APPRENTICESHIP REPORT

Data Scientist in the innovation desk team

X

FEUGA Victor

From September 2022 to September 2023

Company tutor: MARIE Aurelien

Academic supervisor: JOURDAN Astrid

School: CY Tech, PAU

Company: BNP Paribas Personal Finance, RISK, Scoring Center, Merignac

Table of contents

Acknowledgements

I would like to thank first MARIE Aurelien, my company tutor, and TABARY Edouard, the manager of the team, who accepted me for this position at BNP Paribas Personal Finance and allowed me to do this apprenticeship. I would also like to thank all the other members of the team for helping during this year.

Then, I want to thank all the other colleagues or employees I met at the office or talked to via teams. Everyone was kind to me, and it was very pleasant to come work at the BNP offices.

Finally, I would like to thank JOURDAN Astrid, my pedagogical referent, which stayed available during this year if I ever encountered any problem.

Introduction

From September 2022 to September 2023, I did an apprenticeship as a member of the Innovation Desk team at the Scoring Center in the filial Risk of BPN Paribas Personal Finance located in Merignac. Through this year of apprenticeship, I wanted to increase my skills in Data Science, discover new techniques and learn more about the working environment of a big company.

The office is in Merignac near the airport. It's like a private campus with 2 buildings for employee's offices, a big parking lot and a cafeteria. The working environment is very comfortable with new open spaces and soundproofing rooms for meetings. As a Bordeaux native, I was very happy to work in my hometown for a bit after spending most of my student life in Pau.

The team I joined is composed of 5 data scientists (Wael, Julie, Kenza, Aurelien, Angelina) and 1 team manager (Edouard). Their main objective is to discover and introduce machine learning models for different missions related to the scoring center.

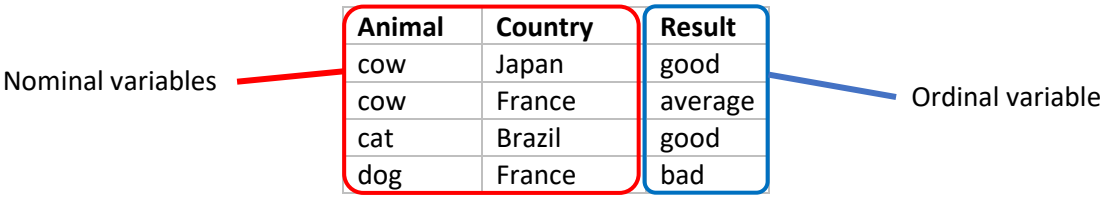
The aim of this apprenticeship was, in a first time, to assist the data scientists of the team by doing a research work about categorical encoders. And then, having a more business-oriented mission by building a scorecard using logistic regression to identify "bad debt" individuals on credit cards for a filial in South Africa.

Chapter 1: Research about categorical variables encoding

1. Introduction

The data preprocessing has a very important influence when it comes to the Machine Learning results. Most of the models can't handle categorical variables (like RandomForest models which are mainly used at the ScoringCenter), except if we change them into numerical values. The model performances will rely on how we will encode those data. Depending on the method used, the predictions can be highly impacted. And so, the precision might be increased or decreased.

There are 2 types of categorical variables. Nominal's ones, which don't have any order notion (ex: "cow, dog, cat" or "France, Japan, USA, Brazil"). And the ordinal's ones, which can be ordered (ex: "good, average, bad"). As presented in the example below:



Animal	Country	Result
cow	Japan	good
cow	France	average
cat	Brazil	good
dog	France	bad

To begin with, it's important to do an overview of the different encoding techniques.
The Kaggle kernel by Shahul E. allows to learn and try some of the most used encoding techniques (such as Label, OneHot, FeatureHashing and Target). For each encoding technique, it runs and tests in a simple logistic regression model.
The article by Baijayanta R. is a state of art of all the current techniques (the author keeps it updated by adding new encoding methods). It presents each method and provides examples on how to do it.
The research paper by John H. and Taghi K. investigates current techniques for representing qualitative data as input to neural networks. Even if the ScoringCenter doesn't work with neural networks, the document explains how work different encoding methods (Label, Count, OneHot, LeaveOneOut and Hashing).
The research paper by Shipra S. explains what's Categorical Data Encoding, presents different encoding techniques (Label, OneHot, Dummy, Effect, Binary N, BaseN, Hash and Target) and when to use them. This research paper introduces the *category_encoders* python library.
The Matt C. article, which is a step-by-step guide, presents us how to use the *category_encoders* python library in a real use case. And provides a "simple" benchmark of the different performances of each encoder on a classification model (XGBClassifier).
The TowardsDataScience article by Denis V. is a benchmark of different categorical encoder techniques present in the *category_encoders* library, using single and double validation. The methodology used in this research's inspired by his work.

There is a lot of ways to encode categorical variables into numerical values to use them in our models. In this document, we will review different of this encoding techniques. We will study what are their Pros and Cons. And will see the results on real application cases.

Our 3 keys objectives are:

- **Performance**, we want to have an encoding technique that can provide better or even results than the current solution.

- **Dimension**, the less columns our dataset obtains after encoding, the better it's for tree-based models. As the dimensionality increases the number of possible combinations of parameters for machine learning algorithms to search through explodes exponentially.

- **Interpretability**, we want to use an encoding method that's simple to understand. And allow us to easily understand how our model works. Let's take the example of feature importance. With this technique we can calculate a score for all the input features for a given model (the scores simply represent the "importance" of each feature). The encoding methods that increase the number of columns (like One Hot Encoding) will increase the number of features. And so, reduce the interpretability of the feature importance.

To make it simple, the following techniques were selected based on the interpretability constraint. And if we can easily make them by our hands (in case the library is disable).

2. Techniques

Let's make a state of art of the different encoding techniques used. For each encoder, we will look at how it works, the pros/cons and how it handles missing values.

i. Increase data frame shape

1. One Hot Encoding

The One Hot Encoding is currently the most used method to encode categorical variables. The encoder creates several additional features based on the number of unique values in the categorical feature.

For example, for the categorical variable "color" with unique values *red*, *blue* and *green* inside, the One Hot Encoder will create 3 new columns named "color_red", "color_blue" and "color_green". Those variables will take the value 1 depending on the observation color. Otherwise, it will be 0.

id	color
1	red
2	blue
3	green
4	blue

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Figure 1: One Hot Encoding

Pros:

- Simple to understand
- Powerful for non-tree models

Cons:

- Can add a lot of new columns (depending on if the encoded variable has a lot of categories)
- Low readability on analysis feature graphs

Handling missing value:

The One Hot Encoder from the library *category_encoders* will by default encode a new value as 0 in every dummy column. In the example below, *Purple* is a new value that didn't exist during the encoder fitting.

	Color	Color_Red	Color_Blue	Color_Green
0	Red	1	0	0
1	Blue	0	1	0
2	Green	0	0	1
3	Purple	0	0	0

Figure 2: One Hot Encoding missing value

Note:

For regression, it's advised to use N-1 columns after encoding in order to exclude redundancy, with N the number of categories in our variable. To do this, we must drop the first column after One Hot Encoding. It allows us to avoid the "dummy variable" trap, a scenario in which two or more variables are highly correlated. In simple terms, one variable can be predicted from the others.

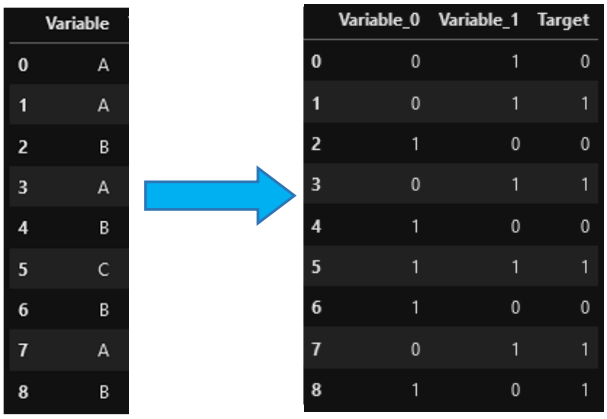
2. Binary Encoding

The Binary Encoding converts the categories from a variable into a binary code. If there is N categories in a column, the encoder will create ***log (base 2)*** ^{N} new columns.

For example, let's encode a variable with 6 different categories. By using the binary encoder, we will only have 3 new columns in our dataset. Meanwhile, with the One Hot Encoding, we would have created 6 new columns. Now, if we want to encode a variable with 120 categories, we will only create 7 new columns with Binary Encoding. Instead of 120 new columns with One Hot Encoding. Here's an example:

Level	Binary encoding	One hot encoding
No	000	000001
Primary	001	000010
Secondary	010	000100
Bsc/BA	011	001000
MSc/MA	100	010000
PhD	101	100000

Figure 3: Binary vs OHE



	Variable
0	A
1	A
2	B
3	A
4	B
5	C
6	B
7	A
8	B

	Variable_0	Variable_1	Target
0	0	1	0
1	0	1	1
2	1	0	0
3	0	1	1
4	1	0	0
5	1	1	1
6	1	0	0
7	0	1	1
8	1	0	1

Figure 4: Binary Encoding

Pros:

- Works well for variables with a high number of categories
- Save storage
- Reduce the dimensional troubles for high cardinality data

Cons:

- Don't give any information with the new columns
- Low readability on the feature graph analysis

Handling missing value:

The Binary Encoder from the library *category_encoders* will by default encode a new value as 0 in every binary column. In the example below, *Purple* is a new value that didn't exist during the encoder fitting.

	Color	Color_0	Color_1
0	Red	0	1
1	Blue	1	0
2	Green	1	1
3	Purple	0	0

Figure 5: Binary Encoding missing value

ii. Conserve the data frame shape

1. Ordinal Encoding

With Ordinal Encoding, each category will be transformed with an integer between 1 and N (N the number of different categories in our variable). If we have ordinal data, an optional mapping dictionary can be passed in argument. In this case, we use the knowledge that there is some true order to the classes themselves. For example, let's say we want to encode a column "Temperature" with 4 different categories (*Hot*, *Cold*, *Very Hot*, *Warm*). We must give a logical order to this value to encode them correctly: *Cold* < *Warm* < *Hot* < *Very Hot.* Otherwise, the classes are assumed to have no true order, and integers are selected at random.

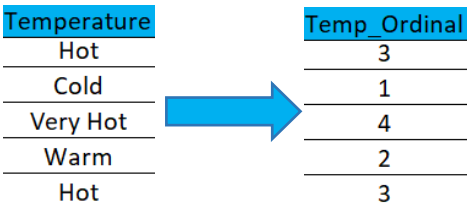


Figure 6: Ordinal Encoding

Pros:

- Easy to understand
- Low storage

Cons:

- If the variable is **ordinal**, the user must do it by hand to give the order of the categories
- Can **create a notion of** order that doesn't exist between the categories if they are **not ordinal**. The rank may result in an undesired bias and separate possible interesting combinations

Handling missing value:

The Ordinal Encoder from the library *category_encoders* will by default encode a new value by -1. In the example below, *Purple* is a new value that didn't exist during the encoder fitting.

	Color	Color_ord
0	Red	1.0
1	Blue	2.0
2	Green	3.0
3	Purple	-1.0

Figure 7: Ordinal Encoder missing value

2. Count Coding

Count encoding replaces each categorical value with the number of times it appears in the dataset.

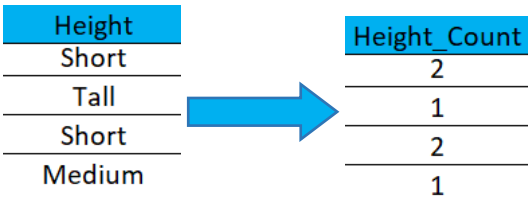


Figure 8: Count Encoding

Pros:

- Easy to understand
- Useful for trees models

Cons:

- If the train sample is different enough from the population, the model may overfit the training data because
- Risk to lose information if 2 categories have the same count

Handling missing value:

The Count Encoder from the library *category_encoders* will by default encode a new value by 0. In the example below, *Purple* is a new value that didn't exist during the encoder fitting.

	Color	Color_count
0	Red	3.0
1	Blue	4.0
2	Green	2.0
3	Purple	0.0

Figure 9: Count Encoding missing value

3. Target Encoding

The Target Encoding uses the information coming from the target variable. This process will transform the values from the categorical variable by the probability to have 1 in the target variable for each category (as explained in *Figure 10*).

As shown in the below graph, we can see we have 4 values *Hot* in the categorical variable "Temperature". 4 *Hot* on 3 have target value of 1 (column "Target "). That's why, the *Hot* values will be encoded by ¾ = 0.75

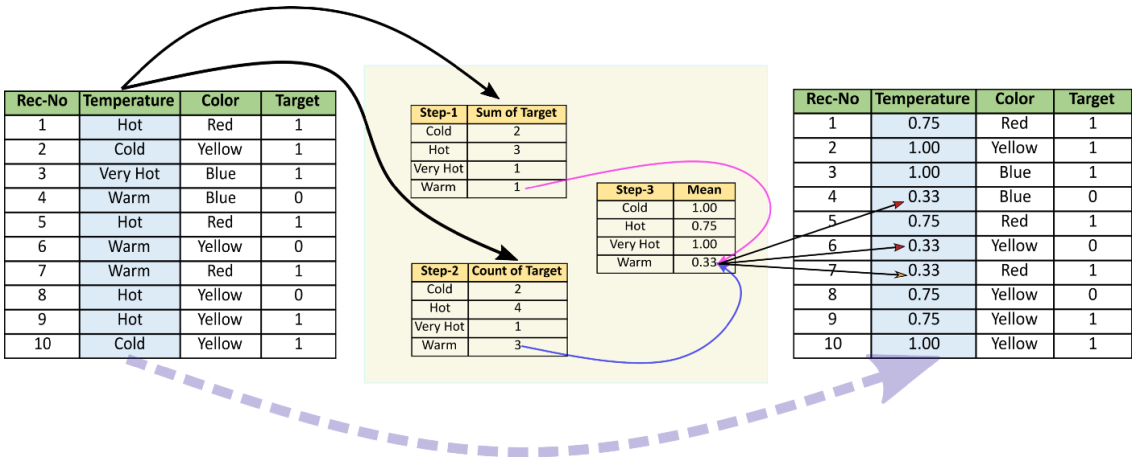


Figure 10: Target Encoding, image by Baijayanta R.

The *category_encoders* library includes a smoothing operation. This technique is particularly useful to handle situations when some of the categories are not well represented.
The example from Max H. blog might help to explain the smoothing idea: "*Imagine a new movie is posted on IMDB and it receives three ratings. Taking into account the three ratings give the movie an average of 9.5. This is surprising because most movies tend to hover around 7, and the very good ones rarely go above 8. The point is that the average can't be trusted because there are too few values. The trick is to "smooth" the average by* **including the average rating over all movies**. *In other*

words, if there aren't many ratings we should rely on the global average rating, whereas if there enough ratings then we can safely rely on the local average."

Pros:

- Simple and powerful

Cons:

- Target Leakage, encode with information of the target variable we are trying to predict
- If the train sample is different enough from the population, the model may overfit the training data because mean values will be different
- High risk when there is a drift on the encoded data when the target proportions change

Handling missing value:

The Target Encoder from the library *category_encoders* will by default encode a new value by the target mean (during the encoder fitting). In the example below, *Purple* is a new value that didn't exist during the encoder fitting.

	Color	Color_tar
0	Red	0.384822
1	Blue	0.319335
2	Green	0.286050
3	Purple	0.333333

Figure 11: Target Encoding missing value

4. Weight Of Evidence Encoding

The Weight Of Evidence Encoding measures the "strength" of a grouping technique to separate good and bad. It's calculated with the following formula:

$$WOEcat = \ln\left(\frac{\%GOODcat}{\%BADcat}\right)$$

Color	Target
Red	1
Blue	1
Red	0
Blue	1
Blue	0

Color_WOE
0
0.693
0
0.693
0.693

Figure 12: Weight of Evidence Encoding

Pros:

- Create a new subclass if 2 variables have the %Good and %Bad

Cons:

- Target Leakage, encode with information of the target variable we are trying to predict
- If the train sample is different enough from the population, the model may overfit the training data because mean values will be different
- High risk when there is a drift on the encoded data when the target proportions change

Handling missing value:

The Weight of Evidence Encoder from the library *category_encoders* will by default encode a new value by 0.00. Meaning this new value is expected to have the same percentage of good and bad. In the example below, *Purple* is a new value that didn't exist during the encoder fitting.

	Color	Color_WoE
0	Red	0.628609
1	Blue	0.117783
2	Green	-1.386294
3	Purple	0.000000

Figure 13: Weight of Evidence Encoding

3. Methodology

Now we are going to see the methodology used to compare the encoders performances on a dataset.

Firstly, we need to have prepared and cleaned our dataset. And make sure that the categorical columns we want to encode have the pandas type 'category'. We will then split our dataset into a train (80%) and a test (20%) sample. Make sure your samples are stratified on the target variable when doing the train test split.

Secondly, we are going to apply the *encoderPerfTraining* function on our train data. This function takes for arguments a list of declared encoding methods, the number of runs we want to perform, the training dataset and a sklearn binary classifier. We will then generate 5 stratified folds (split data into train/validation sets) from our train dataset. And for every fold, we will fit our encoders on the current train fold. Then, transform the train and validation data. Afterwards, we fit the classifier on the encoded train fold to predict the encoded validation fold. With the prediction, we can provide metrics (ROC_AUC, Accuracy, Recall, etc.) for each encoder on each fold. We repeat this operation until we reached the number of runs the user asked.

