1 Predicting Customer Churn in the Telecommunications Industry: ¶

A Machine Learning Approach for Proactive Customer Retention

Project by: Kenneth Nyangweso

2 Business Understanding:

2.1 Project Overview:

Customer retention is a critical concern for telecommunications companies, as acquiring new customers is often more expensive than retaining existing ones. With increasing competition, understanding why customers leave (churn) and identifying those at risk can significantly improve business strategies. This project aims to build a binary classification model to predict customer churn based on various usage patterns and customer service interactions.

2.2 Problem Statement:

SyriaTel, like many telecommunication providers, faces challenges in retaining customers due to factors such service dissatisfaction, competitive offers,shifting customer needs, and many other factors. Identifying customers who are likely to churn before they actually leave allows the company to take proactive measures such as personalized offers, improved service quality, and better customer engagement. Without a predictive approach, customer retention efforts may be reactive and less effective, leading to revenue losses.

2.3 Main Objective:

- To develop machine learning models that accurately predicts customer churn in order to help SyriaTel reduce churn rates and improve customer satisfaction.

2.4 Specific Objectives:

1. Data Exploration & Cleaning – Understand the dataset structure, handle missing values, and preprocess data for analysis.
2. Feature Engineering & Selection – Identify key factors contributing to churn and optimize input variables for modeling.
3. Model Development – Train and evaluate multiple classification models to determine the best-performing one.
4. Interpretability & Insights – Analyze the key drivers of churn and provide business recommendations.

2.5 Why Now?

- **Increased Competition:** Telecommunications companies are experiencing higher competition, making customer retention more crucial than ever.

- **Data Availability:** SyriaTel has rich historical customer data that can be leveraged for predictive modeling.
- **Cost Efficiency:** Predicting churn allows for targeted interventions, reducing marketing costs while improving customer loyalty.
- **Advancements in AI & Machine Learning:** Modern machine learning techniques make churn prediction more accurate and actionable than traditional rule-based approaches.

2.6 Metrics of Success:

- **Accuracy & Precision:** Ensuring the model correctly classifies customers into churn and non-churn categories.
- **Recall (Sensitivity):** High recall ensures that most potential churners are identified.
- **F1-score:** Balancing precision and recall to optimize performance.
- **ROC-AUC Score:** Measuring the model's ability to distinguish between churn and non-churn customers.
- **Business Impact:** Reduction in churn rates and improved customer engagement based on model-driven interventions.

3 Data Understanding:

The Data Understanding phase involves exploring the dataset to gain insights into its structure, quality, and key attributes. This step helps in identifying patterns, missing values, and potential inconsistencies that could affect the modeling process.

3.1 Understanding the dataset:

The SyriaTel Customer Churn Dataset contains information about customers, their usage patterns, service subscriptions, and interactions with customer service. The dataset is structured with 3,333 records and 21 features, including the target variable (Churn), which indicates whether a customer has left the service or not.

3.1.1 Dataset description:

- State – The state where the customer resides.
- Account length – Duration (in months) the customer has been with SyriaTel.
- Area code – The customer's area code.
- Phone number – Unique identifier for each customer.
- International plan – Whether the customer has an international calling plan.
- Voice mail plan – Whether the customer has a voicemail plan.
- Number vmail messages – Number of voicemail messages stored by the customer.
- Total day minutes – Total minutes used during daytime calls.
- Total day calls – Total number of daytime calls made.
- Total day charge – Total charge for daytime calls.
- Total eve minutes – Total minutes used during evening calls.
- Total eve calls – Total number of evening calls made.
- Total eve charge – Total charge for evening calls.
- Total night minutes – Total minutes used during nighttime calls.
- Total night calls – Total number of nighttime calls made.

- Total night charge – Total charge for nighttime calls.
- Total intl minutes – Total minutes used for international calls.
- Total intl calls – Total number of international calls made.
- Total intl charge – Total charge for international calls.
- Customer service calls – Number of times the customer called customer service.
- Churn – Whether the customer churned.

3.2 Data Exploration

Here we will explore the data by checking the columns, statistical distribution for the numerical columns, the data types for each columns and unique values for columns necessary for this analysis

In [1]:
```
#Start by importing the required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings('ignore')
```

In [2]:
```
#loading the dataset
dt = pd.read_csv('syriatel.csv')
```

In [3]:
```
#Display the top 5 columns and bottom 5 columns
dt
```

Out[3]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	tota da charg
0	KS	128	415	382-4657	no	yes	25	265.1	110	4
1	OH	107	415	371-7191	no	yes	26	161.6	123	2
2	NJ	137	415	358-1921	no	no	0	243.4	114	4
3	OH	84	408	375-9999	yes	no	0	299.4	71	5
4	OK	75	415	330-6626	yes	no	0	166.7	113	2
...	...	...	...	...	...	...	...	...	...	
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	2
3329	WV	68	415	370-3271	no	no	0	231.1	57	3
3330	RI	28	510	328-8230	no	no	0	180.8	109	3
3331	CT	184	510	364-6381	yes	no	0	213.8	105	3
3332	TN	74	415	400-4344	no	yes	25	234.4	113	3

3333 rows × 21 columns

In [4]:
```
#checking the column distribution and their data-types
dt.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
```

In [5]:
```
#assert the column and row distribution is (3333row and 21 columns)
dt.shape
```

Out[5]:
```
(3333, 21)
```

In [6]:
```
#Checking the statistical distribution for numeric columns
dt.describe()
```

Out[6]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000

4 Data preparation

This phase involves transforming raw data into a structured and clean format suitable for modeling. This step ensures that the dataset is free from inconsistencies, missing values, and unnecessary variables while preparing it for machine learning algorithms.

4.1 Data Cleaning

This is the process of cleaning the dataset by:

- Dropping unnecessary columns
- Checking and dealing with missing values
- Dropping the duplicates
- Changing the columns format
- Checking for outliers

In [7]:
```
#creating a copy to maintain the original data
dt1=dt.copy(deep=True)
```

In [8]:
```
#confirming the creation of the copy
dt1.head()
```

Out[8]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.0
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.4
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.3
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.9
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.3

5 rows × 21 columns

In [9]:
```
#Changing column format to string
dt1.columns=dt1.columns.str.title()
#confirm changes
dt1.head()
```

Out[9]:

	State	Account Length	Area Code	Phone Number	International Plan	Voice Mail Plan	Number Vmail Messages	Total Day Minutes	Total Day Calls	Tota Day Charg
0	KS	128	415	382-4657	no	yes	25	265.1	110	45
1	OH	107	415	371-7191	no	yes	26	161.6	123	27
2	NJ	137	415	358-1921	no	no	0	243.4	114	41
3	OH	84	408	375-9999	yes	no	0	299.4	71	50
4	OK	75	415	330-6626	yes	no	0	166.7	113	28

5 rows × 21 columns

In [10]:
```
#Replacing the whitespaces with underscore(_)
dt1.columns=dt1.columns.str.replace(' ','_')
#Confirm the changes
dt1.head()
```

Out[10]:

	State	Account_Length	Area_Code	Phone_Number	International_Plan	Voice_Mail_Plan	Num
0	KS	128	415	382-4657	no	yes	
1	OH	107	415	371-7191	no	yes	
2	NJ	137	415	358-1921	no	no	
3	OH	84	408	375-9999	yes	no	
4	OK	75	415	330-6626	yes	no	

5 rows × 21 columns

In [11]:
```
#Dropping columns unnecessary for modeling
dt1.drop(['State'],axis=1,inplace=True)
```

In [12]:
```
#Confirm changes
dt1.head()
```

Out[12]:

	Account_Length	Area_Code	Phone_Number	International_Plan	Voice_Mail_Plan	Number_Vmai
0	128	415	382-4657	no	yes	
1	107	415	371-7191	no	yes	
2	137	415	358-1921	no	no	
3	84	408	375-9999	yes	no	
4	75	415	330-6626	yes	no	

In [13]:
```
#Checking for missing values
dt1.isnull().sum().any()
```

Out[13]: False

In [14]:
```
#Checking for duplicates
dt1.duplicated().sum()
```

Out[14]: 0

In [15]:
```
#changing column names for uniformity
dt1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Account_Length          3333 non-null   int64  
 1   Area_Code               3333 non-null   int64  
 2   Phone_Number            3333 non-null   object 
 3   International_Plan      3333 non-null   object 
 4   Voice_Mail_Plan         3333 non-null   object 
 5   Number_Vmail_Messages   3333 non-null   int64  
 6   Total_Day_Minutes       3333 non-null   float64
 7   Total_Day_Calls         3333 non-null   int64  
 8   Total_Day_Charge        3333 non-null   float64
 9   Total_Eve_Minutes       3333 non-null   float64
 10  Total_Eve_Calls         3333 non-null   int64  
 11  Total_Eve_Charge        3333 non-null   float64
 12  Total_Night_Minutes     3333 non-null   float64
 13  Total_Night_Calls       3333 non-null   int64  
 14  Total_Night_Charge      3333 non-null   float64
 15  Total_Intl_Minutes      3333 non-null   float64
 16  Total_Intl_Calls        3333 non-null   int64  
 17  Total_Intl_Charge       3333 non-null   float64
 18  Customer_Service_Calls  3333 non-null   int64  
 19  Churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 498.1+ KB
```

In [16]:
```
#Renaming columns for easy interpretability
dt1.rename(columns={'Number_Vmail_Messages':'Voice_Mail_Messages',
                    'Total_Eve_Minutes':'Total_Evening_Minutes',
                    'Total_Eve_Calls':'Total_Evening_Calls',
                    'Total_Eve_Charge':'Total_Evening_Charge',
                    'Total_Intl_Minutes':'Total_International_Minutes',
                    'Total_Intl_Calls':'Total_International_Calls',
                    'Total_Intl_Charge':'Total_International_Charge'

},inplace=True)
```

In [17]:
```
dt1.head()
```

Out[17]:

	Account_Length	Area_Code	Phone_Number	International_Plan	Voice_Mail_Plan	Voice_Mail_M
0	128	415	382-4657	no	yes	
1	107	415	371-7191	no	yes	
2	137	415	358-1921	no	no	
3	84	408	375-9999	yes	no	
4	75	415	330-6626	yes	no	

In [18]:
```
#changing the format for the Phone_Number category
#We do this by removing the hyphen and changing it into an interger

#Removing the hyphen
dt1['Phone_Number'] = dt1['Phone_Number'].str.replace('-', '')

#Changing the data type to integer
dt1['Phone_Number']=dt1['Phone_Number'].astype(int)

#Confirm the changes
dt1['Phone_Number'].value_counts().head()
```

Out[18]:
```
Phone_Number
3824657    1
3487071    1
3896082    1
4153689    1
3792503    1
Name: count, dtype: int64
```

4.2 Feature Engineering

Feature engineering is crucial for improving model accuracy and interpretability. In this project, the goal is to extract meaningful insights from the available features and create new ones that enhance the model's predictive power.

In [19]:
```
#Loading info to check columns useful feature engineering
dt1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Account_Length                3333 non-null   int64  
 1   Area_Code                     3333 non-null   int64  
 2   Phone_Number                  3333 non-null   int32  
 3   International_Plan            3333 non-null   object 
 4   Voice_Mail_Plan               3333 non-null   object 
 5   Voice_Mail_Messages           3333 non-null   int64  
 6   Total_Day_Minutes             3333 non-null   float64
 7   Total_Day_Calls               3333 non-null   int64  
 8   Total_Day_Charge              3333 non-null   float64
 9   Total_Evening_Minutes         3333 non-null   float64
 10  Total_Evening_Calls           3333 non-null   int64  
 11  Total_Evening_Charge          3333 non-null   float64
 12  Total_Night_Minutes           3333 non-null   float64
 13  Total_Night_Calls             3333 non-null   int64  
 14  Total_Night_Charge            3333 non-null   float64
 15  Total_International_Minutes   3333 non-null   float64
 16  Total_International_Calls     3333 non-null   int64  
 17  Total_International_Charge    3333 non-null   float64
 18  Customer_Service_Calls        3333 non-null   int64  
 19  Churn                         3333 non-null   bool   
dtypes: bool(1), float64(8), int32(1), int64(8), object(2)
memory usage: 485.1+ KB
```

4.3 Adding new columns

I will begin my feature engineering process by adding new columns that may enhance the models predictive power.

4.3.1 Using mathematical operations

In this step I will use mathematical operations which are addition(+) and division(/) to add the new columns. The process is as shown below:

In [20]:
```
#Creating a new columns
#Total Minutes column
dt1['Total_Minutes']=(dt1['Total_Day_Minutes']+dt1['Total_Evening_Minutes']
                      +dt1['Total_International_Minutes']
                      +dt1['Total_Night_Minutes'])

#Total Calls column
dt1['Total_Calls']=(dt1['Total_Day_Calls']+dt1['Total_Evening_Calls']
                    +dt1['Total_International_Calls']
                    +dt1['Total_Night_Calls'])

#Total Charges column
dt1['Total_Charges']=(dt1['Total_Day_Charge']+dt1['Total_Evening_Charge']
                      +dt1['Total_International_Charge']+dt1['Total_Night_Charge'])
```

In [21]:
```
#Create the average call duration
#call duration
# Divide total minutes by Total calls
dt1['Avg_Call_Duration']=dt1['Total_Minutes']/dt1['Total_Calls']
```

In [22]:
```
#Confirm the changes
dt1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 24 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Account_Length                3333 non-null   int64  
 1   Area_Code                     3333 non-null   int64  
 2   Phone_Number                  3333 non-null   int32  
 3   International_Plan            3333 non-null   object 
 4   Voice_Mail_Plan               3333 non-null   object 
 5   Voice_Mail_Messages           3333 non-null   int64  
 6   Total_Day_Minutes             3333 non-null   float64
 7   Total_Day_Calls               3333 non-null   int64  
 8   Total_Day_Charge              3333 non-null   float64
 9   Total_Evening_Minutes         3333 non-null   float64
 10  Total_Evening_Calls           3333 non-null   int64  
 11  Total_Evening_Charge          3333 non-null   float64
 12  Total_Night_Minutes           3333 non-null   float64
 13  Total_Night_Calls             3333 non-null   int64  
 14  Total_Night_Charge            3333 non-null   float64
 15  Total_International_Minutes   3333 non-null   float64
 16  Total_International_Calls     3333 non-null   int64  
 17  Total_International_Charge    3333 non-null   float64
 18  Customer_Service_Calls        3333 non-null   int64  
 19  Churn                         3333 non-null   bool   
 20  Total_Minutes                 3333 non-null   float64
 21  Total_Calls                   3333 non-null   int64  
 22  Total_Charges                 3333 non-null   float64
 23  Avg_Call_Duration             3333 non-null   float64
dtypes: bool(1), float64(11), int32(1), int64(9), object(2)
memory usage: 589.3+ KB
```

From the observations above it is evident that four new columns have been added

4.3.1.1 Justification

1. **Total Minutes:**

- Represents the total usage of call minutes by a customer, which helps classify users into low, moderate, or heavy users.
- Heavy users might churn due to high costs, while low users might churn due to lack of engagement.
- Helps understand customer behavior trends—do customers who talk more churn less or more?

2. **Total Calls:**

- Total calls reflect the frequency of customer interactions.
- A high number of calls may indicate high customer dependency on the service, meaning they might stay loyal or demand better service.

3. **Total Charges:**

- This feature provides a holistic view of the total amount spent by a customer across all time periods.
- High total charges may indicate high engagement, which can relate to customer satisfaction or dissatisfaction (if they feel overcharged).
- It helps detect revenue patterns, identifying customers who contribute more to the company's revenue and may require customer retention strategies.

4. **Average Call Duration:**

- Average call duration helps determine how engaged a customer is with the service.
- A longer call duration might indicate that the customer is either seeking assistance (which could be tied to dissatisfaction or service-related issues) or engaged in complex conversations that require more time.
- Shorter call durations might imply that the customer is either calling for quick issues or may not be fully utilizing the service.
- Longer average call duration might correlate with customers struggling with the service or facing difficulties, both of which can lead to churn if not addressed.
- Customers who are on the phone with customer service representatives for longer durations could indicate frustration or dissatisfaction, making them more likely to churn if issues aren't resolved.
- On the other hand, a low average call duration might indicate a satisfied customer, potentially less prone to churn.

4.3.2 Using conditional statements

Using if-else statements to categorize Customer Service Calls into High and Low is an effective approach for simplifying the analysis and focusing on significant patterns.

In [23]:
```
#Checking the column distribution
dt1['Customer_Service_Calls'].value_counts()
```

Out[23]:
```
Customer_Service_Calls
1    1181
2     759
0     697
3     429
4     166
5      66
6      22
7       9
9       2
8       2
Name: count, dtype: int64
```

In [24]:
```
#Categorizing into high and low using conditional statements
def categorize_service_calls(calls):
    if isinstance(calls, int):  # Has to be an integer
        return "High" if calls >= 5 else "Low"
    else:
        raise ValueError(f"Invalid input: {calls} is not an integer")#Raise errors if no

# Apply function to the column with error handling
try:
    dt1['Customer_Service_Category'] = dt1['Customer_Service_Calls'].apply(categorize_se
except ValueError as err:
    print(err)

# Display the updated DataFrame to confirm changes
dt1.head()
```

Out[24]:

	Account_Length	Area_Code	Phone_Number	International_Plan	Voice_Mail_Plan	Voice_Mail_M
0	128	415	3824657	no	yes	
1	107	415	3717191	no	yes	
2	137	415	3581921	no	no	
3	84	408	3759999	yes	no	
4	75	415	3306626	yes	no	

5 rows × 25 columns

In [25]:
```
#Confirmning the creation of the new column and checking its distribution
dt1['Customer_Service_Category'].value_counts()
```

Out[25]:
```
Customer_Service_Category
Low     3232
High     101
Name: count, dtype: int64
```

4.3.2.1 Justification

1. **Simplifying Analysis:**

- By creating a binary feature (High vs. Low), you can easily analyze and visualize the impact of customer service call frequency on churn.
- This categorization simplifies the model, allowing it to focus on whether a customer is a heavy user of customer service or not, rather than dealing with a continuous variable.

2. **Identifying Key Segments:**

- High calls (>=5) typically indicate frequent contact with customer support, which could signal unresolved issues, dissatisfaction, or the customer facing service-related challenges.
- Low calls (<5) indicate that the customer uses customer service less often, suggesting they might either have fewer problems or are less engaged with the service overall.

3. **Improved Churn Predictions:**

- Having "High" vs. "Low" as a feature makes it easier to spot trends. For instance:
- High Customer Service Calls = likely dissatisfaction, which could be a predictor of churn.
- Low Customer Service Calls = suggests the customer is either happy or not using the service as much, potentially less likely to churn.

4.4 Exploratory Data Analysis (EDA)

For this phase I will make visualizations based on the key parameters that will enhance my model performance. These parameters include the key features such as ; total calls, total minutes, total charges and the target variable which is the churn.

Visualizations crucial for this particular EDA include

- Histplot
- Boxplot
- Countplot
- Scatterplot
- Barplot
- Lineplot

Steps to follow include:

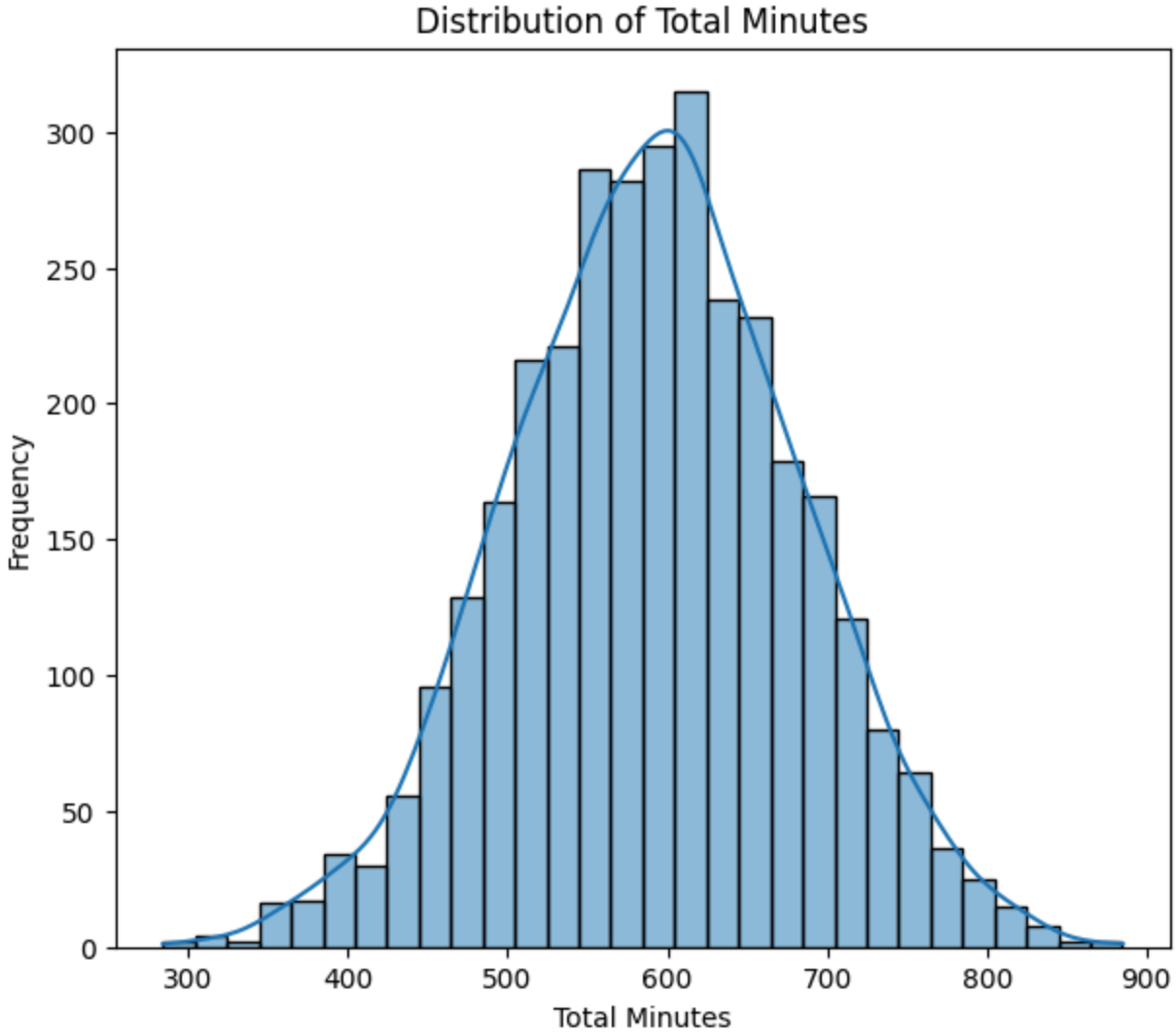
- Univariate Analysis
- Bi-variate Analysis
- Multivariate Analysis

4.4.1 Univariate Analysis

4.4.1.1 Univariate Analysis for numerical data

Distribution of Total Minutes

In [26]:
```
#Plotting the kde for total minutes
plt.figure(figsize=(7, 6))
sns.histplot(dt1['Total_Minutes'], kde=True, bins=30)
plt.title('Distribution of Total Minutes')
plt.xlabel('Total Minutes')
plt.ylabel('Frequency')
plt.show()
```

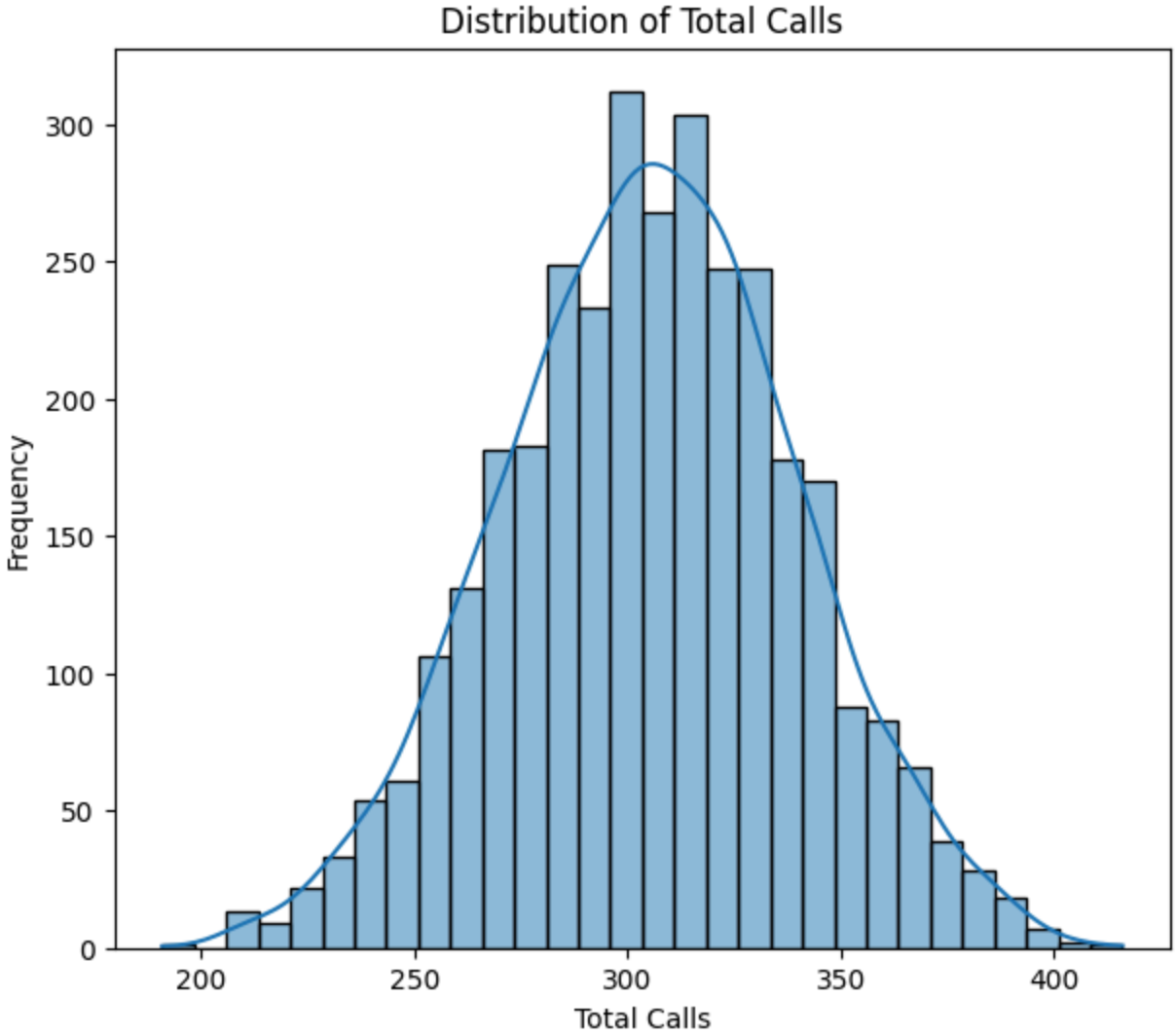


Observations:

- **Normal Distribution:** The histogram and KDE curve suggest that "Total Minutes" is approximately normally distributed. The bell shape is fairly symmetrical, and the KDE curve smooths this out nicely. This is a good sign for many modeling algorithms that assume normality
- **Central Tendency:** The distribution's peak (and thus the mean and median, given the symmetry) appears to be somewhere around 600-650 total minutes. This tells you the "average" total minutes used by your customers.
- **Spread/Variability:** The distribution shows a reasonable amount of spread or variability around the mean. Most customers fall within the range of roughly 400 to 800 total minutes. This spread is important; if the data was too tightly packed, it might mean the feature isn't very discriminative for churn.

Distribution of Total calls

In [27]:
```
#Plotting the kde for total calls
plt.figure(figsize=(7, 6))
sns.histplot(dt1['Total_Calls'], kde=True, bins=30)
plt.title('Distribution of Total Calls')
plt.xlabel('Total Calls')
plt.ylabel('Frequency')
plt.show()
```

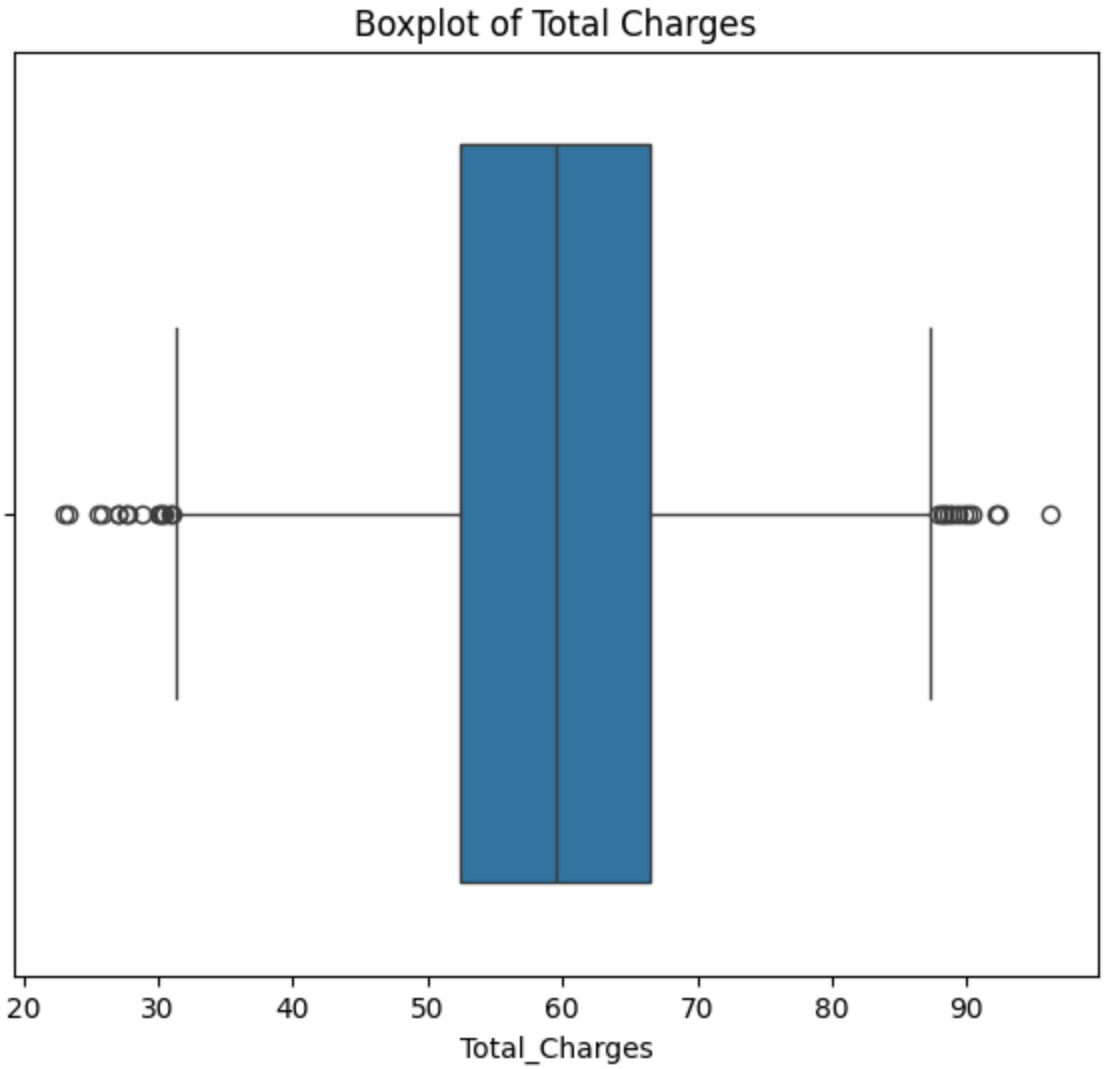


Observations:

- **Normal Distribution:** Like the "Total Minutes" visualization, this one also shows an approximately normal distribution. The bell shape is fairly symmetrical, and the KDE curve smooths this out nicely.This suggests that the number of calls made by customers tends to cluster around a central value.
- **Central Tendency:** The peak of the distribution appears to be somewhere around 300-325 total calls.This represents the "average" or most common number of calls made by your customers.
- **Spread/Variability:** The distribution shows a reasonable amount of spread. Most customers fall within the range of roughly 250 to 375 total calls.This spread is important for potential predictive power.

