NOVA IMS Information Management School

MDSAA

Master Degree Program in

Data Science and Advanced Analytics

Supervised Machine Learning in SAS Viya

Development of a Supervised Machine Learning pipeline in SAS Viya for comparison with a pipeline developed in Python

Guilherme Luís Ataíde Neves

Internship Report

presented as partial requirement for obtaining the Master Degree Program in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

2022

Title:

Subtitle:

Student full name

MDSAA

[this page should not be included in the digital version. Its purpose is only for the printed version]

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

SUPERVISED MACHINE LEARNING IN SAS VIYA

by

Guilherme Luís Ataíde Neves

Internship report presented as partial requirement for obtaining the Master's degree in Advanced Analytics, with a Specialization in Business Analytics

Supervisor: Phd. Roberto André Pereira Henriques

November 2022

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledge the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Guilherme Luís Ataíde Neves

Lisbon, 27th November 2022

ACKNOWLEDGEMENTS

First of all, I want to thank my tutor at SAS Portugal, Sara Santos, and João Manique, Consulting Manager, for their availability, guidance throughout the internship and development of the work presented in this report. I hope the knowledge passed by them to me is going to be helpful during my professional career. On the other hand, I must thank the Human Resources team at SAS for making this internship so fulfilling, having known so many people, including fellow internees from other countries.

Secondly, I would also like to thank my supervisor, Prof. Roberto, for his support during the development of this report, providing good advice and enhancing my knowledge about the Machine Learning field.

Lastly, I must not forget my family, whom I had alongside me during my academic journey, always supporting and encouraging me to make a good report. Without them, the work presented in this report would not be possible to be developed.

ABSTRACT

This internship report details the development of a supervised ML pipeline in SAS Viya, a cloud-based environment composed of several solutions for importing, managing and transforming data and building and deploying predictive models into production environments. As a practical case study, this report showcases the SAS Viya features and capabilities which can be offered to the end-user. A comparison with a similar supervised ML pipeline in Python was made, to highlight both tools' advantages and disadvantages. Thus, analytical tasks were employed, to demonstrate which different supervised ML techniques can be used in each technology.

Furthermore, it was shown that, depending on the experience and knowledge of the end-user, both SAS Viya and Jupyter Notebook/Python are able to produce satisfactory results, being the latter more suited to data scientists with some experience in programming and ML. At the same time, SAS Viya fits more for employees who are getting started in the ML field, due to its point-and-click user interface. On the other hand, building a supervised ML pipeline in SAS Viya can be more straightforward than in Jupyter Notebook/Python, since the code is already developed and the process automatized, while pipeline templates are made available to the user. However, due to its open-source nature, Python has more supervised ML techniques available to be used in Jupyter Notebook.

This report shows that these two solutions can complement each other, as SAS Viya offers good visualizations for data exploration, while Jupyter Notebook/Python can be dedicated to data transformation and predictive models' development.

KEYWORDS

Supervised Machine Learning; Binary Classification; Predictive Models; SAS Viya; Python

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
API	Application Programming Interface
AUC	Area under the Curve
AutoML	Automated Machine Learning
CAS	Cloud Analytic Services
DSAA-BA	Data Science and Advanced Analytics with a major in Business Analytics
EMEA	Europe, Middle East and Africa
GUI	Graphical User Interface
ML	Machine Learning
ReLu	Rectified Linear Unit
REST	Representational State Transfer
RFE	Recursive Feature Elimination
ROC	Receiver Operating Characteristic
VA	Visual Analytics

1. INTRODUCTION

Nowadays, data are generated faster than ever before; consequently, the amount of data stacked in a company's systems grows exponentially. Thus, the organizations and their analytics teams must deal with this constantly incoming flow of data, commonly accessed in an ad-hoc fashion, by different departments. Moreover, these data are not consistent across their data silos in many companies, hence resorting to data governance policies might help to solve that problem, such as tracking data lineage to identify its source, checking how the data has been transformed and consumed and by whom, as well as which people and which models introduced bias into the data. That is why it is paramount to check for all these issues in light of today's data privacy and confidentiality guidelines, together with the opportunity to take trusted decisions based on insights supported by data governance policies (Baer 2020).

Keeping those regards in mind, **SAS Institute Inc.** developed a cloud-native architecture to deal with the increasing data complexity, constantly changing business needs and higher computation power demand, named **SAS Viya**. This environment is composed of several Analytics software solutions which can complement each other to build an entire ML project from end-to-end, ensuring the correct data is imported and transformed according to data privacy and confidentiality guidelines, ML models able to output accurate predictions are created in order to deploy them in a production environment in the future.

However, this internship report is focused on a part of a ML project, the development of a supervised ML pipeline. For that purpose, a SAS Viya application was used, named **SAS Model Studio**, a tool specifically designed for allowing every user from the less experienced to the one with more domain expertise, to develop a supervised ML pipeline. To do so, this software has many analytical tasks available to add to a pipeline, aiming to produce predictive models with a good performance. Nevertheless, SAS Model Studio should not be used alone when designing a supervised ML pipeline, as **SAS VA** should also be utilized for creating visualizations, helpful for exploring the data.

Therefore, the main goal of this internship report is showcasing the capabilities those two SAS Viya applications can offer, through the presentation of a practical case throughout this report to demonstrate that, explaining in detail the analytical tasks which can be part of a supervised ML pipeline developed in SAS Model Studio, since data importing and transformation until models' selection and evaluation steps. In order to do so, a real dataset containing information about marketing campaigns made by a Portuguese banking institution targeting their customers is taken as a starting point.

On the other hand, it is paramount to compare a SAS Model Studio pipeline with a technology where a supervised ML pipeline can be developed as well. Accordingly, as the secondary goal of this report, a comparison between a SAS Model Studio pipeline and a pipeline built in **Jupyter Notebook**, coding in **Python**, is shown in this report, using the same dataset. Additionally, the main differences and common points between the two technologies are highlighted, providing a better understanding of the advantages and disadvantages each one has.

2. INTERNSHIP

As a part of the **Students@SAS** internship programme for the EMEA region, the internee had the opportunity to embark on a 6-month internship at **SAS Portugal**, the **SAS Institute Inc.** offices in Portugal.

SAS Institute is a privately held software company, based in Cary, North Carolina, in the United States. It was founded in 1976 by James Goodnight, current CEO, John Sall, current Executive Vice President, and Anthony James Barr, following the development of a project known as "**Statistical Analysis System**", which further gave its name to SAS Institute. This project was primarily built to analyze how soil, weather and seed varieties affected crop yields, supported by the agricultural department of North Carolina State University and funded by the National Institutes of Health.

Nowadays, SAS solutions stretch over various areas, namely AI and ML, Data Science, predictive analytics, fraud management, risk management, anti-money laundering, multichannel marketing, customer analytics and Internet of Things. Their users are also not limited to use these applications alone, as they can be available or integrated within other software solutions, such as Amazon Web Services, Google Cloud, Red Hat OpenShift or Microsoft Azure, among others.

Since its foundation, SAS has not stopped to grow from year to year and widen their operations across the globe, having more than 12,000 employees and 82,000 business, government and university customer sites in 145 countries. In 2021, SAS registered 3.2 billion dollars in revenue and a 5.2% growth year over year, despite Covid-19 pandemic, with the banking, government and insurance sectors contributing to 59% of the total revenue. Moreover, 88 of the companies represented in the 2021 Fortune 100 were SAS customers or affiliates and SAS topped more than 30 vendor ranking reports in the same year.

On the other hand, SAS has received many awards throughout the years, such as "The Best Place to Work for LGBTQ+ Equality" award, recognizing the company's endeavors on innovation, social impact, diversity and inclusion, as well as workspace culture.

Relatively to the internship itself, the internship main goal was to develop the technical and soft-skills of the internee needed for the Technical Consultant position, being internee integrated in the Professional Services team, in the Consulting Area. The skills to be acquired during the internship were related to developing general knowledge about Data Management, Business Intelligence and Analytics, using SAS technologies, by granting access to several courses in those domains, whose attendance proved to be fundamental to write this report and develop the work described in it.

3. LITERATURE REVIEW

In this chapter, the main concepts addressed in this report are addressed and detailed, namely the visualizations, analytical tasks and predictive models used in the practical case developed in SAS Model Studio and SAS VA, as well as Jupyter Notebook/Python, in order to give some context to the work presented in the Methodology chapter, as well as the Results and Discussion chapter.

3.1. ARTIFICIAL INTELLIGENCE

AI was defined by Richard E. Bellman as the automation of "activities that we associate with human thinking, activities such as decision making, problem-solving, learning" (Bellman 1978), therefore, it is a simulation of human intelligence by machines. It is tied to the goal of programming computers to perform complex tasks, matching or even surpassing human performance capacity. However, this is not possible for machines to achieve in every problem-solving situation, as some specific tasks are yet to be as well performed by them as humans do, due to greater human flexibility to solve distinct problems and the need to resort to knowledge acquired on a daily basis.

3.2. MACHINE LEARNING

ML is a sub-field of AI and it is a set of procedures which automate the extraction of hidden patterns and relationships within the data (Kelleher et al. 2015). It tries to mimic the way humans learn who are taught by experience and trial and error. Likewise, ML is about training algorithms which improve their accuracy through the access to data.

3.3. SUPERVISED MACHINE LEARNING

Supervised ML is a sub-division of ML. Supervised ML is about finding the predictive model whose generalization to unseen data is the best among the candidate models which are searched during the learning process, based on a supervised ML algorithm. This model is trained on a labeled dataset, constituted by training instances, descriptive features and a target feature, allowing the predictive model to learn the relationship between the descriptive features and the target feature, for predicting the outcomes of that target variable on unseen data. Detailed explanation about some of the supervised ML algorithms used are detailed in upcoming chapters.

Nowadays, supervised ML has a lot of real-world applications, such as:

- Predicting customer churn.
- Predicting stock prices.
- Forecast sales.
- Forecast supply and demand.
- Fraud detection.
- Face detection.
- Among others.

However, supervised ML is only useful when one has a large set of training instances, because there are a lot of different cases for a specific domain, existing the need to collect as many examples of that domain as possible for capturing the underlying relationship between a set of predictive variables and

the variable to be predicted. Nonetheless, there are also other issues which arise from selecting a large dataset for a supervised ML model to be trained on, such as the existence of noise in the data (missing values, outliers, high cardinality, lack of data standardization, large set of input features, among others). Therefore, one must follow an ordered set of steps for solving those issues within the data, being them divided in data exploration steps:

- Variable identification.
- Univariate descriptive analysis.
- Bivariate descriptive analysis (Magdum 2022).

And data preparation steps:

- Missing values treatment.
- Outliers' treatment.
- Feature transformation.
- Feature creation (Magdum 2022).
- Feature selection.

After those steps are performed, one is ready to search for the best model which is able to have more prediction accuracy for both training datasets and unseen data, which comprehends the following two stages:

- Model selection.
- Model evaluation.

3.4. DATA EXPLORATION

Data exploration corresponds to the first step in a supervised ML project, where visualizations and statistics are used to understand and characterize the dataset, in terms of size, quantity and accuracy (Jaiswal 2022). Therefore, those techniques are described in the following chapters for understanding their role in the following stages of a supervised ML project.

3.4.1. Variable Identification

The first step in the data exploration phase is **variable identification**, which is about perceiving the meaning of each variable. At this stage, input variables need to be defined, which are features further used to predict the output variable. These variables can have different types, namely categorical (nominal or ordinal) and continuous (interval and ratio), and these features' types should be changed at this phase, if they were previously assigned with the wrong data type. At the same time, feature values which are not known when the target is known or variables with too many missing values should be dropped from the dataset. In the next two steps (univariate and bivariate descriptive analysis), those variables data are analyzed and some takeaways can be assumed, being them fundamental for conducting the following stages of a supervised ML project.

3.4.2. Univariate Descriptive Analysis

Univariate descriptive analysis of a single variable aims to explain the variable distribution in one sample (Canova et al. 2017). There are several techniques and measures for assessing a variable's

distribution, whether **continuous** or **categorical**. For continuous features, **central tendency measures** (mean, median or mode) are used, together with measures of **variability** (range, interquartile range or standard deviation), as well as **quantiles**, checking **minimum and maximum** values, **skewness** (measures the distortion of a variable distribution from the normal distribution) and **kurtosis** (measures the pointiness or flatness of variable's distribution relatively to the normal distribution). For categorical features instead, **frequency tables** and **mode** are the most common used techniques.

Visualizations are also a great source to look at the frequency distribution of each feature. Again, those visualizations are different for continuous and categorical features. For the first group, **histograms** or **barplots** with the data splitted in bins can be plotted for knowing more about the distribution of the data, as well as **boxplots** which can help to identify outliers. For categorical variables, **barplots** are built for grasping the data distribution, as well as identify high and low cardinality.

3.4.3. Bivariate Descriptive Analysis

In **bivariate descriptive analysis**, the main goal is to analyze bivariate data similarly to what is done for analyzing univariate data. Bivariate data is characterized by being data where each observation has two attributes. (Lane 2013).

There are also some ways to evaluate the relationship between two variables. One of them is resorting to Pearson Correlation coefficient, which measures the linear relationship between two interval features, so as a feature's values increase or decrease, the other feature's values increase or decrease by the same amount, respectively (Kelleher et al. 2015). Its values range from -1 to 1, where -1 means perfectly negative linear correlation, 1 perfectly positive linear correlation and 0 no linear correlation between two variables. Usually, the features correlations are analyzed through a **Pearson correlation matrix**.

A **target variable by categorical variables' crosstabulations plot** is also a kind of visualization within the scope of bivariate descriptive analysis. These charts allow the user to perceive the relationship between each class of the target variable and each class of categorical variables, helping to detect which classes are more relevant for defining each target class.

3.5. DATA PREPARATION

In the next chapters, the **data preparation** tasks will be detailed. These tasks must be completed before diving into the model selection and evaluation stages, involving the manipulation of raw and unstructured data in order to transform it into a more structured fashion (Abdallah et al. 2017) ready for being fit into predictive models, in the context of a supervised ML project.

3.5.1. Missing Values' Treatment

Missing values are data points whose variable value does not exist for a specific observation (Kang 2013). It is a very common issue in many studies and there are various problems which arise from it, including:

- Bias is introduced into the estimation of parameters.
- Samples lose their representativeness of the population.
- The phenomenon being studied is more difficult to be perceived.

- Some supervised ML algorithms do not tolerate the existence of missing data or their accuracy is affected.

Still, data can be manipulated to get rid of this issue. One can simply eliminate observations with missing values, however, this should only be applied when data is missing completely at random (MCAR). This type of missing data can be defined as "when the probability that the data are missing is not related to either the specific value which is supposed to be obtained or the set of observed responses" (Kang 2013). Nonetheless, it does not happen for the majority of datasets, so it can lead to a biased estimation of the parameters (Donner 1982).

On the contrary, **imputation methods** are used to replace missing values with other values. For instance, one can input missing data with the mean, median, mode or another constant value for a specific variable. Other examples of imputation methods are replacing missing values with a value estimated from a linear regression based on the values of related variables, or with a value estimated from the distribution of the data, or with the mean value obtained from similar non-missing observations.

3.5.2. Outliers' Treatment

Outliers can be identified within a dataset as samples which are far apart from the remaining data points (Kuhn and Johnson 2013). This kind of observations might occur due to measurement errors, data collected incorrectly or simply because the observation is a true outlier. Furthermore, outliers can reduce the predictive models' accuracy, as they can skew the representativeness of the models.

In order to eliminate this kind of noise from the data, one can apply the rule of thumb which consists in deleting observations whose values are not comprehended between 3 standard deviations from the mean, because it has been studied that it is uncommon for measurement errors to not respecting this threshold when normal random effects impact the value (Morris & Langari 2012). Therefore, this should only be applied for features whose data distribution follows a normal distribution or close to it.

Visualizations, such as **scatterplots** and **boxplots** can be also helpful to detect outliers. In scatterplots, observations which are far away from the majority of the data in a two-dimensional space can be perceived as outliers. On the other hand, boxplots identify outliers as points which do not lie between the minimum and maximum points of the boxplot, as shown in Figure 1.

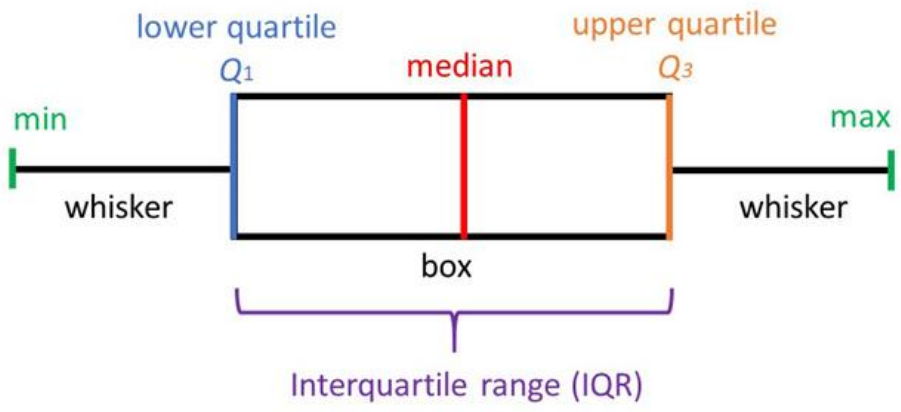


Figure 1 – Example of a boxplot

3.5.3. Feature Transformation

Feature transformation is a task performed within the data cleaning process, being a step of the broader feature engineering process. **Feature engineering** is referred as "the task or process of altering the feature representation of a predictive modeling problem to better fit a training algorithm" (Khurana et al. 2017). In other words, it aims at creating new variables which do not exist in the training dataset for achieving better model accuracy, simplifying data representation and speeding data processing.

Moreover, feature transformation is about replacing existing variables with some specified functions, changing their data distribution or scale and consequently, the relationship between them. For instance, these transformations can be carried out for reducing the skewness and kurtosis of a continuous variable's distribution or for changing the scale of the data. Correcting skewness and kurtosis can be performed through the application of **power transforms** (Zheng & Casari 2018), which are a family of transformations which stabilize the variance, meaning they transform the data distribution of a variable to be closer to a normal distribution, shifting kurtosis value towards 3, while skewness to be 0. At the same time, power transforms are also able to enhance the accuracy of a predictive model, such as neural networks. (Larasati et al. 2018). Other types of power transforms are **log** transformation, **square root** transformation, the **inverse square** transformation (inverse of the squared value plus 1), as well as the **Yeo-Johnson** transformation. The latter applies some transformations to a feature, which are described in Equation 3.1, whether the values are positive or negative. These calculations are also made based on a value λ, which can be selected through maximum likelihood estimation, assuming the variable follows a normal distribution.

$$\psi(y,\lambda) = \begin{cases} \dfrac{(y+1)^{\lambda}-1}{\lambda}, & y \geq 0 \; and \; \lambda \neq 0, \\ \log(y+1), & y \geq 0 \; and \; \lambda = 0, \\ -\dfrac{(-y+1)^{2-\lambda}-1}{2-\lambda}, & y < 0 \; and \; \lambda \neq 2, \\ -\log(-y+1), & y < 0 \; and \; \lambda = 2. \end{cases} \tag{3.1}$$

On the other hand, **feature scaling** consists of changing the scale of a feature, being applied to each feature individually (Zheng & Casari 2018), in order to set every variable to the same scale and reduce the influence of features with high values in the learning process of predictive models. Consequently, this might improve the accuracy of some models, namely logistic regression and neural network, while tree-based models, such as decision tree, random forest and gradient boosting are not affected by that.

Data standardization is an example of feature scaling, consisting of a calculation which subtracts to each feature value the mean, dividing the subtracted value by the standard deviation for that feature. Accordingly, it sets the feature mean and standard deviation to be 0 and 1, respectively. Another kind of data scaling transformation is **Robust Scaler**, which is a method known for being a transformation which is robust to outliers. It subtracts the median to each variable data point and divides the subtracted value by the interquartile range (third quantile, corresponding to the value which is higher than 75% of the data, minus the first quantile, which is the 25% value).

3.5.4. Feature Creation

Feature creation is another fundamental step part of the feature engineering stage. As the name suggests, it encompasses generating new variables derived from the initial set of input features, becoming these newly created features part of the set of input features (Jaiswal 2022). Feature creation should be carried out due to several reasons, namely **handling high cardinality** of categorical variables or **encoding categorical variables**.

Categorical variables' cardinality can be defined as the number of distinct values which that same variable can take (Perlich & Provost 2006), meaning a feature with high cardinality has a lot of unique values. This might be a problem, since it can lead to supervised ML models having low accuracy, while increasing their training time. There are several ways to solve this issue, being one of them aggregating several categories into a single category (might be called "Others", for instance) or creating new categories which aggregate several others.

On the contrary, encoding categorical features is a task of extreme importance. An example of an encoder is **one-hot encoding**, which is one of the most used for labeling categorical features, consisting of generating binary variables, each one corresponding to a level of the categorical feature (Cohen 2013), being this type of encoding suited for logistic regression and neural network models rather than tree-based models. Moreover, using a **label encoder** for labeling the classes of a categorical feature with numerical values might be an option as well, especially if the predictive model is a tree-based model.

3.5.5. Feature Selection

The **feature selection** step is the last task in data preparation stage to be carried out before proceeding into the model selection phase. Feature selection is the set of tasks which involve selecting a set of relevant features in the dataset according to various criteria (Cai et al. 2018), which detailed in this chapter. Thus, feature selection is also considered a dimensionality reduction step, helping to diminish the effect that the **curse of dimensionality** has on the data (Bellman 2021), which states that as the dimensional space increases, the number of observations required for any supervised ML model to perform accurately grows exponentially. Therefore, feature selection methods help to reduce the algorithms' learning time and improve the models' accuracy and interpretability of their outputs. The division of feature selection methods is shown in Figure 2 ("Filter" methods are out of the scope of this report).

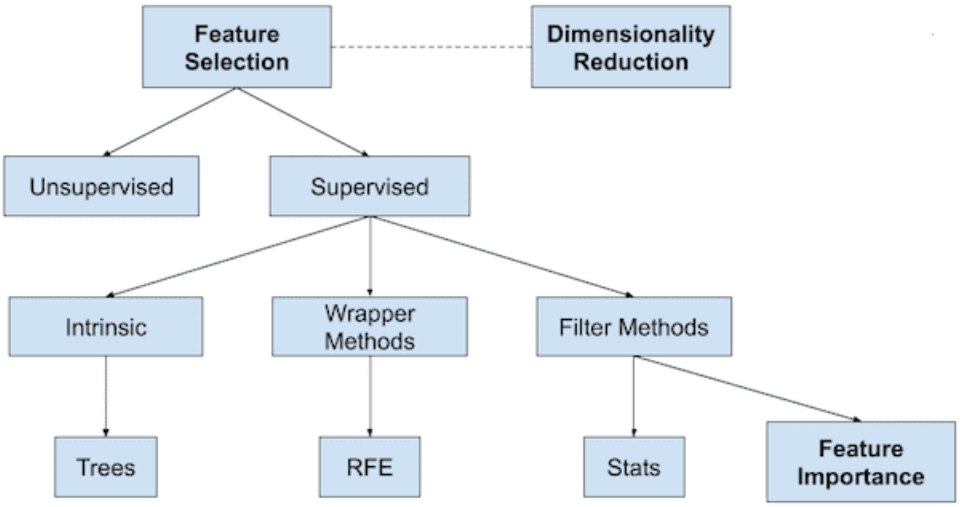


Figure 2 – Feature selection methods division

According to Figure 1.6, there are unsupervised and supervised feature selection methods. **Unsupervised** techniques do not use the target variable such as the **Pearson correlation coefficient**. This coefficient is useful to identify **redundant** features, which are variables correlated with each other, thus adding the same information for explaining the target's variance. Another type of unsupervised technique is the **Phik correlation coefficient** (Baak et al. 2020) which overcomes the disadvantage of Pearson correlation, which evaluates the linear relationship between two interval variables. Accordingly, the Phik correlation is aimed at analyzing non-linear relationships between categorical, ordinal and interval features.

Another type of methods to use for feature selection are **supervised** methods, meaning they use the target variable aiming to remove **irrelevant** variables, whose inclusion in the features' subset do not add any information for explaining the target's variance. These techniques can be also divided into three categories, namely **wrapper**, **intrinsic** and **filter** methods, being the latter out of the scope of this report. The first group is characterized by creating various models using different subsets of predictors and selecting the best subset according to some performance criterion. On the other hand, intrinsic methods include supervised ML models which rank input features according to their importance as part of the learning process (Kuhn & Johnson 2013).

One example of a wrapper method is **RFE** (Guyon et al. 2002). This algorithm works as a backward selection iterative process, because it fits a model to data using the full set of predictors and successively eliminating the least important input variable for the model, which are ranked in each iteration according to their importance for the model. At the beginning of each iteration, the model is rebuilt with the new set of predictors and its performance is calculated, followed by the variables' ranking. The optimal subset of input variables is determined according to the model's performance for each subset. Various supervised ML models can be used in this algorithm, such as logistic and linear regressions, as well as tree-based models.

An example of an intrinsic method is **decision tree**. This predictive model, fully explained in the Predictive Models chapter, orders the variables in a top-down fashion according to their importance,

which is given by **Information Gain** (based on entropy) or **Gini index** measures. Therefore, the most important features are the ones at the top of the decision tree, whereas least important ones are placed at the bottom or not included in the tree.

3.6. MODEL SELECTION

After the data preparation stage is done, data is ready to be fed into supervised ML models, starting the model selection phase.

This stage comprises choosing a predictive model among a set of candidate models for predicting a target variable. These candidate models can comprehend different models with different assumptions, such as the one already described in the **Predictive Models chapter**, or models of the same type, but built with different hyperparameters (e.g. two decision trees with different depths).

However, in order to select one final model for production, one must have enough data to feed into the candidate models. Therefore, the dataset should be split into two different subsets: **training** and **validation** sets, where the training set is a set of several observations used for training the ML models, while the validation set is made up of instances which are used to fine-tune the hyperparameters of a model (Ripley 1996).

There are two main sets of methods to consider when selecting the best predictive model, called **probabilistic measures** and **resampling methods**. Probabilistic measures consider the performance of the supervised ML model along with the complexity of it, adding a penalty term to account for the overfitting of more complex models. Nonetheless, there are some disadvantages associated with probabilistic measures, as these measures are more adapted to linear and logistic regressions, due to model complexity penalty being known. Moreover, this kind of statistics cannot be applied to several types of candidate models and they do not consider the variability of model parameters, thus simpler models are often preferred over more complex ones.

On the contrary, resampling methods, used in the context of this report, aim to estimate the model hyperparameters based on an out-of-sample data, where data points are resampled from the original data to create several datasets (Davison & Hinkley 1997), being these subsamples used to estimate the population parameters.

There are two main resampling methods used, namely the **bootstrap method** and **K-fold cross validation method**. The bootstrap method consists of sampling data from the original sample, splitting these data across several subsamples, all having the same sample size. These data points are randomly selected with replacement, thus some data points can be repeated across the subsamples. After that, estimates of parameters can be calculated for each subsample, being them averaged and compared with samples which were not selected form the original sample. These observations, called “out-of-bag” samples (Kuhn & Johnson 2013), are used to assess the performance of the model, through some evaluation metrics, which are detailed in the next chapter.

K-fold cross validation is an approach which involves splitting the data randomly selected from the original dataset into k folds or subsets, each one of the same size. Usually, this value of k is 5 or 10, because these values have been proved to produce test error rate estimates which are not too much biased or having too high variance (James et al. 2013). The first fold becomes the validation set, leaving the others to be the training datasets, and the evaluation metric is estimated for the validation set.

This process is replicated k times for all subsets, meaning each sample is used for testing the model, as shown in Figure 3, where the fold in blue represents the validation set at each iteration.. In the end, the evaluation metric is averaged over all subsets, becoming it the final metric for assessing the model's performance. One of the most popular types of k-fold cross validation is **stratified k-fold cross validation**. It ensures that the proportion of each target class in the original dataset is retained for all the training sets, as well as for validation sets during the process of cross validation, preserving the target variable distribution for all subsets.

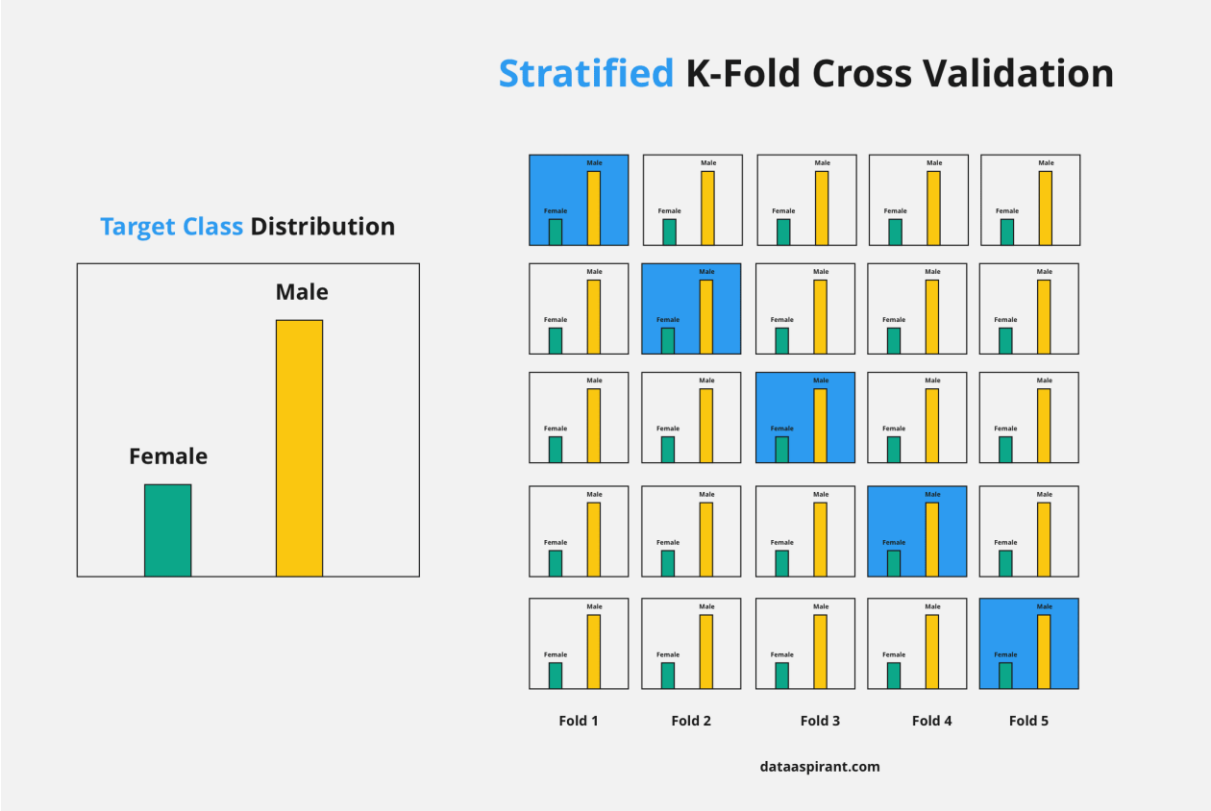


Figure 3 – Stratified k-fold cross validation schema

3.7. Model Evaluation

The **model evaluation** phase permits to assess the model's performance, in other words, the quality of the predictions made by the model on unseen data. This stage is closely tied to the model selection step, due to the need for resorting to a series of evaluation criteria in order to choose the model with the best performance and thus, to be put into production. The confusion matrix, whose schema example is shown in Figure 4, serves as the basis for calculating those evaluation metrics in a binary classification problem, by splitting the predictions into four different groups, whose labels are the following:

- **TP** (True Positives) – observations corresponding to the target class **1** which were correctly predicted.
- **TN** (True Negatives) – cases for class **0** correctly predicted.
- **FP** (False Positives) – instances whose target class is **0**, but were incorrectly predicted.
- **FN** (False Negatives) – data points whose output class is **1**, yet incorrectly predicted.

