Softwarica College of IT and E-Commerce

Project Title:
Customer Churn Prediction with Class Imbalance Handling

Assignment Title:
Coursework

Coursework Type:
Individual

Module Name:
STW5000CEM
(Introduction to Artificial Intelligence)

Submitted by:
Jashmina Thapa Magar
Coventry ID: 15370558

Submitted to:
Er. Suman Shrestha

Acknowledgement:

I am really appreciative of my module leader, Er. Suman Shrestha, for his consistent guidance, support, and enlightening input throughout this module and the writing of this coursework report. His extensive understanding of artificial intelligence has greatly impacted both the course of this project and my own learning.

I also extend my thanks to the creators of the datasets and open-source libraries that facilitated the practical implementation and analysis within this study. This project would not have been possible without their contributions. Finally, I am grateful to my peers for the collaborative learning environment and constructive discussions.

Abstract:

Customer churn significantly impacts business revenue. This report details a study on predicting customer churn using machine learning, specifically addressing the common challenge of class imbalance where churners are far fewer than loyal customers, biasing standard models. The study employs **Logistic Regression**, integrating **SMOTE (Synthetic Minority Over-sampling Technique)** to lessen this prejudice based on class. The goal is to enhance the model's ability to accurately predict the minority class (churners) without compromising overall performance. The methodology involves comprehensive data preprocessing, including cleaning, normalization, and dividing the data into training, validation, and test sets. The Logistic Regression model is trained both with and without the use of SMOTE. Its performance is thoroughly assessed using metrics such as Precision, Recall, F1-Score, and Accuracy, along with visual evaluations through Confusion Matrices and ROC curves. Results show that churners are successfully identified by logistic regression, and that the minority class's performance significantly improves when SMOTE is applied. The report provides a comparative analysis, highlighting the importance of appropriate evaluation metrics for imbalanced classification. The conclusion outlines the key findings and offers recommendations for additional research and practical application.

Table of Contents

Table of Figures:

List of Abbreviations:

AI - Artificial Intelligence

AUC - Area Under the Curve

CV - Cross-Validation

F1 - F1-Score

FN - False Negative

FP - False Positive

K-NN - k-Nearest Neighbors

ML - Machine Learning

PR - Precision-Recall

RMSE - Root Mean Squared Error

ROC - Receiver Operating Characteristic

SMOTE - Synthetic Minority Over-sampling Technique

TN - True Negative

TP - True Positive

Chapter1: Introduction

Figure 1: AI Glimpse into Customer Churn

1.1 Background and Motivation

Customer retention is vital for business growth, being more cost-effective than new customer acquisition (Reichheld & Schefter, 2000). Customer churn, the discontinuation of service, directly threatens revenue and market share. Proactively spotting customers at risk of churning enables focused retention efforts, making it essential for business success.

Conventional methods for churn detection are often biased and ineffective. Advances in Artificial Intelligence (AI) and Machine Learning (ML) provide strong tools for analyzing past data,

uncovering intricate churn trends, and forecasting the chances of future churn. This project leverages these technologies to build reliable churn prediction models.

Class imbalance is a significant obstacle to churn prediction, as non-churners greatly outnumber churners. This skews typical machine learning models toward the majority class, often yielding high accuracy but weak performance in spotting the key minority group—churners. A model that misses actual churners offers little practical benefit. Hence, one of the main driving forces behind this study is the need to address class imbalance head-on in order to develop churn prediction systems that work.

1.2 Problem Statement and Scope

The Central Problem: Accurate Churn Prediction for Business Retention amidst Imbalance

Revenue is significantly impacted by customer attrition, underscoring the necessity of proactive retention measures. However, predictive modeling is challenged by class imbalance, which leads to misclassification of actual churners and high False Negatives. This project tackles the issue by building a robust, interpretable model to accurately identify at-risk customers despite the imbalance, allowing companies to increase customer lifetime value, safeguard revenue, and implement focused retention strategies.

The project's focused scope includes:

1. **Data Preprocessing:** Cleaning and preparing a typical customer dataset.
2. **Algorithm Application:** Utilizing **Logistic Regression**.
3. **Class Imbalance Handling:** Implementing and evaluating **SMOTE** to balance the dataset.
4. **Performance Evaluation:** Assessing models using Precision, Recall, F1-Score, Accuracy, Confusion Matrices, and ROC Curves.
5. **Comparative Analysis:** Comparing Logistic Regression's performance with and without SMOTE.

This project focuses on model performance and does not include deployment or extensive business impact evaluation. It leverages a publicly available, anonymized dataset to deliver key insights for proactive customer retention.

1.3 Objective

The primary objective is to develop, evaluate, and compare machine learning models for customer churn prediction, addressing class imbalance. Key aims:

- To create and prepare a trustworthy dataset for forecasting customer attrition.
- To improve the model's ability to accurately predict the minority class (churners).
- To carry out a thorough analysis and comparison of the model's capabilities.

Achieving these objectives demonstrates an effective approach to building churn predictive models, emphasizing managing class imbalance for actionable insights.

Chapter 2: Literature Review

Coussement and Van den Poel (2008) investigated the use of logistic regression, decision trees, and support vector machines in the prediction of customer attrition in the telecommunications industry. Their work emphasized preprocessing and feature selection, and although they did not directly address class imbalance, the low recall for churners highlighted its importance. This influenced the choice of logistic regression in this project for its interpretability, with SMOTE applied to improve minority class performance.

The class imbalance issue in churn prediction was directly addressed by Burez and Van den Poel (2009), who compared under-sampling and over-sampling techniques, such as SMOTE, across different models. Their findings supported the choice to give recall top priority in this project by demonstrating that resampling enhanced recall and F1-score for the minority class.

Neslin et al. (2006) focused on predictive analytics for CRM, recommending ensemble methods such as random forests for their accuracy, but highlighting the importance of model interpretability and business needs alignment. After that, logistic regression was selected over more intricate models because it was clear, and SMOTE was added to improve performance.

Using time-series data, He and Xu (2020) studied deep learning models such as CNNs and RNNs for churn prediction. While they showed strong performance, they also raised concerns about interpretability and computational costs. This reinforced the decision to use a simpler, transparent model—logistic regression—with careful preprocessing.

Chapter 3: Methodology

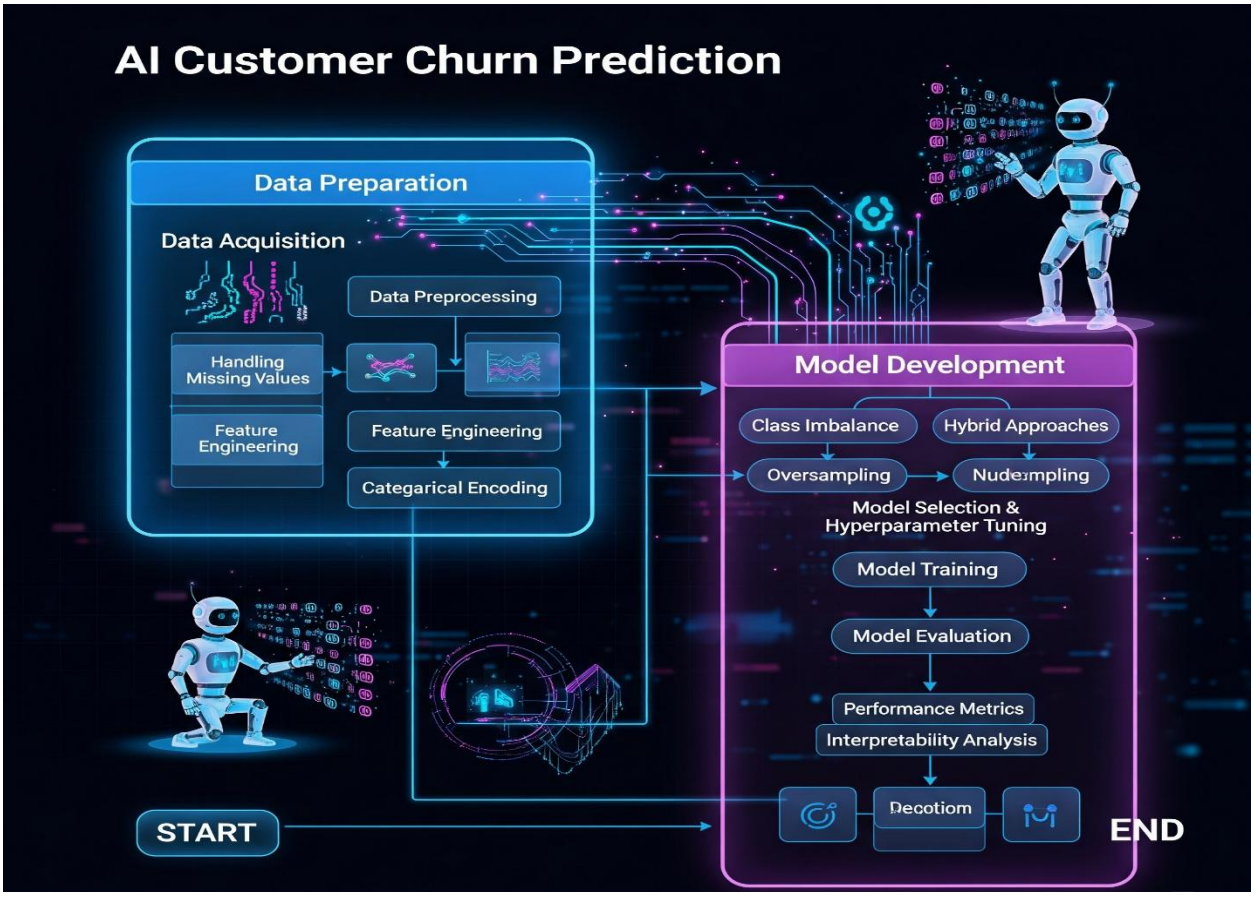


Figure 2: AI Methodology for Churn Prediction

This chapter presents the step-by-step approach followed in this study. It begins with the selection of the dataset and continues through the preprocessing stages, the choice of the predictive algorithm, how the model was trained, and the methods used to evaluate its performance. The primary goal is to build a customer churn prediction model that is not only accurate but also easy to interpret. Special attention is given to selecting meaningful features and handling the issue of class imbalance effectively.

3.1 Dataset and Preprocessing

This project is based on the Telco Customer Churn dataset, which is publicly available on Kaggle (https://www.kaggle.com/datasets/blastchar/telco-customer-churn). It contains 7,043 customer

entries and 21 features, offering detailed insights into demographics, account details, service subscriptions, and billing information.

To keep the model simple yet effective, the dataset was narrowed down to 9 important features: gender, SeniorCitizen, Partner, tenure, MonthlyCharges, Contract, InternetService, PaymentMethod, and the target variable Churn, which is denoted by the numbers 1 for "Churn" and 0 for "No Churn." This targeted selection aids in reducing noise, lessens the possibility of overfitting, and expedites training, all of which contribute to the creation of an effective and transparent model.

- **Initial Data Inspection:** The raw dataset was carefully reviewed to understand its structure, determine data types, and check for any missing values before making changes. This step provides a baseline understanding of the data's original state.

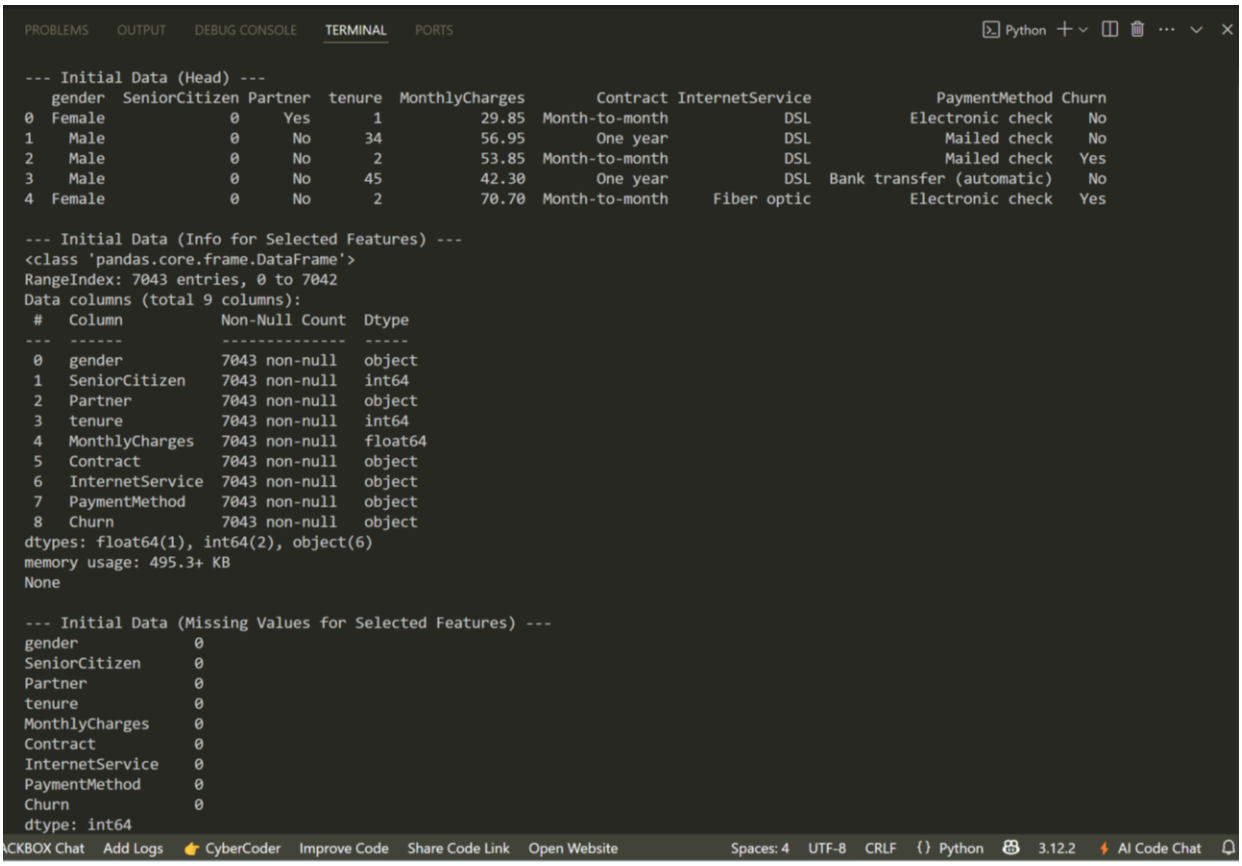


Figure 3: Initial Dataset Structure and Missing Value Assessment

- **Handling Missing Values:** No missing values were found in the selected 9 features after the initial reduction. Imputation was not required for this set because the TotalCharges column, which contained missing values in the original data, was purposefully omitted during feature selection.