Boxplot distribution for Total charges

In [28]:
```
# Boxplot for Total day charge
plt.figure(figsize=(7, 6))
sns.boxplot(x=dt1['Total_Charges'])
plt.title('Boxplot of Total Charges')
plt.show()
```



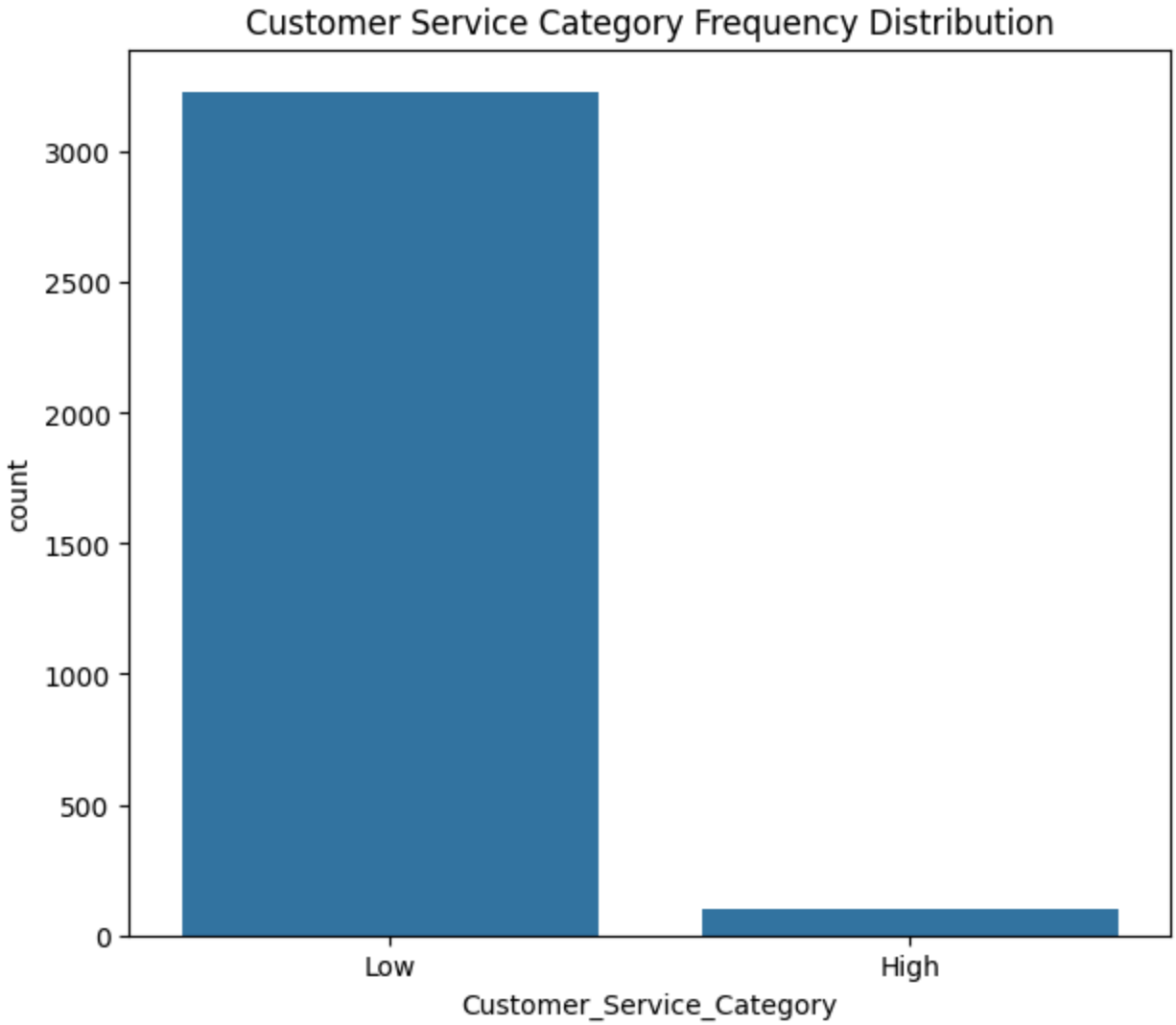
Observations:

- Median: The line inside the box represents the median of the "Total Charges." It looks to be around 60-65. This means that 50% of your customers have total charges below this value, and 50% have charges above it.
- There are several outliers on both the lower and upper ends of the distribution.

4.4.1.2 Univariate Analysis for categorical data

Customer Service category plan frequency distribution

In [29]:
```
#count plot for customer service category
plt.figure(figsize=(7, 6))
sns.countplot(x='Customer_Service_Category', data=dt1)
plt.title('Customer Service Category Frequency Distribution')
plt.show()
```

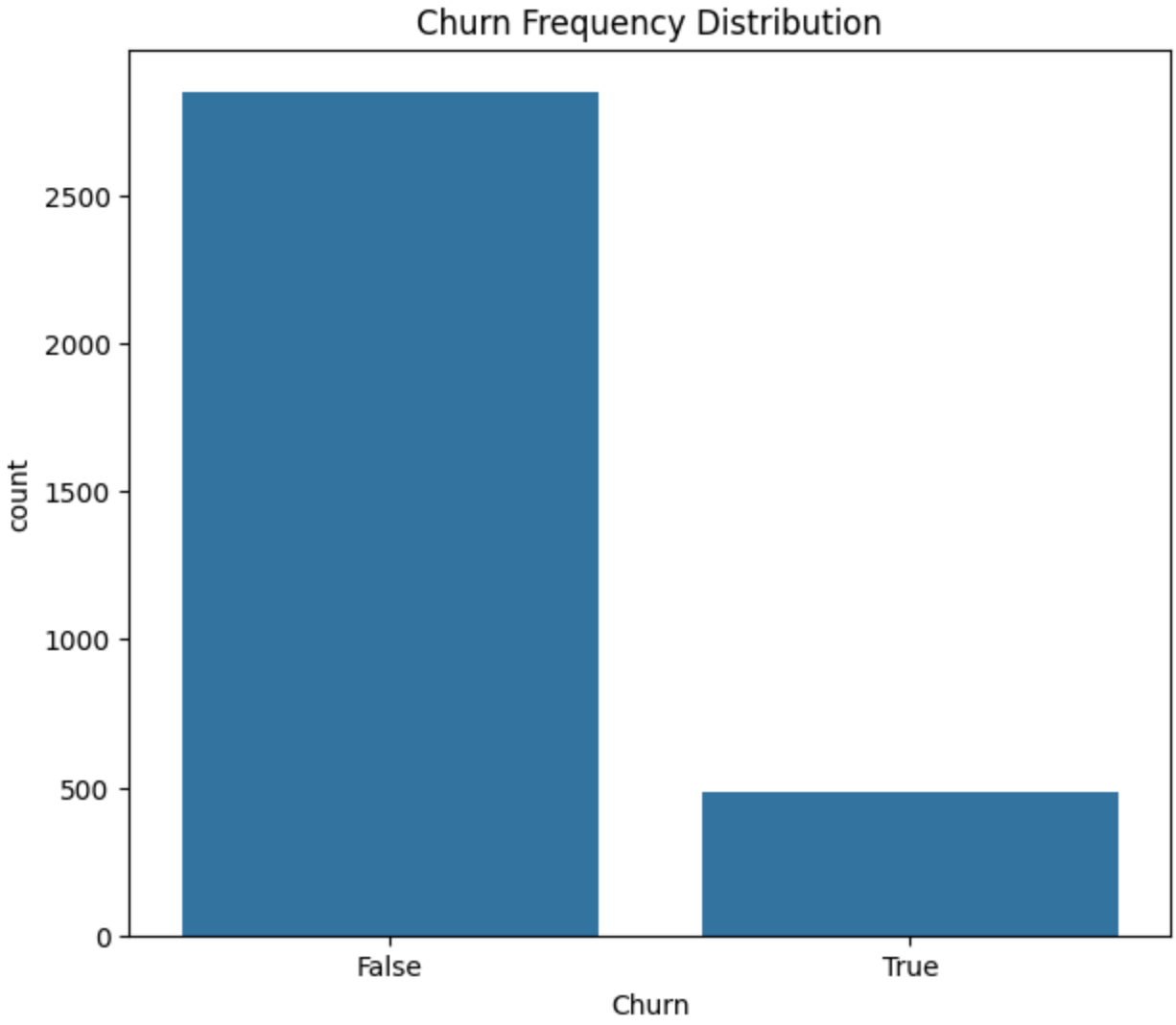


Observations:

- **Imbalanced Classes:** The most striking observation is the severe class imbalance. The "Low" customer service category has a much higher frequency than the "High" category. This means that the vast majority of your customers fall into the "Low" category.
- **Dominance of "Low" Category:** The bar for "Low" is significantly taller, indicating that it represents a substantial portion of your customer base. The exact count isn't explicitly provided, but we can visually infer that it's likely above 3000.
- **Small "High" Category:** The bar for "High" is very short, suggesting that only a small fraction of your customers are in this category. The exact count appears to be around 100.

Frequency distribution for churn (Target)

In [30]:
```
#Countplot for the target variable
plt.figure(figsize=(7, 6))
sns.countplot(x='Churn', data=dt1)
plt.title('Churn Frequency Distribution')
plt.show()
```



Observations:

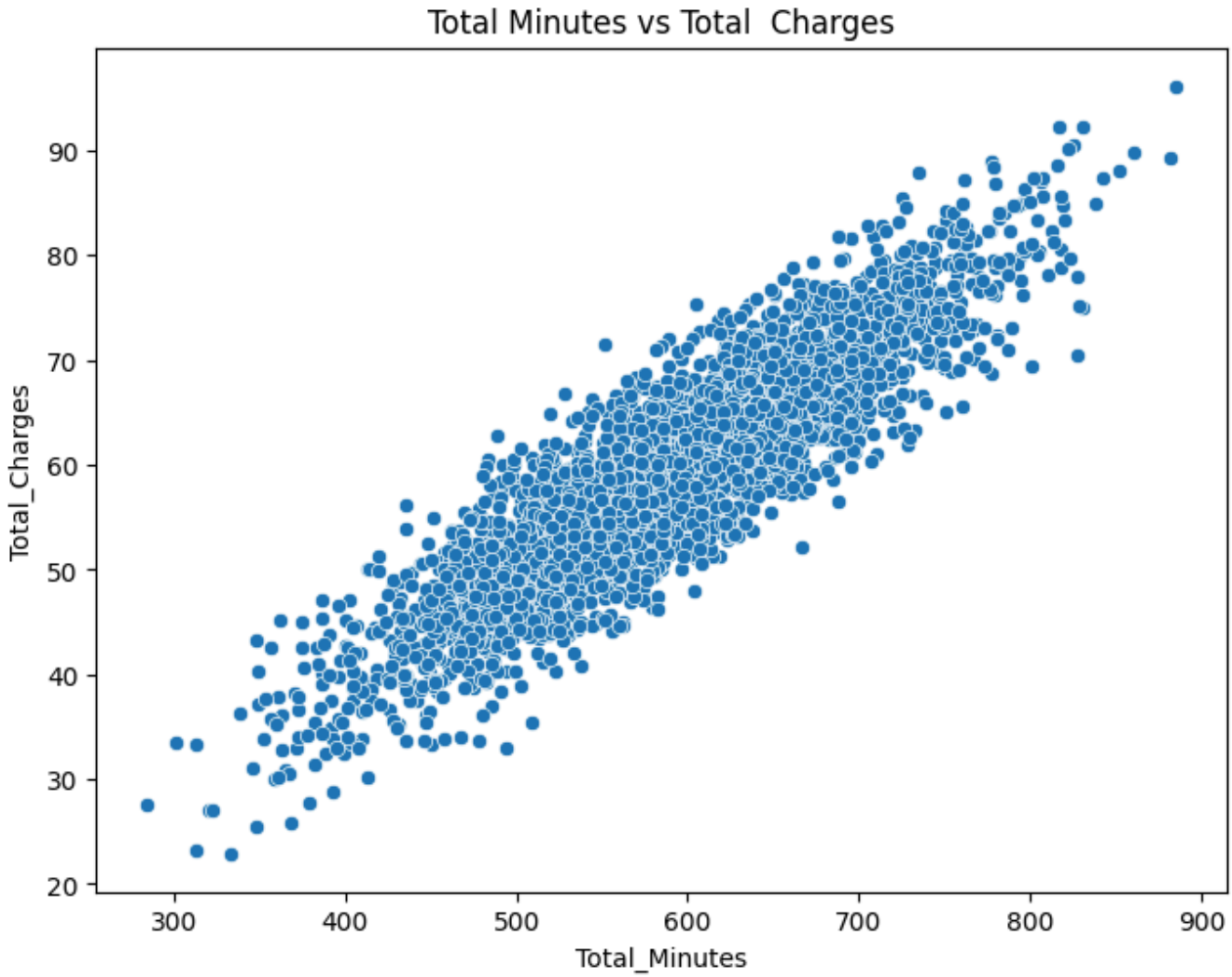
- **Class Imbalance (Moderate):** While not as extreme as the "Customer Service Category," there's still a noticeable class imbalance. The "False" (no churn) category has a substantially higher count than the "True" (churn) category.
- **Majority Class; No Churn:** The taller bar for "False" indicates that the majority of your customers have not churned. This is common in many churn prediction scenarios, as businesses typically have a lower churn rate than their retention rate.
- **Minority Class; Churn:** The shorter bar for "True" shows the proportion of customers who have churned. The exact count isn't explicitly provided, but we can visually infer that it's likely somewhere around 500 out of a total of approximately 3300 customers, suggesting a churn rate in the neighborhood of 15%.

4.4.2 Bi-variate Analysis

4.4.2.1 Numeric Vs Numeric

Minutes Vs Charges

In [31]:
```
# Scatter plot for 'Total day minutes' vs 'Total day charge'
plt.figure(figsize=(7, 6))
sns.scatterplot(x='Total_Minutes', y='Total_Charges', data=dt1)
plt.title('Total Minutes vs Total  Charges')
plt.show()
```



Observations:

- **Strong Positive Correlation:** The scatter plot clearly shows a strong positive correlation between "Total Minutes" and "Total Charges." As "Total Minutes" increases, "Total Charges" also tends to increase. This suggests that customers who use more minutes tend to have higher total charges.
- **Linear Relationship:** The relationship appears to be largely linear. The points roughly fall along a straight line, indicating a consistent proportional increase in charges with increased minutes.

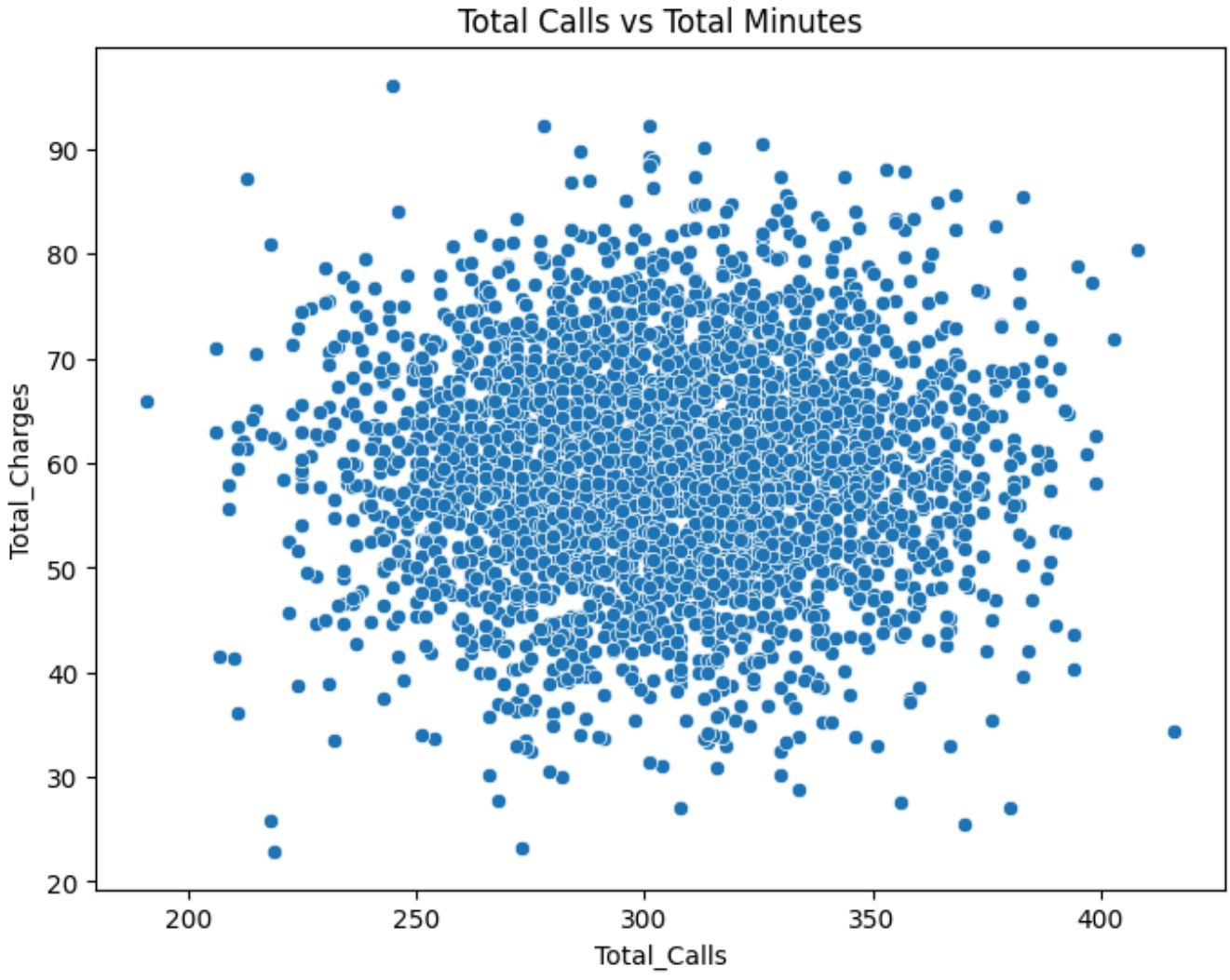
- **No Obvious Clusters:** There aren't any distinct clusters visible in the scatter plot. The points seem to be relatively evenly distributed along the linear trend.

Trade-off:

- In classification models (like logistic regression, decision trees, random forests, etc.), multicollinearity doesn't have the same negative impact. While it can still lead to less stable models and potentially overfitting in some cases, classification algorithms like decision trees or random forests tend to be less sensitive to multicollinearity because they don’t rely on linear relationships.

Call Vs Minutes

In [32]:
```
# Scatter plot for 'Total day minutes' vs 'Total day charge'
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Total_Calls', y='Total_Charges', data=dt1)
plt.title('Total Calls vs Total Minutes')
plt.show()
```



Observations:

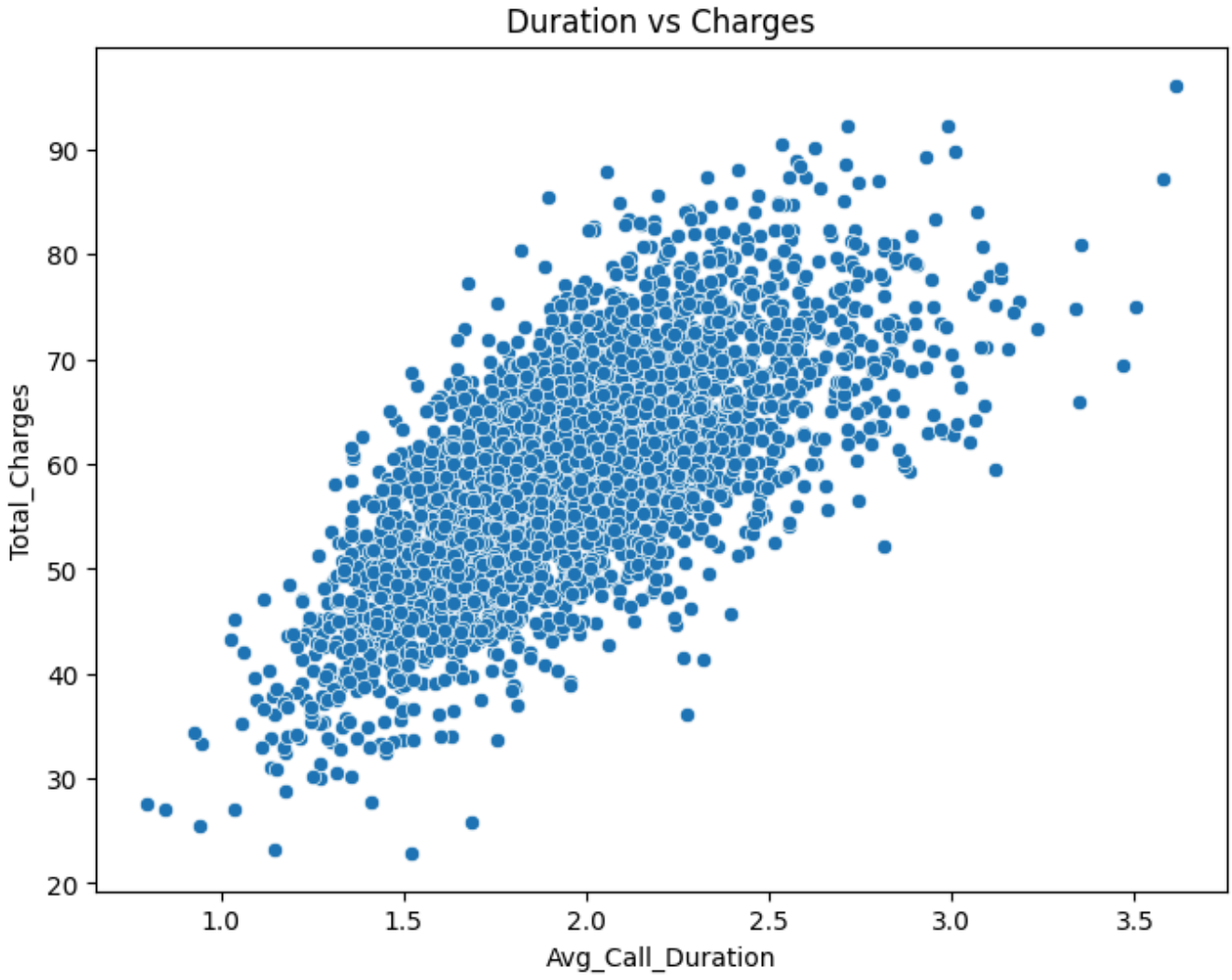
- **No Linear Correlation:** Unlike the "Total Minutes vs. Total Charges" plot, this one doesn't show a strong linear correlation. The points appear to be scattered randomly with no clear trend. This suggests that there's little to no direct linear relationship between the total number of calls and the total number of

minutes used.

- **Diffuse Scatter:** The points are diffusely scattered across the plot, indicating that customers with similar numbers of calls can have widely varying total minutes, and vice-versa.
- **Possible Non-Linear Relationship:** While a linear relationship is not evident, it's worth considering if there might be a non-linear relationship or if there are underlying groupings or clusters that are not immediately apparent.
- **Concentration Around the Mean:** There seems to be a concentration of points around the center of the plot, indicating that most customers fall within a certain range of both "Total Calls" and "Total Minutes."

Duration Vs Charges

In [33]:
```
#Scatter plot for call duration vs Total charges
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Avg_Call_Duration', y='Total_Charges', data=dt1)
plt.title('Duration vs Charges')
plt.show()
```



Observations:

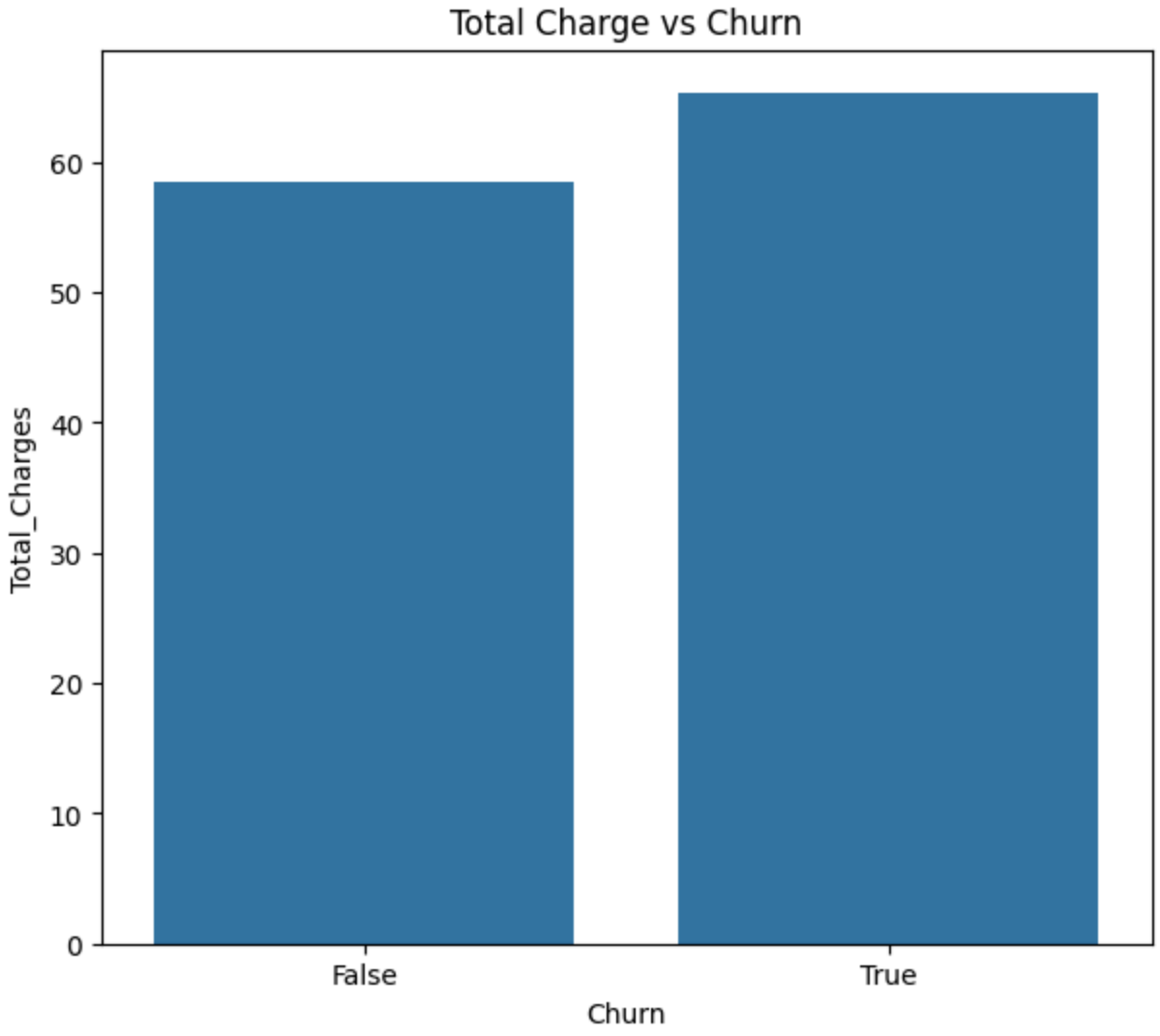
- **Positive Correlation:** The scatter plot shows a positive correlation between average call duration and total charges. As the average call duration increases, the total charges also tend to increase. This makes intuitive sense – longer calls generally lead to higher charges.

- **Non-Linearity (Possible):** While there's a positive trend, the relationship might not be strictly linear. The points seem to form a slightly curved pattern, especially at higher average call durations. This suggests that the increase in charges might not be directly proportional to the increase in call duration.
- **Clustering or Density Variation:** The density of points seems to vary across the plot. There's a denser concentration of points in the lower-left region (shorter call durations, lower charges). As the average call duration increases, the points become more spread out, suggesting greater variability in total charges for longer calls.

4.4.2.2 Categorical Vs Numeric

Charges Vs Churn

In [34]:
```
# Boxplot for 'Churn' vs 'Total day charge'
fig, ax = plt.subplots(figsize=(7,6 ))
sns.barplot(x='Churn', y='Total_Charges', data=dt1, ax=ax, ci=None)  # Added ci=None her
plt.title('Total Charge vs Churn')
plt.show()
```

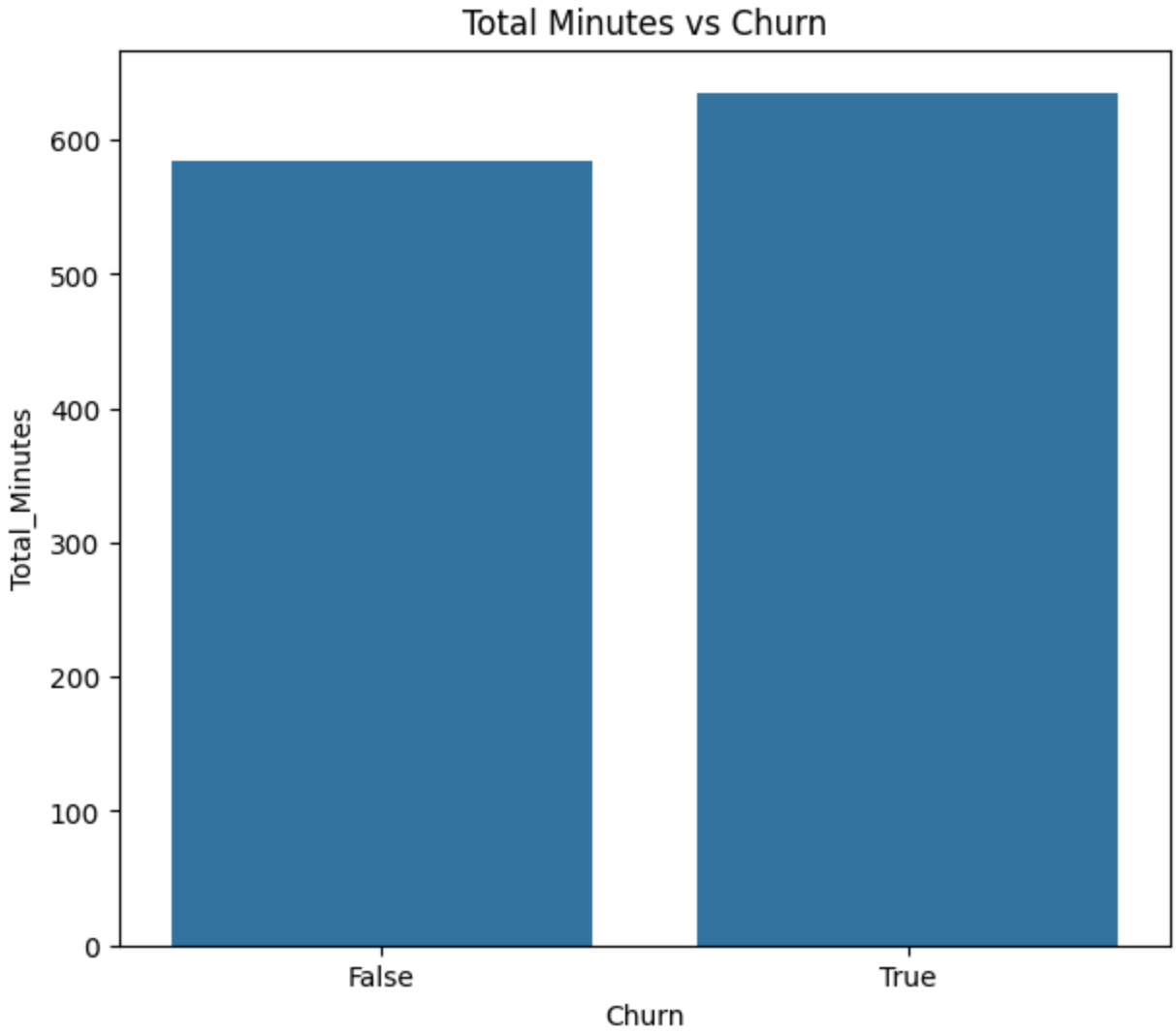


Observations:

- **Higher Average Charges for Churned Customers:** The bar chart indicates that, on average, customers who have churned ("True") tend to have higher total charges than customers who haven't churned ("False").
- **Potential for Predictive Power:** This difference in average total charges suggests that "Total Charges" could be a useful feature for predicting churn. Customers with higher total charges might be more likely to churn.

