Report for the “Statistical Learning” exam

Accidents severity prediction

Riccardo Tenuta
Matr. number 26940A

2023/2024

Summary

Abstract

The end goal of this project has been the analysis and prediction of the severity of vehicle accidents. Inside this report will be described different techniques and methods of statistical learning both from supervised and unsupervised methodologies to perform exploratory analysis and prediction over an existing dataset containing information about the accidents dynamics and variables that impacted their severities. At the beginning it is described the dataset, how it is composed and the most important distributions which explain the accidents, subsequently it will be cover the data pre-processing which is fundamental in order to get a final version of the dataset to be able to perform some of the unsupervised and supervised machine learning algorithms. At the end the conclusions will list the final results of the analysis and the accuracy of the models that have been implemented.

1 The dataset

As said before, the dataset contains information describing the accidents and how this impacted the traffic conditions. All the samples relate to a set of US accidents that happened between 2016 and 2023.

The set of data which has been used for this project is composed by these variables:

- **StartTime**: the date and time the accident starts
- **EndTime**: the date and time the accident finished and the traffic is restored
- **StartLat**: latitude of the accident
- **StartLng**: longitude of the accident
- **Distance**: length in mi of road which has been affected
- **Street**
- **City**
- **State**
- **Zipcode**
- **Temperature**
- **Humidity**
- **Pressure**: air pressure
- **Visibility**: shows the visibility in miles
- **WindSpeed**: shows the wind speed, in miles per hour
- **Precipitation**: shows precipitation amount in inches, if there is any
- **WeatherCondition**: shows the weather condition (rain, snow, thunderstorm, fog, etc.)
- **Amenity**: annotation which indicates presence of amenity in a nearby location
- **Bump**: annotation which indicates presence of speed bump or hump in a nearby location
- **Crossing**: annotation which indicates presence of crossing in a nearby location
- **GiveWay**: annotation which indicates presence of give_way in a nearby location
- **Junction**: annotation which indicates presence of junction in a nearby location
- **NoExit**: annotation which indicates presence of no_exit in a nearby location
- **Railway**: annotation which indicates presence of railway in a nearby location
- **Roundabout**: annotation which indicates presence of roundabout in a nearby location
- **Station**: annotation which indicates presence of station in a nearby location
- **Stop**: annotation which indicates presence of stop in a nearby location
- **TrafficCalming**: annotation which indicates presence of traffic_calming in a nearby location

- **TrafficSignal**: annotation which indicates presence of traffic_signal in a nearby location
- **Severity**: a code representing the severity of the accident (from 1 to 4), with 4 as the highest severity

In total the dataset contains 28 variables or columns which describe the details of the accidents. The majority of them are numeric, such as the latitude, longitude, temperature, humidity and pressure. As we'll see later on, also StartTime and EndTime, which are critical information for an accident, will be converted into numeric variables by taking into consideration only the hour and the minutes (both for start and the end) in order to maintain the information of which moment of the day is happened the accident.
From Amenity to TrafficSignal variables instead contain boolean values, TRUE or FALSE which basically represents whether there was or not the specific condition during the accident. One of the advantages of these columns is that they are already codified by only translating the TRUE values to 1's and FALSE values to 0's so that they can be easily used for the algorithms although for this project they have been excluded.

1.1 Data preprocessing

First of all, the first columns that have been manipulated are the *StartTime* and *EndTime* which were used to extract the hours and minutes when the accidents began. Eventually we ended up with four additional columns, *start_hour*, *start_min*, *end_hour* and *end_min*. These variables are crucial to understand the relation and the impact that the moment of the day had to the accident. We'll see that relation with a plot of the correlation matrix later on.
Other modifications have been the conversion of all the "numeric" columns, such as *Temperature* and *Humidity* to a double type to make sure to have the right data type when we'll run the analysis and algorithms. For this project I excluded the column from *Amenity* to *TrafficSignal* in order to avoid increasing the model's complexity.

Another key step before going through the models was to remove all of those rows which contained null values to have a cleanest dataset and avoid any information loss during the processes. After removing those rows, in total we have 100K elements inside our dataset.

1.2 Exploratory Data Analysis

In order to have a better visibility on the variables inside the dataset let's have a look at some charts and distributions. Before that we can see how the accidents are distributed on the US map.

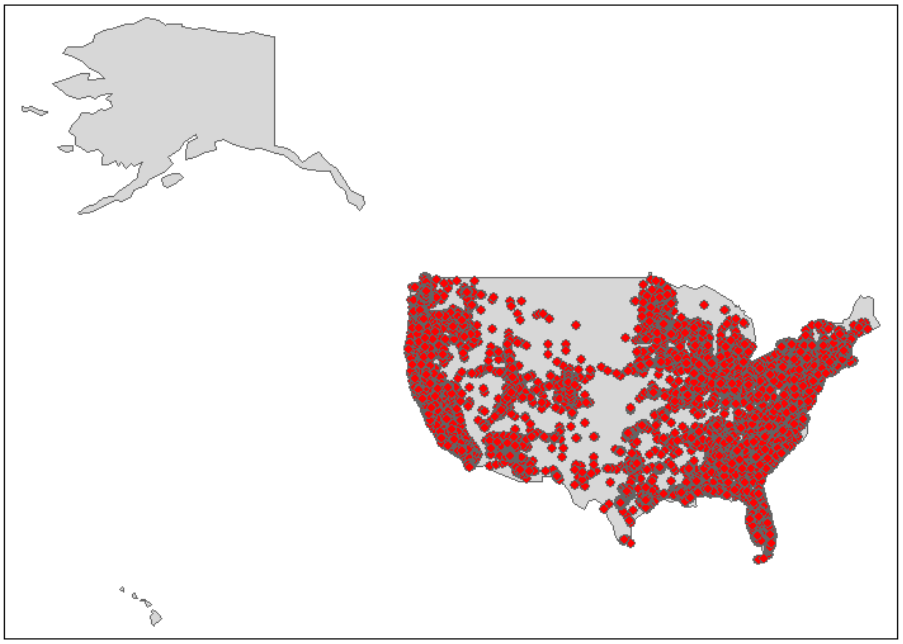

Figure 1.0 - US accidents map

Now let's dive deep into some plots which show the relations between the different variables and how they behave with each other.