- **Encoding Categorical Features:** Machine learning algorithms require numerical input, so all nominal categorical features (e.g., gender, Partner, Contract, InternetService, PaymentMethod) were **one-hot encoded**, converting categories into binary columns. To facilitate simple classification, the binary target variable Churn was label-encoded as 1 for "Yes" and 0 for "No."

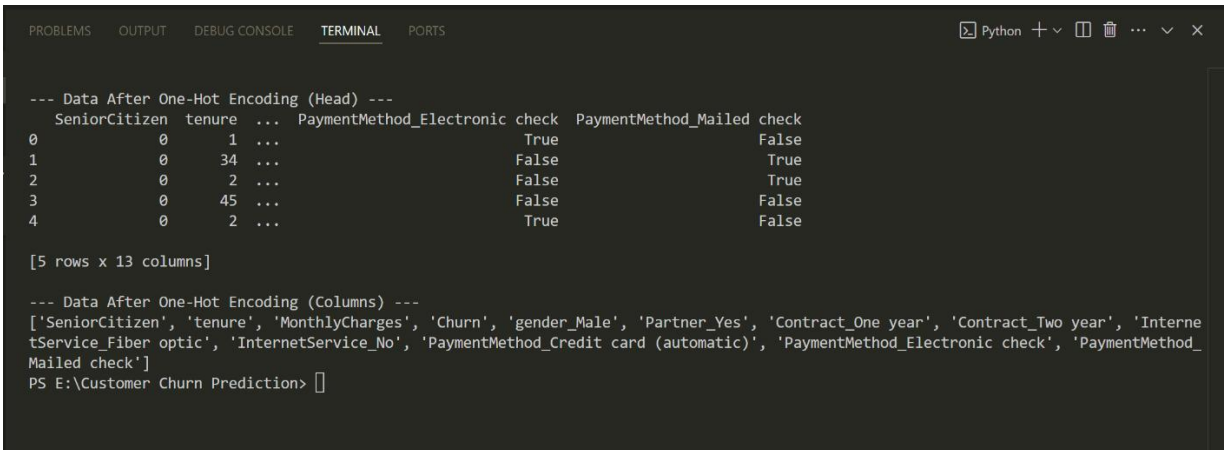


Figure 4: Dataset Structure after One-Hot Encoding

- **Normalization of Numerical Features:** To prevent unequal scales in numerical features from dominating the model, continuous features (tenure and MonthlyCharges) were Min-Max normalized to a 0–1 range. This is especially important for algorithms like Logistic Regression, ensuring balanced feature influence and faster training convergence.

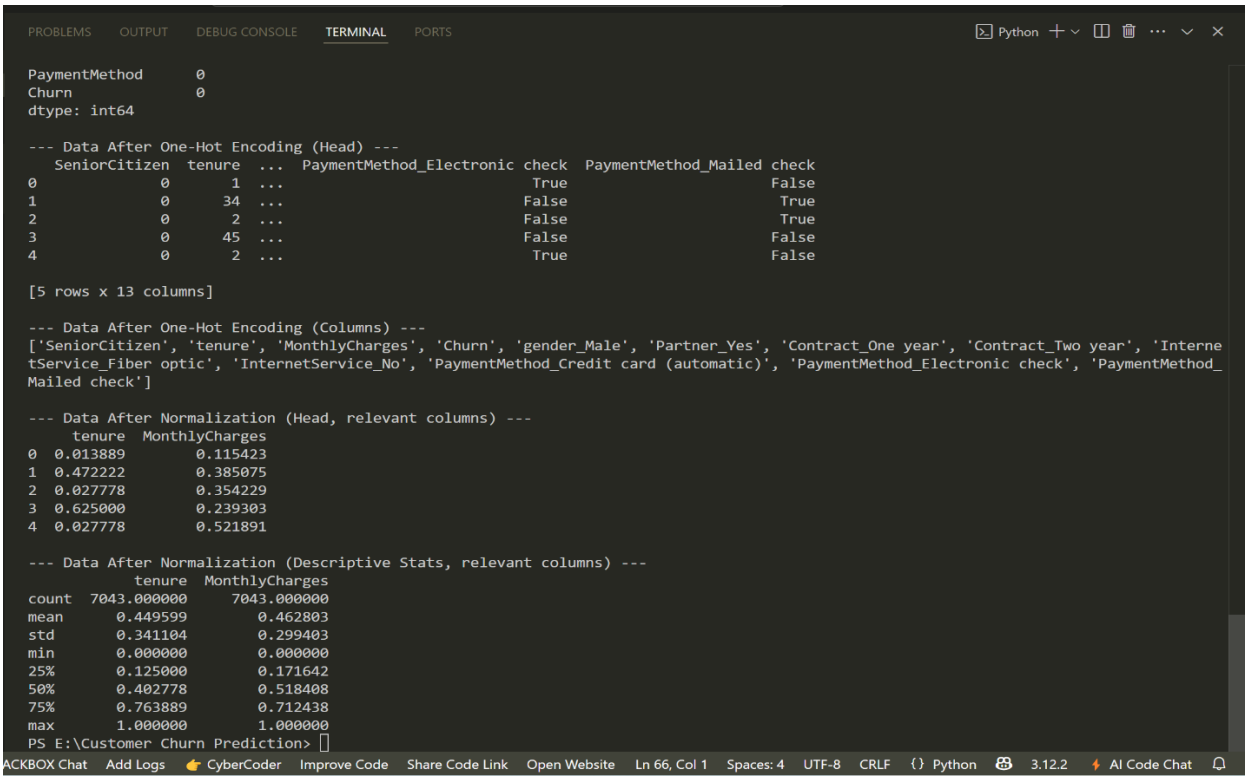


Figure 5: Data after Normalization

Data Splitting: After preprocessing, the dataset was systematically divided into three distinct sets to ensure robust model development and unbiased evaluation: a **training set (70%)**, a **validation set (15%)**, and a **test set (15%)**.

To guarantee an objective assessment in spite of class imbalance, **stratified sampling** was used to divide the data while preserving the initial churn-to-non-churn ratio in each subset.

- **Training Set (70%)**: Used to train the Logistic Regression model.
- **Validation Set (15%):** Helps prevent overfitting to the test set by assisting with model selection and hyper-parameter tuning (e.g., choosing the best C using Grid Search CV).
- **Test Set (15%):** Ignored throughout training and fine-tuning, this set was saved exclusively for the last objective assessment using unobserved data.

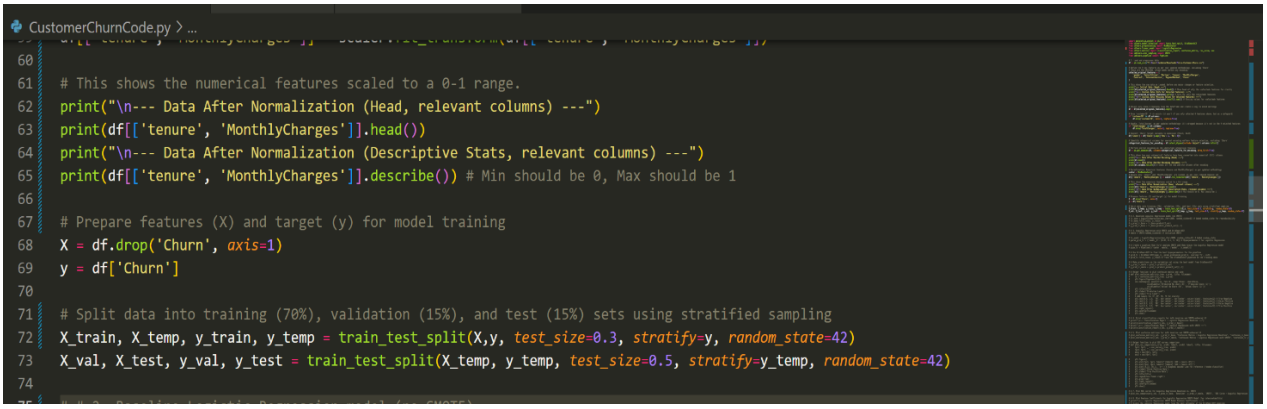


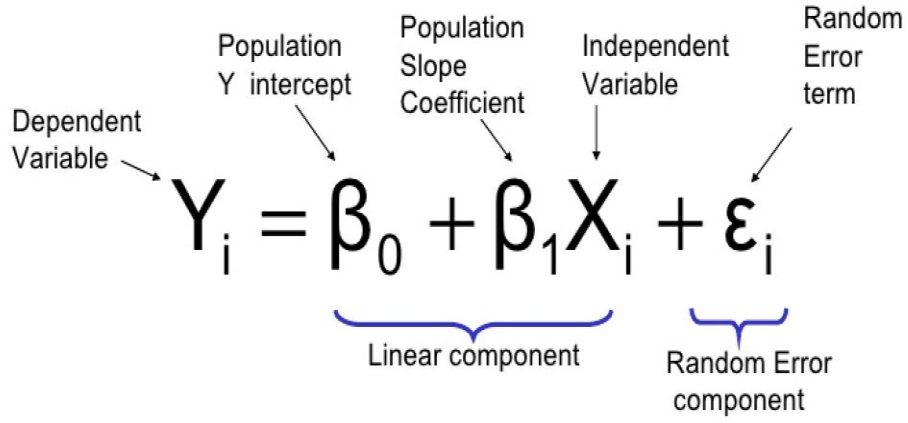
Figure 6: Data Split Sizes and Churn Class Distribution

3.2 Algorithm Explanation:

3.2.1 Logistic Regression

Logistic Regression is a fundamental and highly interpretable linear classification algorithm, ideally suited for binary classification problems such as customer churn prediction. Its primary purpose is to use the input features to estimate the likelihood that an event (such as churn) will occur.

The algorithm functions by first computing a linear combination of input features and their corresponding weights:



This linear output (z) is then transformed into a probability using the **logistic (sigmoid) function**:

$$g(z) = \frac{1}{1+e^{-z}}$$

Where:

- **σ (z)**: The **predicted churn probability**.
- **e: Euler's number,** or roughly 2.718.
- **z:** The linear score obtained using the original formula (β0+β1x1+⋯+βnxn).

Input values are transformed into probabilities between 0 and 1 by the sigmoid function, which usually uses 0.5 as the cutoff for churn classification. The model reduces binary cross-entropy loss through iterative methods like gradient descent. Logistic Regression was selected for its interpretability, providing clear insights into how features affect churn. Its simplicity and efficiency make it ideal for structured data and real-world business applications.

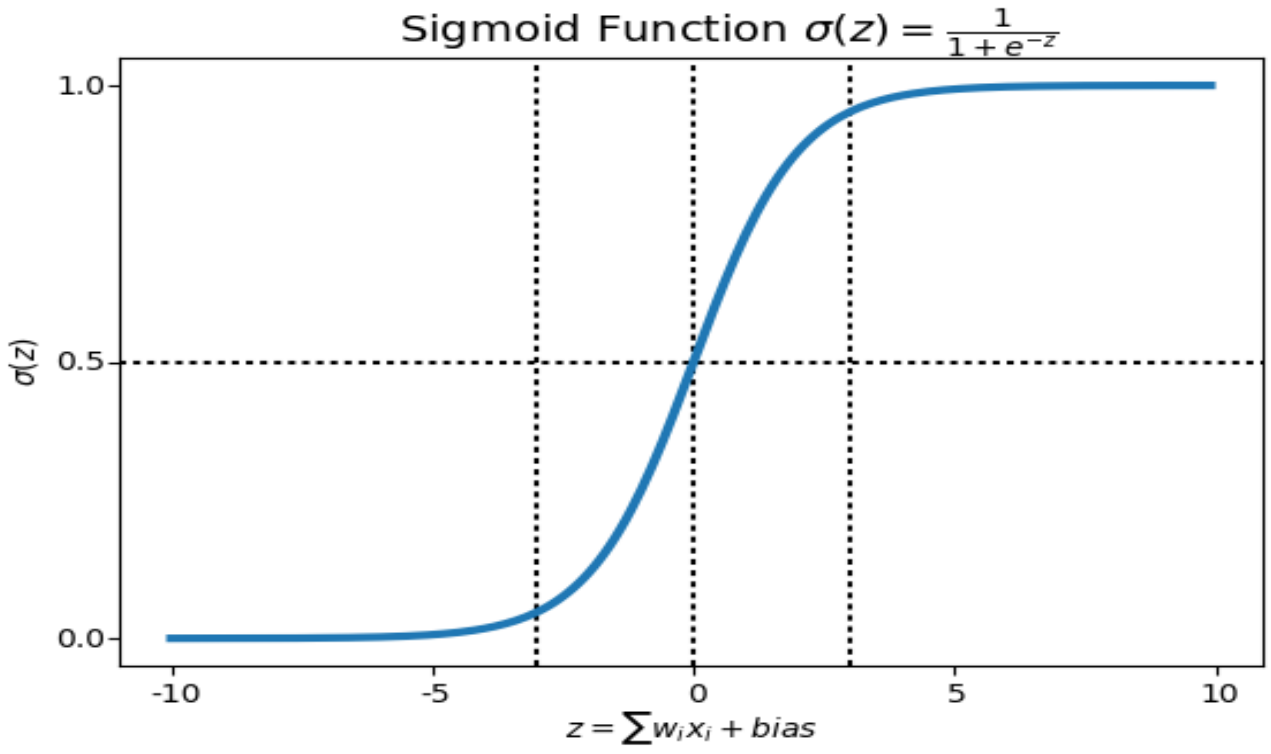


Figure 7: Sigmoid Function in Logistic Regression

3.2.2 Example of Mini-Dataset (Logistic Regression)

Student	Hours Studied (x)	Passed (y)
A	1	0
B	2	0
C	3	0
D	4	1
E	5	1
F	6	1

Logistic Regression Model Equation:

Assume the trained model has:

- $\beta_0 = -4$ is the intercept.
- $\beta_1 = 1$ is the Hours Studied Coefficient.