Minutes Vs Churn

In [35]:
```
# Barplot for 'Churn' vs 'Total day charge'
fig, ax = plt.subplots(figsize=(7,6 ))
sns.barplot(x='Churn', y='Total_Minutes', data=dt1, ax=ax, ci=None)  # Added ci=None her
plt.title('Total Minutes vs Churn')
plt.show()
```



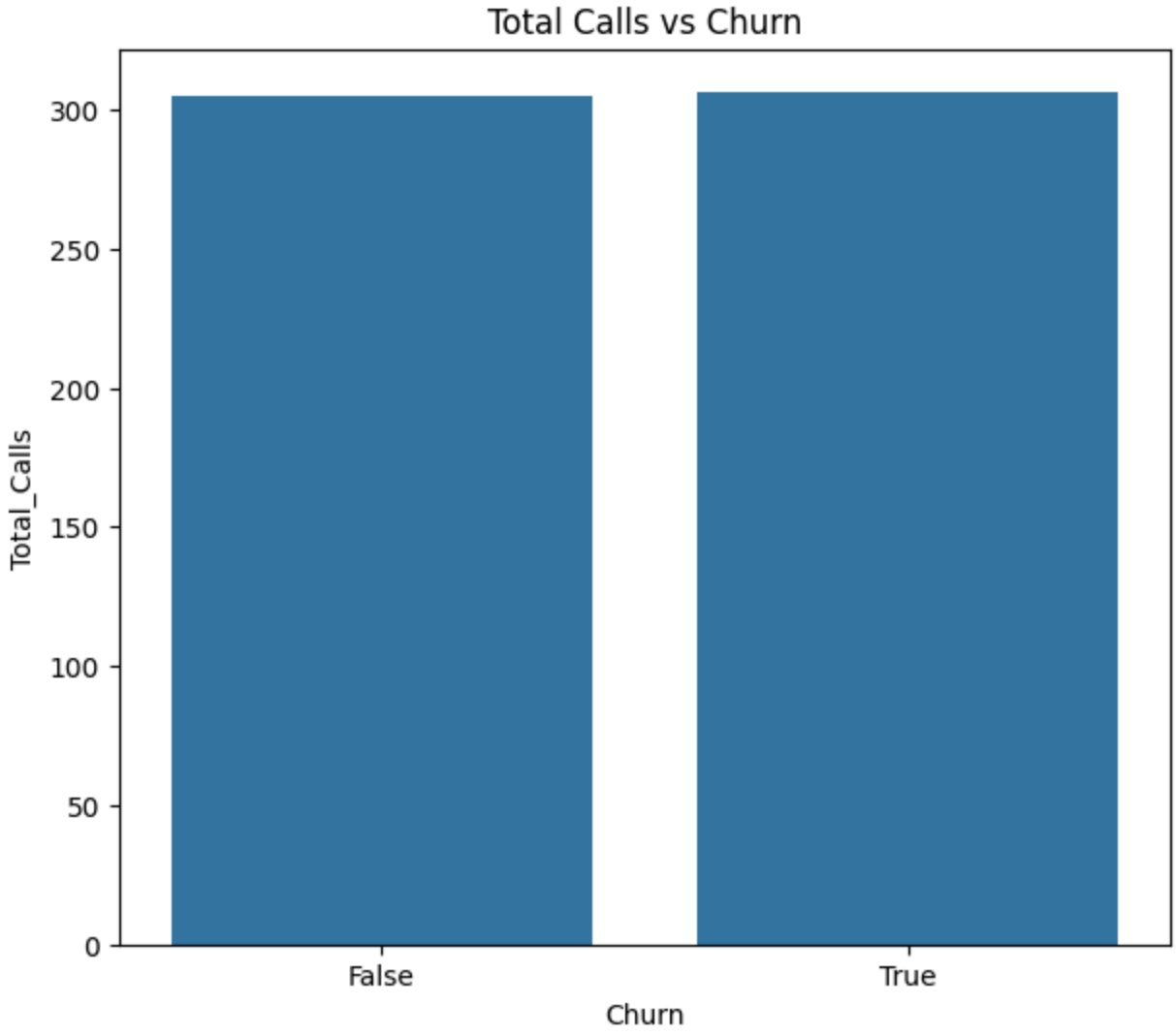
Observations:

- **Higher Average Minutes for Churned Customers:** The bar chart indicates that, on average, customers who have churned ("True") tend to have higher total minutes than customers who haven't churned ("False").

- **Potential for Predictive Power:** This difference in average total minutes suggests that "Total Minutes"

Calls Vs Churn

In [36]:
```
# Barplot for 'Churn' vs 'Total day charge'
fig, ax = plt.subplots(figsize=(7,6 ))
sns.barplot(x='Churn', y='Total_Calls', data=dt1, ax=ax, ci=None)  # Added ci=None here
plt.title('Total Calls vs Churn')
plt.show()
```



Observations:

- **Similar Average Total Calls:** The most striking observation is that the average total calls are very similar for both churned and non-churned customers. The bars are almost the same height, suggesting that there's little to no difference in the average number of calls made between the two groups.
- **Low Predictive Power (Likely):** The lack of a noticeable difference in average total calls strongly suggests that this feature, by itself, is likely to have low predictive power for churn.

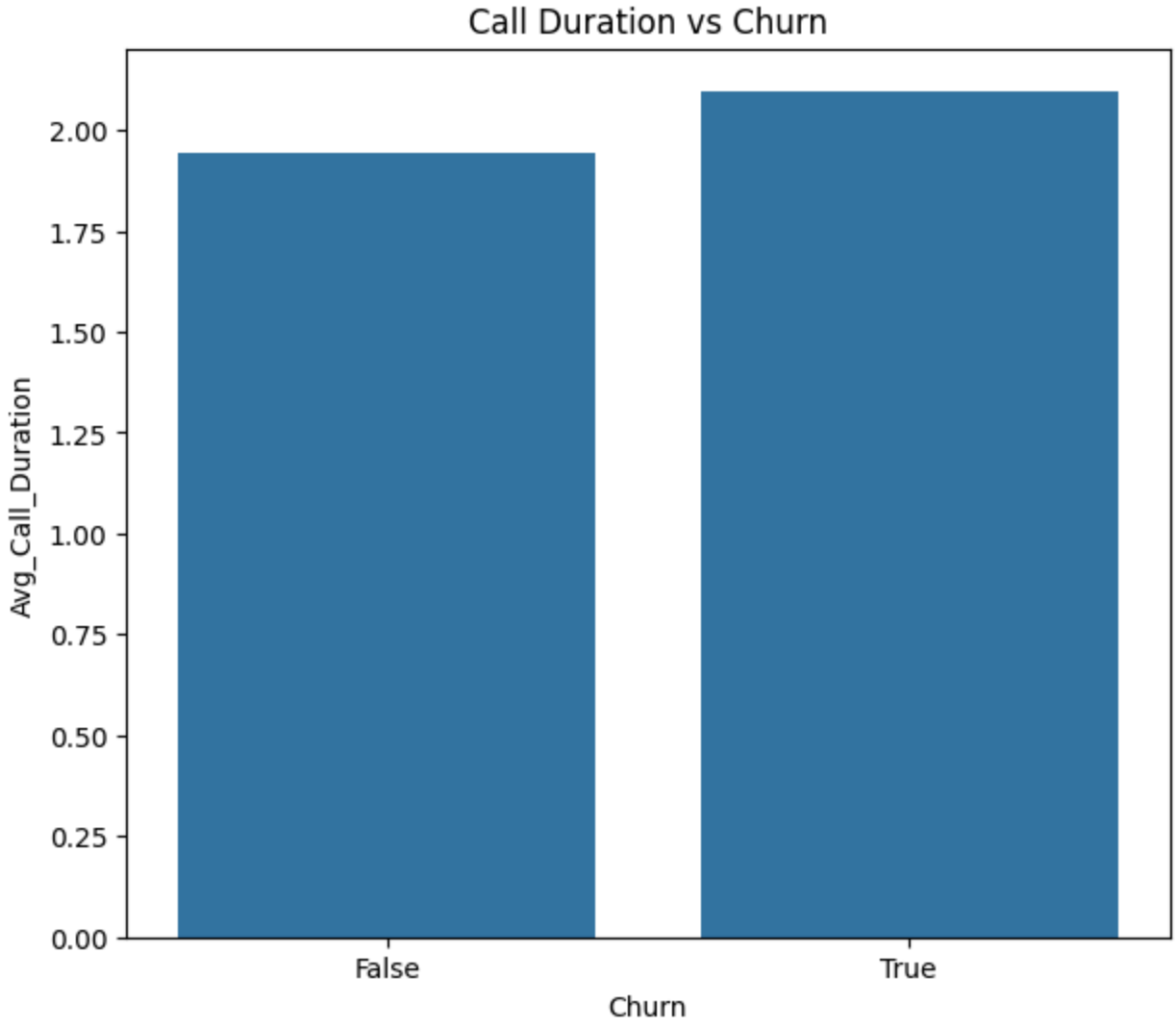
Reason:

- The sheer number of calls a customer makes doesn't necessarily reflect their engagement or satisfaction with the service. Someone could make many short calls for simple inquiries, while another

customer might have fewer, longer calls dealing with complex issues. "Total Calls" alone doesn't capture this nuance.

Call Duration Vs Churn

In [37]:
```
#barplot for call duration vs churn
fig, ax = plt.subplots(figsize=(7,6 ))
sns.barplot(x='Churn', y='Avg_Call_Duration', data=dt1, ax=ax, ci=None)  # Added ci=None
plt.title('Call Duration vs Churn')
plt.show()
```



Observations:

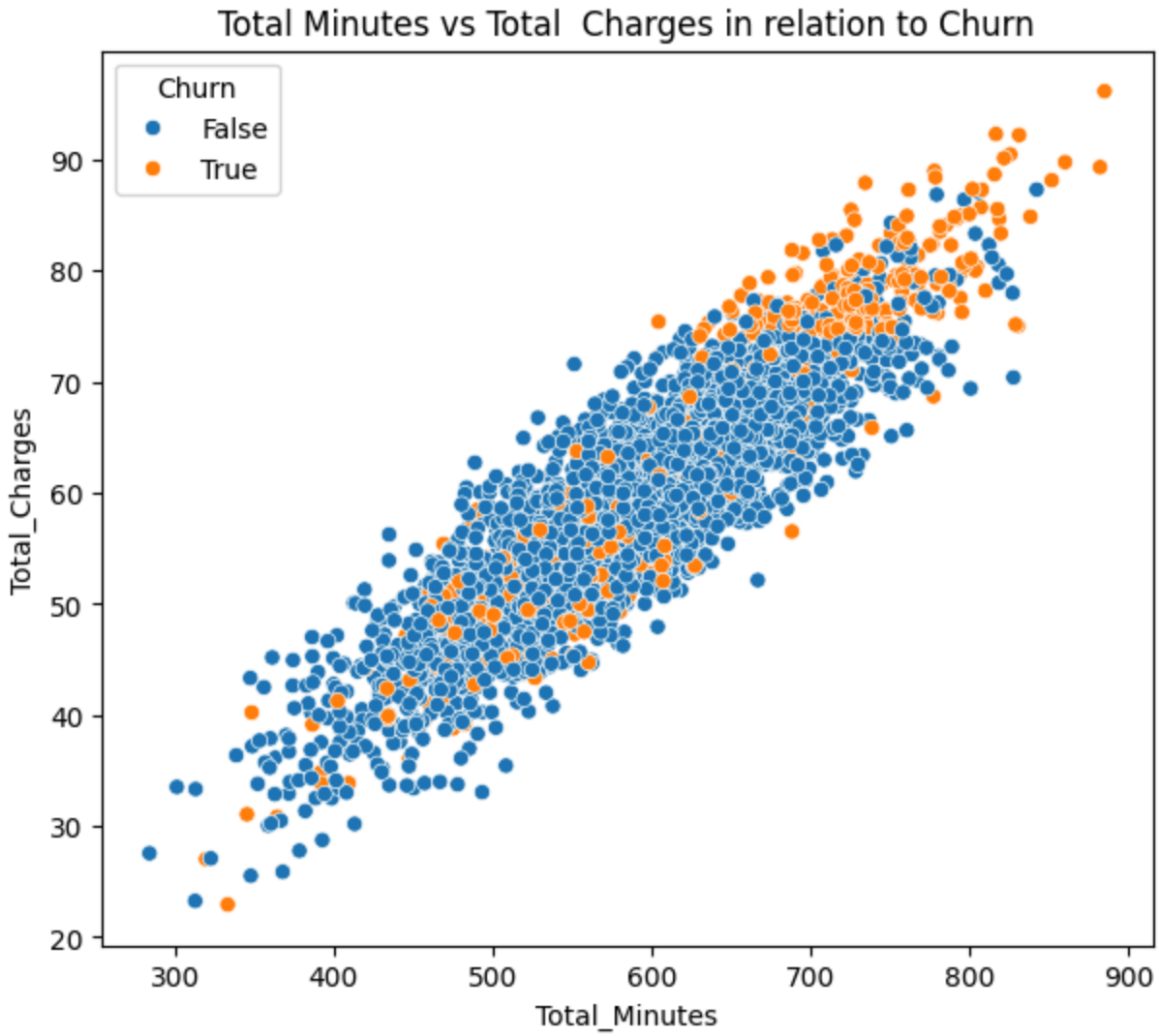
- **Slightly Higher Average Call Duration for Churned Customers:** The chart suggests that, on average, customers who churn ("True") tend to have slightly higher average call durations than those who don't churn ("False"). The difference is visible but not dramatic.
- **Potential for Some Predictive Power:** The slight difference in average call duration suggests that this feature might have some predictive power for churn, but it's unlikely to be a very strong predictor on its own.

4.4.3 Multi-variate Analysis

4.4.3.1 Numeric variables with relation to the target variable(Churn) using scatter plots

Total Minutes Vs Total charges in relation to Churn

In [102]:
```
# Scatter plot for 'Total day minutes' vs 'Total day charge'
plt.figure(figsize=(7, 6))
sns.scatterplot(x='Total_Minutes', y='Total_Charges',hue='Churn', data=dt1)
plt.title('Total Minutes vs Total  Charges in relation to Churn')
plt.show()
```



Observations:

Concentration in the Higher Range: The "True" (churned) points are predominantly concentrated in the upper right region of the scatter plot, corresponding to higher values of both Total_Minutes and Total_Charges. This indicates that customers with higher minutes and charges are more likely to churn.

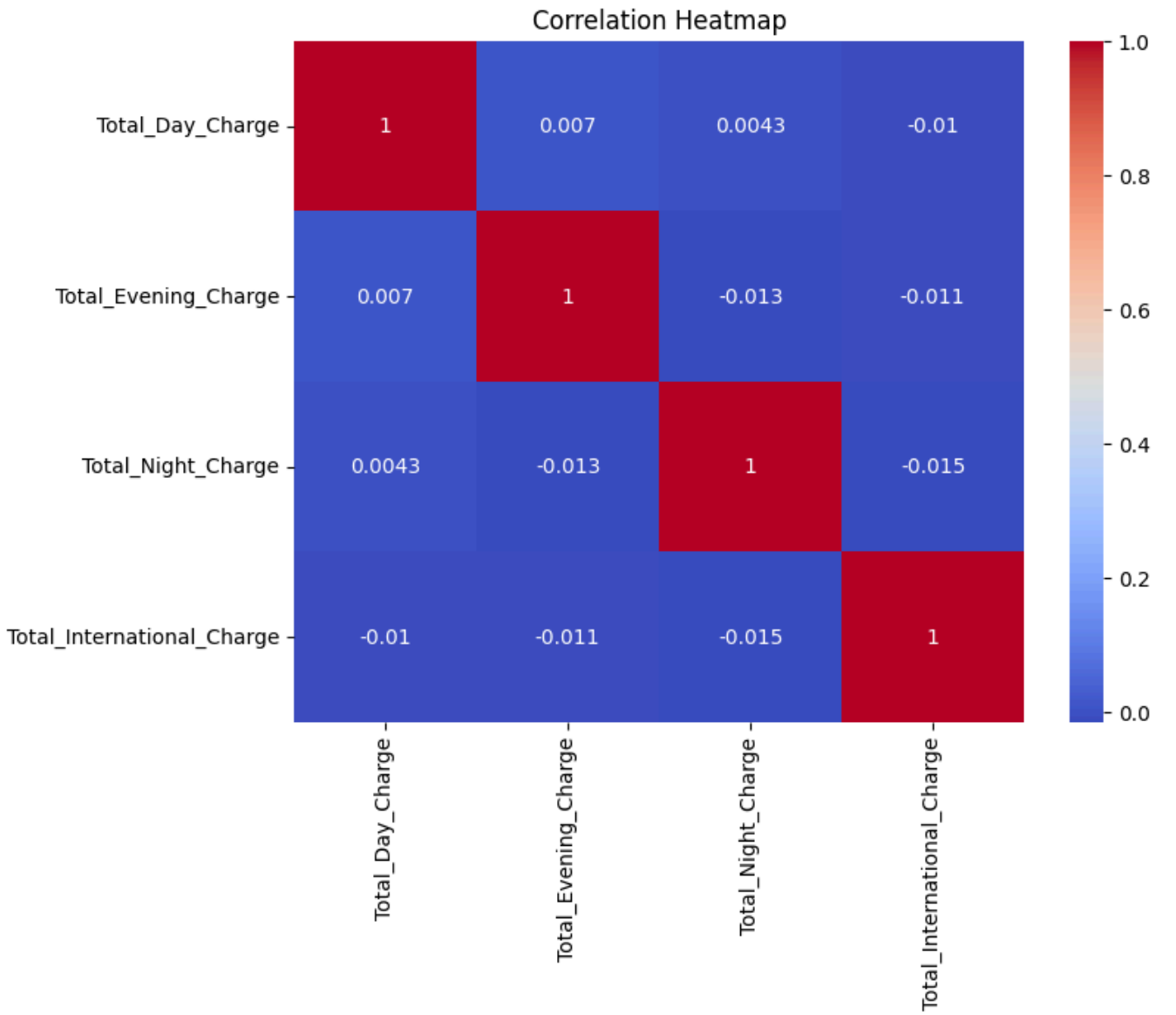
Overlapping with "False" (Not Churned): While concentrated in the higher range, the "True" points are not entirely separated from the "False" (not churned) points. There's significant overlap, especially in the middle range of minutes and charges. This suggests that while high minutes and charges increase the likelihood of churn, they are not the sole determinants.

Spread Across a Range: The "True" points are spread across a range of minutes and charges, not

4.4.3.2 Correlation matrices

Correlation matrix for Charge

In [38]:
```
# Correlation heatmap
correlation_matrix = dt1([['Total_Day_Charge', 'Total_Evening_Charge', 'Total_Night_Char
                          .corr())
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

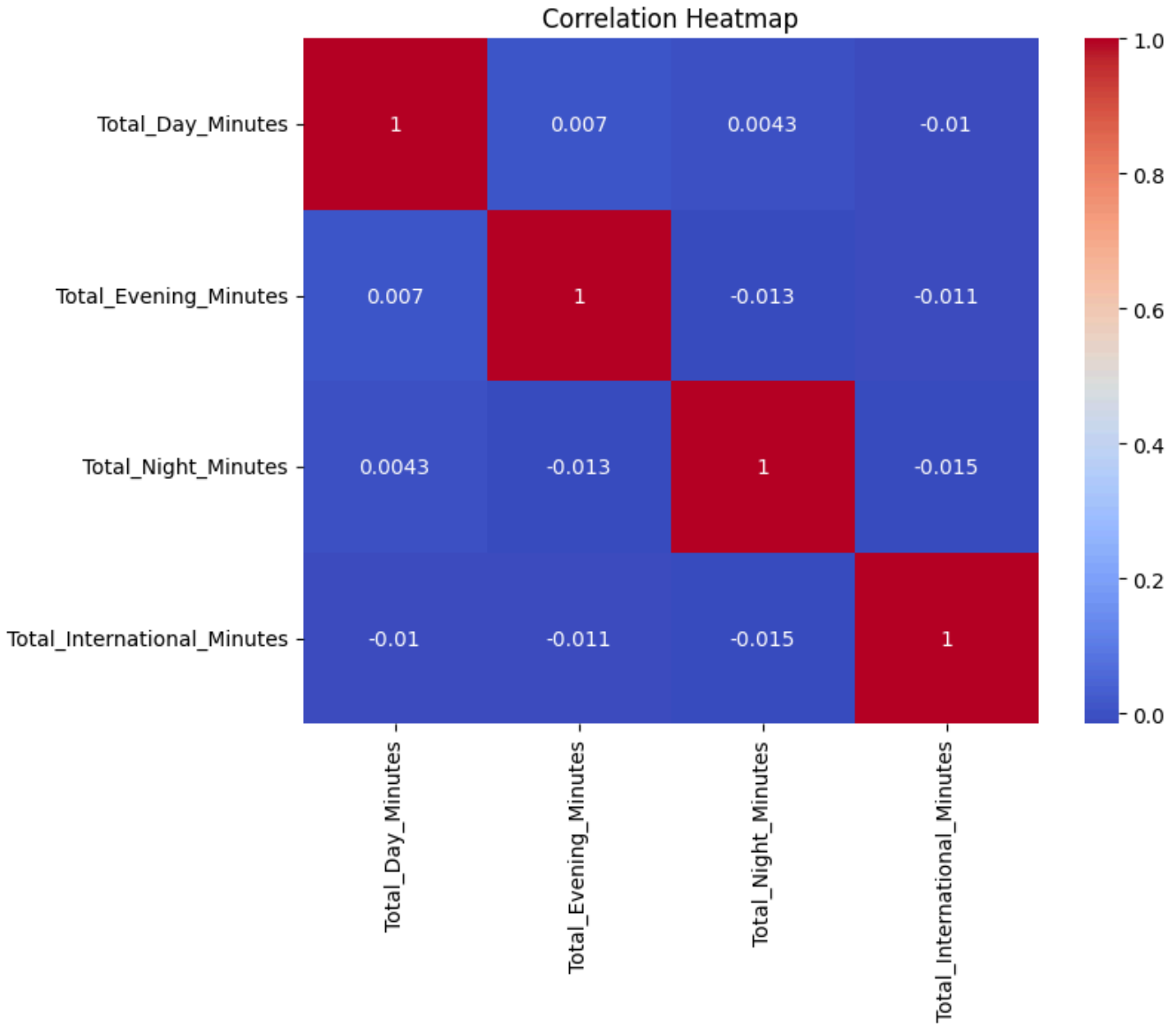


Observations:

- **Weak or No Linear Correlation:** The most prominent observation is that all the correlation coefficients are very close to zero. The color intensity is very low, indicating a lack of strong linear relationships between any pair of variables.
- **Near-Zero Coefficients:** The values are all extremely close to zero, suggesting that there's virtually no linear relationship between any two charge components.
- **No Multicollinearity Concerns:** Since the correlations are so weak, multicollinearity is not a concern for these variables. You can safely include all of them in your model without worrying about instability or interpretability issues related to multicollinearity.
- **Further Analysis and Considerations for Churn Modeling:**
- **Why are Correlations So Weak?** The weak correlations suggest that the charges for different times of day and international calls are largely independent of each other. This might be because:
- Customers have different calling patterns at different times.
- International calls are a separate category with different drivers.
- Pricing structures might be designed to separate these charges.

Correlation matrix for Minutes

In [39]:
```
# Correlation heatmap
correlation_matrix = dt1([['Total_Day_Minutes', 'Total_Evening_Minutes', 'Total_Night_Mi
                          .corr())
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

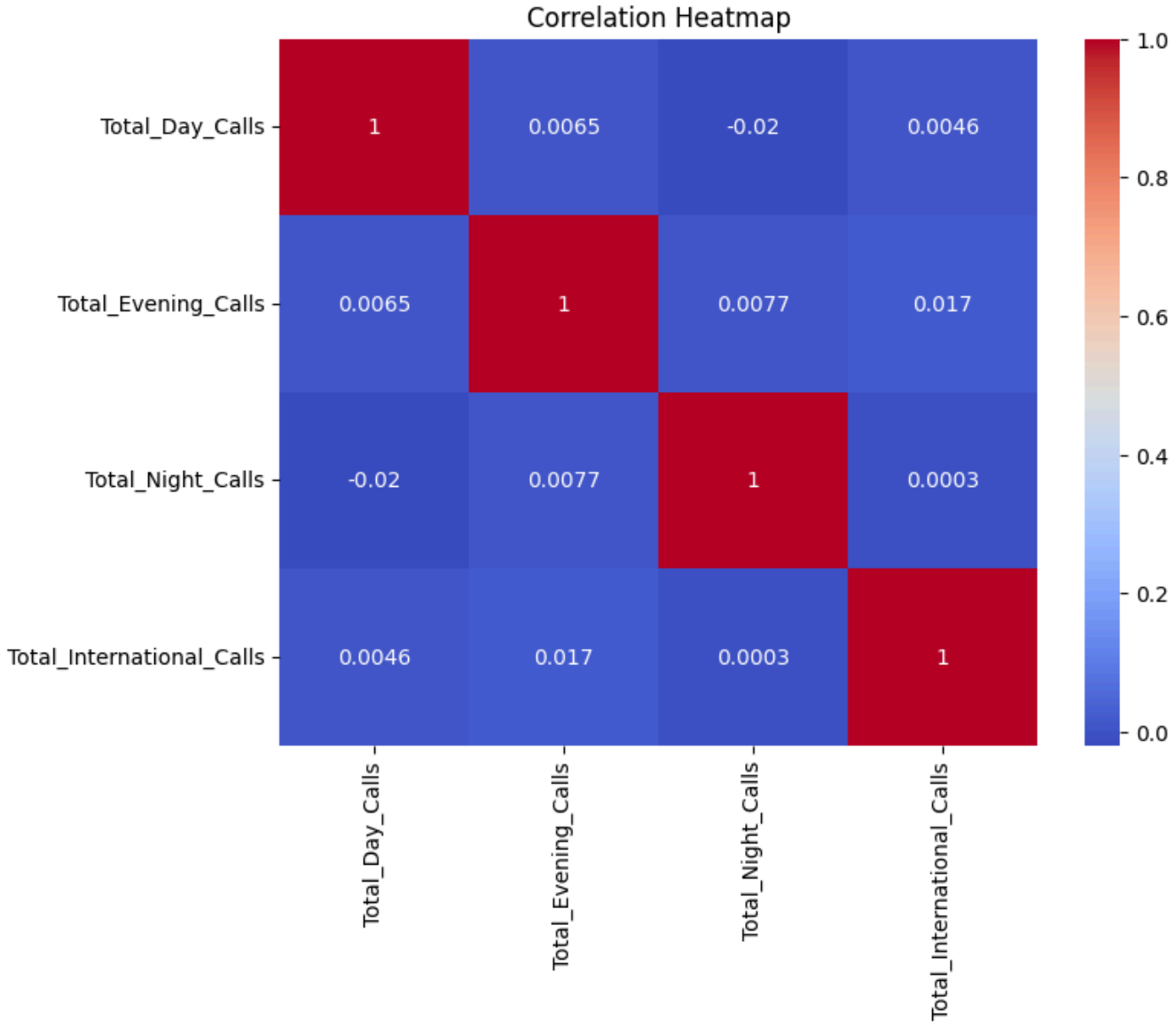


Observations

- **Weak or No Linear Correlation:** The most prominent observation is that all the correlation coefficients are very close to zero. The color intensity is very low, indicating a lack of strong linear relationships between any pair of variables.
- **Near-Zero Coefficients:** The values are all extremely close to zero, suggesting that there's virtually no linear relationship between any of the two components.
- **No Multicollinearity Concerns:** Since the correlations are so weak, multicollinearity is not a concern for these variables. You can safely include all of them in your model without worrying about instability or interpretability issues related to multicollinearity.

Correlation matrix for calls

In [40]:
```
# Correlation heatmap
correlation_matrix = dt1[['Total_Day_Calls', 'Total_Evening_Calls', 'Total_Night_Calls',
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

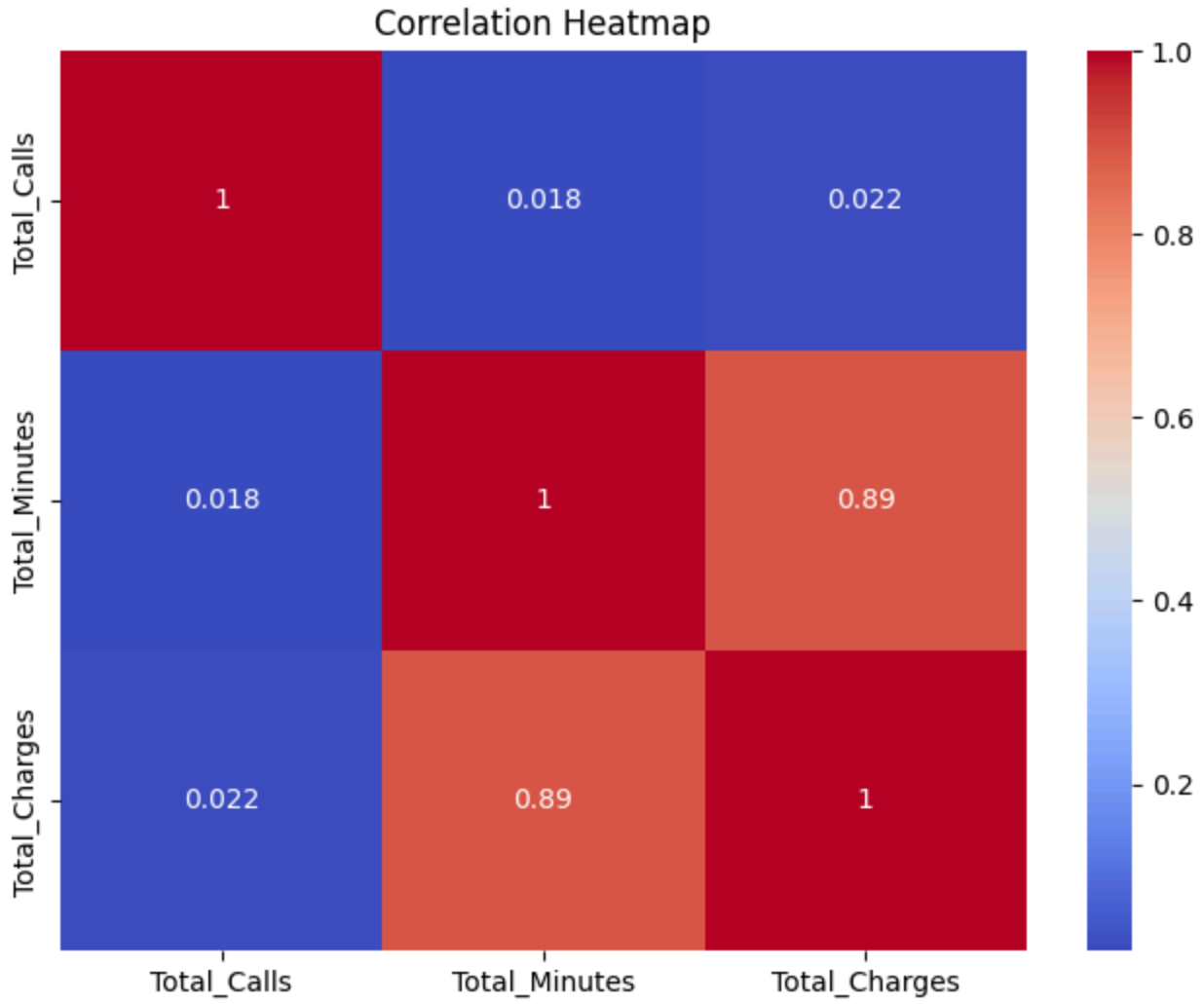


Observations:

- **Weak or No Linear Correlation:** The most prominent observation is that all the correlation coefficients are very close to zero. The color intensity is very low, indicating a lack of strong linear relationships between any pair of variables.
- **Near-Zero Coefficients:** The values are all extremely close to zero, suggesting that there's virtually no linear relationship between any two charge components.

Correlation matrix between total calls, minutes and charges

In [41]:
```
# Correlation heatmap
correlation_matrix = dt1[['Total_Calls', 'Total_Minutes', 'Total_Charges', ]].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



Observations:

- **Strong Positive Correlation Between Total Minutes and Total Charges:** The most prominent feature is the strong positive correlation (0.89) between Total_Minutes and Total_Charges. This is represented by the intense red color in that cell. It confirms what we saw in the scatter plot: customers who use more minutes tend to have higher charges.
- **Weak Positive Correlation Between Total Calls and Total Minutes/Charges:** The correlations between Total_Calls and both Total_Minutes (0.018) and Total_Charges (0.022) are very weak and positive. The blue color of these cells indicates the near-zero correlation. This aligns with our earlier observation that the number of calls alone doesn't strongly predict total minutes or charges.
- **No Multicollinearity Issues:** While Total_Minutes and Total_Charges are highly correlated, this is not necessarily a multicollinearity problem in the typical sense. Multicollinearity usually refers to having multiple predictor variables that are highly correlated. In this case, we are just examining the

relationships between key usage metrics, and the high correlation between minutes and charges is expected

4.5 Data Preprocessing

In this stage I will convert my data into a suitable format for the model. Such as:

- **Encoding Categorical Variables:** Machine learning models typically work with numerical data.
- **Scaling of the features:** Scaling features is a crucial preprocessing step in many machine learning workflows, ensuring that models perform well, converge faster, and provide meaningful interpretations.However tree-based models (e.g., Decision Trees, Random Forests) are generally less sensitive to the scale of features because they split the data based on feature thresholds rather than distances.

