Business Understanding

Introduction

For telecommunications companies such as Syria Tel, the loss of paying subscribers is a major challenge. This leads to a reduction in revenue, increased costs associated with the acquisition of customers and diminished brand loyalty. SyriaTel needs a strong and datadriven approach to proactively tackle customer turnover and increase retention of customers.

Problem Statement

The goal of this project is to use SyriaTel's customer data to create a model for predicting customer churn. SyriaTel will be able to implement targeted retention strategies and reduce customer attrition by using the model to identify customers who are at high risk of churning.

Business Objectives

- Forecasting customer churn to lower client attrition.
- Focusing on high-risk consumers for retention efforts.
- Increasing customer lifetime value by focusing on significant revenue contributors.
- Enhancing client experience to increase satisfaction and loyalty.
- Using model insights for customized marketing campaigns to re-engage potential customers.

Data Description

The project will make use of a dataset from SyriaTel's customer relationship management (CRM) system that contains previous customer data. Features like user demographics, account information, call usage trends, and customer service interactions are anticipated to be included in this data.

Expected Benefits

- Better customer experience: By anticipating problems and knowing how customers behave, businesses can build loyalty and improve the overall customer experience.
- Data-driven decision making: Strategic customer retention initiatives and resource allocation are guided by insights.
- Customized marketing initiatives: Models help with campaign customization for particular clientele groups.
- Proactive customer engagement: By addressing customer complaints and preventing churn, early detection of churn risk helps to prevent it.

Data Understanding

Dataset Overview

The customer relationship management (CRM) system of SyriaTel provides historical customer data in the dataset. Customer demographics, account information, phone usage trends, and customer service exchanges are among the features.

Structure and Features

- **Features:**
  - State of residence
  - Account length
  - Service plans (e.g., international plan, voice mail plan)
  - Call usage metrics (e.g., total minutes, number of calls)
  - Customer service interactions

Potential Issues

- Missing values: Determine appropriate handling procedures after looking for any missing values in the dataset.
- Outliers: Spot any outliers in numerical features and assess whether further action is required.
- Data discrepancies: Check categorical characteristics for errors or inconsistencies that could impact analysis.

Data Preprocessing

To address missing numbers, outliers, and inconsistencies, clean up the dataset. carrying out required transformations, such as scaling numerical characteristics and encoding categorical variables.

Data Exploration

Overview

The first analysis of the SyriaTel customer churn dataset aimed to understand its properties and potential connections to customer attrition. Service plans, consumption patterns, and client demographics are all included in the dataset. Examining feature distribution, spotting anomalies, and figuring out possible connections between variables and the target variable were the key goals.

Importing Libraries

In [13]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
from scipy import stats

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectFromModel

from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,
from sklearn.metrics import accuracy_score, classification_report, confusi

import warnings
warnings.filterwarnings("ignore")
```

Loading CSV data

In [34]:
```
# Load the data from a CSV file
df = pd.read_csv("churn.csv")

# Display first 10 rows
df.head(10)
```

Out[34]:

state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	c
KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	
OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	
NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	
OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	
OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	
AL	118	510	391-8027	yes	no	0	223.4	98	37.98	...	101	
MA	121	510	355-9993	no	yes	24	218.2	88	37.09	...	108	
MO	147	415	329-9001	yes	no	0	157.0	79	26.69	...	94	
LA	117	408	335-4719	no	no	0	184.5	97	31.37	...	80	
WV	141	415	330-8173	yes	yes	37	258.6	84	43.96	...	111	

ows × 21 columns

In [28]:
```
1 df.describe()
```

Out[28]:

y s	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	t
)	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	33
3	100.435644	30.562307	200.980348	100.114311	17.083540	200.872037	100.107711	
9	20.069084	9.259435	50.713844	19.922625	4.310668	50.573847	19.568609	
)	0.000000	0.000000	0.000000	0.000000	0.000000	23.200000	33.000000	
)	87.000000	24.430000	166.600000	87.000000	14.160000	167.000000	87.000000	
)	101.000000	30.500000	201.400000	100.000000	17.120000	201.200000	100.000000	
)	114.000000	36.790000	235.300000	114.000000	20.000000	235.300000	113.000000	
)	165.000000	59.640000	363.700000	170.000000	30.910000	395.000000	175.000000	

In [17]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account length          3333 non-null   int64
 2   area code               3333 non-null   int64
 3   phone number            3333 non-null   object
 4   international plan      3333 non-null   object
 5   voice mail plan         3333 non-null   object
 6   number vmail messages   3333 non-null   int64
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
```

In [ ]:
```
1
```

The **info()** method was used to load the dataset and examine its contents. It was found that each column included 3333 non-null entries. This suggests that the dataset is free of missing values.

