Sri Lanka Institute of Information technology

IT3051 : Fundamental of data mining

FDM_MLB_G03 : Data Detectives

Final report

GitHub repo link : https://github.com/RuviDev/FDM_ChurnPrediction
Software deployment link : https://bankchurnprediction-fdm.streamlit.app/?tab=home

Prepared by ->

Name	ID
A V Amarathunga	IT23192782
M S R V Amararathna	IT23264366
S A R U Amarasinghe	IT23216778
A A Gunawardena	IT23140752

Fundamentals of data mining : IT3051

Credit Card Customers : Predict Churning Customers

Prepared by :
Ashan Amarathunga, Ruvindu Amararathna, Reshika Amarasinghe, Aloka Gunawardana

CONTENT

OBJECTIVE

Our main objective is to make a software for the bank company **Metrics** a financial intuition form that is willing to build a software to identify which existing customers in their system has the highest chance to churn. This document covers the project workflow from the beginning starting from the day we got out dataset confirmed till 5 of October. Each member contributed at their best for the success of the project.

Bank Name : Metrics

Find the Kaggle link from here ->
https://www.kaggle.com/datasets/sakshigoyal7/credit-card-customers/data

01. Introduction

Problem Statement

Customer attrition, commonly referred to as *churn*, is a major challenge in the financial services industry, particularly for credit card providers. When customers stop using their cards or close accounts, the organization faces both **direct financial losses** (reduction in transaction fees, interest, and annual charges) and **indirect costs** such as customer re-acquisition, promotional discounts, and lower opportunities for cross-selling other financial products.

Traditional retention efforts often react **after** churn occurs, which is costly and inefficient. Therefore, being able to **predict churn in advance** allows banks to identify high-risk customers early, implement personalized retention strategies, and optimize marketing investments.

This project focuses on developing a **predictive model** capable of estimating the probability that a customer will churn soon using behavioral, demographic, and transactional features derived from the *BankChurners* dataset. By leveraging supervised learning methods, the project seeks to convert raw customer data into actionable business insights that can improve customer retention rates and long-term profitability.

Objective

The primary objective of this project is to **build, evaluate, and compare multiple supervised machine learning models** for churn prediction. Specifically, the project aims to:

- Develop classification models (e.g., Logistic Regression, Random Forest, XGBoost) to identify customers with a high likelihood of churn.

- Assess model performance using robust metrics such as **ROC-AUC**, **Precision-Recall AUC**, **F1-score**, and **Recall** on the churn (positive) class.
- Analyze and interpret model outputs to understand the **key drivers of churn** and ensure the findings align with real-world domain knowledge—particularly emphasizing behavioral and usage-related patterns over static demographics.
- Provide recommendations for **operational deployment**, allowing marketing or CRM teams to prioritize interventions for customers most at risk.

Success Criteria

The success of the project will be determined by a combination of **quantitative model performance** and **qualitative interpretability**:

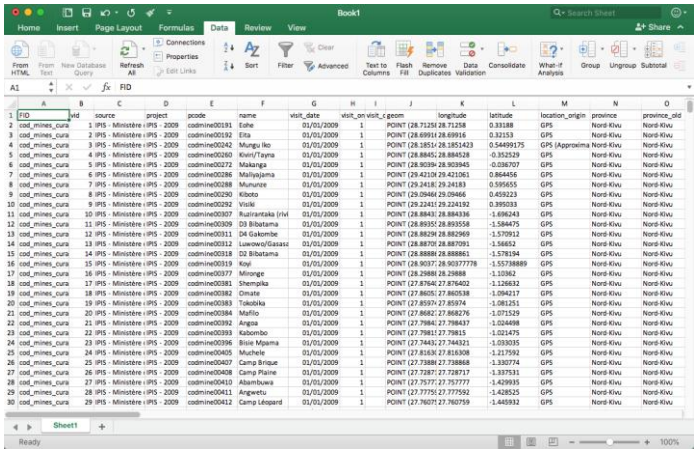
- **Predictive Accuracy:** Achieve high **ROC-AUC** and **PR-AUC** scores on a hold-out test dataset, demonstrating strong discrimination between churners and non-churners.
- **Operational Usefulness:** Deliver competitive **F1-scores and Recall** for the churn class compared to baseline or benchmark models, ensuring practical usability for retention campaigns.
- **Interpretability & Insights:** Extract and visualize feature importance to provide clear, explainable insights into which factors most influence churn decisions. The top predictors should align with business intuition, emphasizing customer activity, transaction frequency, and account behavior rather than purely demographic attributes.
- **Generalizability:** Ensure consistent model performance across validation folds and unseen test data, confirming robustness for real-world deployment.

02. Dataset Manual Check-up

Find the dataset from here -> https://www.kaggle.com/datasets/sakshigoyal7/credit-card-customers/data

Source and Shape

The dataset used for this project is the **BankChurners.csv**, obtained from a publicly available source on **Kaggle**. It contains approximately **10,000 customer records**, each representing an individual credit card account. The dataset comprises a **mixture of numerical and categorical attributes** that describe customer demographics, relationship tenure, product holdings, transaction behavior, and account activity metrics.