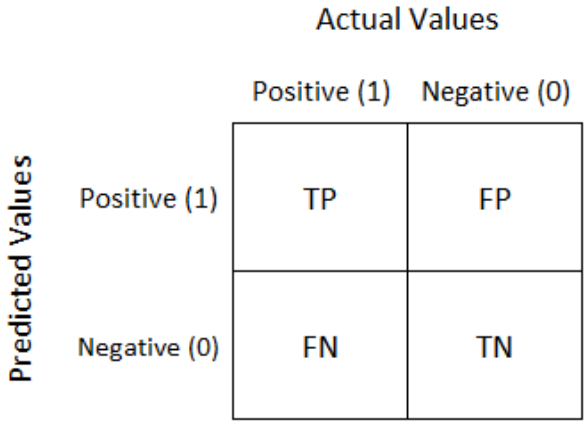


Figure 4 – Confusion matrix schema

There are many evaluation metrics used to assess the models' performance within the context of binary classification problems. Some of them can be calculated based on the confusion matrix, whose values range between 0 and 1, being a higher value associated with a better model's performance (Dalianis 2018). These metrics are the following:

- **Accuracy** (Equation 3.2) - proportion of correctly predicted cases relatively to the total number of observations.
- **Recall** or **sensitivity** (Equation 3.3) - proportion of correctly predicted positive cases out of the total number of positive instances.
- **Precision** (Equation 3.4) - proportion of correctly predicted positive observations relatively to the total number of predicted positive cases.
- **F1 score** (Equation 3.5) - weighted average of recall and precision.

$$\textbf{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.2}$$

$$\textbf{Recall} = \frac{TP}{TP + FN} \tag{3.3}$$

$$\textbf{Precision} = \frac{TP}{TP + FP} \tag{3.4}$$

$$\textbf{F1 score} = \frac{2 \times precision \times recall}{precision + recall} \tag{3.5}$$

Another types of assessment metrics are **lift** and **ROC curve**. Lift is calculated as the predicted percentage of 1s in the dataset divided by the average percentage of 1s in the dataset, giving for each sample proportion the probability of finding 1s compared to the average (e.g. if the proportion is 10% and the lift 3, then it is 3 times more likely to find 1s for that sample than the average proportion of 1s in the data, meaning it is expected to catch 30% of the total 1s in the dataset in the 10% sample). It

can be assessed by looking at a **lift chart**, which can help to distinguish two models in terms of performance, where the model with better performance usually has greater lift for the different sample proportions.

On the other hand, ROC curve is a probability curve, being often analyzed together with **AUC**, which measures the degree of separability between target classes, thus it evaluates whether a model distinguishes well 0s and 1s, in the context of a binary classification problem. Accordingly, predictive models with better performance have higher AUC, ranging from 0 to 1, where 0 means the model is predicting 0s as 1s and vice-versa, while a model with AUC equal to 1, is able to predict all 0s and 1s correctly.

3.8. PREDICTIVE MODELS

In this chapter, the predictive models mentioned throughout the report are explained. The learning algorithms for training the models and their usage in the context of binary classification problems are detailed, along with some figures and equations which help the reader to better understand how they work.

3.8.1. Logistic Regression

In Hyeoun-Ae Park's opinion, **logistic regression** depicts the relationship between several input variables and a categorical target variable, where the likelihood of a certain event to happen is calculated by fitting data to a logistic curve (Park 2013). This S-shaped or sigmoid curve is characterized by a linear and slow growth at start, followed by an exponential growth until it flattens out, varying between 0 and 1. For the special case of a binary classification problem, the binary logistic regression is used and the independent variables (predictive variables) can be either continuous or categorical, while the target variable represents the probability of an event to occur.

There are two models of this algorithm, called **logit** model and **probit** model, which are represented in Figure 5. Some differences exist between them and the most important ones are:

- In terms of predicting y, the logit follows the cumulative distribution function of the logistic distribution, while the probit uses the cumulative distribution function of the standard normal distribution.
- The probit model grows from 0 to 1 faster than the logit model.
- It is easier to interpret the effects of x on y on logit than in probit because the latter follows a standard normal distribution, while the first transforms the odds, which "are the ratio of the probability that an event will occur to the probability that it will not occur" (Park 2013) using the natural logarithm (Peng et al. 2002).

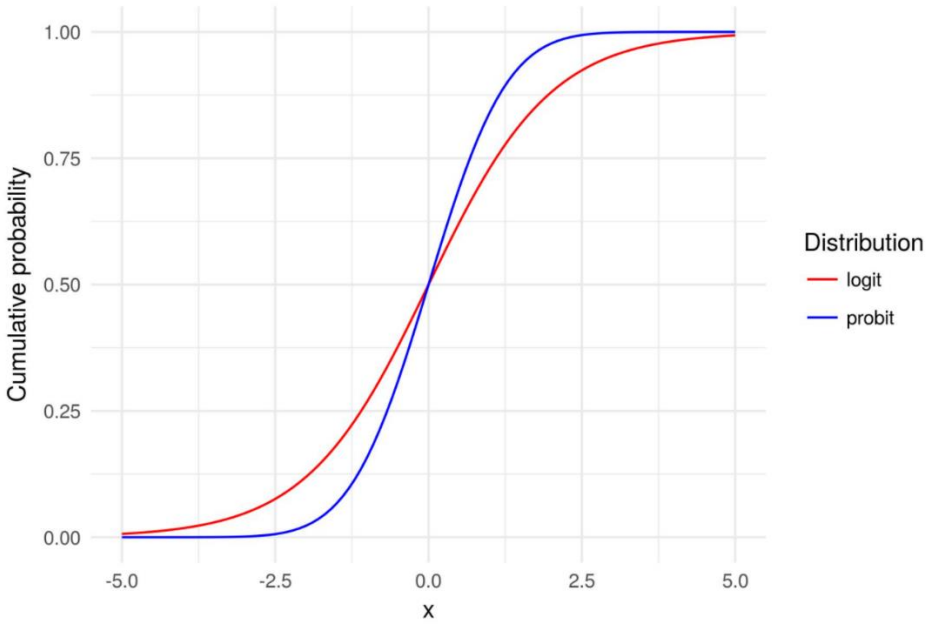


Figure 5 – Logit and probit curves

The event probability for logit can be estimated through Equation 3.6:

$$E(Y = 1|\, x) = \pi(x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}} \tag{3.6}$$

being β_0 the intercept and β_1 the slope of the logistic regression.

On the other hand, the event probability for probit can be estimated through Equation 3.7:

$$E(Y = 1|\, x) = \pi(x) = \Phi(X'\beta) \tag{3.7}$$

where Φ is the cumulative standard normal distribution.

3.8.2. Neural Network

Developed by F. Rosenblatt, the **neural network** algorithm was firstly called **perceptron**. This approach tried to imitate the nervous system, where certain stimuli are received, being perceived by association cells as impulses, which then transmit these signals to response cells, responsible for interpreting and learning as the signals are reinforced from the association cells (Rosenblatt 1958). Similarly, the perceptron (Figure 6) takes the inputs (variables' values) jointly with randomly assigned weights, which are updated in an iterative process (Equation 3.8), for producing outputs based on the weighted sum of the inputs.

$$w^{(r+1)} = w^{(r)} + \phi^n t^n, \tag{3.8}$$

where $w^{(r)}$ is the weight in iteration r, t^n is the value of the output variable for observation n and ϕ^n "a vector which is misclassified by the perceptron" (Bishop 1995).

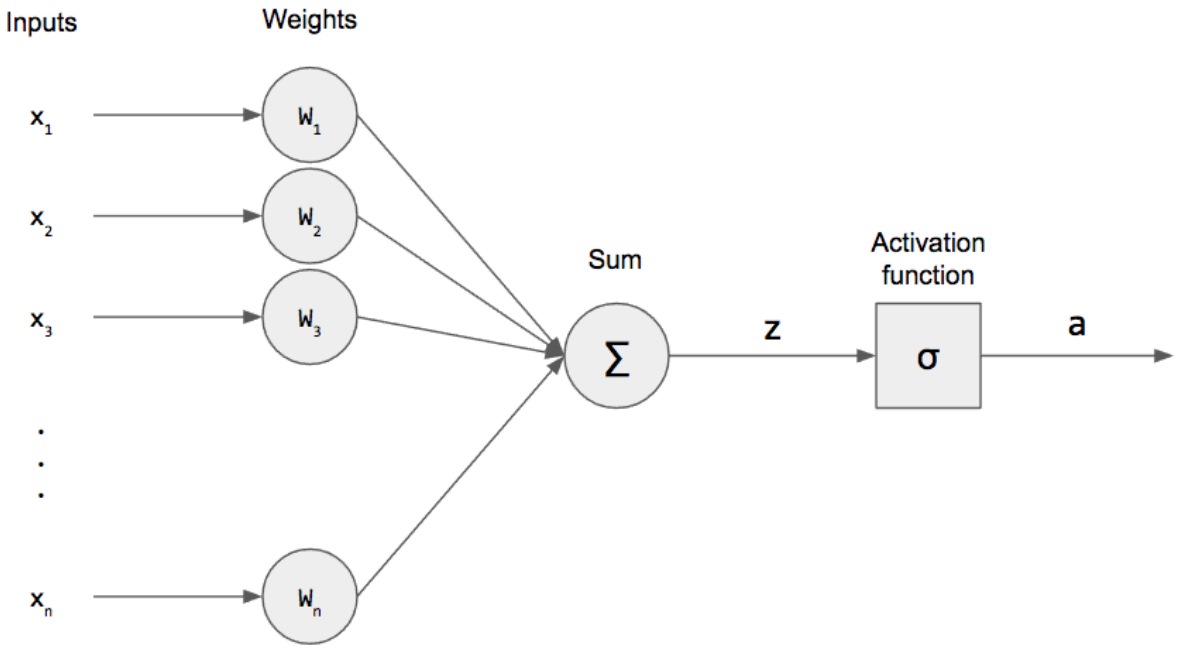


Figure 6 – Perceptron schema

These weights are updated iteratively, where errors are successively minimized by reducing the difference between target output values and predictions, called the error, until a threshold is achieved. This threshold is defined by a step activation function, which defines whether an observation is given 0 or 1 for binary classification problems, although other functions can also be used.

However, the perceptron as a single-layer neural network, is not suited for regression problems. Therefore, another model was developed to allow for solving both classification and regression problems, as well as more complex problems. This algorithm, called **artificial neural network**, also known as **multi-layer perceptron**, shown in Figure 7, is constituted of three layers, namely input, hidden and output layers, which are then composed of neuron, responsible for introducing non-linearity into the algorithm. Predictions' calculations begin with the input layer down to the output layer, passing by the hidden layer (Park & Lek 2016).

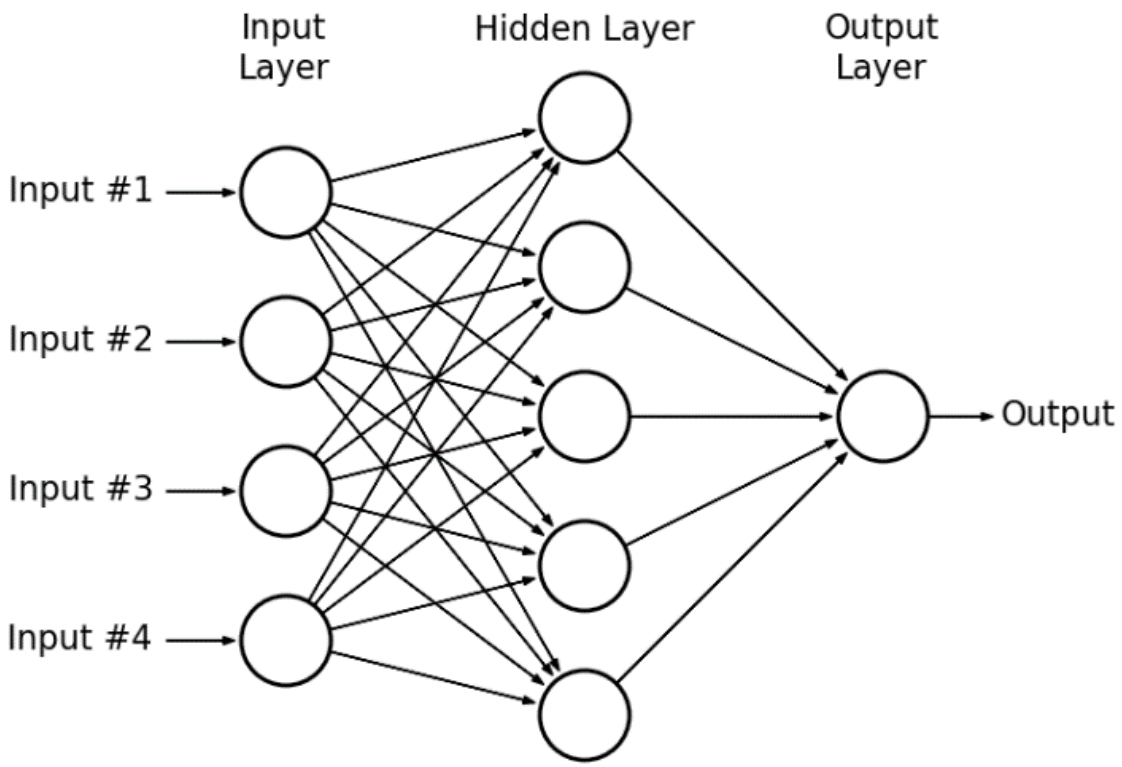


Figure 7 – Multilayer perceptron network

Usually, the input layer is composed of as many neurons as input variables, although one can account for the bias as well. The hidden layer is responsible for adding more complexity into the model, by taking the outputs from the input layer and deliver its outputs to the output layer. It can be composed of several neurons, where a lower number might lead to underfitting, while a too large number can introduce overfitting of the data. However, if one wants to increase the accuracy of the model, one can add one more hidden layer, which creates classification regions of any desired shape (Lippmann 1987). On the other hand, the output layer should only be comprised of a single node, responsible for classifying the instances together with an activation function, which is optional in this layer.

Contrarily to the primary perceptron, the multi-layer neural network uses non-linear functions as activation functions. Among some of the functions used for this purpose are the **sigmoid curve**, mentioned in the **Logistic Regression** chapter and **ReLu** function, represented in Figure 8.

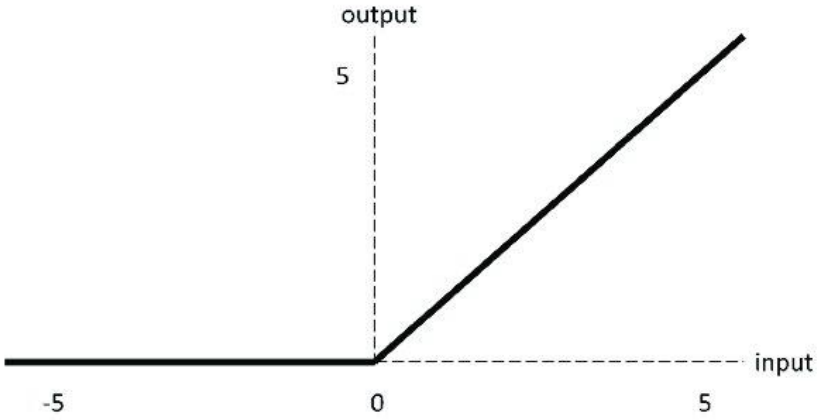


Figure 8 – ReLu activation function

ReLu is widely used nowadays, allowing for faster and more accurate training than the sigmoid function. Being a nearly linear function, it maintains "many of the properties that make linear models easy to optimize with gradient-based methods" (Goodfellow et al. 2016). This characteristic is important for the **backpropagation** algorithm discussed in the next paragraph. It outputs negative input values as zero, while being linear for positive input values and it can be calculated following Equation 3.9.

$$f(x) = \max(0, x) \tag{3.9}$$

These three layers are connected to each other in a **feed-forward** fashion, meaning each neuron in each layer receives as inputs the outputs of the previous layer, producing its own outputs for the neurons in the next layer. This feed-forward neural network allows to backpropagate the errors (difference between predicted values and actual values of the target), in other words, it considers the overall network output error and updates the weights in order to improve the predictions of the outputs (Câmara 2015). Thus, the output of a given neuron H_j is given by Equation 3.10:

$$H_j = \phi(h_j) = \phi\left(\sum_k v_{jk} x_k\right), \tag{3.10}$$

where v_{jk} stands for the weight coefficient between input neuron k and hidden neuron j, while $\phi(u)$ is the activation function for the hidden layer.

On the contrary, the input to the neuron i of the output layer is given by the outputs for the hidden layer, as expressed in Equation 3.11:

$$o_i = \sum_j w_{ij} H_j, \tag{3.11}$$

where w_{ij} is the weight for the connection between output neuron i and hidden neuron j, whereas the output of the neuron i is calculated as follows in Equation 3.12:

$$O_i = \psi(o_i), \tag{3.12}$$

where $\psi(u)$ represents the activation function for the output layer.

These weights are then updated successively for each training instance, following a gradient-descent algorithm, from the output layer down to the input layer, hence the name "backpropagation" algorithm, considering the errors and a specific learning rate.

3.8.3. Decision Tree

Decisions trees are trees that sort observations according to their features' values (Kotsiantis et al. 2006). Therefore, a decision tree starts with a root node or leaf (the top node), which defines the instances' classification according to a feature's values, usually splitting the observations below or above a threshold. This iterative process carries on, being each leaf responsible for selecting the best variable in discretizing between positive target instances ("YES" in Figure 9) and negative ones ("NO") for a binary classification problem, meaning the features are sorted from the root to the leaves in a descending fashion in terms of variable importance. This process is only ended once all the observations in a node are of the same class for the target feature, meaning they are pure, or when partitioning the instances no longer improves the accuracy of the model or if the tree has a predefined depth.

An example of a decision tree is shown in Figure 9, where x_i represents any variable, tl_i the threshold for a feature's value, $x_i \leq tl_i$ a node or leaf and R_i a pure leaf.

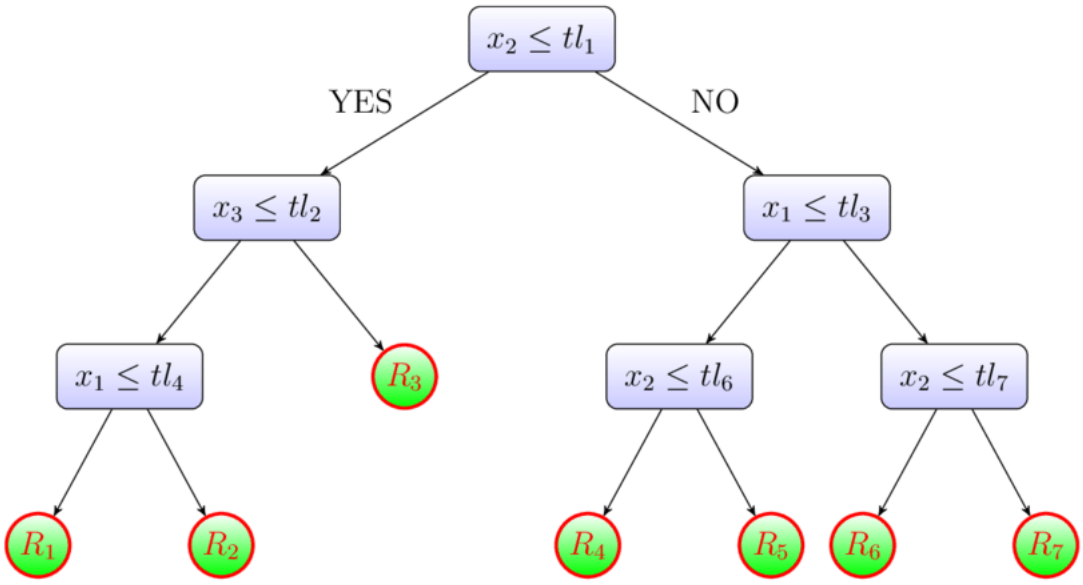


Figure 9 – Decision tree diagram

In other words, the goal in each node is to maximize the **information gain**, a metric which assesses how well a feature is able to distinguish between instances of different target classes (Mitchell 1997). This metric is tied to another measure, called **entropy**, which calculates the impurity of a set of examples S considering the proportion of positive instances p_+ in S, as well as the proportion p_- of negative ones, as given by Equation 3.13:

$$Entropy(S) \equiv -p_+ \log_2(p_+) - p_- \log_2(p_-) \tag{3.13}$$

Thus, taking into account entropy, the information gain, $Gain(S, A)$, is calculated by subtracting the sum of the entropies for each class of a variable. It is measured for all variables and the one whose information gain is higher, it is selected for partitioning the data. In Equation 3.14, it is shown how information gain is calculated, where A represents a variable, whereas $Values(A)$ is the set of all values for feature A and S_v represents the examples whose value for variable A is v.

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{3.14}$$

Another alternative for defining which is the best variable to split the data at a certain node is the **Gini index**. Contrarily to entropy, the feature which minimizes Gini index is selected at each iteration (Liu & Cocea 2018) for splitting the examples, rather than selecting the feature which maximizes entropy. Various formulas must be taken into account for selecting the variable: The process for choosing the attribute which minimizes Gini index at each leaf is:

- Firstly, the Gini index itself is calculated for a dataset D which contains n examples, whose formula is in Equation 3.15.
- Secondly, the Gini index is measured for each variable and each subset partitioned by that feature, being represented in Equation 3.16.
- Finally, the Gini Gain in Equation 3.17 is assessed for each variable. The attribute which maximizes this value is selected for splitting the data.

$$Gini(D) = 1 - \sum_{i=0}^{n} p_i^2 \tag{3.15}$$

$$Gini_A(D) = \sum_{i=0}^{n} \frac{|D_i|}{D} Gini(D_i) \tag{3.16}$$

$$Gini\ Gain\ (A) = \ Gini(D) - Gini_A(D) \tag{3.17}$$

3.8.4. Random Forest

A **random forest** is a ML model which consists of a set of decision trees classifiers, where each tree votes for the class to be assigned to a given observation (Breiman 2001). Accordingly, a random forest grows each tree from a subset of the training data, selected randomly with replacement, a method also known as **bootstrapping**. At the same time, a subset of m attributes is also picked randomly from the set of predictive features for splitting each node, being m held constant for all trees, turning this algorithm faster than a single decision tree, which considers the full set of attributes. Contrarily to a single decision tree, all decision trees grow to their full extent (all leaves are pure), existing no pruning, and then a majority-voting method takes place. This process consists of each base learner voting for each observation in the training dataset to pertain to a certain class of the target variable and assigning that instance the class that gets the most votes. An example of a random forest diagram is shown in Figure 10.

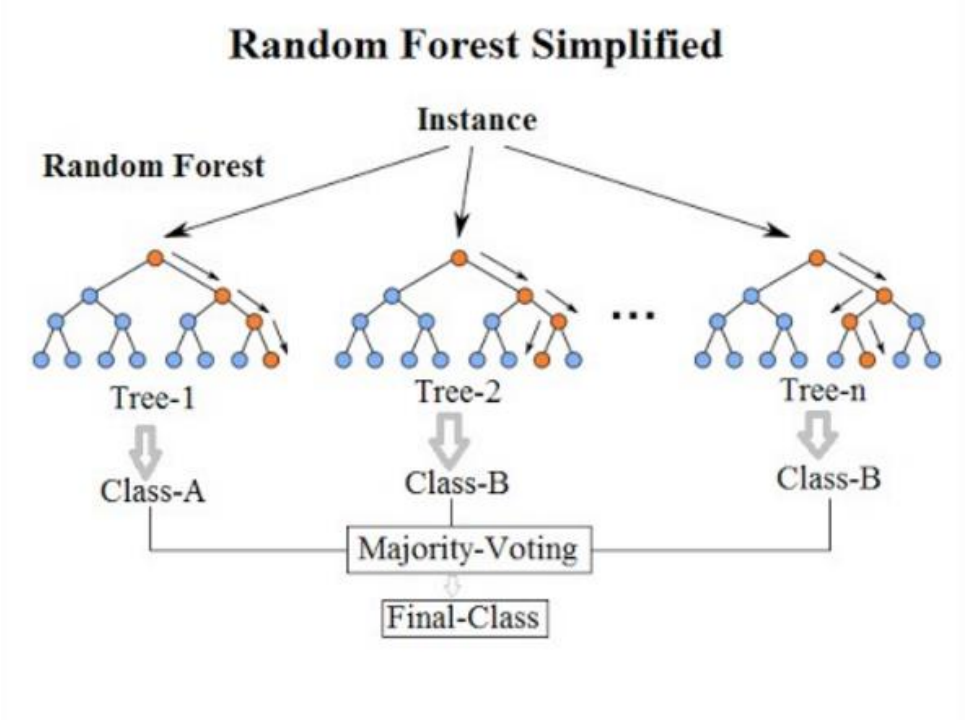


Figure 10 – Random forest diagram

On the other hand, for decreasing the forest error rate, one must consider the correlation between classifiers and their individual strength. Accordingly, a tree with a low error rate is a considered to have high predictive power, however, increasing the predictive power of each tree decreases the forest overall error rate. Thus, these metrics must be balanced, as a higher m increases both and a lower m decreases both, so finding the perfect m is of extreme importance, as this is the only hyperparameter which random forest algorithm is more sensitive to.

3.8.5. Gradient Boosting

Included in the ensemble algorithms' group, there are various **boosting** algorithms. According to Christopher M. Bishop, boosting is used to combine several models in order to create an overall classifier which is able to perform significantly better than any of the models which compose the boosting model (Bishop 2006). These base classifiers, also called weak learners, because they have an identical performance compared to random guessing (Schapire 1990), must be trained sequentially in order to account for examples which were misclassified by previous base learners, complementing each other. Therefore, instances which are well classified throughout this iterative training are given less weight, while data points which are wrongly classified successive base learners are given more weight. Once training is done, classifiers' predictions are then combined following a weighted majority voting scheme (Bishop 2006). Hence, combining the predictions of these weak learners, the boosting algorithm is able to produce accurate predictions, making it a strong classifier.

Gradient boosting is an example of boosting algorithms. Usually, this ensemble classifier is constituted of decision trees for base learners and its main goal, as it happens with all supervised learning algorithms, is to minimize a loss function. When predicting a binary target, this loss function is the negative binomial log-likelihood (Friedman 1999), which is represented in Equation 3.18:

$$L\big(y, F(x_n)\big) = \log(1 + e^{-2yF}) \tag{3.18}$$

where

$$F(x_n) = \frac{1}{2} log[\frac{P(y = 1\,|x_n)}{P(y = |x_n)}] \tag{3.19}$$

The minimization of this loss function is made successively by each base classifier of gradient boosting and so, as the learners are making predictions for a data point, they are getting closer to the true value of the observation, reducing the sum of residuals (difference between actual values and predicted values). It follows the same logic as gradient descent equation, being the predicted values updated according to Equation 3.20.

$$\hat{y}_m = \hat{y}_{m-1} + \eta(-\nabla \mathrm{L}(\mathrm{y}, \hat{y}_{m-1})) \tag{3.20}$$

where m refers to predictions made by the current base learner and $m - 1$ by the previous one.

This iterative process carries on until a threshold for minimizing the loss function is achieved, when the predictions do not improve enough towards the true observations' values.

4. SAS VIYA SOLUTIONS

4.1. SAS VIYA

SAS Viya is a cloud-native architecture suite of software solutions that deals with the increasing data complexity, constantly changing business needs and high computation power demand. This demand occurs due to a constantly incoming flow of data, which is accessed in an ad-hoc fashion, often by different departments in the same enterprise. On the same note, this cloud-based and scalable environment can answer that, having many benefits associated with its usage, such as tracking data lineage for identifying their sources and predictive models' versions more easily, as well as inspecting how the data has been transformed and by which people. Predictive models can also be managed and deployed into production in this set of applications after they were fine-tuned and their performance evaluated, being possible to develop them not only using SAS code, but also open source programming languages, improving developers' productivity. Its ability to work with data of various formats and sizes is also a valuable asset, enabled by servers (CAS) built to ensure the whole environment is operational and running without issues.

Hence, all of the advantages of SAS Viya lead to an improvement of analytical governance, so everyone in the organization has access to the same data in the same central location, serving data scientists, business analysts and application developers across multiple teams. At the same time, maintenance costs are lowered, as well as time wasted on finding integration errors between different technologies.

However, behind the scenes, there is an architecture designed to support those analytical tasks and enable the end-user to benefit from all features derived from SAS Viya usage. This architecture, represented in Figure 11, is comprised of several main components which are explained in detail in the next chapters.

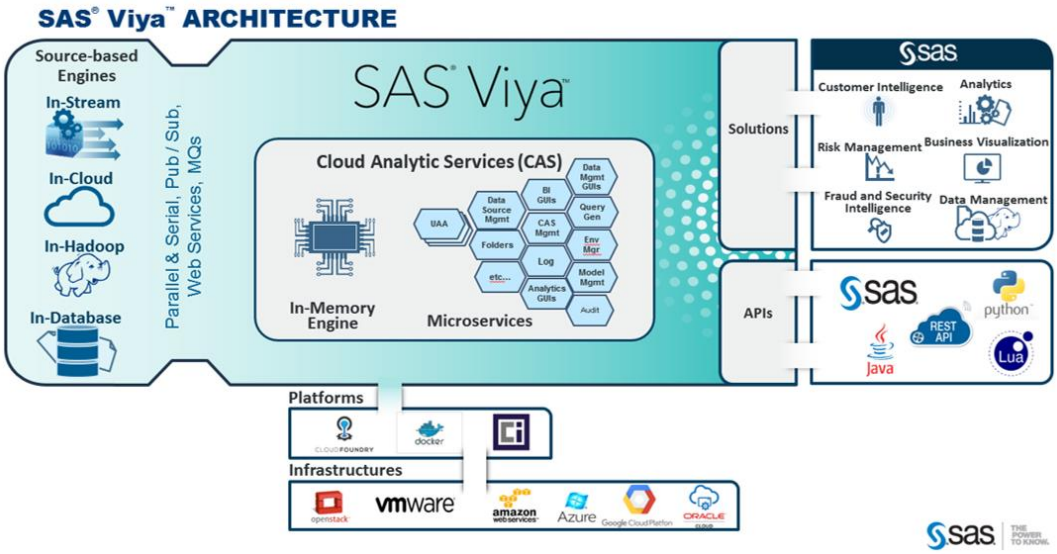


Figure 11 – SAS Viya architecture

4.1.1. Cloud Analytic Services

CAS is a server which can be comprised of a **single machine** (single-machine server architecture) or **several machines** (distributed server architecture) and both architectures allow for programs to be run in multiple threads, increasing performance, especially when data size is large or when data processing requires more computation power, allowing the server to process in-memory data in parallel, for faster loading times.

