LUISS Guido Carli
LIBERA UNIVERSITA' INTERNAZIONALE DEGLI STUDI SOCIALI

Degree Program in Data Science and Management
Course of Data Visualization

Leveraging Machine Learning for Churn Prediction: Techniques, Challenges, and Integration with Customer Lifetime Value

SUPERVISOR
Prof. Blerina Sinaimeri

CANDIDATE
Lorenzo Carucci

STUDENT NUMBER
760981

CO-SUPERVISOR
Prof. Alessio Martino

ACADEMIC YEAR
2023/2024

A sincere thank you to my family for their unconditional support. I would also like to thank the Accenture team, particularly my tutor Saverio Salatino, for the opportunity and support provided throughout the course of my thesis. A special thanks goes to Professor Blerina Sinaimeri for her invaluable guidance

Abstract

This master's thesis provides an in-depth exploration of various machine learning methodologies aimed at improving the prediction of customer churn in the context of digital media services. The case study focuses on a leading provider of digital entertainment, where customer churn presents a significant challenge for long-term business sustainability, with a significant part of the work carried out by Accenture, in Technology - Cloud First division. The research was motivated by the need for efficient predictive models to help mitigate churn and enhance customer retention strategies.

The thesis investigates key machine learning algorithms, including Logistic Regression, Decision Forest, Neural Networks, and Boosted Decision Trees, all central to developing robust churn prediction models. The study employs a comprehensive dataset to construct and test various models, each evaluated for its ability to accurately predict customer churn. Additionally, a feature importance analysis is conducted to identify the most influential factors driving customer churn.

In conclusion, the combined use of multiple machine learning algorithms and ensemble methods significantly enhances the ability to predict customer churn. This enables better customer retention strategies and more informed decision-making within the company. The findings contribute to the field of predictive analytics by comparing various machine learning models and proposing an innovative ensemble approach, providing valuable insights for future applications in customer behaviour analysis and risk management.

For legal and privacy reasons, the real names of Accenture's client company and the device being analysed cannot be used in this work, and from now on, it will be referred to as OTT.

Table of Contents

1. Introduction 7

1.1 Motivations 7

1.2 Context of the Study 8

1.3 Overview of the Thesis 9

2. Literature Review 10

2.1 The Costumer Churn Problem 11

2.2 Key Terms and Definitions 12

2.3 Introduction to Churn Classification Models 13

2.4 Introduction to Gradient Boosting Techniques 16

2.5 LightGBM: An Optimized and Scalable Gradient Boosting Framework 18

2.6 Unbalanced Class Problem 18

2.7 SMOTE Approach 20

3. Data Overview and Methods 22

3.1 Approach Explanation 23

3.2 Input Data 25

3.2.1 Login Table 25

3.2.2 Content Table 26

3.2.3 Subscription 26

3.2.4 Watching Table 27

3.2.5 App Execution Table 28

3.2.6 Purchase 28

3.3 Data Transformation 29

3.4 Data Migration and Cloud Overview 31

3.5 CAR Dataset 34
3.5.1 Data Cleaning 37
3.5.2 Missing Values Handling 39
3.5.3 Data Exploration 42
4. Preprocessing and Model Overview 47
4.1 Data Processing Steps for Churn Analysis 47
4.2 Data Preprocessing: Correlation Matrix Considerations 48
4.2.1 Data Preprocessing: Filter-Based Feature Selection Module Settings 52
4.2.2 Data Preprocessing: Feature Selection 55
4.2.3 Final CAR Dataset 58
4.3 Split and SMOTE Operations 60
4.4 Algorithms Explanation 62
4.4.1 Two-Class Logistic Regression 62
4.4.2 Two-Class Neural Network 63
4.4.3 Two-Class Decision Forest 64
4.4.4 Two-Class Boosted Decision Tree 65
4.5 Result 66
4.6 Tuning 69
4.7 Tuning Result 71
5. Model Implementation 76
5.1 Model Inference: Application for Predictions on New Data 76
5.2 Churn Probability and Targeted Retention Strategies Based on Risk Levels 77
6. Conclusion 79
7. References 81

List Of Figures

Figure 3.1: Process steps

Figure 3.2: Data manipulation process

Figure 3.3: Import Query

Figure 3.4: Azure Dashboard

Figure 3.5: Target Classed Distribution Pie Chart

Figure 3.6: SQL Transformation Input Data

Figure 3.7: SQL Query from Three Class to Two

Figure 3.8: Linear Correlation and Principal Statistics

Figure 3.9: Columns Dropped Operations

Figure 3.10: Cleaning Missing Data Operation

Figure 3.11: Mont Login Distribution

Figure 3.12: Device Used Distribution

Figure 3.13: Content Costumer Distribution

Figure 3.14: App Login Distribution

Figure 4.1: Correlation Matrix

Figure 4.2: Code to calculate PCA variance

Figure 4.3: Filter Based Feature Selection

Figure 4.4: 15 Most Correlated Variables with Target Variable

Figure 4.5: Filter Code High Correlation Variables

Figure 4.6: Variables Correlation above 0.9

Figure 4.7: Final CAR Dataset

Figure 4.8: Split and SMOTE Operation in Azure

Figure 4.9: Models Performance Comparison

Figure 4.10: Models Performance Comparison After Tuning

Figure 4.12: Confusion Matrix Comparison

Figure 4.11: Roc Curves Comparison

Figure 5.1: Churn Strategies

List Of Tables

Table 3.2: Input data tables with related number of rows and columns

Table 3.2: CAR dataset features

Table 3.3: Name and Number of Columns with Missing Values

Table 4.1: Variables Name Dropped and Reason

Table 4.2: Variables Name Dropped from Original Dataset

Table 4.3: Models Performance

Table 4.4: Models Performance After Tuning

Chapter 1

Introduction

1.1 Motivations

Understanding customer behaviour and predicting their future choices is crucial for any company. In today's highly competitive market, businesses must not only attract customers but also retain them by anticipating their needs and preferences. By gaining insights into consumer behaviour, companies can tailor their products and services to meet the evolving demands of their target audience. This proactive approach not only enhances customer satisfaction but also drives business growth and sustainability. Moreover, with the advent of advanced data analytics and machine learning, companies now have powerful tools at their disposal to analyse vast amounts of customer data and derive actionable insights. Therefore, investing in understanding and predicting customer behaviour is not just beneficial but essential for any company aiming to stay ahead in the market. This study is motivated by the need to understand and predict customer behaviour. It aims to harness the capabilities of advanced Machine Learning algorithms to develop effective predictive models that can provide valuable insights into future customer choices. In this way,

businesses can enhance their strategic planning and decision-making processes.

1.2 Context of the Study

This thesis is conducted within the framework of a leading European telecommunications company's operations, with a market capitalization of approximately ten billion euros in 2023. The company offers a wide range of services including fixed and mobile telecommunications, internet services, digital content, and entertainment solutions through its OTT (over the top video system) platform. The OTT platform is the company's on-demand streaming service, providing a vast selection of movies, TV series, documentaries, and sports content. The OTT, therefore, offers the possibility, through a set-top box, to access a portal where various external partner applications such as DAZN, Netflix, and many others are available, as well as the OTT's own app. This study is conducted in collaboration with Accenture, in Technology - Cloud First division, a leader in providing cloud-based solutions to business challenges. This collaboration has enriched the research by providing access to robust, industry-proven methodologies and high-quality data. The research is specifically designed to predict customer behaviour related to content consumption and subscription services on the OTT platform. Understanding and anticipating customer preferences is crucial for the company, as it allows for improving content offerings, increasing customer satisfaction, and optimizing marketing and retention strategies. Moreover, with the advent of advanced Machine Learning and Big Data technologies, the company can analyse large volumes of customer data to extract valuable insights and make more informed decisions. This data-driven approach not only helps enhance the user experience on the OTT platform but also strengthens company's position as a leader in the telecommunications and digital services industry.

1.3 Overview of the Thesis

The thesis is structured to provide a thorough understanding of the customer churn problem and the proposed solutions. Chapter 2 presents an extensive literature review, discussing the issue of customer churn in digital media services and its implications for business sustainability. It also introduces key concepts and machine learning techniques, such as Logistic Regression, Decision Forest, Neural Networks, and Boosted Decision Trees, which form the theoretical foundation of the study. Chapter 3 provides a detailed overview of the dataset used, highlighting all the steps involved to get the final dataset. Chapter 4 is dedicated to presenting the machine learning models developed, evaluating their performance metrics, and emphasizing the importance of different features in influencing churn predictions. In Chapter 5 is discussed the implementation of the best performer model and the possible strategies in churn context. Finally, Chapter 6 concludes the study by summarizing the findings, discussing their implications, and offering insights into the potential of machine learning for predicting customer churn and supporting more effective customer retention strategies.

Chapter 2

Literature Review

Summary

The literature review provides an in-depth analysis of several key topics relevant to the research conducted in this thesis.

The first area of exploration is the issue of churn, a significant problem affecting businesses globally. Emphasizing its consequences the necessity of efficient and accurate predictive models for churn prediction becomes clear.

The review then explores the essential terminology and foundational concepts, equipping readers with a thorough understanding of the language and ideas employed throughout the study.

Further on, the focus shifts to examining the different kinds of churn and specifics of machine learning techniques applied to churn prediction. Particular attention here will be given to the Gradient Boosting Machines (GBMs) technique and to LightGBM, a highly efficient variant of Gradient Boosting Decision Trees.

Finally, the literature review investigates how to handle unbalanced datasets with a particular focus on SMOTE operation.

These insights shed light on the applications of machine learning methods to solve the problem of churn, providing valuable context and guiding the development of the models in this study.

2.1 The Costumer Churn Problem

In recent years, the number of mobile phone users has grown significantly. In developed countries, wireless telecommunications markets are becoming saturated, with mobile phone penetration rates exceeding 100%, indicating more subscriptions than inhabitants. Consequently, customer retention has become a priority for telecommunications operators [1].

The term "Customer Churn" refers to the loss of customers from a service company. It has a substantial impact on company sales and incurs opportunity costs, as acquiring new customers is much more expensive than retaining existing ones. For this reason, businesses implement customer retention strategies and often depend on their ability to forecast which customers are likely to leave [2].

Long-term customers are more profitable and less sensitive to competition, and losing customers results in opportunity costs due to reduced sales. Even a small improvement in customer retention can significantly increase profits [1].

Models for predicting customer churn are designed to identify customers who are most likely to leave, enabling companies to enhance the effectiveness of retention strategies and minimize the costs related to customer loss [1].

As stated in [3], there are three main types of churns:

- Active churn, where the customer decides to terminate the contract and switch to another provider due to dissatisfaction or more advantageous offers.

- Rotational churn, where the customer ends the contract without intending to switch to a competitor, typically due to changes in personal circumstances, such as financial issues or geographic relocation.
- Passive or involuntary churn, where the company itself terminates the contract for reasons such as non-payment of bills.

Additionally, churn can be divided into:

- Total churn, where the contract is completely cancelled.
- Hidden churn, where the contract remains active, but the customer no longer uses the service.
- Partial churn, where the customer significantly reduces their use of the service while maintaining a minimal connection with the company.

2.2 Key Terms and Definitions

In this section, we define several key terms referenced throughout the literature review to enhance readers' understanding of the discussed concepts.

As we said in the previous paragraph Customer Churn refers to the loss of customers. Also known as customer attrition, defection, or turnover, this phenomenon is inevitable for any service business. It holds significant importance for companies, and its measurement is reflected in a key performance indicator known as the churn rate [4]. Churn rate is a critical metric that represents the percentage of customers who discontinue their relationship with the company over a specific period. It is calculated by dividing the number of churned customers by the total number of customers at the start of the period [5].

It can be categorized into voluntary churn where customers leave by their own choice, often due to issues such as unmet expectations or switching to competitors, and involuntary churn. In contrast, involuntary churn occurs for reasons beyond the customer's control, such as failed

payments due to expired credit cards, insufficient funds, or payment system errors [6].

A thorough understanding of these terms is essential for grasping the concept of churn prediction and its implications for maintaining customer relationships and overall business stability.

2.3 Introduction to Churn Classification Models

In this paragraph, the literature on the machine learning algorithms that will be used in this paper will be explored in depth.

In the research community, different machine learning algorithms have been proposed to face the churning prediction problem. Such methods include Logistic Regression, Decision Trees, Neural Network, Regression Analysis, Support Vector Machines, Naïve Bayes, Decision Forest, Sequential Pattern Mining and Market Basket Analysis, Linear Discriminant Analysis, and Rough Set Approach [7].

Here the focus will be on only four algorithms which will be the ones used: Neural Network, Logistic Regression, Decision Forest, and Decision Trees.

Artificial Neural Networks (ANNs) is a common algorithm to address complex problems, such as the churn prediction problem [7].

Artificial neural networks are computational systems designed based on the structure and function of the human brain. They consist of multiple layers of nodes, or artificial "neurons." Each node in a neural network acts as a computational unit that receives one or more inputs, processes them using a set of weights and an activation function, and produces an output. Artificial neural networks are used to tackle complex problems such as pattern recognition, classification, and regression [8].

Artificial neural networks function through a supervised learning process, where the network is trained using pairs of input data and their associated outputs. The process starts with a Forward Pass, during which the inputs

flow through the network and produce a prediction. Following this, the Error Calculation takes place, where the discrepancy between the predicted output and the actual target value is assessed using a cost function, such as mean squared error. Lastly, in the Backward Pass, the computed error is sent backward through the network, allowing the weights to be adjusted by an optimization algorithm like gradient descent, to minimize the error in future iterations [9].

The second model tested is **Logistic Regression**. Regression is a statistical technique used to estimate relationships between different variables. It encompasses a variety of methods for modelling and analysing multiple variables, with a particular focus on the connection between a dependent variable and one or more independent variables. Regression can be divided into linear regression and logistic regression. In the context of customer churn, linear regression is not widely used, as it is more suitable for predicting continuous values. In contrast, Logistic Regression is a probabilistic statistical model used for classification. This type of regression is employed to generate binary predictions on a categorical variable, such as customer churn, based on one or more predictor variables, like customer characteristics. [7].

According to [10] this is how the model works: logistic regression calculates the probability that a given instance belongs to one of the two classes. This is done using the logit function.

$$logit(p) = \log\left(\frac{p}{1-p}\right) = \beta 0 + \beta 1X1 + \beta 2X2 + \cdots + \beta nXn$$

Where p is the probability of belonging to class 1 (positive), β0 is the intercept, and β1,β2,...,βn are the coefficients of the predictor variables X1,X2,...,Xn. Logistic regression uses a sigmoid function to map any real value to a range between 0 and 1:

$$p(X) = \frac{1}{1 + e^{\{-(\beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n)\}}}$$

This result represents the estimated probability that the outcome is positive (class 1). A threshold (usually 0.5) is also used to decide which

class to assign an observation to. If p(X)>0.5 the observation is classified as class 1 otherwise as class 0.

Then I tested a **Decision Tree** that is are hierarchical structures designed to represent a series of decisions that can generate classification rules for a specific dataset. In these structures, leaves denote class labels, while branches represent the combinations of features that lead to those labels. Although Decision Trees may struggle with capturing complex, non-linear relationships between attributes, they can still achieve high accuracy in predicting customer churn, depending on the nature of the data being analysed [7]. In this work, however, a classic version of the Decision Tree will not be used. On Azure ML, an algorithm called Two-Class Boosted Decision Tree will be used. This algorithm uses an implementation of Gradient Boosting. This implementation is optimized for binary classification tasks. Specifically, it leverages the LightGBM framework, which is known for its efficiency and scalability. LightGBM, developed by Microsoft, is a highly efficient gradient boosting framework based on decision tree algorithms, and it forms the core of Azure's boosting capabilities. This topic, however, will be explored in more detail later in this chapter.

The last model is a **Decision Forest** is an ensemble learning method primarily used for classification and regression tasks. It operates by constructing a multitude of decision trees during training time and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. The underlying principle behind Decision Forests is to combine the predictions of multiple decision trees to improve accuracy and avoid overfitting [11]. The strength of the Random Forest algorithm lies in its ability to manage diverse features and capture non-linear relationships, which enhances its overall predictive performance for the churn task [12].

2.4 Introduction to Gradient Boosting Techniques

In these paragraphs, the gradient boosting technique and the LightGBM framework will be explored in more detail as it will be used throughout this paper to address the problem of churn classification.

First, it is important to specify that Boosting is a method through which we can apply an ensemble learning method, which means using more than one classifier. This will reduce the probability of making a wrong prediction, assuming that classifiers are independent. The goal of ensemble methods is to combine different classifiers into a "metaclassifier" that has better generalization performance than each individual classifier alone. Ensemble methods can be built from different classification algorithms (for instance decision trees, SVM, logistic regression classifiers) or using the same classification algorithm and fitting different subsets of the training data (ex-random forest). There are three other methods through we can use ensemble classifiers: majority voting, bootstrap/bagging and stacking but they will not be discussed in this context.

Boosting has been effectively utilized for predicting customer churn in both retail and telecommunications sectors. [7]

The idea behind boosting is that each subsequent predictor focuses on correcting the errors made by the previous predictors. Classifier 1 is trained on the entire dataset, with equal weights assigned to each example. After training, the weights of the examples that were misclassified by Classifier 1 are increased. Classifier 2 is then trained on the same dataset, but with these adjusted weights, emphasizing the harder-to-predict examples. Classifier 3 follows a similar process, focusing even more on the examples that the previous classifiers struggled with. The final prediction is made based on a weighted combination of the predictions from all classifiers, where more accurate classifiers have a greater influence [7].

