# Akademi (Flatiron School) — Data Science & AI, Cohorte 2025

## Third Project – Phase 3

**Student Name**: Micka LOUIS
**Student Pace**: Self-paced
**Submission Deadline**: August 24, 2025
**Instructors' Names**: Wedter JEROME & Geovany Batista Polo LAGUERRE
**Blog Post URL**: https://github.com/Micka-Louis/ds-project-phase-3.git

## Project Title

## Churn Prediction in Telecommunications using Machine Learning

Cover

## Overview

In the telecommunications sector, competition is intense and customers have many options.
**Churn** (the voluntary cancellation of a subscription) represents a significant loss of revenue for companies and can negatively impact long-term profitability.

The goal of this project is to build a **machine learning model** capable of predicting whether a customer is likely to churn, in order to identify high-risk profiles and support targeted retention strategies.

## Business Understanding & Business Problem

Cover **Business Understanding**
Customer retention is a key strategic issue for telecommunications companies.
Reducing churn not only increases revenue but also reduces the costs associated with acquiring new customers.

**Business Problem**
The company faces a critical question:
➡️ *Can we identify in advance the customers who are most likely to leave the service?*

An effective predictive model will enable the business to:

- Anticipate customer churn.
- Target marketing actions and loyalty programs.
- Optimize the allocation of commercial resources.

## 1-Data Understanding

The data comes from SyriaTel and includes information about their customers. The dataset has customer's state of residence, telephone numbers and length of the account. There are columns indicating if the customer has an international plan and voicemail plan, how many voice mails they receive. The dataset includes how many minutes they spend talking, how many calls they make and how much they are charged during day, evening and night periods.

## Stakeholder Audience

Telecommunications executives, marketing teams, and customer success managers can leverage this analysis to better understand customer behavior and identify those most likely to churn.
By leveraging the dataset's detailed billing, call patterns, and service usage data, predictive models can be developed to support retention campaigns, reduce customer loss, and improve overall profitability. Cover

## 1.1 Importing Required Libraries

```
# Data manipulation and analysis
import numpy as np
import pandas as pd

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning and Preprocessing
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import plot_tree
from sklearn.metrics import (
    classification_report,
    f1_score,
    precision_score,
    recall_score,accuracy_score,
    roc_auc_score,
    confusion_matrix
)

# Handling class imbalance
from imblearn.over_sampling import SMOTE
```

## 1.2 Loading the data

```
df= pd.read_csv('data/churn.csv')
```

## Initial Exploration and Data Quality Assessment

```
#Display the first 5 rows of the dataset
df.head()
```

| | state | account length | area code | phone number | international plan | voice mail plan | number vmail messages | total day minutes | total day calls | total day charge | ... | total eve calls | total eve charge |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | KS | 128 | 415 | 382-4657 | no | yes | 25 | 265.1 | 110 | 45.07 | ... | 99 | 16.78 |
| **1** | OH | 107 | 415 | 371-7191 | no | yes | 26 | 161.6 | 123 | 27.47 | ... | 103 | 16.62 |
| **2** | NJ | 137 | 415 | 358-1921 | no | no | 0 | 243.4 | 114 | 41.38 | ... | 110 | 10.30 |
| **3** | OH | 84 | 408 | 375-9999 | yes | no | 0 | 299.4 | 71 | 50.90 | ... | 88 | 5.26 |
| **4** | OK | 75 | 415 | 330-6626 | yes | no | 0 | 166.7 | 113 | 28.34 | ... | 122 | 12.61 |

5 rows × 21 columns

```
#Display the names of all available columns in the DataFrame
df.columns
```

Show hidden output

```
#Display the number of rows and columns in the dataset
df.shape
```

```
(3333, 21)
```

```
df.info()
```

Show hidden output

```
#Display the number of missing values per column, sorted in descending order
df.isnull().sum().sort_values(ascending=False)
```

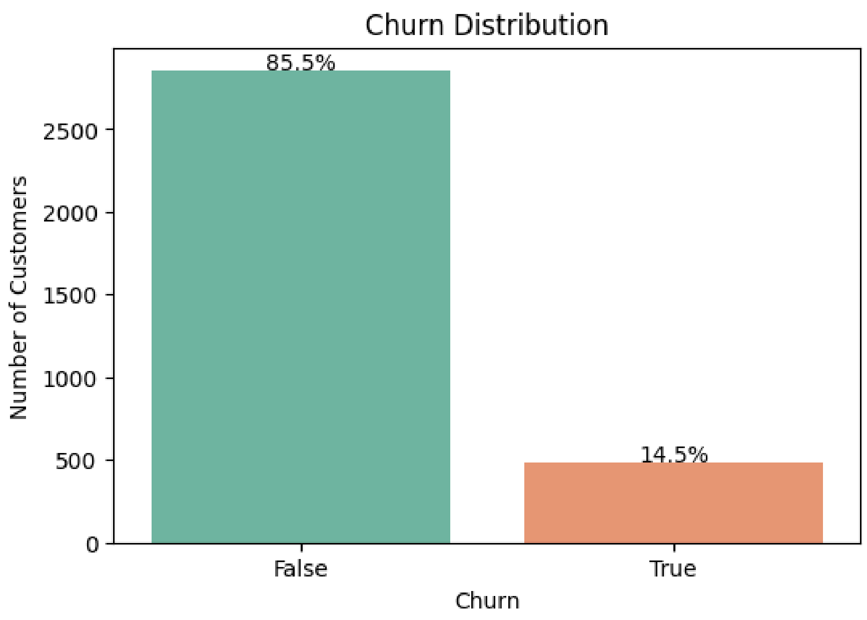
```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

This data set is small, but very clean. There are no missing values to handle. Let's look at what the churn rate is over the data set. We are dealing with a very imbalanced data set.

