# Business Understanding

## Project Goal

**The primary objective of this project is to develop a predictive model that can identify customers at high risk of churn within SyriaTel.**

## Business Context

**Customer churn, the rate at which customers discontinue their service, is a critical concern for telecommunications companies like SyriaTel. High churn rates can lead to significant revenue loss and increased costs associated with acquiring new customers.**

## Value of Churn Prediction

**By accurately predicting which customers are likely to churn, SyriaTel can proactively implement targeted retention strategies. These strategies may include:**

- Offering personalized incentives or discounts
- Improving customer service and support
- Addressing specific pain points or concerns

**Ultimately, effective churn prediction enables SyriaTel to improve customer satisfaction, reduce revenue loss, and optimize customer retention efforts, leading to increased profitability.**

## Data Overview

**The analysis will be based on a dataset containing customer information and service usage details. Key features include:**

- Account information (e.g., account length)
- Service usage (e.g., call durations, data usage, charges)
- Customer demographics (e.g., state, area code)
- Plan details (e.g., international plan, voice mail plan)
- Customer service interactions (e.g., number of customer service calls)

**The goal is to leverage this data to identify patterns and build a model that accurately predicts which customers are most likely to churn.**

In [1]:

```
# Importing necessary Libraries and Packages
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [2]:

```
# Loading the data into a dataframe
```

```
df = pd.read_csv('SyriaTelChurn.csv')
df.head()
```

Out[2]:

| | 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 | total night minutes | total night calls | ch |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | KS | 128 | 415 | 382-4657 | no | yes | 25 | 265.1 | 110 | 45.07 | ... | 99 | 16.78 | 244.7 | 91 | |
| 1 | OH | 107 | 415 | 371-7191 | no | yes | 26 | 161.6 | 123 | 27.47 | ... | 103 | 16.62 | 254.4 | 103 | |
| 2 | NJ | 137 | 415 | 358-1921 | no | no | 0 | 243.4 | 114 | 41.38 | ... | 110 | 10.30 | 162.6 | 104 | |
| 3 | OH | 84 | 408 | 375-9999 | yes | no | 0 | 299.4 | 71 | 50.90 | ... | 88 | 5.26 | 196.9 | 89 | |
| 4 | OK | 75 | 415 | 330-6626 | yes | no | 0 | 166.7 | 113 | 28.34 | ... | 122 | 12.61 | 186.9 | 121 | |

**5 rows × 21 columns**

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

- **Dataset Size: There are 3333 rows (customers) and 21 columns (features).**
- **Data Types: We see the data types of each column.**
  - **We have object (for strings/categorical data)**
  - **Int64 (integers)**
  - **Float64 (floating-point numbers)**
  - **Bool for the churn target variable.**
- **Missing Values: The "Non-Null Count" is 3333 for all columns, which is great news!**
- **It means there are no missing values in this dataset simplifYING data preparation step.**

In [4]:

```
# checking the values of churn data
df['churn'].value_counts()
```

Out[4]:

```
False    2850
True      483
Name: churn, dtype: int64
```

- **This indicates there are have more non-churning customers than churning ones**

In [5]:

```
# Creating a copy of the data to avoid modifying the original DataFrame
df_processed = df.copy()
```

In [6]:

```
# Identifying Categorical columns (excluding the target variable 'churn')
categorical_cols = ['state', 'area code', 'international plan', 'voice mail plan']
```

In [7]:

```
# Performing one-hot encoding
df_processed = pd.get_dummies(df_processed, columns=categorical_cols, drop_first=True)
```

In [8]:

```
# Droping the phone number column
df_processed = df_processed.drop('phone number', axis=1)
```

In [9]:

```
# Displaying the first few rows of processed data and its shape
print("Shape of the processed DataFrame:", df_processed.shape)
print("\n First 5 rows of the processed DataFrame:\n", df_processed.head())
```

```
Shape of the processed DataFrame: (3333, 70)

