# Table of Contents

## Abstract

This project aims to predict which telecom customers are likely to churn. The dataset provided by the telecom company contained 7,043 entries and 21 features; these features included a variety of demographic, account, and service-related factors. We used different machine learning models, including Random Forest, SVM, and Decision Tree, to make our predictions. We used EDA, feature scaling, and feature engineering to preprocess our data. After training and evaluating our various models, we used Grid Search CV for hyperparameter tuning. Our top-performing model (with a performance that was significantly better than the others) was the Random Forest.

Of all the models, the Random Forest model gave the best results, with an accuracy of 79.79% and an F1 score of 78.57% after tuning. The SVM model with a linear kernel achieved the best accuracy of 82%, but balanced the results with an F1 score of only 76%. Performance from Decision Tree models was reasonable, with the best unpruned tree achieving an accuracy of 80% and the best pruned tree achieving 81.12% accuracy. Overall, this project highlights the importance of both preprocessing and model tuning to improve prediction performance. The SVM model overall did the best for this telecom churn prediction task.

# 1. Introduction

The dataset appears to be a telecom churn dataset, which is typically used to predict customer churn based on various features.

a. Overview of the dataset:

**Total Entries**: 7043
**Total Columns**: 21

b. Columns:

**customerID:** Unique identifier for each customer.

**gender:** Customer's gender (Male/Female).

**SeniorCitizen:** Whether the customer is a senior citizen (0 = No, 1 = Yes).

**Partner:** Whether the customer has a partner (Yes/No).

**Dependents:** Whether the customer has dependents (Yes/No).

**tenure:** Number of months the customer has stayed with the company.

**PhoneService:** Whether the customer has phone service (Yes/No).

**MultipleLines:** Whether the customer has multiple lines (Yes/No/No phone service).

**InternetService:** Type of internet service (DSL, Fiber optic, No).

**OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport:** Whether the customer has opted for additional services.

**StreamingTV, StreamingMovies:** Whether the customer has streaming TV/movies services.

**Contract:** Type of contract (Month-to-month, One year, Two year).

**PaperlessBilling:** Whether the customer has opted for paperless billing (Yes/No).

**PaymentMethod:** Customer's payment method (Electronic check, Mailed check).

**MonthlyCharges:** Monthly charge for the customer.

**TotalCharges:** Total charge for the customer (some cleaning required as it's recorded as object type).

**Churn:** Target variable, indicating whether the customer has churned (Yes/No).

c. Dataset Characteristics:

It contains categorical variables (gender, partner, churn) and numerical variables (tenure, monthly charges).

The target variable to predict is likely Churn, indicating whether a customer has left the service.

## 2. Choice of Dependent and Independent Variables and Selection of Algorithms

### a. **Independent Variables:**

This is the variable that is manipulated or controlled by the researcher to observe its effect on the dependent variable. It's the "cause" in a cause-and-effect relationship.

**Demographic Information:**

- **gender: Gender of the customer.**
- **SeniorCitizen: Whether the customer is a senior citizen (1) or not (0).**
- **Partner: Whether the customer has a partner (Yes/No).**
- **Dependents: Whether the customer has dependents (Yes/No).**

**Account Information:**

- **tenure: Number of months the customer has stayed with the company.**
- **Contract: The type of contract (Month-to-month, One year, Two year).**
- **PaperlessBilling: Whether the customer uses paperless billing (Yes/No).**
- **PaymentMethod: The method of payment (e.g., Electronic check, Mailed check).**

**Services Information:**

- **PhoneService: Whether the customer has phone service (Yes/No).**
- **MultipleLines: Whether the customer has multiple phone lines.**
- **InternetService: Type of internet service (DSL, Fiber optic, No).**
- **OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies: Various add-on services (Yes/No).**

**Charges:**

- **MonthlyCharges: The amount charged to the customer monthly.**
- **TotalCharges: The total amount charged to the customer during their tenure.**

b. Dependent Variable:

This variable is what you measure in the experiment and what is affected during the experiment. It is the "effect" or the outcome that depends on the independent variable.

**Churn:**

- **This indicates whether the customer has churned (left the service), which is the outcome we are likely trying to predict.**

c. Selection of Algorithms

- **Support Vektor Machine**

  A support vector machine (SVM) is a supervised learning algorithm used for classification by identifying an optimal hyperplane that maximizes the separation between different classes in an N-dimensional space.

- **Decision Tree**

  A decision tree is a versatile, non-parametric supervised learning algorithm used for both classification and regression. It features a hierarchical structure with a root node, branches, internal nodes, and leaf nodes.

- **Random Forest**

  Random forest is a widely-used machine learning algorithm, credited to Leo Breiman and Adele Cutler. It merges the outputs of multiple decision trees to generate a single outcome. Its popularity stems from its ease of use and versatility, making it suitable for both classification and regression tasks.

# 3. Data Preparation

## a. Exploratory Data Analysis(EDA):

**Missing Values:**

The initial dataset had some missing values as space instead of “Null” firstly, we changed some kind of these data with “Null” values and after checking, we found that the proportion of missing data was less than 5% (%0.16). Since this percentage is minimal, we safely dropped the rows with missing values without significantly affecting the analysis.

**Code Block:**

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn import tree
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import svm
import os
import numpy as np
from sklearn import svm
from matplotlib.colors import ListedColormap
```

```
file_path = os.path.dirname(os.path.abspath(__file__))
file_path = file_path.replace("\\", "/")

# Load the dataset
data = pd.read_csv(file_path + '/Telecom_Churn_Data.csv')

#change space character with Null to count
columns=data.columns
for col in columns:
    data[col]=data[col].apply(lambda x: np.nan if x==" " else x)

data["Partner"]=data["Partner"].apply(lambda x: 1 if x=="Yes" else 0)
data["Dependents"]=data["Dependents"].apply(lambda x: 1 if x=="Yes" else 0)
data["PhoneService"]=data["PhoneService"].apply(lambda x: 1 if x=="Yes" else 0)
data["PaperlessBilling"]=data["PaperlessBilling"].apply(lambda x: 1 if x=="Yes"
else 0)
data["Churn"]=data["Churn"].apply(lambda x: 1 if x=="Yes" else 0)

print("Dataset Size:", data.shape)
print("Data Types:\n", data.dtypes)

#This is seperator for not to confuse code blocks
print("------------------------------------------------------------")

#Set columns Types
data["gender"]=data["gender"].astype("category")
data["SeniorCitizen"]=data["SeniorCitizen"].astype("bool")
data["Partner"]=data["Partner"].astype("bool")
data["Dependents"]=data["Dependents"].astype("bool")
data["PhoneService"]=data["PhoneService"].astype("bool")
data["MultipleLines"]=data["MultipleLines"].astype("category")
data["InternetService"]=data["InternetService"].astype("category")
data["OnlineSecurity"]=data["OnlineSecurity"].astype("category")
data["OnlineBackup"]=data["OnlineBackup"].astype("category")
data["DeviceProtection"]=data["DeviceProtection"].astype("category")
data["TechSupport"]=data["TechSupport"].astype("category")
data["StreamingTV"]=data["StreamingTV"].astype("category")
data["Contract"]=data["Contract"].astype("category")
data["PaperlessBilling"]=data["PaperlessBilling"].astype("bool")
data["PaymentMethod"]=data["PaymentMethod"].astype("category")
data["TotalCharges"]=data["TotalCharges"].astype("float")
data["Churn"]=data["Churn"].astype("bool")

print("Missing Values:")
print()
print(data.isnull().sum())
print("------------------------------------------------------------")
#get null counts
null_counts = data.isnull().sum()
#percentage of null data
null_percentage = (null_counts / len(data))*100
formatted_null_percentage = null_percentage.apply(lambda x: f"%{x:.2f}")
print(formatted_null_percentage)
print("------------------------------------------------------------")

