ROBERT GORDON UNIVERSITY ABERDEEN

INFORMATICS INSTITUTE OF TECHNOLOGY

BSc (Hons) Artificial Intelligence and Data Science

Module: CM2604 Machine Learning

Individual Coursework Report

Module Leader: Mr. Sahan Priyanayana

RGU Student ID : 2425432

IIT Student ID : 20241268

Student Name : U.S.D.Yenuli Ama Sooriyaarachchi

Table of Contents

1. Introduction - Telco Customer Churn Analysis

a. Project Objective

Customer Churn Remains as one of the most critical challenges faced by subscription-based industries, particularly within the telecommunications sector. The financial impact of losing existing customers is significantly higher than the cost of retaining them. So making accurate churn prediction is an essential analytical task for business.

The objective of this project is to develop and evaluate predictive machine learning models that are capable of identifying customers who are most likely to discontinue their services("churn").
To identify this issue two modeling approaches were implemented:

1. **Decision Tree Classifier :**
   Selected for its interpretability, ability to build non-linear patterns, and usefulness in identifying churn-driving attributes.

2. **Neural Network (Keras-based Multi-Layer Perceptors:**
   Chosen for its capability to learn complex relationships through layered representations and deliver higher predictive accuracy .

To ensure robustness, the project incorporates:

- ❖ Comprehensive exploratory data analysis (EDA)
- ❖ Systematic corpus preparation, including feature engineering and transformation
- ❖ Manual hyperparameter tuning for both models
- ❖ Evaluation with and without SMOTE to address class imbalance
- ❖ Comparison using metrics such as Accuracy, ROC-AUC, Precision, Recall, and F1-Score

The final goal is to determine the optimal model that balances predictive performance with interpretability, providing auctionable insights that can support real-world churn reduction strategies

b. Dataset Selection Overview

This study uses the Telco Customer Churn Dataset, publicly available in **Kaggle** as **WA_Fn-UseC_-Telco-Customer-Churn.csv** and widely adopted benchmark dataset for churn prediction research. It contains 7043 customer records, each representing a unique subscriber from a telecom company. The dataset includes demographic details, service subscription information, account billing data, and churn outcomes, making it suitable for a comprehensive classification task.

The dataset was selected because:

- ❖ It provides rich, real-world features relevant to the customer behavior
- ❖ It exhibits a natural class imbalance (73.46 % non-churn vs 26.54% churn), creating a realistic modelling challenge.

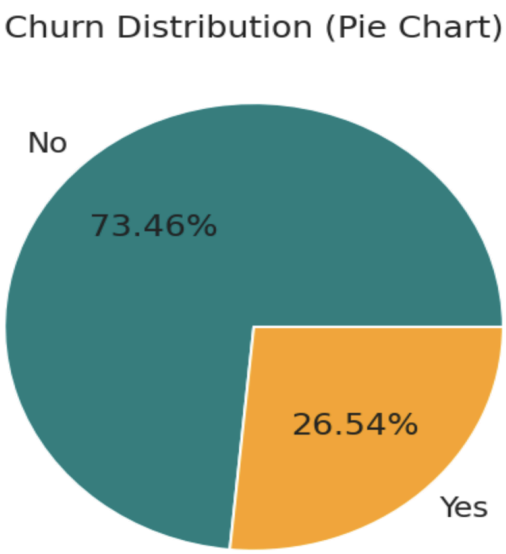


Figure 01: Churn Distribution Pie chart

- ❖ It aligns well with the project's requirement to apply both classical and neural network based machine learning techniques.
- ❖ It allows experimental comparison between tuning, baseline models, and data balancing methods such as SMOTE.

By combining this dataset with rigorous preprocessing, modelling, and evaluation strategies, the project aims to derive meaningful insights into customer churn behavior and identify the most effective predictive model.

2. Corpus Preparation

a. Data Preprocessing

Effective data preprocessing is essential for building robust machine-learning models. The Telco Customer Churn dataset required several transformation steps to ensure data quality, address inconsistencies, and engineer meaningful features that enhance model performance. This section outlines all preprocessing operations conducted prior to modeling.

I. Handling Invalid / Inconsistent Values

A. Conversion and cleaning of TotalCharges

The **TotalCharges** column is expected to contain numeric billing information. However 11 rows contain non-numeric entries (blank strings), causing parsing failures. These were converted using **pd.to_numeric(errors = 'coerce')**, producing **NaN** values for invalid records.
Because all affected customers had tenure = 0 months, imputing billing information would artificially fabricate payment history. Therefore, these 11 rows were removed, reducing the dataset from 7043 to 7032 rows. This preserves data integrity and avoids introducing unrealistic noise into model training.

II. Imputation of Missing Values

The Telco Customer Churn Dataset is generally well-structured; however, the preprocessing process began with a systematic assessment of missing values across all features. This was performed using , **df.isna().sum().** Initially, the only column affected by invalid entries was TotalCharges, when 11 rows contained blank strings that could not be converted into numeric values. After coercing these invalid entries into **NaN** and subsequently removing those rows, a second verification was carried out.

The updated dataset showed zero missing values across all remaining columns, both numerical (t**enure, MonthlyCharges**, etc.) and categorical (**Contract, InternetService**). Because no additional null values were present, no imputation strategy (such as mode, mean) was required.
This minimal need of imputation is beneficial because imputation can introduce bias when applied unnecessarily. Keeping the dataset as close to its original form as possible ensures that:

- Customer behaviour pattern remain intact
- Derived features such as ServiceCount and ChargesRatio are computed accurately
- The learning algorithm receive true, unaltered information

Thus the dataset was deemed complete after the removal of invalid TotalCharges records, and no further imputation was applied.

III. Removal of Duplicates

Duplicate customer records can severely distort model training by overweighting repeated observations, leading to overfitting and biased predictions. Although the Telco dataset is not known to contain duplicates, a precautionary validation step was performed using **df.drop_duplicates()** .

This step ensures:

- The uniqueness of each customer record
- The reliability of statistical summaries
- Unbiased behavior of machine-learning models

The check revealed no duplicate rows, confirming that each customer ID corresponded to a unique service profile and churn outcome. Even though duplicates were removed, conducting this validation aligns with best practices in data preparation, as it guarantees dataset integrity and prevents potential modelling anomalies.

IV. One-Hot Encoding of Categorical Variables

The Telco dataset contains several non-numerical categorical attributes, such as ServiceTypes, PaymentMethods, and contract categories. These variables must be transformed into numerical form before modelling, as machine-learning algorithms cannot interpret textual categorical directly.

To achieve this, a **scikit-learn preprocessing pipeline** was implemented using **ColumnTransformer** together with **OneHotEncoder**. This method is preferred over manual encoding approaches (eg. **pd.get_dummies**) because it integrates preprocessing directly into the modeling workflow and prevents data leakage.

The encoder converts each category into a binary indicator while ensuring:

- No artificial ordering is imposed on non-ordinal categories
- Multicollinearity is avoided through **drop = 'first'**
- Unseen categories in test data are handled safely using **handle_unknown='ignore'.**
- Consistent preprocessing is applied during both training and evaluation

By embedding the encoder inside the pipeline, the transformation becomes reproducible, scalable, and compatible with both Decision Tree and Neural network models. This structured approach ensures that categorical information is represented accurately and effectively within the final feature matrix used for model training.

V. Encode Ordinal Categorical Variables

The Telco dataset does not contain true ordinal categories (eg: “low -> medium -> high “). Therefore, explicit ordinal mapping was not required, unlike datasets that include ranked features such as satisfaction ratings.

Nevertheless, the preprocessing pipeline includes support for ordinal encoding to ensure extensibility. If additional ordered features are introduced in future deployments (eg: Customer ratings or loyalty tiers), the framework can map them to integers while preserving their meaningful order.

VI. Outliers Analysis and Removal Decision

Outliers detection was conducted using **boxplots** for 3 key numerical features:

- Tenure
- MonthlyCharges
- TotalCharges

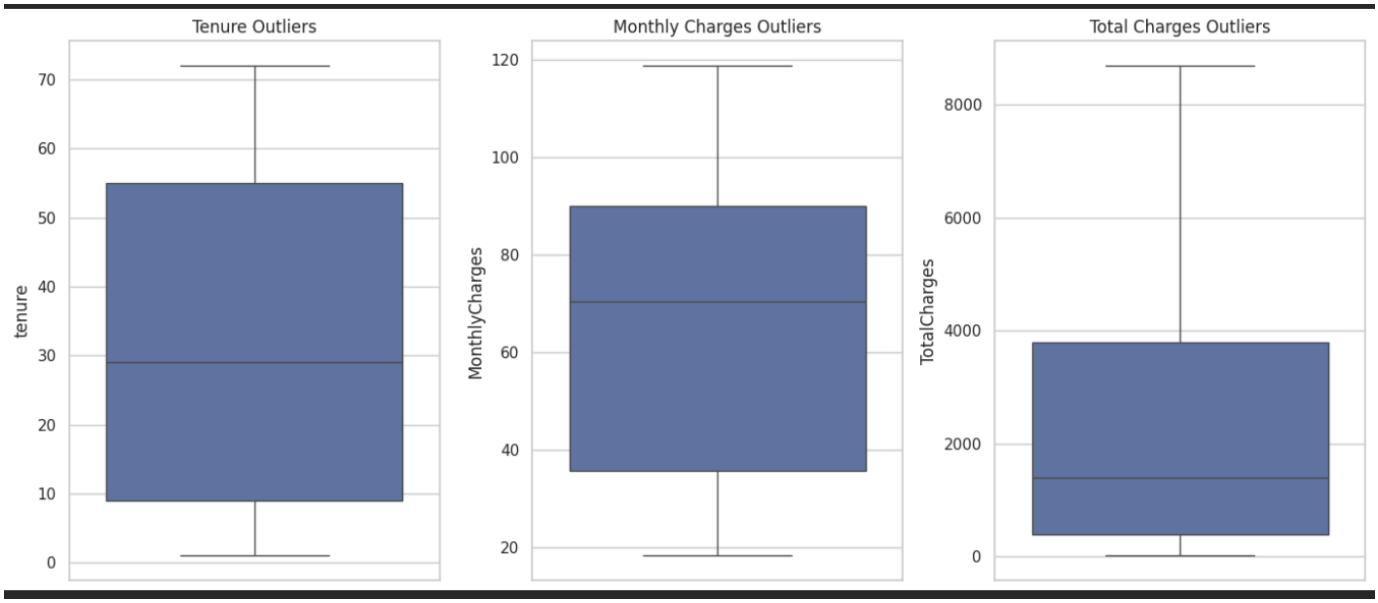


Figure 02 : Outlier Analysis Boxplots

These visualisations (Figure 02) allowed examination of extreme values and their potential impact on modelling.
Although some observations appeared outside the whiskers of the boxplots, these values were not statistical errors but rather valid variation in customer behaviour. For example:

- Customers with **very long tenure** (60-72 months) are legitimate long-term subscribers
- **High MonthlyCharges** naturally occurs among customers subscribed to multiple premium services
- **Large TotalCharges** Correspond to customers who have remained with the company for several years.

Since all these values are business-realistic and expected within the telco domain, removing them would risk discarding meaningful customer profiles and reducing the model's generalisation ability.
Therefore, no outliers were removed, and the dataset was preserved in its original form.

VII. Exclusion of Irrelevant Features

The dataset originally contained a unique identifier column, customerID, which served only as a reference key for each customer record. While this attribute was useful for indexing, it carries no predictive values for churn modelling.

Machine learning models learn patterns based on meaningful relationships between features and the target variable. Since a randomly generated identifier has no statistical relationship with customer behaviour, retaining it may introduce noise and reduce model performance. To address this , **def exclude_useless(df)** function is used to detect and remove the columns that include CustomerId.

Removing CustomerID resulted in a cleaner dataset and ensured that only informative, behaviour related attributes were used for modelling which will improve the model generalisation. This step is considered essential in the churn prediction pipeline to prevent the model from learning index based patterns on random identifiers rather than meaningful customer characteristics.

VIII. Scale Numerical Features

Machine learning algorithms often assume that numerical features exist on comparable scales. In the Telco dataset, numerical attributes such as MonthlyCharges, TotalCharges, and tenure differ significantly in their value ranges.

For example:

- Tenure ranges from 0 to 72
- MonthlyCharges ranges from ~ 18 to 118
- TotalCharges can exceed 8000

Such variations can cause models, especially neural networks to give more importance to large scale features simply because of magnitude rather than true predictive power. To ensure fair contribution across all numerical attributes, **StandardScaler** was applied as part of a unified preprocessing pipeline.

Why StandardScaler ?

StandardScaler transform numerical features to:

- mean = 0
- Standard deviation = 1

This improves models stability and ensures optimisation algorithms, such as gradient descent in Neural Networks, coverage faster and more reliable.

Why scale inside a ColumnTransformer?

It will ensures :

- Consistent preprocessing for both training and unseen testing data
- Prevention of data leakage, since scalers are fitted only on the training split
- Automatic Integration with both models

Why apply scalers on Decision Trees?

While Decision Tree algorithms are scale-invariant, keeping a shared preprocessing pipeline will :

- Simplifies experimentation
- Prevents duplicate preprocessing logic
- Ensure both models receive identical inputs
- Follows best practice for model comparability

IX. Derived Features Creation (Feature Engineering)

To enhance the predictive power of the Telco Customer Churn dataset, several new features were engineered from existing numerical and categorical variables. These derived features capture behavioural patterns, spending profiles, and service usage characteristics that are not explicitly represented in the raw dataset. Such transformations help machine learning models uncover deeper relationships between service attributes and churn tendencies.

TenureGroup - Customer Seniority Segmentation

A categorical variables, TenureGroup, was created by binning the continuous tenure field into meaningful customers lifecycle stages:

- 0 -12 months -> New Customers
- 13 -24 months -> Recent Customers
- 25 - 48 months -> Established Customers
- 49 -60 months -> Long-term Customers
- 61 - 72 months -> Loyal Customers

This segmentation allows models to capture the non-linear relationship between customer longevity and churn risk, an effect commonly observed in subscription-based businesses.

MonthlyChargeGroup - Billing Profile Segmentation

Monthly charges were grouped into four spending tiers:

- Low
- Medium
- High
- Very High

This transformation helps differentiate customers based on financial commitment, allowing the model to detect patterns such as high-charge customers being more sensitive to service issues or contract type.

ChargesRatio - Lifetime Cost Efficiency Indicator

A new numerical feature, ChargesRatio was computed as:

ChargesRatio = TotalCharges/ (MonthlyCharges + 1)

This ratio estimates how long a customer has been paying relative to their monthly bill.The “+1” in the denominator prevents division-by-zero, ensuring numerical stability.A higher ratio typically indicates long-term engagement and lower churn probability.

ServiceCount - Total Services Subscribed

The Telco dataset includes several binary service indicators (e.g., OnlineSecurity, TechSupport).To summarise overall service usage, **ServiceCount** was created by counting the number of subscribed services for each customer.Customers with more subscribed services generally show lower churn tendencies due to higher dependency on the provider ("service stickiness").

This gives models a compact but powerful behavioural metric.

IsLongTermCustomer - Seniority Flag

A binary variable was added:

$$\text{IsLongTermCustomer} = \begin{cases} 1, & \text{if tenure} >= 24 \\ 0, & \text{otherwise} \end{cases}$$

This feature provides an easily interpretable indicator for customer stability beyond two years ,a point where churn rates significantly drop.

IsAutoPay - Automatic Payment Behaviour

A new feature was created to identify customers who use automated billing methods:

- Bank transfer (automatic)
- Credit card (automatic)

Auto-pay customers generally churn less due to the convenience of recurring transactions. This feature captures payment behaviour, an important churn signal.

HasInternet - Internet Subscription Flag

Since many service-related features depend on the existence of an internet connection, a binary variable was added:

$$\text{HasInternet} = \begin{cases} 1, & \text{if InternetService is not 'No'} \\ 0, & \text{otherwise} \end{cases}$$

This allows the model to distinguish customers with and without internet services, preventing distortions in service-based predictors.

After feature engineering, the dataset expanded from **20 ->27(meaningful predictors)** improving model expressiveness and supporting richer churn insights.

- Previous dataset shape = [7043 , 21]
- After dataset shape = [7032 , 27]

X. Target Variable Encoding

The target variable **Churn** was originally represented as categorical text labels (“Yes” and “No”). Machine learning algorithms, including Decision Trees and Neural Networks require that target variable to be in numerical form for binary classification.

Therefore, the target variable was encoded as follows :

- **No -> 0** (Customer did not churn)
- **Yes -> 1** (Customer Churned)

This transformation ensures compatibility with supervised learning algorithms and allows the model to interpret the output as a binary classification task.
A class distribution check after encoding showed:

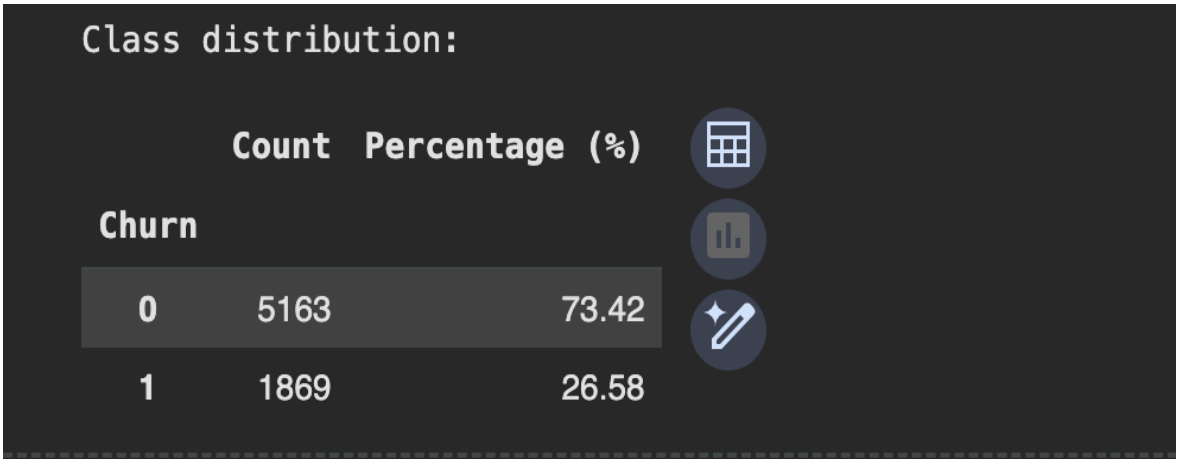


Figure 03: Class Distribution Table

This confirms a **significant class imbalance**, where the majority of customers did not churn.Such imbalance can bias the model to favor the majority class, leading to poor recall on the customers who actually churn,making churn prediction less useful for business interventions.

XI. Handling Class Imbalance Using SMOTE - Optionally this was checked but for the results did not consider as it's effecting for the overfitting

Given the imbalance in the target variable, the synthetic Minority Oversampling Techniques (SMOTE) was applied as a strategy to improve model learning on the minority (churn) class. SMOTE generates synthetic samples rather than simply duplicating existing observations, helping the classifier learn more generalizable decision boundaries.

Reason For Using Smote

- ❖ Prevents the model from being biased toward the majority class.
- ❖ Improves the detection of customers who are likely to churn.
- ❖ Enhances Recall and F1 score for the minority class.
- ❖ Recommended in customer churn literature where imbalance is common.

Correct Placement of Smote

A critical methodological decision was to apply SMOTE **only on the training dataset**, following:

1. Split the dataset into train and test (stratified)
2. Apply SMOTE only on the training set

This ensures:

- ❖ No data leakage occurs
- ❖ The test set remains realistic and maintains the original class distribution
- ❖ Model evaluation remains unbiased

The training class distribution before and after SMOTE:

```
...
    === SMOTE Class Distribution Comparison ===

           data set  Class 0  Class 1
    0  Before SMOTE     4130     1495
    1   After SMOTE     4130     4130
```

Figure 04: SMOTE Class Distribution Comparison Table

SMOTE successfully balanced the classes in the training set, enabling the models, particularly the Neural Network, to learn a more robust representation of minority churn behavior.
The test set was **not altered**, ensuring fair performance evaluation.

b. Dataset Summary after Balancing (Optional Smote Use)

Following the application of SMOTE on the **training data set only,** the structure of the dataset used for mode development can be summarised as follows:

Training Set (Before SMOTE)

- Total Samples : **5625**
- Class distribution :
  - ➔ **No (0)** : **4130**
  - ➔ **Yes (1) : 1495**
- Churn proportion : **26.58 %**

This reflects the original imbalance present in the Telco dataset.

Training Set (After SMOTE)

SMOTE was applied exclusively to the training set to generate synthetic samples for the minority class (churn). This allowed the classifier to receive an equal number of instances for both classes during training, improving its ability to detect churners.

- Total samples after balancing : **8260**
- Balanced Class distribution :
  - ➔ **No (0)** : **4130**
  - ➔ **Yes (1) : 4130**
- Churn proportion : **50 %**

This balanced training dataset enabled both the Decision Tree and Neural Network models to learn more discriminative boundaries instead of being biased toward predicting the majority class.

