P&C Reinsurance modeling

Pure Premium Estimation and Creation of a Reinsurance Program

Authors:
Paul Chasseray
Gauthier Eldin
Aurégann Lefebvre

Supervisors:
B. Franke (Professor)
A. Nahelou (Actuary)
C. Wang (Actuary)

Euro-Institut d'Actuariat
AXA Global P&C
May 15, 2017

Word cloud using text mining in *R*

Acknowledgment

First and foremost, we would like to thank all the people who contributed to the success of our Bureau d'Etude and helped us in the completion of this report.

We would also want to express our sincere gratitude to our mentor at AXA Global P&C, Cheming Wang, for taking the valuable time to explain thoroughly the various tasks that we needed to work on in this project.

Lastly, We would also like to thank the Professors of the EURIA that were involved in the making of our project : Brice Franke and Franck Vermet for their guidance and immeasurable help in supervising our project. A great thanks as well to Anthony Nahelou who gave us directions in order to complete our project thoroughly.

Contents

List of Figures

Introduction

It is critical to price well *reinsurance products*. There are a lot of high risks that hang over a reinsurers head, one *natural catastrophe* affects a whole portfolio. P&C Reinsurance has a high-end market in term of risk profile assessment and pricing techniques, which requires special modeling skills in order to have a proper view on the *risk* and its *volatilities*. The job of the reinsurer is to diversify its *risks* so that a critical loss on a *portfolio* won't affect the company in a significant way. To do that, the products have to be well priced.

As part of our first year of *Masters degree* at the EURIA, We were able to work collectively on a project that allowed us, students, to apply the notions that we have learned in class. The students are helped and tutored in this project by the professors of EURIA and *actuaries* at the partner company. This projects gives the students the opportunity to be introduced to the professional work as well as to work on real and concrete issues. This year *AXA Global P&C* offered the students of the EURIA the opportunity to work on a Bureau d'Etude named "*P&C* reinsurance modeling".

The aims of our project are to estimate a *pure premium* on a specific portfolio of *reinsurance* and to compare *reinsurance programs* that could be adapted to this portfolio. The *risks* that we must price are related to *earthquakes*. In order to analyze these *risks* and to price them we were given simulated data of claims caused by *earthquakes*. The data came from multiple sources and had to be merged to get a full view of what we had to deal with.

In this report, we will first explain the *basics of reinsurance* as they are crucial to understanding our project. After having thoroughly presented the *data* and explained our *specific goals*, we will describe the different *models* that we applied on this data as well as the results we got for each model, before comparing *reinsurance programs* that could be adapted to our data.

Chapter 1

Introduction to reinsurance

The use of the knowledge of *reinsurance* was critical to us because it gave us the information needed in order to fully understand the data that was examined as well as understanding how a *reinsurance* program can be implemented. In order to comprehend this, we will see in this introduction to *reinsurance* the general principles involved but also its origins and evolution over the years (understanding the mistakes made in the past and how they were fixed can help us know what not to do in this program), the purpose of *reinsurance* and its market and finally we will discuss the most important part, the forms of *reinsurance*.

1.1 Definition and general principles

In essence, *reinsurance* can be defined as insurance for insurance companies to face larger *risks*, that the insurance companies cannot handle by themselves. For example, in the case a drought or an *earthquake* that would affect a lot of the portfolio, the insurance companies transfer a part of those *risks* to *reinsurance* companies. It stabilizes the insurer's results. *Reinsurance companies* deal with professional corporate parties.

1.2 History

The first *reinsurance* transaction dates back to 1370. At that time, goods were exchanged mostly by sea and the travels were insured against the *risks*, in particular against the *risks* of piracy.

The first written reference of a *reinsurance* transaction was related to a maritime policy covering the travel of goods from Italy to the Netherlands in 1370. The period of the trip between Spain and the Netherlands was known to be the most dangerous and was *reinsured entirely*. The insurer therefore only kept on its own account the part of the trip between Italy and Spain.

At the time, the contract was passed between the reinsurer and the insurer was more of a bet or a gambling game than an *insurance transaction* in which the probabilities of occurrence, the severity of the claim and the diversification of *risks* are taken into account.

Reinsurance is performed since the XIVth century under the facultative form. For each object that the insurer underwrites, he has the choice to ask for the help of a reinsurer and this reinsurer has the choice to accept or not the *risk*.

Starting in the beginning of the XIXth century, *reinsurance* treaties were also used for *risks* of fire and in the middle of the XIXth century treaties of life *reinsurance* were introduced. In the beginning, it was insurance companies that operated on the insurance market. The business was primarily *proportional reinsurance treaties*. *Reinsurance* evolved tightly with industrialization, due to the facts that that machines were so expensive, the bigger concentration of value in developing cities and the amelioration of transportation means which required higher covers than that only specialized reinsurers could handle and that were capable to spread the *risks* over multiple countries.

The first *obligatory insurance treaty* that covered all the risk of a specific portfolio, was reached in 1821 between La Royale de Paris and Les Propriétaires Réunis de Bruxelles. It regarded *reinsurance* under the reciprocal form. The two players exchanged their *risks* to have a better diversification of their portfolio. However, *reinsurance* was not their principal job. In 1846 was founded the first independent *reinsurance* company called the Kolnische Ruckversicherungsgesellschaft. It was born after a fire devastated the city of Hamburg in 1842. At this point the insurer realized that it was impossible to conserve by themselves *risks* that could accumulate so much. A geographical and time diversification was necessary. That is how the first professional *reinsurance* company was created.

From the 1950s, the *reinsurance* methods started becoming more sophisticated and following the *non-professional reinsurers* stared disappearing. Through the years, the *reinsurance* companies created more and more regulations to stay solvable in all circumstances.

1.3 Purpose

A *reinsurance* treaty is bought by an insurance company (or ceding company as they cede the *risks*) from a *reinsurance* company. The roles of a reinsurer company are to: protect the own funds of the insurer against the gap in results, contribute in improving the *solvency* margin of the insurer, increase the possibilities of underwriting, decrease the need of funds of the insurance company and help in a better monitoring of the *risks*. Reinsurance provides coverage for all kinds of *risks*. The *reinsurance* companies diversify the *risks* by having a portfolio spread over risk and geography.

The reason that insurance companies purchase *reinsurance* is to remove the financial responsibility of the *risks* that they took on. It allows them to free economic and *risk capital* that they put aside to pay for the losses, worry less about the *solvency challenges* that they face as well as reduce their probability of bankruptcy. They help in stabilizing insurance by absorbing some of their losses. By doing so, it allows for the companies to reduce their *risk exposure* and their own *capital requirement.*

With the help of *reinsurance*, insurers can face the *risks* of today and stay solvable as well as keep prices affordable for new clients. Clients (insurance companies) have an incentive to purchase *reinsurance* mostly if the risk that they carry are high or volatile or if they are specialist insurer with a relatively small scope of diversification in their *risks.* In those cases, they rely heavily on *reinsurance.*

1.4 Reinsurance market

Today, there is around 200 companies that offer *reinsurance* worldwide, most of which are specialist reinsurers. Germany, Switzerland and the US host the biggest *reinsurance* companies. However, there are also well large insurance companies that also offer *reinsurance* business (such as *AXA* with *AXA Global P&C).* In 2016, the *reinsurance* shareholders' funds are estimated around 338 billion USD, this shows that the *reinsurance* market has a strong capital base allowing *reinsurance* companies to take on very large and complex *risks.* Many developing countries rely particularly on *reinsurance* because they are very sensitive to *natural catastrophes.*

The *reinsurance* market has been proven to be quite stable as it has handled major *disasters* such as the many *natural catastrophes* that occurred in 2011 (the tsunami and Fukushima in Japan, the floods in Thailand as well as the hurricanes in the US and many more). The reason for this stability is the broad geographical range of the *risks.*

1.5 Forms of reinsurance

1.5.1 Legal criteria

1.5.1.1 Obligatory reinsurance

A treaty, or *obligatory reinsurance* is the situation where an insurer has to cede a portion of the risk of all the policies in a portfolio (such as car policies, health insurances policies. . .) to the reinsurer, and the reinsurer has to accept the risk. It is the most common form of *reinsurance.*

Obligatory *reinsurance* allows the insurance and the *reinsurance* company to have long term relationships as the reinsurer must accept every new *risk* that the insurer has, it is automatically accepted by the terms of the contract. Both parties have to cede or accept any *risks* that are covered in the treaty they agreed on, they cannot exclude a *risk* if it fits within the terms of the contract. Because the treaties work automatically, it is easier for the administration to have a obligatory treaty, but also increases the *risks* of insolvency.

1.5.1.2 Facultative reinsurance

Facultative reinsurance uses a case by case approach. The insurer has the option to reinsure or not the risk and the reinsurer has the option to accept or not the *risk*. The reinsurer and the insurer negotiate each contract. It is usually used for large, unusual or catastrophic *risks* such as a building that are exposed to many different hazards. The *reinsurance* has to be acquainted with all the *risks* that this building carries. Thus it has to be treated separately from other insured objects. Today *facultative reinsurance* is used as an additional cover to some of the *risks* already covered by *obligatory treaties*.

1.5.2 Technichal criteria

Both *facultative* and *obligatory reinsurance* can be proportional or non-proportional.

1.5.2.1 Proportional reinsurance

A *proportional treaty* is an agreement between a reinsurer and a ceding company (the reinsured) in which they divide proportionally the liability as well as the premium. The ratio which the reinsurer takes is defined in the contract and in the case of the loss, the amount the reinsurer will have to pay is proportional to the amount of the loss. The reinsurer also compensates the insurer by what is called the *reinsurance* commission, the equvalent of a percentage of the *premium*.

Quota share *Quota-share* is the simplest form of proportional *reinsurance*. The reinsurer takes an agreed percentage of all the *policies* written by the insurer. The losses work the same way, the reinsurer will have to pay that percentage of the amount of the claim in case of a loss. This type of *reinsurance* is ideal for homogeneous portfolios such as house or car insurance.In the case of quota-share *reinsurance*, the reinsurer will always have to pay when a *claim* occurs even if it's just a little bit. The portion of the risk that the reinsurer takes is called the *ceded risk* and what the insurer keeps for himself is called the *retention*.

Quota share *reinsurance* can be helpful for insurers who are seeking capital relief or want to protect themselves against fluctuations in a portfolio or inflation. However it isn't the best protection for an insurer because in case of an extreme *loss* or an accumulation of loss, the insurer will still have to pay a percentage of all of it. As a result, the insurer might have to take another reinsurance coverage to protect itself from the *extreme losses*, it might therefore be useful to combine Quota-share *reinsurance* with another type of *reinsurance*.

Surplus share Surplus share is the most common kind of proportional *reinsurance*. The primary insurer takes the risk up to a certain amount that is specified in the in the contract and the reinsurer must pay the surplus that exceeds the set amount called the *retention* whereas in case of a loss with quote part *reinsurance*, the reinsurer will have to pay starting from the first dollar of the loss. It still works in proportions but they are calculated with the price of the policy and the *retention* of the insurer. There can be a limit to the *surplus share* and this limit is the maximum amount of *liability* the reinsurer is prepared to take on. It is called a line, it is the amount the insurer keeps as a risk for himself, so we would say that a two line surplus is when the reinsurer assumes coverage of twice what the insurer keeps. This kind of *reinsurance* is mostly useful to cover the largest *risks* in a portfolio. It is more flexible than *Quota-share reinsurance* however it takes more 'administration'.

1.5.2.2 Non-proportional reinsurance

Non-proportional reinsurance emerged in the 1970s at a time where insurance companies were getting stronger and were able to handle more frequent small *risks*. They had to find a way to reinsurer only the larger *risks* and accumulated losses, that could bankrupt the insurance company. *Non-proportional reinsurance* has no fixed division of the *premium* and the claims. This form of *reinsurance* reimburses only for the losses that are above a fixed amount. The reinsurer remains solely liable if a loss exceeds this amount.

All the losses that are below this amount are at the charge of the insurer. The fixed amount is called the *priority*. The advantage of non-proportional *reinsurance* is that the priority reflects the capacity to bare of the insurer, and the administration costs are relatively low seeing they don't have to calculate the proportions for each *risk*.

Excess of loss *Excess of loss* is the most common kind of *non-proportional reinsurance*. It can be set on individual *risks* or occurrence. When the excess of loss is for each *risk*, the reinsurer pays the amount of the loss for every single policy that has been affected. Its participation in each claim is limited by the cover specified in the contract. The *risks* that are usually insured by *excess of loss* are for when a single building burns down or an accumulation of loss due to one event (a storm, flood, ...).

This kind of *reinsurance* is effective for risk mitigation against large single losses. Catastrophe excess of loss is a program that takes into account all the aggregate losses caused by one event within the *reinsurance* treaty.

Stop loss *Stop loss* is a less frequent form of *reinsurance*. This kind of treaty *reinsurance* covers only a part of the annual loss of the insurer that exceeds a certain amount. It is mostly for insurers to protect itself from *fluctuations in claim* between the years. It usually only comes into play when the insurer has suffered a terrible loss, so when the claims exceeds the *premium* received by the primary insurer.

Chapter 2

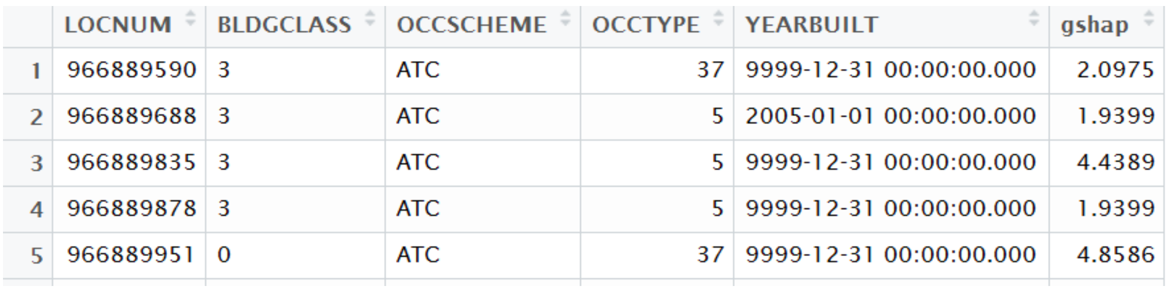
Data

2.1 Data description

The object of this project is to try and find a *reinsurance* program for some simulated data. The data that we received is in reference to the results of *earthquakes* on insured property. Firstly, we took the time to analyze and understand the data. The dataset that we were given was split into three separate files. Our first task was to find the similarities in the three datasets in order to merge them.

The first dataset that we had was called *SiteInfo*. In this dataset there is about 1.5 million entries, in figure 2.1 you can see the first five lines of this table, this dataset contains :

– *LOCNUM* : a number that represents the location of the insured property,
– *BLDGCLASS* : the type of construction,
– *OCCSCHEME* : the occupation scheme,
– *YEARBUILT* : the year that the building was built, if the information is missing, the number 9999 was imputed,
– *gshap* : the global seismic hazard assessment program, it gives an indication of the risk that is present in the specific zone.

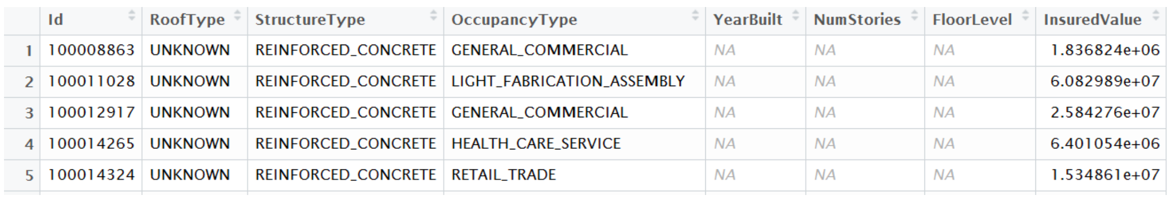


	LOCNUM	BLDGCLASS	OCCSCHEME	OCCTYPE	YEARBUILT	gshap
1	966889590	3	ATC	37	9999-12-31 00:00:00.000	2.0975
2	966889688	3	ATC	5	2005-01-01 00:00:00.000	1.9399
3	966889835	3	ATC	5	9999-12-31 00:00:00.000	4.4389
4	966889878	3	ATC	5	9999-12-31 00:00:00.000	1.9399
5	966889951	0	ATC	37	9999-12-31 00:00:00.000	4.8586

Figure 2.1: *SiteInfo* table

The second dataset we were given is named *Exposample*, We can find around 3000 entries, in figure 2.2 you can see the first five lines of this table, this dataset contains :

- *Id* : the number representation a specific event,
- *RoofType* : the type of roof, for example : flat roof, tiled roof, ...,
- *StructureType* : the structure of the building (what it is made of) such as concrete, steel frame, masonry, mobile home, ...,
- *OccupancyType* : the type of occupant in the building such as retail trade, commercial, hotel, wholesale trade, ...,
- *YearBuilt* : the year the building was built,
- *NumStories* : the number of floors in the building,
- *FloorLevel* : the floor on which the insured property is located,
- *InsuredValue* : the value of the insured property.

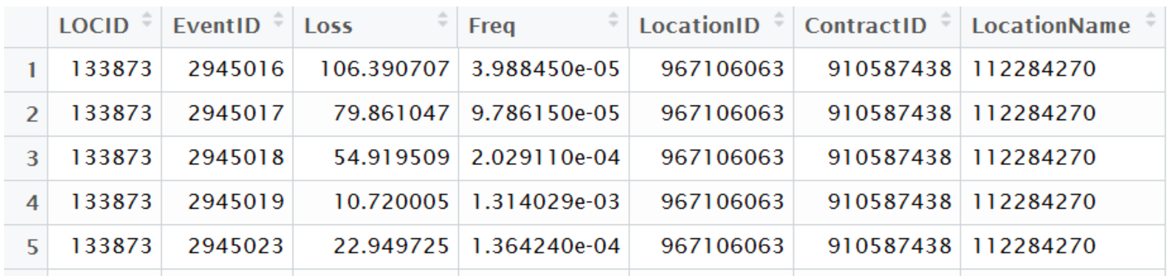


	Id	RoofType	StructureType	OccupancyType	YearBuilt	NumStories	FloorLevel	InsuredValue
1	100008863	UNKNOWN	REINFORCED_CONCRETE	GENERAL_COMMERCIAL	NA	NA	NA	1.836824e+06
2	100011028	UNKNOWN	REINFORCED_CONCRETE	LIGHT_FABRICATION_ASSEMBLY	NA	NA	NA	6.082989e+07
3	100012917	UNKNOWN	REINFORCED_CONCRETE	GENERAL_COMMERCIAL	NA	NA	NA	2.584276e+07
4	100014265	UNKNOWN	REINFORCED_CONCRETE	HEALTH_CARE_SERVICE	NA	NA	NA	6.401054e+06
5	100014324	UNKNOWN	REINFORCED_CONCRETE	RETAIL_TRADE	NA	NA	NA	1.534861e+07

Figure 2.2: *Exposample* table

The final dataset we recived is called the *EventLossTableBySite*, there are around 1.5 million entries, in figure 2.3 you can see the first five lines of this table, the variables present in this dataset are :

- *LocID* : a number representing a location,
- *EventID* : the number associated to each event,
- *Loss* : the loss incurred in a damage,
- *Freq* : the frequency at which seismic disaster occur in that location,
- *LocationID* : another number representing a location,
- *ContractID* : the number of the contract of reinsurance,
- *LocationName* : another number to represent the location of the building.



	LOCID	EventID	Loss	Freq	LocationID	ContractID	LocationName
1	133873	2945016	106.390707	3.988450e-05	967106063	910587438	112284270
2	133873	2945017	79.861047	9.786150e-05	967106063	910587438	112284270
3	133873	2945018	54.919509	2.029110e-04	967106063	910587438	112284270
4	133873	2945019	10.720005	1.314029e-03	967106063	910587438	112284270
5	133873	2945023	22.949725	1.364240e-04	967106063	910587438	112284270

Figure 2.3: *EventLossTableBySite* table

2.2 Data visualization

In order to have a good look at the data we were given, we first created a few graphs to see what the elements of the data that were missing as well as what kind of values we were given. We will present this data in the form of different graphs in order to get an idea of what our data represents. We will present each variable's distribution and then plot this variable against the *Log-PurePremium* where : $PurePremium = Loss \times Frequency$, because *PurePremium* will later be our target variable (see chapter 4).

We first analyzed the data that was missing from our database, in most of the columns there was some information missing. The missing data is summarized in figure 2.4. We can observe that the variables *RoofType*, *NumStories* and *FloorLevel* have more than 80% of the information missing. They will therefore be challenging to include in the models. The variable *YearBuilt* has above 50% of missing data which will also be challenging but more manageable and lastly, the other variables have under 10% of missing data, and are thus workable into our dataset.

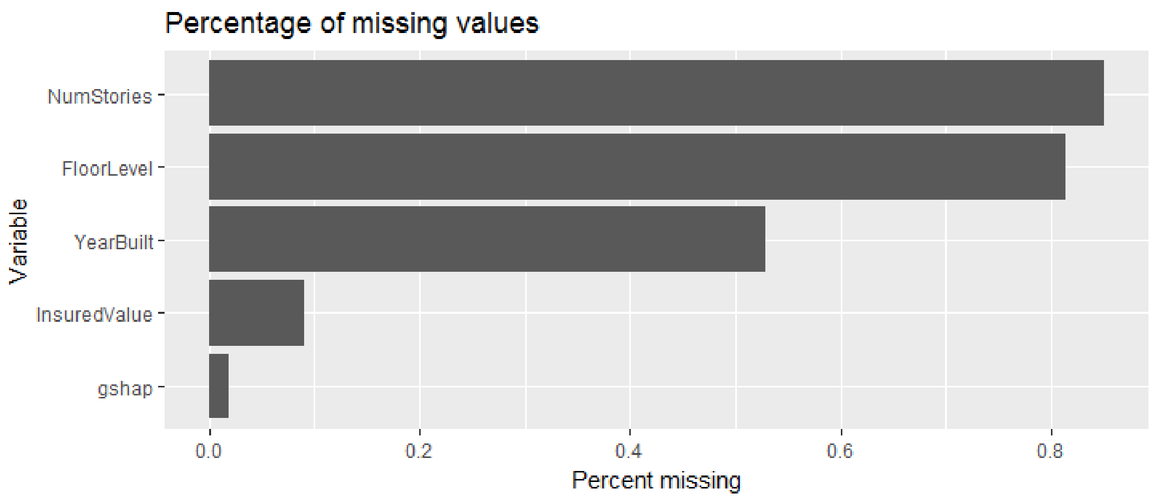


Figure 2.4: Missing data

After the analysis of the missing data we decided to look at the frequency and distribution of the variables.

The first variable we want to look at is the Loss as it is the value that is critical in estimating the pure premium. The loss variable has a mean of $\$30,170$ and ranges from $\$7$ to $\$63,720,000$.

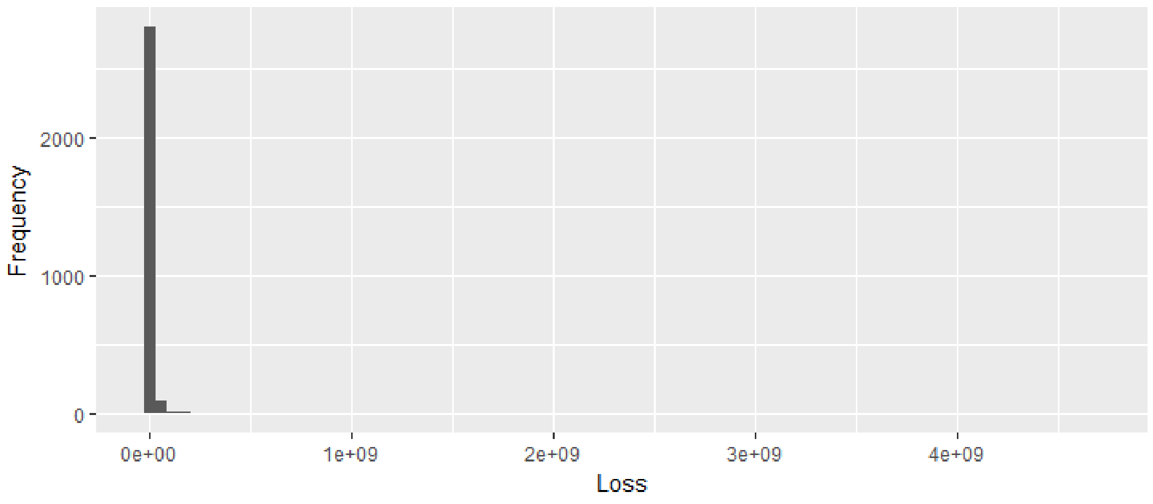


Figure 2.5: Histogram of the *Loss*

As we can see in figure 2.5 the histogram is not very useful due to very few large values and a high number of small values. Thus, in figure 2.6 we can observe the logarithm of the loss incurred in order to see better the extent. The distribution of the *log-loss* is more useful that way because it can be interpreted as different distribution function, such as a *normal distribution.*

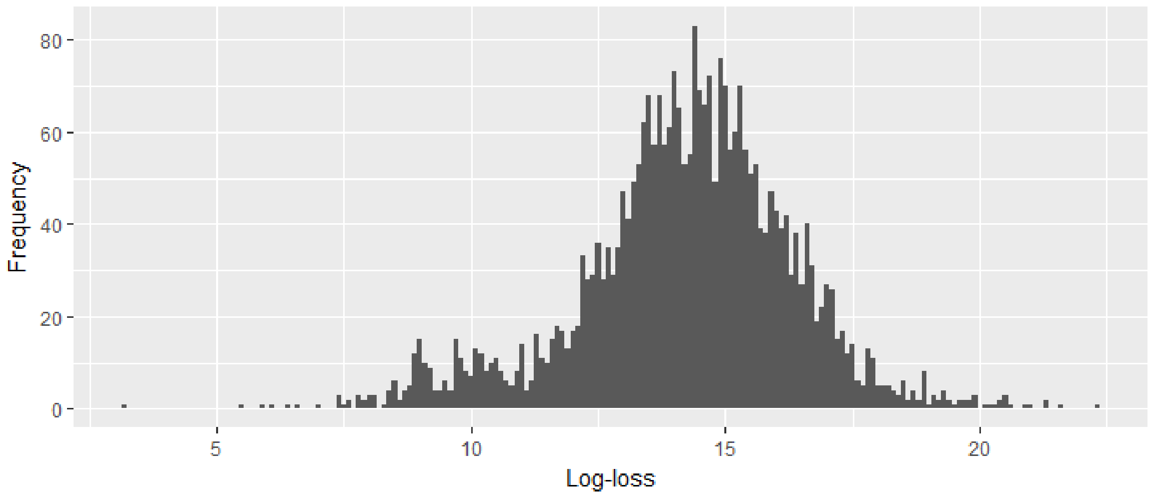


Figure 2.6: Histogram of the *Log-loss*

The second variable we looked at was the roof type. We know that this variable can be important because a flat roof has a higher chance of collapsing in the case of an earthquake. By looking at figure 2.7, we can notice that not taking into account the missing data and the unknown (which is the same as NA), tiled roofs are more common (11% of the time), there are 286 tiled roofs in the data, than flat roofs (5% of the time), there are 131 flat roofs in the data. However there is too much data missing to concur anything.

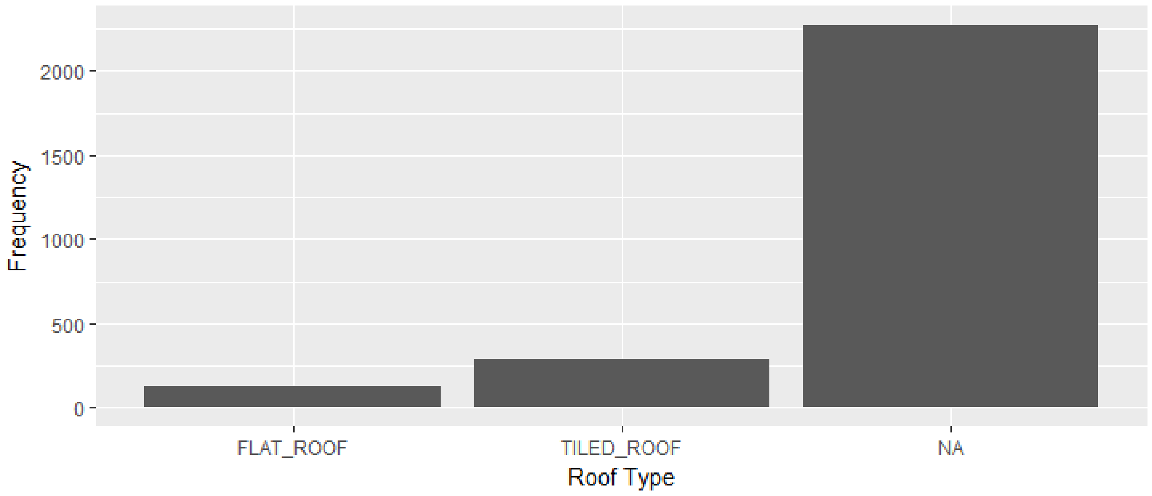


Figure 2.7: Frequency in the *RoofType* variable

In figure 2.8, we can observe a box-plot of the roof types against the Pure Premium. We can see that a flat roof incurs a bit more premium than a tiled roof, however since there are so few information in this variable, it is only speculation. A reason why this difference might be, is the structural integrity of some countries. In countries that are prone to earthquakes, it is common to have more flat roofs. For example in Taiwan the majority of the roofs are flat. In this type of countries the *PurePremium* would be higher.

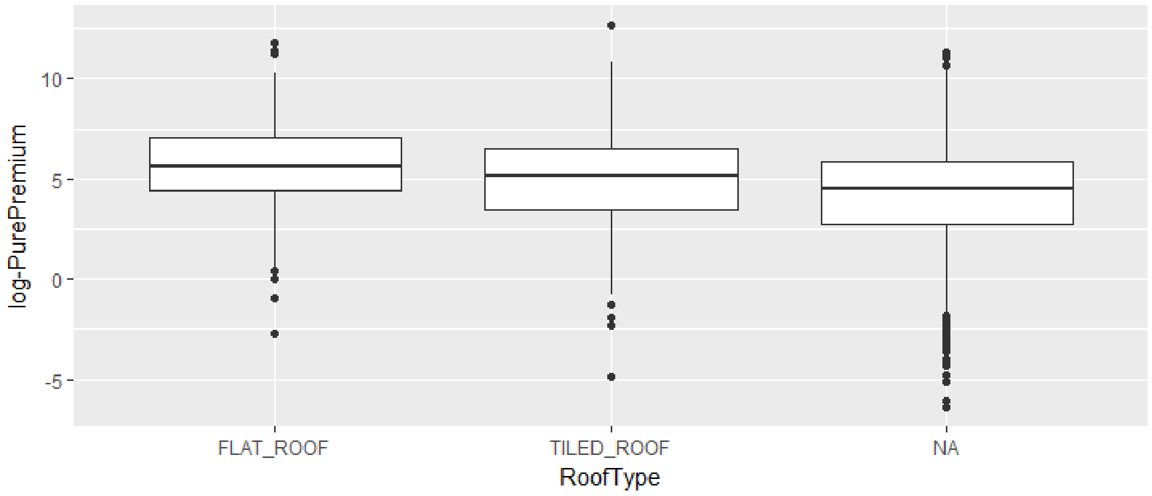


Figure 2.8: Boxplot of the *Log-PurePremium* against *RoofType*

For the *StructureType* variable, we noticed with figure 2.9 that the reinforced concrete structure is dominant with 95% of the types. Steel frame in second with only 4% and all the other structures combined represent just 1%. From this information we can make the hypothesis that this variable can be useful in estimating the loss, there may be a structure type that incurs a larger loss because it is less stable or the material is more expensive.

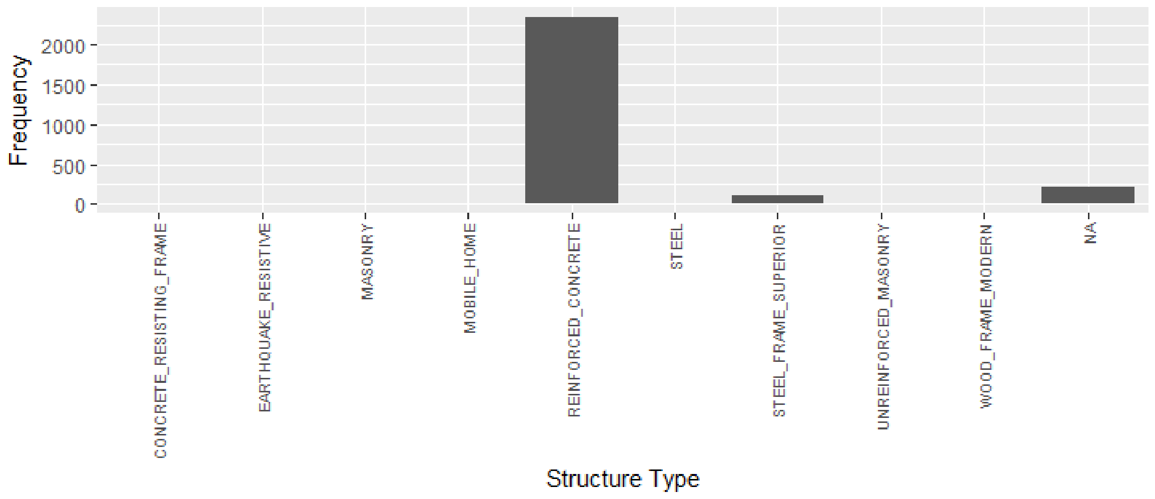


Figure 2.9: Frequency in the *StructureType* variable

The box-plot in figure 2.10 shows that the *pure premium* is spread thoroughly when the structure is reinforced concrete, as for other structure types, steel frame incurs lower premiums as well as higher premiums while just steel incurs higher premium. This might be due to the price of this material.

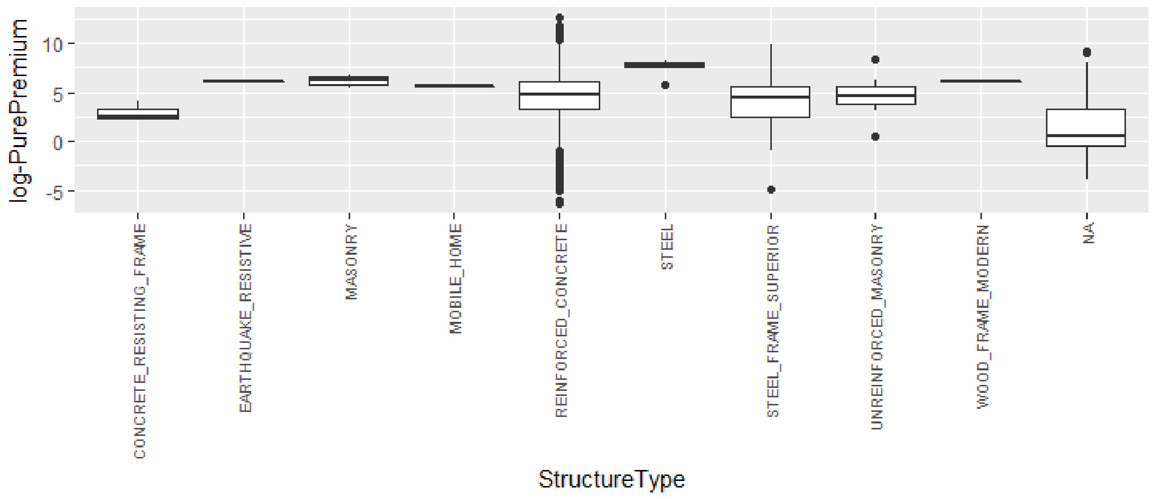


Figure 2.10: Boxplot of the *Log-PurePremium* against the *StructureType*

It is in the *OccupancyType* variable that we saw the most diversity. As we can see in figure 2.11, the most abundant occupancy types are general commercial with 31% and retail trade with 23%. The plurality of this variable can give us an idea of what kind of occupants are more costly in the case of a disaster which will helps us in pricing the *premium*.

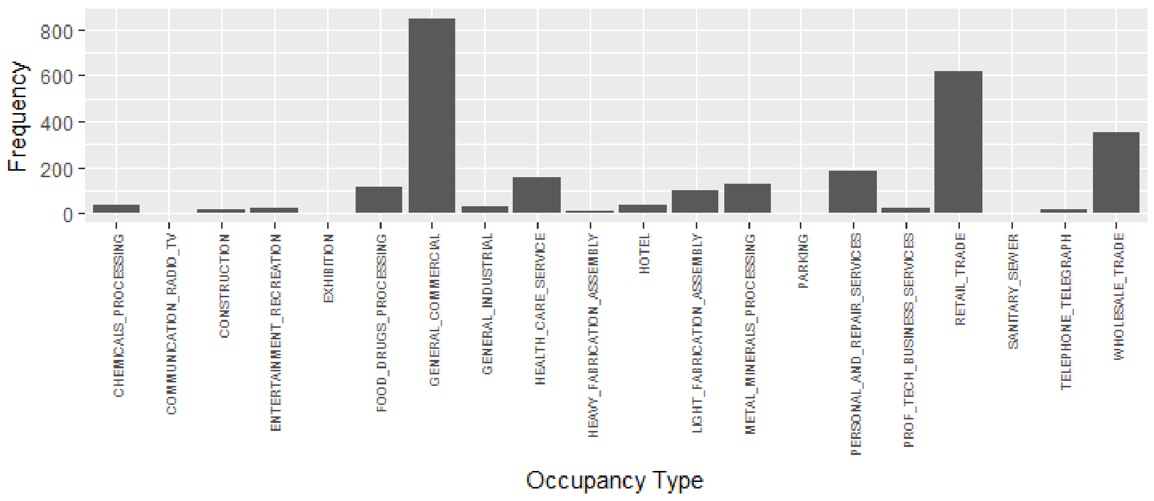


Figure 2.11: Frequency in the *OccupancyType* variable

The following box-plot (figure 2.12) shows that there is indeed diversity in the *OccupancyType*, and it is hard to see whether an occupancy type incurs a larger *PurePremium* than another type. We can see that general commercial has the larger spread and has the more extreme values, while parking has the lowest value however we know that this is because there is only one observation for parking and we can speculate that a parking structure doesn't need much construction and structure, so the *PurePremium* would be inferior.

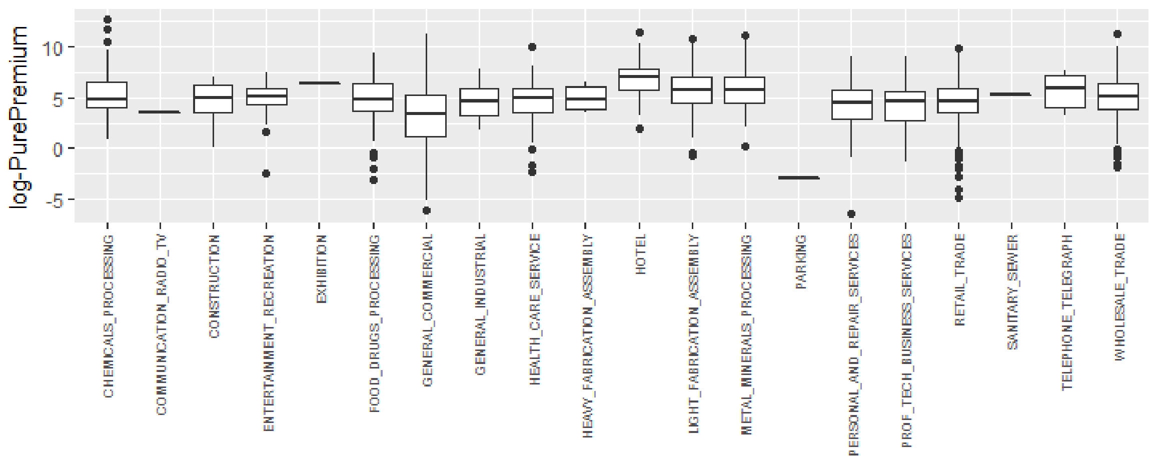


Figure 2.12: Boxplot of the *Log-PurePremium* against the *OccupancyType*

The *year of construction* can indicate us whether older buildings might be more prone to disasters than more recent buildings or if they were build sturdier than newer buildings. As shown in figure 2.13 there is mostly missing values but otherwise, most buildings were built in 1980.

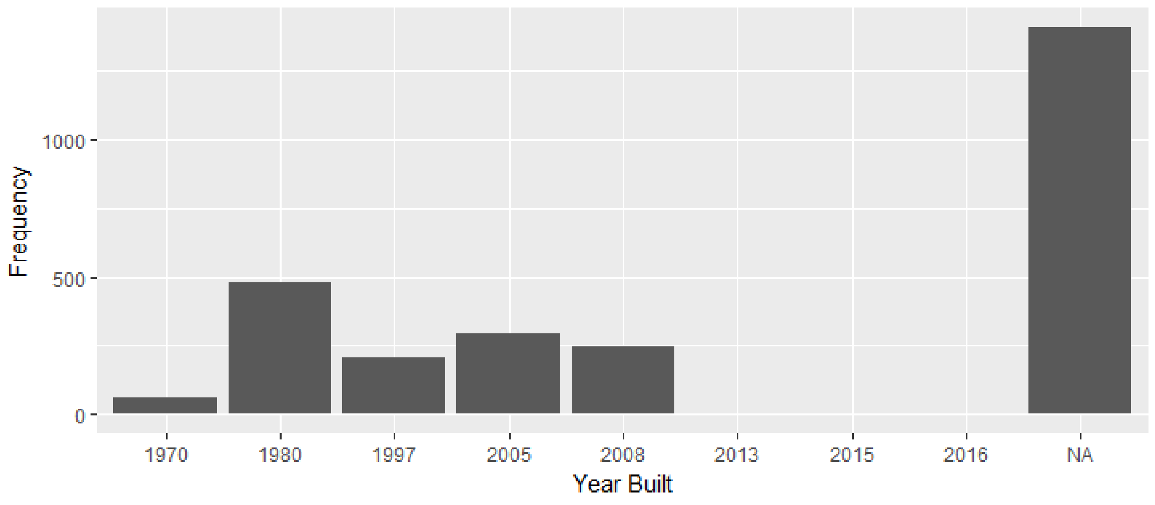


Figure 2.13: Frequency in the *YearBuilt* variable

Figure 2.14 demonstrates that the year that the building was built doesn't affect much the *PurePremium* incurred, however we can see that the later constructions (after 2010) seem to have smaller premiums, this might be because the norms for buildings are more regulated in recent years so they are less likely to provoke a large loss or just that there are not enough observations in the later years to see a large spread.