On the same note, a distributed server is comprised of several machines, one controller and one or more working nodes or machines. Client applications are linked with the worker nodes through the controller, which distributes the data available in-memory and loaded from the client applications into the worker nodes, ensuring different portions of the data are run in parallel, as represented in Figure 12. At last, the results are returned to the controller node in the order they were completed, as shown in Figure 13. Likewise, the server is scaled in a horizontal fashion, meaning the server can have more and more worker nodes as the workload and processing times increase.

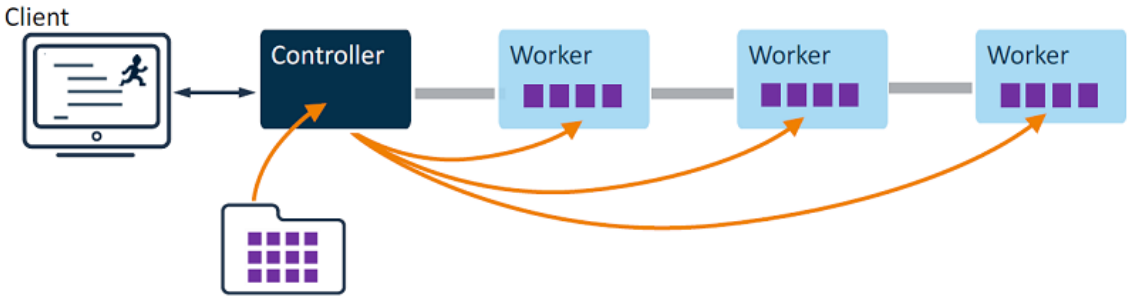


Figure 12 – Schema of the workload distribution in a distributed server

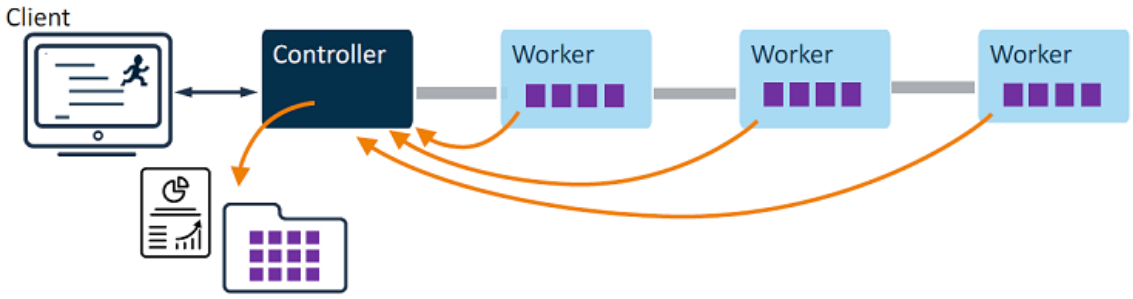


Figure 13 – Schema of the results returned to the controller

A backup controller might also exist in this type of architecture, acting as the primary controller when the latter fails until the server is restarted. If it does not exist and the primary controller fails, all in-memory data are lost, and the server is restarted. If both controllers fail, the server must be restarted ultimately. Furthermore, new connections to the server through the backup controller can be made, while the primary controller is down. In case a worker fails to communicate with the controller, one of the remaining workers takes a copy of the data processed by the lost worker and continues with the data processing.

On the other hand, the **single-machine server** does not have as much computation power as the distributed server. This type of architecture is somewhat different to the distributed server architecture, as there is only one machine dedicated to process the rows of the data available in-memory, acting as both controller and worker node. This single node makes use of various CPUs and threads in order to analyze the data, which is not run in parallel as in a distributed server and when the controller fails, all in-memory data tables are lost and the server is restarted, unlike a distributed server which can integrate a backup controller.

Whether a user opts for a single machine server version of CAS or a distributed one, it is possible to load data into memory from several data sources, namely from:

- SAS datasets.
- Local files formats: CSV, Excel, delimited text files.
- Databases: Hadoop, Impala, Oracle, PostgreSQL, Teradata.
- Social media feed: Twitter, Facebook, Google, YouTube.

4.1.2. Microservices

Another fundamental part of the SAS Viya environment is the **CAS microservices**. They are the backbone of the applications running within SAS Viya, being each microservice assigned to a different set of system functionalities, while enabling the deployment of the required set of services to be used by an application, excluding any other functionalities.

Although there are many microservices, it is worthy to mention the most important ones, such as:

- **SAS Logon** - tied to SAS Logon Manager which is a web application that manages users' authentication into SAS Viya applications.
- **Authorization** - manages the authorization rules for users to access the resources available in the SAS Viya environment.
- **Identities** - retrieves information about the users and groups in a SAS Viya deployment, as well as enables the creation and management of custom groups.
- **Configuration** - contains the health information and the configuration properties of the microservices, which can be altered in order to customize their functionalities.
- **Folders** - maintains a tree-based organization structure for SAS and external content.
- **Preferences** - enables storage and retrieval of user-preferences.

Running an entire SAS Viya deployment based on a microservices architecture can have many benefits. These work-units are not tied together, working independently from each other, thus, when a specific service fails, the remaining ones continue to work, hence only that same service is required to be restarted. Furthermore, it allows for services to be stopped and restarted at any given time, while it does not affect the functioning of the whole system. Start-up times are shortened, because when a SAS Viya application is deployed, only the microservices required for that application to run properly are deployed, whereas scaling is allowed easily, as many instances can be added to a service, spreading out across different machines, resulting in a highly available system.

Moreover, in order to trace the machines where the services' instances are located, SAS Viya resorts to the **SAS Configuration Server**, which stores the configuration settings for each microservice. This

server acts as a registration repository for all microservices deployed in the system, recording their location, so other microservices and clients can be able to communicate with them. Therefore, SAS Configuration Server can establish which microservice is responsible to receive APIs requests, acting as a central location for clients to scan information about the availability of any resource in the SAS Viya environment. How other applications and clients can connect to SAS Viya through microservices is explained in the next chapter.

4.1.3. SAS Viya REST APIs connections

SAS Viya REST APIs make use of HTTP authentication and verbs aiming to enable the communication between a client application and the microservice responsible for the incoming requests, granting access to each other's data and functionalities, as shown in the Figure 14. HTTP protocol is manipulated by web applications to read, update and delete resources. These web applications must be registered in the SAS Viya environment and can be based on any technology, such as some open-source programming languages (Java, Go, C, Python, R, Lua, among others).

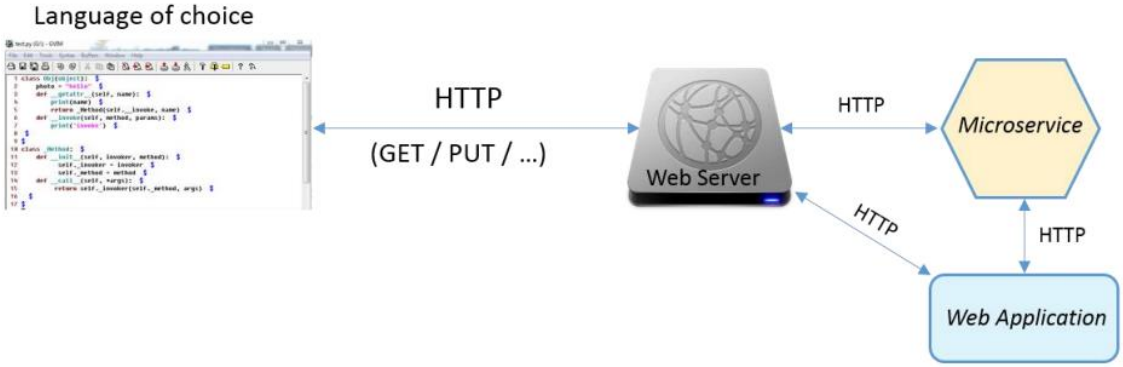


Figure 14 – Connection between a program code and a web application with a microservice

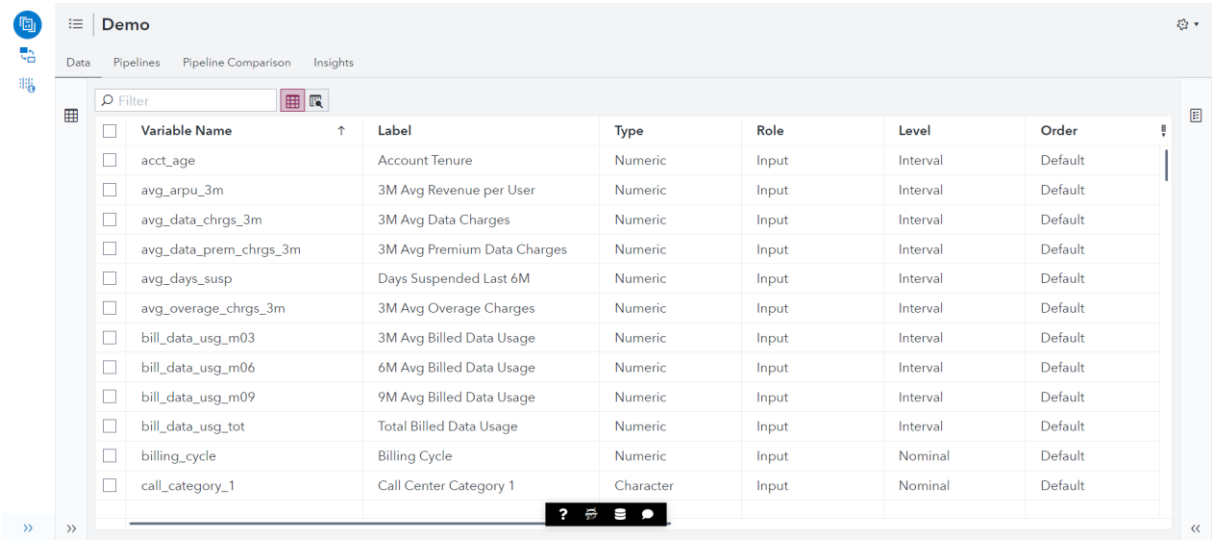
Nevertheless, the CAS Server, SAS Compute Server, microservices and SAS Viya REST APIs connections do not deliver *per se* the end results to a SAS Viya user, as they only support the SAS Viya capabilities. Instead, they support the deployment of several solutions which are designed to answer several business problems in various areas, such as Analytics, Customer Intelligence or Business Visualization, among others (see Figure 11), which provide a point-and-click interface for a user-friendly experience. Some of those applications' frameworks used in this report will be discussed in the next chapters, namely:

- SAS Model Studio.
- SAS VA.

4.2. SAS MODEL STUDIO

Developing a supervised ML project can be longstanding when it comes to perform several tasks within it manually. Nonetheless, these tasks can be performed in an automated fashion, resorting to an **AutoML** solution, which relieves both experienced data scientists and Data Science beginners from carrying out some of those repetitive tasks for instead allocating more time to introduce domain expertise into the process and focus more on searching for possible solutions to the business problem being studied. Thus, the overall time needed to conduct a supervised ML project takes less time, allowing the produced models to be put into production more rapidly.

SAS Model Studio is a tool which offers these many benefits. This SAS Viya built-in solution enables the creation of a supervised ML project, in order to compare and train analytical models using supervised ML algorithms to identify patterns in the data and predict the outcome of a target variable, using a user-friendly and low code user interface. The projects are displayed in a visual and interactive fashion to the user, as displayed in Figure 15.



Variable Name	Label	Type	Role	Level	Order
acct_age	Account Tenure	Numeric	Input	Interval	Default
avg_arpu_3m	3M Avg Revenue per User	Numeric	Input	Interval	Default
avg_data_chrgs_3m	3M Avg Data Charges	Numeric	Input	Interval	Default
avg_data_prem_chrgs_3m	3M Avg Premium Data Charges	Numeric	Input	Interval	Default
avg_days_susp	Days Suspended Last 6M	Numeric	Input	Interval	Default
avg_overage_chrgs_3m	3M Avg Overage Charges	Numeric	Input	Interval	Default
bill_data_usg_m03	3M Avg Billed Data Usage	Numeric	Input	Interval	Default
bill_data_usg_m06	6M Avg Billed Data Usage	Numeric	Input	Interval	Default
bill_data_usg_m09	9M Avg Billed Data Usage	Numeric	Input	Interval	Default
bill_data_usg_tot	Total Billed Data Usage	Numeric	Input	Interval	Default
billing_cycle	Billing Cycle	Numeric	Input	Nominal	Default
call_category_1	Call Center Category 1	Character	Input	Nominal	Default

Figure 15 – Data tab view of a project

A SAS Model Studio project has 4 main tabs, which can be accessed by a user to manage the project's metadata. The first one, the **Data** tab, shows the set of variables of the data source table in a project, being possible to change their settings, namely their **type** (numeric or character), **role** (input, target, ID, text, rejected and partition), **measurement level** (binary, interval, nominal or ordinal) and the **order** of the values (descending or ascending). It also displays some statistics for each variable, such as the **number of levels** (number of classes for class target variables), **number of missing values**, as well as the **minimum**, **maximum** and the **mean** values.

After all metadata modifications are done, the user must follow to the next tab, named **Pipeline**. It allows to build and execute one or more pipelines, each one representing a process flow diagram composed by a sequence of individual nodes, where each corresponds to one or more analytical tasks, namely data exploration, data preparation, model selection and evaluation. Pipelines can be manually generated by the user, as developed in the scope of this report, however, they can also be created by selecting a pipeline from a set of various pre-built templates, made available by SAS, as represented in Figure 16. These pre-developed pipelines range from more basic templates to more advanced and complex ones, regarding the number and variety of tasks included in the pipeline, and they can suit both classification and regression problems.

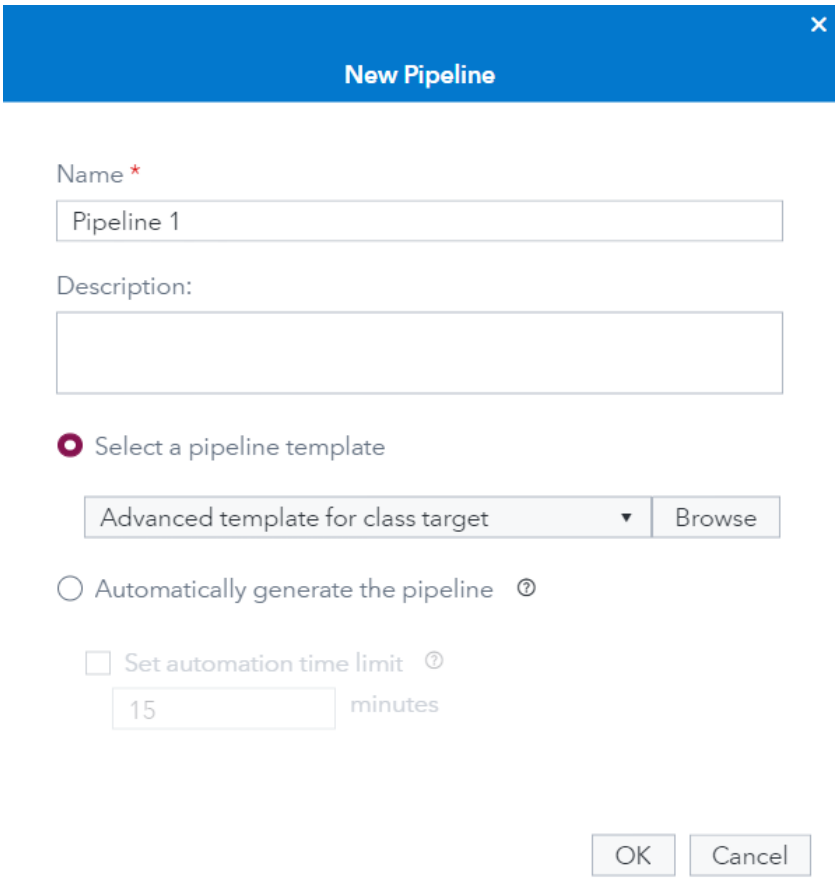


Figure 16 – New pipeline template settings

On the other hand, as an AutoML software, the user can also automatically create a new pipeline based on the given data and an automation time limit, an option available to choose from in the **New Pipeline** window, as seen in Figure 16, providing the chance to every user to develop their own supervised ML workflow.

Still within the Pipeline tab, this AutoML software also enables for automated hyperparameter tuning, also known as **autotuning**, which is the process of finding the optimal values for the hyperparameters of a model, resorting to an algorithmic search. This feature is enabled for six different types of models, namely **Neural Network**, **Decision Tree**, **Random Forest**, **Gradient Boosting**, **Bayesian Network** (for class target variable) and **SVM** (for binary target variable). The **Model Composer** node was specifically developed to perform that task, by enabling the autotuning of multiple models in parallel. The user can customize the properties of this node, such as defining the number of autotuning rounds, the cross-validation method and number of folds, as well as the assessment metric to be improved, as shown in Figure 17. At its completion, information about the champion model, in other words, the model with the best performance, its set of optimal hyperparameters values, as well as a set of charts to assess the models' performance, namely fit statistics, ROC and lift charts. Additionally, it can also produce several charts and statistics to evaluate the model interpretability, including the relative importance of the input variables of a model.

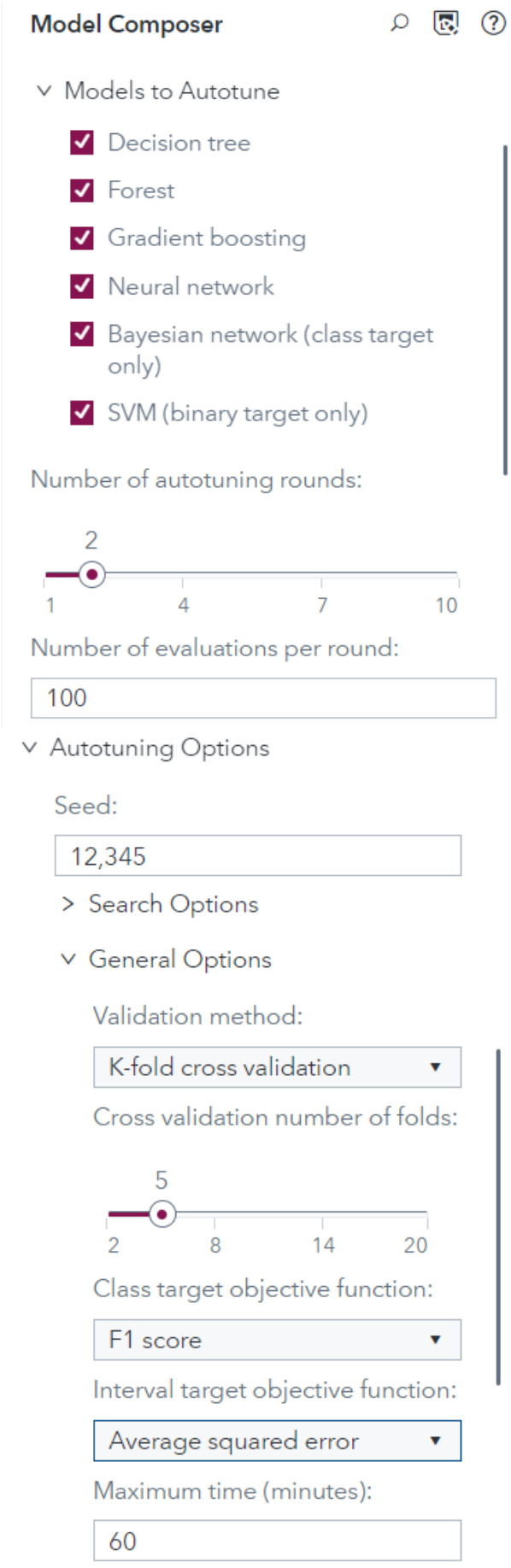

Model Composer
Models to Autotune
Decision tree
Forest
Gradient boosting
Neural network
Bayesian network (class target only)
SVM (binary target only)
Number of autotuning rounds:
2
1
4
7
10
Number of evaluations per round:
100
Autotuning Options
Seed:
12,345
Search Options
General Options
Validation method:
K-fold cross validation
Cross validation number of folds:
5
2
8
14
20
Class target objective function:
F1 score
Interval target objective function:
Average squared error
Maximum time (minutes):
60

Figure 17 – Example of a Model Composer node properties

Once the pipelines are fully developed and executed by the user, the **Pipeline Comparison** should be checked. By default, it displays the results for champion models within a pipeline or between multiple ones, however, one can also compare champion models with their challengers. Furthermore, it produces a table with fit statistics, as well as ROC and lift charts, in order to assess the models' performance on both training and validation sets.

Finally, a project's report is generated by SAS Model Studio in the **Insights** tab, containing several charts to compare the models in terms of assessment metrics, most common variables selected by the models, along with the set of the most important variables for the champion model and its cumulative lift. It also provides a project summary informing the user about the champion model and the user can add project notes to the report as well.

4.3. SAS VISUAL ANALYTICS

SAS VA is another solution which is part of the SAS Viya ecosystem. This low-code and self-service tool allows the user to perform **data exploration**, **reporting** and **predictive modeling** using the same platform, although in this report it is mainly used for data exploration. Its visual, drag-and-drop and point-and-click interface, along with its capability to handle and analyze huge amounts of in-memory data with the help of CAS, permits to discover hidden patterns in the data, improve business decisions and predict future outcomes.

Therefore, SAS VA has many advantages derived from its usage, namely:

- Implementation of dashboards (an example can be visualized in Figure 18) going through the supervised ML project steps in a single platform.
- Both non-experienced professionals and experts in analytics, from decision makers and business analysts to statisticians and data scientists, can use SAS VA.
- Wide variety of graphs and algorithms that enables the user to uncover data insights faster and more easily, due to its low-code interface.

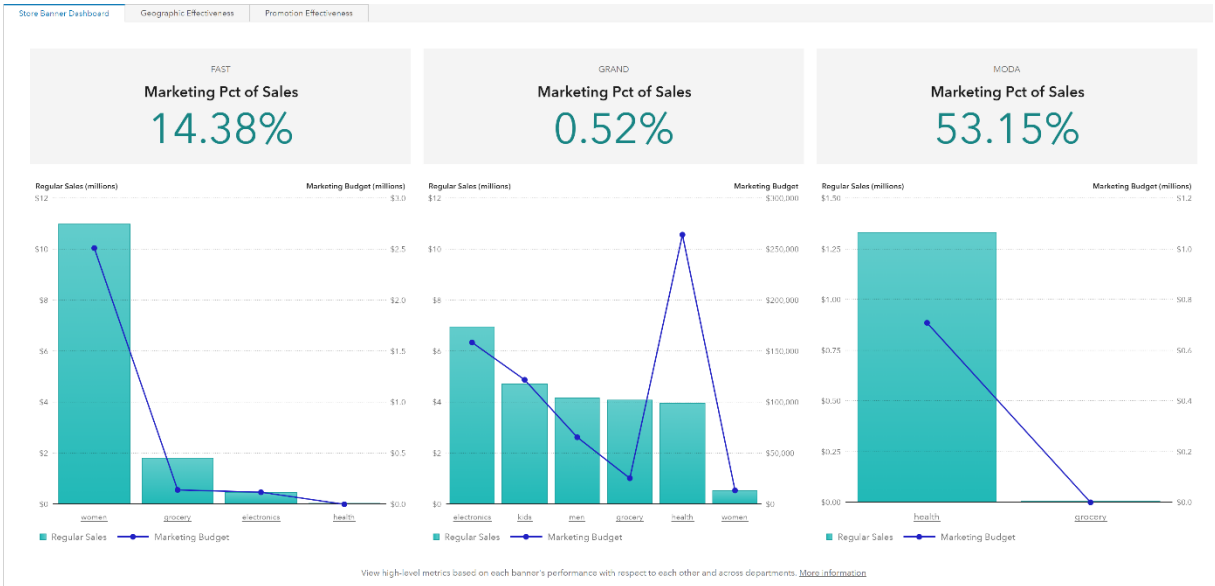


Figure 18 – Example of a dashboard in SAS VA

On the same note, this tool allows to quickly create a visualization through the auto charting feature of SAS VA. It considers the amount of data and its type for producing the most suitable visualization, helping non-technical users to understand the data more easily, although this automatically made graphic can be changed to another one, according to the user's preference.

Its interface is organized in reports, which can have several pages, which might contain many different objects, such as:

- Charts and graphs used for data exploration (e.g. scatter plots, bar charts, histograms, boxplots, correlation matrices, frequency tables, text clouds, among others).
- Visual representations of predictive and forecast models.
- Geographical data representations (maps, geographical coordinates, among others).
- KPIs, text, images and web content for enhancing the visual content of the dashboard.

5. OTHER TECHNOLOGIES

In this chapter, other technologies mentioned in this report are detailed, namely Jupyter Notebook and Python. They were chosen to compare their performance, as well as advantages and disadvantages comparatively to SAS Model Studio and SAS VA solutions due to two main reasons, namely the fact these two technologies were the most commonly used tools throughout the DSAA-BA Master's Masters' degree courses for conducting Data Science projects; secondly, due to Python being the most used programming language in the industry.

5.1. JUPYTER NOTEBOOK

Jupyter Notebook is a browser-based tool which displays the outputs in an interactive fashion, comparatively to the more common console-based solutions. Thus, a user can develop, document and execute code through this application, attached with objects which support its communication to the end-user. Accordingly, there are five main components which constitute the tool system, which is represented in Figure 19.

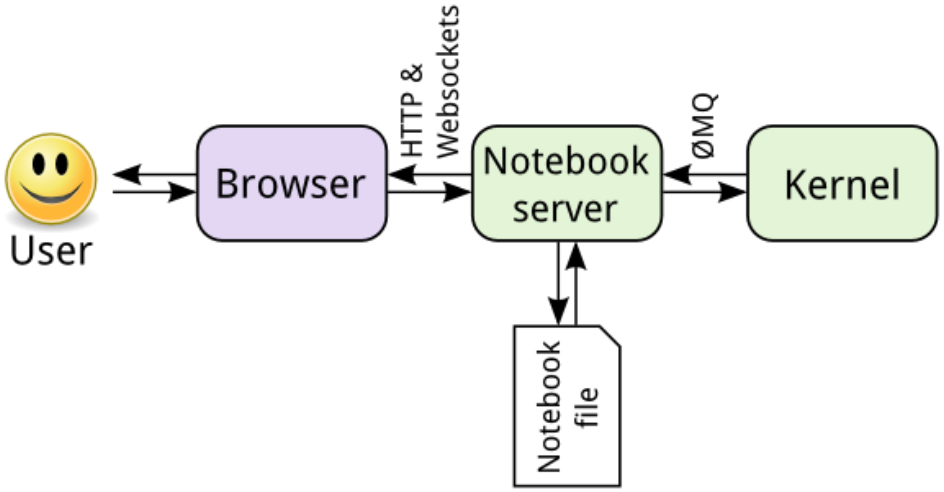


Figure 19 – Jupyter Notebook environment schema

The **in-browser application** enables to execute and edit code on the web, accompanied with its output, whether it is plain code or rich media representations, such as plots and images in various formats (HTML, PNG or SVG). It is possible to comment the code, resorting to the Markdown markup language.

On the other hand, **notebook documents** gather all content visible through the web application in a single representation, such as code and outputs, Markdown comments and media representations., whereas the **notebook server** is responsible for saving and loading notebooks on disk, communicating with the browser to do so.

Another fundamental part of this environment is the **terminal IPython**, which runs in the terminal, where the user is prompted to submit code, which is read and evaluated for printing a response. An interface is also used by all the other components of the Jupyter notebook environment, known as **IPython Kernel**. It is responsible for disk, task and memory management, running user code, as well as serving as the communication bridge between notebooks and the terminal IPython with the hardware.

Thus, tasks are executed depending on the memory and CPU availability and on which process assigned by the kernel to be allocated for task execution.

5.2. PYTHON

Python was used throughout this project embedded in Jupyter Notebook, which provides the programing interface for using this open-source technology. Python is an object-oriented programming language, meaning software design is organized around objects, which represent data which can be manipulated, ranging from real-world entities (e.g. name, age or e-mail of a company's employee) to computer programs.

Moreover, Python has several benefits derived from its usage and application, such as:

- Its simple syntax makes programs easier to write and read.
- Ability to work with data in many different formats, namely numbers, strings, lists, dictionaries, data frames, among other formats.
- It supports error raising and handling, warning the user of errors and making them easier to spot and correct.

Besides Python's standard library which comes with many default programming tasks, one can also resort to the various libraries and packages developed for handling other tasks, such as the ones used throughout this report and detailed below in Table 1:

Library	Description and Usage
pandas	Used for manipulating dataframes, enabling reading and writing data of different formats, merging dataframes, aggregating data, adding and deleting dataframe columns, subsetting dataframes, among other data transformations.
numpy	Used for scientific computing in python, providing several operations, such as mathematical, statistical, linear algebra, logical, sorting, among others. These transformations are performed on arrays and matrices.
scipy	Based on NumPy, this package also makes available several commands for manipulating data in arrays and matrices, as well as several mathematical, statistical and linear algebra transformations. However, it adds some more high-level functions not provided by NumPy.
matplotlib	Used for customizing various visualizations, whether static, animated or interactive. according to the user's preferences with the help of matplotlib functions.
seaborn	Derived from matplotlib, it enables to plot many visualizations for understanding the data contained in pandas dataframes. Barplots, histograms, boxplots and heatmaps were produced with this library in the context of this report.
phik	Gives access to several functions related to the Phik correlation coefficient, including the phik correlation matrix.
scikit-learn (sklearn)	Provides a wide range of functions for ML problems. It enables to build classification and regression predictive models, as well as resort to clustering and dimensionality reduction techniques. Several steps of a ML project can also be handled using this package, such as feature engineering and selection, model selection and evaluation.
scikitplot	Supports the quick creation of visualizations specifically designed for ML.

Table 1 – Python libraries used in this report

6. METHODOLOGY

This report's methodology follows the structure presented in Figure 20 and it was replicated for both SAS Model Studio and Jupyter Notebook/Python, in order to showcase how a supervised ML pipeline can be built in SAS Model Studio and which ML techniques each environment offers to the end user. At the same time, a similar ML pipeline using the same dataset was developed in Jupyter Notebook/Python, to further assess their differences. Thus, each methodology step is detailed for each technology throughout this chapter.