There are many forms of the boosting algorithm, but the most common are AdaBoost and Gradient Boosting. This paragraph will cover only the second.

The key idea behind GBMs (Gradient Boosting Machines) is to iteratively improve the model by focusing on correcting the errors made by previous models. At the heart of GBMs are decision trees. These decision trees are used as the building blocks for the ensemble. The process involves adding trees sequentially, where each new tree is trained to correct the residual errors from the previous trees. This method leverages gradient descent to minimize a loss function, optimizing the model's performance iteratively [13].

According to [13], in more mathematical words, given a loss function $L\big(y, F(x)\big)$, where *y* represents the observed value, $F_{(x)}$ is the model's prediction, and *x* denotes the input features, the goal is to find an $L\big(y, F(x)\big)$ that minimizes the expected value of $L\big(y, F(x)\big)$. The model begins with an initial prediction $F_{0(x)}$, and at each iteration, a new tree $h_{(x)}$ is added to further reduce the loss. The final model $F_{m(x)}$ is composed of the sum of *m* trees:

$$F_{m(x)} = F_{\{m-1\}(x)} + h_{m(x)}$$

The tree $h_{m(x)}$ is selected to minimize the sum of the losses over all training instances. The parameter updates are guided by the gradient of the loss function with respect to the predicted values. In this way, GBMs progressively learn from the residuals of previous trees, continuously improving the reduction of overall loss.

GBMs are highly effective in modelling complex, non-linear relationships, and they are capable of handling large and high-dimensional datasets. In conclusion, GBMs are versatile and powerful algorithms that offer robust predictive capabilities, making them an essential tool in various machine learning applications.

2.5 LightGBM: An Optimized and Scalable Gradient Boosting Framework

LightGBM (Light Gradient Boosting Machine), is a machine learning algorithm that implements the Gradient Boosting Decision Tree (GBDT) method [14].

LightGBM is recognized for its efficiency and strong performance in a variety of machine learning tasks, including multi-class classification. A key distinction between LightGBM and algorithms like XGBoost is the approach to constructing decision trees. Unlike XGBoost, which builds trees level by level, LightGBM grows them leaf-wise, focusing on the leaves that will most effectively reduce loss. To enhance computational efficiency, LightGBM introduces two innovative techniques: Gradient-based One-Side Sampling (GOSS), which speeds up the calculation of information gain by excluding a large portion of data with small gradients, and Exclusive Feature Bundling (EFB), which reduces data dimensionality by grouping mutually exclusive features. These innovations make LightGBM significantly faster than models like XGBoost, while delivering comparable or even superior accuracy [14].

2.6 Unbalanced Class Problem

This session will cover a topic that is prevalent in nearly all churn classification tasks. Class imbalance is a significant feature of many datasets utilized in churn prediction modelling. A common bias of many standard classification algorithms is to show a tendency to favour the majority class, which can negatively impact their performance, particularly when it comes to accurately predicting the minority class [15].

The are different challenges related to class imbalance like, the unequal distribution between classes, which makes it difficult for classifiers to accurately predict the minority class, as many algorithms tend to focus on

overall accuracy, overlooking the imbalance. Additionally, the limited availability of data for the minority class restricts the information available to the classifier, making learning more complex. Another significant challenge is the overlap between classes, where shared characteristics make it difficult to distinguish between the two classes. Finally, internal class imbalance, known as small disjuncts, arises when sub-groups within a class have different sizes, further complicating classification accuracy [16].

There are two common methods to handle class imbalance classification: 1) Data-Level Approaches and 2) Algorithm-Level Approaches.

As described in this paper by Ali, Shamsuddin and Ralescu [16] **data-level approaches** for handling class imbalance focus on manipulating data before classification, primarily through sampling and feature selection techniques.

Sampling can be divided into under-sampling, which reduces instances of the majority class but may remove valuable information, and over-sampling, which increases minority class examples but risks overfitting, as seen with the SMOTE technique. Advanced methods, such as cluster-based sampling, improve accuracy while mitigating the limitations of traditional techniques.

Feature selection helps eliminate irrelevant or redundant features, enhancing the model's ability to distinguish between classes and reducing computational complexity. However, the application of feature selection to imbalanced classes is still underexplored, offering research opportunities to develop more effective approaches. While these methods are essential, they also present challenges like the risk of overfitting and information loss, necessitating the development of more sophisticated techniques.

Now the focus shifts to the different methods aimed at addressing the class imbalance problem through direct modifications to learning algorithms (**Algorithm-Level Approach**) rather than through data manipulation.

As stated in [16] improved algorithms have been adapted to effectively learn from the imbalanced class distribution, enhancing the model's ability to recognize the minority class. Examples like z-SVM and GSVM-RU introduce additional parameters and granular computing techniques to optimize class separation. "One-class learning" algorithms focus exclusively on the minority class, avoiding biases introduced by learning from majority classes. An example is CCNND, which utilizes the distribution of nearest neighbor distances to improve the classification boundary for the minority class. Cost-sensitive learning is another approach, where classifiers are modified to assign higher error costs to the misclassification of the minority class, as seen in cost-sensitive SVMs and neural networks optimized with Partial Swarm Optimization (PSO).

As described in this paper by Zhu, Baesens and Broucke [15] there are also various ensemble solutions to address class imbalance in churn prediction by combining learning models to improve performance.

These include bagging and its variants, such as UnderBagging and SMOTEBagging, which use under-sampling and over-sampling to balance the data. Boosting methods, like RUSBoost and SMOTEBoost, progressively correct errors, while balanced and weighted random forests place greater penalties on minority class misclassifications.

Finally, hybrid solutions like EasyEnsemble and BalanceCascade incorporate under-sampling and sequential processing to enhance predictions.

2.7 SMOTE Approach

In this last paragraph of the chapter more attention will be given to the SMOTE technique since, as will be seen in the chapter on Models, it was the technique implemented.

SMOTE (Synthetic Minority Over-sampling Technique) is an over-sampling technique that generates new synthetic examples for the minority class in an imbalanced dataset. The goal is to balance the class distribution

without simply replicating existing data, but by creating new data points that enhance the model's ability to generalize [17].

This method helps reduce overfitting compared to traditional oversampling, which simply replicates existing minority samples [18].

SMOTE employs the K-nearest neighbours' algorithm to identify the closest neighbours of minority samples. These neighbours are then used to generate new synthetic samples along the line connecting them to the original sample [18].

According to [draft3] the algorithm works like this:

- A starting point is defined: For each example X_0 belonging to the minority class, *K* nearest neighbours are selected, which also belong to the minority class.

A new synthetic example *Z* is generated along the line connecting X_0 and one of its *K* nearest neighbours X. The formula used to generate *Z* is, where *w* is a random number uniformly distributed in the interval [0,1]:

$$Z = X_0 + w \cdot (X - X_0)$$

SMOTE's mathematical formula generates new synthetic samples by calculating the distance between existing minority samples and their nearest neighbours, then multiplying that distance by a random number between 0 and 1[18].

SMOTE offers several advantages and disadvantages. Among its strengths, SMOTE is widely used for its simplicity and effectiveness in improving classifier performance on imbalanced datasets by reducing the overfitting problem that can occur when minority class data is simply replicated. However, SMOTE also has limitations. It may generate unrealistic samples, particularly in high-dimensional spaces, or create samples that intrude into the decision boundary of the majority class [17].

Chapter 3

Data Overview and Methods

Summary

This chapter offers a detailed overview of the data utilized in this study. It begins by presenting a summary of the entire churn process and then shifts the focus to the data.

In particular, the various data sources will be explored in detail, including the Login Table, the Watching Table, the Content Table, the Subscription Table, the App Execution Table and finally the Purchase Table. Each data source is individually explained, providing insights into the nature of the data it offers and its relevance to the study.

Furthermore, this chapter details the necessary data cleaning procedures undertaken to prepare the data for analysis. It explains how missing values were addressed and describes any other modifications or transformations performed to ensure the data is in an appropriate format for the subsequent modelling processes.

In addition, the final Dataset (CAR) obtained from the manipulation of all the tables is presented.

All the six table have been processed and stored in the integrate environment of SQL Server Management Studio (SSMS), while the final dataset CAR has been uploaded to the Microsoft cloud platform Azure Machine Learning Studio.

3.1 Approach Explanation

In this paragraph, each step, which can be seen schematically in the flowchart below, used to develop the churn model will be explained in detail.

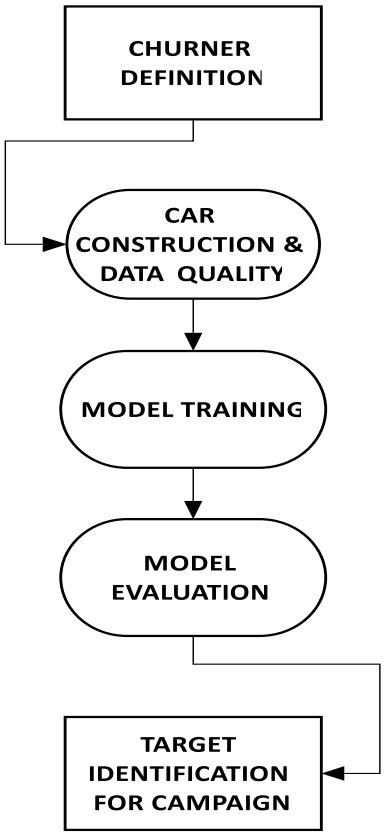

Figure 3.1: Process steps

First, the time frame in which customer behaviours were analysed to form our database is from April 2023 to December 2023, hence nine months. In the first step, the parameters to define who is considered a churner are clarified: a customer who has not logged into the OTT platform in the last

three months (a time interval chosen by the company) or a customer who has cancelled their subscription to the OTT platform. It is important to emphasize that the subscription structure is very complex, as the OTT has various partnerships with several subscription streaming services that offer a wide range of on-demand content, including movies, TV shows, documentaries, and live events, such as DAZN, Netflix, Disney+, and Amazon Prime Video. Therefore, a customer could have a subscription to the OTT but at the same time be subscribed to the other four partners mentioned earlier. If the customer decides to cancel with all four partners but keeps the subscription with the OTT active, they would not be considered a churner.

Subsequently, in the second step, the Customer Analytics Record (CAR) is defined, which represents the actual dataset. The CAR takes data from a wide range of sources, which will be explained in detail later, combining them to have the broadest possible database. The CAR was built by observing customers in the past for whom behaviour is known, and customers were divided into churners or non-churners (following the definition provided in the previous step). The CAR is a table that consolidates all data at the customer level, thus making all data available for modelling and analysis purposes. It is designed to have a single row per customer with all related measures.

In the third and fourth steps, respectively model training and evaluation, various models and configurations of models are tested on the cloud platform Machine Learning Studio. The predictive classification model is trained to recognize and isolate the distinctive characteristics of the two subsets. The result of the training is a function capable of assigning each user a measure of the probability that they will churn or remain loyal. Subsequently, the most performant model in predicting the likelihood of user churn is chosen. This choice is guided by different metrics like AUC, F1-Score, Recall and Precision. From the most performant results, it is possible to compare the different resulting measures to identify the "Best performer" model.

In the last step, target identification for campaign, the "best performer" model is applied to customers for whom behaviour is unknown, and a score (probability of churn/loyalty) is produced for each. Users at high risk of churn are identified and can be subject to specific campaigns or other targeted customer actions.

3.2 Input Data

The following sections offer detailed information about the different sources utilized to construct the CAR database. To obtain the final dataset, six tables were joined together. This information is essential for understanding the dataset's relevance and usefulness in analysing costumer behaviour in the context of churn prediction.
Below, you can see a summary table of the initial data.

Table Names	Number of Rows	Number of Columns
Login	301.488.075	33
Watching	123.452.508	66
Content	76.913	2
Subscription	4.048.695	42
App Execution	71.419.620	35
Purchase	26.942	54

Table 3.1: Input data tables with related number of rows and columns

3.2.1 Login Table

The 'login' table contains information related to all logins to the OTT within the period from April 2023 to December 2023, totalling 9 months. It is important to clarify the interpretation of these logins. 'Login' does not only

refer to the initial access where credentials are entered to log into the OTT, but also to the opening of the OTT on one's device. This conclusion was reached because the presence of multiple daily logins associated with individual customers made it unlikely that each login involved credential entry every time.

The table originally contained 301,488,075 rows and 33 variables, each representing specific aspects of customer login behaviour and device usage over a specified period. Some of the variables included in the dataset are 'Year_Month' , indicates the specific month and year, 'Num_Login' the total number of logins, providing a measure of user engagement (a decline in login frequency can be an early indicator of churn) and 'Login_No_Weekend', 'Login_Weekend' (differentiates between weekday and weekend logins, which can reveal differences in user behaviour that might indicate dissatisfaction or growing disinterest).

These variables collectively enable a comprehensive analysis of customer login behaviour, device preferences, and usage patterns over time, all of which are fundamental for churn prediction.

3.2.2 Content Table

Content" is a table that associates the ID of a content with the description of said content. For example, Netflix (a partner app) is coded as 0; the "Content" table associates ID 0 with the description Netflix.

It originally contained 76.913 rows and 2 columns, respectively ID and Description.

3.2.3 Subscription

In the Subscription table there are information about activation and cancellation of the various services present on the OTT.

The table originally contained 4.048.695 rows and 42 columns. Some columns of interest to our analysis are:

Columns (‘ID_SUBSCRIPTION_START_DATE’, ‘ID_SUBSCRIPTION_ACTIVATION_DATE’, ‘ID_ SUBSCRIPTION_ END_ DATE’) indicating start, activation and end date of subscription. Then there are also columns with information about the first and the last login and the first and the last fruition of the service (‘ID_SUBSCRIPTION_FIRST_USER_LOGIN’, ID_SUBSCRIPTION_LAST_USER_LOGIN’, ID_SUBSCRIPTION_FIRST_USER_FRUITION’, ID_SUBSCRIPTION_LAST_USER_FRUITION’).

3.2.4 Watching Table

The "watching" table contains information related to the duration of viewing, indicating how long the customer remains active on the platform. It also includes information about the channel being watched on the OTT. It is important to specify that the "watching" table provides viewing times and channel data for both OTT and third-party apps. As explained earlier, the OTT not only has its own channels but also acts as an intermediary for the channels of apps with which our company has established partnerships. Therefore, in this table, we do not have specific indications of which partner app the customer is using, only whether they are on an OTT channel or a third-party app.

The table originally contained 123.452.508 rows and 66 variables. Some of the variables included in the dataset are: ‘num_time_band’, ‘h00_06’, ‘h06_12’, ‘h12_15’, ‘h15_18’, ‘h18_21’, ‘h21_24’, ‘morning_slot’, ‘afternoon_slot’, ‘evening_slot’; indicating the distribution of views by time band. Other variables are: ‘Documentaries’, ‘Junior’, ‘Live’, ‘Music’, ‘TV_Programmes’ and ‘Series’; indicating the distribution of views by content genre, useful for understanding user preferences and how changes might predict churn.

3.2.5 App Execution Table

It is a table that indicates how many launches of a specific app have been executed. It originally contained 71.419.620 rows and 35 columns. As previously explained, the 'watching' table does not specify which third-party apps are used. The 'App Execution Table' is useful because it allows us to understand how many launches of a particular app have been made. For example, if in a month the DAZN app is opened thirty times while Disney+ only twice, by comparing these data with the 'watching' table, it is possible to understand the proportions of service usage.

3.2.6 Purchase

Purchase is a table containing information about different financial transactions. These transactions can be of various types since the different apps offer their customers the possibility to purchase or rent specific content on the OTT platform.

The purchase table originally contained 26.942 rows and 54 columns. Some columns of relevance are:'num_purchase' indicating the total number of costumer purchases, 'purchase_RENTAL' and 'purchase_PURCHASE' indicating the type of transaction carried out. There are also information about the type of content purchased ('purchase_Documentaries', 'purchase_Junior', 'purchase_Live', 'purchase_Music', 'purchase_TV_Programmes', 'purchase_Series') and concerning which day of the week the purchase was completed ('purchase_day_Mon', 'purchase_day_Tue', 'purchase_day_Wed', 'purchase_day_Thu', 'purchase_day_Fri', 'purchase_day_Sat', 'purchase_day_Sun'). All these variables provide different insights into customers' content purchasing habits

3.3 Data Transformation

In this paragraph, all the data cleaning and preprocessing procedures applied to the six tables to ensure data accuracy and completeness will be discussed.

In the image below, it is possible to see a schematic representation of how all the data originally present in the tables described in the previous chapters were filtered and manipulated to construct the final CAR dataset.

