org/stable/modules/generated/sklearn.semi_supervised.LabelSpreading.html#sklearn.semi_supervised.LabelSpreading) minimizes a loss function that has regularization properties, as such it is often more robust to noise. The algorithm iterates on a modified version of the original graph and normalizes the edge weights by computing the normalized graph Laplacian matrix. This procedure is also used in [Spectral clustering](http://scikit-learn.org/stable/modules/clustering.html#spectral-clustering).

Label propagation models have two built-in kernel methods. Choice of kernel effects both scalability and performance of the algorithms. The following are available:

rbf (\exp(-\gamma |x-y|^2), \gamma > 0). \gamma is specified by keyword gamma.

knn (1[x' \in kNN(x)]). k is specified by keyword n\_neighbors.

The RBF kernel will produce a fully connected graph which is represented in memory by a dense matrix. This matrix may be very large and combined with the cost of performing a full matrix multiplication calculation for each iteration of the algorithm can lead to prohibitively long running times. On the other hand, the KNN kernel will produce a much more memory-friendly sparse matrix which can drastically reduce running times.

Our main problem here is that we do not have two classes to construct the Semi-supervised algorithm. We just know one class (Fraud Class), therefore it is a not applicable model.

### AGGLOMERATIVE CLUSTERING

The AgglomerativeClustering object performs a hierarchical clustering: each observation starts in its own cluster, and clusters are successively merged together. The linkage criteria determines the metric used for the merge strategy:

Ward minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function but tackled with an agglomerative hierarchical approach.

Maximum or complete linkage minimizes the maximum distance between observations of pairs of clusters.

Average linkage minimizes the average of the distances between all observations of pairs of clusters.

However is computationally expensive when no connectivity constraints are added between samples: it considers at each step all the possible merges.

### GAUSSIAN MIXTURE

The GaussianMixture object implements the expectation-maximization (EM) algorithm for fitting mixture-of-Gaussian models. It can also draw confidence ellipsoids for multivariate models, and compute the Bayesian Information Criterion to assess the number of clusters in the data.

The GaussianMixture comes with different options to constrain the covariance of the difference classes estimated: spherical, diagonal, tied or full covariance. We tried every of them and also we tried different tolerance ratios and number of clusters.

We obtain that we a Tied covariance, a tolerance of 0.29 and n\_clusters=5, our F1=95%, F2=95% and FS = 96.3%. Actually, an extraordinary result. However the time consuming remains a bit high (23 minutes each iteration).

### BAYESIAN MIXTURE

The BayesianGaussianMixture object implements a variant of the Gaussian mixture model with variational inference algorithms.

Variational inference is an extension of expectation-maximization that maximizes a lower bound on model evidence (including priors) instead of data likelihood. The principle behind variational methods is the same as expectation-maximization (that is both are iterative algorithms that alternate between finding the probabilities for each point to be generated by each mixture and fitting the mixture to these assigned points), but variational methods add regularization by integrating information from prior distributions. This avoids the singularities often found in expectation-maximization solutions but introduces some subtle biases to the model. Inference is often notably slower, but not usually as much so as to render usage unpractical.

As result we obtain 6 clusters with F1=96.5%, F2=96.4% and FS=96.5%. Actually, the best result we get.Also it takes 23 minutes.

### MINI BATCH KMEANS

Mini-Batch Kmeans is similar to Kmeans but also it reduces the computational time. As we need to reproduce this every day, we use this version that use random subset in each training iteration.

The MiniBatchKMeans is a variant of the KMeans algorithm which uses mini-batches to reduce the computation time, while still attempting to optimise the same objective function. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution. In contrast to other algorithms that reduce the convergence time of k-means, mini-batch k-means produces results that are generally only slightly worse than the standard algorithm.

The algorithm iterates between two major steps, similar to vanilla k-means. In the first step, b samples are drawn randomly from the dataset, to form a mini-batch. These are then assigned to the nearest centroid. In the second step, the centroids are updated. In contrast to k-means, this is done on a per-sample basis. For each sample in the mini-batch, the assigned centroid is updated by taking the streaming average of the sample and all previous samples assigned to that centroid. This has the effect of decreasing the rate of change for a centroid over time. These steps are performed until convergence or a predetermined number of iterations is reached.

MiniBatchKMeans converges faster than KMeans, but the quality of the results is reduced. In practice this difference in quality can be quite small.

We found that with a Batch Size of 600 and 4 clusters we get a F1=92.9%, F2=92.8% and FS = 92.8%. But most impressive is the time it takes. Just 3 seconds. We get slightly worse results than mixture methods, but it is impressive faster.

### HDBSCAN

HDBSCAN is a recent algorithm developed by some of the same people who write the original DBSCAN paper. Their goal was to allow varying density clusters. The algorithm starts off much the same as DBSCAN: we transform the space according to density, exactly as DBSCAN does, and perform single linkage clustering on the transformed space. Instead of taking an epsilon value as a cut level for the dendrogram however, a different approach is taken: the dendrogram is condensed by viewing splits that result in a small number of points splitting off as points ‘falling out of a cluster’. This results in a smaller tree with fewer clusters that ‘lose points’. That tree can then be used to select the most stable or persistent clusters. This process allows the tree to be cut at varying height, picking our varying density clusters based on cluster stability. The immediate advantage of this is that we can have varying density clusters; the second benefit is that we have eliminated the epsilon parameter as we no longer need it to choose a cut of the dendrogram. Instead we have a new parameter min\_cluster\_size which is used to determine whether points are ‘falling out of a cluster’ or splitting to form two new clusters. This trades an unintuitive parameter for one that is not so hard to choose for EDA (what is the minimum size cluster I am willing to care about?).

### OUTLIER DETECTION MODELS

We have to decide whether a new observation belongs to the same distribution as existing observations (inlier) or should considered as an outlier. We will differentiate in two concepts:

1. Novelty Detection: Data is polluted, but we want to detect anomalies in new observations.
2. Outlier Detection: Training data contains outliers, so we need to fit the central mode, ignoring the deviant observations.

#### ONE-CLASS SVM

We have to analyze if a new observation is so different from the others. Or on the contrary, if it is so similar that we cannot distinguish it from original observations. This is what Novelty Detection try to implement.

In general, it is about to learn from the delimiting contour of the initial observation distribution, plotting in the p-features-dimensional space. Then, if new observations lay within the frontier-delimited, they are considered as coming from the same population than the initial observations. Otherwise, if they lay outside the frontier, we can say they are abnormal.

We will use for Novel Detection the One-Class Support Vector Machine introduced by “Estimating the support of a high-dimensional distribution” Schölkopf, Bernhard, et al. Neural computation 13.7 (2001): 1443-1471.

This is an Unsupervised Algorithm (also works as SL because is SVM) that learns a decision function of what to do with novelty data (it classifies if data is similar or not to training set).

It requires the choice of a kernel and a scalar parameter to define the frontier.

a) Kernel: Kernel in machine learning are a class of algorithms for pattern analysis. They can operate in a high-dimensional implicit feature space, being computationally cheaper. The RBF is usually chosen. This is based on squared Euclidean Distance between feature vectors.

RBF can we writes as:

κ(x,x’)=exp(−γ∥x−x’∥2)

Here you can appreciate that this is a Squared Euclidean Distance for two features corrected by a gamma.

b) Gamma: This is the Kernel Coefficient that configures the sensitivity to differences in feature vectors. If you set gamma too large (gamma tends to infinity), this will end in overfitting. The kernel matrix becomes the unit matrix which leads to a perfect fit of the training data, though an entirely useless model.

The optimal value will depend on data. But you can optimize gamma and kernel parameters using Optunity. See sklearn\_SVM classification.pdf

c) Nu: This corresponds to the probability of finding a new but regular, outside observation outside the frontier.

#### ISOLATION RANDOM FOREST

If we have a high-dimensional datasets we can use random forests to detect outliers. The Isolation Forest algorithm (Liu, et al, 2008) isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximium and minimum values of the selected feature.

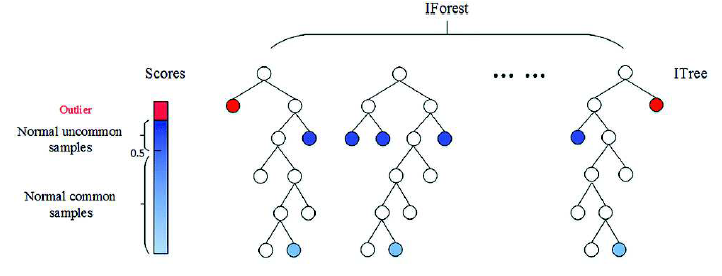
Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.

This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.

Random partitioning produces noticeably shorter paths for anomalies. Hence, when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies.

Now we will generate a little method that test where each data point belongs, i.e., if it is an outlier or it is a normal value, using sklearn.ensemble library.

We implement method IsolationForest. This method return an anomaly score of each sample using the Isolation Forest Algorithm:



Basically, we apply the same logic of Decision Trees. A tree structure is constructed to isolate every minority. That values that are more susceptible to be isolated (important anomalies) are discarded more near the root. Normal points tend to be isolated to the deeper on the tree. With this behavior we can construct an score. Isolation Forest, build finally an esamble of i Trees. In conclusion, anomalies will be that data points that have short AVERAGE path lengths on the iTrees.

If you noticed, the algorithm is **strongly based on that the amount of contamination is known. This is not trivial.**

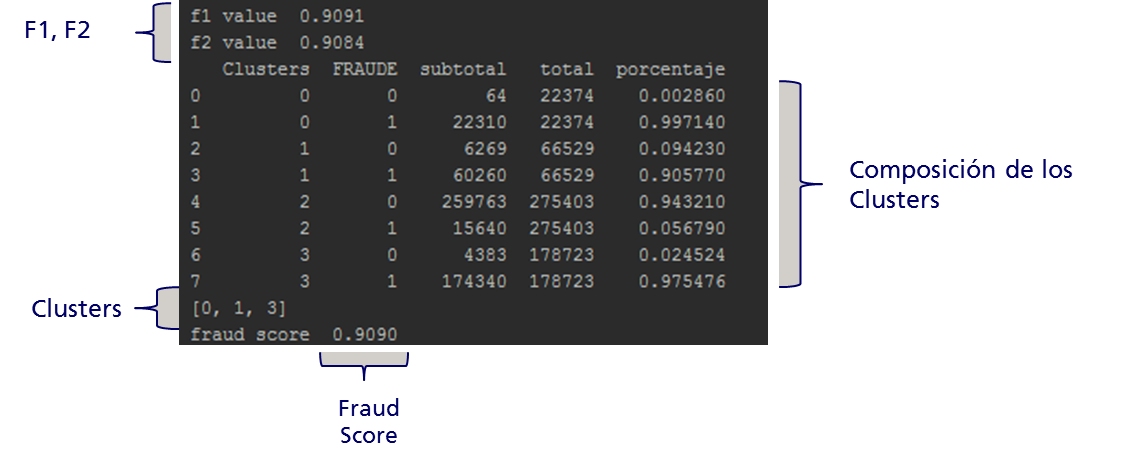
The results are not good at all. We get similar results as DBSCAN.

## UNSUPERVISED MODEL METODOLOGHY

Using the Fraud Score and the Unsupervised Models explained before, we iterate through each specific parameter of each model. In each iteration, we get F1, F2 and the Fraud Score. With this results, we can specifically get the best model.

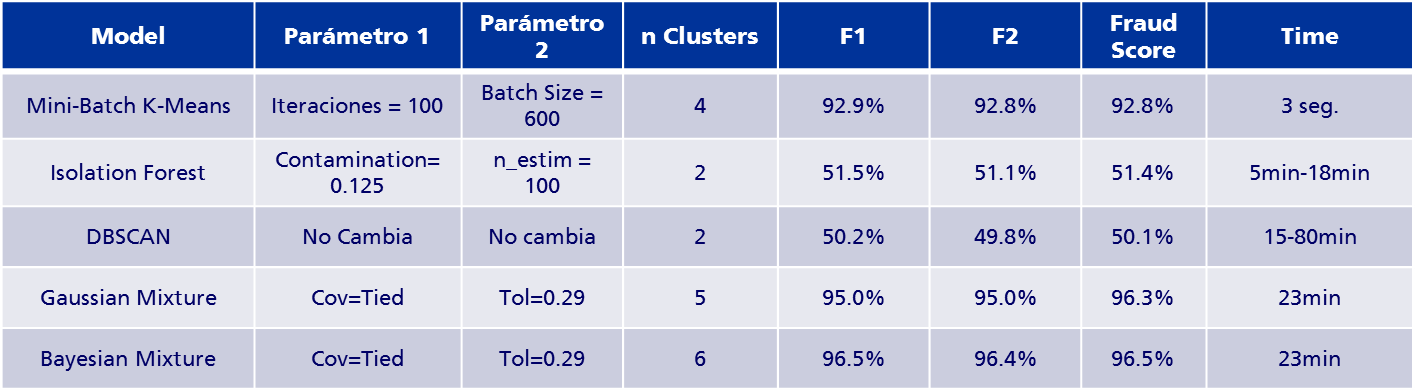
Once, we get the best model, we can reach the two or more classes that defines the clusters associated with Fraud or not Fraud cases. This classes will be the grounds of the Supervised Models.

In each iteration we have something like this:



As you can appreciate in this example, we have clusters [0,1,2,3]. Each of them have two type classes: 1 = Fraud, 0 = Unknown. Based in this information we have to decide which cluster type are they. Based in the 50% rule established before, we get that Cluster [0,1,3] are fraud cluster, because they have more than 50% of fraud cases.

You can appreciate the final results in the next table:



Once we decide the best model, we generate our target variable. We will have something like this:



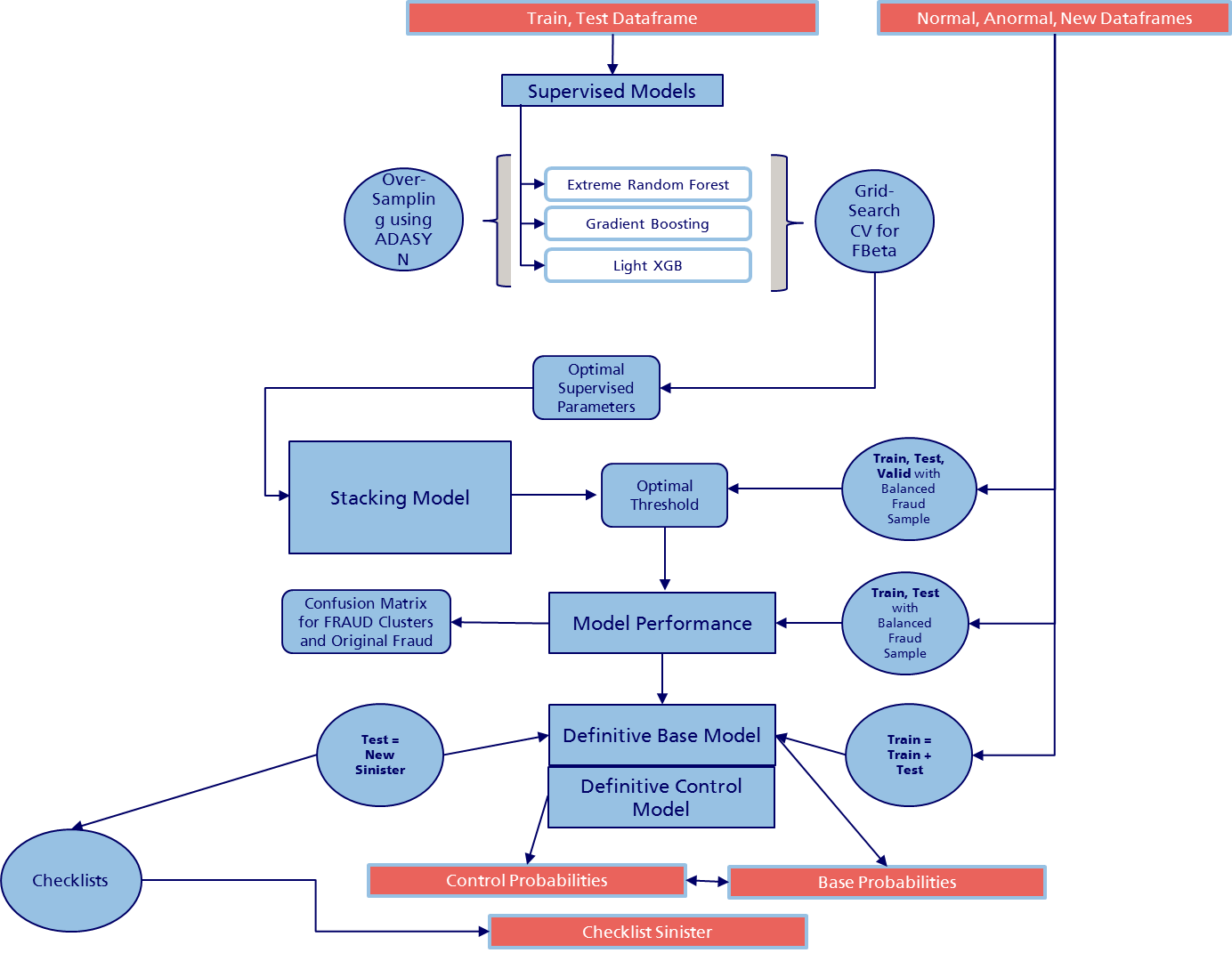
We have the original variable FRAUDE which determines if exist fraud = 1 or unknonwn = 0. Also it determines to which cluster belongs each sinister in Clusters. Using this Cluster variable we generate our Supervised Target Variable. Following the example before, if it belongs to Clusters = [0, 1, 3] we said that they are Fraud Clausters. But also, as you can appreciate in the sinister 140102997, which belongs to the non-Fraud Cluster 2, is considered as Fraud Cluster because originally it was a Fraud sinister.

