MASTER THESIS

Modelling for Science and Engineering

Data mining methodology: Application to an Alzheimer Database

Daniel Verdes Garcia

Setembre 2018

Universitat Autònoma de Barcelona

Abstract

Data mining methodology: Application to an Alzheimer Database

The aim within this work is to explore the different steps in a data mining process. We will take a methodological point of view, exploring the usual steps in a Knowledge Discovery in Databases process. Perhaps of that focus, the project also wants to give results in a concrete application of those techniques to a real problem. We are collaborating with the Terrassa Hospital physicology department, which give us the data and help us all the way in the interpretation of the results. The database concerns batteries of psychological tests each related with a psychological function. We try to find common patterns or structures inside the database that could give rise to new knowledge related with the diagnosis of Alzheimer in its earlier phases.

Acknowledgements

I would first like to thank my thesis advisor Josep Luís Arcos Rosell (IIIA-CSIC) for the opportunity to collaborate with him. He was always available when I needed help with the project and he assisted me all along the work in the IIIA-CSIC and in the visits to the Hospital of Terrassa. It was a pleasure to work with him from an academic, technical and personal point of view.

I would also like to thank the experts from the Learning Systems Department of the IIA-CSIC for all the help and the suggestions they gave me and the good time spent together.

I would also like to acknowledge Maite Garolera, Head of Neuropsychology Unit at Consorci Sanitari de Terrassa, and Neus Cano, MIR, for the good disposition all along the investigation and the help with the psychological aspects of the investigation.

Finally, I must express my very profound gratitude to my parents, sister, friends and specially to my partner Cecilia for all the support and continuous encouragement in the recent months. They were always a fount of motivation and inspiration for me to continue with the project.

Index

List of tables

List of figures

1. Introduction

The aim within this project is to explore the different steps of a data mining process. The document pretends to be as methodological as possible, as is reflected in the structure of the whole document. We follow the typical steps in a Knowledge Discovery in Databases (KDD) process. Besides of that focus, the project also presents results in a concrete application of those techniques to a real problem represented in a database.

This project has been developed in collaboration with the psychology department of Terrassa Hospital, which provided us the data and helped us all the way in the interpretation of the results. The database is composed by batteries of psychological tests each related with a psychological function. We will focus in the structure and meaning of the different parts of the database in the section, *"About the Database"*. Remark that all the data is completely anonymous since one of the very first steps in the process was to anonymize the patient ids. We will expand this explanation in the section "*Data acquisition"*.

We can divide the objectives of this project in three important and very differentiable parts:

- Explore the different steps in a KDD process.
- Explore different techniques for mining the data like linear models, machine learning algorithms, statistical techniques, topological data analysis…
- Get knowledge from the database and interpret it in association with psychology department of Terrassa Hospital.

A fundamental part in the project is related with coding the different scripts used for managing the data. We are going to use Python in all the work since is one of the most used languages in the data science world and has a lot of very well documented libraries for our purpose. Most of the libraries used are open source and the scripts will be included in the appendices so feel free to reuse them for your own!

We hope that this work could be useful as a practical reference for those interested in data mining, although we know that we can't be exhaustive in all the techniques and problems that can arise in the KDD process due to the huge variety of cases that can appear. We will focus specially in the data mining part since the work tries to be more related with the field of mathematics and algorithmics. Technical considerations like the different forms of storing the information will not be discussed in detail.

As we have already mentioned, the structure of the document is organized following the usual steps in a KDD process. This structure gives us the chance to explain in each part both the theory and general considerations about it and then to illustrate it the application to our particular case. We think that this form of dividing the chapters is much more didactic than presenting all the theory in a pretty big "preliminary concepts" section and then talking about each step in the analysis.

Due to that fact, we want to make a brief description of the KDD process. The general concept highlighting the main parts of the process, its importance nowadays, how we arrive here, and future steps. The objective is to motivate the reader to follow next chapters with a general vision in mind that will be helpful for understanding the whole process.

2. The Knowledge Discovery in Databases

First of all, we want to remark that we consider data mining as a part in the knowledge discovery in databases (KDD) process although both terms are often used as synonymous. Both expressions make allusion to the fact of extracting (mining) patterns (knowledge) from data. The difference that we make is related to the fact that KDD covers all the process that goes from the initial form of the data to the final representation of the acquired knowledge, whereas data mining is the part of the process where, having the data in an appropriate form, we start to search patterns in it using different techniques.

The phrase *"we are living in the information age"* is often pronounced nowadays in reference to the fact of living in an interconnected world where the information is much more accessible than years ago. We think that it was the paradigm some years ago but actually now the right sentence could be: *"we are living in the data age".* The point is that our capacity to generate and to store data has dramatically increased. Having data is different from having information, the data mining (or KDD) is precisely this process of extracting information (if possible) from data and from our point of view is one of the reasons of the significantly increase on the generated data nowadays.

In general, we can think in data like raw, unorganized facts with no context that doesn't contain knowledge (or very little) by themselves. In the other hand we can think in information like useful facts that putted in the right context could give us some knowledge. Precisely, this potential knowledge contained in the data is what has recently make those data mining techniques so popular and what makes that a lot of people in the whole world are monitoring data from everything you can imagine.

The paradigm has changed. First we think about what we want to know, then measure it and extract information. Now, we measure everything we can because we have a high chance to extract knowledge from it and almost every knowledge is valuable by somebody. But how does it work? We want to make a brief resume of the different stages of the process.

2.1. Data mining methodology

Here we want to highlight that all the steps are in mutual relationship, they are not just isolated parts of a sequential process. Contrary, a change or a result from one step can influence in the previous and following steps giving up to changes.

Figure 1 - Schema of the KDD process.

2.1.1. Data summarization

Before starting the proper KDD process, it is necessary to obtain some basic information about our data. Basic things like the number of features we have (dimensionality), the number of instances, the type of features (numeric continuous, categorical), the sparsity of the data, basic distribution of each feature data… So we will perform some basic operations to get this basic yet very useful information that will help us at the initial points of the process.

That part also helps us to confirm that we have more or less data with the expected characteristics.

2.1.2. Data preprocessing

Is the process where we put data in the correct form we have thought we need for mining. In average this process takes 60% of the time dedicated to the whole KDD process, so is no a trivial step. We can decompose this step in the following ones:

- Data cleaning: basically filling missing values, identify and remove outliers and resolving inconsistencies.

- Data integration: sometimes there are multiple data sources and in this step one attempts to combine them in a convenient way.
- Data selection: where we take the data that we consider relevant to our study.
- Data reduction: sometimes we try to reduce the amount of information we have due to different facts. In that sense we can attempt to reduce dimensionality (the number of features) or numerosity (the number of different instances).
- Data transformation: here we use different techniques to transform data into better forms for the analysis. Here we cover things like construction of new attributes from the original ones, smoothing features with noise, normalization or different ways of discretization.

2.1.3. Data mining

This is the essential part of the process where we attempt to extract information from the data by using different techniques that came from the fields of statistics and machine learning. Each case can be really different but in general we attempt to:

- Explain something: Finding usual patterns, correlations or an inside structure in the data that could help us to make an interpretation about the subject of study.
- Predict something: Use some of the features we have to predict the expected results of one or more features.

As you can imagine the variety of methods and techniques that are involved in that part is huge and is a field in continuous expansion. From the simplest techniques like linear models or decision trees to the more complex ones like deep learning all are susceptible of being applied. Depending in the case of study we will choose ones and discard others.

2.1.4. Pattern evaluation

Here we identify the patterns found that are really relevant in a statistical sense. In general, we never are hundred per cent sure with a prediction made, a correlation found or a structure showed by the data so we need to apply techniques (specially from the field of statistics) to evaluate the goodness of our findings.

2.1.5. Knowledge presentation

Although it can seem trivial the potential knowledge we can find in the database has to be finally presented in the form of graphics, conclusions, schemes, new data… This part is essential because usually the data scientist is in collaboration with people not related with the field of mathematics so is very important to take the others place and think in ways for easy transmitting the founded knowledge.

3. About the database

The database we are going to employ in order to exemplify the different steps in the KDD process come from the psychology department of Terrassa Hospital. We are in collaboration with them in the search of possible indicators of the development of Alzheimer disease in the earlier stages. In this part of the work we want to present the original form of the database that will be modified across the **data preprocessing** part. This will allow the reader to easier understand how and why we make the different changes in the structure of the database.

The data is stored in series of different excel files each related with a psychological field. At the start of the project we had twelve files and the data stored in a way that the information from a patient was spread between the different files. The names from the different ¿areas? is listed below:

- Atenció & WM
- Bateries Generals
- Bateries Inteligencia
- C_Emocional
- Escales Clniques
- Escales Funcionals
- Executives
- Llenguatge
- Memória
- Screening Cognitiu

- Velocitat
- Visuoespacial-perceptiu&Pràxies

Each area has eight columns or features in common (see figure below) and the rest of the features correspond to the scores in the different tests passed by the patients. As we have mentioned some areas could have some features in common because the scores of this tests are useful to different areas in the diagnosis.

Table 1. Example of the initial form of the excel files for the common columns.

	Sexe	Data Naixament	Diagnòstic	OmissionsPD	OmissionsPT	Porcentaje OmissionsPD	Porcentaje OmissionsPT	ComissionsPD	ComissionsPT
2011-12-20-1328804202352320000	0.0	1953.0	L-PNE	NaN	NaN	NaN	NaN	NaN	NaN
2009-10-302753309390922500096	1.0	1959.0	500421	NaN	NaN	NaN	NaN	NaN	NaN
2015-11-17-5615828347483809792	0.0	1955.0	500970	NaN	NaN	NaN	NaN	NaN	NaN
2012-12-20-6570801355670939648	0.0	1950.0	1601DCO	NaN	NaN	NaN	NaN	NaN	NaN
2015-05-13-2902006548032869888	1.0	1937.0	500667	NaN	NaN	NaN	NaN	NaN	NaN

The eight common features we can see in the table will be reduced to only three among the preprocessing step. As the names are in Catalan language we will explain each one.

First of all, we have the **id**, the id comes from the history number each patient has. This is a number that identifies each person who has access to the health system in Spain in an unambiguous form, in other words each patient in Spain has a number that corresponds only to himself. As this is confidential information one of the earlier steps of the project was to anonymize it obtaining a new number that we use as an id in our database, we will see it later in the data acquisition chapter.

Then we have the feature **Sexe** that correspond to the gender of each patient. **Data Naixement** corresponding to the date of birth. **Diagnòstic** corresponding to the diagnostic the doctor established before derivering the patient to the psychology area. **Desc. Diagnòstic** is a descriptor of the diagnostic with more info than the previous one compound by an alphanumerical code. **Data creació** is the date when the row was created in the excel, so the date when the patient passed the tests. **Codi Diagnòstic Episodi** is the same as Diagnòstic but for this episode in concrete. And the same for **Diagnòstic Episodi** where we have more info of this episode.

The rest of features have three hierarchical levels. We have the global name of the test that is composed by a several items and each item could have several scales of punctuation. They are:

- *PD punctuation*: correspond to direct punctuation. Is measured in the original scale of the test.
- *PT punctuation*: correspond to a normalized punctuation made employing information like the sex and age of the patient.
- *PE punctuation*: correspond to scalar punctuation.
- *PC punctuation*: correspond to the percentile punctuation.

We are not going to enter in detail of how these scores are computed. The important thing is to remark that are different scales and we need to add a third hierarchical level to this features because sometimes we are going to employ only one type of scale or another.

Table 2. Example of the final form of the excel files.

	Sexe	Data Naixament	Diagnòstic	CPT (Continuos Performance Test)					
	Sexe	Data Naixament	Diagnòstic	OmissionsPD	OmissionsPT	Porcentaje OmissionsPD	Porcentaje OmissionsPT	ComissionsPD	ComissionsPT
	Sexe	Data Naixament	Diagnòstic	PD	PT	PD	PT	PD	PT
2011-12-20-1328804202352320000	0.0	1953.0	L-PNE	NaN	NaN	NaN	NaN	NaN	NaN
2009-10-302753309390922500096	1.0	1959.0	500421	NaN	NaN	NaN	NaN	NaN	NaN
2015-11-17-5615828347483809792	0.0	1955.0	500970	NaN	NaN	NaN	NaN	NaN	NaN
2012-12-20-6570801355670939648	0.0	1950.0	1601DCO	NaN	NaN	NaN	NaN	NaN	NaN
2015-05-13-2902006548032869888	1.0	1937.0	500667	NaN	NaN	NaN	NaN	NaN	NaN

To keep it in mind we have some tests that are common to each area and have no hierarchy and other proves that do not need to be common between areas that have a three levels of hierarchy. Also we have info from twelve areas. So we can organize our information in a hierarchical way following the structure resumed in the following schema:

Figure 2. Schema of three level column structure.

4. Data acquisition

In this chapter we are going to talk about the process of acquiring the data. Depending on the case of study or the project we are related to, the data can come from the most diverse sources. Some examples could be: date from clients of some business, census, monetary transactions, relations between users of a social network, commercial flight routes, images taken from the Internet, images from security cameras, different types of audio files like records of calls from users to the customer service or songs published by some music label… everything is susceptible of being analyzed.

The data to be analyzed could be created automatically by some device like the different sensors employed in an autopilot system, the images from a web cam or the posts in an on-line forum or a social network. In this case the data could be analyzed in real time using previous information and giving immediate feedback to a user. Also the data is going to increase with time and you will need to handle this fact. In the other hand the data could be information recorded during a finite amount of time and stored in a static database. This is our case with the hospital's database where we have data stored since the year 2008 to the day we performed the extraction. In general, is common to have a mixture of both cases.

So we have a static database. As the data comes from a hospital the amount of data is going to increase with the time as more patients are treated. In the future we can add extra information to our database in order to increase the accuracy of our analysis, but for this present work the amount of data is fixed.

As we have discussed, there is a huge variety of forms to generate and store the data and the data acquisition process will be different in each case. As we can't cover all the cases we are going to focus in our concrete case study and explain how was the process.

The fact of keeping the anonymity of the patients is going to mark all the data acquisition process. The data acquisition process employs a script made in *Python* using **pandas** library. The script implements two functions: one employed as a filter and the other one used to anonymize the history number of the patient. The point is that we were not allowed to employ information from some patients. To know from which ones we can take information we needed to take in consideration two columns: *Usuari creació* and *Metge Productor*. So the filter discarded those rows of the database which value in this columns was equal to some code from a list of forbidden ones. So for each excel file in the hospital

database, we read it into a **pandas.DataFrame()** object and we applied the filter function to these object several times in order to select only the correct patients.

After that, we took the column *Història* that corresponds to the history number and we inserted a copy of it in the first column position with the name *id*. Once we made that, we overrided the info contained in that column with the new id generated by us. Then we generated a *csv* with the correspondence within *id* and *Història*. This csv was stored in the hospital for internal interpretation of the possible results from the study. To finish the extraction process, we deleted all the columns that were not valid for the study and that have sensible information, that was indicated by the doctors. We also transformed the birthday to the year of birth. Finally, we stored the information into a series of new excels with the same initial name. These excels will be the ones analyzed in this study.

One important thing was that we need to identify each row in the database in an unambiguous form. In the hospital's database this was made using the history number of the patient and the date when the patient passed the psychological tests. There was no problem with keeping the date but we needed to convert the history number into a new id to maintain the patient's anonymity. To make that we employ a hash function. A hash function works in the following way:

"A hash function is a function that takes input of a variable length sequence of bytes and converts it to a fixed length sequence. It is a one-way function. This means if f *is the hashing function, calculating* f(x) *is pretty fast and simple, but trying to obtain* x *again will take years. The value returned by a hash function is often called a hash, message digest, hash value, or checksum. Most of the time a hash function will produce unique output for a given input. However, depending on the algorithm, there is a possibility to find a collision due to the mathematical theory behind these functions." (Torres, 2013)*

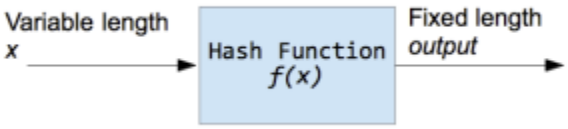


Figure 3. Schema hash function flow. Reprinted from Hashing Strings with Python in *Python Central*, by A. Torres from https://www.pythoncentral.io/hashing-strings-with-python/

So this kind of function allows us to map each history number to an id in an unambiguous form making the reverse mapping very hard. As hash function we choose the built in **hash()** function from *Python*. This function returns for each Python object a hash. It works in a different way for each type of

object as each object has implemented a _hash_() method. In the case of an integer it returns the same integer, but in the case of a string it uses a complex algorithm.

The point is that despite of not being very secure is very fast and accomplished the rules imposed to a cryptographic hash function as:

- it is deterministic so the same message always results in the same hash
- it is quick to compute the hash value for any given message
- it is infeasible to generate a message from its hash value except by trying all possible messages
- a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
- it is infeasible to find two different messages with the same hash value

Also in each execution of the program the returned value from a string will change because the algorithm uses some info from the performance of the computer.

Due to all these facts we thought that this hash function was a good option for us and was not necessary to employ more complex ones. In this way the function implemented by us takes the history number and converts it into a string and then applies this hash function returning the hash value.

The code implemented by themselves is in the annex part so we encourage you to take a look at it after reading this part in order to easy understand the process.

5. Summarization

Before starting the preprocessing part is always important to better know our data. A general mark of how is our data will guide us in choosing which algorithms or techniques we can employ to analyze our data. Things like knowing the amount of missing values, know the type of data, the ranges, number of categories are some of the initial questions we have try to answer. This could be done by taking a look to the database. After that we will want to know more precise information about each feature.

As the amount of data has to be very big we will need to employ statistical tools in order to make a useful description. In general, obtaining some basic statistical measures like the mean, the median or the mode can be very useful to have a basic idea of the behavior of our features. This can be also very helpful in order to detect noisy data or outliers and to deal with missing values. This are measures of central

tendency but also is important to know how disperse is our data. There are several techniques employed for that purpose. We are going to talk about two techniques: First one focused in the **variance** and **standard deviation** and the second one focused in the **interquartile range**.

5.1. Variance and standard deviation

These two statistical measures, which are very related between them, tell us how much our data spreads from the mean. Is important to remark that due to the mathematical formulation of this measures they are not good for asymmetric distributions in the sense that they don't give us information about that asymmetry.

$$\textbf{Variance: } \sigma^2 = \frac{\sum(x-\mu)^2}{N}$$

$$\textbf{Standard deviation: } \sigma = \left[\frac{\sum(x-\mu)^2}{N}\right]^{1/2}$$

Where μ is the median and N the total number of samples. Remark that as we have squared the summands in the summation the units of the variance will be quadratic this yield to a more difficult interpretation. Due to that fact the standard deviation is defined. Is easy to realize that a distribution where all the values are the same will have variance and standard deviation 0. Is also easy to realize that due to its definition the asymmetry of the distribution is not reflected. Due to these facts is the most employed measure of spread of an approximately normal distribution.

5.2. Interquartile range, five number summary and box-plots

Here we outline another approach to understand the distribution of our data. The good thing about it is that is sensible to asymmetric distributions in the sense that will give use information about it and will consider this asymmetry when detecting outliers. Here we are going to talk about *range*, *quantiles*, *quartiles*, *percentiles* and the *interquartile range*. If we have a set of measures x_1, x_2, …, x_N related to a numeric attribute X the **range** of the set is the difference between the largest and smallest values.

If we sort the data in increasing numeric order and we make bins of equal number of points size, the points that correspond to the closure of this bins are called quantiles. Is not always possible to make this split with perfect equal size so we can refine our definition. In words of Han, Pei & Kamber (2011, p.48):

"The kth q-quantile for a given data distribution is the value x such that at most k/q of the data values are less than x and at most (q − k)/q of the data values are more than x, where k is an integer such that 0 < k < q. There are q − 1 number of q-quantiles. The 2-quantile is the data point dividing the lower and upper halves of the data distribution. It corresponds to the median. The 4-quantiles are the three data points that split the data distribution into four equal parts; each part represents one-fourth of the data distribution. They are more commonly referred to as quartiles. The 100-quantiles are more commonly referred to as percentiles; they divide the data distribution into 100 equal-sized consecutive sets. The median, quartiles, and percentiles are the most widely used forms of quantiles."

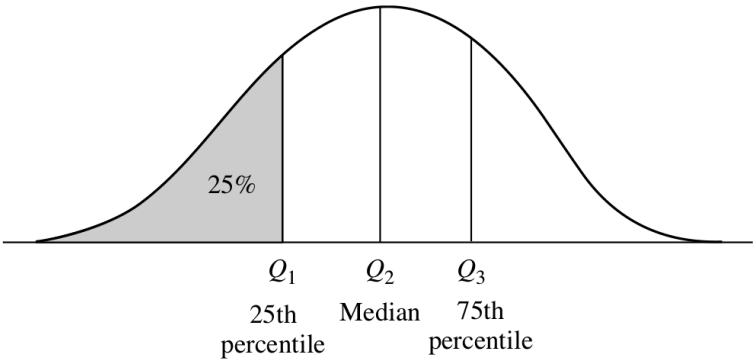


Figure 4. Example of first, second and third quartiles. Reprinted from Data mining: concepts and techniques (p.48) by J. Han, J. Pei & M. Kamber, 2011, Elsevier.

From that we can also define the **interquartile range** (**IQR**) as the distance between the third and first quartiles:

$$IQR = Q_3 - Q_1$$

This give us information about the range covered by the middle half of the data. In the analysis made by use we will use the quartiles in order to have a more precise knowledge about our data distributions and to detect outliers.

A common approach to identify outliers based in this concepts is to set as outliers the points falling at least *1.5 x IQR* above the third quartile or below the first quartile. A useful resume of the spread of the data distribution is made by employing the so called **five number summary** where we write in increasing order the **Minimum**, **Q1** , **Median**, **Q3** , **Maximum**. That kind of summary is even more useful when is expressed in a chart. This kind of graphical representation is call **box-plot** and incorporates the five number summary as follows:

- The edges of the box are the first and third quartiles so the length of the box is the interquartile range.
- The second quartile, the median is marked by a line within the box.
- Two lines called whiskers outside the box extend to the minimum and maximum of the distribution when there are not outliers. If outliers are present, the whiskers represent the edges of the area above and below the *1.5 x IQR* and we also represent the outliers.

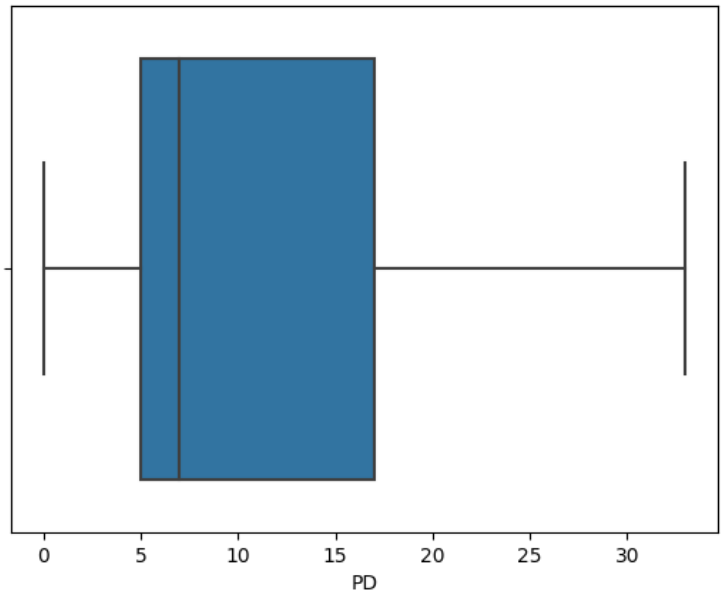


Figure 5. Example of a box plot. Area memoria. Target: lista de palabras del RBANS verbal, Corba aprenentatgePD.

We made use of this kind of plots along the project to better know the distribution of the data and in order to detect outliers. As we will mention in further chapters we realized that in our concrete application outliers in some features are useful because the diagnosis is made from the anomalies in the tests and these anomalies are characterized by this kind of extreme values.

5.3. Missing values

When dealing with databases problems derived from inconsistent or incomplete data tend to appear. As we are analyzing real world data we always need to have this issue into account. Sometimes the information recorded in the database is not complete but yet useful, sometimes there are mistakes when

introducing the data in the database, some sensor of some machine can fail, some unexpected events can change the usual behavior... To solve this kind of problems is one of the very first steps in the data preprocessing step. To achieve that some extra information is always useful.

We can define the **consistency** of a data base as not to violate the internal own rules of the database. Consistency for example could be to introduce a value in a categorical variable that is not expected, introduce something in a format different from the one specified and in the case of relational databases not to have a unique key identifying each row. Ideally databases should have mechanisms to avoid this kind of inconsistencies but sometimes this is not well made or even there isn't any kind of precaution.

The issue of **incomplete** data is even more common. This could be for several reasons like mistakes or an incapability to know the value of a feature in a concrete case or just giving more importance to concrete instances than others. Although incomplete the data is still useful so we need to define some strategies to deal with that. The problem is that most of the machine learning algorithms and mathematical models can't deal with missing values. Sometimes this missing values could reveal information, for example could be an evidence of apathy or displeasure in a survey.

Here we explore some ways to deal with the missing value patterns and distributions. We will basically talk about two usual strategies: the numerical and graphical ones. Both of them are based on the proportion of missing values over the total number of values.

The simplest one is to make some tables putting the percentage of missing values over the total number of registers or just the total number of missing values. Something like the following tables made for our concrete case of analysis:

Table 3. Summary of the number of non-missing values for each column for the atencion area. Output of the function pandas.DataFrame.info()

MultiIndex: 1391 entries, (-7935167588970630144, 2015-07-24) to (-402936765635360320, 2012-05-31)
Data columns (total 36 columns):
(Sexe, Sexe, Sexe) 1391 non-null int64
(Data Naixament, Data Naixament, Data Naixament) 1391 non-null int64
(Diagnòstic, Diagnòstic, Diagnòstic) 1374 non-null object
(CPT (Continuos Performance Test), OmissionsPD, PD) 164 non-null float64
(CPT (Continuos Performance Test), OmissionsPT, PT) 3 non-null float64
(CPT (Continuos Performance Test), Porcentaje OmissionsPD, PD) 137 non-null float64
(CPT (Continuos Performance Test), Porcentaje OmissionsPT, PT) 49 non-null float64
(CPT (Continuos Performance Test), ComissionsPD, PD) 164 non-null float64
(CPT (Continuos Performance Test), ComissionsPT, PT) 161 non-null float64

Table 4. An example of the tables contained in the summary file we use to communicate the amount the data we have to the hospital.

Function	Test	Percentaje of test with data
atencion	Numero total de pacientes	1044
	Dígitos: directos (WAIS)	74,04
	Trail Making Test: TMT-A	43,10
	CPT (Continuos Performance Test)	36,78

The graphical approach is sometimes more intuitive especially when the number of features is large. Several things can be made like heat maps or histograms reflecting the number of values or missing values of each feature or the percentages. Here we present some graphical visualizations in our concrete case.

This kind of plots are really helpful to select or discard features based in the number of missing values and also to find patterns in the missing values form that could give us to better understand why are this missing values here.

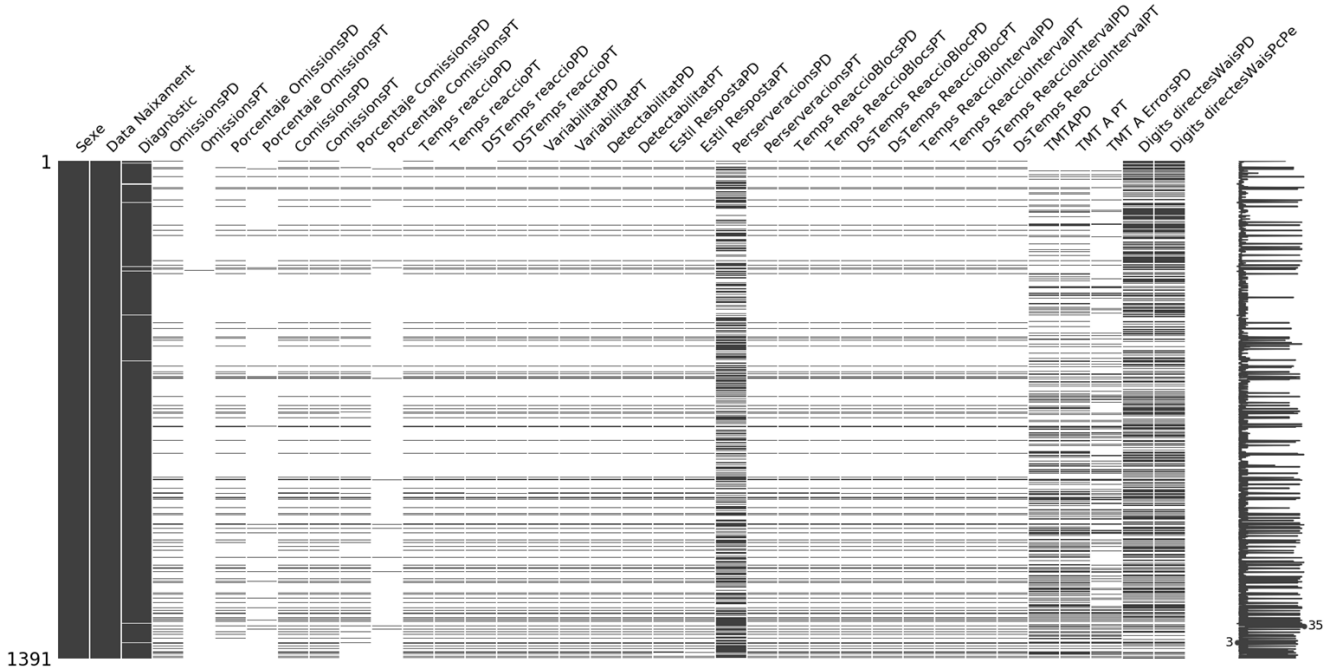


Figure 6. Example of graphical representation to visualize the amount of missing values. Not null values in black, missing values in white. Made employing missingno library

As we have say that the parts from the KDD process are not isolated, explain after this theoretical introduction how we developed this step for our concrete case of study will be really hard and yet difficult to understand. To solve that we will put in a common chapter both the Summarization of data and the Preprocessing steps in application to our case of analysis. We think that in this way it will be easier to understand.

6. Data preprocessing

This is one of the essential parts of the process. In fact, it usually represents between sixty and seventy percent of the time spent in whole the process. Without this part the results from the data mining part would be much less accurate or simply we couldn't have made the mining process. Here we transform the data into new forms more convenient for the analysis. Things like integrating the whole data in the same database, scaling the data, making discretization, filling missing values or reducing the number of instances or features are the typical tasks achieved.

We can divide the tasks related with this part in five different categories: data selection, data integration, data cleaning, data reduction and data transformation. Now we make a brief description of each one:

6.1. Data selection

In this part we select the subset of data we are going to employ in our study. Depending on the case of study there will be features from the database that could be interesting and others than not. Being able to select the correct ones is really important and sometimes require several attempts, experience and probably external help from somebody related with the field of study. There are no general criteria in making this process.

In our concrete case the data selection was made thanks to the help from the doctors in the moment of the data acquisition at the hospital.

6.2. Data integration

When data comes from different sources is necessary to integrate all of them into a unique coherent and ordered database. In our case although all the data comes from the hospital, the data is stored in different excel files each related with an area. After some analysis and due to the amount of missing values we finally integrate all the data into a unique database.

6.3. Data cleaning

This part is essentially related with the missing values, outliers and inconsistencies. Is the most important part of this step since is the one that formats the database into a form that can be properly employed by machine learning algorithms. The other parts can make the mining process easier or improve the results but without this part most of the algorithms couldn't work due to the presence of missing values. Here we will employ most of the knowledge from the *Summarization* step.

We have briefly talked about strategies for dealing with outliers in the *Summarization* chapter. These strategies were based in the use of quantiles. Other popular techniques employed are those related with clustering algorithms. If we can classify the data in similar groups those values that fall apart will be identified as outliers. The problem with that is that usually you need to know the number of clusters. In general, this kind of algorithms are related with a similarity distance measure that usually is hard to define since we don't know the characteristics of the mathematical space related to the data. In our case we

didn’t employ that kind of outlier detection because we know the range of the values of each probe. We simply remove the values that are out of the scale.

Once the outliers are removed we have to think in strategies for dealing with missing values. Several things can be made. The most usual ones are:

- Ignore the rows from the data base with some missing value. The disadvantage with that is clear, the loss of potentially useful data.
- Try to fill the missing values with new measures. The problem with that is that in general will take a lot of time and when the database is not created by ourselves we will need to ask another people for help.
- Use some measure of central tendency. The mean, the median or the most frequent value. Which one to choose will depend in the algorithm employed and the form of the data distribution. The better approach is to try all of them and choose the one that give us more accuracy in our predictions.
- Use some measure of central tendency by class. If we are in a supervised machine learning problem this can improve the results. Again the better choice will be the one that provides more accurate results.
- Try to infer this missing values from the data we have using some model or predictive algorithm.

Excluding first and second methods these approaches bias the data because the filled value may not be correct. The last approach is one of the most popular but in our case, as we are searching for patterns between the features, it has no sense to be used as a filling missing values strategy. In further steps of the investigation maybe we could employ it.

6.4. Data transformation

This part involves transforming data into forms more optimal for the analysis. We include here data discretization and feature scaling. We make a brief presentation of this techniques below:

6.4.1. Data discretization

Is related with methods to convert numerical continuous features into categorical ones. This process has several benefits. You can try more algorithms since there are algorithms that only work with categories. Also you reduce possible noise in the data because of grouping it in ranges. There exists a lot of techniques for this purpose like: Discretization by binning, by clustering or by decision tree.

In our case the discretization process will be based in categories from the hospital. For each type of punctuation, they have several categories. For example, in the case of *Pc* punctuation:

0-10 Deficitario/ 10-30 Bajo/ 30-80 Medio/ 80-95 Máximo

We will assign a number for each category following this rules specified from the hospital. In the case of the *PD* punctuation as it varies with each test and is a relative scale they don't transform the data into categories. In order to discretize it we will employ *discretization by binning*. We choose a number of categories equal to the *PT* punctuation, this is 5 different categories. We will split the data range in 5 sets of equal length and assign each value to one of them.

6.4.2. Feature scaling

Usually each feature in our database will be in a different scale. This yield to several problems with the algorithms. In general features with larger values will domain in the algorithm due to the fact of having bigger variance as the range of values is bigger. Also in algorithms based on distance if we have different scales they will perform poorly. It is also important in models that introduce a penalty to the number of variables as Lasso or Ridge linear regressions since we need to try with different values for the penalty coefficient, λ and the value of this coefficient will depend on the range of the features. There are two main methods for that propose:

- **Normalization:** This method performs a linear transformation on the data points to the range [0, 1] employing the maximum and minimum points. It follows this formula:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

  The good point of that method is that it preserves the relationships among the different values. In the other hand the bad point is that if we don't previously remove the outliers this

transformation could transform the important data points into a much smaller range than the [0, 1] expected.

- **Standardization:** This method also performs a linear transformation on the data points but in order to center them around zero and make unit-variance distribution of data. The following formula is employed:

$$x_{new} = \frac{x - \mu}{\sigma}$$

This method solves the problem described before about the outliers. In fact, what it does is to convert the data into z-score units, standard deviations from the mean value.

In the next chapter we will exemplify how we applied those techniques to our concrete case of analysis. We will do it in combination with a summary that will help us among the process. We will also include some schemes showing the data flow and how we handle all the process relating it with the different steps of KDD. Before starting this chapter, we want to talk about something crucial in data analysis and the amount of data we need to obtain good predictions. We include it here because we want to keep in mind that the amount of instances we have is very important.

6.5. Curse of dimensionality

We want to introduce two important concepts in data mining: numerosity and dimensionality. We can define numerosity as the amount of data we have, for example the number of rows in a database. Dimensionality is known as the number of variables or features present, for example the number of columns in a database.

From a learning perspective our models always try to make generalization from data. This is to extract a general rule from concrete examples. From this point of view, it is easy to figure out that as more examples we have better will be the generalization made. In the other hand is also easy to understand that in general as more features are relevant to the model more complex will be and more difficult will be to extract a general rule. In general, the point is that the numerosity needed increases exponentially with the dimensionality or in other words the number of examples required to achieve the same accuracy with the same model grows in an exponential way as the number of features increases. So in general we need to keep in mind that as more features are involved more instances we will need.

This justify all the discussion about missing values. If for example, we have a feature with a twenty percent of missing values usually will be better to simply avoid this column in the model than to avoid the eighty percent of the instances with null value in that feature.

As our case of analysis is very sparse as we will see in the following chapter having good strategies to deal with that will be crucial to extract conclusions from our analysis.

7. Application to our case of analysis I

As we have mentioned in previous chapters it is difficult to talk in an isolated way about the different steps in the KDD process. Due to that fact we introduce this chapter that in addiction with *Application to our case of analysis II* will cover the study made by us about the Alzheimer disease database. Our work was made in a series of python scripts organized in several *Jupyter notebooks* included in the annexes part. In order to be more didactic we present a schema were we organize all the tasks made till finish the preprocessing part.

The input of the work flow is the initial excel files extracted from the hospital. From each Jupyter notebook we generated a new set of excel files that were the output for the next step. From some of this notebooks we also incorporated graphics and tables. At the end of the data preprocessing part we generated two different databases one with numerical continuous data and the other with categorical data. Apart from this work flow we introduced some scripts that form part of the preprocessing part used to transform the databases in terms of the strategies employed for filling missing values and the feature scaling performed. In the data mining part, we will systematically employ a set of different machine learning algorithms to each one of the files and we will make a comparison between them using different accuracy measures.

The **Figure 7** represents the complete work flow till starting the data mining part. The squares represent the set of excel files that constitute the database, the circles are the different Jupyter notebooks and the hexagons are outputs related with statistical analysis like tables and graphics. The notebooks related with the *Summarization* are colored in light gray and the ones related with the *Preprocessing* in light green.

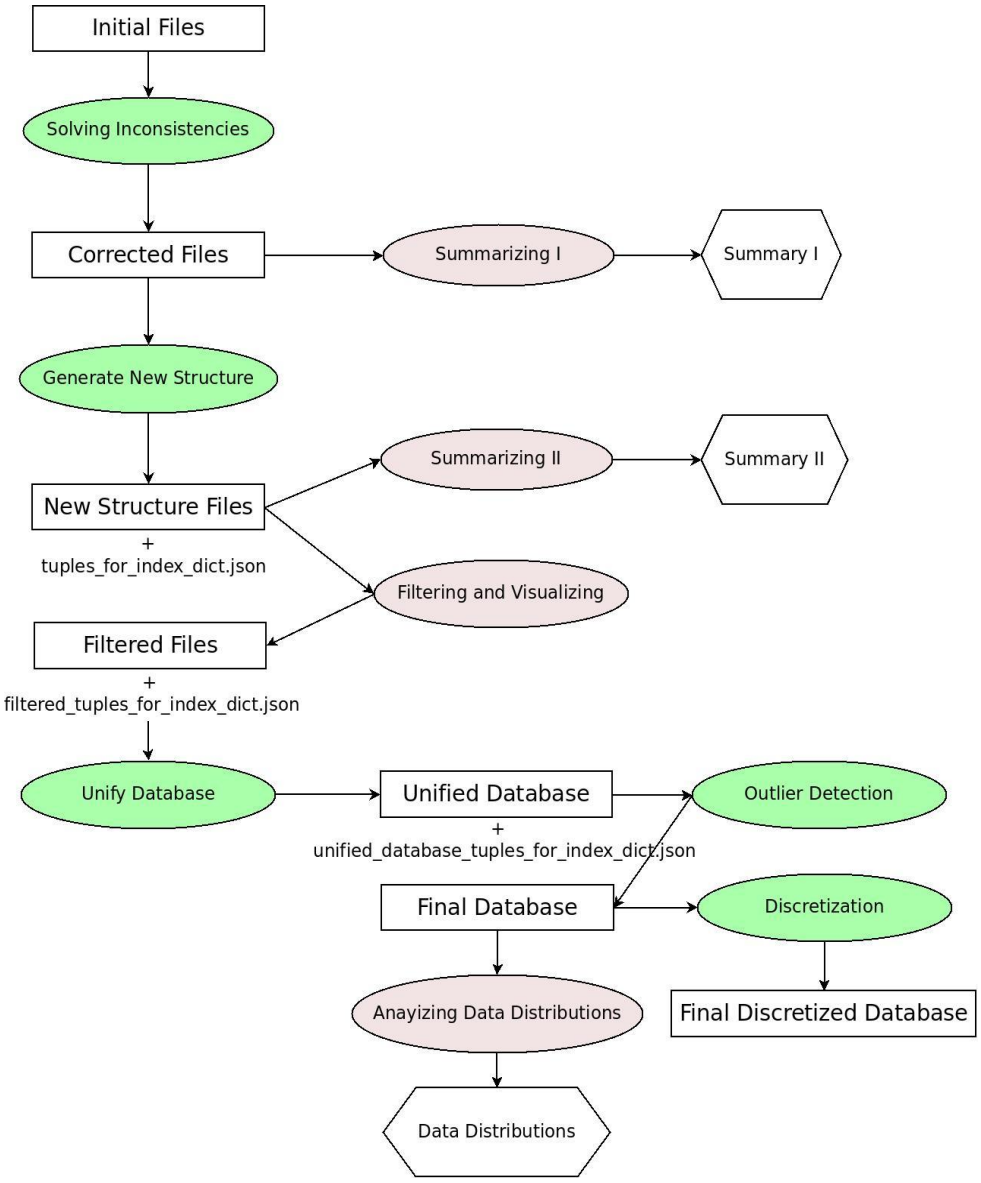


Figure 7. Schema of the whole process from the Initial Files to the Final database. Circles represent Jupyter notebooks, Squares represents sets of files or databases and the hexagons the main summaries made among the process. In light grey the scripts that fall into the Summarization category and in light green the ones that fall into the preprocessing category.

Following this schema, we will explain the different tasks performed by ourselves till the end of the data preprocessing part. Each Jupyter notebook is included in the annexes part so we encourage the reader to take a look on them in order to deeply understand the process. As the amount of tables and graphics generated from the Jupyter notebooks is big we don't include all of them in the schema. There are only

included the most important ones specially due to the fact of being the ones used to communicate the state of the project with the hospital. The rest of outputs from the scripts will be presented here as we explain the process.

7.1. Solving Inconsistencies

We have talked about **consistency** and **incomplete** data. One of the very first task performed in this project was to solve this kind of problems. In the case of consistency, we can search for lot of things and it will depend of the problem. Is essential to use all the information you have from the database, but there is not a general rule. In our case when making the initial statistics we realized that the number of unique keys of the database was lower than the number of rows which yields to an inconsistence since we could not identify in an unambiguous way each patient. We thought that there were duplicated rows but the problem was more complex. The information was split in several rows so we merge it into a unique one. Another inconsistency found was that there were features common between areas and for one patient in some areas this feature was filled with information and in other cases not.

To solve this problems, we implemented a function called *merge_repited_rows_into_one()* that take as input a *pandas.DataFrame()* object and returns the same object after perform some transformations. We merged the double index we have into a combination of both ids and set this as the new index. After that we made a set with this combined ids avoiding possible repetitions. Iterating the database index, we search for those ones with multiple rows associated to the same key and we merged them into one. We drop the old rows and we append the generated one. After that we reset the original double keyed index.

We performed this process for all the initial excel files loading them as a pandas.DataFame() with the use of *pandas.read_excel()* function. We also drop possible duplicated rows, duplicated in the sense that contains exactly the same information. We store the formatted data frames in a dictionary using the excel file name as key.

After that we drop the column *I+5 Temps latenciaPD* from the excel file *Executives.xlsx* because we have some information from the hospital that said that the information contained were erroneous. Finally, we check for the folder *Corrected Files* and create it if not present. Then we store the corrected data frames in this folder as excel files.

We also made some checking to know if the process worked as expected. Here is part of the output were you can see how we merged the info spread in three columns into one:

Table 5. Example of the merging process in the solving inconsistencies Jupyter notebook I.

Digits directesWaisPcPe	Digits inversosWaisPD	Digits inversosWaisPcPe	OmissionsPD	Porcentaje OmissionsPD	Porcentaje OmissionsPT	ComissionsPD	ComissionsPT	Porcentaje ComissionsPD	Porcentaje ComissionsPT
81.69	7.0	80.83	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	3.0	9.0	0.0	6.0	NaN	16.7	NaN
NaN	NaN	NaN	3.0	9.0	0.0	6.0	41.0	16.7	0.0

Table 6. Example of the merging process in the solving inconsistencies Jupyter notebook II.

Digits directesWaisPcPe	Digits inversosWaisPD	Digits inversosWaisPcPe	OmissionsPD	Porcentaje OmissionsPD	Porcentaje OmissionsPT	ComissionsPD	ComissionsPT	Porcentaje ComissionsPD	Porcentaje ComissionsPT
81.69	7.0	80.83	3.0	9.0	0.0	6.0	41.0	16.7	0.0

7.2. Summarizing I

After solving this kind of consistency problems we want to generate some tables with information about the amount of data. This was important for the hospital in order to know the information they have. These initial tables contain the total number of patients treated and the number of patients by area. Also they contain the larger percentages of patients that had passed the tests. In addition, some tables with the number of times that patients pass the battery of tests by area. A probe not passed by a patient is considered as a missing value because is not introduced in the excels. We will see that this missing values were one of the principal difficulties in the project.

We don't want to explain in detail how we program the scripts for this part because they are more related with technical details of how to use the pandas.ExcelWriter(). It should be pointed out that the main process includes the use of the method *isnull()* from pandas.DataFrame() that generates a new data frame with boolean values indicating whether the value of the cell is or not null. With that we iterate each

column counting the number of cells with a *False* value which indicates that the cell doesn't contain a null value. The resulting tables are contained in the annexes part.

One important thing to mention was that as we have explained in the chapter *About the Database* the database was a three level structure of columns. First we have the test group, then the item itself and then the type of punctuation. At the start we only considered a one level columns structure and we thought that each prove from each test starts with the test name followed by some extra info. Due to that fact we iterated the columns searching for a change in the first three letter of the name in order to present a summary about each test and not each prove. When we send this initial summary to the hospital the doctors tell us that the process had not worked as expected.

The supposed structure of the columns doesn't exist and each column had a name that only a psychologist related with this batteries of test could understand. In order to solve that we ask them to elaborate a document containing the names of the tests related with each area and the column names related with the proves of each test. That yields to next section of the chapter.

7.3. Generate New Structure

Once we have the document made by the doctors explaining the required information we make some scripts to generate a new structure of columns. This was a hard part of the whole process due to the amount of tests made to the algorithm in order to see if was working properly and also for the amount of facts to be considered. Another thing that took a lot of time was transcribe the document created by the doctors into a set of .json files. We make that because in that way it was very easy to load this files as Python dictionaries and employ them in the process. To illustrate the form of this dictionaries we present the following image:

```
{
"CPT (Continuos Performance Test)":
["OmissionsPD", "OmissionsPT", "Porcentaje OmissionsPD", "Porcentaje OmissionsPT",
"ComissionsPD", "ComissionsPT", "Porcentaje ComissionsPD", "Porcentaje ComissionsPT",
"Temps reaccioPD", "Temps reaccioPT", "DSTemps reaccioPD", "DSTemps reaccioPT", "VariabilitatPD",
"VariabilitatPT", "DetectabilitatPD", "DetectabilitatPT", "Estil RespostaPD", "Estil RespostaPT",
"PerserveracionsPD", "PerserveracionsPT", "Temps ReaccioBlocsPD", "Temps ReaccioBlocsPT",
"DsTemps ReaccioBlocPD", "DsTemps ReaccioBlocPT", "Temps ReaccioIntervalPD", "Temps ReaccioIntervalPT",
"DsTemps ReaccioIntervalPD", "DsTemps ReaccioIntervalPT"],

"D\u00edgitos: directos (WAIS)": ["Digits directesWaisPD", "Digits directesWaisPcPe"],

"Trail Making Test: TMT-A": ["TMTAPD", "TMT A PT", "TMT A ErrorsPD"]
}
```

Figure 8. Example of one of the json files containing the tests and proves dictionary. We can see the names of the tests as dictionary keys and the names of the columns as a list of strings.

In this Jupyter notebook we load the data from *Corrected Files* and the set of json files. When loading the excels we transform the variable *Sexe* into a binary numeric attribute where 0 represent Female and 1 represent Male. After that we generate a combined id as we have done in the previous Jupyter notebook. Then we iterate all the data frames storing all the combined ids. Then we iterated the list of dictionaries, for each one of them we generated an empty data frame which index is the mentioned set of all ids. Then we search in each excel file loaded for columns that match with the ones stored in the dictionary. We append the data from these columns to the empty data frame having in account the patient combined id. While making that we create a tuple containing three strings the name of the test, the name of the column and the type of punctuation. These tuples are important to generate the three level column structure mentioned. After we iterate all the data frames and all the keys of the dictionary we remove the columns from the new data frame that don't contain information.

The new data frame generated based in the info from the dictionary is stored in a new dictionary in combination with the list of tuples that will constitute the new column structure. After that we store the generated data frames into a set of new excel files. This set of excel files is called *Filtered Files* in the schema. We also store the set of tuples that will form the new column structure in a json file which keys are the names of the new generated data frames. Here we present an image comparing the original structure and the new one.

Table 7. Example of the initial structure of columns.

	Sexe	Data Naixament	Diagnòstic	OmissionsPD	OmissionsPT	Porcentaje OmissionsPD	Porcentaje OmissionsPT	ComissionsPD	ComissionsPT
2011-12-20-1328804202352320000	0.0	1953.0	L-PNE	NaN	NaN	NaN	NaN	NaN	NaN
2009-10-302753309390922500096	1.0	1959.0	500421	NaN	NaN	NaN	NaN	NaN	NaN
2015-11-17-5615828347483809792	0.0	1955.0	500970	NaN	NaN	NaN	NaN	NaN	NaN
2012-12-20-6570801355670939648	0.0	1950.0	1601DCO	NaN	NaN	NaN	NaN	NaN	NaN
2015-05-13-2902006548032869888	1.0	1937.0	500667	NaN	NaN	NaN	NaN	NaN	NaN

Table 8. Example of the final structure of columns.

	Sexe	Data Naixament	Diagnòstic	CPT (Continuos Performance Test)					
	Sexe	Data Naixament	Diagnòstic	OmissionsPD	OmissionsPT	Porcentaje OmissionsPD	Porcentaje OmissionsPT	ComissionsPD	ComissionsPT
	Sexe	Data Naixament	Diagnòstic	PD	PT	PD	PT	PD	PT
2011-12-20-1328804202352320000	0.0	1953.0	L-PNE	NaN	NaN	NaN	NaN	NaN	NaN
2009-10-302753309390922500096	1.0	1959.0	500421	NaN	NaN	NaN	NaN	NaN	NaN
2015-11-17-5615828347483809792	0.0	1955.0	500970	NaN	NaN	NaN	NaN	NaN	NaN
2012-12-20-6570801355670939648	0.0	1950.0	1601DCO	NaN	NaN	NaN	NaN	NaN	NaN
2015-05-13-2902006548032869888	1.0	1937.0	500667	NaN	NaN	NaN	NaN	NaN	NaN

With this new structure we will generate a new summary to give some feedback of the investigation to the hospital.

7.4. load_data

This script is made to load the data into the Jupyter notebooks in an easy way. As now we use a different structure of columns each time we load an excel file with the function *pandas.read_excel()* we need to format it. The script contains a function call *read_and_format()* that takes as arguments the folder where the excel files are stored and the name of the json file where the structure of columns is stored. We also incorporate an optional argument to set if we want to load the data with unique index composed by the date and id or a double index. The script is included in annexes and we encourage the reader to check it in order to better understand this explanation.

7.5. Summarizing II

This script is essentially equal to the Summarizing I previously presented. There are three main differences between them. The whole script is adapted to the new three level structure. The data is loaded employing a script made by us in order to facilitate load the excel files into data frames with the new format. This script is described in the previous section. The last difference is that we present all the percentages of the number of patients that had passed the tests not only the main ones. The summary

generated with the name *Summary II* can be checked in the annexes. We pass this summary to the hospital and the feedback says that now the structure was correctly organized and that the amount of data was the expected.

To complement this summary, we generate a series of graphical descriptions with the use of the *missingno* library. Here is an example of one of them the rest are included in annexes:

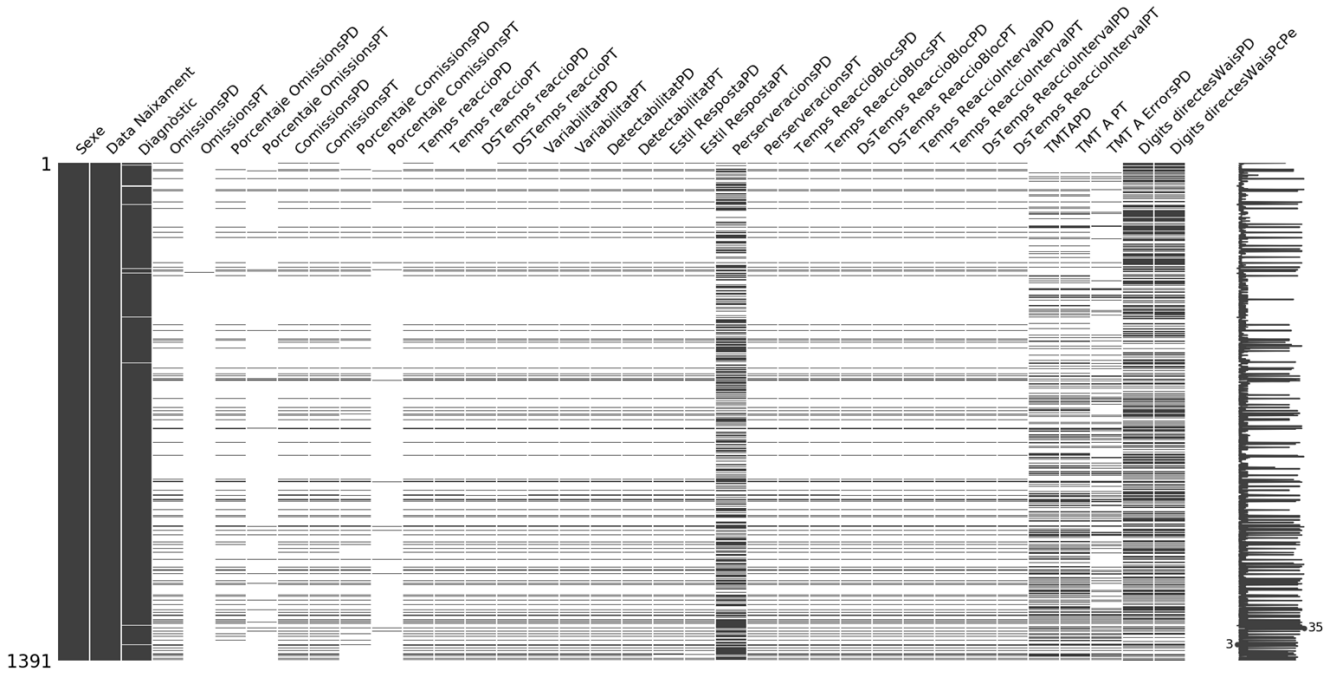


Figure 9. Example of a graphical description of the amount of missing values for the area atencion. We can see that this database is very sparse. Figure made employing missingno library.

As we can see the number of missing values is very big and we need to deal with that in the next section we will talk in great detail about it.

7.6. Filtering and Visualizing

Due to the sparsity of the data base we want to filter out these columns with less than X percentage. We also want to visualize the number of records that results from the filtering process in each file. Also filtering out those patients that could be considered outliers in the age feature is important. We follow the rule of considering an outlier those ages that are beyond 1.5 times the interquartile range. The visualization of the missing values is made in the same Jupyter notebook to know how the amount of

missing values evolve with the filtering process. So we can classify this notebook in the *Summarization* part or in the *Preprocessing* part. The visualization process is made using a graphical description similar to the presented in the last section and a representation based in tables with the amount of data by area and column.