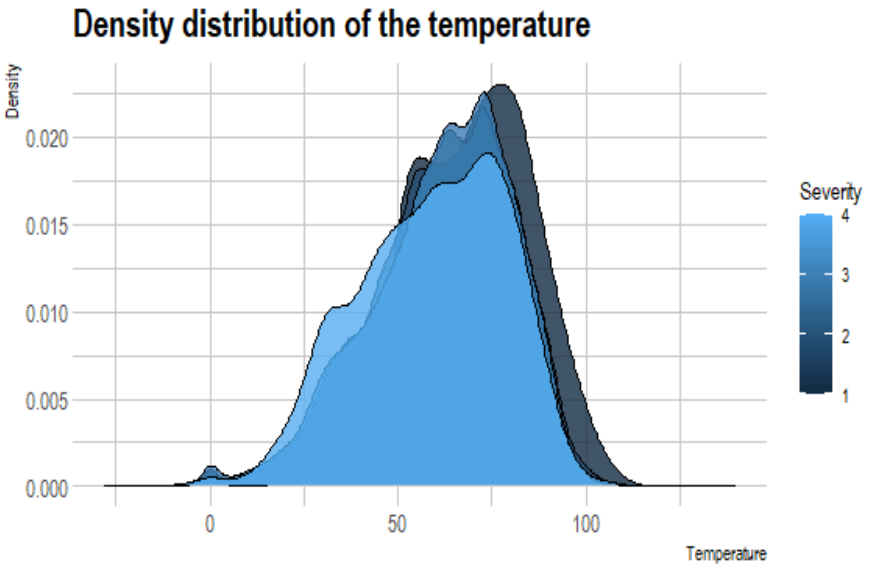

Figure 1.1 - Density distribution of the temperature

From this distribution we can already notice how the density distribution of temperature shows that the higher is the severity and the lower will be the temperature at the moment of accident, probably due to the formation of ice and frost on the road.

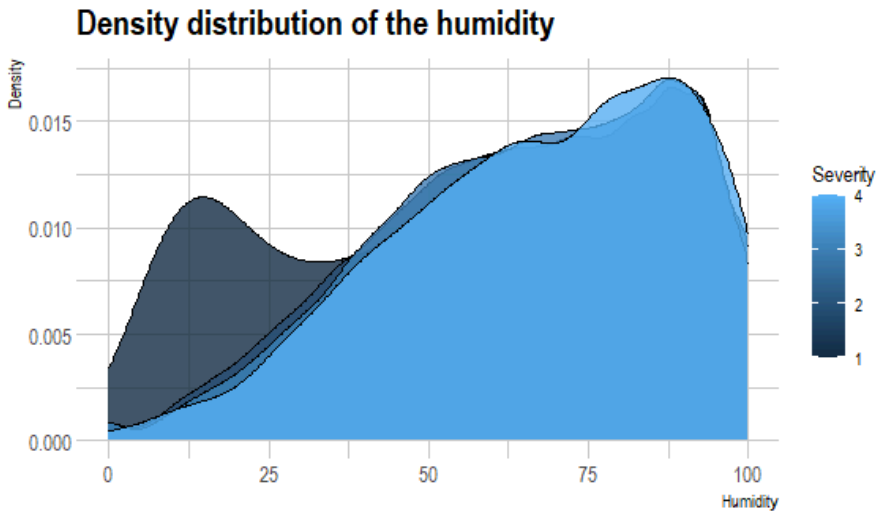

Figure 1.2 - Density distribution of the humidity

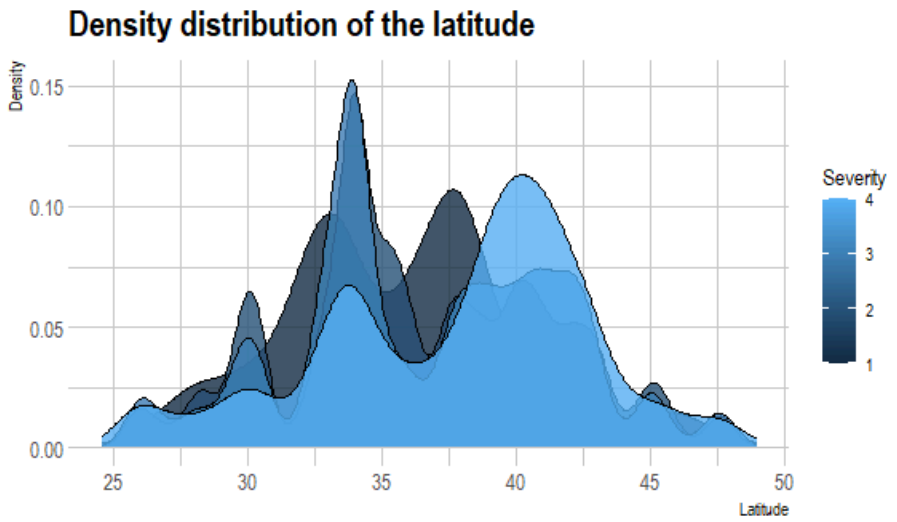

Figure 1.3 - Density distribution of the latitude

As we've seen before, there is probably a kind of correlation between the place (and so also the latitude and the temperature) and the severity of the accident.