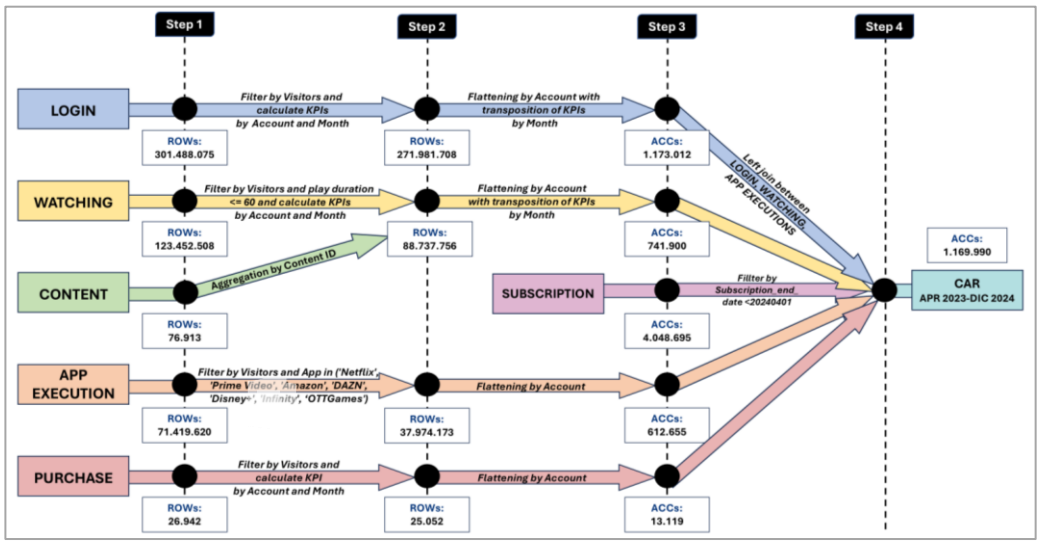

Figure 3.2: Data manipulation process

It is important to note that the tables provided by the client to Accenture were compiled in such a way as to discard, from the start, accounts considered defective (i.e., accounts with too much missing data).

Since the tables were processed using SQL Server Management Studio, they do not contain duplicates. This is because the variable ID_USER_ACCOUNT, present in almost all tables (except the Content table), was used as the primary key, meaning, a duplicate primary key is not possible.

As shown in figure 3.2, each table underwent manipulations to construct the final CAR (Customer Analytics Record). Before explaining the operations performed on each table, let's define what is meant by

flattening and transposition. Flattening refers to converting a complex data structure into a simpler one. In our context, this means taking the original data aggregated for each account and creating a tabular structure where each row represents an account, and each column represents a specific metric for that account. Transposition of KPIs by month refers to organizing the data in such a way that, for each account, the KPIs are broken down by month.

Starting with the Login Table, as shown in the diagram, a filter is applied for the variable visitors, and the KPIs (Key Performance Indicators) for each account and month are calculated, such as monthly, quarterly, and weekday averages. This is done to obtain enough informative data to understand user behaviour. For example, during the time slot from 2:00 PM to 5:00 PM, you know it's either a child returning from school, a housewife, or a retiree. This step was taken because a second part of the project assigned to Accenture involved customer clustering, which, however, will not be covered in this thesis. A group by operation is then performed for each account to obtain the main KPIs.

In the Watching Table, all views with a duration of less than 60 seconds are discarded. This choice is guided by the following consideration: if the viewing time is less than 60 seconds, the customer is merely scrolling through channels without genuine interest in watching that specific channel. For this table as well, watching averages are calculated monthly and associated with the account, followed by a group by operation for each account to obtain the main KPIs. The Content Table is aggregated, as shown in the graph, with the Watching Table in the first step.

In the App Execution Table, a filter for visitors to apps is applied. The aim of this step is to select a reduced number of all available applications because, in the original table, a single app was named in many different ways (for example, Prime Video and Amazon, which are two separate elements in the table, are grouped into the same category). Finally, a group by operation is performed for each individual account.

In the Purchase Table, like the previous ones, a filter for visitors is applied, and the KPIs are calculated for each account and month. A group by operation is then performed for each account.
In the Subscription Table, the filter 'subscription_end_date < 20240401' is applied. This filter excludes from the analysis all accounts whose subscription ends after April 1, 2024. This means that the analysis focuses only on accounts that could potentially cancel the service within the period of interest, namely between April 2023 and December 2023.
In the final step, all tables are combined into a single dataset, which is the CAR (Customer Analytics Record). Specifically, as seen in figure 3.2, the Login, Watching, and App Execution Tables are joined using a left join that retains all the rows from the first table (account) and adds the corresponding rows from the other two tables based on a common identifier, which is the primary key 'account'.

3.4 Data Migration and Cloud Overview

The CAR dataset was subsequently migrated to Machine Learning Studio. On Azure, all the operations described from this point onward were performed. In the figure 3.3 is showed the query needed to import the dataset.

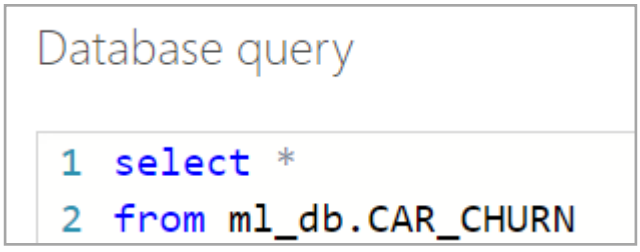

Figure 3.3: Import Query

Instead, the figure 3.4 shows how the operation looks in the new Azure environment.

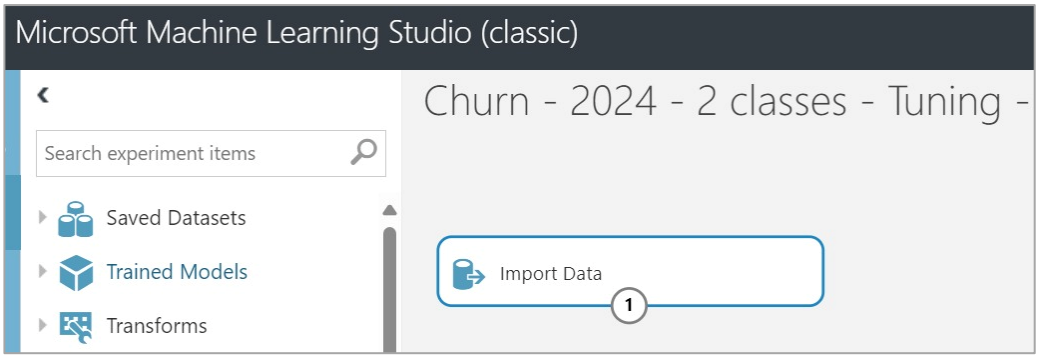

Figure 3.4: Azure Dashboard

Azure Machine Learning Studio is a cloud-based platform developed and officially launched by Microsoft in June 2014. It is a public cloud service provided by Microsoft Azure that works as a computing service over the public internet, making resources such as servers and application available to everyone that want to use them.

As stated in [19] Cloud computing is a rapidly growing technology in the field of business innovation. Thanks to its flexibility and adaptability, it enables new approaches to working, operating, and managing business activities. Platforms like Azure allow users to access files and applications stored in the cloud from any location, eliminating the need to be physically close to the hardware. Cloud computing has proven to be advantageous for both consumers and businesses.

The implementation of machine learning algorithms in the cloud presents numerous opportunities for more efficient resource management. Cloud-based machine learning environments offer preconfigured computer clusters with statistical software, freeing customers from the need to install and manage their own clusters [20]

Specifically, as stated in [21] some advantages of using cloud platform for machine learning tasks are:

- Cloud platforms provide unmatched scalability, enabling businesses to adjust their computing resources up or down effortlessly according to demand. This adaptability is especially advantageous for machine learning tasks that require different levels of computational power and storage capacity.

- Machine learning platforms hosted in the cloud remove the necessity for costly on-premises hardware and software setups. This allows businesses to utilize cutting-edge machine learning tools and technologies without incurring the substantial upfront expenses related to purchasing and maintaining physical infrastructure [22].
- The cloud platform's role is to manage applications through quality control mechanisms, enhancing the collaboration capabilities among multiple groups involved [23].
- Cloud service providers are making significant investments in artificial intelligence and machine learning consistently enhance their platforms with cutting-edge machine learning technologies, including advancements in deep learning, neural networks, and natural language processing. This approach allows businesses to utilize the latest innovations without the necessity of substantial investments in new hardware or software [21].

  There are not only advantages in using cloud platform. Some disadvantages of using cloud platform like Azure Machine Learning Studio for machine learning tasks are:

- A stable and dependable internet connection is essential for machine learning systems that operate in the cloud. Inadequate internet access can interrupt data transmission, resulting in inaccurate outcomes and diminished efficiency. This reliance on internet connectivity can be a major disadvantage in regions where internet connections are unreliable or slow [24]
- One significant drawback of cloud computing is the latency. When data is sent to the cloud for processing and then returned to the client, there can be a noticeable delay. This latency can negatively impact the performance and responsiveness of machine learning models [25].
- Storing data in the cloud requires sharing sensitive information with third-party providers, which raises concerns about security and privacy. Despite the implementation of stringent security protocols by

cloud providers, data breaches and unauthorized access can still happen, putting sensitive and proprietary information at risk [26].

Overall, cloud computing is expected to remain a key player in the future of IT, helping organizations become more agile, efficient, and innovative amidst rapid technological advancements. This trend will likely propel further innovations in AI and machine learning in the years ahead [19].

3.5 CAR Dataset

After manipulating all six tables and merging them, we obtain the final CAR dataset containing the necessary data to profile an individual's behaviours and somehow teach the model which behavioural trends anticipate a possible cancellation of the OTT service.

The following table shows the dataset in its entirety. The CAR consists of 63 features and 1.169.990 observations. It is important clarify that there are some variables whose name has been changed as they contained the original name of the client company and the OTT platform, and as explained at the beginning of this thesis, for legal reasons it is not possible to cite them. The modified variables are marked with two asterisks (**).

Variable Names	Variable Names
ID_USER	sum_NUM_APP_EXECUTIONS
min_year_month	avg_NUM_APP_EXECUTIONS
max_year_month	sum_app_Netflix
count_months_login	sum_app_Dazn
count_months_watching	sum_app_Disney
count_months_purchase	sum_app_Prime_video
count_months_app	sum_app_OTTgames **
sum_login_Weekend	sum_app_Other_Partner
sum_login_no_Weekend	avg_app_Netflix
sum_num_login	avg_app_Dazn
sum_num_view	avg_app_Disney
sum_playback	avg_app_Prime_video
avg_login_Weekend	avg_app_OTTgames **
avg_login_no_Weekend	avg_app_Other_Partner
avg_num_login	flag_Offer_OTTvision **

avg_num_view	flag_Partnership_Amazon_Prime
avg_playback	flag_Partnership_Dazn
avg_flag_fruition	flag_Partnership_Disney
avg_ratio_fruition_login	flag_Partnership_Other_Partner
max_num_deviced	flag_Partnership_Netflix
sum_num_dist_view	flag_Offer_type_Mobile
sum_num_view_linear	flag_Offer_type_OLO
sum_num_view_vod	flag_Offer_type_Residential
sum_num_purchase	sum_Category_Other
sum_sum_purchase_price	sum_login_dev_2ndSCREEN
avg_num_purchase	sum_login_dev_CTV
avg_sum_purchase_price	sum_login_dev_STB
sum_Cinema	sum_login_dev_WEB
sum_Documentaries	sum_login_dev_OTHER
sum_Junior	Target
sum_Music	
sum_TV_Programmes	
sum_Series	

Table 3.2: CAR dataset features

The dependent variable, the one the model tries to predict, is the last feature of table 3.2. The variable 'Target' is a multi-class categorical variable (three classes) with integer data.

The initial idea was to build a highly specific model. By highly specific, I mean a model capable of predicting not only if an individual was likely to churn or remain loyal but also to predict if the churn would occur only on specific partner apps. Remember that the OTT platform is a portal that, in addition to having its own channels, also has channels from third-party partner apps.

Initially, to also monitor the dynamics of the partners, three classes were identified in the dependent variable 'Target': 0 for those who do not cancel anything, 1 for those who cancel some subscriptions but remain loyal to the OTT, and 2 for those who cancel the OTT regardless of what the customer does with other subscriptions.

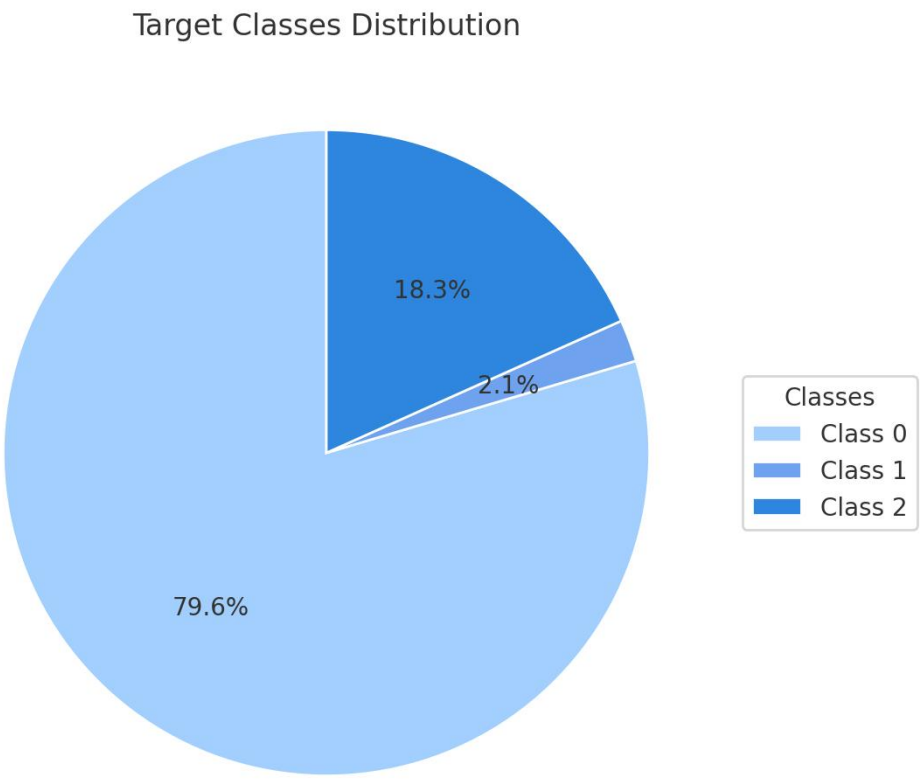

Figure 3.5: Target Classed Distribution Pie Chart

However, this approach, although very interesting, was not feasible because classes 1 and 2 were too small (only 25002 observations for class 1 as we can see from figure 3.5). Therefore, the two classes 0 and 1 were aggregated, resulting in a binary categorical variable where class 0 represents loyal customers and class 1 represents churners. The new dependent variable is called 'TARGET_BINARY'.

In the figure below, the approach just explained is shown in the Azure Machine Leaning environment.

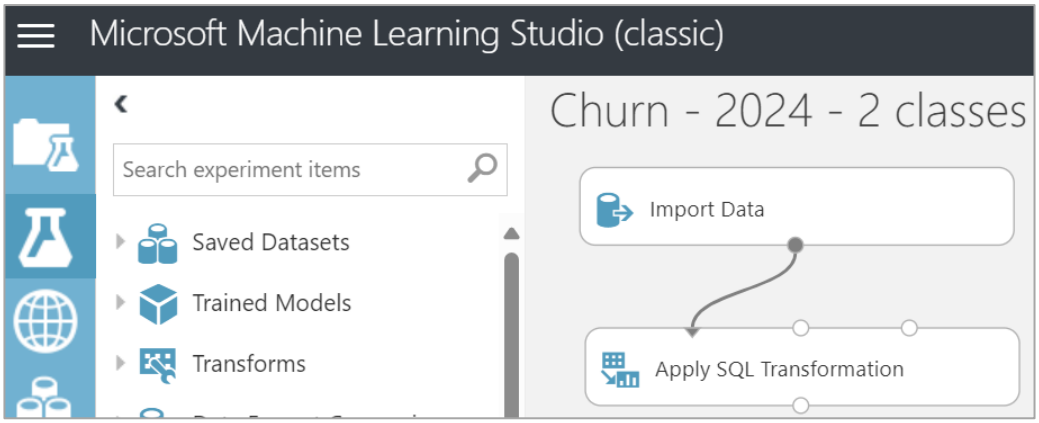

Figure 3.6: SQL Transformation Input Data

Meanwhile, in figure 3.7, you can observe the executed query.

SQL Query Script

```
1 select *,
2 CASE WHEN TARGET = 0 OR TARGET = 1 THEN 0
3      WHEN TARGET = 2 THEN 1 END AS TARGET_BINARY
4 from t1;
```

Figure 3.7: SQL Query from Three Class to Two

3.5.1 Data Cleaning

In this paragraph, the cleaning and exploration operations of the dataset are discussed. A univariate analysis was performed for each feature in the dataset. Univariate analysis involves evaluating each variable individually by observing its statistics. Thanks to the 'Summarize Data' module in Azure, it is possible to observe the main statistics of the variables, such as Unique Value Count, Minimum and Maximum Value, Mean, Standard Deviation, and many others. Strategies were then defined for each variable: 1) Remove the variable from the dataset if it is not useful or significant for the analysis; 2) Keep the variable in the dataset for subsequent analysis; 3) Apply preprocessing techniques to improve the quality of the variable (e.g., handling missing values, etc.). It was decided

to exclude variables that presented a single value (0) for each user. This is because such variables do not provide useful information for the analysis, as they are constant and thus lack variability.