In [20]:
```
1 df.isnull().sum()
```

Out[20]:
```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

The **df.isnull().sum()** method was used to further verify that there were no missing values, and it returned 0 to indicate this. We can confidently move forward with additional research and modeling now that there are no missing values.

Exploratory Data Analysis

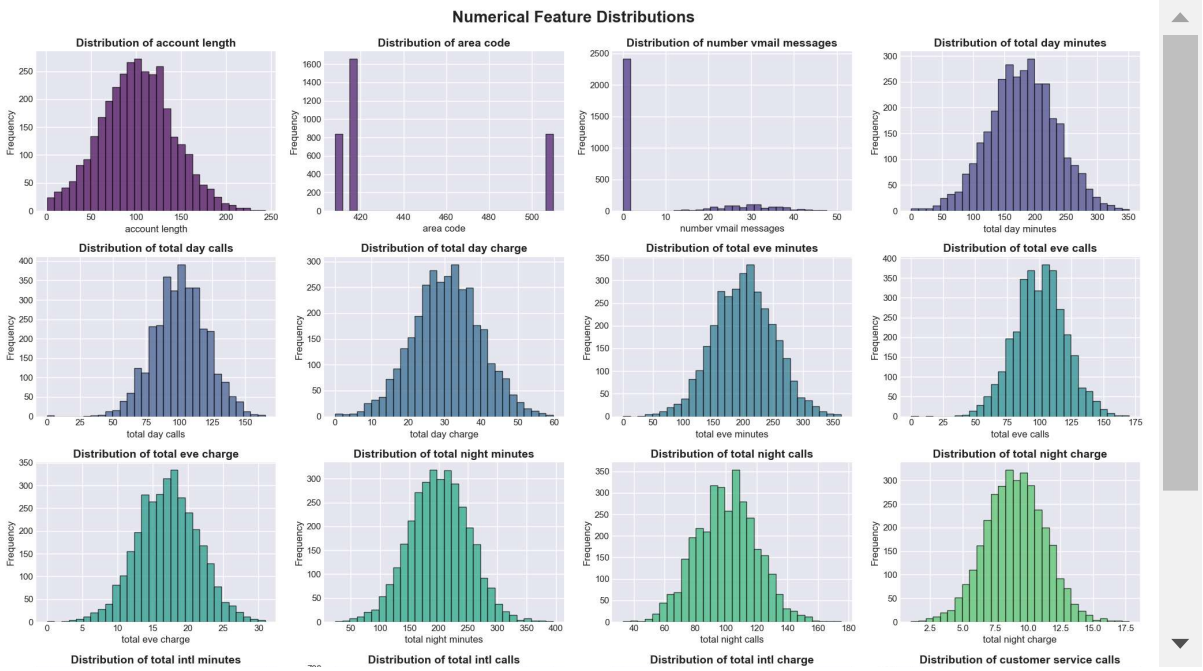
1. Univariate Analysis

Individual variables were analyzed to understand the distribution of the independent variables.

Visualisations

In [35]:

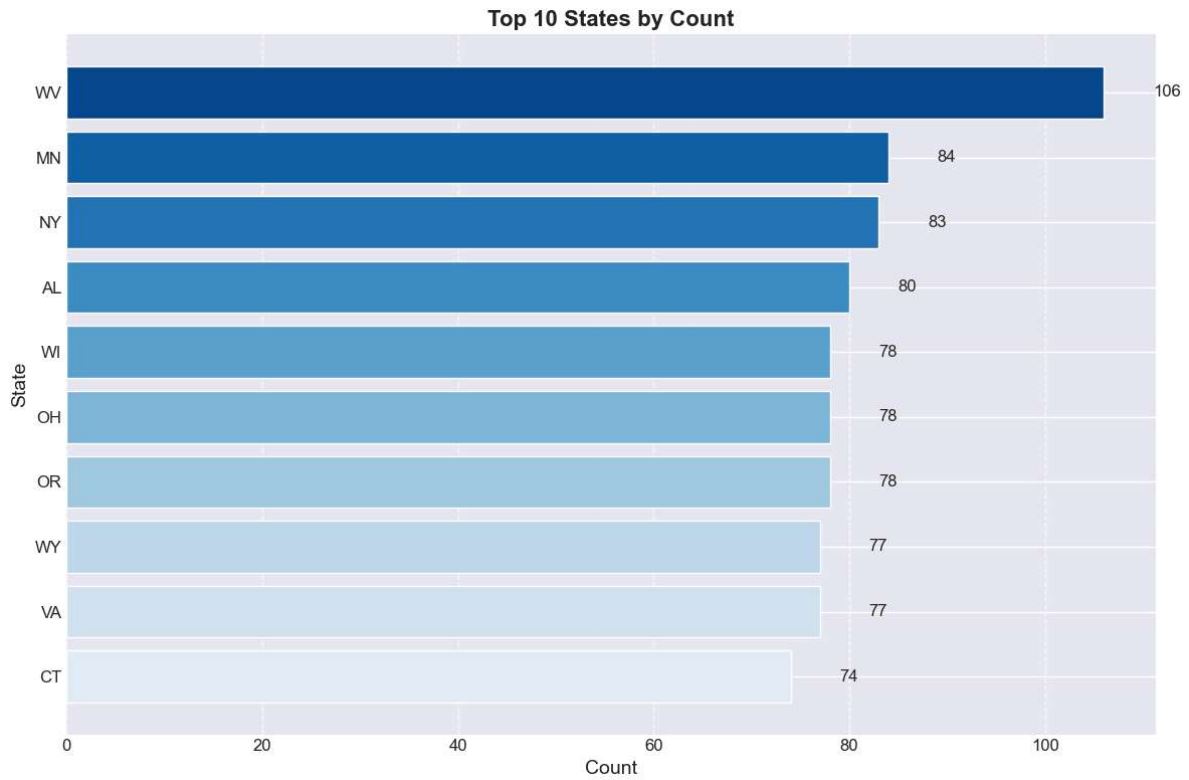
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# Load the data
df = pd.read_csv("churn.csv")

# Select only numerical features
numerical_features = df.select_dtypes(include=[np.number]).columns.tolist(

# Calculate the number of rows and columns needed for subplots
num_features = len(numerical_features)
num_cols = 4
num_rows = math.ceil(num_features / num_cols)

# Set the style
plt.style.use('seaborn-darkgrid')

# Create a color palette
colors = plt.cm.viridis(np.linspace(0, 1, num_features))

# Visualize numerical features
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 15))
axes = axes.flatten()  # Flatten the axes array to make indexing easier

for i, (column, color) in enumerate(zip(numerical_features, colors)):
    ax = axes[i]
    ax.hist(df[column].dropna(), bins=30, color=color, alpha=0.7, edgecolo
    ax.set_title(f'Distribution of {column}', fontsize=14, fontweight='bol
    ax.set_xlabel(column, fontsize=12)
    ax.set_ylabel('Frequency', fontsize=12)
    ax.grid(True)

# Remove empty subplots
for j in range(i + 1, num_rows * num_cols):
    fig.delaxes(axes[j])

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.suptitle('Numerical Feature Distributions', fontsize=20, fontweight='b
plt.show()

```



The numerical data showed a right-skewed pattern in customer tenure and call durations across different time periods, suggesting lower usage patterns. International calls may have higher volumes for a smaller customer segment. Customer service calls showed a multimodal distribution, indicating distinct groups with varying frequency of contacting customer service. Understanding these relationships and potential outliers can help predict churn.