Thirdly, after doing all the runs, we obtain a data frame with all the metrics for each run of each encoder (using the return by *encoderPerfTraining*). And so, we obtain a boxplot of each metric with the function *encoderGraphPerf*. We can now compare which encoder performed better for our classifier on our train dataset.

Finally, we can select the best encoder on the train data based on the different performances. And apply it on the test dataset to see how it performs in a real situation. We can compare the new results to the past ones to see if the new encoding technique changed the performances.

The following schema (*Figure 14*) presents what is the main logic behind this program.

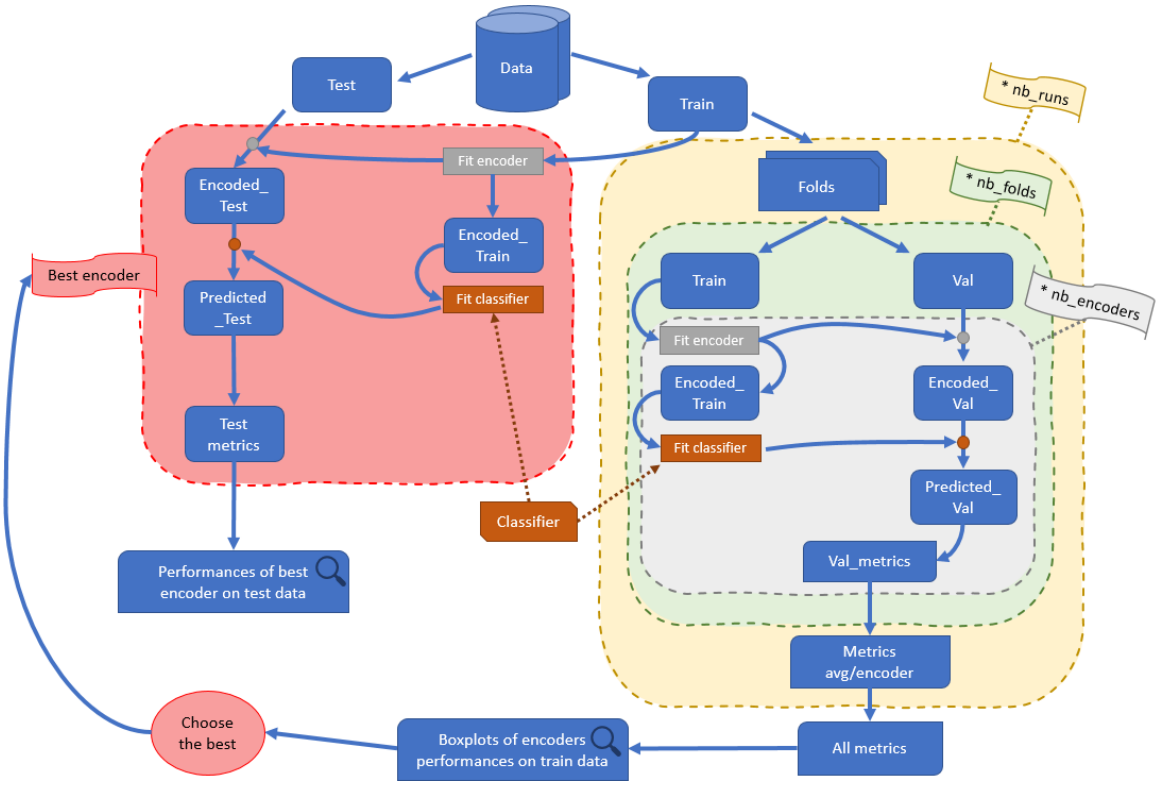


Figure 14: Program schema

" nb_runs"* is a variable given by the user.
"** nb_folds*" is by default set to 5 in the *encoderPerfTraining* function (n_folds=5).
" nb_encoders*" depends on the number of encoders the user wants to test.
The "*Classifier*" is decided and declared by the user. It must be a binary sklearn classifier (for example, a random forest).

This program doesn't give a final encoder choice to the user. But it allows him to have the different performances to make his own choice.

4. Results

Before looking at the results of this program, it is important to present the decision metrics used to evaluate the performances.

i. Metrics presentation

1. Confusion Matrix

Confusion matrices represent counts from predicted and actual values. It can be represented as the following table for a binary classification problem:

		Predicted	
		Negative (0)	Positive (1)
True	Negative (0)	True Negatives (TN)	False Positive (FP)
	Positive (1)	False Negative (FN)	True Positive (TP)

2. Accuracy score

The accuracy score is an evaluation metric that measures the number of correct predictions made by a model. It's calculated by dividing the number of correct predictions by the total number of predictions. Although it's a very easy to understand metric, it doesn't work well on imbalanced datasets. The formula to calculate it is:

$$Accuracy\ score = \frac{TP + TN}{TP + TN + FP + FN}$$

3. Precision score

The precision score is the proportion of actual positives in all the positive predictions. The formula used to calculate it is:

$$Precision\ score = \frac{TP}{TP + FP}$$

4. Recall score

The recall score is the proportion of actual positives predicted correctly. The formula used to calculate it is:

$$Recall\ score = \frac{TP}{TP + FN}$$

5. ROC AUC score

It's mandatory to define what are ROC (Receiver Operating Characteristic) curves to understand the ROC AUC score.
A ROC curve is intended to show how well the model works for every possible threshold. By confronting the True Positive Rate ($TPR = Recall\ score = \frac{TP}{TP+FN}$) to the False Positive Rate ($FPR = \frac{FP}{TN+FP}$).
On the following plots, the green line is when TPR = FPR. While the blue line represents the ROC curve of the classifier. If the ROC curve is exactly on the green line, it means that the classifier has the same predictive power as flipping a coin.

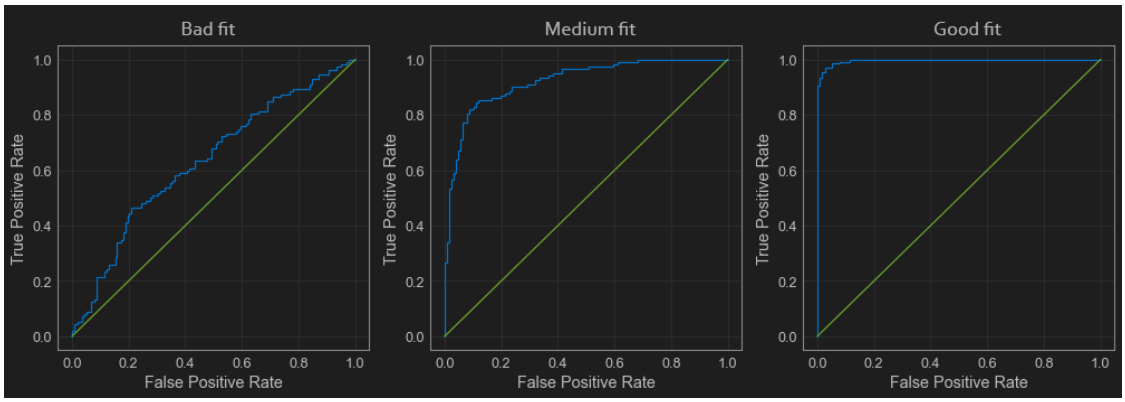


Figure 14: ROC Curves, image by Vinícius T.

As shown on the *Figure 14,* on the left plot the blue line is relatively close to the green one, which means that the classifier is bad. The middle one is a good enough classifier. The rightmost plot shows a good classifier, with the ROC curve closer to the axes and the "elbow" close to the coordinate (0, 1). To make it short, the bigger the area under the blue curve is, the better it is.

The ROC AUC (Area Under the Curve) score is a metric that evaluates a ROC curve. It takes a value between 0.5 and 1. The score measures the area under the curve of the ROC curve. A score around 0.5 means that the model is "bad". And a score near 1 means that the model is "good".

ii. Fraud score FULL-CB backup model

The FULL-CB product is an online payment facility in 3 or 4 times for French Ecommerce platforms. The purchase amounts variate between 90€ and 3000€. On this product, no supporting documents are requested from the client. He only needs to fill an online form. The process is a 100% dematerialized. For the backup model we only use granted data, the initial train dataset has around 674 000 individuals for the train set, around 168 000 individuals for the test set and around 180 000 for the out of time set. They have 280 columns each. The target variable is 'MP_ECH2', which indicates if the person made a fraud or not.

In order to gain time in the Data exploration and Modelling steps, we will use the results obtained by Julie CAVARROC presented in her project. The encoding technique selected for the categorical variables is a One Hot Encoder.

The idea is not to compete against Julie's model by optimizing the process. Instead, we want to see if, with almost the same variables and the same random forest model, we can increase the performances by using a different encoder.

For some confidential reason, the non-categorical features will not be named in this report.

Using **all** columns.

Before the features selection step, the initial project kept 54 variables (after a One Hot Encoding process).

In order to use our encoders as expected, we will reverse the One Hot Encoding process. And keep the origin variables used before the One Hot Encoding. In the end, there are 42 variables created by the One Hot Encoder. After reversing the encoding, the 42 columns became 7 categorical variables.

We are using information about sociodemographic, financial situation, …
Here is a sum up of the final columns:

- 7 numerical
- 5 binaries
- 7 categorical (BANK_CLEAN with 9 values, LEVEL_CLEAN with 4 values, BRAND_CLEAN with 3 values, TYPE_CLEAN with 3 values, ORDERDATE_MONTH with 12 values, DOMAINE_EMAIL with 5 values, SECTEUR_VENDEUR with 7 values)

Here is an extract of the final train data frame before using it:

```
X_train.head()
```

0	31.0	14000.0	LCL	552.0	1.0	1.0	CLASSIC	GPE_MEUBLE_MDM	GMAIL
1	7.0	94541.0	BPCE	415.0	1.0	0.0	GOLD	GPE_MEUBLE_MDM	GMAIL
2	21.0	212724.0	LCL	556.0	1.0	0.0	GOLD	GPE_HIGH_TECH_ELECTRO	GMAIL
3	21.0	19899.0	Others	761.0	1.0	1.0	CLASSIC	GPE_HAB_JARD_BRICO	Others
4	24.0	44013.0	CREDIT_MUTUEL	334.0	0.0	0.0	GOLD	GPE_BUT	Others

Figure 15: FULL-CB Backup Train data extract after reversing One Hot Encoding

The random forest used is the one selected on the source project. So, it's optimized on One Hot Encoded data. The main hyperparameters used are 100 trees and a depth of 16.

Finally, after processing the encoding techniques tester, here are the results after processing 15 runs at the cut-off 1%:

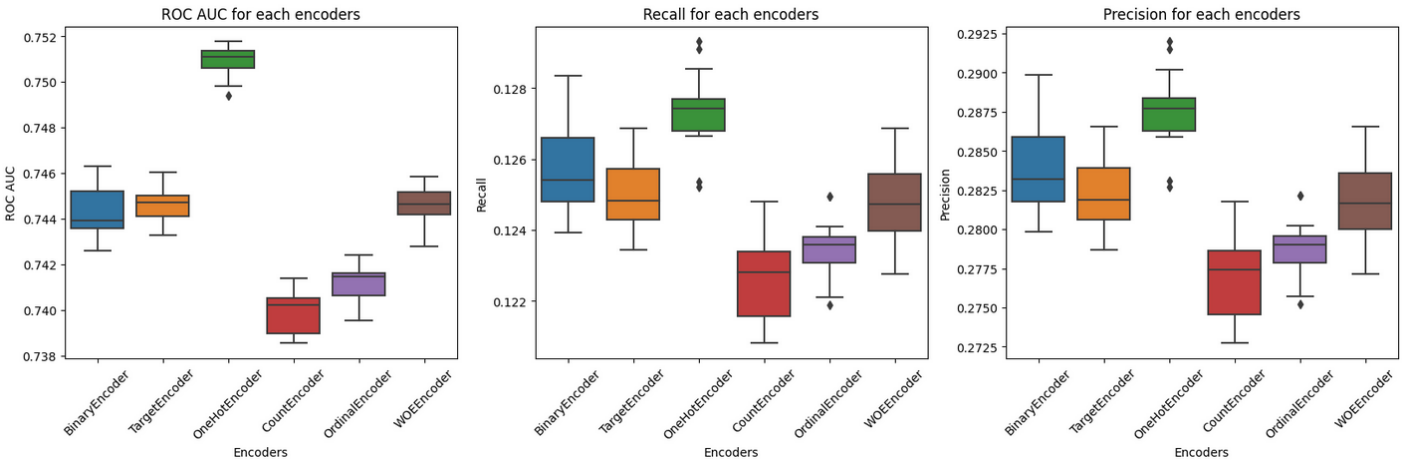


Figure 16: FULL-CB Backup encoding methods metrics

The OneHotEncoder averages better results than the others. But scale wise, the difference in the results is very low. For example, there is only a bit more than 1% of difference between the best OneHotEncoder and the worse CountEncoder on the ROC AUC metric. Although the metrics results come from a RandomForest optimized for a OneHotEncoding preprocessing, the other encoding techniques have similar/close results. So, we can conclude that the other encoding techniques have similar results while having less columns added to the data frame then with OneHotEncoding.

In OneHotEncoding, the data frame had 19 columns before encoding. And had 54 columns after encoding. While with BinaryEncoding, the data frame had 33 columns after encoding. Target, Count,

Ordinal and WeightOfEvidence encoders don't change the shape of the data frame after encoding. So, it kept its original shape with 19 columns.

NOTE: Because Target et WeightOfEvidence encoders are both based on the target, the models have similar results. There is a convergence in the results when we run a lot of iterations using this method.

The next step is to test for each encoder what are the selected columns after the features selection process (developed in the initial work). It will allow us to understand what columns are optimized for each encoding techniques.

Feature selection for **each** encoder.

The initial project has an internal function that select feature based on permutation feature importance. It also does a hyperparameters optimization for a random forest at the same time.

We will use the 19 variables that were selected to do the feature selection in the initial project. Those variables have the following type:

- 7 numerical
- 5 binary
- 7 categorical (BANK_CLEAN, LEVEL_CLEAN, BRAND_CLEAN, TYPE_CLEAN, ORDERDATE_MONTH, DOMAINE_EMAIL, SECTEUR_VENDEUR)

Because One Hot and Binary encoders create new columns, the feature selection process is longer then with the other techniques:

- Between 5 hours 30 minutes and 6 hours for One Hot and Binary
- Between 2hours 40 minutes and 3 hours 40minutes for Ordinal, Count, Target and Weight Of Evidence

We want to compare what features will be selected for each encoder on the same dataset. To do so, we are going to select the iteration that has a good performance (based on precision, recall and roc auc scores) and has reduced the number of columns (in order to reduce complexity).
Here is an overview of the different performances for every encoding technique:

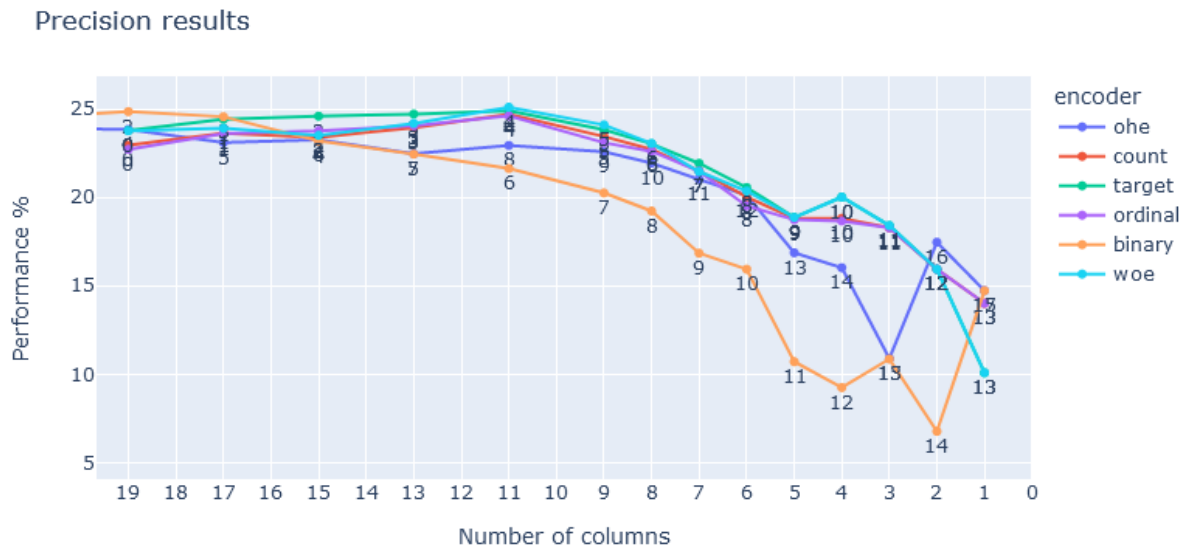


Figure 17: Features selection precision score results

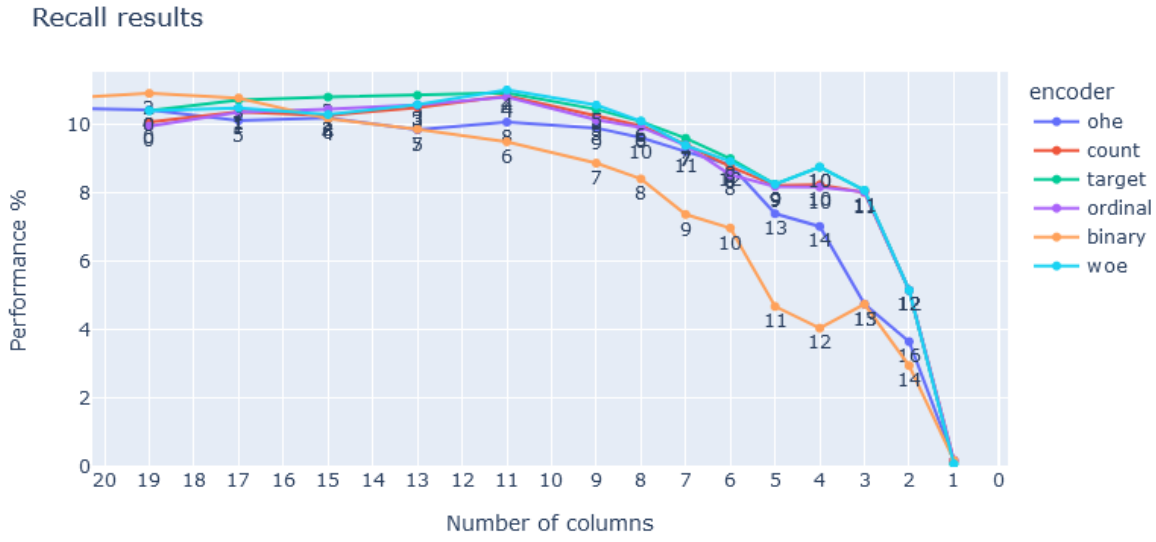


Figure 18: Features selection recall score results

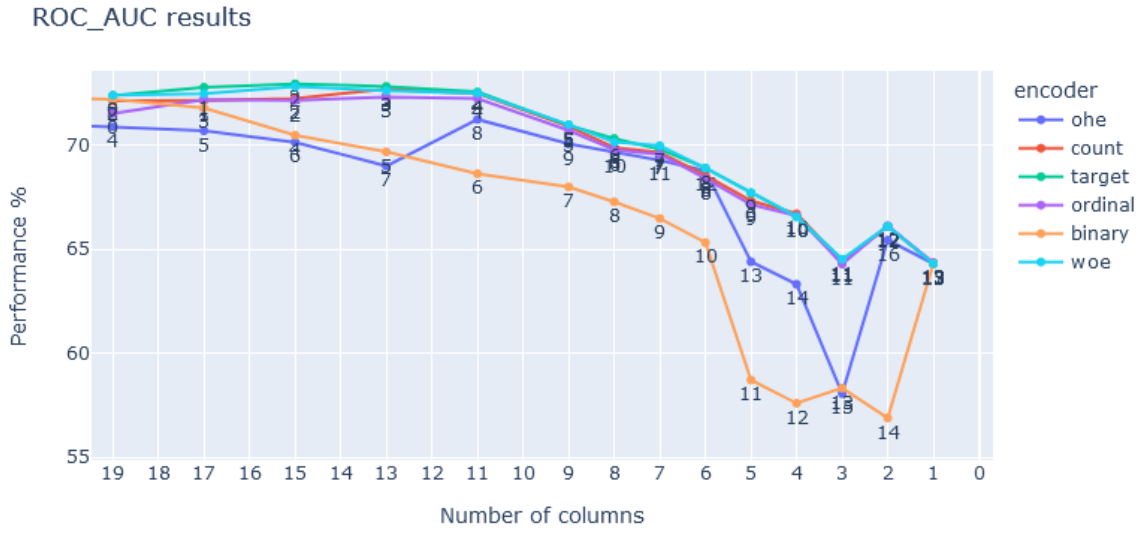


Figure 19: Features selection ROC AUC score results

The x-axis is the number of selected columns in the features selection. The y-axis is the performance of the metric in %. The numbers under each marker represent the iteration number realized in the process.

An interesting thing to observe is that encoders who don't increase the shape (Count, Target, Ordinal, WoE) of the data frame have better results for the same number of columns then the ones who increases the size (OHE, Binary). For example, on the precision (*Figure 18*) for 11 columns, there is around 2% of difference between the TargetEncoder and the OneHotEncoder in favor of the Target for the same number of selected columns.

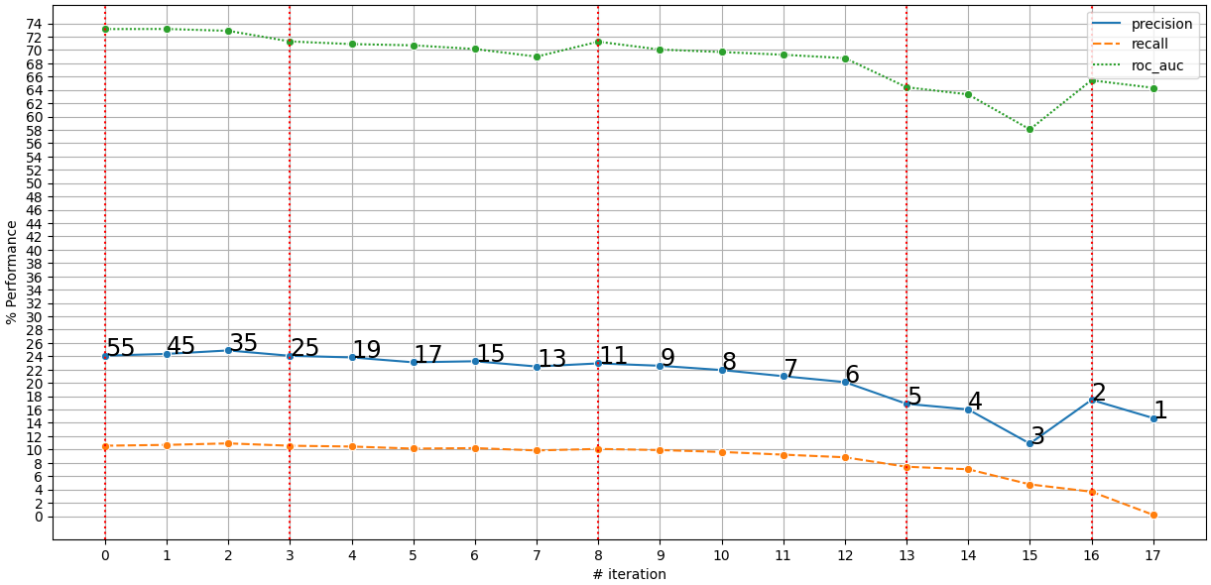
Let's now see the best option for every encoder. To do so, we must select the iteration during the feature selection which doesn't have too many columns and still has good performances on the different metrics. This step is very subjective and depends on the analyst preferences or obligations.

Without looking at the performances, we can already see a difference just with the processing time of the used feature selection function:

- OneHot and Binary: between 5 hours 30 minutes and 6 hours
- Ordinal, Count, Target and WeightOfEvidence: between 2 hours 40 minutes and 3 hours 40 minutes

Here are the performances of the feature selection process for each encoding techniques:

- BinaryEncoding



Here are the results of the feature selection for the OneHotEncoding technique:

Figure 20: OneHotEncoding features selection results

From the previous graph, we can see that the iteration number 8 has good metrics and doesn't have a lot of selected columns (only 11 instead of the 55 initial ones). Among the 11 final columns of this feature selection, the encoded categorical ones are: BANK_CLEAN_BANQUE_POSTALE, LEVEL_CLEAN_CLASSIC, DOMAINE_EMAIL_HOTMAIL, SECTEUR_VENDEUR_GPE_CONFORAMA, BANK_CLEAN_Others, BANK_CLEAN_BNP_PARIBAS.

- BinaryEncoding

Here are the results of the feature selection for the BinaryEncoding technique:

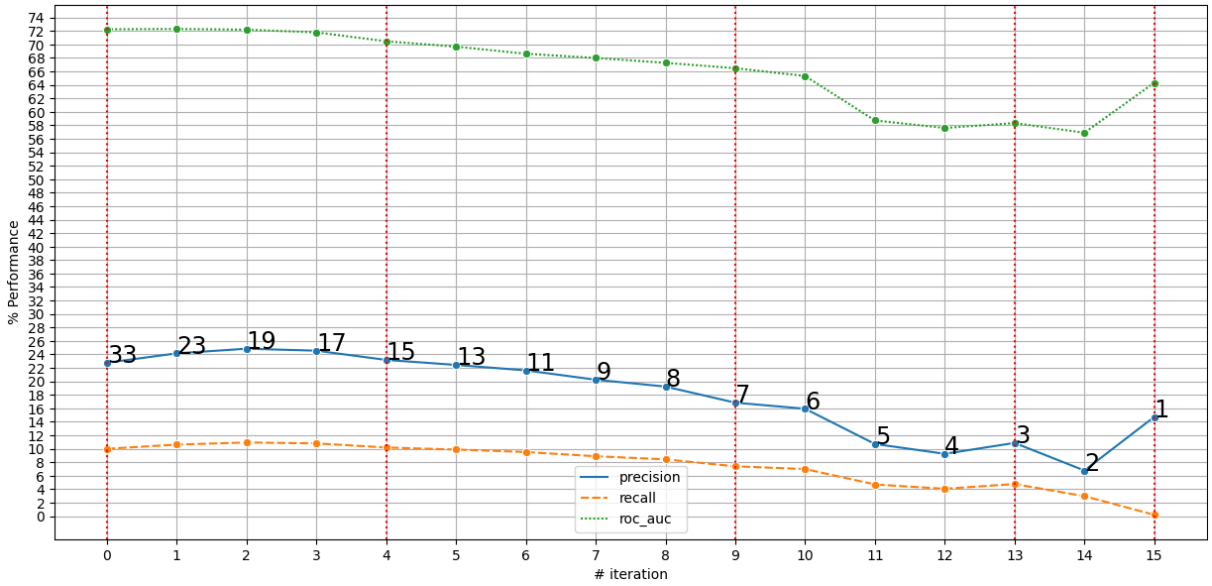


Figure 21: BinaryEncoding features selection results

From the previous graph, we can see that the iteration number 6 has good metrics and doesn't have a lot of selected columns (only 11 instead of the 33 initial ones).
Among the 11 final columns of this feature selection, the encoded categorical ones are: BANK_CLEAN_0, BANK_CLEAN_3, LEVEL_CLEAN_1, DOMAINE_EMAIL_0, LEVEL_CLEAN_2, SECTEUR_VENDEUR_2.

- OrdinalEncoding

Here are the results of the feature selection for the OrdinalEncoding technique:

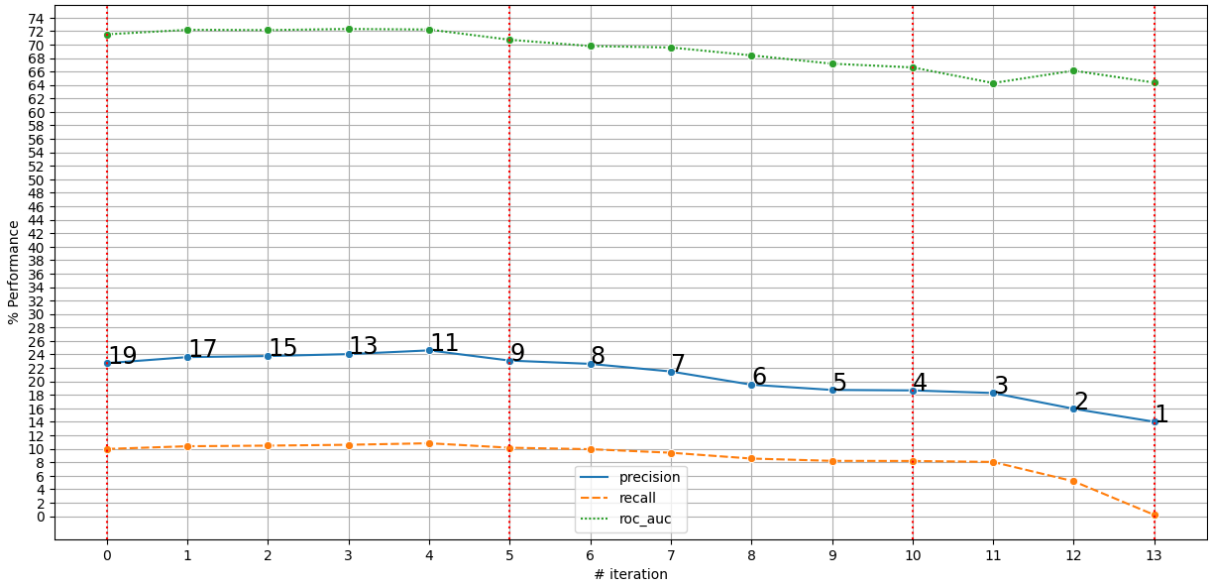


Figure 22: OrdinalEncoding features selection results

From the previous graph, we can see that the iteration number 4 has good metrics and doesn't have a lot of selected columns (only 11 instead of the 19 initial ones).

Among the 11 final columns of this feature selection, the encoded categorical ones are: BANK_CLEAN, DOMAINE_EMAIL, LEVEL_CLEAN, SECTEUR_VENDEUR, TYPE_CLEAN, BRAND_CLEAN.

- CountEncoding

Here are the results of the feature selection for the CountEncoding technique:

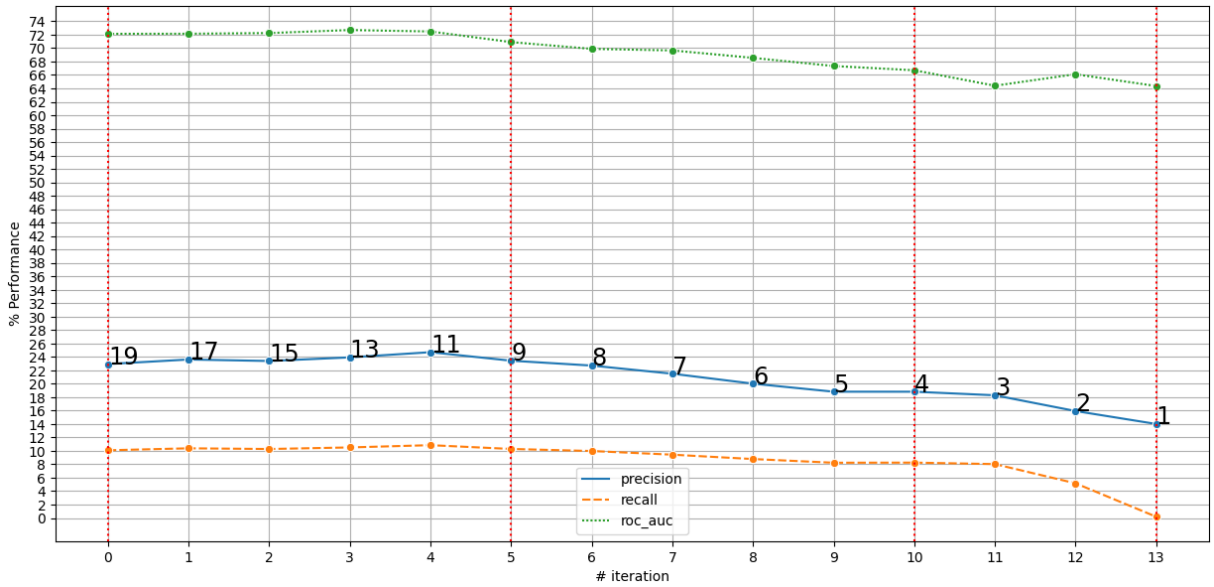


Figure 23: CountEncoding features selection results

From the previous graph, we can see that the iteration number 4 has good metrics and doesn't have a lot of selected columns (only 11 instead of the 19 initial ones).
Among the 11 final columns of this feature selection, the encoded categorical ones are: BANK_CLEAN, DOMAINE_EMAIL, LEVEL_CLEAN, SECTEUR_VENDEUR, TYPE_CLEAN.

- TargetEncoding

Here are the results of the feature selection for the TargetEncoding technique:

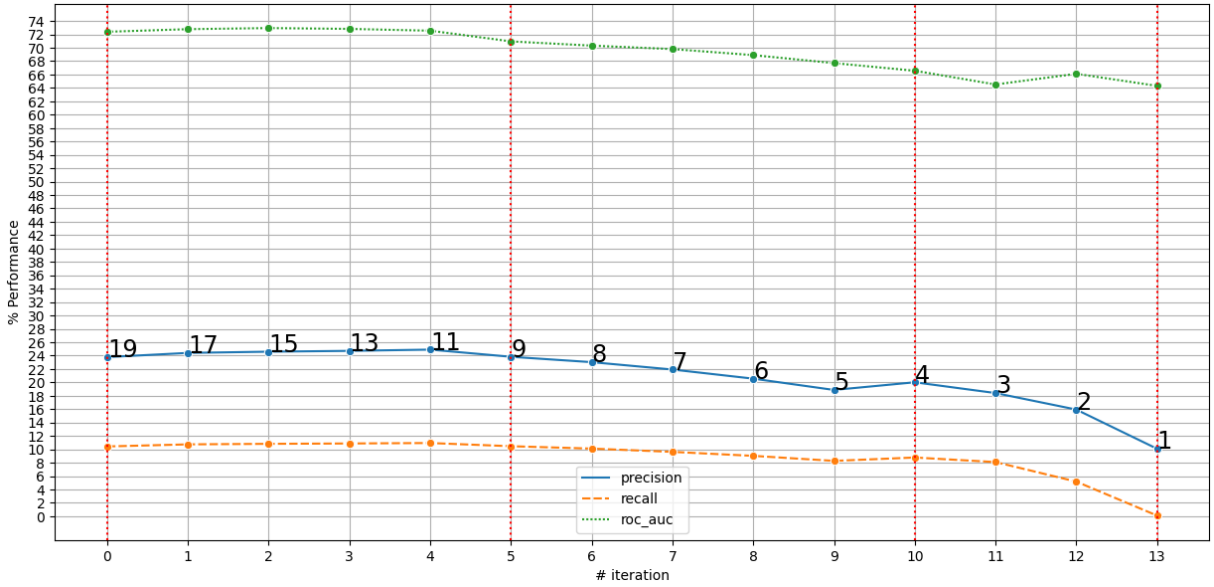


Figure 24: TargetEncoding features selection results

From the previous graph, we can see that the iteration number 4 has good metrics and doesn't have a lot of selected columns (only 11 instead of the 19 initial ones).
Among the 11 final columns of this feature selection, the encoded categorical ones are: DOMAINE_EMAIL, SECTEUR_VENDEUR, LEVEL_CLEAN, BRAND_CLEAN, TYPE_CLEAN.