Each row corresponds to a unique customer, and the columns collectively capture factors that may influence customer retention or attrition, such as credit utilization, card category, transaction frequency, and income category. The dataset provides a well-balanced representation of operational and behavioral characteristics suitable for supervised classification.

Target Definition

The original dataset label is **Attrition_Flag**, which has two possible values:

- "Attrited Customer" – representing customers who have closed or stopped using their cards.

- "Existing Customer" – representing currently active cardholders.

For modeling purposes, this label was transformed into a **binary target variable** named **Churn**,

This transformation simplifies downstream model training and evaluation by converting the churn problem into a **binary classification task**, where 1 represents churned customers and 0 represents retained ones.

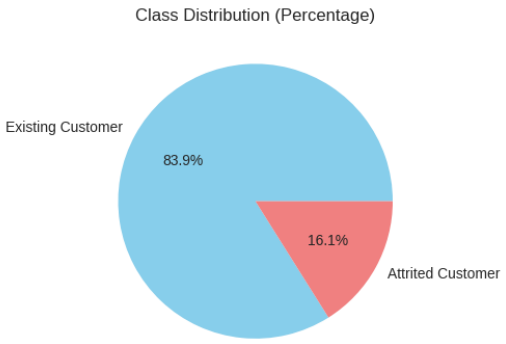
Manual Quality Checks (Performed Before Modeling)

Before feature engineering and model training, several **manual quality assurance checks** were carried out to ensure data integrity and prevent information leakage:

- **Leakage & Identifiers:**
  The column **CLIENTNUM**, which serves only as a unique identifier, was removed as it holds no predictive value. Additionally, two columns beginning with **Naive_Bayes_Classifier_...** were excluded. These columns were identified as **metadata artifacts** generated by Kaggle example notebooks and could lead to **data leakage** if retained.
- **Missingness:**
  A review of null counts confirmed that the dataset contains **no significant missing values**. As a result, no imputation techniques (e.g., SimpleImputer) were necessary in the preprocessing stage of the notebook.
- **Class Imbalance:**
  The **positive class (Churn = 1)** was found to be a **minority**, indicating a moderate class imbalance. This imbalance was explicitly considered in the **evaluation metric selection** (favoring ROC-AUC, PR-AUC, and F1-score) and in **resampling strategies** applied during model development to ensure fair and robust performance.
- **Derived Fields (Used When Applicable):**
  Two additional derived metrics were computed to enhance interpretability:
  - **Avg_Open_To_Buy** = Credit_Limit - Total_Revolving_Bal
  - **Avg_Utilization_Ratio** = Total_Revolving_Bal / Credit_Limit

Impact

The removal of identifier and leakage columns, along with the confirmation of minimal missingness, significantly reduced the **risk of optimistic bias** in the model. This simplification also streamlined preprocessing by eliminating the need for imputation or complex cleaning steps in the Jupyter notebook (.ipynb).



Addressing class imbalance early ensured that model selection and metric interpretation remained aligned with the project's operational goal: **maximizing the recall of churned customers while maintaining strong overall**

discrimination performance.

03. Project Structure (High-Level)

The project follows a structured and modular workflow designed to ensure **data integrity, interpretability, and reproducible experimentation**. Each stage in the pipeline—from data ingestion to final evaluation—was implemented sequentially within Jupyter Notebook (.ipynb) cells to maintain clear traceability of analytical decisions.

Section 01 -> Data Loading & Exploration

The workflow begins with **data ingestion**, where the BankChurners.csv file is read into a Pandas DataFrame. Initial exploration includes verifying column names, ensuring consistency, and applying basic **column hygiene** (removal of spaces, standardizing names, and dropping non-predictive columns such as CLIENTNUM).

The **target variable (Attrition_Flag)** is then **mapped** into a binary format (Churn = 1 for "Attrited Customer", 0 otherwise). Summary statistics and data types are reviewed to confirm correct parsing of categorical and numerical fields before proceeding to further analysis.

Section 02 -> Exploratory Data Analysis (EDA)

This stage aims to uncover **patterns, anomalies, and early signals** of customer churn. Key steps include:

- **Univariate Analysis:** Visualization of feature distributions (histograms, count plots) to assess skewness, outliers, and value ranges.
- **Bivariate Relationships:** Comparison of churn rate across categorical variables (e.g., gender, card category, income range) and continuous variables (e.g., credit limit, transaction amounts).
- **Correlation Matrix:** Calculation of correlation coefficients among numeric variables to detect redundant features and multicollinearity.
- **Churn Rate Insights:** Examination of feature-level churn proportions to identify high-risk behavioral segments.
- **Preliminary Feature Importance:** Using simple models (e.g., Random Forest or Logistic Regression coefficients) to get an early sense of influential predictors.

This analysis provided a strong foundation for feature selection and guided later preprocessing and model tuning decisions.

Section 03 -> Data Preprocessing

The preprocessing stage ensures all features are **numerically and statistically ready** for machine learning models. The key steps include:

- **Categorical Encoding:**
  Applied label or one-hot encoding depending on model requirements. Tree-based models (e.g., Random Forest, XGBoost) efficiently handle label-encoded features, while linear models rely on dummy-variable expansion.
- **Feature Scaling:**
  Minimal scaling was applied to numerical variables only where necessary (e.g., for algorithms sensitive to feature magnitude such as Logistic Regression or Linear SVC).
- **Imbalance Handling:**
  As the dataset exhibited class imbalance, strategies such as **class weighting** and **scale_pos_weight** (for

gradient boosting) were employed. In some configurations, **SMOTE (Synthetic Minority Oversampling Technique)** was tested to augment minority samples during training.