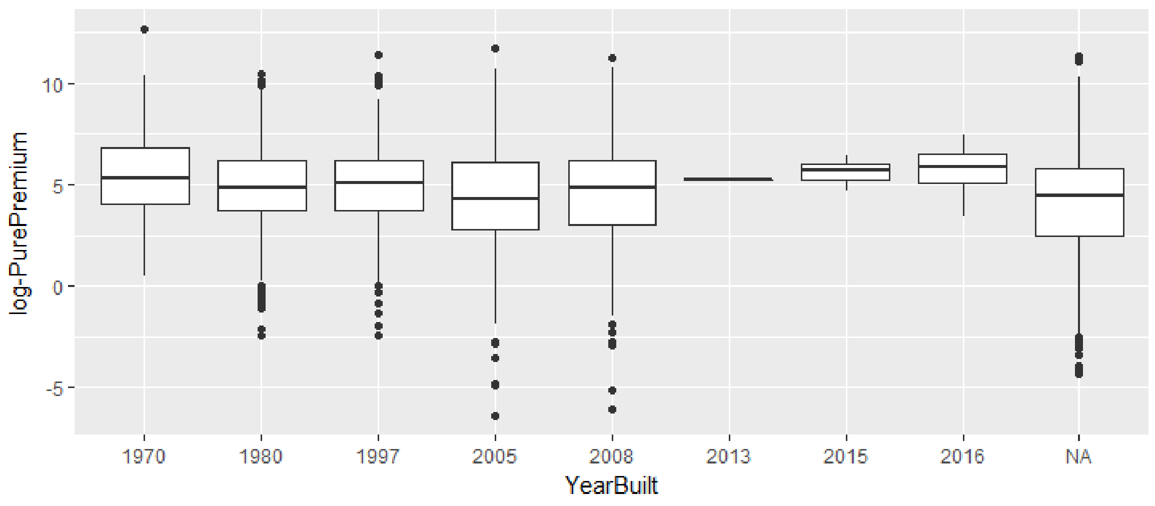


Figure 2.14: Boxplot of the *Log-PurePremium* against the *YearBuilt*

The *number of floors* in the building can vary quite a bit as seen in figure 2.15. There are most commonly 5 floors in the building and then 1 floor, we also see that there can be up to 15 floors in a building which increases the *risks* of loss when a disaster strikes because higher floors are more prone to destruction in the case of an *earthquake*.

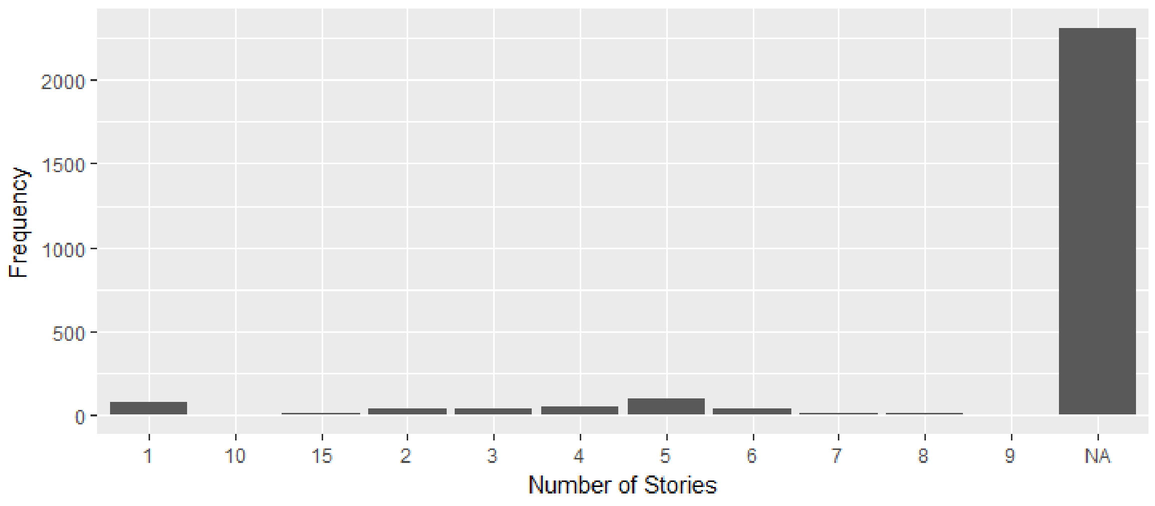


Figure 2.15: Frequency in the *NumStories* variable

The following box-plot displays that the *number of stories* doesn't seem to affect much the *PurePremium* because the higher premiums are when there are 1 or 3 or 15 stories. These numbers seem random. We could speculate that very tall buildings and very small buildings have the biggest *PurePremiums* because there are more prone to *risks*.

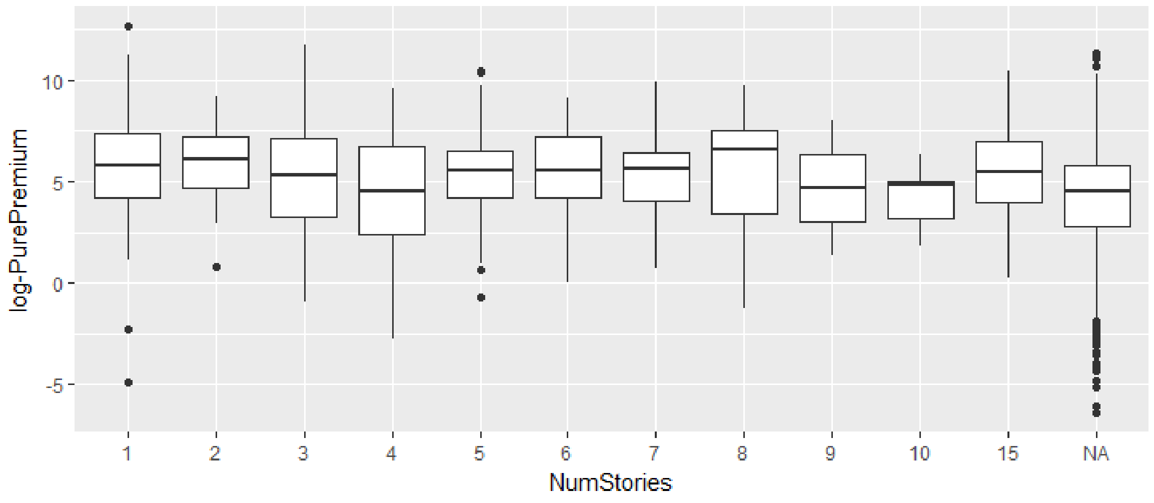


Figure 2.16: Boxplot of the *Log-PurePremium* against the *Numstories*

The business insured can be located on any floor. In figure 2.17, we notice that most of the insured properties are located on the first floor of the building and a bit less likely on the second floor, there are few of the insured properties on the other floors. The fact that they are mostly located on the first floor might be because they are stores and stores usually have a storefront on the ground level (we are making the hypothesis that the floors are numbered the American way, so the first floor is actually the ground floor). With this information we can try to understand the different *PurePremium* due to the floor location of the insured property, but for that we will need to figure out a solution for the *NAs*.

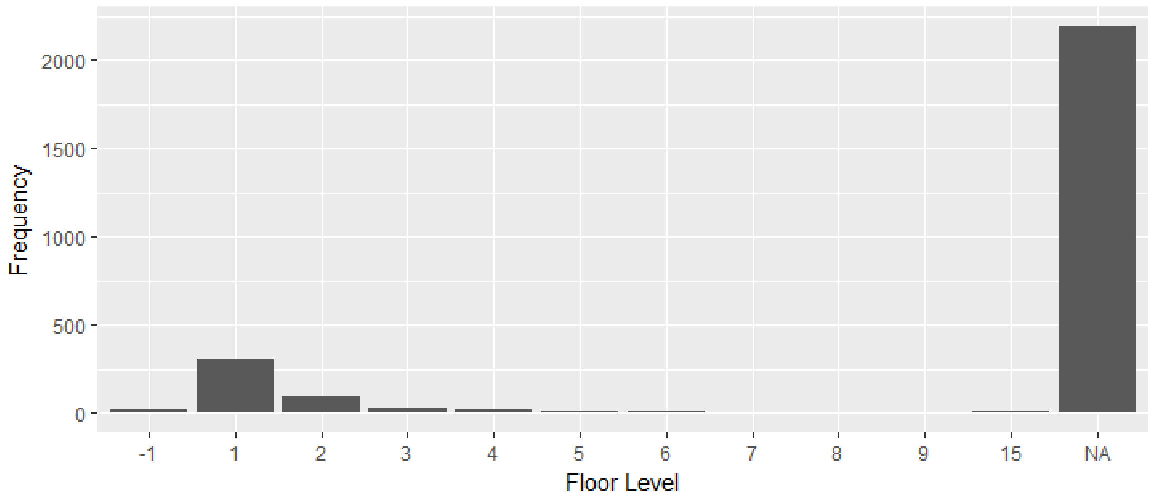


Figure 2.17: Frequency in the *FloorLevel* variable

When the insured property is on the first floor, that is when the spread of *PurePremium* is the highest as seen in figure 2.18. This might be because if a building collapse the first floor will be completely destroyed by the debris of the rest of the building.

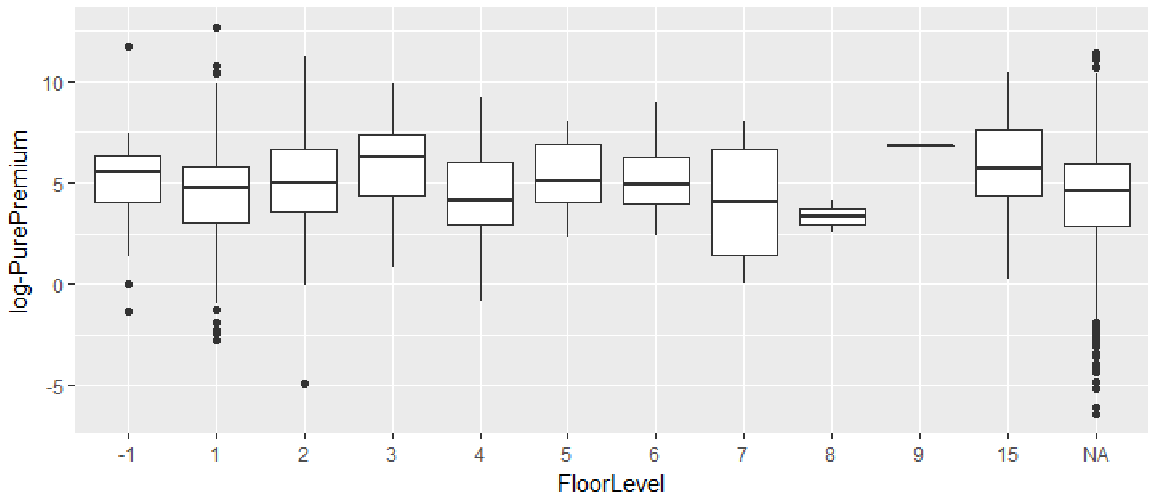


Figure 2.18: Boxplot of the *Log-PurePremium* and the *FloorLevel*

We finally analyzed one of the most important variable in our dataset, the insured value. The mean insured value is of 27.78 million dollars. The median insured value is of \$6.958 million. All the other insured values range between \$9, 940 and \$3.033 billion. This shows us that there is a wide range of insured value with a few really elevated values. In figure 2.19 we can see the distribution of the insured value. However there are too many small values to see the distribution well.

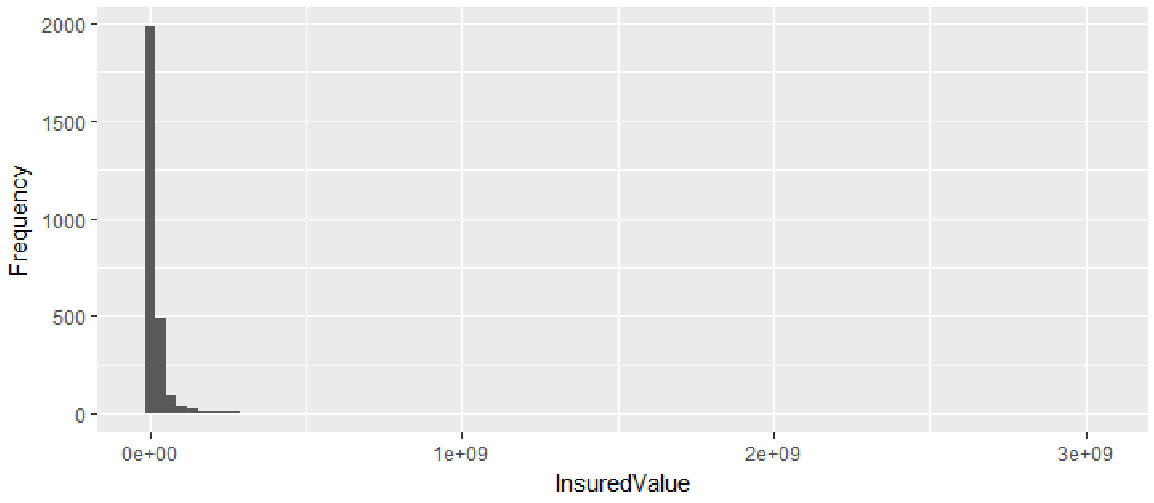


Figure 2.19: Histogram of the *Insured Value*

In figure 2.20, shows a histogram of the logarithm of the *InsuredValue*. It represents better the spread of the losses than just a histogram of the *InsuredValue* because of the few very large claims. The large spike that we can observe the *log-InsuredValue* of 10.92 is the insured value $\$55,760$ that is present 110 times in our data. They are all general commercial and all of the other values are missing for this *InsuredValue*. We can speculate that this could be for example a chain of stores so all the insured values are the same, but this over present value might be due to another feature.

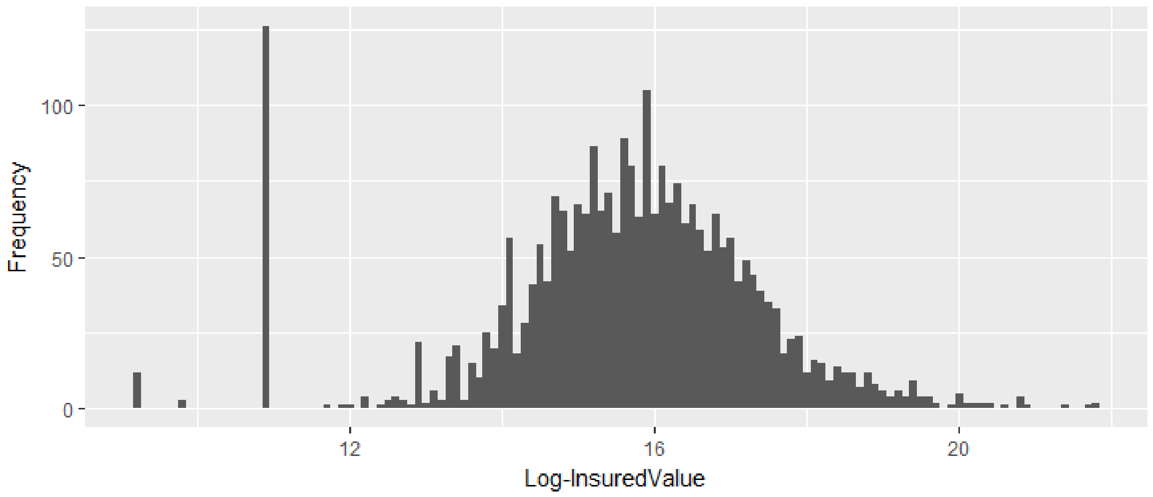


Figure 2.20: Histogram of the *Log-InsuredValue*

Figure 2.21 shows plot of the *Log-InsuredValue* against the *Log-PurePremium*. Before plotting this we suspected that these values would be correlated because a higher insured value would lead to a higher *PurePremium*. The green line represents a linear regression between the *Log-Insured* Value and the *Log-PurePremium* and we can clearly see that it does follow this pattern.

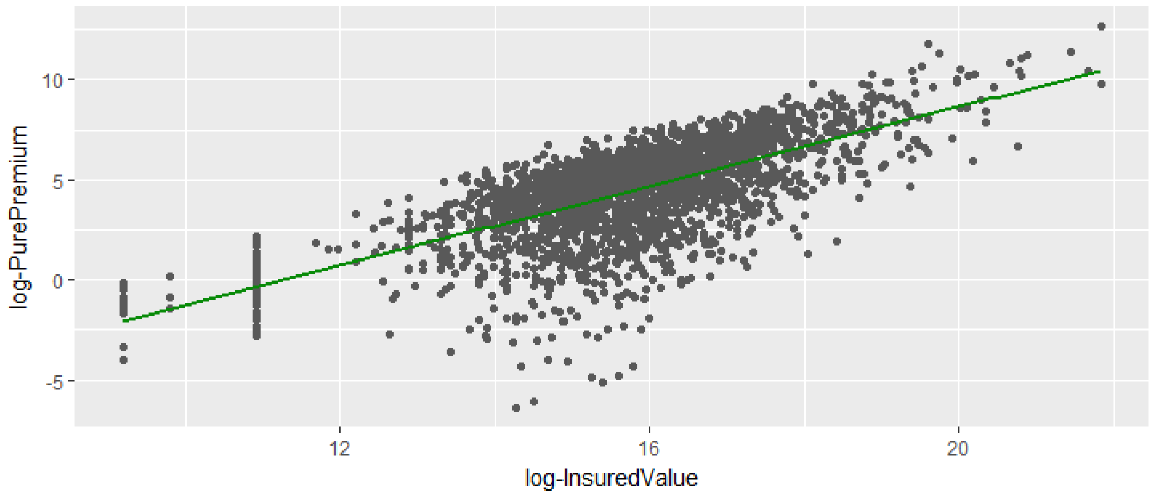


Figure 2.21: Plot of the *Log-PurePremium* against the *Log-InsuredValue*

The *global seismic hazard assessment program* in our dataset ranges from 0.9006 to 6.2089 with a mean of 3.2914. In figure 2.22, the spread of the *gshap* variable is represented. It is noticeable that each spike is after an integer, the values are otherwise spread equally from one to six except for occasional spikes.

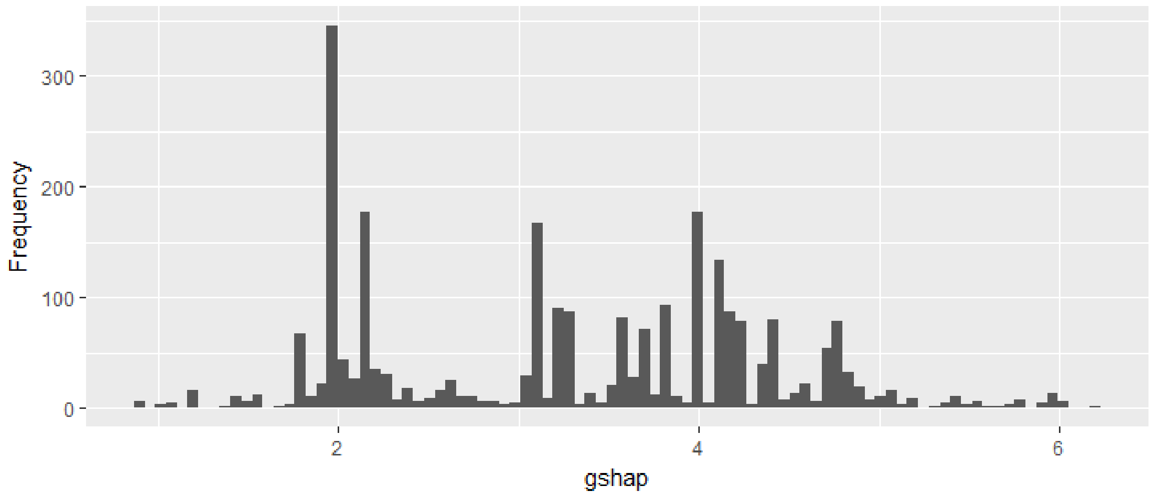


Figure 2.22: Spread of the *gshap* variable

Figure 2.23 shows that the *PurePremium* is distributed over all the values of the *gshap*, we cannot see a pattern with higher premiums where the *gshap* is higher. This can be because zones with higher seismic risks are better prepared for earthquakes with buildings that are more adapted. The green line represents the linear regression between the *gshap* and the *Log-PurePremium*.

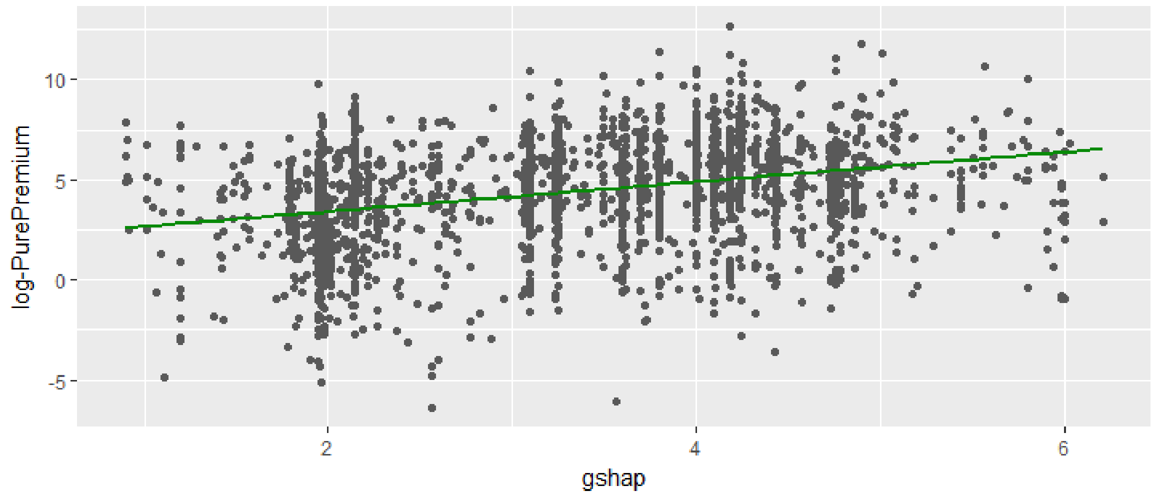
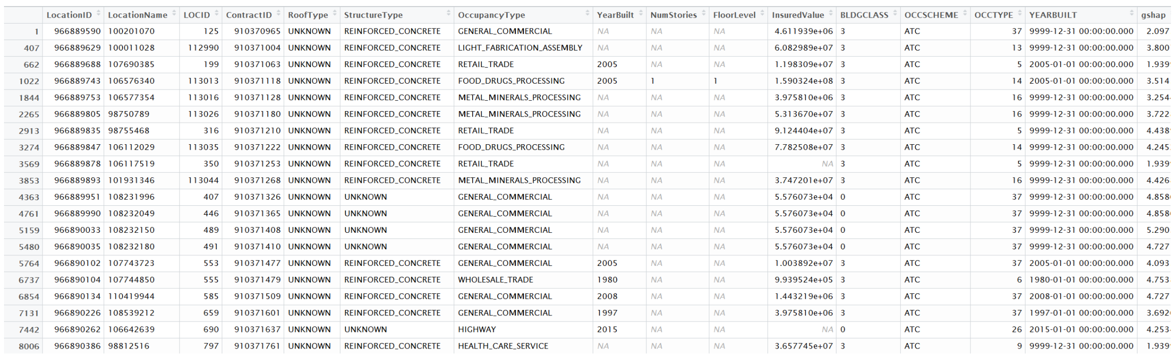


Figure 2.23: Plot of the *Log-PurePremium* against the *gshap*

2.3 Conditioning of the data

In a first step, we combined all the datasets together, the result of this merge is a table with $2,965$ entries and 16 variables. To combine the data, we found the columns that had the same significance, *LocationName* and *LocationID* and we merged the lines. This allowed us to have a table containing all the data needed for this project. We can see a excerpt of this data in figure 2.24



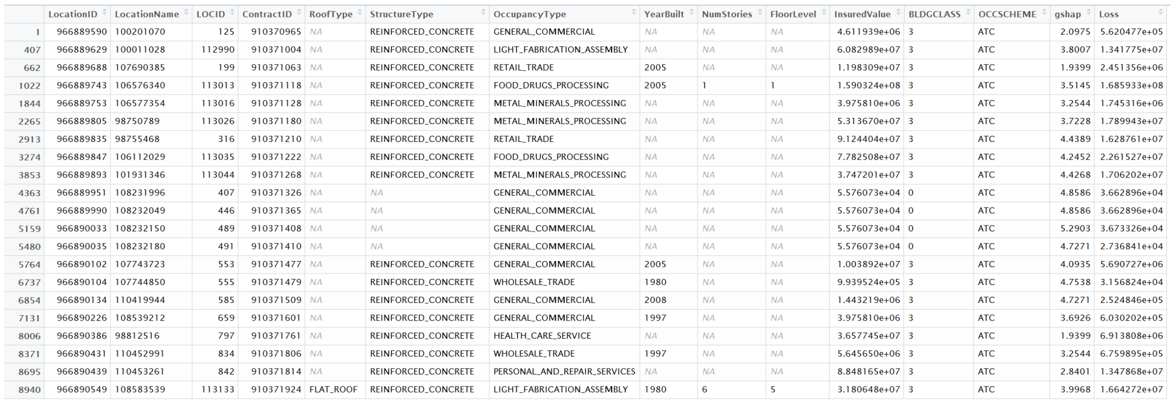
	LocationID	LocationName	LOCID	ContractID	RoofType	StructureType	OccupancyType	YearBuilt	NumStories	FloorLevel	InsuredValue	BLDGCLASS	OCCSCHEME	OCCTYPE	YEARBUILT	gshap
1	966889590	100201070	125	910370965	UNKNOWN	REINFORCED_CONCRETE	GENERAL_COMMERCIAL	NA	NA	NA	4.611939e+06	3	ATC	37	9999-12-31 00:00:00.000	2.097
407	966889629	100011028	112990	910371004	UNKNOWN	REINFORCED_CONCRETE	LIGHT_FABRICATION_ASSEMBLY	NA	NA	NA	6.082989e+07	3	ATC	13	9999-12-31 00:00:00.000	3.800
662	966889688	107690385	199	910371063	UNKNOWN	REINFORCED_CONCRETE	RETAIL_TRADE	2005	NA	NA	1.198309e+07	3	ATC	5	2005-01-01 00:00:00.000	1.939
1022	966889743	106576340	113013	910371118	UNKNOWN	REINFORCED_CONCRETE	FOOD_DRUGS_PROCESSING	2005	1	1	1.590324e+08	3	ATC	14	2005-01-01 00:00:00.000	3.514
1844	966889753	106577354	113016	910371128	UNKNOWN	REINFORCED_CONCRETE	METAL_MINERALS_PROCESSING	NA	NA	NA	3.975810e+06	3	ATC	16	9999-12-31 00:00:00.000	3.254
2265	966889805	98750789	113026	910371180	UNKNOWN	REINFORCED_CONCRETE	METAL_MINERALS_PROCESSING	NA	NA	NA	5.313670e+07	3	ATC	16	9999-12-31 00:00:00.000	3.722
2913	966889835	98755468	316	910371210	UNKNOWN	REINFORCED_CONCRETE	RETAIL_TRADE	NA	NA	NA	9.124404e+07	3	ATC	5	9999-12-31 00:00:00.000	4.438
3274	966889847	106112029	113035	910371222	UNKNOWN	REINFORCED_CONCRETE	FOOD_DRUGS_PROCESSING	NA	NA	NA	7.782508e+07	3	ATC	14	9999-12-31 00:00:00.000	4.245
3569	966889878	106117519	350	910371253	UNKNOWN	REINFORCED_CONCRETE	RETAIL_TRADE	NA	NA	NA	NA	3	ATC	5	9999-12-31 00:00:00.000	1.939
3853	966889893	101931346	113044	910371268	UNKNOWN	REINFORCED_CONCRETE	METAL_MINERALS_PROCESSING	NA	NA	NA	3.747201e+07	3	ATC	16	9999-12-31 00:00:00.000	4.426
4363	966889951	108231996	407	910371326	UNKNOWN	UNKNOWN	GENERAL_COMMERCIAL	NA	NA	NA	5.576073e+04	0	ATC	37	9999-12-31 00:00:00.000	4.858
4761	966889990	108232049	446	910371365	UNKNOWN	UNKNOWN	GENERAL_COMMERCIAL	NA	NA	NA	5.576073e+04	0	ATC	37	9999-12-31 00:00:00.000	4.858
5159	966890033	108232150	489	910371408	UNKNOWN	UNKNOWN	GENERAL_COMMERCIAL	NA	NA	NA	5.576073e+04	0	ATC	37	9999-12-31 00:00:00.000	5.290
5480	966890035	108232180	491	910371410	UNKNOWN	UNKNOWN	GENERAL_COMMERCIAL	NA	NA	NA	5.576073e+04	0	ATC	37	9999-12-31 00:00:00.000	4.727
5764	966890102	107743723	553	910371477	UNKNOWN	REINFORCED_CONCRETE	GENERAL_COMMERCIAL	2005	NA	NA	1.003892e+07	3	ATC	37	2005-01-01 00:00:00.000	4.093
6737	966890104	107744850	555	910371479	UNKNOWN	REINFORCED_CONCRETE	WHOLESALE_TRADE	1980	NA	NA	9.939524e+05	3	ATC	6	1980-01-01 00:00:00.000	4.753
6854	966890134	110419944	585	910371509	UNKNOWN	REINFORCED_CONCRETE	GENERAL_COMMERCIAL	2008	NA	NA	1.443219e+06	3	ATC	37	2008-01-01 00:00:00.000	4.727
7131	966890226	108539212	659	910371601	UNKNOWN	REINFORCED_CONCRETE	GENERAL_COMMERCIAL	1997	NA	NA	3.975810e+06	3	ATC	37	1997-01-01 00:00:00.000	3.692
7442	966890262	106642639	690	910371637	UNKNOWN	UNKNOWN	HIGHWAY	2015	NA	NA	NA	0	ATC	26	2015-01-01 00:00:00.000	4.253
8006	966890386	98812516	797	910371761	UNKNOWN	REINFORCED_CONCRETE	HEALTH_CARE_SERVICE	NA	NA	NA	3.657745e+07	3	ATC	9	9999-12-31 00:00:00.000	1.939

Figure 2.24: A part of the full data table

This first step led us to look more closely at the data to find out if any abnormal values or outliers were present in the dataset, if we had to treat the *NAs* and if we had to delete entries or variables due to a lack of information.

2.3.1 Variables conditioning

We first analyzed the data to understand what we were dealing with, such as if there were any missing data, if there were data lines that seemed out of the ordinary, errors or other problems in the data. We noticed that they were two *YearBuilt* columns so we deleted the one that was harder to work with (the format was more complicated). We also deleted the *OCCSCHEME* column because it gave the same information as *OccupancyType* but less attainable. We also deleted the lines that were duplicates. In figure 2.25, we can see what the data looks like after these modifications.



	LocationID	LocationName	LOCID	ContractID	RoofType	StructureType	OccupancyType	YearBuilt	NumStories	FloorLevel	InsuredValue	BLDGCLASS	OCCSCHEME	gshap	Loss
1	966889590	100201070	125	910370965	NA	REINFORCED_CONCRETE	GENERAL_COMMERCIAL	NA	NA	NA	4.611939e+06	3	ATC	2.0975	5.620477e+05
407	966889629	100011028	112990	910371004	NA	REINFORCED_CONCRETE	LIGHT_FABRICATION_ASSEMBLY	NA	NA	NA	6.082989e+07	3	ATC	3.8007	1.341775e+07
662	966889688	107690385	199	910371063	NA	REINFORCED_CONCRETE	RETAIL_TRADE	2005	NA	NA	1.198309e+07	3	ATC	1.9399	2.451356e+06
1022	966889743	106576340	113013	910371118	NA	REINFORCED_CONCRETE	FOOD_DRUGS_PROCESSING	2005	1	1	1.590324e+08	3	ATC	3.5145	1.685933e+08
1844	966889753	106577354	113016	910371128	NA	REINFORCED_CONCRETE	METAL_MINERALS_PROCESSING	NA	NA	NA	3.975810e+06	3	ATC	3.2544	1.745316e+06
2265	966889805	98750789	113026	910371180	NA	REINFORCED_CONCRETE	METAL_MINERALS_PROCESSING	NA	NA	NA	5.313670e+07	3	ATC	3.7228	1.789943e+07
2913	966889835	98755468	316	910371210	NA	REINFORCED_CONCRETE	RETAIL_TRADE	NA	NA	NA	9.124404e+07	3	ATC	4.4389	1.628761e+07
3274	966889847	106112029	113035	910371222	NA	REINFORCED_CONCRETE	FOOD_DRUGS_PROCESSING	NA	NA	NA	7.782508e+07	3	ATC	4.2452	2.261527e+07
3853	966889893	101931346	113044	910371268	NA	REINFORCED_CONCRETE	METAL_MINERALS_PROCESSING	NA	NA	NA	3.747201e+07	3	ATC	4.4268	1.706202e+07
4363	966889951	108231996	407	910371326	NA	NA	GENERAL_COMMERCIAL	NA	NA	NA	5.576073e+04	0	ATC	4.8586	3.662896e+04
4761	966889990	108232049	446	910371365	NA	NA	GENERAL_COMMERCIAL	NA	NA	NA	5.576073e+04	0	ATC	4.8586	3.662896e+04
5159	966890033	108232150	489	910371408	NA	NA	GENERAL_COMMERCIAL	NA	NA	NA	5.576073e+04	0	ATC	5.2903	3.673326e+04
5480	966890035	108232180	491	910371410	NA	NA	GENERAL_COMMERCIAL	NA	NA	NA	5.576073e+04	0	ATC	4.7271	2.736841e+04
5764	966890102	107743723	553	910371477	NA	REINFORCED_CONCRETE	GENERAL_COMMERCIAL	2005	NA	NA	1.003892e+07	3	ATC	4.0935	5.690727e+06
6737	966890104	107744850	555	910371479	NA	REINFORCED_CONCRETE	WHOLESALE_TRADE	1980	NA	NA	9.939524e+05	3	ATC	4.7538	3.156824e+04
6854	966890134	110419944	585	910371509	NA	REINFORCED_CONCRETE	GENERAL_COMMERCIAL	2008	NA	NA	1.443219e+06	3	ATC	4.7271	2.524846e+05
7131	966890226	108539212	659	910371601	NA	REINFORCED_CONCRETE	GENERAL_COMMERCIAL	1997	NA	NA	3.975810e+06	3	ATC	3.6926	6.030202e+05
8006	966890386	98812516	797	910371761	NA	REINFORCED_CONCRETE	HEALTH_CARE_SERVICE	NA	NA	NA	3.657745e+07	3	ATC	1.9399	6.913808e+06
8371	966890431	110452991	834	910371806	NA	REINFORCED_CONCRETE	WHOLESALE_TRADE	1997	NA	NA	5.645650e+06	3	ATC	3.2544	6.759895e+05
8695	966890439	110453261	842	910371814	NA	REINFORCED_CONCRETE	PERSONAL_AND_REPAIR_SERVICES	NA	NA	NA	8.848165e+07	3	ATC	2.8401	1.347868e+07
8940	966890549	108583539	113133	910371924	FLAT_ROOF	REINFORCED_CONCRETE	LIGHT_FABRICATION_ASSEMBLY	1980	6	5	3.180648e+07	3	ATC	3.9968	1.664272e+07

Figure 2.25: Data with deleted variables

In a second phase we deleted all the lines where *InsuredValue* was missing (there were 268). This was done because these rows would be too hard to analyze with certain important information missing.

2.3.2 NAs conditioning

The first method that we tried to condition the *NAs* was to remove them completely from the dataset (delete the lines with *NAs*) but as we have seen in figure 2.4 in a few variable such as *RoofType*, there are too many values so by removing the lines with *NAs* we would remove more than 80% of the dataset so we decided find a different approach. Another method we tried was to remove these missing values only for the variables with less than 50% of missing values. However once again, we decided that it would remove too much relevant information from our database. We therefore decided to move on to another method than removing the *NAs*.

The second method we tried to remove of the *NAs* was to rename them so that they could become a modality. We renamed every value that was missing by “VM”. We used this modified database for the estimations of the loss however as we ran the different models we realized by creating this new modality it it did not give us good results since in some variables such as roof *RoofType* there is more than 80% of “VM” which made the predominant result “VM” and as this value doesn’t reflect anything. We therefore decided to move on to another approach. Due to this we learnt that the best method to take care of the *NAs* was an imputation method, a method allowing us to replace the missing value by an appropriate value.

The last method we worked on and decided to use in the estimation of the rate is the *MICE method* or *Multivariate Imputation by Chained Equations*. This method was created for complex databases with missing data. The *MICE method* is quite useful when there are values missing in more than one variable since most of the imputation methods only impute in one variables at a time and have to be done over and over again on the other variables.

The main hypothesis of this model is that the missing values must be *missing at random (MAR)*, it means that the propensity for a value of the data to be missing is not related to the missing data, but it is related to some of the observed data. It can be *missing not at random (MNAR)* however it would need more modeling assumptions which would influence the generated imputations.

The *MICE method* method is done by iterations over the data. It takes as an entry a estimation model to impute the values (such as a *regression tree* (see section 3.1.1) or a random forest (see section 3.1.2)). It starts with an initial imputation : it enters, in place of the missing value, the value that is the most common in that column if it is qualitative and the average if it is quantitative. Then it moves on and does the same for all the other variables with missing values in the database starting with the one that has the fewer missing values and goes in increasing oreder.

On the second iteration, it will run the specified estimation model on the first variable (the one that had the fewest missing values) of the data where there were missing values, it will then impute the values predicted by this method and moves on to the next variable where there were missing values and does the same. It will go so on over the whole database for the specified number of iterations. There after is the detailed algorithm explaining the process.

Algorithm 1 MICE imputation algorithm

```
$Y$ a data set of $n$ variables $Y_1, ..., Y_n$ ;
$K$ the number of iterations ;
$\phi$ a chosen estimation model ;
$y_i^{obs}$ observed values, and $y_i^{miss}$ missing values for $Y_i,\ i = 1, ...n$ ;
Sort the $Y$ columns by inscreasing number of missing values ;
First naive completion :
for $i = 1$ to $n$ do
    if $Y_i$ quantitative then
        $y_i^{miss} = \frac{1}{N}\sum_{i=1}^{n} y_i^{obs}$ (average value)
    end
    if $Y_i$ qualitative then
        $y_i^{miss} = \bar{y}_i^{obs}$, whith $\bar{x}$ the mode of $x$ (most common value)
    end
end
Iterative imputation :
for $k = 1$ to $K$ do
    for $i = 1$ to $n$ do
        Adjust $y_i^{obs}$ in function of all the other variables using the chosen model $\phi$.
        Impute $y_i^{miss}$ by the values predicted with the adjusted model.
    end
end
```

The figure 2.26 bellow illustrates that while there is quite a bit of missing data, there seems to be no *pattern* in the missing data, meaning that it is missing at random. We can observe in the rightmost graph, that the pattern always changes in the missing values, for example there never is two variables where the values are always missing together or values that are there only when values of another variable aren't. That is the most important hypothesis in applying the *MICE method*.

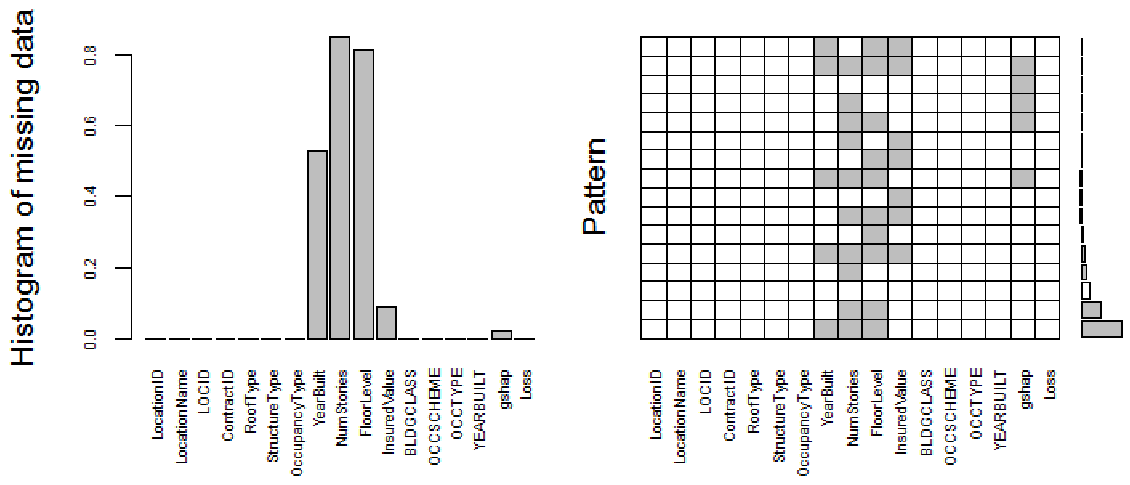


Figure 2.26: Histogram of the missing data with the patterns

With the previous figure we therefore conclude that applying this method is possible. We applied the *MICE method* with the *random forest* model (see section 3.1.2). We chose *random forest* since it is a very useful method for databases that have a lot of qualitative variables such as ours. The number of iterations that we chose is 20. The results of this method were very good, so we used the database modified by *MICE* in the estimation of the *rate*.

Chapter 3

Models theory

In this chapter we will present the models that we will use in the next chapters to predict *pure premium.*

3.1 Presentation of the models

3.1.1 Regression trees

The data consists of p observations of variables X_i and one variable to explain Y, with m modality, observed over n individuals. When the variable to explain is quantitative, the trees are called *regression trees.*

Construction of the tree The construction of a regression tree consists in determining a sequence of *nodes.* A node is defined by the choice of a variable among the predictor variables and a division of the data in two classes. Each node corresponds to a sub-space of the sample.

The division is defined by a threshold value of the quantitative variable chosen or by a modality if the variable is qualitative. At the root there is the whole data ; the procedure is iterates at each of the sub-spaces.