Subsequently, a multivariate analysis was conducted, which considers the relationships between multiple variables simultaneously, using the correlation matrix. The correlation matrix allows us to understand how correlated the different measures are with each other. From the correlation matrix, it emerged that certain variables are strongly correlated with each other. For the time being, the variables are not excluded from the dataset, but this issue will be addressed in the next chapter dedicated to models and, specifically, to data preprocessing. Additionally, thanks to the statistical summary of the variables obtained through the 'Summarize Data' module, it was also possible to identify potential outliers. If the maximum or minimum value is significantly distant from the mean (e.g., more than three times the standard deviation), this may indicate the presence of outliers. However, the extreme values detected were considered consistent with the distributions, so no actions were taken to remove them.

The operations translated into the Azure environment can be seen in the figure below.

Figure 3.8: Linear Correlation and Principal Statistics

Then, as can be seen from Figure 3.9, using the 'Select Columns in Dataset' block, we exclude certain features.

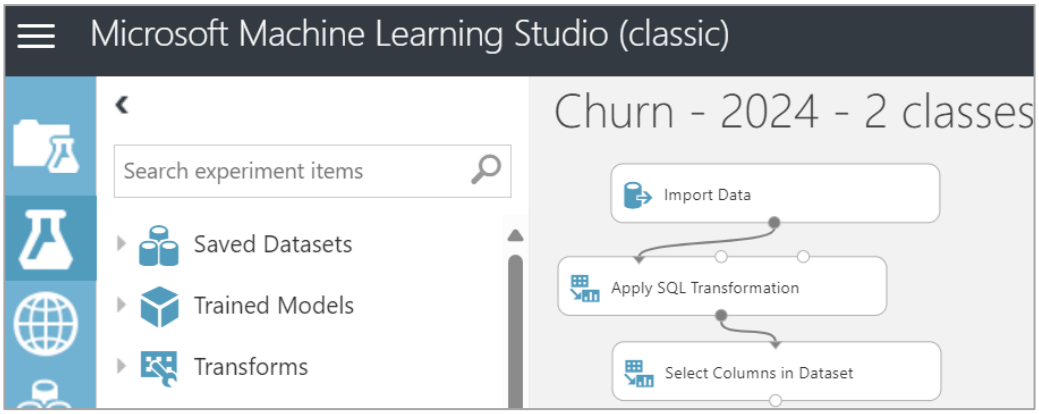

Figure 3.9: Columns Dropped Operations

Six columns are excluded: 'ID_USER', 'min_year_month', 'max_year_month', 'sum_sum_purchase_price', 'sum_login_dev_OTHER' and 'Target'.

Starting with the last one, it is simply excluded because it was the original dependent variable with values 0, 1, and 2, and has been replaced with 'TARGET_BINARY'.

The variable 'sum_sum_purchase_price' was dropped since it contained all null values and the same goes for 'sum_login_dev_OTHER'.

The variables 'min_year_month' and 'max_year_month' (indicating the first and last month of access to the OTT during the analysis period) were excluded because we are not interested in temporal correlation, such as identifying periods with a higher number of churners. Instead, we aim to investigate other types of causes.

The last variable, 'ID_USER', was not included in the analysis as we do not want to associate the churner with a number that only represents the customer's ID. The dataset now consists of 58 features.

3.5.2 Missing Values Handling

In this paragraph, the process of identifying and handling missing values will be discussed. A total of 32 columns containing missing values were identified. As shown in figure 3.13, the missing value cleaning operation in the Azure tool involves the addition of a specific block.

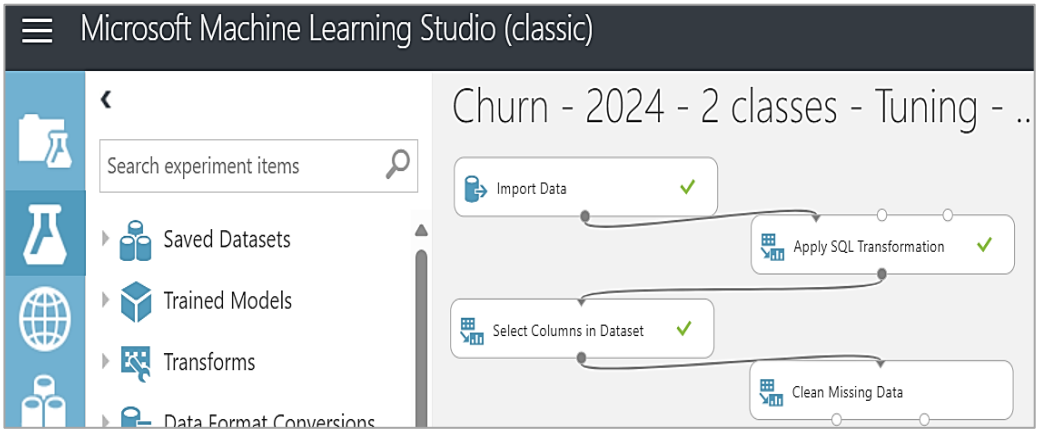

Figure 3.10: Cleaning Missing Data Operation

In table 3.3, we can observe in detail which columns contain missing values and how many missing values there are for each column.

VARIABLE NAMES	MISSING VALUES EACH COLUMNS
sum_playback; sum_num_view; avg_num_view; avg_playback; avg_rapporto_fruition_login; sum_num_dist_view; sum_num_view_linear; sum_num_view_vod; sum_Cinema; sum_Documentaries; sum_Junior; sum_Music; sum_ TV_Programmes; sum_Series; sum_Category_Other;	**431287**
sum_num_purchase; avg_num_purchase; avg_sum_purchase_price;	**1156930**

sum_NUM_APP_EXECUTIONS; avg_NUM_APP_EXECUTIONS; sum_app_Netflix; sum_app_Dazn; sum_app_Disney; sum_app_Prime_video; sum_app_OTTgames **; sum_app_Other_Partner; avg_app_Netflix; avg_app_Dazn; avg_app_Disney; avg_app_Prime_video; avg_app_OTTgames **; avg_app_Other_Partner;	**558885**

Table 3.3: Name and Number of Columns with Missing Values

Missing values can be handled using various techniques, such as imputation (replacing missing data with the mean, median, or mode), deletion (removing rows or columns with missing values), or by using algorithms that are robust to missing data. The choice of method depends on the nature of the data and the impact of missing values on the analysis or model performance.

Missing data can arise from various sources, such as data entry mistakes, non-responses in surveys, or incomplete data collection processes. The process of imputation involves estimating and filling in the missing values using the available data, which helps in ensuring more comprehensive and accurate analyses.

In this sense Azure Machine Learning provides the ability to set parameters for data cleaning. Specifically, there are:

- **The 'Minimum missing value percentage'** which specifies the minimum percentage of missing values in a column for it to be considered for cleaning. The value is set to 0, meaning that all columns, even those with just one missing value, will be considered.

- **The 'Maximum missing value percentage'** specifies the maximum percentage of missing values in a column that can be considered for cleaning. The value is set to 1, meaning that even columns with all values missing will be considered.
- **The 'Cleaning mode'** determines how the missing values will be handled. The parameter is set to "Custom substitution value," which means that the missing values will be replaced with a value specified by the user.
- **The 'Replacement value'** indicates the value that will be used to replace the missing values when the "Cleaning mode" is set to "Custom substitution value." The replacement value is 0. This decision was guided by the simple fact that the CAR was built by initially eliminating variables containing many missing values.

3.5.3 Data Exploration

In this last paragraph is now interesting to better observe the behaviour of some features to try to gain insights from the data contained in the CAR.
In Figure 3.10, we can see the distribution of the variable 'count_months_login'. This variable contains information related to the total number of months in which each user logged in during the analysis period of 9 months.

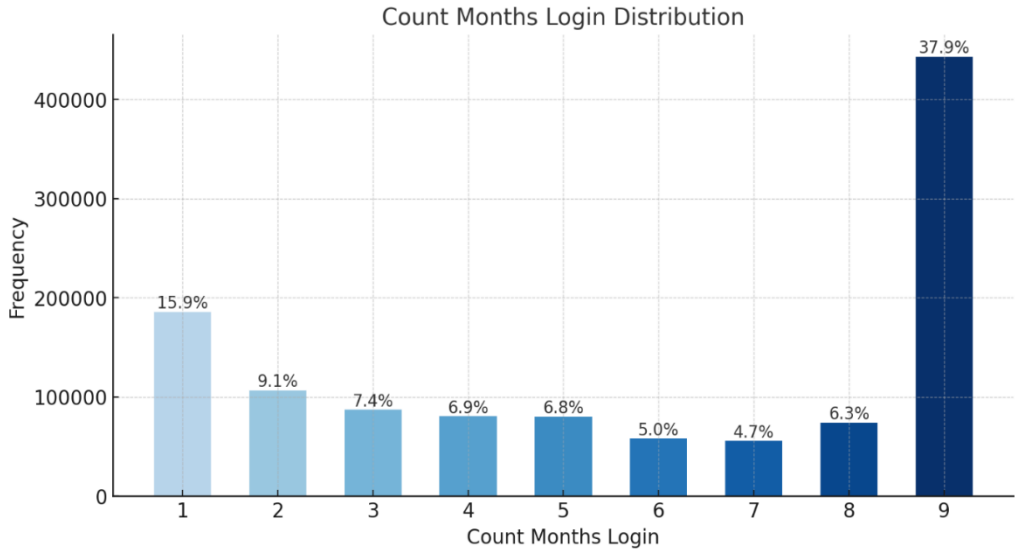

Figure 3.11: Mont Login Distribution

This chart reveals a clear situation, namely that about 38% (443,074) of subscribers logged in at least once each month. This percentage represents the potential core customer base (those who show the greatest loyalty). We can observe a uniform distribution for the other months around 8%. We notice a peak of customers, amounting to 16% of the total, who logged in only 1 month. Users who logged in fewer months (1-2 months) might be at risk of churn and, therefore, are the ones to whom retention strategies should be applied.

In Figure 3.9, we can see the distribution of different modes of accessing the OTT services. It is the analysis of the following variables: sum_login_dev_2ndSCREEN, sum_login_dev_CTV, sum_login_dev_STB, sum_login_dev_WEB. The acronyms related to the types of login devices represent the following categories: 2ndSCREEN for secondary devices such as tablets or smartphones used in combination with another primary device; CTV (Connected TV) for Internet-connected televisions that can access online content; STB (Set-Top Box) for devices connected to a television to receive and decode digital television signals; and WEB for access through a web browser.

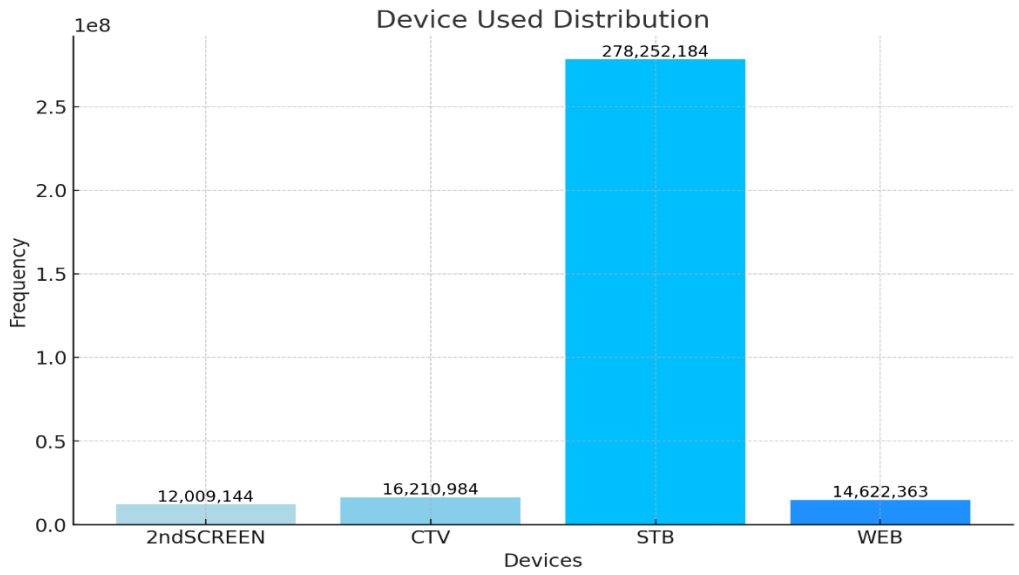

Figure 3.12: Device Used Distribution

It is evident from the histogram that users accessing the OTT service via STB are the majority. STB users are likely stable and loyal users, as these devices are generally associated with long-term contracts with the operator. The use of secondary devices is less frequent compared to other devices. Users who primarily access the service through 2ndSCREEN may be more mobile and less inclined to prolonged viewing sessions. Users using connected TVs represent a significant, but not dominant, part of the user base. This group may include families or users who prefer viewing on larger screens. Finally, users accessing via the web may be younger and more tech-savvy, preferring the flexibility of browser access. Therefore, STB users are the least likely to be at risk of churn, whereas there is a higher risk for users of other devices.

From the figure below, by analysing the variables related to the content consumed by customers, it is possible to better understand their viewing preferences. To improve the visualization and make the representation more uniform, a logarithmic scale was used to compress the differences between the column lengths.

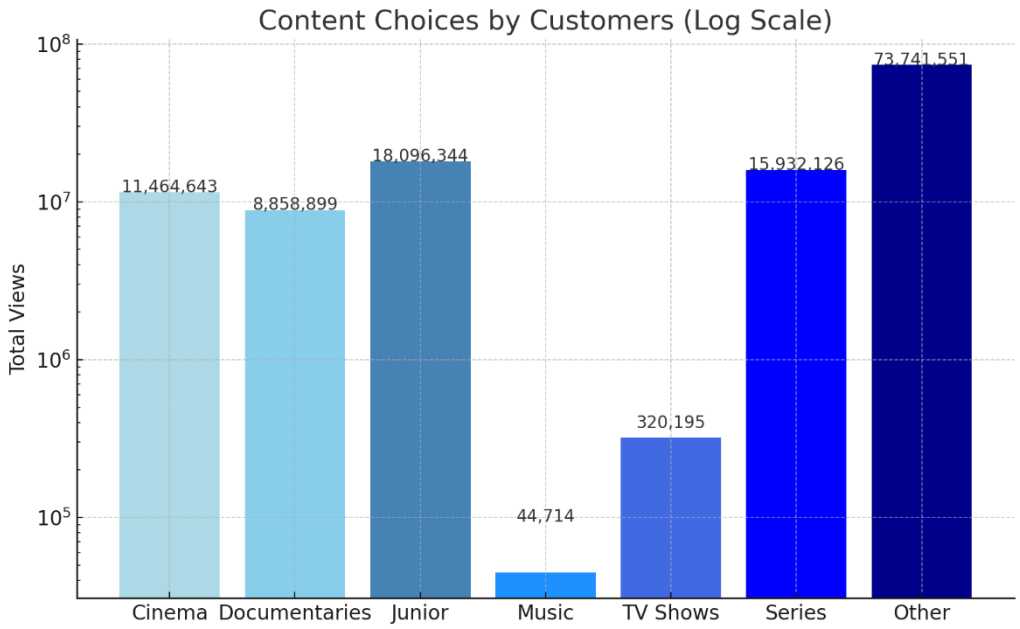

Figure 3.13: Content Costumer Distribution

The chart provides an overview of customer preferences for different content categories, offering useful insights into behaviours that could influence service abandonment. The "Other" category shows a significantly higher number of views compared to the others, suggesting that a large number of customers may primarily use the service for content that is not easily categorized or is niche. The Series and Junior categories exhibit high activity, indicating more engaged customers who are less likely to churn and could benefit from targeted retention strategies. In contrast, the TV Shows and Music categories show fewer views, which may indicate a higher churn risk among these users, highlighting the need to expand the content offering.

The last interesting distribution to watch is the different number of app launch.

Figure 3.14: App Login Distribution

As seen in Figure 3.12, the two main services used are Netflix and DAZN. This high engagement is a positive sign for retention, translating to a lower risk of churn. The OTT Games app shows minimal engagement. This indicates a high risk of churn for this application, as the low engagement may reflect limited interest.

Chapter 4

Preprocessing and Model Overview

Summary

Choosing the right machine learning algorithms is essential for developing accurate predictive models. In this chapter, different commonly used algorithms are evaluated for their effectiveness in forecasting customer churn.

In the following paragraphs, before analysing each model used, a summary will be provided on how the training data was conceived.

Finally, all the algorithms used will be described in detail, and the results of each algorithm will be presented to determine which one performs best. Four models will be described in order: Two-Class Logistic Regression, Two-Class Neural Network, Two-Class Decision Forest, and Two-Class Boosted Decision Tree.

4.1 Data Processing Steps for Churn Analysis

As mentioned in the previous chapter, the data preparation for training the model follows this procedure:

- **Temporal Section:** User data from the last 9 months prior to the analysis date (April 2023 – December 2023) is selected.
- **Churner Definition:**
    1. Users who have terminated their OTT subscription at any point during the observation period are considered churners.
    2. Users who have shown no activity during the last 3 months of the 9-month observation period are also considered churners.

The machine learning models are trained to detect two main indicators of user behaviour:

- **Subscription Cancellation**: Identifies users who cancelled their subscription during the observation period by learning the behavioural patterns leading up to this event.
- **Inactivity**: Recognizes users who have shown no activity for a significant period at the end of the observation period, interpreting this as a potential implicit abandonment of the service.