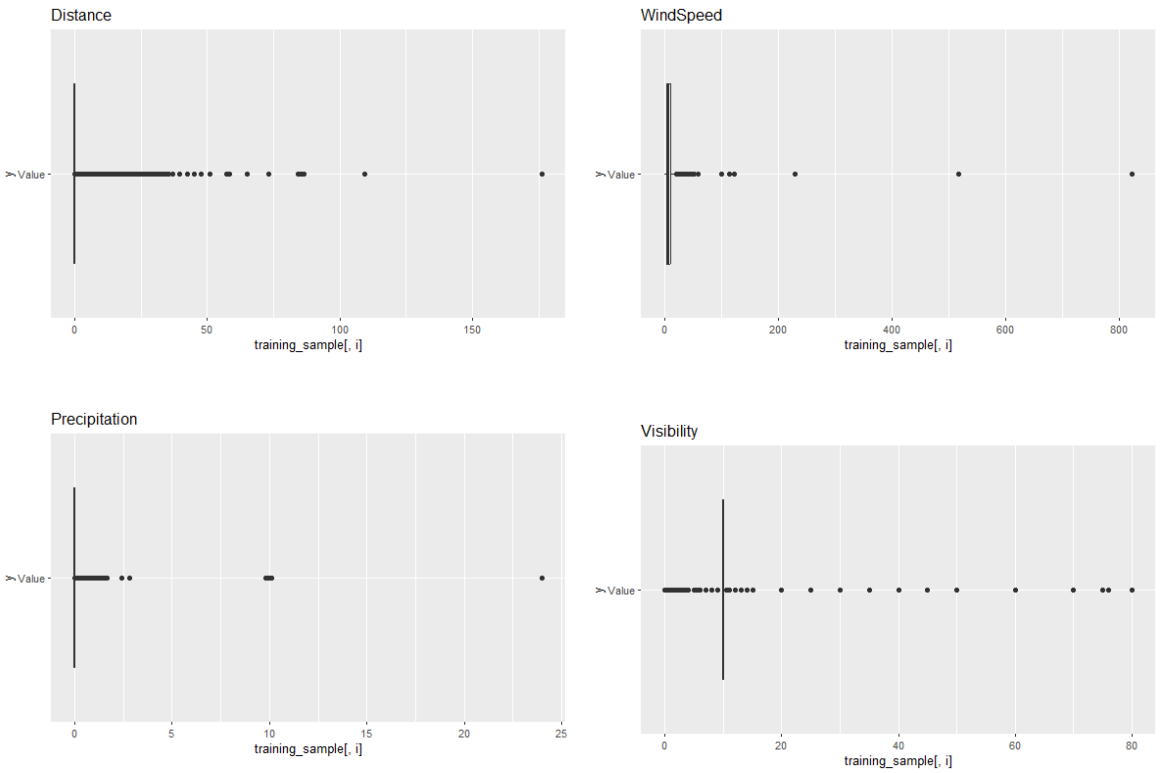

Figure 1.4 - Features boxplots

These boxplots represent the distribution of some of the main features of the dataset we are analyzing. From these boxplots we automatically know that every column has a different scale and range of value, this is obviously something that has been considered during the application of the PCA (Principal Component Analysis) which requires all the features to be on the same scale.

Another important chart in order to visualize the relation between the features is the correlation matrix from which we can understand how much multicollinearity is present in our data.

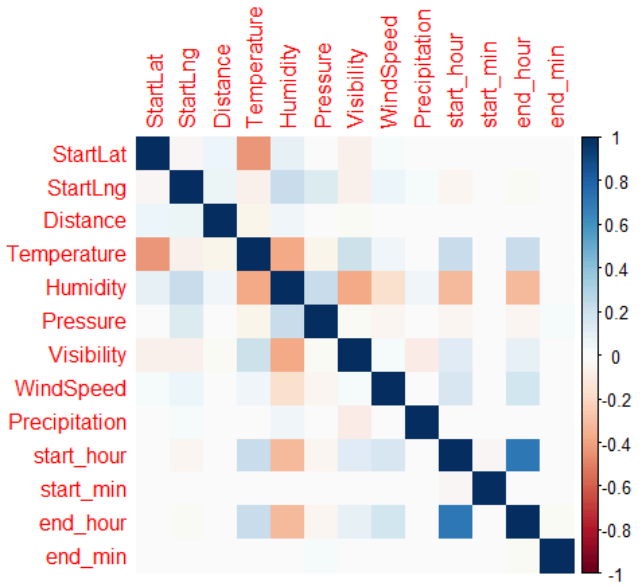

Figure 1.5 - Correlation matrix

Since the severity column represents the target variable in our dataset it's fundamental to analyze also the distribution of all the different classes that are represented. Knowing how many data points we have for all the classes allows us to understand if we may encounter an unbalanced class problem during the classification that we want to perform later for the supervised learning section. Let's plot a barchart counting the data point for each class (severity label).

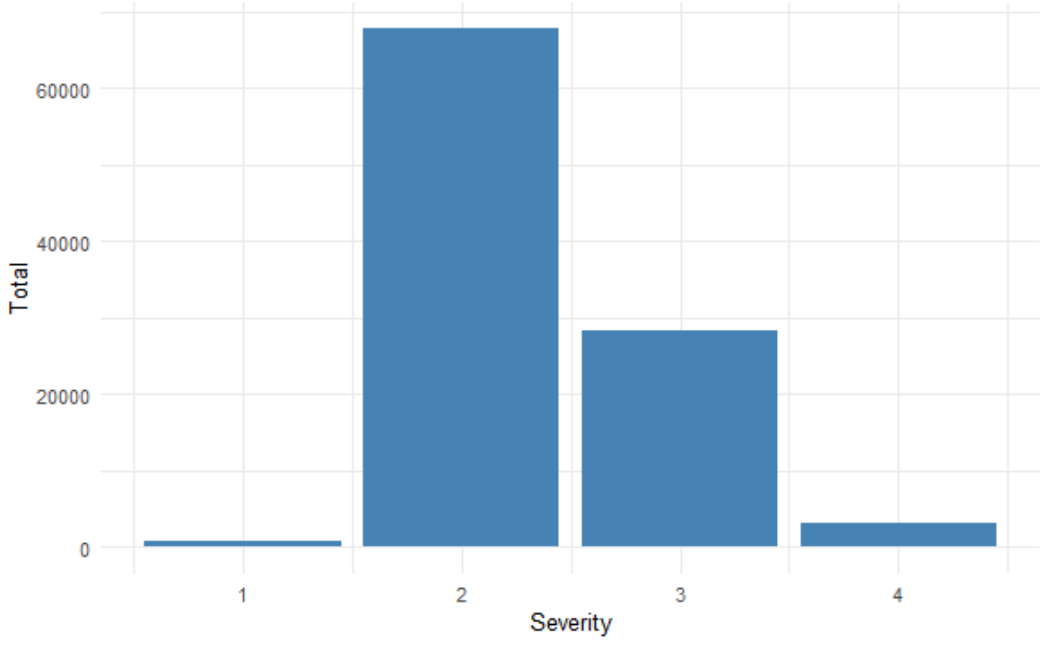

Figure 1.6 - Barchart with the total number of data point for each class

From this last chart it's clear that we have an unbalanced classes problem and before moving to build a classification model it is important to avoid this in order to not produce a biased model which is trained mostly on the same majority class, in this case the severity 2. Therefore the first thing that has been done after the preprocessing is the application of the SMOTE algorithm which is an oversampling technique that generates synthetic data for the minority class.

2 SMOTE (Synthetic Minority Over-sampling Technique)

This method has been applied two times, firstly for the class 1 (the one with the lower severity) and then for the class 4 (the one with the higher severity) in order to have a balanced distribution of sample within the dataset. Specifically the creation of new data points for the minority class is done following these four steps.

1) It is selected from the dataset the minority class (1 or 4).
2) It is then computed the k-nearest neighbors algorithm to select the nearests data points to the ones from the selected class. The distance between the points can be the euclidean distance, manhattan distance or any other distance measure.
3) For each instance it's selected a random neighbor and it will be created a new synthetic instance using the interpolation between the neighbor and the original data point from the minority class. The interpolation creates the new instance doing a linear combination of the original value and its selected neighbor multiplying its value by a random number between 0 and 1.
4) The original minority instances and the generated synthetic instances are combined to form the oversampled minority class which is then combined into the original dataset.