The algorithm needs : a criteria that allows to pick the best division among all the ones that are eligible for the different variables, a rule that allows to decide that a node is terminal (because a tree that is too detailed may lead to over-fitting which is unstable and can result in bad predictions), then it becomes a *leaf*, the algorithm also need the assignment of each leaf to a value (the estimation of the variable to explain).

Division criterion A division is eligible if none of the two nodes that are created from this division is empty. The goal is to divide the individuals into groups that are as homogeneous as possible. The *heterogeneity* can be measured by function that must be : null if the node is homogeneous, and maximal when the values of Y are very dispersed (see the paragraph "criteria of homogeneity").

The division of the node i creates two "offsprings" right and left, they will be noted $(i+1)$ and $(i+2)$, among all these eligible divisions of the node i, the algorithm holds the one that minimizes the sum $C_{(i+1)} + C_{(i+2)}$, with C_i the cost function of the node i (see paragraph "criteria of homogeneity").

Stopping rule and estimations The growth of the tree stops at a given node, which becomes a leaf when the node is homogeneous meaning that there exist no more eligible divisions or when the number of observations in the node is inferior to a chosen threshold. Then at each leaf is associated a value: the means of the observations associated to that leaf.

Criteria of homogeneity We consider the general case of a division in J classes. There are n individuals and J classes of size n_j ; $j = 1, ..., n$ with $n = \Sigma_{j=1}^{J} n_j$, we note $i = 1, ..., n_j$ the individuals of the class j, and y_{ij} the value of Y on the (i, j) individual.

The cost function of the class j is define by :

$$C_j = \sum_{i=1}^{n_j} (y_{ij} - y_{.j})^2, \text{ with } y_{.j} = \sum_{i=1}^{n_j} y_{ij}$$

The cost function of the division is defined by :

$$C = \sum_{j=1}^{J} C_j = \sum_{j=1}^{J} \sum_{i=1}^{n_j} (y_{ij} - y_{.j})^2$$

It is the intra-class variance, also called heterogeneity, that is worth $H = 0$ if and only if $y_{ij} = y_{.j}$ for all i and all j.

The difference in heterogeneity between the non-shared space and the shared-space along the division J is :

$$\delta = \sum_{j=1}^{J} \sum_{i=1}^{n_j} (y_{ij} - y)^2 - \sum_{j=1}^{J} \sum_{i=1}^{n_j} (y_{ij} - y_{.j})^2 = \sum_{j=1}^{J} n_j (y - y_{.j})^2, \text{ where } y = \frac{1}{n} \sum_{j=1}^{J} \sum_{i=1}^{n_j} y_{ij}$$

In our case, for $J = 2$, we get $\delta = n_1 n_2 (y_{.1} - y_{.2})^2$.

The goal at each step is to maximize δ this means finding the variable that gives a division in two classes associated to a minimal intra-class variance.

Pruning The goal is to find an intermediate tree, for this we build a sequence of J pruned sub-trees from the saturated model, which corresponds to a sequence of nested divisions. We obtain this sequence by minimizing a penalized criterion from all trees T pruned from the saturated model T_{max}.

$$\forall \alpha \geq 0,\ Crit_\alpha(T) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_{T,i})^2 + \alpha|T|$$

Where $|T|$ is the number of families (also the number of branches) of T and $\hat{y}_{T,i}$ is the value predicted by T for individual i. By minimizing $Crit_\alpha(T)$ and increasing progressively α, we would receive a sequence of sub-trees having less and less leaves. *Breiman* offers two solutions : it is possible to either use a testing sample independent from the data or proceed by cross validation.

We retain the tree T_j, $j \in \{1, 2, \ldots, J\}$ such as $\frac{1}{n_t}\sum_{i=1}^{n_t}(\tilde{y}_i - \tilde{y}_{T_j,i})^2$ be minimized.

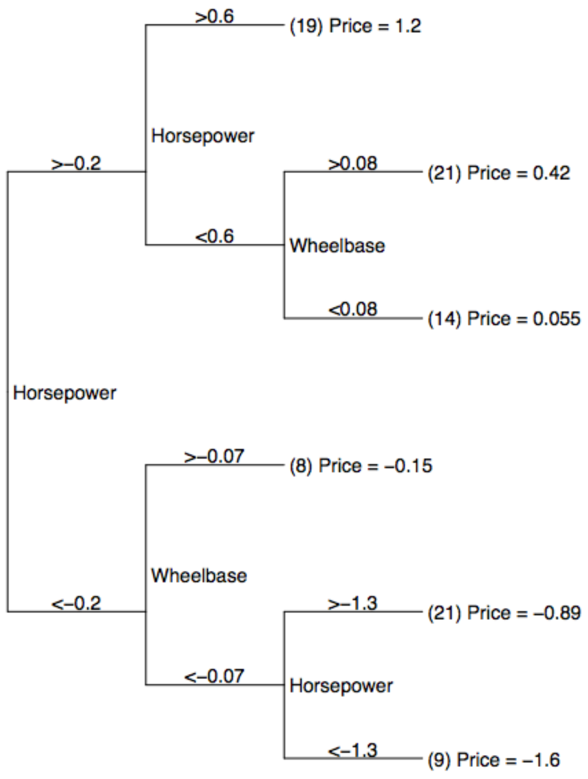


Figure 3.1: Exemple of *regression tree* for predicting price of 1993-model cars

3.1.2 Random forest

In the specific case of *CART (Classification And Regression Trees)* models, *Breiman* proposed a new *bagging* method called *Random Forest.* The main objective was to make the trees more independent from one another by adding a random choosing of the variables in the trees. This approach seems successful in the situation in which the data is highly multidimensional.

Random forest is a machine learning modelization technique that works with standard decision trees and leads it to the next level by combining the trees in order to get different outlook on the data. The random forest give an average of all the nodes that are reached in the several trees.

This model is called random because each of the decision tree will get a random sample of the data and so the trees will each work on their set of data to provide a result and not all the variables are included into the trees so that the different trees might have different variables which offers diversity. Given Y the variable to explain, $X^1, ..., X^p$ the predictor, the random forest algorithm is as follows :

Algorithm 2 Random Forest algorithm

```
$x_0$ the observation to predict ;
$n$ the number of observations ;
$d_n$ the sample ;
$N$ the number of trees ;
$m \in \mathbb{N}^*$ the number of variables randomly sampled.
for $k = 1$ to $N$ do
    Pick a bootstrap sample (random sample with replacement) in $d_n$.
    Build a tree on this bootstrap sample on a set of m variables randomly sampled from $p$
    variables, we write $h(., k)$ the tree built.
end
Result: $h(x_0) = \frac{1}{N} \sum_{k=1}^{N} h(x_0, k)$.
```

The fact that only m predictors are chosen at each step can increase considerably the diversity of the trees by putting forward other variables. Each regression tree is, of course, useful but the combination of all the trees makes this model even stronger. The choosing of the variables m is very sensible and must be done well. The default parameters is : $m = p/3$ (in the regression case).

One of the advantages of a random forest is that it can deal with the missing data and still manage to be very accurate, the bias in this algorithm is the same as the one for a single decision tree and finally it can give an estimation of the most important variables in the database.

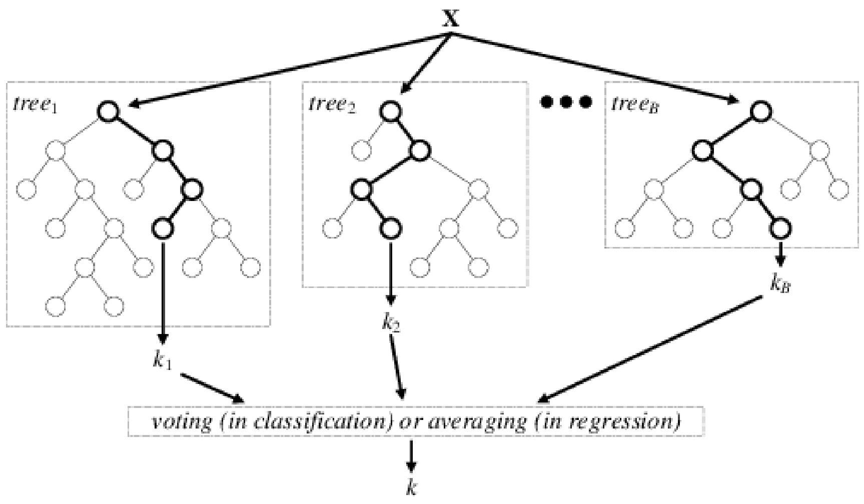


Figure 3.2: Example of *random forest*

3.1.3 Generalized Linear Model

The *Generalized Linear Model (GLM)* is a generalization of the linear model, it is a smoother version of the linear regression. However, in the *GLM*, the errors can have a different distribution model than the normal distribution. In contrast with the linear model, the *GLM* uses a *link function* to transform the variable to explain. In a *GLM*, the dependent variable Y is generated by a distribution in the exponential family (such as normal, binomial, poisson and gamma distributions). The link function gives us a relationship between the mean of the distribution and the linear parameter.

The transformed variable to explain can be written : $f(y) = \beta_0 + \sum \beta_i x_i + u$, where f is the link function. The models belong to the Generalized linear models are composed of three components :

Random Component The random component is the probability distribution of the variable to explain. It is under the form :

$$f(y, \theta, \phi) = \exp\left(\frac{y\theta - v(\theta)}{u(\phi)} + \omega(y, \phi)\right)$$

The distribution of Y comes from an exponential family. The parameter θ is called natural parameter, ϕ is the dispersion parameter that is used to adjust the variance of the model to the observed one.

Deterministic component The linear predictor or deterministic component of the model is the vector with n components :

$$\eta = X\beta = \beta_0 + \sum_{i=1}^{p} X_i\beta_i$$

Where $\beta = (\beta_0, \beta_1, \beta_2, ..., \beta_p)$ is the vector of unknown coefficients that we want to estimate, β_0 is a constant and X_i the predictor variables.

The link function The expectation of the random part $E(Y|X = x)$ depends on the deterministic component $\eta = X\beta$ through a link function, that is monotonous and differentiable.

$$g(E(Y|X = x)) = \eta = X\beta$$
$$E(Y|X = x) = g^{-1}(\eta) = g^{-1}(X\beta)$$

3.1.4 Logistic regression

Logistic regression is a binomial regression model, as in other regression models, its purpose is to explain the effect of a vector X on a variable to explain Y. However in a logistic regression the variable Y is a binomial variable, a qualitative variable with two modalities: success or failure. The values X don't have to be binary. The objective is to try and explain the probabilities :

$$\pi = \mathbb{P}(Z = 1) \text{ or } 1 - \pi = \mathbb{P}(Z = 0)$$

In order to do that, we take a function $f : [0, 1] \to \mathbb{R}$, we therefore look for a linear model that has the form $f(\pi_i) = x_i\beta$ There are three functions that fulfill the previous demands.

- The first one is *probit*, then f is the inverse of the Gaussian distribution function,
- The second one is *log-log* with :

$$f(\pi) = \log(-\log(1 - \pi))$$

- The last one is *logit* :

$$f(\pi) = logit(\pi) = ln\left(\frac{\pi}{1-\pi}\right), \text{ with } f^{-1}(x) = \frac{e^x}{1+e^x}$$

The model the most commonly used is the model with the function *logit*, because it brings simplifications in the estimation, it is a generalized linear model with the link *logit*.

3.1.5 Support Vector Machine

The *Support Vector Machine (SVM)* is a classification method. It was initially a model that predicted a binary variable, meaning a discrimination into two classes, this model has generalized and is now adapted to predict a quantitative variable. It consists in finding an hyper-plan that divides and classifies the data at best. The goal is therefore to find a *kernel function* where the quality of prevision is as best as possible.

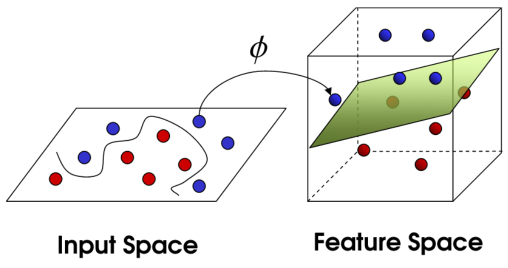


Figure 3.3: Example of *SVM*

When the data is not linearly divisible, it can be possible to find a non linear border. The first step of this method consists in transforming the variable into one that can be linearly divisible.

For when we consider the images of our observations through a non linear application ϕ in a space H with a larger dimension than the initial space, with the help from a kernel function which gives the entry vectors (vector of predictor variables) in a subset where they are linearly divided. The kernel function can be written:

$$k(x, x') = \phi(x).\phi(x')$$

Where k is the kernel function and $\phi : \mathbb{R}^n \to H$ is a function that has as starting space n predictor variables end for ending space the sub-vector space H where the data is linearly divisible. One of the main principle of this model is to integrate a complexity control in the estimation. this means controlling the number of parameters (or support vectors) associated.

These conditions insure that the symetric function k is a kernel if for all x_i possible, the $k(x_i, x_j)$ is a positive definite matrix. In this case we show that there exists a space H and a function ϕ such as $k(x, x') = \phi(x).\phi(x')$.

Commonly used kernels :

- Polynomial kernel : $k(x, x') = (cx.x')^d$, $c \in \mathbb{R}$,
- Gaussian kernel : $k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$, $\sigma > 0$,
- Hyperbolic kernel : $k(x, x') = \tanh(c_1 x.x' + c_2)$, $c_1, c_2 \in \mathbb{R}$.

3.1.6 Gradient Boosting Model

In 2002, *Friedman* suggested a boosting algorithm based on a loss function assumed differentiable called *Gradient Boosting Model (GBM).* The basic principle is to build a sequence of models in a way that at each step, each model added to the combination seems like a step to a better solution. The principal innovation of this model is that this step going in the direction of the gradient of the loss function which is approximated by a regression tree. The algorithm of this method is as follows :

Algorithm 3 Gradient Boosting Model algorithm

```
$x_0$ the observation to predict and $\{(x_i, y_i)\}_{i=1}^n$ a sample ;
$\phi(x)$ a model function of $\{x_1, ..., x_p\} \in \mathbf{R}^p$ ;
$Q(y, \phi(x)) = exp[-y\phi(x)]$ the loss function ;
$M$ the number of iterations ;
$\eta \in ]0; 1]$ the shrinkage parameter ;
Initialize $\hat{\phi}_0 = \arg\min_{\gamma} \sum_{i=1}^{n} Q(y_i, \gamma)$.
for $m = 1$ to $M$ do
    Calculate $r_{im} = -\left[\frac{\delta Q(y_i, \phi(x_i)}{\delta\phi(x_i)}\right]$ for $i = 1, ...n$.
    Adjsut a regression tree $h_m(x)$ to $r_{im}$ giving the terminal leaves, i.e train using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
    Calculate $\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} Q\left(y_i, \hat{\phi}_{m-1}(x_i) + \gamma h_m(x_i)\right)$.
    Update : $\hat{\phi}_m(x) = \hat{\phi}_{m-1}(x) + \eta.\gamma_m.h_m(x)$.
end
Result: $\hat{\phi}_M(x_0)$.
```

The algorithm is initialized by a constant, meaning a tree with only one leaf. The term of the gradient is simply the same as calculating the residuals r_{mj} of the model at the previous step. the correctional terms γ_{jm} are optimized for each tree.

How this model works is that it builds trees one by one and the predictions of each of the trees are summed, the next decision tree is built to try and cover the discrepancy between the current prediction and the real value

This kind of method are called boosting unlike the previous method we talk about such as random forest which are bagging methods. Boosting methods have the same general principle as bagging method, it is the building of a family of models which will averaged to get an estimation of the result. The most noteworthy difference is on the way to build the family of models. In the boosting models, the family are created by recurrence where each model is an adapted version of the previous one and gives more weight at the next estimations to the entries that are poorly adjusted or poorly predicted. The perk of boosting models are that they allow us to avoid over-fitting. The boosting methods can reduce the variance and usually give better results than bagging methods. However only by comparing the errors in boosting and bagging method can optimize the choice of a method for a specific dataset. The *GBM* model can be very effective especially when the shrinkage coefficient is well adjusted.

3.1.7 Extreme Gradient Boosting

The *Extreme Gradient Boosting (XG-Boost)* is based on the GBM model. The main difference between the two models is that the XG-Boost model has more regulation to control the over-fitting which usually gives it a better performance than GBM. This method is used for supervised learning problems, so where we use a training sample with predictor variables X_i to predict Y_i. The tree boosting model is written :

$$\hat{y_i} = \sum_{k=1}^{K} f_k x_i \text{ , } f_k \in F$$

Where F is the ensemble of regression trees possible and K the number of trees. In order to find the best parameters we need to define an objective function which will measure the performance of the model. The objective function must contain training loss and regularization. The objective function is :

$$Obj(\theta) = \sum_{i=1}^{n} d(y_i, \hat{y_i}) + \sum_{k=1}^{K} \beta(f_k)$$

Where d is the loss for each value and $\beta(f_k)$ is the regularization associated to each tree. The loss function measures how predictive the model is. It is commonly the mean square error. The regularization term is there to control how complex the model is and can help us prevent over-fitting.

The goal of the machine learning is to find out which parameters are needed for the model by starting from the parameters of the trees. For that we must find f_k. Since we cannot find all the parameters of the trees at once we will use additive training : from what we have already learned, add trees to the model one by one, $\hat{y_i}^{(t)}$ is the predicted value at the step t where $t \in [0, K]$:

$$\hat{y_i}^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y_i}^{(t-1)} + f_t(x_i)$$

Since the goal is to minimize the objective function, during each step we will choose the tree that minimizes the objective function.

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} d(y_i, \hat{y_i}^{(t)}) + \sum_{k=1}^{t} \beta(f_k) \\ &= \sum_{i=1}^{n} d(y_i, \hat{y_i}^{(t-1)}) + f_t(x_i) + \beta(f_t) \end{aligned}$$

In order to optimize it, we first need to use a Taylor expansion. The result is :

$$Obj^{(t)} \approx \sum_{i=1}^{n} \left(g_i.f_t(x_i) + \frac{1}{2}.h_i.f_t^2(x_i) \right) + \beta(f_t)$$

With :

$$\begin{aligned} g_i &= \partial_{\hat{y}_i^{(t-1)}} d(y_i, \hat{y}_i^{(t-1)}) \\ h_i &= \partial^2_{\hat{y}_i^{(t-1)}} d(y_i, \hat{y}_i^{(t-1)}) \end{aligned}$$

We must now find the regulation factor that will penalize too complex models. $f_t(x_i) = c_{q(x)}, q \in R^T, q : R^d \to \{1, 2, \cdots, T\}$. In this algorithm, we penalize linearly the number of leaves :

$$\beta(f_t) = \gamma.T + \frac{1}{2}.\lambda \sum_{j=1}^{T} c_j^2$$

The parameters λ and γ allow us to adjust the regularization term.

Algorithm 4 XG-Boost algorithm

```
$x_0$ the observation to predict ;
$T$ the number of iterations ;
Initialization $\hat{y_i}^{(0)} = 0$.
for $t = 1$ to $T$ do
    Calculate $g_i = \partial_{\hat{y}_i^{(t-1)}} d(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial^2_{\hat{y}_i^{(t-1)}} d(y_i, \hat{y}_i^{(t-1)})$.
    Build a tree $f_t(x)$ by starting with an empty tree and for each node find the better division according to each variable, and add the best of them, if it minimizes the objective function.
    Add $f_t(x)$ to the model $y_i^{(t)} = y_i^{(t-1)} + \epsilon f_t(x_i)$ where $\epsilon$ is the reduction parameter in the adding of trees in the model.
end
Result: $y_i^{(T)}$.
```

At each step, the magnitude of the adjustment decreases. The shrinkage factor penalizes the addition of another model, to slow down learning and avoid over-learning.

3.2 Performance metrics

3.2.1 Root Mean Squared Error

We will be evaluating our models based on their *Root Mean Squared Error (RMSE)*, meaning that in order to determine which model fits best our data, we will pick the one with the lowest *RMSE*. The formula for the *RMSE* is the following :

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y - y_{pred})^2}$$

3.2.2 Lorenz curve

To compare our models we used the *Lorenz curve*. In the case of insurance modeling, it is a graphic that shows, on the x axis the cumulative percentage of population sorted by predicted pure premium (or the risk) and on the y axis the cumulative percentage of *pure premium*.

Let y_i be the pure premiums of the individual $i = 1, ..., n$ and $\hat{y}_i$ the pure premium estimations given by the model. To map the *Lorenz curve*, we first sort the n pure premiums y_i by their associated predictions so that :

$$\hat{y}_1 \geq \hat{y}_2 \geq \cdots \geq \hat{y}_n$$

With that ordering (from high risks to low risks, based on our predictions), we plot the *Lorenz curve* :

$$\left(\frac{i}{n}, \frac{\sum_{k=1}^{i} y_k}{\sum_{k=1}^{n} y_k}\right)$$

We can see an example of the *Lorenz curve* in figure 3.4. The x and y axis are graduated with numbers between 0 and 1 (or 0 to 100%). The curve labeled "perfect pricing" is plotted using the true pure premium values instead of predictions. The curve labeled "average pricing" represents the lower bound. All the other curves represent the different models. The curve labeled "model 1" is the best model because it is the closest to the "perfect pricing" curve, so the predictions are better. The curve labeled "model 2" is the worst of the models because it is closer to the average pricing. In the graph, we can see that with a certain percentage of population sorted by risk, we can explain a certain percentage of the pure premium.

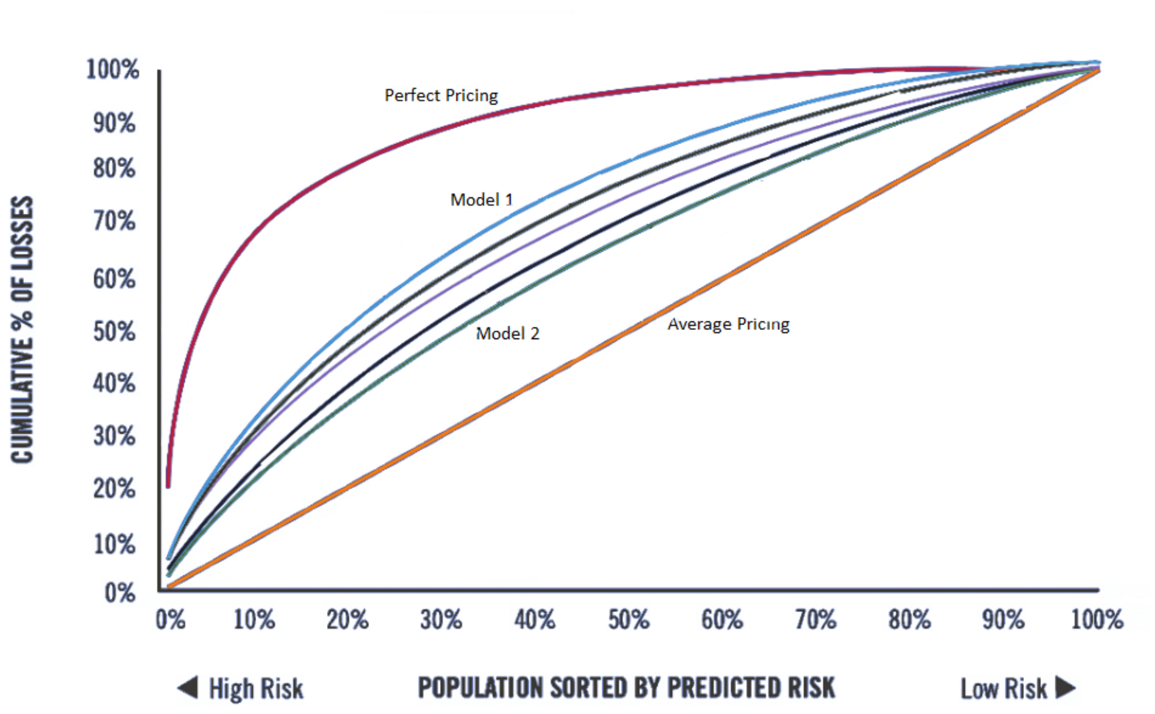


Figure 3.4: Example of a *Lorenz curve*

3.2.3 Receiver Operating Characteristic curve & Area Under Curve

The *Receiver Operating Characteristic curve* was first used during world War II to detect signals. We now use the *ROC curve* to plot true positive rates against false positive rates in order to determine if a model is a good model. To plot the curve we first must know what *true and false positives* are as well as *true and false negatives.*

- "True Positive", TP : when the observation is 1 and the estimation of the model is 1.
- "False Positive", FP : when the observation is 0 but the estimation of the model is 1.
- "False Negative", FN : when the observation is 1 but the estimation of the model is 0.
- "True Negative", TN : when the observation is 0 and the estimation of the model is 0.

$P = TP + FN$ is the number of positives,
$N = TN + FP$ is the number of negatives.

We call the *sensibility* : $Se = \dfrac{TP}{P} \in [0, 1]$

We call the *specificity* : $Sp = \dfrac{TN}{N} \in [0, 1]$

We obtain the *ROC curve* by calculating $Se(s)$ and $1 - Sp(s)$ for s varying between 0 and 1 where s represents the different cut-off points. On the x axis is represented the false positive rate and the y axis the true positive rate. Each point on the *ROC curve* curve represents a sensitivity/specificity pairing that is associated to a particular decision threshold (s). The *ROC curve* curves are insensitive to class imbalance, they can allow us to choose the best trade-off between False Positive rate and True positive rate.

The *Area Under Curve (AUC)* represents the area under the *ROC curve*, so $AUC \in [0, 1]$. The greater the *AUC* is, the better the performance of the model. It can allow us to compare two models with a numeric value. An example of *ROC curves* representing different models can be seen in figure 3.5 with their corresponding *AUC*.

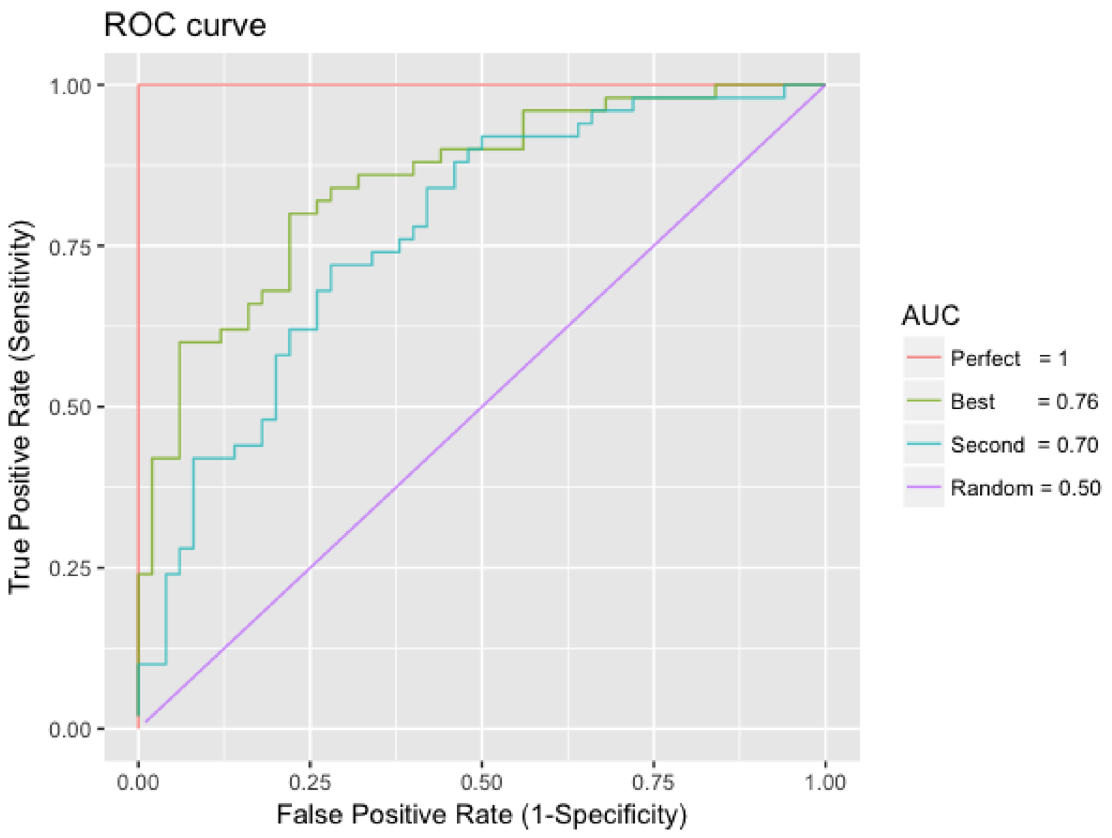


Figure 3.5: Example of *ROC curves*

Chapter 4

Pure premium estimation

One of the main goals of our project is to explain the *pure premium*. The equation for pure premium is :

$$PurePremium = Loss \times frequency$$

We have tried doing this with a few different models in attempt to find the model that is best adapted to our data. Most of the models that we have used, we learned at the *EURIA* such as *linear regression*, *regression trees* and *random forest* however the boosting models, as we did not study them in class, required deeper research to fully grasp.

In this chapter we will see how we fitted each model, and explain if it is or not a good model to predict the data. To estimate the *pure premium* we used two different techniques, the first one was to estimate the *loss* directly and the second one was to estimate the *rate*, which is a function of *loss* (explained in section 4.2). To compare the adequacy of the models used we will compare their *Root Mean Squared Errors*, their *Lorenz curve* and their *ROC curve*, the most significant of these results will be represented in section 4.3.

All the simulations of the different model were done through the software *R*. We decided to use this software as it is the one that we are most familiar with, and the functionalities are very developed especially for *data science*.

To estimate the Pure Premium, we divided the database into two parts, the first sample contained 70% of the database and will be used as the *training sample* and the other 30% will be used as the *testing sample*. We will use the notation $RMSE_{train}$ for the *RMSE* on the training sample and $RMSE_{test}$ for the *RMSE* on the testing sample.

4.1 Loss estimation

In order to calculate the *pure premium*, we will estimate the *loss* and then multiply it by the *frequency*. Estimating the *loss* was done without the imputed data. At this stage of our project the missing values were replaced by "VM" and the lines where the gshap had no value were deleted. All the following models to estimate the *loss* were done with this modality “VM”.

In order to evaluate well the RMSE it is necessary to have the order of magnitude of the variable *Loss*. The basic information of the variable *Loss* are summarized in the table below.

min	1^{st} quantile	median	mean	3^{rd} quantile	max
24	$51,010$	1.73×10^6	10.21×10^6	5.314×10^6	2.474×10^9

4.1.1 Linear regression

To apply the generalized linear model, we need to choose a distribution for the loss. We choose a normal distribution. We suspect that it will fit better to the logarithm of the loss because as we have discussed in section 2.2 the Log-Loss looks like a normal distribution. In our model, the variable Y that we are trying to predict is *Loss*, the predictor variables that we have selected (to avoid new levels problems) are *RoofType*, *StructureType*, *OccupancyType*, *YearBuilt*, *InsuredValue*, and *gshap*.

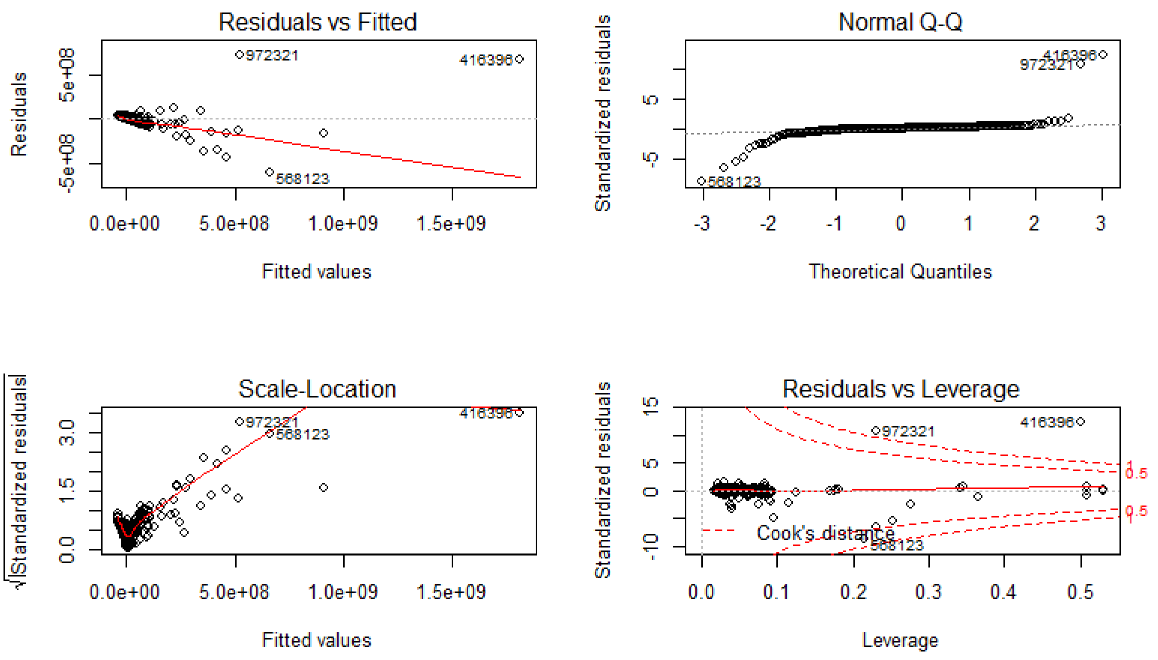


Figure 4.1: Plots of the *linear regression*

As a result, we get $R^2_{adj} = 0.735$ that is satisfying for the model knowing that this is a first non-optimized model where not all of the predictors variables are present. In figure 4.1 we can see the four different graphs.

The first graph (top left) represents the fitted model against the actual model, it shows us whether the *residuals* have linear patterns or not. We can notice that the relationship is not actually linear, there are too many points that are not even close to the fitted line.

The second graph (top right) represents the *QQ-plot* or quantile plot, it shows us whether the *residuals* follow or not a normal distribution. However we can see that the values on the left of the graph are a little under the dotted line while on the right side of the graph they are well above the dotted line, it indicates us that the tail is heavier than the one on a normal distribution, therefore, one of the most important hypothesis of the *linear regression* model is not validated.

The third graph (bottom left) is used to check whether the variance of the *residuals* depends on the linear relationship so if the residuals are spread equally. However we can notice that the residuals seem to spread wider after a certain point, we can thus make the assumption that the variance is not equal, it is *heteroscedastic.* That contradicts another important hypothesis of the *linear regression* model.

The final graph (bottom right) is meant to show *outliers*, as they can be influential. The results can be quite different if we decide to include them or not. This plot is used to watch out for *outlying values.* If a value is outside of the *Cook's distance* (the red dotted lines) it means that it cannot be taken out of the model because it is influential, this mostly shows us that we have to be careful in the future models due to this *outlier.* We can see on our graph that they are many *outliers* in our model.

We therefore reject the model because the hypothesis are not satisfied, but since we saw that the tail was too heavy on the second graph, we decided to try a model on the *log-loss* because, as seen in figure 2.6 the log loss looks more like a *normal distribution.*

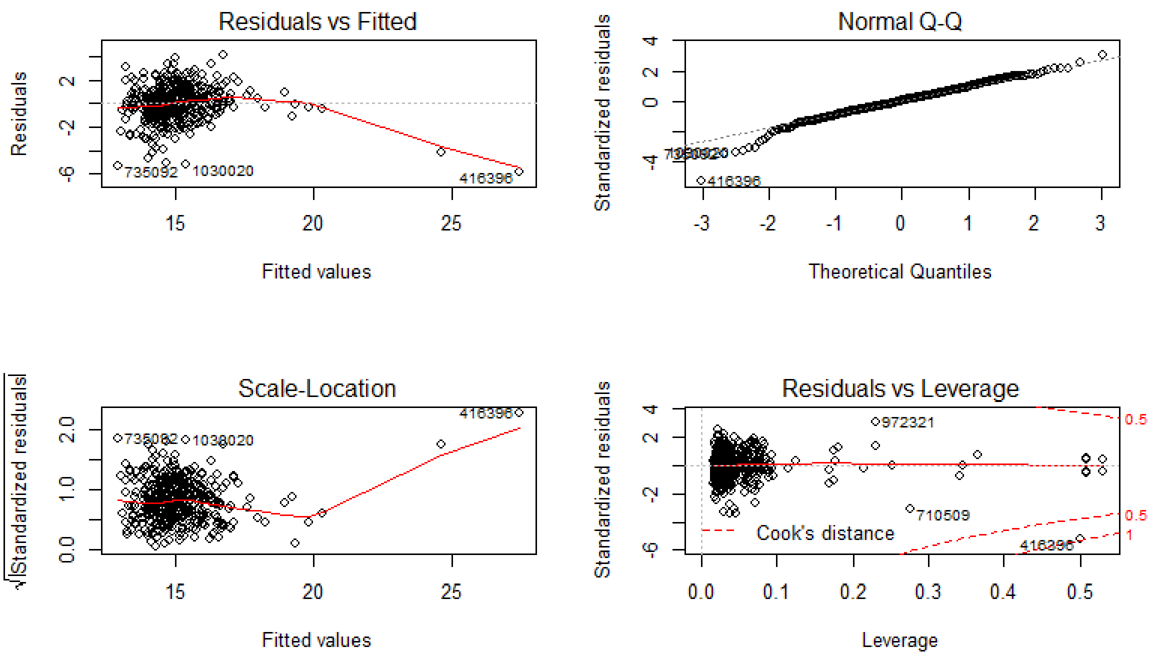


Figure 4.2: Plots of the *linear regression* on the *Log-Loss*

In figure 4.2, in the first graph we see that the *residuals* seem linear except for the two *outliers* on the right of the graph. On the second graph we see that the *tail* corresponds better to a *normal distribution* than it did in figure 4.1, however it doesn't really fit on the bottom left, the tail is still a bit heavy. In the third graph, it looks pretty constant except for the same two *outliers*. Finally in the fourth graph, we see that there is one outlier outside of the *Cook's distance*. In conclusion, with the *log-loss* the hypothesis seem more satisfied but we have a worse $R^2_{adj}(0.3794)$, we still reject it, indeed, because of new levels we could not better the model by adding the other variables and we cannot calculate the *RMSE*, which is principal performance metrics we chose to compare our models.

4.1.2 Regression trees

The next method we tried, to get an estimation of the *loss*, is *regression trees*. When we applied this model to our data, every time we would run the code, the results were different, the regression tree we got as results would vary and use different parameters. Also one of the problems with this method was that if one of the predictor variable was too important, it would use it to split the set at each node.

As we can see in the following figure (figure 4.3) at each node except the last on on the right, the split is done by *InsuredValue*, so a lot of other predictor variables are not included in the result. This figure represents the tree before pruning.

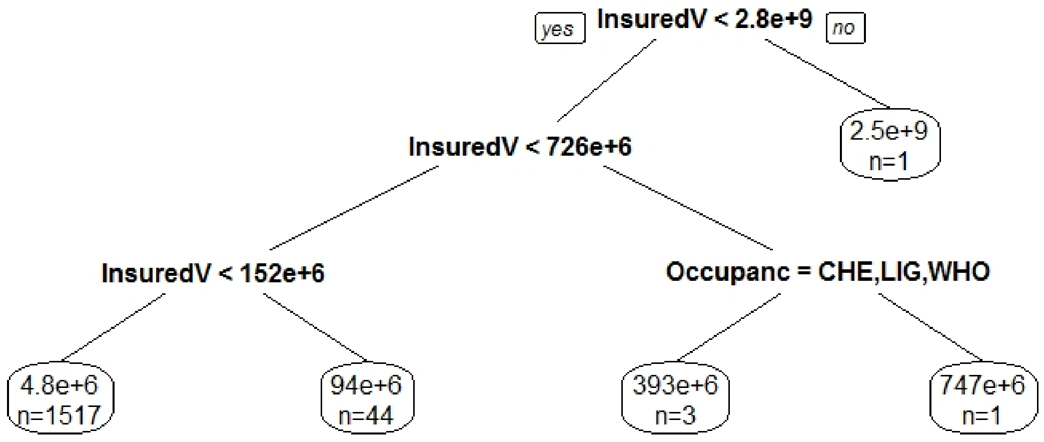


Figure 4.3: Regression tree for *Loss* before pruning

To know when it is better to *prune* the tree we created the graph in figure 4.4. It plots the the error of the regression in function of the size of the tree, *cp* is the set of possible *cost-complexity* pruning for a nested sequence. Here we see that the error is smaller when the tree has a size of four, and the *cp* is pretty small, it would not decrease much if we picked five. We therefore decided to *prune* the trees so that it would only have four leaves.

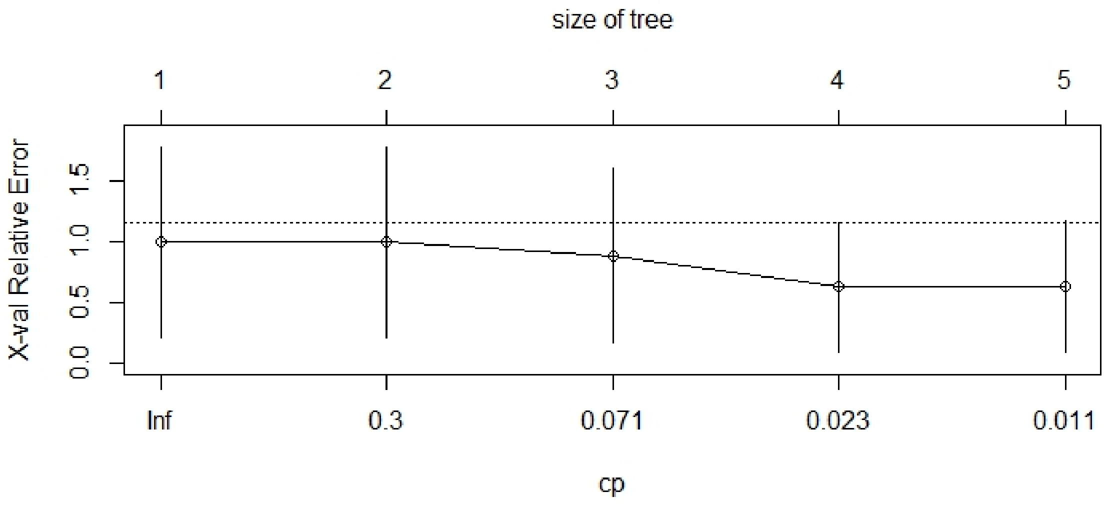


Figure 4.4: Error of the regression tree in function of the tree size

The figure 4.5 represents the *pruned tree* with only four leaves. In this tree the split at each node only uses the *insured value.*

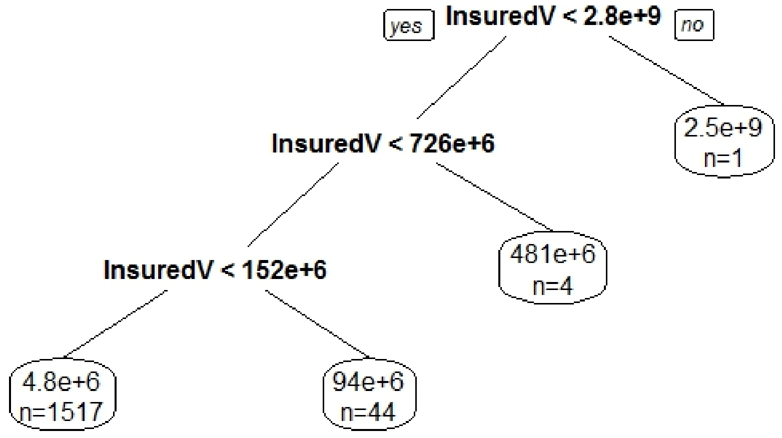


Figure 4.5: Pruned regression tree for *Loss*

The results we obtain with the *regression trees* are quite bad :

$$RMSE_{train} = 27,320,343\ ;\ RMSE_{test} = 67,171,269$$

They depend too much on the *training sample* and on the choice of the parameters. The error might be also due to the too high significance of *InsuredValue* in the trees.