```
df["churn"].value_counts(normalize=True)
```

```
churn
False    0.855086
True     0.144914
Name: proportion, dtype: float64
```

```
# Count plot for churn (fixes FutureWarning)
plt.figure(figsize=(6,4))
sns.countplot(x='churn', data=df, hue='churn', palette='Set2', dodge=False, legend=False)
plt.title('Churn Distribution')
plt.xlabel('Churn')
plt.ylabel('Number of Customers')

# Add percentages on top of each bar
total = len(df)
for p in plt.gca().patches:
    height = p.get_height()
    plt.gca().text(p.get_x() + p.get_width()/2., height + 5,
                   '{:.1f}%'.format(100*height/total),
                   ha="center")
plt.show()
```



The bar chart shows the distribution of the target variable churn.

The majority of customers, around 85%, stayed with the company.

Only about 15% of customers churned (left the service).

This imbalance indicates that the dataset is skewed, which will be important to consider when building predictive models, as the model could otherwise be biased toward predicting customers as non-churners.

## 2-Data Preparation

### 2.1 Feature Engineering: Combining Usage Metrics

To better capture customer behavior, we created new features by combining existing columns:

Total Domestic Minutes – sum of total day minutes, total eve minutes, and total night minutes.

Total Domestic Calls – sum of total day calls, total eve calls, and total night calls.

Total Domestic Charges – sum of total day charge, total eve charge, and total night charge.

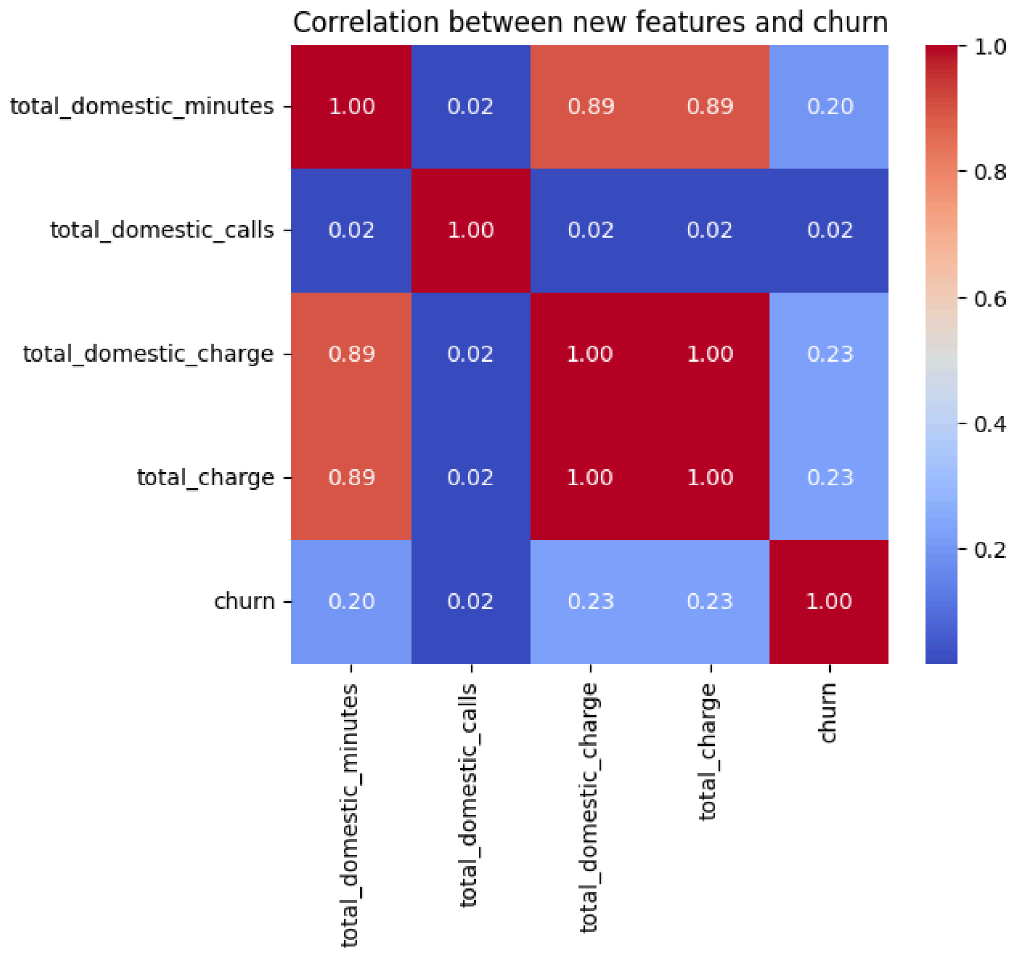
Total Charges – sum of total domestic charge and total intl charge.

These new features provide a more complete view of each customer's usage, which can improve the predictive power of our models.