#Null data proportion is less than %5 so we can remove them
data = data.dropna()
# See results
print("Null-Free Data:")
print(data)
print("------------------------------------------------------------")
```

**Output:**

| Column | Missing Values | Missing Percentages |
|---|---|---|
| customerID | 0 | 0.00% |
| gender | 0 | 0.00% |
| SeniorCitizen | 0 | 0.00% |
| Partner | 0 | 0.00% |
| Dependents | 0 | 0.00% |
| tenure | 0 | 0.00% |
| PhoneService | 0 | 0.00% |
| MultipleLines | 0 | 0.00% |
| InternetService | 0 | 0.00% |
| OnlineSecurity | 0 | 0.00% |
| OnlineBackup | 0 | 0.00% |
| DeviceProtection | 0 | 0.00% |
| TechSupport | 0 | 0.00% |
| StreamingTV | 0 | 0.00% |
| StreamingMovies | 0 | 0.00% |
| Contract | 0 | 0.00% |
| PaperlessBilling | 0 | 0.00% |
| PaymentMethod | 0 | 0.00% |
| MonthlyCharges | 0 | 0.00% |
| TotalCharges | 11 | 0.16% |
| Churn | 0 | 0.00% |

*Table 1*

**Categorical and Numerical Data:**

Several columns were categorical or boolean in nature. These were properly encoded or transformed.

| Column | Data Type (Before) | Data Type (After) |
|---|---|---|
| customerID | object | object |
| gender | object | category |
| SeniorCitizen | int64 | bool |
| Partner | int64 | bool |
| Dependents | int64 | bool |
| tenure | int64 | int64 |
| PhoneService | int64 | bool |
| MultipleLines | object | category |
| InternetService | object | category |
| OnlineSecurity | object | category |
| OnlineBackup | object | category |
| DeviceProtection | object | category |
| TechSupport | object | category |
| StreamingTV | object | category |
| StreamingMovies | object | object |

| | | |
|---|---|---|
| Contract | object | category |
| PaperlessBilling | int64 | bool |
| PaymentMethod | object | category |
| MonthlyCharges | float64 | float64 |
| TotalCharges | object | float64 |
| Churn | int64 | bool |

*Table 2*

Partner, Dependents, PhoneService, PaperlessBilling, and Churn were converted into boolean values (True/False).

Columns like InternetService, Contract, PaymentMethod were treated as categorical.

The TotalCharges column was converted to a float for numerical analysis.

Examples of categoric and numeric variables' distribution are demonstrated on Figure 1.

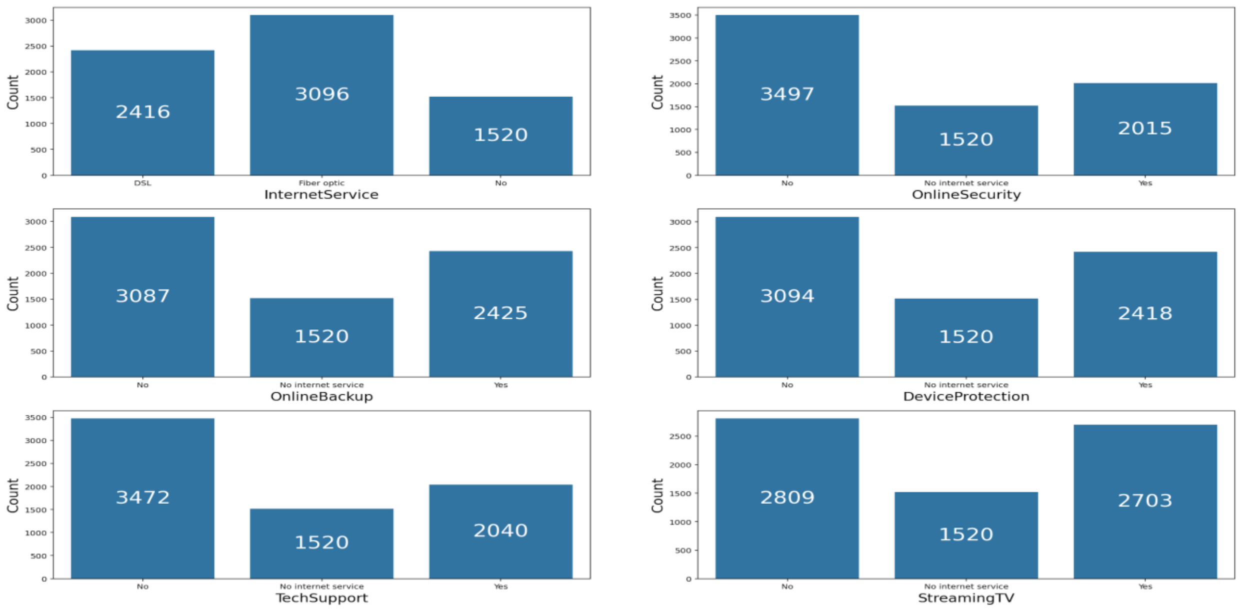


*Figure 1*

**Numeric Statistical Summary:**

The numerical values (like tenure, MonthlyCharges, and TotalCharges) gives insight into the distribution of numeric features.

**Code Block:**

```
# Numberic Statistical  Summary
numeric_summary = data.describe()
print("\nNumeric Statistical Summary:")
print(numeric_summary)
```

| | tenure | MonthlyCharges | TotalCharges |
|---|---|---|---|
| count | 7032 | 7032 | 7032 |
| mean | 32.421786 | 64.798208 | 2283.300441 |
| std | 24.54526 | 30.085974 | 2266.771362 |
| min | 1 | 18.25 | 18.8 |

|  |  |  |  |
|---|---|---|---|
| 25% | 9 | 35.5875 | 401.45 |
| 50% | 29 | 70.35 | 1397.475 |
| 75% | 55 | 89.8625 | 3794.7375 |
| max | 72 | 118.75 | 8684.8 |

Table 3

The tenure column (which represents how long a customer has been with the service) has a wide range, from new customers (0 months) to long-time customers (up to 72 months).

MonthlyCharges and TotalCharges provide a measure of customer spending.

**Categorical Statistical Summary:**

The categorical summary shows the distribution of various service options chosen by customers:

**Code Block:**

```
# Categorical Statistical Summary
categorical_summary = data.describe(include=['object', 'category','bool'])
print("\nCategorical Statistical Summary:")
print(categorical_summary)
```

|  | InternetService | Contract | ... | PaymentMethod | Churn |
|---|---|---|---|---|---|
| count | 7032 | 7032 | ... | 7032 | 7032 |
| unique | 3 | 3 | ... | 4 | 2 |
| top | Fiber optic | Month-to-month | ... | Electronic check | FALSE |
| freq | 3096 | 3875 | ... | 2365 | 5163 |

Table 4

The Contract column shows whether customers have a month-to-month, one-year, or two-year contract.

InternetService, PaymentMethod, and other categorical fields reveal different service types customers choose for.

**Outliers Of Numeric Variables:**

**Code Block:**

```
#Graphs for Numerics
#-----------------------------------------------------------------------------------
plt.figure(figsize=(24,20))
plt.subplot(4, 2, 1)
fig = data.boxplot(column='MonthlyCharges',fontsize=15)
fig.set_title('')
fig.set_ylabel('MonthlyCharges',fontsize=15)