 First 5 rows of the processed DataFrame:
    account length  number vmail messages  total day minutes  total day calls  \
0             128                     25              265.1              110
1             107                     26              161.6              123
2             137                      0              243.4              114
3              84                      0              299.4               71
4              75                      0              166.7              113

   total day charge  total eve minutes  total eve calls  total eve charge  \
0             45.07              197.4               99             16.78
1             27.47              195.5              103             16.62
2             41.38              121.2              110             10.30
3             50.90               61.9               88              5.26
4             28.34              148.3              122             12.61

   total night minutes  total night calls  ...  state_VA  state_VT  state_WA  \
0                244.7                 91  ...         0         0         0
1                254.4                103  ...         0         0         0
2                162.6                104  ...         0         0         0
3                196.9                 89  ...         0         0         0
4                186.9                121  ...         0         0         0

   state_WI  state_WV  state_WY  area code_415  area code_510  \
0         0         0         0              1              0
1         0         0         0              1              0
2         0         0         0              1              0
3         0         0         0              0              0
4         0         0         0              1              0

   international plan_yes  voice mail plan_yes
0                       0                    1
1                       0                    1
```

```
2                       0                       0
3                       1                       0
4                       1                       0

[5 rows x 70 columns]
```

- **One hot encoding has been applied and the phone number has been dropped.**
- **I started with 21 columns now i have 70, the increase is due to the expansion of categorical features into multiple binary columns.**

In [10]:

```
# Splitting data into training and testing with a final hold out set with 70-30 ratios wi
th a random_state=42
# Separating features X and target y
X = df_processed.drop('churn', axis=1)
y = df_processed['churn'].astype(int) # Convert boolean to integers (0 and 1)


# Splitting data into training and temporary set (85% train, 15% temp)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42
, stratify=y)


# Splitting the temporary set into testing and holdout (50% test, 50% holdout of the temp
of the temp set, which is ~15% of the original)
X_test, X_holdout, y_test, y_holdout = train_test_split(X_temp, y_temp, test_size=0.5, r
andom_state=42, stratify=y_temp)


# Printing the shapes of the resulting sets to verify the splits
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of X_holdout:", X_holdout.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
print("Shape of y_holdout:", y_holdout.shape)
```

```
Shape of X_train: (2333, 69)
Shape of X_test: (500, 69)
Shape of X_holdout: (500, 69)
Shape of y_train: (2333,)
Shape of y_test: (500,)
Shape of y_holdout: (500,)
```

**The shapes of my training, testing, and holdout sets look correct based on our intended split. I now have:**

- **Training set: 2333 samples (approximately 70% of the original data) to train our models.**
- **Testing set: 500 samples (approximately 15% of the original data) to evaluate and compare our models during the development process.**
- **Holdout set: 500 samples (approximately 15% of the original data) to provide a final, unbiased evaluation of our best model.**

# Feature Scaling

In [11]:

```
# Identifying numerical columns
numerical_cols = X_train.select_dtypes(include=['int64', 'float64']).columns

# Initialize StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data
scaler.fit(X_train[numerical_cols])
```

```
# Transforming the training, testing and holdout sets
X_train[numerical_cols] = scaler.transform(X_train[numerical_cols])
X_test[numerical_cols] = scaler.transform(X_test[numerical_cols])
X_holdout[numerical_cols] = scaler.transform(X_holdout[numerical_cols])

# Display the first few rows of the scaled training data
print("First 5 rows of scaled X_train:\n", X_train.head())
```

```
First 5 rows of scaled X_train:
       account length  number vmail messages  total day minutes  \
606          0.710730              -0.601437          -0.745204
2468         1.285368               2.388494          -0.609853
1844        -0.588453               1.440467          -0.811951
3187        -1.063155               2.242644           0.597172
3083         1.835023              -0.601437           1.034742

      total day calls  total day charge  total eve minutes  total eve calls  \
606          0.934611         -0.744719           1.720633         0.235871
2468         1.383797         -0.609478           1.648204        -0.212733
1844         0.834792         -0.812340           0.843651         0.086337
3187        -3.008242          0.596788           1.428958        -1.209630
3083         0.385606          1.035231          -1.156963         2.129976