We speculated that the random forest method would give better results because of the randomization effect. It would bring more diversity in the variables used in the trees and therefore give a stronger model.

4.1.3 Random Forest

After having adjusted the *random forest model* to our data, with the default parameters $m = 3$ (number of variable randomly sample) and 10 trees we get the following results :

$$RMSE_{train} = 21,491,152\ ;\ RMSE_{test} = 45,543,067$$

As we suspected the results of the *random forest* are better and more stable than the ones of the regression tree. Indeed the *RMSE* are lower, don't vary as much when we change the parameters and don't depend as much on the training and testing sample.

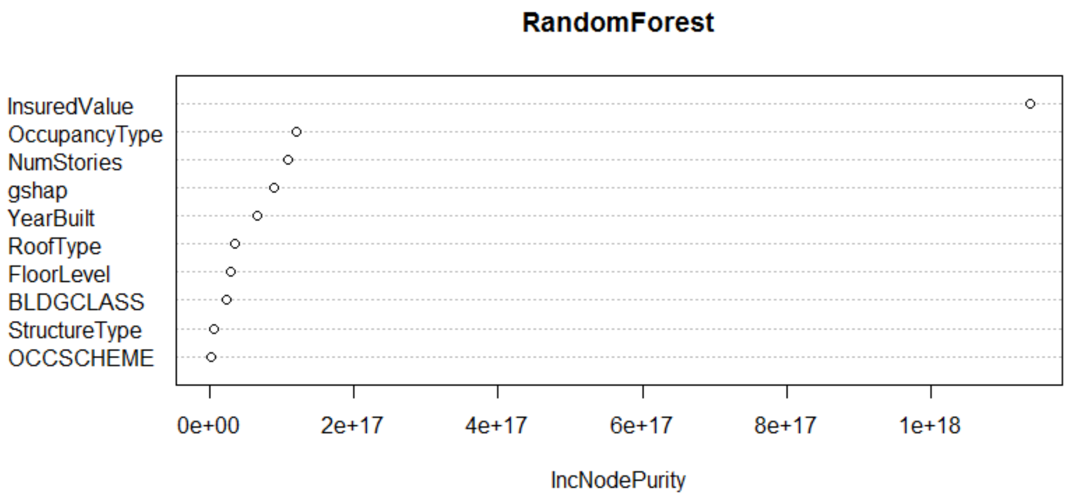


Figure 4.6: Significance of the variables in the random forest

In figure 4.6, we can observe the significance of the different variables in the random forest model. As we can see in this graph, the most relevant variable in explaining the *loss* is again *InsuredValue*, other relevant variables are *YearBuilt* and the *gshap*. Some other variables are relevant but four times less than the *InsuredValue*.

We can see in figure 4.7, that errors are very significant for *extreme values*, we therefore thought it would be better if we tried to adjust two different model : one for *extreme values* and one for average values, we will explain it in the following section.

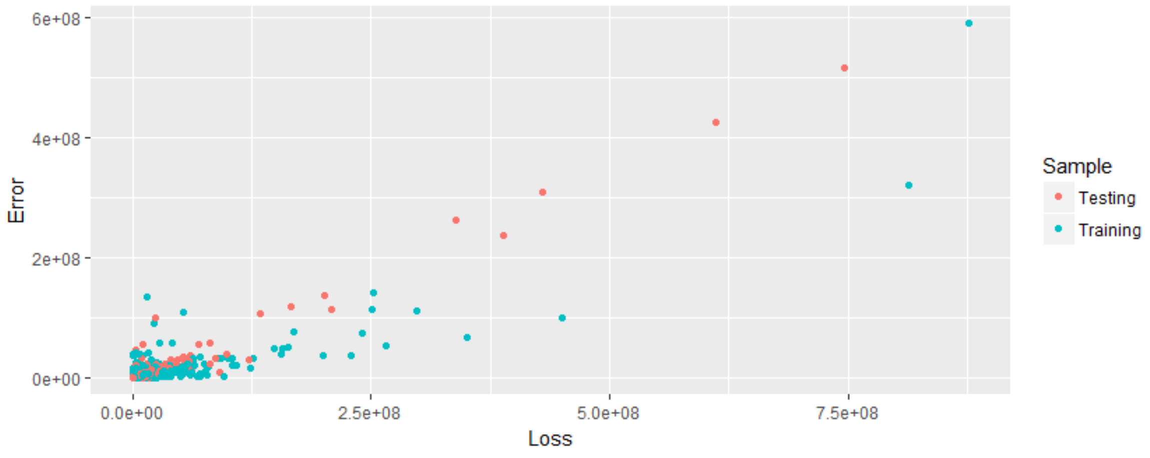


Figure 4.7: plot of the loss against the error

4.1.4 Logistic regression coupled with random forest

As we further worked on our project we noticed that maybe the average losses and the significant losses didn't follow the same model and that is why our error was always too significant. We therefore decided to divide the losses in two, we created a new variable. The very large losses above a specific threshold were affected the number one while the smaller or average losses were given the number zero. To choose the value above which an observation will be considered as extreme or not we use the *mean residual life plot*, we have to chose the threshold as big as possible so that the values above will really be extreme, but not too big to be able to adjust a random forest on the extreme values.

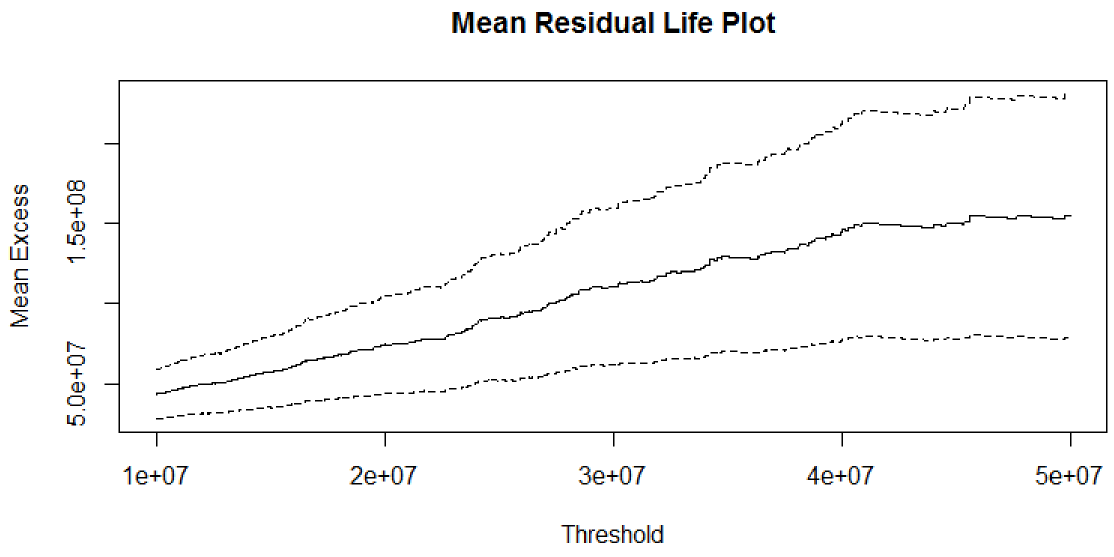


Figure 4.8: Mean residual life plot to find the threshold

This graph is based on the stability by censoring of the GPD Generalized Pareto Distribution (GPD), let X be the random variable that describes *loss*. We remind that :

$$X \sim \text{GPD}(\sigma\ , \kappa) \ \Leftrightarrow \ F_X(x,\sigma,\kappa) = \begin{cases} 1 - \left(1 + \dfrac{\kappa x}{\sigma}\right)^{-1/\kappa} & \text{for } \kappa \neq 0, \\ 1 - \exp\left(-\dfrac{x}{\sigma}\right) & \text{for } \kappa = 0. \end{cases}$$

The censoring of the GPD follows the principle : if $\mathcal{L}(X - u_0 \mid X \geq u_0) \simeq \text{GPD}(\sigma_{u_0}\ , \kappa)$,
Then, for $u \geq u_0$: $\mathcal{L}(X - u \mid X \geq u) \simeq \text{GPD}(\sigma_u\ , \kappa)$ with $\sigma_u = \sigma_{u_0} + \kappa(u - u_0)$.
The empirical mean residual life plot is the plot of:

$$\left(u, \frac{1}{n_u} \sum\nolimits_{i=1}^{n_u} (x_{(i)} - u)\right)$$

Where $x_{(1)}, \dots, x_{(n_u)}$ are the n_u observations that exceed the threshold u. If the exceedances of a threshold u_0 are GPD, the empirical mean residual life plot should be approximately linear for $u > u_0$ according to the stability by *censoring principle*. The confidence intervals in the plot are symetric intervals based on the approximate normality of sample means.

In figure 4.8, we can see that an appropriate threshold to distinguish the extreme values from the normal values is 3.10^7. This threshold holds 120 observations above it.

We then used a logistic regression to make a model to predict whether the value will be extreme or not. Once the logistic model has been applied to our data, we applied the *random forest* model to a data-set with only losses with the number one given by the logistic regression to give an estimation for the extreme claims. We applied another forest for when the predicted value given by the logistic regression was zero. It gave us two separate *random forest* to predict the values.

One of the major step in logistic regression is to optimize the *threshold* for the logistic regression. We can see in the two following figures the error in function of the threshold on the training database and on the testing database.

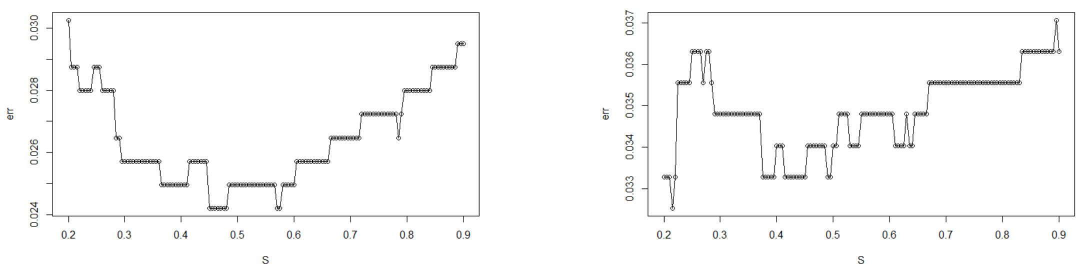


Figure 4.9: Optimization of logistic regression threshold

In figure 4.1.4, the left graph is the error in function of the *threshold* in the training database and on the right it is on the testing database. We can see that the default *threshold*: 0.5 is the most adapted to our data since it minimizes the error in both graphs. Once this *threshold* is chosen we worked the logistic regression and as we can see in figure 4.10 the result is very good. The *ROC curve* for the testing sample is almost perfect and we find an AUC of 0.98.

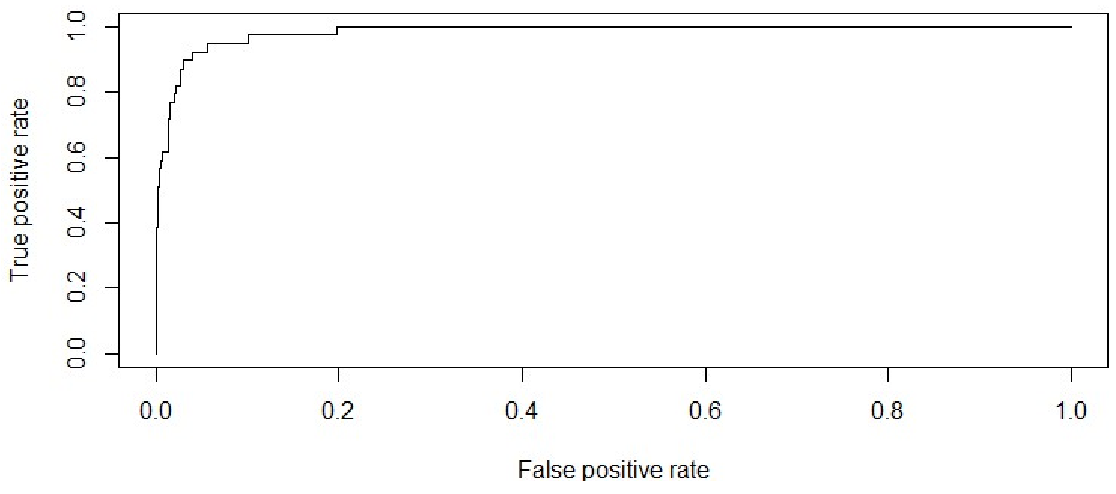


Figure 4.10: ROC curve on the testing sample

In figure 4.11, we can see the significance of the different variables in the *random forest* ran on earthquakes that were given the value zero in the logistic regression. As suspected, the *InsuredValue* is still the most significant variable in this model, the other significant variables are *OccupancyType*, *YearBuilt*, and *NumStories* we see here the *InsuredValue* is at least still twice as significant as the other variables and *OccupancyType* which was not very significant in the normal *random forest* and is quite significant in this model.

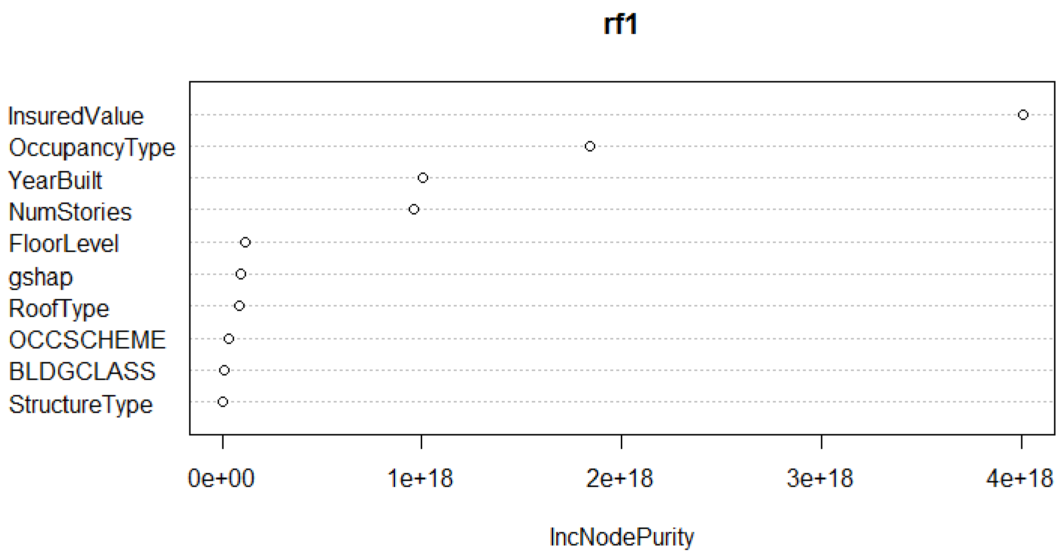


Figure 4.11: Significance of variables for average values

In figure 4.12 we can see the significance of the different variables in the *random forest* ran on earthquakes with *extreme values*, so that were given the value one in the *logistic regression*. In this second random forest, the *InsuredValue* is still the most significant variable, the other significant variables are *gshap*, *OccupancyType*, *YearBuilt*, and *NumStories*. The *InsuredValue* here is very significant, almost three times as significant as the other variables and the *gshap* is also very significant, that is probably due to the fact they they are almost directly related to the loss.

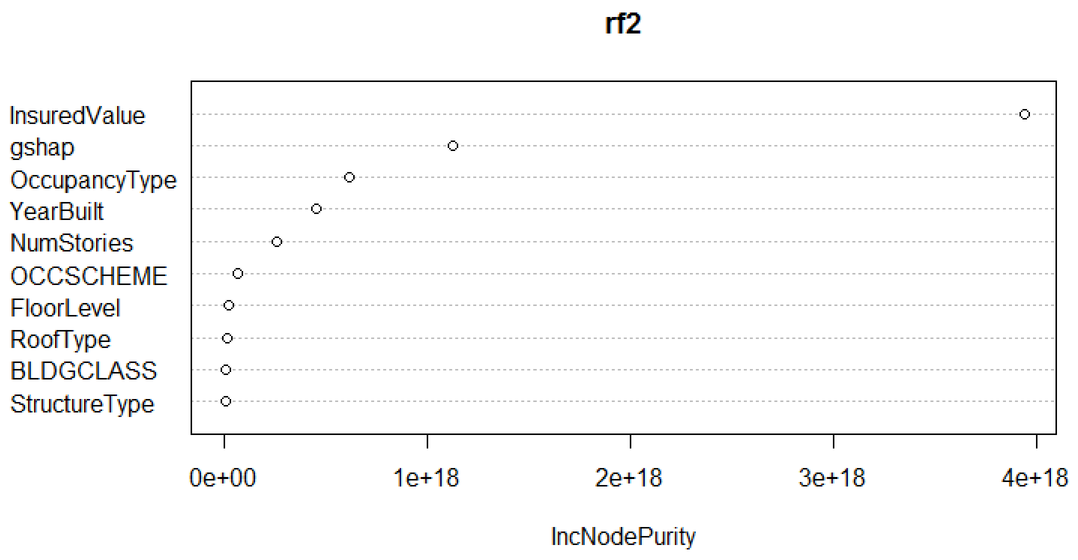


Figure 4.12: Significance of variables for extreme values

After working the whole model (with the random forests) we get :

$$RMSE_{train} = 67,726,643 \ ; \ RMSE_{test} = 75,861,793$$

The result have decreased in efficiency in this model, this is due to the fact that if the *logistic regression* predicts a normal value to be extreme, it will run the *random forest* for *extreme values* on it and predict a value so much larger for this value that the error just for this value will be huge. That is why the error is so large in this model.

At this point we realized that estimating the *loss* might not be the best solution to estimate the *pure premium* and that insured value was always too significant, that is why we decided to estimate the *rate*.

4.2 Rate estimation

As we have seen in the previous section, it is hard to estimate the *loss*. Another problem with estimating the *loss* was that to get the *pure premium*, we would have to estimate the *frequency* as well, which would have given worse results. We therefore decided that estimating *pure premium* might be a better approach but once again, the relation of *pure premium* with insured value was too close, which overshadowed the other variables. In order to remove the importance of the *insured value* we decided to modelize the *rate* :

$$Rate = \frac{PurePremium}{InsuredValue}, \text{ with } PurePremium = Loss \times Freq$$

Hence, in this section of our report we will present the models we used to predict the *rate* variable, so we decided to reorganize our database by dividing it into two datasets :

– The first being *PPbystite*, this database was meant to be used in the data science and modelization part of our project. Since the goal is to modelize *PurePremium*, we added a column *pure premium* calculated with *loss* and *frequency*, and since *loss* and *frequency* were not useful anymore we deleted them. The variables included in this database are *LocationID*, *Rate*, *BLDGCLASS*, *OCCSCHEME*, *OCCTYPE*, *YEARBUILT*, *gshap*, *RoofType*, *SctureType*, *OccupancyType*, *YearBuilt*, *NumStories*, *FloorLevel.*

– The second database is called *ELT*, it is the *Event Loss Table*, the variables it contains are : *EventID*, *Loss*, *Freq.* This database will be used later in the project in the reinsurance section.

Since the *rate* variable is very small, it is hard to interpret well the RMSE of *rate*, we therefore decided that once we estimated the *rate*, we multiplied them by the *insured value* in order to get the *pure premium* and we compared the *RMSE* based on the *pure premium.* In order to evaluate well the *RMSE* it is necessary to have the order of magnitude of the variable *pure premium.* The basic information of *pure premium* are summarized in the table below.

min	1^{st} quantile	median	mean	3^{erd} quantile	max	standard deviation
0	19	106.2	966.1	408.8	$323,369.2$	$7,778$

In this section, we had deleted the observations without *InsuredValue* and performed the imputation of the data with the *MICE method.* We did not delete *gshap* this time because the *MICE method* predicted it well, thus the models presented next are done on a database with no missing values and on the entire data (except for the few insured value missing that we deleted). We also changed the way we processed the data by using the *R* package *data table.*

4.2.1 Random forest

As the *regression trees* did not give us such good in the estimation of the *loss*, we decided to start the *rate* estimation directly with a *random forest*. The parameters we choose for this *random forest* are two as the number of predictor variables randomly sampled (default parameter) and 10 trees.

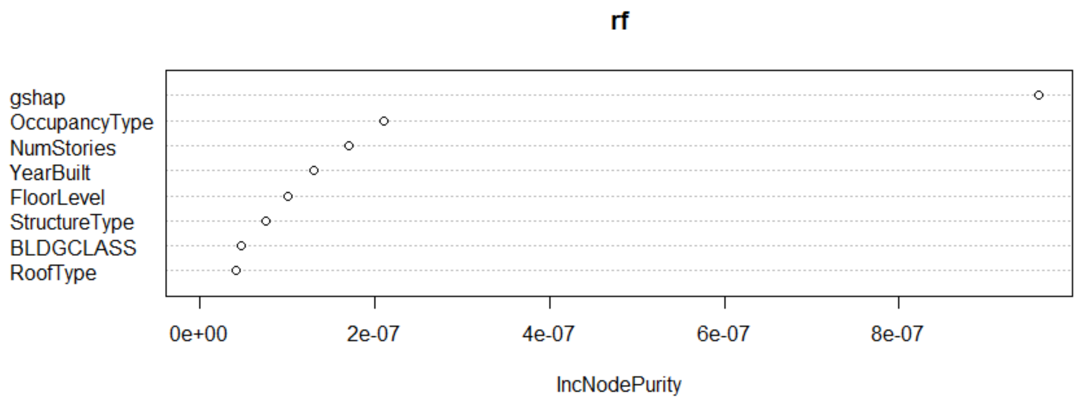


Figure 4.13: Significance of the variables in the random forest

In figure 4.13 we can observe the *significance* of the different variables in the *random forest* model. As we can see in this graph, the most relevant variable in explaining the *rate* is *gshap* which is logical as it is an indicator of seismic activity and we are modeling pure premium for an *earthquakes* portfolio. All the other variables are relevant in estimating the *rate* however much less than the *gshap*. We can see that all the variable have significance in this model which was not the case in the estimation of the loss by *random forest*.

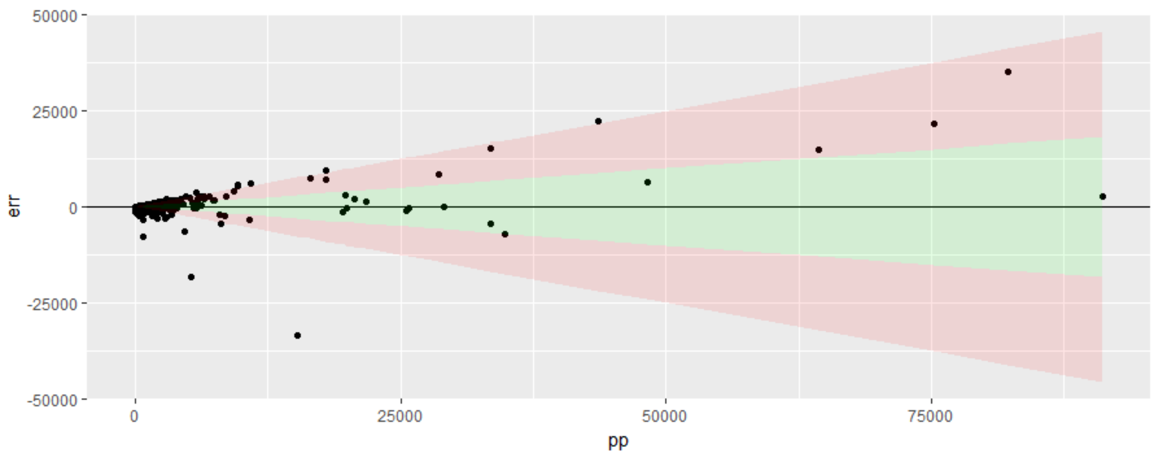


Figure 4.14: Plot of *PurePremium* against the error on the *training sample*

We can see in figure 4.14, the green area represents the values where the error is less or equal to 20% of the *pure premium* and the red area represents the values where the error is above 20% but less than 50% of the *pure premium*. We created these colored areas because we can't compare an error of 10 on a *pure premium* of 100 to an error of 10 on a *pure premium* of $100,000$.

Figure 4.14 represents the *pure premium* against the error on the training sample. We can see in that except for a few values on the right of the graphs, the values are all in the limit of the 50% of the *pure premium*. We can also observe that it predicted well the largest value of the *pure premium*.

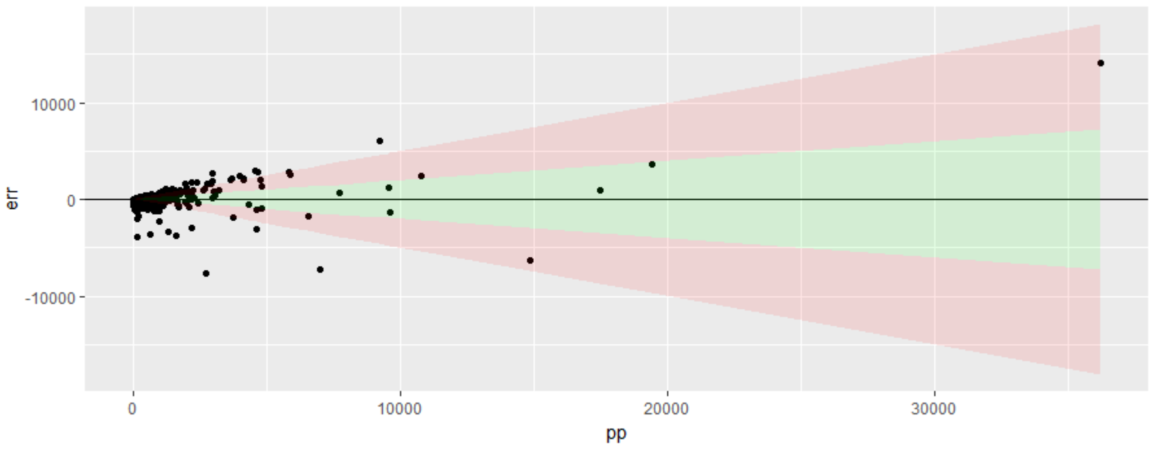


Figure 4.15: Plot of the *PurePremium* against the error on the *testing sample*

We can see in figure 4.14 that the smaller values have errors that are not too large but in contrary to the *training sample* the extreme value, in this sample the error is much more significant, it is here around 40% larger than the predicted value. As a result, the *random forest* gives us the following *root mean square errors* :

$$RMSE_{train} = 3,216\ ;\ RMSE_{test} = 4,256$$

They do not seem that large, however they are around four times larger that the mean of the *pure premium*. This is due to the very large standard deviation of the *pure premium*, indeed as we can see on figures 4.14 and 4.15, the extreme values impact largely the *RMSE*. We can also see that it is a situation of over-fitting as the RMSE of the testing sample is 25% larger than the one of the learning sample, to remedy this, we will see in section 4.2.6.1 an hyper-parametrization of this model.

In figure 4.16 we can see that the *Lorenz curve* for the *random forest* model is pretty close to the perfect model and the shape similar, that means this estimation model gives a pricing very close to what it should be. In the perfect model the premium of the 20% riskier policies represent 88% of the cost, in our model they represent 82% of the cost. However the *RMSE* as we have mentioned before is still quite large.

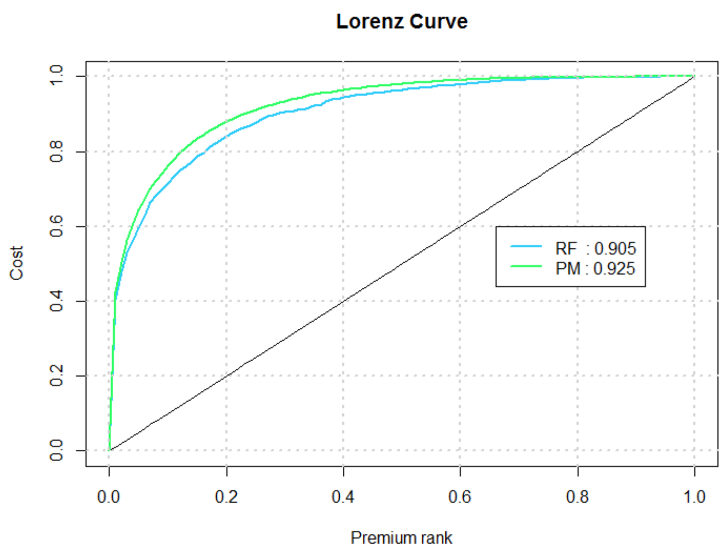


Figure 4.16: *Lorenz curve* for *Random Forest*

4.2.2 Generalized Linear Model

For the *generalized linear model* we had to choose a distribution for the family parameter. We can see in 4.17, the histogram is the distribution of the *rate* variable and the red line represents the fited *gamma distribution*. We can clearly see that the *rate* follows a *gamma distribution*, therefore we decided to apply a *gamma distribution* to our *GLM* to estimate the *rate*. We used the *h2o* package with the the function *h2o.glm* to modelize the rate.

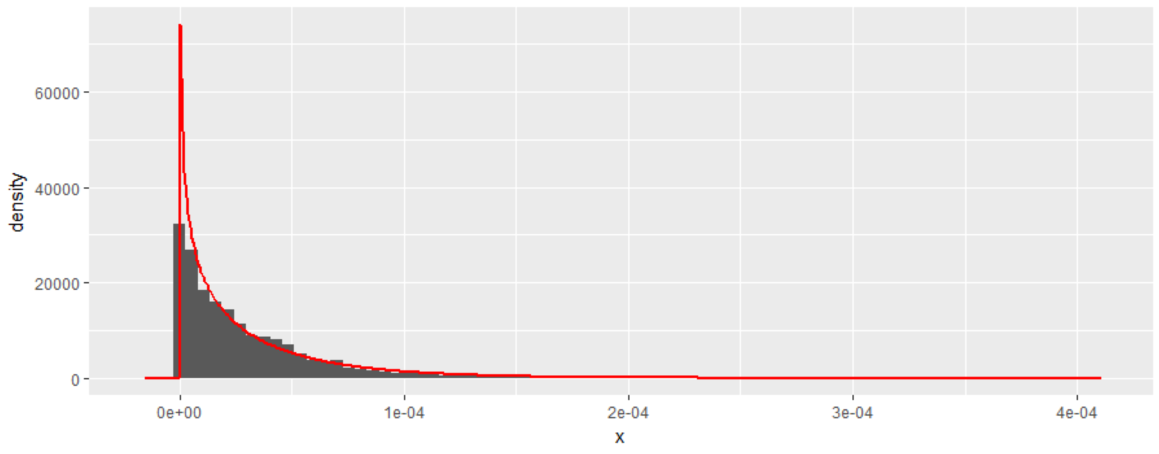


Figure 4.17: Histogram of the rate

The default link function in *R* for the gamma family is $g(x) = {}^1/_x$. In actuarial sciences the pricing function usually used is $g(x) = \log(x)$. We tried both, but finally chose the *R* default function because the results were better. Thus *GLM* present below is done by using the function : $g(x) = {}^1/_x$.

Because of the previous graph, we thought that model would fit well, however as we can see in figure 4.18 the error is above 20% for almost all the point and above 50% for a lot of the points on the *training sample*. This might be due the fact that it is not well fitted for the *extreme values* and the very small values, which composes the majority of our dataset.

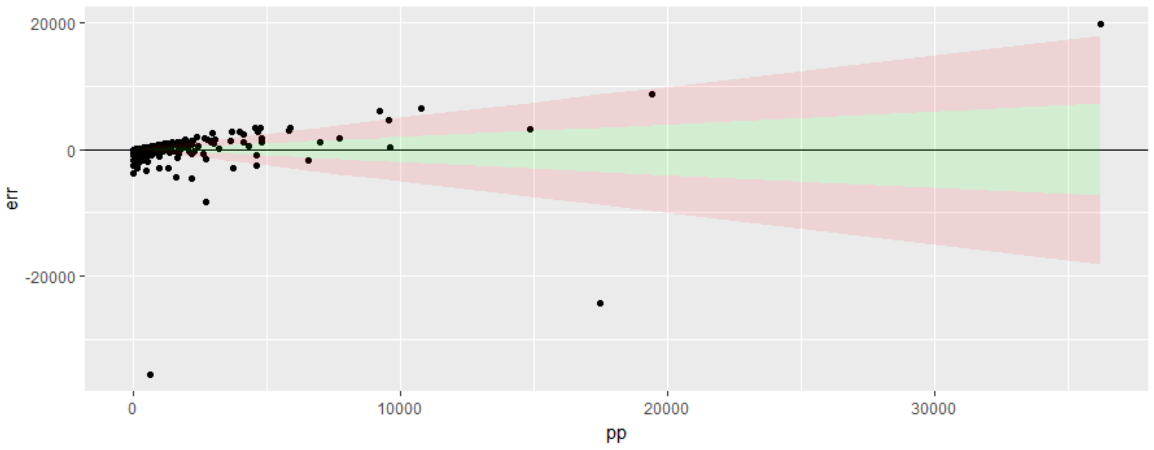


Figure 4.18: Plot of the Pure Premium against the error on the training sample

Since the errors were very large in the *training sample*, it is not surprising that they are also very large in the *testing sample*. As we can see in figure 4.19 most of the values have at least a 20% error and a lot of large values are outside of the red area, so have more than a 50% error in estimation.

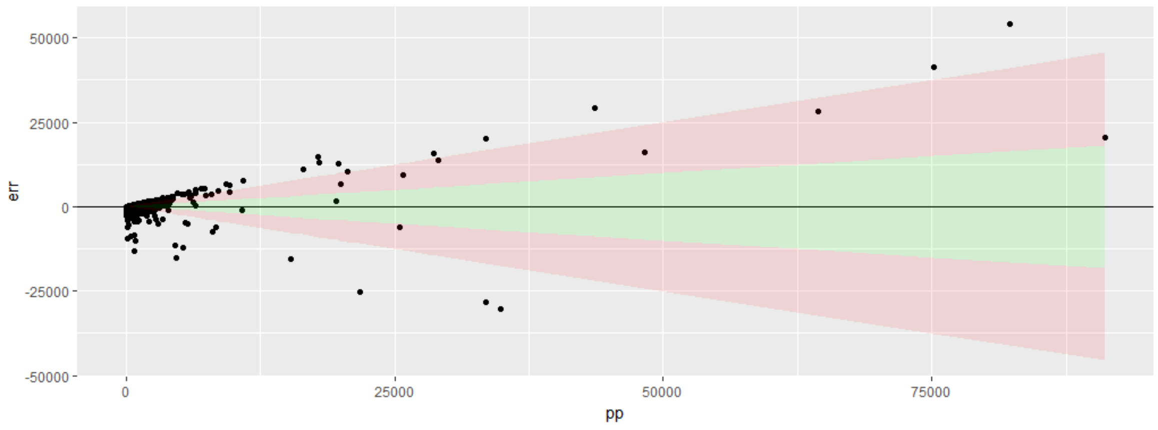


Figure 4.19: Plot of the Pure Premium against the error on the testing sample

The *root mean square errors* we calculated for the *gamma GLM* are:

$$RMSE_{train} = 5,067 \; ; \; RMSE_{test} = 5,404$$

The Lorenz curve in figure 4.28 seems close to the perfect model and the shape looks similar, however it is not as good as the *Lorenz curve* in the *random forest* (figure 4.16) : the area under the curve is of 0.892 for this model while in the *random forest* model it was 0.901. The RMSE for this model are not very good in comparison to the *random forest.*

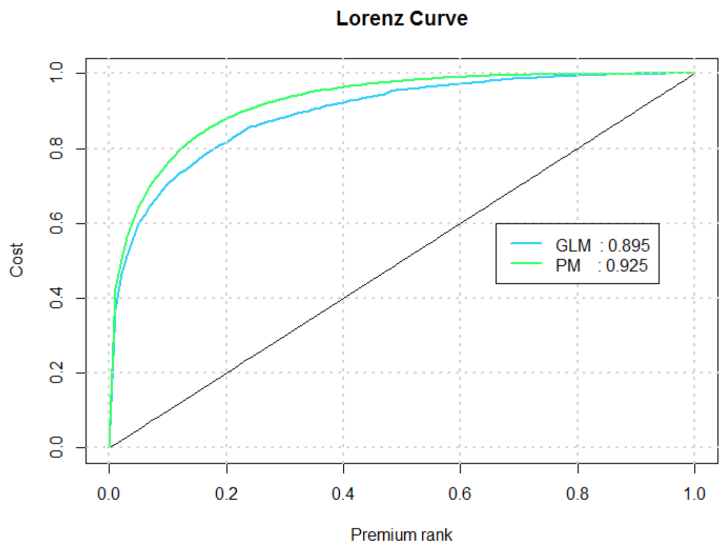


Figure 4.20: *Lorenz curve* for *GLM*

We could have improved this model by doing some variable selection, but the main problem with this model is that it doesn't deal with new levels in predictions, it would therefore be too complicated to improve, and compare to the other models. For exemple, to calculate the *RMSE* above, we first had to fix the testing and training sample in order to get rid of the new levels.

4.2.3 Gradient Boosting Model

We first ran a *Gradient Boosting Model* with 300 *trees* and a *shrinkage coefficient* of 0.6. The *root mean square errors* we calculated for this model are :

$$RMSE_{train} = 4,098 \; ; \; RMSE_{test} = 4,426$$

Figure 4.21 shows the reduction of *squared error* for each variable, this means that it plots the reduction in the *squared error* when we add a new variable to the model. We see that the *gshap* has a significant *relative influence* in this model followed by *OccupancyType* and *NumStories* which still have significance but three times less than the *gshap*. The other variables have much less significance.

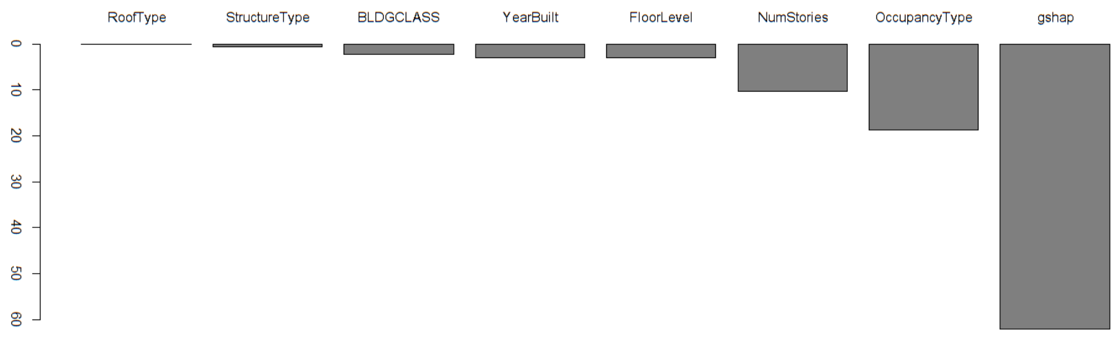


Figure 4.21: Relative influence of variables in *GBM*

Figure 4.22 plots the importance of each modality of a variable in the model. So for example in the *gshap* graph, the values from 4 to 4.5 and the values from 5 to 5.5 are the values of the *gshap* which helps the most the estimation of the rate, and in the *RoofType* variable, the flat roof is more useful in the prediction than the tiled roof which makes sense because the flat roofs usually entitle a higher premium. For the variable *NumStories*, we can see that its usefulness in the prediction increases with the number of stories.

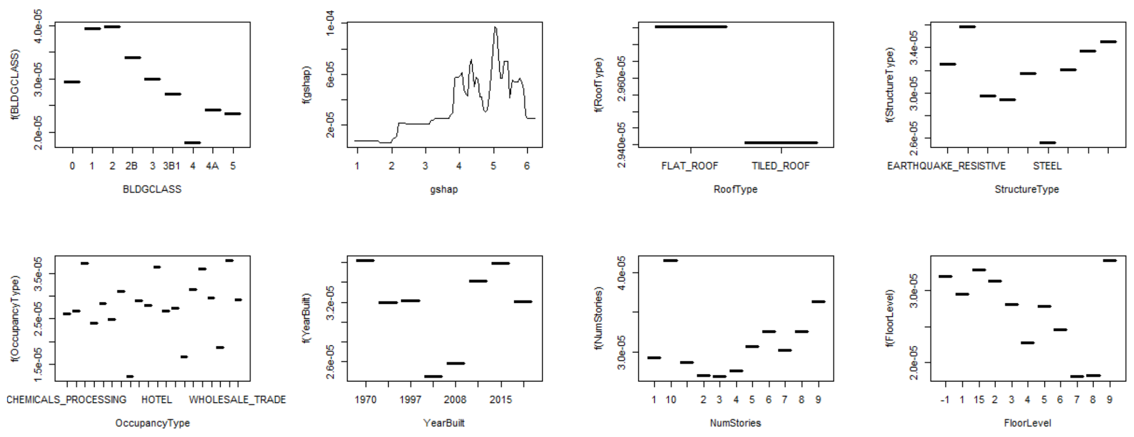


Figure 4.22: Marginal plots of *GBM*

As an output of this first model, we produce the graph in figure 4.23. This graph shows us for which number of trees the error will be minimized. We can see that after a certain threshold the error on the *training sample* (black line) decreases while the error on the *testing sample* (green line) increases. Therefore we choose the*number of iterations* (*i.e* the number of trees) to be 76 (blue dotted line) otherwise it creates over-fitting.