plt.subplot(4, 2, 2)
fig = data.boxplot(column='TotalCharges',fontsize=15)
fig.set_title('')
fig.set_ylabel('TotalCharges',fontsize=15)
```

**MonthlyCharges:**

**Mean**: The green line in the box plot represents the median (the middle value). Seen that the average monthly charge is around 65.

**Minimum and Maximum**: The lowest and highest points of the graph show the minimum and maximum values. Monthly charges range between 20 and 120.

**Quartiles**: The upper and lower bounds of the box represent the 25th and 75th percentiles, respectively. This means 50% of the users pay between approximately 40 and 80 in monthly charges.

**Outliers**: There are no points indicating outliers in the graph, suggesting that the distribution appears fairly normal.

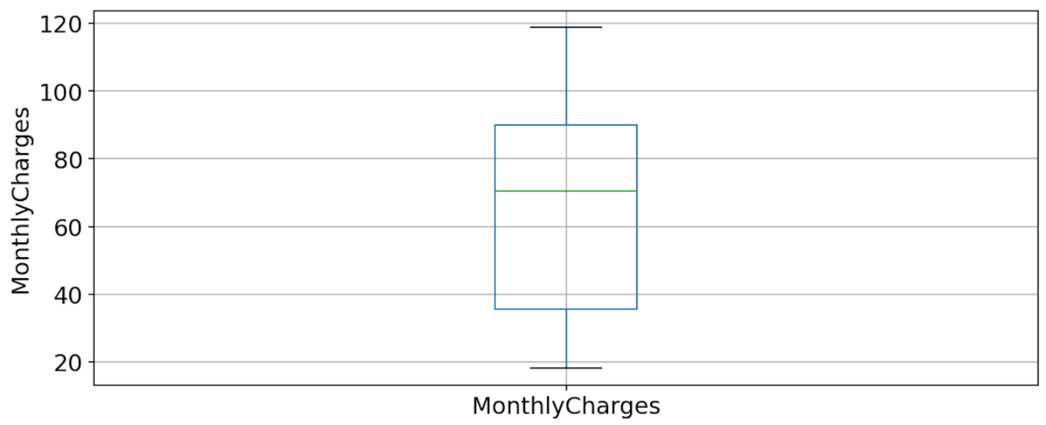


*Figure 2*

<u>**TotalCharges**</u>:

**Mean**: The median for total charges is around 2000.
**Minimum and Maximum**: Total charges range from 0 to as high as almost 8000.
**Quartiles**: Looking at the lower and upper bounds of the box, we can say that 50% of the users have total charges between 1000 and 4000.
**Outliers**: No outliers are visible, indicating that most of the data falls within these limits.

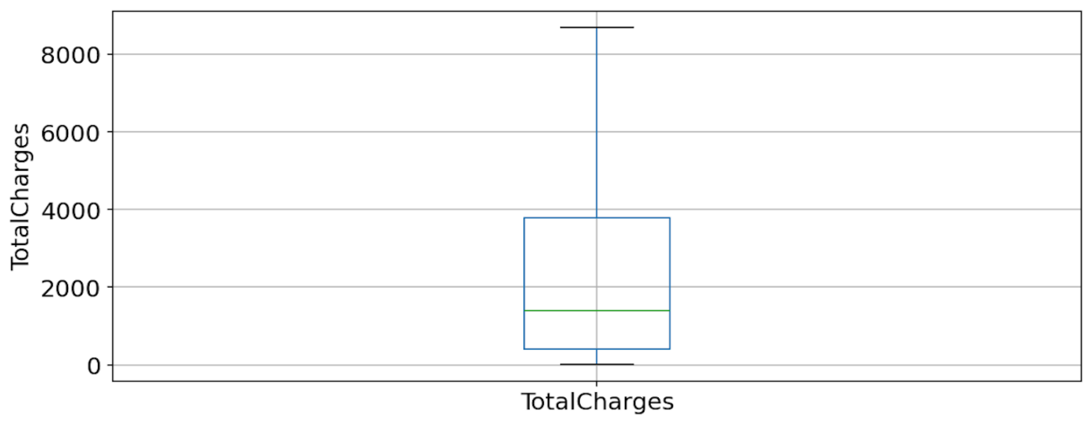


*Figure 3*

<u>**Correlation Matrix:**</u>

<u>**Code Block:**</u>

```
#Correlation between numeric values
numerical_columns = ['tenure', 'MonthlyCharges', 'TotalCharges']
correlation_matrix = data[numerical_columns].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
```

```
plt.title("Correlation Matrix for Numerical Features")
plt.show()
print(correlation_matrix)
```

The correlation matrix shows the relationships between numerical columns:

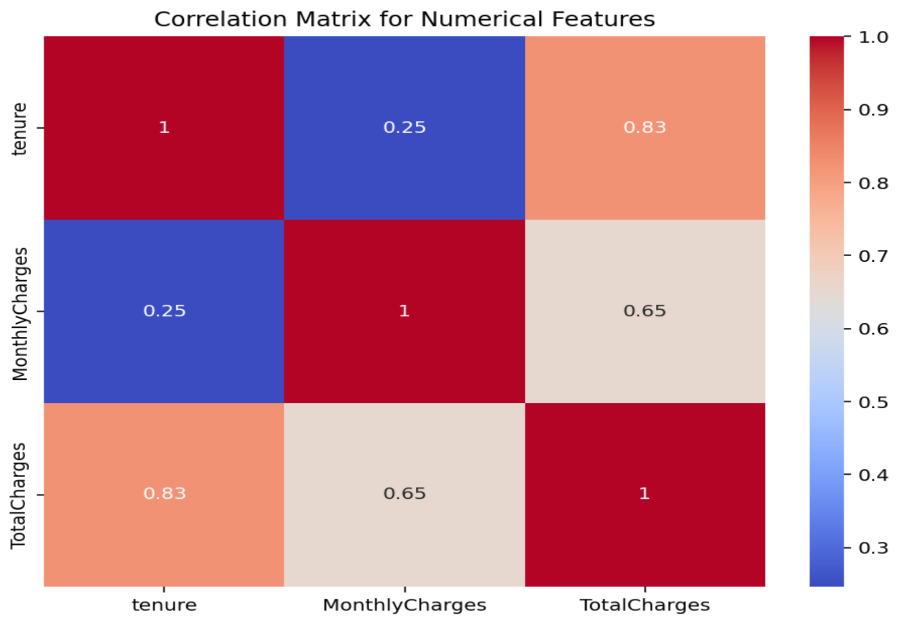


*Figure 4*

**tenure and TotalCharges**: A strong positive correlation (0.83) exists between these variables. This indicates that as the tenure (time with the service) increases, the total charges for the customer also increase, which is expected since longer-term customers accumulate more charges.

**MonthlyCharges and TotalCharges**: A moderately strong positive correlation (0.65) exists here. This suggests that higher monthly charges generally lead to a higher total charge.

**tenure and MonthlyCharges**: A weak positive correlation (0.247) indicates that tenure and monthly charges are not strongly related. This means that customers with longer tenure don't necessarily pay significantly different monthly charges compared to newer customers.

## Churn Percentage:

The target variable in the dataset is Churn, which represents whether a customer has left the service or not.

We observed that a certain percentage of customers churn, which will be important for any future predictive models. The proportion of customers who churn (leave the service) provides critical insights for the business as they can use this information to reduce churn and retain customers.

| Churn Category | Percentage |
|---|---|
| FALSE | %73.42 |
| TRUE | %26.57 |

*Table 5*

Distribution of target variable is demonstrated on Figure 5:

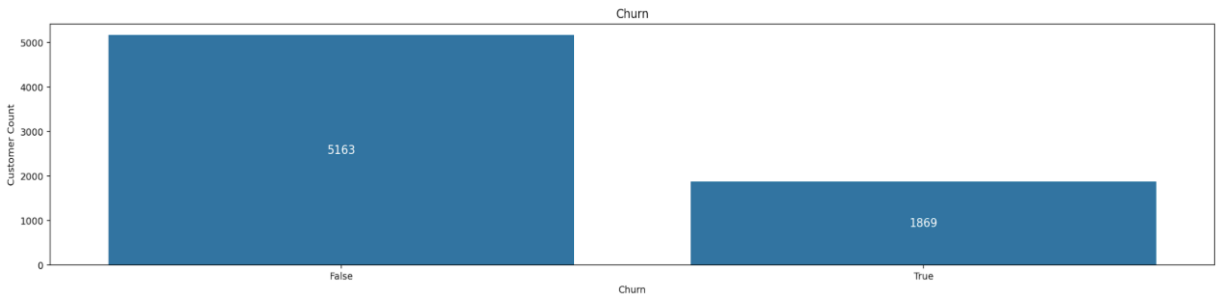


*Figure 5*

Introduction: This report outlines the steps taken to preprocess and visualize the Telecom Churn dataset. The primary goal was to prepare the data for analysis by handling discrepancies, encoding categorical variables, and scaling numerical features.

## b. Feature Engineering

### Object-Type Column Exploration

The script identifies the columns that contain object-type data (likely categorical features) and prints their unique values.

**Code Block:**

