MEMORIA: Accident Rate Project

KSCHOOL 26 Ed MASTER DATA SCIENCE – Madrid July 2021

By: Sara Sabino Martínez

An incidence rate (also known as an accident rate) is a way of measuring the safety of a workplace. The rate is calculated based on the number of accidents that happens during a particular period of time.

The Occupational Safety and Health Agency (OSHA), the federal agency that establishes safety regulations for workplaces in US, has a formula for calculating incidence rates. To determine the accident rate in a workplace, start by multiplying the number of accidents by 200,000. Then divide the result by the number of labour hours.

Accidents directly impact two crucial factors for business: money and reputation. Companies have been trying to reduce their injuries rates during the last years, indeed incidence rates have fallen 75% since 1972. Although this good metric there is still a lot of work to do. In 2019, employees in the US suffered 2.8 million workplace injuries, in Spain 1.3 million accidents occurred during 2019.

Objective

The main objectives of this project are, in first place, to understand and analyse the data to reach to a conclusion that allows the company to broad the knowledge and, to predict which employees are going to have an accident to be able to avoid those situations.

This prediction will help companies to avoid those dangerous situations by being more aware of them and which factors influence more.

Instructions and structure

In order to run the data please download the notebooks contained in this repository.

Each notebook contains at the top a cell with the code dependencies and version used during its execution. The data needed is in the following shared folder: https://drive.google.com/drive/folders/1pzxQWsVrMSZhSZ3pHYQtUpX33XMRjN9g?usp=sharing

The front end app is develop in streamlit and the link is shared below.

Above a description of the structure of the repo and the data folder is described:

Repo Structure

```
├── Data mode
│    ├── G_Plantas y Tech_streamlit.csv
│    ├── Total_staff_by_employee.csv
│    ├── random_forest_model
│    └── staff_encoded.csv
│
├── Images
│
├── src / Notebooks
│    ├── 00_Data_generation.ipynb
│    ├── 01_Initial_Analysis.ipynb
│    ├── 02_Exploratory_Staff_data.ipynb
│    ├── 02_1_Exploratory_final_data.ipynb
│    ├── 03_Correlation_analysis_&_feature_selection.ipynb
│    ├── 04_Model_selection.ipynb
│    ├── 05_streamlit_app.py
│    └── requirements.txt
│
├── Memoria_Accident_Rate.pdf
└── README.md
```

Shared Folder structure

```
├── Data
│    ├── HS_Accidentabilidad.csv
│    ├── G_Plantas y Tech.csv
│    ├── G_Plantas y Tech_streamlit.csv
│    ├── Datos_plantilla.xlsx
│    ├── Total_staff_by_employee.csv
│    ├── random_forest_model
└── └──staff_encoded.csv
```

Starting point & random data generation

The data received at first place contained a table of employees who have had an accident in a particular company and the details associated to those employees. All this data were anonymized for protection purposes.

As in this Project the idea was to study which employees are going to have an accident, we needed to have the entire staff to feed the model. The idea was to keep all those rows with the employees who have suffered an accident and generate the other employees which have not suffered one.

Based on the number we had about the proportion of that company we have generated the other part the staff.

This table was processed and cleaned in the notebooks to get a clean dataset to feed the algorithms.

During the feature selection to prepare the data it was realized that we have an unbalanced dataset that will need to be balanced to achieve better results in the training. As we have a huge dataset, and to not loss information, we performed the over sample technique, completing our dataset with accidents that have already occurred in the same circumstances for an employee.

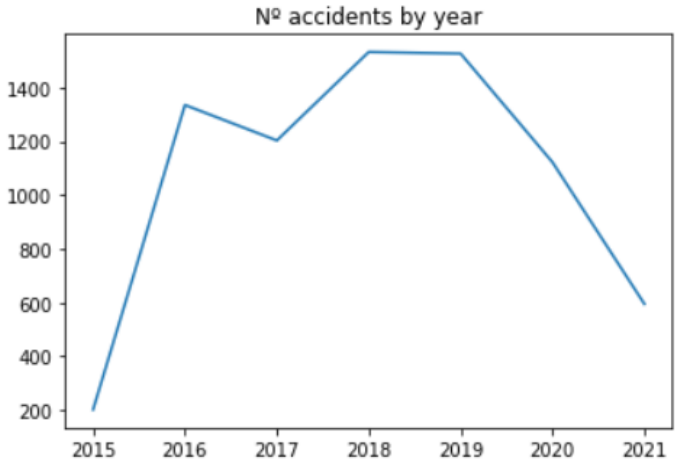
All these data stages we had have been analysed to be able to understand the staff characteristics and have a general knowledge to have a better criterion to perform the models.