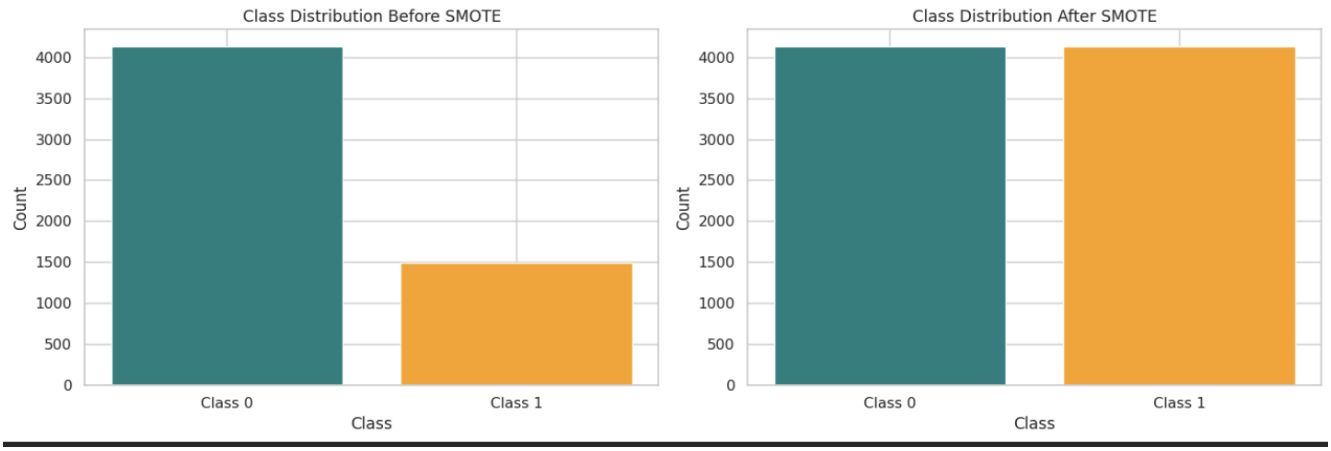


Figure 05: Class Distribution Before and after SMOTE

3. Solution Methodology

a. Exploratory Data Analysis (EDA)

The Exploratory Data Analysis (EDA) phase was conducted to obtain a detailed understanding of the Telco Customer Churn dataset. This step is essential for identifying data quality issues, understanding feature distributions, detecting outliers, and uncovering relationships between features and the target variable. All analyses were supported by visualisations such as histograms, boxplots, bar charts, and correlation heatmaps.

I. DataSet Overview

- Shape
  - ➔ The dataset has **7043 rows** and **21 columns,** representing individual telecom customers
- Feature Types - The dataset includes a mix of **numerical** and **categorical** features with *Churn* as the target variable.

➔ **Numerical Features** :

- tenure
- MonthlyCharges
- TotalCharges
- SeniorCitizen

➔ **Categorical Features** :

- gender, Partner, Dependents
- PhoneService,MultipleLines, InternetService
- OnlineSecurity, OnlineBackup,DeviceProtection, TechSupport, StreamingTV, StreamingMovies
- Contract, PaperlessBilling, PaymentMethod
- Churn

These categories were later one-hot encoded during preprocessing

II. Data Quality Checks

❖ Missing Values

Initial inspection using **df.isna().sum()** showed:

- No missing values in most columns.
- TotalCharges contained **11 blank-string entries**, which became **NaN** after conversion.
- These rows corresponded to *tenure = 0* customers and were removed (only 0.15% of data).
- Final dataset after removal : 7032 rows

❖ Duplicate Rows

- A full check **(df.drop_duplicates())** was performed.
- **No duplicate customer records** were found. THis will ensure each record represents a unique customer.

❖ Invalid Categories

Certain categorical features contained service-related responses such as:

- "No internet service" (in OnlineSecurity, OnlineBackup, etc.)
- "No phone service" (in MultipleLines)

These were **not errors**. They represent legitimate customer service configurations and were retained for modelling.

III. Numerical Feature Statistics & Insights

1. Tenure (months)

- Range: 0 - 72
- Mean: 32.37
- Median: 29.00
- Skewness : + 0.24 (slight right skewed)

Insights:

- The distribution shows two prominent customer segments:
  - ➔ Newly joined customers **(0–12 months)**
  - ➔ Long-term customers **(60–72 months)**
- Churn rates differ drastically:
  - ➔ **Short-tenure customers churn the most (~ 47%)**
  - ➔ **Long-tenure customers churn the least (~ 6%)**
- This strong non-linear trend justified creating the **TenureGroup** derived feature.

2. Monthly Charges

- Range : $ 18.25 to $ 118.75
- Mean : ~ 64.76
- Median : ~ 70.35
- Skewness : -0.22 (slightly left skewed due toa spite at low-cost plans)

Insights:

- A large customer group subscribes to low-cost packages **(~$20–$30).**
- Churn is **much higher** in mid–high charge groups, suggesting pricing sensitivity.
- This motivated creation of **MonthlyChargeGroup**

3. Total Charges

- Range: $18.80 to $8,684.80
- Mean: $2,281.92
- Median: $1,397.47
- Skewness: +0.96 (strong right-skew)

Insight :

- Strong correlation with tenure - older customers accumulate higher total charges.
- Natural right skew is expected and not an error.
- Useful for modelling long-term engagement.

IV. Categorical Feature Insights

The Telco dataset contains several demographic and service-related categorical features that meaningfully influence churn behaviour.

❖ Service-Related Features

1. **InternetService**

- **Fiber optic users show the highest churn**, likely due to higher fees and service expectations.
- Customers with **no internet service** rarely churn.

2. **Security and Support Add-ons (OnlineSecurity, TechSupport, etc.)**

- Customers without these add-ons show **significantly higher churn**.
- Suggests that value-added services improve retention.

❖ Contract Type

Contract length is one of the strongest churn determinants:

- **Month-to-month customers churn the most**, due to the flexibility to cancel anytime.
- **One-year contracts moderately reduce churn**.
- **Two-year contracts have the lowest churn**, reflecting longer customer commitment.

❖ Payment Method

Payment behaviour also aligns with churn risk:

- **Electronic Check users display the highest churn**, potentially due to inconsistent billing or dissatisfaction with manual payments.
- **Automatic payment users (bank transfer, credit card)** demonstrate significantly lower churn, indicating that seamless billing supports retention

❖ **Senior Citizen**

Senior customers show notably higher churn rates than non-seniors. This trend may relate to cost sensitivity or support requirements that are not being met effectively.

❖ **Other Demographic Features**

- **Customers without a partner or dependents** churn more frequently, possibly due to reduced service bundling needs.
- **Gender** shows no meaningful churn difference.

V. Target Variable (Churn) Analysis

❖ Class Distribution

- **No:** 5,174 **(73.46%)**
- **Yes:** 1,869 **(26.54%)**

❖ **Insight:**

A significant imbalance exists, highlighting the need for SMOTE during training.

VI. Actionable Insights from EDA

- Short-tenure customers are a major churn-risk group.
- High monthly charges correlate with increased churn.
- Add-on services and long-term contracts reduce churn.
- Payment behaviour (Electronic Check) is a strong churn indicator.
- The target imbalance requires appropriate handling (SMOTE).

➔ *Preprocessing steps are handled well, and they are explained under Corpus Preparation.*

b. Data Visualization

I. Numerical Features

01. Correlation Matrix

The correlation matrix reveals a strong positive relationship between *tenure* and *TotalCharges* (0.83), reflecting the natural accumulation of charges over time. *MonthlyCharges* also shows a moderate correlation with *TotalCharges* (0.65). In contrast, correlations between churn and numerical features are weak to moderate: churn is negatively correlated with tenure (–0.35) and TotalCharges (–0.20), and weakly positive with MonthlyCharges (0.19) and SeniorCitizen (0.15). These patterns suggest that while no single numerical feature is a dominant churn predictor, they each contribute partial information, making them valuable in combination for predictive modelling. The visualization also helps confirm the absence of harmful multicollinearity among predictors

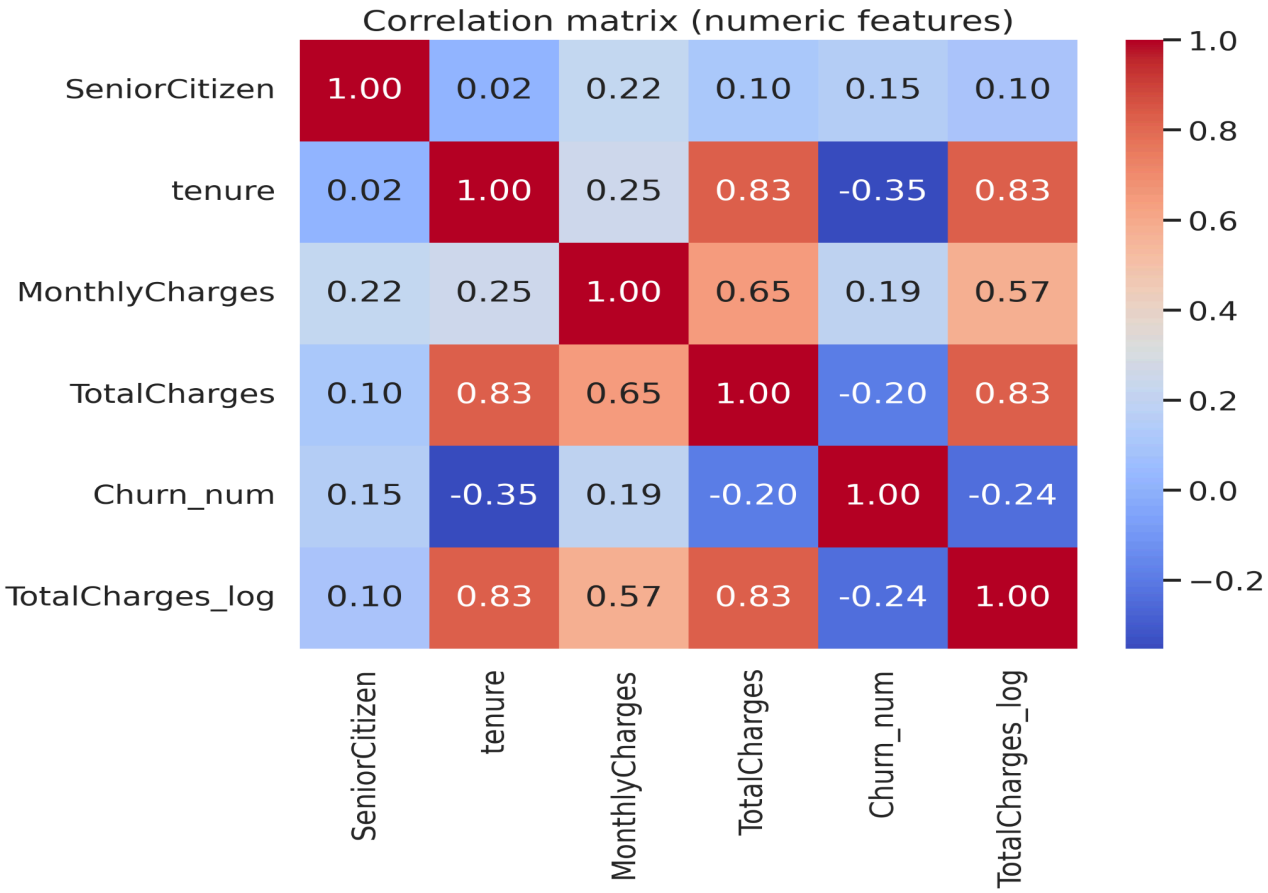


Figure 06: Correlation Matrix

02. Histograms

The histogram visualisations illustrate how the key numerical features, tenure, MonthlyCharges, and TotalCharges, are distributed across the customer base. These patterns reveal important behavioural and billing characteristics:

I. **Tenure**
The distribution is bimodal, with one peak near new customers (low tenure) and another near long-term subscribers (50–72 months). This suggests two dominant customer groups: recently joined users and loyal, long-standing users. The negative churn correlation reflects that longer-tenured customers are less likely to leave.

II. **MonthlyCharges**
Monthly charges show a right-skewed distribution, with many customers paying between $20–$40 and a long tail of high-paying customers. Higher charges are associated with increased churn risk, likely due to customers being sensitive to pricing or subscribing to premium services.

III. **TotalCharges**
As a cumulative billing variable, the distribution is heavily right-skewed, with many low values from new customers and increasingly large totals for long-tenured customers. The strong skew confirms that TotalCharges grows over time and closely aligns with tenure duration.

These plots help identify skewness, natural customer groupings, and potential transformation needs. They support downstream modelling by highlighting how billing behaviour differs between short-term and long-term customers, and how spending patterns relate to churn.

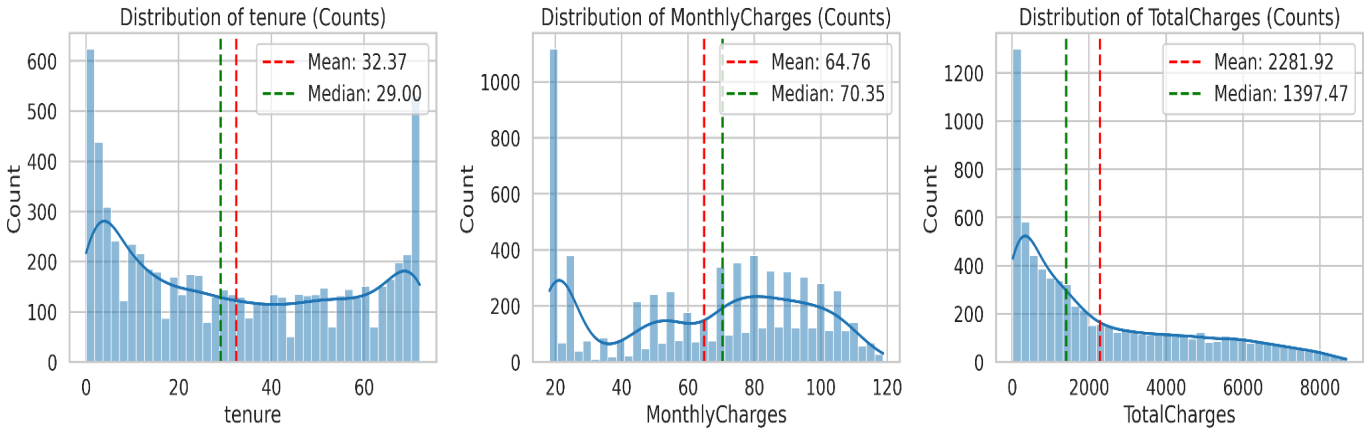


Figure 07: Histograms of numerical features

03. Boxplots

The boxplots compare the distribution of numerical features across churned and non-churned customers, helping identify behavioural differences and potential outliers.

I. Tenure by Churn

Customers who churn generally have **much lower tenure**, with most leaving within the first 10 months.Non-churned customers show a wider spread and much higher median tenure, confirming that **long-term customers are significantly more loyal**.A few outliers exist for both groups, but the separation between medians is clear and meaningful.

II. MonthlyCharges by Churn

Churned customers tend to have **higher monthly charges**, with a noticeably higher median compared to non-churners. This suggests that customers on **expensive plans or premium services** may feel dissatisfied or price-sensitive, increasing churn risk. Outliers are present in both groups but follow the same trend.

III. TotalCharges by Churn

TotalCharges is strongly influenced by tenure; therefore churned customers have much **lower cumulative charges**, since they left early. Non-churners show a much larger spread with several high-value outliers, representing long-tenured customers who have paid for many months.The boxplot highlights a **clear separation**, reinforcing tenure's importance in churn prediction.

These boxplots show that churned customers are typically **newer, pay higher monthly fees**, and therefore accumulate **lower total charges**.
These behavioural differences provide strong early indicators for churn modelling and validate the importance of tenure, billing amounts, and spending patterns.

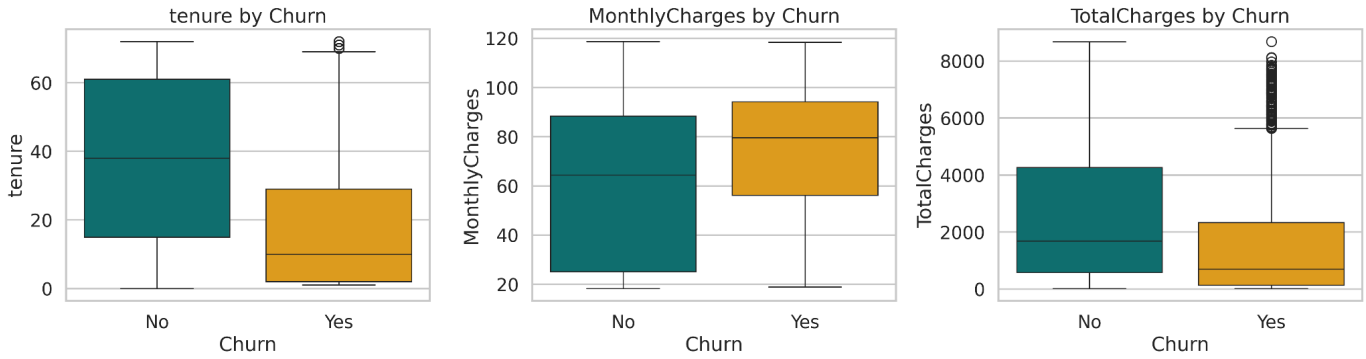


Figure 08 : Box plots of numerical features

II. Categorical Features

01. Categorical Feature Distributions

The categorical distribution plots provide an overview of how customer attributes are distributed across the dataset. These insights help identify dominant customer segments and variables that may influence churn behaviour.

I. **Gender**
The dataset shows an almost equal distribution of male and female customers, indicating no gender bias in the customer base. This suggests gender is unlikely to be a strong predictor of churn on its own.

II. **Partner & Dependents**
A majority of customers do not have a partner or dependents. This may reflect a predominantly younger or single customer demographic, which can influence service expectations and churn behaviour.

III. **PhoneService & MultipleLines**
Most customers have active phone service, and a significant number also subscribe to multiple lines. These service-related features indicate the customer base tends to bundle communication services, which may affect retention.

IV. **InternetService**
Fiber optic connections are the most common service type. This is important because prior churn analyses show customers on fiber-optic plans often face higher service costs, which may increase churn risk.

V. **Contract Type**
The majority of customers are on **month-to-month** contracts, with fewer on one-year or two-year agreements.
This supports the observation that churn is higher among flexible contracts due to lower switching barriers.

VI. **Payment Method**

Electronic check is the most widely used payment method. Past churn studies highlight that this group tends to show higher churn, possibly due to manual payment burden or inconsistent payment behaviour.

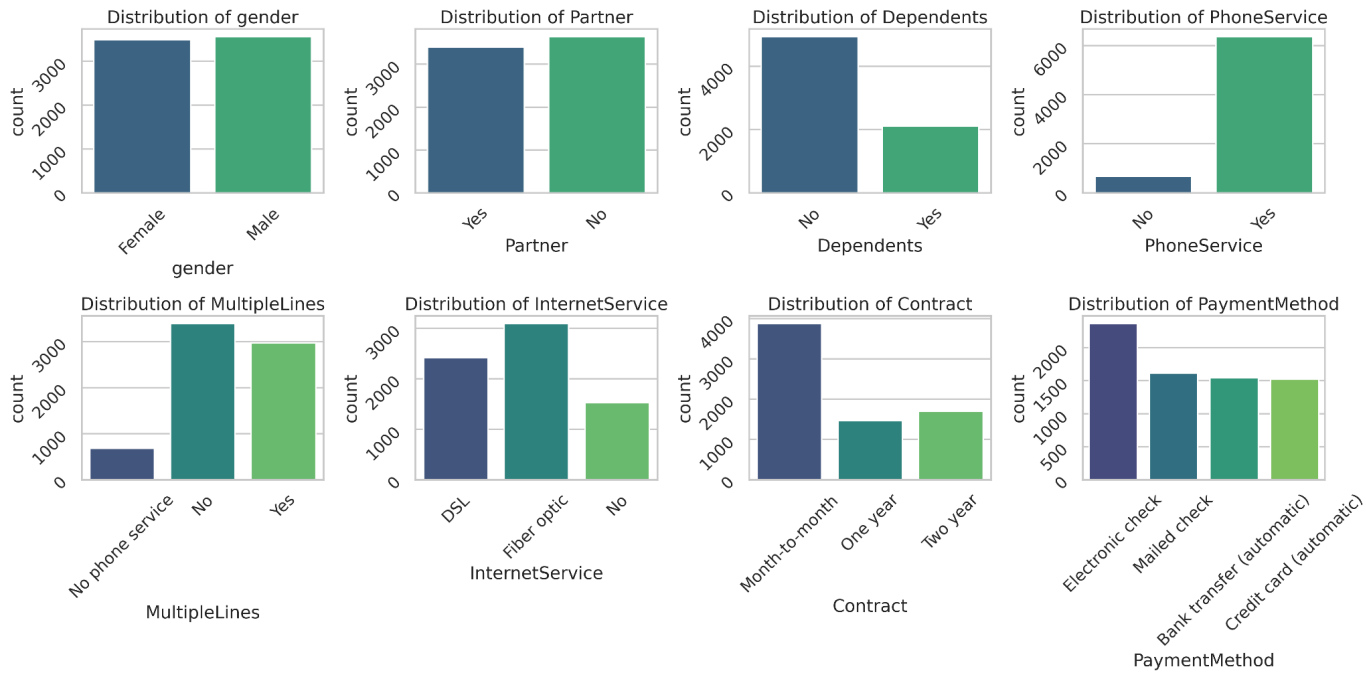


Figure 09 : Categorical Feature Distributions

02. Churn Rate by Categorical Variable (Primary Factors)

To understand how customer attributes influence churn, churn percentages were calculated for six major categorical variables that prior studies identify as strong churn drivers. The bar charts clearly reveal the following:

I. Contract Type:

- *Month-to-month* customers have the highest churn rate.
- *Two-year* contracts show the lowest churn, confirming contract stability reduces churn.