We iterate each database loaded and check the number of values that contain each column. If the column doesn't fulfill a specified threshold we remove it. We also implement a function to visualize the result. The function takes a dictionary where all the data frames are stored and represent the amount of patients for each column and area in a column. The 0 column is the number of columns that remains. The 1, 2, and 3 columns correspond to the *Sexe*, *Data Naixement* and *Diagnòstic* that are almost always present. Now we present several tables with the results from filtering with several thresholds of thirty, fifty and sixty percent:

Table 9. Comparison of the number of patients that result from the filtration process by area and column. The First column is the number of remaining features. The first, second and three represent the Sexe, Data Naixament and Diagnostic. Table for 30% threshold.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
WM	5.0	772.0	772.0	762.0	770.0	771.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
atencion	8.0	1391.0	1391.0	1374.0	772.0	773.0	766.0	449.0	450.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
capacidad_visuoconstructiva	11.0	922.0	922.0	911.0	398.0	312.0	545.0	538.0	483.0	467.0	318.0	323.0	0.0	0.0	0.0	0.0	0.0
capacidad_visuoespacial	5.0	553.0	553.0	545.0	376.0	378.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
capacidad_visuoperceptiva	5.0	84.0	84.0	82.0	73.0	73.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
conducta_emocional	17.0	45.0	45.0	45.0	45.0	42.0	45.0	41.0	45.0	41.0	45.0	41.0	45.0	41.0	45.0	41.0	45.0
escalas_clinicas	4.0	153.0	153.0	151.0	153.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
funcion_ejecutiva	13.0	1266.0	1266.0	1247.0	451.0	458.0	583.0	582.0	582.0	579.0	586.0	412.0	410.0	404.0	0.0	0.0	0.0
lenguaje	7.0	886.0	886.0	871.0	394.0	397.0	387.0	388.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
memoria	35.0	1372.0	1372.0	1355.0	546.0	541.0	546.0	545.0	522.0	472.0	470.0	522.0	470.0	524.0	471.0	527.0	516.0
praxias	9.0	114.0	114.0	112.0	66.0	64.0	66.0	64.0	98.0	96.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
velocidad_motora	9.0	638.0	638.0	630.0	359.0	359.0	478.0	482.0	379.0	374.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
velocidad_procesamiento_informacion	12.0	769.0	769.0	757.0	471.0	474.0	468.0	470.0	462.0	462.0	424.0	385.0	384.0	0.0	0.0	0.0	0.0

Table 10. Comparison of the number of patients that result from the filtration process by area and column. The First column is the number of remaining features. The first, second and three represent the Sexe, Data Naixament and Diagnostic. Table for 50% threshold.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
WM	5.0	772.0	772.0	762.0	770.0	771.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
atencion	6.0	1391.0	1391.0	1374.0	772.0	773.0	766.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
capacidad_visuoconstructiva	7.0	922.0	922.0	911.0	545.0	538.0	483.0	467.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
capacidad_visuoespacial	5.0	553.0	553.0	545.0	376.0	378.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
capacidad_visuoperceptiva	5.0	84.0	84.0	82.0	73.0	73.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
conducta_emocional	17.0	45.0	45.0	45.0	45.0	42.0	45.0	41.0	45.0	41.0	45.0	41.0	45.0	41.0	45.0	41.0	45.0	41.0
escalas_clinicas	4.0	153.0	153.0	151.0	153.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
funcion_ejecutiva	3.0	1266.0	1266.0	1247.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
lenguaje	3.0	886.0	886.0	871.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
memoria	5.0	1372.0	1372.0	1355.0	959.0	766.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
praxias	9.0	114.0	114.0	112.0	66.0	64.0	66.0	64.0	98.0	96.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
velocidad_motora	9.0	638.0	638.0	630.0	359.0	359.0	478.0	482.0	379.0	374.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
velocidad_procesamiento_informacion	11.0	769.0	769.0	757.0	471.0	474.0	468.0	470.0	462.0	462.0	424.0	385.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 11. Comparison of the number of patients that result from the filtration process by area and column. The First column is the number of remaining features. The first, second and three represent the Sexe, Data Naixament and Diagnostic. Table for 60% threshold.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
WM	5.0	772.0	772.0	762.0	770.0	771.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
atencion	3.0	1391.0	1391.0	1374.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
capacidad_visuoconstructiva	3.0	922.0	922.0	911.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
capacidad_visuoespacial	5.0	553.0	553.0	545.0	376.0	378.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
capacidad_visuoperceptiva	5.0	84.0	84.0	82.0	73.0	73.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
conducta_emocional	17.0	45.0	45.0	45.0	45.0	42.0	45.0	41.0	45.0	41.0	45.0	41.0	45.0	41.0	45.0	41.0	45.0	41.0
escalas_clinicas	4.0	153.0	153.0	151.0	153.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
funcion_ejecutiva	3.0	1266.0	1266.0	1247.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
lenguaje	3.0	886.0	886.0	871.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
memoria	4.0	1372.0	1372.0	1355.0	959.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
praxias	5.0	114.0	114.0	112.0	98.0	96.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
velocidad_motora	5.0	638.0	638.0	630.0	478.0	482.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
velocidad_procesamiento_informacion	9.0	769.0	769.0	757.0	471.0	474.0	468.0	470.0	462.0	462.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

We can conclude several things from these tables. First of all, there are some areas that we cannot analyze because the amount of data is limited: *praxias*, *escalas_clinicas*, *conducta_emocional* and *capacidad_visuoperceptiva*. We have noticed this previously from the analysis of the summaries. We

conclude that the best value for the threshold is fifty percent. With this threshold we obtain a significant amount of data. The number of columns that results from the filter is very low so the initial idea of making first an analysis of each area and then a general analysis between the areas was discarded. We will choose the option of unify everything in a unique database keeping in mind which column correspond to which area. Another important thing that we can conclude is that the data is very sparse. For example, in the case of *memoria* we have 1372 rows in the data base and the column with more data have 959 rows. So there are 413 extra rows with data sparse in other columns. This is easy to visualize in the following map.

Figure 10. Graphical description of the amount of missing values for the memoria area. We don't include the name of each column because it will result in an ugly building figure. We can see that the data base is very sparse. Figure made employing messingno library.

This is normal because the hospital has some main tests that they pass always another more concrete ones that are less frequent. We also realize that the structure of columns in the excels was not well organized so we decided to transform it into the three level structure described in chapter *About the database*.

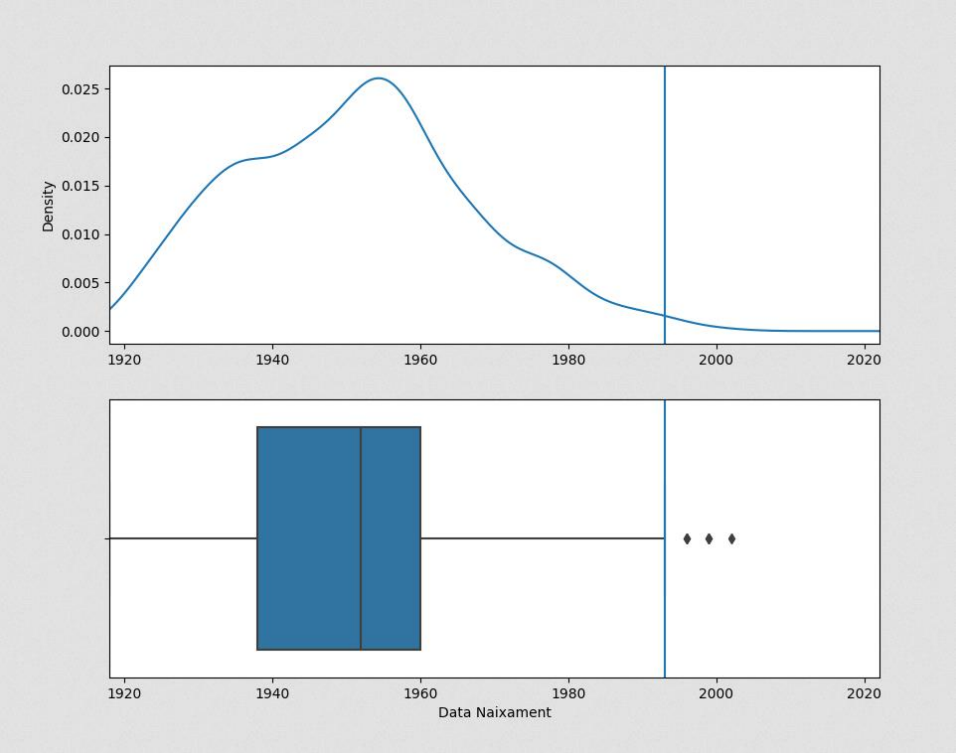


Figure 11. Example of a graphical representation of the outliers in the Data Naixament feature. The first plot represents the distribution of data in a line plot and the second one in a box plot. The 1.rxIQR line is represented in both.

As we are searching for patterns between the different tests we thought that the best approach was trying to get conclusions about this main tests and then if possible try with some of the other ones in further steps of the investigation.

As the study was specially related with the development of Alzheimer disease we visualized and eliminated those patients that were outliers in the *Data Naixament* feature, the year of birth. As this disease has more incidence in older people. We also exclude these patients because as they are very unique in the database we can have anonymity problems. In order to filter out the outliers we employ a method described in the *Summarization* section. This method employs the concept of interquartile range. The previous figure we can observe the age distribution for the *memoria* area and the box-plot with the outliers. We also put a line representing the *1.5xIQR*. We systematically apply this method to the databases that have passed through the filter.

As we have removed a lot of columns we need to generate a new dictionary with the tuples employed to set the three level structure. After generating this dictionary, we store it into a json file. We also store

the resulting databases from the filtering process into a new set of excel files. Employing this excel files and the generated json we will create a unified database in the next step.

7.7. Unify Databases

As we have described, it is necessary to unify all the database with the most common proves passed by the patients. We load all the excels from the filtered files set. In order to correctly merge all the databases into one we need to change the structure of the columns in order to easier compare them. We made that in case of that some areas had columns in common. We merged all the columns into a unique database using the combined id as reference. After making that we did some visualization.

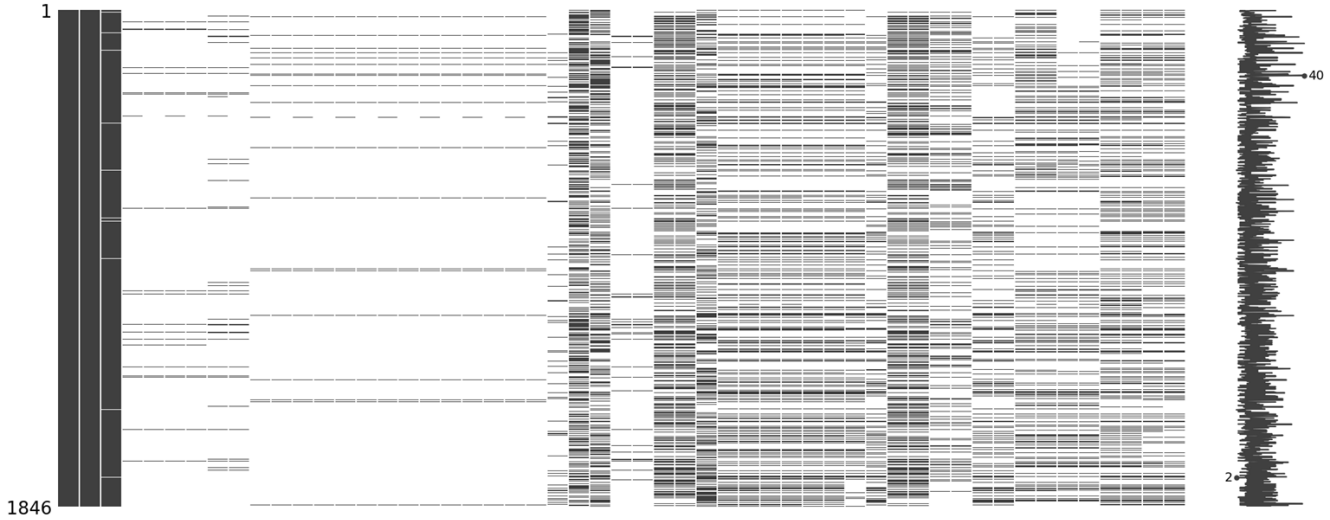


Figure 12. Graphical representation of the amount of missing values for the unified database without filtering. Made employing missingno library.

From this picture we can see that a lot of columns with very less data remains. This is due to the sparsity of the data among the databases and fact of have filtered them in a separate way. We will filter again this database in order to exclude these columns. We can see the result in the next figure.

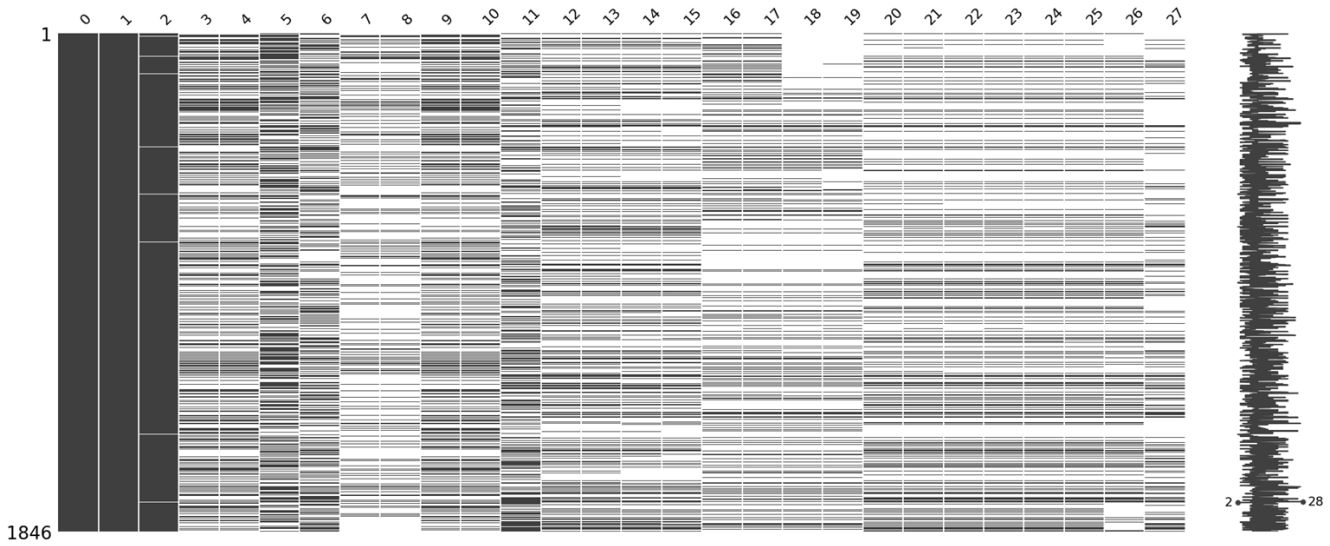


Figure 13. Graphical representation of the amount of missing values for the unified database after filtering. Made employing missingno library.

We drop the columns with less than 20 percent of rows. The result is a still very sparse database. We will see if employing some strategies for filling missing values we can final achieve some results with good accuracy. The database and the three level structure of columns is saved into an excel and json files respectively.

7.8. Outlier detection

In this notebook we filter out possible outliers introduced in the database due to human errors. To make that we use the categories from the hospital. We know that the maximum value is one hundred except for *PD* punctuation. The values that exceeds this magnitude will be transformed into missing values. In the case of PD punctuation as we don't know the ranges we let the values as they are but we search for values very big with only one digit after the decimal point because we think that in this cases there was a mistake when introducing the value in the database.

7.9. Discretization

Here we implement discretization of data employing categories from the hospital. For each type of punctuation, the hospital has different forms of assign categorical values. For example, in the case of *PT* punctuation:

60-40 límite inferior/ 30-40 déficit leve/ 20-30 déficit moderado/ <20 déficit grave

So we used this defined ranges to form categorical variables but we assigned a numerical value to then. This process is made by defining series of lists of bins for each kind of punctuation. In some especial cases for some proves the rules change so we also need to include this special cases. Finally, in the case of *PD* punctuation as is a relative scale and is different for each prove there is not rules defined. We split the data into five categories, as in the case of *PT* punctuation. The bins are made by taking the range of data and splitting it into equal size parts. Finally, the resulting database is stored.

7.10. Analyzing Data Distribution

Here we make some plots to easy understand the form of the final database in order to accomplish the task. First of all, we make a general graphical description of the correlations between variables. It's a matrix of plots where we show the scatter plot of each feature against the other. In the main diagonal we present the data distribution of each feature. This image is included in annexes. In the plot we can see some linear correlation between features which give us the hope of finally find some interesting patterns. As the image is very general we also performed a plot representing in a set of histograms the distribution of each feature.

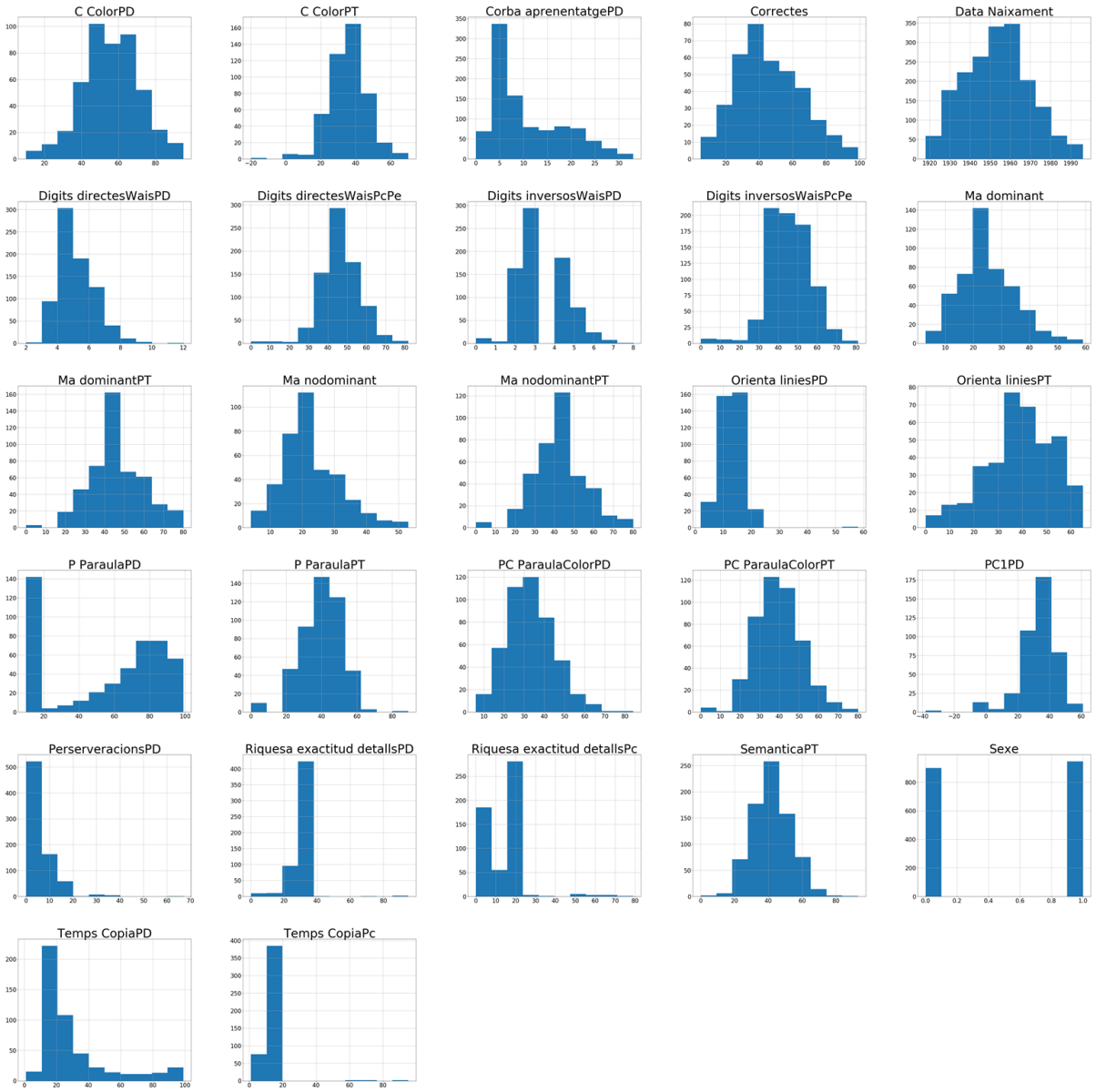


Figure 14. Histograms with the data distribution by feature for the final database. At the start of the process

The image is also included in the annexes part. What we can see in this graphic is in general terms two types of distributions. Some of them follows a more or less normally distributed. The others present a lot of values in the same range and some values very extreme. We think that this extreme values are related with patients with a bad diagnosis. We take the decision of not filter them out as outliers because they are in the test range. Provably these values would be helpful if our task was to infer a diagnosis. As we are searching for patterns between the proves at the start of the data mining process we don't really know if these extreme values will be helpful or not.

7.10.1. Some notes after starting the mining process

During the process of data mining we realized that some of the features have distributions that yield to a misinterpretation of the results. The mentioned features don't follow a normal distribution and have most of the values in the same range. The problem is that when attempting to predict the value of this features some models like the linear ones just give us the mean of the column, the error is very low since all the values are very close to it but we cannot extract information about the relation with other columns. To solve this problem, we performed some plots of the distributions of each column with the objective of eliminate these columns. After eliminating them we have the following distributions:

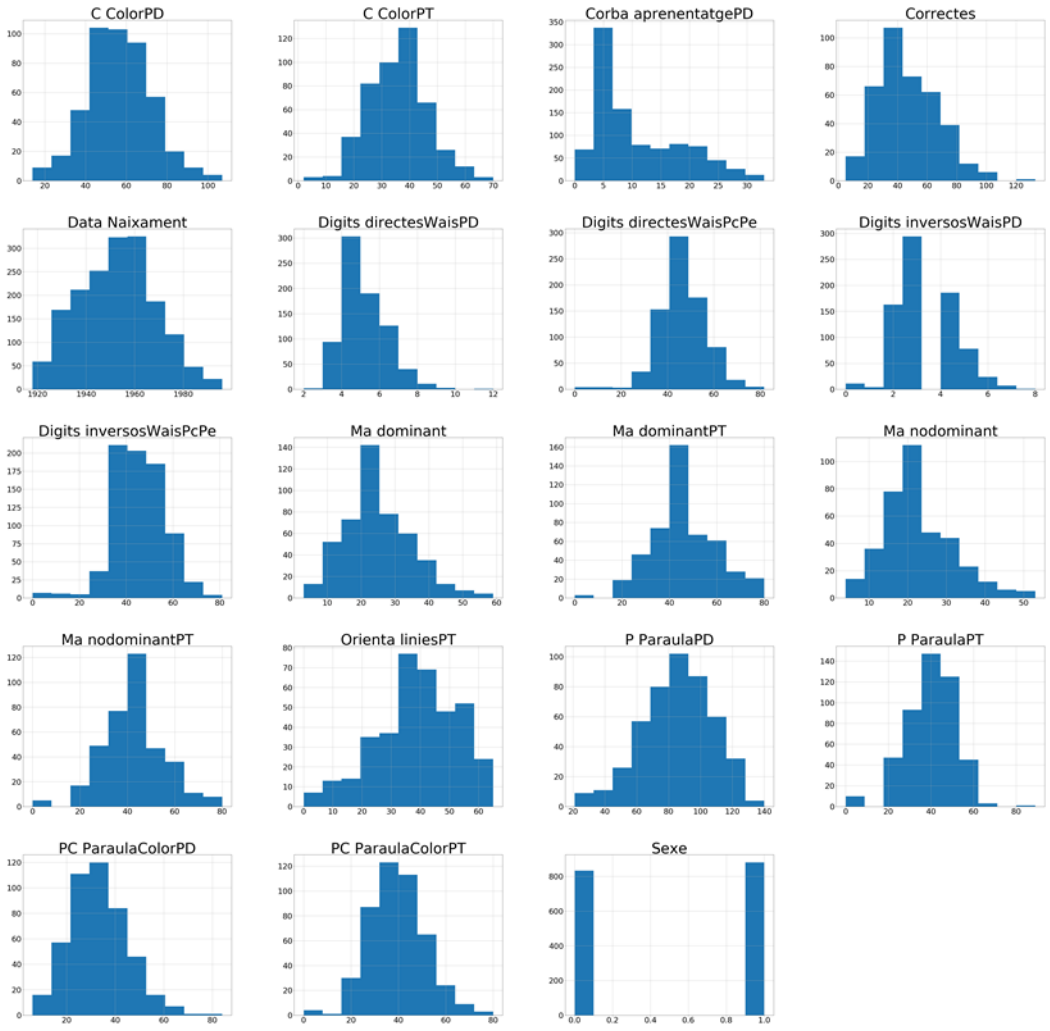


Figure 15. Histograms with the data distribution by feature for the final database. At the end of the process.

The distributions seem to be okay now. But after filling the missing values they can change and give us problems so in the data mining part of the process we will make some additional plots to solve this. Another interesting plot is the following one where we represent all the distributions together:

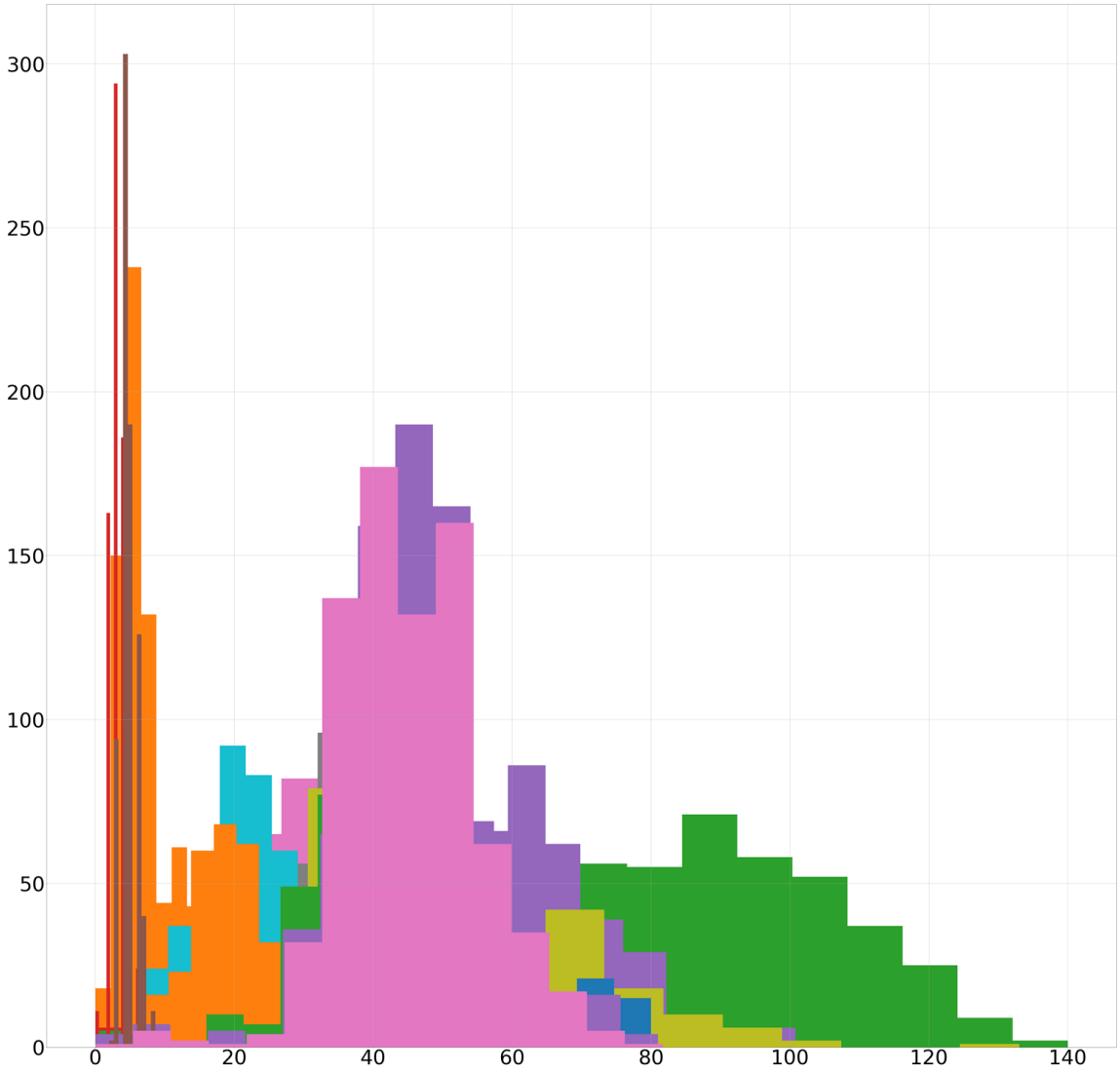


Figure 16. Comparison between the different distributions

7.11. Filling Values

We create two different scripts for filling the missing values. For the continuous case we make use of the function *sklearn.preprocessing.Imputer()* that allows the option of choosing between the mean, median or most frequent value. In the discrete case we iterate each column filling the missing values with the use of the method *fillna()*. We employ the maximum value of the category to emulate the most

pessimistic case when the patients have the worst results in the tests, the minimum for the opposite scenario and the value of the middle for the neutral case. The code can be founded in annexes.

8. Data mining

In this chapter we present basic a general mark of this step of the KDD process. Is usual to identify the KDD process as data mining but the KDD process covers more steps. With data mining we refer to the process of obtaining knowledge from data stored in a database employing several techniques like mathematical models and machine learning algorithms. To achieve this task as we have seen is really important to properly transform the original database in order to clean it and select the most important features. The point is that data mining represents around a twenty percent of the whole process so identify it with the whole KDD process is not correct from our point of view. That data mining is nowadays very popular due to the tools employed and the interest in this tools by companies and researchers.

Perhaps of that there are some general considerations to be taken in account. We are going to talk about the general mark of this techniques in order to easier understand what we do in our application to a concrete case. The amount of models and algorithms susceptible to be employed is really huge and some of them are really complex. We are not going to talk about all of them. We will only mention the ones employed by us in the project. They will not be treated in great detail as the amount of literature generated about it is really huge. We will do a brief introduction of each of them in a way that can be understood by a reader not related with the field.

8.1. Overview

We will follow the definition of machine learning proposed by Tom Mitchell (1997, p.14) that stands that:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

In the application to data mining the task T would be predicting something or finding some pattern. The experience E would be the number of examples in the database and the performance measure a confidence interval or other statistical measures, we will talk in great detail about this measures in the next chapter.

If we attend at the kind of experience that we are going to provide to the learning algorithm several things can be considered. In general, we can identify experience with data we have. The algorithm will

learn from it inferring possible hidden rules and patterns. Data can be dynamically generated by some device or stored in a database. Attending to the kind of data we provide we can split the different machine learning algorithms into three groups:

- Supervised learning: the provided data is labeled; we know the correct prediction. For example, we want to infer the value of some of the features based in the other ones. The learning task is said to be supervised because the learner is prepared through a training process in which we tell it when the required task is well performed or not. We can do that because we know the outcome.
- Unsupervised learning: the provided data is not labeled; we don't know the correct result. The model will be prepared by inducing possible structures present in the data. The learning task will be to induce a general rule that governs the data or to classify sets of instances by similarity given a distance measure.
- Semi-Supervised learning: There is a mixture of labeled and unlabeled data, sometimes we know the correct result and sometimes not. There is a desired prediction but the model will also employ the unlabeled examples to learn possible structures that could help in the resolution of the problem.

We intentionally left out reinforcement learning because is not related with our project. In this kind of problems, the model renamed as *agent* is supposed to decide the best action to choose based on his current state.

This classification is made attending to the experience provided if we attend to the kind of tasks that the algorithm can made we will fall in one of the following categories:

- Regression: We attempt to predict the value of a continuous output based on a set of features.
- Classification: We attempt to predict the classes of a categorical output based on a set of features.
- Clustering: We attempt to group similar instances together based in a similarity distance measure.
- Association: We attempt to extract rules and patterns from the data set. In general, the rules explain relationships between different variables.

Depending on the data we have the tasks performed could be different. To be didactic we resume the associations between the experience or data provided and the type of tasks that can be performed in the following schema.

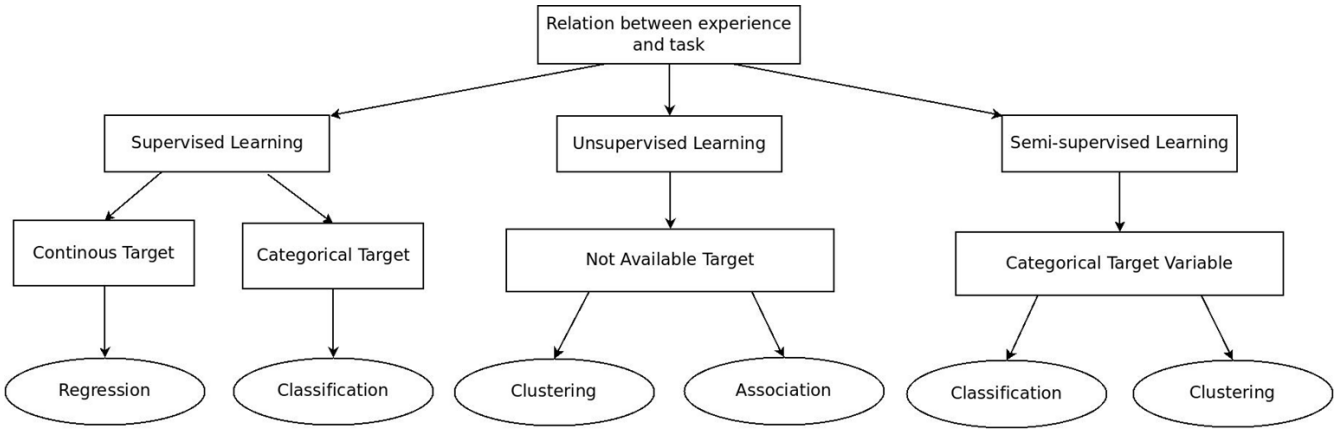


Figure 17. Schema relating the experience provided to the algorithm with the kind of tasks performed.

After training the model we need to validate the predictions made. Due to that fact in most of the cases we will split the available data into two or three sets: The training, test and validation sets. There are different strategies for making this split. We will list some of them with this advantages and disadvantages.

- Internal validation: There is not split made. The same data employed in the training set is employed in the validation process. The problem with that is that we can incur in overfitting the model. This means that the model fits so well the given data that when giving a different behavior the model could be biased and not performing as expected. The good point is that as we don't split the data the method is useful when the amount of data is small and is easy to implement.
- External validation: We test the model with new data, a randomly chosen sample from the initial set. The bad point about these method is that it requires more data than the previous one and are more difficult to implement. The good point is that eliminate overfitting problems so the estimation of the performance is more realistic at least for a similar population. Depending on how we split the data we can talk about:
    - ↘ Train-Test: We randomly choose a twenty-five percent of the data to be the test set. The other seventy-five percent will be the train set. We adjust the model employing the train set and then estimate the performance with the test set. The estimator based in this method use to have high variance when we don't have much data so in this cases the method may not be reliable.

- ↘ K-fold cross validation: We randomly split the data into *K* sets. We iteratively take one of the sets as test set and the remaining ones as training set. After having using all the *K* sets as test set we take as estimator the mean of the resulting values or a sample of them.
- ↘ Lift one out cross validation: Is the extreme case of K-fold validation where *K* equal to the number of instances. If we have *N* instances of data, we take one of them as test set. With the remaining *N-1* instances we train the model. We evaluate the model with this unique instance and repeat the process till have used all the instances as test set. After that we take as final estimator the mean of the results from the evaluation on each instance. The method is useful when the amount of data isn't too much because we don't waste data. The bad thing is that is computationally expensive as we have *N* models to evaluate. As we only take one instance each time we need to be careful with outliers.
- ↘ Train-Test-Validation: We split the data into three sets. Train, test and validation. The usual proportion is (75%, 15%, 15%) or (40%, 30%, 30%) if enough data. This method is usually employed when making comparison between different algorithms. We employ the train and test set as in the Train-Test validation for each model. The validation set that remain is used to test the model in a more reliable way. To be more exhaustive we should finally employ fully external data to test its working performance.

8.2. Choosing machine learning algorithms

- Linear models: They are easy to interpret, not computationally very expensive and exist several types. Some of them employs regularization with different norms. In this category we have Ordinary least squares (OLS), least absolute shrinkage and selection operator (LASSO), Ridge and elastic net. In our project we will make use of OLS and LASSO.

Here we talk about how we have chosen the algorithms to employ. We take a data driving approach to select the set of algorithms to employ also we think about the computation effort and the complexity of the interpretation. In the case of the continuous database we will apply regression algorithms and classification for the discrete one.

8.2.1. Regression

- Nearest neighbors: This algorithm makes a local approximation of the function that defines the output based on the *K* nearest points to the one that is being approximated. Due to that fact it employs a distance metric, the most common are euclidean and manhattan. It's also important to properly choose the number of neighbors. For this purpose, some techniques can be employed like bootstrapping. In the case of classification, the output is one of the classes chosen by a majority vote of the neighbors. In the case of regression, the output is a continuous value calculated by the average of the values of its neighbors.
- Decision trees: There exist several methods for constructing decision trees, one of the most popular is CARTS (Classification and Regression Tree) methodology. The method searches for the feature that better splits the data into different sets. You can see that as choosing recursively the feature which knowledge of its value give us more information about the output. This process will construct a tree which nodes are different features. The children of each of this nodes are the values that the next meaningful feature can take based on the previous one. So the nodes up of the tree are the most relevant features and the conform you go down of the tree the features start to being less important. A common problem with these algorithms is overfitting. This could be corrected by specifying a maximum length for the tree or by post pruning it.

8.2.2. Classification

- Nearest neighbors and Decision trees: works in the same way we define in the previous section but the output is a class instead of a continuous value.
- Logistic Regression: Similar to linear models logistic regression is a type of classification algorithm that uses an equation as representation like linear regression. This equation is the logistic or sigmoid function defined as:

$$f(x) = \frac{1}{1 + e^{-x}}$$

The model combined linearly input values assigning weights to them as in a linear regression model. A possible example:

$$f(x) = \frac{e^{b_0+b_1 \cdot x}}{1 + e^{b_0+b_1 \cdot x}}$$

The key difference is that the output value modeled is a binary value rather than a continuous one. Although the method is usually employed in binary classification problems it could be changed to work in multiclass problems for example taking strategies like One-vs-All. We will not enter in a detailed description of this cases.

- Support vector machines: A simple and brief description of this kind of algorithms is the search of the *n*-dimensional hyperplanes that better split the data in different classes. Several approaches can be made for that. In general, the algorithm has two criteria: first choose the hyperplanes that better split the data into the right classes and second choose the ones that maximizes the distance from the points of one class to the plane. The number of hyperplanes so called support vectors will vary with the number of classes.

8.3. Measures employed

Left to talk about the measures employed to evaluate the performance. This is a fundamental part of the data mining process for several reasons. It allows us to: quantify the model performance, compare different models in order to choose the best one and evaluate if the model is reliable enough. Depending on the type of algorithm and outcome the available measures will be different. We also need to take into account the characteristics of each measure in order to avoid biased conclusions. We will relate each measure with the previous tasks described. In general, this measures compare employing different mechanisms the difference between the true values and the predicted ones, due to that fact is so important to employ a test set as we have exposed in the previous section.

8.3.1. Continuous target measures

This measures are employed when we make predictions about a continuous value. This measures come from the field of statistics and are employed to measure the similarity between two variables of

paired observations that express the same phenomenon. This apply to the case of predicted versus observed values. We will list some of them:

- Mean absolute error (MAE): Is defined as the average between all the absolute errors. The absolute error is defined as the absolute value of the difference between the predicted and true value. This estimator is not normalized in the sense that if the predictions are very bad it can achieve very high values in the other hand values close to 0 indicate a good performance. Being *y* and *y'* vectors of length *N* representing the true and predicted values respectively we can define:

$$e_i = |y_i - y_i'|$$
$$MAE = \frac{1}{N}\sum_{i=1}^{N} e_i$$

- Root mean square error (RMSE): Is defined as the square root of the mean of the squared absolute errors. In the previous context we can define:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (y_i - y_i')^2}$$

In general, the problem with this measures is the stability of them in the sense that we can have very good predictions for most of the cases but if in a few cases the predictions are very bad they can yield to a biased interpretation of the estimator. For these reason is very important to previously filter possible outliers.

- Coefficient of determination (R^2): This a normalized estimator for linear regression models that can take values between 0 and 1 being 0. The better model fits the data the closer the value is to 1. Some definitions of this estimator also allows negative values changing the range from -1 to 1. The estimator quantifies the relation between the variance explained by the model and the total variance of the dependent variable. W can calculate it with the following formulas:

$$R^2 = 1 - \frac{\sum_{n=1}^{N}(y_i - y_i')^2}{\sum_{n=1}^{N}(y_i - \hat{y})^2} = 1 - \frac{RSS}{TSS}$$

Where $\hat{y}$ is the mean of the true values, *RSS* is the residual sum of squares that express the variance between the predicted and true values and *TSS* is the total sum of squares that express the variance of the target. As this estimator is only employed in linear regression models the value of *RSS* can't be greater

than *TSS* as every line different from the one defined by the mean of the dependent variable values will fit the data at least as good as it. Some problems with this estimator are that as it express proportion in the variance a model that fits the data poorly but have variance similar to *TSS* could give to values close to 1. Due to that fact a graphical representation of the residuals is always needed. The other problem with this estimator is the so called inflation of R^2. This estimator weakly increase with number of regressors in the model so it can yield to very complex models that in fact will fit that data only a little bit better than simpler ones. Due to that fact a penalty in the number of regressors can be included. This yields to the definition of the *adjusted coefficient of determination*.

8.3.2. Binary target

In the cases when the target is categorical and we only have two different classes there are some specific measures in most of the cases derived from the concepts of true positive, false positive, true negative and false negative. This kind of measures comes from medical testing. A positive is refereed as the fact of suffering a disease and negative of not suffering it. When your diagnosis is correct it is said to be true and when incorrect to be false. This explain the refereed concepts and derived measures. In binary target classification normally one variable is the negation of the other one so the assign of what is a positive and what is a negative is clear.

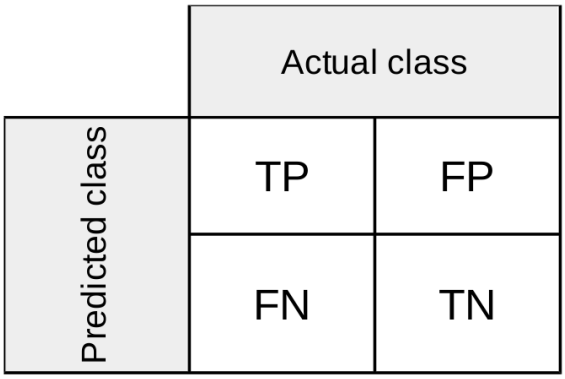


Figure 18. Hypothesis testing binary target

We define the most employed:

- Accuracy: It measures the proportion of correct predictions over the total.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- Precision: It measures the proportion of positives over the examples predicted as positive.

$$Precision = \frac{TP}{TP + FP}$$

- Recall: It measures the fraction of actual positives that are predicted as positive.

$$Recall = \frac{TP}{TP + FN}$$

- False positive rate: It measures the fraction of actual negatives that are predicted as positive.

$$FPR = \frac{FP}{FP + TN}$$

- F1-score: It's the harmonic average of the precision and recall.

$$F1score = 2\frac{precision \cdot recall}{precision + recall}$$

In general, what we seek for models with high precision and recall. This is summarized in the *F1* score.

8.3.3. Multiclass categorical target

This measures are employed when you are in a classification problem with multiple classes. In general, the measures are a set of scores that indicates the certainty of the model that the predictions belong to each of the classes.

- Confusion matrix: In one of the most employed measures and several measures can be derived from it. We compare the predicted classes with the actual ones. In general, the vertical axis of the matrix corresponds to the predicted values and the horizontal axis to the actual ones. In this way each row in the matrix represent the amount of times that the model classifies an instance in each one of the classes. The diagonal shows the number of correct classifications and the of diagonal cells shows the number of misclassified predictions.

You can employ the confusion matrix as it is but is easier to derive some measures. The usual approach is to make a One-vs-All classification. In this approach we take one of the classes as positive and group all the classes in one class as negative. With this approach we can extend all the concepts from binary targets to multiclass target. In the next schema we put an example of how to interpret this confusion matrix in terms of true positive, false positive, true negative and false negative for the first class.

	Actual class			
Predicted class	TP1	FP1	FP1	FP1
	FN1	TN1	TN1	TN1
	FN1	TN1	TN1	TN1
	FN1	TN1	TN1	TN1

Figure 19. Confusion matrix four classes showing the TP, FP, TN and FN cases for the first class in the One-vs-All approach.

Once we understand that, we can list the previous measures for each class or we can average them. When averaging we can take two approaches micro and macro:

- Micro-average method: You generalize the original formulas from binary target to this multiclass setting. In order to make that you employ the rue positive, false positive, true negative and false negative of each class. This method is preferable if you suspect there can be class imbalance. For *K* classes the recall and precision formulas are transformed into:

$$Precision = \frac{\sum_{i=1}^{K} TP_i}{\sum_{i=1}^{K} TP_i + FP_i}$$

$$Recall = \frac{\sum_{i=1}^{K} TP}{\sum_{i=1}^{K} TP + FN}$$

- Macro-average method: You compute the measure based in the arithmetic mean of the individual measures for each class. This is not recommendable when you suspect there could be class imbalance. The good point of that method is that is computationally less expensive. In this terms we compute precision and recall as:

$$Precision = \frac{1}{K}\sum_{i=1}^{K} Precision_i$$

$$Recall = \frac{1}{K}\sum_{i=1}^{K} Recall_i$$

Finally, we could talk about proximity measures employed in clustering and about measures employed in association but as we don't apply these techniques to our problem we prefer to end here the discussion.

9. Application to our case of analysis II

Here we present the data mining process in our case of analysis. In *Application to our case of analysis I* we described the summarization and preprocessing parts of the project. At the end of preprocessing part, we generated two data frames: *Final Database* and *Final Discretized Database* the first one with the original continuous values and the other one with the values converted into classes. We also presented strategies for filling missing values for both cases. We divide this section into two cases of analysis the continuous and discrete ones. The structure of the analysis is the same in both cases. We applied several algorithms trying to predict the values from one feature employing the values from the rest of features excluding the ones that we know are already related since they form part of the same test. We make a comparison between the different algorithms and finally we attempt to interpret the model and find relations between columns.

9.1. Continuous case

At the beginning of the analysis we load the final database and generate three databases following different strategies for filling missing values from it: *df_mean*, *df_median* and *df_most_frequent*. They contain the original information with the missing values replaced by the mean, median and most frequent values of each column. After running the series of scripts the first time, we realized that there were some problems.

We have filtered out a lot of columns in the preprocessing part with the *Filtering and Visualizing* and the *Unify Database* notebooks. This filtering process was made based in the percentage of rows over the

total for each column. We couldn't be very strict because we will remove almost all columns. So we have a final database where for some columns the amount of missing values was close to fifty percent. Due to that fact when we fill the missing values, the resulting data distribution of each feature has most of the values around the mean, the median or the most frequent value. With that kind of distribution, if the model prediction is just the mean value of the distribution, the average error over all the predictions will be very low. This yield to a misinterpretation of the relations between columns.

In order to solve that we choose another strategy. We iterate each column of the database removing from the index all the cells that have nan values for this column. We employ this index to select the rows from the other columns. If this resulting subset of each column has a proportion of no missing values over the total bigger than a specific threshold we append it to the database that we will analyze. With a threshold of eighty percent we find a subset of the total columns with enough values to analyze. We repeat the process with a threshold of seventy percent and we find another subset of data possible to analyze. Here we can show the structure of the two final subsets of data that we will analyze:

Table 12. Resulting subsets with a threshold of 80%.

	Sexe	Data Naixament	Diagnòstic	Orientación de líneas RBANS	Lista de palabras del RBANS_verbal	Dígitos: inversos (WAIS)		Dígitos: directos (WAIS)	
	Sexe	Data Naixament	Diagnòstic	Orienta liniesPT	Corba aprenentatgePD	Digits inversosWaisPD	Digits inversosWaisPcPe	Digits directesWaisPD	Digits directesWaisPcPe
	Sexe	Data Naixament	Diagnòstic	PT	PD	PD	Pe	PD	Pe
2017-02-07-5020426281260010496	1	1939	1002DEP	13.00	15.00000	2.00000	38.9300	3.000000	35.860000
2014-09-02-2639338705842160128	1	1951	291.2	51.00	22.00000	3.00000	44.0000	5.000000	50.000000
2010-01-127570110366824930304	1	1934	1601DCO	23.57	6.00000	3.00000	49.5700	4.000000	44.480000
2013-11-05-7301915874375520256	1	1966	1601TDE	65.00	24.00000	5.00000	51.0000	6.000000	50.000000

Table 13. Resulting subsets with a threshold of 70%.

	Sexe	Data Naixament	Diagnòstic	Llista de palabras del RBANS_verbal	Verbal: Stroop						Visual: Clave de números - Codificación (WAIS-III)
	Sexe	Data Naixament	Diagnòstic	Corba aprenentatgePD	P ParaulaPD	P ParaulaPT	C ColorPD	C ColorPT	PC ParaulaColorPD	PC ParaulaColorPT	Correctes
	Sexe	Data Naixament	Diagnòstic	PD	PD	PT	PD	PT	PD	PT	Correctes
2013-11-12-1937500370690820096	1	1984	500997	6.09	87.618421	41.634868	58.205298	37.047841	34.286667	40.344482	27.0
2009-06-302840165524484780032	1	1968	1601DCO	7.00	122.000000	56.000000	84.000000	52.000000	60.000000	64.000000	70.0
2013-12-051075829897044720000	1	1957	500356	6.00	84.000000	38.000000	50.000000	30.000000	33.000000	38.000000	48.0

After selecting these subsets, we have to fill the missing values. We employ the mean, the median and the most frequent value as we described at the start of the process. We make visualization to ensure that there isn't any problem with the distributions.

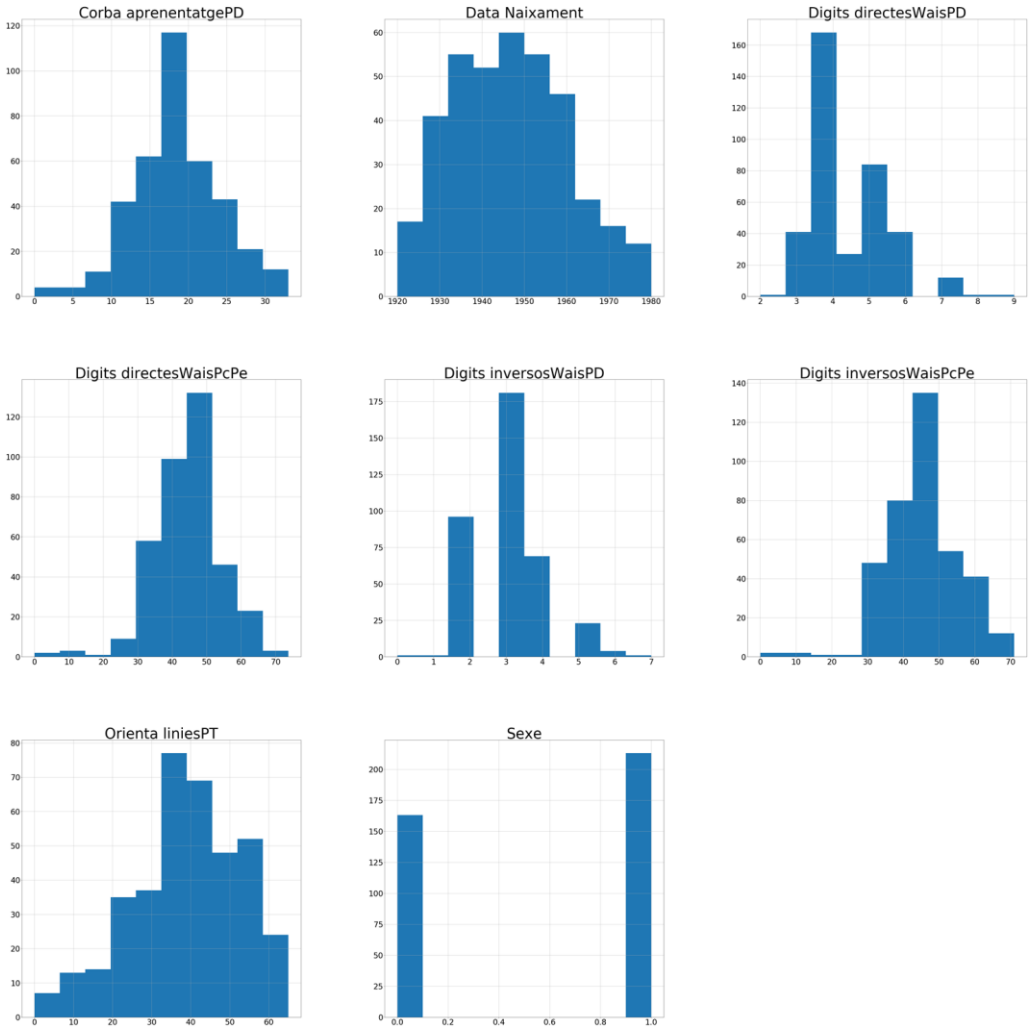


Figure 20. Visualization of the column distribution for the first subset of data

We localize problems with the columns (*'Dígitos: inversos (WAIS)', 'Digits inversosWaisPD', 'PD'*) and (*'Dígitos: directos (WAIS)', 'Digits directesWaisPD', 'PD'*) and we remove them. The resulting histogram shows the distributions of the columns of the subset that remains.

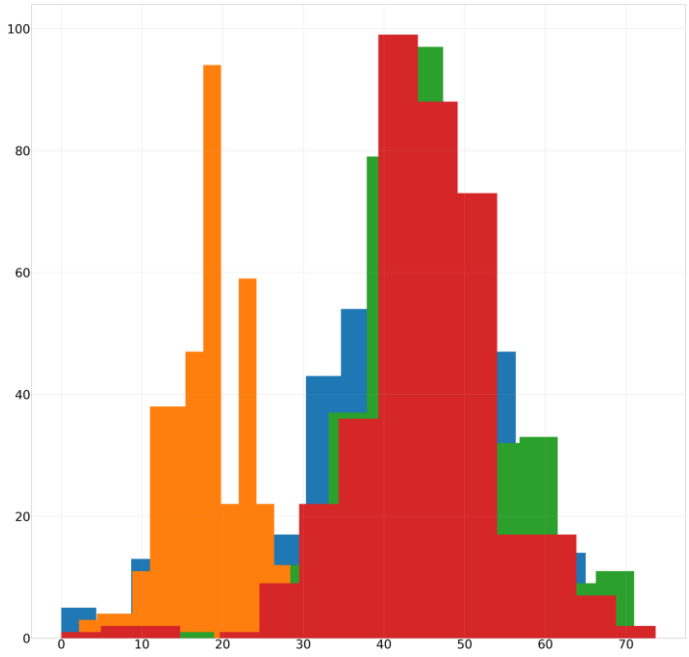


Figure 21. Comparison between distributions for the first subset after removing problematic columns.

The same analysis is made for the second subset. The comparison of the data distribution for the resulting features is showed in the next figure:

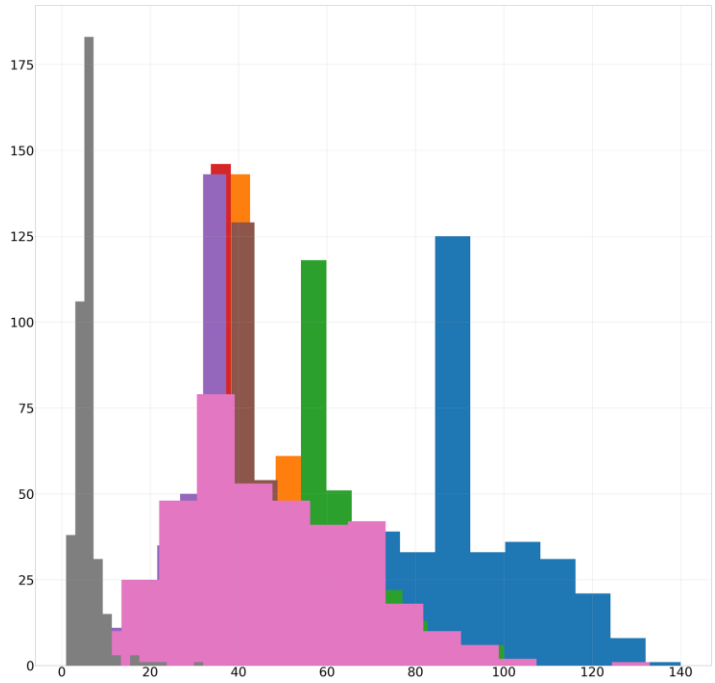


Figure 22. Comparison between distributions for the second subset after removing problematic columns.

In general, we see that the data distributions have low variance as most of the values are around the mean. We will see how this yield to problems in the performance.

Now we describe the analysis process made for each of the columns of the resulting subsets. We create an object called *relation_between_models* where we store the *Mean Absolute Error* (*MAE*), the *Root Mean Squared Error* (*RMSE*), the weights computed by the model and the name of the column we attempt to predict.

The main process is made by the function *finding_patterns_between_columns()* it takes as input the target *Y*, the features employed *X*, the name of the predicted feature *name_predicted_column*, the model and the object *relation_between_columns*. Inside of the function we apply *K*-fold validation with *K*=10. We fit the model with the train sets, with the predicted values we calculate the *MAE* and *RMSE*. We also get the weights and average them over the *K* sets. For more details, the Jupyter notebook is included in the annexes part.

This function is systematically applied over all the columns employing different models: Ordinary Least Squares, Least Absolute Shrinkage and Selection Operator, Nearest Neighbors, Support Vector Machine and Decision Trees. We have tried with different types of normalization but finally we don't apply it. This is because the scales of the different features are similar and the algorithm of LASSO implemented in *sklearn* allow to include an intercept. Standardizing the features could be an option but we try it and gives worst results. The good point of didn't normalize is that the estimators *MAE* and *RMSE* are easier to interpret.

In the case of LASSO, NN and DT we have to choose parameters. In LASSO the penalization parameter, in NN the number of nearest neighbors and in DT the maximum depth of the tree. To make that we made an array of values for this parameters and check the algorithms with them. In the case of LASSO, the best alpha is around *8.5* and in the case of NN the number of neighbors is four.

9.1.1. Comparison between algorithms. First subset

All the information from each case of prediction is stored in a set of excel files. We employ this excel files to make a comparison. We average the *MAE* estimator across each predicted column and represent it in the following histogram:

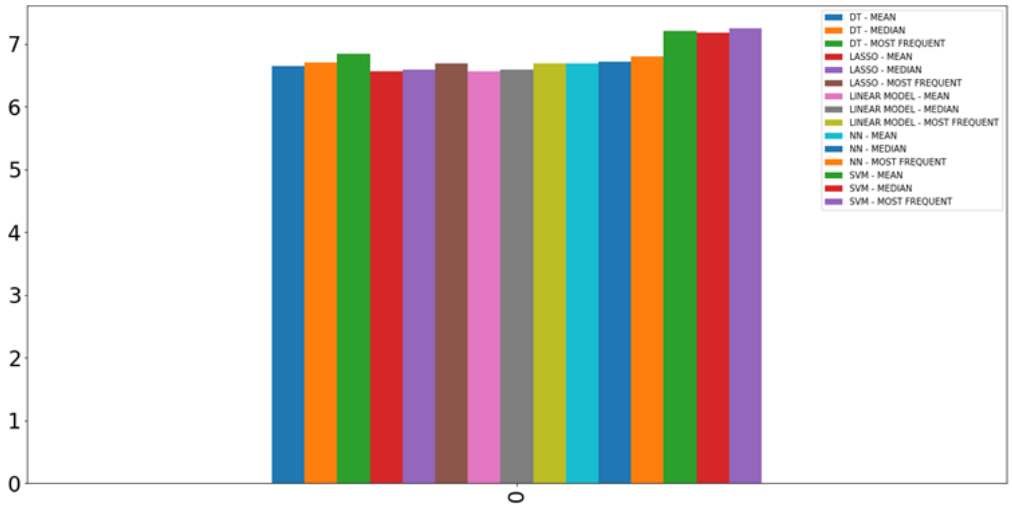


Figure 23. Histogram comparing different models and filling missing values strategies.