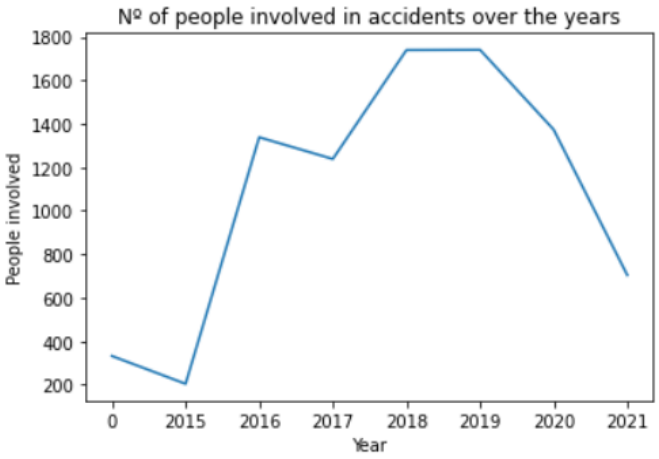
Analysis conclusions

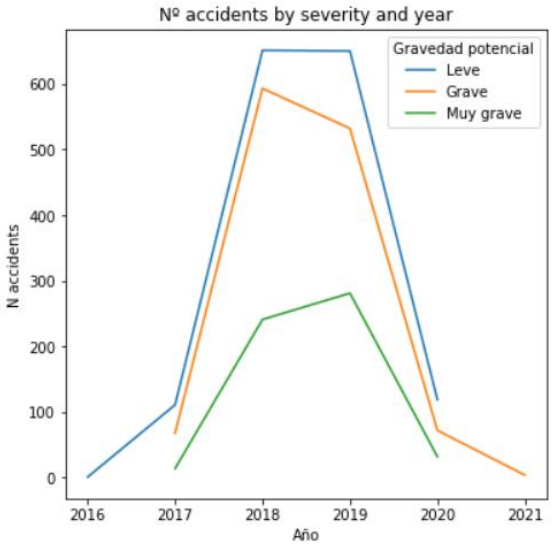
We are going to describe below the most important points that have been discovered during the data analysis.

Accidents

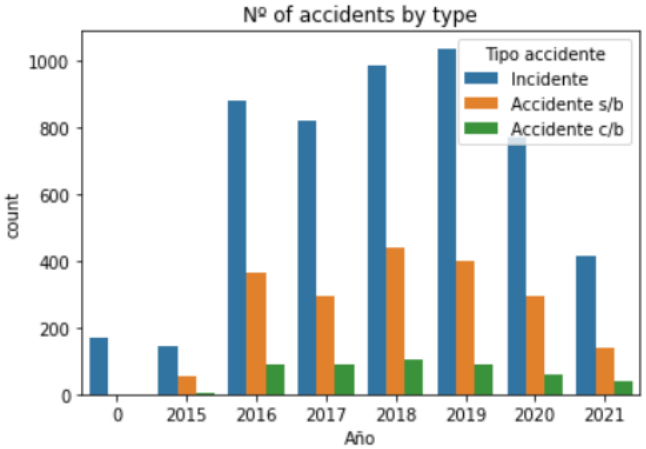
The number of accidents has been decreasing since 2019.



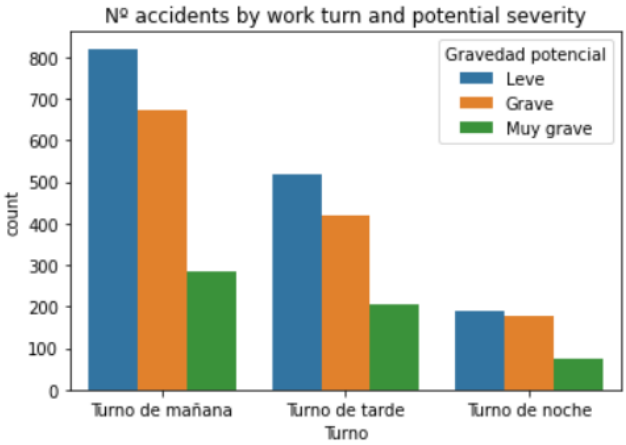




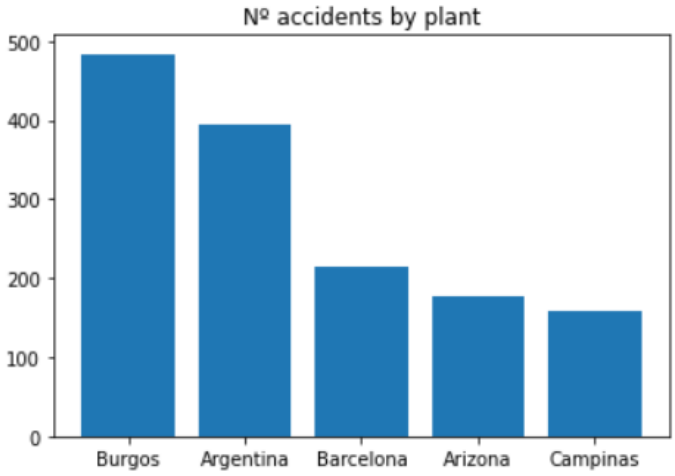
The most common type are the Incidents which are the ones with low severity.
Accidente s/b means accidents without medical leave
Accidente c/b means accidents with medical leave. This are the less common type.



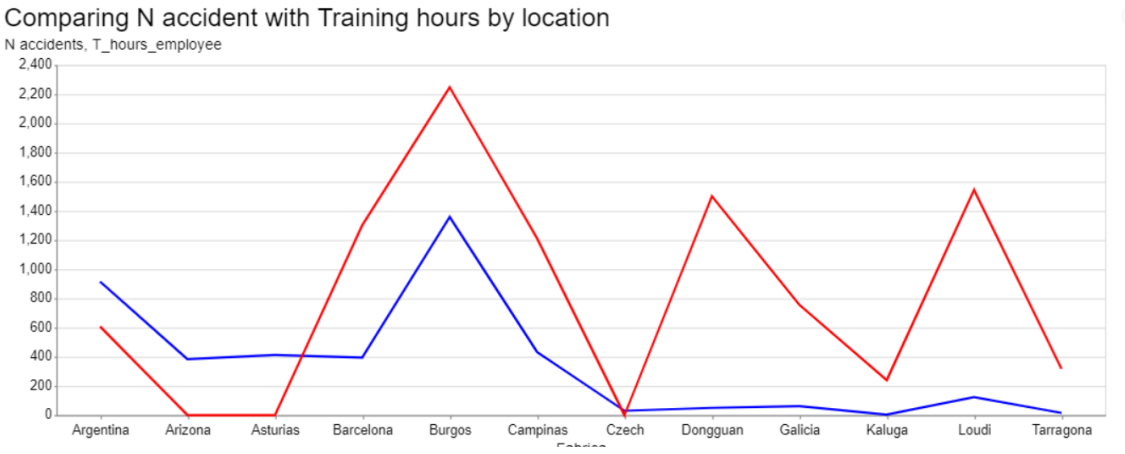
The work turn with more accidents is the morning turn:



The locations with more accidents are Argentina and Burgos (Spain):

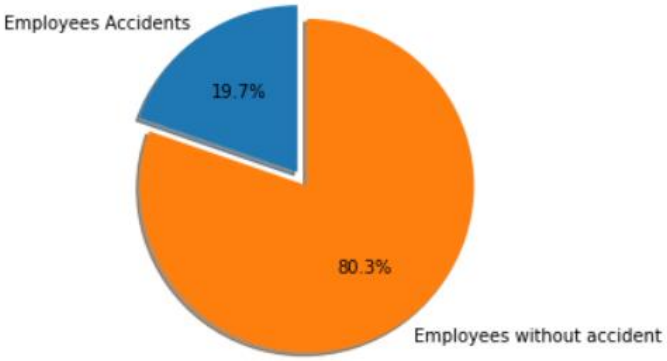


The blue line represents the Nº of accidents, and the red one represents the nº training hours. We can see that the company has invested a huge number of training hours in Burgos location as it is the one with more accidents. They could reduce the training in the locations with less accidents and invest in Argentina which seems to need it more.

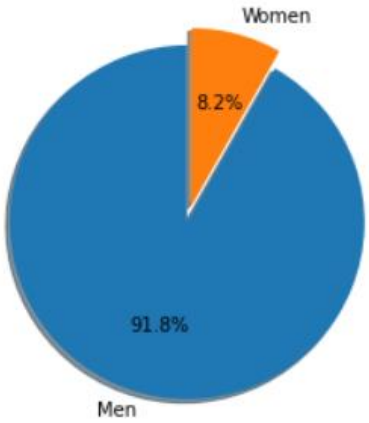


Staff

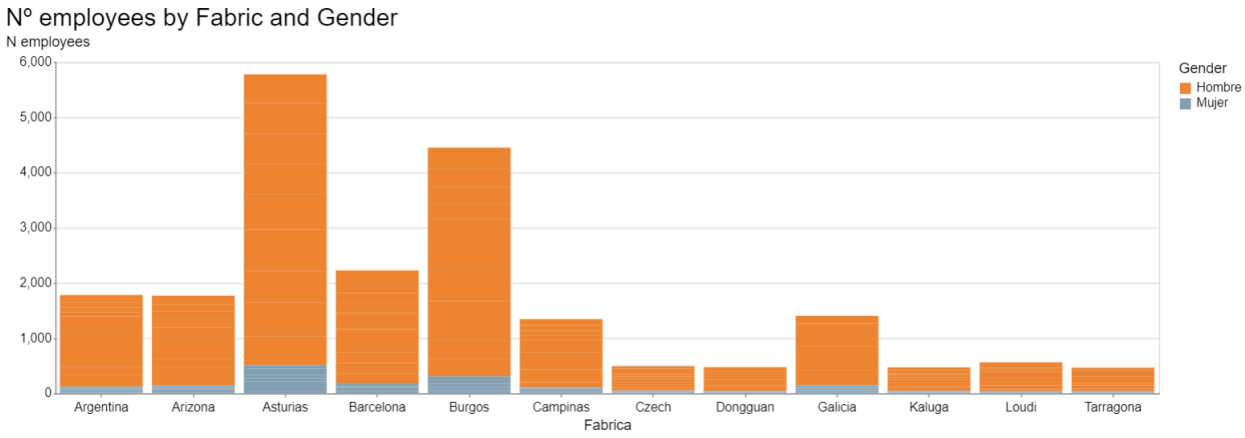
With a total of 21.210 employees, a 19.7% has suffered an accident during its time in time company.



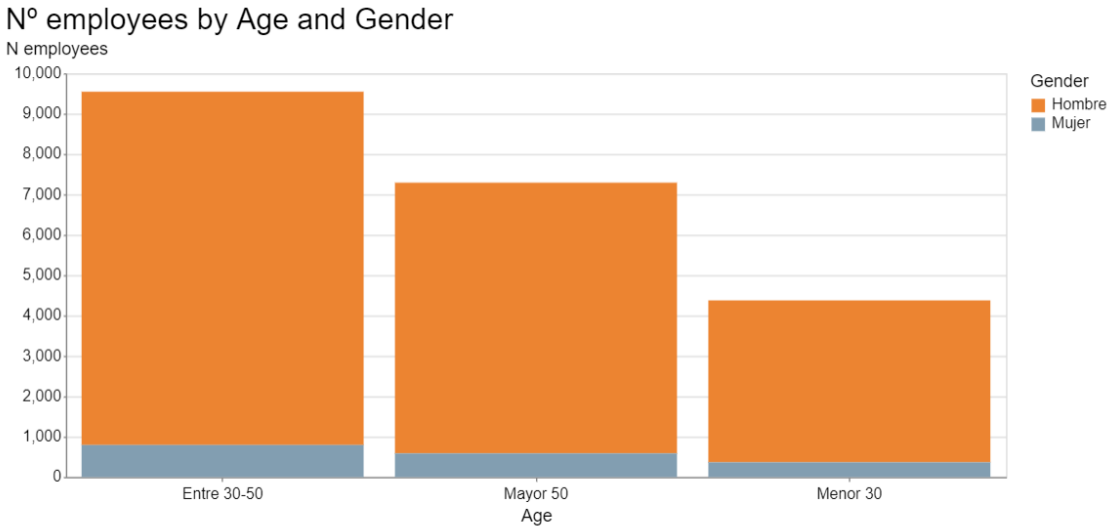
Men represents a 91.8% of staff employees:



Asturias and Burgos are the locations with more employees:



The predominant age range is between 30-50 years old



Modeling

Seeking best Machine Learning model

We are going to try several machine learning models and see which one has better metrics when predicting if an employee is going to have an accident or not. Below we are going to detail the models and its parameters as well as its results.