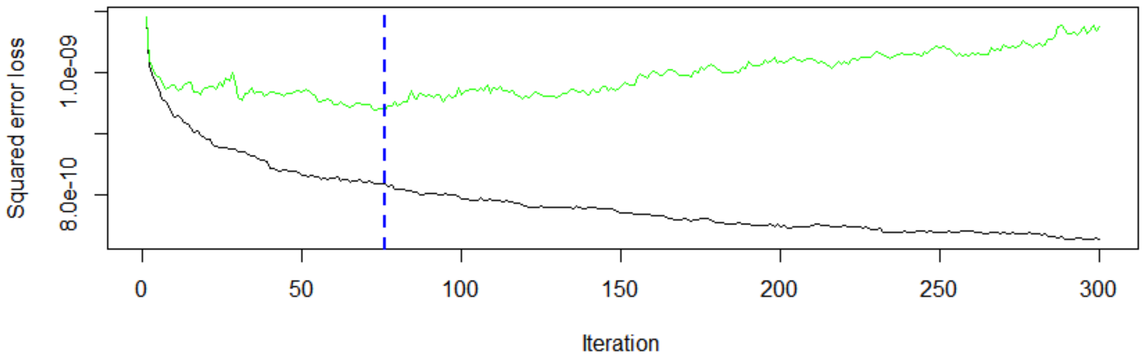


Figure 4.23: Plot of the *GBM* performance against the number of iterations

We ran a new model with the number of trees changed to 76 and the RMSE has worsened for the training sample, but it has improved for the testing sample :

$$RMSE_{train} = 4,321 \ ; \ RMSE_{test} = 4,351$$

Figure 4.24 shows the reduction of *squared error* for each variable in the adjusted model with only 76 trees instead of 300. We see that the *gshap* still has the most significant influence in this model, once again followed by the occupancy type however this times *OccupancyType* has six times less significance than *gshap*. The other variables have much less significance.

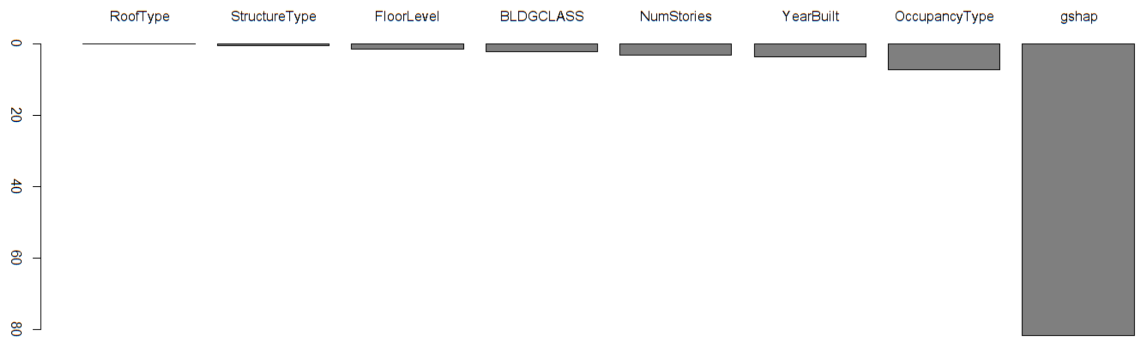


Figure 4.24: Relative influence of variables in *GBM*

In figure 4.25, we see the importance of each modality of a variable in the model. The *gshap* graph shows about the same result as in figure 4.22 so that *gshap* around 4 and 5 are the values of the *gshap* which help the most the estimation of the *rate*. However in this case both modalities of the *RoofType* variable have the same importance in the prediction. For the variable *YearBuilt*, we can see that its usefulness in the prediction is larger when its value is $1,970$ and lower at values around $2,000$.

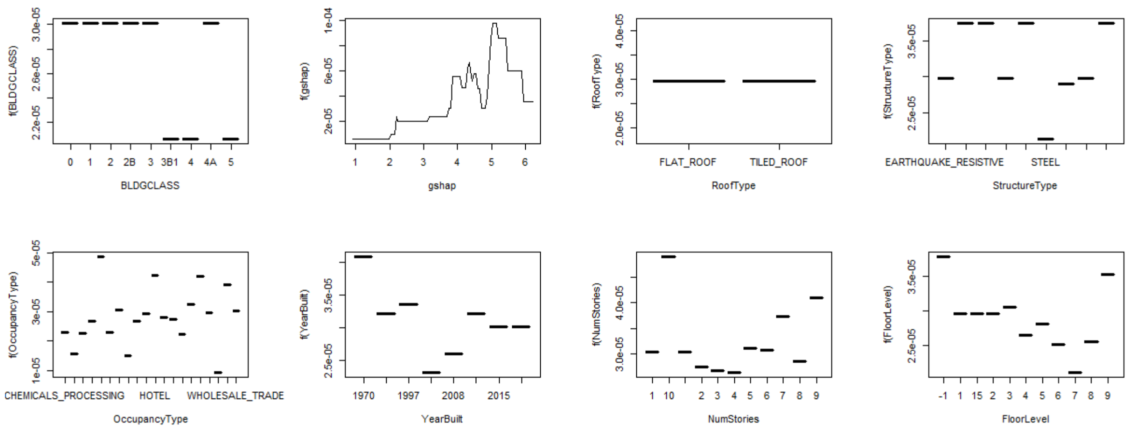


Figure 4.25: Marginal plots of *GBM*

In figure 4.26, the error seem almost always within the 50% area and is very close to zero for the largest value on the *training sample* which is quite good. For the average values (around $2,500$) the values are usually well predicted, it is mostly for the very small values that the error is relatively larger.

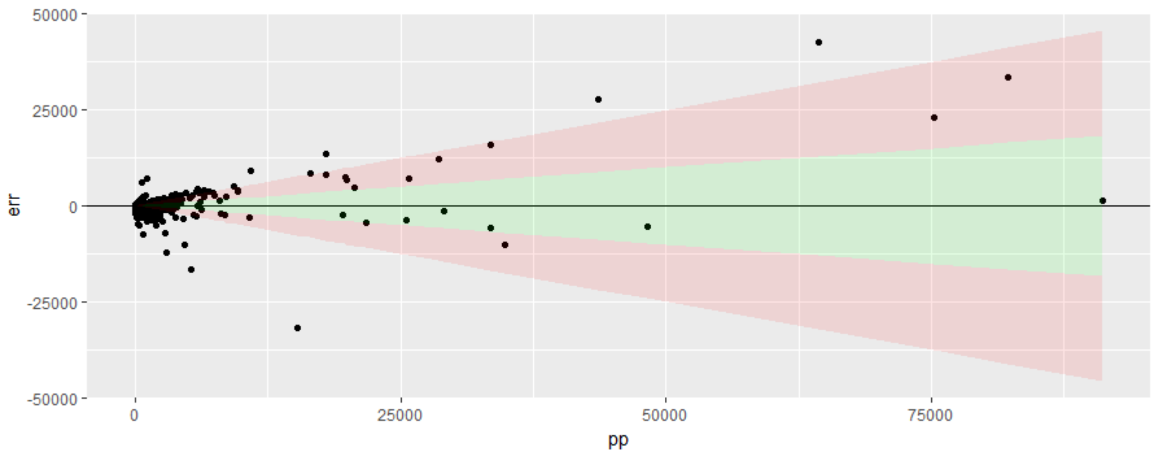


Figure 4.26: Plot of *PurePremium* against the error on the *training sample*

We see in figure 4.27, that the error is larger than in the *training sample*, for example the error for the largest value is around 25% of the *pure premium* while it is lower than 5% in the *training sample*. However the error difference between the two samples seem pretty small.

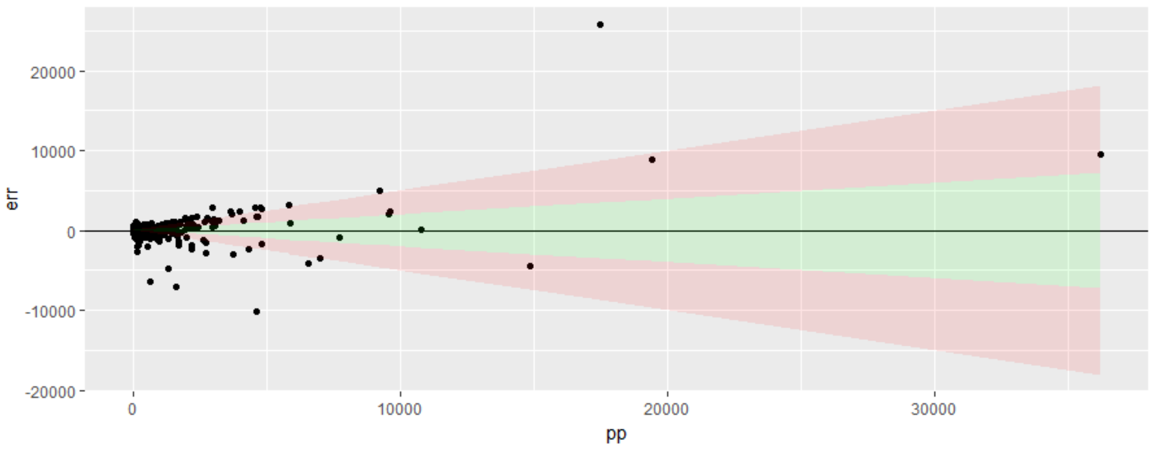


Figure 4.27: Plot of *PurePremium* against the error on the *testing sample*

In the following *Lorenz curve*, the curve labeled *GBM1* represents the model with 300 trees and the model labeled *GBM2* represents the model with 76 *trees* (both with a *shrinkage coefficient* of 0.6). We can see that both of the curves have the right shape, the same as the perfect model. However we see that the second model gives a finer estimation. That can also be seen in the *RMSE*, for the *testing sample*, the second model has a lower error.

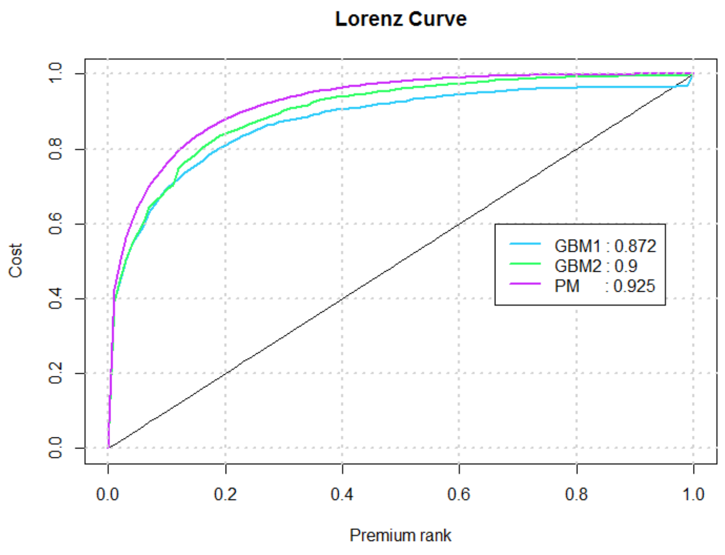


Figure 4.28: *Lorenz curve* for *GBM*

We can see that the model with only 76 *trees* instead of 300 incurs a smaller *RMSE*, we can adjust this model even more by *hyper-parametrization*. We will see in section 4.2.6.2, that the results of the hyper-parametrization are :

$$RMSE_{train} = 3,502 \ ; \ RMSE_{test} = 4,681$$

We have succeeded in decreasing the *RMSE* for the *training sample*, but it has increased largely for the *testing sample*. Therefore we decided to try the *XG-Boost* method as it uses a more regularized model formalization to control *over-fitting*, which gives better performances.

4.2.4 Extreme Gradient Boosting

It was impossible to use XG-Boost to estimate the rate as the values were too small, so we estimated $rate \times 10^6$. We had to set some parameters in order to work the XG-Boost model. We chose the default parameters: the maximum depth is six, the shrinkage coefficient is 0.3 and the minimum number of observation per leaf is one. The only parameter we had to set ourselves was the number of iterations which we set at 200.

Figure 4.29 represents in the first graph the number of leafs that exist at each depth so for example, there is about 500 leafs at the depth six. We can clearly see that the largest number of leafs are at depth seven, at that depth, there are about 2200 leafs.

The first graph is supposed to help us find out which maximum depth is acceptable. So if the higher depth had barely any leaf it would mean that we over-fitted the model, we can see here that there is quite a number of leaf here, so we decided not to change this parameter.

The second graph plots the average number of observation per leaf at each depth, for example at depth five there are an average of two observations per leaf and at depth seven there is an average of 70 observations per leaf.

This graph can be useful to figure out the min child, which is the minimum number of observations that the leafs can contain. If there were very few values in the higher depth, it would once again show that the model is over-fitted, however it is not the case here, so we do not need to change the minimum child parameter.

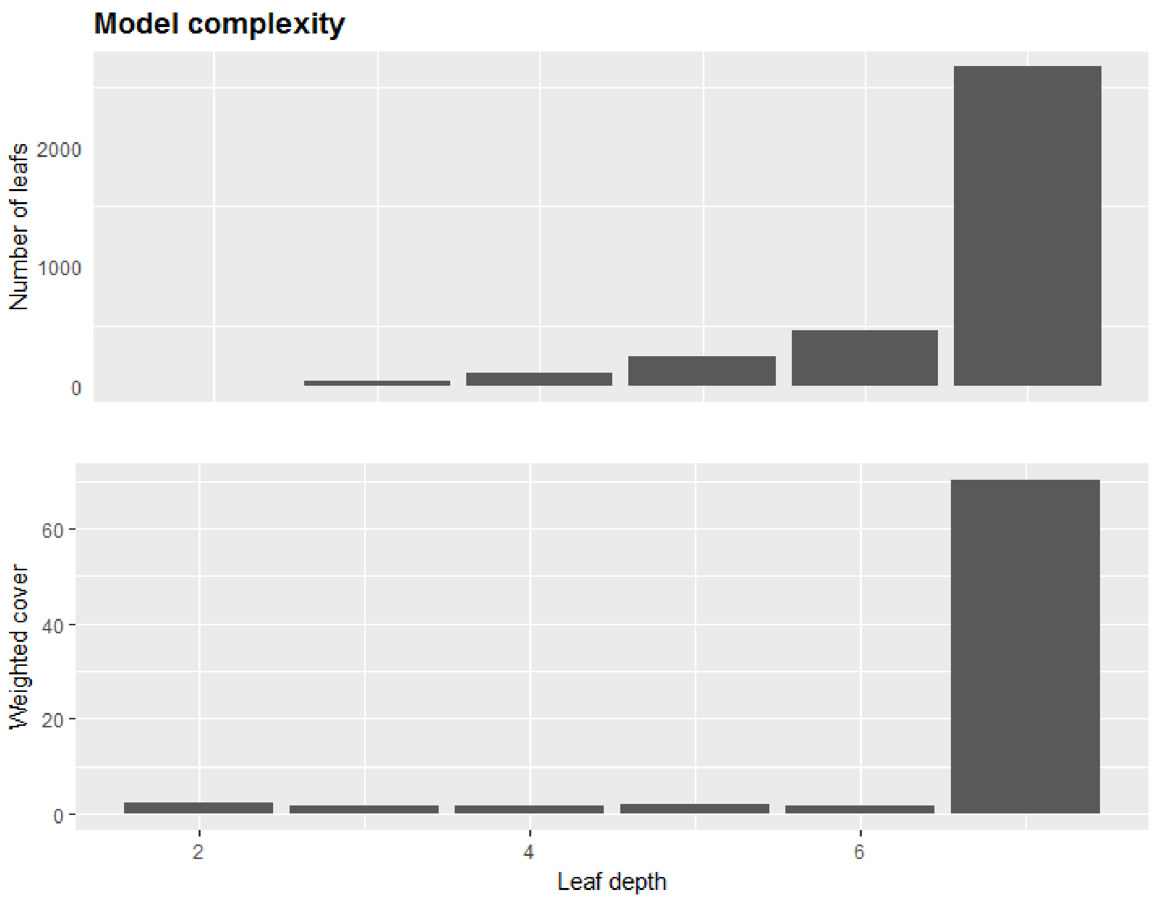


Figure 4.29: Depth of a leaf against the number of leafs and the number of observations

In figure 4.30, we can see the significance of each variable in the model, we see that once again *gshap* is the most significant variable in the predictions. The clusters represent variables that have the same range of significance, so YearBuilt, NumStories and OccupancyType have the same range of significance in estimating the rate.

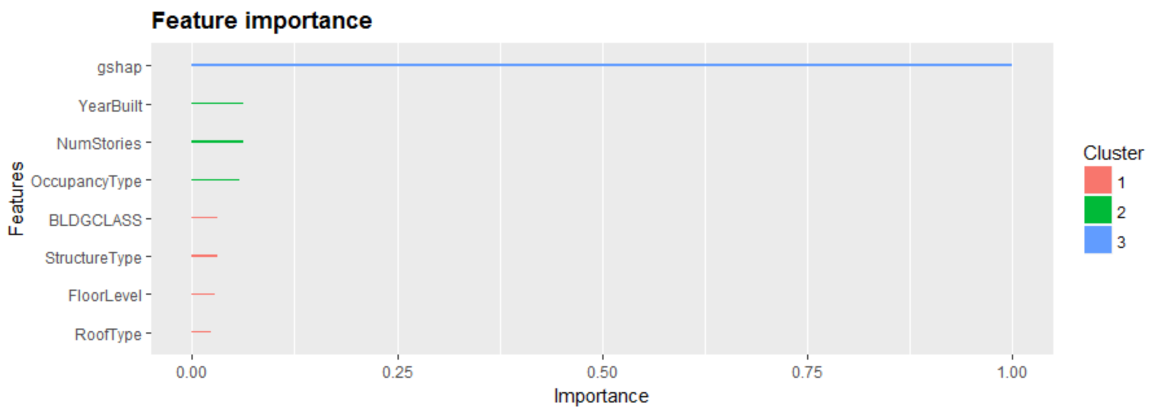


Figure 4.30: Significance of variables in XG-Boost

In figure 4.31, there is the example of one iteration in the XG-Boost model which gives a simple regression tree. The parameters are therefore one iteration with depth four. We can clearly see that gshap is the most used variable to split the data.

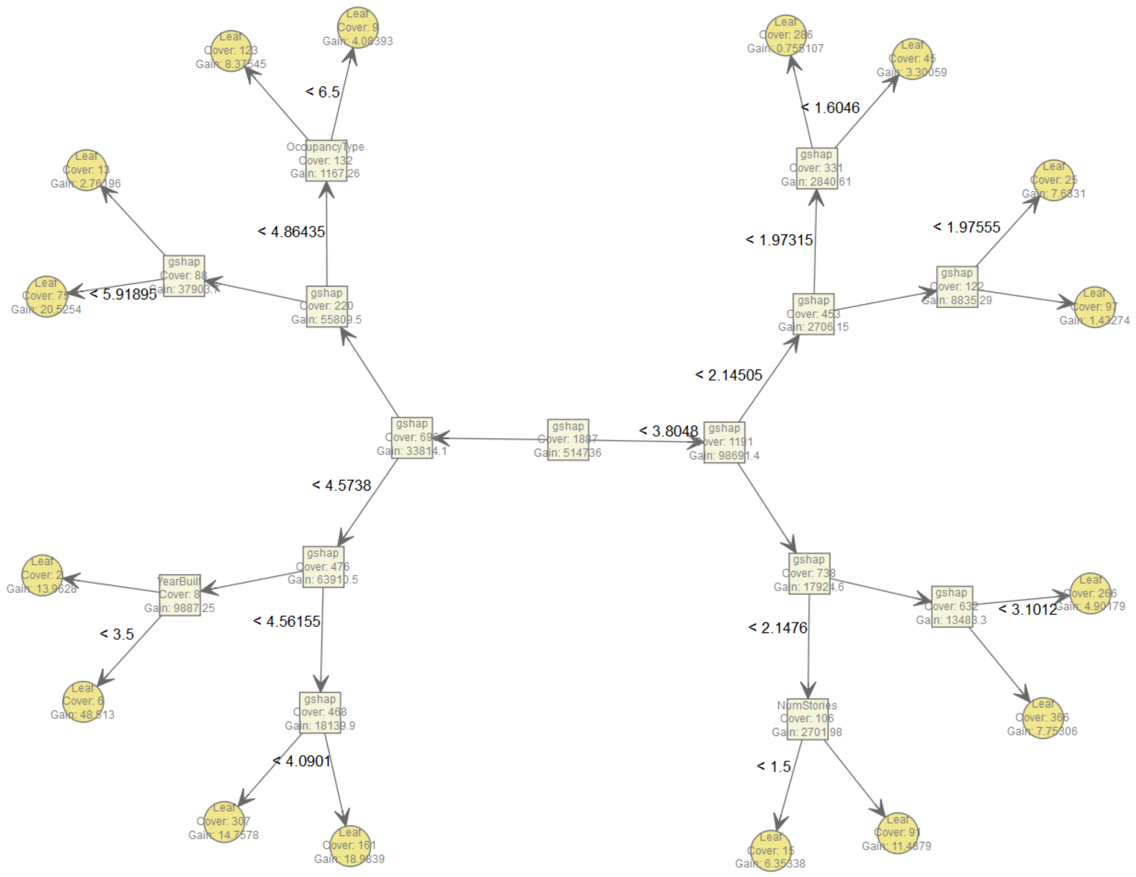


Figure 4.31: plot of one iteration in XG-Boost

In figure 4.32, there is the example of ten iteration in the XG-Boost model with depth four. This graph is just to show that this model is very complex and difficult to interpret. In the final model (section 4.2.6.3) there are 80 iterations with a maximum depth of 7.

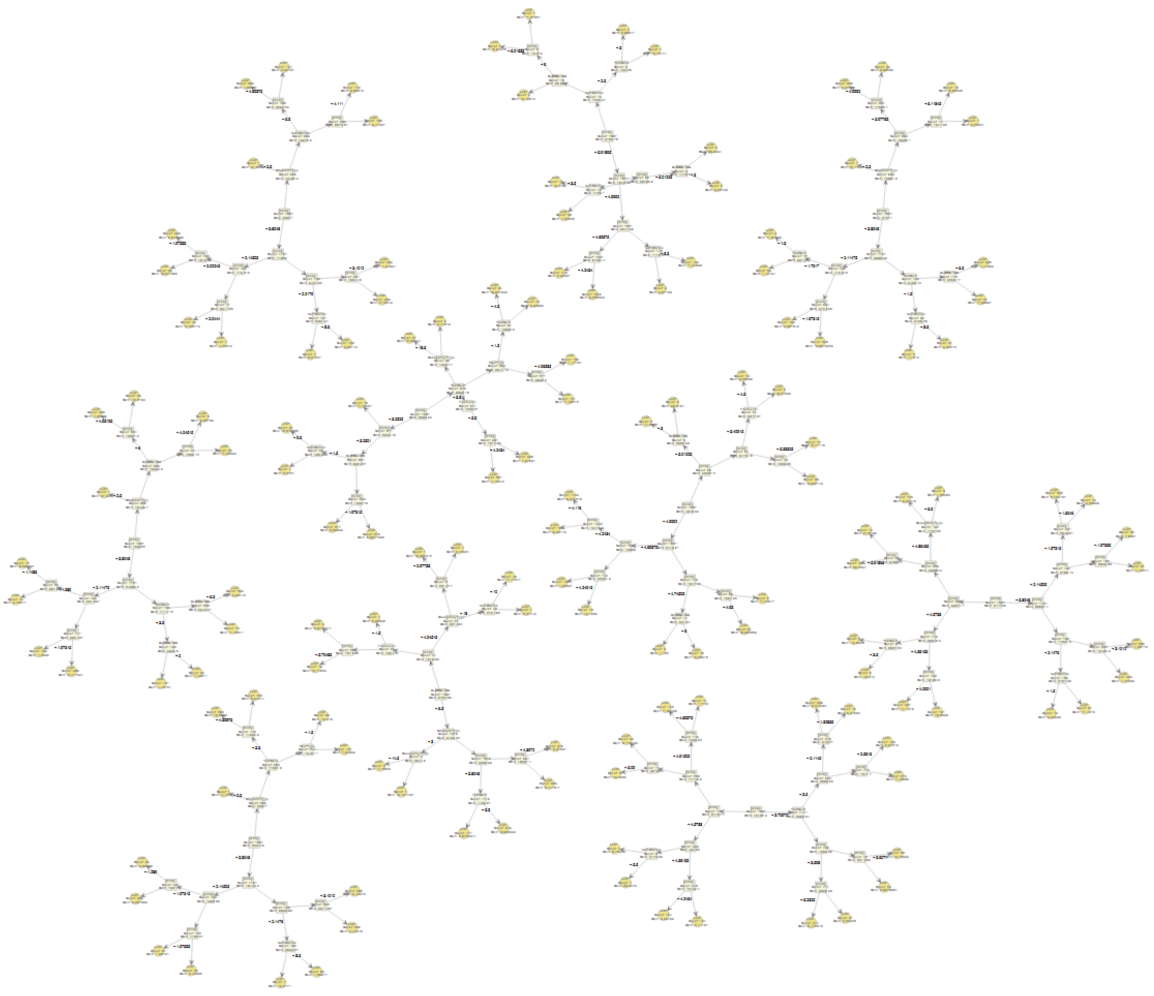


Figure 4.32: plot of 10 iterations in XG-Boost

We can see in figure 4.33 that the errors on the *training sample* are very small, except for a few small values they are all within the 20% error area. We can therefore expect a small *RMSE*.

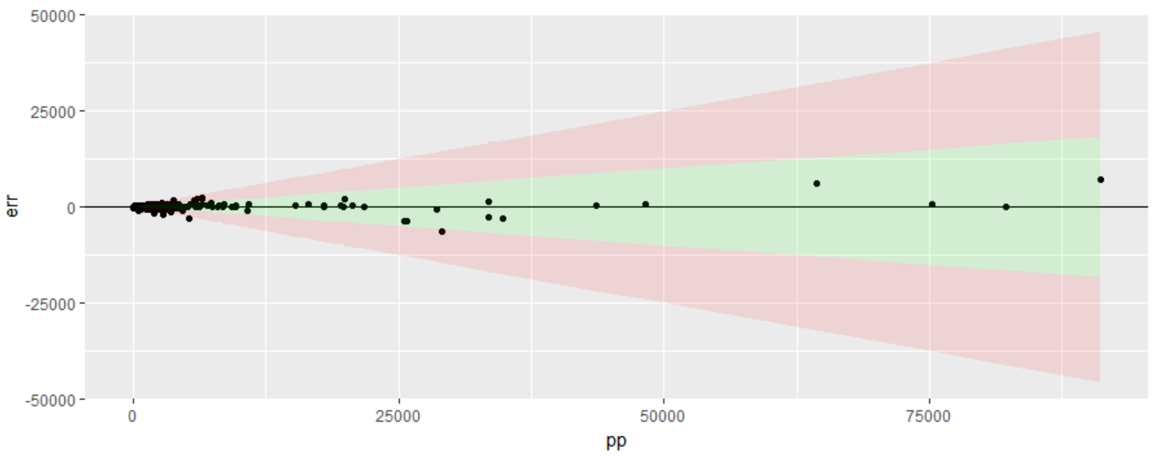


Figure 4.33: Plot of *PurePremium* against the error on the *training sample*

We see in figure 4.34 that the error is much larger than in the *training sample*, in this sample most of the values are out of the 50% boundary in contrary to the *training sample*. We therefore suspect a much larger *RMSE* for the *testing sample*.

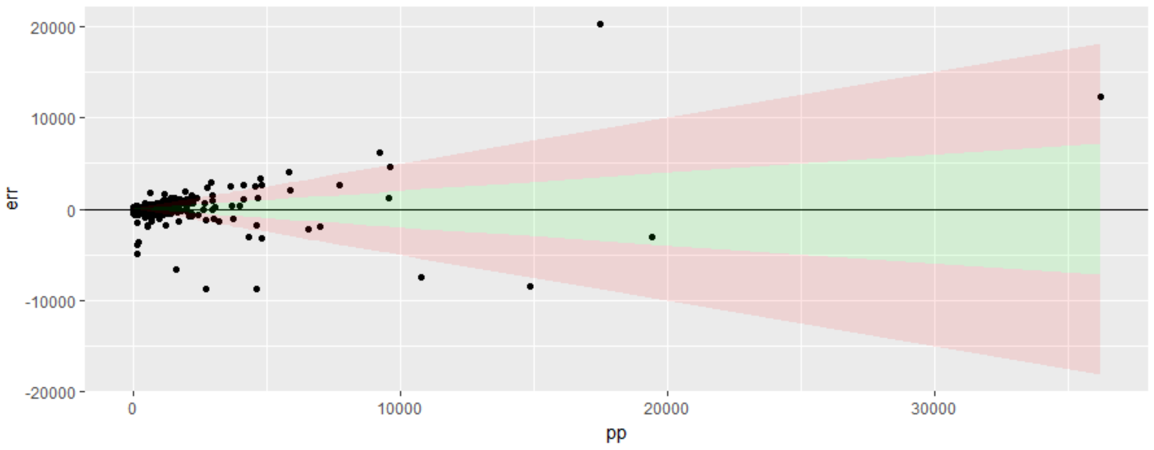


Figure 4.34: Plot of *PurePremium* against the error on the *testing sample*

In the following *Lorenz curve*, we see that once again the *Lorenz curve* for the model is close to the perfect pricing and has the same shape, however it seems that the area under the curve is a bit smaller than in some previous models.

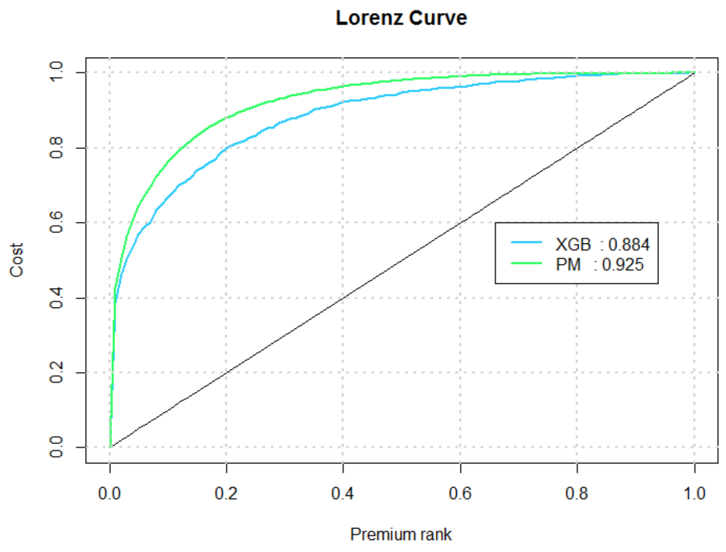


Figure 4.35: *Lorenz curve* for *XG-Boost*

For the *XG-Boost model*, we get the following RMSE:

$$RMSE_{train} = 336\ ;\ RMSE_{test} = 4,046$$

We expected this over fitting, indeed, control over-fitting is the most difficult and crucial step of the XG-Boost model. To improve this, we will try to do a hyper-parametrization on this model in section 4.2.6.3, indeed the shrinkage coefficient is here of 0.3, so by decreasing it we can find a finer estimation.

4.2.5 Support Vector Machine

After having tried *bagging methods* and *boosting methods*, we decided to try another type of method: *Support Vector Machine*. We chose the *radial kernel* which is the same as the *Gaussian kernel*, which we discussed in section 3.1.5.

In figure 4.36, the errors almost always seem outside the 20% area except for five values and there are also quite a bit that are as well out of the 50% area. We can therefore see that the predictions on the *training sample* are quite bad. However the largest value has an error of only 25% which is not that poor as a prediction of an *extreme value*.

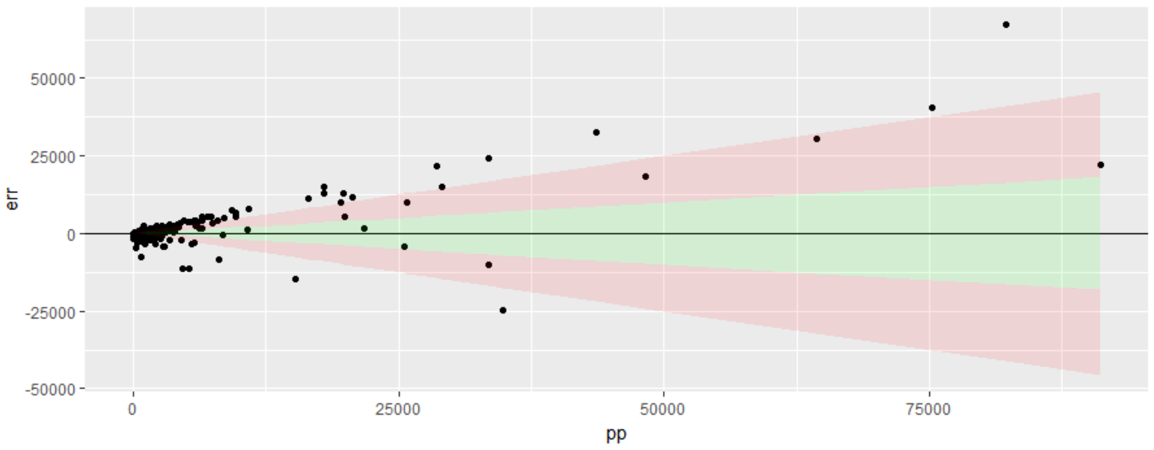


Figure 4.36: Plot of *PurePremium* against the error on the *training sample*

We see in figure 4.37 that the error is larger than in the *training sample*, for example the error for the largest value is at least 60% of the *pure premium*. There are very few values in the green area and a lot of small values outside have errors larger than 50% of the *pure premium*.

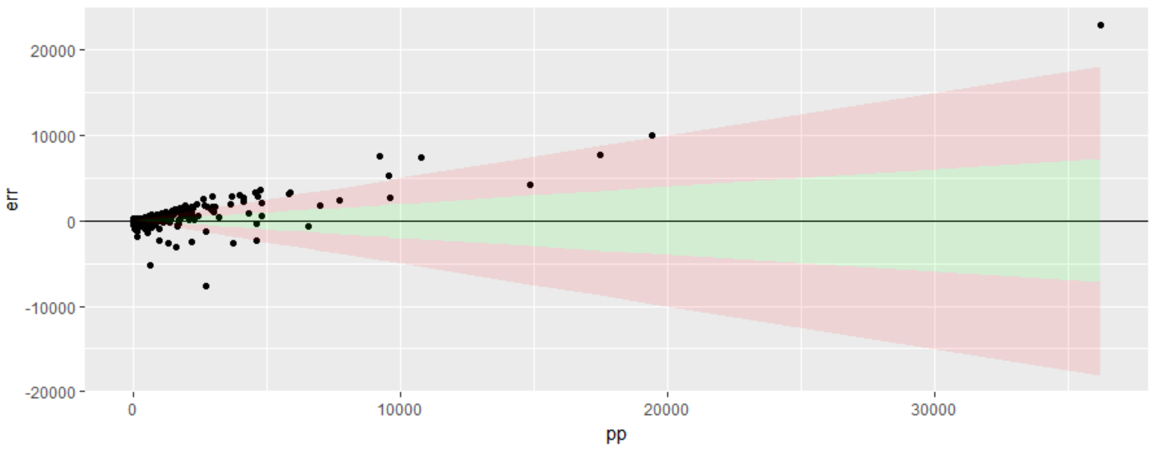


Figure 4.37: Plot of *PurePremium* against the error on the *testing sample*

For the *SVM model*, we get the following RMSE:

$$RMSE_{train} = 4,311 \ ; \ RMSE_{test} = 5,697$$

In the following *Lorenz curve* (figure 4.38), the *SVM model* has the same shape as the perfect pricing and is pretty close to it however in previous models we have seen smaller areas under the curve (such as in the *GBM* model). The *RMSE* are not very good for this model, and it is much larger in the *testing sample* than in the *training sample*. We will try to minimize this error by *hyper-parametrization* for this model in the next section.

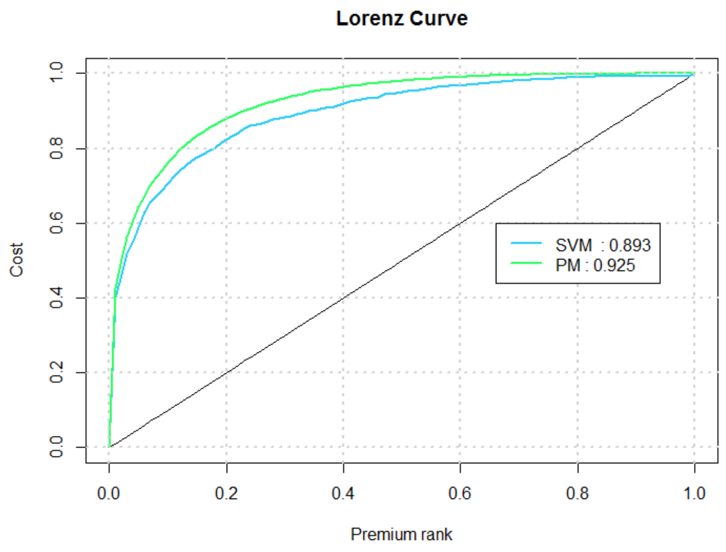


Figure 4.38: *Lorenz curve* for *SVM*

4.2.6 Hyper Parametrization

4.2.6.1 Random Forest

We applied the *random forest* algorithm to estimate the *rate*, however our main goal was to try to find the best forest : the one with the most efficient parameters. The parameters we need to vary and the intervals we make them vary on are :

- The number of predictor variables in the data : $[1\ ;7]$,
- The number of trees in the forest : $[5\ ;600]$,
- The number of predictor variables randomly : $[1;\ \ 7]$.

We have thought about three solutions : minimize the *RMSE* on the learning base, or on the training base, or the sum of both *RMSE*.

To do so, we have made a program that creates a new data base with only n predictors of the 8 by iteration. It adjusts a *random forest*, looks for which predictor has less importance in the forest and deletes it, and continues until we have the number of predictors we want. Once this data table is created, we vary the number of trees, and the number of variable randomly sample, we do this for $n = 1, ..., 8$.

After trying out different parameters of the forest on the *learning database*, we found that the adapted parameters were seven predictor variables in the data, two variable randomly sampled and 255 trees. We can see the following results in figure 4.39. In this graph: "var" is the number of predictor variables in the data, "varforet" is the number of variables randomly sampled for the forest, "abres" is the number of trees in the forest and "errap" is the error for the training sample.

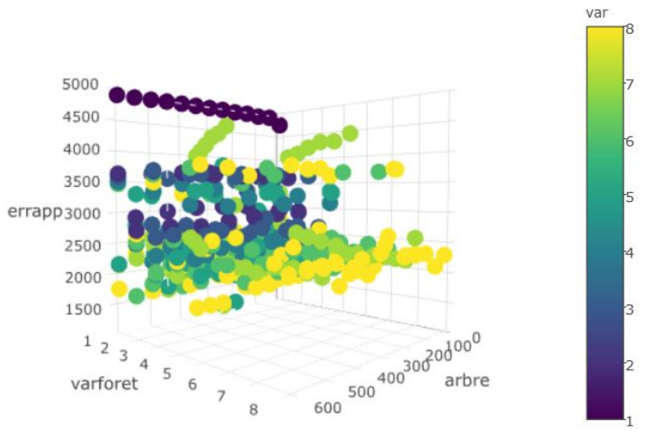


Figure 4.39: $RMSE_{train}$ in function of the parameters

The parameters we found for the best forest for the *testing database* are seven predictor variables in the data base (*RoofType* has been deleted) with two randomly sampled and 145 trees. We can see in figure 4.40 the error with these parameters.

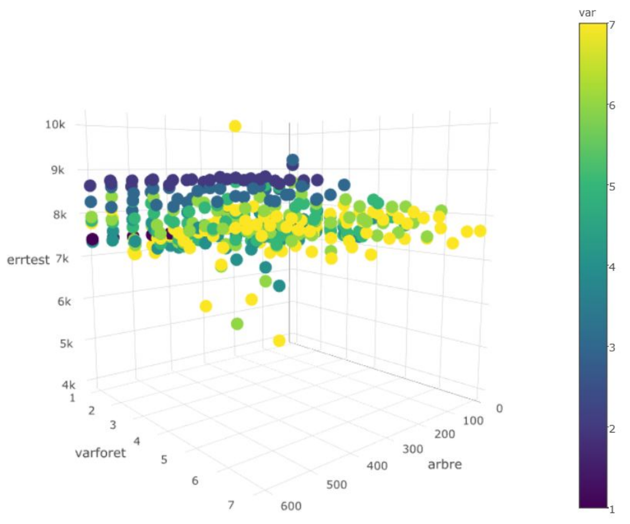


Figure 4.40: $RMSE_{test}$ in function of the parameters

Finally we tried for the sum of the two *RMSE*. We found that it was seven predictor variables in the data, three variable randomly sampled and 125 trees.

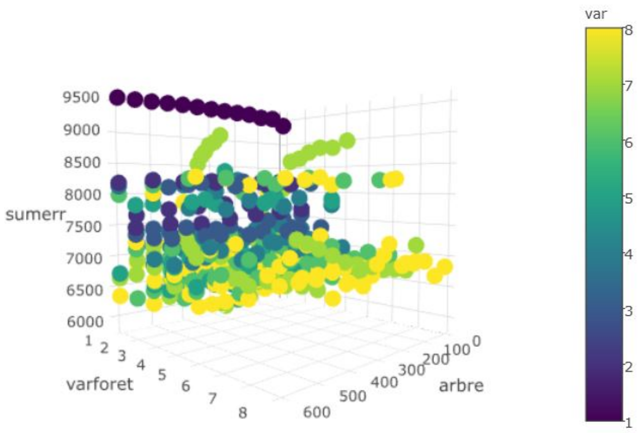


Figure 4.41: $RMSE_{test} + RMSE_{train}$ in function of the parameters

Considering the over-fitting situation, we decided to minimize the *RMSE* on the testing data base. We therefore decided to keep as parameters : seven predictor variable from the data, two randomly sampled and 145 trees.