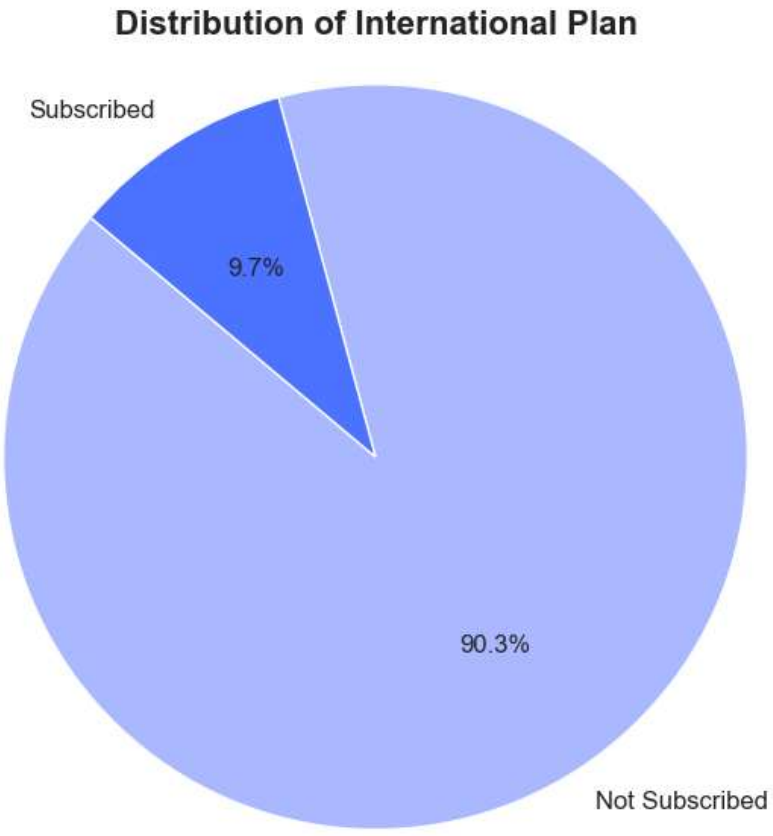
In [43]:

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

# Get the top ten states by count
top_states = df['state'].value_counts().sort_values(ascending=False).head(

# Plotting
plt.figure(figsize=(12, 8))
bars = plt.barh(top_states.index, top_states.values, color=sns.color_palet

# Adding annotations
for bar, count in zip(bars, top_states.values):
    plt.text(bar.get_width() + 5, bar.get_y() + bar.get_height()/2, f'{cou
             va='center', ha='left', fontsize=12)

# Customizing plot aesthetics
plt.title('Top 10 States by Count', fontsize=16, fontweight='bold')
plt.xlabel('Count', fontsize=14)
plt.ylabel('State', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.gca().invert_yaxis()  # Invert y-axis to have the highest count at the
plt.tight_layout()

plt.show()

```



Customers primarily from West Virginia, Minnesota, and New York make up SyriaTel's customer base.

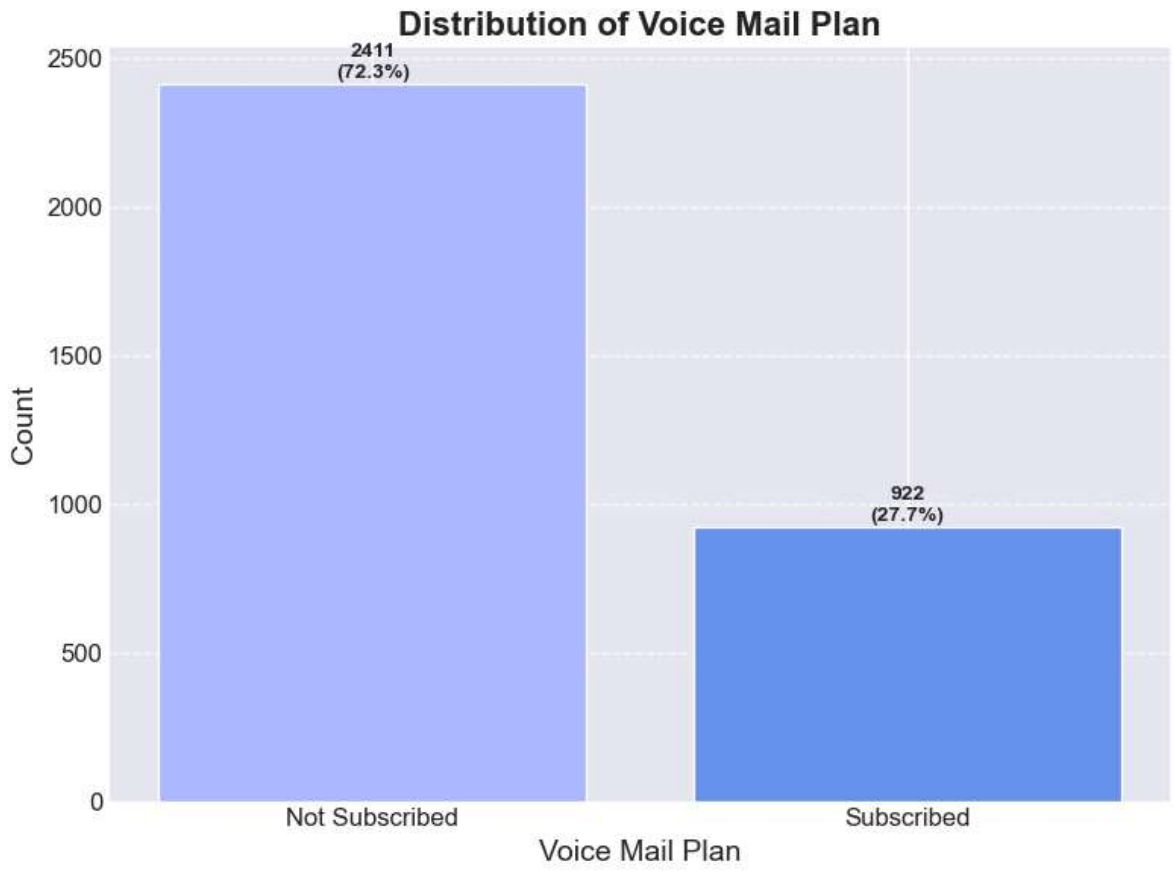
In [56]:
```
import matplotlib.pyplot as plt

# Define custom colors
light_blue = '#4c74ff'
lighter_blue = '#aabaff'

# Calculate the percentage of each category
international_plan_counts = df['international plan'].value_counts()
labels = ['Subscribed' if x == 'yes' else 'Not Subscribed' for x in interr
sizes = international_plan_counts.values
colors = [lighter_blue, light_blue]