To compare the results between models we are going to analyse the following metrics:

- Accuracy: is the number of correctly predicted data points out of all the data points.
- Precision: is the ratio of correctly predicted positive observations to the total predicted positive observations.

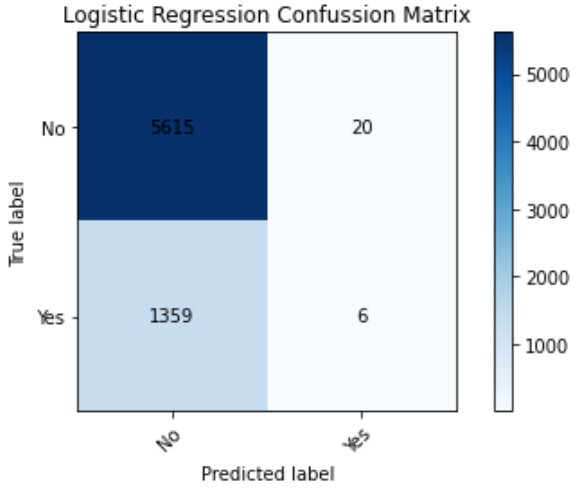
Both metrics are important for our models as we need to be sure that we have a good precision in both classes, it is crucial to identify those 'yes' classes.

Logistic Regression

At first, we tried to run the model with the unbalanced data to check how the model Will behave. The results were a Little poor because the precision of the “yes” class was low.

- accuracy: 0.80

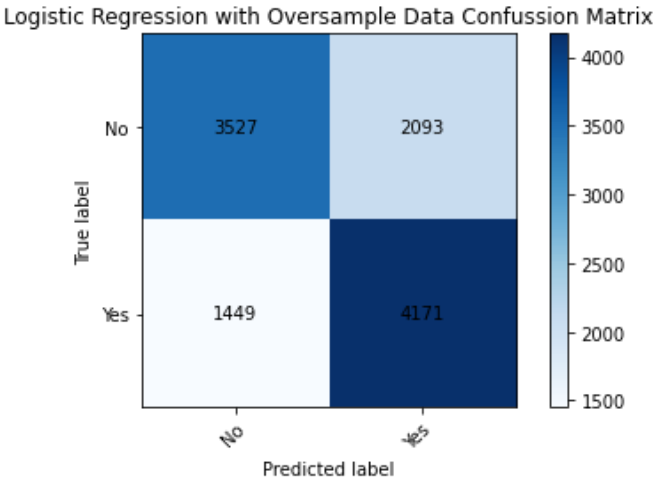
	precision	recall	f1-score	support
No	0.81	1.00	0.89	5635
Yes	0.23	0.00	0.01	1365
accuracy			0.80	7000
macro avg	0.52	0.50	0.45	7000
weighted avg	0.69	0.80	0.72	7000



As these results are not valid, because it is very important for us to predict when an accident is going to happen (yes), we are going to perform the model again but with the balanced dataset this time.

- accuracy: 0.68

	precision	recall	f1-score	support
No	0.71	0.63	0.67	5620
Yes	0.67	0.74	0.70	5620
accuracy			0.68	11240
macro avg	0.69	0.68	0.68	11240
weighted avg	0.69	0.68	0.68	11240

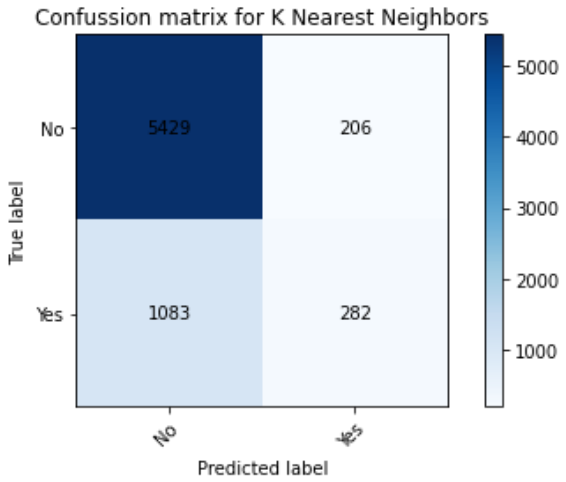


K Nearest Neighbours

Performing the model with the unbalanced dataset

- accuracy: 0.81

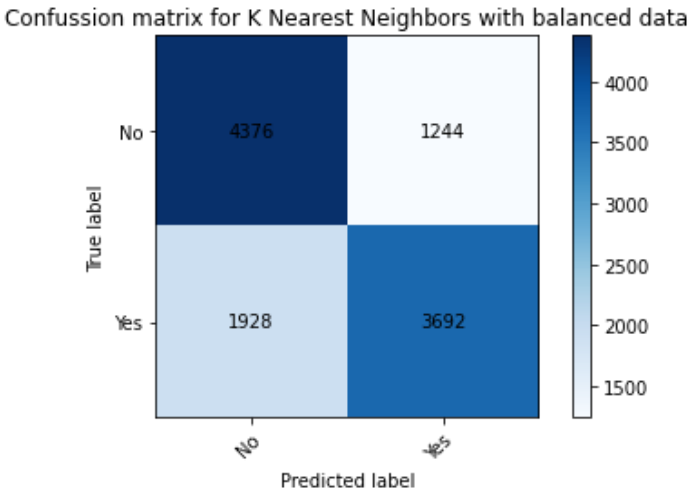
	precision	recall	f1-score	support
No	0.83	0.96	0.89	5635
Yes	0.58	0.21	0.30	1365
accuracy			0.82	7000
macro avg	0.71	0.59	0.60	7000
weighted avg	0.78	0.82	0.78	7000



Although the previous results, were not bad with the unbalanced dataset we are going to try with the balanced one and see if there is any improvement.