In figure 4.42, we can see the absolute value of the error in the *random forest* model on the *training sample*, the red points represent the *optimized model*, the one with hyper-parametrization, and the line is the fitted linear model. The blue points represents the default model, the one presented in the previous section and the line is the fitted linear model. So on the *training sample* the default model gave better predictions.

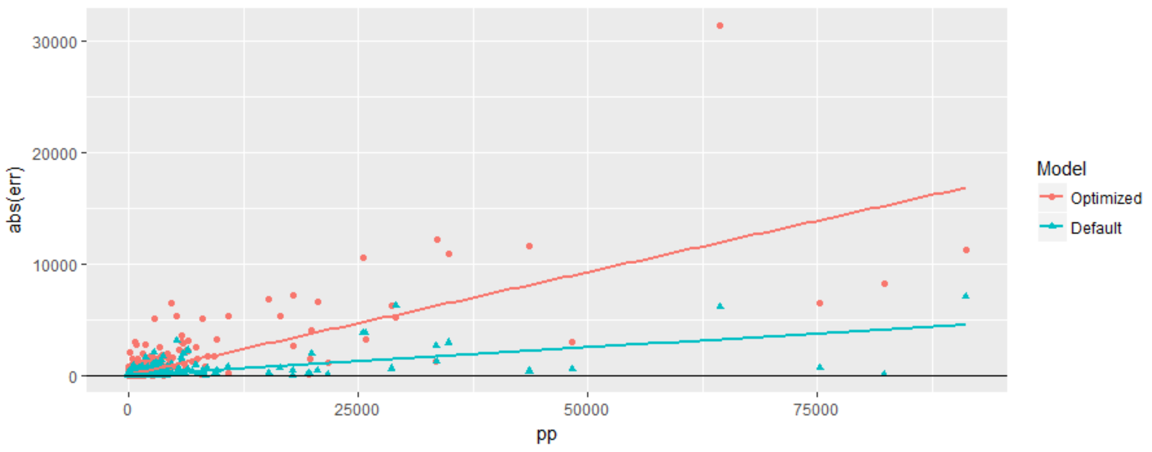


Figure 4.42: Default vs optimized forest on the *training sample*

In figure 4.43, we can see we can see the absolute value of the error is much smaller for the optimized model than for the default model, we can therefore admit that the optimization of the *random forest* gave use better predictions.

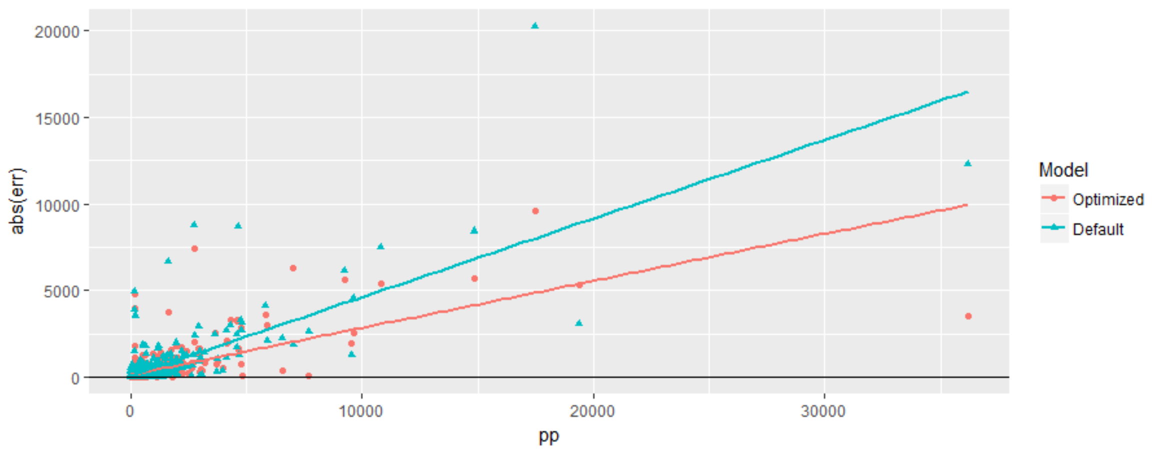


Figure 4.43: Default vs optimized forest on the *testing sample*

Here are the RMSE on the training and testing sample for the default *random forest* and the optimized *random forest*:

$$RMSE_{train}^{def} = 3,216 \; ; \; RMSE_{test}^{def} = 4,256$$

$$RMSE_{train}^{opti} = 3,520 \; ; \; RMSE_{test}^{opti} = 4,212$$

In figure 4.44 we can see the Lorenz curves for the default and optimized *random forest* model. The curve for the optimized model is a little below the curve for the default model. We have seen before that the estimations are better on the testing database for the optimized *random forest*, so even if the estimations are a little better in the optimized model, the pricing corresponds maybe a little less to the risk profile in the optimized *random forest*.

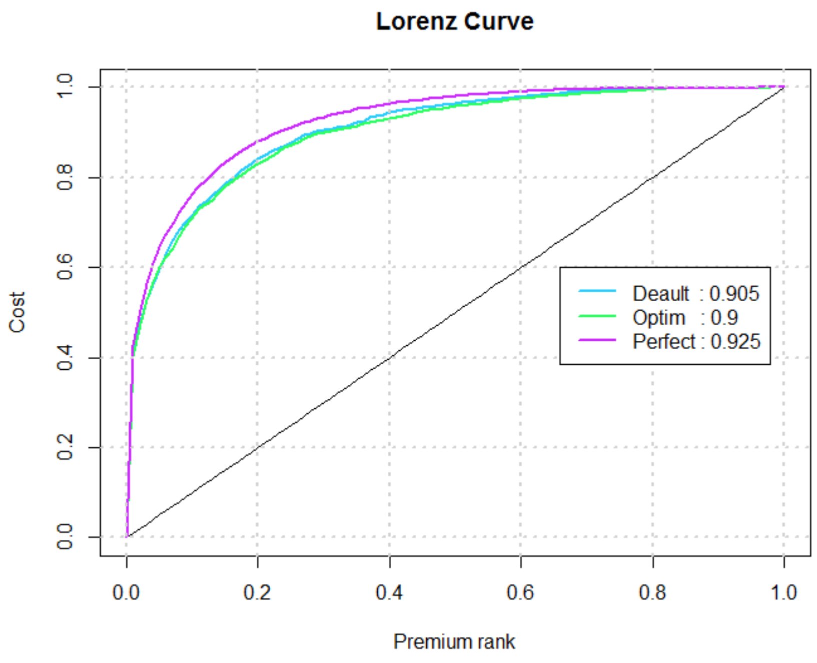


Figure 4.44: *Lorenz curve* for the *random forest* model

We have seen that the *RMSE* on the testing sample is smaller for this optimized *random forest* but the difference is very small (only reduced by 44), also the *Lorenz curve* is not as good and the *RMSE* on the *training sample* has increased thus we can conclude that this optimization was not very useful. We therefore reject this model and keep the previous default model.

4.2.6.2 Gradient Boosting Model

To optimize the *Gradient Boosting Model* we decide to try a new method, which is not fundamentally different from the previous method. We used the *caret* package . The principle is the same, choose intervals for some of the parameters and make them vary.

The difference is that we don't have to make loops, indeed the caret function takes as entry a metric error (*RMSE* in our case), the grid of the parameters intervals and the model. Then it calculates each one of the metric errors and returns the parameters that minimizes that error. The parameters we need to vary and the intervals we make them vary on are :

- The maximum depth of variable interactions $[1\,;500]$,
- The shrinkage coefficient $\{0.2\,;0.1\,;0.01\,;0.001\}$,
- The number of trees $[0\,;2,500]$.

The parameters that we found that optimized this model are 750 trees, 3 for the maximum depth of variable interactions, and a shrinkage coefficient of 0.1.

In figure 4.45, we can see the error in the *GBM model* on the *training sample* for the optimized model is above the default model, therefore for this sample the predictions are better for the default model. This is due to a correction of the *over-fitting* on the training sample.

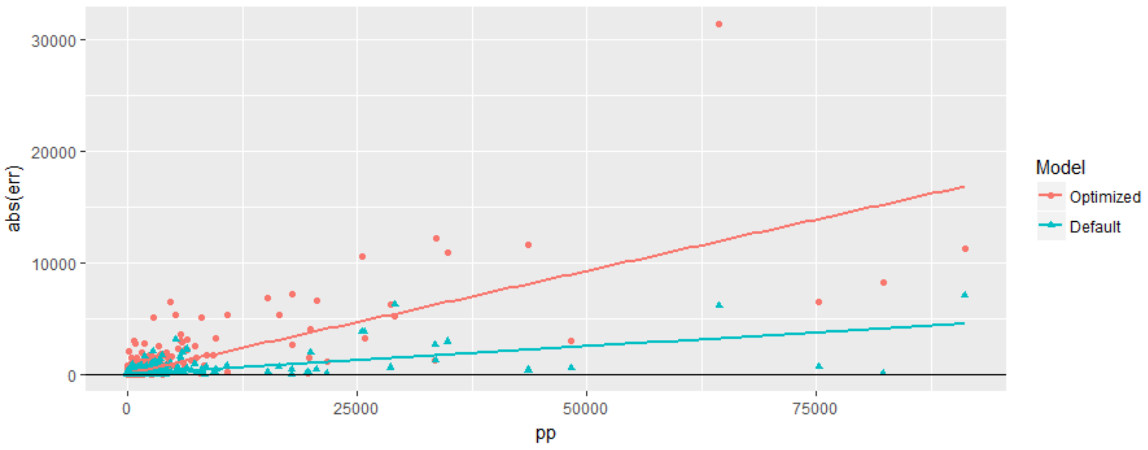


Figure 4.45: Default vs optimized *GBM* on the *training sample*

In figure 4.46, we can see the error in the *GBM model* on the *training sample* for the optimized is lower than for the default model, therefore the predictions are better in the optimized model. This is due to the *hyper-parametrization* done for the optimized model.

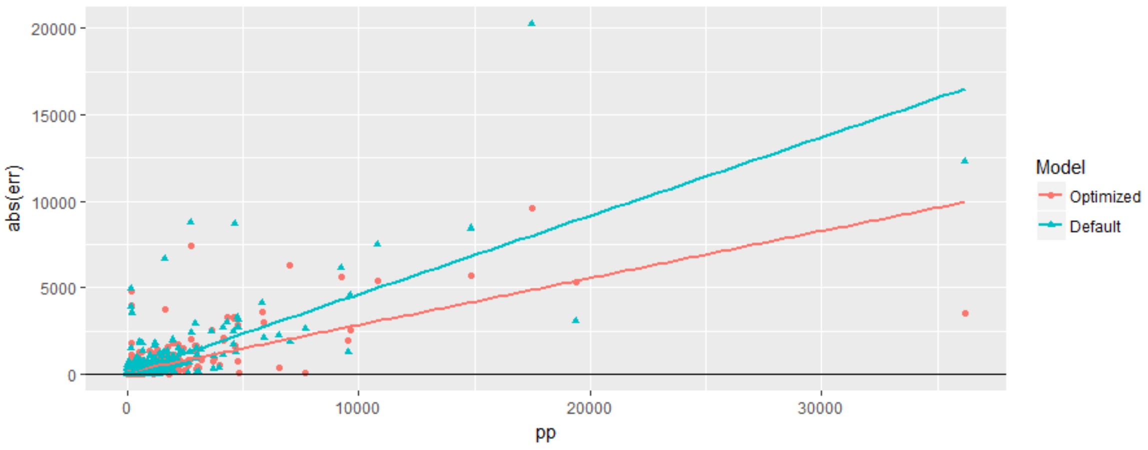


Figure 4.46: Default vs optimized *GBM* on the *testing sample*

The *RMSE* we obtain for the *GBM models* are:

$$RMSE_{train}^{def} = 3,502 \; ; \; RMSE_{test}^{def} = 4,681$$

$$RMSE_{train}^{opti} = 4,034. \; ; \; RMSE_{test}^{opti} = 4,350$$

In figure 4.47 we can see the *Lorenz curves* for the default and optimized *GBM model.* The curve for the optimized model is a little between the default1 and default2 curves. So the default two model, the one with 76 trees, has a better *pricing* of the risk profile than the optimized model, however the difference is small. The *RMSE* has decreased from the default to the optimized model for the *testing database* so the predictions are better and it has increased in the *training sample* to correct the *over-fitting*.

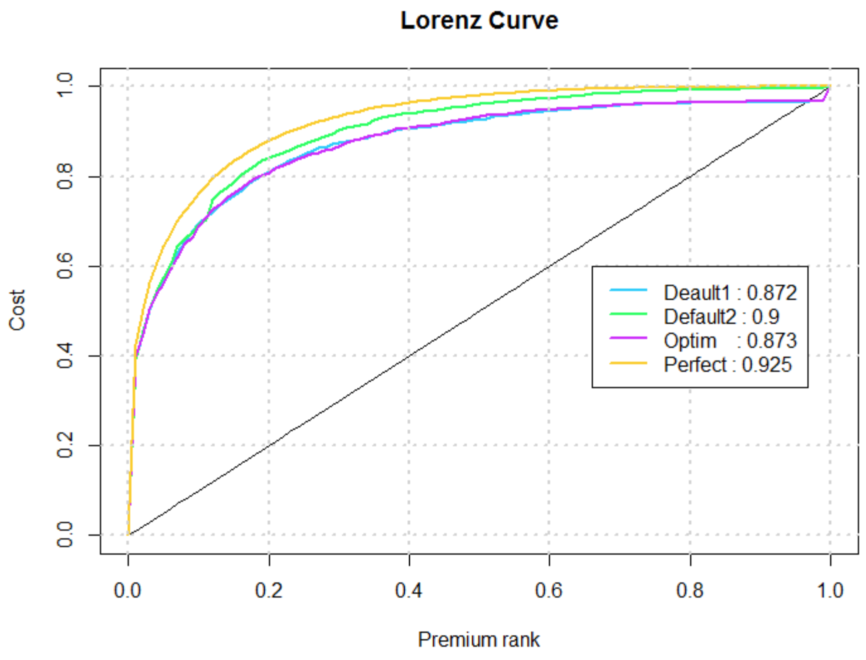


Figure 4.47: *Lorenz curve* for the *GBM* model

Since there is few differences in the two models, but we still have decreased the *RMSE* by more than 300 on the *testing database* and have corrected the over-fitting on the *training database* will keep the optimized model.

4.2.6.3 Extreme Gradient Boosting

XG-Boost is a very complexe model, that need to be correctly tuned to avoid *over-fitting*. This model has plenty of parameters, but we decided to focus on three of them, which have a bigger impact on the model. The parameters we need to vary and the intervals we make them vary on are :

- The maximum number of iterations $[1\ ; 500]$,
- The shrinkage coefficient $[0.1\ ; 0.9]$,
- The maximal depth of the trees $[1\ ; 60]$.

For this model, we did not use the *caret* package, because *XG-Boost* was already too complex, we therefore decided to do the optimization “manually” using loops. Once again, considering the *over-fitting* situation we chose to minimize $RMSE_{test}$.

The parameters that we found that optimized this model are 80 iterations with a maximum depth of seven and a shrinkage coefficient of 0.1.

In figure 4.48, we can see the absolute value of the error in the *XG-Boost* model on the *training sample*, we can see that the default model incurred smaller errors than the optimized model. So there is a decrease in the efficiency of the predictions in the *training sample*, this can be explained as the decrease of the *over-fitting*.

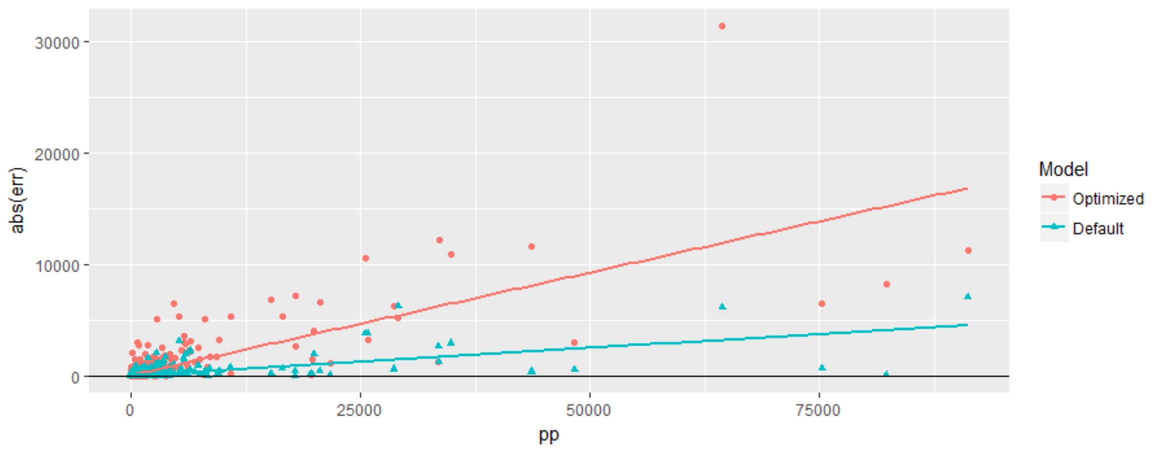


Figure 4.48: Default vs optimized *XG-Boost* on the *training sample*

In figure 4.49, we can see we can see the absolute value of the error is much smaller for the optimized model than for the default model, we can therefore see that the optimization of the *XG-Boost* gave us better predictions.

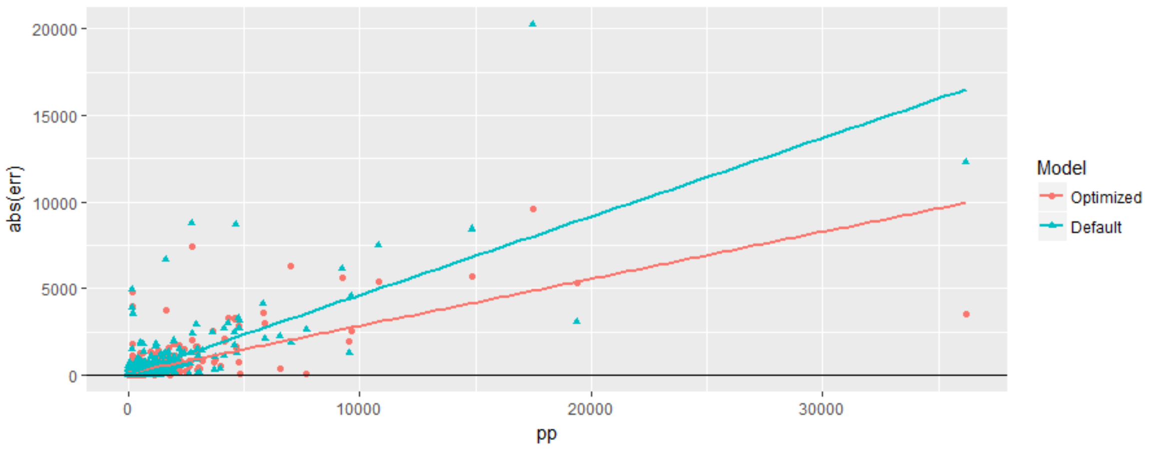


Figure 4.49: Default vs optimized *XG-Boost* on the *testing sample*

The *RMSE* for the *XG-Boost* model are the following:

$$RMSE^{def}_{train} = 336 \quad ; \; RMSE^{def}_{test} = 4,046$$

$$RMSE^{opti}_{train} = 2,304 \; ; \; RMSE^{opti}_{test} = 3,672$$

In figure 4.50 we can see the *Lorenz curves* for the default and optimized *XG-Boost* model. The curve for the optimized model is exactly at the same place than the the default *Lorenz curves.*

In terms of *pricing* the default model corresponds to the same price profile as the optimized model, so to determine which model is better, we can only compare the *RMSE*. We see that the *RMSE* on the training sample has increased but this is explained by a decrease in over fitting and the *RMSE* on the testing sample has decreased, which shows that the predictions are better.

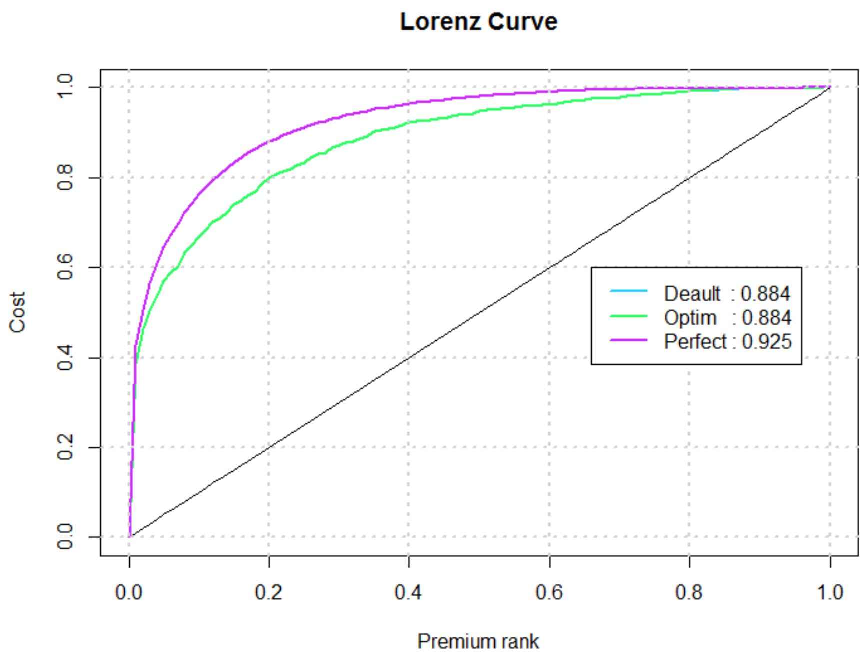


Figure 4.50: *Lorenz curve* for the *XG-Boost* model

We can therefore conclude that the optimized *XG-Boost* model gives better prediction than the default one, it is thus the model we will keep to compare to the others in section 4.3. Nonetheless there is still over-fitting, but as we said in the begin of this section, *XG-Boost* is very complex with lots of parameters. We could have done a second optimization using these parameters (for example the minimum loss reduction required to make a further partition on a leaf node of the tree, or the minimum sum of instance weight needed in a child).

4.2.7 Support Vector Machine

For the *hyper-parametrization* of the *SVM model* we decided, as seen in the section 4.2.6.2, to use the *caret* package. The parameters we need to vary and the intervals we make them vary on are :

- The cost of constraints violation $[5 \,; 500]$,
- The epsilon in the loss function $[0 \,; 1]$.

The parameters that we found are : 0.5 for epsilon and 16 for the cost. In figure 4.51, we can see the absolute value of the error in the *SVM model* on the *training sample*, the optimized model has a smaller error than the default model, there is a increase in the efficiency of the predictions in the *training sample*.

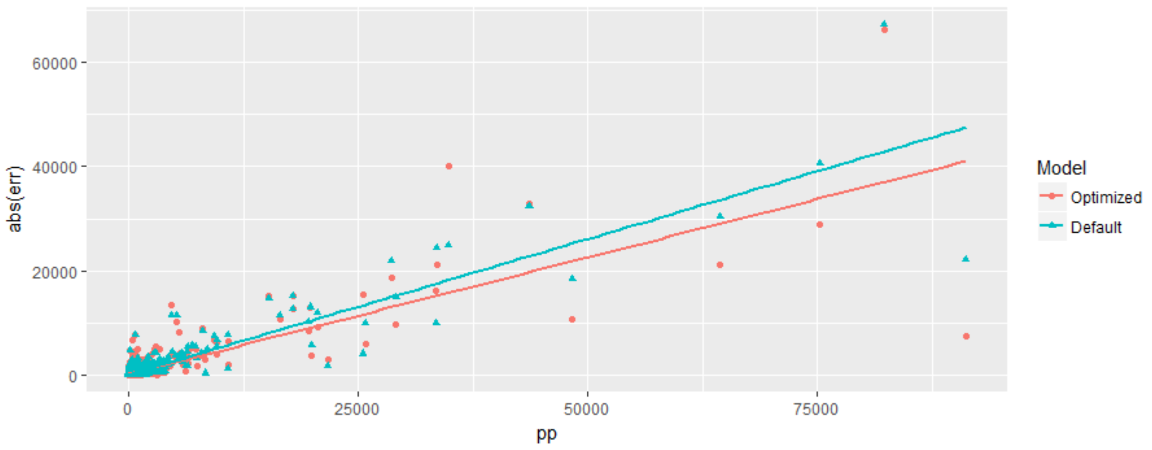


Figure 4.51: Default vs optimized *SVM* on the *training sample*

In figure 4.52, we can see we can see that the absolute value of the error is about the same for the optimized model than for the default model. It is hared to tell which has the smaller error, it seems the optimized model error might be a little smaller on the *testing sample*

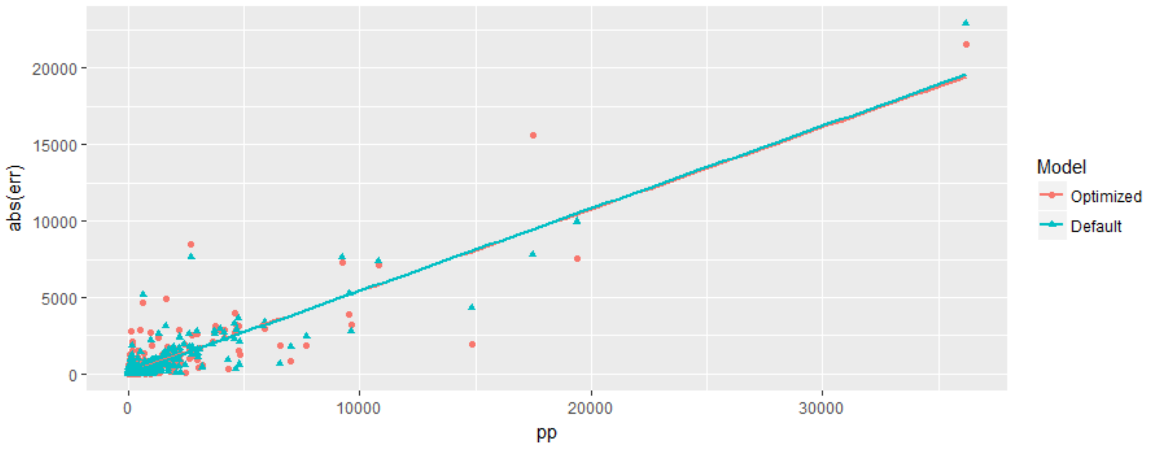


Figure 4.52: Default vs optimized *SVM* on the *testing sample*

The *RMSE* we obtain for the *SVM model* are the following:

$$RMSE_{train}^{def} = 4,311 \; ; \; RMSE_{test}^{def} = 5,697$$

$$RMSE_{train}^{opti} = 5,130 \; ; \; RMSE_{test}^{opti} = 4,305$$

In figure 4.53 we can see the *Lorenz curves* for the default and optimized *SVM* model. The curve for the optimized model is a little bit higher than the default one, so the predictions correspond to the right risk profile.

We see that the *RMSE* for the testing database is smaller on the optimized model than on the default model, however it is smaller on the *testing sample* than on the *training sample* for the optimized model. This is unusual, this might be explain by the fact that to find the optimized model, *R* tests a lot of possibilities and we asked that it only minimizes the testing *RMSE* but we didn't state anything for the training *RMSE*.

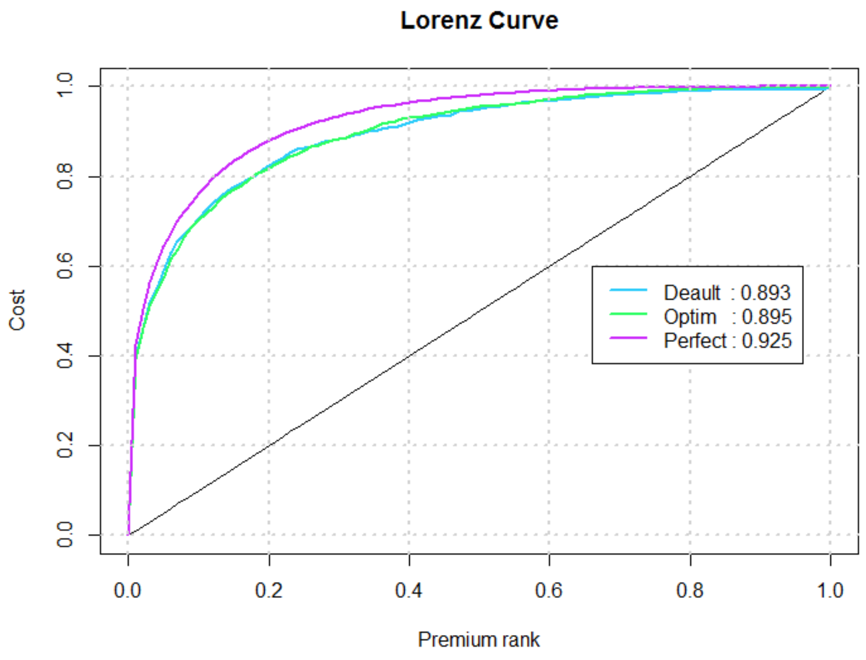


Figure 4.53: *Lorenz curve* for the *SVM* model

As we have mentioned before, the *RMSE* is smaller for the default model on the *training sample* but it is smaller for the optimized model on the *testing sample*. We therefore decided to keep the model which minimizes the sum of the *RMSE*. Thus, we decided to keep the optimized model.

4.3 Results summary

In table 4.1, we can see the result that each model gave, in order to compare them properly. In this table we can find for all the models that predict the *rate* : the *root mean square error* for the *training sample* and the *testing sample*, if the model can handle missing data, if there is a possibility of*over-fitting*, and the complexity of the model.

Model	$RMSE_{train}$	$RMSE_{test}$	NAs	Over-fitting	Complexity
XG-Boost	$2,304$	$3,672$	+	- -	- -
Random forest	$3,216$	$4,256$	+	+	-
GBM	$4,034$	$4,350$	-	-	-
GLM	$5,067$	$5,404$	-	+	+ +
SVM	$5,130$	$4,305$	-	+	-

Table 4.1: Summary of the modelization results

NB : in the table above, we put "-" for GBM in NAs section because in theory it can deal with missing values, but the model implemented in R doesn't. In fact, it is one of the main objective of XG-Boost, which deals with functions who use the R objects of class "sparse matrix" so the model can handles missing values.

In the *NAs* column, the "-" means that model is not built to deals with *NAs* and the "+" means it can run with *NAs*. In the *over-fitting* column, the "- -" means that the model often over-fits, the "-" sign means that it has a tendency to over fit and the "+" means it seldom over-fits. In the complexity column, "- -" means that the model is complex in building and "++" means that it is easy to build.

We can see the different *Lorenz curves* of all the models that we accepted in figure 4.54 on the next page. We see that the *random forest* model gives us the best *Lorenz curve*, however all the models have the right shape and they are all very close together.

In table 4.1 we see all the *root mean squared errors* of the accepted models and we clearly see that the *XG-Boost* model has a lower *RMSE* on the *training sample* as well as the *testing sample*. The perks of the *XG-Boost model* is that there are a lot of parameters, thus there are a lot of settings that we can vary and since we still have some *over-fitting* there is still room to improve this model. In conclusion this model has the best results and still has a lot of potential for improvement, it is therefore the model that is the best for our data.

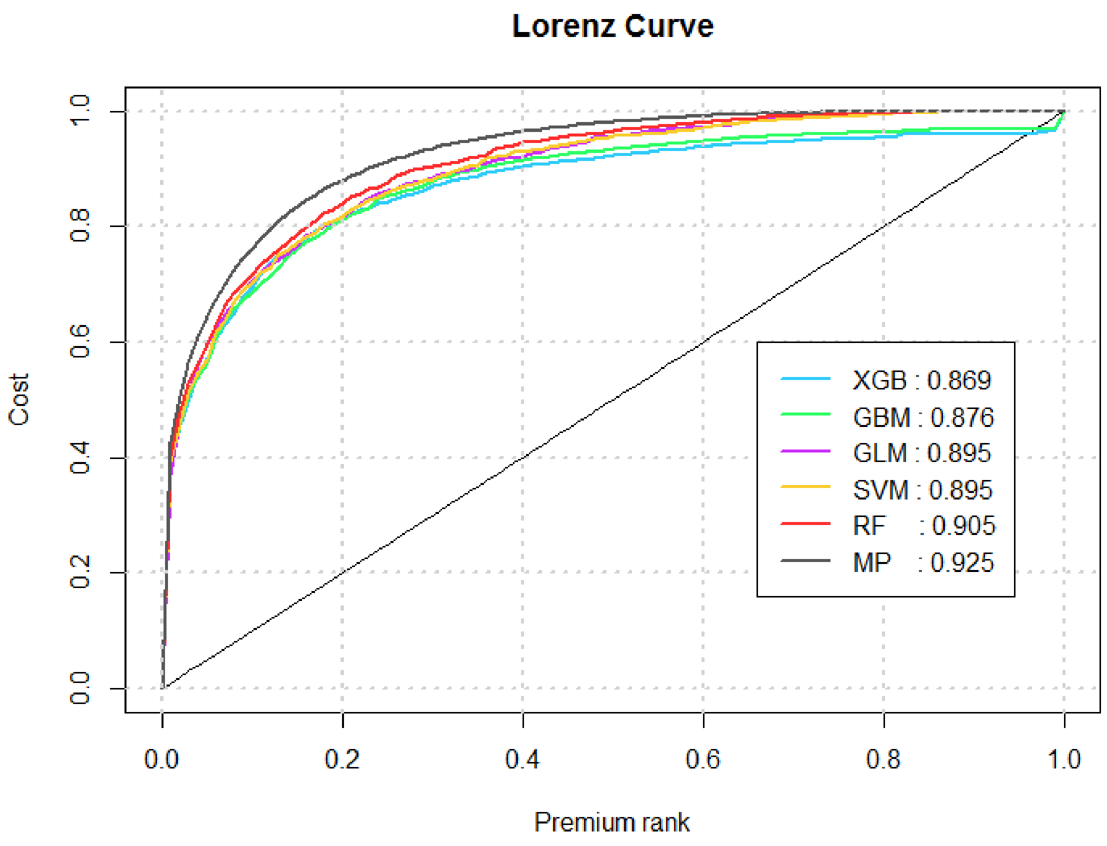


Figure 4.54: *Lorenz curves*

Chapter 5

Reinsurance treaties comparison

One of the main goals of our project is to succeed in applying reinsurance programs to our portfolio. In order to compare the reinsurance program we created an *R-Shiny* application. It can be found at this address: *https://gauthiereldin.shinyapps.io/beshiny/*. The reinsurance treaties we choose to compare are *Excess of Loss* and *Stop-Loss* (Non Proportional Reinsurance : see section 1.5.2.2) . We decided on these options because they are the most suited to reinsurance for natural disasters.

5.1 R-Shiny application

This application was made by creating a *year loss table* using the *event loss table* :

1. Let $e_i,\ i = 1...I$ be the i event of the *event loss table*,
2. We simulated $X_i \underset{iid}{\sim} \mathcal{U}([0\,;1])$ for $i = 1, ..., I$,
3. If $X_i \leq freq(e_i)$, we consider that the event e_i happened.

In this application, we can select :

– The number n of years that we wish to simulate,
– The treaty type : *Excess of Loss*, *Stop-Loss*, both, or none,
– For all the treaties selected we can chose their characteristics (see section 1.5.2.2).

After entering these parameters a graph will show the portion of the claims that are losses and the portion that is ceded to the reinsurer. In output section of the application, we will have :

– The number of claims in the n years,
– The *total loss* per year,
– The amount that is ceded to the reinsurer per year for the specific reinsurance treaty, it is calculated by taking the amount that exceeds the priority line in a *Stop-Loss* treaty and the amount that is between the priority and the line in the *Excess of Loss* treaty
– The *net loss per year*, it is the loss that is not reinsured,

- The observed *pure premium* of event loss table for the *Stop-Loss* or *Excess of Loss*,
- The *technical premium* which is calculated with the following formula :

$$TechnicalPremium = \frac{PurePremium + \beta\ \sqrt{var(TreatyResult)}}{1-\alpha}$$

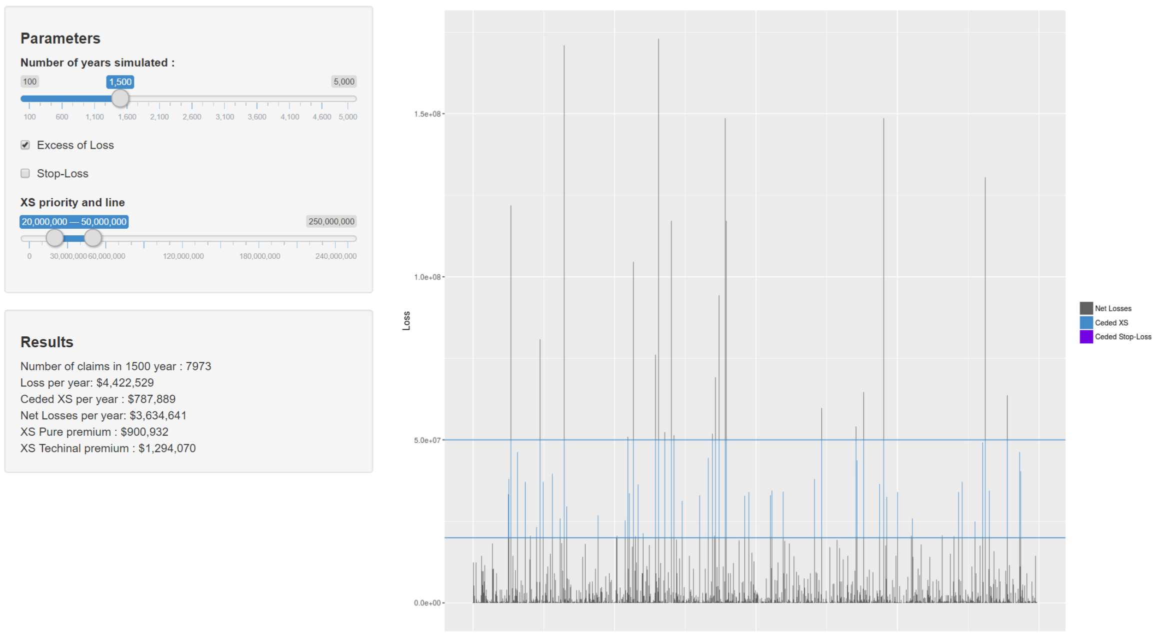
Where α is the expense factor, β is the risk loading factor and where the event follows a binomial distribution, thus :

$$TreatyResult = Loss \times (1 - Freq) \times Freq$$

The parameters α and β depend on the line of business. In the *natural catastrophes* line of business $\alpha \in [0.10\,;0.15]$ and $\beta \in [0.10\,;0.15]$, we decided to set $\alpha = \beta = 0.1$.

5.2 Example

5.2.1 Excess of Loss



Parameters

Number of years simulated : 1,500 (100 – 5,000)

☑ Excess of Loss

☐ Stop-Loss

XS priority and line 20,000,000 — 50,000,000 (0 – 250,000,000)

Results

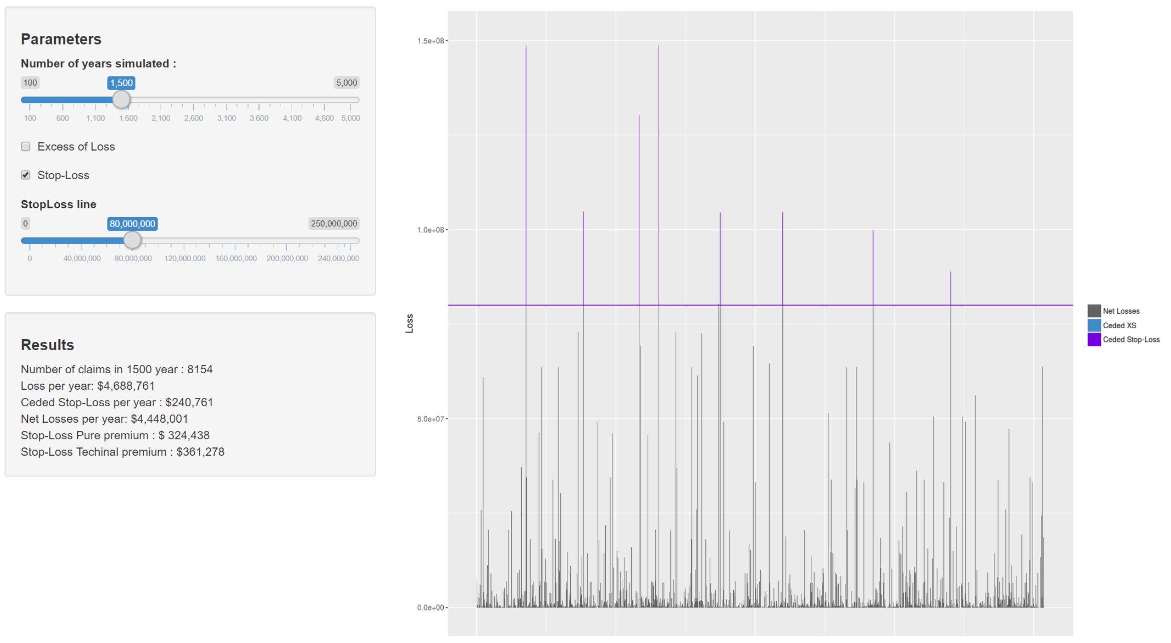
Number of claims in 1500 year : 7973
Loss per year: $4,422,529
Ceded XS per year : $787,889
Net Losses per year: $3,634,641
XS Pure premium : $900,932
XS Techinal premium : $1,294,070

Figure 5.1: *Excess of Loss* reinsurance treaty

In figure 5.1 there is an example of an *Excess of Loss (XS)* treaty simulated over $1,500$ years where the priority is of $\$20$ million and the line is of $\$50$ million. We can see that the number of claims in $1,500$ years is $7,973$. The total loss per year is around $\$4.4$ million, the amount the reinsurer will have to pay of this loss per year with this *Excess of Loss* treaty is around $\$787,889$.

The *pure premium* incurred for this treaty is around $\$900,932$ per year, it is the loss multiplied by the frequency. The *technical premium* takes into account the management expense as well as a risk margin, in this simulated scenario it is of about $\$1.3$ million. We can see in this simulation that *technical premium* is above the ceded loss, which would mean that the reinsurer is not loosing money. Moreover, this technical premium does not include the commercial margin, so this means that the reinsurer should be making some profit from this.