```
def combine(name, *cols):
    "This function will name a new column and add as many columns as necessary"
    df[name] = sum(cols)


#Let's find total domestic minutes for each customer.
combine("total_domestic_minutes",
        df["total day minutes"],
        df["total eve minutes"],
        df["total night minutes"])

#Let's find total domestic calls for each customer.
combine("total_domestic_calls",
        df["total day calls"],
        df["total eve calls"],
        df["total night calls"])

#Let's find total domestic charges for each customer.
combine("total_domestic_charge",
        df["total day charge"],
        df["total eve charge"],
        df["total night charge"])

#Let's find total charges for each customer.
combine("total_charge",
        df["total_domestic_charge"],
        df["total intl charge"])


# Check correlation with churn
correlation = df[['total_domestic_minutes', 'total_domestic_calls', 'total_domestic_charge', 'total_charge', 'ch
print(correlation['churn'].sort_values(ascending=False))
```

```
churn                     1.000000
total_charge              0.231549
total_domestic_charge     0.226962
total_domestic_minutes    0.196457
total_domestic_calls      0.019651
Name: churn, dtype: float64
```

```
plt.figure(figsize=(6,5))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation between new features and churn")
plt.show()
```



The new features show the following correlation with churn:

total_charge (0.23) and total_domestic_charge (0.23) have the strongest positive correlation.

total_domestic_minutes (0.20) has a moderate correlation.

total_domestic_calls (0.02) is almost uncorrelated.

Insight: Total charges are the most relevant features for predicting churn, while the number of calls is not very informative.

```
# Drop irrelevant columns
df = df.drop(['state', 'phone number', 'area code'], axis=1)
```

## 3-Exploratory Data Analysis (EDA)

This section introduces two reusable functions for automated univariate analysis:

```
 -plot_categorical_univariate() → Plots countplots for all categorical variables.

 -plot_numeric_univariate() → Plots histograms and boxplots for all numerical variables.
```

This allows us to quickly explore the distribution of our data and detect imbalances or outliers.

### 3.1 Function for categorical variables

```
def plot_categorical_univariate(df, cat_cols):
    """
    Displays countplots for each categorical variable.
    """
    for col in cat_cols:
        plt.figure(figsize=(6,4))
        sns.countplot(x=col, data=df, hue=col, palette='Set2', legend=False)
        plt.title(f"Distribution of {col}")
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()
```

## 3.2 Function for numerical variables

```
def plot_numerical_univariate(df, num_cols):
    """
    Displays histograms and boxplots for each numerical variable.
    """
    for col in num_cols:
        fig, axes = plt.subplots(1, 2, figsize=(12,4))

        # Histogram
        sns.histplot(df[col], kde=True, ax=axes[0], color='skyblue')
        axes[0].set_title(f"Histogram of {col}")

        # Boxplot
        sns.boxplot(x=df[col], ax=axes[1], color='orange')
        axes[1].set_title(f"Boxplot of {col}")

        plt.tight_layout()
        plt.show()
```

```
# Identify categorical and numerical columns
cat_cols = df.select_dtypes(include=['object']).columns
num_cols = df.select_dtypes(include=['float64', 'int64'])

# Call the functions
plot_categorical_univariate(df, cat_cols)
plot_numerical_univariate(df, num_cols)
```

Show hidden output

**Customer Service Calls**

The histogram shows that most customers made between 0 and 3 calls to the customer service center, with a peak at 1 call. The boxplot indicates a median of around 2 calls, but there are noticeable outliers where customers made more than 5 calls.

`Interpretation:` A high number of service calls often reflects unresolved issues or dissatisfaction. These customers are more likely to churn, making this variable an important predictor.

**Total Charge**

The histogram suggests that the distribution of total charges is approximately normal, centered around 60.The boxplot shows that most customers are concentrated between 50 and 70, with a few outliers at very low or very high charges.

`Interpretation:` Customers with unusually high charges may perceive the service as too costly and become more prone to switching providers. Conversely, customers with very low charges might represent new or low-usage customers.

**General Conclusion**

From this exploratory analysis, it appears that both service-related variables (e.g., number of customer service calls) and billing-related variables (e.g., total charges) have a strong influence on churn behavior.

- `High service calls suggest dissatisfaction → higher churn risk.`
- `High total charges may discourage customers → higher churn risk.`
- `Outliers in both variables represent specific customer segments that require close monitoring.`

These findings highlight that churn is largely driven by a mix of customer experience and cost perception.

## 3.3 Bivariate Analysis of Customer Behavior and Churn

```
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Customer Service Calls vs Churn (Countplot)
plt.figure(figsize=(8,5))
sns.countplot(x='customer service calls', hue='churn', data=df, palette="Set2")
plt.title("Customer Service Calls vs Churn", fontsize=14, fontweight="bold")
plt.xlabel("Number of Customer Service Calls")
plt.ylabel("Count")
plt.legend(title="Churn", labels=["No", "Yes"])
plt.show()

# 2. Churn Rate by Customer Service Calls (Normalized lineplot)
churn_rate_calls = df.groupby('customer service calls')['churn'].mean()
plt.figure(figsize=(8,5))
sns.lineplot(x=churn_rate_calls.index, y=churn_rate_calls.values, marker='o')
plt.title("Churn Rate by Customer Service Calls", fontsize=14, fontweight="bold")
plt.xlabel("Number of Customer Service Calls")
plt.ylabel("Churn Rate")
plt.show()