- accuracy: 0.71

	precision	recall	f1-score	support
0	0.69	0.78	0.73	5620
1	0.75	0.66	0.70	5620
accuracy			0.72	11240
macro avg	0.72	0.72	0.72	11240
weighted avg	0.72	0.72	0.72	11240



Decision Tree

In order to achieve better results we did a GridSearchCV to find the best hyperparameters for our model. To solve our problem with the minority class 'Yes', we tried to assign proportionally calculated weights to the model but it delivered bad results as well.

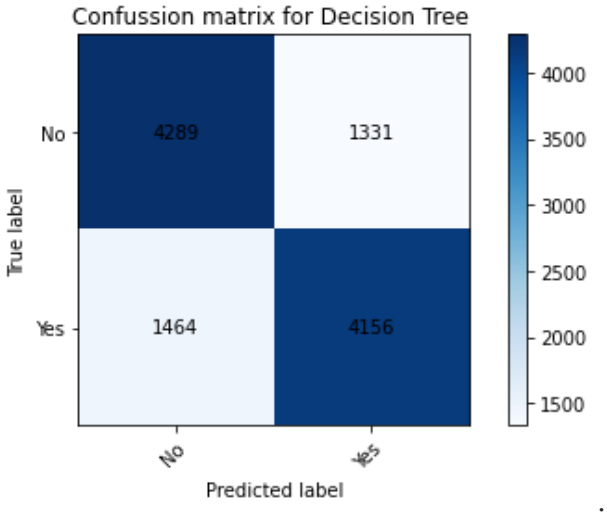
The second option that we tried was to oversample the minority class to balance our classes and achieve better results.

Latest results obtained with the oversample technique and hyperparameter optimization:

- accuracy: 0.75

	precision	recall	f1-score	support
No	0.75	0.76	0.75	5620

Yes	0.76	0.74	0.75	5620
accuracy			0.75	11240
macro avg	0.75	0.75	0.75	11240
weighted avg	0.75	0.75	0.75	11240



Random Forest Algorithm

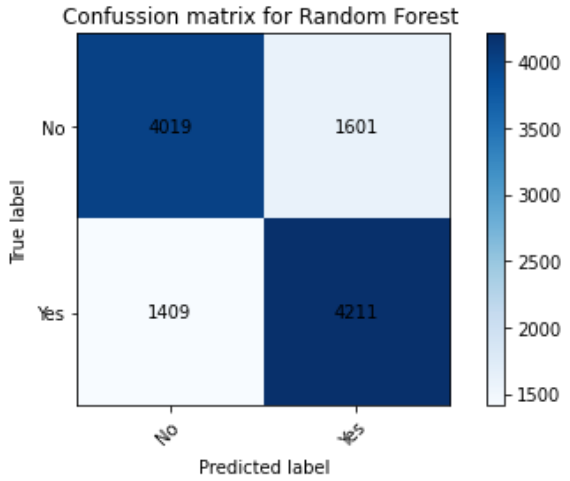
As we have seen that the models needs to be balanced, we have performed the random forest with the balanced data from previous steps.

We have also performed a GridSearchCv to try get the best results we can get, although we could not increase the first results obtained.

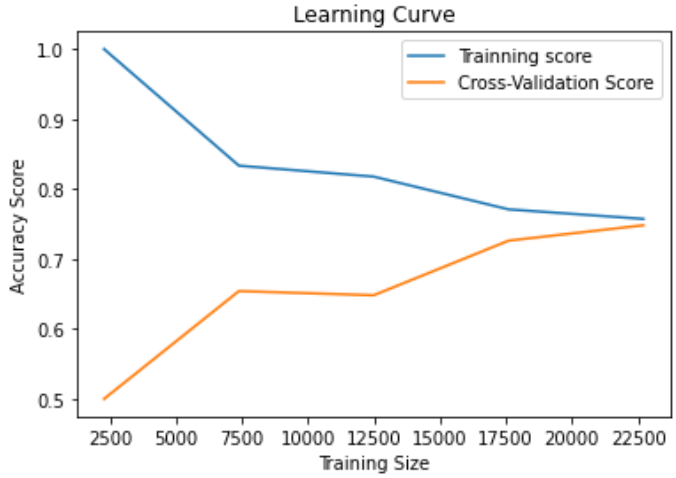
In this model we achieved the following results:

- accuracy: 0.73

	precision	recall	f1-score	support
No	0.74	0.72	0.73	5620
Yes	0.72	0.75	0.74	5620
accuracy			0.73	11240
macro avg	0.73	0.73	0.73	11240
weighted avg	0.73	0.73	0.73	11240



Learning curve of the model:

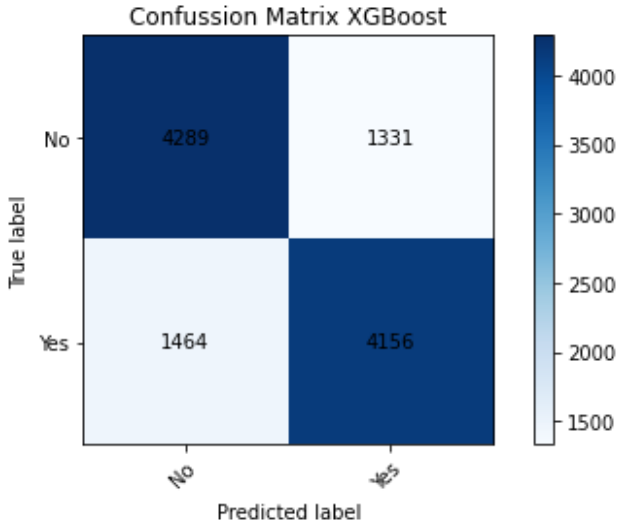


XGBoost

We applied a GridSearchCV to find the best parameters, the results obtained were the following ones:

- accuracy : 0.75

	precision	recall	f1-score	support
No	0.75	0.76	0.75	5620
Yes	0.76	0.74	0.75	5620
accuracy			0.75	11240
macro avg	0.75	0.75	0.75	11240
weighted avg	0.75	0.75	0.75	11240



SVM: Support Vector Machine

As SVM works well with small datasets and not awesome with large ones and it will take forever to optimize with cross validations, we down sample both categories to see how it will perform. We reduced both labels to 2000, having a total dataset of 4000.