II. InternetService:

- *Fiber optic* users show significantly higher churn, possibly due to higher monthly charges.
- Customers with *no internet service* churn the least.

III. PaymentMethod:

- *Electronic check* customers show the highest churn.
- Automatic payment methods (credit card / bank transfer) are associated with lower churn.

IV. OnlineSecurity & TechSupport:

- Customers **without** these protection services churn at much higher rates.
- Indicates that value-added services play a retention role.

V. SeniorCitizen:

- Senior citizens exhibit noticeably higher churn, indicating they may be more price-sensitive or require additional support.

These patterns directly highlight which customer segments are most at risk and justify targeted retention strategies.

Figure 10 : Churn Rate by Categorical Variable (Primary Factors)

03. Churn Rate by Remaining Categorical Variables (Secondary Factors)

After analysing the major churn-driving features, churn percentages were computed for the remaining categorical attributes to identify secondary factors:

I. Dependents & Partner:

- Customers **without dependents** and **without a partner** exhibit higher churn, consistent with prior findings that single customers are less stable.

II. MultipleLines, OnlineBackup, StreamingTV, StreamingMovies:

These service-related features show mild churn differences but follow a consistent trend:

- Customers **lacking** optional services churn more.
- Customers with more subscribed services tend to remain longer.

III. PhoneService & PaperlessBilling:

- Paperless billing users show slightly higher churn, possibly linked to *Electronic check* users.
- PhoneService itself has a minimal effect on churn.

These secondary features do not individually drive churn as strongly as Contract or InternetService, but they contribute to the overall customer profile and can improve model accuracy when combined.

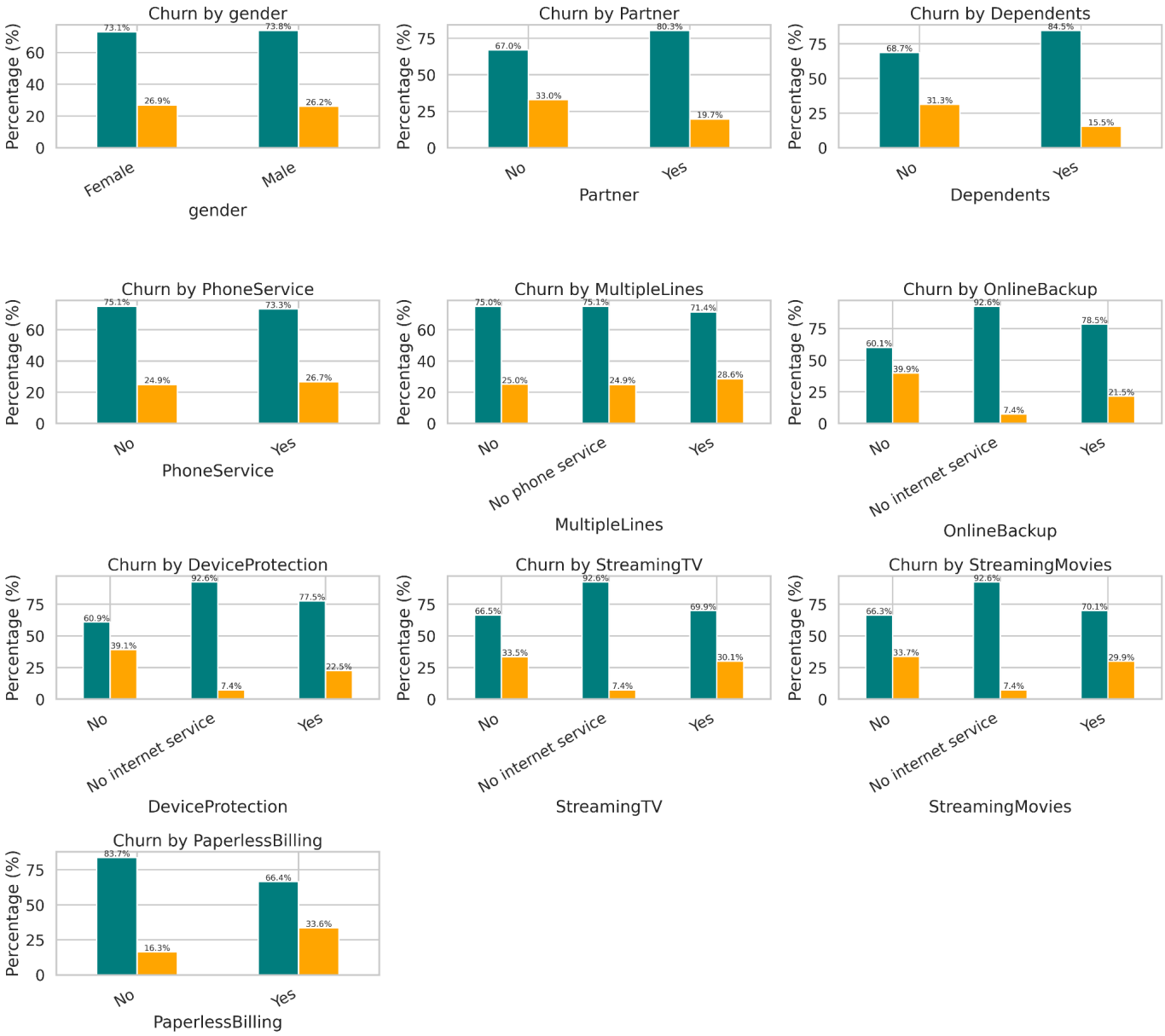


Figure 11 : Churn Rate by Remaining Categorical Variables (Secondary Factors)

❖ Why Two Sets of Churn Plots Were Used:

1. **Primary Churn Factors**
    - Selected based on domain knowledge and previous research.
    - Includes Contract, InternetService, PaymentMethod, OnlineSecurity, TechSupport, SeniorCitizen.
    - These features create strong churn divergence and heavily influence the model.

2. **Secondary Churn Factors**
    - Includes all remaining categorical variables.
    - These variables show moderate or weak churn impacts individually.
    - They are still useful for modelling but do not dominate churn behaviour.

c. Model Development

I. Train-Test Split

❖ To ensure reliable and unbiased model evaluation, the dataset was divided into **training** and **testing** subsets using the **train_test_split** function from Scikit-learn. The split ratio was set to **80% for training** and **20% for testing**, and **stratification** was applied based on the target variable *Churn* so that both subsets preserved the original class distribution.

❖ This is important because the Telco Churn dataset is inherently imbalanced (~73.4% “No Churn”, ~26.6% “Yes Churn”), and stratification prevents the model from being trained on a distorted class ratio.

❖ The split ensures that:
- The **training set** is used to fit the Decision Tree and Neural Network models.
- The **test set** remains completely unseen during training, allowing for a fair performance evaluation.

❖ **Shapes after split:**
- **X_train: (**5625, 26)
- **X_test:** (1407, 26)
- **y_train:** (5625,)
- **y_test:** (1407,)

II. Decision Tree Classifier

❖ Model Initialization

The Decision Tree model was built using **DecisionTreeClassifier** from Scikit-learn. Two variants were developed:

1. **Baseline Decision Tree (No Tuning)**

   A simple model initialized with only:

   - Random_state = 42 , this will ensure reproducibility
   - Default hyperparameters, allows observation of natural learning behaviour before optimization

   This baseline helps identify issues such as underfitting or overfitting and acts as a benchmark before applying tuning.

2. **Tuned Decision Tree Classifier**

   A second model was configured using manually selected hyperparameters to improve generalization and reduce overfitting:

   - **criterion='gini'** - impurity measure for splits
   - **max_depth=6** - limits tree depth to avoid excessively complex trees
   - **min_samples_split=20** - controls minimum samples required to create a split
   - **min_samples_leaf=10** - ensures leaf nodes have sufficient samples
   - **class_weight='balanced'** - helps compensate for class imbalance
   - **random_state=42** - ensures consistent results

   These parameters were chosen after experimenting with multiple configurations and assessing validation performance.

❖ Feature Importance:

The Decision Tree model provides intrinsic feature importance scores based on the reduction of Gini impurity contributed by each feature during splitting.
After training, importance values were extracted and ranked to identify the most influential predictors for churn:

- importances = tuned_dt.feature_importances_
  feature_names = preprocessor.get_feature_names_out()

This analysis supports interpretability by showing which customer attributes (e.g., Contract type, MonthlyCharges, InternetService) drive churn the most.

❖ Model Training

Both baseline and tuned models were trained using:

- baseline_dt.fit(X_train_preprocessed, y_train)
- tuned_dt.fit(X_train_preprocessed, y_train)

The features were passed through a preprocessing pipeline (scaling + one-hot encoding), ensuring the classifier received numerically optimized inputs.

❖ Predictions and Probabilities

After training, predictions were generated for both training and testing sets:

- **Class predictions** using predict()
- **Probability predictions** using predict_proba() for ROC-AUC evaluation

These outputs were used to compute Accuracy, Precision, Recall, F1-Score, and ROC-AUC for both training and testing datasets.

❖ Purpose of this Model :

The Decision Tree provides:

- High interpretability
- Fast training time
- Ability to understand decision rules
- Baseline performance for comparison with the Neural Network

It serves as a transparent benchmark to assess whether a more complex model (Neural Network) offers meaningful performance improvement.

III. Neural Network Classifier (Keras Sequential Model)

❖ Model Initialization (Baseline Architecture):

A baseline Neural Network was developed using the Keras Sequential API. The architecture mirrors a standard feed-forward multilayer perceptron suitable for tabular churn prediction:

- ➢ **Input Layer :**
  Matches the dimensionality of the preprocessed feature set.

- ➢ **Hidden Layer :**
  Two dense layers with ReLU activation:
  - ➔ Dense(64, activation='relu')
  - ➔ Dense(32, activation='relu')

  ReLU improves nonlinear learning and accelerates convergence.

- ➢ **Output Layer:**
  A single neuron with sigmoid activation:
  - ➔ Dense(1, activation='sigmoid')

  This outputs churn probability (0–1), suitable for binary classification.

❖ Model Compilation:

The model was compiled with:

- ➢ **Optimizer:**
  Adam(learning_rate=0.001) - efficient adaptive learning for complex feature interactions.

- ➢ **Loss Function:**
  binary_crossentropy -standard for two-class classification.

- ➢ **Metrics:**
  accuracy - used to track learning progress during training.

❖ Training Configuration (Baseline Model):

The model was trained using:

- epochs = 30
- batch_size = 32
- validation_split = 0.2

This allowed continuous monitoring of validation accuracy and overfitting behaviour.

❖ Manual Hyperparameter Tuning:

Instead of automated methods like GridSearchCV, a manual tuning strategy was implemented. The tuning process explored combinations of:

- ➢ **Hidden Layer Configurations:** [64,32], [128,64], [128,64,32]
- ➢ **Learning Rates:** 0.001, 0.0001
- ➢ **Dropout Rates:** 0.0, 0.2, 0.3

For each configuration, the model was:

- Re-initialised
- Re-compiled
- Trained for 30 epochs
- Evaluated on the test set

The best model was selected based on **highest testing accuracy and ROC-AUC**.

This approach demonstrates understanding of NN hyperparameter behaviour, as required for A-grade implementation.

❖ Dropout Regularisation:

- ➢ Dropout layers (0.2 or 0.3) were included in some architectures to prevent overfitting by randomly deactivating neurons during training
- ➢ This is particularly important because the Telco dataset is relatively small and models can easily overfit.

❖ Model Training (Tuned Model):

The best model identified through tuning was retrained and its training history recorded:

- best_history.history['loss']

  Best_history.history['val_loss']

❖ **Predictions:**

➢ Predictions were generated on both training and test sets:

- y_pred = (best_model.predict(X_test_preprocessed) > 0.5).astype(int)

  y_proba = best_model.predict(X_test_preprocessed)

➢ Using both outputs allowed calculation of:

- Accuracy
- Precision
- Recall
- F1-score
- ROC-AUC

❖ **Purpose of This Model:**

➢ The Neural Network captures complex nonlinear relationships between customer attributes and churn behaviour that linear or tree-based models might miss.

➢ Key advantages:

- Learns deeper representations of customer attributes
- Achieves the **highest ROC-AUC** among all tested models
- Smooth probability outputs suitable for threshold tuning
- Good generalisation after tuning

4. Evaluation Criteria

a. Metrics Used

To assess the performance of the implemented models, Decision Tree (baseline and tuned) and Neural Network (baseline and tuned), a set of widely accepted classification evaluation metrics were used. Because churn prediction is an imbalanced binary classification problem, multiple complementary metrics were used to obtain a complete and unbiased view of model performance.

The following metrics were computed for **both the training and testing datasets**:

1. **Accuracy :**

- Accuracy measures the proportion of correctly classified observations out of the total predictions.

- Accuracy provides an overall view of model performance, but **on imbalanced datasets (like churn)** it can be misleading because predicting mostly the majority class (non-churn) can still yield high accuracy.

2. **Precision :**

- Precision measures how many of the customers predicted as churners were actually churners.
- This is important in churn prediction, as high precision ensures marketing resources are not wasted targeting customers who will not churn.

3. **Recall :**

- Recall is crucial for churn detection because missing an actual churner is more costly to a business than incorrectly flagging a loyal customer.

4. **F1 Score :**

- It balances both false positives and false negatives and is more reliable than accuracy for imbalanced data.

5. **ROC-AUC Score (Primary Metric) :**

- It handles class imbalance better than accuracy.
- It reflects true ranking capability of churn probabilities.

6. **Confusion Matrix :**

- Confusion matrices were displayed for each model to visualize:
    - ➔ True positives (correct churn predictions)
    - ➔ True negatives (correct non-churn predictions)
    - ➔ False positives
    - ➔ False negatives

- This supports business interpretation by showing:
    - ➔ how many loyal customers were wrongly targeted
    - ➔ how many churners were missed

b. Overfitting Detection

1. Decision Tree Classifier

The results of the baseline Decision Tree model show a clear indication of overfitting.

- **Training Accuracy:** ~99.88%
- **Testing Accuracy:** ~71.29%

This large performance gap indicates that the model has memorized patterns in the training data rather than learning generalizable relationships. Decision Trees naturally tend to overfit when not pruned or controlled, especially when allowed to grow deep. The tuned Decision Tree model, with limited depth and adjusted hyperparameters, significantly reduced the overfitting effect, achieving a more balanced performance across training and testing data.

Key Indicators of Overfitting (DT):

- Extremely high training accuracy (>99%)
- Significantly lower test accuracy (~71%)
- Large difference between training (1.0) and test ROC-AUC (0.6425)

This shows that the baseline tree was too complex, while the tuned model improved generalization.

2. Neural Network Model

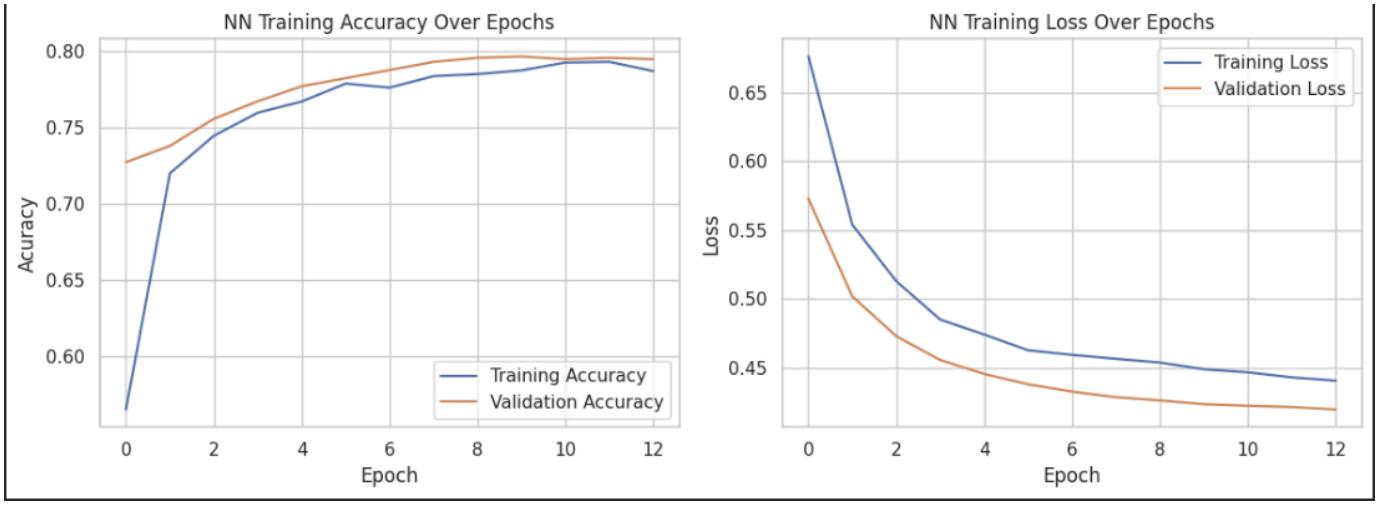


Figure 12 : Accuracy and loss curves NN

The Neural Network did not show overfitting, and the training curves proves that:

- **Training vs Validation Accuracy:** Both curves increase steadily and remain close to each other and Validation accuracy does not diverge or drop after several epochs.

- **Training vs Validation Loss:** Both curves decrease smoothly without sudden spikes, with validation loss consistently lower or similar to training loss.

This pattern shows that the model is learning meaningful patterns without memorizing the training data. The use of:

- Batch Normalization
- Dropout layers - [[64,32], [128,64], [128, 64, 32]]
- Early stopping
- Moderate number of epochs (20)

helped the NN maintain good generalization. This confirms that the Neural Network is stable and well-generalized for the churn prediction task.

5. Model Evaluation

a. Decision Tree Result

Baseline Decision Tree Performance

The baseline Decision Tree model shows clear signs of overfitting, with extremely high training performance but noticeably lower results on the test set.

Training Performance

- Training Accuracy**: 99.8756%**
- Training ROC-AUC**: 1.0**
- Precision (Train):
  - ➔ Class 0: **1.0**
  - ➔ Class 1: **1.0**
- Recall (Train):
  - ➔ Class 0: **1.0**
  - ➔ Class 1: **1.0**
- **F1-Score (Train):**
  - ➔ Class 0**: 1.0**
  - ➔ Class 1: **1.0**

- **Confusion Matrix (Train Set)**

The training confusion matrix shows that the model performs extremely well on the training data, almost perfectly separating churn and non-churn customers:

➔ **True Negatives (4129)**: Almost all non-churn customers were correctly identified.
➔ **False Positives (1)**: Only one non-churn customer was incorrectly predicted as churn.
➔ **False Negatives (6)**: Very few churn customers were missed.
➔ **True Positives (1489)**: Most churn customers were correctly classified.

Interpretation:
The near-perfect separation indicates that the tree memorized patterns in the training data rather than learning generalizable decision boundaries. This behaviour is characteristic of **overfitting**, especially because decision trees easily grow highly specific decision boundaries unless restricted.