# 3. Total Charge vs Churn (Boxplot)
plt.figure(figsize=(8,5))
sns.boxplot(x='churn', y='total_charge', data=df, hue='churn', palette="Set3", legend=False)
plt.title("Total Charge Distribution by Churn (Boxplot)", fontsize=14, fontweight="bold")
plt.xlabel("Churn")
plt.ylabel("Total Charge")
plt.show()

# 4. Total Charge vs Churn (Violin Plot)
plt.figure(figsize=(8,5))
sns.violinplot(x='churn', y='total_charge', data=df, hue='churn', palette="Pastel1", inner="quartile", legend=Fa
plt.title("Total Charge Distribution by Churn (Violin Plot)", fontsize=14, fontweight="bold")
plt.xlabel("Churn")
plt.ylabel("Total Charge")
plt.show()
```

Show hidden output

**Customer Service Calls vs Churn**
The majority of customers contact customer service only 0 to 2 times. However, customers who called 4 times or more show a significantly higher probability of churning. This suggests that frustration or dissatisfaction with customer service is a major driver of churn.

`Business Insight:` Monitor customers who frequently contact support and intervene quickly to improve their experience.

**Total Charge Distribution by Churn**
Customers who churn (True) generally have higher total charges compared to non-churners. Their median and quartile

values are consistently higher. This indicates that high-usage (and therefore high-billing) customers are more likely to leave, likely due to the perception of excessive costs.

`Business Insight:` Consider offering discounts, tailored packages, or loyalty programs for high-billing customers to mitigate churn risk.

**Conclusion before Modeling**

- `Two key variables strongly influence churn:`

*Customer service calls → dissatisfaction and customer experience.*

*Total charges → perception of high cost.*

These findings suggest that churn is primarily linked to customer dissatisfaction (frequent service calls) and pricing concerns (high charges). These variables should be closely monitored and integrated as priority features in the machine learning model.

## 4- Modeling

The goal of the modeling phase is to predict which customers are likely to churn using the data we prepared. We will apply classification techniques to learn patterns from customer behavior, service usage, and account features.

Our approach is iterative: we start with a simple, interpretable model to establish a baseline, and then explore more complex models to improve predictive performance. Each model will be evaluated using appropriate metrics to ensure it provides meaningful insights for the business.

This process allows us not only to predict churn but also to understand the factors that contribute most to customer retention and departure.

```python
# Encode categorical features
# -------------------------
df['international plan'] = df['international plan'].map({'yes':1, 'no':0})
df['voice mail plan'] = df['voice mail plan'].map({'yes':1, 'no':0})
df['churn'] = df['churn'].astype(int)

# Features and target
X = df.drop('churn', axis=1)
y = df['churn']

# -------------------------
# Step 1: Train (60%) and Temp (40%)
# -------------------------
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.4, random_state=42, stratify=y
)

# -------------------------
# Step 2: Validation (20%) and Test (20%) from Temp
# -------------------------
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp
)

# Check sizes
print(f"Train size: {X_train.shape[0]}")
print(f"Validation size: {X_val.shape[0]}")
print(f"Test size: {X_test.shape[0]}")
```

```
Train size: 1999
Validation size: 667
```

```
Test size: 667
```

```
# Step 3: Apply SMOTE ONLY on the training set
# -------------------------
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)

print("Before SMOTE:", y_train.value_counts())
print("After SMOTE:", y_train_res.value_counts())

# -------------------------
# Step 4: Scale features
# -------------------------
scaler = StandardScaler()

# Fit on training set (resampled)
X_train_scaled = scaler.fit_transform(X_train_res)

# Transform validation and test
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

```
Before SMOTE: churn
0    1709
1     290
Name: count, dtype: int64
After SMOTE: churn
0    1709
1    1709
Name: count, dtype: int64
```

### Encode categorical variables

```
Convert international plan and voice mail plan to numeric (1/0) so models can process them.

Convert churn to integer.
```

### Train/Validation/Test Split (60/20/20)

```
Split dataset to train (60%), validation (20%), and test (20%).

stratify=y keeps the churn proportion consistent.
```

### Apply SMOTE on training only

```
Balances the minority class (churn) in the training set.

Validation and test sets remain untouched to avoid data leakage.
```

### Feature Scaling

```
Standardize numerical features (mean=0, std=1).

Fit scaler on training set, transform validation and test sets.
```

`Key point:`
This pipeline prepares the data for modeling while ensuring balanced training, realistic validation/test evaluation, and consistent feature scales.