In general, the worst scores are achieved for the case of filling missing values with the most frequent value of each column. The best scores are achieved with LASSO. The values of *MAE* are not very good because we achieve an average value around seven and the vale ranges are around seventy so we have around a ten percent of error. We make a more exhaustive analysis of it but first in order to have a better comparison we also make the following plot comparing LASSO and OLS among the different columns:

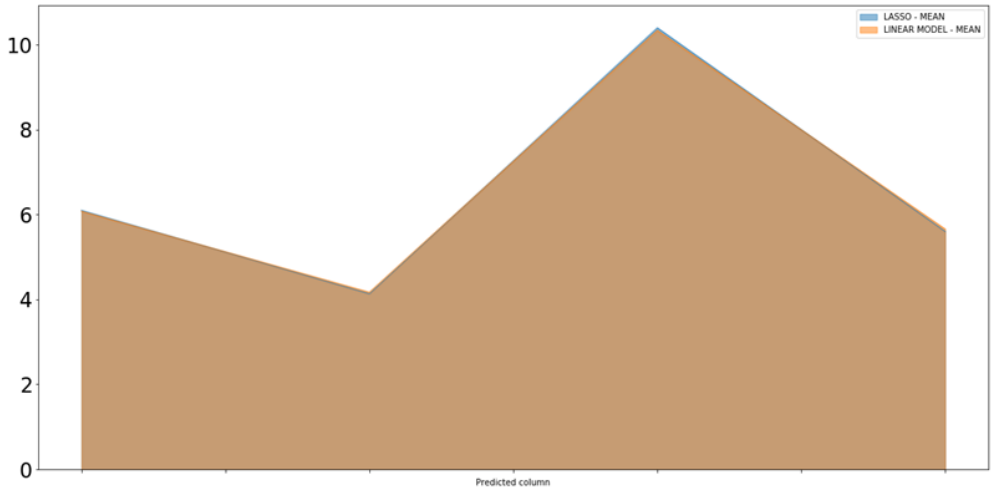


Figure 24. Area chart comparing the MAE values for each column between LASSO and OLS.

We can see that they achieve similar results among the columns. The good point of LASSO is that as it penalizes the number of predictors in the model we can check for relations among the columns in order to see which of them are more related. We had difficulties to understand the error in order to easy interpret them we include the *Mean Absolute Percentage Error* (*MAPE*). The results are the following in the case of linear regression that remember achieves the best results of the set of algorithms employed. We also include the data ranges of each column:

Table 14. MAE, MAPE and RMSE scores for linear regression model and the coefficients related to each predictor for subset one.

MAE	MAPE	RMSE	Dígitos: inversos (WAIS)	Lista de palabras del RBANS_verbal	Orientación de líneas RBANS	Dígitos: directos (WAIS)
MAE	MAPE	RMSE	Digits inversosWaisPcPe	Corba aprenentatgePD	Orienta liniesPT	Digits directesWaisPcPe
MAE	MAPE	RMSE	Pe	PD	PT	Pe
6.080078	9.333271	7.981272	X	0.0774718	0.111562	0.600104
4.171715	41.230697	5.314733	0.033923	X	0.118643	0.046537
10.343106	12.652138	12.940270	0.29061	0.69671	X	-0.00694999
5.658907	7.372175	7.534378	0.526015	0.091681	-0.0019559	X

Table 15. Size of the data ranges of each feature for subset one.

Dígitos: inversos (WAIS)	Digits inversosWaisPcPe	Pe	71.00
Lista de palabras del RBANS_verbal	Corba aprenentatgePD	PD	33.00
Orientación de líneas RBANS	Orienta liniesPT	PT	65.00
Dígitos: directos (WAIS)	Digits directesWaisPcPe	Pe	73.68

We see that we have high errors in most of the cases. In fact, those errors that seems to be low are due to the fact of having data distributions with a lot of values close to the mean. In order to check that we run LASSO with a very high penalty coefficient. With that we will have the coefficients associated with all the predictors equal to 0 and due to the implementation of LASSO in *sklearn* the resulting value will be the intercept that is equal to the mean. We can see the results in the following tables:

Table 16. MAE, MAPE and RMSE scores for LASSO model with extreme penalty value and the coefficients related to each predictor for subset one.

MAE	MAPE	RMSE	Dígitos: inversos (WAIS)	Lista de palabras del RBANS_verbal	Orientación de líneas RBANS	Dígitos: directos (WAIS)
MAE	MAPE	RMSE	Digits inversosWaisPcPe	Corba aprenentatgePD	Orienta liniesPT	Digits directesWaisPcPe
MAE	MAPE	RMSE	Pe	PD	PT	Pe
7.540215	12.443969	9.982315	X	0	0	0
4.235201	37.912879	5.613056	0	X	0	0
11.094095	12.605779	13.846042	0	0	X	0
6.295146	10.691601	9.139173	0	0	0	X

With that we conclude that we cannot extract conclusions form the analysis of this subset of data. We think that the main problem in the analysis is the amount of data and the distribution of this values. We will see if we can solve these problems in the discrete case.

9.1.2 Comparison between algorithms. Second subset

Here we replicate the same analysis for the second subset. In the following plot we represent the comparison between the different algorithms.

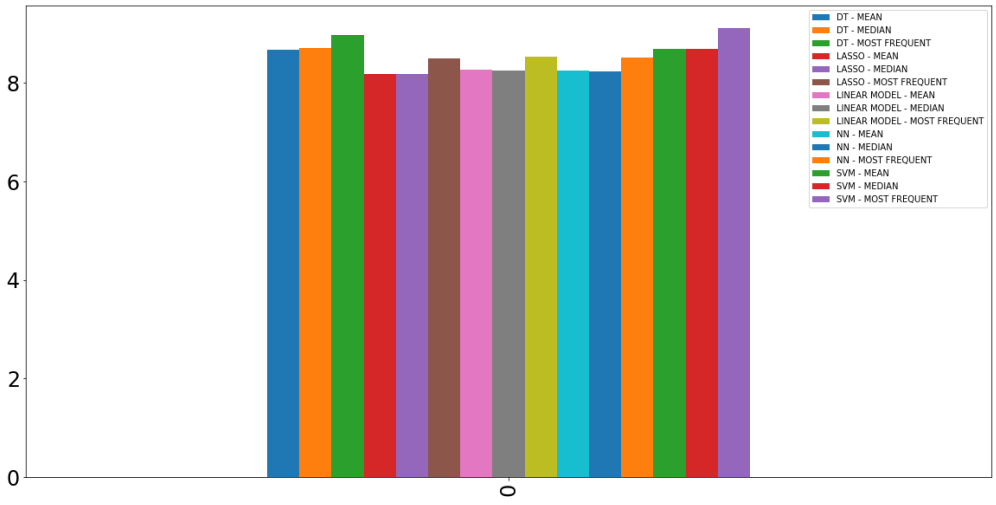


Figure 25. Histogram distribution for pairing the average MEA among all the algorithms.

The results are the same as in the previous section. The models perform better for the case of the filling missing values with the mean value. In this case LASSO performs better than OLS. The comparison between them is showed in the following chart.

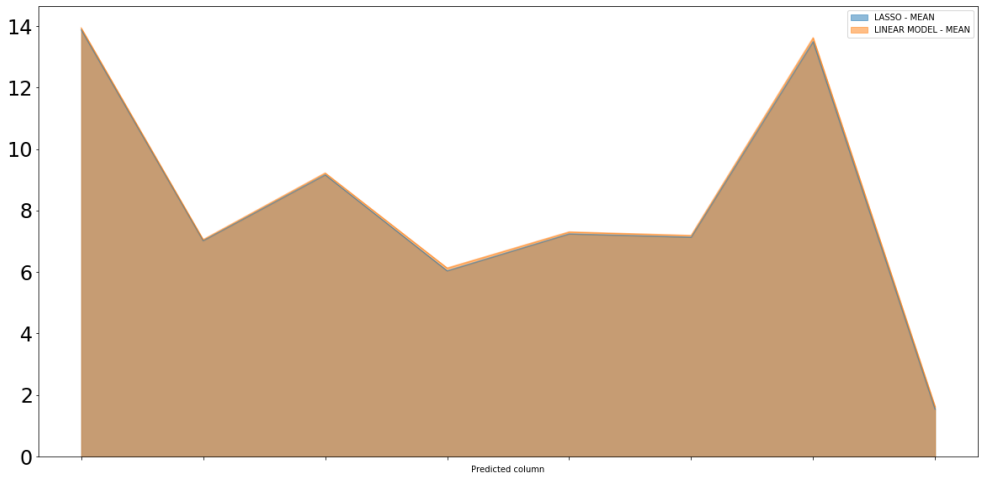


Figure 26. Comparison between LASSO and OLS for each target feature.

The results are almost the same for both algorithms. We show the tables with the weights and errors for the case of OLS. We also present the data ranges of each column. As in the previous case we conclude that there is not a significant relation between the different features.

Table 17. MAE, MAPE and RMSE scores for linear regression model and the coefficients related to each predictor for subset two.

MAE	MAPE	RMSE	Verbal: Stroop						Visual: Clave de números - Codificación (WAIS-III)	Lista de palabras del RBANS_verbal
MAE	MAPE	RMSE	P ParaulaPD	P ParaulaPT	C ColorPD	C ColorPT	PC ParaulaColorPD	PC ParaulaColorPT	Correctes	Corba aprenentatgePD
MAE	MAPE	RMSE	PD	PT	PD	PT	PD	PT	Correctes	PD
13.939949	19.707756	18.337884	X	-	-	-	-	-	0.426665	0.0861653
7.056150	23.308463	9.454333	-	X	-	-	-	-	0.160561	0.183087
9.222497	17.125177	12.081422	-	-	X	-	-	-	0.338182	0.317431
6.131312	18.647884	8.139601	-	-	-	X	-	-	0.196902	0.199707
7.306589	18.701766	9.482882	-	-	-	-	X	-	0.23722	0.145577
7.190383	12.450013	9.369012	-	-	-	-	-	X	0.213345	0.0955687
13.621296	82.870667	17.659305	0.0799879	-0.00248244	0.357128	0.12983	0.441947	-0.0918384	X	-0.241678
1.624461	27.783403	2.778791	-0.0924672	0.164881	0.112448	-0.114234	0.024548	-0.0314803	-0.00641131	X

Table 18. Size of the data ranges of each feature for subset two.

Verbal: Stroop	P ParaulaPD	PD	119.0
	P ParaulaPT	PT	87.0
	C ColorPD	PD	86.0
	C ColorPT	PT	68.0
	PC ParaulaColorPD	PD	78.0
	PC ParaulaColorPT	PT	78.0
Visual: Clave de números - Codificación (WAIS-III)	Correctes	Correctes	128.0
Lista de palabras del RBANS_verbal	Corba aprenentatgePD	PD	31.0

Again the error is very big and we cannot make any conclusions. The problem is due to the low variance in the distributions. To demonstrate that we force LASSO as in the previous section.

Table 19. MAE, MAPE and RMSE scores for LASSO model with extreme penalty value and the coefficients related to each predictor for subset two.

MAE	MAPE	RMSE	Verbal: Stroop						Visual: Clave de números - Codificación (WAIS-III)	Lista de palabras del RBANS_verbal
MAE	MAPE	RMSE	P ParaulaPD	P ParaulaPT	C ColorPD	C ColorPT	PC ParaulaColorPD	PC ParaulaColorPT	Correctes	Corba aprenentatgePD
MAE	MAPE	RMSE	PD	PT	PD	PT	PD	PT	Correctes	PD
14.525520	18.790080	20.173544	X	-	-	-	-	-	0	0
7.233258	23.183318	10.114340	-	X	-	-	-	-	0	0
9.732984	16.713543	13.869168	-	-	X	-	-	-	0	0
6.237147	19.380665	9.008207	-	-	-	X	-	-	0	0
7.385349	16.866585	10.519801	-	-	-	-	X	-	0	0
7.243886	12.562011	10.198438	-	-	-	-	-	X	0	0
16.274930	85.519594	20.038630	0	0	0	0	0	0	X	0
1.540905	24.127975	2.706759	0	0	0	0	0	0	0	X

We can see that these analyses don't allow us to make conclusions about the relations between the models.

9.1.3. Conclusion from this step of the analysis

The previous analysis for the first and second subsets of data show as that the distribution of data is a problem and the low variance of them doesn't allow us to properly train a model. In order to solve that we will try a different approach transforming the continuous values into two classes. Normal and extreme values.

10. Discrete case

Here we tried another approach for finding patterns between the different features. As we have seen in the previous section when the set of algorithms was applied to the different data subsets we didn't find any significant enough relationship. At the start of the data mining process the initial approach to turn the database into categorical values was to follow the different classes employed in the hospital.

With the results from the previous section we realized that this approach had too many classes to success. The problems from the continuous case will appear again. In fact, we made some attempts with this multiclass setting and the initial results where bad in terms of the metrics employed. Due to that, we finally tried another approach: The continuous values would be divided into a binary target. The class 1 represents values of healthy people and the class 2 represents those extreme values related with some kind of disease. The discretization process is made in a new Jupyter notebook called *Discretization (Binary)*, that is also included in the annexes part. We simply changed the ranges of each class to reflect the mentioned behavior. For the case of PD punctuation there weren't any indications from the hospital of what are unusual values so we made the split employing the mean values as threshold.

The point was that we still had problems with missing values due to the sparse behavior of the data base. In order to solve that, we selected a subset of data with a reasonable amount of missing values as we did in the continuous case. We also developed two strategies for filling messing values. The optimistic one where the missing values were filled with class one values and the pessimistic one where the messing values were filled with class two values. Among the mining process we made a comparison between the two approaches in order to know if we are biasing too much the model with these assumptions. The main problem is that these strategies give rise to a very unbalanced classes with the problems that behaves in this kind of settings. We hope that the comparison between both cases of analysis allows us to solve this problem.

The plots with the distribution of values between classes before and after filling missing values can be checked in annexes. (Histogram description binary 1, el optimistic 2 y el pessimistic 2)

The whole mining process was analogous to the continuous case. We defined an object to contain the information related with the performance measures employed and the weights from each algorithm. As in the previous case, the name of the object was *relation_between_columns* and it had as attributes a confusion matrix and the precession, recall and F1 score of the two classes. We implemented a function similar to the described *finding_patterns_between_columns()* employed in the continuous case. It also

employed *K*-fold validation and returned the mentioned object with the average values of each estimator among the different test groups.

We attempted to predict when a patient has values in the test that could be susceptible of being related with having a disease. So the important class to determine was the class two. Due to that fact, although we also calculated the other performance measures for class one, the measures employed to select the best algorithm are those related with class two.

The set of algorithms employed in this case is: *Logistic Regression*, *Support Vector Machine*, *Nearest Neighbors* and *Decision Trees*. In the case of NN and DT we had to select the number of neighbors and the number max depth of the tree. We employed the F1 score in order to choose which of this values was better. It should be mentioned that for the case of Support Vector Machine we initially tried with a *linear kernel* but the performance was poor so we moved to a *radial basis function kernel*. The problem with this last one is that the interpretation of the results is harder.

10.1. Comparison between algorithms

Here we present a brief comparison between the different algorithms employed. As in the previous case, the comparison is made based in a set of histograms that compare the average results of each algorithm among the different measures.

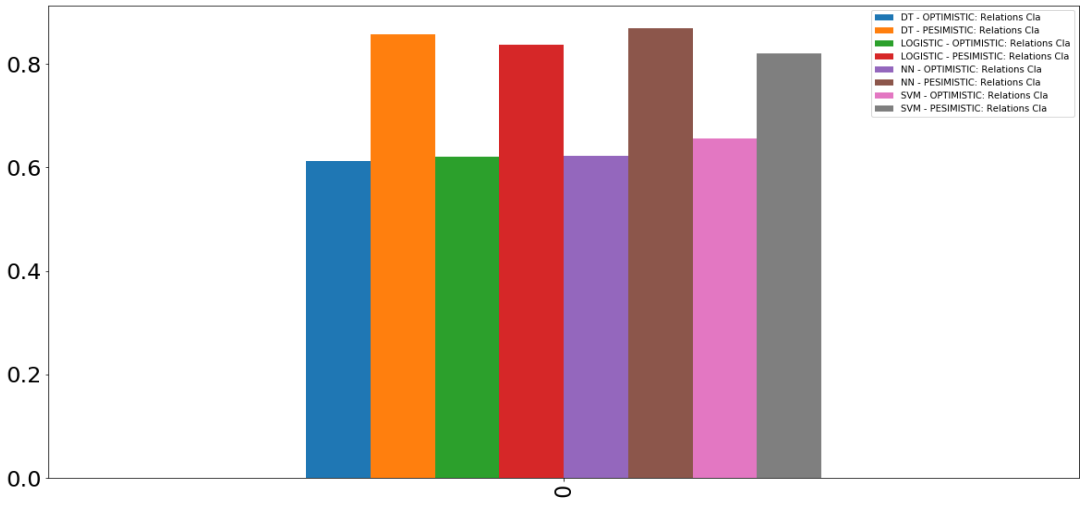


Figure 27. Performance comparison based on average precision.

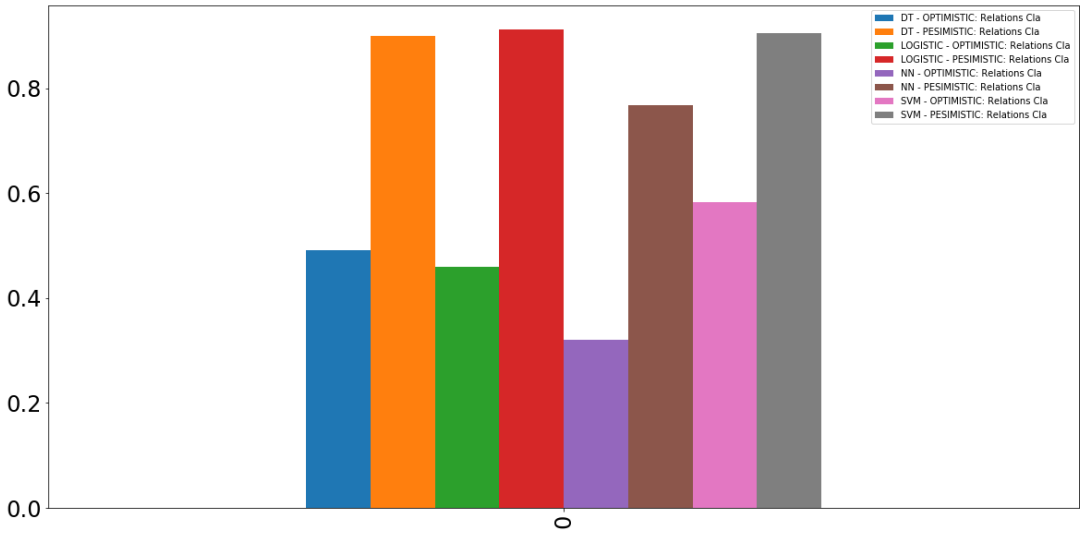


Figure 28. Performance comparison based on average recall

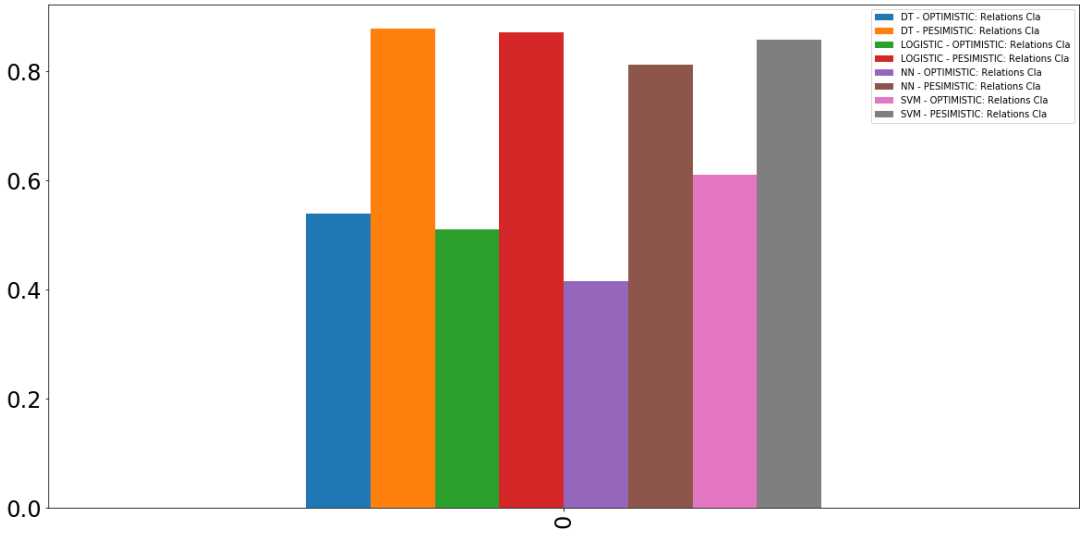


Figure 29. Performance comparison based on average F1 score.

The results were the expected. In the case of the pessimistic approach the estimators achieved best scores as we are evaluating the class two and we have included some missing values in this approach. The results from SVM were significant in the sense that it seemed to be the one that less biased the results if we compare the score achieved for the optimistic and pessimistic approaches. For the case of this

algorithm we also include some graphic representing the evolution of the F1 scores among the different target features for the pessimistic and optimistic cases.

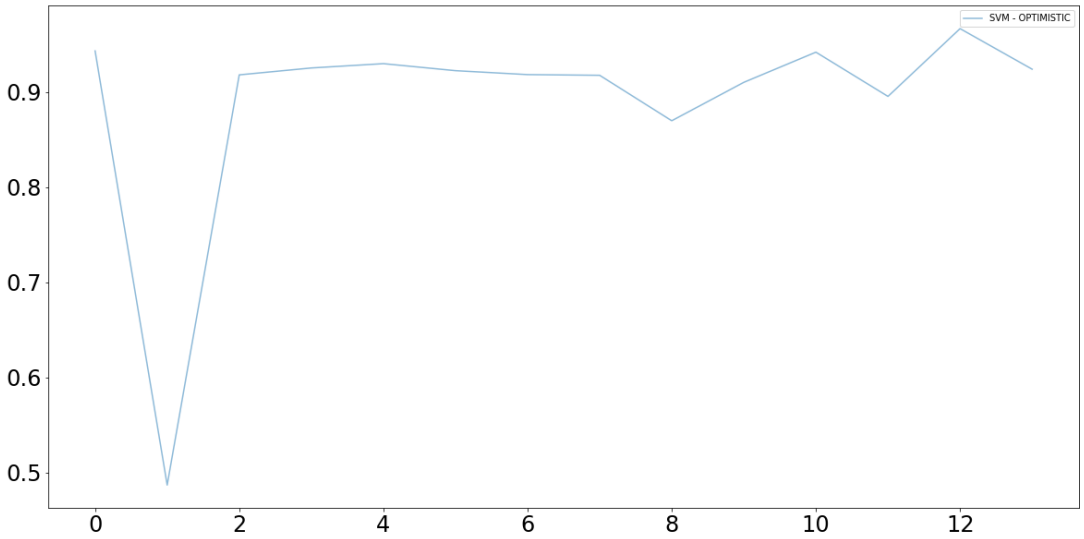


Figure 30. F1 score values along different targets. Optimistic case.

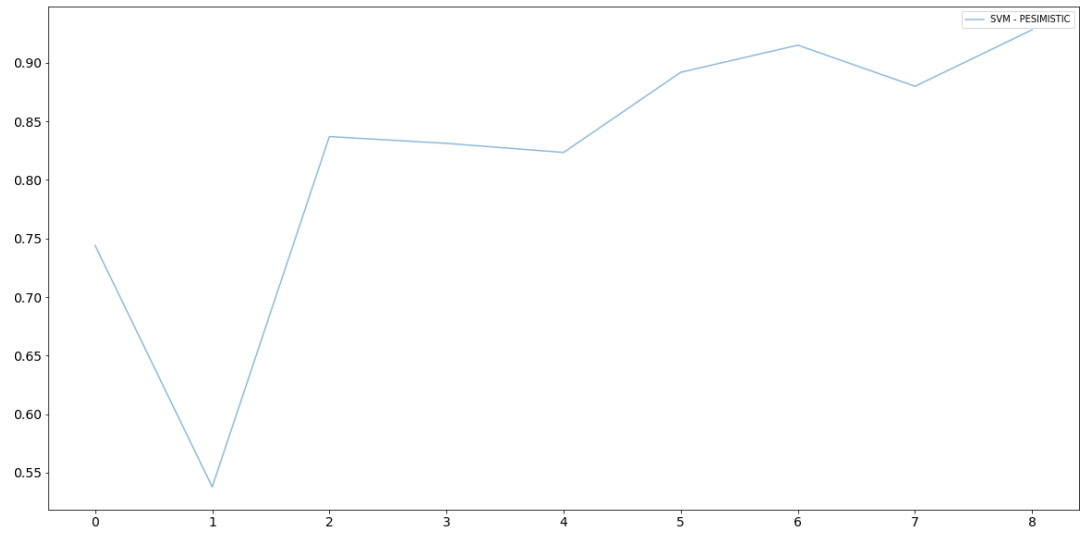


Figure 31. F1 score values along different targets. Pessimistic case.

This F1 scores gives us an idea of the reliability of the model in the attempt of predicting the different features. In the next section we will try to give an interpretation for the founded results.

10.2. Interpreting the results

Finally, we selected SVM as the algorithm with better performance. Although there were algorithms that achieved better results for the pessimistic case this one got a similar result in this case and the difference with the optimistic setting was not as big as in the others. In general, this kind of algorithms perform better in unbalanced classes. The problem is that the interpretation is harder than for example with *Decision Trees* especially with the *radial basis function kernel* employed. Due to that fact we didn't made an analysis describing the importance of the different features in order to predict the target. This was a problem because one of our objectives in the investigation was to describe the relations founded between the different features, but, as the investigation isn't finished yet, it could be an interesting set of first results that could guide us in further steps.

In order to analyze the results, we present a set of tables with the exact values of the different estimators for the prediction of the different features. The tables are also included in annexes for deeper detail.

Table 20. Summary of different estimators for class one and two among all targets. Optimistic case.

Predicted column	Precision_1	Recall_1	F1_1	Precision_2	Recall_2	F1_2
Lista de palabras del RBANS_verbal --> Corba aprenentatgePD --> PD	0,66	0,94	0,78	0,94	0,64	0,76
Sexe12 --> Sexe12 --> Sexe12	0,52	0,49	0,50	0,54	0,58	0,56
Verbal: Stroop --> P ParaulaPD --> PD	0,89	0,92	0,91	0,61	0,54	0,58
Verbal: Stroop --> P ParaulaPT --> PT	0,88	0,93	0,90	0,65	0,53	0,58
Verbal: Stroop --> C ColorPD --> PD	0,91	0,93	0,92	0,67	0,60	0,63
Verbal: Stroop --> C ColorPT --> PT	0,87	1,00	0,93		0,00	
Verbal: Stroop --> PC ParaulaColorPD --> PD	0,91	0,92	0,91	0,63	0,58	0,60
Verbal: Stroop --> PC ParaulaColorPT --> PT	0,91	0,92	0,91	0,56	0,54	0,55
Visual: Clave de números - Codificación (WAIS-III) --> Correctes --> Correctes	0,91	0,92	0,91	0,59	0,57	0,58
Dígitos: inversos (WAIS) --> Digits inversosWaisPD --> PD	0,97	0,87	0,92	0,75	0,93	0,83
Dígitos: inversos (WAIS) --> Digits inversosWaisPcPe --> Pe	0,92	0,91	0,92	0,71	0,75	0,73
TAPPING de McQuarrie --> Ma dominant --> Ma dominant	0,81	0,94	0,87	0,35	0,12	0,18
TAPPING de McQuarrie --> Ma dominantPT --> PT	0,80	0,90	0,85	0,51	0,33	0,40
TAPPING de McQuarrie --> Ma nodominant --> Ma nodominant	0,84	1,00	0,92		0,00	
TAPPING de McQuarrie --> Ma nodominantPT --> PT	0,83	1,00	0,91		0,00	
Orientación de líneas RBANS --> Orienta liniesPT --> PT	0,84	1,00	0,91	0,00	0,00	
Dígitos: directos (WAIS) --> Digits directesWaisPD --> PD	0,87	0,97	0,92	0,93	0,75	0,83
Dígitos: directos (WAIS) --> Digits directesWaisPcPe --> Pe	0,91	0,92	0,92	0,75	0,71	0,73

Table 21. Summary of different estimators for class one and two among all targets. Optimistic case.

Predicted column	Precision_1	Recall_1	F1_1	Precision_2	Recall_2	F1_2
Lista de palabras del RBANS_verbal --> Corba aprenentatgePD --> PD	0,65	0,66	0,66	0,74	0,74	0,74
Sexe12 --> Sexe12 --> Sexe12	0,55	0,32	0,40	0,54	0,76	0,63
Verbal: Stroop --> P ParaulaPD --> PD		0,00		0,83	1,00	0,91
Verbal: Stroop --> P ParaulaPT --> PT		0,00		0,85	1,00	0,92
Verbal: Stroop --> C ColorPD --> PD		0,00		0,83	1,00	0,91
Verbal: Stroop --> C ColorPT --> PT	0,58	0,37	0,45	0,84	0,92	0,88
Verbal: Stroop --> PC ParaulaColorPD --> PD	0,50	0,01	0,02	0,83	1,00	0,91
Verbal: Stroop --> PC ParaulaColorPT --> PT	0,47	0,15	0,22	0,82	0,96	0,89
Visual: Clave de números - Codificación (WAIS-III) --> Correctes --> Correctes		0,00		0,86	1,00	0,92
Dígitos: inversos (WAIS) --> Digits inversosWaisPD --> PD	0,56	0,17	0,26	0,89	0,98	0,93
Dígitos: inversos (WAIS) --> Digits inversosWaisPcPe --> Pe	0,69	0,65	0,66	0,92	0,93	0,92
TAPPING de McQuarrie --> Ma dominant --> Ma dominant		0,00		0,87	1,00	0,93
TAPPING de McQuarrie --> Ma dominantPT --> PT		0,00		0,91	1,00	0,95
TAPPING de McQuarrie --> Ma nodominant --> Ma nodominant		0,00		0,89	1,00	0,94
TAPPING de McQuarrie --> Ma nodominantPT --> PT		0,00		0,91	1,00	0,95
Orientación de líneas RBANS --> Orienta liniesPT --> PT	0,61	0,42	0,49	0,88	0,94	0,91
Dígitos: directos (WAIS) --> Digits directesWaisPD --> PD		0,00		0,95	1,00	0,97
Dígitos: directos (WAIS) --> Digits directesWaisPcPe --> Pe	0,65	0,69	0,66	0,93	0,92	0,92

From this results we can conclude several things. First of all, the difference between the estimators related to class one and class two was clear in the two settings. Although the separated analysis of each case could make us conclude that some features are well predicted, as in the case of (*TAPPING de McQuarrie, Ma dominantPT , PT*), for the pessimistic case with values of 0.91 for the precision, 1.00 for the recall and 0.95 for the F1 score this analysis was biased. If we look at the same estimators in the optimistic case we have 0.51 for the precision, 0.33 for the recall and 0.40 for the F1 score and this indicate us that the high scores achieved were due to the filling values strategy.

As we can see in the tables, the same thing happened with several features. But there were some of them that achieved good results in both cases, pessimistic and optimistic. They also achieved reasonably good results predicting the class one. This list of relations is summarized in the following table.

Table 22. Comparison between optimistic and pessimistic cases for the most reliable features.

	Optimistic			Pessimistic		
Predicted column	**Precision_2**	**Recall_2**	**F1_2**	**Precision_2**	**Recall_2**	**F1_2**
Lista de palabras del RBANS_verbal --> Corba aprenentatgePD --> PD	0,94	0,64	0,76	0,74	0,74	0,74
Sexe12 --> Sexe12 --> Sexe12	0,54	0,58	0,56	0,54	0,76	0,63
Dígitos: inversos (WAIS) --> Digits inversosWaisPD --> PD	0,75	0,93	0,83	0,89	0,98	0,93
Dígitos: inversos (WAIS) --> Digits inversosWaisPcPe --> Pe	0,71	0,75	0,73	0,92	0,93	0,92
Dígitos: directos (WAIS) --> Digits directesWaisPD --> PD	0,93	0,75	0,83	0,95	1,00	0,97
Dígitos: directos (WAIS) --> Digits directesWaisPcPe --> Pe	0,75	0,71	0,73	0,93	0,92	0,92

The listed features had reasonable scores for the estimators in both settings. We also included the feature Sexe although it didn't have good results because we found interesting that there wasn't any correlation. This illustrates how this set of test had no correlation with the sexe of the patients. For the case of (*Lista de palabras del RBANS_verbal, Corba aprenentatgePD, PD*) it was interesting to find that the algorithm performed better in the optimistic setting. This is due to the fact that it is one of the classes with better balance. As in the case of continuous analysis, we found correlation between the digits *directes* and *inversos* of the *WAIS* test.

Taking the optimistic case as the minimum level of performance it seems that in further steps of the investigation, if we collect more data allowing to balance the classes, we could achieve some good results with that strategy.

11. Conclusions

Along this work we have explored the different steps in a KDD process from a theoretical point of view and we have applied this knowledge to a real problem. The theoretical parts cover most of the basic topics in this kind of processes, although is not possible to cover all due to the huge variety of scenarios.

In the application to our problem, we have succeeded in the first steps giving an initial description of the data we had and applying this knowledge in the preprocessing part, solving inconsistencies selecting features and dealing with sparse data. We have also succeeded in showing how the KDD steps are not isolated, as a result from one of the steps can influence in the previous ones.

The main problem along all the project was the poor quality of the database in terms of consistency and sparsity. Both problems were solved following different strategies but the sparsity forced us to discard most of the information. In the data mining step, this sparsity and the number of features and instances we analyzed didn't allow us to achieve firm results. A deeper analysis will be done with the help of the doctors from Terrassa Hospital in further steps of the investigation. At least now we know which are the problems with the database and that has given us ideas on how to solve them.

The possibility of collecting more data would improve the results minimizing the sparsity of the database. An analysis based in a few classes would perform better than attempted to model the continuous case. A feasible approximation to the problem would be the analysis of normal and anomaly values in the tests. This could be interesting in the sense that, although we could not truly predict the exact value, we could know when performing a test is going to be helpful or not in the diagnosis. This approach has to be explored with the help of the doctors in order to correctly set the threshold that separates for each test a normal result from an extreme one.

From the point of view of the application to our case of analysis this work has to be understated as a static picture of the investigation that is currently in process. So the results presented are not the final ones, just a detailed description of the last stages achieved in the investigation.

12. Bibliographical references

Books

Han, J., Pei, J., & Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

John Lu, Z. Q. (2010). The elements of statistical learning: data mining, inference, and prediction. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, *173*(3), 693-694.

Marsland, S. (2011). *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC.

Mitchell, T. M. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, *45*(37), 870-877.

Websites

Torres, A. (2013, May 8). *Hashing Strings with Python*. Retrieved from Python Central https://www.pythoncentral.io/hashing-strings-with-python/