      total eve charge  total night minutes  total night calls  ...  state_VA  \
606           1.720493            -1.426410          -0.156352  ...         0
2468          1.649100             0.263000          -1.027532  ...         0
1844          0.843041             0.140608          -1.335008  ...         0
3187          1.428009             0.200801           1.995977  ...         0
3083         -1.155986             2.331624           1.586009  ...         0

      state_VT  state_WA  state_WI  state_WV  state_WY  area code_415  \
606          0         0         0         0         0              1
2468         0         0         0         1         0              0
1844         0         0         0         0         0              0
3187         0         0         0         1         0              0
3083         0         0         0         0         0              0

      area code_510  international plan_yes  voice mail plan_yes
606               0                       0                    0
2468              1                       1                    1
1844              1                       0                    1
3187              0                       0                    1
3083              1                       0                    0

[5 rows x 69 columns]
```

- **The output shows that the numerical features in X_train DataFrame have been scaled with values of both positive and negative.**

# Building Classification Models

## Logistic Regression

In [12]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Initialize the Logistic Regression model
# I will use small value of C(inverse of regularization strenght) for simplicity
logreg_model = LogisticRegression(random_state=42, solver='liblinear', C=0.1)

# Train the model on the training data
logreg_model.fit(X_train, y_train)

# Make prediction on testing data
```

```
y_pred_logreg = logreg_model.predict(X_test)


# Evaluating the model
print("Baseline Logistic Regression Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred_logreg))
print("\nClassification_report:\n", classification_report(y_test, y_pred_logreg))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_logreg))
```

```
Baseline Logistic Regression Model Evaluation:
Accuracy: 0.868

Classification_report:
              precision    recall  f1-score   support

           0       0.88      0.99      0.93       427
           1       0.68      0.18      0.28        73

    accuracy                           0.87       500
   macro avg       0.78      0.58      0.60       500
weighted avg       0.85      0.87      0.83       500


Confusion Matrix:
 [[421   6]
 [ 60  13]]
```

### Analysis of this baseline model

**The model has a high overall accuracy (86.8%) which shows that the model is much better at predicting non-churn than churn.**

- **The recall for the churn class (class 1) is very low (0.18). This means the model is missing a large proportion of the customers who actually churn. In a business context, this is a problem because we want to identify as many potential churners as possible so we can try to retain them.**
- **The precision for the churn class (0.68) is better than the recall, indicating that when the model predicts churn, it's correct a decent amount of the time. However, because the recall is so low, it's not identifying many of the actual churners.**

# Next Steps

- **Focus on the Churn Class: Since my primary goal is to predict churn, i need to improve the model's ability to correctly identify churning customers (increase recall for class 1).**
- **Hyperparameter Tuning: I used a default or simple setting for my Logistic Regression model. Tuning the hyperparameters (like the regularization strength C) might improve performance.**
- **Try a Different Model: Decision Tree can be interpretable and might perform differently.**

In [13]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, recall_score

# Defining the parameter grid to search
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

# Create a scorer for recall of the positive class (churn-class 1)
recall_scorer = make_scorer(recall_score, pos_label=1)

# Initialize the logistic Regression Model
logreg = LogisticRegression(random_state=42, solver='liblinear', penalty='l2')

# Perform GridSearchCV
grid_search = GridSearchCV(logreg, param_grid, scoring=recall_scorer, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)
```

```
# Print the best parameters and the best recall score
print("Best hyperparameters for Logistic Regression:", grid_search.best_params_)
print("Best recall score on training data:", grid_search.best_score_)

# Get the best model on the test set
best_logreg = grid_search.best_estimator_

# Evaluate the best model on the test set
y_pred_best_logreg = best_logreg.predict(X_test)

print("\nEvaluation of the best Logistic Regression Model on the test set:")
print("Accuracy:", accuracy_score(y_test, y_pred_best_logreg))
print("\nClassification Report:", classification_report(y_test, y_pred_best_logreg))
print("\nConfusion Matrix:", confusion_matrix(y_test, y_pred_best_logreg))
```

```
Best hyperparameters for Logistic Regression: {'C': 100}
Best recall score on training data: 0.2573748902546093