## 4.1 Evaluation Function

We define a reusable function `evaluate()` to compute key metrics and plot the confusion matrix for any model.
This allows us to evaluate **F1-score, Precision, Recall, ROC-AUC**, and visualize performance easily for **train, validation, and test sets**.
It works for Logistic Regression, Decision Tree, Random Forest, and any model supporting `predict` (and optionally `predict_proba` for ROC-AUC).

```
def evaluate(model, X, y, dataset_name="Dataset"):
    """
    Evaluate a model on given data and plot metrics and confusion matrix.
    """
    y_pred = model.predict(X)
    y_prob = model.predict_proba(X)[:, 1] if hasattr(model, "predict_proba") else None

    # Metrics
    acc = accuracy_score(y, y_pred)
    f1 = f1_score(y, y_pred, pos_label=1)
    precision = precision_score(y, y_pred, pos_label=1)
    recall = recall_score(y, y_pred, pos_label=1)
    roc_auc = roc_auc_score(y, y_prob) if y_prob is not None else "N/A"

    print(f"--- {dataset_name} Metrics ---")
    print(f"Accuracy: {acc:.2f}")
    print(f"F1-score (churn): {f1:.2f}")
    print(f"Precision (churn): {precision:.2f}")
    print(f"Recall (churn): {recall:.2f}")
    print(f"ROC-AUC: {roc_auc}")
    print("\nClassification Report:")
    print(classification_report(y, y_pred))

    # Confusion Matrix
    cm = confusion_matrix(y, y_pred)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix ({dataset_name})')
    plt.show()
```

## 4.2 Logistic Regression

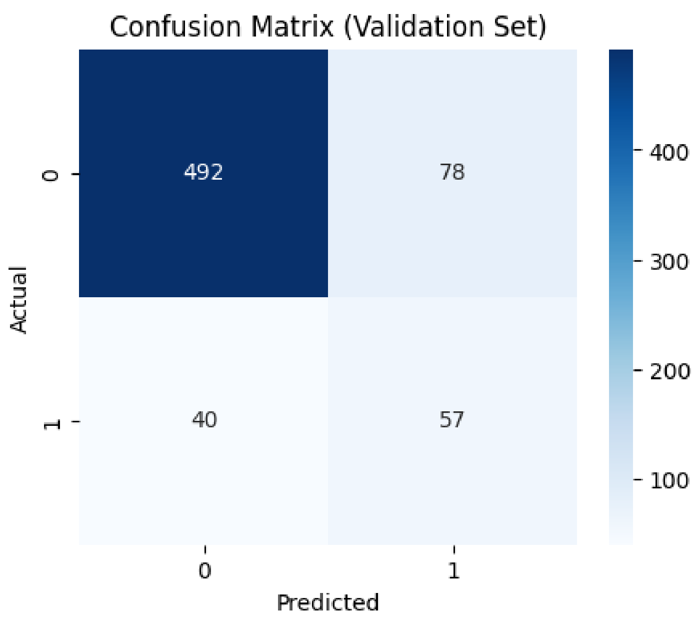
```
# Initialize model
log_reg = LogisticRegression(max_iter=1000, random_state=42)

# Define hyperparameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']  # needed to support l1
}

# Grid search with 5-fold cross-validation
grid_search = GridSearchCV(estimator=log_reg,
                           param_grid=param_grid,
                           scoring='f1',   # focus on F1-score for churn
                           cv=5,
                           n_jobs=-1,
                           verbose=1)

# Fit on scaled training set (after SMOTE)
grid_search.fit(X_train_scaled, y_train_res)
```

```
# Best hyperparameters and score
print("Best hyperparameters:", grid_search.best_params_)
print("Best F1-score (CV):", grid_search.best_score_)
```

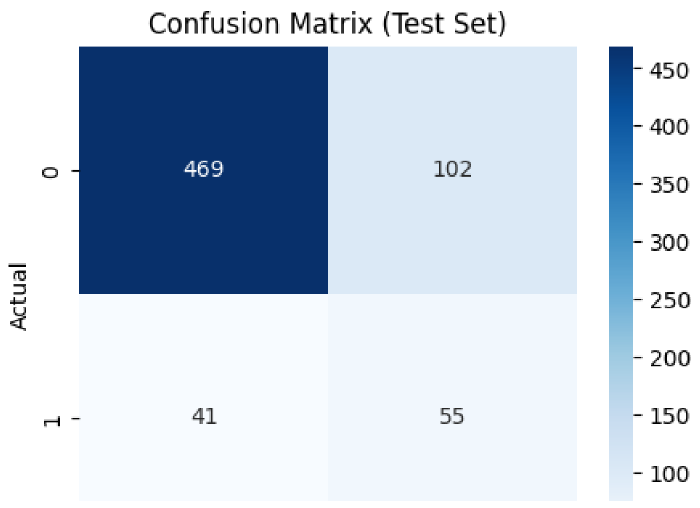
```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best hyperparameters: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
Best F1-score (CV): 0.8072424967136176
```

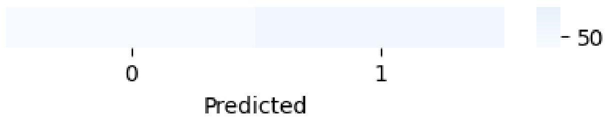
```
# Get the best model
best_log_reg = grid_search.best_estimator_

# Evaluate on validation and test sets
evaluate(best_log_reg, X_val_scaled, y_val, dataset_name="Validation Set")
evaluate(best_log_reg, X_test_scaled, y_test, dataset_name="Test Set")
```

```
--- Validation Set Metrics ---
Accuracy: 0.82
F1-score (churn): 0.49
Precision (churn): 0.42
Recall (churn): 0.59
ROC-AUC: 0.795496473141617

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.86      0.89       570
           1       0.42      0.59      0.49        97

    accuracy                           0.82       667
   macro avg       0.67      0.73      0.69       667
weighted avg       0.85      0.82      0.83       667
```



```
--- Test Set Metrics ---
Accuracy: 0.79
F1-score (churn): 0.43
Precision (churn): 0.35
Recall (churn): 0.57
ROC-AUC: 0.8075379451255108

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.82      0.87       571
           1       0.35      0.57      0.43        96

    accuracy                           0.79       667
   macro avg       0.63      0.70      0.65       667
weighted avg       0.84      0.79      0.81       667
```





**Classification Report Insights (Logistic Regression, F1-focused)**

`F1-score (overall focus)` = 0.43–0.49 (churn class)

**Class 0 (Not Churned):**

`Precision` = 0.92 → Most customers predicted as non-churn actually did not churn.