# Plotting
plt.figure(figsize=(8, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle
plt.title('Distribution of International Plan', fontsize=16, fontweight='b
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a cir
plt.tight_layout()

plt.show()

```



only 9.7% are subcribed to the international plan.

In [55]:

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

# Replace 'yes' and 'no' with 'Subscribed' and 'Not Subscribed'
df['voice mail plan'] = df['voice mail plan'].replace({'yes': 'Subscribed

# Calculate counts for each category
counts = df['voice mail plan'].value_counts()

# Plotting a bar plot
plt.figure(figsize=(8, 6))
bars = plt.bar(counts.index, counts, color=[lighter_blue, light_blue])

# Adding annotations
for bar, count in zip(bars, counts):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 5, f'{cou
             ha='center', va='bottom', fontsize=10, fontweight='bold')

# Customizing plot aesthetics
plt.title('Distribution of Voice Mail Plan', fontsize=16, fontweight='bold
plt.xlabel('Voice Mail Plan', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

plt.show()

```



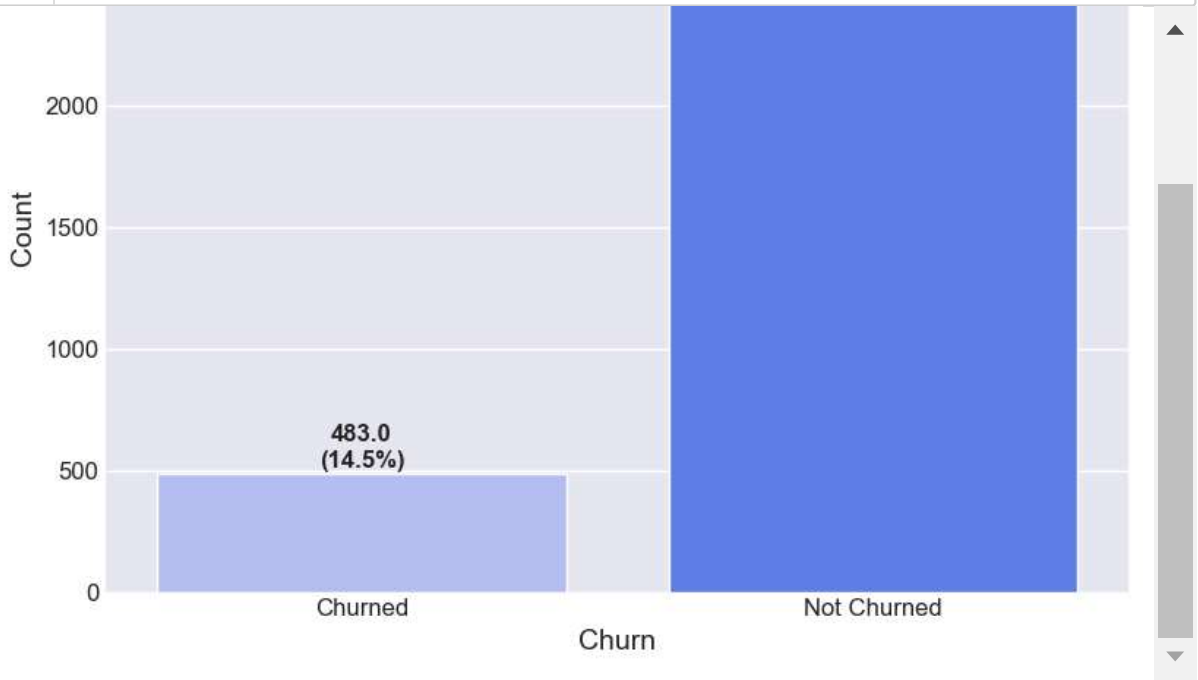
only 27.7% of customers have subscribed to voice mail service

Distribution of Churn

The 'churn' variable delineates whether a customer has churned or not. The endeavor encompassed a univariate analysis to elucidate the distribution of churn within the dataset.

In [61]:

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

# Define custom colors
light_blue = '#4c74ff'
lighter_blue = '#aabaff'

# Replace 'True' and 'False' with 'Churned' and 'Not Churned'
df['churn'] = df['churn'].replace({True: 'Churned', False: 'Not Churned'})

# Calculate churn percentages
churn_counts = df['churn'].value_counts()
churn_percentages = churn_counts / churn_counts.sum() * 100

# Plotting
plt.figure(figsize=(8, 6))
ax = sns.countplot(data=df, x='churn', palette=[lighter_blue, light_blue],

# Adding annotations
total = len(df['churn'])  # Total number of records
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2., height + 10, f'{height}\n({hei
            ha='center', va='bottom', fontsize=12, fontweight='bold')

