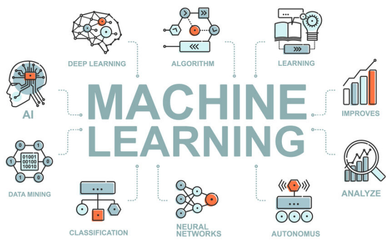
Deep Learning for Insurance and Finance

Final Report



LDATS2310

Godfrin - Siméon

60921800

August 2024

Contents

1 Part 1 : Supervised learning

1.1 Introduction

The goal of the first part of this project is to train a neural network model that accurately predicts whether an individual is likely to have a car accident, given characteristics such as age and income. I first went through the preprocessing of my data. I then search the hyperparameters space through the hyperband algorithm to find the best possible hyperparameters of my neural network. Finally, I analyzed the predictions of my model with the help of different tools. The data presentation is deliberately omitted here, assuming that the reader is already familiar with it.

1.2 Preprocessing

First, I decided to suppress the variable "Claim Amount", which represent the amount of money the assurance should pay to cover the crash. Despite having a moderate Pearson correlation of 0.527 with my variable of interest 'Crash', it was prudent to exclude 'Claim Amount' to prevent data leakage. When taken into account, my model reached $99, ..\%$ accuracy on unseen data. I also extracted the month of birth from the variable "DOB" and then suppressed it since we had the remaining information with the variable "Age".
I converted all categorical variables with two categories into binary variables.

1.2.1 Correlation

I analyzed the correlation between my numerical variables and suppressed variables highly correlated (having 0.6 or greater in absolute value) It results in the suppression of the variable "Red" due to high correlation with the variable "Gender".
I analyzed the correlation between my categorical variable using the Cramer's V technique. It is a measure of the association between two categorical variables based on the Pearson's chi-squared statistics. I found out that my variable "Education" and "Occupation" were highly correlated (0.64) and decided to suppress the variable "Occupation".

1.2.2 Dummification

I dummified my variables "Education", "Month" and "Car Type" into k - 1 categories because machine learning model require numerical inputs. I wanted to keep the ordinal relationship within the variable "Education" but lacked the time to re train models.

1.2.3 Scaling

I scaled my variable which had a variance greater than 2 using a standard scaler.

1.2.4 Balancing the dataset

The dataset is highly imbalanced, there is only 26.84% of crash (class 1). It could cause the model to have a bias toward class 0, e.g. simply only predict it and still reach $73, 16\%$ accuracy. To solve this issue and create a balanced dataset, I tried 3 different techniques : SMOTE, equivalent balancing,random oversampling.
What I call "equivalent balancing" is just keeping the same proportion within both training and testing set. This methods didn't perform well and it makes sense : I am not exacerbating the capacity of my model that recognizes the minority class.
The random oversampling technique is just randomly picking points from the minority class and adding them to the existing ones.
The SMOTE technique gives the best results and stands for Synthetic Minority Oversampling TEchnique. It consists of creating new points from the minority class that are similar but not identical. It use the k-nearest

neighbors of a minority point, select one among them and randomly put a new point in the line between the initial point from the minority class and this random point in its neighbourhood.
I also implemented class's weights to mitigate the effect of the unbalancing by according more importance to the minority class. They were computed using the inverse class frequency method.

1.3 Neural Network models

Neural networks offer significant flexibility in architecture design, including choices in the number and types of layers, the number of neurons per layer, activation functions, optimizers, and loss functions.
For all my neural network, I used a binomial deviance/loss function and hyperbolic tangent as the activation function because I find it the best performing and most stable for the gradient.
The activation function introduces non-linearity to the neural network, enabling it to model complex patterns. Without it, the neurons would simply perform linear regression.
To avoid overfitting, I enabled the model tuner to add dropout layers, which randomly deactivate a portion of the weights in the input layer during training. Dropouts force the neural network to learn an ensemble of different models. During each iteration, different neurons are deactivated, requiring the network to find multiple internal pathways to process the same information, thus capturing deeper representations. The model can't rely solely on information that it learned on previous iterations. Also, since some neurons are turned off, the number of weights to compute is reduced.

I initialized the weights of each layer using a normal distribution, which led to better results and improved gradient stability.
The vanishing or exploding gradient problem arises during backpropagation when the derivatives become progressively smaller or larger as they propagate backward through the layers, making training difficult or impossible.
The backpropagation is applying the chain rule to decompose the gradient into the product of two gradients: the gradient of the loss with respect to the layer's output, and the gradient of the output with respect to the weighted sum of inputs.

1.3.1 The hyperband algorithm

The goal of this algorithm is to find the best architecture for a neural network in the enormous space parameter. It's an hyper parameter optimization tool. The Hyperband algorithm optimizes hyperparameters by maximizing the use of available resources, specifically the number of iterations, to explore the parameter space efficiently. After having defined the parameter space, the algorithm will randomly try for a limited number of iteration configurations, using an early-stopping strategy, and will only keep the best performing one. It reallocate the iteration of the least performing half to the better-performing configurations. This process, known as successive halving, is repeated until the algorithm convergence toward the best configuration of parameters.

1.3.2 Models found

The best-performing model was not the one with the lowest deviance. I chose Model 2 because it exhibited a better Balanced Classification Rate (BCR) while maintaining a deviance nearly identical to the other models.
The BCR, or balanced classification rate, is basically the average accuracy of the model for both classes, i.e. the average between the true positive rate and the true negative rate.
Given that their deviances are almost identical, I consider BCR to be a more relevant criterion for evaluating the performance of a model designed to predict classes in an imbalanced dataset framework.
The following table shows the performance of each model run 15 times.

Models	**Deviance**	**Accuracy**	**BCR**
Model 1	5 560 152.5	0.7575	0.7232
Model 2	5 587 603.0	0.7386	0.7349
Model 3	6 733 978.5	0.7357	0.6949
Model 4	7 008 550.0	0.7446	0.6959

