UNIVERSITY CARLOS III OF MADRID

Master in Actuarial Science and Quantitative Finance

Comparative performance analysis between Gradient Boosting models and GLMs for non-life pricing

by

Viktor Martínez de Lizarduy Kostornichenko

with student ID
100417251

and with the supervision of
José Miguel Rodríguez-Pardo del Castillo

and
Jesús Simón del Potro

June 2021

Esta tesis es propiedad del autor. No está permitida la reproducción total o parcial de este documento sin mencionar su fuente. El contenido de este documento es de exclusiva responsabilidad del autor, quien declara que no se ha incurrido en plagio y que la totalidad de referencias a otros autores han sido expresadas en el texto.

En caso de obtener una calificación igual o superior a 9.0 (Sobresaliente), autorizo la publicación de este trabajo en el centro de Documentación de la Fundación Mapfre.

✓ Sí, autorizo a su publicación.
No, desestimo su publicación.

Abstract

Modelling the behavior of risks is one of the most fundamental pillars in the insurance business throughout all its branches. Actuarial practitioners have always been interested in finding the best statistical tools to capture the nature of the risks they undertake from their clients, and in the last decades these techniques have thrived through the implementation and expansion of Machine Learning, both to process and handle large amounts of data, as well as to carry out advanced computations.

Specifically, and as the purpose of this document, we will be focusing on the Gradient Boosting algorithms from the sub-family of ensemble methods used for regression to predict and model basic pricing variables such as frequency and claim severities, and compare their predictive and pricing capabilities with classical Generalized Linear Models.

In our study case of a French insurance motor portfolio, we found that Gradient Boosting models have a stronger predictive performance and a higher pricing ability to adjust the premiums to both high risk and low risk profiles. And finally, we conclude that these models can be used to support and improve GLMs and their pricing results as Machine Learning continues to settle in the actuarial modeling paradigm.

Key words: *Data Science, Machine Learning, Insurance, Pricing, Modeling, Performance*

Resumen

La modelización de riesgos y su comportamiento es una de las ramas fundamentales del negocio asegurador a lo largo de todas sus ramas. Los actuarios profesionales siempre han buscado las mejores herramientas estadísticas para capturar la naturaleza de los riesgos suscritos a sus clientes, y en las últimas décadas éstas han prosperado gracias a la implementación y expansión de las técnicas de Machine Learning, tanto para procesar y gestionar grandes cantidades de datos como para desarrollar tareas de computación avanzada.

En concreto, y como propósito de este documento, nos centraremos en los algoritmos de Gradient Boosting de la subfamilia de métodos de ensamble, utilizándolos para la regresión y predicción de variables de tarificación básicas como frecuencias y severidades, para comparar sus capacidades predictivas y de tarificación frente a los GLM tradicionales.

En nuestro estudio sobre una cartera francesa de pólizas de riesgo frente a terceros, encontramos que los modelos de Gradient Boosting tienen una capacidad predictiva superior y una mayor habilidad para ajustar las primas a perfiles de alto y bajo riesgo. Finalmente concluimos que estos modelos pueden ser usados para apoyar y mejorar los GLMs y sus resultados de tarificación mientras el Machine Learning continúa su asentamiento en el paradigma de modelización actuarial.

To the family and friends who fueled me with passion, our struggles and dreams have made us who we are.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

The insurance industry has grown constantly through the past century, specializing in as many lines of business as it is possible to imagine. Its aim is to manage the risks that their clients transfer to them by hands of the formal contract bonding both parties, all of it in exchange of a pre-established premium. These risks have developed into many branches in very different fields and environments, such as marine and air transport, legal defense, business interruption compensation, natural disasters, workforce insure and many other spheres like the well-known life insurance, health, and pension schemes.

It is thus self-evident that the type and nature of the risks that the insurance company will have to manage are very particular in each situation, and will most likely be tied to macro-economic, environmental, legal and political factors driving the behavior, the potential impact and even the interactions between variables. All these properties make the insured risk highly dynamic and complex in their effects and in the dependencies amongst them.

It is for this reason that the insurance industry has been trying, almost since its inception, to correctly capture and model the behavior of the risks they undertake, in order to know the extent of the losses they will have to assume to compensate their clients in accordance with the terms of the policy. And this is the goal that gives birth to the pricing process in the insurance business.

According to Parodi, P. (2015) the pricing process in any general insurance company begins with familiarizing with the risk and its nature, taking into account the information about exposure, cover and past claims. This initial yet fundamental stage is labeled by the author as the *“Risk costing subprocess”*.

It is in the context of this essential sub-process that the newest and most sophisticated modern statistical learning techniques come into play. It might already be well known for the reader that in the last decades, these methods have thrived in an unprecedented level by the hand of the Machine Learning tools.

Machine Learning (ML) has extended its implementations in fields such as genetics, medicine, commerce, astronomy and many other fields, and its eventual incursion into insurance and actuarial science was almost inevitable. Particularly, and as the focus of this work, we will be presenting and testing some of the most state-of-the art applications of the sub-family of Ensemble Techniques of ML algorithms in the risk costing stage of pricing.

The main benefits of Ensemble Techniques come from their capability to capture complex dependency structures and interactions among predictor variables (capturing the full nature of the risk) as well as non-linear relationships, all of it without renouncing to the interpretability of results.

1.2. Goals of the study

The purpose of this work is to dive into some of the most notorious and influential Machine Learning methods based on Ensembles of Trees implemented in regression for the purpose of modelling and predicting loss costs in a general insurance environment. Particularly, we will be implementing Gradient Boosting (GB) algorithms like Stochastic Gradient Boosting Regression through the open-source software implementation of LightGBM.

The main advantages of these methods will be assessed in a portfolio of French Motor Third Party-Liability (TPL) policies and compared with a standard GLM model in order to illustrate their performance in terms of predictive strength through the Generalization Error measure and through pure-premium comparison for a Pricing Machine based on both the GLM and the GBM approaches. The case study and analysis will be performed in a python-based programming environment using the most up-to-date libraries for ML modelling and testing.

The interpretability of the GB methods will be shown through Partial Dependency Plots (PDPs) in order to illustrate the non-linearities that are captured by the model as well as the potential interactions across features. All of this from a data-driven perspective, based on the Ensembles of Trees approach.

Our final goal is to prove the suitability of GB algorithms for building comprehensive data-driven loss costing and pricing structures that are suited for the big-data environment of present-day insurance companies.

1.3. Brief description

The document's second chapter will first introduce the literary review surrounding the standard GML models for pricing and loss costing, giving a brief outline of their fundamental properties and their fitting process, as well as some of their most important advantages and disadvantages. Next, we will revise the basics of decision trees and their algorithmic nature in most ML models alongside some of their applications in insurance risk modelling. The following sections will deepen into the Ensemble Techniques revolving around Decision Trees and how these can significantly enhance their predictive power, presenting along the way the GB algorithms of our interest.

The third chapter will present the case study we will use as a basis for our models training and comparison. We will describe and explore the main properties of our input dataset, which belongs to a French Motor Insurance company with a portfolio of TPL policies.

The fourth chapter will focus on the models fitting methodologies, such as the feature engineering for both GLM and GB models as well as the necessary hyperparameter tunning of the latter.

The fifth chapter will show the comparative results and generalization errors between GLMs and GBM models, along with other comparative analysis for pure premium prediction and the PDPs for our GB models, interpreting the most relevant insights about the features' dependency structures.

Finally, the sixth chapter will summarize the conclusions about the comparative analysis and the main strengths as a formal loss costing/pricing model.

1.4. Main findings and conclusions

In the frequencies model, our fine-tuned GBM model clearly outperforms the frequency GLM with statistical significance on all parameters, according to the measure of GE though Poisson Deviance. On the severities side, less features were used as predictors, with only 5 features on the GLM and 4 on the GBM, yet when comparing the predictions from both models, in term of Gamma GE the GBM model performed better as well.

After computing the pure premium in both proposed pricing models, we saw that the Loss Ratio on the testing portfolio was 97.04% for the GLM pricing machine and 97.14% for the GBM based scheme. Both models yielded a similar average pure premium although the latter has a higher dispersion in its premium distribution.

Our QQplot analysis found that in fact the GBM pricing scheme charges higher premiums to the riskiest profiles compared to the GLMs, while also tending to charge a lower premium to profiles it deems as less risky.

Also, through Partial Dependency Plots (PDPs) we see that our GBM models of choice successfully capture non-linear and non-monotonic dependency structures. And in the severity PDPs we also see that the levels present in them for continuous features can be easily separated into intervals, in order to bin their values into a categorical feature that could be used to reinforce the training of GLMs.

The main conclusion is that although as of this day GBM models might not be able to fully substitute GLMs in the tariff environment common to most general insurance companies, they can still provide valuable analytical support to both back test and reinforce GLM training and results, while also enriching the insight obtained for pricing analytics.

2. Literature Review

The next review will explore the theoretical and mathematical basis of the models that will be applied in our study from a statistical learning perspective. The first stop will depart from the fundamentals about GLM models and their importance in general insurance pricing since their first implementation in the 1980s.

2.1. GLMs and Traditional Loss Cost modelling

In the preface of Hastie, T. *et al.*, (2009) we have that nowadays "*vast amounts of data are being generated in many fields, and the statistician's job is to (...) extract important patterns and trends, and understand what 'the data says'*". This is the basic concept of statistical learning, and GLMs in insurance are just another family of methods designed to extract information about the undertaken risks and applying the gained knowledge onto several business stages like:

- Demand: modelling new business and renewals through elasticity curves and cancellation probabilities.
- Reserving: determining IBNR and IBNER through individual policy information as well as modeling times until settlement.
- Loss Costing: establish the 'fair' quotation to each policy based on their underlying degree of risk and their general behavior under different scenarios.

Although our work will focus on the latter, there are even more diverse insurance applications in different lines of business, many of them still in development up to this date.

2.1.1. The context for GLM applications

The insurance industry and particularly the General Insurance business was severely constricted before the 1990s across most of developed countries due to strong regulations regarding pricing and tariff policies. Companies could not reflect their risk costing analytic models into policy prices, fully capturing the risk profile of each policyholder, and such scenario limited the spread and development incentives for new statistical tools and models.

The first applications of GLMs in General Insurance Pricing appeared in the 1980s and it was not until the second half of the next decade, when the regulations around pricing and tariff policies were gradually lifted, that GLM could spread throughout the industry and become the pricing insignia of non-life insurance companies across many markets.

The key to their success in the industry was that they enabled to generalize the basic linear regression process to multiple distributions commonly used in actuarial analytics such as the Poisson, the Binomial, the Gamma or the Inverse Gaussian distributions and, to modify the basic additive linear relation (1) between response variable and predictors to set other more convenient relationships (2) like the multiplicative structure that is so commonly used in pricing/tariff models.

$$\boldsymbol{Y} = \boldsymbol{X_1} + \boldsymbol{X_2} + (\dots) + \boldsymbol{X_n} \qquad (1)$$

$$g(\boldsymbol{Y}) = \boldsymbol{X_1} + \boldsymbol{X_2} + (\dots) + \boldsymbol{X_n} \quad (2)$$

Where $\boldsymbol{Y}$ is the response variable and $\boldsymbol{X_i}$ $(i = 1, 2, \dots, n)$ are the predictors. The function $g(\,.\,)$ is known as the link between response and predictors in the GLM context. Although the statistical theory is very general and can be used in multiple fashions (using many different link functions and response distributions), we find that in the actuarial practice the responses are a very particular set of distributions known as the EDM (Exponential Dispersion Models) as quoted from Ohlsson, E. & Johansson, B., (2015). In addition, the most commonly used link is the log-link, which enables the multiplicative structure:

$$\ln \boldsymbol{Y} = \boldsymbol{X_1} + \boldsymbol{X_2} + (\dots) + \boldsymbol{X_n};$$

$$Y = e^{\boldsymbol{X_1}+\boldsymbol{X_2}+(\dots)+\boldsymbol{X_n}} = e^{X_1}e^{X_2}(\dots)e^{X_n}; \qquad (3)$$

$$Y = \gamma_1\gamma_2(\dots)\gamma_n;$$

$$\text{where } \gamma_i = e^{X_i}$$

However, in the pricing scheme, the predictors are not simply aggregated by the configuration defined in (3). Instead, they follow a relativity scheme where there is a base value γ_0 that stands for a risk group with homogeneous characteristics, meaning that they share the same or very similar values across all $\boldsymbol{X_i}$ predictors. The reference for γ_0 is set to a sub-group with many policies, with the rest of the γ_i acting as relativities with respect to the base subgroup.

$$Y = \gamma_0\gamma_1\gamma_2(\dots)\gamma_n \qquad (4)$$

This configuration was preferred because it allowed analysts to see the relative effect of each predictor's value that differed with respect to the base sub-group. And the intuitiveness of this approach and the ease of communication have contributed to its use up until this day.

2.1.1.1. The EDM family

The more basic linear models cannot adapt to the nature of the variables studied in the insurance business context, the ones that are of interest for actuaries. The reason lies in the non-normality of variables such as frequencies, severities, aggregated claims or renewal probabilities. Here the first generalization of GLMs comes into play by fitting a distribution from the EDM family.

Each of the distributions in group of distributions follows the next scheme:

$$f_Y(y) = exp\left(\frac{y\theta - a(\theta)}{\phi/\omega}\right) c(y, \phi, \omega) \qquad (5)$$

Where θ is the location/canonical parameter, ϕ is the dispersion parameter, ω is an exposure parameter and the cumulant function *a(θ)*[1]. The normalizing term $c(y,\phi,\omega)$ is independent from the canonical parameter and is thus not analyzed any further in the GLM setting.

By the specification of these parameters, we can obtain commonly known distributions such as the Binomial, Negative Binomial and Poisson in the discrete case, or Normal, Gamma and Inverse Gaussian for continuous variables[2].

Both the GLMs applied in our case study and our GBM models will fit one of the EDM distributions to our response variables of interest. And the reason these types of distributions are considered suited for models such as frequencies and severities lie in the intuitive fact that the former is counted by positive integer values while the latter is defined for positive real numbers, and in both cases the distributions are remarkably skewed.

This flexibility alongside the choice of an appropriate link-function are the bedrocks for GLMs development in insurance applications.

2.1.1.2. Deviance measures and performance

After building an effective GLM in the context of insurance variables we must face the constant arrival of new information and policies as time passes by. This entails that our model will be tested by new observations (policies, claims and claim amounts), then the differences between our predictions in terms of the predicted mean[3] $\mu(\boldsymbol{X_i})$ and the actual observed value $\boldsymbol{Y_i}$ will need to be measured and assessed in order to determine if these are due to the natural random process of our variable of interest or if, on the contrary, our model is failing to capture some of the essential trends and pattern behaviors of our variable expressed by the $\boldsymbol{X_i}$ values.

In linear regression, these deviances from the predicted values are measured through the sum of squared residuals. But in the GLM context we no longer have only gaussian residuals for our predicted values, so this basic measure will also need to be generalized. We can do this for the different distributions in the EDM family by means of the Deviance measures, and these depart from the basic Maximum Likelihood Estimation (MLE) method used for GLMs fitting. The concept of deviance is important not only in the context if GLMs but in any statistical learning method used for regression and with a predictive approach. To fully understand it we need to break down the following expression:

$$LR = 2\left[L_{full} - L_{\hat{\mu}}\right] \qquad (6)$$

[1] For further mathematical details and properties about the canonical function and other elements from the EDM family, the reader can refer to section 2.4 of Denuit *et al.*, (2019).

[2] The parametrization for the different EDM members can the consulted in Appendix A

[3] $\boldsymbol{X_i}$ is a vector of *n* predictor variables or features. We previously defined them as $\boldsymbol{x_1} + \boldsymbol{x_2} + (\dots) + \boldsymbol{x_n}$

$$LR = \frac{2}{\phi}\sum_{i=1}^{n}\omega_i[y_i(\widetilde{\theta_\iota} - \widehat{\theta_\iota}) - a(\widetilde{\theta_\iota}) + a(\widehat{\theta_\iota})] \qquad (7)$$

LR represent the Likelihood Ratio, and it is defined as two times the difference between the log-likelihood of a saturated model (L_{full}) and the current log-likelihood fitted model $L_{\widehat{\mu}}$ representing in our case the GLM model. The saturated model is the one that perfectly fits the data used to build the model (train data) and it essentially reproduces the observations Y_i as they are. It makes sense then that this model attains the maximum possible value of log-likelihood, so it is used as a benchmark to see to what extent is our model fitting the available data.

In equation (7) we can see the expanded expression, also known as the *scaled deviance* due to the presence of the dispersion parameter ϕ. Therefore, the Deviance measure can be written as:

$$D(y_i, \hat{\mu}) = 2\phi\left[L_{full} - L_{\widehat{\mu}}\right] =$$

$$= 2\sum_{i=1}^{n}\omega_i[y_i(\widetilde{\theta_\iota} - \widehat{\theta_\iota}) - a(\widetilde{\theta_\iota}) + a(\widehat{\theta_\iota})] \qquad (8)$$

The previous expression involves the cumulant function $a(\ldots)$ as well as the canonical parameter θ_i. This means that the formula for the deviance will be particular for each EDM family member distribution. We will illustrate the exact expression of the deviance for the distributions used in our models in chapter 5 for Methodology.

2.1.2. GLMs currently in the industry and in research

GLMs have a strong foundation on traditional statistical theory both through hypothesis testing and confidence intervals (Ohlsson, E. & Johansson, B., 2015). The latter is used to assess the relativities for sub-groups with little information available for the policies that are included in them in terms of registered claims.

But not only that, GLMs are a consolidated methodology and procedure in general insurance companies across the world. Practitioners not only from the pricing and analytics teams but also from management and strategy use the information from these models to make key decisions.

The multiplicative structure helps determine tariff strategies in an interpretative fashion such as when the age is between 18 and 25 for drivers and the model shows a relativity of plus 30% in frequency and/or severity, therefore knowing the effect of each characteristic on the main tariff. Additional effects like the gender of the driver, or the age of the car, will have an added relativity that will help to understand and communicate the information of decision makers.

Although GLMs help to capture many types of information about the policyholders, there are still limitations to this, as the deregulation that began in the 90s has not been absolute, and there are some legal limitations that the actuarial pricing analyst will have to consider. This often leads to a pricing division between technical rates, using all the information available for analytical purposes, and restricted rates, used for tariffs but limited by regulations.

In terms of software, GLMs are included in standard open-source libraries in R, Python, SAS, MATLAB and other widely used programming environments as well as commercial specialized software provided by consultancy firms.

Most of these resources seek to address the following steps in model training and testing:

- Feature engineering: after selecting the variables, we have to arrange them properly before fitting the model. GLMs work well with categorical values and can even bin into categories numerical and continuous variables, rearranging the feature space through dummy encoding[4].
- Fitting of the model: using MLE by the numerical optimization methods already coded in the functions.
- Testing the results: both in-sample and out-of-sample Generalization Error as well as statistical significance of the fitted parameters alongside confidence intervals.

But the research around GLMs in insurance applications is still in constant expansion as for this date. Some developments are being concentrated on innovative technology-based lines of business such as the work of Sun, S. *et al*., (2020) using GLMs for driving risk assessment using data from Internet of Vehicles technology in the context of Usage-Based Insurance. Other relatively recent works on GLMs can be found in Davoudi Kakhki, F. *et al.*, (2018) about worker's compensation of large claims or the work of Ng, S. *et al*., (2019) on GLMs for pricing with deductibles in non-life insurance.

Next, we will move to the algorithmic paradigm of statistical learning through decision trees, discussing their operating nature and their potential contributions in the risk costing process.

2.2. Decision Trees and applications

Now we move to a different approach towards predicting our response y_i. In this section, we will review a broad family of methods that do not have the intention of replacing the sophistication of GLM for risk costing/tariff analysis by themselves. But rather, they will contribute with some key advantages that will be thoroughly discussed latter.

Decision Trees seek to partition the empirical distribution of the features in a way that each subset corresponds with a specific output or values of the response y_i. Through many of these partitions, mainly based on conditionals on the values of the features $\boldsymbol{x_i}$ (one at a time) and in different stages following a hierarchical top-down structure, we end up

[4] We will discuss dummy encoding in the Methodology Chapter and its configuration in the development of our case study.

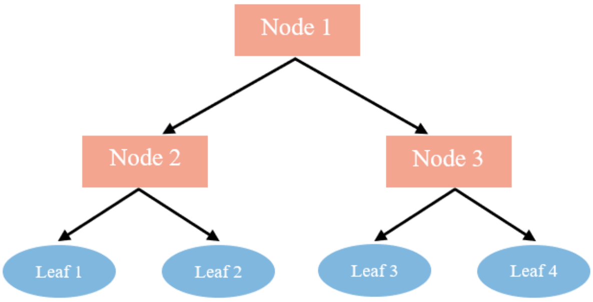
with cells of observations that are predicted/explained by the conditionals applied to its features by the Decision Tree.

These models can be applied to Regression or to Classification problems, and although our focus will be on the former, both of them have valid applications in insurance. We will review the fundamental properties of each of them in the following subsections.

2.2.1. Classification Trees

The first type of Decision Tree are the ones that seek to predict the output of a response that takes a limited set of values, most of the times we will be talking about a few different values that can go from boolean type yes/no responses to categorical responses such as blood type, region of origin for a person based on their accent or, the renovation or cancellation of a policy based on previous behavior pattern.

Figure 1: Basic scheme of a Decision Tree.



Source: own elaboration

The goal of Classification Trees (CDT) is to partition the available data in a way that the cells of data, more commonly known as leaves or terminal nodes, are homogenous. Each partition is defined as a decision node, where a specific condition based on one of the features is applied. The condition is always one of true/false, separating the data into two splits, those who yielded true to the left and those resulting in false to the right. Then the same process continues until the tree has reached its maximum depth or no further improvements in homogeneity can be archived.

To illustrate the basic mechanics, we will first define the data available as the training data, and the process of building the model as learning from the data.

We depart from a set of $\{(y_i, \boldsymbol{x_i}); i = 1, \dots, N\}$ available observations of our case study, where $\boldsymbol{x_i} = \{x_{1i}, x_{2i}, \dots, x_{3i}\}$ are the features values for each observation and y_i is the value for the response for each observation. From here, the models are traditionally trained using only a portion of the available observations, commonly using around 70% or 80% of the available data as a training set, leaving the rest as a test set of observations used to evaluate the predictive performance of the model.

Now, as we mentioned before, the partitions or splits of the training data based on conditional values of the features will be made by the algorithm of the Decision Tree by minimizing a measure of impurity. This measure is commonly the Gini Index:

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2 \qquad (9)$$

Where $p_{i,k} = \frac{Class\ k\ Observations\ in\ node\ i}{Total\ Observations\ in\ node\ i}$

If we are dealing with a portfolio renewal problem with n = 2, we have k = 1 for renewal and k = 2 for cancellation, while $i = 1, 2, \dots, \mathcal{B}$ with $\mathcal{B}$ as the total number of nodes. This is not the only measure for impurity, but is the simplest and most computationally efficient alternative. Sometimes, the Entropy Index[5] is also considered, though its results are almost identical to the Gini index and its run time tends to be slightly higher than the latter.

With our impurity target measure we will have trees capable of adapting to our data excessively well, not only capturing the main trends of the data but also reproducing the noise in our samples, with leads to the problem of overfitting. To avoid this problem, the Decision Tree model has several hyperparameters used for limiting the expansion of the Tree both as restrictions before partitioning (pre-pruning) and after building the tree (post-pruning). Here are some of examples:

Pre-pruning:

- *Maximum Depth*: in figure 1 the depth is 2, having two levels of decision nodes with node 1 in level 1 and nodes 2 and 3 for level 2. This parameter must be set accordingly with the nature of the problem to limit the expansion of the tree.
- *Minimum number of samples at node*: if the number of samples is not big enough, then partitioning again might not render any solid conclusions.
- *Minimum number of samples at leaf*: if the resulting samples in each of the two leaves born form the partition at the node are not enough, then evaluating the classification output for those leaves might not be reliable.
- *Maximum number of leaves*: similarly, to maximum depth, this parameter intends to control the expansion of the tree.

Post-pruning:

- *Minimal Cost-Complexity Pruning*: a method of regularization through a parameter called *alpha* used to penalize the excess of complexity of a tree when some of its splits cannot render a good enough improvement in homogeneity in its child nodes or child leaves.

Although there are more parameters conditioning the learning of a Decision Tree model, the ones just mentioned are considered the most important, and will be relevant

[5] The Entropy measure for impurity is equal to $\sum_{i=1}^{n} -p_{i,k}\, log_2(p_{i,k})$ and it is included as an optional argument in mainstream packages such as scikit-learn in Python.

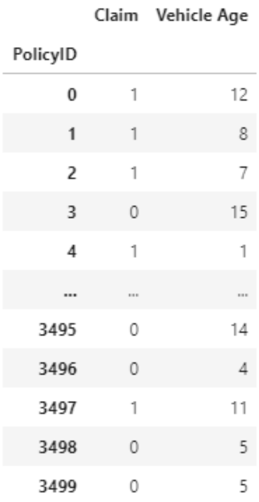
latter in our work, particularly in the Methodology chapter when setting the hyperparameter tunning process, a fundamental step in any Machine Learning model where the optimal values for these parameters are chosen based on each particular case study.

2.2.1.1. Example of Classification Tree

We just explained the essentials of Decision Trees and their operating procedure for classification problems, but it is also convenient to show a practical example of this model performing and working to provide predictions.

In the following example, we have simulated a dataset of motor insurance policies[6] for different vehicles where the response variable y_i is 1 if there was a claim reported during the observed period and 0 if there was no claim. The sole feature of the dataset corresponds to the vehicle age with range from 0 years to 20 years.

Table 1: Simulated dataset of motor insurance claims



	Claim	Vehicle Age
PolicyID		
0	1	12
1	1	8
2	1	7
3	0	15
4	1	1
...	...	...
3495	0	14
3496	0	4
3497	1	11
3498	0	5
3499	0	5

Source: own elaboration

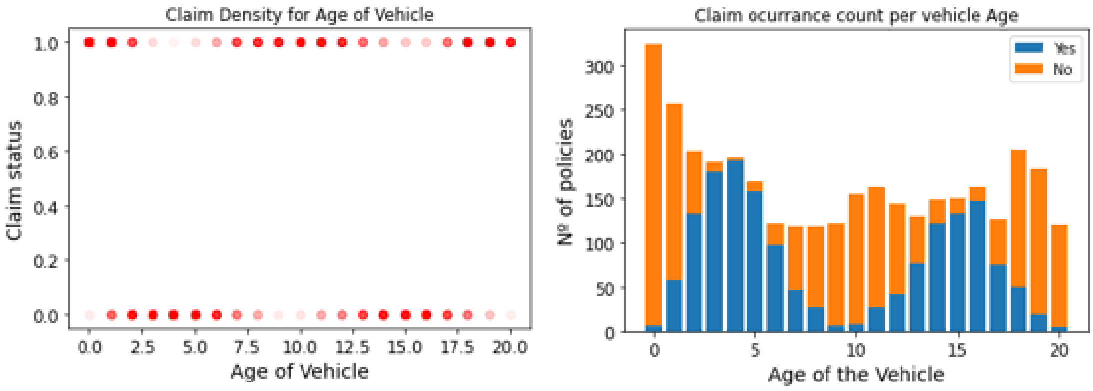
As shown on table 1 we have a total of 3500 simulated policies in a data frame of 2 variables, one of them being the feature used to predict the occurrence of a claim. To understand why a more common model such as logistic regression cannot fit the data properly, we must visualize the claim occurrence based on the vehicle age and see the type of complexity we are dealing with.

In figure 2 we can see both the density of claim occurrence or not occurrence for each vehicle age as well as a bar plot with frequencies, respectively. Through both plots we see that ages 0 and 1 corresponding to new vehicles are very prone to reporting claims, but this tendency reverts for the next years, with vehicles of ages between 2 and 6 having a much lower propensity. However, the vehicles of ages above 6 years and until ages of 13 have once again a very high propensity for filing claims, which reverts once more for

[6] The details on the data and the simulation criteria can be found in Appendix B along with the code in Python used to arrange the data, fit the model and illustrate the results.

ages between 14 and 16, with vehicles of age 17 or higher having a high propensity once more[7].

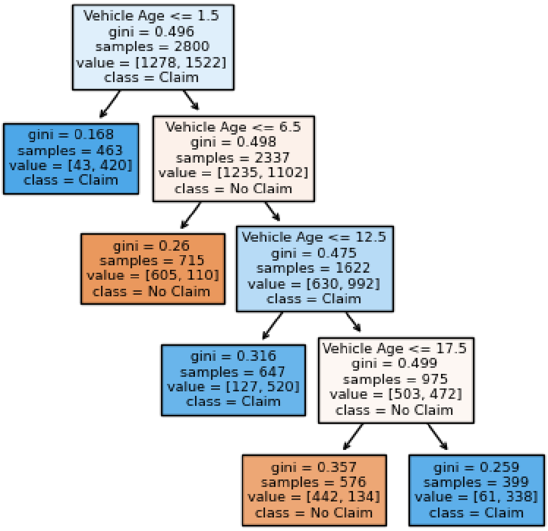
Figure 2: Claim occurrence for different vehicle ages with density of frequencies of claim/no claim (left) and bar chart of frequencies yes/no claim (right)



Source: own elaboration

If we were to renew these policies, how would we predict which ones are going to file a claim and which ones will not? Here the CDT algorithm can construct a tree model that will help us to determine the classification criteria.

Figure 3: Classification Decision Tree for simulated dataset



Source: own elaboration

With this in mind, we fit a CDT[8] to a portion of our policies as our training set, having 80% of our original policies to train the tree. After fitting the model with a *maximum depth* of 4 and *maximum leaves* of 5 we obtain the tree in figure 3.

[7] This example was made with the intention of illustrating a complex dependency structure for one feature, it does not intend to be highly realistic or empirically based, yet it can be a worthy approach to some real complexities present in insurance data.

[8] We used the function DecisionTreeClassifier(…) from scikit-learn, for more information the reader can refer to Appendix B or to the official documentation (scikit-learn, 2021).

The decision nodes can be distinguished from the leaves by the first line inside the box, corresponding to the condition made for the partition. in this case, we will always partition based on the vehicle age feature.

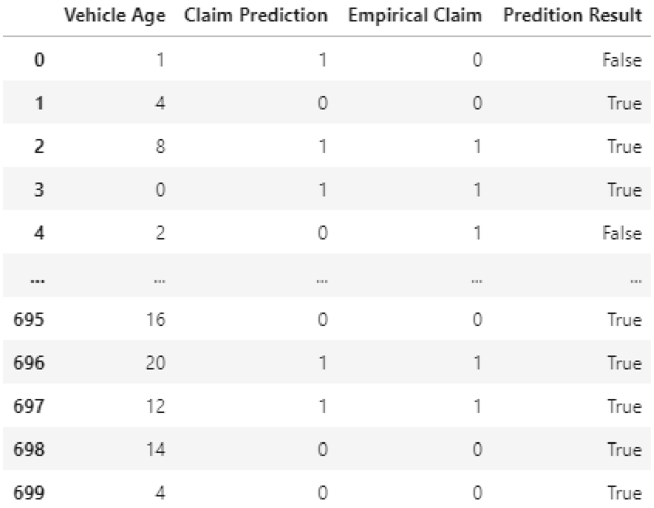
the next line of information is the Gini Index for each node/leaf. Note that the lowest impurities correspond to the leaf nodes. The next two lines give information about the number of samples/policies in each node/leaf as well as the classification distribution between Claim and No Claim.

According to the tree, those vehicles of ages[9] from 0 to 1; 7 to 12 and 18 to 20 will fill a claim during the period, while vehicles of ages from 2 to 6 or 13 to 17 will not report any claims.

Finally, each leaf classifies based on the majority observations, and this criterion will be used to classify new observations or policies that might be renewed for the next period as well as new business policies that might enter our portfolio.

In order to test the predictions of our tree, we will use the test set of policies to check the accuracy of the tree. For this we will compare the predicted claim for the feature values with the empirical information already available in the test set.

Table 2: Test dataset and predicted values from Decision Tree compared



	Vehicle Age	Claim Prediction	Empirical Claim	Predition Result
0	1	1	0	False
1	4	0	0	True
2	8	1	1	True
3	0	1	1	True
4	2	0	1	False
...	...	...	...	...
695	16	0	0	True
696	20	1	1	True
697	12	1	1	True
698	14	0	0	True
699	4	0	0	True

Source: own elaboration

After checking the accuracy of our predictions in our test set of 700 policies we have that 86% of the policies performed the way we expected. In practice, this result will have to be compared with different ML models and at different hyperparameter settings. In our case study we will address these procedures more carefully, explaining the main methods used to correctly handle them.

[9] The tree is splitting for ages 1.5, 6.5, 12.5 and 17.5; even though the ages are integer values, this is because the recursive algorithm (CART) used to fit the tree uses standardized binary splits. Further details about the splitting algorithm will be provided in subsection 2.2.3. about the CART algorithm.

2.2.2. Regression Trees

Regression Decision Trees (RDT) seek to predict numerical discrete or continuous variables such as claim counts of claim severities, respectively. Here the tree will partition the data in a similar fashion to CDTs, but instead of searching for homogeneous partitions of the data in terms of a specific value of the response they will try to homogenize the values in each leaf in a way that they are similar or closely oscillating around a common mean. This mean will typically be the output of each leaf, and it will be particular and representative of the Decision Tree prediction for each leaf.

To illustrate some of the mathematical expressions, we will closely follow the notations from Denuit, M. *et al.*, (2020). The following equation is for the predicted value of the tree model:

$$\hat{\mu}(x_i) = \sum_{t=\mathcal{T}} \widehat{c_t}\, I[x_i \in \chi_t] \qquad (10)$$

Where $\widehat{c_t}$ is the predicted value of the tree for one leaf *t* from $\mathcal{T}$ leaves, and the function I will ensure that the predicted value corresponds to that of the leaf where x_i is located. χ_i represents the partition of the feature space obtained through the conditions at each note until arriving at the leaf, $x_i \in \chi_t$ means that the observation's features match the conditions to be inside the particular χ_t for leaf t.

As we have already mentioned, the measure for the homogeneity here is based on how similar or close are the predicted values $\widehat{c_t}$ with the actual y_i values for our observations. The lower the discrepancy or deviance between y_i and $\widehat{c_t}$ the better is the performance of our prediction on the data.

In the context of RDT models, the measure for this discrepancy is the Loss Function $L(y_i, \widehat{\upmu_\iota})$ which is applied trough all observations to obtain the Generalization Error (GE):

$$\widehat{Err}(\hat{\upmu}) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \widehat{\upmu_\iota}) \qquad (11)$$

The Loss Function typically used in practice is the Squared Residuals measure, thus obtaining the GE through the Sum of Squared Residuals (SSR). From here we can already see that the fitting of RDTs is based on Least Squared Residuals, because we ultimately seek to minimize $\widehat{Err}(\hat{\upmu})$.

However, our application setting is in the EDM family of distributions, so the Loss Function we need will correspond to the measure of Deviance.

2.2.2.1. Deviance measures in the context of the EDM family

If we want our regression tree to predict count variables such as claim counts or frequencies of continuous non-negative variables like claim amounts or severities, then as in GLMs we will have to use an appropriate measure of deviance for the chosen distribution for our model. Here we are focused in choosing a support distribution that is realistic for the response variable at hand.

Recovering equation (8) from our GLMs section we have that the GE is:

$$\widehat{Err}(\hat{\mu}) = \frac{D(y_i,\hat{\mu})}{N} \qquad (12)$$

This measure will be used in our case study for both frequency and severity predictions. But next, we will have a brief look to an example of RDT using the same simulated data as in sub-section 2.2.1.1. to compare both results and see the fundamental difference in approach even for the same case study

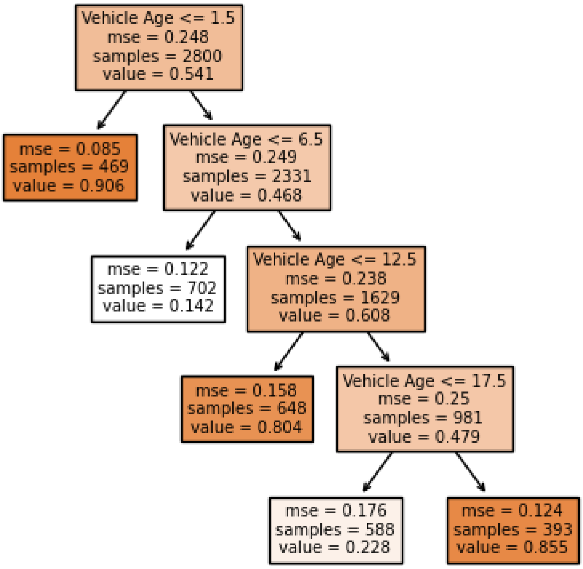
2.2.2.2. Example of Regression Tree

Our previously simulated data with Claim Occurrence 1 or 0 and feature Vehicle Age will now be used in a regression tree that will no longer classify as “Claim” or “No Claim” but will render a value between 0 and 1, and this can be interpreted as the probability of occurrence of a Claim during the observed period.

Although our response is not a continuous variable, it is possible to use these models to obtain predictions based on numeric values of our response. In this case we want to highlight some key differences with respect to CDT models, and for this purpose we have fit a RDT *maximum depth* of 4 and *maximum leaves* of 5.

In Figure 4 we can see the resulting tree. We can notice that the structure is the same as for the CDT in the previous section, mainly due to the similar nature of the features and their effects on the response and the fact that we recycled the hyperparameters.

Figure 4: Regression Decision Tree for simulated dataset



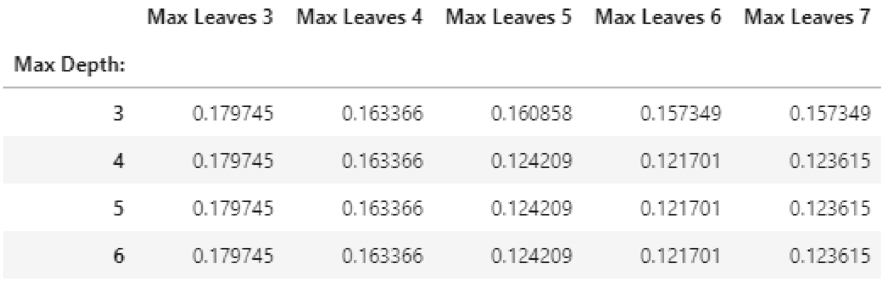
Source: own elaboration

Now the prediction for each leaf corresponds to the predicted value, or the probability that a claim occurs during the next observed period. And the pattern between the splits is similar to that of the CDT and the initially inferred propensity of age intervals 0-1, 7-12 and 18-20 being much higher than that of intervals 2-6 and 13-17.

Also, the Mean Squared Error is present in the first line of the leaves, and we can see how it its significantly lower than that of the decision nodes. Yet in terms of performance of the aggregated model, we must estimate the GE through the MSE to assess its accuracy using a test set, similar to the classification example.

We have calculated the MSE for our test samples and we obtained a value of *0.1242* which is not very informative unless we compare it with other measures coming from different ML models or even with trees build based on other hyperparameter specifications.