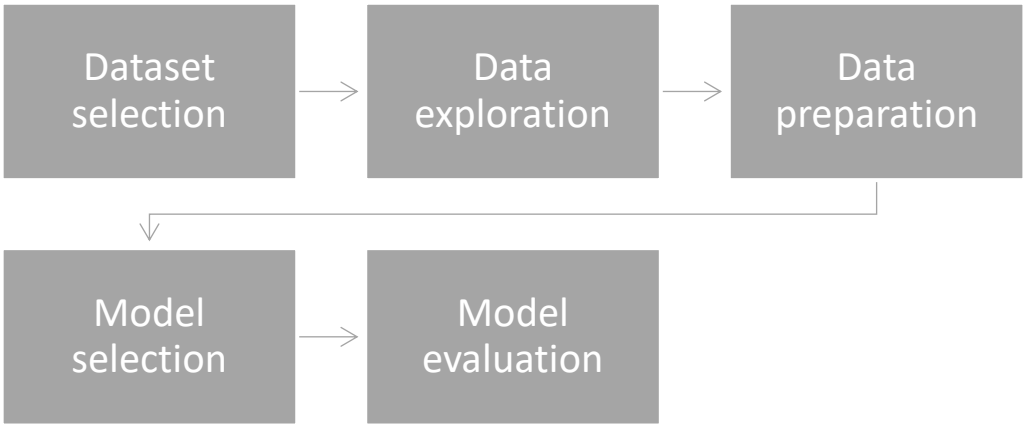


Figure 20 – Methodology stages

6.1. DATASET

The dataset selected for the practical case presented in this report is composed of 10,554 observations and 18 variables, containing data about marketing campaigns made by a Portuguese banking institution towards their customers. Several socioeconomic, previous and current campaigns-related attributes are included in the dataset, which are used to predict if a customer subscribes a term deposit, in the context of this supervised ML project. Those features are detailed in Table 2:

Variable	Type	Description
Age	Interval	Customer's age
balance	Interval	Current account balance
campaign	Interval	Number of contacts performed during the current campaign for a customer
contact	Nominal	Contact communication type
customer_id	Interval	Unique identifier for a customer
day	Nominal	Last contact day of the week
default	Binary	Has credit in default?
duration	Interval	Last contact duration (in seconds)
Education	Nominal	Education degree
housing	Binary	Has housing loan?
JOB	Nominal	Type of job
loan	Binary	Has personal loan?
martial	Nominal	Marital status

month	Nominal	Last contact month
pdays	Interval	Number of days since the last contact from a previous campaign (-1 means the customer was not previously contacted)
poutcome	Nominal	Outcome of the previous marketing campaign
previous	Interval	Number of contacts performed before the current campaign for a customer
y	Binary	Has the customer subscribed a term deposit?

Table 2 – Features in the dataset and corresponding descriptions

6.2. DATA EXPLORATION

6.2.1. SAS Model Studio/SAS Visual Analytics

In **SAS Model Studio**, the user must firstly create a new project before building a supervised ML pipeline. As mentioned before, one can choose a pipeline template, let SAS Model Studio develop a pipeline automatically based on the given data or start a pipeline from scratch. The last option was preferred, by selecting the option **Blank template** in the **Template** box, as shown in Figure 21. Additionally, the pipeline must have a data source, which must firstly be loaded into CAS memory, whose name is labeled in the **Data** box after the dataset is loaded.

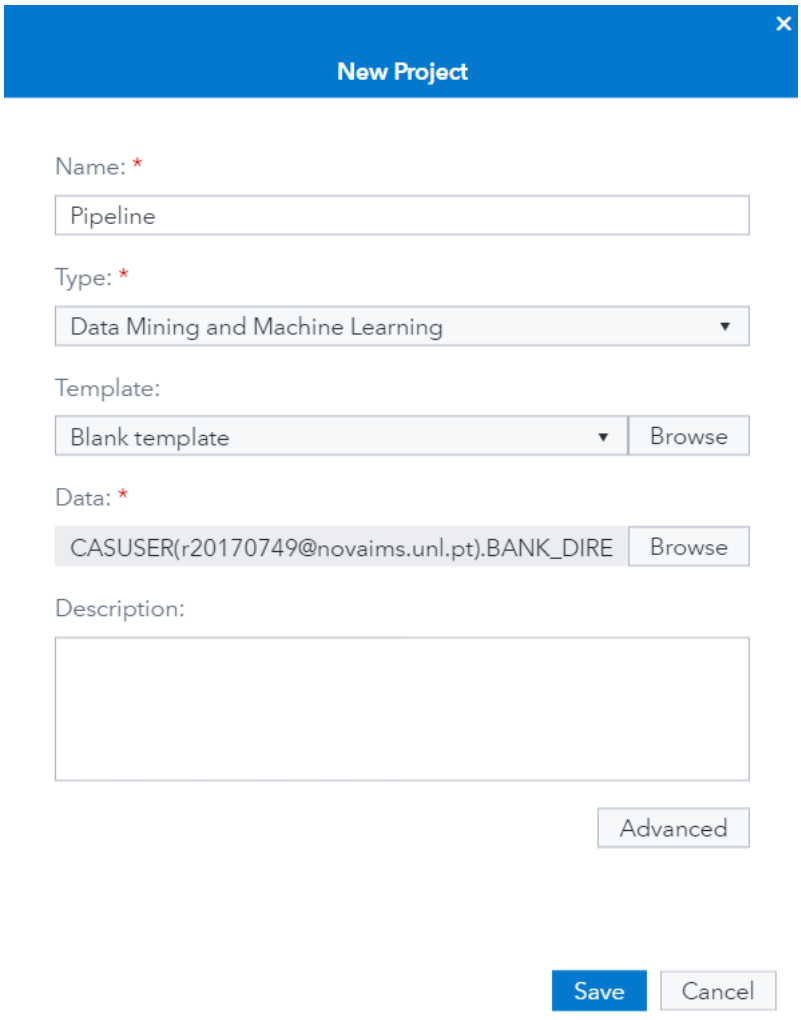


Figure 21 – New project window

Besides defining the projects' metadata, there are some **advanced settings** which need to be pre-defined as well, such as:

- Defining the percentage of the original data to be split into training and validation set in the **Data Partition** group, as shown in Figure 22.
- In the **Advisor Options** group, the maximum number of categories per categorical variable can be set, as well as the minimum number of unique values for a variable to be considered as interval. The maximum percentage of missing values a variable can also be defined in this group.
- In the **Event-Based sampling** group, the user can perform over-sampling or under-sampling, maintaining 100% of cases with the target variable class to be predicted.

Nevertheless, for the purpose of this report, the settings for the two last mentioned groups were not changed, since the variables' types can be defined during the project development, as it is detailed in this chapter, as well as it is not needed to perform event-based sampling due to the two classes of the output variables having a similar number of observations.

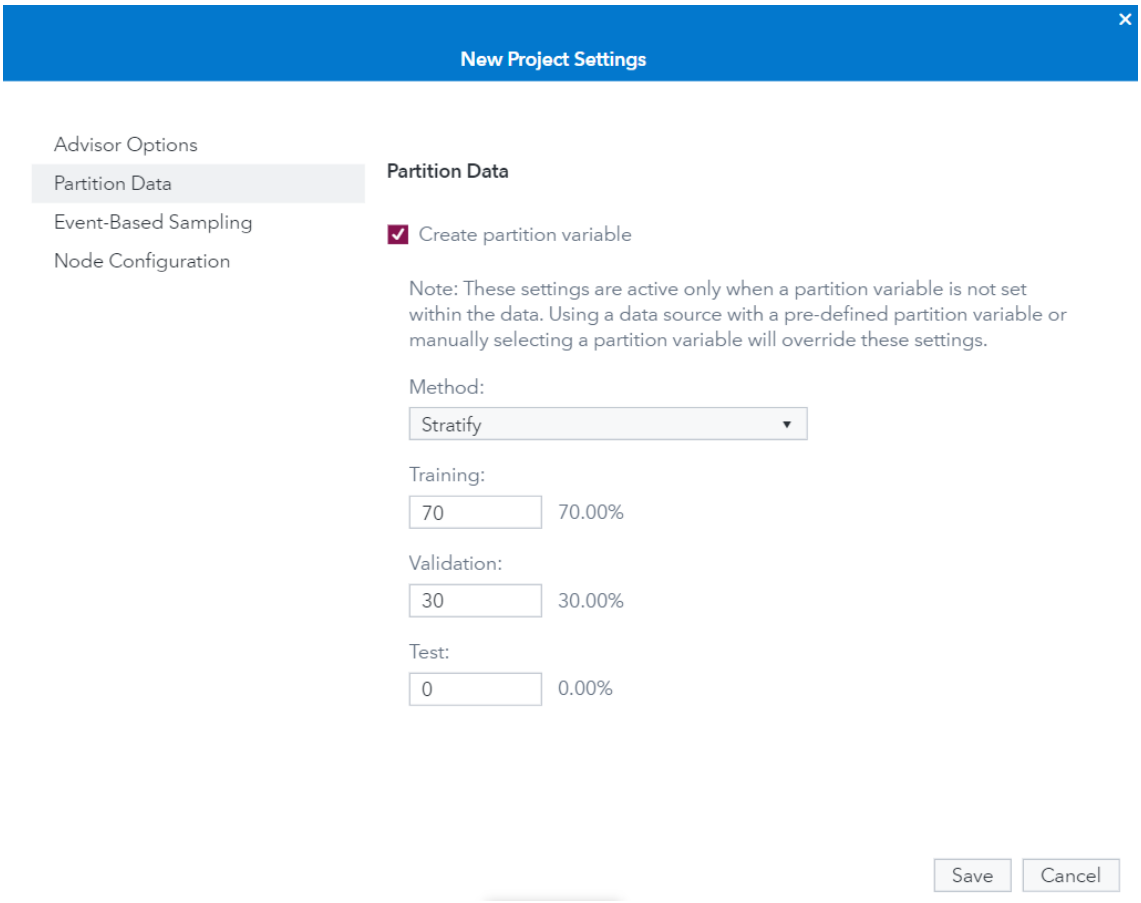


Figure 22 – Partition Data group settings

After the project is created and all the needed settings are fulfilled, samples of the data source can be viewed in the **Data** tab. For instance, a sample of the top 15 customers of the dataset taken from SAS Model Studio is showed in Figure 23 and Figure 24.

Education	martial	JOB	Age	customer_id	default	balance	housing	loan	contact	day
tertiary	single	manage…	30	122482	no	347	no	no	cellular	22
primary	married	blue-col…	60	119725	no	3462	no	no	cellular	7
tertiary	single	manage…	40	103490	no	157	yes	no	unknown	15
tertiary	married	entrepr…	45	126218	no	3689	yes	no	cellular	19
primary	married	blue-col…	38	104835	no	0	yes	yes	unknown	20
secondary	married	services	34	132649	no	703	no	no	cellular	17
tertiary	single	technician	31	130037	no	665	yes	no	cellular	4
secondary	single	manage…	28	128633	no	1861	no	no	cellular	29
secondary	married	services	43	109028	no	1450	yes	no	unknown	4
primary	married	blue-col…	52	120811	no	2488	no	no	cellular	13
tertiary	single	manage…	32	111335	no	7419	yes	no	unknown	18
secondary	married	blue-col…	49	124023	no	102	no	no	cellular	29
primary	married	unempl…	30	126890	no	1379	no	yes	cellular	20
unknown	divorced	unknown	66	140581	no	1993	yes	no	cellular	6
secondary	single	admin.	29	130363	no	1261	no	no	cellular	5

Figure 23 – Sample of the first 15 customers in the dataset in SAS Model Studio

month	durat…	cam…	pdays	previ…	pout…	y
aug	229	2	-1	0	unknown	no
aug	125	2	-1	0	unknown	no
may	68	2	-1	0	unknown	no
nov	517	2	2	3	failure	no
may	165	2	-1	0	unknown	no
apr	282	2	1	2	failure	no
feb	156	1	-1	0	unknown	no
jan	43	2	-1	0	unknown	no
jun	227	2	-1	0	unknown	no
aug	176	2	-1	0	unknown	no
jun	248	29	-1	0	unknown	no
aug	210	2	-1	0	unknown	no
nov	172	2	-1	0	unknown	no
jul	117	4	-1	0	unknown	no
feb	605	1	-1	0	unknown	no

Figure 24 – Sample of the first 15 customers in the dataset in SAS Model Studio

After the dataset is visualized, the **variable identification** step must be carried out. It involves checking and modifying the variables' types if needed, as well as dropping some variables which are out of the scope or not needed for predicting the target variable. Accordingly, there were some changes to be made, resorting to the **Data** tab, being them the following:

- In SAS Model Studio, **duration** role was changed from **input** to **rejected**, because at the time the call is performed, the duration is not known. Therefore, **duration** is not considered for the remainder of the tasks in the pipeline. **customer_id** role was set to **ID**.
- The type (level in SAS Model Studio) of **pdays** changed from **nominal** to **interval**, while the level of day was modified to **nominal**.

In SAS Model Studio, there is a node which was designed to take care of the first stage of a ML project, named **Data Exploration**. This node, represented in Figure 25, is able to display several graphs and tables with several statistics for performing both **univariate** and **bivariate descriptive analyses** which can be replicated for all variables in the dataset. Some examples of those visualizations are shown in Appendix 1, from Figures 40 to 45, such as:

- Number of observations.
- Number of missing values per variable.
- Table with the interval variables moments, displaying some statistics for each interval variable.
- Barplots for class variables.
- Barplots for interval variables (values splitted in bins).
- Target variable by categorical variables' crosstabulations plots.

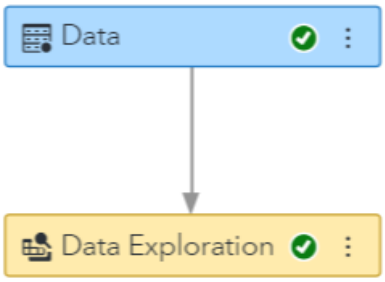


Figure 25 – Pipeline designed for data exploration

However, there are some graphs which are not available for analysis in the Data Exploration node results, such as **boxplots** and the **Pearson correlation coefficient matrix**, being needed to resort to SAS VA (see Figures 46 to 48 in Appendix 1) for producing these visualizations.

Therefore, by looking at the tables and graphs produced by SAS Model Studio and SAS VA, it is possible to get some takeaways, enabled by their user-friendly appearance, such as:

- **JOB**, **day** and **month** have high cardinality (more than 5 categories).
- **balance**, **campaign** and **previous** have high kurtosis. On the other hand, all features have skewed distributions, except for **Age**.
- There are no missing values.
- **Age**, **balance**, **campaign** and **previous** have outliers.
- 65.3% of the customers which are on default did not subscribe to a term deposit for the current campaign.
- 64.6% of the customers which have a loan did not subscribe to a term deposit
- 61.4% of customers which have a housing loan do not subscribe to a deposit, whereas 60.3% of the customers which do not have a housing loan subscribe to a deposit.
- 68.2% of the retired people and 71.7% of the students in the dataset subscribed to a term deposit, while 63% of people with blue-collar jobs do not subscribe to a term deposit.

- 75.7% of customers whose contact communication way is unknown do not subscribe to a term deposit.
- More than 80% of the people which were contacted in December, March, September and October subscribed to a term deposit, while in May 64.5% of customers did not.
- 92.7% of the customers which subscribed to a term deposit in the previous marketing campaign subscribed again.
- Only **previous** and **pdays** have some correlation with each other among the set of continuous variables, corresponding to 0.41.

6.2.2. Jupyter Notebook/Python

Contrarily to SAS Model Studio, in Jupyter Notebook/Python it is only needed to import the dataset as a **Pandas dataframe** in the notebook, not being needed to create a pipeline. Using this tool, it is also possible to visualize samples of different sizes, such as the one represented in Figure 26 and Figure 27.

	Education	martial	JOB	Age	customer_id	default	balance	housing	loan	contact	day
0	tertiary	single	management	30	122482	no	347	no	no	cellular	22
1	primary	married	blue-collar	60	119725	no	3462	no	no	cellular	7
2	tertiary	single	management	40	103490	no	157	yes	no	unknown	15
3	tertiary	married	entrepreneur	45	126218	no	3689	yes	no	cellular	19
4	primary	married	blue-collar	38	104835	no	0	yes	yes	unknown	20
5	secondary	married	services	34	132649	no	703	no	no	cellular	17
6	tertiary	single	technician	31	130037	no	665	yes	no	cellular	4
7	secondary	single	management	28	128633	no	1861	no	no	cellular	29
8	secondary	married	services	43	109028	no	1450	yes	no	unknown	4
9	primary	married	blue-collar	52	120811	no	2488	no	no	cellular	13
10	tertiary	single	management	32	111335	no	7419	yes	no	unknown	18
11	secondary	married	blue-collar	49	124023	no	102	no	no	cellular	29
12	primary	married	unemployed	30	126890	no	1379	no	yes	cellular	20
13	unknown	divorced	unknown	66	140581	no	1993	yes	no	cellular	6
14	secondary	single	admin.	29	130363	no	1261	no	no	cellular	5

Figure 26 – Sample of the first 15 customers in the dataset in Jupyter Notebook/Python

month	duration	campaign	pdays	previous	poutcome	y
aug	229	2	-1	0	unknown	no
aug	125	2	-1	0	unknown	no
may	68	2	-1	0	unknown	no
nov	517	2	2	3	failure	no
may	165	2	-1	0	unknown	no
apr	282	2	1	2	failure	no
feb	156	1	-1	0	unknown	no
jan	43	2	-1	0	unknown	no
jun	227	2	-1	0	unknown	no
aug	176	2	-1	0	unknown	no
jun	248	29	-1	0	unknown	no
aug	210	2	-1	0	unknown	no
nov	172	2	-1	0	unknown	no
jul	117	4	-1	0	unknown	no
feb	605	1	-1	0	unknown	no

Figure 27 – Sample of the first 15 customers in the dataset in Jupyter Notebook/Python

The **variable identification** step was also executed in Jupyter Notebook/Python, where similar changes made in SAS Model Studio were applied to the data:

- **duration** was eliminated from the dataset and **customer_id** was set as the index column.
- **pdays** changed its type from **nominal** to **interval** and the type of **day** was modified to **nominal**.

Unlike SAS Model Studio, there is not a way to conduct automatically univariate and bivariate descriptive analyses, therefore, one needs to resort to hard code to produce the same visualizations in Jupyter Notebook/Python, being them shown from Figure 49 to Figure 56 in Appendix 2. Nevertheless, the same conclusions taken from SAS Model Studio and SAS VA can be drawn through the analysis of Jupyter Notebook/Python visualizations.

6.3. Data Preparation

6.3.1. SAS Model Studio/SAS Visual Analytics

SAS Model Studio allows both **customized** data preparation steps and for a more **black-box** approach, letting the software find on its own the best data preparation techniques to apply to data. All these tasks are represented in Figure 28, which shows the pipeline that was used for this report.

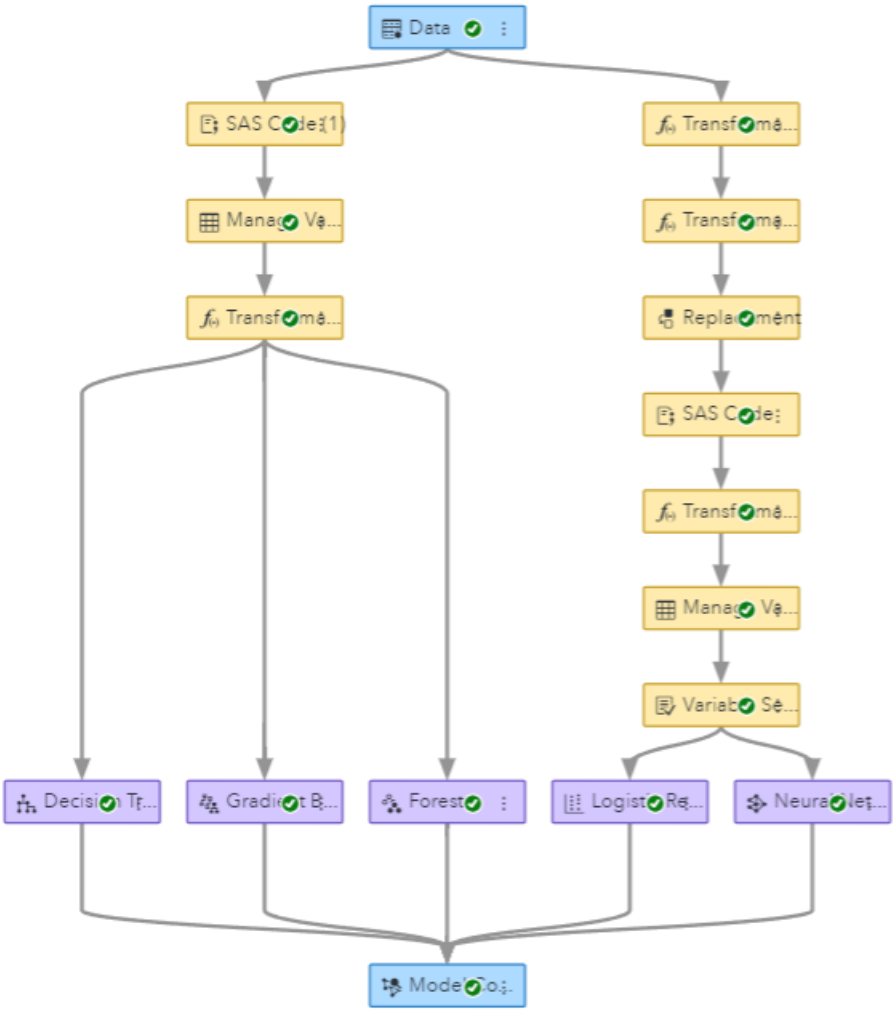


Figure 28 – Supervised ML project pipeline

After the pipeline is created, the data preparation stage must be carried out following the usual steps, aiming to solve the data issues found when first exploring the data. Different types of data preparation methods were developed for preparing the data to train **logistic regression** and **neural network** models, as well as **tree-based** models (Decision Tree, Gradient Boosting and Random Forest). However, the latter do not need so much data preparation, as they can perform well even in the presence of outliers, missing values or different data scales, thus, two different ordered sets of data preparation tasks were developed for both groups. On the same note, SAS Model Studio applies those transformations to the data to both training and validation sets, although every node in the pipeline outputs the results only for the training dataset.

Having that in mind, the data preparation stage for logistic regression and neural network starts with the **Transformations** node. The user is able to apply data transformations to both continuous and categorical variables using this node, aiming to stabilize variances, correct non-normality or standardize the data. Accordingly, two Transformations nodes were added to the pipeline, each one aiming to reduce skewness and kurtosis for continuous input features, by applying the **Best** transformation, which performs several transformations for each variable based on a criterion (for these two nodes, the criterion were **Moment skewness** and **Moment kurtosis**, respectively). In the end, the transformation with the best Chi-square test relatively to the target variable is chosen. For the first Transformations node, 5 new variables were created, namely:

- INVSQRT_pdays, the inverse square root transformation for variable pdays.
- INV_campaign and INV_previous, the inverse function of campaign and previous, respectively.
- LG10_balance, the logarithm base 10 transformation for balance.
- SQRT_Age, the square root of Age.

For the second Transformations node, only 4 new variables were generated from the set of features created in the previous Transformations node, since the software considered SQRT_Age to not suffer from high kurtosis:

- INV_INVSQRT_pdays and INV_INV_campaign, the inverse variables of INVSQRT_pdays and INV_campaign, respectively.
- SQRT_INV_previous, the square root function of INV_previous.
- SQR_LG10_balance, the square transformation of LG10_balance.

After applying the transformations mentioned above, only **SQR_LG10_balance** suffers from high kurtosis, registering 7.99, existing no variable whose value for skewness is far from the desired value for this statistic, which is 0, as looking at both Transformations nodes results summaries, presented in Figures 57 and 58 in Appendix 3.

After correcting non-normality in the data, one must eliminate any outliers which might exist within the data. For achieving that, the **Replacement** node was added to the pipeline, which might be used not only to detect outliers for continuous features according to a predefined criterion, but also to replace unknown categorical variables classes with a missing value or the mode. However, all categories of categorical features were labeled from the start, thus this transformation was ignored. Otherwise, for continuous variables, outliers were identified as data points which lied above or below 3 standard deviations from the mean, considering the data distribution for each of those features was close to a normal distribution. Accordingly, 3 outliers were identified for **INV_INV_campaign**, 25 for **SQRT_Age** and 139 for **SQR_LG10_balance**, while **INV_INVSQRT_pdays** and **SQRT_INV_previous** had no outliers, as represented in Figure 59 in Appendix 3. Consequently, 5 new features were created, even though **INV_INVSQRT_pdays** and **SQRT_INV_previous** did not have any extreme observations (**REP_INV_INV_CAMPAIGN**, **REP_SQRT_AGE**, **REP_SQR_LG10_BALANCE**, **REP_INV_INVSQRT_PDAYS** and **REP_SQRT_INV_PREVIOUS**), where outliers were replaced by a missing value.

Following outliers' detection, they need to be deleted from the data, since this kind of observations were substituted by missing values. For that reason, a **SAS code** node was included in the pipeline, allowing for the user to manipulate the dataset, modify variable settings, build predictive models or create several visualizations. In this case, it was used to create a new variable called **delete_flag**, which identifies the data points to drop from the data, corresponding to **1** if so and **0** if to be kept in the dataset. Since the outliers are dropped from the dataset, looking at the boxplots of all continuous input variables might be important to detect if there are still some extreme observations. Thus, if one analyses the boxplots produced in SAS VA once again, it can be noticed that **REP_SQR_LG10_BALANCE** might still have some outliers.

In this same node, the feature engineering steps were applied to the data. Accordingly, high cardinality for categorical values, namely **JOB**, **month** and **day**, was handled, where categories were aggregated and transformed into new ones, therefore new variables were created as the following:

- JOB categories equal to **management**, **blue-collar**, **technician**, **admin**. and **services** were merged into a single class, called **well-paid job**.
- **Quarter** was created from month, being the months aggregated into quarters.
- From Day, the variable **days** was generated, splitting the days of a month into three periods (**1-10**, **11-20**, **21-31**).

In the same SAS Code node, **one-hot encoding** was applied to all categorical variables, turning them to be binary variables, whose names are identified as **ORIGINAL_VARIABLE_NAME_CATEGORY**. Consequently, the original categorical variables' roles were set to **rejected** in this node, meaning they are ignored in upcoming steps.

After outliers' elimination, high cardinality treatment and one-hot encoding, continuous variables need to be transformed in order to set them to the same scale to avoid logistic regression and neural network models to give more importance to features with higher values. Thus, another **Transformations** node was added, this time applying the **Standardization** transformation to the data, in order to change the data scale such that every continuous variable mean and standard deviation are set to **0** and **1**, respectively. In the results summary of this node, it can also be noticed that the standardized variable of **REP_SQR_LG10_balance** called **STD_REP_SQR_LG10_BALANCE** had its kurtosis reduced by a significant amount down to 1.52 (Figure 60 in Appendix 3).

Before proceeding to feature selection, it is necessary to check variables' types, roles and levels and change them if needed, since there were many variables which were created in previous steps. To do so, SAS makes available the **Manage Variables** node, which, in this case, enabled to set the role for **quarter** and **days** to **rejected**, as verified in Figure 61 in Appendix 3, since both were created in the SAS code node, which does not allow to create and reject a variable in the same SAS code node. All other features' metadata was not modified.

After feature engineering, **feature selection** is the step to be performed before predictive models can be trained. SAS also developed a node specifically designed for this task, called **Variable Selection**. Six techniques for selecting the subset of input features to train the supervised ML models are available with this node, namely:

- Unsupervised selection.
- Fast Supervised selection.
- Decision tree selection.
- Linear Regression selection.
- Forest selection.
- Gradient Boosting selection.

Unsupervised selection consists of detecting which subset of variables explain the maximum amount of variance. In the context of this project, the maximum number of features to consider for selection was 15, whereas the correlation between input features and an incremental variance cutoff which eliminates features below a specific threshold was set to 0.001.

Fast Supervised selection involves selecting the set of variables which explain the maximum amount of variance in the target variable, having the same conditions as unsupervised selection for adding a feature into the subset.

Decision tree selection was the third chosen method used. It trains a decision tree, aiming to identify which features the model selects according to a relative variable importance threshold, in this case, set to 0.02.

The remaining available methods were not employed, due to already existing an intrinsic feature selection method being used (decision tree selection), thus forest selection and gradient boosting selection were not applied. Linear regression method was not utilized as well, as linear regression is suited for regression problems and not classification problems as this one.

The **combination criterion** used to select the final subset of input features was **Selected by a majority**, so an input feature must only be selected if chosen by at least two of the methods listed above. Therefore, according to the outputs given by each method, the selected features were the ones listed in Table 3, which were retrieved from the results summary of the Variable Selection node (Figure 62 in Appendix 3):

Variable	Type	Description
CONTACT_UNKNOWN	Binary	1 if contact is unknown and 0 otherwise
EDUCATION_TERTIARY	Binary	1 if education is tertiary and 0 otherwise
HOUSING_NO	Binary	1 if housing is no and 0 if yes
JOB_RETIRED	Binary	1 if job is retired and 0 otherwise
LOAN_NO	Binary	1 if loan is no and 0 if yes
MARITAL_MARRIED	Binary	1 if marital is married and 0 otherwise
POUTCOME_SUCCESS	Binary	1 if poutcome is success and 0 otherwise
QUARTER_2	Binary	1 if quarter is 2 and 0 otherwise
QUARTER_3	Binary	1 if quarter is 3 and 0 otherwise
STD_REP_INV_INV_CAMPAIGN	Interval	Transformed campaign variable
STD_REP_SQR_LG10_BALANCE	Interval	Transformed balance variable

Table 3 – Features subset to train logistic regression and neural network in SAS Model Studio

On the contrary, for **tree-based models**, the presence of outliers, missing values and different data scales is not a problem. Nevertheless, one must still reduce the cardinality of categorical variables and perform **feature creation**, thus a **SAS code** node was added to perform those tasks. For decreasing the number of categories, the exact same code executed in the SAS code node added for the logistic regression and neural network part of the pipeline was added, as well as for feature creation. However, one-hot encoding was not employed to transform categorical features, but instead a label encoder called **Level Encoding** in SAS Model Studio, which simply assigns a numerical value to each class of a categorical variable. This encoder was used resorting to the **Transformations** node, due to categorical input variables, called **class inputs** in SAS Model Studio, needing to be encoded in order to be interpreted by tree-based models' algorithms. In addition to this, a **Manage Variables** node was also added in order to reject the variables **month** and **days**. Consequently, after the data preparation steps were carried out for preprocessing the data before training the decision tree, random forest and gradient boosting models, the subset of features in the training and validation sets consisted in variables listed in Table 4:

Variable	Type	Description
Age	Interval	Original Age
balance	Interval	Original balance
campaign	Interval	Original campaign
pdays	Interval	Original pdays
previous	Interval	Original previous
LEVENC_contact	Interval	Level encoded contact
LEVENC_days	Interval	Level encoded days
LEVENC_default	Interval	Level encoded default
LEVENC_Education	Interval	Level encoded Education
LEVENC_housing	Interval	Level encoded housing
LEVENC_JOB	Interval	Level encoded JOB
LEVENC_loan	Interval	Level encoded loan
LEVENC_martial	Interval	Level encoded martial
LEVENC_poutcome	Interval	Level encoded poutcome
LEVENC_quarter	Interval	Level encoded quarter

Table 4 – Features subset to train tree-based models in SAS Model Studio

6.3.2. Jupyter Notebook/Python

In Jupyter Notebook/Python, the data preparation phase was carried out in a slightly different fashion compared to SAS Model Studio, as there are some techniques which are not available in both tools, therefore some of them were put to use in Jupyter Notebook/Python, since they can enrich more the data relatively to the methods available in SAS Model Studio.