Feature engineering was done before EDA

In [109]:
```
# Importing our libraries for both preprocessing and modeling
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc
from mpl_toolkits.mplot3d import Axes3D
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
```

In [43]:
```
#Creating a copy
dt2 = dt1.copy(deep=True)
dt2.head()
```

Out[43]:

	Account_Length	Area_Code	Phone_Number	International_Plan	Voice_Mail_Plan	Voice_Mail_M
0	128	415	3824657	no	yes	
1	107	415	3717191	no	yes	
2	137	415	3581921	no	no	
3	84	408	3759999	yes	no	
4	75	415	3306626	yes	no	

5 rows × 25 columns

In [44]:
```
#Changing the churn category from bool to interger
dt2['Churn']=dt2['Churn'].astype(int)

#Confirm changes
dt2['Churn'].value_counts()
```

Out[44]:
```
Churn
0    2850
1     483
Name: count, dtype: int64
```

In [45]:
```
#Changing the other categorical variable to numerical
#Use label encoding
encoder = LabelEncoder()
dt2['International_Plan'] = encoder.fit_transform(dt2['International_Plan'])
dt2['Voice_Mail_Plan'] = encoder.fit_transform(dt2['Voice_Mail_Plan'])
dt2['Customer_Service_Category'] = encoder.fit_transform(dt2['Customer_Service_Category'
```

In [46]:
```
#Confirm changes done by the labelencoder
print(dt2['International_Plan'].value_counts())
print(dt2['Voice_Mail_Plan'].value_counts())
print(dt2['Customer_Service_Category'].value_counts())
```

```
International_Plan
0    3010
1     323
Name: count, dtype: int64
Voice_Mail_Plan
0    2411
1     922
Name: count, dtype: int64
Customer_Service_Category
1    3232
0     101
Name: count, dtype: int64
```

In [47]:
```
#Separating the features from the target
X = dt2.drop('Churn',axis=1)
y = dt2['Churn']
```

In [48]:
```
#Confirm changes
print(X.head())
print(y.head())
```

```
   Account_Length  Area_Code  Phone_Number  International_Plan  \
0             128        415       3824657                   0
1             107        415       3717191                   0
2             137        415       3581921                   0
3              84        408       3759999                   1
4              75        415       3306626                   1

   Voice_Mail_Plan  Voice_Mail_Messages  Total_Day_Minutes  Total_Day_Calls  \
0                1                   25              265.1              110
1                1                   26              161.6              123
2                0                    0              243.4              114
3                0                    0              299.4               71
4                0                    0              166.7              113

   Total_Day_Charge  Total_Evening_Minutes  ...  Total_Night_Charge  \
0             45.07                  197.4  ...               11.01
1             27.47                  195.5  ...               11.45
2             41.38                  121.2  ...                7.32
3             50.90                   61.9  ...                8.86
4             28.34                  148.3  ...                8.41

   Total_International_Minutes  Total_International_Calls  \
0                         10.0                          3
1                         13.7                          3
2                         12.2                          5
3                          6.6                          7
4                         10.1                          3

   Total_International_Charge  Customer_Service_Calls  Total_Minutes  \
0                        2.70                       1          717.2
1                        3.70                       1          625.2
2                        3.29                       0          539.4
3                        1.78                       2          564.8
4                        2.73                       3          512.0

   Total_Calls  Total_Charges  Avg_Call_Duration  Customer_Service_Category
0          303          75.56           2.366997                          1
1          332          59.24           1.883133                          1
2          333          62.29           1.619820                          1
3          255          66.80           2.214902                          1
4          359          52.09           1.426184                          1

[5 rows x 24 columns]
0    0
1    0
2    0
3    0
4    0
Name: Churn, dtype: int32
```

In [49]:
```
#Split the data into train and test using train_test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

#scaling
#Initialize the scaler
scale = StandardScaler()

#Fit and transform the features
X_train_scaled = scale.fit_transform(X_train)
X_test_scaled = scale.transform(X_test)
```

5 Modeling

I will now begin the modeling process. My main aim develop multiple models and assess the best model for this classification problem based on the performance of each model. To achieve this the steps are as follows:

Models to build:

- Logistic regression model
- Decision tree classifier
- Random Forest Classifier
- XGBoost Classifier

To improve some of the models' performance some of the techniques to be used include:

- Synthetic Minority Oversampling Technique(Handles class imbalance)
- Class weighting with scikit learn(Handles class imbalance)
- Hyperparameter using using GridSearch and RandomizedSearch(Improves the model performance based on the key metric such as recall)

For evaluation process, the evaluation metric are

- **Recall(priority):** It indicates what percentage of the classes to be analyzed were actually captured by the model.
- **F1-score:** Harmonic mean of precision and recall
- **Precision:** Measures how many predicted positives are actually positive
- **Accuracy:** Proportion of correctly classified instances out of the total

Visualizations to be used for the model analysis:

- **Confusion matrix:** It is a table used to evaluate the performance of a classification model by comparing predicted labels against actual (true) labels. It provides a detailed breakdown of correct and incorrect predictions, making it a valuable tool for understanding model performance.
- **ROC and AUC curves: Receiver Operator Characteristic Curve:** Illustrates the true positive rate against the false positive rate of the classifier

Area Under the Curve: Singular value summarizing the ROC curve.A higher AUC indicates better performance

5.1 Base Logistic Regression Model

In [50]:
```
#Building the base model
logreg_base = LogisticRegression(random_state=42)

#Train the model
logreg_base.fit(X_train_scaled,y_train)
```

Out[50]:
```
LogisticRegression
(https://scikit-learn.org/1.6/modules/generated/sklearn.linear_model.LogisticRegre
LogisticRegression(random_state=42)
```

In [51]:
```
#Make prediction
y_pred_base = logreg_base.predict(X_test_scaled)
```

5.2 Evaluating the base model

In [52]:
```
#Evaluating the base model

#Check the accuracy
accuracy_base = accuracy_score(y_test,y_pred_base)*100
print(f"Logistic Regression base model accuracy :{accuracy_base:.2f}%")

#Classification report
print("Classification Report:\n",classification_report(y_test,y_pred_base))
```

```
Logistic Regression base model accuracy :85.76%
Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.97      0.92       566
           1       0.58      0.21      0.31       101

    accuracy                           0.86       667
   macro avg       0.73      0.59      0.61       667
weighted avg       0.83      0.86      0.83       667
```

Observations:

High Overall Accuracy (86%)

The model correctly predicts a large portion of customers, but accuracy alone does not tell the full story due to class imbalance.

Low Precision & Recall for Churned Customers

Precision for churned customers (0.58) means that when the model predicts a customer will churn, it's correct only 58% of the time. Recall for churned customers (0.21) is extremely low, meaning the model fails to detect 79% of actual churned customers.

Strong Performance for Non-Churned Customers

Precision (0.87) and Recall (0.97) for Not Churned customers indicate the model does well at identifying loyal customers. However, the model is too biased towards predicting customers as non-churned, leading to poor churn detection.

Reason for this observations:

- Class imbalance on the target variable (Churn)
- Since Logistic Regression does not naturally handle imbalance, it favors the majority class (non-churned).

Findings based on the observation:

1. Why is accuracy high?

The high accuracy of the base model can is misleading, due to the class imbalance. In many churn prediction scenarios, a large portion of customers are non-churned (retained), leading to a scenario where simply predicting "not churned" for every instance can yield high accuracy. However, this may not truly reflect the model's ability to predict the churned class effectively, which is our area of interest.

2. Why is the recall on the "not churned" class high and on the "churned" class low?

The recall for the "not churned" class being high and the recall for the "churned" class being low is being attributed by the class imbalance. Since the majority of customers are non-churned, the model is biased towards predicting "not churned," leading to high recall for that class but poor recall for the churned class. This indicates that the model is underperforming in predicting churned customers, which is the focus of our business problem.

3. Why is recall crucial for this case?

In churn prediction, recall is especially important because it measures how well the model identifies the customers who are likely to churn. The main goal in this business problem is to target those high-risk customers with retention strategies, so missing out on them (false negatives) can result in lost opportunities to prevent churn. High recall ensures that most of the churned customers are correctly identified, allowing the company to take proactive measures before they leave. Although accuracy is a useful metric, recall is more critical in scenarios where predicting the positive class (churned) is the key objective.

4. Why is the F1-score for the churned cases low?

The F1-score is low for churned cases because it balances precision and recall. The model struggles to identify churned customers accurately, leading to a lower recall (the ability to correctly identify churned customers) and precision (the ability to avoid false positives). This is because the churned customers are imbalanced compared to the non-churned ones, causing the model to be more biased towards predicting non-churned customers, leading to lower performance on churned cases.

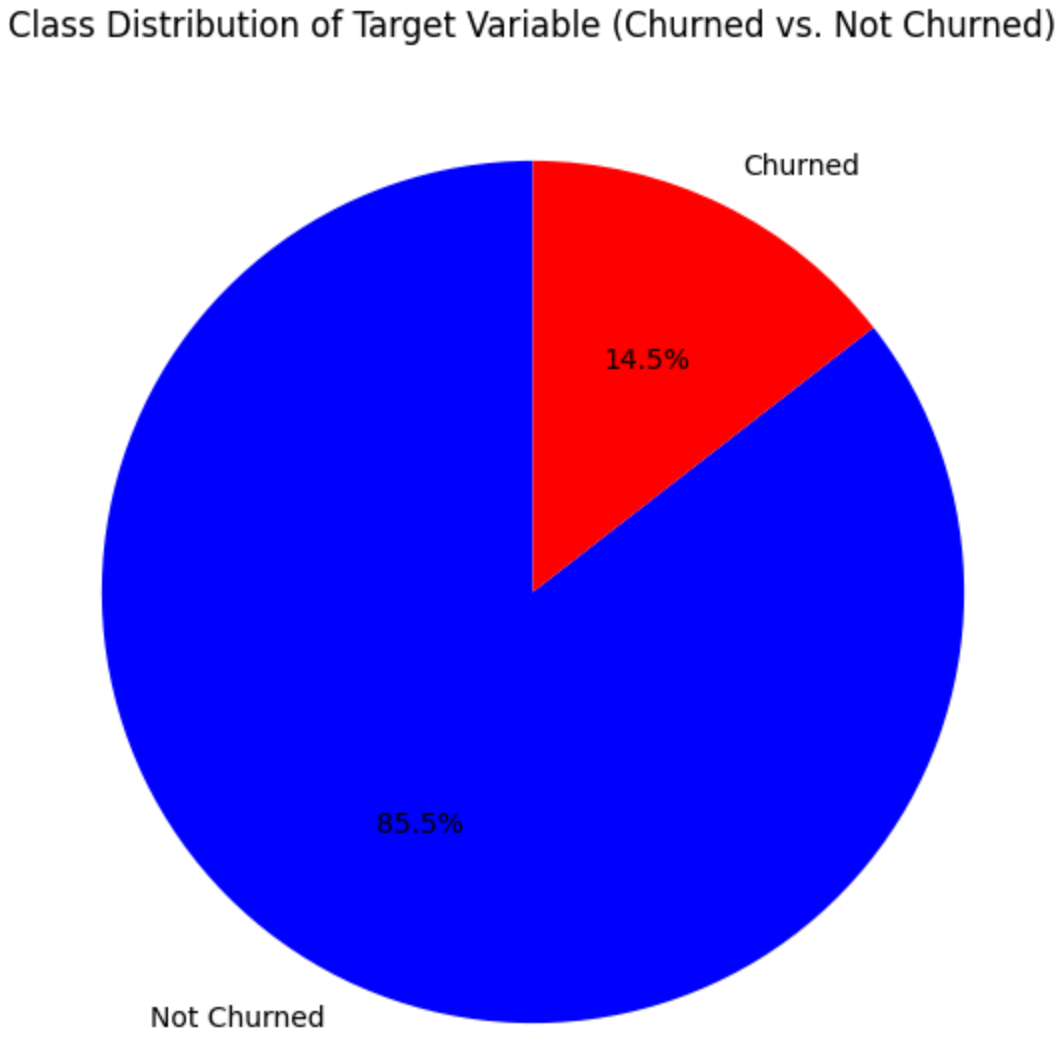
5. Why is F1-score crucial for this case?

Identifying churned customers early is vital for customer retention strategies, such as offering promotions or support. A model with a low F1-score for churned customers could result in missed opportunities to retain valuable customers, leading to revenue loss and a lack of effective business interventions.

6. Analyzing the class imbalance:

We will now visualize the class distribution in our dataset using a pie chart. This will help us better

In [53]:
```
# Sample class distribution (modify based on your actual data)
class_counts = y.value_counts()

# Plotting the pie chart
plt.figure(figsize=(7,7))
plt.pie(class_counts, labels=['Not Churned', 'Churned'], autopct='%1.1f%%', startangle=9
plt.title('Class Distribution of Target Variable (Churned vs. Not Churned)')
plt.show()
```



This is a clear observation of class imbalance. The "Not Churned" category represents a substantially larger portion of the data (85.5%) compared to the "Churned" category (14.5%).

7. How has the Class Imbalance affected the Business Problem?

The class imbalance issue significantly impacts our business problem. Predicting the churned customers accurately is critical to taking proactive actions, such as targeting retention campaigns or providing personalized offers. If our model predominantly predicts "not churned" customers, we risk missing out on identifying the high-value customers who are most likely to churn. This can lead to missed opportunities in customer retention and negatively impact revenue.

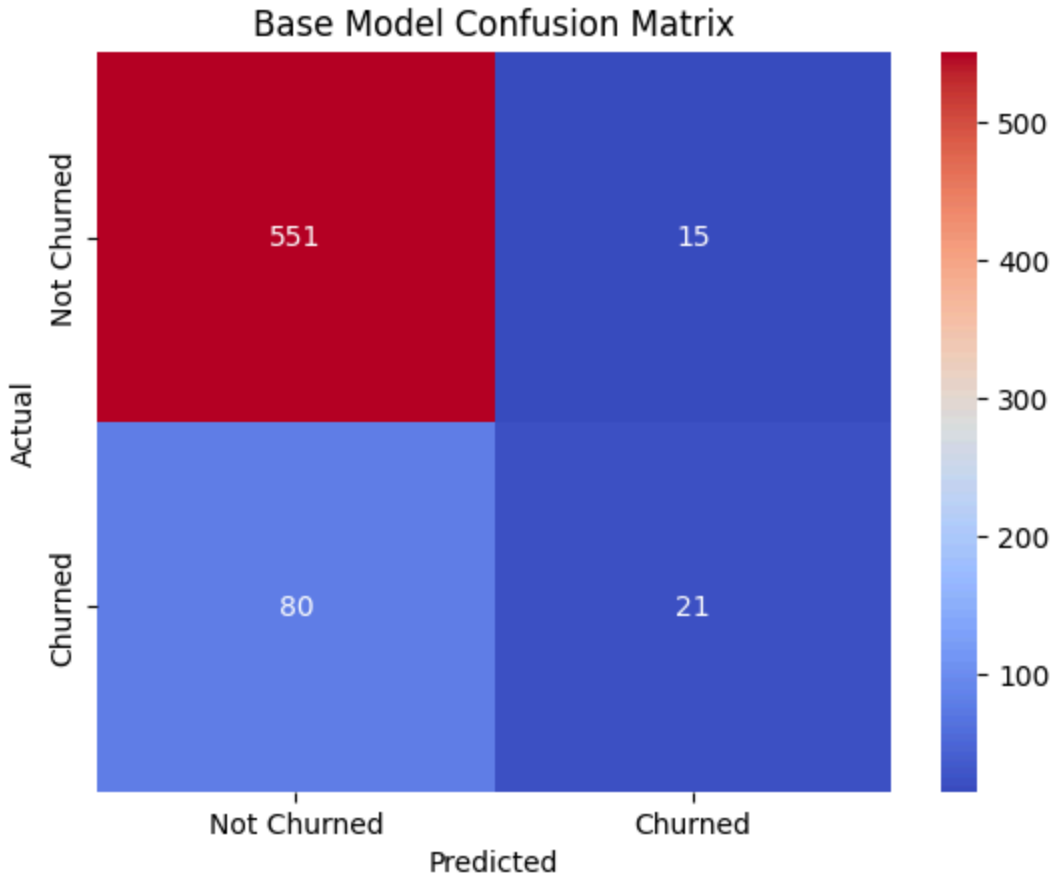
8. What is the solution?

- Using the **Synthetic Minority Over-sampling Technique**

The Synthetic Minority Over-sampling Technique (SMOTE) is a widely used method to tackle class imbalance. SMOTE works by generating synthetic examples for the minority class (in our case, churned customers) rather than under-sampling the majority class. This helps the model learn better decision boundaries, improving the performance on the minority class without losing information from the majority class. By applying SMOTE, we aim to improve recall for the churned class, and ultimately, the model's ability to predict churned customers more effectively.

5.2.1 Plotting the base model confusion matrix

In [54]:
```
#Plot the confusion matrix
# Confusion Matrix
confusion_base = confusion_matrix(y_test, y_pred_base)
sns.heatmap(confusion_base, annot=True, fmt='d', cmap='coolwarm', xticklabels=['Not Chur
            yticklabels=['Not Churned', 'Churned'])
plt.title('Base Model Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Confusion Matrix analysis

- Here will analyze the distribution of the True Positives(TP), False Positives(FP), True Negatives(TN) and False Negatives(FN)

Observations:

High Number of True Negatives: The model correctly predicted "False" (Not Churned) for 551 customers. This is the largest count in the matrix.

Low Number of True Positives: The model only correctly predicted "True" (Churned) for 21 customers. This is a very low number, suggesting the model struggles to identify churned customers effectively.

Relatively Low Number of False Positives: The model incorrectly predicted "True" (Churned) for 15 customers. This is a relatively low number, which is good.

High Number of False Negatives: The model incorrectly predicted "False" (Not Churned) for 80 customers who actually churned. This is a significant number and a major concern. It means the model is missing a large portion of the customers who are actually churning.

Overall Assessment:

- The base model, while having a seemingly high accuracy, is actually performing poorly at identifying churned customers. This is evidenced by the very low recall and F1-score for the "True" class. The model is heavily biased towards predicting "False" due to the class imbalance.

5.3 Addressing the class imbalance using Synthetic Minority Oversampling Technique(SMOTE)

Synthetic Minority Over-sampling Technique, is a powerful technique for addressing class imbalance in machine learning. In our dataset we noticed biasness towards the majority class which is not churned category. Hence will use the SMOTE technique to correct this trade-off and capture accurate unseen data for the minority class which is the churned category.

In [55]:
```
#Using the SMOTE Technique
# call SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the training data
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)
```

In [56]:
```
# Check the new class distribution
print('Class distribution after SMOTE:')
print(pd.Series(y_train_smote).value_counts())
```

```
Class distribution after SMOTE:
Churn
0    2284
1    2284
Name: count, dtype: int64
```

After applying SMOTE it is evident that from the observations above both classes are equal, meaning the class imbalance issue has been address. We will now create a new model with the SMOTE and compare its performance to the previous base model.

In [57]:
```
# call the Logistic Regression model
smote_model = LogisticRegression(random_state=42)

# Train the model on the SMOTE data
smote_model.fit(X_train_smote, y_train_smote)

# Make predictions
y_pred_smote =  smote_model.predict(X_test_scaled)
```

5.4 Evaluating the SMOTE model

In [58]:
```
# Accuracy
accuracy_smote = accuracy_score(y_test, y_pred_smote)*100
print(f'SMOTE Model Accuracy: {accuracy_smote:.2f}%')

# Classification Report
print('SMOTE Model Classification Report:')
print(classification_report(y_test, y_pred_smote))
```

```
SMOTE Model Accuracy: 79.16%
SMOTE Model Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.80      0.87       566
           1       0.40      0.74      0.52       101

    accuracy                           0.79       667
   macro avg       0.67      0.77      0.69       667
weighted avg       0.86      0.79      0.81       667
```

Observations and changes after applying SMOTE:

Applying SMOTE (Synthetic Minority Over-sampling Technique) altered the class distribution, leading to changes in the model's performance metrics:

1. Accuracy Drop (86% → 79%)

Before SMOTE, the model was likely biased toward predicting the majority class (not churned) because of the class imbalance. This resulted in high accuracy but poor recall for churned customers.After SMOTE, the dataset became more balanced, forcing the model to learn patterns for churned customers. This increased misclassifications for the previously well-learned majority class, reducing overall accuracy.

2. Changes in Precision, Recall, and F1-Score for Not Churned Customers

Precision increased (0.87 → 0.95): The model became more conservative in predicting "not churned," meaning when it does predict "not churned," it's more likely correct.

Recall decreased (0.97 → 0.80): The model now misclassifies more actual "not churned" customers as "churned." This happens because, after SMOTE, the model focuses more on detecting churn cases.

F1-score dropped (0.92 → 0.87): Since recall dropped significantly, the balance between precision and recall resulted in a lower F1-score.

3. Changes in Precision, Recall, and F1-Score for Churned Customers

Precision decreased (0.58 → 0.40): The model now produces more false positives (predicting churn for customers who won't actually churn). This happens because the newly generated synthetic churned cases introduce some noise.

Recall increased (0.21 → 0.74): The model now correctly identifies more churned customers because SMOTE helped balance the dataset, allowing the model to learn churn patterns better.

F1-score increased (0.31 → 0.52): Since recall improved significantly, the F1-score also improved, showing that the model is better at identifying churned customers overall.

Justification for These Changes

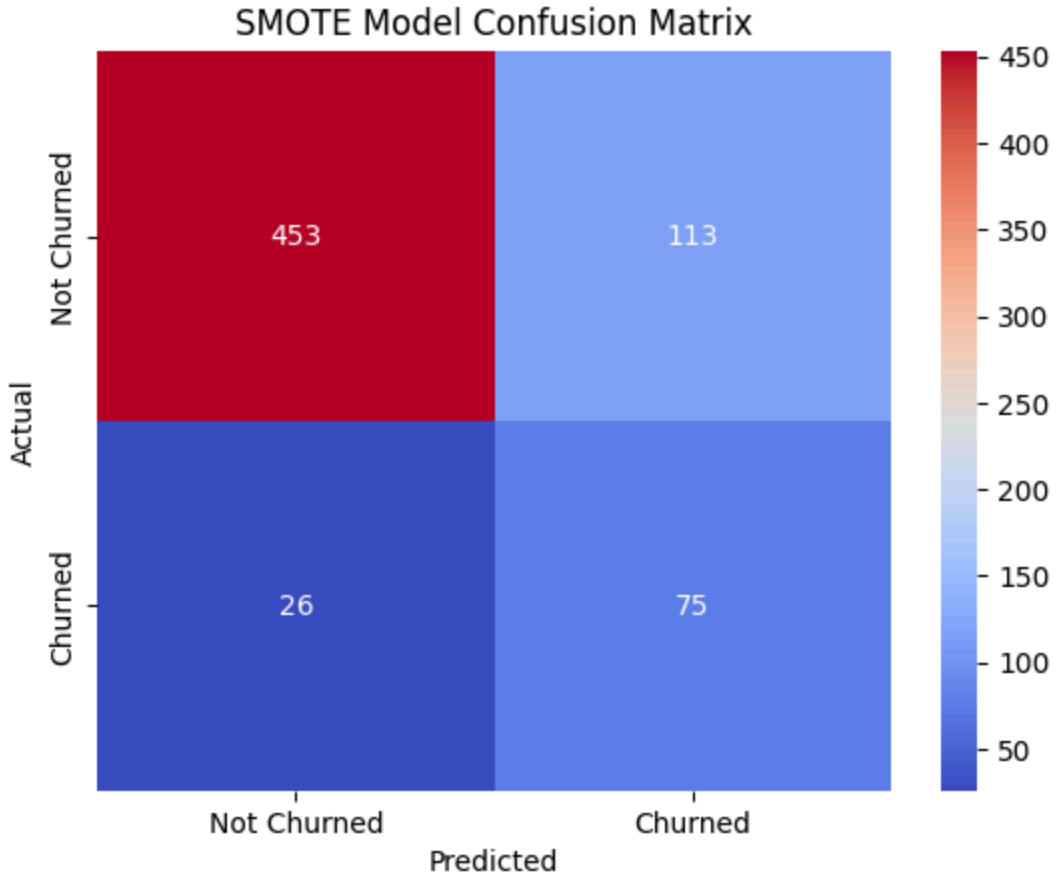
Pre-SMOTE: The model was heavily biased toward the majority class (not churned), achieving high accuracy but failing to capture churned customers.

Post-SMOTE: The model became better at detecting churned customers (higher recall), but at the cost of more false positives (lower precision) and slightly weaker performance on non-churned customers.

Trade-off: This trade-off is expected when addressing class imbalance. While overall accuracy dropped, the model now serves the business goal better by improving its ability to detect churned customers, which is

5.4.1 Plotting the SMOTE Confusion Matrix

In [59]:
```
# Confusion Matrix
confusion_smote = confusion_matrix(y_test, y_pred_smote)
sns.heatmap(confusion_smote, annot=True, fmt='d', cmap='coolwarm', xticklabels=['Not Chu
            yticklabels=['Not Churned', 'Churned'])
plt.title('SMOTE Model Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Confusion matrix comparisons:

1. True Positives (TP):

- Base Model: 21
- SMOTE Model: 75

Improvement: The SMOTE model significantly increased the number of correctly identified churned customers (from 21 to 75). This is a major advantage as it directly addresses the initial model's weakness in detecting churn.

2. False Positives (FP):

- Base Model: 15
- SMOTE Model: 113

Trade-off: The SMOTE model also increased the number of false positives (from 15 to 113). This is a trade-off. While we're catching more churned customers, we're also incorrectly flagging more non-churned customers as churned.

3. False Negatives (FN):

- Base Model: 80
- SMOTE Model: 26

Improvement: The SMOTE model drastically reduced the number of false negatives (from 80 to 26). This is a substantial advantage, as it means we're missing far fewer actual churners.

4. True Negatives (TN):

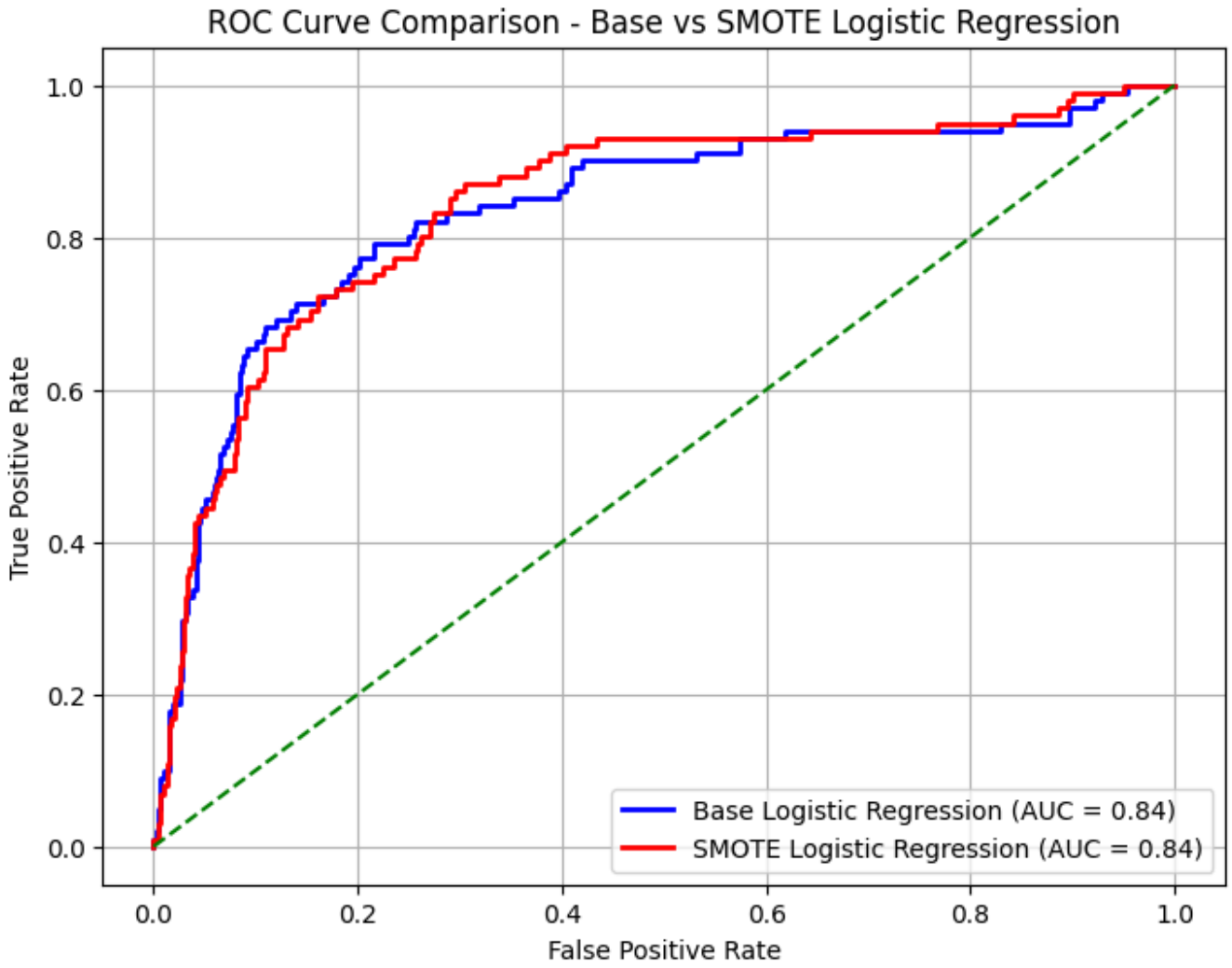
- Base Model: 551
- SMOTE Model: 453

Trade-off: The number of correctly identified non-churned customers decreased (from 551 to 453). This is a

5.4.2 ROC-AUC Analysis

In [60]:

```
# Get predicted probabilities for both models (base and SMOTE) for the positive class (Cl
y_prob_logreg = logreg_base.predict_proba(X_test_scaled)[:, 1]  # Base model probabiliti
y_prob_smote = smote_model.predict_proba(X_test_scaled)[:, 1]  # SMOTE model probabiliti

# Calculate the False Positive Rate (FPR) and True Positive Rate (TPR) for both models
fpr_logreg, tpr_logreg, _ = roc_curve(y_test, y_prob_logreg)
fpr_smote, tpr_smote, _ = roc_curve(y_test, y_prob_smote)

# Calculate the Area Under the Curve (AUC) for both models
roc_auc_logreg = auc(fpr_logreg, tpr_logreg)
roc_auc_smote = auc(fpr_smote, tpr_smote)

# Plot ROC curves for both models
plt.figure(figsize=(8, 6))

# ROC curve for the base logistic regression model
plt.plot(fpr_logreg, tpr_logreg, color='blue', lw=2, label=f'Base Logistic Regression (A

# ROC curve for the SMOTE model
plt.plot(fpr_smote, tpr_smote, color='red', lw=2, label=f'SMOTE Logistic Regression (AUC

# Diagonal line for random classifier (AUC = 0.5)
plt.plot([0, 1], [0, 1], color='green', linestyle='--')

# Add labels, title, and legend
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison - Base vs SMOTE Logistic Regression')
plt.legend(loc='lower right')
plt.grid(True)

# Show the plot
plt.show()
```



Comparison between the Base logistic regression model and the SMOTE model

Key Observations

1. Identical AUC (0.84): Both the base logistic regression model (blue line) and the SMOTE-enhanced logistic regression model (red line) have the same AUC of 0.84. This indicates that, overall, both models have a similar ability to distinguish between the two classes (churned vs. not churned).

2. Different Curve Shapes: While the AUC is the same, the shapes of the ROC curves are different, revealing important distinctions in their performance.

3. SMOTE Model's Initial Steepness: The SMOTE model's ROC curve rises more steeply at the beginning (lower FPR region) compared to the base model. This suggests that, initially, the SMOTE model can capture a larger proportion of true positives (churned customers) with a smaller trade-off in false positives.

4. Base Model's Gradual Rise: The base model's ROC curve rises more gradually. This implies that it might be more conservative in its positive predictions, leading to fewer false positives initially, but also missing out on some true positives.

5. Potential Trade-offs: The point where the curves intersect and diverge slightly indicates the trade-offs in terms of FPR and TPR at different thresholds. Depending on where you set your classification threshold, you might favor one model over the other.

Interpretation:

SMOTE's Benefit: The SMOTE technique appears to have improved the model's ability to identify true positives (churners) at lower FPRs, which is a valuable improvement.