13. Annexes

Figure 1 – Missing values map atencion

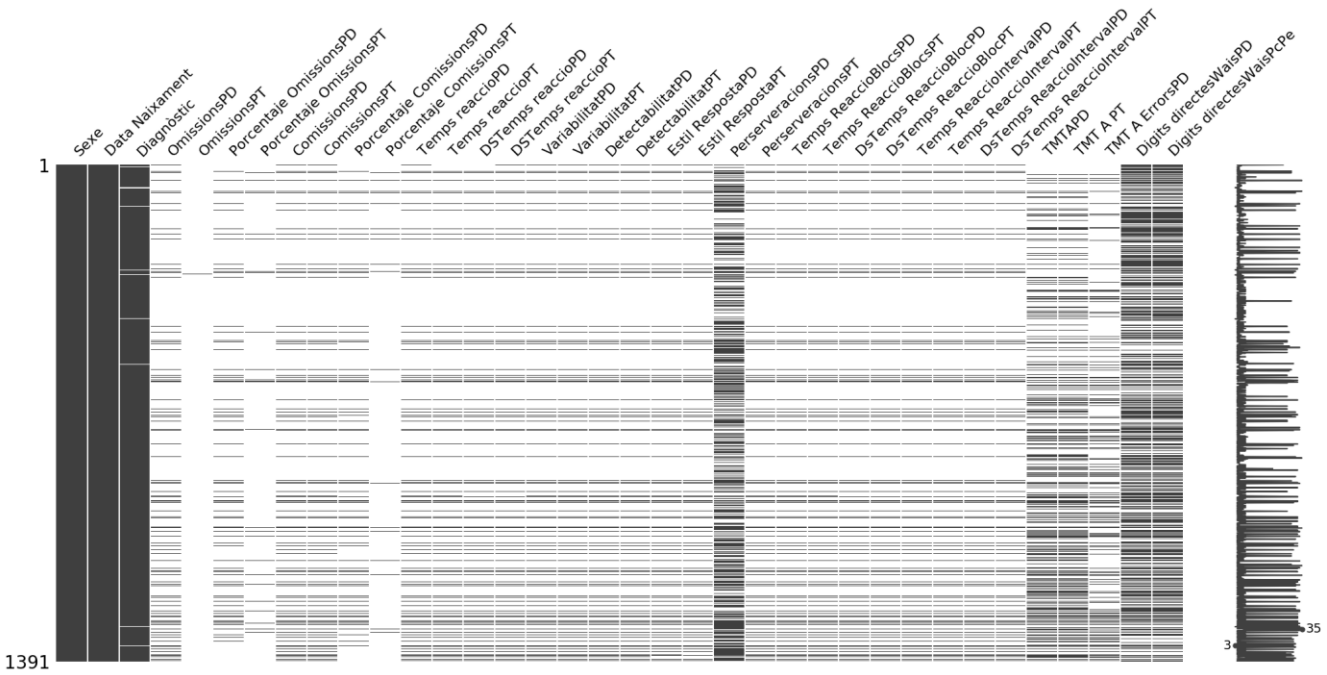


Figure 2 – Missing values map capacidad visuoconstructiva

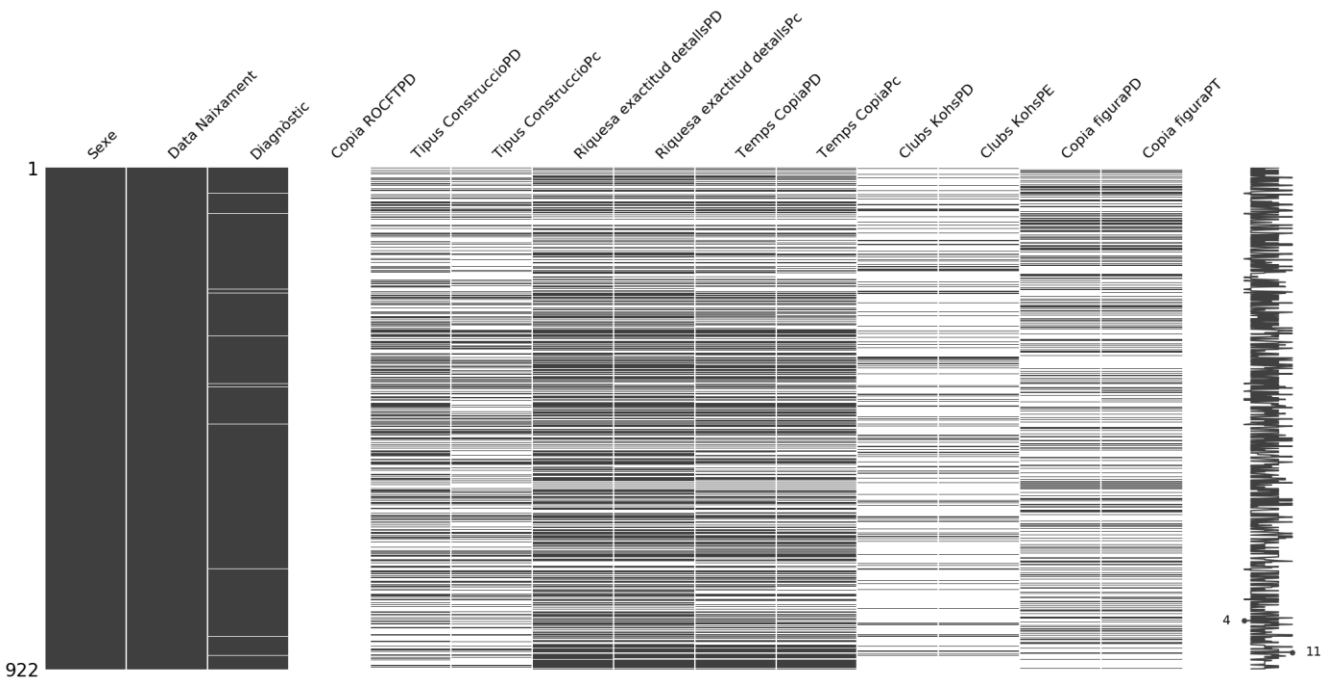


Figure 3 – Missing values map capacidad visuoespacial

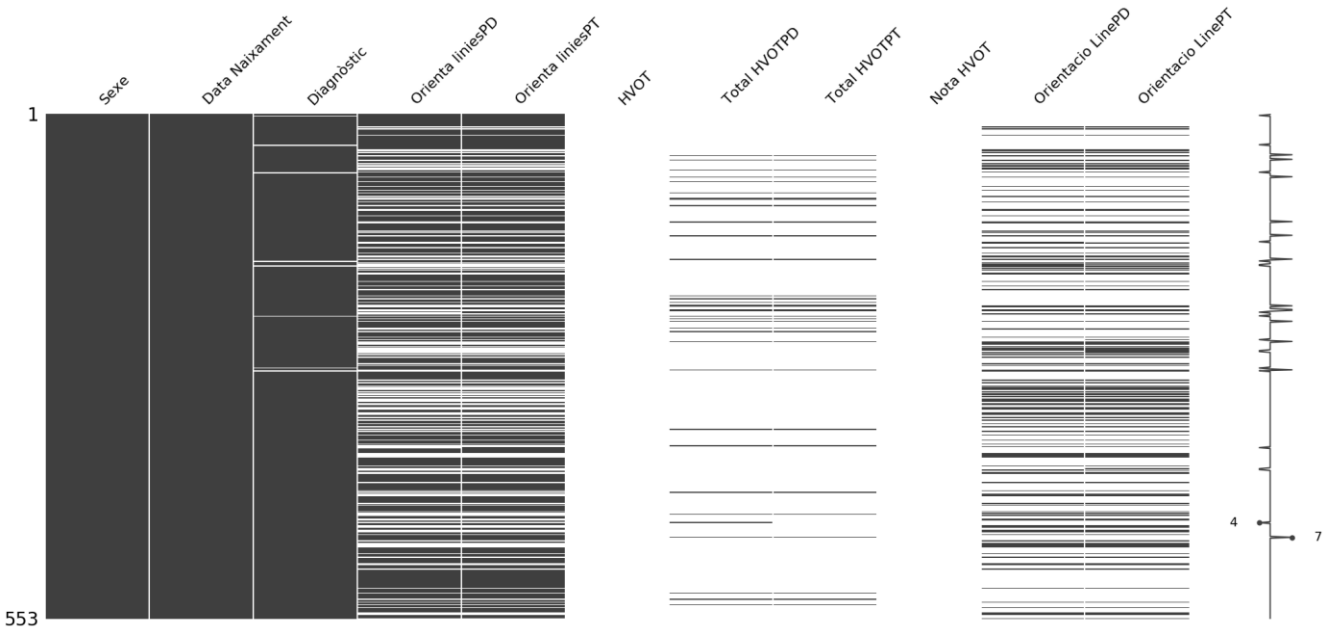


Figure 4 – Missing values map capacidad visuoperceptiva

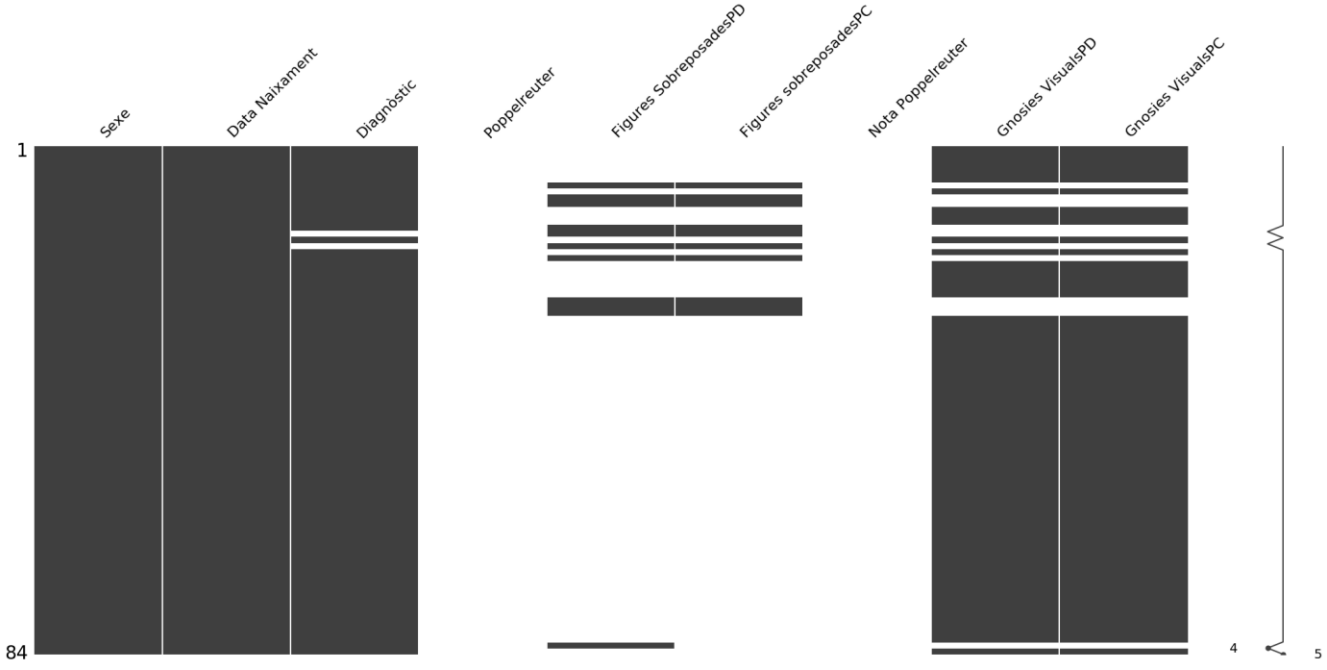


Figure 5 – Missing values map conducta emocional

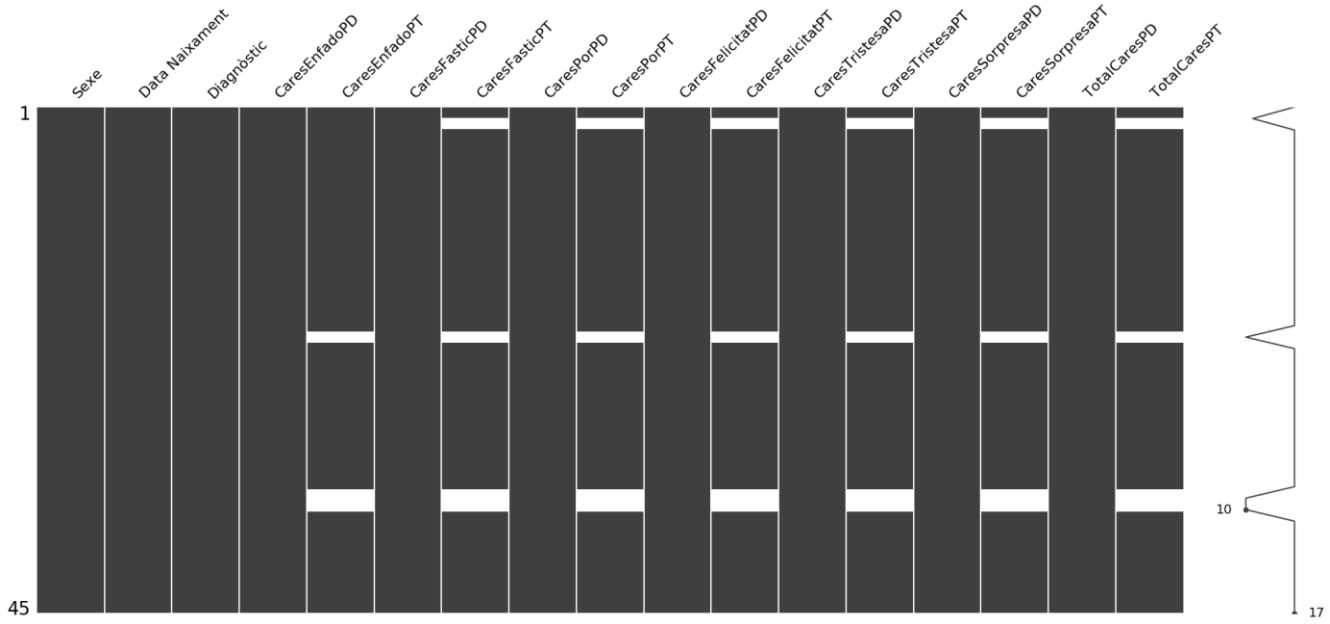


Figure 6 – Missing values map escalas clinicas

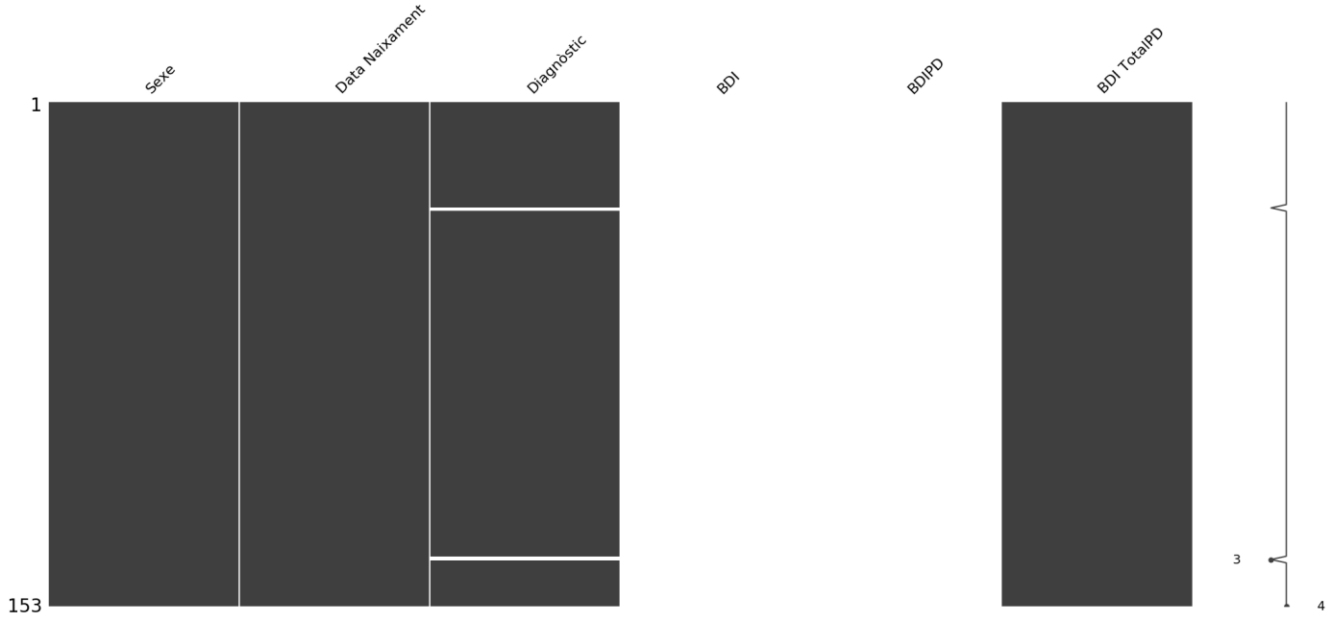


Figure 7 – Missing values map funcion ejecutiva

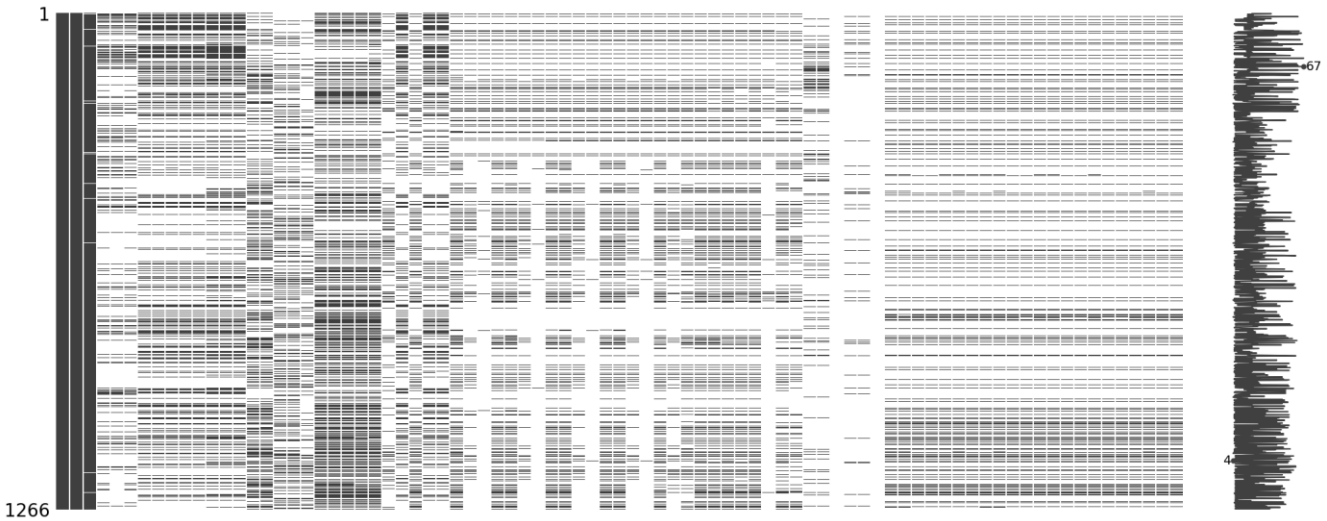


Figure 8 – Missing values map lenguaje

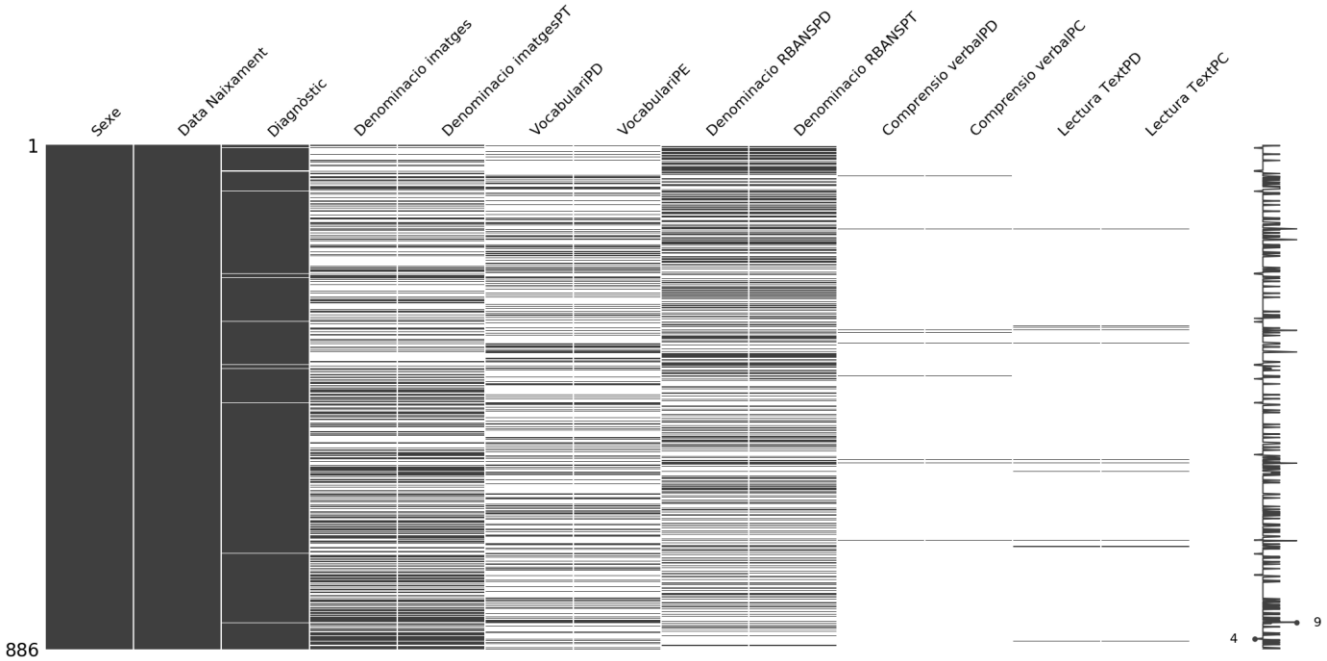


Figure 9 – Missing values map memoria

Figure 10 – Missing values map praxias

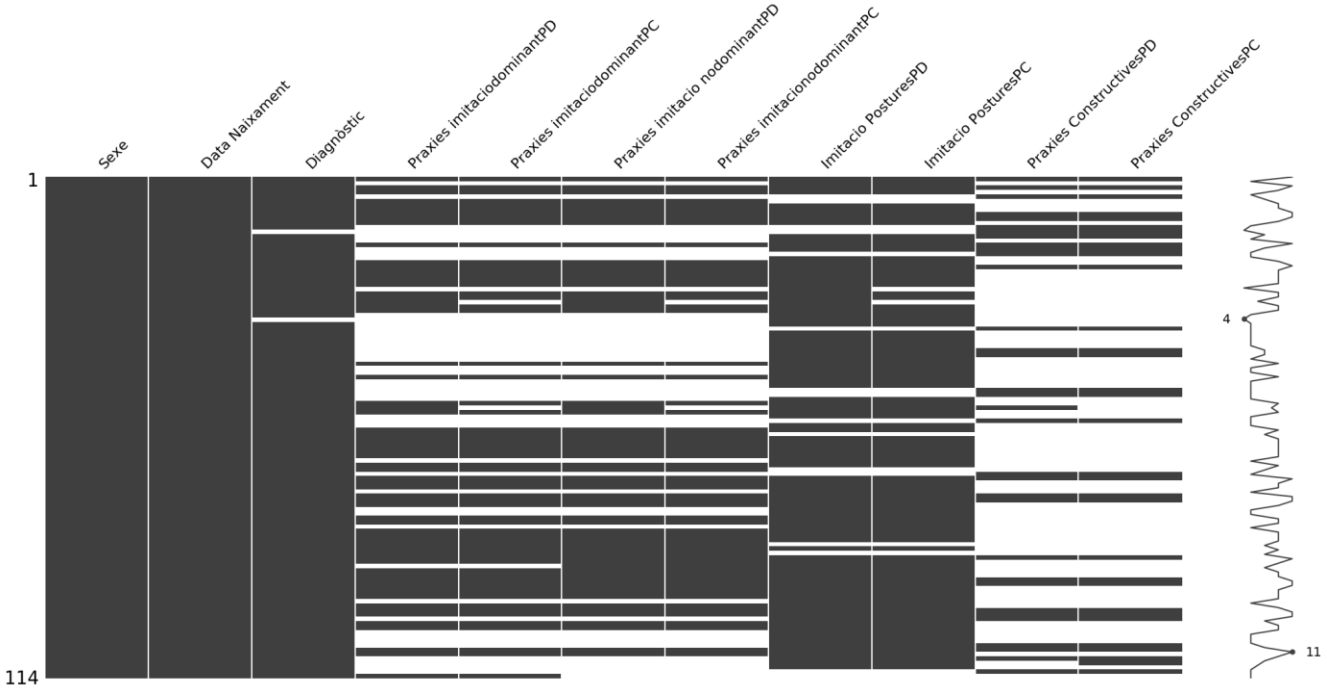


Figure 11 – Missing values map velocidad motora

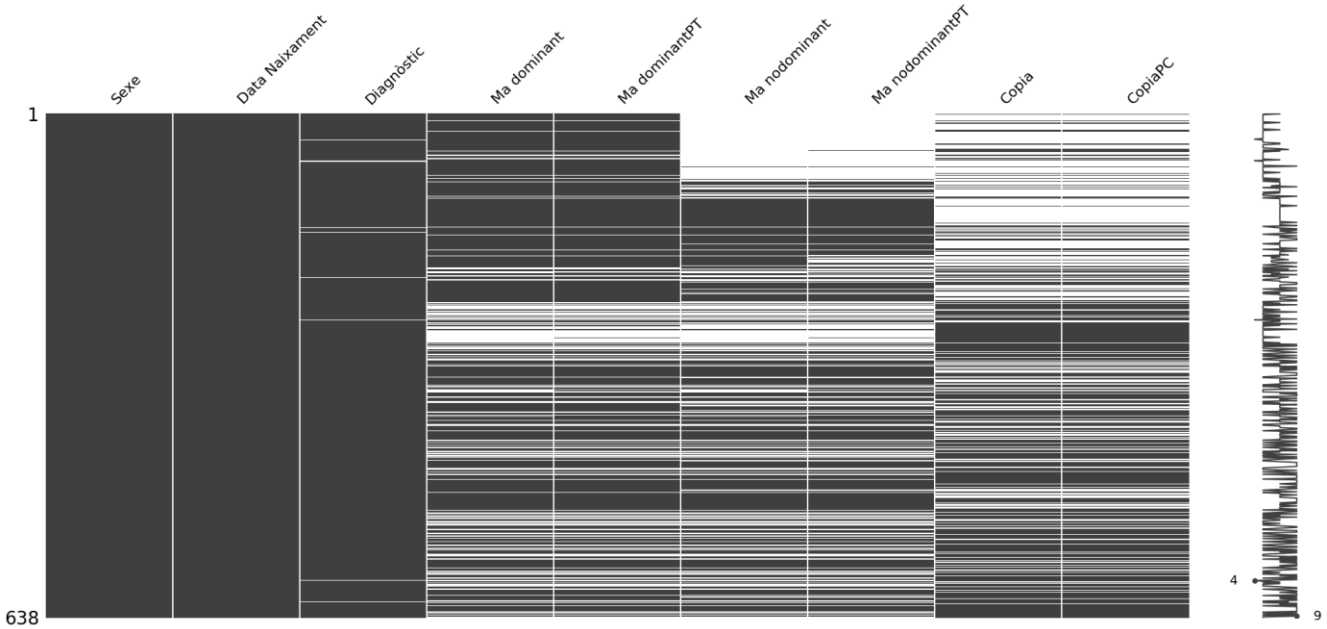


Figure 12 – Missing values map procesamiento informacion

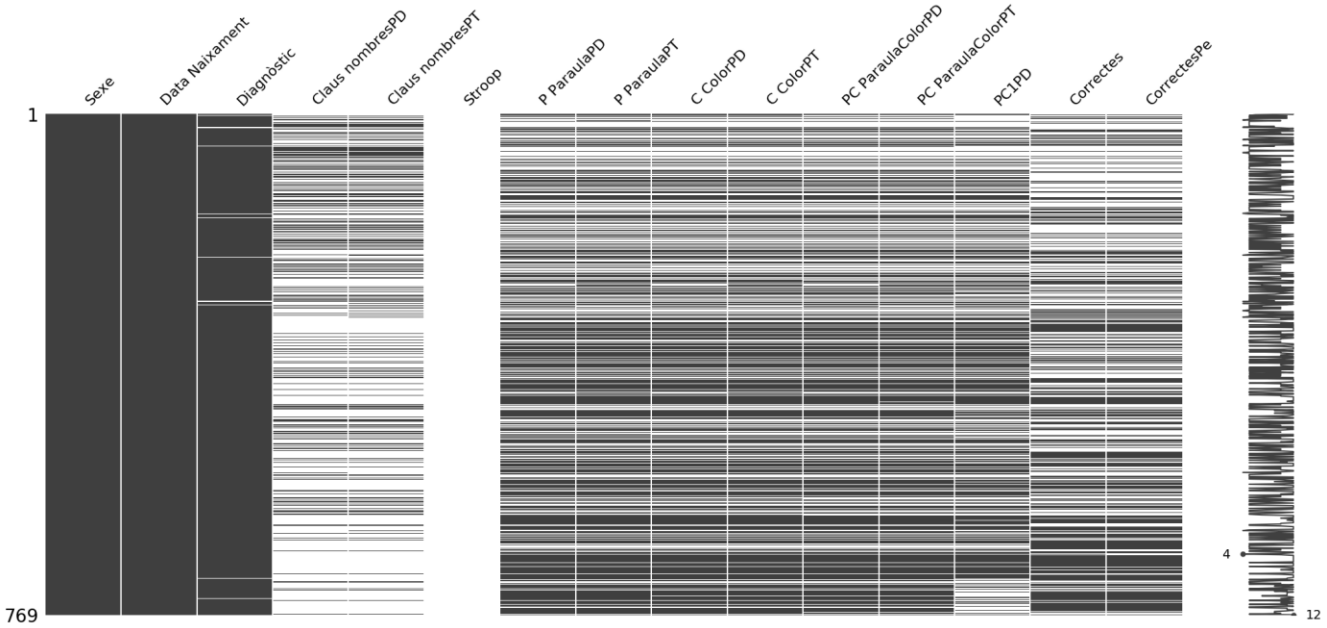


Figure 13 – Missing values map WN

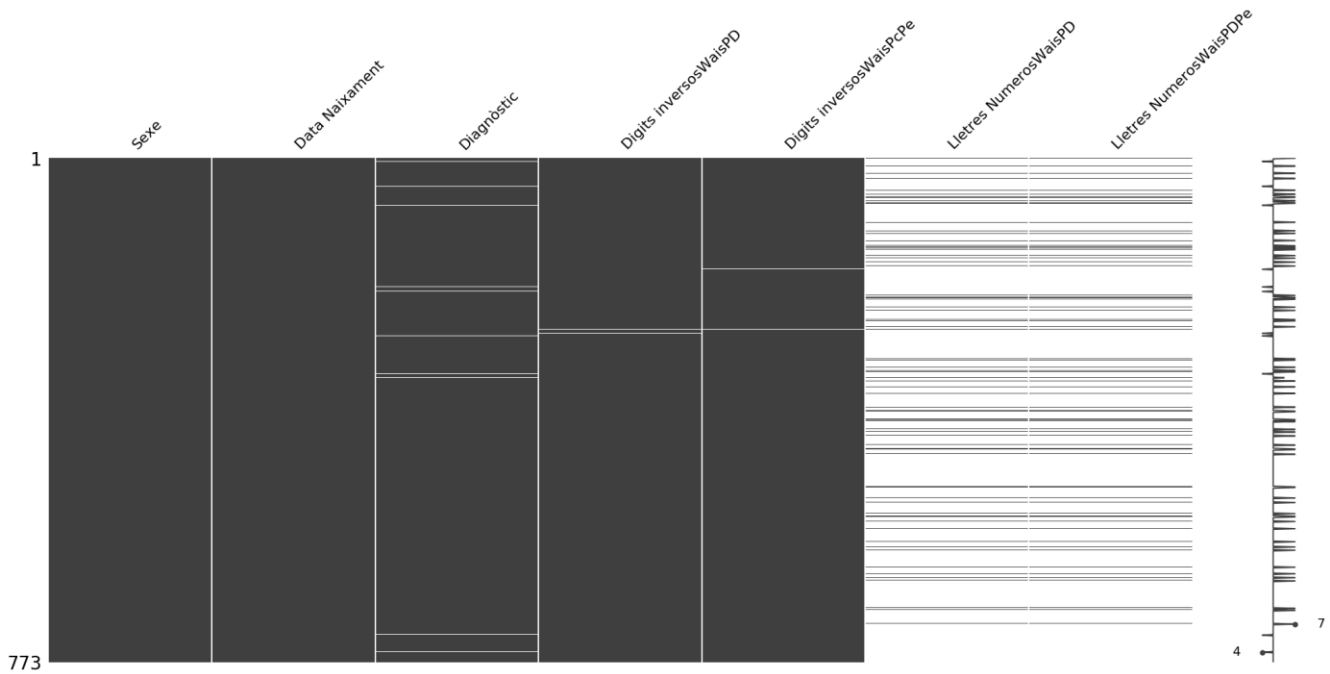


Figure 14 - Global description unified database

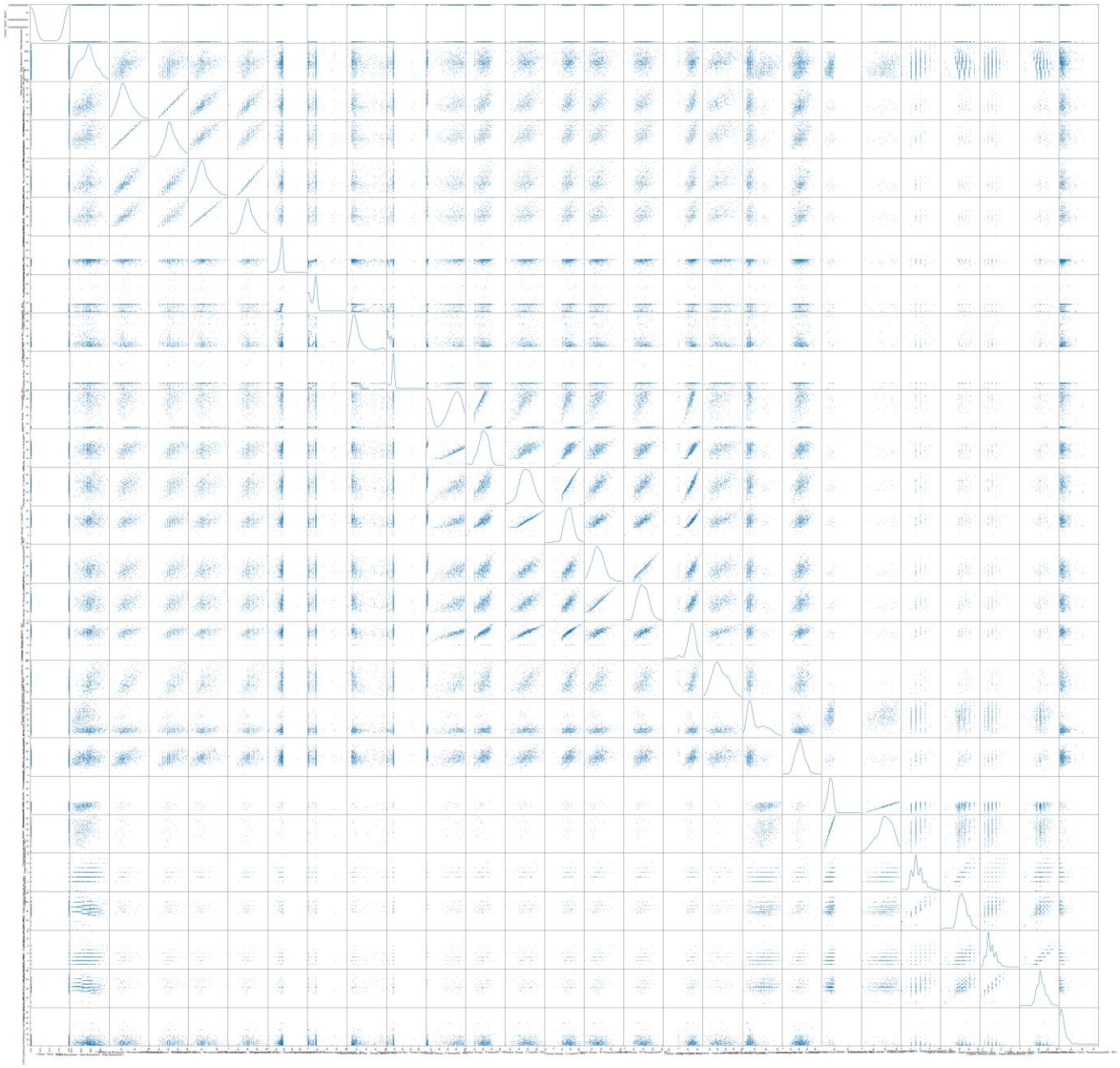


Figure 15 – Histogram discrete case data distribution. Binary optimistic.

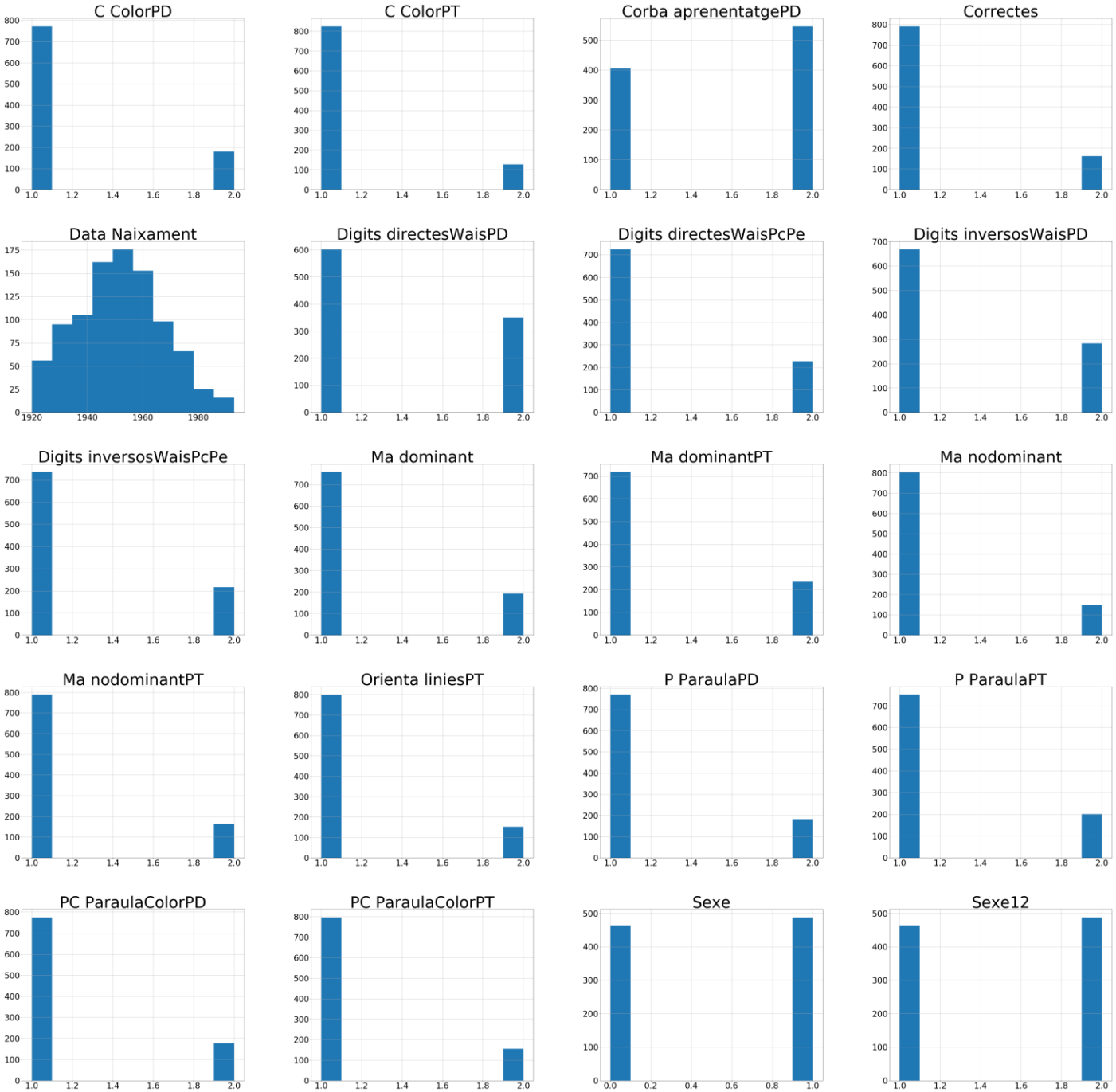
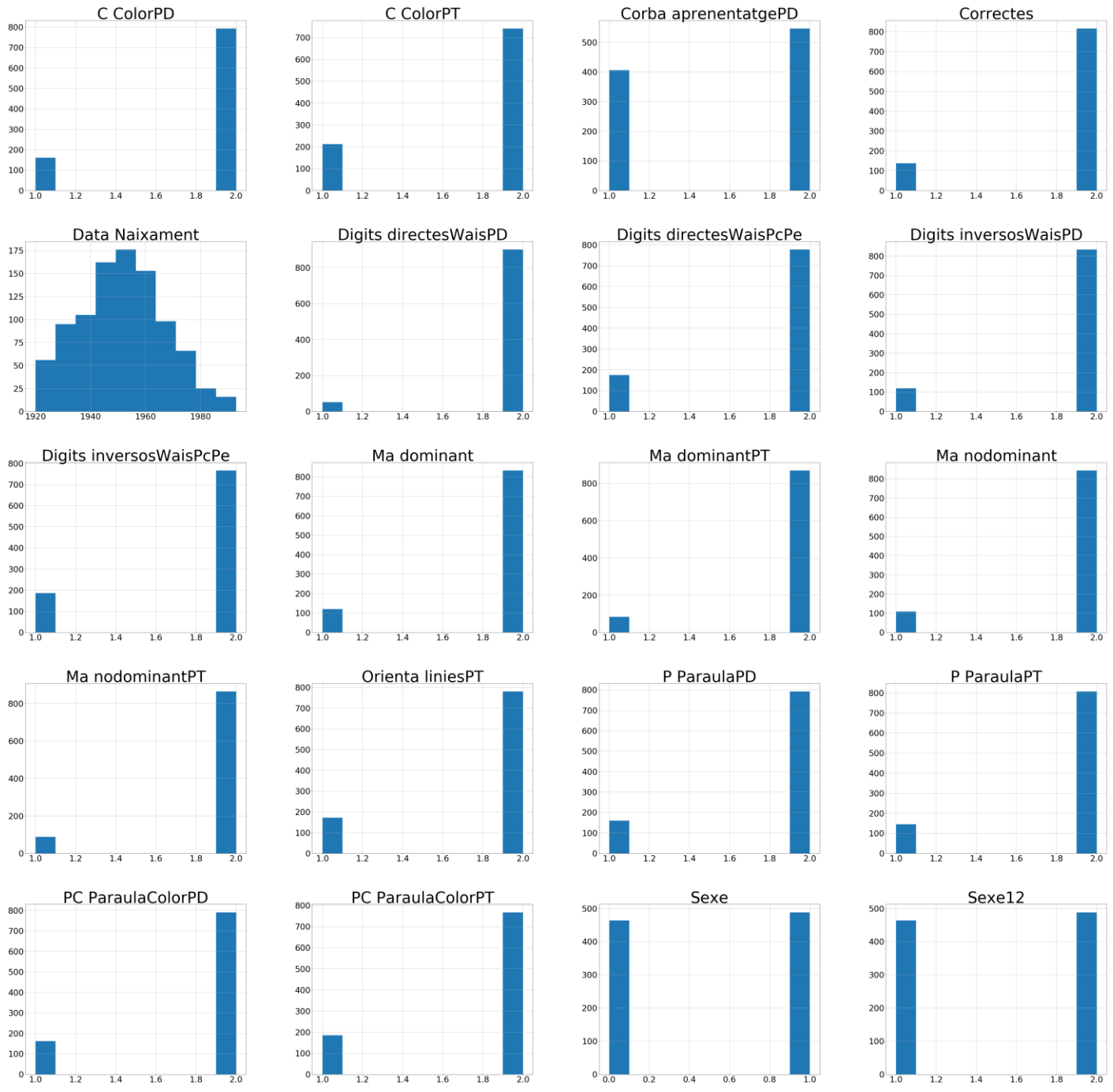


Figure 16 – Histogram discrete case data distribution. Binary pessimistic.



extraccion_de_datos_hospital

September 3, 2018

1 1. Jupyter Notebook: Script para la extracción anónima de datos

El script se encarga de leer cada fichero de entrada y eliminar todas las posibles columnas susceptibles de ser confidenciales y/o no necesarias para el estudio.

In [23]:
```
# Funcion para filtrado por multiples columnas y códigos
def filtro2(columna, codigo, data_frame):
    """
    Devuelve una data frame nueva con las filas de la data frame original que cumplían
    El filtro es de tipo OR, cualquier columna que cumpla una de las condiciones es añ
    Debe de haber una correspondencia entre la posición de la columna a filtrar en el
    posición del código a establecer para esa columna en el array de códigos.
    """
    new_data_frame = pd.DataFrame()
    new_data_frame = data_frame[~(data_frame[columna] == codigo)]
    new_data_frame = new_data_frame.drop_duplicates()
    return new_data_frame
```

In [29]:
```
import pandas as pd

def get_new_id(ID):
        ID = str(ID)
        return hash(ID)

nombres_ficheros_datos = ['Atenció & WM.xlsx',
                          'Bateries Generals.xlsx',
                          'Bateries Inteligencia.xlsx',
                          'C_Emocional.xlsx',
                          'Escales Clniques.xlsx',
                          'Escales Funcionals.xlsx',
                          'Executives.xlsx',
                          'Llenguatge.xlsx',
                          'Memória.xlsx',
                          'Screening Cognitiu.xlsx',
                          'Velocitat.xlsx',
                          'Visuoespacial-perceptiu&Pràxies.xlsx']
```

```
# Columnas a suprimir: Pacient, Codi postal, Població, Acte assitecial, Desc. acte assi
# Nom Metge Productor, Especialitat Productora, Desc. Especialitat Productora, Ubicació
# Metge Consumidor, Nom Metge Consumidor, Especialitat Consumidora, Desc. Especialitat
# Desc. Ubicació Consumidora, Data inici, Data final, Usuari creació, Hora  creació, Re
# Episodi, Descripció Episodi,

for nombre in nombres_ficheros_datos:
    df = pd.read_excel('Ficheros iniciales/' + nombre)

    df1 = filtro2('Usuari creació', 1, df)
    df1 = filtro2('Usuari creació', 2, df1)
    df1 = filtro2('Usuari creació', 3, df1)
    df1 = filtro2('Usuari creació', 4, df1)
    df1 = filtro2('Usuari creació', 5, df1)
    df1 = filtro2('Usuari creació', 6, df1)
    df1 = filtro2('Usuari creació', 7, df1)
    df1 = filtro2('Usuari creació', 8, df1)

    df1 = filtro2('Metge Productor', 1, df1)
    df1 = filtro2('Metge Productor', 2, df1)
    df1 = filtro2('Metge Productor', 3, df1)
    df1 = filtro2('Metge Productor', 4, df1)
    df1 = filtro2('Metge Productor', 5, df1)
    df1 = filtro2('Metge Productor', 6, df1)
    df1 = filtro2('Metge Productor', 7, df1)
    df1 = filtro2('Metge Productor', 8, df1)

    df1.insert(0,'id', df1['Història'])
    df1['id'] = df1['id'].apply(get_new_id)

    id_vs_historial = pd.concat([df1['id'], df1['Història']], axis=1)

    id_vs_historial.to_csv('id_vs_historia.csv')


    columns_to_delete = ['Història', 'Pacient', 'Codi postal', 'Població', 'Acte assite
     'Nom Metge Productor', 'Especialitat Productora', 'Desc. Especialitat Productora',
     'Metge Consumidor', 'Nom Metge Consumidor', 'Especialitat Consumidora', 'Desc. Esp
     'Desc. Ubicació Consumidora', 'Data inici', 'Data final', 'Nom Usuari creació', 'U
     'Episodi', 'Descripció Episodi']

    for i in columns_to_delete:
        del df1[i]

    df1['Data Naixament'] = df1['Data Naixament'].apply(lambda x: str(x))
    df1['Data Naixament'] = df1['Data Naixament'].apply(lambda x: x.split('-')[0])

    df1.to_excel('Ficheros generados/' + nombre)
```

```
            df1 = df1.set_index(['id','Data creació'])
```

```
In [1]: # Not allowed
```

```
In [26]: # Not allowed
```

Solving inconsistencies

September 3, 2018

1 Solving inconsistencies:

Here we solve inconsistencies we found in the data frames. Each row in each data set must to be indetified in a unambiguous form with the id and the creation date. Due to errors introducing the data in the excel files we have info of one row distributed in several rows. We want to merge all the info in one unique row.

In [1]:
```
import os
import numpy as np
import pandas as pd
```

In [2]:
```
# Function used to merge all the data contained in rows with the same index given rise t
# rows completed
def merge_repited_rows_into_one(df):
    df = df.reset_index()
    df['Data creació'] = df['Data creació'].apply(lambda x: str(x).split(' ')[0])
    df['combined_id'] = df['Data creació'] + df['id'].apply(lambda x: str(x))

    df = df.set_index('combined_id')

    combined_ids = set(df.index)

    for Id in combined_ids:
        if len(df.loc[Id].shape) != 1:
            row_to_be_completed = df.loc[Id].head(1).copy()
            rows_with_extra_data = df.loc[Id].tail(-1).copy()
            df = df.drop(Id)
            for i in range(len(rows_with_extra_data)):
                row_to_be_completed = row_to_be_completed.fillna(rows_with_extra_data.il
            df = df.append(row_to_be_completed)

    df = df.reset_index()
    df = df.set_index(['id', 'Data creació'])

    return df
```

In [3]:
```
"""
Limpieza preliminar de datos: Nos interesa eliminar posibles registros duplicados y erro
```

```
    Cada fichero representa una funcion psicológica y hay pruebas compartidas entre estas fu
    todos los ficheros. Por ello una de las primeras tareas será incluir en cada fichero toa
    conjunto de ficheros que sean propias de esta función. Además incluimos algunos gráficos

    Columnas con problemas por intervalo abierto en los percentiles:
    MEMORIA -> ROCFT o Figura Complexa de Rey
    VELOCITAT -> Clau Numérica WAIS-III


    EXECUTIVES -> !!!!!I+5 Temps latenciaPD (No esta generada bien, no usarla)!!!!!
    """
    # Comenzamos eliminando los duplicados presentes
    nombres_ficheros_datos = ['Atenció & WM.xlsx',
                              'Bateries Generals.xlsx',
                              'Bateries Inteligencia.xlsx',
                              'C_Emocional.xlsx',
                              'Escales Clniques.xlsx',
                              'Escales Funcionals.xlsx',
                              'Executives.xlsx',
                              'Llenguatge.xlsx',
                              'Memória.xlsx',
                              'Screening Cognitiu.xlsx',
                              'Velocitat.xlsx',
                              'Visuoespacial-perceptiu&Pràxies.xlsx']

    data_frames_dict = dict()
    for i in nombres_ficheros_datos:
        df = pd.read_excel('../Initial files/' + i)
        df = df.drop_duplicates()
        df = df.set_index(['id', 'Data creació'])
        df = merge_repited_rows_into_one(df)
        data_frames_dict[i] = df

    # Eliminamos la columna I+5 Temps latenciaPD presente en Executives.xlsx
    del data_frames_dict['Executives.xlsx']['I+5 Temps latenciaPD']
```

In [4]:
```
    # Guardamos los ficheros corregidos en una carpeta nueva.
    if not os.path.exists('../Corrected Files'):
        os.makedirs('../Corrected Files')

    for nombre in nombres_ficheros_datos:
        df = data_frames_dict[nombre].copy()
        df = df.reset_index()
        df = df.set_index('id')
        df.to_excel('../Corrected Files/' + nombre)
```

In [5]:
```
    # Checking consistency
    df1 = pd.read_excel('../Initial files/' + 'Atenció & WM.xlsx')
```

```
df1 = df1.drop_duplicates()
df1 = df1.set_index(['id', 'Data creació'])
df2 = data_frames_dict['Atenció & WM.xlsx'].copy()
print(len(df1.columns), len(df2.columns)) # Correcto
```

```
168 169
```

In [6]:
```
pd.set_option('display.height', 1000)
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

In [7]:
```
df1.loc[5265739105367749632].dropna(axis=1, how='all')
```

Out[7]:

Data creació	Sexe	Data Naixament	Diagnòstic	Desc. Diagnòstic	TMTAPD	TMT A PT
2008-10-31	H	1940	1601DCO	DETERIORAMENT COGNITIU	44.0	53.0
2008-10-31	H	1940	1601DCO	DETERIORAMENT COGNITIU	NaN	NaN
2008-10-31	H	1940	1601DCO	DETERIORAMENT COGNITIU	NaN	NaN

In [8]:
```
df2.loc[5265739105367749632].dropna(axis=1, how='all')
```

Out[8]:

Data creació	combined_id	Sexe	Data Naixament	Diagnòstic	Desc.
2008-10-31	2008-10-315265739105367749632	H	1940	1601DCO	DETERIORAME

Summarizing I

September 3, 2018

1 Summarizing the available Data:

We want to make some basic tables with the total amount of patients, the amount patient in each function and the percentaje of records over the total for the tests with more information.

In [1]:
```
"""
Summarizing the available Data: We want to make some basic tables with the total amount
patient in each function and the percentaje of records over the total for the tests with
"""
import pandas as pd
import numpy as np
import collections

pd.set_option('display.float_format', lambda x: '%.2f' % x)


# Creating an array with each excel converted into a data_frame
nombres_ficheros_datos = ['Atenció & WM.xlsx',
                          'Bateries Generals.xlsx',
                          'Bateries Inteligencia.xlsx',
                          'C_Emocional.xlsx',
                          'Escales Clniques.xlsx',
                          'Escales Funcionals.xlsx',
                          'Executives.xlsx',
                          'Llenguatge.xlsx',
                          'Memória.xlsx',
                          'Screening Cognitiu.xlsx',
                          'Velocitat.xlsx',
                          'Visuoespacial-perceptiu&Pràxies.xlsx']

data_frames = []
for i in nombres_ficheros_datos:
    df = pd.read_excel('../Corrected Files/' + i)
    df = df.drop_duplicates()
    df = df.set_index(['id', 'Data creació'])
    data_frames.append(df)
```

```
# We want for the total amount of data and for each file, the total number of registers
# for the main tests passed
# For each file (we need to split the different tests into groups in order to have a mor
# the available data)

# This function is used to sort the data of an array by de values of the second componen
def getKey(item):
    return item[1]

writer = pd.ExcelWriter('../Summaries/Summary I.xlsx')

ids = []
row_to_write = -6
for df, name in zip(data_frames, nombres_ficheros_datos):
    df = df.dropna(axis='columns', how='all')
    name = name[:-5]
    number_of_tests = []

    index = df.index.get_level_values(0)
    for id in index.drop_duplicates():
        ids.append(id)
        number_of_tests.append(len(df.loc[id]))

    number_of_records = len(index)
    df_nulls = df.isnull()
    columns = df.columns[7:]
    number_of_records_each_group = []

    # We search for the change in the test group by identifying the change in three firs
    last_prefix = ''
    for column in columns:
        actual_prefix = column[:3]
        if (actual_prefix != last_prefix):
            number_of_records_each_group.append([column, df_nulls[column][df_nulls[colum
        last_prefix = column[:3]

    number_of_records_each_group = sorted(number_of_records_each_group, key=getKey, reve
    number_of_records_each_group = [[x[0] ,float((100 * x[1] / number_of_records))] for
    number_of_records_each_group.insert(0,['Numero total pacientes', number_of_records])

    row_to_write = row_to_write + 10

    summary = pd.DataFrame([name], columns=['Function'])
    summary.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)

    summary = pd.DataFrame(number_of_records_each_group[:7], columns=['Test','Percentaje
    summary.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, startcol=1, ind
```

```
    counter=dict(collections.Counter(number_of_tests))
    summary = pd.DataFrame(list(counter.items()), columns=['Number of tests','Number of

    summary.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, startcol=4, ind

# The total number for all the files
number_of_patients = len(set(ids))

d = {'Total number of patients': [number_of_patients]}
total = pd.DataFrame(data=d)

total.to_excel(writer, sheet_name='Sheet1', index=False)

# Close the Pandas Excel writer and output the Excel file
writer.save()
```

Generate New Structure

September 3, 2018

1 Generate new data structure:

This script generates a new estructure for the data frame columns, using pandas function MultiIndex.from_tuples(). The different tuples that configurate the structure are stored in a dict serialized in a json file.

Using it: Just make sure that this script, the excel files and the json file are in the same folder and then execute the script.

```
In [1]: import pandas as pd
        import numpy as np
        import os
        import json
```

```
In [2]: def convert_sex_into_boolean(df_sexe):
            if df_sexe == 'H':
                df_sexe = 1
            elif df_sexe == 'D':
                df_sexe = 0
            return df_sexe


        nombres_ficheros_datos = ['Atenció & WM.xlsx',
                                  'Bateries Generals.xlsx' ,
                                  'Bateries Inteligencia.xlsx',
                                  'C_Emocional.xlsx',
                                  'Escales Clniques.xlsx',
                                  'Escales Funcionals.xlsx',
                                  'Executives.xlsx',
                                  'Llenguatge.xlsx',
                                  'Memória.xlsx',
                                  'Screening Cognitiu.xlsx',
                                  'Velocitat.xlsx',
                                  'Visuoespacial-perceptiu&Pràxies.xlsx']

        data_frames = []
        data_frames_dict = dict()
        for i in nombres_ficheros_datos:
            df = pd.read_excel('../Corrected Files/' + i)
```

```
        df.drop_duplicates
        df['combined_id'] = df['Data creació'] + df['id'].apply(lambda x: str(x))
        df = df.set_index(['combined_id'])
        df['Sexe'] = df['Sexe'].apply(convert_sex_into_boolean)

        old_columns = df.columns.tolist()

        new_columns = [old_columns.pop(0), old_columns.pop(2), old_columns.pop(2), old_colum
                       old_columns.pop(2), old_columns.pop(0), old_columns.pop(0)]

        for j in old_columns:
            new_columns.append(j)

        df = df[new_columns]
        data_frames.append(df)
        data_frames_dict[i] = df
```

In [3]:
```
# The different tuples that configurate the structure are stored
# in a dict serialized in a json file.
with open('atencion.json', 'r') as fp:
    atencion_dict = json.load(fp)

with open('WM.json', 'r') as fp:
    WM_dict = json.load(fp)

with open('velocidad_procesamiento_informacion.json', 'r') as fp:
    velocidad_procesamiento_informacion_dict = json.load(fp)

with open('velocidad_motora.json', 'r') as fp:
    velocidad_motora_dict = json.load(fp)

with open('capacidad_visuoconstructiva.json', 'r') as fp:
    capacidad_visuoconstructiva_dict = json.load(fp)

with open('capacidad_visuoespacial.json', 'r') as fp:
    capacidad_visuoespacial_dict = json.load(fp)

with open('capacidad_visuoperceptiva.json', 'r') as fp:
    capacidad_visuoperceptiva_dict = json.load(fp)

with open('praxias.json', 'r') as fp:
    praxias_dict = json.load(fp)

with open('lenguaje.json', 'r') as fp:
    lenguaje_dict = json.load(fp)

with open('memoria.json', 'r') as fp:
    memoria_dict = json.load(fp)
```

```
with open('funcion_ejecutiva.json', 'r') as fp:
    funcion_ejecutiva_dict = json.load(fp)

with open('conducta_emocional.json', 'r') as fp:
    conducta_emocional_dict = json.load(fp)

with open('escalas_clinicas.json', 'r') as fp:
    escalas_clinicas_dict = json.load(fp)


info_diccionarios = [('atencion', atencion_dict),
                     ('WM', WM_dict),
                     ('velocidad_procesamiento_informacion', velocidad_procesamiento_in
                     ('velocidad_motora', velocidad_motora_dict),
                     ('capacidad_visuoconstructiva', capacidad_visuoconstructiva_dict),
                     ('capacidad_visuoespacial', capacidad_visuoespacial_dict),
                     ('capacidad_visuoperceptiva', capacidad_visuoperceptiva_dict),
                     ('praxias', praxias_dict),
                     ('lenguaje', lenguaje_dict),
                     ('memoria', memoria_dict),
                     ('funcion_ejecutiva', funcion_ejecutiva_dict),
                     ('conducta_emocional', conducta_emocional_dict),
                     ('escalas_clinicas', escalas_clinicas_dict)]
```

In [4]:
```
data_frames_new_format_dict = {'atencion': 0, 'WM': 0, 'velocidad_procesamiento_informac
                               'capacidad_visuoconstructiva': 0, 'capacidad_visuoespacia
                               'praxias': 0, 'lenguaje': 0, 'memoria': 0, 'funcion_ejecu
                               'escalas_clinicas': 0}

for i, diccionario in info_diccionarios:
    diccionario['Sexe'] = ['Sexe']
    diccionario['Data Naixament'] = ['Data Naixament']
    diccionario['Diagnòstic'] = ['Diagnòstic']

# Ahora mismo tenemos la lista de todos los diccionarios con todas las columnas que debe
# diccionario más grande donde los alamacenaremos, la informacion de los datos esta ahor
# Cambiamos el planteamiento: lo vamos a hacer de la manera más lógica que sería:
#   1. Para cada prueba de cada key de cada diccionario buscamos en todos los ficheros y
#   2. Una vez hallamos acabado con ese diccionario se concatenan todas las df creadas y
#   3. Para la correcta creación de las tuplas que harán de indice de columnas debemos c
#   que no haya repeticiones.

# We realise that we need a data frame with all differet ids from patient in order to pe
list_of_ids = []
for nombre in nombres_ficheros_datos:
    df = data_frames_dict[nombre].copy()
    actual_index = df.index.get_values()
```

```
            for i in actual_index:
                list_of_ids.append(i)

        list_of_ids = set(list_of_ids)

        for nombre, diccionario in info_diccionarios:
            list_of_tuples_for_index = []
            df_nueva = pd.DataFrame(index=list_of_ids)
            for key in diccionario.keys():
                for prueba in diccionario[key]:
                    flag = False
                    if prueba not in df_nueva.columns:
                        for nombre_viejo in nombres_ficheros_datos:
                            df_vieja = data_frames_dict[nombre_viejo].copy()
                            if prueba in list(df_vieja.columns):
                                flag = True
                                try:
                                    df_nueva = df_nueva.join(df_vieja[prueba])
                                except ValueError:
                                    df_nueva[prueba] = df_nueva[prueba].fillna(df_vieja[prueba])
                                    pass
                        if prueba[-3:] == 'CPT' and flag:
                            list_of_tuples_for_index.append((key, prueba, prueba))
                        elif (prueba[-2:] == 'PT' or prueba[-2:] == 'PD' or prueba[-2:] == 'Pt'
                              prueba[-2:] == 'pt' or prueba[-2:] == 'pd' or prueba[-2:] == 'pT'
                              prueba[-2:] == 'Pe' or prueba[-2:] == 'Pc') and flag:
                            list_of_tuples_for_index.append((key, prueba, prueba[-2:]))
                        elif flag:
                            list_of_tuples_for_index.append((key, prueba, prueba))
            df_nueva_copy = df_nueva.copy()
            del df_nueva_copy['Sexe'], df_nueva_copy['Data Naixament'], df_nueva_copy['Diagnòsti
            df_nueva = pd.concat([df_nueva['Sexe'],df_nueva['Data Naixament'],df_nueva['Diagnòst
            df_nueva = df_nueva.loc[df_nueva.iloc[:,3:].dropna(axis=0, how='all').index]

            data_frames_new_format_dict[nombre] = [df_nueva, list_of_tuples_for_index]
```

In [6]:
```
        # Guardamos la info sin estructurar y la estructura en un JSON
        if not os.path.exists('../New Structure Files'):
            os.makedirs('../New Structure Files')

        tuples_for_index_dict = dict()
        for key in data_frames_new_format_dict:
            data_frames_new_format_dict[key][0].to_excel('../New Structure Files/' + key + '.xls

            list_of_tuples_for_index = data_frames_new_format_dict[key][1].copy()
            list_of_tuples_for_index_correction = []

            for tupla in list_of_tuples_for_index:
```

```
                if tupla not in (('Sexe', 'Sexe', 'Sexe'),
                                 ('Data Naixament', 'Data Naixament', 'Data Naixament'),
                                 ('Diagnòstic', 'Diagnòstic', 'Diagnòstic')):
                    list_of_tuples_for_index_correction.append(tupla)

            tuples_for_index_dict[key] = [('Sexe', 'Sexe', 'Sexe'),
                                          ('Data Naixament', 'Data Naixament', 'Data Naixament')
                                          ('Diagnòstic', 'Diagnòstic', 'Diagnòstic')] + list_of_

        with open('tuples_for_index_dict.json', 'w') as fp:
            json.dump(tuples_for_index_dict, fp)
```

In [8]:
```
df = data_frames_new_format_dict['atencion'][0].copy()

index = df.index
columns = pd.MultiIndex.from_tuples( tuples_for_index_dict['atencion'])

data = [list(df.iloc[i,:].get_values()) for i in range(len(df))]

df_new = pd.DataFrame(data, columns=columns, index=index)

df_new.to_excel('../New structure xample for the document.xlsx')
```

Summarizing II

September 3, 2018

1 Summarizing the available data (new structure):

In [1]:
```
# Import libraries
import pandas as pd
import collections
import load_data2
import os
```

In [2]:
```
data_frames_dict, tuples_new_structure_dict = load_data2.read_and_format('../New Structu
```

In [3]:
```
# Summarizing available data:
# Incluir los datos sobre el numero de veces que el paciente ha pasado la prueba y ya es
# Podría ser intresante calcular la proporcion de H/M y la media de edad!!!
if not os.path.exists('../Summaries'):
    os.makedirs('../Summaries')

writer = pd.ExcelWriter('../Summaries/Summary II.xlsx')
row_to_write = 4
ids = []
for key in data_frames_dict.keys():
    df = data_frames_dict[key].copy()
    columns = tuples_new_structure_dict[key]

    number_of_patients = []
    number_of_patients_per_group = []
    number_of_tests = []

    copy = ''
    # Almacenamos ids para ver el número total de pacientes en todos los ficheros
    for id in df.index.get_level_values(0).drop_duplicates():
            ids.append(id)
            number_of_tests.append(len(df.loc[id]))

    for lvl0, lvl1, lvl2 in columns: # Probar zip esto de los lvels así es una guarrada!
        if lvl0 == 'Sexe':
            total_number_of_patients = len(df[lvl0].dropna(axis=0, how='all'))
            number_of_patients_per_group.append(['Numero total de pacientes',total_numbe
```

```
                continue
            if lvl0 in ['Data Naixament','Diagnòstic']:
                continue
            if copy != lvl0:
                percentage_of_patients = len(df[lvl0].dropna(axis=0, how='all'))/total_numbe
                number_of_patients_per_group.append([lvl0,percentage_of_patients*100])
            copy = lvl0

        summary = pd.DataFrame([key], columns=['Function'])
        summary.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)

        first_element = number_of_patients_per_group[0]
        number_of_patients_per_group_sorted = sorted(number_of_patients_per_group[1:],key=la
        number_of_patients_per_group_sorted.insert(0,number_of_patients_per_group[0])
        number_of_patients_per_group = number_of_patients_per_group_sorted
        number_of_patients_per_group_sorted.clear

        summary = pd.DataFrame(number_of_patients_per_group, columns=['Test','Percentaje of
        summary.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, startcol=1, ind

        # Aqui contamos el numero de veces que se pasa cada test
        counter=dict(collections.Counter(number_of_tests))
        summary = pd.DataFrame(list(counter.items()), columns=['Number of tests','Number of
        summary.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, startcol=4, ind

        row_to_write = row_to_write + len(number_of_patients_per_group) + 3

number_of_patients_total = len(set(ids))

d = {'Total number of patients': [number_of_patients_total]}
total = pd.DataFrame(data=d)

total.to_excel(writer, sheet_name='Sheet1', index=False)

# Close the Pandas Excel writer and output the Excel file
writer.save()
```

Filtering and Visualizing

September 3, 2018

1 Filtering and visualizing:

Due to the sparsity of the data base we want to filter out this columns with less than X percentaje. We also wants to visualize the number of records that results from the filtering proces in each file. We also filter out those patients that could be considered outliers in the age feature. We follow the rule of considering an outlier thoose ages that are beyond 1.5 times the intercuartile range.

```
In [1]: # Import libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import load_data
        import os
        import json
```

```
In [2]: data_frames_dict, tuples_new_structure_dict = load_data.read_and_format('../New Structur
```

1.0.1 Filtrado basado en el porcentaje de datos de la columna:

Especificamos el porcentaje minimo de pacientes que han pasado la prueba para que la columna se pueda considerar. Los modelos predictivos que vamos a usar mas adelante no permiten incoporar valores nulos o NaN, por ello debemos borrar estos valores antes de proceder con el algoritmo. Algunas columnas que tienen un porcentaje de valores muy bajo harían que nios quedasemos sin datos apenas. Es por esto que se hace este filtrado previo. Además de para reducir el numero de features que sometemos a estudio.

```
In [3]: def get_info_about_number_patients_by_column(data_frames_dict):
            columnas_que_resultan_de_filtrar_dict = dict()

            for ambito in data_frames_dict.keys():
                df = data_frames_dict[ambito].copy()
                n_pacientes = []
                for lvl0, lvl1, lvl2 in df.columns:
                    n_pacientes.append(len(df[lvl0][lvl1][lvl2].dropna()))
                info = np.zeros(150)
                info[0] =  len(df.columns.get_level_values(2))
```

```
            for i in range(len(n_pacientes)):
                info[i + 1] = n_pacientes[i]
            columnas_que_resultan_de_filtrar_dict[ambito] = info

        columnas_que_resultan_de_filtrar_df = pd.DataFrame(columnas_que_resultan_de_filtrar_
        del columnas_que_resultan_de_filtrar_dict

        columnas_que_resultan_de_filtrar_df = columnas_que_resultan_de_filtrar_df.T
        columnas_que_resultan_de_filtrar_df = columnas_que_resultan_de_filtrar_df.loc[:, (co
        return columnas_que_resultan_de_filtrar_df
```

In [4]:
```
# Vamos a hacerlo filtrando directamente:
data_frames_dict_reduced = dict()
patients_percentaje = 50

for ambito in data_frames_dict.keys():
    df = data_frames_dict[ambito].copy()
    number_of_records = len(df)
    df_nulls = df.isnull()
    for lvl0, lvl1, lvl2 in tuples_new_structure_dict[ambito]:
        if (df_nulls[lvl0][lvl1][lvl2][df_nulls[lvl0][lvl1][lvl2] == False].count() * 10
            df = df.drop((lvl0, lvl1, lvl2), axis=1)
    data_frames_dict_reduced[ambito] = df
```

In [5]:
```
columnas_que_resultan_de_filtrar_df = get_info_about_number_patients_by_column(data_fram
```

In [6]:
```
pd.set_option('display.height', 1000)
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

In [7]:
```
columnas_que_resultan_de_filtrar_df
```

Out[7]:

	0	1	2	3	4	5	6
WM	5.0	773.0	773.0	763.0	771.0	771.0	0.0
atencion	6.0	1391.0	1391.0	1374.0	772.0	773.0	766.0
capacidad_visuoconstructiva	7.0	922.0	922.0	911.0	545.0	538.0	483.0
capacidad_visuoespacial	5.0	553.0	553.0	545.0	376.0	378.0	0.0
capacidad_visuoperceptiva	5.0	84.0	84.0	82.0	73.0	73.0	0.0
conducta_emocional	17.0	45.0	45.0	45.0	45.0	42.0	45.0
escalas_clinicas	4.0	153.0	153.0	151.0	153.0	0.0	0.0
funcion_ejecutiva	3.0	1266.0	1266.0	1247.0	0.0	0.0	0.0
lenguaje	3.0	886.0	886.0	871.0	0.0	0.0	0.0
memoria	5.0	1372.0	1372.0	1355.0	766.0	959.0	0.0
praxias	9.0	114.0	114.0	112.0	66.0	64.0	66.0
velocidad_motora	9.0	638.0	638.0	630.0	359.0	359.0	478.0
velocidad_procesamiento_informacion	11.0	769.0	769.0	757.0	471.0	474.0	468.0

In [8]:
```
sum = 0
for n_col in columnas_que_resultan_de_filtrar_df[0]:
```

```
            sum = sum + (n_col - 3)

        sum
```

```
Out[8]: 50.0
```

```
In [9]: # del data_frames_dict_reduced['praxias'], data_frames_dict_reduced['escalas_clinicas']
        # del data_frames_dict_reduced['capacidad_visuoperceptiva'], data_frames_dict_reduced['c
```

```
In [21]: data_frames_dict = data_frames_dict_reduced
         del data_frames_dict_reduced
```

1.0.2 Filtro basado en percentiles sobre el campo edad:

Realizamos este filtro previo especialmente por temas de anonimato pues sería relativamente fácil identificar a estos pacientes por ser muy pocos y jovenes. Además alguno de ellos no llega a la mayoría de edad. Por otro lado intentar encontrar patrones relazionados con el desarrollo de Alzeheimer el cual se da en edades más avanzadas estos datos podrían actuar como ruido.

```
In [11]: # Eliminamos todos los ids que son outliers en el campo edad para todos los ficheros po
         outliers_edad_dict = dict()
         data_frames_dict_age_filtered = dict()

         for ambito in data_frames_dict.keys():
             df = data_frames_dict[ambito].copy()

             q75, q25 = np.percentile(df['Data Naixament']['Data Naixament']['Data Naixament'],
             iqr = q75 - q25

             min = q25 - (iqr*1.5)
             max = q75 + (iqr*1.5)

             outliers_edad_dict[ambito] = [min, max]

             df = df.reset_index()
             ids_max = df.loc[df['Data Naixament']['Data Naixament']['Data Naixament'] > max, 'i
             ids_min = df.loc[df['Data Naixament']['Data Naixament']['Data Naixament'] < min, 'i
             df = df.set_index('id')
             df = df.drop(ids_max)
             df = df.drop(ids_min)
             df = df.reset_index()
             df = df.set_index(['id', 'date'])
             data_frames_dict_age_filtered[ambito] = df
```

```
In [16]: # Visualización de los outliers eliminados
         %matplotlib notebook

         df = data_frames_dict['memoria'].copy()
```

```
        i = 'Data Naixament'

        q75, q25 = np.percentile(df[i][i][i], [75 ,25])
        iqr = q75 - q25

        min = q25 - (iqr*1.5)
        max = q75 + (iqr*1.5)

        plt.figure(figsize=(10,8))
        plt.subplot(211)
        plt.xlim(df[i][i][i].min(), df[i][i][i].max()*1.01)
        plt.axvline(x=min)
        plt.axvline(x=max)

        ax = df[i][i][i].plot(kind='kde')

        plt.subplot(212)
        plt.xlim(df[i][i][i].min(), df[i][i][i].max()*1.01)
        sns.boxplot(x=df[i][i][i])
        plt.axvline(x=min)
        plt.axvline(x=max)

        plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
In [13]: # Visualización de los outliers eliminados
         %matplotlib notebook

         df = data_frames_dict_age_filtered['memoria'].copy()

         i = 'Data Naixament'

         q75, q25 = np.percentile(df[i][i][i], [75 ,25])
         iqr = q75 - q25

         min = q25 - (iqr*1.5)
         max = q75 + (iqr*1.5)

         plt.figure(figsize=(10,8))
         plt.subplot(211)
         plt.xlim(df[i][i][i].min(), df[i][i][i].max()*1.01)
         plt.axvline(x=min)
         plt.axvline(x=max)
```

```
        ax = df[i][i][i].plot(kind='kde')

        plt.subplot(212)
        plt.xlim(df[i][i][i].min(), df[i][i][i].max()*1.01)
        sns.boxplot(x=df[i][i][i])
        plt.axvline(x=min)
        plt.axvline(x=max)

        plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

In [14]:
```
# Tomamos como base de datos aquella con los outliers del campo de edad ya filtrados
data_frames_dict = data_frames_dict_age_filtered
del data_frames_dict_age_filtered
```

In [15]:
```
# Para guardar correctamente ver como suprimir las columnas
filtered_tuples_for_index_dict = dict()
for key in data_frames_dict.keys():
    tuples_for_index = []
    for lvl0, lvl1, lvl2 in data_frames_dict[key].columns:
        tuples_for_index.append((lvl0, lvl1, lvl2))
    filtered_tuples_for_index_dict[key] = tuples_for_index
```

In [16]:
```
# Guardamos la info sin estructurar y la estructura en un JSON
if not os.path.exists('../Filtered Files/'):
    os.makedirs('../Filtered Files/')

for key in data_frames_dict:
    df = data_frames_dict[key].copy()
    df = df.reset_index()

    df.index = df['date'] + df['id'].apply(lambda x: str(x))

    del df['date'], df['id']

    df.columns = range(df.columns.shape[0])
    df.to_excel('../Filtered Files/' + key + '.xlsx')

with open('filtered_tuples_for_index_dict.json', 'w') as fp:
    json.dump(filtered_tuples_for_index_dict, fp)
```

In [17]:
```
# Esto era para ver la integridad de los datos en el proceso, borrar de aqui pa abajo
df1 = pd.read_excel('../New Structure Files/' + 'memoria.xlsx')
```

```
df1 = df1.reset_index()

df1['date'] = df1['index'].apply(lambda x: x[:10])
df1['id'] = df1['index'].apply(lambda x: x[10:])

del df1['index']
df1 = df1.set_index(['id', 'date'])

df2 = data_frames_dict['memoria'].copy()
```

Unify Databases

September 3, 2018

1 Unificate Database:

As we see that the number of features that results from the filtering process is very low (when filtering thoose columns with less than 40% of records over the total of patients) we are going to unificate all the diferent databases into one in order to have more features. We will start the mining process with that database. Is very important to merge the columns in an appropiate form in order to respect the tests passed by each patient.

In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import load_data
import os
import json
```

In [2]:
```
data_frames_dict, tuples_new_structure_dict = load_data.read_and_format('../Filtered Fil
```

In [3]:
```
merged_columns_dict = dict()
data_frames_merged_columns_dict = dict()
for key in data_frames_dict.keys():
    df = data_frames_dict[key].copy()
    merged_columns = []
    for lvl0, lvl1, lvl2 in df.columns:
        merged_columns.append(lvl0 + "@" + lvl1 + "@" + lvl2)
    merged_columns_dict[key] = merged_columns
    index = df.index
    data = [list(df.iloc[i,:].get_values()) for i in range(len(df))]

    df = pd.DataFrame(data, columns=merged_columns, index=index)
    data_frames_merged_columns_dict[key] = df
```

In [4]:
```
data_frames_dict = data_frames_merged_columns_dict
del data_frames_merged_columns_dict
```

In [5]:
```
list_of_ids = []
for key in data_frames_dict.keys():
```

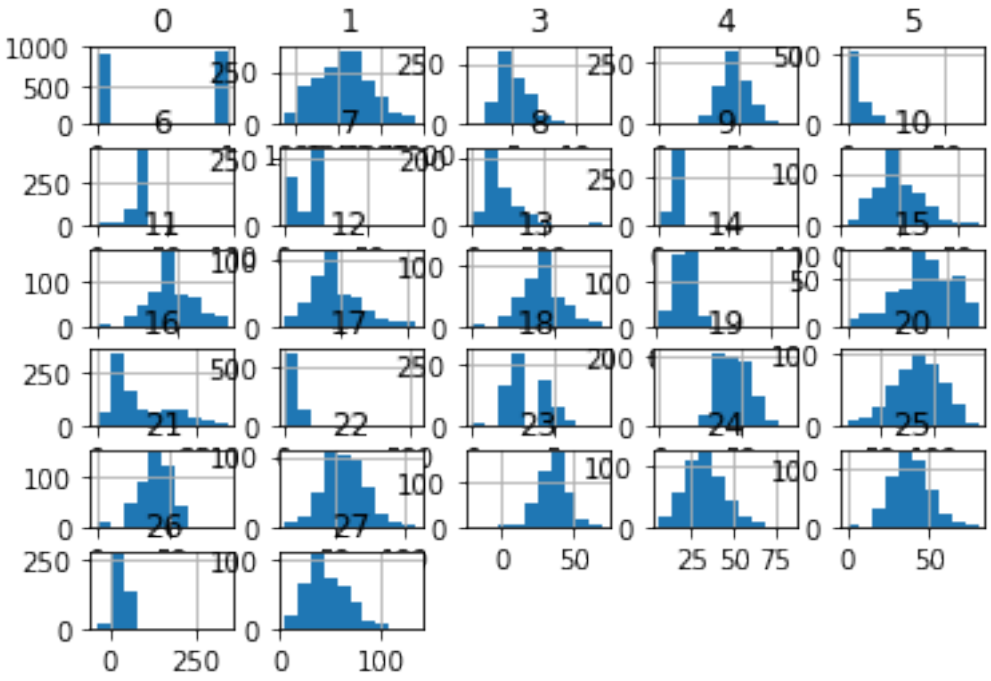
```
            df = data_frames_dict[key].copy()
            actual_index = df.index.get_values()
            for i in actual_index:
                list_of_ids.append(i)

        list_of_ids = set(list_of_ids)
        df_total = pd.DataFrame(index=list_of_ids)

        for key in data_frames_dict.keys():
            df = data_frames_dict[key].copy()
            for col in df.columns:
                try:
                    df_total = df_total.join(df[col])
                except ValueError:
                    df_total[col] = df_total[col].fillna(df[col])
                    pass
```

In [6]:
```
        # Vamos a hacerlo filtrando directamente:
        data_frames_dict_reduced = dict()
        patients_percentaje = 20

        df_nulls = df_total.isnull()
        number_of_records = len(df_total)
        df_new = df_total.copy()
        for column in df_total.columns:
            if (df_nulls[column][df_nulls[column] == False].count() * 100) / number_of_records <
                df_new.drop((column), axis=1, inplace=True)

        len(df_new.columns)
```

