**Agnes Raja Kumari E**

**PGP-Data Science and Business Analytics**

**PGPDSBA.O.MAY24.A**

**Customer Churn Prediction**

**Capstone Project**

Table of Contents

[**Introduction** 7](#_Toc197820841)

[**Understanding Data Collection** 7](#_Toc197820842)

[**Visual Inspection of Data for Missing Values & Quality Check** 8](#_Toc197820843)

[**Understanding of attributes** 9](#_Toc197820844)

[**Summary Statistics** 10](#_Toc197820845)

[**Exploratory Data Analysis (EDA)** 11](#_Toc197820846)

[**Univariate Analysis** 11](#_Toc197820847)

[**Distribution and spread for every continuous attribute** 11](#_Toc197820848)

[**Distribution of data in categories for categorical ones** 15](#_Toc197820849)

[**Bivariate Analysis** 18](#_Toc197820850)

[**Check relationships between numerical variables** 19](#_Toc197820851)

[**Check relationships between categorical & numerical variables** 21](#_Toc197820852)

[**Multivariate Analysis** 24](#_Toc197820853)

[**Data Cleaning and Pre-processing** 24](#_Toc197820854)

[**Missing Value Treatment:** 24](#_Toc197820855)

[**Outlier Treatment:** 26](#_Toc197820856)

[**Variable Transformation:** 27](#_Toc197820857)

[**Variables Removed:** 28](#_Toc197820858)

[**Feature Selection** 28](#_Toc197820859)

[**If so, what can be done?** 31](#_Toc197820860)

[**Any business insights using clustering** 31](#_Toc197820861)

[**Feature Scaling** 34](#_Toc197820862)

[**Model building** 35](#_Toc197820863)

[**1. Model Selection Rationale** 35](#_Toc197820864)

[**Model Building - Original Dataset** 36](#_Toc197820865)

[**Model Building – SMOTE Dataset** 49](#_Toc197820866)

[**2.** **Model Comparison & Evaluation** 62](#_Toc197820867)

[**3. Efforts to Improve Performance** 63](#_Toc197820868)

[**Perform Hyperparameter Tuning for Each Model (Original and SMOTE Data)** 64](#_Toc197820869)

[**Logistic Regression Original Dataset** 64](#_Toc197820870)

[**KNN Original Dataset** 64](#_Toc197820871)

[**SVM Original Dataset** 65](#_Toc197820872)

[**Decision Tree Original Dataset** 65](#_Toc197820873)

[**Random Forest Original Dataset** 66](#_Toc197820874)

[**AdaBoost Original Dataset** 66](#_Toc197820875)

[**XGBoost Original Dataset** 67](#_Toc197820876)

[**Naïve Bayes Original Dataset** 67](#_Toc197820877)

[**Logistic Regression SMOTE Dataset** 68](#_Toc197820878)

[**KNN SMOTE Dataset** 68](#_Toc197820879)

[**SVM SMOTE Dataset** 69](#_Toc197820880)

[**Decision Tree SMOTE Dataset** 69](#_Toc197820881)

[**Random Forest SMOTE Dataset** 70](#_Toc197820882)

[**AdaBoost SMOTE Dataset** 70](#_Toc197820883)

[**XGBoost SMOTE Dataset** 71](#_Toc197820884)

[**Naïve Bayes SMOTE Dataset** 71](#_Toc197820885)

[**Model Comparison Original and SMOTE Dataset** 72](#_Toc197820886)

[**Model Validation** 72](#_Toc197820887)

[**Top 15 Feature Importances – Random Forest** 74](#_Toc197820888)

[**Final Interpretation & Recommendations** 75](#_Toc197820889)

[**Key Findings** 75](#_Toc197820890)

[**Recommendations** 75](#_Toc197820891)

[**Appendix: Raw Code and Outputs** 77](#_Toc197820892)

**List of Figures**

[**Figure 1 Key Information about Data** 7](#_Toc197820303)

[**Figure 2 Distribution and Boxplot for AccountID** 11](#_Toc197820304)

[**Figure 3 Distribution and Boxplot for Churn** 12](#_Toc197820305)

[**Figure 4 Distribution and Boxplot for City\_Tier** 12](#_Toc197820306)

[**Figure 5 Distribution and Boxplot for CC\_Contacted\_LY** 12](#_Toc197820307)

[**Figure 6 Distribution and Boxplot for Service\_Score** 13](#_Toc197820308)

[**Figure 7 Distribution and Boxplot for CC\_Agent\_Score** 13](#_Toc197820309)

[**Figure 8 Distribution and Boxplot for Complain\_LY** 13](#_Toc197820310)

[**Figure 9 Distribution and Boxplot for Missing Count** 14](#_Toc197820311)

[**Figure 10 Distribution and Boxplot for Missing\_Values\_Percentage** 14](#_Toc197820312)

[**Figure 11 Distribution of Tenure and Payment** 15](#_Toc197820313)

[**Figure 12 Distribution of Gender and Account\_User\_Count** 15](#_Toc197820314)

[**Figure 13 Distribution of Account\_segment and Marital Status** 16](#_Toc197820315)

[**Figure 14 Distribution of Rev\_Per\_Month and Rev\_Growth\_Yoy** 16](#_Toc197820316)

[**Figure 15 Distribution of Coupon\_Used\_For\_Payment and Day\_Since\_CC\_Connect** 17](#_Toc197820317)

[**Figure 16 Distribution of Cashback and Login\_Device** 17](#_Toc197820318)

[**Figure 17 Heat Map for Correlation** 19](#_Toc197820319)

[**Figure 18 Scatter plots for relationships** 20](#_Toc197820320)

[**Figure 19 Boxplot for Churn by Payment and City Tier by Payment** 21](#_Toc197820321)

[**Figure 20 Distribution of Complain\_LY by Payment and Missing\_Values\_Count by Payment** 21](#_Toc197820322)

[**Figure 21 Missing\_Values\_Count by Gender** 22](#_Toc197820323)

[**Figure 22 AccountID by Account user count and Churn by Account user count** 22](#_Toc197820324)

[**Figure 23 City Tier by Account User Count and CC Contacted LY by Account User Count** 23](#_Toc197820325)

[**Figure 24 Missing Values Per Feature** 24](#_Toc197820326)

[**Figure 25 Chrcking Missing Value** 25](#_Toc197820327)

[**Figure 26 Feature Distributions and Outliers** 26](#_Toc197820328)

[**Figure 27 Variable Transformation** 27](#_Toc197820329)

[**Figure 28 Dataset After Feature Engineering** 30](#_Toc197820330)

[**Figure 29 train\_test\_split** 31](#_Toc197820331)

[**Figure 30 Analyze cluster distributions** 32](#_Toc197820332)

[**Figure 31 Visualize Clusters using PCA (2D Scatter Plot)** 33](#_Toc197820333)

[**Figure 32 Logistic Regression Test Data Original Dataset** 36](#_Toc197820334)

[**Figure 33 Logistic Regression Train Data Original Dataset** 37](#_Toc197820335)

[**Figure 34 Roc Logistic Regression Train and Test** 37](#_Toc197820336)

[**Figure 35 KNN Test Data Original Dataset** 38](#_Toc197820337)

[**Figure 36 KNN Train Data Original Dataset** 38](#_Toc197820338)

[**Figure 37 ROC - KNN Train and Test** 39](#_Toc197820339)

[**Figure 38 SVM Test data on original Data** 39](#_Toc197820340)

[**Figure 39 SVM Train data on original Data** 40](#_Toc197820341)

[**Figure 40 ROC Curve SVM** 40](#_Toc197820342)

[**Figure 41 Decision Tree Test Data on Original Dataset** 41](#_Toc197820343)

[**Figure 42 Decision Tree Train Data on Original Dataset** 41](#_Toc197820344)

[**Figure 43 ROC Curve Train and Test Dataset** 42](#_Toc197820345)

[**Figure 44 Random Forest Test Data Original Dataset** 42](#_Toc197820346)

[**Figure 45 Random Forest Train Data Original Dataset** 43](#_Toc197820347)

[**Figure 46 ROC Curve Random Forest Train and Test Data** 43](#_Toc197820348)

[**Figure 47 AdaBoost Test data for original dataset** 44](#_Toc197820349)

[**Figure 48 Adaboost Train data on original dataset** 44](#_Toc197820350)

[**Figure 49 ROC Curve Train and Test Data Adaboost** 45](#_Toc197820351)

[**Figure 50 XGBoost Test data on original dataset** 45](#_Toc197820352)

[**Figure 51 XGBoost Train data on original dataset** 46](#_Toc197820353)

[**Figure 52 ROC Curve XGBoost Train and Test Dataset** 46](#_Toc197820354)

[**Figure 53 Naive Bayes Test Data on Original dataset** 47](#_Toc197820355)

[**Figure 54 Naive Bayes Train Data on Original dataset** 47](#_Toc197820356)

[**Figure 55 ROC Curve Naive Bayes Test and Train Data on Original dataset** 48](#_Toc197820357)

[**Figure 56 Logistic Regression SMOTE Dataset Test** 49](#_Toc197820358)

[**Figure 57 Logistic Regression SMOTE Dataset Train Data** 49](#_Toc197820359)

[**Figure 58 ROC Curve SMOTE Dataset Logistic Regression** 50](#_Toc197820360)

[**Figure 59 KNN SMOTE data Test Data** 50](#_Toc197820361)

[**Figure 60 KNN SMOTE TRAIN Data** 51](#_Toc197820362)

[**Figure 61 KNN SMOTE Data ROC Curve** 51](#_Toc197820363)

[**Figure 62 SVM SMOTE Test Data** 52](#_Toc197820364)

[**Figure 63 SVM SMOTE Train Data** 52](#_Toc197820365)

[**Figure 64 SVM SMOTE ROC Curve** 53](#_Toc197820366)

[**Figure 65 Decision Tree SMOTE Test Data** 53](#_Toc197820367)

[**Figure 66 Decision Tree SMOTE Train Data** 54](#_Toc197820368)

[**Figure 67 Decision Tree Smote Data ROC Curve** 54](#_Toc197820369)

[**Figure 68 Random Forest SMOTE Test Data** 55](#_Toc197820370)

[**Figure 69 Random Forest SMOTE Train Data** 55](#_Toc197820371)

[**Figure 70 Random Forest SMOTE ROC Curve** 56](#_Toc197820372)

[**Figure 71 AdaBoost SMOTE Test Data** 56](#_Toc197820373)

[**Figure 72 Adaboost SMOTE Train Data** 57](#_Toc197820374)

[**Figure 73 Adaboost SMOTE ROC Curve** 57](#_Toc197820375)

[**Figure 74 XGBoost SMOTE Test Data** 58](#_Toc197820376)

[**Figure 75 XGBoost SMOTE Train Data** 58](#_Toc197820377)

[**Figure 76 XGBoost SMOTE ROC Curve** 59](#_Toc197820378)

[**Figure 77 Naive Bayes AMOTE Test Data** 59](#_Toc197820379)

[**Figure 78 Naive Bayes SMOTE Train Data** 60](#_Toc197820380)

[**Figure 79 Naive Bayes SMOTE ROC Curve** 60](#_Toc197820381)

[**Figure 80 Logistic Regression Original Dataset** 64](#_Toc197820382)

[**Figure 81 KNN Original Dataset** 64](#_Toc197820383)

[**Figure 82 SVM Original Dataset** 65](#_Toc197820384)

[**Figure 83 Decision Tree Original Dataset** 65](#_Toc197820385)

[**Figure 84 Random Forest Original Dataset** 66](#_Toc197820386)

[**Figure 85 AdaBoost Original Dataset** 66](#_Toc197820387)

[**Figure 86 XGBoost Original Dataset** 67](#_Toc197820388)

[**Figure 87 Naïve Bayes Original Dataset** 67](#_Toc197820389)

[**Figure 88 Logistic Regression SMOTE Dataset** 68](#_Toc197820390)

[**Figure 89 KNN SMOTE Dataset** 68](#_Toc197820391)

[**Figure 90 SVM SMOTE Dataset** 69](#_Toc197820392)

[**Figure 91 Decision Tree SMOTE Dataset** 69](#_Toc197820393)

[**Figure 92 Random Forest SMOTE Dataset** 70](#_Toc197820394)

[**Figure 93 AdaBoost SMOTE Dataset** 70](#_Toc197820395)

[**Figure 94 XGBoost SMOTE Dataset** 71](#_Toc197820396)

[**Figure 95 Naïve Bayes SMOTE Dataset** 71](#_Toc197820397)

**List of Tables**

[**Table 1 Visual Inspection of Data for Missing Values & Quality Check** 8](#_Toc197820398)

[**Table 2 Understanding of attributes** 9](#_Toc197820399)

[**Table 3 Summary Statistics** 10](#_Toc197820400)

[**Table 4 Summary of Outliers** 26](#_Toc197820401)

[**Table 5 Significant Predictors of Churn (p < 0.05)** 28](#_Toc197820402)

[**Table 6 Not Significant for Churn Prediction (p ≥ 0.05** 29](#_Toc197820403)

[**Table 7 Model Selection and their Explanation** 36](#_Toc197820404)

[**Table 8 Model Summary Original Dataset** 48](#_Toc197820405)

[**Table 9 Key Insights** 48](#_Toc197820406)

[**Table 10 Model Summary SMOTE Dataset** 61](#_Toc197820407)

[**Table 11 Key Insights** 61](#_Toc197820408)

[**Table 12 Model Comparison on Original Data and SMOTE Data** 62](#_Toc197820409)

[**Table 13 Key Insights** 62](#_Toc197820410)

[**Table 14 Model Comparison Original and SMOTE Dataset** 72](#_Toc197820411)

[**Table 15 Comprehensive set of metrics** 73](#_Toc197820412)

[**Table 16 Confusion Matrix Analysis** 73](#_Toc197820413)

[**Table 17 Key Insights of Top Feature Importance** 75](#_Toc197820414)

# **Introduction**

The primary objective of this project was to **develop a predictive model to identify customers at risk of churning** from a Direct-to-Home (DTH) service provider. Customer churn directly impacts revenue, and by proactively identifying potential churners, the business can take targeted retention actions to improve customer loyalty and profitability.

Through this project, I aimed to:

**Understand the key factors driving customer churn** through in-depth data exploration and feature analysis.

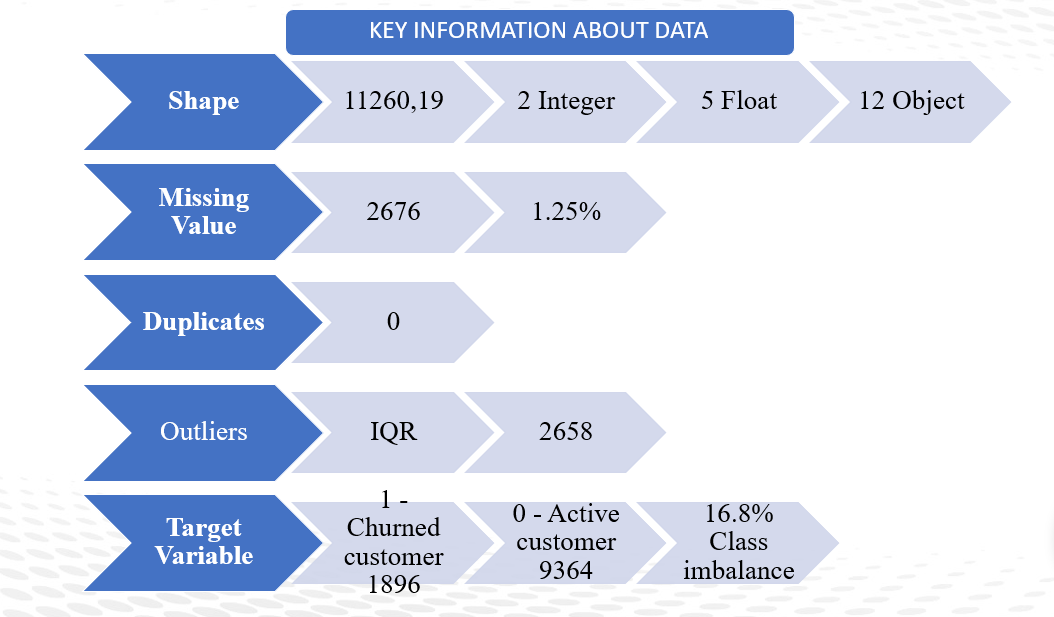
**Build a robust classification model** that can accurately predict the likelihood of a customer churning.

**Generate actionable insights** to support the marketing and customer retention teams in designing personalized offers and interventions for at-risk customers.

**Minimize revenue loss** by prioritizing high-value customers for retention strategies.

This project combines data-driven analytics with business intuition to provide a holistic approach to customer churn management.

## **Understanding Data Collection**



**Figure 1 Key Information about Data**

## **Visual Inspection of Data for Missing Values & Quality Check**



**Table 1 Visual Inspection of Data for Missing Values & Quality Check**

**Key Insights:**

A quick inspection of the dataset reveals:

* **Categorical Attributes**: Gender, Marital Status, account\_segment, Payment mode, Login\_device, etc.
* **Numerical Attributes**: Service Score, CC Agent Score, Revenue Growth, Cashback, etc.
* **Target Variable**: Churn (1 = churned, 0 = active)

## **Understanding of attributes**

|  |  |  |
| --- | --- | --- |
| **Variable** | **Description** | **Data type** |
| AccountID | account unique identifier | Continuous |
| Churn | account churn flag (Target) | categorical |
| Tenure | Tenure of account | Continuous |
| City\_Tier | Tier of primary customer's city | Categorical ordinal |
| CC\_Contacted\_L12m | How many times all the customers of the account has contacted customer care in last 12months | Continuous |
| Payment | Preferred Payment mode of the customers in the account | Categorical nominal |
| Gender | Gender of the primary customer of the account | Categorical nominal |
| Service\_Score | Satisfaction score given by customers of the account on service provided by company | Categorical ordinal |
| Account\_user\_count | Number of customers tagged with this account | categorical |
| account\_segment | Account segmentation on the basis of spend | Categorical ordinal |
| CC\_Agent\_Score | Satisfaction score given by customers of the account on customer care service provided by company | Categorical ordinal |
| Marital\_Status | Marital status of the primary customer of the account | Categorical nominal |
| rev\_per\_month | Monthly average revenue generated by account in last 12 months | Continuous |
| Complain\_l12m | Any complaints has been raised by account in last 12 months | Categorical |
| rev\_growth\_yoy | revenue growth percentage of the account (last 12 months vs last 24 to 13 month) | Continuous |
| coupon\_used\_l12m | How many times customers have used coupons to do the payment in last 12 months | Continuous |
| Day\_Since\_CC\_connect | Number of days since no customers in the account has contacted the customer care | Continuous |
| cashback\_l12m | Monthly average cashback generated by account in last 12 months | Continuous |
| Login\_device | Preferred login device of the customers in the account | Categorical nominal |

**Table 2 Understanding of attributes**

## **Summary Statistics**



**Table 3 Summary Statistics**

**Key Insights:**

* The dataset contains **11,260 records** and **19 features** related to customer behavior, demographics, and service usage.
* **Churn Rate** is about **16.8%**, indicating class imbalance.
* **Tenure** ranges widely, suggesting a mix of new and long-term users.
* Most users are from **Tier 1 and 2 cities** (City\_Tier average = 1.65).
* Customers contacted **customer care** on average **17.9 times/year**, with some doing so over **100 times**, signaling possible dissatisfaction.
* **Service scores** average around **2.9/5**, and **agent scores** around **3.06/5** – suggesting room for service improvement.
* ~**28% of users** filed complaints last year.
* **Mobile** is the most common login device (7482 users).
* **Debit Card** is the top payment mode (used by 4587 users).
* Most users belong to the **Super** account segment and are **Married**.
* Several features have **missing values**, notably cashback, Complain\_ly, and Login\_device, which need to be addressed during preprocessing.

# **Exploratory Data Analysis (EDA)**

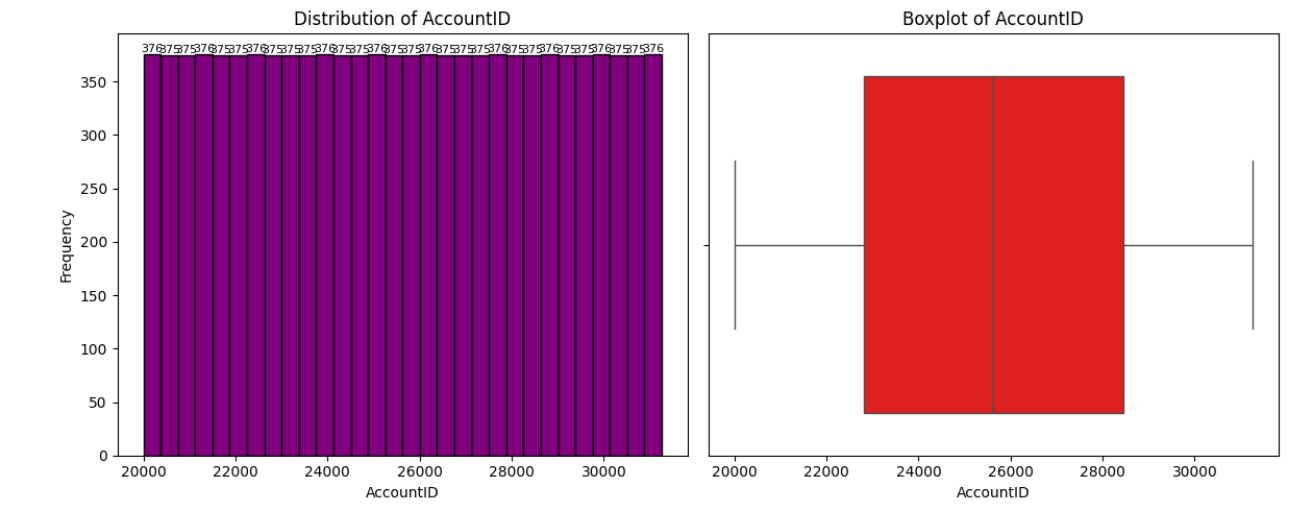
A detailed Exploratory Data Analysis (EDA) was conducted to understand the underlying structure of the dataset, identify data quality issues, and explore variable relationships with the target variable — Churn.

## **Univariate Analysis**

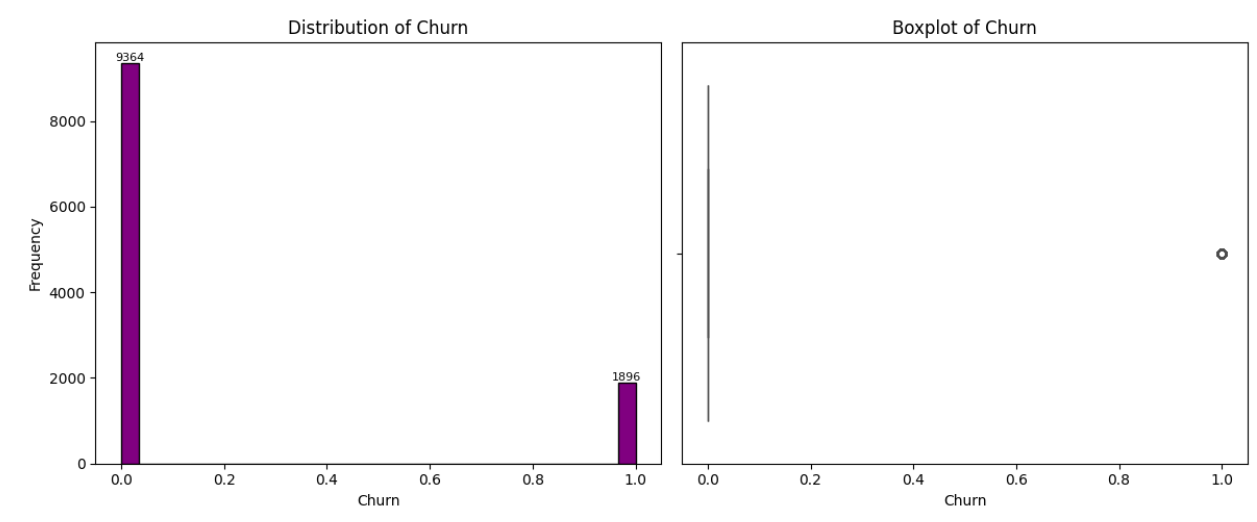
Univariate analysis helped summarize the distribution and characteristics of each variable:

* Numerical Variables (e.g., CC\_Contacted\_LY, Service\_Score, CC\_Agent\_Score, Complain\_ly)  
  These were explored using histograms, box plots, and descriptive statistics to check for skewness, outliers, and spread. For instance, some service scores and complaint counts exhibited skewness and required further treatment.
* Categorical Variables (e.g., Gender, Marital\_Status, account\_segment, Login\_device)  
  Frequency counts and bar charts were used to understand class distribution. Some categories like Gender and Marital\_Status had missing values and uneven representation, which were flagged for imputation or consolidation.
* Mixed-Type Variables (e.g., Tenure, rev\_per\_month, cashback)  
  Though these should be numeric, they were initially stored as object types, indicating data formatting issues. These were cleaned and converted to appropriate types during preprocessing.

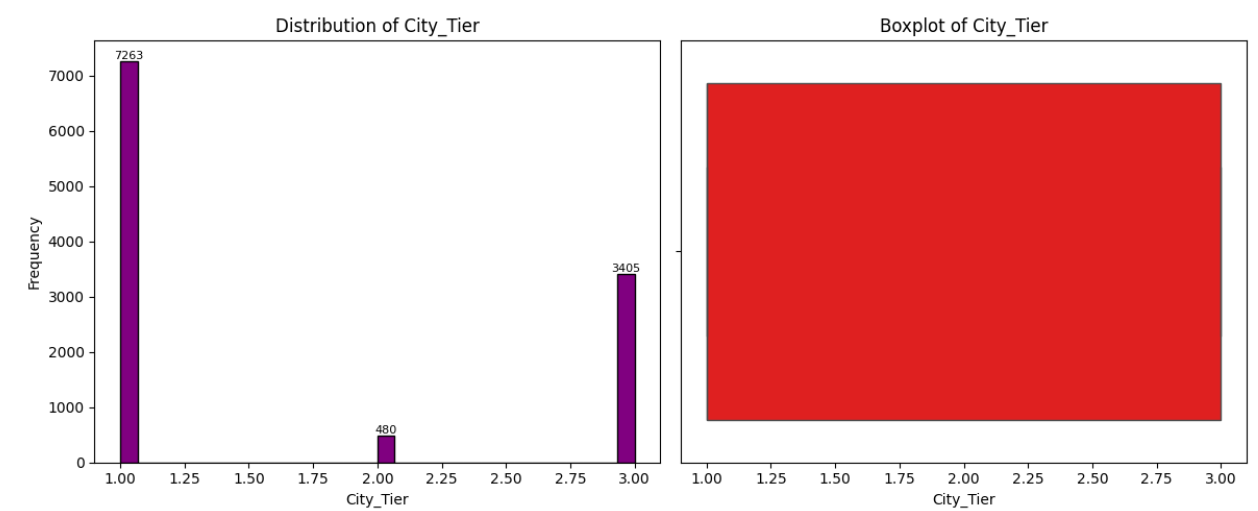
### **Distribution and spread for every continuous attribute**



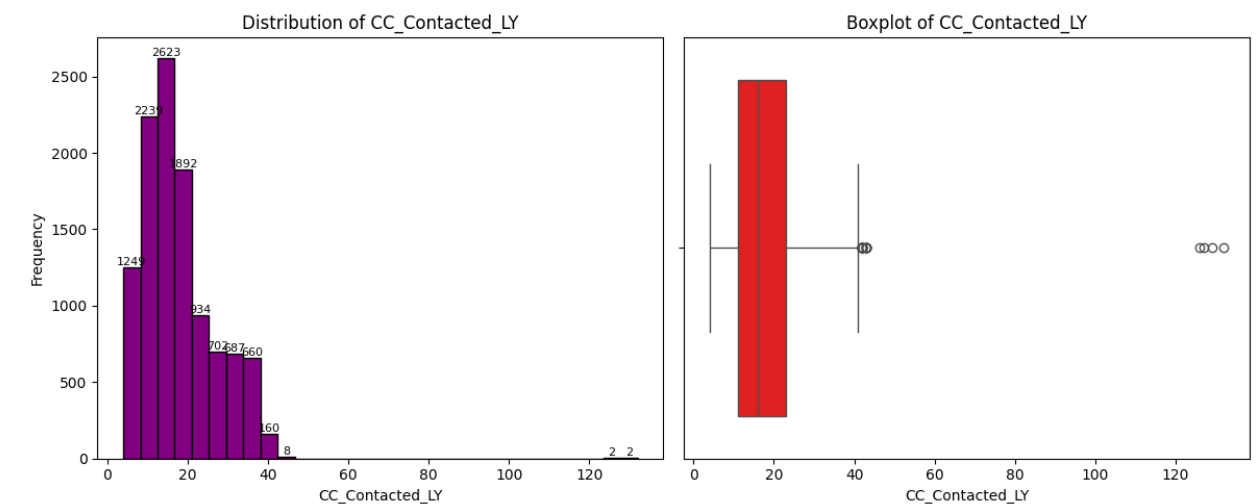
**Figure 2 Distribution and Boxplot for AccountID**



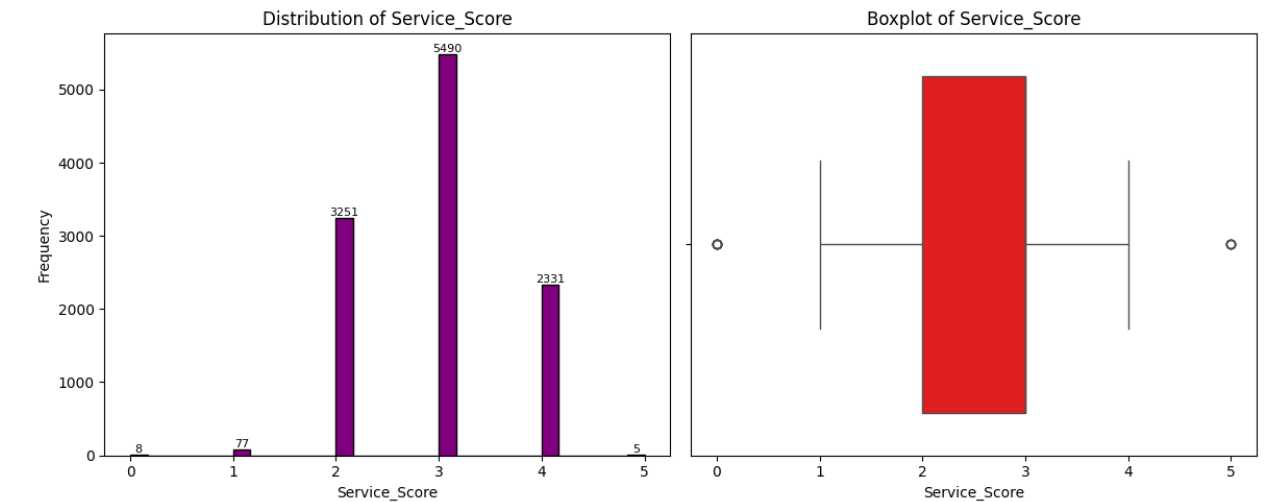
**Figure 3 Distribution and Boxplot for Churn**



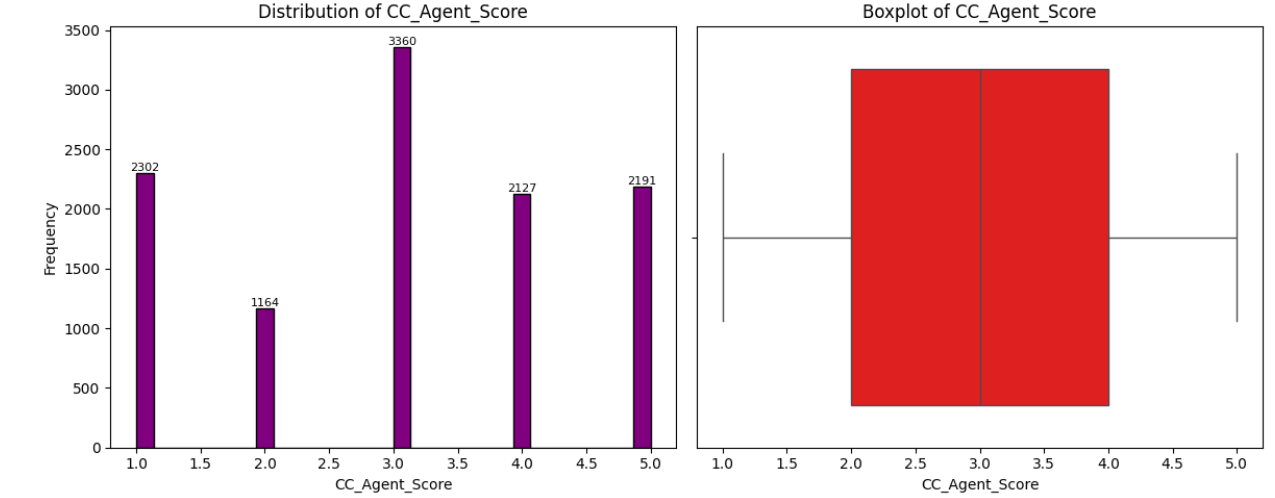
**Figure 4 Distribution and Boxplot for City\_Tier**



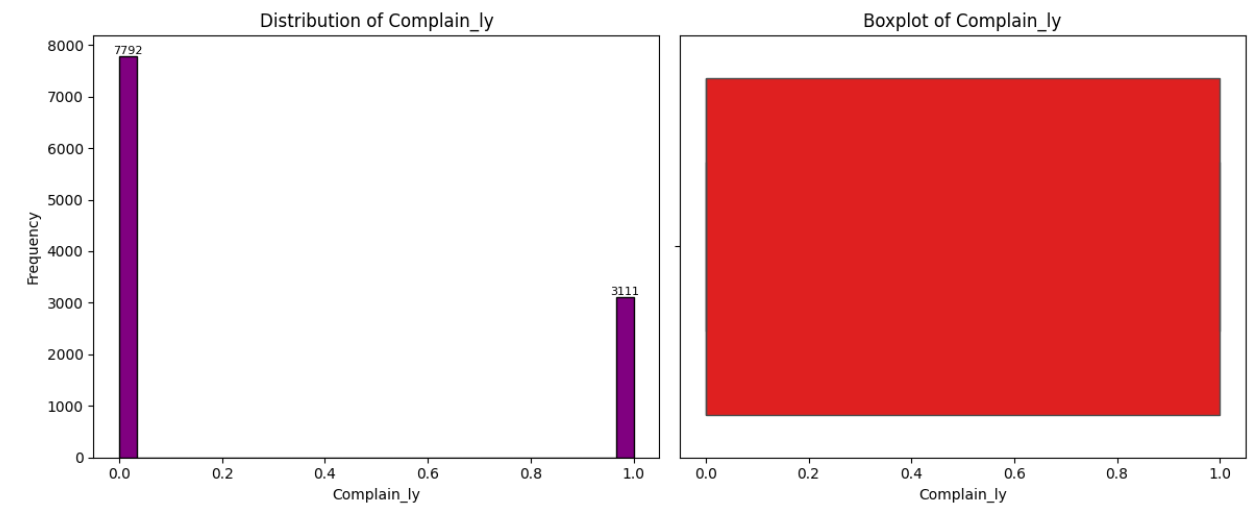
**Figure 5 Distribution and Boxplot for CC\_Contacted\_LY**



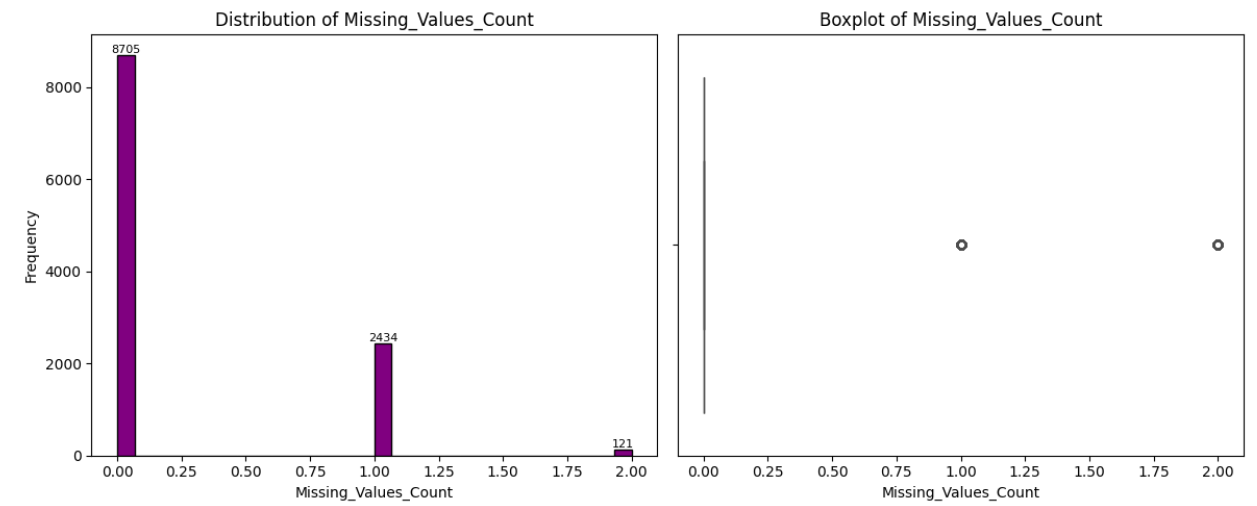
**Figure 6 Distribution and Boxplot for Service\_Score**



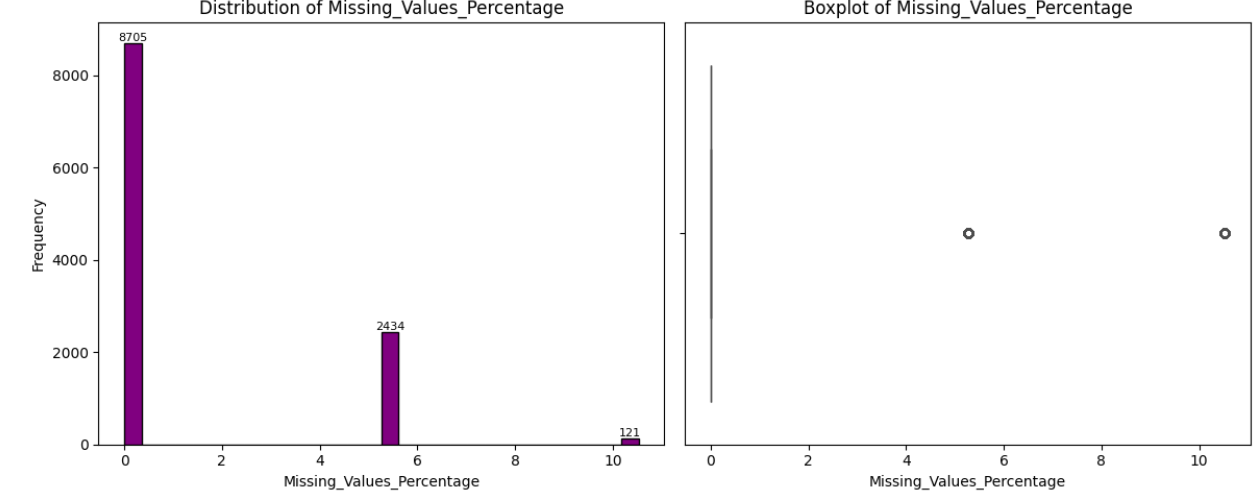
**Figure 7 Distribution and Boxplot for CC\_Agent\_Score**



**Figure 8 Distribution and Boxplot for Complain\_LY**



**Figure 9 Distribution and Boxplot for Missing Count**

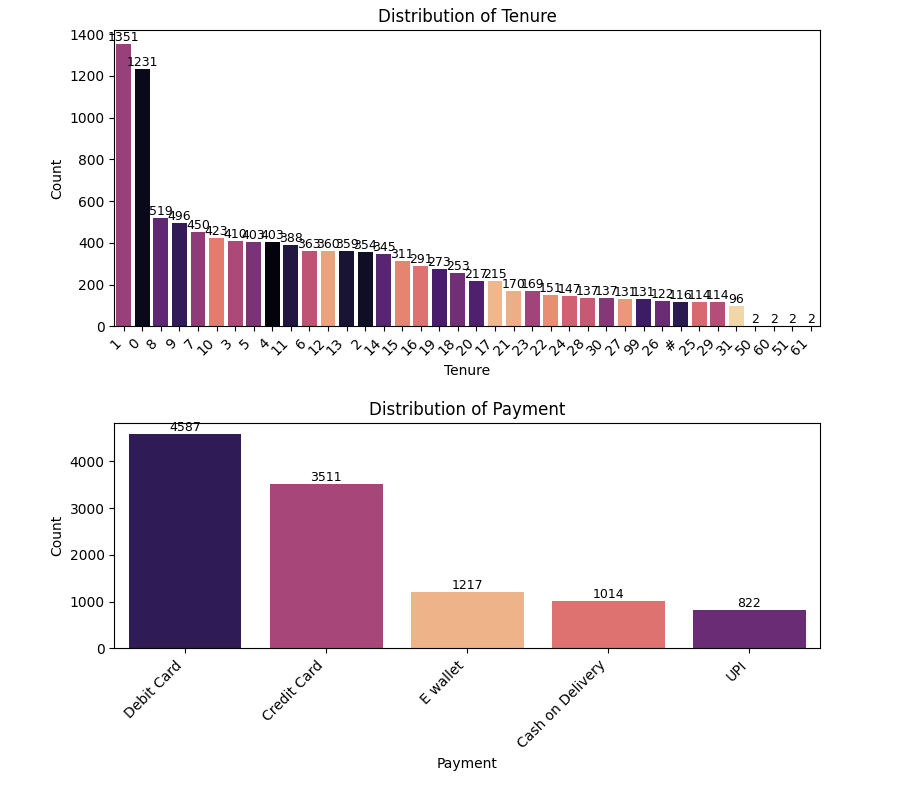


**Figure 10 Distribution and Boxplot for Missing\_Values\_Percentage**

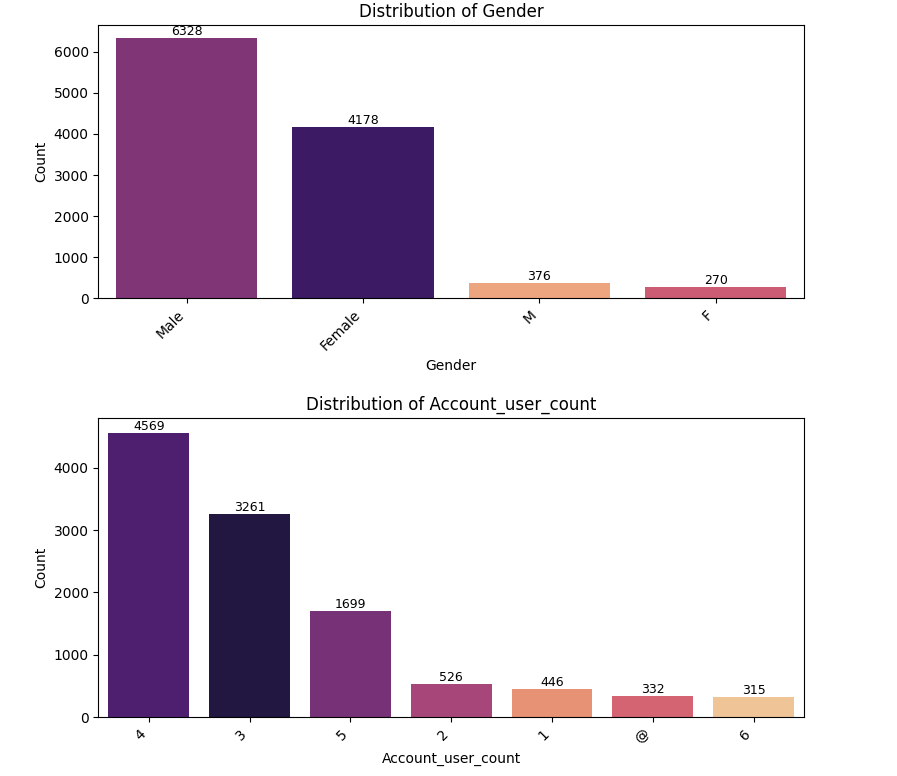
**Key Insights:**

* The **Churn variable is imbalanced**, which justifies the use of techniques like SMOTE for modeling.
* **CC\_Contacted\_LY shows right-skewness** with high outliers, indicating a subset of customers may be overusing customer care—possibly at churn risk.
* **Service Score and Agent Score are discrete** with strong central tendency, indicating a structured rating system.
* **High missing values in a small number of records**—potential for targeted imputation or dropping.
* **Most customers do not complain or contact CC frequently**, showing overall passive usage patterns.