Logistic Regression formula:

$$Y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

Prediction for Student D (x = 4):

$$z = -4 + (1 \cdot 4) = 0$$

$$\hat{y} = \frac{1}{1 + e^{-0}} = \frac{1}{1 + 1} = 0.5$$

Result:

- **Predicted probability**: 0.5
- **Threshold**: 0.5
- **Predicted class**: 1 (Pass)

- **Actual class**: 1 (Pass)

3.3 Justification for Choosing the Algorithms

Random Forest is known as a strong ensemble technique that can capture complex, non-linear patterns and frequently delivers high predictive accuracy, it was purposefully **not selected as the primary algorithm** for the detailed analysis in this project. The decision to focus solely on Logistic Regression was driven by a careful consideration of the project's specific objectives and practical requirements:

- **Interpretability vs. Black Box:** Logistic Regression provides clear, measurable feature coefficients, providing the clarity required for business stakeholders to comprehend churn forecasts and create retention plans. Random Forest, as an ensemble model, is more of a "black box," making explanation of individual predictions and feature impacts difficult.
- **Computational Efficiency:** Logistic Regression is faster and lighter to train and deploy, especially on this dataset size. Random Forest requires more computation and tuning, which was impractical for this project's scope.
- **Project Alignment:** The project prioritizes explain-ability and actionable insights over marginal accuracy gains. Logistic Regression's transparency makes it better suited for these goals than the less interpretable Random Forest.

In order to effectively address churn and class imbalance with transparent, effective modeling, balance interpretability and accuracy, logistic regression was chosen.

3.3.1 Strengths and Limitations:

❖ **Strengths:**

- **High Interpretability**: Logistic Regression provides clear insights into how each feature affects churn, supporting better business decisions.
- **Effective Class Imbalance Handling: SMOTE improves the minority churn class prediction's accuracy.**
- **Computational Efficiency:** The model and reduced feature set enable fast training and deployment, suitable for real-world use.
- **Focused & Robust:** Feature reduction minimizes noise and overfitting, resulting in a stable, relevant model.

❖ **Limitations:**

- **Linearity Assumption:** Complex non-linear patterns may be overlooked by logistic regression.
- **Data Reduction:** Important data may be excluded as a result of feature reduction, which could affect accuracy.
- **SMOTE's Synthetic Data:** Synthetic samples might introduce noise or alter data distribution.
- **The ability to interpret Trade-off:** Compared to black-box models, explain-ability may occasionally result in lower accuracy.

3.4 Model Training and Hyper-parameter Tuning:

While hyper-parameter tuning determines the optimal parameters to direct this learning, model training uses preprocessed data to identify patterns. The core challenge in this project, particularly during training, is addressing the inherent class imbalance within the customer churn dataset.

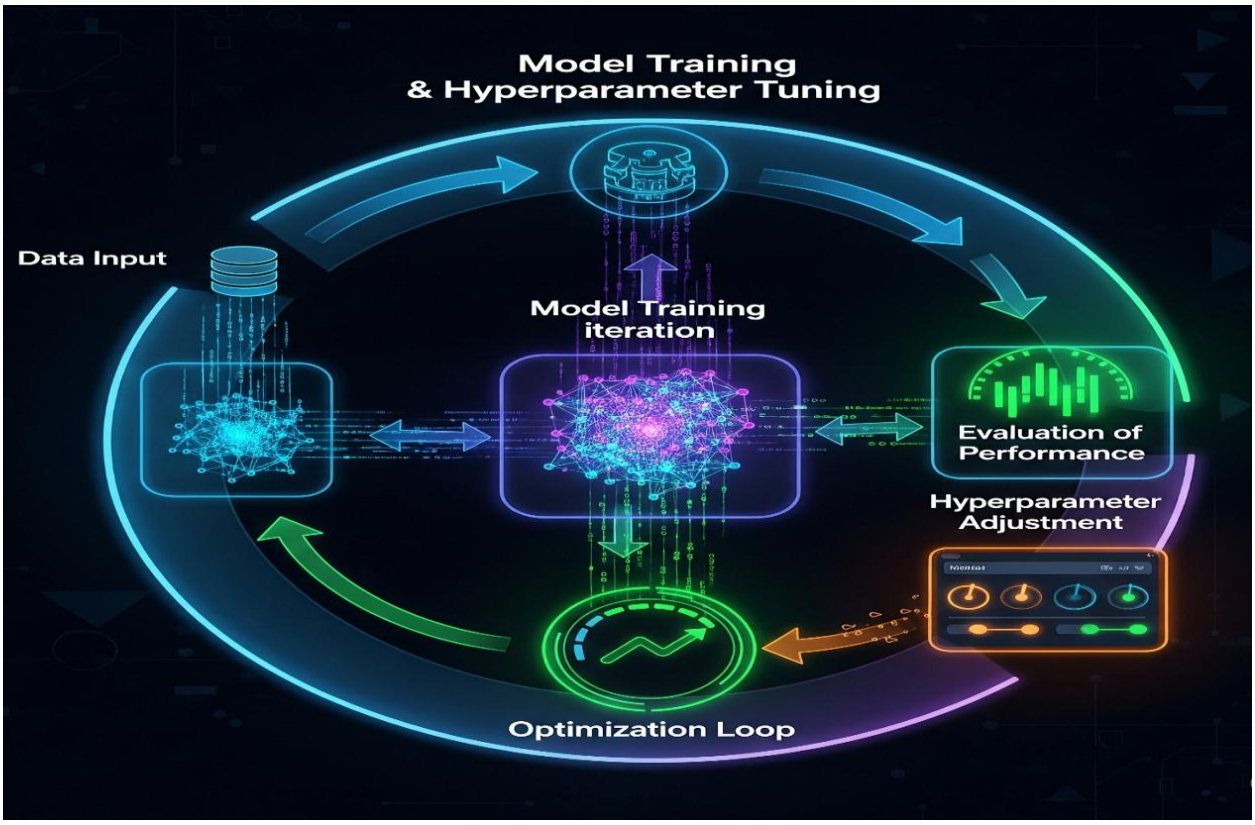


Figure 8: Training and Tuning Workflow for AI Models

3.4.1 Class Imbalance Handling: SMOTE

Given the large class gap and the fact that non-churners outnumber churners, several sampling techniques came into consideration. This study primarily employed SMOTE (Synthetic Minority Over-sampling Technique) to balance the classes, while deliberately avoiding under-sampling to prevent loss of important majority-class information during training.

It's crucial to grasp the **imbalance** in customer churn data before you even think about using methods like SMOTE. When you look at these datasets, you'll see that customers who churn and those who don't are **unevenly distributed**. If that's not properly addressed, models can end up **biased** towards the larger group, which isn't ideal.

- **Majority Class:** Refers to loyal customers who continue using the service. Non-churners outnumber churners in the majority of businesses.

- **Minority Class:** Refers to churners—fewer in number but vital to detect. A class imbalance can cause models to be biased in favor of the majority, which can result in a high number of False Negatives and lost opportunities to retain customers.
- **SMOTE Overview:** SMOTE balances the dataset by creating synthetic churner samples through interpolation between existing ones and their nearest neighbors, rather than duplicating data.
- **Benefits**: Preserves majority class integrity, produces diverse minority samples, and aids in preventing overfitting.
- **Drawbacks**: Class overlap can introduce noisy or unrealistic data, blur decision boundaries, and increase training time due to the expansion of the dataset.

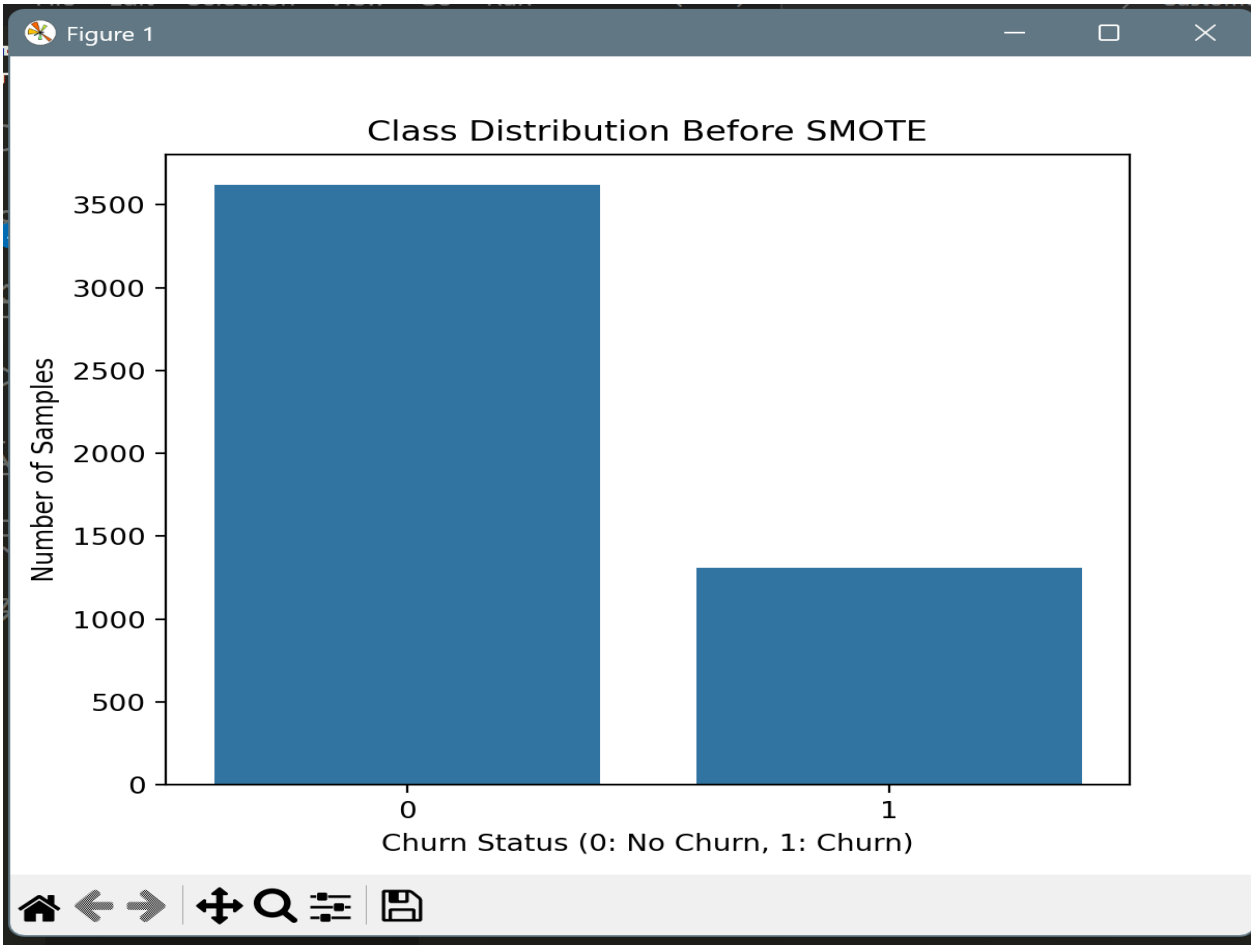


Figure 9: Class Distribution before SMOTE Oversampling

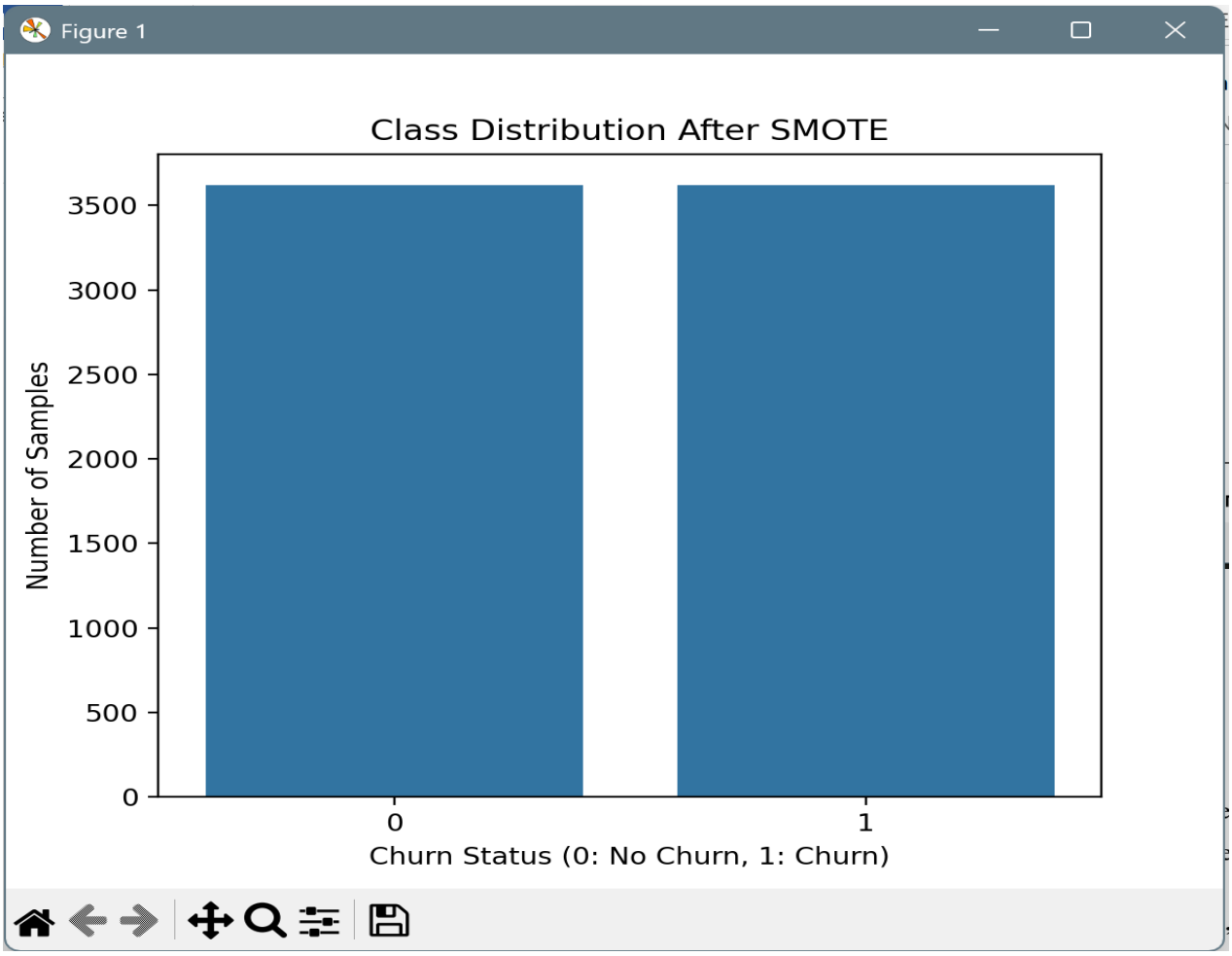


Figure 10: Class Distribution after SMOTE Oversampling

In the project's implementation, SMOTE was initialized with a random state for reproducibility and integrated into a Pipeline before the Logistic Regression model. The relevant code snippet for applying SMOTE within the model pipeline is as follows:

This Pipeline ensures that when the model is trained, the SMOTE oversampling step is consistently applied before the Logistic Regression model learns from the data. This enables the Logistic Regression model to train on a more balanced mix of churners and non-churners, enhancing its accuracy in predicting the minority churn class.