# Title and labels
plt.title('Distribution of Churn', fontsize=16, fontweight='bold')
plt.xlabel('Churn', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Show plot
plt.tight_layout()
plt.show()

```



A detailed analysis of churn was conducted by counting the number of consumers who stopped using the service and the number of customers who continued. The following are the outcomes:

Total number of churned customers: 483 There were 2850 clients that remained loyal.

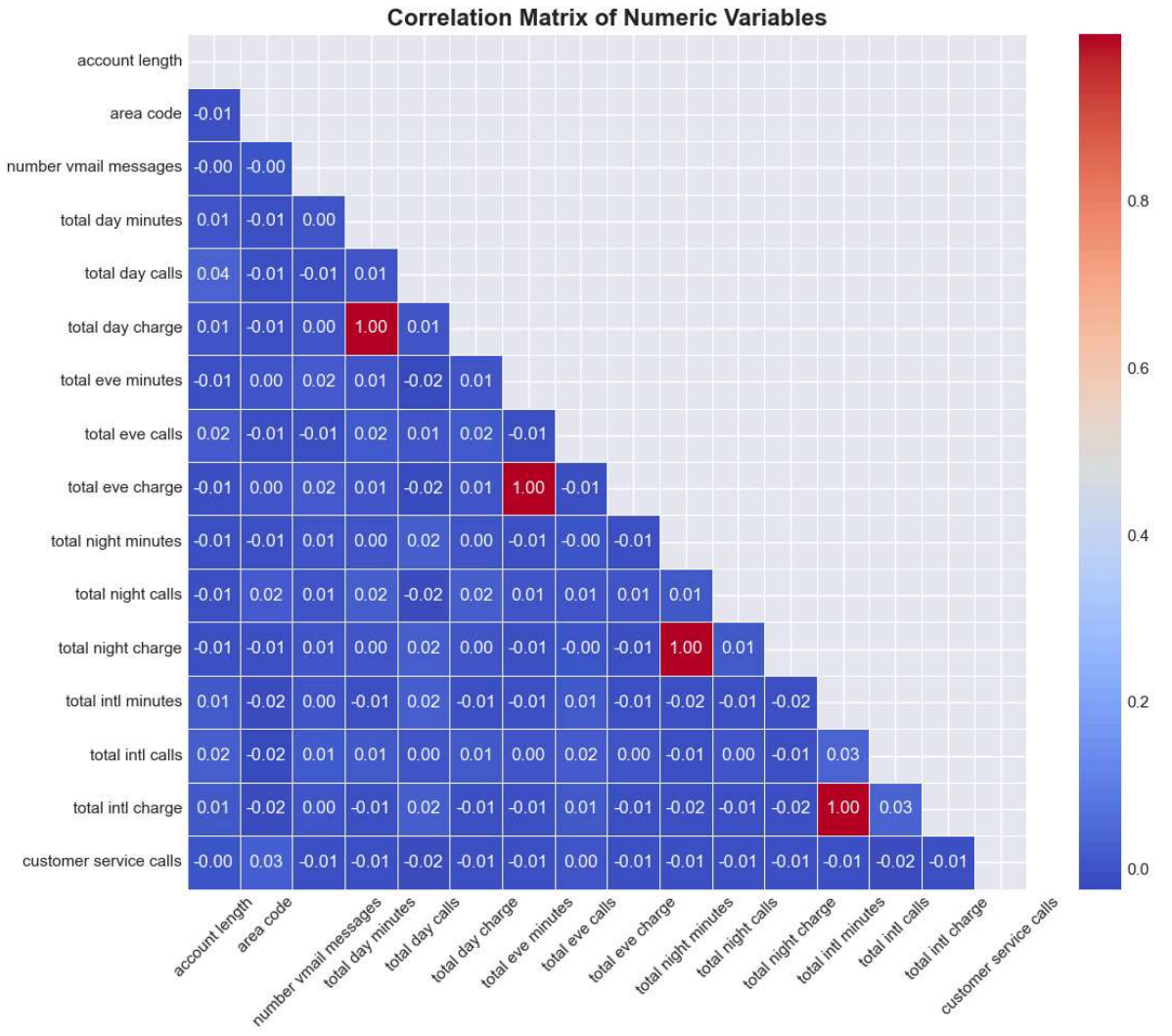
Multivariate analysis

This entailed looking at several variables' relationships at once. This study looked at the relationship between several features and the aim variable (customer churn) when taken into account collectively.

The correlations between the various variables in the dataset were found using a correlation matrix.

In [101]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Select only numeric columns
numeric_df = df.select_dtypes(include=['float64', 'int64'])

# Step 2: Visualize correlation matrix
correlation_matrix = numeric_df.corr()

# Zero out lower triangular part
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool), k=0)

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", li
plt.title('Correlation Matrix of Numeric Variables', fontsize=16, fontweig
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

```



Correlation Matrix Analysis: Call Numbers, Minutes, Charges

- Significant positive relationships between call numbers, minutes, and charges.
- Higher call spending leads to more calls.
- Weak positive link between call minutes and account length.
- Customer service calls show weak correlations, unrelated to calling habits or account tenure.
- International calls show favorable link, leading to increased spending.

Data Preprocessing

Label Encoding converts label variables in "international plan", "voice mail plan", and "churn" columns to numeric form. The "Yes" and "No" categories are converted to 1 and 0, respectively, representing presence or absence. In the "churn" column, "False" and "True" categories are converted to 0 and 1, respectively, representing customer churn.

In [104]:

```
from sklearn.preprocessing import LabelEncoder

# Define categorical columns to be label encoded
cat_cols = ["international plan", "voice mail plan"]

# Apply label encoding to each categorical column
df[cat_cols] = df[cat_cols].apply(LabelEncoder().fit_transform)

# Display the first ten rows to verify the changes
print(df.head(10))
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge
0	KS	128	415	382-4657	0	1	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01	10.0	3	2.70
1	OH	107	415	371-7191	0	1	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45	13.7	3	3.70
2	NJ	137	415	358-1921	0	0	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32	12.2	5	3.29
3	OH	84	408	375-9999	1	0	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86	6.6	7	1.78
4	OK	75	415	330-6626	1	0	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41	10.1	3	2.73
5	AL	118	510	391-8027	1	0	0	223.4	98	37.98	...	101	18.75	203.9	118	9.18	6.3	6	1.70
6	MA	121	510	355-9993	0	1	24	218.2	88	37.09	...	108	29.62	212.6	118	9.57	7.5	7	2.03
7	MO	147	415	329-9001	1	0	0	157.0	79	26.69	...	94	8.76	211.8	96	9.53			
8	LA	117	408	335-4719	0	0	0	184.5	97	31.37	...	80	29.89	215.8	90	9.71			
9	WV	141	415	330-8173	1	1	37	258.6	84	43.96	...	111	18.87	326.4	97	14.69			

```
7                 7.1                6              1.92
8                 8.7                4              2.35
9                11.2                5              3.02
```

```
   customer service calls  churn
0                       1  False
1                       1  False
2                       0  False
3                       2  False
4                       3  False
5                       0  False
6                       3  False
7                       0  False
8                       1  False
9                       0  False

[10 rows x 21 columns]
```

One-hot encoding is a method that converts categorical variables into numerical format for machine learning algorithms, especially useful for dealing with variables with multiple categories.

In [105]:

```
from sklearn.preprocessing import OneHotEncoder

# Create an instance of the OneHotEncoder
encoder = OneHotEncoder(sparse=False, dtype=np.int64)

# Encode the "state" column
encoded_state = encoder.fit_transform(df[["state"]])

# Create a DataFrame with the encoded state columns
dummy_df_state = pd.DataFrame(encoded_state, columns=encoder.get_feature_r