Evaluation of the best Logistic Regression Model on the test set:
Accuracy: 0.868

Classification Report:               precision    recall  f1-score   support

           0       0.89      0.97      0.93       427
           1       0.61      0.27      0.38        73

    accuracy                           0.87       500
   macro avg       0.75      0.62      0.65       500
weighted avg       0.85      0.87      0.85       500


Confusion Matrix: [[414  13]
 [ 53  20]]
```

### Analysis of Tuned Logistic Regression

- **By tuning the C hyperparameter with the goal of maximizing recall for the churn class, the model has achieved a better ability to identify actual churners. The recall for class 1 increased from 0.18 to 0.27, meaning the model now correctly identifying a larger proportion of customers who will churn.**

**However, this improvement in recall has come at a slight cost:**

```
- The precision for the churn class has decreased slightly (from 0.68 to 0.61), mea
ning that when the model predicts churn, it is now slightly less likely to be corre
ct.
- The number of false positives (predicting churn when the customer doesn't) has in
creased.
```

- **The overall accuracy remains the same, which again highlights that accuracy isn't the best metric to focus on in imbalanced datasets.**

# Decision Tree

In [14]:

```
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree Classifier
tree_model = DecisionTreeClassifier(random_state=42)

#Train the model on the training data
tree_model.fit(X_train, y_train)

#Make predictions on the testing data

y_pred_tree = tree_model.predict(X_test)
```

```
# Evaluate the model
print("\nDecision Tree Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred_tree))
print("\nClassification Report:", classification_report(y_test, y_pred_tree))
print("\nConfusion Matrix:", confusion_matrix(y_test, y_pred_tree))
```

```
Decision Tree Model Evaluation:
Accuracy: 0.92

Classification Report:                precision    recall  f1-score   support

           0       0.94      0.96      0.95       427
           1       0.76      0.66      0.71        73

    accuracy                           0.92       500
   macro avg       0.85      0.81      0.83       500
weighted avg       0.92      0.92      0.92       500


Confusion Matrix: [[412  15]
 [ 25  48]]
```

## Analysis of Decision Tree model

- **The Decision Tree model appears to be significantly outperforming both the baseline and the tuned Logistic Regression models, especially in its ability to predict customer churn (higher recall, precision, and F1-score for class 1). It has a much better balance between correctly identifying churners and minimizing false negatives (missing actual churners).**
- **While the Decision Tree has a slightly higher number of false positives compared to the baseline Logistic Regression, the substantial improvement in correctly identifying churners (fewer false negatives) is likely more valuable for SyriaTel. Missing fewer potential churners allows for more opportunities to intervene and retain them.**

## Next Steps

- **Hyperparameter Tuning for the Decision Tree**
- **Feature Importance**
- **Model Interpretation for the Presentation for non -technical**

## Hyperparameter tuning for the Decision Tree

In [15]:

```
# Define the parameter grid for decision Tree
param_grid_tree = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 3, 5]
}

# Creating a scorer for recall of the positive class (churn - class 1)
recall_scorer = make_scorer(recall_score, pos_label=1)

# Initialize the Decision Tree Classifier
tree = DecisionTreeClassifier(random_state=42)

# Perform GridSearchCV
grid_search_tree = GridSearchCV(tree, param_grid_tree, scoring=recall_scorer, cv=5, n_jo
bs=-1)
grid_search_tree.fit(X_train, y_train)
```

```
# print the best hyperparameters and the best recall score
print("Best hyperparameters for Decision Tree:", grid_search_tree.best_params_)
print("Best recall score on training data:", grid_search_tree.best_score_)

# Get the best model for GridSearchCV
best_tree = grid_search_tree.best_estimator_

# Evaluate the best model on the test set
y_pred_best_tree = best_tree.predict(X_test)

print("\nEvaluation of the best Decision Tree model on the test set:")
print("Accuracy:", accuracy_score(y_test, y_pred_best_tree))
print("\nClassification Report:", classification_report(y_test, y_pred_best_tree))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_best_tree))
```