Table 1: Comparison of Models Based on Deviance, Accuracy, and BCR

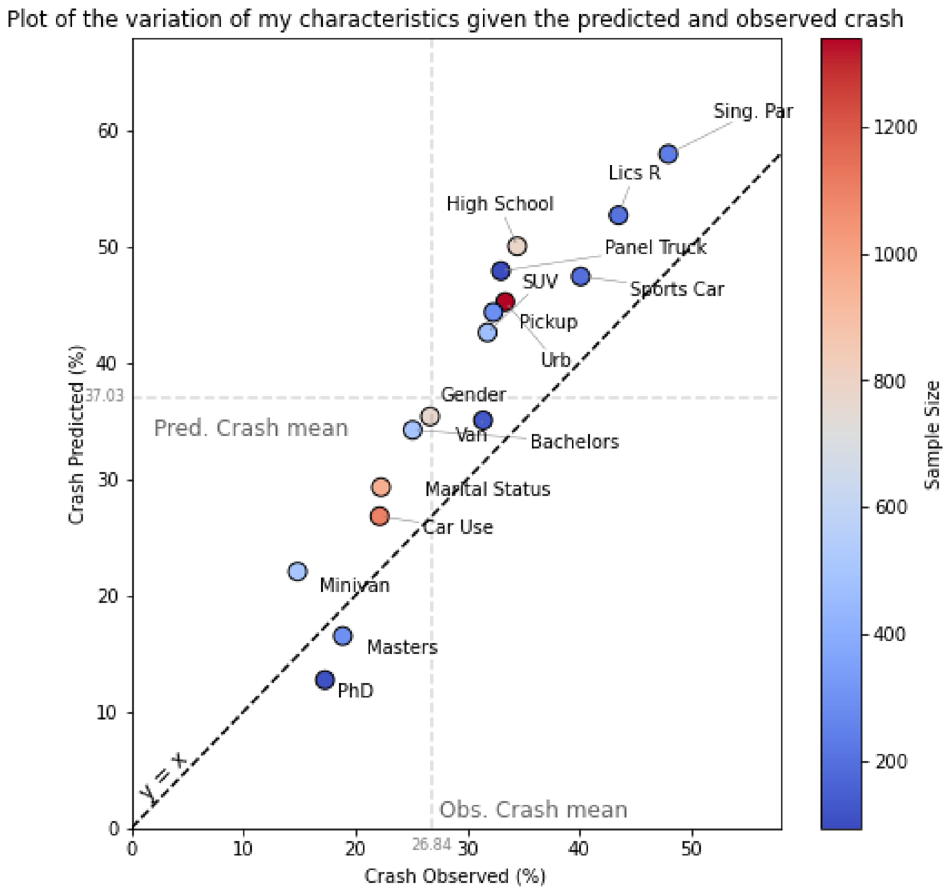
My best model consists of the following architecture (every layer has a *tanh* activation function and initialize its weights from a normal distribution). See the others model in the appendix.

- First Layer: 208 units,
- Second Dense Layer: 112 units
- Dropout Layer: 50% dropout
- Third Dense Layer: 128 units
- Fourth Dense Layer: 240 units
- Fifth Dense Layer: 192 units
- Dropout Layer: 30% dropout
- Sixth Dense Layer: 16 units
- Output Layer, *sigmoid* activation
- Adam optimizer with *learning rate* of 0.001

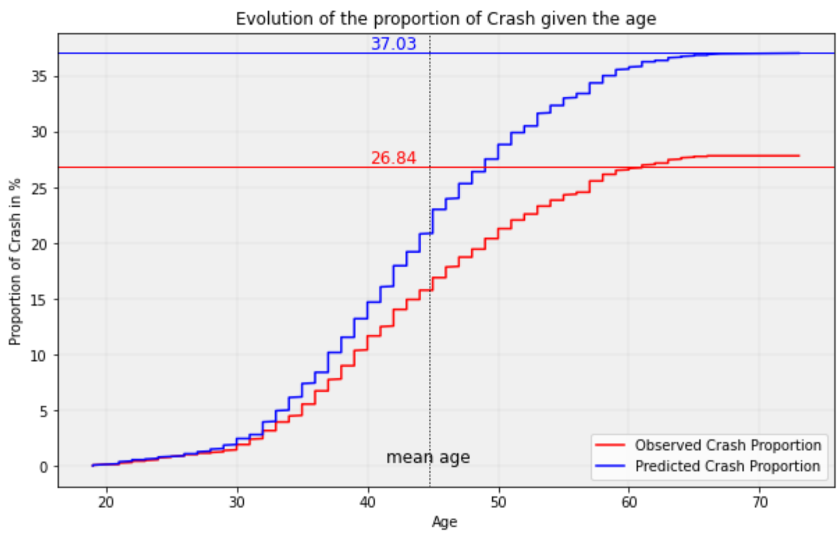
1.3.3 Model analysis

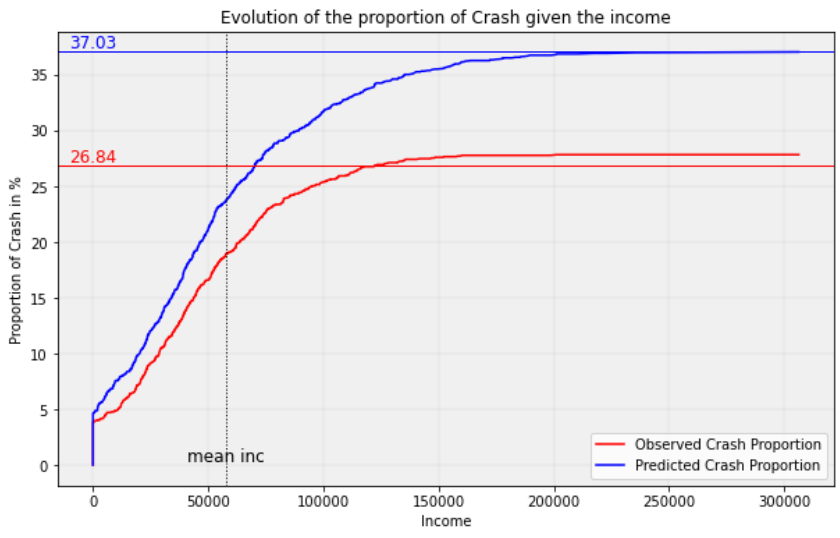
Model 2 predicts more crashes than actually occurred, with a prediction rate of 37.03% compared to the actual rate of 26.84% in the dataset. These predictions are represented by the gray dashed line in the plot.

The following graph illustrates the variation in crash proportions, both observed and predicted, across different categorical levels. For instance, the point 'PhD' being below the dashed line means that for all the individuals having a PhD in my testing set, my model predicts less crash than there actually are for this category. But it's the least represented level of the variable *Education*, as indicate its color. It can reflect the model performance relative to the amount of data available for each category.



Out of the 16 characteristics tested, 14 are above the dashed line. For these points, the model made an average error of 7.82%. The error is represented by the distance between the points and the dashed line on the graph. The dashed line represent the perfect scenario where my model would predict the perfect proportions
However, the model does not make significant errors across the characteristics. We can see it graphically by the absence of points in the lower left and upper right rectangle. Those rectangles indicate area where my predictions would be far away from my observations.
My model predicts well particularly well the categories *PhD, Masters, Van* and *Car Use*.
Overall, the model shows systematic overestimation and the sample size doesn't seem to have an impact on the performance of the model.
The trend is the same given any variable. The following two graphs shows the variation of proportion of crash given, respectively, the *age* and the *income*. They also underline the tendency of the model.