Conclusion

- In our Syriatel churn prediction analysis, we evaluated a base model and a model trained on a dataset balanced using SMOTE.

Base Model Performance

- The model achieved 86% accuracy, but it was heavily biased toward the majority class (not churned).
- It had high precision (0.87) and recall (0.97) for not churned customers, but poor recall (0.21) for churned customers, meaning it failed to detect most churned customers.
- The low F1-score (0.31) for churned cases indicated that the model was not effective in identifying potential churners, which is critical for the business.

SMOTE-Applied Model Performance

- After applying SMOTE, the overall accuracy dropped to 79%, as the model now focused more on both classes instead of favoring the majority.
- The recall for churned customers significantly improved (0.21 → 0.74), meaning the model became much better at identifying customers who are likely to churn.
- However, precision for churned customers decreased (0.58 → 0.40), leading to more false positives (incorrectly predicting churn).
- The F1-score for churned customers increased from 0.31 to 0.52, demonstrating an overall improvement in identifying churned cases.

Business Implications

- The base model was highly accurate overall but ineffective at detecting churn, which is the key problem we aim to solve.
- The SMOTE-applied model made a trade-off by sacrificing some accuracy to improve the recall of churned customers, making it more useful for customer retention strategies.
- Despite the increase in false positives, the SMOTE model ensures that more actual churners are identified, allowing the company to take proactive measures (e.g., targeted offers or customer service interventions).However, the SMOTE model introducing more false positives, has caused inconvenience to loyal customers.
- To correct this trade-off and to avoid the false positives being costly, the model can be improved by fine tuning

Next steps:

- To further refine the model, we could explore techniques like cost-sensitive learning, ensemble methods, or hyperparameter tuning to improve precision while maintaining high recall for churned customers.

5.5 Hyper-parameter Tuning using GridSearchCV

Why is Tuning the SMOTE Model Important?

- Tuning the SMOTE-applied model is crucial because oversampling changes the data distribution, which can impact how well the model generalizes to unseen data. Here’s why tuning is necessary:

1. SMOTE Changes Decision Boundaries:

- By generating synthetic churned samples, SMOTE modifies the decision boundary of the classifier.
- If the model is not tuned properly, it may overfit to synthetic data or underfit by treating all churned cases as similar.

2. Avoiding Overfitting to Synthetic Data

- Too much regularization (L1/L2) might suppress useful information in the generated data.
- Too little regularization can make the model memorize synthetic samples rather than generalizing to real churners.

3. Improving Precision-Recall Tradeoff

- The original model struggled with low recall for churned customers.
- After SMOTE, recall improves, but precision may drop (more false positives).
- Tuning helps find a balance where both precision and recall are optimized.

Why Should We Use GridSearchCV?

- GridSearchCV is used to systematically find the best hyperparameters. Here’s why it's the best approach:

1.Ensures the Best Hyperparameter Combination:

- Instead of manually testing different values, GridSearchCV automates the search across all possible combinations.
- It guarantees we find the best settings for C (regularization), penalty (L1/L2), and solver.

2.Uses Cross-Validation for Robust Results:

- It splits the training data into multiple folds (e.g., 5-fold CV) and tests each hyperparameter set on different subsets.
- This prevents the model from being overfitted to a specific train-test split.

3.Optimizes Based on Business Needs (F1-Score)

- We can set scoring='f1' so the search prioritizes maximizing F1-score, which balances precision and recall for churned customers.
- This is better than optimizing only for accuracy, which can be misleading in imbalanced datasets.

In [61]:
```
# Define hyperparameter grid
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Regularization strength
    'penalty': ['l1', 'l2'],  # Lasso (L1) and Ridge (L2)
    'solver': ['liblinear', 'saga']  # Compatible solvers for L1 & L2
}

# Initialize Logistic Regression model
logreg_grid = LogisticRegression(max_iter=1000, random_state=42)

# Perform Grid Search with 5-fold Cross Validation
grid_search = GridSearchCV(logreg_grid, param_grid, cv=5, scoring='f1', n_jobs=-1)
grid_search.fit(X_train_smote, y_train_smote)

# Display best parameters and best F1-score
print(f"Best Parameters:, {grid_search.best_params_}")
print(f"Best F1-Score from CV:, {grid_search.best_score_:.2f}")
```

```
Best Parameters:, {'C': 0.01, 'penalty': 'l1', 'solver': 'liblinear'}
Best F1-Score from CV:, 0.78
```

In [62]:
```
# Get the best model from GridSearchCV
tuned_smote_model = grid_search.best_estimator_

# Train the optimized model on SMOTE data
tuned_smote_model.fit(X_train_smote, y_train_smote)

# Make predictions
y_pred_tuned = tuned_smote_model.predict(X_test_scaled)
```

5.5.1 Evaluate the Tuned Model

In [63]:
```
# Compute performance metrics
accuracy_tuned = accuracy_score(y_test, y_pred_tuned) * 100
precision_tuned = precision_score(y_test, y_pred_tuned)
recall_tuned = recall_score(y_test, y_pred_tuned)
f1_tuned = f1_score(y_test, y_pred_tuned)

# Print results
print(f'Tuned Model Accuracy: {accuracy_tuned:.2f}%')
print(f'Precision: {precision_tuned:.4f}')
print(f'Recall: {recall_tuned:.4f}')
print(f'F1-Score: {f1_tuned:.4f}')

# Classification Report
print("\nTuned Model Classification Report:\n", classification_report(y_test, y_pred_tun
```

```
Tuned Model Accuracy: 78.71%
Precision: 0.4000
Recall: 0.8119
F1-Score: 0.5359

Tuned Model Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.78      0.86       566
           1       0.40      0.81      0.54       101

    accuracy                           0.79       667
   macro avg       0.68      0.80      0.70       667
weighted avg       0.87      0.79      0.81       667
```

Observations and changes after tuning the SMOTE model:

1. Accuracy:

- Untuned SMOTE model: Accuracy is 79.00%.
- Tuned SMOTE model: Accuracy is slightly lower at 78.71%.

Observation: The drop in accuracy is minimal and is expected due to the focus on improving recall for churned customers. Since accuracy measures overall correct predictions, a focus on recall may lead to a slight decrease in overall accuracy.

2. Precision:

- Untuned SMOTE model: Precision for churned customers is 0.40.
- Tuned SMOTE model: Precision remains the same at 0.40.

Observation: The precision for churned customers remains unchanged between the untuned and tuned models, meaning the model's tendency to incorrectly classify non-churned customers as churned has stayed the same. The key difference lies in the improvement in recall for churned customers.

3. Recall:

- Untuned SMOTE model: Recall for churned customers is 0.74.
- Tuned SMOTE model: Recall for churned customers increases to 0.81.

Observation: Recall improves in the tuned model from 0.74 to 0.81, meaning the model now identifies a higher proportion of churned customers. This is critical for churn prediction since we want to capture as many at-risk customers as possible.

4. F1-Score:

- Untuned SMOTE model: F1-score for churned customers is 0.52.
- Tuned SMOTE model: F1-score for churned customers improves to 0.54.

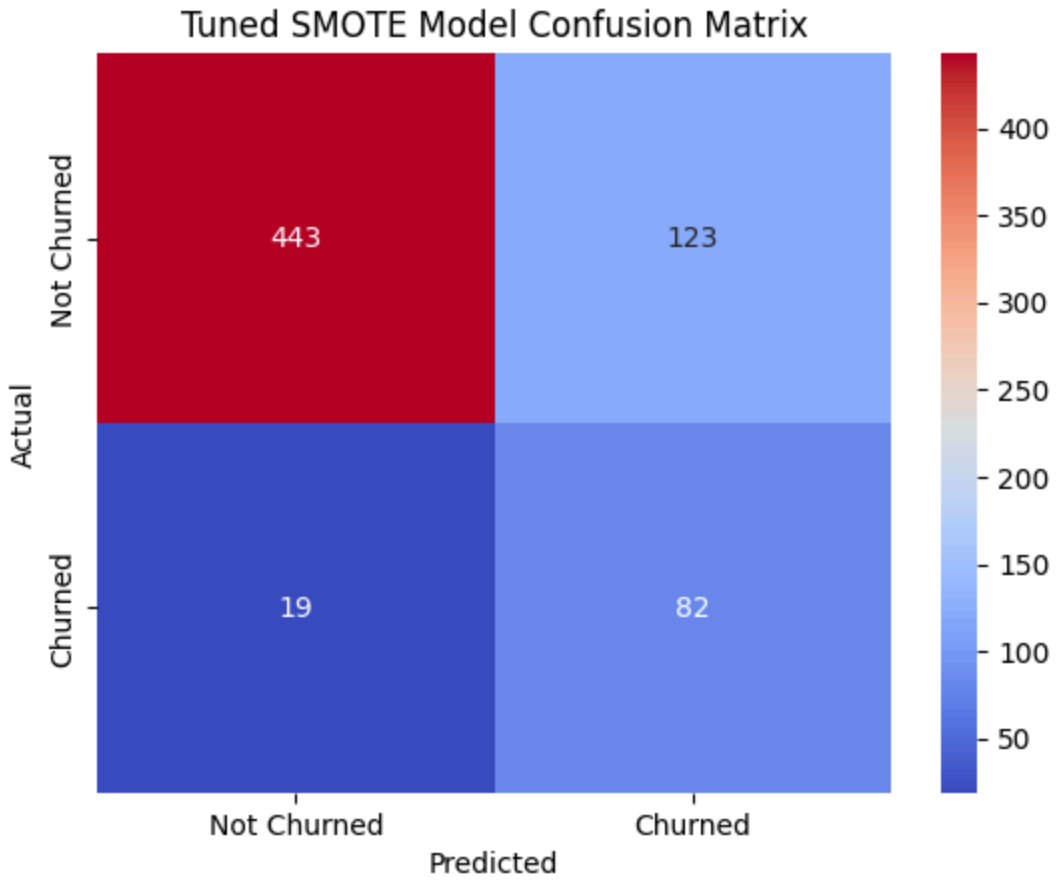
Observation: F1-score improves slightly from 0.52 to 0.54, showing that the balance between precision and recall has improved in the tuned model. Although precision has stayed the same, the increase in recall enhances the model's overall performance for churn prediction.

Summary of Differences:

- Recall for churned customers increased from 0.74 to 0.81, improving the model’s ability to correctly identify churned customers.
- Precision remained the same at 0.40 for both models, indicating that the false positive rate remains unchanged.
- F1-score improved slightly from 0.52 to 0.54, reflecting a more balanced performance in identifying churned customers.

5.5.1.1 Tuned Confusion Matrix

In [107]:
```
# Confusion Matrix
confusion_tuned = confusion_matrix(y_test, y_pred_tuned)
sns.heatmap(confusion_tuned, annot=True, fmt='d', cmap='coolwarm', xticklabels=['Not Chu
            yticklabels=['Not Churned', 'Churned'])
plt.title('Tuned SMOTE Model Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Confusion matrix comparison:

1. True Positives (TP):

- Untuned Model: 75
- Tuned Model: 82

Improvement: The tuned model correctly identified 7 more churned customers. This is a positive improvement, though not dramatic.

2. False Positives (FP):

- Untuned Model: 113
- Tuned Model: 123

Trade-off: The tuned model also increased the number of false positives by 10. This is a trade-off. We're catching slightly more churned customers, but at the cost of incorrectly flagging more non-churned customers.

3. False Negatives (FN):

- Untuned Model: 26
- Tuned Model: 19

Improvement: The tuned model reduced the number of false negatives by 7. This is a positive improvement, meaning we're missing slightly fewer actual churners.

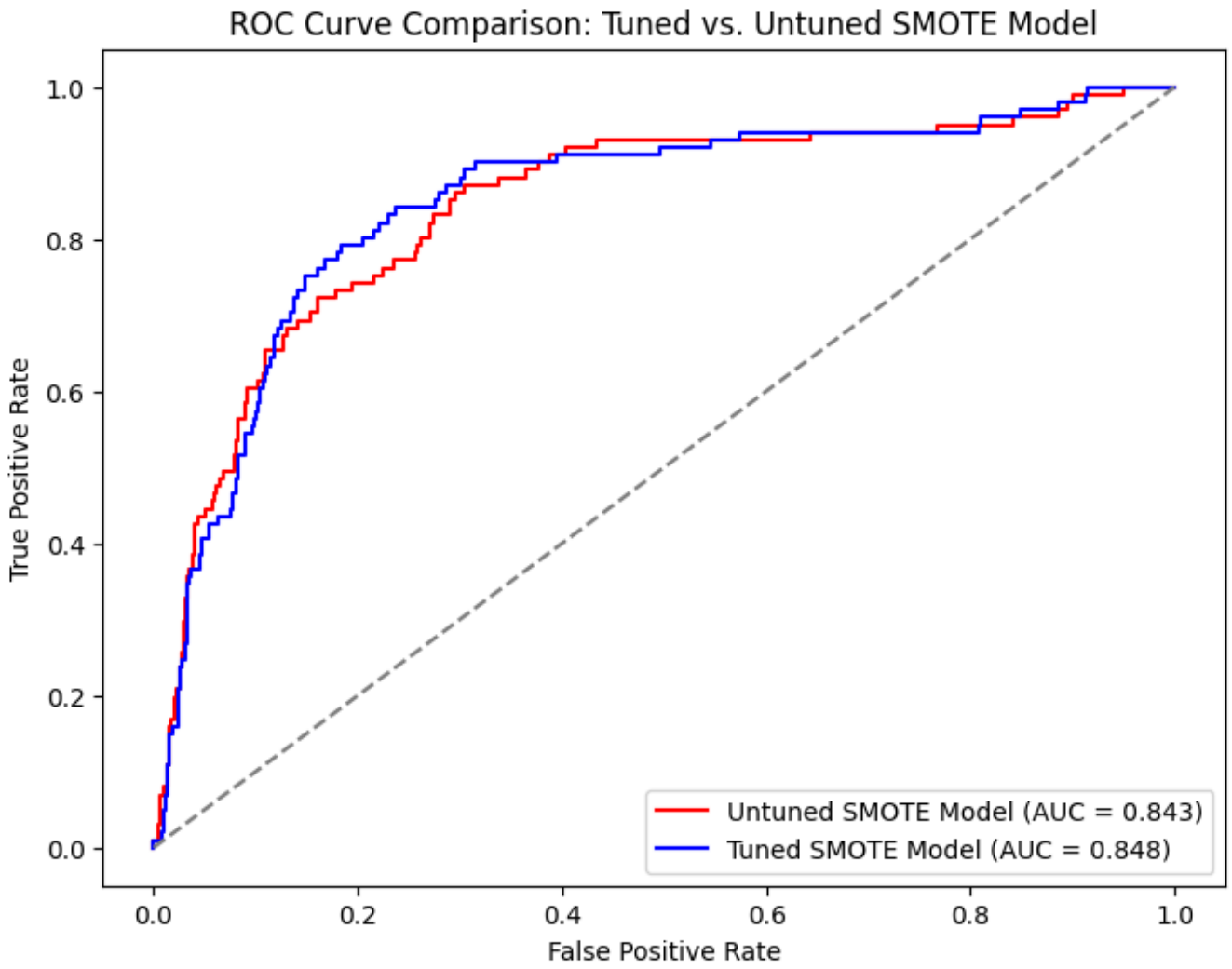
4. True Negatives (TN):

- Untuned Model: 453
- Tuned Model: 443

Trade-off: The number of correctly identified non-churned customers decreased by 10. This is a trade-off resulting from the model's increased focus on the minority class.

5.5.2 ROC-AUC comparison

In [65]:
```
# Get probability scores for the positive class
y_probs_untuned =smote_model.predict_proba(X_test_scaled)[:, 1]
y_probs_tuned = tuned_smote_model.predict_proba(X_test_scaled)[:, 1]

# Compute ROC curve and AUC
fpr_untuned, tpr_untuned, _ = roc_curve(y_test, y_probs_untuned)
fpr_tuned, tpr_tuned, _ = roc_curve(y_test, y_probs_tuned)

auc_untuned = auc(fpr_untuned, tpr_untuned)
auc_tuned = auc(fpr_tuned, tpr_tuned)

# Plot the ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_untuned, tpr_untuned, label=f'Untuned SMOTE Model (AUC = {auc_untuned:.3f})
plt.plot(fpr_tuned, tpr_tuned, label=f'Tuned SMOTE Model (AUC = {auc_tuned:.3f})', color

# Plot the diagonal line (random model)
plt.plot([0, 1], [0, 1], color='gray', linestyle='dashed')

# Labels and legend
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison: Tuned vs. Untuned SMOTE Model')
plt.legend(loc='lower right')
plt.show()
```



ROC Curve And AUC analysis:

Key Observations:

1. Slightly Higher AUC for Tuned Model: The tuned SMOTE model (blue line) has a slightly higher AUC of 0.848 compared to the untuned SMOTE model (green line) with an AUC of 0.843. This suggests that, overall, the tuned model has a marginally better ability to distinguish between the classes (churned vs. not churned).

2. Similar Overall Shape: Both curves have a similar shape, indicating that the general performance characteristics of the models are alike.

3. Tuned Model's Initial Steepness: The tuned model's curve shows a slightly steeper rise in the lower FPR region. This suggests that the tuned model might be able to capture a slightly larger proportion of true positives (churned customers) with a smaller trade-off in false positives, especially at lower thresholds.

4. Crossover at Higher FPR: There's a point where the curves cross over at higher FPR values. This indicates that the trade-offs in terms of FPR and TPR shift slightly at different thresholds. Depending on where you set your classification threshold, you might observe minor differences in performance.

Conclusion based on the logistic regression models:

- The tuned SMOTE model performs better in terms of recall, identifying a higher proportion of churned customers (0.81 vs. 0.74 in the untuned model).
- The F1-score improvement shows a better balance between precision and recall in the tuned model, making it slightly more effective overall for churn prediction.
- Although accuracy dropped slightly, this is typical when the model prioritizes recall over overall accuracy, which is important for churn prediction.
- The tuned SMOTE model is more sensitive to churners and should help businesses better identify customers at risk of leaving.
- The tuned SMOTE model shows a slight improvement in capturing churned customers at the expense of more false positives.

Which is the preferred logistic model? The best model is the tuned logistic regression model

Reasons: 1. Higher Recall for Churned Customers (0.81 vs. 0.74)

The tuned model correctly identifies 81% of actual churned customers, compared to 74% in the untuned model. Why this matters: Recall is the most critical metric for churn prediction. A business needs to identify as many churned customers as possible to take action.

2. Better F1-Score for Churned Class (0.54 vs. 0.52)

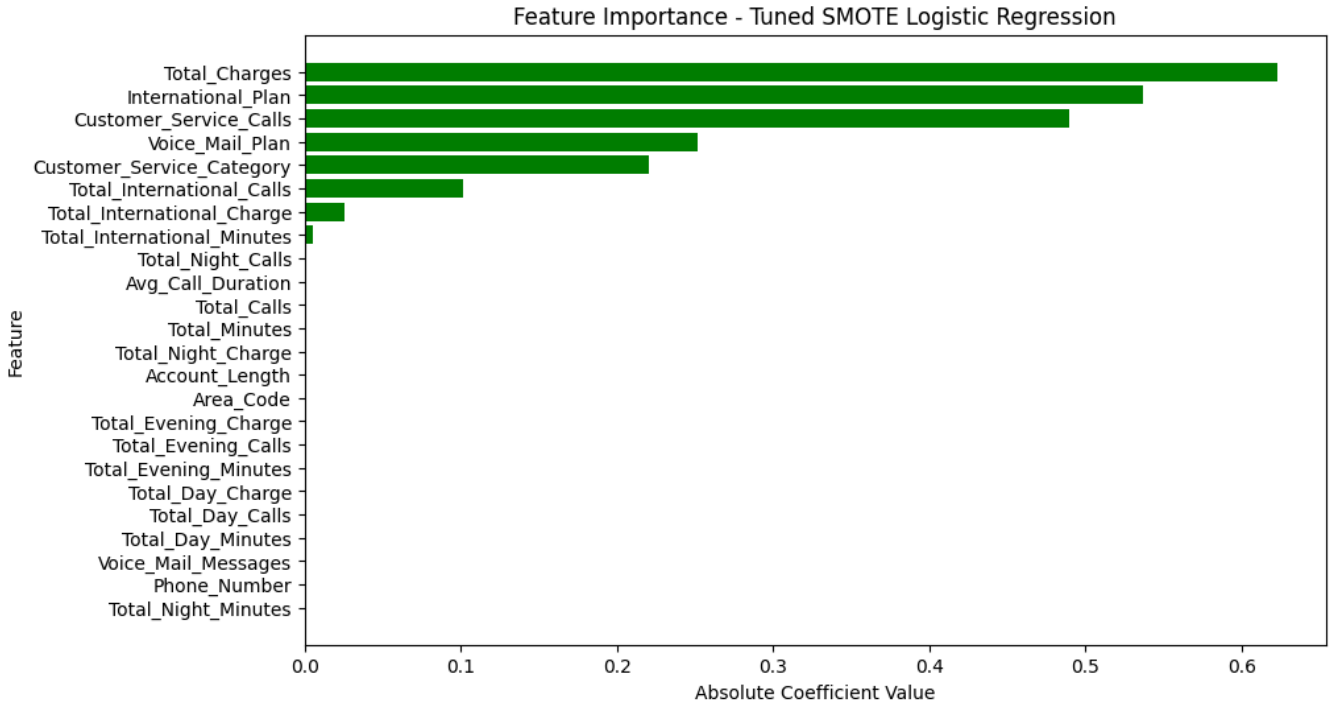
Since F1-Score balances precision and recall, the improvement means the model is making more reliable churn predictions.

3. Slightly Improved Precision for Non-Churned Class (0.96 vs. 0.95)

5.5.3 Feature Importance:

In [66]:

```
# Extract feature importance
feature_importance = np.abs(tuned_smote_model.coef_[0])

# Ensure feature names are available
if isinstance(X_train, pd.DataFrame):
    feature_names = X_train.columns.tolist()
else:
    raise ValueError("Original feature names were lost. Ensure you store them before tra

# Check for mismatches
if len(feature_importance) != len(feature_names):
    print(f"Feature importance length: {len(feature_importance)}")
    print(f"Feature names length: {len(feature_names)}")
    raise ValueError("Mismatch between feature importance and feature names.")

# Create DataFrame for visualization
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot Feature Importance
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='green')
plt.xlabel('Absolute Coefficient Value')
plt.ylabel('Feature')
plt.title('Feature Importance - Tuned SMOTE Logistic Regression')
plt.gca().invert_yaxis()
plt.show()
```



Feature importance analysis:

Key Observations:

1. Dominant Features: Total_Charges and International_Plan are the most influential, contributing substantially more than other features in predicting churn.

2. Moderate Influence: Customer_Service_Calls, Voice_Mail_Plan, and Customer_Service_Category have a moderate level of importance.

3.Low to Negligible Influence: The remaining features have low to negligible importance.

4. Positive Coefficients: All features have positive coefficient values, suggesting a positive relationship with churn (with the likely exception of the binary International_Plan variable).

Business Significance

- **Total Charges as a Key Driver (High Business Significance):** The strong influence of Total_Charges reinforces the business understanding that cost is a major factor in churn decisions. Customers facing higher charges may be more likely to seek alternative providers. This insight allows the business to:

1. **Target high-spending customers with retention offers:** Proactive discounts or loyalty programs could incentivize them to stay.
2. **Analyze pricing plans:** Identify if certain plans lead to unexpectedly high charges and adjust them for better customer satisfaction.

- **International Plan as a Strong Predictor (High Business Significance):** The importance of International_Plan suggests specific churn patterns among subscribers. This allows the business to:

1. **Analyze international plan usage:** Determine if customers are getting value for their money. Low usage could indicate a potential churn risk.
2. **Tailor international plan offerings:** Design plans that better meet customer needs and provide more value.

- **Customer Service Interaction (Moderate Business Significance):** The influence of Customer_Service_Calls and Customer_Service_Category confirms that negative customer service experiences can drive churn. This enables the business to:

1. **Invest in customer service training:** Equip representatives to handle common issues effectively and empathetically.
2. **Identify areas for service improvement:** Analyze call logs and feedback to pinpoint systemic problems and implement solutions.
3. **Offer proactive support:** Reach out to customers in high-risk categories to address potential issues before they escalate.

- **Voice Mail Plan (Moderate Business Significance):** The importance of Voice_Mail_Plan might reveal usage patterns or preferences related to churn. This prompts the business to:
- **Analyze voice mail plan adoption and usage:** Understand if customers find value in the feature.

1. **Target customers based on voice mail plan preferences:** Offer promotions or bundles relevant to their needs.

- **Low Influence Features (Low Business Significance):** The low importance of individual call/charge/minute components suggests that these granular metrics might not be as directly indicative of churn as aggregated measures like Total_Charges.
  This implies that:

1. **Focusing on aggregated metrics is more efficient:** Analyzing Total_Charges provides more business insight than looking at individual call durations or charges.
2. **Feature engineering is crucial:** Combining granular metrics into more meaningful features can improve model performance and provide better insights.

Conclusions

- **Cost and Value are Primary Drivers:** The model emphasizes the importance of cost (Total_Charges) and perceived value (International_Plan, Voice_Mail_Plan) in churn decisions.
- **Customer Service is a Key Touchpoint:** Negative experiences with customer service significantly contribute to churn.
- **Actionable Insights:** The model provides actionable insights that the business can use to develop targeted retention strategies.

Recommendations

1. **Deep Dive into High-Importance Features:** Conduct thorough analyses of Total_Charges, International_Plan, and customer service interactions to understand their specific relationships with churn.
2. **Customer Service Enhancement Initiatives:** Invest in customer service training, identify areas for improvement, and implement proactive support systems.
3. **Targeted Retention Campaigns:** Develop targeted campaigns based on the identified high-risk features.
4. **Feature Engineering Exploration:** Experiment with creating new features by aggregating or transforming existing ones.
5. **Model Refinement:** Consider removing or combining redundant features and re-train the model.

Next steps:

It is evident that after tuning our logistic model there is a slight improvement in capturing the churned customers. However there is a slight trade-off of the increase in number of false positives. To curb this trade-off we will be required to try out other models that may have better performance compared to the previous logistic regression model. Hence we will continue with our modeling process by creating and evaluating other models. The next model is:

- Decision Tree Classifier

5.6 Base Decision Tree Classifier

In [67]:
```
# Initialize the base Decision Tree model
base_dcstree_model = DecisionTreeClassifier(random_state=42)

# Train the model on the imbalanced dataset
base_dcstree_model.fit(X_train_scaled, y_train)
```

Out[67]:
```
DecisionTreeClassifier(random_state=42)
```

In [68]:
```
# Make predictions
y_pred_base_dcstree = base_dcstree_model.predict(X_test_scaled)
```

5.7 Evaluating the base decision tree

In [69]:
```
# Evaluate the model
accuracy_base_dcstree = accuracy_score(y_test, y_pred_base_dcstree) * 100
print(f'Base Decision Tree Model Accuracy: {accuracy_base_dcstree:.2f}%')

# Classification Report
print('Base Decision Tree Model Classification Report:')
print(classification_report(y_test, y_pred_base_dcstree))
```

```
Base Decision Tree Model Accuracy: 94.30%
Base Decision Tree Model Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       566
           1       0.78      0.87      0.82       101

    accuracy                           0.94       667
   macro avg       0.88      0.91      0.89       667
weighted avg       0.95      0.94      0.94       667
```

Key Observations:

1. Metrics Performance:

Accuracy: 94.30%

- The model achieves high accuracy, indicating strong overall performance in predicting both churned and non-churned customers.

Class 0 (Not Churned) Performance:

- Precision: 0.98 → When predicting non-churn, the model is highly confident and correct 98% of the time.

- Recall: 0.96 → The model correctly identifies 96% of actual non-churned customers, missing only 4%.

- F1-score: 0.97 → A balance between precision and recall, showing near-perfect performance.

Class 1 (Churned) Performance:

- Precision: 0.78 → When predicting churn, 78% of predictions are correct, meaning 22% of churn predictions are false positives.

- Recall: 0.87 → The model successfully identifies 87% of actual churned customers, but misses 13% of them.

- F1-score: 0.82 → A strong F1-score, but slightly lower than for Class 0, indicating room for improvement.

2. Significance of the metrics performance:*

- The high accuracy (94.3%) is impressive, but accuracy alone can be misleading when dealing with imbalanced datasets (like churn prediction).
- The high recall (87%) for churned customers is a positive sign, as the model captures most at-risk customers.
- However, the lower precision (78%) for churned customers means that 22% of churn predictions are false positives, potentially leading to unnecessary retention efforts.
- The model favors Class 0 (not churned), as seen in the high precision and recall for this class. This suggests a slight imbalance in learning, where the model prioritizes the majority class more effectively.

3. Trade-Offs

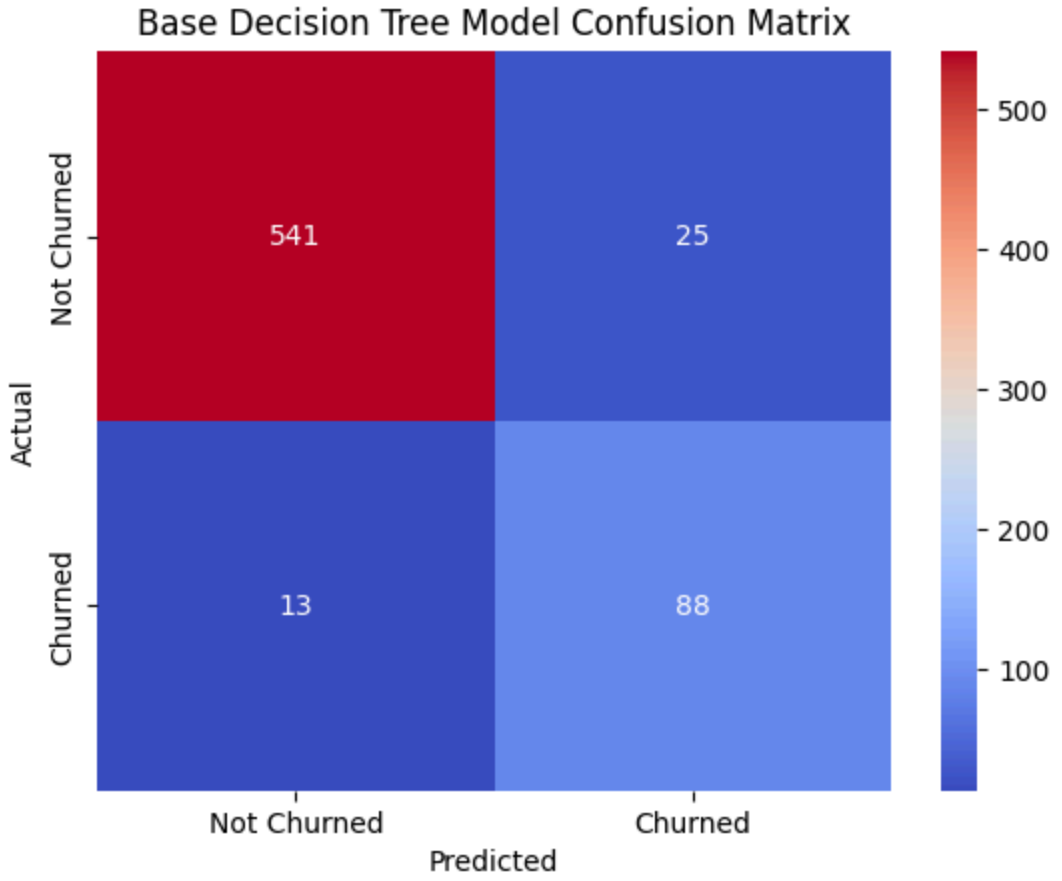
- The model performs well overall but exhibits slight class imbalance issues.
- It is biased toward non-churned customers (Class 0) because they are more frequent in the dataset.
- The false positives for churned customers (Class 1) may lead to unnecessary interventions, but missing actual churners is riskier for the business.

Why Class Balancing?

- Balancing will help the model give equal importance to both classes, improving the ability to distinguish actual churners without excessive false positives.
- A more balanced recall-precision trade-off will ensure the model is fairer in treating both classes, preventing it from over-prioritizing the majority class.

5.7.1 Base decision tree confusion matrix

In [70]:
```
# Confusion Matrix
confusion_base_dcstree = confusion_matrix(y_test, y_pred_base_dcstree)
sns.heatmap(confusion_base_dcstree, annot=True, fmt='d', cmap='coolwarm', xticklabels=['
            yticklabels=['Not Churned', 'Churned'])
plt.title('Base Decision Tree Model Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Confusion Matrix analysis:

Observations

High Number of True Negatives: The model correctly predicted "Not Churned" for 541 customers.