```
Best hyperparameters for Decision Tree: {'criterion': 'entropy', 'max_depth': None, 'min_
samples_leaf': 1, 'min_samples_split': 10}
Best recall score on training data: 0.7459174714661984

Evaluation of the best Decision Tree model on the test set:
Accuracy: 0.928

Classification Report:               precision    recall  f1-score   support

           0       0.95      0.97      0.96       427
           1       0.79      0.68      0.74        73

    accuracy                           0.93       500
   macro avg       0.87      0.83      0.85       500
weighted avg       0.92      0.93      0.93       500


Confusion Matrix:
 [[414  13]
 [ 23  50]]
```

### Analysis of Tuned Decision Tree

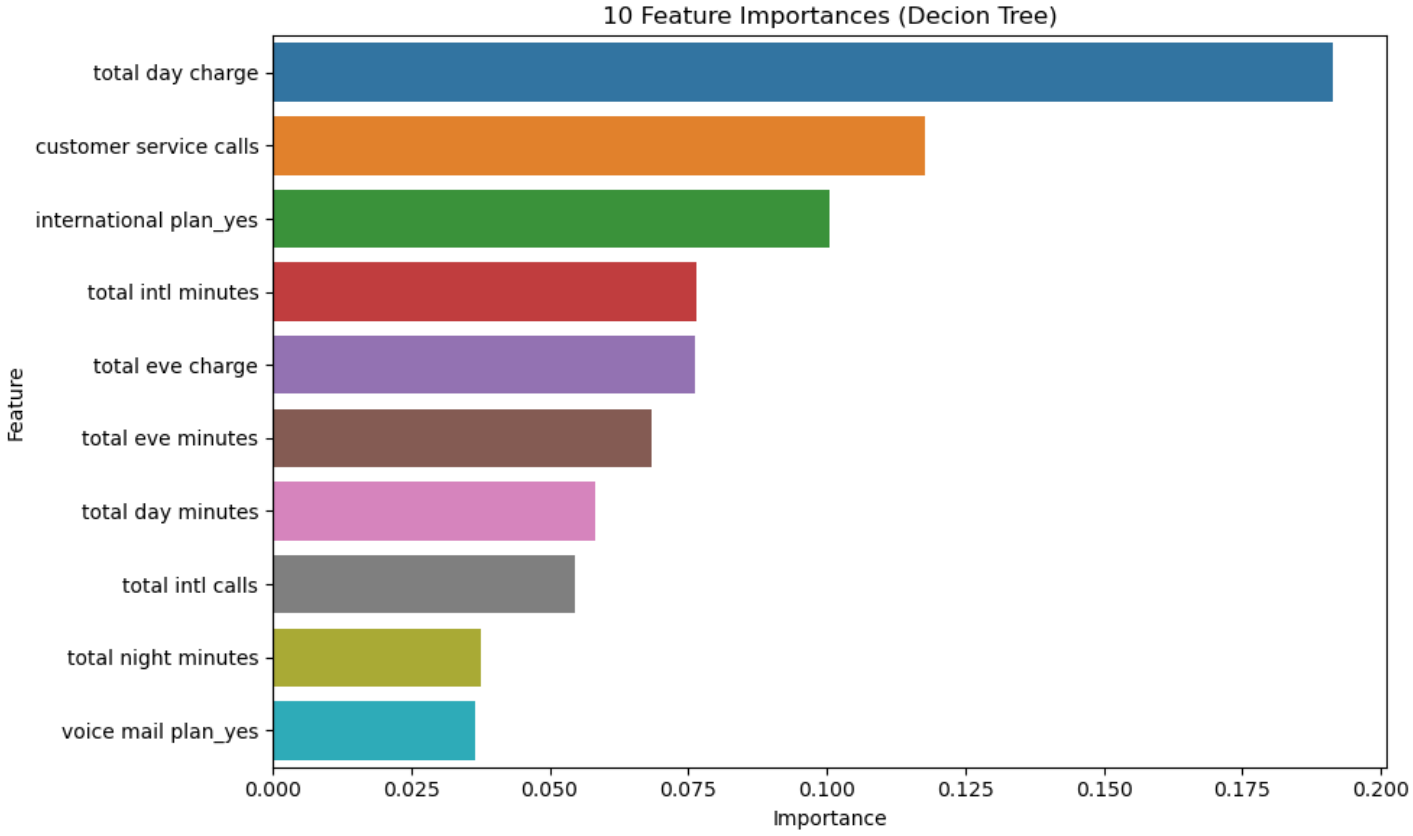
- **The hyperparameter tuning has indeed led to a slightly better performing Decision Tree model on the test set. We see improvements in accuracy, precision, recall, and F1-score for the churn class. The model is now even better at correctly identifying customers who will churn while also reducing the number of false positives and false negatives.**
- **Given these results, the tuned Decision Tree appears to be the best model so far.**

# Visualizing

In [16]:

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

# Get feature importance from best Decision Tree Model
feature_importances = best_tree.feature_importances_

# Get the names of the features
feature_names = X_train.columns

# Create a DataFrame to store feature importance
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances
})

importance_df_sorted = importance_df.sort_values(by='Importance', ascending=False)

# Plotting the top N important in descending order
top_n = 10
plt.figure(figsize=(10, 6))
```

```
sns.barplot(x='Importance', y='Feature', data=importance_df_sorted.head(top_n))
plt.title(f"{top_n} Feature Importances (Decion Tree)")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()


# Print the sorted features
print("\nSorted Feature Importances:")
print(importance_df_sorted)
```




```
Sorted Feature Importances:
                    Feature  Importance
4          total day charge    0.191460
14   customer service calls    0.117724
67   international plan_yes    0.100450
11       total intl minutes    0.076616
7          total eve charge    0.076298
..                      ...         ...
35                 state_ME    0.000000
36                 state_MI    0.000000
37                 state_MN    0.000000
38                 state_MO    0.000000
34                 state_MD    0.000000

[69 rows x 2 columns]
```