Table 3: Short hyperparameter configuration MSE comparison



	Max Leaves 3	Max Leaves 4	Max Leaves 5	Max Leaves 6	Max Leaves 7
Max Depth:					
3	0.179745	0.163366	0.160858	0.157349	0.157349
4	0.179745	0.163366	0.124209	0.121701	0.123615
5	0.179745	0.163366	0.124209	0.121701	0.123615
6	0.179745	0.163366	0.124209	0.121701	0.123615

Source: own elaboration

In Table 3 we can see the results by using a few models with changing values for *maximum depth* as well as *maximum leaves*, where some configurations end with a better performance on the training set than others. Particularly, we see that the tree with *max. depth* ≥ *4* and *max. leaves* = *6* has the minimum RSE.

This procedure is used in hyperparameter tuning, although in a much more complex and complete fashion, using other additional hyperparameters in the comparison process.

2.2.2.3. The CART algorithm

We had a brief look to the way a Decision Tree is fitted to data in order to predict a response variable. In ML, all procedures for fitting and training a model are based on algorithms, this is, a set of sequential steps arranged to arrive at a specific target or goal, usually in a repetitive and automated manner and being able to respond to the specific task on the spot.

The CART algorithm is exactly that, is a sequence of repetitive steps organized under the criteria of the Decision Tree model. In environments such as Python's scikit-learn this algorithm is the standard for training trees by using a binary splitting criterion, where each decision node can only output two leaves based on the condition applied to the feature that provides the best gain in information, both in terms of minimum impurity and minimum MSE/deviance.

However, even though CART tries to find the best possible split at each node, following the restrictions imposed by the hyperparameters, it does so in a recursive way, and this has a key practical implication. The algorithms with this type of operating criteria are called *"greedy"*, mainly because it only searches for the optimal splits form the top-down nodes of the tree.

This means that in a *max depth = 2* tree the best split from the first node plus the best split nodes from the second level could actually be worse than if we had chosen, let us say, the second-best split in the first node and then the best splits on second level, meaning that the resulting impurity/deviance would be less than the one obtained with the standard CART algorithm. Yet still, we settle for this solution because to find the true optimal tree is computationally infeasible for most case-studies, unless the dataset is very small.

Although we will use the CART algorithm in our applications, scikit-learn includes other complementary algorithms like ID3, C4.5 and C5.0. Additional information about these algorithms can be found in the official scikit-learn documentation.

2.2.3. Main contributions and drawbacks

Decision Trees are methods that come with a set of key advantages:

- They can capture non-linear relationships between features and response.
- They can naturally capture interactions between features, the higher the allowed depth of the tree, then higher will be the order of interactions between predictors that we will be able to model.
- They are easy to interpret, which is important in a decision-making environment where communicating is highly regarded.
- Pre-training procedures like feature engineering can be treated easily, comparing to other ML models, and missing values are also handled through imputation and estimation rules.

However, their limitations and disadvantages must also be regarded, as they are the reason why Decision Trees cannot effectively substitute other methods such as GLMs in insurance modelling by themselves:

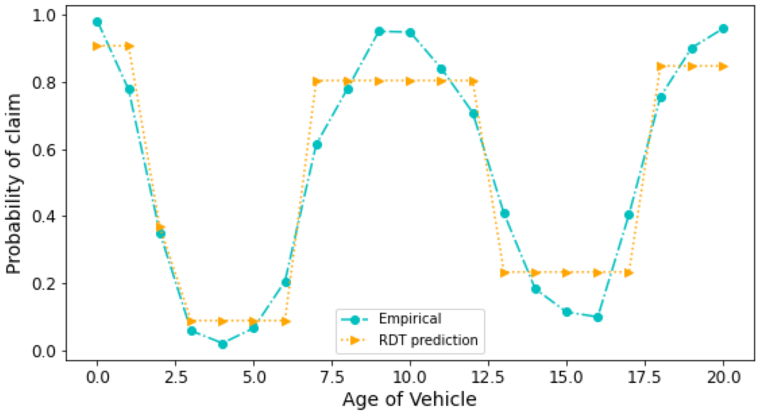
- They a very sensitive to training data, with small modifications in training samples being able to affect drastically the overall partition and predictions.
- Having the natural tendency to overfit the data, these methods need a careful hyperparameter tunning to cover this risk, although limiting their expansion and applying pruning techniques is commonly very helpful for this purpose.
- The lack of smoothness due to the rectangular partition of the feature space. Though there are smoothing procedures that can be applied over the predictions of the tree, for predicting continuous variables the jump-like predictions are commonly not precise enough .

  We can visualize in figure 5 the lack of smoothness in one of our examples[10] of RDT by comparing the empirical probability of claim occurrence for each vehicle age against the predicted probability of our model.

For these reasons we will not be building our case study based solely on Decision Trees. Instead, we will use Ensembles of Trees that will address many of the drawbacks of individual trees, providing a solid framework for effective predictive modelling.

[10] In the illustration in Figure 5 we see the prediction for the RDT with *maximum depth = 4* and *maximum leaves = 6*.

Figure 5: Regression Tree predicted probability vs. empirical probability for Vehicle Age



Source: own elaboration

However, simple Decision Trees still have applications in particular contexts, even for insurance. One example is the contribution from Quan, Z. & Valdez, E., (2018), based on multivariate decision trees for an American local property insurance fund offering multi-line insurance coverage, the authors finally find *"an improvement in prediction accuracy from that based on simply the univariate trees"*.

2.2.3.1. The variance/bias tradeoff

As we were mentioning Decision Trees are very sensitive to training data, and tend to overfit to it, this leads to a severe risk of poor generalization on unseen data. This is part of the reason that we carefully select the hyperparameters upon which our model is built, yet most of the times we sacrifice the capability of the tree to extract information about the data for its capability for correctly predict o closely predict new instances.

The variance of a ML model is the amount of difference in error of our model with respect to a testing set, and it will be high when our model has little error for the training set, because it has adapted very well to it, but when tested in unseen data, the error is much higher. This phenomenon is handled by managing the bias of our model.

The level of bias can be defined as the extent to which a model is limited on capturing the true relationship between the predictors and the response. Usually what happens is that even when our model is complex enough to capture all the elements of the dependency between predictors and response, and thus having little bias, the testing results are not satisfying because the model ends up capturing some of the noise in the data, hindering its ability to predict and increasing its variance. This is why when training a model, we account for the level of complexity in it in order to avoid overfitting and control the variance when testing

In Machine Learning, we will seek the optimal tradeoff between variance and bias by the tunning of our hyperparameters, and if our model cannot produce an acceptable balance, then we will have to use more sophisticated models with the ability to keep a low level in bias, and at the same time significantly reduce the variance of our models.

In Decision Trees, we achieve this with the ensemble techniques, and even when we accomplish our goal in this regard, we usually do it at the cost of the simplicity and interpretability of individual trees, yet the wide success of these techniques have proven it to be worthwhile.

2.3. Ensembles of Trees

In order to tackle the main limitations of some relatively "*weak*" or limited learners like basic Decision Trees, the concept of ensemble was introduced. An ensemble of learners uses the information from many different individual learners in order to make an aggregated prediction about a response.

We can compare it to a counseling of many individuals where each of them gives their opinion, and we understand that each individual has something to say about the matter because they have assessed different experiences and perspectives about the topic, yet their sole opinion might be biased or incomplete, so we aggregate their opinions in order to make the best and most complete decision.

By referencing to the example from Géron, A., (2019) in Chapter 7, we know that when throwing a slightly biased coin, we might not obtain the most probable outcome, yet when throwing the coin thousands of times, the majority outcome will most likely be the one with the bias in its favor. But the other key idea born from this example is that the ensemble will only outperform if there is little correlation between the individual learners.

From another perspective, many learners combined can "diversify" away the noise amongst them. Yet, when these are correlated, the power of diversification greatly decreases. That is why in many ensemble methods we will add particular randomization parameters to decrease this correlation.

To strengthen an ensemble, you can either diversify each individual learner (using different models) or you can diversify the training set on which they are built. In our work, we deal with the second alternative, because for most of high-performance learners based on ensembles, it is more feasible to diversify the context upon each learner grows than try to search for enough methods that are both diverse in nature and in performance.

In the next section we will use an essential method for accomplishing our goal regarding the training set. This method will be important later on in the construction of the Stochastic Gradient Boosting algorithm and in the fundamental principles of AdaBoost as a basis for boosting trees in general.

2.3.1. Random sampling through bootstrap

We pointed out that Decision Trees were very sensitive to the specific training set they were built upon, with changes in just a few instances being able to modify the whole structure of the tree. This is why the "opinion" of a single tree is deemed as incomplete or insufficient in itself in many applications.

However, this is no reason to completely disregard the opinion of a single tree, due to its advantages in capturing interactions and non-linearities of the data in a flexible way, just like we have already remarked. And because each different layout of the training data leads to valuable information about the predicted variable, we can compile a big number of trees built under many different training sets.

The purpose of bootstrap is exactly that, to resample the available instances of a training set in order to obtain a new training set for our next learner. Following with the notation of Denuit, M. *et al.*, (2020), if we have a training set $\mathcal{D} = \{(y_i, x_i); i = 1, \dots, N\}$ and do an N sized sampling with replacement then we will have a new training set $\mathcal{D}_\Theta$ where Θ is an index vector of the instances from the original training set that were selected to be part of $\mathcal{D}_\Theta$.

But in order to obtain an ensemble of learners from this procedure, we will need to repeat this procedure M times, therefore obtaining $\mathcal{D}_{\Theta_1}, \mathcal{D}_{\Theta_2}, \dots, \mathcal{D}_{\Theta_M}$ resampled training sets. And thus, for each $\mathcal{D}_{\Theta_i}$ with $i = 1, 2, \dots, M$ there will be an individual learner, extracting information from that outlay of instances in a separate manner from the rest.

The main tree-based ensembles that derive from the bootstrap resampling methods are Bagging Trees and Random Forests. In the next section we will quickly visit their principals and contributions.

2.3.2. Bagging Trees and Random Forests

In the case of Bagging Trees, we train M decision trees for M different resampled training sets. If we quickly revisit out example from section 2.2.2.2. in the application for regression, then we can extend the example by using the tree with *max. depth ≥ 4* and *max. leaves = 6* (the tree with the best performance) and apply the bagging of trees with the same configuration.

In this case, we have chosen to set $M = 100$ and the resulting MSE was 0.1210 compared to the value of 0.1217 of the individual tree shown in Table 3. We can see there is indeed and improvement in predictive accuracy.

2.3.2.1. Out-of-sample evaluation

It is a statistical fact that around 37% of all instances are not selected to be part of a particular training set $\mathcal{D}_{\Theta_i}$ and therefore, the algorithm will not learn from those instances for building the model. Although this might seem like a potential waste of information, it is actually very useful to validate the model in the learning stage, because we can evaluate the performance of our model on those samples to choose hyperparameters or to compare models before actually using the testing set, which commonly used at the last stage once the model is fully trained to test the Generalization Error measure.

2.3.2.2. Recent applications in insurance for Random Forests

However, to truly explore the capabilities of ensembles like bagging trees, more complex application domains are needed. In general, for classification, Bagging trees has the option of making the prediction based in majority voting, or by averaging the predicted probabilities. In the case of Regression, we average the result of all individual learners.

Now that we understand the basis of Bagging Trees, it is important you point out their main limitation, and it is that the resampling of training sets is not enough in itself to reduce the correlation between trees, and this is because in Bagging, the trees are typically grown without restrictions, so the tendency of the CART algorithm to building similar learners leads to "reusing" information across all iterations.

That is why Random Forests come into the scene with an additional randomization feature that helps to decrease the correlation among individual trees. They do so by selecting just a portion of all the different features or predictors that are available to match the response. This hyperparameter can the named as the number of features per tree (mfpt).

To see an example of insurance applications using Random Forests we can refer to the study by Hanafy, M. & Ming, R., (2021) where different Machine Learning methods are used to predict claim occurrence. In this case the authors used methods like Random Forests, XGBoost, K-Nearest Neighbors, Naïve Bayes or Logistic Regression; all with a classification application for a binary response, 1 for occurrence, and 0 for no occurrence of claims.

The authors use data from a Brazilian insurer with over a million of observations and 59 features from which only 35 we used. In the hyper-parameter tunning, the value chosen for the mfpt was of 28, meaning that this will be the number of randomly selected features for each tree.

It is also common, in the scikit-learn environment, to use a default value for the mfpt equal to the root of the total number of features. And for this case we would have that $mfpt \approx 6$. In any case, this additional randomization feature is part of the reason why Random Forests are considered one of the most effective ML algorithms in terms of prediction. In fact, Hanafy, M. & Ming, R., (2021) conclude that this is the best performing method for the proposed application based on measures like sensitivity, specificity, precision or AUC.

The strength of Random Forests relies in its ability to keep a low bias, yet at the same time, to significantly reduce the variance when tested. However, as we mentioned in the previous section, we do it at the expense of the simple and easy-to-communicate interpretability of basic trees. However, ensemble methods have resources to handle this problem, and we will make use of them in our Methodology chapter. These resources are Partial Dependency Plots and feature importance plots.

2.3.3. Boosting Trees

The ensembles described in the previous section can already perform classification and regression in an effective manner, providing extremely valuable information to the analyst about the variables, and their relationship with the response and between themselves. But there is another approach to ensembles of trees that has proven its worth over many applications, and it is the well-known boosting family of ensembles .

To understand boosting we must take a different outlook to bagging, where all trees can be built simultaneously without a specific order and all of their predictions are worth exactly the same. Let us look at the common results from a Decision Tree model, where we have a prediction $\hat{\mu}(\boldsymbol{x_i})$, which differs across the dataset from the actual value y_i by a measure expressed in the loss function $L(y_i, \widehat{\mu_i})$. Here we can extract each particular difference between pairs y_i and $\boldsymbol{x_i}$ by a residual measure expressed as $r_i = y_i - \hat{\mu}(\boldsymbol{x_i})$, and this measure will be the most essential element to understand the boosting approach.

The residual r_i can be interpreted as the part of y_i that could not be explained or captured by our model and, by the bias principle, it does not have to be due to noise, but it could also include trends in the data not yet discovered. The idea of boosting is to work on the 'mistakes' made by previous learners, and in the case of Boosting Trees, we will try to explain the residuals made on the previous tree by fitting new trees that will aim at predicting those errors, whether they are mismatches in a classification problem, or in a regression application, as is our current concern.

One of the common ancestors to Boosting Trees is considered to be the AdaBoost algorithm. In the next section we will briefly outline its working pattern to better understand afterwards the purpose of Gradient Boosting Trees.

2.3.3.1. The AdaBoost algorithm

This classification algorithm was first proposed by Freund & Schapire, (1997) and it can be briefly described as an algorithm that seeks to improve over the mistakes made by a basic Decision Tree with one decision node (weak learner) by fitting a new tree over a re-weighted version of the training set that gives a higher weight or importance to the instances that were misclassified.

This way the algorithm forces the new trees to correctly classify those instances, and at each iteration it evaluates their predictions, establishing an '*amount of say*' measure that will define the final influence for that learner over the aggregated voting process of the ensemble.

So, the key differences to the Bagging approach are that:

- The order upon which the trees are built is important.
- Each tree has a different *'amount of say'* in the final classification depending on its classification performance.
- Each tree is not grown in an unrestricted manner, instead they are restricted to only one decision node with two terminal nodes or leaves. This is why they are also known as '*stumps'*.

The fact that we aggregate single-split trees means that we are combining information from different *'weak learners'*, where each of them extracts a limited but valuable amount of information. Therefore, when aggregating many weak learners, we get a powerful result capable of broadly capturing hidden trends in the data.

AdaBoost was originally designed to aggregate many types of learners like Decision Trees, yet for our purposes we will show the basic steps of the algorithm when the weak learners are tree stumps:

Algorithm 1:

```
➢ Step 1: Initialize sample weights $\omega_{ti} = 1/N$
➢ Step 2: for $\boldsymbol{t = 1\ to\ M}$:
    I.   Fit a tree stump to available training set $\mathcal{D}_t$.
    II.  Compute the weighted error of the stump on the training samples:
             $\varepsilon_t = \frac{\sum_{i=1}^{m} \omega_{ti} I\left(y_i \neq \widehat{\mu_t}(x_i)\right)}{\sum_{i=1}^{m} \omega_{ti}}$
    III. Compute amount-of-say:
             $\alpha_t = ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
    IV.  Update weights and normalize:
             $\omega_{(t+1)i} = \omega_{ti}\, exp\left[\alpha_t I\left(y_i \neq \mu_t(x_i)\right)\right] \times C$
         where
             $C = \frac{1}{\sum_{i=1}^{N} \omega_{(t+1)i}}$
    end for
➢ Step 3: Compute output: $\hat{\mu}^{AdaBoost}(x_i) = \sum_{t=1}^{M} \widehat{\mu_t}(x_i)$
```

In the next section we will visit another fundamental element in the Boosting Trees learning process, and it will be present in the algorithms discussed ahead.

2.3.3.2. Forward Stagewise Additive Modelling for Boosting Trees

We have already established that in the predictive modeling environment of ML algorithms the goal is to approximate the function that links the features/predictors to our response variable, this is:

$$f(x): \boldsymbol{x} \longrightarrow y \qquad (13)$$

We say we approximate the function as $\hat{f}(x)$ because the true function is always unknown in a real-world environment. In a Boosting algorithm, we will build $\hat{f}(x)$ based on many individual weak learners, just like we depicted in the AdaBoost example.

Therefore, we can define the boosting function as (14):

$$\widehat{f^{boost}}(\boldsymbol{x}) = \sum_{t=1}^{M} \beta_t h(\boldsymbol{x}; \boldsymbol{a_t})$$

This is known as Additive Modelling, and in our case the function $h(x; a_t)$ will be a Decision Tree with restricted growth, so we can define our weak learner as $T(\boldsymbol{x}; \boldsymbol{a_t})$ where $\boldsymbol{a_t}$ are the set of parameters delimiting the feature space partitions.

Also, and coming from the generalization presented for GLMs, we can modify the relationship between response and predictors form the identity link to a link-function of choice surrounding the left side of equation (14) by the inverse $g^{-1}(\dots)$. Thus, and by taking the corresponding Loss Function, we arrange the following optimization (15):

$$argmin_{\{\beta_t;\boldsymbol{a_t}\}_1^M} \sum_{i=1}^{N} L\left(y_i, g^{-1}\left(\sum_{t=1}^{M} \beta_t T(\boldsymbol{x_i}; \boldsymbol{a_m}) \right) \right)$$

In practice, the previous equation is solved through a greedy Forward Stagewise approach, this means that we will add each learner in a sequential fashion, just like in basic Decision Trees when optimizing the trees without readjusting the previous steps.

These are the elements of the widely known Forward Stagewise Additive Modelling (FSAM), a fundamental part of the Boosting Trees algorithms applied in practice. This procedure guarantees a computationally viable alternative, and the evidence of success of boosting algorithms among many applications has shown that the potential loss in optimality is not severe in most cases ,while the final results are still quite remarkable.

In the next section, we will show the basics of the Gradient Boosting algorithm for regression, as well as some recent applications in insurance loss cost modelling.

2.3.4. Gradient Boosting Trees and Loss Costing applications

Also known as Gradient Boosting Machines (GBMs), this algorithm was fully developed and presented by Friedman, J., (2001) as a way of applying a FSAM approach to build a strong and effective learner based in many weak learners, mainly small Decision Trees.

Gradient Boosting Trees are very complete algorithms both for classification and regression (GBTC and GBTR respectively) thanks to their computational efficiency and use of randomization and regularization parameters that decorrelate individual trees and avoid overfitting. Also, most of their software supports have great capacity for handling missing values, and the preprocessing of features is not very demanding, making GBM a very data-adaptative model. Their applications in fields like insurance have been growing and thriving during the last decade and are expected to continue on that trend on the foreseeable future.

Now we will detail the steps of the algorithm and explain the role of the main hyperparameters:

Algorithm 2:

- Step 1: From training set $\mathcal{D}$, initialize first base predictor:

$$f_0(x) = argmin_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$$

- Step 2: **for *t* = 1 to M:**

  I. Compute partial residuals:

$$r_{i,t} = -\frac{\partial L\big(y_i, f_{t-1}(\boldsymbol{x_i})\big)}{\partial f_{t-1}(\boldsymbol{x_i})} \qquad where\ i = 1, \dots, N$$

  II. Fit weak Decision Tree *t* on $r_{i,t}$ values:

$$\widehat{\boldsymbol{a_t}} = argmin_{\boldsymbol{a_t}} \sum_{i=1}^{N} \left(r_{t,i} - T(\boldsymbol{x_i}; \boldsymbol{a_t})\right)^2$$

  III. Compute output values of the tree for each leaf j in $1, \dots, \mathcal{C}$:

$$\widehat{\gamma_{J,t}} = \text{argmin}_{\gamma_{j,t}} \sum_{j=1}^{\mathcal{C}} L\left(y_i, g^{-1}\big(f_{t-1}(\boldsymbol{x_i}) + \gamma_{j,t}\big)\right)$$

  IV. Update $f_t(\boldsymbol{x_i}) = f_{t-1}(\boldsymbol{x_i}) + \upsilon\, T(\boldsymbol{x_i}; \boldsymbol{a_t})$

  end for

- Step 3: Compute output: $g^{-1}\big(f_M(\boldsymbol{x_i})\big) = \hat{\mu}^{GradientBoost}(\boldsymbol{x_i})$

In the previous algorithm we have γ as the output or prediction from each weak learner starting from the learner $f_0(x)$, which in practice can be any initial guess according to the judgment of the practitioner, or it can even be a previous model like a GLM or a basic linear regression that will be reevaluated against the GBTR or the GBTC in the case of an initial logistic regression model.

In step 2.I we have the negative gradient, and it represents the partial derivative of the chosen loss function with respect to the initial prediction γ. This is the basic step to compute the so-called partial residuals, called ‘partial’ because they do not intend nor always are the residuals we know as $r_i = y_i - \hat{\mu}(\boldsymbol{x_i})$. Instead, partial residuals are an intermediate metric spread across all the iterations of the algorithm from 1 to *M*, and they will also depend in the loss function of choice.

The weak learner will be trained by the usual optimization CART algorithm to obtain all partition parameters a_t. Once we have built the corresponding tree, we will calculate the output for each leaf by the optimization in step 2.III where we will take into account the accumulated information in $f_{t-1}(\boldsymbol{x_i})$ plus the information coming from the newest tree.

With these outputs, we will be able to add the tree to the function $f_{t-1}(\boldsymbol{x_i})$ built up to that moment. However, it is very common in GBM applications that the information coming from each tree is only added partially, and the proportion to include for the update is determined by the regularization parameter υ with range from 0 to 1. The inclusion of υ as a hyperparameter for our model is called shrinkage, and although a low value can make computation for the hole model much slower, it can also provide with much effective results in terms of overall performance.

By sequentially repeating the process, we can arrive at out final predictor in step 3. Although this is the main scheme for GBM algorithms, we can add other randomization features like the bootstrap resampling without replacement at the begging of each iteration in step 2. This randomization is done to obtain a new training set $\mathcal{D}_{\Theta}$ that will be used to fit the next tree, and it is used to alleviate the computation process by reducing the variance of the model thanks to a lower correlation among trees. Its implementation means we are using Stochastic Gradient Boosting either for regression or for classification (SGBR and SGBC respectively).

Other hyperparameters like *max. depth* of individual trees, will define the weakness of each learner but also their ability to capture interactions between features, this will be an important tradeoff to consider in the tunning process.

To wrap up, it is important to notice that although interpretability for GBM models might be lower than that of an individual tree, as it is natural for any ensemble technique, partial dependency plots and feature importance plots might suffice for this matter.

2.3.4.1. Applications of Gradient Boosting Trees in insurance

The impact of GBM algorithms in insurance modelling is on a growing trend as of this day. The interest in the actuarial research community keeps growing since some of the first proposals of Gradient Boosting applications like the one from Guelman, L., (2012) where the algorithm is used for loss costing by modelling both frequencies and severities separately on a dataset from major Canadian insurer.

The author of the previous paper compares the performance of the algorithm with a GLM by comparing the rates or pure premiums (predicted loss costs) of both models as a ratio of GBM on GLM. The main conclusion is that *“the GLM-loss ratio increases whenever the GB model would suggest to charge a higher rate relative to the GLM”*. The paper concludes that the behavior of the GLM Loss Ratio to higher rates from the GBM model shows a better performance of the latter.

The interest in expanding the influence of many ML methods, including GBM models, has been acknowledged the Society of Actuaries in the survey from Diana, *et al.*, (2019) where techniques such as regularization, Decision Trees, Ensembles, Neural Nets or Multivariate Additive Regression Splines (MARS) are presented in different examples of insurance applications.

In addition, Gradient Boosting has been innovating its software implementations even in recent years, and this proves that this type of algorithms still has a room for improvement and further sophistication. One example is the XGBoost implementation (Chen & Guestrin, 2016) which is already renowned for being used in ML competitions like Kaggle. In this software packages, the main goals are to be *“efficient, flexible and portable”*.

Another valuable application of GBM is in the context of Claim Reserving, developed by Ahlgren, M., (2018). In this study, once again, the performance of the GBM model is tested against a classical GLM, yet the most interesting conclusion is that the former actually has a higher prediction error than the latter. The author points out that the conclusion may be caused by a lower number of features included in the dataset, with less complex interactions and non-linearities. As one of the most important conclusion we have that *"the GLM already possesses certain features that make it suitable for claims reserving without making as many adjustments in the model implementation"* in contrast with the rather more complex *"fine tuning"* of a GBM model.

The fields inside the actuarial practice are more diverse than the ones just mentioned, and the variants of GBM models developed for loss costing are expected to grow and gain sophistication in the next years. One of these new proposals are the model from Su, X. & Bai, M., (2020) based on a Stochastic Gradient Boosting model that takes into account the dependency between frequencies and severities. The model is labeled by the authors as D-FSBoost (Gradient Boosting Frequency-Severity Model).

3. Case study and data

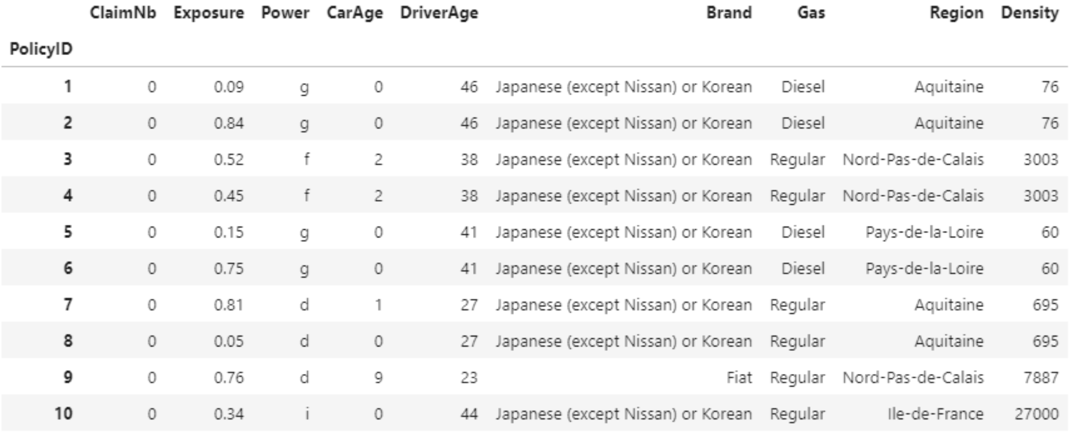
The dataset used in our work seeks to reflect the multivariate and bigdata environment common in insurance companies. Therefore, we have chosen a policy portfolio large enough to extract statistically significant relationships from, using several features to predict the behavior of common actuarial response variables.

3.1. Description of the dataset

The data set in question belongs to the CASdatasets repository in the R package[11] with the same name. In the documentation for the referenced package, we find our data in the freMTPL section as "*French Motor Third-Part Liability datasets*". In particular, we will be using the following datasets described in the documentation:

- freMTPLfreq: a motor third-part liability portfolio comprised of 413,169 policies with a set of risk features along with the observed claims number for each policy according to the time period or exposure of observation, ranging from 0 to 1.
- freMTPLsev: contains a total of 16,181 claim amounts for the frequencies reported in the policies inside the portfolio. The policies are identified by a Policy ID column in each dataset.

Table 4: Frequencies for exposure period and risk features in freMTPLfreq dataset:



	ClaimNb	Exposure	Power	CarAge	DriverAge	Brand	Gas	Region	Density
PolicyID									
1	0	0.09	g	0	46	Japanese (except Nissan) or Korean	Diesel	Aquitaine	76
2	0	0.84	g	0	46	Japanese (except Nissan) or Korean	Diesel	Aquitaine	76
3	0	0.52	f	2	38	Japanese (except Nissan) or Korean	Regular	Nord-Pas-de-Calais	3003
4	0	0.45	f	2	38	Japanese (except Nissan) or Korean	Regular	Nord-Pas-de-Calais	3003
5	0	0.15	g	0	41	Japanese (except Nissan) or Korean	Diesel	Pays-de-la-Loire	60
6	0	0.75	g	0	41	Japanese (except Nissan) or Korean	Diesel	Pays-de-la-Loire	60
7	0	0.81	d	1	27	Japanese (except Nissan) or Korean	Regular	Aquitaine	695
8	0	0.05	d	0	27	Japanese (except Nissan) or Korean	Regular	Aquitaine	695
9	0	0.76	d	9	23	Fiat	Regular	Nord-Pas-de-Calais	7887
10	0	0.34	i	0	44	Japanese (except Nissan) or Korean	Regular	Ile-de-France	27000

Source: own elaboration based on dataset from (Dutang & Charpentier, 2020)

A similar dataset is included in the same freMTPL section, these are the tables freMTPL2freq and freMTPL2sev, and they include a different set of features and observations. However, works like the one from Noll, A. *et al.*, (2015), where different ML methods like Grandient Boosting and Neural Networks are used for modeling frequencies and are compared with a Generalized Linear Model. However, in our work we will focus on the previously described dataset for the modeling process.

[11] Check the official documentation in the official web documentation file from (Dutang & Charpentier, 2020): http://cas.uqam.ca/pub/web/CASdatasets-manual.pdf

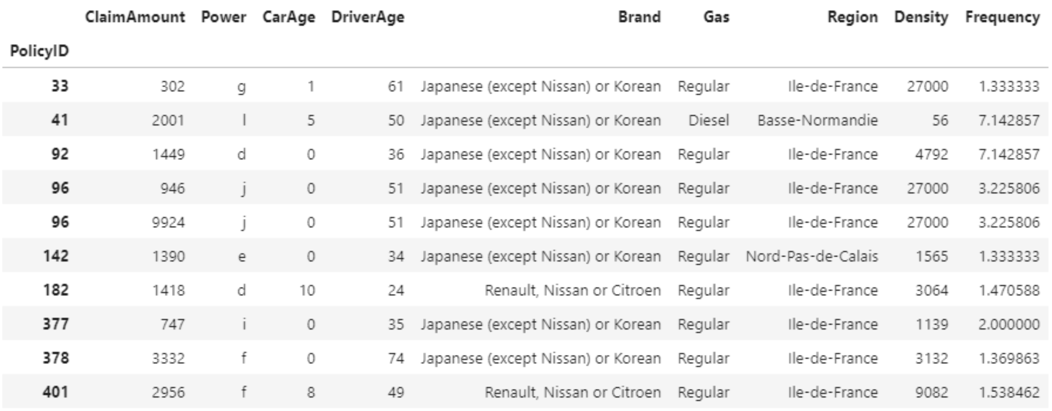
We can also give a brief description based on the documentation from (Dutang & Charpentier, 2020) for each feature included in the datasets before moving on to the Exploratory Data Analysis (EDA). In the case of freMTPLfreq we have:

- PolicyID: identifies each policy and enables to match with claims in freMTPLsev.
- ClaimNb: Number of claims for the observed exposure period.
- Exposure: The period of exposure measured in years.
- Power: Categorical value describing the power of the car.
- CarAge: Vehicle age measured in years.
- DriverAge: The age of the driver measured in years (minimum 18 years for French drivers).
- Brand: Categorical variable arranged in 7 brand groups[12].
- Gas: Car gas type, either diesel or regular.
- Region: 10 different regions in France.
- Density: the population density as inhabitants per km^2 in the driver's city of residence.

These features can be matched to the claims in freMTPLsev, so that they can be used as predictors for the response variable ClaimAmount. Although the EDA was done partially in R for variables' distribution visualization, the data wrangling, preparation and modeling was performed in Python.

In the next table we can see the result of a joint that matches the freMTPLsev dataset with the freMTPLfreq table using Policy ID as the join columns. For this we have used the pandas package with the function merge( ). The table shows the first 10 claim amounts:

Table 5: Claim Amounts



PolicyID	ClaimAmount	Power	CarAge	DriverAge	Brand	Gas	Region	Density	Frequency
33	302	g	1	61	Japanese (except Nissan) or Korean	Regular	Ile-de-France	27000	1.333333
41	2001	l	5	50	Japanese (except Nissan) or Korean	Diesel	Basse-Normandie	56	7.142857
92	1449	d	0	36	Japanese (except Nissan) or Korean	Regular	Ile-de-France	4792	7.142857
96	946	j	0	51	Japanese (except Nissan) or Korean	Regular	Ile-de-France	27000	3.225806
96	9924	j	0	51	Japanese (except Nissan) or Korean	Regular	Ile-de-France	27000	3.225806
142	1390	e	0	34	Japanese (except Nissan) or Korean	Regular	Nord-Pas-de-Calais	1565	1.333333
182	1418	d	10	24	Renault, Nissan or Citroen	Regular	Ile-de-France	3064	1.470588
377	747	i	0	35	Japanese (except Nissan) or Korean	Regular	Ile-de-France	1139	2.000000
378	3332	f	0	74	Japanese (except Nissan) or Korean	Regular	Ile-de-France	3132	1.369863
401	2956	f	8	49	Renault, Nissan or Citroen	Regular	Ile-de-France	9082	1.538462

Source: own elaboration based on dataset from (Dutang & Charpentier, 2020)

Note that we have dropped the variable 'Exposure' originally from the frequencies table. This is because claim amounts occur following the number of claims, and their distribution is not directly affected by the length of the exposure period. However, as we

[12] See the exact description of each categorical value in Dutang & Charpentier, (2020).

have just mentioned, the number of claims can have a dependency with the amounts for an equal exposure.

The independency between frequencies and severities has been a common assumption through many actuarial analysis settings, but in practice it can be necessary to relax this assumption to account for cases when drivers with higher frequency tend to have many small-amount claims, this can be due to several environmental or behavioral factors that could then be mapped into a series of features that could enable the modelling of this relationship.

Some proposals like the one from Garrido, J. *et al.*, (2016) is to use a conditional approach where the claim counts are modeled through Poisson regression, while claims are modelled by a log-link GLM with the claim counts as an additional predictor variable. This way the authors obtain a multiplicative structure for the pure premium where the dependency between frequencies and severities is accounted by an implicit correction factor, both convenient and easy to interpret for the analyst.

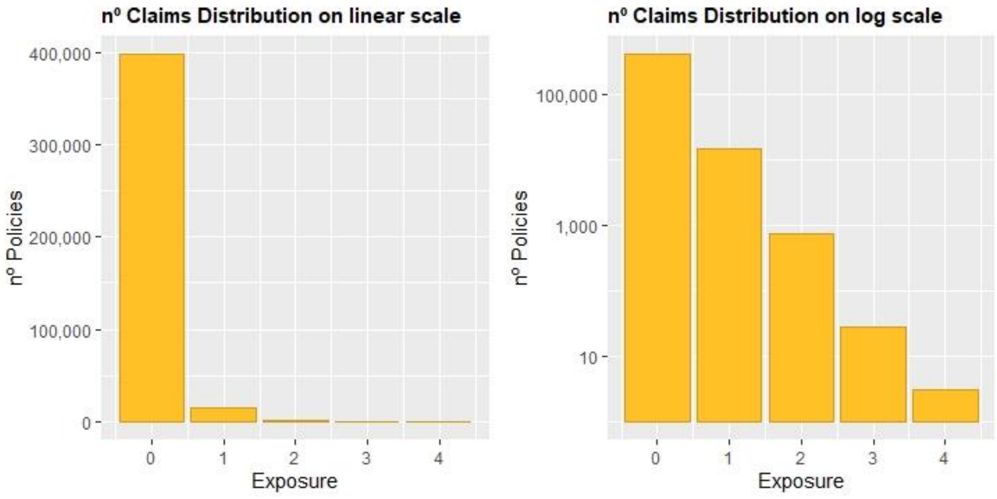
For our modelling applications a similar approach will be taken, by using as an additional predictor for the freMTPLsev table labeled ‘Frequency’ that will be defined as the ratio between the columns ‘ClaimNb’ and ‘Exposure’. This way we emulate an empirical estimate of the frequency for each policy that filed at least one claim, and this estimate can be used as a means to penalize those policies in case of renewal, similar to a bonus-malus system.

3.2. Exploratory Data Analysis (EDA)

3.2.1. Claim Counts analysis

The first stage of our EDA will be to visualize the data contained in both of our dataset tables. We will first start by visualizing our first response, claim counts or ‘ClaimNb’, in our dataset freMTPLfreq. In the fitted models, we intend to assume a Poisson distribution for the claim counts, but it is important to contrast this hypothesis with the data.

Figure 6: Claim counts distribution for different scales



Source: own elaboration

In our first depiction in Figure 6 we see a linear scaling for the claim counts, but the overwhelming proportion for 0 is an obstacle to fully appreciating the presence of higher claim counts, so the log scale helps to see the proportions for each value of ‘ClaimNb’.