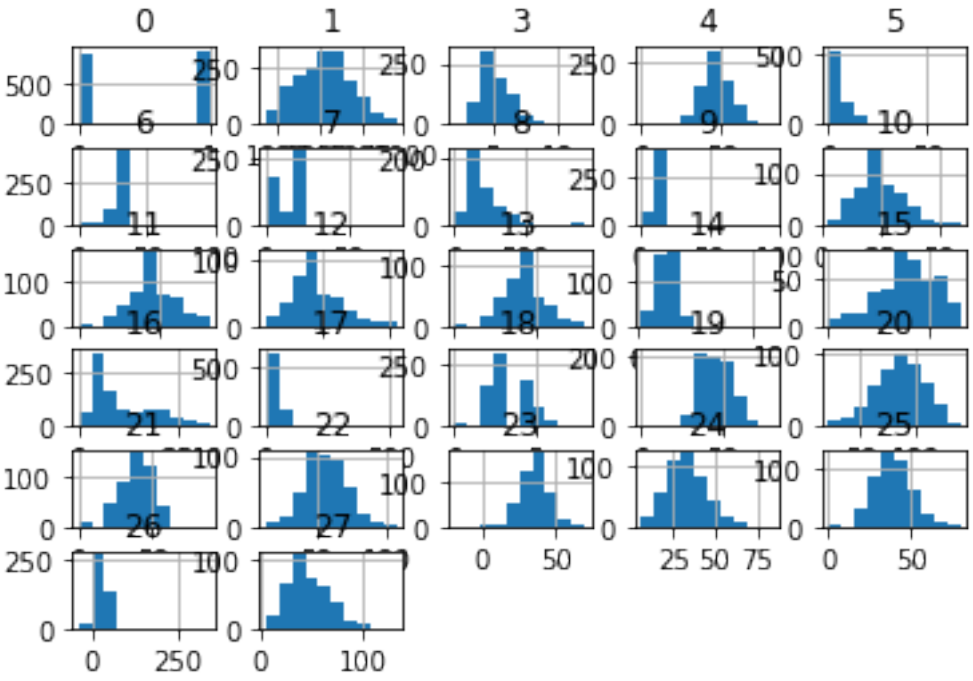
Out[6]: 28

In [7]:
```
        df_total = df_new
        del df_new
```

In [8]:
```
        df = df_total.copy()
        df.columns = [i for i in range(len(df.columns))]
        df.hist()
        plt.show()
```




```
In [25]: df = df_total.iloc[:,3:].copy()
         df = df.dropna(how='all', axis=0)
         index = df.index
         df_total = df_total.loc[index]

In [10]: del df

In [11]: df = df_total.copy()
         df.columns = [i for i in range(len(df.columns))]
         df.hist()
         plt.show()
```

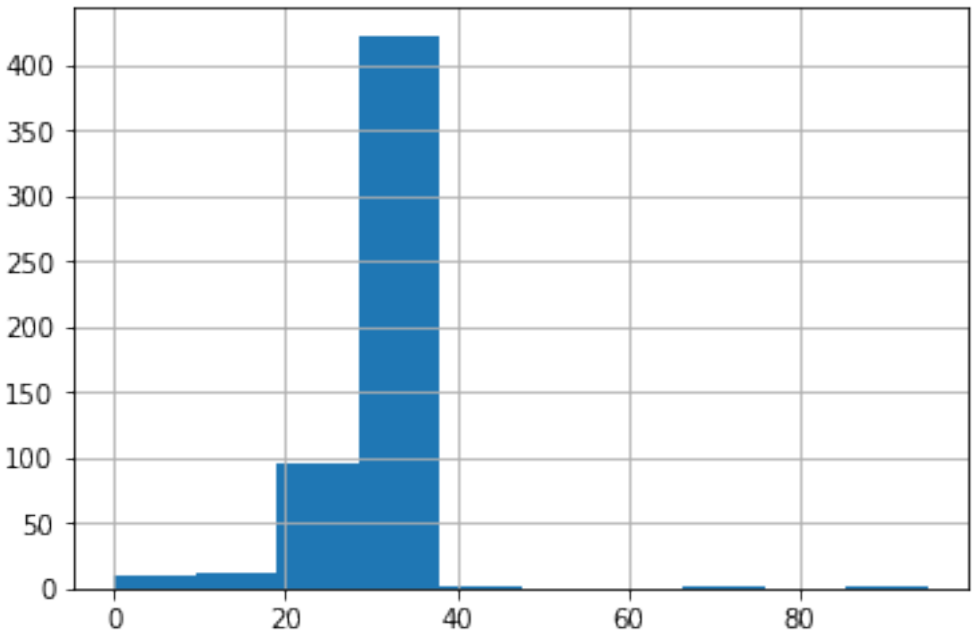



```
In [12]: for column in df_total.columns:
             print(column)
```

```
Sexe@Sexe@Sexe
Data Naixament@Data Naixament@Data Naixament
Diagnòstic@Diagnòstic@Diagnòstic
Dígitos: directos (WAIS)@Digits directesWaisPD@PD
Dígitos: directos (WAIS)@Digits directesWaisPcPe@Pe
CPT (Continuos Performance Test)@PerserveracionsPD@PD
Figura Compleja de Rey@Riquesa exactitud detallsPD@PD
Figura Compleja de Rey@Riquesa exactitud detallsPc@Pc
Figura Compleja de Rey@Temps CopiaPD@PD
Figura Compleja de Rey@Temps CopiaPc@Pc
TAPPING de McQuarrie@Ma dominant@Ma dominant
TAPPING de McQuarrie@Ma dominantPT@PT
TAPPING de McQuarrie@Ma nodominant@Ma nodominant
TAPPING de McQuarrie@Ma nodominantPT@PT
Orientación de líneas RBANS@Orienta liniesPD@PD
Orientación de líneas RBANS@Orienta liniesPT@PT
Lista de palabras del RBANS_verbal@Corba aprenentatgePD@PD
Animales_semantica@SemanticaPT@PT
Dígitos: inversos (WAIS)@Digits inversosWaisPD@PD
Dígitos: inversos (WAIS)@Digits inversosWaisPcPe@Pe
Verbal: Stroop@P ParaulaPD@PD
Verbal: Stroop@P ParaulaPT@PT
```

```
Verbal: Stroop@C ColorPD@PD
Verbal: Stroop@C ColorPT@PT
Verbal: Stroop@PC ParaulaColorPD@PD
Verbal: Stroop@PC ParaulaColorPT@PT
Verbal: Stroop@PC1PD@PD
Visual: Clave de números - Codificación (WAIS-III)@Correctes@Correctes
```

```
In [14]: df_total['Figura Compleja de Rey@Riquesa exactitud detallsPD@PD'].hist()
         plt.show()
```



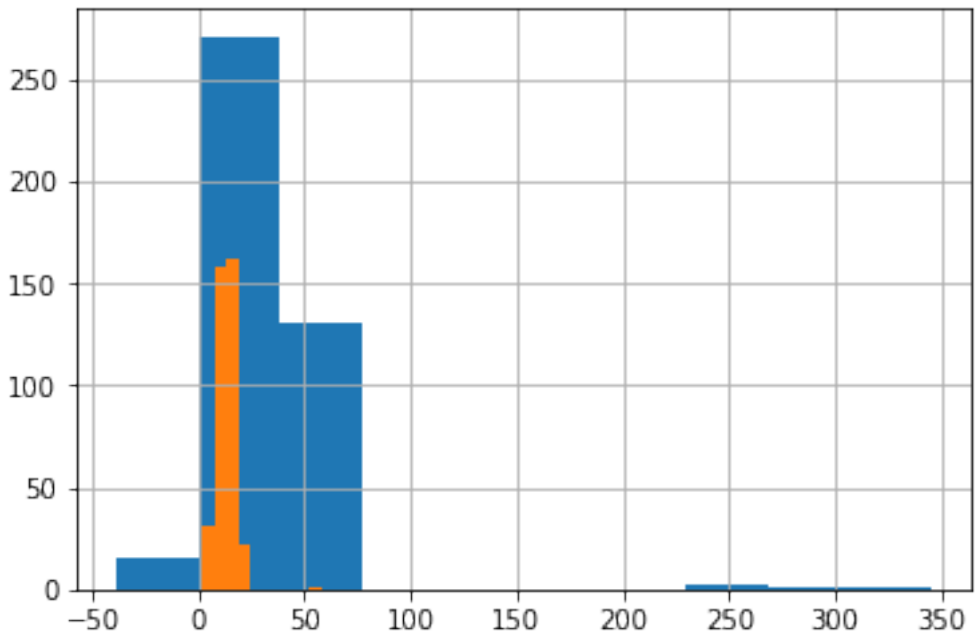

```
In [15]: delete_columns = ['CPT (Continuos Performance Test)@PerserveracionsPD@PD',
                          'Figura Compleja de Rey@Riquesa exactitud detallsPD@PD',
                          'Figura Compleja de Rey@Riquesa exactitud detallsPc@Pc',
                          'Figura Compleja de Rey@Temps CopiaPD@PD',
                          'Figura Compleja de Rey@Temps CopiaPc@Pc']

         for column in delete_columns:
             del df_total[column]
```

```
In [16]: for column in df_total.columns:
             print(column)
```

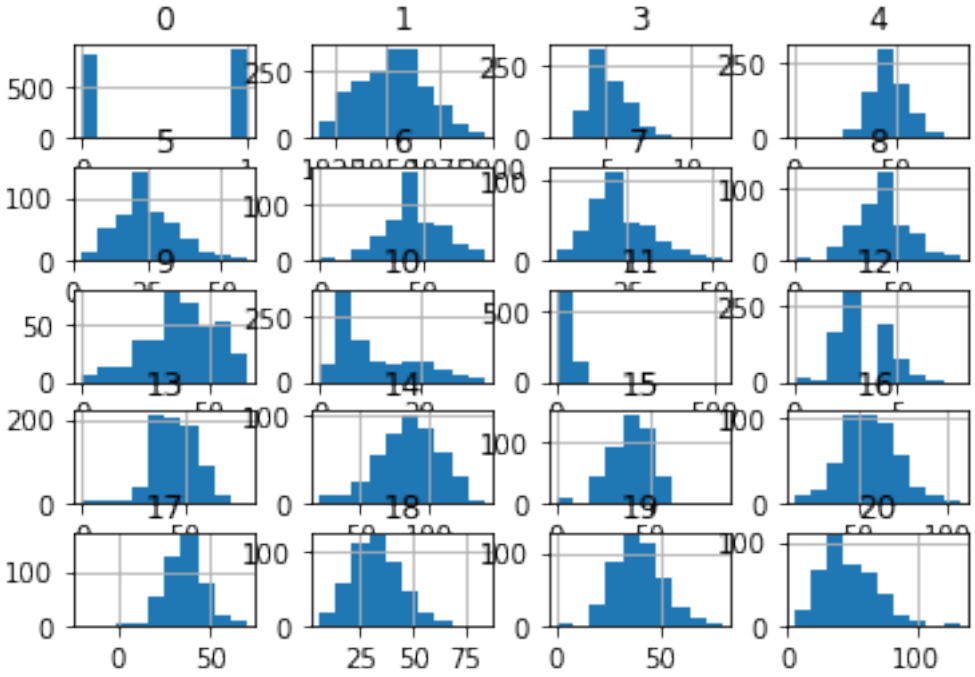
```
Sexe@Sexe@Sexe
Data Naixament@Data Naixament@Data Naixament
Diagnòstic@Diagnòstic@Diagnòstic
```

```
Dígitos: directos (WAIS)@Digits directesWaisPD@PD
Dígitos: directos (WAIS)@Digits directesWaisPcPe@Pe
TAPPING de McQuarrie@Ma dominant@Ma dominant
TAPPING de McQuarrie@Ma dominantPT@PT
TAPPING de McQuarrie@Ma nodominant@Ma nodominant
TAPPING de McQuarrie@Ma nodominantPT@PT
Orientación de líneas RBANS@Orienta liniesPD@PD
Orientación de líneas RBANS@Orienta liniesPT@PT
Lista de palabras del RBANS_verbal@Corba aprenentatgePD@PD
Animales_semantica@SemanticaPT@PT
Dígitos: inversos (WAIS)@Digits inversosWaisPD@PD
Dígitos: inversos (WAIS)@Digits inversosWaisPcPe@Pe
Verbal: Stroop@P ParaulaPD@PD
Verbal: Stroop@P ParaulaPT@PT
Verbal: Stroop@C ColorPD@PD
Verbal: Stroop@C ColorPT@PT
Verbal: Stroop@PC ParaulaColorPD@PD
Verbal: Stroop@PC ParaulaColorPT@PT
Verbal: Stroop@PC1PD@PD
Visual: Clave de números - Codificación (WAIS-III)@Correctes@Correctes
```

```
In [17]: df_total['Verbal: Stroop@PC1PD@PD'].hist()
         df_total['Orientación de líneas RBANS@Orienta liniesPD@PD'].hist()
         plt.show()
         del df_total['Verbal: Stroop@PC1PD@PD']
         del df_total['Orientación de líneas RBANS@Orienta liniesPD@PD']
```



```
In [18]: df = df_total.copy()
         df.columns = [i for i in range(len(df.columns))]
         df.hist()
         plt.show()
```

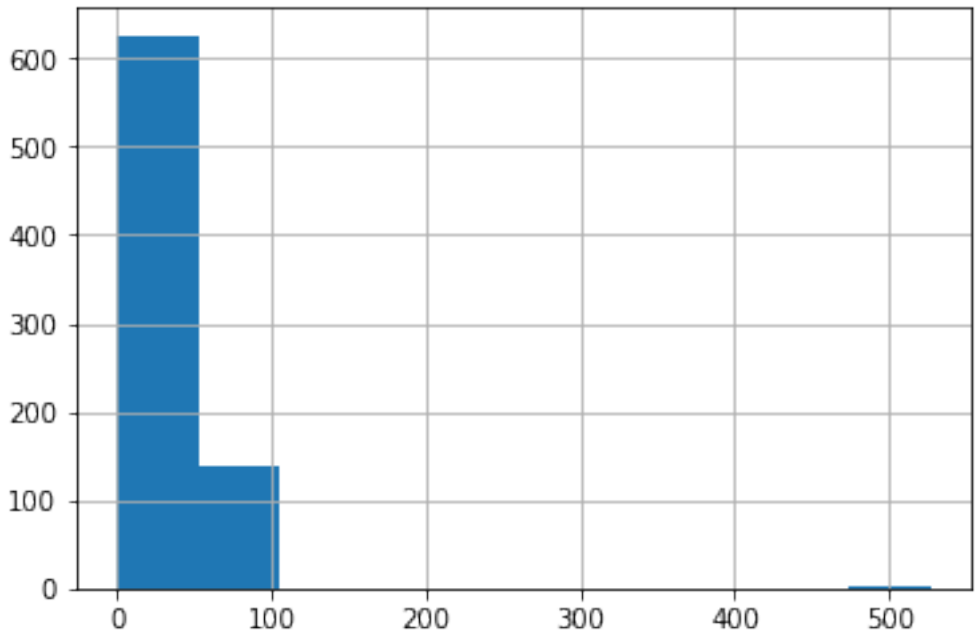



```
In [18]: for column in df_total.columns:
             print(column)
```

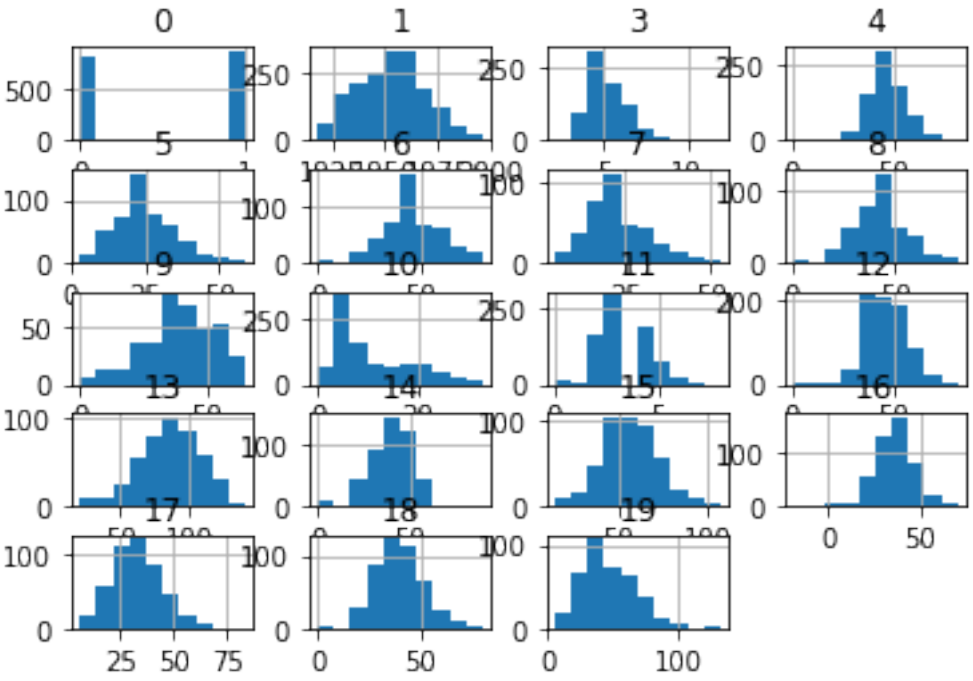
```
Sexe@Sexe@Sexe
Data Naixament@Data Naixament@Data Naixament
Diagnòstic@Diagnòstic@Diagnòstic
Verbal: Stroop@P ParaulaPD@PD
Verbal: Stroop@P ParaulaPT@PT
Verbal: Stroop@C ColorPD@PD
Verbal: Stroop@C ColorPT@PT
Verbal: Stroop@PC ParaulaColorPD@PD
Verbal: Stroop@PC ParaulaColorPT@PT
Visual: Clave de números - Codificación (WAIS-III)@Correctes@Correctes
Dígitos: inversos (WAIS)@Digits inversosWaisPD@PD
Dígitos: inversos (WAIS)@Digits inversosWaisPcPe@Pe
TAPPING de McQuarrie@Ma dominant@Ma dominant
TAPPING de McQuarrie@Ma dominantPT@PT
TAPPING de McQuarrie@Ma nodominant@Ma nodominant
```

```
TAPPING de McQuarrie@Ma nodominantPT@PT
Lista de palabras del RBANS_verbal@Corba aprenentatgePD@PD
Animales_semantica@SemanticaPT@PT
Orientación de líneas RBANS@Orienta liniesPT@PT
Dígitos: directos (WAIS)@Digits directesWaisPD@PD
Dígitos: directos (WAIS)@Digits directesWaisPcPe@Pe
```

```
In [19]: df_total['Animales_semantica@SemanticaPT@PT'].hist()
         plt.show()
         del df_total['Animales_semantica@SemanticaPT@PT']
```




```
In [20]: df = df_total.copy()
         df.columns = [i for i in range(len(df.columns))]
         df.hist()
         plt.show()
```

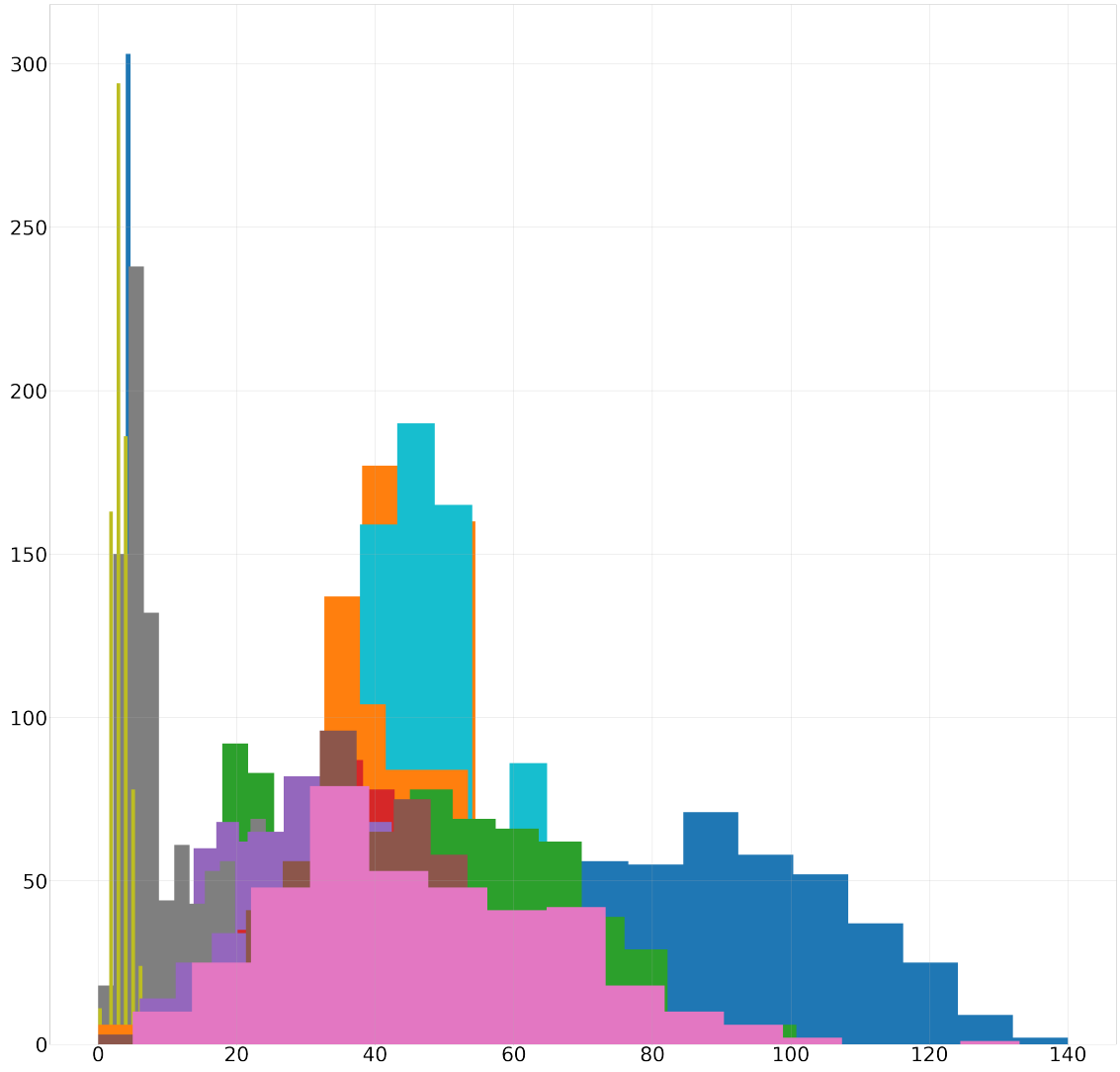



```
In [21]: for column in df_total.columns:
             print(column)
```

```
Sexe@Sexe@Sexe
Data Naixament@Data Naixament@Data Naixament
Diagnòstic@Diagnòstic@Diagnòstic
Verbal: Stroop@P ParaulaPD@PD
Verbal: Stroop@P ParaulaPT@PT
Verbal: Stroop@C ColorPD@PD
Verbal: Stroop@C ColorPT@PT
Verbal: Stroop@PC ParaulaColorPD@PD
Verbal: Stroop@PC ParaulaColorPT@PT
Visual: Clave de números - Codificación (WAIS-III)@Correctes@Correctes
Dígitos: inversos (WAIS)@Digits inversosWaisPD@PD
Dígitos: inversos (WAIS)@Digits inversosWaisPcPe@Pe
TAPPING de McQuarrie@Ma dominant@Ma dominant
TAPPING de McQuarrie@Ma dominantPT@PT
TAPPING de McQuarrie@Ma nodominant@Ma nodominant
TAPPING de McQuarrie@Ma nodominantPT@PT
Lista de palabras del RBANS_verbal@Corba aprenentatgePD@PD
Orientación de líneas RBANS@Orienta liniesPT@PT
Dígitos: directos (WAIS)@Digits directesWaisPD@PD
Dígitos: directos (WAIS)@Digits directesWaisPcPe@Pe
```

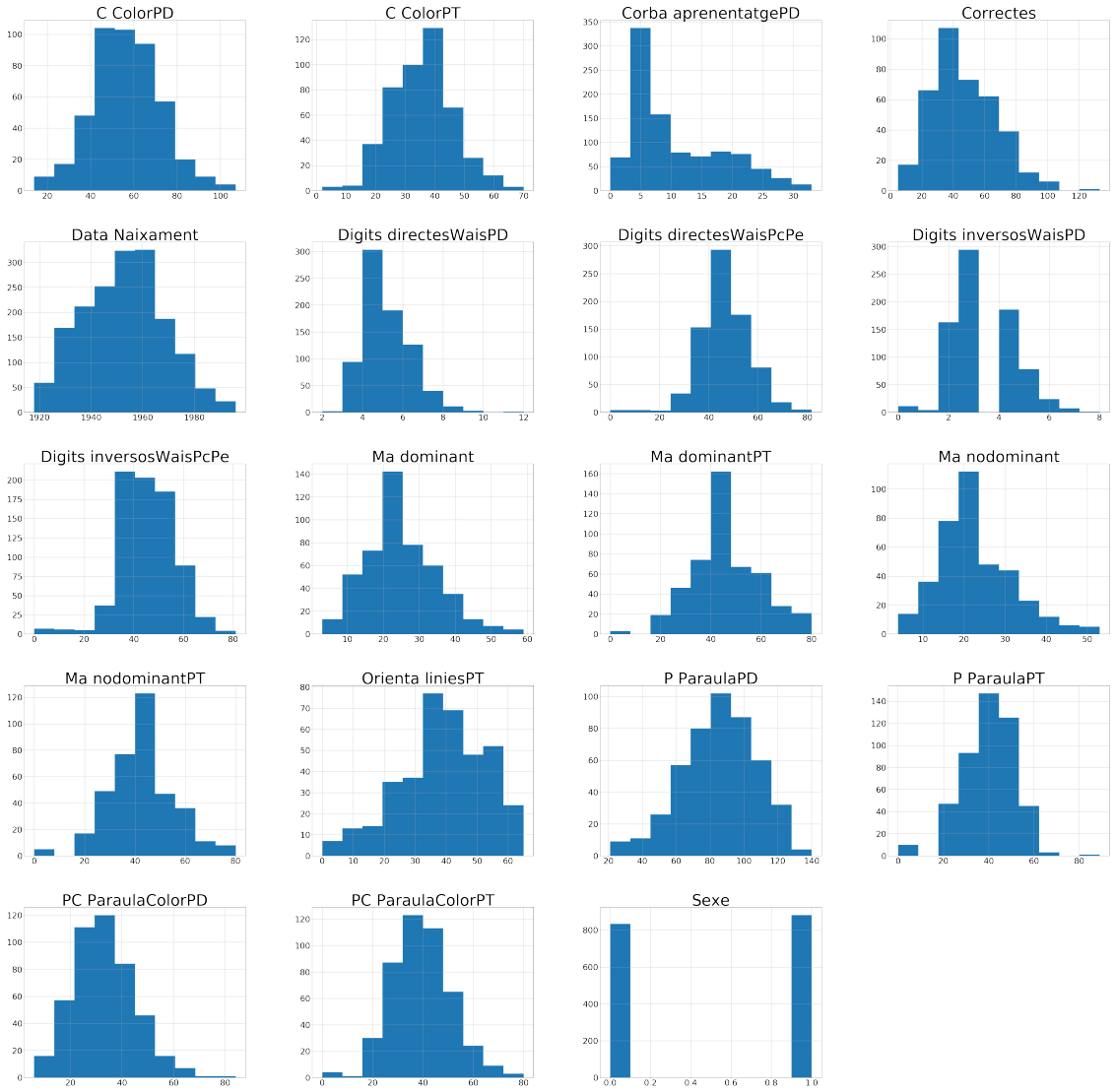
In [21]:
```
df_total['Verbal: Stroop@C ColorPT@PT'] = df_total['Verbal: Stroop@C ColorPT@PT'].apply
```

In [22]:
```
df = df_total.copy()
params = {'axes.titlesize':'64',
          'xtick.labelsize':'84',
          'ytick.labelsize':'84'}
matplotlib.rcParams.update(params)

for column in df.iloc[:,3:].columns:
    df[column].hist(bins=15,figsize=(80,80))

plt.savefig('../Figures/Comparison unified database.png',bbox_inches='tight')
plt.show()
```



In [23]:
```
df_copy = df.copy()
```

```
columns = []
for column in df_copy.columns:
    columns.append(column.split("@")[1])

df_copy.columns = columns
params = {'axes.titlesize':'64',
          'xtick.labelsize':'34',
          'ytick.labelsize':'34'}

matplotlib.rcParams.update(params)
df_copy.hist(figsize=(80,80))
plt.savefig('../Figures/histogram description2', bbox_inches='tight')
```



In [25]: `list_tuples_new_structure = []`

```
        for col in df_total.columns:
            list_tuples_new_structure.append((col.split("@")[0], col.split("@")[1], col.split("
```

In [26]:
```
        # Guardamos la info sin estructurar y la estructura en un JSON
        if not os.path.exists('../Unified Database/'):
            os.makedirs('../Unified Database/')

        df = df_total.copy()

        df.columns = range(df.columns.shape[0])
        df.to_excel('../Unified Database/' + "Unified Database" + '.xlsx')

        with open('unified_database_tuples_for_index_dict.json', 'w') as fp:
            json.dump(list_tuples_new_structure, fp)
```

Outlier Detection

September 3, 2018

1 Outlier Detection:

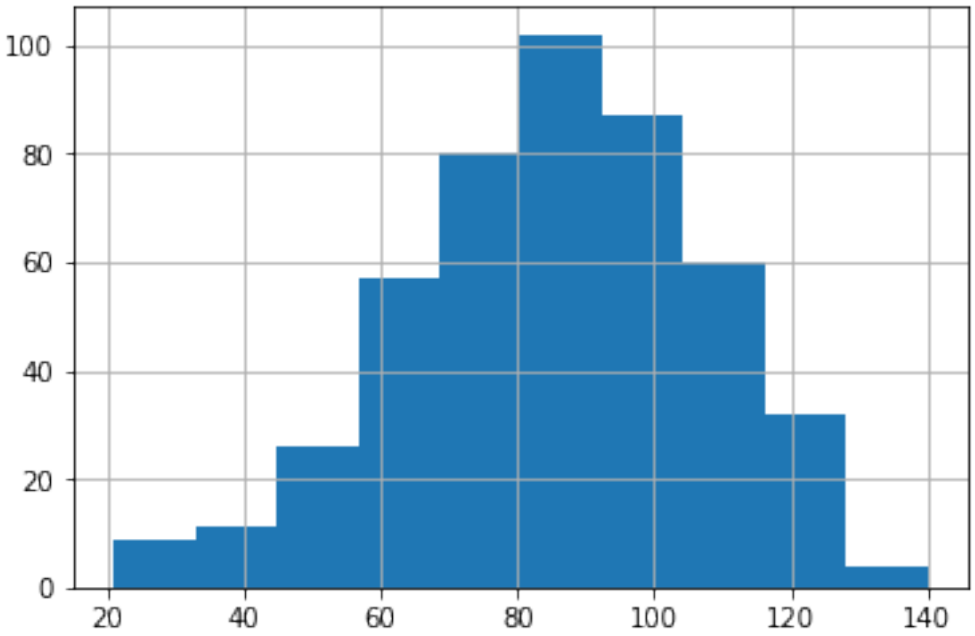
Here we filter out posible outliers introduced in the database due to human errors. To make that we use the categories from the hospital. In the case of PD punctuation as we don't know the ranges we let the values as they are. We also search for those value very big with only one decimal because we think that there is an error in the deciaml point.

```
In [19]: # Import libraries
         import pandas as pd
         import numpy as np
         import load_unified_data
         import matplotlib.pyplot as plt
         import os
         import json
```

```
In [20]: data_frames, tuples_new_structure = load_unified_data.read_and_format('../Unified Datab
```

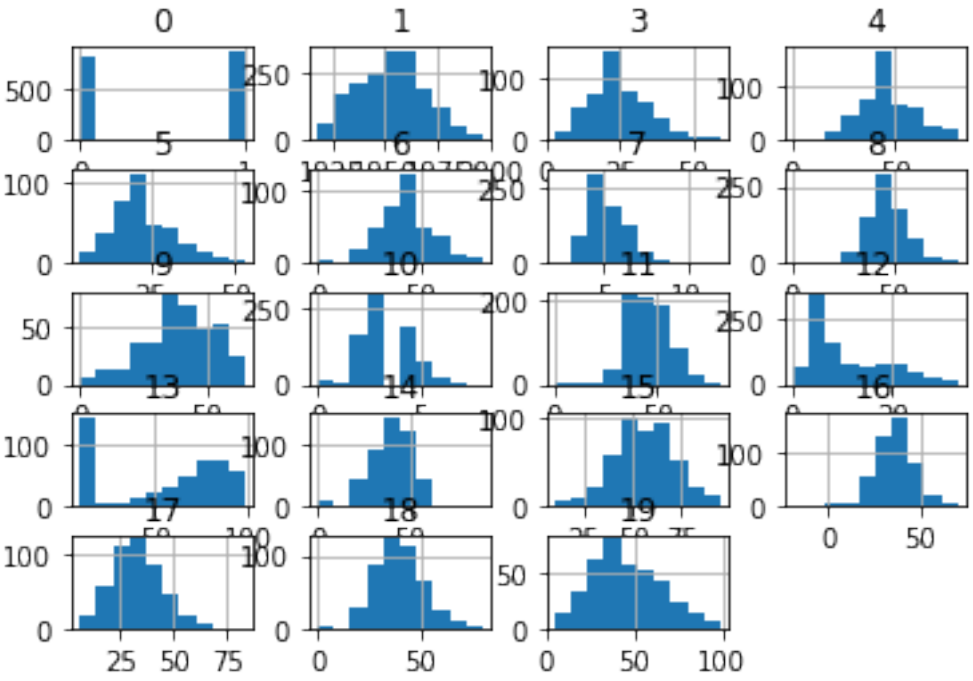
```
In [9]: def filter_outliers(column):
            try:
                col_control = column * 10 % 10
                if column >= 100 and col_control.is_integer():
                    column = column / 10
                elif column >= 100 and ~col_control.is_integer():
                    column = np.nan
            except:
                pass
            return column
```

```
In [18]: data_frames['Verbal: Stroop']['P ParaulaPD']['PD'].hist()
         plt.show()
```




```
In [10]: for column in data_frames.columns:
             data_frames[column] = data_frames[column].apply(filter_outliers)
```

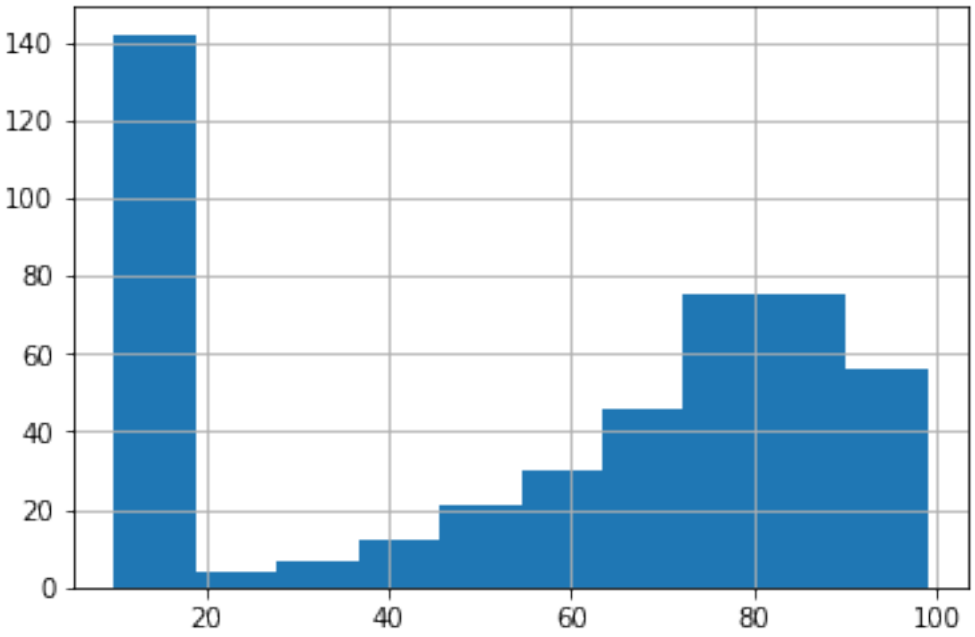
```
In [11]: df = data_frames.copy()
         df.columns = [i for i in range(len(df.columns))]
         df.hist()
         plt.show()
```




```
In [14]: for column in data_frames.columns:
             print(column)
```

```
('Sexe', 'Sexe', 'Sexe')
('Data Naixament', 'Data Naixament', 'Data Naixament')
('Diagnòstic', 'Diagnòstic', 'Diagnòstic')
('TAPPING de McQuarrie', 'Ma dominant', 'Ma dominant')
('TAPPING de McQuarrie', 'Ma dominantPT', 'PT')
('TAPPING de McQuarrie', 'Ma nodominant', 'Ma nodominant')
('TAPPING de McQuarrie', 'Ma nodominantPT', 'PT')
('Dígitos: directos (WAIS)', 'Digits directesWaisPD', 'PD')
('Dígitos: directos (WAIS)', 'Digits directesWaisPcPe', 'Pe')
('Orientación de líneas RBANS', 'Orienta liniesPT', 'PT')
('Dígitos: inversos (WAIS)', 'Digits inversosWaisPD', 'PD')
('Dígitos: inversos (WAIS)', 'Digits inversosWaisPcPe', 'Pe')
('Lista de palabras del RBANS_verbal', 'Corba aprenentatgePD', 'PD')
('Verbal: Stroop', 'P ParaulaPD', 'PD')
('Verbal: Stroop', 'P ParaulaPT', 'PT')
('Verbal: Stroop', 'C ColorPD', 'PD')
('Verbal: Stroop', 'C ColorPT', 'PT')
('Verbal: Stroop', 'PC ParaulaColorPD', 'PD')
('Verbal: Stroop', 'PC ParaulaColorPT', 'PT')
('Visual: Clave de números - Codificación (WAIS-III)', 'Correctes', 'Correctes')
```

In [15]:
```
data_frames['Verbal: Stroop']['P ParaulaPD']['PD'].hist()
plt.show()
```



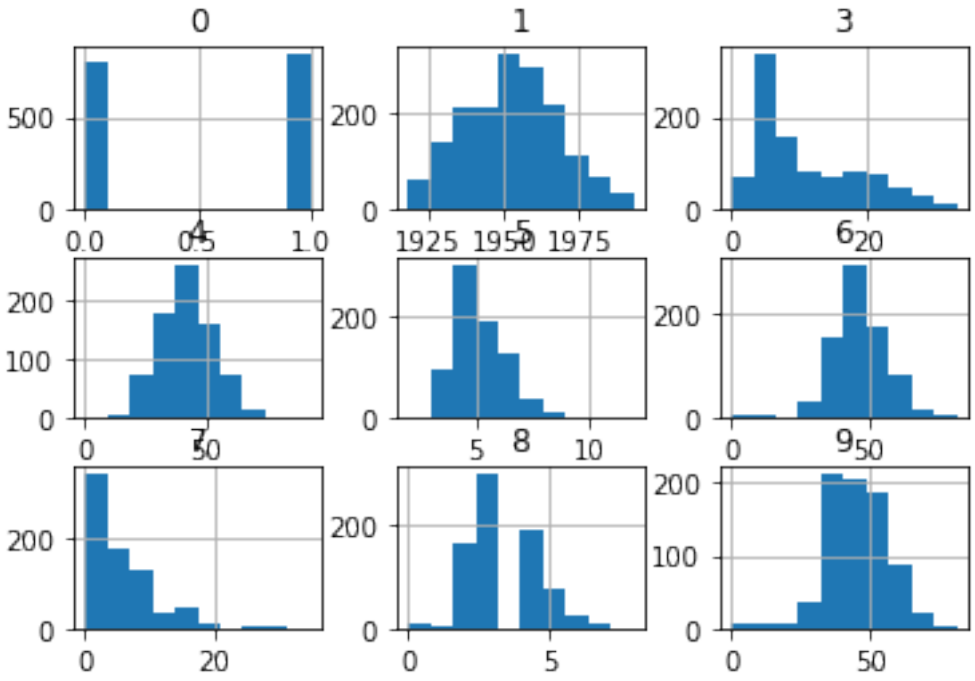
In [13]:
```
df = data_frames['CPT (Continuos Performance Test)']['PerserveracionsPD']['PD'].copy()
df = df.apply(lambda x: x if x < 40 else np.nan)
data_frames['CPT (Continuos Performance Test)'] = data_frames['CPT (Continuos Performan
data_frames['CPT (Continuos Performance Test)']['PerserveracionsPD']['PD'].hist()
plt.show()
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

~/ENV/lib/python3.5/site-packages/pandas/core/indexes/base.py in get_loc(self, key, meth
   2524             try:
-> 2525                 return self._engine.get_loc(key)
   2526             except KeyError:


pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()
```

```
    pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_


    pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_


    KeyError: 'CPT (Continuos Performance Test)'


During handling of the above exception, another exception occurred:


    KeyError                                  Traceback (most recent call last)

    <ipython-input-13-3091d40315c9> in <module>()
----> 1 df = data_frames['CPT (Continuos Performance Test)']['PerserveracionsPD']['PD'].copy
      2 df = df.apply(lambda x: x if x < 40 else np.nan)
      3 data_frames['CPT (Continuos Performance Test)'] = data_frames['CPT (Continuos Perfor
      4 data_frames['CPT (Continuos Performance Test)']['PerserveracionsPD']['PD'].hist()
      5 plt.show()


    ~/ENV/lib/python3.5/site-packages/pandas/core/frame.py in __getitem__(self, key)
   2135             return self._getitem_frame(key)
   2136         elif is_mi_columns:
-> 2137             return self._getitem_multilevel(key)
   2138         else:
   2139             return self._getitem_column(key)


    ~/ENV/lib/python3.5/site-packages/pandas/core/frame.py in _getitem_multilevel(self, key)
   2179
   2180     def _getitem_multilevel(self, key):
-> 2181         loc = self.columns.get_loc(key)
   2182         if isinstance(loc, (slice, Series, np.ndarray, Index)):
   2183             new_columns = self.columns[loc]


    ~/ENV/lib/python3.5/site-packages/pandas/core/indexes/multi.py in get_loc(self, key, met
   2070
   2071         if not isinstance(key, tuple):
-> 2072             loc = self._get_level_indexer(key, level=0)
   2073             return _maybe_to_slice(loc)
   2074


    ~/ENV/lib/python3.5/site-packages/pandas/core/indexes/multi.py in _get_level_indexer(sel
   2360         else:
```

```
   2361
-> 2362             loc = level_index.get_loc(key)
   2363             if isinstance(loc, slice):
   2364                 return loc


 ~/ENV/lib/python3.5/site-packages/pandas/core/indexes/base.py in get_loc(self, key, meth
   2525                 return self._engine.get_loc(key)
   2526             except KeyError:
-> 2527                 return self._engine.get_loc(self._maybe_cast_indexer(key))
   2528
   2529         indexer = self.get_indexer([key], method=method, tolerance=tolerance)


 pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


 pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


 pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_


 pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_


 KeyError: 'CPT (Continuos Performance Test)'
```

```
In [31]: df = data_frames.copy()
         df.columns = [i for i in range(len(df.columns))]
         df.hist()
         plt.show()
```



In [33]:
```
# Guardamos la info sin estructurar y la estructura en un JSON
if not os.path.exists('../Final Database/'):
    os.makedirs('../Final Database/')

df = data_frames.copy()

df.columns = range(df.columns.shape[0])
df.to_excel('../Final Database/' + "Final Database" + '.xlsx')

with open('final_database_tuples_for_index_dict.json', 'w') as fp:
    json.dump(tuples_new_structure, fp)
```

Discretization

September 3, 2018

1 Discretization of the available data:

Here we transform the continous scores from the tests into categorical variables. Following this mapping:

PUNTUACIONES:

- Puntuación Típica (PT) - PT: 50 +- 10 (40 límite inferior/ 30-40 déficit leve/ 20-30 déficit moderado/ <20 déficit grave
- Puntuación Escalar (Pe) – Pe: 10 +- 3 (No solemos especificar gravedad)
- Percentiles (Pc) Pc: 0-10 Deficitario/ 10-30 Bajo/ 30-80 Medio/ 80-95 Máximo

```
In [1]: # Import libraries
        import pandas as pd
        import numpy as np
        import load_unified_data
        import os
        import json
```

```
In [2]: data_frame, tuples_structure = load_unified_data.read_and_format('../Unified Database','
```

```
In [3]: # Tenemos que crear un diccionario de bins
        bins_dict = {'PT': [0, 20, 30, 40, 50, 100],
                     'Pe': [0, 7, 13, 100],
                     'Pc': [0, 16, 100],
                     'Pc Test Barcelona': [0, 10, 30, 80, 95],
                     'CPT': [0, 60, 100],
                     'BDI': [0, 10, 16, 20, 30, 40, 100],
                     'BRIEF-A': [0, 65, 100],
                     'AB (SKT)': [0, 4, 8, 13, 18, 23, 27],
                     'MMSE': [0, 10, 17, 23, 100],
                     'MEC': [0, 30, 100],
                     'MOCA': [0, 26, 100],
                     'FAB': [0, 12, 100],
                     'TAM': [0, 32, 100],
                     'Puntuació línies': [0, 17, 19, 21, 23, 25, 27, 29, 30],
                     'Pc línies': [0, 1.5, 4, 9, 22, 40, 56, 72, 86, 100],
                     'Discriminació visual (Benton)': [0, 23, 24, 25, 100],
                     'VOT': [0, 10, 20, 25, 100]}
```

In [4]:
```
# Implementamos bucle. Probamos con un df solo:
df = data_frame.copy()
index = df.index
df_discretized = df.iloc[:,:3].copy()

for lvl0, lvl1, lvl2 in df.columns:
    if lvl0 in ["Sexe", "Data Naixament", "Diagnòstic"]:
        continue
    try:
        bins = bins_dict[lvl2]
        datos = pd.DataFrame(np.digitize(df[lvl0][lvl1][lvl2], bins), index=index)
        datos.loc[datos.loc[datos.iloc[:,0] == len(bins)].index] = np.nan
        df_discretized = pd.concat([df_discretized, datos], axis=1)
    except:
        # Discretizing by binning
        min_value = df[lvl0][lvl1][lvl2].min()
        max_value = df[lvl0][lvl1][lvl2].max()
        bins = np.linspace(min_value, max_value, 6)
        datos = pd.DataFrame(np.digitize(df[lvl0][lvl1][lvl2], bins), index=index)
        datos.loc[datos.loc[datos.iloc[:,0] == len(bins)].index] = np.nan
        df_discretized = pd.concat([df_discretized, datos], axis=1)
        pass

data = [list(df_discretized.iloc[i,:].get_values()) for i in range(len(df_discretized))]
columns = pd.MultiIndex.from_tuples(tuples_structure)
df_discretized_structure = pd.DataFrame(data, columns=columns, index=index)
```

In [5]:
```
# Guardamos la info sin estructurar y la estructura en un JSON
if not os.path.exists('../Final Discretized Database/'):
    os.makedirs('../Final Discretized Database/')

df = df_discretized.copy()

df.columns = range(df.columns.shape[0])
df.to_excel('../Final Discretized Database/' + "Final Discretized Database" + '.xlsx')

with open('final_discretized_database_tuples_for_index_dict.json', 'w') as fp:
    json.dump(tuples_structure, fp)
```

Analyzing Data Distributions

September 3, 2018

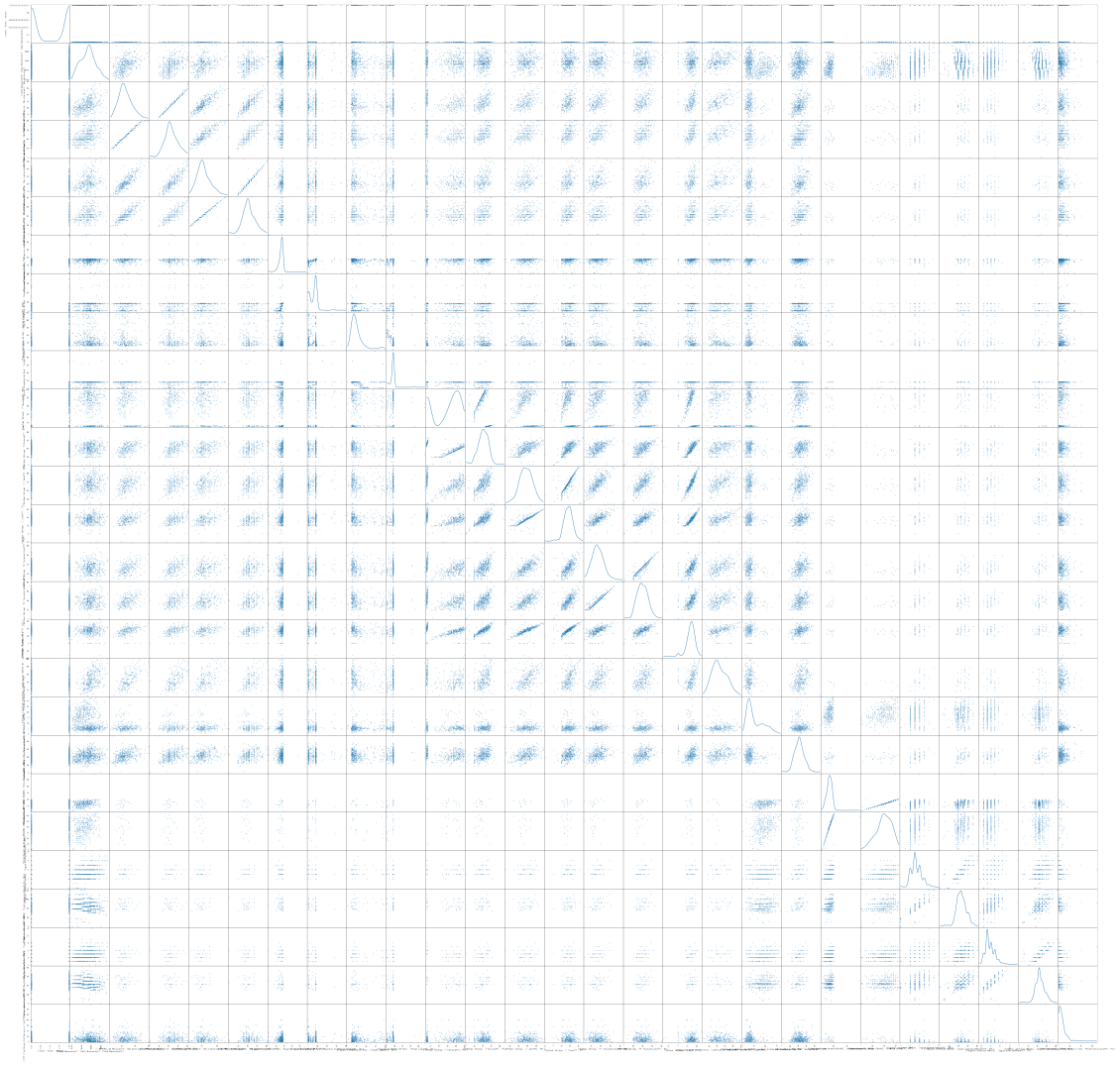
1 Analyzing Data Distributions:

Here we analizy the distributions of the resulting columns based in graphical descriptions.

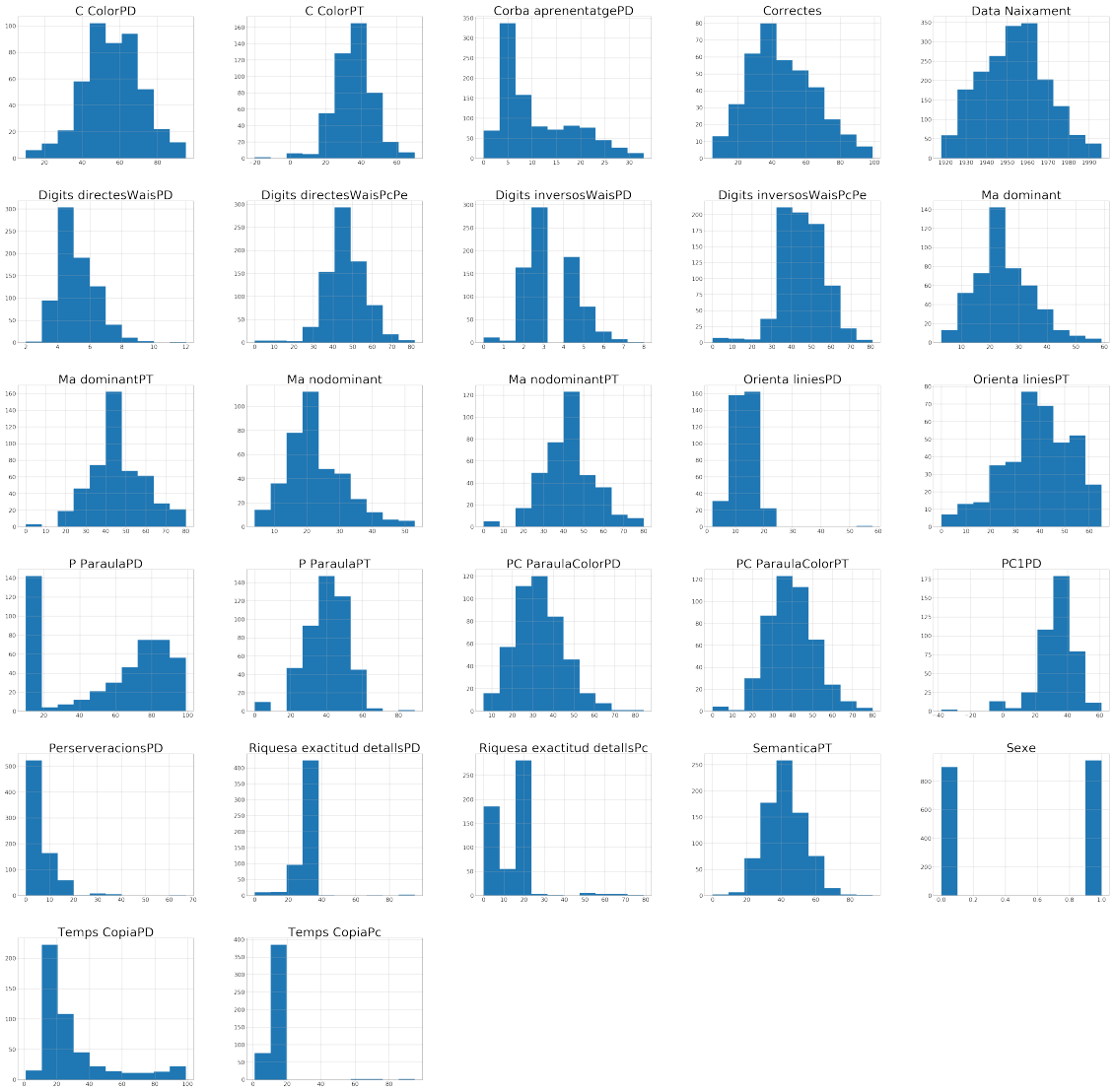
```
In [27]: # Import libraries
         import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import matplotlib
         import load_unified_data
```

```
In [2]: data_frame, tuples_structure = load_unified_data.read_and_format('../Final Database','fi
```

```
In [12]: fig_size = plt.rcParams["figure.figsize"]
         fig_size[0] = 80
         fig_size[1] = 80
         pd.plotting.scatter_matrix(data_frame, diagonal='kde')
         plt.savefig('../Figures/global description', bbox_inches='tight')
```



```
In [20]: df = data_frame.copy()
         index = df.index
         columns = df.columns.get_level_values(1)
         data = [list(df.iloc[i,:].get_values()) for i in range(len(df))]
         df = pd.DataFrame(data, columns=columns, index=index)

In [30]: params = {'axes.titlesize':'50',
                   'xtick.labelsize':'24',
                   'ytick.labelsize':'24'}
         matplotlib.rcParams.update(params)
         df.hist()
         plt.savefig('../Figures/histogram description', bbox_inches='tight')
```


C ColorPD
C ColorPT
Corba aprenentatgePD
Correctes
Data Naixament
Digits directesWaisPD
Digits directesWaisPcPe
Digits inversosWaisPD
Digits inversosWaisPcPe
Ma dominant
Ma dominantPT
Ma nodominant
Ma nodominantPT
Orienta liniesPD
Orienta liniesPT
P ParaulaPD
P ParaulaPT
PC ParaulaColorPD
PC ParaulaColorPT
PC1PD
PerserveracionsPD
Riquesa exactitud detallsPD
Riquesa exactitud detallsPc
SemanticaPT
Sexe
Temps CopiaPD
Temps CopiaPc

Discretization (Binary)

September 3, 2018

1 Discretization of the available data:

Here we transform the continous scores from the tests into categorical variables. Following this mapping:
PUNTUACIONES:

- Puntuación Típica (PT) - PT: 50 +- 10 (40 límite inferior/ 30-40 déficit leve/ 20-30 déficit moderado/ <20 déficit grave
- Puntuación Escalar (Pe) – Pe: 10 +- 3 (No solemos especificar gravedad)
- Percentiles (Pc) Pc: 0-10 Deficitario/ 10-30 Bajo/ 30-80 Medio/ 80-95 Máximo

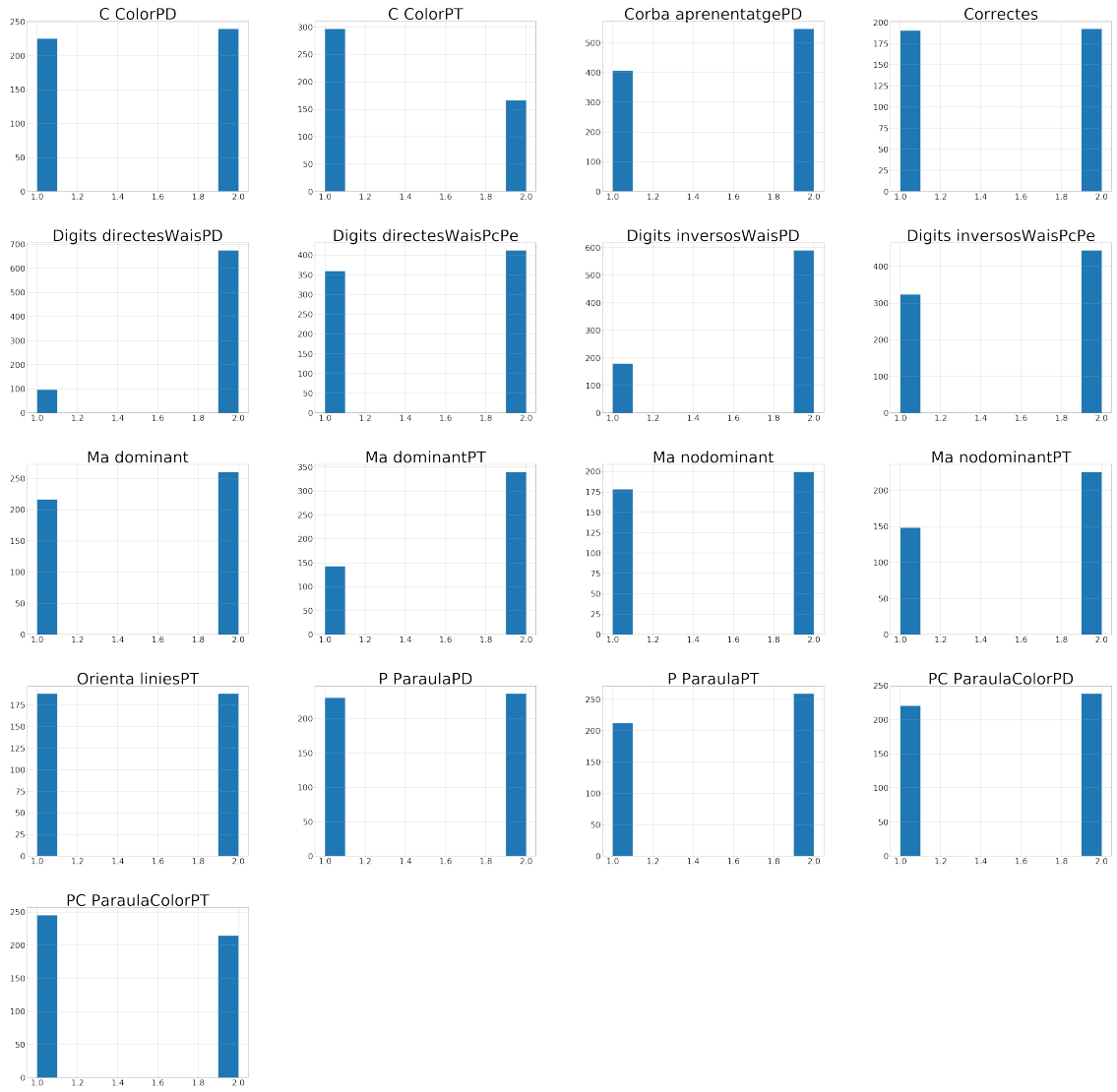
In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import load_unified_data
import matplotlib.pyplot as plt
import matplotlib
import os
import json
```

In [2]:
```
data_frame, tuples_structure = load_unified_data.read_and_format('../Unified Database','
```

In [3]:
```
# Tenemos que crear un diccionario de bins
bins_dict = {'PT': [0, 40, 100],
             'Pc': [0, 30, 80, 100],
             'Pc Test Barcelona': [0, 30, 80, 100],
             'CPT': [0, 40, 60, 100],
             'BDI': [0, 16, 100],
             'BRIEF-A': [0, 65, 100],
             'AB (SKT)': [0, 18, 100],
             'MMSE': [0, 23, 100],
             'MEC': [0, 30, 100],
             'MOCA': [0, 26, 100],
             'FAB': [0, 12, 100],
             'TAM': [0, 32, 100],
             'Puntuació línies': [0, 20, 100],
             'VOT': [0, 20, 100]}
```

In [4]:
```
# Implementamos bucle. Probamos con un df solo:
df = data_frame.copy()
index = df.index
df_discretized = df.iloc[:,:3].copy()

for lvl0, lvl1, lvl2 in df.columns:
    if lvl0 in ["Sexe", "Data Naixament", "Diagnòstic"]:
        continue
    try:
        bins = bins_dict[lvl2]
        datos = pd.DataFrame(np.digitize(df[lvl0][lvl1][lvl2], bins), index=index)
        datos.loc[datos.loc[datos.iloc[:,0] == len(bins)].index] = np.nan
        df_discretized = pd.concat([df_discretized, datos], axis=1)
    except:
        # Discretizing by binning
        mean = df[lvl0][lvl1][lvl2].describe().loc['50%']
        minimum = df[lvl0][lvl1][lvl2].describe().loc['min']
        maximum = df[lvl0][lvl1][lvl2].describe().loc['max']
        bins = np.array((minimum, mean, maximum))
        datos = pd.DataFrame(np.digitize(df[lvl0][lvl1][lvl2], bins), index=index)
        datos.loc[datos.loc[datos.iloc[:,0] == len(bins)].index] = np.nan
        df_discretized = pd.concat([df_discretized, datos], axis=1)
        pass

data = [list(df_discretized.iloc[i,:].get_values()) for i in range(len(df_discretized))]
columns = pd.MultiIndex.from_tuples(tuples_structure)
df_discretized_structure = pd.DataFrame(data, columns=columns, index=index)
```

In [7]:
```
df_copy = df_discretized_structure.iloc[:,3:].copy()

df_copy.columns = df_copy.columns.get_level_values(1)
params = {'axes.titlesize':'64',
          'xtick.labelsize':'34',
          'ytick.labelsize':'34'}

matplotlib.rcParams.update(params)
df_copy.hist(figsize=(80,80))
plt.savefig('../Figures/histogram description binary 1', bbox_inches='tight')
plt.show()
```



In [6]:
```
# Guardamos la info sin estructurar y la estructura en un JSON
if not os.path.exists('../Final Discretized Database/'):
    os.makedirs('../Final Discretized Database/')

df = df_discretized.copy()

df.columns = range(df.columns.shape[0])
df.to_excel('../Final Discretized Database/' + "Final Discretized Database" + '.xlsx')

with open('final_discretized_database_tuples_for_index_dict.json', 'w') as fp:
    json.dump(tuples_structure, fp)
```

Filling Missing Values

September 3, 2018

1 Filling missing values:

Here is an example of the scripts employed for filling missing values for the continous and discrete cases.