Moreover, correcting non-normality in the data was the first step for transforming the data used to train logistic regression and neural network, as in SAS Model Studio. Thus, the **PowerTransformer** class was applied to both training and validation sets, resorting to the **Yeo-Johnson** method, which was not applied to the interval variables in SAS Model Studio, despite being also available among the transformation's methods. Indeed, this transformation corrected non-normality in the data to some extent, as represented in Figure 29, even though **balance** still recorded a high kurtosis (41.84), similarly to what registered in SAS Model Studio.

	Kurtosis	Skewness
Age	-0.485708	0.014790
balance	41.841113	2.050818
campaign	-1.286294	0.267968
pdays	-0.812979	1.086616
previous	-0.736095	1.108064

Figure 29 – Kurtosis and skewness values for interval features

Outlier' detection and removal is the next task to proceed with, as it is easier to eliminate outliers after the data distribution of each input continuous feature is closer to a normal distribution. Therefore, the same method applied in SAS Model Studio was also used in Jupyter Notebook/Python, by removing

the observations below or above 3 standard deviations from the mean. 153 outliers were removed in the training dataset, meaning 2.1% of the total number of clients was eliminated, as shown in Figure 30. Obviously, the same technique was performed for the validation set.

```
Number of outliers: 153
Percentage of removed clients: 2.1%
```

Figure 30 – Number of outliers removed and corresponding percentage

However, when analyzing the boxplots of the transformed continuous input features, one can notice that the variable **balance** still has some outliers, which also happened in SAS Model Studio. It will affect some of the later transformations made to the data, such as **feature scaling**.

Following outliers' elimination, **high cardinality of categorical features** needs to be approached for both datasets to be trained by tree-based models and for logistic regression and neural network. In Jupyter Notebook/Python, the newly created categories are the same as in SAS Model Studio, since in both tools they were generated using custom code. Following that, the categorical features were encoded. However, they were encoded differently for the dataset further used to train tree-based models and the dataset for logistic regression and neural network. For the first dataset, the classes of categorical variables were assigned with numerical values, using the class **OrdinalEncoder**, which is a type of label encoder, while for the second dataset, they were **one-hot encoded**, a transformation performed by the **OneHotEncoder** class, which automatically creates new binary variables from the class of each categorical feature, instead of manually coding it, as in SAS Model Studio. The names of the variables produced for tree-based dataset and logistic regression and neural network dataset follow the same logic as the one-hot encoded variables created in SAS Model Studio, with the name of the original variable followed by the name of the class corresponding to the

Before feature selection, **feature scaling** step needs to be performed, thus the **RobustScaler** class was added to the code for transforming data scales for all continuous input features, since this transformation is prone to the existence of outliers which still might exist within the data, which were identified to still exist for the feature **balance**. Contrarily to Robust Scaler, standardization is more affected by outliers, which was used in SAS Model Studio due to Robust Scaler method not being available.

After all data transformations are performed, **feature selection** is the step to tackle, before going into the model selection and evaluation phases for logistic regression and neural network models. In Jupyter Notebook/Python, one of the methods used was RFE, which cannot be applied in SAS Model Studio in the context of a classification problem. The other techniques applied were decision tree features' importance, correlation within the set of input variables and correlation of input variables with the target using Phik correlation coefficient.

In the case of **RFE**, a decision tree was used to train the data and select which variables' subset had the highest accuracy in predicting the output variable, being this done over 5 iterations, where in each one a subset of features was selected. The subset of features with the highest accuracy was the one chosen to be combined with the other selection methods for selecting the final subset, being them represented in Figure 31.

```
array(['housing_no', 'loan_no', 'JOB_well-paid JOB', 'martial_divorced',
       'martial_married', 'martial_single', 'Education_primary',
       'Education_secondary', 'Education_tertiary', 'contact_unknown',
       'days_1-10', 'days_11-20', 'days_21-31', 'quarter_1.0',
       'quarter_2.0', 'quarter_3.0', 'quarter_4.0', 'poutcome_success',
       'Age', 'balance', 'campaign', 'pdays', 'previous'], dtype=object)
```

Figure 31 – Subset of features selected by RFE

For the **decision tree selection**, the metric used to order the features in terms of variable importance to the model was the **average between Gini index and entropy**, only selecting whose feature importance is higher than 0.02. Therefore, the variables selected were the ones represented in the bar chart of Figure 32.

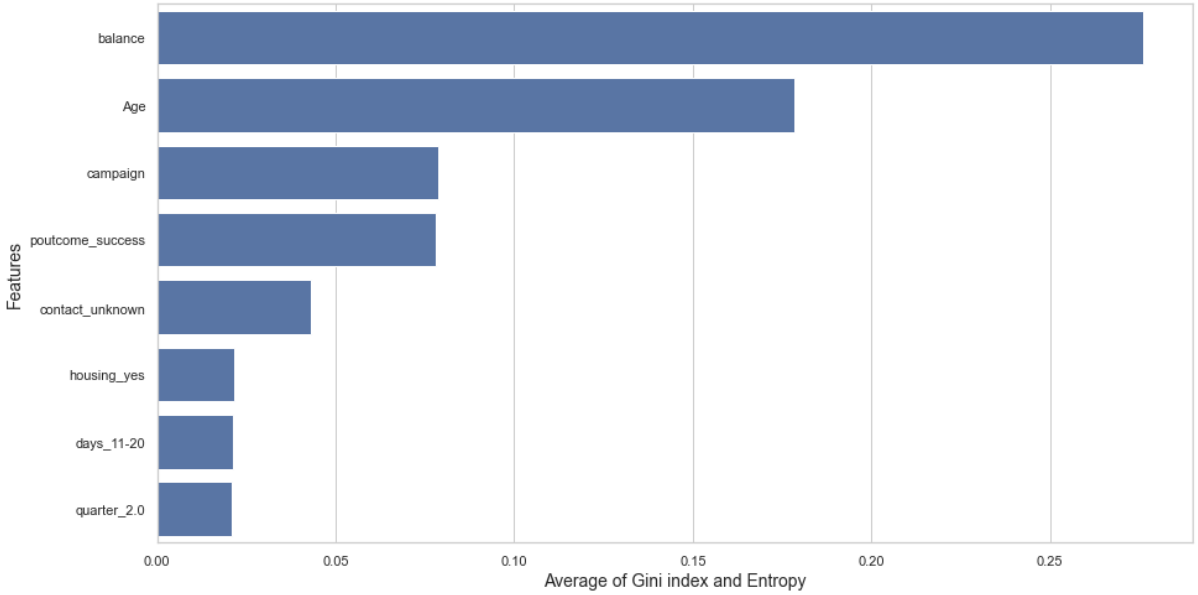


Figure 32 – Barplot with the features which have a Gini index and entropy average higher than 0.02

Relatively to the correlation between input features and output variable, **poutcome_success**, **contact_unknown**, **poutocome_unknown**, **pdays**, **contact_cellular**, **housing_no** and **housing_yes** were identified to be the variables to be more correlated with the target, by analyzing the Phik correlation matrix represented in Figure 33.

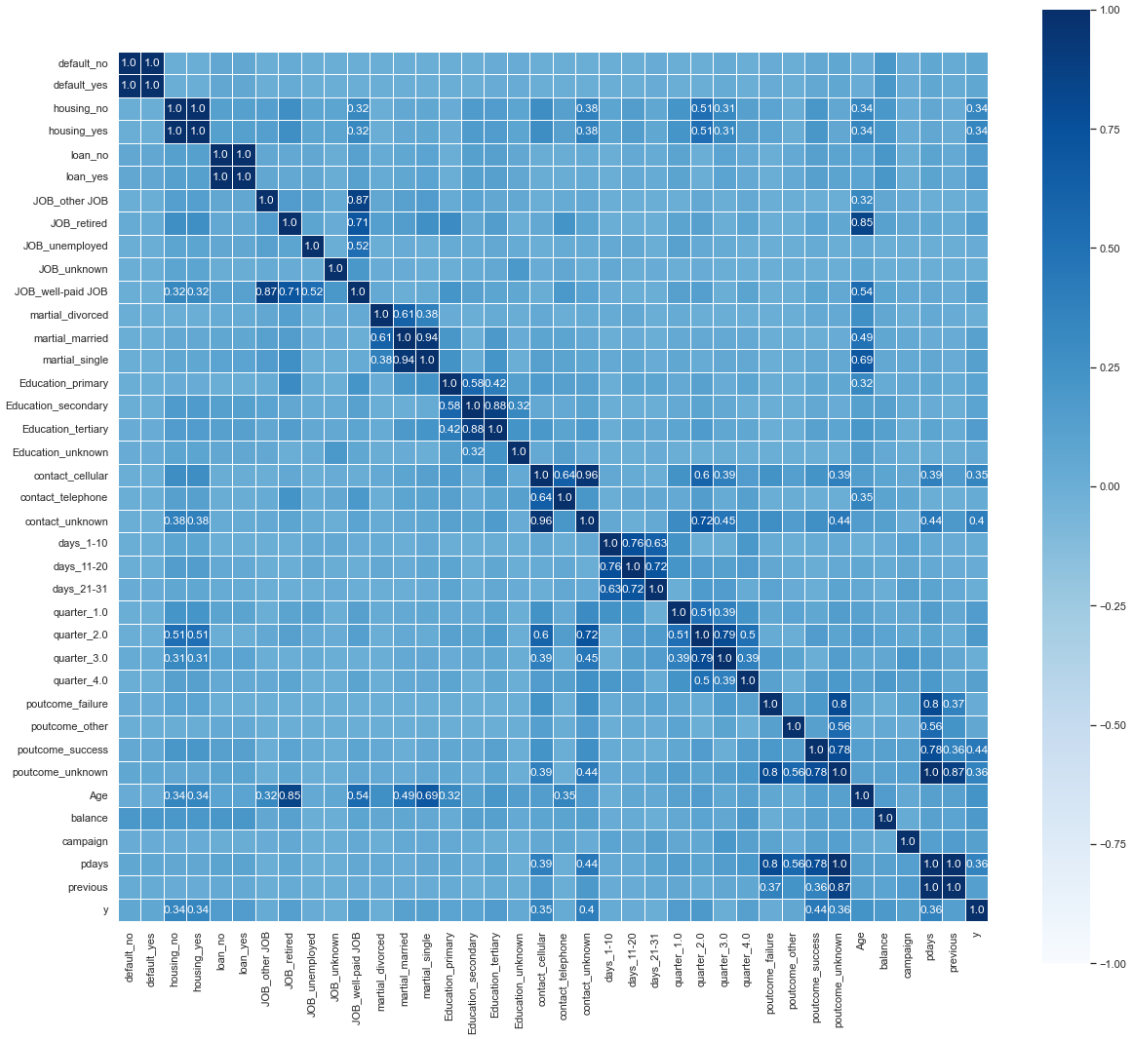


Figure 33 – Phik correlation matrix

Those last three methods were combined by a majority voting criterion, in order to further evaluate the correlation between those input features and drop any variable which might be redundant. According to this, **housing_no**, **pdays** and **quarter_2** were dropped due to their correlation between other input features being higher than 0.5, meaning the final subset of variables for training logistic regression and neural network models is composed by the features listed in Table 5:

Variable	Type	Description
Age	Interval	Transformed Age variable
balance	Interval	Transformed balance variable
campaign	Interval	Transformed campaign variable
poutcome_success	Binary	1 if poutcome is success and 0 otherwise
contact_unknown	Binary	1 if contact is unknown and 0 otherwise
housing_yes	Binary	1 if housing is yes and 0 if no
days_11-20	Binary	1 if days_11-20 is 11-20 and 0 otherwise

Table 5 – Features subset to train logistic regression and neural network in Python

In the other hand, the features selected to fit tree-based models remain the same as the original ones, although they were transformed.

6.4. MODEL SELECTION AND EVALUATION

Following the data preparation steps, predictive models should be trained on the transformed data, having their performance assessed against a validation set, which emulates the real-world data whose predictions will be made by the models when put into production. Therefore, five different types of supervised ML models were fitted, namely **logistic regression**, **neural network**, **decision tree**, **random forest** and **gradient boosting** for both SAS Model Studio and Jupyter Notebook/Python. Thus, the steps to guarantee that one selects the best predictive model are detailed in this chapter.

6.4.1. SAS Model Studio/SAS Visual Analytics

Although the **Model Composer** node is available in some SAS Model Studio versions for autotuning the hyperparameters of the models, they were not autotuned for this supervised ML project, since this functionality was not available for the SAS Model Studio version which was used. Nonetheless, the hyperparameters for these models were set according to some fine-tuning done following a trial-and-error process supported by domain knowledge. The specifications for each model are shown in the properties for each model in Figures 63 to 73 in Appendix 4.

After the predictive models are fine-tuned, predictions on the training and validation datasets were made. Therefore, several diagrams, tables and charts are produced in each model node, being several of them common to all these nodes, namely a **fit statistics** table, a **lift** chart, a **ROC** chart, an **accuracy** chart and a **F1 score** chart, as well as a **confusion matrix**, called **event classification plot** in SAS Model Studio. However, there are some of those visualizations shown in Figures 74 to 83 in Appendix 4, which are model-specific, including:

- logistic regression – charts with parameter estimates and their corresponding t values.
- neural network – neural network diagram and iteration plot.
- decision tree – tree diagram, treemap, pruning error plot and variable importance tables.
- random forest – error plot and variable importance table.
- gradient boosting – error plot and variable importance table.

Despite displaying assessment metrics for each model individually, it is paramount that those metrics are also easily compared between models in a single node. The **Model Comparison** node does so, as it is mainly used to compare the performance of supervised models in a pipeline, displaying a table with **fit statistics**, charts for **accuracy**, **lift** and **ROC** and of each model in the pipeline, each of them represented in Figures 84, 85 and 86 in Appendix 4, as well as **F1 score** charts, being this node automatically added to the pipeline once it has at least a predictive model node. Nevertheless, it is common practice to choose one of those assessment metrics for comparing the performance between models. In this case, F1 score is the metric which suits the most to this kind of problem, since predicting if a customer will subscribe to a term deposit is only performed after they are contacted, being important to account for predicting accurately both the positive cases and the negative ones.

Therefore, the F1 score charts for the training set (Figure 34) and validation set (Figure 35) should be analyzed, together with the two tables generated by the **Pipeline Comparison** tab, represented in Figure 36 and Figure 37. These two tables help to compare the models' performance for both training

and validation sets for a specified metric, in this case, F1 score. Therefore, one can conclude the following, considering the 0.5 cutoff:

- The **random forest** model registers the highest F1 score (0.733) for the training dataset. The model with the lowest F1 score for the training set is **neural network** (0.673).
- The **logistic regression** model records the highest F1 score for the validation set, with 0.687. As such, SAS Model Studio set this model to be the **champion model** automatically, due to its F1 score being the greatest for the validation set among the models in the pipeline. On the other hand, the model with the lowest F1 score is the decision tree model (0.670).
- Any model showed any significant signs of overfitting or underfitting.

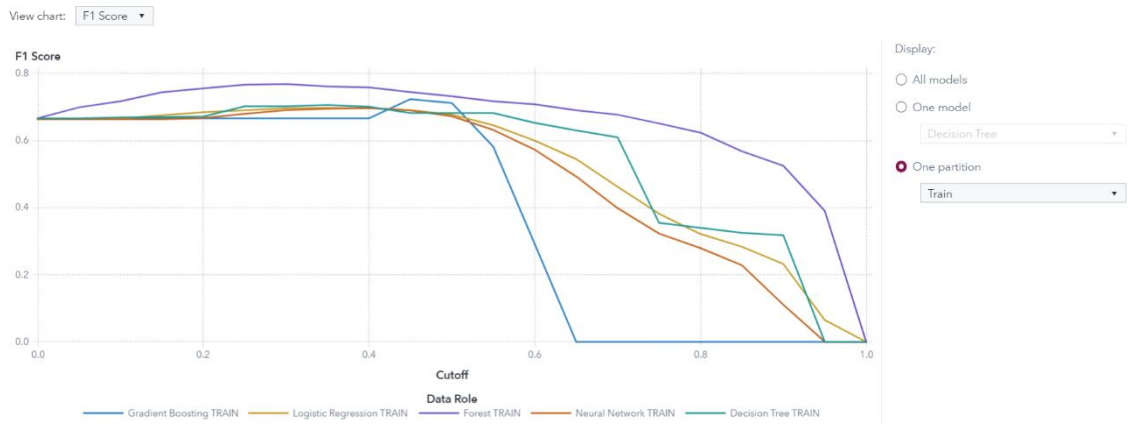


Figure 34 – Line chart with the F1 score for the training set and for different cutoffs

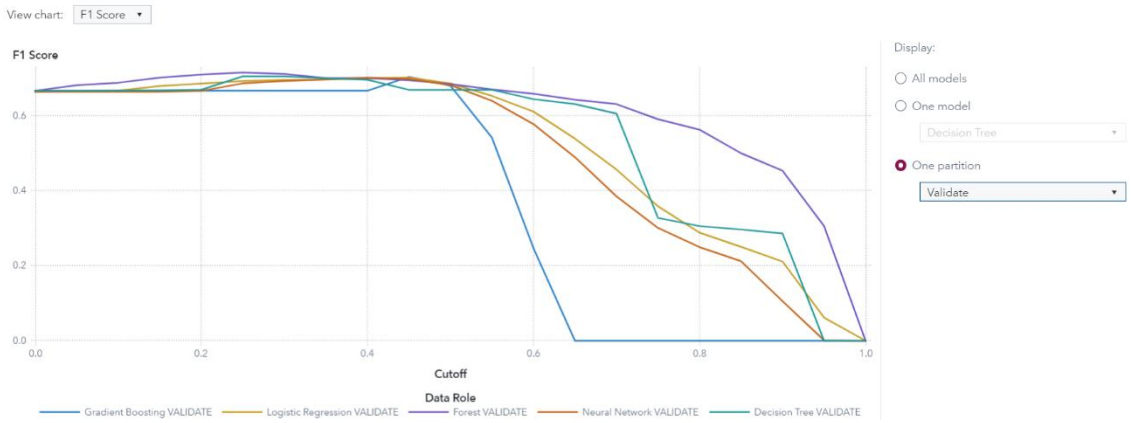
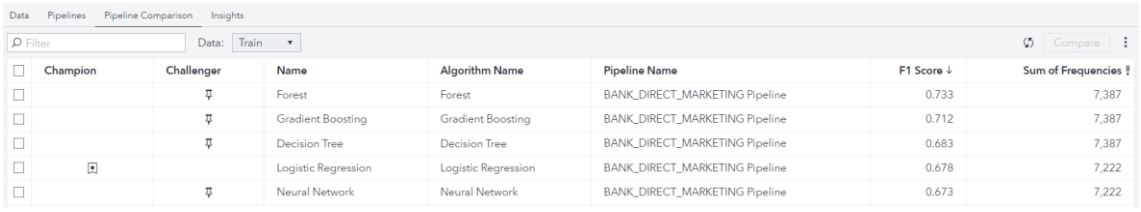
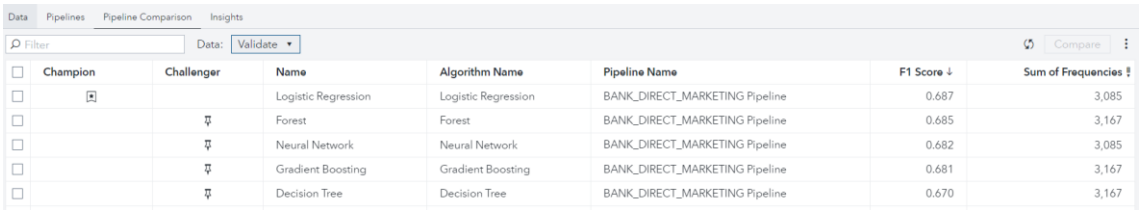


Figure 35 – Line chart with the F1 score for the validation set and for different cutoffs



Champion	Challenger	Name	Algorithm Name	Pipeline Name	F1 Score ↓	Sum of Frequencies
		Forest	Forest	BANK_DIRECT_MARKETING Pipeline	0.733	7,387
		Gradient Boosting	Gradient Boosting	BANK_DIRECT_MARKETING Pipeline	0.712	7,387
		Decision Tree	Decision Tree	BANK_DIRECT_MARKETING Pipeline	0.683	7,387
		Logistic Regression	Logistic Regression	BANK_DIRECT_MARKETING Pipeline	0.678	7,222
		Neural Network	Neural Network	BANK_DIRECT_MARKETING Pipeline	0.673	7,222

Figure 36 – Table with F1 score models' values for the training set



Champion	Challenger	Name	Algorithm Name	Pipeline Name	F1 Score ↓	Sum of Frequencies
		Logistic Regression	Logistic Regression	BANK_DIRECT_MARKETING Pipeline	0.687	3,085
		Forest	Forest	BANK_DIRECT_MARKETING Pipeline	0.685	3,167
		Neural Network	Neural Network	BANK_DIRECT_MARKETING Pipeline	0.682	3,085
		Gradient Boosting	Gradient Boosting	BANK_DIRECT_MARKETING Pipeline	0.681	3,167
		Decision Tree	Decision Tree	BANK_DIRECT_MARKETING Pipeline	0.670	3,167

Figure 37 – Table with F1 score models' values for the validation set

Although SAS Model Studio defined the logistic model as the champion model, the user can set another model from the set of trained models to be the champion, due to several reasons, besides the performance of the models (e.g. model interpretability), however the logistic regression model was kept as the champion due to being the model with the highest F1 score for the validation set and the lack of overfitting.

After the champion model is selected, the user might want to check **Insights** tab, as it generates a report containing various charts, as shown in Figure 38. These visualizations may help the end user to identify which were the most common variables used in the models, which variables had the most relative importance for the champion model, as well the corresponding values for the metric chosen to assess their performance and the cumulative lift for the champion model. A project summary is also produced automatically by the software, although the user is also allowed to add their own notes to enrich the report and convey more information to the viewer.

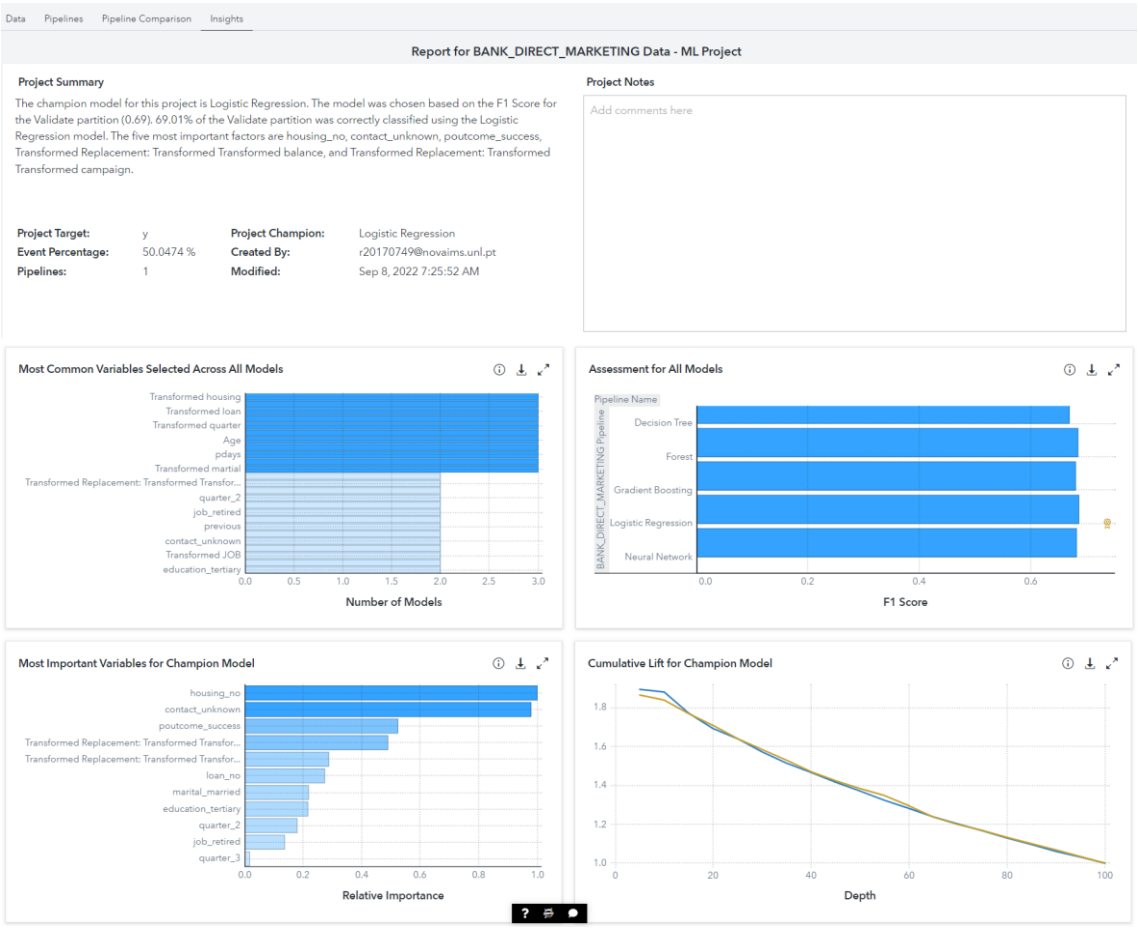


Figure 38 – Insights tab report

6.4.2. Jupyter Notebook/Python

Model selection and evaluation stages were developed in Jupyter Notebook/Python with some notable differences compared to SAS Model Studio. Accordingly, the hyperparameters' tuning of the predictive models was made in an automated fashion, using a class named **GridSearchCV**, which fits the data into the predictive model and generates predictions from it. The cross-validation method used was the stratified 5-fold method, resorting to the **StratifiedKFold** class. Therefore, models with different hyperparameters are trained on different subsets of data, being their performance evaluated through the calculation of a pre-defined assessment metric on test data, which is F1 score in this case. At the end of the search, the model with the best performance among all candidates is given, along with its hyperparameters, as shown in an example for the best logistic regression model (Figure 87 in Appendix 5.

The selected model is then refitted to the training data and predictions are made, in order to compute a table with several assessment metrics, namely accuracy, precision, recall and F1 score. This table, produced by the **classification_report** class, is similar to the fit statistics table generated by SAS Model Studio in each model's node, although only the most important ones are represented in it. An example of a classification_report table for the chosen logistic regression model is represented in Figures 88 and 89 in Appendix 5 for both training and validation sets, which can be copied for all other models.

Additionally, a confusion matrix and a lift chart can also be created after the predictions are made. The confusion matrix can be produced through the **ConfusionMatrixDisplay** class from **sklearn** (an

example for both training and validation sets are represented in Figures 90 and 91 in Appendix 5), being these matrices somewhat different from the ones produced in SAS Model Studio, while the lift chart could be generated resorting to **plot_lift_curve** class from **scikitplot** library, whose example is shown in Figure 92 in Appendix 5. Additionally, a **ROC** chart is also possible to be built, represented in Figure 93 in Appendix 5, to compare this metric across the set of trained models.

Similarly to SAS Model Studio, it is also possible to create charts in Jupyter Notebook/Python to compare the assessment metrics values between training and validation sets. Hence, two barplots for both F1 score (Figure 39) and accuracy (in figure 94 in Appendix 5) were produced for comparing performance between models.

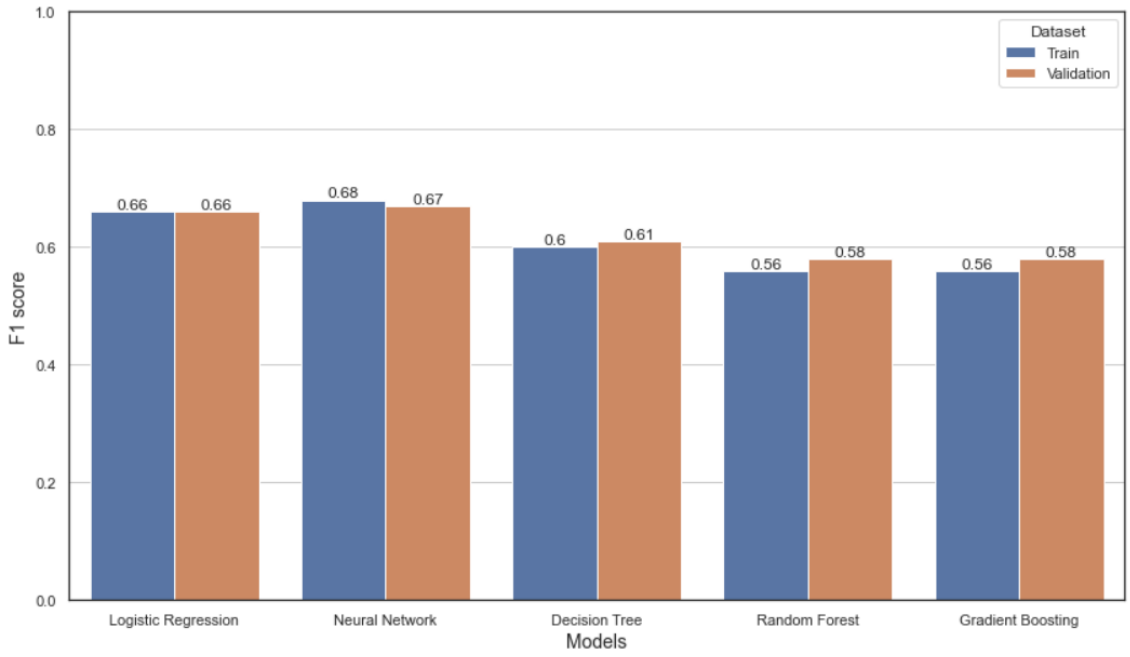


Figure 39 – Bar chart comparing F1 score between models

If one analyzes the barplot represented in Figure 39, one can conclude that the **neural network** model has the best performance among all models, considering the F1 score for training and validation set, although the **logistic regression** model also has a good performance. Therefore, one of these two models should be selected to be implemented in a production environment, which is out of the scope of this report. Additionally, these two models do not suffer from overfitting or underfitting, which is also verified for the remaining models. On the contrary, **tree-based** models scored very low for both training and validation datasets compared to the two other models, contrarily to their scores in SAS Model Studio, where they recorded the highest values for the training set and similar F1 score values to the other two models for the validation set.