`Recall` = 0.82–0.86 → Majority of actual non-churn customers were correctly identified.

**Class 1 (Churned):**

`Precision` = 0.35–0.42 → Many predicted churns are false positives.

`Recall` = 0.57–0.59 → Over half of actual churned customers were correctly identified.

**ROC-AUC Score = 0.80–0.81**

Indicates a good ability to distinguish between churned and non-churned customers.

**Accuracy Results**

`Validation`: 82%
`Test`: 79%
The model demonstrates strong performance, with a slight decrease in accuracy on the test set, showing that it generalizes well to unseen data.

**Confusion Matrix Insights:**

`Validation:` 492/570 non-churn correctly predicted; 57/97 churn correctly predicted.

`Test:` 469/571 non-churn correctly predicted; 55/96 churn correctly predicted.

**Insight:**

The model detects most churners reasonably well (recall ~0.57–0.59), which is critical for a telecom company.

*Many false positives (low precision) → could trigger unnecessary retention actions.*

Using F1-score highlights the trade-off between catching churners and limiting false alarms.

Further tuning or more complex models (Decision Tree, Random Forest) could improve performance, especially for churn class precision.

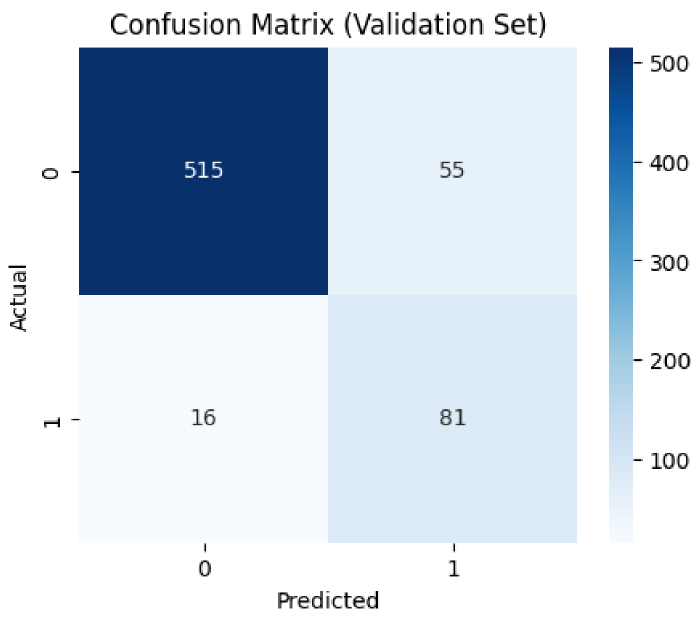
## 4.3 Decision Tree

```
# Convert scaled arrays to DataFrames to keep column names
# ----------------------------
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train_res.columns)
X_val_scaled_df   = pd.DataFrame(X_val_scaled, columns=X_train_res.columns)
X_test_scaled_df  = pd.DataFrame(X_test_scaled, columns=X_train_res.columns)

# ----------------------------
# Initialize Decision Tree
# ----------------------------
dtree = DecisionTreeClassifier(random_state=42)

# ----------------------------
# Define hyperparameter grid
# ----------------------------
param_grid = {
    'max_depth': [3, 5, 7, 9, None],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 5, 10],
    'criterion': ['gini', 'entropy']
}

# ----------------------------
# GridSearchCV for tuning
# ----------------------------
grid_search_dt = GridSearchCV(
    estimator=dtree,
    param_grid=param_grid,
    scoring='f1',           # Focus on F1-score for churn
    cv=5,
    n_jobs=-1,
    verbose=1
)

# Fit on scaled training set
grid_search_dt.fit(X_train_scaled_df, y_train_res)

# ----------------------------
# Best parameters and score
# ----------------------------
print("Best hyperparameters:", grid_search_dt.best_params_)
print("Best F1-score (CV):", grid_search_dt.best_score_)
```

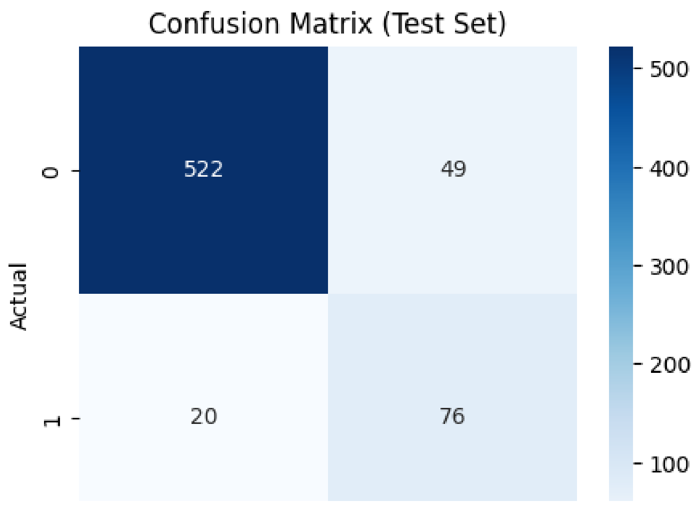
```
Fitting 5 folds for each of 160 candidates, totalling 800 fits
Best hyperparameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split'
Best F1-score (CV): 0.9036368833853524
```