# Concatenate the encoded state columns with the original DataFrame
one_df = pd.concat([df.drop(columns=["state"]), dummy_df_state], axis=1)

one_df.head(10)

```

Out[105]:

ernational plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	...	state_SD	state_TN	state_TX	s
0	1	25	265.1	110	45.07	197.4	...	0	0	0	
0	1	26	161.6	123	27.47	195.5	...	0	0	0	
0	0	0	243.4	114	41.38	121.2	...	0	0	0	
1	0	0	299.4	71	50.90	61.9	...	0	0	0	
1	0	0	166.7	113	28.34	148.3	...	0	0	0	
1	0	0	223.4	98	37.98	220.6	...	0	0	0	
0	1	24	218.2	88	37.09	348.5	...	0	0	0	
1	0	0	157.0	79	26.69	103.1	...	0	0	0	
0	0	0	184.5	97	31.37	351.6	...	0	0	0	
1	1	37	258.6	84	43.96	222.0	...	0	0	0	

In [107]:
```
# Verify the changes
print(one_df.head(10))
```

```
   account length  area code phone number  international plan  \
0             128        415     382-4657                   0
1             107        415     371-7191                   0
2             137        415     358-1921                   0
3              84        408     375-9999                   1
4              75        415     330-6626                   1
5             118        510     391-8027                   1
6             121        510     355-9993                   0
7             147        415     329-9001                   1
8             117        408     335-4719                   0
9             141        415     330-8173                   1

   voice mail plan  number vmail 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
5                0                      0              223.4               98
6                1                     24              218.2               88
7                0                      0              157.0               79
8                0                      0              184.5               97
9                1                     37              258.6               84

   total day charge  total eve minutes  ...  state_SD  state_TN  state_TX  \
0             45.07              197.4  ...         0         0         0
1             27.47              195.5  ...         0         0         0
2             41.38              121.2  ...         0         0         0
3             50.90               61.9  ...         0         0         0
4             28.34              148.3  ...         0         0         0
5             37.98              220.6  ...         0         0         0
6             37.09              348.5  ...         0         0         0
7             26.69              103.1  ...         0         0         0
8             31.37              351.6  ...         0         0         0
9             43.96              222.0  ...         0         0         0

   state_UT  state_VA  state_VT  state_WA  state_WI  state_WV  state_WY
0         0         0         0         0         0         0         0
1         0         0         0         0         0         0         0
2         0         0         0         0         0         0         0
3         0         0         0         0         0         0         0
4         0         0         0         0         0         0         0
5         0         0         0         0         0         0         0
6         0         0         0         0         0         0         0
7         0         0         0         0         0         0         0
8         0         0         0         0         0         0         0
9         0         0         0         0         0         1         0

[10 rows x 71 columns]
```

In [114]:
```
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# Example DataFrame
df = pd.DataFrame({
    'area code': [408, 415, 408, 510],
    'other_column': [1, 2, 3, 4]
})

# Check if "area code" column exists in the DataFrame
if "area code" in df.columns:
    # One-hot encoding for "area code" column and updating the DataFrame
    encoder = OneHotEncoder(sparse_output=False)
    encoded = encoder.fit_transform(df[["area code"]])
    df = df.join(pd.DataFrame(encoded, columns=encoder.get_feature_names_

df.head(10)

```

Out[114]:

	other_column	area code_408	area code_415	area code_510
0	1	1.0	0.0	0.0
1	2	0.0	1.0	0.0
2	3	1.0	0.0	0.0
3	4	0.0	0.0	1.0

Scaling numerical features is important for algorithms that are sensitive to feature scales, such as gradient descent-based algorithms and distance-based algorithms.

In [115]:
```
newdf = one_df
```

In [121]:
```
newdf.dtypes
```

Out[121]:
```
account length           int64
area code                int64
international plan       int64
voice mail plan          int64
number vmail messages    int64
                         ...
state_VT                 int64
state_WA                 int64
state_WI                 int64
state_WV                 int64
state_WY                 int64
Length: 70, dtype: object
```

In [123]:

```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Define numeric columns
numeric_columns = newdf.select_dtypes(include=['number']).columns

# Drop rows with NaN values in numeric columns
newdf.dropna(subset=numeric_columns, inplace=True)

# Check if there are numeric columns to scale
if numeric_columns.empty:
    print("No numeric columns found")
else:
    # Clean and scale numeric columns
    scaler = MinMaxScaler()
    newdf[numeric_columns] = scaler.fit_transform(newdf[numeric_columns].a

    # Print the first few rows of the scaled DataFrame
    print(newdf[numeric_columns].head(10))
```

	account length	area code	international plan	voice mail plan	\
0	0.524793	0.068627	0.0	1.0	
1	0.438017	0.068627	0.0	1.0	
2	0.561983	0.068627	0.0	0.0	
3	0.342975	0.000000	1.0	0.0	
4	0.305785	0.068627	1.0	0.0	
5	0.483471	1.000000	1.0	0.0	
6	0.495868	1.000000	0.0	1.0	
7	0.603306	0.068627	1.0	0.0	
8	0.479339	0.000000	0.0	0.0	
9	0.578512	0.068627	1.0	1.0	

	number vmail messages	total day minutes	total day calls	\
0	0.490196	0.755701	0.666667	
1	0.509804	0.460661	0.745455	
2	0.000000	0.693843	0.690909	
3	0.000000	0.853478	0.430303	
4	0.000000	0.475200	0.684848	
5	0.000000	0.636830	0.593939	
6	0.470588	0.622007	0.533333	
7	0.000000	0.447548	0.478788	
8	0.000000	0.525941	0.587879	
9	0.725490	0.737172	0.509091	

	total day charge	total eve minutes	total eve calls	...	state_SD	\
0	0.755701	0.542755	0.582353	...	0.0	
1	0.460597	0.537531	0.605882	...	0.0	
2	0.693830	0.333242	0.647059	...	0.0	
3	0.853454	0.170195	0.517647	...	0.0	
4	0.475184	0.407754	0.717647	...	0.0	
5	0.636821	0.606544	0.594118	...	0.0	
6	0.621898	0.958207	0.635294	...	0.0	
7	0.447518	0.283475	0.552941	...	0.0	
8	0.525989	0.966731	0.470588	...	0.0	
9	0.737089	0.610393	0.652941	...	0.0	

	state_TN	state_TX	state_UT	state_VA	state_VT	state_WA	state_WI	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	state_WV	state_WY
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0
5	0.0	0.0
6	0.0	0.0
7	0.0	0.0