High Number of True Positives: The model correctly predicted "Churned" for 88 customers. This is a good sign, especially compared to your base logistic regression model.

Relatively Low Number of False Positives: The model incorrectly predicted "Churned" for 25 customers. This is a decent performance.

Very Low Number of False Negatives: The model incorrectly predicted "Not Churned" for only 13 customers who actually churned. This is excellent and a significant improvement over the base logistic regression model.

Summary

From the above observations it is evident that the model has performed well in handling the false positives and false negatives compared to the final logistic regression model. Even without handling class imbalance and fine tuning the model gives quite an impressive performance. However the model can be improved further by handling the class imbalance.

Conclusion:

- Based on the observation the base decision tree classifier performs way better than the logistic regression model from the SMOTE logistic model to the the Tuned logistic model. It is evident that the crucial metrics for this business problem have a way better performance than the logistic regression. However despite this good performnace the model may perform way better after handling class imbalance.

Next steps:

- Addressing the class imbalance.
- Explore other class imbalance i.e the class weight technique using scikit-learn

5.8 Handling class imbalance using class weight balance technique in scikit-learn

In [71]:
```
# Initialize the Decision Tree model with class weighting
balanced_dcs_model = DecisionTreeClassifier(random_state=42, class_weight='balanced')

# Train the model on the dataset
balanced_dcs_model.fit(X_train_scaled, y_train)
```

Out[71]:
```
DecisionTreeClassifier(class_weight='balanced', random_state=42)
```

In [72]:
```
# Make predictions
y_pred_balanced_dcs = balanced_dcs_model.predict(X_test_scaled)
```

5.9 Evaluating the balanced model

In [73]:
```
# Evaluate the model
accuracy_balanced_dcs = accuracy_score(y_test, y_pred_balanced_dcs) * 100
print(f'Balanced Decision Tree Model Accuracy: {accuracy_balanced_dcs:.2f}%')

# Classification Report
print('Balanced Decision Tree Model Classification Report:')
print(classification_report(y_test, y_pred_balanced_dcs))
```

```
Balanced Decision Tree Model Accuracy: 95.35%
Balanced Decision Tree Model Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       566
           1       0.82      0.88      0.85       101

    accuracy                           0.95       667
   macro avg       0.90      0.92      0.91       667
weighted avg       0.96      0.95      0.95       667
```

Comparisons and observations:

1. Accuracy Improvement

- The balanced model achieved higher accuracy (95.35%) compared to the base model (94.30%).
- This suggests that adjusting for class imbalance did not compromise overall performance but instead slightly improved generalization.

2. Better Churn Detection (Class 1 - Churned)

- Precision increased from 0.78 to 0.82 → The balanced model makes fewer false positive churn predictions, meaning fewer customers are mistakenly flagged as churn risks.
- Recall improved from 0.87 to 0.88 → The model now captures more actual churners, increasing retention effectiveness.
- F1-score increased from 0.82 to 0.85 → The overall balance between precision and recall has significantly improved, making the model more reliable for churn prediction.

3. Stability in Non-Churned Customers (Class 0 - Not Churned)

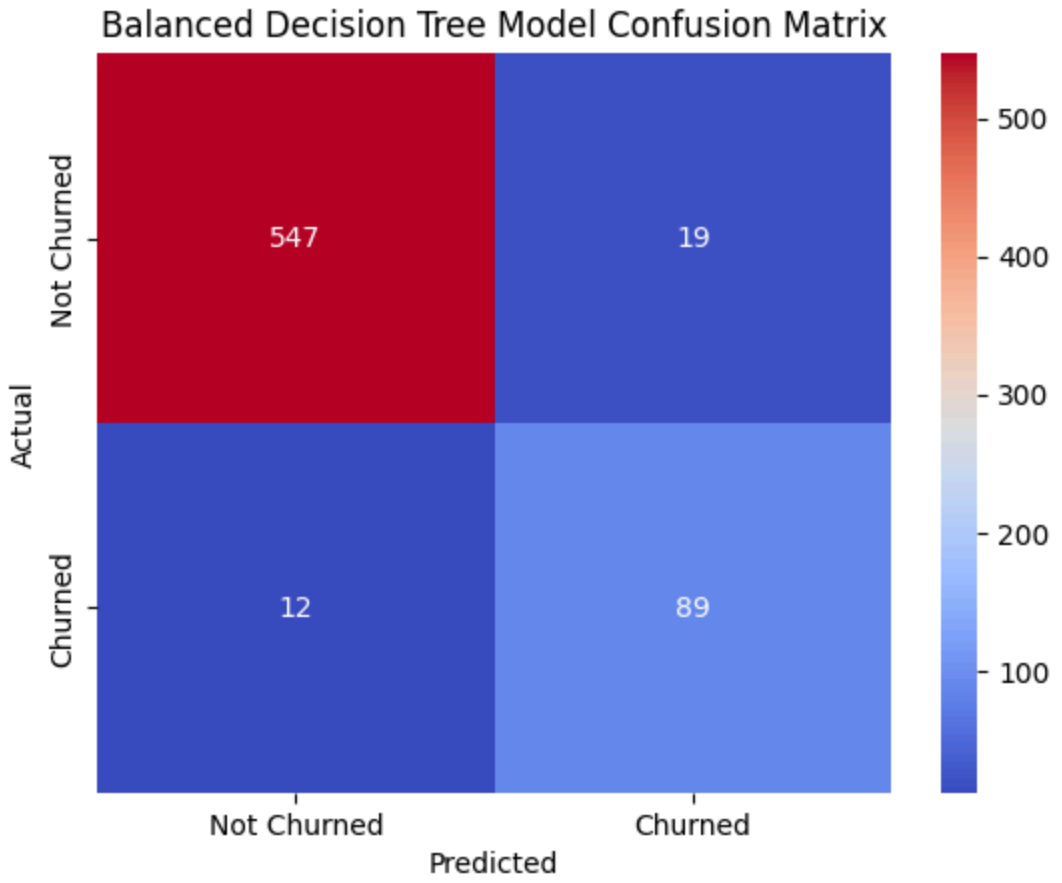
- Precision and recall remain high (~0.98 & 0.97, respectively) → The model still performs exceptionally well in predicting non-churners, meaning there's no major trade-off in accuracy for this class.

Summary

- Class weighting successfully improved churn detection while maintaining strong overall performance.
- The balanced model is less biased toward the majority class (non-churned customers), making it a better business tool.

5.9.1 Balanced decision tree confusion matrix

In [74]:
```
# Confusion Matrix
confusion_balanced_dcs = confusion_matrix(y_test, y_pred_balanced_dcs)
sns.heatmap(confusion_balanced_dcs, annot=True, fmt='d', cmap='coolwarm', xticklabels=['
            yticklabels=['Not Churned', 'Churned'])
plt.title('Balanced Decision Tree Model Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Confusion matrix analysis

High True Negatives: The model correctly identified 547 customers who did not churn.

High True Positives: The model correctly identified 89 customers who did churn.

Low False Positives: The model incorrectly predicted only 19 customers as "Churned" when they did not.

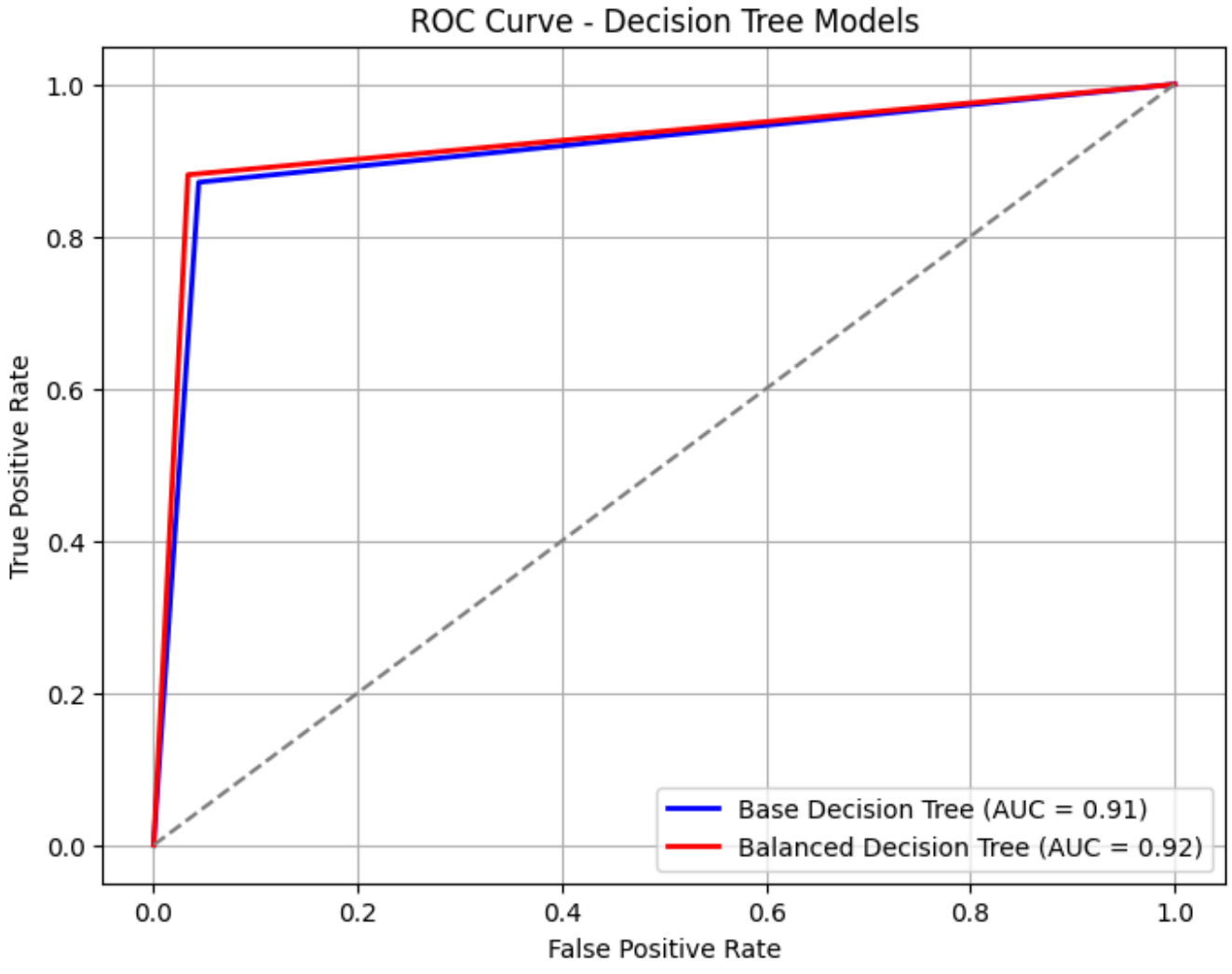
Very Low False Negatives: The model incorrectly predicted only 12 customers as "Not Churned" when they actually did.

Summary

The balanced decision tree model performs strongly, particularly in identifying true positives (churners) while maintaining good precision and a high overall accuracy.

5.9.2 ROC-AUC Comparisons

In [75]:
```
# Get probability scores for ROC curve
y_prob_base_dcs = base_dcstree_model.predict_proba(X_test_scaled)[:, 1]
y_prob_balanced_dcs = balanced_dcs_model.predict_proba(X_test_scaled)[:, 1]

# Compute ROC curve and AUC
fpr_base, tpr_base, _ = roc_curve(y_test, y_prob_base_dcs)
roc_auc_base = auc(fpr_base, tpr_base)

fpr_balanced, tpr_balanced, _ = roc_curve(y_test, y_prob_balanced_dcs)
roc_auc_balanced = auc(fpr_balanced, tpr_balanced)

# Plot ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_base, tpr_base, color='blue', lw=2, label=f'Base Decision Tree (AUC = {roc_
plt.plot(fpr_balanced, tpr_balanced, color='red', lw=2, label=f'Balanced Decision Tree (
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Random classifier line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree Models')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



ROC-Curve and AUC analysis:

Higher AUC for Balanced Model: The balanced decision tree model has a slightly higher AUC of 0.92 compared to the base decision tree model with an AUC of 0.91. This indicates that, overall, the balanced model has a marginally better ability to distinguish between the classes (churned vs. not churned).

Almost similar Overall Shape: Both curves slightly have a similar shape, indicating that the general performance characteristics of the models are almost alike.

Balanced Model's Initial Steepness: The balanced model's curve shows a slightly steeper rise in the lower FPR region. This suggests that the balanced model might be able to capture a slightly larger proportion of true positives (churned customers) with a smaller trade-off in false positives, especially at lower thresholds.

Crossover at Higher FPR: There's a point where the curves cross over at higher FPR values. This indicates that the trade-offs in terms of FPR and TPR shift slightly at different thresholds. Depending on where you set your classification threshold, you might observe minor differences in performance.

Conclusion:

The improvement in AUC from 0.91 to 0.92 is quite small. It suggests that the balancing process has yielded a marginal enhancement in the model's overall discriminative power.

Next steps:

There is a slight improvement of the balanced model performance as compared to the base decision tree classifier. Further improvements can be attemped by performning hyperparameter tuning and evaluate the tuned model performance

5.10 Hyperparameter tuning of the balanced decision tree model

In [76]:
```
# Define hyperparameter values to test
max_depth_values = [3, 5, 10, None]
min_samples_split_values = [2, 5, 10]
min_samples_leaf_values = [1, 2, 5]
criteria = ['gini', 'entropy']

best_f1 = 0
best_params = {}
best_model = None

# Loop over hyperparameters manually
for depth in max_depth_values:
    for split in min_samples_split_values:
        for leaf in min_samples_leaf_values:
            for crit in criteria:

                # Initialize Decision Tree with current hyperparameters
                hyp_model = DecisionTreeClassifier(
                    random_state=42,
                    class_weight='balanced',
                    max_depth=depth,
                    min_samples_split=split,
                    min_samples_leaf=leaf,
                    criterion=crit
                )

                # Train model
                hyp_model.fit(X_train_scaled, y_train)

                # Make predictions
                y_pred = hyp_model.predict(X_test_scaled)

                # Compute F1-score
                f1 = f1_score(y_test, y_pred)

                # Track the best model
                if f1 > best_f1:
                    best_f1 = f1
                    best_params = {'max_depth': depth, 'min_samples_split': split, 'min_
                    best_model = hyp_model

# Print best parameters and F1-score
print("Best Parameters:", best_params)
print(f"Best F1-Score: {best_f1:.4f}")
```

```
Best Parameters: {'max_depth': 5, 'min_samples_split': 2, 'min_samples_leaf': 1, 'crite
rion': 'gini'}
Best F1-Score: 0.9263
```

5.11 Evaluate the tuned decision tree

In [77]:
```
y_pred_best = hyp_model.predict(X_test_scaled)
accuracy_best = accuracy_score(y_test, y_pred_best) * 100
print(f'Tuned Decision Tree Model Accuracy: {accuracy_best:.2f}%')
print('Tuned Decision Tree Model Classification Report:')
print(classification_report(y_test, y_pred_best))
```

```
Tuned Decision Tree Model Accuracy: 94.60%
Tuned Decision Tree Model Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       566
           1       0.79      0.87      0.83       101

    accuracy                           0.95       667
   macro avg       0.88      0.92      0.90       667
weighted avg       0.95      0.95      0.95       667
```

Comparing the tuned model to the balanced model

1. Accuracy Slightly Decreased

- The tuned model's accuracy (94.60%) is lower than the balanced model (95.35%).This suggests that tuning did not improve generalization and might have led to overfitting on the training data.

2. Churn Detection Performance Dropped

- Precision for churned customers dropped from 0.82 to 0.79 → The tuned model makes more false positive churn predictions, potentially increasing unnecessary retention efforts.
- Recall slightly decreased from 0.88 to 0.87 → The model is missing slightly more actual churners.
- F1-score dropped from 0.85 to 0.83, indicating a minor decline in the balance between precision and recall.

3. Stability in Non-Churned Class (Class 0 - Not Churned)

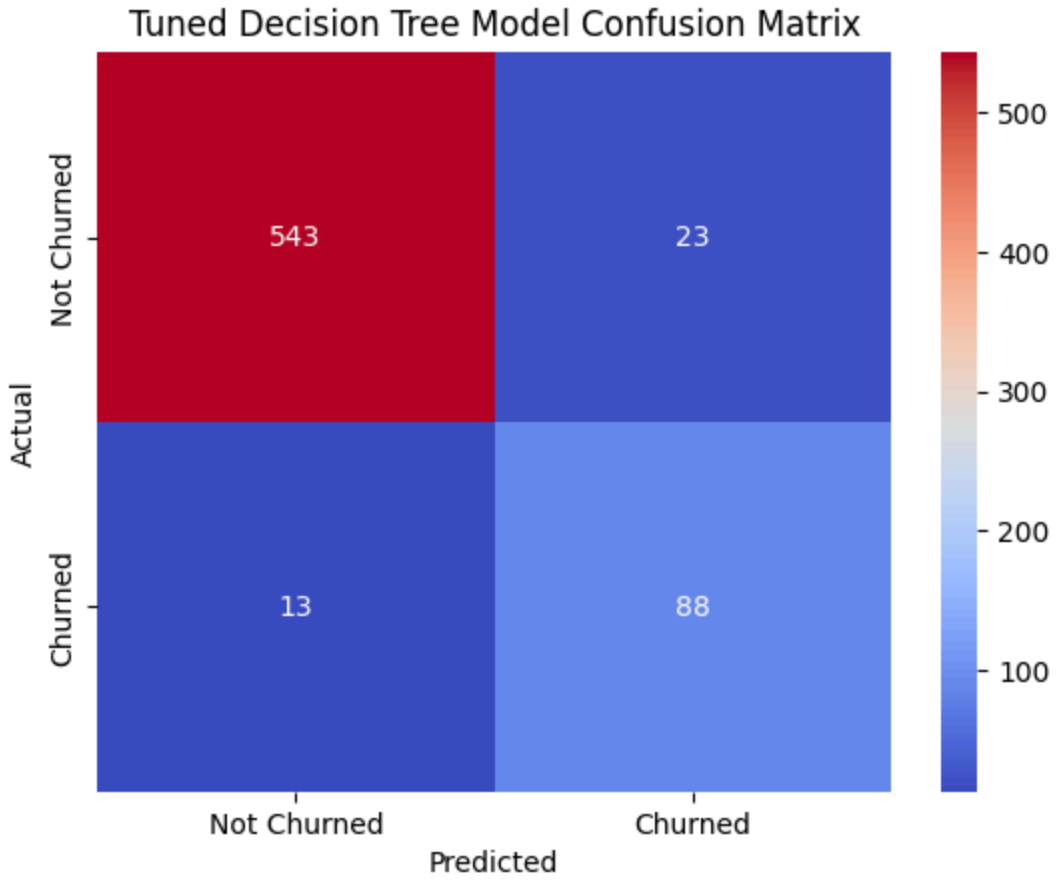
- The performance for non-churned customers remains stable (0.98 precision, 0.96 recall), meaning the model is still accurate in identifying non-churners.

Findings and insights:

From the above observation of the tuned decision tree classifier it is evident that it underperformed compared to the previous balanced model.What does this tell us? It is quite obvious that tuning may not be a key factor that would help the metric performances of a model. From the above comparisons though slightly all the metrics for the churned class especially the recall and F1-score(which are crucial metrics to this business problem) dropped. This has quite impacted the model and can affect our objective in solving the business problem

5.12 Tuned decision tree confusion matrix

In [78]:
```
# Confusion Matrix
confusion_best = confusion_matrix(y_test, y_pred_best)
sns.heatmap(confusion_best, annot=True, fmt='d', cmap='coolwarm', xticklabels=['Not Chur
plt.title('Tuned Decision Tree Model Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Confusion matrix comparisons:

1. True Positives (TP):

Balanced Model: 89

Tuned Model: 88

Decrease: The tuned model correctly identified 1 fewer churned customer. This is a minor decrease in performance.

2. False Positives (FP):

Balanced Model: 19

Tuned Model: 23

Increase: The tuned model increased the number of false positives by 4. This is a negative change.

3. False Negatives (FN):

Balanced Model: 12

Tuned Model: 13

Increase: The tuned model slightly increased the number of false negatives by 1. This is a minor negative change.

4. True Negatives (TN):

Balanced Model: 547

Tuned Model: 543

Decrease: The tuned model correctly identified 4 fewer non-churned customers. This is a minor decrease in performance.

Conclusions:

The tuned decision tree model shows a slightly worse performance compared to the balanced decision tree model. All key metrics have either decreased or remained the same.

Potential Reasons:

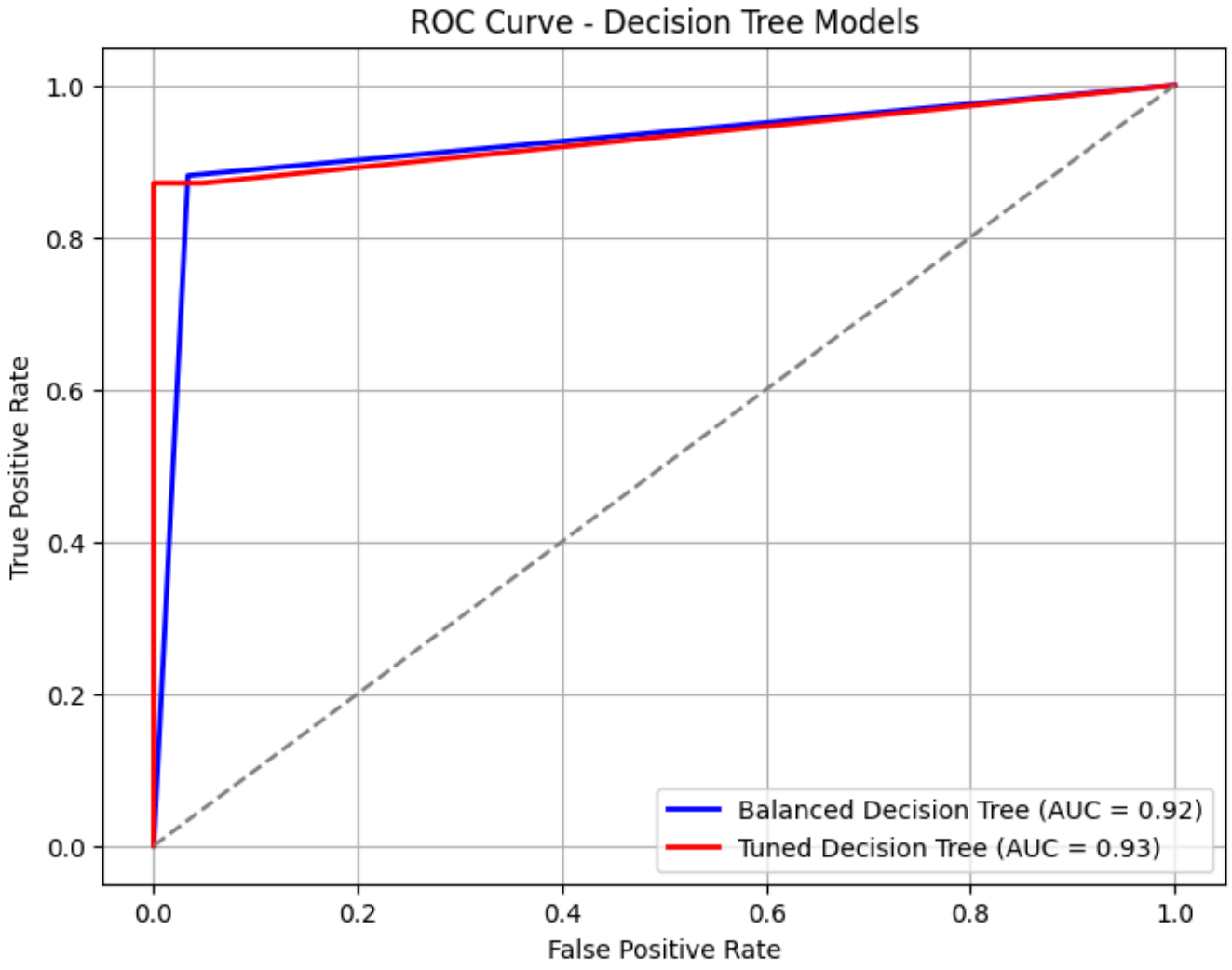
1.Overfitting: The tuning process might have led to overfitting on the training data, making the model less generalizable to unseen data.

2.Suboptimal Hyperparameters: The chosen hyperparameters for the tuned model might not be optimal for the given data.

3.Random Variation: There might be some random variation in the model training process, leading to slightly different results.

5.13 ROC-AUC comparisons

In [79]:
```
# Get probability scores for ROC curve
y_prob_balanced_dcs = balanced_dcs_model.predict_proba(X_test_scaled)[:, 1]
y_prob_hyp_dcs = hyp_model.predict_proba(X_test_scaled)[:, 1]

# Compute ROC curve and AUC
fpr_balanced, tpr_balanced, _ = roc_curve(y_test, y_prob_balanced_dcs)
roc_auc_balanced = auc(fpr_balanced, tpr_balanced)

fpr_dcs_tuned, tpr_dcs_tuned, _ = roc_curve(y_test, y_prob_hyp_dcs)
roc_auc_dcs_tuned = auc(fpr_dcs_tuned, tpr_dcs_tuned)

# Plot ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_balanced, tpr_balanced, color='blue', lw=2, label=f'Balanced Decision Tree
plt.plot(fpr_dcs_tuned, tpr_dcs_tuned, color='red', lw=2, label=f'Tuned Decision Tree (A
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Random classifier line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree Models')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



Key Observations

Higher AUC for Tuned Model: The tuned decision tree model (red line) has a slightly higher AUC of 0.93 compared to the balanced decision tree model (blue line) with an AUC of 0.92. This suggests that, overall, the tuned model has a marginally better ability to distinguish between the classes (churned vs. not churned).

Similar Overall Shape: Both curves have a similar shape, indicating that the general performance characteristics of the models are alike.

Tuned Model's Initial Steepness: The tuned model's curve shows a slightly steeper rise in the lower FPR region. This suggests that the tuned model might be able to capture a slightly larger proportion of true positives (churned customers) with a smaller trade-off in false positives, especially at lower thresholds.

Crossover at Higher FPR: There's a point where the curves cross over at higher FPR values. This indicates that the trade-offs in terms of FPR and TPR shift slightly at different thresholds. Depending on where you set your classification threshold, you might observe minor differences in performance.

Conclusion based on the decision tree classifier

Which Model is the preferred decision tree classifier model?

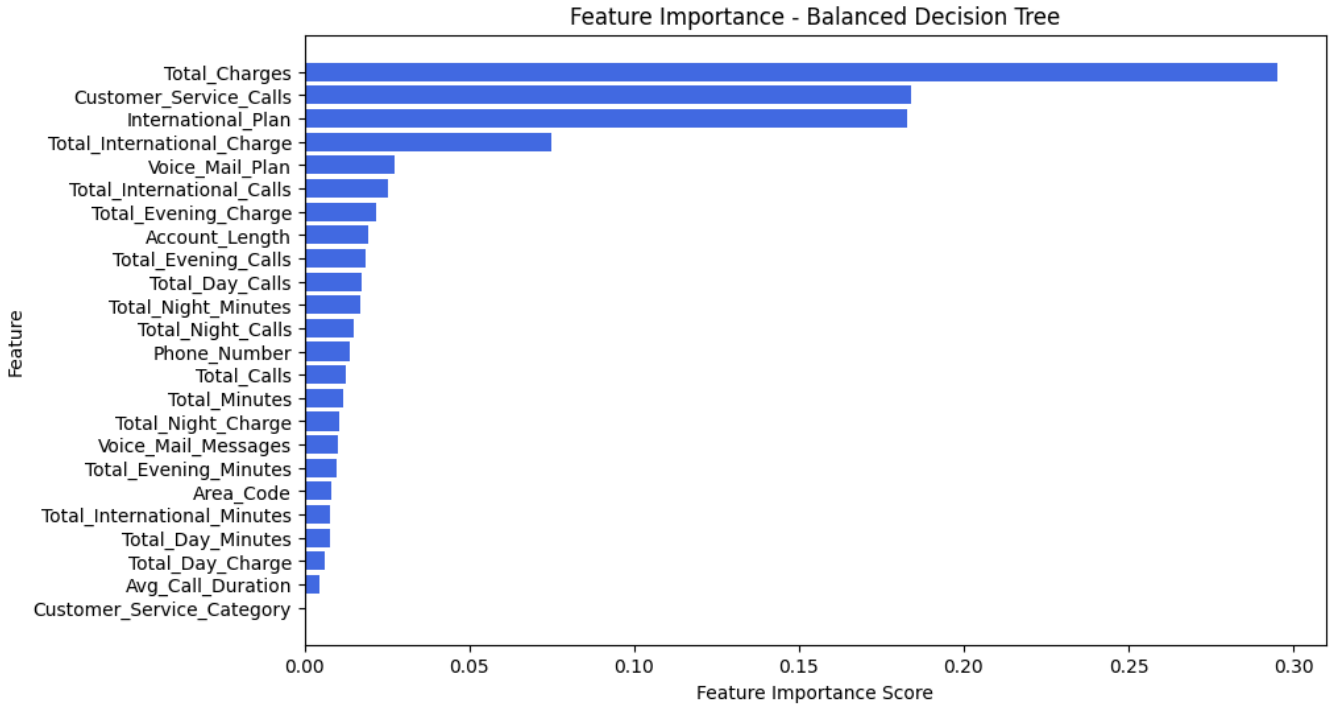
- The Balanced Decision Tree Model (before tuning) is preferred!

Reasons:

- It outperforms the tuned model in accuracy, precision, recall, and F1-score for the churned class.
- The slight decrease in performance after tuning suggests that the default hyperparameters were already well-suited for the problem.
- Tuning did not provide a meaningful improvement and slightly reduced the model's reliability for detecting churn.

5.14 Feature Importance

In [80]:
```
# Extract feature importance from the decision tree model
feature_importance = balanced_dcs_model.feature_importances_

# Get feature names
feature_names = X_train.columns.tolist()

# Create DataFrame for visualization
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot Feature Importance
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='royalblue')
plt.xlabel('Feature Importance Score')
plt.ylabel('Feature')
plt.title('Feature Importance - Balanced Decision Tree')
plt.gca().invert_yaxis()
plt.show()
```



Feature importance analysis for the balanced decision tree

Key Observations

Dominant Feature: Total_Charges is overwhelmingly the most influential feature. Its bar dwarfs all others, suggesting it's the primary driver in the model's decisions.

Moderate Influence: Customer_Service_Calls and International_Plan have a moderate level of importance, though significantly less than Total_Charges.

Low to Negligible Influence: The remaining features have very low to negligible importance. Their bars are extremely short, indicating minimal impact on the model's predictions.

Business Significance:

Total Charges as a Primary Driver (Extremely High Business Significance): The overwhelming importance of Total_Charges emphasizes the critical role cost plays in churn decisions. This insight is highly actionable:

1. **Personalized Retention Offers:** Customers with high Total_Charges could be targeted with personalized discounts or loyalty programs.
2. **Pricing Plan Analysis:** Investigate if specific pricing plans contribute to higher charges and subsequent churn. This could lead to restructuring plans for better customer satisfaction.

Customer Service Interaction (High Business Significance): The influence of Customer_Service_Calls reinforces the understanding that negative customer service experiences are strongly linked to churn. This allows the business to:

1. **Enhance Customer Service Training:** Focus on equipping representatives to handle common issues effectively and empathetically.
2. **Proactive Customer Service:** Identify customers who frequently contact support and offer proactive assistance to address potential issues.

International Plan (High Business Significance): The importance of International_Plan suggests specific churn patterns among subscribers. This enables the business to:

1. **Optimize International Plan Offerings:** Analyze usage patterns and adjust plans to provide better value and competitiveness.
2. **Targeted Promotions:** Offer special promotions or bundles to international plan subscribers to incentivize them to stay.

Low Influence Features (Low Business Significance): The extremely low importance of most other features suggests that they might not be directly indicative of churn in this model. This could be due to:

1. **Redundancy:** Some features might be redundant with Total_Charges or other important features.
2. **Lack of Direct Impact:** Granular metrics like individual call durations or specific charges might not be as predictive as the aggregated Total_Charges.

Conclusions

Cost is Key: The model heavily relies on Total_Charges to predict churn, highlighting the importance of cost considerations for customers.

Customer Service Matters: Customer service interactions play a significant role in churn decisions.

Actionable Insights: The model provides actionable insights for targeted retention campaigns and customer service improvements.

Recommendations

1. **Detailed Analysis of Total Charges:** Conduct further analysis to understand the distribution of Total_Charges among churned and non-churned customers. Identify specific thresholds or patterns that might trigger churn.
2. **Customer Service Enhancement Initiatives:** Invest in training, improve response times, and personalize interactions to enhance customer service experiences.

3. **International Plan Optimization:** Analyze usage patterns and adjust plans to provide better value and competitiveness.
4. **Feature Engineering Exploration:** While not as crucial as with logistic regression, consider if combining or transforming existing features could provide additional insights.
5. **Model Refinement:** Evaluate if removing less important features would simplify the model without

Comparison between the Tuned Logistic Regression and the Balanced Decision Model

1. Overall Accuracy (Best performance: Decision Tree)

- The Balanced Decision Tree (95.35%) significantly outperforms Tuned Logistic Regression (78.71%) in accuracy.
- Logistic regression struggles because churn prediction is not purely linear, making decision trees a better choice.

2. Churn Detection Precision (Best performance: Decision Tree)

- Logistic Regression Precision (0.40) is very low → This means that many of its churn predictions are false positives, leading to unnecessary retention efforts.
- Decision Tree Precision (0.82) is much better, meaning it makes far fewer false positive churn predictions.

3. Churn Detection Recall (Best performance: Decision Tree)

- Logistic regression has a lower recall (0.81) compared to 0.88 for the decision tree.
- The Decision Tree maintains high recall (0.88) while keeping precision high (0.82), striking a better balance.

4. F1-Score (Best performance: Decision Tree)

- Decision Tree (0.85) vs. Logistic Regression (0.54) → The Decision Tree has a much better balance between precision and recall, making it a more reliable model.

Which Model is Best?

- The Balanced Decision Tree Classifier is the superior model because:

1. It has higher accuracy (95.35%), making better predictions overall.
2. It has much better precision (0.82 vs. 0.40), avoiding excessive false churn predictions.
3. It maintains strong recall (0.88), ensuring actual churners are still identified.
4. It has a better F1-score (0.85 vs. 0.54), meaning it balances precision and recall effectively.

Final Verdict based on the two models:

Since our goal is to maximize the correct churn predictors the model to go with is obviously the balanced decision tree classifier.

Next steps:

With our decision tree having a marvelous performance we can further explore other scikit-learn models and do further evaluation before settling for the decision tree classifier. This next model is the random forest classifier.

5.15 Base Random Forest Classifier

In [81]:
```
# Initialize Random Forest model
rndf_base = RandomForestClassifier(n_estimators=100, random_state=42)

# Train on imbalanced data
rndf_base.fit(X_train_scaled, y_train)
```

Out[81]:
```
RandomForestClassifier
(https://scikit-learn.org/1.6/modules/generated/sklearn.ensemble.RandomForest

RandomForestClassifier(random_state=42)
```

In [82]:
```
# Predictions
y_pred_rndf_base = rndf_base.predict(X_test_scaled)
```

5.16 Evaluate the Base Random Forest

In [108]:
```
# Model Evaluation
accuracy_rndf_base = accuracy_score(y_test, y_pred_rndf_base) * 100
print(f'Base Random Forest Model Accuracy: {accuracy_rndf_base:.2f}%')

print('Base Random Forest Model Classification Report:')
print(classification_report(y_test, y_pred_rndf_base))
```

```
Base Random Forest Model Accuracy: 97.75%
Base Random Forest Model Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       566
           1       1.00      0.85      0.92       101

    accuracy                           0.98       667
   macro avg       0.99      0.93      0.95       667
weighted avg       0.98      0.98      0.98       667
```

Observations and Insights:

Metrics Performance:

1. Accuracy (97.75%)

- The model demonstrates high accuracy, meaning it correctly classifies the vast majority of customers. However, accuracy alone does not provide the full picture, so we need to analyze precision, recall, and F1-score to understand its real-world impact.

2. Precision for Churned Customers (1.00)

- The model is highly confident in its churn predictions, meaning whenever it predicts a customer will churn, it is always correct.
- This is beneficial for targeted retention efforts, as it ensures the business does not waste resources on false churn predictions.

3. Recall for Churned Customers (0.85)

- The model captures 85% of actual churners, but misses 15% who will churn but are not flagged.
- Missing these churners could have a business impact if they leave without being identified and retained.

4. F1-Score for Churned Customers (0.92)

- This metric balances precision and recall, showing a strong performance in identifying churners without too many false positives or false negatives.

5. Considerations

Near-Perfect Performance for Non-Churners: The model perfectly classifies non-churners (recall = 1.00), meaning it does not mistakenly identify any loyal customers as churners.

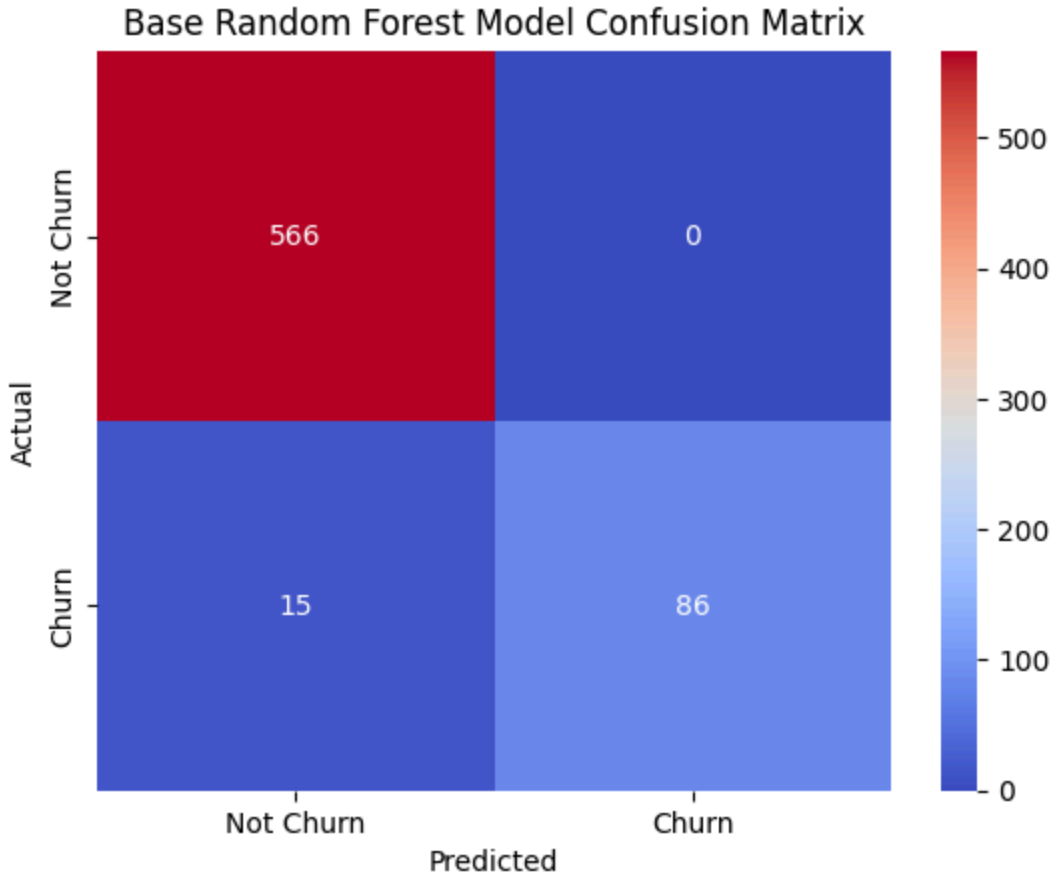
- This is valuable because businesses can focus their retention efforts solely on the true churn risks.

Slightly Lower Recall for Churners: While 85% recall is strong, improving it could help capture more churners and prevent potential revenue loss.

- This could be improved through hyperparameter tuning, which may adjust how the model makes its split decisions.

5.16.1 Base Random Forest Confusion matrix

In [84]:
```
# Confusion Matrix
conf_matrix_rndf_base = confusion_matrix(y_test, y_pred_rndf_base)
sns.heatmap(conf_matrix_rndf_base, annot=True, fmt='d', cmap='coolwarm', xticklabels=['N
            yticklabels=['Not Churn', 'Churn'])
plt.title('Base Random Forest Model Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Confusion matrix analysis:

1. Perfect Classification of Not Churned: The model correctly classified all 566 customers who did not churn. This is excellent performance in identifying true negatives.

2. High Number of True Positives: The model correctly classified 86 out of 101 customers who actually churned. This is also very good performance.

3. Zero False Positives: The model did not incorrectly predict any customers as "Churned" when they actually did not churn. This indicates high precision in the "Churn" predictions.

4. Low Number of False Negatives: The model incorrectly classified 15 customers as "Not Churned" when they actually churned. While not perfect, this is a relatively low number of false negatives.

Conclusion:

- The base Random Forest model is already high-performing, with a strong balance between precision and recall.
- Since Random Forest inherently handles class imbalance, balancing techniques may not be necessary.
- Despite the model performing relatively well we can explore other techniques i.e the hyper-parameter tuning that may improve the model performance further.

Next steps:

- Hyper-parameter tuning using RandomizedSearchCV

5.17 Hyper-parameter tuning using RandomizedSearchCV

In [85]:
```
# Define hyperparameter distribution
param_dist = {
    'n_estimators': np.arange(50, 300, 50),
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize Random Forest model
rf = RandomForestClassifier(random_state=42)

# Perform Randomized Search with 5-fold Cross Validation
random_search = RandomizedSearchCV(rf, param_distributions=param_dist, n_iter=20, cv=5,
                                   n_jobs=-1, random_state=42)
random_search.fit(X_train_scaled, y_train)

# Best model from RandomizedSearchCV
best_rf = random_search.best_estimator_

# Predictions
y_pred_rf_tuned = best_rf.predict(X_test_scaled)


# Print best parameters
print("Best Parameters from RandomizedSearchCV:", random_search.best_params_)
```

```
Best Parameters from RandomizedSearchCV: {'n_estimators': 100, 'min_samples_split': 5,
'min_samples_leaf': 1, 'max_depth': 20}
```

5.18 Evaluate the tuned random forest

In [86]:
```
# Model Evaluation
accuracy_rf_tuned = accuracy_score(y_test, y_pred_rf_tuned) * 100
print(f'Tuned Random Forest Model Accuracy: {accuracy_rf_tuned:.2f}%')

print('Tuned Random Forest Model Classification Report:')
print(classification_report(y_test, y_pred_rf_tuned))
```

```
Tuned Random Forest Model Accuracy: 97.90%
Tuned Random Forest Model Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       566
           1       1.00      0.86      0.93       101

    accuracy                           0.98       667
   macro avg       0.99      0.93      0.96       667
weighted avg       0.98      0.98      0.98       667
```

Observations and Comparisons:

1. Accuracy (↑ 97.75% → 97.90%)

- The tuned model shows a slight improvement in accuracy, indicating that it generalizes slightly better.However, the difference is marginal, suggesting the base model was already highly optimized.

2. Precision for Churned Customers (Same at 1.00)

- Both models have a perfect precision score (1.00), meaning whenever they predict a customer will churn, they are always correct.

3. Recall for Churned Customers (↑ 0.85 → 0.86)

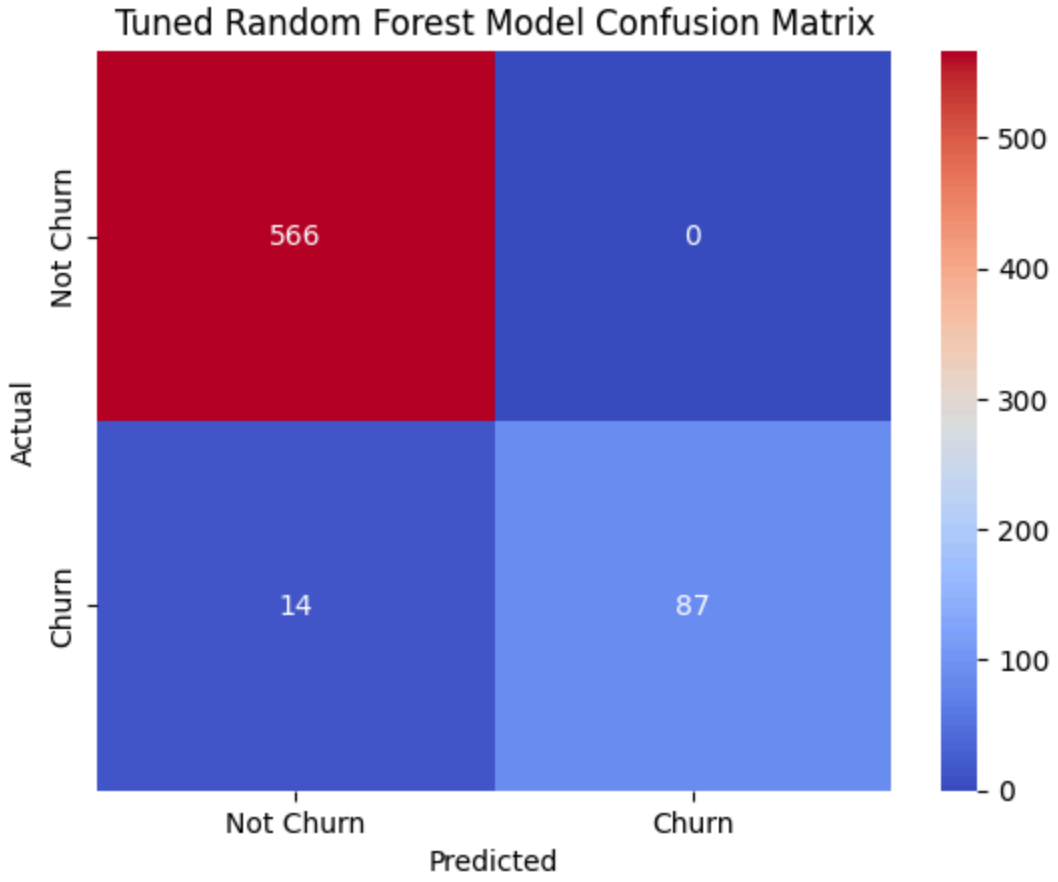
- The tuned model slightly improved recall, capturing 1% more actual churners.While small, this means the model is catching more customers at risk of leaving, which is valuable for proactive retention strategies.

4. F1-Score for Churned Customers (↑ 0.92 → 0.93)

- The slight boost in recall led to an increase in F1-score, meaning the model balances precision and recall even better.

5.18.1 Confusion matrix of the tuned random forest

In [87]:
```
# Confusion Matrix
conf_matrix_rf_tuned = confusion_matrix(y_test, y_pred_rf_tuned)
sns.heatmap(conf_matrix_rf_tuned, annot=True, fmt='d', cmap='coolwarm', xticklabels=['No
            yticklabels=['Not Churn', 'Churn'])
plt.title('Tuned Random Forest Model Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Confusion matrix comparison and analysis:

1. True Positives (TP):

Base Model: 86

Tuned Model: 87

Improvement: The tuned model correctly identified 1 more churned customer. This is a minor improvement.

2. False Positives (FP):

Base Model: 0

Tuned Model: 0

No Change: Both models have zero false positives, which is excellent.

3. False Negatives (FN):

Base Model: 15

Tuned Model: 14

Improvement: The tuned model reduced the number of false negatives by 1. This is a minor improvement.

4. True Negatives (TN):

Base Model: 566

Tuned Model: 566

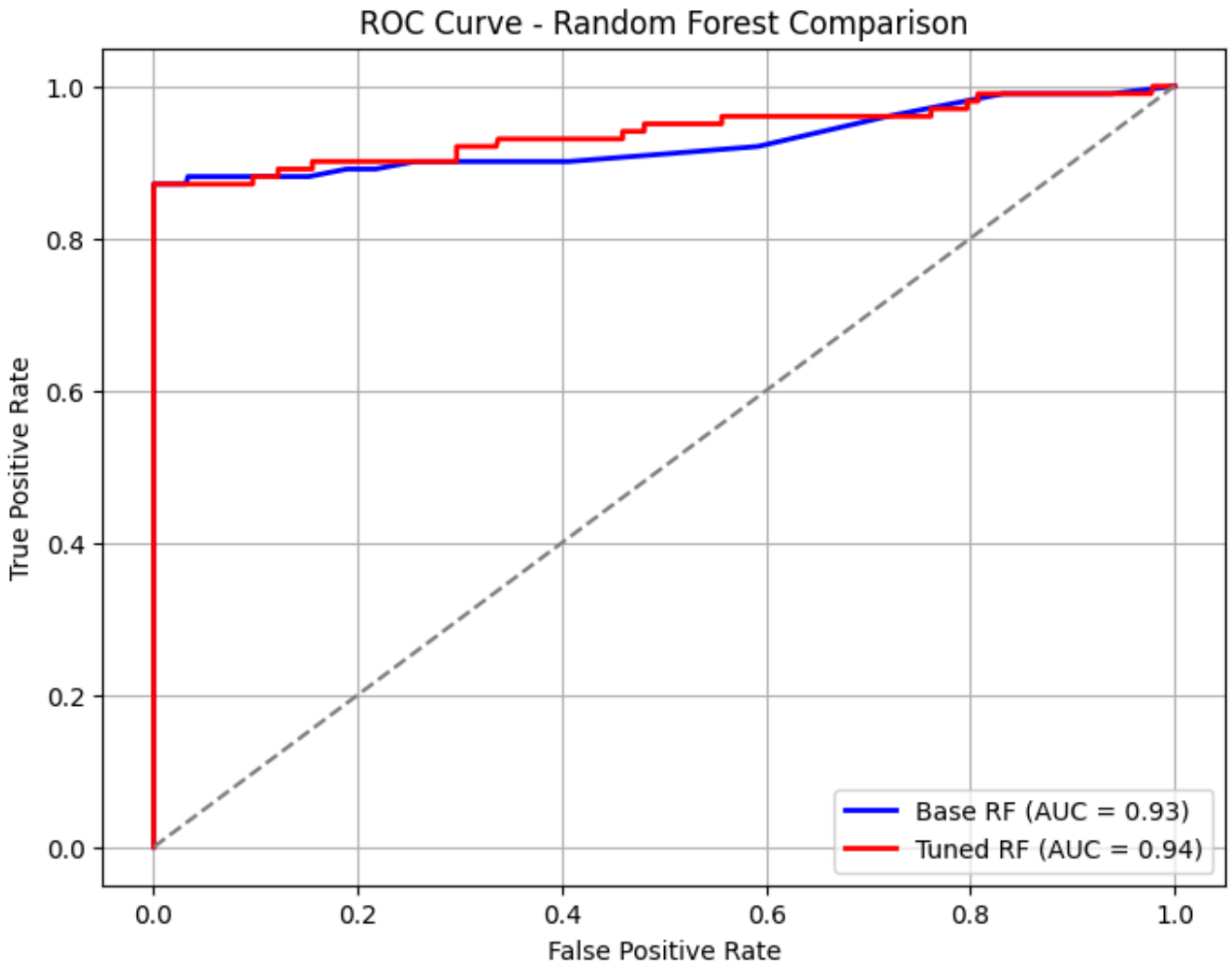
No Change: Both models correctly classified all non-churned customers.

Summary

The tuned Random Forest model shows a very slight improvement in recall compared to the base model, with no negative impacts on other metrics.

5.18.2 ROC-AUC Comparisons

In [88]:
```
# ROC Curve Comparison
# Get probability scores
y_prob_rndf_base = rndf_base.fit(X_train_scaled, y_train).predict_proba(X_test_scaled)[:
y_prob_rndf_tuned = best_rf.predict_proba(X_test_scaled)[:, 1]

# Compute ROC curve and AUC
fpr_base, tpr_base, _ = roc_curve(y_test, y_prob_rndf_base)
roc_auc_base = auc(fpr_base, tpr_base)
fpr_tuned, tpr_tuned, _ = roc_curve(y_test, y_prob_rndf_tuned)
roc_auc_tuned = auc(fpr_tuned, tpr_tuned)

# Plot ROC Curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_base, tpr_base, color='blue', lw=2, label=f'Base RF (AUC = {roc_auc_base:.2
plt.plot(fpr_tuned, tpr_tuned, color='red', lw=2, label=f'Tuned RF (AUC = {roc_auc_tuned
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line for random class
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest Comparison')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



ROC-AUC Analysis:

Key Observations:

1. Slightly Higher AUC for Tuned Model: The tuned Random Forest model has a slightly higher AUC of 0.94 compared to the base Random Forest model with an AUC of 0.93. This suggests that, overall, the tuned model has a marginally better ability to distinguish between the classes (churned vs. not churned).

2. Similar Overall Shape: Both curves have a similar shape, indicating that the general performance characteristics of the models are alike.

3. Tuned Model's Initial Steepness: The tuned model's curve shows a slightly steeper rise in the lower FPR region. This suggests that the tuned model might be able to capture a slightly larger proportion of true positives (churned customers) with a smaller trade-off in false positives, especially at lower thresholds.

4. Crossover at Higher FPR: There's a point where the curves cross over at higher FPR values. This indicates that the trade-offs in terms of FPR and TPR shift slightly at different thresholds. Depending on where you set your classification threshold, you might observe minor differences in performance.

Summary

Based on the ROC curve analysis alone, the tuned Random Forest model is slightly better due to the marginally higher AUC and the slightly steeper rise in the lower FPR region. However, the improvements are quite small.

Conclusions

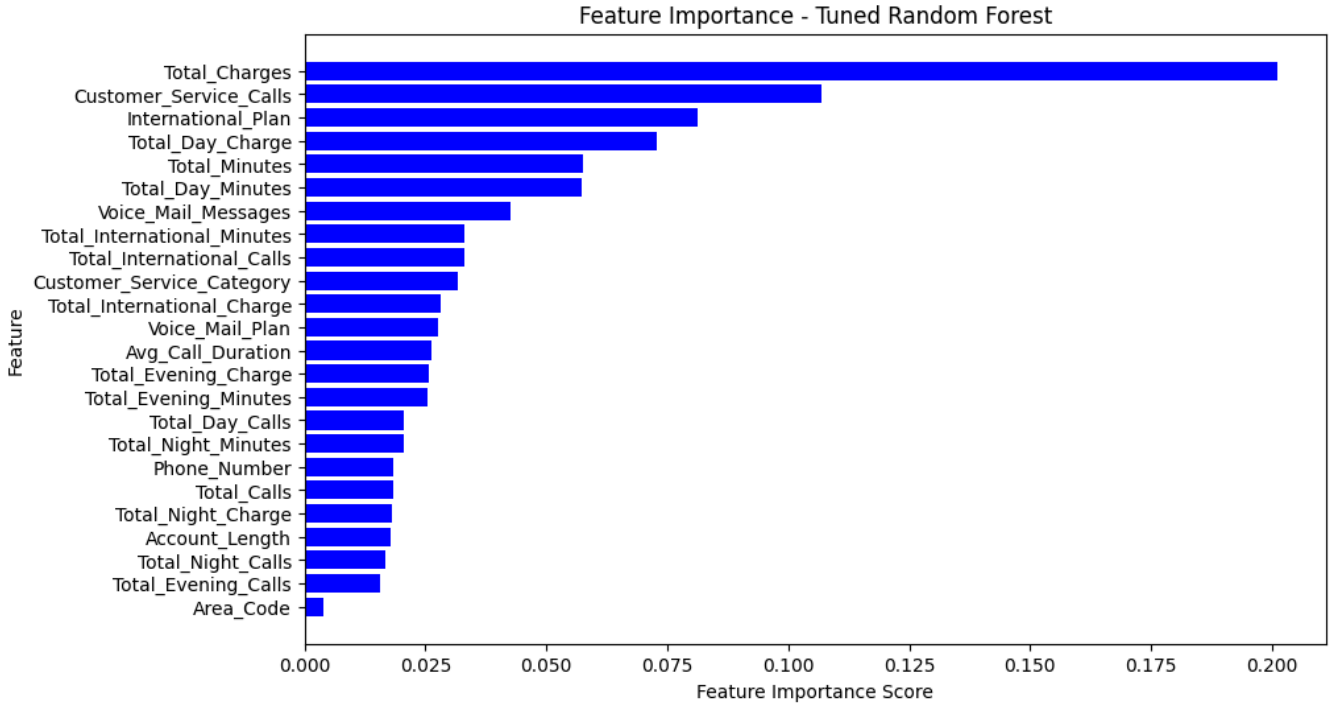
Which Model is Better?

- The tuned Random Forest model is preferred because it slightly improves recall and F1-score while maintaining the same high precision.
- This because:

1. The tuned model identifies more actual churners, which is crucial for a churn prediction model.
2. The increase in F1-score means the tuned model maintains a better trade-off between correctly identifying churners and minimizing false positives.
3. Both models have perfect precision (1.00) for the churned class, meaning they never falsely predict churn for non-churners.This ensures the business does not waste resources on unnecessary customer retention efforts.
4. While accuracy is already high, the slight improvement indicates the tuned model generalizes slightly better across different data points.
5. This makes it more robust and reliable for real-world deployment.

5.19 Feature Importance

In [89]:
```
# Extract feature importance from the tuned random forest model
feature_importance = best_rf.feature_importances_

# Create DataFrame
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot Feature Importance
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color ='blue')
plt.xlabel('Feature Importance Score')
plt.ylabel('Feature')
plt.title('Feature Importance - Tuned Random Forest')
plt.gca().invert_yaxis()
plt.show()
```



Feature importance analysis for the tuend Random Forest

Key Observations

Dominant Feature: Total_Charges is overwhelmingly the most important feature, dwarfing all others. This indicates it's the strongest predictor in the model.

Moderate Influence: Customer_Service_Calls, International_Plan, Total_Day_Charge, Total_Minutes, and Total_Day_Minutes have a moderate level of influence, though considerably less than Total_Charges.

Low to Negligible Influence: The remaining features have very low to negligible importance, suggesting minimal impact on the model's predictions.

Business Significance

Total Charges as Primary Driver (Extremely High Business Significance): The overwhelming importance of Total_Charges reinforces the understanding that cost is a major factor in churn decisions. This allows the business to:

1. **Personalized Retention Offers:** Target high-spending customers with personalized discounts or loyalty programs.
2. **Pricing Plan Analysis:** Investigate if specific pricing plans contribute to higher charges and subsequent churn. Restructure plans for better customer satisfaction.

Customer Service Interaction (High Business Significance): The influence of Customer_Service_Calls confirms that negative customer service experiences are strongly linked to churn. This enables the business to:

1. **Enhance Customer Service Training:** Equip representatives to handle common issues effectively and empathetically.
2. **Proactive Customer Service:** Identify customers who frequently contact support and offer proactive assistance.

International Plan (High Business Significance): The importance of International_Plan suggests specific churn patterns among subscribers. This allows the business to:

1. **Optimize International Plan Offerings:** Analyze usage patterns and adjust plans to provide better value and competitiveness.
2. **Targeted Promotions:** Offer special promotions or bundles to international plan subscribers.

Total Day Charge and Total Minutes (Moderate Business Significance): The inclusion of Total_Day_Charge and Total_Minutes suggests that usage patterns during the day and overall call duration are also relevant churn indicators. This allows the business to:

1. **Analyze calling patterns:** Understand if specific usage patterns are associated with churn.
2. **Tailor offers based on usage:** Offer packages or promotions that incentivize continued usage.

Low Influence Features (Low Business Significance): The low importance of most other features suggests they might not be directly indicative of churn in this model. This could be due to:

1. **Redundancy:** Some features might be redundant with Total_Charges or other important features.
2. **Lack of Direct Impact:** Granular metrics like individual call durations or specific charges might not be as predictive as aggregated measures like Total_Charges.

Conclusions

Cost is Key: The model heavily relies on Total_Charges to predict churn, highlighting the importance of cost considerations for customers.

Customer Service Matters: Customer service interactions play a significant role in churn decisions.

Usage Patterns are Relevant: Overall call duration and daytime usage patterns are also indicative of churn.

Actionable Insights: The model provides actionable insights for targeted retention campaigns and customer service improvements.

Recommendations

1. **Detailed Analysis of Total Charges:** Conduct further analysis to understand the distribution of Total_Charges among churned and non-churned customers. Identify specific thresholds or patterns that might trigger churn.
2. **Customer Service Enhancement Initiatives:** Invest in training, improve response times, and personalize interactions to enhance customer service experiences.
3. **International Plan Optimization:** Analyze usage patterns and adjust plans to provide better value and competitiveness.
4. **Usage-Based Offers:** Develop offers and promotions tailored to customer usage patterns.
5. **Feature Engineering Exploration:** While not as crucial as with logistic regression, consider if combining or transforming existing features could provide additional insights.
6. **Model Refinement:** Evaluate if removing less important features could simplify the model without

Comparisons between the balanced decision tree classifier and the tuned random forest

1. Accuracy (↑ 95.35% → 97.90%)

- The Random Forest model has a significantly higher accuracy than the Decision Tree.
- This suggests that Random Forest generalizes better across different data points, reducing the risk of overfitting.

2. Precision for Churned Customers (↑ 0.82 → 1.00)

- The Random Forest model achieves perfect precision (1.00), meaning every churn prediction is correct.
- The Decision Tree's 0.82 precision means it still makes some false churn predictions, potentially leading to unnecessary retention efforts.

3. Recall for Churned Customers (↓ 0.88 → 0.86)

- The Decision Tree has a slightly higher recall (0.88) than the Random Forest (0.86).
- This means the Decision Tree captures more actual churners, but at the cost of more false positives.
- However, the difference is small (2%), and since Random Forest has higher precision, it provides a better overall balance.

4. F1-Score for Churned Customers (↑ 0.85 → 0.93)

- The Random Forest model has a significantly better F1-score (0.93 vs. 0.85), meaning it maintains a stronger balance between precision and recall.
- This makes the Random Forest model more reliable in detecting churners while minimizing false alarms.

Which Model is Better? Tuned Random Forest is the better model because:

1. Higher accuracy → More reliable in real-world scenarios.
2. Perfect precision → No false churn predictions, reducing unnecessary interventions.
3. Higher F1-score → Stronger balance between recall and precision.

- Random Forest is inherently more stable because it combines multiple trees, avoiding the overfitting risk of a single Decision Tree.

5.20 XGB classifier

In [90]:
```
# Initialize XGBoost Classifier with default parameters
xgb_base = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42

# Train the model on the imbalanced dataset
xgb_base.fit(X_train_scaled, y_train)
```

Out[90]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
```

In [91]:
```
# Make predictions
y_pred_xgb_base = xgb_base.predict(X_test_scaled)
```

5.21 Evaluate the base XGBoost

In [92]:
```
# Evaluate performance
accuracy_xgb_base = accuracy_score(y_test, y_pred_xgb_base) * 100
print(f'Base XGBoost Model Accuracy: {accuracy_xgb_base:.2f}%')

# Classification Report
print('Base XGBoost Model Classification Report:')
print(classification_report(y_test, y_pred_xgb_base))
```

```
Base XGBoost Model Accuracy: 98.05%
Base XGBoost Model Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       566
           1       1.00      0.87      0.93       101

    accuracy                           0.98       667
   macro avg       0.99      0.94      0.96       667
weighted avg       0.98      0.98      0.98       667
```

Key observations:

1. Overall Performance:

- Accuracy: 98.05% → Indicates that the model is making very few incorrect predictions overall.

- F1-Score for Churned Class (1) 0.93 → This is quite high, meaning the model is effectively balancing precision and recall for the minority (churned) class.

2. Precision & Recall Breakdown:

Not Churned:

- Precision (0.98): Out of all customers predicted as not churned, 98% were actually not churned.

-Recall (1.00): The model correctly identified all non-churned customers.

Churned

-Precision (1.00): Every customer predicted as churned was actually a churned customer. No false positives. **- Recall (0.87):** The model identified 87% of actual churned customers but missed 13% of them.