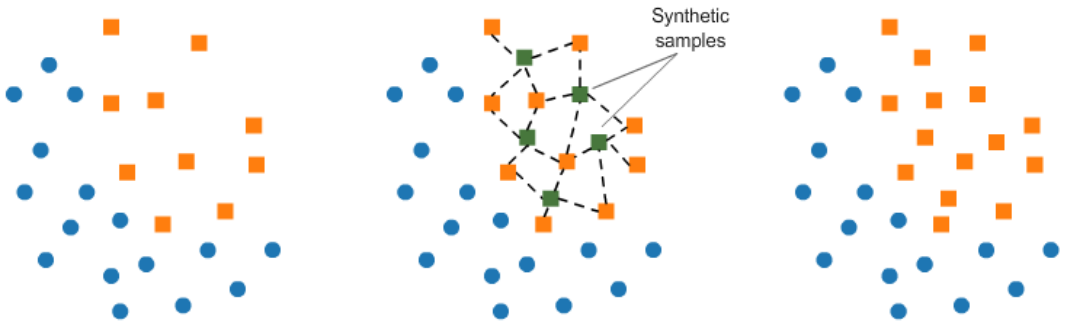

Figure 1.7 - Visualization of how SMOTE technique works

This technique has been used through the *smotefamily*, the main function needed two parameters: *k* and *dup_size*. Specially the duplication size is important to set how many synthetic data will be injected into the original dataset, the right trade-off between balancing the classes distribution and not insert too many artificial data points in the sample cause this might infect the way the model predicts the real test data.

We applied this algorithm to avoid a unbalanced-class problem we encountered, as we'll see later we run a supervised learning algorithm to classify our data points and predict the severity and it was noticed that with the original data and so the unbalanced classes the accuracy level reached was lower in respect with the accuracy obtained with also the synthetic data. This means that the model was biased towards a specific class, in this case the class 2. Right now it's not been covered yet the supervised learning method used but it is explained in the relative section. Let's see the new classes distribution with also the new synthetic instances generated by the SMOTE technique.

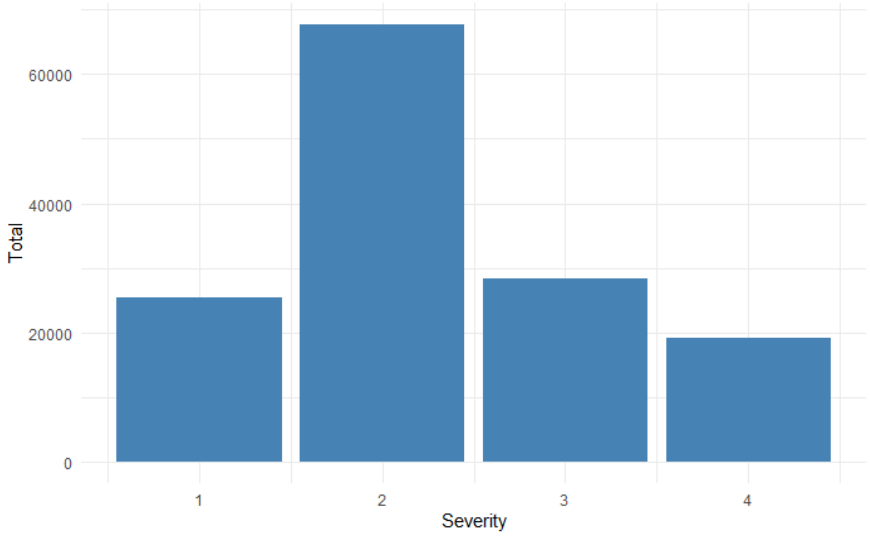

Figure 1.8 - Severity class distribution after SMOTE

3 Unsupervised learning

For the unsupervised learning part, it has been decided to apply and use two algorithms, the PCA (Principal Component Analysis) and the Hierarchical Clustering. The main objective in using the PCA has been to reduce the dimensionality of our dataset in order to also have the possibility to plot in a bi-dimensional space the data points. Hierarchical clustering instead has been applied to the dataset as a way to find patterns and group similar accidents which have similar characteristics. Let's dive deep into those methodologies and how they have been exploited in this use-case.

3.1 Principal Component Analysis

Principal Component Analysis is a dimensionality reduction technique widely used for visualizing and simplifying high-dimensional datasets. This technique will create new features, called Principal Components, which summarize the variance between the real features and are linearly independent with each other. In brief, PCA finds the principal components finding the directions along which the data varies the most. Every principal component is orthogonal to each other and each of them capture the most variance, for example the first-PC captures the most variance direction, the second-PC captures the second most variance direction and so on. Each of these PCs is influenced and composed, with different weights by the original features present in the dataset.

Starting from Figure 1.5, we can tell that some of the columns are correlated, both positively or negatively, this means there is the risk of introducing some problems when it comes to classifying the data points. For example in our dataset we have some sort of correlation between the features *Humidity* and *Temperature*, *Temperature* and *StartLat*, *Humidity* and *Visibility* and *Humidity* and *start_hour*. The final goal of the PCA will be to try to remove this relation of dependence between the features in a lower-dimensional space.

This is a list of problems which can be raised by having features that are dependent with each other:

- **Multicollinearity**: features contain redundant information and this can lead to an unreliable model.
- **Overfitting**: correlated features can cause the model to learn from some specific patterns and perform poorly with new data.
- **Difficulty in interpretation**: it will be difficult for the model to interpret the individual contribution of each feature to the model's prediction, specially during classification tasks as we'll see later.

Before applying the PCA to the dataset's features need to be scaled to the same scale.

To do that we simply set the *scale* parameter to *TRUE*. After applying the PCA technique these are the first five principal components:

	PC1	PC2	PC3	PC4	PC5
StartLat	0.1726836724	-0.52057293	-0.46048305	-0.14049700	0.123608318
StartLng	0.1401237818	-0.17977825	0.56768926	-0.23616122	-0.195073321
Distance	0.0587603461	-0.19038501	0.02821191	-0.27484700	-0.547701081
Temperature	-0.3977770935	0.39374578	0.28514057	0.07070843	-0.061195191
Humidity	0.4769378419	-0.13684325	0.25387518	0.09141125	0.062642205
Pressure	0.1461875149	-0.07943004	0.48446042	-0.28895280	0.349060210
Visibility	-0.2794342705	0.23911024	-0.16256981	-0.46156919	0.005626127
WindSpeed	-0.1796831752	-0.28058973	0.04403613	-0.13160943	-0.199671464
Precipitation	0.0375158009	-0.08806552	0.12968114	0.68892753	-0.035492218
start_hour	-0.4632429755	-0.40572058	0.12657829	0.08058367	0.114035366
start_min	0.0031729506	0.01705100	-0.04761436	-0.01463141	0.445068761
end_hour	-0.4622162750	-0.41422329	0.13803112	0.08797114	0.106187864
end_min	0.0008882526	0.01501660	0.04360744	-0.18072949	0.503639561

Figure 1.9 - First five principal components

From the result of the first five PCs we can already notice which features impact more every principal component, for example we can conclude that *Humidity* is the column that mostly influenced the variance between our data points and so impacted more on the first component.