```
df.columns = df.columns.str.strip().str.lower()
print("\nColumn Names after standardization:")
print(df.columns.tolist())

object_cols = df.select_dtypes(include=['object']).columns.tolist()
print("\nObject-type columns:")
print(object_cols)

for col in object_cols:
    unique_values = df[col].unique()
    print(f"\nUnique values in '{col}':")
    print(unique_values)
```

### Binary Encoding

Certain columns with binary values (e.g., gender, partner, churn) are mapped to numerical values (0 or 1). This includes columns like:

gender: {'Female': 0, 'Male': 1}

churn: {'Yes': 1, 'No': 0}

For each column mapped, the unique values are printed after the transformation.

**Code Block:**

```
binary_mappings = {
    'gender': {'Female': 0, 'Male': 1},
    'partner': {'Yes': 1, 'No': 0},
    'dependents': {'Yes': 1, 'No': 0},
    'phoneservice': {'Yes': 1, 'No phone service': 0, 'No': 0},
    'onlinesecurity': {'Yes': 1, 'No internet service': 0, 'No': 0},
    'onlinebackup': {'Yes': 1, 'No': 0},
    'deviceprotection': {'Yes': 1, 'No': 0},
    'techsupport': {'Yes': 1, 'No': 0},
    'streamingtv': {'Yes': 1, 'No': 0},
    'streamingmovies': {'Yes': 1, 'No': 0},
    'paperlessbilling': {'Yes': 1, 'No': 0},
    'churn': {'Yes': 1, 'No': 0}
}
```

**Handling Missing Values**

The script checks for missing values after binary mapping and prints the results.

**One-Hot Encoding**

For columns with multiple categories, such as contract and internetservice, one-hot encoding is applied to convert them into numerical representations (creating binary columns for each category).

**Numeric Conversion**

The script converts totalcharges and monthlycharges to numeric data types, handling any errors during conversion (e.g., non-numeric values are coerced into NaN).

**Checking 'Tenure' Column**

The presence of the tenure column is verified, and if it is missing, the script exits.

**Dropping Unnecessary Columns**

The customerid column is dropped, as it is not needed for analysis.

**Imputation of Missing Values**

The script identifies numeric columns and fills missing values with the mean of the respective column.

**Feature Engineering**

charges_per_month: A new feature is created by dividing totalcharges by tenure (with 0 tenure replaced by 1 to avoid division by zero).

high_monthly_charge: A binary feature is created to indicate whether monthlycharges is above the median value.

**Code Block:**

```
#14. Feature Engineering
#Create ‘charges_per_month’
df['charges_per_month'] = df['totalcharges'] / df['tenure'].replace(0, 1)
print("\nCreated 'charges_per_month' feature.")

#Create ‘hig_monthly_charge’
median_charge = df['monthlycharges'].median()
```

```
df['high_monthly_charge'] = (df['monthlycharges'] > median_charge).astype(int)
print("Created 'high_monthly_charge' feature based on the median of
'monthlycharges'.")
```

**Correlation Matrix**

A subset of the data is selected to compute the correlation matrix for the churn and various other features like gender, partner, and dependents.

**Code Block:**

```
df_f = df[[
    'gender', 'partner', 'dependents', 'phoneservice', 'onlinesecurity',
    'onlinebackup', 'deviceprotection', 'techsupport', 'streamingtv',
    'streamingmovies', 'paperlessbilling', 'churn'
]]

# correlation matrix
correlation_matrix = df_f.corr()
print("\nCorrelation Matrix:")
print(correlation_matrix)
```

### c. Feature Scaling

**Handling Discrepancies:**

Certain columns (onlinebackup, deviceprotection, techsupport, streamingtv, streamingmovies) were rounded to correct discrepancies.

**Code Block:**

```
onlinebackup = [1.0, 0.0, 1.0, ..., 0.0]
deviceprotection = [0.0, 1.0, 0.0, ..., 1.0]
```

**Feature Engineering:**

A new binary feature, long_term_customer, was created to identify customers with tenure greater than the average tenure.

The long_term_customer feature categorizes customers based on their tenure compared to the average:

**Code Block:**

```
long_term_customer = [0, 1, 0, ..., 1]
```

A value of 1 indicates customers with tenure above the average, highlighting a potential segment of loyal customers.

**Encoding Categorical Variables:**

One-hot encoding was applied to the paymentmethod column, excluding the first category to avoid multicollinearity.

Binary encoding was applied to several categorical columns to convert them into numerical format.

**Code Block:**

```
paymentmethod_Mailed_check = [False, True, True, ..., False]
```

**Converting Categorical Data into Binary:**

Additional categorical variables were converted into binary format, such as:

**Code Block:**

```
paymentmethod_Credit_card_automatic = [0, 0, 0, ..., 0]
service internetservice_No = [0, 0, 0, ..., 0]
```

This step ensures that all categorical data is represented in a format suitable for machine learning algorithms.

**Feature Scaling:**

Numerical columns (tenure, monthlycharges, totalcharges) were standardized using StandardScaler from the sklearn.preprocessing module.

The numeric columns (tenure, monthlycharges, totalcharges) were standardized to have a mean of 0 and a standard deviation of 1. The scaled values for these features are shown below:

**Code Block:**

```
tenure = [-1.277445, 0.066327, -1.236724, ...]
monthlycharges = [-1.160323, -0.259629, -0.362660, ...]
totalcharges = [-0.994971, -0.173876, -0.960399, ...]
```

Standard scaling is essential for ensuring that all features contribute equally to the analysis, particularly for algorithms sensitive to the scale of input data.

**Data Visualization and Insights:**

Reviewing the histograms for the tenure, monthlycharges, and totalcharges both before and after scaling:

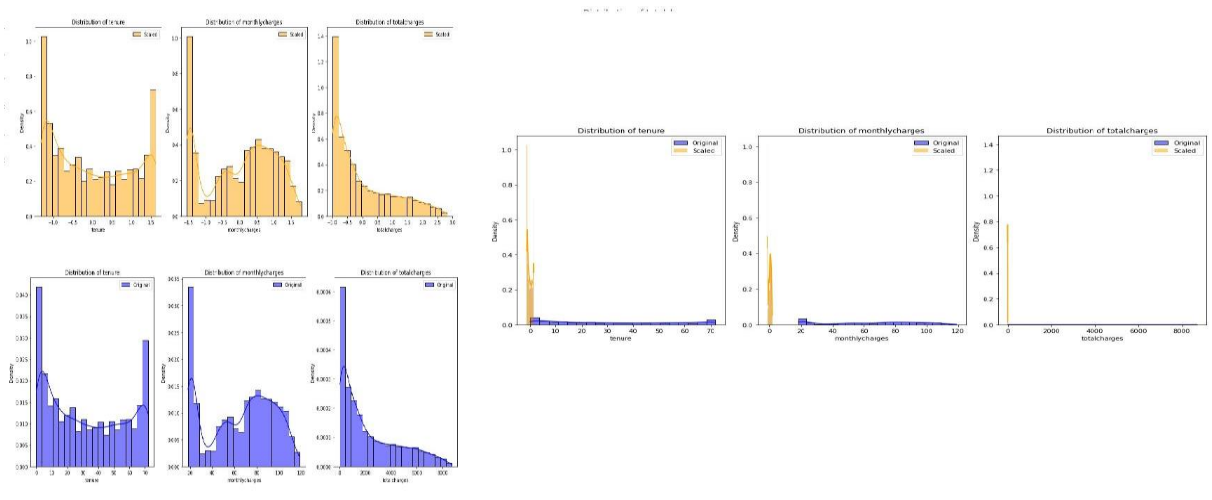


*Figure 6*

**Tenure:**

The original data shows two modes, indicating groups of new versus long-term customers.

The scaled data centers these groups around zero, emphasizing their relative positions in the customer lifecycle.

**Monthly Charges:**

The original data is right-skewed, suggesting many customers are on lower-cost plans, with fewer on high-cost plans.

Scaling normalizes this feature, which helps in comparing its effects directly with other features in a model.

**Total Charges:**

The original total charges are also right-skewed, showing many lower spending customers.

The scaled data, now centered and reduced in range, is more amenable to statistical methods that assume data normality.

# 4. Random Forest Model

The analysis includes two phases: the initial Random Forest model and the best model after hyperparameter tuning using GridSearchCV. Both models were evaluated using various metrics, including precision, F1 score, accuracy, and confusion matrices.

a. Data Preparation:

The dataset was loaded from a CSV file. Features were separated into X (independent variables) and y (dependent variable indicating churn). The data was split into training and testing sets, with 70% used for training and 30% for testing.

b. Model Evaluation:

**Initial Random Forest Model:** The first Random Forest classifier was trained with default parameters, including 10 trees and no depth limit. The results were as follows:

**Accuracy:** 78.37%
**Precision:** 76.95%
**Recall:** 78.37%
**F1 Score:** 76.93%

**Confusion Matrix:**