# Interpretetion

- **Analysis shows that the total amount a customer is charged during the day is the strongest predictor of whether they will leave. Customers with higher day charges are more likely to churn.**
- **The number of times a customer contacts customer service is also a critical indicator. Customers who call support more frequently are at a higher risk of churning. This suggests that addressing customer service issues promptly and effectively could be key to retention.**
- **Customers who have signed up for an international plan are also more likely to churn. SyriaTel might want to investigate the reasons behind this – perhaps related to pricing, service quality for international calls, or the needs of these customers.**
- **The amount of time customers spend on international calls and their evening call charges also play a role in predicting churn.**

### Areas to focus on

- **Analyze pricing structures, especially for day usage and international plans.**
- **Improve their customer service processes to reduce the number of calls and increase customer satisfaction.**
- **Consider targeted retention efforts for customers with international plans or those with high day/evening charges.**

### Evaluating the tuned Decision Tree on the holdout set

In [17]:

```
y_pred_holdout_tree = best_tree.predict(X_holdout)

print("\nEvaluation of the best Decision Tree model on the holdout set:")
print("Accuracy:", accuracy_score(y_holdout, y_pred_holdout_tree))
print("\nClassification Report:\n", classification_report(y_holdout, y_pred_holdout_tree)
)
print("\nConfusion Matrix:\n", confusion_matrix(y_holdout, y_pred_holdout_tree))
```

```
Evaluation of the best Decision Tree model on the holdout set:
Accuracy: 0.924

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.96      0.96       428
           1       0.74      0.72      0.73        72

    accuracy                           0.92       500
   macro avg       0.85      0.84      0.84       500
weighted avg       0.92      0.92      0.92       500


Confusion Matrix:
 [[410  18]
 [ 20  52]]
```

# Final Analysis

**The performance of our tuned Decision Tree model on the holdout set is consistent with its performance on the test set. This gives confidence that the model generalizes well to unseen data and is not overfitting.**

- **It achieves a high overall accuracy (92.4%).**
- **More importantly for our business goal, it has a good recall for the churn class (0.72), meaning it correctly identifies a significant portion of the customers who will actually churn.**
- **The precision for the churn class (0.74) is also respectable, indicating that when the model predicts churn, it is correct a good amount of the time.**