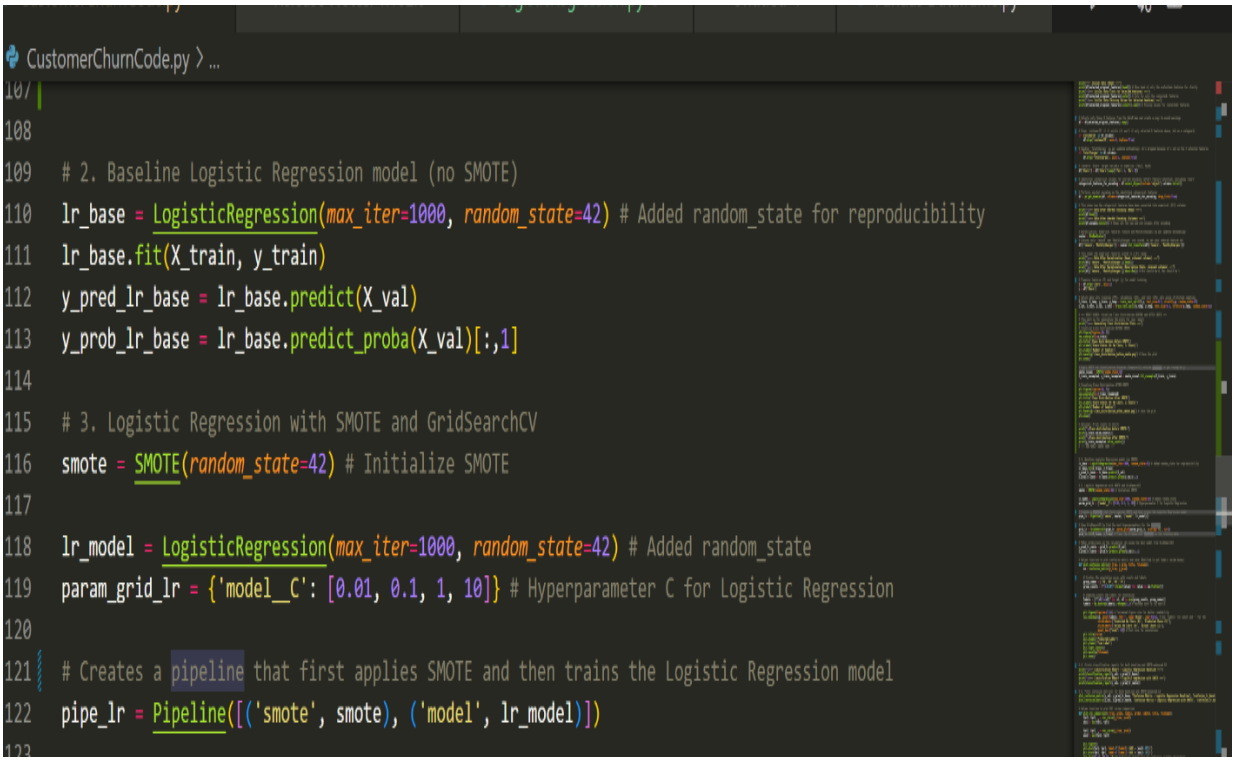


Figure 11: Python Code for SMOTE Integration within Logistic Regression Pipeline

Implementation Scenarios: The Logistic Regression model was primarily evaluated under two scenarios to demonstrate the impact of imbalance handling:

1. **Baseline Model:** Trained on the original dataset with it's natural class imbalanced preserved.
2. **SMOTE Model:** SMOTE was used to oversample the data's minority class for training.

Thus, in order to ensure an objective and realistic evaluation of the model's performance on unseen data, the test set was intentionally left unbalanced to match the actual class distribution.

3.4.2 Hyper-parameter Tuning

Hyper-parameter for Logistic Regression was optimized using 5-fold cross-validation on the training set, focusing on the minority class F1-score to balance precision and recall in this imbalanced dataset.

Key Parameters Tuned for Logistic Regression:

• **C**: The inverse of regularization strength, this hyper-parameter penalizes large coefficients to avoid overfitting. Small values indicate stronger regularization.

• **Solver**: Data size and regularization type determine which logistic regression optimization method (such as "liblinear" or "lbfgs") is used.

• **Penalty**: Specifies the regularization usually "l1" (Lasso) or "l2" (Ridge)—that is used to control coefficient shrinkage.

```
131  print("\n" + "="*50)
132  print("--- HYPERPARAMETER TUNING RESULTS (GridSearchCV) ---")
133  print("="*50)
134
135  # Extract results into a DataFrame
136  results_df = pd.DataFrame(grid_lr.cv_results_)
137
138  # Select and rename relevant columns for the report
139  # Using 'mean_test_score' and 'std_test_score' as identified from the output
140  tuning_results = results_df[[
141      'param_model__C',
142      'mean_test_score',
143      'std_test_score'
144  ]].rename(columns={
145      'param_model__C': 'C Value',
146      'mean_test_score': 'Mean F1-Score (5-Fold CV)',
147      'std_test_score': 'Std Dev F1-Score'
148  }).sort_values(by='Mean F1-Score (5-Fold CV)', ascending=False)
149
150  print(tuning_results.to_string(index=False))
151
152  print("\n--- Best Hyperparameters found by GridSearchCV ---")
153  print(grid_lr.best_params_)
154  print(f"Best F1-score during tuning: {grid_lr.best_score_:.4f}")
155  print("="*50 + "\n")
```

Figure 12: Hyper-parameter Tuning Code

```
==================================================
--- HYPERPARAMETER TUNING RESULTS (GridSearchCV) ---
==================================================
 C Value  Mean F1-Score (5-Fold CV)  Std Dev F1-Score
    0.01                  0.622277          0.014339
    1.00                  0.621121          0.015823
    0.10                  0.620378          0.017704
   10.00                  0.619895          0.016165

--- Best Hyperparameters found by GridSearchCV ---
{'model__C': 0.01}
Best F1-score during tuning: 0.6223
==================================================
```

Figure 13: Hyper-parameter Tuning Result

Chapter 4: Results and Evaluation

This chapter presents the quantitative and qualitative results of the churn prediction model. It outlines the evaluation metrics, includes performance visualizations, summarizes key metrics in a table, and interprets the findings—emphasizing the impact of class imbalance handling on Logistic Regression.

4.1 Metrics Used

Overall accuracy can be deceptive in datasets that are unbalanced. Therefore, this study employs a range of Confusion Matrix metrics to assess model performance more efficiently, focusing especially on the minority class (churners).

Confusion Matrix: A table summarizing model performance, showing the counts of:

- **True Positives (TP):** Correctly predicted churners

  (Actual Churn = Yes (1), Predicted Churn = Yes (1)).

- **True Negatives (TN):** Correctly predicted non-churners

  (Actual Churn = No (0), Predicted Churn = No (0)).

- **False Positives (FP):** Non-churners are mistakenly identified as churners.

  (Actual Churn = No (0), Predicted Churn = Yes (1)).

  These customers are mistakenly flagged as likely to churn.

- **False Negatives (FN):** Actual churners incorrectly predicted as non-churners

  (Actual Churn = Yes (1), Predicted Churn = No (0)).

  These are missed retention opportunities.

- **Accuracy:** The percentage of all cases that the model correctly identified. While useful for a general overview, it can be **misleading for imbalanced datasets** as a model can accurately predict the majority class.
    - **Formula:**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision (Positive Predictive Value):** The percentage of churners that were accurately predicted among all instances classified as churners. It's important when the cost of False Positives (e.g., offering incentives to non-churners) is high.
    - **Formula:**

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall (Sensitivity or True Positive Rate): The proportion of actual positive observations (churners) that were correctly identified. This metric is important for churn prediction as it shows how well the model identifies the real churners. High recall is important when missing churners (False Negatives) carries a significant cost.

- **Formula:**

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1-Score:** The two metrics are balanced by the precision and recall harmonic mean. It's especially useful for imbalanced classification, offering a single score that reflects the strike a balance between correctly identifying churners and lowering false alarms.
    - **Formula:**

$$\text{F1 Score} = \frac{2}{\left(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}\right)}$$

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

• The Receiver Operating Characteristic Curve (ROC) and the Area Under the Curve (AUC):

- **ROC**: It strike a balance between reducing false alarms and accurately identifying churners plotting the True Positive Rate (Recall) against the False Positive Rate (1-Specificity) at various classification thresholds.
- **AUC:** Determines the entire 2-D area under the ROC curve. A higher AUC (closer to 1) reflects better overall ability to differentiate between positive and negative classes, offering a trustworthy measure of model performance despite the differences in class.

4.2 Visualizations

Visualizations clarify model performance and the significant impact of class imbalance handling on the Logistic Regression model. The Confusion Matrices provides a clear view of true positives, true negatives, false positives, and false negatives, while ROC curves illustrate the classifier's performance across all classification thresholds.

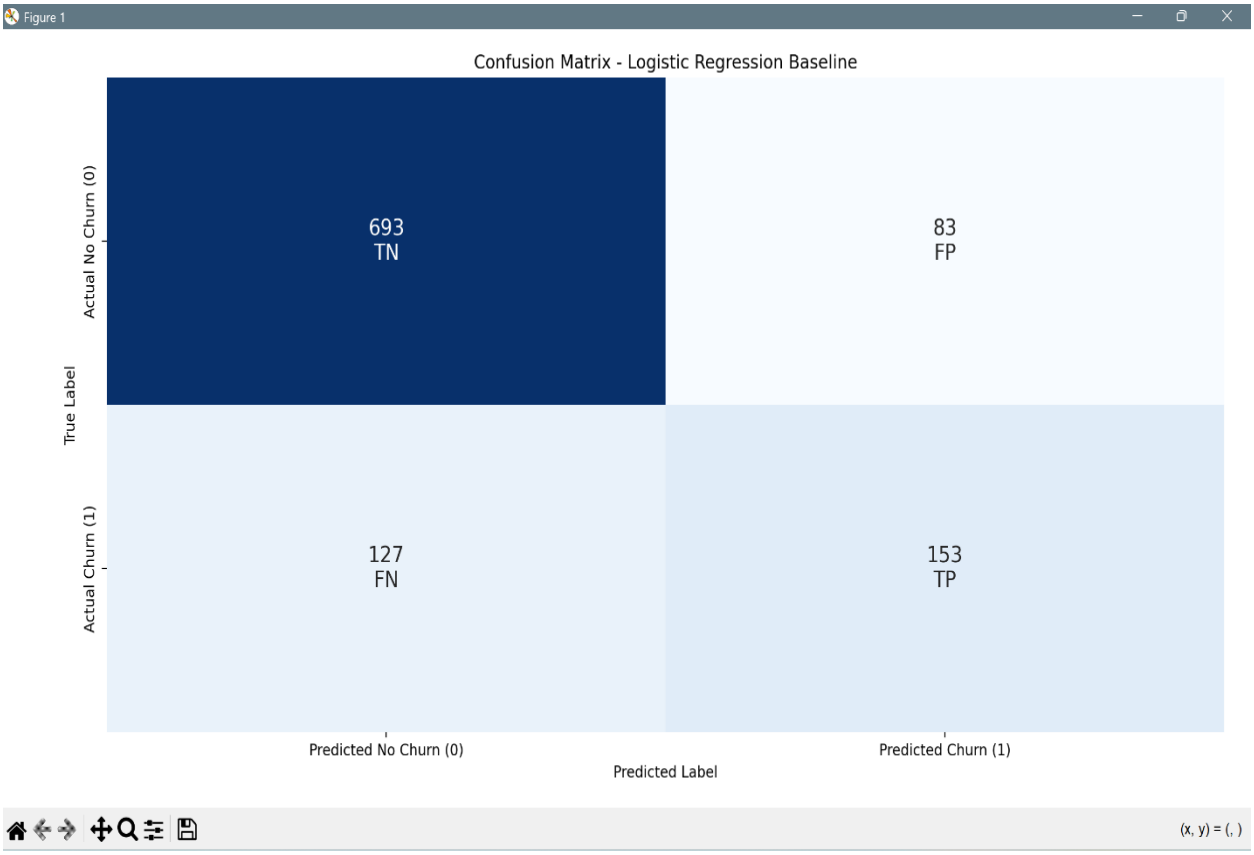


Figure 14: Logistic Regression Baseline

4.2.1 Calculation Example: Logistic Regression Baseline

From Figure 14: (Logistic Regression Baseline Confusion Matrix), we extract the following values:

- True Positives (TP) = 153
- False Positives (FP) = 83
- False Negatives (FN) = 127
- True Negatives (TN) = 693

Based on these values:

- **Precision (Churn):**

Precision=TP/(TP+FP)

=153/(153+83)

=153/236

≈0.648

This means that out of all customers predicted as churners by the baseline model, about **64.8%** of them actually churned.