5.2.2 Stop Loss



Parameters

Number of years simulated :

100 — 1,500 — 5,000

100 600 1,100 1,600 2,100 2,600 3,100 3,600 4,100 4,600 5,000

☐ Excess of Loss

☑ Stop-Loss

StopLoss line

0 — 80,000,000 — 250,000,000

0 40,000,000 80,000,000 120,000,000 160,000,000 200,000,000 240,000,000

Results

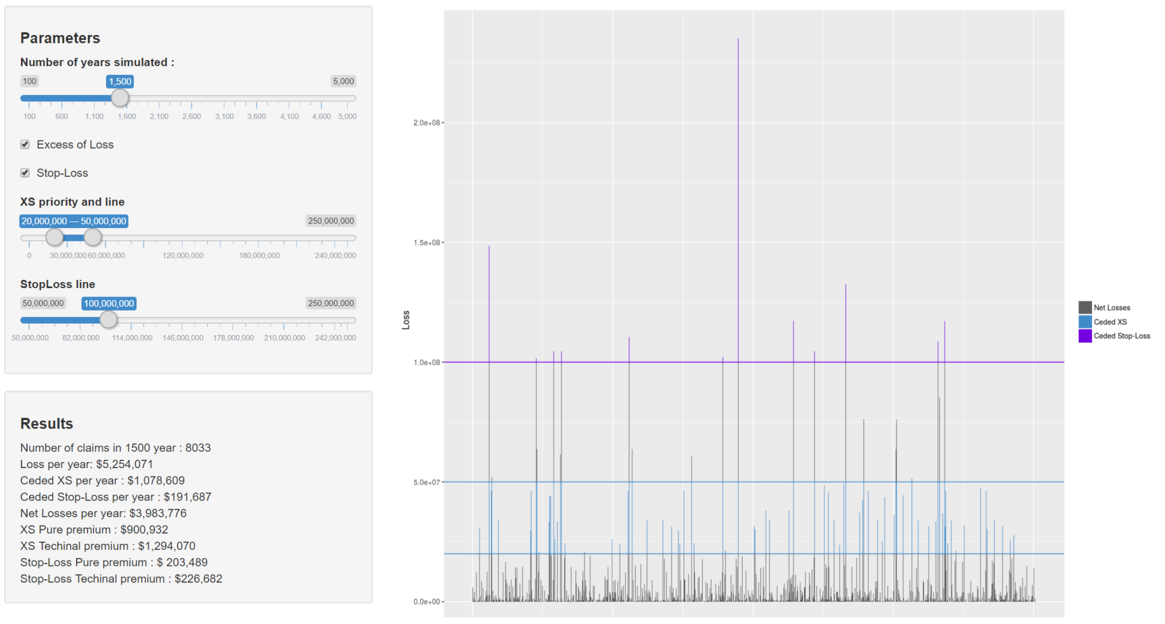
Number of claims in 1500 year : 8154
Loss per year: $4,688,761
Ceded Stop-Loss per year : $240,761
Net Losses per year: $4,448,001
Stop-Loss Pure premium : $ 324,438
Stop-Loss Techinal premium : $361,278

Figure 5.2: *Stop-Loss* reinsurance treaty

We can see in figure 5.2 an example of a *Stop-Loss (SL)* treaty simulated over $1,500$ years where the line is of $\$80$ million. We can see that the number of claims in 1500 years is $8,154$. The total loss per year is around $\$4.69$ million, the amount the reinsurer will have to pay with this *Stop-Loss* treaty per year is of $\$240,761$.

The *pure premium* incurred for this treaty is around $\$324,438$ per year. The technical premium is of $\$361,278$ in this scenario. We can see in this simulation that *technical premium* is above the ceded loss, that would mean that the reinsurer is at least breaking even.

5.2.3 Stop Loss combined with Excess of Loss



Parameters

Number of years simulated : 1,500 (100 – 5,000)

[✓] Excess of Loss

[✓] Stop-Loss

XS priority and line 20,000,000 — 50,000,000 (0 – 250,000,000)

StopLoss line 100,000,000 (50,000,000 – 250,000,000)

Results

Number of claims in 1500 year : 8033
Loss per year: $5,254,071
Ceded XS per year : $1,078,609
Ceded Stop-Loss per year : $191,687
Net Losses per year: $3,983,776
XS Pure premium : $900,932
XS Techinal premium : $1,294,070
Stop-Loss Pure premium : $ 203,489
Stop-Loss Techinal premium : $226,682

Figure 5.3: *Excess of Loss* combined with *Stop-Loss* reinsurance treaty

We can see in figure 5.3 an example of a *Stop-Loss* treaty combined with an *Excess of Loss* treaty simulated over $1,500$ years where the priority of the *XS* is \$20 million and the line of the *XS* is \$50 million while the line of the *SL* is of \$100 million. The number of claims in $1,500$ years is $8,033$. The total loss per year is around \$5.25 million, the amount the reinsurer will have to pay per year is \$1, 078, 609 for the *Excess of Loss* treaty and \$191, 687 for the *Stop-Loss* treaty.

The *pure premium* for the *XS* treaty is around \$900, 932 per year while it is \$203, 489 million per year for the *SL* treaty. The *technical premium* is of \$1, 294, 070 for the *Excess of Loss* treaty and \$226, 682 for the *Stop-Loss* treaty in this scenario. We can see that the *pure premium* the *XS* treaty is smaller than the loss incurred for the reinsurer. This could be bad for the reinsurer which explains why the *technical premium* takes into account a risk margin: to make sure they break even. Here we can see that the *technical premium* is higher than the loss so the reinsurer still breaks even.

Conclusion

As the production cycle of the insurer is reversed, it is critical to have well priced products. These *estimations* must be very precise to make sure that the insurer as well as the reinsurer stay viable. If it is not they could go bankrupt, the goal is therefore to have a good equilibrium between the *premiums* and the losses.

The main aim of this project was to estimate the *pure premium* for a portfolio of *earthquakes* with the help of different computing tools. We found out that the models that were adapted the best to our data was the *XG-Boost* model. It was the model that gave us the lowest *root means square error* on the *training sample* as well as on the *testing sample*. Moreover the *Lorenz curve* for this model was quite good. We also chose this model because it has a great potential for improvement since there are a lot of parameters that we can vary. An extension of this project could be to vary more of the parameters of *XG-Boost* to find an even better model for predicting the pure premium.

A secondary goal of our project was to compare reinsurance programs. We decided on an application that could show the different lines and priority of two different *reinsurance treaties* to see the cost and the premium it would incur for the reinsurer. A furthering of this objective could be to decide on a reinsurance treaty either *Excess of Loss* or *Stop Loss* and determine the appropriate priority and line that would work well for the reinsurer.

We have seen through this project : *data science* techniques and modelization as well as *premium pricing* tools that will be necessary for when we become *actuaries*. It also allowed us to apply a lot of the theory that we have learned over the past two years at the *EURIA* as wells as learned some new theory. We have also developed our programming skills, by learning to code new models such as *XG-Boost*, and how to optimize it, but also by using new *R* packages for data treatments (*data.table*, *MICE*), or data vizulation (*ggplot2*). This *Bureau d'Etude* consisted for us of a very rewarding first professional experience.

References

1. Franck VERMET. *Apprentissage Statistique, EURIA*, 2016.
2. Pierre AILLIOT. *Modèles Lineaire, EURIA*, 2016
3. Pierre AILLIOT. *Valeures extremes, EURIA*, 2016
4. Arthur CHARPENTIER. *Additional thoughts about Lorenz Curves*, 2015
5. Arthur CHARPENTIER. *Improving segmentation with Lorenz Curves*, 2015.
6. Stef VAN BUUREN. *Multivariate Imputation by Chained Equations in R*, 2011.
7. Philippe BESSE. *Apprentissage Statistique & Data mining*, 2008.
8. Laurent ROUVIÈRE. *Introduction aux méthodes d'agrégation : boosting, bagging et forêts aléatoires*, 2015.
9. Leo BREIMAN. *Random Forests*, 2001
10. Swiss Re. *The essential guide to reinsurance*, 2013

Appendix

1 Functions

```r
# Lorenz curve
LorenzCurve <- function(cost,premium)
{
  x = (0:100) / 100
  plot(x,x, type = "l", main = "Lorenz Curve", xlab = "Premium rank", ylab = "Cost")
  l = NULL
  area = NULL
  require(data.table)
  premium = as.data.table(premium)
  color=c("#2ECCFA","#2EFE64","#CC2EFA","#FACC2E","#FE2E2E","#585858")
  for(i in 1:ncol(premium))
  {
    ordercost = cost[order(premium[[i]],decreasing = T)]
    cumcost = cumsum(ordercost)/sum(ordercost)
    lines(x, c(0,quantile(cumcost, 1:100/100)), col = color[i], lwd = 2)
    area = c(area,mean(cumcost))
    grid(lwd = 2)
  }
  legend(x = 0.66,y = 0.60,legend = paste0(colnames(premium), ' : ', round(area,3)),
col = color,lty = c(1,1), lwd = 2)
}

# NA check
checkNAplot <- function(x) {
  vars <- ""
  perc <- 0
  type <- ""
  for (i in 1:ncol(x)) {
    if (anyNA(x[, i])) {
      vars <- c(vars, names(x)[i])
      type <- c(type, "Missing values")
      perc <- c(perc, length(which(is.na(x[, i]))) / (nrow(x)))
    }
  }
  if (length(vars) > 1) {
    vars <- vars[-1]
    type <- type[-1]
    perc <- perc[-1]
    vars <- factor(vars)
    naData <-
      data.frame(variables = vars,
                 type = type,
                 percentage = perc)
    naData$variables <-
      factor(naData$variables, levels = naData$variables[order(naData$perc)])
    plot <-
      ggplot(naData, aes(x = variables, y = percentage)) + geom_bar(stat = "identity
")
    + xlab("Variable") + ylab("Percent missing") + ggtitle("Percentage of missing va
lues")
    + coord_flip()
    print("Checking NA process completed.")
    return(plot)
  }
  else
    print("No NA")
}
```

2 Preparation of the work environment

Loading of the packages, initialization of the *h2o* environnement

```
rm(list = ls())
gc()

remove.packages(pkgs='data.table')
setwd(paste(getwd(), '/1.Data', sep = ''))

load.libraries <-
  c('lazyeval' , 'ggplot2', 'h2o' ,'rpart', 'rpart.plot' , 'mice' , 'readr' ,
    'randomForest' , 'hydroGOF' , 'gbm' , 'xgboost' , 'plotly' , 'MASS' , 'e1071' ,
    'corrplot','ipred','caret', 'Metrics','data.table','stringr','evd','nnet','ROCR'
,
    'leaps','VIM','pacman','mice','lattice','DiagrammeR','Ckmeans.1d.dp','plyr')

install.lib <- load.libraries[!load.libraries %in% installed.packages()]
for (libs in install.lib) {
  install.packages(libs)
}
s
 apply(load.libraries, require, character = TRUE)
rm(libs, install.lib, load.libraries)

h2o.init()
h2o.removeAll()
```

3 Data vizualisation

```
A = dat # /!\ TO RUN THIS YOU NEED TO RUN 4.1 BEFORE
B = LossByLoc # /!\ TO RUN THIS YOU NEED TO RUN 5.1 BEFORE

ggplot(B, aes(x = Loss)) + geom_histogram(bins = 80) +
  labs(x = "Loss", y = "Frequency")

ggplot(B, aes(x = log(Loss))) + geom_histogram(binwidth = 0.1) +
  labs(x = "Log-Loss", y = "Frequency")

ggplot(A, aes(x = OccupancyType)) + geom_bar() + theme(axis.text.x = element_text(
  angle = 90,
  hjust = 1,
  vjust = 0.5,
  size = 6
)) +
  labs(x = "Occupancy Type", y = "Frequency")

ggplot(A, aes(x = RoofType)) + geom_bar() +
  labs(x = "Roof Type", y = "Frequency")

ggplot(A, aes(x = YearBuilt)) + geom_bar() +
  labs(x = "Year Built", y = "Frequency")

ggplot(A, aes(x = NumStories)) + geom_bar() +
  labs(x = "Number of Stories", y = "Frequency") + scale_x_discrete(limits =
                                                                      c("1", "2", "3
", "4", "5", "6", "7", "8", "9", "10", "15", NA))


ggplot(A, aes(x = FloorLevel)) + geom_bar() +
  labs(x = "Floor Level", y = "Frequency") + scale_x_discrete(limits = c("-1", "1",
"2", "3", "4", "5", "6", "7", "8", "9", "15", NA))


names(table(A$StructureType))
A$StructureType[which(A$StructureType == "REINFORCED_CONCRETE_PRECAST_MOMENT_RESISTI
GING_FRAME")] =
  "CONCRETE_RESISTING_FRAME"
ggplot(A, aes(x = StructureType)) + geom_bar() + theme(axis.text.x = element_text(
   angle = 90,
   hjust = 1,
   vjust = 0.5,
   size = 7
 )) +
   labs(x = "Structure Type", y = "Frequency")


 ggplot(A, aes(x = InsuredValue)) + geom_histogram(bins = 90) +
   labs(x = "InsuredValue", y = "Frequency")

ggplot(A, aes(x = log(InsuredValue))) + geom_histogram(binwidth = 0.1) +
  labs(x = "Log-InsuredValue", y = "Frequency")

ggplot(A, aes(x = gshap)) + geom_histogram(bins = 90) +
  labs(x = "gshap", y = "Frequency")

####boxplot de chaque variable par rapport a PurePremium

ggplot(A, aes(x = OccupancyType, y = log(PP))) + geom_boxplot() + theme(axis.text.x
=
                                                                          element_te
xt(
                                                                            angle =
90,
                                                                            hjust =
1,
                                                                            vjust =
0.5,
                                                                            size = 6
                                                                          )) +
  labs(x = "OccupancyTpe", y = "log-PurePremium")

ggplot(A, aes(x = RoofType, y = log(PP))) + geom_boxplot() +
  labs(x = "RoofType", y = "log-PurePremium")

ggplot(A, aes(x = NumStories, y = log(PP))) + geom_boxplot() +
  labs(x = "NumStories", y = "log-PurePremium") + scale_x_discrete(limits =
                                                                      c("1", "2", "3"
, "4", "5", "6", "7", "8", "9", "10", "15", NA))

ggplot(A, aes(x = YearBuilt, y = log(PP))) + geom_boxplot() +
  labs(x = "YearBuilt", y = "log-PurePremium")

ggplot(A, aes(x = FloorLevel, y = log(PP))) + geom_boxplot() +
  labs(x = "FloorLevel", y = "log-PurePremium") + scale_x_discrete(limits =
                                                                      c("-1", "1", "2
", "3", "4", "5", "6", "7", "8", "9", "15", NA))


ggplot(A, aes(x = StructureType, y = log(PP))) + geom_boxplot() + theme(axis.text.x
=
                                                                          element_te
xt(
                                                                            angle =
90,
                                                                            hjust =
1,
                                                                            vjust =
0.5,
                                                                            size = 7
                                                                          )) +
  labs(x = "StructureType", y = "log-PurePremium")
```

```
ggplot(A, aes(x = gshap, y = log(PP))) + geom_point(colour = "#595959") +
  geom_smooth(
    method = "lm",
    se = FALSE,
    color = '#088A08',
    size = 1
  ) +
  labs(x = "gshap", y = "log-PurePremium")

ggplot(A, aes(x = log(InsuredValue), y = log(PP))) + geom_point(colour =
                                                                  "#595959") + geom_
smooth(
                                                                    method = "lm",
                                                                    se = FALSE,
                                                                    color = '#088A08
',
                                                                    size = 1
                                                                  ) +
  labs(x = "log-InsuredValue", y = "log-PurePremium")
  aggr(
    B,
    col = c('white', 'grey'),
    labels = names(data),
    cex.axis = .55,
    gap = 3,
    ylab = c("Histogram of missing data", "Pattern")
  )
aggr_plot <- aggr(B, col=c('white','grey'), labels=names(data), cex.axis=.55, gap=3,
ylab=c("Histogram of missing data","Pattern"))
```

4 Loss estimation

4.1 Data treatment

Data loading

```
load(paste(getwd(),"/EventLossTableBySite_sample.RData",sep=""))
load(paste(getwd(),"/expo_sample.RData",sep=""))
load(paste(getwd(),"/SiteInfo.RData",sep=""))
```

Data treatment

```
# Rename the three columns of expo sample and site info to be able to merge the thre
e tables
names(expo_sample)[1] = "LocationName"
names(SiteInfo)[1] = "LocationID"

# Merge of the tables
z = merge(EventLossTableBySite_sample, expo_sample, by = "LocationName")
z = merge(z, SiteInfo, by = "LocationID", sort = TRUE)

# Table of Loss by LocationID
L = aggregate(z$Loss, list(z$LocationID), sum)
names(L) = c("LocationID", "Loss")
LossByLoc = unique((merge(z[,-(4:6)], L, by = "LocationID")))

# Table of Loss by EventID
LossByEvent = aggregate(z$Loss, list(z$EventID), sum)
names(LossByEvent) = c("EventID", "Loss")
```

```
# Pure premium of each site
PurePremium = aggregate(z$Loss * z$Freq, list(z$LocationID), sum)
names(PurePremium) = c("LocationID", "PurePremium")

# Removing duplicates YearBuilt and OCCTYPE
LossByLoc = LossByLoc[,-c(1:4,14,15)]

# Replace UNKNOWN by NA
LossByLoc$StructureType[LossByLoc$StructureType == "UNKNOWN"] = NA
LossByLoc$RoofType[LossByLoc$RoofType == "UNKNOWN"] = NA

# Replacing NA by new modality "VM"
LossByLoc$RoofType[which(is.na(LossByLoc$RoofType))]="VM"
LossByLoc$StructureType[which(is.na(LossByLoc$StructureType))]="VM"
LossByLoc$OccupancyType[which(is.na(LossByLoc$OccupancyType))]="VM"
LossByLoc$YearBuilt[which(is.na(LossByLoc$YearBuilt))]="VM"
LossByLoc$NumStories[which(is.na(LossByLoc$NumStories))]="VM"
LossByLoc$FloorLevel[which(is.na(LossByLoc$FloorLevel))]="VM"

# Transforming qualitative variables into factor variable
for (i in colnames(LossByLoc[, sapply(LossByLoc, is.character)])) {
  LossByLoc[, i] <- as.factor(unlist(LossByLoc[, i]))
}
rm(i)

# Deletion of the 268 observations'without InsuredValue or gshap
LossByLoc = LossByLoc[-which(is.na(LossByLoc$InsuredValue)),]
LossByLoc = LossByLoc[-which(is.na(LossByLoc$gshap)),]

# Creation of LogLoss
LogLossByLoc = LossByLoc
LogLossByLoc$Loss = sapply(LossByLoc$Loss, log)
names(LogLossByLoc)[dim(LogLossByLoc)[2]] = "LogLoss"
```

Save output

```
save(LogLossByLoc, file = "3.Output/logdat.Rdata")
save(LossByLoc, file = "3.Output/dat.Rdata")
```

Creation of training and testing samples

```
n = dim(LossByLoc)[1]
BA = sample(1:n, dim(LossByLoc)[1] * 0.70)   #training sample
BT = setdiff(1:n, BA)
```

4.2 Linear model

Linear model on loss

```
fit1 = lm(
  Loss ~.,
  data = LossByLoc
)
summary(fit1)
par(mfrow = c(2, 2))
plot(fit1)
```

Linear model on Log-loss

```
fit2 = lm(
  LogLoss ~.,
  data = LogLossByLoc
)
summary(fit2)
par(mfrow = c(2, 2))
plot(fit2)
```

4.3 Regression trees

Removing some levels

```
# the first step is to remove some of the modality to avoid new levels, check if thi
s is not removing too much observations
length(LossByLoc$StructureType[LossByLoc$StructureType == "WOOD_FRAME_MODERN"][-whic
h(is.na(LossByLoc$StructureType[LossByLoc$StructureType == "WOOD_FRAME_MODERN"]))])
length(LossByLoc$StructureType[LossByLoc$StructureType == "EARTHQUAKE_RESISTIVE"][-w
hich(is.na(LossByLoc$StructureType[LossByLoc$StructureType == "EARTHQUAKE_RESISTIVE"
]))])
length(LossByLoc$StructureType[LossByLoc$StructureType == "MOBILE_HOME"][-which(is.n
a(LossByLoc$StructureType[LossByLoc$StructureType == "MOBILE_HOME"]))])
length(LossByLoc$StructureType[LossByLoc$StructureType == "REINFORCED_CONCRETE_PRECA
ST_MOMENT_RESISTIGING_FRAME"][- which(is.na(LossByLoc$StructureType[LossByLoc$Struct
ureType == "REINFORCED_CONCRETE_PRECAST_MOMENT_RESISTIGING_FRAME"]))])
length(LossByLoc$StructureType[LossByLoc$StructureType == "MASONRY"][-which(is.na(Lo
ssByLoc$StructureType[LossByLoc$StructureType == "MASONRY"]))])
length(LossByLoc$StructureType[LossByLoc$StructureType == "STEEL"][-which(is.na(Loss
ByLoc$StructureType[LossByLoc$StructureType == "STEEL"]))])
length(LossByLoc$FloorLevel[LossByLoc$FloorLevel == "7"][-which(is.na(LossByLoc$Floo
rLevel[LossByLoc$FloorLevel == "7"]))])
length(LossByLoc$FloorLevel[LossByLoc$FloorLevel == "8"][-which(is.na(LossByLoc$Floo
rLevel[LossByLoc$FloorLevel == "8"]))])
length(LossByLoc$FloorLevel[LossByLoc$FloorLevel == "9"][-which(is.na(LossByLoc$Floo
rLevel[LossByLoc$FloorLevel == "9"]))])
length(LossByLoc$OccupancyType[LossByLoc$OccupancyType == "COMMUNICATION_RADIO_TV"])
length(LossByLoc$OccupancyType[LossByLoc$OccupancyType == "PARKING"])
length(LossByLoc$OccupancyType[LossByLoc$OccupancyType == "SANITARY_SEWER"])
length(LossByLoc$OccupancyType[LossByLoc$OccupancyType == "EXHIBITION"])

#   REINFORCED_CONCRETE : 3
#   MANSONRY : 4
#   STELL : 5
#   FloorLevel :
#     8 : 2
#     7 : 4
#   All others : 1
# => We can delete these levels, it will not have an impact on our data base

LossByLoc = LossByLoc[-which(LossByLoc$StructureType == "WOOD_FRAME_MODERN"),]
LossByLoc = LossByLoc[-which(LossByLoc$StructureType == "EARTHQUAKE_RESISTIVE"),]
LossByLoc = LossByLoc[-which(LossByLoc$StructureType == "MOBILE_HOME"),]
LossByLoc = LossByLoc[-which(
  LossByLoc$StructureType == "REINFORCED_CONCRETE_PRECAST_MOMENT_RESISTIGING_FRAME"
),]
LossByLoc = LossByLoc[-which(LossByLoc$StructureType == "MASONRY"), ]
LossByLoc = LossByLoc[-which(LossByLoc$StructureType == "STEEL"), ]
LossByLoc = LossByLoc[-which(LossByLoc$OccupancyType == "COMMUNICATION_RADIO_TV"),]
LossByLoc = LossByLoc[-which(LossByLoc$OccupancyType == "PARKING"),]
LossByLoc = LossByLoc[-which(LossByLoc$OccupancyType == "SANITARY_SEWER"),]
LossByLoc = LossByLoc[-which(LossByLoc$OccupancyType == "EXHIBITION"),]
LossByLoc = LossByLoc[-which(LossByLoc$OccupancyType ==
"HEAVY_FABRICATION_ASSEMBLY" ), ]
```

```
LossByLoc = LossByLoc[-which(LossByLoc$FloorLevel == "7"), ]
LossByLoc = LossByLoc[-which(LossByLoc$FloorLevel == "8"),]
LossByLoc = LossByLoc[-which(LossByLoc$FloorLevel == "9"),]
LossByLoc = LossByLoc[-which(LossByLoc$NumStories == "9"),]
```

First regression tree

```
dev.off()
tree.reg = rpart(Loss ~ .,
                 minsplit = 3,
                 xval = 2,
                 data = LossByLoc[BA, ])
prp(tree.reg, extra = 1)
printcp(tree.reg)
plotcp(tree.reg)

tree.opti = prune(tree.reg, cp = tree.reg$cptable[which.min(tree.reg$cptable[, 4]),
1])
prp(tree.opti, extra = 1)

rmse(predict(tree.opti, LossByLoc[BT, ]), LossByLoc[BT, ]$Loss)
rmse(predict(tree.opti, LossByLoc[BA, ]), LossByLoc[BA, ]$Loss)
```

Mean error on 100 trees for Loss by cross validation

```
k = 100
merr = NULL

for (i in 1:k) {
  n = dim(LossByLoc)[1]
  BA = sample(1:n, dim(LossByLoc)[1] * 0.60)    #training sample
  BT = setdiff(1:n, BA)

  tree.reg = rpart(Loss ~ .,
                   minsplit = 3,
                   xval = 2,
                   data = LossByLoc[BA, ])
  tree.opti = prune(tree.reg, cp = tree.reg$cptable[which.min(tree.reg$cptable[, 4])
, 1])
  pred = as.numeric(predict(tree.opti, LossByLoc[BT, ]))
  err = pred - LossByLoc$Loss[BT]
  merr[i] = mean(err)
}
err = mean(abs(merr))
err
```

Mean error on 100 trees for LogLoss by cross validation

```
k = 1e3
merr = NULL
for (i in 1:k) {
  n = dim(LogLossByLoc)[1]
  BA = sample(1:n, dim(LogLossByLoc)[1] * 0.60)    #training sample
  BT = setdiff(1:n, BA)

  tree.reg = rpart(
    LogLoss ~ InsuredValue + gshap + as.factor(RoofType) +
      as.factor(StructureType) + as.factor(OccupancyType) +
      as.factor(OCCSCHEME) + as.factor(BLDGCLASS) + as.factor(FloorLevel) +
      as.factor(NumStories),
    minsplit = 2,
    xval = 2,
    data = LogLossByLoc[BA, ]
  )
  tree.opti = prune(tree.reg, cp = tree.reg$cptable[which.min(tree.reg$cptable[, 4])
, 1])
```

Mean error on 100 trees for LogLoss by cross validation

```
  tree.opti = prune(tree.reg, cp = tree.reg$cptable[which.min(tree.reg$cptable[, 4])
, 1])
  pred = as.numeric(predict(tree.opti, LogLossByLoc[BT, ]))
  err = exp(pred) - LossByLoc$Loss[BT]
  merr[i] = mean(err)
}
err = mean(abs(merr))
err
```

Mean error on 100 trees for LogLoss by cross validation

```
n = dim(LossByLoc)[1]

for (i in 1:n) {
  BA = (1:n)[-i]
  BT = setdiff(1:n, BA)

  tree.reg = rpart(
    Loss ~ InsuredValue + gshap + as.factor(RoofType) +
      as.factor(StructureType) + as.factor(OccupancyType) +
      as.factor(OCCSCHEME) + as.factor(BLDGCLASS) + as.factor(FloorLevel) +
      as.factor(NumStories),
    minsplit = 10,
    xval = 2,
    data = LossByLoc[BA, ]
  )
  tree.opti = prune(tree.reg, cp = tree.reg$cptable[which.min(tree.reg$cptable[, 4])
, 1])
  merr[i] = mean(as.numeric(predict(tree.opti, LossByLoc[BT, ])) - LossByLoc$Loss[BT
])
}
err = mean(abs(merr))
err
```

4.4 Random forest

Adjustment of the random forest

```
RandomForest <- randomForest(Loss ~.,
                             data=LossByLoc[BA,],ntree=10)
summary(RandomForest)
importance(RandomForest)
varImpPlot(RandomForest)

rmse(predict(RandomForest,LossByLoc[BA,]),LossByLoc[BA,]$Loss)
rmse(predict(RandomForest,LossByLoc[BT,]),LossByLoc[BT,]$Loss)

LossByLoc$Sample=0
LossByLoc[BA,]$Sample="Training"
LossByLoc[BT,]$Sample="Testing"
LossByLoc$Error =0
LossByLoc[BA,]$Error= abs(predict(RandomForest,LossByLoc[BA,])-LossByLoc[BA,]$Loss)
LossByLoc[BT,]$Error=abs(predict(RandomForest,LossByLoc[BT,])-LossByLoc[BT,]$Loss)
```

Plot of the error

```
ggplot(LossByLoc[which(LossByLoc$Loss<1e9),],aes(x = Loss,y=Error,color=Sample))+geo
m_point(size=1.5)
LossByLoc$Sample=NULL
LossByLoc$Error=NULL
```

4.5 Logistic regression + random forest

Finding the threshold for extreme values

```
tlim = c(1e7, 5e7)
mrlplot(LossByLoc$Loss, tlim = tlim)

s = 3e7
```

Creation of a new variable “cata”, equal to 1 if normal value and 2 if extreme value

```
LossByLoc[, dim(LossByLoc)[2] + 1] = 1
names(LossByLoc)[dim(LossByLoc)[2]] = "cata"
LossByLoc[which(LossByLoc$Loss > s), dim(LossByLoc)[2]] = 2
LossByLoc$cata = as.factor(LossByLoc$cata)
```

Logistic regression

```
reglog <- multinom(cata ~ InsuredValue + gshap, data = LossByLoc[BA, ])

# Threshold optimization
par(mfrow = c(2, 1))
regtest = predict(reglog, LossByLoc[BA, ], type = "probs")
S = seq(0.2, 0.9, by = 0.005)
l = length(S)
err = rep(0, l)
for (i in 1:l)
{
  Pred.tr = (regtest > S[i])
  a = table(LossByLoc[BA, ]$cata, Pred.tr)
  err[i] = (a[1, 2] + a[2, 1]) / length((LossByLoc[BA, ]$cata))
}
plot(S, err)
lines(S, err)
S[which.min(err)]
min(err)


regtest = predict(reglog, LossByLoc[BT, ], type = "probs")
S = seq(0.2, 0.9, by = 0.005)
l = length(S)
err = rep(0, l)
for (i in 1:l)
{
  Pred.tr = (regtest > S[i])
  a = table(LossByLoc[BT, ]$cata, Pred.tr)
  err[i] = (a[1, 2] + a[2, 1]) / length((LossByLoc[BT, ]$cata))
}
plot(S, err)
lines(S, err)
S[which.min(err)]
min(err)

dev.off()
regtraining <- predict(reglog, LossByLoc[BT, ], type = "probs")
fr <-
  data.frame(score = regtraining, label = LossByLoc[BT, ]$cata)
pred <- prediction(fr$score, fr$label)
```

```r
# Error rate in function of the threshold
perf1 <- performance(pred, "fpr")
plot(perf1)
perf1 <- performance(pred, "fnr")
plot(perf1, col = 'red', add = TRUE)
perf1 <- performance(pred, "err")
plot(perf1, col = 'green', add = TRUE)

# => Keep default threshold : 0.5


# ROC curve
regtraining <- predict(reglog, LossByLoc[BT, ], type = "probs")
fr <-
  data.frame(score = regtraining, label = LossByLoc[BT, ]$cata)
pred <- prediction(fr$score, fr$label)
perf <- performance(pred, "tpr", "fpr")
plot(perf, col = 'black')
#Calcul du AUC (aire sous la courbe)
perf2 <- performance(pred, "auc")
perf2@y.values[[1]]
```

Adjusting the forests to the two predicted types of observations

```r
# Prediction of the Loss when cata = 2
suba = LossByLoc[which(LossByLoc[BA, ]$cata == 2), ]
subt = LossByLoc[which(LossByLoc[BT, ]$cata == 2), ]

rf2 <- randomForest(Loss ~ . - cata,
                    data = suba, ntree = 10)
summary(rf2)
importance(rf2)
varImpPlot(rf2)

regapp <- as.numeric(predict(rf2, suba))
erra = abs(regapp - suba$Loss)

regtest <- as.numeric(predict(rf2, subt))
errt = abs(regtest - subt$Loss)

mean(errt)
mean(erra)

# Prediction of the Loss when cata = 1
suba = LossByLoc[which(LossByLoc[BA, ]$cata == 1), ]
subt = LossByLoc[which(LossByLoc[BT, ]$cata == 1), ]

rf1 <- randomForest(Loss ~ . - cata,
                    data = suba, ntree = 10)
summary(rf1)
importance(rf1)
varImpPlot(rf1)

regapp <- as.numeric(predict(rf1, suba))
erra = abs(regapp - suba$Loss)

regtest <- as.numeric(predict(rf1, subt))
errt = abs(regtest - subt$Loss)

mean(errt)
mean(erra)
```

Whole model prediction

```
cata = predict(reglog, LossByLoc[BA, ])
LossByLoc[BA, dim(LossByLoc)[2]] = cata

suba1 = LossByLoc[BA, ][which(LossByLoc[BA, ]$cata == 1), ]
suba2 = LossByLoc[BA, ][which(LossByLoc[BA, ]$cata == 2), ]
suba1$pred <- as.numeric(predict(rf1, suba1))
suba2$pred <- as.numeric(predict(rf2, suba2))

final = rbind(suba1, suba2)
rmse(final$pred, final$Loss)

cata = predict(reglog, LossByLoc[BT, ])
LossByLoc[BT, dim(LossByLoc)[2]] = cata

suba1 = LossByLoc[BT, ][which(LossByLoc[BT, ]$cata == 1), ]
suba2 = LossByLoc[BT, ][which(LossByLoc[BT, ]$cata == 2), ]
suba1$pred <- as.numeric(predict(rf1, suba1))
suba2$pred <- as.numeric(predict(rf2, suba2))

final = rbind(suba1, suba2)
rmse(final$pred, final$Loss)
```

5 Rate estimation

5.1 Data treatment

```
install.package('data.table')
install.package('stringr')
library(data.table)
library(stringr)

ELTperSite <-
get(load(file = file.path(
  "1.Data", "EventLossTableBySite_sample.RData"
)))
expo <- get(load(file = file.path("1.Data", "expo_sample.RData")))
load(file = file.path("1.Data", "SiteInfo.RData"))

# List elements
# ls()
rm(EventLossTableBySite_sample)
rm(expo_sample)
ls()

# Caculate Pure Premium
head(ELTperSite)
summary(ELTperSite)
ELTperSite[, PP := Loss * Freq, ]
ELTperSite[, summary(PP)]

# Merge
head(ELTperSite)
dim(ELTperSite)
head(SiteInfo)
head(expo)
```

```
# LocationID=LOCNUM
dat <-
  merge(ELTperSite,
        SiteInfo,
        by.x = 'LocationID',
        by.y = 'LOCNUM',
        all.x = T)
dim(dat)
head(dat)

# LocationName = Id
AllMergedData <-
  merge(dat,
        expo,
        by.x = 'LocationName',
        by.y = 'Id',
        all.x = T)
head(AllMergedData)
rm(dat)
save(AllMergedData, file = "3.Output/AllMergedData.Rdata")

# PP Per site
head(AllMergedData)
AllMergedData[, summary(LocationID), ]
PPbySite <-
  AllMergedData[, list(PP = sum(PP), InsuredValue = mean(InsuredValue)), by =
                  c("LocationID")]
dim(PPbySite)
varnames <- names(AllMergedData)[-c(1, 3, 4, 5, 6, 7, 8, 20)]
# varnames
# [1] "LocationID"    "BLDGCLASS"     "OCCSCHEME"     "OCCTYPE"       "YEARBUILT"
"gshap"
# [7] "RoofType"      "StructureType" "OccupancyType" "YearBuilt"     "NumStories"
"FloorLevel"
x <- AllMergedData[, varnames, with = FALSE]
setkey(x, "LocationID")
x1 <- unique(x)
PPbySite <- merge(PPbySite, x1, by = "LocationID", all.x = T)
save(PPbySite, file = "3.Output/PPbySite.Rdata")

# Total ptf Loss Per Event
ELT <-
  AllMergedData[, list(Loss = sum(Loss), Freq = mean(Freq)), by = c("EventID")]
save(ELT, file = "3.Output/ELT.Rdata")
summary(PPbySite)

# NA treatment 1
dat <- PPbySite[!is.na(InsuredValue), ]

dat$YEARBUILT
dat[, YEARBUILT := as.numeric(substring(YEARBUILT, 1, 4)), ]
dat[YEARBUILT == 9999, YEARBUILT := NA, ]
dat$StructureType[dat$StructureType == "UNKNOWN"] = NA
dat$RoofType[dat$RoofType == "UNKNOWN"] = NA
dat$OccupancyType[which(is.na(dat$OccupancyType))] = NA
dat$YearBuilt[which(is.na(dat$YearBuilt))] = NA
dat$NumStories[which(is.na(dat$NumStories))] = NA dat
$FloorLevel[which(is.na(dat$FloorLevel))] = NA

# Convert into factor
for (i in colnames(dat[, sapply(dat, is.character)])) {
  dat[, i] <- as.factor(unlist(dat[, i]))

 m(i)

# Removing YearBuilt, OCCSCHME and Occtype because they are dupplicate
dat[, OCCSCHEME := NULL, ] #OccupancyType
dat[, OCCTYPE := NULL, ] #OccupancyType
dat[, YEARBUILT := NULL, ] #YearBuilt
```

}
r

```
# Calulate pure premium
dat[, rate := PP / InsuredValue, ]

# NA treatement 2 : MICE method
imputed.data = mice(dat,
                    m = 1,
                    method = 'rf',
                    maxit = 5)
completeData = complete(imputed.data, 1, include = FALSE) #sum(is.na(completeData))
il ne reste plus de NA
xyplot(imputed.data, YearBuilt ~ gshap, main = "V?rification de l'imputation")
dat.imput = completeData
rm(completeData)
rm(imputed.data)
dat = dat.imput

# Creation of the training and testing samples
set.seed(1) # fix the random generator to build the samples because this one doesn't
have problems of new levels
dat = as.data.frame(dat)
n = dim(dat)[1]
TR = sample(1:n, dim(dat)[1] * 0.70) #training
TS = setdiff(1:n, TR) #test
datTR = dat[TR, -c(1:3)]
datTS = dat[TS, -c(1:3)]
```

5.2 Random forest

First Random forest

```
rf <- randomForest(rate ~ ., data = datTR,mtry=2,ntree=10)
varImpPlot(rf)
rmse(predict(rf, datTR) *  dat[TR,]$InsuredValue, dat[TR,]$PP)
rmse(predict(rf, datTS) * dat[TS,]$InsuredValue, dat[TS,]$PP)
```

Hyper-parametrization

```
ete = NULL
etr = NULL                                                    }
l = seq(from = 1, to = 8, by = 1) #nombre de variable explicatives
n = seq(from = 5, to = 600, by = 20) #nombre d'arbre
m = seq(from = 1, to = 8, by = 1)
b <- sample(1:nrow(x), nrow(x) * 0.7, replace = F)
for (h in 1:length(l)) {
  ete[[h]] = matrix(rep(0, length(n) * length(m)), ncol = length(n))
}
etr = ete

for (u in 1:length(l)) {
  x = dat[TR, -c(1:3)]
  for (v in 1:(dim(x)[2] - l[u] - 1)) {
    rm <-
      h2o.randomForest(
        x = 1:(dim(x)[2] - 1),
        y = length(x),
        y = length(x),
        training_frame = as.h2o(x)
      )
    x = x[, -which(colnames(x) == h2o.varimp(rm)$variable[length(h2o.varimp(rm)$vari
able)])]
  }
  c=1
  for (i in m) {
    if (i <= l[u]) {
      k = 1
      for (j in n) {
        set.seed(1212)
        f <- randomForest(rate ~ .,
                          data = x,
                          ntree = j,
                          mtry = i)
        etr[[u]][c, k] = rmse(predict(f, datTR) * dat[TR, ]$InsuredValue, dat[TR, ]$
PP)
        ete[[u]][c, k] = rmse(predict(f, datTS) * dat[TS, ]$InsuredValue, dat[TS, ]$
PP)
        k = k + 1
        print(k)
        print(c)
      }
    }
    c = c + 1
  }
}
save(ete, file = 'ete.Rdata')
save(etr, file = 'etr.Rdata')

rferr = NULL
for (i in 1:length(l)) {
  rferr$var = c(rferr$var, rep(l[i], length(ete[[i]][which(ete[[i]] != 0)])))
  rferr$varforet = c(rferr$varforet, sort(rep(m[m - l[i] <= 0], length(n))))
  rferr$arbre = rep(n, length(rferr$var))
  rferr$errtest = c(rferr$errtest, as.numeric(ete[[i]][which(ete[[i]] != 0)]))
  rferr$errapp = c(rferr$errapp, as.numeric(etr[[i]][which(etr[[i]] != 0)]))
}
View(rferr)
rferr <- as.data.frame(rferr)

rferr$sumerr = rferr$errapp + rferr$errtest
rferr[which.min(rferr$errapp), ]
rferr[which.min(rferr$errtest), ]
rferr[which.min(rferr$sumerr), ]

plot_ly(
  rferr,
  x =  ~ arbre,
  y =  ~ varforet,
  z =  ~ err,
  color = ~ var
)

j = rferr[which.min(rferr$errtest), ]$arbre # j = 145
i = rferr[which.min(rferr$errtest), ]$varforet # i = 2
u = rferr[which.min(rferr$errtest), ]$var # u = 7
```