```
8       0.0       0.0
9       1.0       0.0

[10 rows x 69 columns]
```

splitting the data to guarantee an objective assessment of the models ie: the data must be divided into training and testing sets before modeling can begin.

In [125]:

```
from sklearn.model_selection import train_test_split

# Define the features (X) and target variable (y)
X = newdf.drop(columns=['churn'])  # Features
y = newdf['churn']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# Print the shapes of the training and testing sets
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (2666, 69)
y_train shape: (2666,)
X_test shape: (667, 69)
y_test shape: (667,)
```

class imbalance can be averted by the use of random oversampling through the resample function addresses class imbalance, ensuring a fairer representation of data and potentially enhancing the model's performance in handling imbalanced datasets

In [134]:
```
from sklearn.utils import resample
import numpy as np

# Combine X_train and y_train to resample
train_data = np.column_stack((X_train, y_train))

# Separate the majority and minority classes
majority_class = train_data[train_data[:, -1] == np.bincount(y_train).argr
minority_class = train_data[train_data[:, -1] == np.bincount(y_train).argr

# Perform random oversampling on the minority class
minority_class_upsampled = resample(minority_class,
                                    replace=True,  # Sample with replaceme
                                    n_samples=majority_class.shape[0],  #
                                    random_state=42)  # Reproducible resul

# Combine majority class with upsampled minority class
resampled_data = np.vstack((majority_class, minority_class_upsampled))

# Separate X and y
X_train_resampled = resampled_data[:, :-1]
y_train_resampled = resampled_data[:, -1].astype(int)  # Convert y to inte

# Verify the result
print("Original class distribution:", np.bincount(y_train))
print("Resampled class distribution:", np.bincount(y_train_resampled))

```

```
Original class distribution: [2284  382]
Resampled class distribution: [2284 2284]
```

Modelling

Introduction

Modelling is a pivotal stage in every machine learning project, where we train predictive models on available data to accurately predict or decide outcomes for unseen data. In the context of the SyriaTel Customer Churn project, our goal is to develop a classifier that forecasts whether a customer will churn based on various features such as call usage, account details, and customer service interactions.

Training Approaches

For the SyriaTel Customer Churn project, we will utilize multiple machine learning algorithms to train our predictive models. These algorithms encompass:

- **Logistic Regression**: A linear model used for binary classification tasks, predicting the probability of a binary outcome based on predictor variables.
- **Random Forests** are derived from a group of decision trees in which forecasts are generated by voting or average across trees that have been trained on various data subsets.

- **XGBoost**: An optimized implementation of gradient boosting, sequentially building an ensemble of weak learners to enhance predictive performance.

*__Logistic regression__ is a widely used algorithm for binary classification tasks, like predicting customer churn, by estimating the probability of an instance belonging to a specific class. The logistic regression model is trained using numerical and encoded categorical features to predict customer churn. Its performance is evaluated using accuracy, precision, recall, F1-score, and ROC-AUC Score. Accuracy measures the proportion of correctly classified instances, precision measures the proportion of true positive predictions, recall measures the proportion of true positive predictions, F1-score provides a balanced measure, and ROC-AUC Score measures the model's ability to distinguish between positive and negative instances.

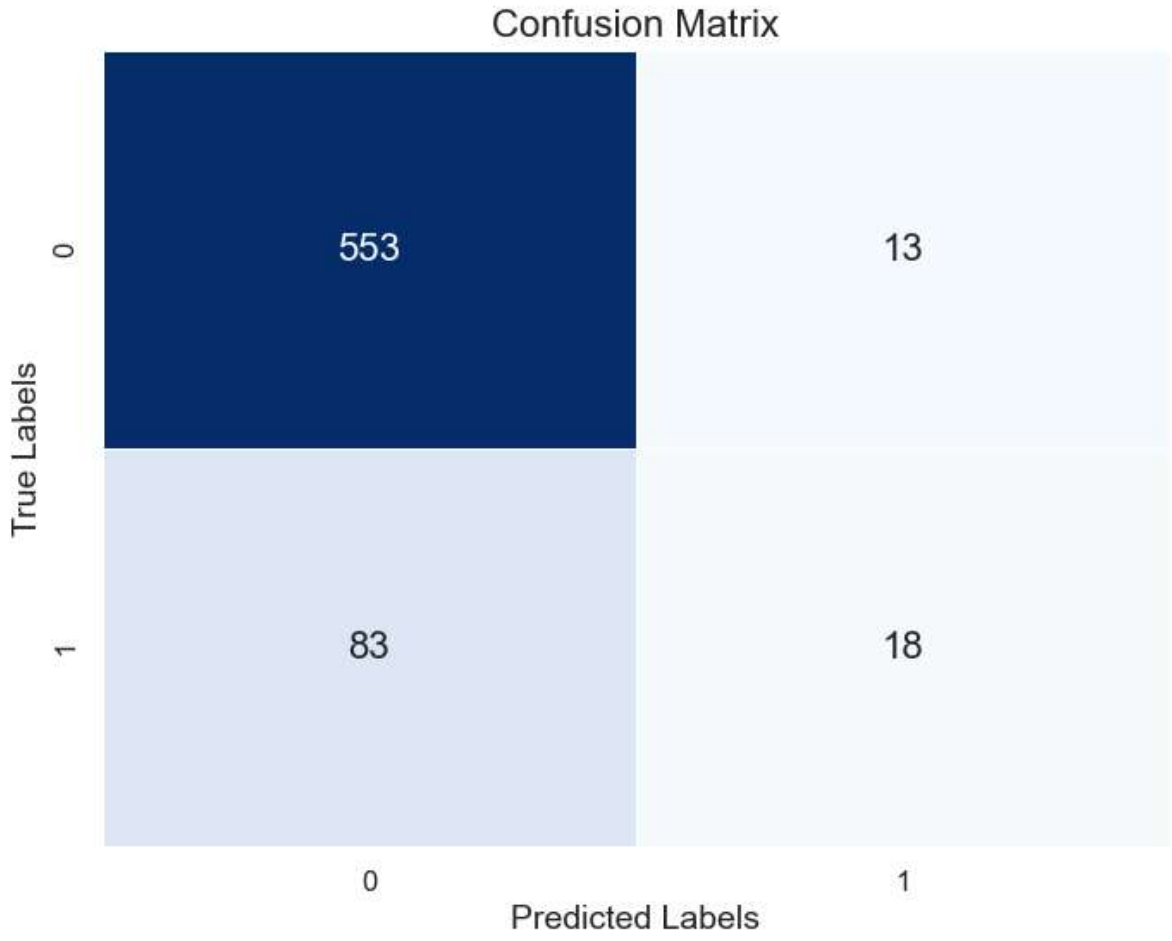
We employ cross-validation to obtain robust estimates of the logistic regression model's performance. Cross-validation assesses the model's generalization ability and mitigates the risk

In [135]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Instantiate the Logistic Regression model
log_reg_model = LogisticRegression(random_state=42)

# Train the model
log_reg_model.fit(X_train, y_train)

# Predict on the test set
y_pred = log_reg_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.856071964017991

Classification Report:
              precision    recall  f1-score   support

       False       0.87      0.98      0.92       566
        True       0.58      0.18      0.27       101

    accuracy                           0.86       667
   macro avg       0.73      0.58      0.60       667
weighted avg       0.83      0.86      0.82       667
```