7. RESULTS AND DISCUSSION

After developing a supervised ML pipeline in SAS Model Studio and Jupyter Notebook/Python while highlighting their capabilities, it is possible to compare both technologies and take note of the advantages and disadvantages of each other.

For instance, SAS Model Studio has a great advantage over Jupyter Notebook/Python, which is enabling less-experienced people in the field of supervised ML to build a pipeline, since they can pick one up from a set of templates made available by SAS Model Studio or even allow the software to automatically generate a pipeline. In Jupyter Notebook/Python, it is not possible to generate an end-to-end supervised ML project given a data source on its own, so the programmer has to always build it from scratch, which is derived from its hard code nature. This difference is also noticed in terms of the visualizations produced in SAS Model Studio, which are already embedded in the nodes themselves, while in Jupyter Notebook/Python one must often build charts from scratch, although there are some already developed through libraries classes, such as **confusion matrices**, **classification reports** and **lift charts**, being only needed to provide some parameters. Hence, SAS Model Studio allows a data scientist to introduce more domain knowledge into the project, rather than wasting time on working on code for creating visualizations, which could be better used to identify flaws in the supervised ML project. Nevertheless, this feature can be useful, due to one being capable to customize the visualizations according to the user's preferences in Jupyter Notebook/Python, although it can also be done to some extent in SAS Visual Analytics or even through SAS code, if one is not satisfied with the graphs generated by SAS Model Studio.

Moreover, the low code and user-friendly GUI of SAS Model Studio enables to easily view how a supervised ML project is structured and which tasks are executed, whereas in Jupyter Notebook/Python it can become more difficult to understand the workflow as a whole.

Relatively to the supervised ML project itself, SAS Model Studio has a great feature which Jupyter Notebook/Python does not have, the **Best** transformation, available in the **Transformations** node. This node has many methods available for changing the data distribution, correcting non-normality and stabilizing variances, thus with the help of Best transformation, this software is able to select the transformation which achieves the best results according to some given criteria. In Jupyter Notebook/Python, this can also be done, however, it can take more time to find which transformation achieves the best result, because the iterative process needs to be developed by the programmer, contrarily to SAS Model Studio where it is already implemented.

Nonetheless, as an open-source programming language, Jupyter Notebook/Python has many libraries resources available, thus several transformations developed in Jupyter Notebook/Python are not available in SAS Model Studio, such as some of the techniques detailed in previous chapters, namely Robust Scaler, besides some visualizations, such as Phik correlation matrix, as well as RFE, an example of a feature selection method. Consequently, this could hamper a supervised ML project, reducing the performance of the predictive models. However, for this specific project, it was not verified, since the ML models achieved similar results when it comes down to models' performance, although different models were considered to have better performance in each tool and the specifications found were different as well. In SAS Model Studio, the model found by the internee to have the best performance, resorting to trial-and-error, was the logistic regression model, with a F1 score of 0.678 for training set and 0.687 for the validation set, since there is neither overfitting nor underfitting in this model.

Although this model was considered to hold the best performance among the set of predictive models trained, SAS Model Studio set automatically the logistic regression as the champion model, since it recorded the highest F1 score for the validation set. On the other hand, the neural network model was deemed the model with better performance, with 0.68 F1 score for the training set and 0.67 for the validation set. Therefore, neither SAS Model Studio nor Jupyter Notebook/Python can be said to achieve better models' performance generally.

Bearing all those assumptions in mind, SAS Viya solutions and Jupyter Notebook/Python can be said to complement each other, as these two environments can be used together when developing a supervised ML pipeline, being responsible for different tasks within it. For instance, by taking advantage of the various visualizations made available and easiness to create them, one can choose SAS Model Studio and SAS VA to produce the visualizations, mainly resorting to the Data Exploration node in SAS Model Studio and SAS VA when Data Exploration node charts do not fulfill the user's requirements. On the other hand, Jupyter Notebook/Python could serve as the platform to implement the data transformation tasks, as well as the model selection and evaluation stages, due to its wide range of ML techniques and predictive models available, as well as the hyperparameters' autotuning and various assessment metrics used to evaluate the model's performance.

8. CONCLUSION

By having access to the different courses which the internee had access to during the internship, it was possible to attain the main goal of this internship report, which was to exemplify the development of a supervised ML pipeline in SAS Viya. Therefore, the capabilities offered by SAS Model Studio and SAS VA were successfully showcased, namely the kind of visualizations and analytical techniques for exploring and transforming the data, as well as building predictive models. The secondary goal of this report was achieved as well, aimed at building another pipeline in Jupyter Notebook, coding in Python, in order to compare it with the one produced in SAS Model Studio, applying ML techniques to the same data.

Therefore, the attainment of both goals permitted to reach some conclusions about the advantages and disadvantages of each tool, whether they are related with the amount of ML methods and visualizations available, as well as the quickness to build and perform each analytical task. Hence, listing the pros and cons led to conclude that SAS Model Studio and SAS VA are able to allow the user to perform data exploration in the context of a supervised ML project, a very important stage for its success, since the takeaways retained from this step influence all decisions made in further ones, affecting the predictive models' accuracy. This is possible due to these SAS Viya solutions providing a simple and easy-to-use GUI, yet extensive enough to allow the user to build different visualizations for exploring the data. For instance, it was shown that Data Exploration node in SAS Model Studio is able to produce a lot of useful visualizations in a short period of time, even though one might resort to SAS VA, where more visualizations are available.

On the other hand, Python executed in a Jupyter Notebook environment can be used for the remaining stages of a supervised ML pipeline, stretching from data preparation to model selection and evaluation stages. The internee came to this conclusion since one can access and make use of several libraries which have many pre-built functions, helping to solve many of the data issues (missing values, outliers, feature engineering, among others) one might find within the data. The extensive library of predictive models and hyperparameter autotuning are also valuable resources.

Furthermore, the internee expects that the work presented in this internship report can be helpful to shed some light over SAS Viya applications and their capabilities in the ML field, although many companies already incorporate SAS technologies in their operations.

However, the work presented in this report could not be achieved out without the knowledge passed on to the internee by the Consulting area colleagues, integrated in the Professional Services team, which helped to understand the SAS Viya capabilities and the solutions which could suit the most to the fulfilment of the work showcased in this report.

9. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

There were some limitations which the internee faced during the development of this report, including the version of SAS Viya which the internee had access to. For instance, this SAS Viya version did not include some of the features which were detailed in the SAS Model Studio chapter, namely the Model Composer node, which permits to autotune the models' hyperparameters, while this was done in Jupyter Notebook/Python. Thus, with the right SAS Viya version, it would be interesting to compare the rapidness and which hyperparameters values both tools would find to produce the most accurate predictive models according to some criteria, in order to assess which tool could generate the best ML models in terms of performance.

In future works, one could go further in exploring and demonstrating how SAS Model Studio could be connected with another tool within the SAS Viya environment for deploying SAS Model Studio models into production environments, called **SAS Model Manager**. This tool also allows to evaluate the deployed models' performance across time, using various assessment metrics, including the ones presented throughout this report, while enabling to retrain the models with unseen data when performance decays below a pre-defined threshold. Unfortunately, this was the main reason why SAS Model Manager features were not presented in this report, since the SAS Viya version used did not allow to do so. Once again, it would be worthy to compare SAS Viya solutions performance and capabilities compared with the ones made available by Jupyter Notebook/Python in the ML models deployment field and assessing which connections each tool has with production environments.

On the same note, one could try to import Python models into SAS Model Manager using an API specifically developed for that, to take advantage of the various production environments which can be used to deploy models, given the fact that the SAS Viya version used allows to do so.

Relatively to the dataset used, one could also pick a dataset containing more observations and features to assess how SAS Viya servers could handle data of a bigger size and if it would affect the rapidness of the generation of outputs, to test and demonstrate the computation power of CAS.

10.REFERENCES

Abdallah, Z. S., Du, L., & Webb, G. I. (2017). Data Preparation. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of Machine Learning and Data Mining* (pp. 318–327). Springer US. https://doi.org/10.1007/978-1-4899-7687-1_62

Akbari, A., Ng, L., & Solnik, B. (2021). Drivers of economic and financial integration: A machine learning approach. *Journal of Empirical Finance*, 61, 82–102. https://doi.org/10.1016/j.jempfin.2020.12.005

Baak, M., Koopman, R., Snoek, H., & Klous, S. (2020). A new correlation coefficient between categorical, ordinal and interval variables with Pearson characteristics. *Computational Statistics & Data Analysis*, *152*, 107043. https://doi.org/10.1016/j.csda.2020.107043

Baer, T. (2020). SAS® Viya® and the cloud: How SAS is changing the game it invented: A dbInsight white paper for SAS. *SAS*. Retrieved October 11, 2022, from https://www.sas.com/content/dam/SAS/documents/analyst-reports-papers/en/dbinsight-sas-viya-goes-cloud-native-111484.pdf

Bellman, R. (1978). *An introduction to artificial intelligence: Can computers think?* Boyd & Fraser Pub. Co.

Bellman, R. (2021). *Dynamic Programming*. Princeton University Press. https://doi.org/10.2307/j.ctv1nxcw0f

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Clarendon Press ; Oxford University Press.

Bishop, C. M. (2006). *Pattern recognition and Machine Learning*. Springer.

Bourn, E. (2018). *SAS® Viya® Service Layer Architecture Overview*. Retrieved September 22, 2022, from https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2272-2018.pdf

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

Brownlee, J. (2019). *How to Choose a Feature Selection Method For Machine Learning*. Machinelearningmastery. Retrieved September 20, 2022, from https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

Cai, J., Luo, J., Wang, S., & Yang, S. (2018). Feature selection in Machine Learning: A new perspective. *Neurocomputing*, *300*, 70–79. https://doi.org/10.1016/j.neucom.2017.11.077

Câmara, D. (2015). *Bio-Inspired Networking*. ISTE Press Ltd. ; Elsevier Ltd.

Canova, S., Cortinovis, D. L., & Ambrogi, F. (2017). How to describe univariate data. *Journal of Thoracic Disease*, *9*(6), 1741–1743. https://doi.org/10.21037/jtd.2017.05.80

Carey, M. (2018). *Exploring interactive reports with SAS Visual Analytics*. SAS Users. Retrieved September 22, 2022, from https://blogs.sas.com/content/sgf/2018/05/31/exploring-interactive-reports-with-sas-visual-analytics/

Cohen. (2013). *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences* (3rd ed.). Routledge. https://doi.org/10.4324/9780203774441

Crossing, V. (2020). *How to load weather data into SAS Visual Analytics or SAS Visual Statistics. – Visual Crossing Weather*. Retrieved September 20, 2022, from https://www.visualcrossing.com/resources/documentation/weather-data-tutorials/how-to-load-weather-data-into-sas-viya/

Dalianis, H. (2018). *Clinical Text Mining*. Springer International Publishing. https://doi.org/10.1007/978-3-319-78503-5

Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap Methods and their Application* (1st ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511802843

Deshpande, M. (2022). *Perceptrons: The First Neural Networks*. GameDev Academy. Retrieved September 20, 2022, from https://gamedevacademy.org/perceptrons-the-first-neural-networks/

Donner, A. (1982). The Relative Effectiveness of Procedures Commonly Used in Multiple Regression Analysis for Dealing with Missing Values. *The American Statistician*, *36*(4), 378. https://doi.org/10.2307/2683092

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*, *46*(1/3), 389–422. https://doi.org/10.1023/A:1012487302797

Jagannath, V. (2017). *File:Random forest diagram complete.png - Wikimedia Commons*. Retrieved September 20, 2022, from https://commons.wikimedia.org/w/index.php?curid=68995764

Jaiswal, A. S. (2022). Importance of Data Exploration in Data Analysis A Review Paper *International Journal of Advanced Research in Science, Communication and Technology*, *2*(2), 362–366. https://doi.org/10.48175/568 362

James, G., Witten, D., Hastie, T., & Tibshirani, R. (Eds.). (2013). *An introduction to statistical learning: With applications in R*. Springer.

Jupyter. (n.d.). Retrieved September 22, 2022, from https://test-jupyter.readthedocs.io/en/latest/architecture/how_jupyter_ipython_work.html

Kang, H. (2013). The prevention and handling of the missing data. *Korean Journal of Anesthesiology*, *64*(5), 402. https://doi.org/10.4097/kjae.2013.64.5.402

Karwatka, P. (2021). *Monolithic architecture vs microservices: Which is better? | Divante*. Retrieved September 22, 2022, from https://www.divante.com/blog/monolithic-architecture-vs-microservices

Kelleher, J. D., Mac Namee, B., & D'Arcy, A. (2015). *Fundamentals of Machine Learning for predictive data analytics: Algorithms, worked examples, and case studies*. The MIT Press.

Khurana, U., Samulowitz, H., & Turaga, D. (2017). *Feature Engineering for Predictive Modeling using Reinforcement Learning*. https://doi.org/10.48550/ARXIV.1709.07150

Kotsiantis, S. B., Zaharakis, I. D., & Pintelas, P. E. (2006). Machine Learning: A review of classification and combining techniques. *Artificial Intelligence Review*, *26*(3), 159–190. https://doi.org/10.1007/s10462-007-9052-3

Kuhn, M., & Johnson, K. (2013). *Applied predictive modeling*. Springer.

Kumar, A. (2022). *Logit vs Probit Models: Differences, Examples*. Data Analytics. Retrieved September 20, 2022, from https://vitalflux.com/logit-vs-probit-models-differences-examples/

Lane, D. M. (2013) *Introduction to Statistics: An Interactive eBook*.

Larasati, A., Dwiastutik, A., Ramadhanti, D., & Mahardika, A. (2018). The effect of Kurtosis on the accuracy of artificial neural network predictive model. *MATEC Web of Conferences*, *204*, 02018. https://doi.org/10.1051/matecconf/201820402018

Lippmann, R. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, *4*(2), 4–22. https://doi.org/10.1109/MASSP.1987.1165576

Liu, H., & Cocea, M. (2018). Induction of classification rules by Gini-index based rule generation. *Information Sciences*, *436–437*, 227–246. https://doi.org/10.1016/j.ins.2018.01.025

Magdum, R. (2022). What is Data Exploration? And its Importance in Data Analytics. *International Journal of Advanced Research in Science, Communication and Technology*, *9*(1), 1482–1485.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Mohamed, H., Negm, A., Zahran, M., & Saavedra, O. C. (2015). ASSESSMENT OF ARTIFICIAL NEURAL NETWORK FOR BATHYMETRY ESTIMATION USING HIGH RESOLUTION SATELLITE IMAGERY IN SHALLOW LAKES: CASE STUDY EL BURULLUS LAKE. *International Water Technology Journal*, 5(4).

Morris, A. S., & Langari, R. (2012). *Measurement and instrumentation: Theory and application*. Academic Press.

Narkhede, S. (2021). *Understanding Confusion Matrix - Towards Data Science*. Medium. Retrieved September 20, 2022, from https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Park, H.-A. (2013). An Introduction to Logistic Regression: From Basic Concepts to Interpretation with Particular Attention to Nursing Domain. Journal of Korean Academy of Nursing, 43(2), 154. https://doi.org/10.4040/jkan.2013.43.2.154

Park, Y.-S., & Lek, S. (2016). Artificial Neural Networks. In *Developments in Environmental Modelling* (Vol. 28, pp. 123–140). Elsevier. https://doi.org/10.1016/B978-0-444-63623-2.00007-4

Peng, C.-Y. J., Lee, K. L., & Ingersoll, G. M. (2002). An Introduction to Logistic Regression Analysis and Reporting. *The Journal of Educational Research*, *96*(1), 3–14. https://doi.org/10.1080/00220670209598786

Perlich, C., & Provost, F. (2006). Distribution-based aggregation for relational learning with identifier attributes. *Machine Learning*, *62*(1–2), 65–105. https://doi.org/10.1007/s10994-006-6064-1

Polamuri, S. (2020). *Stratified K-Fold Cross Validation*. Dataaspirant. Retrieved September 20, 2022, from https://dataaspirant.com/8-stratified-k-fold-cross-validation/

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks* (1st ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511812651

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

Sadovy, L. (2017). *SAS Viya: What's in it for me, the business?* Blogs. Retrieved September 22, 2022, from https://blogs.sas.com/content/sascom/2017/09/11/sas-viya-whats-business/

SAS Help Center. (n.d.). Retrieved September 22, 2022, from https://documentation.sas.com/doc/en/pgmsascdc/v_022/pgmgs/p0heseo288cnp6n1rrzzkei1unzs.htm#n0rcwrjpugcsdtn1fez1f1wqeddg

SAS Help Center. (n.d.). Retrieved September 22, 2022, from https://documentation.sas.com/doc/en/pgmsascdc/v_022/pgmgs/p1nb447f2s2eqrn1690bhu3q5l1c.htm

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, *5*(2), 197–227. https://doi.org/10.1007/BF00116037

Sirigiri, R. K. (2021). *Box-and-whisker Plot(Box Plot)*. Medium. Retrieved September 20, 2022, from https://medium.com/@ravikiransirigiri4215/box-and-whisker-plot-box-plot-9f595839ee21

Sultan, H. H., Salem, N. M., & Al-Atabany, W. (2019). Multi-Classification of Brain Tumor Images Using Deep Neural Network. *IEEE Access, 7,* 69215–69225. https://doi.org/10.1109/ACCESS.2019.2919122

Zheng, A., & Casari, A. (2018). *Feature engineering for Machine Learning: Principles and techniques for data scientists* (First edition). O'Reilly.

APPENDIX 1

Partition Used	Number of Observations
All data	10,554

Figure 40 – Number of observations summary

Variable Name	Number of Missing Values	Percentage Missing
Age	0	0
Education	0	0
JOB	0	0
balance	0	0
campaign	0	0
contact	0	0
day	0	0
default	0	0
housing	0	0
loan	0	0
martial	0	0
month	0	0
pdays	0	0
poutcome	0	0
previous	0	0
y	0	0

Figure 41 – Number of missing values' summary

Variable Name	Minimum	Maximum	Mean	Standard Devi...	Skewness	Kurtosis	Relative Varia...	Mean plus 2 SD	Mean minus 2 ...
Age	18	95	41.2268	11.9977	0.8442	0.5614	0.2910	65.2223	17.2313
balance	-3,058	81,204	1,550.7061	3,133.4772	7.7109	119.4512	2.0207	7,817.6604	-4,716.2482
campaign	1	50	2.4748	2.6166	5.0998	44.6303	1.0573	7.7081	-2.7585
pdays	-1	2	-0.3346	1.1592	1.2604	-0.2358	3.4649	1.9839	-2.6531
previous	0	275	0.8534	3.4752	48.4859	3,689.2925	4.0721	7.8039	-6.0971

Figure 42 – Statistics summary

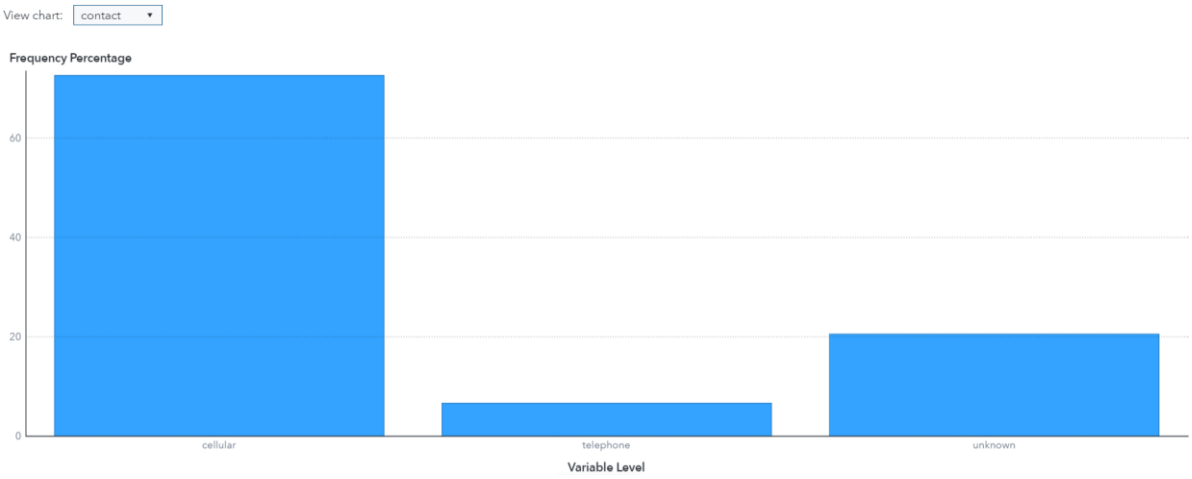


Figure 43 – Example of a barplot for a categorical variable

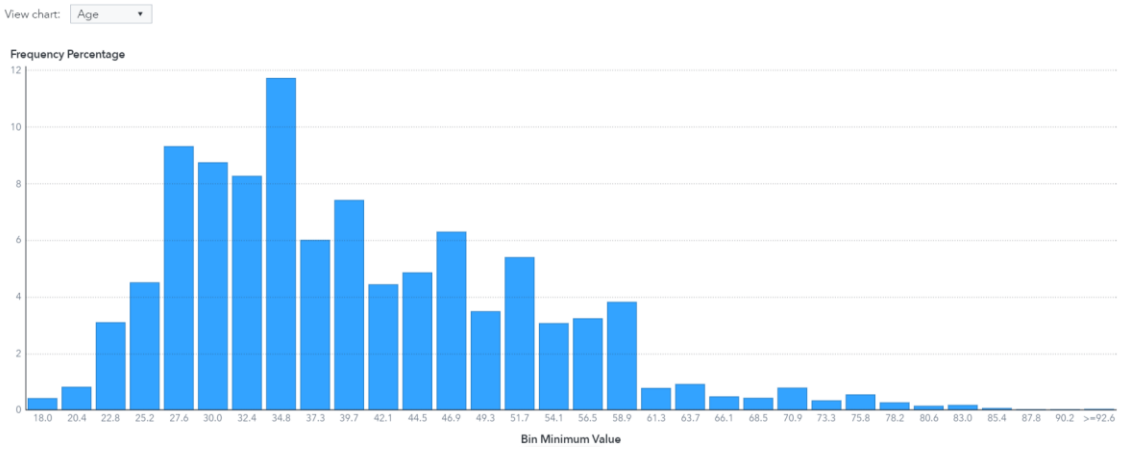


Figure 44 – Example of a barplot for a continuous variable

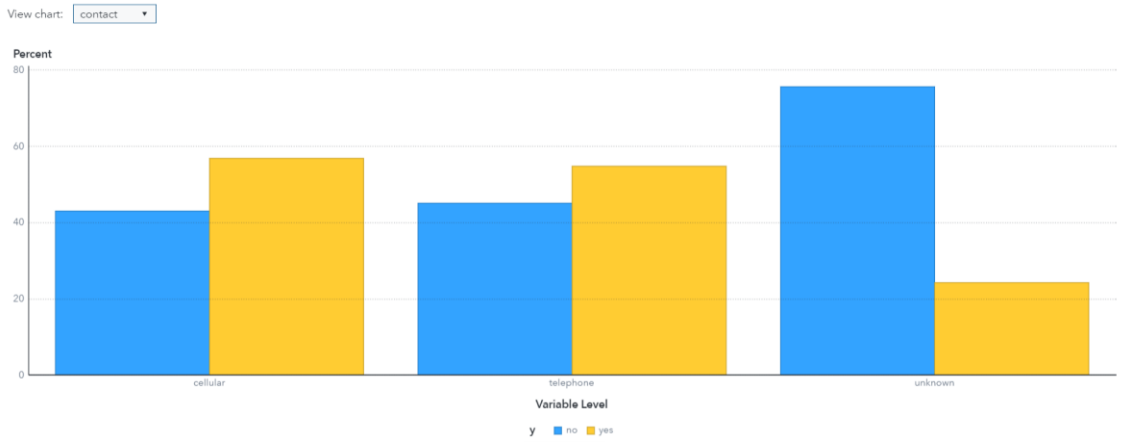


Figure 45 – Example of a target variable by categorical variable crosstabulations' plot

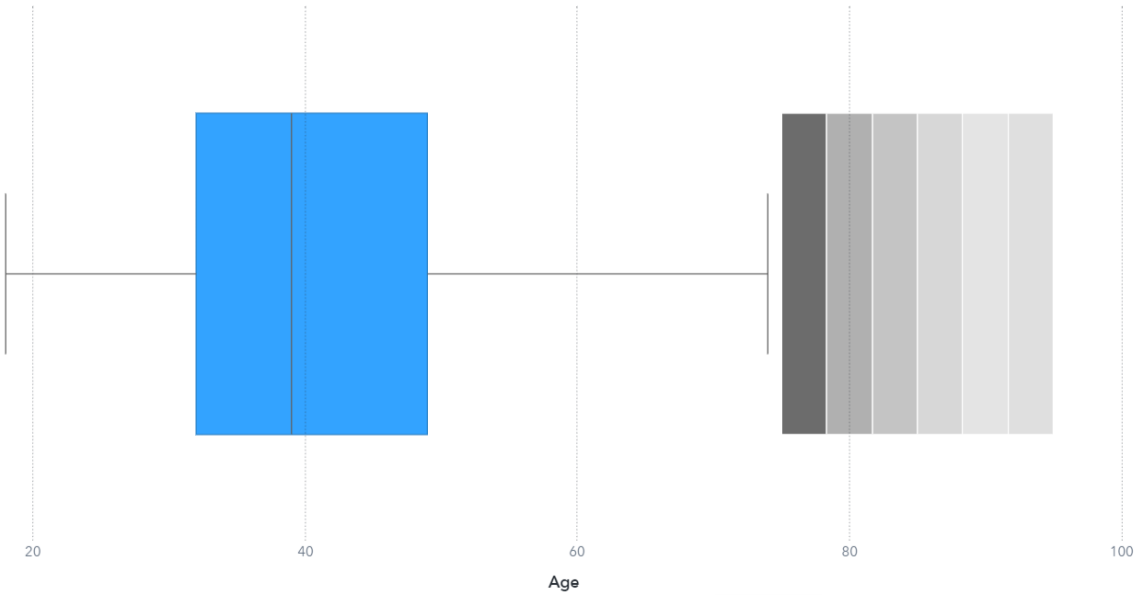


Figure 46 – Example of a boxplot

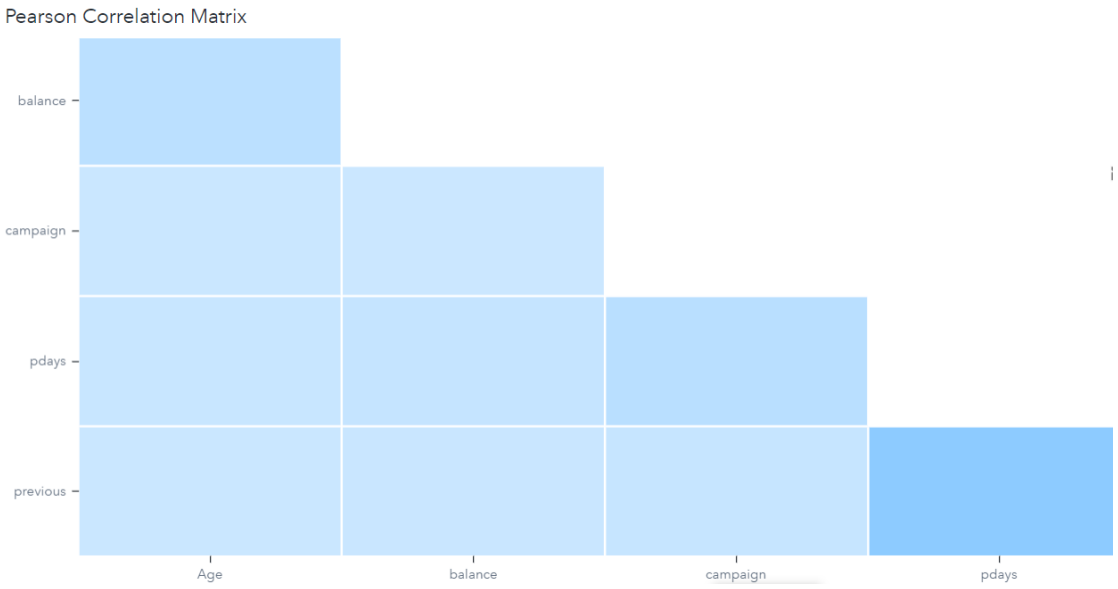


Figure 47 – Pearson correlation coefficient matrix

Eixo X	Eixo Y	Correlação
Age	previous	0.0128
Age	pdays	0.0232
Age	campaign	-0.0076
Age	balance	0.1107
balance	previous	0.0171
balance	pdays	0.0455
balance	campaign	-0.0049
campaign	previous	-0.0384
campaign	pdays	-0.1242
pdays	previous	0.4075