Final model

```
x = dat[TR,-c(1:3)]
for (v in 1:(dim(x)[2] - u - 1)) {
  rm <-
    h2o.randomForest(
      x = 1:(dim(x)[2] - 1),
      y = length(x),
      training_frame = as.h2o(x)
    )
  x = x[,-which(colnames(x) == h2o.varimp(rm)$variable[length(h2o.varimp(rm)$variabl
e)])]
}

rf.final <- randomForest(rate ~ .,
                         data = datTR,
                         ntree = j,
                         mtry = i)
varImpPlot(rf.final)

rmse(predict(rf.final, datTR) * dat[TR,]$InsuredValue, dat[TR,]$PP)
rmse(predict(rf.final, datTS) * dat[TS,]$InsuredValue, dat[TS,]$PP)
```

```r
e1tr = NULL
e1tr$pred = predict(rf.final, datTR) *  dat[TR,]$InsuredValue
e1tr$pp = dat[TR,]$PP
e1tr$err = e1tr$pp - e1tr$pred
e1tr$Model = "Optimized"
e1tr = as.data.frame(e1tr)

e2tr = NULL
e2tr$pred = predict(rf, datTR) *  dat[TR,]$InsuredValue
e2tr$pp = dat[TR,]$PP
e2tr$err = e2tr$pp - e2tr$pred
e2tr$Model = "Default"
e2tr = as.data.frame(e2tr)
etr = as.data.frame(rbind(e1tr, e2tr))


e1ts = NULL
e1ts$pred = predict(rf.final, datTS) *  dat[TS,]$InsuredValue
e1ts$pp = dat[TS,]$PP
e1ts$err = e1ts$pp - e1ts$pred
e1ts$Model = "Optimized"
e1ts = as.data.frame(e1ts)

e2ts = NULL
e2ts$pred = predict(rf, datTS) *  dat[TS,]$InsuredValue
e2ts$pp = dat[TS,]$PP
e2ts$err = e2ts$pp - e2ts$pred
e2ts$Model = "Default"
e2ts = as.data.frame(e2ts)
ets = as.data.frame(rbind(e1ts, e2ts))

ggplot(etr[which(etr$pp < 1e5), ], aes(x = pp, y = abs(err))) +  geom_point(aes(colo
ur = Model, shape = Model)) + geom_smooth(aes(colour = Model), method = lm, se=FALSE
, fullrange = TRUE) + geom_abline(slope = 0, intercept = 0,lwd =.1)
ggplot(ets[which(ets$pp < 1e5), ], aes(x = pp, y = abs(err))) +  geom_point(aes(colo
ur = Model, shape = Model)) + geom_smooth(aes(colour=Model),method=lm, se=FALSE, ful
lrange=TRUE )+ geom_abline(slope=0, intercept=0,lwd=.1)

ggplot(e2tr[which(e2tr$pp < 1e5), ], aes(x = pp, y = err)) + geom_point() + geom_abl
ine(slope = 0, intercept = 0, lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20
),fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "re
d",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)
ggplot(e2ts[which(e2ts$pp < 1e5), ], aes(x = pp, y = err)) + geom_point() + geom_abl
ine(slope = 0, intercept = 0, lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20
),fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "re
d",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)

ggplot(e1tr[which(e1tr$pp < 1e5), ], aes(x = pp, y = err)) + geom_point() + geom_abl
ine(slope = 0, intercept = 0 ,lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20
),fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "re
d",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)
ggplot(e1ts[which(e1ts$pp < 1e5), ], aes(x = pp, y = err)) + geom_point() + geom_abl
ine(slope = 0, intercept = 0, lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20
),fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "re
d",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)
```

Lorenz curve

```
preddtr = predict(rf, dat[TR, ]) * dat[TR,]$InsuredValue
predotr =  predict(rf.final, dat[TR, ]) * dat[TR,]$InsuredValue

preddts = predict(rf, dat[TS, ]) * dat[TS,]$InsuredValue
predots =  predict(rf.final, dat[TS, ]) * dat[TS,]$InsuredValue

premium = as.matrix(NA)
cost = as.vector(dat[TS,]$PP)
premium = cbind(preddts, cost)
colnames(premium) = c("RF ", "PM")
LorenzCurve(cost, premium)

premium = as.matrix(NA)
cost = as.vector(dat[TS,]$PP)
premium = cbind(preddts, predots, cost)
colnames(premium) = c("Deault ", "Optim  ", "Perfect")
LorenzCurve(cost, premium)
```

5.3 Generalized Linear Model

Finding the distribution

```
# 4.1 Fiding the distribution
x = dat$rate
den <- density(x)
d <- data.frame(x = den$x,
                y = den$y,
                limits = c(0, 0))
fit.params <- fitdistr(dat$rate, "gamma")
ggplot(data = d) +
  geom_histogram(data = as.data.frame(x), aes(x = x, y = ..density..), bins =
                   80) +
  geom_line(aes(
    x = d$x,
    y = dgamma(d$x, fit.params$estimate["shape"], fit.params$estimate["rate"])
  ), color = "red", size = 1)
```

Building the model

```
Y = "rate"
X = c(
  "YearBuilt"  ,
  "BLDGCLASS"  ,
  "FloorLevel"  ,
  "gshap",
  "StructureType",
  "OccupancyType",
  "RoofType",
  "NumStories"
)  # Variables explicatives

glm <- h2o.glm(
  y = Y,
  x = X,
  training_frame = as.h2o(datTR),
  keep_cross_validation_predictions = TRUE,
  family = "gamma" #,link='log'
)
print(glm)
glm@model$coefficients # Coeff pour chaque variable

rmse(as.vector(predict(glm, as.h2o(datTR))) * dat[TR,]$InsuredValue, dat[TR,]$PP)
rmse(as.vector(predict(glm, as.h2o(datTS))) * dat[TS,]$InsuredValue, dat[TS,]$PP)
```

Plotting results

```r
e1tr = NULL
e1tr$pred = as.vector(predict(glm, as.h2o(datTR)) *  dat[TR,]$InsuredValue
e1tr$pp = dat[TR,]$PP
e1tr$err = e1tr$pp - e1tr$pred
e1tr = as.data.frame(e1tr)

e1ts = NULL
e1ts$pred = as.vector(predict(glm, as.h2o(datTS)) *  dat[TS,]$InsuredValue
e1ts$pp = dat[TS,]$PP
e1ts$err = e1ts$pp - e1ts$pred
e1ts = as.data.frame(e1ts)

ggplot(e1tr[which(e1tr$pp < 1e5),], aes(x = pp, y = err)) + geom_point() + geom_abli
ne(slope = 0, intercept = 0 ,lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20)
,fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "red
",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)
ggplot(e1ts[which(e1ts$pp < 1e5), ], aes(x = pp, y = err)) + geom_point() + geom_abl
ine(slope = 0, intercept = 0, lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20
),fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "re
d",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)
```

Lorenz curve

```r
preddts = as.vector(predict(glm, as.h2o(datTS))) * dat[TS,]$InsuredValue
predots = as.vector(predict(glm, as.h2o(datTS))) * dat[TS,]$InsuredValue

premium = as.matrix(NA)
cost = as.vector(dat[TS,]$PP)
premium = cbind(preddts, cost)
colnames(premium) = c("GLM ", "PM   ")
LorenzCurve(cost, premium)
```

5.4 Gradient Boosting Model

First model

```r
# 5.1 First model
  gbm <- gbm(
    rate ~ .,
    data = datTR,
    distribution = "gaussian",
    cv.folds = 2,
    n.trees = 300,
    shrinkage = 0.6
  )
  rmse(predict(gbm, datTR, n.trees = 300) * dat[TR,]$InsuredValue, dat[TR,]$PP)
  rmse(predict(gbm, datTS, n.trees = 300) * dat[TS,]$InsuredValue, dat[TS,]$PP)
```

Second model with the optimal number of trees

```r
gbm.perf(gbm, method = 'cv')
a = gbm.perf(gbm)
gbm2 <- gbm(
  rate ~ .,
  data = datTR, distribution = "gaussian", cv.folds = 2, n.trees = a, shrinkage = 0.6
)
rmse(predict(gbm2, datTR, n.trees = a) * dat[TR,]$InsuredValue, dat[TR,]$PP)
rmse(predict(gbm2, datTS, n.trees = a) * dat[TS,]$InsuredValue, dat[TS,]$PP)

summary(gbm, plotit = TRUE)
par(mfrow = c(2, 4))
for (i in 1:8) {
  plot(gbm2, i.var = i)
}
```

Hyper-Parametrization using caret

```r
caretGrid <-
  expand.grid(
    interaction.depth = c(1, 3, 5),
    n.trees = (0:50) * 50,
    shrinkage = c(0.2, 0.1, 0.01, 0.001),
    n.minobsinnode = 10
  )
metric <- "RMSE"
trainControl <- trainControl(method = "cv", number = 2)

gbm.caret <-
  train(
    rate ~ .,
    data = dat[TR,-c(1:3)],
    distribution = "gaussian",
    method = "gbm",
    trControl = trainControl,
    verbose = FALSE,
    tuneGrid = caretGrid,
    metric = metric,
    bag.fraction = 0.75
  )

print(gbm.caret)
# "The final values used for the model were n.trees = 750, interaction.depth = 3, sh
rinkage = 0.1 and n.minobsinnode = 10."
```

Building the final GBM model

```r
gbm.final <-
  gbm(
    rate ~ .,
    data = datTR,
    distribution = "gaussian",
    n.trees = 750,
    interaction.depth = 3,
    shrinkage = 0.1 ,
    n.minobsinnode = 10
  )
rmse(predict(gbm.final, datTR, n.trees = 750) * dat[TR, ]$InsuredValue,
     dat[TR, ]$PP)
rmse(predict(gbm.final, datTS, n.trees = 750) * dat[TS, ]$InsuredValue,
     dat[TS, ]$PP)
```

Plotting results

```r
e1tr = NULL
e1tr$pred = predict(gbm.final, n.trees = a, datTR) *  dat[TR, ]$InsuredValue
e1tr$pp = dat[TR, ]$PP
e1tr$err = e1tr$pp - e1tr$pred
e1tr$Model = "Optimized"
e1tr = as.data.frame(e1tr)

e2tr = NULL
e2tr$pred = predict(gbm2, n.trees = a, datTR) *  dat[TR, ]$InsuredValue
e2tr$pp = dat[TR, ]$PP
e2tr$err = e2tr$pp - e2tr$pred
e2tr$Model = "Default"
e2tr = as.data.frame(e2tr)
etr = as.data.frame(rbind(e1tr, e2tr))
```

```
e1ts = NULL
e1ts$pred = predict(gbm.final, n.trees = a, datTS) *  dat[TS, ]$InsuredValue
e1ts$pp = dat[TS, ]$PP
e1ts$err = e1ts$pp - e1ts$pred
e1ts$Model = "Optimized"
e1ts = as.data.frame(e1ts)

e2ts = NULL
e2ts$pred = predict(gbm2, n.trees = a, datTS) *  dat[TS, ]$InsuredValue
e2ts$pp = dat[TS, ]$PP
e2ts$err = e2ts$pp - e2ts$pred
e2ts$Model = "Default"
e2ts = as.data.frame(e2ts)
ets = as.data.frame(rbind(e1ts, e2ts))

ggplot(etr[which(etr$pp < 1e5), ], aes(x = pp, y = abs(err))) +  geom_point(aes(colo
ur = Model, shape = Model)) + geom_smooth(aes(colour = Model), method = lm, se=FALSE ,
fullrange = TRUE) + geom_abline(slope = 0, intercept = 0,lwd =.1)
ggplot(ets[which(ets$pp < 1e5), ], aes(x = pp, y = abs(err))) +  geom_point(aes(colo
ur = Model, shape = Model)) + geom_smooth(aes(colour=Model),method=lm, se=FALSE, ful
lrange=TRUE )+ geom_abline(slope=0, intercept=0,lwd=.1)

ggplot(e2tr[which(e2tr$pp < 1e5), ], aes(x = pp, y = err)) + geom_point() + geom_abl
ine(slope = 0, intercept = 0, lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-
pp*0.20 ),fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill
= "re d",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill =
"red",alpha=0.1) ggplot(e2ts[which(e2ts$pp < 1e5), ], aes(x = pp, y = err)) +
geom_point() + geom_abl ine(slope = 0, intercept = 0, lwd = .1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20 ),fill = "green",alpha=0.1)+
geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "re d",alpha=0.1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)

ggplot(e1tr[which(e1tr$pp < 1e5), ], aes(x = pp, y = err)) + geom_point() + geom_abl
ine(slope = 0, intercept = 0 ,lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-
pp*0.20 ),fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill
= "re d",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill =
"red",alpha=0.1) ggplot(e1ts[which(e1ts$pp < 1e5), ], aes(x = pp, y = err)) +
geom_point() + geom_abl ine(slope = 0, intercept = 0, lwd = .1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20 ),fill = "green",alpha=0.1)+
geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "re d",alpha=0.1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)
```

Lorenz curve

```
preddtr1 = predict(gbm2, n.trees = a, dat[TR,]) * dat[TR, ]$InsuredValue
preddtr2 = predict(gbm, n.trees = 300, dat[TR,]) * dat[TR, ]$InsuredValue
predotr =  predict(gbm.final, n.trees = 750, dat[TR,]) * dat[TR, ]$InsuredValue

preddts1 = predict(gbm2, dat[TS,], n.trees = a) * dat[TS, ]$InsuredValue
preddts2 = predict(gbm, dat[TS,], n.trees = 300) * dat[TS, ]$InsuredValue
predots =  predict(gbm.final, dat[TS,], n.trees = 750) * dat[TS, ]$InsuredValue

premium = as.matrix(NA)
cost = as.vector(dat[TS, ]$PP)
premium = cbind(preddts1, preddts2, cost)
colnames(premium) = c("GBM1", "GBM2", "PM      ")
LorenzCurve(cost, premium)

premium = as.matrix(NA)
cost = as.vector(dat[TS, ]$PP)
premium = cbind(preddts1, preddts2, predots, cost)
colnames(premium) = c("Deault1", "Default2", "Optim    ", "Perfect")
LorenzCurve(cost, premium)
```

5.5 Extreme Gradient Boosting

Data treatment for XGBoost entry

```r
pacman::p_load(xgboost)

data = data.matrix(datTR[, -dim(datTR)[2]])
label = datTR$rate * 1e6
xgmat = xgb.DMatrix(data, label = label)
```

First model

```r
xgb <- xgboost(
  data = xgmat,
  nrounds = 200
)
rmse((predict(xgb, data.matrix(datTR)) / 1e6) *  dat[TR,]$InsuredValue, dat[TR,]$PP)
rmse((predict(xgb, data.matrix(datTS)) / 1e6) * dat[TS,]$InsuredValue, dat[TS,]$PP)

importance_matrix <- xgb.importance(colnames(data), model = xgb)
xgb.ggplot.importance(importance_matrix,rel_to_first = TRUE, xlab = "Relative import
ance")
xgb.ggplot.deepness(xgb,which="med.weight")
xgb.plot.multi.trees(xgb,feature_names = colnames(data), features_keep = 2)

xgbtest <- xgboost(
  data = xgmat,
  nrounds = 1,
  max_depth=2
)
xgb.plot.multi.trees(xgbtest,feature_names = colnames(data))
xgb.plot.tree(feature_names =colnames(data), model = xgbtest)
```

Hyper-parametrization

```r
nrounds = (seq(0, 500, length.out = 50))
nrounds[1] = 1
max_depth = floor(seq(0, 60, by = 10))
max_depth[1] = 1
eta = 0.1

# others parameters we could vary
# eta = seq(0.1, 0.9, length.out = 9)
# gamma = floor(seq(0, 100, length.out = 10))
# min_child_weight = floor(seq(0, 100, length.out = 10))
# max_delta_step = floor(seq(0, 100, length.out = 10))
# subsample = round(seq(0, 1, length.out = 10), digits = 2)
# colsample_bytree = round(seq(0.2, 1, length.out = 10), digits = 2)
# colsample_bylevel = round(seq(0.2, 1, length.out = 10), digits = 2)

xgberr = NULL
i = 1
for (a in nrounds) {
    for (c in max_depth) {
      xgb.fit = xgboost(
        data = xgmat,
        nrounds = a,
        eta = eta,
        max_depth = c
      )
    xgberr$nrounds[i] = a
    xgberr$max.depth[i] = c
    xgberr$errapp[i] = rmse((predict(xgb.fit, data.matrix(datTR)) / 1e6) *  dat[TR,]
$InsuredValue, dat[TR,]$PP))
```

```
    xgberr$errtest[i] = rmse((predict(xgb.fit, data.matrix(datTS)) / 1e6) * dat[TS,]
$InsuredValue, dat[TS,]$PP)
    i = i + 1
  }
}
xgberr = as.data.frame(xgberr)
xgberr$sumerr = xgberr$errapp + xgberr$errtest
xgberr[which.min(xgberr$errapp), ]
xgberr[which.min(xgberr$errtest), ]
xgberr[which.min(xgberr$sumerr), ]
save(xgberr, file = 'xgberr.RData')

m = xgberr[which.min(xgberr$errtest),]$max.depth
n = xgberr[which.min(xgberr$errtest),]$nrounds
  # n = 80 and m = 7 eta =0.1
```

Final model

```
xgb.final <- xgboost(
  data = xgmat,
  nrounds = 80,
  eta=0.1,
  max_depth=7
)
rmse((predict(xgb.final, data.matrix(datTR)) / 1e6) *  dat[TR,]$InsuredValue, dat[TR
,]$PP)
rmse((predict(xgb.final, data.matrix(datTS)) / 1e6) * dat[TS,]$InsuredValue, dat[TS,
]$PP)

importance_matrix <- xgb.final.importance(colnames(data), model = xgb)
xgb.ggplot.importance(importance_matrix,rel_to_first = TRUE, xlab = "Relative import
ance")
xgb.ggplot.deepness(xgb)
xgb.plot.multi.trees(xgb,feature_names = colnames(data), features_keep = 5)
```

Plotting results

```
e1tr = NULL
e1tr$pred = (predict(xgb.final, data.matrix(datTR))/ 1e6) *  dat[TR, ]$InsuredValue
e1tr$pp = dat[TR, ]$PP
e1tr$err = e1tr$pp-e1tr$pred
e1tr$Model = "Optimized"
e1tr=as.data.frame(e1tr)

e2tr = NULL
e2tr$pred = (predict(xgb, data.matrix(datTR))/ 1e6) *  dat[TR, ]$InsuredValue
e2tr$pp = dat[TR, ]$PP
e2tr$err = e2tr$pp-e2tr$pred
e2tr$Model = "Default"
e2tr=as.data.frame(e2tr)
etr=as.data.frame(rbind(e1tr,e2tr))

e1ts = NULL
e1ts$pred = (predict(xgb.final, data.matrix(datTS))/ 1e6) *  dat[TS, ]$InsuredValue
e1ts$pp = dat[TS, ]$PP
e1ts$err = e1ts$pp-e1ts$pred
e1ts$Model = "Optimized"
e1ts=as.data.frame(e1ts)
```

```
e2ts = NULL
e2ts$pred = (predict(xgb, data.matrix(datTS)) / 1e6) *  dat[TS, ]$InsuredValue e2ts$pp
= dat[TS, ]$PP
e2ts$err = e2ts$pp-e2ts$pred
e2ts$Model = "Default"
e2ts=as.data.frame(e2ts)
ets=as.data.frame(rbind(e1ts,e2ts))

ggplot(etr[which(etr$pp < 1e5), ], aes(x = pp, y = abs(err))) +  geom_point(aes(colo
ur = Model, shape = Model)) + geom_smooth(aes(colour = Model), method = lm, se=FALSE ,
fullrange = TRUE) + geom_abline(slope = 0, intercept = 0,lwd =.1)
ggplot(ets[which(ets$pp < 1e5), ], aes(x = pp, y = abs(err))) +  geom_point(aes(colo
ur = Model, shape = Model)) + geom_smooth(aes(colour=Model),method=lm, se=FALSE, ful
lrange=TRUE )+ geom_abline(slope=0, intercept=0,lwd=.1)

ggplot(e2tr[which(e2tr$pp < 1e5), ], aes(x = pp, y = err)) + geom_point() + geom_abl
ine(slope = 0, intercept = 0, lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-
pp*0.20 ),fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill
= "re d",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill =
"red",alpha=0.1) ggplot(e2ts[which(e2ts$pp < 1e5), ], aes(x = pp, y = err)) +
geom_point() + geom_abl ine(slope = 0, intercept = 0, lwd = .1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20 ),fill = "green",alpha=0.1)+
geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "re d",alpha=0.1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)

ggplot(e1tr[which(e1tr$pp < 1e5), ], aes(x = pp, y = err)) + geom_point() + geom_abl
ine(slope = 0, intercept = 0 ,lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-
pp*0.20 ),fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill
= "re d",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill =
"red",alpha=0.1) ggplot(e1ts[which(e1ts$pp < 1e5), ], aes(x = pp, y = err)) +
geom_point() + geom_abl ine(slope = 0, intercept = 0, lwd = .1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20 ),fill = "green",alpha=0.1)+
geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "re d",alpha=0.1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)
```

Lorenz curve

```
preddtr = predict(xgb, data.matrix(dat[TR,])/1e6) * dat[TR, ]$InsuredValue
predotr = predict(xgb.final, data.matrix(dat[TR,])/1e6) * dat[TR, ]$InsuredValue

preddts = predict(xgb, data.matrix(dat[TS,])/1e6) * dat[TS, ]$InsuredValue
predots = predict(xgb.final, data.matrix(dat[TS,])/1e6) * dat[TS, ]$InsuredValue

premium=as.matrix(NA)
cost = as.vector(dat[TS, ]$PP)
premium = cbind(preddts, cost)
colnames(premium) = c( "XGB ", "PM  ")
LorenzCurve(cost, premium)

premium=as.matrix(NA)
cost = as.vector(dat[TS, ]$PP)
premium = cbind(preddts,predots, cost)
colnames(premium) = c("Deault ", "Optim  ", "Perfect")
LorenzCurve(cost, premium)
```

5.6 Support Vector Machine

First model

```
fit.svm = svm(rate ~ . ,
              data = datTR,
              kernel = 'radial')
rmse(predict(fit.svm, datTR) * dat[TR,]$InsuredValue, dat[TR,]$PP)
rmse(predict(fit.svm, datTS) * dat[TS,]$InsuredValue, dat[TS,]$PP)
```

Hyper-parametrization using caret

```
tuneResult <-
  tune(svm,
       rate ~ . ,
       data = datTR,
       ranges = list(epsilon = seq(0, 1, 0.1),
                     cost = 2 ^
                       (2:9)))
print(tuneResult) # Meilleur svm
plot(tuneResult, theta = 1) # Grille de r?sultat des simulations faites pour la rech
erche du meilleur svm

#epsilon = 0.5, cost = 16
```

Building final model

```
svm.final = svm(
  rate ~ . ,
  data = datTR,
  epsilon = tuneResult$best.parameters[1],
  cost = tuneResult$best.parameters[2]
)
rmse(predict(svm.final, datTR) * dat[TR,]$InsuredValue, dat[TR,]$PP)
rmse(predict(svm.final, datTS) * dat[TS,]$InsuredValue, dat[TS,]$PP)
```

Plotting results

```
e1tr = NULL
e1tr$pred = predict(svm.final, datTR) *  dat[TR,]$InsuredValue
e1tr$pp = dat[TR,]$PP
e1tr$err = e1tr$pp - e1tr$pred
e1tr$Model = "Optimized"
e1tr = as.data.frame(e1tr)

e2tr = NULL
e2tr$pred = predict(fit.svm, datTR) *  dat[TR,]$InsuredValue
e2tr$pp = dat[TR,]$PP
e2tr$err = e2tr$pp - e2tr$pred
e2tr$Model = "Default"
e2tr = as.data.frame(e2tr)
etr = as.data.frame(rbind(e1tr, e2tr))

e1ts = NULL
e1ts$pred = predict(svm.final, datTS) *  dat[TS,]$InsuredValue
e1ts$pp = dat[TS,]$PP
e1ts$err = e1ts$pp - e1ts$pred
e1ts$Model = "Optimized"
e1ts = as.data.frame(e1ts)
```

```
e2ts = NULL
e2ts$pred = predict(fit.svm, datTS) *  dat[TS,]$InsuredValue
e2ts$pp = dat[TS,]$PP
e2ts$err = e2ts$pp - e2ts$pred
e2ts$Model = "Default"
e2ts = as.data.frame(e2ts)
ets = as.data.frame(rbind(e1ts, e2ts))

ggplot(etr[which(etr$pp < 1e5), ], aes(x = pp, y = abs(err))) +  geom_point(aes(colo
ur = Model, shape = Model)) + geom_smooth(aes(colour = Model), method = lm, se=FALSE ,
fullrange = TRUE)
ggplot(ets[which(ets$pp < 1e5), ], aes(x = pp, y = abs(err))) +  geom_point(aes(colo
ur = Model, shape = Model)) + geom_smooth(aes(colour=Model),method=lm, se=FALSE, ful
lrange=TRUE )

ggplot(e2tr[which(e2tr$pp < 1e5), ], aes(x = pp, y = err)) + geom_point() + geom_abl
ine(slope = 0, intercept = 0, lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-
pp*0.20 ),fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill
= "re d",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill =
"red",alpha=0.1) ggplot(e2ts[which(e2ts$pp < 1e5), ], aes(x = pp, y = err)) +
geom_point() + geom_abl ine(slope = 0, intercept = 0, lwd = .1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20 ),fill = "green",alpha=0.1)+
geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "re d",alpha=0.1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)

ggplot(e1tr[which(e1tr$pp < 1e5), ], aes(x = pp, y = err)) + geom_point() + geom_abl
ine(slope = 0, intercept = 0 ,lwd = .1) + geom_ribbon(aes(ymin=pp*0.20,ymax=-
pp*0.20 ),fill = "green",alpha=0.1)+ geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill
= "re d",alpha=0.1) + geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill =
"red",alpha=0.1) ggplot(e1ts[which(e1ts$pp < 1e5), ], aes(x = pp, y = err)) +
geom_point() + geom_abl ine(slope = 0, intercept = 0, lwd = .1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=-pp*0.20 ),fill = "green",alpha=0.1)+
geom_ribbon(aes(ymin=-pp*0.20,ymax=-pp*0.50),fill = "re d",alpha=0.1) +
geom_ribbon(aes(ymin=pp*0.20,ymax=pp*0.50),fill = "red",alpha=0.1)
```

Lorenz curve

```
preddtr = predict(fit.svm, dat[TR,]) * dat[TR, ]$InsuredValue
predotr =  predict(svm.final, dat[TR,]) * dat[TR, ]$InsuredValue

preddts = predict(fit.svm, dat[TS,]) * dat[TS, ]$InsuredValue
predots =  predict(svm.final, dat[TS,]) * dat[TS, ]$InsuredValue

premium = as.matrix(NA)
cost = as.vector(dat[TS, ]$PP)
premium = cbind(preddts, cost)
colnames(premium) = c("SVM ", "PM")
LorenzCurve(cost, premium)

premium = as.matrix(NA)
cost = as.vector(dat[TS, ]$PP)
premium = cbind(preddts, predots, cost)
colnames(premium) = c("Deault ", "Optim  ", "Perfect")
LorenzCurve(cost, premium)
```

5.7 Overall Lorenz curve

```
predglmTest = as.vector(h2o.predict(glm, as.h2o(dat[TS, ]))) * dat[TS,]$InsuredValue
predgbmTest = predict(gbm.final, datTS, n.trees = 750) * dat[TS,]$InsuredValue
predxgbTest = (predict(xgb, data.matrix(datTS)) / 1e6) *  dat[TS, ]$InsuredValue
predrfTest = predict(rf, datTS) * dat[TS, ]$InsuredValue
predsvmTest =  predict(svm.final, datTS) * dat[TS, ]$InsuredValue

premium = as.matrix(NA)
cost = as.vector(dat[TS,]$PP)
premium = cbind(predxgbTest, predgbmTest, predglmTest, predsvmTest, predrfTest, cost)
colnames(premium) = c("XGB", "GBM", "GLM", "SVM", "RF    ", "MP   ")
LorenzCurve(cost, premium)
```

6 Shiny app

6.1 User interface

```
library(shiny)
library(shinythemes)
library(shinydashboard)

dashboardPage(
  dashboardHeader(title = "Reinsurance modeling", titleWidth = 240),
  dashboardSidebar(
    width = 240,
    sidebarMenu(
      id = "tabs",
      menuItem("Table", tabName = "table", icon = icon("table")),
      menuItem(
        "Choice of a treaty",
        tabName = "plot",
        icon = icon("dollar")
      ),
      menuItem("Code", tabName = "code", icon = icon("file-code-o")),
      menuItem(
        "About",
        tabName = "about",
        icon = icon("question"),
        selected = TRUE
      )
    )
  ),
  dashboardBody(
    tags$style(
      "
      body {
      -moz-transform: scale(0.94, 0.94); /* Moz-browsers */
      zoom: 0.94; /* Other non-webkit browsers */
      zoom: 94%; /* Webkit browsers */
      }
      "
    ),
    theme = shinytheme("cosmo"),
    tags$head(tags$style(
      HTML('.content{
           background-color: #ffffff; }'))),
    tabItems(
      tabItem(tabName = "plot",

              fluidRow(
                column(
                  4,
                  wellPanel(
          h4(strong("Parameters")),

          sliderInput(
            "year",
            "Number of years simulated :",
            min = 100,
            max = 5000,
            value = 400,
            step = 100
          ),

          checkboxInput('xs', 'Excess of Loss', value = TRUE),

          checkboxInput('sl', 'Stop-Loss', value = FALSE),

          conditionalPanel(
            condition = "input.xs == true",
            sliderInput(
              "valuexs",
              label = "XS priority and line",
              min = 0,
              max = 25e7,
              value = c(2e7, 5e7),
              step = 1e7
            )
          ),

          conditionalPanel(
            condition = "input.sl == true",
            sliderInput(
              "slline",
              "StopLoss line",
              min = 0,
              max = 25e7,
              value = 1e8,
              step = 2e7
            )
          )
        ),
        wellPanel(
          h4(strong("Results")),
          textOutput("text1"),
          textOutput("text2"),
          textOutput("text3"),
          textOutput("text4"),
          textOutput("text5"),
          textOutput("text6"),
          textOutput("text7"),
          textOutput("text8"),
          textOutput("text9")
        ),
        br(),
        br(),
        br(),
        br(),
        br(),
        br(),
        br(),
        br(),
        br(),
        br()
      ),
      column(8,
                        plotOutput("distPlot", height = 770))
               )),

       tabItem(
         tabName = "table",
         box(
           width = NULL,
           status = "primary",
           solidHeader = TRUE,
           title = "Tables",
           selectInput(
             "dataset",
             "Choose a dataset:",
             choices = c("SiteInfo", "EventLossTableBySite", "expo_sample")
           ),
           downloadButton('downloadTable', 'Download'),
           br(),
           br(),
           dataTableOutput('table')
         ),
         br(),
         br(),
         br(),
         br(),
         br(),
         br(),
         br(),
         br(),
         br(),
         br()
       ),
       tabItem(
         tabName = "code",
         includeHTML("Final.html"),
         br(),
         br(),
         br(),
         br(),
         br(),
         br(),
         br(),
         br(),
         br(),
         br(),
         br()
       ),

       tabItem(tabName = 'about',
               fluidRow(
                 column(
                   6,
                   includeMarkdown('about.Rmd')
                   ,
                   br(),
                   br(),
                   br(),
                   br(),
                   br(),
                   br(),
                   br(),
                   br(),
                   br()

                 ),
                 fluidRow(column(
                  6,
                  p("Understanding our project through word cloud using text mining
in R :"),
                  plotOutput("cloud"),
                  sidebarPanel(
                    width = 10,
                    sliderInput(
                      "freq",
                      "Minimum Frequency:",
                      min = 1,
                      max = 50,
                      value = 15
                    ),
                    sliderInput(
                      "max",
                      "Maximum Number of Words:",
                      min = 1,
                      max = 100,
                      value = 100
                       ))))))))))
```

6.2 Server

```
rm(list = ls())

library(shiny)
library(ggplot2)
library(readr)
library(RColorBrewer) #display.brewer.all()
library("tm")
library("SnowballC")
library("wordcloud")


shinyServer(function(input, output, session) {
  load("d.RData")
  YLT = NULL
  ELT <- read.csv(file = "EventLossTableBySite.csv", row.names = 1)
  EventLossTableBySite <- ELT
  SiteInfo <- read.csv(file = "SiteInfo.csv", row.names = 1)
  expo_sample <- read.csv(file = "expo_sample.csv", row.names = 1)

  output$cloud <-
    renderPlot({
      wordcloud(
        words = d$word,
        freq = d$freq,
        scale = c(4, 0.5),
        min.freq = input$freq,
        max.words = input$max,
        colors = brewer.pal(8, "Dark2")
      )
    })

  observe({
    xsline <- input$valuexs[2]
    if (input$xs == TRUE) {
      updateSliderInput(session, "slline",  min = xsline)
    }
    else{
      updateSliderInput(session, "slline",  min = 0)
    }
  })

  values <- reactiveValues()
  ELTR <- reactive({
    xs  =  NULL  ;  sl  =  NULL
    ELT <-
      read.csv(file = "EventLossTableBySite.csv", row.names = 1)
    if (input$xs == TRUE) {
      xs[1] = input$valuexs[1]
      xs[2] = input$valuexs[2]
      ELT$XS = 0
      ELT[which((ELT$Loss < xs[2]) &
                  (ELT$Loss >= xs[1])), ]$XS = ELT[which((ELT$Loss < xs[2]) &
                                                           (ELT$Loss >= xs[1])), ]$L
oss - xs[1]
      ELT[which((ELT$Loss >= xs[2]) &
                  (ELT$Loss >= xs[1])), ]$XS = xs[2] - xs[1]
      ELT$PPXS = ELT$Freq * ELT$XS
      ELT$VarXS = 0
      ELT[which(ELT$XS != 0), ]$VarXS = ELT[which(ELT$XS != 0), ]$Loss *
        (1 - ELT[which(ELT$XS != 0), ]$Freq) * ELT[which(ELT$XS != 0), ]$Freq
      ELT$PTXS = (ELT$PPXS + 0.10 * ELT$VarXS) / (1 - .10)
    }
    if (input$sl == TRUE) {
      sl = input$slline
      ELT$SL = 0
      ELT[which(ELT$Loss >= sl), ]$SL = ELT[which(ELT$Loss >= sl), ]$Loss - sl
      ELT$PPSL = ELT$Freq * ELT$SL
      ELT$VarSL = 0
      ELT[which(ELT$SL != 0), ]$VarSL = ELT[which(ELT$SL != 0), ]$Loss *
        (1 - ELT[which(ELT$SL != 0), ]$Freq) * ELT[which(ELT$SL != 0), ]$Freq ELT
      $PTSL = (ELT$PPSL + 0.10 * sqrt(ELT$VarSL)) / (1 - .10)
    }
    return(ELT)
  })

  YLTR <- reactive({
    i = input$year
    YLT = NULL
    for (j in 1:i) {
      YLT = rbind(YLT, ELT[which(ELT$Freq >= runif(length(ELT$Freq), 0, 1)), ])
    }
    YLT$Event = 1:dim(YLT)[1]
    YLT$Treaty = "Net Losses"
    return(YLT)
  })

  YLTR2 <- reactive({
    YLT <- YLTR()
    n = dim(YLT)[1]
    Y = YLT
    if (input$xs == TRUE) {
      y1 = input$valuexs[1]
      y2 = input$valuexs[2]
      if (dim(YLT[which(YLT$Loss >= y2),])[1] != 0) {
        y = YLT[which(YLT$Loss >= y2),]
        a = y$Event
        yy = y
        yy$Loss = min(y2, y$Loss) - y1
        yy$Treaty = "Ceded XS"
        y$Treaty = "Net Losses "
        y$Loss = y$Loss - y2
        YLT[which(YLT$Loss > y2),]$Loss = y1
        YLT = rbind(YLT, y, yy)
      }
      else{
        a = 0
      }
      if (dim(YLT[which((y1 <= YLT[1:n,]$Loss) &
                          (YLT[1:n,]$Loss <= y2) &
                          (match(YLT[1:n,]$Event, a, nomatch = 0) == 0)),])[1] != 0) {
        y = YLT[which((y1 <= YLT[1:n,]$Loss) &
                          (YLT[1:n,]$Loss <= y2) &
                          (match(YLT[1:n,]$Event, a, nomatch = 0) == 0)),]
        YLT[which((y1 <= YLT[1:n,]$Loss) &
                     (YLT[1:n,]$Loss <= y2) &
                     (match(YLT[1:n,]$Event, a, nomatch = 0) == 0)),]$Loss = y1
        y$Treaty = "Ceded XS"
        y$Loss = y$Loss - y1
        YLT = rbind(YLT, y)
      }
    }
    if (input$sl == TRUE) {
      y3 = input$slline
      if (dim(Y[which(Y$Loss - y3 >= 0),])[1] > 0) {
        YY = Y[which(Y$Loss - y3 >= 0),]
        YY$Loss = YY$Loss - y3
        YY$Treaty = 'Ceded Stop-Loss'
        c = YY$Event
        if (input$xs == TRUE) {
          YLT[which((match(YLT$Event, c, nomatch = 0) != 0) &
                      (YLT$Treaty == 'Net Losses ')),]$Loss = YLT[which((match(YLT$E
vent, c, nomatch =
                                                                                  0)
!= 0) &
                                                                           (YLT$Treat
y == 'Net Losses ')),]$Loss - YY$Loss
        }
        else{
          YLT[which((match(YLT$Event, c, nomatch = 0) != 0) &
                      (YLT$Treaty == 'Net Losses')),]$Loss = YLT[which((match(YLT$Ev
ent, c, nomatch =
                                                                                 0) !
= 0) &
                                                                          (YLT$Treaty
== 'Net Losses')),]$Loss - YY$Loss
        }
        YLT = rbind(YLT, YY)
      }
    }


    YLT = YLT[order(YLT$Loss),]
    YLT <-
      within(YLT, Treaty <-
               factor(
                 Treaty,
                 levels = c('Ceded Stop-Loss', 'Net Losses ', "Ceded XS", 'Net Losse
s')
               ))
    return(YLT)
  })


  output$distPlot <- renderPlot({
    y1 = -1e10
    y2 = -1e10
    y3 = -1e10
    YLT <- YLTR2()
    ELT <- ELTR()
    values$v1 = length(unique(YLT$Event))
    values$v2 = sum(YLT$Loss) / input$year
    values$v3 = sum(YLT[which(YLT$Treaty == 'Ceded XS'),]$Loss) / input$year
    values$v4 = sum(YLT[which(YLT$Treaty == 'Ceded Stop-Loss'),]$Loss) /
      input$year
    values$v5 = sum(YLT[which((YLT$Treaty == 'Net Losses') |
                                (YLT$Treaty == 'Net Losses ')),]$Loss) /
      input$year

    values$v6 = sum(ELT$PPXS)
    values$v7 = sum(ELT$PTXS)
    values$v8 = sum(ELT$PPSL)
    values$v9 = sum(ELT$PTSL)


    if (input$xs == TRUE) {
      y1 = input$valuexs[1]
      y2 = input$valuexs[2]

    }
    if (input$sl == TRUE) {
      y3 = input$slline
    }


    ggplot(YLT) +   geom_col(aes(x = Event, y = Loss, fill = Treaty), width =
                              7) +
      scale_fill_manual(
        values = c(
          'Ceded Stop-Loss' = '#7401DF',
          'Net Losses ' = '#606060',
          'Ceded XS' = '#428BCA',
          'Net Losses' = '#606060'
        ),
        limits = c('Net Losses ', 'Ceded XS', 'Ceded Stop-Loss')
      ) +
      geom_abline(intercept = y1, col = '#428BCA') + geom_abline(intercept = y2, col
= '#428BCA') + geom_abline(intercept = y3, col = '#7401DF') +
      theme(
        legend.title = element_blank(),
        axis.text.x = element_blank(),
        axis.title.x = element_blank(),
        axis.ticks.x = element_blank()
      )
  })

  output$text1 <- renderText({
    paste("Number of claims in ", input$year, " year : ", values$v1)
  })
  output$text2 <- renderText({
    paste('Loss per year: $',
          format(
            round(as.numeric(values$v2), 0),
            nsmall = 0,
            big.mark = ","
          ),
          sep = "")
  })

  observe({
    xs <- input$xs
    sl <- input$sl
    if (xs == TRUE) {
      output$text3 <- renderText({
        paste('Ceded XS per year : $',
              format(
                round(as.numeric(values$v3), 0),
                nsmall = 0,
                big.mark = ","
              ),
              sep = "")
      })
      output$text6 <- renderText({
        paste('XS Pure premium : $',
              format(
                round(as.numeric(values$v6), 0),
                nsmall = 0,
                big.mark = ","
              ),
              sep = "")
      })
      output$text7 <- renderText({
        paste('XS Techinal premium  : $',
              format(
                round(as.numeric(values$v7), 0),
                nsmall = 0,
                big.mark = ","
              ),
              sep = "")
      })

    }
    else{
      output$text3 <- renderText({

      })
      output$text6 <- renderText({

      })
      output$text7 <- renderText({

      })
    }

    if (sl == TRUE) {
      output$text4 <- renderText({
        paste('Ceded Stop-Loss per year : $',
              format(
                round(as.numeric(values$v4), 0),
                nsmall = 0,
                big.mark = ","
              ),
              sep = "")
      })
      output$text8 <- renderText({
        paste('Stop-Loss Pure premium : $',
              format(
                round(as.numeric(values$v8), 0),
                nsmall = 0,
                big.mark = ","
              ))
      })
      output$text9 <- renderText({
        paste('Stop-Loss Techinal premium  : $',
              format(
                round(as.numeric(values$v9), 0),
                nsmall = 0,
                big.mark = ","
              ),
              sep = "")
      })
    }

    else{
      output$text4 <- renderText({

      })
      output$text8 <- renderText({

      })
      output$text9 <- renderText({

      })
    }
    if ((sl == TRUE) | (xs == TRUE)) {
      output$text5 <- renderText({
        paste('Net Losses per year: $',
              format(
                round(as.numeric(values$v5), 0),
                nsmall = 0,
                big.mark = ","
              ),
              sep = "")
      })
    }
    else{
      output$text5 <- renderText({

      })
    }

    datasetInput <- reactive({
      ELT <- read.csv(file = "EventLossTableBySite.csv", row.names = 1)
      EventLossTableBySite <- ELT
      SiteInfo <- read.csv(file = "SiteInfo.csv", row.names = 1)
      expo_sample <-
        read.csv(file = "expo_sample.csv", row.names = 1)
      switch(
        input$dataset,
        "EventLossTableBySite" = EventLossTableBySite,
        "SiteInfo" = SiteInfo,
        "expo_sample" = expo_sample
      )
    })

    output$table <- renderDataTable({
      datasetInput()
    }, options = list(
      autoWidth = TRUE,
      pageLength = 14,
      lengthMenu = c(15, 50, 100, 500)
    ))

    output$downloadTable <- downloadHandler(
      filename = function() {
        paste(input$dataset, '.csv', sep = '')
      },
      content = function(file) {
        write.csv(datasetInput(), file)
      }
    )
  })
})
```