- **Simplified Pipeline:**
  No imputation (SimpleImputer) or probability calibration steps were used, as missingness was negligible and probabilistic smoothing was not required for the main evaluation metrics.

Section 04 -> Model Evaluation & Selection

A broad range of **supervised classification algorithms** was implemented and compared, covering both linear and ensemble-based learners:

- **Linear Models:** Logistic Regression (LR), Linear SVC
- **Tree-based Models:** Random Forest (RF), Gradient Boosting (GBC), AdaBoost, XGBoost, LightGBM

Each model was evaluated using stratified training-test splits to maintain class balance and reproducibility. The experiments also compared models trained **with and without SMOTE** to assess the impact of oversampling on recall and F1 performance.

Key metrics included **ROC-AUC**, **PR-AUC**, **F1-score**, and **Recall** on the churn class. Visualization tools such as ROC and Precision-Recall curves were used to highlight discrimination performance and threshold trade-offs.

Section 05 -> Training & Testing Workflow

The final stage involved **parameter tuning, model fitting, and performance validation**.

- A **limited grid search** was applied to Gradient Boosting to optimize key hyperparameters (e.g., learning rate, number of estimators).
- A **larger grid search** was initially explored for XGBoost but later constrained to reduce runtime.
- The best-performing models were re-trained on the full training set and evaluated on the **hold-out test split**.

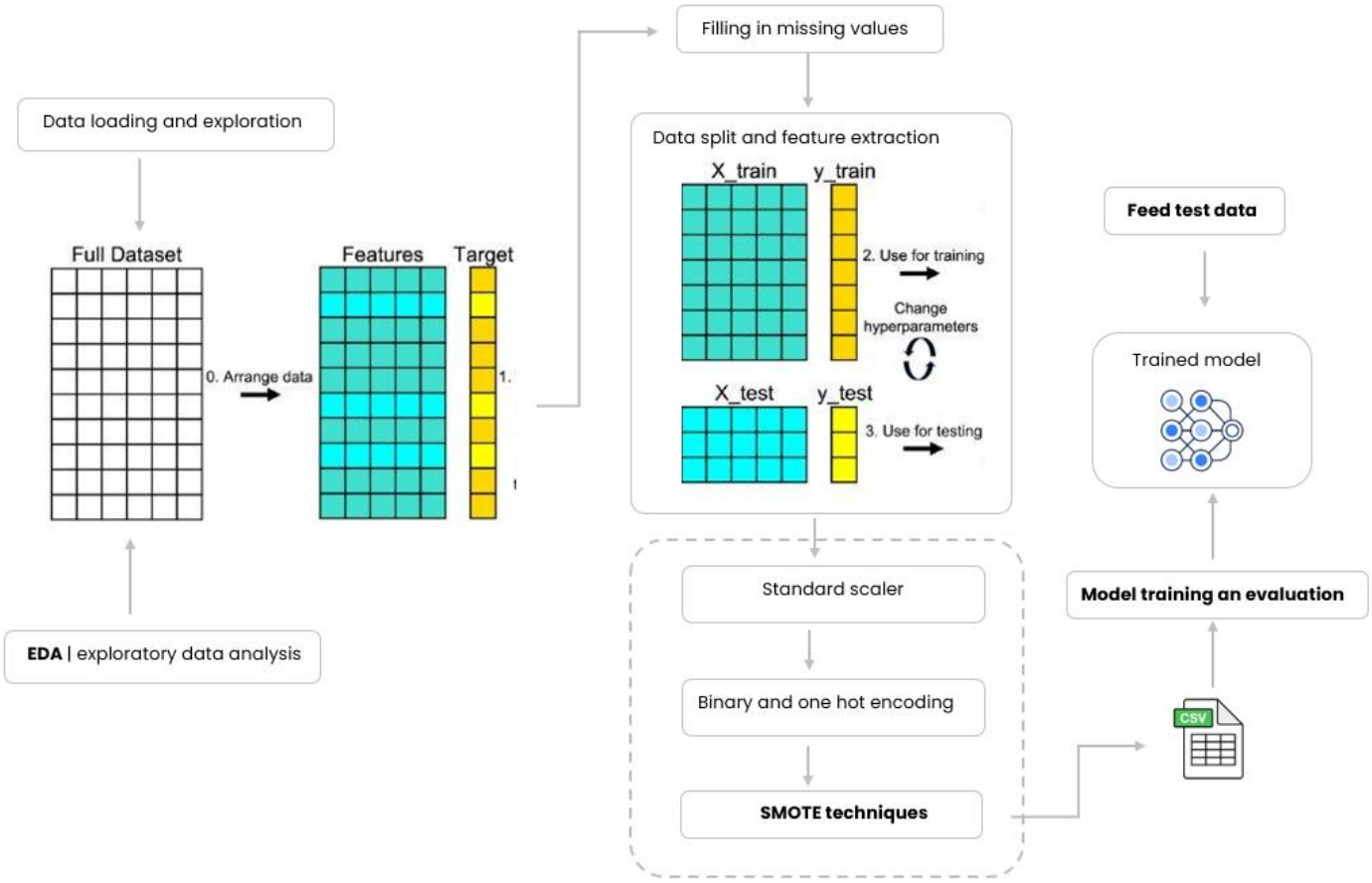
Comprehensive evaluation included:

- ROC and PR curve generation for discrimination analysis.
- F1/Recall comparison across candidate models.
- Predictions on unseen data to confirm generalization ability and business readiness.