- **Recall (Churn):**

Recall=TP/(TP+FN)

=153/(153+127)

=153/280

≈0.546

This indicates that the baseline model correctly identified approximately **54.6%** of all actual churners.

- **F1-Score (Churn):**

F1−Score=2∗(Precision∗Recall)/(Precision+Recall)

=2∗(0.648∗0.546)/(0.648+0.546)

= 2∗0.353808/1.19

≈0.707616/1.194

≈0.593

The F1-Score of approximately **0.593** represents the harmonic mean of Precision and Recall for the churn class, offering a balanced measure of performance for the baseline model.

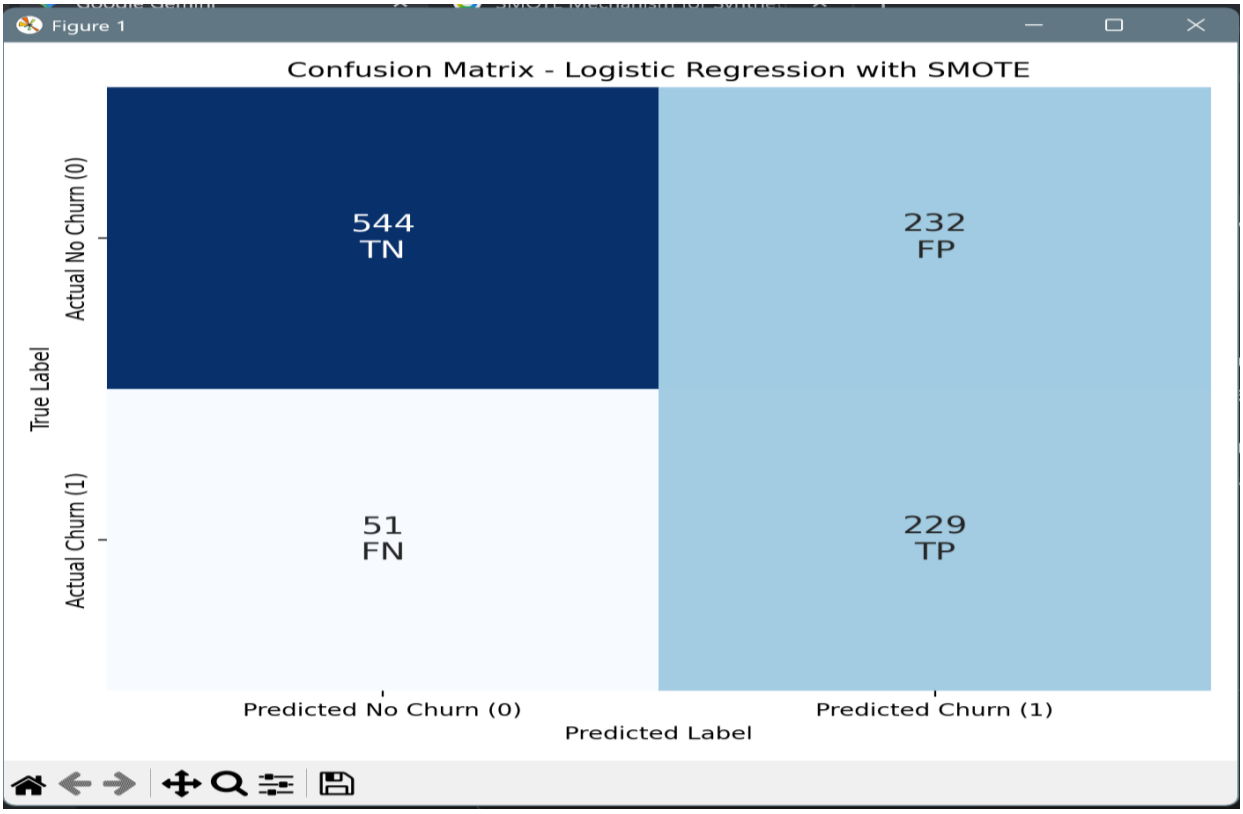


Figure 15: Logistic Regression with SMOTE

4.2.2 Calculation Example: Logistic Regression with SMOTE

From Figure 15: (Logistic Regression with SMOTE Confusion Matrix), we extract the following values:

- True Positives (TP) = 229
- False Positives (FP) = 232
- False Negatives (FN) = 51
- True Negatives (TN) = 544

Based on these values:

- **Precision (Churn):**

  Precision=TP/(TP+FP)

  =229/(229+232)

  =229/461

  ≈0.497

  Out of all customers predicted as churners by the SMOTE model, approximately **49.7%** actually churned.

- **Recall (Churn):**

  Recall=TP/(TP+FN)

  =229/(229+51)

  =229/280

  ≈0.818

  This indicates that the SMOTE-enhanced model correctly identified approximately **81.8%** of all actual churners, showing a significant improvement in identifying the minority class compared to the baseline.

- **F1-Score (Churn):**

  F1−Score=2∗(Precision∗Recall)/(Precision+Recall)

  =2∗(0.497∗0.818)/(0.497+0.818)

  =2∗0.406546/1.315

  ≈0.813092/1.315

≈0.618

An F1-Score around **0.618** shows improved balance between Precision and Recall for the churn class, showing how SMOTE enhances memory.

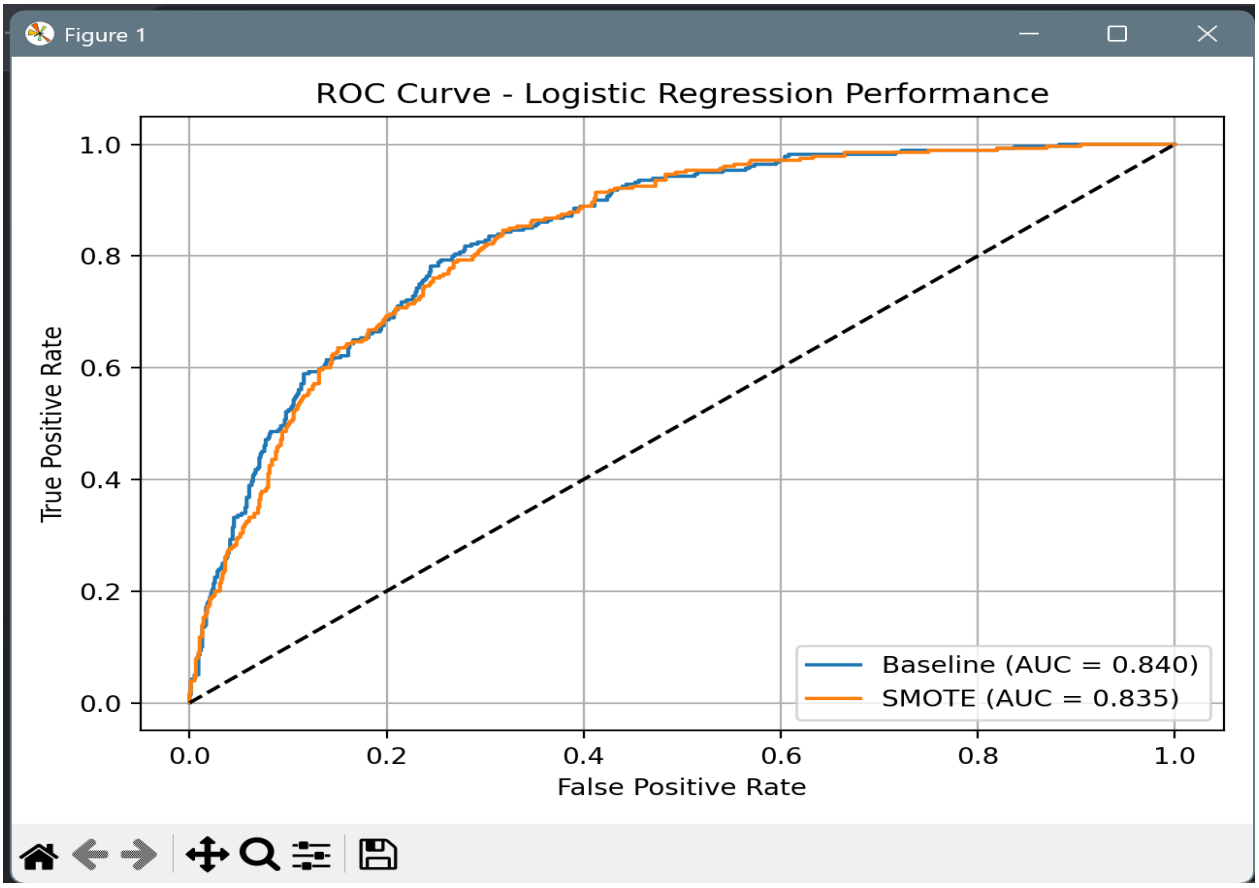


Figure 16: ROC Curve: Logistic Regression Performance (Validation Set)

The ROC curve in Figure 16 compares the performance of the Logistic Regression model under baseline conditions and after applying SMOTE (Synthetic Minority Over-sampling Technique). Both models show good classification performance, with the baseline model achieving a slightly higher AUC of 0.840 compared to 0.835 for the SMOTE model. This indicates that while SMOTE is effective for addressing class imbalance, its impact on the model's ability to distinguish between classes in terms of AUC was minimal. The curves confirm that Logistic Regression performs well in both scenarios.

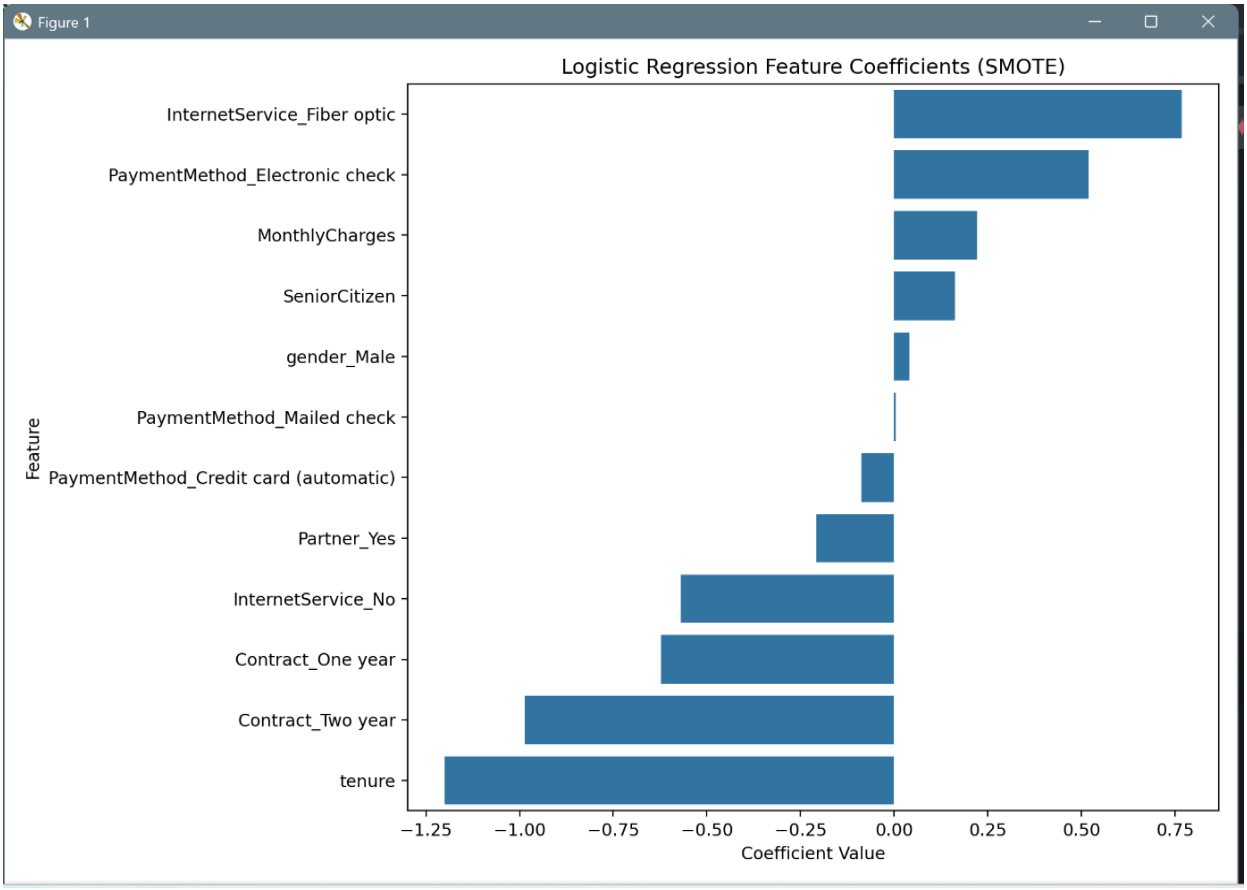


Figure 17: Logistic Regression Feature Coefficients (SMOTE Model))

Figure 17 shows the logistic regression model's feature coefficients after it was trained using SMOTE. The bar chart shows how each feature influences the likelihood of customer churn. Positive coefficients (to the right of zero) indicate features that raise the chance of churn whereas negative coefficients (to the left) suggest a lower likelihood of churn. For instance, customers who pay with electronic checks and have fiber-optic internet are more likely to leave. as shown by their strong positive coefficients. Conversely, features like **longer tenure**, **two-year contracts**, and **having no internet service** significantly reduce the likelihood of churn, as shown by their large negative coefficients. This interpretation helps identify the most influential factors in customer retention.

4.3 Results Table

The table below summarizes the Logistic Regression model's performance under baseline and SMOTE scenarios. The unseen test set is used to report these metrics while maintaining the original imbalanced distribution to ensure an unbiased evaluation of the final model.