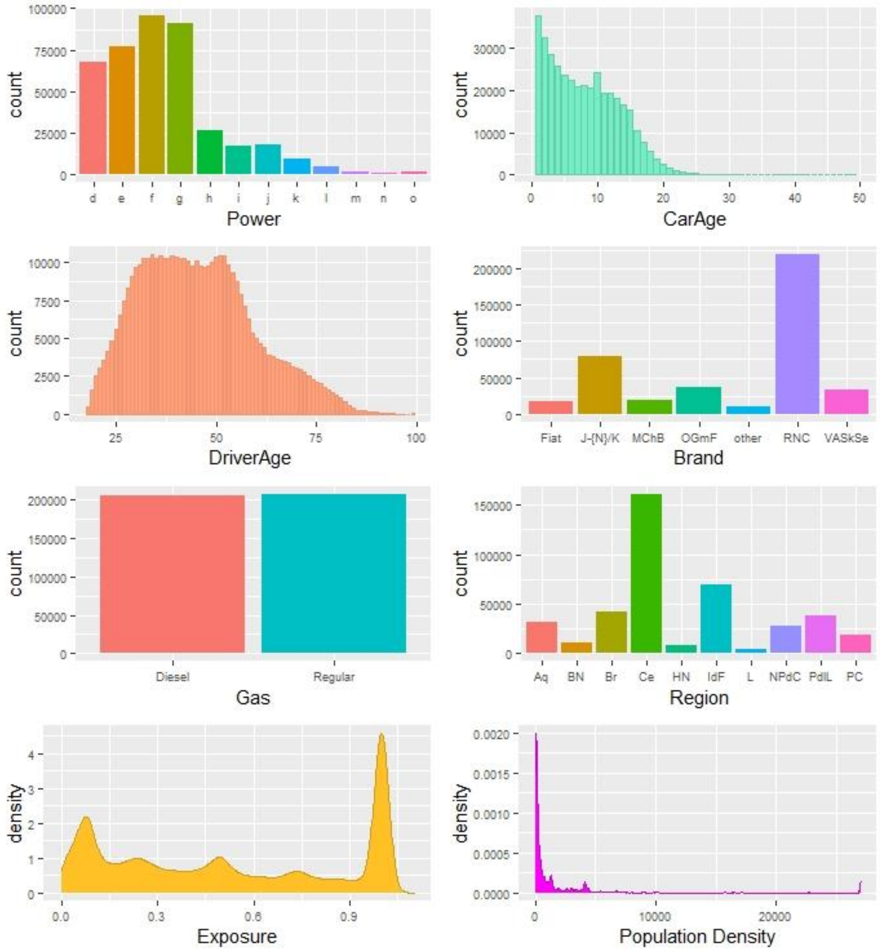
While ‘ClaimNb’ is 0 in roughly 400,000 policies, the number of observations with just 1 claim reported are around 14,000. From here the number of policies with 2 claims drops to less than a thousand, while 3 claims are as infrequent as just in 28 policies. And the maximum value of 4 claims reported is only present in a few observations out of the whole portfolio.

Now we must assess if a Poisson assumption for the distribution is reasonable, and for that purpose we compute the mean and the variance based on the data and we obtain a value of 0.03916 for the mean and a value of 0.04164 for the variance. These values being roughly similar is an indication that there are no obvious signs of overdispersion in the count distribution, so the assumption can be maintained.

3.2.2. Feature analysis

In the next figure we will show a plotting of the distributions for the different features included in freMTPLfreq. We have in total 7 predictor features plus the exposure, with a total of 4 categorical variables and 4 numeric variables.

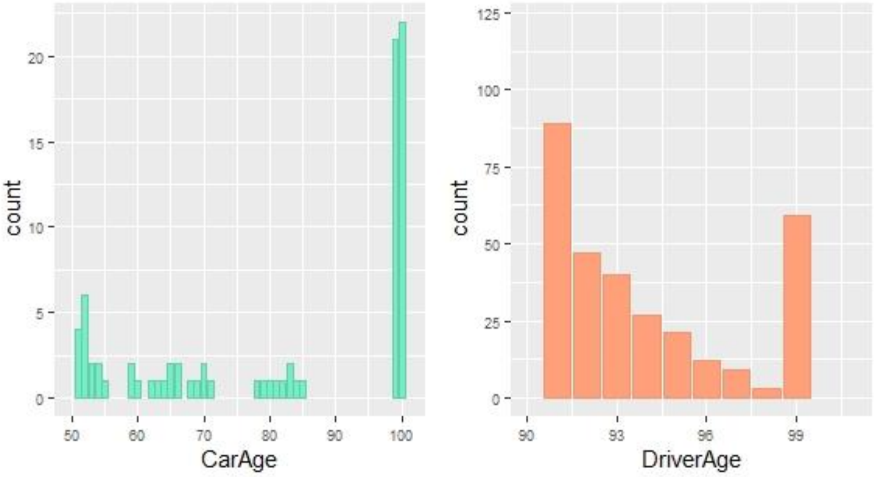
Figure 7: Features distributions



Source: own elaboration

Features like CarAge have outlier values, so it is important to assess the most extreme values of the distribution and make the decision about excluding them or not based on a specific threshold. In this case, we decided to eliminate all policies with CarAge above 50 years, which led to a deletion of 82 Policies and 3 Claim Amounts. We performed a similar procedure on DriverAge equal to 99 years, this time eliminating 59 additional Policies with also additional 3 Claim Amounts. The final row count for freMTPLfreq is 413,028 observed Policies and 16,175 reported claims in freMTPLsev.

Figure 8: Outliers distribution based on selected criteria



Source: own elaboration

We can see that the presence of Policies for CarAge > 50 years is residual, while for DriverAge we decided to keep values up to 98 years, only deleting the cap at 99 years due to the sudden peak will respect to the immediately previous values, leading us to think these might be miss imputations, because otherwise we would expect a slow but steady decease in counts as DriverAge value increases.

Finally, we must address the feature ‘Density’, which has a distribution concentrated mostly on low values. However, there are concentrations on value ranges that are separated by far from the initial concentration, as we could see in the bottom right plot in figure 7. To use this variable in our models, we have considered proposals such as to transform the feature into a discrete categorical one with ranges of values in each category, as well as applying a log-scale, but our initial strategy will be to re-scale it to thousands of inhabitants per km^2, calling this new version ‘DensityS’.

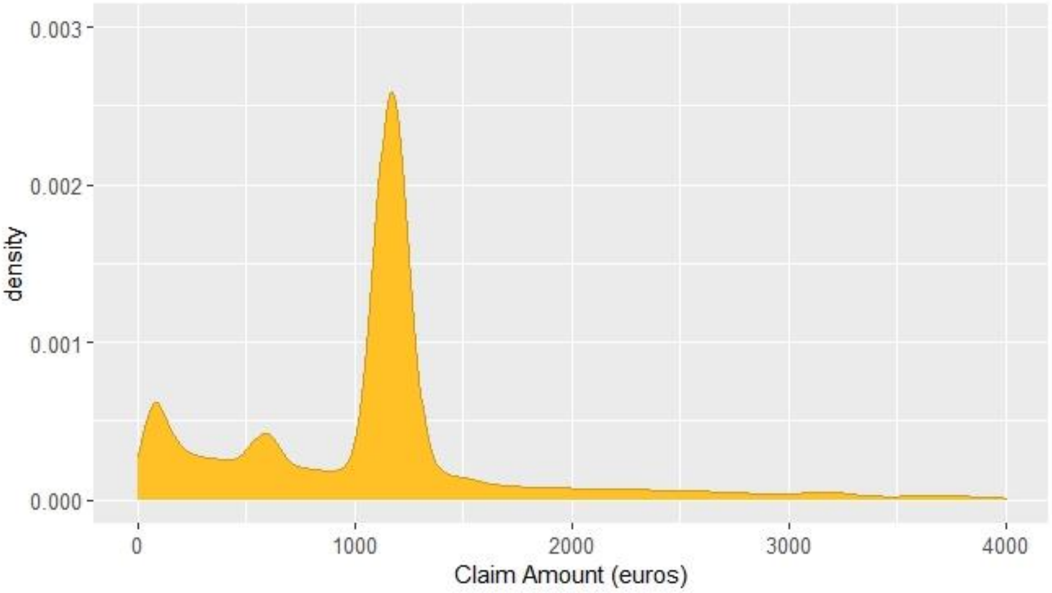
3.2.3. Claim Amounts Analysis

The next brief description should be about the available data about claims, which are exactly 16,181 claim amounts, yet after applying the previous feature preprocessing, we are left with 16,175 amounts.

In the next figure we will show the aggregated distribution of the claims amounts (we will assume that the amounts are measured in euros €). Particularly, we will see how the distribution is highly asymmetric and right skewed. The purpose of the modelling for severities is to predict the behavior of this response, and in the case of models like GLMs we have suitable alternatives for its modeling and prediction. Given that the variable is, in nature, non-negative in values and continuous, we need a regression support for such properties, and the Gamma regression is one of the most known alternatives.

In figure 9 we can observe a shortened version of the density distribution for our variable, and at a first glance it does not seem to match the shape of a Gamma distribution. Yet now we are working at an aggregated level, for the claims reported in the global portfolio, which is heterogeneous in risk nature amongst the policies they represent. Therefore, the overdispersion of the risk in many risk classes is something common and natural in insurance portfolios, that is why in many application GLMs perform an implicit risk class classification where each resulting group tends to have a more homogenous profile based on the features used to train the model.

Figure 9: Density plot of Claim Amounts (€) based on available data



Source: own elaboration

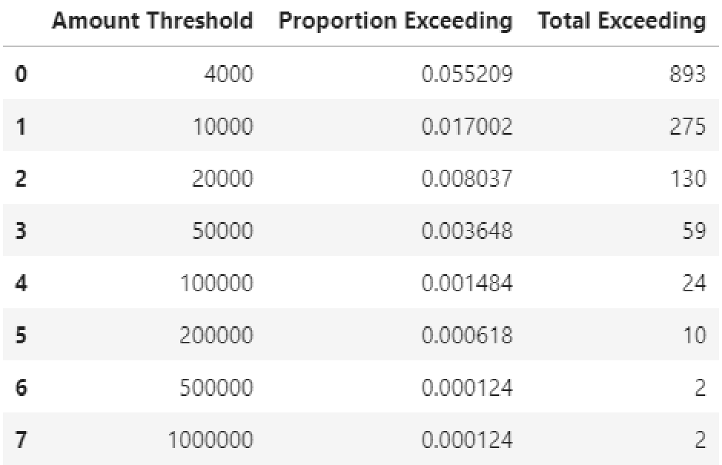
In the previous figure we have only shown a limited domain for the Claim Amounts registered in freMTPLsev. Specifically, we have established a range from 0€ to 4,000€ to show in the plot, because as a matter of fact, the highest registered claims exceed by far the mean or mode values. This is the case of long tailed distributions, very common in actuarial practice and modelling.

Now we can obtain information about the distribution of Claim Amounts, and we do so by checking what proportion of claims out of the whole claims record are above a specific threshold.

We used table 6 to summarize this information, and now we know that over a 5% of all claims were not plotted in figure 9. We also know that there are 24 amounts above 100,000€ in value, where we could have expected that the observed amounts would decrease to extinction with values not much higher than the latter.

However, as it is common in long-tailed distributions, the last values in our tail are again much higher than our last threshold, given that we have observed two amounts higher than a million €.

Table 6: Claim Amounts right tail distribution for a set of values



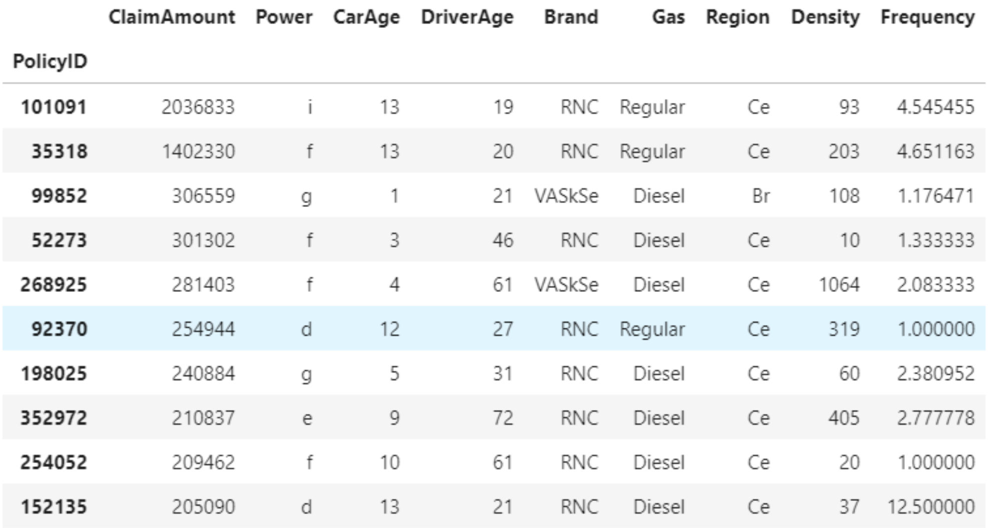
	Amount Threshold	Proportion Exceeding	Total Exceeding
0	4000	0.055209	893
1	10000	0.017002	275
2	20000	0.008037	130
3	50000	0.003648	59
4	100000	0.001484	24
5	200000	0.000618	10
6	500000	0.000124	2
7	1000000	0.000124	2

Source: own elaboration

In practice, such claims exist as the so called '*catastrophic'* events. This name is not only representative of a massive event of environmental origins, but it revolves around many circumstances converging at the same place and time and having an exponentially augmented effect on the final response.

The presence of such amounts must be carefully considered by the actuarial analyst because these can be considered as outliers in the general data and if not treated properly, they can severely affect the overall modelling process. Another way of visualizing the highest amounts is by ranking the corresponding policies, similar to our representation in table 7.

Table 7. Highest Claim Amounts and feature information



	ClaimAmount	Power	CarAge	DriverAge	Brand	Gas	Region	Density	Frequency
PolicyID									
101091	2036833	i	13	19	RNC	Regular	Ce	93	4.545455
35318	1402330	f	13	20	RNC	Regular	Ce	203	4.651163
99852	306559	g	1	21	VASkSe	Diesel	Br	108	1.176471
52273	301302	f	3	46	RNC	Diesel	Ce	10	1.333333
268925	281403	f	4	61	VASkSe	Diesel	Ce	1064	2.083333
92370	254944	d	12	27	RNC	Regular	Ce	319	1.000000
198025	240884	g	5	31	RNC	Diesel	Ce	60	2.380952
352972	210837	e	9	72	RNC	Diesel	Ce	405	2.777778
254052	209462	f	10	61	RNC	Diesel	Ce	20	1.000000
152135	205090	d	13	21	RNC	Diesel	Ce	37	12.500000

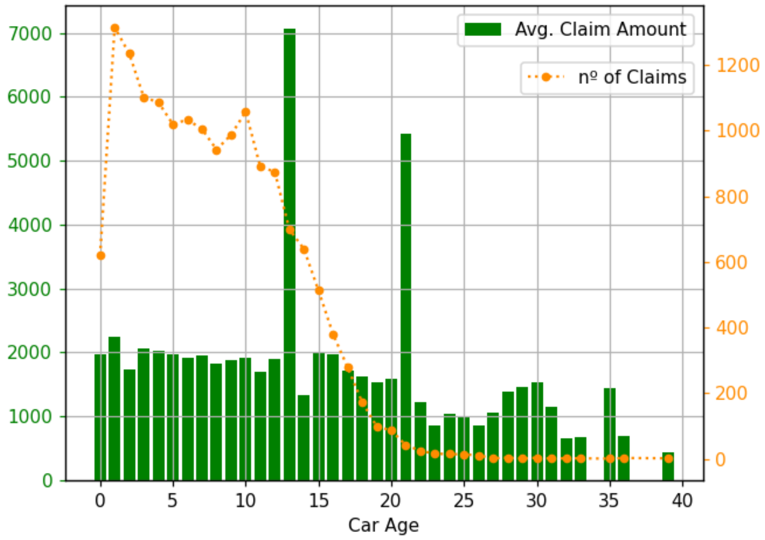
Source: own elaboration

To obtain the representation in table 7 we sorted the Claims information table by the ClaimAmount variable in descending order, so that the first row shown corresponds to the highest claim and so on. We will assume that the highest observed amounts are legitimate, and not an imputation error on the IT level of the company, as this would also have to be double-checked in practice.

3.2.4. Effect of outliers on average Claim Amounts

For us to have a visual idea on how an extreme claim amount can affect the dependency between features and response, we have elaborated a plot representing the average amount for each CarAge value. Figure 10 shows a bar plot for the average claim and a dotted line giving the number of claims reported, both for each car age group.

Figure 10. Average Claim Amounts and number of claim for each car age



Source: own elaboration

The immediate first aspect of the plot that calls our attention are the two spikes at ages 13 and 21. If we recall the results in table 7 then we will realize that the two highest claims were both for a policy of $CarAge = 13$, and this apparent coincidence is enough to sky-rocket the average claim value in that age. This can be considered a problem because we cannot know if the car age 13 is truly riskier or this is just a case of random noise. We see now how only 2 extreme values can affect the average estimate for an age with relatively high number of claims of over 600, in this case making the result more than three times higher than that of neighbor car ages.

Something similar happens with $CarAge = 21$, although here the number of claims if much lower, with only 42 reported claims. This makes the average value much more sensitive to extreme values, and in this case, the Claim Amount causing the spike is ranked 12th out of all claims, amounting to a total of 182,568€.

All these aspects will be important in order to determine the filtering threshold used to rule out extreme values or outliers previous to modelling claims.

4. Methodology

In our methodology chapter we will dive into the modelling criteria and training procedures for our selected models. We will begin by preparing our data for the GLM training of both frequencies and severities, rearranging the features and interpreting the coefficient results, statistical significance and relativities.

After comparing different configurations for our GLMs and selecting the most appropriate ones, we will focus on the training and tunning of our Gradient Boosting methods, where we will consider different hyperparameters, their added value into the modeling process and their basic interpretation.

Some essential differences in the modeling approach between GLMs and GBMs will be considered along the way, and these will set up some of the conclusions made further along our work once we start evaluating the performance of each model and their strengths and advantages in a corporate-analytical pricing environment.

4.1. GLM model fitting for frequencies

Our main software tool for GLM training, evaluation and interpretation will be the *statsmodels* python library. This package is open-source, easily accessible and well documented, having all the configurations to correctly run our fittings by following our case study particularities[13].

4.1.1. Feature selection and preprocessing

For training our GLM model, we will have to select the features that yield statistically significant parameters. For this purpose, we have arranged our initially available data for frequencies as shown in section 3.1. in table 4.

Variables such as DriverAge, CarAge or Density will be kept as continuous variables, while the four categorical features (Power, Brand, Gas and Region) will have to be encoded into a dummy encoding, as we will soon discuss. In this subsection, we will be exploring in a graphical representation how influential is each categorical variable in the response, and this will be done by grouping the information in our original tables for frequencies (table 4) and the one for claim amounts (table 5).

The tool for manipulating and ordering data used across our work is the python library *pandas* for data science and analytics[14]. The DataFrame will be the main structure to analyze our data across different scenarios and purposes, and some of the most important

[13] For more information about the statsmodels library and its main configuration and features the reader can access the online official documentation of the different functions, parameters, attributes or performance evaluating methods in: statsmodels v0.13.0.dev0, (2021).

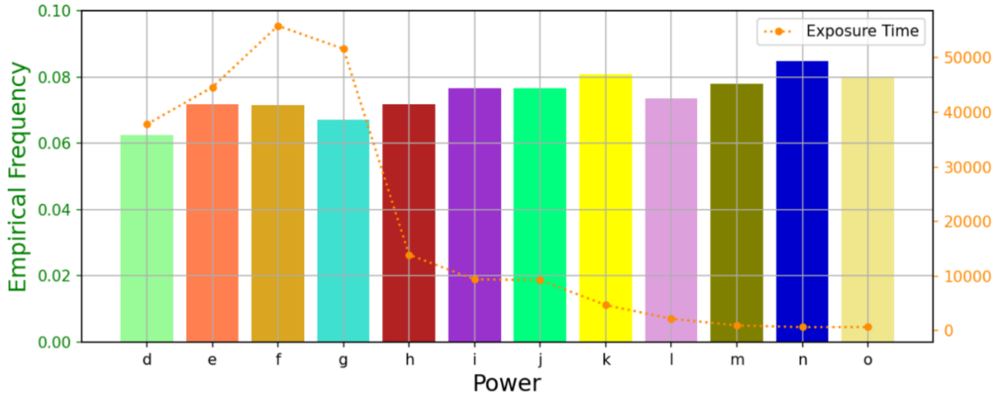
[14] The official documentation (McKinney & Pandas Development Team, 2021) for this library provides all details and insights on each data manipulation function and methodology.

pandas functions used will be mentioned, although for further information on our application and usage for them the reader can refer to our code in Appendix C.

4.1.1.1. Marginal dependency Plots

In figure 11 we can see the result of grouping our frequency table by the Power feature, which has a total of 11 different categorical values, each of them with a different average empirical frequency calculated as the ratio of the sum of ClaimNb (number of claims) on the sum of Exposure, for each of its unique values.

Figure 11: Empirical Frequency for each unique value of Power

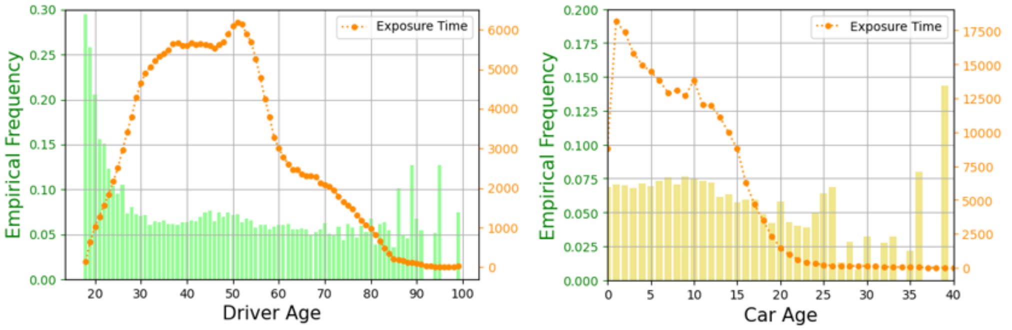


Source: own elaboration

We can see clear differences in empirical frequencies ranging from values of 0.06 to 0.08, and this could be enough for us to understand the influence of each category on the response variable. However, in the plot we also see a dotted orange line indicating the empirical support in favor of those values coming from the amount of exposure years that have been observed for each of the categories.

We will obviously trust more the empirical frequency values that are supported by a large amount of exposure time, and the GLM model will also regard this in the confidence interval and significance estimations.

Figure 12: Empirical frequency for each group of Driver Age (left) and each Car Age (right)

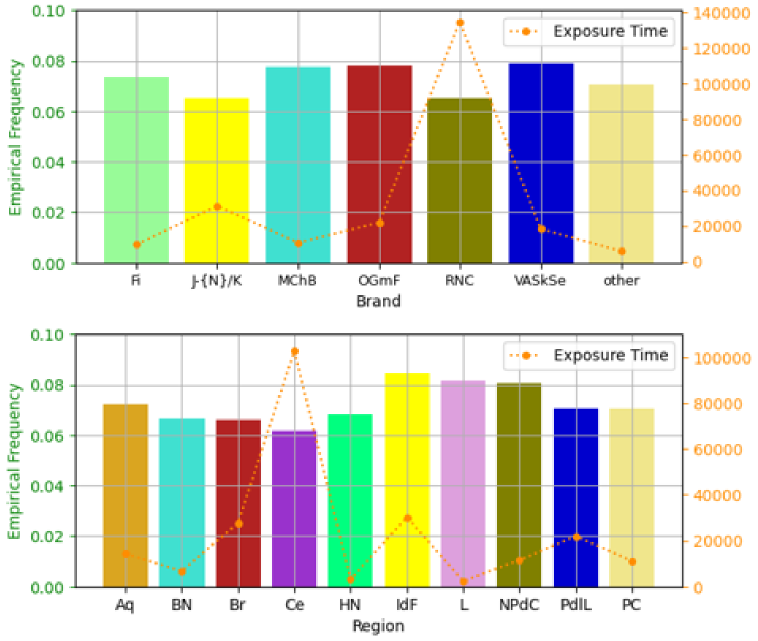


Source: own elaboration

In figure 12 we can observe the evolution of the empirical frequency as Driver Age increases. It is clear that the youngest drivers have a frequency much higher that more mature drivers, although the relationship is not completely monotonic nor perfectly linear. Something similar happens with the age of the car, the main tendency looks to be decreasing, and values beyond 20 become very rare, so a spike around these ages should not be of great concern.

Similar information can be extracted from plots on other variables, particularly, we can see the frequency levels derived from features Brand and Region in figure 13:

Figure 13: Empirical Frequency for Brand (above) and Region (bellow) features



Source: own elaboration

In both of the last depicted categorical features we see one category spanning a much bigger proportion from the total exposure while many other categories have only a residual amount of exposure time to back them. In the case of Brand this category is *Renault*, *Nissan or Citroen*, while for Region it will be the *Centre* region.

An excessive number of categories like the ones we observed in the previous plots can make more difficult the modelling process due to the lack of significance in their parameters. This is why it will be important to consider complementary strategies for processing this type of features in case the significance values in our model are not satisfying.

4.1.1.2. Categorical feature simplification

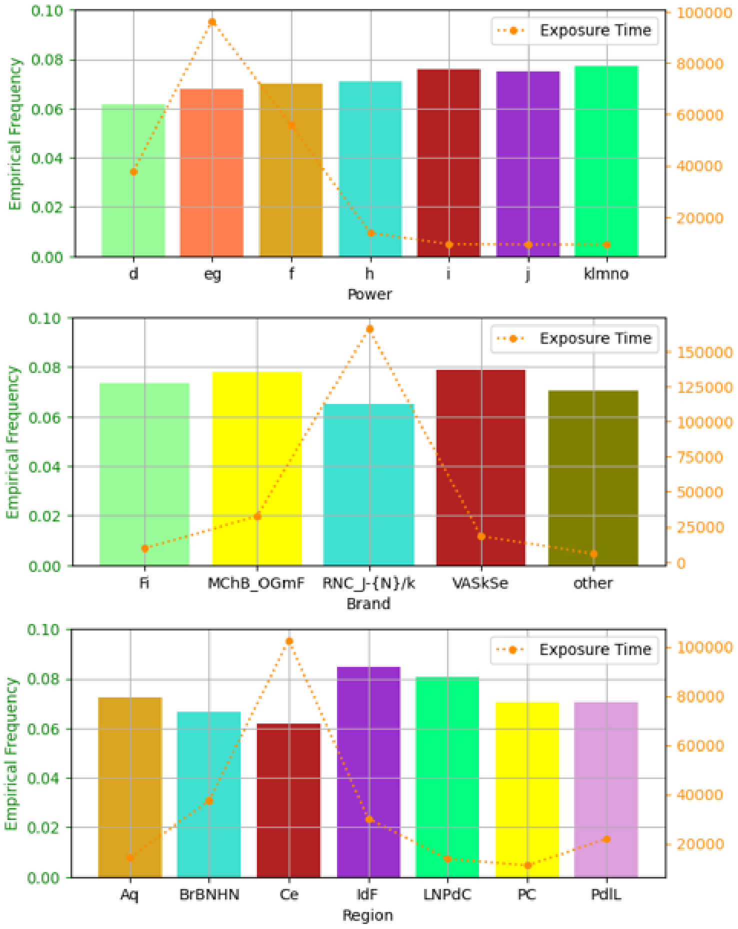
As the data provided from less representative categories tends to be much noisy, we propose a feature engineering technique that seeks to merge some of the less influential categories in a more significant unique categorical value that might have better training properties for our model.

The approach for aggregation can follow not only the least representative values method, but also one focused on merging categories with almost identical impact on the response variable, like values *"e"* and *"f"* in the Power feature as depicted in figure 11. Having the two just mentioned criteria, we can apply a quick string value replacement through *pandas'* data frames and generate a new version of our feature:

- Power: we went from 12 initial categories to 7 final categories by fusing *"e"* and *"g"* values into an *"eg"* value[15], and we also merged the least representative *"k", "l", "m", "n" and "o"* values into a *"klmno"* value.
- Brand: from the original 7 categories we move to 5 by fusing *"RNC"* and "*J-{N}/k*" into *"RNC or J-{N}/k"*, as well as "MChB" and "OGmF" into *"MChB or OGmF"*.
- Region: from the 10 initial categories we ended up with 7 by combining *"Br"*, *"BN"* and *"HN"* into *"BrBNHN"* and also *"L"* and *"NPdC"* into *"LNPdC"*.

This simplification could be carried even further depending on the application, but for our purposes it is more than enough. The new versions of the features where branded as PowerSmpl, BrandSmpl, and RegionSmpl and their marginal dependency plots can the seen in the next figure:

Figure 14: Empirical Frequency PowerSmpl, BrandSmpl and RegionSmpl



Source: own elaboration

Our main train set will now have a higher number of features, but at each training process we will only use a portion of these, given that RegionSmpl might be chosen in replacement of Region and so on.

[15] The reason for merging "e" with "g" instead of "f" is explained in subsection 4.1.2. on Frequency parameter significance.

4.1.1.3. Establishing the frequency train set

By using a function from scikit-learn[16] we perform a split in our current frequency table. The split is done so that 80% of all observed policies end up in the train set, while the rest will be put aside for the test set.

We have also fixed a random seed in order to guarantee the reproducibility of our results. And the *train_test_split* function as a default argument for stratification of each value in ClaimNb. As a result, the train set for frequencies has 330,422 instances while the test set is left with a total of 82,606.

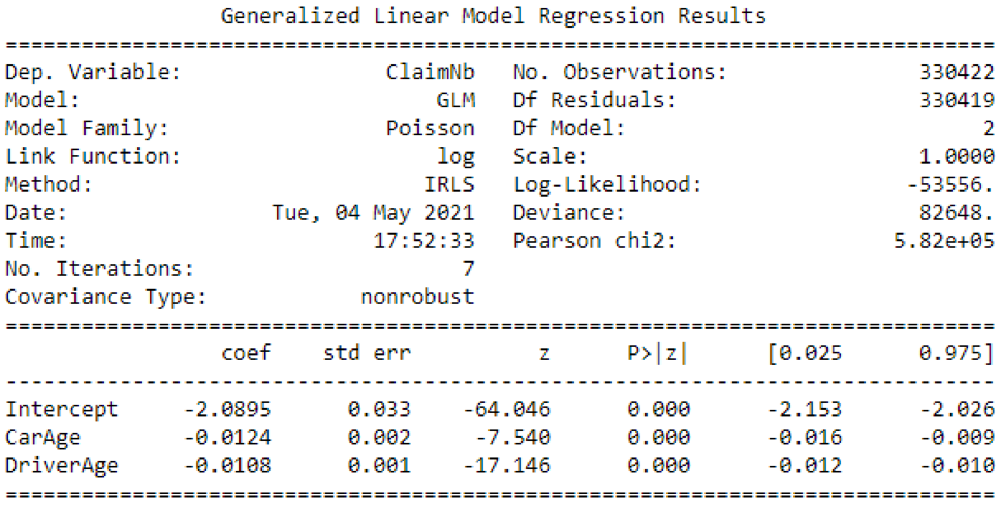
4.1.2. Frequency parameter significance

In this section, we will begin by tying our first GLM for frequencies by using the statsmodels function GLM(…). Specifically, we will be fitting a Poisson regression model with weighted observations. This means that we will be required to provide tree data inputs: the train set containing the features, then the response variable, and finally the sample weights. But out of all the available features, we will begin by selecting just a few, and we will specify this through the following model formula:

$$ClaimNb \sim CarAge + DriverAge$$

The Poisson regression configuration has the log-link function by default, which makes sense given that it is the canonical link. Therefore, after fitting the model we have the next summary table:

Table 8: First Poisson regression with two predictors



```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:               ClaimNb   No. Observations:               330422
Model:                           GLM   Df Residuals:                   330419
Model Family:                Poisson   Df Model:                            2
Link Function:                   log   Scale:                          1.0000
Method:                         IRLS   Log-Likelihood:                -53556.
Date:               Tue, 04 May 2021   Deviance:                       82648.
Time:                       17:52:33   Pearson chi2:                 5.82e+05
No. Iterations:                    7
Covariance Type:           nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -2.0895      0.033    -64.046      0.000      -2.153      -2.026
CarAge        -0.0124      0.002     -7.540      0.000      -0.016      -0.009
DriverAge     -0.0108      0.001    -17.146      0.000      -0.012      -0.010
==============================================================================
```

Source: own elaboration

The information in table 8 provides details on the number of observations, the log-likelihood or the Chi-squared value. The method for fitting is IRLS, meaning *"iteratively reweighted least squares"* and the degrees of freedom are 2, corresponding to the two

[16] To read the proper documentation for the scikit-learn modules and functions the reader can refer to the official web documentation (scikit-learn, 2021).

additional parameters estimated for out predictors. The measure of deviance will be of great importance for us, and in order to calculate it for a test set after making predictions, we have defined our own python function, and we have tested it to obtain the same value that is shown in the GLM summary table after predicting the fitted values of the train set.

Poisson Deviance (16):

$$2\sum_{i=1}^{n}\left(y_i\, ln\frac{y_i}{\widehat{\mu}_i} + (n_i - y_i)\right)$$

with $y_i\, ln\, y_i = 0$ when $y_i = 0$

In the lower section of the table however, we find the information about significance, and for this model we see a strong significance in all parameters, meaning that their information is very reliable and the dependence of ClaimNb with CarAge and DriverAge is well founded.

$$(P > |z|) < 0.05 \qquad (17)$$

$$(P > |z|) < 0.001 \qquad (18)$$

Our main target will be to select a model where all parameters comply with the first of the two significance level barriers just depicted. Though a model where most of parameters comply with the second threshold is also desirable.

4.1.3. Choosing the optimal Frequency GLM

In order to find a model that can extract the maximum amount of information about the relationship between features and response, we tested several combinations of predictor variables[17], adding each at a time and assessing the significance of the coefficients.

In our final model, we have used all 7 features from the frequencies table, but two of the categorical features were used in their simplified version. The feature configuration used is thus the following:

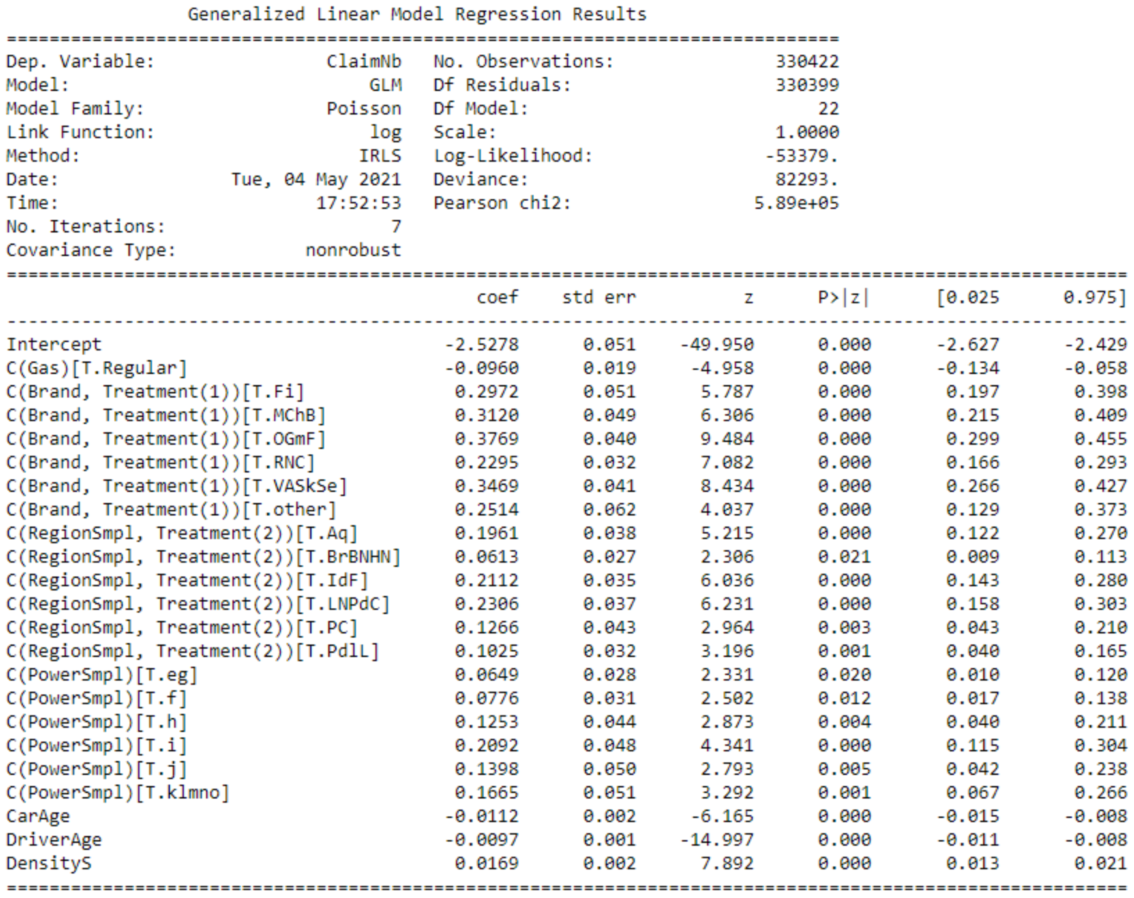
- CarAge: linear variable.
- DriverAge: linear variable.
- Gas: dummy encoded categorical at Diesel as base.
- PowerSmpl: dummy encoded categorical at *d* as base
- Brand: dummy encoded categorical at *J-{N}/K* as base.
- RegionSmpl: dummy encoded categorical at *Ce* as base.
- DensityS: linear variable.

The resulting model fit will be used to predict the test set frequencies and to evaluate the Generalization Error through our Poisson Deviance function. This model will also be further used in the comparison with the Gradient Boosting model, both in terms of

[17] More frequency GLMs can be found in our code, presented in Appendix C.

predictive accuracy and in terms of final premium. The summary of the fit can be seen in the next table:

Table 9: Best Poisson GLM for frequencies



```
                  Generalized Linear Model Regression Results
================================================================================
Dep. Variable:                 ClaimNb   No. Observations:                330422
Model:                             GLM   Df Residuals:                    330399
Model Family:                  Poisson   Df Model:                            22
Link Function:                     log   Scale:                           1.0000
Method:                           IRLS   Log-Likelihood:                 -53379.
Date:                 Tue, 04 May 2021   Deviance:                        82293.
Time:                         17:52:53   Pearson chi2:                  5.89e+05
No. Iterations:                      7
Covariance Type:             nonrobust
==========================================================================================================
                                             coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------------------------------
Intercept                                 -2.5278      0.051    -49.950      0.000      -2.627      -2.429
C(Gas)[T.Regular]                         -0.0960      0.019     -4.958      0.000      -0.134      -0.058
C(Brand, Treatment(1))[T.Fi]               0.2972      0.051      5.787      0.000       0.197       0.398
C(Brand, Treatment(1))[T.MChB]             0.3120      0.049      6.306      0.000       0.215       0.409
C(Brand, Treatment(1))[T.OGmF]             0.3769      0.040      9.484      0.000       0.299       0.455
C(Brand, Treatment(1))[T.RNC]              0.2295      0.032      7.082      0.000       0.166       0.293
C(Brand, Treatment(1))[T.VASkSe]           0.3469      0.041      8.434      0.000       0.266       0.427
C(Brand, Treatment(1))[T.other]            0.2514      0.062      4.037      0.000       0.129       0.373
C(RegionSmpl, Treatment(2))[T.Aq]          0.1961      0.038      5.215      0.000       0.122       0.270
C(RegionSmpl, Treatment(2))[T.BrBNHN]      0.0613      0.027      2.306      0.021       0.009       0.113
C(RegionSmpl, Treatment(2))[T.IdF]         0.2112      0.035      6.036      0.000       0.143       0.280
C(RegionSmpl, Treatment(2))[T.LNPdC]       0.2306      0.037      6.231      0.000       0.158       0.303
C(RegionSmpl, Treatment(2))[T.PC]          0.1266      0.043      2.964      0.003       0.043       0.210
C(RegionSmpl, Treatment(2))[T.PdlL]        0.1025      0.032      3.196      0.001       0.040       0.165
C(PowerSmpl)[T.eg]                         0.0649      0.028      2.331      0.020       0.010       0.120
C(PowerSmpl)[T.f]                          0.0776      0.031      2.502      0.012       0.017       0.138
C(PowerSmpl)[T.h]                          0.1253      0.044      2.873      0.004       0.040       0.211
C(PowerSmpl)[T.i]                          0.2092      0.048      4.341      0.000       0.115       0.304
C(PowerSmpl)[T.j]                          0.1398      0.050      2.793      0.005       0.042       0.238
C(PowerSmpl)[T.klmno]                      0.1665      0.051      3.292      0.001       0.067       0.266
CarAge                                    -0.0112      0.002     -6.165      0.000      -0.015      -0.008
DriverAge                                 -0.0097      0.001    -14.997      0.000      -0.011      -0.008
DensityS                                   0.0169      0.002      7.892      0.000       0.013       0.021
==========================================================================================================
```

Source: own elaboration

We see that our main goal was accomplished in this model training. The majority of the coefficients are strongly significant ($P > |z| = 0.000$) and only a few of them have a weak statistical significance, like Power *“eg”* or Region *“BrBNHN”*.