This structured pipeline ensured that model performance was both **statistically sound** and **operationally viable** for deployment in real-world churn management systems.

Building pipeline -> Architecture diagram

Now we will explore each section carefully and understand the complete building process. The pipeline can be illustrated as follows:



Section 01 — Data Loading & Exploration

The purpose of this stage is to establish a **clean, reproducible, and analysis-ready dataset** suitable for supervised machine learning. All subsequent Modeling steps depend on a well-prepared foundation with consistent feature definitions, reliable target encoding, and the removal of potential leakage or irrelevant identifiers. This section ensures that the dataset structure and preprocessing logic are **fully deterministic**—meaning the same code can be rerun to obtain identical results for validation or deployment.

Steps (Logic and Implementation Summary)

1. **Load Dataset**
   The raw CSV file is read into a Pandas DataFrame:
2. **Remove Non-Predictive or Leaky Columns**
   - **CLIENTNUM** — dropped because it serves purely as a unique customer identifier and holds no predictive signal.
   - **Columns with prefix Naive_Bayes_Classifier_** — removed because these are **Kaggle artifact columns**, likely representing meta-scores or prior probability estimates that could **leak target information** into the model.

3. **Create Binary Target Variable**
   The target column Attrition_Flag is transformed into a binary numerical variable named **Churn**. This simplifies the prediction problem to a binary classification task (1 = churned, 0 = active).

4. **Validate Dataset Integrity**
   Basic dataset diagnostics are performed:

   - **Shape Check:** Verify total rows and columns after dropping unnecessary fields.
   - **Data Types:** Ensure categorical and numerical variables are properly typed.
   - **Class Distribution:** Examine balance between churned and non-churned classes using value_counts().
     These checks confirm data consistency and help in planning imbalance mitigation strategies later.

5. **Train-Test Split (Stratified)**
   The dataset is split into **training and testing subsets** using stratification to preserve the original churn proportion: This ensures fair model evaluation by maintaining class distribution consistency between splits.

Outcome

At the end of this phase, a **clean and structured base table** is established, containing only valid predictive features and a clearly defined binary target (Churn). The dataset is free from leakage and unnecessary identifiers, ensuring unbiased modeling.

This consistent data foundation serves as the entry point for all downstream processes, including **exploratory analysis, feature engineering, and model training**.

Section 02 — Exploratory Data Analysis

Purpose

The Exploratory Data Analysis (EDA) phase was conducted to **uncover behavioral patterns, detect outliers, and validate domain hypotheses** about the drivers of customer churn.
The primary assumption guiding this analysis was that **customer activity metrics** (such as transaction frequency and amount changes) would be **more predictive of churn** than static demographic factors like age, gender, or income category.

Through visual exploration, statistical summaries, and correlation inspection, this step helps confirm whether churn behavior is indeed rooted in **engagement decline and inactivity signals** rather than static personal attributes.

EDA Activities and Key Findings

1. Distributions by Churn
Feature distributions were plotted separately for churned and retained customers to visually assess behavioral differences.
The following trends were observed:

- **Total_Trans_Ct** and **Total_Trans_Amt** were **significantly lower** for churned customers, indicating reduced transaction frequency and monetary engagement.
- **Total_Ct_Chng_Q4_Q1** and **Total_Amt_Chng_Q4_Q1**, representing quarter-over-quarter activity change, were also **lower for churners**, suggesting a **decline in card usage** prior to attrition.

- **Months_Inactive_12_mon** tended to be **higher** among churned customers, implying longer inactivity periods.
- **Contacts_Count_12_mon** was often **higher** for churners, possibly reflecting **increased service interactions before leaving**, such as complaint calls or account closure inquiries.

These distribution analyses confirm that **activity intensity** and **recency of engagement** are central churn signals.

2. Outlier Inspection (Box / Violin Plots)

Boxplots and violin plots were used to visualize distribution spread and detect outliers, particularly in transaction-related variables.
Findings include:

- Heavy-tailed distributions for **transaction amount and count** across both groups.
- Churners clustered toward the **lower quartiles**, confirming that most attrited customers fall within the **low-activity tail** of these variables.
- No evidence of extreme outliers that would require removal, allowing raw behavioral diversity to be preserved for modeling.

