MEMORIA: Accident Rate Project

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 happen during a particular period of time.

The Occupational Safety and Health Agency (OSHA), the federal agency that establishes safety regulations for workplaces, 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 labor 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 occured during 2019.

This project aims to help reduce those accidents numbers by predicting if an employee is going to suffer one. This prediction will help companies to avoid those dangerous situations by being more aware of them and which factors influence more.

As well as predicting the accidents/no accident of the employees on our company, an analysis has been made to identify the company structure, workers profile, location etc...

Repo 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
│
├── 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
│
└── README.md
```

Stating point & random data generation

anonimizar datos

Analysis conclusions

Modeling

Feature understanding and selection

need to oversample data

Seeking best Machine Learning model

We are going to try multiple 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.

In order to compare the results between models we are going to analyze 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.

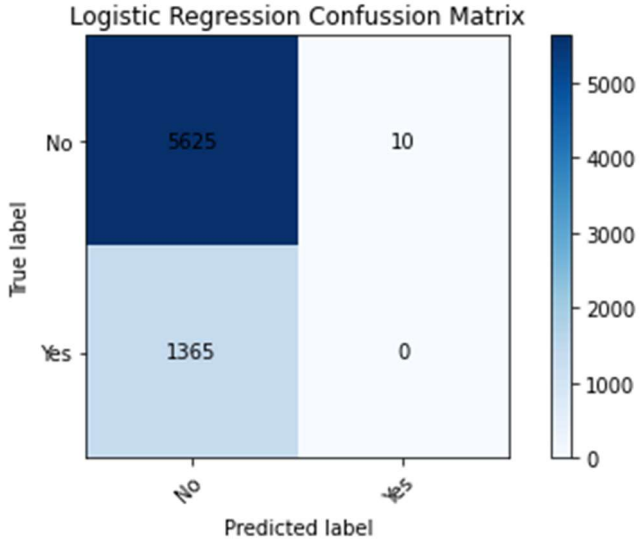
Both metrics are important for our models as we need to be sure that we have a good precision in both classe, it is crucial to identify those 'yes' classes.

Logistic Regression

At first it seems this results were good because the accuracy of the model is 0.89, but if we focus on the performance with the 'Yes' label is 0, so the model is always predicting 'No'. We need to improve that prediction by balancing the dataset and running again the model.

- accuracy: 0.80

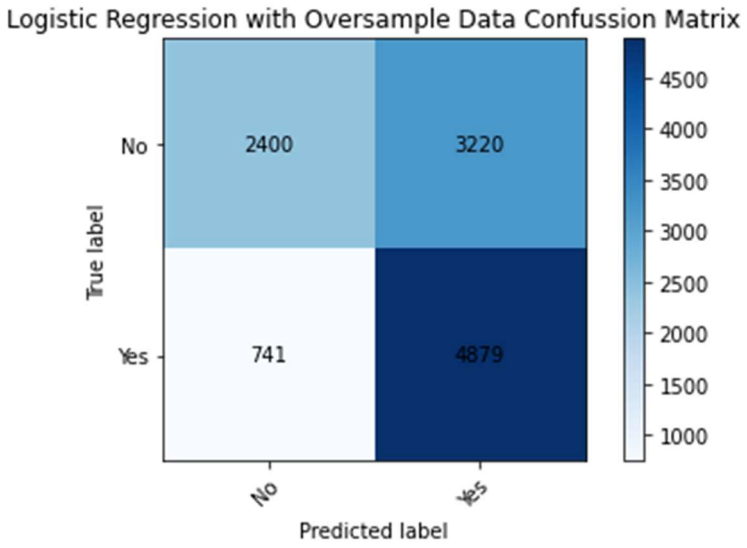
	precision	recall	f1-score	support
No	0.80	1.00	0.89	5635
Yes	0.00	0.00	0.00	1365
accuracy			0.80	7000
macro avg	0.40	0.50	0.45	7000
weighted avg	0.65	0.80	0.72	7000



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

- accuracy: 0.66

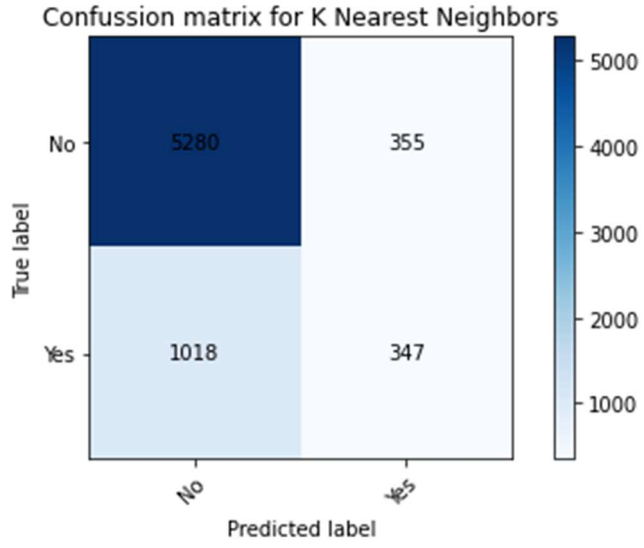
	precision	recall	f1-score	support
No	0.78	0.45	0.57	5620
Yes	0.61	0.87	0.72	5620
accuracy			0.66	11240
macro avg	0.70	0.66	0.65	11240
weighted avg	0.70	0.66	0.65	11240



K Nearest Neighbors

- accuracy: 0.80

	precision	recall	f1-score	support
No	0.84	0.94	0.88	5635
Yes	0.49	0.25	0.34	1365
accuracy			0.80	7000
macro avg	0.67	0.60	0.61	7000
weighted avg	0.77	0.80	0.78	7000

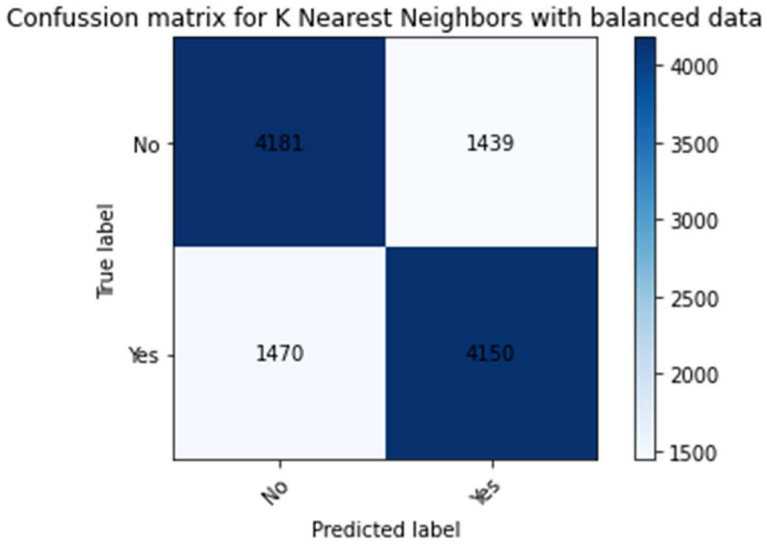


Although the previous results, were not bad with the unbalanced dataset we are going to try with the balanced one and see if there is any improvement.

-accuracy: 0.74

	precision	recall	f1-score	support

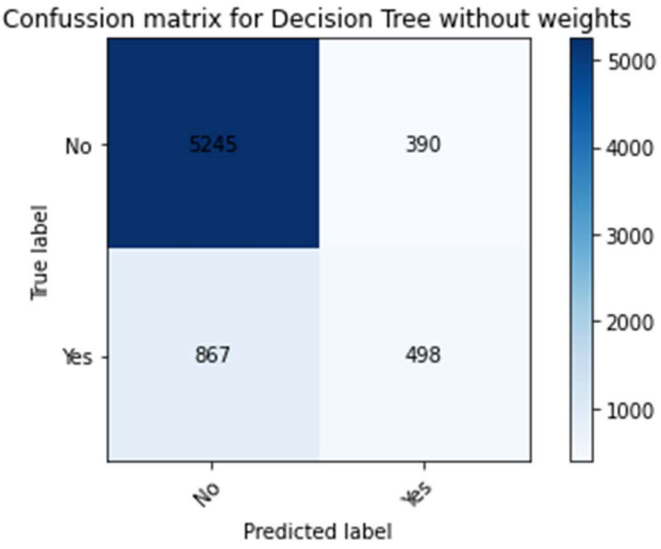
0	0.72	0.77	0.74	5620
1	0.75	0.70	0.73	5620
accuracy			0.73	11240
macro avg	0.74	0.73	0.73	11240
weighted avg	0.74	0.73	0.73	11240



Decision Tree

In order to achieve better results we did a GridSearchCV to find the best hiperparameters 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.

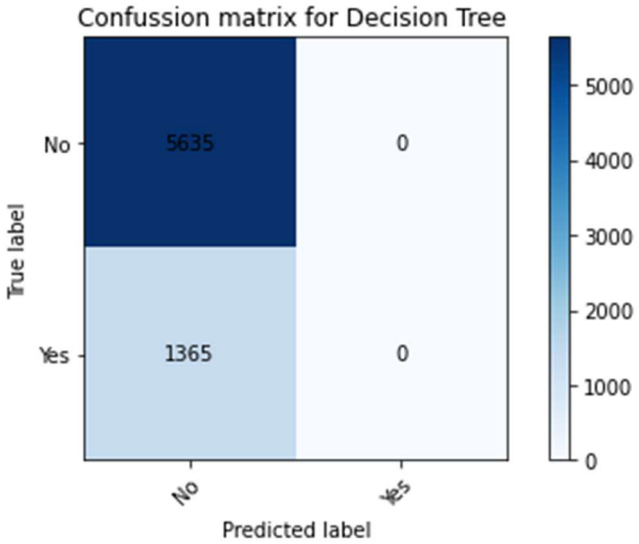
First result obtained:



- accuracy : 0.82

Applying the best parameters by the GridSearchCV and without the balanced dataset: accuracy of 0.8 but with a 0 for the 'Yes' class.

- accuracy : 0.80

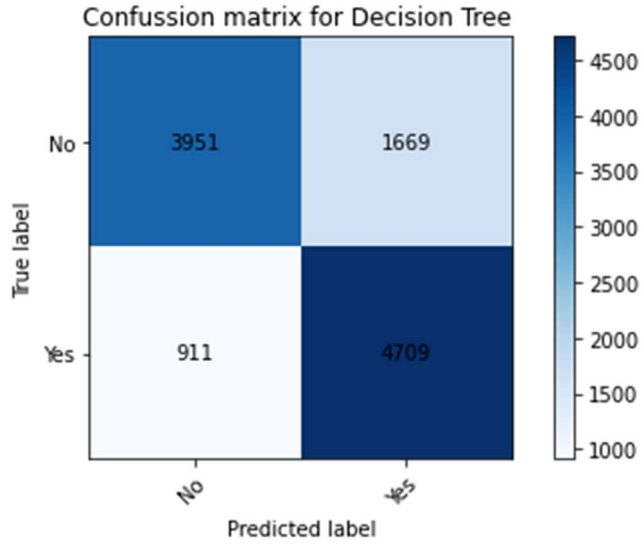


Latest results obtained with the oversample technique and hiperparameter optimization:

- accuracy: 0.77

	precision	recall	f1-score	support
No	0.79	0.73	0.76	5620
Yes	0.75	0.81	0.78	5620
accuracy			0.77	11240
macro avg	0.77	0.77	0.77	11240
weighted avg	0.77	0.77	0.77	11240

Althought the total accuracy of the model has decreased we achieved to got almost the same accuracy for both classes.



Random Forest Algorithm

As we have seen that the models need 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 couldnt increase the first results obtained.

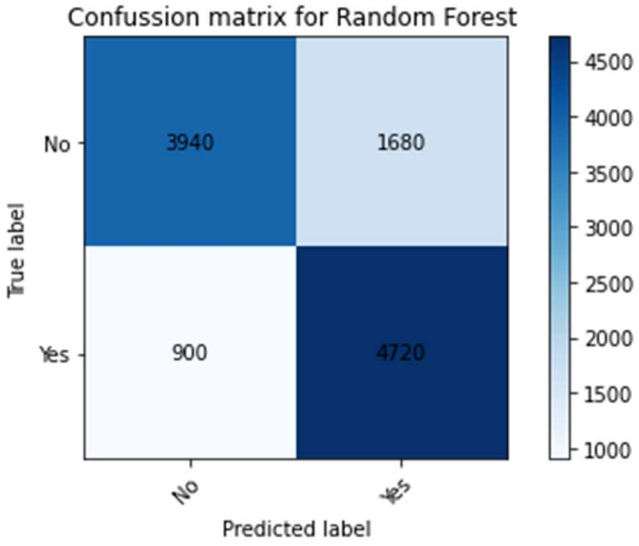
In this model we achieved the following results:

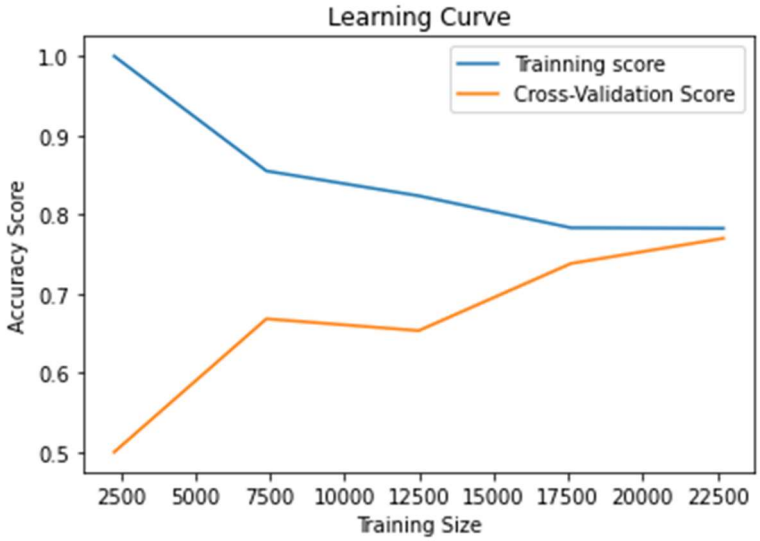
- accuracy: 0.76

```
              precision    recall  f1-score   support

          No       0.79      0.68      0.73      5620
         Yes       0.72      0.82      0.77      5620

    accuracy                           0.75     11240
   macro avg       0.76      0.75      0.75     11240
weighted avg       0.76      0.75      0.75     11240
```



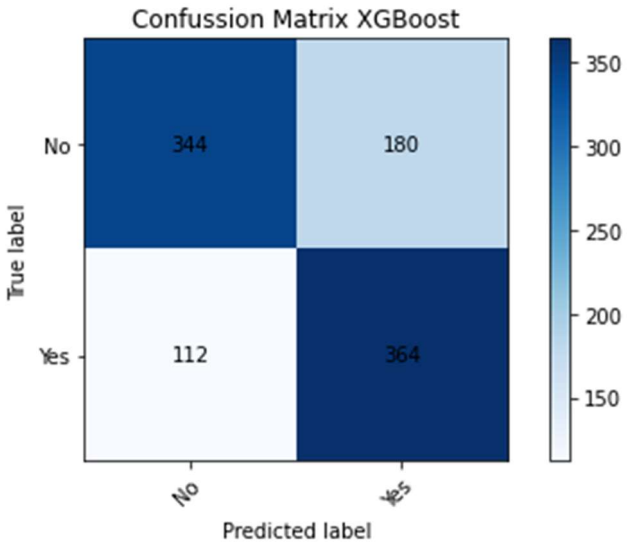


XGBoost

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

- accuracy : 0.77

	precision	recall	f1-score	support
No	0.79	0.71	0.75	5620
Yes	0.74	0.81	0.77	5620
accuracy			0.76	11240
macro avg	0.76	0.76	0.76	11240
weighted avg	0.76	0.76	0.76	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 downsample 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:

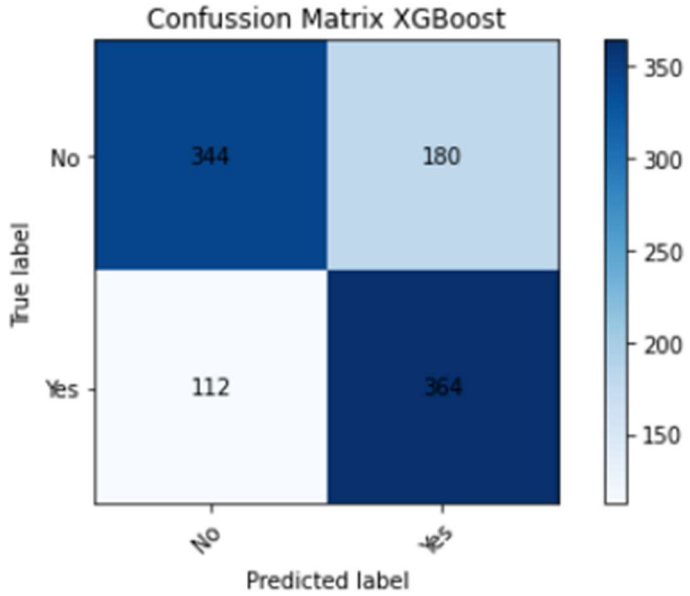
- accuracy: 0.69

	precision	recall	f1-score	support
No	0.75	0.63	0.68	524
Yes	0.65	0.76	0.70	476
accuracy			0.69	1000
macro avg	0.70	0.70	0.69	1000
weighted avg	0.70	0.69	0.69	1000

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

- accuracy: 0.71

	precision	recall	f1-score	support
No	0.75	0.66	0.70	524
Yes	0.67	0.76	0.71	476
accuracy			0.71	1000
macro avg	0.71	0.71	0.71	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 a large number of 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 are able to 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 : 0.66 (+/- 0.00) [Logistic Regression]
Accuracy : 0.76 (+/- 0.01) [Random Forest]
Accuracy : 0.62 (+/- 0.00) [Naive Bayes]
Accuracy : 0.67 (+/- 0.01) [Voting_Classifier_Hard]
Accuracy : 0.74 (+/- 0.00) [Voting_Classifier_Soft]
```

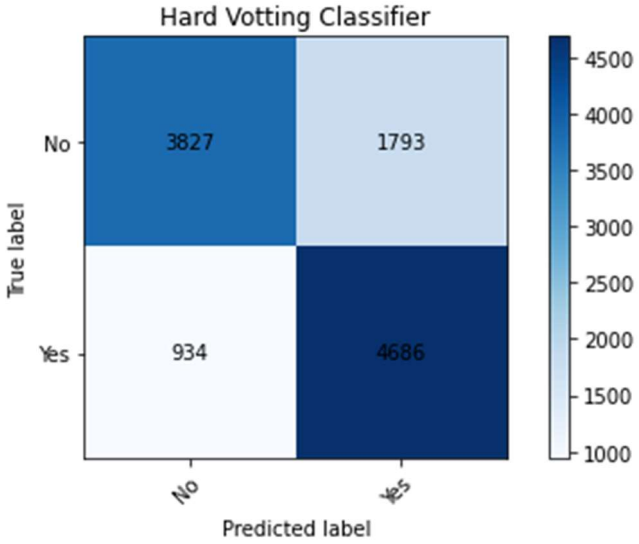
Then in order to try to increase the results we combined Random Forest, XGBoost and Decision Tree. These were the models that achieved better metrics. With the votting classifier techniques the results obtained were the following, compared to the individual models results:

```
Accuracy : 0.76 (+/- 0.01) [XGBoost]
Accuracy : 0.75 (+/- 0.01) [Random Forest]
Accuracy : 0.76 (+/- 0.01) [Decision Tree]
Accuracy : 0.76 (+/- 0.01) [Voting_Classifier_Hard]
Accuracy : 0.76 (+/- 0.01) [Voting_Classifier_Soft]
```

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

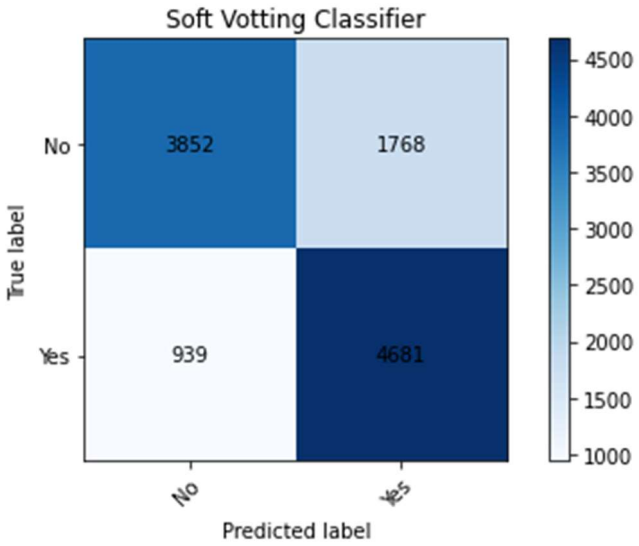
Hard Votting Classifier

- accuracy : 0.75
- precision 'Yes': 0.72
- precision 'No': 0.80



Soft Votting Classifier

- accuracy : 0.75
- precision 'Yes': 0.73
- precision 'No': 0.80



Bagging or Bootstrap Aggregation : Random Forest

This technique fits several models and average their predicitions 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: 0.758
- std: 0.006

Applying Deep Learning: Neural Networks

In order 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 trainning data.

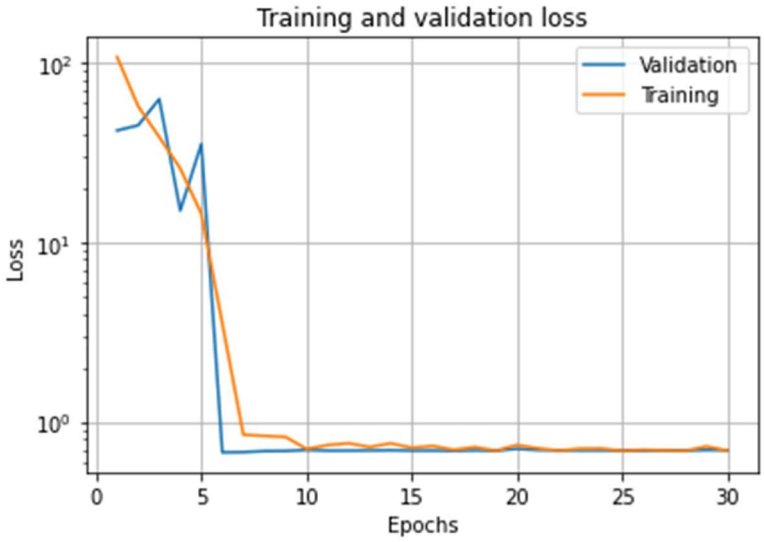
First model use:

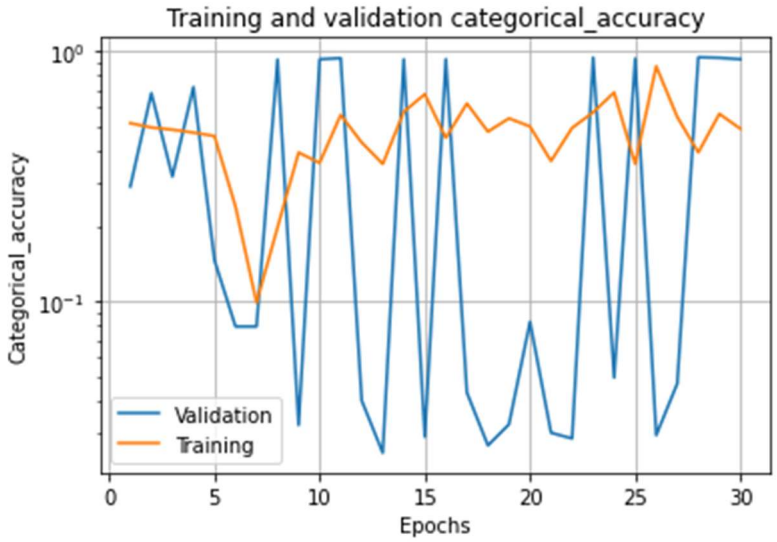
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                512
_________________________________________________________________
dense_1 (Dense)              (None, 16)                528
_________________________________________________________________
```

```
dense_2 (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: 0.92
- loss: 0.69





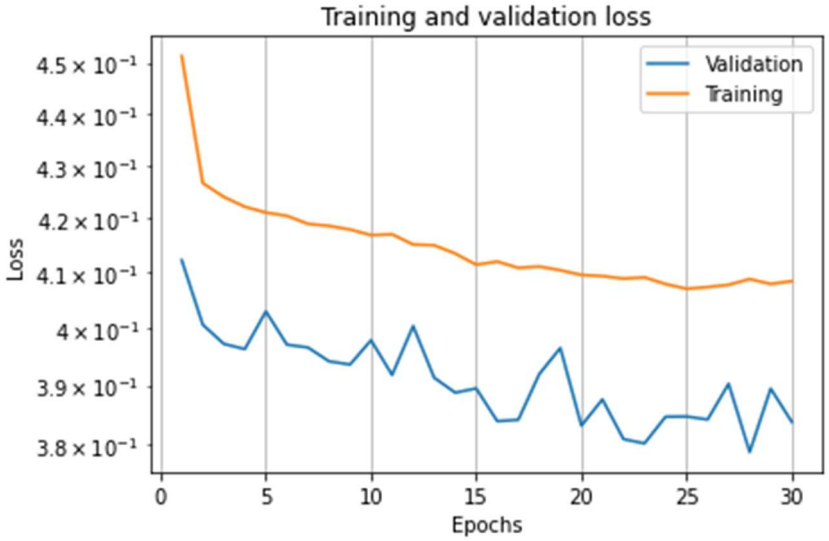
After normalizing the data and use the reduced dataset, we create the following model:

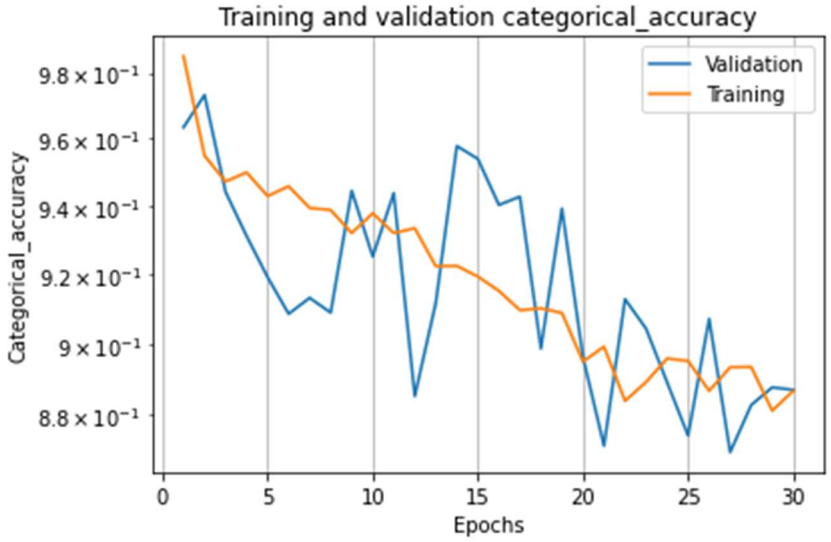
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 32)                416
_________________________________________________________________
dense_4 (Dense)              (None, 16)                528
```

```
_________________________________________________________________
dense_5 (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:

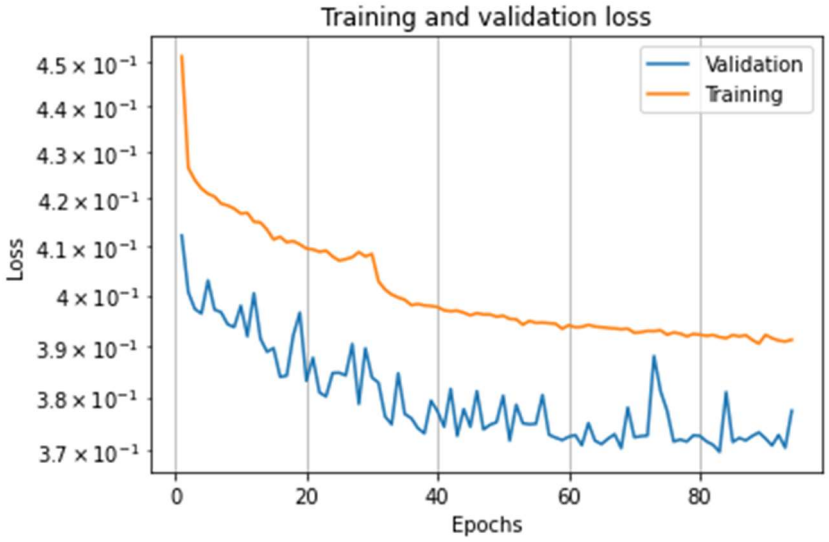
- accuracy: 0.88
- loss: 0.40

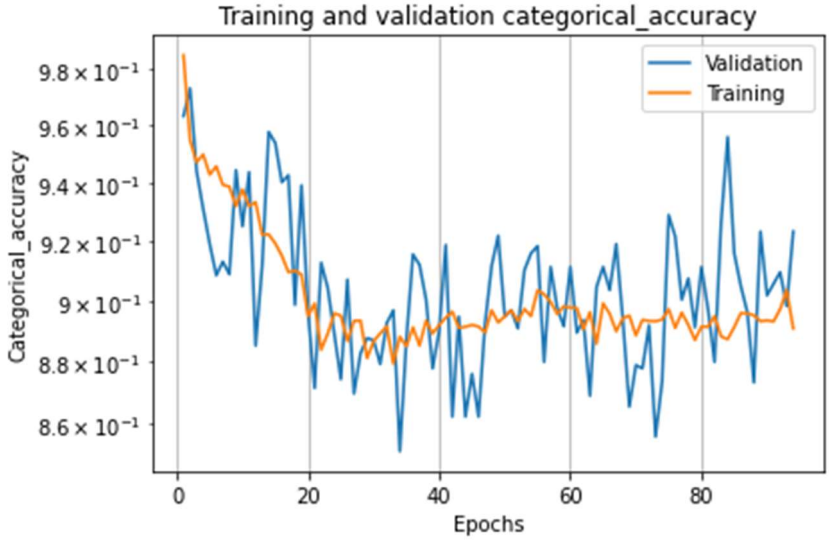




The second time we run this model we did it with 64 epochs and a batch_size of 40. We obtained the following results:

- Categorical accuracy: 0.92
- Loss: 0.40





Front-end: Streamlit App

Link to the Application:
https://share.streamlit.io/sarasabino/accident_rate_project/main/Src/Notebooks/05_streamlit_app.py