The model is saved and stored for the performance analysis in Chapter 5.

4.2. GLM fitting for claim amounts

Coming back from the conclusions of section 3.2.4. we are now aware of the impact of extreme values in claim amounts modelling and we thus decide to proceed with the following strategy:

- We filter our claims database by taking out outliers, which were set to be all claim amounts above 10,000€ in order to maintain the modeling process stable both for the current GLM as well as for the GBM model tested further on.
- In total, 275 claim amounts where dropped, representing 1.70% of the initial dataset. All claims coming from real and verified compensations of high magnitude can be modelled separately based on an extreme-value-theory

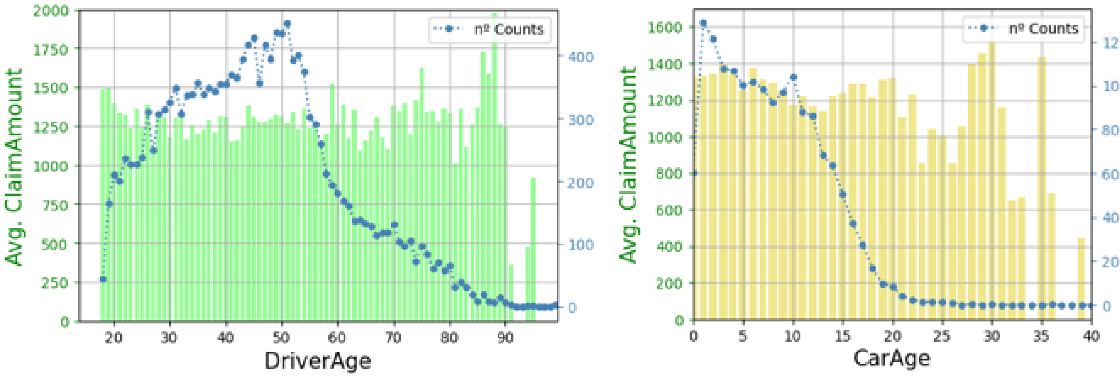
setting where reinsurance can be involved, working as an additional loading to the pure premium that will be calculated in our work.

- Additionally, and in order to keep the matching integrity between datasets, the frequency table was preprocessed before the training of our first model for frequencies[18].
- Finally, we are left with a total of 15,906 claims, and after applying the filter discussed in section 3.2.2. we end up with 15,900 claims ready for modelling.

4.2.1. Feature selection for claim amounts

The claim amounts also have marginal dependencies across features, and here we will depict some of them. First on figure 15 we have the plots for DriverAge and CarAge, and here we can already see some differences with their equivalent for frequencies in figure 12.

Figure 15: Average Claim Amount for each Driver Age (left) and each Car Age (right)



Source: own elaboration

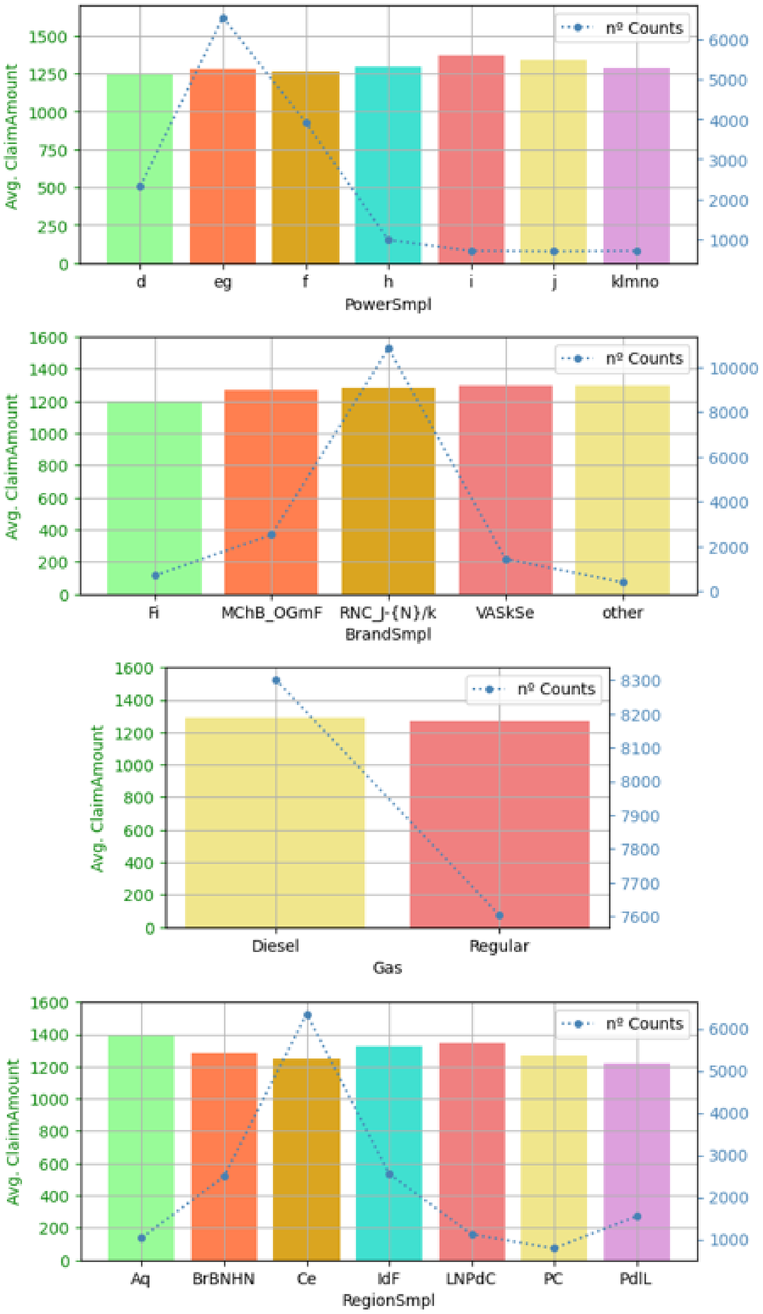
From both figures it is already evident that in features like DriverAge the linear tendency is much less evident, and the higher value for bars between ages 18, 24 is less evident and is also supported by less data, and the same happens for elderly ages. In the CarAge feature the trend is more reliable, although for both variables it will be important to contrast this with the significance values of our trained GLMs.

For categorical features, the dependencies will be even less evident when plotting, in figure 16 we will check the marginal dependencies for Power, Brand and Region, both of them already in their simplified versions.

The relative differences for many of the categories throughout PowerSmpl is very small, and this can already give us an idea of the incoming hardships for finding significance across their parameters. In the case of BrandSmpl and Gas we find a similar scenario, and in the case of RegionSmpl we have some more noticeable differences. However, in the end it will be the model training that will decide the significance of their parameters.

[18] The method for taking out these frequencies was based on pandas data frame sub-setting and indexing, for further details on the correction the reader can refer to our code in Appendix C, close to the begging in the “Claim Amounts Preprocessing” section.

Figure 16: Average Claim Amount for Categorical features on simplified versions



Source: own elaboration

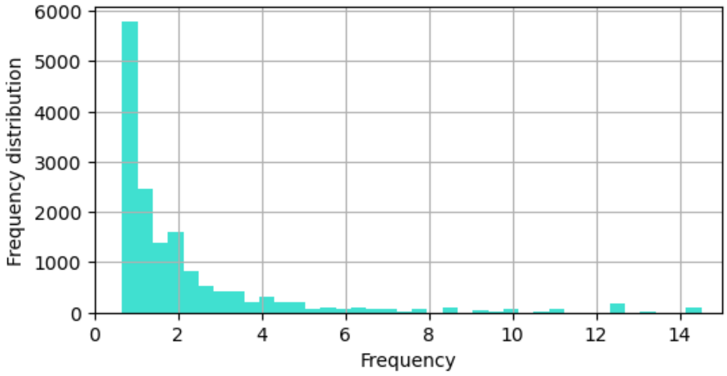
It would be possible to further simplify the categories, into an even lower number of unique values, but there is risk of information loss in this approach, so we settle for testing the significance of our different available feature versions.

4.2.1.1. The Frequency feature as a predictor

As we were pointing out in our Case Study chapter, the claims table will use an additional feature to predict the average amount. This feature will try to reuse the information from ClaimNb and Exposure, in a way to depict how ‘fast’ the claims were reported.

In a way, the Frequency feature explains how the amount of the claim evolves for profiles that report more claims in a shorter period of time. It will be a continuous variable with positive real values, and its distribution can be visualized through the histogram plot in figure 17.

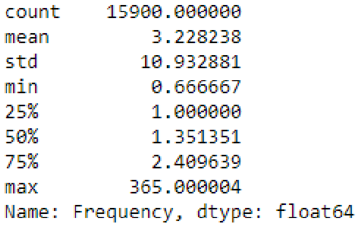
Figure 17: Frequency feature histogram



Source: own elaboration

There is a clear concentration around values of 1, due to those policies observed though a full year and having reported just one claim, but then we progressively see policies that reported one claim in less time or even reported 2 or more claims. The distribution is highly skewed as it is to be expected when doing a ratio on very low exposures.

Table 10: Summary statistics for the Frequency feature



```
count    15900.000000
mean         3.228238
std         10.932881
min          0.666667
25%          1.000000
50%          1.351351
75%          2.409639
max        365.000004
Name: Frequency, dtype: float64
```

Source: own elaboration

In the basic statistics from table 10 we see the effects of the severe skewness in a mean value of 3.228 against a median value of 1.351. This can be easily explained by the presence of values like the maximum of 365, and this is obviously caused by a policy that was observed for just 1 day and filed one claim[19].

This feature will be carefully assessed on the results evaluation, because there is a notorious bias when generalizing to policies that have no reported a claim yet, and to which we will need to establish a pure premium.

4.2.1.2. Establishing the claim amounts train set

For splitting the available Claims table, we will use the indexed values of PolicyID from the splitting already performed in the Freqs (frequencies) tables. The idea behind

[19] Though the presence of such outliers might worry many analysts, in our model fittings we have observed that these values do not have a negative impact on the modeling process and the significance of the parameters. Therefore, no feature preprocessing was performed for this matter.

this is to make sure that all claims falling into the Freqs train set are exactly matched in the Claims train set, and the same for the test sets.

In total, we will have 12,653 claim amounts in our train set, and 3,247 amounts for testing. The available data for building the model is thus much less than what we had for frequencies, and even though it is still a substantial number of observed claims, the implications for significance will be inevitable.

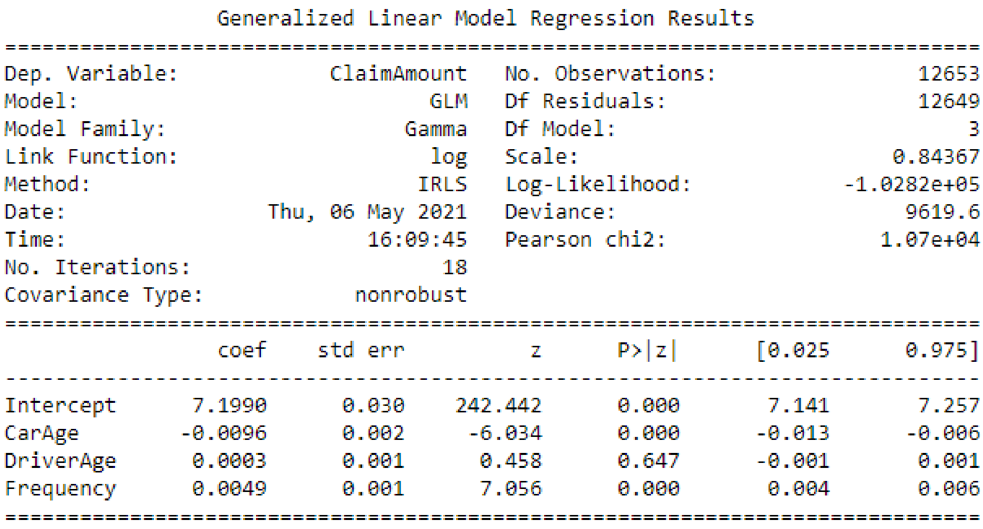
4.2.2. Claim Amount Parameter significance

The first GLM for claim amounts will have just the few basic predictors, and it will be built under a Gamma regression fitting, using the Gamma deviance as a loss function. Particularly, the initial formula will be the following:

$$ClaimAmount \sim CarAge + DriverAge + Frequency$$

We train the first model and inspect the summary information in the table shown below.

Table 11: First GLM for Claim Amounts summary table



```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:            ClaimAmount   No. Observations:                12653
Model:                            GLM   Df Residuals:                    12649
Model Family:                   Gamma   Df Model:                            3
Link Function:                    log   Scale:                         0.84367
Method:                          IRLS   Log-Likelihood:            -1.0282e+05
Date:                Thu, 06 May 2021   Deviance:                       9619.6
Time:                        16:09:45   Pearson chi2:                 1.07e+04
No. Iterations:                    18
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      7.1990      0.030    242.442      0.000       7.141       7.257
CarAge        -0.0096      0.002     -6.034      0.000      -0.013      -0.006
DriverAge      0.0003      0.001      0.458      0.647      -0.001       0.001
Frequency      0.0049      0.001      7.056      0.000       0.004       0.006
==============================================================================
```

Source: own elaboration

Now the information from the marginal plots in the previous figure 15 is corroborated. We see that despite the solid significance of the CarAge feature, the feature DriverAge is not significant. The additional feature Frequency has high significance as well, and its relativity will be > 1, given that the coefficient has a positive value.

From this point, each of the other available features were introduced, one at a time. If the coefficients were not statistically significant then the variable was discarded from the model. The significance criteria will be the same as for the frequencies GLM.

4.2.2.1. Discretization for continuous features through bucketing

Additionally, we arranged a version from DriverAge where we bucketed the values if this feature in order to obtain a categorical value called DriverAgeB by the next criteria (19):

$$If\quad 18 \leq DriverAge \leq 23 \quad then \quad DriverAgeB = \text{'}18 - 23\text{'}$$

$$If\quad 24 \leq DriverAge \leq 69 \quad then \quad DriverAgeB = \text{'}24 - 69\text{'}$$

$$If\quad DriverAge > 70 \quad then \quad DriverAgeB = \text{'} + 70\text{'}$$

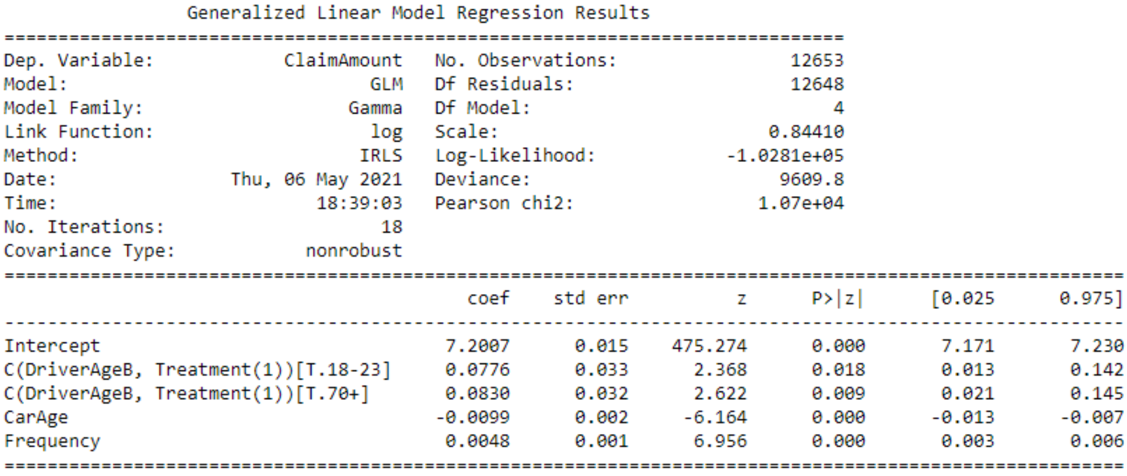
4.2.3. Choosing the optimal GLM for claim amounts

We give a brief summary of the variables that were not used in the final model due to their lack of statistical significance:

- Categorical features in their initial versions: Brand, Gas, Power and Region.
- Categorical simplified features: BrandSmpl, PowerSmpl and RegionSmpl.
- Continuous features: DensityS

After fitting the GLM with a total of 6 features, we obtain the information represented in the summary table:

Table 12: Best Gamma GLM for claim amounts



```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:            ClaimAmount   No. Observations:                12653
Model:                            GLM   Df Residuals:                    12648
Model Family:                   Gamma   Df Model:                            4
Link Function:                    log   Scale:                         0.84410
Method:                          IRLS   Log-Likelihood:            -1.0281e+05
Date:                Thu, 06 May 2021   Deviance:                       9609.8
Time:                        18:39:03   Pearson chi2:                 1.07e+04
No. Iterations:                    18
Covariance Type:            nonrobust
=========================================================================================================
                                            coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------------------------
Intercept                                 7.2007      0.015    475.274      0.000       7.171       7.230
C(DriverAgeB, Treatment(1))[T.18-23]      0.0776      0.033      2.368      0.018       0.013       0.142
C(DriverAgeB, Treatment(1))[T.70+]        0.0830      0.032      2.622      0.009       0.021       0.145
CarAge                                   -0.0099      0.002     -6.164      0.000      -0.013      -0.007
Frequency                                 0.0048      0.001      6.956      0.000       0.003       0.006
=========================================================================================================
```

Source: own elaboration

In this model we have an acceptable statistical significance for our coefficients, even when DriverAgeB ’18-13’ and ‘70+’ come closer to our first threshold. However, we did not succeed at including more predictors, due to the lack in significance, so this rather simple model will be our way of interpreting and predicting the claim amounts. The formula for the Gamma deviance will be the following (20):

$$2\sum_{i=1}^{n}\left(-\ln\frac{y_i}{\widehat{\mu_i}}+\frac{y_i-\widehat{\mu_i}}{\widehat{\mu_i}}\right)$$

4.3. GLM pricing machine

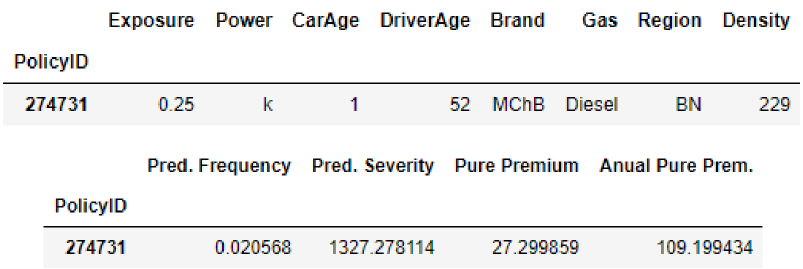
Once we have both of our selected GLMs we can apply them onto new unseen policies and establish a pure premium by following the basic formula (21):

$$Pure\ premium = E[frequency] \times E[severity]$$

And the policies to price will be those from the frequency test set, given that it is a fairly large amount of instances, with over 82 thousand PolicyID numbers, we can assess the performance of an aggregated model in a big-data environment typical of insurance companies.

By using the specific method in statsmodels for fitted GLMs, we can perform both of our predictions of interest. Here is a random profile selected from the frequency test set table:

Table 13: Single profile Pure Premium prediction



PolicyID	Exposure	Power	CarAge	DriverAge	Brand	Gas	Region	Density
274731	0.25	k	1	52	MChB	Diesel	BN	229

PolicyID	Pred. Frequency	Pred. Severity	Pure Premium	Anual Pure Prem.
274731	0.020568	1327.278114	27.299859	109.199434

Source: own elaboration

The profile just depicted was observed for just 3 months, yet we can use the predictions from the GLMs to establish the predicted frequency for that exposure period and the expected claim amount. However, in practice, we will be renewing or subscribing premiums on a yearly basis, so it is not inappropriate to calculate our premium assuming an Exposure = 1 value. This is why we calculate the Annual Pure Premium in the last Column, and it is as simple as taking the ratio between the Pure Premium and the Exposure values, this way we can 'normalize' the predictions so that all of our test policies are projected for a one-year period.

4.3.1. The Pure Premium Analysis table

Once we have all predictions of frequencies, claim amounts, and pure premiums for our test set, we will need to effectively summarize the information in a way that we can contrast the predicted values from the observed ones.

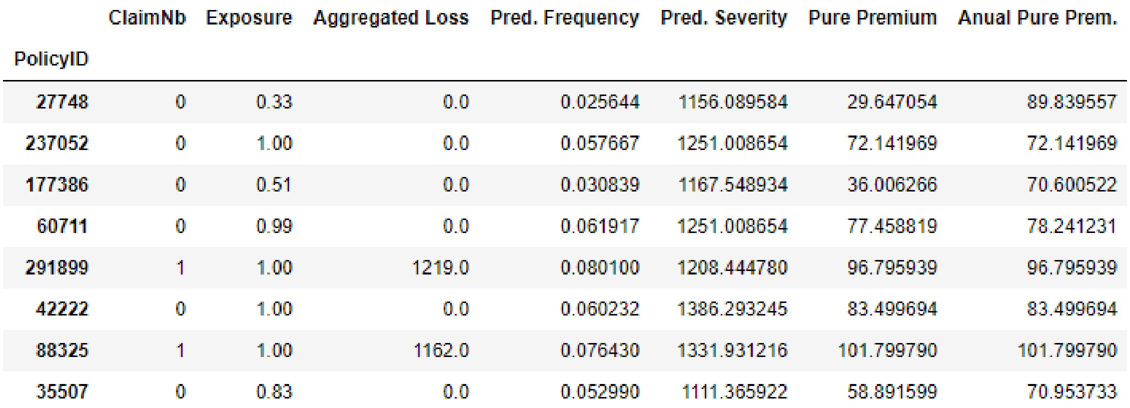
We will do this by organizing the available columns in the following two groups:

- Observed information: *ClaimNb*, *Exposure* and *Aggregated Loss*
  - The Aggregated Loss column is obtained by matching in a backwards fashion the information of claim amounts into the frequency tables, and this is done by summing up all claims coming from each of the policies, both for all PolicyID values with no reported claims, and for PolicyID values with several reported claims.
- Predicted information: *Pred. Frequency*, *Pred. Severity*, *Pure Premium* and *Annual Pure Prem.*
  - The Annual Pure Prem. columns is only here for visual purposes, because given that the empirical information is conditioned to the Exposure time of each policy, when comparing the total premiums

coming from our GLMs predictions with the aggregated losses we will need to do so on the same Exposure-scaling.

After arranging the information, we can see some of its entries in the table displayed bellow:

Table 14: Premium Analysis Table for GLMs



PolicyID	ClaimNb	Exposure	Aggregated Loss	Pred. Frequency	Pred. Severity	Pure Premium	Anual Pure Prem.
27748	0	0.33	0.0	0.025644	1156.089584	29.647054	89.839557
237052	0	1.00	0.0	0.057667	1251.008654	72.141969	72.141969
177386	0	0.51	0.0	0.030839	1167.548934	36.006266	70.600522
60711	0	0.99	0.0	0.061917	1251.008654	77.458819	78.241231
291899	1	1.00	1219.0	0.080100	1208.444780	96.795939	96.795939
42222	0	1.00	0.0	0.060232	1386.293245	83.499694	83.499694
88325	1	1.00	1162.0	0.076430	1331.931216	101.799790	101.799790
35507	0	0.83	0.0	0.052990	1111.365922	58.891599	70.953733

Source: own elaboration

This table will be revisited in Chapter 5 on the section of Comparative Premium Analysis along with an equivalent table for GBM models.

Right now we have established all the methodological steps for the GLM part of our case study, and we can move forward with the Gradient Boosting models in a similar structure but by pointing out some of the most important difference between the two modelling approaches.

4.4. GBM model fitting for frequencies

As we have already anticipated, we will be using the Gradient Boosting implementation from LightGBM, a framework originally developed by Microsoft. Before moving to the training criteria for our first GBRT model we will review some of the most distinctive features in this implementation through its official documentation.

4.4.1. Distinctive technical aspects of the LightGBM implementation

As stated in the documentation (Microsoft & LightGBM Contributors, 2021) the main advantages of this Gradient Boosting framework are the following:

- *"Faster training speed and higher efficiency"*.
- *"Lower memory usage.*
- *"Better accuracy"*.
- *"Support of parallel, distributed, and GPU learning"*.
- *"Capable of handling large-scale data"*.

One of the keys to speed and efficiency in terms of memory usage of LightGBM is, according to the *Features* section in the documentation, that: *"LightGBM uses histogram-based algorithms (...), which bucket continuous feature (attribute) values into discrete bins. This speeds up training and reduces memory usage"*.

Another key operating feature is the "*Leaf-wise (Best-first) Tree Growth*" and it is described as splitting the leaf that provides the maximum gain, weather it is in terms of impurity or in terms of deviance reduction. This contrasts with the most common optimization procedure based on a level-wise growth, the splits are evaluated or performed following the order based on the current depth level.

It is also worth mentioning the treatment of categorical features, which are processed directly instead of first applying the already common one-hot-encoding, described by the documentation as *"suboptimal for tree learners"*.

After considering the previous particularity, we proceed to describe the encoding method we will use for our categorical variables.

4.4.2. Response-based encoding for categorical features

In Machine Learning, the different algorithmic applications and case studies require to adapt the information embedded in features through feature engineering, so that the ML algorithms can process them correctly and learn the necessary trends concerning the response variable.

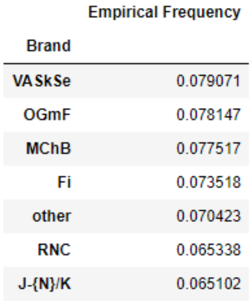
Some of the most common encoding methods are the following:

- Dummy encoding: already used in our GLM training process
- One-hot-encoding: similar to the previous but with no base or reference category, with each possible value being assigned with its own binary column.
- Ordinal encoding: each unique value is assigned to an integer value, this way we could end up transforming a string variable into a numeric, as long as we keep the reference of the label corresponding to each integer.

These are just some of the existing encoding schemes, but our proposal will actually elaborate further on the Ordinal encoding. In fact, we will assign each integer value for each category based on the average response (y).

To understand the fundamental idea of Response-based encoding we will use the summary data from our Brand feature in table 15 by sorting the column *'Empirical Frequency'* in descending order. The category with the highest empirical frequency is *'VASkSe'* corresponding to Volkswagen, Audi, Skoda or Seat. The next categories have a progressively lower average response, similar to a monotonic downward trend.

Table 15: Categorical Response-based encoding based on feature Brand



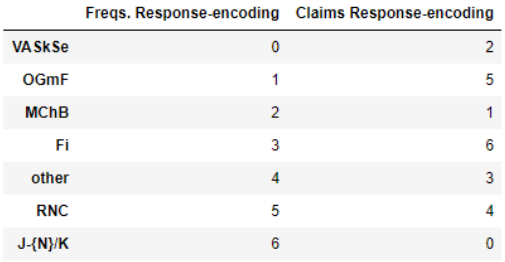
	Empirical Frequency
Brand	
VASkSe	0.079071
OGmF	0.078147
MChB	0.077517
Fi	0.073518
other	0.070423
RNC	0.065338
J-{N}/K	0.065102

Source: own elaboration

In order to assign each integer, starting from 0 for the highest response category, 1 for the second and so on; we used python dictionaries as the data structure to match the labels to these values, and we did so with both responses: *'Empirical Frequency'* and *'ClaimAmount'*.

The next table shows the Brand categories, with the values as integers that were assigned to replace them:

Table 16: Brand Response-encoding for frequencies and for claim amounts



	Freqs. Response-encoding	Claims Response-encoding
VASkSe	0	2
OGmF	1	5
MChB	2	1
Fi	3	6
other	4	3
RNC	5	4
J-{N}/K	6	0

Source: own elaboration

It is very important to know that the order does not have to be the same for different response variables, as we can see in this example. This means that we can only use each Response-encoded version of the feature exclusively with that response, otherwise the monotonicity cannot be guaranteed, and the train process can be compromised.

In our case study, we have performed and used in training the following categorical Response-encodings:

- For Frequencies:
  - Initial categorical features: *Gas*, *Power*, *Brand* and *Region* become *GasT*, *PowerT*, *BrandT* and *RegionT* (T stands for 'target').
- For Claims:
  - Simplified categorical features: *PowerSmpl*, *BrandSmpl* and *RegionSmpl* become *PowerSmplT2*, *BrandSmplT2*, *RegionSmplT2*.

In the LightGBM setting this type of encoding helps training because with basic Ordinal encoding, each tree will need a higher *max. depth* hyperparameter in order to fully capture the non-monotonic evolution of the response across categories. With Response-based encoding, a GBM with *max. depth = 1* will be able to fully capture the implied trend.

4.4.3. Basic GBM model fitting for frequencies

Through the function LGBMRegressor(…), from the LightGBM package in python, the first proposed model fit will be trained over the same 330,422 instances that we splitted for the train set in sub-section 4.1.1.3. for our frequency GLM. We maintain the same data set partition to guarantee the homogeneity and comparability between all model fittings.

The last-mentioned function is one adapted on a scikit-learn API, and this will prove to be useful for tools like Partial Dependency Plots (PDP). This first training will isolate the following columns from the frequency table:

- Response variable (y): column *'ClaimNb' / 'Exposure'*.
- Predictor variables/features $(\boldsymbol{x})$: *'PowerT', 'CarAge', 'DriverAge', 'BrandT', 'GasT', 'RegionT', 'DensityS'.*
- Sample weights (w): *'Exposure'*.

The parameter/hyperparameter configuration we will use for this model, is detailed bellow:

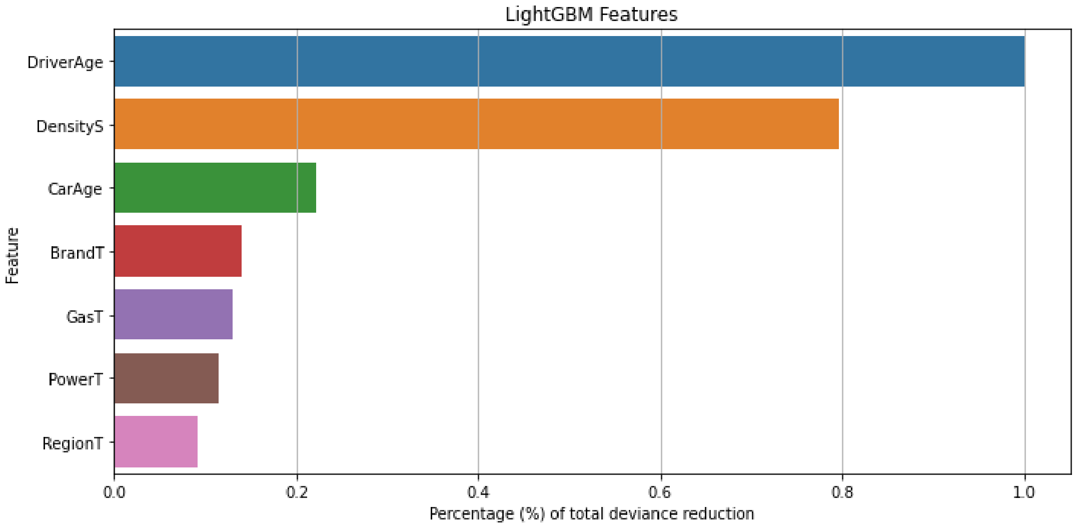
- $Max.depth = 4$
- $Learning\ rate = 0.05$
- $nº\ of\ estimators\ (trees) = 200$
- $Objective = \text{poisson}$
- $Min.child\ samples = 500$

In the documentation for LightGBM we can find details for many other parameters and hyperparameters. But we will depart from this setting and improve upon it in section 4.4.4. Hyperparameter-tunning for frequencies.

4.4.3.1. Results of the first GBM fitting:

The total runtime for the model based on the previously shown configuration was 2.77 seconds[20]. Now that the model is built, we can consult some of its results through basic attributes like the features importance.

Figure 18: Feature Importance for each predictor



Source: own elaboration

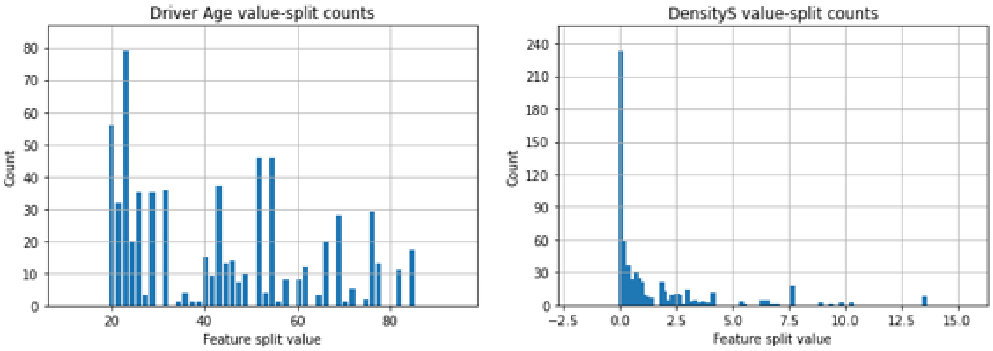
[20] Even with the same random seed applied to the model fitting method in the LGBMRegressor function, the time varies slightly on each run. The time obtained can be generated through our code in the GBRT for Frequencies section (See Appendix C).

In figure 18 we have the Feature Importance in relative terms, which is calculated by taking all deviance reductions (gains) across all trees (estimators) for each of the predictors. This way, the feature with the higher gains will be set to one, and the following features will be a proportion of it. Although relativization schemes can be modified into each feature importance as a proportion of all gains instead of those of just the most important feature.

The feature that provided the highest Poisson Deviance reductions on the train samples across all 200 trees was the Driver Age, followed closely by population Density. Car Age ended in third place, though it falls back significantly from the first two. The rest of the features have less impact on feature reduction, yet they are still not negligible.

There are feature selection procedures that could eliminate the less important features for a higher parsimony in the model. However, we are primarily based on the significance for the variables already tested in our GLM models, meaning that if these variables already provided significant results in our optimal frequency GLM, then they do hold valuable information about the response even if it is at a smaller scale.

Figure 19: Value split-counts for DriverAge (left) and DensityS (right)

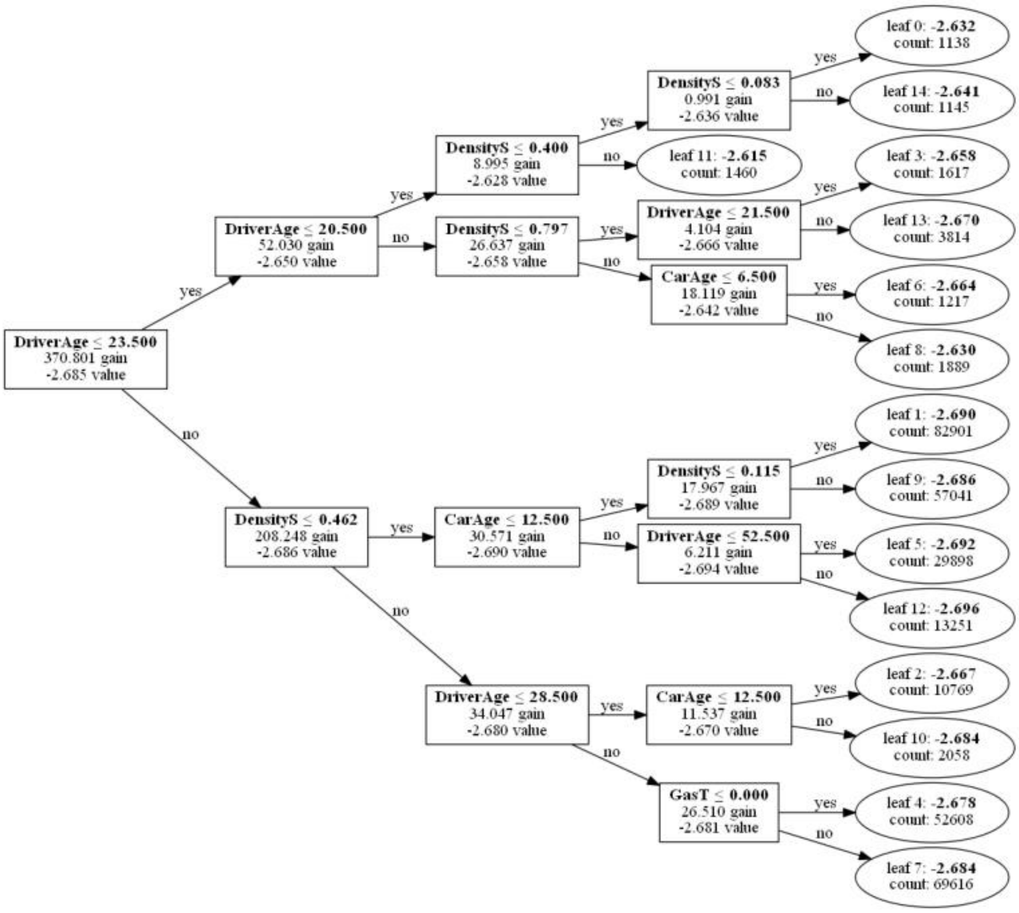


Source: own elaboration

There is additional information about the splits performed from each feature, and two examples are depicted in figure 19, particularly for our most important predictors as of figure 18. These plots show the cut-values that were used from the two most important features across all trees. We know from the Decision Tree literature that at each node, the best split from the respective feature will be selected, but now we can keep track of the splits, and we can know the part of the distribution of each feature that concentrates the majority of splits, meaning that there lies more information about the relationship of the feature with the response around those points.

Finally, we can access the structure of each fitted tree from our model, this is very convenient in order to have a visual interpretation of the core elements of the model.