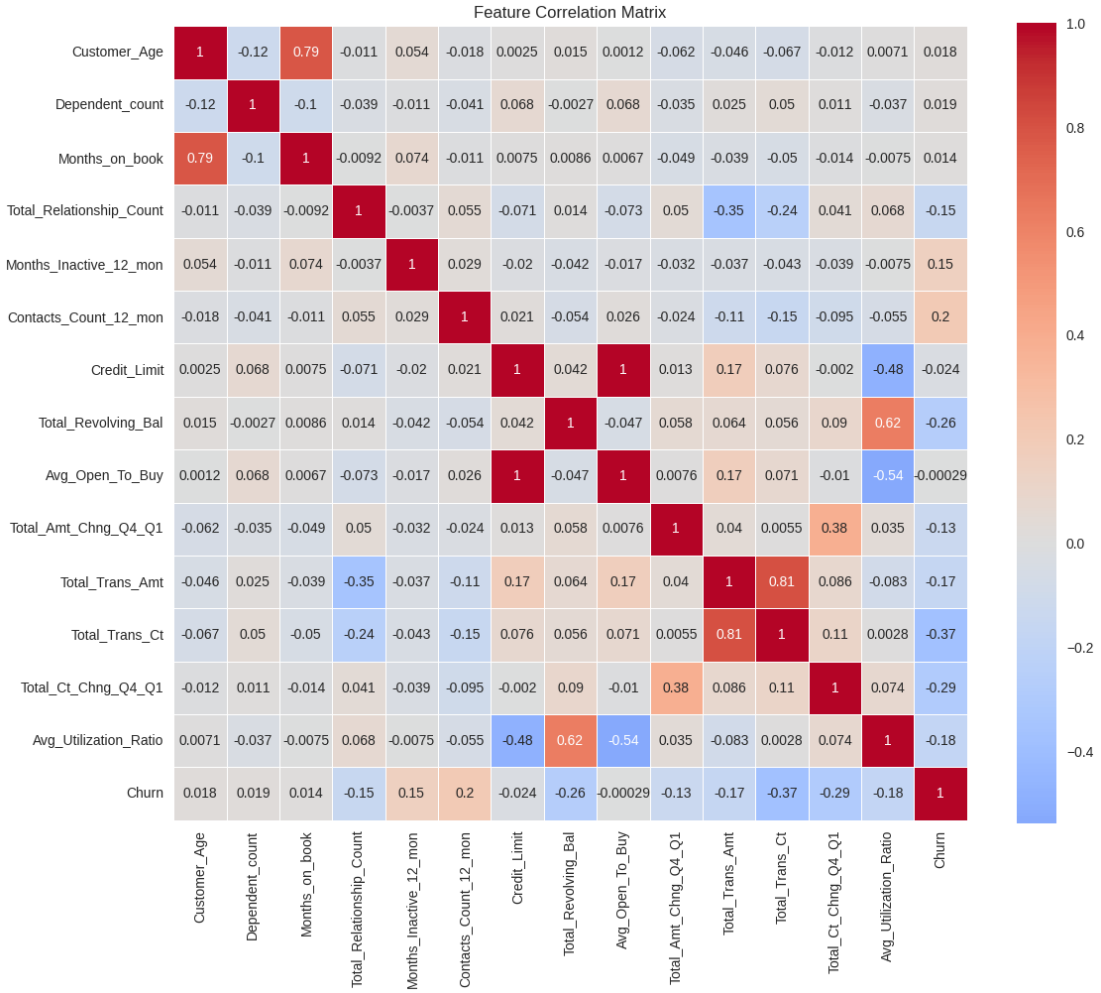
3. Churn Rate vs. Features

- **Numeric Variables (Binned Analysis):**
  When features such as Total_Trans_Ct or Total_Trans_Amt were discretized into bins, the **churn rate showed a clear inverse relationship**—as activity increases, the probability of churn decreases.
- **Categorical Variables (Grouped Analysis):**
  Grouped bar plots revealed that customers with the **"Blue" card type** and **low-income categories** exhibit **higher churn rates**, especially when combined with low transaction activity.
  This supports the idea that lower-value, low-engagement customer segments are at higher risk.

4. Correlation Analysis (Heatmap)

A correlation heatmap including the binary Churn variable was computed to assess relationships among numeric features.
Observations:

- Stronger negative correlations were found between churn and **activity-based metrics** (Total_Trans_Ct, Total_Amt_Chng_Q4_Q1, etc.), reinforcing the behavioral link to attrition.
- Demographic variables (e.g., Customer_Age, Gender, Income_Category) showed **weak or negligible correlations**, indicating limited predictive power.



5. Preliminary Feature Importance (Tree-Based Estimate)

A simple tree-based model (e.g., Random Forest or Gradient Boosting) was trained to obtain **relative feature importance scores**.

Top-ranked churn predictors included:

- **Total_Trans_Ct** — transaction frequency
- **Total_Ct_Chng_Q4_Q1** — quarter-over-quarter change in transaction count
- **Total_Amt_Chng_Q4_Q1** — quarter-over-quarter change in transaction amount
- **Months_Inactive_12_mon** — inactivity duration
- **Contacts_Count_12_mon** — number of service contacts
- **Utilization metrics** such as Avg_Utilization_Ratio

These findings quantitatively confirm that **engagement behavior** and **credit usage patterns** are the dominant factors driving churn, while static profile attributes contribute minimally.

Section 03 — Data Pre-processing

Purpose

The purpose of this stage is to **transform the cleaned dataset into model-ready numeric features**, ensuring that all inputs are compatible with the supervised algorithms used while maintaining fairness and generalization.Special attention was given to **class imbalance**, **scaling**, and **encoding**, as these directly affect model performance and stability.