This process allows the model to adapt and predict two types of churn behaviour: Long-term users who may cancel their subscription after an extended period; and more recent users who subscribe and cancel within a short time frame. The goal is to create a flexible, business-oriented model capable of predicting churn behaviour with sufficient advance notice. This long-term approach ensures that the client company has enough time to develop effective and targeted campaigns aimed at retaining customers

4.2 Data Preprocessing: Correlation Matrix Considerations

As mentioned in section 3.5.1, after calculating the Correlation Matrix (visible in Figure 4.3), the presence of several highly correlated variables was observed. This observation leads us to consider the issue of multicollinearity, a statistical concept that occurs when two or more independent variables in a regression model are strongly correlated with

each other. This can cause various problems for the model, making it difficult to determine the individual effect of each independent variable on the target variable. High multicollinearity can lead to unstable regression coefficients, wide confidence intervals, and can reduce the precision of the estimates [27].

Variable names have not been entered for privacy reasons.

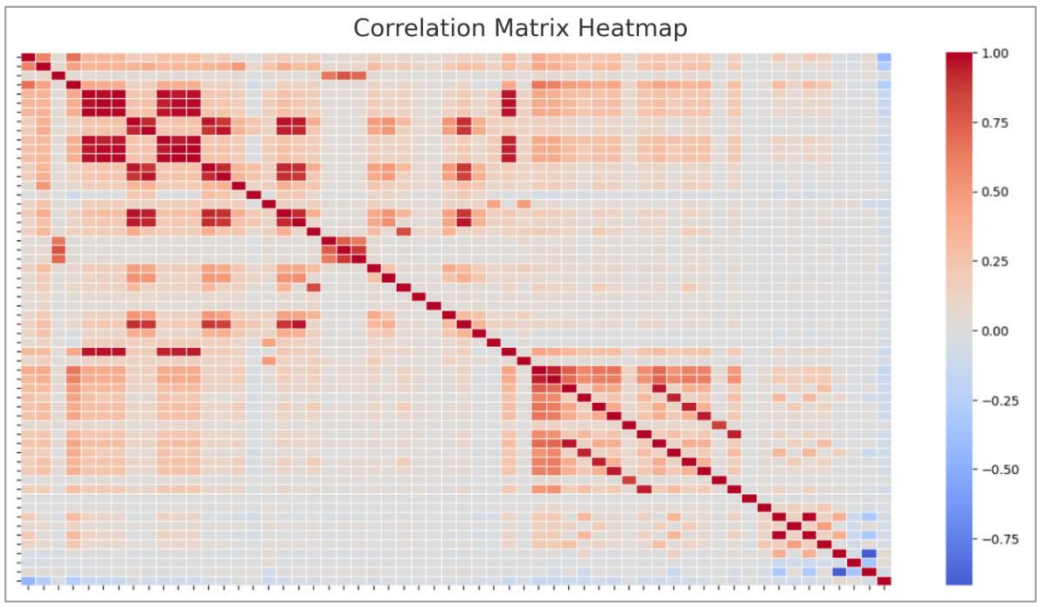

Figure 4.1: Correlation Matrix

Several techniques were tried to address the high correlation between some variables in the CAR. The first approach taken was PCA (Principal Component Analysis).

PCA is a mathematical method that converts a set of potentially correlated variables into a smaller set of uncorrelated variables, known as principal components. The first principal component captures the maximum possible variance in the data, while each subsequent component captures the maximum variance remaining [28].

The general idea was to apply PCA (Principal Component Analysis) to only two algorithms (Logistic Regression and Neural Network) and to keep the original variables for the other two (Decision Forest and Boosted Decision Tree). This approach was designed to leverage the specific strengths of each model.

As stated in [29], in the case of Logistic Regression and Neural Networks, PCA is useful because it reduces the dimensionality of the dataset, eliminating multicollinearity, which can negatively affect linear models like logistic regression. Moreover, in neural networks, dimensionality reduction can decrease the number of parameters to optimize, reducing training time and mitigating the risk of overfitting, especially in smaller networks or with limited datasets.

On the other hand, nonlinear models such as Decision Forest and Boosted Decision Tree are designed to handle correlated variables without necessarily requiring dimensionality reduction. These algorithms can leverage the full range of available features, including correlated ones, to enhance the robustness and stability of the model. Additionally, PCA might remove useful information that these models could exploit due to their ability to capture nonlinear interactions between variables [30].

On Azure, after creating a subset containing only the independent variables, the PCA (Principal Component Analysis) block was applied, setting the "Number of dimensions to reduce to" parameter initially to 10. This parameter specifies the number of principal components you want to retain after applying PCA. As a result, the PCA transforms the original dataset into a new set composed of ten variables, called principal components.

The next step was to check the variance explained by the 10 principal components using the following filter:

```
explained_variance = pca.explained_variance_ratio_
print("Variance explained for each principal component:")
for i, variance in enumerate(explained_variance):
    print(variance)
```

Figure 4.2: Code to calculate PCA variance

To calculate the variance explained by each principal component, I conducted an analysis using a local development environment, as Azure

does not offer the capability to perform this specific analysis. The results of this analysis indicated that the first principal component captures most of the information contained in the original data, while the last few components explain little variance.

So, I tried increasing the number of principal components first to 15 and then to 30 to test if any changes occurred with the increase in variables. The result was consistent with the initial attempts, with the total variance being explained almost exclusively by the first principal component, PC1. Specifically, PC1 explained 98.51% of the total variance, while the remaining components contributed only marginally, cumulatively explaining less than 1.5%. This result suggests that nearly all the information in the original dataset is captured by a single direction of variation, making PCA ineffective for distributing information across multiple principal components. Consequently, I decided to abandon the PCA approach, as dimensionality reduction to multiple components could overly simplify the model, potentially compromising its ability to capture meaningful relationships present in the original data. Instead, I opted for a feature selection approach, which allows retaining only the most relevant variables for the predictive model. This method not only preserves crucial information but also enhances model interpretability, as the selected features can be analysed individually to understand their contribution to the decision-making process. Feature selection proved to be a more suitable strategy for improving model effectiveness, reducing noise, and focusing on variables that provide real predictive value.

4.2.1 Data Preprocessing: Filter-Based Feature Selection Module Settings

Before to apply the Feature Selection in this paragraph will be analyse the correlation between the dependent variable 'TARGET_BINARY' and the other 57 variables that make up the CAR. The goal is to identify the variables most correlated with the dependent variable and thus the most significant for our prediction. As shown in Figure 4.1, this operation is performed in the Azure environment using a specific filter

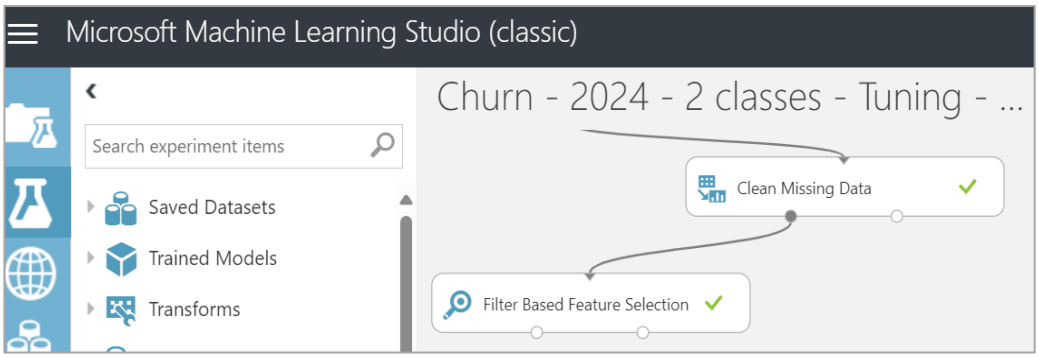

Figure 4.3: Filter Based Feature Selection

The "Filter Based Feature Selection" is a pre-model technique that is applied before model training. It uses statistical criteria to independently evaluate the importance of each variable, without considering the model construction process. This technique employs methods like mutual information, Chi-square, correlation, and others to assess the relevance of features concerning the target variable. Importantly, it does not depend on the type of model that will be trained later, as it is solely based on statistical relationships between the features and the target variable. Because of its independence from the model, it is generally faster and less computationally intensive compared to methods that require model training.

Azure Machine Learning Studio provides the ability to set different parameters for the configuration of the "Filter Based Feature Selection" module. Here there are the details of the configured parameters:

- **Feature scoring method:** This parameter specifies the method used to evaluate the importance of each feature concerning the target variable. The parameter is set to Mutual Information. Mutual information is a statistical measure that quantifies the amount of information a feature provides about the target variable. In other words, it measures the dependency between a feature and the target variable. Features with higher mutual information scores are considered more relevant for the model.
- **Target column:** Specifies the column in the dataset that represents the target variable, the one you want to predict. The target column used to calculate mutual information is TARGET_BINARY. This is the dependent variable or the variable you want to predict in the model. Feature selection will thus be based on how well the features explain or correlate with this target variable.
- **Number of desired features:** This parameter allows you to specify the number of features to select based on their scores. By setting this value, you can limit the number of features retained for the model. For example, if you set this parameter to 5, the module will select the top 5 most relevant features. This is useful for reducing the dataset's dimensionality and simplifying the model. This parameter is set to 1, meaning the module will select only the most relevant feature (the one with the highest mutual information score). This can be useful if you want a very simple model or are conducting a preliminary analysis.

So, this configuration allows you to identify the single most informative feature concerning the target variable.

In Figure 4.2, you can see the 15 most significant variables in terms of correlation with the target variable. The following figure is a spider chart that visualizes the correlations of all variables around a common centre.

Each axis represents a variable, and the distance from the centre represents the correlation value.

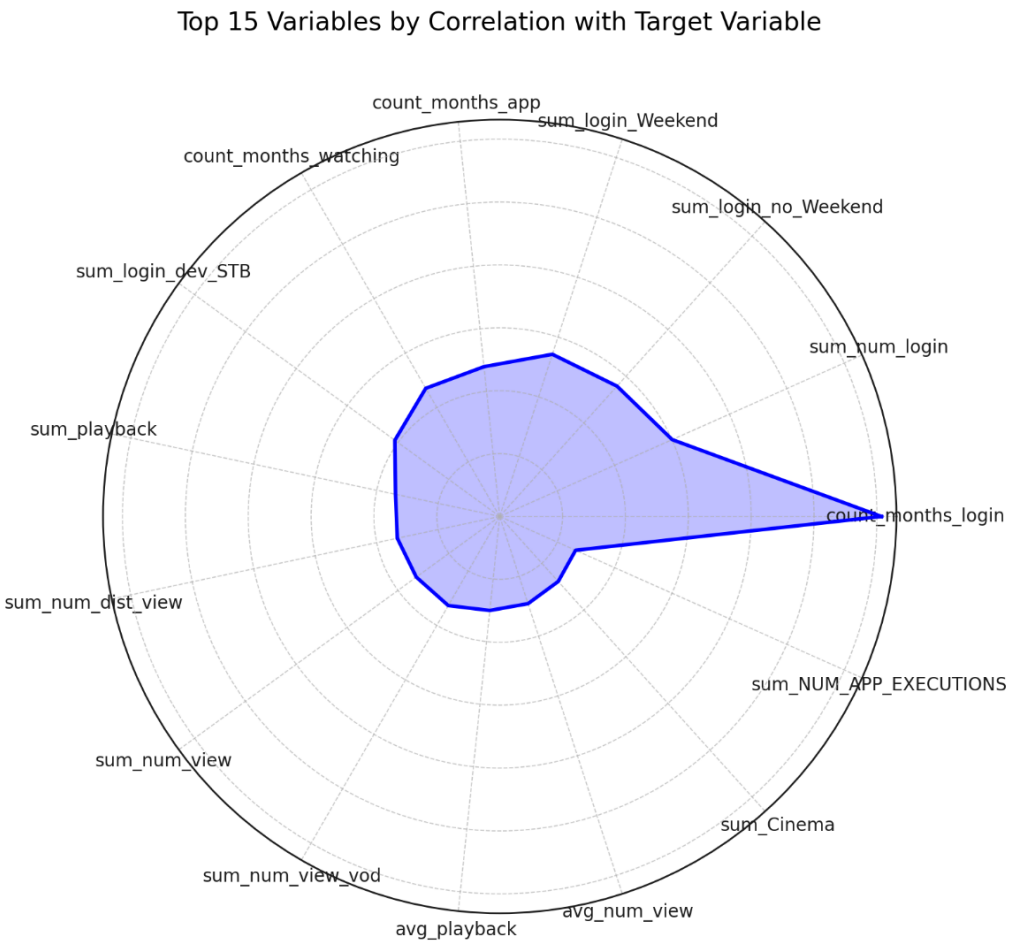

Figure 4.4: 15 Most Correlated Variables with Target Variable

As we can see from the chart the most significant variable, based on the correlation with the target variable, is "count_months_login" with a correlation index of 0.1214. However, it's important to note that the significance of a variable depends on the context and the type of model you're using. Even though this variable has the highest correlation, it might not be the most influential in a complex predictive model, where other variables or interactions between variables could play a crucial role.

The average correlation of the 15 variables most correlated with the target variable is approximately 0.0449. This value represents a rather low

correlation, suggesting that, on average, the variables have a moderate influence on the target variable. However, as mentioned earlier, there are some important considerations to keep in mind: Even if the individual correlations are low, this does not necessarily mean that the variables are irrelevant to the model. They could contribute significantly when combined in a complex model like logistic regression, decision trees, or neural networks. If non-linear models (such as Decision Forest, Gradient Boosting, etc.) are used, the relationships between the variables and the target may not be well represented by simple linear correlation. In these cases, even variables with low correlation can prove to be very important. Moreover, the success of the churn prediction model does not depend solely on correlations but on how the model performs on evaluation metrics such as ROC-AUC, Accuracy, Recall, Precision, and F1 score.

4.2.2 Data Preprocessing: Feature Selection

In this paragraph, the operations performed to remove certain variables to prepare the final dataset for use in algorithms will be discussed.

It was decided to analyse and process only the variables with a high correlation above 0.9, as the aim was to avoid losing too much information. Therefore, a filter was applied that considered only a correlation threshold of 0.9.

```
high_correlation_pairs = correlation_matrix.stack().reset_index()
high_correlation_pairs.columns = ['Variable_1', 'Variable_2', 'Correlation']
high_correlation_pairs = high_correlation_pairs[
    (high_correlation_pairs['Variable_1'] != high_correlation_pairs['Variable_2']) &
    (high_correlation_pairs['Correlation'].abs() > 0.9)
]
print(high_correlation_pairs)
```

Figure 4.5: Filter Code High Correlation Variables

In the Network graph below, it is possible to see the most relevant correlation:

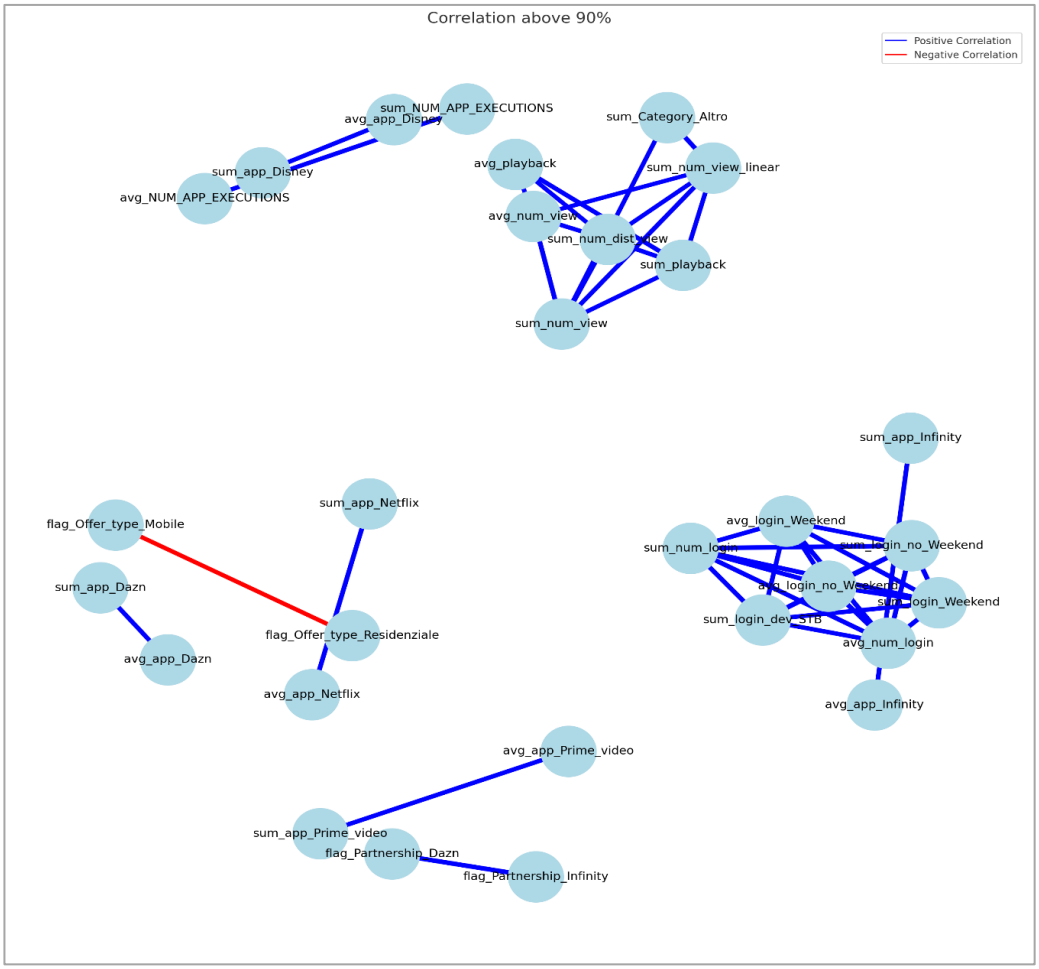

Figure 4.6: Variables Correlation above 0.9

The chart shows all the high correlations, both positive and negative. Tangled nodes can be observed. This is due to the presence of some variables that are correlated multiple times: sum_num_login, avg_num_login, sum_login_no_Weekend, sum_login_Weekend, avg_login_no_Weekend, avg_login_Weekend. The methodology for applying Feature Selection was based on combining the removal of highly correlated variables with the Filter-Based Feature Selection explained in the previous paragraph. Using this filter, based on the influence that the variables have on the dependent variable (TARGET_BINARY), allows you to retain only the features that have a significant impact on the prediction.