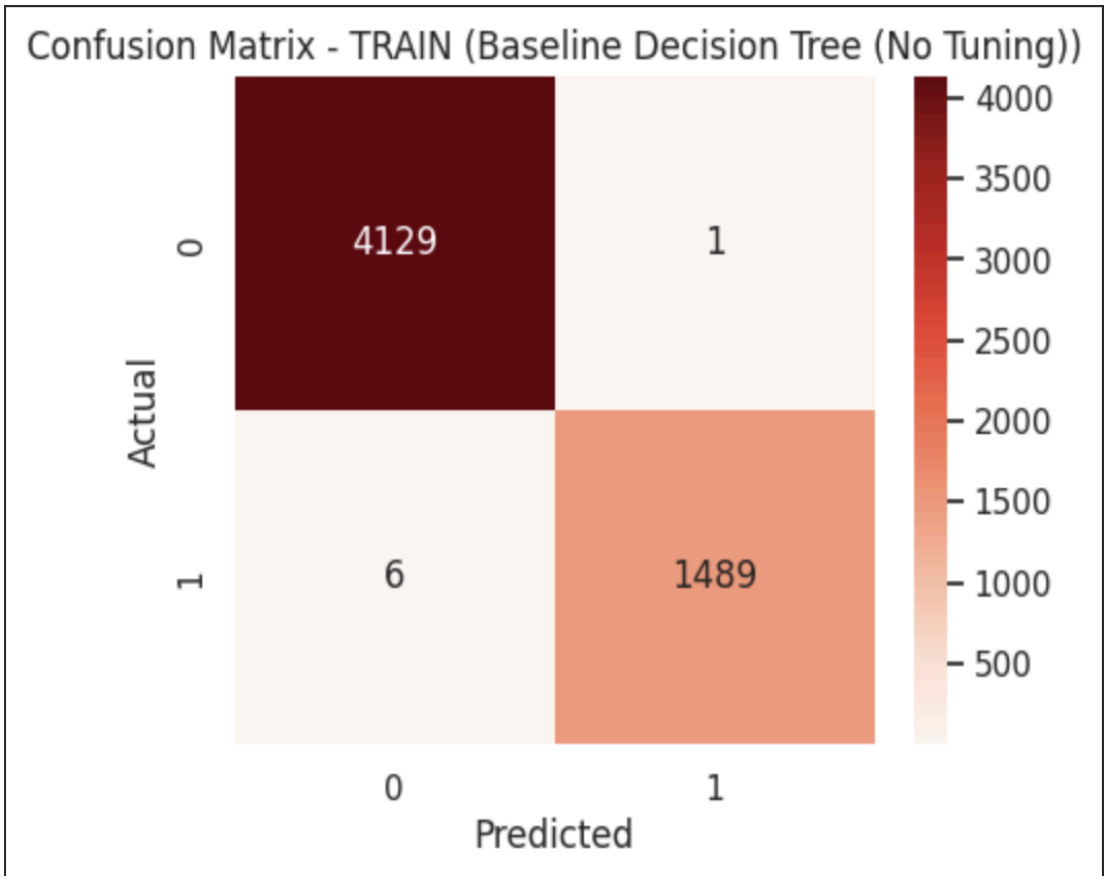


Figure 13 :Confusion matrix -DT- Train(Base)

Test Performance

- Training Accuracy**: 71.2864%**
- Training ROC-AUC**: 0.6452**
- Precision (Train):
  - ➔ Class 0: **0.81**
  - ➔ Class 1: **0.46**
- Recall (Train):
  - ➔ Class 0: **0.79**
  - ➔ Class 1: **0.50**
- **F1-Score (Test):**
  - ➔ Class 0**: 0.80**
  - ➔ Class 1: **0.48**
- **Confusion Matrix (Test Set)**
  - ➔ **True Negatives (816):** The model still identifies many non-churn customers correctly.
  - ➔ **False Positives (217):** A large number of non-churn customers were incorrectly predicted as churn.
  - ➔ **False Negatives (187):** Half of the actual churn customers were missed.
  - ➔ **True Positives (187):** Only a small portion of churn cases were correctly predicted.

  Interpretation:

  Unlike the training set, the model fails to generalize well:

  - High **false positives** → model incorrectly flags many loyal customers as churn.
  - High **false negatives** → model misses many churners, reducing business value.
  - Balanced true positives and false negatives show **weak churn detection ability**.

  This confirms **severe overfitting**, consistent with:

  - Large difference between training and testing accuracy
  - Significant drop in ROC-AUC
  - Poor minority-class performance (Class 1: Churn)

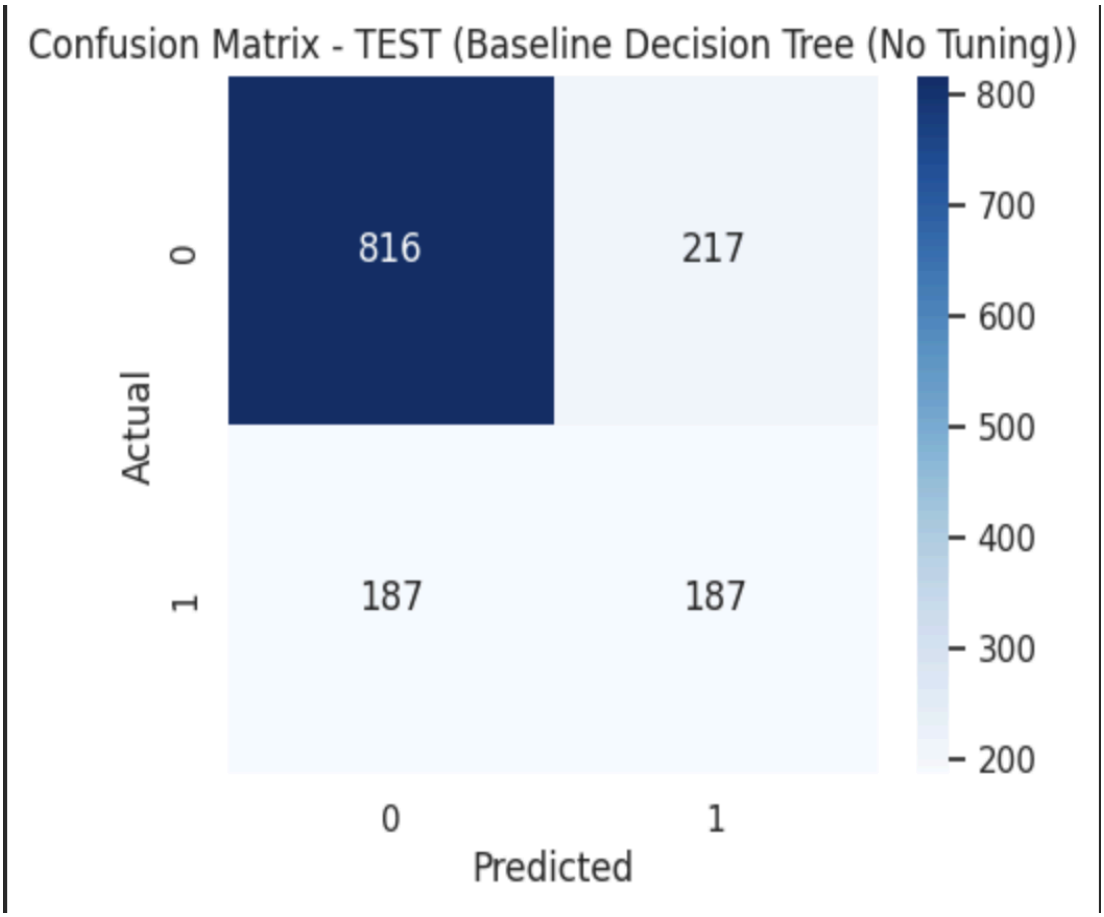


Figure 14 :Confusion matrix -DT- Test(Base)

Tuned Decision Tree Performance

The tuned Decision Tree model shows a clear improvement in generalization compared to the baseline model. Hyperparameter tuning helped reduce overfitting and improved the model's ability to detect churners.

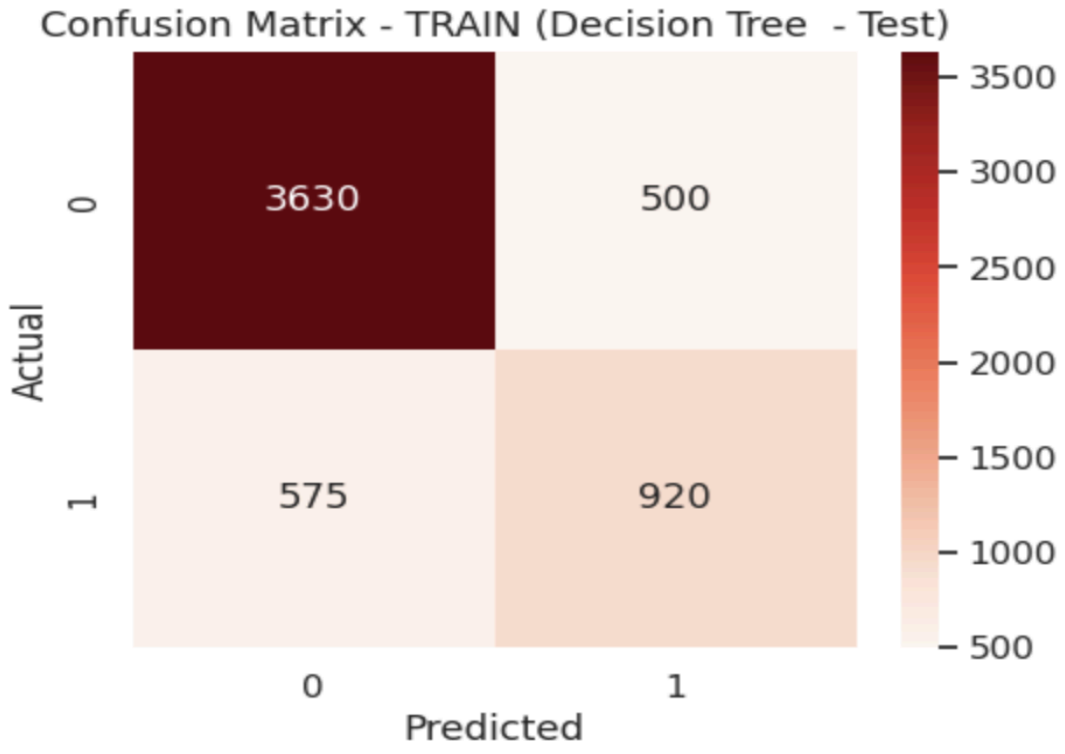
Training Performance

- **Training Accuracy**: 80.89%
- **Training ROC-AUC**: 0.8745

- **Precision (Train)**
  - ➔ Class 0 (No Churn): 0.86
  - ➔ Class 1 (Churn): 0.65
- **Recall (Train)**
  - ➔ **Class 0:** 0.88
  - ➔ **Class 1:** 0.62
- **F1-Score (Train)**
  - ➔ **Class 0:** 0.87
  - ➔ **Class 1:** 0.63
- **Confusion Matrix (Train)**
  - ➔ **True Negatives:** 3630
  - ➔ **True Positives:** 920
  - ➔ **False Positives:** 500
  - ➔ **False Negatives:** 575

Meaning:

- The model correctly identifies most non-churners.
- It detects churners reasonably well, reducing FN compared to baseline.
- Misclassifications between classes are more balanced now.



Interpretation (Train)

The tuned model still performs better on the majority class but does a **significantly better job recognizing churners** compared to the baseline.

Overfitting is reduced because training accuracy is not extremely high.

Test Performance

- **Test Accuracy:** 78.82%
- **Test ROC-AUC:** 0.8226
- **Precision (Test)**
  - ➔ Class 0: 0.60
  - ➔ Class 1: 0.58
- **Recall (Test)**
  - ➔ Class 0: 0.79
  - ➔ Class 1: 0.59
- **F1-Score (Test)**
  - ➔ Class 0: 0.69
  - ➔ Class 1: 0.59
- **Confusion Matrix (Test)**
  - ➔ **True Negatives:** 889
  - ➔ **True Positives:** 220
  - ➔ **False Positives:** 144
  - ➔ **False Negatives:** 154

  Meaning:

  - The model correctly identifies many churners (220), much better than the baseline.
  - False negatives (missed churners) are reduced.
  - Error distribution is more balanced, showing improved model generalization.

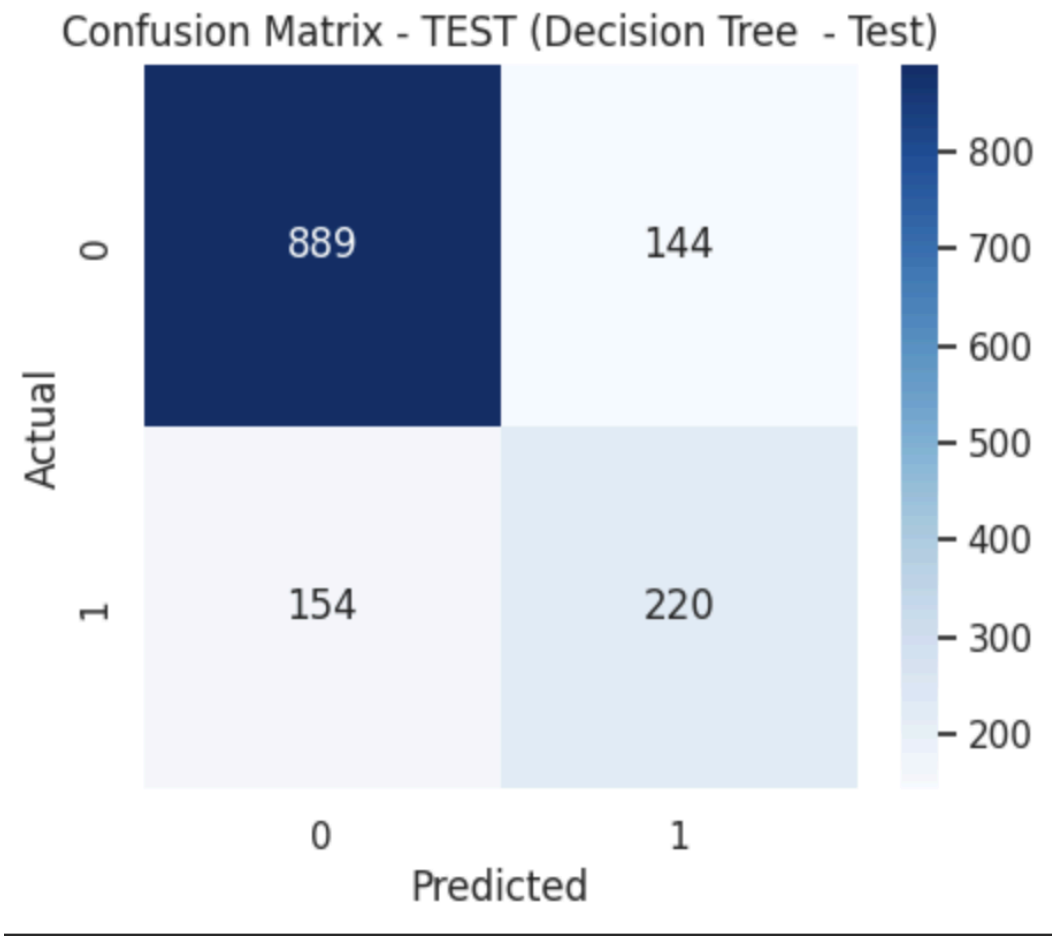


Figure 15 :Confusion matrix -DT- Test(Tune)

Interpretation (Test)

The tuned model achieves **balanced performance** between classes and performs far better on the minority churn class compared to the baseline.

- **Precision–Recall Curve**

The PR curve shows a PR-AUC = 0.63, meaning the model maintains reasonable precision as recall increases, which is especially important for churn prediction where positive cases (churners) are rare.

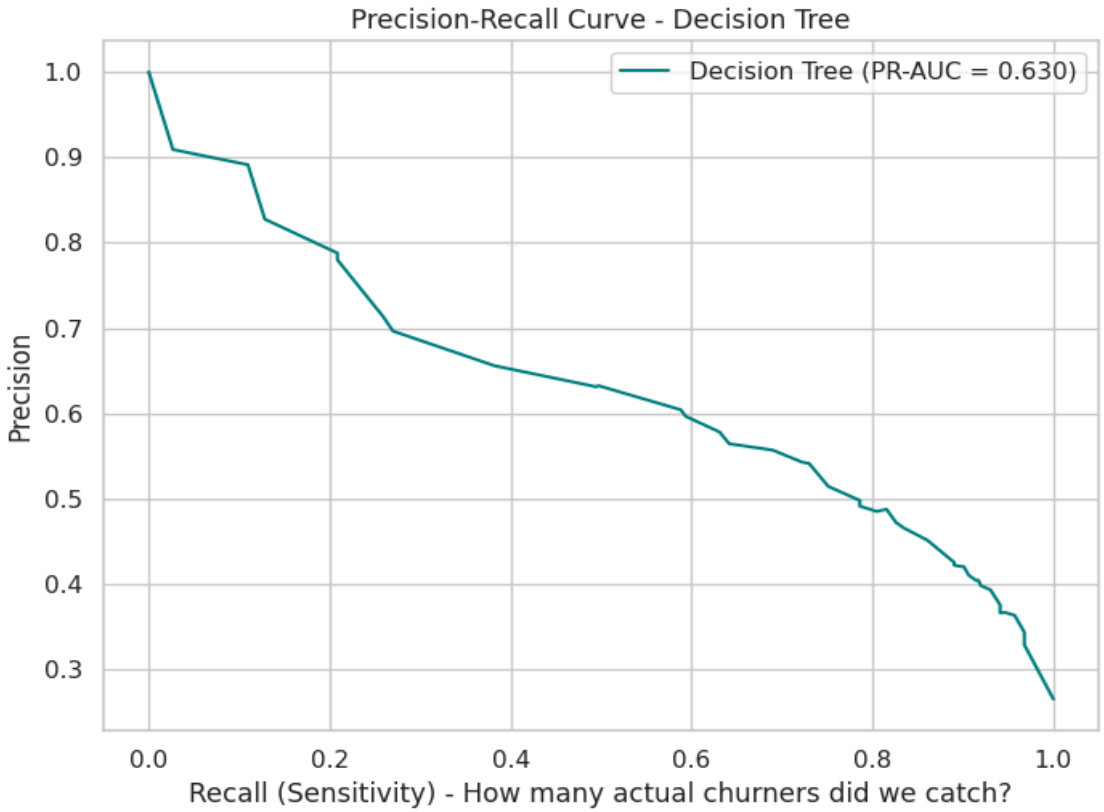


Figure 16 :Precision–Recall Curve DT

- **Feature Importance (Top Influential Features)**

  Interpretation:

❖ Contract type is the strongest churn predictor (long-term contracts → less churn)

❖ Billing-related variables (charges and tenure) strongly affect churn probability.

❖ Fiber optic users show a higher churn tendency.

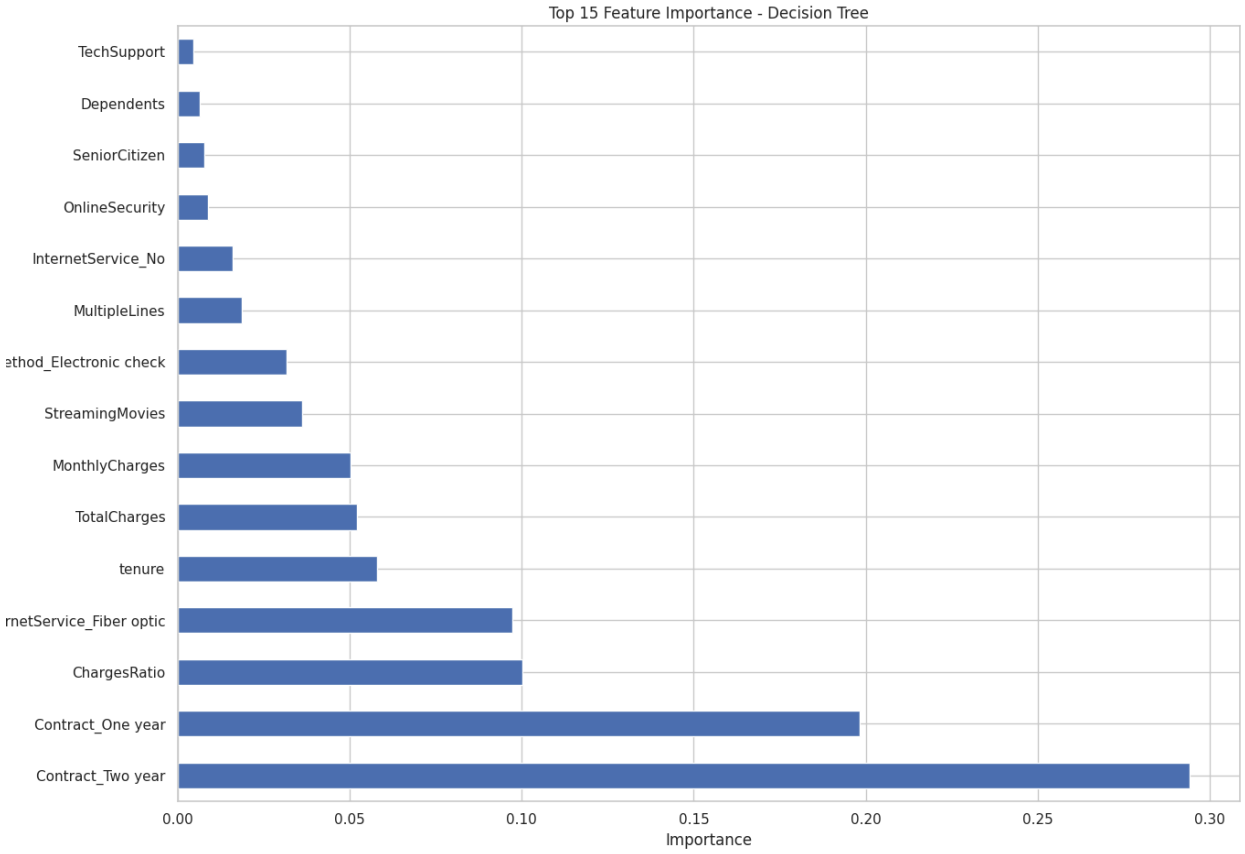


Figure 17 :Feature Importance DT

Overall Evaluation

The tuned Decision Tree delivers a substantial improvement over the baseline:

- ❖ Higher test accuracy
- ❖ Much higher ROC-AUC
- ❖ Better minority-class detection
- ❖ Reduced overfitting
- ❖ More generalizable confusion matrix results

While not as strong as your Neural Network, the tuned DT is a **well-performing and interpretable** model.

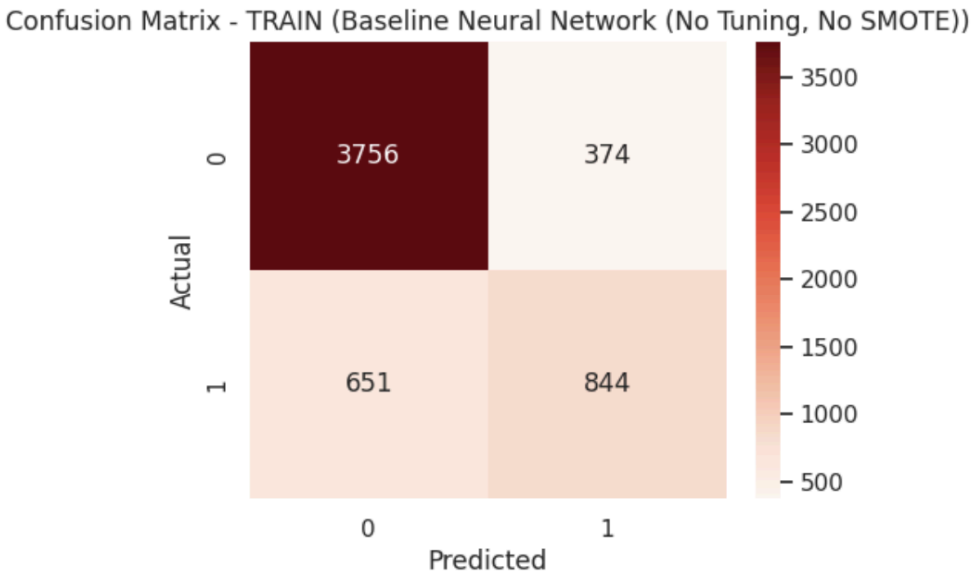
b. Neural Network Model Result

Baseline Neural Network (No Tuning)

The baseline Neural Network demonstrates moderate predictive strength, performing better than the Decision Tree baseline,especially in terms of ROC-AUC and balanced classification across both classes. However, the model still shows mild underfitting, as reflected in relatively close training and testing performance.

Training Performance

- **Training Accuracy:** 81.778%
- **Training ROC-AUC:** 0.8649
- **Precision (Train):**
  - Class 0 (No Churn): 0.85
  - Class 1 (Churn): 0.69
- **Recall (Train):**
  - Class 0: 0.91
  - Class 1: 0.56
- **F1-Score (Train):**
  - Class 0: 0.88
  - Class 1: 0.62
- **Confusion Matrix Interpretation (Train):**

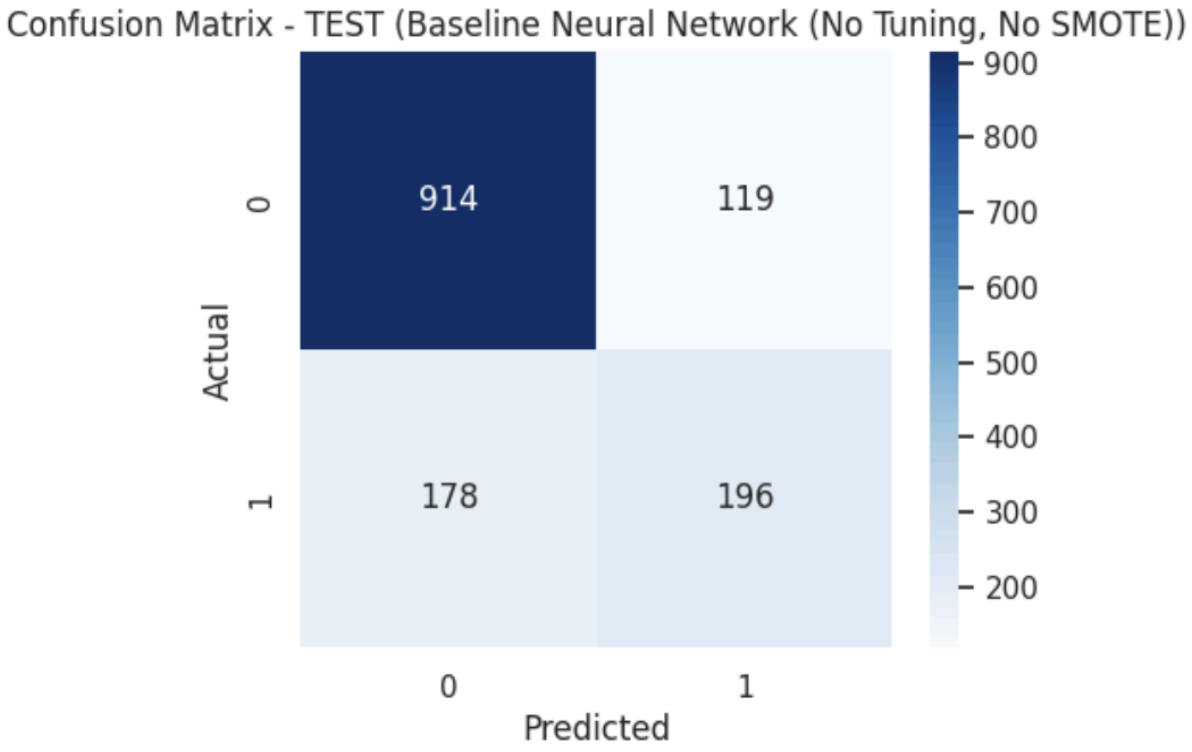


➔ The model correctly identifies most non-churn customers.
➔ It struggles more with minority churn cases, misclassifying **651 churners** as non-churn.

Test Performance

- **Test Accuracy:** ~78.7%
- **Test ROC-AUC:** ~0.828
- **Precision (Test):**
  - Class 0: 0.84
  - Class 1: 0.62
- **Recall (Test):**
  - Class 0: 0.88
  - Class 1: 0.51
- **F1-Score (Test):**
  - Class 0: 0.86
  - Class 1: 0.56

Confusion Matrix Interpretation (Test):



➔ The model generalizes well and maintains performance close to training.

➔ Although detection of churners (class 1) improves compared to the Decision Tree baseline, recall remains moderate.

Tuned Neural Network

The tuned Neural Network is the best-performing model among all models tested.

With optimized layers, dropout, learning rate, and epochs, the model achieves the highest ROC-AUC and stable generalization.

Training Performance

- **Training Accuracy:** 80.5867%
- **Training ROC-AUC:** 0.8517
- **Precision (Train):**
    - Class 0: 0.85
    - Class 1: 0.66
- **Recall (Train):**
    - Class 0: 0.90
    - Class 1: 0.55
- **F1-Score (Train):**
    - Class 0: 0.87
    - Class 1: 0.60
- **Confusion Matrix Interpretation (Train):**

➔ The model correctly classified 3718 non-churn customers and 815 churn customers.

➔ 412 false positives: customers predicted to churn but actually stayed.

➔ 680 false negatives: churn customers incorrectly predicted as non-churn.

➔ The model handles both classes more evenly compared to the baseline NN, though the majority class (non-churn) is still easier to predict.

➔ Since the training and test results are similar, the model is not overfitting.

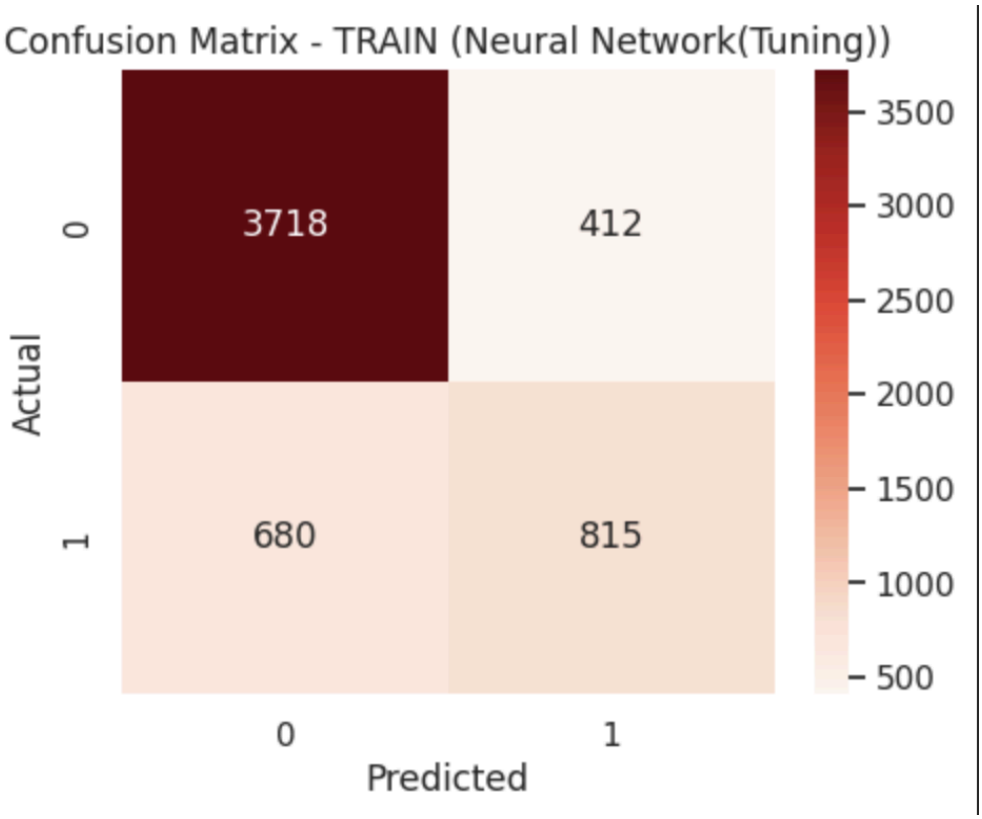


Figure 18 :Confusion Matrix Interpretation (Train)

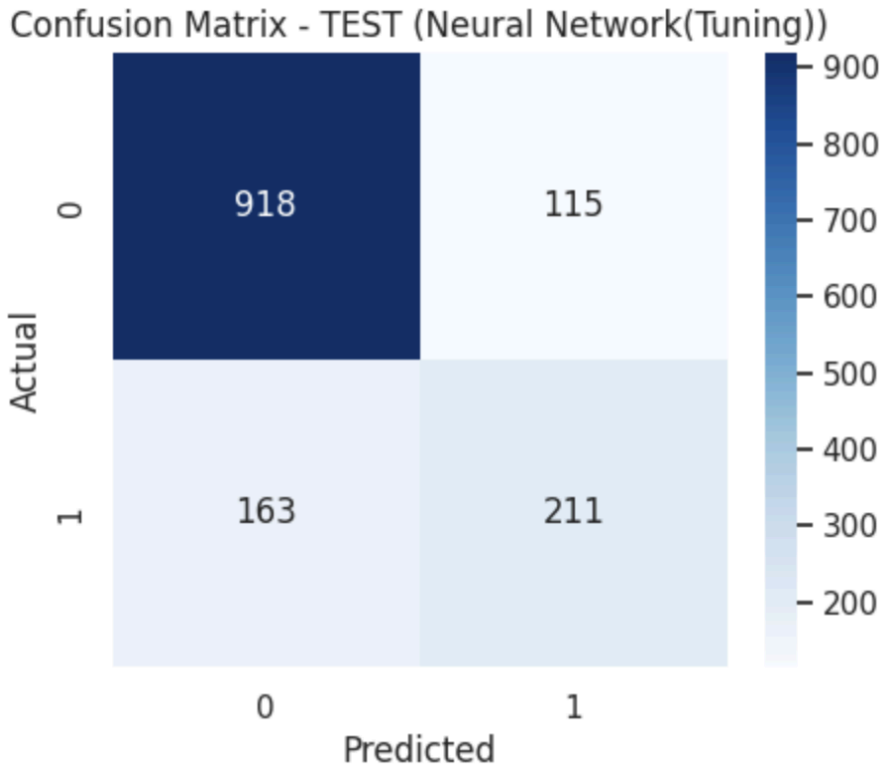
Test Performance

- **Test Accuracy: 79.6%**
- **Test ROC-AUC: 0.8325 (highest among all models)**
- **Precision (Test):**
    - Class 0: 0.85
    - Class 1: 0.63
- **Recall (Test):**
    - Class 0: 0.85
    - Class 1: 0.55
- **F1-Score (Test):**
    - Class 0: 0.85
    - Class 1: 0.59

- **Confusion Matrix Interpretation (Test):**

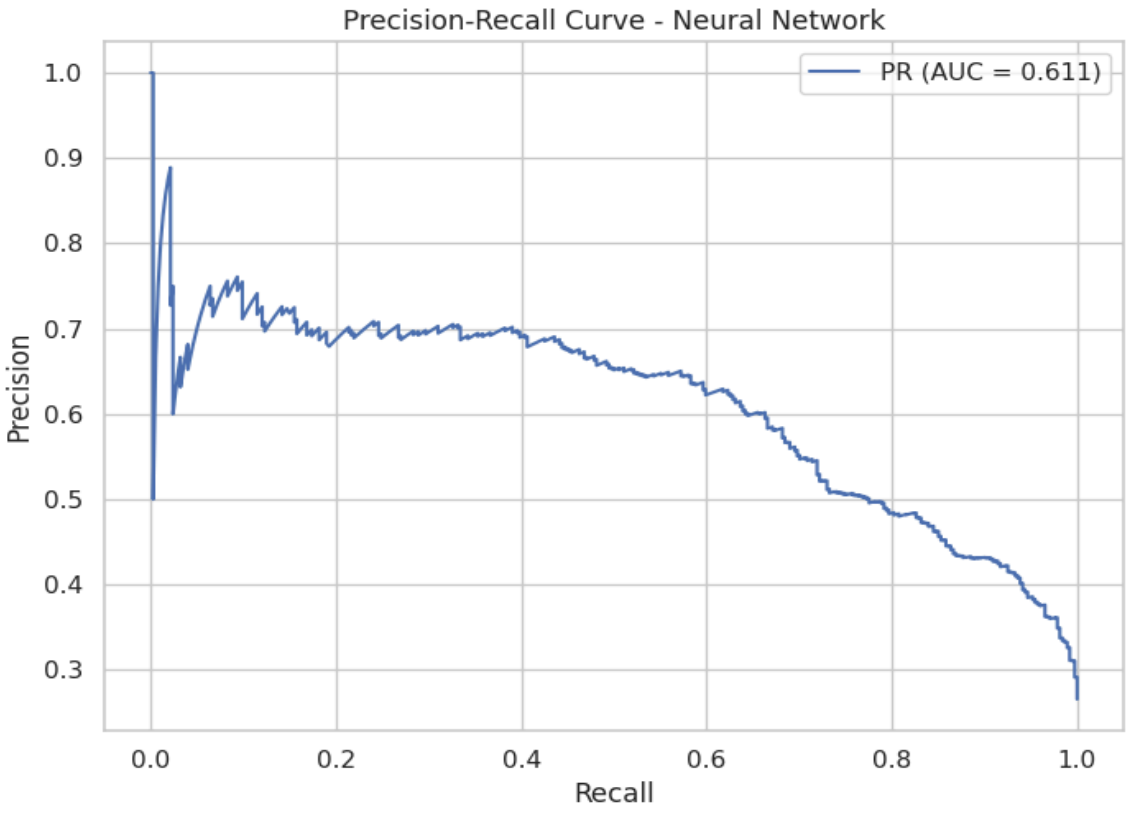
Interpretation

- The model performs well on unseen data:
  - **918 true negatives:** correctly predicted non-churn.
  - **211 true positives:** correctly predicted churn.
- **115 false positives:**
  - These are customers incorrectly predicted as churn.
  - This is acceptable because false positives are less harmful than missing actual churners.
- **163 false negatives:**
  - These are actual churners the model failed to detect.
  - This number is significantly lower than the baseline NN and Decision Tree baseline -> strong improvement.



- **Precision–Recall Curve (Tuned NN)**

  ➔ **PR-AUC ~ 0.611**
  ➔ Indicates a stable ability to catch churners with reasonable precision.



Overall Conclusion

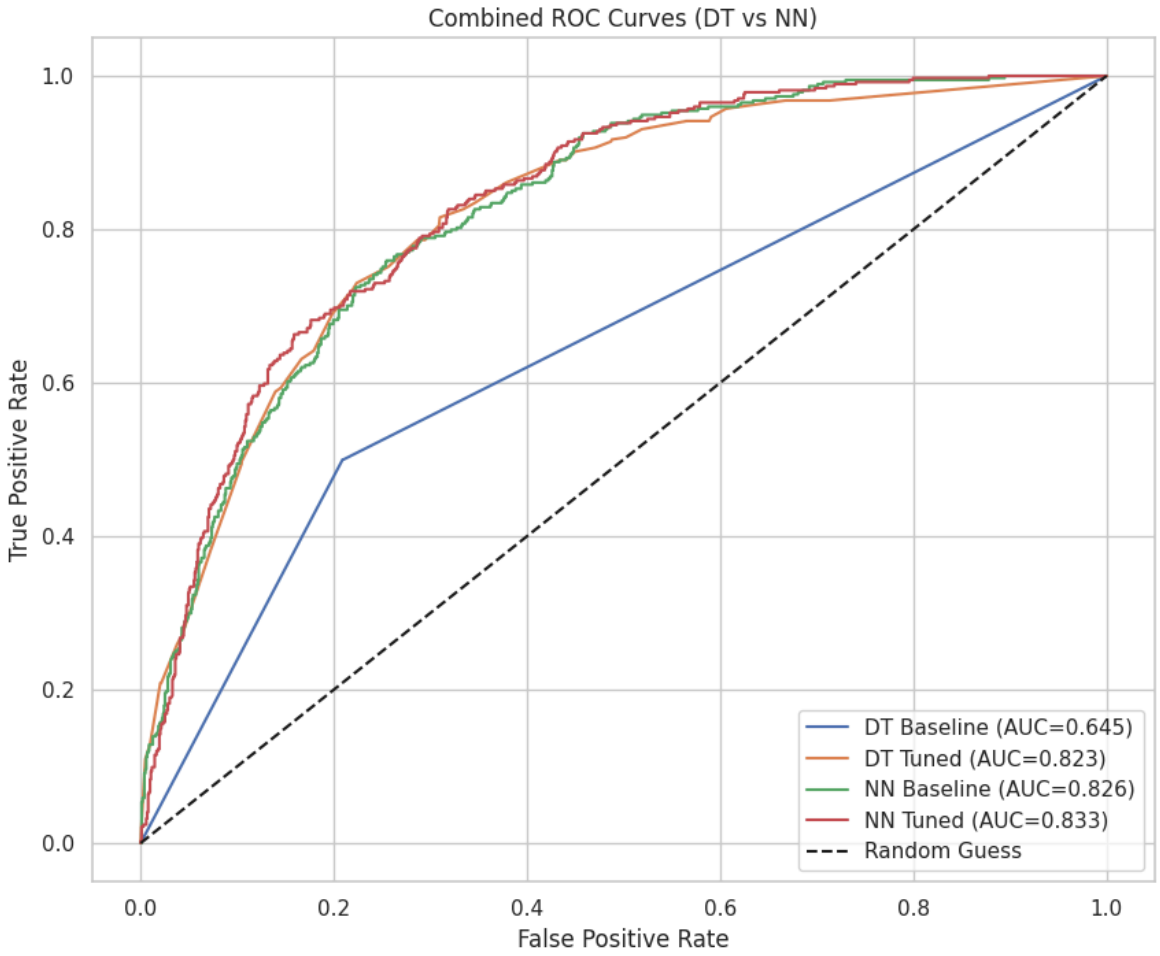
- The tuned Neural Network shows the best balance between predicting churners and non-churners.
- The reduction in false negatives compared to the other models makes it the most suitable model for customer churn prediction, because detecting churners is often the primary business goal.
- The confusion matrices show good generalization, minimal overfitting, and stable behavior on new data.

c. ROC-AUC Comparison Results

The combined ROC-AUC plot compares the classification performance of all four main models: **Decision Tree Baseline, Decision Tree Tuned, Neural Network Baseline, and Neural Network Tuned**. ROC-AUC measures how well a model separates churners from non-churners across all classification thresholds. Higher curves and higher AUC values indicate better discrimination ability.

- Model AUC Scores
  - **Decision Tree Baseline:** 0.645
  - **Decision Tree Tuned:** 0.823
  - **Neural Network Baseline:** 0.826
  - **Neural Network Tuned: 0.833** (Highest)

- Interpretation:
  - The **baseline Decision Tree** performs the worst, with a low AUC (0.645) and a curve close to the random-guess line, indicating limited predictive power.
  - **Tuning the Decision Tree** significantly improves performance, increasing the AUC to 0.823. The ROC curve becomes steeper, showing better identification of churners.
  - The **baseline Neural Network** already performs strongly (AUC = 0.826), outperforming both decision tree versions except the tuned one.
  - The **Tuned Neural Network achieves the best overall performance**, with the highest ROC-AUC (0.833).
    - ➔ Its ROC curve stays consistently above all other models.
    - ➔ This indicates superior ability to distinguish churners from non-churners across thresholds.
- Conclusion
  - The ROC-AUC comparison clearly shows that:
    - ➔ **The Tuned Neural Network is the best-performing model**, offering the strongest generalization and the most reliable churn discrimination.
    - ➔ The Tuned Decision Tree performs well, but the Neural Network maintains a slight but clear advantage in predictive power.



d. Overall Result Comparison

A comprehensive comparison of all four models-Decision Tree Baseline, Decision Tree Tuned, Neural Network Baseline, and Neural Network Tuned-was carried out using the key evaluation metrics: Accuracy, Precision, Recall, F1-Score, and ROC-AUC. The following insights summarize the performance across test data.

1. Test Accuracy Comparison

- NN Tuned (0.80) achieved the highest accuracy, slightly outperforming NN Baseline (0.789) and DT Tuned (0.788).
- DT Baseline (0.713) performed the lowest, mainly due to overfitting on the training set.
- **Winner: Neural Network (Tuned)**

2. Precision Comparison

- NN Tuned (0.635) and NN Baseline (0.620) achieved the highest precision for the churn class.
- DT Baseline (0.463) was the lowest, indicating many false positives.
- **Winner: Neural Network (Tuned)**

3. Recall Comparison

- DT Tuned (0.588) performed slightly better than NN models in recall for the churn class.
- NN Tuned achieved 0.564, showing a good balance but not the highest.
- **Winner:** Decision Tree (Tuned)

4. F1-Score Comparison

- NN Tuned (0.603) achieved the best F1-score, demonstrating a strong balance between precision and recall.
- NN Baseline (0.569) and DT Tuned (0.596) were close, but DT Baseline poorly scored (0.481).
- **Winner: Neural Network (Tuned)**

5. ROC-AUC Comparison

- NN Tuned (0.834) achieved the highest ROC-AUC, indicating the strongest ability to separate churn vs. non-churn classes.
- NN Baseline (0.826) and DT Tuned (0.823) also performed well.
- DT Baseline (0.645) showed weak discrimination.
- **Winner: Neural Network (Tuned)**

Overall Conclusion

The Tuned Neural Network is the best-performing model across most evaluation metrics. It provides:

- Highest **test accuracy**
- Highest **F1-score**
- Highest **ROC-AUC**
- Stronger generalization compared to Decision Trees

Meanwhile, the **Tuned Decision Tree** offers competitive recall and reasonable ROC-AUC but does not outperform the neural models overall.

The **Baseline Decision Tree** significantly overfits (Train Accuracy ~ 99%, Test Accuracy ~ 71%), making it unsuitable as a final model.

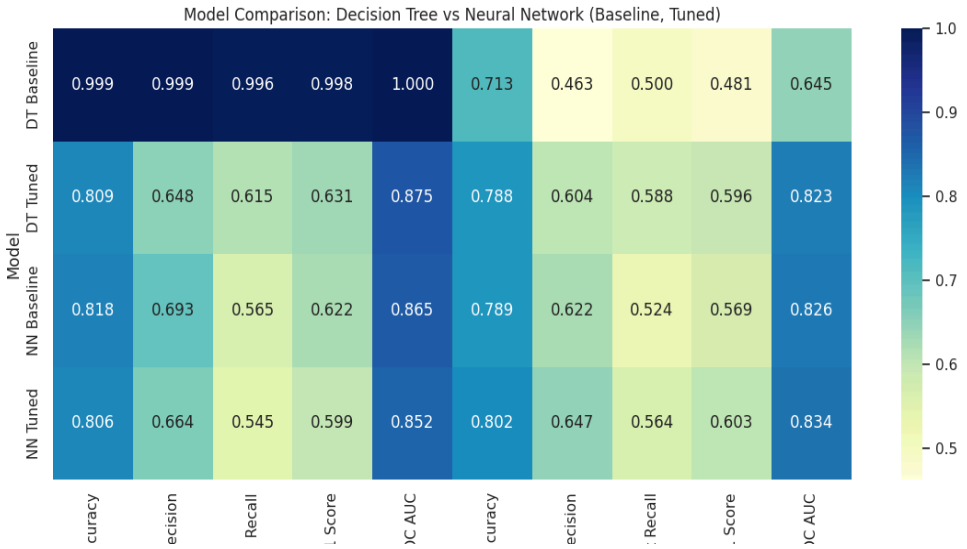


Figure 19 :Comparison Table

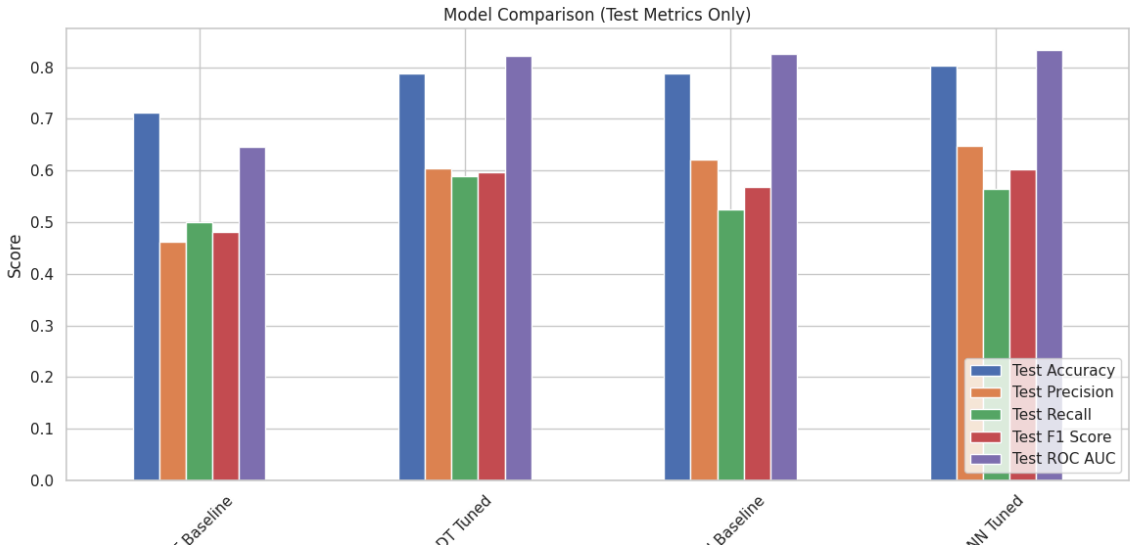


Figure 20 :Comparison BarChart

6. Ethical Considerations and Post-Deployment Strategy

6.1 Ethical Strategies Followed During Model Development

During the development of the Decision Tree and Neural Network models, several ethical principles were considered to ensure responsible use of AI:

- **Fairness and Bias Awareness**
  - ➔ The dataset was checked for imbalance (majority “No Churn”).
  - ➔ SMOTE was used only for experimentation and not included in the final comparison to avoid artificially inflated performance.
  - ➔ No sensitive personal attributes were included, reducing risk of discriminatory outcomes.

- **Transparency and Explainability**
  - ➔ Feature importance (Decision Tree) and performance metrics (confusion matrices, ROC/PR curves) were included to help explain how the models behave.
  - ➔ Clear reporting of limitations prevents misinterpretation of model capability.
- **Responsible Data Handling**
  - ➔ Data used is publicly available and anonymised.
  - ➔ A proper train-test split was applied before preprocessing to avoid information leakage.
- **Preventing Misleading Results**
  - ➔ Both training and testing metrics were evaluated to detect overfitting.
  - ➔ Hyperparameter tuning (GridSearch) was used to improve performance without bias.

6.2 Post-Deployment Strategy

Once deployed, the model requires ongoing ethical and operational oversight:

- **Continuous Monitoring**
  - ➔ Regularly track model accuracy, recall, and ROC-AUC to identify performance drops or concept drift.
- **Scheduled Retraining**
  - ➔ Retrain the model periodically using new customer data to maintain relevance
- **Privacy and Compliance**
  - ➔ Ensure customer data is stored securely and used only for intended business purposes.

- **Feedback Integration**

  - ➔ Use real churn outcomes and business insights to refine decision thresholds or retraining schedules.

7. Experimental Results

The experiments compared four models, **Baseline Decision Tree, Tuned Decision Tree, Baseline Neural Network, and Tuned Neural Network**, using consistent evaluation metrics (Accuracy, Precision, Recall, F1-score, and ROC-AUC).Results show that **hyperparameter tuning improved both models**, especially in generalisation ability.

The **Tuned Neural Network** achieved the best overall test performance with:

- **Accuracy ~80%**
- **Precision ~ 0.64**
- **Recall ~ 0.56**
- **F1-score ~ 0.60**
- **ROC-AUC ~ 0.834** (highest among all models)

The **Tuned Decision Tree** also improved significantly over the baseline, achieving:

- **Accuracy ~ 78.8%**
- **F1-score ~ 0.596**
- **ROC-AUC ~ 0.823**

Across all metrics, the baseline models underperformed, mainly due to limited complexity (DT) or insufficient training epochs and optimisation (NN).

The combined ROC-AUC curve clearly showed that NN Tuned > NN Baseline ≈ DT Tuned > DT Baseline, indicating stronger predictive ability and better class separation for the tuned Neural Network.

8. Limitation in the Models

Despite improved performance after tuning, several limitations remain:

➔ **Class Imbalance**

- The dataset is naturally imbalanced (~73% No churn, 27% Yes).
- Although training used the original imbalance (SMOTE shown only experimentally), both models struggled with recall for the churn class, leading to more false negatives.

➔ **Decision Tree Overfitting**

- The baseline Decision Tree showed:
  - **Training accuracy ~ 99%**
  - **Test accuracy ~ 71%**

    This indicates strong overfitting, as the tree memorised patterns instead of learning general rules.

➔ **Neural Network Sensitivity**

- The Neural Network requires:
  - Careful tuning of epochs
  - Learning rate adjustments
  - Enough training iterations

➔ **Limited Explainability for NN**

- Decision Trees offer feature importance, but Neural Networks act as a “black box,” making it harder to understand the exact decision boundaries.

9. Further Enhancements

To improve the model performance and strengthen generalisation, the following enhancements are recommended:

1. **Use Balanced Training Techniques**

   Although SMOTE was demonstrated experimentally, future work can include:

   - SMOTE + Tomek Links
   - Class-weighted loss in Neural Network
   - Cost-sensitive Decision Trees

   These can help improve churn recall without sacrificing accuracy.

2. **Build More Advanced Models**

   Consider using:

   - **Random Forest**
   - **XGBoost / LightGBM**
   - **Deep Neural Networks**

   These techniques usually outperform single trees and shallow networks on tabular data.

3. **Hyperparameter Tuning with Search Algorithms**

   Use:

   - RandomizedSearchCV
   - Bayesian Optimization
   - Optuna

   For better NN architecture tuning (layers, neurons, activations, dropout rates).

10. GIT Repository

Link: https://github.com/Yenuli0808/CM2604_Telco_Customer_Churn-CW.git

11. Appendix - Source Code

Task 01: Exploratory Data Analysis (EDA)

1: Import Libraries

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
sns.set_context("talk")
%matplotlib inline
sns.set_style('whitegrid')
from numpy._core.defchararray import title
print("✅ Libraries imported successfully!")
```

2:Import DataSet From GIT Hub

```
# Load the dataset FROM GitHub
!wget -q
https://raw.githubusercontent.com/Yenuli0808/CM2604_Telco_Customer_Churn-CW/main/data/Telco-Customer-Churn.csv
# Load the data
df = pd.read_csv('Telco-Customer-Churn.csv')
print("✅ Dataset loaded successfully!")
print("\n=== Data set overview ===")
print(f"Dataset shape: {df.shape}")  # (rows, columns)
```

1. 3:Initial Inspections

```
# First look at the data
print("==== FIRST 5 ROWS ====\n")
display(df.head(20))
```

```
print("==== BASIC INFO ====")
df.info()
print("==== MISSING VALUES ====\n")
print(df.isnull().sum())
print(df.isnull().sum()[df.isnull().sum() > 0])
print("==== DUPLICATE ROWS ====\n")
print(df.duplicated().sum())
print("==== UNIQUE VALUES ====\n")
print(df.nunique())
```

```
print("\n=== CHECKING FOR PLACEHOLDER MISSING VALUES ===")
print("\n Unique values in each column: \n")
for col in df.columns:
    unique_vals = df[col].unique()
    if len(unique_vals) < 10:  # Only show columns with few unique values
        print(f"{col}: {unique_vals}")
# Total charges are often stored as object -> converting it to numeric
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
n_nan_tc=df['TotalCharges'].isna().sum()
print(f"\nNumber of NaN values in 'TotalCharges'after conversion: {n_nan_tc}")
if n_nan_tc>0:
  median_tc = df['TotalCharges'].median()
  print("Imputing TotalCharges NaN with median: ",round(median_tc,2))
  df['TotalCharges'].fillna(median_tc, inplace=True)
#ensuring Churn values are exactly yes/no
print("\nUnique values in 'Churn' column: ",df['Churn'].unique())
df['Churn_num'] = df['Churn'].map({'No': 0, 'Yes': 1})
# Remove customerID (not used in EDA computations but keep raw file untouched)
if 'customerID' in df.columns:
    #  not permanently drop from original df; create working copy
    df_work = df.drop(columns=['customerID']).copy()
else:
    df_work = df.copy()
print("Working dataframe shape:", df_work.shape)
# Basic Dataset Summary
numeric_cols = df.select_dtypes(include=['int64','float64']).columns.tolist()
cat_cols = df.select_dtypes(include=['object']).columns.tolist()
print("Numeric columns:", numeric_cols)
print("\nCategorical columns:", cat_cols)
# Fix TotalCharges for analysis
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
df['TotalCharges'].fillna(df['TotalCharges'].median(), inplace=True)
```

Target Variable : Churn Distribution

```
print("==== TARGET BALANCE ====\n")
print("Target distribution (counts & percent):")
churn_dist = pd.DataFrame({
    "Count": df['Churn'].value_counts(),
    "Percentage (%)": (df['Churn'].value_counts(normalize=True)*100).round(2)
})
churn_dist
# Churn Distribution pie chart
plt.figure(figsize=(8,6))
df['Churn'].value_counts().plot(kind='pie', autopct='%1.2f%%',colors=['teal','orange'])
plt.title('Churn Distribution (Pie Chart)')
plt.ylabel('')
plt.show()
#Churn Distribution bar Chart
```

```
plt.figure(figsize=(8,6))
sns.countplot(x='Churn', data=df, palette=['teal','orange'])
plt.title('Churn Distribution (Bar Chart)')
plt.show()
df['Churn'].value_counts()
# Numerical Summary
print("==== Numerical Features Summary ====\n")
display(df_work[numeric_cols].describe().T)

# Categorical  Summary
df.describe(include='object').T

#Numerical distributional plots
for col in ['tenure','MonthlyCharges','TotalCharges']:
    plt.figure(figsize=(10,4))
    sns.histplot(df_work[col], kde=True, stat='count', bins=40)
    mean = df_work[col].mean()
    med = df_work[col].median()
    plt.axvline(mean, color='red', linestyle='--', label=f"Mean: {mean:.2f}")
    plt.axvline(med, color='green', linestyle='--', label=f"Median: {med:.2f}")
    plt.title(f"Distribution of {col} (Counts)")
    plt.legend()
    plt.show()
# Categorical Distributions
categorical_cols = ['gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'Contract',
'PaymentMethod']
fig, axes = plt.subplots(2, 4, figsize=(20,10))
axes = axes.flatten()
for i, col in enumerate(categorical_cols):
    sns.countplot(x=col, data=df, ax=axes[i], palette='viridis')
    axes[i].set_title(f'Distribution of {col}')
    axes[i].tick_params(rotation=45)
plt.tight_layout()
plt.show()
```

Correlation Heatmap

```
plt.figure(figsize=(10,8))
corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True,fmt=".2f", cmap="coolwarm")
plt.title("Correlation matrix (numeric features)")
plt.show()
# Assuming 'Churn_num' is the numeric representation of churn (0/1)
print(corr['Churn_num'])
# Correlation with Churn_num
print("Correlation with Churn (numeric):")
display(corr['Churn_num'].sort_values(ascending=False))
```

Advanced EDA: Categorical Feature vs Churn

Box-Plots

```
# Tenure vs Churn
```

```
plt.figure(figsize=(10,6))
sns.boxplot(x='Churn', y='tenure', data=df,palette=['teal', 'orange'])
plt.title('Tenure vs Churn')
plt.show()
print("\n Churners have much lower tenure!")
# monthlyCharges vs Churn
plt.figure(figsize=(10,6))
sns.boxplot(x='Churn', y='MonthlyCharges', data=df,palette=['teal', 'orange'])
plt.title('Monthly Charges vs Churn')
plt.show()
print("\n Churners pay highly monthly charges!")

# TotalCharges vs Churn
plt.figure(figsize=(10,6))
sns.boxplot(x='Churn', y='TotalCharges', data=df,palette=['teal', 'orange'])
plt.title('Total Charges vs Churn')
plt.show()
# Summary in one
plt.figure(figsize=(18,5))
plt.subplot(1,3,1)
sns.boxplot(x='Churn', y='tenure', data=df_work,palette=['teal', 'orange'])
plt.title('tenure by Churn')
plt.subplot(1,3,2)
sns.boxplot(x='Churn', y='MonthlyCharges', data=df_work,palette=['teal', 'orange'])
plt.title('MonthlyCharges by Churn')
plt.subplot(1,3,3)
sns.boxplot(x='Churn', y='TotalCharges', data=df_work,palette=['teal', 'orange'])
plt.title('TotalCharges by Churn')
plt.tight_layout()
plt.show()
```

Key Categorical Features vs Churn

```
# Contract vs Churn (Bar Plot)
#computing percentage
ct = pd.crosstab(df_work['Contract'], df_work['Churn'], normalize='index')*100
plt.figure(figsize=(10,6))
ax = ct.plot(kind='bar', stacked=False, figsize=(10,6), color=['teal', 'orange'])
plt.title('Contract Type vs Churn (%)',fontsize=14)
plt.ylabel("Percentage (%)")
plt.xlabel("Contract Type")
plt.xticks(rotation=0)
for p in ax.patches:
   percentage = p.get_height()
   if percentage > 0:
      ax.text(p.get_x() + p.get_width()/2, percentage + 1, f"{percentage:.2f}%", ha='center', fontsize=10)
plt.legend(["No", "Yes"], title="Churn")
plt.tight_layout()
plt.show()
# Top features to show as tables
```

```
top_predictors = ['Contract']
for col in top_predictors:
    if col in df_work.columns:
        print(f"\n--- {col} churn rates (percent) ---")
        display((pd.crosstab(df_work[col], df_work['Churn'], normalize='index')*100).round(2))
print("\n Month-to-month -> Very high churn!")
# Internet Service vs Churn
ct = pd.crosstab(df_work['InternetService'], df_work['Churn'], normalize='index')*100
plt.figure(figsize=(10,6))
ax = ct.plot(kind='bar', stacked=False, figsize=(10,6), color=['teal', 'orange'])
plt.title('Internet Service vs Churn (%)',fontsize=14)
plt.ylabel("Percentage (%)")
plt.xlabel("Contract Type")
plt.xticks(rotation=0)
for p in ax.patches:
    percentage = p.get_height()
    if percentage > 0:
        ax.text(p.get_x() + p.get_width()/2, percentage + 1, f"{percentage:.2f}%", ha='center', fontsize=10)
plt.legend(["No", "Yes"], title="Churn")
plt.tight_layout()
plt.show()
# Top features to show as tables
top_predictors = ['InternetService']
for col in top_predictors:
    if col in df_work.columns:
        print(f"\n--- {col} churn rates (percent) ---")
        display((pd.crosstab(df_work[col], df_work['Churn'], normalize='index')*100).round(2))
print("\n Fiber optic -> Very high churn!")
# PaymentMethod vs Churn
ct = pd.crosstab(df_work['PaymentMethod'], df_work['Churn'], normalize='index')*100
plt.figure(figsize=(10,6))
ax = ct.plot(kind='bar', stacked=False, figsize=(10,6), color=['teal', 'orange'])
plt.title('Payment Method vs Churn (%)',fontsize=14)
plt.ylabel("Percentage (%)")
plt.xlabel("Contract Type")
plt.xticks(rotation=40)
for p in ax.patches:
    percentage = p.get_height()
    if percentage > 0:
        ax.text(p.get_x() + p.get_width()/2, percentage + 1, f"{percentage:.2f}%", ha='center', fontsize=10)
plt.legend(["No", "Yes"], title="Churn")
plt.tight_layout()
plt.show()
# Top features to show as tables
top_predictors = ['PaymentMethod']
for col in top_predictors:
    if col in df_work.columns:
        print(f"\n--- {col} churn rates (percent) ---")
```