Let's plot the scree plot which shows how much variance is captured by every principal component.

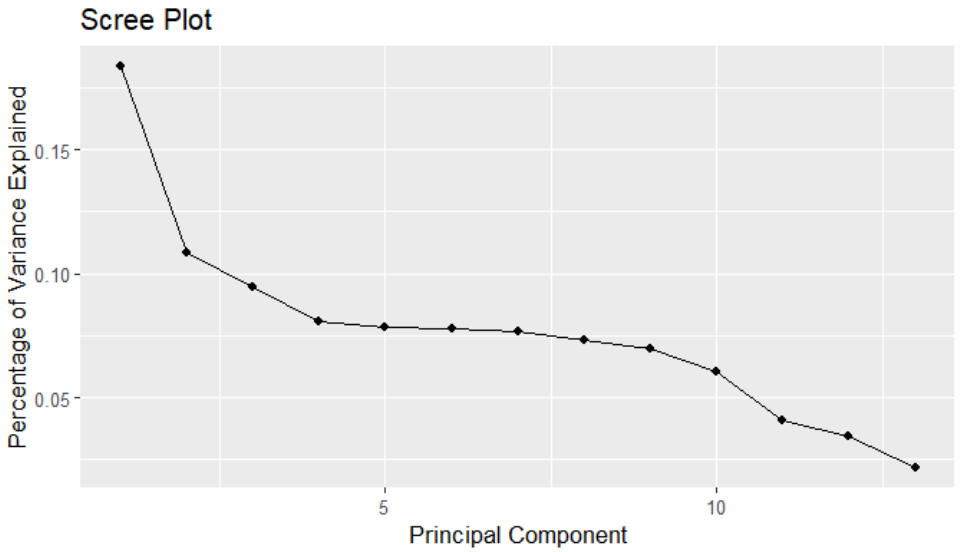

Figure 2.0 - Scree plot of all the PCs extracted

In order to have a clearer view of the datasets and of the PCA result we plotted the first two principal components generating a bi-dimensional visualization of the directions that maximize the most the variance.

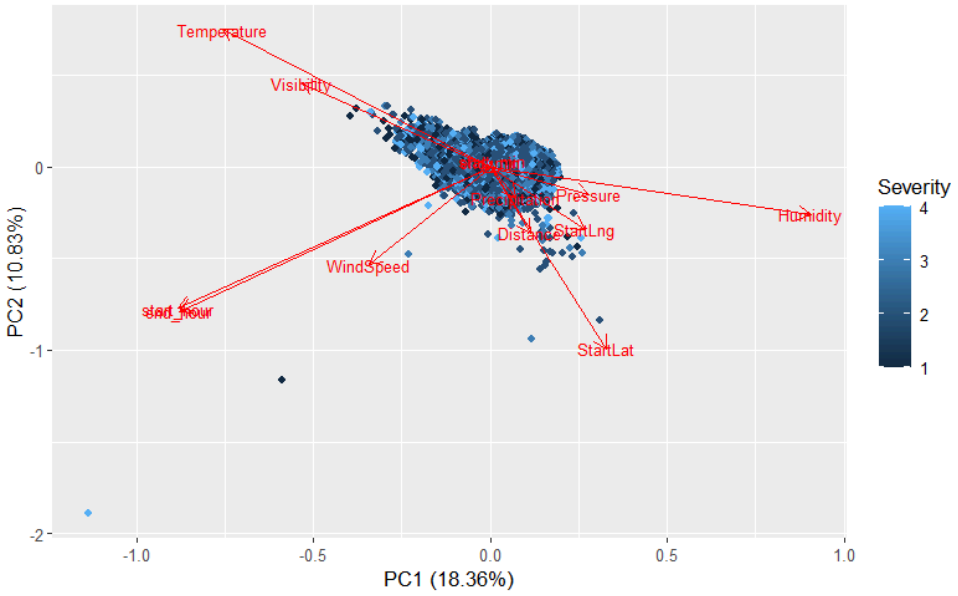

Figure 2.1 - PC1 and PC2

With this latest plot we could visualize our data points but mostly we were able to see what are the directions that maximize the variance and so it's pretty clear that *Humidity*, *Temperature* and *start_hour* were the most impacting features in terms of how much the data vary within the sample. Knowing that, now we have a new dataset represented by all these principal components which summarize all the information avoiding the initial correlations between the original columns. These principal components can now be used to build a classification model which optimizes its accuracy and minimizes the risk of overfitting.