This is particularly useful for improving model performance and reducing the risk of overfitting. By retaining only the most influential variables, the resulting model will be easier to interpret, which can be especially important if you need to justify the model's decisions in a business context. Therefore, for pairs of highly correlated variables, the general approach will be to keep only the variable with greater importance relative to the target. Additionally, if a variable has very low importance, it will be analysed and considered for removal, even if it is not correlated with other variables.

It is important to specify, however, that it was decided not to remove the 15 most important variables in predicting the target variable (as shown in Figure 4.4), regardless of whether they exhibited high correlations, because they have a significant influence and should be retained in the model.

After reviewing the combination of the two metrics described earlier, ten variables were removed, either because they were highly correlated, less influential with respect to the target variable, or for both reasons. Table 4.1 shows which variables were removed and the reason for their removal.

VARIABLE NAME	REASON FOR REMOVAL
avg_rapporto_fruition_login	High correlation
avf_flag_fruition	Little influence
avg_login_Weekend	High correlation/ Little influence
avg_login_no_Weekend	High correlation/ Little influence
avg_NUM_APP_EXECUTIONS	High correlation
avg_app_Disney	Little influence
avg_app_Dazn	Little influence
avg_app_Netflix	Little influence
avg_app_Other_Partner	Little influence
flag_offer_type_olo	High correlation

Table 4.1: Variables Name Dropped and Reason

4.2.3 Final CAR Dataset

In the previous paragraphs, all the preprocessing techniques applied to the original dataset have been discussed.

The original CAR contained 63 features. After analysing and cleaning the dataset of any duplicates, missing values, and other values that would have hindered proper analysis, and after creating a new target variable, the final CAR that will be used to train the four models for our churn prediction consists of 48 variables.

Thus, 16 variables were removed, namely:

VARIABLE NAME
'ID_USER'
'min_year_month'
'max_year_month'
'sum_sum_purchase_price'
'sum_login_dev_OTHER'
'Target'
'avg_ratio_fruition _login'
'avf_flag_fruition'
'avg_login_Weekend'
'avg_login_no_Weekend'
'avg_NUM_APP_EXECUTIONS'
'avg_app_Disney'
'avg_app_Dazn'
'avg_app_Netflix'
'avg_app_Other_Partner'
'flag_offer_type_olo'

Table 4.2: Variables Name Dropped from Original Dataset

In Figure 4.7, all the variables are shown in relation to their impact on predicting the dependent variable "TARGET_BINARY." Naturally, the dependent variable will have a maximum value in the chart.

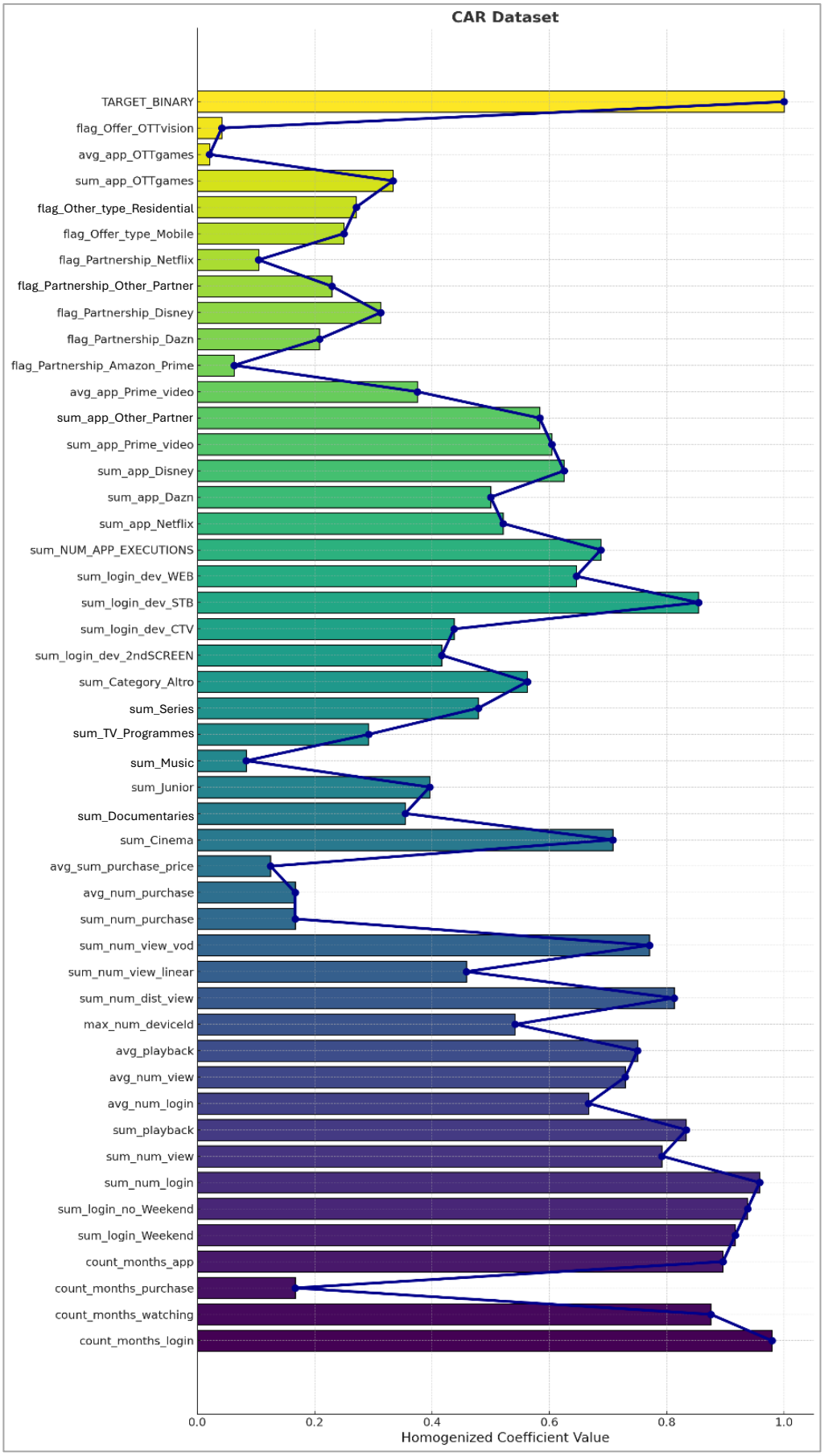

Figure 4.7: Final CAR Dataset

4.3 Split and SMOTE Operations

Once the final CAR is obtained, the Split Data function in Azure is applied to divide the dataset into Training Data and Test Data. The parameters set for splitting the data are as follows:

- **Splitting mode**: Set to "Split Rows". It means that the division is done at the row level. This is the most common approach, where a certain number of rows are separated to create two distinct data sets.
- **Fraction of rows in the first output dataset**: Assigned the value 0.7. This indicates that 70% of the rows from the original dataset will be included in the first output (which corresponds to the training set). The remaining 30% will be allocated to the second output (test set).
- **Stratified split**: It is a parameter that, when set, ensures that the split will maintain the same proportion of the target variable in both the training set and the test set. This is particularly important in classification problems with imbalanced classes. This parameter is set to True. When the stratified split is enabled, a column must be selected for stratification. This column should be the target variable (TARGET_BINARY) to ensure that the proportion of classes remains consistent between the training and test sets. Thus, with this parameter, both sets (training and test) will maintain an accurate representation of the original class distribution.

This parameter is important because the SMOTE technique will now be applied, and the stratified split ensures that the test set retains the original class distribution, providing a fair comparison between the model trained on balanced data and a test set that represents reality. SMOTE is applied after the dataset has been split using Split Data. The training set (which contains 70% of the data) is passed to SMOTE to generate synthetic examples of the minority class. The stratification in the Split Data module ensures that the training set is also representative of the class distribution, and SMOTE operates on this set to further balance the classes. It is important to apply SMOTE only to the training set. This is because the

purpose of SMOTE is to improve the model's training on balanced data, not to alter the final evaluation of the model. Here are the parameters of the SMOTE operation:

- **Label column**: Selected on the target column TARGET_BINARY, which contains the classes the model is trying to predict. SMOTE will generate new synthetic examples for the minority class based on this column.
- **SMOTE percentage**: Set to 100. This value indicates the percentage of oversampling for the minority class. A value of 100% means that SMOTE will add as many synthetic examples as there are existing ones in the minority class, effectively doubling the number of examples for that class.
- **Number of nearest neighbours**: Set to 1. This parameter indicates how many nearest neighbours are considered when creating new synthetic examples. A value of 1 means that each new synthetic example will be created by referencing only one nearest neighbour, resulting in new points that are very similar to the existing samples.

Below, you can see the operations performed in the Azure environment.

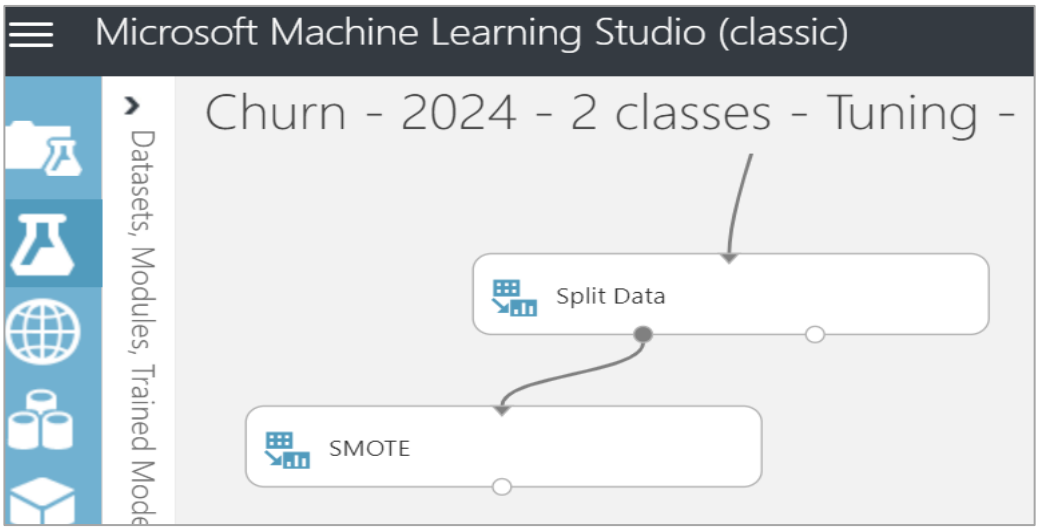

Figure 4.8: Split and SMOTE Operation in Azure

4.4 Algorithms Explanation

In the following paragraphs, the four algorithms used will be analysed. After trying different algorithms and different parameter configurations for each model, only the four best performing algorithms will be presented.
First the parameters used for each algorithm will be described, then the results of the pre-tuning models will be shown, and finally after tuning.
All the models have been tested on Azure environment. On Azure, all the blocks related to the model were connected to the SMOTE block.

4.4.1 Two-Class Logistic Regression

The first machine learning algorithm is the Two-Class Logistic Regression. The algorithm has several parameters:

- **Create trainer mode**: it is set to Single Parameter. This setting means that the algorithm will be trained with a single set of parameters.
- **Optimization tolerance**: this parameter is set to 1E-07: This is the tolerance level for the optimization process. It indicates the precision required for the algorithm to consider the optimization complete. A lower value (like 1E-07) means higher precision but may require more computational time.
- **L1 regularization weight**: it is set to 0.01. L1 regularization adds a penalty equal to the absolute value of the magnitude of coefficients. This parameter helps in feature selection by forcing some of the coefficient values to be exactly zero. The value 0.01 indicates the weight of this regularization.
- **L2 regularization weight**: it is set to 0.01. L2 regularization adds a penalty equal to the square of the magnitude of coefficients. This helps in reducing the model complexity and preventing overfitting. A weight of 0.01 is being applied to the L2 regularization in this case.

- **Memory size for L-BFGS**: the parameter is set to 50. L-BFGS is a limited-memory version of the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, which is used for solving optimization problems. The memory size here refers to the number of past updates that the algorithm will store to approximate the Hessian matrix. A size of 50 is a common choice.
- **Random number seed**: it is set to 1234. This seed value is used to ensure the reproducibility of results. By setting a specific seed, you ensure that the random processes within the algorithm (like initialization of weights) are consistent across different runs.

4.4.2 Two-Class Neural Network

The second algorithm is the Two-Class Neural Network. The algorithm has several parameters:

- **Create trainer mode**: it is set to Single Parameter. This setting means that the algorithm will be trained with a single predefined set of parameters.
- **Hidden layer specification**: set to fully connected case. This parameter indicates that all nodes in the hidden layer are fully connected to the nodes in the next layer, which is typical for traditional neural networks.
- **Number of hidden nodes**: parameter set to 100. This is the number of neurons in the hidden layer of the neural network. A higher number can capture more complexity in the data, but it can also increase the risk of overfitting and computational requirements.
- **Learning rate:** it is set to 0.1. The learning rate controls how quickly the algorithm updates the weights in response to the error it makes on a given batch. A value of 0.1 is relatively high, which can lead to faster convergence but with the risk to skipping over local minima in the cost function.

- **Number of learning iterations**: parameter set to 100. This parameter indicates how many times the algorithm will pass over the entire dataset during training. A higher number can improve learning.
- **The initial learning weight**: it is set to 0.1. This is the initial value assigned to the weights of the neural network before training begins. This parameter can affect the speed and success of the model's convergence.
- **The type of normalizer**: it is set to Min-Max normalizer. This type of normalization scales the data so that the values are within a specified minimum and maximum range, typically between 0 and 1. It is useful for ensuring that all features carry the same weight during training.
- **Shuffle examples**: set on, meaning that the training examples will be shuffled at each iteration to prevent learning a specific order, which can improve the model's generalization.

4.4.3 Two-Class Decision Forest

Then the next algorithm is the Two-Class Decision Forest. The parameters of the algorithm will be explained here:

- **Resampling method:** set to Bagging. Bagging is a resampling method that creates different subsets of the original dataset by sampling with replacement. Each tree in the forest is trained on one of these subsets. This approach reduces the variance of the model and improves its generalization ability.
- **Create trainer mode:** Single Parameter: This setting indicates that the algorithm will be trained with a single set of parameters.
- **Number of decision trees:** set to 8. This parameter indicates the number of decision trees included in the forest. A higher number of trees can improve the model's accuracy but also increases computational cost. In this case, 8 trees are used.

- **Maximum depth of the trees:** set to 32. This parameter limits the maximum depth that each decision tree can reach. Greater depth allows the tree to capture more details in the dataset, but it also increases the risk of overfitting.
- **Number of random splits per node:** set to128. When constructing a decision tree, for each node, the algorithm considers a certain number of random splits of the data to determine the best split. This parameter, set to 128, indicates that 128 possible splits are considered for each node.
- **Minimum number of samples per leaf node:** set to 1. This parameter defines the minimum number of samples that must be present in a leaf node (a terminal node of the tree). A value of 1 means that a leaf node can contain even a single sample, which can lead to greater granularity in the results.

4.4.4 Two-Class Boosted Decision Tree

Then the last algorithm is the Two-Class Boosted Decision Tree. The parameters of the algorithm will be explained here:

- **Create trainer mode:** set to Parameter Range. This setting indicates that the algorithm will be trained across a range of parameter values, allowing for hyperparameter tuning. The best-performing model is selected based on this range.
- **Maximum number of leaves per tree:** set to 2, 8, 32, 128: This parameter specifies the maximum number of leaves (terminal nodes) a tree can have. Multiple values are provided, meaning the algorithm will explore different tree complexities during training. More leaves can capture more detail but can also lead to overfitting.
- **Minimum number of samples per leaf node:** set to 1, 10, 50: This defines the minimum number of samples that must be present in a leaf

node. Lower values allow more granular decisions, while higher values prevent overfitting by requiring more data in each terminal node.