It is important to note that we did **not apply imputation or probability calibration**.
This design choice was **intentional** because:

- The dataset exhibited **minimal or no missingness**, removing the need for SimpleImputer.
- The experimental focus was on **model discrimination** (ROC-AUC, PR-AUC) and **F1 performance**, not calibrated probabilities.

Techniques Used

1. Categorical Encoding
To convert categorical attributes into numeric representations:

- **One-Hot Encoding:**
  Applied for models that require purely numerical inputs such as **Logistic Regression**, **Linear SVC**, and **some Gradient Boosting implementations**.
  Encoding expands each categorical feature into a binary vector of indicators.
- **Compatibility Handling:**
  The parameter handle_unknown='ignore' (or equivalent logic) was enabled to prevent inference-time errors when unseen category values appear during testing or deployment.

This step ensured a consistent feature space between the training and test datasets.

2. Feature Scaling (Selective)
Scaling was selectively applied to maintain computational stability for models sensitive to magnitude differences:

- **StandardScaler** was used for algorithms such as **Logistic Regression** and **Linear SVC**, which assume features centered around zero with unit variance.
- Tree-based models (e.g., **Random Forest**, **Gradient Boosting**, **XGBoost**, **LightGBM**) **did not require scaling**, since they rely on relative thresholds and data splits rather than Euclidean distances.

This selective approach preserved interpretability and computational efficiency without unnecessary transformations.

3. Class Imbalance Handling
Customer churn represented a **minority class**, making imbalance control essential for fair learning. The following strategies were employed depending on the model type:

- **Class Weights (for Linear Models):**
  - class_weight='balanced' was applied in **Logistic Regression** and **Linear SVC** to automatically adjust loss contributions based on inverse class frequencies.
  - This ensured that minority class errors (churn misclassifications) carried proportionally higher penalty during optimization.
- **Balanced Random Forest:**
  Either the "balanced" parameter or adjusted class_weight values were used to bias tree splits toward churned customers, improving recall without oversampling.
- **XGBoost's scale_pos_weight:**
  Computed as the ratio (negative_samples / positive_samples) within the training split.

This parameter **increases the gradient impact** of positive samples (churners) during training while keeping data unchanged—an efficient alternative to oversampling.

- **SMOTE (Synthetic Minority Oversampling Technique):**
  As an experimental trial, **SMOTE** was applied on the training split to generate synthetic churn samples for **RF**, **XGB**, and **LGBM** models.
  It aimed to improve minority representation by interpolating new samples between existing churn points.

Section 04 — Model Evaluation & Selection

Candidate Models

A diverse set of **supervised learning algorithms** was implemented and compared to identify the most effective approach for predicting customer churn.
Each model was tuned and evaluated using consistent data splits, metrics, and preprocessing strategies.

The candidate models included:

- **Logistic Regression (balanced)** — baseline linear classifier with class weighting to offset imbalance.
- **Random Forest (balanced)** — ensemble of decision trees using bootstrapped samples and weighted splits.
- **Gradient Boosting Classifier (GBC)** — sequential additive model minimizing exponential loss.
- **AdaBoost** — adaptive boosting model emphasizing misclassified samples in successive iterations.
- **XGBoost** — optimized gradient boosting algorithm with built-in handling for class imbalance (scale_pos_weight).
- **Linear SVC (balanced)** — linear Support Vector Machine using class weighting and hinge loss.
- **LightGBM** — gradient boosting framework leveraging leaf-wise tree growth for improved efficiency.

These models collectively cover both **linear and non-linear** learners, ensuring a comprehensive comparison between interpretable baselines and high-performing ensemble methods.

Evaluation Protocol

All candidate models were evaluated under a **standardized and reproducible testing framework**, consistent with the logic implemented.

Train/Test Setup

- **Stratified Split:** The dataset was partitioned into training and testing subsets using stratified sampling to preserve the original churn ratio.
- **No Probability Calibration:** To ensure consistency and focus on discrimination performance, no post-training calibration (e.g., Platt scaling or isotonic regression) was applied.
- **Hyperparameter Tuning:**
  - Limited grid searches were performed for Gradient Boosting and XGBoost.
  - Broader sweeps were computationally constrained but targeted at learning rate, number of estimators, and depth parameters.

Evaluation Metrics

Each model was evaluated using the following metrics, focusing primarily on the **churn (positive) class**:

- **Precision, Recall, and F1-score** — to assess operational relevance and classification balance.
- **ROC-AUC** and **PR-AUC (when available)** — to evaluate overall discrimination across thresholds.
- **Decision Threshold:** Predictions were converted to binary labels using a **0.50 cutoff** on predicted probabilities or equivalent decision function outputs.

This consistent evaluation framework ensured fairness and comparability across all models and imbalance-handling configurations.