```
[[1402  137]
 [ 320  254]]
```

c. Best Model after Hyperparameter Tuning:

The second model was trained using GridSearchCV to identify the best combination of hyperparameters. After tuning, the best model had the following parameters:

**n_estimators:** 15

**max_depth:** 20

**min_samples_split**: 5

**min_samples_leaf:** 2

**max_features:** 'sqrt'

**The results of the best model were:**

**Precision:** 78.64%

**F1 Score**: 78.57%

**Accuracy:** 79.79%

**Confusion Matrix:**

```
[[1410  129]
 [ 298  276]]
```

d. Model Conclusion:

The Random Forest Classifier demonstrated significant improvement after hyperparameter tuning. The initial model had a reasonable accuracy of 78.37%, but after tuning, the accuracy increased to 79.79% . The tuned model also showed better balance in predicting both churners and non-churners, with improved precision, F1 score, and a reduced number of misclassifications in the confusion matrix. This highlights the importance of model tuning in improving the model's ability to generalize and make accurate predictions in a telecom churn prediction task.

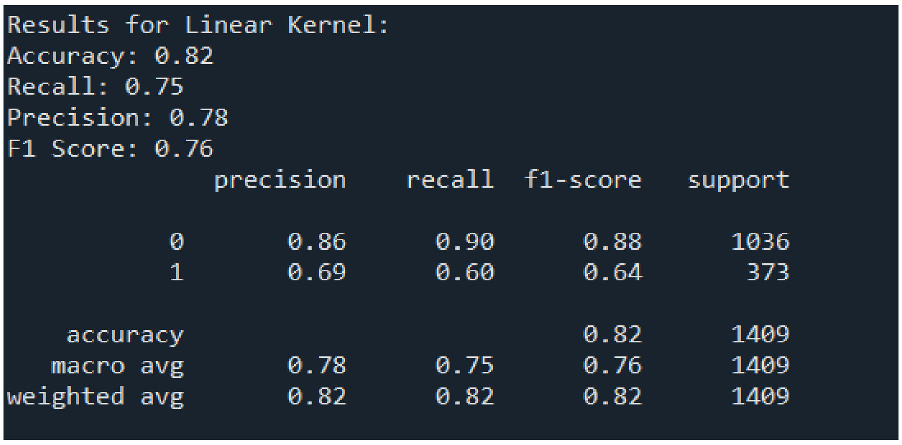
## 5. Support Vector Machine Model (SVM)

**Code Block:**

```
X = data.drop('churn', axis=1)
y = data['churn']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# SVM with Linear kernel
linear_model = svm.SVC(kernel='linear', random_state=42)
linear_model.fit(X_train, y_train)
```

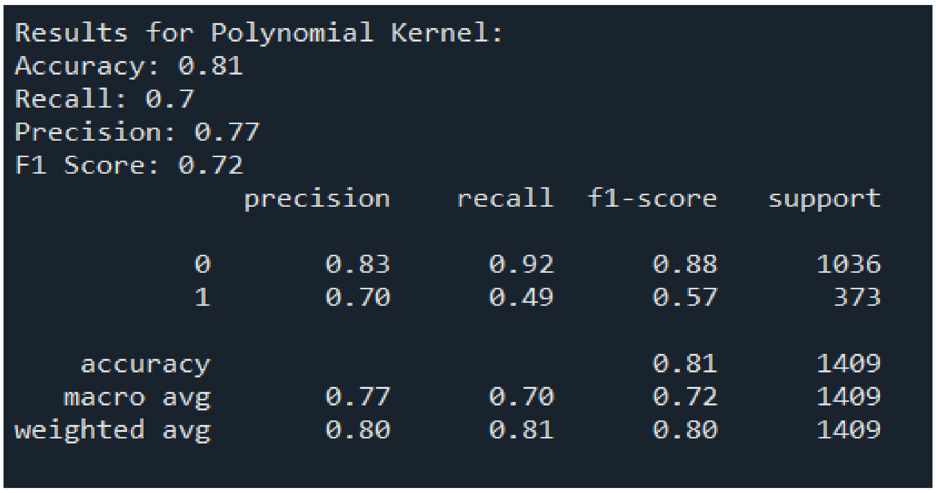
```python
y_pred_linear = linear_model.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)
recall_linear = recall_score(y_test, y_pred_linear, average='macro')
precision_linear = precision_score(y_test, y_pred_linear, average='macro')
f1_linear = f1_score(y_test, y_pred_linear, average='macro')
classification_report_linear = classification_report(y_test, y_pred_linear)

print("Results for Linear Kernel:")
print(f"Accuracy: {round(accuracy_linear, 2)}")
print(f"Recall: {round(recall_linear, 2)}")
print(f"Precision: {round(precision_linear, 2)}")
print(f"F1 Score: {round(f1_linear, 2)}")
print(classification_report_linear)
print("\n" + "-"*50 + "\n")

# SVM with Polynomial kernel
poly_model = svm.SVC(kernel='poly', random_state=42)
poly_model.fit(X_train, y_train)
y_pred_poly = poly_model.predict(X_test)
accuracy_poly = accuracy_score(y_test, y_pred_poly)
recall_poly = recall_score(y_test, y_pred_poly, average='macro')
precision_poly = precision_score(y_test, y_pred_poly, average='macro')
f1_poly = f1_score(y_test, y_pred_poly, average='macro')
classification_report_poly = classification_report(y_test, y_pred_poly)

print("Results for Polynomial Kernel:")
print(f"Accuracy: {round(accuracy_poly, 2)}")
print(f"Recall: {round(recall_poly, 2)}")
print(f"Precision: {round(precision_poly, 2)}")
print(f"F1 Score: {round(f1_poly, 2)}")
print(classification_report_poly)
print("\n" + "-"*50 + "\n")

# SVM with RBF kernel
rbf_model = svm.SVC(kernel='rbf', random_state=42)
rbf_model.fit(X_train, y_train)
y_pred_rbf = rbf_model.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
recall_rbf = recall_score(y_test, y_pred_rbf, average='macro')
precision_rbf = precision_score(y_test, y_pred_rbf, average='macro')
f1_rbf = f1_score(y_test, y_pred_rbf, average='macro')
classification_report_rbf = classification_report(y_test, y_pred_rbf)

print("Results for RBF Kernel:")
print(f"Accuracy: {round(accuracy_rbf, 2)}")
print(f"Recall: {round(recall_rbf, 2)}")
print(f"Precision: {round(precision_rbf, 2)}")
print(f"F1 Score: {round(f1_rbf, 2)}")
print(classification_report_rbf)
print("\n" + "-"*50 + "\n")