Figure 48 – Pearson correlation coefficient table

APPENDIX 2

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10554 entries, 122482 to 144504
Data columns (total 16 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Education  10554 non-null  object
 1   martial    10554 non-null  object
 2   JOB        10554 non-null  object
 3   Age        10554 non-null  Int64
 4   default    10554 non-null  object
 5   balance    10554 non-null  int64
 6   housing    10554 non-null  object
 7   loan       10554 non-null  object
 8   contact    10554 non-null  object
 9   day        10554 non-null  int64
 10  month      10554 non-null  object
 11  campaign   10554 non-null  int64
 12  pdays      10554 non-null  int64
 13  previous   10554 non-null  int64
 14  poutcome   10554 non-null  object
 15  y          10554 non-null  object
dtypes: Int64(1), int64(5), object(10)
memory usage: 1.4+ MB
```

Figure 49 – Number of observations and missing values summary

	Age	balance	campaign	pdays	previous
count	10554.0000	10554.0000	10554.0000	10554.0000	10554.0000
mean	41.2268	1550.7061	2.4748	-0.3346	0.8534
std	11.9977	3133.4772	2.6166	1.1592	3.4752
min	18.0000	-3058.0000	1.0000	-1.0000	0.0000
25%	32.0000	125.2500	1.0000	-1.0000	0.0000
50%	39.0000	567.0000	2.0000	-1.0000	0.0000
75%	49.0000	1766.7500	3.0000	1.0000	1.0000
max	95.0000	81204.0000	50.0000	2.0000	275.0000

Figure 50 – Statistics summary

	Kurtosis	Skewness
Age	0.561417	0.844221
balance	119.451176	7.710864
campaign	44.630286	5.099815
pdays	-0.235818	1.260353
previous	3689.292493	48.485876

Figure 51 – Kurtosis and skewness values for each continuous variable

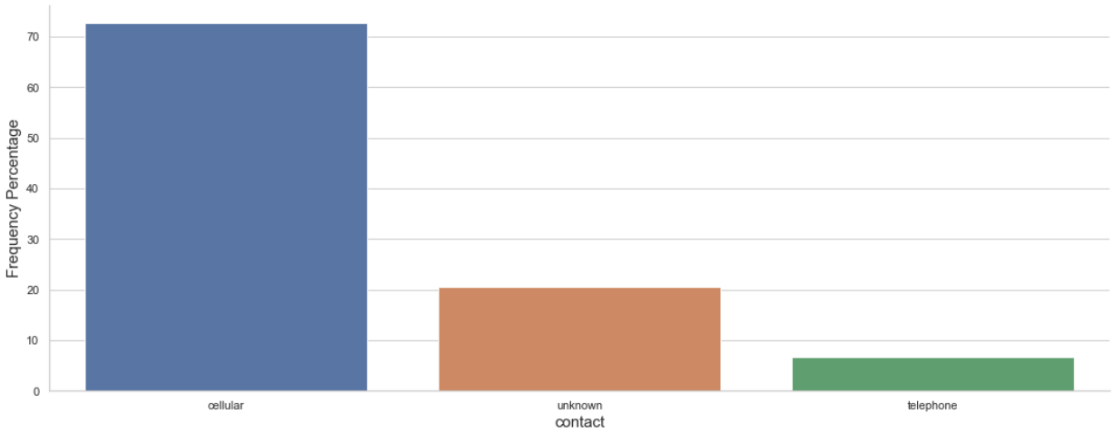


Figure 52 – Example of a barplot for a categorical variable

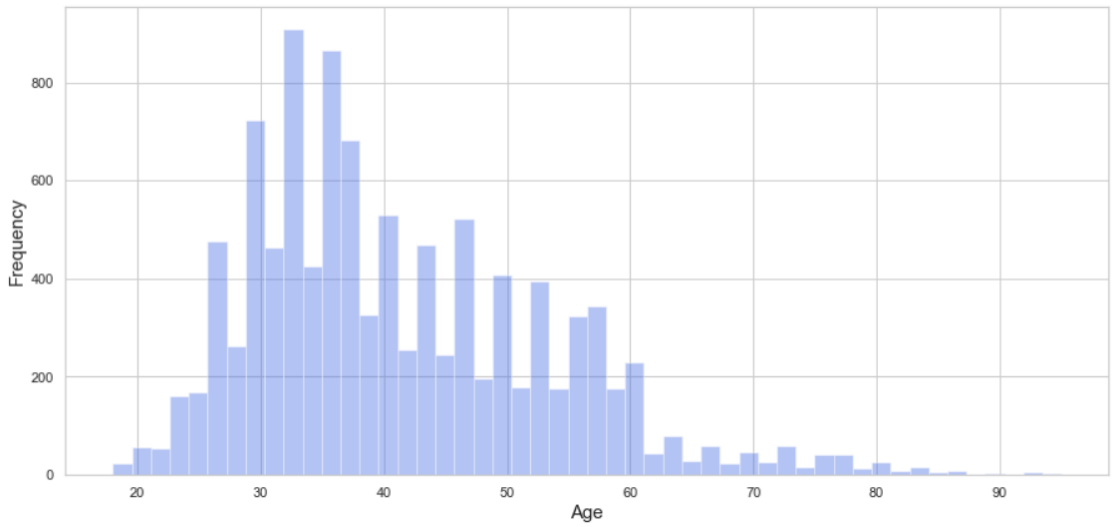


Figure 53 – Example of a barplot for a continuous variable

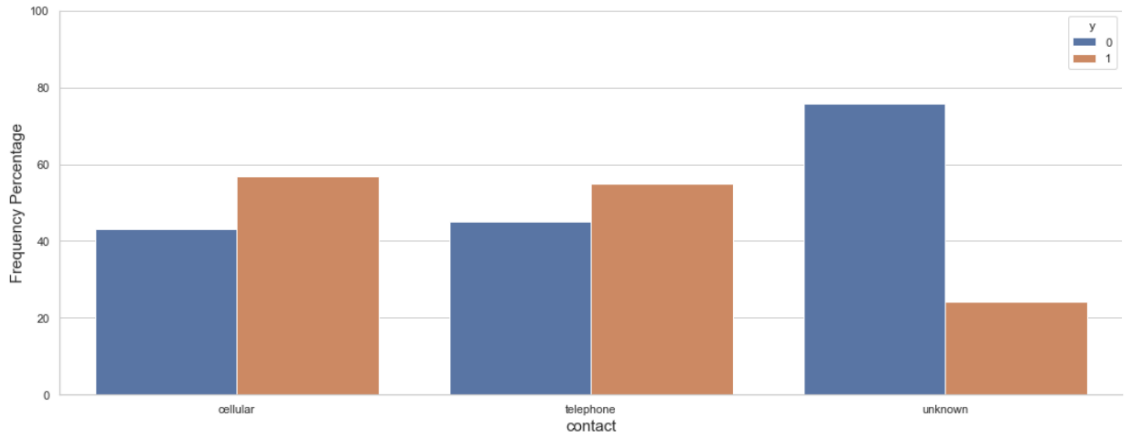


Figure 54 – Example of a target variable by categorical variable crosstabulations' plot

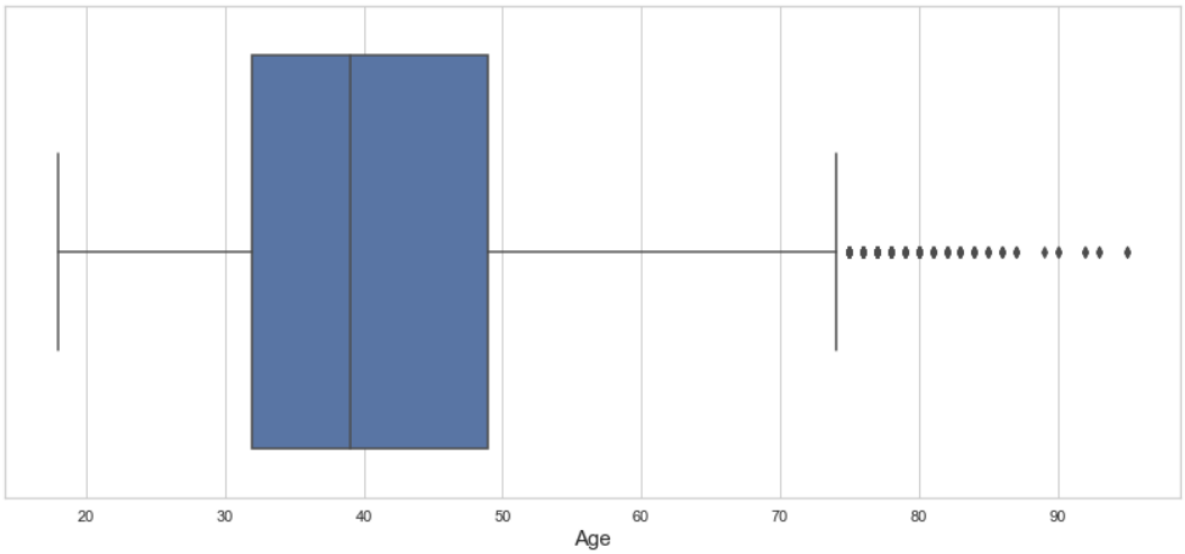


Figure 55 – Example of a boxplot

Figure 56 – Pearson correlation coefficient matrix

APPENDIX 3

Transfor...	Method	Ranking ...	Input Var...	Formula	Number ...	Minimum	Maximum	Mean	Standard...	Skewness	Kurtosis
INVSQRT_pdays	INVSQRT	Moment Skewness	pdays	1/(SQRT('pdays'n + 2))	0	0.5000	1	0.8786	0.2061	-1.1293	-0.6831
INV_campaign	INVERSE	Moment Skewness	campaign	1/('campaign'n + 1)	0	0.0196	0.5000	0.3628	0.1334	-0.3803	-1.0838
INV_previous	INVERSE	Moment Skewness	previous	1/('previous'n + 1)	0	0.0036	1	0.8236	0.3067	-1.2769	-0.1357
LG10_balance	LOG10	Moment Skewness	balance	log10('balance'n + 3059)	0	0	4.9256	3.6187	0.1746	0.5005	29.1389
SQRT_Age	SQRT	Moment Skewness	Age	SQRT('Age'n + 1)	0	4.3589	9.7980	6.4438	0.8960	0.5131	-0.1722

Figure 57 – Transformations node (1) results summary

Transfor...	Method	Ranking ...	Input Var...	Formula	Number ...	Minimum	Maximum	Mean	Standard...	Skewness	Kurtosis
INV_INVSQRT_pdays	INVERSE	Moment Kurtosis	INVSQRT_pdays	1/('INVSQRT_pdays'n + 1)	0	0.5000	0.6667	0.5397	0.0677	1.1481	-0.6152
INV_INV_campaign	INVERSE	Moment Kurtosis	INV_campaign	1/('INV_campaign'n + 1)	0	0.6667	0.9808	0.7412	0.0763	0.6530	-0.5190
SQRT_INV_previous	SQRT	Moment Kurtosis	INV_previous	SQRT('INV_previous'n + 1)	0	1.0018	1.4142	1.3450	0.1212	-1.3156	0.0136
SQR_LG10_balance	SQUARE	Moment Kurtosis	LG10_balance	('LG10_balance'n)**2	0	0	24.2619	13.1253	1.2898	1.9263	7.9898

Figure 58 – Transformations node (2) results summary

Name	Variable Label	Train	Role	Variable Level
INV_INVSQRT_PDAYS	Transformed Transformed pdays	0	INPUT	INTERVAL
INV_INV_CAMPAIGN	Transformed Transformed campaign	3	INPUT	INTERVAL
SQRT_AGE	Transformed Age	25	INPUT	INTERVAL
SQRT_INV_PREVIOUS	Transformed Transformed previous	0	INPUT	INTERVAL
SQR_LG10_BALANCE	Transformed Transformed balance	139	INPUT	INTERVAL

Figure 59 – Replacement node results summary

Transform...	Method	Input Vari...	Formula	Number o...	Minimum	Maximum	Mean	Standard ...	Skewness	Kurtosis	Variable L...
STD_REP_INV_INVSQRT_PDAYS	STANDARDIZE	REP_INV_INVSQRT_PDAYS	('REP_INV_INVSQRT_PDAYS'n-0.5396451796)/0.0676649157	0	-0.5859	1.8772	0.0000	1.0000	1.1522	-0.6063	Transformed Replacement : Transformed Transformed pdays
STD_REP_INV_INV_CAMPAIGN	STANDARDIZE	REP_INV_INV_CAMPAIGN	('REP_INV_INV_CAMPAIGN'n-0.7411046067)/0.0761269832	0	-0.9778	3.0028	0.0000	1.0000	0.6457	-0.5411	Transformed Replacement : Transformed Transformed campaign
STD_REP_SQRT_AGE	STANDARDIZE	REP_SQRT_AGE	('REP_SQRT_AGE'n-6.4299142161)/0.8809560436	0	-2.3509	3.0427	0.0000	1.0000	0.4608	-0.3288	Transformed Replacement : Transformed Age
STD_REP_SQRT_INV_PREVIOUS	STANDARDIZE	REP_SQRT_INV_PREVIOUS	('REP_SQRT_INV_PREVIOUS'n-1.3451103111)/0.1211339445	0	-2.8341	0.5705	0.0000	1.0000	-1.3206	0.0275	Transformed Replacement : Transformed Transformed previous
STD_REP_SQR_LG10_BALANCE	STANDARDIZE	REP_SQR_LG10_BALANCE	('REP_SQR_LG10_BALANCE'n-13.028297363)/1.0570526897	0	-3.5404	3.7499	0.0000	1.0000	1.3438	1.5226	Transformed Replacement : Transformed Transformed balance

Figure 60 – Transformations node (3) results summary

Metadata Changes

Name	Role	Level	Order	Transform
quarter	REJECTED			
days	REJECTED			

Figure 61 – Manage Variables node metadata changes

Variable Selection

Name	Variable Label	Variable Level	Role
Y		BINARY	TARGET
CONTACT_UNKNOWN		NOMINAL	INPUT
EDUCATION_TERTIARY		NOMINAL	INPUT
HOUSING_NO		NOMINAL	INPUT
JOB_RETIRED		NOMINAL	INPUT
LOAN_NO		NOMINAL	INPUT
MARITAL_MARRIED		NOMINAL	INPUT
POUTCOME_SUCCESS		NOMINAL	INPUT
QUARTER_2		NOMINAL	INPUT
QUARTER_3		NOMINAL	INPUT
STD_REP_INV_INV_CAMPAIGN	Transformed Replacement: Transformed Transformed campaign	INTERVAL	INPUT
STD_REP_SQR_LG10_BALANCE	Transformed Replacement: Transformed Transformed balance	INTERVAL	INPUT

Figure 62 – Features selected by variable selection node after voting combination

APPENDIX 4

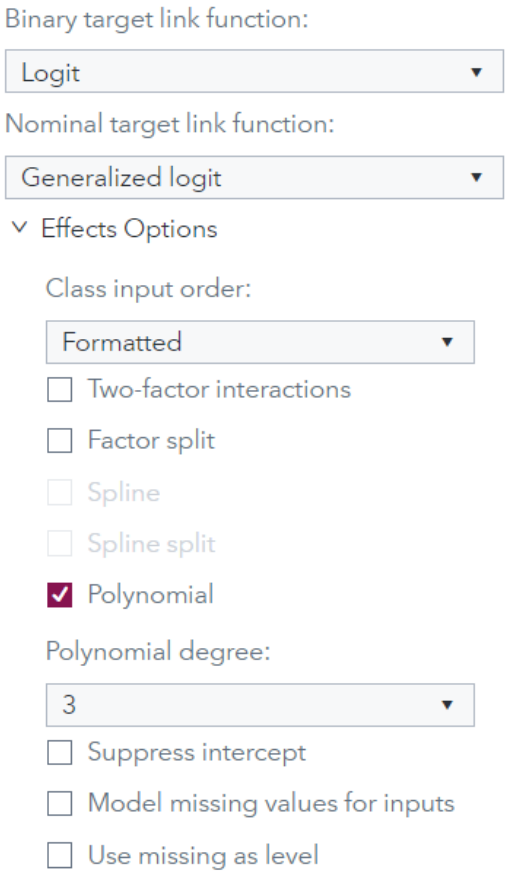


Figure 63 – Logistic regression node properties

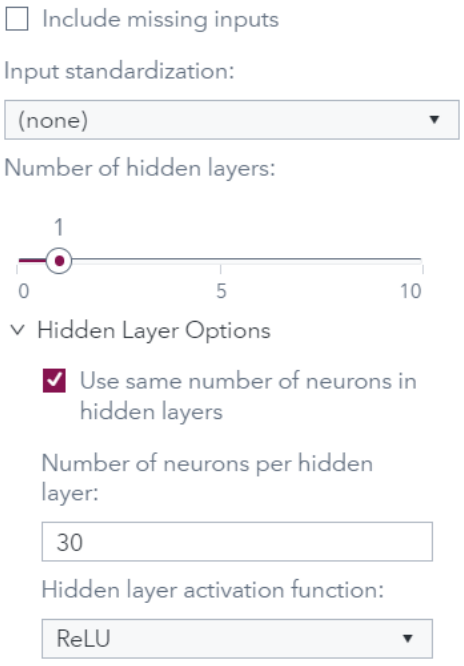


Figure 64 – Neural network node properties

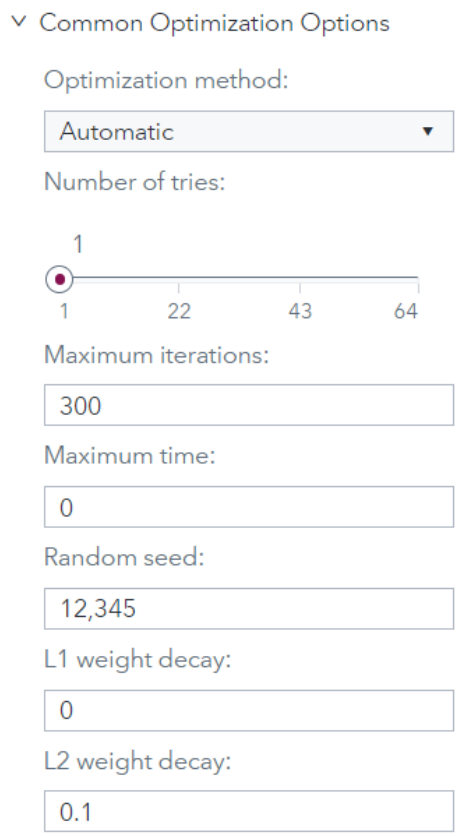


Figure 65 – Neural network node properties

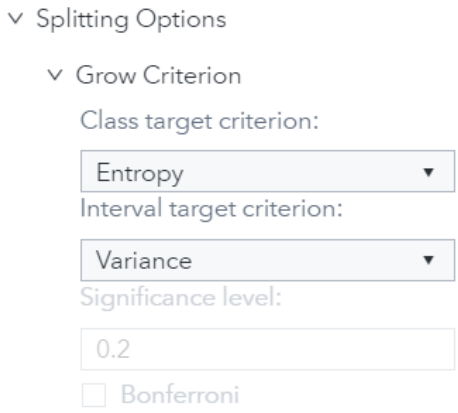


Figure 66 – Decision tree node properties

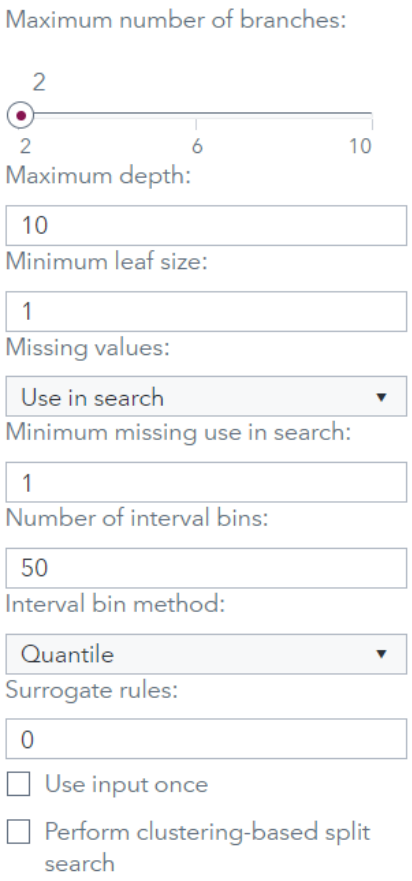


Figure 67 – Decision tree node properties

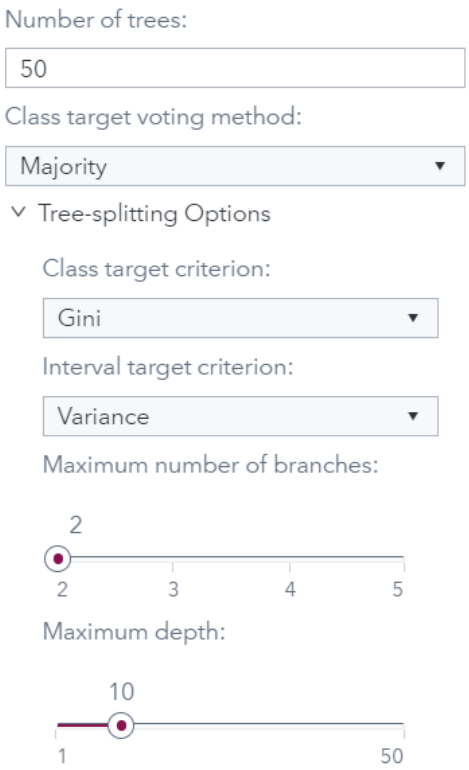


Figure 68 – Random forest node properties

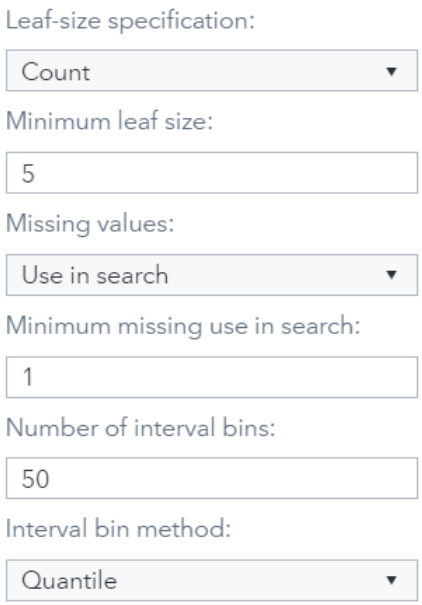


Figure 69 – Random forest node properties

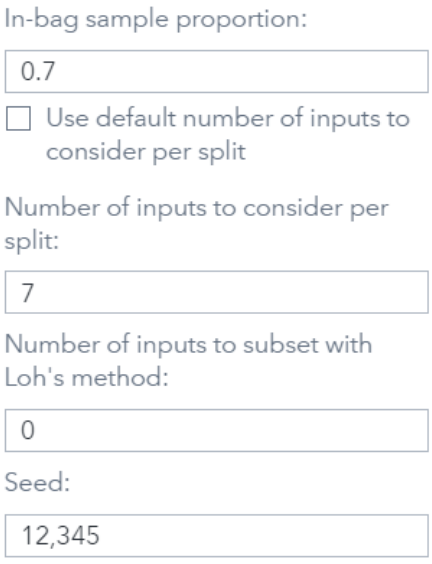


Figure 70 – Random forest node properties

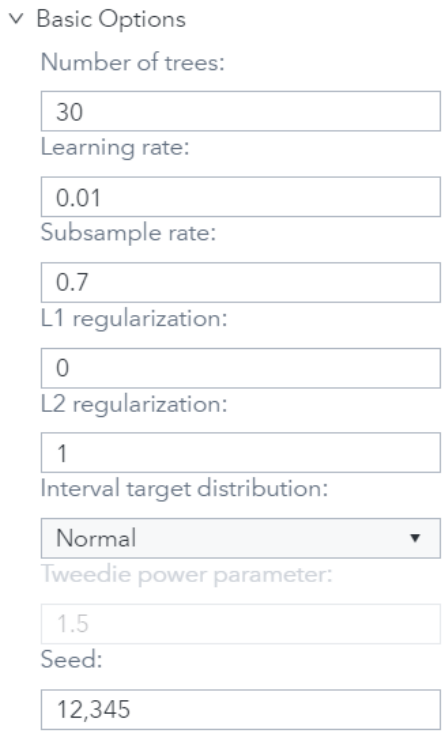


Figure 71 – Gradient boosting node properties

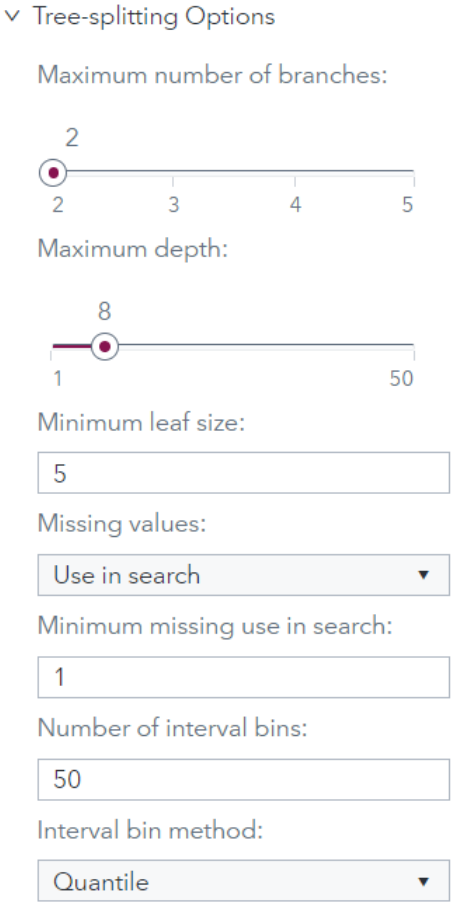


Figure 72 – Gradient boosting node properties

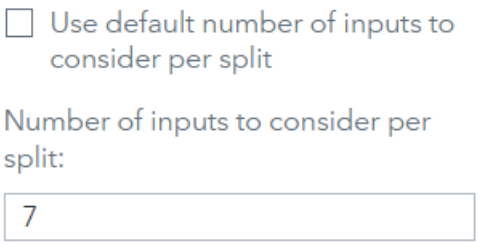
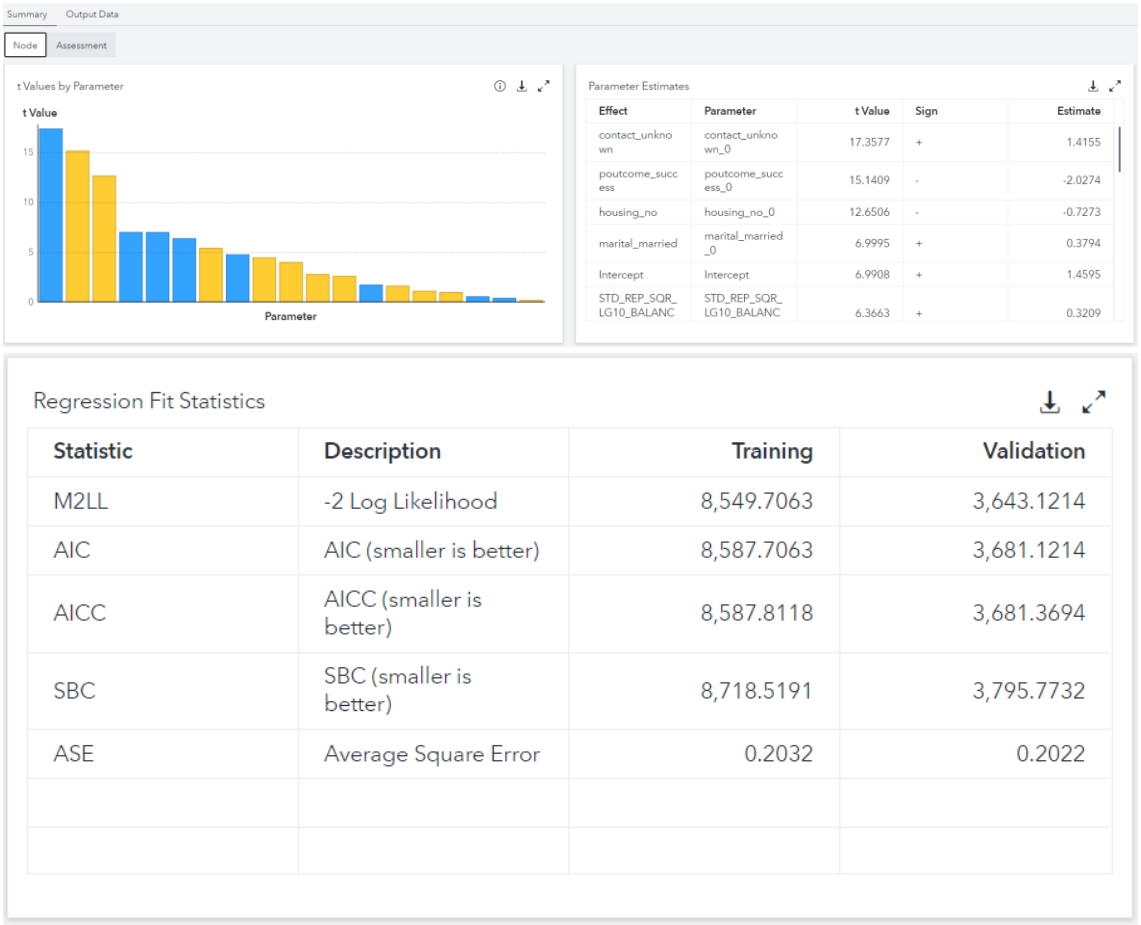


Figure 73 – Gradient boosting node properties



Regression Fit Statistics

Statistic	Description	Training	Validation
M2LL	-2 Log Likelihood	8,549.7063	3,643.1214
AIC	AIC (smaller is better)	8,587.7063	3,681.1214
AICC	AICC (smaller is better)	8,587.8118	3,681.3694
SBC	SBC (smaller is better)	8,718.5191	3,795.7732
ASE	Average Square Error	0.2032	0.2022

Figure 74 – Logistic regression node results

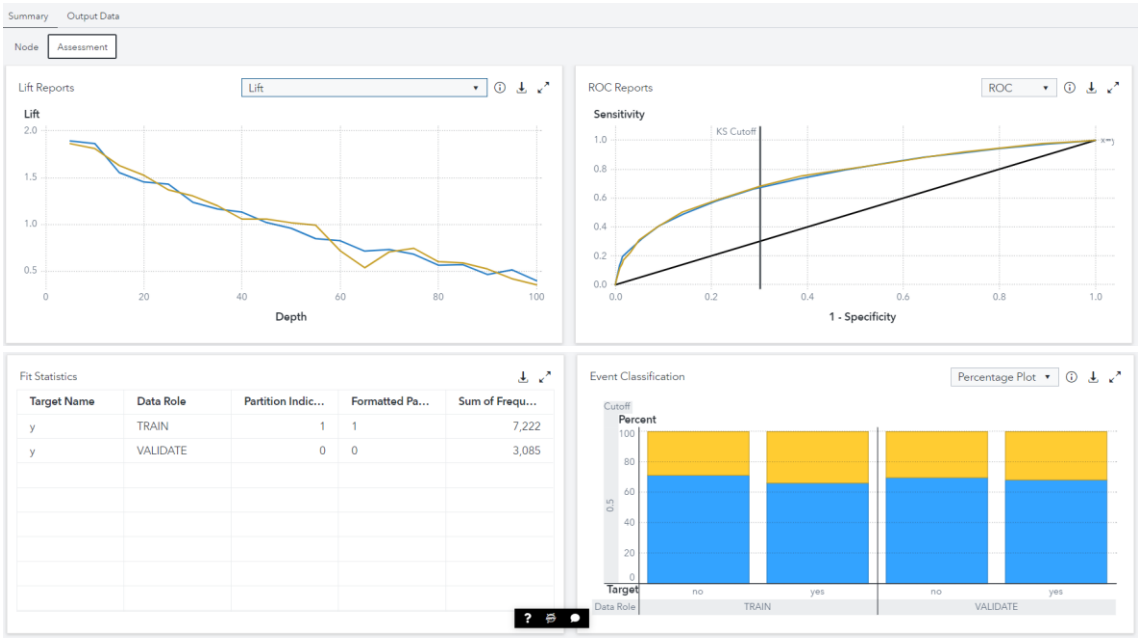


Figure 75 – Logistic regression node results

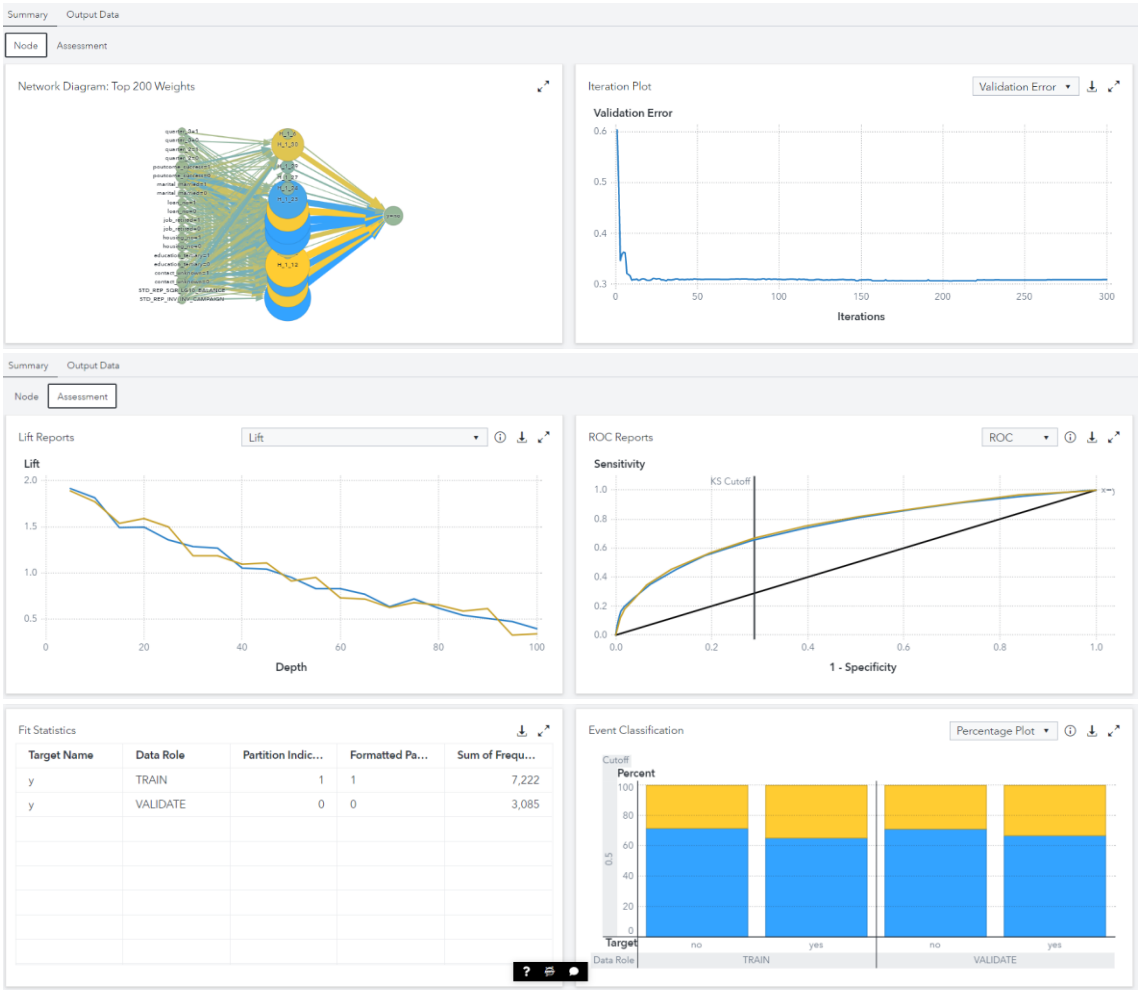


Figure 76 – Neural network node results

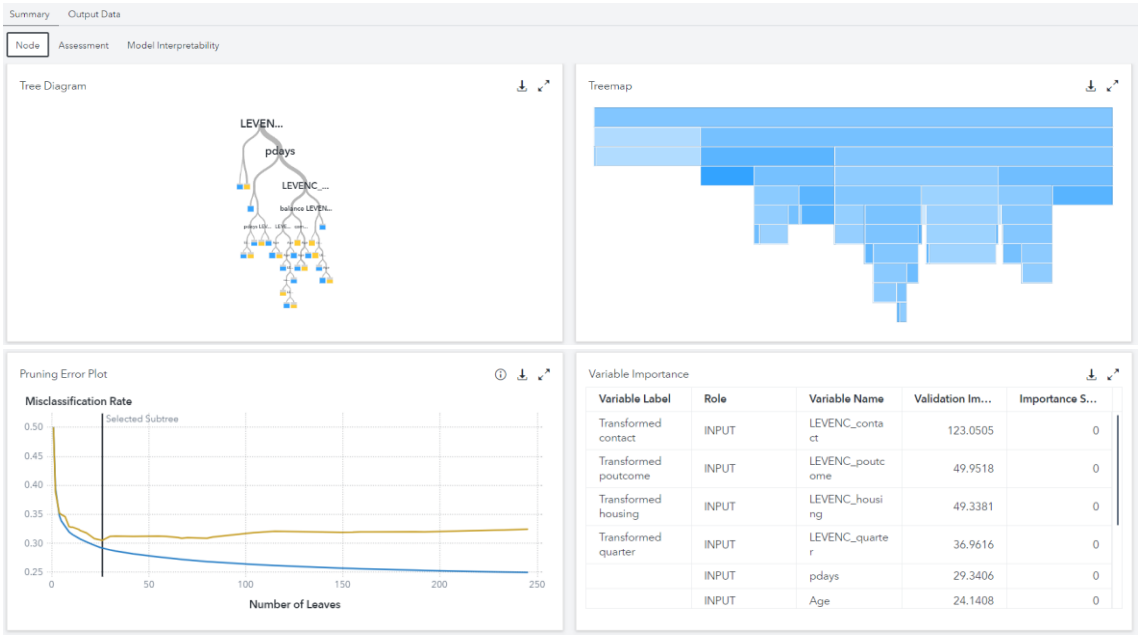


Figure 77 – Decision tree node results

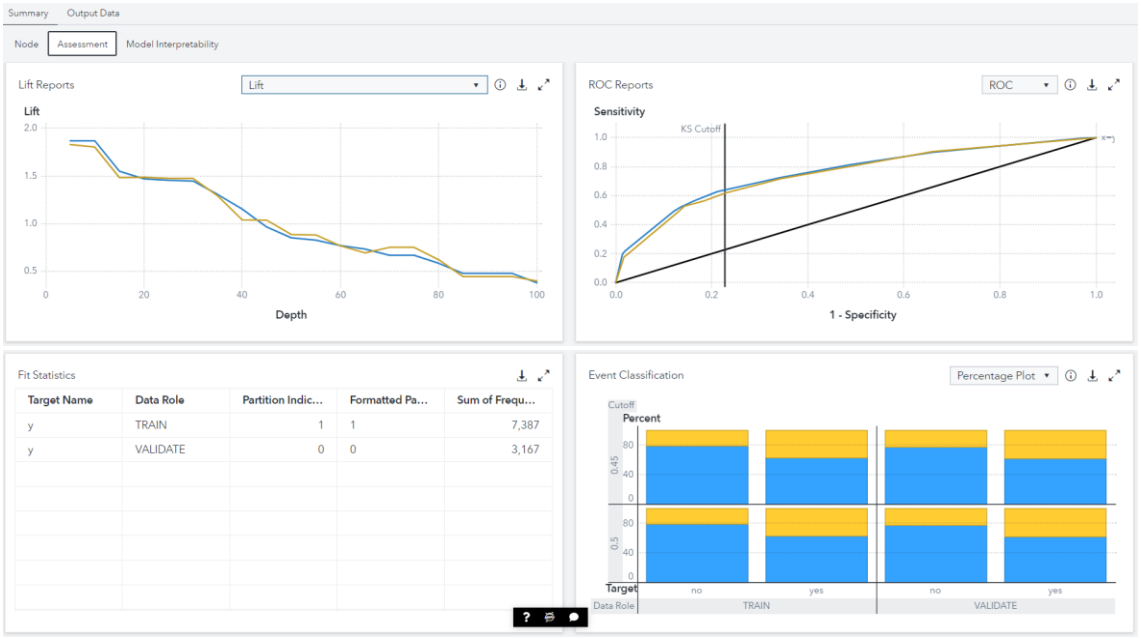


Figure 78 – Decision tree node results

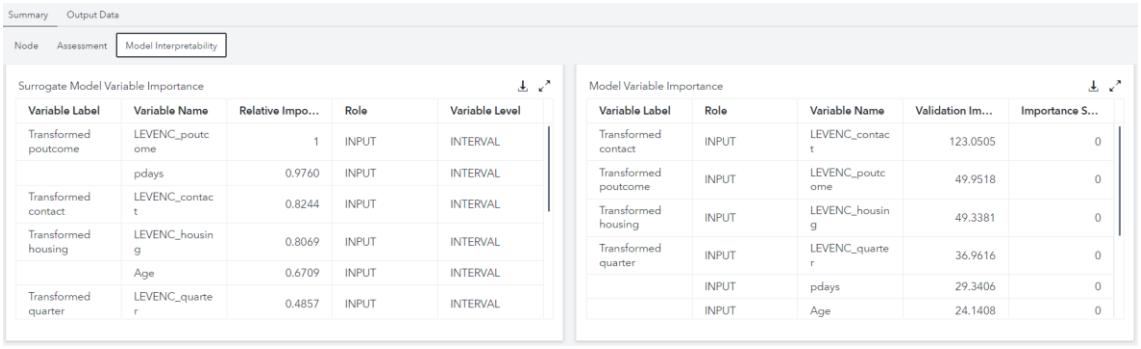


Figure 79 – Decision tree node results

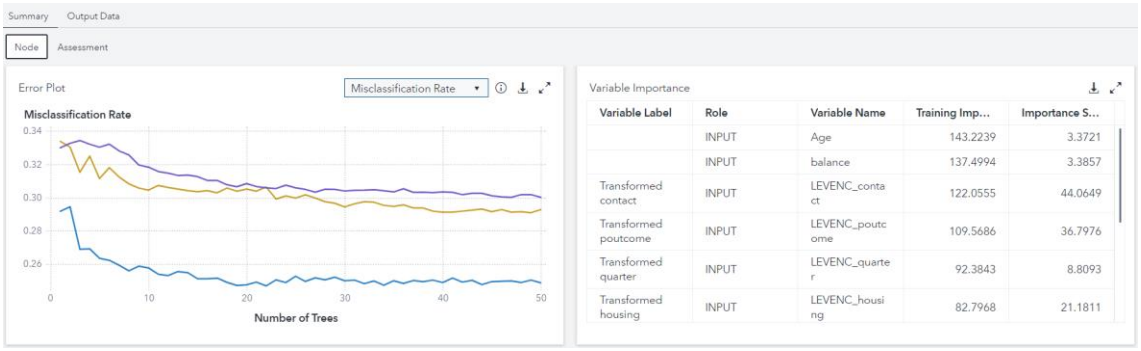


Figure 80 – Random forest node results

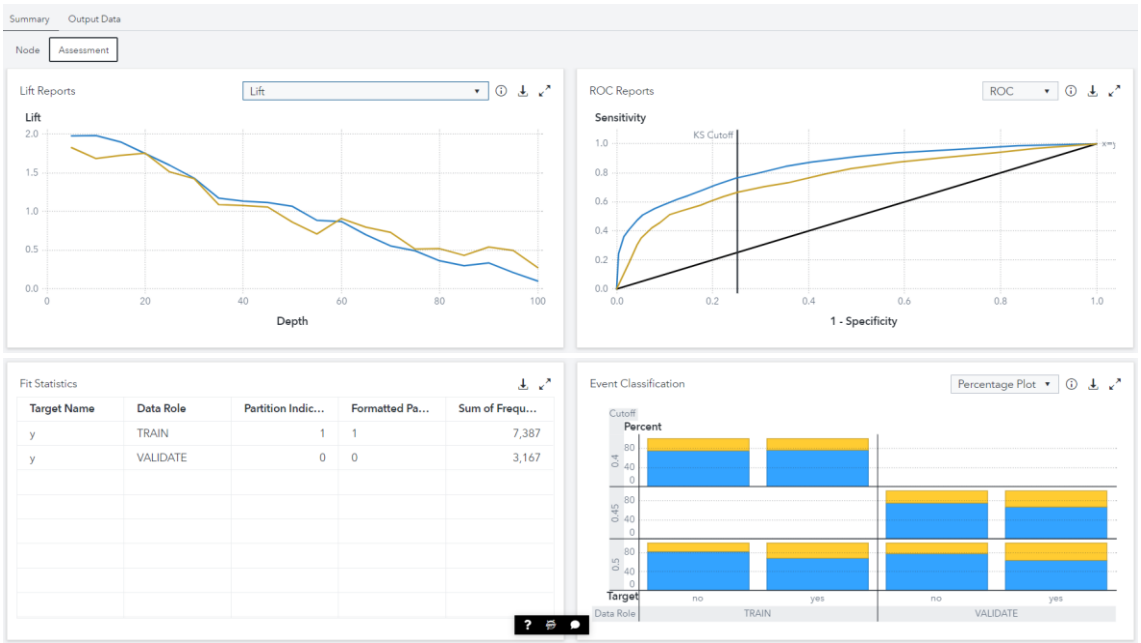


Figure 81 – Random forest node results

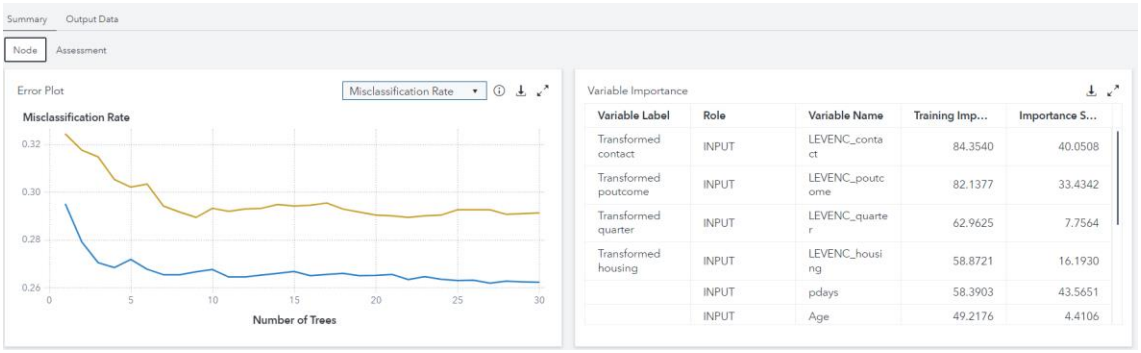


Figure 82 – Gradient boosting node results

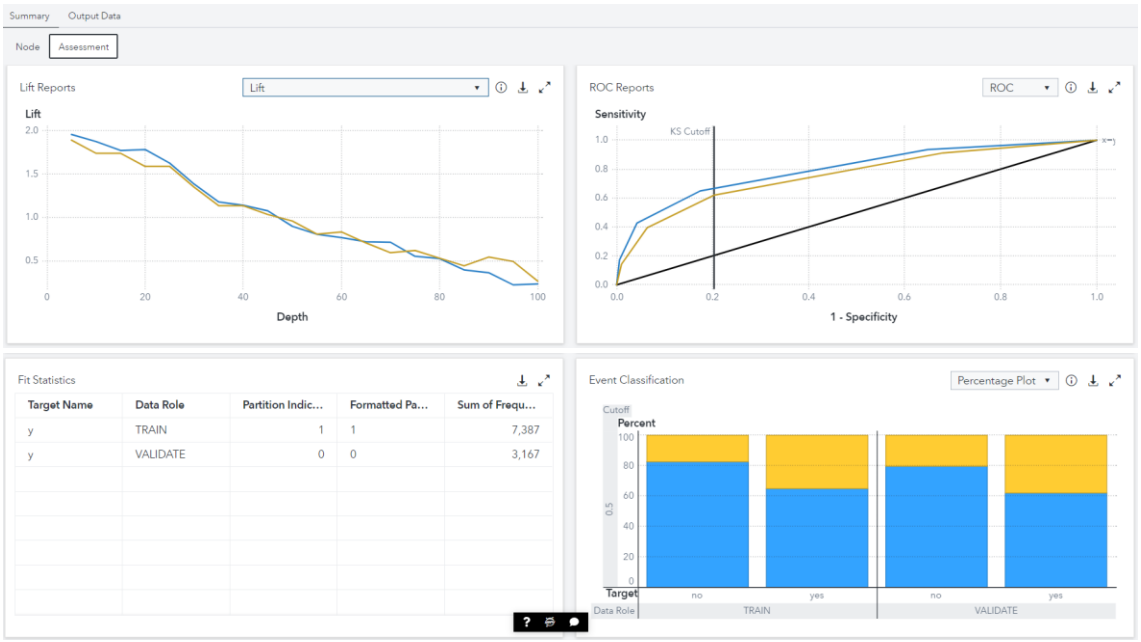


Figure 83 – Gradient boosting node results

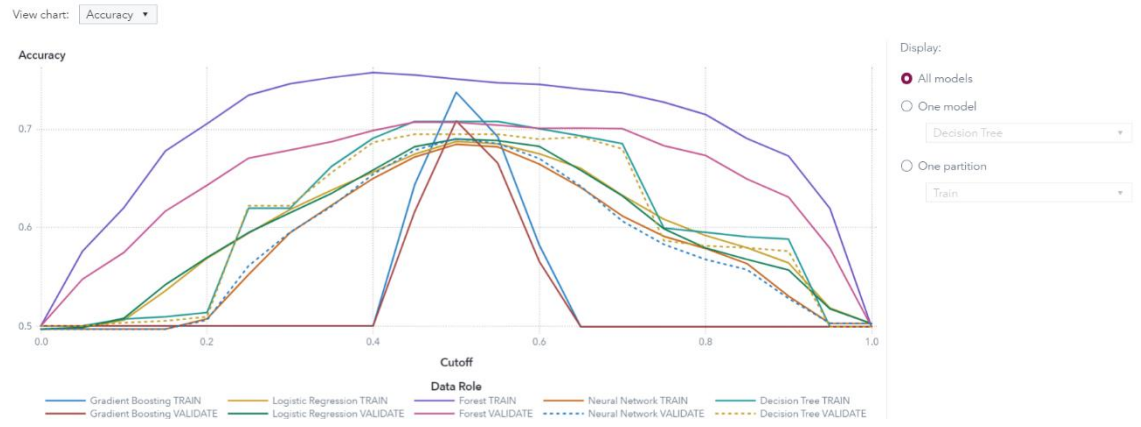


Figure 84 – Accuracy chart for all models and partition sets

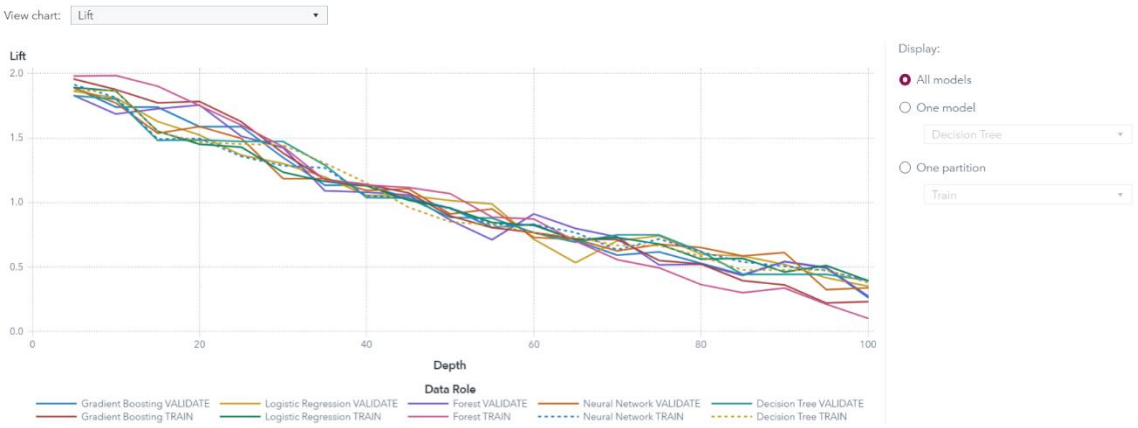


Figure 85 – Lift chart for all models and partition sets

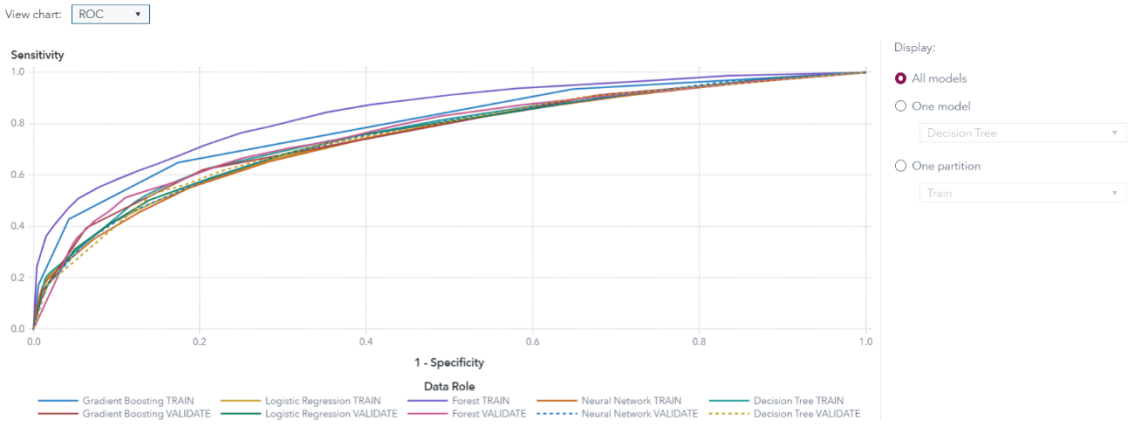


Figure 86 – ROC chart for all models and partition sets

APPENDIX 5