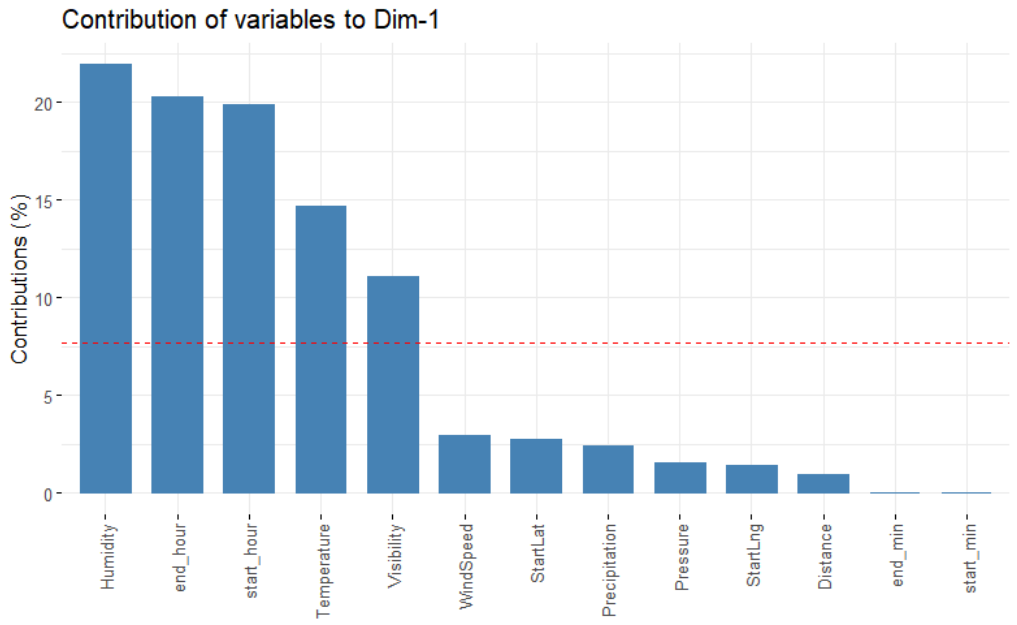

Figure 2.2 - Features impacting first principal component

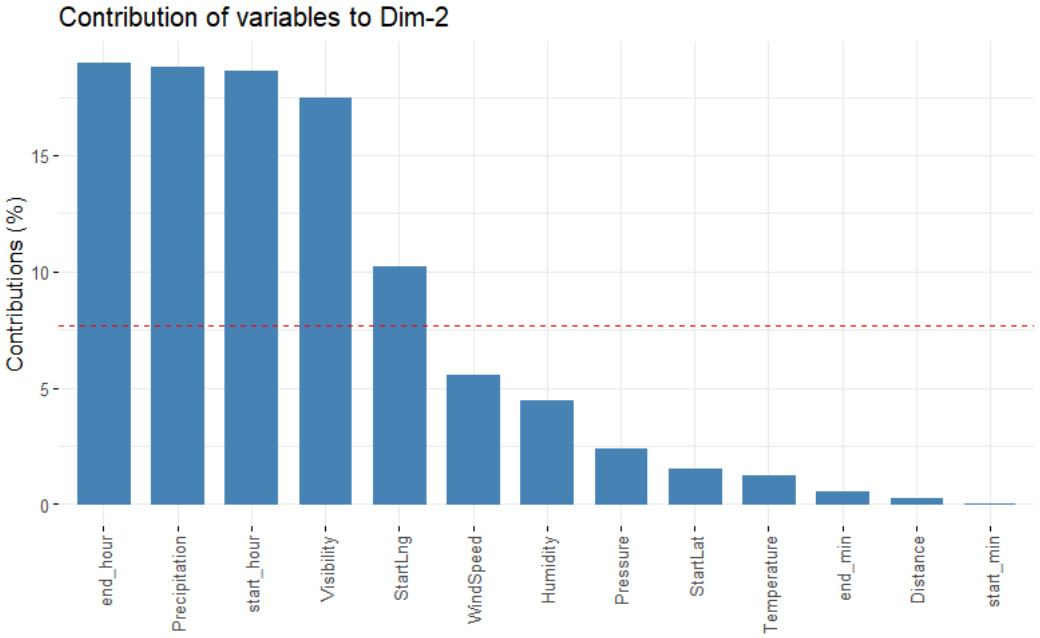

Figure 2.3 - Features impacting second principal component

3.2 Hierarchical clustering

Hierarchical clustering is a machine learning algorithm which is used for clustering and creating groups of similar data points together. With the term “Hierarchical” we mean that the algorithm generates a hierarchy of clusters where each cluster is grouped in another bigger cluster as well. The process of creation of the clusters continues until all the data points are merged in only one big group.

There are two approach for this algorithm:

- **Agglomerative**: This approach is the most common and used. It all starts with every data point in a cluster and then the process aggregates every data point with each other until there is only one single big cluster.
- **Divisive**: The algorithm starts with one single cluster and then recursively divides it until every data point is a cluster.

This machine learning algorithm is based on the concept of the dendrogram. The dendrogram is a tree-like structure which captures the relationship between all the data points. Going up on the structure means to increase the size of each cluster and the opposite means to decrease the cluster’s size but at the same time to have a higher granularity. Thanks to this structure we can evaluate what is the best level to cut the tree and we can generate multiple splits which can lead us to different clustering solutions. Indeed, one of its advantage it’s that with this algorithm we do not have to know or define preventively the number of clusters that we want but the dendrogram tree can be sliced at different levels producing a variety of possible results even though to do this it needs manual intervention. To build up the entire tree structure the algorithm calculates every step the distance, using one of the distance measures, between the already grouped data points.

Let’s dive deep into the practical application. We used ad input for the hierarchical clustering of the principal components that we obtained at the step before in order to simplify and streamline the clustering using data that fully or in part captures the variance in the data. To execute the clustering in a reasonable time without ending up with the full memory it has been taken a sample from the dataset of 1000 data points and then applied the clustering only of this subset.

Here is the final dendrogram:

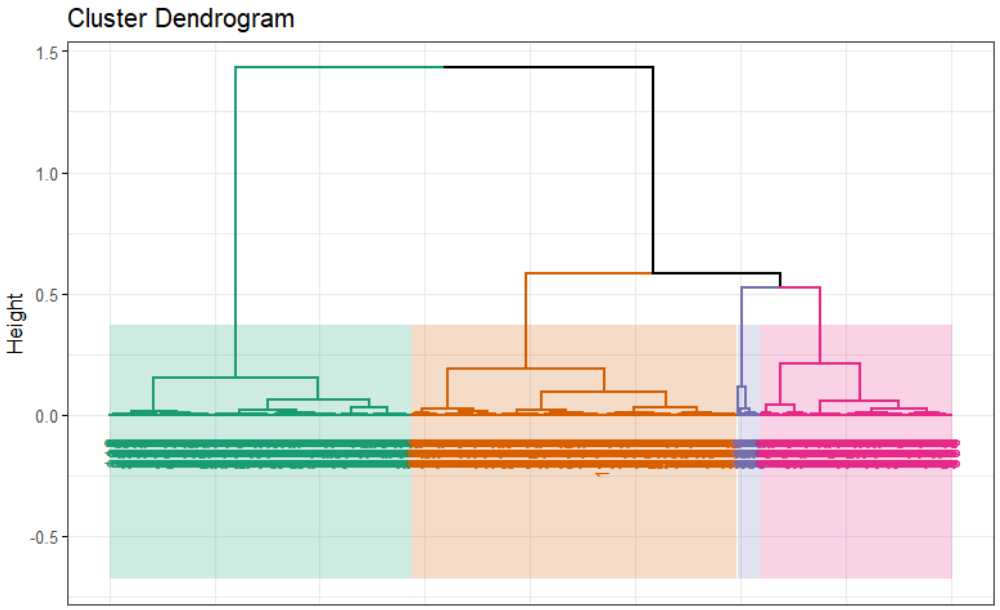

Figure 2.2 - Dendrogram

From the dendrogram we understood that from this sample there are four main clusters which optimize the infra-cluster distance. These four groups may be related to the severity (1 to 4) classes with which the data have been classified and can show interesting patterns and relations between the data points. Here we see the number of data points associated with each cluster.