```
In [1]: # Import libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import load_unified_data
        import load_unified_data_discrete
        from sklearn.preprocessing import Imputer

In [2]: data_frame, tuples_structure = load_unified_data.read_and_format('../Final Database', 'f

In [3]: data_frame_discrete, tuples_structure_discrete = load_unified_data_discrete.read_and_for

In [18]: # Continous case
         mode = ['mean', 'median', 'most_frequent']

         df = data_frame.copy()
         imp = Imputer(missing_values=np.nan, strategy=mode[1])
         df.iloc[:,3:] = imp.fit_transform(df.iloc[:,3:])

In [16]: # Discrete case
         mode_discrete = {'optimistic': 'min', 'pesimistic': 'max', 'neutral': '50%'}
         df_discrete = data_frame_discrete.copy()

         for column in df_discrete.iloc[:, 3:].columns:
             value = round(df_discrete.describe().loc[mode_discrete['neutral']][column])
             df_discrete[column] = df_discrete[column].fillna(value)
```

Data mining continous case

September 3, 2018

1 Data mining continous case:

We will implement a function that recibes a data base and returns a set of meassures for quantify the performance of the algorithm.

```
In [1]: # Import libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib
        import load_unified_data
        from sklearn.preprocessing import Imputer
        from sklearn.model_selection import KFold
        from sklearn import linear_model
        from sklearn import svm
        from sklearn import neighbors
        from sklearn.tree import DecisionTreeRegressor

In [2]: data_frame, tuples_structure = load_unified_data.read_and_format('../Unified Database',

In [3]: mode = ['mean', 'median', 'most_frequent']

        df_mean_array = []
        df_median_array = []
        df_most_frequent_array = []
        df = data_frame.copy()
        columns = set(df.iloc[:,3:].columns.get_level_values(0))
        for column in columns:
            actual_column = df[column].copy().dropna()
            index = actual_column.index
            df_actual = df.loc[index]
            long = len(df_actual)
            info_actual_columns = []
            for column_actual in df_actual.iloc[:,3:].columns:
                percentage = df_actual[column_actual].describe().loc['count'] * 100 / long
                if percentage < 70.0:
                    del df_actual[column_actual]
```

```
            df_mean = df_actual.copy()
            imp = Imputer(missing_values=np.nan, strategy=mode[0])
            df_mean.iloc[:,3:] = imp.fit_transform(df_mean.iloc[:,3:])
            if len(set(df_actual.iloc[:,3:].columns.get_level_values(0))) > 1:
                df_mean_array.append(df_mean)

            df_median = df_actual.copy()
            imp = Imputer(missing_values=np.nan, strategy=mode[1])
            df_median.iloc[:,3:] = imp.fit_transform(df_median.iloc[:,3:])
            if len(set(df_mean.iloc[:,3:].columns.get_level_values(0))) > 1:
                df_median_array.append(df_median)

            df_most_frequent = df_actual.copy()
            imp = Imputer(missing_values=np.nan, strategy=mode[2])
            df_most_frequent.iloc[:,3:] = imp.fit_transform(df_most_frequent.iloc[:,3:])
            if len(set(df_mean.iloc[:,3:].columns.get_level_values(0))) > 1:
                df_most_frequent_array.append(df_most_frequent)
```

In [4]:
```
for df in df_mean_array:
    print(len(df.columns), len(df))
```

```
9 376
7 770
10 454
11 383
7 768
```

In [5]:
```
df1 = df_mean_array[-2].copy()
df1
```

Out[5]:

	Sexe	Data Naixament	Diagnòstic	Verbal: Stroop P	Paraula PD
	Sexe	Data Naixament	Diagnòstic		PD
2013-12-19-3199504984151427072	1	1959	1601TP	100.000000	
2014-02-25-5076076858799773696	1	1959	101EPI	108.000000	
2013-07-042048871307966173952	1	1957	1601TRA	104.000000	
2013-10-086383101741557695488	0	1964	L-PNE	110.000000	
2014-07-081733670778237366016	0	1946	3UR0108	91.000000	
2011-11-094148544000739942912	0	1967	1002DEP	87.618421	
2017-04-184748903970519549952	0	1960	1601TBP	87.618421	
2016-01-08-6056684102447908864	1	1964	500303	95.000000	
2014-01-217632868070695424000	0	1945	500973	77.000000	
2011-05-18-3778282014351102976	1	1955	1601DN	95.000000	
2013-12-24-1139302521462558976	1	1963	501881	86.000000	
2010-02-05-1477311665734011904	0	1962	500868	87.618421	
2015-10-02-2953031948695728128	0	1958	502149	102.000000	
2013-05-283082888525163273216	1	1968	501042	122.000000	

2016-11-222717391739930320896	0	1962	500176	31.000000
2016-11-177740231173833011200	0	1970	1901ICT	83.000000
2009-10-286180148857292527616	0	1952	1601DCO	104.000000
2011-05-12-3778282014351102976	1	1955	1601DN	87.618421
2014-10-175604819893483879424	1	1964	1601DCO	87.618421
2011-07-08-2543111189480854016	0	1955	1002DEP	88.000000
2015-04-30128771044112546400	0	1960	3UR0133	117.000000
2013-11-12-1937500370690820096	1	1984	500997	87.618421
2012-05-242679390329982595072	1	1937	1601DCO	74.000000
2011-06-21-4034696200536380928	0	1929	501503	88.000000
2017-01-195063911599998003200	0	1965	309.28	126.000000
2014-10-283013508979533191168	1	1956	1601TP	87.618421
2012-12-11-1667285258272548096	0	1953	1601DCO	68.000000
2017-12-15-988477930897196288	1	1962	502184	117.000000
2015-09-248685386897285894144	0	1964	1601TRM	91.000000
2014-10-28-6453551274044865536	1	1960	500303	125.000000
...	...	...	...	...
2009-12-098466316493670267904	1	1983	1601DCO	64.000000
2016-10-25-7521791065392616448	1	1983	3UR0042	107.000000
2009-02-204857606362581025792	0	1952	1601DCO	32.000000
2014-12-16-1921873807210128896	1	1953	2401DC	67.000000
2015-11-17-5615828347483806720	0	1955	500970	109.000000
2014-01-13-7824983083899136000	0	1958	3UR0036	60.000000
2009-03-105351069396620515328	1	1972	1601TOC	100.000000
2016-04-01251684784402089408	0	1958	503541	100.000000
2012-03-16-7832625167647364096	0	1932	1601DCO	121.000000
2009-03-06-9014219213997190144	0	1972	1601DCO	87.000000
2012-02-212048871307966173952	1	1957	1601TRA	117.000000
2011-10-215883003970360637440	1	1960	1801EST	100.000000
2014-02-144152603450360238080	1	1952	1601LUD	83.000000
2013-01-115469160301247024128	0	1959	1601DCO	112.000000
2009-01-27-1787421095273548032	1	1947	2401PKD	93.000000
2017-07-18-283170816278083488	1	1954	1601TP	78.000000
2014-02-20-1230614677113268992	1	1954	500269	81.000000
2010-03-16-540841777105317184	1	1960	1801EST	119.000000
2016-05-317633822930609668096	1	1956	500124	78.000000
2012-06-21-1921873807210128896	1	1953	1801EST	85.000000
2016-11-18-311216742844917312	1	1962	500972	104.000000
2011-12-016566472276843258880	1	1958	1801EST	87.618421
2011-03-103783613273261840896	1	1948	2401DC	87.618421
2009-07-30-7184619100629243904	0	1952	1601DCO	111.000000
2014-03-073816239996082851840	1	1960	500355	76.000000
2015-07-091075829897044720000	1	1957	500356	73.000000
2012-08-098936423932091164672	0	1945	500746	74.000000
2013-03-073523415482885629952	1	1946	1601TBP	116.000000
2016-07-05-3828694392371445760	0	1942	2401DC	87.618421
2012-04-12-5275502836970302464	1	1951	1601DCO	76.000000

	P ParaulaPT	C ColorPD	C ColorPT
	PT	PD	PT
2013-12-19-3199504984151427072	50.000000	56.000000	37.000000
2014-02-25-5076076858799773696	50.000000	65.000000	40.000000
2013-07-042048871307966173952	48.000000	63.000000	39.000000
2013-10-086383101741557695488	51.000000	85.000000	53.000000
2014-07-081733670778237366016	41.000000	53.000000	32.000000
2011-11-094148544000739942912	41.634868	58.205298	37.047841
2017-04-184748903970519549952	41.634868	58.205298	37.047841
2016-01-08-6056684102447908864	48.000000	55.000000	36.000000
2014-01-217632868070695424000	35.000000	62.000000	38.000000
2011-05-18-3778282014351102976	42.000000	49.000000	28.000000
2013-12-24-1139302521462558976	38.000000	81.000000	50.000000
2010-02-05-1477311665734011904	41.634868	58.205298	37.047841
2015-10-02-2953031948695728128	50.000000	61.000000	30.000000
2013-05-283082888525163273216	57.000000	93.000000	59.000000
2016-11-222717391739930320896	19.000000	29.000000	19.000000
2016-11-177740231173833011200	41.000000	57.000000	37.000000
2009-10-286180148857292527616	48.000000	64.000000	38.000000
2011-05-12-3778282014351102976	41.634868	58.205298	37.047841
2014-10-175604819893483879424	41.634868	58.205298	37.047841
2011-07-08-2543111189480854016	40.000000	63.000000	40.000000
2015-04-30128771044112546400	55.000000	74.000000	46.000000
2013-11-12-1937500370690820096	41.634868	58.205298	37.047841
2012-05-242679390329982595072	33.000000	63.000000	39.000000
2011-06-21-4034696200536380928	40.000000	68.000000	42.000000
2017-01-195063911599998003200	59.000000	91.000000	58.000000
2014-10-283013508979533191168	41.634868	58.205298	37.047841
2012-12-11-1667285258272548096	30.000000	48.000000	29.000000
2017-12-15-988477930897196288	54.000000	85.000000	53.000000
2015-09-248685386897285894144	45.000000	60.000000	39.000000
2014-10-28-6453551274044865536	58.000000	70.000000	44.000000
...	...	...	...
2009-12-098466316493670267904	28.000000	74.000000	46.000000
2016-10-25-7521791065392616448	49.000000	36.000000	21.000000
2009-02-204857606362581025792	20.000000	24.000000	20.000000
2014-12-16-1921873807210128896	29.000000	46.000000	27.000000
2015-11-17-5615828347483806720	55.000000	61.000000	40.000000
2014-01-13-7824983083899136000	26.000000	47.000000	28.000000
2009-03-105351069396620515328	46.000000	49.000000	29.400000
2016-04-01251684784402089408	53.000000	52.000000	38.000000
2012-03-16-7832625167647364096	56.000000	77.000000	48.000000
2009-03-06-9014219213997190144	38.000000	69.000000	42.000000
2012-02-212048871307966173952	54.000000	80.000000	50.000000
2011-10-215883003970360637440	46.000000	60.000000	37.000000
2014-02-144152603450360238080	38.000000	54.000000	34.000000
2013-01-115469160301247024128	52.000000	86.000000	54.000000

2009-01-27-1787421095273548032	42.000000	57.000000	34.000000
2017-07-18-283170816278083488	39.000000	54.000000	35.000000
2014-02-20-1230614677113268992	42.000000	51.000000	34.000000
2010-03-16-540841777105317184	54.000000	60.000000	36.000000
2016-05-317633822930609668096	40.000000	61.000000	40.000000
2012-06-21-1921873807210128896	39.000000	60.000000	37.000000
2016-11-18-311216742844917312	52.000000	52.000000	34.000000
2011-12-016566472276843258880	41.634868	58.205298	37.047841
2011-03-103783613273261840896	41.634868	58.205298	37.047841
2009-07-30-7184619100629243904	52.000000	68.000000	42.000000
2014-03-073816239996082851840	34.000000	49.000000	30.000000
2015-07-091075829897044720000	36.000000	60.000000	40.000000
2012-08-098936423932091164672	33.000000	56.000000	34.000000
2013-03-073523415482885629952	54.000000	67.000000	42.000000
2016-07-05-3828694392371445760	41.634868	58.205298	37.047841
2012-04-12-5275502836970302464	30.000000	48.000000	29.000000

	PC ParaulaColorPD PD	PC ParaulaColorPT PT
2013-12-19-3199504984151427072	30.000000	40.000000
2014-02-25-5076076858799773696	29.000000	34.000000
2013-07-042048871307966173952	32.000000	37.000000
2013-10-086383101741557695488	43.000000	48.000000
2014-07-081733670778237366016	36.000000	20.000000
2011-11-094148544000739942912	34.286667	40.344482
2017-04-184748903970519549952	34.286667	40.344482
2016-01-08-6056684102447908864	32.000000	42.000000
2014-01-217632868070695424000	38.000000	43.000000
2011-05-18-3778282014351102976	34.000000	39.000000
2013-12-24-1139302521462558976	40.000000	46.000000
2010-02-05-1477311665734011904	34.286667	40.344482
2015-10-02-2953031948695728128	33.000000	42.000000
2013-05-283082888525163273216	61.000000	66.000000
2016-11-222717391739930320896	14.000000	19.000000
2016-11-177740231173833011200	41.000000	46.000000
2009-10-286180148857292527616	37.000000	42.000000
2011-05-12-3778282014351102976	34.286667	40.344482
2014-10-175604819893483879424	34.286667	40.344482
2011-07-08-2543111189480854016	50.000000	54.000000
2015-04-30128771044112546400	43.000000	48.000000
2013-11-12-1937500370690820096	34.286667	40.344482
2012-05-242679390329982595072	41.000000	46.000000
2011-06-21-4034696200536380928	42.000000	46.000000
2017-01-195063911599998003200	42.000000	47.000000
2014-10-283013508979533191168	34.286667	40.344482
2012-12-11-1667285258272548096	25.000000	30.000000
2017-12-15-988477930897196288	48.000000	53.000000

2015-09-248685386897285894144	35.000000	45.000000
2014-10-28-6453551274044865536	58.000000	62.000000
...	...	...
2009-12-098466316493670267904	28.000000	32.000000
2016-10-25-752179106539261644 8	63.000000	68.000000
2009-02-204857606362581025792	19.000000	24.000000
2014-12-16-1921873807210128896	25.000000	30.000000
2015-11-17-5615828347483806720	36.000000	46.000000
2014-01-13-7824983083899136000	17.000000	22.000000
2009-03-105351069396620515328	47.000000	52.000000
2016-04-01251684784402089408	28.000000	33.000000
2012-03-16-7832625167647364096	38.000000	43.000000
2009-03-06-9014219213997190144	29.000000	34.000000
2012-02-212048871307966173952	29.000000	34.000000
2011-10-215883003970360637440	31.000000	36.000000
2014-02-144152603450360238080	41.000000	46.000000
2013-01-115469160301247024128	52.000000	57.000000
2009-01-27-1787421095273548032	41.000000	46.000000
2017-07-18-283170816278083488	27.000000	37.000000
2014-02-20-1230614677113268992	27.000000	36.000000
2010-03-16-540841777105317184	47.000000	52.000000
2016-05-317633822930609668096	38.000000	48.000000
2012-06-21-1921873807210128896	33.000000	38.000000
2016-11-18-311216742844917312	40.000000	50.000000
2011-12-016566472276843258880	34.286667	40.344482
2011-03-103783613273261840896	34.286667	40.344482
2009-07-30-7184619100629243904	46.000000	51.000000
2014-03-073816239996082851840	27.000000	32.000000
2015-07-091075829897044720000	37.000000	47.000000
2012-08-098936423932091164672	45.000000	50.000000
2013-03-073523415482885629952	46.000000	52.000000
2016-07-05-3828694392371445760	34.286667	40.344482
2012-04-12-5275502836970302464	32.000000	37.000000

	Visual: Clave de números - Codificación (WAIS-III) \
	Correctes
	Correctes
2013-12-19-3199504984151427072	61.0
2014-02-25-5076076858799773696	39.0
2013-07-042048871307966173952	53.0
2013-10-086383101741557695488	85.0
2014-07-081733670778237366016	23.0
2011-11-094148544000739942912	57.0
2017-04-184748903970519549952	17.0
2016-01-08-6056684102447908864	28.0
2014-01-217632868070695424000	20.0
2011-05-18-3778282014351102976	33.0
2013-12-24-1139302521462558976	82.0

2010-02-05-1477311665734011904	9.0
2015-10-02-2953031948695728128	56.0
2013-05-283082888525163273216	91.0
2016-11-222717391739930320896	38.0
2016-11-177740231173833011200	37.0
2009-10-286180148857292527616	59.0
2011-05-12-3778282014351102976	33.0
2014-10-175604819893483879424	36.0
2011-07-08-2543111189480854016	57.0
2015-04-30128771044112546400	59.0
2013-11-12-1937500370690820096	27.0
2012-05-242679390329982595072	30.0
2011-06-21-4034696200536380928	39.0
2017-01-195063911599998003200	84.0
2014-10-283013508979533191168	45.0
2012-12-11-1667285258272548096	24.0
2017-12-15-988477930897196288	63.0
2015-09-248685386897285894144	93.0
2014-10-28-6453551274044865536	71.0
...	...
2009-12-098466316493670267904	49.0
2016-10-25-7521791065392616448	73.0
2009-02-204857606362581025792	23.0
2014-12-16-1921873807210128896	39.0
2015-11-17-5615828347483806720	33.0
2014-01-13-7824983083899136000	41.0
2009-03-105351069396620515328	59.0
2016-04-01251684784402089408	48.0
2012-03-16-7832625167647364096	47.0
2009-03-06-9014219213997190144	47.0
2012-02-212048871307966173952	29.0
2011-10-215883003970360637440	48.0
2014-02-144152603450360238080	48.0
2013-01-115469160301247024128	61.0
2009-01-27-1787421095273548032	49.0
2017-07-18-283170816278083488	32.0
2014-02-20-1230614677113268992	38.0
2010-03-16-540841777105317184	63.0
2016-05-317633822930609668096	27.0
2012-06-21-1921873807210128896	46.0
2016-11-18-311216742844917312	32.0
2011-12-016566472276843258880	42.0
2011-03-103783613273261840896	43.0
2009-07-30-7184619100629243904	65.0
2014-03-073816239996082851840	28.0
2015-07-091075829897044720000	59.0
2012-08-098936423932091164672	24.0
2013-03-073523415482885629952	42.0

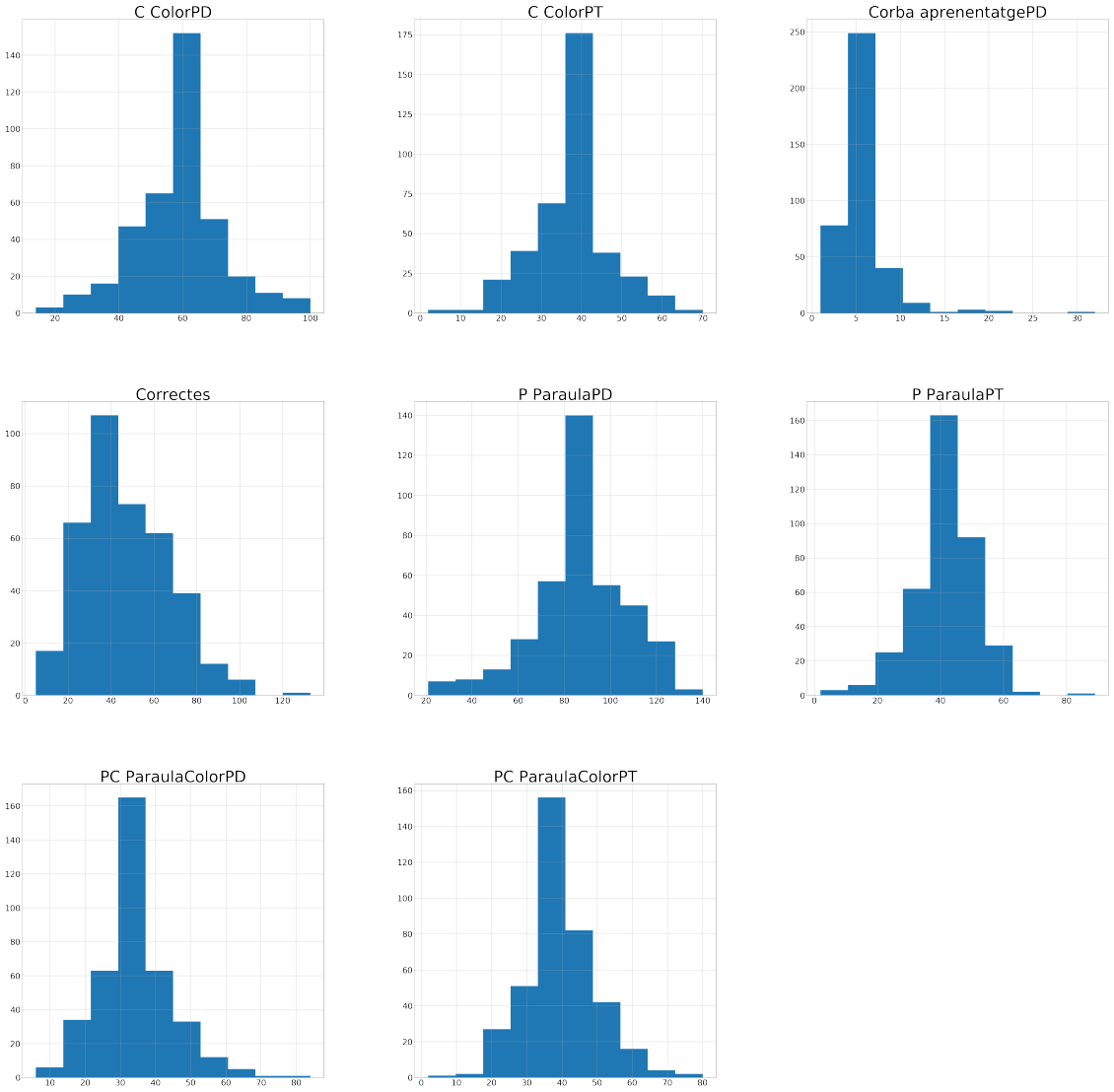
2016-07-05-3828694392371445760	16.0
2012-04-12-5275502836970302464	35.0

	Lista de palabras del RBANS_verbal Corba aprenentatgePD PD
2013-12-19-3199504984151427072	7.00
2014-02-25-5076076858799773696	4.00
2013-07-042048871307966173952	4.00
2013-10-086383101741557695488	2.00
2014-07-081733670778237366016	6.00
2011-11-094148544000739942912	6.00
2017-04-184748903970519549952	17.00
2016-01-08-6056684102447908864	4.00
2014-01-217632868070695424000	6.09
2011-05-18-3778282014351102976	6.09
2013-12-24-1139302521462558976	6.09
2010-02-05-1477311665734011904	6.09
2015-10-02-2953031948695728128	6.00
2013-05-283082888525163273216	10.00
2016-11-222717391739930320896	5.00
2016-11-177740231173833011200	7.00
2009-10-286180148857292527616	6.00
2011-05-12-3778282014351102976	5.00
2014-10-175604819893483879424	5.00
2011-07-08-2543111189480854016	7.00
2015-04-30128771044112546400	7.00
2013-11-12-1937500370690820096	6.09
2012-05-242679390329982595072	6.09
2011-06-21-4034696200536380928	4.00
2017-01-195063911599998003200	3.00
2014-10-283013508979533191168	4.00
2012-12-11-1667285258272548096	6.00
2017-12-15-988477930897196288	4.00
2015-09-248685386897285894144	6.00
2014-10-28-6453551274044865536	6.00
...	...
2009-12-098466316493670267904	5.00
2016-10-25-7521791065392616448	4.00
2009-02-204857606362581025792	3.00
2014-12-16-1921873807210128896	6.09
2015-11-17-5615828347483806720	3.00
2014-01-13-7824983083899136000	6.09
2009-03-105351069396620515328	6.00
2016-04-01251684784402089408	8.00
2012-03-16-7832625167647364096	6.00
2009-03-06-9014219213997190144	6.00
2012-02-212048871307966173952	3.00

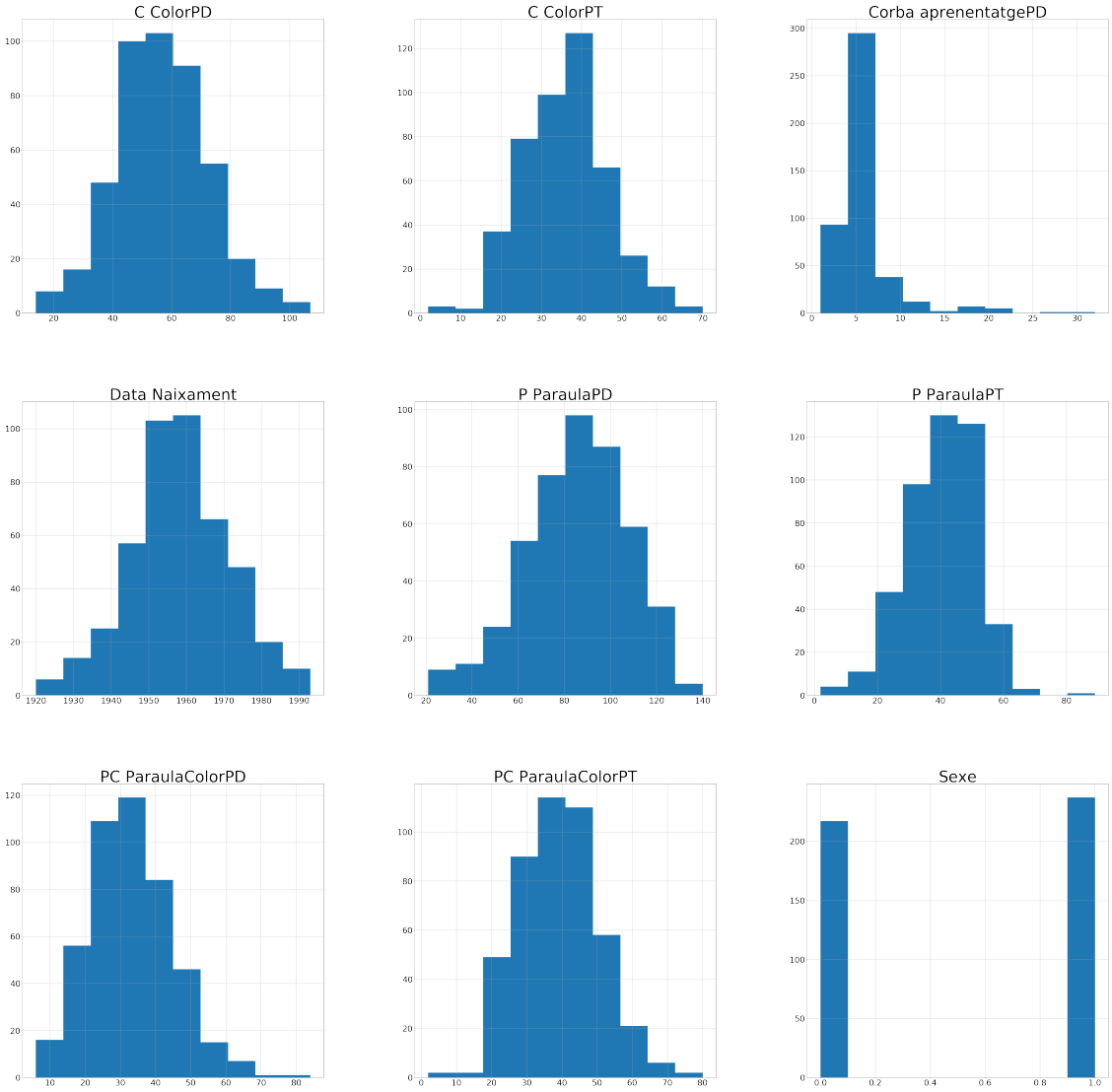
```
2011-10-215883003970360637440                                    11.00
2014-02-144152603450360238080                                    11.00
2013-01-115469160301247024128                                     7.00
2009-01-27-1787421095273548032                                   13.00
2017-07-18-283170816278083488                                     4.00
2014-02-20-1230614677113268992                                    8.00
2010-03-16-540841777105317184                                     4.00
2016-05-317633822930609668096                                     4.00
2012-06-21-1921873807210128896                                    5.00
2016-11-18-311216742844917312                                     8.00
2011-12-016566472276843258880                                     6.09
2011-03-103783613273261840896                                     6.09
2009-07-30-7184619100629243904                                   11.00
2014-03-073816239996082851840                                     1.00
2015-07-091075829897044720000                                     7.00
2012-08-098936423932091164672                                     6.09
2013-03-073523415482885629952                                     4.00
2016-07-05-3828694392371445760                                    5.00
2012-04-12-5275502836970302464                                    6.09

[383 rows x 11 columns]
```

In [6]:
```
df = df1.copy()
columns = df.iloc[:,3:].columns
df = pd.DataFrame(df, columns=columns)
df

df_copy = df.copy()

df_copy.columns = df_copy.columns.get_level_values(1)
params = {'axes.titlesize':'64',
          'xtick.labelsize':'34',
          'ytick.labelsize':'34'}

matplotlib.rcParams.update(params)
df_copy.hist(figsize=(80,80))
plt.savefig('../Figures/histogram description second subset', bbox_inches='tight')
np.std(df_copy)
```

Out[6]:
```
P ParaulaPD             20.151239
P ParaulaPT             10.101634
C ColorPD               13.823701
C ColorPT                9.032444
PC ParaulaColorPD       10.541890
PC ParaulaColorPT       10.230186
Correctes               20.030549
Corba aprenentatgePD     2.779339
dtype: float64
```

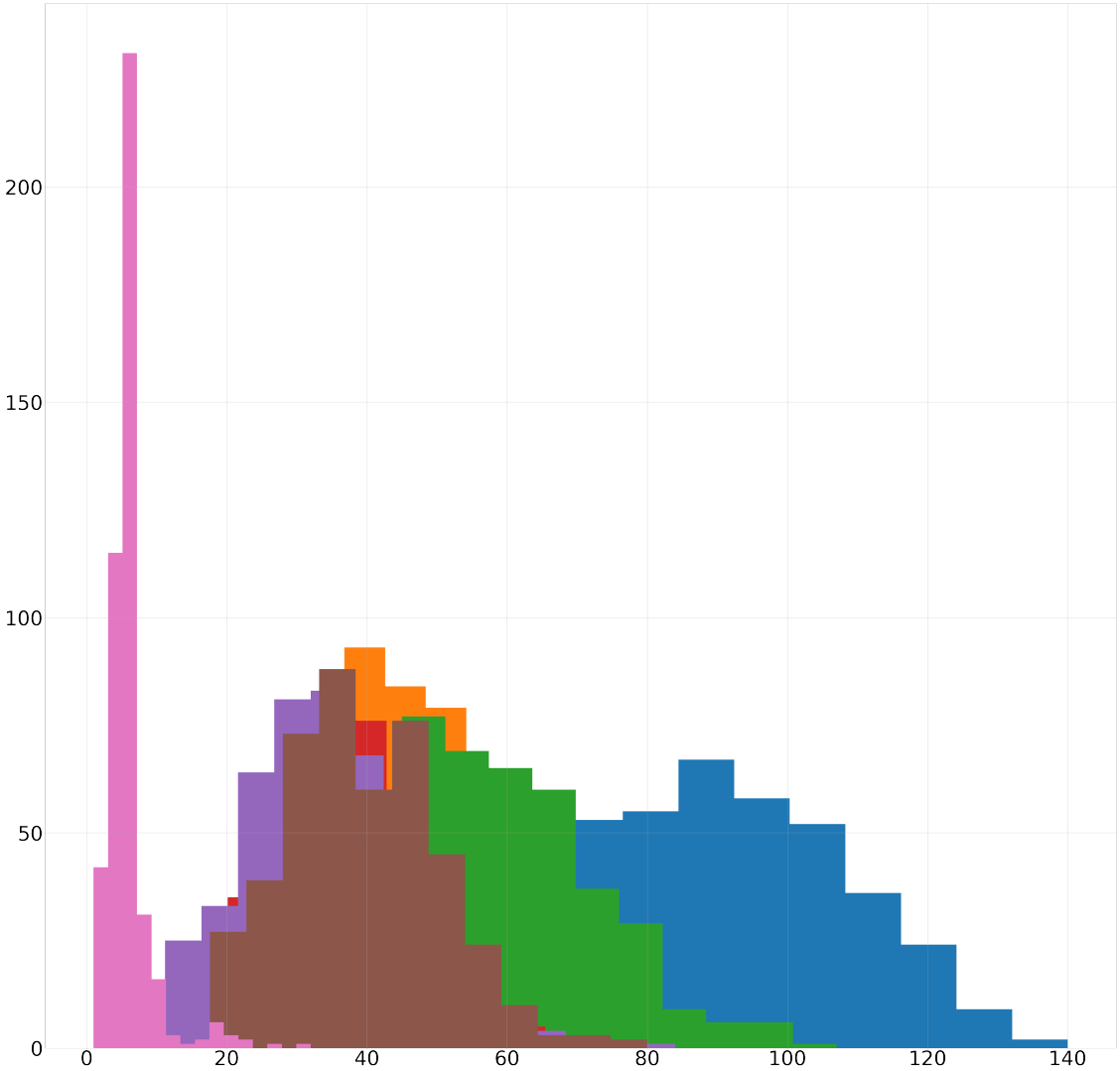



```
In [7]: df_mean = df_mean_array[2].copy()
        df_median = df_median_array[2].copy()
        df_most_frequent = df_most_frequent_array[2].copy() # Hacerlo con todos y a ver que sale

In [8]: df_copy = df_mean.copy()

        df_copy.columns = df_copy.columns.get_level_values(1)
        params = {'axes.titlesize':'64',
                  'xtick.labelsize':'34',
                  'ytick.labelsize':'34'}

        matplotlib.rcParams.update(params)
        df_copy.hist(figsize=(80,80))
        plt.savefig('../Figures/histogram description second subset', bbox_inches='tight')
```




```
In [9]: """df_mean.drop(('Dígitos: inversos (WAIS)', 'Digits inversosWaisPD', 'PD'), axis=1, inp
        df_mean.drop(('Dígitos: directos (WAIS)', 'Digits directesWaisPD', 'PD'), axis=1, inplac

        df_median.drop(('Dígitos: inversos (WAIS)', 'Digits inversosWaisPD', 'PD'), axis=1, inpl
        df_median.drop(('Dígitos: directos (WAIS)', 'Digits directesWaisPD', 'PD'), axis=1, inpl

        df_most_frequent.drop(('Dígitos: inversos (WAIS)', 'Digits inversosWaisPD', 'PD'), axis=
        df_most_frequent.drop(('Dígitos: directos (WAIS)', 'Digits directesWaisPD', 'PD'), axis=

Out[9]: "df_mean.drop(('Dígitos: inversos (WAIS)', 'Digits inversosWaisPD', 'PD'), axis=1, inpla

In [10]: df = df_mean.copy()
         params = {'axes.titlesize':'64',
                   'xtick.labelsize':'84',
```

```
          'ytick.labelsize':'84'}
matplotlib.rcParams.update(params)

for column in df.iloc[:,3:].columns:
    df[column].hist(bins=15,figsize=(80,80))

plt.savefig('../Figures/Comparison unified database subset 2.png',bbox_inches='tight')
plt.show()
```




```
In [11]: """mode = ['mean', 'median', 'most_frequent']

         df_mean = data_frame.copy()
         imp = Imputer(missing_values=np.nan, strategy=mode[1])
         df_mean.iloc[:,3:] = imp.fit_transform(df_mean.iloc[:,3:])
```

```
        df_median = data_frame.copy()
        imp = Imputer(missing_values=np.nan, strategy=mode[1])
        df_median.iloc[:,3:] = imp.fit_transform(df_median.iloc[:,3:])

        df_most_frequent = data_frame.copy()
        imp = Imputer(missing_values=np.nan, strategy=mode[2])
        df_most_frequent.iloc[:,3:] = imp.fit_transform(df_most_frequent.iloc[:,3:])"""
```

Out[11]: "mode = ['mean', 'median', 'most_frequent']\n\ndf_mean = data_frame.copy()\nimp = Imput

```
In [12]: # Creating object for storing information
         class relation_between_columns:
             """Clase base regresion lineal dos variables"""

             def __init__(self):
                 self.MAE = 0.0
                 self.RMSE = 0.0
                 self.MAPE = 0.0
                 self.weight = []
                 self.name_predicted_column = tuple()

             def __str__(self):
                 return str(self.__dict__)

In [13]: # Here we define a function for easy obtain
         def finding_patterns_between_columns(X, Y, name_predicted_column, model, relation):
             splitting_method = KFold(5)

             MAE_array = []
             RMSE_array = []
             MAPE_array = []
             coefs_array = []
             for train_index, test_index in splitting_method.split(X):
                 X_train, X_test = X.iloc[train_index, 3:], X.iloc[test_index, 3:]
                 Y_train, Y_test = Y.iloc[train_index], Y.iloc[test_index]
                 model.fit(X_train, Y_train)
                 Y_predicted = model.predict(X_test)
                 Y_test = np.array(Y_test)
                 N = len(Y_test)
                 # MAE
                 MAE = (1/N)*(np.sum(np.abs(Y_test - Y_predicted)))
                 MAE_array.append(MAE)
                 # RMSE
                 RMSE = np.sqrt((1/N)*(np.sum((Y_test - Y_predicted)**2)))
                 RMSE_array.append(RMSE)
                 # MAPE
                 MAPE_actual = []
```

```
            for i in range(len(Y_test)):
                if Y_test[i] != 0:
                    MAPE = np.abs(Y_test[i] - Y_predicted[i])/np.abs(Y_test[i])
                MAPE_actual.append(MAPE)
            MAPE_array.append(np.mean(MAPE))
            # Weight
            try:
                coef = model.coef_
                coefs_array.append(coef)
            except:
                pass

        relation.MAE = np.mean(MAE_array)
        relation.RMSE = np.mean(RMSE_array)
        relation.MAPE = np.mean(MAPE_array)
        relation.name_predicted_column = name_predicted_column

        try:
            coefficients = []
            for j in range(len(coefs_array[0])):
                coefficient = []
                for i in range(len(coefs_array)):
                    coefficient.append(coefs_array[i][j])
                coefficient = np.mean(coefficient)
                coefficients.append(coefficient)
            relation.weight = coefficients
        except:
            pass

        return relation
```

1.1 Triying to infer one column from the other ones:

We employ the previous function in order to predict the values from one column employing the other ones. We try with several algorithms and with several strategies of filling missing values.

1.2 Linear regression (OLS)

```
In [14]: """# We scale the values of the three different dataframes
         df = df_mean.copy()
         df_norm = normalize(df.iloc[:,3:], norm='l2')
         df.iloc[:,3:] = df_norm
         df_mean_norm = df

         df = df_median.copy()
         df_norm = normalize(df.iloc[:,3:], norm='l2')
         df.iloc[:,3:] = df_norm
         df_median_norm = df
```

```
        df = df_most_frequent.copy()
        df_norm = normalize(df.iloc[:,3:], norm='l2')
        df.iloc[:,3:] = df_norm
        df_most_frequent_norm = df"""
```

Out[14]: "# We scale the values of the three different dataframes \ndf = df_mean.copy()\ndf_norm

```
In [15]: # OLS - mean
         df = df_mean.copy()
         model = linear_model.LinearRegression(fit_intercept=True)
         relations = []
         for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
             Y = df[lvl0][lvl1][lvl2].copy()
             X = df.copy()
             X = X.drop(lvl0, level=0, axis=1)
             relation = relation_between_columns()
             name_predicted_column = (lvl0, lvl1, lvl2)
             relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
             relations.append(relation)

In [16]: # Visualizing correlations:
         writer = pd.ExcelWriter('../Summaries/LINEAR MODEL - MEAN: Relations.xlsx')
         row_to_write = 0
         for relation in relations:
             table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
             string = relation.name_predicted_column
             string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
             table['Predicted column'] = [string]
             table['MAE'] = relation.MAE
             table['RMSE'] = relation.RMSE
             table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
             row_to_write = row_to_write + 3

         writer.save()

In [17]: # OLS - median
         df = df_median.copy()
         model = linear_model.LinearRegression(fit_intercept=True)
         relations = []
         for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
             Y = df[lvl0][lvl1][lvl2].copy()
             X = df.copy()
             X = X.drop(lvl0, level=0, axis=1)
             relation = relation_between_columns()
             name_predicted_column = (lvl0, lvl1, lvl2)
             relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
             relations.append(relation)
```

In [18]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/LINEAR MODEL - MEDIAN: Relations.xlsx')
row_to_write = 0
for relation in relations:
    table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
    string = relation.name_predicted_column
    string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
    table['Predicted column'] = [string]
    table['MAE'] = relation.MAE
    table['RMSE'] = relation.RMSE
    table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
    row_to_write = row_to_write + 3

writer.save()
```

In [19]:
```
# OLS - most frequent
df = df_most_frequent.copy()
model = linear_model.LinearRegression(fit_intercept=True)
relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop(lvl0, level=0, axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [20]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/LINEAR MODEL - MOST FREQUENT: Relations.xlsx')
row_to_write = 0
for relation in relations:
    table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
    string = relation.name_predicted_column
    string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
    table['Predicted column'] = [string]
    table['MAE'] = relation.MAE
    table['RMSE'] = relation.RMSE
    table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
    row_to_write = row_to_write + 3

writer.save()
```

1.3 Linear regression (LASSO)

In [21]:
```
"""# We scale the values of the three different dataframes
scaler=StandardScaler()
```

```
        df = df_mean.copy()
        df_scaled = scaler.fit_transform(df.iloc[:,3:])
        df.iloc[:,3:] = df_scaled
        df_mean_scaled = df

        df = df_median.copy()
        df_scaled = scaler.fit_transform(df.iloc[:,3:])
        df.iloc[:,3:] = df_scaled
        df_median_scaled = df

        df = df_most_frequent.copy()
        df_scaled = scaler.fit_transform(df.iloc[:,3:])
        df.iloc[:,3:] = df_scaled
        df_most_frequent_scaled = df"""
```

```
Out[21]: '# We scale the values of the three different dataframes \nscaler=StandardScaler()\n\nd
```

```
In [64]: # Choose value of lambda
         df = df_mean.copy()
         lambdas = np.linspace(0,8,18)

         MAE_scores_lasso = []
         for n in lambdas:
             MAE_scores = []
             model = linear_model.Lasso(alpha=n)
             for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
                 Y = df[lvl0][lvl1][lvl2].copy()
                 X = df.copy()
                 X = X.drop(lvl0, level=0, axis=1)
                 relation = relation_between_columns()
                 name_predicted_column = (lvl0, lvl1, lvl2)
                 relation = finding_patterns_between_columns(X, Y, name_predicted_column, model,
                 MAE_scores.append(relation.MAE)
             MAE_scores = np.mean(MAE_scores)
             MAE_scores_lasso.append((n, MAE_scores))

         MAE_scores_lasso
```

```
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:12: UserWarning: With alpha=0,
  if sys.path[0] == '':
/home/dani/ENV/lib/python3.5/site-packages/sklearn/linear_model/coordinate_descent.py:477: UserW
  positive)
/home/dani/ENV/lib/python3.5/site-packages/sklearn/linear_model/coordinate_descent.py:491: Conve
  ConvergenceWarning)
```

```
Out[64]: [(0.0, 9.64770520959651),
          (0.47058823529411764, 9.638294607737492),
          (0.9411764705882353, 9.632337754291822),
```

```
 (1.4117647058823528, 9.629369684643109),
 (1.8823529411764706, 9.628723682999546),
 (2.3529411764705883, 9.628733120753965),
 (2.8235294117647056, 9.629657696000706),
 (3.2941176470588234, 9.630184650639787),
 (3.764705882352941, 9.630123037382944),
 (4.235294117647059, 9.630074002450892),
 (4.705882352941177, 9.629871229509288),
 (5.176470588235294, 9.629620319778152),
 (5.647058823529411, 9.629415113806951),
 (6.117647058823529, 9.629401084299625),
 (6.588235294117647, 9.629393424744228),
 (7.0588235294117645, 9.629393424744228),
 (7.529411764705882, 9.629393424744228),
 (8.0, 9.629393424744228)]
```

In [65]:
```
# LASSO - mean
df = df_mean.copy()
model = linear_model.Lasso(alpha=1.8)
relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop(lvl0, level=0, axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [66]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/LASSO - MEAN: Relations.xlsx')
row_to_write = 0
for relation in relations:
    table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
    string = relation.name_predicted_column
    string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
    table['Predicted column'] = [string]
    table['MAE'] = relation.MAE
    table['RMSE'] = relation.RMSE
    table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
    row_to_write = row_to_write + 3

writer.save()
```

In [67]:
```
# LASSO - median
df = df_median.copy()
model = linear_model.Lasso(alpha=1.8)
relations = []
```

```
        for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
            Y = df[lvl0][lvl1][lvl2].copy()
            X = df.copy()
            X = X.drop(lvl0, level=0, axis=1)
            relation = relation_between_columns()
            name_predicted_column = (lvl0, lvl1, lvl2)
            relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
            relations.append(relation)
```

In [68]:
```
        # Visualizing correlations:
        writer = pd.ExcelWriter('../Summaries/LASSO - MEDIAN: Relations.xlsx')
        row_to_write = 0
        for relation in relations:
            table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
            string = relation.name_predicted_column
            string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
            table['Predicted column'] = [string]
            table['MAE'] = relation.MAE
            table['RMSE'] = relation.RMSE
            table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
            row_to_write = row_to_write + 3

        writer.save()
```

In [69]:
```
        # LASSO - most frequent
        df = df_most_frequent.copy()
        model = linear_model.Lasso(alpha=1.8)
        relations = []
        for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
            Y = df[lvl0][lvl1][lvl2].copy()
            X = df.copy()
            X = X.drop(lvl0, level=0, axis=1)
            relation = relation_between_columns()
            name_predicted_column = (lvl0, lvl1, lvl2)
            relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
            relations.append(relation)
```

In [70]:
```
        # Visualizing correlations:
        writer = pd.ExcelWriter('../Summaries/LASSO - MOST FREQUENT: Relations.xlsx')
        row_to_write = 0
        for relation in relations:
            table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
            string = relation.name_predicted_column
            string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
            table['Predicted column'] = [string]
            table['MAE'] = relation.MAE
            table['RMSE'] = relation.RMSE
            table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
```

```
            row_to_write = row_to_write + 3

        writer.save()
```

1.4 Support Vector Machine

```
In [29]: """# We scale the values of the three different dataframes
         scaler=StandardScaler()

         df = df_mean.copy()
         df_scaled = scaler.fit_transform(df.iloc[:,3:])
         df.iloc[:,3:] = df_scaled
         df_mean_scaled = df

         df = df_median.copy()
         df_scaled = scaler.fit_transform(df.iloc[:,3:])
         df.iloc[:,3:] = df_scaled
         df_median_scaled = df

         df = df_most_frequent.copy()
         df_scaled = scaler.fit_transform(df.iloc[:,3:])
         df.iloc[:,3:] = df_scaled
         df_most_frequent_scaled = df"""

Out[29]: '# We scale the values of the three different dataframes \nscaler=StandardScaler()\n\nd

In [30]: # SVM - mean
         df = df_mean.copy()
         model = svm.SVR()
         relations = []
         for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
             Y = df[lvl0][lvl1][lvl2].copy()
             X = df.copy()
             X = X.drop((lvl0,lvl1,lvl2), axis=1)
             relation = relation_between_columns()
             name_predicted_column = (lvl0, lvl1, lvl2)
             relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
             relations.append(relation)

In [31]: # Visualizing correlations:
         writer = pd.ExcelWriter('../Summaries/SVM - MEAN: Relations.xlsx')
         row_to_write = 0
         for relation in relations:
             table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
             string = relation.name_predicted_column
             string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
             table['Predicted column'] = [string]
             table['MAE'] = relation.MAE
             table['RMSE'] = relation.RMSE
```

```
            table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
            row_to_write = row_to_write + 3

        writer.save()
```

In [32]:
```
# SVM - median
df = df_median.copy()
model = svm.SVR()
relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop((lvl0,lvl1,lvl2), axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [33]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/SVM - MEDIAN: Relations.xlsx')
row_to_write = 0
for relation in relations:
    table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
    string = relation.name_predicted_column
    string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
    table['Predicted column'] = [string]
    table['MAE'] = relation.MAE
    table['RMSE'] = relation.RMSE
    table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
    row_to_write = row_to_write + 3

writer.save()
```

In [34]:
```
# SVM - most frequent
df = df_most_frequent.copy()
model = svm.SVR()
relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop((lvl0,lvl1,lvl2), axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [35]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/SVM - MOST FREQUENT: Relations.xlsx')
row_to_write = 0
```

```
        for relation in relations:
            table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
            string = relation.name_predicted_column
            string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
            table['Predicted column'] = [string]
            table['MAE'] = relation.MAE
            table['RMSE'] = relation.RMSE
            table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
            row_to_write = row_to_write + 3

        writer.save()
```

1.5 Nearest neighbors

In [36]:
```
"""# We scale the values of the three different dataframes
df = df_mean.copy()
df_norm = normalize(df.iloc[:,3:], norm='l2')
df.iloc[:,3:] = df_norm
df_mean_norm = df

df = df_median.copy()
df_norm = normalize(df.iloc[:,3:], norm='l2')
df.iloc[:,3:] = df_norm
df_median_norm = df

df = df_most_frequent.copy()
df_norm = normalize(df.iloc[:,3:], norm='l2')
df.iloc[:,3:] = df_norm
df_most_frequent_norm = df"""
```

Out[36]:
```
"# We scale the values of the three different dataframes \ndf = df_mean.copy()\ndf_norm
```

In [37]:
```
# Choose the number of neighbors
df = df_mean.copy()
n_neighbors = np.linspace(1,60,30)
weights = 'uniform'

MAE_scores_n_neighbors = []
for n in n_neighbors:
    MAE_scores = []
    model = neighbors.KNeighborsRegressor(int(n), weights=weights)
    for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
        Y = df[lvl0][lvl1][lvl2].copy()
        X = df.copy()
        X = X.drop(lvl0, level=0, axis=1)
        relation = relation_between_columns()
        name_predicted_column = (lvl0, lvl1, lvl2)
        relation = finding_patterns_between_columns(X, Y, name_predicted_column, model,
```

```
            MAE_scores.append(relation.MAE)
        MAE_scores = np.mean(MAE_scores)
        MAE_scores_n_neighbors.append((n, MAE_scores))

MAE_scores_n_neighbors # 40 Vecinos
```

```
Out[37]: [(1.0, 12.149982535386519),
 (3.0344827586206895, 10.631047064878349),
 (5.068965517241379, 10.575776596631197),
 (7.1034482758620685, 10.101590872681287),
 (9.137931034482758, 9.927767420287418),
 (11.172413793103448, 9.750210430519036),
 (13.206896551724137, 9.757063459375525),
 (15.241379310344826, 9.756523606454934),
 (17.275862068965516, 9.749123760163735),
 (19.310344827586206, 9.68048387705495),
 (21.344827586206897, 9.65069641651583),
 (23.379310344827584, 9.624389829051054),
 (25.413793103448274, 9.63490386657288),
 (27.448275862068964, 9.618393106109748),
 (29.48275862068965, 9.645716044315861),
 (31.51724137931034, 9.680271035055705),
 (33.55172413793103, 9.694422454188459),
 (35.58620689655172, 9.682600369250647),
 (37.62068965517241, 9.701069474637936),
 (39.6551724137931, 9.687922267925659),
 (41.689655172413794, 9.707122542992105),
 (43.72413793103448, 9.695886956017125),
 (45.75862068965517, 9.692845090423162),
 (47.79310344827586, 9.68464759601805),
 (49.82758620689655, 9.69311352979019),
 (51.86206896551724, 9.701377637646146),
 (53.89655172413793, 9.678548034298585),
 (55.93103448275862, 9.68278284044626),
 (57.9655172413793, 9.674831711910581),
 (60.0, 9.674642454887051)]
```

```
In [71]: # NN - mean
         df = df_mean.copy()
         n_neighbors = 27
         weights = 'uniform'
         model = neighbors.KNeighborsRegressor(n_neighbors, weights=weights)
         relations = []
         for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
             Y = df[lvl0][lvl1][lvl2].copy()
             X = df.copy()
             X = X.drop(lvl0, level=0, axis=1)
             relation = relation_between_columns()
```

```
            name_predicted_column = (lvl0, lvl1, lvl2)
            relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
            relations.append(relation)
```

In [72]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/NN - MEAN: Relations.xlsx')
row_to_write = 0
for relation in relations:
    table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
    string = relation.name_predicted_column
    string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
    table['Predicted column'] = [string]
    table['MAE'] = relation.MAE
    table['RMSE'] = relation.RMSE
    table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
    row_to_write = row_to_write + 3

writer.save()
```

In [73]:
```
# NN - median
df = df_median.copy()
n_neighbors = 27
weights = 'uniform'
model = neighbors.KNeighborsRegressor(n_neighbors, weights=weights)
relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop(lvl0, level=0, axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [74]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/NN - MEDIAN: Relations.xlsx')
row_to_write = 0
for relation in relations:
    table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
    string = relation.name_predicted_column
    string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
    table['Predicted column'] = [string]
    table['MAE'] = relation.MAE
    table['RMSE'] = relation.RMSE
    table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
    row_to_write = row_to_write + 3

writer.save()
```

```
In [75]: # NN - most frequent
         df = df_most_frequent.copy()
         n_neighbors = 27
         weights = 'uniform'
         model = neighbors.KNeighborsRegressor(n_neighbors, weights=weights)
         relations = []
         for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
             Y = df[lvl0][lvl1][lvl2].copy()
             X = df.copy()
             X = X.drop(lvl0, level=0, axis=1)
             relation = relation_between_columns()
             name_predicted_column = (lvl0, lvl1, lvl2)
             relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
             relations.append(relation)