```
LogisticRegression
LogisticRegression(C=0.01, class_weight='balanced', max_iter=25, penalty='l1',
                   random_state=0, solver='saga')
```

Figure 87 – Example of Logistic regression hyperparameters

	precision	recall	f1-score	support
0	0.67	0.66	0.66	3606
1	0.66	0.67	0.67	3628
accuracy			0.66	7234
macro avg	0.66	0.66	0.66	7234
weighted avg	0.66	0.66	0.66	7234

Figure 88 – Example of a classification report for the training set

	precision	recall	f1-score	support
0	0.66	0.64	0.65	1551
1	0.65	0.68	0.66	1544
accuracy			0.66	3095
macro avg	0.66	0.66	0.66	3095
weighted avg	0.66	0.66	0.66	3095

Figure 89 – Example of a classification report for the validation set

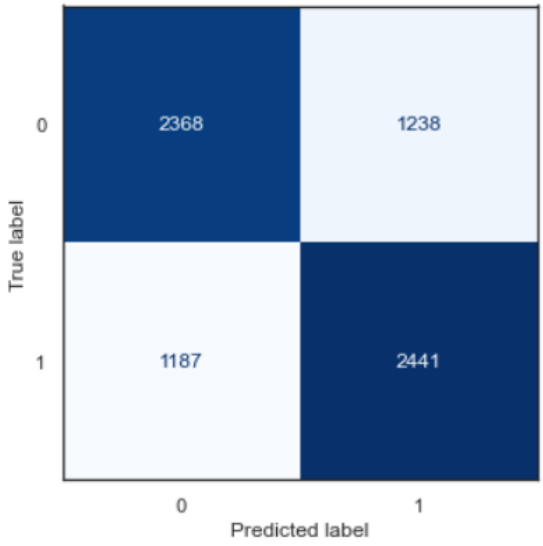


Figure 90 – Example of a confusion matrix for training set

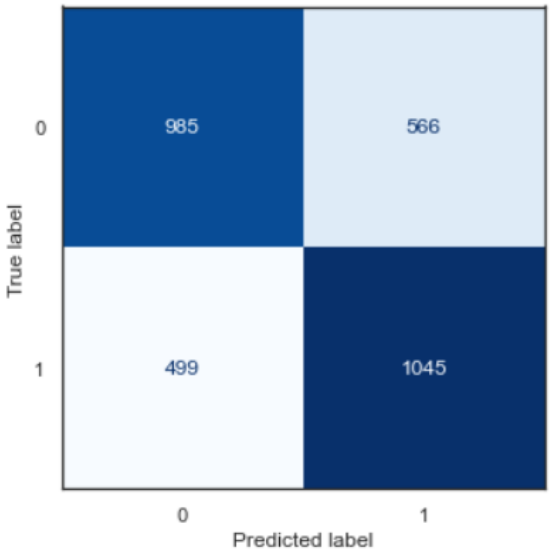


Figure 91 – Example of a confusion matrix for validation set

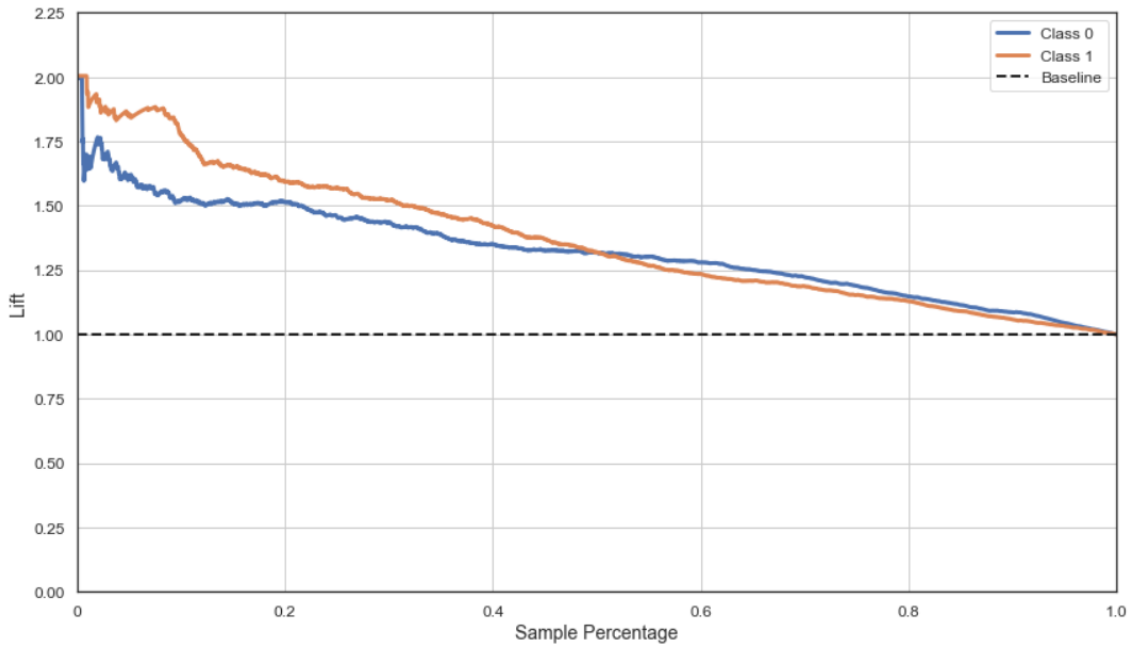


Figure 92 – Example of a lift chart

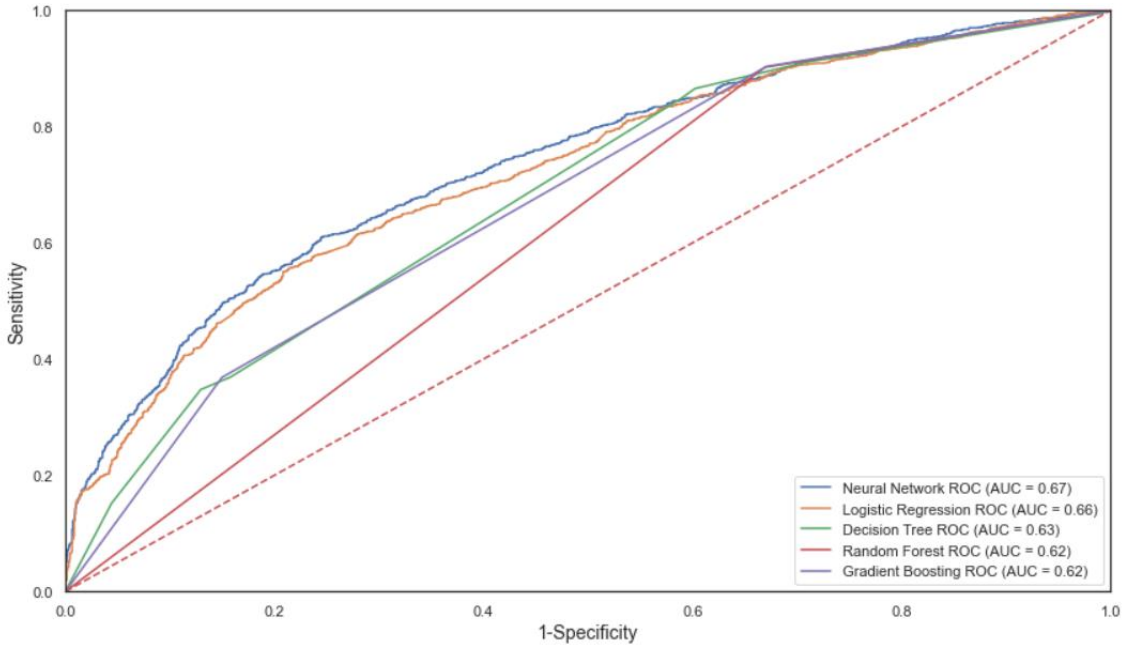


Figure 93 – ROC chart

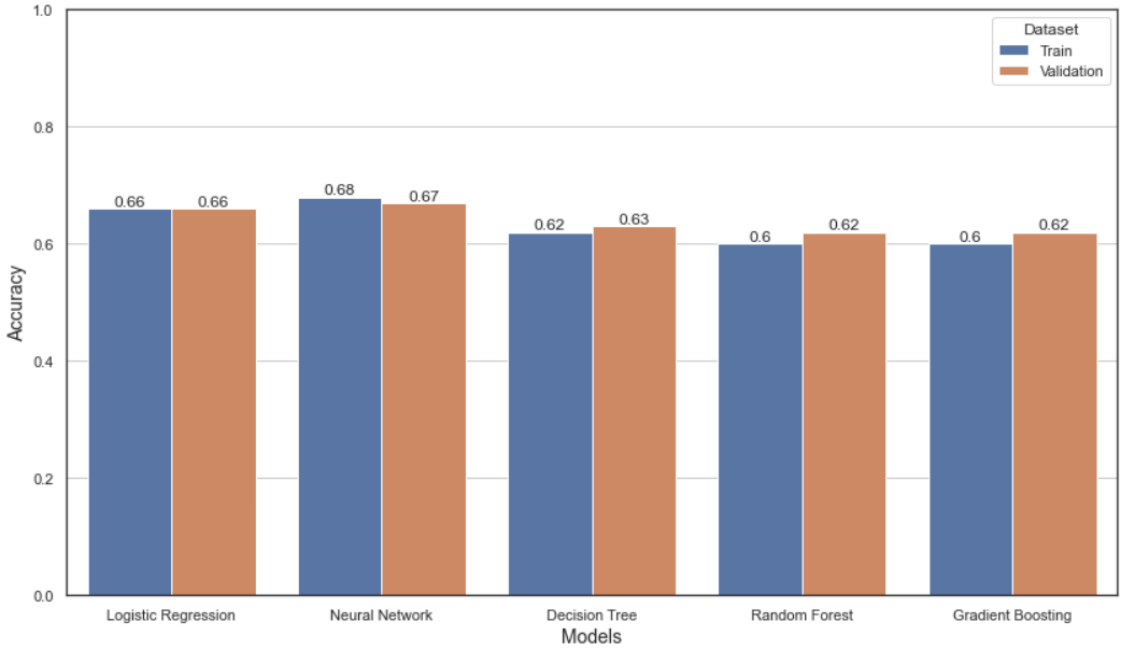


Figure 94 – Accuracy comparison barplot

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa