A Multi-Model and Explainable Machine Learning Framework for Cross-Industry Churn Prediction

Anish Rao

Student ID: 20066423

Dublin Business School

Applied Research Project submitted in partial fulfilment of the requirements

for the degree of

MSc in Data Analytics

January 2026

Declaration

I declare that this Applied Research Project that I have submitted to Dublin Business School for the award of MSc in Data Analytics is the result of my own investigations, except where otherwise stated, where it is clearly acknowledged by references. Furthermore, this work has not been submitted for any other degree.

Anish Rao

Student Number: 20066423

January 2026

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Professor Dr. Baidyanath Biswas, for his invaluable guidance and support throughout this research project. His expertise and constructive feedback were instrumental in shaping this work. I am grateful to Dublin Business School and the faculty of the MSc in Data Analytics programme for providing the knowledge and resources that enabled this research. I also thank the researchers who made their datasets publicly available on Mendeley Data, making this cross-industry analysis possible. Finally, my appreciation goes to my family and friends for their continuous support and encouragement throughout this academic journey.

Abstract

Customer churn is when a customer decides to stop using a business's service. This is a big challenge for businesses since retaining a customer cost less than acquiring new ones. This research compares six machine learning algorithms - Logistic Regression, Random Forest, XGBoost, CatBoost, Support Vector Machine, and K-Nearest Neighbors, for predicting churn across two sectors, telecommunications and insurance. Two datasets are used - one with 8,454 business customers from a Bulgarian telecom operator another with 40,284 customers from a Spanish insurance company. The methodology includes data preprocessing, SMOTE for class imbalance, GridSearchCV for hyperparameter tuning, and cross-validation. Performance was measured using accuracy, precision, recall, F1-score, and ROC-AUC. Results showed insurance models performed much better than telecommunications models, showing how quality and size of dataset impact model effectiveness. SHAP analysis also revealed different feature importance patterns across sectors. A Streamlit application was developed for generating predictions and compare model performance.

Table of contents

List of Figures

List of tables

Chapter 1

1. Introduction

1.1 Background

Customer churn, also known as the discontinuation of a customer's relationship with a service provider. It represents one of the most critical challenges faced by businesses across industries today. Lots of researches have shown getting new customers costs five to six times more than retaining the existing ones (Verbeke et al., 2012). In sectors like telecommunication and insurance, understanding and predicting churn has become important to maintain profitability and market share.

Traditional approaches for churn management mostly depended on reactive strategies, addressing customer dissatisfaction only after it happened. However, availability increase in customer data and advancement in analytical capabilities have enabled a shift towards predictive and proactive approaches. Many machine learning techniques now let us find at-risk customers before they churn which allows businesses to implement targeted retention strategies (Larivière and Van den Poel, 2005). This research project explores how six different machine learning algorithms perform in predicting churn across two distinct industries (telecommunication and insurance), helping in finding out how predictive accuracy and feature importance patterns vary between the two industries.

1.2 Importance of Churn Prediction

Churn prediction provides lot of advantages by helping in active customer retention instead of just damage control. When businesses identify at-risk customers early, they can try to use targeted techniques for intervention like personalized offers, service improvements, or loyalty programmes before customers leave. Customer retention is more cost-effective than new acquisition as long-term customers provide higher profits and are usually less sensitive to competitive marketing activities and therefore become less costly to serve and may provide new referrals through positive word-of-mouth (Verbeke et al., 2012).

Apart from the immediate financial benefits, churn prediction models also provide strategic insights into a customer's behaviour pattern and satisfaction drivers. By identifying the factors most strongly affecting the churn, organisations can figure out any underlying systemic issues in their service, customer experience, or pricing structures. All of this information can be very valuable in highly competitive markets where a customer's expectations evolve very fast and switching costs continue to decline.

1.3 Advances in Machine Learning for Customer Analytics

Recent developments in machine learning have improved the customer analytics capabilities a lot. Traditional statistical methods are useful for establishing baseline predictions, but they usually struggle to capture any complex or non-linear relationships that are present in customer data. Newer algorithms like Random Forest, XGBoost and CatBoost can automatically see intricate interactions between variables without ever needing any explicit specifications or instructions (Breiman, 2001; Chen and Guestrin, 2016; Prokhorenkova et al., 2018).

There have also been lots of advances in model interpretability. While traditional black-box models have shown higher predictive accuracy, their non-transparency creates challenges for businesses to come up with any solutions and regulatory compliance. The development of explainable frameworks like SHAP and LIME have helped in resolving this issue. These frameworks help non-technical stakeholders to understand which customers will churn and the reason behind them churning (Lundberg and Lee, 2017; Ribeiro et al., 2016). This interpretability helps in translating model outputs into actual actionable business strategies.

1.4 Challenges in Multi-Industry Prediction

Developing churn prediction models across different industries have many methodological challenges. Customer behaviour patterns, definitions of churn and predictive features vary a lot between different industries. In telecommunication for example churn usually happens more quickly than other sectors. It is driven by factors like service quality, pricing and offers from competitors. Churn in insurance however happens over longer timeframes and may be influenced by different coverage needs, life events, policy renewals. (Risselada et al., 2010; Vafeiadis et al., 2015).

Different dataset characteristics increase challenges. The telecommunications dataset used in this research has 8,454 customer data with a churn rate of 6.49% (Tokmakov, 2024). The insurance dataset has 40,284 customer data with a much higher churn rate of 23.83% (Guillen et al., 2021). This difference in sample size and class imbalance needs consideration for sampling techniques, and model selections. The feature engineering also differs between the industries as telecommunications relies on usage patterns and insurance uses demographic factors.

1.5 Overview of Approach

This research project does a comprehensive analysis by comparing six machine learning algorithms - Logistic Regression, Random Forest, XGBoost, CatBoost, Support Vector Machine, and K-Nearest Neighbors; across the two datasets. The methodology includes careful data preprocessing like handling missing values, encoding categorical variables, scaling numerical variables, and handling of class imbalance using Synthetic Minority Over-sampling Technique (SMOTE) (Chawla et al., 2002). Hyperparameter tuning is also done using GridSearchCV with stratified cross-validation.

Model evaluation is done using multiple metrics - accuracy, precision, recall, F1-score, and ROC-AUC to help see complete predictive performances (Fawcett, 2006). In addition to these comparisons, SHAP analysis helps in providing insights on feature importance patterns within each industry (Lundberg and Lee, 2017). The research project then finishes off with a Streamlit web application that enables users to do real-time churn prediction using the best models and provides an interactive interface to compare all model metrics and performances.

1.6 Research Objectives

This research has three primary objectives.

- Which machine learning algorithms (Logistic Regression, Random Forest, XGBoost, CatBoost, SVM, KNN) have the best predictive performance in the telecom and insurance sectors for customer churn?
- Which are the most important features in both these sectors for customer churn, and how do these differ between telecom and insurance sectors?
- A Practical implementation using a functional web application that integrates both model predictions and model analysis/comparisons.

The final artefact helps in bridging the gap between academic research and business application. Showing how advanced machine learning techniques can be deployed in user-friendly formats suitable for non-technical stakeholders.

1.7 Rationale and Significance

This research addresses a critical gap in churn prediction literature by giving a systematic cross-industry comparison using standard methodologies. While there are lots of studies about churn prediction in individual sectors, there are very few that have investigated how model performances and feature importance patterns differ across multiple industries when similar techniques are applied. This comparative research provides practical insights for organisations that consider model deployment and theoretical understanding of churn dynamics.

The research significance extends from just algorithm comparison and shows how practically these models can be deployed and interpreted. By developing a functional web application integrated with all model metrics for comparison and SHAP visualisations, this project shows how machine learning techniques can be used for business use. This implementation also shows how valuable it is with the growing emphasis on responsible AI practices in the modern day and the need for transparent, explainable predictions in customer-facing applications.

1.8 Structure of the Report

The rest of this report is as follows:

- **Chapter 2:** reviews relevant literature on churn prediction, machine learning techniques, and model explainability.
- **Chapter 3:** details of the research methodology like dataset description, preprocessing techniques, and model development approach.
- **Chapter 4:** describes the implementation of the research design and Streamlit web application.
- **Chapter 5:** shows all the results and analysis
- **Chapter 6:** discusses the findings and limitations.
- **Chapter 7:** concludes with summary of contributions and recommendations for future research.

Chapter 2

2. Literature Review

2.1 Overview of Existing Churn Prediction Methods

Customer churn prediction has evolved into a very important and impacting issue in service-based industries. The economic impact of customers churning is also very high. Research shows that acquiring new customers cost companies five to six times more than it would cost to retain the existing customers (Verbeke et al., 2012). This difference in cost is more impactful in competitive markets, which changes churn prediction from a tactical tool to a very important part of business survival and profitability.

The churn prediction domain has gone through many changes in the last few decades. Initial methods depended mostly on simple rule-based systems, such as identifying customers that weren't engaging with services within specified timeframes. These methods only provided baseline insights, and they were not good enough for capturing the complex nature of customer churning decisions. The newer methods however have methodologies that use machine learning to analyse hundreds of variables simultaneously, detecting subtle patterns that are not detected by traditional analytical approaches.

Industry specific characteristics heavily influence churn and prediction approaches. In telecommunications, customer churn happens more frequently, usually monthly. This makes it important for rapid identification of at-risk customers and immediate intervention. Insurance sectors however usually have longer timescale happening usually annually depending on policy renewal cycles. This gives a more extended window for customer engagement but needs different analytical frameworks to figure out renewal decisions.

2.2 Traditional Machine Learning Approaches

Logistic regression has served as foundation for churn prediction for decades, valued for its statistical foundation and how simple it is. It gives transparent coefficient interpretations which help stakeholders to identify which factors contribute the most in customers leaving (Hosmer and Lemeshow, 2013). Its main advantage is how easy it is to explain results to stakeholders who create retention strategies.

Support Vector Machines are effective when dealing with high-dimensional feature spaces in customer analytics. Vapnik (1998) established theoretical foundations for SVM's capacity to find best decision boundaries. They do have a big drawback of high computational requirements and low interpretability that make them less used in business contexts that need transparent models.

K-Nearest Neighbors is a very straightforward approach, identifying churners by comparing behaviour of similar/comparable customers. Cover and Hart (1967) provide a methodology useful when specific customer segments show different churn patterns. However, KNN's sensitivity to feature scaling and slow performance with large data presents challenges in production environments with large-scale customer databases.

2.3 Ensemble Methods and Advanced Algorithms

Random Forest was a significant advancement in churn prediction capabilities since it addressed the drawbacks of basic decision trees while also providing interpretability. Combining multiple decision trees by random sampling of features produces more accurate predictions while also giving complete feature importance rankings (Breiman, 2001). Some studies consistently show Random Forests getting superior accuracy across different contexts with the additional benefit of providing actionable insights about which customer features most strongly predict churn (Larivière and Van den Poel, 2005). This random sampling helps overcome overfitting where models learn noise rather than actual patterns. The algorithm achieves stability by constructing each tree using different random subsets of features. This provides generalisability which is important for reliable business deployment.

XGBoost is one of the preferred choices for churn prediction because of its ability to learn complex customer behaviour patterns while also reducing overfitting risks. This gradient boosting framework was developed by Chen and Guestrin (2016) to provide iterative model improvement. It can recognise subtle patterns that other frameworks would usually ignore. Some recent implementations when configured properly have shown accuracy rates up to 85-95%. It works specially well with telecommunication applications where usage patterns show lots of complexity.

CatBoost is one of the latest evolutions in gradient boosting algorithm. It specifically improves on the handling of categorical data which is very common in customer analytics. Prokhorenkova et al. (2018) created the algorithm so that it requires minimal data preprocessing while also maintaining accuracy comparable to XGBoost. The built-in categorical variable handling helps in eliminating challenges such as information loss or introducing bias in customer segment variables. This helps in faster deployment and still have good performance.

2.4 Industry-Specific Applications

Telecommunication companies have helped the most in churn prediction research with detailed usage data availability and highly competitive pressures. This sector usually faces the highest churn rates which makes it very important to make accurate predictions for targeted retention efforts. Neural network approaches were developed by Mozer et al. (2000) specifically for telecommunications churn prediction, which showed significant improvements on performance by traditional statistical methods. Their research provided evaluation frameworks and approaches for feature engineering that are still used even today for telecommunication analytics.

Vafeiadis et al. (2015) compared multiple machine learning techniques for the telecommunication churn prediction. They found that ensemble methods would consistently outperform the traditional methodologies. Their evaluation also concluded that the telecommunications industry has rich usage data and very frequent customer interactions that provide perfect conditions for comprehensive predictive modelling. The study now serves as a benchmark for algorithm comparison studies in churn prediction.

A new profit-driven approach for telecommunications churn prediction was introduced by Verbeke et al. (2012). It shifted focus from accuracy maximisation to business impact optimisation. Their research showed that models that considered the economic impact of different prediction error types would provide better business outcomes compared to the other accuracy focused alternatives. This work established the importance of aligning objectives of the organisation with evaluation metrics instead of just purely statistical criteria.

Insurance churn prediction however has different challenges due to the annual policy decision cycles rather than continuous customer engagement. This long time-frame structure allows for longer intervention windows but also requires understanding of long-term customer relationship patterns and long lifecycle effects. Dynamic modelling approaches for predicting customer lifetime value for insurance sectors was developed by Risselada et al. (2010). They show how hierarchical techniques can be used by insurance companies to understand customer value evolution over a period and come up with the best intervention strategies.

Larivière and Van den Poel (2005) used Random Forests and regression forest methods for insurance churn. This helped in advancing prediction capability by forecasting churn probability and customer profitability at the same time. Their work showed that ensemble methods give more sophisticated retention ideas that consider both, likelihood of leaving and customer value. This helps in proper resource allocation that maximises return on investments.

Some recent research has also shown more refinement in insurance churn prediction. Spiteri and Azzopardi's (2018) work on motor insurance churn using ensemble methods showed that claim history and policy characteristics were primary predictive factors.

2.5 Model Evaluation and Class Imbalance

Modern day churn prediction research goes further than just simple accuracy metrics to cover both business impact and practical implementation requirements. Having multiple evaluation metrics is nowadays the standard practice to evaluate better model performances when churn rate is low. Fawcett (2006) came up with a comprehensive guidance on analysing ROC and evaluating classification performance which helped establish standards that are followed in modern churn prediction. His work mainly emphasised the importance of different types of prediction errors and their related business costs when evaluating models performance.

Class imbalance is a persistent challenge where churning customers are in the minority of the dataset that again require complex evaluation approaches. Chawla et al. (2002) developed SMOTE (Synthetic Minority Over-sampling Technique) to help with this issue. They showed that basic accuracy metrics can be misleading when churn rates are low and established precision, recall, F1-score, and ROC-AUC as standard evaluation criteria. They provided more refined assessments of model performance across different operating points.

2.6 Model Explainability and Interpretability

Organisations nowadays have an increasing need for understanding not just which customers might leave their company but also why they decide to leave. This has created a demand for explainable AI methods that can interpret the outputs from machine learning models for non-technical stakeholders. SHAP (SHapley Additive exPlanations) is the leading approach for understanding how different customer characteristics contribute to predicting churn. This framework was developed by Lundberg and Lee (2017), based on cooperative game theory that provided mathematically rigorous feature attributions that business stakeholders can trust with confidence.

SHAP analysis usually shows that monthly charges, contract type and interactions with customer support are the main churn drivers. The model-agnostic nature of this framework means that it provides consistent interpretation over different algorithms. This helps in doing a comparative analysis of how different model weights have different factors. This is very helpful when organisations need to decide between multiple different models with similar predictive accuracy.

LIME (Local Interpretable Model-agnostic Explanations) was developed by Ribeiro et al. (2016) for understand individual predictions. SHAP provides global feature importance on all predictions while LIME helps in explaining singular customer cases for high value customers analysis. However, LIME's focus on local explanations needs very high computational requirements for individual explanations which does limit its use in large-scale operational contexts where batch predictions and global insights are more helpful.

2.7 Cross-Industry Comparisons and Research Gaps

Even with all the research for individual industries, there is still very limited comparisons for cross-industries that use consistent and comprehensive methods. Most studies focus on single industries, which makes it difficult to figure out which insights require industry specific adaptation, and which can be transferred across different sectors. This is a limitation that makes it very difficult to come up with general theories for customer churn that can help with creating an analytical plan of action in different business situations.

Evidence suggests that ensemble methods are able to show more consistent performances in different industries when compared to the traditional methods. This shows ensemble methods are better for varying customer data types and business models. The performances in these methods do however vary by each sector. Telecommunications industries usually work well with XGBoost for handling the complex patterns in usage, while insurance industries benefit more from Random Forests and CatBoost as it also takes care of regulatory requirements by providing model transparency.

Some modern-day research for individual sectors shows lots of methodological advances. Saha et al. (2024) further used advanced deep learning approach with ChurnNet. They were able to get accuracy rates more than 95% by adding convolutional layers and attention mechanisms. While these advances are very valuable, it still shows how rare it is to have any improvements for cross-industry comparisons, and the emphasis is mostly on industry-specific optimization.

In a similar way, even though explainable AI techniques are used for individual churn prediction studies, the comparison of these interpretation methods among different industries and algorithms remains under-researched. This is a good opportunity to see how these techniques can be reused or transferred to help with both theoretical advancement and practical application. This is also important nowadays with the increase in emphasising of responsible AI practices and regulatory requirements and need for transparent models on customer-facing applications.

This research tries to address these gaps by comparing six different machine learning algorithms systematically on telecommunications and insurance sectors while also using consistent evaluation procedures. By using the same methodologies to both datasets from the two industries, this research provides practical evidence about algorithm transferability and also shows any sector specific patterns in customer churn behaviour. This research uses multiple metrics for evaluation and explainability analysis to give recommendations for model selection and theoretical insights on challenges for cross-industry churn prediction.

Chapter 3

3. Methodology

3.1 Research Design Overview

This research uses a Full Factorial Design to make a thorough comparison of six machine learning algorithms on telecommunications and insurance sectors. The design will be including the following six algorithms - Logistic Regression, Random Forest, XGBoost, CatBoost, Support Vector Machine, and K-Nearest Neighbors on datasets with different business contexts (telecommunications Bulgarian dataset and insurance Spanish dataset).

The factorial design helps in checking multiple different hyperparameter configurations for each algorithm. They are tested using the same preprocessing pipelines and evaluation procedures for both the datasets. Using the same methodologies helps in observing performance differences that show algorithm characteristics and industry-specific patterns, instead of just methodological artefacts. The research also uses a stratified cross-validation with five folds to give the best performance estimates and help in maintaining class distribution inside each fold. The complete pipeline uses a fixed random seed to make sure the results are always reproducible with multiple runs and help in comparing the algorithms properly.

3.2 Dataset Description

The research uses two real world datasets having different business models and complement each other well. Both datasets are taken from peer-reviewed repositories which means that they are academically credible and have high data quality. Both datasets have contrasting properties for sample size, churn rate, types of customers and different business models which give the perfect conditions for checking if algorithms can be transferred over the industries.

3.2.1 Telecom Dataset.

The telecommunications dataset is taken from a big Bulgarian telecommunications company which is publicly available on Mendeley Data (Tokmakov, 2024). The dataset has 8,454 business-to-business (B2B) records of small, medium and large business companies. There are 14 features/attributes in the dataset for each customer record like customer value segments, subscription status, revenue metrics, and usage patterns. The dataset also has one target variable "churn" which shows whether the business customer stayed with the company or decided to leave.

The dataset shows a very low churn rate of 6.49%, this is an expected pattern in B2B models where customers rarely churn because of long-term contracts or don't want to have the extra switching costs. Customer segmentation in the dataset has categories like SME (Small and Medium Enterprises), Platinum, Gold, Silver, and Lead. These show different tiers of value and relationships. Some of the key features for churn are suspended subscriber counts, average fixed and mobile revenue, total revenue, and ARPU (Average Revenue Per User). These help in giving proper insights about customer engagements and patterns in profitability.

Table 3.1 Telecommunications Dataset Characteristics

Dataset size	Number of features	Churn rate	Customer type	Source
8,454 records	14 attributes	6.49%	B2B	Tokmakov (2024)

3.2.2 Insurance Dataset.

The insurance dataset is taken from a Spanish insurance company which is also publicly available on Mendeley Data (Guillen et al., 2021). The dataset has the details of 40,284 business-to-customer (B2C) records of customers that were tracked over a duration of five years between 2010-2014. There are 18 features/attributes in the dataset for each customer record like demographic characteristics, policy details, vehicle information, claim history, and payment methods. The dataset has a target variable "retention" that tells if a customer continued with the company or not.

The insurance dataset has a comparatively larger churn rate of 23.83%, which is almost four times more than the telecommunication churn rate. This is also expected as individual customers are more likely to switch their insurance policies as their contracts end and lower switching costs from competitors. Some of the key features for churn are claim history (both motor and home insurance claims), payment methods, client seniority, insured capital values, and demographic factors such as age and gender. Since the customers are tracked over five years, we get a better understanding of long-term customer retention patterns.

Table 3.2 Insurance Dataset Characteristics

Dataset size	Number of features	Churn rate	Customer type	Source
40,284 records	18 attributes	23.83%	B2C	Guillen et al. (2021)

3.2.3 Dataset Comparison.

The difference in the characteristics of both datasets give perfect conditions for checking how well algorithms can be transferred. The telecommunications dataset has B2B relationships with low rate of churns, contract issues and are more focused on revenue and usage metrics. The insurance dataset however has B2C relationships with higher churn rate, yearly policy renewal cycles and focus more on demographic factors and claim experiences. The difference in the dataset sizes also helps us to see how well an algorithm performance can improve with higher sample size.

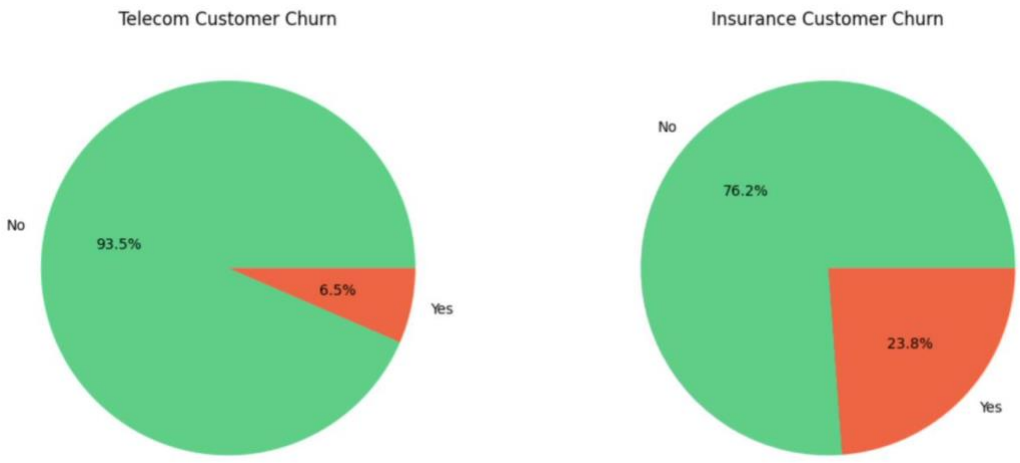


Figure: 3.1 Customer churn rates between telecommunications and insurance sectors

3.3 Exploratory Data Analysis

After performing exploratory data analysis different churn patterns could be seen in both industries that helped in deciding the preprocessing methods and feature selection to do. Since both industries show different churn dynamics it validates the cross-industry comparison approach while also showing the need for some industry-specific analytical considerations.

In telecommunications, the churn patterns seem to mostly vary on customer value segments. The high value segments including SME, Platinum, and Gold have between 8.1-8.8% churn rates and low value segments like Lead and Silver show no churn at all as in *Fig 3.2.* Looking at this, we can focus retention strategies on higher-value customers instead of entry level/newer customers. This could mean expectations of higher service quality could be the reason behind the high churn.

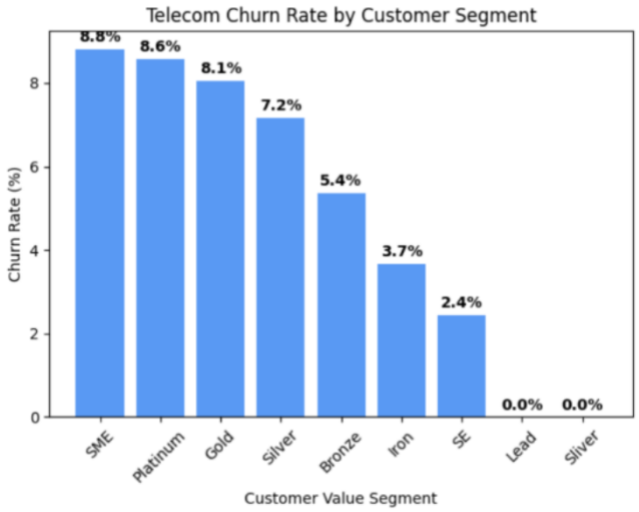


Figure: 3.2 Telecommunications churn rates by customer value segment

Insurance churn rates also seem to vary mostly on the type of claims. Customers with no claims ever show lowest churn rates of 23.4% (highest retention of 76.6%). Customers having claims for both motor and home or just motor showed churn rates of at least 41% as in Fig 3.3. This suggests that customers might not be happy after they make the claims and then end up choosing to not renew their policies and go with other competitors

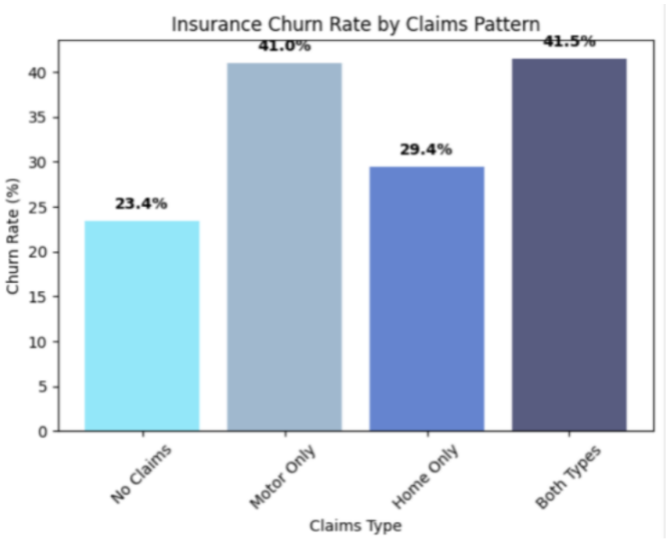


Figure: 3.3 Insurance customer churn rates by claim history

After analysing revenue in telecommunications, we can see that customers that churned have a slightly higher average ARPU of 25.00 BGN (Bulgarian Lev) while retained customers were around 24.00 BGN. Most of the customers also seem to cluster around the median of 19.31 BGN as shown in Fig. 3.4. This tells us that the customers expect higher quality services and when they are not happy, they leave even though they have high enough revenues.

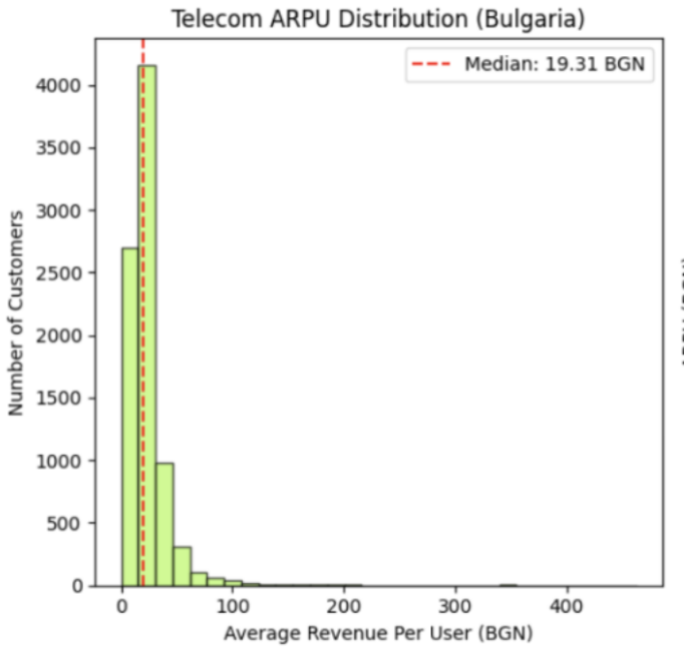


Figure: 3.4 ARPU distribution comparison of retained and churned telecom customers

After analysing the demographics in the insurance dataset, it showed that retained customers had a slightly higher average age of 60.6 years compared to churned customers having average age of 56.7 years. This slight difference in age could mean that older customers show more loyalty or maybe because of higher switching costs and considering their health or lesser policy options by competitors.

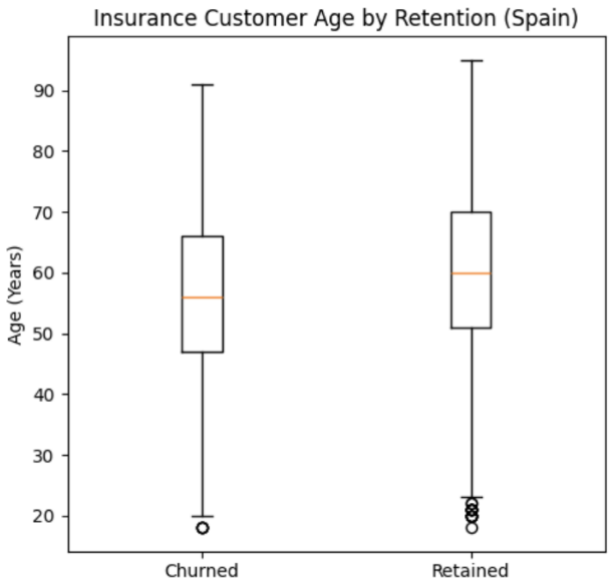


Figure: 3.5 Age distribution comparison of retained and churned insurance customers

3.4 Data Preprocessing

A thorough and careful data preprocessing helps in ensuring model reliability and helps for having fair comparison of performance of all algorithms. The preprocessing pipeline designed in this research is done so that it remains consistent for both datasets while also allowing for industry-specific data characteristics. All the preprocessing steps were done using the scikit-learn library functions (scikit-learn, 2019). This makes sure that the complete code is reproducible and follows best practices in machine learning workflows.

3.4.1 Data Cleaning.

The first step in data cleaning was to remove any duplicate entries and format column names. Any duplicate records had to be removed to prevent any types of data leakage. Column names were cleaned so there were no extra whitespaces. Target variables were also normalised to have a consistent binary format (0 = retained, 1 = churn) in both datasets to avoid any confusions.

3.4.2 Missing Value Treatment.

For handling missing values domain specific strategies were used depending on the type of the variable. Numerical variables were imputed using medians and for categorical variables mode was used. This was necessary so that we preserve dataset-specific patterns by having separate imputers for both industries making sure replacement values are relevant/appropriate.

3.4.3 Categorical Encoding.

All categorical variables were transformed using label encoding. In this each unique category is mapped to a different integer value. Any missing categorical values were also given a 'Missing' category before encoding. Both datasets used different encoders and mappings to prevent any conflicts in data.

3.4.4 Numerical Scaling.

All numerical variables were standardised using a simple StandardScaler, which transforms variables to have zero mean and unit variance. By doing this scaling it helps distance-based algorithms like KNN and SVM and normal models like Logistic Regression that can show sensitivity to differences in feature magnitudes. Both datasets have different scalers being used to again prevent any conflicts in data.

3.4.5 Handling Class Imbalance (SMOTE).

In the telecommunications dataset, we see a very big class imbalance. To prevent models being biased towards the majority class, I decided to use synthetic oversampling. The Synthetic Minority Over-Sampling Technique (SMOTE), developed by Chawla et al. (2002), helps to overcome this issue. It generates synthetic examples of the minority class by doing linear interpolation between existing minority instances and their nearest neighbors. This then helps in having equal class representation but also not having any duplicate data to prevent overfitting.

SMOTE was applied only inside the cross-validation training folds which helps in preventing any data leakage. All the training folds were applied SMOTE separately, which made sure that none of the validation folds had any synthetic data in them. The sampling strategy parameter was set to ‘auto’ so that it automatically balances the classes. This helped the algorithm to have more examples of minority class.

3.5 Feature Engineering

For selection of features, it is important to have domain knowledge about churn drivers. Both the datasets gained a lot from carefully selected feature sets in such a way that it would properly capture customer patterns while also avoiding unnecessary information or noise that may affect the models performance.

3.5.1 Telecom Feature Selection.

The telecommunications feature set focused more on customer value segmentation, subscription status, and revenue metrics. The CRM (Customer relationship management) value segments and effective segments divided the customers by the size of business and their relationship with the business which helps find out which customers are most important. The subscription metrics included active, inactive, and suspended subscriber counts which give indicators of engagement levels and how healthy an account is. Finally, to find out the customers monetary value and spending patterns, revenue variables like mobile, fixed, and total revenue streams, and ARPU calculations were included. These features combine both the customers lifecycle and contribution financially which helps the model to identify the most at-risk customers that need more attention.

3.5.2 Insurance Feature Selection.

The insurance feature set had demographic characteristics, policy attributes, vehicle information, and claims history. Demographic variables included age and gender that help in figuring out life factors that may influence a customer's loyalty. Policy-specific features like payment methods, insured capital values, and policy counts show types of interaction customers have with the business. Claims type (motor, home or both) and frequency of claims are also important as they showed how important they are for prediction revealed when exploratory

analysis was done. Client seniority helps in measuring how long a customer was with the company which is important for finding patterns in customer experience over time.

3.6 Model Development

3.6.1 Algorithm Selection.

For this research, six algorithms were selected for a thorough performance comparison. Logistic regression gives us a baseline performance and provides good value for its simplicity despite some limitations with complex patterns (Hosmer and Lemeshow, 2013). SVM uses a maximum margin principle for classification and kernel methods that give non-linear decision boundaries (Vapnik, 1998). KNN uses instance-based learning where it classifies based on how similar it is to training examples.

Random Forests is an ensemble learning method that combines multiple decision trees to improve accuracy of predictions and reduce overfitting (Breiman, 2001). XGBoost uses gradient boosting and regularization which improves model performance by sequential error correction (Chen and Guestrin, 2016). CatBoost provides an advanced gradient boosting method that has inbuilt categorical variable handling which gives more advantage to it as there are lots of categorical features in both datasets (Prokhorenkova et al., 2018). This algorithm selection contains traditional statistical methods, instance-based approach and newer ensemble techniques which help in a thorough assessment of which types of algorithms are best for cross-industry churn prediction.

3.6.2 Pipeline Architecture.

Scikit-learn was used as the main library for the pipeline implementation, including data preprocessing steps and model fitting. This design helps in preventing any kind of data leakage by making sure all preprocessed data is used only on training data before applying on validation sets. For any algorithms that need probability or any special handling, wrapper classes were made to maintain consistency and handle any algorithm specific requirements. SVM needed to use a CalibratedClassifierCV wrapper to make sure that it would output probability estimates for threshold optimization. The complete pipeline is made so that the results can be easily reproduced and simplifies hyperparameter optimization.

3.6.3 Hyperparameter Optimization.

For hyperparameter optimization, GridSearchCV was used with stratified cross-validation so that all parameter combinations can be explored and while maintaining good performance. The hyperparameter ranges were selected based on algorithm characteristics and some experimentations to make sure proper coverage of optimal configurations.

Table 3.3 Hyperparameter Search Grids for Algorithm Optimization

Algorithm	Hyperparameter	Values Explored
Logistic Regression	Regularisation Strength (C)	[0.01, 0.1, 1, 10]
Random Forest	Number of Trees (n_estimators) Maximum Depth (max_depth)	[100, 200] [6, 12, None]
XGBoost	Number of Trees (n_estimators) Tree Depth (max_depth) Learning Rate Positive Class Weight (scale_pos_weight)	[100, 200] [3, 6] [0.01, 0.05, 0.1] [1, 5, 10]
CatBoost	Iteration Count (iterations) Tree Depth (depth)	[150, 300] [4, 6]
SVM	Regularisation Strength (C)	[0.01, 0.1, 1, 10]
KNN	Number of Neighbors (n_neighbors) Weighting Scheme (weights)	[3, 5, 7] ['uniform', 'distance']

3.7 Model Training and Validation

3.7.1 Train-Test Split Strategy.

For the train-test split, 80/20 split was decided. The 80% training data provides good enough data for proper model fitting and cross-validation. The 20% test set also provides enough sample size for decent performance estimation. Random seed of 42 was used everywhere to make sure results were reproducible and direct performance comparisons between algorithms

3.7.2 Cross-Validation Approach.

Stratified 5-fold cross-validation is used in this research to give proper performance estimates during hyperparameter optimization. Each of the folds maintained the class distribution proportions so that minority class is represented the same in all iterations. SMOTE was also applied independently inside each training fold to prevent any synthetic data from leaking into the validation set. This careful implementation of cross-validation makes sure that the performance metrics that we get are the actual capabilities of the model and not just due to specific data split or preprocessing choices.

3.7.3 Threshold Optimization.

Classification thresholds were optimized so that we can get the best F1-score on validation set to get balanced precision and recall. The default 0.5 probability thresholds usually don’t give good results for imbalanced datasets where minority class predictions would benefit from adjusted decision boundaries. For optimising the thresholds the values would go from 0.01 to 0.99 in increments of 0.01 and selecting the value with the maximum F1-score.

3.8 Model Explainability Framework

SHAP (SHapley Additive exPlanations) is used for interpreting model outputs. It is a game-theoretic framework that provides accurate feature attribution for machine learning predictions (Lundberg and Lee, 2017). SHAP values quantify all feature's contribution to every individual prediction by Shapley values from cooperative game theory. This makes sure that all feature importance estimates are consistent and locally accurate. This framework has a model-agnostic nature which gives consistent interpretation for all different algorithms, which helps for a proper comparative analysis of how different models weight different features.

SHAP analysis can generate both global feature importance and individual prediction explanations. For global importance, it aggregates all SHAP values for all predictions and identifies which features have the largest impact on model outputs. In this research, TreeExplainer is used for tree-based models and LinearExplainer for the remaining algorithms.

3.9 Evaluation Metrics

3.9.1 Classification Metrics.

For the model evaluation, multiple metrics were used. Accuracy measures the overall correct classification percentage. Precision calculates how often the model makes a correct prediction while Recall calculates how many of the actual churned customers did the model predict correctly. F1-score is the harmonic mean of Precision and Recall which is good for getting balanced performance measures for imbalanced datasets. ROC-AUC (Area Under the Receiver Operating Characteristic Curve) calculates performance of the model on all possible decision thresholds. This measures discriminative capability independent of specific threshold choices (Fawcett, 2006).

3.9.2 Model Selection Criteria.

The primary model selection was decided by F1-score and ROC-AUC so that we can have a balanced performing model and threshold-independent discriminative capability. F1-score was optimized during hyperparameter tuning so that models would have a good enough precision-recall balance for deployment. All four metrics were used for the final model selection to see if superior performance was due to actual algorithmic advantages or just lucky threshold effects.

Chapter 4

4. Research Design and Implementation

This chapter describes how the methodology from Chapter 3 was implemented. The complete implementation includes how the preprocessing pipeline was developed, model training architecture, how explainability framework was integrated and the creation of the web application. All code was developed completely in Python 3.13.3 with the help of industry-standard libraries like scikit-learn, pandas, NumPy, Streamlit and more. The code has been designed so that it can be reproduced and maintained easily. Complete source code is provided in **Appendix A.**

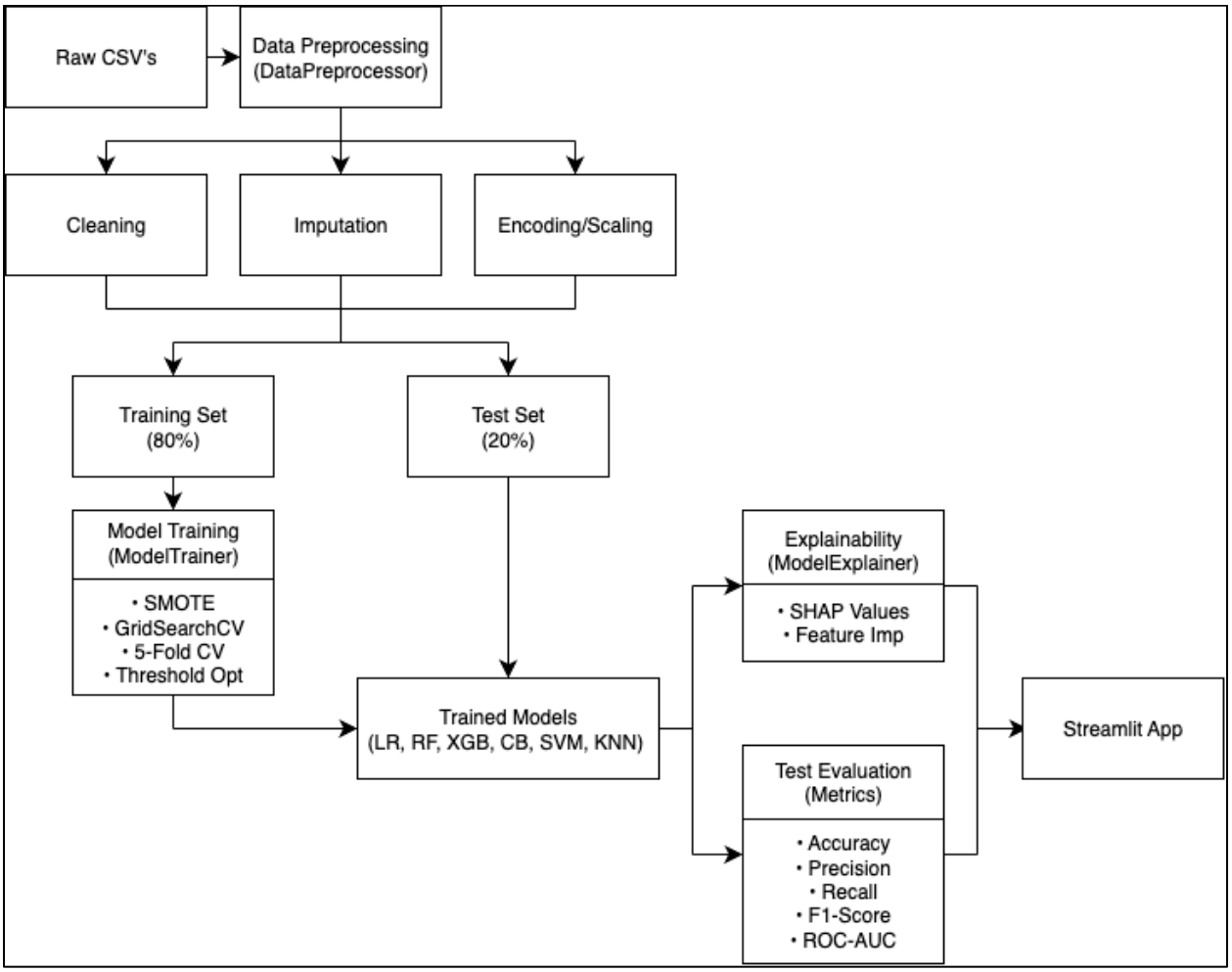


Figure: 4.1 Implementation Pipeline Architecture

4.1 Data Loading

The data loading methods are kept in the DataPreprocessor class which reads the csv files for the telecommunication and insurance datasets using pandas. The insurance dataset had the target variable as “retained” which was reversed to create a new target column “churn” (retained= ‘yes’ → churn=0, retained= ‘no’ → churn=1). This was done so we have the same target variable interpretation for both industries. The method automatically detects target column names over different variations and normalize the target variable and map it to binary format (1 for churned, 0 for retained). Any non-standard or missing values in target were coerced to 0 using the Pandas function ‘to_numeric’ to prevent any processing failures.

4.2 Data Preprocessing Implementation

4.2.1 DataPreprocessor Class Architecture.

The preprocessing pipeline was kept inside a DataPreprocessor class. It is designed so that it does the same preprocessing and considers any industry-specific transformation requirements. The class mainly follows the scikit-learn transformer pattern. In this we implement ‘fit’ and ‘transform’ methods and that help in proper train-test splitting without any data leakage.

The class has two separate loading methods for the datasets to avoid any conflict and stores all the fitted encoders, scalars and imputers with identifiers describing which dataset they are for. This was very important so that the preprocessing transformations learn only from one industry and do not contaminate the other. The class also helps in tracking categorical and numerical columns dynamically so that it adapts to each datasets feature set automatically without any manual help.

4.2.2 Preprocessing Pipeline Execution.

The preprocessing pipeline executes one-by-one starting with data loading, data cleaning, imputing missing values, categorical encoding, and feature scaling. Data cleaning step removed any duplicate records and column names were standardized. For the missing values scikit-learn has a SimpleImputer method using which the numerical features were imputed using median and categorical features were imputed using most-frequent as mentioned in Section 3.4.2

For categorical encoding, label encoding was done by mapping each unique category to an integer with “Missing” encoding for missing values. Feature scaling used StandardScaler for numerical columns to have zero mean and unit variance. The class saves separate scalers for each dataset for reusing.

4.3 Model Training Implementation

4.3.1 Model Trainer Class Design.

The model training pipeline was kept inside a ModelTrainer class. It handles the instantiating of algorithms, training with train set, hyperparameter optimization, cross-validation, and performance evaluation. The class was designed to allow flexibility and adding more code for any new algorithms without much modification of any configurations or restructuring the core training logic. This approach also helped in faster experimentation while maintaining the code quality for deployment.

This class was mainly done using scikit-learn methods combined with SMOTE oversampling with algorithm-specific models. It also made sure that SMOTE happened only inside the cross-validation training folds and not before to prevent any data leakage of synthetic minority classes. For each of the algorithms, the class would instantiate the appropriate model objects and parameters, which was then further improved by using grid search.

4.3.2 Training Pipeline for Telecom and Insurance.

The pipeline was executed separately for telecommunications and insurance to maintain independence of industries. Both pipelines used 80/20 stratified split to keep the class distribution the same for both splits and have sufficient data for developing the model and estimating performance.

For all of the six algorithms, the training pipeline also goes through hyperparameter optimization using GridSearchCV with 5-fold stratified cross-validation. Inside each fold, SMOTE was applied only on the training set and not on the validation set. The grid search helped to verify all combinations of parameters that are specified and select the most optimal values by maximizing cross-validated F1-score.

4.3.3 Grid Search Implementation.

GridSeachCV is used for conducting a complete exhaustive search in the hyperparameter space that is defined in the config.py file. For each of the algorithm and dataset combinations, the method fits all of the candidate models. It then records the cross-validation scores for every combination and stores the best hyperparameter configuration.

After hyperparameter optimization is done, the class then does threshold optimization on the validation set to further improve the F1-score. For this process the model goes through decision thresholds from 0.01 to 0.99 in increments of 0.01 and selects the value that gives the best precision-recall balance depending on each dataset's specific class distribution.

The final model evaluation used the optimized models with the tuned thresholds on the test sets to generate definitive performance metrics for cross-industry comparison. Models were then serialized using Python's joblib library to be used later for explainability and prediction on the web interface. Training execution times were also recorded to check computational efficiency for all algorithms and datasets.

4.4 Model Explainability Implementation

4.4.1 SHAP Analysis Setup.

The model interpretability pipeline was kept inside a ModelExplainer class for all SHAP related methods. The method first checks the type of algorithm used in the model and uses the appropriate explainer object - TreeExplainer for tree-based models (Random Forest, XGBoost, CatBoost), LinearExplainer for Logistic Regression, and KernelExplainer for remaining algorithms (SVM, KNN). This algorithm-specific explainer helps in optimizing computational efficiency and provides the best SHAP computations.

The SHAP values are computed for all test set predictions to calculate combinations to individual churn predictions of each feature. Since the insurance dataset had lots of values, the analysis used only a representative subset. This helps in reducing computation time but also maintains sufficient statistical power for good and reliable feature importance estimation. The SHAP values were computed once and then saved for future visualization generation.

4.4.2 Feature Importance Visualization.

The SHAP visualizations were generated to better understand feature importance patterns for non-technical stakeholders. Bar plots aggregated the SHAP values for all the predictions and then ranking the features by the average impact magnitude to help identify the most influential churn predictors. This visualization can help in validating domain knowledge about customer behaviour drivers in the industries. The visualization is then exported and saved as a PNG to be used for the web application.

4.5 Streamlit Web Application

4.5.1 Application Architecture.

A web-based demonstration application was made using the Streamlit framework to provide users with a good interactive way to access the trained models and help with real time churn prediction. The methods for this interface are kept inside a ChurnPredictionApp class to help with model loading, processing user inputs, generating predictions, and visualize the results.

The application starts off with loading the trained models and caching it so that there is less response time when users interact with them. All metrics, images, and SHAP plots were also loaded along with the data preprocessors to help with transforming the data input by users.

4.5.2 User Interface Design.

The prediction UI consisted mainly of three functions – industry selection, a form for inputting the customer data, and predicting the churn. Users can select which industry they want to get predictions for, and it dynamically changes input form fields depending on the industry. The input form fields have appropriate widgets, dropdowns for categorical variables, and numerical inputs for continuous variables. These help in making sure the correct type of data is used for prediction, and there are no errors.

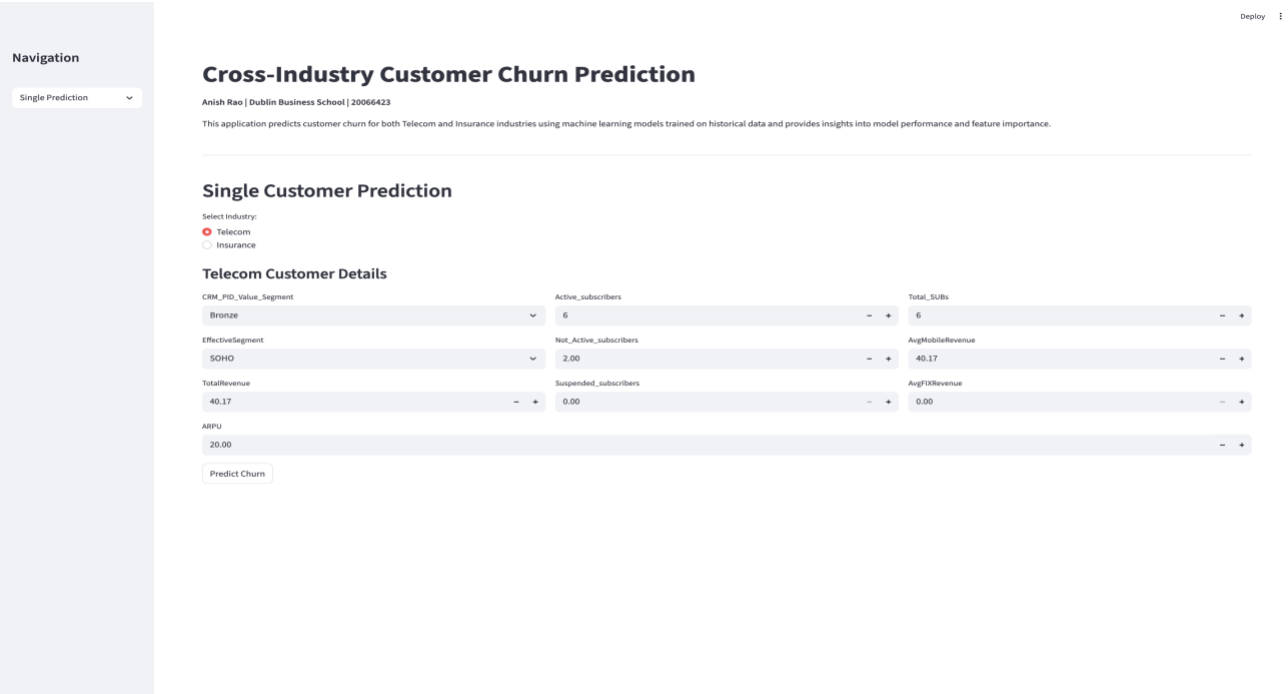


Figure: 4.2 Single Prediction interface

A similar batch prediction UI was also made which takes in a csv file and shows an error if the features required for the specific industry prediction don't match with the csv file uploaded to ensure no issues with predictions. A small preview of the user's uploaded file is also displayed so they can see if they uploaded the correct file. Users can also download the prediction results in csv format.

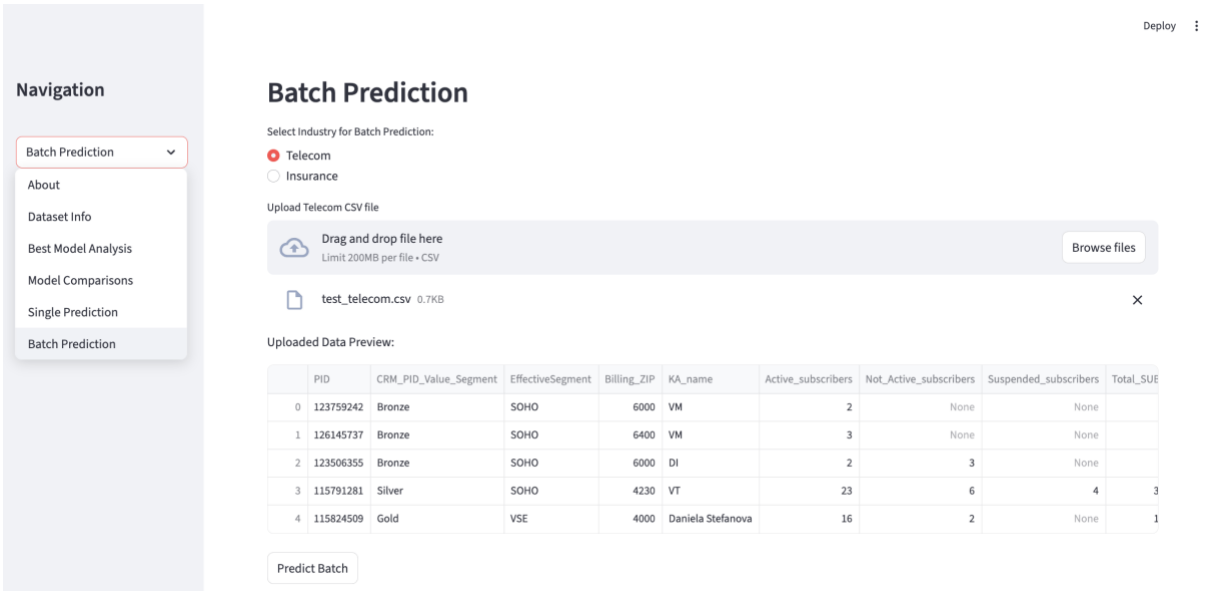


Figure: 4.3 Batch Prediction interface

4.5.3 Model Performance Comparison.

The model comparison UI helps stakeholders and developers to evaluate all algorithmic performances for both industries. The interface has two complimentary visualization modes. The first one is a side-by-side metric comparison for selected algorithms. Users can select any algorithm and bar graphs generated dynamically for all four metrics.

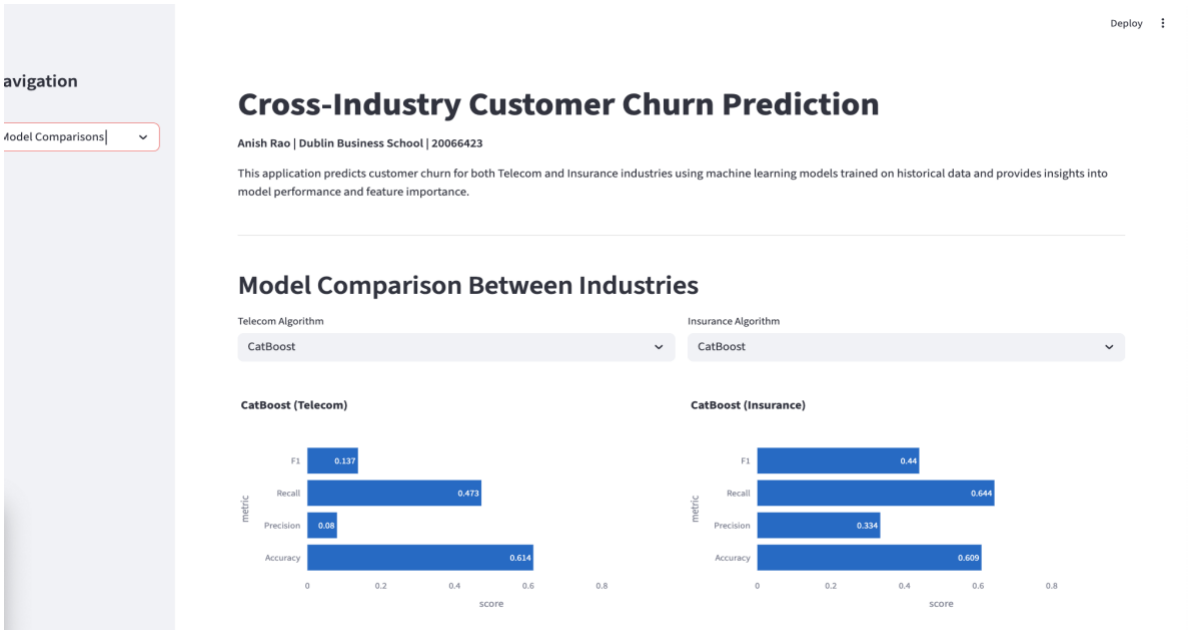
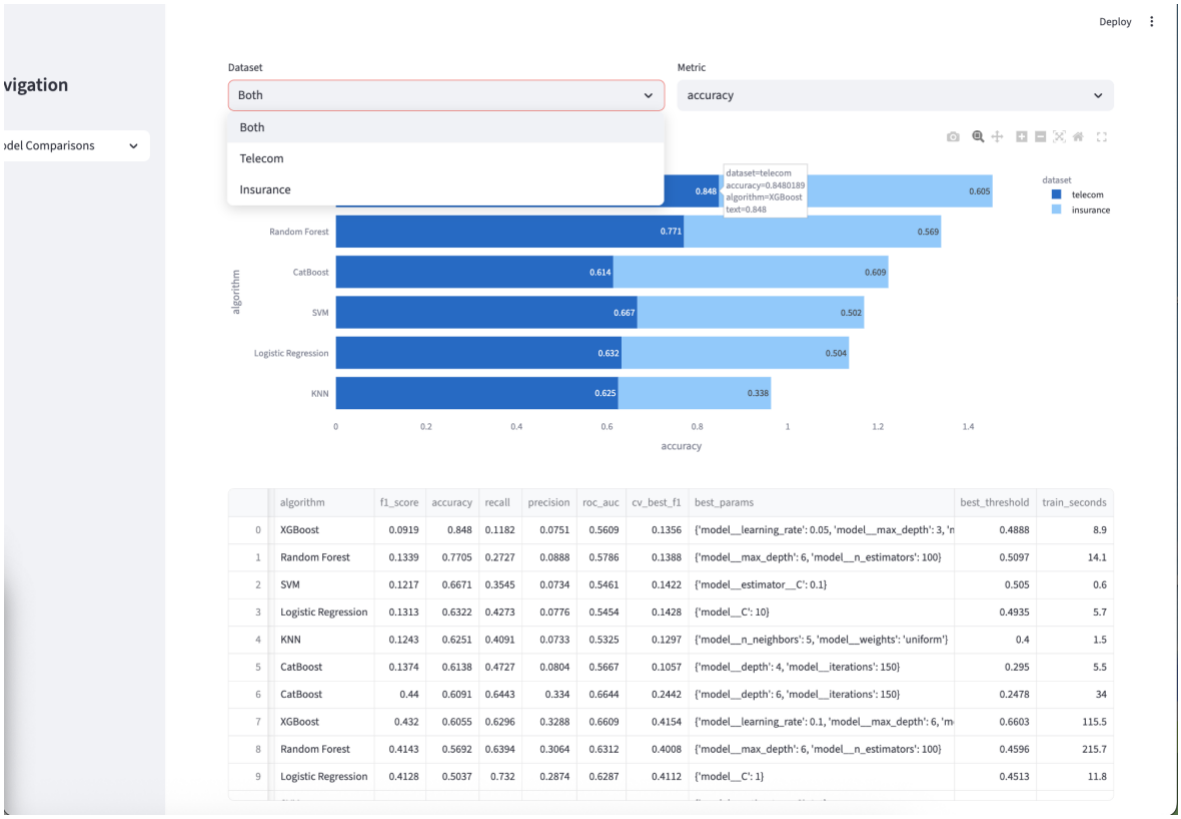


Figure: 4.4 Algorithm performance comparisons

The comprehensive comparison view has a tabular view of performance data metrics for all six algorithms which changes dynamically based on the chosen dataset (telecom, insurance or both). Grouped horizontal bar charts are also generated for all six algorithms vs the chosen metric (accuracy, precision, recall, F1-score or roc_auc)



	algorithm	f1_score	accuracy	recall	precision	roc_auc	cv_best_f1	best_params	best_threshold	train_seconds
0	XGBoost	0.0919	0.848	0.1182	0.0751	0.5609	0.1356	{'model__learning_rate': 0.05, 'model__max_depth': 3, 'n	0.4888	8.9
1	Random Forest	0.1339	0.7705	0.2727	0.0888	0.5786	0.1388	{'model__max_depth': 6, 'model__n_estimators': 100}	0.5097	14.1
2	SVM	0.1217	0.6671	0.3545	0.0734	0.5461	0.1422	{'model__estimator__C': 0.1}	0.505	0.6
3	Logistic Regression	0.1313	0.6322	0.4273	0.0776	0.5454	0.1428	{'model__C': 10}	0.4935	5.7
4	KNN	0.1243	0.6251	0.4091	0.0733	0.5325	0.1297	{'model__n_neighbors': 5, 'model__weights': 'uniform'}	0.4	1.5
5	CatBoost	0.1374	0.6138	0.4727	0.0804	0.5667	0.1057	{'model__depth': 4, 'model__iterations': 150}	0.295	5.5
6	CatBoost	0.44	0.6091	0.6443	0.334	0.6644	0.2442	{'model__depth': 6, 'model__iterations': 150}	0.2478	34
7	XGBoost	0.432	0.6055	0.6296	0.3288	0.6609	0.4154	{'model__learning_rate': 0.1, 'model__max_depth': 6, 'm	0.6603	115.5
8	Random Forest	0.4143	0.5692	0.6394	0.3064	0.6312	0.4008	{'model__max_depth': 6, 'model__n_estimators': 100}	0.4596	215.7
9	Logistic Regression	0.4128	0.5037	0.732	0.2874	0.6287	0.4112	{'model__C': 1}	0.4513	11.8

Figure: 4.5 Comprehensive metric and algorithm comparison

4.5.4 Deployment Configuration.

The application has inbuilt functionality by Streamlit for straightforward deployment to many cloud platforms like AWS, Streamlit cloud or Heroku. For dependency management, a requirements.txt file is kept so there are no issues with missing libraries/dependencies. The application handles any errors, missing files or invalid user inputs for smooth functioning and avoids any crashes.

Chapter 5

5. Results and Analysis

This chapter presents all the results from training and evaluating the six machine learning algorithms on telecommunications and insurance datasets. The performance for each model is recorded using five metrics - accuracy, precision, recall, F1-score, and ROC-AUC. The results show a lot of differences between the different algorithms and datasets. The ensemble methods showed consistently better results than the traditional approaches. The analysis examines the algorithm-specific performance and feature importance insights from SHAP explainability.

5.1 Telecom Model Results

CatBoost had the highest F1-score of 0.14 on telecommunications data, with accuracy of 61%, precision of 8%, recall of 47%, and ROC-AUC of 0.567. It also had an optimized decision threshold of 0.29 to improve F1-score because of high class imbalance. Random Forest (RF) and Logistic Regression (LR) had closer F1-score of 0.13 with RF having much higher accuracy of 77% but much lower recall than CatBoost and LR. SVM and KNN produced decent results which were closer all round to CatBoost. XGBoost had the highest accuracy of 85% but lowest recall (12%) resulting in a very low F1-score of 0.09.

Table 5.1 Telecommunications Model Performance

Algorithm	Accuracy	Precision	Recall	F1-score	ROC-AUC	Threshold
CatBoost	0.61	0.08	0.47	0.14	0.57	0.29
Random Forest	0.77	0.09	0.27	0.13	0.58	0.51
Logistic Regression	0.63	0.08	0.43	0.13	0.54	0.49
KNN	0.62	0.07	0.41	0.12	0.53	0.4
SVM	0.67	0.07	0.35	0.12	0.55	0.5
XGBoost	0.85	0.07	0.12	0.09	0.56	0.49

These results show how the challenge of minority class prediction can affect the models. Thresholds ranged from 0.29 to 0.51 to get good results. Even with SMOTE, the model's precision stayed below 9%. The Cross-validation F1-scores during hyperparameter optimization ranged from 0.106 to 0.143 showing very less improvement by grid search. Training times ranged from 0.6 seconds (SVM) to 14.1 seconds (Random Forest).

5.2 Insurance Model Results

CatBoost again had the highest F1-score of 0.44 on the insurance data, which is more than three times improvement from telecommunication data with similar accuracy of 61%, precision of 33%, recall of 64% and ROC-AUC of 0.664. XGBoost performed better with a closer F1-score of 0.43 and accuracy of 60% and similar results with CatBoost for other metrics. Logistic regression and SVM had almost the same performances. KNN got the highest recall (89%) but had the lowest accuracy (34%).

Table 5.2 Insurance Model Performance

Algorithm	Accuracy	Precision	Recall	F1-score	ROC-AUC	Threshold
CatBoost	0.61	0.33	0.64	0.44	0.66	0.25
Random Forest	0.57	0.31	0.64	0.41	0.63	0.46
Logistic Regression	0.50	0.29	0.73	0.41	0.63	0.45
KNN	0.34	0.25	0.89	0.39	0.57	0.14
SVM	0.50	0.29	0.73	0.41	0.63	0.45
XGBoost	0.60	0.33	0.63	0.43	0.66	0.66

All algorithms show much better precision, and very high recalls showing significant improvements from telecommunication data and having at least 0.39 F1-score. The higher churn rate of 23.83% means more minority classes to learn from. The Cross-validation F1-scores during hyperparameter optimization ranged from 0.244 to 0.415 meaning that grid search was more effective to improve the results. Training times increased a lot ranging from 11 seconds (SVM) to 215.7 seconds (Random Forest) which was expected due to larger sample size.

5.3 Cross-Industry Algorithm Comparison

When comparing the results from both industries, we can see that CatBoost had the highest F1-scores of 0.13 (telecommunications) and 0.44 (insurance). XGBoost ranked second for insurance (0.43) but was the lowest in telecommunications (0.09). This shows sensitivity of XGBoost to data characteristics. Random Forest was second for telecommunications (0.13) and third for insurance (0.41) suggesting it is a good alternative for transferability. The traditional algorithms (Logistic Regression, SVM, and KNN) had decent performances for both industries.

The performance gap between industries was a lot. Insurance had approximately three times more F1-score than telecommunications. This could be due to combined influence of class imbalance severity, dataset size, and industry-specific predictability patterns. Threshold optimization strategies ranged between 0.29 - 0.51 compared to insurance having slightly lower and higher boundary ranges for insurance between 0.14 - 0.66.

5.4 Explainability Analysis

5.4.1 SHAP Feature Importance – Telecom.

SHAP results showed that *Total_SUBs* was the most influential telecommunication churn predictor by contributing 23% of the feature importance. *Not_Active_subscribers* was second with 22%, followed by *Active_subscribers* with 17%. The revenue-based features had decent prediction power with *AvgMobileRevenu*e (13%), *ARPU* (13%), and *TotalRevenue* (10%) having a total of 36%. Customer segmentation variables had very low contribution showing low relevance for churn prediction in this B2B telecommunications context.

Table 5.3 Telecommunications Top 10 SHAP Features

Feature	SHAP Importance
Total_SUBs	0.23
Not_Active_subscribers	0.22
Active_subscribers	0.17
AvgMobileRevenue	0.13
ARPU	0.13
TotalRevenue	0.10
EffectiveSegment	0.07
CRM_PID_Value_Segment	0.06
AvgFIXRevenue	0.02
Suspended_subscribers	0.001

5.4.2 SHAP Feature Importance – Insurance.

Insurance SHAP results showed that *age_of_car_M* was the most dominant feature with 36% importance. *Age_client* came in second with 25% contribution followed by *Car_power_M* with 16%. *Client_Seniority* (14%) and *num_policiesC* (11%) show that customer relationships with a company impact churn a lot. Other temporal and geographic features like *year* (10%), *gender* (5%) and *metro_code* (3%) provided contextual predictive ability. It is worth noting that claims-related variables showed the least importance with *Claims1* only having 1.7% and *Claims2* with 0.2% contribution.

Table 5.4 Insurance Top 10 SHAP Features

Feature	SHAP Importance
age_of_car_M	0.36
Age_client	0.25
Car_power_M	0.16
Client_Seniority	0.14
num_policiesC	0.11
year	0.10
Insuredcapital_continent_re	0.08
Insuredcapital_content_re	0.07
gender	0.05
metro_code	0.03

5.4.3 Cross-Industry Feature Comparison.

The patterns in feature importance diverge a lot between the two industries. Telecommunications had around 62% prediction power with subscription related features (*Total_SUBs, Active_subscribers, Not_Active_subscribers*) with revenue features providing 36%. Insurance dataset had 36% in just the vehicle age. Other demographic features like *Age_client* (25%), *Car_power* (16%) dominated along with *age_of_car_M*.

The importance distribution seems to be more balanced in telecommunications, with the top three features ranging between 17-23% while insurance had only one feature dominating at 36%. Geographic and segmentation variables were showing very low importance in both the datasets and most of the categorical features below 10% individual contribution.

5.5 Streamlit Application Demonstration

The Streamlit application successfully loaded and was able to perform predictions properly. For the demonstration of single customer prediction, the default values for the input data form in the insurance dataset were used as in Fig. 5.1. After clicking on the predict churn button, the results are displayed, in this case Churn = No with 64.21% confidence along with a bar chart displaying the probabilities for both churn and retention. This probability-based output helps in better risk assessment than just binary results, which is helpful for customer retention strategies.

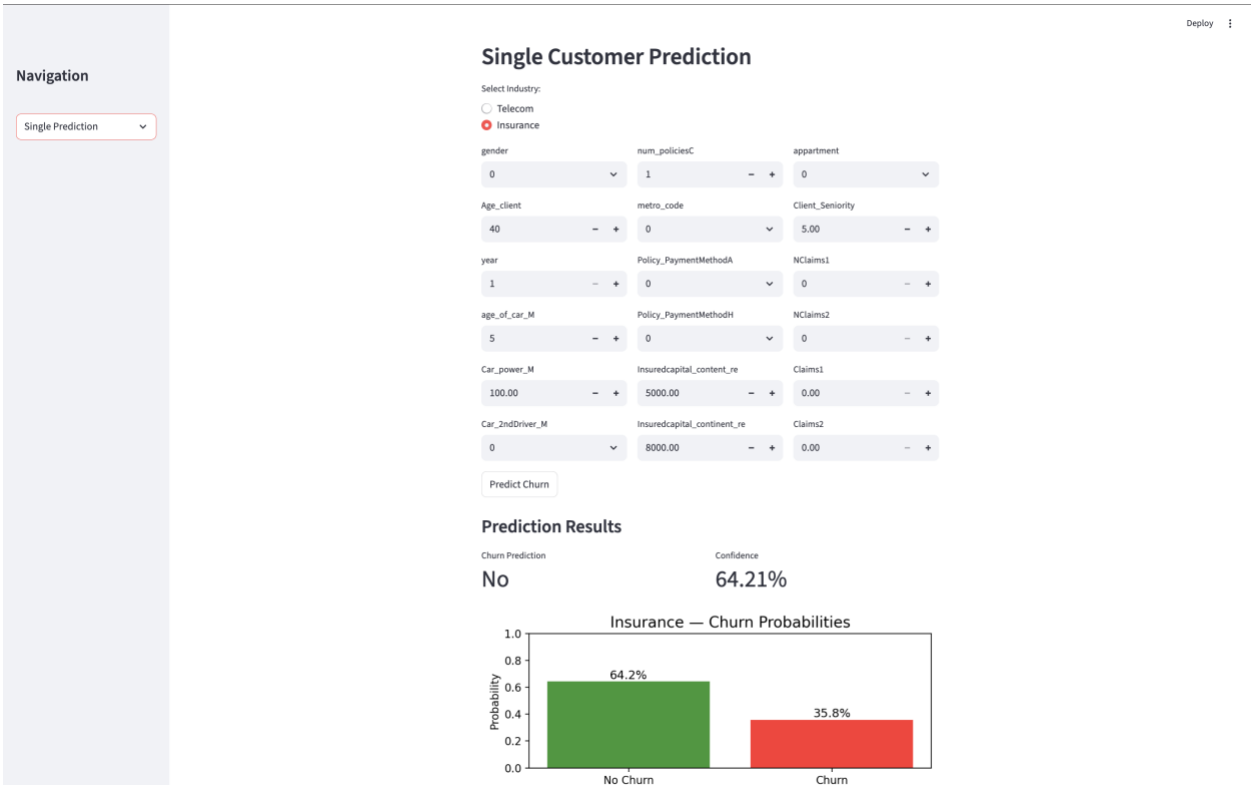


Figure: 5.1 Single prediction results

For the demonstration of batch customer prediction, a sample csv file with few rows of data for telecommunications model was used. When a user uploads a file, a small preview is displayed so the user can verify the correct dataset is used as in Fig. 5.2. After clicking on the predict churn button, the results are displayed with probability scores ranging from 4% to 64%. The final prediction distribution was three retentions (churn prediction = 0) and five churn predictions (churn prediction = 1). Users also have an option to download the results showing which customers churned and which not and providing similar probability-based results as the single predictions interface.

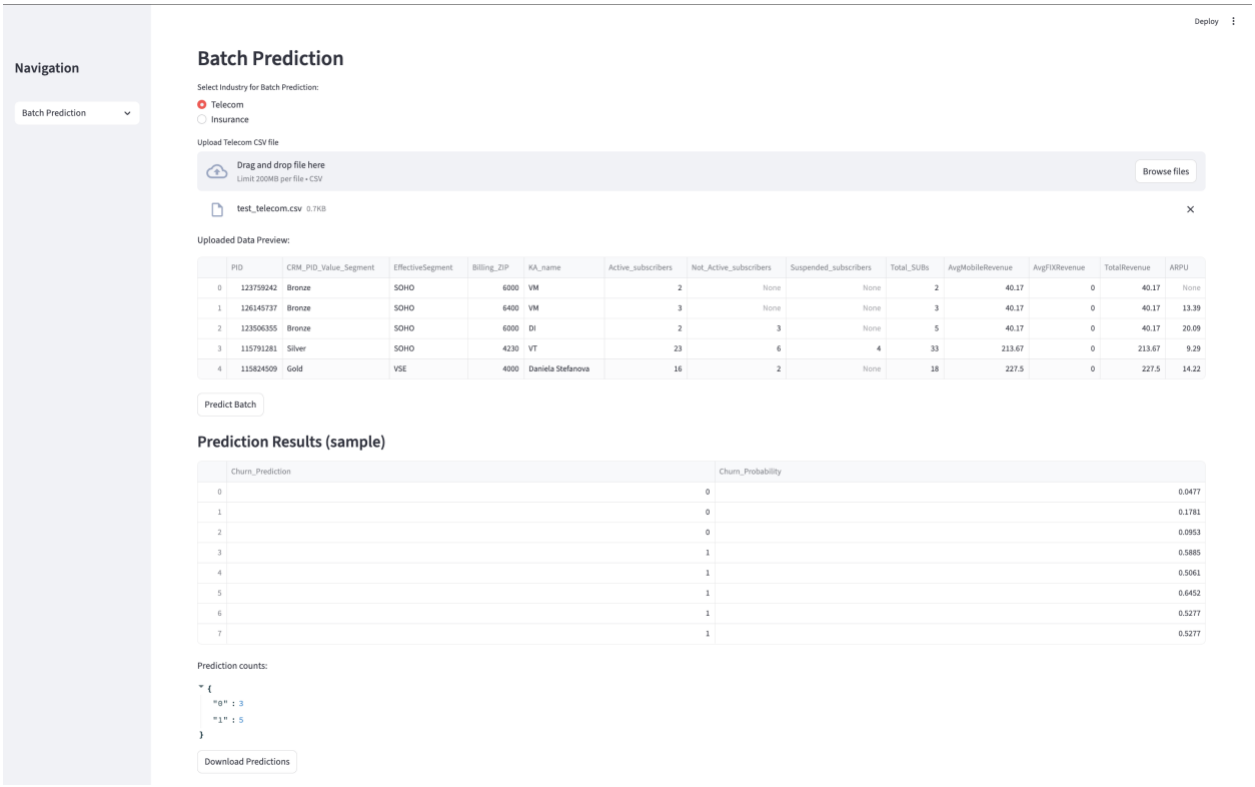


Figure: 5.2 Batch prediction results

The application outputs show the practical utility of the models trained earlier for deployment. The continuous probability scores help in better risk assessment than simple binary classifications that may support in deciding intervention strategies. The batch processing functionality helps with scalability for businesses that may want to periodically assess their customer base, with downloadable csv files.

Chapter 6

6. Discussion

6.1 Overview of Findings

The results show very big performance differences between algorithms and industries. CatBoost was the best performing in both telecommunications (F1=0.14) and insurance (F1=0.44). Ensemble methods would also constantly outperform the traditional methods. The insurance models had approximately three times more F1-score than telecommunications. This shows the influence of good data characteristics and problem of class imbalance. This chapter will interpret the findings and what they mean for practical deployment and theoretical understanding of cross-industry churn prediction.

6.2 Algorithm Performance Interpretation

6.2.1 Telecommunications Performance Analysis.

The telecommunications F1-scores ranged from 0.09 to 0.14, showing the importance of good sample size and challenge of class imbalance. The small dataset of just 8,454 records created difficulty for even the best algorithms. SMOTE oversampling showed very little improvement when class representation is severely low. The synthetic examples could not properly capture the actual real churn patterns when the original examples are so less.

Given these issues, the F1-score of 0.14 by CatBoost is not too bad. Datasets with less than 10% minority class have consistently shown to have very low precision similar as in our case. Suggesting that the results are more realistic and not due to model failure.

In the case of XGBoost we can see how important it is to have more than one evaluation metric. Even though it had an accuracy of 85%, the model had the lowest F1-score of 0.09. This could mean the model has no true positives and lots of true negatives which is not really ideal. Thus, showing higher accuracy doesn’t always mean the model is good for predicting churn.

6.2.2 Telecommunications Performance Analysis.

The insurance models showed much better performance with F1-scores ranging from 0.39 to 0.44. This improvement in telecommunications can be because of multiple factors. There was a higher churn rate of 23.83% meaning the model had more examples to learn from and given the much larger dataset size also meant that these models had over thirty times more samples to learn from.

This dataset characteristic also made SMOTE implementation much better as it had more examples to create synthetic ones and could capture churn patterns more easily than telecommunications. The combinations of higher churn rate and larger dataset size allowed SMOTE to create better quality artificial examples resulting in better performances

6.2.3 Algorithm Transferability.

CatBoost had consistent performance on both industries which shows its robustness in different business contexts. This is expected as the inbuilt ordered boosting helps in reducing overfitting which is one of the biggest problems with gradient boosting. The ability to handle categorical features natively lets it adapt to data easily without much processing. This robustness makes CatBoost a safer choice over other algorithms for deployment.

XGBoost showed very high sensitivity to the class distribution. It performed well on insurance data (F1=0.43) where there was not much imbalance but failed on telecommunications (F1=0.09) where there was much higher class imbalance. This suggests that XGBoost suffered a lot from overfitting and will need more careful dataset considerations before deployment.

Traditional algorithms (Logistic Regression, SVM, KNN) had mediocre performances but were consistent in both industries. This tells us that simpler models can struggle to capture complex churn patterns.

6.3 Feature Importance Insights

6.3.1 Telecommunications Feature Patterns.

SHAP analysis showed telecommunications churn is affected by operational metrics related to subscriptions (*Total_SUBs, Active_subscribers, Not_Active_subscribers*) having 62% influence altogether. Most of rest of the weightage came from revenue metrics (*ARPU, mobile revenue, total revenue*) with 36% importance. This focus on operational metrics is expected with a B2B relationship management, where service usage and revenue generation directly relate to customer satisfaction and engagement levels.

6.3.2 Insurance Feature Patterns.

Insurance churn prediction was mostly dominated by vehicle characteristics with just *age_of_car_M* contributing 36%. Other factors like *Age_client* (25%), *Car_powe*r (16%) were the bulk of remaining factors. This matches with how insurance companies do their risk assessment.

The very low importance of claims variables (*Claims1* = **1.7%**, *Claims2* = **0.2%**) is contradictory with what the exploratory analysis showed where higher churn rates were seen with customers having claims meaning there is some correlation with the variables. Claims usually correlate with vehicle age, older vehicles have higher claims and higher non-renewal rates. SHAP analysis showing the actual predictive power of the vehicle age rather than the claims tells us how important it is to have global model-based feature importance rather than local univariate correlation analysis for identifying the actual underlying churn drivers.

6.3.3 Cross Industry Feature Comparison.

The feature patterns confirm some fundamental differences in churn drivers between B2B (telecommunications) and B2C (insurance) business models. Telecommunications focus more on operational metrics that business can actively manage by improving services or adjusting prices. Insurance depends more on demographics and characteristics of the asset that may need segmentation-based strategies instead of normal service interventions.

6.4 Business Implications and Recommendations

Based on these results, CatBoost is recommended for both industries because of the higher F1-scores and consistent performance in both industries. If organisations care more about explainability over performance, then Random Forest can be a good alternative that gives comparable results with more informative decision tree visualisations.

Threshold optimization was also important for deployment. Telecommunications models benefited from lower thresholds (0.29-0.51) and insurance models had moderate thresholds (0.14-0.66) for better balance. These threshold strategies may need to be adjusted based on specific business costs of false positives (wasted retention efforts) versus false negatives (lost customers). For operational retention strategies, telecommunications should monitor the subscription status and ARPU trends. They can reach out to customers with declining revenue patterns or if inactive subscribers increase. Insurance providers should focus on age-based risk segmentation, targeting older vehicles for any renewal policies and maybe introducing loyalty programmes for long-term clients.

6.5 Limitations

The research has lots of limitations that impact the generalizability. The very small size and extreme class imbalance of the telecommunications dataset affect the performance of models and may not actually represent broader B2B contexts. The geographic specificity (Bulgaria telecommunications, Spain insurance) also limits the transferability to other regions that may have different regulatory requirements or customer behaviours.

Effectiveness of SMOTE is still uncertain for extreme imbalance cases as the synthetic examples may not capture the actual minority class patterns. The concern with generalizability further extends to temporal factors also. Customer behaviours could evolve models will need periodic retraining. The insurance data was collected over a decade ago and may not reflect the current market trends or customer expectations. Deployment in production environment would need continuous monitoring and ethical considerations and constant model updates to maintain prediction accuracy as market conditions change.

Chapter 7

7. Conclusion

7.1 Research Summary

This research did a comprehensive comparison of six machine learning algorithms (Logistic Regression, Random Forest, XGBoost, CatBoost, Support Vector Machine, K-Nearest Neighbors) for predicting customer churn in telecommunications and insurance industries. The research uses two real-world datasets having different characteristics - a telecommunications dataset with 8,454 B2B records with 6.49% churn rate, and insurance dataset with 40,284 B2C records with 23.83% churn rate. The methodology includes standard data preprocessing, then SMOTE for class imbalance, GridSearchCV for hyperparameter optimization and stratified cross-validation for getting best performing models. The trained models were then evaluated using multiple metrics like accuracy, precision, recall, F1-score, and ROC-AUC to properly assess the models performances.

The research also uses SHAP analysis to get insights on the feature importance patterns in both industries and developed a functional Streamlit web application for real-time prediction and interactive model comparison. This application shows how machine learning techniques can be easily implemented in a user-friendly way which is good for non-technical stakeholders.

7.2 Key Findings and Contributions

The research helps in making many contributions to churn prediction literature. The cross-industry comparison revealed that ensemble methods were consistently outperforming the traditional approaches, with CatBoost as expected being the best performer in both industries. The gap in performance in both industries also shows how dataset characteristics can impact the models' performance.SHAP analysis reveals different churn drivers between B2B and B2C models. Telecommunications focus more on operational metrics while insurance focused more on demographic factors. These insights help stakeholders to focus on actual causal factors rather than any correlated outcomes when deciding retention strategies.

The web application also helps in bridging the gap between academic research and practical development, showing how machine learning models can be made accessible to stakeholders using user-friendly interfaces.

7.3 Practical Application

The research helps give actionable guidance for businesses that want to implement churn prediction systems. Algorithm selection, threshold optimization and retention strategies should be adjusted based on business objectives. The web application shows the feasibility of model deployment. It is also important for businesses to understand the limitations of dataset for having realistic performance expectations and effective intervention strategies.

7.4 Future Work

There are lots of opportunities to improve this research. Advanced sampling techniques other than SMOTE like ADASYN or Borderline-SMOTE could help in importing the quality of synthetic examples. Deep learning techniques like ChurnNet, LSTM networks or transformer models could be better in capturing any temporal patterns which are needed for markets that undergo change a lot and for industries with long customer lifecycles.

Ensemble stacking methods that use predictions from different algorithms and combine them may get incremental performance improvements. Real-time deployment studies with A/B testing could also validate how effective models can be when deployed. Multi-industry meta-learning frameworks could explore transfer learning approaches to possibly transfer learning across business domains.

Future research should also focus more on temporal dynamics. This can be done with online learning systems that can adapt to evolving customer behaviours and markets. This can help expand into sectors like subscription services, banking or healthcare and further understand the generalizability of algorithms.

7.5 Final Remarks

This research successfully showed the complexities of cross-industry churn prediction by doing a systematic algorithm comparison. The findings give practical insights for model deployment but also shows the fundamental differences between B2B and B2C patterns in churn. As organisations are more relying on data for predictive analysis, it is important to understand data characteristics and industry-specific patterns for effective implementation that balances accuracy, interpretability and operational feasibility.

References

Breiman, L. (2001) 'Random forests', Machine Learning, 45(1), pp. 5-32. doi:10.1023/A:1010933404324.

Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P. (2002) 'SMOTE: Synthetic Minority Over-sampling Technique', Journal of Artificial Intelligence Research, 16, pp. 321-357. doi: 10.1613/jair.953.

Chen, T. and Guestrin, C. (2016) 'XGBoost: A scalable tree boosting system', in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco, CA, USA, 13-17 August. New York: ACM, pp. 785-794. doi: 10.1145/2939672.2939785.

Cover, T. and Hart, P. (1967) 'Nearest neighbor pattern classification', IEEE Transactions on Information Theory, 13(1), pp. 21-27. doi: 10.1109/TIT.1967.1053964.

Fawcett, T. (2006) 'An introduction to ROC analysis', Pattern Recognition Letters, 27(8), pp. 861-874. doi: 10.1016/j.patrec.2005.10.010.

Guillen, M., Nielsen, J.P. and Pérez-Marín, A.M. (2021) 'Near-miss telematics in motor insurance', Journal of Risk and Insurance, 88(3), pp. 569-589. Available at: https://doi.org/10.17632/vfchtm5y7j.1 (Accessed: 4 July 2025).

Hosmer, D.W. and Lemeshow, S. (2013) Applied Logistic Regression. 3rd edn. Hoboken, NJ: John Wiley & Sons.

Larivière, B. and Van den Poel, D. (2005) 'Predicting customer retention and profitability by using random forests and regression forests techniques', Expert Systems with Applications, 29(2), pp. 472-484. doi: 10.1016/j.eswa.2005.04.043.

Lundberg, S.M. and Lee, S.I. (2017) 'A unified approach to interpreting model predictions', in Advances in Neural Information Processing Systems 30 (NIPS 2017). Long Beach, CA, USA, 4-9 December, pp. 4765-4774.

Mozer, M.C., Wolniewicz, R., Grimes, D.B., Johnson, E. and Kaushansky, H. (2000) 'Predicting subscriber dissatisfaction and improving retention in the wireless telecommunications industry', IEEE Transactions on Neural Networks, 11(3), pp. 690-696. doi: 10.1109/72.846740.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V. and Gulin, A. (2018) 'CatBoost: Unbiased boosting with categorical features', in Advances in Neural Information Processing Systems 31 (NeurIPS 2018). Montréal, Canada, 3-8 December, pp. 6638-6648.

Ribeiro, M.T., Singh, S. and Guestrin, C. (2016) '"Why should I trust you?" Explaining the predictions of any classifier', in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco, CA, USA, 13-17 August. New York: ACM, pp. 1135-1144. doi: 10.1145/2939672.2939778.

Risselada, H., Verhoef, P.C. and Bijmolt, T.H.A. (2010) 'Staying power of churn prediction models', Journal of Interactive Marketing, 24(3), pp. 198-208. doi: 10.1016/j.intmar.2010.04.002.

Saha, S., Saha, C., Haque, M.M., Alam, M.G.R. and Talukder, A. (2024) 'ChurnNet: Deep learning enhanced customer churn prediction in telecommunication industry', IEEE Access, 12, pp. 4471-4484. doi: 10.1109/ACCESS.2024.3349950.

scikit-learn (2019). scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/ (Accessed: 18 October 2025).

Spiteri, M. and Azzopardi, G. (2018) 'Customer churn prediction for a motor insurance company', in Proceedings of the 2018 Thirteenth International Conference on Digital Information Management (ICDIM). Berlin, Germany, 24-26 September. IEEE, pp. 173-178. doi: 10.1109/ICDIM.2018.8847066.

Tokmakov, D. (2024) Customer Churn Dataset containing real records from a leading Bulgarian telecom operator, specifically for business customers, including small, medium, and large companies. Mendeley Data, V1. Available at: https://doi.org/10.17632/nrb55gr66h.1 (Accessed: 6 July 2025)

Vafeiadis, T., Diamantaras, K.I., Sarigiannidis, G. and Chatzisavvas, K.C. (2015) 'A comparison of machine learning techniques for customer churn prediction', Simulation Modelling Practice and Theory, 55, pp. 1-9. doi: 10.1016/j.simpat.2015.03.003.

Vapnik, V. (1998) Statistical Learning Theory. New York: John Wiley & Sons.

Verbeke, W., Dejaeger, K., Martens, D., Hur, J. and 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), pp. 211-229. doi: 10.1016/j.ejor.2011.09.031.

Appendix A: Source Code

All code snippets

requirements.txt - All Python dependencies required for the pipeline

requirements.txt

```
catboost==1.2.8
imbalanced-learn==0.14.0
joblib==1.5.2
plotly==6.5.0
matplotlib==3.10.7
numpy==2.3.5
pandas==2.3.3
scikit-learn==1.7.2
shap==0.50.0
streamlit==1.51.0
xgboost==3.1.2
```

config.py - Centralized configuration and hyperparameter grids

src > config.py > ...

You, 4 minutes ago | 1 author (You)

```python
# Data paths
TELECOM_DATA_PATH = "data/telecom_data.csv"
INSURANCE_DATA_PATH = "data/insurance_data_with_churn.csv"
PROCESSED_DATA_PATH = "data/processed/"
ARTIFACTS_DIR = "artifacts/"

# Model paths
MODEL_SAVE_PATH = "models/"
TELECOM_MODEL_NAME = "telecom_model.pkl"
INSURANCE_MODEL_NAME = "insurance_model.pkl"

# Model parameters
RANDOM_STATE = 42
TEST_SIZE = 0.2
CV_FOLDS = 5
GRID_PARAMS = {
    'Logistic Regression':  {'model__C': [0.01, 0.1, 1, 10]},
    'Random Forest':  {'model__n_estimators':[100,200], 'model__max_depth':[6,12,None]},
    'XGBoost': {'model__n_estimators':[100,200], 'model__max_depth':[3,6],
                'model__learning_rate':[0.01,0.05,0.1], 'model__scale_pos_weight':[1,5,10]},
    'CatBoost': {'model__iterations':[150,300], 'model__depth':[4,6]},
    'SVM': {'model__estimator__C': [0.01, 0.1, 1, 10]},
    'KNN': {'model__n_neighbors':[3,5,7], 'model__weights':['uniform','distance']}
}
```

src > config.py > ...

```python
# Feature columns
TELECOM_FEATURES = [
    'CRM_PID_Value_Segment', 'EffectiveSegment',
    'Active_subscribers', 'Not_Active_subscribers', 'Suspended_subscribers',
    'Total_SUBs', 'AvgMobileRevenue', 'AvgFIXRevenue', 'TotalRevenue', 'ARPU',
]

INSURANCE_FEATURES = [
    'gender', 'Age_client', 'year', 'age_of_car_M', 'Car_power_M', 'Car_2ndDriver_M',
    'num_policiesC', 'metro_code', 'Policy_PaymentMethodA', 'Policy_PaymentMethodH',
    'Insuredcapital_content_re', 'Insuredcapital_continent_re', 'appartment',
    'Client_Seniority', 'NClaims1', 'NClaims2', 'Claims1', 'Claims2'
]

# Target column
TARGET_COLUMN_TELECOM = 'CHURN'
TARGET_COLUMN_INSURANCE = 'churn'
TARGET_COLUMN = 'TARGET'

# SMOTE parameters
USE_SMOTE = True
SMOTE_SAMPLING_STRATEGY = 'auto'
SMOTE_RANDOM_STATE = RANDOM_STATE

# Streamlit app config
crm_choices = ["Bronze", "Iron", "Gold", "Silver", "Lead", "Platinum", "SME", "SE", "Sliver"]
eff_choices = ["SOHO", "VSE", "Other", "SME", "LE", "SE"]
ka_choices = ["VM", "DI", "MT", "AD", "RJ", "VU", "VT"]
```

data_preprocessing.py - DataPreprocessor class implementation

src > data_preprocessing.py > ...

You, 49 minutes ago | 1 author (You)

```python
import pandas as pd      You, 3 weeks ago • added all code
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
import joblib
import os
from config import *

You, 49 minutes ago | 1 author (You)
class DataPreprocessor:
    """
    This class is for cleaning, preprocessing and feature engineering
    """

    def __init__(self):
        self.scalers = {}
        self.encoders = {}
        self.imputers = {}
        self.current_dataset = None

    def load_telecom(self, file_path):
        print("Loading telecom data")
        self.current_dataset = "telecom"
        df = pd.read_csv(file_path)
        df = self._clean_df(df)

        target_cols = [TARGET_COLUMN_TELECOM, 'churn', 'CHURN', 'Churn']
        target = [c for c in target_cols if c in df.columns]
        if not target:
            raise ValueError(f"No target column in telecom data. Columns: {list(df.columns)}")
        target = target[0]

        if df[target].dtype == object:
            df[target] = df[target].str.strip().str.lower().map(
                {'yes': 1, 'no': 0, 'y': 1, 'n': 0, 'true': 1, 'false': 0})
        df[target] = pd.to_numeric(df[target], errors='coerce').fillna(0).astype(int)
        df[TARGET_COLUMN] = df[target]

        return df

```

```python
    def save_preprocessors(self, save_path):
        os.makedirs(save_path, exist_ok=True)
        joblib.dump(self.scalers, os.path.join(save_path, 'scalers.pkl'))
        joblib.dump(self.encoders, os.path.join(save_path, 'encoders.pkl'))
        joblib.dump(self.imputers, os.path.join(save_path, 'imputers.pkl'))

    def load_preprocessors(self, save_path):
        self.scalers = joblib.load(os.path.join(save_path, 'scalers.pkl'))
        self.encoders = joblib.load(os.path.join(save_path, 'encoders.pkl'))
        self.imputers = joblib.load(os.path.join(save_path, 'imputers.pkl'))
```

```python
    def load_insurance(self, file_path):
        print("Loading insurance data")
        self.current_dataset = "insurance"
        df = pd.read_csv(file_path)
        df = self._clean_df(df)

        target_cols = [TARGET_COLUMN_INSURANCE, 'churn', 'CHURN', 'Churn']
        target = [c for c in target_cols if c in df.columns]
        if not target:
            raise ValueError(f"No target column in insurance data. Columns: {list(df.columns)}")
        target = target[0]

        if df[target].dtype == object:
            df[target] = df[target].str.strip().str.lower().map(
                {'yes': 1, 'no': 0, 'y': 1, 'n': 0, 'true': 1, 'false': 0})
        df[target] = pd.to_numeric(df[target], errors='coerce').fillna(0).astype(int)
        df[TARGET_COLUMN] = df[target]

        return df

    def _clean_df(self, df):
        df = df.drop_duplicates().reset_index(drop=True)
        df.columns = df.columns.str.strip()

        num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
        cat_cols = df.select_dtypes(include=['object', 'category']).columns.tolist()

        num_key = f"{self.current_dataset}__numeric" if self.current_dataset else "global__numeric"
        cat_key = f"{self.current_dataset}__categorical" if self.current_dataset else "global__categorical"

        if len(num_cols) > 0:
            imputer_num = SimpleImputer(strategy='median')
            df[num_cols] = imputer_num.fit_transform(df[num_cols])
            self.imputers[num_key] = imputer_num

        if len(cat_cols) > 0:
            imputer_cat = SimpleImputer(strategy='most_frequent')
            df[cat_cols] = imputer_cat.fit_transform(df[cat_cols])
            self.imputers[cat_key] = imputer_cat

        return df
```

```python
    def prepare_features(self, df, dataset_type, fit=True):
        self.current_dataset = dataset_type
        df = df.copy()
        df.columns = df.columns.str.strip()

        if dataset_type == 'telecom':
            feature_columns = [c for c in TELECOM_FEATURES if c in df.columns]
        else:
            feature_columns = [c for c in INSURANCE_FEATURES if c in df.columns]

        X = df[feature_columns].copy()

        # Encode categorical features
        cat_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()
        for col in cat_cols:
            key = f"{dataset_type}__{col}"
            if fit:
                vals = X[col].astype(str).fillna('MISSING').unique().tolist()
                mapping = {v: i for i, v in enumerate(vals)}
                self.encoders[key] = mapping
                X[col] = X[col].astype(str).fillna('MISSING').map(mapping).astype(int)
            else:
                mapping = self.encoders.get(key, {})
                X[col] = X[col].astype(str).fillna('MISSING').apply(lambda v: mapping.get(v, -1)).astype(int)

        # Scale numerical features
        num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
        scaler_key = f"{dataset_type}_scaler"
        if len(num_cols) > 0:
            if fit:
                scaler = StandardScaler()
                X[num_cols] = scaler.fit_transform(X[num_cols])
                self.scalers[scaler_key] = scaler
            else:
                scaler = self.scalers.get(scaler_key)
                if scaler is not None:
                    X[num_cols] = scaler.transform(X[num_cols])

        return X
```

model_training.py - ModelTrainer class implementation

src > model_training.py > ...

```python
import numpy as np      You, 3 weeks ago • added all code …
import pandas as pd
import os
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold, cross_val_predict
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, precision_recall_curve
from sklearn.base import clone
import joblib
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline
import warnings
import time
warnings.filterwarnings('ignore')

from config import *

You, 56 minutes ago | 1 author (You)
class ModelTrainer:
    """
    Handles training, evaluation and comparison of multiple ML algorithms
    """

    def __init__(self):
        self.models = {
            'Logistic Regression': LogisticRegression(random_state=RANDOM_STATE,class_weight='balanced', max_iter=1000),
            'Random Forest': RandomForestClassifier(random_state=RANDOM_STATE, class_weight='balanced'),
            'XGBoost': XGBClassifier(random_state=RANDOM_STATE, eval_metric='logloss', use_label_encoder=False),
            'CatBoost': CatBoostClassifier(random_state=RANDOM_STATE, verbose=False),
            'SVM': CalibratedClassifierCV(estimator=LinearSVC(random_state=RANDOM_STATE, class_weight='balanced',
                                                              max_iter=5000),method='sigmoid',cv=3),
            'KNN': KNeighborsClassifier()
        }

        self.best_models = {}
        self.model_performance = {}
        self.trained_columns = {}
```

```python
    def train_all(self, X, y, dataset_name):
        print(f"\nTraining for {dataset_name} dataset")
        print(f"{'-'*50}")
        os.makedirs(os.path.join(ARTIFACTS_DIR, dataset_name), exist_ok=True)

        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=TEST_SIZE, random_state=RANDOM_STATE, stratify=y
        )

        results = {}
        for name, base_model in self.models.items():
            print(f"\nTraining {name}")

            if USE_SMOTE:
                sm = SMOTE(sampling_strategy=SMOTE_SAMPLING_STRATEGY, random_state=SMOTE_RANDOM_STATE)
                pipeline = ImbPipeline([('smote', sm), ('model', clone(base_model))])
            else:
                pipeline = ImbPipeline([('model', clone(base_model))])

            grid = GRID_PARAMS.get(name, {})
            skf = StratifiedKFold(n_splits=CV_FOLDS, shuffle=True, random_state=RANDOM_STATE)
            search = GridSearchCV(pipeline, param_grid=grid, scoring='f1', cv=skf, n_jobs=-1, verbose=0)
```

```
            start_time = time.time()
            search.fit(X_train, y_train)
            train_time = time.time() - start_time

            best_pipeline = search.best_estimator_
            cv_f1 = float(getattr(search, 'best_score_', np.nan))
            params = getattr(search, 'best_params_', None)

            train_proba = cross_val_predict(
                best_pipeline, X_train, y_train, cv=skf, method='predict_proba', n_jobs=-1)[:, 1]
            precisions, recalls, thresholds = precision_recall_curve(y_train, train_proba)
            f1s = 2 * (precisions * recalls) / (precisions + recalls + 1e-12)
            if len(thresholds) > 0:
                idx = int(np.nanargmax(f1s[:-1]))
                chosen_threshold = float(thresholds[idx])
            else:
                chosen_threshold = 0.5

            proba = best_pipeline.predict_proba(X_test)[:, 1]
            preds = (proba >= chosen_threshold).astype(int)
```

```
            # metrics
            accuracy = accuracy_score(y_test, preds)
            precision = precision_score(y_test, preds, zero_division=0)
            recall = recall_score(y_test, preds, zero_division=0)
            f1 = f1_score(y_test, preds, zero_division=0)
            roc_auc = roc_auc_score(y_test, proba) if (len(np.unique(y_test)) > 1) else np.nan

            results[name] = {
                'model': best_pipeline,
                'accuracy': accuracy,
                'precision': precision,
                'recall': recall,
                'f1_score': f1,
                'roc_auc': roc_auc,
                'cv_mean': cv_f1,
                'best_params': params,
                'best_threshold': chosen_threshold,
                'train_seconds': round(train_time, 1)
            }
            print(f"f1: {f1:.4f}, roc_auc: {roc_auc if not np.isnan(roc_auc) else 'N/A'} (cv_f1: {cv_f1 if not np.isnan(cv_
        self.model_performance[dataset_name] = results
        rows = []
        for alg, vals in results.items():
            rows.append({
                'dataset': dataset_name,
                'algorithm': alg,
                'f1_score': vals['f1_score'],
                'accuracy': vals['accuracy'],
                'recall': vals['recall'],
                'precision': vals['precision'],
                'roc_auc': vals['roc_auc'],
                'cv_best_f1': vals.get('cv_mean'),
                'best_params': str(vals.get('best_params')),
                'best_threshold': vals.get('best_threshold'),
                'train_seconds': vals.get('train_seconds')
            })
        pd.DataFrame(rows).sort_values('f1_score', ascending=False).to_csv(
            os.path.join(
                ARTIFACTS_DIR, dataset_name, f"metrics_grid_{dataset_name}.csv"), index=False)
        return results
```

```
    def get_best_model(self, dataset_name, metric='f1_score'):
        if dataset_name not in self.model_performance:
            raise ValueError(f"No results for {dataset_name}")
        performance = self.model_performance[dataset_name]
        valid_models = [name for name, vals in performance.items() if not np.isnan(
            vals.get(metric, np.nan))]
        if not valid_models:
            best_model_name = max(performance.keys(), key=lambda x: performance[x].get('cv_mean', 0))
        else:
            best_model_name = max(valid_models, key=lambda x: performance[x][metric])

        best_model = performance[best_model_name]['model']
        self.best_models[dataset_name] = best_model

        print(f"\nBest model for {dataset_name}: {best_model_name}")
        print(f"Best {metric}: {performance[best_model_name][metric]:.4f}")

        return best_model_name, best_model

    def save_model(self, model, file_path):
        joblib.dump(model, file_path, compress=3)
        print(f"Model saved to {file_path}")

    def load_model(self, file_path):
        model = joblib.load(file_path)
        print(f"Model loaded from {file_path}")
        return model
```

Explainability.py - ModelExplainer class for SHAP analysis

src > explainability.py > ...

```
You, 1 hour ago | 1 author (You)
1   import shap      You, 3 weeks ago • added all code
2   import matplotlib.pyplot as plt
3   import pandas as pd
4   import numpy as np
5
    You, 1 hour ago | 1 author (You)
6   class ModelExplainer:
7       """
8       Explainability using SHAP
9       """
10
11      def __init__(self):
12          self.shap_explainers = {}
13
14      def run_shap(self, model, X, feature_names, dataset_name):
15          print(f"\nStart SHAP analysis for {dataset_name}")
16
17          explainer = shap.Explainer(model, X)
18          shap_values = explainer(X)
19          self.shap_explainers[dataset_name] = explainer
20
21          shap_mean = np.mean(np.abs(shap_values.values), axis=0)
22          sorted_idx = np.argsort(shap_mean)[::-1]
23          n_top = min(15, len(feature_names))
24          top_indices = sorted_idx[:n_top]
25          top_features = [feature_names[i] for i in top_indices]
26          top_values = shap_mean[top_indices]
27
28          plt.figure(figsize=(8, 6))
29          plt.barh(range(len(top_features))[::-1], top_values[::-1])
30          plt.yticks(range(len(top_features)), top_features[::-1])
31          plt.title(f"Top {n_top} SHAP features - {dataset_name}")
32          plt.xlabel("mean(|SHAP value|)")
33          plt.tight_layout()
34          plt.savefig(f"shap_bar_{dataset_name}.png", dpi=300, bbox_inches='tight')
35          plt.close()
36
37          return shap_values
38
39      def compare_features(self, shap1, features1, shap2, features2,
40                           label_1="telecom", label_2="insurance", top_n=10):
41          imp1 = np.abs(shap1).mean(axis=0)
42          imp2 = np.abs(shap2).mean(axis=0)
43
44          df1 = pd.DataFrame({"feature": features1, "importance": imp1}).sort_values(
45              "importance", ascending=False)
46          df2 = pd.DataFrame({"feature": features2, "importance": imp2}).sort_values(
47              "importance", ascending=False)
48
49          print(f"\nTop {top_n} features - {label_1}:")
50          print(df1.head(top_n).to_string(index=False))
51
52          print(f"\nTop {top_n} features - {label_2}:")
53          print(df2.head(top_n).to_string(index=False))
54
55          return df1, df2
```

Main.py - full pipeline execution

```
src > main.py > ...
    You, 1 hour ago | 1 author (You)
1   import os      You, 3 weeks ago • added all code …
2   import pandas as pd
3   from data_preprocessing import DataPreprocessor
4   from model_training import ModelTrainer
5   from explainability import ModelExplainer
6   from config import *
7
8   def main():
9       print("Starting Cross-Industry Churn Prediction")
10
11      preprocessor = DataPreprocessor()
12      trainer = ModelTrainer()
13      explainer = ModelExplainer()
14
15      os.makedirs(PROCESSED_DATA_PATH, exist_ok=True)
16      os.makedirs(MODEL_SAVE_PATH, exist_ok=True)
17      os.makedirs(ARTIFACTS_DIR, exist_ok=True)
18
19
20      try:
21          # STEP 1 Data Load and Preprocessing
22          print("\nStep 1 - Data Load and process")
23          print("-" * 50)
24
25          telecom_df = preprocessor.load_telecom(TELECOM_DATA_PATH)
26          insurance_df = preprocessor.load_insurance(INSURANCE_DATA_PATH)
27          print("Telecom rows:", telecom_df.shape[0], "Insurance rows:", insurance_df.shape[0])
28
29          X_telecom = preprocessor.prepare_features(telecom_df, 'telecom', fit=True)
30          y_telecom = telecom_df[TARGET_COLUMN]
31
32          X_insurance = preprocessor.prepare_features(insurance_df, 'insurance', fit=True)
33          y_insurance = insurance_df[TARGET_COLUMN]
34          trainer.random_state = RANDOM_STATE
35
36          print(f"Telecom features {X_telecom.shape}")
37          print(f"Insurance features {X_insurance.shape}")
38
39          preprocessor.save_preprocessors(PROCESSED_DATA_PATH)
```

```
41          # STEP 2 Model Training
42          print("\nStep 2 - Model Training")
43          print("-" * 50)
44
45          telecom_performance = trainer.train_all(X_telecom, y_telecom, 'telecom')
46          best_telecom_name, best_telecom_model = trainer.get_best_model('telecom')
47          print("Best telecom model:", best_telecom_name)
48
49          insurance_performance = trainer.train_all(X_insurance, y_insurance, 'insurance')
50          best_insurance_name, best_insurance_model = trainer.get_best_model('insurance')
51          print("Best insurance model:", best_insurance_name)
52
53          best_telecom_model = best_telecom_model.named_steps["model"] if hasattr(
54              best_telecom_model, "named_steps"
55              ) and "model" in best_telecom_model.named_steps else best_telecom_model
56          best_insurance_model = best_insurance_model.named_steps["model"] if hasattr(
57              best_insurance_model, "named_steps"
58              ) and "model" in best_insurance_model.named_steps else best_insurance_model
59
```

```
    # STEP 3 Model Explainability
    print("\nStep 3 - SHAP Explainability")
    print("-" * 50)

    try:
        shap_telecom = explainer.run_shap(best_telecom_model, X_telecom,
                                          list(X_telecom.columns), 'telecom')
    except Exception as e:
        print("SHAP for telecom failed:", e)
        shap_telecom = None

    try:
        Nrows = 1000
        X_sample = X_insurance.sample(n=min(Nrows, len(X_insurance)), random_state=RANDOM_STATE)
        if best_insurance_name == "SVM":
            model_fn = lambda X: best_insurance_model.predict_proba(X)[:, 1]
            shap_insurance = explainer.run_shap(model_fn, X_sample, list(X_sample.columns),
                                                'insurance')
        else:
            shap_insurance = explainer.run_shap(best_insurance_model, X_sample,
                                                list(X_sample.columns), 'insurance')
    except Exception as e:
        print("SHAP for insurance failed:", e)
        shap_insurance = None

    try:
        if shap_telecom is not None and shap_insurance is not None:
            shap_tel = shap_telecom.values if hasattr(shap_telecom, "values") else shap_telecom
            shap_ins = shap_insurance.values if hasattr(shap_insurance, "values") else shap_insurance
            telecom_imp_df, insurance_imp_df = explainer.compare_feature_importance(
                shap_tel, list(X_telecom.columns),
                shap_ins, list(X_insurance.columns),
                label_1="telecom", label_2="insurance", top_n=10
            )
            print("Feature importance comparison done.")
            telecom_imp_df.to_csv("artifacts/telecom/telecom_top10.csv", index=False)
            insurance_imp_df.to_csv("artifacts/insurance/insurance_top10.csv", index=False)
        else:
            print("Skipping feature importance comparison (missing SHAP results).")

    except Exception as e:
        print("Comparison of feature importance failed:", e)
```

```
    # STEP 4 Saving Models
    print("\nStep 4 - Saving Models")
    print("-" * 50)
    telecom_path = os.path.join(MODEL_SAVE_PATH, TELECOM_MODEL_NAME)
    insurance_path = os.path.join(MODEL_SAVE_PATH, INSURANCE_MODEL_NAME)

    trainer.save_model(best_telecom_model, telecom_path)
    trainer.save_model(best_insurance_model, insurance_path)
    print("\nPipeline completed successfully!")
    print(f"Best Telecom Model: {best_telecom_name}")
    print(f"Best Insurance Model: {best_insurance_name}")
```

```
        # Print final performance summary
        print("\nFinal Performance Summary")
        print("Telecom Dataset:")
        summary = []
        for dataset_name, performance, best_name in [
            ("Telecom", telecom_performance, best_telecom_name),
            ("Insurance", insurance_performance, best_insurance_name)
        ]:
            print(f"\n{dataset_name} Dataset:")
            if best_name in performance:
                perf = performance[best_name].copy()
                perf.pop('model', None)
                perf['model_name'] = best_name

                for metric in ['accuracy','precision','recall','f1_score','roc_auc','cv_mean']:
                    val = perf.get(metric)
                    if isinstance(val, float):
                        print(f"  {metric}: {val:.4f}")
                    else:
                        print(f"  {metric}: {val}")

                perf['dataset'] = dataset_name.lower()
                summary.append(perf)

            else:
                print("  No performance info available.")
                summary.append({'dataset': dataset_name.lower(), 'model_name': best_name})

            metrics_df = pd.DataFrame(summary)
            cols = ['dataset','model_name'] + [c for c in metrics_df.columns if c not in (
                'dataset','model_name')]
            metrics_df = metrics_df[cols]
            metrics_df.to_csv("artifacts/metrics_all.csv", index=False)

    except Exception as e:
        print(f"Error in pipeline execution: {str(e)}")
        raise

def run_streamlit_app():
    print("Starting Streamlit application")
    os.system("streamlit run src/streamlit_app.py")

if __name__ == "__main__":
    main()
    run_streamlit_app()
```

streamlit_app.py - Web application for predictions and comparisons

src > streamlit_app.py > ...

You, 1 hour ago | 1 author (You)

```python
import streamlit as st      You, 3 weeks ago • added all code
import plotly.express as px
import pandas as pd
import joblib
import matplotlib.pyplot as plt
import sys
import os
from data_preprocessing import DataPreprocessor

sys.path.append(os.path.join(os.path.dirname(__file__), '..', 'src'))

from config import *

```

You, 1 hour ago | 1 author (You)

```python
class ChurnPredictionApp:
    """
    Streamlit application for churn prediction and analysis
    """

    def __init__(self):
        self.telecom_model = None
        self.insurance_model = None
        self.preprocessor = None

        self.load_models()

    def load_models(self):
        telecom_path = os.path.join(MODEL_SAVE_PATH, TELECOM_MODEL_NAME)
        insurance_path = os.path.join(MODEL_SAVE_PATH, INSURANCE_MODEL_NAME)
        prep_dir = PROCESSED_DATA_PATH

        if os.path.exists(telecom_path):
            self.telecom_model = joblib.load(telecom_path)
        else:
            st.info("No telecom model found.")

        if os.path.exists(insurance_path):
            self.insurance_model = joblib.load(insurance_path)
        else:
            st.info("No insurance model found.")

        if os.path.exists(prep_dir):
            p = DataPreprocessor()
            p.load_preprocessors(prep_dir)
            self.preprocessor = p
        else:
            st.info("No preprocessors folder found.")
```

```
    def run(self):
        """Run the Streamlit application"""
        st.set_page_config(
            page_title="Cross-Industry Customer Churn Prediction",
            layout="wide"
        )

        st.title("Cross-Industry Customer Churn Prediction")
        st.markdown("""**Anish Rao | Dublin Business School | 20066423**""")
        st.markdown("""
        This application predicts customer churn for both Telecom and Insurance industries
        using machine learning models trained on historical data and provides insights
                    into model performance and feature importance.
        """)
        st.markdown("---")

        st.sidebar.title("Navigation")
        page = st.sidebar.selectbox(
            "",
            ["About","Dataset Info", "Best Model Analysis",
             "Model Comparisons", "Single Prediction", "Batch Prediction"]
        )

        if page == "Single Prediction":
            self.single_prediction()
        elif page == "Batch Prediction":
            self.batch_prediction()
        elif page == "Best Model Analysis":
            self.model_analysis()
        elif page == "Model Comparisons":
            self.model_comparison()
        elif page == "About":
            self.about_page()
        else:
            self.about_dataset()

    def single_prediction(self):
        """Single prediction interface"""
        st.header("Single Customer Prediction")

        industry = st.radio("Select Industry:", ("Telecom", "Insurance"))

        if industry == "Telecom":
            self.telecom_prediction_form()
        else:
            self.insurance_prediction_form()
```

```python
    def telecom_prediction_form(self):
        """Telecom prediction form"""
        st.subheader("Telecom Customer Details")

        c1, c2, c3 = st.columns(3)

        with c1:
            crm_seg = st.selectbox("CRM_PID_Value_Segment", crm_choices, index=0)
            eff_seg = st.selectbox("EffectiveSegment", eff_choices, index=0)
            total_revenue = st.number_input("TotalRevenue", min_value=0.0, value=40.17)
        with c2:
            active_subs = st.number_input("Active_subscribers", min_value=0, value=6)
            not_active = st.number_input("Not_Active_subscribers", min_value=0.0, value=2.0)
            suspended = st.number_input("Suspended_subscribers", min_value=0.0, value=0.0)
        with c3:
            total_subs = st.number_input("Total_SUBs", min_value=0, value=6)
            avg_mobile = st.number_input("AvgMobileRevenue", min_value=0.0, value=40.17)
            avg_fix = st.number_input("AvgFIXRevenue", min_value=0.0, value=0.0)

        arpu = st.number_input("ARPU", min_value=0.0, value=20.0)

        if st.button("Predict Churn"):
            if crm_seg == "Sliver":
                crm_seg = "Silver"

            input_data = {
                "CRM_PID_Value_Segment": crm_seg,
                "EffectiveSegment": eff_seg,
                "TotalRevenue": total_revenue,
                "Active_subscribers": active_subs,
                "Not_Active_subscribers": not_active,
                "Suspended_subscribers": suspended,
                "Total_SUBs": total_subs,
                "AvgMobileRevenue": avg_mobile,
                "AvgFIXRevenue": avg_fix,
                "ARPU": arpu
            }

            pred = self.predict_telecom(input_data)
            self.display_prediction_results(pred, "Telecom")
```

```python
    def insurance_prediction_form(self):
        """Insurance prediction form"""
        c1, c2, c3 = st.columns(3)

        with c1:
            gender = st.selectbox("gender", [0, 1], index=0)
            age = st.number_input("Age_client", min_value=18, max_value=120, value=40)
            year = st.number_input("year", min_value=1, max_value=10, value=1)
            age_of_car = st.number_input("age_of_car_M", min_value=0, max_value=80, value=5)
            car_power = st.number_input("Car_power_M", min_value=1.0, max_value=600.0, value=100.0)
            second_driver = st.selectbox("Car_2ndDriver_M", [0, 1], index=0)

        with c2:
            num_policies = st.number_input("num_policiesC", min_value=0, max_value=10, value=1)
            metro = st.selectbox("metro_code", [0, 1], index=0)
            payA = st.selectbox("Policy_PaymentMethodA", [0, 1], index=0)
            payH = st.selectbox("Policy_PaymentMethodH", [0, 1], index=0)
            insured_content = st.number_input("Insuredcapital_content_re", min_value=0.0, value=5000.0)
            insured_continent = st.number_input("Insuredcapital_continent_re", min_value=0.0, value=8000.0)

        with c3:
            appartment = st.selectbox("appartment", [0, 1], index=0)
            client_seniority = st.number_input("Client_Seniority", min_value=0.0, value=5.0)
            nclaims1 = st.number_input("NClaims1", min_value=0, max_value=100, value=0)
            nclaims2 = st.number_input("NClaims2", min_value=0, max_value=100, value=0)
            claims1 = st.number_input("Claims1", min_value=0.0, value=0.0)
            claims2 = st.number_input("Claims2", min_value=0.0, value=0.0)

        if st.button("Predict Churn"):
            input_data = {
                "gender": int(gender),
                "Age_client": int(age),
                "year": int(year),
                "age_of_car_M": int(age_of_car),
                "Car_power_M": float(car_power),
                "Car_2ndDriver_M": int(second_driver),
                "num_policiesC": int(num_policies),
                "metro_code": int(metro),
                "Policy_PaymentMethodA": int(payA),
                "Policy_PaymentMethodH": int(payH),
                "Insuredcapital_content_re": float(insured_content),
                "Insuredcapital_continent_re": float(insured_continent),
                "appartment": int(appartment),
                "Client_Seniority": float(client_seniority),
                "NClaims1": int(nclaims1),
                "NClaims2": int(nclaims2),
                "Claims1": float(claims1),
                "Claims2": float(claims2)
            }

            pred = self.predict_insurance(input_data)
            self.display_prediction_results(pred, "Insurance")
```

```python
    def predict_telecom(self, input_data):
        try:
            df = pd.DataFrame([input_data])

            if self.preprocessor:
                df = self.preprocessor.prepare_features(df, 'telecom', fit=False)

            prob = self.telecom_model.predict_proba(df)[0]
            prediction = self.telecom_model.predict(df)[0]

            return {
                "prediction": int(prediction),
                "probability": float(prob[1]),
                "confidence": float(max(prob))
            }
        except Exception as e:
            st.error(f"Telecom prediction error: {e}")
            return None

    def predict_insurance(self, input_data):
        try:
            df = pd.DataFrame([input_data])

            if self.preprocessor:
                df = self.preprocessor.prepare_features(df, 'insurance', fit=False)

            prob = self.insurance_model.predict_proba(df)[0]
            prediction = self.insurance_model.predict(df)[0]

            return {
                "prediction": int(prediction),
                "probability": float(prob[1]),
                "confidence": float(max(prob))
            }
        except Exception as e:
            st.error(f"Insurance prediction error: {e}")
            return None
```

```python
    def display_prediction_results(self, prediction, industry):
        """Display prediction results"""
        if prediction is None:
            return

        st.subheader("Prediction Results")

        col1, col2 = st.columns(2)
        pred_value = prediction.get("prediction", 0)
        probability = prediction.get("probability", 0)
        confidence = prediction.get("confidence", probability)
        with col1:
            st.metric("Churn Prediction", "Yes" if pred_value == 1 else "No")
        with col2:
            st.metric("Confidence", f"{confidence:.2%}")
        fig, ax = plt.subplots(figsize=(6, 2.5))
        values = [1 - probability, probability]
        labels = ["No Churn", "Churn"]
        colors = ["green", "red"]

        ax.bar(labels, values, color=colors, alpha=0.8)
        ax.set_ylim(0, 1)
        ax.set_ylabel("Probability", fontsize=10)
        ax.set_title(f"{industry} - Churn Probabilities", fontsize=14)
        for i, v in enumerate(values):
            ax.text(i, v + 0.02, f"{v:.1%}", ha="center", fontsize=11)

        plt.tight_layout()
        st.pyplot(fig, use_container_width=True)
        plt.close(fig)
```

```python
    def batch_prediction(self):
        """Batch prediction interface"""
        st.header("Batch Prediction")

        industry = st.radio("Select Industry for Batch Prediction:", ("Telecom", "Insurance"))

        uploaded_file = st.file_uploader(f"Upload {industry} CSV file", type=['csv'])

        if uploaded_file is not None:
            try:
                df = pd.read_csv(uploaded_file)
                st.write("Uploaded Data Preview:")
                st.dataframe(df.head())
                if df.empty:
                    st.warning("Uploaded file is empty.")
                    return

                if st.button("Predict Batch"):
                    with st.spinner("Processing predictions..."):
                        result = self.batch_predict(df, industry)
                        if result is None:
                            st.error("Batch prediction failed (see logs).")
                            return

                        preds = result['predictions']
                        probs = result['probabilities']
                        df['Churn_Prediction'] = preds
                        df['Churn_Probability'] = probs

                        # Display results
                        st.subheader("Prediction Results (sample)")
                        st.dataframe(df[['Churn_Prediction', 'Churn_Probability']].head(10))

                        churn_counts = df['Churn_Prediction'].value_counts().to_dict()
                        st.write("Prediction counts:", churn_counts)

                        # Download results
                        csv = df.to_csv(index=False)
                        st.download_button(
                            label="Download Predictions",
                            data=csv,
                            file_name=f"{industry.lower()}_churn_predictions.csv",
                            mime="text/csv"
                        )
            except Exception as e:
                st.error(f"Error processing file: {str(e)}")
```

```
    def batch_predict(self, df, industry):
        """Make batch predictions"""
        try:
            if industry == "Telecom":
                model = self.telecom_model
            else:
                model = self.insurance_model

            if model is None:
                st.error("Requested model is not loaded.")
                return None

            if self.preprocessor:
                processed_df = self.preprocessor.prepare_features(df.copy(), industry.lower(), fit=F
            else:
                processed_df = df

            if processed_df is None or processed_df.shape[0] == 0:
                st.error("Preprocessed data is empty.")
                return None

            predictions = model.predict(processed_df)
            probabilities = model.predict_proba(processed_df)[:, 1]

            return {
                'predictions': list(predictions),
                'probabilities': list(map(float, probabilities))
            }
        except Exception as e:
            st.error(f"Batch prediction error: {str(e)}")
            return None
```

```
337     def model_analysis(self):
338         """Model analysis and insights"""
339         st.header("Model Analysis and Insights")
340 
341         ARTIFACTS_DIR = "artifacts"
342         telecom_path = os.path.join(ARTIFACTS_DIR, "telecom/telecom_top10.csv")
343         insurance_path = os.path.join(ARTIFACTS_DIR, "insurance/insurance_top10.csv")
344         metrics_path = os.path.join(ARTIFACTS_DIR, "metrics_all.csv")
345 
346         st.subheader("Best Model Metrics")
347         metrics_df = pd.read_csv(metrics_path)
348 
349         if "Model" in metrics_df.columns:
350             metrics_df = metrics_df.set_index("Model")
351         for c in metrics_df.select_dtypes(include=["float", "int"]).columns:
352             metrics_df[c] = metrics_df[c].round(4)
353         st.dataframe(metrics_df, use_container_width=True)
354 
355         st.subheader("Telecom vs Insurance Comparison")
356         metrics = ["accuracy", "f1_score", "precision", "recall"]
357         labels = ["Accuracy", "F1", "Precision", "Recall"]
358         df_tmp = metrics_df.copy()
359         df_tmp.columns = df_tmp.columns.str.lower()
360 
361         row_tel = df_tmp[df_tmp["dataset"] == "telecom"].iloc[0]
362         row_ins = df_tmp[df_tmp["dataset"] == "insurance"].iloc[0]
363         vals_tel = [float(row_tel[m]) for m in metrics]
364         vals_ins = [float(row_ins[m]) for m in metrics]
365 
366         order = []
367         data = []
368         for name, t, i in zip(labels, vals_tel, vals_ins):
369             order += [f"{name} — Telecom", f"{name} — Insurance"]
370             data.append((f"{name} — Telecom", t, "Telecom"))
371             data.append((f"{name} — Insurance", i, "Insurance"))
372 
373         df_plot = pd.DataFrame(data, columns=["label", "score", "dataset"])
374         df_plot["label"] = pd.Categorical(df_plot["label"], categories=order, ordered=True)
375 
376         fig = px.bar(df_plot, x="label", y="score", color="dataset", text=df_plot["score"].round(3),
377                      category_orders={"label": order})
378         fig.update_layout(
379             yaxis=dict(range=[0, 1]), showlegend=False,
380             xaxis_tickangle=-45, height=420, margin=dict(t=40,b=120))
381         fig.update_traces(textposition="outside")
382 
383         st.plotly_chart(fig, use_container_width=True)
```

```
        st.subheader("Top 10 Churn Drivers")
        telecom_top = pd.read_csv(telecom_path)
        insurance_top = pd.read_csv(insurance_path)
        telecom_top = telecom_top.sort_values(
            "importance", ascending=False).head(10).reset_index(drop=True)
        insurance_top = insurance_top.sort_values(
            "importance", ascending=False).head(10).reset_index(drop=True)

        combined = pd.DataFrame({
            "Telecom Feature": telecom_top["feature"].values,
            "Telecom Importance": telecom_top["importance"].values,
            "Insurance Feature": insurance_top["feature"].values,
            "Insurance Importance": insurance_top["importance"].values
        })

        combined["Telecom Importance"] = combined["Telecom Importance"].map(
            lambda x: round(float(x), 4))
        combined["Insurance Importance"] = combined["Insurance Importance"].map(
            lambda x: round(float(x), 4))

        st.dataframe(combined, use_container_width=True)

        st.subheader("SHAP Plots for best models")
        left_col, right_col = st.columns(2)
        with left_col:
            st.image("shap_bar_telecom.png", caption="Telecom — SHAP Bar",
                     width=550, use_container_width=True)
        with right_col:
            st.image("shap_bar_insurance.png", caption="Insurance — SHAP Bar",
                     width=550, use_container_width=True)
```

```
def model_comparison(self):
    st.header("Model Comparison Between Industries")

    tel = pd.read_csv("artifacts/telecom/metrics_grid_telecom.csv")
    ins = pd.read_csv("artifacts/insurance/metrics_grid_insurance.csv")
    tel.columns = tel.columns.str.lower()
    ins.columns = ins.columns.str.lower()

    col1, col2 = st.columns(2)
    telecom_algo = col1.selectbox("Telecom Algorithm", sorted(tel["algorithm"].unique()))
    insurance_algo = col2.selectbox("Insurance Algorithm", sorted(ins["algorithm"].unique()))

    metrics = ["accuracy", "precision", "recall", "f1_score"]
    labels = ["Accuracy", "Precision", "Recall", "F1"]

    left, right = st.columns(2)
    with left:
        row = tel[tel["algorithm"] == telecom_algo].iloc[0]
        df_t = pd.DataFrame({"metric": labels, "score": [row[m] for m in metrics]})
        fig_t = px.bar(df_t, x="score", y="metric", orientation="h", text=df_t["score"].round(3), title=f"{telecom_algo} (Telecom)")
        fig_t.update_layout(xaxis=dict(range=[0, 1]), height=360)
        st.plotly_chart(fig_t, use_container_width=True, key="tel_chart")

    with right:
        row = ins[ins["algorithm"] == insurance_algo].iloc[0]
        df_i = pd.DataFrame({"metric": labels, "score": [row[m] for m in metrics]})
        fig_i = px.bar(df_i, x="score", y="metric", orientation="h", text=df_i["score"].round(3), title=f"{insurance_algo} (Insurance)")
        fig_i.update_layout(xaxis=dict(range=[0, 1]), height=360)
        st.plotly_chart(fig_i, use_container_width=True, key="ins_chart")

    df = pd.concat([tel, ins], ignore_index=True)

    c1, c2 = st.columns(2)
    dataset = c1.selectbox("Dataset", ["Both", "Telecom", "Insurance"])
    metric = c2.selectbox("Metric", ["f1_score", "accuracy", "precision", "recall", "roc_auc"])

    if dataset != "Both":
        df = df[df["dataset"] == dataset.lower()]

    df = df.sort_values(metric, ascending=False).reset_index(drop=True)

    fig = px.bar(df, x=metric, y="algorithm", color="dataset", orientation="h", text=df[metric].round(3))
    fig.update_layout(height=450, yaxis={'categoryorder': 'total ascending'})
    st.plotly_chart(fig, use_container_width=True, key="all_chart")
    st.dataframe(df, use_container_width=True)
```

```
    def about_page(self):
        """About page"""
        st.header("About This Application")

        st.markdown("""
        ## Cross-Industry Churn Prediction System

        This application is made for a masters research project that demonstrates:

        - **Machine Learning**: Implementation of 6 different algorithms for churn prediction
        - **Cross-Industry Analysis**: Comparison between Telecom and Insurance sectors
        - **Model Explainability**: SHAP for feature importance and local explanations
        - **Web Interface**: User-friendly Streamlit application for predictions and model insights

        This study addresses two main research questions:
        1. Which machine learning algorithms(Logistic Regression, Random Forest, XGBoost, CatBoost, SVM,KNN)
        have the best predictive performance in the telecom and insurance sectors for customer churn?

        2. Which are the most important features in both these sectors for customer churn,
        and how do these differ between telecom and insurance sectors?

        ### Models Implemented:
        - Logistic Regression
        - Random Forest
        - XGBoost
        - CatBoost
        - Support Vector Machine
        - K-Nearest Neighbors

        ### Features:
        - Single customer prediction
        - Batch prediction from CSV files
        - Model performance analysis
        - Feature importance visualization
        - Cross-industry comparisons

        **Note**: This is only for demonstration. For production use,
        additional data, validation, security measures, and model training/monitoring will be needed.

        **Author**: Anish Rao, Dublin Business School, 20066423

        anish.rao888@gmail.com
        """)
```

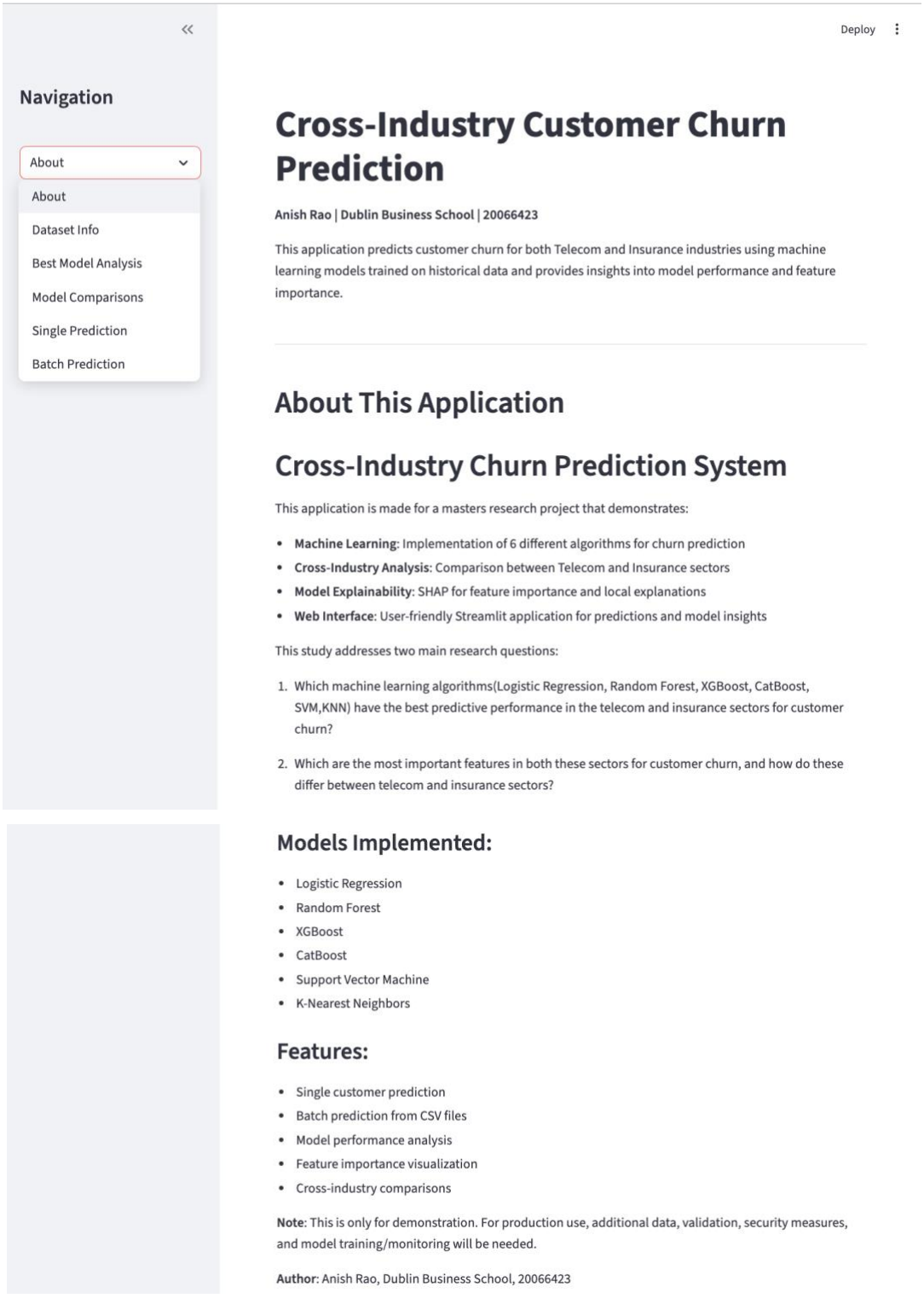
```
def main():
    app = ChurnPredictionApp()
    app.run()

if __name__ == "__main__":
    main()
```

```python
    def about_dataset(self):
        """About dataset page"""

        st.markdown("""
        ## Dataset Overview

        This project uses two anonymised, real-world datasets from the telecom and insurance
                industries to compare churn behavior across different business models.

        ---

        ## Telecom Dataset
        - **8454 business customers** (Bulgarian operator)
        - **Churn rate:** 6.49%
        - Higher churn seen in **Gold/Platinum/SME** segments.

        """)
        left_col, right_col = st.columns(2)
        with left_col:
            st.image("images/telecom_churn_pie.png", caption="Telecom Churn Distribution",
                     width=450, use_container_width=True)
        with right_col:
            st.image("images/telecom_segment_churn.png", caption="Telecom Churn by Segment",
                     width=450, use_container_width=True)

        st.markdown("🔗 https://data.mendeley.com/datasets/nrb55gr66h/1")

        st.markdown("---")

        st.markdown("""
        ## Insurance Dataset
        - **40,284 customers** tracked over 5 years (Spanish insurer)
        - **Churn rate:** 23.83%
        - Highest churn among customers with **recent claims**.

        """)
        left_col, right_col = st.columns(2)
        with left_col:
            st.image("images/insurance_churn_pie.png", caption="Insurance Churn Distribution",
                     width=450, use_container_width=True)
        with right_col:
            st.image("images/insurance_claim_churn.png", caption="Insurance Churn by Claim Type",
                     width=450, use_container_width=True)

        st.markdown("🔗 https://data.mendeley.com/datasets/vfchtm5y7j/1")

        st.markdown("---")
```

```python
        st.markdown("""
        ## Comparison
        """)

        st.markdown("""
        <style>
        .quick-table {width:100%; border-collapse:collapse;}
        .quick-table th, .quick-table td {
            border:1px solid #ddd; padding:6px; font-size:14px;
        }
        .quick-table th {background:#f7f7f7; font-weight:600;}
        </style>

        <table class="quick-table">
        <tr><th>Aspect</th><th>Telecom</th><th>Insurance</th></tr>
        <tr><td>Business Model</td><td>B2B Contracts</td><td>B2C Annual Renewal</td></tr>
        <tr><td>Churn %age</td><td>6.49%</td><td>23.83%</td></tr>
        <tr><td>Main Driver</td><td>Customer Value Segment</td><td>Claim History</td></tr>
        <tr><td>Missing Data</td><td>Some</td><td>None</td></tr>
        </table>
        """, unsafe_allow_html=True)

        st.markdown("---")
```

Appendix B: Streamlit App Screenshots

About page



Dataset info page

Navigation

Dataset Info

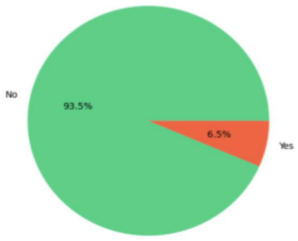
Deploy

Dataset Overview

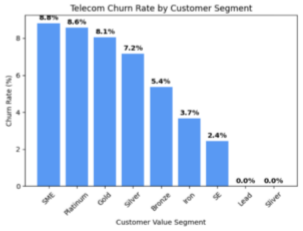
This project uses two anonymised, real-world datasets from the telecom and insurance industries to compare churn behavior across different business models.

Telecom Dataset

- **8454 business customers** (Bulgarian operator)
- **Churn rate:** 6.49%
- Higher churn seen in **Gold/Platinum/SME** segments.



Telecom Churn Distribution



Telecom Churn by Segment

https://data.mendeley.com/datasets/nrb55gr66h/1

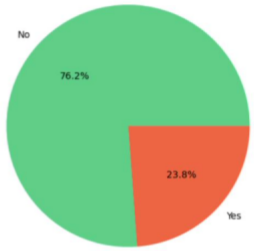
Navigation

Dataset Info

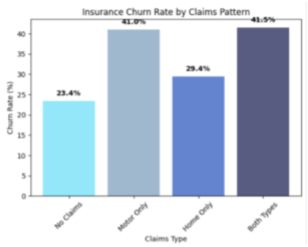
Deploy

Insurance Dataset

- **40,284 customers** tracked over 5 years (Spanish insurer)
- **Churn rate:** 23.83%
- Highest churn among customers with **recent claims**.



Insurance Churn Distribution



Insurance Churn by Claim Type

https://data.mendeley.com/datasets/vfchtm5y7j/1

Comparison

Aspect	Telecom	Insurance
Business Model	B2B Contracts	B2C Annual Renewal
Churn %age	6.49%	23.83%
Main Driver	Customer Value Segment	Claim History
Missing Data	Some	None

Best model analysis page

Deploy

Navigation

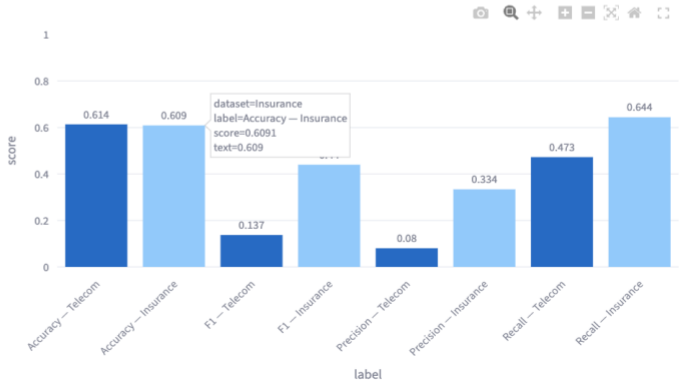
Best Model Analysis

Model Analysis and Insights

Best Model Metrics

	dataset	model_name	accuracy	precision	recall	f1_score	roc_auc	cv_mean	best_params
0	telecom	CatBoost	0.6138	0.0804	0.4727	0.1374	0.5667	0.1057	{'model__dept
1	insurance	CatBoost	0.6091	0.334	0.6443	0.44	0.6644	0.2442	{'model__dept

Telecom vs Insurance Comparison



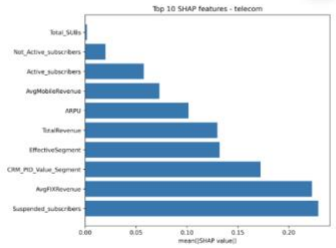
Navigation

Best Model Analysis

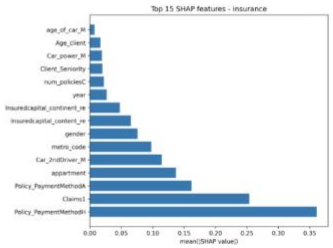
Top 10 Churn Drivers

	Telecom Feature	Telecom Importance	Insurance Feature	Insurance Importance
0	Total_SUBs	0.229	age_of_car_M	0.361
1	Not_Active_subscribers	0.2228	Age_client	0.2537
2	Active_subscribers	0.1724	Car_power_M	0.1618
3	AvgMobileRevenue	0.1321	Client_Seniority	0.1369
4	ARPU	0.1299	num_policiesC	0.1146
5	TotalRevenue	0.1016	year	0.0979
6	EffectiveSegment	0.073	Insuredcapital_continent_re	0.076
7	CRM_PID_Value_Segment	0.0576	Insuredcapital_content_re	0.0652
8	AvgFIXRevenue	0.02	gender	0.0477
9	Suspended_subscribers	0.0018	metro_code	0.0267

SHAP Plots for best models

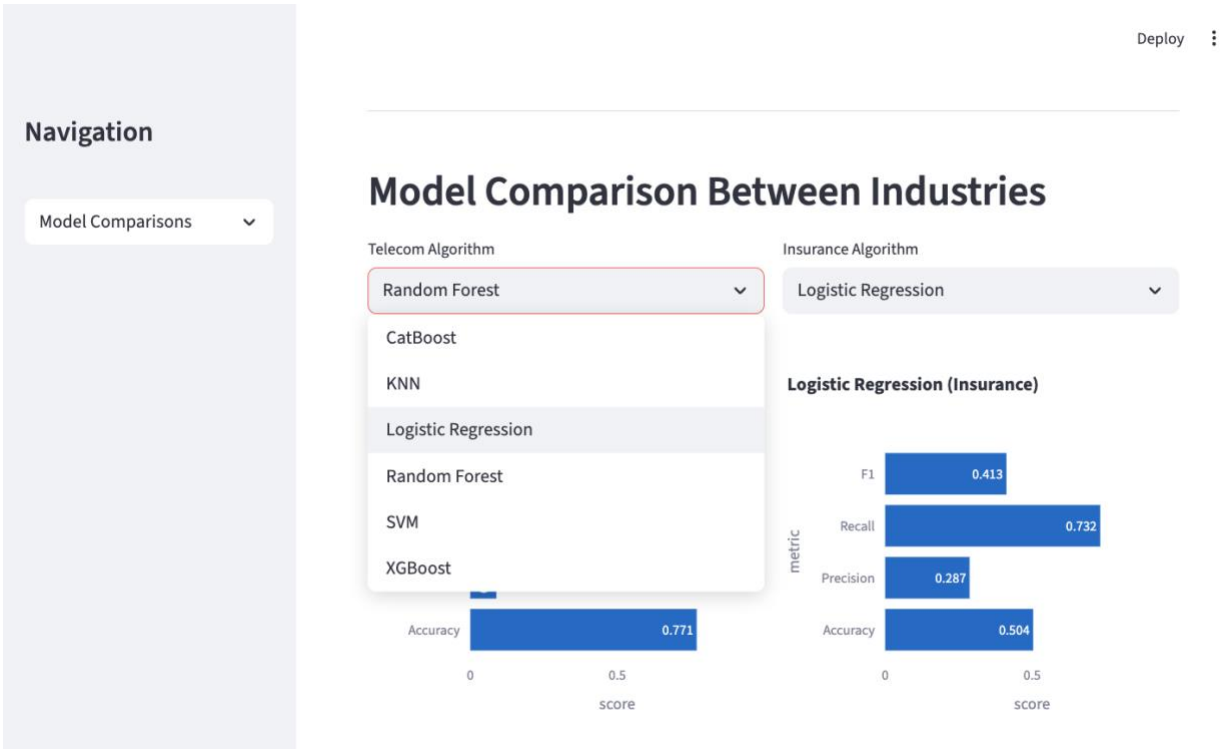


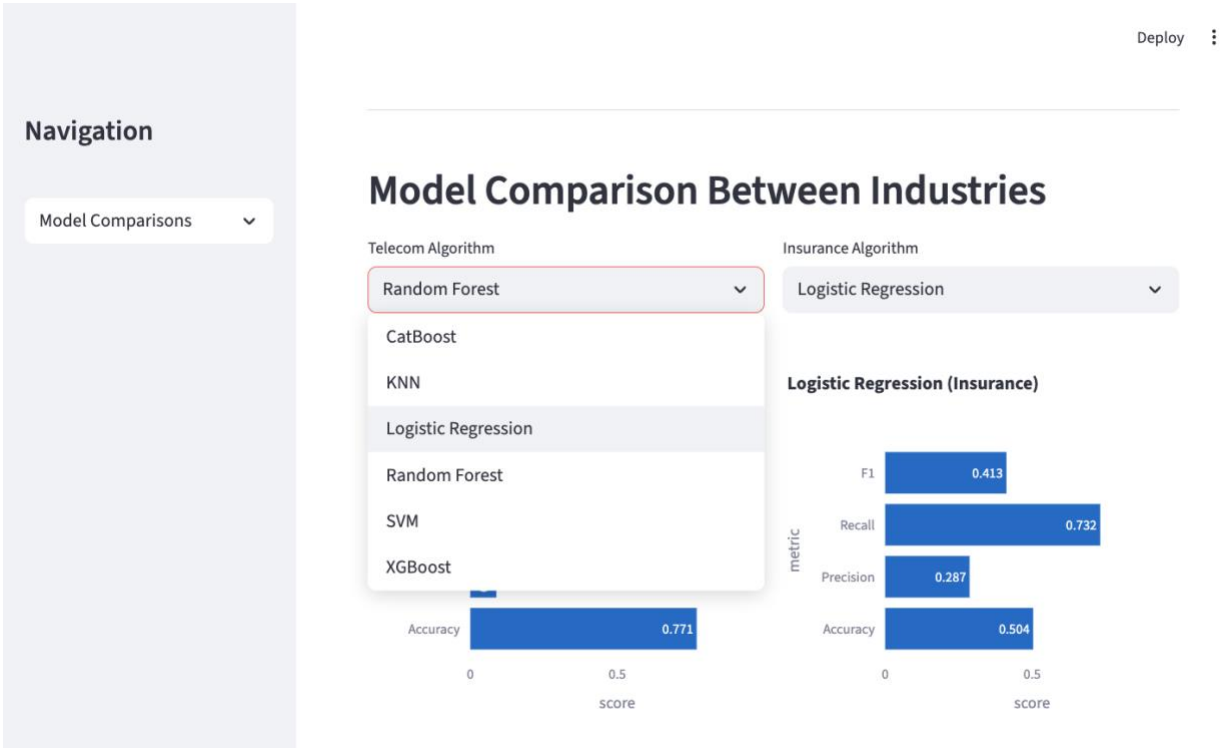
Telecom — SHAP Bar



Insurance — SHAP Bar

Model comparisons page





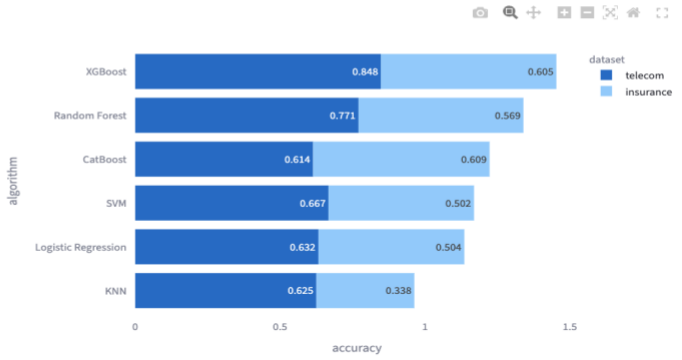
Navigation

Model Comparisons

Deploy

Dataset: Both

Metric: accuracy



	dataset	algorithm	f1_score	accuracy	recall	precision	roc_auc	cv_best_f1	best_p
0	telecom	XGBoost	0.0919	0.848	0.1182	0.0751	0.5609	0.1356	{'mode
1	telecom	Random Forest	0.1339	0.7705	0.2727	0.0888	0.5786	0.1388	{'mode
2	telecom	SVM	0.1217	0.6671	0.3545	0.0734	0.5461	0.1422	{'mode
3	telecom	Logistic Regression	0.1313	0.6322	0.4273	0.0776	0.5454	0.1428	{'mode
4	telecom	KNN	0.1243	0.6251	0.4091	0.0733	0.5325	0.1297	{'mode
5	telecom	CatBoost	0.1374	0.6138	0.4727	0.0804	0.5667	0.1057	{'mode
6	insurance	CatBoost	0.44	0.6091	0.6443	0.334	0.6644	0.2442	{'mode
7	insurance	XGBoost	0.432	0.6055	0.6296	0.3288	0.6609	0.4154	{'mode
8	insurance	Random Forest	0.4143	0.5692	0.6394	0.3064	0.6312	0.4008	{'mode
9	insurance	Logistic Regression	0.4128	0.5037	0.732	0.2874	0.6287	0.4112	{'mode

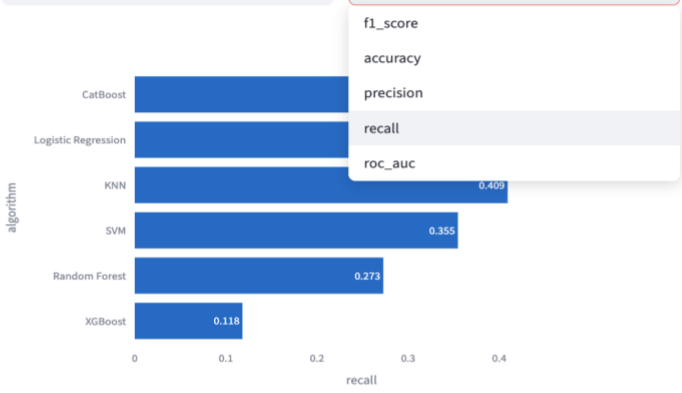
Navigation

Model Comparisons

Deploy

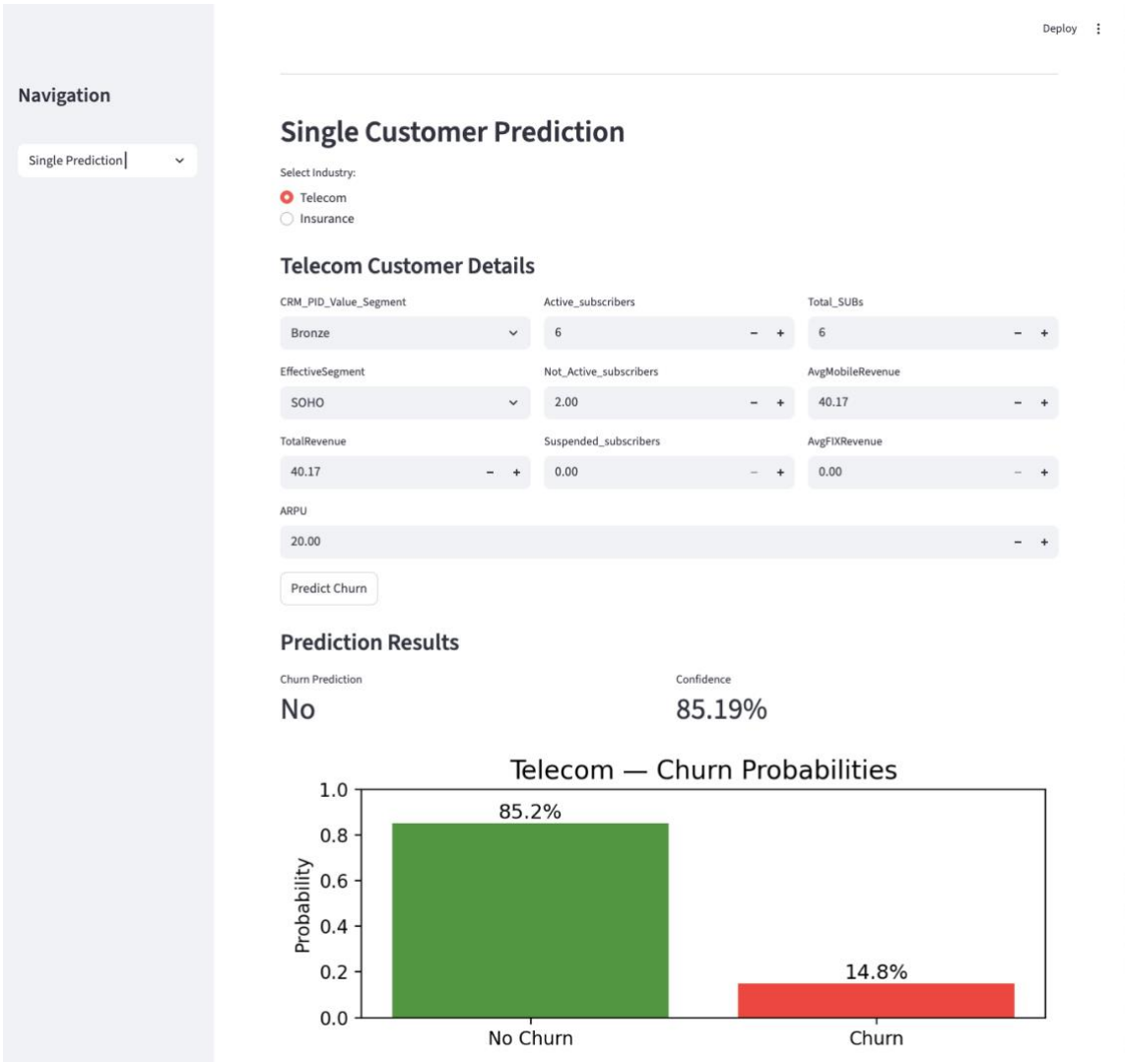
Dataset: Telecom

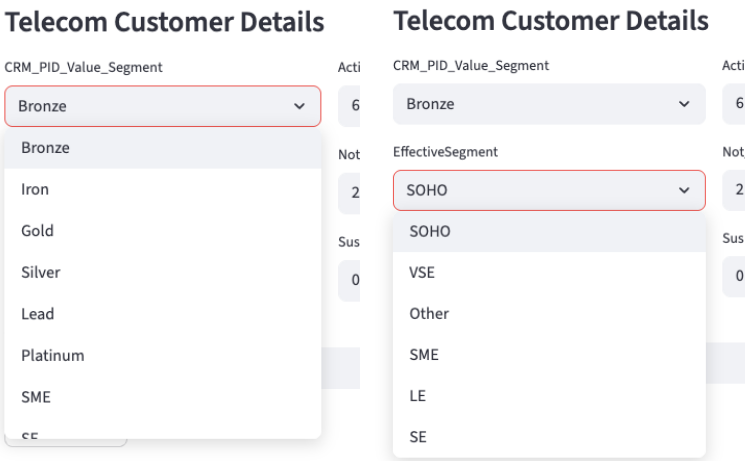
Metric: recall



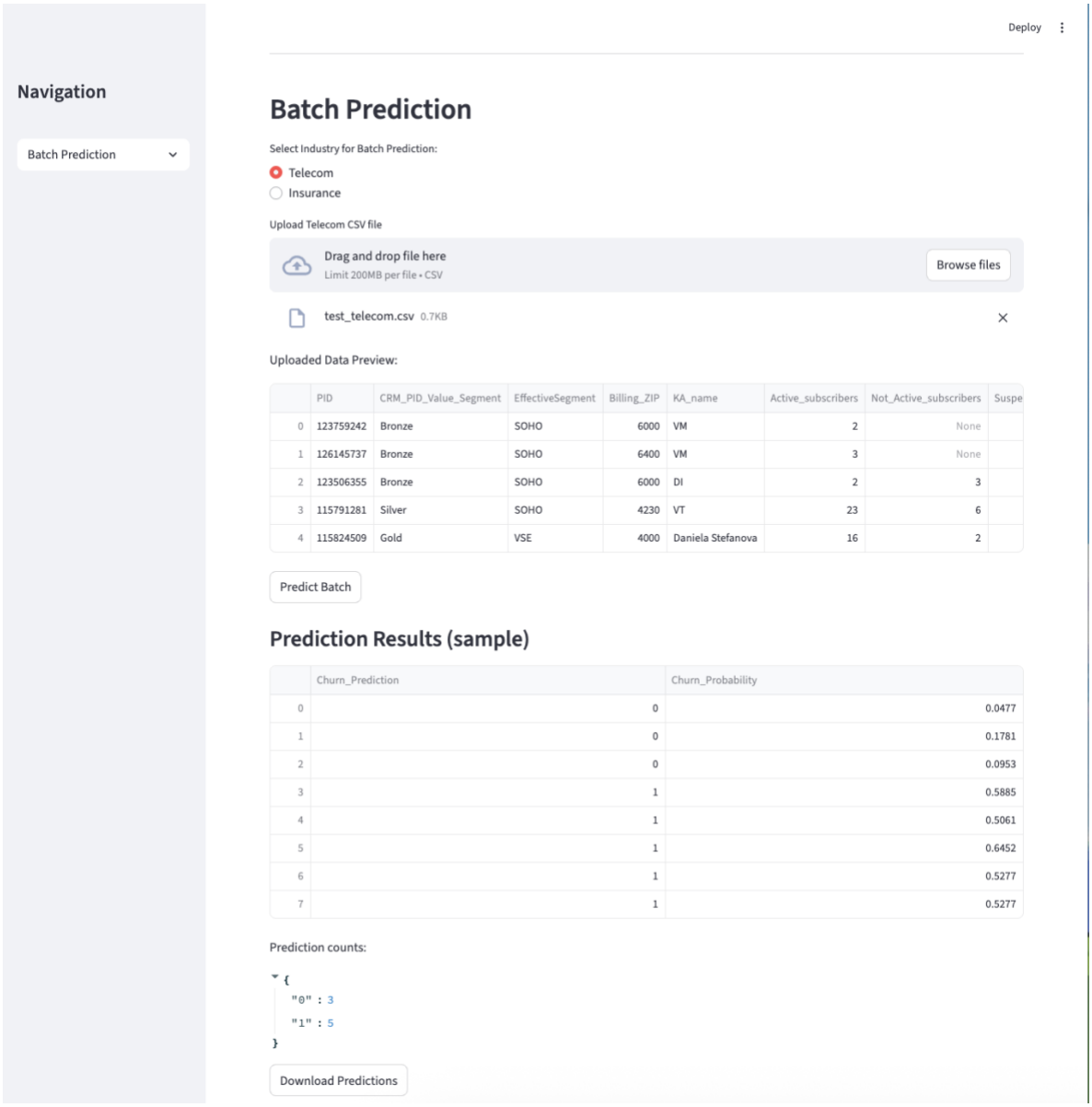
	dataset	algorithm	f1_score	accuracy	recall	precision	roc_auc	cv_best_f1	best_p
0	telecom	CatBoost	0.1374	0.6138	0.4727	0.0804	0.5667	0.1057	{'mode
1	telecom	Logistic Regression	0.1313	0.6322	0.4273	0.0776	0.5454	0.1428	{'mode
2	telecom	KNN	0.1243	0.6251	0.4091	0.0733	0.5325	0.1297	{'mode
3	telecom	SVM	0.1217	0.6671	0.3545	0.0734	0.5461	0.1422	{'mode
4	telecom	Random Forest	0.1339	0.7705	0.2727	0.0888	0.5786	0.1388	{'mode
5	telecom	XGBoost	0.0919	0.848	0.1182	0.0751	0.5609	0.1356	{'mode

Single prediction page





Batch prediction page



Prediction results csv

Appendix C: Feature Description

Table 1 - Insurance Dataset Feature Descriptions

Feature Name	Type	Description	Used in Model
PolID	Identifier	Policy holder identifier	No
gender	Binary	1 = Male 0 = Female	Yes
Age_client	Integer	Customer age in years	Yes
year	Integer	Observation year	Yes
age_of_car_M	Integer	Vehicle age	Yes
Car_power_M	Integer	Vehicle power	Yes
Car_2ndDriver_M	Binary	1 = 2nd driver exists 0 = No 2nd driver	Yes
num_policiesC	Integer	Total policies held	Yes
metro_code	Binary	1 = Urban/Metropolitan 0 = Rural	Yes
Policy_PaymentMethodA	Binary	1 = Annual auto payment 0 = Monthly auto payment	Yes
Policy_PaymentMethodH	Binary	1 = Annual home payment 0 = Monthly home payment	Yes
Insuredcapital_content_re	Float	Home contents value (€)	Yes
Insuredcapital_continent_re	Float	Home building value (€)	Yes
appartment	Binary	1 = Insurance for apartment 0 = Insurance for home	Yes
Client_Seniority	Integer	Customer tenure years	Yes
NClaims1	Integer	Auto claims count	Yes
NClaims2	Integer	Home claims count	Yes
Claims1	Float	Auto claim cost (€)	Yes
Claims2	Float	Home claims cost (€)	Yes
Types	Categorical	1 = No claim 2 = Only auto claim 3 = Only home claim 4 = Both auto and home claim	No
Retention	Binary	Retention: 1 = Policy renewed (No churn) 0 = Policy not renewed (Churn)	No
churn (this feature was added manually to be inverse of retention)	Binary	Churn (inverse of Retention): 1 = Churned 0 = Not Churned	Target

Table 2 - Telecommunications Dataset Feature Descriptions

Feature Name	Type	Description	Used in Model
PID	Identifier	Customer identifier	No
CRM_PID_Value_Segment	Categorical	Customer value tier	Yes
EffectiveSegment	Categorical	Operational customer segment	Yes
Billing_ZIP	Categorical	Customer location code	No
KA_name	Categorical	Account manager name	No
Active_subscribers	Integer	Active subscriber count	Yes
Not_Active_subscribers	Integer	Inactive subscriber count	Yes
Suspended_subscribers	Integer	Suspended subscriber count	Yes
Total_SUBs	Integer	Total subscriber count	Yes
AvgMobileRevenue	Float	Average revenue from mobile services (BGN)	Yes
AvgFIXRevenue	Float	Average revenue from fixed services (BGN)	Yes
TotalRevenue	Float	Total customer revenue (BGN)	Yes
ARPU	Float	Revenue per user (BGN)	Yes
CHURN	Binary	Churn: 1 = Churned 0 = Not Churned	Target

Appendix D: Supplementary Results

Table 1 – Hyperparameter Search Grids

Algorithm	Hyperparameter	Values Explored
Logistic Regression	Regularisation Strength (C)	[0.01, 0.1, 1, 10]
Random Forest	Number of Trees (n_estimators) Maximum Depth (max_depth)	[100, 200] [6, 12, None]
XGBoost	Number of Trees (n_estimators) Tree Depth (max_depth) Learning Rate Positive Class Weight (scale_pos_weight)	[100, 200] [3, 6] [0.01, 0.05, 0.1] [1, 5, 10]
CatBoost	Iteration Count (iterations) Tree Depth (depth)	[150, 300] [4, 6]
SVM	Regularisation Strength (C)	[0.01, 0.1, 1, 10]
KNN	Number of Neighbors (n_neighbors) Weighting Scheme (weights)	[3, 5, 7] ['uniform', 'distance']

Table 2 – Telecom Test Set Results

	CatBoost	Random Forest	Logistic Regression	KNN	SVM	XGBoost
F1-score	0.14	0.13	0.13	0.12	0.12	0.09
Accuracy	0.61	0.77	0.63	0.62	0.67	0.85
Recall	0.47	0.27	0.43	0.41	0.35	0.12
Precision	0.08	0.09	0.08	0.07	0.07	0.07
ROC_AC	0.57	0.58	0.54	0.53	0.55	0.56
CV_F1	0.11	0.14	0.14	0.13	0.14	0.14
Best Params	depth=4, iter=150	depth=6, trees=100	C=10	k=5, uniform	C=0.1	lr=0.05, depth=3
Best Threshold	0.29	0.51	0.49	0.4	0.5	0.49
Train Time (S)	5.5	14.1	5.7	1.5	0.6	8.9

Table 3 – Insurance Test Set Results

	CatBoost	Random Forest	Logistic Regression	KNN	SVM	XGBoost
F1-score	0.44	0.41	0.41	0.39	0.41	0.43
Accuracy	0.61	0.57	0.50	0.34	0.50	0.60
Recall	0.64	0.64	0.73	0.89	0.73	0.63
Precision	0.33	0.31	0.29	0.25	0.29	0.33
ROC_AUC	0.66	0.63	0.63	0.57	0.63	0.66
CV_F1	0.24	0.4	0.41	0.36	0.41	0.42
Best Params	depth=6, iter=150	depth=6, trees=100	C=1	k=7, uniform	C=0.1	lr=0.1, depth=6
Best Threshold	0.25	0.46	0.45	0.14	0.45	0.66
Train Time (S)	34	215.7	11.8	74.8	11	115.5

Table 4 – Telecom SHAP rankings

Feature	SHAP Importance
Total_SUBs	0.23
Not_Active_subscribers	0.22
Active_subscribers	0.17
AvgMobileRevenue	0.13
ARPU	0.13
TotalRevenue	0.10
EffectiveSegment	0.07
CRM_PID_Value_Segment	0.06
AvgFIXRevenue	0.02
Suspended_subscribers	0.001

Table 5 – Insurance SHAP rankings

Feature	SHAP Importance
age_of_car_M	0.36
Age_client	0.25
Car_power_M	0.16
Client_Seniority	0.14
num_policiesC	0.11
year	0.10
Insuredcapital_continent_re	0.08
Insuredcapital_content_re	0.07
gender	0.05
metro_code	0.03
Car_2ndDriver_M	0.02
appartment	0.02
Policy_PaymentMethodA	0.02
Claims1	0.02
Policy_PaymentMethodH	0.01
NClaims1	0.01
Claims2	0.002
NClaims2	0.002