Figure 20: First Decision Tree of the GBM model



Source: own elaboration

The tree in figure 20 represents the first of the 200 Decision Trees fitted in the model, and as the first tree, it will launch a first estimation if the average response 'ClaimNb' but given the log-link function, this relationship will not be 1 to 1. We remember the basic EDM equivalence for this scenario (21):

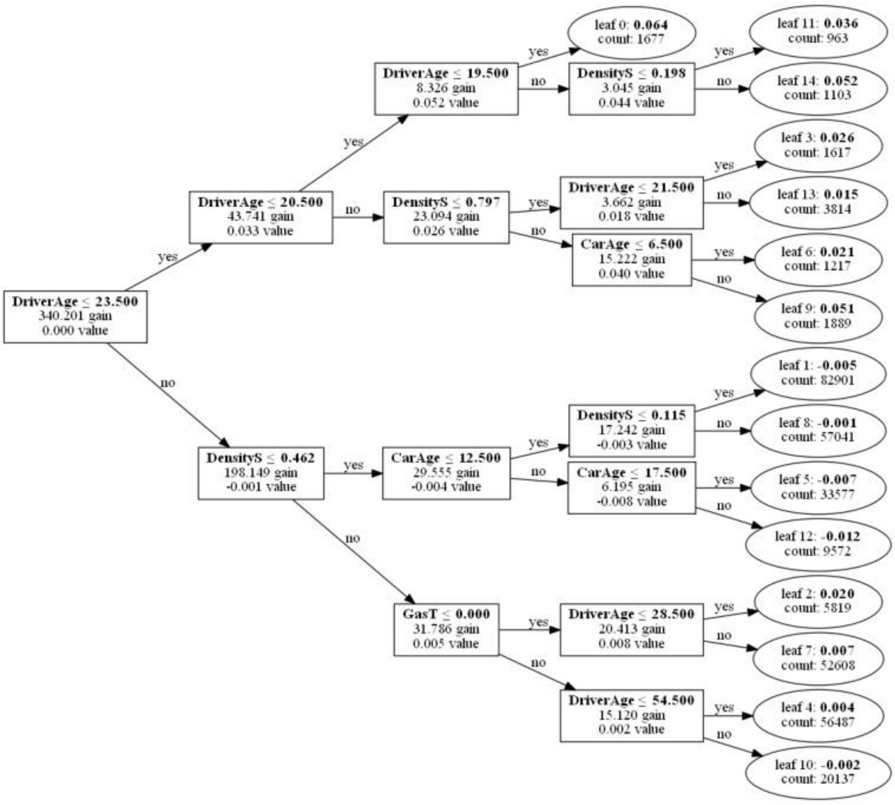
$$\hat{\mu}(x) = e^{score}$$

The term *'score'* refers to the additive process of boosting, as we described in the algorithm in section 2.4.3. and it includes the parameters of all trees in their partitioning process. Therefore, the output values in each leaf from the depicted tree will be used to compute the pseudo-residuals, and these will be successively corrected by the next trees.

The next fitted tree shows the beginning of this process, which is the boosting process itself. Its structure can be seen in figure 21 and the values in the leaves are the corrections over the outputs obtained in the first tree.

This process will continue until reaching the maximum number of estimators. And even with the current couple of trees, we can clearly see that DriverAge and DensityS are leading the majority of the splits.

Figure 21: Second Decision Tree of the GBM model



Source: own elaboration

4.4.4. Hyperparameter-tunning for frequencies

Now, it is important that we improve the current model by choosing the best set of hyperparameters, and we will do so by applying a grid search upon different values of the next basic hyperparameters:

- Learning rate: [0.05, 0.02, 0.01, 0.005]
- Nº estimators: [200, 500, 700, 1000]
- Max. depth: [4, 5, 6]

The resulting grid will have a total of $4 \times 4 \times 3 = 48$ models, and we will perform an exhaustive search to find the best out of all these options. The term ‘exhaustive’ means that we will evaluate the GE from a validation set in each of the 48 combinations, and our goal will be to determine the model that generalizes better to unseen instances. This is done through the Cross-Validation (CV) evaluation approach.

In the next subsection we will quicky describe this procedure, and our customized tool for performing it in python code in order to make it compatible with the LightGBM train and fit function.

4.4.4.1. Cross-Validation

Cross-Validation is commonly used in the Machine Learning environment to assess the predictive capabilities of a model without using yet the samples of the test set. This is done because, as a common best practice, the test set should never be used to arrive at a particular model configuration or a combination of hyperparameters.

This is because otherwise, our model will still be 'learning' in some way from those unseen instances, adapting itself to them even if it is with a minor influence than in fitting. Thus, a separate set is used for validation of the new fitted model, and it is generally taken from a fraction of the train set.

Cross-Validation does exactly that but by splitting the original train set into K 'folds' or buckets of instances, and so the validation process is repeated in K iterations where each of the folds will be used to validate the GE while the rest will be used as the train folds for fitting. This process ensures that all instances of the original train set are eventually used for fitting, and for evaluating the validation GE.

We show bellow the basic structure of our CV algorithm:

Algorithm 3:

- Step 1: Split the training set into K-folds of equivalent size:

$$\mathcal{D} = \{(y_i, x_i); i = 1, \dots, N\} = \sum_{j=1}^{K} \mathcal{D}_{\Theta_j}$$

- Step 2: **for** ***j = 1 to K***:

  I. Establish D_{Θ_j} as validation set and $D_{\overline{\Theta_j}}$ as train set.

  II. Initiate and fit GBM model on $\mathrm{D}_{\overline{\Theta_j}}$ with specific hyperparameters (see Algorithm 2).

  III. Use the fitted model to predict the response based on $\mathcal{D}_{\Theta_j}(\boldsymbol{x})$.

  IV. Compare actual validation responses $\mathcal{D}_{\Theta_j}(y)$ with the obtained predictions through the evaluation measure (Deviance) to obtain the Generalization Error: $\widehat{Err_j}(\hat{\mu})$.

  V. Store the result for this iteration.

- Step 3: Compute the average validation GE:

$$\widehat{Err_{val}}(\hat{\mu}) = \frac{1}{K}\sum_{j=1}^{K} \widehat{Err_j}(\hat{\mu})$$

This algorithm will be performed for all hyperparameter combinations, and in each of them we will register not only the evaluation GE, but also the runtime, in order to make sure the computational efficiency matches a minimum requirement. This means that if the grid search gives us a model with the best accuracy, but at the price of drastically increasing the runtime, then we will choose the next model in case it has a substantially lower runtime.

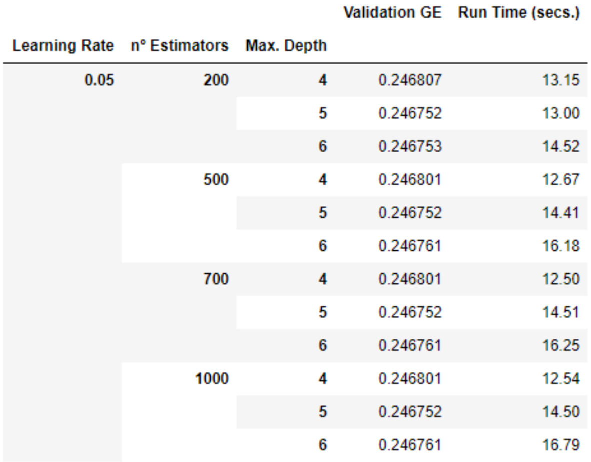
Additionally, in step 2.II. we have an additional parameter in the fitting process, this is the Early Stopping rounds. Its purpose lies under the fact that even if we could add more trees to our boosting algorithm, this might no longer improve the generalization capabilities of our current model. So, if after a certain number of iterations we no longer have an improvement in generalization, the model will stop adding any more trees. This number is the value for the hyperparameter, and in our case it is set to a value of 20.

The Early Stopping rounds are part of the CV process, and it helps to increase efficiency and to save runtime.

4.4.4.2. Grid search for optimal GBM model for frequencies

The Grid search process took 25.01 minutes to complete in a machine with processor Intel(R) Core(TM) i7-6500U CPU and RAM of 8GB. All results were stored into a summary data frame. The first 12 runs are displayed in the next table:

Table 17: Summary table for results of Grid Search

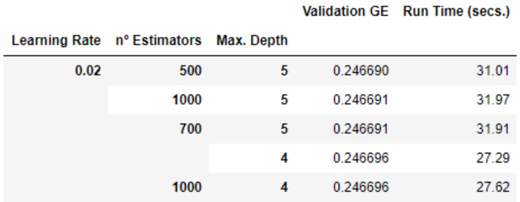


			Validation GE	Run Time (secs.)
Learning Rate	n° Estimators	Max. Depth		
0.05	200	4	0.246807	13.15
		5	0.246752	13.00
		6	0.246753	14.52
	500	4	0.246801	12.67
		5	0.246752	14.41
		6	0.246761	16.18
	700	4	0.246801	12.50
		5	0.246752	14.51
		6	0.246761	16.25
	1000	4	0.246801	12.54
		5	0.246752	14.50
		6	0.246761	16.79

Source: own elaboration

We used the multi-index capabilities of *pandas* data frames to correctly order and organize the information on GE values and runtimes. And if we sort the information by the column *'Validation GE'* in ascending order we get the information in table 18.

Table 18: Best performing models from the Grid Search



			Validation GE	Run Time (secs.)
Learning Rate	n° Estimators	Max. Depth		
0.02	500	5	0.246690	31.01
	1000	5	0.246691	31.97
	700	5	0.246691	31.91
		4	0.246696	27.29
	1000	4	0.246696	27.62

Source: own elaboration

In table 18 we see the top 5 best performing models, and all of them have $Learning\ Rate\ =\ 0.02$ and the podium also have $Max.depth\ =\ 5$. In terms of runtime, these best models are all in the middle from the distribution based on the available configurations, and no runtime is substantially lower for the model ranked just below, so we finally decide to choose the first model as the optimal GBM for frequencies:

- The selected GBM for frequencies will have:
  - $Learning\ Rate\ =\ 0.02$
  - $nº\ Estimators\ =\ 500$
  - $Max.depth\ =\ 5$

4.4.4.3. Stochastic GBM for frequencies

In our basic Grid search we tried different configurations, yet inside them we left by default the fraction of features hyperparameter, belonging to the already defined Stochastic variant of Gradient boosting, were additional randomization features are included to lower the correlation between trees.

The default value was 0.6, meaning that at each tree only a 60% of all available features will be tested for the optimal split. In our case, by rounding up this proportion from the 7 predictors used, we get that 4 features will be randomly selected at each iteration of the boosting process.

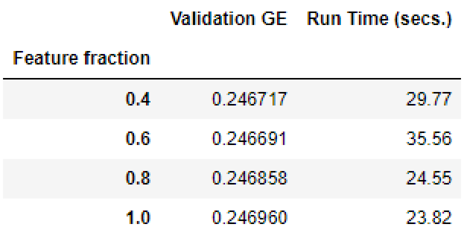
There are other randomization parameters like bagging fraction, which takes the idea from bagging trees and random forests in order to improve the diversification among trees and speed up training. Yet in our case, after using the early stopping rounds to improve runtime, and having validation GEs from the CV process, we decided this parameter does not add any additional value to our current study case.

The simple search was built by trying the next values for the feature fraction:

- $Featurefraction$: $[0.4, 0.6, 0.8, 1]$ equivalent to number of features at each tree of [3, 4, 6, 7] respectively

The results are displayed bellow:

Table 19: Stochastic GBM tunning of feature fraction



	Validation GE	Run Time (secs.)
Feature fraction		
0.4	0.246717	29.77
0.6	0.246691	35.56
0.8	0.246858	24.55
1.0	0.246960	23.82

Source: own elaboration

It can be seen that our initial default value of 0.6 was in fact the best option, yet we also note that the runtime is higher for this value, compared to values 0.8 and 1 with barely two thirds of the runtime.

With all this, our final GBM model for frequencies is tunned and ready to make predictions on a test set, as well as to make interpretations through Partial Dependency Plots. All these topics will be carefully reviewed in Chapter 5 for Performance and Results.

4.5. GBM model fitting for claim amounts.

Now it is time to move to the claim amounts modeling through GBM. And here we face similar constraints to the GLM in terms of the available data set for training and the lower predictive power from the features having a lower statistical significance for its parameters.

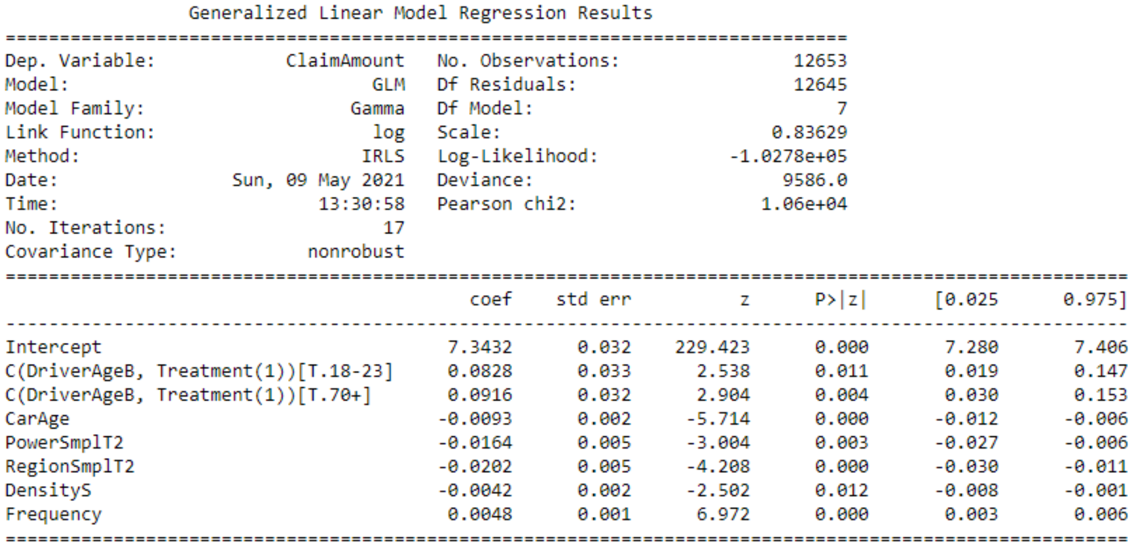
Just like with GBM for frequencies, we will fit a first model configuration and then improve upon it through a Grid search for its hyperparameters. Then we will point out the main differences with its equivalent for frequencies and its GLM counterpart.

4.5.1. Basic GBM model fitting for claim amounts

- Response variable (y): column 'ClaimAmount'.
- Predictor variable/features (x): 'CarAge', 'DriverAge', 'PowerSmplT2', 'RegionSmplT2', 'DensityS', 'Frequency'

The reason for not including BrandSmplT2 or GasT is because when using an alternative GLM for claim amounts, separate from our optimal model from section 4.2.3., which uses the previously established features but with DriverAgeB, we get significant coefficients for all variables.

Table 20: Alternative GLM for claim amounts



```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:            ClaimAmount   No. Observations:                12653
Model:                            GLM   Df Residuals:                    12645
Model Family:                   Gamma   Df Model:                            7
Link Function:                    log   Scale:                         0.83629
Method:                          IRLS   Log-Likelihood:            -1.0278e+05
Date:                Sun, 09 May 2021   Deviance:                       9586.0
Time:                        13:30:58   Pearson chi2:                 1.06e+04
No. Iterations:                    17
Covariance Type:            nonrobust
========================================================================================================
                                          coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------------------------------
Intercept                               7.3432      0.032    229.423      0.000       7.280       7.406
C(DriverAgeB, Treatment(1))[T.18-23]    0.0828      0.033      2.538      0.011       0.019       0.147
C(DriverAgeB, Treatment(1))[T.70+]      0.0916      0.032      2.904      0.004       0.030       0.153
CarAge                                 -0.0093      0.002     -5.714      0.000      -0.012      -0.006
PowerSmplT2                            -0.0164      0.005     -3.004      0.003      -0.027      -0.006
RegionSmplT2                           -0.0202      0.005     -4.208      0.000      -0.030      -0.011
DensityS                               -0.0042      0.002     -2.502      0.012      -0.008      -0.001
Frequency                               0.0048      0.001      6.972      0.000       0.003       0.006
========================================================================================================
```

Source: own elaboration

This is why we will start working with these features, and start simplifying from there, to take out less important features.

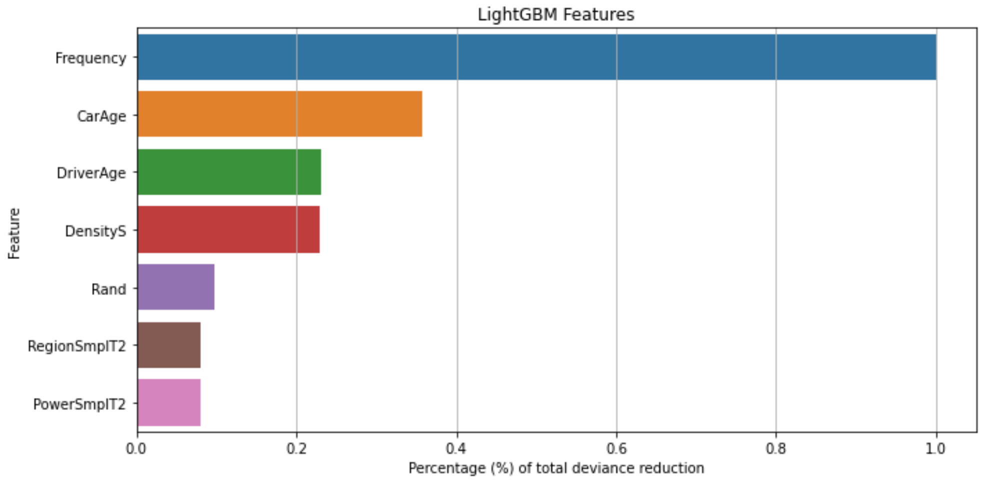
In order to do this, we add a pure *'Random'* feature generated as noise from random numbers. This feature will also participate in the boosting process, so we will discard all features with lower importance that this benchmark predictor.

But before showing the results, we quicky define the initial hyperparameter setting:

- $Max.depth\ = 2$
- $Learning\ rate\ =\ 0.02$
- $n^{\underline{o}}\ of\ estimators = 500$
- $Objective = \text{gamma}$
- $Min.child\ samples = 100$

Now we initiate and fit the LGBMRegressor, and runtime was less than 1 second, mainly due to the lower amount of data, with 12,653 instances on the train set. The resulting feature importance plot is shown below:

Figure 22: Feature importance for initial GBM for claim amounts



Source: own elaboration

Now we can see that RegionSmplT2 and PowerSmplT2 are below the implicit threshold of the Random feature. Thus, we will delete them from our model and keep only the first 4 features as predictors. This way we ensure a higher parsimony of our model given that the data amount is much lower than in our frequencies database.

With this feature selection, we can move to the tunning of the parameters in order to optimize the current model. The same process of CV will be used with Early stopping rounds and an exhaustive Grid search will be performed although with a smaller extension and much lower runtime.

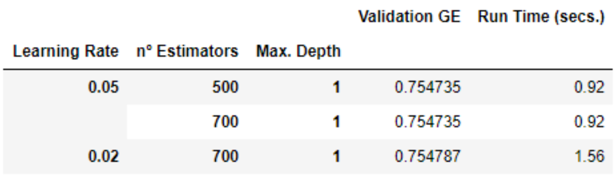
4.5.2. Hyperparameter-tunning for frequencies

The grid used for search in this case was somewhat smaller, and its configuration can be seen below:

- *Learning rate:* [0.05, 0.02, 0.01]
- *nº Estimators:* [200, 500, 700]
- *Max. depth:* [1, 2, 3]

The length of this grid will span across 27 different model combinations. The summary table will compile the results, and now we show its top 3 best performing models in terms of evaluation GE:

Table 21: Top three best performing models for claim amounts



			Validation GE	Run Time (secs.)
Learning Rate	nº Estimators	Max. Depth		
0.05	500	1	0.754735	0.92
	700	1	0.754735	0.92
0.02	700	1	0.754787	1.56

Source: own elaboration

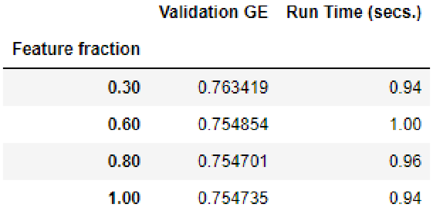
We see a tie between the two first models, and the runtimes are very similar as well. Due to the early stopping rounds in the CV model, it is very likely that both models stopped adding trees before reaching the 500 estimators, and for this reason, the best decision is to stay with the first ranked model, due to its lower number of estimators.

We also repeated the process of Stochastic GBM tunning by choosing the best value for Features fraction, and we considered the following grid:

- Feature fraction: [*0.3, 0.6, 0.8, 1*] equivalent to 1, 2, 3 and 4 features, respectively.

The results show that the feature fraction of 0.8 is performing better than the rest, meaning that we will be using tree out of four features at each iteration of the boosting trees process. The results are shown below:

Table 22. Stochastic GBM for claim amounts tunning



Feature fraction	Validation GE	Run Time (secs.)
0.30	0.763419	0.94
0.60	0.754854	1.00
0.80	0.754701	0.96
1.00	0.754735	0.94

Source: own elaboration

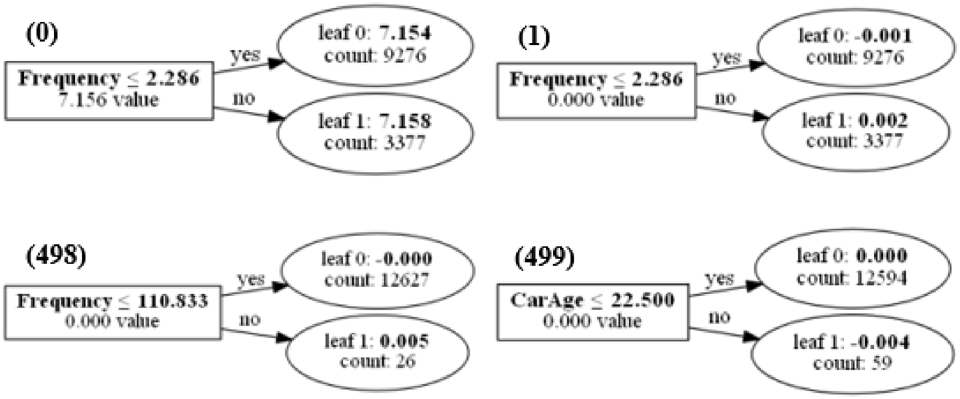
Therefore our chosen model will have the next hyperparameters:

- $Learning\ Rate\ =\ 0.02$
- $nº\ Estimators\ =\ 500$
- $Max.depth\ =\ 1$
- $Feature\ fraction = 0.8$

What might call the attention of the reader is that the maximum depth is 1, meaning that there are no significant interactions between features that the model might be missing. And as we noted in Chapter 2, these trees are called stumps.

In order to make a better illustration, we will display the first two and the last two stumps from the chosen GBM for claim amounts:

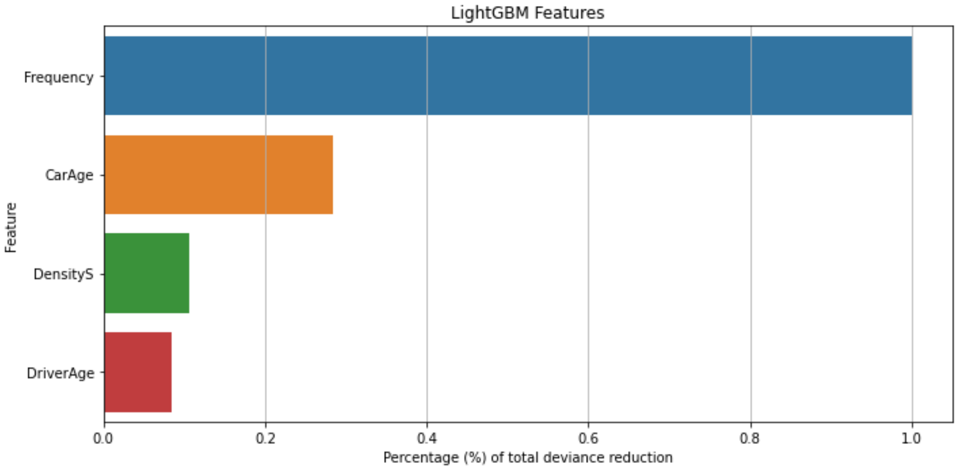
Figure 23: First two and last two stumps from claim amounts GBM



Source: own elaboration

Frequency has a big presence, as it was expected. We can confirm this by plotting again the feature importance for the tunned model.

Figure 24: Feature importance for tunned GBM for claim amounts



Source: own elaboration

Now we store this model for the performance analysis in Chapter 5 and proceed to build the pure premium predictions for the GBM models we have trained and optimized.

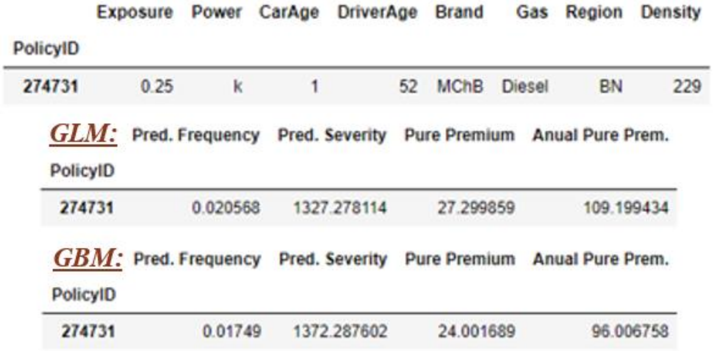
4.6. GBM pricing machine

This exercise will follow the same steps as in the GLM scenario, but this time the predictions will be made by the two LightGBM models we have previously chosen.

Now we can recover the example from table 13 about the random profile selected from the frequency test set and priced with the optimal GLMs. Only now it will also have the predictions from the GBM models.

In the example from table 23 we can already see that GLMs and GBM models are making clearly different predictions. In terms of frequency, the GLM yields a higher predicted value, but in terms of expected claim amount, the GBM model yields the highest prediction. The resulting pure premium is, however, higher for the GLM based pricing model.

Table 23: Single profile Pure premium for GLMs and GBM models



	Exposure	Power	CarAge	DriverAge	Brand	Gas	Region	Density
PolicyID								
274731	0.25	k	1	52	MChB	Diesel	BN	229

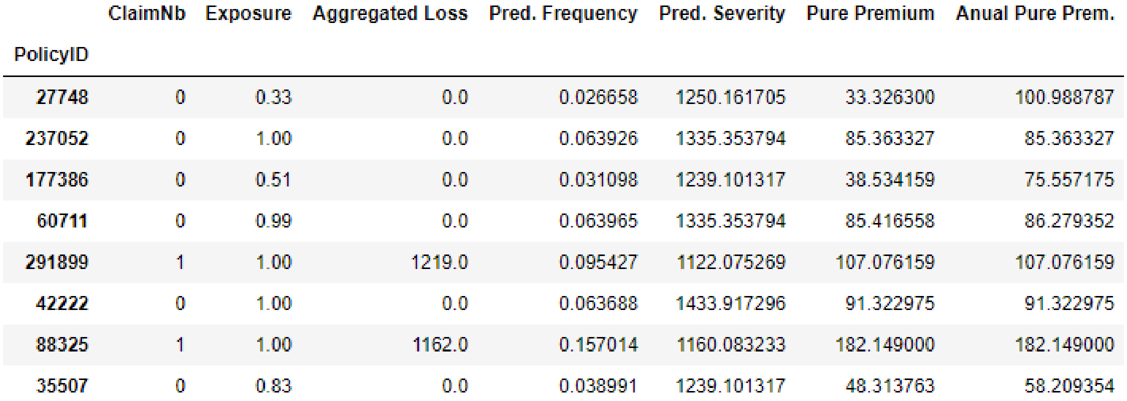
GLM:	Pred. Frequency	Pred. Severity	Pure Premium	Anual Pure Prem.
PolicyID				
274731	0.020568	1327.278114	27.299859	109.199434

GBM:	Pred. Frequency	Pred. Severity	Pure Premium	Anual Pure Prem.
PolicyID				
274731	0.01749	1372.287602	24.001689	96.006758

Source: own elaboration

A deeper prediction comparative analysis will be performed in Chapter 5, and it will be done by elaborating the same Pure Premium Analysis table from section 4.3.1. but for the GBM model predictions. And the result is displayed in the following table:

Table 24: Premium Analysis Table for GBM models



	ClaimNb	Exposure	Aggregated Loss	Pred. Frequency	Pred. Severity	Pure Premium	Anual Pure Prem.
PolicyID							
27748	0	0.33	0.0	0.026658	1250.161705	33.326300	100.988787
237052	0	1.00	0.0	0.063926	1335.353794	85.363327	85.363327
177386	0	0.51	0.0	0.031098	1239.101317	38.534159	75.557175
60711	0	0.99	0.0	0.063965	1335.353794	85.416558	86.279352
291899	1	1.00	1219.0	0.095427	1122.075269	107.076159	107.076159
42222	0	1.00	0.0	0.063688	1433.917296	91.322975	91.322975
88325	1	1.00	1162.0	0.157014	1160.083233	182.149000	182.149000
35507	0	0.83	0.0	0.038991	1239.101317	48.313763	58.209354

Source: own elaboration

Both the predictions from table 14 and 24 will be put face to face to study the predictive profiles of both the GLMs and the GBM models in Chapter 5. Several metrics will be used to assess how well both approaches adapted to the test dataset of policies, and some fundamental conclusions will be drawn.

With this, all relevant methodology for our work has been covered, going through software tools, statistical principles, visualization tools, and data wrangling techniques.

5. Performance and results

After completing our case study and training all of our models, we finally have the outputs to evaluate which models are more accurate for predicting the core actuarial random variables of claim frequencies and claim amounts.

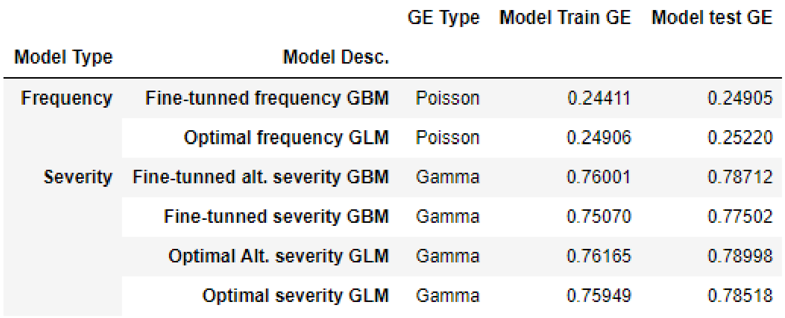
And by ‘accurate’ we mean those models that can detect and adjust more effectively to the real underlying risk profile. In other words, when a specific risk class or group of policies with very similar characteristics have consistently shown a higher (lower) claim frequency or/and a higher (lower) severity, the model will yield higher (lower) predictions in terms of frequency and/or severity. The final result will be a premium that correctly represents and adjusts to the true risk nature of each profile.

In the next section, we will present the results of this precision test both for the chosen Generalized Linear Models and for the Gradient Boosting Regression Trees according to the GE obtained from the Deviance measures.

5.1. Comparative predictive performance

We have registered the GE results for both train set and test set predictions, even though we will mainly focus on the latter. The table including these metrics is shown below:

Table 25: Performance comparison table



		GE Type	Model Train GE	Model test GE
Model Type	Model Desc.			
Frequency	Fine-tunned frequency GBM	Poisson	0.24411	0.24905
	Optimal frequency GLM	Poisson	0.24906	0.25220
Severity	Fine-tunned alt. severity GBM	Gamma	0.76001	0.78712
	Fine-tunned severity GBM	Gamma	0.75070	0.77502
	Optimal Alt. severity GLM	Gamma	0.76165	0.78998
	Optimal severity GLM	Gamma	0.75949	0.78518

Source: own elaboration

As we already noted in our work, the test set GE can be expected to be slightly higher than the train set GE, and we can clearly see this effect here on table 25. But when it comes to the difference between models, we must compare those models that have the same GE type, meaning that they were, in practice, modelling the same actuarial random variable.

When comparing frequency or number of claims models the Gradient Boosting Regression Tree has a clearly lower Generalization Error, as measured on an unseen portfolio of policies, than the Generalized Linear Model. Similarly, for severity or claim amounts models the GBRT models outperform both of their GLM counterparts.

As a reminder of the feature configuration we have for Frequency models:

- Fine-tunned frequency GBM: $ClaimNb/Exposure \sim CarAge + DriverAge + GasT + PowerT + BrandT + RegionT + DensityS$ $* \{Sample\ weights = Exposure\} *$
- Optimal frequency GLM: $ClaimNb \sim CarAge + DriverAge + Gas + PowerSmpl + Brand + RegionSmpl + DensityS$ $* \{Sample\ weights\ =\ Exposure\} *$

And for Claim Amounts:

- Fine-tunned severity GBM: $ClaimAmount \sim CarAge + DriverAge + DensityS + Frequency$
- Fine-tunned alt. severity GBM: $ClaimAmount \sim CarAge + DriverAge + DensityS$
- Optimal severity GLM: $ClaimAmount \sim CarAge + DriverAgeB + Frequency$
- Optimal Alt. severity GLM: $ClaimAmount \sim CarAge + DriverAgeB + PowerSmplT + RegionSmplT + DensityS$

Therefore the two models with the best predictive performance are:

- The Fine-tunned frequency GBM model for the number of claims.
- The Fine-tunned severity GBM model for claim amounts.

The previous results were contrasted by running the whole modeling project with 5 different random seeds for the train/test split from the original frequency table, which conditions all of our models. In all 5 runs the two previously mentioned models were once again outperforming their GLM counterparts.

Regarding the two alternative severity models, Fine-tunned alt. severity GBM and Optimal Alt. severity GLM, the former comes from the model chosen at the end of section 4.5.2. but excluding the feature ‘Frequency’, while the latter discards this same feature but for the model shown at the beginning of section 4.5.1. in Table 20. The reasoning for this strategy will be fully explained in section 5.3. for premium comparative analysis.

5.2. Partial dependency plots

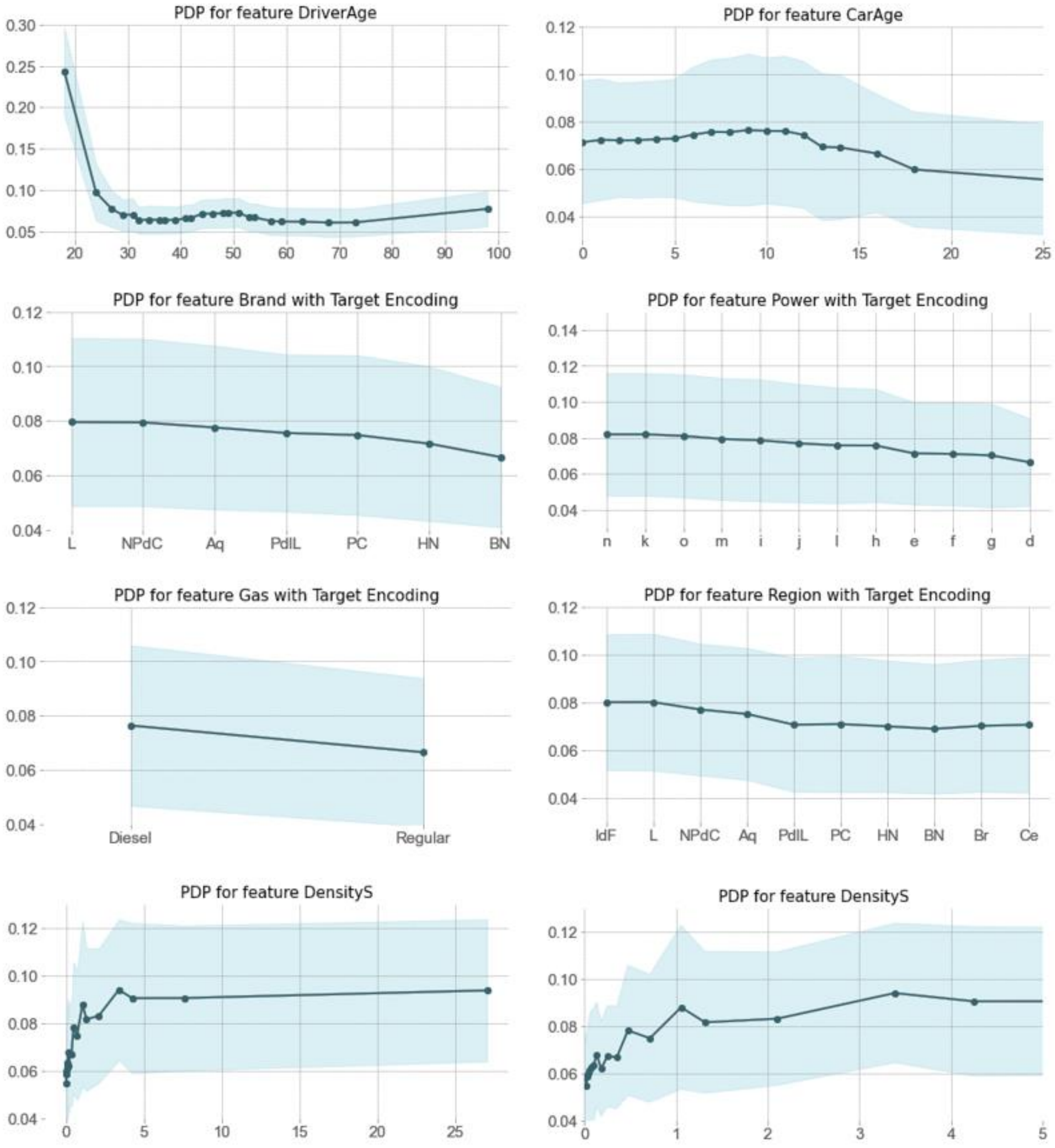
After determining the best performing models, we can finally make use of the most important interpretative tools for complex ML models like GBMs. And these tools are the Partial dependency plots (PDPs), used for assessing the ‘learned’ information from the data by our model.

The idea of PDPs is simple, they use a set of observations (maybe the train set observations or even the test set observations) to make predictions of the response variable for each sample. And for that particular instance, the prediction from the model will be repeated but by modifying the value of one feature of interest in a way that we can assess the sensitivity of the prediction result for its whole range of values.

As a quick example, if our first instance has $DriverAge = 26$ then we will use our GBM model for frequencies to make the prediction of the response. After this, we will repeat the predictions for other possible values of *DriverAge* and register the response. After performing this process on all samples, we average the response values for each *DriverAge* value, and this would give us the plot line for the PDP of the *DriverAge* feature.

In the next figure, we will depict the PDPs for all predictors in the Fine-tunned frequency GBM model.

Figure 25: PDPs for Fine-tunned frequency GBM model



Source: own elaboration

We can immediately see patters that the GLMs would not be able to capture in the same fashion. We list some of them below:

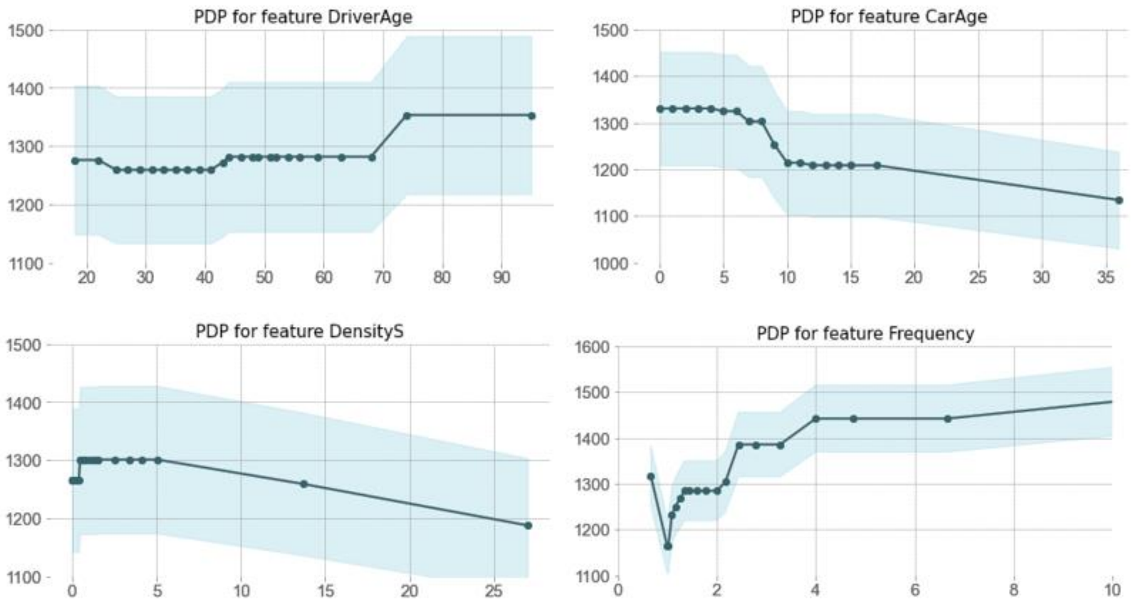
- DriverAge and CarAge have non-monotonic trend patterns that the linearity of GLMs cannot assimilate.

- DensityS seems to have a more complex dependency structure than any other feature as we can see in the last two PDPs, this can the studied further by the analyst though the effect of different relevant cities on the response.
- The categorical features, with the exception of Region, have all a descending monotonic trend, although the linearity is again not perfect, which shows that our model could learn with a higher precision the effect on response for all categories.

It is important to note, that for continuous variables like DriverAge, CarAge or DensityS, not all possible values are used to make the predictions on each instance. Instead, a set of quantile points are taken from the distribution of each feature to compute the predictions, these are called grid points, and they are the points we can see on the PDPs, drawing the main line.

The same set of plots can be derived for the features included in the Fine-tunned severity GBM.

Figure 26: PDPs for Fine-tunned severity GBM model



Source: own elaboration

Here once again we observe very detailed dependency structures for DriverAge and CarAge that could not be captured by a linear model. Some other valuable insights are:

- The possibility to establish 4 relativity levels based on DriverAge, because the PDP shows 4 intervals with a specific response level, and this information could also be introduced to a GLM with a categorical DriverAge configuration through binning.
- Something similar can be done with CarAge, with a total of 2 relativity level based on response.
- DensityS could have 3 relativities derived from the PDP, being equivalent to 3 population density intervals as city profiles.

- The complexity of Frequency gives an idea of the amount of information contained in this feature about the expected claim amount. This information cannot be disregarded, and it can even be helpful in reserving problems like IBNER estimation.

The Frequency feature has a major weight on the predictions of the model, and this makes sense with the feature importance plot seen in figure 22. In fact, the higher the importance of a figure across the model, the more information about the dependency structure will be available.

This, however, might have some undesirable effects on the pricing process for the unseen policies in the frequencies test set. We will see in the next section what these effects are and what was our approach for dealing with them.

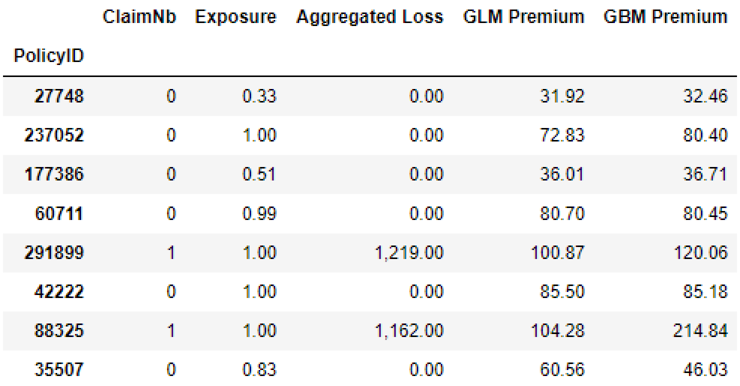
5.3. Pure premium comparative analysis

If we recall the pricing machines from sections 4.3. and 4.6. we will see that the model for severities included the feature 'Frequency' in both cases. However this will lead to a significant bias when predicting the claim amounts of profiles with no reported claims.

Given that we fitted our severity models on the train claim dataset, in all of those instances there was at least one reported claim, and the 'Frequency' feature had a value equal or above 1 on 99.91% of the samples. However, something very different happens in the context of a frequency table, like the frequency test table we intend to use for pricing, where $Frequency \geq 1$ would only be true for a 3.83% of all test profiles.

This extreme change in distribution would make the severity predictions lower, and therefore most of the premiums would tend to be inferior in value, which is something undoubtedly dangerous for an insurance company.

Table 26: Comparative pure premium table



PolicyID	ClaimNb	Exposure	Aggregated Loss	GLM Premium	GBM Premium
27748	0	0.33	0.00	31.92	32.46
237052	0	1.00	0.00	72.83	80.40
177386	0	0.51	0.00	36.01	36.71
60711	0	0.99	0.00	80.70	80.45
291899	1	1.00	1,219.00	100.87	120.06
42222	0	1.00	0.00	85.50	85.18
88325	1	1.00	1,162.00	104.28	214.84
35507	0	0.83	0.00	60.56	46.03

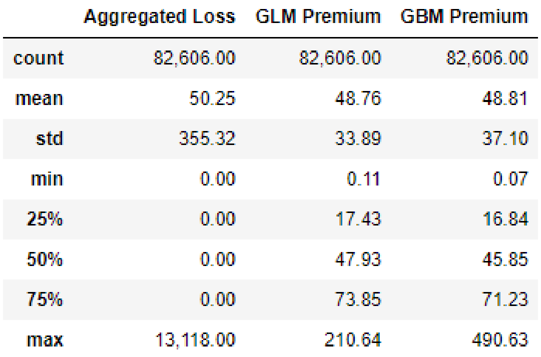
Source: own elaboration

This is why the final comparative premium analysis will be done through the alternative severity models, both for the GLM and the GBM model. The predictions based on the two adjusted pricing machines have been put face to face in a unique table, as depicted in table 26, in order to directly see the difference in pure premiums.

5.3.1. Summary statistics tables

The information in table 26 from the previous section compiles not only the two predicted pure premiums for each policy, but also the observed losses in the ‘Aggregated Loss’ column. This information can be analyzed through basic statistics about the pricing process like those presented in the next table:

Table 27: Basic summary statistics for Comparative table



	Aggregated Loss	GLM Premium	GBM Premium
count	82,606.00	82,606.00	82,606.00
mean	50.25	48.76	48.81
std	355.32	33.89	37.10
min	0.00	0.11	0.07
25%	0.00	17.43	16.84
50%	0.00	47.93	45.85
75%	0.00	73.85	71.23
max	13,118.00	210.64	490.63

Source: own elaboration

Some of the fundamental aspects to extract from this table are the following:

1. The mean aggregated loss is compared with the average premiums for both GLMs and GBM schemes. And we can see that both predicted premiums fall short to the average loss in the test portfolio.
2. Although the mean premium from the GBM pricing machine is slightly higher than its GLM analog, what is more interesting is its higher standard deviation, with almost a 10% additional dispersion compared to the GLM premiums.

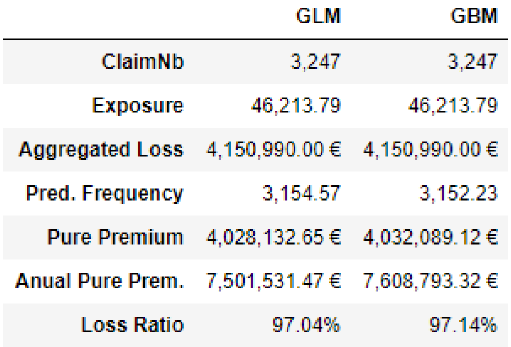
The source of our shortfall can come from the frequencies in the test set, or from the claim amounts, so we quicky calculate the average values for both actuarial variables on the train set, and then on the test set, and the results are:

- The empirical frequency on train set was 0.068202 and 0.07026 on test set, being the difference of a 3.018% in favor of the test set.
- The average severity on train set was 1,281.16€ while on test set this value was 1,278.41€. The relative difference was 0.215% in favor of the train set.

We conclude that the excess in loss on the test set comes from a higher total frequency, and this is purely an effect from the chosen random seed for splitting the original frequency table. This can be used as a ‘stressed’ scenario where our pricing schemes will have to adapt in order to cover the additional losses.

Before concluding this section, we must also show the loss performance table, where we will contrast if any of our pricing schemes was able to cover the previously mentioned shortfall.

Table 28: Comparative total aggregated loss statistics



	GLM	GBM
ClaimNb	3,247	3,247
Exposure	46,213.79	46,213.79
Aggregated Loss	4,150,990.00 €	4,150,990.00 €
Pred. Frequency	3,154.57	3,152.23
Pure Premium	4,028,132.65 €	4,032,089.12 €
Anual Pure Prem.	7,501,531.47 €	7,608,793.32 €
Loss Ratio	97.04%	97.14%

Source: own elaboration

In table 27 we see the result of calculating the total value for different metrics at a portfolio level. We see the total number of claims reported (ClaimNb) throughout all samples in the frequency test set, as well as the total number of observed years (Exposure). The Aggregated Loss row shows us the total incurred loss in the portfolio, and it is the metric that both the GLM and GBM model will have to match.

And although the GBM pricing machine gets closer to the actual loss, both of them are below its actual value. The shortfall can also be calculated in terms of Loss Ratio shown at the last row, being the only relative measure. As we expected, the Loss Ratio is slightly higher for the GBM scheme.

This type of stressed scenarios is very common to see in the insurance practice, where the actuarial random variables can naturally fluctuate in our favor or against it. In our current scenario, a common alternative is to establish a *Relative Security Loading*, of 10% additional to our premium.

5.3.2. Comparative scatter plots for predicted premiums

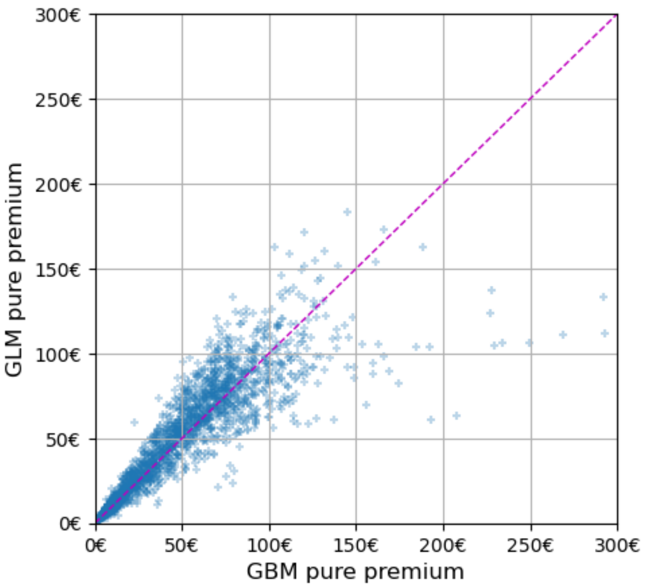
The information in table 27 and our second remark about the GBM pure premiums having a higher dispersion compared to the GLM setting rases some questions about how the distributions of the predicted premiums from both methodologies compare with each other.

Basically, we want to know how the models are working in practice and what ‘pricing styles’ correspond to each of them, in term of how they are treating riskier or less risky profiles.

Thus, our first visualization aimed at this goal will be a scatter plot that faces each GBM pure premium prediction with its GLM counterpart, which will help to visualize the degree of dispersion of the models' predictions between each other.

We will know that the predictions do not differ significantly across policies if all points lie close to the diagonal line. However, if the points tend to separate significantly from this line, then we will have evidence that there are many discrepancies between the two pricing models.

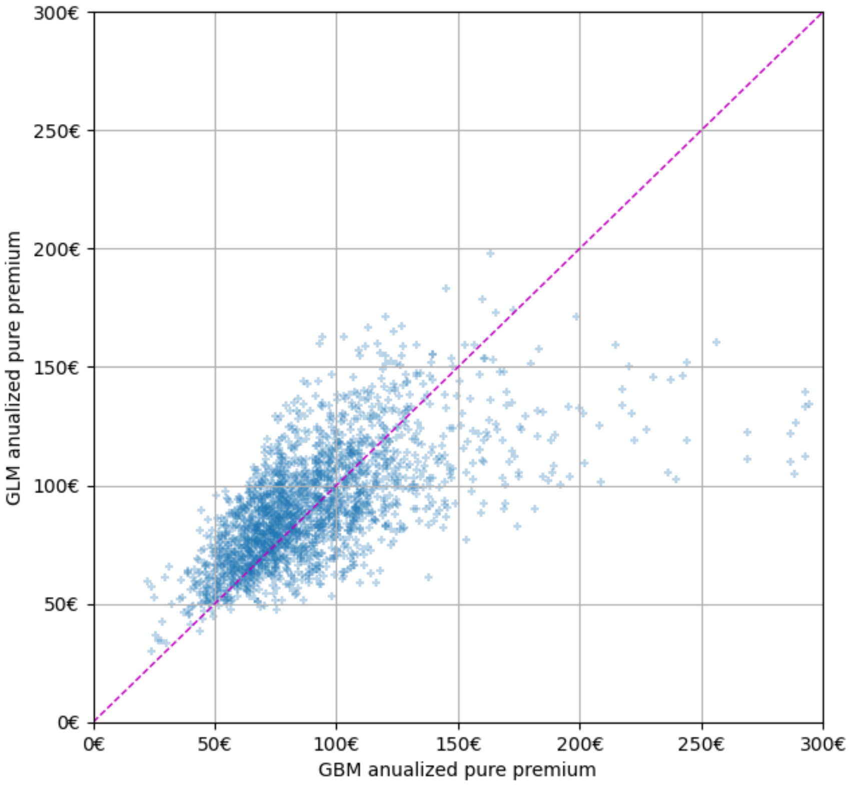
Figure 27: Comparative scatter plot for pure premium



Source: own elaboration

The scatter plot in figure 27 reveals an important discrepancy between the pricing models. And although the trend following the diagonal line is clear, the dispersion also increases with the value of the predicted premium for both pricing models. One of the most important and insightful details from the plot is the 'leakage' of points in the lower triangle from the diagonal, specifically towards the sections on the right, corresponding to the higher GBM pure premium values.

Figure 28: Comparative scatter plot for annualized pure premium



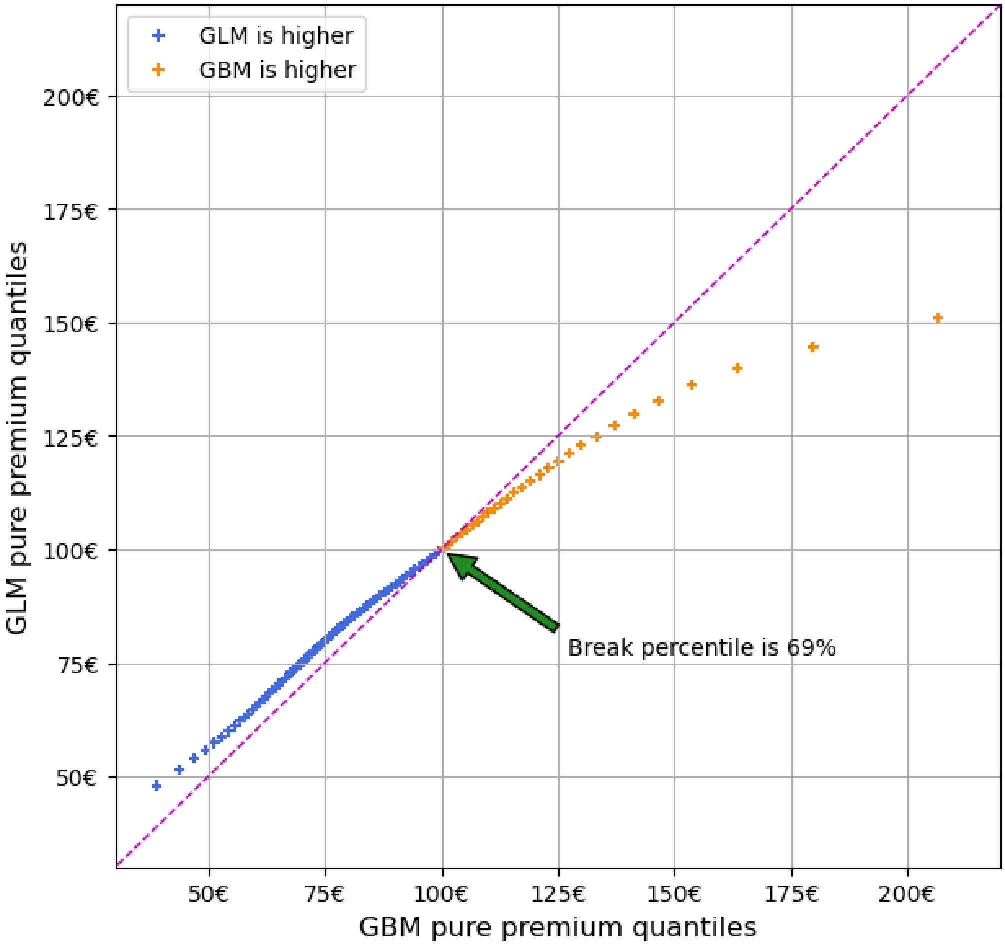
Source: own elaboration

The 'leakage' effect means that the GBM models is pricing some profiles with a much higher premium than the GLMs. But these policies are scaled to the exposure of the observations, which can delude some important information about the overall discrepancies. And given that in reality the new policies are usually subscribed for periods of one year, we can set the same comparison but for the annualized pure premiums in a new scatter plot in figure 28.

Now we can see additional information about the difference in premiums depending on the pricing scheme. Not only the 'leakage' is more evident now, but also the higher concentration of points in the upper triangle towards lower premium values. This means that the GBM pricing method is charging a lower premium to many policies than the charge from the GLM counterpart.

The interpretation of these two facts is key to understanding the differences in pricing between the two modeling approaches. And for this purpose we will analyze the similarity between the distributions of their annualized pure premiums. A common method this type of comparison is through a QQplot, and we will use it to compare the percentiles from 1% to 99% of the annualized pure premium from the GBM pricing method with the same percentiles from the GLM pricing method.

Figure 29: QQplot for pure premium pricing schemes distributions



Source: own elaboration

As we were already anticipating from the previous plot, the distribution of the pure premiums from both GLMs and GBM models has significant differences on both ends when annualized. Specifically, we see that for values below 100€, the premiums calculated by the GLMs are in fact higher than those from GBM models. This changes after percentile 69%, where GBM models surpass GLMs in their predicted premiums and the closer to the right end of the distribution, the higher the tuition from the GBM based pricing machine.

In this sense, we see that GBM models charge a significantly higher premium to the profiles it considers as riskier, while also charging less to the perceived as less risky profiles, all of it when comparing to the GLM based pricing machine. Based on these results, we conclude that the GBM pricing machine has a higher capacity for differentiating high risk profiles from those with lower risk.

6. Conclusions

Gradient Boosting algorithms have all the necessary statistical properties to effectively capture the risk nature of different insurance profiles, both for predicting actuarial variables like frequencies and severities and for building a full pricing scheme where the basis for a final tariff can be calculated.

In fact, GBM based pricing models have proven to possess a higher predictive performance and a better risk profile assessment capacity than their predecessors based on Generalized Linear Models. In this sense, an insurance company can rely on the information and insights provided by these Ensembles of Trees in order to enhance the overall pricing process and seek to improve its corporate results.

However, regulations in the field of pricing, solvency and internal auditing can still hamper the expansion of Gradient Boosting as a dominant tariff scheme, given that GLMs have an interpretative structure already cemented and trusted in the industry and across stakeholders, whilst GBMs are still considered as part of a '*black-box*' paradigm where the complexity of the inner process still generates doubts across analysts and supervisors.

In this regard, the integration of Machine Learning in the insurance corporate environment is expected to be slow, although constant in time. For the purpose of our proposed implementation, we consider that Gradient Boosting algorithms can already be implemented and deployed in insurance companies as a complementary analytical and comparative tool that enhances the training for GLMs and their tariff criteria, by giving proposed feature intervals based in the information of Partial Dependency Plots in order to obtain a set of more representative relativities. This way, the ease of GBM models for detecting non-linearities and non-monotonic patterns can already be leveraged in the GLM context.

This strategy is just part of the overall transformation process that is currently taking place in the insurance analytical environment, it is possible that better implementations of Gradient Boosting will be developed and eventually used for similar purposes. One example is the rapid growth of Neural Networks based applications, and their reach spans across multiple potential actuarial applications.

In addition, the pricing process can be enriched not only by the contributions of GBM models for basic risk costing, as we have seen in our work, but also for more advanced stages like those depicted by Parodi, P. (2015) as the *"High-level Pricing Process"*, including security and risk loadings, expense loadings as well as capital loadings.

Other complementary ML applications can be proposed for the extreme claim amounts filtered out in our study case, because in reality those claims also must be modeled and carefully studied by the analyst, and in this context proposals like the one from Valdivia, L. (2020) can be useful, where Generative Adversarial Networks (GAN) are used to model infrequent events such as extreme values, similar to our excluded policies, which were much less frequent than the common claim amounts. One of the main conclusions of this work is that the implementation of GANs along with a classical Extreme Value Theory model yields significantly better results.

This last application proves that our proposed strategy of using an advanced Machine Learning method like Gradient Boosting along with a more traditional statistical method like GLMs in order to improve the overall analytical process can actually be a successful strategy and it is worth being considered in today’s high-level insurance and actuarial environments.

7. Bibliography

7.1. Literary references

Ahlgren, M. (2018). Claims Reserving using Gradient Boosting and Generalized Linear Models. *KTH, Matematisk statistik.*

Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324

Chen, T., & Guestrin, C. (Aug 13, 2016). *XGBoost. Paper presented at the 785-794.* https://doi.org/10.1145/2939672.2939785 http://dl.acm.org/citation.cfm?id=2939785

Davoudi Kakhki, F., Freeman, S., & Mosher, G. (2018). Analyzing Large Workers' Compensation Claims Using Generalized Linear Models and Monte Carlo Simulation. *Safety (Basel),* 4(4), 57. https://doi.org/10.3390/safety4040057

Denuit, M., Hainaut, D., & Trufin, J. (2019). Effective Statistical Learning Methods for Actuaries I. *Springer International Publishing AG*.

Denuit, M., Hainaut, D., & Trufin, J. (2020). Effective Statistical Learning Methods for Actuaries II. *Springer International Publishing AG*.

Diana, A., Griffin, J. E., Oberoi, J. S., & Yao, J. (2019). Machine-Learning Methods for Insurance Applications - a survey. *Society of Actuaries*. https://kar.kent.ac.uk/71090

Freund, Y., & Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1), 119-139. https://doi.org/10.1006/jcss.1997.1504

Garrido, J., Genest, C., & Schulz, J. (2016). Generalized linear models for dependent frequency and severity of insurance claims. *Insurance, Mathematics & Economics*, 70, 205-215. https://doi.org/10.1016/j.insmatheco.2016.06.006

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition (2nd ed.). *O'Reilly Media, Inc*.

Guelman, L. (2012). Gradient boosting trees for auto insurance loss cost modeling and prediction. *Expert Systems with Applications*, 39(3), 3659-3667. https://doi.org/10.1016/j.eswa.2011.09.058

Hanafy, M., & Ming, R. (2021). Machine Learning Approaches for Auto Insurance Big Data. *Risks (Basel)*, 9(2), 42. https://doi.org/10.3390/risks9020042

Hastie, T., Tibshirani, R., & Friedman, J. (2009). Elements of Statistical Learning. *Springer*.

Jerome H. Friedman. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5), 1189-1232. https://doi.org/10.2307/2699986

Maillart, A. (2021). Toward an explainable machine learning model for claim frequency: a use case in car insurance pricing with telematics data. *European Actuarial Journal*, https://doi.org/10.1007/s13385-021-00270-5

Ng, S., Lestari, D., & Devila, S. (2019). Generalized linear model for deductible pricing in non-life insurance. *AIP Conference Proceedings*, 2168(1) https://doi.org/10.1063/1.5132465

Noll, A., Salzmann, R., & Wuthrich, M. V. (2015). Case Study: French Motor Third-Party Liability Claims. *SSRN Electronic Journal*, https://doi.org/10.2139/ssrn.316476

Ohlsson, E., & Johansson, B. (2015). Non-Life Insurance Pricing with Generalized Linear Models. *Springer Berlin / Heidelberg*.

Parodi, P. (2015). Pricing in General Insurance (1st ed.). *Chapman and Hall/CRC*. https://doi.org/10.1201/b17525

Quan, Z., & Valdez, E. A. (2018). Predictive analytics of insurance claims using multivariate decision trees. *Dependence Modeling*, 6(1), 377-407. https://doi.org/10.1515/demo-2018-0022

Su, X., & Bai, M. (2020). Stochastic gradient boosting frequency-severity model of insurance claims. *PloS One*, 15(8), e0238000. https://doi.org/10.1371/journal.pone.0238000

Sun, S., Bi, J., Guillen, M., & Pérez-Marín, A. M. (2020). Assessing Driving Risk Using Internet of Vehicles Data: An Analysis Based on Generalized Linear Models. *Sensors (Basel, Switzerland)*, 20(9), 2712. https://doi.org/10.3390/s20092712

Valdivia Ameller, L. A. (2020). Uso de datos sintéticos provenientes de redes neuronales para la mejora de la modelización de la severidad de eventos infrecuentes. *Centro De Documentación Fundación Mapfre,* https://documentacion.fundacionmapfre.org/documentacion/publico/i18n/catalogo_imagenes/grupo.do?path=1108389

7.2. Software documentation

Dutang, C., & Charpentier, A. (2020). CASdatasets-manual. http://cas.uqam.ca/pub/web/CASdatasets-manual.pdf

McKinney, W., & Pandas Development Team. (2021). pandas: powerful Python data analysis toolkit [computer software]. https://pandas.pydata.org/docs/pandas.pdf

Microsoft, & LightGBM Contributors. (2021). LightGBM. https://lightgbm.readthedocs.io/en/latest/

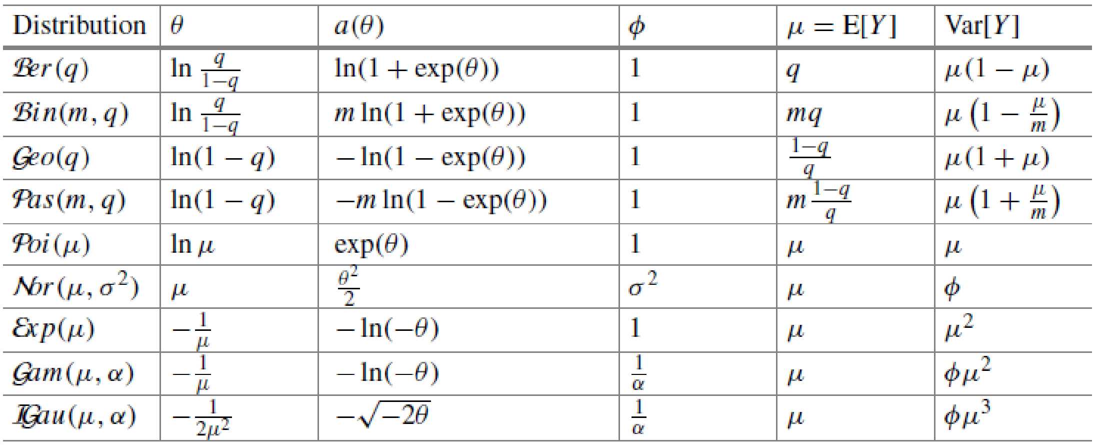
scikit-learn, 0. 2. 2. (2021). User Guide. https://scikit-learn.org/stable/user_guide.html

statsmodels v0.13.0.dev0. (2021). Generalized Linear Models. https://www.statsmodels.org/stable/glm.html

Appendix

A. Parametrization for EDM family member distributions

The following table was extracted from Denuit *et al.*, (2019) and it represents the the parameters and cumulant function of different functions belonging to the EDM family of distributions. In each row we also have the mean and variance for illustrative purposes.



Distribution	θ	$a(\theta)$	ϕ	$\mu = \mathrm{E}[Y]$	$\mathrm{Var}[Y]$
$\mathcal{B}er(q)$	$\ln \frac{q}{1-q}$	$\ln(1+\exp(\theta))$	1	q	$\mu(1-\mu)$
$\mathcal{B}in(m,q)$	$\ln \frac{q}{1-q}$	$m\ln(1+\exp(\theta))$	1	mq	$\mu\left(1-\frac{\mu}{m}\right)$
$\mathcal{G}eo(q)$	$\ln(1-q)$	$-\ln(1-\exp(\theta))$	1	$\frac{1-q}{q}$	$\mu(1+\mu)$
$\mathcal{P}as(m,q)$	$\ln(1-q)$	$-m\ln(1-\exp(\theta))$	1	$m\frac{1-q}{q}$	$\mu\left(1+\frac{\mu}{m}\right)$
$\mathcal{P}oi(\mu)$	$\ln\mu$	$\exp(\theta)$	1	μ	μ
$\mathcal{N}or(\mu,\sigma^2)$	μ	$\frac{\theta^2}{2}$	σ^2	μ	ϕ
$\mathcal{E}xp(\mu)$	$-\frac{1}{\mu}$	$-\ln(-\theta)$	1	μ	μ^2
$\mathcal{G}am(\mu,\alpha)$	$-\frac{1}{\mu}$	$-\ln(-\theta)$	$\frac{1}{\alpha}$	μ	$\phi\mu^2$
$\mathcal{IG}au(\mu,\alpha)$	$-\frac{1}{2\mu^2}$	$-\sqrt{-2\theta}$	$\frac{1}{\alpha}$	μ	$\phi\mu^3$

Source: Denuit, M., Hainaut, D., & Trufin, J. (2019). *Effective Statistical Learning Methods for Actuaries I*

B. Simulated dataset for Decision Tree illustration

Our simulated dataset was developed in a Python Jupyter Lab environment with the purpose of illustrating how can Decision Trees approach a complex variable dependency structure.

```
import sys
import sklearn
import pandas as pd
import numpy as np
import os
np.random.seed(42)

# To plot pretty figures
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
```

#Artificial Dataset

#The simulation will generate and artificial dataset for claims reported for cars of different ages. Our predicted variable #will be Claim, and will take values of 1 (claim reported) or 0 (no claim). The predictor and sole feature will be Vehicle Age #and will take different integer values from 0 to 20.

```
import random
random.seed(28)
age = np.random.binomial(n = 20, p = 0.05, size = 1000)
age2 = np.random.binomial(n = 20, p = 0.2, size = 1000)
age3 = np.random.binomial(n = 20, p = 0.5, size = 1000)
age4 = np.random.binomial(n = 20, p = 0.75, size = 500)
age5 = np.random.binomial(n = 20, p = 0.95, size = 500)

np.unique(age, return_counts = True)
np.unique(age3, return_counts = True)
np.unique(age4, return_counts = True)
np.unique(age5, return_counts = True)
ageveh = np.concatenate((age, age2, age3, age4, age5))
claim = np.concatenate((np.repeat(1, 1000), np.repeat(0, 1000), np.repeat(1, 1000), np.repeat(0, 5
00), np.repeat(1, 500)))
ageveh.shape
claim.shape
```

#We must now re arrange the data into a DataFrame

```
dataset = pd.DataFrame({'Claim': claim, 'Vehicle Age': ageveh})
dataset
yes = dataset.loc[dataset["Claim"] == 0, "Vehicle Age"].value_counts()
no = dataset.loc[dataset["Claim"] == 1, "Vehicle Age"].value_counts()
yes = yes.sort_index()
no = no.sort_index()
yes.head(10)
no.head(10)
ind = list(range(4000))
rnd = random.sample(ind, k = 4000)
len(rnd)

dataset = dataset.reindex(rnd)

dataset["PolicyID"] = ind
dataset = dataset.set_index(keys="PolicyID")
#dataset.reset_index()
```

```
dataset
fig, ax = plt.subplots()
ax.scatter(dataset["Vehicle Age"], dataset["Claim"], alpha = 0.01, color = "r")
ax.set_xlabel("Age of Vehicle")
ax.set_ylabel("Claim status")
ax.set_title("Claim Density for Age of Vehicle")
plt.show()

type(yes)

yesno = pd.merge(left = yes, right = no, how = "outer", left_index = True, right_index = True)
yesno.columns = ['Yes', 'No']
yesno = yesno.fillna(0)
yesno['Yes'] = yesno['Yes'].astype('int64')
yesno
fig, ax = plt.subplots()
ax.bar(yesno.index, yesno["Yes"], label = "Yes")
ax.bar(yesno.index, yesno["No"], bottom = yesno["Yes"], label = "No")
ax.set_ylabel("Nº of policies")
ax.set_xlabel("Age of the Vehicle")
ax.set_title("Claim ocurrance count per vehicle Age")
ax.legend()
```

#Decision Tree for Classification

#We will use sklearn DecisionTreeClassifier to illustrate and example based in our artificial dataset.

```
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn import metrics

X = dataset["Vehicle Age"].to_numpy().reshape(-1, 1)
y = dataset["Claim"].to_numpy().reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
clf = tree.DecisionTreeClassifier(max_depth=4, max_leaf_nodes=5)

clf = clf.fit(X_train, y_train)

tree.plot_tree(clf)
```

C. GLM and GBM modeling and comparison

This is the source code for the main analysis of our work, it covers the tables, figures and procedures depicted and described in Chapters 3, 4, 5 and 6. All outputs have been cleared, although some comments and section headlines are still present in order to ease identifying each part.