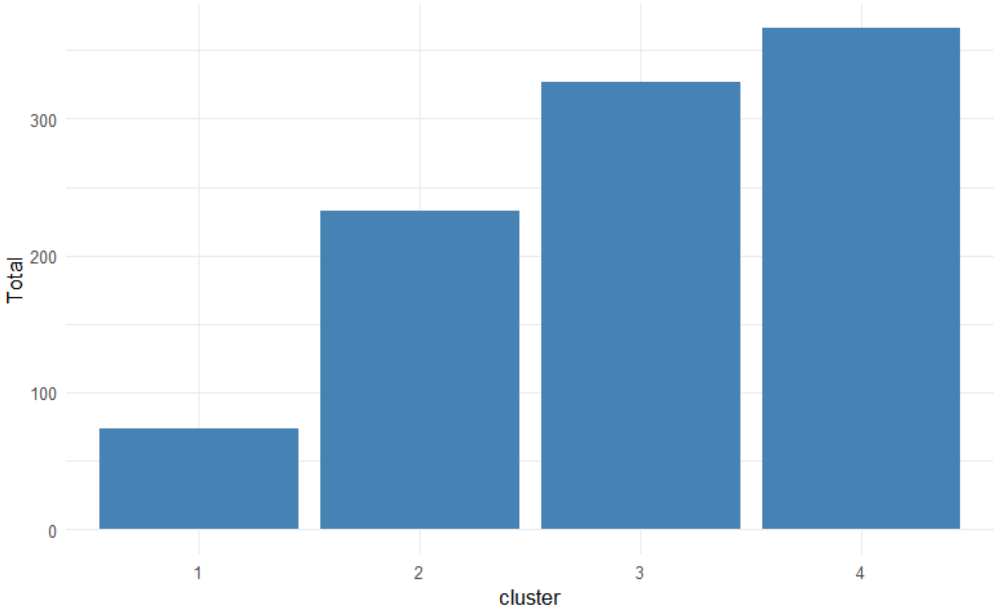

Figure 2.3 - Clusters distribution

Analyzing the characteristics of each group starting from the features that compose the 2 PCs we notice that they differ in *Temperature* and *Humidity*:

	cluster	Average
1	4	74.17921
2	3	63.19881
3	2	51.85291
4	1	46.47511

	cluster	Average
1	1	92.14633
2	2	78.45271
3	3	62.10025
4	4	45.18097

in *Start Hour* and *Visibility*:

	cluster	Average
1	4	16.703524
2	3	12.348644
3	1	12.331855
4	2	6.125525

	cluster	Average
1	4	9.995607
2	3	9.720162
3	2	9.072746
4	1	3.068893

4 Supervised learning

In respect of unsupervised learning algorithms with supervised learning the model learns to classify or to make predictions from an already labeled dataset. The model is trained on a dataset which contains for each data point the right answer that might be a number to predict or a class if it's a classification problem. A supervised learning algorithm can solve two type of problems:

- **Regression**: the target column is a numeric value and the model trains to learn to predict the right value for each data point such as a problem of price prediction.
- **Classification**: the classification problem instead means that the target column in the dataset represents a class, so a sort of group label which identifies and splits our data points in different categories or classes. If the class value has only two values such as TRUE or FALSE we have a binary classification problem while for multiple categories, such as type of objects or type of diseases, we have a multi classification problem.

In our use-case the target column that we wanted to predict is the severity of the accidents. The severity column in the dataset classifies what is the severity of each accident and so we went through a multi-classification problem.

4.1 Gradient Boosting

The technique that has been used to build a statistical model which predicts the right severity starting from all the 13 features that are within the dataset is the gradient boosting. Gradient boosting falls under the umbrella of the ensemble methods which represents all of those machine learning algorithms that build a strong model by collecting multiple of the so called "weaker models", collecting multiple weak models in order to build a stronger model allows the algorithm to increase its accuracy and avoid the overfitting. Gradient boosting is a boosting method which means that iteratively the model improves the performance by train new weak learners on the residual errors of the previous models.

The key idea behind gradient boosting is to construct an ensemble of weak learners (decision trees) where each new tree is trained to minimize the errors made by the previous trees. This process continues until a desired level of performance is achieved or a predetermined number of trees is reached. Let's dive deep into the model that has been built.

First of all, we needed to split the data set in the training set and test set with a percentage of first 80% for the training set and the remaining 20% for the test set. The data we've been using were already shuffled and as we saw before, we used a dataset that also included new synthetic generated data points in order to avoid or decrease the unbalanced class problem. To compute this algorithm it's been used the library XGBoost.

4.2 XGBoost

Some of the key parameters of this algorithm are:

- **the objective function**: we used the the softmax function which produce a vector of probabilities for each of the available class
- **max_depth**: the maximum depth of each decision tree used is 3, increasing this value could lead to much more complex trees at each iteration
- **learning rate**: this is the most important parameter which defines the level of contributions of each weak learner and this will impact how many trees and iteration the model needs to converge. In this case we decided to set it at 0.2.

We also specified the number of rounds or iteration that the model should iterate to 200 with an early stopping value at 3, so possibly after 3 iterations where the loss function value has not decreased the computation is stopped.

After training the model we obtain the two loss function during the iterations:

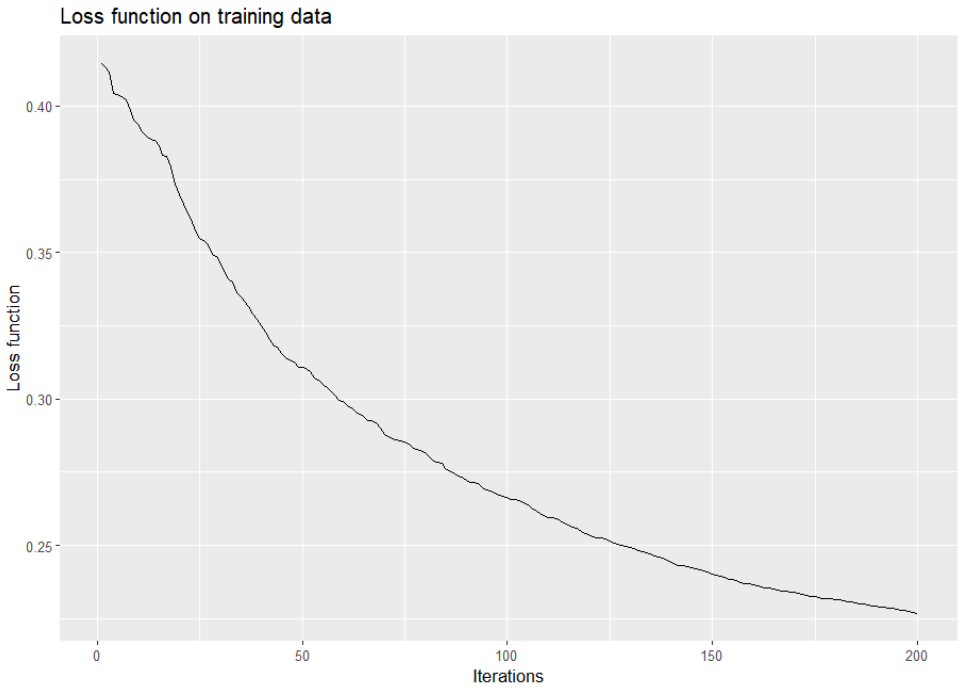

Figure 2.4 - Loss function on the training data

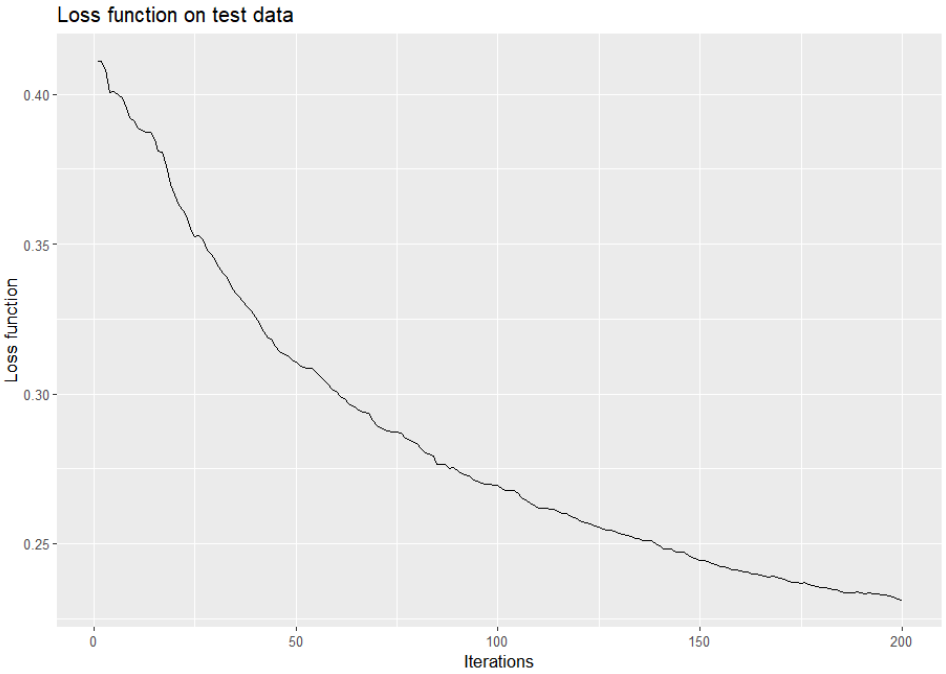

Figure 2.5 - Loss function on the test data

Seeing the loss function that we got after training the model we know that the model is not overfitting. The final error (multiclass classification error), which is the evaluation metric the loss function keeps track of, is 0.22 on the training set and 0.23 on the test set meaning that on unseen data points the model has almost the same probability of misclassifying the severity's accident.
To compute a final evaluation metric for defining the accuracy of the model we built the confusion matrix, and then calculated the accuracy as the (TP + TN) / (TP + FP + FN + TN). The accuracy reached was 77%.

One particular aspect that has been taken care was the initial overfitting a biased that was making performing poorly the model, in fact since the original dataset contained unbalanced class at the beginning the classification was not predicting properly the data of each class because of some bias and the accuracy was a little bit lower (around 70%). Using the SMOTE technique and also shuffling the dataset allowed us to improve the model's performance even though to reach a higher accuracy we may continue to test and tune the model with different hyper-parameters.

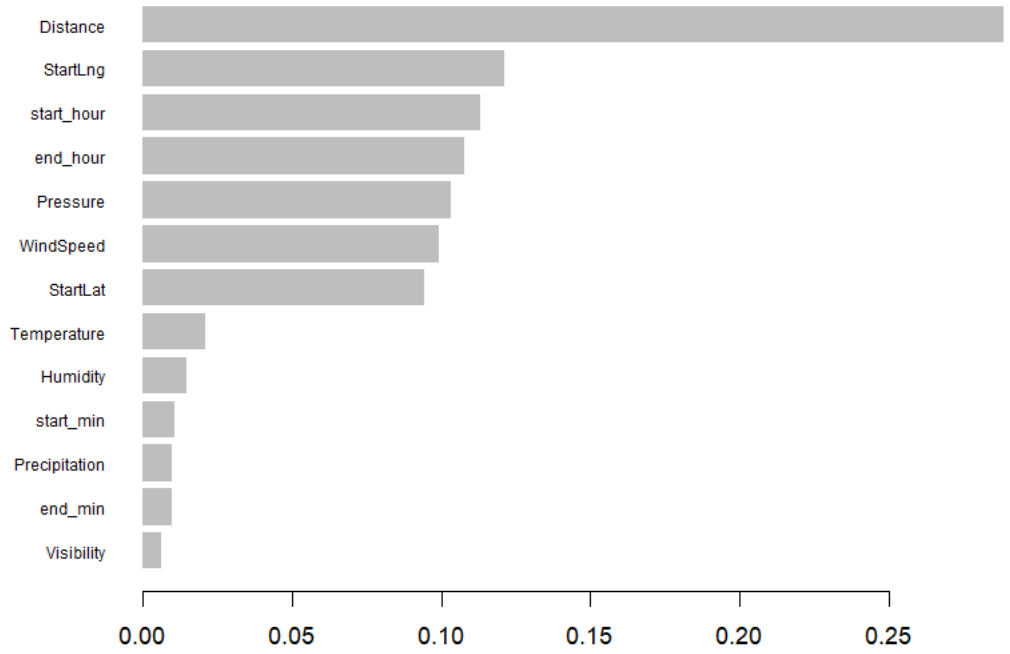

Figure 2.6 - Features importance

From the same library we extracted the features' importance and from this last chart we can understand what are the main features which impact the split in the final decision tree. In this case the *Distance*, the *StartLng* and the *start_hour* are the columns that have the highest information gain and maximize this value to produce the optimal split to predict the correct class.

5 Conclusions

Different statistical learning models have been used for this analysis and each of them has been useful to explore the data points, extract some patterns and relation between the data and predict the severity of new accidents.

The Principal Component Analysis allowed us to summarize the data and define new features which are uncorrelated with each other, this is useful when reducing the complexity of a model or to avoid overfitting.

The hierarchical clustering has been fundamental to extract and group, not taking into consideration the actual classification, the main characteristic of the data points based on the first two principal components and how the most impacting columns such as *Temperature* and *Humidity* are those who differentiate the most the accidents.

The last model, the Gradient boosting from the supervised learning section, has been the most important to predict the severity's accidents. With the XGBoost library we built a model with the softmax objective function to optimize for and with the goal to reduce its evaluation metric, the multiclass classification error rate. Using the boosting technique combined with this ensemble method it has been reached an accuracy level of 77%.

Further tuning of the hyperparameters or a higher class balance in the original dataset would have allowed to improve the total accuracy of the model.