- **Learning rate:** set to 0.025, 0.05, 0.1, 0.2, 0.4: The learning rate controls how much the model adjusts in response to each error it finds. Lower values make smaller adjustments, which can lead to more stable training but may require more iterations. The algorithm will test several different learning rates.
- **Number of trees constructed:** set to 20, 100, 500: This parameter specifies how many trees the boosting algorithm will create. More trees can improve accuracy but also increase training time. The algorithm will evaluate the impact of using different numbers of trees.
- **Random number seed:** set to 1234: This value is used to ensure the reproducibility of results by controlling the random processes within the algorithm, such as sampling and initialization.
- **Allow unknown categories:** This option is checked, meaning the algorithm will handle unknown categories in categorical features instead of failing or ignoring them. It increases the model's robustness when encountering unseen categories in new data.

4.5 Result

In this section, the results of the models explained in the previous paragraphs will be analysed. The table 4.3 shows the performance of all four models in terms of Accuracy, Precision, Recall, F1-Score, and AUC. Before to show the result a brief explanation of these performance measures:

- **Accuracy:** The proportion of correctly predicted instances (both true positives and true negatives) out of the total number of instances. It measures overall correctness:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision:** The proportion of true positive predictions out of all positive predictions. It indicates how many of the predicted positives are actually correct:

$$\frac{TP}{TP + FP}$$

- **Recall:** The proportion of true positives out of all actual positives. It measures the model's ability to find all relevant instances.

$$\frac{TP}{TP + FN}$$

- **F1-Score:** The harmonic mean of precision and recall, balancing both metrics. It's useful when there's an uneven class distribution.

$$2 \times \frac{Precision + Recall}{Precision \times Recall}$$

- **AUC (Area Under the ROC Curve):** Measures the model's ability to distinguish between classes. A higher AUC indicates a better model. It represents the probability that the model ranks a random positive instance higher than a random negative one. It has a range from 0 to 1, where 1 is ideal.

Here the result of the models:

Model	Accuracy	Precision	Recall	F1 Score	AUC
Boosted Decision Tree	0.728	0.387	0.835	0.529	0.814
Decision Forest	0.84	0.554	0.643	0.595	0.868
Logistic Regression	0.813	0.49	0.639	0.555	0.851
Neural Network	0.183	0.183	1.0	0.309	0.436

Table 4.3: Models Performance

In Figure 4.9, you can better observe the metrics to make a comparison and determine which model is the most performant.

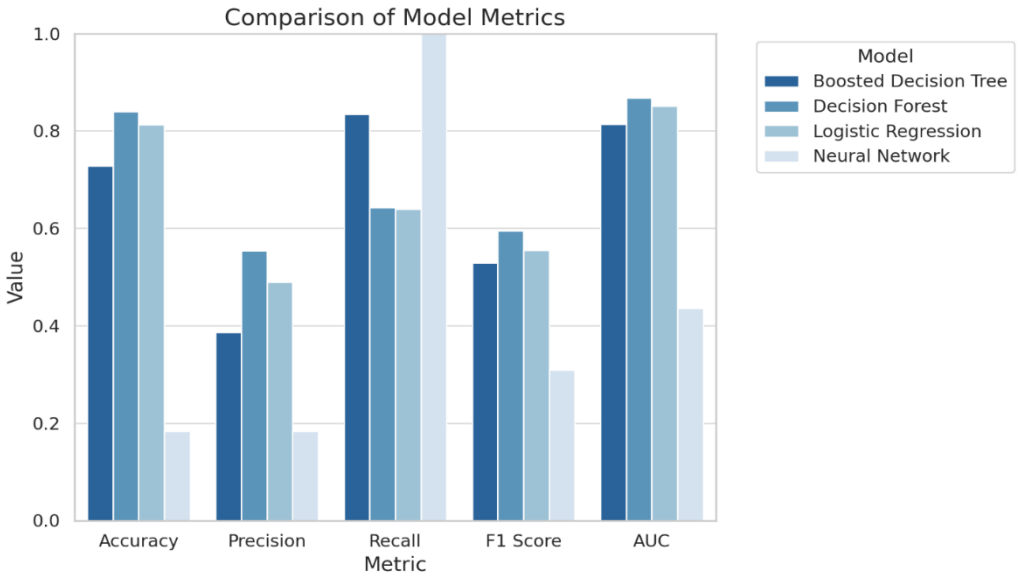

Figure 4.9: Models Performance Comparison

Before commenting on the model results, it is important to note that initially, the models were tested on the original unbalanced version of the dataset, but the results were inferior to those obtained later with the application of SMOTE. Machine learning models, when trained on unbalanced datasets, tend to favour the majority class at the expense of the minority class. This often leads to seemingly good metrics (such as accuracy) but hides the real issue: a poor ability to correctly predict the minority class, as evidenced by low recall or F1-score values for the minority class. In the case of my model, it was expected that the unbalanced dataset would lead to unsatisfactory results, as the model would not be able to effectively learn the patterns of the minority class. This behaviour is consistent with the literature on the subject, which emphasizes the importance of balancing the dataset. It is not possible to show the results as in Table 4.3 due to technical limitations related to the Azure platform, but the results followed the trend described above.

Analysing the results of the pre-tuning models, we can observe different performances depending on the metrics considered. Before tuning, the **Decision Forest** seems to be the most balanced and reliable model.

It stands out as the model with the best overall performance, with high accuracy (0.84) and the best AUC (0.868), indicating a strong ability to distinguish between classes. Its F1 score (0.595) is also the highest among the models, suggesting a good balance between precision and recall. This model seems to be robust even before any tuning, likely due to its ability to combine many decision trees and mitigate the risks of overfitting.

Logistic Regression offers solid overall performance, with an accuracy of 0.813 and an AUC of 0.851. However, its F1 score of 0.555 indicates that while it has decent precision, it may not capture all positives as effectively as the Decision Forest.

Boosted Decision Tree shows an interesting performance with very high recall (0.835) but rather low precision (0.387), indicating that while it is good at identifying positive cases, it also makes many false positives. This leads to a relatively low F1 score (0.529). However, its accuracy (0.728) and AUC (0.814) suggest that the model has good potential, which could be realized through careful tuning.

Finally, **Neural Network** is the outlier among the pre-tuning models. While it has perfect recall (1.0), its precision (0.183) and accuracy (0.183) are extremely low, implying that the model predicts almost everything as positive, leading to unsatisfactory performance across most metrics. This is common in neural network models when they are not adequately optimized.

4.6 Tuning

Tuning in machine learning refers to the process of optimizing a model's parameters to improve its performance.

According to [31] there are two main types of parameters in tuning:

- **Model parameters**: These are the internal parameters of the model that are learned from the data during the training process.
- **Hyperparameters**: These are external parameters that are not learned from the data but need to be set before the model is trained. Examples of hyperparameters include the learning rate in a neural network model or the number of trees in a random forest.

Tuning primarily focuses on optimizing hyperparameters, and there are several techniques to do this:

- **Grid Search**: A brute-force method where a grid of possible values is defined for each hyperparameter, and the model is trained on every possible combination. This method is exhaustive but can be computationally expensive.
- **Random Search**: Instead of trying all possible combinations, random combinations are selected for testing. This method is less computationally expensive than grid search and can be effective in finding good hyperparameters.
- **Bayesian Optimization**: A more sophisticated approach that builds a probabilistic model of the model's performance as a function of the hyperparameters and uses this model to choose which hyperparameter values to test.

The goal of tuning is to find the combination of hyperparameters that optimizes a performance metric, such as accuracy, precision, recall, or another measure relevant to the specific problem being solved. In this thesis, it was decided to optimize based on the AUC. This choice is guided by the fact that the application of SMOTE generally tends to improve the AUC since by balancing the classes, the model has more information to distinguish between positives and negatives. Although SMOTE might lower Precision due to an increase in false positives, the AUC often remains high or improves because it considers the model's performance across all possible classification thresholds, not just a fixed threshold. A good AUC indicates that the model, even after the application of SMOTE, is still

effective in discriminating between classes despite potential issues with Precision. The AUC is a metric that measures a model's ability to distinguish between positive and negative classes, regardless of the chosen threshold. In a churn prediction model, a high AUC is particularly useful because it indicates that the model effectively differentiates between customers at risk of churn and those who are likely to remain loyal. Since the AUC is independent of the decision threshold, it allows for greater flexibility in business strategy without compromising the model's performance. In summary, a high AUC makes the model robust and adaptable, ensuring accurate identification of churners and effectively supporting the company's retention efforts. Two techniques available on Azure ML were performed:

- **Random sweep**: the chosen tuning method is Random Search.
- **Random Grid:** a combination of Grid Search and Random Search. With Random Grid, a random selection of hyperparameter combinations is chosen from a predefined grid of values, instead of systematically exploring all combinations.

4.7 Tuning Result

It is important to highlight before showing the tuning results that cross-validation was performed for all models both before and after tuning. Cross-validation is a widely used statistical technique in machine learning to assess a model's ability to generalize to independent data beyond the training set. This process provides an average estimate of the model's performance. Cross-validation minimizes the risk of overfitting, ensuring the model is not overly optimized for a single dataset, and efficiently utilizes the entire dataset since each data point is used for both training and testing [32]. The table below shows the results of the models after tuning.

Model	Accuracy	Precision	Recall	F1 Score	AUC
Logistic Regression	0.815	0.488	0.642	0.558	0.852
Neural Network	0.332	0.273	1.0	0.429	0.861
Decision Forest	0.845	0.564	0.672	0.613	0.891
Boosted Decision Tree	0.846	0.566	0.682	0.619	0.895

Table 4.4: Models Performance After Tuning

In Figure 4.10, you can better observe the metrics to make a comparison and determine which model is the most performant.

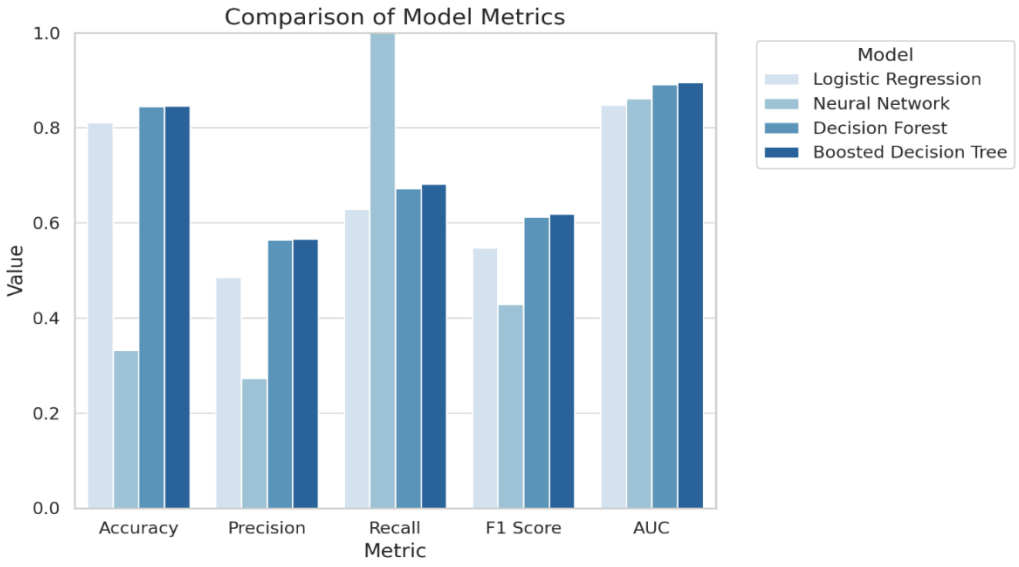

Figure 4.10: Models Performance Comparison After Tuning

First, it is evident that both in the pre-tuning and post-tuning models, the Precision of all models is relatively low. The use of SMOTE can significantly influence a model's Precision. SMOTE generates new synthetic instances for the minority class, thereby increasing the number of positive examples in the dataset. This improves the model's ability to correctly recognize positives, leading to an increase in Recall. However, it can also result in a higher number of false positives, particularly if the model struggles to distinguish between the synthetic instances and those that belong to the

majority class. For this reason, as explained in the previous paragraph, it was decided to consider AUC as the reference metric.

After tuning, we can see that the performance of the models has changed, the **Neural Network** saw the most significant improvements after tuning. Its accuracy, precision, and F1 Score all improved considerably, and the AUC showed a massive increase, indicating a much better balance between the true positive and false positive rates. However, the recall stayed at 1.0, indicating the model is still biased toward positive predictions, which might explain why precision remains relatively low compared to other models. For **Logistic Regression** the tuning has led to small but positive improvements in most metrics, with a slight gain in accuracy, recall, F1 Score, and AUC. Precision saw a very small decrease, but it is negligible. These small changes indicate that the model has become marginally more effective after tuning, but the improvements are subtle and may not be significant enough for practical changes in performance. **The Decision Forest** saw minor improvements across all metrics after tuning. Precision, recall, F1 Score, and AUC all increased slightly, making it a more balanced model overall. While the improvements are not dramatic, they suggest that the tuning process helped the model achieve a slightly better performance. Finally, The **Boosted Decision Tree** saw significant improvements in accuracy, precision, F1 score, and AUC. However, its recall dropped quite a bit (−0.153), suggesting the model may have become more conservative in predicting positive cases, improving precision at the expense of recall. Overall, this trade-off seems to have paid off, as the F1 Score and AUC both improved, and it now stands as the best model overall.

The Boosted Decision Tree benefited from tuning due to its iterative nature and its ability to better optimize hyperparameters to correct errors, resulting in a more robust and high-performing model compared to the other models post-tuning. This explains why it became the best model after tuning, surpassing even the Decision Forest, which was initially the top performer before tuning.

After identifying the Boosted Decision Tree as the model with the best overall performance, it is useful to further explore the behaviour of each model through two fundamental tools: ROC Curves and Confusion Matrices. The ROC Curves provide an overview of the models' performance, highlighting their ability to balance true positive and false positive rates.

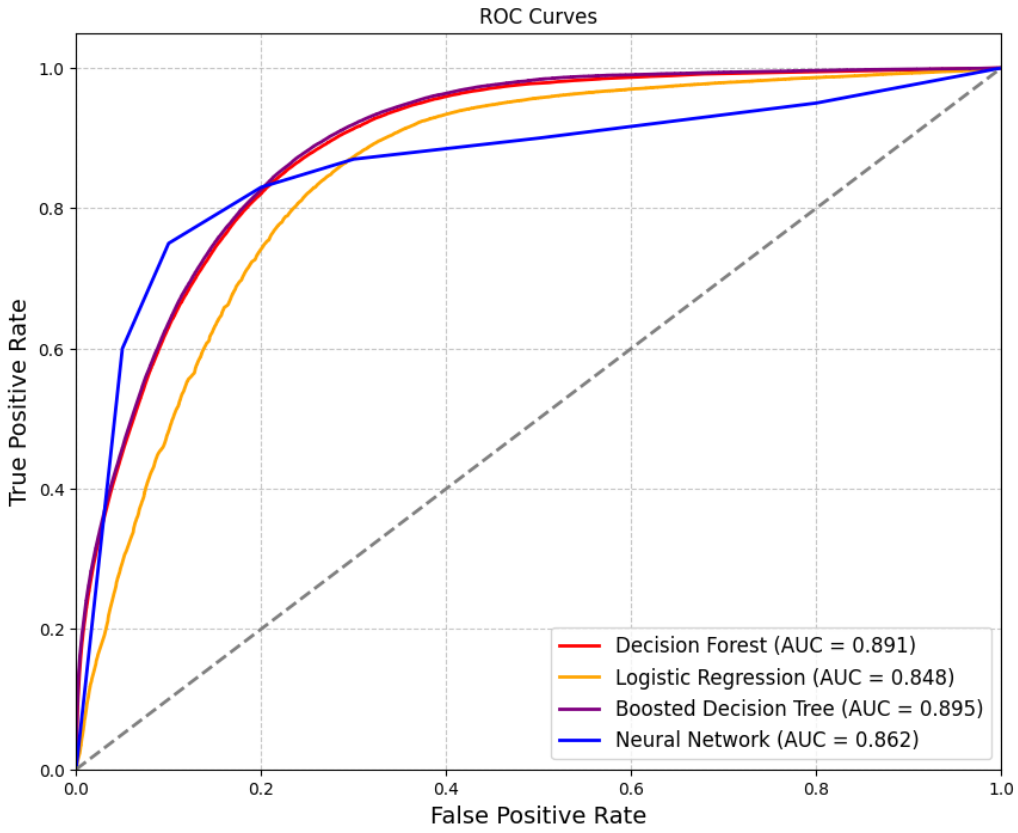

Figure 4.11: Roc Curves Comparison

As we can see from 4.11 the Boosted Decision Tree has the best overall performance, followed closely by the Decision Forest. Logistic Regression and Neural Network models show relatively weaker performance. When selecting a model based on its ability to balance the true positive rate against the false positive rate (discriminating between classes), the Boosted Decision Tree is the top choice, with a very high AUC.

Instead, the Confusion Matrices allow us to examine the balance between correct and incorrect predictions, giving us a clear view of false positives and false negatives.