Here is where the Unsupervised transform to a Supervised method, and therefore it evolves as a Semi-Supervised Algorithm.

Once we have our Target Variable FRAUDE\_Clusters we can continue to the Supervised metodhology.

## SUPERVISED MODEL

Now our problem has simplified. We have dependant variables and a independent fraud variable. However, we have to iterate again through different models. **Why we insist on several iterations?** Remember this is a dinamyc problem, which probabily will modify its behavior through the time. Maybe a good predictive model would not capture fraud in the future if it changes its pattern, and another yes. **Remember, people are not passive agents.**



Here we can exploit the benefits of synthetic oversamping using a predictive method as ADASYN, because we do not need to identify the id\_sinister. We have tried several models, but by computationally cost we had to reduce the scope to Gradient Boosting, Light XGB and Extreme Randomize Tree.

We iterate each of them using a GridSearch and using Cross Validation with 10 Train-Test splits. Finally we are trying to maximaze a FBeta score, which will put higher attention to the Recall (We are using a beta = 2).

### GRADIENT BOOSTING

GB is an adaption from the Adaboost algorithm, and it is applied to every loss function. It applies a sequential approximation to group importance, where the importance of a base learner is judged in terms of previous base learners . Therefore, data distribution is not perturbed by a random sub-sampling; instead we use observation weights. Unlike bagging and RF, our coefficients are not estimated after processing the model. They are calculated sequentially for each iteration where we get the best .

What GB does is to train a set of trees, where every tree is trained on the error of the previous ensemble models. GB starts the same way as bagging, but pays attention to the areas that have more mistakes. This gives a better approximation, without the need of a greater depth. Also, this is an essential advantage. Instead of RF, GB can work great based on weak learners in terms of high bias and low variance (even as small as decision stumps). GB reduces error mainly by reducing bias, and also to some extent variance, by aggregating the output from many models.

In summary, RF trains with random sample of data in addition to randomizing features. It trusts randomization to have better generalization performance on out-sample data. On the other hand, GB additionally tries to find the optimal linear combination of trees, where the final model is the weighted sum of predictions of individual trees applied to training data.

### LIGHT XGB

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms. Also, it is surprisingly very fast, hence the word ‘Light’.

There has been only a slight increase in accuracy and auc score by applying Light GBM over XGBOOST but there is a significant difference in the execution time for the training procedure. Light GBM is almost 7 times faster than XGBOOST and is a much better approach when dealing with large datasets.

### RADIUS NEIGHBORS CLASSIFIER (WE DISCARDED THIS)

Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

In cases where the data is not uniformly sampled, radius-based neighbors classification in RadiusNeighborsClassifier can be a better choice. The user specifies a fixed radius r, such that points in sparser neighborhoods use fewer nearest neighbors for the classification. For high-dimensional parameter spaces, this method becomes less effective due to the so-called “curse of dimensionality”.

### EXTREMELY RANDOMIZED TREES

Bagging only constructs trees using bootstrap samples of data. RF however, also uses a random sample on predictors before every node is split, until the tree is fulfilled. This gives higher independence between trees, because of the combination of bootstrap samples and random draws of predictors. Consequently, we can take advantage from averaging a large number of trees (and so getting better levels of variance reduction). Also, we can gain on bias reduction, because we can employ a very large number of predictors (even more than observations), and local feature predictors can play a role in the tree construction. In our example, we will not gain with this, because we will only be focusing on our 12 variables, for the models to be comparable.

In conclusion we get the advantages of bagging, but also on a less propensity to overfitting (each tree fits, or overfits, a part of the training set, and in the end their errors cancel out, at least partially), and, as we will see later, it is easier compared to Gradient Boosting (GB) in terms of tuning. In this way, RF works great with fully grown decision trees (low bias, high variance). It tackles the error reduction task in the opposite way, by reducing variance. Instead of GB, trees are made uncorrelated to maximize the decrease in variance, but RF cannot reduce bias (slightly higher than a simple binary tree). Hence, the need for large unpruned trees, so as the bias is as low as possible.

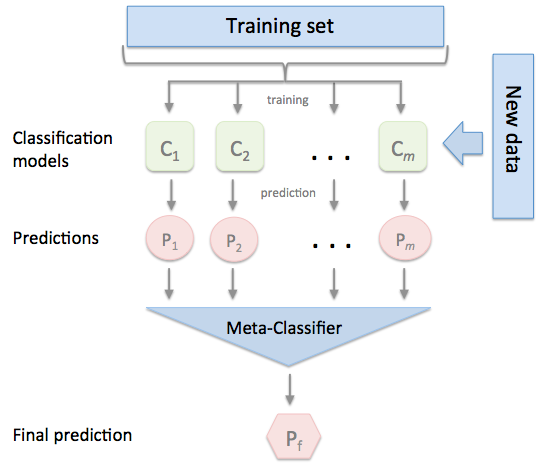
Unlike in decision trees, we can reduce the number of observations in terminal nodes, because RF tends less to overfitting.

In Extremely Randomized Trees (ERT)[[2]](#footnote-2), randomness goes one step further in the way splits are computed. As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule. This usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias.

The main parameters to adjust when using these methods is n\_estimators and max\_features. The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute. In addition, note that results will stop getting significantly better beyond a critical number of trees. The latter is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, but also the greater the increase in bias. Empirical good default values are max\_features=n\_features for regression problems, and max\_features=sqrt(n\_features) for classification tasks (where n\_features is the number of features in the data). Good results are often achieved when setting max\_depth=None in combination with min\_samples\_split=1 (i.e., when fully developing the trees). Bear in mind though that these values are usually not optimal, and might result in models that consume a lot of RAM. The best parameter values should always be cross-validated. In addition, note that in random forests, bootstrap samples are used by default (bootstrap=True) while the default strategy for extra-trees is to use the whole dataset (bootstrap=False). When using bootstrap sampling the generalization accuracy can be estimated on the left out or out-of-bag samples. This can be enabled by setting oob\_score=True.

### MODEL STACKING

Also we try different combinations of Stacking Model. Stacking (also called meta ensembling) is a model ensembling technique used to combine information from multiple predictive models to generate a new model. Often times the stacked model (also called 2nd-level model) will outperform each of the individual models due its smoothing nature and ability to highlight each base model where it performs best and discredit each base model where it performs poorly. For this reason, stacking is most effective when the base models are significantly different.



However, in general, stacking produces small gains with a lot of added complexity – not worth it for most businesses.

We stacked each combination possible. However the results are not better than the individual models.

The results of the best parameters of each models are as follows:

|  |
| --- |
| n\_estimators\_gb = 300 |
| max\_depth\_gb = 200 |
| sampling\_gb = 'ADASYN' |
| num\_leaves = 2000 |
| n\_estimators\_lgb = 300 |
| max\_bin = 500 |
| sampling\_lgb = 'ADASYN' |
| max\_depth = 200 |
| oob\_score = True |
| class\_weight = 'balanced\_subsample' |
| base\_sampling = None |
| control\_sampling = 'ADASYN' |
| bootstrap = True |
| n\_estimators = 500 |

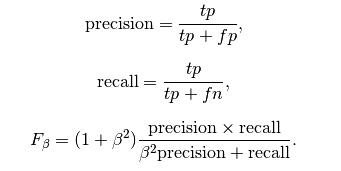
### THRESHOLD OPTIMATIZON

Once we have the individual best parameters for each model, we proceed to get the optimal threshold for each of them. Finally, this threshold will impose the restriction to investigate or not a case. Our strategy consists in use a FBeta Score with beta=2, to give the most importance to the recall.

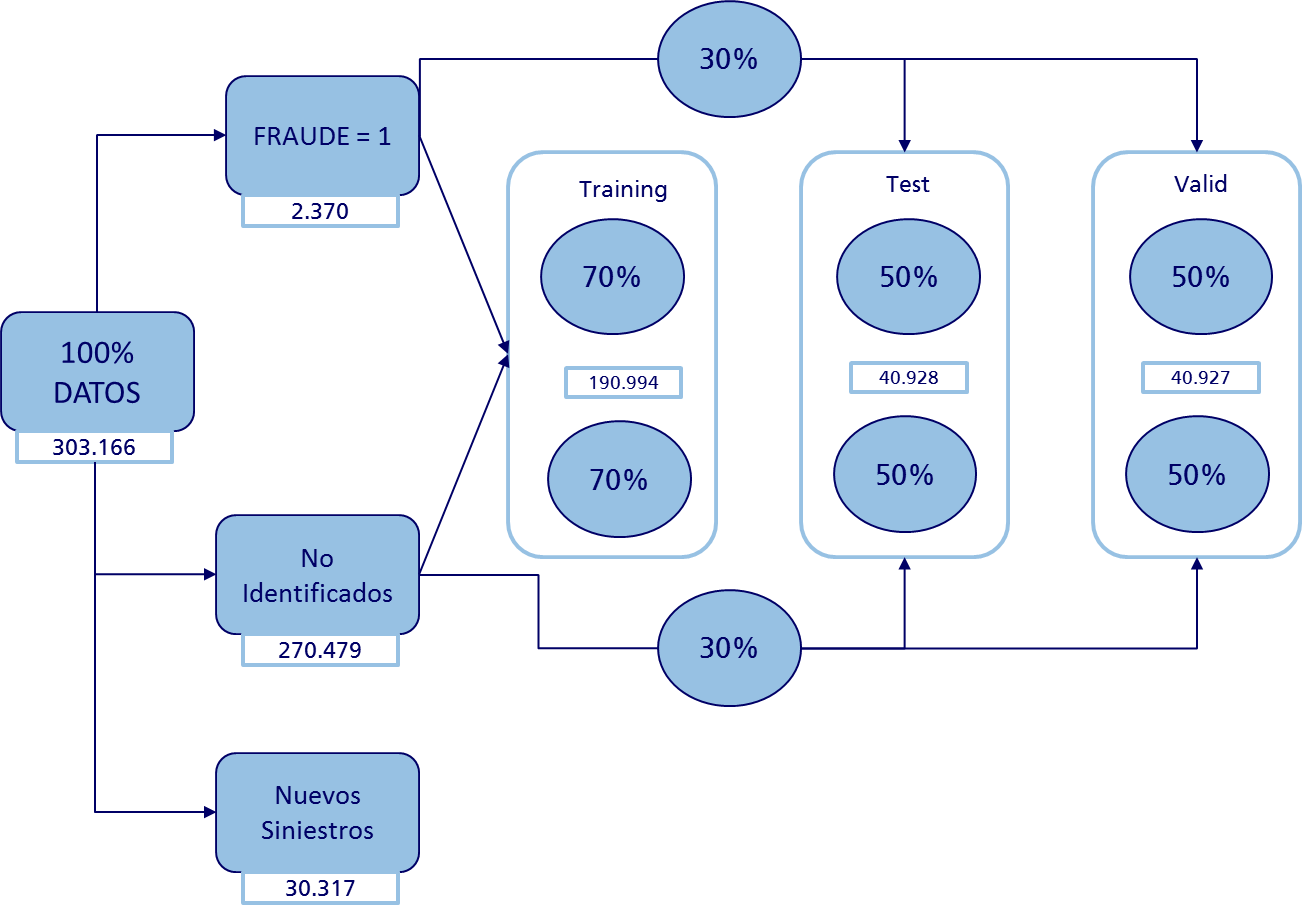
I think is important understand why we use FBETA with a higher weight over the Recall. So I will take a minute to explain this statistc formulation.

Intuitively, precision is the ability of the classifier not to label as positive a sample that is negative, and recall is the ability of the classifier to find all the positive samples.

The F-measure (F\_beta and F\_1 measures) can be interpreted as a weighted harmonic mean of the precision and recall. A F\_beta measure reaches its best value at 1 and its worst score at 0. With beta = 1, F\_beta and F\_1 are equivalent, and the recall and the precision are equally important. But with a beta =2 we give more importance to the recall. Finally we want to be capable to detect all the possible fraude, regardless investigation not fraud cases.

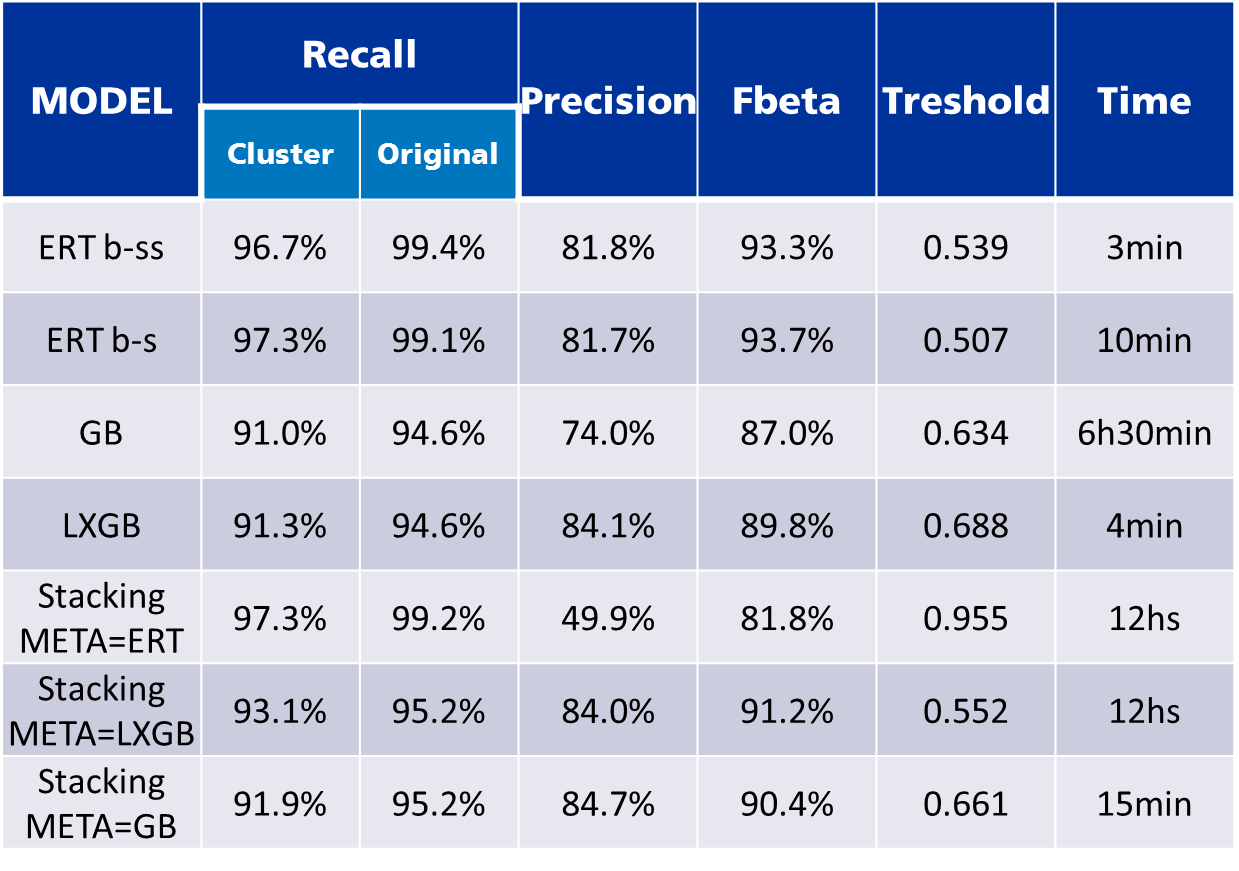


For that we divide our distribution as follows:



We have saved 10% of the sample (Nuevos Siniestros) to be evaluated at the end. Then we divide our distribution between Training-Test-Valid, using the Valid to parametrize the Treshold.