1.4 LIME analysis

LIME (Local Interpretable Model-Agnostic Explanations) is an algorithm designed to explain the predictions of a classifier by approximating its behavior locally with an interpretable model. The algorithm tries to solve the "black box" (interpretability) problem of some machine learning such as the neural networks.
I'll be analyzing four types of points misclassified points :

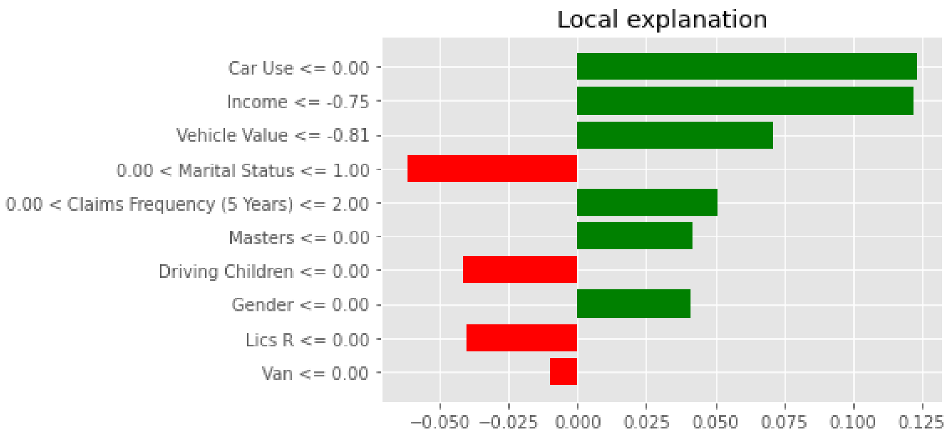
- Type 1 : class 0 $\rightarrow$ pred. as 1 with confidence
- Type 2 : class 0 $\rightarrow$ pred. as class 1 without confidence
- Type 3 : class 1 $\rightarrow$ pred. as 0 with confidence
- Type 4 : class 1 $\rightarrow$ pred. as class 0 without confidence

Confidence refers to the value of the model's output. If the value falls within $[0.43, 0.57]$, the prediction is considered to be made without confidence. The lime plots that follows will only takes into consideration the 10 most influential features.

1.4.1 Type 1

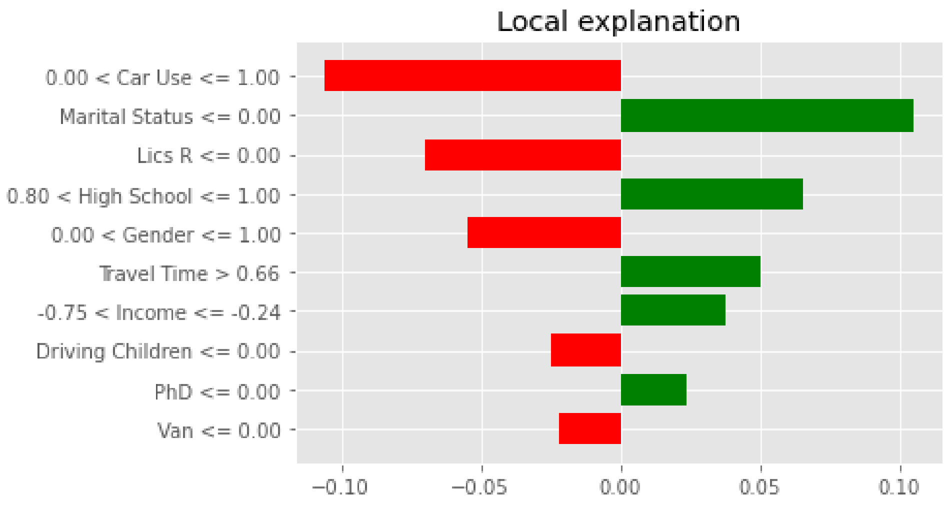
The first point analyzed, individual 133 in my testing set, is a class 0 misclassified as a class 1 and the model was pretty sure of its prediction (0.7831). The most influential variables leading to the prediction of class 1 are

Car Use $== 0$, *Income* ≤ -0.75, and *Vehicle Value* ≤ -0.81. It means that his car were for commercial use. Its Income and vehicle values were just a little bit above the 25 th quartiles, meaning that in the testing sets he's within the 25% of poorest people and with the lowest car value.
Conversely, being married, not driving with children, and not having a revoked license influenced the model toward predicting class 0.
There is 58.17% women in the type 1 and the variable seem to push toward the class 1, the crash, for this individual. 96.15% of the subset population lives in an urban area, which is way above the 80% in the original dataset and is a significant difference.



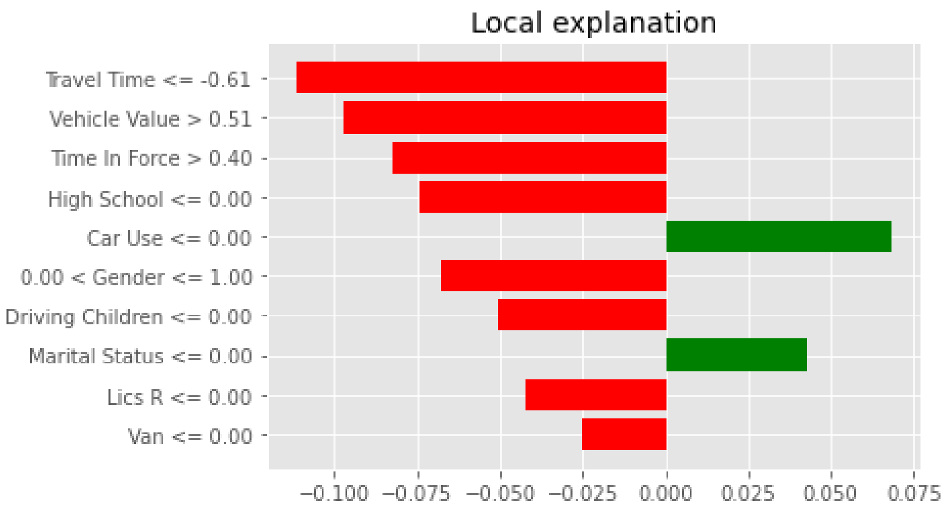
1.4.2 Type 2

The point analyzed is the individual 625 in my testing set. It is class 0 misclassified as class 1 without confidence (0.5192). The following plot is obviously more balanced. We again find *Car Use* and *Marital Status* as the most influential variables but in the other way around : the car is for private use and this person isn't married. The former impact negatively and the latter positively. The variable *High School* have a strong impact toward class 1. This person's income is just above the 25th percentile, indicating that she is slightly wealthier than 25% of individuals in the training set. The variable travel time had a strong positive impact, and its value lies above the 75 percentile across all my dataset, meaning its travel time is among the highest.
The subset of type 2 also have a statistically significant difference in the proportion of person coming from an Urban environment, 94.44% against 79.19% in the whole dataset.