# French Motor TPL

```
import sys
import sklearn
import pandas as pd
import numpy as np
import os
np.random.seed(42)

# To plot pretty figures

import matplotlib as mpl
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
```

# Data Import for French Motor TPL Portfolio

```
freMTPLfreq = pd.read_csv("C:\\Users\\usuario\\Desktop\\Uni\\Máster 2º Año\\Cuarto Cuatrimestre\\TFM\\TFM code and data\\freMTPLfreq.csv", index_col = 'PolicyID')
freMTPLsev = pd.read_csv("C:\\Users\\usuario\\Desktop\\Uni\\Máster 2º Año\\Cuarto Cuatrimestre\\TFM\\TFM code and data\\freMTPLsev.csv", index_col = 'PolicyID')
```

# Basic Exploratory Data Analysis

```
# pd.options.display.max_rows = 20
freqs0 = freMTPLfreq
freqs0.head(10)

freMTPLfreq['Brand'].value_counts()

freqs0.describe()

freqs0['DriverAge'].value_counts().sort_index()[70:] # Delete all values above 50, probably outliers

freqs0[['Power', 'Brand', 'Gas', 'Region']] = freqs0[['Power', 'Brand', 'Gas', 'Region']].astype("category")
freqs0.info()

freqs0['Brand'].cat.categories
```

# Additional feature versions are introduced:

```
freqs0['DensityS'] = freqs0['Density']/1000
freqs0['DriverAgeB'] = pd.cut(freqs0.DriverAge, bins = [0, 23.5, 69.5, 100], labels = ['18-23', '24-69', '70+'])
freqs0['Frequency'] = freqs0['ClaimNb'] / freqs0['Exposure']
```

# Claim Amounts preprocessing

# All claim amounts above 10_000 are filtered out from uor database

```
# claims1.loc[claims1["ClaimAmount"] > 10_000, "ClaimAmount"] = 10_000
outclaims = freMTPLsev[freMTPLsev['ClaimAmount'] > 10_000]
claims = freMTPLsev[freMTPLsev['ClaimAmount'] <= 10_000]
claims.shape

print(f'Total initial claim amounts where {freMTPLsev.shape[0]}')
print(f'Total current claim amounts are {claims.shape[0]}')
print(f'The outlier claims where {outclaims.shape[0]}, which is a {outclaims.shape[0]/freMTPLsev.shape[0]:.2%} of the total dataset')

corrector = outclaims.groupby(level = 0).count()
corrector['ClaimAmount']

freqs = freqs0.copy()
freqs0.loc[corrector.index, 'ClaimNb']
```

# We correct the frequencies based on the information from the filtered out counts to maintain data integrity.

```
freqs.loc[corrector.index, 'ClaimNb'] = freqs0.loc[corrector.index, 'ClaimNb'] - corrector['ClaimAmount']
freqs.loc[corrector.index, 'ClaimNb']

freqs['ClaimNb'].sum()
```

# Renaming categories

```
freqs['Brand'] = freqs['Brand'].cat.rename_categories({'Fiat':'Fi', 'Japanese (except Nissan) or Korean':'J-{N}/K',
                                                       'Mercedes, Chrysler or BMW':'MChB', 'Opel, General Motors or Ford':'OGmF',
                                                       'Renault, Nissan or Citroen':'RNC', 'Volkswagen, Audi, Skoda or Seat':'VASkSe', 'other':'other'})
freqs['Region'] = freqs['Region'].cat.rename_categories({"Aquitaine":"Aq", "Basse-Normandie":"BN", "Bretagne":"Br", "Centre":"Ce",
                                                         "Haute-Normandie":"HN", "Ile-de-France":"IdF", "Limousin":"L", "Nord-Pas-de-Calais":"NPdC",
                                                         "Pays-de-la-Loire":"PdlL", "Poitou-Charentes":"PC"})
```

# Simplifying categories

```
freqs["PowerSmpl"] = freqs.Power.astype(str)
freqs.loc[freqs["PowerSmpl"]=="e", "PowerSmpl"] = "eg"
freqs.loc[freqs["PowerSmpl"]=="g", "PowerSmpl"] = "eg"
freqs.loc[freqs["PowerSmpl"]=="l", "PowerSmpl"] = "klmno"
freqs.loc[freqs["PowerSmpl"]=="m", "PowerSmpl"] = "klmno"
freqs.loc[freqs["PowerSmpl"]=="n", "PowerSmpl"] = "klmno"
freqs.loc[freqs["PowerSmpl"]=="o", "PowerSmpl"] = "klmno"
freqs.loc[freqs["PowerSmpl"]=="k", "PowerSmpl"] = "klmno"
freqs["PowerSmpl"] = freqs.PowerSmpl.astype("category")
freqs["PowerSmpl"].value_counts()

freqs["RegionSmpl"] = freqs.Region.astype(str)
freqs.loc[freqs["RegionSmpl"]=="Br", "RegionSmpl"] = "BrBNHN"
freqs.loc[freqs["RegionSmpl"]=="BN", "RegionSmpl"] = "BrBNHN"
freqs.loc[freqs["RegionSmpl"]=="HN", "RegionSmpl"] = "BrBNHN"
freqs.loc[freqs["RegionSmpl"]=="L", "RegionSmpl"] = "LNPdC"
freqs.loc[freqs["RegionSmpl"]=="NPdC", "RegionSmpl"] = "LNPdC"
freqs["RegionSmpl"] = freqs.RegionSmpl.astype("category")
freqs["RegionSmpl"].value_counts()

freqs.Region.value_counts()

freqs["BrandSmpl"] = freqs.Brand.astype(str)
freqs.loc[freqs["BrandSmpl"]=="RNC", "BrandSmpl"] = "RNC_J-{N}/k"
freqs.loc[freqs["BrandSmpl"]=="J-{N}/K", "BrandSmpl"] = "RNC_J-{N}/k"
freqs.loc[freqs["BrandSmpl"]=="MChB", "BrandSmpl"] = "MChB_OGmF"
freqs.loc[freqs["BrandSmpl"]=="OGmF", "BrandSmpl"] = "MChB_OGmF"
freqs["BrandSmpl"] = freqs.BrandSmpl.astype("category")
freqs["BrandSmpl"].value_counts()
```

```
# Feature Summary Tables

CarAgeData = freqs.groupby(['CarAge']).sum()[['ClaimNb', 'Exposure']]
CarAgeData['Empirical Frequency'] = CarAgeData['ClaimNb'] / CarAgeData['Exposure']

DriverAgeData = freqs.groupby(['DriverAge']).sum()[['ClaimNb', 'Exposure']]
DriverAgeData['Empirical Frequency'] = DriverAgeData['ClaimNb'] / DriverAgeData['Exposure']

PowerData = freqs.groupby(['Power']).sum()[['ClaimNb', 'Exposure']]
PowerData['Empirical Frequency'] = PowerData['ClaimNb'] / PowerData['Exposure']

BrandData = freqs.groupby(['Brand']).sum()[['ClaimNb', 'Exposure']]
BrandData['Empirical Frequency'] = BrandData['ClaimNb'] / BrandData['Exposure']

GasData = freqs.groupby(['Gas']).sum()[['ClaimNb', 'Exposure']]
GasData['Empirical Frequency'] = GasData['ClaimNb'] / GasData['Exposure']

RegionData = freqs.groupby(['Region']).sum()[['ClaimNb', 'Exposure']]
RegionData['Empirical Frequency'] = RegionData['ClaimNb'] / RegionData['Exposure']

PowerSmplData = freqs.groupby(['PowerSmpl']).sum()[['ClaimNb', 'Exposure']]
PowerSmplData['Empirical Frequency'] = PowerSmplData['ClaimNb'] / PowerSmplData['Exposure']

RegionSmplData = freqs.groupby(['RegionSmpl']).sum()[['ClaimNb', 'Exposure']]
RegionSmplData['Empirical Frequency'] = RegionSmplData['ClaimNb'] / RegionSmplData['Exposure']

BrandSmplData = freqs.groupby(['BrandSmpl']).sum()[['ClaimNb', 'Exposure']]
BrandSmplData['Empirical Frequency'] = BrandSmplData['ClaimNb'] / BrandSmplData['Exposure']

# Target/Response encoding

cats = ['Power', 'Brand', 'Gas', 'Region', 'PowerSmpl', 'RegionSmpl', 'BrandSmpl']
catsT = [cat+'T' for cat in cats]
tables = [PowerData, BrandData, GasData, RegionData, PowerSmplData, RegionSmplData, BrandSmplData]
keys = [data.sort_values(by = 'Empirical Frequency', ascending = False).index for data in tables]

# We program the encoder function that will take any set of categorical values sorted decreasingly
according
# to their mean effect on the response.
def TargetEncoder(keys):
    dicts = {}
    values = 0
    for i in keys:
            dicts[i] = values
            values += 1
    return dicts
Encodings = {cat:TargetEncoder(key) for (cat,key) in zip(catsT,keys)}
print(Encodings)

# We assign new versions of our categorical variables and establish the target encoding for them
for i in range(len(cats)):
    freqs[catsT[i]] = freqs[cats[i]]
freqs = freqs.replace(Encodings)

# The unpdated frequency table can the visualized:
freqs.head()

BrandSmplTData = freqs.groupby(['BrandSmplT']).sum()[['ClaimNb', 'Exposure']]
BrandSmplTData['Empirical Frequency'] = BrandSmplTData['ClaimNb'] / BrandSmplTData['Exposure']
BrandSmplTData

claims = pd.merge(left = claims, right = freqs, how = "left", left_index = True, right_index = Tru
e)
```

```
claims['Frequency'] = claims['ClaimNb'] / claims['Exposure']
claims = claims.drop(['ClaimNb','Exposure'], axis = 1)
claims.head(10)

claims.describe()
```

# Claim Amount Target Encoding

```
DriverAgeData['Avg. ClaimAmount'] = claims.groupby(['DriverAge']).mean()[['ClaimAmount']]
DriverAgeData['Vol. ClaimAmount'] = claims.groupby(['DriverAge']).std()[['ClaimAmount']]

CarAgeData['Avg. ClaimAmount'] = claims.groupby(['CarAge']).mean()[['ClaimAmount']]
CarAgeData['Vol. ClaimAmount'] = claims.groupby(['CarAge']).std()[['ClaimAmount']]

PowerData['Avg. ClaimAmount'] = claims.groupby(['Power']).mean()[['ClaimAmount']]
PowerData['Vol. ClaimAmount'] = claims.groupby(['Power']).std()[['ClaimAmount']]

BrandData['Avg. ClaimAmount'] = claims.groupby(['Brand']).mean()[['ClaimAmount']]
BrandData['Vol. ClaimAmount'] = claims.groupby(['Brand']).std()[['ClaimAmount']]

GasData['Avg. ClaimAmount'] = claims.groupby(['Gas']).mean()[['ClaimAmount']]
GasData['Vol. ClaimAmount'] = claims.groupby(['Gas']).std()[['ClaimAmount']]

RegionData['Avg. ClaimAmount'] = claims.groupby(['Region']).mean()[['ClaimAmount']]
RegionData['Vol. ClaimAmount'] = claims.groupby(['Region']).std()[['ClaimAmount']]

PowerSmplData['Avg. ClaimAmount'] = claims.groupby(['PowerSmpl']).mean()[['ClaimAmount']]
PowerSmplData['Vol. ClaimAmount'] = claims.groupby(['PowerSmpl']).std()[['ClaimAmount']]

BrandSmplData['Avg. ClaimAmount'] = claims.groupby(['BrandSmpl']).mean()[['ClaimAmount']]
BrandSmplData['Vol. ClaimAmount'] = claims.groupby(['BrandSmpl']).std()[['ClaimAmount']]

RegionSmplData['Avg. ClaimAmount'] = claims.groupby(['RegionSmpl']).mean()[['ClaimAmount']]
RegionSmplData['Vol. ClaimAmount'] = claims.groupby(['RegionSmpl']).std()[['ClaimAmount']]

cats = ['PowerSmpl', 'RegionSmpl', 'BrandSmpl']
catsT2 = [cat+'T2' for cat in cats]
tables = [PowerSmplData, RegionSmplData, BrandSmplData]
keys = [data.sort_values(by = 'Avg. ClaimAmount', ascending = False).index for data in tables]

Encodings2 = {cat:TargetEncoder(key) for (cat,key) in zip(catsT2,keys)}
print(Encodings2)

# We assign new versions of our categorical variables and establish the target encoding for them
for i in range(len(cats)):
    freqs[catsT2[i]] = freqs[cats[i]]
freqs = freqs.replace(Encodings2)

claims = pd.merge(left = claims, right = freqs[catsT2], how = "left", left_index = True, right_ind
ex = True)
```

# Target-Response encoding visualization

```
Bkeys = BrandData.sort_values(by = 'Avg. ClaimAmount', ascending = False).index
BrandEnc = TargetEncoder(Bkeys)
BrandEnc

BrandData[['Empirical Frequency']].sort_values(by='Empirical Frequency', ascending = False)

BrandEnc1Table = pd.DataFrame(data = list(Encodings['BrandT'].values()),
                              index = list(Encodings['BrandT'].keys()), columns = ['Freqs. Response
-encoding'])
BrandEnc2Table = pd.DataFrame(data = list(BrandEnc.values()),
                             index = list(BrandEnc.keys()), columns = ['Claims Response-encoding']
)
BrandEncTable = pd.merge(left = BrandEnc1Table, right = BrandEnc2Table, how='left', left_index = T
rue, right_index = True)
BrandEncTable

BrandData[['Avg. ClaimAmount']].sort_values(by='Avg. ClaimAmount', ascending = False)

claims.info()
```

# Assesing Poisson assumption for number of claims

```
claimcountdistr = pd.DataFrame({'nº Policies':freqs.ClaimNb.value_counts().values, 'ClaimNb':freqs
.ClaimNb.value_counts().index})
claimcountdistr.set_index('ClaimNb')

mean = freqs['ClaimNb'].describe()['mean']
variance = freqs['ClaimNb'].describe()['std'] ** 2
print('Mean is {}'.format(round(mean, 5)))
print('Variance is {}'.format(round(variance, 5)))

variance/mean

# Compute and print sample mean of the number of satellites: sat_mean
fr_mean = np.mean(freqs.ClaimNb)

print('Sample mean is', round(fr_mean, 3))

# Compute and print sample variance of the number of satellites: sat_var
fr_var = np.var(freqs.ClaimNb)
print('Sample variance is', round(fr_var, 3))

# Compute ratio of variance to mean
print('Variance on mean ratio is', round(fr_var/fr_mean, 3))
```

# Feature preprocesing:

# CarAge

```
outliers = freqs[freqs['CarAge'] > 50]
print('We will be deleating', sum(outliers['ClaimNb']), 'Claim Amounts and a total of', outliers.s
hape[0], 'observed Policies')

filterfreq = list(outliers.index.values)
freqs = freqs.drop(filterfreq)
freqs.shape

filtersev = list(outliers[outliers['ClaimNb']>0].index.values)
claims = claims.drop(filtersev)
claims.shape
```

# DriverAge

```
threshold = 98
outliers =freqs[freqs['DriverAge'] > threshold]
print('We will be deleating', sum(outliers['ClaimNb']), 'Claim Amounts and a total of', outliers.s
hape[0], 'observed Policies')

freqs = freqs[freqs['DriverAge'] <= threshold]
freqs.shape
```

```
filtersev = list(outliers[outliers['ClaimNb']>0].index.values)
claims = claims.drop(filtersev)
claims.shape
```

# Density

```
freqs['DensityS'].value_counts().sort_index()[-10:]

quants = np.append(np.arange(0.1, 1, 0.1), np.arange(0.95, 0.99, 0.01))
freqs['DensityS'].quantile(q=quants)
```

# Categorical frequency dependency plots

```
def MarginalDPlot(dataset, feature, colors, response = "Empirical Frequency",
                  xlims = None, ylims = None, size = (6,4), measure = "Exposure"):
    fig, ax = plt.subplots(figsize=size, dpi=100)

    # Plot the CO2 variable in blue
    ax.bar(dataset.index, dataset[response], label = response, color = colors)
    ax.grid(True)
    # ax.set_title("Average Claim Amounts and nº of reported Claims for each Driver Age", size = 10)
    ax.set_xlabel(feature, fontsize=10)
    ax.set_ylabel(response, fontsize=10)
    ax.set_ylim(ylims)
    ax.set_xlim(xlims)
    ax.yaxis.label.set_color('g')
    ax.tick_params(axis='y', colors='g')

    # Create a twin Axes that shares the x-axis
    ax2 = ax.twinx()
    # Plot the relative temperature in red
    if measure == "Exposure":
        ylabel = "Exposure Time"
        ycolor = "darkorange"
    else:
        ylabel = "nº Counts"
        ycolor = "steelblue"

    ax2.plot(dataset.index, dataset[measure], label = ylabel, color = ycolor, linestyle = ":", marker = "o", markersize=4)
    ax2.legend(loc = 'best', bbox_to_anchor = (0.5, 0.5, 0.5, 0.5))
    ax2.tick_params(axis='y', colors=ycolor)

MarginalDPlot(dataset = DriverAgeData, colors = ["palegreen"], feature = "DriverAge", ylims = [0, 0.3])

MarginalDPlot(dataset = CarAgeData, colors = ["khaki"], feature = "CarAge", xlims = [0, 40], ylims = [0, 0.2])

clspower = ["palegreen", "coral", "goldenrod", "turquoise", "firebrick",
       "darkorchid", "springgreen", "yellow", "plum", "olive", "mediumblue", "khaki"]
MarginalDPlot(dataset = PowerSmplData, colors = clspower, feature = "Power", ylims = [0, 0.1], size = (7, 3))

clsbrand = ["palegreen", "yellow", "turquoise", "firebrick", "olive", "mediumblue", "khaki"]
MarginalDPlot(dataset = BrandData, colors = clsbrand, feature = "Brand", ylims = [0, 0.1], size = (7, 3))

clsregion = ["goldenrod", "turquoise", "firebrick", "darkorchid", "springgreen", "yellow", "plum", "olive", "mediumblue", "khaki"]
MarginalDPlot(dataset = RegionSmplData, colors = clsregion, feature = "Region", ylims = [0, 0.1], size = (7, 3))
```

# First GLM fitting trial

# Frequencies

```python
# Import statsmodels
import statsmodels.api as sm
from statsmodels.formula.api import glm
from sklearn.model_selection import train_test_split as tts
# seeds = [2802, 2303, 1410, 2105, 2403]
seed = 2802
freqs_train, freqs_test = tts(freqs, test_size = 0.2, random_state = seed)

pd.options.display.float_format = '{:.4f}'.format
print(f'The train set for frequencies has {freqs_train.shape[0]} instances')
freqs_train['ClaimNb'].value_counts(normalize=True)

print(f'The test set for frequencies has {freqs_test.shape[0]} instances')
freqs_test['ClaimNb'].value_counts(normalize=True)

model2 = glm('ClaimNb ~ CarAge + DriverAge', data = freqs_train, family = sm.families.Poisson(), exposure = np.asarray(freqs_train['Exposure'])).fit()

# Print model summary
print(model2.summary())

# Define model formula
formula2 = 'ClaimNb ~ CarAge + DriverAge + C(Gas)'

# Fit GLM
model3 = glm(formula2, data = freqs_train, family = sm.families.Poisson(), exposure = np.asarray(freqs_train['Exposure'])).fit()

# Print model summary
print(model3.summary())

# Define model formula
formula3 = 'ClaimNb ~ CarAge + DriverAge + C(Gas) + C(Brand, Treatment(1))'

# Fit GLM
model4 = glm(formula3, data = freqs_train, family = sm.families.Poisson(), exposure = np.asarray(freqs_train['Exposure'])).fit()

# Print model summary
print(model4.summary())

from patsy import dmatrix
freqs_train_matrix = dmatrix('CarAge + DriverAge + C(Gas) + C(Brand, Treatment(1)) + C(RegionSmpl, Treatment(2)) + C(PowerSmpl) + DensityS', data = freqs_train, return_type = 'dataframe')

from statsmodels.genmod.generalized_linear_model import GLM
# Fit GLM
model7 = GLM(endog = freqs_train['ClaimNb'], exog = freqs_train_matrix, family = sm.families.Poisson(), exposure = freqs_train['Exposure']).fit()

# Print model summary
print(model7.summary())
```

# Testing and predicting Frequencies with the GLM

```python
# The following lists will compile the performance results and descriptions from all chosen models
modeltype = []
modeldesc = []
GEtype = []
```

```
modeltrainGE = []
modeltestGE = []

freqs_test_matrix = dmatrix('CarAge + DriverAge + C(Gas) + C(Brand, Treatment(1)) + C(RegionSmpl, 
Treatment(2)) + C(PowerSmpl) + DensityS', data = freqs_test, return_type = 'dataframe')
freqs_test_matrix.shape

# Custom made function to calculate Poisson Deviance from actual response 'y' and predictions.

def DevianceP(y_i, mu_i):
    D = np.empty(shape = y_i.shape[0])
    for i in range(y_i.shape[0]):
        if y_i[i] == 0:
            D[i] = mu_i[i] - y_i[i]
        else:
            D[i] = y_i[i] * np.log(y_i[i] / mu_i[i]) - (y_i[i] - mu_i[i])

    return(2 * sum(D))

freqs_fitted = model7.predict()
Dpois = DevianceP(freqs_train['ClaimNb'].values, freqs_fitted.values)
GEpoisGLMtrain = Dpois / freqs_train['ClaimNb'].shape[0]

modeltype.append('Frequency')
modeldesc.append('Optimal frequency GLM')
GEtype.append('Poisson')
modeltrainGE.append(round(GEpoisGLMtrain, 5))

print('Based on Train data: Deviance is', round(Dpois,2), 'and Generalization Error is', round(GEp
oisGLMtrain,5))

freqs_pred = model7.predict(exog = freqs_test_matrix, exposure = freqs_test['Exposure'])
Dpois = DevianceP(freqs_test['ClaimNb'].values, freqs_pred.values)
GEpoisGLMtest = Dpois / freqs_test['ClaimNb'].shape[0]

modeltestGE.append(round(GEpoisGLMtest, 5))

print('Based on Test data: Deviance is', round(Dpois,2), 'and Generalization Error is', round(GEpo
isGLMtest,5))
```

# Severities

# Analysis of the reported claims

```
# Here we can obtain information about the distribution of Claim Amounts, and we do so by cheking 
what proportion out of the whole claims record are above a specific threshold
claims1 = claims.copy()
threshold = 10_000
total = freMTPLsev.loc[freMTPLsev['ClaimAmount'] > threshold, 'ClaimAmount'].shape[0]
proportion = total /freMTPLsev.shape[0]
print('The proportion of claim amounts above', threshold, 'is', round(proportion, 5), 'with a tota
l of', total, 'claims')

def Proportion(df, feat, threshold):
    total = df.loc[df[feat] > threshold, feat].shape[0]
    proportion = total /df.shape[0]
    return proportion, total


thresholds = [4000, 10_000, 20_000, 70_000, 50_000, 100_000, 150_000, 200_000, 500_000, 1_000_000]
Props = np.empty(shape = len(thresholds))
Totals = np.empty(shape = len(thresholds))
i = 0
```

```
for thres in thresholds:
    Props[i], Totals[i] = Proportion(df = freMTPLsev, feat = 'ClaimAmount', threshold = thres)
    i += 1

DistrInfo = pd.DataFrame({'Amount Threshold':thresholds, 'Proportion Exceeding':Props, 'Total Exce
eding':Totals.astype(int)})

DistrInfo

threshold = 10_000
Prop, Total = Proportion(df = freMTPLsev, feat = 'ClaimAmount', threshold = threshold)
print('The proportion of claim amounts above', threshold, 'is', round(Prop, 5), 'with a total of',
Total, 'claims')

MarginalDPlot(dataset = DriverAgeData, colors = ["palegreen"], feature = "DriverAge",
              ylims = [0, 2000], response = "Avg. ClaimAmount", measure = "ClaimNb")

MarginalDPlot(dataset = CarAgeData, colors = ["khaki"], feature = "CarAge",
              xlims = [0, 40], ylims = [0, 1_700], response = "Avg. ClaimAmount", measure = "Claim
Nb", size = (5, 4))

clspower = ["palegreen", "coral", "goldenrod", "turquoise", "lightcoral", "khaki", "plum"]
MarginalDPlot(dataset = PowerSmplData, colors = clspower, feature = "PowerSmpl", ylims = [0, 1_700
], size = (7, 3),
              response = "Avg. ClaimAmount", measure = "ClaimNb")

clsbrand = ["palegreen", "coral", "goldenrod", "lightcoral", "khaki"]
MarginalDPlot(dataset = BrandData, colors = clsbrand, feature = "Brand", ylims = [0, 1_600], size
= (7, 3),
              response = "Avg. ClaimAmount", measure = "ClaimNb")

clsbrand = ["palegreen", "coral", "goldenrod", "turquoise", "lightcoral", "khaki", "plum"]
MarginalDPlot(dataset = RegionSmplData, colors = clsbrand, feature = "RegionSmpl", ylims = [0, 1_6
00], size = (7, 3),
              response = "Avg. ClaimAmount", measure = "ClaimNb")

clsgas = ["khaki", "lightcoral"]
MarginalDPlot(dataset = GasData, colors = clsgas, feature = "Gas", ylims = [0, 1_600], size = (5,
3),
              response = "Avg. ClaimAmount", measure = "ClaimNb")

fig, ax = plt.subplots(figsize=(6, 3), dpi=100)

# Plot the CO2 variable in blue
ax.hist(claims1['Frequency'], label = "Frequency distribution", bins = 1000, color = ["turquoise"]
)
ax.grid(True)
# ax.set_title("Average Claim Amounts and nº of reported Claims for each Driver Age", size = 10)
ax.set_xlabel("Frequency", fontsize=10)
ax.set_ylabel("Frequency distribution", fontsize=10)
ax.set_xlim([0, 15])
```

# What are the highest claims? Could they affect the modelling process?

```
print('Frequency is equal or higher that 1 in {:.2%} of instances'.format(claims1[claims1['Frequen
cy']>=1].shape[0] / claims1['Frequency'].shape[0]))
print('Frequency is equal or higher that 1 in {:.2%} of instances'.format(freqs_test[freqs_test['F
requency']>=1].shape[0] / freqs_test['Frequency'].shape[0]))

# %matplotlib widget
%matplotlib inline
import matplotlib.pyplot as plt

# Initalize a Figure and Axes
```

```
fig, ax = plt.subplots(figsize=(8, 4), dpi=100)
# Empirical Frequency
# Plot the CO2 variable in blue
ax.hist(claims['ClaimAmount'], bins = 100, color = "b")
ax.set_xlim([0, 10000])
ax.grid(True)
# ax.set_title("Average Claim Amounts and nº of reported Claims for each Driver Age", size = 10)
ax.set_xlabel("Claim Amount (euros)", fontsize=10)

# Based on the train-test split peformed on freqs table, we must split the claims table accordingl
y to the claim Amounts contained in both resulting parts from freqs.
claims_train = pd.merge(left = claims1['ClaimAmount'], right = freqs_train, how = "inner", left_in
dex = True, right_index = True)
claims_train = claims_train.drop(['ClaimNb','Exposure'], axis = 1)

claims_test = pd.merge(left = claims1['ClaimAmount'], right = freqs_test, how = "inner", left_inde
x = True, right_index = True)
claims_test = claims_test.drop(['ClaimNb','Exposure'], axis = 1)
claims_train.describe()

claims_test.describe()

claims_train.shape[0]/claims1.shape[0] + claims_test.shape[0]/claims1.shape[0]

from statistics import mean
from statistics import variance as var

media = mean(claims1['ClaimAmount'].values)
varianza = var(claims1['ClaimAmount'])
print('The mean equals', media)
print('The variance equals', round(varianza, 3), 'and the standard deviation equals', round(varian
za ** (1/2), 3))
```

# GLM training for claims

```
claims_train_matrix1 = dmatrix('CarAge + DriverAge + Frequency', data = claims_train, return_type
= 'dataframe')
models1 = GLM(endog = claims_train['ClaimAmount'], exog = claims_train_matrix1, family = sm.famili
es.Gamma(link = sm.families.links.log)).fit()
print(models1.summary())
```

# Can we add any more significant variables to the model?

```
freqs_train.loc[freqs_train['ClaimNb']>0,'ClaimNb'].value_counts()

sevF = 'CarAge + C(DriverAgeB, Treatment(1)) + Frequency'
claims_train_matrix2 = dmatrix(sevF, data = claims_train, return_type = 'dataframe')
models2 = GLM(endog = claims_train['ClaimAmount'], exog = claims_train_matrix2, family = sm.famili
es.Gamma(link = sm.families.links.log)).fit()
print(models2.summary())

# sevF = 'CarAge + C(DriverAgeB, Treatment(1)) + PowerSmplT2 + RegionSmplT2 + DensityS + Frequency
'
sevF1 = 'CarAge + C(DriverAgeB, Treatment(1)) + PowerSmplT + RegionSmplT + DensityS'
claims_train_matrix3 = dmatrix(sevF1, data = claims_train, return_type = 'dataframe')
models3 = GLM(endog = claims_train['ClaimAmount'], exog = claims_train_matrix3, family = sm.famili
es.Gamma(link = sm.families.links.log)).fit()
print(models3.summary())
```

# Predicting and testing Claims with the GLM

```
claims_test_matrix = dmatrix(sevF, data = claims_test, return_type = 'dataframe')
claims_test_matrix1 = dmatrix(sevF1, data = claims_test, return_type = 'dataframe')
claims_test_matrix.shape

def DevianceG(y_i, mu_i):
    D = np.empty(shape = y_i.shape[0])
    for i in range(y_i.shape[0]):
            D[i] = - np.log(y_i[i] / mu_i[i]) + (y_i[i] - mu_i[i]) / mu_i[i]
    return(2 * sum(D))
```

# Train set predictions

```
claims_fitted = models2.predict()
Dgamma = DevianceG(claims_train['ClaimAmount'].values, claims_fitted)
GEgammaGLMtrain = Dgamma / claims_train['ClaimAmount'].shape[0]

modeltype.append('Severity')
modeldesc.append('Optimal severity GLM')
GEtype.append('Gamma')
modeltrainGE.append(round(GEgammaGLMtrain, 5))

print('Based on Train data: Deviance is', round(Dgamma, 2), 'and Generalization Error is', round(G
EgammaGLMtrain, 5))

claims_fitted1 = models3.predict()
Dgamma1 = DevianceG(claims_train['ClaimAmount'].values, claims_fitted1)
GEgammaGLMtrain1 = Dgamma1 / claims_train['ClaimAmount'].shape[0]

modeltype.append('Severity')
modeldesc.append('Optimal Alt. severity GLM')
GEtype.append('Gamma')
modeltrainGE.append(round(GEgammaGLMtrain1, 5))

print('Based on Train data: Deviance is', round(Dgamma1,2), 'and Generalization Error is', round(G
EgammaGLMtrain1, 5))

print(f'Average claim amount is {claims_train.ClaimAmount.mean():.6}€')

print(f'Average fitted claim amounts for optimal GLM is {claims_fitted.mean():.6}€')
print(f'Average fitted claim amounts for alternative optimal GLM is {claims_fitted1.mean():.6}€')
```

# Test set predictions

```
claims_pred = models2.predict(exog = claims_test_matrix)
Dgamma = DevianceG(claims_test['ClaimAmount'].values, claims_pred.values)
GEgammaGLMtest = Dgamma / claims_test['ClaimAmount'].shape[0]

modeltestGE.append(round(GEgammaGLMtest, 5))

print('Based on Test data: Deviance is', round(Dgamma,2), 'and Generalization Error is', round(GEg
ammaGLMtest, 5))

claims_pred1 = models3.predict(exog = claims_test_matrix1)
Dgamma1 = DevianceG(claims_test['ClaimAmount'].values, claims_pred1.values)
GEgammaGLMtest1 = Dgamma1 / claims_test['ClaimAmount'].shape[0]

modeltestGE.append(round(GEgammaGLMtest1, 5))

print('Based on Test data: Deviance is', round(Dgamma1,2), 'and Generalization Error is', round(GE
gammaGLMtest1, 5))

print(f'Average claim amount is {claims_test.ClaimAmount.mean():.6}€')

print(f'Average test predicted claim amounts for optimal GLM is {claims_pred.mean():.6}€')
print(f'Average test predicted claim amounts for alternative optimal GLM is {claims_pred1.mean():.
6}€')
```

# **Loss Costing with GLMs**

```
freqs_test1 = freqs_test.copy()

def PricingGLM(test, freqGLM, sevGLM, freqFormula, sevFormula):
    # The test data has to be rearranged into the two different GLM models matrices following thei
r feature specification
```

```
    freq_matrix = dmatrix(freqFormula, data = test, return_type = 'dataframe')
    sev_matrix = dmatrix(sevFormula, data = test, return_type = 'dataframe')

    # The GLMs must make their predictions for both frequencies and severities
    freq_pred = freqGLM.predict(exog = freq_matrix, exposure = test['Exposure'])
    sev_pred = sevGLM.predict(exog = sev_matrix)

    # Once we have the predictions, we can easily compute the pure premium
    pure_premium = freq_pred * sev_pred
    return freq_pred, sev_pred, pure_premium

freqF = 'CarAge + DriverAge + C(Gas) + C(Brand, Treatment(1)) + C(RegionSmpl, Treatment(2)) + C(Po
werSmpl) + DensityS'

# We aggregate the loss incurred by each policy, so that we can join this information with the fre
qs table
AggregatedLoss = claims.groupby(level = 0).sum()['ClaimAmount']
# We initiate the PremiumAnalysis table, were the summary information will be compiled
PremAnalysisGLM = freqs_test1[['ClaimNb', 'Exposure']].copy()
# We perform the joint with PremiumAnalysis as the target
PremAnalysisGLM = pd.merge(PremAnalysisGLM, AggregatedLoss, how = 'left', left_index = True, right
_index = True)
PremAnalysisGLM.fillna(0, inplace = True)
PremAnalysisGLM = PremAnalysisGLM.rename(columns = {'ClaimAmount':'Aggregated Loss'})
PremAnalysisGLM['Pred. Frequency'], PremAnalysisGLM['Pred. Severity'], PremAnalysisGLM['Pure Premi
um'] = PricingGLM(freqs_test1, freqGLM = model7, sevGLM = models3, freqFormula = freqF, sevFormula
= sevF1)
```

# Premium Analysis table

```
PremAnalysisGLM['Anual Pure Prem.'] = PremAnalysisGLM['Pure Premium'] / PremAnalysisGLM['Exposure'
]
PremAnalysisGLM.iloc[15:23]

PremAnalysisGLM.describe()
```

# Single profile prediction

```
original_feat = list(freqs0.columns)
del(original_feat[0:1])
del(original_feat[-3:])
print(original_feat)

seed = 2303
freqs_test1.iloc[[seed]][original_feat]

PremAnalysisGLM.iloc[[seed]][['Pred. Frequency', 'Pred. Severity', 'Pure Premium', 'Anual Pure Pre
m.']]
```

# Gradient Boosting Model Fitting

# GBRT for Frequencies

```
# There are some basic imports that are worth having at our disposal
from datetime import datetime
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold
import seaborn as sns
import lightgbm as lgb

from pdpbox import pdp, get_dataset, info_plots

# Only the original set of features will be tested in the first LightGBM example, this configurati
on can be rearranged afterwards
initial_conf = ['ClaimNb', 'Exposure', 'PowerT', 'CarAge', 'DriverAge', 'BrandT', 'GasT', 'RegionT
', 'DensityS']
advanced_conf = ['ClaimNb', 'Exposure', 'PowerSmplT', 'CarAge', 'DriverAge', 'BrandT', 'GasT', 'Re
gionSmplT', 'DensityS']
# We will apply this feature configuration to the train set, and if we decide to finally test the
model, then we will have to do the same in the test set
freqs_train2 = freqs_train.copy()
freqs_train2 = freqs_train2[initial_conf]

# The implementation of GBM models requires arranging the data in the following structure:
freqs_train2_y = freqs_train2['ClaimNb'].values / freqs_train2['Exposure'].values
freqs_train2_w = freqs_train2['Exposure'].values
freqs_train2_X = freqs_train2.drop(['ClaimNb', 'Exposure'], axis = 1)
#LightGBM requieres to convert our categoricals into integers that is why we select the versions o
f categoricals ending in T

freqs_train2_y.shape

# We will use the sklearn API for lightGBM for better compatibility with related tools
seed = 1008
start=datetime.now()
GBMmodel = lgb.LGBMRegressor(max_depth = 4, learning_rate = 0.05, n_estimators = 200,
                             objective = "poisson", min_child_samples = 1000,
                             importance_type = "gain", random_state = seed)
GBMmodel.fit(freqs_train2_X, freqs_train2_y, sample_weight = freqs_train2_w)
stop=datetime.now()
execution_time_lgbm = stop-start

# Sample weights are correctly taken into account

print("LightGBM execution time is: ", execution_time_lgbm)

GBMmodel.feature_importances_
```

# Predicting with LightGBM for frequecies

```
freqs_test2 = freqs_test.copy()
freqs_test2 = freqs_test2[initial_conf]
freqs_test2_y = freqs_test2['ClaimNb'].values
freqs_test2_w = freqs_test2['Exposure'].values
freqs_test2_X = freqs_test2.drop(['ClaimNb', 'Exposure'], axis = 1)
# freqs_test2_X = freqs_test2_X.replace(cat_to_int)
#, weight = freqs_train2_w
fitted_y = GBMmodel.predict(freqs_train2_X)
pred_y = GBMmodel.predict(freqs_test2_X)

freqs_test2_X.info()

av_em_frq = sum(freqs_train['ClaimNb']) / sum(freqs_train['Exposure'])
print(av_em_frq)
print(mean(freqs_train['ClaimNb']))

# A table for the comparison between predictions both from GLM and for GBM models in the test sets
:
freqs_test_results = freqs_test2[['ClaimNb', 'Exposure']]
freqs_test_results['GLM prediction'] = freqs_pred
freqs_test_results['GBM prediction'] = pred_y * freqs_test2_w
# The same table but for fitted values from the test set:
freqs_train_results = freqs_train2[['ClaimNb', 'Exposure']]
freqs_train_results['GLM prediction'] = freqs_fitted
freqs_train_results['GBM prediction'] = fitted_y * freqs_train2_w
```

# Ratios for Train set

```
av_em_frq = sum(freqs_train_results['ClaimNb']) / sum(freqs_train_results['Exposure'])
#av_em_frq = mean(freqs_train_results['ClaimNb'] / freqs_train_results['Exposure'])
print('Average empirical frequency', round(av_em_frq, 6))
av_GLM_frq = sum(freqs_train_results['GLM prediction']) / sum(freqs_train_results['Exposure'])
print('Average GLM frequency', round(av_GLM_frq, 6))
```

```
av_GBM_frq = sum(freqs_train_results['GBM prediction']) / sum(freqs_train_results['Exposure'])
print('Average GBM frequency', round(av_GBM_frq, 6))

print(mean(freqs_test_results['Exposure']))
print(av_GBM_frq/av_GLM_frq)
```

# Ratios for Test set

```
av_em_frq = sum(freqs_test_results['ClaimNb']) / sum(freqs_test_results['Exposure'])
#av_em_frq = mean(freqs_train_results['ClaimNb'] / freqs_train_results['Exposure'])
print('Average empirical frequency', round(av_em_frq, 4))
av_GLM_frq = sum(freqs_test_results['GLM prediction']) / sum(freqs_test_results['Exposure'])
print('Average GLM frequency', round(av_GLM_frq, 6))
av_GBM_frq = sum(freqs_test_results['GBM prediction']) / sum(freqs_test_results['Exposure'])
print('Average GBM frequency', round(av_GBM_frq, 6))
```

# Prediction comparison table between GLM and GBM model

```
freqs_train_results.sum()

freqs_test_results.sum()
```

# Performance results

```
Dpois = DevianceP(freqs_train_results['ClaimNb'].values, freqs_train_results['GBM prediction'].val
ues)
GEpois = Dpois / freqs_train_results['ClaimNb'].shape[0]
print('Based on Train data: Deviance is', round(Dpois,2), 'and Generalization Error is', round(GEp
ois,5))

Dpois = DevianceP(freqs_test_results['ClaimNb'].values, freqs_test_results['GBM prediction'].value
s)
GEpois = Dpois / freqs_test_results['ClaimNb'].shape[0]
print('Based on Test data: Deviance is', round(Dpois,2), 'and Generalization Error is', round(GEpo
is,5))
```

# Feature importance

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# sorted(zip(clf.feature_importances_, X.columns), reverse=True)
importances = GBMmodel.feature_importances_ / max(GBMmodel.feature_importances_)
feature_imp = pd.DataFrame(sorted(zip(importances, GBMmodel.feature_name_)),
columns=['Value','Feature'])

plt.figure(figsize=(10, 5))
plt.grid(True)
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value", ascending=False))

plt.title('LightGBM Features')
plt.xlabel('Percentage (%) of total deviance reduction')
plt.tight_layout()
plt.show()
```

# Basic Tree visualization

```
import os
os.environ["PATH"] += os.pathsep + 'C:/ProgramData/Anaconda3/Library/bin/graphviz/'

lgb.plot_tree(GBMmodel, tree_index = 0, show_info = ["internal_value", "leaf_count", "split_gain"]
, figsize=(10, 6), dpi = 150, orientation = 'horizontal')

np.exp(-2.685)
```

```
lgb.plot_split_value_histogram(GBMmodel, feature = "DriverAge", width_coef = 0.8)
plt.title('DriverAge value-split counts')

lgb.plot_split_value_histogram(GBMmodel, feature = "DensityS", width_coef = 2)
plt.title('DensityS value-split counts')
```

# Quick GBM training and performance evaluation

```
# freqs_train2_X['Rand'] = [random.random() for i in range(freqs_train2_X.shape[0])]
start=datetime.now()
GBMmodel0 = lgb.LGBMRegressor(max_depth = 5, learning_rate = 0.05, n_estimators = 200,
                             objective = "poisson", min_child_samples = 1000,
                             feature_fraction = 0.6, importance_type = "gain", random_state = seed
)
GBMmodel0.fit(freqs_train2_X, freqs_train2_y, sample_weight = freqs_train2_w)

fitted_y = GBMmodel0.predict(freqs_train2_X)
pred_y = GBMmodel0.predict(freqs_test2_X)

freqs_train_results['GBM prediction'] = fitted_y * freqs_train2_w
freqs_test_results['GBM prediction'] = pred_y * freqs_test2_w

Dpoistrain = DevianceP(freqs_train_results['ClaimNb'].values, freqs_train_results['GBM prediction'
].values)
GEpoisGBMtrain = Dpoistrain / freqs_train_results['ClaimNb'].shape[0]
Dpoistest = DevianceP(freqs_test_results['ClaimNb'].values, freqs_test_results['GBM prediction'].v
alues)
GEpoisGBMtest = Dpoistest / freqs_test_results['ClaimNb'].shape[0]

end=datetime.now()
runtime = end - start
print("total run time was:", runtime)
print('Based on Train data: Deviance is', round(Dpoistrain,2), 'and Generalization Error is', roun
d(GEpoisGBMtrain,5))
print('Based on Test data: Deviance is', round(Dpoistest,2), 'and Generalization Error is', round(
GEpoisGBMtest,5))

modeltype.append('Frequency')
modeldesc.append('Fine-tunned frequency GBM')
GEtype.append('Poisson')
modeltrainGE.append(round(GEpoisGBMtrain, 5))
modeltestGE.append(round(GEpoisGBMtest, 5))

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# sorted(zip(clf.feature_importances_, X.columns), reverse=True)
importances = GBMmodel0.feature_importances_ / max(GBMmodel0.feature_importances_)
feature_imp = pd.DataFrame(sorted(zip(importances, GBMmodel0.feature_name_)), columns=['Value','Fe
ature'])

plt.figure(figsize=(10, 5))
plt.grid(True)
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value", ascending=False))

plt.title('LightGBM Features')
plt.xlabel('Percentage (%) of total deviance reduction')
plt.tight_layout()
plt.show()

lgb.plot_split_value_histogram(GBMmodel0, feature = "CarAge", width_coef = 1)
```

# Cross Validation through early stopping