- WeightOfEvidence

Here are the results of the feature selection for the WeightOfEvidence technique:

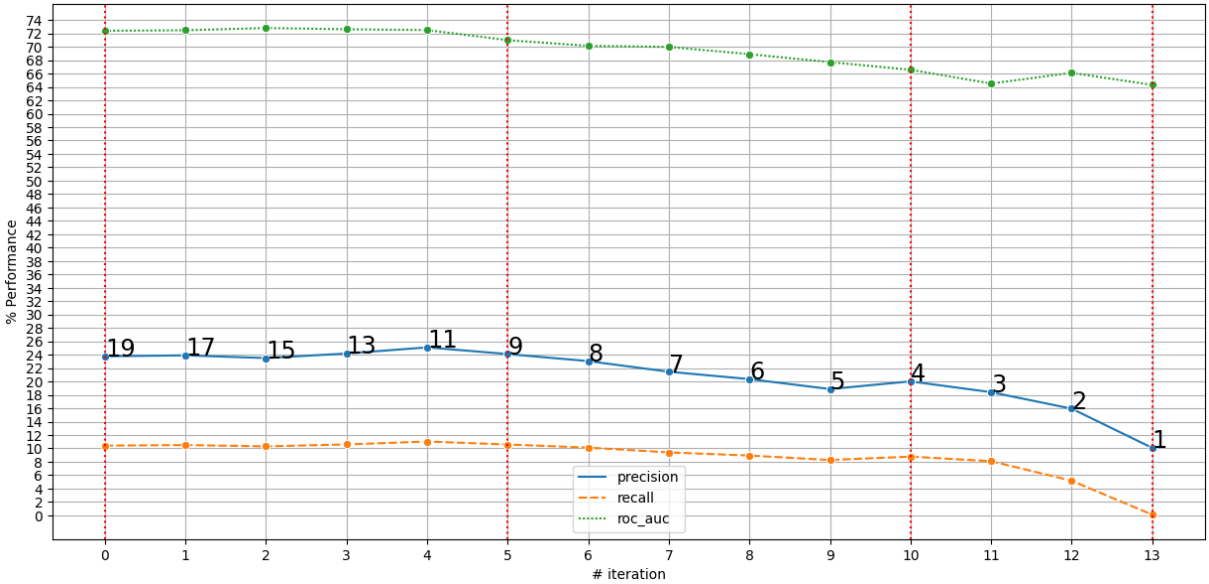


Figure 25: WeightOfEvidence features selection results

From the previous graph, we can see that the iteration number 4 has good metrics and doesn't have a lot of selected columns (only 11 instead of the 19 initial ones).
Among the 11 final columns of this feature selection, the encoded categorical ones are: DOMAINE_EMAIL, SECTEUR_VENDEUR, LEVEL_CLEAN, BRAND_CLEAN, TYPE_CLEAN

Now, let's compare the performances on the test dataset. For a given encoder, we will use the 11 columns selected in the Feature Selection process. And run a random search with cross validation in order to find the "best set" of hyperparameters for a random forest. Once we have this set of hyperparameters, we can train and test the Random Forest on the encoded data.

The following schema resumes the process steps for 1 encoder:

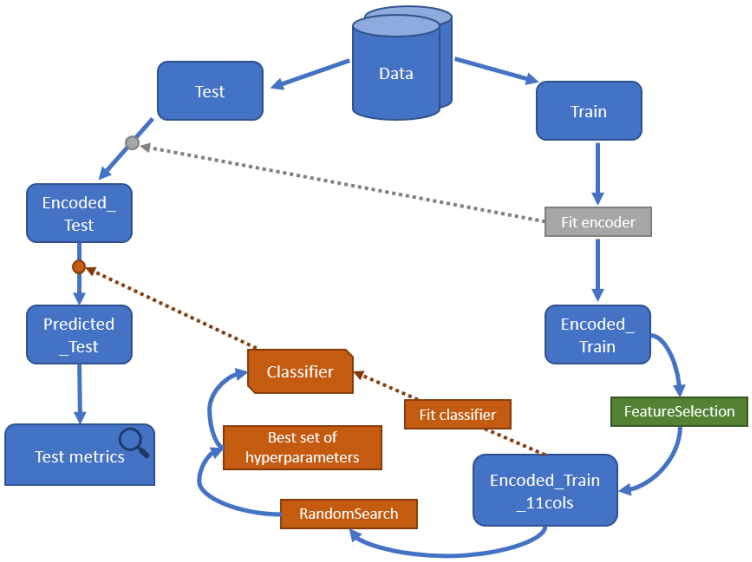


Figure 26: Steps schema

After processing this program, we obtained the following table on the **test** data with a cut-off set at 1%:

Encoder	Precision 1%	Recall 1%	ROC AUC
One Hot Encoder	0,235	0,104	0,702
Binary Encoder	0,252	0,111	0,707
Ordinal Encoder	0,284	0,126	0,734
Count Encoder	0,282	0,125	0,734
Target Encoder	**0,287**	**0,127**	**0,741**
Weight Of Evidence	0,278	0,123	0,739

On the above table, the Target Encoder is the best technique. It outperforms the One Hot Encoder by 5.2% on the precision.

We obtained the following table on the **out of time** data with a cut-off set at 1%:

Encoder	Precision 1%	Recall 1%	ROC AUC
One Hot Encoder	0,233	0,115	0,696
Binary Encoder	0,229	0,101	0,699
Ordinal Encoder	0,265	0,124	0,724
Count Encoder	**0,267**	**0,132**	0,728

Target Encoder	0,258	**0,132**	**0,731**
Weight Of Evidence	0,26	**0,132**	**0,731**

On the above table, the Count Encoder is the best technique. It outperforms the One Hot Encoder by 3.4% on the precision.

Also, it can be interesting to study the difference between the feature importance of their RandomForest. To do so, we will compare the OneHotEncoder and the Target:

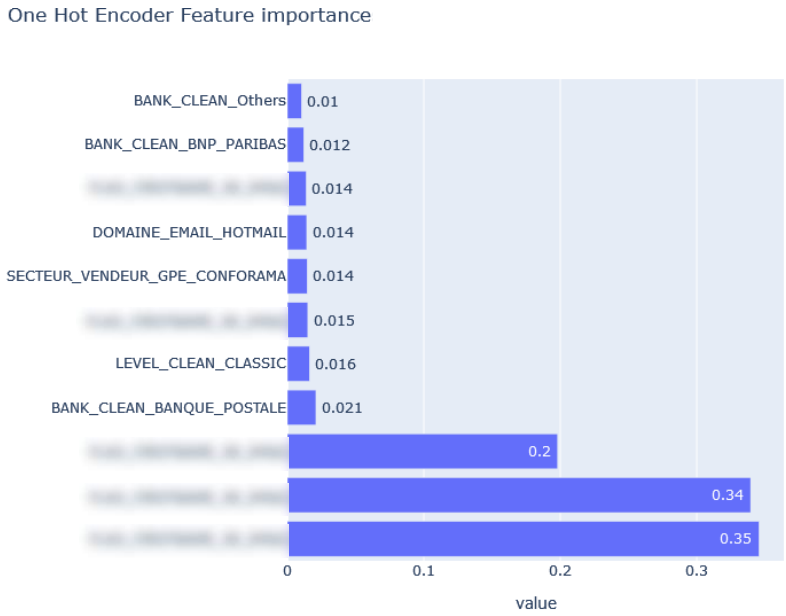


Figure 27: One Hot Encoder Feature importance

In the Feature importance using the One Hot Encoder, 6 features come from 4 encoded categorical variables:

- BANK_CLEAN → BANK_CLEAN_BANQUE_POSTALE, BANK_CLEAN_Others, BANK_CLEAN_BNP_PARIBAS
- LEVEL_CLEAN → LEVEL_CLEAN_CLASSIC
- DOMAINE_EMAIL → DOMAINE_EMAIL_HOTMAIL
- SECTEUR_VENDEUR → SECTEUR_VENDEUR_GPE_CONFORAMA

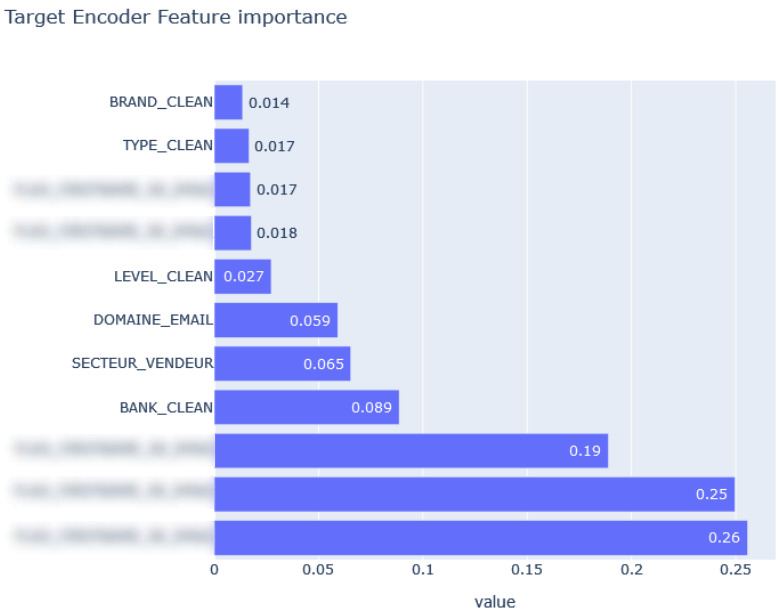


Figure 28: Target Encoder Feature importance

In the Feature importance using the Target Encoder, 6 features come from 6 encoded categorical variables. But compared to the One Hot Encoder, we keep all the information in the variables:

- BANK_CLEAN (all 9 values)
- LEVEL_CLEAN (all 4 values)
- DOMAINE_EMAIL (all 5 values)
- SECTEUR_VENDEUR (all 7 values)
- BRAND_CLEAN (all 3 values)
- TYPE_CLEAN (all 3 values)

In term of interpretability in the feature importance, the Target Encoder seems more "logical" because we conserve all the information of each categorical variable. Plus, it allows us to see the "real" importance of a feature in the Machine Learning model (and not just the importance of 1 feature). For example, if we sum the importance of the 3 BANK_CLEAN variables in the One Hot Encoder Feature importance, we obtain 0.01 + 0.012 + 0.021 = 0.043. It's half of the feature importance of BANK_CLEAN in the Target Encoder (0.089).

We can conclude that the Target Encoder is a better technique in this scenario because of its positive impact in the dimension, interpretability and increase in the performances. It would have increased the performances of a "traditional model" (using the One Hot Encoding technique).

iii. DomoFinance project

The DomoFinance train data frame has 106 625 lines and 7 columns. The variables have the following type:

- 4 numerical (LOAN_DURATION, APPLICANT_AGE, APPLICANT_SALARY, RATIO_REQUEST_AMOUNT_BY_APPLICANT_SALARY)

- 3 categorical (PRODUCT_TYPE, MARITAL_STATUS, CSP_STATUS)

The target variable is "BAD_PAYER" which indicates if the individual has committed a fraud or not.

When running the encoding techniques tester with the default RandomForest provided by scikit-learn, we obtain the following results (the threshold is set at 1%):

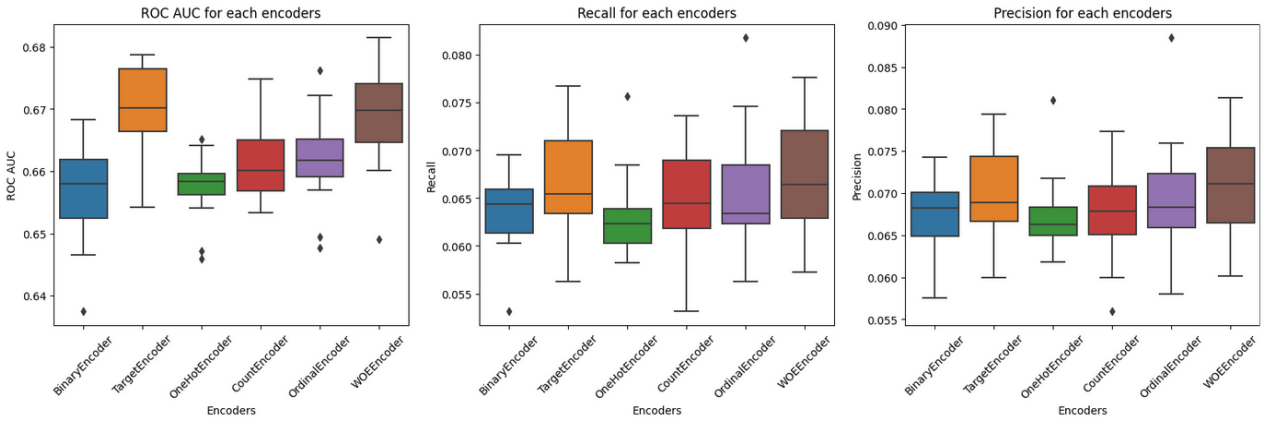


Figure 29: DomoFinance encoding methods metrics

We can see that with this RandomForest, the Target and Weight of Evidence encoders have better metrics performances then others.

Now we want to see how these encoders perform compare to others when using the "best" RandomForest for their cases. To do so, we will use a random search (combined with cross validation) function in order to determine what are the best hyperparameters set for a RandomForest using the Weight of Evidence or Target encoding.

- **Weight of Evidence** best set of hyperparameters:

```
'n_estimators': 200,
'max_depth': 44,
'max_features': 'log2',
'max_samples': 0.5,
'min_samples_split': 10,
'bootstrap': True,
'criterion': 'gini',
'class_weight': 'balanced'
```

Figure 30: DomoFinance WOE "best" RandomForest hyperparameters

Using the above parameters, we obtained the following results during the cross-validation on the train set:

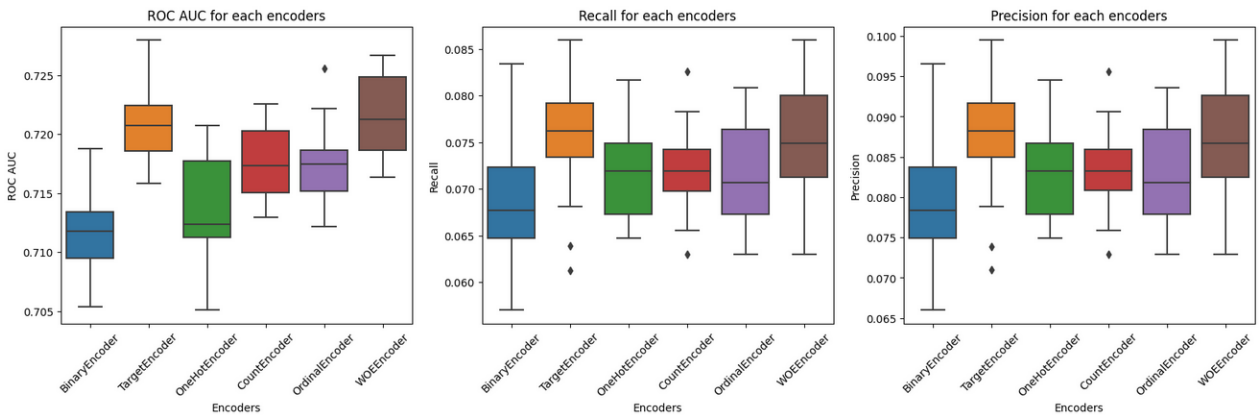


Figure 31: DomoFinance metrics with WOE "best" RandomForest hyperparameters

As we can see, WoE and Target encoders performed the best on the train dataset for all the 3 metrics.

On the test dataset we obtained the following results:

Encoder	Precision 1%	Recall 1%	ROC AUC
Binary Encoder	0.072	0.065	0.724
Target Encoder	0.094	0.082	0.730
One Hot Encoder	0.070	0.061	0.731
Count Encoder	0.075	0.066	0.724
Ordinal Encoder	0.056	0.049	0.726
WOE Encoder	0.075	0.065	0.734

Figure 32: DomoFinance metrics with WOE "best" RandomForest hyperparameters prediction on the test dataset

- **Target** best set of hyperparameters:

```
'n_estimators': 250,
'max_depth': 24,
'max_features': 0.3,
'max_samples': 0.5,
'min_samples_split': 14,
'bootstrap': True,
'criterion': 'gini',
'class_weight': 'balanced'
```

Figure 33: DomoFinance Target "best" RandomForest hyperparameters

Using the above parameters, we obtained the following results during the cross-validation on the train set:

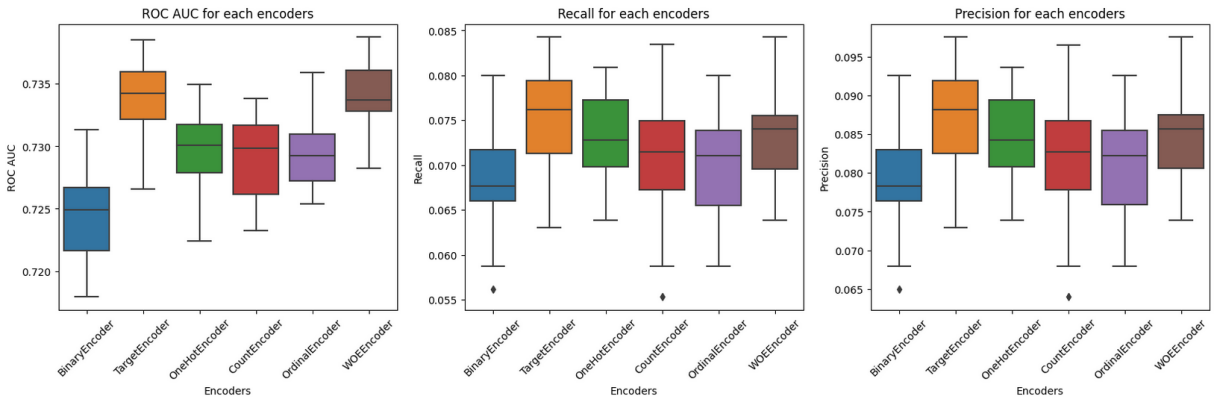


Figure 34: DomoFinance metrics with Target "best" RandomForest hyperparameters

As we can see, WoE and Target encoders perform the best on this train dataset for all the 3 metrics.

On the test dataset we obtained the following results:

Encoder	Precision 1%	Recall 1%	ROC AUC
Binary Encoder	0.084	0.074	0.741
Target Encoder	0.089	0.078	0.740
One Hot Encoder	0.103	0.090	0.733
Count Encoder	0.094	0.082	0.732
Ordinal Encoder	0.061	0.053	0.749
WOE Encoder	0.075	0.066	0.729

Figure 35: DomoFinance metrics with Target "best" RandomForest hyperparameters prediction on the test dataset

- **Binary** best set of hyperparameters:

```
'n_estimators': 250,
'max_depth': 34,
'max_features': 0.4,
'max_samples': 0.5,
'min_samples_split': 14,
'bootstrap': True,
'criterion': 'gini',
'class_weight': None
```

Figure 36: DomoFinance Binary "best" RandomForest hyperparameters

Using the above parameters, we obtained the following results during the cross-validation on the train set:

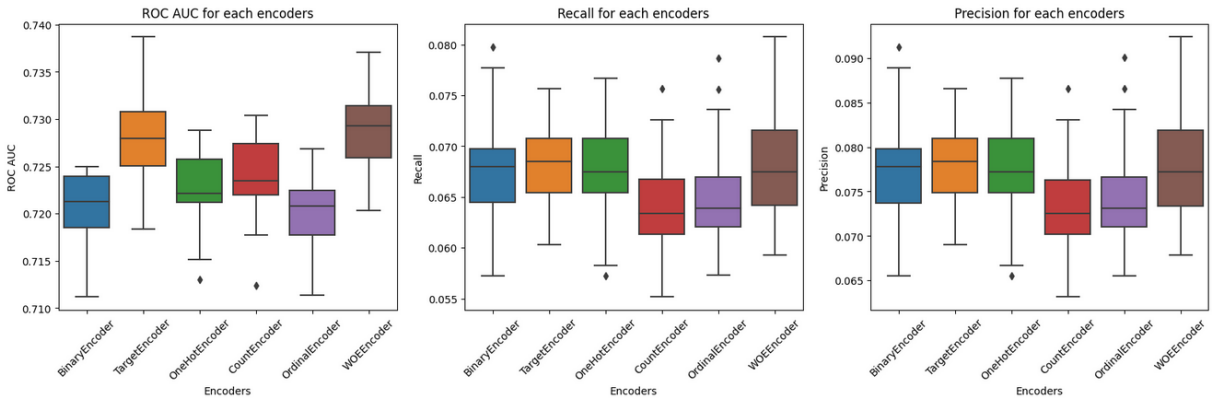


Figure 37: DomoFinance metrics with Binary "best" RandomForest hyperparameters

As we can see, WoE and Target encoders perform the best on this train dataset for all the 3 metrics.

On the test dataset we obtained the following results:

Encoder	Precision 1%	Recall 1%	ROC AUC
Binary Encoder	0.086	0.074	0.743
Target Encoder	0.075	0.066	0.751
One Hot Encoder	0.108	0.094	0.739
Count Encoder	0.089	0.078	0.743
Ordinal Encoder	0.089	0.078	0.747
WOE Encoder	0.070	0.061	0.747

Figure 38: DomoFinance ROC_AUC with Binary "best" RandomForest hyperparameters prediction on the test dataset

- **OneHotEncoder** best set of hyperparameters:

```
'n_estimators': 150,
'max_depth': 22,
'max_features': 0.4,
'max_samples': 0.6,
'min_samples_split': 14,
'bootstrap': True,
'criterion': 'gini',
'class_weight': None
```

Figure 39: DomoFinance OHE "best" RandomForest hyperparameters

Using the above parameters, we obtained the following results during the cross-validation on the train set:

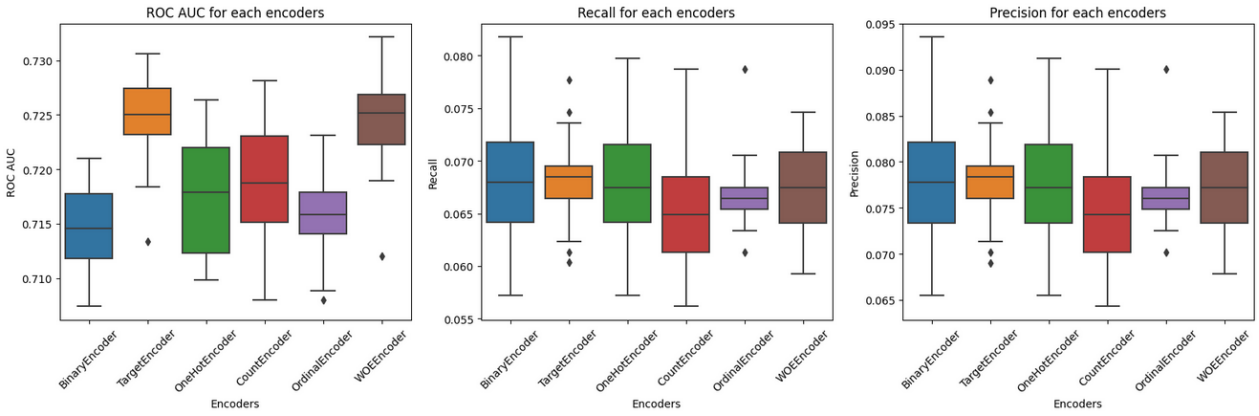


Figure 40: DomoFinance metrics with OHE "best" RandomForest hyperparameters

As we can see, WoE and Target encoders perform the best on this train dataset for the ROC AUC metric. But on recall and precision Binary, OHE and Count encoders are better (but these metrics relate on the threshold).

On the test dataset we obtained the following results:

Encoder	Precision 1%	Recall 1%	ROC AUC
Binary Encoder	0.080	0.070	0.731
Target Encoder	0.070	0.061	0.749
One Hot Encoder	0.089	0.079	0.738
Count Encoder	0.085	0.074	0.743
Ordinal Encoder	0.089	0.078	0.727
WOE Encoder	0.070	0.061	0.742

Figure 41: DomoFinance ROC_AUC with OHE "best" RandomForest hyperparameters prediction on the test dataset

- **CountEncoder** best set of hyperparameters:

```
'n_estimators': 250,
'max_depth': 32,
'max_features': 'sqrt',
'max_samples': 0.8,
'min_samples_split': 8,
'bootstrap': True,
'criterion': 'gini',
'class_weight': None
```

Figure 42: DomoFinance Count "best" RandomForest hyperparameters

Using the above parameters, we obtained the following results during the cross-validation on the train set:

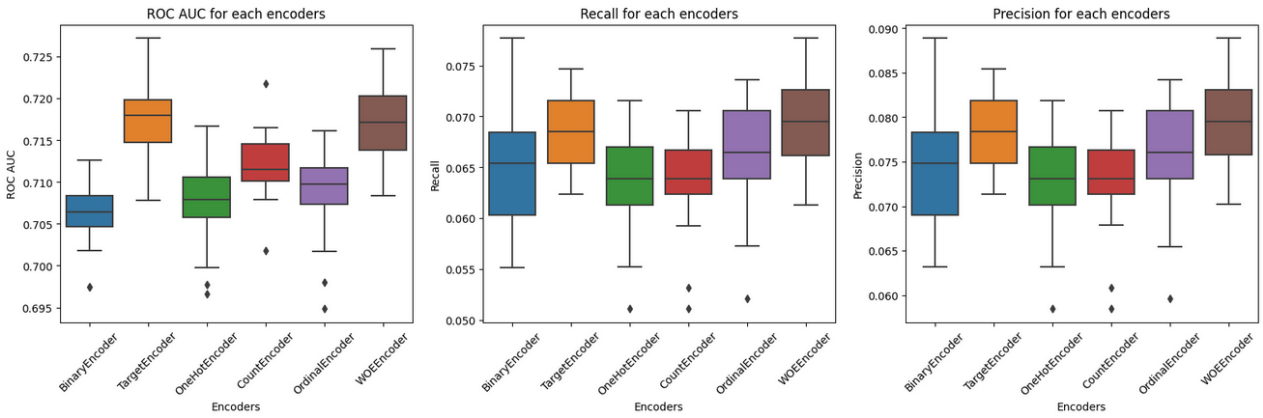


Figure 43: DomoFinance metrics with Count "best" RandomForest hyperparameters

As we can see, WoE and Target encoders perform the best on this train dataset for the ROC AUC metric. On recall and precision Binary, Target and WOE encoders are better.

On the test dataset we obtained the following results:

Encoder	Precision 1%	Recall 1%	ROC AUC
Binary Encoder	0.080	0.070	0.701
Target Encoder	0.080	0.070	0.737
One Hot Encoder	0.089	0.078	0.723
Count Encoder	0.080	0.070	0.726
Ordinal Encoder	0.070	0.061	0.720
WOE Encoder	0.080	0.070	0.732

Figure 44: DomoFinance metrics with Count "best" RandomForest hyperparameters prediction on the test dataset

- **OrdinalEncoder** best set of hyperparameters:

```
'n_estimators': 250,
'max_depth': 22,
'max_features': 0.3,
'max_samples': 0.6,
'min_samples_split': 10,
'bootstrap': True,
'criterion': 'gini',
'class_weight': None
```

Figure 45: DomoFinance Ordinal "best" RandomForest hyperparameters

Using the above parameters, we obtained the following results during the cross-validation on the train set:

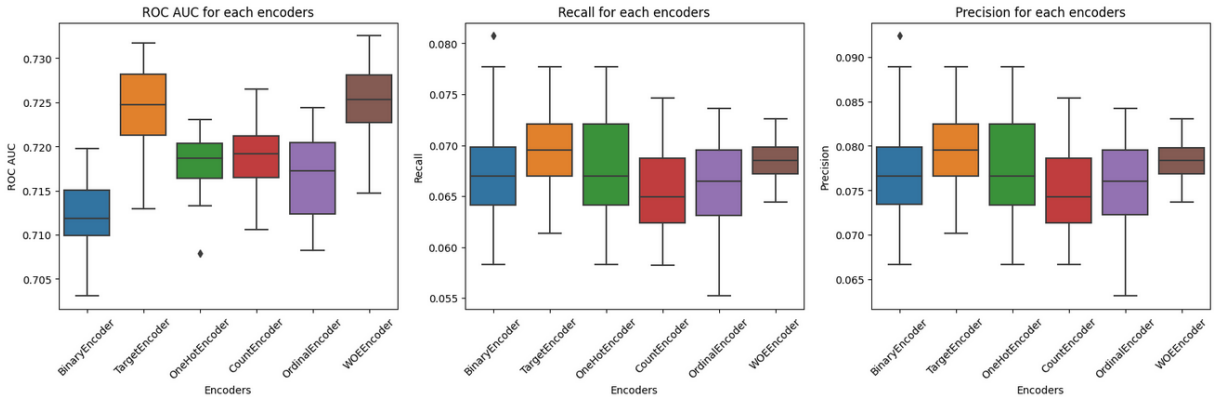


Figure 46: DomoFinance metrics with Ordinal "best" RandomForest hyperparameters

As we can see, WoE and Target encoders perform the best on this train.

On the test dataset we obtained the following results:

Encoder	Precision 1%	Recall 1%	ROC AUC
Binary Encoder	0.089	0.078	0.721
Target Encoder	0.093	0.082	0.742
One Hot Encoder	0.085	0.074	0.727
Count Encoder	0.070	0.061	0.746
Ordinal Encoder	0.094	0.082	0.732
WOE Encoder	0.070	0.061	0.737

Figure 47: DomoFinance metrics with Count "best" RandomForest hyperparameters prediction on the test dataset

5. Conclusion

As we saw, the target-based encoding technics (Target and WeightOfEvidence) provide good, if not better, results than the classic OneHotEncoder used in the scoring center. While keeping a lower number of columns.

One of the main advantages of using a technique that doesn't increase the number of columns, is that it allows us to gain a lot of time on some process (like Feature Selection). And has feature importance process that makes more sense by keeping all the information inside a variable. But it doesn't mean that we should replace the current method by these new ones. The performance depends on the dataset. And it could variate from one to another. That's why I recommend comparing One Hot Encoder to the target-based one on the same data to make the final choice (and if you have the time).

An interesting next step should be to study "what is the impact of the different encoding techniques when there is a drift in the data?". To answer this question, we could use the Adaptive Random Forest and compare how the different models (encoded with different methods) react to the drift.

Also, it could be interesting to mix up the techniques on the same data depending on what variable we want to encode. For example, when we have a low cardinality categorical variable than OneHotEncoder could be used. But if the cardinality is high, the use of a target-based technique should be considered in order the keep a low dimension. Or in the scenario we have an ordinal data we should use the OrdinalEncoder and assign the correct order to the values inside.

6. References

- Shahul, ES (2020) - An Overview of Encoding Techniques, Kaggle.com
- Baijayanta, ROY (2023) - All about Categorical Variable Encoding, Towardsdatascience.com
- John, HANCOCK and Taghi, KHOSHGOFTAAR (2020) - Survey on categorical data for neural networks, Journalofbigdata.springeropen.com
- Shipra, SAXENA (2022) - Here's All you Need to Know About Encoding Categorical Data (with Python code), Analyticsvidhya.com
- Matt, CLARKE (2021) - How to use transform categorical variables using encoders, Practicaldatasience.co.uk
- Denis, VOROTYNTSEV (2019) - Benchmarking categorical encoders, Towardsdatascience.com
- Max, HALFORD (2018) - Target encoding one the right way, Maxhalford.github.io
- *category_encoders* - Github.io, Documentation
- Vinícius Trevisan (2022) - Interpreting ROC Curve and ROC AUC for Classification Evaluation, Towardsdatascience.com
- Julie CAVARROC (2022) - fra-f-fullcb project

Chapter 2: South Africa Logistical Regression model for RCS

1. Introduction

The Scoring Center teams often use Logistical Regression models. The aim of these models is to attribute a "score" based on the input information of a client. It is a grade based on the values of his explanatory variables (for example: age, Socio-Professional Category, income).

A score function can be represented by a score grid such as the following one:

Age	**< 28**	**28 ≤ Age < 50**	**50+**
	90	110	125
SPC	**Student**	**Other**	**Dentist, notary**
	90	102	128
Income	**< 800**	**800 ≤ Income < 4 000**	**4 000+**
	90	119	135

Figure 48: Score card example

Based on this grid, an individual's score can be calculated by doing the sum of the weights corresponding to each one of his attributes.

For example, in the score presented in the above table, a 32-year-old dentist with a monthly income above 4,000 euros would have a score of 373 (= 110 + 128 + 135).

Now that we can calculated a score, it's possible to build subpopulations sorted by the model's predicted risk level on an application population. And so, we can build a table to determine what actions to take according to the different costumer's profiles:

Score	Predicted risk level	Possible action
≤ 352	High	Refusal
353 – 366	Average	Require further study
367 – 380	Moderate	Loan application accepted (*under the limit of 5 000€)*
381+	Low	Loan application accepted

Figure 49: Action table example

In order to be used in credit risk management, the score model must be **accurate**, **discriminating**, and **generalizable**:

- A model is **accurate** when, on the development population, the predicted risk rate per score range is close to the observed rate (i.e. when it represents accurately the reality).
- A score is **discriminating** on a population when it manages to rank it in subpopulations whose scale of variation of risk rates is wide. It can also be said that a high-performing score makes it possible to discriminate the population of interest into two classes defined by the target.
- It is said that a score has a good ability to **generalize** when it is discriminating and is accurate on a population that has not been used for development, whose characteristics are like those of the development population.

2. Context

The RCS Group is an entity based in South Africa, being part of BNP Paribas Personal Finance since 2014. RCS offers a range of retail financial services and products under its brand and in partnership with retailers in different markets. The core purpose of RCS is to improve people's lifestyle being innovative in offering desirable and accessible credit products. Its offerings include clients store cards, cash and retail loans besides insurance offerings such as personal accident, critical illness, and income protection. The idea is to enable RCS customers to enjoy the convenience, comfort and peace of mind offered by these credit solutions.

Risk score models are used in credit risk management to assess the solvency of clients, as well as the potential risk of non-repayment of loans granted by credit institutions or banks. In our case, this score was needed to predict if a client will be a "bad dept" on credit cards payment, only based on his granting data.

The entity asked to develop a scorecard using a Logistic Regression.