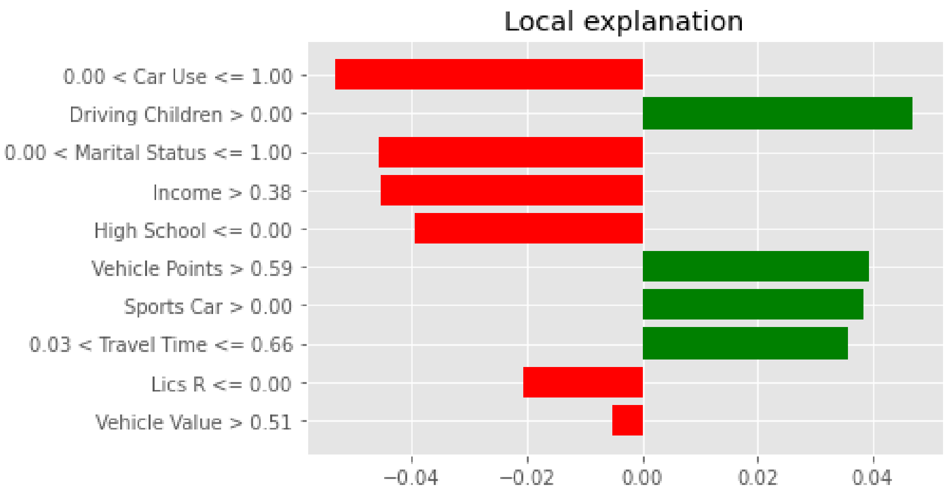
1.4.3 Type 3

The point analyzed is the individual 14 in my testing set. It is class 1 misclassified as class 0 with (high) confidence (0.06) by my model. Four variables have a strong negative impact as you can see in the following plot. We again find the variable *Travel Time*, but the other way around, the travel time of this individual is in the 14 th percentile, meaning that its among the top 76% of the shortest travel time. Intuitively, the finding of my model seem to make sense : a shorter travel time means less chance of crash. The same goes for the Vehicle Value : an more expansive car could mean a safer car. Again, not being married seem to push toward having a crash, according to the model for this specific example, which I find funny. The variable *High School*$=$ 0 arises again to push toward the class 0.



1.4.4 Type 4

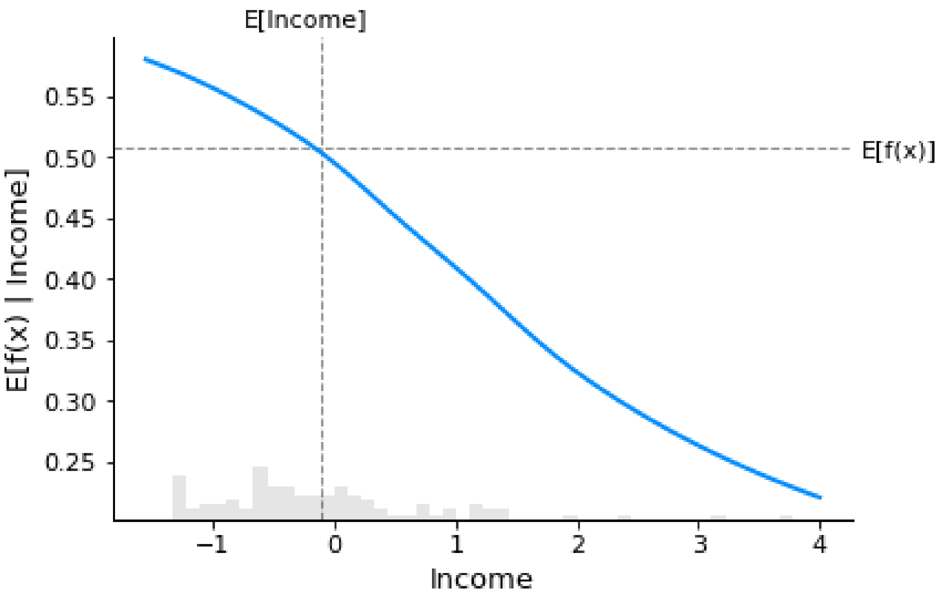
The point analyzed is the individual 467 in my testing set. It is a class 1 misclassified as class 0 without confidence (0.49). Using its car for private usage, being married, having a relatively high income within the data set and not being in high school push toward the class 0. On the opposite, driving with children, having a sport car and a relatively long travel time push toward the class 1. These interactions intuitively make sense, except for the variable *Driving Children*, which unexpectedly has a positive impact on the prediction.



1.5 Shapley's value

Before delving into the interpretation of the SHAP values, it's important to acknowledge that these results should be viewed with caution. The analysis presented here is based on a limited subsample of 100 data points (due to computational limits) and focuses on a single individual. While SHAP values offer valuable insights into the model's behavior by explaining how different features contribute to its predictions, the findings in this case may not be fully representative of the broader population. As such, these results should not be taken too literally.

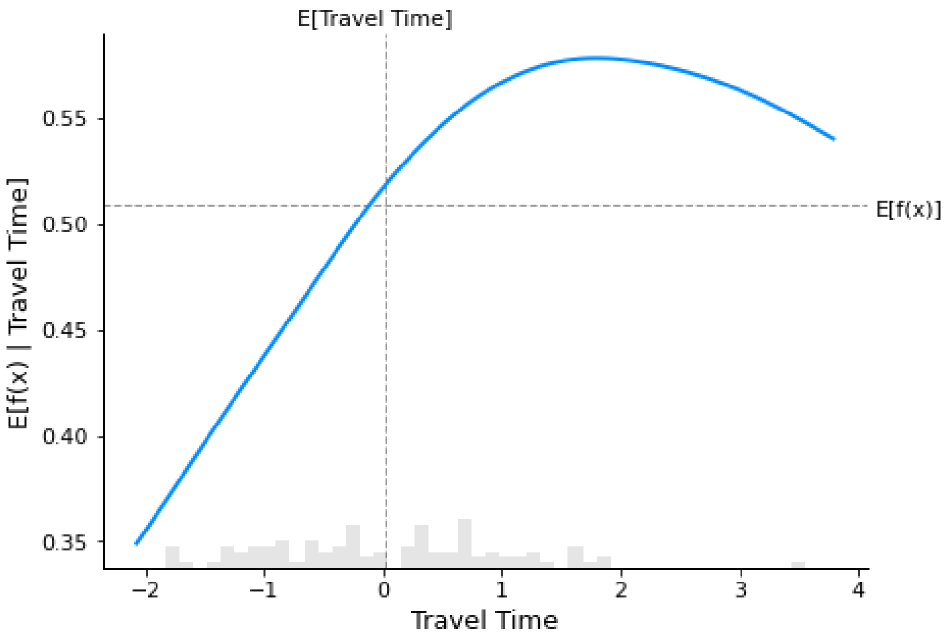
The partial depence plot of *Income* shows a negative relation between income and the predicted output. This indicates that individuals with higher incomes are less likely to be predicted as being involved in a crash (class 1). Instead, higher income levels push the prediction toward the 'no crash' class (class 0)."
This result makes total sense and shows that our model has a grasp of the "reality." This result aligns with real-world observations, where individuals with higher incomes might have access to safer vehicles, better driving conditions, or more resources to avoid risky situations, thereby reducing their likelihood of being involved in a crash.
Furthermore, this finding is consistent with the insights gained from the LIME analysis, where income was also shown to be a significant factor in predicting crash risk. The consistency across different interpretability techniques—SHAP and LIME—reinforces our confidence that the model has correctly learned the relationship between income and crash risk.