We used a Gaussian radial basis function (RBF) kernell because it is commonly use for general-purpose.

The first time we performed this model we achieved the following results:

- accuracy: 0.72

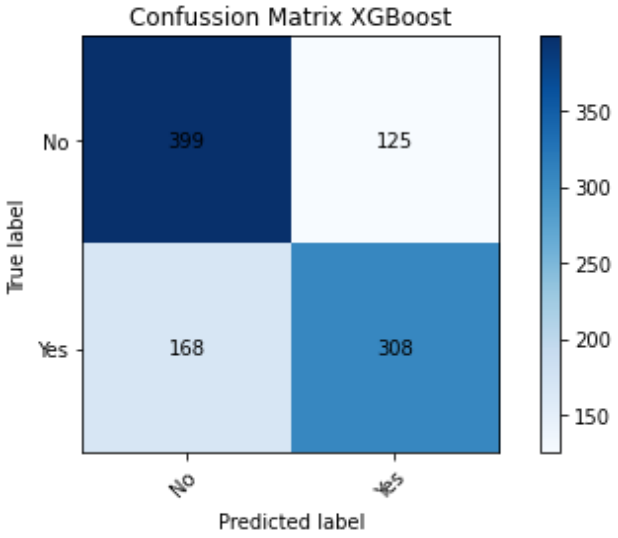
	precision	recall	f1-score	support
No	0.74	0.73	0.73	524
Yes	0.71	0.71	0.71	476
accuracy			0.72	1000
macro avg	0.72	0.72	0.72	1000
weighted avg	0.72	0.72	0.72	1000

Performing the GridSearchCV to find the best parameters we have increase a little bit the model's metrics.

- accuracy: 0.70

	precision	recall	f1-score	support
No	0.70	0.76	0.73	524
Yes	0.71	0.65	0.68	476
accuracy			0.71	1000

```
   macro avg       0.71      0.70      0.70      1000
weighted avg       0.71      0.71      0.71      1000
```



We tried to perform the model changing to a linnear kernell, as it is used when there is many features in the dataset.

Ensemble models

An Ensemble is a combination of machine learning models, each model makes a different prediction, the different predictions get combined to obtain a unique prediction. We are going to try voting and bagging techniques to see if we can increase our predictions.

Voting Classifier

Each model makes a prediction, the final prediction will be the one voted by more models. There are two types of votting classifiers:

- Hard voting classifiers consider the class output and then takes the majority.
- Soft voting classifiers takes a probability score and then averages out the probabilities.

At first we started by trying to combine not too dificult models, we tried Logistic Regression, Random Forest and Naive Bayes. The obtained results compared to the models results were:

```
Accuracy : 0.67 (+/- 0.01) [Logistic Regression]

Accuracy : 0.74 (+/- 0.00) [Random Forest]

Accuracy : 0.59 (+/- 0.01) [Naive Bayes]
```

```
Accuracy : 0.69 (+/- 0.02) [Voting_Classifier_Hard]

Accuracy : 0.71 (+/- 0.00) [Voting_Classifier_Soft]
```

Then in order to try to increase the results we combined Random Forest, XGBoost and Decision Tree. These were the models that achieved better metrics. With the votting classifier techniques the results obtained were the following, compared to the individual models results:

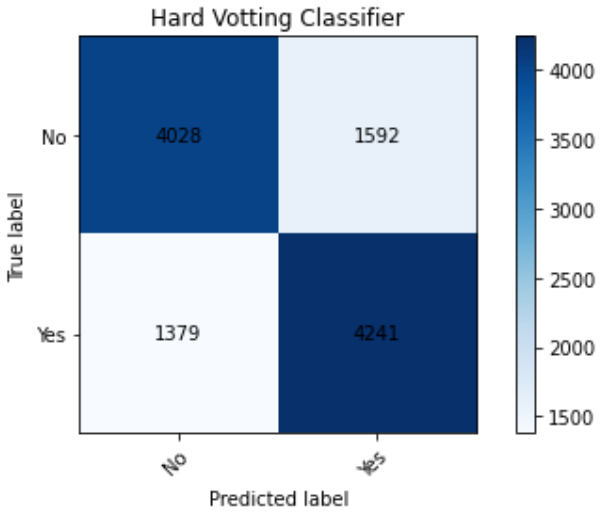
```
Accuracy : 0.74 (+/- 0.00) [XGBoost]
Accuracy : 0.73 (+/- 0.00) [Random Forest]
Accuracy : 0.74 (+/- 0.00) [Decision Tree]
Accuracy : 0.74 (+/- 0.00) [Voting_Classifier_Hard]
Accuracy : 0.74 (+/- 0.00) [Voting_Classifier_Soft]
```

The results from the voting classifiers increase in comparison with the first attempt. Lets see those results in detail:

Hard Votting Classifier

- accuracy : 0.73

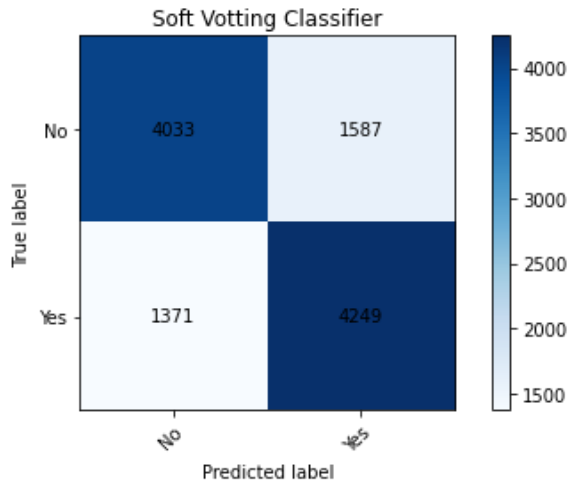
	precision	recall	f1-score	support
No	0.74	0.72	0.73	5620
Yes	0.73	0.75	0.74	5620
accuracy			0.74	11240
macro avg	0.74	0.74	0.74	11240
weighted avg	0.74	0.74	0.74	11240