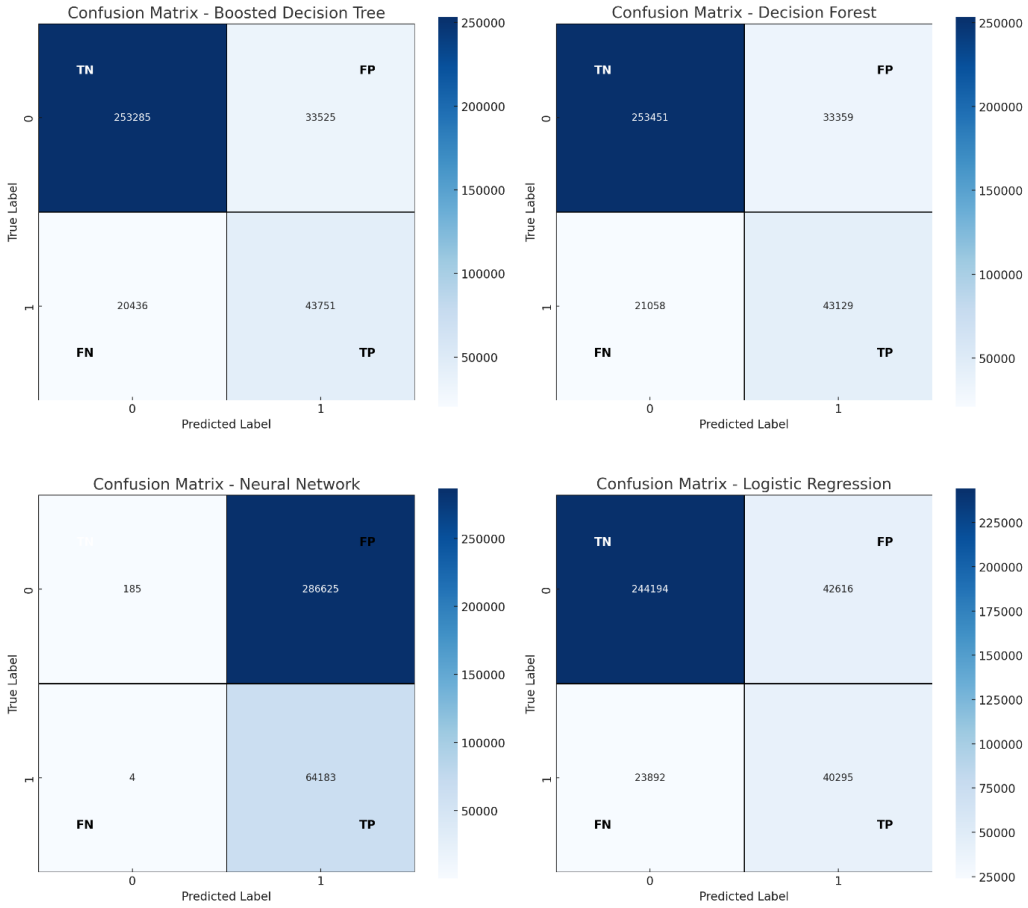

Figure 4.12: Confusion Matrix Comparison

As seen from the confusion matrices above, the Boosted Decision Tree has a good balance between true positives and true negatives. The number of false positives and false negatives is relatively low, which contributes to its high AUC and F1 score. The Decision Forest has slightly more false negatives and fewer true positives compared to the Boosted Decision Tree, but overall, its performance is very similar. It also has slightly fewer false positives. The Neural Network has an extremely high recall, meaning it captures almost all the true positives (only four false negatives). However, it struggles with precision due to an extremely high rate of false positives. This is reflected in its low precision and overall AUC, as it is heavily biased toward predicting positives. Logistic Regression has a higher number of false positives and false negatives compared to the tree-based models, which leads to its lower AUC and overall weaker performance. It captures fewer true positives and has more false positives compared to the better-performing models.

Chapter 5

Model Implementation

Summary

In the previous chapter, the models used, and their respective results were described. Throughout this thesis, almost all the steps outlined in figure 3.1 have been covered. In this final chapter, the last missing step, Target Identification for Campaigns, will be discussed. In this step, the "best performing" model is applied to customers with unknown behaviour, generating a score (churn/loyalty probability) for everyone. Customers at high risk of churn are identified and can be targeted with specific campaigns or other personalized actions.

5.1 Model Inference: Application for Predictions on New Data

As highlighted in Chapter 4, the "best performer" model is the Two-Class Boosted Decision Tree. Therefore, this is the model to be deployed into production, meaning that the algorithm will be transferred from Azure ML to a new environment agreed upon with the client. In this environment, whenever new data is received, the model is retrained and used to make inferences.

Inference is the process through which a trained model is applied to make predictions or estimates on new data. In other words, once the model has been trained, it is used to predict or estimate outcomes on data it hasn't seen during training. For example, in the case of a churn classification model, after being trained on a historical dataset, the model is used to make inferences on new customers. It takes the features of a new customer as input and estimates the probability that this customer will churn. Inference thus allows the model to be applied beyond the training context, providing useful predictions for future decision-making.

5.2 Churn Probability and Targeted Retention Strategies Based on Risk Levels

The machine learning model created returns a numerical value between 0 and 1. This number represents the probability that a customer will churn. Essentially, a value close to 0 indicates a low likelihood of churn, while a value close to 1 suggests a high probability that the customer will leave the service. The threshold used to decide whether to classify a customer as being at risk of churn is set at 0.5. If the probability returned by the model exceeds this threshold, the customer is classified as a potential churner. This approach allows for more targeted decision-making, such as adopting marketing or retention strategies for customers with a high probability of churn. However, it is important to adopt different strategies based on the levels of churn risk. Not all at-risk customers have the same value, so a good strategy should also consider Customer Lifetime Value (CLV).

For example, for customers with a low probability of churn, below 30%, there is no need to adopt intensive measures; instead, it is better to focus on loyalty campaigns, such as reward programs or promotions for complementary products, to strengthen the relationship. For customers with a churn probability between 30% and 60%, proactive engagement strategies are advisable, such as moderate discounts or personalized

offers, to maintain attention and prevent churn. When the churn probability exceeds 60%, customers are at risk and require more aggressive actions, such as significant discounts or personalized plans, to retain them. For customers with a churn probability above 80%, a more aggressive retention strategy may be useful, but only for high-value customers; for others, it may be better not to intervene to save resources. The ideal approach involves segmenting customers based on their risk level and tailoring marketing actions, accordingly, focusing efforts on the most at-risk and high-value customers while avoiding overburdening those with a low probability of churn.

To further illustrate how churn probability influences retention strategies, the following chart provides a breakdown of the different risk levels, and the corresponding actions recommended for each customer segment.

The orange line represents the churn probability predicted by the model, ranging from 0 (low churn risk) to 1 (high churn risk). This line visualizes how the model assigns churn likelihoods. Retention strategies and thresholds, shown by purple dashed lines, are applied to different probability segments.

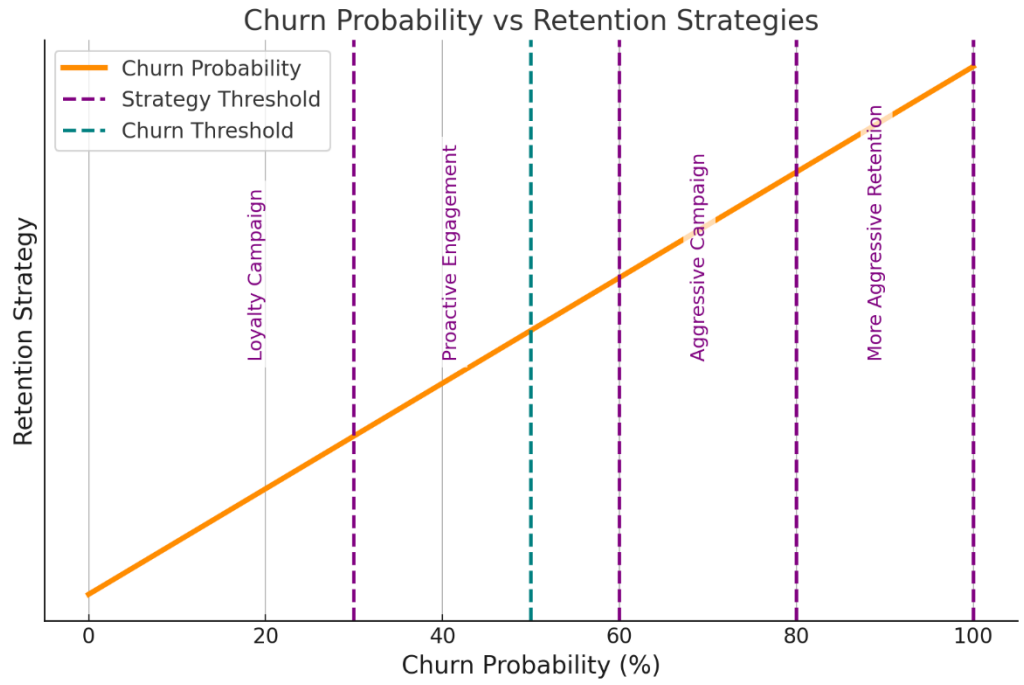

Figure 5.1: Churn Strategies

Chapter 6

Conclusion

This thesis successfully explored various machine learning techniques for predicting customer churn in a digital media industry context, specifically focusing on a leading OTT platform.

The findings of this research provide valuable insights into customer behaviour patterns and offer actionable strategies for retention. Moreover, the methodologies developed in this study can be further applied to other sectors, paving the way for more personalized and efficient customer engagement strategies.

The models tested, including Logistic Regression, Decision Forest, Neural Networks, and Boosted Decision Trees, were evaluated for their performance in predicting churn behavior using various metrics such as Accuracy, Precision, Recall, F1-score, and AUC.

One of the key challenges addressed in this study was the issue of class imbalance, where the number of churners was significantly smaller compared to non-churners. The use of SMOTE (Synthetic Minority Over-sampling Technique) proved to be a successful approach to balance the dataset, improving the models' ability to identify churners without compromising performance on the non-churners.

The results demonstrated that ensemble methods, particularly Decision Forest and Boosted Decision Trees, outperformed simpler models such as

Logistic Regression. These models benefited from their ability to handle complex patterns in the data and leverage non-linear relationships between features. Furthermore, the analysis underscored the importance of feature selection and engineering, as certain features related to user behavior, such as login frequency and content consumption, played a critical role in churn prediction.

In conclusion, this thesis not only highlights the effectiveness of machine learning in predicting customer churn but also emphasizes the importance of addressing class imbalance and selecting relevant features. The insights gained can guide future improvements in customer retention strategies, ultimately contributing to business sustainability in the highly competitive digital media industry. For future work, expanding the dataset to include more user interactions and testing additional algorithms, such as deep learning models, could further enhance predictive performance.

References

[1] Verbeke, W., Dejaeger, K., Martens, D., Hur, J., & Baesens, B. (2012). *New insights into churn prediction in the telecommunication sector: A profit driven data mining approach*. European Journal of Operational *Research, 218*(1), 211-229. *https://doi.org/10.1016/j.ejor.2011.09.031*

[2] Amin A, Anwar S, Adnan A, Nawaz M, Alawfi K, Hussain A, Huang K. (2017) *Customer churn prediction in the telecommunication sector using a rough set approach*. Neurocomputing; 237:242–54.

[3] Lazarov, V., & Capota, M. (2007). *Churn Prediction*. Technische Universität München.

[4] Barsotti, A., Gianini, G., Mio, C., Lin, J., Babbar, H., Singh, A., Taher, F., & Damiani, E. (2024). *A decade of churn prediction techniques in the TelCo domain: A survey*. SN Computer Science, 5*(404)*. *https://doi.org/10.1007/s42979-024-02722-7*

[5] Scherer, M. (2023). *Predicting Churn Rate in Companies*. In: Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., Zurada, J.M. (eds) Artificial Intelligence and Soft Computing. ICAISC 2023. Lecture Notes in Computer Science(), vol 14126. Springer, Cham. https://doi.org/10.1007/978-3-031-42508-0_16

[6] Ekawati, A. D. (2019). *Predictive analytics in employee churn: A systematic literature review*. Journal of Management Information and Decision Sciences, 22(4), 387-397. https://www.abacademies.org/articles/predictive-analytics-in-employee-churn-a-systematic-literature-review.pdf.

[7] Vafeiadis, T., Diamantaras, K. I., Sarigiannidis, G., & Chatzisavvas, K. C. (2015). A comparison of machine learning techniques for customer churn prediction. *Simulation Modelling Practice and Theory, 55*, 1-9. https://doi.org/10.1016/j.simpat.2015.03.003

[8] Schmidhuber, J. (2015). *Deep learning in neural networks: An overview*. Neural Networks, *61*, 85-117. https://doi.org/10.1016/j.neunet.2014.09.003

[9] Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0

[10] Starbuck, C. (2023). Logistic Regression. In: The Fundamentals of People Analytics. Springer, Cham. https://doi.org/10.1007/978-3-031-28674-2_12

[11] Biau, G., Scornet, E. (2022). *A Random Forest Guided Tour*. International Journal of Data Science and Analytics, *15*(1), 65-90. https://doi.org/10.1007/s41060-022-00280-3

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

[13] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5), 1189–1232. https://doi.org/10.1214/aos/1013203451

[14] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, 30. https://doi.org/10.5555/3294996.3295074

[15] Zhu, B., Baesens, B., & vanden Broucke, S. K. L. M. (2017). *An empirical comparison of techniques for the class imbalance problem in churn prediction*. Information Sciences, 408, 84–99. https://doi.org/10.1016/j.ins.2017.04.015

[16] Ali, A., Shamsuddin, S. M., & Ralescu, A. L. (2015). *Classification with class imbalance problem: A review*. International Journal of Advance Soft Computing and Applications, 5(3), 1-19.

[17] Elreedy, D., & Atiya, A. F. (2019). *A Comprehensive Analysis of Synthetic Minority Oversampling Technique (SMOTE) for Handling Class Imbalance*. Information Sciences, 505, 32-64. https://doi.org/10.1016/j.ins.2019.07.070

[18] Pradipta, G. A., Sanjaya, I. N. H., Wardoyo, R., Musdholifah, A., & Ismail, M. (2021). SMOTE for handling imbalanced data problem: A review. *Sixth International Conference on Informatics and Computing (ICIC)*, 20-29.
https://doi.org/10.1109/ICIC54025.2021.9632912

[19] Islam, R., Patamsetti, V. , Gadhi, A. , Gondu, R. , Bandaru, C. , Kesani, S. and Abiona, O. (2023) *The Future of Cloud Computing: Benefits and Challenges*. International Journal of Communications, Network and System Sciences, 16, 53-65.
10.4236/ijcns.2023.164004.

[20] Khan, T., Tian, W., Zhou, G., Ilager, S., Gong, M., & Buyya, R. (2022). *Machine learning (ML)-centric resource management in cloud computing: A review and future directions*. Journal of Network and Computer Applications*, 204*, 103405. https://doi.org/10.1016/j.jnca.2022.103405

[21] Butt, U. A., Mehmood, M., Shah, S. B. H., Amin, R., Shaukat, M. W., Raza, S. M., Suh, D. Y., & Piran, M. J. (2020). *A review of machine learning algorithms for cloud computing security*. Electronics, *9*(9), 1379. https://doi.org/10.3390/electronics9091379

[22] Qayyum A, Ijaz A, Usama M, Iqbal W, Qadir J, Elkhatib Y and Al-Fuqaha A (2020) Securing Machine Learning in the Cloud: A Systematic Review of Cloud Machine Learning Security. Front. Big Data 3:587139. https://doi.org/10.3389/fdata.2020.587139

[23] Wankhede, P., Talati, M., & Chinchamalatpure, R. (2020). *Comparative study of cloud platforms - Microsoft Azure, Google Cloud Platform and Amazon EC2*. International Journal of Research in Engineering and Applied Sciences*, 05*(02), 60-64.

[24] Tilly Kenyon (2021, June 15). *The advantages and disadvantages of AI in cloud computing*. AI Magazine. Retrieved August 2, 2024, from https://aimagazine.com (AI News).

[25] Okeke, F. (n.d.). *Disadvantages of Cloud Computing*. TechRepublic. Retrieved August 2, 2024, from https://www.techrepublic.com (TechRepublic).

[26] Wawira, M. (2024, August 2). *13 Disadvantages of Cloud Computing: Solve Cloud Challenges*. Cloudwards. Retrieved August 2, 2024, from https://www.cloudwards.net

[27] Murel, J., & Kavlakoglu, E. (2023, November 21). *What is multicollinearity?* IBM from https://www.ibm.com/topics/multicollinearity

[28] Jolliffe, I. T. (2002). *Principal component analysis* (2nd ed.). Springer New York, NY. https://doi.org/10.1007/b98835

[29] Chan, J.Y.-L.; Leow, S.M.H.; Bea, K.T.; Cheng, W.K.; Phoong, S.W.; Hong, Z.-W.; Chen, Y.-L. Mitigating the Multicollinearity Problem and Its Machine Learning Approach: A Review. *Mathematics* 2022, *10*, 1283. https://doi.org/10.3390/math10081283

[30] Lindner, T., Puck, J. & Verbeke, A. (2022) Beyond addressing multicollinearity: Robust quantitative analysis and machine learning in international business research. *J Int Bus Stud* 53, 1307–1314. https://doi.org/10.1057/s41267-022-00549-z

[31] Bergstra, J., & Bengio, Y. (2012). *Random Search for Hyper-Parameter Optimization*. Journal of Machine Learning Research, 13(Feb): 281-305 DOI: 10.5555/2188385.2188395

[32] Kohavi, R. (1995). *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*. Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI). Vol. 2, pp. 1137-1143. 10.5555/1643031.1643047