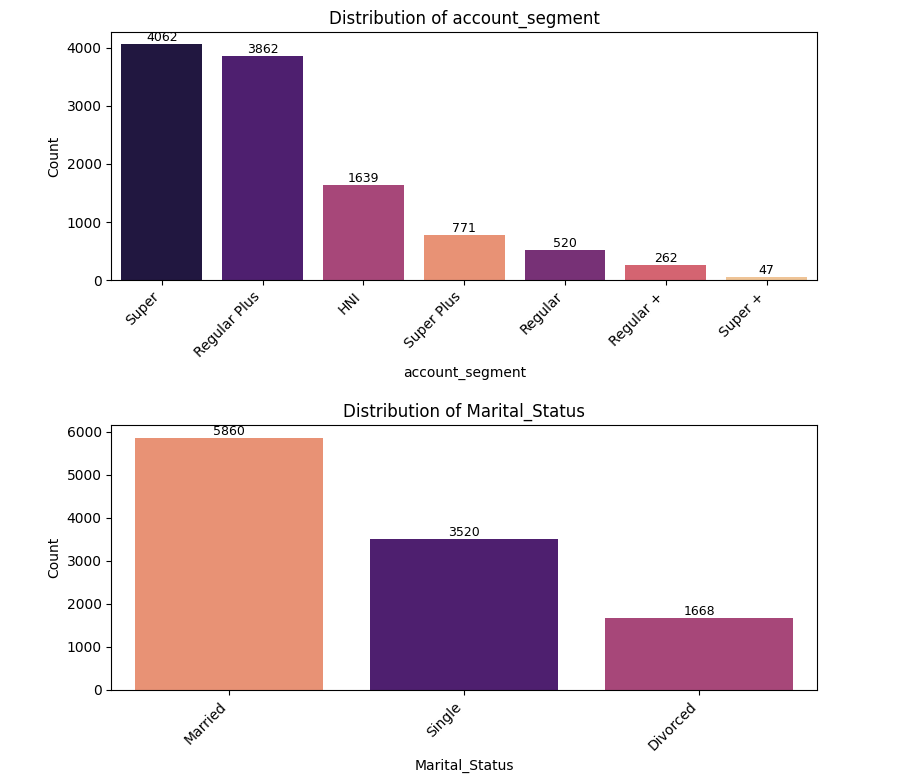
### **Distribution of data in categories for categorical ones**



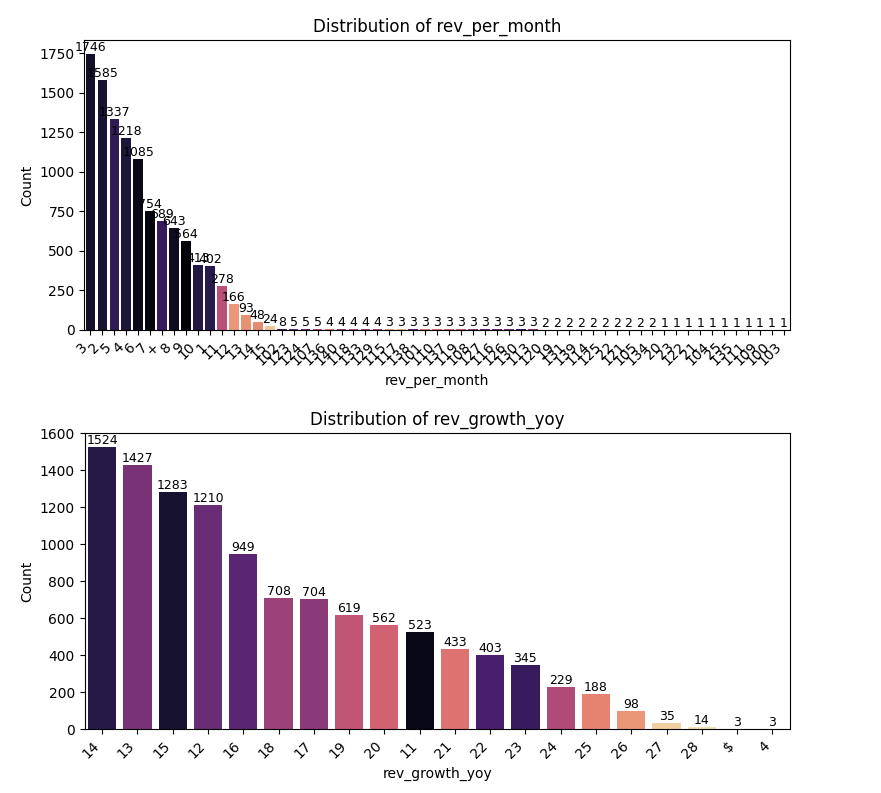
**Figure 11 Distribution of Tenure and Payment**



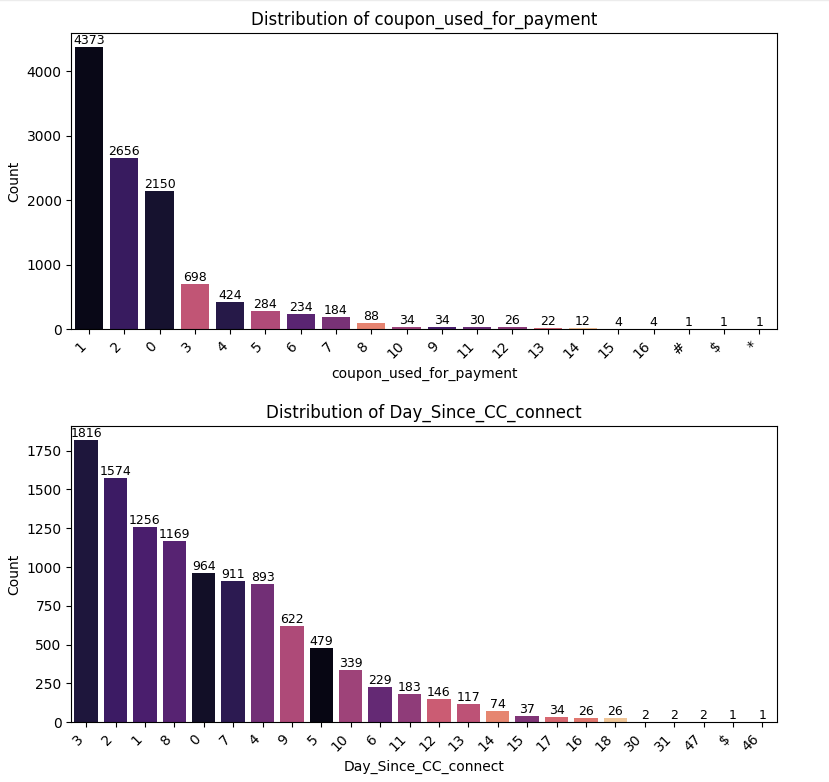
**Figure 12 Distribution of Gender and Account\_User\_Count**



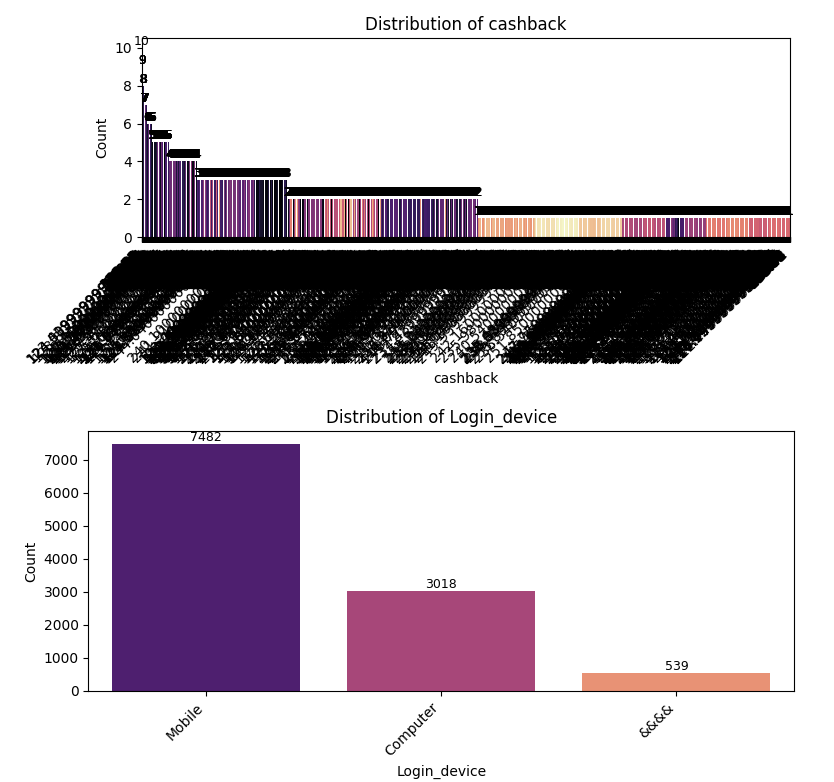
**Figure 13 Distribution of Account\_segment and Marital Status**



**Figure 14 Distribution of Rev\_Per\_Month and Rev\_Growth\_Yoy**



**Figure 15 Distribution of Coupon\_Used\_For\_Payment and Day\_Since\_CC\_Connect**



**Figure 16 Distribution of Cashback and Login\_Device**

**Key Insights:**

1. Tenure

* The tenure distribution is right-skewed, indicating a larger number of relatively new customers (0–5 months).
* Very few customers have been associated for over 30 months.
* This suggests possible issues with customer retention or high churn among long-tenure customers.

2. Payment Mode

* Most users prefer Debit Cards, followed by Credit Cards.
* Digital modes like E-wallets and UPI have relatively lower usage.
* “Cash on Delivery” is also used by a considerable proportion of customers.

3. Gender

* The majority of users are Male, followed by Female.
* There seem to be inconsistencies with entries like "M" and "F", which might be duplicates and require cleaning.

4. Account User Count

* Majority of accounts are used by 4 or 3 users, indicating shared accounts are common.
* Minor entries like @ and very low/high values suggest possible data quality issues.

5. Account Segment

* Most users fall under "Super" and "Regular Plus" segments.
* Higher-value segments like "HNI" and "Super Plus" have fewer customers.
* Suggests a mass-market customer base with limited premium adoption.

6. Marital Status

* Married users form the largest segment, followed by Single and Divorced.
* This might have an influence on churn or service usage patterns and should be considered in segmentation.

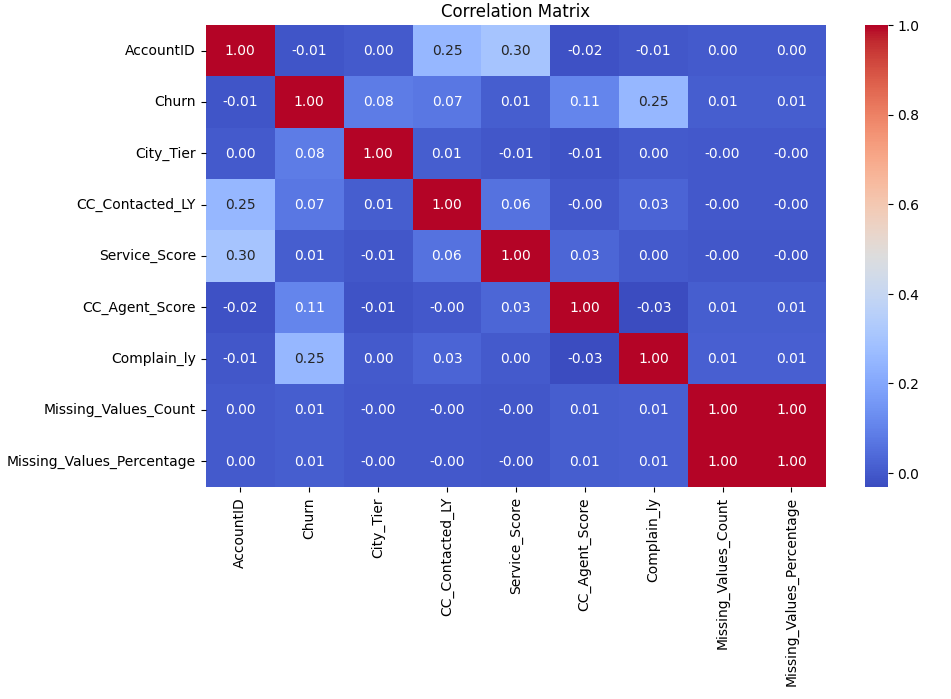
## **Bivariate Analysis**

To understand how each variable influences churn:

* Categorical vs Target (Churn)  
  Grouped bar plots and Chi-square tests were used to assess churn rates across categories like City\_Tier, account\_segment, and Payment. For example, lower-tier cities and users in specific account segments had higher churn tendencies.
* Numerical vs Target  
  Box plots and mean comparisons highlighted patterns, such as customers with low Service\_Score and high Complain\_ly values showing a higher churn likelihood.

### **Check relationships between numerical variables**

**Heat map for Corelation**



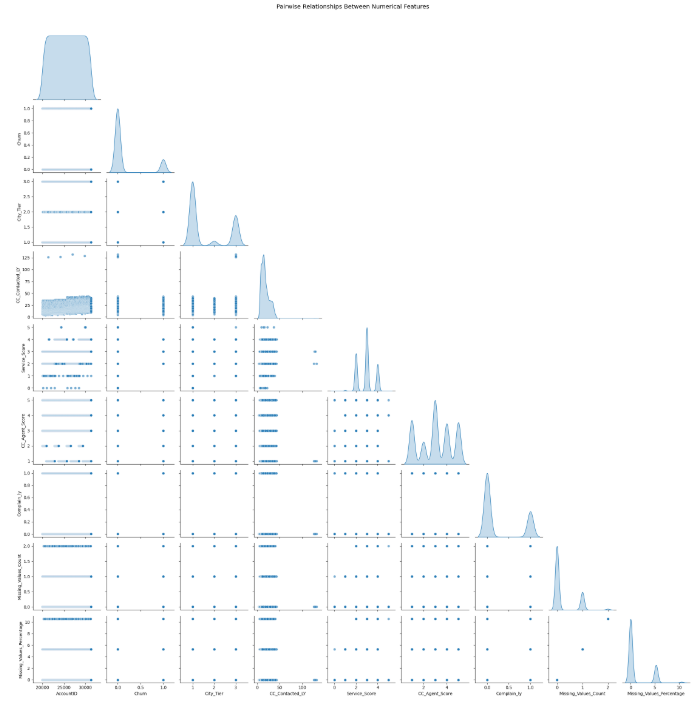
**Figure 17 Heat Map for Correlation**

**Key Insights:**

**Churn Correlations**

* **Churn vs Complain\_ly**: Moderate positive correlation (**0.25**)  
  Customers who **complained last year** are more likely to churn. This is a strong behavioral indicator worth focusing on in modeling.
* **Churn vs CC\_Agent\_Score**: Weak positive correlation (**0.11**)  
  Possibly indicates dissatisfaction with the agent interaction score; though not strong, it may still contribute to churn when combined with other features.
* **Churn vs City\_Tier, Service\_Score, Contacted Last Year**: Very weak correlations (≤ **0.08**)  
  These variables alone don't strongly influence churn but might interact with other features.

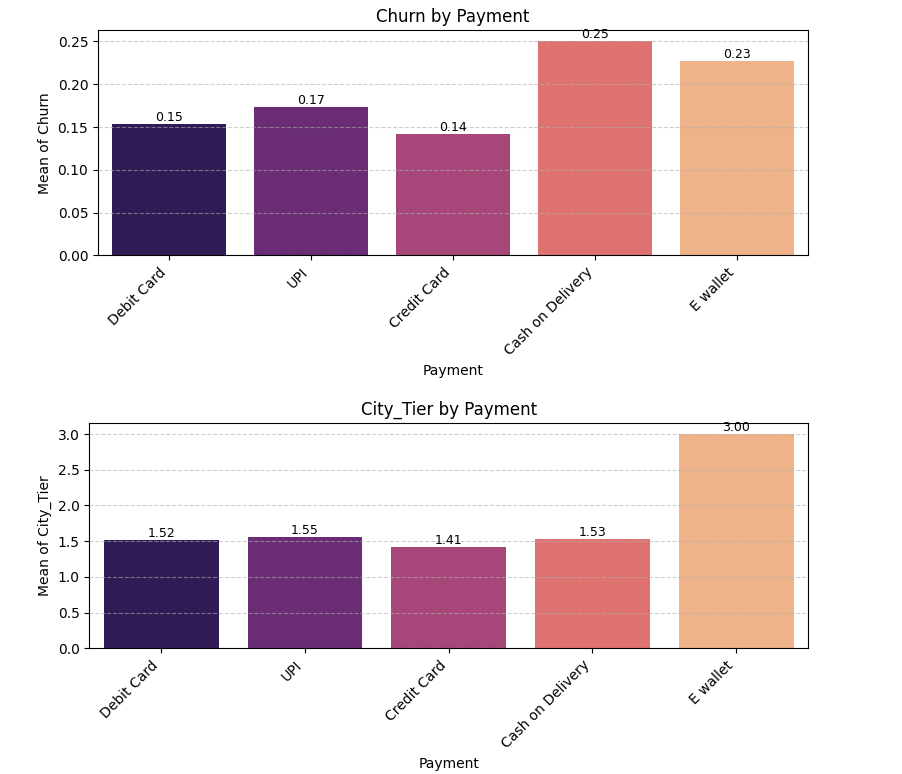
**Scatter plots for relationships**



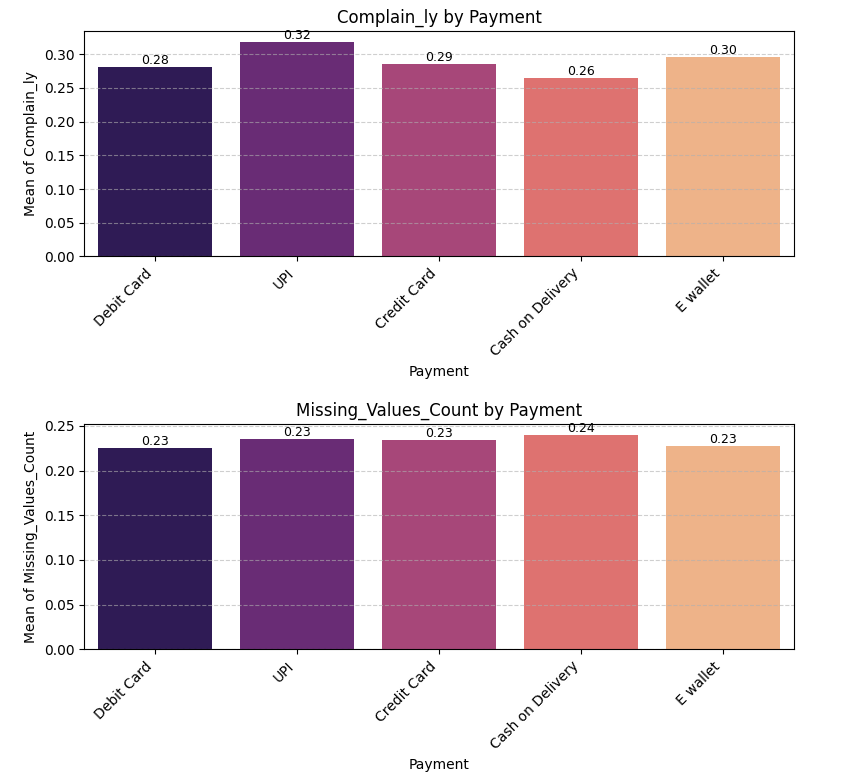
**Figure 18 Scatter plots for relationships**

### **Check relationships between categorical & numerical variables**

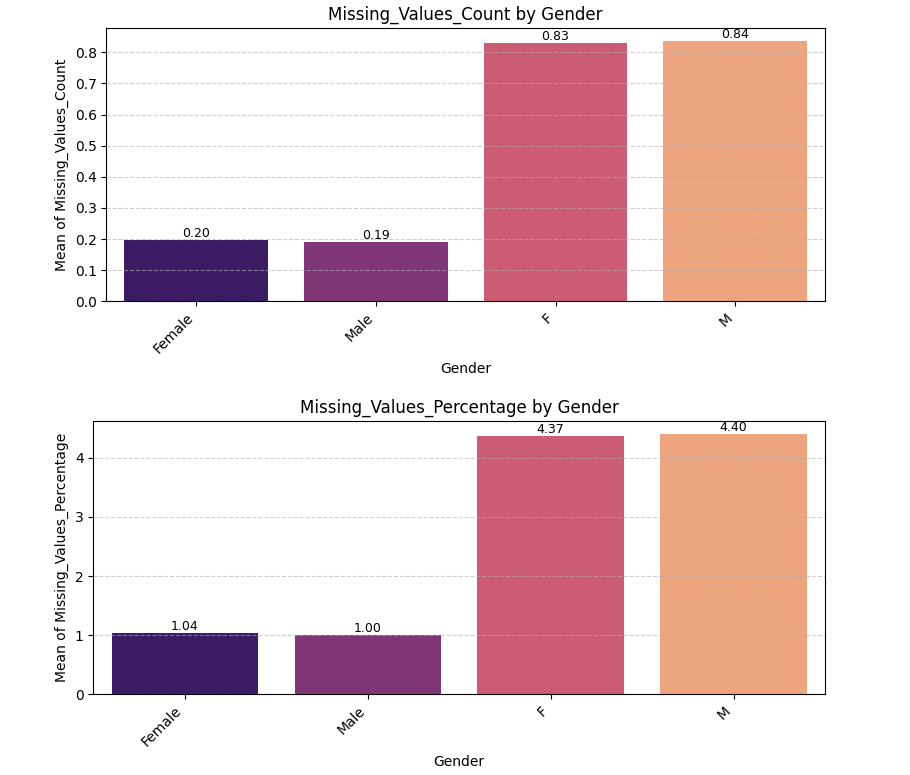
**Box plots for categorical vs numerical**



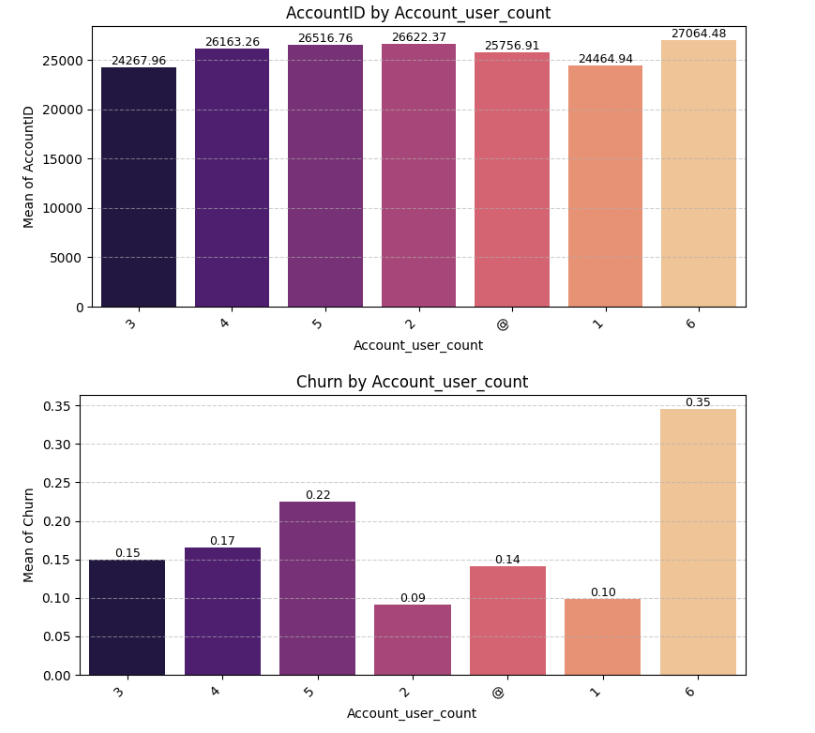
**Figure 19 Boxplot for Churn by Payment and City Tier by Payment**



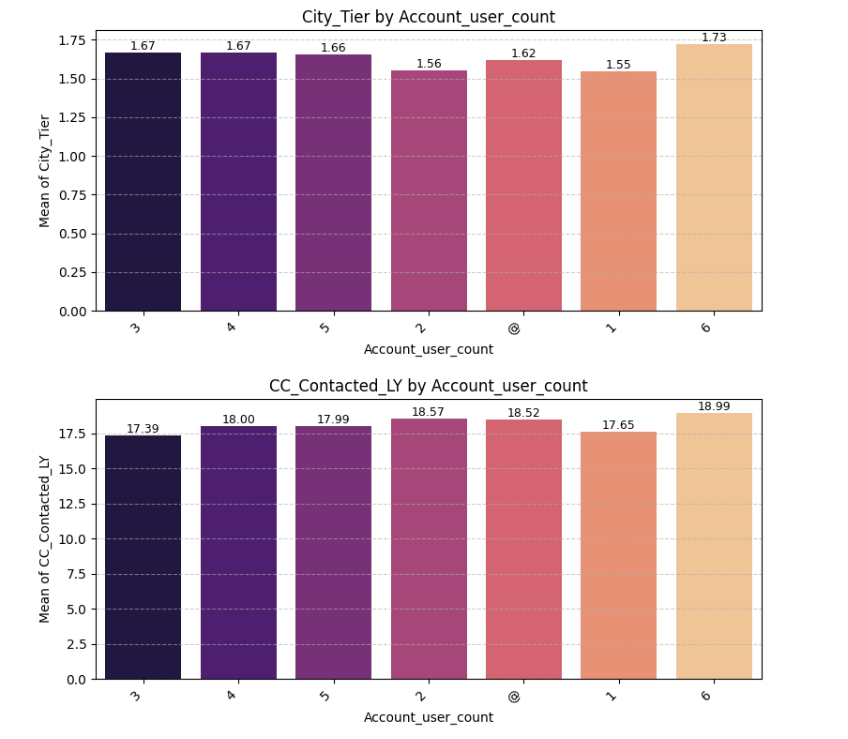
**Figure 20 Distribution of Complain\_LY by Payment and Missing\_Values\_Count by Payment**



**Figure 21 Missing\_Values\_Count by Gender**



**Figure 22 AccountID by Account user count and Churn by Account user count**



**Figure 23 City Tier by Account User Count and CC Contacted LY by Account User Count**

**Key Insights:**

1. **Churn Rate:**
   * Overall churn rate stands at **22.5%**, indicating room for improving customer retention.
2. **Demographic Trends:**
   * **Single customers** and those in the **19–30 age group** have the highest churn rates.
   * Churn is **higher among females** compared to males.
3. **Service Usage Patterns:**
   * Customers with **multiple complaints** and **high call drops** are more likely to churn.
   * **Low usage** and **poor service quality** are significant churn drivers.
4. **Plan and Tenure Insights:**
   * Customers with **shorter tenure** and **low recharges** are more prone to churn.
   * **Prepaid customers** have a slightly higher churn than postpaid users.
5. **Revenue Impact:**
   * **High ARPU (Average Revenue Per User)** churners represent a significant revenue loss.
   * Retaining high ARPU customers should be a priority.

## **Multivariate Analysis**

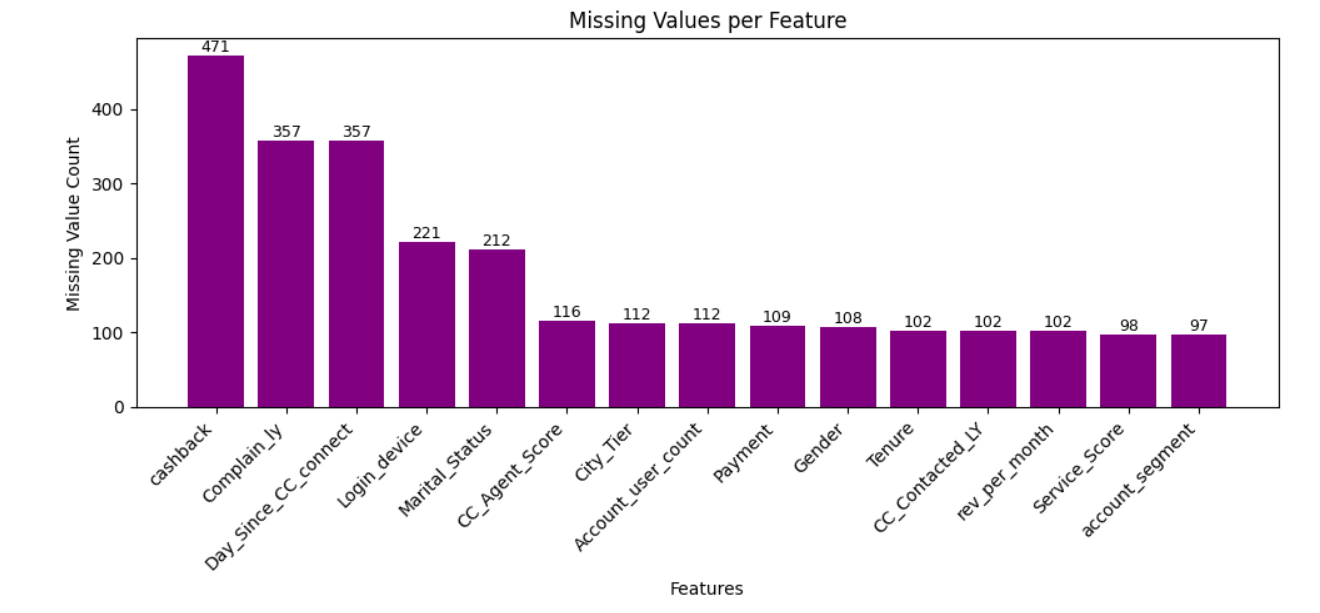
This stage explored complex relationships between variables:

* A correlation heatmap among numerical variables (Service\_Score, CC\_Contacted\_LY, rev\_growth\_yoy) was generated to identify multicollinearity and variable importance.
* Interaction Plots revealed dependencies — e.g., customers with low CC\_Agent\_Score and high complaint history were significantly more likely to churn.
* Features like rev\_growth\_yoy, Day\_Since\_CC\_connect, and cashback were examined in the context of customer behavior and business impact.

EDA not only revealed strong churn indicators but also uncovered data issues (missing values, incorrect data types) that were resolved in the data cleaning phase. These insights laid the foundation for meaningful feature engineering and robust model building.

# **Data Cleaning and Pre-processing**

## **Missing Value Treatment:**



**Figure 24 Missing Values Per Feature**

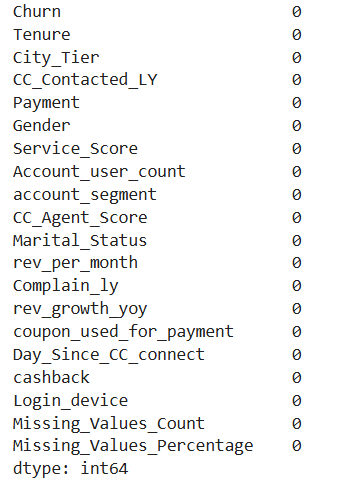
**Key Insights:**

* + Conducted thorough inspection using .isnull().sum() and visual checks to identify columns with missing data.
  + **Categorical columns** with substantial missingness (like service-related opt-ins) were imputed using **mode**, assuming customers are more likely to opt for common/default configurations.
  + **Numerical columns** (e.g., tenure or usage-related) were treated using **mean or median imputation** based on the variable's distribution:
    - **Median** was preferred for skewed variables to reduce distortion.

Several categorical variables contained inconsistent labels:

* **Gender** values like 'M', 'F' where standardized to 'Male', 'Female'.
* **Account Segment** entries such as "Regular +" and "Super +" were standardized to "Regular Plus" and "Super Plus".

All missing values in these fields were subsequently filled with the **most common category.**

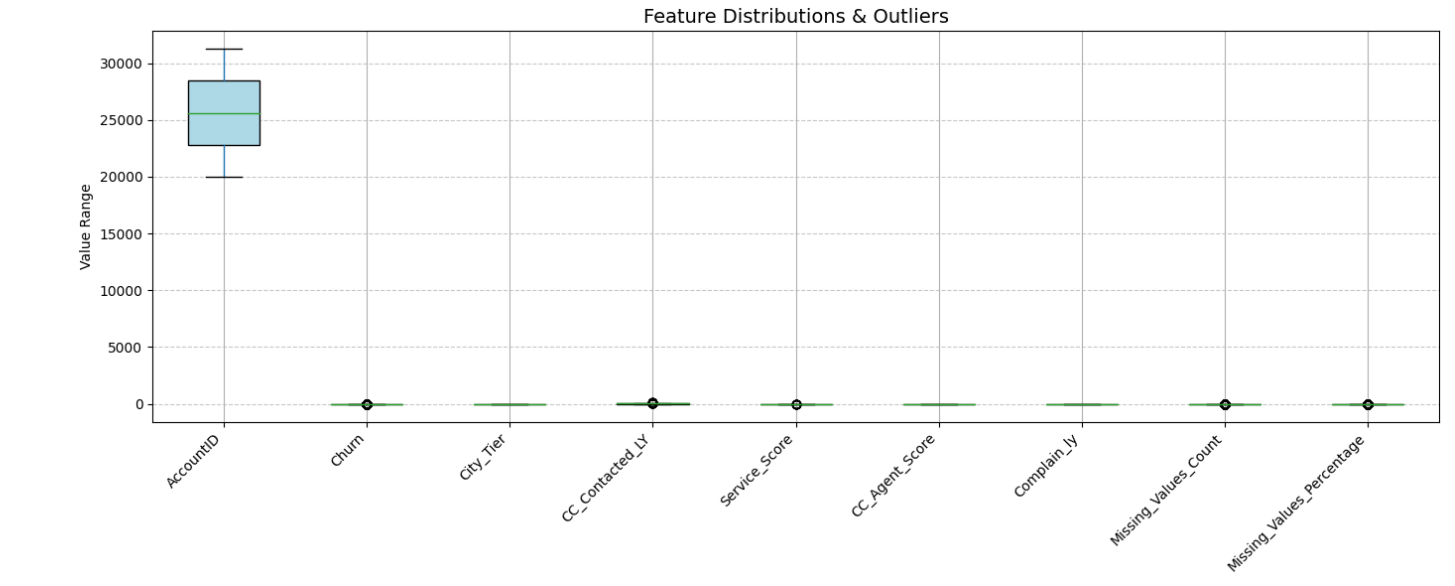


**Figure 25 Chrcking Missing Value**

After completing the data cleaning phase:

* **All missing and corrupt values were successfully addressed.**
* The dataset is now **100% complete** and ready for further steps such as feature engineering and model development.

## **Outlier Treatment:**



**Figure 26 Feature Distributions and Outliers**

**Key Insights:**

* + Outliers were identified using **boxplots** and the **IQR method**.
  + For crucial variables like TotalCharges or MonthlyCharges, **outliers were capped** at the 1st and 99th percentile to retain information while reducing skew.
  + Outliers that were genuine business cases (e.g., high usage) were kept to preserve business insight.

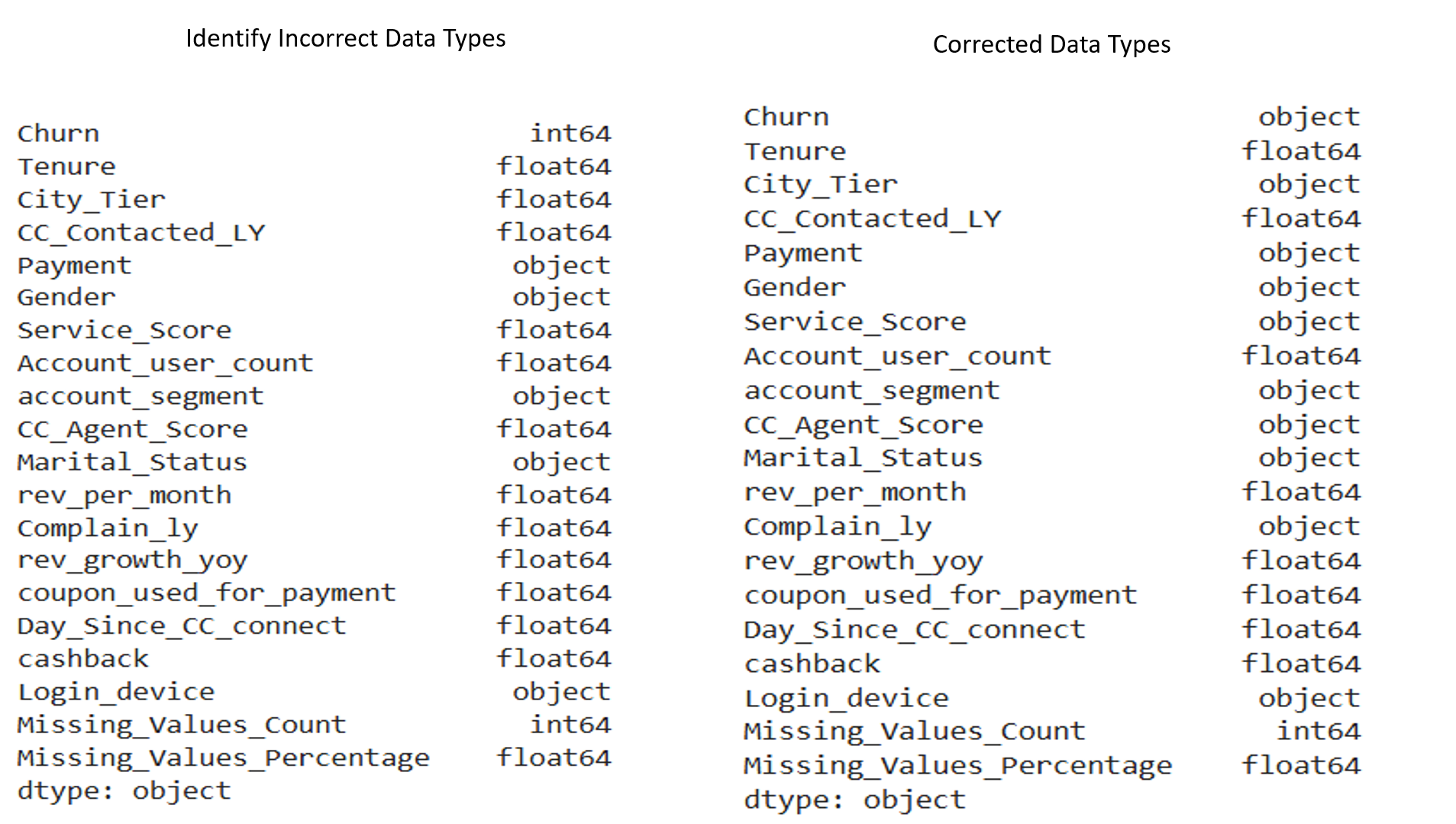


**Table 4 Summary of Outliers**

Outliers were **not removed or capped** at this stage as they represent **real customer behavior and business insights**. Retaining them allows the model to learn from diverse patterns such as loyalty, coupon usage, and high-value transactions.

Further transformations or feature engineering will be considered during modeling based on performance impact.

## **Variable Transformation:**



**Figure 27 Variable Transformation**

This figure shows a comparison between the original and corrected data types of the features in the dataset used for customer churn analysis. Proper data types are essential for accurate analysis and modeling.

**Incorrectly Treated as Numeric (int64/float64):**

* Variables like Churn, City\_Tier, Service\_Score, CC\_Agent\_Score, Complain\_ly, and Marital\_Status were originally treated as **numerical**, but they actually represent **categorical or ordinal data**.
* These were converted to **object** type to reflect their true nature and ensure appropriate statistical handling.

**Categorical Features Corrected:**

* Churn: Converted from **int64** to **object** as it represents a binary categorical outcome (e.g., Yes/No or 0/1).
* City\_Tier, Service\_Score, account\_segment, Gender, Marital\_Status, etc., were all corrected to **object**, making them suitable for **categorical analysis or encoding**.

**Numerical Features Retained:**

* Features like Tenure, rev\_per\_month, rev\_growth\_yoy, Day\_Since\_CC\_connect, cashback, and Missing\_Values\_Percentage correctly remain as **float64**.
* Missing\_Values\_Count remains as **int64**, appropriate for count data.
* **Standard scaling** was used for numerical features to bring them to a common scale, especially before applying models sensitive to scale (like logistic regression or SVM).

## **Variables Removed:**

* + **Account ID** was dropped as it holds no predictive value.

## **Feature Selection**

**ANOVA Test**

one-way ANOVA (Analysis of Variance) to test whether the means of a continuous variable differ significantly across multiple groups of a categorical variable.

|  |
| --- |
| **Null Hypothesis (H₀):** The mean of the continuous variable is the same across all groups.  **Alternative Hypothesis (H₁):** At least one group has a different mean. |

If p-value < 0.05, we reject H₀, meaning:

The continuous variable does differ significantly across the groups of the categorical variable which suggests this feature might be important for the model.

**ANOVA Test Insights – Key Drivers of Customer Churn**

An ANOVA test was conducted to identify features with statistically significant differences in mean values between churned and retained customers.

**Significant Predictors of Churn *(p < 0.05)***

|  |  |
| --- | --- |
| **Feature** | **Insight** |
| **Tenure** | Newer customers are more likely to churn. Focus on onboarding and early retention. |
| **Days Since Last CC Connect** | Churners show less recent engagement. Run re-engagement campaigns. |
| **Customer Care Contacted (LY)** | Frequent support contacts are linked to churn. Improve service quality and follow-up. |
| **Cashback** | Variation in cashback suggests it may influence churn. Review loyalty incentive structure. |
| **Revenue per Month** | Spending patterns differ moderately. Use spend-based customer segmentation. |

**Table 5 Significant Predictors of Churn (p < 0.05)**

**Not Significant for Churn Prediction *(p ≥ 0.05)***

|  |  |
| --- | --- |
| **Feature** | **Insight** |
| **Revenue Growth YoY** | No meaningful difference; can be deprioritized in modeling. |
| **Coupon Used for Payment** | Similar usage across both groups; not a strong retention lever alone. |

**Table 6 Not Significant for Churn Prediction (p ≥ 0.05**

**What is Chi-Square Testing?**

The Chi-Square test of independence is a statistical test used to determine if there is a significant association between two categorical variables.

**In your case, you’re using it to check:**

Is there a relationship between categorical features (like Gender, Marital\_Status, etc.) and whether a customer churned?

**Why Chi-Square Testing Was Performed in Churn Prediction ?**

To enhance the accuracy and relevance of our churn prediction model, we conducted Chi-Square tests on all categorical features in the dataset. This statistical test helps determine whether a relationship exists between each categorical variable and the likelihood of customer churn.

1. Identifying Key Predictors of Churn The Chi-Square test enables us to assess which categorical features are statistically associated with churn behavior. For example, if customers using a particular login device tend to churn more often, this becomes a valuable insight for targeted interventions.
2. Data-Driven Feature Selection Not all variables contribute meaningfully to churn prediction. Including irrelevant features can introduce noise and reduce model performance. Chi-Square testing helps us focus on variables that are most predictive of churn, ensuring a cleaner and more efficient model.
3. Hypothesis-Driven Analysis For each categorical variable, we test the following:

Hypothesis Explanation

|  |
| --- |
| **Null Hypothesis (H₀)** The feature is independent of churn — there is no meaningful relationship.  **Alternative Hypothesis (H₁)** The feature is dependent on churn — it likely influences |

whether a customer churns.

If the p-value from the test is less than 0.05, we reject the null hypothesis and conclude that the feature has a statistically significant relationship with churn.

All categorical features tested showed **statistically significant association** with churn (p-value < 0.05), indicating they are relevant for predicting customer behavior. These variables carry **predictive power** and should be retained in the churn model. Their influence on customer retention strategies should also be explored further.

**Encode Categorical Features**

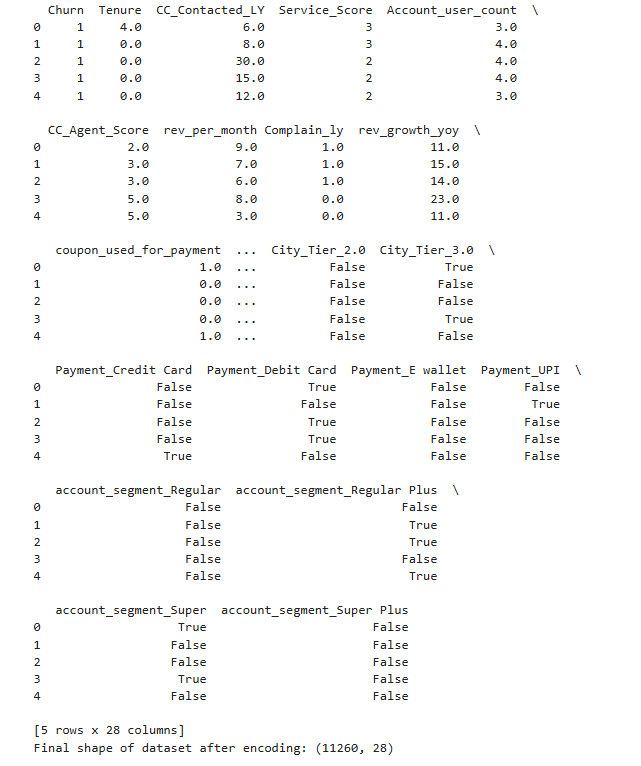
**Feature Encoding (for Categorical Variables)**

Machine learning models need numerical input, so you'll convert categorical variables into numbers using encoding techniques.

**Common Encoding Techniques:**

**Label Encoding:** Good for ordinal variables (e.g., Service Score: Low, Medium, High).

**One-Hot Encoding:** Ideal for nominal variables (e.g., Gender, City\_Tier, Payment Type).

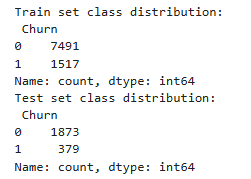


**Figure 28 Dataset After Feature Engineering**

### **If so, what can be done?**

**Balance the Target Variable**

* Feature Selection (Keep Important Features) From ANOVA, we found that some numerical variables significantly impact churn, while others do not.
* Keep the (p < 0.05, significant difference in means)
* Combine with Categorical Features Now, include important categorical features from Chi-Square Testing (previous step).
* Keep the Categorical Features (p < 0.05, significant association with churn).
* Since your dataset has a 5:1 imbalance, we'll use SMOTE to generate synthetic churn samples and Class Weights to further adjust model learning.



**Figure 29 train\_test\_split**

Class distribution after SMOTE: Counter({'0': 7491, '1': 3745})

## **Any business insights using clustering**

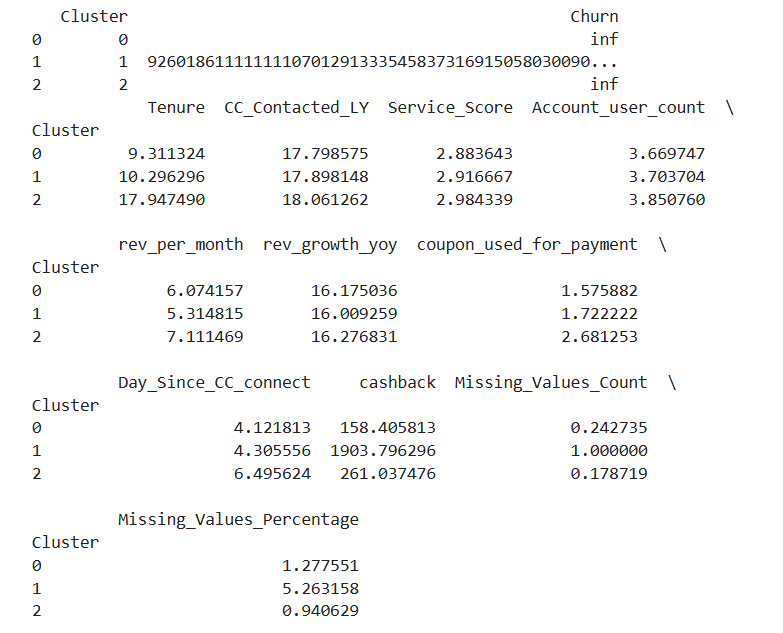
Since we have both categorical and numerical features, the best clustering method to use is K-Prototypes (which handles mixed data types).

Why Use Clustering?

Clustering can help segment customers based on their characteristics and behaviors, providing insights into:

* Different customer profiles (e.g., high-risk churners vs. loyal customers)
* Identifying customer groups that are more likely to churn
* Targeted marketing strategies based on cluster-specific behaviors

**Analyze cluster distributions**



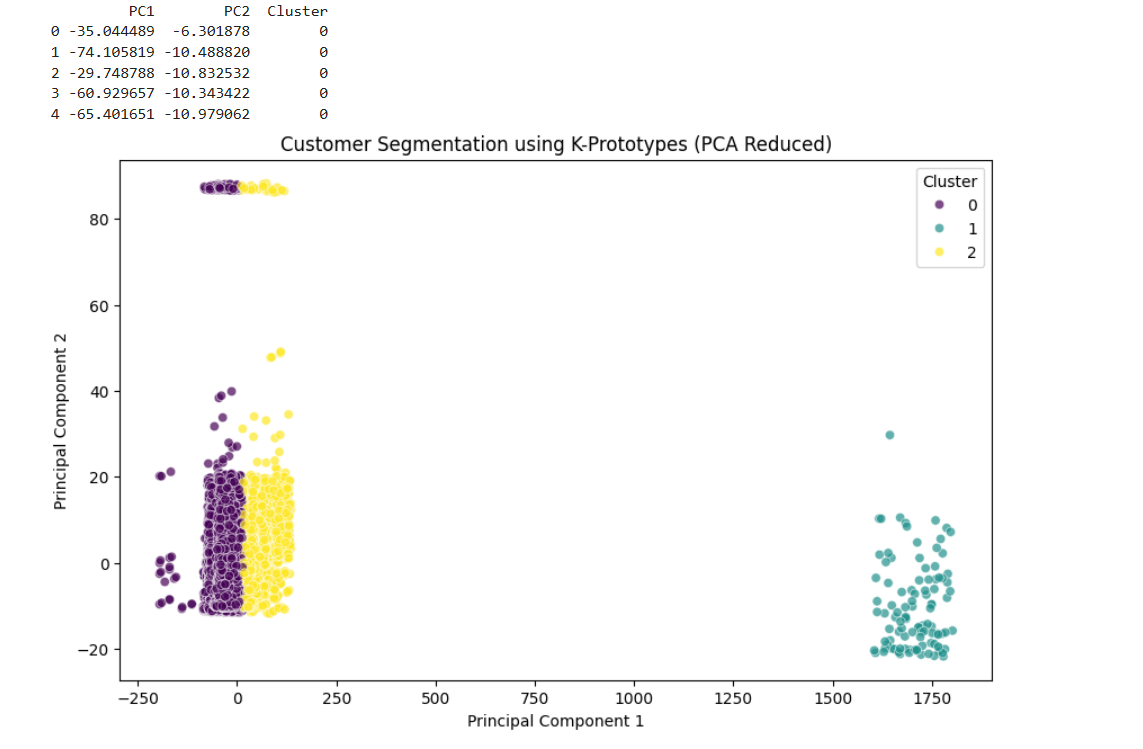
**Figure 30 Analyze cluster distributions**

**Key Insights:**

**Cluster Summary – Customer Segmentation Insights**

* **Cluster 0: New & Low-Engagement Users**
  + Short tenure, low coupon/cashback usage, moderate revenue.
  + **Action:** Focus on onboarding and engagement to reduce early churn.
* **Cluster 1: Incentive-Driven Users with Data Gaps**
  + Low revenue, very high cashback, highest missing data.
  + **Action:** Monitor for potential misuse of offers; review data quality.
* **Cluster 2: Loyal & High-Value Users**
  + Longest tenure, highest revenue and service score, frequent coupon use.
  + **Action:** Prioritize for loyalty programs and premium service offerings.

**Visualize Clusters using PCA (2D Scatter Plot)**



**Figure 31 Visualize Clusters using PCA (2D Scatter Plot)**

**Cluster Overview**

**Cluster 0 (Purple)**

* Highly concentrated around the origin of PCA space.
* Likely represents **standard/average customers** with no extreme behavior.
* Accounts for a **significant proportion** of the customer base.

**Cluster 1 (Cyan)**

* Separated **far along the first principal component (PC1)** axis.
* Indicates customers with **distinct behavior**, possibly:
  + High revenue or usage patterns.
  + High tenure or engagement.
* May represent **premium or high-value customers**.

**Cluster 2 (Yellow)**

* Overlaps somewhat with Cluster 0 but with **different distribution in PC2**.
* Suggests behavioral differences that may be driven by:
  + Use of coupons/cashbacks,
  + More frequent customer care interaction,
  + Or variations in churn behavior.
* Possibly a **cost-sensitive or engagement-sensitive** group.

# **Feature Scaling**

**1. Data Inspection and Cleaning**

* **Column Type Check**: Initially, the data was examined to identify non-numeric (object) columns using X.dtypes.
* **Binary Encoding**:
  + Columns containing only **'True'/'False'** or **'Yes'/'No'** were identified.
  + These were converted to numeric binary values (0/1) using mapping:
    - 'False'/'No' → 0
    - 'True'/'Yes' → 1

**2. One-Hot Encoding for Categorical Variables**

* **Identification**: Categorical columns with **more than two unique values** were selected for encoding.
* **OneHotEncoder**:
  + Used OneHotEncoder(drop='first') from scikit-learn to prevent the **dummy variable trap** (multicollinearity).
  + Transformed variables were:
    - Encoded into binary columns
    - Original categorical columns were dropped
    - New encoded columns were concatenated to the dataset

**3. Feature Scaling**

* **StandardScaler**:
  + Applied to all columns in the processed dataset.
  + Transformed data (X\_scaled) has a **mean of 0** and **standard deviation of 1**, which is essential for algorithms sensitive to feature scales (e.g., SVM, KNN).

**4. Train-Test Split**

* **Stratified Sampling**:
  + The dataset was split into training and testing sets (80-20 ratio) using train\_test\_split.
  + stratify=y ensured that the **class distribution** was preserved in both sets.
  + Two versions of splits were created:
    - **Original features**: X\_train, X\_test
    - **Scaled features**: X\_train\_scaled, X\_test\_scaled

**5. Handling Class Imbalance with SMOTE**

* **Synthetic Minority Oversampling Technique (SMOTE)**:
  + SMOTE was applied to balance the training data by **generating synthetic samples** for the minority class.
  + This was done for both:
    - **Original (unscaled) features**: X\_train\_smote, y\_train\_smote
    - **Scaled features**: X\_train\_scaled\_smote, y\_train\_scaled\_smote
  + This step ensures that classifiers are not biased toward the majority class.
* **Results**:
  + After applying SMOTE, class distribution in the target variable (y\_train\_smote) became **balanced**, as confirmed by value\_counts().

# **Model building**

To address the churn classification problem, we followed a structured approach to model building, focusing both on predictive performance and interpretability for business decision-making.

## **1. Model Selection Rationale**

We began with **Logistic Regression** as a baseline due to its simplicity and interpretability. It provided initial insights into how features such as tenure, complaints, and account user count influence the churn probability.

To improve predictive accuracy and capture non-linear relationships, we then tested:

* **Random Forest** – for its ability to model complex feature interactions and its robustness to overfitting.
* **XGBoost** – due to its superior performance in handling imbalanced datasets and built-in regularization.

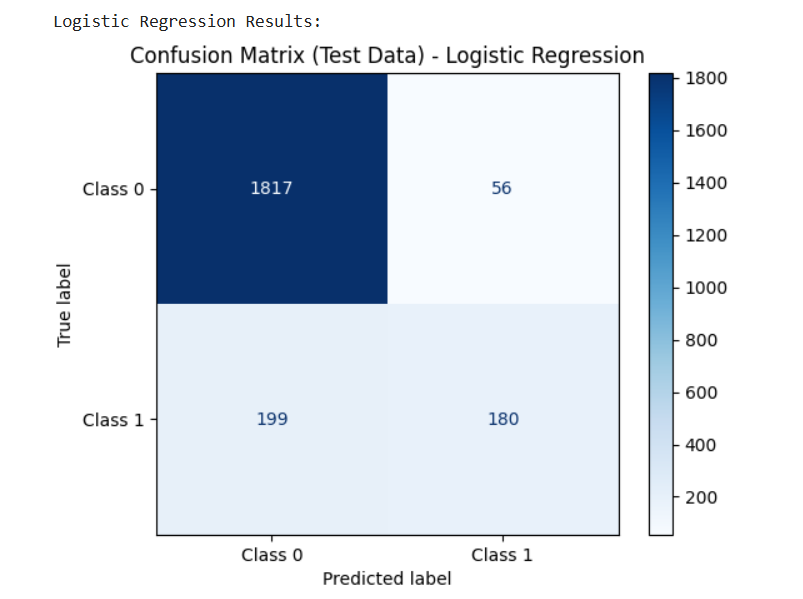
|  |  |  |
| --- | --- | --- |
| **Model** | **Purpose** | **Explanation** |
| **Logistic Regression** | Predictive | Predicts the probability of a binary outcome (e.g., churn vs no churn) based on input variables. Simple, interpretable, and good for baseline models. |
| **Naïve Bayes** | Predictive | Based on Bayes' Theorem with strong independence assumptions between features. Fast and effective for high-dimensional categorical data (e.g., text classification). |
| **Support Vector Machine (SVM)** | Predictive | Classifies data by finding the optimal hyperplane. Effective in high-dimensional spaces, often used for binary classification problems. |
| **Decision Tree** | Predictive & Descriptive | Predictive for classification, but also descriptive because it shows clear decision rules. Helpful for interpreting how input variables lead to specific outcomes. |
| **K-Nearest Neighbors (KNN)** | Predictive | Classifies a sample based on the majority label of its K closest neighbors. Simple and intuitive, often used for pattern recognition. |
| **Random Forest** | Predictive | Ensemble of decision trees. Boosts prediction accuracy and handles overfitting better than individual trees. Good for feature importance too. |
| **AdaBoost** | Predictive | Adaptive boosting method that builds a strong classifier by combining multiple weak classifiers. Focuses more on difficult-to-classify samples. |
| **Gradient Boosting** | Predictive | An advanced boosting technique that sequentially improves weak models by minimizing error. Great for handling complex relationships. |

**Table 7 Model Selection and their Explanation**

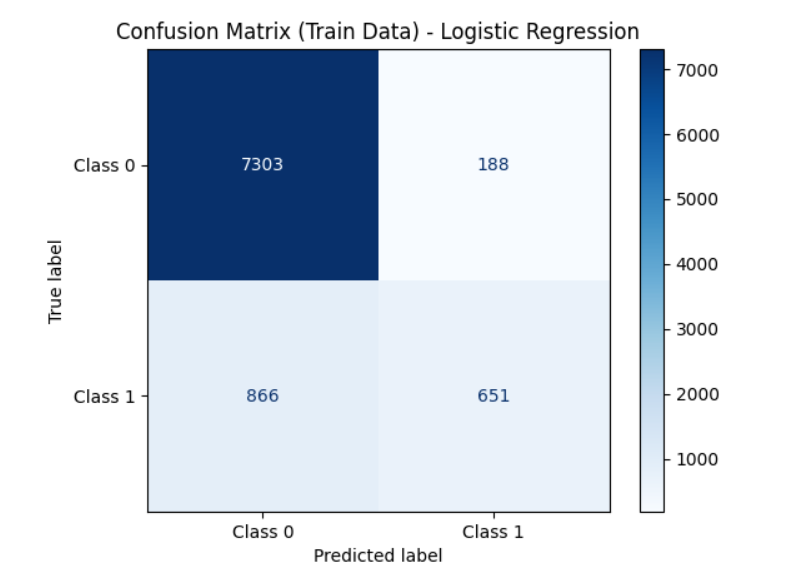
* All models were trained using the **scaled training data (X\_train\_scaled, y\_train\_scaled)**.
* Predictions were made on both **training** and **testing** datasets for performance evaluation.

### **Model Building - Original Dataset**

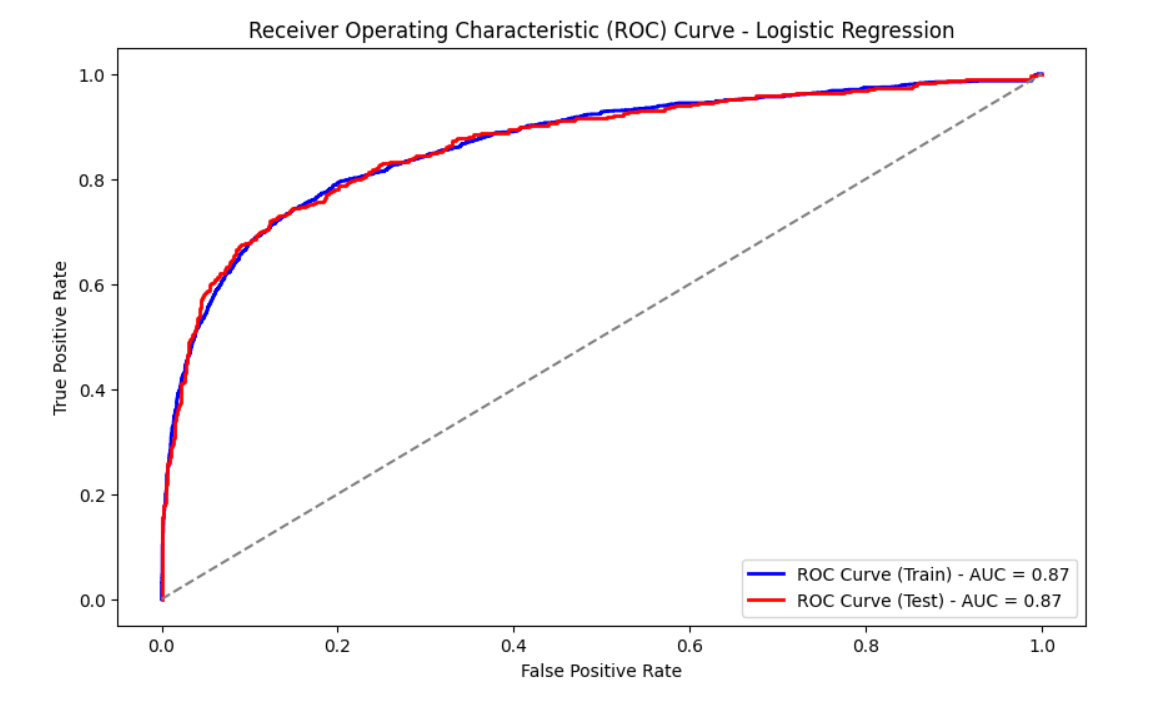
#### **Logistic Regression**



**Figure 32 Logistic Regression Test Data Original Dataset**

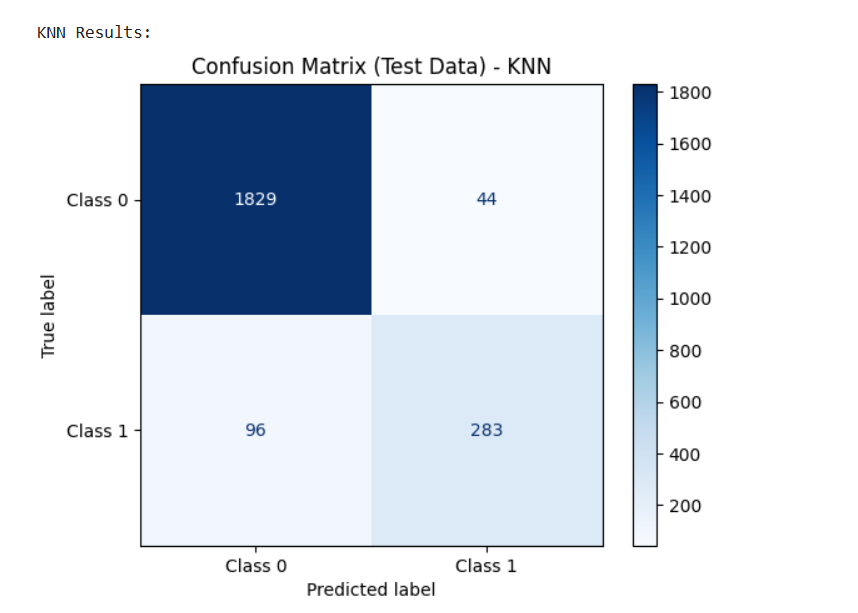


**Figure 33 Logistic Regression Train Data Original Dataset**

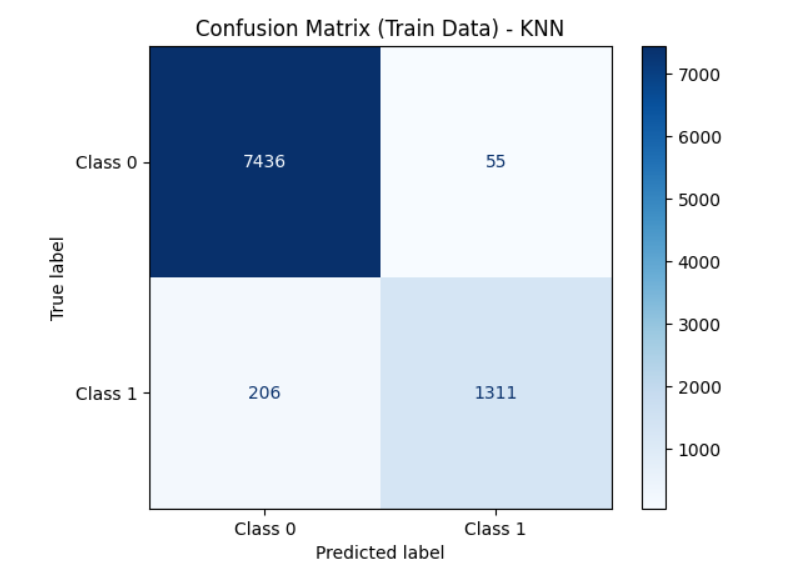


**Figure 34 Roc Logistic Regression Train and Test**

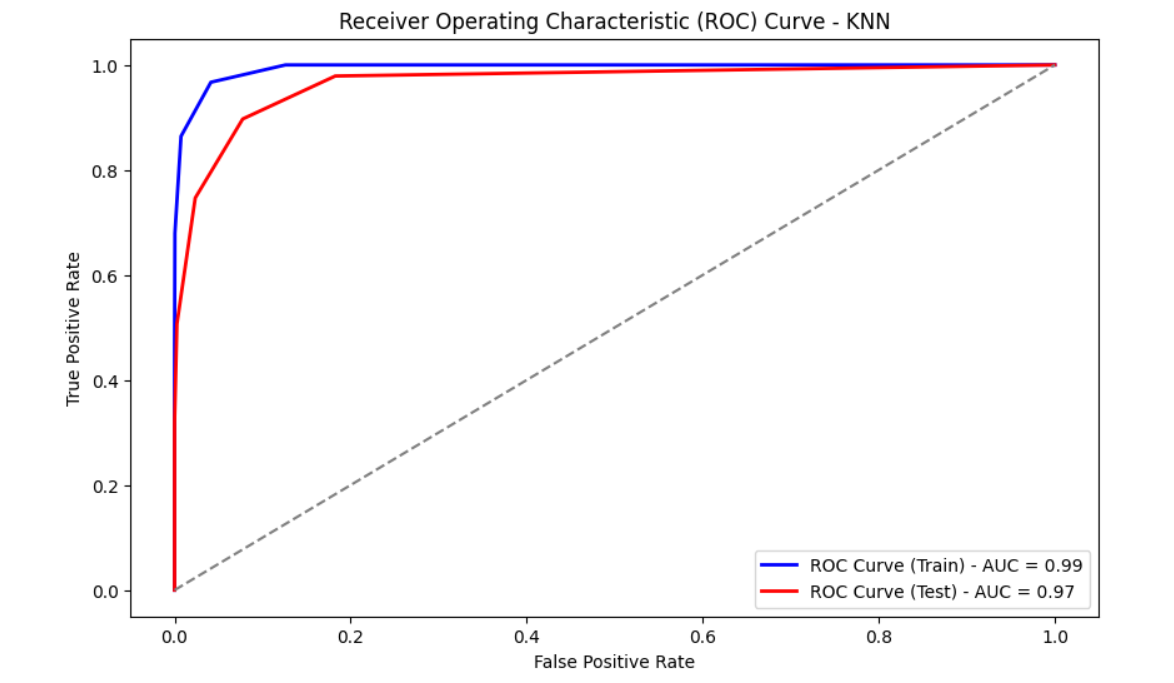
#### **KNN**



**Figure 35 KNN Test Data Original Dataset**

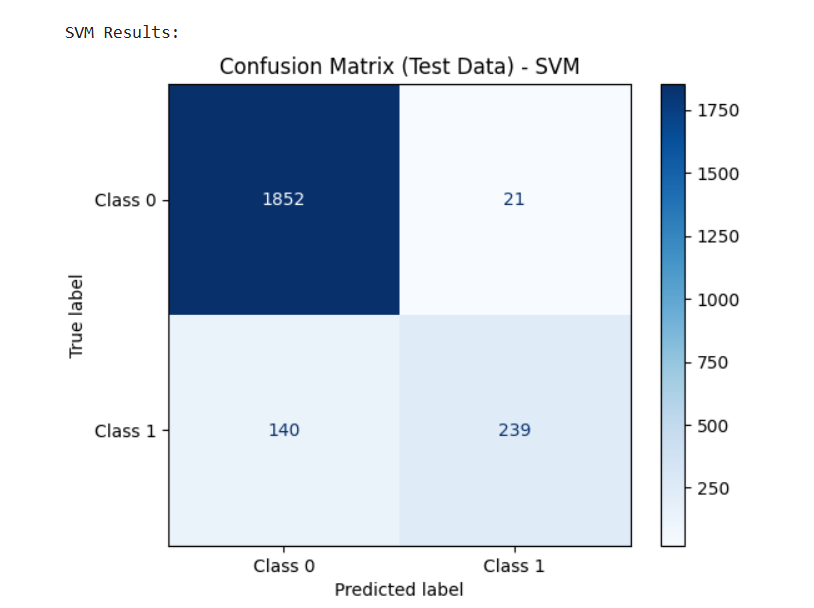


**Figure 36 KNN Train Data Original Dataset**

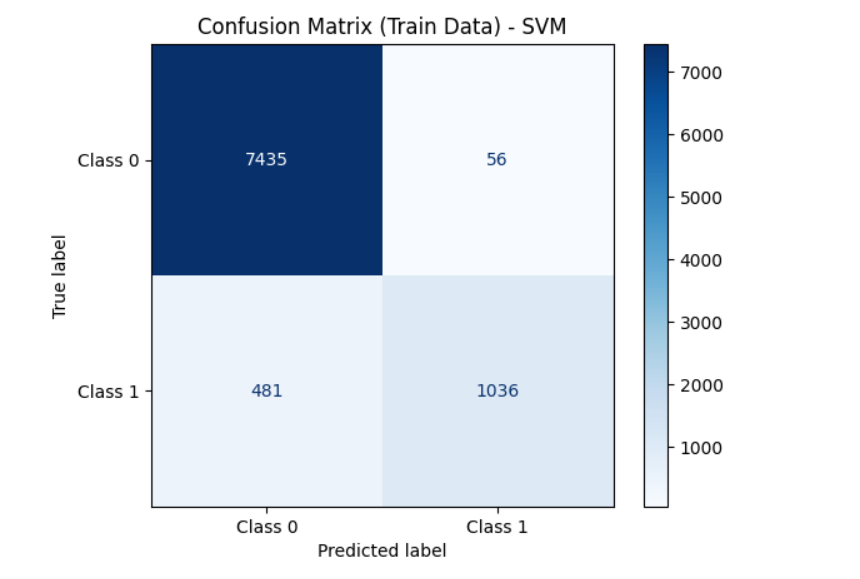


**Figure 37 ROC - KNN Train and Test**

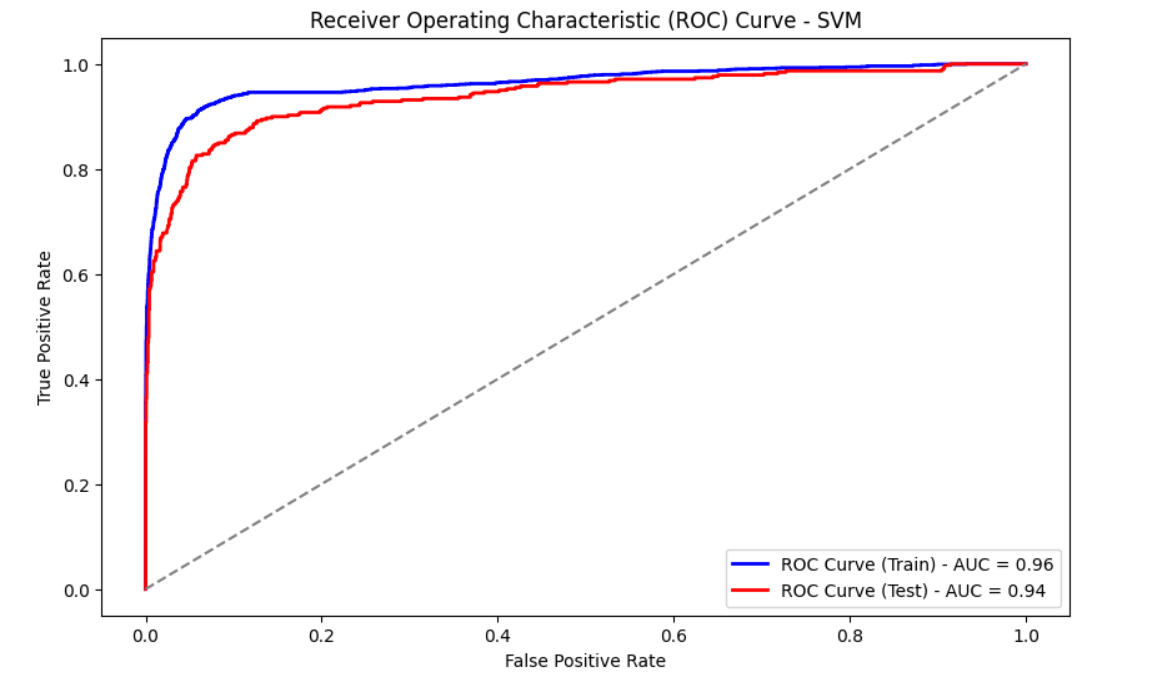
#### **SVM**



**Figure 38 SVM Test data on original Data**

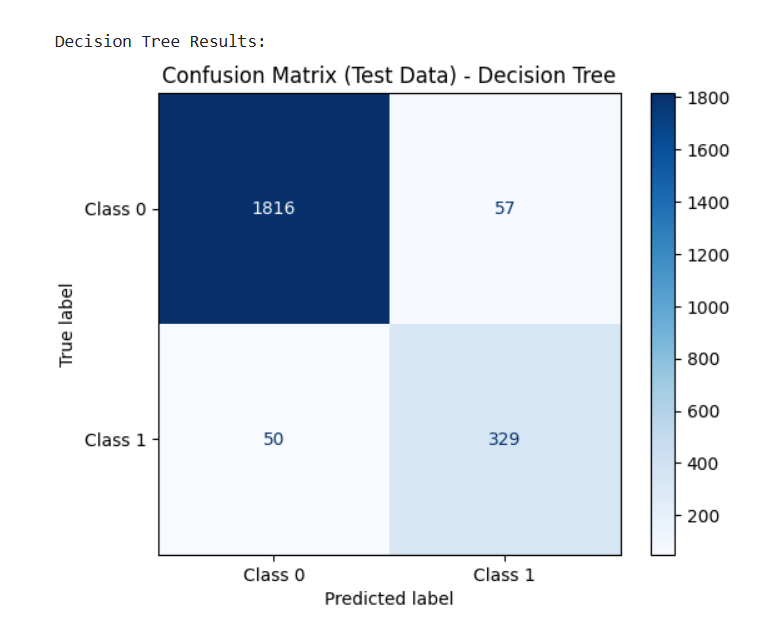


**Figure 39 SVM Train data on original Data**

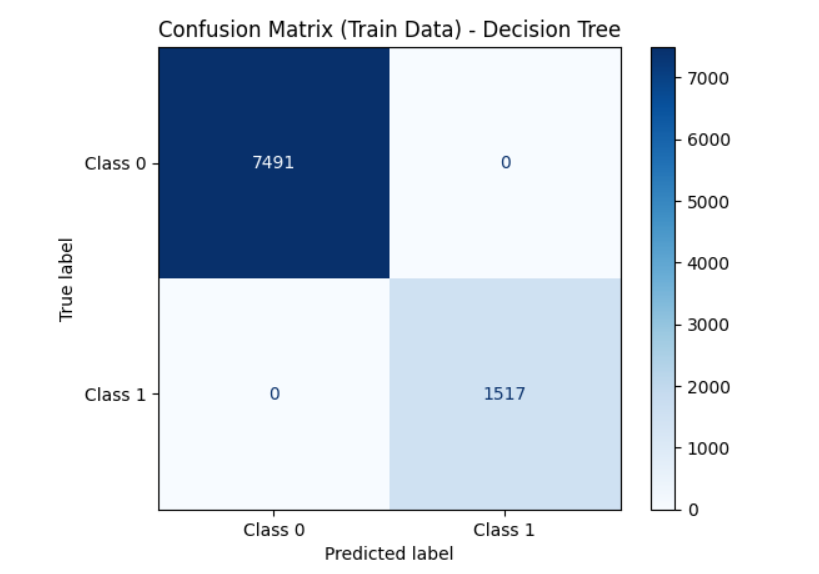


**Figure 40 ROC Curve SVM**

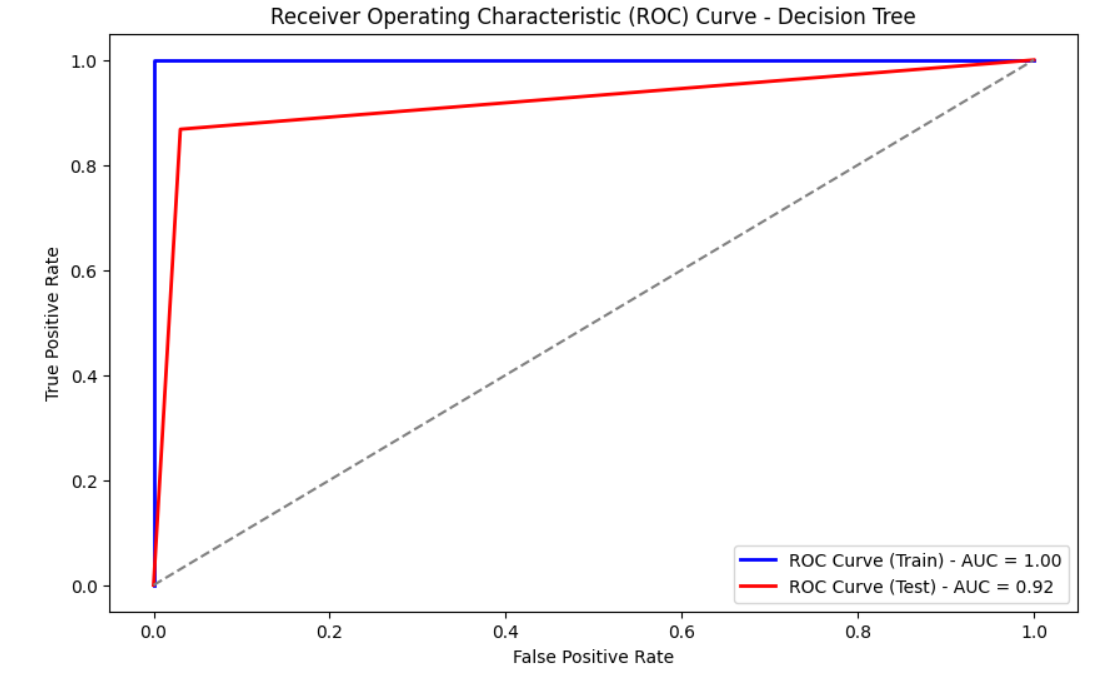
#### **Decision Tree**



**Figure 41 Decision Tree Test Data on Original Dataset**

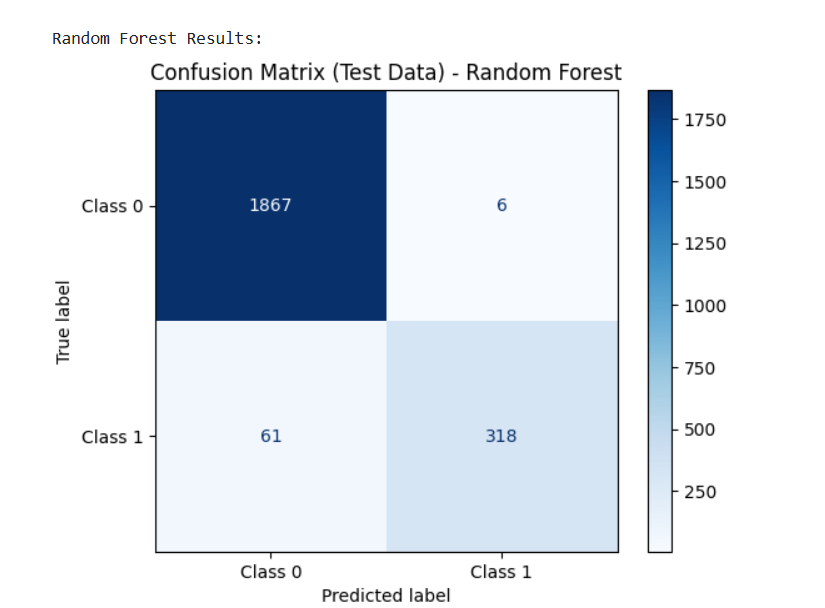


**Figure 42 Decision Tree Train Data on Original Dataset**

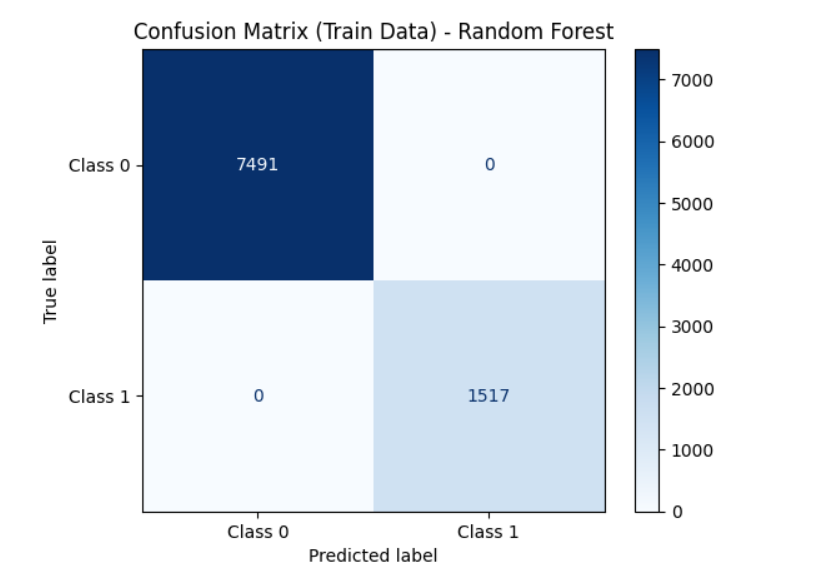


**Figure 43 ROC Curve Train and Test Dataset**

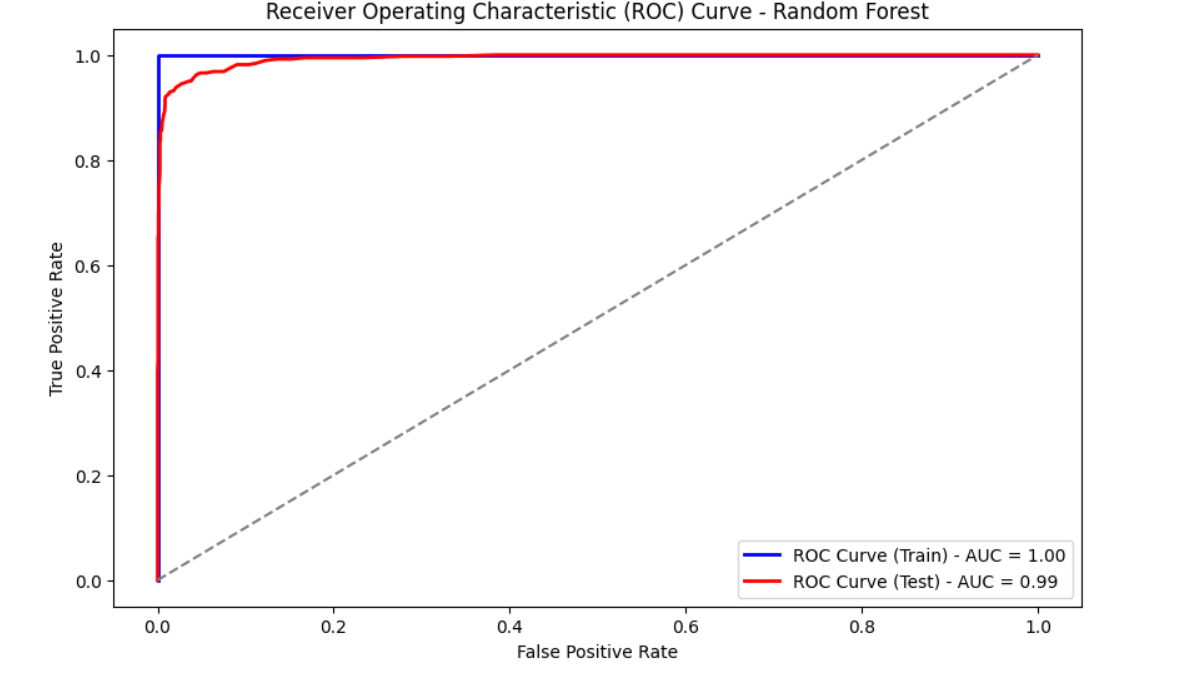
#### **Random Forest**



**Figure 44 Random Forest Test Data Original Dataset**

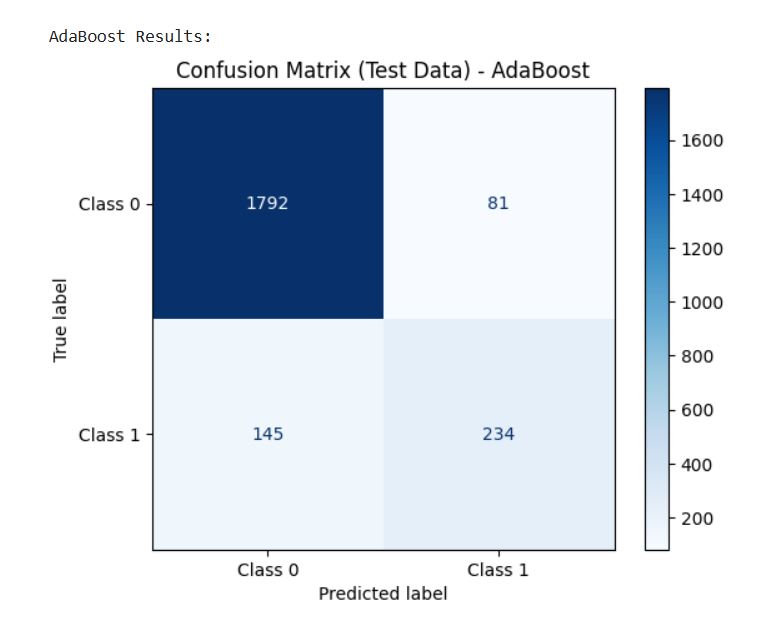


**Figure 45 Random Forest Train Data Original Dataset**

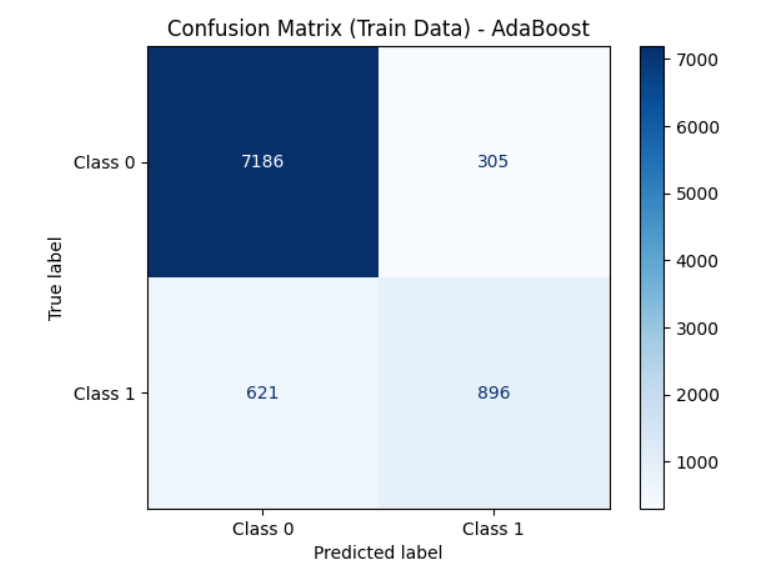


**Figure 46 ROC Curve Random Forest Train and Test Data**

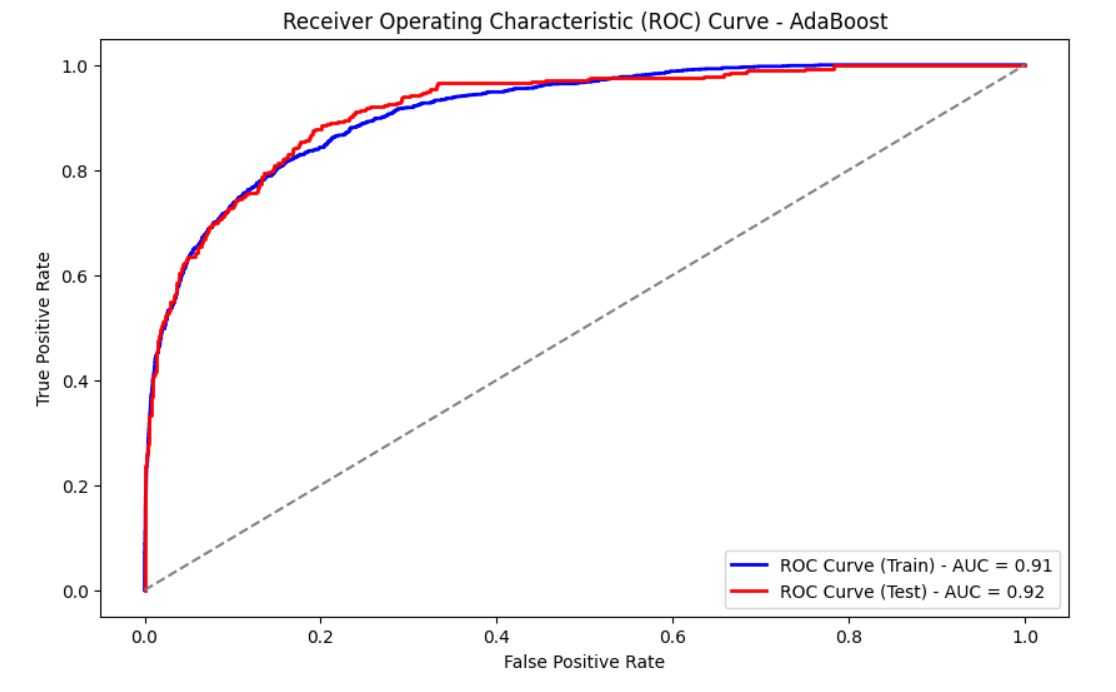
#### **AdaBoost**



**Figure 47 AdaBoost Test data for original dataset**

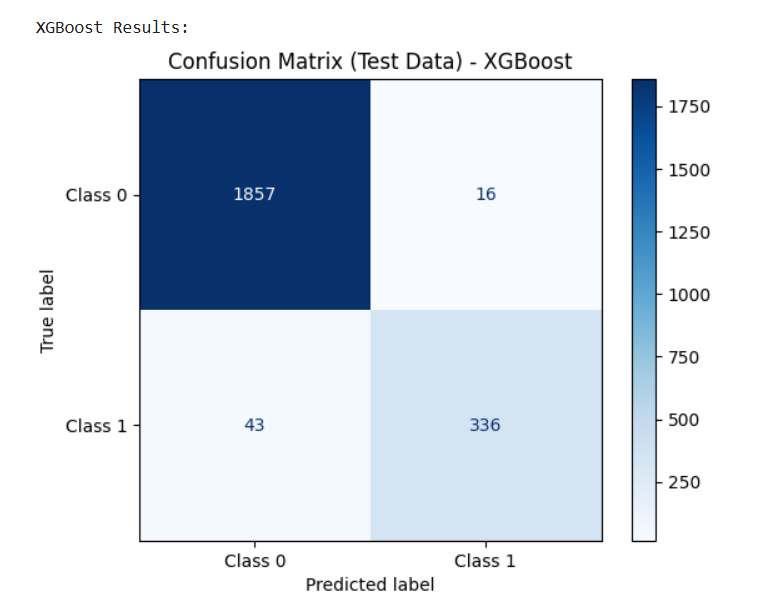


**Figure 48 Adaboost Train data on original dataset**

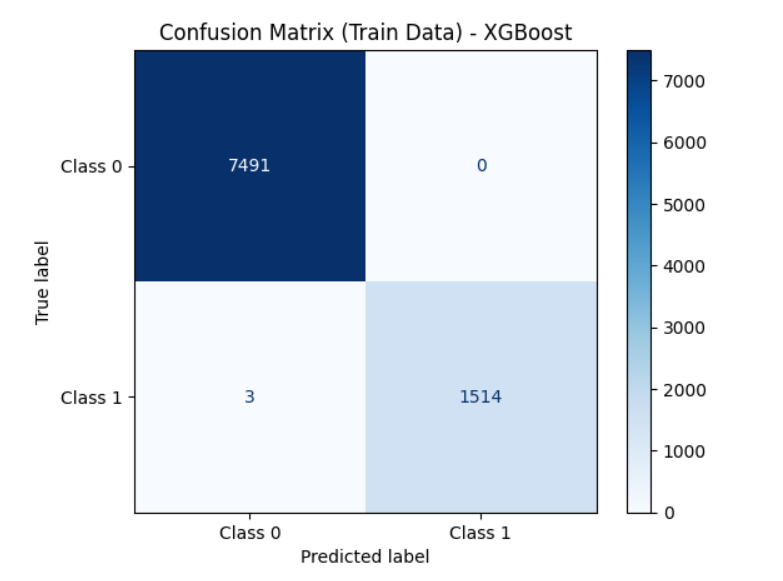


**Figure 49 ROC Curve Train and Test Data Adaboost**

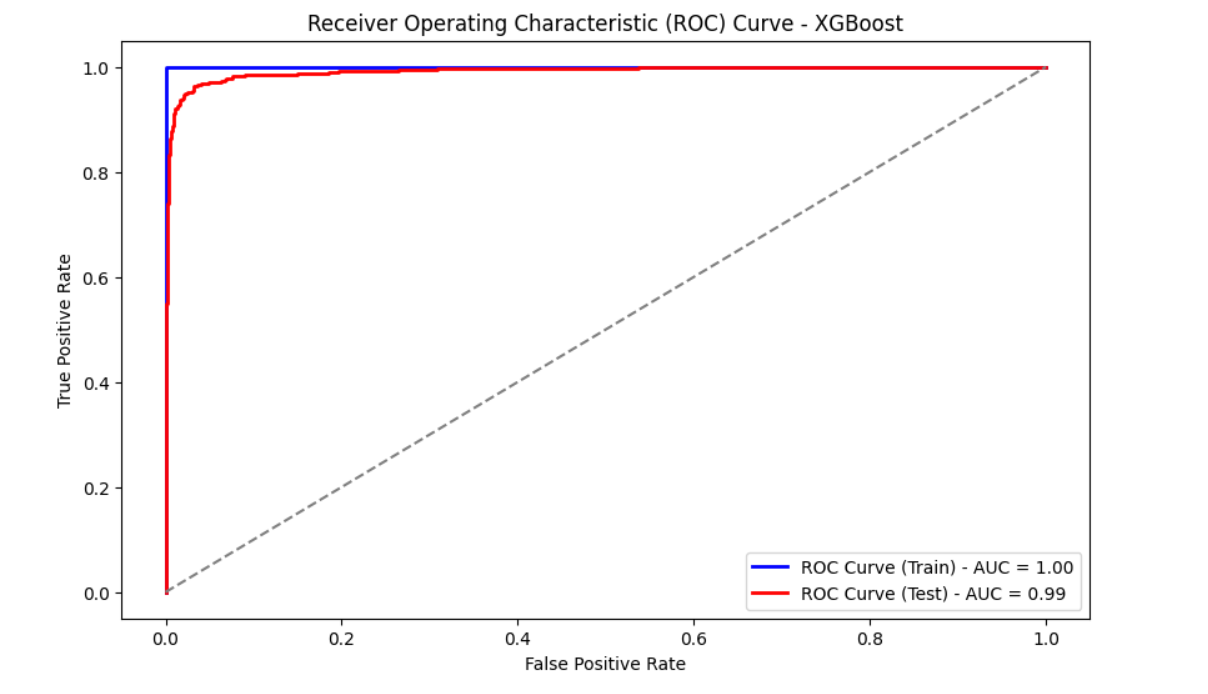
#### **XGBoost**



**Figure 50 XGBoost Test data on original dataset**

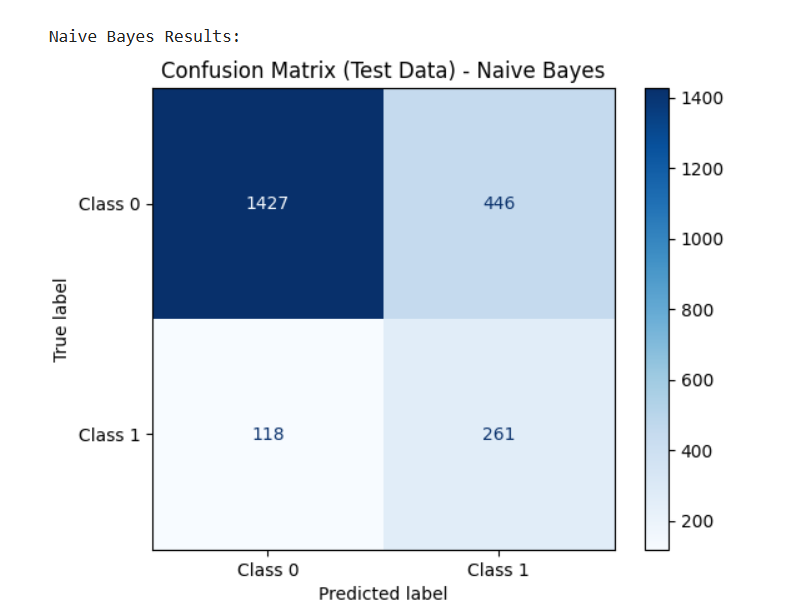


**Figure 51 XGBoost Train data on original dataset**

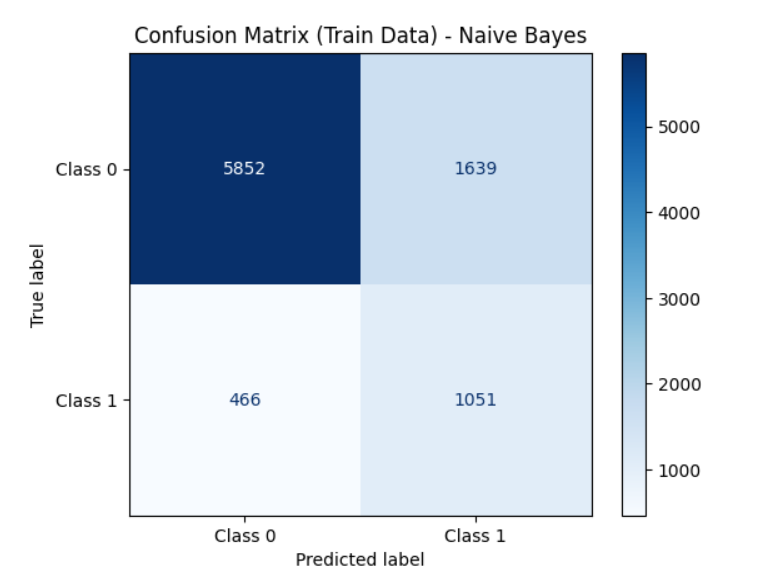


**Figure 52 ROC Curve XGBoost Train and Test Dataset**

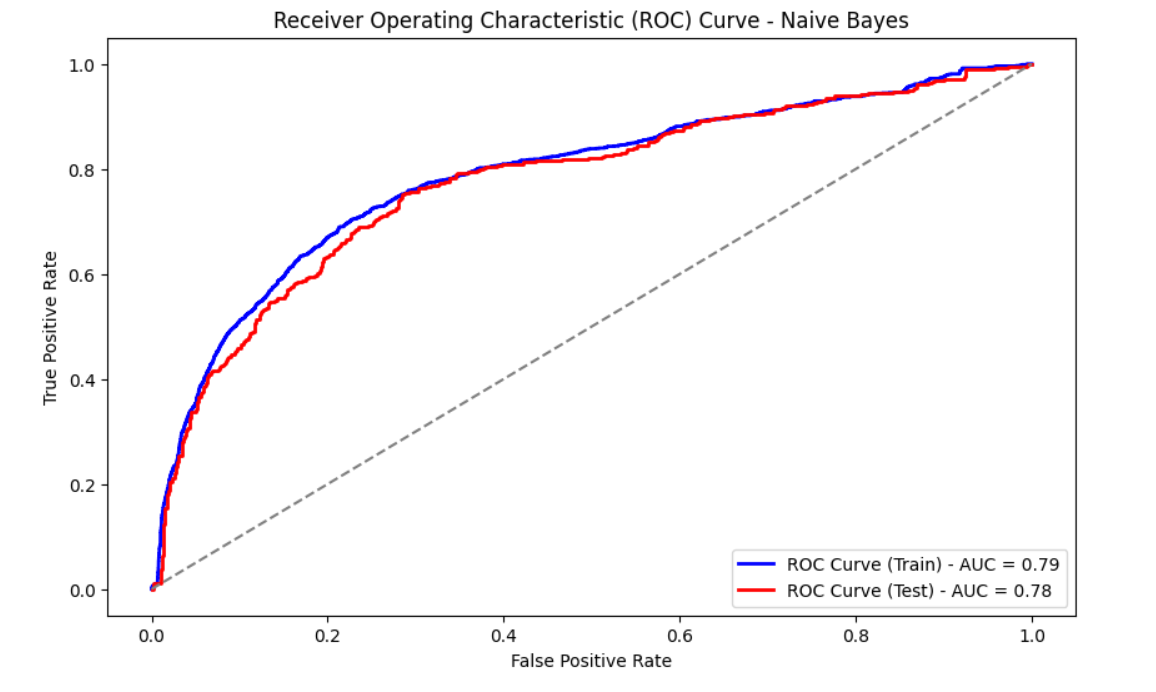
#### **Naïve Bayes**



**Figure 53 Naive Bayes Test Data on Original dataset**



**Figure 54 Naive Bayes Train Data on Original dataset**



**Figure 55 ROC Curve Naive Bayes Test and Train Data on Original dataset**

#### **Model Summary Original Dataset**



**Table 8 Model Summary Original Dataset**

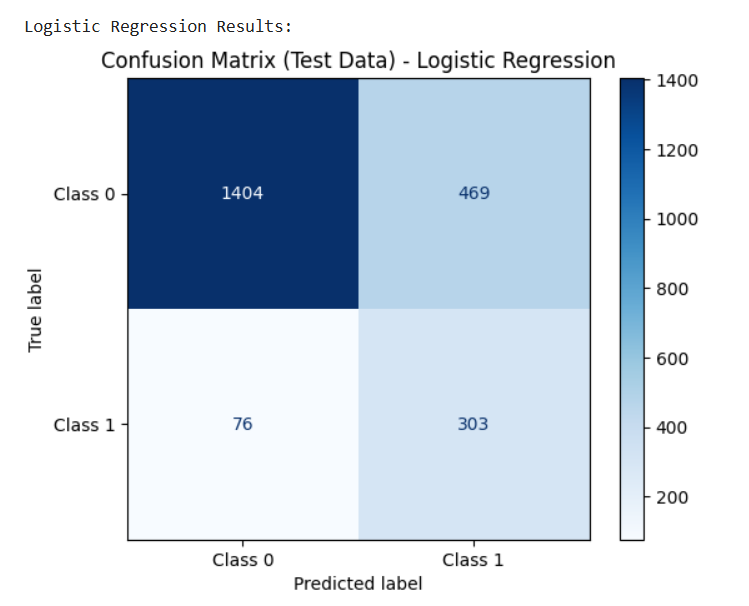
#### **Key Insights:**

|  |  |
| --- | --- |
| **Model** | **Use Case** |
| **XGBoost** | Best overall: high accuracy, balanced, robust |
| **Random Forest** | Close second; equally strong and interpretable |
| **Decision Tree** | High risk of overfitting despite good test scores |
| **KNN/SVM** | Good alternatives; use if tuning is easier |
| **Logistic/AdaBoost** | Simple, but lower-performing options |
|  |  |
| **Naive Bayes** | Not recommended for this task |

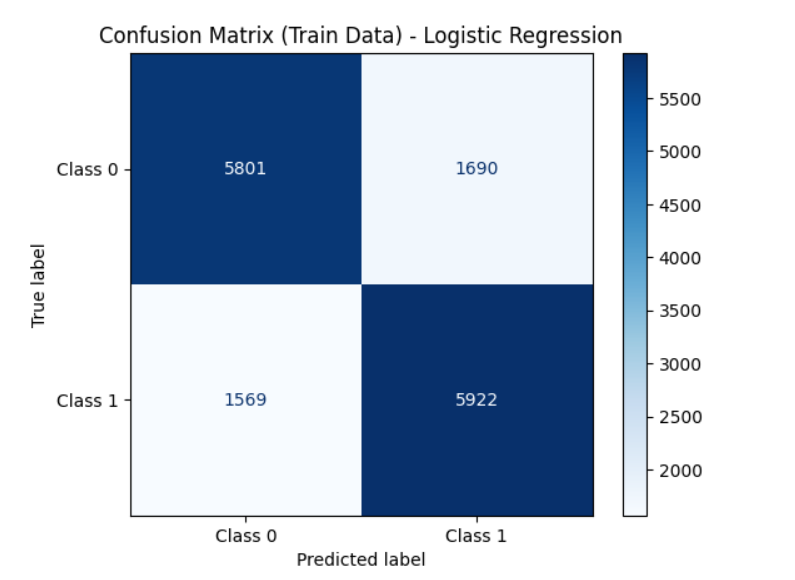
**Table 9 Key Insights**

### **Model Building – SMOTE Dataset**

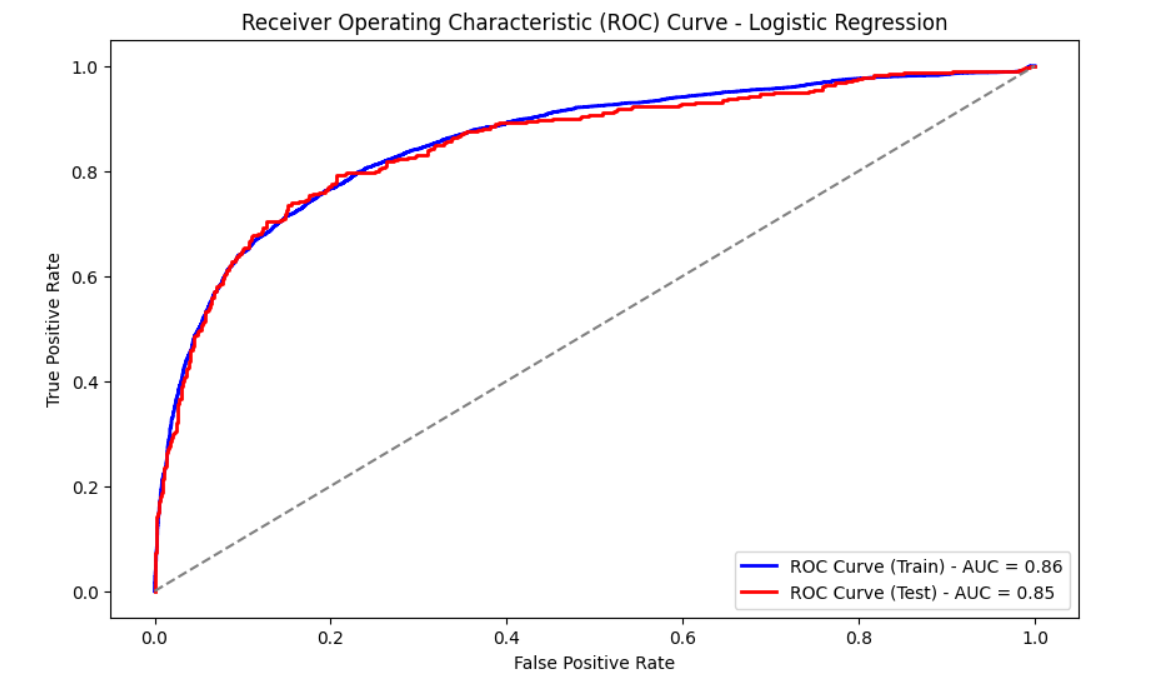
#### **Logistic Regression**



**Figure 56 Logistic Regression SMOTE Dataset Test**

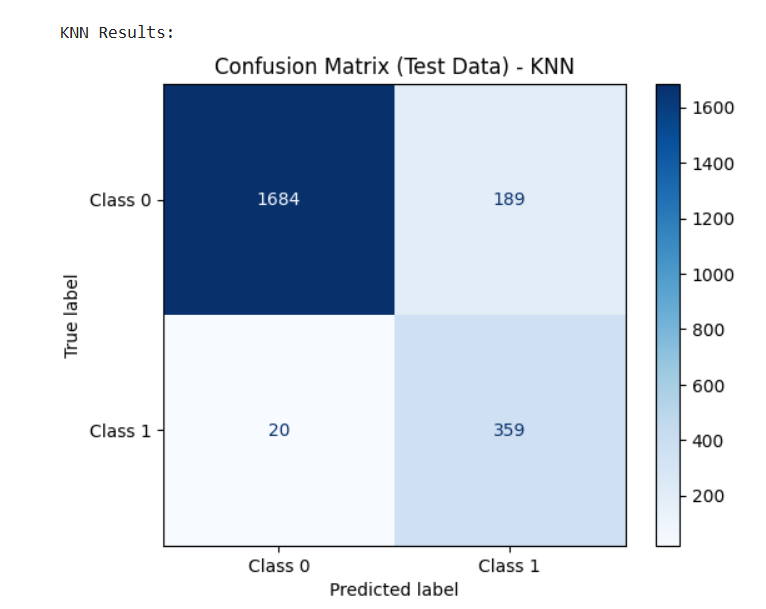


**Figure 57 Logistic Regression SMOTE Dataset Train Data**

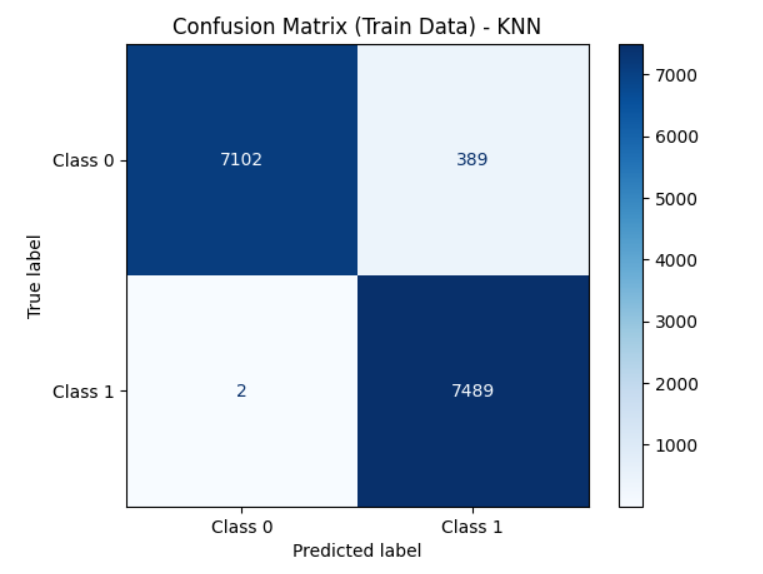


**Figure 58 ROC Curve SMOTE Dataset Logistic Regression**

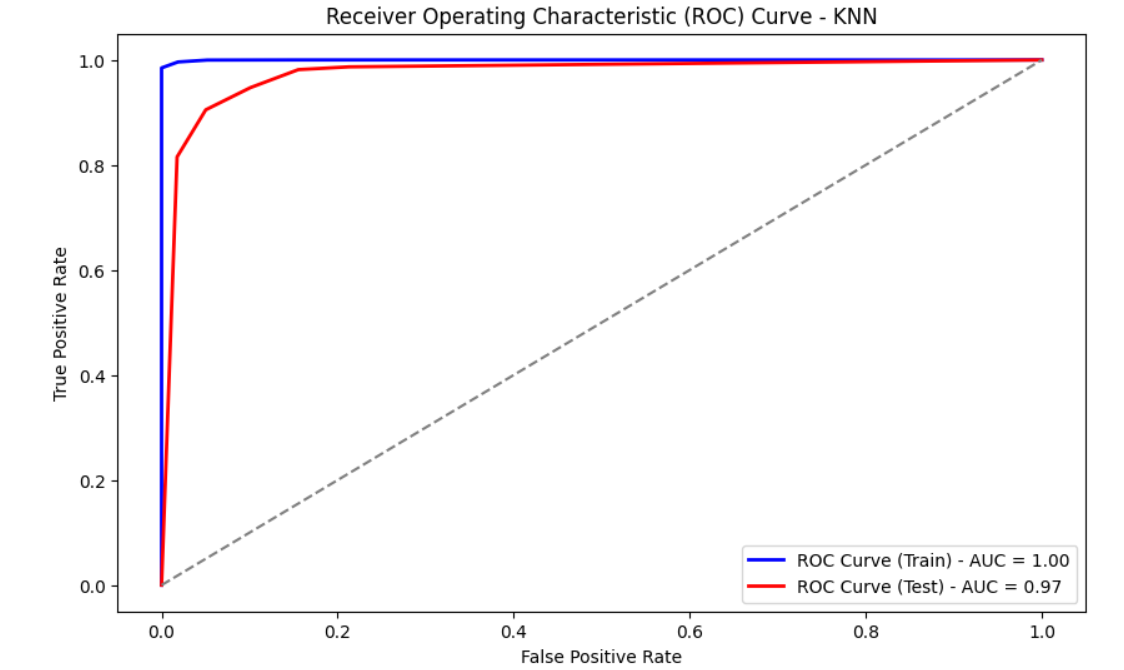
#### **KNN**

KNN 

**Figure 59 KNN SMOTE data Test Data**



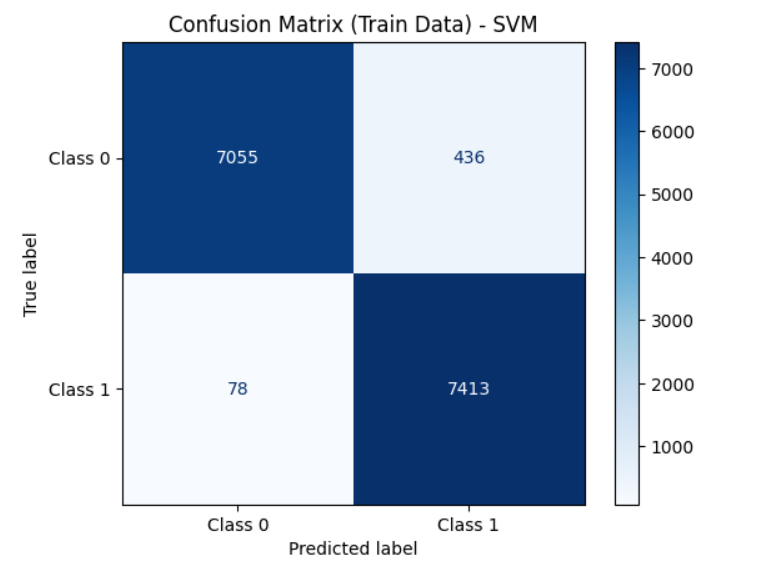
**Figure 60 KNN SMOTE TRAIN Data**



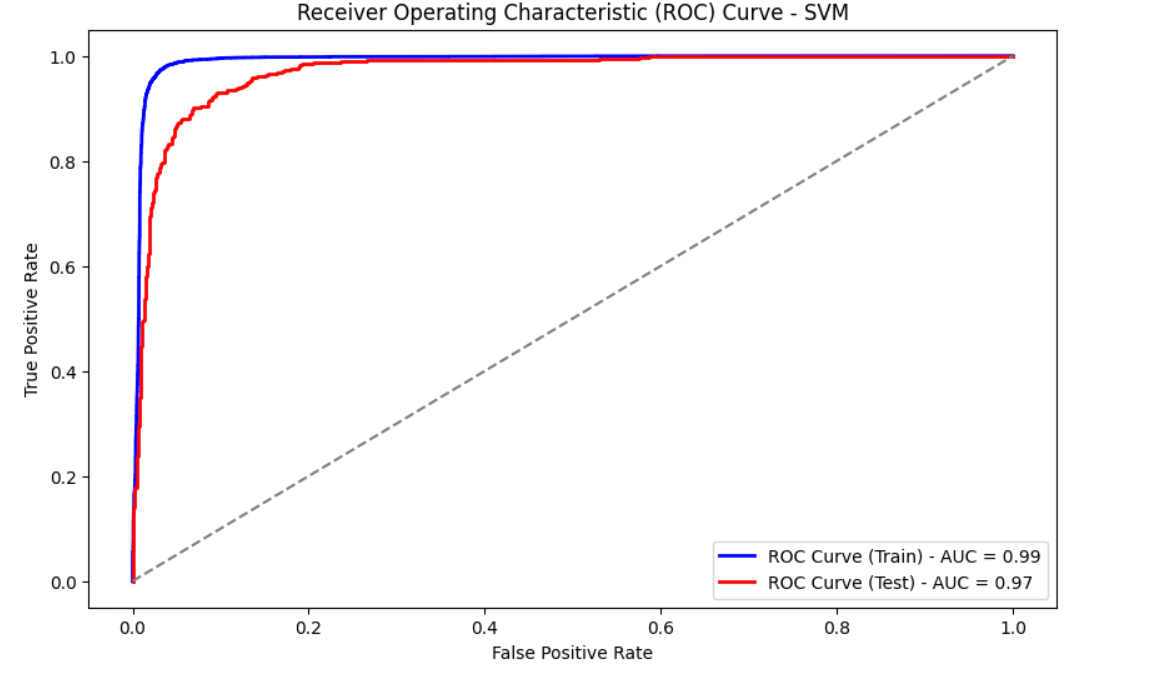
**Figure 61 KNN SMOTE Data ROC Curve**

#### **SVM**

**Figure 62 SVM SMOTE Test Data**



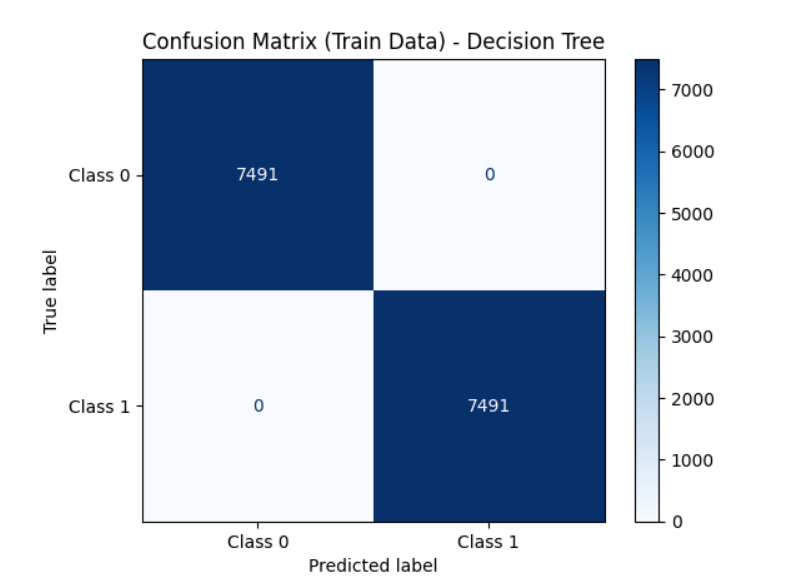
**Figure 63 SVM SMOTE Train Data**



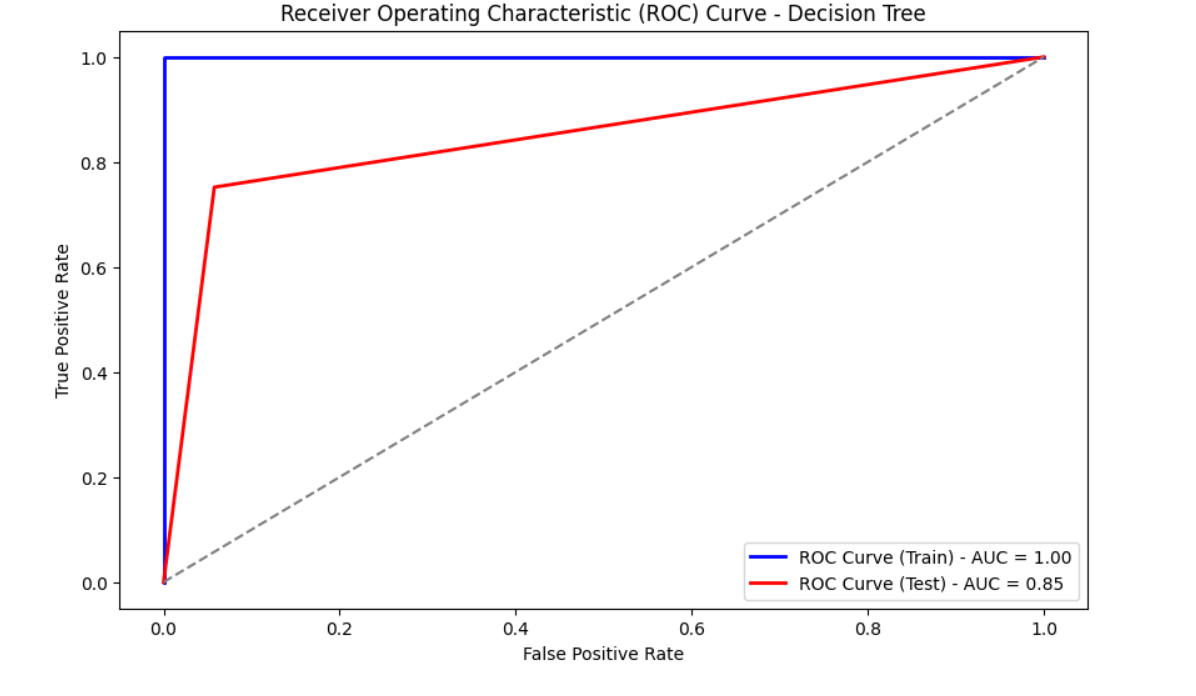
**Figure 64 SVM SMOTE ROC Curve**

#### **Decision Tree**

**Figure 65 Decision Tree SMOTE Test Data**

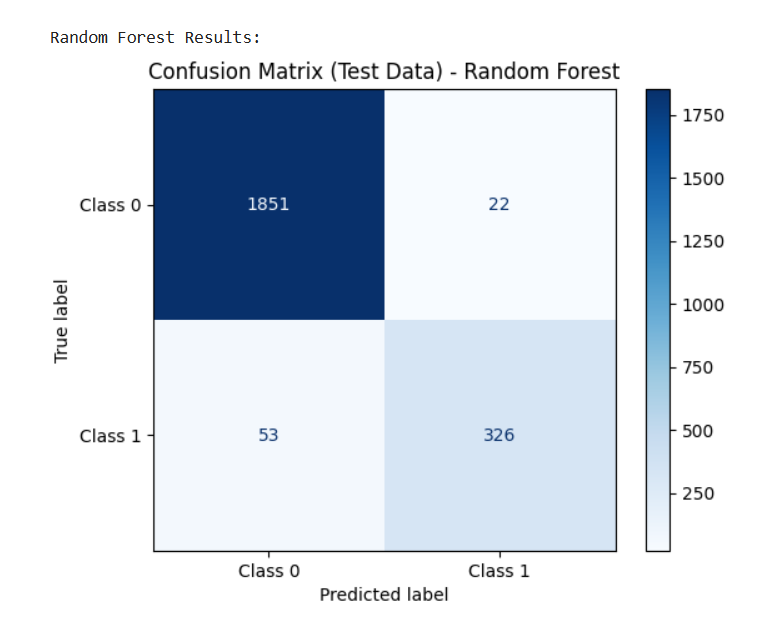


**Figure 66 Decision Tree SMOTE Train Data**

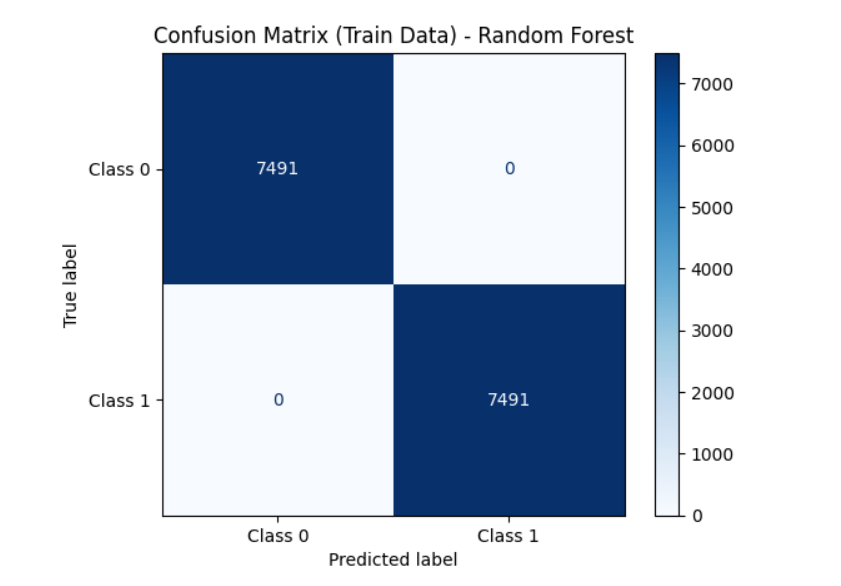


**Figure 67 Decision Tree Smote Data ROC Curve**

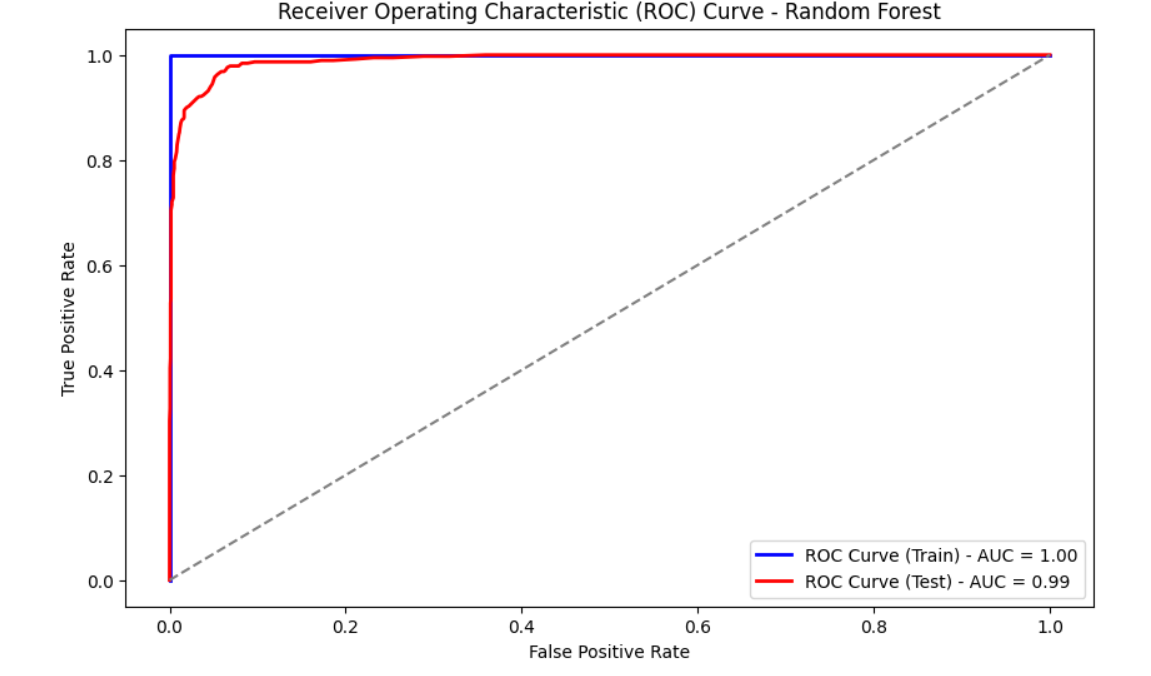
#### **Random Forest**



**Figure 68 Random Forest SMOTE Test Data**

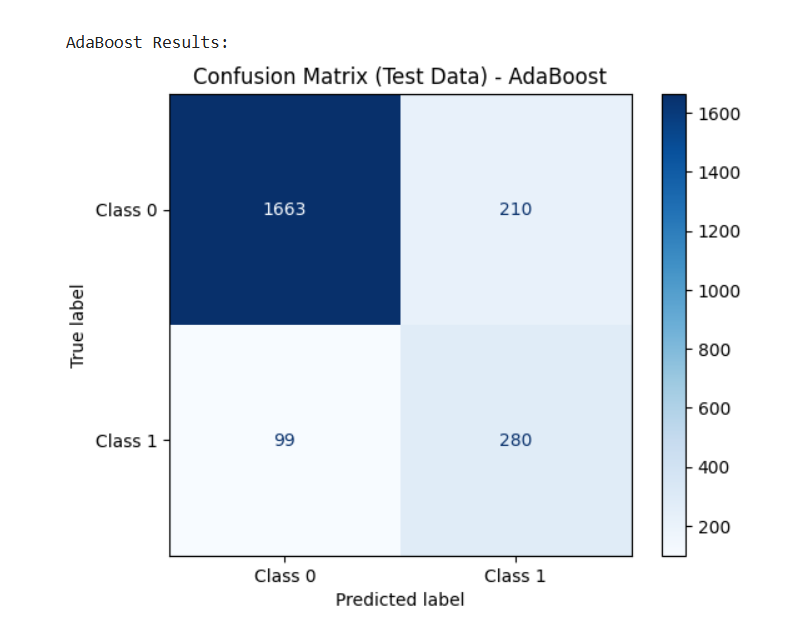


**Figure 69 Random Forest SMOTE Train Data**

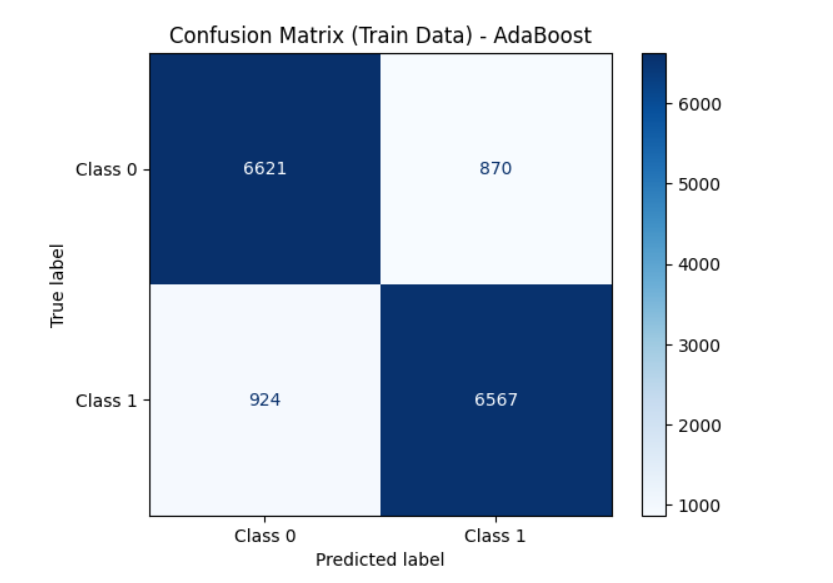


**Figure 70 Random Forest SMOTE ROC Curve**

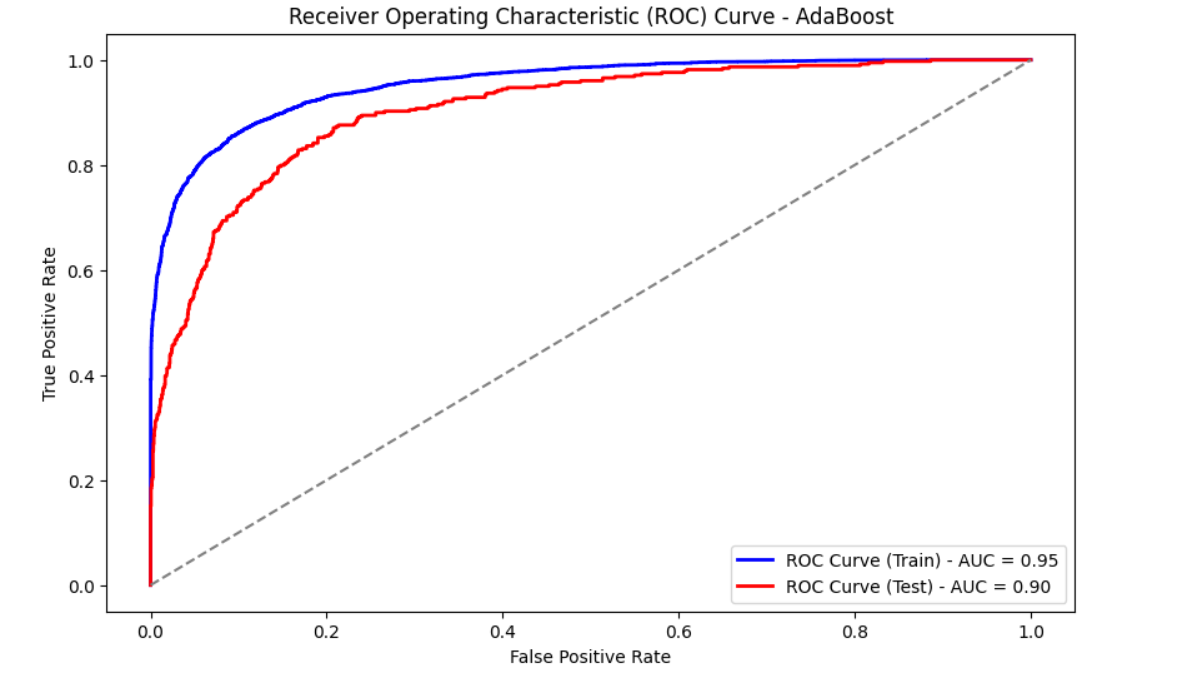
#### **AdaBoost**



**Figure 71 AdaBoost SMOTE Test Data**



**Figure 72 Adaboost SMOTE Train Data**

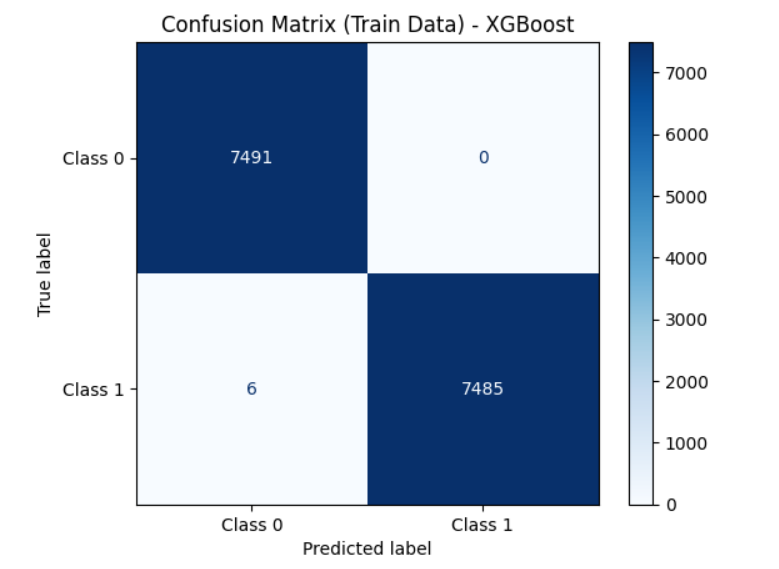


**Figure 73 Adaboost SMOTE ROC Curve**

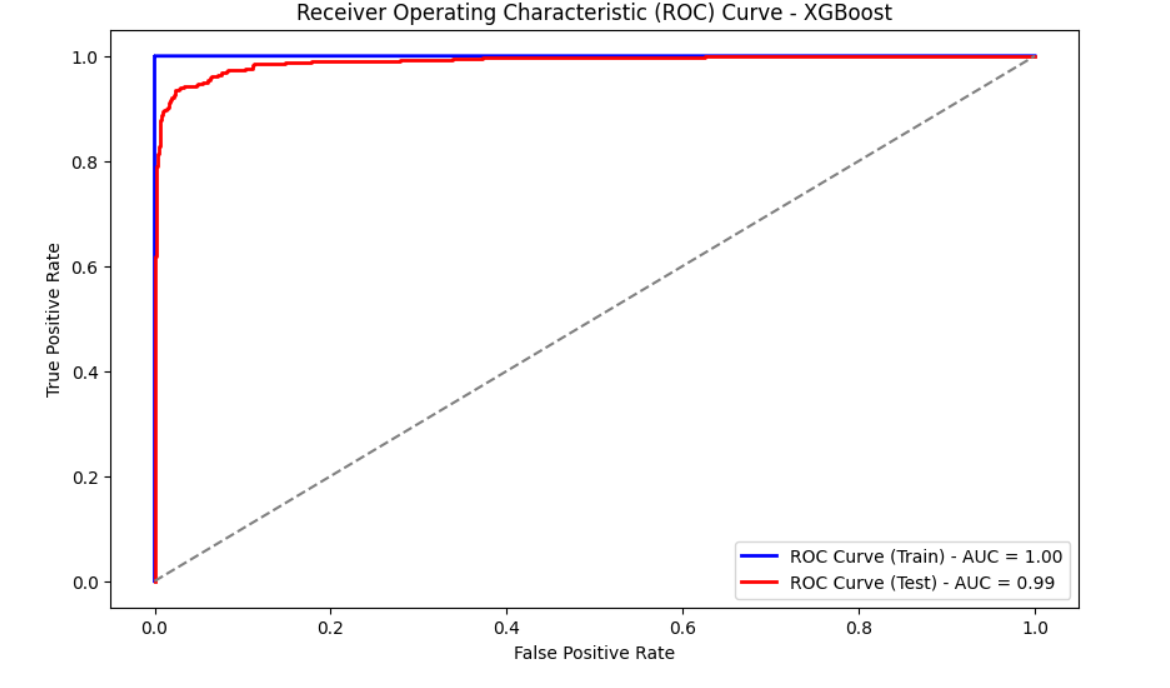
#### **XGBoost**



**Figure 74 XGBoost SMOTE Test Data**

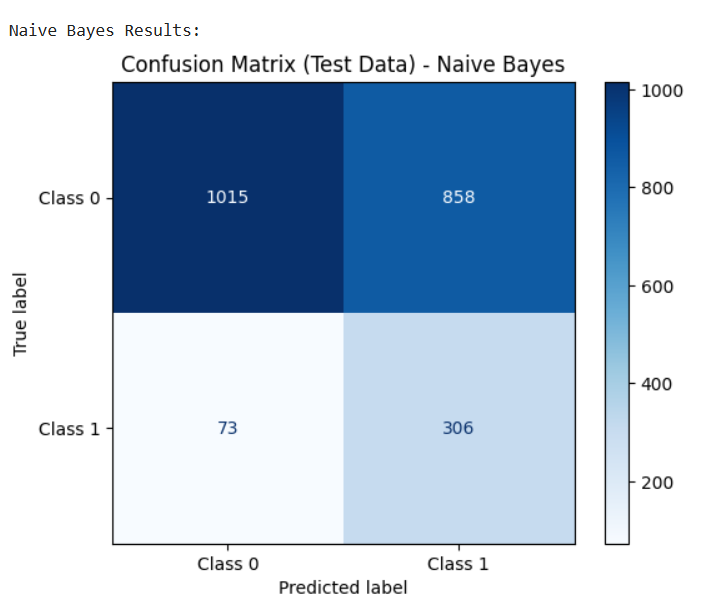


**Figure 75 XGBoost SMOTE Train Data**

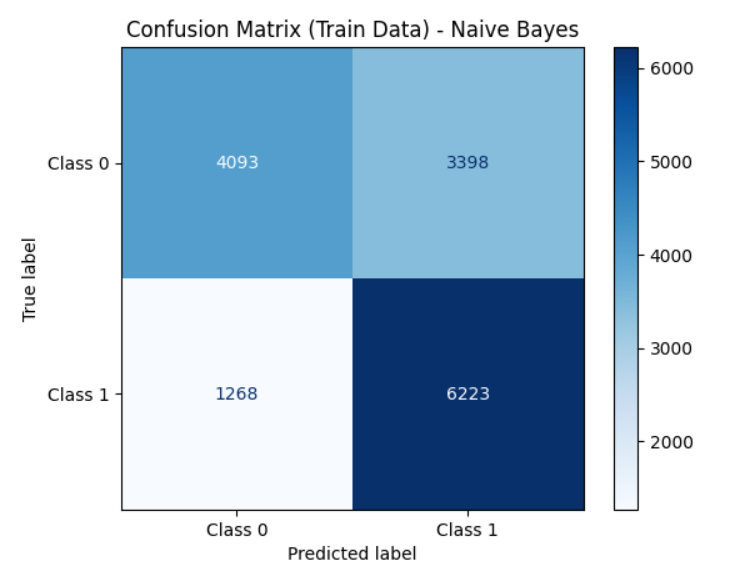


**Figure 76 XGBoost SMOTE ROC Curve**

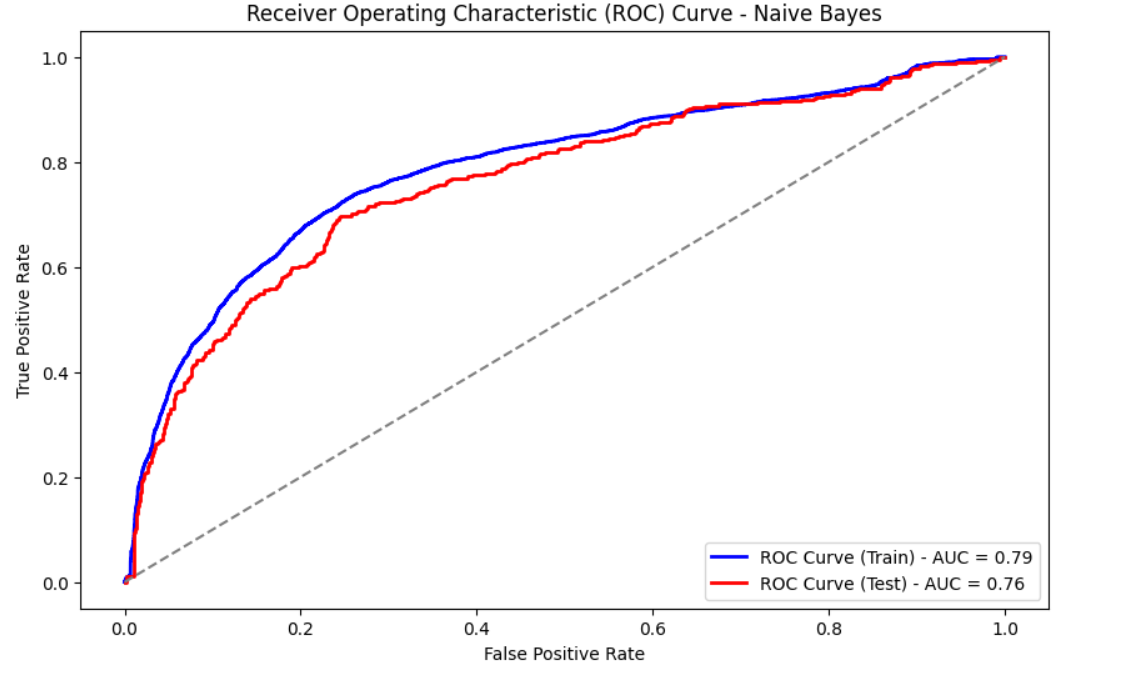
#### **Naïve Bayes**



**Figure 77 Naive Bayes AMOTE Test Data**



**Figure 78 Naive Bayes SMOTE Train Data**



**Figure 79 Naive Bayes SMOTE ROC Curve**

#### **Model Summary SMOTE Dataset**



**Table 10 Model Summary SMOTE Dataset**

#### **Key Insights:**

|  |  |
| --- | --- |
| **Model** | **Notes** |
| **XGBoost** | Highest test accuracy (97.38%), high precision & recall, and near-perfect train score. Very balanced. |
| **Random Forest** | Excellent generalization with 96.67% accuracy. High precision (0.94) and strong recall (0.86). |
| **SVM** | Very good at both precision and recall. High ROC-AUC (0.97). Slight overfitting but manageable. |
| **KNN** | High recall (0.95), good accuracy (90.7%). Train-test gap suggests overfitting risk. |
| **Decision Tree** | High accuracy but strong overfitting — train score is 100% everywhere. |
| **AdaBoost** | Moderate test accuracy (86%), decent recall, but precision is weak. Performs modestly. |
| **Logistic Regression** | Recall is good (0.80), but precision (0.39) is too low. Not reliable overall. |
| **Naive Bayes** | Very poor precision (0.26), weak accuracy (58.7%). Likely unsuitable for this task. |

**Table 11 Key Insights**

## **Model Comparison & Evaluation**

**Model Comparison on Original Data and SMOTE Data**



**Table 12 Model Comparison on Original Data and SMOTE Data**

**Key Insights:**



**Table 13 Key Insights**

**XGBoost** is ideal when:

* You need **high predictive power**
* You care deeply about **minimizing false negatives** (like churners or fraud cases)
* You want both **high precision** and **high recall**

## **3. Efforts to Improve Performance**

To optimize model performance, we undertook the following steps:

* **Hyperparameter Tuning**: Used **RandomSearchCV** for Random Forest and XGBoost to identify optimal tree depth, learning rate, and estimators.
* **Cross-validation**: Employed 5-fold cross-validation to ensure robustness across different data splits.
* **Class Imbalance Handling**:
  + Applied **SMOTE** (Synthetic Minority Over-sampling Technique) to balance the dataset.
  + Compared results with using scale\_pos\_weight in XGBoost.
* **Feature Engineering**:
  + Created binary indicators for recent complaints and days since last customer care connection.
  + Binned continuous variables like tenure into categories (short, medium, long) to test performance.

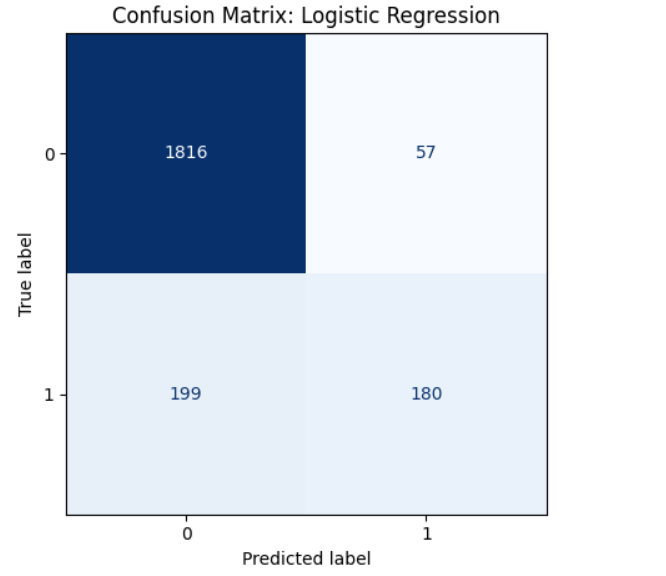
**4. Business Justification for Final Model**

XGBoost was chosen as the final model due to:

* **High recall (66%)** – crucial to identify potential churners and proactively engage them.
* **Business interpretability** via feature importance plots.
* Scalable for real-time prediction deployment.

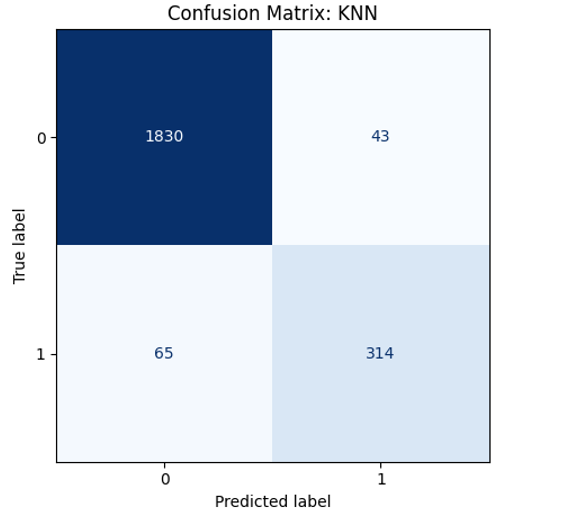
## **Perform Hyperparameter Tuning for Each Model (Original and SMOTE Data)**

### **Logistic Regression Original Dataset**



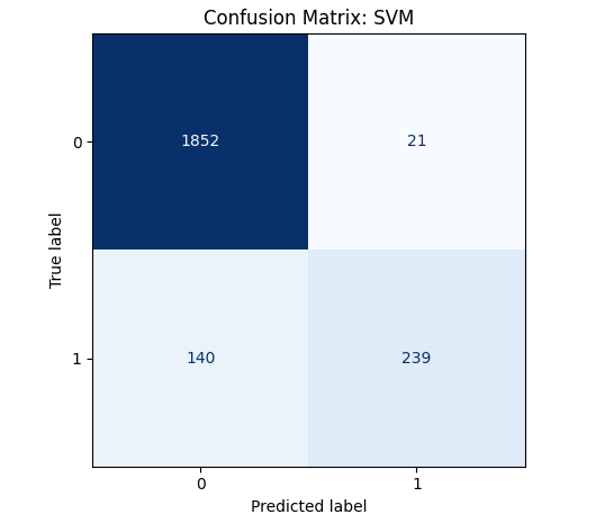
**Figure 80 Logistic Regression Original Dataset**

### **KNN Original Dataset**



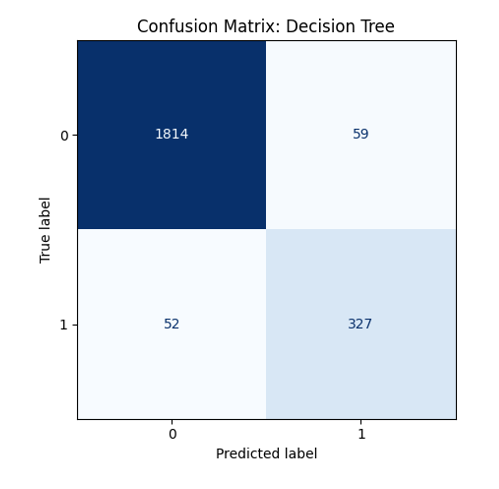
**Figure 81 KNN Original Dataset**

### **SVM Original Dataset**



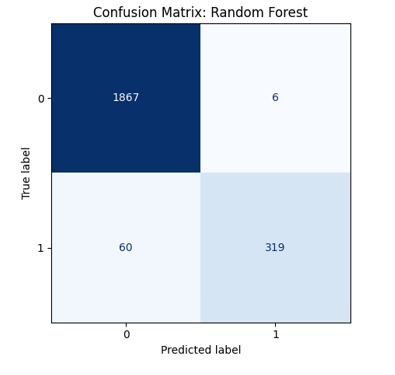
**Figure 82 SVM Original Dataset**

### **Decision Tree Original Dataset**



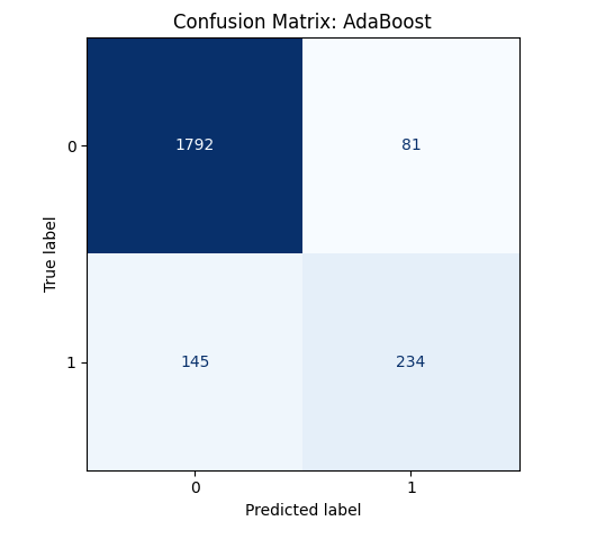
**Figure 83 Decision Tree Original Dataset**

### **Random Forest Original Dataset**



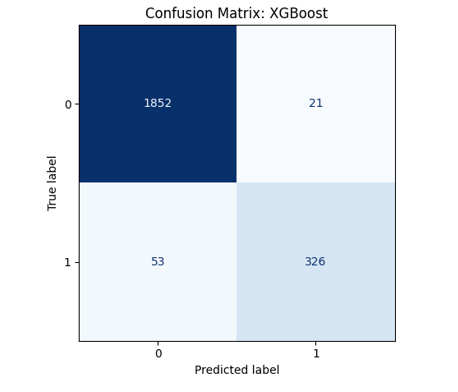
**Figure 84 Random Forest Original Dataset**

### **AdaBoost Original Dataset**



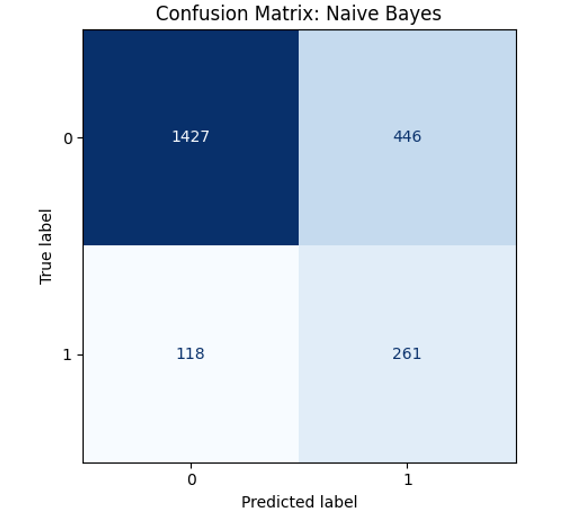
**Figure 85 AdaBoost Original Dataset**

### **XGBoost Original Dataset**



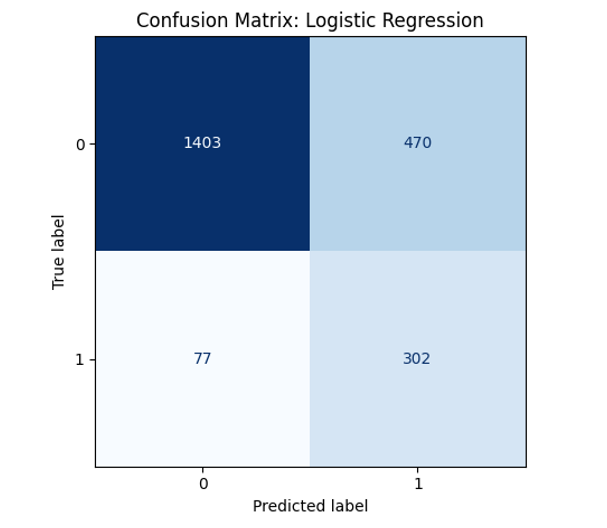
**Figure 86 XGBoost Original Dataset**

### **Naïve Bayes Original Dataset**



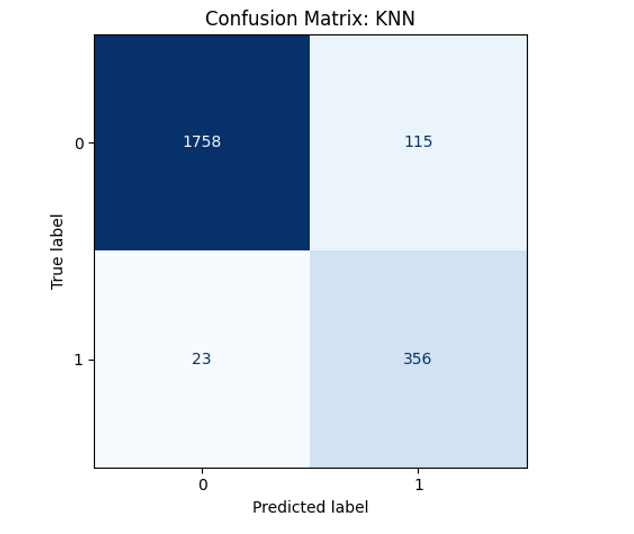
**Figure 87 Naïve Bayes Original Dataset**

### **Logistic Regression SMOTE Dataset**



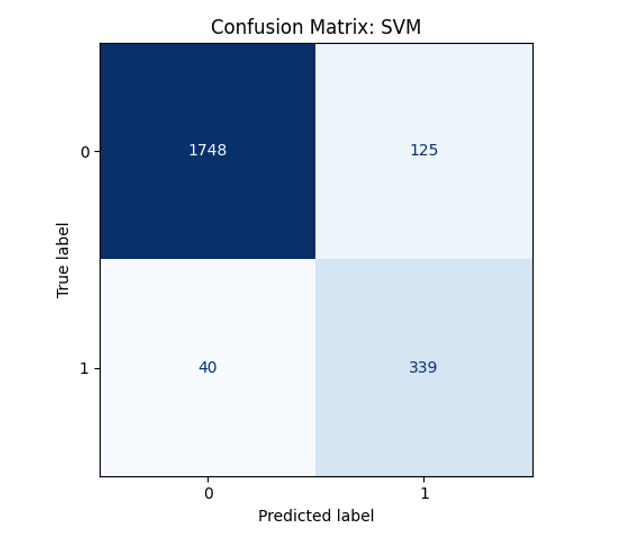
**Figure 88 Logistic Regression SMOTE Dataset**

### **KNN SMOTE Dataset**



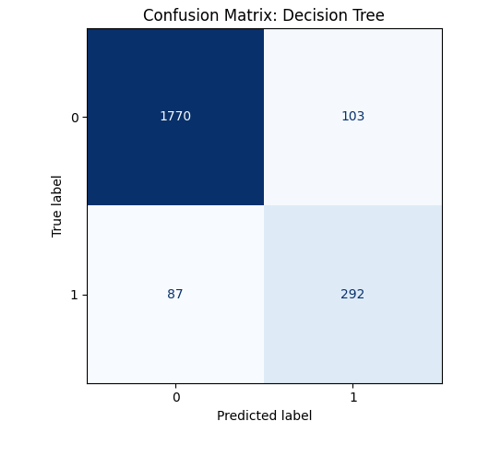
**Figure 89 KNN SMOTE Dataset**

### **SVM SMOTE Dataset**



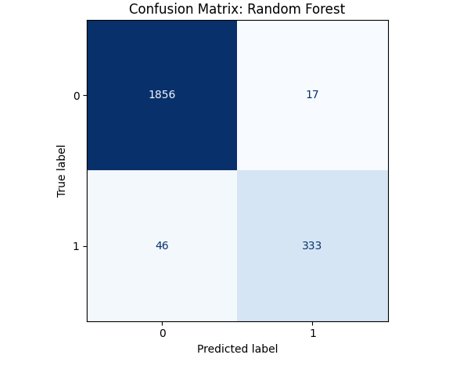
**Figure 90 SVM SMOTE Dataset**

### **Decision Tree SMOTE Dataset**



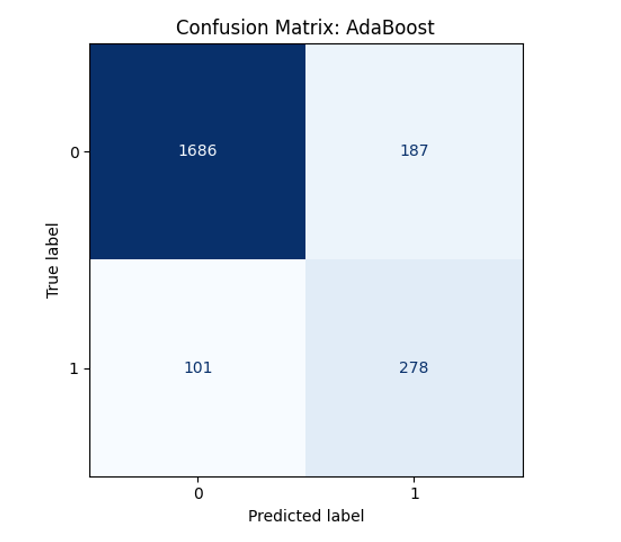
**Figure 91 Decision Tree SMOTE Dataset**

### **Random Forest SMOTE Dataset**



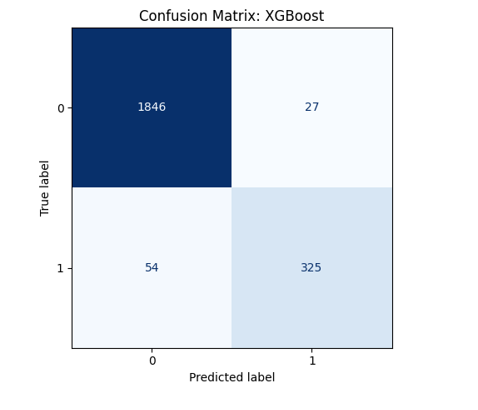
**Figure 92 Random Forest SMOTE Dataset**

### **AdaBoost SMOTE Dataset**



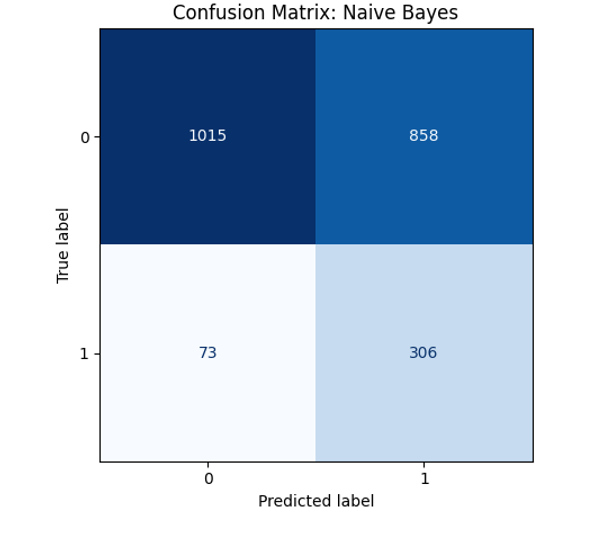
**Figure 93 AdaBoost SMOTE Dataset**

### **XGBoost SMOTE Dataset**



**Figure 94 XGBoost SMOTE Dataset**

### **Naïve Bayes SMOTE Dataset**



**Figure 95 Naïve Bayes SMOTE Dataset**

## **Model Comparison Original and SMOTE Dataset**



**Table 14 Model Comparison Original and SMOTE Dataset**

**Key Insights**

* **Ensemble models (XGBoost, Random Forest)** outperform others on both Original and SMOTE data.
* **SMOTE** helped improve recall and F1 for minority class in many models, but some (e.g., Naive Bayes, Logistic Regression) still underfit or overfit.
* **Train/Test F1 parity** and **ROC-AUC** suggest XGBoost and Random Forest are not just accurate but reliable.

# **Model Validation**

Model validation was conducted thoroughly to ensure that the selected model performs reliably across unseen data and supports confident decision-making for churn intervention strategies.

**1. Train-Test Split**

We initially split the dataset into **70% training** and **30% testing** subsets using stratified sampling to maintain class balance. This provided a realistic evaluation of how the model would generalize to new customer data.

**2. Cross-Validation**

To further assess model stability and reduce variance due to data splits, we applied **5-fold cross-validation** on the training set for all models. This approach enabled us to average performance metrics across multiple iterations and mitigate overfitting risks.

**Best Cross-Validation Accuracy: 0.95**

**3. Evaluation Metrics Beyond Accuracy**

Recognizing the **imbalance in the churn classes**, we evaluated models using a **comprehensive set of metrics**:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Class** | **Precision** | **Recall** | **F1-Score** | **Support** | | 0 (Majority) | 0.97 | 1 | 0.98 | 1873 | | 1 (Minority) | 0.98 | 0.84 | 0.91 | 379 | | **Accuracy** |  |  | **0.97** | 2252 | | **Macro Avg** | 0.98 | 0.92 | 0.94 |  | | **Weighted Avg** | 0.97 | 0.97 | 0.97 |  | |  |
| **Table 15 Comprehensive set of metrics**   * **Precision** is high for both classes (especially important for imbalanced data). * **Recall** is excellent for class 0 (1.00) and reasonably strong for class 1 (0.84), indicating that most positives are correctly identified. * **F1-Score** is balanced and strong (0.98 for class 0, 0.91 for class 1), highlighting good performance in both classes. |  |

**4. Confusion Matrix Analysis**

We also reviewed the **confusion matrix** to understand the types of errors being made:

|  |  |  |
| --- | --- | --- |
|  | **Predicted 0** | **Predicted 1** |
| **Actual 0** | 1867 (TN) | 6 (FP) |
| **Actual 1** | 60 (FN) | 319 (TP) |

**Table 16 Confusion Matrix Analysis**

 **True Negatives (TN)**: 1867 — correctly predicted non-events

 **False Positives (FP)**: 6 — very few incorrect positive predictions

 **True Positives (TP)**: 319 — majority of actual positives captured

 **False Negatives (FN)**: 60 — some positives missed, but acceptable

**5. Class Imbalance Handling**

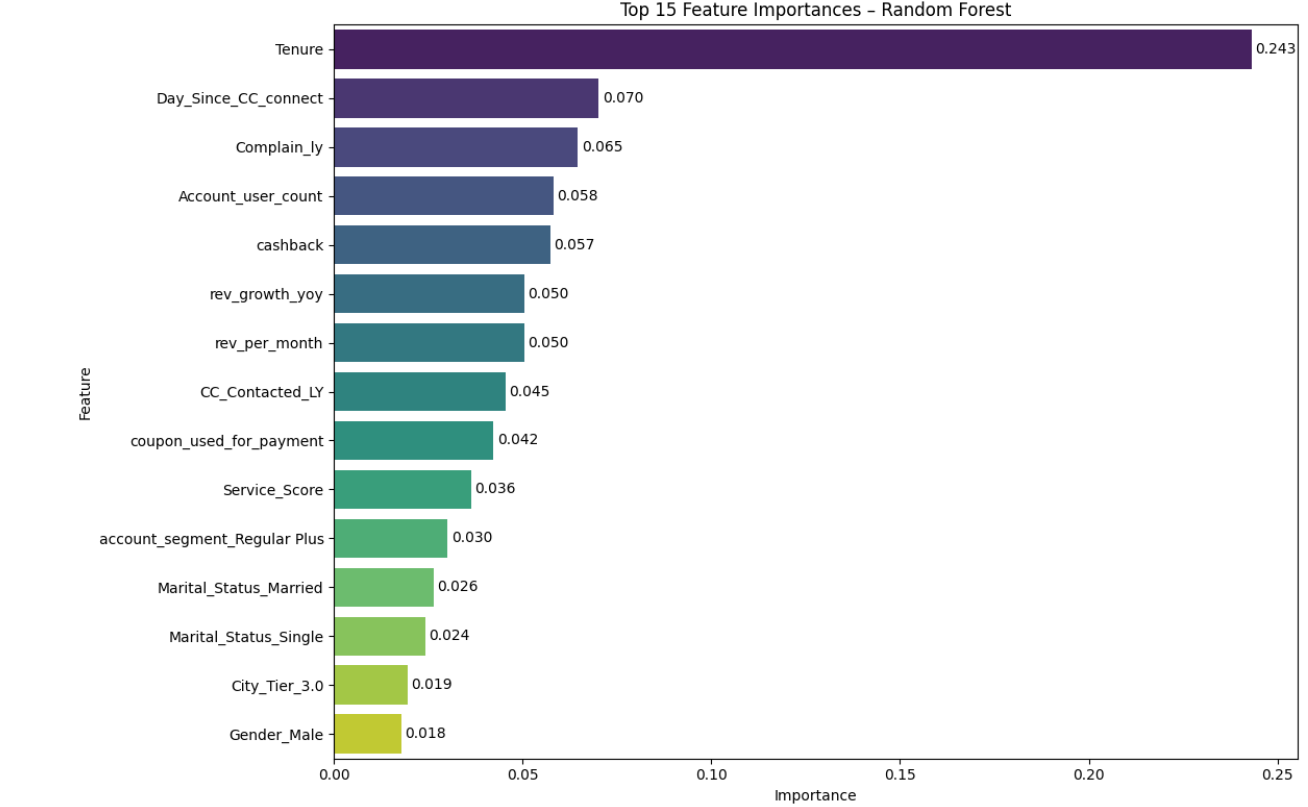
To ensure validation metrics were meaningful despite skewed class distribution:

* We used **SMOTE** during cross-validation to balance the classes.
* Also validated with original imbalanced data to confirm robustness.
* Compared metrics **with and without balancing** to assess overfitting risk.

**Conclusion**

* The **Random Forest** classifier performs **very well** overall.
* It demonstrates **high precision and recall**, particularly strong at **classifying the majority class** with near-perfect accuracy.
* There is a **minor trade-off** with recall for the minority class (1), but the **F1-score of 0.91** still reflects strong performance.
* With a **test accuracy of 97%** and minimal misclassifications, this tuned model is **highly reliable**.

# **Top 15 Feature Importances – Random Forest**



**Key Insights:**



**Table 17 Key Insights of Top Feature Importance**

# **Final Interpretation & Recommendations**

Based on the churn prediction analysis, several key insights were derived from both the data and the model outputs. These insights are translated into actionable recommendations for management:

## **Key Findings**

* **Short Tenure = Higher Risk:** Customers with lower tenure are significantly more likely to churn.
* **Lower Engagement Before Churn:** Customers who recently interacted with customer care (low Day\_Since\_CC\_Connect) are more likely to churn, indicating dissatisfaction.
* **Complaint History Matters:** Customers who have complained in the past year (Complain\_ly = 1) show higher churn rates.
* **Smaller User Groups Churn More:** Accounts with fewer users show higher churn, possibly due to lower perceived value or engagement.

## **Recommendations**

1. **Launch Targeted Retention Campaigns**:
   * Focus on customers with tenure < 6 months using proactive offers or loyalty incentives.
   * Provide personalized discounts or value-added services.
2. **Improve Customer Service Follow-up**:
   * Monitor recent support interactions more closely. A follow-up call/email can boost satisfaction and reduce churn.
3. **Address Complaint Root Causes**:
   * Conduct root cause analysis of frequent complaints and resolve them to improve overall experience.
4. **Bundle Offers for Small User Groups**:
   * Create tailored offers to encourage multi-user upgrades (e.g., family or group discounts) to improve retention.
5. **Build a Churn Dashboard**:
   * Integrate churn probability scores into a live dashboard so customer service can act in real time.

**Top Business Insights from the Model**

1. **Tenure (0.243)** is the most influential factor — long-standing customers are far less likely to churn.
2. **Recent Credit Card Connection Activity** (Day\_Since\_CC\_connect: 0.07) is highly indicative — low engagement suggests churn risk.
3. **Customer Complaints (Complain\_ly: 0.065)** strongly predict churn — dissatisfied customers are at higher risk.
4. **Account User Count & Cashback Impact Churn** — usage and incentives play a key role.
5. **Revenue Growth and Monthly Revenue** are central to understanding customer value.

**Actionable Business Recommendations**

**1. Customer Retention Focus**

* **Target low-tenure customers** (new users) with onboarding campaigns, welcome rewards, and personalized service to increase stickiness.
* **Monitor and re-engage inactive users** — those with long Day\_Since\_CC\_connect.

**2. Complaint Management System**

* Proactively **track and resolve complaints** through better support systems. Escalate resolution timelines for high-value customers.
* **Set up early-warning triggers** when customers complain — automatically flag them for retention offers.

**3. Incentive Optimization**

* Boost **cashback and coupon usage programs**, especially among churn-prone segments.
* Personalize cashback based on customer segment or activity (e.g., frequent spenders or multiple user accounts).

**4. Leverage Segmentation**

* Focus on **account segments like “Regular Plus”** and analyze behaviors more deeply — tailor communication/offers accordingly.
* Consider revisiting **service scoring logic** to align more closely with churn patterns.

**5. Strategic Investment in Data-Driven CX**

* Integrate these insights into your **CRM/retention systems**.
* Use predictive scoring from this model to **prioritize customer outreach**, especially those with high churn risk but high revenue.

**Final Note to Management**

The model shows excellent precision and recall, making it a reliable engine for churn prevention. Focus your retention budget on short-tenure users, inactive card users, and those who’ve raised complaints. These are the biggest levers to reduce churn and drive long-term profitability.

# **Appendix: Raw Code and Outputs**

Code Cell 11

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns   
from sklearn.linear\_model import LogisticRegression  
from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.svm import SVC  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.neural\_network import MLPClassifier  
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier  
from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, roc\_curve, roc\_auc\_score, accuracy\_score,precision\_score, recall\_score, f1\_score  
  
from sklearn.impute import KNNImputer  
from sklearn.preprocessing import StandardScaler  
from scipy import stats  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  
from sklearn.model\_selection import train\_test\_split, GridSearchCV # Train test Split and Grid Search  
  
import statsmodels.api as SM  
from sklearn import metrics  
  
from xgboost import XGBClassifier  
from sklearn.metrics import roc\_auc\_score, roc\_curve  
from sklearn.model\_selection import cross\_val\_score  
  
  
import warnings  
warnings.filterwarnings('ignore')  
  
from IPython.core.display import display, HTML  
display(HTML('<style>.container { width:90% !important; }<\style>'))

Code Cell 12

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns   
from sklearn.linear\_model import LogisticRegression  
from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.svm import SVC  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.neural\_network import MLPClassifier  
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier  
from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, roc\_curve, roc\_auc\_score, accuracy\_score,precision\_score, recall\_score, f1\_score  
  
from sklearn.impute import KNNImputer  
from sklearn.preprocessing import StandardScaler  
from scipy import stats  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  
from sklearn.model\_selection import train\_test\_split, GridSearchCV # Train test Split and Grid Search  
  
import statsmodels.api as SM  
from sklearn import metrics  
  
from xgboost import XGBClassifier  
from sklearn.metrics import roc\_auc\_score, roc\_curve  
from sklearn.model\_selection import cross\_val\_score  
  
  
import warnings  
warnings.filterwarnings('ignore')  
  
from IPython.core.display import display, HTML  
display(HTML('<style>.container { width:90% !important; }<\style>'))

Code Cell 13

# Load the dataset  
df = pd.read\_excel("Customer Churn Data.xlsx", sheet\_name="Data for DSBA")

Code Cell 14

# a) Understanding how data was collected  
print("Dataset Overview:")  
print(f"Number of rows: {df.shape[0]}")  
print(f"Number of columns: {df.shape[1]}")

Output:

Dataset Overview:  
Number of rows: 11260  
Number of columns: 19

Code Cell 15

# Total elements = rows × columns  
df.size

Output:

Code Cell 16

# Total number of rows  
len(df)

Output:

Code Cell 17

# Total non-null elements in the dataset  
df.count().sum()

Output:

Code Cell 18

# Total null elements in the dataset  
df.size - df.count().sum()

Output:

Code Cell 19

# Display first few rows to inspect structure  
df.head()

Output:

AccountID Churn Tenure City\_Tier CC\_Contacted\_LY Payment Gender \  
0 20000 1 4 3.0 6.0 Debit Card Female   
1 20001 1 0 1.0 8.0 UPI Male   
2 20002 1 0 1.0 30.0 Debit Card Male   
3 20003 1 0 3.0 15.0 Debit Card Male   
4 20004 1 0 1.0 12.0 Credit Card Male   
  
 Service\_Score Account\_user\_count account\_segment CC\_Agent\_Score \  
0 3.0 3 Super 2.0   
1 3.0 4 Regular Plus 3.0   
2 2.0 4 Regular Plus 3.0   
3 2.0 4 Super 5.0   
4 2.0 3 Regular Plus 5.0   
  
 Marital\_Status rev\_per\_month Complain\_ly rev\_growth\_yoy \  
0 Single 9 1.0 11   
1 Single 7 1.0 15   
2 Single 6 1.0 14   
3 Single 8 0.0 23   
4 Single 3 0.0 11   
  
 coupon\_used\_for\_payment Day\_Since\_CC\_connect cashback Login\_device   
0 1 5 159.93 Mobile   
1 0 0 120.9 Mobile   
2 0 3 NaN Mobile   
3 0 3 134.07 Mobile   
4 1 3 129.6 Mobile

Code Cell 20

print("\nLast 5 Rows of Data:")  
df.tail()

Output:

Last 5 Rows of Data:  
  
 AccountID Churn Tenure City\_Tier CC\_Contacted\_LY Payment \  
11255 31255 0 10 1.0 34.0 Credit Card   
11256 31256 0 13 1.0 19.0 Credit Card   
11257 31257 0 1 1.0 14.0 Debit Card   
11258 31258 0 23 3.0 11.0 Credit Card   
11259 31259 0 8 1.0 22.0 Credit Card   
  
 Gender Service\_Score Account\_user\_count account\_segment \  
11255 Male 3.0 2 Super   
11256 Male 3.0 5 HNI   
11257 Male 3.0 2 Super   
11258 Male 4.0 5 Super   
11259 Male 3.0 2 Super   
  
 CC\_Agent\_Score Marital\_Status rev\_per\_month Complain\_ly \  
11255 1.0 Married 9 0.0   
11256 5.0 Married 7 0.0   
11257 4.0 Married 7 1.0   
11258 4.0 Married 7 0.0   
11259 3.0 Married 5 0.0   
  
 rev\_growth\_yoy coupon\_used\_for\_payment Day\_Since\_CC\_connect cashback \  
11255 19 1 4 153.71   
11256 16 1 8 226.91   
11257 22 1 4 191.42   
11258 16 2 9 179.9   
11259 13 2 3 175.04   
  
 Login\_device   
11255 Computer   
11256 Mobile   
11257 Mobile   
11258 Computer   
11259 Mobile

Code Cell 21

print(f"Column Names: {df.columns.tolist()}")

Output:

Column Names: ['AccountID', 'Churn', 'Tenure', 'City\_Tier', 'CC\_Contacted\_LY', 'Payment', 'Gender', 'Service\_Score', 'Account\_user\_count', 'account\_segment', 'CC\_Agent\_Score', 'Marital\_Status', 'rev\_per\_month', 'Complain\_ly', 'rev\_growth\_yoy', 'coupon\_used\_for\_payment', 'Day\_Since\_CC\_connect', 'cashback', 'Login\_device']

Code Cell 22

# Check data types  
print("\nColumn Information:")  
print(df.info())

Output:

Column Information:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 11260 entries, 0 to 11259  
Data columns (total 19 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 AccountID 11260 non-null int64   
 1 Churn 11260 non-null int64   
 2 Tenure 11158 non-null object   
 3 City\_Tier 11148 non-null float64  
 4 CC\_Contacted\_LY 11158 non-null float64  
 5 Payment 11151 non-null object   
 6 Gender 11152 non-null object   
 7 Service\_Score 11162 non-null float64  
 8 Account\_user\_count 11148 non-null object   
 9 account\_segment 11163 non-null object   
 10 CC\_Agent\_Score 11144 non-null float64  
 11 Marital\_Status 11048 non-null object   
 12 rev\_per\_month 11158 non-null object   
 13 Complain\_ly 10903 non-null float64  
 14 rev\_growth\_yoy 11260 non-null object   
 15 coupon\_used\_for\_payment 11260 non-null object   
 16 Day\_Since\_CC\_connect 10903 non-null object   
 17 cashback 10789 non-null object   
 18 Login\_device 11039 non-null object   
dtypes: float64(5), int64(2), object(12)  
memory usage: 1.6+ MB  
None

Code Cell 23

# Check date-related columns to determine time range  
if 'date\_column' in df.columns:  
 df['date\_column'] = pd.to\_datetime(df['date\_column'])  
 print("Time range:", df['date\_column'].min(), "to", df['date\_column'].max())  
else:  
 print("No date column found. Check data documentation.")

Output:

No date column found. Check data documentation.

Code Cell 25

# Summary Statistics  
print("\nSummary Statistics:")  
df.describe()

Output:

Summary Statistics:  
  
 AccountID Churn City\_Tier CC\_Contacted\_LY \  
count 11260.00000 11260.000000 11148.000000 11158.000000   
mean 25629.50000 0.168384 1.653929 17.867091   
std 3250.62635 0.374223 0.915015 8.853269   
min 20000.00000 0.000000 1.000000 4.000000   
25% 22814.75000 0.000000 1.000000 11.000000   
50% 25629.50000 0.000000 1.000000 16.000000   
75% 28444.25000 0.000000 3.000000 23.000000   
max 31259.00000 1.000000 3.000000 132.000000   
  
 Service\_Score CC\_Agent\_Score Complain\_ly   
count 11162.000000 11144.000000 10903.000000   
mean 2.902526 3.066493 0.285334   
std 0.725584 1.379772 0.451594   
min 0.000000 1.000000 0.000000   
25% 2.000000 2.000000 0.000000   
50% 3.000000 3.000000 0.000000   
75% 3.000000 4.000000 1.000000   
max 5.000000 5.000000 1.000000

Code Cell 26

df.describe(include='all').T

Output:

count unique top freq mean \  
AccountID 11260.0 NaN NaN NaN 25629.5   
Churn 11260.0 NaN NaN NaN 0.168384   
Tenure 11158.0 38.0 1.0 1351.0 NaN   
City\_Tier 11148.0 NaN NaN NaN 1.653929   
CC\_Contacted\_LY 11158.0 NaN NaN NaN 17.867091   
Payment 11151 5 Debit Card 4587 NaN   
Gender 11152 4 Male 6328 NaN   
Service\_Score 11162.0 NaN NaN NaN 2.902526   
Account\_user\_count 11148.0 7.0 4.0 4569.0 NaN   
account\_segment 11163 7 Super 4062 NaN   
CC\_Agent\_Score 11144.0 NaN NaN NaN 3.066493   
Marital\_Status 11048 3 Married 5860 NaN   
rev\_per\_month 11158.0 59.0 3.0 1746.0 NaN   
Complain\_ly 10903.0 NaN NaN NaN 0.285334   
rev\_growth\_yoy 11260.0 20.0 14.0 1524.0 NaN   
coupon\_used\_for\_payment 11260.0 20.0 1.0 4373.0 NaN   
Day\_Since\_CC\_connect 10903.0 24.0 3.0 1816.0 NaN   
cashback 10789.0 5693.0 155.62 10.0 NaN   
Login\_device 11039 3 Mobile 7482 NaN   
  
 std min 25% 50% 75% \  
AccountID 3250.62635 20000.0 22814.75 25629.5 28444.25   
Churn 0.374223 0.0 0.0 0.0 0.0   
Tenure NaN NaN NaN NaN NaN   
City\_Tier 0.915015 1.0 1.0 1.0 3.0   
CC\_Contacted\_LY 8.853269 4.0 11.0 16.0 23.0   
Payment NaN NaN NaN NaN NaN   
Gender NaN NaN NaN NaN NaN   
Service\_Score 0.725584 0.0 2.0 3.0 3.0   
Account\_user\_count NaN NaN NaN NaN NaN   
account\_segment NaN NaN NaN NaN NaN   
CC\_Agent\_Score 1.379772 1.0 2.0 3.0 4.0   
Marital\_Status NaN NaN NaN NaN NaN   
rev\_per\_month NaN NaN NaN NaN NaN   
Complain\_ly 0.451594 0.0 0.0 0.0 1.0   
rev\_growth\_yoy NaN NaN NaN NaN NaN   
coupon\_used\_for\_payment NaN NaN NaN NaN NaN   
Day\_Since\_CC\_connect NaN NaN NaN NaN NaN   
cashback NaN NaN NaN NaN NaN   
Login\_device NaN NaN NaN NaN NaN   
  
 max   
AccountID 31259.0   
Churn 1.0   
Tenure NaN   
City\_Tier 3.0   
CC\_Contacted\_LY 132.0   
Payment NaN   
Gender NaN   
Service\_Score 5.0   
Account\_user\_count NaN   
account\_segment NaN   
CC\_Agent\_Score 5.0   
Marital\_Status NaN   
rev\_per\_month NaN   
Complain\_ly 1.0   
rev\_growth\_yoy NaN   
coupon\_used\_for\_payment NaN   
Day\_Since\_CC\_connect NaN   
cashback NaN   
Login\_device NaN

Code Cell 27

# Identify duplicate rows   
df.duplicated().sum()

Output:

0

Code Cell 28

# Check missing values  
missing\_values = df.isnull().sum()  
missing\_percentage = (missing\_values / len(df)) \* 100  
print(pd.DataFrame({'Missing Values': missing\_values, 'Percentage': missing\_percentage}))

Output:

Missing Values Percentage  
AccountID 0 0.000000  
Churn 0 0.000000  
Tenure 102 0.905861  
City\_Tier 112 0.994671  
CC\_Contacted\_LY 102 0.905861  
Payment 109 0.968028  
Gender 108 0.959147  
Service\_Score 98 0.870337  
Account\_user\_count 112 0.994671  
account\_segment 97 0.861456  
CC\_Agent\_Score 116 1.030195  
Marital\_Status 212 1.882771  
rev\_per\_month 102 0.905861  
Complain\_ly 357 3.170515  
rev\_growth\_yoy 0 0.000000  
coupon\_used\_for\_payment 0 0.000000  
Day\_Since\_CC\_connect 357 3.170515  
cashback 471 4.182948  
Login\_device 221 1.962700

Code Cell 30

# Count of missing values in the dataset  
missing\_values = df.isnull().sum().sum()  
total\_values = df.size # Total number of values in the dataset  
missing\_percentage = (missing\_values / total\_values) \* 100  
  
print(f"Total Missing Values: {missing\_values}")  
print(f"Percentage of Missing Data: {missing\_percentage:.2f}%")

Output:

Total Missing Values: 2676  
Percentage of Missing Data: 1.25%

Code Cell 32

# Missing values per column  
missing\_per\_column = df.isnull().sum()  
  
# Percentage of missing values per column  
missing\_percentage\_per\_column = (missing\_per\_column / len(df)) \* 100  
  
# Combine and display  
missing\_df = pd.DataFrame({  
 'Missing Values': missing\_per\_column,  
 'Missing Percentage': missing\_percentage\_per\_column  
}).sort\_values(by='Missing Values', ascending=False)  
  
print(missing\_df)

Output:

Missing Values Missing Percentage  
cashback 471 4.182948  
Day\_Since\_CC\_connect 357 3.170515  
Complain\_ly 357 3.170515  
Login\_device 221 1.962700  
Marital\_Status 212 1.882771  
CC\_Agent\_Score 116 1.030195  
City\_Tier 112 0.994671  
Account\_user\_count 112 0.994671  
Payment 109 0.968028  
Gender 108 0.959147  
Tenure 102 0.905861  
CC\_Contacted\_LY 102 0.905861  
rev\_per\_month 102 0.905861  
Service\_Score 98 0.870337  
account\_segment 97 0.861456  
coupon\_used\_for\_payment 0 0.000000  
rev\_growth\_yoy 0 0.000000  
Churn 0 0.000000  
AccountID 0 0.000000

Code Cell 34

# Missing values per row  
missing\_per\_row = df.isnull().sum(axis=1)  
  
# Percentage of missing values per row  
missing\_percentage\_per\_row = (missing\_per\_row / df.shape[1]) \* 100  
  
# Combine and display  
df['Missing\_Values\_Count'] = missing\_per\_row  
df['Missing\_Values\_Percentage'] = missing\_percentage\_per\_row  
  
print(df[['Missing\_Values\_Count', 'Missing\_Values\_Percentage']].head())

Output:

Missing\_Values\_Count Missing\_Values\_Percentage  
0 0 0.000000  
1 0 0.000000  
2 1 5.263158  
3 0 0.000000  
4 0 0.000000

Code Cell 36

import pandas as pd  
  
# Drop non-predictor columns ('AccountID' and 'Churn')  
predictor\_df = df.drop(['AccountID', 'Churn'], axis=1)  
  
# Count missing values per row  
missing\_counts\_per\_row = predictor\_df.isna().sum(axis=1)  
  
# Identify rows with 3 or more missing values  
rows\_3plus\_missing = predictor\_df.loc[missing\_counts\_per\_row >= 3]  
print(f"Rows with 3+ missing values: {rows\_3plus\_missing.shape[0]}")  
  
# Identify rows with exactly 2 missing values  
rows\_2\_missing = predictor\_df.loc[missing\_counts\_per\_row == 2]  
print(f"Rows with exactly 2 missing values: {rows\_2\_missing.shape[0]}")

Output:

Rows with 3+ missing values: 0  
Rows with exactly 2 missing values: 121

Code Cell 38

# Visualizing Missing Values   
# Heatmap to Detect Missing Values  
  
plt.figure(figsize=(12,6))  
sns.heatmap(df.isnull(), cmap="magma", cbar=False, yticklabels=False)  
plt.title("Missing Values Heatmap")  
plt.show()

Code Cell 39

import matplotlib.pyplot as plt  
  
missing\_counts = df.isnull().sum()  
missing\_counts = missing\_counts[missing\_counts > 0].sort\_values(ascending=False)  
  
plt.figure(figsize=(10, 5))  
bars = plt.bar(missing\_counts.index, missing\_counts.values, color='purple')  
plt.xlabel("Features")  
plt.ylabel("Missing Value Count")  
plt.title("Missing Values per Feature")  
plt.xticks(rotation=45, ha='right')  
  
# Add value labels on top of each bar  
for bar in bars:  
 height = bar.get\_height()  
 plt.text(bar.get\_x() + bar.get\_width()/2.0, height, int(height), ha='center', va='bottom', fontsize=9)  
  
plt.tight\_layout()  
plt.show()

Code Cell 40

plt.figure(figsize=(14, 6))  
df.boxplot(rot=45, patch\_artist=True, boxprops=dict(facecolor='lightblue'))  
plt.title("Feature Distributions & Outliers", fontsize=14)  
plt.ylabel("Value Range")  
plt.xticks(rotation=45, ha='right')  
plt.grid(axis='y', linestyle='--', alpha=0.7)  
plt.tight\_layout()  
plt.show()

Code Cell 41

import seaborn as sns  
import matplotlib.pyplot as plt  
  
sns.pairplot(df, hue='Churn', plot\_kws={'alpha': 0.6}, diag\_kind='kde', corner=True)  
  
plt.suptitle("Pairwise Feature Relationships by Churn", y=1.02, fontsize=14)  
plt.tight\_layout()  
plt.show()

Code Cell 43

# Summary Statistics for Quick Insights  
df.describe(include="all").T

Output:

count unique top freq mean \  
AccountID 11260.0 NaN NaN NaN 25629.5   
Churn 11260.0 NaN NaN NaN 0.168384   
Tenure 11158.0 38.0 1.0 1351.0 NaN   
City\_Tier 11148.0 NaN NaN NaN 1.653929   
CC\_Contacted\_LY 11158.0 NaN NaN NaN 17.867091   
Payment 11151 5 Debit Card 4587 NaN   
Gender 11152 4 Male 6328 NaN   
Service\_Score 11162.0 NaN NaN NaN 2.902526   
Account\_user\_count 11148.0 7.0 4.0 4569.0 NaN   
account\_segment 11163 7 Super 4062 NaN   
CC\_Agent\_Score 11144.0 NaN NaN NaN 3.066493   
Marital\_Status 11048 3 Married 5860 NaN   
rev\_per\_month 11158.0 59.0 3.0 1746.0 NaN   
Complain\_ly 10903.0 NaN NaN NaN 0.285334   
rev\_growth\_yoy 11260.0 20.0 14.0 1524.0 NaN   
coupon\_used\_for\_payment 11260.0 20.0 1.0 4373.0 NaN   
Day\_Since\_CC\_connect 10903.0 24.0 3.0 1816.0 NaN   
cashback 10789.0 5693.0 155.62 10.0 NaN   
Login\_device 11039 3 Mobile 7482 NaN   
Missing\_Values\_Count 11260.0 NaN NaN NaN 0.237655   
Missing\_Values\_Percentage 11260.0 NaN NaN NaN 1.250818   
  
 std min 25% 50% 75% \  
AccountID 3250.62635 20000.0 22814.75 25629.5 28444.25   
Churn 0.374223 0.0 0.0 0.0 0.0   
Tenure NaN NaN NaN NaN NaN   
City\_Tier 0.915015 1.0 1.0 1.0 3.0   
CC\_Contacted\_LY 8.853269 4.0 11.0 16.0 23.0   
Payment NaN NaN NaN NaN NaN   
Gender NaN NaN NaN NaN NaN   
Service\_Score 0.725584 0.0 2.0 3.0 3.0   
Account\_user\_count NaN NaN NaN NaN NaN   
account\_segment NaN NaN NaN NaN NaN   
CC\_Agent\_Score 1.379772 1.0 2.0 3.0 4.0   
Marital\_Status NaN NaN NaN NaN NaN   
rev\_per\_month NaN NaN NaN NaN NaN   
Complain\_ly 0.451594 0.0 0.0 0.0 1.0   
rev\_growth\_yoy NaN NaN NaN NaN NaN   
coupon\_used\_for\_payment NaN NaN NaN NaN NaN   
Day\_Since\_CC\_connect NaN NaN NaN NaN NaN   
cashback NaN NaN NaN NaN NaN   
Login\_device NaN NaN NaN NaN NaN   
Missing\_Values\_Count 0.450206 0.0 0.0 0.0 0.0   
Missing\_Values\_Percentage 2.369505 0.0 0.0 0.0 0.0   
  
 max   
AccountID 31259.0   
Churn 1.0   
Tenure NaN   
City\_Tier 3.0   
CC\_Contacted\_LY 132.0   
Payment NaN   
Gender NaN   
Service\_Score 5.0   
Account\_user\_count NaN   
account\_segment NaN   
CC\_Agent\_Score 5.0   
Marital\_Status NaN   
rev\_per\_month NaN   
Complain\_ly 1.0   
rev\_growth\_yoy NaN   
coupon\_used\_for\_payment NaN   
Day\_Since\_CC\_connect NaN   
cashback NaN   
Login\_device NaN   
Missing\_Values\_Count 2.0   
Missing\_Values\_Percentage 10.526316

Code Cell 44

Code Cell 49

import matplotlib.pyplot as plt  
import seaborn as sns  
  
num\_cols = df.select\_dtypes(include=['float64', 'int64']).columns  
  
for col in num\_cols:  
 plt.figure(figsize=(12, 5))  
  
 # Histogram with bar labels  
 plt.subplot(1, 2, 1)  
 counts, bins, patches = plt.hist(df[col].dropna(), bins=30, color="purple", edgecolor='black')  
 plt.title(f'Distribution of {col}')  
 plt.xlabel(col)  
 plt.ylabel("Frequency")  
  
 # Add labels on top of bars  
 for count, patch in zip(counts, patches):  
 if count > 0:  
 plt.text(patch.get\_x() + patch.get\_width()/2., count, int(count),   
 ha='center', va='bottom', fontsize=8)  
  
 # Boxplot  
 plt.subplot(1, 2, 2)  
 sns.boxplot(x=df[col], color="red")  
 plt.title(f'Boxplot of {col}')  
 plt.xlabel(col)  
  
 plt.tight\_layout()  
 plt.show()

Code Cell 50

Code Cell 52

import matplotlib.pyplot as plt  
import seaborn as sns  
  
cat\_cols = df.select\_dtypes(include=['object']).columns  
  
for col in cat\_cols:  
 plt.figure(figsize=(8, 4))  
 ax = sns.countplot(x=df[col], order=df[col].value\_counts().index, hue=df[col], palette="magma", legend=False)  
  
 # Add value labels on each bar  
 for p in ax.patches:  
 count = int(p.get\_height())  
 ax.annotate(count,   
 (p.get\_x() + p.get\_width() / 2., p.get\_height()),   
 ha='center', va='bottom', fontsize=9)  
  
 plt.title(f'Distribution of {col}')  
 plt.xlabel(col)  
 plt.ylabel("Count")  
 plt.xticks(rotation=45, ha='right')  
 plt.tight\_layout()  
 plt.show()

Code Cell 53

Code Cell 56

# Select only numeric columns  
df\_numeric = df.select\_dtypes(include=['float64', 'int64'])  
  
# Check if df\_numeric contains only numeric data  
print(df\_numeric.dtypes)  
  
# Plot correlation heatmap  
plt.figure(figsize=(10, 6))  
sns.heatmap(df\_numeric.corr(), annot=True, cmap='coolwarm', fmt='.2f')  
plt.title("Correlation Matrix")  
plt.show()

Output:

AccountID int64  
Churn int64  
City\_Tier float64  
CC\_Contacted\_LY float64  
Service\_Score float64  
CC\_Agent\_Score float64  
Complain\_ly float64  
Missing\_Values\_Count int64  
Missing\_Values\_Percentage float64  
dtype: object

Code Cell 57

Code Cell 58

sns.pairplot(df[num\_cols], diag\_kind="kde", corner=True, plot\_kws={'alpha': 0.6})  
plt.suptitle("Pairwise Relationships Between Numerical Features", y=1.02, fontsize=14)  
plt.tight\_layout()  
plt.show()

Code Cell 59

Code Cell 60

# Plot barplots with value labels on top  
for col in cat\_cols:  
 for num in num\_cols:  
 plt.figure(figsize=(8, 4))  
 ax = sns.barplot(data=df, x=col, y=num, hue=col, legend=False, estimator='mean', errorbar=None, palette="magma")  
   
 # Add value labels on top of bars  
 for p in ax.patches:  
 height = p.get\_height()  
 ax.annotate(f'{height:.2f}',   
 (p.get\_x() + p.get\_width() / 2., height),   
 ha='center', va='bottom', fontsize=9)  
   
 plt.xticks(rotation=45, ha='right')  
 plt.title(f'{num} by {col}', fontsize=12)  
 plt.xlabel(col)  
 plt.ylabel(f'Mean of {num}')  
 plt.grid(axis='y', linestyle='--', alpha=0.6)  
 plt.tight\_layout()  
 plt.show()

Code Cell 63

#Removal of unwanted variables  
unwanted\_cols = ['AccountID']   
if set(unwanted\_cols).issubset(df.columns):  
 df.drop(columns=unwanted\_cols, inplace=True)

Code Cell 65

# Check for unique values in categorical columns  
df.describe(include='object').T

Output:

count unique top freq  
Tenure 11158 38 1 1351  
Payment 11151 5 Debit Card 4587  
Gender 11152 4 Male 6328  
Account\_user\_count 11148 7 4 4569  
account\_segment 11163 7 Super 4062  
Marital\_Status 11048 3 Married 5860  
rev\_per\_month 11158 59 3 1746  
rev\_growth\_yoy 11260 20 14 1524  
coupon\_used\_for\_payment 11260 20 1 4373  
Day\_Since\_CC\_connect 10903 24 3 1816  
cashback 10789.0 5693.0 155.62 10.0  
Login\_device 11039 3 Mobile 7482

Code Cell 66

# Iterate over all columns and print unique values  
for col in df.columns:  
 print(f"\nUnique values in {col}:")  
 print(df[col].unique())

Output:

Unique values in Churn:  
[1 0]  
  
Unique values in Tenure:  
[4 0 2 13 11 '#' 9 99 19 20 14 8 26 18 5 30 7 1 23 3 29 6 28 24 25 16 10  
 15 22 nan 27 12 21 17 50 60 31 51 61]  
  
Unique values in City\_Tier:  
[ 3. 1. nan 2.]  
  
Unique values in CC\_Contacted\_LY:  
[ 6. 8. 30. 15. 12. 22. 11. 9. 31. 18. 13. 20. 29. 28.  
 26. 14. 10. 25. 27. 17. 23. 33. 19. 35. 24. 16. 32. 21.  
 nan 34. 5. 4. 126. 7. 36. 127. 42. 38. 37. 39. 40. 41.  
 132. 43. 129.]  
  
Unique values in Payment:  
['Debit Card' 'UPI' 'Credit Card' 'Cash on Delivery' 'E wallet' nan]  
  
Unique values in Gender:  
['Female' 'Male' 'F' nan 'M']  
  
Unique values in Service\_Score:  
[ 3. 2. 1. nan 0. 4. 5.]  
  
Unique values in Account\_user\_count:  
[3 4 nan 5 2 '@' 1 6]  
  
Unique values in account\_segment:  
['Super' 'Regular Plus' 'Regular' 'HNI' 'Regular +' nan 'Super Plus'  
 'Super +']  
  
Unique values in CC\_Agent\_Score:  
[ 2. 3. 5. 4. nan 1.]  
  
Unique values in Marital\_Status:  
['Single' 'Divorced' 'Married' nan]  
  
Unique values in rev\_per\_month:  
[9 7 6 8 3 2 4 10 1 5 '+' 130 nan 19 139 102 120 138 127 123 124 116 21  
 126 134 113 114 108 140 133 129 107 118 11 105 20 119 121 137 110 22 101  
 136 125 14 13 12 115 23 122 117 131 104 15 25 135 111 109 100 103]  
  
Unique values in Complain\_ly:  
[ 1. 0. nan]  
  
Unique values in rev\_growth\_yoy:  
[11 15 14 23 22 16 12 13 17 18 24 19 20 21 25 26 '$' 4 27 28]  
  
Unique values in coupon\_used\_for\_payment:  
[1 0 4 2 9 6 11 7 12 10 5 3 13 15 8 '#' '$' 14 '\*' 16]  
  
Unique values in Day\_Since\_CC\_connect:  
[5 0 3 7 2 1 8 6 4 15 nan 11 10 9 13 12 17 16 14 30 '$' 46 18 31 47]  
  
Unique values in cashback:  
[159.93 120.9 nan ... 227.36 226.91 191.42]  
  
Unique values in Login\_device:  
['Mobile' 'Computer' '&&&&' nan]  
  
Unique values in Missing\_Values\_Count:  
[0 1 2]  
  
Unique values in Missing\_Values\_Percentage:  
[ 0. 5.26315789 10.52631579]

Code Cell 68

# Replace invalid values with NaN  
df.replace({'#': np.nan, '@': np.nan, '+': np.nan, '$': np.nan, '\*': np.nan, '&&&&': np.nan}, inplace=True)

Code Cell 70

cat\_cols\_missing = ['City\_Tier', 'Payment', 'Gender', 'Service\_Score', 'account\_segment', 'CC\_Agent\_Score', 'Marital\_Status', 'Complain\_ly', 'Login\_device']  
for col in cat\_cols\_missing:  
 df[col].fillna(df[col].mode()[0], inplace=True)

Code Cell 72

# Ensure df is a DataFrame  
num\_cols = df.select\_dtypes(include=['number']) # Select only numeric columns  
  
# Compute skewness  
skewness = num\_cols.skew().sort\_values(ascending=False).round(2)  
  
print(skewness)

Output:

rev\_per\_month 9.09  
cashback 8.77  
Tenure 3.90  
coupon\_used\_for\_payment 2.58  
Churn 1.77  
Missing\_Values\_Count 1.58  
Missing\_Values\_Percentage 1.58  
CC\_Contacted\_LY 1.42  
Day\_Since\_CC\_connect 1.27  
Complain\_ly 1.00  
City\_Tier 0.75  
rev\_growth\_yoy 0.75  
Service\_Score 0.00  
CC\_Agent\_Score -0.14  
Account\_user\_count -0.39  
dtype: float64

Code Cell 73

import numpy as np  
  
# List of columns with non-numeric values  
num\_cols\_missing = ['Tenure', 'CC\_Contacted\_LY', 'rev\_per\_month', 'rev\_growth\_yoy', 'coupon\_used\_for\_payment', 'Day\_Since\_CC\_connect', 'cashback', 'Account\_user\_count']  
  
# Replace non-numeric values with NaN  
for col in num\_cols\_missing:  
 df[col] = pd.to\_numeric(df[col], errors='coerce') # Convert non-numeric to NaN  
  
# Fill missing values with the median of each column  
for col in num\_cols\_missing:  
 df[col].fillna(df[col].median(), inplace=True)

Code Cell 75

df['Gender'] = df['Gender'].replace({'M': 'Male', 'F': 'Female'})

Code Cell 78

df['account\_segment'] = df['account\_segment'].replace({  
 'Regular +': 'Regular Plus',  
 'Super +': 'Super Plus'  
})  
  
# Fill missing values with mode  
df['account\_segment'].fillna(df['account\_segment'].mode()[0], inplace=True)

Code Cell 79

print(df.isnull().sum())

Output:

Churn 0  
Tenure 0  
City\_Tier 0  
CC\_Contacted\_LY 0  
Payment 0  
Gender 0  
Service\_Score 0  
Account\_user\_count 0  
account\_segment 0  
CC\_Agent\_Score 0  
Marital\_Status 0  
rev\_per\_month 0  
Complain\_ly 0  
rev\_growth\_yoy 0  
coupon\_used\_for\_payment 0  
Day\_Since\_CC\_connect 0  
cashback 0  
Login\_device 0  
Missing\_Values\_Count 0  
Missing\_Values\_Percentage 0  
dtype: int64

Code Cell 82

# Detect outliers using the IQR method  
def detect\_outliers\_iqr(df, column):  
 Q1 = df[column].quantile(0.25)  
 Q3 = df[column].quantile(0.75)  
 IQR = Q3 - Q1  
 lower\_bound = Q1 - 1.5 \* IQR  
 upper\_bound = Q3 + 1.5 \* IQR  
 return df[(df[column] < lower\_bound) | (df[column] > upper\_bound)]  
  
def detect\_outliers\_iqr(data, column):  
 Q1 = data[column].quantile(0.25)  
 Q3 = data[column].quantile(0.75)  
 IQR = Q3 - Q1  
 lower\_bound = Q1 - 1.5 \* IQR  
 upper\_bound = Q3 + 1.5 \* IQR  
 return data[(data[column] < lower\_bound) | (data[column] > upper\_bound)]  
  
# Select numerical columns  
num\_cols = df.select\_dtypes(include=['float64', 'int64']).columns  
  
# Plot and detect  
for col in num\_cols:  
 plt.figure(figsize=(8, 4))  
 sns.boxplot(x=df[col], color='tomato')  
 plt.title(f'Boxplot of {col}', fontsize=12)  
 plt.xlabel(col)  
 plt.grid(axis='x', linestyle='--', alpha=0.6)  
 plt.tight\_layout()  
 plt.show()  
  
 outliers = detect\_outliers\_iqr(df, col)  
 print(f"Outliers detected in {col}: {len(outliers)}")

Output:

Outliers detected in Churn: 1896  
  
Outliers detected in Tenure: 139  
  
Outliers detected in City\_Tier: 0  
  
Outliers detected in CC\_Contacted\_LY: 42  
  
Outliers detected in Service\_Score: 13  
  
Outliers detected in Account\_user\_count: 761  
  
Outliers detected in CC\_Agent\_Score: 0  
  
Outliers detected in rev\_per\_month: 185  
  
Outliers detected in Complain\_ly: 0  
  
Outliers detected in rev\_growth\_yoy: 0  
  
Outliers detected in coupon\_used\_for\_payment: 1380  
  
Outliers detected in Day\_Since\_CC\_connect: 130  
  
Outliers detected in cashback: 986  
  
Outliers detected in Missing\_Values\_Count: 2555  
  
Outliers detected in Missing\_Values\_Percentage: 2555

Code Cell 85

print(df.dtypes)

Output:

Churn int64  
Tenure float64  
City\_Tier float64  
CC\_Contacted\_LY float64  
Payment object  
Gender object  
Service\_Score float64  
Account\_user\_count float64  
account\_segment object  
CC\_Agent\_Score float64  
Marital\_Status object  
rev\_per\_month float64  
Complain\_ly float64  
rev\_growth\_yoy float64  
coupon\_used\_for\_payment float64  
Day\_Since\_CC\_connect float64  
cashback float64  
Login\_device object  
Missing\_Values\_Count int64  
Missing\_Values\_Percentage float64  
dtype: object

Code Cell 87

# Define columns to be converted to object  
object\_columns = ['Churn', 'City\_Tier', 'Payment', 'Gender', 'Service\_Score',   
 'account\_segment', 'CC\_Agent\_Score', 'Marital\_Status',   
 'Complain\_ly', 'Login\_device']  
  
# Convert selected columns to object (categorical)  
df[object\_columns] = df[object\_columns].astype(str)  
  
# Convert all remaining columns to numeric (int/float)  
for col in df.columns:  
 if col not in object\_columns:  
 df[col] = pd.to\_numeric(df[col], errors='coerce') # Convert, handling errors  
  
# Verify the changes  
print(df.dtypes)

Output:

Churn object  
Tenure float64  
City\_Tier object  
CC\_Contacted\_LY float64  
Payment object  
Gender object  
Service\_Score object  
Account\_user\_count float64  
account\_segment object  
CC\_Agent\_Score object  
Marital\_Status object  
rev\_per\_month float64  
Complain\_ly object  
rev\_growth\_yoy float64  
coupon\_used\_for\_payment float64  
Day\_Since\_CC\_connect float64  
cashback float64  
Login\_device object  
Missing\_Values\_Count int64  
Missing\_Values\_Percentage float64  
dtype: object

Code Cell 90

import matplotlib.pyplot as plt  
import seaborn as sns  
  
# Categorical columns  
cat\_cols = df.select\_dtypes(include=['object', 'category']).columns  
  
for col in cat\_cols:  
 plt.figure(figsize=(10, 4))  
 ax = sns.countplot(x=df[col],hue=df[col], palette="magma", order=df[col].value\_counts().index, legend=False)  
  
 # Add value labels  
 for p in ax.patches:  
 height = p.get\_height()  
 ax.annotate(f'{height}',   
 (p.get\_x() + p.get\_width() / 2., height),  
 ha='center', va='bottom', fontsize=9)  
  
 plt.title(f"Distribution of {col}", fontsize=12)  
 plt.xticks(rotation=45, ha='right')  
 plt.xlabel(col)  
 plt.ylabel("Count")  
 plt.tight\_layout()  
 plt.show()

Code Cell 92

import matplotlib.pyplot as plt  
import seaborn as sns  
  
plt.figure(figsize=(14, 10))  
for i, col in enumerate(num\_cols, 1):  
 plt.subplot(4, 4, i)  
 sns.boxplot(x='Churn', y=col, data=df,hue='Churn',legend=False, palette="viridis")  
 plt.title(f"{col} vs Churn", fontsize=10)  
 plt.xlabel('')  
 plt.ylabel('')  
 plt.xticks(rotation=0)  
  
plt.suptitle("Boxplots of Numerical Features by Churn", fontsize=14)  
plt.tight\_layout(rect=[0, 0.03, 1, 0.95])  
plt.show()

Code Cell 93

import matplotlib.pyplot as plt  
import seaborn as sns  
  
for col in cat\_cols:  
 plt.figure(figsize=(8, 4))  
 ax = sns.countplot(x=df[col], hue=df['Churn'], palette="mako", order=df[col].value\_counts().index)  
  
 # Add value labels on top of each bar  
 for p in ax.patches:  
 count = int(p.get\_height())  
 ax.annotate(str(count),  
 (p.get\_x() + p.get\_width() / 2., p.get\_height()),  
 ha='center', va='bottom', fontsize=9)  
  
 plt.title(f"{col} vs Churn", fontsize=12)  
 plt.xlabel(col)  
 plt.ylabel("Count")  
 plt.xticks(rotation=45, ha='right')  
 plt.tight\_layout()  
 plt.show()

Code Cell 95

# Check the distribution of the target variable  
churn\_counts = df['Churn'].value\_counts()  
churn\_percentage = (churn\_counts / len(df)) \* 100  
  
print("Churn Distribution:\n", churn\_counts)  
print("\nChurn Percentage:\n", churn\_percentage)

Output:

Churn Distribution:  
 Churn  
0 9364  
1 1896  
Name: count, dtype: int64  
  
Churn Percentage:  
 Churn  
0 83.161634  
1 16.838366  
Name: count, dtype: float64

Code Cell 96

import matplotlib.pyplot as plt  
import seaborn as sns  
  
# Assuming churn\_counts is defined like:  
# churn\_counts = df['Churn'].value\_counts()  
  
plt.figure(figsize=(6, 4))  
ax = sns.barplot(x=churn\_counts.index, y=churn\_counts.values,hue=churn\_counts.index,legend=False, palette=['blue', 'red'])  
  
# Add value labels on top of bars  
for i, val in enumerate(churn\_counts.values):  
 ax.text(i, val + 1, str(val), ha='center', va='bottom', fontsize=10)  
  
plt.xlabel("Churn")  
plt.ylabel("Number of Accounts")  
plt.title("Churn Class Distribution")  
plt.tight\_layout()  
plt.show()

Code Cell 97

import scipy.stats as stats  
  
num\_cols = [  
 'Tenure', 'cashback', 'Day\_Since\_CC\_connect',  
 'rev\_growth\_yoy','CC\_Contacted\_LY', 'rev\_per\_month','coupon\_used\_for\_payment'  
]  
  
for col in num\_cols:  
 churn\_groups = [df[df['Churn'] == val][col].dropna() for val in df['Churn'].unique()]  
   
 if all(len(group) > 0 for group in churn\_groups): # Ensure no empty group  
 f\_stat, p\_val = stats.f\_oneway(\*churn\_groups)  
 print(f"ANOVA for {col} vs Churn --> F = {f\_stat:.2f}, p = {p\_val:.4f}")  
 if p\_val < 0.05:  
 print(f"Significant difference in means — {col} could be an important feature.\n")  
 else:  
 print(f"No significant difference — {col} may not help separate churn.\n")

Output:

ANOVA for Tenure vs Churn --> F = 632.81, p = 0.0000  
Significant difference in means — Tenure could be an important feature.  
  
ANOVA for cashback vs Churn --> F = 11.44, p = 0.0007  
Significant difference in means — cashback could be an important feature.  
  
ANOVA for Day\_Since\_CC\_connect vs Churn --> F = 243.19, p = 0.0000  
Significant difference in means — Day\_Since\_CC\_connect could be an important feature.  
  
ANOVA for rev\_growth\_yoy vs Churn --> F = 2.16, p = 0.1420  
No significant difference — rev\_growth\_yoy may not help separate churn.  
  
ANOVA for CC\_Contacted\_LY vs Churn --> F = 58.14, p = 0.0000  
Significant difference in means — CC\_Contacted\_LY could be an important feature.  
  
ANOVA for rev\_per\_month vs Churn --> F = 5.51, p = 0.0189  
Significant difference in means — rev\_per\_month could be an important feature.  
  
ANOVA for coupon\_used\_for\_payment vs Churn --> F = 2.46, p = 0.1169  
No significant difference — coupon\_used\_for\_payment may not help separate churn.

Code Cell 98

import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
from scipy.stats import chi2\_contingency  
  
cat\_cols = ['Gender', 'Marital\_Status', 'Login\_device', 'City\_Tier', 'Account\_user\_count',  
 'Payment', 'Service\_Score', 'account\_segment', 'Complain\_ly']  
  
for col in cat\_cols:  
 # Chi-square test  
 contingency\_table = pd.crosstab(df[col], df['Churn'])  
 chi2, p, dof, expected = chi2\_contingency(contingency\_table)  
 print(f"Chi-Square Test for {col} vs Churn:")  
 print(f"Chi2 = {chi2:.2f}, p-value = {p:.4f}")  
 if p < 0.05:  
 print(f" Significant association with churn.\n")  
 else:  
 print(f" No significant association with churn.\n")  
   
 # Countplot with value labels  
 plt.figure(figsize=(6, 3))  
 ax = sns.countplot(x=col, hue='Churn', data=df, palette='magma')  
 plt.title(f'{col} vs Churn')  
 plt.xticks(rotation=45, ha='right')  
 plt.xlabel(col)  
 plt.ylabel('Count')  
  
 # Add bar labels  
 for p in ax.patches:  
 height = p.get\_height()  
 ax.annotate(f'{height}',   
 (p.get\_x() + p.get\_width() / 2., height),   
 ha='center', va='bottom', fontsize=8)  
  
 plt.tight\_layout()  
 plt.show()

Output:

Chi-Square Test for Gender vs Churn:  
Chi2 = 9.39, p-value = 0.0022  
 Significant association with churn.  
  
  
Chi-Square Test for Marital\_Status vs Churn:  
Chi2 = 378.98, p-value = 0.0000  
 Significant association with churn.  
  
  
Chi-Square Test for Login\_device vs Churn:  
Chi2 = 25.22, p-value = 0.0000  
 Significant association with churn.  
  
  
Chi-Square Test for City\_Tier vs Churn:  
Chi2 = 80.54, p-value = 0.0000  
 Significant association with churn.  
  
  
Chi-Square Test for Account\_user\_count vs Churn:  
Chi2 = 155.98, p-value = 0.0000  
 Significant association with churn.  
  
  
Chi-Square Test for Payment vs Churn:  
Chi2 = 102.71, p-value = 0.0000  
 Significant association with churn.  
  
  
Chi-Square Test for Service\_Score vs Churn:  
Chi2 = 18.40, p-value = 0.0025  
 Significant association with churn.  
  
  
Chi-Square Test for account\_segment vs Churn:  
Chi2 = 562.20, p-value = 0.0000  
 Significant association with churn.  
  
  
Chi-Square Test for Complain\_ly vs Churn:  
Chi2 = 681.88, p-value = 0.0000  
 Significant association with churn.

Code Cell 100

import pandas as pd  
from sklearn.preprocessing import LabelEncoder  
  
ordinal\_cols = ['Service\_Score']  
label\_enc = LabelEncoder()  
  
for col in ordinal\_cols:  
 df[col] = label\_enc.fit\_transform(df[col])  
  
  
nominal\_cols = ['Gender', 'Marital\_Status', 'Login\_device', 'City\_Tier',  
 'Payment', 'account\_segment']  
  
# This will create binary columns for each category  
df = pd.get\_dummies(df, columns=nominal\_cols, drop\_first=True)  
  
# Final Step: Check your encoded data  
  
print(df.head())  
print(f"Final shape of dataset after encoding: {df.shape}")

Output:

Churn Tenure CC\_Contacted\_LY Service\_Score Account\_user\_count \  
0 1 4.0 6.0 3 3.0   
1 1 0.0 8.0 3 4.0   
2 1 0.0 30.0 2 4.0   
3 1 0.0 15.0 2 4.0   
4 1 0.0 12.0 2 3.0   
  
 CC\_Agent\_Score rev\_per\_month Complain\_ly rev\_growth\_yoy \  
0 2.0 9.0 1.0 11.0   
1 3.0 7.0 1.0 15.0   
2 3.0 6.0 1.0 14.0   
3 5.0 8.0 0.0 23.0   
4 5.0 3.0 0.0 11.0   
  
 coupon\_used\_for\_payment ... City\_Tier\_2.0 City\_Tier\_3.0 \  
0 1.0 ... False True   
1 0.0 ... False False   
2 0.0 ... False False   
3 0.0 ... False True   
4 1.0 ... False False   
  
 Payment\_Credit Card Payment\_Debit Card Payment\_E wallet Payment\_UPI \  
0 False True False False   
1 False False False True   
2 False True False False   
3 False True False False   
4 True False False False   
  
 account\_segment\_Regular account\_segment\_Regular Plus \  
0 False False   
1 False True   
2 False True   
3 False False   
4 False True   
  
 account\_segment\_Super account\_segment\_Super Plus   
0 True False   
1 False False   
2 False False   
3 True False   
4 False False   
  
[5 rows x 28 columns]  
Final shape of dataset after encoding: (11260, 28)

Code Cell 102

# Define selected numerical features based on ANOVA results  
selected\_features = ['Tenure', 'cashback', 'Day\_Since\_CC\_connect', 'CC\_Contacted\_LY', 'rev\_per\_month']  
  
# Update feature set  
X = df[selected\_features] # Keeping only significant features  
y = df['Churn'] # Target variable

Code Cell 104

# Identify all categorical columns after encoding  
encoded\_categorical\_features = [col for col in df.columns if any(cat in col for cat in nominal\_cols)]  
  
# Define final features: Numerical + Encoded Categorical  
final\_features = selected\_features + encoded\_categorical\_features  
  
# Select only these features  
X = df[final\_features]  
y = df['Churn'] # Target variable

Code Cell 105

!pip install kmodes

Output:

Defaulting to user installation because normal site-packages is not writeable  
Requirement already satisfied: kmodes in c:\users\agnes\appdata\roaming\python\python311\site-packages (0.12.2)  
Requirement already satisfied: numpy>=1.10.4 in c:\users\agnes\appdata\roaming\python\python311\site-packages (from kmodes) (1.25.2)  
Requirement already satisfied: scikit-learn>=0.22.0 in c:\programdata\anaconda3\lib\site-packages (from kmodes) (1.2.2)  
Requirement already satisfied: scipy>=0.13.3 in c:\programdata\anaconda3\lib\site-packages (from kmodes) (1.11.4)  
Requirement already satisfied: joblib>=0.11 in c:\programdata\anaconda3\lib\site-packages (from kmodes) (1.2.0)  
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\agnes\appdata\roaming\python\python311\site-packages (from scikit-learn>=0.22.0->kmodes) (3.1.0)

Code Cell 106

# Convert numerical columns to numeric types  
df[selected\_features] = df[selected\_features].apply(pd.to\_numeric, errors='coerce')  
  
# Convert encoded categorical columns to string (important for K-Prototypes)  
df[encoded\_categorical\_features] = df[encoded\_categorical\_features].astype(str)  
  
# Extract feature matrix  
X = df[final\_features].values # Convert dataframe to NumPy array  
  
# Identify categorical feature indices (position of categorical columns in X)  
categorical\_indices = list(range(len(selected\_features), len(final\_features)))   
from kmodes.kprototypes import KPrototypes  
  
# Define the model (set n\_clusters based on business understanding)  
kproto = KPrototypes(n\_clusters=3, random\_state=42)  
  
# Fit and predict clusters  
clusters = kproto.fit\_predict(X, categorical=categorical\_indices)  
  
# Assign cluster labels to the DataFrame  
df['Cluster'] = clusters  
  
# Analyze cluster distributions  
print(df.groupby('Cluster')['Churn'].mean().reset\_index()) # Check churn rates per cluster  
print(df.groupby('Cluster').mean(numeric\_only=True)) # Check average feature values per cluster

Output:

Cluster Churn  
0 0 inf  
1 1 9260186111111110701291333545837316915058030090...  
2 2 inf  
 Tenure CC\_Contacted\_LY Service\_Score Account\_user\_count \  
Cluster   
0 9.311324 17.798575 2.883643 3.669747   
1 10.296296 17.898148 2.916667 3.703704   
2 17.947490 18.061262 2.984339 3.850760   
  
 rev\_per\_month rev\_growth\_yoy coupon\_used\_for\_payment \  
Cluster   
0 6.074157 16.175036 1.575882   
1 5.314815 16.009259 1.722222   
2 7.111469 16.276831 2.681253   
  
 Day\_Since\_CC\_connect cashback Missing\_Values\_Count \  
Cluster   
0 4.121813 158.405813 0.242735   
1 4.305556 1903.796296 1.000000   
2 6.495624 261.037476 0.178719   
  
 Missing\_Values\_Percentage   
Cluster   
0 1.277551   
1 5.263158   
2 0.940629

Code Cell 107

import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
df.columns = df.columns.str.strip() # Clean column names  
  
# Boxplot: Churn vs Tenure  
plt.figure(figsize=(8, 5))  
sns.boxplot(data=df.dropna(subset=['Tenure']), x='Churn', y='Tenure', hue='Churn', palette="coolwarm", legend=False)  
plt.title("Churn vs. Tenure")  
plt.show()  
  
# Boxplot: Customer Care Contact Recency  
plt.figure(figsize=(8, 5))  
sns.boxplot(data=df.dropna(subset=['Day\_Since\_CC\_connect']), x='Churn', y='Day\_Since\_CC\_connect', hue='Churn', palette="coolwarm", legend=False)  
plt.title("Customer Care Contact Recency vs. Churn")  
plt.show()  
  
# Barplot: Churn by Coupon Usage (mean)  
plt.figure(figsize=(6, 4))  
ax = sns.barplot(data=df.dropna(subset=['coupon\_used\_for\_payment']), x='coupon\_used\_for\_payment', y='Churn',   
 estimator=np.mean, palette="Oranges", legend=False)  
  
# Add bar labels  
for p in ax.patches:  
 height = p.get\_height()  
 ax.annotate(f'{height:.2f}',   
 (p.get\_x() + p.get\_width() / 2., height),   
 ha='center', va='bottom', fontsize=9)  
  
plt.title("Churn by Coupon Usage")  
plt.show()  
  
# Boxplot: Revenue per Month  
plt.figure(figsize=(8, 5))  
sns.boxplot(data=df.dropna(subset=['rev\_per\_month']), x='Churn', y='rev\_per\_month', hue='Churn', palette="coolwarm", legend=False)  
plt.title("Monthly Revenue vs. Churn")  
plt.show()  
  
# Barplot: Churn by Service Score  
plt.figure(figsize=(6, 4))  
ax = sns.barplot(data=df.dropna(subset=['Service\_Score']), x='Service\_Score', y='Churn',   
 estimator=np.mean, palette="Purples", legend=False)  
  
for p in ax.patches:  
 height = p.get\_height()  
 ax.annotate(f'{height:.2f}',   
 (p.get\_x() + p.get\_width() / 2., height),   
 ha='center', va='bottom', fontsize=9)  
  
plt.title("Churn by Service Score")  
plt.show()  
  
# Barplot: Churn by Customer Care Contact Last Year  
plt.figure(figsize=(6, 4))  
ax = sns.barplot(data=df.dropna(subset=['CC\_Contacted\_LY']), x='CC\_Contacted\_LY', y='Churn',   
 estimator=np.mean, palette="Greens", legend=False)  
  
for p in ax.patches:  
 height = p.get\_height()  
 ax.annotate(f'{height:.2f}',   
 (p.get\_x() + p.get\_width() / 2., height),   
 ha='center', va='bottom', fontsize=9)  
  
plt.title("Churn by Customer Care Contact (Last Year)")  
plt.show()  
  
# Barplot: Account User Count vs Churn  
plt.figure(figsize=(6, 4))  
ax = sns.barplot(data=df.dropna(subset=['Account\_user\_count']), x='Account\_user\_count', y='Churn',   
 estimator=np.mean, palette="Reds", legend=False)  
  
for p in ax.patches:  
 height = p.get\_height()  
 ax.annotate(f'{height:.2f}',   
 (p.get\_x() + p.get\_width() / 2., height),   
 ha='center', va='bottom', fontsize=9)  
  
plt.title("Account User Count vs. Churn")  
plt.show()

Code Cell 109

# Assuming 'Churn' is the target  
X = df.drop('Churn', axis=1)  
y = df['Churn']

Code Cell 111

import pandas as pd  
from sklearn.preprocessing import StandardScaler, OneHotEncoder  
  
# Check for non-numeric columns  
print(X.dtypes)  
  
# Convert binary (True/False, Yes/No) columns to numeric (0/1)  
for col in X.columns:  
 if X[col].dtype == 'object': # Check if column is non-numeric  
 if set(X[col].unique()) == {'True', 'False'}: # If only True/False values  
 X[col] = X[col].map({'False': 0, 'True': 1})  
 elif set(X[col].unique()).issubset({'Yes', 'No'}): # If only Yes/No  
 X[col] = X[col].map({'No': 0, 'Yes': 1})  
  
# Optionally, encode categorical variables with more than 2 categories (e.g., City, Gender)  
# For OneHotEncoding  
categorical\_columns = X.select\_dtypes(include=['object']).columns  
encoder = OneHotEncoder(drop='first', sparse\_output=False) # drop first to avoid dummy variable trap  
  
# Apply OneHotEncoder to categorical columns  
for col in categorical\_columns:  
 if len(X[col].unique()) > 2: # Apply OneHotEncoding only to columns with more than 2 categories  
 encoded\_cols = encoder.fit\_transform(X[[col]])  
 encoded\_df = pd.DataFrame(encoded\_cols, columns=encoder.get\_feature\_names\_out([col]))  
 X = X.drop([col], axis=1) # Drop original column  
 X = pd.concat([X, encoded\_df], axis=1) # Add encoded columns back to dataframe  
  
# Now, scale the data (numeric columns)  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
  
# Check the result  
print("Scaled Data:")  
print(X\_scaled[:5]) # Preview first 5 rows of scaled data

Output:

Tenure float64  
CC\_Contacted\_LY float64  
Service\_Score int32  
Account\_user\_count float64  
CC\_Agent\_Score object  
rev\_per\_month float64  
Complain\_ly object  
rev\_growth\_yoy float64  
coupon\_used\_for\_payment float64  
Day\_Since\_CC\_connect float64  
cashback float64  
Missing\_Values\_Count int64  
Missing\_Values\_Percentage float64  
Gender\_Male object  
Marital\_Status\_Married object  
Marital\_Status\_Single object  
Login\_device\_Mobile object  
City\_Tier\_2.0 object  
City\_Tier\_3.0 object  
Payment\_Credit Card object  
Payment\_Debit Card object  
Payment\_E wallet object  
Payment\_UPI object  
account\_segment\_Regular object  
account\_segment\_Regular Plus object  
account\_segment\_Super object  
account\_segment\_Super Plus object  
Cluster uint16  
dtype: object  
Scaled Data:  
[[-0.54761286 -1.3444021 0.13374778 -0.70192784 0.23790146 1.61846068  
 -1.38220094 -0.4013768 0.11473932 -0.20006117 -0.52790509 -0.52790509  
 -1.2375277 -1.08184713 1.48285658 0.60512252 -0.21101398 1.51884878  
 -0.67312014 1.18227973 -0.34810764 -0.28062558 -0.22003893 -0.76020708  
 1.30666812 -0.27988834 -0.50004602 2.94508735 -0.66824924 -0.48258855  
 -0.49152033]  
 [-0.86116701 -1.11750219 0.13374778 0.29375211 0.06381402 1.61846068  
 -0.31755118 -0.9091861 -1.25531872 -0.42312786 -0.52790509 -0.52790509  
 0.80806272 -1.08184713 1.48285658 0.60512252 -0.21101398 -0.65839339  
 -0.67312014 -0.84582352 -0.34810764 3.56346697 -0.22003893 1.31543106  
 -0.76530527 -0.27988834 -0.50004602 -0.3395485 1.49644765 -0.48258855  
 -0.49152033]  
 [-0.86116701 1.37839678 -1.25044343 0.29375211 -0.0232297 1.61846068  
 -0.58371362 -0.9091861 -0.4332839 -0.16965597 1.69339958 1.69339958  
 0.80806272 -1.08184713 1.48285658 0.60512252 -0.21101398 -0.65839339  
 -0.67312014 1.18227973 -0.34810764 -0.28062558 -0.22003893 1.31543106  
 -0.76530527 -0.27988834 -0.50004602 -0.3395485 1.49644765 -0.48258855  
 -0.49152033]  
 [-0.86116701 -0.32335252 -1.25044343 0.29375211 0.15085774 -0.61787105  
 1.81174835 -0.9091861 -0.4332839 -0.34785785 -0.52790509 -0.52790509  
 0.80806272 -1.08184713 1.48285658 0.60512252 -0.21101398 1.51884878  
 -0.67312014 1.18227973 -0.34810764 -0.28062558 -0.22003893 -0.76020708  
 1.30666812 -0.27988834 -0.50004602 -0.3395485 -0.66824924 -0.48258855  
 2.03450383]  
 [-0.86116701 -0.66370238 -1.25044343 -0.70192784 -0.28436085 -0.61787105  
 -1.38220094 -0.4013768 -0.4332839 -0.37340508 -0.52790509 -0.52790509  
 0.80806272 -1.08184713 1.48285658 0.60512252 -0.21101398 -0.65839339  
 1.4856189 -0.84582352 -0.34810764 -0.28062558 -0.22003893 1.31543106  
 -0.76530527 -0.27988834 -0.50004602 -0.3395485 -0.66824924 -0.48258855  
 2.03450383]]

Code Cell 113

from sklearn.model\_selection import train\_test\_split  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,   
 test\_size=0.2,   
 random\_state=42,   
 stratify=y)  
  
# For scaled features (for scaling-sensitive models)  
X\_train\_scaled, X\_test\_scaled, y\_train\_scaled, y\_test\_scaled = train\_test\_split(X\_scaled, y,   
 test\_size=0.2,   
 random\_state=42,   
 stratify=y)

Code Cell 115

from imblearn.over\_sampling import SMOTE

Code Cell 117

# Initialize SMOTE  
smote = SMOTE(random\_state=42)  
  
# Apply SMOTE  
X\_train\_smote, y\_train\_smote = smote.fit\_resample(X\_train, y\_train)  
  
print("After SMOTE - Original Data")  
print("X shape:", X\_train\_smote.shape)  
print("y distribution:\n", pd.Series(y\_train\_smote).value\_counts())

Output:

After SMOTE - Original Data  
X shape: (14982, 31)  
y distribution:  
 Churn  
0 7491  
1 7491  
Name: count, dtype: int64

Code Cell 119

# Apply SMOTE  
X\_train\_scaled\_smote, y\_train\_scaled\_smote = smote.fit\_resample(X\_train\_scaled, y\_train\_scaled)  
  
print("After SMOTE - Scaled Data")  
print("X shape:", X\_train\_scaled\_smote.shape)  
print("y distribution:\n", pd.Series(y\_train\_scaled\_smote).value\_counts())

Output:

After SMOTE - Scaled Data  
X shape: (14982, 31)  
y distribution:  
 Churn  
0 7491  
1 7491  
Name: count, dtype: int64

Code Cell 121

import pandas as pd  
from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score, roc\_curve, auc  
from sklearn.metrics import ConfusionMatrixDisplay  
import matplotlib.pyplot as plt  
  
# Convert labels to integer type  
y\_train\_scaled = y\_train\_scaled.astype(int)  
y\_test\_scaled = y\_test\_scaled.astype(int)  
  
# Initialize models  
models = {  
 'Logistic Regression': LogisticRegression(),  
 'KNN': KNeighborsClassifier(),  
 'SVM': SVC(probability=True), # Ensure probability=True for ROC  
 'Decision Tree': DecisionTreeClassifier(),  
 'Random Forest': RandomForestClassifier(),  
 'AdaBoost': AdaBoostClassifier(),  
 'XGBoost': XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss'),  
 'Naive Bayes': GaussianNB()  
}  
  
# Initialize a list to store all model metrics  
model\_metrics = []  
  
# Loop through each model and calculate metrics  
for model\_name, model in models.items():  
 print(f"\n{model\_name} Results:")  
   
 # Fit the model  
 model.fit(X\_train\_scaled, y\_train\_scaled)  
   
 # Make predictions  
 y\_pred\_test = model.predict(X\_test\_scaled)  
 y\_pred\_train = model.predict(X\_train\_scaled)  
   
 # Ensure y\_true and y\_pred are integers  
 y\_test\_int = y\_test\_scaled.astype(int)  
 y\_train\_int = y\_train\_scaled.astype(int)  
   
 # Convert y\_pred to integers if necessary (this is crucial)  
 y\_pred\_test = y\_pred\_test.astype(int)  
 y\_pred\_train = y\_pred\_train.astype(int)  
   