Here are the results:

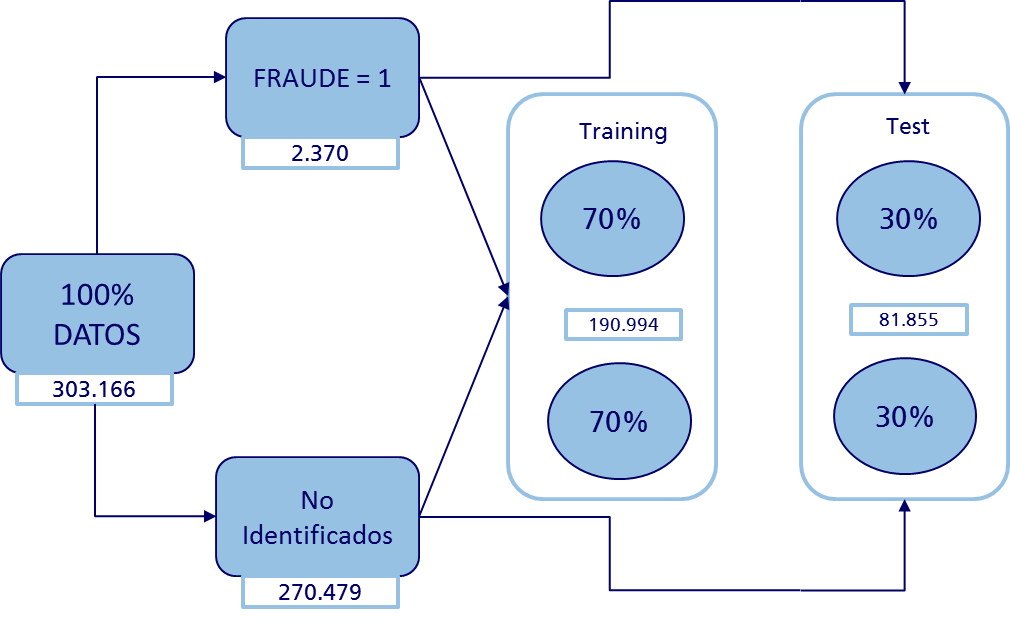


As we use just half of the total sample the results will be slighthly worst than final results. But you can appreciate that we have more than 91% of recall in every model, and a precision rounding 80%. They are extremely good results.

As you can see, the two first models are the best in terms of FBeta, the automatized parameter that decide the model to finally apply. They are the Extremely Randomized Tree methods. First of them(ERT b-ss) is a balanced Subsample strategy. That is, in each tree node we balanced the sample tried to be split. The ERT-bs balance using ADASYN oversampling method. They get very similar results, but ERT b-ss tend to be faster.

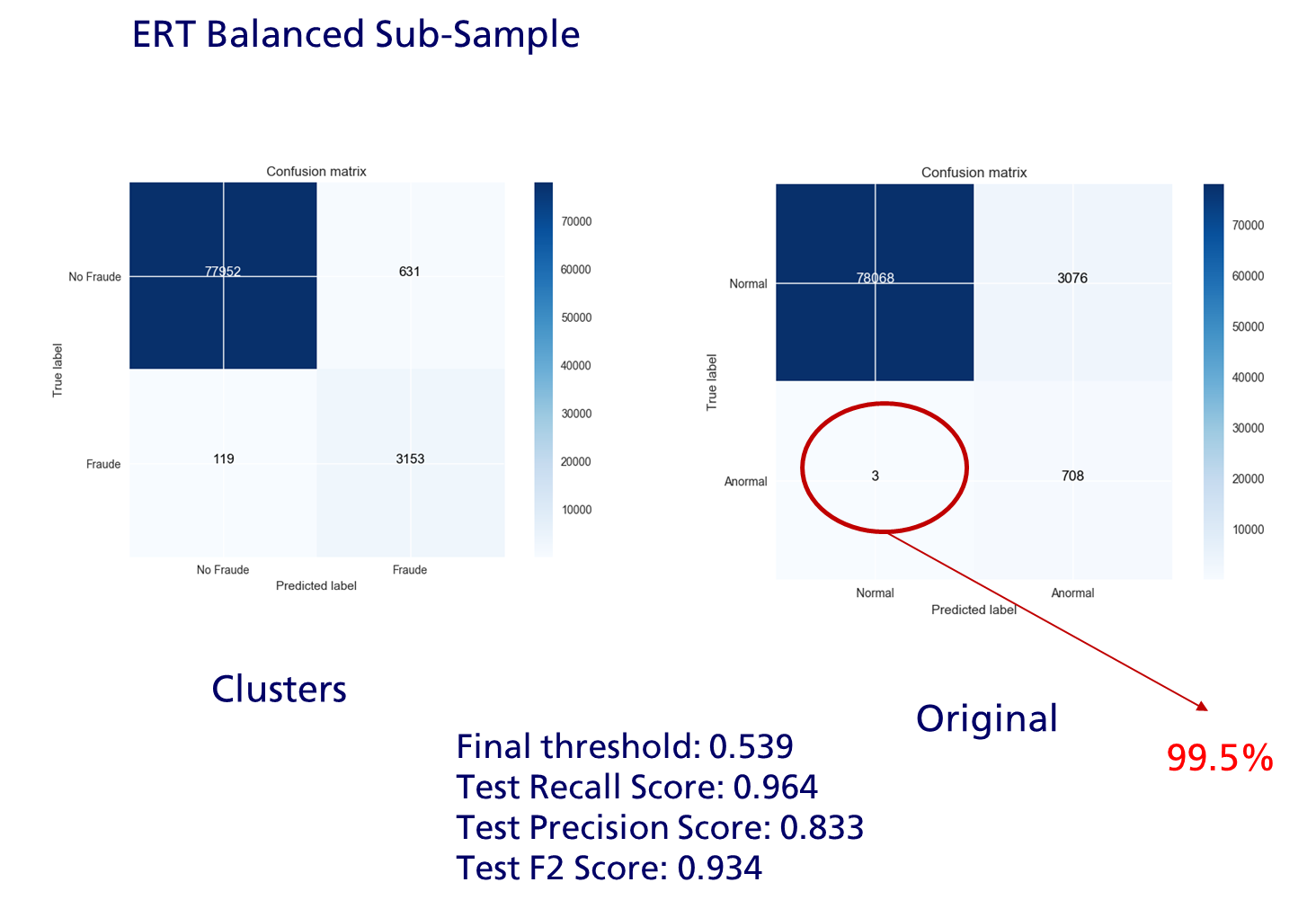
### MODEL PERFORMANCE

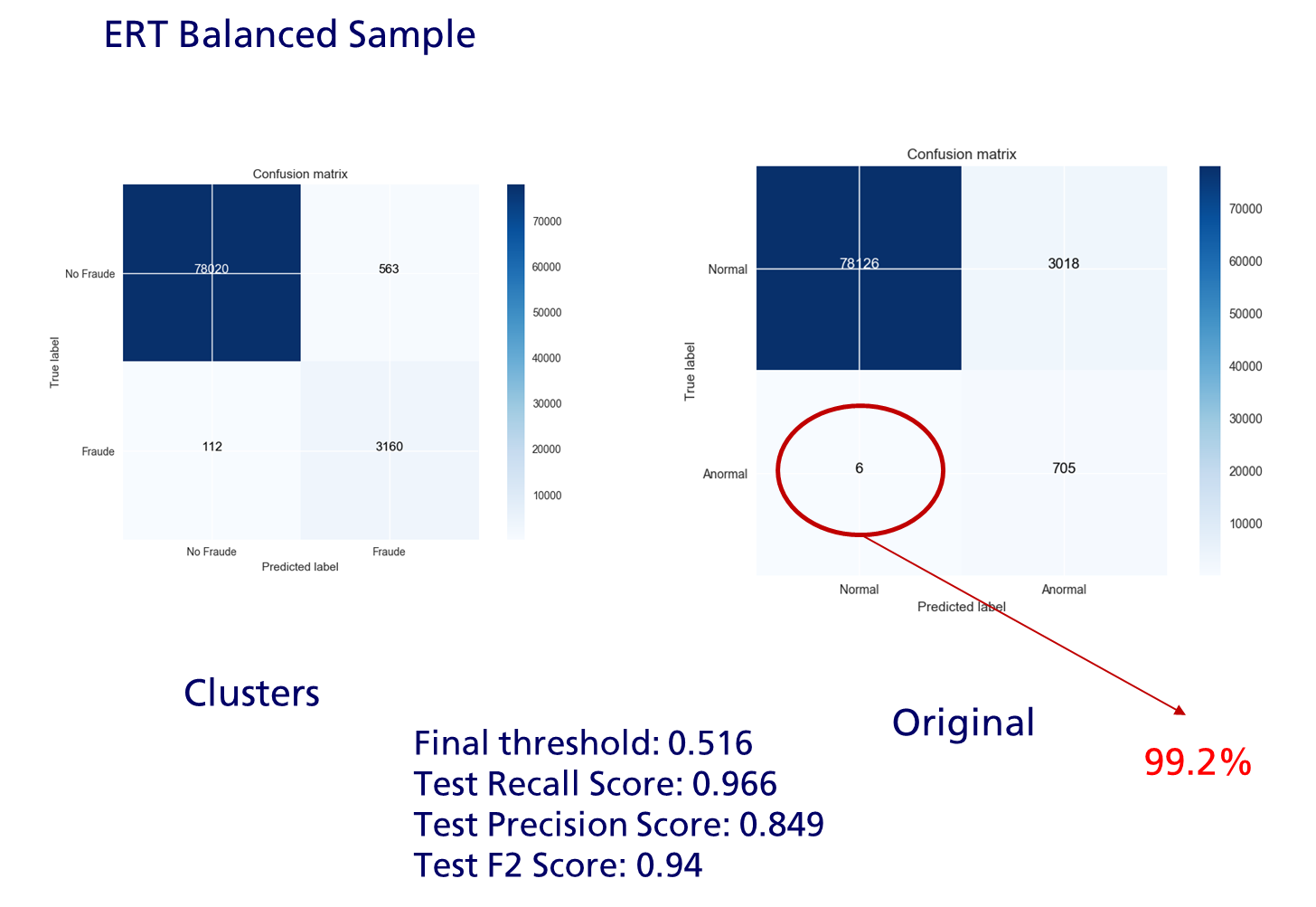
Once we have the optimal model, their optimal parameters and also its optimal threshold, we can evaluate the performance using the whole sample.



However, we continue keeping our 10% sample to be the sacred final test.

Here, we compute the confussion matrix of the best two models (ERT b-ss and ERT b-s). Each output is automatically saved day by day in a specific folder for control reasons.

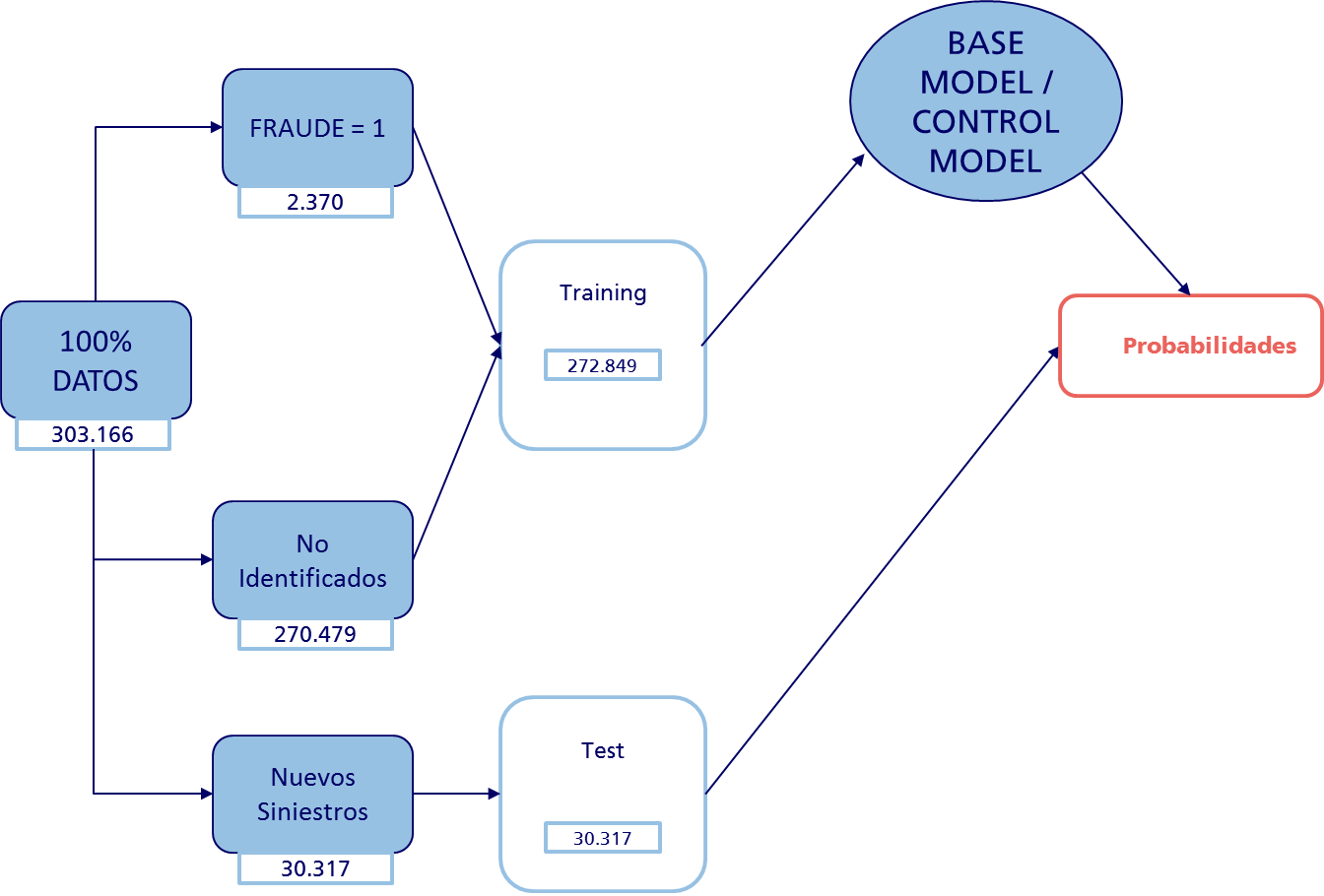




Let me explain the first of them. As you had noticed, we have two Recall values. Cluster Recall and Original Recall. The left figure represents the result using as target variable our constructed Fraud Cluster Variable. As you noticed, we capture 3153 cases as Fraud-Fraud. But we cannot well identify 119 cases. However, we do not know if they are actually Fraud cases. We have to watch the right confussion matrix. Actually, only 3 cases were identified fraud. This implies that over the original data we had missdetected only 3 cases in a sample of 81.855. This is really impressive in a dataset with only 0.08% of information.

### MODEL APPLIED

Now, the final step consists in apply the model. In this cases we will try it in our 10% sample saved at the beginning. It will be something like this:



Here you notice that we have two models, a base model and a control model. This is because we want to know if actually we are predicting the same sinister with two well functioning models. They are ERT b-ss and ERT b-s. ERT b-ss will basically be our Base Model. It will calculate the probabilities and the threshold (and also add some information) like this:



But then, we are going to calculate the same for the Control Model and merged both. And we are going to get something like this:

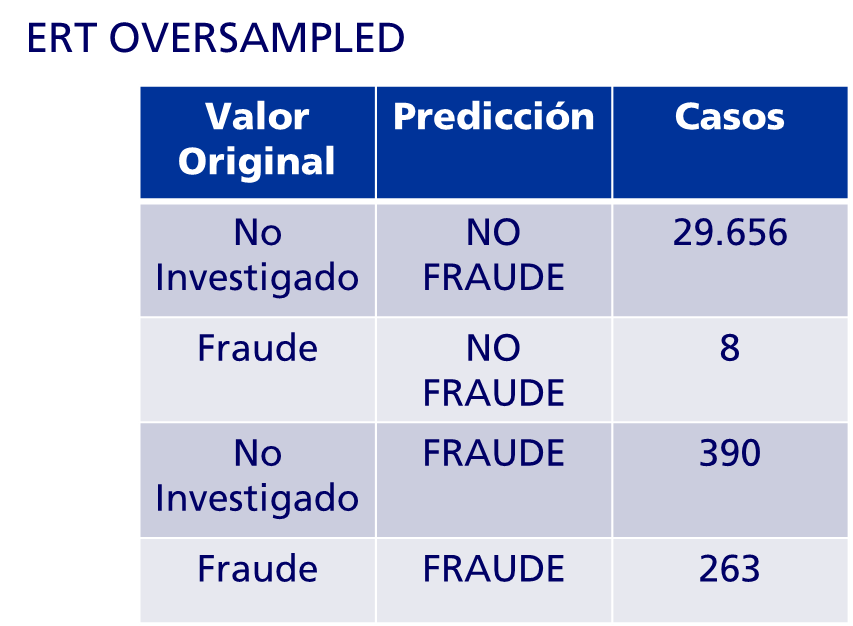


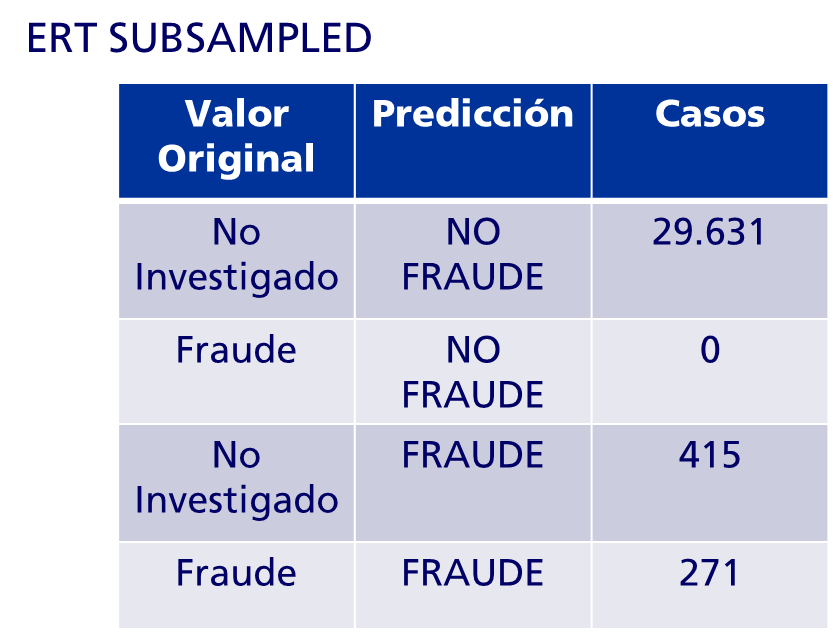
This table will be saved for us each day. So we can have a strict control and also we can post-evaluate the real functioning of the model. See the following table:



Basically, we are going to delete that cases that are identified by the first model but not by the control model. This will save time to the Investigation Office, deleting cases that are ambiguous.

Here are the results of both models:





As you can appreciate, the ERT Subsampled has predicted every case of Original Fraud. This imply a recall of 100%. But also added 415 cases, which were originally not investigated. This cases are which will finally decide if we are doing a better job predicting than the original rule model.

The ERT Oversampled has failed on detecting 8 cases. It is not a bad result (It is a recall of 97%), but it is not perfect. However it detected less cases as originally unknown cases (390). If you had noticed before this model has a worst recall but a better precision, so it is perfect to be the Control Model.

As you can see, the results are really amazing. Remember the original complexity of the problem.

Now it is time to cross both models:

|  |  |  |
| --- | --- | --- |
|  | **ERT - SS** | **ERT – OS** |
| CASOS FRAUDE | 686 | 654 |
| CASOS NO FRAUDE | 29.631 | 29.664 |
| FN | - | 8 |
| RECALL SOBRE ORIGINAL | 100% | 97% |
| CASOS IGUALES | 638 | 638 |
| CASOS EN i Y NO j | 48 | 15 |
| TASA COINCIDENCIA(CASOS IGUALES/CASOS FRAUDE) | 93% | 97.5% |
| CASOS NO INVESTIGADOS | 415 | 390 |
| CASOS EN i Y NO j | 40 | 15 |
| CASOS IGUALES | 375 | 375 |
| TASA COINCIDENCIA | 90.3% | 96.2% |

The ERT–SS has detected 686 cases as Fraud, agains 654 of ERT–OS. We have 0 False Negatives with the former, and 8 with ERT-OS. Both have great Recall values. However, from this Fraud Cases, they coincide on 638 cases. We have 48 cases that are in ERT-SS but not in ERT-OS, in contrast to 15 cases that are in ERT-SS but not in ERT-OS.

From this cases, 415 are originally unknown cases using ERT-SS. And 390, using ERT-OS. And actually, 375 cases are in the intersection of them. Basically both are predicting the same cases in between 90-96% of the cases.

This 375 will be the final output that the Investigation Office will receive each day.

### INVESTIGATION OFFICE CONCLUSIONS

This 375 will determine if we are doing a better job than the current rule based model.

# MACHINE LEARNING IMPLEMENTATION

Once we have the final output, it is time to restart the process. Finally, this is the important Machine Learning notion.

See the following highly simplified machine learning implementation:



We start with new open sinister. They enter in the Daily processed explained in the [Daily-Preprocessing Section](#_DAILY_PRE-PROCESSING). Once the data is preprocessed it enters in the predictive model explained in the [Model Application Section](#_MODEL_APPLICATION). If it is lower than the specified threshold it will not part of the output. On the contrary, it will be the output that daily receive the Investigation Office. Also, it could be part of the output if it involves some of the checklist rules applied (explaind in the [Checklists Section](#_CHECKLISTS)). The Investigation Office will receive the new probable fraud sinister in a daily file.

Here we have to make a statement. If the sinister has been registred before as a fraud sinister, and it doe not change significatively the probability, it will not be sended. We refer as significatively if it does not change the probability by a -5%/+5%.

Once the sinister is resolved and/or closed, it will belong to the training core dataset. This training core will be update weekly with the closed and resolved sinister, as explained in the [Weekly Pre-Processing Section](#_WEEKLY_PRE-PROCESSING).

# CHECKLISTS

Here, we are going to explain what the checklists are. Essentially, they are alerts that are triggered under certain circumstances. We have two types.

-Mandatory: If exists just one alert, it is mandatory to warn to the Investigation Office.

-Not mandatory: Warn or not is based on the processor’s experience. However, if there are two or more checklists, it is mandatory.

|  |  |  |  |
| --- | --- | --- | --- |
| **Núm. Check** | **Reportar SIEMPRE (Obligatorias)** | **Tipología** | **Ramo** |
| **1** | Conocido Clan / familia reincidente | General | Todos |
| **2** | Implicado con antecedentes de fraude recurrente y en conocimiento del Tramitador | General | Todos |
| **3** | Cambio reciente de coberturas y declaración inmediata de siniestro que afecta al cambio (p.ej. aumento de capitales) | General | Todos |
| **4** | 2 ó + siniestros con daños importantes (> 5.000 euros en Hogar y Comunidades ó >10.000 en Comercio) (\*) | General | Todos |
| **5** | 2 siniestros de robo con joyas del mismo asegurado | Robo | Hogar |
| **6** | Daños eléctricos en ausencia de tormentas + incidencias en la red y 2 o mas reclamaciones por daños eléctricos en el año. | Daños eléctricos | Todos |
| **7** | Un siniestro por anualidad repetitivo (atraco) | Robo/Atraco | Hogar |

|  |  |  |  |
| --- | --- | --- | --- |
| **Núm. Check** | **Valoración por Tramitador (2 ó + indicios en mismo expediente comunicar siempre)** | **Tipología** | **Ramo** |
| **8** | Descripción sospechosa, nerviosismo o insistencia en declararse culpable o con cambio de versión. Agresividad, falta de cooperación…. | General | Todos |
| **9** | Indicios de connivencia: similitud de apellidos, domicilios próximos entre los implicados en el siniestro, etc. | General | Todos |
| **10** | Disposición para aceptar un arreglo rápido a bajo coste | Responsabilidad Civil | Todos |
| **11** | Reclamación de objetos que no constan en la denuncia | Robo | Todos |
| **12** | Si no se perita y se aporta únicamente factura de reparación cunado se declaren **filtraciones y humedades,** o si se perita cuando el siniestro ocurre dentro de los 3 primeros meses desde la contratación. | Daños por agua | Todos |
| **13** | No acredita preexistencias y/o las facturas que presentan no son a nombre del tomador | General | Todos |
| **14** | Reiteración de 3 o + siniestros parecidos en el mismo riesgo en un plazo de tres meses | General | Todos |

As you can see, computationally, it is harder to asses the not-mandatory alerts. Therefore, we have to put our focus on the mandatory alerts. For that we construct a module named checklist.py. It has to classes, ‘checklist\_obligatorias’ and ‘checklist\_valoracion’.

## CHECKLIST\_OBLIGATORIAS CLASS

Here we have 8 methods that represent the seven mandatory checklists (the check number six is divided in two methods).

### CHECKLIST 1

It is reffered to a situation where the operator can detect a fraudulent family known. For us it is a bit difficult to systematize this behavior. We do not have the proper information for that. However, we expect in the future to work with graph networks (See [Section Future Challenges](#_GRAPHS_NETWORKS) for more information).

### CHECKLIST 2

It indicates if the person involved in the sinister has fraud records. In this case it is very simple. We just retrieve the number of fraud sinister for that person (using the blacklist).

**def checklist2**(df\_test\_id:pd.DataFrame):  
 *"""  
 Implicado con antecedentes de fraude recurrente y en conocimiento del Tramitador* ***:return****: This return a Dataframe with the columns 'id\_siniestro', 'checklist2', where 'checklist2' counts the   
 number of times an ID\_FISCAL appears in the Blacklist  
 """* file\_blacklist = ReadCsv.load\_csv(STRING.processed\_blacklist)  
 df\_bl = DfUtils.processing\_file(file\_blacklist, delimiter=';')  
 df\_bl = df\_bl.drop\_duplicates(subset=['id\_siniestro', 'NIF'], keep='last')  
 df\_bl = df\_bl[['NIF']]  
 df\_bl['NIF'] = df\_bl['NIF'].map(str)  
 df\_bl = df\_bl.groupby(['NIF']).count().reset\_index(drop=**False**)  
 file\_df = df\_test\_id[['id\_siniestro', 'id\_fiscal']]  
 file\_df = file\_df['id\_fiscal'].map(str)  
 file\_df = pd.merge(file\_df, df\_bl, how='left', left\_on='id\_fiscal', right\_on='NIF')  
 **del** file\_df['NIF']  
 **del** file\_df['id\_fiscal']  
 file\_df['Count'] = file\_df['Count'].fillna(0)  
 file\_df.columns = ['checklist2']  
  
 **return** file\_df

### CHECKLIST 3

It warns about coverage changes in the policy, inmediatly the sinister has occurred. In that case, we take the variable ‘hist\_movimiento\_mod\_garantias\_fecha’ that is in the HISTORICO\_MOVIMIENTO\_POLIZA\_REFERENCIA. This variable indicates the last guarantee movement before the sinister. Then, it is easy, we take the date differences between these and the sinister occurance.

**def checklist3**(df\_test\_hist\_mov\_pol\_ref:pd.DataFrame):  
 *"""Cambio reciente de coberturas y declaración inmediata de siniestro que afecta al cambio   
 (p.ej. aumento de capitales)* ***:return****: This return a Dataframe with the columns 'id\_siniestro', 'checklist3', where 'checklist3' is the   
 difference between the last guarantee modification and the sinister occurance.  
 """* df\_test\_hist\_mov\_pol\_ref = df\_test\_hist\_mov\_pol\_ref[['id\_siniestro', 'hist\_movimiento\_mod\_garantias\_fecha',  
 'hist\_movimiento\_siniestro\_fecha']]  
  
 **for** i **in** 'hist\_movimiento\_mod\_garantias\_fecha', 'hist\_movimiento\_siniestro\_fecha':  
 df\_test\_hist\_mov\_pol\_ref = pd.to\_datetime(df\_test\_hist\_mov\_pol\_ref[i], format='%Y-%m-%d', errors='coerce')  
  
 df\_test\_hist\_mov\_pol\_ref['checklist3'] = pd.Series(df\_test\_hist\_mov\_pol\_ref['hist\_movimiento\_siniestro\_fecha']  
 - df\_test\_hist\_mov\_pol\_ref[  
 'hist\_movimiento\_mod\_garantias\_fecha']).dt.days  
  
 **del** df\_test\_hist\_mov\_pol\_ref['hist\_movimiento\_mod\_garantias\_fecha']  
 **del** df\_test\_hist\_mov\_pol\_ref['hist\_movimiento\_siniestro\_fecha']  
  
 **return** df\_test\_hist\_mov\_pol\_ref

### CHECKLIST 4

It denotes how many sinister are about important damages. It alerts cases where we have the same person with two or more sinister with a damage amount of 5000 euros or greater.

For that, we have to sum the initial Reserve of each sinister. Because that is the first amount which is claimed by the insured. We have to be careful about duplicates, because our Reserve bottle is constructed on the basis of payments. In consequences, different payments are associated with the same initial reserve.

We make the analysis by Policy and by NIF, because we can capture people with more than one Policy. For that, we have to mix the Reserve bottle with the ID bottle.

**def checklist4**(df\_po\_reserva\_test:pd.DataFrame, df\_test\_id: pd.DataFrame):  
 *"""  
 2 ó + siniestros con daños importantes (> 5.000 euros en Hogar y Comunidades ó >10.000 en Comercio) (\*)  
 # Se toma la reserva inicial del siniestro.* ***:param*** *df\_test\_id:* ***:return****: This return a Dataframe with the columns 'id\_siniestro', 'checklist4\_poliza', 'checklist4\_nif', where   
 'checklist4\_' represents how many sinister (by policy/nif) has an initial reserve >= 5000 since 2015  
 """* # We need the PO\_RESERVA base to get past sinister  
 reserva\_base = ReadCsv.load\_csv(STRING.reserva\_file)  
 reserva\_base = DfUtils.processing\_file(reserva\_base, delimiter=';')  
  
 # We need id base to cross the past sinister  
 id\_base = ReadCsv.load\_csv(STRING.id\_file)  
 id\_base = DfUtils.processing\_file(id\_base, delimiter=';')  
 id\_base = id\_base[['id\_siniestro', 'id\_fiscal']]  
 df\_test\_id = df\_test\_id[['id\_siniestro', 'id\_fiscal']]  
 id\_base = pd.concat([id\_base, df\_test\_id], aixs=0)  
  
 # We take the variables we need and concat the new sinister with past sinister  
 reserva\_indem\_base = reserva\_base[['id\_siniestro', 'po\_res\_cobertura\_id', 'po\_res\_indem']]  
 reserva\_indem = df\_po\_reserva\_test[['id\_siniestro', 'po\_res\_cobertura\_id', 'po\_res\_indem']]  
  
 reserva\_indem = pd.concat([reserva\_indem, reserva\_indem\_base], axis=0)  
 **del** reserva\_base  
 **del** reserva\_indem\_base  
  
 # We merge with ID by sinister  
 reserva\_indem['id\_siniestro'] = reserva\_indem['id\_siniestro'].map(int)  
 id\_base['id\_siniestro'] = id\_base['id\_siniestro'].map(int)  
  
 reserva\_indem = pd.merge(reserva\_indem, id\_base, how='left', on='id\_siniestro')  
  
 # We calculate the initial RESERVA for each policy and create the variable RESERVA > 5000  
 reserva\_indem = reserva\_indem.drop\_duplicates(subset=['id\_siniestro', 'po\_res\_cobertura\_id',  
 'po\_res\_indem'],  
 keep='first')  
 **del** reserva\_indem['po\_res\_cobertura\_id']  
 reserva\_indem['po\_res\_indem'] = reserva\_indem['po\_res\_indem'].map(float)  
 reserva\_indem = reserva\_indem.groupby(['id\_siniestro', 'id\_poliza', 'id\_fiscal'])[  
 'po\_res\_indem'].sum().reset\_index()  
 reserva\_indem['po\_res\_indem\_mayor\_5000'] = pd.Series(0, index=reserva\_indem.index)  
 reserva\_indem.loc[reserva\_indem['po\_res\_indem'] >= 5000, 'po\_res\_indem\_mayor\_5000'] = 1  
  
 # Now we have the values by sinister, we group by id\_poliza and by nif  
 poliza\_indem = reserva\_indem.groupby(['id\_poliza'])['po\_res\_indem\_mayor\_5000'].sum().reset\_index()  
 nif\_indem = reserva\_indem.groupby(['id\_fiscal'])['po\_res\_indem\_mayor\_5000'].sum().reset\_index()  
  
 # Merge the results  
 poliza\_indem['id\_poliza'] = poliza\_indem['id\_poliza'].map(str)  
 reserva\_indem['id\_poliza'] = reserva\_indem['id\_poliza'].map(str)  
 poliza\_indem.columns = ['id\_poliza', 'checklist4\_poliza']  
  
 nif\_indem['id\_fiscal'] = nif\_indem['id\_fiscal'].map(str)  
 reserva\_indem['id\_fiscal'] = reserva\_indem['id\_fiscal'].map(str)  
 nif\_indem.columns = ['id\_fiscal', 'checklist4\_nif']  
  
 reserva\_indem = pd.merge(reserva\_indem, poliza\_indem, how='left', on='id\_poliza')  
 reserva\_indem = pd.merge(reserva\_indem, nif\_indem, how='left', on='id\_fiscal')  
 **del** reserva\_indem['id\_poliza']  
 **del** reserva\_indem['id\_fiscal']  
  
 # We need just to take the new sinister  
 df\_po\_reserva\_test = df\_po\_reserva\_test[['id\_siniestro']]  
 df\_po\_reserva\_test['id\_siniestro'] = df\_po\_reserva\_test['id\_siniestro'].map(int)  
 df\_po\_reserva\_test = pd.merge(df\_po\_reserva\_test, reserva\_indem, how='left', on='id\_siniestro')  
  
 **return** df\_po\_reserva\_test

### CHECKLIST 5

Two or more sinister related to the theft of jewellery. As in the previous case we have to cross the Id bottle with the reserve bottle. Basically, we have to find the sinister associated with jewellery coverage. Then, taking out the duplicate cases, we make a count by NIF and by Policy.

**def checklist5**(df\_reserva:pd.DataFrame, df\_reserva\_test:pd.DataFrame, df\_id:pd.DataFrame, df\_id\_test: pd.DataFrame):  
 *"""  
 2 siniestros de robo con joyas del mismo asegurado* ***:param*** *df\_test\_id:* ***:return****: This return a Dataframe with the columns 'id\_siniestro', 'checklist5\_poliza', 'checklist5\_nif', where   
 'checklist5\_' represents how many sinister (by policy/nif) belongs to JOYAS coverage  
 """* **if** df\_reserva\_test **is not None**:  
 df\_reserva\_test = df\_reserva\_test[['id\_siniestro', 'id\_poliza', 'po\_res\_cobertura']]  
 df\_reserva\_base = df\_reserva[['id\_siniestro', 'id\_poliza', 'po\_res\_cobertura']]  
 **del** df\_reserva\_test  
 **del** df\_reserva\_base  
 df\_reserva = pd.concat([df\_reserva\_base, df\_reserva\_test], axis=0)  
  
 df\_reserva = df\_reserva[['id\_siniestro', 'id\_poliza', 'po\_res\_cobertura']]  
 df\_reserva = df\_reserva.drop\_duplicates(subset=['id\_siniestro', 'po\_res\_cobertura'])  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('EXC.'), 'po\_res\_cobertura'] = 'CONTENIDO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('JOY'), 'po\_res\_cobertura'] = 'INCIDENCIA'  
  
 df\_reserva = df\_reserva[df\_reserva['po\_res\_cobertura'] == 'INCIDENCIA']  
  
  
 # We merge with ID by sinister  
 **if** df\_id\_test **is not None**:  
 df\_id\_test = df\_id\_test[['id\_siniestro', 'id\_fiscal']]  
 df\_id\_base = df\_id[['id\_siniestro', 'po\_res\_cobertura']]  
 **del** df\_id\_test  
 **del** df\_id\_base  
 df\_id = pd.concat([df\_id\_base, df\_id\_test], axis=0)  
  
 df\_id = df\_id[['id\_siniestro', 'id\_fiscal']]  
 df\_reserva['id\_siniestro'] = df\_reserva['id\_siniestro'].map(int)  
 df\_id['id\_siniestro'] = df\_id['id\_siniestro'].map(int)  
  
 reserva\_cobertura = pd.merge(df\_reserva, df\_id, how='left', on='id\_siniestro')  
  
 # We calculate the COUNT of JOYAS  
 reserva\_cobertura = reserva\_cobertura.drop\_duplicates(subset=['id\_siniestro'], keep='first')  
 # Now we have the values by sinister, we group by id\_poliza and by nif  
 reserva\_cobertura['checklist5\_poliza'] = reserva\_cobertura['id\_poliza'].groupby(  
 reserva\_cobertura['id\_siniestro']).transform('count')  
  
 reserva\_cobertura['checklist5\_nif'] = reserva\_cobertura['id\_fiscal'].groupby(  
 reserva\_cobertura['id\_siniestro']).transform('count')  
  
 # Simplify  
 **del** reserva\_cobertura['id\_poliza']  
 **del** reserva\_cobertura['id\_fiscal']  
 **del** reserva\_cobertura['po\_res\_cobertura']  
  
 **return** reserva\_cobertura

### CHECKLIST 6a

This is the most difficult checklist and the most time consuming. Therefore, we have to make some tricks to simplify the future automatization process. I will explain step by step to have a deep understanding.

When a sinister comes, the Investigation Officer analyzes what type of coverages are involved. If that coverage is associated with electrical damages, they check in the internet if it was raining in the sinister day (in that specific location).

Here we have two basic problems. We need to get the specific day and we need to get the specific location. With this information we have to search in the internet. This manually process is not too much efficient.

We have counted how many cases we have with this type of coverage. They are more than 36,000. Imaging search manually. In our developed algorithm we are spending 1 second per each sinister. It seems really faster, but this implies ten hours of web scrapping.

However, in the future, we will just analyze a few cases per day. Perhaps none. Therefore, the first time we are going to spent that 10 hours to create an auxiliary database that matches each sinister.

We are going to scrap the meteostat.net web. This is a very good one, because even if our input city name is not totally correct, this page approximate to the better results. Also, in cases with similar city names, it returns the best cases, so then we can research which corresponds to Spain for example.

The problem is that we cannot enter directly. First, we have to enter to a proxy server to the main page. Search for the name. Choose which the true corresponding name is. And then, we have to move to the daily data, specifying which the correct day is. This several steps can have some troubles. Even, we can get nothing. Therefore, we have to make many handly error efforts. Let me start.

The first step is to get three bottles: Fecha, Hogar and Reserva. We need Fecha because it contains the sinister occurance. Hogar contains the location and Reserva contains the coverage involved in the sinister. Also, we load an auxiliar file which contains the id\_siniestro, and two variables related to this checklist. The first time it will be empty, so we have to add a FileNotFound Exception. Once we have this implemented, this file will contain the core basis of sinister. Then we just need to add the new sinister variables.

So, here we load this auxiliary file:

**try**:  
 auxiliar\_file = ReadCsv.load\_csv(STRING.auxiliar\_weather)  
 auxiliar\_file = DfUtils.processing\_file(auxiliar\_file, delimiter=';')  
**except** FileNotFoundError:  
 auxiliar\_file = pd.DataFrame(columns=['id\_siniestro', 'checklist6a', 'checklist6a\_PP'])

We add a variable to identify which sinister has already values:

auxiliar\_file\_reduced = auxiliar\_file[['id\_siniestro']]  
auxiliar\_file\_reduced['sinister\_exist'] = pd.Series(1, index=auxiliar\_file\_reduced.index)

From Fecha bottle we get Year, Month and Day to search later in the website.

df\_fecha\_test = df\_fecha\_test[['id\_siniestro', 'fecha\_siniestro\_ocurrencia']]  
df\_fecha\_test['fecha\_siniestro\_ocurrencia'] = pd.to\_datetime(df\_fecha\_test['fecha\_siniestro\_ocurrencia'],  
 format='%Y-%m-%d', errors='coerce')  
df\_fecha\_test['year'] = pd.DatetimeIndex(df\_fecha\_test['fecha\_siniestro\_ocurrencia']).year  
df\_fecha\_test['month'] = pd.DatetimeIndex(df\_fecha\_test['fecha\_siniestro\_ocurrencia']).month  
df\_fecha\_test['day'] = pd.DatetimeIndex(df\_fecha\_test['fecha\_siniestro\_ocurrencia']).day

From Hogar we obtain ‘hogar\_poblacion’ variable that returns the City/Town associated with the insured object.

df\_hogar\_test = df\_hogar\_test[['auditCodigoSiniestroReferencia', 'hogar\_poblacion']]  
df\_hogar\_test.columns = ['id\_siniestro', 'hogar\_poblacion']

We bring the Reserva bottle and we search the coverages associated with Electrical incidences. We also add other incidences that we think they can give us information. Basically, the choosen coverages are:

-DAÑOS ELECTRICOS

-RAYOS

-PEDRISCO

-LLUVIA

This last three are associated to Atmosferical Damages and not to Electrical damages.

We have a total of 36733.

df\_reserva\_test = df\_reserva\_test[['id\_siniestro', 'po\_res\_cobertura']]  
  
df\_reserva\_test.loc[df\_reserva\_test['po\_res\_cobertura'].str.contains('ELECT'), 'po\_res\_cobertura'] = 'INCIDENCIA'  
df\_reserva\_test.loc[df\_reserva\_test['po\_res\_cobertura'] == 'DE', 'po\_res\_cobertura'] = 'INCIDENCIA'  
df\_reserva\_test.loc[df\_reserva\_test['po\_res\_cobertura'].str.contains('RAYO'), 'po\_res\_cobertura'] = 'INCIDENCIA'  
df\_reserva\_test.loc[df\_reserva\_test['po\_res\_cobertura'].str.contains('PEDRISCO'), 'po\_res\_cobertura'] = 'INCIDENCIA'  
df\_reserva\_test.loc[df\_reserva\_test['po\_res\_cobertura'].str.contains('LLUVIA'), 'po\_res\_cobertura'] = 'INCIDENCIA'  
  
df\_reserva\_test = df\_reserva\_test[df\_reserva\_test['po\_res\_cobertura'] == 'INCIDENCIA']  
  
df\_reserva\_test = df\_reserva\_test.drop\_duplicates(subset=['id\_siniestro', 'po\_res\_cobertura'], keep='last')

The second step is to cross the tables and generate additional variables that are the variables we are going to fill. This variables are: ‘checklist6a\_PP’ indicates how many milimiters has rained, ‘checklist\_6a’=1 if there is an incidence but ‘checklist6a\_PP’ = 0.

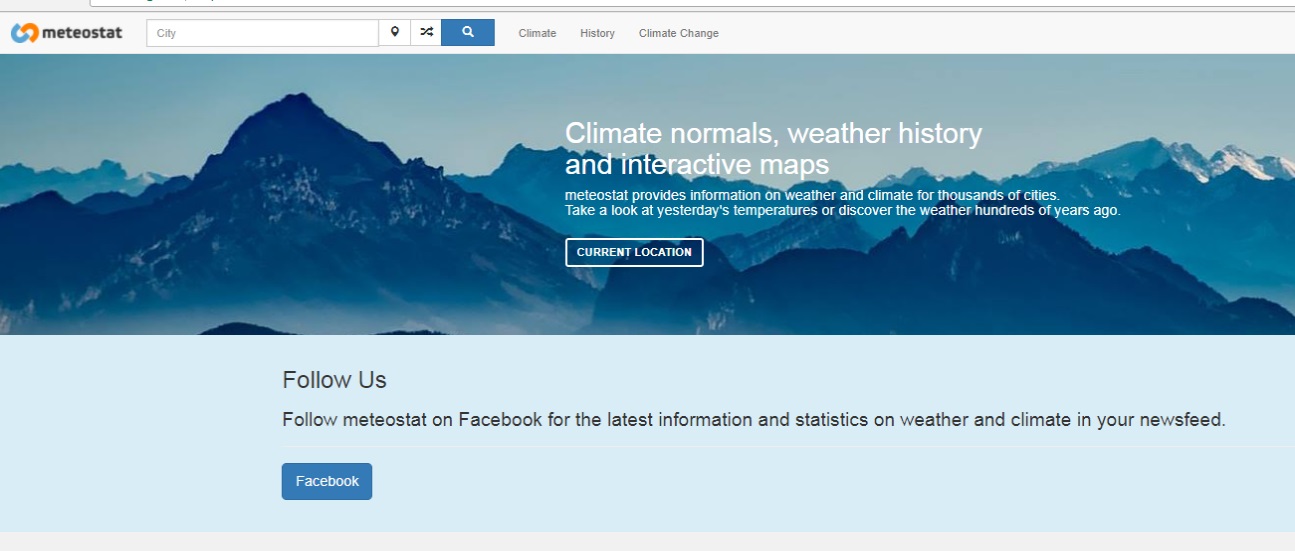
df\_hogar\_test['id\_siniestro'] = df\_hogar\_test['id\_siniestro'].map(int)  
df\_fecha\_test['id\_siniestro'] = df\_fecha\_test['id\_siniestro'].map(int)  
df\_reserva\_test['id\_siniestro'] = df\_reserva\_test['id\_siniestro'].map(int)  
  
file\_df = pd.merge(df\_fecha\_test, df\_hogar\_test, how='left', on='id\_siniestro')  
file\_df = pd.merge(file\_df, df\_reserva\_test, how='left', on='id\_siniestro')  
file\_df = file\_df.dropna(subset=['hogar\_poblacion'], how='any')  
  
file\_df['checklist6a'] = pd.Series(0, index=file\_df.index)  
file\_df['checklist6a\_PP'] = pd.Series(0, index=file\_df.index)  
file\_df = file\_df[file\_df['po\_res\_cobertura'] == 'INCIDENCIA'].reset\_index(drop=**True**)  
  
file\_df = pd.merge(file\_df, auxiliar\_file\_reduced, how='left', on='id\_siniestro')  
file\_df = file\_df[file\_df['sinister\_exist'] != 1]  
**del** file\_df['sinister\_exist']  
**del** auxiliar\_file\_reduced

We are going to iterate over each row of the Dataframe using iterrows and the local variables index and row as the row and the index of the Dataframe, respectively.

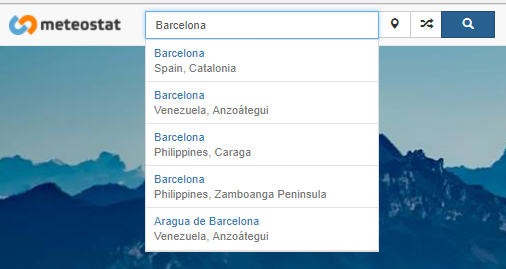
Therefore, we are going to watch the code for just one row but that is extensively to every row.

We set two local variables https = 'https://www.meteostat.net/search' and the proxy server variable setted in ‘proxies’.

This is the website:



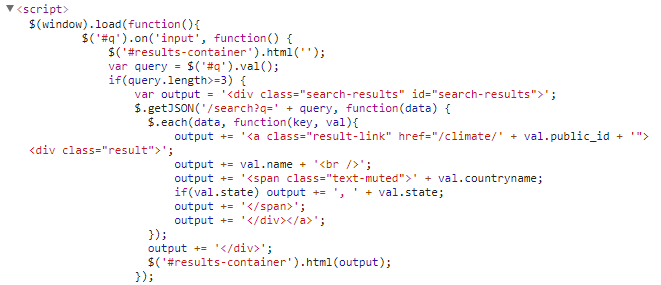
We need to input a value in the input box and make a search. The web however return several options for the cities. So first, we need to make a choice.



When we input a name in the box, dynamically the web is using the action Search:



As you can see, we have an input text type which name is ‘q’, an form with action ‘/search’ using the ‘GET’ method. Basically is using the following JSON script:



Which will search any query ‘q’ and will put it in the result-container div.

With this in mind, we have to replicate this query for each row and get the possible options that are returned.

We will name ‘q’ as the ‘hogar\_poblacion’ variable. We need to correct some Ñ encoding mistakes in the bottle before.

Then we open a Session method. And we loop a connection with the website to avoid HttpConnections errors. This connection is made with a get method and using as parameter our query variable.

connected = **False  
while not** connected:  
 **try**:  
 page = session.get(https, params=data, proxies=proxies)  
 connected = **True  
 except**:  
 **pass**page = page.text

Here start the conditionals to overcome any error.

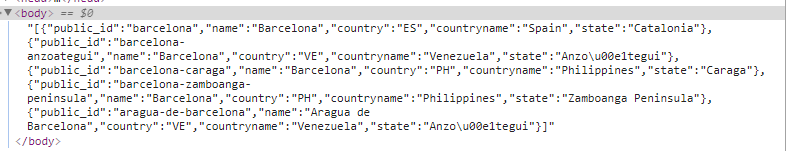
The first possibility is that we do not get any result. This is frequent when we have really small towns (with population up to 1,000). Also we have some troubles with the Spanish islands.

In this case the web return a null value.

What we do here is to call a dictionary which correct most of this values. Obviously, it is imposible to overcome every new possibility. If the value exist in the dict, it will try a new connection with the website. If not, will assing a None value to ‘q’.

**if** page == 'null':  
 **del** page  
 **del** session  
 **del** data  
 dict = STRING.Parameters.correct\_city\_names  
 exist = dict.get(q)  
 **if** exist:  
 q = dict[q]  
 data = {  
 'q:': str(q)  
 }  
 session = requests.Session()  
 connected = **False  
 while not** connected:  
 **try**:  
 page = session.get(https, params=data, proxies=proxies)  
 connected = **True** page = page.text  
 **except**:  
 **pass  
 else**:  
 q = **None**

If q is not None, it implies the website has give us an answer. The web returns a kind of dictionary as follows:



With this, we construct a table wich contains:

-public\_id: The name associated with the html.

-name: City Name.

-Country: Country code name.

-countryname: Country name.

-state: State name.

Once we have the table we just need to take the ‘ES’ options and get the public\_id which will serve us to go directly to the daily stats.

**if** q **is not None**:  
 # If q exists we have to check if it is corresponds to SPAIN  
 page = page.replace('}', ']')  
 page = page.replace('{', '[')  
 page = page.replace(':', '')  
 head = ['public\_id', 'name', 'country', 'countryname', 'state']  
 **for** h **in** head:  
 page = page.replace(h, '')  
 page = re.sub('\"\"', '', page)  
 page = page.replace('null', "0")  
  
 page = ast.literal\_eval(page)  
  
 page = pd.DataFrame(page, columns=head)  
 page = page[page['country'] == 'ES'].reset\_index(drop=**True**)  
  
 # Si existe en España traemos el primer valor  
 **if not** page.empty:  
 page = page.at[0, 'public\_id']

We have to create another conditional. There are exceptional cases that the city exists but it is not well returned. Therefore we have to harcode the URL site. They are very rare cases. We just found one in the first 10,000 cases. Anyway, we have to lead with it.

If ‘q’ is not empty, but when we construct the table we find that ‘ES’ does not exist. We have two cases. This rare cases where the City actually exists. And the cases which the City is related to other country.

We will fix the first cases. And we cannot do anything with the second cases but indicates it.

For the first cases, we go through the URL and write the city, and repeat the process of creating a table. The second cases are the result q = ‘None’

session = requests.Session()  
connected = **False  
while not** connected:  
 **try**:  
 page = session.get(https + '?q=' + str(q), proxies=proxies)  
 connected = **True** page = page.text  
 **except**:  
 **pass**page = page.replace('}', ']')  
page = page.replace('{', '[')  
page = page.replace(':', '')  
head = ['public\_id', 'name', 'country', 'countryname', 'state']  
**for** h **in** head:  
 page = page.replace(h, '')  
page = re.sub('\"\"', '', page)  
page = page.replace('null', "0")  
  
page = ast.literal\_eval(page)  
  
page = pd.DataFrame(page, columns=head)  
page = page[page['country'] == 'ES'].reset\_index(drop=**True**)  
**if not** page.empty:  
 page = page.at[0, 'public\_id']  
**else**:  
 q = **None**

Once we have the ‘public\_id’ of the city. We are able to access to its daily data. Lucky to us, when we enter in the city weather html, in the source code is href link we need:



Therefore, we just need to reach that href link.

As you can see, this link is composed by ‘history/daily/’ + the weather station number + Year + Month.

Every time we access to a City, this city has a assigned the nearest weather station. With this information plus our Sinister Occurance date we can reach the final daily html.

Using BeatifulSoup lib, we retrieve every href link.

soup = BeautifulSoup(page.content, 'html.parser')  
links = soup.find\_all('a', href=**True**)  
link\_text = []  
**for** a **in** links:  
 **if** a.text.strip():  
 link\_text.append(a['href'])

And we just keep the daily link.

link\_text = [i **for** i **in** link\_text **if** '/history/daily' **in** i]  
link\_text = list(set(link\_text))

There are some cases that does not have daily data. In consequence, we have to mark that cases.

For the remaining cases we get something like this:

[/history/daily/08181/2016-09](https://www.meteostat.net/history/daily/08181/2016-09)

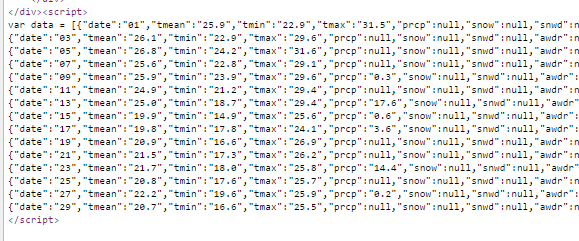
But we want to choose the date. So we need to substract the last seven values and assign the Month and Year related to the sinister.

**if not** link\_text:  
 row['checklist6a'] = 'NOT-INFORMED'  
**if** row['checklist6a'] != 'NOT-INFORMED':  
 link\_daily = link\_text[0]  
 link\_daily = link\_daily[:-7]  
  
 # With the id, now we can go to the daily history  
 year = row['year']  
 month = row['month']  
 day = row['day']  
  
 **if** len(str(month)) < 2:  
 month = '0' + str(month)  
  
 **if** len(str(day)) < 2:  
 day = '0' + str(day)  
  
 https = 'https://www.meteostat.net/' + link\_daily + str(year) + '-' \  
 + str(month)

Then, again, we make a new connection.

connected = **False  
while not** connected:  
 **try**:  
 page = session.get(https, proxies=proxies)  
 connected = **True  
 except**:  
 **pass**

The data is saved in a script section, within a list name ‘var data’. Here is all the monthly data choosen for the location.



We have to call for the specified day and the ‘prcp’ values. For that we construct a table with the whole days and columns (We have also to clean the data):

soup = BeautifulSoup(page.content, 'html.parser')  
rain = soup.find\_all('script')  
  
rain = [i.get\_text().strip() **for** i **in** rain **if** 'var data' **in** i.get\_text()]  
rain = str(rain)  
rain = rain.replace('var data = ', '')  
rain = rain.replace('}', ']')  
rain = rain.replace('{', '[')  
rain = rain.replace(':', '')  
rain = rain.replace(';', '')  
  
rain = rain[2:]  
rain = rain[:-2]  
**if** rain != '':  
 head = ['date', 'tmean', 'tmin', 'tmax', 'prcp', 'snow', 'snwd', 'awdr', 'awnd', 'tsun', 'wsfg']  
 **for** i **in** head:  
 rain = rain.replace(i, '')  
 rain = re.sub('\"\"', '', rain)  
  
 rain = rain.replace('null', '"0"')  
 rain = ast.literal\_eval(rain)  
  
 df = pd.DataFrame(rain, columns=head)  
 df = df[df['date'] == str(day)].reset\_index(drop=**True**)

Finally we obtain the rain value and put it on ‘checklist6a\_PP’. If rain==0, checklist6a = 1.

**try**:  
 rain = df.at[0, 'prcp']  
 **except** KeyError:  
 rain = 0  
  
 row['checklist6a\_PP'] = rain  
  
 **if** rain == 0:  
 row['checklist6a'] = 1  
  
**else**:  
 row['checklist6a'] = 'NOT-INFORMED'

Once we have each row, we redefine the auxiliary file.

file\_df = file\_df[['id\_siniestro', 'checklist6a', 'checklist6a\_PP']]  
auxiliar\_file = pd.concat([auxiliar\_file, file\_df], axis=0)  
  
auxiliar\_file.to\_csv(STRING.auxiliar\_weather, sep=';', encoding='latin1', index=**False**)  
**return** auxiliar\_file

### CHECKLIST 6b

It is a continuation of checklist6. We have to check if the person had two or more electrical damages sinister in the last year.

Again, we have to search in each sinister date row. Then, we create a temporal window between after and before this date. Always searching for electrical damages.

Finally we count how many sinister in that window are.

**def checklist6b**(df\_fecha\_test:pd.DataFrame, df\_fecha:pd.DataFrame, df\_reserva\_test:pd.DataFrame, df\_reserva:pd.DataFrame):  
 *"""  
 y 2 o mas reclamaciones por daños   
 eléctricos en el año.* ***:param*** *df\_test\_id:* ***:return****: This return a Dataframe with the columns 'id\_siniestro', 'checklist6b', where   
 'checklist6b' counts how many Electrical sinister the person has (Temporal Space = 1 year before the   
 sinister occurance)  
 """* # First we reduce the universe of sinister by RESERVA = DAÑOS ELECTRICOS  
 **if** df\_reserva\_test **is not None**:  
 df\_reserva\_test = df\_reserva\_test[['id\_siniestro', 'po\_res\_cobertura']]  
 df\_reserva\_base = df\_reserva[['id\_siniestro', 'po\_res\_cobertura']]  
 **del** df\_reserva\_test  
 **del** df\_reserva\_base  
 df\_reserva = pd.concat([df\_reserva\_base, df\_reserva\_test], axis=0)  
  
 df\_reserva = df\_reserva[['id\_siniestro', 'po\_res\_cobertura']]  
 df\_reserva = df\_reserva.drop\_duplicates(subset=['id\_siniestro', 'po\_res\_cobertura'])  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('ELECT'), 'po\_res\_cobertura'] = 'INCIDENCIA'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'] == 'DE', 'po\_res\_cobertura'] = 'INCIDENCIA'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('RAYO'), 'po\_res\_cobertura'] = 'INCIDENCIA'  
  
 df\_reserva = df\_reserva[df\_reserva['po\_res\_cobertura'] == 'INCIDENCIA']  
  
 # We bring the test bottle and the Date Bottle (because we need the past sinister)  
 **if** df\_fecha\_test **is not None**:  
 df\_fecha\_test = df\_fecha\_test[['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia']]  
 df\_fecha\_base = df\_fecha[['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia']]  
 **del** df\_fecha\_test  
 **del** df\_fecha\_base  
  
 df\_fecha = pd.concat([df\_fecha\_base, df\_fecha\_test], axis=0)  
  
 df\_fecha = df\_fecha[['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia']]  
 df\_fecha = df\_fecha.drop\_duplicates(subset=['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia'])  
 # We cross to the left of df\_reserva to get only the sinister with INCIDENCIA  
 df\_reserva['id\_siniestro'] = df\_reserva['id\_siniestro'].map(int)  
 df\_fecha['id\_siniestro'] = df\_fecha['id\_siniestro'].map(int)  
 df = pd.merge(df\_reserva, df\_fecha, on='id\_siniestro', how='left')  
 **del** df\_reserva  
 **del** df\_fecha  
  
 df['id\_poliza'] = df['id\_poliza'].map(str)  
 df['fecha\_siniestro\_ocurrencia'] = pd.to\_datetime(df['fecha\_siniestro\_ocurrencia'],  
 format='%Y-%m-%d', errors='coerce')  
  
 df['checklist\_6b'] = pd.Series(0, index=df.index)  
  
 **for** index, row **in** df.iterrows():  
 fecha\_ocurrencia = row['fecha\_siniestro\_ocurrencia']  
 poliza = row['id\_poliza']  
  
 # Filtramos la póliza que estamos analizando  
 df\_i = df[df['id\_poliza'] == poliza]  
  
 # Ahora filtramos un año para adelante y un año para atrás  
 df\_i = df\_i[df\_i['fecha\_siniestro\_ocurrencia'] <= fecha\_ocurrencia + datetime.timedelta(days=365)]  
  
 df\_i = df\_i[df\_i['fecha\_siniestro\_ocurrencia'] >= fecha\_ocurrencia - datetime.timedelta(days=365)]  
 df\_i = df\_i.drop\_duplicates(subset=['id\_siniestro'])  
  
 count\_sinister = df\_i['id\_siniestro'].count()  
 row['checklist\_6b'] = count\_sinister  
  
 df = df[['id\_siniestro', 'checklist\_6b']]  
  
 **return** df

### CHECKLIST 7

We have to count how many sinister are associated to mugging. Therefore we reduce our universe by filtering robbery coverages involved.

Then, we create a year window we are we check how many sinister appears in each row (using the policy as a key value).

**def checklist7**(df\_fecha\_test:pd.DataFrame, df\_fecha: pd.DataFrame, df\_reserva\_test: pd.DataFrame,  
 df\_reserva: pd.DataFrame):  
 *"""Un siniestro por anualidad repetitivo (atraco)"""* # Es parecido al de joyas. pero no me queda claro. Tengo que ver si tiene un atraco por año?  
  
 # First we reduce the universe of sinister by RESERVA = ATRACOS o EXPOLIACION  
 **if** df\_reserva\_test **is not None**:  
 df\_reserva\_test = df\_reserva\_test[['id\_siniestro', 'po\_res\_cobertura']]  
 df\_reserva\_base = df\_reserva[['id\_siniestro', 'po\_res\_cobertura']]  
 **del** df\_reserva\_test  
 **del** df\_reserva\_base  
 df\_reserva = pd.concat([df\_reserva\_base, df\_reserva\_test], axis=0)  
  
 df\_reserva = df\_reserva[['id\_siniestro', 'po\_res\_cobertura']]  
 df\_reserva = df\_reserva.drop\_duplicates(subset=['id\_siniestro', 'po\_res\_cobertura'])  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('ATR'), 'po\_res\_cobertura'] = 'INCIDENCIA'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('EXPO'), 'po\_res\_cobertura'] = 'INCIDENCIA'  
  
 df\_reserva = df\_reserva[df\_reserva['po\_res\_cobertura'] == 'INCIDENCIA']  
  
 # We bring the test bottle and the Date Bottle (because we need the past sinister)  
 **if** df\_fecha\_test **is not None**:  
 df\_fecha\_test = df\_fecha\_test[['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia']]  
 df\_fecha\_base = df\_fecha[['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia']]  
 **del** df\_fecha\_test  
 **del** df\_fecha\_base  
  
 df\_fecha = pd.concat([df\_fecha\_base, df\_fecha\_test], axis=0)  
  
 df\_fecha = df\_fecha[['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia']]  
 df\_fecha = df\_fecha.drop\_duplicates(subset=['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia'])  
  
 # We cross to the left of df\_reserva to get only the sinister with INCIDENCIA  
 df\_reserva['id\_siniestro'] = df\_reserva['id\_siniestro'].map(int)  
 df\_fecha['id\_siniestro'] = df\_fecha['id\_siniestro'].map(int)  
 df = pd.merge(df\_reserva, df\_fecha, on='id\_siniestro', how='left')  
 **del** df\_reserva  
 **del** df\_fecha  
  
  
 df['id\_poliza'] = df['id\_poliza'].map(str)  
 df['fecha\_siniestro\_ocurrencia'] = pd.to\_datetime(df['fecha\_siniestro\_ocurrencia'],  
 format='%Y-%m-%d', errors='coerce')  
  
 df['checklist\_7'] = pd.Series(0, index=df.index)  
  
 **for** index, row **in** df.iterrows():  
 fecha\_ocurrencia = row['fecha\_siniestro\_ocurrencia']  
 poliza = row['id\_poliza']  
  
  
 # Filtramos la póliza que estamos analizando  
 df\_i = df[df['id\_poliza'] == poliza]  
  
 # Ahora filtramos un año para adelante y un año para atrás  
 df\_i = df\_i[df\_i['fecha\_siniestro\_ocurrencia'] <= fecha\_ocurrencia + datetime.timedelta(days=365)]  
 df\_i = df\_i[df\_i['fecha\_siniestro\_ocurrencia'] >= fecha\_ocurrencia - datetime.timedelta(days=365)]  
 df\_i = df\_i.drop\_duplicates(subset=['id\_siniestro'])  
  
 count\_sinister = df\_i['id\_siniestro'].count()  
 row['checklist\_7'] = count\_sinister  
  
  
 df = df[['id\_siniestro', 'checklist\_7']]  
  
 **return** df

## CHECKLIST\_VALORACION CLASS

### CHECKLIST 14

Actually, this is the only checklist we can assess. It is referred to sinister that are repetitives. We have to check 3 similar sinister within 3 months.

It ise exactly the same code as Checklist 7. The only difference is that we use 3 months, and that we also have to iterate each coverage type.

**def checklist14**(df\_fecha\_test:pd.DataFrame, df\_fecha:pd.DataFrame, df\_reserva\_test:pd.DataFrame, df\_reserva:pd.DataFrame):  
 *"""Reiteración de 3 o + siniestros parecidos en el mismo riesgo en un plazo de tres meses"""* # Es parecido al de joyas. pero no me queda claro. Tengo que ver si tiene un atraco por año?  
  
 # First we reduce the universe of sinister by RESERVA = ATRACOS o EXPOLIACION  
 **if** df\_reserva\_test **is not None**:  
 df\_reserva\_test = df\_reserva\_test[['id\_siniestro', 'po\_res\_cobertura']]  
 df\_reserva\_base = df\_reserva[['id\_siniestro', 'po\_res\_cobertura']]  
 **del** df\_reserva\_test  
 **del** df\_reserva\_base  
 df\_reserva = pd.concat([df\_reserva\_base, df\_reserva\_test], axis=0)  
  
 df\_reserva = df\_reserva[['id\_siniestro', 'po\_res\_cobertura']]  
 df\_reserva = df\_reserva.drop\_duplicates(subset=['id\_siniestro', 'po\_res\_cobertura'])  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('PORCALOR'), 'po\_res\_cobertura'] = 'CALOR'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('VALLAS'), 'po\_res\_cobertura'] = 'VALLAS'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('OTR'), 'po\_res\_cobertura'] = 'OTRO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('RC'), 'po\_res\_cobertura'] = 'RC'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('ROBO'), 'po\_res\_cobertura'] = 'ROBO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('HURTO'), 'po\_res\_cobertura'] = 'ROBO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('ATR'), 'po\_res\_cobertura'] = 'ROBO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('EXPO'), 'po\_res\_cobertura'] = 'ROBO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('CRIS'), 'po\_res\_cobertura'] = 'CRISTALES'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('INUNDA'), 'po\_res\_cobertura'] = 'INUNDACION'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('ELECT'), 'po\_res\_cobertura'] = 'ELECTRICIDAD'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'] == 'DE', 'po\_res\_cobertura'] = 'ELECTRICIDAD'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('CHOQUE'), 'po\_res\_cobertura'] = 'CHOQUE'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('INC'), 'po\_res\_cobertura'] = 'INCENDIO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('AGUA'), 'po\_res\_cobertura'] = 'AGUA'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('RAYO'), 'po\_res\_cobertura'] = 'ATMOSFERICO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('VIENTO'), 'po\_res\_cobertura'] = 'ATMOSFERICO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('LLUVIA'), 'po\_res\_cobertura'] = 'ATMOSFERICO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('PEDRISCO'), 'po\_res\_cobertura'] = 'ATMOSFERICO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('JUR'), 'po\_res\_cobertura'] = 'DEF\_JURIDICA'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('LLAVE'), 'po\_res\_cobertura'] = 'LLAVES'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('VANDAL'), 'po\_res\_cobertura'] = 'VANDALISMO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('ALIM'), 'po\_res\_cobertura'] = 'ALIMENTOS'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('EXC.'), 'po\_res\_cobertura'] = 'CONTENIDO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('JOY'), 'po\_res\_cobertura'] = 'JOYAS'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('METALICO'), 'po\_res\_cobertura'] = 'METALICO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('VITRO'), 'po\_res\_cobertura'] = 'VITROCERAMICA'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('ACC'), 'po\_res\_cobertura'] = 'ACCIDENTE'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('VV\_EXT'), 'po\_res\_cobertura'] = 'EXTERIOR'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('ESTET'), 'po\_res\_cobertura'] = 'ESTETICA'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('DAEST'), 'po\_res\_cobertura'] = 'ESTETICA'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('DEST'), 'po\_res\_cobertura'] = 'ESTETICA'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('RL'), 'po\_res\_cobertura'] = 'RL'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('INMB'), 'po\_res\_cobertura'] = 'INMUEBLE'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('MAT'), 'po\_res\_cobertura'] = 'MATERIAL'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('ALQ'), 'po\_res\_cobertura'] = 'ALQUILER'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('SAN'), 'po\_res\_cobertura'] = 'SANITARIO'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('FRAUD'), 'po\_res\_cobertura'] = 'FRAUDE\_TARJETAS'  
 df\_reserva.loc[df\_reserva['po\_res\_cobertura'].str.contains('CDO'), 'po\_res\_cobertura'] = 'CONTENIDO'  
  
  
 # We bring the test bottle and the Date Bottle (because we need the past sinister)  
 **if** df\_fecha\_test **is not None**:  
 df\_fecha\_test = df\_fecha\_test[['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia']]  
 df\_fecha\_base = df\_fecha[['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia']]  
 **del** df\_fecha\_test  
 **del** df\_fecha\_base  
  
 df\_fecha = pd.concat([df\_fecha\_base, df\_fecha\_test], axis=0)  
  
 df\_fecha = df\_fecha[['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia']]  
 df\_fecha = df\_fecha.drop\_duplicates(subset=['id\_siniestro', 'id\_poliza', 'fecha\_siniestro\_ocurrencia'])  
  
 # We cross to the left of df\_reserva to get only the sinister with INCIDENCIA  
 df\_reserva['id\_siniestro'] = df\_reserva['id\_siniestro'].map(int)  
 df\_fecha['id\_siniestro'] = df\_fecha['id\_siniestro'].map(int)  
 df = pd.merge(df\_reserva, df\_fecha, on='id\_siniestro', how='left')  
 **del** df\_reserva  
 **del** df\_fecha  
  
 df['id\_poliza'] = df['id\_poliza'].map(str)  
 df['fecha\_siniestro\_ocurrencia'] = pd.to\_datetime(df['fecha\_siniestro\_ocurrencia'],  
 format='%Y-%m-%d', errors='coerce')  
  
 df['checklist\_14'] = pd.Series(0, index=df.index)  
  
 **for** index, row **in** df.iterrows():  
 fecha\_ocurrencia = row['fecha\_siniestro\_ocurrencia']  
 poliza = row['id\_poliza']  
 cobertura = row['po\_res\_cobertura']  
  
 # Filtramos la póliza que estamos analizando  
 df\_i = df[df['id\_poliza'] == poliza]  
 df\_i = df\_i[df\_i['po\_res\_cobertura'] == cobertura]  
  
 # Ahora filtramos un año para adelante y un año para atrás  
 df\_i = df\_i[df\_i['fecha\_siniestro\_ocurrencia'] <= fecha\_ocurrencia + datetime.timedelta(days=93)]  
 df\_i = df\_i[df\_i['fecha\_siniestro\_ocurrencia'] >= fecha\_ocurrencia - datetime.timedelta(days=93)]  
 df\_i = df\_i.drop\_duplicates(subset=['id\_siniestro'])  
  
 count\_sinister = df\_i['id\_siniestro'].count()  
  
 row['checklist\_14'] = count\_sinister  
  
 df = df[['id\_siniestro', 'checklist\_14']]

# UTILS

It contains modules associated to useful tolos applied all over the Project.

## FUZZY RULES

Here is a partial Fuzzy Rules implementation package. As we are using actually just the NIF corrector, we simplified the package. Here you can finde the nif\_corrector module which is the module that is employed on several bottles.

It has several methods which try to identify id types and then to calculate if the value is correct.

It is a bit large because it has several conditions, therefore we beg to check it into the project source.

## DATAFRAME UTILS

It standarizes some processes associated with dataframes variables. In general they are simple processes that are better writted in a line. You can check the code for eac method explanation:

**def load\_df\_file**(self):  
 *"""* ***:return****: It returns a CSV as a DF  
 """* file\_load = pd.read\_csv(self, delimiter=';', encoding='latin1')  
 **return** file\_load  
  
**def statistic\_df**(self, output = **False**):  
  
 *"""  
 This returns a describe() and a null analysis of the file used as input.  
 Also, if output = True, it returns two CSV files in doc\_output\statistics  
 wit names describe.csv and null.csv  
 """* print('DESCRIBE-------------------------')  
 describe\_df = self.describe(include = 'all')  
 print(describe\_df)  
 print(' ')  
  
 print('NULL VALUES-----------------------')  
  
 null = self.isnull().sum()  
 print(null)  
  
 print(' ')  
 print('INFO-----------------------------')  
 print(self.info())  
 print(' ')  
  
 **if** output == **True**:  
 **import** os  
 **import** time  
  
 os.chdir(STRING.path\_project)  
  
  
 timestr = time.strftime('%Y%m%d-%H%M%S')  
  
 name\_file = 'doc\_output\statistics\\' + timestr +'\_describe.csv'  
 null\_file = 'doc\_output\statistics\\' + timestr + '\_null.csv'  
 print('File Exported as ' + name\_file)  
  
 describe\_df.to\_csv(name\_file, sep =';', header= **True**, index=**True**,encoding='latin1')  
 null.to\_csv(null\_file, sep=';', header=**True**, index=**True**, encoding='latin1')  
  
**def del\_var**(names: list, df):  
 *"""  
 It takes a list of names and drop each of them.* ***:param*** *df: input Dataframe* ***:return****: Dataframe without the columns listed.  
 """* df = df.drop(names, axis = 1)  
  
 **return** df  
  
**def df\_fillna**(names: list, df, value):  
 *"""  
 Fill each cell nan value of the list name with the values 'value'* ***:param*** *df: input dataframe* ***:param*** *value: Value used as inputation value* ***:return****: Dataframe with values replaced  
 """* **for** i **in** names:  
 df[i] = df[i].fillna(value)  
  
 **return** df  
  
**def string\_categoric**(names: list, var\_grouped: str, df, type\_agg = 'count'):  
 *"""  
 It takes a list of variables and print a groupby by a choosen var\_grouped variable.* ***:param*** *var\_grouped: Variables that want to be showed in the groupby* ***:param*** *df: input Dataframe* ***:param*** *type\_agg: Operation type. 'count' as default.* ***:return****: It prints the result  
  
 Eg:  
 DfUtils.string\_categoric(names, 'ID\_DOSSIER', file)  
 This will take each name from the list 'names' and will be groupby 'ID\_DOSSIER'  
 with an operation default = 'count' using 'file' Dataframe as inpùt.  
  
 """* **for** i **in** names:  
 df = df.drop\_duplicates(subset = [var\_grouped], keep ='last')  
 count\_motivo = df.groupby(i)[var\_grouped].agg([type\_agg])  
 print(count\_motivo)  
  
**def values\_variables**(df, path):  
 *"""  
 It takes a DF and it returns the unique values that take each column.* ***:return****: Return a csv to the path indicated with each column  
 """* list\_values = []  
  
 **for** i **in** df.columns:  
 **if** (df[i].dtype == np.float64 **or** df[i].dtype == np.int64):  
 values = df[i].unique()  
  
 list\_values.append([i, values])  
  
 write\_csv.WriteCsv.write\_csv(list\_values, path)  
  
**def processing\_file**(file, delimiter=',', nan\_values='?'):  
 *"""  
 It loads and gives a csv format to the BOTTLE raw file  
 """* list\_id = []  
  
 **with** file **as** csvfile:  
 reader\_line = csv.reader(csvfile, delimiter= delimiter, quotechar='"', quoting=csv.QUOTE\_ALL)  
 **for** lines **in** reader\_line:  
 list\_id.append(lines)  
  
 df = pd.DataFrame.from\_records(list\_id, index=**None**)  
  
 new\_header = df.iloc[0]  
 new\_header = new\_header.str.replace('\"', '')  
 new\_header = new\_header.str.replace(' ', '')  
 df = df[1:]  
 df = df.rename(columns=new\_header)  
 df = df.replace(nan\_values, np.nan)  
  
 **return** df  
  
  
**def processing\_file\_without\_header**(file, delimiter =',', first\_row = **True**):  
 *"""  
 It loads and gives a csv format to the BOTTLE raw file  
 """* list\_id = []  
  
 **with** file **as** csvfile:  
 reader\_line = csv.reader(csvfile, delimiter= delimiter, quotechar='"', quoting=csv.QUOTE\_ALL)  
 **for** lines **in** reader\_line:  
 list\_id.append(lines)  
  
 df = pd.DataFrame.from\_records(list\_id, index=**None**)  
 **if** first\_row == **False**:  
 df = df[1:]  
 df = df.replace('?', np.nan)  
  
 **return** df

## FRAUD SCORE

Please, refers to [Fraud Score Section](#_FRAUD_SCORE).

## GRAPH NETWORKS

Please, refers to [Graph Networks Section](#_GRAPHS_NETWORKS).

## MODEL UTILS

Here we have some complex code that helps to the models’ development.

a) plot\_confusion\_matrix: It plot a fancy confusion matrix using several pyplot options.

**def plot\_confusion\_matrix**(cm, classes, title='Confusion matrix', cmap=plot.cm.Blues):  
 *"""  
 This function prints and plots the confusion matrix.  
 Normalization can be applied by setting `normalize=True`.  
 Copyed from a kernel by joparga3 https://www.kaggle.com/joparga3/kernels  
 """* plot.figure()  
 plot.imshow(cm, interpolation='nearest', cmap=cmap)  
 plot.title(title)  
 plot.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plot.xticks(tick\_marks, classes, rotation=0)  
 plot.yticks(tick\_marks, classes)  
  
 thresh = cm.max() / 2.  
 **for** i, j **in** itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plot.text(j, i, cm[i, j],  
 horizontalalignment="center",  
 color="white" **if** cm[i, j] > thresh **else** "black")  
  
 plot.tight\_layout()  
 plot.ylabel('True label')  
 plot.xlabel('Predicted label')  
 plot.show()

b) over\_sampling: We may have to use some over sampling strategy. We actually do not know which the optimal strategy is. Therefore we create a flexible method to try several of them at once. For that we use imblearn lib, and try ADASYN and SMOTE methods over a theoretical Training Set using a ‘minority’ oversampling ratio.