Soft Votting Classifier

- accuracy : 0.73

	precision	recall	f1-score	support
No	0.75	0.72	0.73	5620
Yes	0.73	0.76	0.74	5620
accuracy			0.74	11240
macro avg	0.74	0.74	0.74	11240
weighted avg	0.74	0.74	0.74	11240



Bagging or Bootstrap Aggregation: Random Forest

This technique fits several models and average their predictions in order to obtain a model with lower variance. We run this technique with the default parameters and obtained a mean accuracy = 0.758 and a std = 0.006. Although we try to improve the results by increasing the parameters to 400 estimators and the ones obtain in the GridSearchCv min_samples_leaf=1, max_depth=9, we didnt obtained a better off. The results were almost the same as the first time:

- Mean accuracy: 0.735
- std: 0.006

Applying Deep Learning: Neural Networks

To evaluate the neural networks models we are going to perform the following metrics:

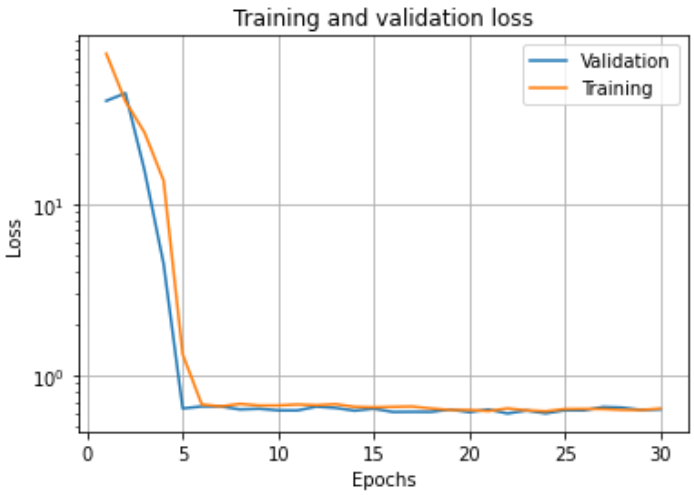
- Categorical accuracy: is the number of correctly predicted data points out of all the data points.
- Loss: determine how far the predicted values deviate from the actual values in the training data.

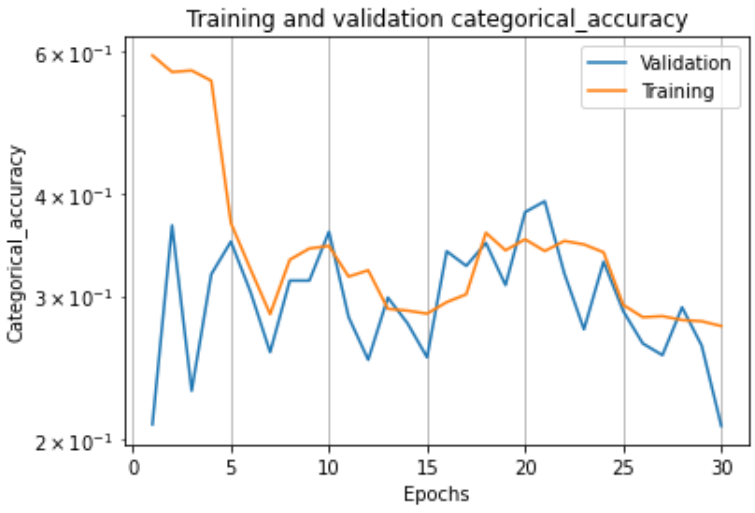
First model used:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 32)                512
_________________________________________________________________
dense_7 (Dense)              (None, 16)                528
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 34
=================================================================
Total params: 1,074
Trainable params: 1,074
Non-trainable params: 0
_________________________________________________________________
```

Before normalizing the data the neural network delivered the following results:

- accuracy: 0.21
- loss: 0.61



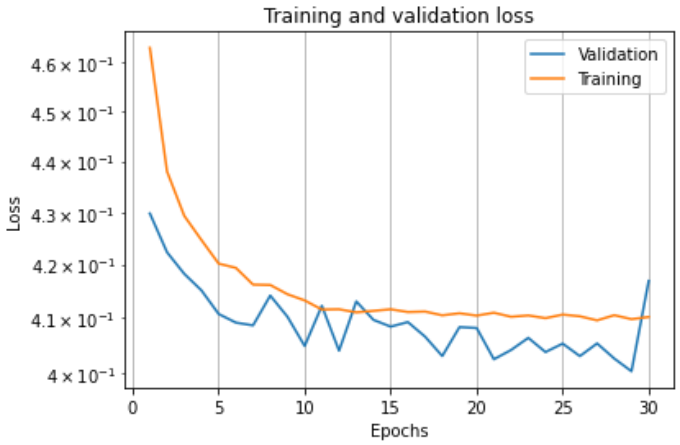


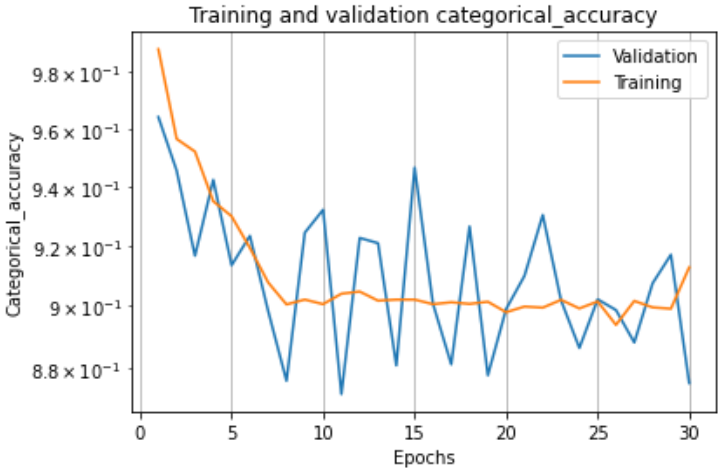
After normalizing the data and use the reduced dataset, we create the following model:

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 32)                416
_________________________________________________________________
dense_10 (Dense)             (None, 16)                528
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 34
=================================================================
Total params: 978
Trainable params: 978
Non-trainable params: 0
_________________________________________________________________
```

In the first and second layer we used 'relu' as activation function and in the third we used 'softmax'. The results obtained the first time with 30 epochs and a batch_size of 10:

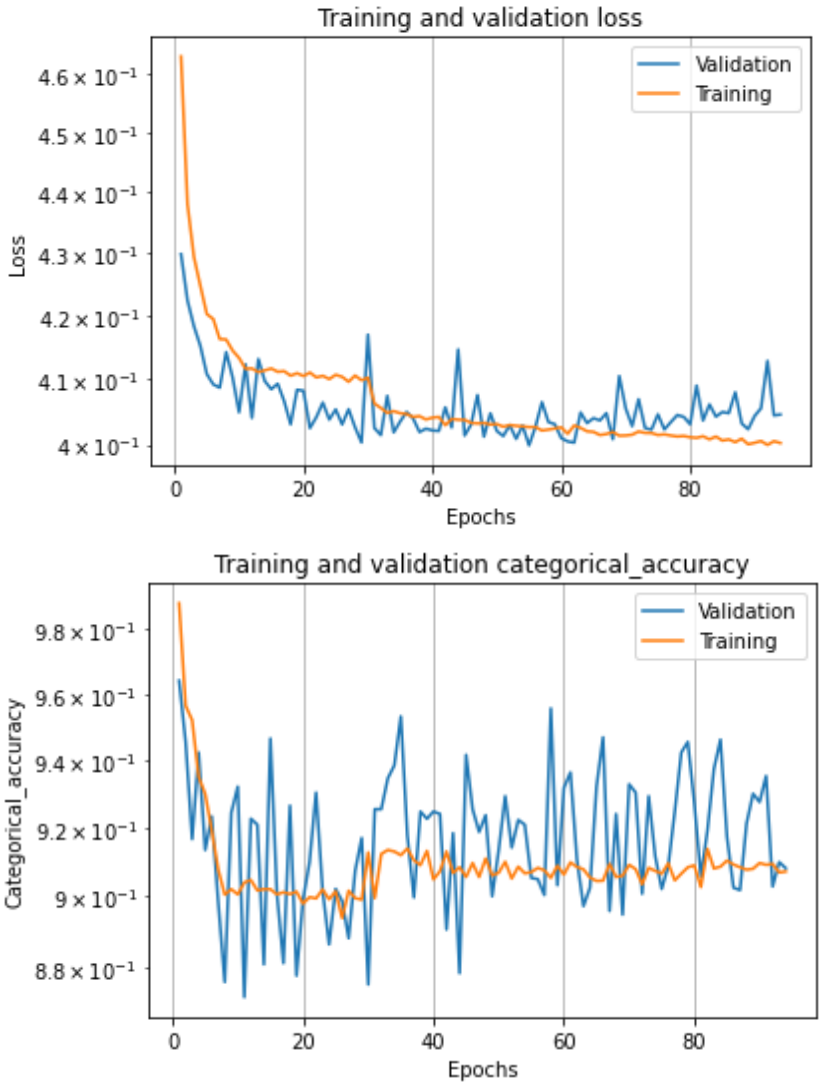
- accuracy: 0.87
- loss: 0.41





The second time we run this model we did it with 64 epochs and a batch_size of 40. We obtained the following results:

- Categorical accuracy: 0.90
- Loss: 0.40



Conclusions and future lines

On the table below we can see a summary of all the results obtained in each model that has been performed.

The best results were achieved by the Decision Tree algorithms and the Neuronal Networks. The worse result was obtained by the Logistic Regression model.

Model	Accuracy	Prec No	Prec Yes
Logistic Regression	0.684875	0.708802	0.665868
K Neighbors	0.717794	0.694162	0.747974
Decision Tree	**0.751335**	0.745524	0.757427
Random Forest	0.732206	0.74042	0.724535
XGBoost	0.751335	0.745524	0.757427
SVM	0.707000	0.703704	0.711316
Soft-votting	0.736833	0.746299	0.728067
Hard-votting	0.735676	0.74496	0.72707
Bagging Classifier	0.734703		
Neuronal Network	**0.908714**		

The future lines for this Project could be, to add more predictions to the application. At this point we can predict if an employee is going to suffer an accident or not, but it could be great to add the prediction of the severity of the accident, or in which place/time/type of work will occur.

It could be also interesting to add the factor "time" to the predictions, and design how the systems is going to be feed with new data. It is not the same to predict if an employee is going to have an accident today that predict it next year.

To conclude the front-end application could be enhance by working on a more user-friendly flow as well as in a more pretty designed interface.

Front-end: Streamlit App

The front-end of this project is develop in a streamlit application, the code can be found in this notebook. The app is thought to be for a client in a company that wants to see a analytical dashboard of the accidents in the company and then is going to have the option to complete a form in order to find if a particular employee is goign to have an accident.

The model used in the application is a Random Forest Model.

https://share.streamlit.io/sarasabino/accident_rate_project/main/Src/Notebooks/05_streamlit_app.py

In addition to the front-end application, in all the notebooks there are charts to illustrate the analysis as well as the training of the models.