Results Summary

-> *Without SMOTE (using only native class weighting and imbalance parameters)*

Model	Precision (Churn)	Recall	F1-Score (Churn)	AUC
Logistic Regression (Balanced)	0.521181	0.831967	0.640848	0.917348
Random Forest (Balanced)	0.916883	0.723361	0.808706	0.984568
Gradient Boosting Classifier	0.931604	0.809426	0.866228	0.98751
AdaBoost Classifier	0.892449	0.799108	0.843243	0.980832
XGBoost Classifier	0.905544	0.903689	0.904615	0.992149
Linear SVC (Balanced)	0.518614	0.827869	0.637727	N/A
LightGBM Classifier	0.877470	0.909836	0.893360	0.992796

-> *With SMOTE (synthetic oversampling applied to training data)*

Model	Precision (Churn)	Recall	F1-Score (Churn)	AUC
Random Forest (SMOTE)	0.863539	0.829918	0.846395	0.983004
XGBoost (SMOTE)	0.915074	0.883197	0.898853	0.990843
LightGBM (SMOTE)	0.903564	0.883197	0.893264	0.991210

The highest-performing models—**XGBoost** and **LightGBM**—consistently achieved state-of-the-art AUC values above 0.99, confirming exceptional discriminative power.

Selection Rationale

The **primary selection criterion** was **operational F1-score** on the **churn (positive) class**, ensuring that the final model achieves a strong balance between **recall** (capturing churners) and **precision** (avoiding false positives).

Key reasoning points:

- **XGBoost (without SMOTE)** achieved the **best overall F1-score (≈0.905)** while maintaining an **AUC ≈ 0.992**, placing it among the top-performing models in discrimination and operational balance.
- **LightGBM** achieved a slightly higher AUC (≈0.993) but with a marginally lower F1-score, reflecting slightly reduced precision despite excellent recall.
- Given the business objective—**prioritizing accurate churn detection without excessive false positives**—XGBoost demonstrated the **most stable and deployable performance**.

<----------Therefore, XGBoost (no SMOTE) was selected as the final model for the pipeline---------->

Section 05 — Model Training & Testing

Hyperparameter Tuning

Model tuning was conducted to identify a configuration that achieved strong predictive performance while maintaining runtime efficiency and generalization. Two key models—**Gradient Boosting** and **XGBoost**—were prioritized for tuning, given their superior baseline results in earlier experiments.

- **Gradient Boosting (GBC)**
  A **compact grid search** (approximately eight parameter combinations) was tested, focusing on the most influential hyperparameters:

  - n_estimators
  - learning_rate
  - max_depth

  The goal was to balance bias–variance trade-offs and training runtime. Results showed that moderate learning rates (0.05–0.1) and medium tree depths (3–5) provided consistent performance without overfitting.

- **XGBoost**
  For XGBoost, a **comprehensive grid search** (~1,944 candidate configurations) was originally designed, spanning key structural and regularization parameters:

Parameter	Range Tested
n_estimators	100–500
max_depth	3–8
learning_rate	0.03–0.1
subsample	0.8
colsample_bytree	0.6–1.0
min_child_weight	1–10
reg_lambda	0.5–2.0
gamma	0–1.0
scale_pos_weight	computed dynamically (negatives / positives)

  Given 5-fold cross-validation across 1,944 combinations, this would require approximately **9,720 model fits**, making the full grid **computationally expensive**.

  To ensure practicality, the final configuration adopted a **strong baseline parameter set** derived from earlier trials:

```
'classifier_n_estimators':    [300, 600],
'classifier_learning_rate':   [0.05, 0.10],
'classifier_max_depth':       [3, 4],
'classifier_min_child_weight': [1, 3],
'classifier_subsample':       [0.8],
'classifier_colsample_bytree': [0.8],
'classifier_reg_lambda':      [1.0, 5.0],
```

'classifier_gamma': [0],
'classifier_scale_pos_weight': [pos_weight],

Best XGBoost parameters found:

'colsample_bytree': 0.8,
'gamma': 0,
'learning_rate': 0.1,
'max_depth': 4,
'min_child_weight': 1,
'n_estimators': 600,
'reg_lambda': 1.0,
'scale_pos_weight': np.float64(5.223002633889377),
'subsample': 0.8

This tuned configuration achieved **high and stable out-of-sample F1 and AUC**, with minimal overfitting risk.

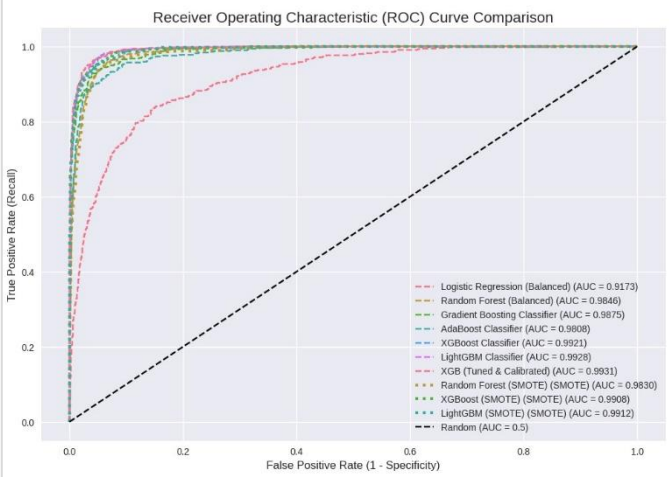
- **Rationale**
  Further expanding the grid provided **diminishing returns** in metric improvement while increasing the risk of validation overfitting.
  Hence, the chosen configuration emphasizes **stability and generalization**, supported by **model-native imbalance control** (scale_pos_weight) rather than synthetic resampling (SMOTE).