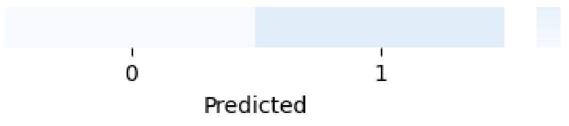
```
    # Evaluate on validation and test sets
    # ----------------------------
    best_dtree = grid_search_dt.best_estimator_

    evaluate(best_dtree, X_val_scaled_df, y_val, dataset_name="Validation Set")
    evaluate(best_dtree, X_test_scaled_df, y_test, dataset_name="Test Set")
```

```
--- Validation Set Metrics ---
Accuracy: 0.89
F1-score (churn): 0.70
Precision (churn): 0.60
Recall (churn): 0.84
ROC-AUC: 0.8692801591607887

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.90      0.94       570
           1       0.60      0.84      0.70        97

    accuracy                           0.89       667
   macro avg       0.78      0.87      0.82       667
weighted avg       0.92      0.89      0.90       667
```



```
--- Test Set Metrics ---
Accuracy: 0.90
F1-score (churn): 0.69
Precision (churn): 0.61
Recall (churn): 0.79
ROC-AUC: 0.8529261529480444

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.91      0.94       571
           1       0.61      0.79      0.69        96

    accuracy                           0.90       667
   macro avg       0.79      0.85      0.81       667
weighted avg       0.91      0.90      0.90       667
```





The model effectively detects the majority of churners (**Recall** ~0.79–0.84).

**Precision** is reasonable (≈0.60) → fewer false positives compared to Logistic Regression.

**F1-score** is high (~0.69–0.70) → indicates a good balance between Precision and Recall for churn detection.

**ROC-AUC** > 0.85 → the model efficiently distinguishes between churn and non-churn.

**Accuracy Results – Decision Tree**

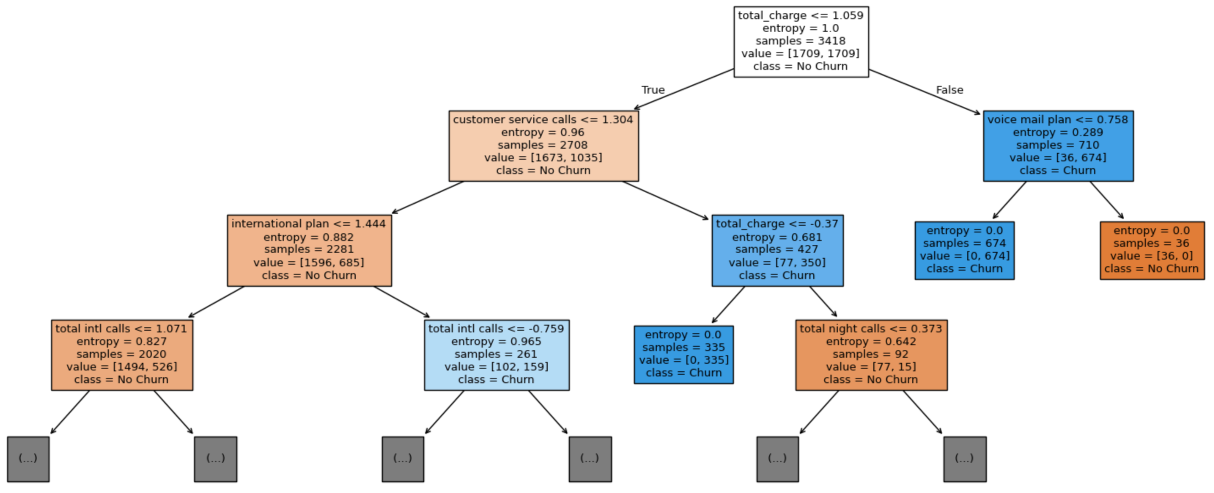
$\text{Validation}: 89\%$

$\text{Test}: 90\%$

The Decision Tree model shows excellent performance, with consistent accuracy between the validation and test sets, indicating good generalization and low risk of overfitting.