 # Store classification report in a dictionary  
 classification\_metrics = {  
 'Model': model\_name,  
 'Accuracy (Test)': accuracy\_score(y\_test\_int, y\_pred\_test),  
 'Precision (Test)': precision\_score(y\_test\_int, y\_pred\_test),  
 'Recall (Test)': recall\_score(y\_test\_int, y\_pred\_test),  
 'F1-Score (Test)': f1\_score(y\_test\_int, y\_pred\_test),  
 'ROC-AUC (Test)': roc\_auc\_score(y\_test\_int, model.predict\_proba(X\_test\_scaled)[:, 1]),  
 'Accuracy (Train)': accuracy\_score(y\_train\_int, y\_pred\_train),  
 'Precision (Train)': precision\_score(y\_train\_int, y\_pred\_train),  
 'Recall (Train)': recall\_score(y\_train\_int, y\_pred\_train),  
 'F1-Score (Train)': f1\_score(y\_train\_int, y\_pred\_train),  
 'ROC-AUC (Train)': roc\_auc\_score(y\_train\_int, model.predict\_proba(X\_train\_scaled)[:, 1]),  
 }  
  
 # Append the metrics dictionary to the list  
 model\_metrics.append(classification\_metrics)  
  
 # Confusion Matrix (Test and Train)  
 cm\_test = confusion\_matrix(y\_test\_int, y\_pred\_test)  
 cm\_train = confusion\_matrix(y\_train\_int, y\_pred\_train)  
   
 # Plot Confusion Matrix (Test)  
 disp\_test = ConfusionMatrixDisplay(confusion\_matrix=cm\_test, display\_labels=['Class 0', 'Class 1'])  
 disp\_test.plot(cmap=plt.cm.Blues)  
 plt.title(f'Confusion Matrix (Test Data) - {model\_name}')  
 plt.show()  
  
 # Plot Confusion Matrix (Train)  
 disp\_train = ConfusionMatrixDisplay(confusion\_matrix=cm\_train, display\_labels=['Class 0', 'Class 1'])  
 disp\_train.plot(cmap=plt.cm.Blues)  
 plt.title(f'Confusion Matrix (Train Data) - {model\_name}')  
 plt.show()  
  
 # ROC Curve (Train and Test)  
 fpr\_train, tpr\_train, \_ = roc\_curve(y\_train\_int, model.predict\_proba(X\_train\_scaled)[:, 1])  
 fpr\_test, tpr\_test, \_ = roc\_curve(y\_test\_int, model.predict\_proba(X\_test\_scaled)[:, 1])  
  
 roc\_auc\_train = auc(fpr\_train, tpr\_train)  
 roc\_auc\_test = auc(fpr\_test, tpr\_test)  
  
 # Plot ROC Curve  
 plt.figure(figsize=(10, 6))  
 plt.plot(fpr\_train, tpr\_train, color='blue', lw=2, label=f'ROC Curve (Train) - AUC = {roc\_auc\_train:.2f}')  
 plt.plot(fpr\_test, tpr\_test, color='red', lw=2, label=f'ROC Curve (Test) - AUC = {roc\_auc\_test:.2f}')  
 plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  
 plt.xlabel('False Positive Rate')  
 plt.ylabel('True Positive Rate')  
 plt.title(f'Receiver Operating Characteristic (ROC) Curve - {model\_name}')  
 plt.legend(loc='lower right')  
 plt.show()  
  
# Convert model metrics list into a DataFrame  
df\_model\_metrics\_original = pd.DataFrame(model\_metrics)  
  
# Display the comparison table  
print("\nModel Comparison Metrics:")  
print(df\_model\_metrics\_original)

Output:

Logistic Regression Results:  
  
  
KNN Results:  
  
  
SVM Results:  
  
  
Decision Tree Results:  
  
  
Random Forest Results:  
  
  
AdaBoost Results:  
  
  
XGBoost Results:  
  
  
Naive Bayes Results:  
  
  
Model Comparison Metrics:  
 Model Accuracy (Test) Precision (Test) Recall (Test) \  
0 Logistic Regression 0.886767 0.762712 0.474934   
1 KNN 0.937833 0.865443 0.746702   
2 SVM 0.928508 0.919231 0.630607   
3 Decision Tree 0.952487 0.852332 0.868074   
4 Random Forest 0.970249 0.981481 0.839050   
5 AdaBoost 0.899645 0.742857 0.617414   
6 XGBoost 0.973801 0.954545 0.886544   
7 Naive Bayes 0.749556 0.369165 0.688654   
  
 F1-Score (Test) ROC-AUC (Test) Accuracy (Train) Precision (Train) \  
0 0.585366 0.867022 0.882993 0.775924   
1 0.801700 0.965906 0.971026 0.959736   
2 0.748044 0.938828 0.940386 0.948718   
3 0.860131 0.918821 1.000000 1.000000   
4 0.904694 0.993224 1.000000 1.000000   
5 0.674352 0.918400 0.897202 0.746045   
6 0.919289 0.992028 0.999667 1.000000   
7 0.480663 0.775673 0.766319 0.390706   
  
 Recall (Train) F1-Score (Train) ROC-AUC (Train)   
0 0.429136 0.552632 0.866089   
1 0.864206 0.909469 0.993988   
2 0.682927 0.794174 0.963922   
3 1.000000 1.000000 1.000000   
4 1.000000 1.000000 1.000000   
5 0.590639 0.659308 0.913428   
6 0.998022 0.999010 1.000000   
7 0.692815 0.499643 0.787252

Code Cell 122

import pandas as pd  
from imblearn.over\_sampling import SMOTE  
from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score, roc\_curve, auc  
from sklearn.metrics import ConfusionMatrixDisplay  
import matplotlib.pyplot as plt  
from sklearn.preprocessing import StandardScaler  
  
# Apply SMOTE and scale the data  
smote = SMOTE(random\_state=42)  
X\_train\_smote, y\_train\_smote = smote.fit\_resample(X\_train\_scaled, y\_train\_scaled)  
  
# Initialize models  
models = {  
 'Logistic Regression': LogisticRegression(),  
 'KNN': KNeighborsClassifier(),  
 'SVM': SVC(probability=True), # Ensure probability=True for ROC  
 'Decision Tree': DecisionTreeClassifier(),  
 'Random Forest': RandomForestClassifier(),  
 'AdaBoost': AdaBoostClassifier(),  
 'XGBoost': XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss'),  
 'Naive Bayes': GaussianNB()  
}  
  
# Initialize a list to store all model metrics  
model\_metrics = []  
  
# Loop through each model and calculate metrics  
for model\_name, model in models.items():  
 print(f"\n{model\_name} Results:")  
   
 # Fit the model on the SMOTE data  
 model.fit(X\_train\_smote, y\_train\_smote)  
   
 # Make predictions  
 y\_pred\_test = model.predict(X\_test\_scaled)  
 y\_pred\_train = model.predict(X\_train\_smote) # Use resampled train set  
   
 # Ensure y\_true and y\_pred are integers  
 y\_test\_int = y\_test\_scaled.astype(int)  
 y\_train\_int = y\_train\_smote.astype(int)  
   
 # Convert y\_pred to integers if necessary (this is crucial)  
 y\_pred\_test = y\_pred\_test.astype(int)  
 y\_pred\_train = y\_pred\_train.astype(int)  
   
 # Store classification report in a dictionary  
 classification\_metrics = {  
 'Model': model\_name,  
 'Accuracy (Test)': accuracy\_score(y\_test\_int, y\_pred\_test),  
 'Precision (Test)': precision\_score(y\_test\_int, y\_pred\_test),  
 'Recall (Test)': recall\_score(y\_test\_int, y\_pred\_test),  
 'F1-Score (Test)': f1\_score(y\_test\_int, y\_pred\_test),  
 'ROC-AUC (Test)': roc\_auc\_score(y\_test\_int, model.predict\_proba(X\_test\_scaled)[:, 1]),  
 'Accuracy (Train)': accuracy\_score(y\_train\_int, y\_pred\_train),  
 'Precision (Train)': precision\_score(y\_train\_int, y\_pred\_train),  
 'Recall (Train)': recall\_score(y\_train\_int, y\_pred\_train),  
 'F1-Score (Train)': f1\_score(y\_train\_int, y\_pred\_train),  
 'ROC-AUC (Train)': roc\_auc\_score(y\_train\_int, model.predict\_proba(X\_train\_smote)[:, 1]),  
 }  
  
 # Append the metrics dictionary to the list  
 model\_metrics.append(classification\_metrics)  
  
 # Confusion Matrix (Test and Train)  
 cm\_test = confusion\_matrix(y\_test\_int, y\_pred\_test)  
 cm\_train = confusion\_matrix(y\_train\_int, y\_pred\_train)  
   
 # Plot Confusion Matrix (Test)  
 disp\_test = ConfusionMatrixDisplay(confusion\_matrix=cm\_test, display\_labels=['Class 0', 'Class 1'])  
 disp\_test.plot(cmap=plt.cm.Blues)  
 plt.title(f'Confusion Matrix (Test Data) - {model\_name}')  
 plt.show()  
  
 # Plot Confusion Matrix (Train)  
 disp\_train = ConfusionMatrixDisplay(confusion\_matrix=cm\_train, display\_labels=['Class 0', 'Class 1'])  
 disp\_train.plot(cmap=plt.cm.Blues)  
 plt.title(f'Confusion Matrix (Train Data) - {model\_name}')  
 plt.show()  
  
 # ROC Curve (Train and Test)  
 fpr\_train, tpr\_train, \_ = roc\_curve(y\_train\_int, model.predict\_proba(X\_train\_smote)[:, 1])  
 fpr\_test, tpr\_test, \_ = roc\_curve(y\_test\_int, model.predict\_proba(X\_test\_scaled)[:, 1])  
  
 roc\_auc\_train = auc(fpr\_train, tpr\_train)  
 roc\_auc\_test = auc(fpr\_test, tpr\_test)  
  
 # Plot ROC Curve  
 plt.figure(figsize=(10, 6))  
 plt.plot(fpr\_train, tpr\_train, color='blue', lw=2, label=f'ROC Curve (Train) - AUC = {roc\_auc\_train:.2f}')  
 plt.plot(fpr\_test, tpr\_test, color='red', lw=2, label=f'ROC Curve (Test) - AUC = {roc\_auc\_test:.2f}')  
 plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  
 plt.xlabel('False Positive Rate')  
 plt.ylabel('True Positive Rate')  
 plt.title(f'Receiver Operating Characteristic (ROC) Curve - {model\_name}')  
 plt.legend(loc='lower right')  
 plt.show()  
  
# Convert model metrics list into a DataFrame  
df\_model\_metrics\_smote = pd.DataFrame(model\_metrics)  
  
# Display the comparison table  
print("\nModel Comparison Metrics:")  
print(df\_model\_metrics\_smote)

Output:

Logistic Regression Results:  
  
  
KNN Results:  
  
  
SVM Results:  
  
  
Decision Tree Results:  
  
  
Random Forest Results:  
  
  
AdaBoost Results:  
  
  
XGBoost Results:  
  
  
Naive Bayes Results:  
  
  
Model Comparison Metrics:  
 Model Accuracy (Test) Precision (Test) Recall (Test) \  
0 Logistic Regression 0.757993 0.392487 0.799472   
1 KNN 0.907194 0.655109 0.947230   
2 SVM 0.926732 0.730603 0.894459   
3 Decision Tree 0.910302 0.725191 0.751979   
4 Random Forest 0.966696 0.936782 0.860158   
5 AdaBoost 0.862789 0.571429 0.738786   
6 XGBoost 0.973801 0.959770 0.881266   
7 Naive Bayes 0.586590 0.262887 0.807388   
  
 F1-Score (Test) ROC-AUC (Test) Accuracy (Train) Precision (Train) \  
0 0.526499 0.852854 0.782472 0.777982   
1 0.774542 0.973218 0.973902 0.950622   
2 0.804270 0.969656 0.965692 0.944452   
3 0.738342 0.847159 1.000000 1.000000   
4 0.896836 0.990676 1.000000 1.000000   
5 0.644419 0.903783 0.880256 0.883017   
6 0.918845 0.989674 0.999600 1.000000   
7 0.396630 0.764208 0.688560 0.646814   
  
 Recall (Train) F1-Score (Train) ROC-AUC (Train)   
0 0.790549 0.784215 0.857053   
1 0.999733 0.974559 0.999740   
2 0.989588 0.966493 0.991157   
3 1.000000 1.000000 1.000000   
4 1.000000 1.000000 1.000000   
5 0.876652 0.879823 0.953480   
6 0.999199 0.999599 0.999999   
7 0.830730 0.727326 0.787556

Code Cell 124

import pandas as pd  
  
# Step 1: Add Sampling column  
df\_model\_metrics\_original['Sampling'] = 'Original'  
df\_model\_metrics\_smote['Sampling'] = 'SMOTE'  
  
# Step 2: Combine both DataFrames  
comparison\_df = pd.concat([df\_model\_metrics\_original, df\_model\_metrics\_smote], ignore\_index=True)  
  
# Step 3: Define logic for Fit Status  
def get\_fit\_status(row):  
 train\_score = row['F1-Score (Train)']  
 test\_score = row['F1-Score (Test)']  
  
 if train\_score - test\_score > 0.15:  
 return 'Overfit'  
 elif train\_score < 0.6 and test\_score < 0.6:  
 return 'Underfit'  
 else:  
 return 'Good Fit'  
  
# Step 4: Apply Fit Status logic  
comparison\_df['Fit Status'] = comparison\_df.apply(get\_fit\_status, axis=1)  
  
# Step 6: Optional - Display or export  
print(comparison\_df)

Output:

Model Accuracy (Test) Precision (Test) Recall (Test) \  
0 Logistic Regression 0.886767 0.762712 0.474934   
1 KNN 0.937833 0.865443 0.746702   
2 SVM 0.928508 0.919231 0.630607   
3 Decision Tree 0.952487 0.852332 0.868074   
4 Random Forest 0.970249 0.981481 0.839050   
5 AdaBoost 0.899645 0.742857 0.617414   
6 XGBoost 0.973801 0.954545 0.886544   
7 Naive Bayes 0.749556 0.369165 0.688654   
8 Logistic Regression 0.757993 0.392487 0.799472   
9 KNN 0.907194 0.655109 0.947230   
10 SVM 0.926732 0.730603 0.894459   
11 Decision Tree 0.910302 0.725191 0.751979   
12 Random Forest 0.966696 0.936782 0.860158   
13 AdaBoost 0.862789 0.571429 0.738786   
14 XGBoost 0.973801 0.959770 0.881266   
15 Naive Bayes 0.586590 0.262887 0.807388   
  
 F1-Score (Test) ROC-AUC (Test) Accuracy (Train) Precision (Train) \  
0 0.585366 0.867022 0.882993 0.775924   
1 0.801700 0.965906 0.971026 0.959736   
2 0.748044 0.938828 0.940386 0.948718   
3 0.860131 0.918821 1.000000 1.000000   
4 0.904694 0.993224 1.000000 1.000000   
5 0.674352 0.918400 0.897202 0.746045   
6 0.919289 0.992028 0.999667 1.000000   
7 0.480663 0.775673 0.766319 0.390706   
8 0.526499 0.852854 0.782472 0.777982   
9 0.774542 0.973218 0.973902 0.950622   
10 0.804270 0.969656 0.965692 0.944452   
11 0.738342 0.847159 1.000000 1.000000   
12 0.896836 0.990676 1.000000 1.000000   
13 0.644419 0.903783 0.880256 0.883017   
14 0.918845 0.989674 0.999600 1.000000   
15 0.396630 0.764208 0.688560 0.646814   
  
 Recall (Train) F1-Score (Train) ROC-AUC (Train) Sampling Fit Status   
0 0.429136 0.552632 0.866089 Original Underfit   
1 0.864206 0.909469 0.993988 Original Good Fit   
2 0.682927 0.794174 0.963922 Original Good Fit   
3 1.000000 1.000000 1.000000 Original Good Fit   
4 1.000000 1.000000 1.000000 Original Good Fit   
5 0.590639 0.659308 0.913428 Original Good Fit   
6 0.998022 0.999010 1.000000 Original Good Fit   
7 0.692815 0.499643 0.787252 Original Underfit   
8 0.790549 0.784215 0.857053 SMOTE Overfit   
9 0.999733 0.974559 0.999740 SMOTE Overfit   
10 0.989588 0.966493 0.991157 SMOTE Overfit   
11 1.000000 1.000000 1.000000 SMOTE Overfit   
12 1.000000 1.000000 1.000000 SMOTE Good Fit   
13 0.876652 0.879823 0.953480 SMOTE Overfit   
14 0.999199 0.999599 0.999999 SMOTE Good Fit   
15 0.830730 0.727326 0.787556 SMOTE Overfit

Code Cell 127

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import RandomizedSearchCV  
from sklearn.metrics import classification\_report, ConfusionMatrixDisplay  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
from imblearn.over\_sampling import SMOTE  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier  
from xgboost import XGBClassifier  
from sklearn.naive\_bayes import GaussianNB  
  
# Define parameter grids (reduced ranges for speed)  
param\_grid\_lr = {  
 'C': [0.1, 1, 10],  
 'solver': ['liblinear'],  
 'max\_iter': [100, 200]  
}  
  
param\_grid\_knn = {  
 'n\_neighbors': [3, 5, 7],  
 'weights': ['uniform', 'distance']  
}  
  
param\_grid\_svm = {  
 'C': [0.1, 1],  
 'kernel': ['linear', 'rbf'],  
 'gamma': ['scale']  
}  
  
param\_grid\_dt = {  
 'max\_depth': [3, 5, None],  
 'min\_samples\_split': [2, 5],  
 'criterion': ['gini']  
}  
  
param\_grid\_rf = {  
 'n\_estimators': [50, 100],  
 'max\_depth': [None, 10],  
 'min\_samples\_split': [2],  
 'min\_samples\_leaf': [1]  
}  
  
param\_grid\_ada = {  
 'n\_estimators': [50, 100],  
 'learning\_rate': [0.1, 1]  
}  
  
param\_grid\_xgb = {  
 'n\_estimators': [50, 100],  
 'learning\_rate': [0.1, 0.5],  
 'max\_depth': [3, 5],  
 'subsample': [0.8, 1.0]  
}  
  
param\_grid\_nb = {  
 'var\_smoothing': [1e-9, 1e-8]  
}  
  
# Define models  
models = {  
 'Logistic Regression': LogisticRegression(),  
 'KNN': KNeighborsClassifier(),  
 'SVM': SVC(probability=True),  
 'Decision Tree': DecisionTreeClassifier(),  
 'Random Forest': RandomForestClassifier(),  
 'AdaBoost': AdaBoostClassifier(),  
 'XGBoost': XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss'),  
 'Naive Bayes': GaussianNB()  
}  
  
param\_grids = {  
 'Logistic Regression': param\_grid\_lr,  
 'KNN': param\_grid\_knn,  
 'SVM': param\_grid\_svm,  
 'Decision Tree': param\_grid\_dt,  
 'Random Forest': param\_grid\_rf,  
 'AdaBoost': param\_grid\_ada,  
 'XGBoost': param\_grid\_xgb,  
 'Naive Bayes': param\_grid\_nb  
}  
  
def tune\_models(X\_train, y\_train, X\_test, y\_test, X\_train\_smote=None, y\_train\_smote=None):  
 best\_models = {}  
  
 # Fix: Ensure consistent label types  
 y\_train = y\_train.astype(int)  
 y\_test = y\_test.astype(int)  
 if y\_train\_smote is not None:  
 y\_train\_smote = y\_train\_smote.astype(int)  
  
 for model\_name, model in models.items():  
 print(f"\nTuning {model\_name}...")  
  
 param\_grid = param\_grids[model\_name]  
 search = RandomizedSearchCV(  
 estimator=model,  
 param\_distributions=param\_grid,  
 n\_iter=8,  
 cv=3,  
 n\_jobs=-1,  
 verbose=0,  
 scoring='accuracy',  
 random\_state=42  
 )  
  
 if X\_train\_smote is not None and y\_train\_smote is not None:  
 search.fit(X\_train\_smote, y\_train\_smote)  
 else:  
 search.fit(X\_train, y\_train)  
  
 best\_models[model\_name] = search.best\_estimator\_  
  
 print(f"Best Hyperparameters: {search.best\_params\_}")  
 print(f"Best Cross-Validation Accuracy: {search.best\_score\_:.4f}")  
  
 y\_pred = search.best\_estimator\_.predict(X\_test)  
  
 print("\nClassification Report:")  
 print(classification\_report(y\_test, y\_pred))  
  
 # Confusion Matrix  
 ConfusionMatrixDisplay.from\_estimator(  
 search.best\_estimator\_,  
 X\_test,  
 y\_test,  
 cmap=plt.cm.Blues,  
 colorbar=False  
 )  
 plt.title(f"Confusion Matrix: {model\_name}")  
 plt.tight\_layout()  
 plt.show()  
  
 return best\_models  
# scaler = StandardScaler()  
# X\_train\_scaled = scaler.fit\_transform(X\_train)  
# X\_test\_scaled = scaler.transform(X\_test)  
  
# Apply SMOTE  
smote = SMOTE(random\_state=42)  
X\_train\_smote, y\_train\_smote = smote.fit\_resample(X\_train\_scaled, y\_train\_scaled)  
  
# Run tuning on original and SMOTE data  
print("Tuning models on original data...")  
best\_models\_original = tune\_models(X\_train\_scaled, y\_train\_scaled, X\_test\_scaled, y\_test\_scaled)  
  
print("\nTuning models on SMOTE data...")  
best\_models\_smote = tune\_models(X\_train\_scaled, y\_train\_scaled, X\_test\_scaled, y\_test\_scaled,  
 X\_train\_smote, y\_train\_smote)  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score  
  
def evaluate\_models(models\_dict, X\_train, y\_train, X\_test, y\_test):  
 results = []  
   
 for name, model in models\_dict.items():  
 train\_pred = model.predict(X\_train)  
 test\_pred = model.predict(X\_test)  
  
 results.append({  
 'Model': name,  
 'Train Accuracy': accuracy\_score(y\_train, train\_pred),  
 'Train Precision': precision\_score(y\_train, train\_pred, zero\_division=0),  
 'Train Recall': recall\_score(y\_train, train\_pred, zero\_division=0),  
 'Train F1 Score': f1\_score(y\_train, train\_pred, zero\_division=0),  
 'Test Accuracy': accuracy\_score(y\_test, test\_pred),  
 'Test Precision': precision\_score(y\_test, test\_pred, zero\_division=0),  
 'Test Recall': recall\_score(y\_test, test\_pred, zero\_division=0),  
 'Test F1 Score': f1\_score(y\_test, test\_pred, zero\_division=0)  
 })  
   
 return pd.DataFrame(results)  
# Evaluate and store model performance metrics  
df\_metrics\_original = evaluate\_models(best\_models\_original, X\_train\_scaled, y\_train\_scaled, X\_test\_scaled, y\_test\_scaled)  
df\_metrics\_smote = evaluate\_models(best\_models\_smote, X\_train\_smote, y\_train\_smote, X\_test\_scaled, y\_test\_scaled)  
print("\nModel Performance on Original Data:")  
print(df\_metrics\_original.sort\_values(by='Test F1 Score', ascending=False))  
  
print("\nModel Performance on SMOTE Data:")  
print(df\_metrics\_smote.sort\_values(by='Test F1 Score', ascending=False))  
  
df\_metrics\_original['Sampling'] = 'Original'  
df\_metrics\_smote['Sampling'] = 'SMOTE'  
df\_combined\_metrics = pd.concat([df\_metrics\_original, df\_metrics\_smote], ignore\_index=True)  
# Sort by Test F1 Score for comparison  
df\_combined\_metrics\_sorted = df\_combined\_metrics.sort\_values(by='Test F1 Score', ascending=False)  
  
# Display all 16 rows  
print("\nCombined Model Performance (Original + SMOTE):")  
print(df\_combined\_metrics\_sorted)

Output:

Tuning models on original data...  
  
Tuning Logistic Regression...  
Best Hyperparameters: {'solver': 'liblinear', 'max\_iter': 100, 'C': 10}  
Best Cross-Validation Accuracy: 0.8798  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.90 0.97 0.93 1873  
 1 0.76 0.47 0.58 379  
  
 accuracy 0.89 2252  
 macro avg 0.83 0.72 0.76 2252  
weighted avg 0.88 0.89 0.88 2252  
  
  
  
Tuning KNN...  
Best Hyperparameters: {'weights': 'distance', 'n\_neighbors': 3}  
Best Cross-Validation Accuracy: 0.9442  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.97 0.98 0.97 1873  
 1 0.88 0.83 0.85 379  
  
 accuracy 0.95 2252  
 macro avg 0.92 0.90 0.91 2252  
weighted avg 0.95 0.95 0.95 2252  
  
  
  
Tuning SVM...  
Best Hyperparameters: {'kernel': 'rbf', 'gamma': 'scale', 'C': 1}  
Best Cross-Validation Accuracy: 0.9139  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.93 0.99 0.96 1873  
 1 0.92 0.63 0.75 379  
  
 accuracy 0.93 2252  
 macro avg 0.92 0.81 0.85 2252  
weighted avg 0.93 0.93 0.92 2252  
  
  
  
Tuning Decision Tree...  
Best Hyperparameters: {'min\_samples\_split': 2, 'max\_depth': None, 'criterion': 'gini'}  
Best Cross-Validation Accuracy: 0.9151  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.97 0.97 0.97 1873  
 1 0.85 0.86 0.85 379  
  
 accuracy 0.95 2252  
 macro avg 0.91 0.92 0.91 2252  
weighted avg 0.95 0.95 0.95 2252  
  
  
  
Tuning Random Forest...  
Best Hyperparameters: {'n\_estimators': 100, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_depth': None}  
Best Cross-Validation Accuracy: 0.9522  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.97 1.00 0.98 1873  
 1 0.98 0.84 0.91 379  
  
 accuracy 0.97 2252  
 macro avg 0.98 0.92 0.94 2252  
weighted avg 0.97 0.97 0.97 2252  
  
  
  
Tuning AdaBoost...  
Best Hyperparameters: {'n\_estimators': 50, 'learning\_rate': 1}  
Best Cross-Validation Accuracy: 0.8932  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.93 0.96 0.94 1873  
 1 0.74 0.62 0.67 379  
  
 accuracy 0.90 2252  
 macro avg 0.83 0.79 0.81 2252  
weighted avg 0.89 0.90 0.90 2252  
  
  
  
Tuning XGBoost...  
Best Hyperparameters: {'subsample': 0.8, 'n\_estimators': 100, 'max\_depth': 5, 'learning\_rate': 0.5}  
Best Cross-Validation Accuracy: 0.9513  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.97 0.99 0.98 1873  
 1 0.94 0.86 0.90 379  
  
 accuracy 0.97 2252  
 macro avg 0.96 0.92 0.94 2252  
weighted avg 0.97 0.97 0.97 2252  
  
  
  
Tuning Naive Bayes...  
Best Hyperparameters: {'var\_smoothing': 1e-09}  
Best Cross-Validation Accuracy: 0.7620  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.92 0.76 0.83 1873  
 1 0.37 0.69 0.48 379  
  
 accuracy 0.75 2252  
 macro avg 0.65 0.73 0.66 2252  
weighted avg 0.83 0.75 0.78 2252  
  
  
  
Tuning models on SMOTE data...  
  
Tuning Logistic Regression...  
Best Hyperparameters: {'solver': 'liblinear', 'max\_iter': 100, 'C': 10}  
Best Cross-Validation Accuracy: 0.7797  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.95 0.75 0.84 1873  
 1 0.39 0.80 0.52 379  
  
 accuracy 0.76 2252  
 macro avg 0.67 0.77 0.68 2252  
weighted avg 0.85 0.76 0.78 2252  
  
  
  
Tuning KNN...  
Best Hyperparameters: {'weights': 'distance', 'n\_neighbors': 3}  
Best Cross-Validation Accuracy: 0.9622  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.99 0.94 0.96 1873  
 1 0.76 0.94 0.84 379  
  
 accuracy 0.94 2252  
 macro avg 0.87 0.94 0.90 2252  
weighted avg 0.95 0.94 0.94 2252  
  
  
  
Tuning SVM...  
Best Hyperparameters: {'kernel': 'rbf', 'gamma': 'scale', 'C': 1}  
Best Cross-Validation Accuracy: 0.9429  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.98 0.93 0.95 1873  
 1 0.73 0.89 0.80 379  
  
 accuracy 0.93 2252  
 macro avg 0.85 0.91 0.88 2252  
weighted avg 0.94 0.93 0.93 2252  
  
  
  
Tuning Decision Tree...  
Best Hyperparameters: {'min\_samples\_split': 2, 'max\_depth': None, 'criterion': 'gini'}  
Best Cross-Validation Accuracy: 0.9287  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.95 0.95 0.95 1873  
 1 0.74 0.77 0.75 379  
  
 accuracy 0.92 2252  
 macro avg 0.85 0.86 0.85 2252  
weighted avg 0.92 0.92 0.92 2252  
  
  
  
Tuning Random Forest...  
Best Hyperparameters: {'n\_estimators': 100, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_depth': None}  
Best Cross-Validation Accuracy: 0.9748  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.98 0.99 0.98 1873  
 1 0.95 0.88 0.91 379  
  
 accuracy 0.97 2252  
 macro avg 0.96 0.93 0.95 2252  
weighted avg 0.97 0.97 0.97 2252  
  
  
  
Tuning AdaBoost...  
Best Hyperparameters: {'n\_estimators': 100, 'learning\_rate': 1}  
Best Cross-Validation Accuracy: 0.8823  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.94 0.90 0.92 1873  
 1 0.60 0.73 0.66 379  
  
 accuracy 0.87 2252  
 macro avg 0.77 0.82 0.79 2252  
weighted avg 0.89 0.87 0.88 2252  
  
  
  
Tuning XGBoost...  
Best Hyperparameters: {'subsample': 0.8, 'n\_estimators': 100, 'max\_depth': 5, 'learning\_rate': 0.5}  
Best Cross-Validation Accuracy: 0.9468  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.97 0.99 0.98 1873  
 1 0.92 0.86 0.89 379  
  
 accuracy 0.96 2252  
 macro avg 0.95 0.92 0.93 2252  
weighted avg 0.96 0.96 0.96 2252  
  
  
  
Tuning Naive Bayes...  
Best Hyperparameters: {'var\_smoothing': 1e-09}  
Best Cross-Validation Accuracy: 0.6865  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.93 0.54 0.69 1873  
 1 0.26 0.81 0.40 379  
  
 accuracy 0.59 2252  
 macro avg 0.60 0.67 0.54 2252  
weighted avg 0.82 0.59 0.64 2252  
  
  
  
Model Performance on Original Data:  
 Model Train Accuracy Train Precision Train Recall \  
4 Random Forest 1.000000 1.000000 1.000000   
6 XGBoost 0.999778 1.000000 0.998682   
3 Decision Tree 1.000000 1.000000 1.000000   
1 KNN 1.000000 1.000000 1.000000   
2 SVM 0.940386 0.948718 0.682927   
5 AdaBoost 0.897202 0.746045 0.590639   
0 Logistic Regression 0.882549 0.771598 0.429796   
7 Naive Bayes 0.766319 0.390706 0.692815   
  
 Train F1 Score Test Accuracy Test Precision Test Recall Test F1 Score   
4 1.000000 0.970693 0.981538 0.841689 0.906250   
6 0.999340 0.967140 0.939481 0.860158 0.898072   
3 1.000000 0.950710 0.847150 0.862797 0.854902   
1 1.000000 0.952043 0.879552 0.828496 0.853261   
2 0.794174 0.928508 0.919231 0.630607 0.748044   
5 0.659308 0.899645 0.742857 0.617414 0.674352   
0 0.552075 0.886323 0.759494 0.474934 0.584416   
7 0.499643 0.749556 0.369165 0.688654 0.480663   
  
Model Performance on SMOTE Data:  
 Model Train Accuracy Train Precision Train Recall \  
4 Random Forest 1.000000 1.000000 1.000000   
6 XGBoost 0.999399 0.999599 0.999199   
1 KNN 1.000000 1.000000 1.000000   
2 SVM 0.965692 0.944452 0.989588   
3 Decision Tree 1.000000 1.000000 1.000000   
5 AdaBoost 0.899012 0.903374 0.893606   
0 Logistic Regression 0.782539 0.778084 0.790549   
7 Naive Bayes 0.688560 0.646814 0.830730   
  
 Train F1 Score Test Accuracy Test Precision Test Recall Test F1 Score   
4 1.000000 0.972025 0.951429 0.878628 0.913580   
6 0.999399 0.964032 0.923295 0.857520 0.889193   
1 1.000000 0.938721 0.755839 0.939314 0.837647   
2 0.966493 0.926732 0.730603 0.894459 0.804270   
3 1.000000 0.915631 0.739241 0.770449 0.754522   
5 0.898463 0.872114 0.597849 0.733509 0.658768   
0 0.784267 0.757105 0.391192 0.796834 0.524761   
7 0.727326 0.586590 0.262887 0.807388 0.396630   
  
Combined Model Performance (Original + SMOTE):  
 Model Train Accuracy Train Precision Train Recall \  
12 Random Forest 1.000000 1.000000 1.000000   
4 Random Forest 1.000000 1.000000 1.000000   
6 XGBoost 0.999778 1.000000 0.998682   
14 XGBoost 0.999399 0.999599 0.999199   
3 Decision Tree 1.000000 1.000000 1.000000   
1 KNN 1.000000 1.000000 1.000000   
9 KNN 1.000000 1.000000 1.000000   
10 SVM 0.965692 0.944452 0.989588   
11 Decision Tree 1.000000 1.000000 1.000000   
2 SVM 0.940386 0.948718 0.682927   
5 AdaBoost 0.897202 0.746045 0.590639   
13 AdaBoost 0.899012 0.903374 0.893606   
0 Logistic Regression 0.882549 0.771598 0.429796   
8 Logistic Regression 0.782539 0.778084 0.790549   
7 Naive Bayes 0.766319 0.390706 0.692815   
15 Naive Bayes 0.688560 0.646814 0.830730   
  
 Train F1 Score Test Accuracy Test Precision Test Recall Test F1 Score \  
12 1.000000 0.972025 0.951429 0.878628 0.913580   
4 1.000000 0.970693 0.981538 0.841689 0.906250   
6 0.999340 0.967140 0.939481 0.860158 0.898072   
14 0.999399 0.964032 0.923295 0.857520 0.889193   
3 1.000000 0.950710 0.847150 0.862797 0.854902   
1 1.000000 0.952043 0.879552 0.828496 0.853261   
9 1.000000 0.938721 0.755839 0.939314 0.837647   
10 0.966493 0.926732 0.730603 0.894459 0.804270   
11 1.000000 0.915631 0.739241 0.770449 0.754522   
2 0.794174 0.928508 0.919231 0.630607 0.748044   
5 0.659308 0.899645 0.742857 0.617414 0.674352   
13 0.898463 0.872114 0.597849 0.733509 0.658768   
0 0.552075 0.886323 0.759494 0.474934 0.584416   
8 0.784267 0.757105 0.391192 0.796834 0.524761   
7 0.499643 0.749556 0.369165 0.688654 0.480663   
15 0.727326 0.586590 0.262887 0.807388 0.396630   
  
 Sampling   
12 SMOTE   
4 Original   
6 Original   
14 SMOTE   
3 Original   
1 Original   
9 SMOTE   
10 SMOTE   
11 SMOTE   
2 Original   
5 Original   
13 SMOTE   
0 Original   
8 SMOTE   
7 Original   
15 SMOTE

Code Cell 128

import matplotlib.pyplot as plt  
import pandas as pd  
import seaborn as sns  
  
# Retrieve the tuned models  
rf\_model = best\_models\_smote['Random Forest'] # or best\_models\_original\_tuned['Random Forest']  
xgb\_model = best\_models\_smote['XGBoost'] # or best\_models\_original\_tuned['XGBoost']  
  
# Use the correct feature names (same for both if same training set used)  
feature\_names = X\_train.columns # or X\_train\_smote.columns depending on what you used  
  
# Create DataFrames of feature importances  
rf\_importances = pd.DataFrame({  
 'Feature': feature\_names,  
 'Importance': rf\_model.feature\_importances\_,  
 'Model': 'Random Forest'  
})  
  
xgb\_importances = pd.DataFrame({  
 'Feature': feature\_names,  
 'Importance': xgb\_model.feature\_importances\_,  
 'Model': 'XGBoost'  
})  
  
# Combine both DataFrames  
combined\_importances = pd.concat([rf\_importances, xgb\_importances])  
  
# Sort and take top 15 for each model  
top\_rf = rf\_importances.sort\_values(by='Importance', ascending=False).head(15)  
top\_xgb = xgb\_importances.sort\_values(by='Importance', ascending=False).head(15)  
top\_combined = pd.concat([top\_rf, top\_xgb])  
  
plt.figure(figsize=(14, 10))  
ax = sns.barplot(data=top\_combined, x='Importance', y='Feature', hue='Model')  
  
# Add labels on each bar  
for p in ax.patches:  
 width = p.get\_width()  
 ax.text(width + 0.001, # X position (slightly outside the bar)  
 p.get\_y() + p.get\_height()/2, # Y position (center of the bar)  
 f'{width:.3f}', # Text (rounded to 3 decimals)  
 va='center')  
  
plt.title('Top 15 Feature Importances – Random Forest vs XGBoost')  
plt.tight\_layout()  
plt.show()

Code Cell 129

import matplotlib.pyplot as plt  
import pandas as pd  
import seaborn as sns  
  
# Retrieve the tuned Random Forest model  
rf\_model = best\_models\_smote['Random Forest'] # or best\_models\_original\_tuned['Random Forest']  
  
# Use the correct feature names  
feature\_names = X\_train.columns # or X\_train\_smote.columns depending on what you used  
  
# Create DataFrame for Random Forest feature importances  
rf\_importances = pd.DataFrame({  
 'Feature': feature\_names,  
 'Importance': rf\_model.feature\_importances\_,  
 'Model': 'Random Forest'  
})  
  
# Sort and take top 15 features  
top\_rf = rf\_importances.sort\_values(by='Importance', ascending=False).head(15)  
  
# Plot  
plt.figure(figsize=(12, 8))  
ax = sns.barplot(data=top\_rf, x='Importance', y='Feature', palette='viridis')  
  
# Add labels on each bar  
for p in ax.patches:  
 width = p.get\_width()  
 ax.text(width + 0.001, p.get\_y() + p.get\_height()/2,  
 f'{width:.3f}', va='center')  
  
plt.title('Top 15 Feature Importances – Random Forest')  
plt.tight\_layout()  
plt.show()

Code Cell 130

import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
  
# Retrieve the tuned Random Forest model  
rf\_model = best\_models\_smote['Random Forest']  
  
# Use the correct feature names  
feature\_names = X\_train.columns  
  
# Create DataFrame for Random Forest feature importances  
rf\_importances = pd.DataFrame({  
 'Feature': feature\_names,  
 'Importance': rf\_model.feature\_importances\_,  
 'Model': 'Random Forest'  
})  
  
# Sort and take top 5 features  
top\_rf = rf\_importances.sort\_values(by='Importance', ascending=False).head(5)  
top\_features = top\_rf['Feature'].values  
  
# Plot bivariate plots for each top feature  
for feature in top\_features:  
 plt.figure(figsize=(8, 5))  
   
 if df[feature].dtype == 'object' or df[feature].nunique() < 10:  
 ax = sns.countplot(data=df, x=feature, hue='Churn', palette='Set2')  
 plt.title(f'{feature} vs Churn')  
 plt.xticks(rotation=45)  
  
 # Add count labels  
 for container in ax.containers:  
 ax.bar\_label(container, label\_type='edge', fontsize=10)  
   
 else:  
 ax = sns.boxplot(data=df, x='Churn', y=feature, palette='Set2')  
 plt.title(f'{feature} Distribution by Churn')  
  
 # Optionally add median values  
 medians = df.groupby('Churn')[feature].median()  
 for i, median in enumerate(medians):  
 ax.text(i, median, f'{median:.1f}', horizontalalignment='center', color='black', weight='bold')  
  
 plt.tight\_layout()  
 plt.show()