```
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
```

```
%%capture
from sklearn.model_selection import KFold

GBMmodelCV = lgb.LGBMRegressor(max_depth = 5, learning_rate = 0.05, n_estimators = 200,
                              objective = "poisson", min_child_samples = 1000,
                              feature_fraction = 0.4, importance_type = "gain", random_state = seed
)
kf = KFold(n_splits=5, shuffle=True)
GEpoisCV = []
predicts = []
start=datetime.now()
for train_index, test_index in kf.split(freqs_train2_X, freqs_train2_y):
    print("###")
    X_train, X_val = freqs_train2_X.iloc[train_index], freqs_train2_X.iloc[test_index]
    y_train, y_val = freqs_train2_y[train_index], freqs_train2_y[test_index]
    w_train, w_val = freqs_train2_w[train_index], freqs_train2_w[test_index]
    GBMmodelCV.fit(X_train, y_train, sample_weight = w_train, eval_set=[(X_val, y_val)],
            early_stopping_rounds=25)
    eval_pred = GBMmodelCV.predict(X_val) * w_val
    y_val = y_val * w_val
    Dpois = DevianceP(y_val, eval_pred)
    GEpoisCV.append(Dpois / y_val.shape[0])
    predicts.append(GBMmodelCV.predict(freqs_test2_X))

end=datetime.now()
runtime = end - start
print("total run time was:", runtime.total_seconds())

print("total run time for GBM with 5-CV was:", runtime)
print("average eval GE is:", round(mean(GEpoisCV), 5))
# Hyper-parameter tunning for GBM frequencies

# The dictionary helps to compile the different values for the hyperparameters that we will be tes
ting
params_dt = {
    'learning_rate': [0.05, 0.02, 0.01, 0.005],
    'n_estimators': [200, 500, 700, 1000],
    'max_depth': [4, 5, 6]
}

# Now we will define the function that will do the whole process of K-fold cross-validation for a
GBM model with
# a different set of hyperparameters, all of it based on our training information
def HyperpTrialGBM(nfolds, X, y, sample_weights, l_rate = 0.01, n_est = 1000, max_d = 5, frac_ft =
0.6):

    seed = 2802
    GBMmodel1 = lgb.LGBMRegressor(max_depth = max_d, learning_rate = l_rate, n_estimators = n_est,
                              objective = 'poisson', min_child_samples = 1000,
                              feature_fraction = frac_ft, importance_type = "gain", random_state =
seed)

    kf = KFold(n_splits=nfolds, shuffle=False)
    GEpoisCV = []
    start=datetime.now()

    for train_index, test_index in kf.split(X, y):

        X_train, X_val = X.iloc[train_index], X.iloc[test_index]
        y_train, y_val = y[train_index], y[test_index]
        w_train, w_val = sample_weights[train_index], sample_weights[test_index]

        GBMmodel1.fit(X_train, y_train, sample_weight = w_train,
                      eval_set=[(X_val, y_val)], early_stopping_rounds=20)
        eval_pred = GBMmodel1.predict(X_val) * w_val
        y_val = y_val * w_val
        Dpois = DevianceP(y_val, eval_pred)
        GEpoisCV.append(Dpois / y_val.shape[0])

    GEpoisCVav = mean(GEpoisCV)
    end=datetime.now()
    runtime = end - start

    return GEpoisCVav, runtime

# %%capture
# # Initiate the list for Generalization Error, and then also for run time
# GEs = []
# runtimes = []
# l_rate = []
# n_estimators = []
# max_d = []

# for i in params_dt['learning_rate']:
#     for j in params_dt['n_estimators']:
#         for l in params_dt['max_depth']:
#             GE, time = HyperpTrialGBM(nfolds = 5, X = freqs_train2_X, y = freqs_train2_y,
#                                       sample_weights = freqs_train2_w, l_rate = i, n_est = j, ma
x_d = l)
#             GEs.append(GE)
#             runtimes.append(time)
#             l_rate.append(i)
#             n_estimators.append(j)
#             max_d.append(l)

# runtimes0 = runtimes.copy()
# runtimes1 = [round(time.total_seconds(),2) for time in runtimes0]

# TunningResults = pd.DataFrame({'Learning Rate':l_rate, 'nº Estimators':n_estimators, 'Max. Depth
':max_d,
#                                'Validation GE':GEs, 'Run Time (secs.)':runtimes1})
# TunningResults = TunningResults.set_index(['Learning Rate', 'nº Estimators', 'Max. Depth'])

# TunningResults.head(12)

# print('Total runtime of grid search was {}'.format(round(TunningResults['Run Time (secs.)'].sum(
)/60, 2)), 'minutes')

# TunningResults.sort_values(by = "Validation GE").head()

# Optimal = TunningResults['Validation GE'].idxmin(axis = 1)
# Chosen = TunningResults.sort_values(by = "Validation GE").index[1]
# TunningResults['Validation GE'].min()
# print(Chosen)

# print('The best performing GBM model has learning rate', Optimal[0],
#       ', nº of trees', Optimal[1], 'and maximum depth', Optimal[2])
# print('The second best GBM model will be that with learning rate', Chosen[0], ', nº of trees', C
hosen[1],
#       'and maximum depth', Chosen[2])
```

# Fine-tunning for Stochastic GBM

```
# params_dt = {
#     'feature_fraction': [0.4, 0.6, 0.8, 1]
# }

# %%capture
# GEs = []
# runtimes = []

# for i in params_dt['feature_fraction']:
#     GE, time = HyperpTrialGBM(nfolds = 5, X = freqs_train2_X, y = freqs_train2_y,
#                                    sample_weights = freqs_train2_w, l_rate = Chosen[0],
#                                    n_est = Chosen[1], max_d = Chosen[2], frac_ft = i)
#     GEs.append(GE)
#     runtimes.append(time)

# for i in range(len(params_dt['feature_fraction'])):
#     print("Total run time for optimal GBM with feature_fraction = {} was:".format(params_dt['fea
ture_fraction'][i]), runtimes[i], "while eval GE is:", round(GEs[i], 6))
# print("")
# for i in range(len(params_dt['feature_fraction'])):
#     print("Total run time for second optimal GBM with feature_fraction = {} was:".format(params_
dt['feature_fraction'][i]), runtimes[i+4], "while eval GE is:", round(GEs[i+4], 7))

# runtimes1 = [round(time.total_seconds(),2) for time in runtimes]
# sTunningResults = pd.DataFrame({'Feature fraction':params_dt['feature_fraction'],
#                                 'Validation GE':GEs, 'Run Time (secs.)':runtimes1})
# sTunningResults = sTunningResults.set_index(['Feature fraction'])

# sTunningResults
```

# Partial Dependency Plots

```
from pdpbox import pdp, get_dataset, info_plots

params = {
    # plot title and subtitle
    # matplotlib color map for ICE lines
#     'linestyle': ":",
    'line_cmap': 'Blues',
    # pdp line color, highlight color and line width
    'pdp_color': '#1A4E5D',
    'pdp_hl_color': '#FEDC00',
    'pdp_linewidth': 2,
    # horizon zero line color and with
    'zero_color': '#E75438',
    'zero_linewidth': 1,
    # pdp std fill color and alpha
    'fill_color': '#66C2D7',
    'fill_alpha': 0.1,
    # marker size for pdp line
    'markersize': 6

}

def PDPlot(model, feature, trainset, xlims = None, ylims = None, n_grid = 25):
    pdp_dist = pdp.pdp_isolate(model=model, dataset=trainset
                               , model_features=trainset.columns, num_grid_points = n_grid
                               , feature=feature)

    fig, axes = pdp.pdp_plot(pdp_dist, feature, center = False, plot_params = params, figsize = (8
, 6))
    axes['pdp_ax'].set_ylim(ylims)
    axes['pdp_ax'].set_xlim(xlims)
    axes['pdp_ax'].set_xlabel("", fontsize=15)
    axes['pdp_ax'].set_title(f'PDP for feature {feature}', fontsize=15)
    axes['pdp_ax'].tick_params(axis='both',  labelsize=15)

# PDPdriverage = PDPlot(GBMmodel0, 'DriverAge', freqs_train2_X, ylims = [0.04, 0.30])

# PDPcarage = PDPlot(GBMmodel0, 'CarAge', freqs_train2_X, ylims = [0.03, 0.12], xlims = [0, 25])
# PDPcarage = PDPlot(GBMmodel0, 'DensityS', freqs_train2_X, ylims = [0.04, 0.13], n_grid = 20)

# start = datetime.now()
# PDPcarage = PDPlot(GBMmodel0, 'DensityS', freqs_train2_X, ylims = [0.04, 0.13], xlims = [0, 5],
n_grid = 20)
# end = datetime.now()
# time = end - start
# print('Total runtime was {}'.format(time))

# pdp_dist1 = pdp.pdp_isolate(model=GBMmodel0, dataset=freqs_train2_X
#                             , model_features=freqs_train2_X.columns, num_grid_points = 7, grid_ty
pe = 'equal'
#                             , feature='BrandT')

# fig, axes = pdp.pdp_plot(pdp_dist1, 'BrandT', center = False, plot_params = params, figsize = (8
, 6))
# axes['pdp_ax'].set_ylim([0.04, 0.12])
# axes['pdp_ax'].set_xlabel("", fontsize=15)
# axes['pdp_ax'].set_title('PDP for feature Brand with Target Encoding', fontsize=15)
# axes['pdp_ax'].set_xticklabels(labels = list(Encodings['RegionT'].keys()))
# axes['pdp_ax'].tick_params(axis='both',  labelsize=15)

# pdp_dist2 = pdp.pdp_isolate(model=GBMmodel0, dataset=freqs_train2_X
#                             , model_features=freqs_train2_X.columns, num_grid_points = 12, grid_t
ype = 'equal'
#                             , feature='PowerT')

# import matplotlib.ticker as ticker
# fig, axes = pdp.pdp_plot(pdp_dist2, 'PowerT', center = False, plot_params = params, figsize = (8
, 6))
# axes['pdp_ax'].set_ylim([0.03, 0.15])
# axes['pdp_ax'].set_xlabel("", fontsize=15)
# axes['pdp_ax'].set_title('PDP for feature Power with Target Encoding', fontsize=15)
# axes['pdp_ax'].xaxis.set_ticks(np.arange(0, 12, 1))
# axes['pdp_ax'].set_xticklabels(labels = list(Encodings['PowerT'].keys()))

# axes['pdp_ax'].tick_params(axis='both',  labelsize=15)

# pdp_dist3 = pdp.pdp_isolate(model=GBMmodel0, dataset=freqs_train2_X
#                             , model_features=freqs_train2_X.columns, num_grid_points = 2, grid_ty
pe = 'equal'
#                             , feature='GasT')

# fig, axes = pdp.pdp_plot(pdp_dist3, 'GasT', center = False, plot_params = params, figsize = (8,
6))
# axes['pdp_ax'].set_ylim([0.04, 0.12])
# axes['pdp_ax'].set_xlim([-0.3, 1.3])
# axes['pdp_ax'].set_xlabel("", fontsize=15)
# axes['pdp_ax'].set_title('PDP for feature Gas with Target Encoding', fontsize=15)
# # axes['pdp_ax'].xaxis.set_ticks(np.arange(0, 12, 1))
# axes['pdp_ax'].set_xticklabels(labels = list(Encodings['GasT'].keys()))

# axes['pdp_ax'].tick_params(axis='both',  labelsize=15)

# pdp_dist4 = pdp.pdp_isolate(model=GBMmodel0, dataset=freqs_train2_X
#                             , model_features=freqs_train2_X.columns, num_grid_points = 10, grid_t
ype = 'equal'
#                             , feature='RegionT')

# import matplotlib.ticker as ticker
# fig, axes = pdp.pdp_plot(pdp_dist4, 'RegionT', center = False, plot_params = params, figsize = (
8, 6))
# axes['pdp_ax'].set_ylim([0.03, 0.12])
# axes['pdp_ax'].set_xlabel("", fontsize=15)
# axes['pdp_ax'].set_title('PDP for feature Region with Target Encoding', fontsize=15)
# axes['pdp_ax'].xaxis.set_ticks(np.arange(0, 10, 1))
# axes['pdp_ax'].set_xticklabels(labels = list(Encodings['RegionT'].keys()))

# axes['pdp_ax'].tick_params(axis='both',  labelsize=15)
```

# GBRT for Claim Amounts

```
advanced_conf = ['ClaimAmount', 'CarAge', 'DriverAge', 'PowerSmplT2', 'RegionSmplT2', 'DensityS',
'Frequency']
claims_train2 = claims_train[advanced_conf]
claims_test2 = claims_test[advanced_conf]
import random
claims_train2['Rand'] = [random.random() for i in range(claims_train2.shape[0])]

claims_train2.describe()

# The implementation of GBM models requires arranging the data in the following structure:
claims_train2_y = claims_train2['ClaimAmount'].values
claims_train2_X = claims_train2.drop(['ClaimAmount'], axis = 1)
#LightGBM requieres to convert our categoricals into integers that is why we select the versions o
f categoricals ending in T

claims_train2_X.shape

# We will use the sklearn API for lightGBM for better compatibility with related tools
seed = 2303
start=datetime.now()
GBMmodelS0 = lgb.LGBMRegressor(max_depth = 2, learning_rate = 0.02, n_estimators = 500,
                              objective = "gamma", min_child_samples = 100,
                              importance_type = "gain", random_state = seed)
GBMmodelS0.fit(claims_train2_X, claims_train2_y)
stop=datetime.now()
execution_time_lgbm = stop-start

# Sample weights are correctly taken into account
print("LightGBM execution time is: ", execution_time_lgbm)

lgb.plot_tree(GBMmodelS0, tree_index = 0, show_info = ["internal_value", "leaf_count"], figsize=(8
, 4), dpi = 100, orientation = 'horizontal')

# sorted(zip(clf.feature_importances_, X.columns), reverse=True)
importances = GBMmodelS0.feature_importances_ / max(GBMmodelS0.feature_importances_)
feature_imp = pd.DataFrame(sorted(zip(importances, GBMmodelS0.feature_name_)), columns=['Value','F
eature'])

plt.figure(figsize=(10, 5))
plt.grid(True)
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value", ascending=False))

plt.title('LightGBM Features')
plt.xlabel('Percentage (%) of total deviance reduction')
plt.tight_layout()
plt.show()

last_conf = ['ClaimAmount', 'CarAge', 'DriverAge', 'DensityS', 'Frequency']
claims_train2 = claims_train[last_conf]
claims_test2 = claims_test[last_conf]

# The implementation of GBM models requires arranging the data in the following structure:
claims_train2_y = claims_train2['ClaimAmount'].values
claims_train2_X = claims_train2.drop(['ClaimAmount'], axis = 1)
#LightGBM requieres to convert our categoricals into integers that is why we select the versions o
f categoricals ending in T
```

```
claims_train2_X
```

# Quick GBM fitting

```
start=datetime.now()
seeed = 1410
GBMmodelS = lgb.LGBMRegressor(max_depth = 1, learning_rate = 0.02, n_estimators = 500,
                             objective = "gamma", feature_fraction = 0.8,
                             importance_type = "gain", random_state = seed)
GBMmodelS.fit(claims_train2_X, claims_train2_y)
stop=datetime.now()
execution_time_lgbm = stop-start

# sorted(zip(clf.feature_importances_, X.columns), reverse=True)
importances = GBMmodelS.feature_importances_ / max(GBMmodelS.feature_importances_)
feature_imp = pd.DataFrame(sorted(zip(importances, GBMmodelS.feature_name_)), columns=['Value','Feature'])

plt.figure(figsize=(10, 5))
plt.grid(True)
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value", ascending=False))

plt.title('LightGBM Features')
plt.xlabel('Percentage (%) of total deviance reduction')
plt.tight_layout()
plt.show()

lgb.plot_tree(GBMmodelS, tree_index = 0, show_info = ["internal_value", "leaf_count"], figsize=(4, 2), dpi = 100, orientation = 'horizontal')
lgb.plot_tree(GBMmodelS, tree_index = 498, show_info = ["internal_value", "leaf_count"], figsize=(4, 2), dpi = 100, orientation = 'horizontal')
lgb.plot_tree(GBMmodelS, tree_index = 1, show_info = ["internal_value", "leaf_count"], figsize=(4, 2), dpi = 100, orientation = 'horizontal')
lgb.plot_tree(GBMmodelS, tree_index = 499, show_info = ["internal_value", "leaf_count"], figsize=(4, 2), dpi = 100, orientation = 'horizontal')
```

# Alternative GBM fitting for claim amounts

```
last_conf1 = ['ClaimAmount', 'CarAge', 'DriverAge', 'DensityS']
claims_train3 = claims_train[last_conf1]
claims_test3 = claims_test[last_conf1]

claims_train3_y = claims_train3['ClaimAmount'].values
claims_train3_X = claims_train3.drop(['ClaimAmount'], axis = 1)

start=datetime.now()
seeed = 1410
GBMmodelS1 = lgb.LGBMRegressor(max_depth = 1, learning_rate = 0.02, n_estimators = 500,
                             objective = "gamma", feature_fraction = 1,
                             importance_type = "gain", random_state = seed)
GBMmodelS1.fit(claims_train3_X, claims_train3_y)
stop=datetime.now()
execution_time_lgbm = stop-start
```

# Predicting with LightGBM for severities

```
claims_test2_y = claims_test2['ClaimAmount'].values
claims_test2_X = claims_test2.drop(['ClaimAmount'], axis = 1)

fitted_y = GBMmodelS.predict(claims_train2_X)
pred_y = GBMmodelS.predict(claims_test2_X)

# A table for the comparison between predictions both from GLM and for GBM models in the test sets :
```

```
claims_test_results = claims_test2[['ClaimAmount']]
claims_test_results['GLM prediction'] = claims_pred
claims_test_results['GBM prediction'] = pred_y
# The same table but for fitted values from the test set:
claims_train_results = claims_train2[['ClaimAmount']]
claims_train_results['GLM prediction'] = claims_fitted
claims_train_results['GBM prediction'] = fitted_y

claims_train_results.sum()
# Performance results

Dgamma = DevianceG(claims_train_results['ClaimAmount'].values, claims_train_results['GBM prediction'].values)
GEgammaGBMtrain = Dgamma / claims_train_results['ClaimAmount'].shape[0]

modeltype.append('Severity')
modeldesc.append('Fine-tunned severity GBM')
GEtype.append('Gamma')
modeltrainGE.append(round(GEgammaGBMtrain, 5))

print('Based on Train data: Deviance is', round(Dgamma,2), 'and Generalization Error is', round(GEgammaGBMtrain, 5))

Dgamma = DevianceG(claims_test_results['ClaimAmount'].values, claims_test_results['GBM prediction'].values)
GEgammaGBMtest = Dgamma / claims_test_results['ClaimAmount'].shape[0]

modeltestGE.append(round(GEgammaGBMtest, 5))

print('Based on Test data: Deviance is', round(Dgamma,2), 'and Generalization Error is', round(GEgammaGBMtest, 5))
# Performance results for alternative model

claims_test3_y = claims_test3['ClaimAmount'].values
claims_test3_X = claims_test3.drop(['ClaimAmount'], axis = 1)

fitted_y = GBMmodelS1.predict(claims_train3_X)
pred_y = GBMmodelS1.predict(claims_test3_X)

Dgamma = DevianceG(claims_train_results['ClaimAmount'].values, fitted_y)
GEgammaGBMtrain = Dgamma / claims_train_results['ClaimAmount'].shape[0]

modeltype.append('Severity')
modeldesc.append('Fine-tunned alt. severity GBM')
GEtype.append('Gamma')
modeltrainGE.append(round(GEgammaGBMtrain, 5))

print('Based on Train data: Deviance is', round(Dgamma,2), 'and Generalization Error is', round(GEgammaGBMtrain, 5))

Dgamma = DevianceG(claims_test_results['ClaimAmount'].values, pred_y)
GEgammaGBMtest = Dgamma / claims_test_results['ClaimAmount'].shape[0]

modeltestGE.append(round(GEgammaGBMtest, 5))

print('Based on Test data: Deviance is', round(Dgamma,2), 'and Generalization Error is', round(GEgammaGBMtest, 5))
# Hyper-parameter tunning for GBM severities

# The dictionary helps to compile the different values for the hyperparameters that we will be testing
params_dt = {
    'learning_rate': [0.05, 0.02, 0.01],
    'n_estimators': [200, 500, 700],
    'max_depth': [1, 2, 3]
}

# Now we will define the function that will do the whole process of K-fold cross-validation for a
GBM model with
# a different set of hyperparameters, all of it based on our training information
def HyperpTrialGBM(nfolds, X, y, l_rate = 0.01, n_est = 1000, max_d = 5, frac_ft = None):

    seed = 2802
    GBMmodel = lgb.LGBMRegressor(max_depth = max_d, learning_rate = l_rate, n_estimators = n_est,
                                 objective = 'gamma', random_state = seed,
                                 feature_fraction = frac_ft, importance_type = "gain")

    kf = KFold(n_splits=nfolds, shuffle=False)
    GEgammaCV = []
    start=datetime.now()

    for train_index, test_index in kf.split(X, y):

        X_train, X_val = X.iloc[train_index], X.iloc[test_index]
        y_train, y_val = y[train_index], y[test_index]

        GBMmodel.fit(X_train, y_train,
                     eval_set=[(X_val, y_val)], early_stopping_rounds=20)
        eval_pred = GBMmodel.predict(X_val)
        Dpois = DevianceG(y_val, eval_pred)
        GEgammaCV.append(Dpois / y_val.shape[0])

    GEgammaCVav = mean(GEgammaCV)
    end=datetime.now()
    runtime = end - start

    return GEgammaCVav, runtime

%%capture

# Initiate the list for Generalization Error, and then also for run time
GEs = []
runtimes = []
l_rate = []
n_estimators = []
max_d = []

for i in params_dt['learning_rate']:
    for j in params_dt['n_estimators']:
        for l in params_dt['max_depth']:
            GE, time = HyperpTrialGBM(nfolds = 5, X = claims_train2_X, y = claims_train2_y,
                                      l_rate = i, n_est = j, max_d = l)
            GEs.append(GE)
            runtimes.append(round(time.total_seconds(), 2))
            l_rate.append(i)
            n_estimators.append(j)
            max_d.append(l)

TunningResults = pd.DataFrame({'Learning Rate':l_rate, 'nº Estimators':n_estimators, 'Max. Depth':
max_d,
                               'Validation GE':GEs, 'Run Time (secs.)':runtimes})
TunningResults = TunningResults.set_index(['Learning Rate', 'nº Estimators', 'Max. Depth'])
TunningResults.head()

TunningResults['Run Time (secs.)'].sum()

TunningResults.sort_values(by = "Validation GE").head(3)

Optimal = TunningResults['Validation GE'].idxmin(axis = 1)
Worst = TunningResults['Validation GE'].idxmax(axis = 1)
```

```
print('The best performing GBM model has learning rate', Optimal[0],
      ', nº of trees', Optimal[1], 'and maximum depth', Optimal[2])
print('The worst performing GBM model has learning rate', Worst[0],
      ', nº of trees', Worst[1], 'and maximum depth', Worst[2])
```

# Fine-tunning for Stochastic GBM

```
%%capture
params_dt = {
    'feature_fraction': [0.3, 0.6, 0.8, 1]
}


GEs = []
runtimes = []

for i in params_dt['feature_fraction']:
    GE, time = HyperpTrialGBM(nfolds = 5, X = claims_train2_X, y = claims_train2_y,
                            l_rate = Optimal[0],
                            n_est = Optimal[1], max_d = Optimal[2], frac_ft = i)
    GEs.append(GE)
    runtimes.append(round(time.total_seconds(), 2))

for i in range(len(params_dt['feature_fraction'])):
    print("Total run time for optimal GBM with feature_fraction = {} was:".format(params_dt['featu
re_fraction'][i]), runtimes[i], "while eval GE is:", round(GEs[i], 6))
pd.options.display.float_format = '{:,.2f}'.format
sTunningResults = pd.DataFrame({'Feature fraction':params_dt['feature_fraction'],
                               'Validation GE':GEs, 'Run Time (secs.)':runtimes})
sTunningResults = sTunningResults.set_index(['Feature fraction'])
sTunningResults['Validation GE'] = sTunningResults['Validation GE'].map('{:,.6f}'.format)

sTunningResults
```

# Partial Dependency Plots

```
def PDPlot(model, feature, trainset, xlims = None, ylims = None, n_grid = 25):
    pdp_dist = pdp.pdp_isolate(model=model, dataset=trainset
                               , model_features=trainset.columns, num_grid_points = n_grid
                               , feature=feature)

    fig, axes = pdp.pdp_plot(pdp_dist, feature, center = False, plot_params = params, figsize = (8
, 6))
    axes['pdp_ax'].set_ylim(ylims)
    axes['pdp_ax'].set_xlim(xlims)
    axes['pdp_ax'].set_xlabel("", fontsize=15)
    axes['pdp_ax'].set_title(f'PDP for feature {feature}', fontsize=15)
    axes['pdp_ax'].tick_params(axis='both',  labelsize=15)

PDPdriverage = PDPlot(GBMmodelS, 'DriverAge', claims_train2_X, ylims = [1100, 1500])
PDPcarage = PDPlot(GBMmodelS, 'CarAge', claims_train2_X, ylims = [1000, 1500])
PDPdensityS = PDPlot(GBMmodelS, 'DensityS', claims_train2_X, ylims = [1100, 1500])
PDPdensityS = PDPlot(GBMmodelS, 'Frequency', claims_train2_X, ylims = [1100, 1600], xlims = [0, 10
], n_grid = 29)
```

# Loss Costing with GBM models

```
freqs_test.shape

def PricingGBM(test, weights, freqGBM, sevGBM):

    # The GBMs must make their predictions for both frequencies and severities
    testF = test[freqGBM.feature_name_]
    freq_pred = freqGBM.predict(testF)
    freq_pred = freq_pred * weights

    testS = test[sevGBM.feature_name_]
    sev_pred = sevGBM.predict(testS)
    # Once we have the predictions, we can easily compute the pure premium
    pure_premium = freq_pred * sev_pred
    return freq_pred, sev_pred, pure_premium

# We aggregate the loss incurred by each policy, so that we can join this information with the fre
qs table
AggregatedLoss = claims1.groupby(level = 0).sum()['ClaimAmount']
# We initiate the PremiumAnalysis table, were the summary information will be compiled
PremAnalysisGBM = freqs_test[['ClaimNb', 'Exposure']].copy()
# We perform the joint with PremiumAnalysis as the target
PremAnalysisGBM = pd.merge(PremAnalysisGBM, AggregatedLoss, how = 'left', left_index = True, right
_index = True)
PremAnalysisGBM.fillna(0, inplace = True)
PremAnalysisGBM = PremAnalysisGBM.rename(columns = {'ClaimAmount':'Aggregated Loss'})
PremAnalysisGBM['Pred. Frequency'], PremAnalysisGBM['Pred. Severity'], PremAnalysisGBM['Pure Premi
um'] = PricingGBM(freqs_test,

weights = freqs_test2_w,

freqGBM = GBMmodel0,

sevGBM = GBMmodelS1)

PremAnalysisGBM['Anual Pure Prem.'] = PremAnalysisGBM['Pure Premium'] / PremAnalysisGBM['Exposure'
]
PremAnalysisGBM.describe()

PremAnalysisGBM.iloc[15:23]

seed = 2303
PremAnalysisGBM.iloc[[seed]][['Pred. Frequency', 'Pred. Severity', 'Pure Premium', 'Anual Pure Pre
m.']]
```

# Final comparative Analysis between GLMs and GBM models

# Predictive Performance

```
pd.options.display.float_format = '{:,.6f}'.format
PerformanceTable = pd.DataFrame({'Model Type': modeltype, 'Model Desc.': modeldesc,
                                 'GE Type': GEtype, 'Model Train GE': modeltrainGE, 'Model test GE
': modeltestGE})
PerformanceTable = PerformanceTable.set_index(['Model Type', 'Model Desc.'])
PerformanceTable.sort_index(level = 0)
```

# Premium Analysis

```
pd.options.display.float_format = '{:,.2f}'.format
CompStatistics = pd.DataFrame({'GLM':PremAnalysisGLM.sum().values, 'GBM':PremAnalysisGBM.sum().val
ues},
                              index = PremAnalysisGLM.sum().index)

LossRatio = CompStatistics.loc[['Pure Premium']].values / CompStatistics.loc[['Aggregated Loss']]
LossRatio = LossRatio.rename(index = {'Aggregated Loss':'Loss Ratio'})
CompStatistics = CompStatistics.drop('Pred. Severity')
CompStatistics.loc['ClaimNb'] = CompStatistics.loc['ClaimNb'].map('{:,.0f}'.format)
CompStatistics.loc['Aggregated Loss'] = CompStatistics.loc['Aggregated Loss'].map('{:,.2f} €'.form
at)
CompStatistics.loc['Pure Premium'] = CompStatistics.loc['Pure Premium'].map('{:,.2f} €'.format)
CompStatistics.loc['Anual Pure Prem.'] = CompStatistics.loc['Anual Pure Prem.'].map('{:,.2f} €'.fo
rmat)

CompStatistics = CompStatistics.append(LossRatio)
CompStatistics.loc['Loss Ratio'] = CompStatistics.loc['Loss Ratio'].map('{:,.2%}'.format)
CompStatistics
```

# Comparative Premium Tables

```
ComparativePrem = PremAnalysisGLM[['ClaimNb', 'Exposure', 'Aggregated Loss', 'Pure Premium']]
ComparativePrem = ComparativePrem.rename(columns = {'Pure Premium':'GLM Premium'})
ComparativePrem['GBM Premium'] = PremAnalysisGBM['Pure Premium']
ComparativePrem[15:23]
pd.options.display.float_format = '{:,.2f}'.format
ComparativePrem.describe()[['Aggregated Loss', 'GLM Premium', 'GBM Premium']]
train_fr = freqs_train['ClaimNb'].sum()/freqs_train['Exposure'].sum()
print('The empirical frequency on train set was {}'.format(round(train_fr, 6)))
test_fr = freqs_test['ClaimNb'].sum()/freqs_test['Exposure'].sum()
print('The empirical frequency on test set was {}'.format(round(test_fr, 6)))
print('The difference amounts to a {:,.3%}'.format(round(test_fr/train_fr-1, 6)))
train_cl = claims_train['ClaimAmount'].mean()
test_cl = claims_test['ClaimAmount'].mean()
print('The average severity on train set was {:,.2f}€'.format(round(train_cl, 6)))
print('The average severity on test set was {:,.2f}€'.format(round(test_cl, 6)))
print('The difference amounts to a {:,.3%}'.format(round(test_cl/train_cl-1, 6)))
```

# Quantile analysis

```
import statsmodels.api as sm
import pylab as py

quants = np.array(range(1,100,1))/100

GLMquants = [np.quantile(PremAnalysisGLM['Anual Pure Prem.'], i) for i in quants]
GBMquants = [np.quantile(PremAnalysisGBM['Anual Pure Prem.'], i) for i in quants]
```

# QQplot

```
higher = [1 if gbm > glm else 0 for glm, gbm in zip(GLMquants, GBMquants)]

from matplotlib.pyplot import figure
from matplotlib.colors import ListedColormap

colormap = ListedColormap(['royalblue', 'darkorange'])
classes = ['GLM is higher', 'GBM is higher']

fig, ax = plt.subplots(figsize=(7, 7), dpi=100)
scat = ax.scatter(x = GBMquants, y = GLMquants, s = 20, c = higher, cmap = colormap, marker = "+")
ax.yaxis.set_major_formatter('{x:1.0f}€')
ax.xaxis.set_major_formatter('{x:1.0f}€')
ax.grid(True)
ax.set_xlim([30, 220])
ax.set_ylim([30, 220])
ax.plot([0,220], [0, 220], '--', lw=1, color = 'm')
ax.set_xlabel("GBM pure premium quantiles", fontsize=12)
ax.set_ylabel("GLM pure premium quantiles", fontsize=12)
ax.annotate('Break percentile is {}%'.format(len(GLMquants)-sum(higher)), xy=(100, 100), xytext=(127, 77),
            arrowprops=dict(facecolor='forestgreen', shrink=0.05))
ax.legend(handles=scat.legend_elements()[0], labels=classes)

GBMquants[len(GLMquants)-sum(higher)]
```

# Premium scatter plot

```
premiums = ComparativePrem.sample(2000)

upperlim = 300
fig, ax = plt.subplots(figsize=(5, 5), dpi=100)
ax.scatter(x = premiums['GBM Premium'], y = premiums['GLM Premium'], alpha=0.3, s = 15, marker = "+")
ax.yaxis.set_major_formatter('{x:1.0f}€')
ax.xaxis.set_major_formatter('{x:1.0f}€')
ax.grid(True)
ax.set_xlim([0, upperlim])
ax.set_ylim([0, upperlim])
ax.plot([0,upperlim], [0, upperlim], '--', lw=1, color = 'm')
ax.set_xlabel("GBM pure premium", fontsize=12)
ax.set_ylabel("GLM pure premium", fontsize=12)

upperlim = 300
fig, ax = plt.subplots(figsize=(7, 7), dpi=100)
ax.scatter(x = premiums['GBM Premium']/premiums['Exposure'], y = premiums['GLM Premium']/premiums['Exposure'], alpha=0.3, s = 15, marker = "+")
ax.yaxis.set_major_formatter('{x:1.0f}€')
ax.xaxis.set_major_formatter('{x:1.0f}€')
ax.grid(True)
ax.set_xlim([0, upperlim])
ax.set_ylim([0, upperlim])
ax.plot([0,upperlim], [0, upperlim], '--', lw=1, color = 'm')
ax.set_xlabel("GBM anualized pure premium", fontsize=10)
ax.set_ylabel("GLM anualized pure premium", fontsize=10)
```

D. Visualizations through ggplot2 in R

Some of the initial EDA plots and figures from Chapter 3 where made through ggplot2 in an R environment, given that the original data set used in court case study comes from the R package CASdatasets.

# French Motor Third-Party Liability datasets

```
library(xts)
library(sp)
library(zoo)
library(knitr)
library(ggplot2)
library(dplyr)

library(CASdatasets)
```

# Importing the data

```
data(freMTPLfreq)
data(freMTPLsev)
data(freMTPL2freq)
data(freMTPL2sev)
```

# EDA for CAS dataset

# freMTPLfreq

```
ClaimsNb <- freMTPLfreq$ClaimNb
freMTPLfreq$ClaimNb <- array(freMTPLfreq$ClaimNb)
levels(freMTPLfreq[["Brand"]]) <- c("Fiat", "J-{N}/K", "MChB", "OGmF", "other", "RNC", "VASkSe")
levels(freMTPLfreq$Region) <- c("Aq", "BN", "Br", "Ce", "HN", "IdF", "L", "NPdC", "PdlL", "PC")
str(freMTPLfreq)

library(ggpubr)
library(gridExtra)

str(freMTPLsev)

library(tidyverse)
path = "C:\\Users\\usuario\\Desktop\\Uni\\Máster 2º Año\\Cuarto Cuatrimestre\\TFM"

write.csv(freMTPLfreq, file = str_c(path, "\\", "freMTPLfreq.csv"), row.names = FALSE)
write.csv(freMTPLsev, file = str_c(path, "\\", "freMTPLsev.csv"), row.names = FALSE)
```

# Variables exploration

```
ggplot(freMTPL2freq) +
  geom_density(aes(x = Density), fill = "goldenrod1", colour = "goldenrod3") +
  labs(title = "Histogram of Density Distribution", x = "Density", y = "Count")

 exposure <- ggplot(freMTPL2freq) +
  geom_density(aes(x = Exposure), fill = "goldenrod1", colour = "goldenrod3") +
  theme(axis.text = element_text(size = 7)) +
  xlim(0, 1.1)

table(freMTPLfreq$ClaimNb)
lin <- ggplot(freMTPLfreq) +
  geom_bar(aes(x = ClaimNb), fill = "goldenrod1", colour = "goldenrod3") +
  scale_y_continuous(labels = scales::comma) +
  labs(title = "nº Claims Distribution on linear scale", x = "Exposure", y = "nº Policies") +
  theme(plot.title = element_text(size = 11, face = "bold"))

log <- ggplot(freMTPLfreq) +
  geom_bar(aes(x = ClaimNb), fill = "goldenrod1", colour = "goldenrod3") +
  scale_y_log10(labels = scales::comma, n.breaks = 5) +
  labs(title = "nº Claims Distribution on log scale", x = "Exposure", y = "nº Policies") +
  theme(plot.title = element_text(size = 11, face = "bold"))

grid.arrange(lin, log, nrow = 1)

Power <- ggplot(freMTPLfreq) +
  geom_bar(aes(x = Power, fill = Power)) +
  theme(axis.text = element_text(size = 7)) +
  theme(legend.position = "none")
CarAge <- ggplot(freMTPLfreq) +
  geom_bar(aes(x = CarAge), fill = "aquamarine2", colour = "aquamarine3") +
  theme(axis.text = element_text(size = 7)) +
  xlim(0, 50)
DriverAge <- ggplot(freMTPLfreq) +
  geom_bar(aes(x = DriverAge),fill = "lightsalmon1", colour = "lightsalmon2") +
  theme(axis.text = element_text(size = 7))
Brand <- ggplot(freMTPLfreq) +
  geom_bar(aes(x = Brand, fill = Brand)) +
  theme(legend.position = "none") +
  theme(axis.text = element_text(size = 7))
Gas <- ggplot(freMTPLfreq) +
  geom_bar(aes(x = Gas, fill = Gas)) +
  theme(legend.position = "none") +
  theme(axis.text = element_text(size = 7))
Region <- ggplot(freMTPLfreq) +
  geom_bar(aes(x = Region, fill = Region)) +
  theme(legend.position = "none") +
  theme(axis.text = element_text(size = 7))
Density <- ggplot(freMTPLfreq) +
  geom_density(aes(x = Density), fill = "magenta1", colour = "magenta3") +
  labs(x = "Population Density", y = "density") +
  theme(axis.text = element_text(size = 7))
grid.arrange(Power, CarAge, DriverAge, Brand, Gas, Region, exposure, Density, nrow = 4)

outl <- ggplot(freMTPLfreq) +
    geom_bar(aes(x = CarAge), fill = "aquamarine2", colour = "aquamarine3") +
    # labs(title = "CarAge outliers distribution") +
    theme(axis.text = element_text(size = 7), plot.title = element_text(size = 11, face = "bold"))
+
    xlim(40, 101)
outl2 <- ggplot(freMTPLfreq) +
    geom_bar(aes(x = DriverAge), fill = "lightsalmon1", colour = "lightsalmon2") +
    theme(axis.text = element_text(size = 7), plot.title = element_text(size = 11, face = "bold"))
+
    xlim(90, 101)
grid.arrange(outl, outl2, nrow = 1)

amounts <- ggplot(freMTPLsev) +
  geom_density(aes(x = ClaimAmount), fill = "goldenrod1", colour = "goldenrod3") +
  theme(axis.text = element_text(size = 10)) +

  xlim(0, 4000) +
  ylim(0, 0.003) +
  labs(x = "Claim Amount (euros)", y = "density")

quant <- c(seq(0.8, 1, 0.02), 0.99)
results <- qgamma(quant, shape = 0.010221898037467, scale = 208376.1736022709)

analysis <- cbind(quant, results)
```