In [76]: # Visualizing correlations:
         writer = pd.ExcelWriter('../Summaries/NN - MOST FREQUENT: Relations.xlsx')
         row_to_write = 0
         for relation in relations:
             table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
             string = relation.name_predicted_column
             string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
             table['Predicted column'] = [string]
             table['MAE'] = relation.MAE
             table['RMSE'] = relation.RMSE
             table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
             row_to_write = row_to_write + 3

         writer.save()
```

1.6 Decision trees

```
In [77]: # Choosing max depth
         df = df_mean.copy()
         max_depth = np.linspace(1,8,15)
         MAE_scores_decision_tree = []
         for n in max_depth:
             MAE_scores = []
             model = DecisionTreeRegressor(max_depth=n)
             for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
                 Y = df[lvl0][lvl1][lvl2].copy()
                 X = df.copy()
                 X = X.drop(lvl0, level=0, axis=1)
                 relation = relation_between_columns()
                 name_predicted_column = (lvl0, lvl1, lvl2)
                 relation = finding_patterns_between_columns(X, Y, name_predicted_column, model,
                 MAE_scores.append(relation.MAE)
             MAE_scores = np.mean(MAE_scores)
```

```
            MAE_scores_decision_tree.append((n, MAE_scores))

        MAE_scores_decision_tree
```

```
Out[77]: [(1.0, 9.620216531715945),
          (1.5, 9.620216531715945),
          (2.0, 9.575419735985063),
          (2.5, 9.575419735985063),
          (3.0, 9.617752147861378),
          (3.5, 9.617752147861378),
          (4.0, 9.692833565820127),
          (4.5, 9.692833565820127),
          (5.0, 9.746593832477327),
          (5.5, 9.751067950512276),
          (6.0, 9.800062802477438),
          (6.5, 9.79257577033165),
          (7.0, 9.807562036115163),
          (7.5, 9.796293293867192),
          (8.0, 9.830362168146424)]
```

```
In [78]: # DT - mean
         df = df_mean.copy()
         max_depth = 3
         model = DecisionTreeRegressor(max_depth=max_depth)
         relations = []
         for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
             Y = df[lvl0][lvl1][lvl2].copy()
             X = df.copy()
             X = X.drop(lvl0, level=0, axis=1)
             relation = relation_between_columns()
             name_predicted_column = (lvl0, lvl1, lvl2)
             relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
             relations.append(relation)
```

```
In [46]: # Visualizing correlations:
         writer = pd.ExcelWriter('../Summaries/DT - MEAN: Relations.xlsx')
         row_to_write = 0
         for relation in relations:
             table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
             string = relation.name_predicted_column
             string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
             table['Predicted column'] = [string]
             table['MAE'] = relation.MAE
             table['RMSE'] = relation.RMSE
             table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
             row_to_write = row_to_write + 3

         writer.save()
```

In [79]:
```
# DT - median
df = df_median.copy()
max_depth = 3
model = DecisionTreeRegressor(max_depth=max_depth)
relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop(lvl0, level=0, axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [80]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/DT - MEDIAN: Relations.xlsx')
row_to_write = 0
for relation in relations:
    table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
    string = relation.name_predicted_column
    string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
    table['Predicted column'] = [string]
    table['MAE'] = relation.MAE
    table['RMSE'] = relation.RMSE
    table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
    row_to_write = row_to_write + 3

writer.save()
```

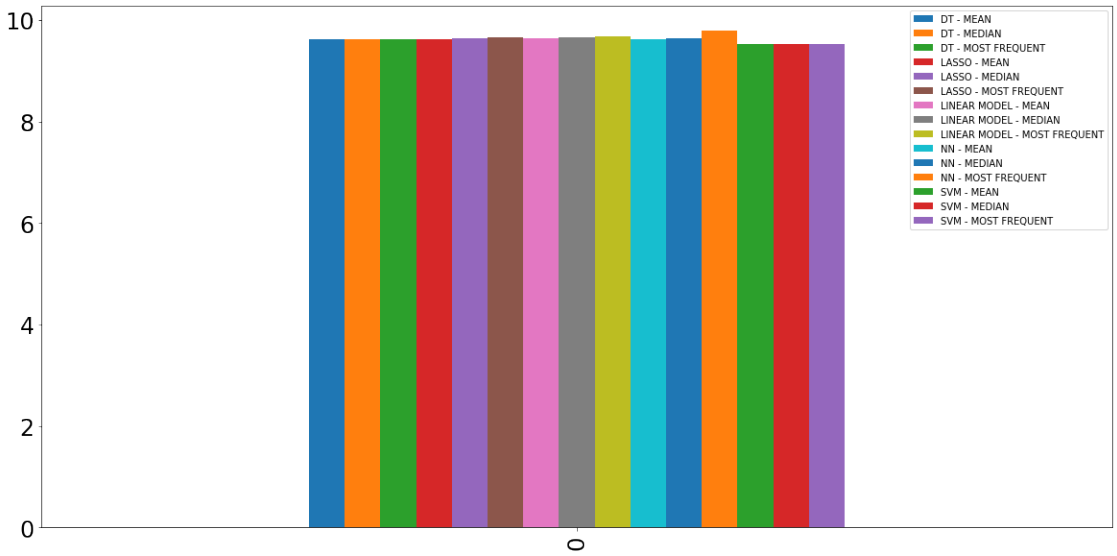
In [81]:
```
# DT - most frequent
df = df_most_frequent.copy()
max_depth = 3
model = DecisionTreeRegressor(max_depth=max_depth)
relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop(lvl0, level=0, axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [82]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/DT - MOST FREQUENT: Relations.xlsx')
row_to_write = 0
for relation in relations:
    table = pd.DataFrame(columns=['Predicted column', 'MAE', 'RMSE'])
```

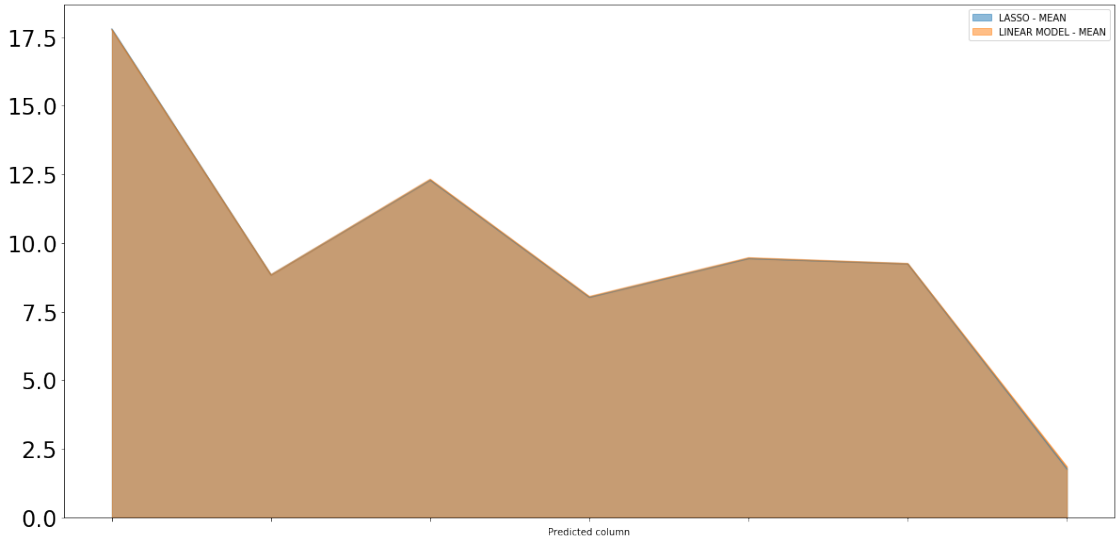
```
            string = relation.name_predicted_column
            string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
            table['Predicted column'] = [string]
            table['MAE'] = relation.MAE
            table['RMSE'] = relation.RMSE
            table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
            row_to_write = row_to_write + 3

        writer.save()
```

1.7 Comparing algorithms

```
In [84]: # Now we make a comparison between algorithms based in the mean values of MAE
         name_folders = ['LINEAR MODEL - MEAN: Relations.xlsx',
                         'LINEAR MODEL - MEDIAN: Relations.xlsx',
                         'LINEAR MODEL - MOST FREQUENT: Relations.xlsx',
                         'LASSO - MEAN: Relations.xlsx',
                         'LASSO - MEDIAN: Relations.xlsx',
                         'LASSO - MOST FREQUENT: Relations.xlsx',
                         'NN - MEAN: Relations.xlsx',
                         'NN - MEDIAN: Relations.xlsx',
                         'NN - MOST FREQUENT: Relations.xlsx',
                         'DT - MEAN: Relations.xlsx',
                         'DT - MEDIAN: Relations.xlsx',
                         'DT - MOST FREQUENT: Relations.xlsx',
                         'SVM - MEAN: Relations.xlsx',
                         'SVM - MEDIAN: Relations.xlsx',
                         'SVM - MOST FREQUENT: Relations.xlsx']

         name_MAE_dict = dict()
         for name in name_folders:
             df = pd.read_excel('../Summaries/' + name)
             df = df.set_index('Predicted column')
             df = df.drop('Predicted column')
             df = df.dropna()
             MAE = np.mean(df['MAE'])
             name_MAE_dict[name[:-16]] = MAE

         params = {'axes.titlesize':'64',
                   'xtick.labelsize':'24',
                   'ytick.labelsize':'24'}
         matplotlib.rcParams.update(params)

         df_hist = pd.DataFrame(name_MAE_dict, index=[0])
         df_hist.plot.bar(figsize=(20,10))
         plt.savefig('../Figures/histogram comparison subset 2.png')
         plt.show()
```



In [85]:
```
name_folders = ['LASSO - MEAN: Relations.xlsx',
                'LINEAR MODEL - MEAN: Relations.xlsx']

    name_MAE_dict = dict()
    for name in name_folders:
        df = pd.read_excel('../Summaries/' + name)
        df = df.set_index('Predicted column')
        df = df.drop('Predicted column')
        df = df.dropna()
        MAE = df['MAE']
        name_MAE_dict[name[:-16]] = MAE


    df_area = pd.DataFrame(name_MAE_dict)
    df_area.plot.area(figsize=(20,10), stacked=False, alpha=0.5)
    plt.savefig('../Figures/area comparison between LASSO and OLS second subset')
    plt.show()
```



1.8 Interpreting the model:

```
In [86]: # The best candidate is linear model - mean. As we are searching for relations between
         df = df_mean.copy()
         model = linear_model.LinearRegression(fit_intercept=True)
         relations = []
         for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
             Y = df[lvl0][lvl1][lvl2].copy()
             X = df.copy()
             X = X.drop(lvl0, level=0, axis=1)
             relation = relation_between_columns()
             name_predicted_column = (lvl0, lvl1, lvl2)
             relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
             relations.append(relation)

In [87]: df_coefs = df.copy()
         df_coefs = pd.DataFrame(columns=df_coefs.columns, index=[i for i in range(1,5)])
         df_coefs = df_coefs.drop(('Sexe', 'Sexe', 'Sexe'), axis=1)
         df_coefs = df_coefs.drop(('Data Naixament', 'Data Naixament', 'Data Naixament'), axis=1
         df_coefs = df_coefs.drop(('Diagnòstic', 'Diagnòstic', 'Diagnòstic'), axis=1)

         i = 0
         for rel in relations:
             i = i + 1
             j = 0
             df_coefs_rel = df_coefs.copy()
             predicted_column = rel.name_predicted_column
             columns = df_coefs_rel.columns.drop(predicted_column[0],level=0)
             for column in columns:
```

```
            df_coefs.loc[i,column] = rel.weight[j]
            df_coefs.loc[i,predicted_column] = 'X'
            j = j + 1

    df_coefs.insert(0, column=('RMSE','RMSE','RMSE'), value=0)
    df_coefs.insert(0, column=('MAPE','MAPE','MAPE'), value=0)
    df_coefs.insert(0, column=('MAE','MAE','MAE'), value=0)

    i = 0
    for rel in relations:
        i = i + 1
        df_coefs.loc[i,'MAE'] = rel.MAE
        df_coefs.loc[i,'MAPE'] = rel.MAPE*100
        df_coefs.loc[i,'RMSE'] = rel.RMSE
```

In [88]:
```
pd.set_option('display.height', 1000)
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

In [89]:
```
df_coefs = df_coefs.fillna('-')
df_coefs
```

Out[89]:

	MAE	MAPE	RMSE	Verbal: Stroop				
	MAE	MAPE	RMSE	P ParaulaPD	P ParaulaPT	C ColorPD	C ColorPT	PC
	MAE	MAPE	RMSE	PD	PT	PD	PT	
1	17.773250	12.851460	22.128573	X	-	-	-	
2	8.841375	19.866728	11.095293	-	X	-	-	
3	12.318762	23.941124	15.461740	-	-	X	-	
4	8.046701	26.249091	10.068588	-	-	-	X	
5	9.459578	14.946775	11.952479	-	-	-	-	
6	9.249146	17.101472	11.453738	-	-	-	-	
7	1.845123	20.650323	3.246784	-0.068054	0.102101	0.0593145	-0.0439693	

In [57]:
```
df = df_mean.copy()
model = linear_model.Lasso(alpha=200)
relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop(lvl0, level=0, axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [58]:
```
df_coefs = df.copy()
df_coefs = pd.DataFrame(columns=df_coefs.columns, index=[i for i in range(1,5)])
df_coefs = df_coefs.drop(('Sexe', 'Sexe', 'Sexe'), axis=1)
```

```
df_coefs = df_coefs.drop(('Data Naixament', 'Data Naixament', 'Data Naixament'), axis=1
df_coefs = df_coefs.drop(('Diagnòstic', 'Diagnòstic', 'Diagnòstic'), axis=1)

i = 0
for rel in relations:
    i = i + 1
    j = 0
    df_coefs_rel = df_coefs.copy()
    predicted_column = rel.name_predicted_column
    columns = df_coefs_rel.columns.drop(predicted_column[0],level=0)
    for column in columns:
        df_coefs.loc[i,column] = rel.weight[j]
        df_coefs.loc[i,predicted_column] = 'X'
        j = j + 1

df_coefs.insert(0, column=('RMSE','RMSE','RMSE'), value=0)
df_coefs.insert(0, column=('MAPE','MAPE','MAPE'), value=0)
df_coefs.insert(0, column=('MAE','MAE','MAE'), value=0)

i = 0
for rel in relations:
    i = i + 1
    df_coefs.loc[i,'MAE'] = rel.MAE
    df_coefs.loc[i,'MAPE'] = rel.MAPE*100
    df_coefs.loc[i,'RMSE'] = rel.RMSE
```

In [59]:
```
df_coefs = df_coefs.fillna('-')
df_coefs
```

Out[59]:

	MAE	MAPE	RMSE	Verbal: Stroop				
	MAE	MAPE	RMSE	P ParaulaPD	P ParaulaPT	C ColorPD	C ColorPT	PC Pa
	MAE	MAPE	RMSE	PD	PT	PD	PT	
1	17.836738	12.865835	22.134629	X	-	-	-	
2	8.837174	19.909011	11.076710	-	X	-	-	
3	12.293351	24.004664	15.442693	-	-	X	-	
4	8.023846	26.321582	10.056249	-	-	-	X	
5	9.430978	15.073246	11.922680	-	-	-	-	
6	9.240128	17.174473	11.441962	-	-	-	-	
7	1.743539	13.181972	3.203175	0	0	0	0	

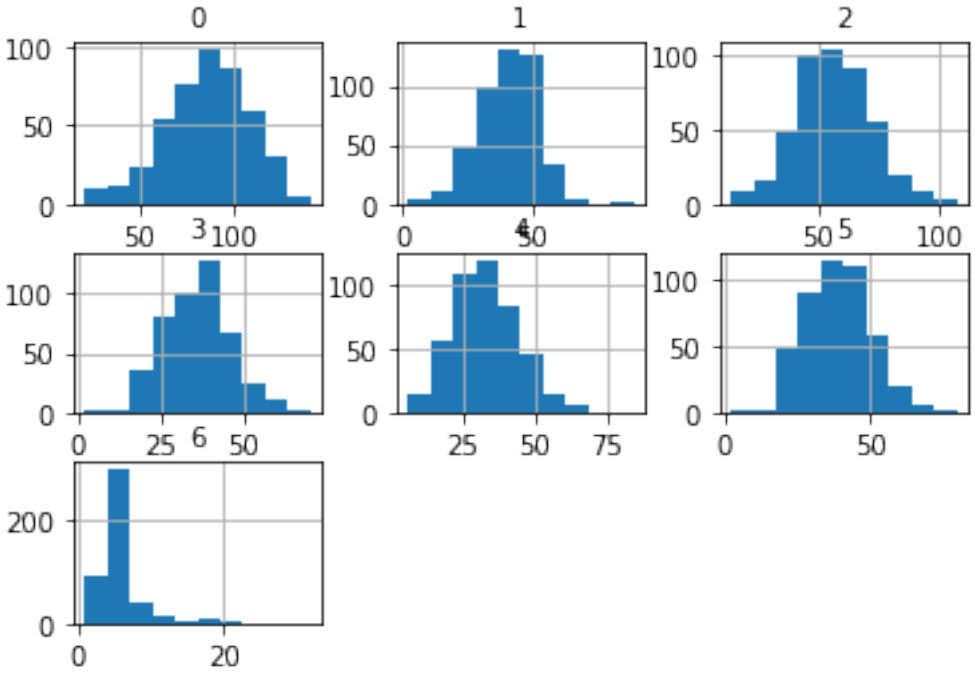
In [60]:
```
df.iloc[:,3:].describe().loc['max'] - df.iloc[:,3:].describe().loc['min']
```

Out[60]:

Verbal: Stroop		
P ParaulaPD	PD	119.0
P ParaulaPT	PT	87.0
C ColorPD	PD	93.0
C ColorPT	PT	68.0
PC ParaulaColorPD	PD	78.0
PC ParaulaColorPT	PT	78.0

```
Lista de palabras del RBANS_verbal  Corba aprenentatgePD  PD     31.0
dtype: float64
```

In [61]:
```
params = {'axes.titlesize':'10',
          'xtick.labelsize':'10',
          'ytick.labelsize':'10'}
matplotlib.rcParams.update(params)

df_hist = df.iloc[:,3:].copy()
df_hist.columns = [i for i in range(len(df_hist.columns))]
df_hist.hist()
plt.show()
```



In [62]:
```
# Representing the model:
params = {'axes.titlesize':'10',
          'xtick.labelsize':'14',
          'ytick.labelsize':'14'}
matplotlib.rcParams.update(params)

A = df.loc[:, ('Dígitos: inversos (WAIS)','Digits inversosWaisPcPe','Pe')].copy()
B = df.loc[:, ('Lista de palabras del RBANS_verbal','Corba aprenentatgePD','PD')].copy(
C = df.loc[:, ('Orientación de líneas RBANS','Orienta liniesPT','PT')].copy()
D = df.loc[:, ('Dígitos: directos (WAIS)','Digits directesWaisPcPe','Pe')].copy()

A_train, A_test = train_test_split(A, test_size=0.4)
```

```
B_train, B_test = train_test_split(B, test_size=0.4)
C_train, C_test = train_test_split(C, test_size=0.4)
D_train, D_test = train_test_split(D, test_size=0.4)

plt.scatter(A, D,  color='black')


plt.show()
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-62-816d0948fd07> in <module>()
  5 matplotlib.rcParams.update(params)
  6
----> 7 A = df.loc[:, ('Dígitos: inversos (WAIS)','Digits inversosWaisPcPe','Pe')].copy()
  8 B = df.loc[:, ('Lista de palabras del RBANS_verbal','Corba aprenentatgePD','PD')].co
  9 C = df.loc[:, ('Orientación de líneas RBANS','Orienta liniesPT','PT')].copy()


~/ENV/lib/python3.5/site-packages/pandas/core/indexing.py in __getitem__(self, key)
1365             except (KeyError, IndexError):
1366                 pass
-> 1367             return self._getitem_tuple(key)
1368         else:
1369             # we by definition only have the 0th axis


~/ENV/lib/python3.5/site-packages/pandas/core/indexing.py in _getitem_tuple(self, tup)
856     def _getitem_tuple(self, tup):
857         try:
--> 858             return self._getitem_lowerdim(tup)
859         except IndexingError:
860             pass


~/ENV/lib/python3.5/site-packages/pandas/core/indexing.py in _getitem_lowerdim(self, tup
970         # we may have a nested tuples indexer here
971         if self._is_nested_tuple_indexer(tup):
--> 972             return self._getitem_nested_tuple(tup)
973
974         # we maybe be using a tuple to represent multiple dimensions here


~/ENV/lib/python3.5/site-packages/pandas/core/indexing.py in _getitem_nested_tuple(self,
1049
```

```
   1050             current_ndim = obj.ndim
-> 1051             obj = getattr(obj, self.name)._getitem_axis(key, axis=axis)
   1052             axis += 1
   1053


    ~/ENV/lib/python3.5/site-packages/pandas/core/indexing.py in _getitem_axis(self, key, ax
   1625         # fall thru to straight lookup
   1626         self._has_valid_type(key, axis)
-> 1627         return self._get_label(key, axis=axis)
   1628
   1629


    ~/ENV/lib/python3.5/site-packages/pandas/core/indexing.py in _get_label(self, label, axi
    143             raise IndexingError('no slices here, handle elsewhere')
    144
--> 145         return self.obj._xs(label, axis=axis)
    146
    147     def _get_loc(self, key, axis=None):


    ~/ENV/lib/python3.5/site-packages/pandas/core/generic.py in xs(self, key, axis, level, d
   2333
   2334         if axis == 1:
-> 2335             return self[key]
   2336
   2337         self._consolidate_inplace()


    ~/ENV/lib/python3.5/site-packages/pandas/core/frame.py in __getitem__(self, key)
   2135             return self._getitem_frame(key)
   2136         elif is_mi_columns:
-> 2137             return self._getitem_multilevel(key)
   2138         else:
   2139             return self._getitem_column(key)


    ~/ENV/lib/python3.5/site-packages/pandas/core/frame.py in _getitem_multilevel(self, key)
   2179
   2180     def _getitem_multilevel(self, key):
-> 2181         loc = self.columns.get_loc(key)
   2182         if isinstance(loc, (slice, Series, np.ndarray, Index)):
   2183             new_columns = self.columns[loc]


    ~/ENV/lib/python3.5/site-packages/pandas/core/indexes/multi.py in get_loc(self, key, met
   2090             key = _values_from_object(key)
```

```
   2091             key = tuple(map(_maybe_str_to_time_stamp, key, self.levels))
-> 2092             return self._engine.get_loc(key)
   2093
   2094         # -- partial selection or non-unique index


    pandas/_libs/index.pyx in pandas._libs.index.MultiIndexObjectEngine.get_loc()


    pandas/_libs/index.pyx in pandas._libs.index.MultiIndexObjectEngine.get_loc()


    pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


    pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


    pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_


    pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_


    KeyError: ('Dígitos: inversos (WAIS)', 'Digits inversosWaisPcPe', 'Pe')
```

Data mining discrete case

September 3, 2018

1 Data mining discrete case:

We will implement a function that recibes a data base and returns a set of meassures for quantify the performance of the algorithm.

```
In [1]: # Import libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib
        import seaborn as sns
        import load_unified_data_discrete
        from sklearn.model_selection import KFold
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import classification_report
        from sklearn import linear_model
        from sklearn import svm
        from sklearn import neighbors
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.metrics import confusion_matrix

In [2]: data_frame_discrete, tuples_structure_discrete = load_unified_data_discrete.read_and_for

In [3]: mode_discrete = {'optimistic': 'min', 'pesimistic': 'max'}

        df_optimistic_array = []
        df_pesimistic_array = []

        df = data_frame_discrete.copy()
        columns = set(df.iloc[:,3:].columns.get_level_values(0))
        for column in columns:
            actual_column = df[column].copy().dropna()
            index = actual_column.index
            df_actual = df.loc[index]
            long = len(df_actual)
            info_actual_columns = []
            for column_actual in df_actual.iloc[:,3:].columns:
```

```
                percentage = df_actual[column_actual].describe().loc['count'] * 100 / long
                if percentage < 70.0:
                    del df_actual[column_actual]

            df_optimistic = df_actual.copy()
            for column in data_frame_discrete.iloc[:, 3:].columns:
                value = round(data_frame_discrete.describe().loc[mode_discrete['optimistic']][co
                df_optimistic[column] = data_frame_discrete[column].fillna(value)
            if len(set(df_optimistic.iloc[:,3:].columns.get_level_values(0))) > 1:
                df_optimistic_array.append(df_optimistic)

            df_pesimistic = df_actual.copy()
            for column in data_frame_discrete.iloc[:, 3:].columns:
                value = round(data_frame_discrete.describe().loc[mode_discrete['pesimistic']][co
                df_pesimistic[column] = data_frame_discrete[column].fillna(value)
            if len(set(df_pesimistic.iloc[:,3:].columns.get_level_values(0))) > 1:
                df_pesimistic_array.append(df_pesimistic)
```

```
In [4]: for df in df_optimistic_array:
            print(len(df.columns), len(df))
```

```
20 451
20 764
20 952
20 382
20 768
20 362
20 376
```

```
In [5]: for df in df_optimistic_array:
            print(df.columns.get_level_values(0))
```

```
Index(['Sexe', 'Data Naixament', 'Diagnòstic', 'Verbal: Stroop',
       'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop',
       'Verbal: Stroop', 'Lista de palabras del RBANS_verbal',
       'Visual: Clave de números - Codificación (WAIS-III)',
       'Dígitos: inversos (WAIS)', 'Dígitos: inversos (WAIS)',
       'TAPPING de McQuarrie', 'TAPPING de McQuarrie', 'TAPPING de McQuarrie',
       'TAPPING de McQuarrie', 'Orientación de líneas RBANS',
       'Dígitos: directos (WAIS)', 'Dígitos: directos (WAIS)'],
      dtype='object')
Index(['Sexe', 'Data Naixament', 'Diagnòstic', 'Dígitos: inversos (WAIS)',
       'Dígitos: inversos (WAIS)', 'Dígitos: directos (WAIS)',
       'Dígitos: directos (WAIS)', 'Verbal: Stroop', 'Verbal: Stroop',
       'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop',
       'Visual: Clave de números - Codificación (WAIS-III)',
       'TAPPING de McQuarrie', 'TAPPING de McQuarrie', 'TAPPING de McQuarrie',
       'TAPPING de McQuarrie', 'Lista de palabras del RBANS_verbal',
```

```
       'Orientación de líneas RBANS'],
      dtype='object')
Index(['Sexe', 'Data Naixament', 'Diagnòstic',
       'Lista de palabras del RBANS_verbal', 'Verbal: Stroop',
       'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop',
       'Verbal: Stroop', 'Visual: Clave de números - Codificación (WAIS-III)',
       'Dígitos: inversos (WAIS)', 'Dígitos: inversos (WAIS)',
       'TAPPING de McQuarrie', 'TAPPING de McQuarrie', 'TAPPING de McQuarrie',
       'TAPPING de McQuarrie', 'Orientación de líneas RBANS',
       'Dígitos: directos (WAIS)', 'Dígitos: directos (WAIS)'],
      dtype='object')
Index(['Sexe', 'Data Naixament', 'Diagnòstic', 'Verbal: Stroop',
       'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop',
       'Verbal: Stroop', 'Visual: Clave de números - Codificación (WAIS-III)',
       'Lista de palabras del RBANS_verbal', 'Dígitos: inversos (WAIS)',
       'Dígitos: inversos (WAIS)', 'TAPPING de McQuarrie',
       'TAPPING de McQuarrie', 'TAPPING de McQuarrie', 'TAPPING de McQuarrie',
       'Orientación de líneas RBANS', 'Dígitos: directos (WAIS)',
       'Dígitos: directos (WAIS)'],
      dtype='object')
Index(['Sexe', 'Data Naixament', 'Diagnòstic', 'Dígitos: inversos (WAIS)',
       'Dígitos: inversos (WAIS)', 'Dígitos: directos (WAIS)',
       'Dígitos: directos (WAIS)', 'Verbal: Stroop', 'Verbal: Stroop',
       'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop',
       'Visual: Clave de números - Codificación (WAIS-III)',
       'TAPPING de McQuarrie', 'TAPPING de McQuarrie', 'TAPPING de McQuarrie',
       'TAPPING de McQuarrie', 'Lista de palabras del RBANS_verbal',
       'Orientación de líneas RBANS'],
      dtype='object')
Index(['Sexe', 'Data Naixament', 'Diagnòstic', 'TAPPING de McQuarrie',
       'TAPPING de McQuarrie', 'TAPPING de McQuarrie', 'TAPPING de McQuarrie',
       'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop',
       'Verbal: Stroop', 'Verbal: Stroop',
       'Visual: Clave de números - Codificación (WAIS-III)',
       'Dígitos: inversos (WAIS)', 'Dígitos: inversos (WAIS)',
       'Lista de palabras del RBANS_verbal', 'Orientación de líneas RBANS',
       'Dígitos: directos (WAIS)', 'Dígitos: directos (WAIS)'],
      dtype='object')
Index(['Sexe', 'Data Naixament', 'Diagnòstic', 'Dígitos: inversos (WAIS)',
       'Dígitos: inversos (WAIS)', 'Lista de palabras del RBANS_verbal',
       'Orientación de líneas RBANS', 'Dígitos: directos (WAIS)',
       'Dígitos: directos (WAIS)', 'Verbal: Stroop', 'Verbal: Stroop',
       'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop', 'Verbal: Stroop',
       'Visual: Clave de números - Codificación (WAIS-III)',
       'TAPPING de McQuarrie', 'TAPPING de McQuarrie', 'TAPPING de McQuarrie',
       'TAPPING de McQuarrie'],
      dtype='object')
```

In [6]:
```
# Binary case
df_optimistic = df_optimistic_array[2].copy()

df_pesimistic = df_pesimistic_array[2].copy()

a = df_optimistic['Sexe']['Sexe']['Sexe'].apply(lambda x: 1 if x == 0 else 2)

b = df_pesimistic['Sexe']['Sexe']['Sexe'].apply(lambda x: 1 if x == 0 else 2)

df_optimistic.insert(4, column=('Sexe12','Sexe12','Sexe12'), value=a)
df_pesimistic.insert(4, column=('Sexe12','Sexe12','Sexe12'), value=b)
```

In [7]:
```
df_optimistic
```

Out[7]:

	Sexe	Data Naixament	Diagnòstic \
	Sexe	Data Naixament	Diagnòstic
	Sexe	Data Naixament	Diagnòstic
2012-02-07-3094248760095479808	0	1953	1601DAL
2008-10-23-4978555423559782400	0	1959	1601DCO
2017-05-18-4689198199369149440	0	1955	500303
2013-12-19-3199504984151427072	1	1959	1601TP
2014-02-25-5076076858799773696	1	1959	101EPI
2010-12-144650200895339849728	1	1944	3UR0171
2015-02-13-651871482488781824	1	1928	207EAC
2013-07-042048871307966173952	1	1957	1601TRA
2015-01-081524797290136683008	0	1952	1601DCO
2014-03-11-8591197668126770176	1	1922	501772
2010-01-143299776405939630080	1	1946	10106
2013-10-086383101741557695488	0	1964	L-PNE
2014-10-022178887206724697088	0	1977	3UR0133
2017-07-27-5987028499592949760	1	1944	1601DCO
2015-06-045809237815719999488	0	1930	500885
2014-03-13-3679307940707129856	0	1920	500657
2008-12-19-7933846893055049728	0	1942	NaN
2014-07-081733670778237366016	0	1946	3UR0108
2014-04-11-6874778161861040128	1	1976	2401DEC
2011-11-094148544000739942912	0	1967	1002DEP
2017-04-27-937487139953573376	0	1982	502108
2017-04-184748903970519549952	0	1960	1601TBP
2016-01-08-6056684102447908864	1	1964	500303
2011-02-24-1362401263924740096	1	1948	3UR0036
2009-05-295752104861709540352	1	1938	1601DCO
2015-04-166167587522835896320	0	1969	1601TDM
2015-10-02-2953031948695728128	0	1958	502149
2014-07-181402706660063879936	0	1935	500968
2011-08-26900106112460610560	1	1966	2401DV
2011-09-02785110356163871360	1	1958	1601TP
...	...	...	...

2011-05-17-5744439083839869952	1	1922	3UR0036
2014-07-312511218136851899904	0	1959	2401DC
2009-03-09-3998874973550231040	1	1938	1601DC0
2013-10-25-1937500370690820096	1	1984	500997
2010-03-16-540841777105317184	1	1960	1801EST
2008-11-27-8117622520237889536	0	1928	1708
2014-06-106032564288064509952	1	1935	332.0
2013-02-15-18620931705533700	0	1928	2401DC
2009-12-283134469428844723200	1	1934	1801EST
2009-07-30-7066130099458830336	0	1944	1601DC0
2016-05-317633822930609668096	1	1956	500124
2012-06-21-1921873807210128896	1	1953	1801EST
2016-11-18-311216742844917312	1	1962	500972
2009-07-29-8265931828440649728	1	1957	1601DC0
2009-03-315942456735192465408	1	1956	1601DC0
2009-07-30-7184619100629243904	0	1952	1601DC0
2012-12-102700869859969658880	1	1951	1801EST
2014-03-073816239996082851840	1	1960	500355
2015-07-091075829897044720000	1	1957	500356
2011-09-09-2820949354713164800	0	1963	3UR0017
2008-10-15282679175839416000	0	1956	1601DC0
2012-04-171106219260751149952	1	1930	1601DC0
2013-02-28368568132944748032	1	1933	1601DC0
2013-03-073523415482885629952	1	1946	1601TBP
2011-02-22-2420683185350238208	0	1947	1801EST
2009-03-13-6859066878463339520	1	1931	1601DC0
2013-04-16-4042990949616907776	0	1948	2401DC
2014-10-032178887206724697088	0	1977	3UR0133
2016-07-05-3828694392371445760	0	1942	2401DC
2009-04-14-9045844499657849856	0	1931	1601DC0

	Lista de palabras del RBANS_verbal	Sexe12	\
	Corba aprenentatgePD	Sexe12	
	PD	Sexe12	
2012-02-07-3094248760095479808	2.0	1	
2008-10-23-4978555423559782400	2.0	1	
2017-05-18-4689198199369149440	1.0	1	
2013-12-19-3199504984151427072	2.0	2	
2014-02-25-5076076858799773696	1.0	2	
2010-12-144650200895339849728	2.0	2	
2015-02-13-651871482488781824	1.0	2	
2013-07-042048871307966173952	1.0	2	
2015-01-081524797290136683008	1.0	1	
2014-03-11-8591197668126770176	2.0	2	
2010-01-143299776405939630080	2.0	2	
2013-10-086383101741557695488	1.0	1	
2014-10-022178887206724697088	2.0	1	
2017-07-27-5987028499592949760	2.0	2	

2015-06-045809237815719999488	2.0	1
2014-03-13-3679307940707129856	2.0	1
2008-12-19-7933846893055049728	1.0	1
2014-07-081733670778237366016	1.0	1
2014-04-11-6874778161861040128	2.0	2
2011-11-094148544000739942912	1.0	1
2017-04-27-937487139953573376	1.0	1
2017-04-184748903970519549952	2.0	1
2016-01-08-6056684102447908864	1.0	2
2011-02-24-1362401263924740096	2.0	2
2009-05-295752104861709540352	1.0	2
2015-04-166167587522835896320	1.0	1
2015-10-02-2953031948695728128	1.0	1
2014-07-181402706660063879936	1.0	1
2011-08-26900106112460610560	1.0	2
2011-09-02785110356163871360	2.0	2
...	...	...
2011-05-17-5744439083839869952	2.0	2
2014-07-312511218136851899904	2.0	1
2009-03-09-3998874973550231040	1.0	2
2013-10-25-1937500370690820096	2.0	2
2010-03-16-540841777105317184	1.0	2
2008-11-27-8117622520237889536	2.0	1
2014-06-106032564288064509952	2.0	2
2013-02-15-18620931705533700	2.0	1
2009-12-2831344694288447232 00	2.0	2
2009-07-30-7066130099458830336	2.0	1
2016-05-317633822930609668096	1.0	2
2012-06-21-1921873807210128896	1.0	2
2016-11-18-311216742844917312	2.0	2
2009-07-29-8265931828440649728	2.0	2
2009-03-315942456735192465408	1.0	2
2009-07-30-7184619100629243904	2.0	1
2012-12-102700869859969658880	1.0	2
2014-03-073816239996082851840	1.0	2
2015-07-091075829897044720000	2.0	2
2011-09-09-2820949354713164800	1.0	1
2008-10-15282679175839416000	2.0	1
2012-04-171106219260751149952	2.0	2
2013-02-28368568132944748032	2.0	2
2013-03-073523415482885629952	1.0	2
2011-02-22-2420683185350238208	1.0	1
2009-03-13-6859066878463339520	2.0	2
2013-04-16-4042990949616907776	1.0	1
2014-10-032178887206724697088	1.0	1
2016-07-05-3828694392371445760	1.0	1
2009-04-14-9045844499657849856	2.0	1

	Verbal: Stroop				\
	P ParaulaPD	P ParaulaPT	C ColorPD	C ColorPT	
	PD	PT	PD	PT	
2012-02-07-3094248760095479808	1.0	1.0	1.0	1.0	
2008-10-23-4978555423559782400	1.0	1.0	1.0	1.0	
2017-05-18-4689198199369149440	1.0	1.0	1.0	1.0	
2013-12-19-3199504984151427072	2.0	2.0	2.0	1.0	
2014-02-25-5076076858799773696	2.0	2.0	2.0	2.0	
2010-12-144650200895339849728	1.0	1.0	1.0	1.0	
2015-02-13-651871482488781824	1.0	1.0	1.0	1.0	
2013-07-042048871307966173952	2.0	2.0	2.0	1.0	
2015-01-081524797290136683008	1.0	1.0	1.0	1.0	
2014-03-11-8591197668126770176	1.0	1.0	1.0	1.0	
2010-01-143299776405939630080	1.0	1.0	1.0	1.0	
2013-10-086383101741557695488	2.0	2.0	2.0	2.0	
2014-10-022178887206724697088	1.0	1.0	1.0	1.0	
2017-07-27-5987028499592949760	1.0	1.0	1.0	1.0	
2015-06-045809237815719999488	1.0	1.0	1.0	1.0	
2014-03-13-3679307940707129856	1.0	1.0	1.0	1.0	
2008-12-19-7933846893055049728	1.0	1.0	1.0	1.0	
2014-07-081733670778237366016	2.0	2.0	1.0	1.0	
2014-04-11-6874778161861040128	1.0	1.0	1.0	1.0	
2011-11-094148544000739942912	1.0	1.0	1.0	1.0	
2017-04-27-937487139953573376	2.0	2.0	2.0	2.0	
2017-04-184748903970519549952	1.0	1.0	1.0	1.0	
2016-01-08-6056684102447908864	2.0	2.0	1.0	1.0	
2011-02-24-1362401263924740096	1.0	1.0	1.0	1.0	
2009-05-295752104861709540352	1.0	1.0	1.0	1.0	
2015-04-166167587522835896320	1.0	1.0	1.0	1.0	
2015-10-02-2953031948695728128	2.0	2.0	2.0	1.0	
2014-07-181402706660063879936	1.0	1.0	1.0	1.0	
2011-08-26900106112460610560	1.0	1.0	1.0	1.0	
2011-09-02785110356163871360	1.0	1.0	1.0	1.0	
...	...	...	...	...	
2011-05-17-5744439083839869952	1.0	1.0	1.0	1.0	
2014-07-312511218136851899904	1.0	1.0	1.0	1.0	
2009-03-09-3998874973550231040	1.0	1.0	1.0	1.0	
2013-10-25-1937500370690820096	1.0	1.0	1.0	1.0	
2010-03-16-540841777105317184	2.0	2.0	2.0	1.0	
2008-11-27-8117622520237889536	1.0	1.0	1.0	1.0	
2014-06-106032564288064509952	1.0	1.0	1.0	1.0	
2013-02-15-18620931705533700	1.0	1.0	1.0	1.0	
2009-12-283134469428844723200	1.0	1.0	1.0	1.0	
2009-07-30-7066130099458830336	1.0	1.0	1.0	1.0	
2016-05-317633822930609668096	1.0	2.0	2.0	2.0	
2012-06-21-1921873807210128896	1.0	1.0	2.0	1.0	
2016-11-18-311216742844917312	2.0	2.0	1.0	1.0	
2009-07-29-8265931828440649728	1.0	1.0	1.0	1.0	

2009-03-315942456735192465408	1.0	1.0	1.0	1.0
2009-07-30-7184619100629243904	2.0	2.0	2.0	2.0
2012-12-102700869859969658880	2.0	2.0	2.0	2.0
2014-03-073816239996082851840	1.0	1.0	1.0	1.0
2015-07-091075829897044720000	1.0	1.0	2.0	2.0
2011-09-09-2820949354713164800	2.0	2.0	2.0	2.0
2008-10-15282679175839416000	1.0	1.0	1.0	1.0
2012-04-171106219260751149952	1.0	1.0	1.0	1.0
2013-02-28368568132944748032	1.0	1.0	1.0	1.0
2013-03-073523415482885629952	2.0	2.0	2.0	2.0
2011-02-22-2420683185350238208	1.0	1.0	1.0	1.0
2009-03-13-6859066878463339520	1.0	1.0	1.0	1.0
2013-04-16-4042990949616907776	1.0	1.0	1.0	1.0
2014-10-032178887206724697088	1.0	1.0	1.0	1.0
2016-07-05-3828694392371445760	1.0	1.0	1.0	1.0
2009-04-14-9045844499657849856	1.0	1.0	1.0	1.0

		...	\
	PC ParaulaColorPD	...	
	PD	...	
2012-02-07-3094248760095479808	1.0	...	
2008-10-23-4978555423559782400	1.0	...	
2017-05-18-4689198199369149440	1.0	...	
2013-12-19-3199504984151427072	1.0	...	
2014-02-25-5076076858799773696	1.0	...	
2010-12-144650200895339849728	1.0	...	
2015-02-13-651871482488781824	1.0	...	
2013-07-042048871307966173952	2.0	...	
2015-01-081524797290136683008	1.0	...	
2014-03-11-8591197668126770176	1.0	...	
2010-01-143299776405939630080	1.0	...	
2013-10-086383101741557695488	2.0	...	
2014-10-022178887206724697088	1.0	...	
2017-07-27-5987028499592949760	1.0	...	
2015-06-045809237815719999488	1.0	...	
2014-03-13-3679307940707129856	1.0	...	
2008-12-19-7933846893055049728	1.0	...	
2014-07-081733670778237366016	2.0	...	
2014-04-11-6874778161861040128	1.0	...	
2011-11-094148544000739942912	1.0	...	
2017-04-27-937487139953573376	2.0	...	
2017-04-184748903970519549952	1.0	...	
2016-01-08-6056684102447908864	2.0	...	
2011-02-24-1362401263924740096	1.0	...	
2009-05-295752104861709540352	1.0	...	
2015-04-166167587522835896320	2.0	...	
2015-10-02-2953031948695728128	2.0	...	
2014-07-181402706660063879936	1.0	...	

2011-08-26900106112460610560	1.0	...
2011-09-02785110356163871360	1.0	...
...	...	...
2011-05-17-5744439083839869952	1.0	...
2014-07-312511218136851899904	1.0	...
2009-03-09-3998874973550231040	1.0	...
2013-10-25-1937500370690820096	1.0	...
2010-03-16-540841777105317184	2.0	...
2008-11-27-8117622520237889536	1.0	...
2014-06-106032564288064509952	1.0	...
2013-02-15-18620931705533700	1.0	...
2009-12-283134469428844723200	1.0	...
2009-07-30-7066130099458830336	1.0	...
2016-05-317633822930609668096	2.0	...
2012-06-21-1921873807210128896	2.0	...
2016-11-18-311216742844917312	2.0	...
2009-07-29-8265931828440649728	1.0	...
2009-03-315942456735192465408	1.0	...
2009-07-30-7184619100629243904	2.0	...
2012-12-102700869859969658880	2.0	...
2014-03-073816239996082851840	1.0	...
2015-07-091075829897044720000	2.0	...
2011-09-09-2820949354713164800	2.0	...
2008-10-15282679175839416000	1.0	...
2012-04-171106219260751149952	1.0	...
2013-02-28368568132944748032	1.0	...
2013-03-073523415482885629952	2.0	...
2011-02-22-2420683185350238208	1.0	...
2009-03-13-6859066878463339520	1.0	...
2013-04-16-4042990949616907776	1.0	...
2014-10-032178887206724697088	1.0	...
2016-07-05-3828694392371445760	1.0	...
2009-04-14-9045844499657849856	1.0	...

	Visual: Clave de números - Codificación (WAIS-III) \ Correctes Correctes
2012-02-07-3094248760095479808	1.0
2008-10-23-4978555423559782400	1.0
2017-05-18-4689198199369149440	1.0
2013-12-19-3199504984151427072	2.0
2014-02-25-5076076858799773696	1.0
2010-12-144650200895339849728	1.0
2015-02-13-651871482488781824	1.0
2013-07-042048871307966173952	2.0
2015-01-081524797290136683008	1.0
2014-03-11-8591197668126770176	1.0
2010-01-143299776405939630080	1.0

2013-10-086383101741557695488	2.0
2014-10-022178887206724697088	1.0
2017-07-27-5987028499592949760	1.0
2015-06-045809237815719999488	1.0
2014-03-13-3679307940707129856	1.0
2008-12-19-7933846893055049728	1.0
2014-07-081733670778237366016	1.0
2014-04-11-6874778161861040128	1.0
2011-11-094148544000739942912	2.0
2017-04-27-937487139953573376	1.0
2017-04-184748903970519549952	1.0
2016-01-08-6056684102447908864	1.0
2011-02-24-1362401263924740096	1.0
2009-05-295752104861709540352	1.0
2015-04-166167587522835896320	1.0
2015-10-02-2953031948695728128	2.0
2014-07-181402706660063879936	1.0
2011-08-26900106112460610560	1.0
2011-09-02785110356163871360	1.0
...	...
2011-05-17-5744439083839869952	1.0
2014-07-312511218136851899904	1.0
2009-03-09-3998874973550231040	1.0
2013-10-25-1937500370690820096	1.0
2010-03-16-540841777105317184	2.0
2008-11-27-8117622520237889536	1.0
2014-06-106032564288064509952	1.0
2013-02-15-18620931705533700	1.0
2009-12-283134469428844723200	1.0
2009-07-30-7066130099458830336	1.0
2016-05-317633822930609668096	1.0
2012-06-21-1921873807210128896	2.0
2016-11-18-311216742844917312	1.0
2009-07-29-8265931828440649728	1.0
2009-03-315942456735192465408	1.0
2009-07-30-7184619100629243904	2.0
2012-12-102700869859969658880	1.0
2014-03-073816239996082851840	1.0
2015-07-091075829897044720000	2.0
2011-09-09-2820949354713164800	1.0
2008-10-15282679175839416000	1.0
2012-04-171106219260751149952	1.0
2013-02-28368568132944748032	1.0
2013-03-073523415482885629952	1.0
2011-02-22-2420683185350238208	1.0
2009-03-13-6859066878463339520	1.0
2013-04-16-4042990949616907776	1.0
2014-10-032178887206724697088	1.0

2016-07-05-3828694392371445760		1.0
2009-04-14-9045844499657849856		1.0

	Dígitos: inversos (WAIS)	\
	Digits inversosWaisPD	
	PD	
2012-02-07-3094248760095479808	1.0	
2008-10-23-4978555423559782400	1.0	
2017-05-18-4689198199369149440	1.0	
2013-12-19-3199504984151427072	1.0	
2014-02-25-5076076858799773696	1.0	
2010-12-144650200895339849728	2.0	
2015-02-13-651871482488781824	1.0	
2013-07-042048871307966173952	1.0	
2015-01-081524797290136683008	1.0	
2014-03-11-8591197668126770176	1.0	
2010-01-143299776405939630080	2.0	
2013-10-086383101741557695488	1.0	
2014-10-022178887206724697088	1.0	
2017-07-27-5987028499592949760	2.0	
2015-06-045809237815719999488	1.0	
2014-03-13-3679307940707129856	2.0	
2008-12-19-7933846893055049728	1.0	
2014-07-081733670778237366016	1.0	
2014-04-11-6874778161861040128	2.0	
2011-11-094148544000739942912	1.0	
2017-04-27-937487139953573376	1.0	
2017-04-184748903970519549952	1.0	
2016-01-08-6056684102447908864	1.0	
2011-02-24-1362401263924740096	2.0	
2009-05-295752104861709540352	1.0	
2015-04-166167587522835896320	1.0	
2015-10-02-2953031948695728128	1.0	
2014-07-181402706660063879936	1.0	
2011-08-26900106112460610560	1.0	
2011-09-02785110356163871360	1.0	
...	...	
2011-05-17-5744439083839869952	2.0	
2014-07-312511218136851899904	2.0	
2009-03-09-3998874973550231040	1.0	
2013-10-25-1937500370690820096	2.0	
2010-03-16-540841777105317184	1.0	
2008-11-27-8117622520237889536	1.0	
2014-06-106032564288064509952	2.0	
2013-02-15-18620931705533700	2.0	
2009-12-283134469428844723200	1.0	
2009-07-30-7066130099458830336	2.0	
2016-05-317633822930609668096	1.0	

2012-06-21-1921873807210128896	1.0
2016-11-18-311216742844917312	1.0
2009-07-29-8265931828440649728	2.0
2009-03-315942456735192465408	1.0
2009-07-30-7184619100629243904	1.0
2012-12-102700869859969658880	1.0
2014-03-073816239996082851840	1.0
2015-07-091075829897044720000	2.0
2011-09-09-2820949354713164800	1.0
2008-10-15282679175839416000	2.0
2012-04-171106219260751149952	1.0
2013-02-28368568132944748032	2.0
2013-03-073523415482885629952	2.0
2011-02-22-2420683185350238208	1.0
2009-03-13-6859066878463339520	1.0
2013-04-16-4042909049616907776	1.0
2014-10-032178887206724697088	1.0
2016-07-05-3828694392371445760	1.0
2009-04-14-9045844499657849856	1.0

	Digits inversosWaisPcPe	TAPPING de McQuarrie \ Ma dominant
	Pe	Ma dominant
2012-02-07-3094248760095479808	1.0	1.0
2008-10-23-4978555423559782400	1.0	1.0
2017-05-18-4689198199369149440	1.0	2.0
2013-12-19-3199504984151427072	1.0	1.0
2014-02-25-5076076858799773696	1.0	2.0
2010-12-144650200895339849728	1.0	1.0
2015-02-13-651871482488781824	1.0	1.0
2013-07-042048871307966173952	1.0	2.0
2015-01-081524797290136683008	1.0	1.0
2014-03-11-8591197668126770176	1.0	1.0
2010-01-143299776405939630080	1.0	1.0
2013-10-086383101741557695488	1.0	1.0
2014-10-022178887206724697088	1.0	1.0
2017-07-27-5987028499592949760	2.0	1.0
2015-06-045809237815719999488	1.0	1.0
2014-03-13-3679307940707129856	2.0	1.0
2008-12-19-7933846893055049728	1.0	1.0
2014-07-081733670778237366016	1.0	1.0
2014-04-11-6874778161861040128	1.0	1.0
2011-11-094148544000739942912	1.0	1.0
2017-04-27-937487139953573376	1.0	1.0
2017-04-184748903970519549952	1.0	1.0
2016-01-08-6056684102447908864	1.0	1.0
2011-02-24-1362401263924740096	2.0	1.0
2009-05-295752104861709540352	1.0	1.0

2015-04-166167587522835896320	1.0	1.0
2015-10-02-2953031948695728128	1.0	2.0
2014-07-181402706660063879936	1.0	1.0
2011-08-26900106112460610560	1.0	1.0
2011-09-02785110356163871360	1.0	1.0
...	...	...
2011-05-17-5744439083839869952	2.0	1.0
2014-07-312511218136851899904	2.0	1.0
2009-03-09-3998874973550231040	1.0	1.0
2013-10-25-1937500370690820096	1.0	1.0
2010-03-16-540841777105317184	1.0	2.0
2008-11-27-8117622520237889536	1.0	1.0
2014-06-106032564288064509952	2.0	1.0
2013-02-15-18620931705533700	2.0	1.0
2009-12-283134469428844723200	1.0	1.0
2009-07-30-7066130099458830336	2.0	1.0
2016-05-317633822930609668096	1.0	1.0
2012-06-21-1921873807210128896	1.0	1.0
2016-11-18-311216742844917312	1.0	1.0
2009-07-29-8265931828440649728	1.0	1.0
2009-03-315942456735192465408	1.0	2.0
2009-07-30-7184619100629243904	1.0	2.0
2012-12-102700869859969658880	1.0	1.0
2014-03-073816239996082851840	1.0	1.0
2015-07-091075829897044720000	2.0	2.0
2011-09-09-2820949354713164800	1.0	1.0
2008-10-15282679175839416000	2.0	1.0
2012-04-171106219260751149952	1.0	1.0
2013-02-28368568132944748032	2.0	1.0
2013-03-073523415482885629952	2.0	1.0
2011-02-22-2420683185350238208	1.0	1.0
2009-03-13-6859066878463339520	1.0	1.0
2013-04-16-4042990949616907776	1.0	1.0
2014-10-032178887206724697088	1.0	1.0
2016-07-05-3828694392371445760	1.0	1.0
2009-04-14-9045844499657849856	1.0	1.0

\

	Ma dominantPT	Ma nodominant	Ma nodominantPT
	PT	Ma nodominant	PT
2012-02-07-3094248760095479808	1.0	1.0	1.0
2008-10-23-4978555423559782400	1.0	1.0	1.0
2017-05-18-4689198199369149440	2.0	2.0	2.0
2013-12-19-3199504984151427072	1.0	1.0	1.0
2014-02-25-5076076858799773696	2.0	1.0	1.0
2010-12-144650200895339849728	1.0	1.0	1.0
2015-02-13-651871482488781824	1.0	1.0	1.0
2013-07-042048871307966173952	2.0	2.0	2.0

2015-01-081524797290136683008	1.0	1.0	1.0
2014-03-11-8591197668126770176	1.0	1.0	1.0
2010-01-143299776405939630080	1.0	1.0	1.0
2013-10-086383101741557695488	1.0	1.0	1.0
2014-10-022178887206724697088	1.0	1.0	1.0
2017-07-27-5987028499592949760	1.0	1.0	1.0
2015-06-045809237815719999488	1.0	1.0	1.0
2014-03-13-3679307940707129856	1.0	1.0	1.0
2008-12-19-7933846893055049728	1.0	1.0	1.0
2014-07-081733670778237366016	1.0	1.0	1.0
2014-04-11-6874778161861040128	1.0	1.0	1.0
2011-11-094148544000739942912	1.0	1.0	1.0
2017-04-27-937487139953573376	1.0	1.0	1.0
2017-04-184748903970519549952	1.0	1.0	1.0
2016-01-08-6056684102447908864	1.0	1.0	1.0
2011-02-24-1362401263924740096	1.0	1.0	1.0
2009-05-295752104861709540352	1.0	1.0	1.0
2015-04-166167587522835896320	1.0	1.0	1.0
2015-10-02-2953031948695728128	2.0	2.0	2.0
2014-07-181402706660063879936	1.0	1.0	1.0
2011-08-26900106112460610560	1.0	1.0	1.0
2011-09-02785110356163871360	1.0	1.0	1.0
...	...	...	...
2011-05-17-5744439083839869952	1.0	1.0	1.0
2014-07-312511218136851899904	1.0	1.0	1.0
2009-03-09-3998874973550231040	1.0	1.0	1.0
2013-10-25-1937500370690820096	1.0	1.0	1.0
2010-03-16-540841777105317184	2.0	2.0	2.0
2008-11-27-8117622520237889536	1.0	1.0	1.0
2014-06-106032564288064509952	1.0	1.0	1.0
2013-02-15-18620931705533700	1.0	1.0	1.0
2009-12-283134469428844723200	1.0	1.0	1.0
2009-07-30-7066130099458830336	1.0	1.0	1.0
2016-05-317633822930609668096	2.0	1.0	1.0
2012-06-21-1921873807210128896	1.0	1.0	1.0
2016-11-18-311216742844917312	1.0	1.0	1.0
2009-07-29-8265931828440649728	1.0	1.0	1.0
2009-03-315942456735192465408	2.0	1.0	1.0
2009-07-30-7184619100629243904	2.0	2.0	2.0
2012-12-102700869859969658880	1.0	1.0	1.0
2014-03-073816239996082851840	1.0	1.0	1.0
2015-07-091075829897044720000	2.0	2.0	2.0
2011-09-09-2820949354713164800	1.0	1.0	1.0
2008-10-15282679175839416000	1.0	1.0	1.0
2012-04-171106219260751149952	1.0	1.0	1.0
2013-02-28368568132944748032	1.0	1.0	1.0
2013-03-073523415482885629952	1.0	1.0	1.0
2011-02-22-2420683185350238208	1.0	1.0	1.0

2009-03-13-6859066878463339520	1.0	1.0	1.0
2013-04-16-4042990949616907776	1.0	1.0	1.0
2014-10-032178887206724697088	1.0	1.0	1.0
2016-07-05-3828694392371445760	1.0	1.0	1.0
2009-04-14-9045844499657849856	1.0	1.0	1.0

	Orientación de líneas RBANS \ Orienta liniesPT PT
2012-02-07-3094248760095479808	1.0
2008-10-23-4978555423559782400	1.0
2017-05-18-4689198199369149440	1.0
2013-12-19-3199504984151427072	1.0
2014-02-25-5076076858799773696	1.0
2010-12-144650200895339849728	2.0
2015-02-13-651871482488781824	1.0
2013-07-042048871307966173952	1.0
2015-01-081524797290136683008	1.0
2014-03-11-8591197668126770176	2.0
2010-01-143299776405939630080	2.0
2013-10-086383101741557695488	1.0
2014-10-022178887206724697088	1.0
2017-07-27-5987028499592949760	1.0
2015-06-045809237815719999488	1.0
2014-03-13-3679307940707129856	1.0
2008-12-19-7933846893055049728	1.0
2014-07-081733670778237366016	1.0
2014-04-11-6874778161861040128	2.0
2011-11-094148544000739942912	1.0
2017-04-27-937487139953573376	1.0
2017-04-184748903970519549952	1.0
2016-01-08-6056684102447908864	1.0
2011-02-24-1362401263924740096	2.0
2009-05-295752104861709540352	2.0
2015-04-166167587522835896320	1.0
2015-10-02-2953031948695728128	1.0
2014-07-181402706660063879936	1.0
2011-08-26900106112460610560	1.0
2011-09-027851103561638713 60	1.0
...	...
2011-05-17-5744439083839869952	1.0
2014-07-312511218136851899904	2.0
2009-03-09-3998874973550231040	1.0
2013-10-25-1937500370690820096	1.0
2010-03-16-540841777105317184	1.0
2008-11-27-8117622520237889536	1.0
2014-06-106032564288064509952	1.0
2013-02-15-18620931705533700	1.0

2009-12-283134469428844723200	1.0
2009-07-30-7066130099458830336	1.0
2016-05-317633822930609668096	1.0
2012-06-21-1921873807210128896	1.0
2016-11-18-311216742844917312	1.0
2009-07-29-8265931828440649728	1.0
2009-03-315942456735192465408	1.0
2009-07-30-7184619100629243904	1.0
2012-12-102700869859969658880	1.0
2014-03-073816239996082851840	1.0
2015-07-091075829897044720000	1.0
2011-09-09-2820949354713164800	1.0
2008-10-15282679175839416000	2.0
2012-04-171106219260751149952	1.0
2013-02-28368568132944748032	2.0
2013-03-073523415482885629952	1.0
2011-02-22-2420683185350238208	1.0
2009-03-13-6859066878463339520	1.0
2013-04-16-4042990949616907776	1.0
2014-10-032178887206724697088	1.0
2016-07-05-3828694392371445760	1.0
2009-04-14-9045844499657849856	1.0

	Dígitos: directos (WAIS) \ Digits directes	WaisPD PD
2012-02-07-3094248760095479808		2.0
2008-10-23-4978555423559782400		1.0
2017-05-18-4689198199369149440		1.0
2013-12-19-3199504984151427072		1.0
2014-02-25-5076076858799773696		1.0
2010-12-144650200895339849728		2.0
2015-02-13-651871482488781824		1.0
2013-07-042048871307966173952		1.0
2015-01-081524797290136683008		1.0
2014-03-11-8591197668126770176		1.0
2010-01-143299776405939630080		2.0
2013-10-086383101741557695488		1.0
2014-10-022178887206724697088		1.0
2017-07-27-5987028499592949760		2.0
2015-06-045809237815719999488		2.0
2014-03-13-3679307940707129856		2.0
2008-12-19-7933846893055049728		2.0
2014-07-081733670778237366016		1.0
2014-04-11-6874778161861040128		2.0
2011-11-094148544000739942912		1.0
2017-04-27-937487139953573376		1.0
2017-04-184748903970519549952		2.0

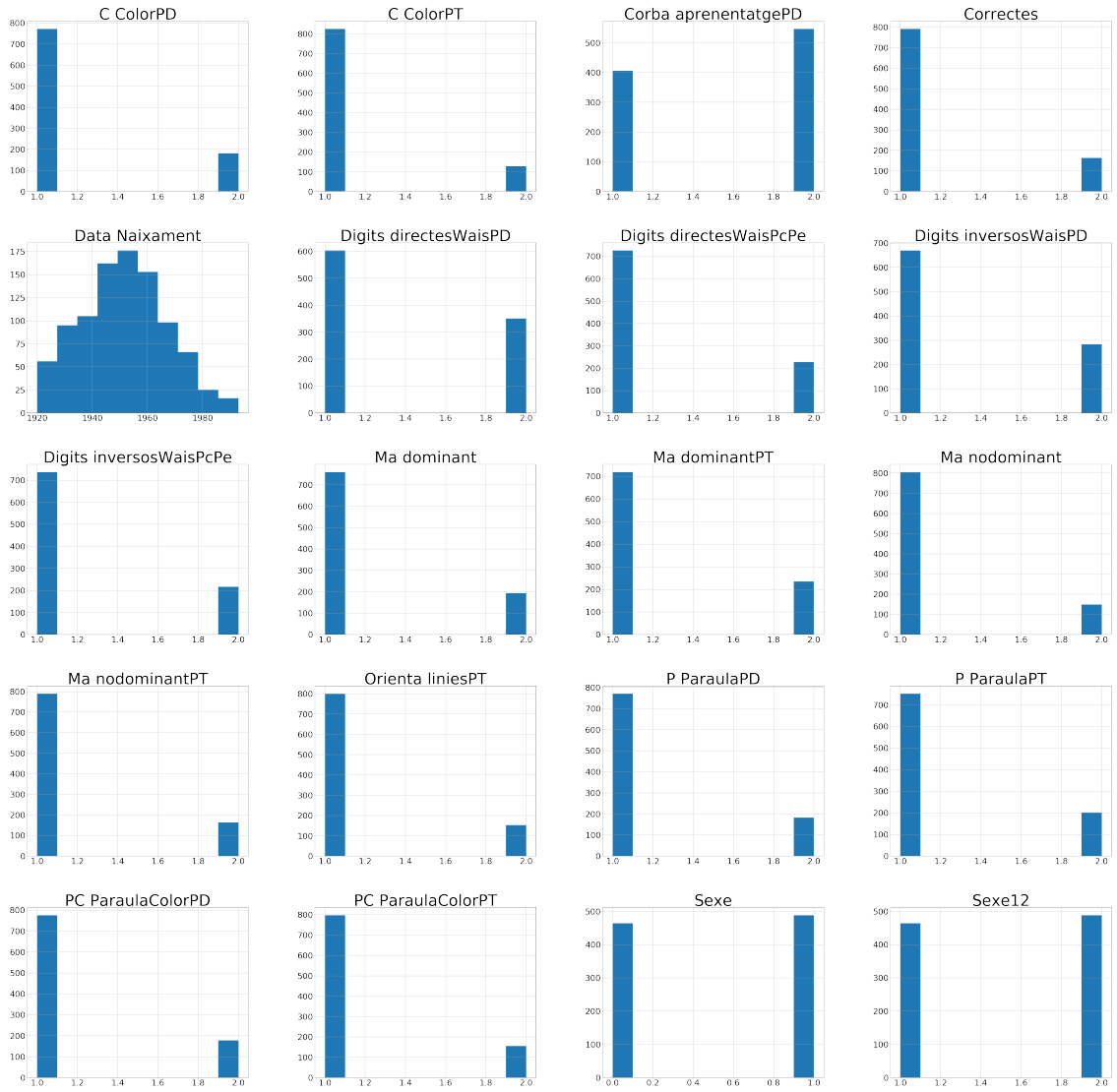
2016-01-08-6056684102447908864	1.0
2011-02-24-1362401263924740096	2.0
2009-05-295752104861709540352	1.0
2015-04-166167587522835896320	1.0
2015-10-02-2953031948695728128	1.0
2014-07-181402706660063879936	2.0
2011-08-26900106112460610560	1.0
2011-09-02785110356163871360	1.0
...	...
2011-05-17-5744439083839869952	2.0
2014-07-312511218136851899904	2.0
2009-03-09-3998874973550231040	1.0
2013-10-25-1937500370690820096	2.0
2010-03-16-540841777105317184	1.0
2008-11-27-8117622520237889536	2.0
2014-06-106032564288064509952	1.0
2013-02-15-18620931705533700	2.0
2009-12-283134469428844723200	1.0
2009-07-30-7066130099458830336	2.0
2016-05-317633822930609668096	1.0
2012-06-21-1921873807210128896	1.0
2016-11-18-311216742844917312	1.0
2009-07-29-8265931828440649728	2.0
2009-03-315942456735192465408	1.0
2009-07-30-7184619100629243904	1.0
2012-12-102700869859969658880	1.0
2014-03-073816239996082851840	1.0
2015-07-091075829897044720000	2.0
2011-09-09-2820949354713164800	1.0
2008-10-15282679175839416000	2.0
2012-04-171106219260751149952	2.0
2013-02-28368568132944748032	2.0
2013-03-073523415482885629952	2.0
2011-02-22-2420683185350238208	1.0
2009-03-13-6859066878463339520	2.0
2013-04-16-4042990949616907776	1.0
2014-10-032178887206724697088	1.0
2016-07-05-3828694392371445760	1.0
2009-04-14-9045844499657849856	2.0

	Digits directesWaisPcPe Pe
2012-02-07-3094248760095479808	1.0
2008-10-23-4978555423559782400	1.0
2017-05-18-4689198199369149440	1.0
2013-12-19-3199504984151427072	1.0
2014-02-25-5076076858799773696	1.0

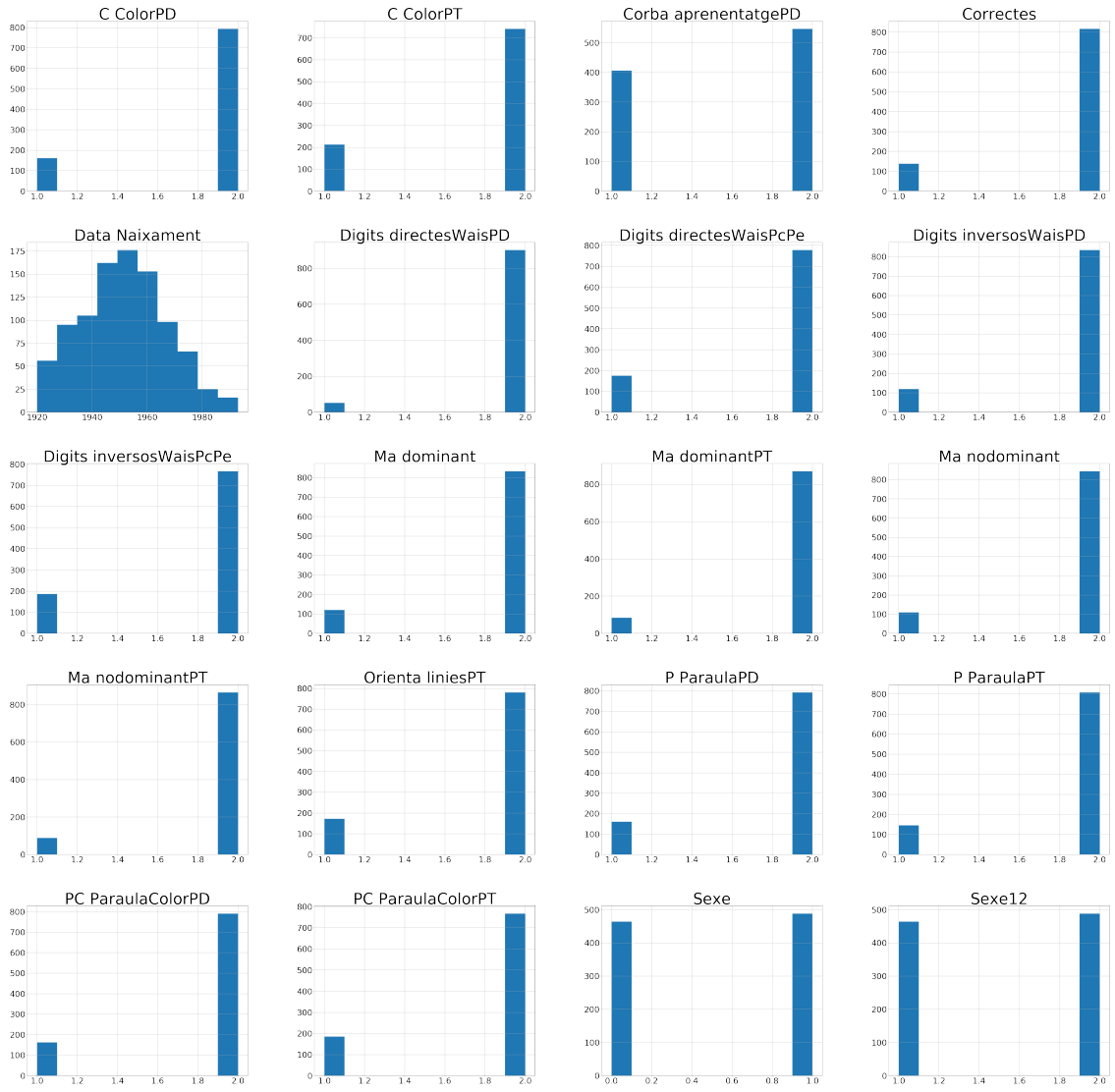
```
2010-12-144650200895339849728                    1.0
2015-02-13-651871482488781824                    1.0
2013-07-042048871307966173952                    1.0
2015-01-081524797290136683008                    1.0
2014-03-11-8591197668126770176                   1.0
2010-01-143299776405939630080                    2.0
2013-10-086383101741557695488                    1.0
2014-10-022178887206724697088                    1.0
2017-07-27-5987028499592949760                   2.0
2015-06-045809237815719999488                    2.0
2014-03-13-3679307940707129856                   2.0
2008-12-19-7933846893055049728                   2.0
2014-07-081733670778237366016                    1.0
2014-04-11-6874778161861040128                   1.0
2011-11-094148544000739942912                    1.0
2017-04-27-937487139953573376                    1.0
2017-04-184748903970519549952                    2.0
2016-01-08-6056684102447908864                   1.0
2011-02-24-1362401263924740096                   1.0
2009-05-295752104861709540352                    1.0
2015-04-166167587522835896320                    1.0
2015-10-02-2953031948695728128                   1.0
2014-07-181402706660063879936                    2.0
2011-08-26900106112460610560                     1.0
2011-09-027851103561638713 60                    1.0
...                                              ...
2011-05-17-5744439083839869952                   2.0
2014-07-312511218136851899904                    2.0
2009-03-09-3998874973550231040                   1.0
2013-10-25-1937500370690820096                   1.0
2010-03-16-540841777105317184                    1.0
2008-11-27-8117622520237889536                   2.0
2014-06-106032564288064509952                    1.0
2013-02-15-18620931705533700                     2.0
2009-12-2831344694428844723200                   1.0
2009-07-30-7066130099458830336                   1.0
2016-05-317633822930609668096                    1.0
2012-06-21-1921873807210128896                   1.0
2016-11-18-311216742844917312                    1.0
2009-07-29-8265931828440649728                   1.0
2009-03-315942456735192465408                    1.0
2009-07-30-7184619100629243904                   1.0
2012-12-102700869859969658880                    1.0
2014-03-073816239996082851840                    1.0
2015-07-091075829897044720000                    2.0
2011-09-09-2820949354713164800                   1.0
2008-10-15282679175839416000                     2.0
2012-04-171106219260751149952                    2.0
```