# SVM with Sigmoid kernel
sigmoid_model = svm.SVC(kernel='sigmoid', random_state=42)
sigmoid_model.fit(X_train, y_train)
y_pred_sigmoid = sigmoid_model.predict(X_test)
accuracy_sigmoid = accuracy_score(y_test, y_pred_sigmoid)
recall_sigmoid = recall_score(y_test, y_pred_sigmoid, average='macro')
precision_sigmoid = precision_score(y_test, y_pred_sigmoid, average='macro')
f1_sigmoid = f1_score(y_test, y_pred_sigmoid, average='macro')
classification_report_sigmoid = classification_report(y_test, y_pred_sigmoid)

print("Results for Sigmoid Kernel:")
print(f"Accuracy: {round(accuracy_sigmoid, 2)}")
print(f"Recall: {round(recall_sigmoid, 2)}")
print(f"Precision: {round(precision_sigmoid, 2)}")
print(f"F1 Score: {round(f1_sigmoid, 2)}")
print(classification_report_sigmoid)
```

**Outputs:**



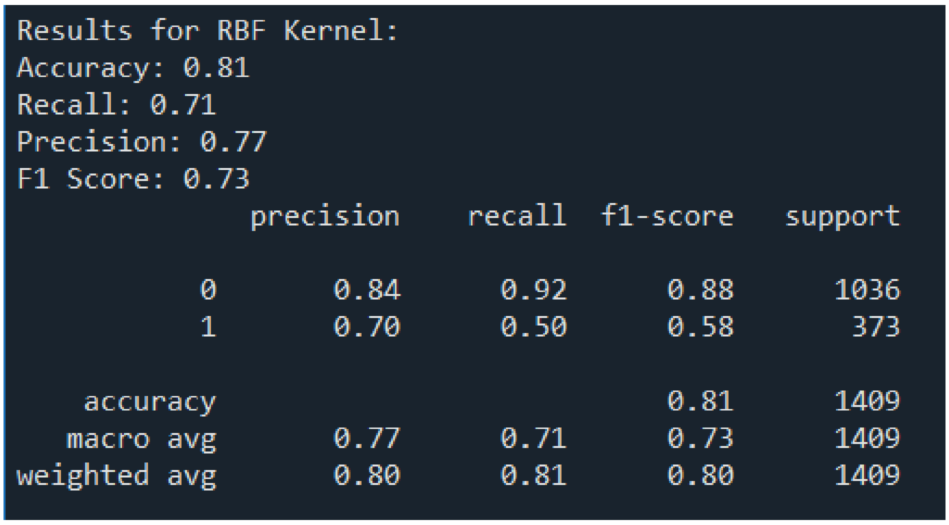
```
Results for Linear Kernel:
Accuracy: 0.82
Recall: 0.75
Precision: 0.78
F1 Score: 0.76
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      1036
           1       0.69      0.60      0.64       373

    accuracy                           0.82      1409
   macro avg       0.78      0.75      0.76      1409
weighted avg       0.82      0.82      0.82      1409
```

*Figure 7*



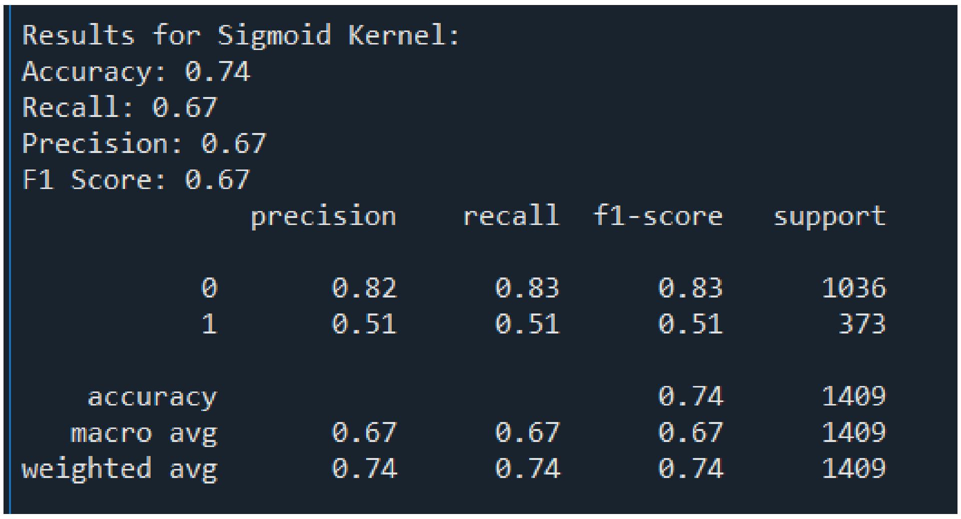
```
Results for Polynomial Kernel:
Accuracy: 0.81
Recall: 0.7
Precision: 0.77
F1 Score: 0.72
              precision    recall  f1-score   support

           0       0.83      0.92      0.88      1036
           1       0.70      0.49      0.57       373

    accuracy                           0.81      1409
   macro avg       0.77      0.70      0.72      1409
weighted avg       0.80      0.81      0.80      1409
```

*Figure 8*



```
Results for RBF Kernel:
Accuracy: 0.81
Recall: 0.71
Precision: 0.77
F1 Score: 0.73
              precision    recall  f1-score   support

           0       0.84      0.92      0.88      1036
           1       0.70      0.50      0.58       373

    accuracy                           0.81      1409
   macro avg       0.77      0.71      0.73      1409
weighted avg       0.80      0.81      0.80      1409
```

*Figure 9*



```
Results for Sigmoid Kernel:
Accuracy: 0.74
Recall: 0.67
Precision: 0.67
F1 Score: 0.67
              precision    recall  f1-score   support

           0       0.82      0.83      0.83      1036
           1       0.51      0.51      0.51       373