3. Data

For some confidential reason, we will hide a maximum of information about the used variables while keeping the report understandable.

a. Presentation

The nature of data used are internal and Credit Bureau.
For this model, we will be only using "granting" data. Which means that it will only use the data provided by the client when he/she fulfilled the form to give his/her information.

We have 44 exploratory variables. We are using data related to the applicant incomes, related to his/her job, related to his/her expenses, to his/her sociodemographic situation, external information, and product related information.

The target variable is 'BD' (Bad Debt), which is equal to 1 if the client did default 2 times on his credit card payment. Otherwise, it's equal to 0.

There are 7273 individuals in the dataset. With 29.8% (2168 individuals) of BD.

We are working with a low volume of data (less than 10 000 individuals) and a risky population.

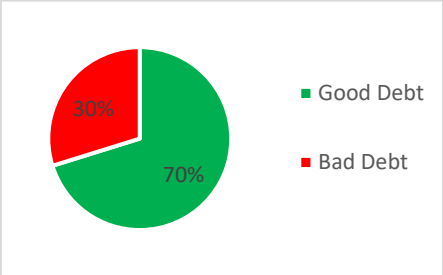
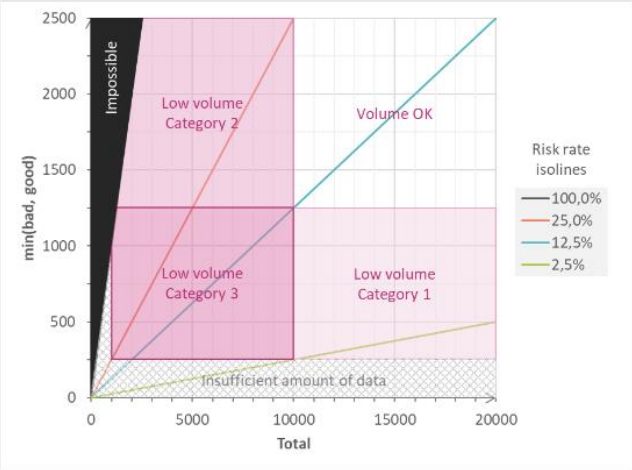


Figure 50: Risk repartition



Based on the Scoring Center guideline on low volume data, we are in a Category 2 of the low volume scenarios. The total amount of available data is limited (< 10 000) but there is a high-risk rate, giving more than 1250 bad payers.

Figure 51: Low volumes scenarios

b. Data Analysis

The top3 variables with the highest Information Value (IV) are:

- Store dealer ID with 0.51
- Applicant gross income with 0.50 (and a Gini of 0.23)
- Expensed money for transports per month with 0.23 (and a Gini of 0.14)

To have an easier reading of the graphs for the data analysis of categorical data, we will group the categories with less than 5% of occurrence to a new category called 'OTHER'. For example, when looking at the variable store dealer ID, we have 579 unique values. Because a graph with this high number of categories is not readable, after grouping, we obtained the follow categories:

- MISSING with around 5.88% of occurrence
- 595886 with around 6.21% of occurrence
- OTHER (the new category) with around 88.00% of occurrence

Thanks to this method we can have a better understanding of the data. And so, we obtained the following graphics (the black dash line being the total averaged risk and the black dot being the averaged risk level in the category):

- Store dealer ID

Figure 52: Store dealer ID analysis

We can observe that the store dealer number 595886 and the MISSING one have a lower risk level than the average. These two categories allow us to positively “discriminate” some individuals based on where they contracted their credit card.

- Applicant gross income

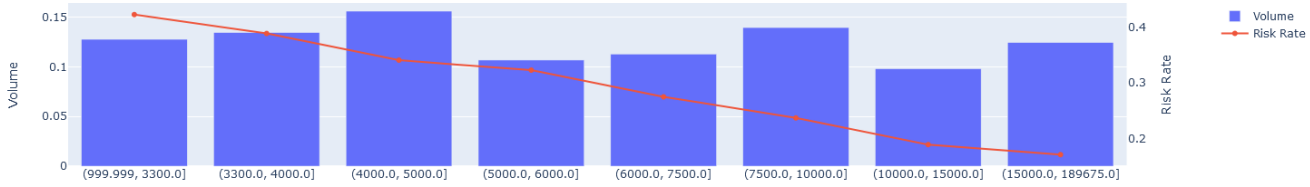


Figure 53: Applicant gross income analysis

We can observe that increasing the applicant gross income, lowers the risk to be a “bad debt”. With a stark difference of risk between the lowest cut (less than 3300 income), with 43% of risky, and the highest incomes (more then 15000), with 17% of risky.

- Expensed money for transports per month

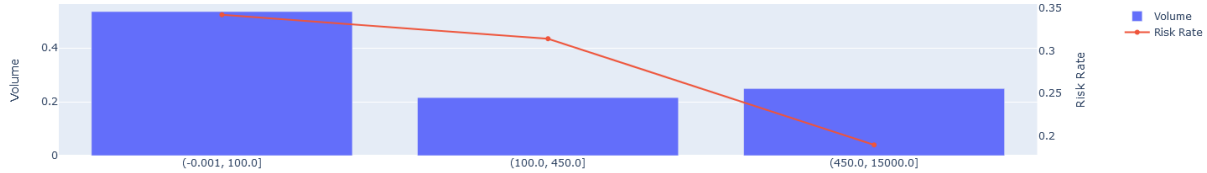


Figure 54: Expensed money for transports per month analysis

We can observe that increasing the expense related to the transport lowers the risk rate. The risk goes from 34% when having less than 100 for the transports. To 19% when having more than 450. So, an individual who allocates more money for his monthly transports is less risky than the others.

For some confidential reason, we will only use the month for the data collection.

The period is too short to analyze the stability of the data. The data was collected over 2 different periods. The first is from November to mid-December. And the second one is from February to May. There is a big break is the data collection starting from mid-December to mid-February.

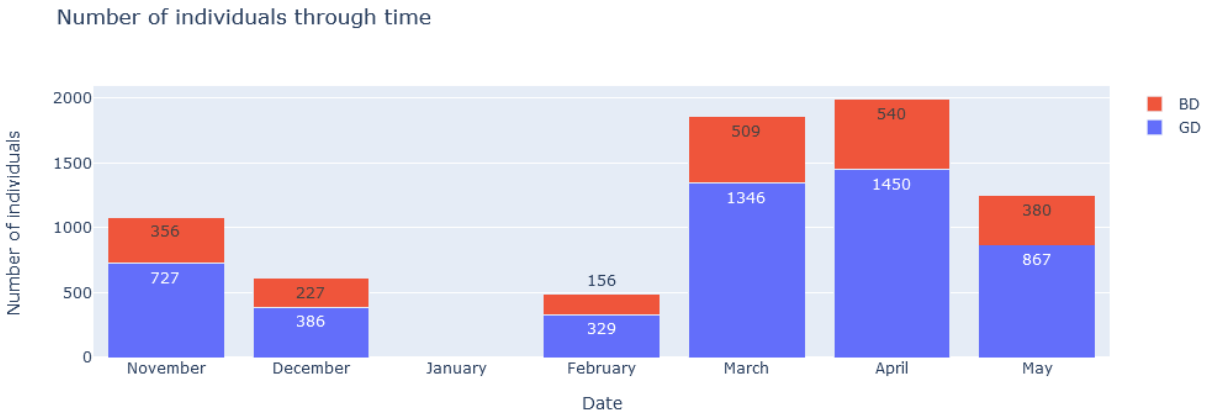


Figure 55: Population per month analysis

Thanks to the above graph, we can see that most of the credit card were contracted after February.

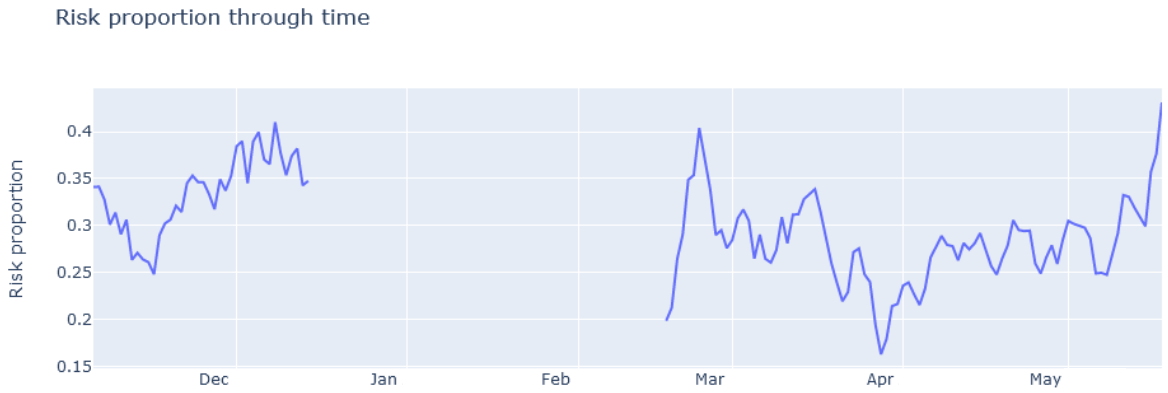


Figure 56: Risk proportion through time

On the above graph, we can observe that the risk proportion is slightly higher on the first part of the data collection than on the second part.

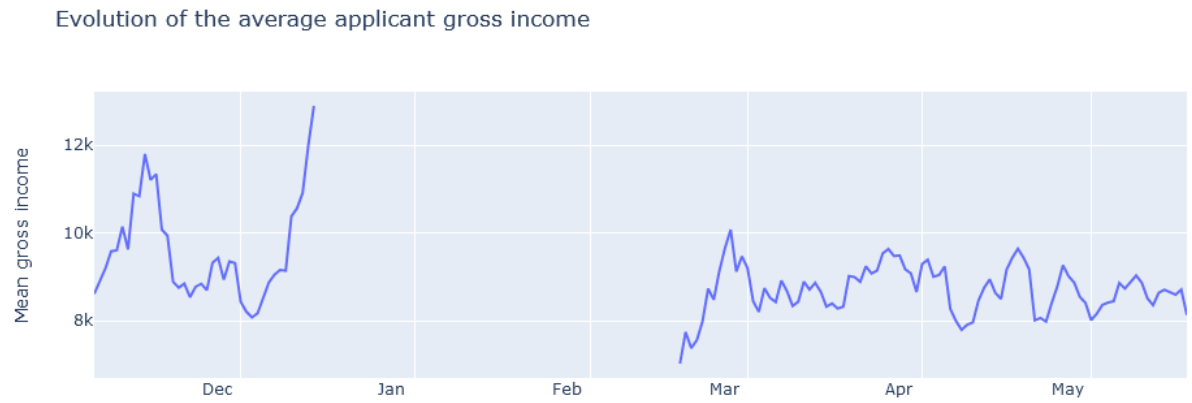


Figure 57: Average applicant gross income through time

The average income of the applicants seems stable through time.

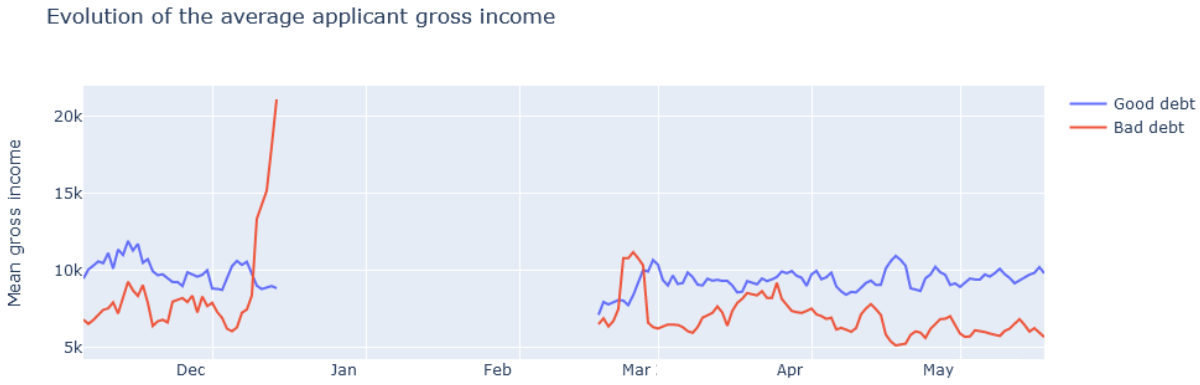


Figure 58: Average applicant gross income through time grouped by BD

This time, when looking at the average gross income but grouped by “Good debt” and “Bad debt”, we observe that the bad debts have a lower gross income than the good one. This discrimination is respected through time.

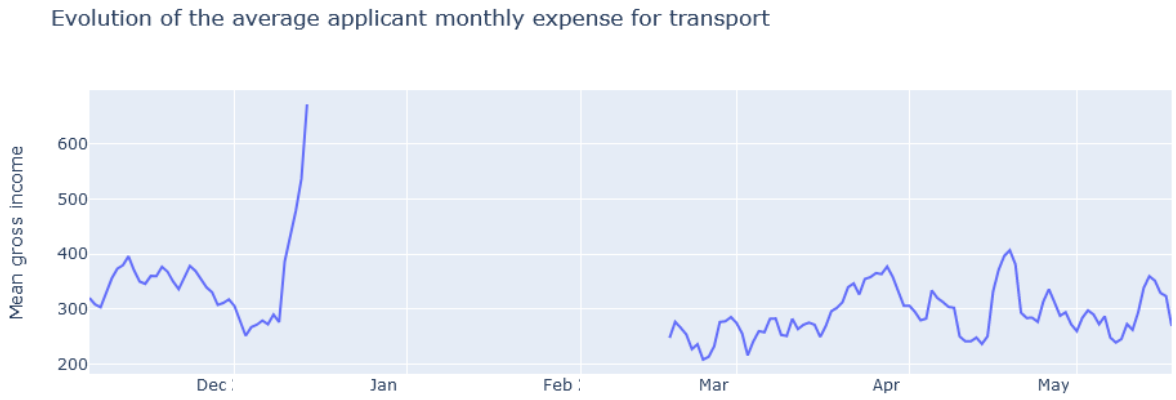


Figure 59: Average applicant transport expenses

The average monthly expense for transport of the applicants seems stable through time.

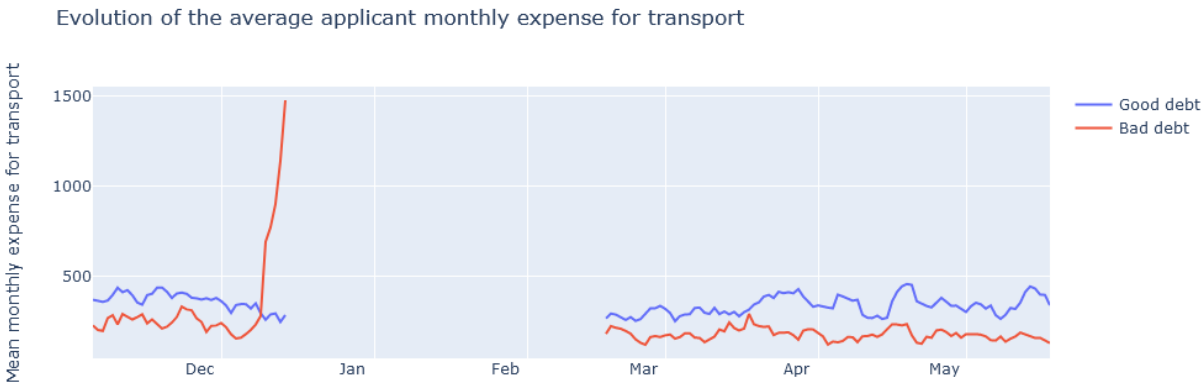


Figure 60: Average applicant transport expenses grouped by BD

This time, when looking at the monthly expense for transport but grouped by “Good debt” and “Bad debt”, we can observe that the bad debts have a lower monthly expense for transport than the good one. This discrimination is respected through time.

c. Modelization data base representativity

When build a granting score, we only focus on the financed population because when know their payment comportment. This population may not be representative of the total demand. That is what we call the selection bias.

d. Preselection of exploratory variables

There are only two potential explicative variables (more than 0.02 of IV) with a volume of missing values greater than 5%:

- Store dealer ID (IV = 0.514 and missing values volume = 5.789%)
- External score (IV = 0.232 and missing values volume = 41.565%)

4. Methodology

a. Description

The model adjustment was realized thanks to the internal tool of the Scoring Center and results in 2 steps:

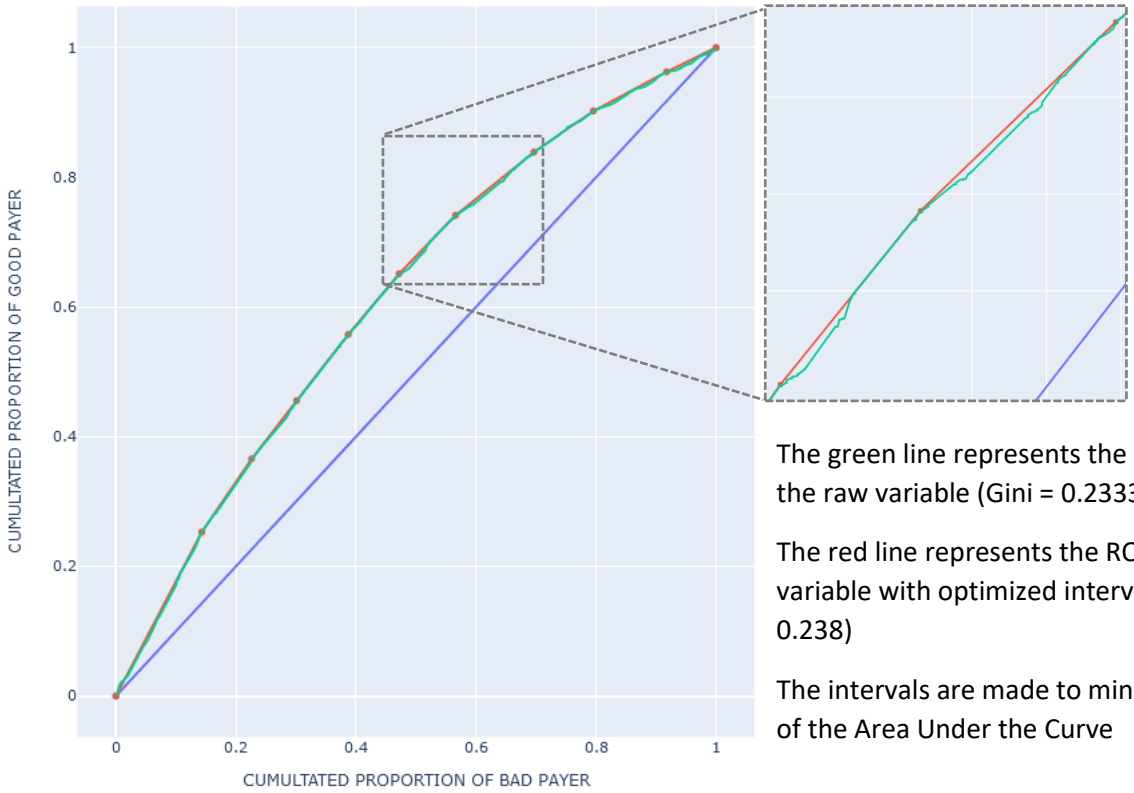
- Cutting the variables based on the maximization on the Gini index, while respecting constraint of minimal volumes at 5% (low volumes scenario)
- Iterative logistic regression which adds to the model (at each step) the variable having the highest value of additional information

First step of modelling is quantitative variables binning. This binning should respect the following rules:

There must be at least 5% of the population in all modalities

Binning should respect business sense.