```
2013-02-28368568132944748032                    2.0
2013-03-073523415482885629952                   2.0
2011-02-22-2420683185350238208                  1.0
2009-03-13-6859066878463339520                  2.0
2013-04-16-4042990949616907776                  1.0
2014-10-032178887206724697088                   1.0
2016-07-05-3828694392371445760                  1.0
2009-04-14-9045844499657849856                  2.0

[952 rows x 21 columns]
```

In [8]:
```
df_copy = df_optimistic.copy()

df_copy.columns = df_copy.columns.get_level_values(1)
params = {'axes.titlesize':'64',
          'xtick.labelsize':'34',
          'ytick.labelsize':'34'}

matplotlib.rcParams.update(params)
df_copy.hist(figsize=(80,80))
plt.savefig('../Figures/histogram description binary optimistic 2', bbox_inches='tight')
plt.show()
```




```
In [9]: df_copy = df_pesimistic.copy()

        df_copy.columns = df_copy.columns.get_level_values(1)
        params = {'axes.titlesize':'64',
                  'xtick.labelsize':'34',
                  'ytick.labelsize':'34'}

        matplotlib.rcParams.update(params)
        df_copy.hist(figsize=(80,80))
        plt.savefig('../Figures/histogram description binary pesimistic 2', bbox_inches='tight')
        plt.show()
```



In [10]:
```
# Creating object for storing information
class relation_between_columns:
    """Clase base regresion lineal dos variables"""

    def __init__(self):
        self.confusion_matrix = 0.0
        self.precision1 = 0.0
        self.recall1 = 0.0
        self.f1_score1 = 0.0
        self.precision2 = 0.0
        self.recall2 = 0.0
        self.f1_score2 = 0.0
        self.report = 0.0
        self.weight = []
```

```
            self.name_predicted_column = tuple()

        def __str__(self):
            return str(self.__dict__)
```

```
In [11]: # Here we define a function for easy obtain
         def finding_patterns_between_columns(X, Y, name_predicted_column, model, relation):
            splitting_method = KFold(5)

            conf_matrix = 0
            confusion_matrix_array = []
            for train_index, test_index in splitting_method.split(X):
                X_train, X_test = X.iloc[train_index, 3:], X.iloc[test_index, 3:]
                Y_train, Y_test = Y.iloc[train_index], Y.iloc[test_index]
                model.fit(X_train, Y_train)
                Y_predicted = model.predict(X_test)
                Y_test = np.array(Y_test)
                # Weight
                try:
                    coef = model.coef_
                    coefs_array.append(coef)
                except:
                    pass
                # Confusion matrix
                conf_matrix = confusion_matrix(Y_test, Y_predicted)
                confusion_matrix_array.append(conf_matrix)

            relation.name_predicted_column = name_predicted_column
            conf_matrix = np.matrix([[1,0],[0,1]])
            for i in range(2):
                for j in range(2):
                    element = []
                    for n in range(len(confusion_matrix_array)):
                        element.append(confusion_matrix_array[n][i][j])
                    conf_matrix[i,j] = np.sum(element)

            relation.confusion_matrix = conf_matrix

            TP1 = conf_matrix[0,0]
            TN1 = conf_matrix[1,1]
            FP1 = conf_matrix[1,0]
            FN1 = conf_matrix[0,1]

            precision1 = TP1 / (TP1 + FP1)
            recall1 = TP1 / (TP1 + FN1)
            f1_score1 = 2 * (precision1*recall1)/(precision1 + recall1)

            relation.precision1 = precision1
```

```
            relation.recall1 = recall1
            relation.f1_score1 = f1_score1

            TP2 = conf_matrix[1,1]
            TN2 = conf_matrix[0,0]
            FP2 = conf_matrix[0,1]
            FN2 = conf_matrix[1,0]

            precision2 = TP2 / (TP2 + FP2)
            recall2 = TP2 / (TP2 + FN2)
            f1_score2 = 2 * (precision2*recall2)/(precision2 + recall2)

            relation.precision2 = precision2
            relation.recall2 = recall2
            relation.f1_score2 = f1_score2

            try:
                coefficients = []
                for j in range(len(coefs_array[0])):
                    coefficient = []
                    for i in range(len(coefs_array)):
                        coefficient.append(coefs_array[i][j])
                    coefficient = np.mean(coefficient)
                    coefficients.append(coefficient)
                relation.weight = coefficients
            except:
                pass

            return relation
```

In [12]:
```
        # Here we define a function for easy obtain
        def finding_patterns_between_columns_report(X, Y, name_predicted_column, model, relatio
            splitting_method = KFold(2)

            conf_matrix = 0
            confusion_matrix_array = []
            X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33)

            model.fit(X_train, Y_train)
            Y_predicted = model.predict(X_test)
            Y_test = np.array(Y_test)

            relation.report = classification_report(Y_test, Y_predicted)

            try:
                coefficients = []
                for j in range(len(coefs_array[0])):
                    coefficient = []
```

```
                for i in range(len(coefs_array)):
                    coefficient.append(coefs_array[i][j])
                coefficient = np.mean(coefficient)
                coefficients.append(coefficient)
            relation.weight = coefficients
        except:
            pass

        return relation
```

1.1 Triying to infer one column from the other ones:

We employ the previous function in order to predict the values from one column employing the other ones. We try with several algorithms and with several strategies of filling missing values.

1.2 Logistic Regression

In [13]:
```
# LR - optimistic
df = df_optimistic.copy()

# Turn up tolerance for faster convergence
model = linear_model.LogisticRegression(multi_class='multinomial',penalty='l1', solver=

relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop(lvl0, level=0, axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [14]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/LOGISTIC - OPTIMISTIC: Relations Classification.x
row_to_write = 0
for rel in relations:
    table = pd.DataFrame(columns=['Predicted column', 'Precision_1', 'Recall_1', 'F1_1'
    string = rel.name_predicted_column
    string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
    table['Predicted column'] = [string]
    table['Precision_1'] = rel.precision1
    table['Recall_1'] = rel.recall1
    table['F1_1'] = rel.f1_score1
    table['Precision_2'] = rel.precision2
    table['Recall_2'] = rel.recall2
    table['F1_2'] = rel.f1_score2
    table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
```

```
            row_to_write = row_to_write + 3

        writer.save()
```

```
In [15]: # LR - pesimistic
         df = df_pesimistic.copy()

         # Turn up tolerance for faster convergence
         model = linear_model.LogisticRegression(multi_class='multinomial',penalty='l1', solver=

         relations = []
         for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
             Y = df[lvl0][lvl1][lvl2].copy()
             X = df.copy()
             X = X.drop(lvl0, level=0, axis=1)
             relation = relation_between_columns()
             name_predicted_column = (lvl0, lvl1, lvl2)
             relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
             relations.append(relation)
```

```
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:39: RuntimeWarning: invalid val
```

```
In [16]: # Visualizing correlations:
         writer = pd.ExcelWriter('../Summaries/LOGISTIC - PESIMISTIC: Relations Classification.x
         row_to_write = 0
         for rel in relations:
             table = pd.DataFrame(columns=['Predicted column', 'Precision_1', 'Recall_1', 'F1_1'
             string = rel.name_predicted_column
             string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
             table['Predicted column'] = [string]
             table['Precision_1'] = rel.precision1
             table['Recall_1'] = rel.recall1
             table['F1_1'] = rel.f1_score1
             table['Precision_2'] = rel.precision2
             table['Recall_2'] = rel.recall2
             table['F1_2'] = rel.f1_score2
             table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
             row_to_write = row_to_write + 3

         writer.save()
```

1.3 Support Vector Machine

```
In [17]: # SVM - optimistic
         df = df_optimistic.copy()

         # Turn up tolerance for faster convergence
         model = svm.SVC()
```

```
        relations = []
        for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
            Y = df[lvl0][lvl1][lvl2].copy()
            X = df.copy()
            X = X.drop(lvl0, level=0, axis=1)
            relation = relation_between_columns()
            name_predicted_column = (lvl0, lvl1, lvl2)
            relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
            relations.append(relation)
```

```
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:52: RuntimeWarning: invalid val
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:54: RuntimeWarning: invalid val
```

In [18]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/SVM - OPTIMISTIC: Relations Classification.xlsx')
row_to_write = 0
for rel in relations:
    table = pd.DataFrame(columns=['Predicted column', 'Precision_1', 'Recall_1', 'F1_1'
    string = rel.name_predicted_column
    string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
    table['Predicted column'] = [string]
    table['Precision_1'] = rel.precision1
    table['Recall_1'] = rel.recall1
    table['F1_1'] = rel.f1_score1
    table['Precision_2'] = rel.precision2
    table['Recall_2'] = rel.recall2
    table['F1_2'] = rel.f1_score2
    table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
    row_to_write = row_to_write + 3

writer.save()
```

In [19]:
```
# SVM - pesimistic
df = df_pesimistic.copy()

# Turn up tolerance for faster convergence
model = svm.SVC()

relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop(lvl0, level=0, axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

```
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:39: RuntimeWarning: invalid val
```

```
In [20]: # Visualizing correlations:
         writer = pd.ExcelWriter('../Summaries/SVM - PESIMISTIC: Relations Classification.xlsx')
         row_to_write = 0
         for rel in relations:
             table = pd.DataFrame(columns=['Predicted column', 'Precision_1', 'Recall_1', 'F1_1'
             string = rel.name_predicted_column
             string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
             table['Predicted column'] = [string]
             table['Precision_1'] = rel.precision1
             table['Recall_1'] = rel.recall1
             table['F1_1'] = rel.f1_score1
             table['Precision_2'] = rel.precision2
             table['Recall_2'] = rel.recall2
             table['F1_2'] = rel.f1_score2
             table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
             row_to_write = row_to_write + 3

         writer.save()
```

1.4 Support Vector Machine (unbalance penalty)

```
In [21]: # SVM - optimistic
         df = df_optimistic.copy()

         # Turn up tolerance for faster convergence
         model = svm.SVC(class_weight='balanced')

         relations = []
         for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
             Y = df[lvl0][lvl1][lvl2].copy()
             X = df.copy()
             X = X.drop(lvl0, level=0, axis=1)
             relation = relation_between_columns()
             name_predicted_column = (lvl0, lvl1, lvl2)
             relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
             relations.append(relation)
```

```
In [22]: # Visualizing correlations:
         writer = pd.ExcelWriter('../Summaries/SVM Penalty - OPTIMISTIC: Relations Classificatio
         row_to_write = 0
         for rel in relations:
             table = pd.DataFrame(columns=['Predicted column', 'Precision_1', 'Recall_1', 'F1_1'
             string = rel.name_predicted_column
             string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
             table['Predicted column'] = [string]
```

```
        table['Precision_1'] = rel.precision1
        table['Recall_1'] = rel.recall1
        table['F1_1'] = rel.f1_score1
        table['Precision_2'] = rel.precision2
        table['Recall_2'] = rel.recall2
        table['F1_2'] = rel.f1_score2
        table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
        row_to_write = row_to_write + 3

    writer.save()
```

In [23]:
```
# SVM - pesimistic
df = df_pesimistic.copy()

# Turn up tolerance for faster convergence
model = svm.SVC(class_weight='balanced')

relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop(lvl0, level=0, axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [24]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/SVM Penalty - PESIMISTIC: Relations Classificatio
row_to_write = 0
for rel in relations:
    table = pd.DataFrame(columns=['Predicted column', 'Precision_1', 'Recall_1', 'F1_1'
    string = rel.name_predicted_column
    string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
    table['Predicted column'] = [string]
    table['Precision_1'] = rel.precision1
    table['Recall_1'] = rel.recall1
    table['F1_1'] = rel.f1_score1
    table['Precision_2'] = rel.precision2
    table['Recall_2'] = rel.recall2
    table['F1_2'] = rel.f1_score2
    table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
    row_to_write = row_to_write + 3

writer.save()
```

1.5 Nearest Neighbors

In [25]:
```
# Choose the number of neighbors
df = df_optimistic.copy()
n_neighbors = np.linspace(1,60,60)
weights = 'uniform'

f1_scores_n_neighbors = []
for n in n_neighbors:
    f1_scores = []
    model = neighbors.KNeighborsClassifier(int(n))
    for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
        Y = df[lvl0][lvl1][lvl2].copy()
        X = df.copy()
        X = X.drop(lvl0, level=0, axis=1)
        relation = relation_between_columns()
        name_predicted_column = (lvl0, lvl1, lvl2)
        relation = finding_patterns_between_columns(X, Y, name_predicted_column, model,
        f1_scores.append(relation.f1_score2)
    f1_scores = np.mean(f1_scores)
    f1_scores_n_neighbors.append((n, f1_scores))

f1_scores_n_neighbors # 2 Vecinos
```

```
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:54: RuntimeWarning: invalid val
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:52: RuntimeWarning: invalid val
```

Out[25]:
```
[(1.0, 0.48946623846187226),
 (2.0, 0.41484559796893467),
 (3.0, 0.5161083073645156),
 (4.0, 0.4607998344337347),
 (5.0, 0.5276863996648824),
 (6.0, 0.4940373532691237),
 (7.0, 0.5350873071026254),
 (8.0, 0.4956930641407109),
 (9.0, 0.5308778568039826),
 (10.0, 0.5035355585330131),
 (11.0, 0.5355285454048142),
 (12.0, 0.502446490749755),
 (13.0, 0.5211951165382571),
 (14.0, 0.49680443488879583),
 (15.0, 0.5229677238983221),
 (16.0, 0.49589548018416907),
 (17.0, 0.5176810878168734),
 (18.0, 0.4838163332681335),
 (19.0, 0.5091708591388495),
 (20.0, 0.4791338967826204),
 (21.0, 0.5042096267380316),
```

```
 (22.0, 0.47585133952930203),
 (23.0, 0.49148192634720345),
 (24.0, 0.45327003837947305),
 (25.0, 0.4811267584440222),
 (26.0, 0.44598324634304665),
 (27.0, 0.46785240043828746),
 (28.0, 0.4393334421328078),
 (29.0, 0.4595714960148746),
 (30.0, 0.4316119462705019),
 (31.0, 0.453612483313811),
 (32.0, 0.4302894683699478),
 (33.0, 0.45240334316205044),
 (34.0, nan),
 (35.0, 0.44373922671522537),
 (36.0, 0.420895226751341),
 (37.0, 0.43747549654478984),
 (38.0, 0.4141668612585169),
 (39.0, 0.43635897574801685),
 (40.0, 0.4177618803505733),
 (41.0, 0.42489607852433703),
 (42.0, nan),
 (43.0, 0.4195967655311815),
 (44.0, 0.39979677241643263),
 (45.0, 0.42562788109430216),
 (46.0, 0.4099275664107434),
 (47.0, 0.42925210591388324),
 (48.0, 0.4200651905822293),
 (49.0, 0.42656442344399587),
 (50.0, 0.40952740665111603),
 (51.0, 0.42652707646347526),
 (52.0, 0.41234944487398506),
 (53.0, nan),
 (54.0, nan),
 (55.0, nan),
 (56.0, nan),
 (57.0, nan),
 (58.0, nan),
 (59.0, nan),
 (60.0, nan)]
```

In [26]:
```
# Choose the number of neighbors
df = df_pesimistic.copy()
n_neighbors = np.linspace(1,60,60)
weights = 'uniform'

f1_scores_n_neighbors = []
for n in n_neighbors:
    f1_scores = []
```

```
            model = neighbors.KNeighborsClassifier(int(n))
            for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
                Y = df[lvl0][lvl1][lvl2].copy()
                X = df.copy()
                X = X.drop(lvl0, level=0, axis=1)
                relation = relation_between_columns()
                name_predicted_column = (lvl0, lvl1, lvl2)
                relation = finding_patterns_between_columns(X, Y, name_predicted_column, model,
                f1_scores.append(relation.f1_score2)
            f1_scores = np.mean(f1_scores)
            f1_scores_n_neighbors.append((n, f1_scores))

        f1_scores_n_neighbors # 40 Vecinos
```

```
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:41: RuntimeWarning: invalid val
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:39: RuntimeWarning: invalid val
```

```
Out[26]: [(1.0, 0.8489602727891996),
          (2.0, 0.8133183459350188),
          (3.0, 0.8782156428970398),
          (4.0, 0.8544333977417915),
          (5.0, 0.8784907126670389),
          (6.0, 0.8667599306749578),
          (7.0, 0.8862034474191278),
          (8.0, 0.8755777548852469),
          (9.0, 0.8879165904539991),
          (10.0, 0.8809390301019513),
          (11.0, 0.887473663747071),
          (12.0, 0.8836691372930984),
          (13.0, 0.8882026426941522),
          (14.0, 0.8835290136372215),
          (15.0, 0.8901437160960577),
          (16.0, 0.883295172231858),
          (17.0, 0.8879372605114765),
          (18.0, 0.8858922804893974),
          (19.0, 0.8896204195953614),
          (20.0, 0.8863729611268364),
          (21.0, 0.8897128084933986),
          (22.0, 0.8862377919333044),
          (23.0, 0.8905988047655158),
          (24.0, 0.8859352457823367),
          (25.0, 0.8904875327912694),
          (26.0, 0.8850441368350221),
          (27.0, 0.88925373578338),
          (28.0, 0.8860103886530957),
          (29.0, 0.8897105697836998),
          (30.0, 0.8873855784818321),
```

```
 (31.0, 0.8904913620993933),
 (32.0, 0.886640815364768),
 (33.0, 0.8888772402497679),
 (34.0, 0.8877033397159205),
 (35.0, 0.8902194049255385),
 (36.0, 0.8872396971428013),
 (37.0, 0.8904079562989234),
 (38.0, 0.8873015832496353),
 (39.0, 0.8890339925939892),
 (40.0, 0.8886787895930979),
 (41.0, 0.8909451476714203),
 (42.0, 0.8886947364791767),
 (43.0, 0.8899350220540432),
 (44.0, 0.8884644333022543),
 (45.0, 0.8904618794396681),
 (46.0, 0.8886399228741045),
 (47.0, 0.8907634004924054),
 (48.0, 0.888244571287003),
 (49.0, 0.8908538113855238),
 (50.0, 0.8894417966906344),
 (51.0, 0.8889453369931477),
 (52.0, 0.8886356752888143),
 (53.0, 0.8898223921857331),
 (54.0, 0.8879924405765335),
 (55.0, 0.8903059274015462),
 (56.0, 0.8890444708051609),
 (57.0, 0.8903504495188089),
 (58.0, 0.889182747605707),
 (59.0, 0.8893924797029619),
 (60.0, 0.8894567577813661)]
```

In [27]:
```
# NN - optimistic
df = df_optimistic.copy()
model = neighbors.KNeighborsClassifier(n_neighbors=2)
relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop(lvl0, level=0, axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [28]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/NN - OPTIMISTIC: Relations Classification.xlsx')
row_to_write = 0
for rel in relations:
```

```
        table = pd.DataFrame(columns=['Predicted column', 'Precision_1', 'Recall_1', 'F1_1'
        string = rel.name_predicted_column
        string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
        table['Predicted column'] = [string]
        table['Precision_1'] = rel.precision1
        table['Recall_1'] = rel.recall1
        table['F1_1'] = rel.f1_score1
        table['Precision_2'] = rel.precision2
        table['Recall_2'] = rel.recall2
        table['F1_2'] = rel.f1_score2
        table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
        row_to_write = row_to_write + 3

    writer.save()
```

In [29]:
```
# NN - pesimistic
df = df_pesimistic.copy()
model = neighbors.KNeighborsClassifier(n_neighbors=2)
relations = []
for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
    Y = df[lvl0][lvl1][lvl2].copy()
    X = df.copy()
    X = X.drop(lvl0, level=0, axis=1)
    relation = relation_between_columns()
    name_predicted_column = (lvl0, lvl1, lvl2)
    relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
    relations.append(relation)
```

In [30]:
```
# Visualizing correlations:
writer = pd.ExcelWriter('../Summaries/NN - PESIMISTIC: Relations Classification.xlsx')
row_to_write = 0
for rel in relations:
    table = pd.DataFrame(columns=['Predicted column', 'Precision_1', 'Recall_1', 'F1_1'
    string = rel.name_predicted_column
    string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
    table['Predicted column'] = [string]
    table['Precision_1'] = rel.precision1
    table['Recall_1'] = rel.recall1
    table['F1_1'] = rel.f1_score1
    table['Precision_2'] = rel.precision2
    table['Recall_2'] = rel.recall2
    table['F1_2'] = rel.f1_score2
    table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
    row_to_write = row_to_write + 3

writer.save()
```

1.6 Decision Tree

In [31]:
```
# Choose the number of neighbors
df = df_optimistic.copy()
max_depth = np.linspace(1,20,20)
weights = 'uniform'

f1_scores_max_depth = []
for n in max_depth:
    f1_scores = []
    model = DecisionTreeClassifier(max_depth=n)
    for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
        Y = df[lvl0][lvl1][lvl2].copy()
        X = df.copy()
        X = X.drop(lvl0, level=0, axis=1)
        relation = relation_between_columns()
        name_predicted_column = (lvl0, lvl1, lvl2)
        relation = finding_patterns_between_columns(X, Y, name_predicted_column, model,
        f1_scores.append(relation.f1_score2)
    f1_scores = np.mean(f1_scores)
    f1_scores_max_depth.append((n, f1_scores))

f1_scores_max_depth # 13
```

```
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:52: RuntimeWarning: invalid val
```

Out[31]:
```
[(1.0, nan),
 (2.0, nan),
 (3.0, 0.5037588065008731),
 (4.0, 0.5322355665930272),
 (5.0, 0.549254091666468),
 (6.0, 0.548680190382188),
 (7.0, 0.531726945888083),
 (8.0, 0.5407595455106038),
 (9.0, 0.5337025921501447),
 (10.0, 0.5382255048790094),
 (11.0, 0.5377945005237585),
 (12.0, 0.5386568460882727),
 (13.0, 0.5377161068997975),
 (14.0, 0.5377414497180646),
 (15.0, 0.5395112307632596),
 (16.0, 0.5380936855934872),
 (17.0, 0.5373126664938694),
 (18.0, 0.5376693661062384),
 (19.0, 0.5380754204324127),
 (20.0, 0.5376859709006605)]
```

In [32]:
```
# Choose the number of neighbors
df = df_pesimistic.copy()
```

```
        max_depth = np.linspace(1,20,20)
        weights = 'uniform'

        f1_scores_max_depth = []
        for n in max_depth:
            f1_scores = []
            model = DecisionTreeClassifier(max_depth=n)
            for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
                Y = df[lvl0][lvl1][lvl2].copy()
                X = df.copy()
                X = X.drop(lvl0, level=0, axis=1)
                relation = relation_between_columns()
                name_predicted_column = (lvl0, lvl1, lvl2)
                relation = finding_patterns_between_columns(X, Y, name_predicted_column, model,
                f1_scores.append(relation.f1_score2)
            f1_scores = np.mean(f1_scores)
            f1_scores_max_depth.append((n, f1_scores))

        f1_scores_max_depth # 13
```

```
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:39: RuntimeWarning: invalid val
/home/dani/ENV/lib/python3.5/site-packages/ipykernel_launcher.py:41: RuntimeWarning: invalid val
```

```
Out[32]: [(1.0, 0.8822018539415364),
          (2.0, 0.8893263793491849),
          (3.0, 0.886524651472924),
          (4.0, 0.8860699630281395),
          (5.0, 0.8836331574813087),
          (6.0, 0.8845611177639712),
          (7.0, 0.8832251094925031),
          (8.0, 0.8800889141766856),
          (9.0, 0.8797605246506401),
          (10.0, 0.8783859227696004),
          (11.0, 0.8780385202718199),
          (12.0, 0.8778850885436552),
          (13.0, 0.8778002751789953),
          (14.0, 0.8772440152647514),
          (15.0, 0.8782211300593439),
          (16.0, 0.8781510966330607),
          (17.0, 0.878395798541681),
          (18.0, 0.8782238779141507),
          (19.0, 0.878631984272392),
          (20.0, 0.8783816058854997)]
```

```
In [33]: # NN - optimistic
         df = df_optimistic.copy()
         model = DecisionTreeClassifier(max_depth=13)
```

```
        relations = []
        for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
            Y = df[lvl0][lvl1][lvl2].copy()
            X = df.copy()
            X = X.drop(lvl0, level=0, axis=1)
            relation = relation_between_columns()
            name_predicted_column = (lvl0, lvl1, lvl2)
            relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
            relations.append(relation)
```

In [34]:
```
        # Visualizing correlations:
        writer = pd.ExcelWriter('../Summaries/DT - OPTIMISTIC: Relations Classification.xlsx')
        row_to_write = 0
        for rel in relations:
            table = pd.DataFrame(columns=['Predicted column', 'Precision_1', 'Recall_1', 'F1_1'
            string = rel.name_predicted_column
            string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
            table['Predicted column'] = [string]
            table['Precision_1'] = rel.precision1
            table['Recall_1'] = rel.recall1
            table['F1_1'] = rel.f1_score1
            table['Precision_2'] = rel.precision2
            table['Recall_2'] = rel.recall2
            table['F1_2'] = rel.f1_score2
            table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
            row_to_write = row_to_write + 3

        writer.save()
```

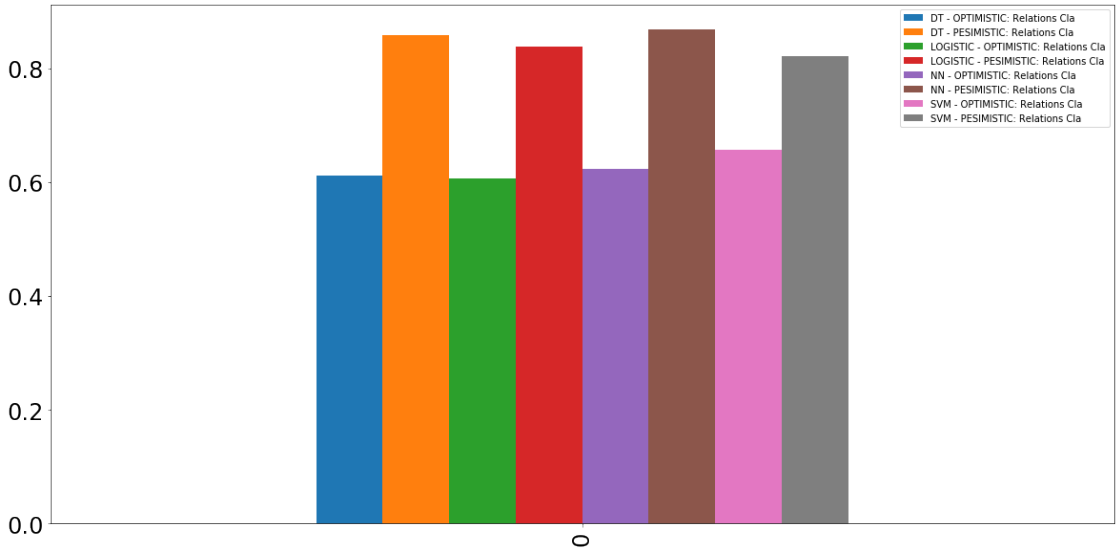
In [35]:
```
        # NN - pesimistic
        df = df_pesimistic.copy()
        model = DecisionTreeClassifier(max_depth=13)
        relations = []
        for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
            Y = df[lvl0][lvl1][lvl2].copy()
            X = df.copy()
            X = X.drop(lvl0, level=0, axis=1)
            relation = relation_between_columns()
            name_predicted_column = (lvl0, lvl1, lvl2)
            relation = finding_patterns_between_columns(X, Y, name_predicted_column, model, rel
            relations.append(relation)
```

In [36]:
```
        # Visualizing correlations:
        writer = pd.ExcelWriter('../Summaries/DT - PESIMISTIC: Relations Classification.xlsx')
        row_to_write = 0
        for rel in relations:
            table = pd.DataFrame(columns=['Predicted column', 'Precision_1', 'Recall_1', 'F1_1'
            string = rel.name_predicted_column
```

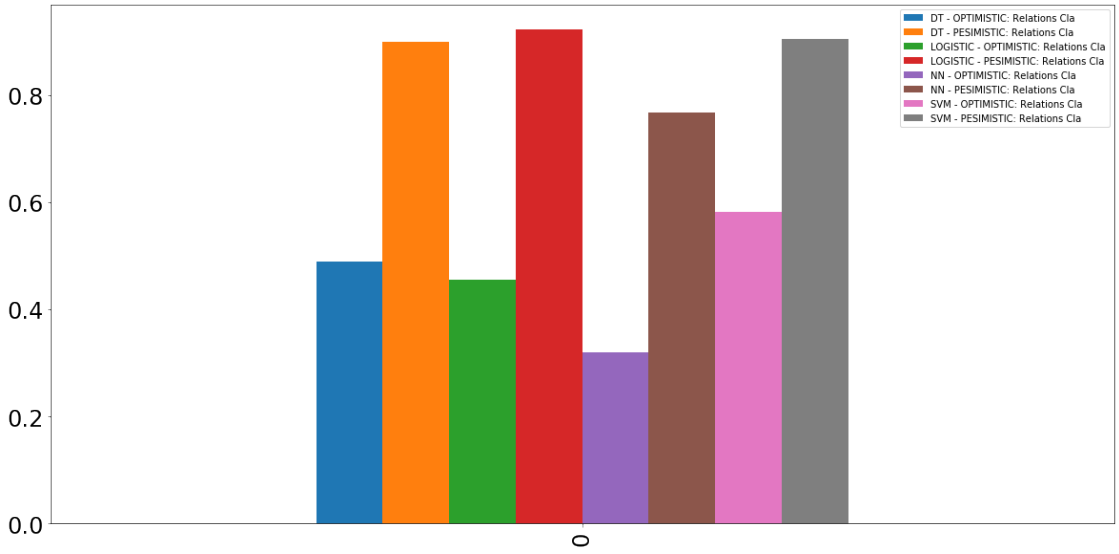
```
            string = str(string[0] + ' --> ' + string[1] + ' --> ' + string[2])
            table['Predicted column'] = [string]
            table['Precision_1'] = rel.precision1
            table['Recall_1'] = rel.recall1
            table['F1_1'] = rel.f1_score1
            table['Precision_2'] = rel.precision2
            table['Recall_2'] = rel.recall2
            table['F1_2'] = rel.f1_score2
            table.to_excel(writer, sheet_name='Sheet1', startrow=row_to_write, index=False)
            row_to_write = row_to_write + 3

        writer.save()
```

1.7 Comparing Algorithms

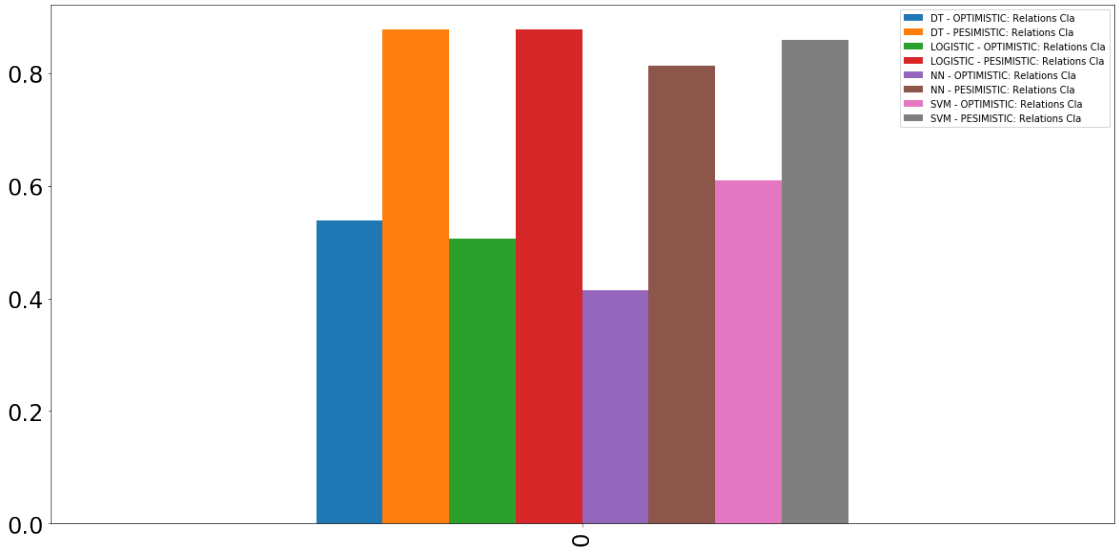
```
In [37]: # Now we make a comparison between algorithms
         name_folders = ['LOGISTIC - OPTIMISTIC: Relations Classification.xlsx',
                         'LOGISTIC - PESIMISTIC: Relations Classification.xlsx',
                         'NN - OPTIMISTIC: Relations Classification.xlsx',
                         'NN - PESIMISTIC: Relations Classification.xlsx',
                         'DT - OPTIMISTIC: Relations Classification.xlsx',
                         'DT - PESIMISTIC: Relations Classification.xlsx',
                         'SVM - OPTIMISTIC: Relations Classification.xlsx',
                         'SVM - PESIMISTIC: Relations Classification.xlsx']

         name_F1_score_dict = dict()
         for name in name_folders:
             df = pd.read_excel('../Summaries/' + name)
             df = df.set_index('Predicted column')
             df = df.drop('Predicted column')
             df = df.dropna()
             F1_score = np.mean(df['Precision_2'])
             name_F1_score_dict[name[:-16]] = F1_score

         params = {'axes.titlesize':'64',
                   'xtick.labelsize':'24',
                   'ytick.labelsize':'24'}
         matplotlib.rcParams.update(params)

         df_hist = pd.DataFrame(name_F1_score_dict, index=[0])
         df_hist.plot.bar(figsize=(20,10))
         plt.savefig('../Figures/Classification histogram comparison Precision.png')
         plt.show()
```



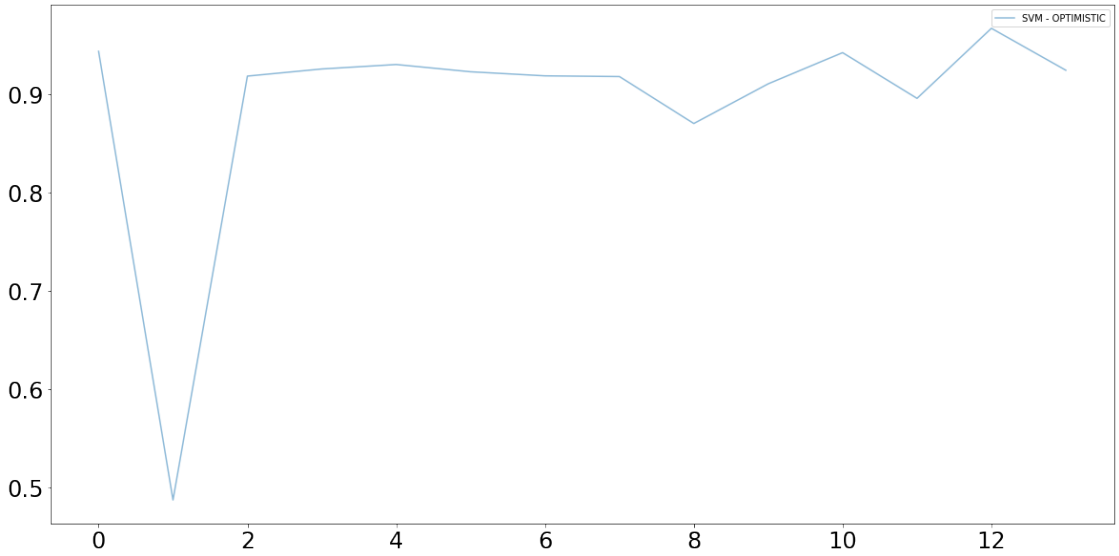
In [38]:
```
# Now we make a comparison between algorithms
name_folders = ['LOGISTIC - OPTIMISTIC: Relations Classification.xlsx',
                'LOGISTIC - PESIMISTIC: Relations Classification.xlsx',
                'NN - OPTIMISTIC: Relations Classification.xlsx',
                'NN - PESIMISTIC: Relations Classification.xlsx',
                'DT - OPTIMISTIC: Relations Classification.xlsx',
                'DT - PESIMISTIC: Relations Classification.xlsx',
                'SVM - OPTIMISTIC: Relations Classification.xlsx',
                'SVM - PESIMISTIC: Relations Classification.xlsx']

name_F1_score_dict = dict()
for name in name_folders:
    df = pd.read_excel('../Summaries/' + name)
    df = df.set_index('Predicted column')
    df = df.drop('Predicted column')
    df = df.dropna()
    F1_score = np.mean(df['Recall_2'])
    name_F1_score_dict[name[:-16]] = F1_score

params = {'axes.titlesize':'64',
          'xtick.labelsize':'24',
          'ytick.labelsize':'24'}
matplotlib.rcParams.update(params)

df_hist = pd.DataFrame(name_F1_score_dict, index=[0])
df_hist.plot.bar(figsize=(20,10))
plt.savefig('../Figures/Classification histogram comparison Recall.png')
plt.show()
```



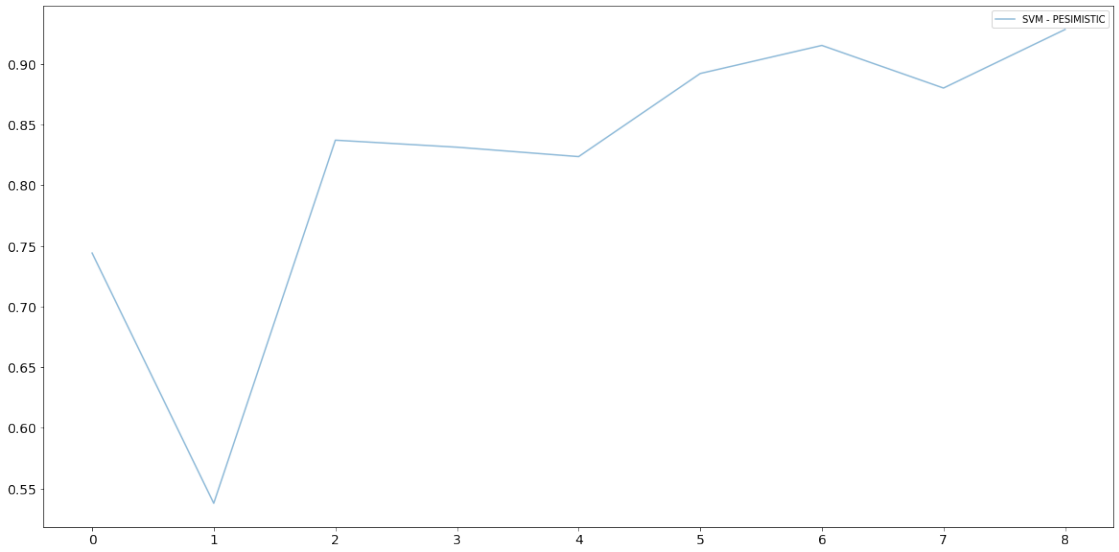
In [39]:
```
# Now we make a comparison between algorithms
name_folders = ['LOGISTIC - OPTIMISTIC: Relations Classification.xlsx',
                'LOGISTIC - PESIMISTIC: Relations Classification.xlsx',
                'NN - OPTIMISTIC: Relations Classification.xlsx',
                'NN - PESIMISTIC: Relations Classification.xlsx',
                'DT - OPTIMISTIC: Relations Classification.xlsx',
                'DT - PESIMISTIC: Relations Classification.xlsx',
                'SVM - OPTIMISTIC: Relations Classification.xlsx',
                'SVM - PESIMISTIC: Relations Classification.xlsx']

name_F1_score_dict = dict()
for name in name_folders:
    df = pd.read_excel('../Summaries/' + name)
    df = df.set_index('Predicted column')
    df = df.drop('Predicted column')
    df = df.dropna()
    F1_score = np.mean(df['F1_2'])
    name_F1_score_dict[name[:-16]] = F1_score

params = {'axes.titlesize':'64',
          'xtick.labelsize':'24',
          'ytick.labelsize':'24'}
matplotlib.rcParams.update(params)

df_hist = pd.DataFrame(name_F1_score_dict, index=[0])
df_hist.plot.bar(figsize=(20,10))
plt.savefig('../Figures/Classification histogram comparison F1 score.png')
plt.show()
```



In [40]:
```
name_folders = ['SVM - OPTIMISTIC: Relations Classification.xlsx']

name_F1_score_dict = dict()
for name in name_folders:
    df = pd.read_excel('../Summaries/' + name)
    df = df.set_index('Predicted column')
    df = df.drop('Predicted column')
    df = df.dropna()
    if name[-41:-31] == 'OPTIMISTIC':
        F1_score = np.array(df['Recall_1'])
    else:
        F1_score = np.array(df['Recall_2'])
    name_F1_score_dict[name[:-31]] = list(F1_score)


df_area = pd.DataFrame(name_F1_score_dict)
df_area.plot.line(figsize=(20,10), stacked=False, alpha=0.5)
plt.savefig('../Figures/area comparison classification F1 Optimistic')
plt.show()
```



In [41]:
```
name_folders = ['SVM - PESIMISTIC: Relations Classification.xlsx']

name_F1_score_dict = dict()
for name in name_folders:
    df = pd.read_excel('../Summaries/' + name)
    df = df.set_index('Predicted column')
    df = df.drop('Predicted column')
    df = df.dropna()
    if name[-41:-31] == 'OPTIMISTIC':
        F1_score = np.array(df['Precision_1'])
    else:
        F1_score = np.array(df['Precision_2'])
    name_F1_score_dict[name[:-31]] = list(F1_score)

params = {'axes.titlesize':'24',
          'xtick.labelsize':'14',
          'ytick.labelsize':'14'}
matplotlib.rcParams.update(params)

df_area = pd.DataFrame(name_F1_score_dict)
df_area.plot.line(figsize=(20,10), stacked=False, alpha=0.5)
plt.savefig('../Figures/area comparison classification F1 pesimistic')
plt.show()
```



1.8 Interpreting the model

```
In [42]: # SVM - optimistic
         df = df_optimistic.copy()

         # Turn up tolerance for faster convergence
         model = svm.SVC()
         support_vectors = []
         for lvl0, lvl1, lvl2 in df.iloc[:,3:].columns:
             Y = df[lvl0][lvl1][lvl2].copy()
             X = df.copy()
             X = X.iloc[:,3:].drop(lvl0, level=0, axis=1)
             X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33)
             model.fit(X_train,Y_train)
             model.support_

In [43]: support_vectors

Out[43]: []

In [44]: df_coefs = df.copy()
         df_coefs = pd.DataFrame(columns=df_coefs.columns, index=[i for i in range(1,5)])
         df_coefs = df_coefs.drop(('Sexe', 'Sexe', 'Sexe'), axis=1)
         df_coefs = df_coefs.drop(('Data Naixament', 'Data Naixament', 'Data Naixament'), axis=1
         df_coefs = df_coefs.drop(('Diagnòstic', 'Diagnòstic', 'Diagnòstic'), axis=1)

         i = 0
         for rel in relations:
             i = i + 1
```

```
            j = 0
            df_coefs_rel = df_coefs.copy()
            predicted_column = rel.name_predicted_column
            columns = df_coefs_rel.columns.drop(predicted_column[0],level=0)
            for column in columns:
                df_coefs.loc[i,column] = rel.weight[j]
                df_coefs.loc[i,predicted_column] = 'X'
                j = j + 1

        df_coefs.insert(0, column=('Precision_1','Precision_1','Precision_1'), value=0)
        df_coefs.insert(0, column=('Recall_1','Recall_1','Recall_1'), value=0)
        df_coefs.insert(0, column=('F1_1','F1_1','F1_1'), value=0)
        df_coefs.insert(0, column=('Precision_2','Precision_2','Precision_2'), value=0)
        df_coefs.insert(0, column=('Recall_2','Recall_2','Recall_2'), value=0)
        df_coefs.insert(0, column=('F1_2','F1_2','F1_2'), value=0)

        i = 0
        for rel in relations:
            i = i + 1
            df_coefs.loc[i,'MAE'] = rel.MAE
            df_coefs.loc[i,'MAPE'] = rel.MAPE*100
            df_coefs.loc[i,'RMSE'] = rel.RMSE


    ---------------------------------------------------------------------------

    IndexError                                Traceback (most recent call last)

    <ipython-input-44-769d65625c16> in <module>()
     13     columns = df_coefs_rel.columns.drop(predicted_column[0],level=0)
     14     for column in columns:
---> 15         df_coefs.loc[i,column] = rel.weight[j]
     16         df_coefs.loc[i,predicted_column] = 'X'
     17         j = j + 1


    IndexError: list index out of range


In [ ]: pd.set_option('display.height', 1000)
        pd.set_option('display.max_rows', 500)
        pd.set_option('display.max_columns', 500)
        pd.set_option('display.width', 1000)

In [ ]: df_coefs = df_coefs.fillna('-')
        df_coefs

In [ ]: rel.weight
```