    accuracy                           0.74      1409
   macro avg       0.67      0.67      0.67      1409
weighted avg       0.74      0.74      0.74      1409
```

*Figure 10*

## General Performance

**Linear Kernel:** This has the highest accuracy (82%), along with good Recall (75%) and F1 Score (76%). It offers a balanced performance across most metrics, making it a strong candidate.

**RBF Kernel:** While it also shows high accuracy (81%) and a fairly strong Recall (71%), it struggles with class 1, as its Recall for class 1 is only 50%. This suggests that the model might have difficulties identifying minority classes.

**Polynomial Kernel:** With 81% accuracy, it's close to the Linear Kernel in performance. However, both its Recall (70%) and F1 Score (72%) are slightly lower than the Linear Kernel.

**Sigmoid Kernel:** This kernel has the lowest accuracy (74%), Recall (67%), and F1 Score (67%), which makes it the weakest performer overall compared to the others.

**Result:**

**Based on these results, the Linear Kernel is the best option due to its balanced and consistent performance.**

## 6. Decision Tree Classifier

The analysis includes two models based on different criteria (Entropy and Gini), followed by a pruning process to enhance model performance. The results are evaluated using various metrics, including precision, F1 score, accuracy, and confusion matrices.

a. Data Preparation:

The dataset was loaded from a CSV file, and the features were separated into X (independent variables) and y (dependent variable, which indicates churn). The data was then split into training and testing sets, with 80% of the data used for training and 20% for testing.

b. Model Evaluation:

**Model with Entropy Criterion:**

The Decision Tree Classifier was first trained using the entropy criterion. The results were as follows:

**Precision:** 0.51
**F1 Score:** 0.50
**Accuracy:** 73.95%
**Confusion Matrix:**

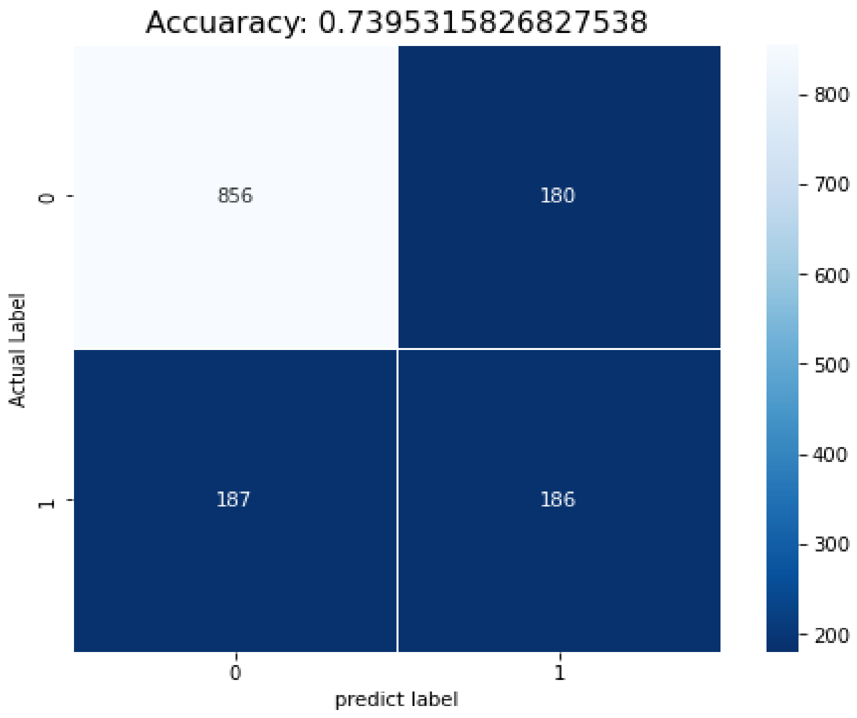


*Figure 11*

**<u>Classification Report:</u>**

The model performed well in predicting non-churners (class 0) with a precision of 0.82, but struggled with churners (class 1), achieving a precision of only 0.51.

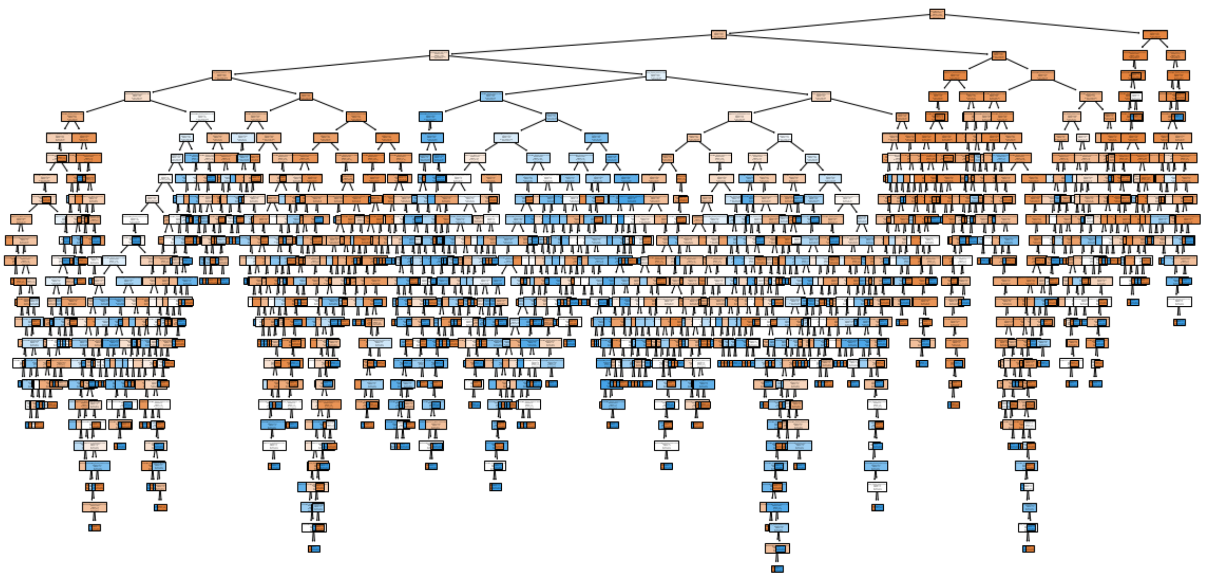


*Figure 12*

**<u>Model with Gini Criterion:</u>**

The second model used the Gini criterion for training. The results were:

**Precision:** 0.46

**F1 Score:** 0.46

**Accuracy:** 71.33%

**Confusion Matrix:**

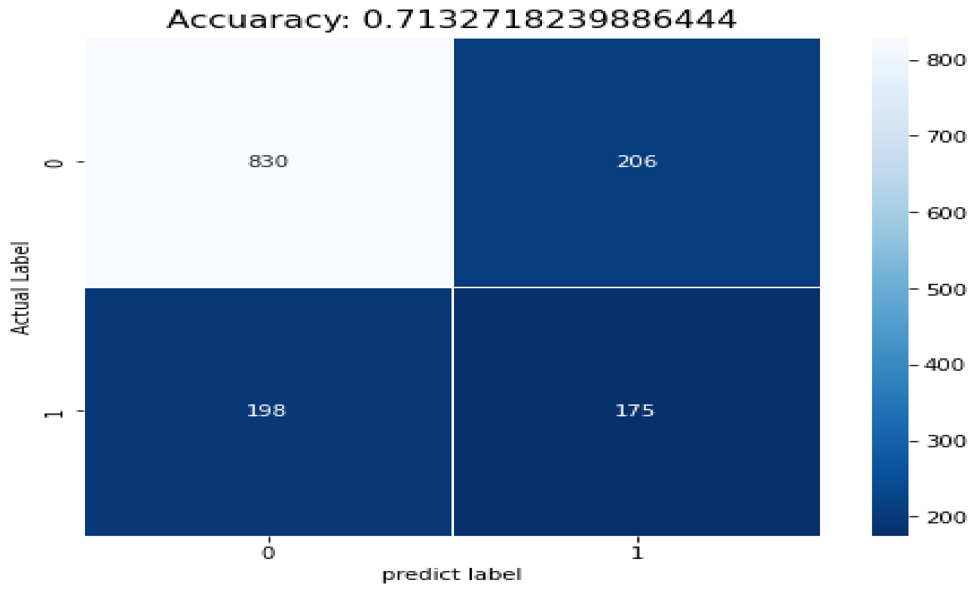


*Figure 13*

<u>**Classification Report:**</u>

Similar to the entropy model, this model also performed better on non-churners (precision of 0.81) compared to churners (precision of 0.46).

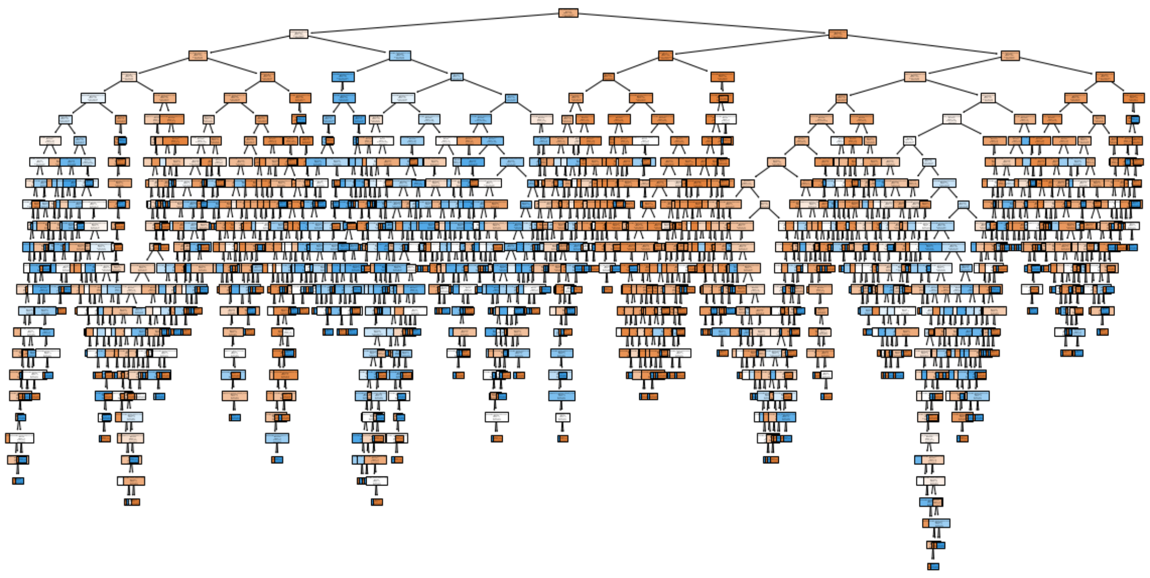


*Figure 14*

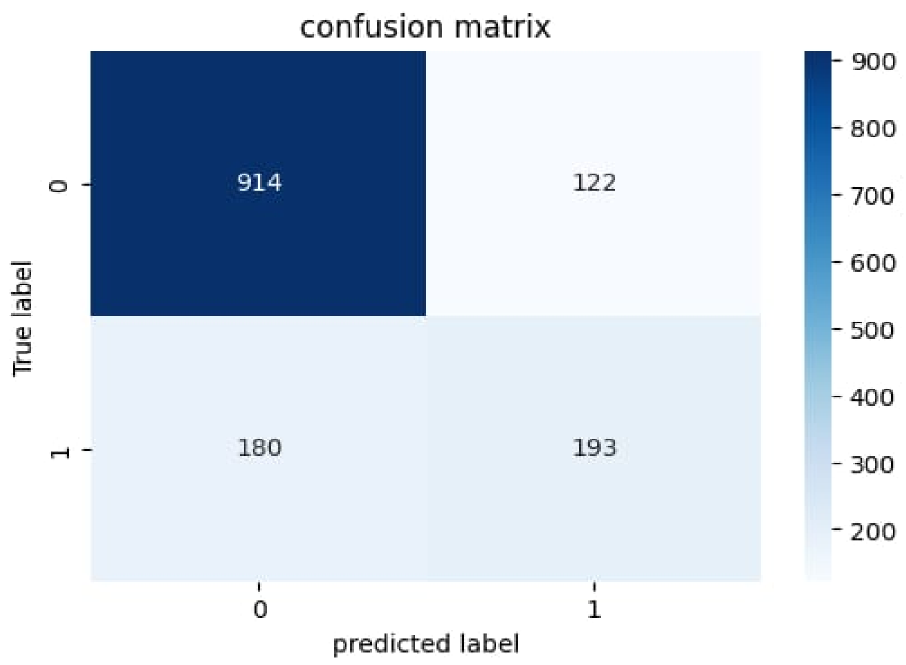
<u>**Model after Hyperparameter Tuning:**</u>

A comprehensive parameter grid was used to find the optimal combination of hyperparameters using **GridSearchCV**:

**criterion**: gini, entropy
**splitter**: best, random
**max_depth**: None, 10, 20, 30, 40
**min_samples_split**: 2, 10, 20
**min_samples_leaf**: 1, 5, 10
**max_features**: None, sqrt, log2

**The results of the tuned model on the test set were as follows:**
**Accuracy**: 78.56%
**Precision**: 0.61
**F1 Score**: 0.56

**Confusion Matrix:**



<u>**Model After Pruning:**</u>