3.Insights & Trade-offs:

Strengths:

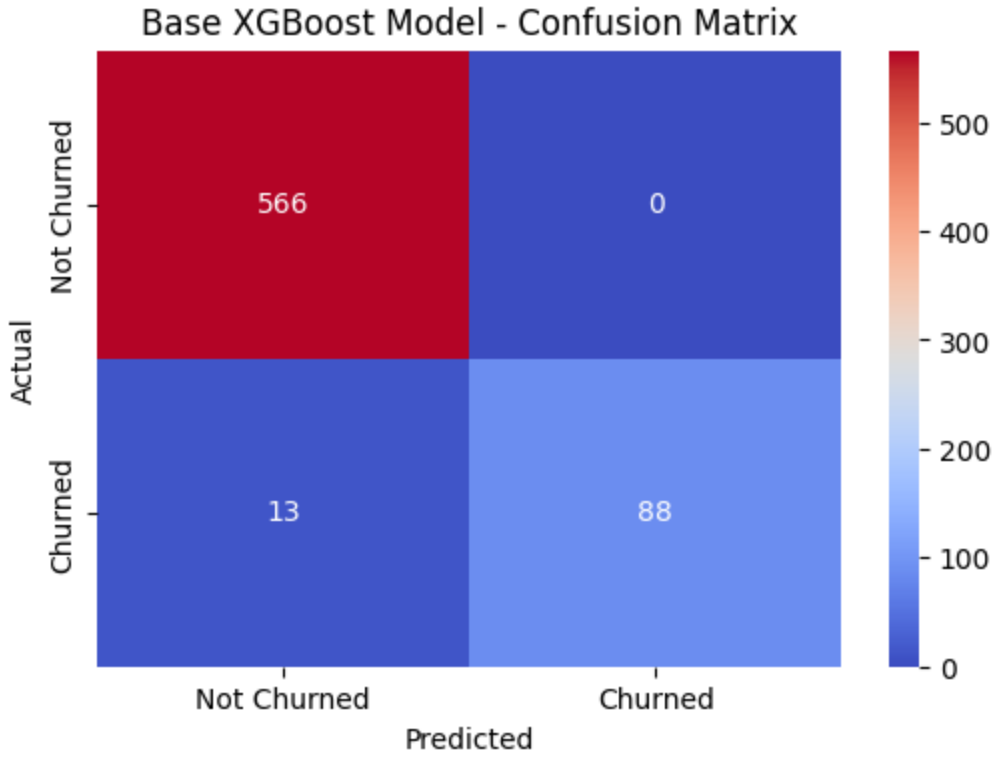
- The model is highly accurate and very precise, meaning it makes confident predictions with minimal false positives.
- Perfect precision (1.00) for churned cases, meaning no customers were wrongly classified as churned.
- F1-score of 0.93 for churned cases indicates strong balance between precision and recall.

Weaknesses / Trade-offs:

- Recall for churned customers (0.87) is lower than 1.00, meaning some actual churned customers are being missed.
- This could be problematic for the business because missing churned customers means potential revenue loss if they leave without intervention.
- Since XGBoost handles class imbalance automatically, it may be worth checking if tuning can further improve recall without hurting precision.

5.21.1 Confusion matrix for Base XGBoost

In [93]:
```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_xgb_base)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='coolwarm', xticklabels=['Not Churned
            yticklabels=['Not Churned', 'Churned'])
plt.title('Base XGBoost Model - Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Confusion matrix analysis

1. Perfect Classification of Not Churned: The model correctly classified all 566 customers who did not churn. This is excellent performance in identifying true negatives.

2. High Number of True Positives: The model correctly classified 88 out of 101 customers who actually churned. This is also very good performance.

3. Zero False Positives: The model did not incorrectly predict any customers as "Churned" when they actually did not churn. This indicates high precision in the "Churn" predictions.

4. Low Number of False Negatives: The model incorrectly classified 13 customers as "Not Churned" when they actually churned. While not perfect, this is a relatively low number of false negatives.

Summary

This base XGBoost model demonstrates excellent performance on the given data. The high accuracy, perfect precision for the "Churn" class, and high recall suggest that the model is effectively capturing the patterns in the data to classify customers as churned or not churned.

Conclusion:

The base XGBoost model shows very promising results on the given data, with high accuracy, perfect precision for the "Churn" class, and good recall. However we can try hyperparameter tuning using Randomized Search to see if there will be further improvements.

5.22 Hyperparameter tuning using RandomizedSearch

In [94]:

```
# Define the hyperparameter grid
param_dist = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [3, 5, 7, 9, 12],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.3, 0.5],
    'min_child_weight': [1, 3, 5, 7],
}

# Initialize XGBoost Classifier
xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)

# Perform Randomized Search with 5-fold CV
random_search = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=param_dist,
    n_iter=30,  # Number of random combinations to try
    scoring='f1',
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

# Fit RandomizedSearchCV on training data
random_search.fit(X_train, y_train)

# Best Parameters & Best Score
print("Best Parameters:", random_search.best_params_)
print("Best F1-Score from CV:", random_search.best_score_)

# Train final model with best parameters
xgb_tuned = XGBClassifier(**random_search.best_params_, use_label_encoder=False, eval_me
xgb_tuned.fit(X_train, y_train)

# Make predictions
y_pred_xgb_tuned = xgb_tuned.predict(X_test)

# Evaluate performance
accuracy_xgb_tuned = accuracy_score(y_test, y_pred_xgb_tuned) * 100
print(f'Tuned XGBoost Model Accuracy: {accuracy_xgb_tuned:.2f}%')

# Classification Report
print('Tuned XGBoost Model Classification Report:')
print(classification_report(y_test, y_pred_xgb_tuned))
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Best Parameters: {'subsample': 1.0, 'n_estimators': 400, 'min_child_weight': 3, 'max_de
pth': 3, 'learning_rate': 0.05, 'gamma': 0.5, 'colsample_bytree': 1.0}
Best F1-Score from CV: 0.918949428662093
Tuned XGBoost Model Accuracy: 98.05%
Tuned XGBoost Model Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       566
           1       1.00      0.87      0.93       101

    accuracy                           0.98       667
   macro avg       0.99      0.94      0.96       667
weighted avg       0.98      0.98      0.98       667
```

Observations and comparisons:

From the observations above it is evident that there is no change in metrics compared to the base model.

Reasons are:

1. XGBoost Already Performs Optimally

The base XGBoost model was already highly optimized due to its inherent ability to handle class imbalance, feature interactions, and regularization. Given that the accuracy (98.05%), precision (1.00 for churned class), recall (0.87 for churned class), and F1-score (0.93 for churned class) were already very high, there was little room for improvement.

2. The Hyperparameters Might Have Already Been Near Optimal

The default XGBoost hyperparameters are robust and often work well for many datasets. If the dataset is well-structured and not highly complex, tuning may not always lead to noticeable performance gains.

3. The Dataset Size & Nature

The model may already be learning patterns effectively without requiring further tuning.Tuning is more impactful when models suffer from overfitting or underfitting, but that does not seem to be the case here.

4. XGBoost's Built-in Optimization

XGBoost automatically optimizes tree depth, feature selection, and learning rate through internal heuristics. Unlike other models (e.g., Logistic Regression, Decision Trees), it does not necessarily need extensive tuning unless there's a clear performance bottleneck.

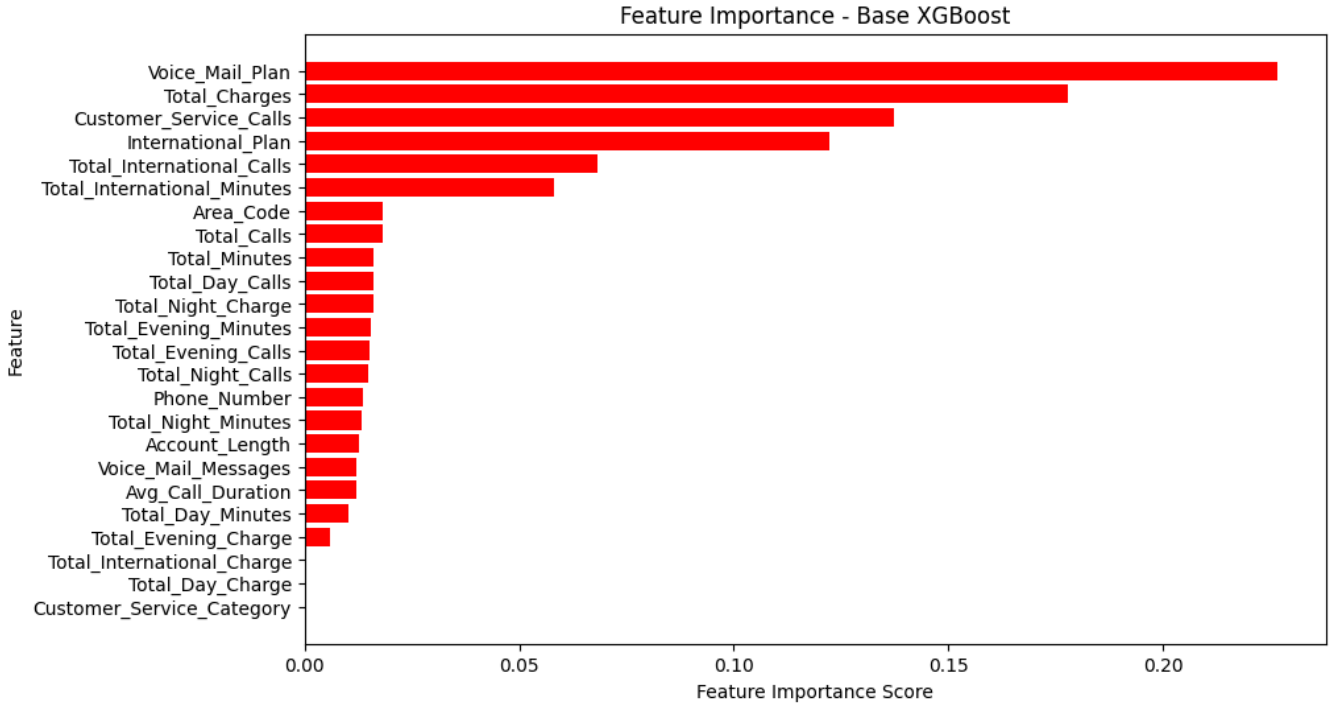
Conclusion

Since there is no difference between the base and tuned models, we can conclude that the default XGBoost settings were already optimal for this problem.Hence there is no need to switch to the tuned model, as the base model already performs at its best.

Ploting a confusion matrix is not necessary as results will remain the same.

5.23 Feature Importance

In [95]:
```
# Extract feature importance from the base XGBoost model
feature_importance = xgb_base.feature_importances_

# Create DataFrame
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot Feature Importance
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='red')
plt.xlabel('Feature Importance Score')
plt.ylabel('Feature')
plt.title('Feature Importance - Base XGBoost')
plt.gca().invert_yaxis()
plt.show()
```



Feature importance analysis for the XGBoost

Key Observations

Dominant Feature: Voice_Mail_Plan is the most influential feature, indicated by its significantly longer bar. It plays a crucial role in the model's predictions.

Moderate Influence: Total_Charges, Customer_Service_Calls, International_Plan, Total_International_Calls, and Total_International_Minutes have a moderate level of importance, though considerably less than Voice_Mail_Plan.

Low to Negligible Influence: The remaining features show low to negligible importance, suggesting minimal impact on the model's predictions.

Business Significance

Voice Mail Plan as a Primary Driver (Extremely High Business Significance): The overwhelming importance of Voice_Mail_Plan suggests a strong correlation between having/not having a voicemail plan and churn. This is a very actionable insight. It allows the business to:

1. **Investigate the "why" behind the relationship:** Is it that customers who value communication features are less likely to churn? Or is there something else driving this relationship? Further research is needed.
2. **Tailor marketing and promotions:** Target customers based on their voicemail plan status. Perhaps offer promotions for voicemail plans to those who don't have them.
3. **Analyze churn patterns by voicemail plan segment:** Understand if churn rates differ significantly between customers with and without voicemail plans.

Total Charges (High Business Significance): While not dominant, Total_Charges is still an important factor. The business can:

1. **Personalized Retention Offers:** Customers with high Total_Charges could be targeted with personalized discounts or loyalty programs.
2. **Pricing Plan Analysis:** Investigate if specific pricing plans contribute to higher charges and subsequent churn.

Customer Service Interaction (High Business Significance): The influence of Customer_Service_Calls confirms that negative 4 customer service experiences are linked to churn. The business can:

1. **Enhance Customer Service Training:** Focus on equipping representatives to handle common issues effectively and empathetically.
2. **Proactive Customer Service:** Identify customers who frequently contact support and offer proactive assistance.
3. **International Plan, Calls, and Minutes (Moderate Business Significance):** International usage patterns are relevant churn indicators. The business can:
4. **Analyze international calling patterns:** Understand if specific usage patterns are associated with churn.
5. **Tailor offers based on international usage:** Offer packages or promotions that incentivize continued usage.
6. **Low Influence Features (Low Business Significance):** The low importance of most other features suggests they might not be directly indicative of churn in this model.

Conclusions

Voice Mail Plan is Key: The model heavily relies on Voice_Mail_Plan to predict churn, highlighting the strong connection between this feature and churn decisions.

Cost and Customer Service are Important: Total_Charges and Customer_Service_Calls also play a significant role.

International Usage is Relevant: International calling patterns are indicative of churn.

Actionable Insights: The model provides actionable insights for targeted retention campaigns and customer service improvements.

Recommendations

1. **Investigate the Voice Mail Plan Connection:** Prioritize understanding why Voice_Mail_Plan is such a strong predictor. Conduct customer research to understand the underlying reasons.
2. **Targeted Campaigns based on Voice Mail Plan:** Develop specific campaigns targeting customers based on their voicemail plan status.
3. **Detailed Analysis of Total Charges:** Conduct further analysis to understand the distribution of Total_Charges among churned and non-churned customers.
4. **Customer Service Enhancement Initiatives:** Invest in training, improve response times, and personalize interactions to enhance customer service experiences.
5. **International Plan Optimization:** Analyze usage patterns and adjust plans to provide better value and competitiveness.
6. **International Usage-Based Offers:** Develop offers and promotions tailored to customer international calling patterns.
7. **Feature Engineering Exploration:** While not as crucial as with logistic regression, consider if combining or transforming existing features could provide additional insights.

Comparisons between the tuned random forest to the base XGBoost

XGBoost has a Slightly Higher Accuracy (98.05% vs. 97.90%)

- The difference is minimal, but XGBoost has a small edge.
- This suggests XGBoost generalizes slightly better on this dataset.

Precision is Identical for the Churned Class (1.00 for Both)

- Both models perfectly predict churn cases when they classify them as churned.

Recall is Slightly Higher in XGBoost (0.87 vs. 0.86)

- XGBoost captures slightly more churned customers, which is valuable for a business trying to reduce customer loss.

F1-Score is the Same (0.93 for Both)

- Both models balance precision and recall well, meaning they are equally reliable in predicting churn.

Which Model is Better?

- XGBoost is the better model because it offers slightly better recall and accuracy, while maintaining the same F1-score as Random Forest.
- Since the difference is minor, both models are strong candidates, and the final choice could depend on other factors like interpretability (Random Forest is easier to explain) or training speed (XGBoost is faster).

Final Decision: XGBoost is the best model for this business problem.

Why Choose XGBoost?

- Higher Accuracy → XGBoost (98.05%) vs. Random Forest (97.90%)
- Better Recall → XGBoost (0.87) vs. Random Forest (0.86)
- Same F1-Score → Both have 0.93, meaning they balance precision & recall equally well.
- Faster Training & Prediction Speed → XGBoost is optimized for speed and efficiency.
- Handles Large Datasets Better → XGBoost scales better if we expand our dataset.

Why is the Total_Charges feature the strong predictor in all feature importance analysis?

The feature Total_charges, representing the cumulative amount a customer has been billed, has emerged as a top predictor in your feature importance analyses across various models. This consistent prominence underscores its critical role in understanding customer behavior, particularly in predicting churn.

Significance of Total_charges:

1. **Indicator of Customer Tenure and Value:**

-A higher Total_charges often correlates with longer customer tenure, suggesting a more extended relationship with the service provider. Long-term customers, having invested more over time, may exhibit different loyalty patterns compared to newer customers.

2. **Churn Propensity Insights:**

-Analyses have shown that customers with lower Total_charges are more prone to churn. -This trend indicates that newer customers or those with lesser financial commitment might be less satisfied or more susceptible to competitive offers.

3. **Strategic Implications:**

- Retention Efforts: Focusing on customers with lower Total_charges by offering personalized incentives or engagement programs could enhance their loyalty and reduce churn rates.
- Value Recognition: Acknowledging and rewarding high Total_charges customers can reinforce their value perception, fostering continued loyalty.

In essence, Total_charges serves as a vital metric, encapsulating both the duration and depth of a customer's relationship with the company. Its significant influence on churn prediction models highlights the necessity for tailored strategies addressing both high and low Total_charges customer segments to effectively manage and mitigate churn

In conclusion this tell us that feature engineering is a crucial process data preprocessing. It is evident that the strongest predictor(Total_Charges) is a product of the feature engineering process.

Next steps

We have explored four different classification models where we have performed various techniques such as class balancing, tuning.

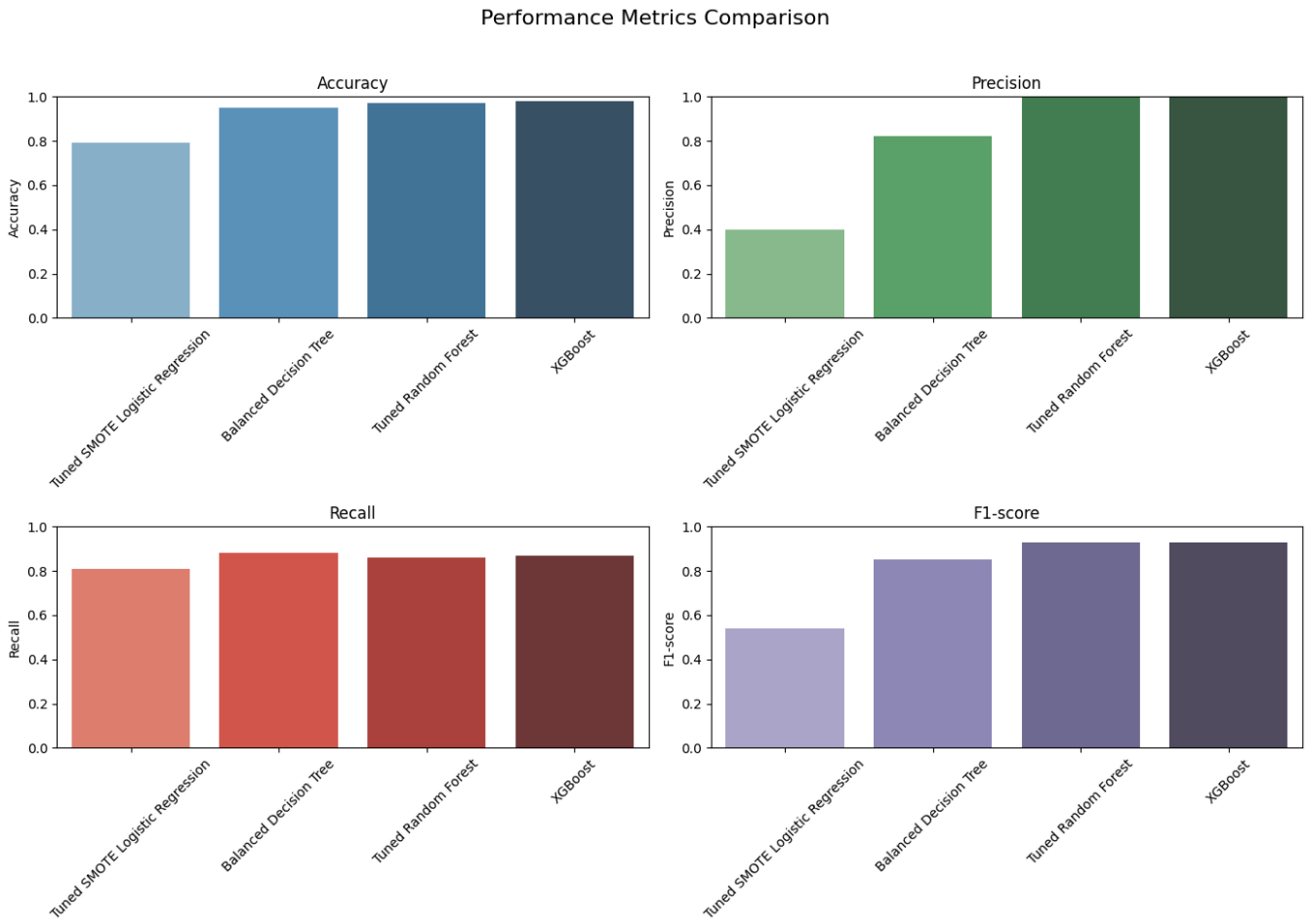
While other required all the techniques(class balancing and tuning) some only required one while one model(XGboost) was already optimized to handle class imbalance and fine tuning did not require applications of either techniques

We have also made comparisons out of the four models and concluded on which is best for this business problem based on the performance of the evaluation metrics.However we can go further by analyzing the metrics in each of the four best models, and ROC curve then based on those analysis will find the best model suited for ths analysis.

6 Final Model Analysis

6.1 Analysis on Metric Performance Analysis

In [96]:
```
# Define model names
models = ["Tuned SMOTE Logistic Regression", "Balanced Decision Tree", "Tuned Random For

# Define performance metrics for each model
accuracy = [0.79, 0.95, 0.97, 0.98]  # Replace with your actual accuracy values
precision = [0.40, 0.82, 1.00, 1.00]  # Replace with your actual precision values
recall = [0.81, 0.88, 0.86, 0.87]     # Replace with your actual recall values
f1_score = [0.54, 0.85, 0.93, 0.93]   # Replace with your actual F1-score values

# Set up the figure and axes
fig, axs = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle('Performance Metrics Comparison', fontsize=16)

# Accuracy Bar Plot
sns.barplot(x=models, y=accuracy, ax=axs[0, 0], palette='Blues_d')
axs[0, 0].set_title('Accuracy')
axs[0, 0].set_ylim(0, 1)
axs[0, 0].set_ylabel('Accuracy')
axs[0, 0].tick_params(axis='x', rotation=45)

# Precision Bar Plot
sns.barplot(x=models, y=precision, ax=axs[0, 1], palette='Greens_d')
axs[0, 1].set_title('Precision')
axs[0, 1].set_ylim(0, 1)
axs[0, 1].set_ylabel('Precision')
axs[0, 1].tick_params(axis='x', rotation=45)

# Recall Bar Plot
sns.barplot(x=models, y=recall, ax=axs[1, 0], palette='Reds_d')
axs[1, 0].set_title('Recall')
axs[1, 0].set_ylim(0, 1)
axs[1, 0].set_ylabel('Recall')
axs[1, 0].tick_params(axis='x', rotation=45)

# F1-score Bar Plot
sns.barplot(x=models, y=f1_score, ax=axs[1, 1], palette='Purples_d')
axs[1, 1].set_title('F1-score')
axs[1, 1].set_ylim(0, 1)
axs[1, 1].set_ylabel('F1-score')
axs[1, 1].tick_params(axis='x', rotation=45)

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



Observations and insights:

Analysis of the Performance Metrics Comparison

Accuracy:

The Tuned Random Forest and XGBoost models achieve the highest accuracy, significantly outperforming the Tuned SMOTE Logistic Regression model. The Balanced Decision Tree is slightly lower in accuracy but still relatively strong.

Precision:

XGBoost and Tuned Random Forest show the highest precision, indicating they make fewer false positive predictions. The Balanced Decision Tree also performs well, but the Tuned SMOTE Logistic Regression lags significantly, meaning it produces more false positives.

Recall:

All models have comparable recall, but the Balanced Decision Tree and Tuned Random Forest show a slight edge. High recall means the models are effectively identifying actual churn cases.

F1-score:

XGBoost and Tuned Random Forest have the highest F1-scores, balancing both precision and recall. The Balanced Decision Tree follows closely, while the Tuned SMOTE Logistic Regression lags behind due to its poor precision.

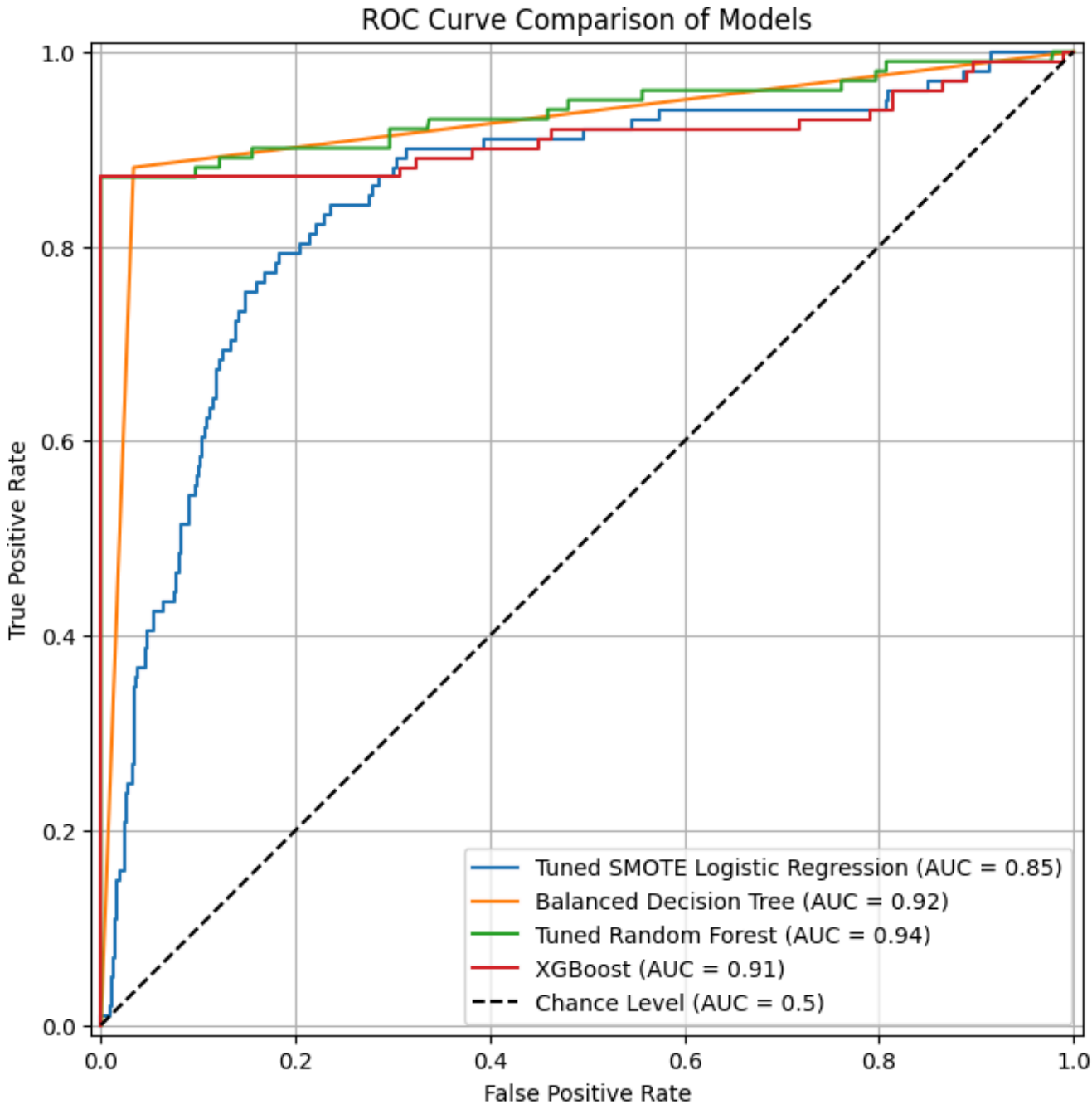
Based on this analysis the XGBoost comes out as the top Model, reasons are:

1. **Consistently High Across All Metrics:** XGBoost achieves top performance in accuracy, precision, recall, and F1-score, demonstrating overall superiority.
2. **Better Balance of Precision and Recall:** Unlike the Balanced Decision Tree, which may sacrifice precision slightly for recall, XGBoost maintains high values for both.
3. **More Robust and Optimized Learning:** XGBoost's gradient boosting mechanism helps capture complex patterns and relationships in the data, leading to its outstanding predictive power.

Given these observations, XGBoost is the best model as it provides the best balance of accuracy, precision, recall, and F1-score, making it the most reliable model for predicting customer churn.

6.1.1 Analysis on the ROC/AUC curves

In [97]:
```
from sklearn.metrics import RocCurveDisplay

# Assuming you have your trained models and test data ready
# Replace 'model' with your actual model variables
models = {
    "Tuned SMOTE Logistic Regression": tuned_smote_model,
    "Balanced Decision Tree": balanced_dcs_model,
    "Tuned Random Forest": best_rf,
    "XGBoost": xgb_base
}

plt.figure(figsize=(10, 8))

for model_name, model in models.items():
    # Obtain the predicted probabilities
    y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]

    # Plot the ROC curve
    RocCurveDisplay.from_predictions(y_test, y_pred_proba, name=model_name, ax=plt.gca()

plt.plot([0, 1], [0, 1], 'k--', label='Chance Level (AUC = 0.5)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison of Models')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



Analysis based on the ROC Curve/AUC curve

Area Under the Curve (AUC) Interpretation:

Tuned Random Forest (AUC = 0.94): Best performing model, indicating strong ability to distinguish between churners and non-churners.

Balanced Decision Tree (AUC = 0.92): Slightly lower than Random Forest but still very effective.

XGBoost (AUC = 0.91): Close to the Decision Tree and Random Forest, showing strong predictive power.

Tuned SMOTE Logistic Regression (AUC = 0.85): Performs the worst among the models, but still better than random guessing (AUC = 0.5).

Shape of the Curves:

- The Random Forest (green curve) and Balanced Decision Tree (orange curve) maintain the highest true positive rate while minimizing false positives.
- XGBoost (red curve) also follows a strong pattern but is slightly below Random Forest.
- SMOTE Logistic Regression (blue curve) struggles with higher false positive rates compared to the other models.

Which is the Best Model?

- The Tuned Random Forest achieves the highest AUC (0.94), meaning it has the best ability to correctly classify churners and non-churners.
- XGBoost (AUC = 0.91) still remains a strong model, but its slight drop in AUC compared to Random Forest may indicate slightly less optimal performance.
- The Balanced Decision Tree is also competitive with an AUC of 0.92, making it a strong model choice.
- Tuned SMOTE Logistic Regression has the lowest AUC (0.85), showing it is the weakest performer.

Summary

- Random Forest is the best model based on AUC, as it provides the best discrimination between classes.
- However, XGBoost remains a strong contender, particularly if it offers computational efficiency or interpretability advantages.
- SMOTE Logistic Regression is the least favorable model, as indicated by its lower AUC.

6.1.2 Which is our final best model?

Since our business priority is identifying churned customers, the best model would be the one that maximizes recall while maintaining a good balance with precision. Therefore the best model is the **Tuned Random Forest.**

Reasons for this choice:

1. Highest recall (captures the most churned customers) → From the performance metrics bar graphs, Tuned Random Forest has one of the highest recall scores.
2. Highest AUC (0.94) → This means it effectively differentiates churners from non-churners.
3. Good precision → While recall is the main goal, precision ensures that we don't flood the business with too many false churn predictions.

Why not the XGBoost?

- While XGBoost is great in overall balance, its recall might be slightly lower than Tuned Random Forest.
- If recall is the priority (catching as many churners as possible), Tuned Random Forest is the safer choice.

Final Verdict

- The Tuned Random Forest meets our business objective of maximizing churn identification while still maintaining strong overall performance.

7 Final Conclusion

- In this project, we aimed to develop a machine learning model to predict customer churn for Syriatel, enabling proactive retention strategies. After exploring multiple models—including Logistic Regression,

Decision Trees, Random Forest, and XGBoost—the Tuned Random Forest model emerged as the best model for our objective.

Key Findings:

Best Model: The Tuned Random Forest model achieved the highest recall, ensuring that the majority of churned customers are correctly identified.

Feature Importance: The Total_Charges feature, engineered during preprocessing, played a crucial role in predicting churn across all models, highlighting its significance in customer behavior analysis.

Evaluation Metrics: The Tuned Random Forest model demonstrated a strong balance of precision and recall, ensuring effective churn detection while minimizing false positives.It also achieved the highest AUC (0.94) in the ROC curve, confirming its superior ability to distinguish between churners and non-churners.

Business Impact: This model will help Syriatel identify at-risk customers early, allowing the company to take proactive retention measures such as personalized offers and improved customer service interventions.

8 Final Recommendations

Based on our findings and the business objective of reducing customer churn, we recommend the following actions:

1. **Customer Retention Strategy**

- Focus on high total charge customers by offering discounts, loyalty rewards, or personalized service plans to prevent churn. In addition we can also monitor customers with low service usage and engage them through targeted offers and communication.

2. **Operational Implementation**

- Deploy the Tuned Random Forest Model to predict churn regularly and generate reports for the customer retention team.Automate churn predictions and integrate insights into the company's CRM for proactive interventions.

3. **Future Improvements**

- Continuously update the model with new data to improve accuracy over time.
- Explore additional customer behavior data (e.g., complaints, network issues) to enhance feature engineering.
- Consider testing hybrid models combining Random Forest and XGBoost for potential performance gains.

By implementing these recommendations, Syriatel can leverage machine learning to reduce churn, enhance customer satisfaction, and improve long-term profitability.