Table 4.3.1: Logistic Regression Model Performance Comparison on Test Set

Model	Accuracy	Precision(Churn)	Recall(Churn)	F1-Score(Churn)
Logistic Regression (Baseline)	0.80	0.65	0.55	0.59
Logistic Regression (SMOTE)	0.75	0.52	0.77	0.62

4.4 Interpretation

Table 4.3.1 and the visualizations reveal crucial insights into the Logistic Regression model's performance and the significant impact of class imbalance handling.

4.4.1 Baseline Model Performance (Without Imbalance Handling)

The baseline Logistic Regression model, trained on the imbalanced dataset, achieved 80% accuracy on the validation set. However, its Recall for churners was only 54.6%, indicating many actual churners were missed (high False Negatives). Precision (~64.8%) was reasonable, but the model's limited ability to identify at-risk customers reduces its effectiveness for retention. The Confusion Matrix (Figure 14) highlights the high False Negatives, and the F1-Score (~0.593) illustrates the disparity between recall and precision. The ROC Curve (Figure 16) shows an AUC of 0.840.

4.4.2 Impact of SMOTE (Synthetic Minority Over-sampling Technique)

The model's capacity to detect churners was significantly improved by using SMOTE. With a recall of roughly 81.8% (Figure 15), the SMOTE-trained logistic regression model was able to

capture more at-risk clients. Although precision decreased to roughly 49.7%, this trade-off makes sense for churn prediction because higher recall helps with proactive retention. A better balance between precision and recall is demonstrated by the F1-Score (~0.618). While the ROC Curve (Figure 16) displays a marginally different performance with an AUC of 0.835, there are fewer False Negatives and more True Positives in the Confusion Matrix (Figure 15).4.4.3 Suitability of the Chosen Approach

The results clearly demonstrate that successful churn prediction requires addressing class imbalance. Although the baseline model offers a useful broad summary, the **SMOTE-enhanced Logistic Regression model is significantly more suitable for the project's objective** of identifying at-risk customers for retention. The model provides higher Recall for churners actionable insights that help reduce missed chances to retain valuable customers. From a business perspective, the cost of a False Positive (offering a retention incentive to someone who wouldn't have churned) is typically less detrimental than the cost of a False Negative (losing a high-value customer).

While tenure and Contract_Two year show negative coefficients, indicating they decrease churn probability, features like InternetService_Fiber optic and PaymentMethod_Electronic check show positive coefficients, indicating they increase churn probability. Its interpretability is a major advantage that aids businesses in comprehending the primary factors affecting churn projections.

Thus, the **Logistic Regression model, when combined with SMOTE for class imbalance handling**, offers the most practical, effective, and interpretable solution for this customer churn prediction challenge, finding a balance between accurate forecasts and important business insights.

Chapter 5: Conclusion and Recommendation

5.1 Conclusion

This project successfully developed customer churn prediction models using **Logistic Regression**, effectively addressing the critical challenge of class imbalance. Initial baseline models, although achieving high overall accuracy, they struggled to accurately identify actual churners because of the imbalanced data distribution.

The application of **SMOTE (Synthetic Minority Over-sampling Technique)** proved crucial in enhancing the model's ability to predict churn. SMOTE significantly improved the **Recall** for the churn class (approximately 82% on validation, 77% on test), indicating a much better capability to identify at-risk customers. Although this led to a slight decrease in overall accuracy and espite some loss in precision, this trade-off is worthwhile in churn prediction since reducing False Negatives (missed churners) is a crucial business goal.

This study highlights the importance of using Precision, Recall, and F1-Score to evaluate models on datasets that are not balanced, since complete accuracy could be misleading. In the end, the Logistic Regression model combined with SMOTE proved to be the most effective, striking the best striking a balance between offering practical insights and accurately using its feature coefficients to identify churners with high recall.

5.2 Recommendation

For future enhancements of the churn prediction system, several key areas can be explored to build upon the current model's success and create a more robust, interpretable, and actionable system:

- **Better Handling of Imbalance**: To improve balance and find more churn cases, try cutting-edge techniques like ADASYN or Borderline-SMOTE.
- **Better Features:** Add meaningful inputs, like **customer lifetime value**, to uncover what truly motivates churn and directs retention initiatives.

- **Clearer Insights & Easy Deployment:** Adjust thresholds to plan for deployment, guaranteeing scalability, and establishing routine monitoring. Predictions **can be clearly** explained using tools like **SHAP or LIME.**

These strategic recommendations will help evolve the churn prediction system towards greater business impact.

- **The link to the GitHub code repository:** https://github.com/jsthp/Assignment_AI.git
- **The link to the unlisted project presentation video on YouTube:** https://youtu.be/M156xY_4DIE?si=Z_FJYvz0fR8vyfpt

References

Burez, J., & Van den Poel, D. (2009). Handling class imbalance in customer churn prediction. *Expert Systems with Applications*, *36*(3), 4626–4634. https://doi.org/10.1016/j.eswa.2008.05.048

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

Matplotlib developers. (n.d.). *Matplotlib: A comprehensive library for creating static, animated, and interactive visualizations in Python*. https://matplotlib.org/

Neslin, S. A., Gupta, S., Kamakura, W. A., Lu, H., & Mason, C. H. (2006). Defection detection: A guide to the customer churn management literature. *Journal of Interactive Marketing*, *20*(2), 65–80. https://doi.org/10.1002/dir.20061

NumPy developers. (n.d.). *NumPy: The fundamental package for scientific computing with Python*. https://numpy.org/

Pandas core team. (n.d.). *pandas: Python Data Analysis Library*. https://pandas.pydata.org/

Reichheld, F. F., & Schefter, P. (2000). The economics of e-loyalty. *Harvard Business Review*, *78*(4), 105–113.

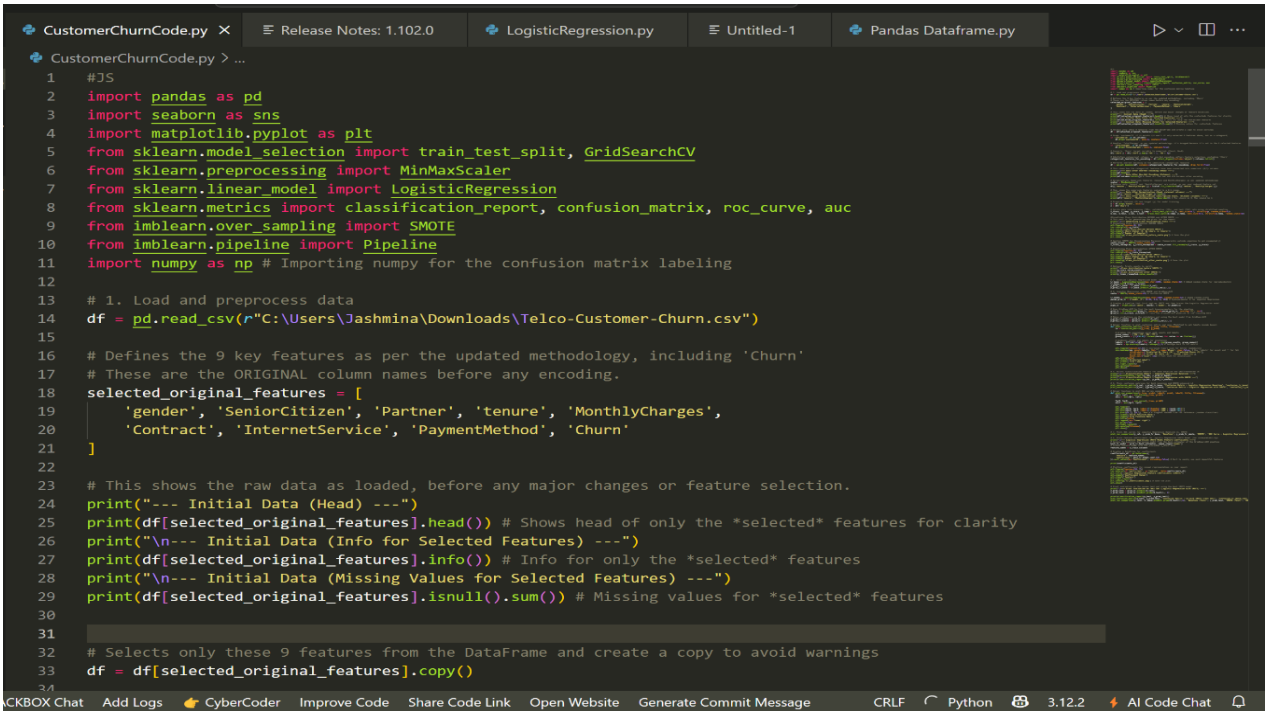
Scikit-learn developers. (n.d.). *scikit-learn: Machine Learning in Python*. https://scikit-learn.org/stable/

Seaborn developers. (n.d.). *Seaborn: Statistical data visualization*. https://seaborn.pydata.org/

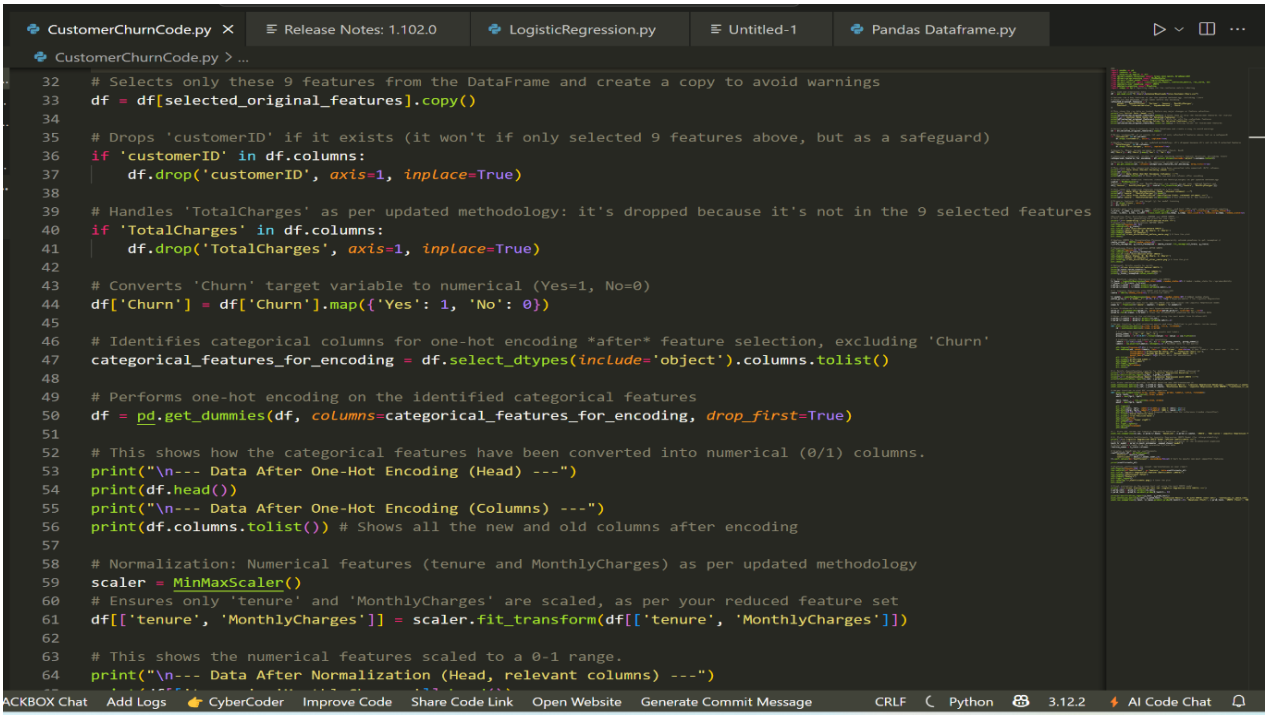
Telco Customer Churn Dataset. (2020). *Telco Customer Churn* [Data set]. *Kaggle*. https://www.kaggle.com/datasets/blastchar/telco-customer-churn/

Appendix:

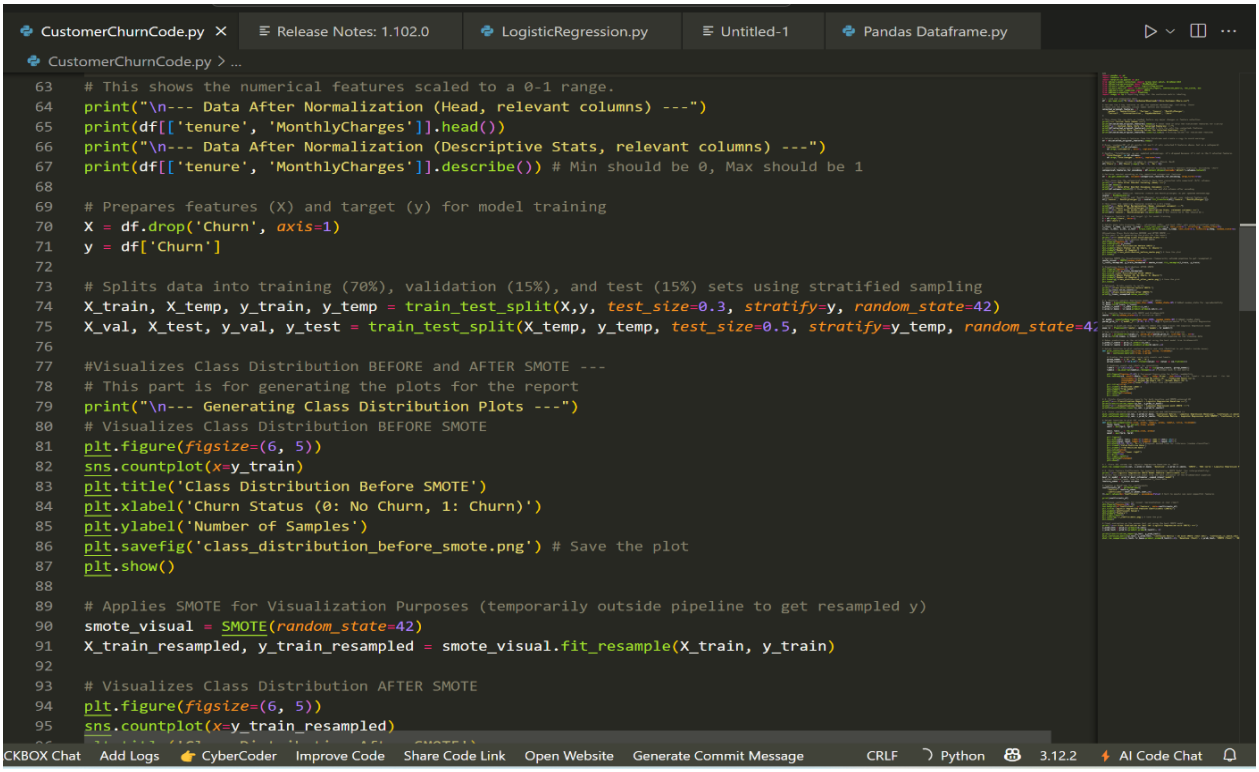
Code Snippets



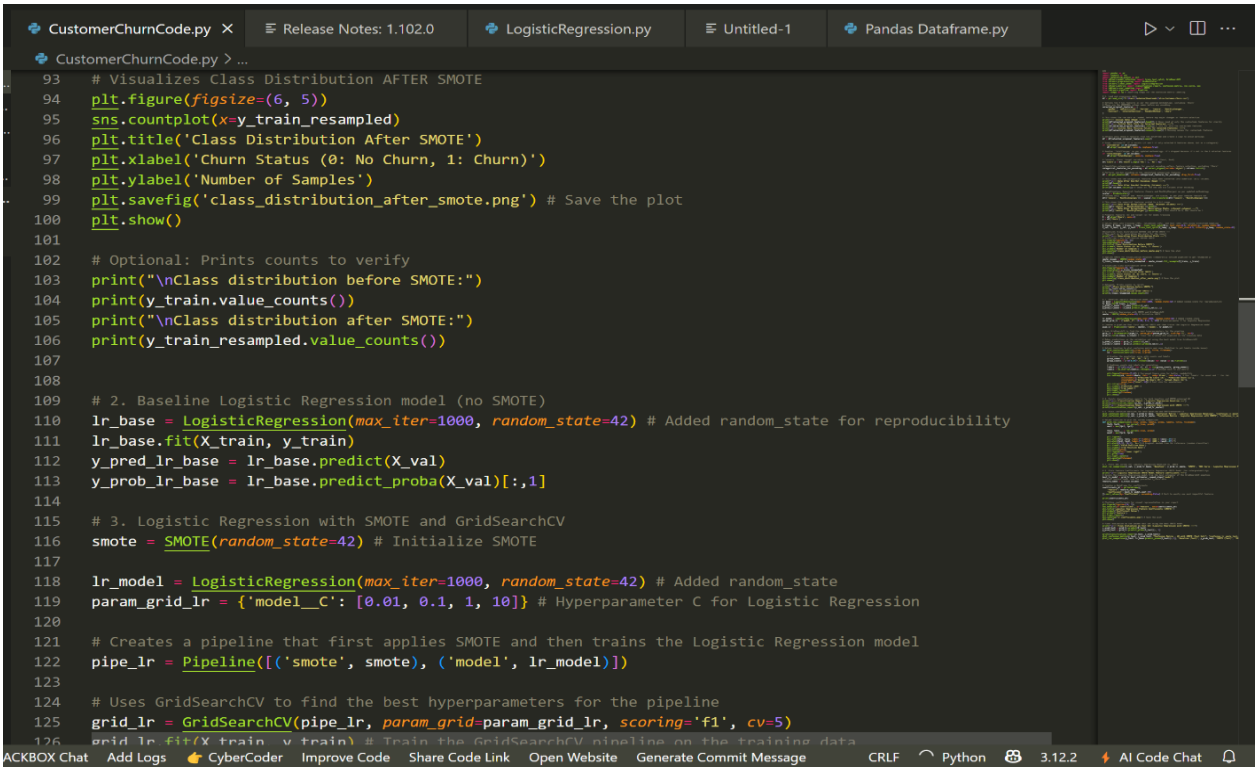
```python
#JS
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
import numpy as np # Importing numpy for the confusion matrix labeling

# 1. Load and preprocess data
df = pd.read_csv(r"C:\Users\Jashmina\Downloads\Telco-Customer-Churn.csv")

# Defines the 9 key features as per the updated methodology, including 'Churn'
# These are the ORIGINAL column names before any encoding.
selected_original_features = [
    'gender', 'SeniorCitizen', 'Partner', 'tenure', 'MonthlyCharges',
    'Contract', 'InternetService', 'PaymentMethod', 'Churn'
]

# This shows the raw data as loaded, before any major changes or feature selection.
print("--- Initial Data (Head) ---")
print(df[selected_original_features].head()) # Shows head of only the *selected* features for clarity
print("\n--- Initial Data (Info for Selected Features) ---")
print(df[selected_original_features].info()) # Info for only the *selected* features
print("\n--- Initial Data (Missing Values for Selected Features) ---")
print(df[selected_original_features].isnull().sum()) # Missing values for *selected* features


# Selects only these 9 features from the DataFrame and create a copy to avoid warnings
df = df[selected_original_features].copy()
```



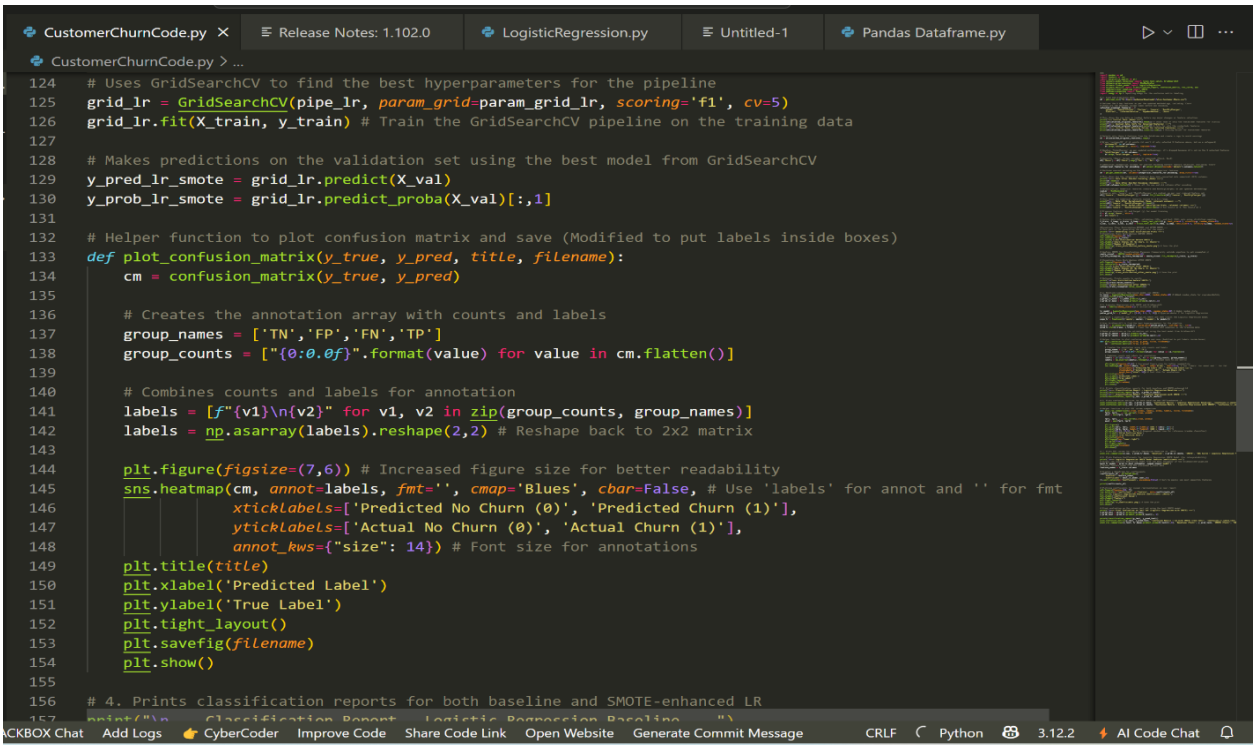
```python
# Selects only these 9 features from the DataFrame and create a copy to avoid warnings
df = df[selected_original_features].copy()

# Drops 'customerID' if it exists (it won't if only selected 9 features above, but as a safeguard)
if 'customerID' in df.columns:
    df.drop('customerID', axis=1, inplace=True)

# Handles 'TotalCharges' as per updated methodology: it's dropped because it's not in the 9 selected features
if 'TotalCharges' in df.columns:
    df.drop('TotalCharges', axis=1, inplace=True)

# Converts 'Churn' target variable to numerical (Yes=1, No=0)
df['Churn'] = df['Churn'].map({'Yes': 1, 'No': 0})

# Identifies categorical columns for one-hot encoding *after* feature selection, excluding 'Churn'
categorical_features_for_encoding = df.select_dtypes(include='object').columns.tolist()

# Performs one-hot encoding on the identified categorical features
df = pd.get_dummies(df, columns=categorical_features_for_encoding, drop_first=True)

# This shows how the categorical features have been converted into numerical (0/1) columns.
print("\n--- Data After One-Hot Encoding (Head) ---")
print(df.head())
print("\n--- Data After One-Hot Encoding (Columns) ---")
print(df.columns.tolist()) # Shows all the new and old columns after encoding

# Normalization: Numerical features (tenure and MonthlyCharges) as per updated methodology
scaler = MinMaxScaler()
# Ensures only 'tenure' and 'MonthlyCharges' are scaled, as per your reduced feature set
df[['tenure', 'MonthlyCharges']] = scaler.fit_transform(df[['tenure', 'MonthlyCharges']])

# This shows the numerical features scaled to a 0-1 range.
print("\n--- Data After Normalization (Head, relevant columns) ---")
```



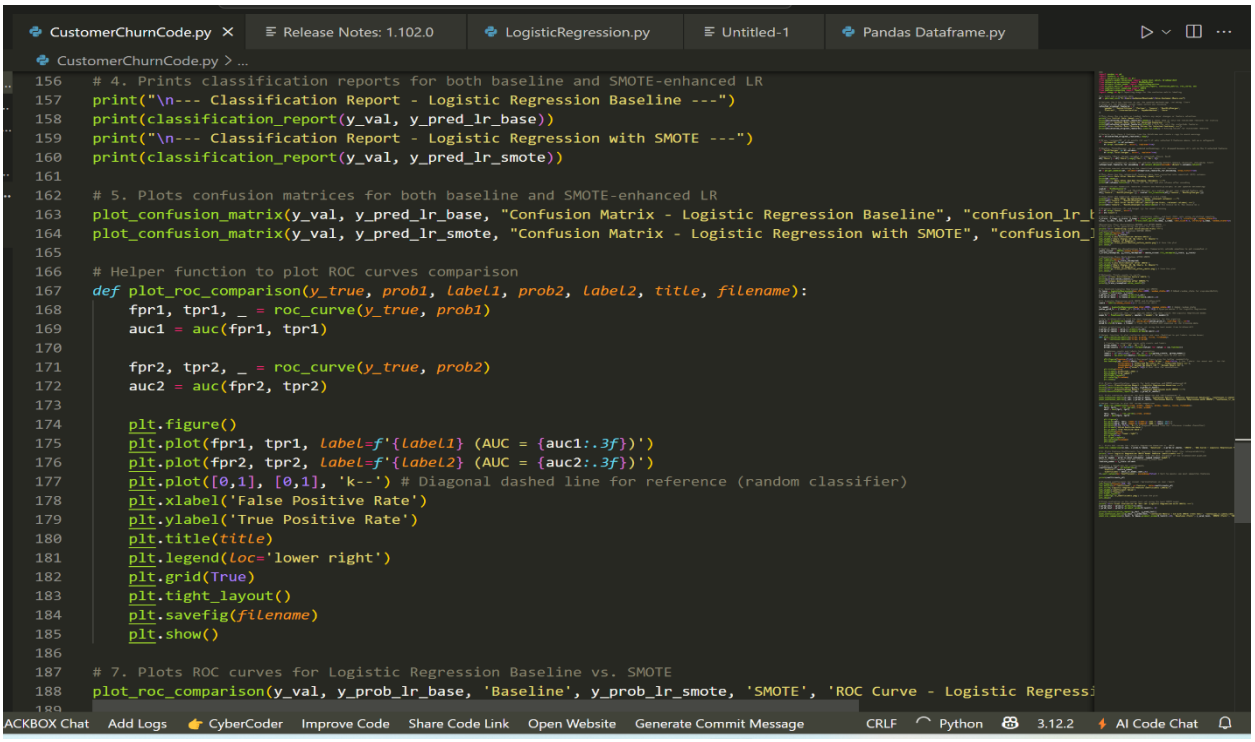
```
# This shows the numerical features scaled to a 0-1 range.
print("\n--- Data After Normalization (Head, relevant columns) ---")
print(df[['tenure', 'MonthlyCharges']].head())
print("\n--- Data After Normalization (Descriptive Stats, relevant columns) ---")
print(df[['tenure', 'MonthlyCharges']].describe()) # Min should be 0, Max should be 1

# Prepares features (X) and target (y) for model training
X = df.drop('Churn', axis=1)
y = df['Churn']

# Splits data into training (70%), validation (15%), and test (15%) sets using stratified sampling
X_train, X_temp, y_train, y_temp = train_test_split(X,y, test_size=0.3, stratify=y, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42

#Visualizes Class Distribution BEFORE and AFTER SMOTE ---
# This part is for generating the plots for the report
print("\n--- Generating Class Distribution Plots ---")
# Visualizes Class Distribution BEFORE SMOTE
plt.figure(figsize=(6, 5))
sns.countplot(x=y_train)
plt.title('Class Distribution Before SMOTE')
plt.xlabel('Churn Status (0: No Churn, 1: Churn)')
plt.ylabel('Number of Samples')
plt.savefig('class_distribution_before_smote.png') # Save the plot
plt.show()

# Applies SMOTE for Visualization Purposes (temporarily outside pipeline to get resampled y)
smote_visual = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote_visual.fit_resample(X_train, y_train)

# Visualizes Class Distribution AFTER SMOTE
plt.figure(figsize=(6, 5))
sns.countplot(x=y_train_resampled)
```