The final model was trained after applying pruning techniques to reduce overfitting. The results were significantly improved:

**Precision:** 0.69
**F1 Score:** 0.59
**Accuracy:** 81.12%
**Confusion Matrix:**

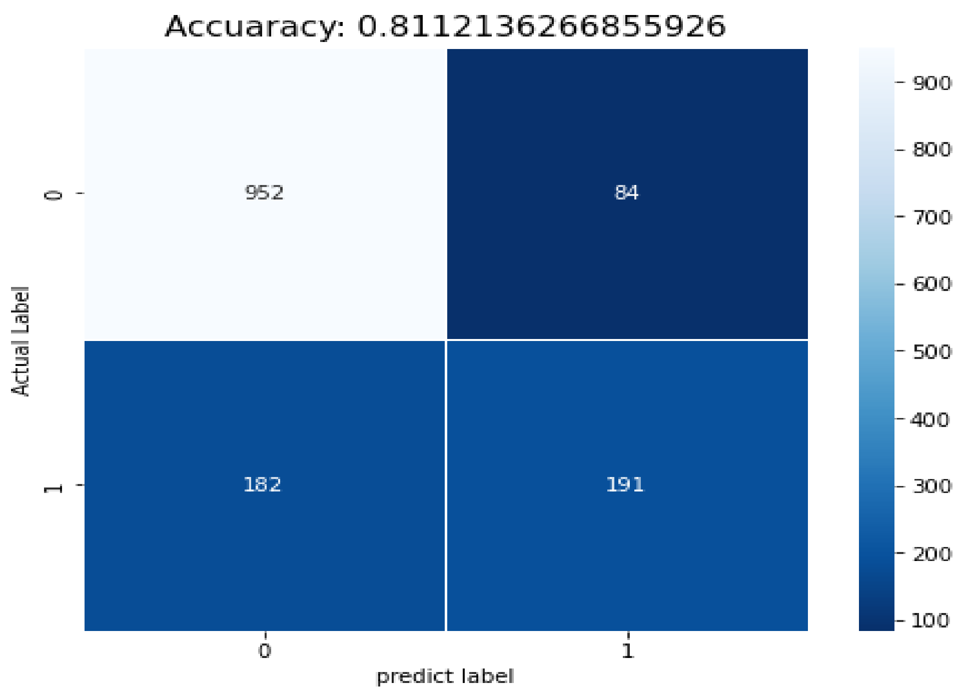


*Figure 15*

**Classification Report:**

The pruned model showed a better balance, with a precision of 0.84 for non-churners and 0.69 for churners, indicating improved predictive capability.

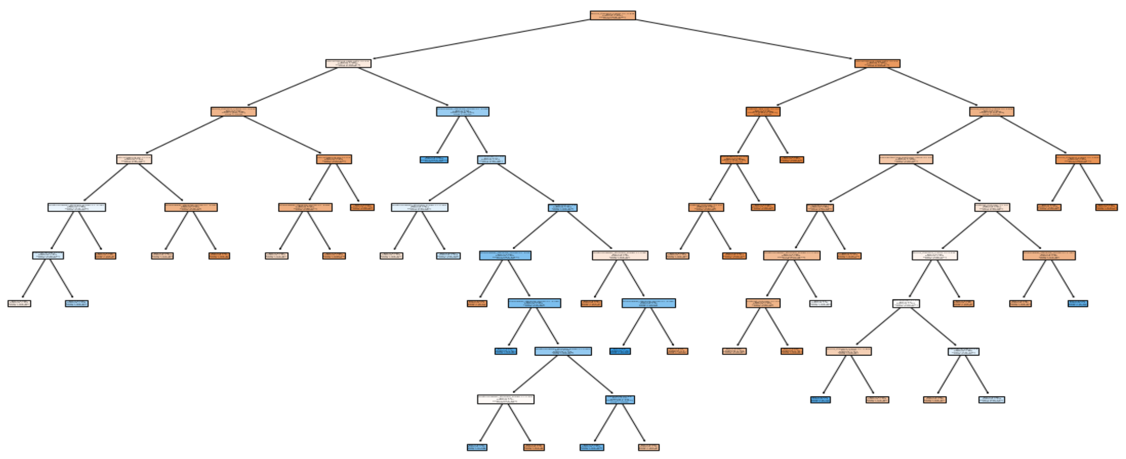


*Figure 16*

c. Model Conclusion:

The Decision Tree Classifier demonstrated varying performance based on the criterion used for training. The entropy-based model provided a reasonable accuracy of 73.95%, while the Gini-based model performed slightly worse at 71.33%. However, after applying pruning techniques, the model's accuracy improved to 81.12%, indicating that pruning effectively reduced overfitting and enhanced the model's ability to generalize on unseen data. The results highlight the importance of model selection and tuning in achieving better predictive performance in churn analysis.

## 7. Conclusion

In our Telecom Churn Prediction project, we set out to understand which customers might leave the service, using machine learning to analyze key factors like demographics, account details, and service usage. We tested several models such as Random Forest, SVM, and Decision Trees, to see which one could best predict customer churn. Along the way, we learned how crucial data preparation and fine-tuning the models were in improving accuracy.

Among the models we used, the SVM model with linear kernel, achieving an impressive 82% accuracy and a 76% F1 score. This means it was especially good at identifying both those likely to churn and those who would stay. Random Forest also performed well, reaching 79.79% accuracy after tweaking its parameters, while Decision Trees showed improvement after pruning to prevent overfitting.

This project demonstrated how machine learning, combined with thoughtful data processing and optimization, can effectively predict customer churn. These findings provide telecom companies with valuable insights, enabling them to take proactive steps to retain customers and improve satisfaction, ultimately supporting long-term business success.

# 8. Bibliography

Anon., n.d. *IBM.* [Online]
Available at: https://www.ibm.com/topics/random-forest

Anon., n.d. *IBM.* [Online]
Available at: https://www.ibm.com/topics/decision-trees

Boswell, D., 2002. Introduction to Support Vector Machines. *Introduction to Support Vector Machines,* p. 1.

Camastra, F. & Vinciarelli, A., 2008. *Machine Learning for Audio, Image and Video Analysis Theory and Applications.* s.l.:Springer.

Fieguth, P., 2022. *An Introduction to Pattern Recognition and Machine Learning.* s.l.:Springer.

Phatthanaphan, T., 2023. *GitHub.* [Online]
Available at: https://github.com/ohmthanap/Telecom-Customer-Churn-Predictions/blob/main/Telco_Customer_Churn_Dataset.csv