**def over\_sampling**(xTrain, yTrain, model='ADASYN', neighbors=nbh.base.KNeighborsMixin):  
 *"""  
 It generate synthetic sampling for the minority class using the model specificed. Always it has  
 to be applied to the training set.* ***:param*** *xTrain: X training set.* ***:param*** *yTrain: Y training set.* ***:param*** *model: 'ADASYN' or 'SMOTE'* ***:param*** *neighbors: number of nearest neighbours to used to construct synthetic samples.* ***:return****: xTrain and yTrain oversampled  
 """* xTrainNames = xTrain.columns.values.tolist()  
 yTrainNames = yTrain.columns.values.tolist()  
  
 **if** model == 'ADASYN':  
 model = ADASYN(random\_state=42, ratio='minority', n\_neighbors=neighbors)  
  
 **if** model == 'SMOTE':  
 model = SMOTE(random\_state=42, ratio='minority', k\_neighbors=neighbors, m\_neighbors='svm')  
  
 Train, yTrain = model.fit\_sample(xTrain, yTrain)  
  
 xTrain = pd.DataFrame(xTrain, columns=[xTrainNames])  
 yTrain = pd.DataFrame(yTrain, columns=[yTrainNames])  
  
 **return** xTrain, yTrain

c) under\_sampling: It is the opposite to over sampling. Here we try to resize the majority class using AllKNN method.

**def under\_sampling**(xTrain, yTrain, neighbors=nbh.base.KNeighborsMixin):  
 *"""  
 It reduces the sample size for the majority class using the model specificed. Always it has  
 to be applied to the training set.* ***:param*** *xTrain: X training set.* ***:param*** *yTrain: Y training set.* ***:param*** *neighbors: size of the neighbourhood to consider to compute the  
 average distance to the minority point samples* ***:return****: xTrain and yTrain oversampled  
 """* xTrainNames = xTrain.columns.values.tolist()  
 yTrainNames = yTrain.columns.values.tolist()  
  
 model = AllKNN(random\_state=42, ratio='majority', n\_neighbors=neighbors)  
  
 Train, yTrain = model.fit\_sample(xTrain, yTrain)  
  
 xTrain = pd.DataFrame(xTrain, columns=[xTrainNames])  
 yTrain = pd.DataFrame(yTrain, columns=[yTrainNames])  
  
 **return** xTrain, yTrain

## OUTLIERS

Here there are several outlier methods. You can see the explanation in the [Outliers Section.](#_OUTLIERS)

**def outliers\_df**(file\_df, col\_name, not\_count\_zero = **True**, just\_count\_zero = **False**, smirnov = **True**):  
  
  
 file\_df\_col= file\_df[col\_name].dropna()  
 file\_df\_col = file\_df\_col.convert\_objects(convert\_numeric=**True**)  
  
 **if** not\_count\_zero == **True**:  
 file\_df\_col= file\_df\_col[file\_df\_col > 0]  
 **if** just\_count\_zero == **True**:  
 file\_df\_col = file\_df\_col[file\_df\_col >= 0]  
  
 file\_df[col\_name] = file\_df[col\_name].convert\_objects(convert\_numeric=**True**)  
  
 # Marcamos las colas de la distribución 0.05 y 0.95  
 list\_outlier = []  
 outlier\_percentile = Outliers.percentile\_based\_outlier(file\_df\_col)  
 **for** ax, func **in** zip(file\_df\_col, outlier\_percentile):  
 **if** func == **True**: # True is outlier  
 list\_outlier.append(ax)  
 list\_outlier = set(list\_outlier)  
  
 name = str(col\_name) + '\_outlier\_5\_95'  
 file\_df[name] = pd.Series(0, index=file\_df.index)  
 file\_df[name] = file\_df.apply(  
 **lambda** x: 1  
 **if** x[col\_name] **in** list\_outlier  
 **else** 0, axis=1)  
  
 **if** smirnov == **True**:  
 # Marcamos outliers con smirnov  
 max\_smirnov = smirnov\_grubbs.max\_test\_outliers(file\_df\_col, alpha=0.10)  
 min\_smirnov = smirnov\_grubbs.min\_test\_outliers(file\_df\_col, alpha=0.10)  
  
 **if** max\_smirnov:  
 max\_thresold = min(max\_smirnov)  
 name\_max = str(col\_name) + '\_max\_smirnov'  
 file\_df[name\_max] = pd.Series(0, index=file\_df.index)  
 file\_df[name\_max] = file\_df.apply(  
 **lambda** x: 1  
 **if** x[col\_name] < max\_thresold  
 **else** 0, axis=1)  
 **if** min\_smirnov:  
 min\_thresold = max(min\_smirnov)  
 name\_min = str(col\_name) + '\_min\_smirnov'  
 file\_df[name\_min] = pd.Series(0, index=file\_df.index)  
 file\_df[name\_min] = file\_df.apply(  
 **lambda** x: 1  
 **if** x[col\_name] < min\_thresold  
 **else** 0, axis=1)  
  
 #MAD  
 outliers\_mad = Outliers.mad\_based\_outlier(file\_df\_col)  
 list\_outlier = []  
 **for** ax, func **in** zip(file\_df\_col, outliers\_mad):  
 **if** func == **True**: # True is outlier  
 list\_outlier.append(ax)  
 list\_outlier = set(list\_outlier)  
 name = str(col\_name) + '\_mad\_outlier'  
 file\_df[name] = pd.Series(0, index=file\_df.index)  
 file\_df[name] = file\_df.apply(  
 **lambda** x: 1  
 **if** x[col\_name] **in** list\_outlier  
 **else** 0, axis=1)  
  
  
 **return** file\_df  
  
  
  
**def outliers\_mad**(file\_df, col\_name, not\_count\_zero = **True**, just\_count\_zero = **False**):  
  
 file\_df\_col = file\_df[col\_name].dropna()  
 file\_df\_col = pd.to\_numeric(file\_df\_col, errors='coerce')  
  
 file\_df\_col = file\_df\_col.fillna(file\_df\_col.median())  
  
 **if** not\_count\_zero:  
 file\_df\_col = file\_df\_col[file\_df\_col > 0]  
 **if** just\_count\_zero:  
 file\_df\_col = file\_df\_col[file\_df\_col >= 0]  
  
 file\_df[col\_name] = pd.to\_numeric(file\_df[col\_name], errors='coerce')  
 file\_df[col\_name] = file\_df[col\_name].fillna(file\_df[col\_name].median())  
  
 # MAD  
 outliers\_mad = Outliers.mad\_based\_outlier(file\_df\_col)  
 list\_outlier = []  
 **for** ax, func **in** zip(file\_df\_col, outliers\_mad):  
 **if** func: # True is outlier  
 list\_outlier.append(ax)  
 list\_outlier = set(list\_outlier)  
 name = str(col\_name) + '\_mad\_outlier'  
 file\_df[name] = pd.Series(0, index=file\_df.index)  
 file\_df[name] = file\_df.apply(  
 **lambda** x: 1  
 **if** x[col\_name] **in** list\_outlier  
 **else** 0, axis=1)  
  
 **return** file\_df  
  
  
**def mad\_based\_outlier**(points, thresh=3.5):  
 **if** len(points.shape) == 1:  
 points = points[:, **None**]  
  
 median = np.median(points, axis=0)  
  
 diff = np.sum((points - median) \*\* 2, axis=-1)  
  
 diff = np.sqrt(diff)  
  
 med\_abs\_deviation = np.median(diff)  
  
 modified\_z\_score = 0.6745 \* diff / med\_abs\_deviation  
  
 **return** modified\_z\_score > thresh  
  
  
**def percentile\_based\_outlier**(data, threshold=95):  
 diff = (100 - threshold) / 2.0  
 minval, maxval = np.percentile(data, [diff, 100 - diff])  
 **return** (data < minval) | (data > maxval)  
  
  
**def outliers\_test\_values**(file\_df, base\_df, col\_name, not\_count\_zero=**True**, just\_count\_zero=**False**):  
  
 # base\_df  
 base\_df\_col = base\_df[col\_name].dropna()  
 base\_df\_col = pd.to\_numeric(base\_df\_col, errors='coerce')  
  
 base\_df\_col = base\_df\_col.fillna(base\_df\_col.median())  
  
 **if** not\_count\_zero:  
 base\_df\_col = base\_df\_col[base\_df\_col > 0]  
 **if** just\_count\_zero:  
 base\_df\_col = base\_df\_col[base\_df\_col >= 0]  
  
 base\_df[col\_name] = pd.to\_numeric(base\_df[col\_name], errors='coerce')  
 base\_df[col\_name] = base\_df[col\_name].fillna(base\_df[col\_name].median())  
  
 # test df  
 file\_df\_col = file\_df[col\_name].dropna()  
 file\_df\_col = pd.to\_numeric(file\_df\_col, errors='coerce')  
 file\_df\_col = file\_df\_col[col\_name].dropna()  
  
 **if** not\_count\_zero:  
 file\_df\_col = file\_df\_col[file\_df\_col > 0]  
 **if** just\_count\_zero:  
 file\_df\_col = file\_df\_col[file\_df\_col >= 0]  
  
 # MAD  
 median, med\_abs\_deviation = Outliers.mad\_based\_outlier\_parameters(base\_df\_col)  
 **if** len(file\_df\_col.shape) == 1:  
 points = file\_df\_col[:, **None**]  
  
 diff = np.sum((points - median) \*\* 2, axis=-1)  
 diff = np.sqrt(diff)  
 modified\_z\_score = 0.6745 \* diff / med\_abs\_deviation  
  
 outliers\_mad = modified\_z\_score > 3.5  
  
 list\_outlier = []  
 **for** ax, func **in** zip(file\_df\_col, outliers\_mad):  
 **if** func: # True is outlier  
 list\_outlier.append(ax)  
 list\_outlier = set(list\_outlier)  
 name = str(col\_name) + '\_mad\_outlier'  
 file\_df[name] = pd.Series(0, index=file\_df.index)  
 file\_df[name] = file\_df.apply(  
 **lambda** x: 1  
 **if** x[col\_name] **in** list\_outlier  
 **else** 0, axis=1)  
  
 **return** file\_df  
  
**def mad\_based\_outlier\_parameters**(points):  
 **if** len(points.shape) == 1:  
 points = points[:, **None**]  
  
 median = np.median(points, axis=0)  
  
 diff = np.sum((points - median) \*\* 2, axis=-1)  
  
 diff = np.sqrt(diff)  
  
 med\_abs\_deviation = np.median(diff)  
  
 **return** median, med\_abs\_deviation

## READ\_CSV

Some standardized methods to read CSV files.

**def load\_csv**(file: str):  
 *"""  
 It load a file using read mode and a latin encoding.* ***:return****: A readed file  
 """* input\_file = open(file, 'r', newline='', encoding='latin1')  
 **return** input\_file  
  
**def processing\_txt\_without\_header**(file, separator=';', nan\_val='?', header=**False**, change\_decimal=**False**):  
 *"""  
 It loads a .txt file as a Dataframe and return a processed df without header  
 """* df = pd.read\_csv(file, sep=separator, header=**None**, encoding='latin1', quotechar='"')  
 **if** header == **True**:  
 df = df[1:]  
  
 **if** change\_decimal == **True**:  
 df = pd.read\_csv(file, sep=separator, header=**None**, encoding='latin1', quotechar='"', decimal=',')  
  
 df = df.replace(nan\_val, np.nan)  
  
 **return** df  
  
**def load\_blacklist**(file):  
 input\_file = open(file, 'r', newline='\n', encoding = 'latin1')  
 **return** input\_file

## TO\_DOCX

This is a model that was released in an earlier step. It basically output a docx document for the univariate analysis process. We are using docx library. Therefore, we ask you to see the [Univariate Analysis Section.](#_UNIVARIATE_ANALYSIS)

## UNIVARIATE ANALYSIS

Here we implement several basic statistics analysis. It is explained in [Data Understanding Section.](#_DATA_UNDERSTANDING)

## PROCESS\_UTILS

Please, refers to [Processing Table Section](#_PROCESSING_TABLE).

## TRAIN\_TEST\_UTILS

It is explanined in [Train-Valid-Test Section](#_TRAIN-VALID-TEST).

## WRITE CSV

The inverse of Read CSV. It simplifies the process of writing a CSV file using a list.

**def write\_csv**(input: list, output\_file\_path, header = **None**):  
  
 **with** open(output\_file\_path,'w', newline='') **as** file:  
 wr = csv.writer(file, delimiter = ';', quoting = csv.QUOTE\_ALL)  
 **if** header:  
 wr.writerow(header)  
  
 **for** i **in** input:  
 wr.writerow(i)

## CHECKLISTS

Please, refers to [Ckecklist Section](#_CHECKLISTS).

# EUROPA

# STRING

STRING.py is a module that contain each string used in the whole project. This simplify text modifications and also avoids harcoding practice. It is basically divided among several categories:

1) PATH FILES: Refer to the main paths inside the project.

2) INPUT FILES: The input file names.

3) OUTPUT FILES: The output file names.

4) TEST FILES: The new bottles file names.

5) AUXILIAR FILES: The auxiliary file names created during the process.

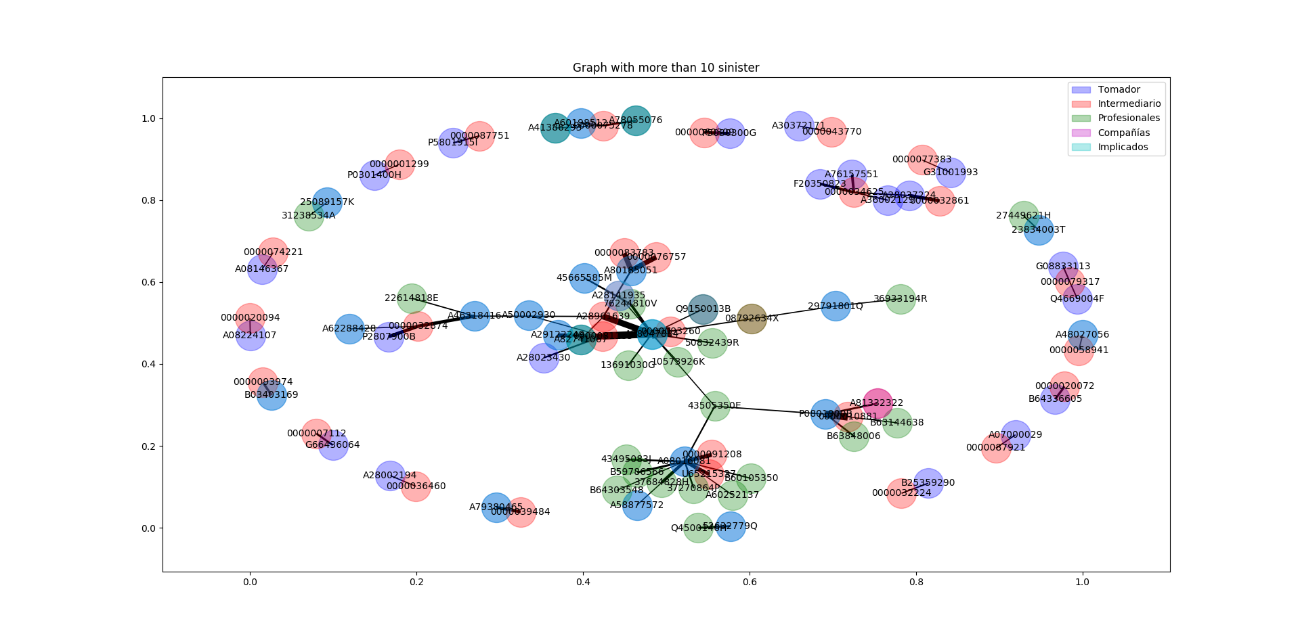
6) DUMMIES FILLNA: The automatic columns generated during the process. This is relevant to fill the nan values included in the new bottles. See section [APPEND AND MARK TEST](#_11._APPEND_AND).

7) EUROPA: Different types of strings associated to the EUROPA bottle.

8) A class of Parameters that are essential for the analysis. DO NOT FORGET TO CHANGE THEM IF THERE ARE NEW BOTTLES.

# FUTURE CHALLENGES

## GRAPH NETWORKS



## TEXT MINING

# ANNEX

## OTHER RESULTS

1. A range within one probable error on either side of the mean will include 50% of the data values. This is 0.6745\*sigma. Or what is the same, Q(0.75) = 0.6745 and Q(0.25) = 0.6745 [↑](#footnote-ref-1)
2. P. Geurts, D. Ernst., and L. Wehenkel, “Extremely randomized trees”, Machine Learning, 63(1), 3-42, 2006 [↑](#footnote-ref-2)