Visualization (ROC & PR Curves)

The evaluation visuals confirmed that the final XGBoost model achieved excellent discrimination between churned and active customers

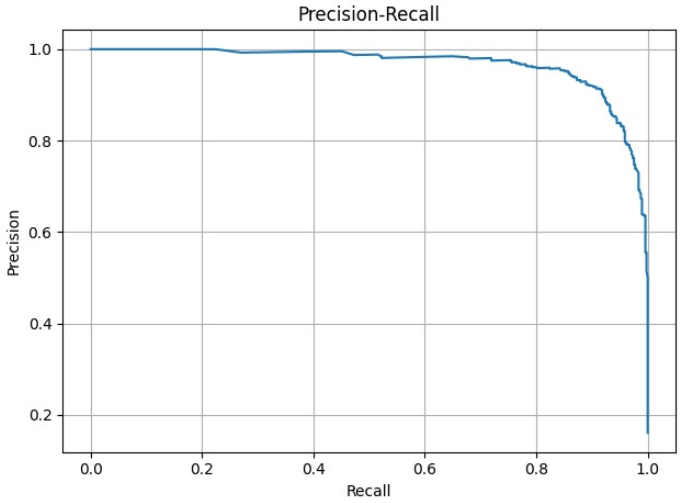
ROC Curve (Receiver Operating Characteristic):

The curve lies close to the **top-left corner**, indicating strong true-positive performance and minimal false positives.**AUC ≈ 0.992** on the hold-out set, reflecting near-perfect ranking capability.



PR Curve (Precision–Recall):

When plotted, the PR curve exhibited a **high area under the curve**, reinforcing robust performance on the **positive (churn)** class under imbalance conditions.Precision remained consistently high even as recall increased, confirming effective handling of minority-class sensitivity.



Predicting on New Data

The trained XGBoost model supports both **single-record** and **batch inference** modes, allowing straightforward extension for real-world applications.

Single Record Prediction

- Input: a structured dictionary or DataFrame row containing all required features.
- Optional: compute any **derived fields** such as Avg_Open_To_Buy or Avg_Utilization_Ratio if used during training.
- Output:
    - **p(churn):** predicted probability of attrition.
    - **label:** binary output (1 if p ≥ 0.5, else 0).

This process allows analysts to score individual customers interactively within the notebook.

Batch Prediction

- A CSV file of customer records is passed through the same preprocessing pipeline.
- Model outputs **p(churn)** and **predicted labels** for each record.
- Results can be exported as a table for downstream actions such as marketing prioritization or CRM tagging.

Index	CLIENTNUM	churn_probability	predicted_label
0	100001	1.000000	Attrited
1	100002	1.000000	Attrited
19	100020	1.000000	Attrited
2	100003	0.993866	Attrited
3	100004	0.905206	Attrited
5	100006	0.828016	Attrited
9	100010	0.524791	Attrited
4	100005	0.524773	Attrited
14	100015	0.326869	Existing
6	100007	0.245997	Existing

Contribution by each member

1. Amarathunga A.V. -> IT23192782

Role overview: Lead the problem framing and define what success means for the business.
Key responsibilities:

- Chose and validated the public Kaggle **BankChurners** dataset (over 10,000 rows, no missing values, no privacy issues).
- Wrote the **business goals**: predict churn probability, act early, reduce revenue loss.

- Set the **success metrics**: ROC-AUC, PR-AUC, and **recall for churners**.
- Prepared the **Statement of Work (SOW)** and kept a **backup dataset** ready.

2. Amararathna M.S.R.V -> IT23264366

Role overview: Build the data and model pipeline; select the stack and produce metrics.
Key responsibilities:

- Built the pipeline in **Google Colab** using **Python**, **pandas/NumPy**, and **scikit-learn**.
- Cleaned data; **dropped leakage columns** and **IDs**; set up a **ColumnTransformer** for numeric and categorical features.
- Handled class imbalance with **class weights** and **SMOTE**; trained **Logistic Regression**, **Random Forest**, and **XGBoost**.
- **Calibrated** probabilities with **CalibratedClassifierCV (hist)** to make risk scores reliable.
- Saved a **joblib** bundle (pipeline + expected columns) and generated the main **evaluation metrics**.

3. Amarasinghe M.S.R.V -> IT23216778

Role overview: Run experiments, ensure quality, and solve modeling issues.
Key responsibilities:

- Compared **SMOTE vs. class weights**; ran **cross-validation**; tuned parameters to improve **recall on churners**.
- Checked and fixed **data leakage**; removed redundant features (e.g., **Avg_Open_To_Buy**).
- Designed **threshold selection** using a simple **ROI idea** (value per retained customer, contact cost, treatment effectiveness).
- Prepared the **confusion matrix**, **PR-AUC**, and a **metrics file** for the app.

4. Gunawardana A.A. -> IT23140752

Role overview: Build the Streamlit app and outline deployment and next steps.
Key responsibilities:

- Built the **Streamlit** app with **EDA**, **Batch Scoring** (with a **sample CSV** button), and **Single-Customer what-if** (slider to test changes).
- Added **reason codes** and the **ROI threshold helper** designed by the team.
- Handled **caching**, **requirements**, and packaging the **model bundle**.
- Drafted the **deployment plan**, **monitoring** ideas (drift, data quality, KPIs), and the **demo script** for the video.

---------- END OF THE DOCUMENT ----------