To do so, we tried to maximize the Gini of the variable with the intervals. While respecting the constraints. Here is an example with the applicant gross income:



The green line represents the ROC curve of the raw variable (Gini = 0.2333)

The red line represents the ROC curve of the variable with optimized intervals (Gini = 0.238)

The intervals are made to minimize the loss of the Area Under the Curve

Figure 61: Binning modalities method

Qualitative variables are binned (regrouped) using business insight and descriptive statistics (such as Weight of Evidence for example). When working on categorical data, we want to regroup the categories that don't respect the volume constraint with the nearest group of categories based on the risk rate.

Missing values were analyzed to eliminate variables with too many of them, and to understand the reasons behind these missing values. The logical missing values were identified and marked by special values in all credit bureau variables.

The following iterative process was applied to select variables or cross-variables for each tested model:

- Step 1: Statistical ranking based on Information Value of binned variables
- Step 2: The most powerful variables is chosen as the first variable of the model
- Step 3: Remaining variables are ranked by delta IV (remaining Information Value that is not taken into account by the model), and the most powerful variable is added into the model.

This iterative process continues till there is not enough remaining IV available. After each step, several checks were performed:

- Nullity Wald test on logistic regressions coefficients
- Nested dummy test
- Business meaning and global grid consistency with the local teams.

If one of the given above checks was not fulfilled, the variable could be dropped, or its binning could be adjusted. Performance level was also checked: if a variable does not improve model's discrimination, it is not added. Indeed, a model with fewer characteristics is preferable, other things being equal.

b. Sampling

After discussing with the low volumes referent, we decided to train the model on the entire database. And then, when we obtained the selected features and the final categories, evaluate it with a cross-test on 5 folds.

c. Final model selection

For some confidential reason, we will change the variable names by *X_i* for the selected variables in the scorecards.

The first scorecard obtained was the following:

Variable	Modalities	Score	Risk rate	Significative test
X_1	(-inf, 3764]	38	0.43	True
	(3764, 4120]	45	0.37	True
	(4120, 4860]	50	0.34	False
	(4860, 5412]	49	0.33	False
	(5412, 6010]	51	0.32	False
	(6010, 7200]	54	0.29	False
	(7200, 9900]	57	0.24	False
	(9900, 12462]	61	0.21	False
	(12462, 20183]	67	0.17	False
	(20183, inf]	68	0.16	False
X_2	('2.0', '33.0')	38	0.42	True
	('17.0', '26.0', '6.0')	44	0.38	False
	('11.0',)	50	0.35	True
	('12.0', '4.0')	46	0.30	False
	('23.0',)	53	0.30	False

	('1.0', '20.0', '25.0', '3.0', '8.0')	51	0.27	False
	('13.0',)	54	0.25	False
	('14.0', '15.0', '16.0', '19.0', '21.0', '22.0', '27.0', '30.0', '5.0', '7.0')	60	0.22	False
X_3	(-inf, 158.0]	38	0.34	True
	(158.0, 480.0]	42	0.31	False
	(480.0, 570.0]	44	0.24	False
	(570.0, 960.0]	53	0.19	True
	(960.0, inf)	57	0.13	False
X_4	('CMGC', 'DSAR', 'DSCS', 'FICE', 'FISW', 'MEUR', 'NPOL', 'OILC', 'SDFW', 'SDIE')	38	0.38	True
	('NPOS',)	43	0.37	False
	('CMCM', 'DSOR', 'FIPO', 'OCSH')	45	0.34	False
	('CMRE', 'DSLT', 'FIMT', 'FIRE', 'NPOH', 'NPUA', 'OIOT')	47	0.30	False
	('AGRI', 'FIHF', 'MEOM', 'OIPS')	52	27	False
	('CMAR', 'CMAT', 'CMCA', 'CMMT', 'CMNM', 'CMOG', 'CMOP', 'CMSP', 'DSAA', 'DSCB', 'DSGA', 'DSGS', 'DSME', 'DSOW', 'DSPS', 'DSUP', 'FIDP', 'FINR', 'FIOF', 'MEOG', 'MEPS', 'NPPO', 'NPRO', 'NPSU', 'OACN', 'OCAB', 'OCAI', 'OCNE', 'OCOT', 'OIAR', 'OIHC', 'OIMP', 'OIRM', 'OITP', 'OPSA', 'SDAH', 'SDPB')	57	0.21	False
X_5	(1.0, inf)	38	0.36	True
	(-1.0, 1.0]	43	0.34	False
	(-inf, -1.0]	55	0.25	True

X_6	('15', '32', '38', '52', '60')	38	0.39	True
	('14', '30', '34', '40', '43', '59')	41	0.33	False
	('9',)	46	0.29	True
	('31', '46')	41	0.24	False
	('16', '22', '25', '27', '29', '37', '44', '61')	53	0.18	True

Figure 62: First scorecard

The objective is to obtain a scorecard with only "True" values in the "Significative test" column. To do so, we will group the different modalities based on the risk rate. Applying this method might change the Information Value of the variables. And so, might change the final selected variables over processing different iterations.

In the end after grouping the modalities to obtain "True" in the "Significative test", we obtained the following scorecard for the IV:

Variable	Modalities	Information Value
X_1	(-inf, 3764]	0.06
	(3764, 6010]	0.01
	(6010, 12462]	0.02
	(12462, inf)	0.08
TOTAL		**0.17**
X_2	High_risk	0.04
	Medium_risk	0.00
	Low_risk	0.05
TOTAL		**0.09**
X_4	High_risk	0.02
	Medium_risk	0.00
	Low_risk	0.06
TOTAL		**0.09**
X_3	(-inf, 480]	0.02
	(480, inf]	0.08
TOTAL		**0.10**
X_6	High_risk	0.02
	Medium_risk	0.00
	Low_risk	0.04
TOTAL		**0.06**
X_7	(6, inf)	0.01
	(1, 6]	0.00
	(0, 1]	0.00
	(-inf, 0]	0.01
TOTAL		**0.02**
X_8	(-inf, 0.118]	0.00
	(0.118, inf)	0.05
TOTAL		**0.05**

Figure 63: Final scorecard with IV

In order to ease the reading, we renamed the modalities (only for categorical variables) based on their risk level:

- X_2:
  - High_risk: 2.0, 33.0, 17.0, 26.0, 6.0, 11.0
  - Medium_risk: 12.0, 4.0, 23.0, 1.0, 20.0, 25.0, 3.0, 8.0
  - Low_risk: 13.0, 14.0, 15.0, 16.0, 19.0, 21.0, 22.0, 27.0, 30.0, 5.0, 7.0
- X_4:
  - High_risk: CMGC, DSAR, DSCS, FICE, FISW, MEUR, NPOL, OILC, SDFW, SDIE
  - Medium_risk: CMCM, DSOR, FIPO, OCSH, CMRE, DSLT, FIMT, FIRE, NPOH, NPUA, OIOT, AGRI, FIHF, MEOM, OIPS
  - Low_risk: CMAR, CMAT, CMCA, CMMT, CMNM, CMOG, CMOP, CMSP, DSAA, DSCB, DSGA, DSGS, DSME, DSOW, DSPS, DSUP, FIDP, FINR, FIOF, MEOG, MEPS, NPPO, NPRO, NPSU, OACN, OCAB, OCAI, OCNE, OCOT, OIAR, OIHC, OIMP, OIRM, OITP, OPSA, SDAH, SDPB
- X_6:
  - High_risk: 15, 32, 38, 52, 60, 14, 30, 34, 40, 43, 59
  - Medium_risk: 9, 31, 46
  - Low_risk: 16, 22, 25, 27, 29, 37, 44, 61
  -

There are no remaining variables with a Delta IV > 0.02.

We obtained the following final scorecard:

Variable Name	Variable Description	Modalities and coefficients			
X_1	Related to the applicant income	(-inf, 3764]	(3764, 6010]	(6010, 12462]	(12462, inf)
		39	49	58	70
X_2	Related to the applicant job	High_risk	Medium_risk	Low_risk	
		39	44	53	
X_4	Related to the applicant job	High_risk	Medium_risk	Low_risk	
		39	45	55	
X_3	Related to the applicant expenses	(-inf, 480]	(480, inf]		
		39	50		
X_6	Related to the product	High_risk	Medium_risk	Low_risk	
		39	44	53	
X_7	Credit Bureau information	(6, inf)	(1, 6]	(0, 1]	(-inf, 0]
		39	52	59	65
X_8	Related to the applicant expenses	(-inf, 0.118]	(0.118, inf)		
		39	51		

Figure 64: Final scorecard with coefficients

As said before, because we are working on a low volume scenario, we don't have any test or validation samples. But we can evaluate the model by using techniques that train multiple times the model on a part of the data, different each time, and each time evaluate this model on the left-out data.

One of this evaluation methods is using cross-test (with k-fold test), we cut the total dataset into k parts, train the model k times on k-1 folds. And each time evaluate it on the left-out fold. The recommended parameter is k ∈ ⟦3;5⟧. In our case k is equal to 5 folds.

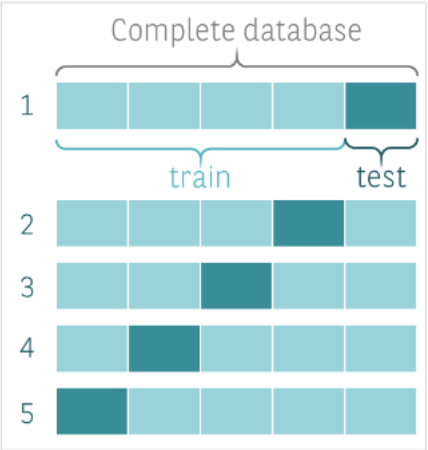


Figure 65: Cross test explanation

With this method, using five folds, we obtained the following performances results (using quartiles). Looking at the Gini:

Figure 66: Gini using StratifieKFold

We can expect to have a Gini value on the train between 0.33 and 0.35, with a median at 0.34 (mean of 0.341). And a Gini value on the test between 0.30 and 0.375, with a median at 0.33 (mean of 0.333).

Looking at the Log Odds slope:

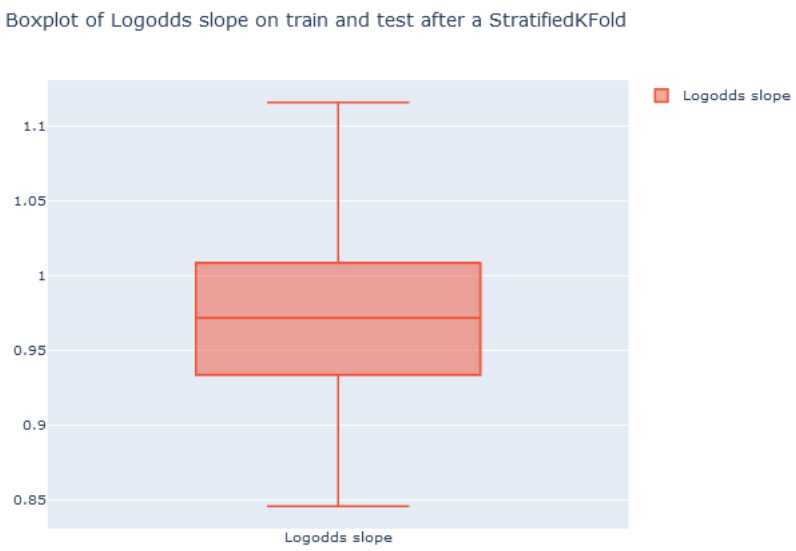


Figure 67: Log Odds using StratifiedKFold

We can expect to have a Log Odds slope value between 0.846 and 1.116, with a median at 0.972 (mean of 0.974). Being around 1 is very nice.

Looking at the Stability Index:

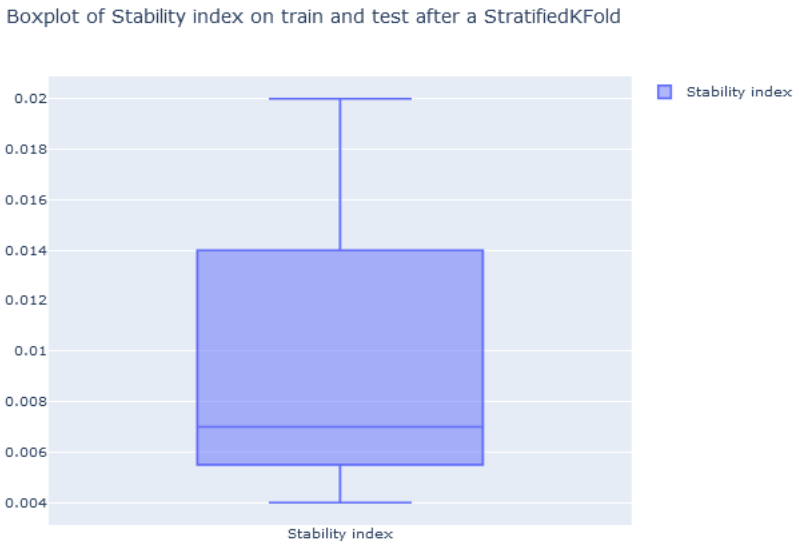


Figure 68: Stability Index using StratifiedKFold

We can expect to have a Stability Index value between 0.002 and 0.004, with a median at 0.007 (mean of 0.0098). Being the nearest to 0 is nice.

Let's now look at the risk rate in each of these categories:

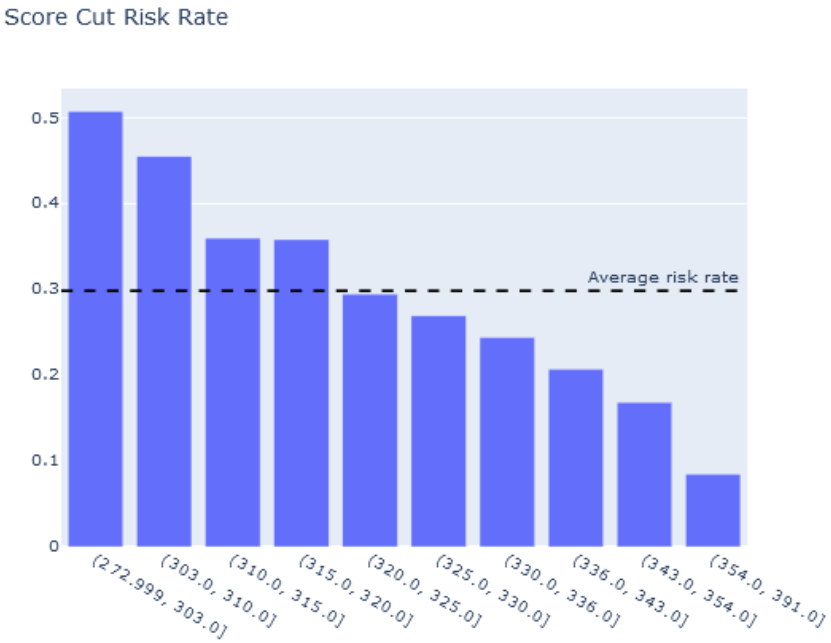


Figure 69: Risk rate by score cuts

When looking at the above graph, we can see that the risk rate is decreasing by cuts (there is no inversion). Just for the intervals (310.0, 315.0] and (315.0, 320.0], the risk rates are very close (0.359 and 0.358).

Here is an overview of the volumes and risk rate of the modalities for the selected variables:

X_1

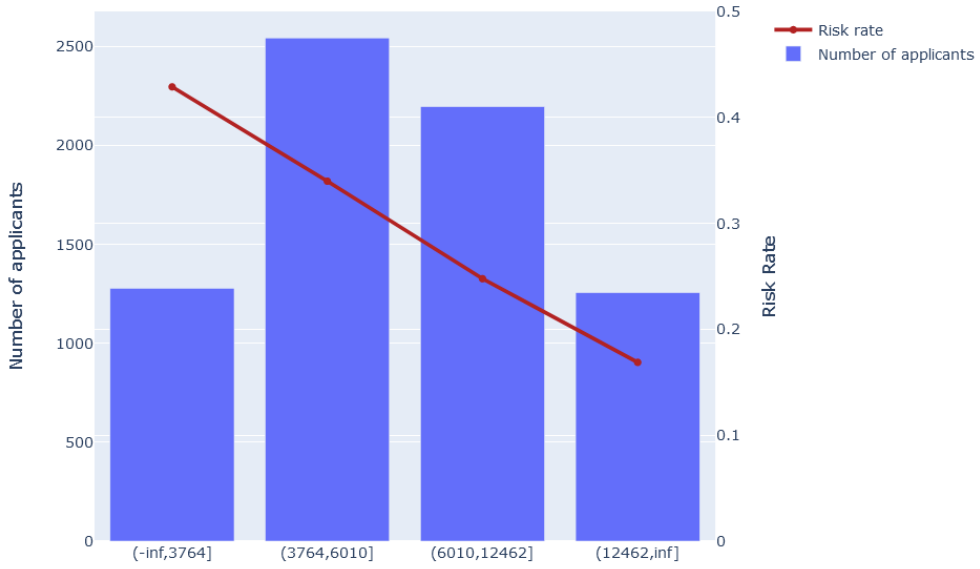


Figure 70: X_1 analysis

- X_1 : increasing the variable lowers the risk

X_2

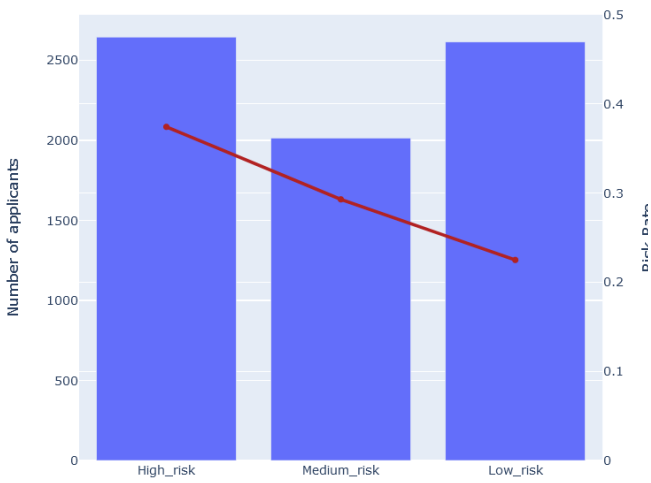


Figure 71: X_2 analysis

- X_2: it's possible to regroup some modalities into High, Medium et Low risk categories

X_4

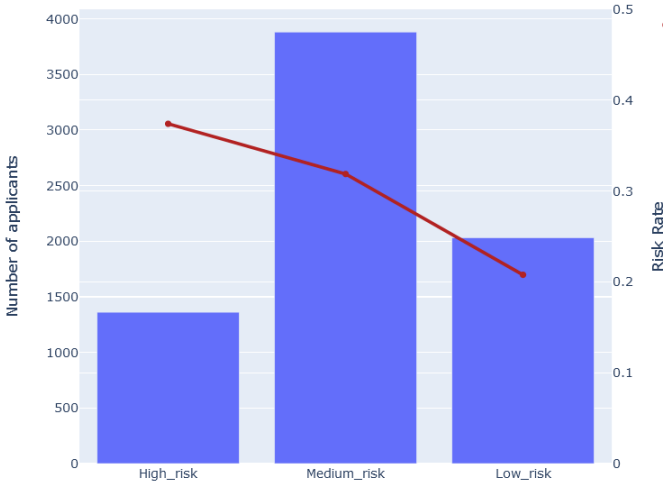


Figure 72: X_4 analysis

- X_4: it's possible to regroup some modalities into High, Medium et Low risk categories

X_6

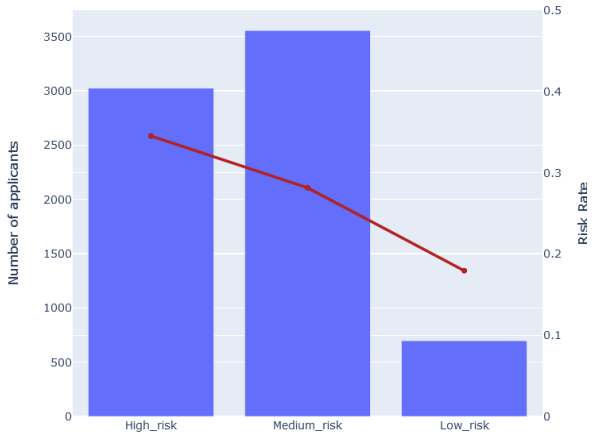


Figure 73: X_6 analysis

- X_6: it's possible to regroup some modalities into High, Medium et Low risk categories

X_7

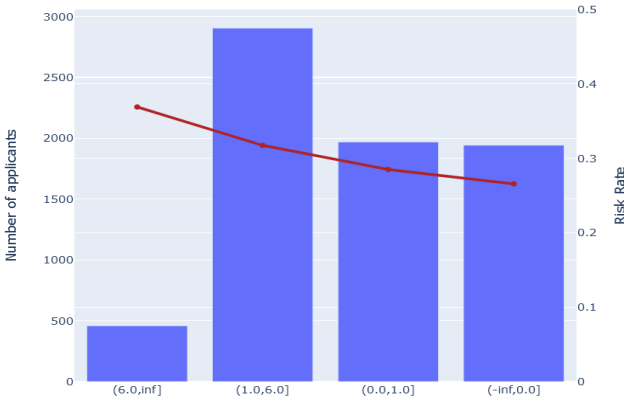


Figure 74: X_7 analysis

- X_7: lowering the value lowers the risk

X_8

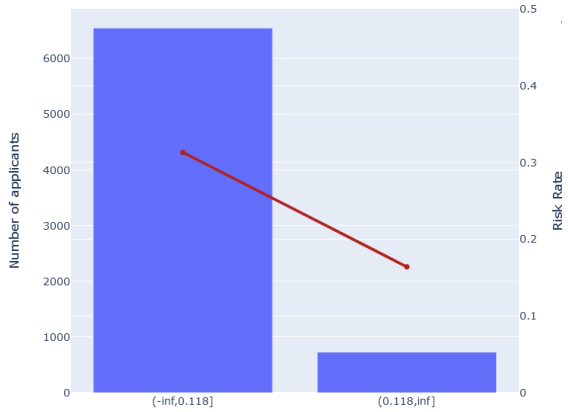


Figure 75: X_8 analysis

- X_8: increasing the value lowers the risk

X_3

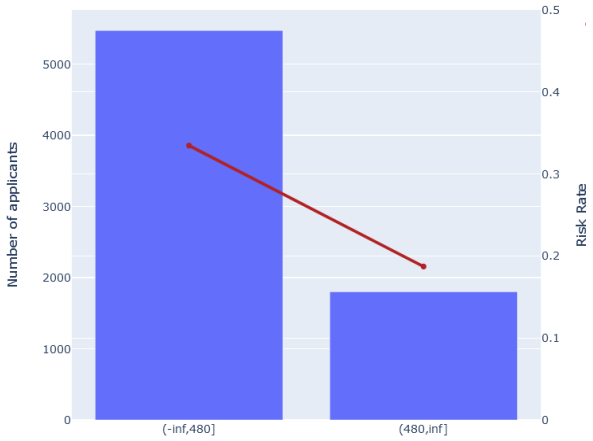


Figure 76: X_3 analysis

- X_3: increasing the value lowers the risk

It's important to check if the selected variables aren't correlated (>0.5) by looking at the Cramer V matrix:

	X_1	X_2	X_4	X_3	X_6	X_7	X_8
X_1	1	0.15	0.11	**0.31**	0.08	0.08	0.19
X_2		1	0.11	0.16	0.12	0.06	0.13
X_4			1	0.19	0.12	0.05	0.14
X_3				1	0.22	0.11	0.20
X_6					1	0.07	0.16
X_7						1	0.08
X_8							1

Figure 77: Cramer V matrix analysis

The highest correlation is observed between the variables "X_1" and "X_3" at 0.31 (< 0.5).

We can affirm that the selected variables aren't highly correlated, based on the threshold set by the Scoring Center (0.5).

5. Example

To clarify the way of using this model, here is a quick example on how to use it.

Let’s create a fake applicant named John:

- X_1 = 4 023
- X_2 = High risk
- X_4 = Medium risk
- X_3 = 495
- X_6 = High risk
- X_7 = 5
- X_8 = 0,1

Using the above scorecard, John will obtain the following score:

John Score = 49 + 39 + 45 + 50 + 39 + 52 + 39 = 313

Figure 78: John’s score

This score puts him in a risky category (with a risk rate of 35.9%). Which is 6 points more than the average risk rate in the data set.

The entity will have to determine a table to define what action to take regarding John’s application.

6. Conclusion

We were able to build a Logistical Regression model to attribute a score to an application based on the following granting information of the applicant:

- The gross income
- The employment types
- The sector of employment
- The expense for the transports
- The code of the product
- The number of "credit report" made to Credit Bureau
- The proportion of his/her allocated to the rent

We obtained a good model as we can see on its Gini curve:

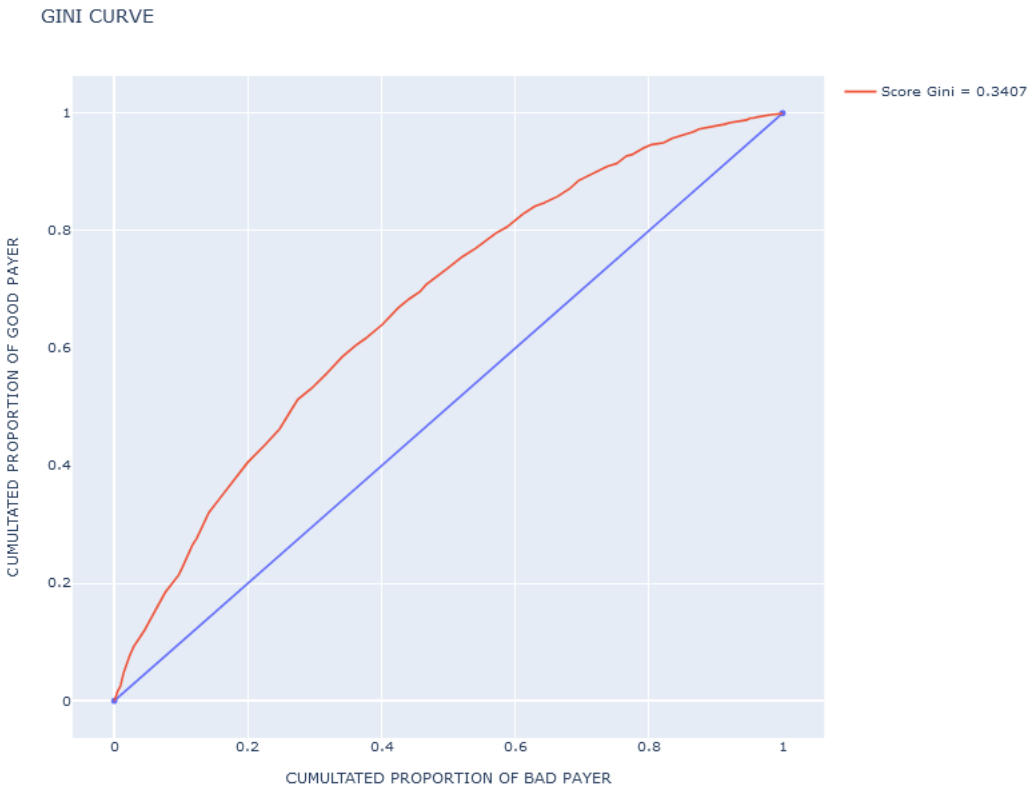


Figure 79: Final model Gini

The red line is detached from the blue one. So, it means that our model is discriminating.

Another good metric to evaluate our model is the "max Kolmogorov–Smirnov test". It compares the Good Debt and Bad Debt cumulative distributions and returns the maximum difference between them.

Score	Bad Debt – Good Debt
(272.999, 303.0]	10.86%
(303.0, 310.0]	18.63%
(310.0, 315.0]	21.52%
(315.0, 320.0]	**24.42%**
(320.0, 325.0]	24.24%
(325.0, 330.0]	22.92%
(330.0, 336.0]	20.47%
(336.0, 343.0]	16.24%
(343.0, 354.0]	9.8%
(354.0, 391.0]	0.0%

Figure 80: Final model Kolmogorov–Smirnov test

In our case, we can see that the highest difference is for score bin (315.0, 320.0] with a value of 24.42%.

Although we have some good metrics to evaluate our model, there is a some very important weakness.

The first one is due to the low volume scenario. We are working without any validation and/or test set. The consequence is that it's difficult to evaluate the final model performances.

Then we don’t know what to do for unknown categories. Once we grouped the known modalities, where should we put the unknown ones? The usual way to deal with this in the Scoring Center is to attribute the new unknown modality to the riskiest group. But this method lowers the model efficiency.

Finally, the last one is the time robustness. Because we are working within a very short period (data collected on less than a year) we don’t know the robustness of the model and stability of the variables through time. It could be very interesting to obtain an out of time sample to evaluate our model on an unknown period.

The only way to solve these problems would be to increase the size of the data. But it will be time consuming to collect more information. And the entity is in the need of a scorecard.

The next step for this project is to being reviewed by the higher instances and other members of the mission in order to be validated or not.

General conclusion

The first part of my apprenticeship, on the categorical encoders, was very interesting. I discovered a lot of new techniques on how to deal with the categorical variables. Even if, during this project, it was sometime a bit rough to deal with the apprenticeship rhythm (being at the school with my friends then working alone on my side at the company for days), I'm happy I developed my research and autonomous skills.

The second part on the RCS project was probably the one I liked the most. Working on a real business project with a real business goal was very nice. I also learned a lot on the methodology of a business case and how to document your work. It was what I expected from this apprenticeship year.

In the end, I really enjoyed doing this apprenticeship at BNP Paribas PF. During this year, I discovered a lot of new techniques and developed some skills through my works. I applied knowledges I obtained at the apprenticeship to some school projects or personal projects. And vice versa. Thanks to the research I did on the categorical encoding techniques, I was able to obtain better results and gained time on a school project. It was my first experience in a large company.

I'm very grateful toward the other members of the team. I learned a lot from them and hope that my work will be useful for them.

Annexes

VICTOR FEUGA

DATA SCIENTIST / DATA ANALYST

SUMMARY

Soon to graduate with an engineering degree (equivalent to a M2) in Computer Science at CY Tech specialized in Artificial Intelligence, **I am looking for a job in the field of Data Science / Data Analysis.**
But I'm open to any other opportunity.
I thrive to **discover new cultures** and **living aboard**.

CONTACT

Phone number : (+33) 6 81 60 71 24
E-mail : vicfeuga@orange.fr
Address : 80 quai des Queyries, 33100 Bordeaux, France
: linkedin.com/in/victor-feuga/
: github.com/vicfeuga
: huggingface.co/vicfeuga

HARD SKILLS

Artificial Intelligence (Deep Learning, Image processing, Transformers, Reinforcement Learning)
Programming (Python, streamlit)
Data Handling (Python x Pandas, SQL)
Data Analysis (Plotly, seaborn)
Statistical and mathematical tools

LANGUAGES

French : Native language
English : Advanced (TOEIC score 970/990)
Spanish : Limited professional competence
Japanese : Beginner

INTERESTS AND HOBBIES

2nd in the Data Challenge 2023 (Creation of a web application for the automated analysis of lithologies using AI)
Winner of IA Pau's Data Challenge 2022 (improvement of an eco-friendly search engine)
Finalist at the Battle Nouvelle Aquitaine 2022 of AI Pau (AI allowing the matching of job offers and jobbers)

Sports : American football, Snowboard, Basketball, Surf, Rugby
Cooking
Analogue photography

EXPERIENCES

Apprentice Data Scientist at BNP Paribas Personal Finance in the Scoring Center
Research and innovation

Mérignac, Nouvelle-Aquitaine, France | September 2022 - September 2023

- Research on different categorical variable encoding techniques for Machine Learning models
- Development of a credit card granting scorecard for an entity in South Africa (logistic regression)
- Development of classification models in Machine Learning
- Data analysis and management

Research internship at the Knowledge System laboratory, Osaka Metropolitan University
Artificial Intelligence in education

Sakai, Japan | May 2022 - August 2022

- Development of an emotion prediction system based on facial action units extracted with the open source application OpenFace.
- Machine Learning, Data analysis
- Research on Computer Supported Collaborative Learning (CSCL) systems and on facial expression of emotions with facial actions

DIPLOMAS

Master degree of Software engineering specialized in AI

CY Tech, Pau, France | September 2020 to September 2023

- Programming and procedural algorithms
- Web development (HTML, CSS, JS, PHP)
- Computer programming (Java, C, SQL, Python, C++)
- Data Exploration
- Artificial intelligence (Deep Learning, Machine Learning)
- 2 interships + 1 apprenticeship

University Diploma of Higher Education in Statistics and Data Analysis

UPPA, Pau, France | September 2018 à June 2020

- Statistics and mathematical tools
- Statistics and business intelligence
- Computer programming (HTML/CSS, JS, Python, SQL, SAS, R)
- Statistical study on the runners in Béarn region
- Statistical study on discrimination on UPPA campuses

References available upon request

Actualized resume