The shapley partial dependence plot of *Travel Time* shows us an interesting relation : for low travel times (around -2 to 0), the plot shows an upward trend, indicating that as "Travel Time" increases the predicted probability of a crash also increases. This suggests that, initially, longer travel times are associated with a higher risk of a crash.

Then we reach a peak where the predicted crash probability is the highest. For this individual, the travel time's value between 1 and 2 represent the maximum risk to have a crash.

Beyond this peak, as the variable keep increasing, the probability suprisingly begins to curb. An explanation could be that after a certain distance, the individuals reach an highwaty and the risk are reduced. Conversely, the initial increase might reflect urban driving scenarios where moderate travel times are associated with higher risk.



1.6 PDP and ICE

I was unable to get KerasRegressor to work properply which prevented me from performing PDP and ICE. Additionally, I encountered issues with Shapley's values and did not have time to resolve them

2 Part 2 : Unsupervied learning

2.1 Preprocessing

This part of the project required less preprocessing.
I started by categorizing the variable *Age* into 4 categories based on common sense rather than the percentiles of the distribution. The first category, if using the latter approach to have an equal number of individuals per category, would look like the following: $[0 \to 39]$. Below is how I categorized the variable age:

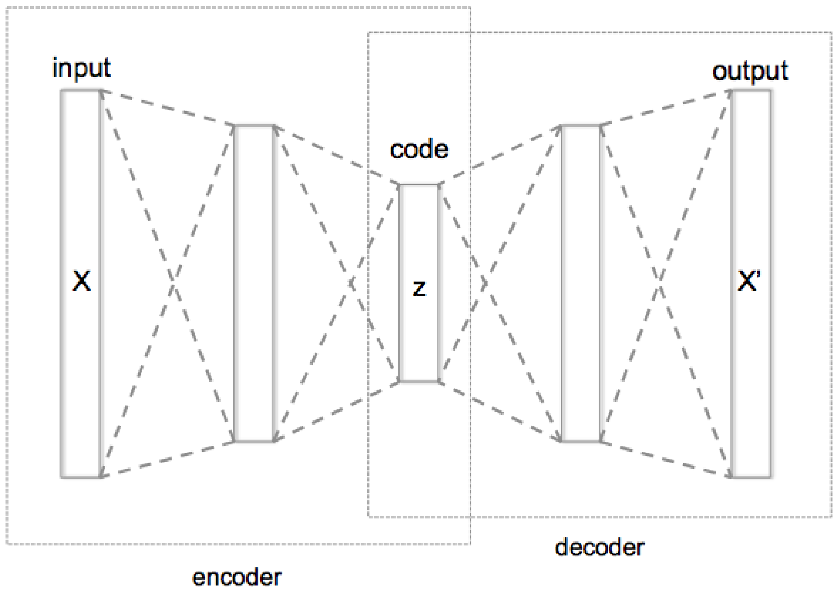
- *Young Adults*: $[0 \to 30]$
- *Early Middle Age*: $]30 \to 40]$
- *Late Middle Age*: $]40 \to 50]$
- *Older Adults*: $]50 \to \infty]$

I analyzed the correlation using Cramer's V technique. The pair *Education* and *Occupation*,*Car Type* and *Gender* and *Occupation* and *Car Use* are correlated at respectively 0.64, 0.71 and 0.57. Despite these correlations, I decided to retain all variables because each provides unique information. Additionally, their correlations, while moderate, are not excessively high.I also considered using the Correlated Variational Auto encoder to further explore these relationships, but I lacked the time to do so.

I dummified or transformed into a binary format all my categorical variables to put them in a numerical form that machine learning algorithm understand. The final step of my preprocessing was creating my training set by splitting my data into two part.

2.2 Variational auto-encoder

The variational auto-encoder (VAE) is a generative model, which is a special type of neural network architecture. The VAE extends the concepts of auto-encoders by introducing a probabilistic approach to learning the latent (a reduced) representations of the data. These models are built based on two analogous parts with "opposite" goal that are in contact with each other through the latent space.
On one hand, the encoder's mission is to compress the input data into a lower-dimensional latent space, while simultaneously learning the distribution of the data.
On the other hand, the decoder works in reverse. His goal is to learn how to generate data that is as close to the original inputs as possible, starting from points in the latent space.
The reduced-dimensional representation, referred to as the latent space, serves as a compact, abstract encoding of the input data. The encoder approximates the distribution of the latent variables, which are assumed to follow a normal distribution.
The loss function used to train my model combines a weighted MSE reconstruction loss with KL divergence measure. The former is used to gauge the difference between the output and the input and the latter mesure the difference between the learned distribution of latent variables and a Gaussian distribution.// The KL divergence term in the loss function is closely related to the maximization of the Evidence Lower Bound (ELBO). The ELBO serves as a tractable approximation for the marginal log-likelihood of the data, which is generally intractable to compute directly. By maximizing the ELBO during training, we are indirectly maximizing the log-likelihood of the observed data under the model. This ensures that the VAE is learning a representation of the data that not only reconstructs it well but also adheres to a structured latent space aligned with the prior distribution.



2.2.1 Results of my models

I trained four models using the hyperband tuner. My hyper parameters were the number of layers, the number of neurons per layer, their activation function and the latent dimension. There is two types of model based on two architecture, one based on a simpler model (model 1 and 3) and the other which include dropout layers to prevent overfitting.(model 2 and 4)
To be variational, the VAE process of reconstruction process is a bit peculiar. The model introduce a new variable, ϵ, which is drawn from $\sim \mathcal{N}(0,1)$. It uses this variable to create random new point by adding to the latent variable's mean, z , a proportion, ϵ, of z's standard deviation. The generation is as follow : $z = \mu_z + \epsilon * \sigma_z)$.
The table below represents the mean reconstruction loss, its standard deviation over 20 iterations, and the latent dimension for each model. The best model is the model 4. It has both the lowest reconstruction loss and standard

Models	**Reconstruction loss**	**stand. dev.**	lat. dim.
Model 1	13988.58	5221.92	16
Model 2	13255.11	1870.31	8
Model 3	14053.1	6184.24	18
Model 4	11715.08	1941.7	12

Table 2: Comparison of Models Based on mean reconstruction loss and its standard deviation

deviation. We can see that the model with dropout layer have better performance and smaller variance in their results. The model 4 has the following architecture.

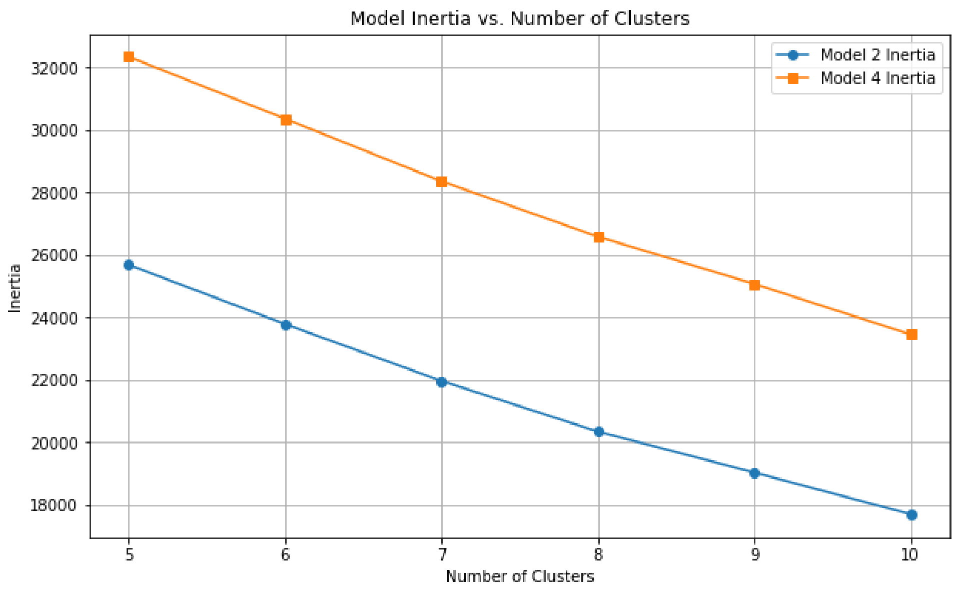
- Input Layer: Input shape (23)
- First Dense Layer: 192 units, *selu* activation
- Dropout Layer: 50% dropout
- Second Dense Layer: 256 units, *selu* activation
- Latent Layer (z_mean): 12 units
- Latent Layer (z_log_var): 12 units
- Decoder Input: Input shape (12)
- First Dense Layer in Decoder: 256 units, *selu* activation
- Second Dense Layer in Decoder: 128 units, *selu* activation
- Third Dense Layer in Decoder: 128 units, *softplus* activation
- Output Layer: Output shape (nc), *sigmoid* activation
- Adam optimizer

2.3 KMeans

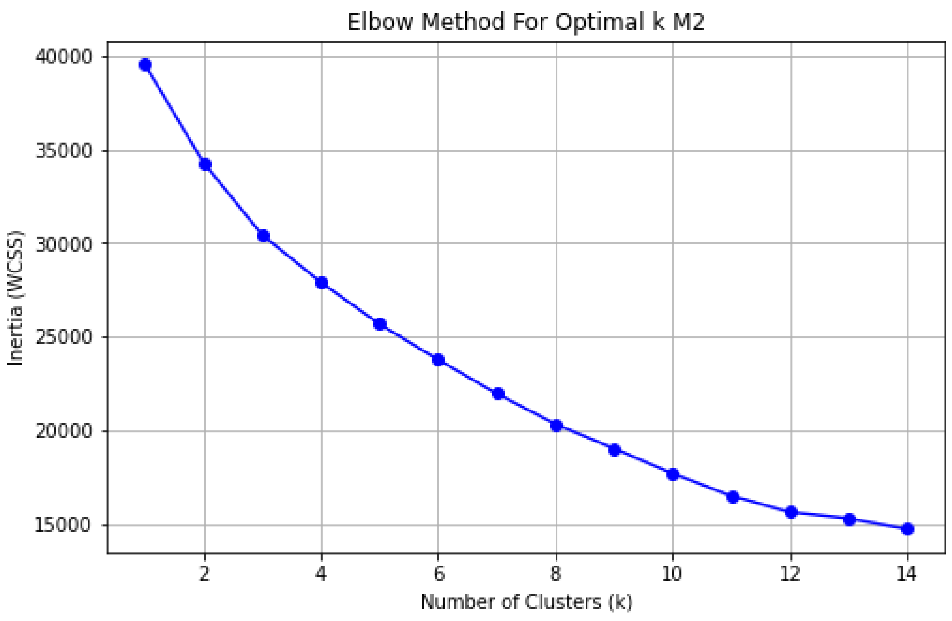
For this part, the previously presented VAE models will be trained on my whole dataset. Then, I will use the encoder to decrease the dimension of my dataset to its latent dimension (12 or 8 given the model). The goal of this dimension reduction is to avoid the curse of dimensionality (complexity and sparsity increase with the number of dimension) and allow my KMeans algorithm to perform in optimal conditions.

The KMeans++ is a clustering algorithm used in unsupervised learning to find and regroup similar points around clusters. The algorithm works by iteratively assigning data points to the nearest cluster center and then updating the cluster centers based on the mean of the points assigned to each cluster. The algorithm works as follows: it starts by randomly selecting the first centroid (cluster center). The remaining centroids are then initialized by selecting data points with a probability proportional to the square of their distance from the nearest existing centroid. After initializing the centroids, the algorithm assigns each data point to the cluster with the closest centroid. The centroids are then recalculated as the mean of all data points assigned to each cluster. This process of assignment and updating of centroids is repeated iteratively until convergence is reached, meaning the centroids no longer change. The final result is a partition of the data into K clusters, where each point is assigned to the nearest centroid. The number of centroids is chosen to minimize the deviance, which I guess it in case is the inertia.

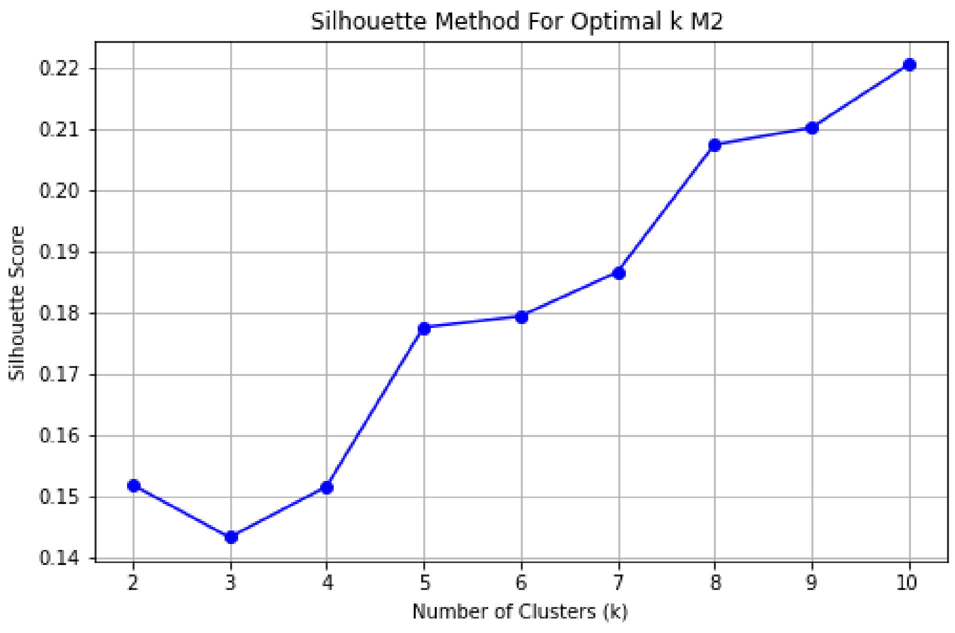
Two out of my 4 models were good candidates to be used to reduce the dimension : model 2 and model 4. Not only that they are the best performing models, but they also lay the smallest latent dimensions. (cf dimensionality curse) To choose the best model among them, I compared the inertia of KMeans algorithm trained on the compressed dataset for each model. The plot below is unanimous : the model 2 gives the best performances.



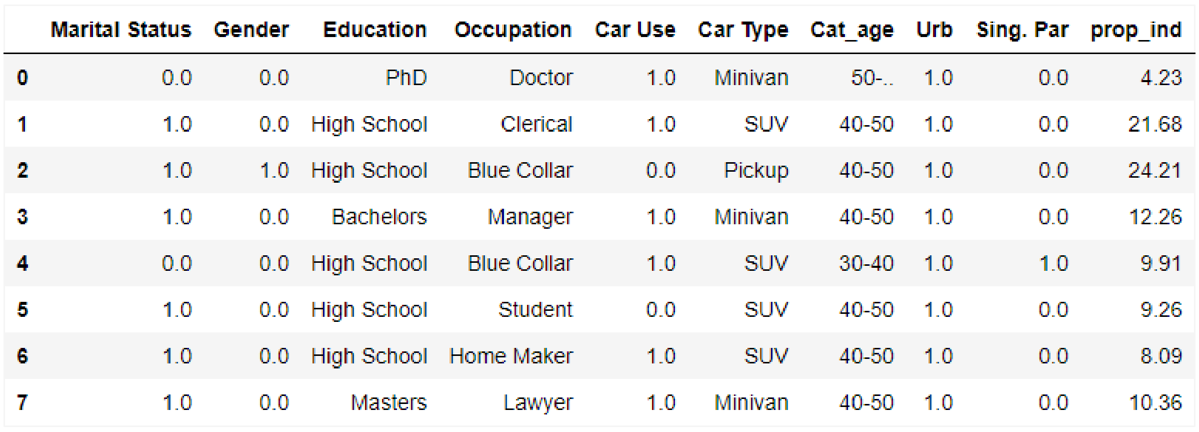
The next crucial step is choosing the optimal number of clusters. To accomplish this, I employed the elbow graphical technique. It is used to identify the optimal number of clusters in KMeans clustering. The method involves plotting the inertia (or within-cluster sum of squares) against the number of clusters. The optimal number of clusters is found at the ’elbow point’ of the plot, where the rate of decrease in inertia slows. From the following graph, we can see that the tipping is between 7 and 10 clusters.



I also used a silhouette plot to help me out find the best number of clusters. The silhouette score is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). It ranges from -1 to 1, where a higher score indicates that the object is well-matched to its own cluster and poorly matched to neighboring clusters. The following plot identify 2 interesting plateau. One between5 and 7 clusters and the other between 8 and 10 clusters. I chose 7 clusters.



Finally, I identify for each cluster a profile. Each line of the in table below for each variable represent the most occurring level.
My model and algorithm seem to have done a correct job because the profiles are well balanced in term of proportion of individuals (*prop ind*). We can identify one profile of *male* (represented by 1) and of *Single Parent*, which are two minority levels, every level of *Education* and almost every level of *Occupation*.



	Marital Status	Gender	Education	Occupation	Car Use	Car Type	Cat_age	Urb	Sing. Par	prop_ind
0	0.0	0.0	PhD	Doctor	1.0	Minivan	50-..	1.0	0.0	4.23
1	1.0	0.0	High School	Clerical	1.0	SUV	40-50	1.0	0.0	21.68
2	1.0	1.0	High School	Blue Collar	0.0	Pickup	40-50	1.0	0.0	24.21
3	1.0	0.0	Bachelors	Manager	1.0	Minivan	40-50	1.0	0.0	12.26
4	0.0	0.0	High School	Blue Collar	1.0	SUV	30-40	1.0	1.0	9.91
5	1.0	0.0	High School	Student	0.0	SUV	40-50	1.0	0.0	9.26
6	1.0	0.0	High School	Home Maker	1.0	SUV	40-50	1.0	0.0	8.09
7	1.0	0.0	Masters	Lawyer	1.0	Minivan	40-50	1.0	0.0	10.36

3 Appendix

3.1 Remark and side notes

This work has been done with the help of chatGPT4.

I know that I'm not using the function smote() in the optimal way. I should use smotenc() since I have categorical value and dummify after splitting my dataset into training and testing. But surprinsingly enough, this wrong way of doing it, gets better results.

I also know that I have a small dataleake through my standardization. I should split first then standardize. But again, I lacked time.//
At least I am honest about it.

3.2 Neural Networks model

Model 1

- Input Layer: 144 units
- Dense Layer: 112 units
- Output Layer

Model 3

- Input Layer: 128 units
- First Dense Layer: 144 units
- Second Dense Layer: 96 units
- Dropout Layer: 20% dropout
- Third Dense Layer: 224 units
- Dropout Layer: 40% dropout
- Output Layer

Model 4

- Input Layer: 224 units
- First Dense Layer: 144 units
- Second Dense Layer: 144 units
- Third Dense Layer: 240 units
- Output Layer

3.3 VAE models

Model 1

- Input Layer: Input shape (nc)
- First Dense Layer: 224 units, *tanh* activation
- Latent Layer (z_mean): 16 units
- Latent Layer (z_log_var): 16 units
- Decoder Input: Input shape (16)
- First Dense Layer in Decoder: 208 units, *selu* activation
- Second Dense Layer in Decoder: 128 units, *relu* activation
- Third Dense Layer in Decoder: 192 units, *relu* activation
- Fourth Dense Layer in Decoder: 80 units, *softplus* activation

- Output Layer: Output shape (nc), *tanh* activation
- Adam optimizer

Model 2

- Input Layer: Input shape (nc)
- First Dense Layer: 256 units, *tanh* activation
- Dropout Layer: 30% dropout
- Latent Layer (z_mean): 8 units
- Latent Layer (z_log_var): 8 units
- Decoder Input: Input shape (8)
- First Dense Layer in Decoder: 32 units, *tanh* activation
- Second Dense Layer in Decoder: 224 units, *tanh* activation
- Third Dense Layer in Decoder: 256 units, *softplus* activation
- Output Layer: Output shape (nc), *sigmoid* activation
- RMSprop optimizer

Model 3

- Input Layer: Input shape (nc)
- First Dense Layer: 240 units, *selu* activation
- Second Dense Layer: 160 units, *tanh* activation
- Third Dense Layer: 160 units, *tanh* activation
- Latent Layer (z_mean): 18 units
- Latent Layer (z_log_var): 18 units
- Decoder Input: Input shape (18)
- First Dense Layer in Decoder: 48 units, *softplus* activation
- Second Dense Layer in Decoder: 48 units, *tanh* activation
- Third Dense Layer in Decoder: 208 units, *tanh* activation
- Fourth Dense Layer in Decoder: 128 units, *relu* activation
- Output Layer: Output shape (nc), *sigmoid* activation
- Adam optimizer