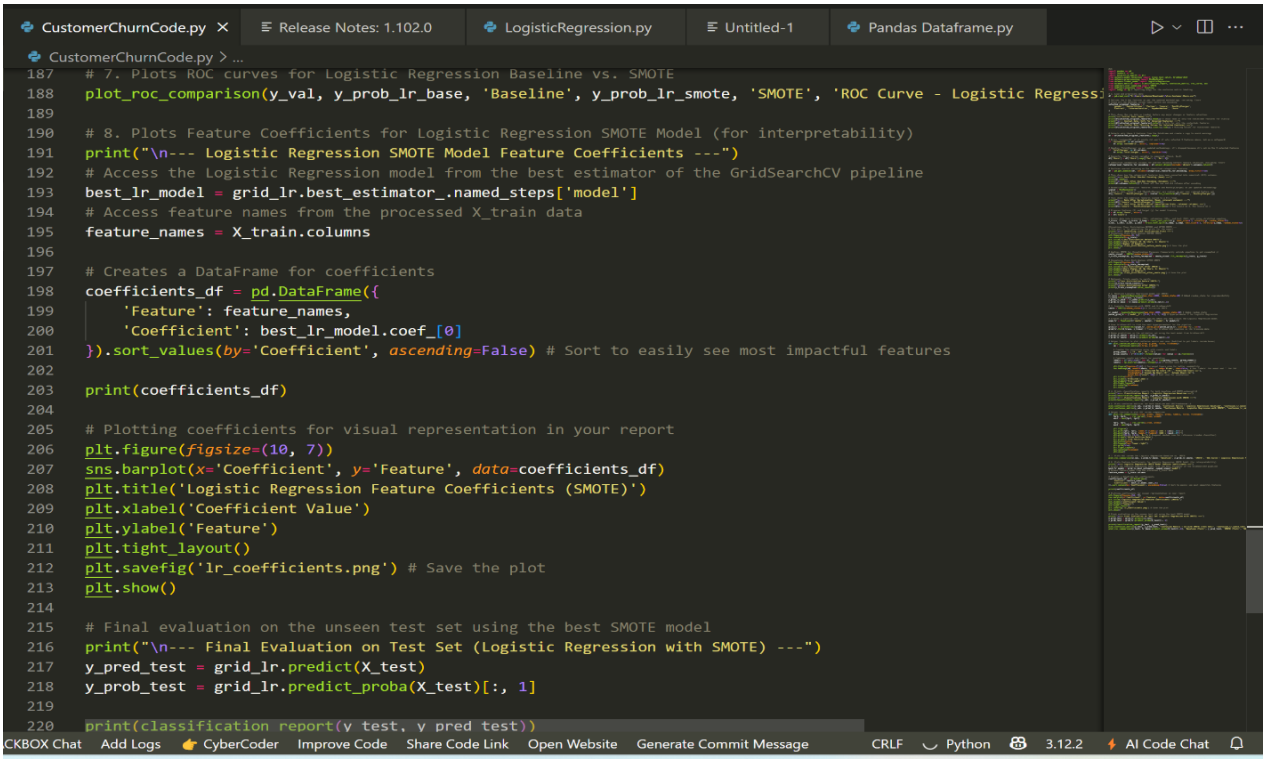
```
# Visualizes Class Distribution AFTER SMOTE
plt.figure(figsize=(6, 5))
sns.countplot(x=y_train_resampled)
plt.title('Class Distribution After SMOTE')
plt.xlabel('Churn Status (0: No Churn, 1: Churn)')
plt.ylabel('Number of Samples')
plt.savefig('class_distribution_after_smote.png') # Save the plot
plt.show()

# Optional: Prints counts to verify
print("\nClass distribution before SMOTE:")
print(y_train.value_counts())
print("\nClass distribution after SMOTE:")
print(y_train_resampled.value_counts())


# 2. Baseline Logistic Regression model (no SMOTE)
lr_base = LogisticRegression(max_iter=1000, random_state=42) # Added random_state for reproducibility
lr_base.fit(X_train, y_train)
y_pred_lr_base = lr_base.predict(X_val)
y_prob_lr_base = lr_base.predict_proba(X_val)[:,1]

# 3. Logistic Regression with SMOTE and GridSearchCV
smote = SMOTE(random_state=42) # Initialize SMOTE

lr_model = LogisticRegression(max_iter=1000, random_state=42) # Added random_state
param_grid_lr = {'model__C': [0.01, 0.1, 1, 10]} # Hyperparameter C for Logistic Regression

# Creates a pipeline that first applies SMOTE and then trains the Logistic Regression model
pipe_lr = Pipeline([('smote', smote), ('model', lr_model)])

# Uses GridSearchCV to find the best hyperparameters for the pipeline
grid_lr = GridSearchCV(pipe_lr, param_grid=param_grid_lr, scoring='f1', cv=5)
```



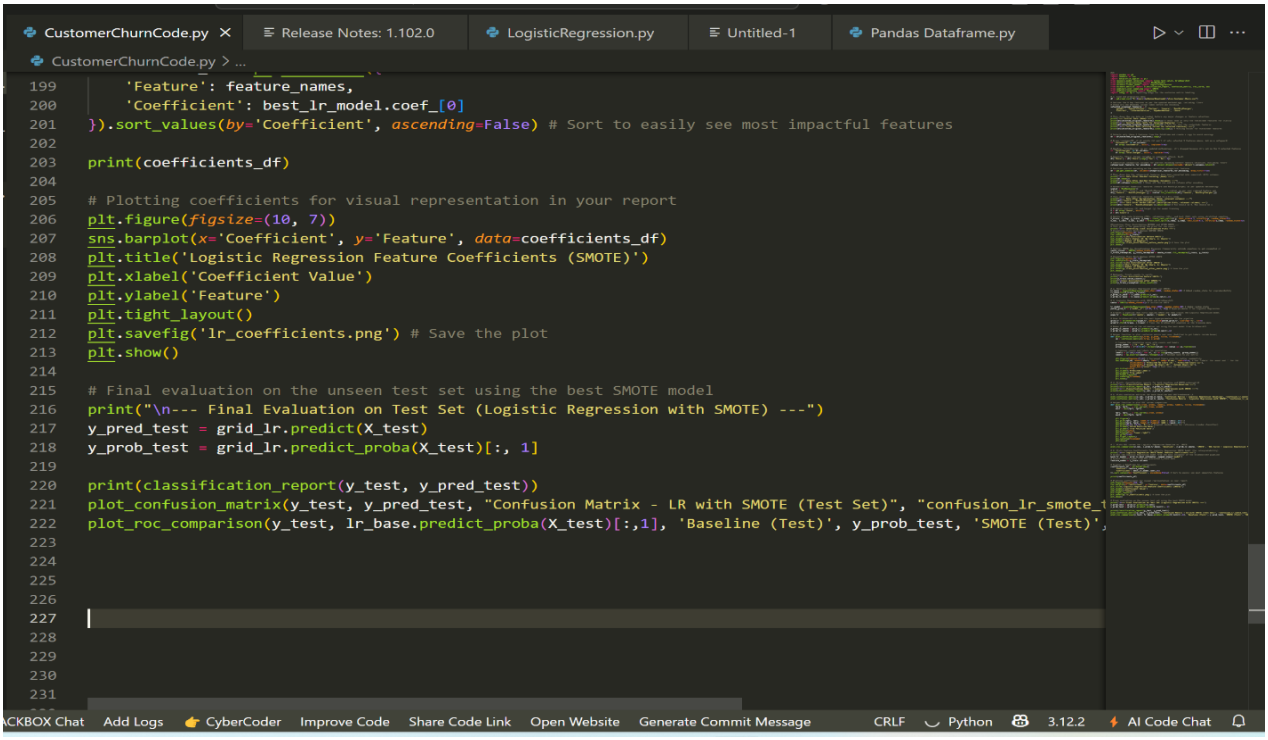
```
# Uses GridSearchCV to find the best hyperparameters for the pipeline
grid_lr = GridSearchCV(pipe_lr, param_grid=param_grid_lr, scoring='f1', cv=5)
grid_lr.fit(X_train, y_train) # Train the GridSearchCV pipeline on the training data

# Makes predictions on the validation set using the best model from GridSearchCV
y_pred_lr_smote = grid_lr.predict(X_val)
y_prob_lr_smote = grid_lr.predict_proba(X_val)[:,1]

# Helper function to plot confusion matrix and save (Modified to put labels inside boxes)
def plot_confusion_matrix(y_true, y_pred, title, filename):
    cm = confusion_matrix(y_true, y_pred)

    # Creates the annotation array with counts and labels
    group_names = ['TN','FP','FN','TP']
    group_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]

    # Combines counts and labels for annotation
    labels = [f"{v1}\n{v2}" for v1, v2 in zip(group_counts, group_names)]
    labels = np.asarray(labels).reshape(2,2) # Reshape back to 2x2 matrix

    plt.figure(figsize=(7,6)) # Increased figure size for better readability
    sns.heatmap(cm, annot=labels, fmt='', cmap='Blues', cbar=False, # Use 'labels' for annot and '' for fmt
                xticklabels=['Predicted No Churn (0)', 'Predicted Churn (1)'],
                yticklabels=['Actual No Churn (0)', 'Actual Churn (1)'],
                annot_kws={"size": 14}) # Font size for annotations
    plt.title(title)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.tight_layout()
    plt.savefig(filename)
    plt.show()

# 4. Prints classification reports for both baseline and SMOTE-enhanced LR
```



```
# 4. Prints classification reports for both baseline and SMOTE-enhanced LR
print("\n--- Classification Report - Logistic Regression Baseline ---")
print(classification_report(y_val, y_pred_lr_base))
print("\n--- Classification Report - Logistic Regression with SMOTE ---")
print(classification_report(y_val, y_pred_lr_smote))

# 5. Plots confusion matrices for both baseline and SMOTE-enhanced LR
plot_confusion_matrix(y_val, y_pred_lr_base, "Confusion Matrix - Logistic Regression Baseline", "confusion_lr_b
plot_confusion_matrix(y_val, y_pred_lr_smote, "Confusion Matrix - Logistic Regression with SMOTE", "confusion_

# Helper function to plot ROC curves comparison
def plot_roc_comparison(y_true, prob1, label1, prob2, label2, title, filename):
    fpr1, tpr1, _ = roc_curve(y_true, prob1)
    auc1 = auc(fpr1, tpr1)

    fpr2, tpr2, _ = roc_curve(y_true, prob2)
    auc2 = auc(fpr2, tpr2)

    plt.figure()
    plt.plot(fpr1, tpr1, label=f'{label1} (AUC = {auc1:.3f})')
    plt.plot(fpr2, tpr2, label=f'{label2} (AUC = {auc2:.3f})')
    plt.plot([0,1], [0,1], 'k--') # Diagonal dashed line for reference (random classifier)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(filename)
    plt.show()

# 7. Plots ROC curves for Logistic Regression Baseline vs. SMOTE
plot_roc_comparison(y_val, y_prob_lr_base, 'Baseline', y_prob_lr_smote, 'SMOTE', 'ROC Curve - Logistic Regressi
```



```
# 7. Plots ROC curves for Logistic Regression Baseline vs. SMOTE
plot_roc_comparison(y_val, y_prob_lr_base, 'Baseline', y_prob_lr_smote, 'SMOTE', 'ROC Curve - Logistic Regress

# 8. Plots Feature Coefficients for Logistic Regression SMOTE Model (for interpretability)
print("\n--- Logistic Regression SMOTE Model Feature Coefficients ---")
# Access the Logistic Regression model from the best estimator of the GridSearchCV pipeline
best_lr_model = grid_lr.best_estimator_.named_steps['model']
# Access feature names from the processed X_train data
feature_names = X_train.columns

# Creates a DataFrame for coefficients
coefficients_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': best_lr_model.coef_[0]
}).sort_values(by='Coefficient', ascending=False) # Sort to easily see most impactful features

print(coefficients_df)

# Plotting coefficients for visual representation in your report
plt.figure(figsize=(10, 7))
sns.barplot(x='Coefficient', y='Feature', data=coefficients_df)
plt.title('Logistic Regression Feature Coefficients (SMOTE)')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.tight_layout()
plt.savefig('lr_coefficients.png') # Save the plot
plt.show()

# Final evaluation on the unseen test set using the best SMOTE model
print("\n--- Final Evaluation on Test Set (Logistic Regression with SMOTE) ---")
y_pred_test = grid_lr.predict(X_test)
y_prob_test = grid_lr.predict_proba(X_test)[:, 1]

print(classification_report(y_test, y_pred_test))
```



```
    'Feature': feature_names,
    'Coefficient': best_lr_model.coef_[0]
}).sort_values(by='Coefficient', ascending=False) # Sort to easily see most impactful features

print(coefficients_df)

# Plotting coefficients for visual representation in your report
plt.figure(figsize=(10, 7))
sns.barplot(x='Coefficient', y='Feature', data=coefficients_df)
plt.title('Logistic Regression Feature Coefficients (SMOTE)')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.tight_layout()
plt.savefig('lr_coefficients.png') # Save the plot
plt.show()

# Final evaluation on the unseen test set using the best SMOTE model
print("\n--- Final Evaluation on Test Set (Logistic Regression with SMOTE) ---")
y_pred_test = grid_lr.predict(X_test)
y_prob_test = grid_lr.predict_proba(X_test)[:, 1]

print(classification_report(y_test, y_pred_test))
plot_confusion_matrix(y_test, y_pred_test, "Confusion Matrix - LR with SMOTE (Test Set)", "confusion_lr_smote_t
plot_roc_comparison(y_test, lr_base.predict_proba(X_test)[:,1], 'Baseline (Test)', y_prob_test, 'SMOTE (Test)',
```

Supplementary Data Visualizations