```
        display((pd.crosstab(df_work[col], df_work['Churn'], normalize='index')*100).round(2))
print("\n Electronic check -> Very high churn!")
# OnlineSecurity vs Churn
ct = pd.crosstab(df_work['OnlineSecurity'], df_work['Churn'], normalize='index')*100
plt.figure(figsize=(10,6))
ax = ct.plot(kind='bar', stacked=False, figsize=(10,6), color=['teal', 'orange'])
plt.title('Online Security vs Churn (%)',fontsize=14)
plt.ylabel("Percentage (%)")
plt.xlabel("Contract Type")
plt.xticks(rotation=0)
for p in ax.patches:
    percentage = p.get_height()
    if percentage > 0:
        ax.text(p.get_x() + p.get_width()/2, percentage + 1, f"{percentage:.2f}%", ha='center', fontsize=10)
plt.legend(["No", "Yes"], title="Churn")
plt.tight_layout()
plt.show()
# Top features to show as tables
top_predictors = ['OnlineSecurity']
for col in top_predictors:
    if col in df_work.columns:
        print(f"\n--- {col} churn rates (percent) ---")
        display((pd.crosstab(df_work[col], df_work['Churn'], normalize='index')*100).round(2))

# TechSupport vs Churn
ct = pd.crosstab(df_work['TechSupport'], df_work['Churn'], normalize='index')*100
plt.figure(figsize=(10,6))
ax = ct.plot(kind='bar', stacked=False, figsize=(10,6), color=['teal', 'orange'])
plt.title('Tech Support vs Churn (%)',fontsize=14)
plt.ylabel("Percentage (%)")
plt.xlabel("Contract Type")
plt.xticks(rotation=0)
for p in ax.patches:
    percentage = p.get_height()
    if percentage > 0:
        ax.text(p.get_x() + p.get_width()/2, percentage + 1, f"{percentage:.2f}%", ha='center', fontsize=10)
plt.legend(["No", "Yes"], title="Churn")
plt.tight_layout()
plt.show()
# Top features to show as tables
top_predictors = ['TechSupport']
for col in top_predictors:
    if col in df_work.columns:
        print(f"\n--- {col} churn rates (percent) ---")
        display((pd.crosstab(df_work[col], df_work['Churn'], normalize='index')*100).round(2))

# SeniorCitizen vs Churn
ct = pd.crosstab(df_work['SeniorCitizen'], df_work['Churn'], normalize='index')*100
```

```
plt.figure(figsize=(10,6))
ax = ct.plot(kind='bar', stacked=False, figsize=(10,6), color=['teal', 'orange'])
plt.title('Senior Citizen vs Churn (%)',fontsize=14)
plt.ylabel("Percentage (%)")
plt.xlabel("Contract Type")
plt.xticks(rotation=0)
for p in ax.patches:
    percentage = p.get_height()
    if percentage > 0:
        ax.text(p.get_x() + p.get_width()/2, percentage + 1, f"{percentage:.2f}%", ha='center', fontsize=10)
plt.legend(["No", "Yes"], title="Churn")
plt.tight_layout()
plt.show()
# Top features to show as tables
top_predictors = ['SeniorCitizen']
for col in top_predictors:
    if col in df_work.columns:
        print(f"\n--- {col} churn rates (percent) ---")
        display((pd.crosstab(df_work[col], df_work['Churn'], normalize='index')*100).round(2))
print("Senior citizens churn more!")
# Summary in one
selected = [c for c in ['Contract','InternetService','PaymentMethod','OnlineSecurity','TechSupport','SeniorCitizen'] if c in
df_work.columns]
n = len(selected)
cols = 3
rows = (n+cols-1)//cols
fig, axes = plt.subplots(rows, cols, figsize=(cols*6, rows*4))
axes = axes.flatten()
for ax, col in zip(axes, selected):
    ct = pd.crosstab(df_work[col], df_work['Churn'], normalize='index')*100
    ct.plot(kind='bar', stacked=False, ax=ax, color=['teal', 'orange'], legend=False)
    ax.set_title(f"Churn by {col}")
    ax.set_ylabel("Percentage (%)")
    ax.set_xticklabels(ax.get_xticklabels(), rotation=30, ha='right')
    # annotate
    for p in ax.patches:
        h = p.get_height()
        if h > 0:
            ax.text(p.get_x()+p.get_width()/2, h+0.8, f"{h:.2f}%", ha='center', fontsize=9)
# remove unused axes
for i in range(len(selected), len(axes)):
    fig.delaxes(axes[i])
plt.tight_layout()
plt.show()
# compute percentage churn per category for each categorical variable(Remaining ones)
cat_cols_for_plot = df_work.select_dtypes(include=['object']).columns.tolist()
if 'Churn' in cat_cols_for_plot:
    cat_cols_for_plot.remove('Churn')  # exclude target
```

```
# Remove columns that have already been plotted in detail
excluded_cols = ['Contract', 'InternetService', 'PaymentMethod', 'OnlineSecurity', 'TechSupport', 'SeniorCitizen']
cat_cols_for_plot = [col for col in cat_cols_for_plot if col not in excluded_cols]
n = len(cat_cols_for_plot)
cols = 3
rows = (n + cols - 1) // cols  # Calculate rows needed
fig, axes = plt.subplots(rows, cols, figsize=(cols * 6, rows * 4))
axes = axes.flatten() # Flatten for easier iteration
for i, col in enumerate(cat_cols_for_plot):
    ax = axes[i]
    ct = pd.crosstab(df_work[col], df_work['Churn'], normalize='index') * 100
    ct.plot(kind='bar', stacked=False, ax=ax, color=['teal', 'orange'], legend=False)
    ax.set_title(f"Churn by {col}")
    ax.set_ylabel("Percentage (%)")
    ax.set_xticklabels(ax.get_xticklabels(), rotation=30, ha='right')
    # annotate percentages
    for p in ax.patches:
        h = p.get_height()
        if h > 0:
            ax.text(p.get_x() + p.get_width() / 2, h + 0.8, f"{h:.1f}%", ha='center', fontsize=9)
# Hide any unused subplots
for i in range(n, len(axes)):
    fig.delaxes(axes[i])
plt.tight_layout()
plt.show()
```

02:Data Cleaning and Preprocessing

2.1: Import Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="whitegrid")
print("✅ Libraries imported successfully!")
```

2.2: Load Dataset

```
url="https://raw.githubusercontent.com/Yenuli0808/CM2604_Telco_Customer_Churn-CW/main/data/Telco-Customer-Churn.csv"
df = pd.read_csv(url)
print("✅ Dataset loaded successfully!")
print("\n=== Data set overview ===")
print(f"Dataset shape: {df.shape}")  # (rows, columns)
```

2.3: Initial Inspection

```
# First look at the data
print("==== FIRST 5 ROWS ====\n")
display(df.head())
```

```
print("\n==== Dataset Info ====\n")
df.info()
print("\n===== SUMMARY STATS =====")
df.describe(include="all")
```

2.4: Basic Column Fixes

```
# Convert Total Charges to numeric values
#Turning Invalid Values into NaN
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
#count invalid rows
invalid_rows = df['TotalCharges'].isnull().sum()
print(f"Number of invalid rows in 'TotalCharges': {invalid_rows}")
#Removing Invalid Rows
df = df.dropna(subset=['TotalCharges']).reset_index(drop=True)
print("\n----- Shape after dropping invalid rows -----")
print("Dataset New Shape:",df.shape)
# Remove Duplicate Values
before = df.shape[0]
df = df.drop_duplicates()
after = df.shape[0]
# Dropping CustomerID as it's not a predective feature
def exclude_useless(df):
    if 'customerID' in df.columns:
        df = df.drop('customerID', axis=1)
        print("Dropping customerID (as it's not useful for modeling)")
    else:
        print("customerID column not found, likely already dropped.")
    return df
df = exclude_useless(df)
print("\n----- Shape after dropping customerID -----")
print("Dataset New Shape:",df.shape,"\n")
# Quick look over dataset
df.head()
# Convert Binary Yes/No columns to 0/1
# 1) Finding yes/no columns
binary_cols = [col for col in df.columns
               if df[col].nunique() == 2 and
               sorted(df[col].unique().tolist()) == ['No','Yes']]
#2) converting them to 0/1
for col in binary_cols:
    df[col] = df[col].map({'Yes': 1,'No': 0})
print("\n✅ All binary columns encoded to 0/1!")
print("\nConverted binary columns: ", binary_cols)
print("\nQuick look on Data set after conversion:\n")
df.head(10)
# Fix Internet-Dependant Columns
```

```
# As many columns contains "No Internet Service" -> will be treated as 0
# 1) Finding columns with this - "No internet service" as a unique value
cols_with_no_internet_service = []
for col in df.columns:
    if 'No internet service' in df[col].unique():
        cols_with_no_internet_service.append(col)
print(f"Columns containing 'No internet service': {cols_with_no_internet_service}")
# 2) Converting "No internet service" to "No" in these columns
for col in cols_with_no_internet_service:
    df[col] = df[col].replace({'No internet service':'No'})
    df[col] = df[col].map({'Yes': 1,'No': 0})
print(f"\n✅ Converted 'No internet service' to 'No' in columns: {cols_with_no_internet_service}")
# Now, let's verify a few columns if they are updated
print("\nQuick look on Data set after conversion (first 5 rows):\n")
display(df[cols_with_no_internet_service].head())
# Fix MultipleLines
# "No phone service" -> same as "No" coverting them to 0
df["MultipleLines"] = df["MultipleLines"].replace({"No phone service": "No"})
df["MultipleLines"] = df["MultipleLines"].map({"Yes": 1, "No": 0})
```

2.5: Outlier Analysis

```
# Boxplots for Outlier Inspection
plt.figure(figsize=(14,6))
plt.subplot(1,3,1)
sns.boxplot(y=df["tenure"])
plt.title("Tenure Outliers")
plt.subplot(1,3,2)
sns.boxplot(y=df["MonthlyCharges"])
plt.title("Monthly Charges Outliers")
plt.subplot(1,3,3)
sns.boxplot(y=df["TotalCharges"])
plt.title("Total Charges Outliers")
plt.tight_layout()
plt.show()
print("\nOutliers were inspected but not removed, as they represent real customer behaviour.")
```

2.5: Feature Engineering

```
# Tenure Groups
bins = [0,12,24,48,60,72]
labels = ["0-12","13-24","25-48","49-60","61-72"]
df["TenureGroup"] = pd.cut(df["tenure"], bins=bins, labels=labels, include_lowest=True)
print("\n✅Tenure groups created!")
df["TenureGroup"].value_counts()
# 2. Monthly Charge Groups
df["MonthlyChargeGroup"] = pd.cut(df["MonthlyCharges"],
                                  bins=[0,35,70,100,200],
                                  labels=["Low","Medium","High","Very High"],
```

```
                          include_lowest=True)
print("\n✅ Monthly charge groups created!")
df["MonthlyChargeGroup"].value_counts()
# Derived Features
# Charges Ratio (This will avoid division by zero)
df["ChargesRatio"] = df["TotalCharges"] / (df["MonthlyCharges"] + 1)
print("\n✅Charges ratio created!")
# Service Count
service_cols = ['PhoneService', 'MultipleLines', 'OnlineSecurity', 'OnlineBackup',
          'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies']
df['ServiceCount'] = df[service_cols].sum(axis=1)
print("\n✅ Service Count created!")
print("\nQuick look at ServiceCount:\n")
display(df[['ServiceCount']].head())
# TenureMonths Category: Seniority Flag
df["IsLongTermCustomer"] = (df["tenure"] >= 24).astype(int)
print("\n✅Seniority flag created!")
print("\nQuick look at IsLongTermCustomer:\n")
display(df[['IsLongTermCustomer']].head())
#AutoPay Feature (PaymentMethod grouping)
df["IsAutoPay"] = df["PaymentMethod"].isin([
   "Bank transfer (automatic)",
   "Credit card (automatic)"
]).astype(int)
print("\n✅AutoPay flag created!")
print("\nQuick look at IsAutoPay:\n")
display(df[['IsAutoPay']].head())
# HasInternet Flag
df["HasInternet"] = (df["InternetService"] != "No").astype(int)
print("\n✅HasInternet flag created!")
print("\nQuick look at HasInternet:\n")
display(df[['HasInternet']].head())
print("\n✅Feature engineering completed Successfully!")
print(f"\nDataset new shape: {df.shape}\n")
df.head()
```

2.6: Final Cleaning Summary

```
# check weather if there any missing values
missing_values = df.isnull().sum()
print("Missing Values:\n", missing_values)
print("\n===== FINAL CLEANING SUMMARY =====")
print("Final dataset shape:", df.shape)
# Define categorical and numerical columns
categorical_cols = df.select_dtypes(include='object').columns.tolist()
numerical_cols = df.select_dtypes(exclude='object').columns.tolist()
print("\nCategorical features:", categorical_cols)
```

```
print("\nNumerical features:", numerical_cols)
print("\nClass distribution:\n")
class_dist = pd.DataFrame({
   "Count": df["Churn"].value_counts(),
   "Percentage (%)": (df["Churn"].value_counts(normalize=True) * 100).round(2)
})
class_dist
```

2.7: Saving Cleaned Dataset

```
# Saving Clean DataSet
cleaned_file = 'Cleaned_Telco_Customer_Churn.csv'
df.to_csv(cleaned_file, index=False)
print("✅ Cleaned dataset saved successfully as", cleaned_file,"!")
```

Step 03: Modeling Decision Tress and Neural Network Models

3.a: Import Libraries

```
import sys, os, joblib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (accuracy_score, classification_report, confusion_matrix,
                             roc_auc_score, roc_curve, precision_recall_curve, auc, brier_score_loss,precision_score, recall_score,
f1_score)
from sklearn.inspection import permutation_importance
from imblearn.over_sampling import SMOTE
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
RANDOM_STATE = 42
print("Versions Being Used :\n")
print("Python:", sys.version.split()[0])
print("pandas:", pd.__version__, "numpy:", np.__version__)
import sklearn
print("sklearn:", sklearn.__version__)
import imblearn
print("imblearn:", imblearn.__version__)
import tensorflow as tf
print("tensorflow:", tf.__version__)
```

```
print("\n✅ Libraries imported successfully!")
```

3.b: Import Cleaned Data Set

```
# Load the cleaned data set from the git
url_clean =
"https://raw.githubusercontent.com/Yenuli0808/CM2604_Telco_Customer_Churn-CW/main/data/Cleaned_Telco_Customer_Churn.csv"
df = pd.read_csv(url_clean)
print("✅ Cleaned Dataset loaded successfully!")
print("\n=== Cleaned Data set overview ===")
print(f"Dataset shape: {df.shape}")
```

3.c: Quick Look-up on Dataset

```
# First look at the cleaned dataset
print("==== FIRST 10 ROWS ====\n")
df.head(10)
print("\nClass distribution:\n")
class_dist = pd.DataFrame({
   "Count": df["Churn"].value_counts(),
   "Percentage (%)": (df["Churn"].value_counts(normalize=True) * 100).round(2)
})
class_dist
```

3.d: Train-Test Split (stratified) and Variable Lists

```
# Defining features and target
X = df.drop("Churn", axis=1)
y = df["Churn"]
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, stratify=y, random_state=RANDOM_STATE
)
# Class distribution in training set
churn_table_train = pd.DataFrame({
   'Count': y_train.value_counts(),
   'Percentage (%)': (y_train.value_counts(normalize=True) * 100).round(2)
})
# Class distribution in testing set
churn_table_test = pd.DataFrame({
   'Count': y_test.value_counts(),
   'Percentage (%)': (y_test.value_counts(normalize=True) * 100).round(2)
})
print("Training set: ", X_train.shape)
print("\nClass distribution in training set: ")
print(churn_table_train)
print("\nTesting set: ", X_test.shape)
print("\nClass distribution in testing set: ")
print(churn_table_test)
# Combine the tables for a comprehensive view
combined_churn_distribution = pd.concat({
```

```
    'Train': churn_table_train,
    'Test': churn_table_test
}, axis=1)
print("\nCombined Class Distribution (Train vs. Test):\n")
print(combined_churn_distribution)
# Identifying categorical and numerical columns
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()
print("Categorical columns:", categorical_cols,"\n")
print("Numerical columns:", numerical_cols)
```

3.e: Preprocessing Pipeline

```
# Preprocessing pipeline
preprocessor = ColumnTransformer([
    ('num', StandardScaler(),numerical_cols),
    ('cat', OneHotEncoder(drop='first', handle_unknown='ignore'),categorical_cols)
])
print("✅ Preprocessing pipeline created successfully!")
print("\n=== Preprocessing pipeline overview ===")
preprocessor
# Fit transform train, transform test
X_train_preprocessed = preprocessor.fit_transform(X_train)
X_test_preprocessed = preprocessor.transform(X_test)
print("✅ Preprocessing completed successfully!")
print("\n=== Preprocessed Data set overview ===")
print(f"\nTraining set shape: {X_train_preprocessed.shape}")
print(f"Testing set shape: {X_test_preprocessed.shape}")
```

3.f: Helper Evaluate Functions

```
def print_eval_full(model_name,
                    y_train, y_train_pred, y_train_proba,
                    y_test,  y_test_pred,  y_test_proba):
    print("="*60)
    print(f"MODEL EVALUATION: {model_name}")
    print("="*60)
    # -------------------------
    # TRAINING METRICS
    # -------------------------
    print("\n TRAINING PERFORMANCE")
    print("Accuracy:", round(accuracy_score(y_train, y_train_pred)*100, 4))
    print("ROC AUC:", round(roc_auc_score(y_train, y_train_proba), 4))
    print("\nClassification Report (Train):")
    print(classification_report(y_train, y_train_pred))
    cm_train = confusion_matrix(y_train, y_train_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm_train, annot=True, fmt='d', cmap='Reds')
    plt.title(f"Confusion Matrix - TRAIN ({model_name})")
```

```
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()
# -------------------------
    # TESTING METRICS
    # -------------------------
    print("\n TESTING PERFORMANCE")
    print("Accuracy:", round(accuracy_score(y_test, y_test_pred)*100, 4))
    print("ROC AUC:", round(roc_auc_score(y_test, y_test_proba), 4))
    print("\nClassification Report (Test):")
    print(classification_report(y_test, y_test_pred))
    cm_test = confusion_matrix(y_test, y_test_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm_test, annot=True, fmt='d', cmap='Blues')
    plt.title(f"Confusion Matrix - TEST ({model_name})")
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()
    print("="*60)
```

3.1: Decision Tree Model

3.1.1: Hyperparameter tuning for Decision Tress

```
depths = [3,5,7,10,None]
min_sample_split = [2,5,10]
min_sample_leaf = [1,2,4]
criterion = ['gini', 'entropy']
best_dt_auc = -1
best_dt_params = {}
best_model_dt = None
results = []
for d in depths:
  for ms in min_sample_split:
    for ml in min_sample_leaf:
      for crit in criterion:
        dt = DecisionTreeClassifier(
          max_depth=d,
          min_samples_split=ms,
          min_samples_leaf=ml,
          criterion=crit,
          random_state=RANDOM_STATE
      )
      dt.fit(X_train_preprocessed, y_train)
      proba = dt.predict_proba(X_test_preprocessed)[:, 1]
      auc_score = roc_auc_score(y_test, proba)
      results.append((auc_score, d, ms, ml))
      if auc_score > best_dt_auc:
        best_dt_auc = auc_score
```

```
        best_model_dt = dt
        best_dt_params = {'depths':d, 'min_sample_split':ms, 'min_sample_leaf':ml, "criterion":crit }
print("Best DT Parameters:", best_dt_params)
print("\nBest DT AUC Score:", best_dt_auc)
#Showing top 5 configs by AUC
results_sorted = sorted(results, reverse=True, key=lambda x: x[0])[:5]
print("\nTop 5 DT Configurations:")
for r in results_sorted:
  print(f"AUC: {r[0]}, Depth: {r[1]}, Min Sample Split: {r[2]}, Min Sample Leaf: {r[3]}")
# Croo validation on train for best config (stability)
cv_scores = cross_val_score(best_model_dt, X_train_preprocessed, y_train, cv=5, scoring='roc_auc', n_jobs=-1)
print("\nCV AUC (train) for best DT: %.4f ± %.4f" % (cv_scores.mean(), cv_scores.std()))
```

3.1.2: Decision Tree Evaluation

```
y_proba_dt = best_model_dt.predict_proba(X_test_preprocessed)[:,1]
y_pred_dt = (y_proba_dt > 0.5).astype(int)
y_pred_proba_train_dt = best_model_dt.predict_proba(X_train_preprocessed)[:,1]
y_pred_train_dt = (y_pred_proba_train_dt > 0.5).astype(int)
print_eval_full("Decision Tree  - Test",
            y_train, y_pred_train_dt, y_pred_proba_train_dt,
            y_test, y_pred_dt, y_proba_dt)
```

3.1.2.1: ROC Curve

```
fpr_dt, tpr_dt, _ =roc_curve(y_test, y_proba_dt)
plt.figure(figsize=(7,6))
plt.plot(fpr_dt, tpr_dt, label=f'ROC Curve (AUC = {roc_auc_score(y_test, y_proba_dt):.3f})')
plt.plot([0,1], [0,1], 'k--')
plt.title('Decision Tree - ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.savefig('roc_curve_dt.png')
plt.show()
```

3.1.2.2: DT-Precision Recall Curve

```
y_pred_proba_dt = best_model_dt.predict_proba(X_test_preprocessed)[:, 1]
precision_dt, recall_dt, thresholds_dt = precision_recall_curve(y_test, y_pred_proba_dt)
pr_auc_dt = auc(recall_dt, precision_dt)
plt.figure(figsize=(8,6))
plt.plot(recall_dt, precision_dt, label=f'Decision Tree (PR-AUC = {pr_auc_dt:.3f})', color='teal')
plt.xlabel('Recall (Sensitivity) - How many actual churners did we catch?')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve - Decision Tree')
plt.legend()
plt.grid(True)
plt.savefig('pr_curve_dt.png')
plt.show()
```

3.1.3: DT- Feature Imporatance

```
encoded_feature_names = preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_cols).tolist()
feature_names = numerical_cols + encoded_feature_names
importance = best_model_dt.feature_importances_
feat_imp = pd.Series(importance, index=feature_names).sort_values(ascending=False)[:15]
plt.figure(figsize=(15,11))
feat_imp.plot(kind='barh')
plt.title('Top 15 Feature Importance - Decision Tree')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.savefig('feature_importance_dt.png')
plt.show()
```

3.1.4: BaseLine Decision Tree (No-Tuning)

```
baseline_dt = DecisionTreeClassifier(random_state=RANDOM_STATE)
baseline_dt.fit(X_train_preprocessed, y_train)
y_proba_base_dt = baseline_dt.predict_proba(X_test_preprocessed)[:,1]
y_pred_base_dt = (y_proba_base_dt > 0.5).astype(int)
print_eval_full(
   "Baseline Decision Tree (No Tuning)",
   y_train,
   baseline_dt.predict(X_train_preprocessed),
   baseline_dt.predict_proba(X_train_preprocessed)[:, 1],
   y_test,
   y_pred_base_dt,
   y_proba_base_dt
)
```

3.2: Neural Network Model (Keras)

3.2.1: Hyperparameter tuning for NN

```
import time
from tensorflow.keras.callbacks import EarlyStopping
layer_configs = [[64, 32], [128, 64], [128, 64, 32]]
learning_rates = [1e-3, 1e-4]
dropouts = [0.0,0.2]
epochs = 20
batch_size = 32
best_nn_acc = -1
best_nn = None
best_nn_params = {}
best_history = None
es = EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)
t0 = time.time()
for layers in layer_configs:
   for lr in learning_rates:
```

```
    for drop in dropouts:
        tf.keras.backend.clear_session()
        model = Sequential()
        model.add(Dense(layers[0], activation='relu', input_shape=(X_train_preprocessed.shape[1],)))
        if drop > 0:
            model.add(Dropout(drop))
        for units in layers[1:]:
            model.add(Dense(units, activation='relu'))
            if drop > 0:
                model.add(Dropout(drop))
        model.add(Dense(1, activation='sigmoid'))
        model.compile(optimizer=Adam(learning_rate=lr), loss='binary_crossentropy', metrics=['accuracy'])
        history = model.fit(X_train_preprocessed, y_train, validation_split=0.2,
                            epochs=epochs, batch_size=batch_size,callbacks=[es], verbose=0)
        test_loss, test_acc = model.evaluate(X_test_preprocessed, y_test, verbose=0) # Added verbose=0 to suppress
output during tuning
        if test_acc > best_nn_acc:
            best_nn_acc = test_acc
            best_nn = model
            best_nn_params = {'layers': layers, 'lr': lr, 'dropout': drop}
            best_history = history
print("Best NN Parameters (manual tuning):", best_nn_params)
print("Best NN Testing Accuracy:", best_nn_acc)
print("Time elapsed (s):", round(time.time()-t0,2),"seconds")
```

3.2.2: NN Evaluation

```
y_proba_nn = best_nn.predict(X_test_preprocessed).ravel()
y_pred_nn = (y_proba_nn > 0.5).astype(int)
y_proba_train_nn = best_nn.predict(X_train_preprocessed).ravel()
y_pred_train_nn = (y_proba_train_nn > 0.5).astype(int)
print_eval_full("Neural Network(Tuning)",
            y_train, y_pred_train_nn, y_proba_train_nn,
            y_test, y_pred_nn, y_proba_nn)
```

3.2.2.1: ROC Curve

```
fpr_nn, tpr_nn, _ = roc_curve(y_test, y_proba_nn.flatten())
plt.figure(figsize=(7,6))
plt.plot(fpr_nn, tpr_nn, label=f'ROC (AUC = {roc_auc_score(y_test, y_proba_nn.flatten()):.3f})')
plt.plot([0,1], [0,1], 'k--')
plt.title('ROC Curve - Neural Network')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.show()
```

3.2.2.2: NN-Precision Recall Curve

```
precision_nn, recall_nn, _ = precision_recall_curve(y_test, y_proba_nn.flatten())
```

```
plt.figure(figsize=(8,6))
plt.plot(recall_nn, precision_nn, label=f'PR (AUC = {auc(recall_nn, precision_nn):.3f})')
plt.title('Precision-Recall Curve - Neural Network')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
plt.savefig('pr_curve_nn.png')
plt.show()
```

3.2.2.3: Training Curves for NN

```
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
plt.plot(best_history.history['accuracy'], label='Training Accuracy')
plt.plot(best_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('NN Training Accuracy Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Acuracy')
plt.legend()
plt.subplot(2, 2, 2)
plt.plot(best_history.history['loss'], label='Training Loss')
plt.plot(best_history.history['val_loss'], label='Validation Loss')
plt.title('NN Training Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
```

3.2.3: Baseline NN (without tuning)

```
tf.keras.backend.clear_session()
baseline_nn = Sequential([
   Dense(32, activation='relu', input_shape=(X_train_preprocessed.shape[1],)),
   Dense(1, activation='sigmoid')
])
baseline_nn.compile(optimizer=Adam(0.001), loss='binary_crossentropy', metrics=['accuracy'])
history_base_nn = baseline_nn.fit(
   X_train_preprocessed, y_train,
   validation_split=0.2,
   epochs=20, batch_size=32, verbose=0
)
y_proba_base_nn = baseline_nn.predict(X_test_preprocessed).ravel()
y_pred_base_nn = (y_proba_base_nn > 0.5).astype(int)
# Calculate predictions and probabilities for the training set
y_proba_train_base_nn = baseline_nn.predict(X_train_preprocessed).ravel()
y_pred_train_base_nn = (y_proba_train_base_nn > 0.5).astype(int)
print_eval_full("Baseline Neural Network (No Tuning, No SMOTE)",
        y_train, y_pred_train_base_nn, y_proba_train_base_nn,
```

y_test, y_pred_base_nn, y_proba_base_nn)

Step 03: Model Comparison (Decision Tress vs Neural Networks)

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
def get_metrics(model_name, y_train, y_pred_train, y_proba_train, y_test, y_pred, y_proba):
 return{
    "Model": model_name,
    # Train Metrics
    "Train Accuracy": accuracy_score(y_train, y_pred_train),
    "Train Precision": precision_score(y_train, y_pred_train),
    "Train Recall": recall_score(y_train, y_pred_train),
    "Train F1 Score": f1_score(y_train, y_pred_train),
    "Train ROC AUC": roc_auc_score(y_train, y_proba_train),
    # Test Metrics
    "Test Accuracy": accuracy_score(y_test, y_pred),
    "Test Precision": precision_score(y_test, y_pred),
    "Test Recall": recall_score(y_test, y_pred),
    "Test F1 Score": f1_score(y_test, y_pred),
    "Test ROC AUC": roc_auc_score(y_test, y_proba)
 }
results = []
# --- Fix Start ---
# Calculate missing training predictions and probabilities for Baseline Decision Tree
# Assuming 'baseline_dt' and 'X_train_preprocessed' are available from previous cells.
#y_proba_train_dt_base = baseline_dt.predict_proba(X_train_preprocessed)[:, 1]
#y_train_pred_dt_base = (y_proba_train_dt_base > 0.5).astype(int)
y_train_pred_dt_base = baseline_dt.predict(X_train_preprocessed)
y_proba_train_dt_base = baseline_dt.predict_proba(X_train_preprocessed)[:, 1]
# Use existing variable names for Tuned DT training metrics
y_train_pred_dt_tuned = y_pred_train_dt
y_train_proba_dt_tuned = y_pred_proba_train_dt
# Use existing variable names for Tuned NN training metrics
y_train_pred_nn_tuned = y_pred_train_nn
y_train_proba_nn_tuned = y_proba_train_nn
# --- Fix End ---
# === DECISION TREE MODELS ===
results.append(
   get_metrics(
      "DT Baseline",
      y_train, y_train_pred_dt_base, y_proba_train_dt_base,
      y_test, y_pred_base_dt, y_proba_base_dt
   )
)
```

```
results.append(
    get_metrics(
        "DT Tuned",
        y_train, y_train_pred_dt_tuned, y_train_proba_dt_tuned,
        y_test, y_pred_dt, y_proba_dt
    )
)
# === NEURAL NETWORK MODELS ===
results.append(
    get_metrics(
        "NN Baseline",
        y_train, y_pred_train_base_nn, y_proba_train_base_nn,
        y_test, y_pred_base_nn, y_proba_base_nn
    )
)
results.append(
    get_metrics(
        "NN Tuned",
        y_train, y_train_pred_nn_tuned, y_train_proba_nn_tuned,
        y_test, y_pred_nn, y_proba_nn
    )
)
comparison_df = pd.DataFrame(results)
comparison_df = comparison_df.round(4)
comparison_df
plt.figure(figsize=(14,6))
sns.heatmap(comparison_df.set_index("Model"), annot=True, cmap="YlGnBu", fmt=".3f")
plt.title("Model Comparison: Decision Tree vs Neural Network (Baseline, Tuned)")
plt.savefig("model_comparison.png")
plt.show()
plt.figure(figsize=(10,8))
def plot_model_roc(model, X_test, y_test, label):
    proba = model.predict_proba(X_test)[:,1] if hasattr(model, "predict_proba") \
            else model.predict(X_test).flatten()
    fpr, tpr, _ = roc_curve(y_test, proba)
    auc_score = roc_auc_score(y_test, proba)
    plt.plot(fpr, tpr, label=f"{label} (AUC={auc_score:.3f})")
# Decision Tree
plot_model_roc(baseline_dt, X_test_preprocessed, y_test, "DT Baseline")
plot_model_roc(best_model_dt, X_test_preprocessed, y_test, "DT Tuned")
# Neural Network
plot_model_roc(baseline_nn, X_test_preprocessed, y_test, "NN Baseline")
plot_model_roc(best_nn, X_test_preprocessed, y_test, "NN Tuned")
# Random guess line
plt.plot([0,1], [0,1], 'k--', label="Random Guess")
```

```
plt.title("Combined ROC Curves (DT vs NN)")
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.legend()
plt.savefig("combined_roc.png")
plt.show()
metrics_to_plot = ["Test Accuracy", "Test Precision", "Test Recall", "Test F1 Score", "Test ROC AUC"]
plt.figure(figsize=(14,6))
comparison_df.set_index("Model")[metrics_to_plot].plot(kind="bar", figsize=(14,6))
plt.title("Model Comparison (Test Metrics Only)")
plt.ylabel("Score")
plt.xticks(rotation=45)
plt.legend(loc="lower right")
plt.savefig("model_comparison_test_metrics.png")
plt.show()
best_model_row = comparison_df.sort_values("Test ROC AUC", ascending=False).iloc[0]
print("==================================================")
print("📌 FINAL RECOMMENDATION")
print("==================================================")
print("Best Performing Model:", best_model_row["Model"])
print("ROC AUC:", best_model_row["Test ROC AUC"])
print("Testing Accuracy:", best_model_row["Test Accuracy"])
print("\nReason:")
if "NN" in best_model_row["Model"]:
    print("→ Neural Network generalises better and captures non-linear relationships.")
else:
    print("→ Decision Tree performed best and is more interpretable for business use.")
```

3.4:Synthetic Minority Oversampling Technique (SMOTE) - Exceptional checkup

#Only apply for the training dataset.

```
import pandas as pd
import numpy as np
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=RANDOM_STATE)
X_train_sm, y_train_sm = sm.fit_resample(X_train_preprocessed, y_train)
before_smote = np.bincount(y_train)
after_smote = np.bincount(y_train_sm)
data = {
    'data set': ['Before SMOTE', 'After SMOTE'],
    'Class 0': [before_smote[0], after_smote[0]],
    'Class 1': [before_smote[1], after_smote[1]]
}
df_smote_comparison = pd.DataFrame(data)
print("\n=== SMOTE Class Distribution Comparison ===\n")
```

```
print(df_smote_comparison)
# Get counts before and after SMOTE
before_smote_counts = np.bincount(y_train)
after_smote_counts = np.bincount(y_train_sm)
labels = ['Class 0', 'Class 1']
# Create a figure with two subplots side-by-side
fig, axes = plt.subplots(1, 2, figsize=(14, 5))
# Histogram for Class Distribution Before SMOTE
axes[0].bar(labels, before_smote_counts, color=['teal', 'orange'])
axes[0].set_title('Class Distribution Before SMOTE')
axes[0].set_xlabel('Class')
axes[0].set_ylabel('Count')
axes[0].tick_params(axis='x', rotation=0)
# Histogram for Class Distribution After SMOTE
axes[1].bar(labels, after_smote_counts, color=['teal', 'orange'])
axes[1].set_title('Class Distribution After SMOTE')
axes[1].set_xlabel('Class')
axes[1].set_ylabel('Count')
axes[1].tick_params(axis='x', rotation=0)
plt.tight_layout()
plt.show()
```

3.4.1: Decision Tree tuning (with SMOTE)

```
best_auc_sm = -1
best_dt_sm = None
best_params_sm = None
results_sm = []
for d in depths:
   for ms in min_sample_split:
      for ml in min_sample_leaf:
         dt = DecisionTreeClassifier(max_depth=d, min_samples_split=ms,
                                     min_samples_leaf=ml, random_state=RANDOM_STATE)
         dt.fit(X_train_sm, y_train_sm)
         proba = dt.predict_proba(X_test_preprocessed)[:,1]
         auc_score = roc_auc_score(y_test, proba)
         results_sm.append((auc_score, d, ms, ml))
         if auc_score > best_auc_sm:
            best_auc_sm = auc_score
            best_dt_sm = dt
            best_params_sm = (d, ms, ml)
print("Best DT (SMOTE) params:", best_params_sm, "AUC:", best_auc_sm)
cv_scores_sm = cross_val_score(best_dt_sm, X_train_sm, y_train_sm, cv=5, scoring='roc_auc', n_jobs=-1)
print("CV AUC (SMOTE train) for best DT:", cv_scores_sm.mean(), cv_scores_sm.std())
```

3.4.1.1: Evaluate DT (With SMOTE)

```
y_proba_dt_sm = best_dt_sm.predict_proba(X_test_preprocessed)[:,1]
y_pred_dt_sm = (y_proba_dt_sm > 0.5).astype(int)
```

```
y_pred_proba_train_dt_sm = best_dt_sm.predict_proba(X_train_sm)[:,1]
y_pred_train_dt_sm = (y_pred_proba_train_dt_sm > 0.5).astype(int)
print_eval_full("Decision Tree (with SMOTE)",
            y_train_sm, y_pred_train_dt_sm, y_pred_proba_train_dt_sm,
            y_test, y_pred_dt_sm, y_proba_dt_sm)
```

3.4.2: Neural Network Training (With SMOTE)

```
# tune (small grid) on SMOTE data
best_nn_acc_sm = -1
best_nn_sm = None
best_hist_sm = None
best_params_nn_sm = None
es = EarlyStopping(
   monitor='val_loss',
   patience=3,
   restore_best_weights=True
)
t0 = time.time()
for layers in layer_configs:
   for lr in learning_rates:
      for dr in dropouts:
         tf.keras.backend.clear_session()
         model = Sequential()
         model.add(Dense(layers[0], activation='relu', input_shape=(X_train_sm.shape[1],)))
         if dr > 0:
            model.add(Dropout(dr))
         for units in layers[1:]:
            model.add(Dense(units, activation='relu'))
            if dr > 0:
               model.add(Dropout(dr))
         model.add(Dense(1, activation='sigmoid'))
         model.compile(optimizer=Adam(learning_rate=lr), loss='binary_crossentropy', metrics=['accuracy'])
         hist = model.fit(X_train_sm, y_train_sm, validation_split=0.2,
                     epochs=epochs, batch_size=batch_size,callbacks=[es], verbose=0)
         test_loss, test_acc = model.evaluate(X_test_preprocessed, y_test, verbose=0)
         if test_acc > best_nn_acc_sm:
            best_nn_acc_sm = test_acc
            best_nn_sm = model
            best_hist_sm = hist
            best_params_nn_sm = {"layers": layers, "lr": lr, "dropout": dr}
print("Best NN (SMOTE) params:", best_params_nn_sm)
print("Best NN (SMOTE) test accuracy:", best_nn_acc_sm)
print("Time elapsed (s):", round(time.time()-t0,1))
```

3.4.2.1: Evaluate NN (With SMOTE)

```
y_proba_nn_sm = best_nn_sm.predict(X_test_preprocessed).ravel()
y_pred_nn_sm = (y_proba_nn_sm > 0.5).astype(int)
```

```
y_pred_proba_train_nn_sm = best_nn_sm.predict(X_train_sm).ravel()
y_pred_train_nn_sm = (y_pred_proba_train_nn_sm > 0.5).astype(int)
print_eval_full("Decision Tree (with SMOTE)",
        y_train_sm, y_pred_train_nn_sm, y_pred_proba_train_nn_sm,
        y_test, y_pred_nn_sm, y_proba_nn_sm)
```

3.4.2.2: Training Curves for NN

```
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
plt.plot(best_history.history['accuracy'], label='Training Accuracy')
plt.plot(best_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('NN Training Accuracy (SMOTE)')
plt.xlabel('Epoch')
plt.ylabel('Acuracy')
plt.legend()
plt.subplot(2, 2, 2)
plt.plot(best_history.history['loss'], label='Training Loss')
plt.plot(best_history.history['val_loss'], label='Validation Loss')
plt.title('NN Training Loss (SMOTE)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
```

3.4.2.3: ROC Comparison Plots (SMOTE)

```
# Define the helper function plot_roc
def plot_roc(y_true, y_score, label):
   fpr, tpr, _ = roc_curve(y_true, y_score)
   auc_score = roc_auc_score(y_true, y_score)
   plt.plot(fpr, tpr, label=f'{label} (AUC = {auc_score:.3f})')
# Plot for Decision Tree Models
plt.figure(figsize=(7,6))
plot_roc(y_test, y_proba_dt, label="DT (No SMOTE)")
plot_roc(y_test, y_proba_dt_sm, label="DT (SMOTE)")
plt.plot([0,1],[0,1],'k--')
plt.title("ROC Curves - Decision Tree Models")
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)
plt.show()
# Plot for Neural Network Models
plt.figure(figsize=(7,6))
plot_roc(y_test, y_proba_nn, label="NN (No SMOTE)")
plot_roc(y_test, y_proba_nn_sm, label="NN (SMOTE)")
plt.plot([0,1],[0,1],'k--')
plt.title("ROC Curves - Neural Network Models")
```

```
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)
plt.show()
comparison = pd.DataFrame({
   "Model": ["DT_no_smote","NN_no_smote","DT_smote","NN_smote"],
   "Test Accuracy": [
      accuracy_score(y_test, y_pred_dt),
      accuracy_score(y_test, y_pred_nn),
      accuracy_score(y_test, y_pred_dt_sm),
      accuracy_score(y_test, y_pred_nn_sm)
   ],
   "Test ROC AUC": [
      roc_auc_score(y_test, y_proba_dt),
      roc_auc_score(y_test, y_proba_nn),
      roc_auc_score(y_test, y_proba_dt_sm),
      roc_auc_score(y_test, y_proba_nn_sm)
   ]
})
comparison.round(4)
```

12. Reference

- Geek for Geek [2025] What is Exploratory Data Analysis? Available at: https://www.geeksforgeeks.org/data-analysis/what-is-exploratory-data-analysis/ (Accessed: 3rd December 2025)

- Geek for Geek [2025] Understanding the Confusion Matrix in Machine Learning. Available at: https://www.geeksforgeeks.org/machine-learning/confusion-matrix-machine-learning/ (Accessed: 7th December 2025)