In conclusion, the Decision Tree outperforms Logistic Regression in detecting churners and serves as a strong baseline before exploring Random Forest models.

```
plt.figure(figsize=(20,8))
plot_tree(best_dtree, feature_names=X_train_res.columns, class_names=['No Churn','Churn'], filled=True, max_depth
plt.show()
```



## 4.4 Random Forest

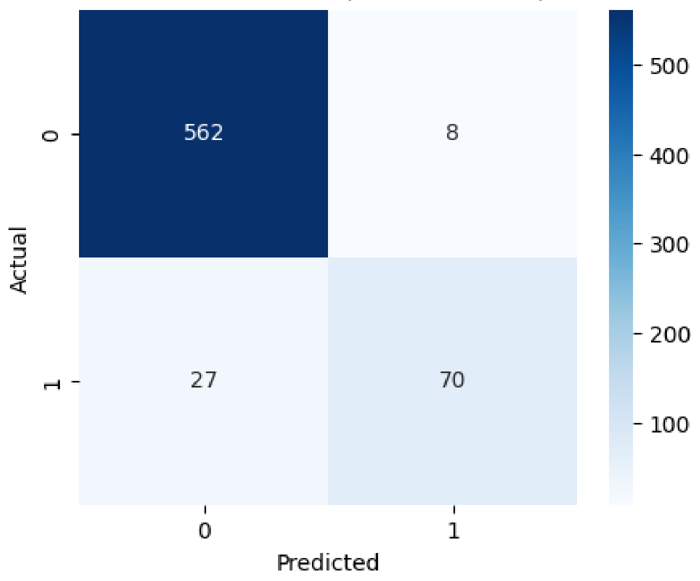
```
#  Initialize Random Forest
# ----------------------------
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

# ----------------------------
# Define hyperparameter grid for RandomizedSearchCV
# ----------------------------
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 7, 9, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

# -----------------------------
# RandomizedSearchCV for tuning
# -----------------------------
rand_search_rf = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_grid,
    n_iter=30,              # explore 30 random combinations
    scoring='f1',           # focus on churn F1-score
    cv=5,
    n_jobs=-1,
    verbose=1,
    random_state=42
)

# Fit on training set after SMOTE
rand_search_rf.fit(X_train_scaled_df, y_train_res)

# -----------------------------
# Best parameters and score
# -----------------------------
print("Best hyperparameters:", rand_search_rf.best_params_)
print("Best F1-score (CV):", rand_search_rf.best_score_)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Best hyperparameters: {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': None
Best F1-score (CV): 0.9598987972626615
```

```
# -----------------------------
# Evaluate on validation and test sets
# -----------------------------
best_rf = rand_search_rf.best_estimator_

evaluate(best_rf, X_val_scaled_df, y_val, dataset_name="Validation Set")
evaluate(best_rf, X_test_scaled_df, y_test, dataset_name="Test Set")
```

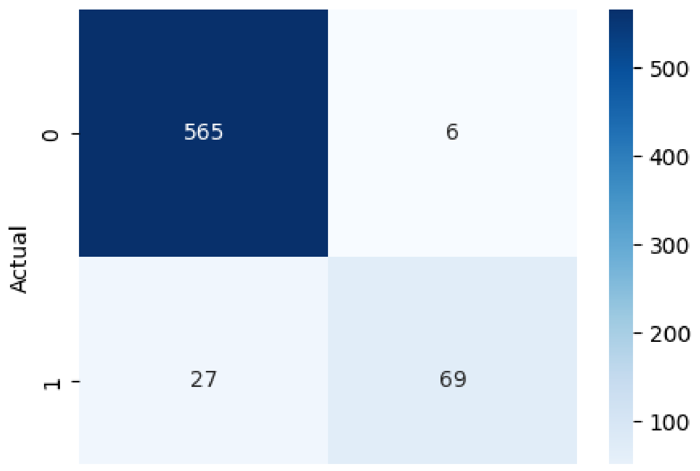
```
--- Validation Set Metrics ---
Accuracy: 0.95
F1-score (churn): 0.80
Precision (churn): 0.90
Recall (churn): 0.72
ROC-AUC: 0.9339301862904685

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       570
           1       0.90      0.72      0.80        97

    accuracy                           0.95       667
   macro avg       0.93      0.85      0.88       667
weighted avg       0.95      0.95      0.95       667
```

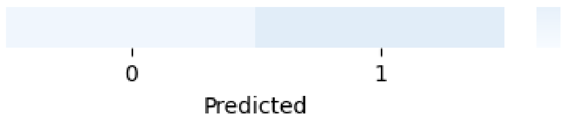
Confusion Matrix (Validation Set)



```
--- Test Set Metrics ---
Accuracy: 0.95
F1-score (churn): 0.81
Precision (churn): 0.92
Recall (churn): 0.72
ROC-AUC: 0.8666356538237011

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       571
           1       0.92      0.72      0.81        96

    accuracy                           0.95       667
   macro avg       0.94      0.85      0.89       667
weighted avg       0.95      0.95      0.95       667
```

Confusion Matrix (Test Set)





The model successfully detects the majority of churners (**Recall** ~0.72).

**Precision** is very high (~0.90–0.92) → very few false positives.

**F1-score** is high (~0.80–0.81) → excellent balance between Precision and Recall for churn detection.

**ROC-AUC** is high → the model effectively distinguishes between churn and non-churn.

**Accuracy Results – Random Forest**

`Validation`: 95%

`Test`: 95%

The Random Forest model achieves very high accuracy on both validation and test sets, showing excellent predictive power and strong generalization.

`Conclusion:` Random Forest outperforms both Logistic Regression and Decision Tree, particularly in terms of Precision, Accuracy and F1-score for churners.

## 4.5 Top Features Driving Customer Churn Prediction

```
# Best Random Forest from RandomizedSearchCV
best_rf = rand_search_rf.best_estimator_

# Get feature importances
importances = best_rf.feature_importances_
features = X_train_scaled_df.columns  # Make sure this is the DataFrame used for training
```