3.4 Sources

Wikipedia. (n.d.). Cramér's V. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Cram%C3%A9r%27s_V

Singh, M. (2019, August 25). Understanding categorical correlations with chi-square test and Cramér's V. Medium. Retrieved from https://medium.com/@manindersingh120996/understanding-categorical-correlations-with-chi-square-test-and-cramers-v-a54fe153b1d6

CoderzColumn. (n.d.). How to use LIME to understand sklearn model predictions. CoderzColumn. Retrieved from https://coderzcolumn.com/tutorials/machine-learning/how-to-use-lime-to-understand-sklearn-models-predictions

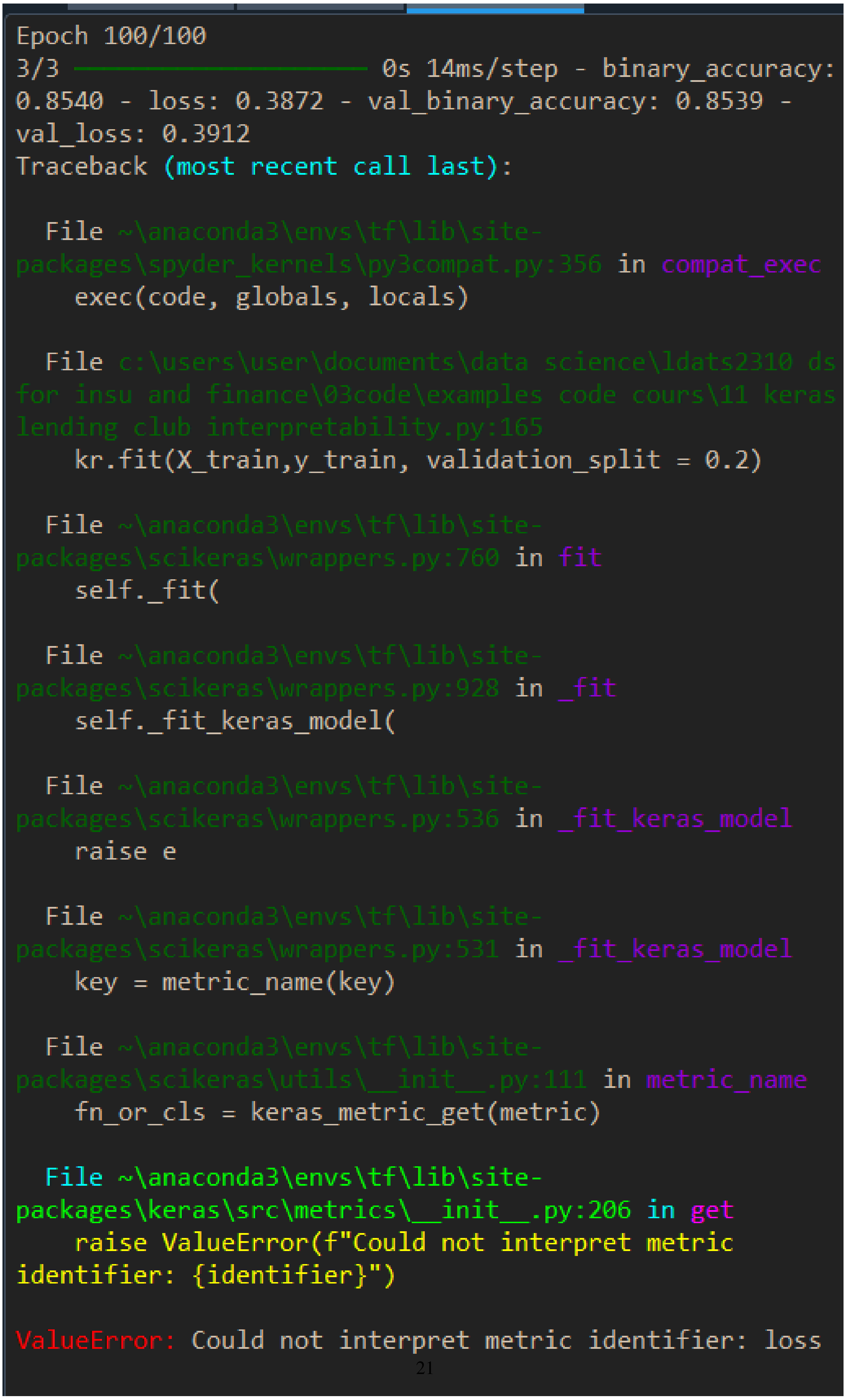
Kobia. (n.d.). Imbalanced data: SMOTE. Kobia. Retrieved from https://kobia.fr/imbalanced-data-smote/

Jamieson, K. (n.d.). Hyperband algorithm. Retrieved from https://homes.cs.washington.edu/~jamieson/hyperband.html#:\protect\protect\unhbox\voidb@x\hbox{~:text=Hyperband%20Algorithm,a%20small%20number%20of%20iterations}

IBM. (n.d.). Variational autoencoder (VAE). IBM. Retrieved from https://www.ibm.com/think/topics/variational-autoencoder

3.5 Problem encounter

I could not make the "KerasRegressor" function work, even in your code and I didn’t had the time to solve the issue.
The error message is as follow ;
“ValueError : Could not interpret metric identifier : loss”



```
Epoch 100/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - binary_accuracy:
0.8540 - loss: 0.3872 - val_binary_accuracy: 0.8539 -
val_loss: 0.3912
Traceback (most recent call last):

  File ~\anaconda3\envs\tf\lib\site-
packages\spyder_kernels\py3compat.py:356 in compat_exec
    exec(code, globals, locals)

  File c:\users\user\documents\data science\ldats2310 ds
for insu and finance\03code\examples code cours\11 keras
lending club interpretability.py:165
    kr.fit(X_train,y_train, validation_split = 0.2)

  File ~\anaconda3\envs\tf\lib\site-
packages\scikeras\wrappers.py:760 in fit
    self._fit(

  File ~\anaconda3\envs\tf\lib\site-
packages\scikeras\wrappers.py:928 in _fit
    self._fit_keras_model(

  File ~\anaconda3\envs\tf\lib\site-
packages\scikeras\wrappers.py:536 in _fit_keras_model
    raise e

  File ~\anaconda3\envs\tf\lib\site-
packages\scikeras\wrappers.py:531 in _fit_keras_model
    key = metric_name(key)

  File ~\anaconda3\envs\tf\lib\site-
packages\scikeras\utils\__init__.py:111 in metric_name
    fn_or_cls = keras_metric_get(metric)

  File ~\anaconda3\envs\tf\lib\site-
packages\keras\src\metrics\__init__.py:206 in get
    raise ValueError(f"Could not interpret metric
identifier: {identifier}")

ValueError: Could not interpret metric identifier: loss
```