In [141]:

```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            annot_kws={"size": 16}, linewidths=0.5)
plt.xlabel('Predicted Labels', fontsize=14)
plt.ylabel('True Labels', fontsize=14)
plt.title('Confusion Matrix', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



Despite having an accuracy of 85%, the logistic regression model's recall and precision for predicting churn are both poor at 18% and 58%, respectively, suggesting a large miss of actual churn cases. The model might not be the best option for this classification assignment, according to these results.

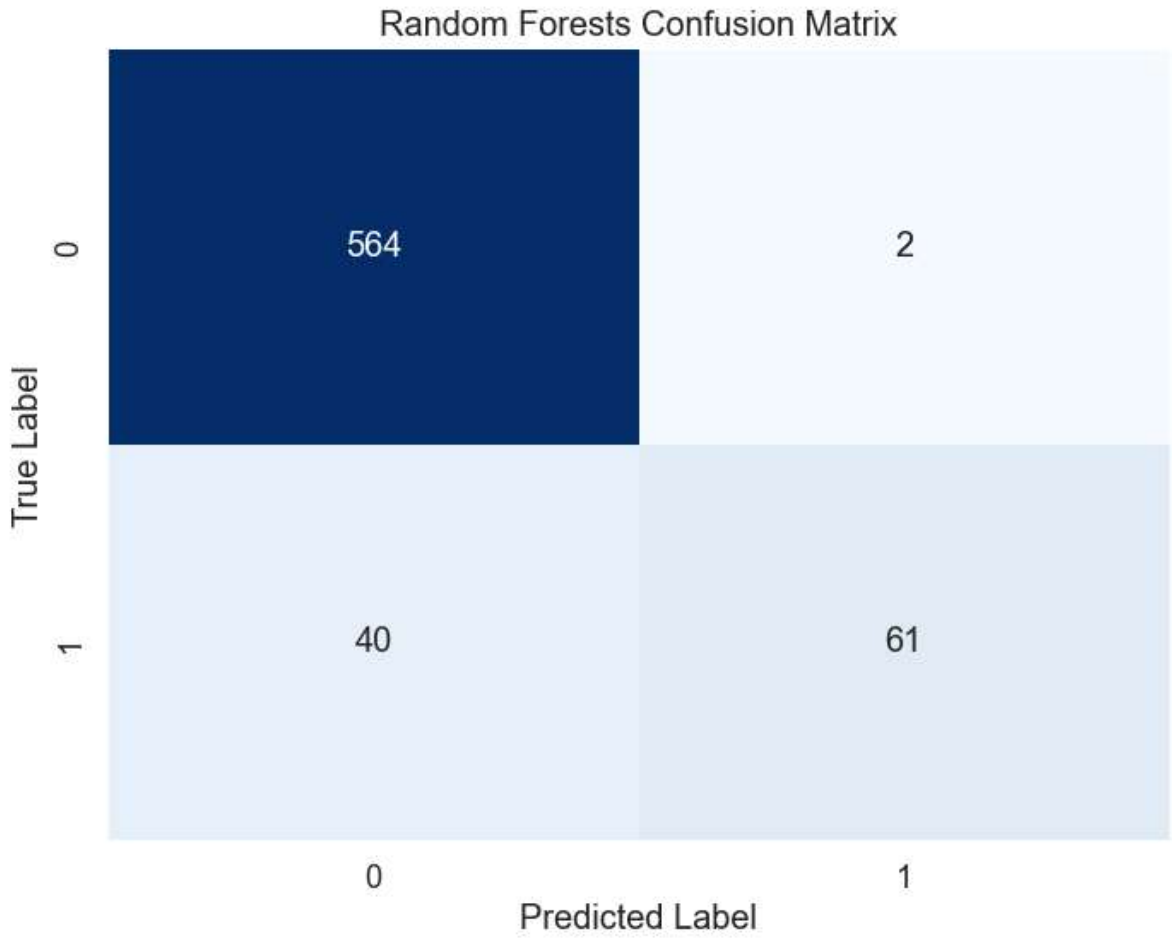
Random Forests

In [147]:

```
from sklearn.ensemble import RandomForestClassifier

# Initialize Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Train the model
rf_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred_rf = rf_classifier.predict(X_test)

# Calculate accuracy for Random Forests
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Accuracy:", accuracy_rf)

# Generate classification report for Random Forests
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))
```

```
Accuracy: 0.9370314842578711

Classification Report:
              precision    recall  f1-score   support

       False       0.93      1.00      0.96       566
        True       0.97      0.60      0.74       101

    accuracy                           0.94       667
   macro avg       0.95      0.80      0.85       667
weighted avg       0.94      0.94      0.93       667
```

In [148]:
```
# Generate confusion matrix for Random Forests
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Random Forests Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



The model exhibits high accuracy at 93.7%, with high precision, recall, and F1-score in the majority class ('False'). However, it tends to miss some 'True' samples, with a lower recall at 60%. The weighted averages show robust performance across both classes.

XG Boost

In [144]:

```
from xgboost import XGBClassifier

# Initialize XGBoost classifier
xgb = XGBClassifier()

# Train the model
xgb.fit(X_train, y_train)

# Predict on the test set
y_pred_xgb = xgb.predict(X_test)

# Calculate accuracy
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print("Accuracy:", accuracy_xgb)

# Generate classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred_xgb))
```

```
Accuracy: 0.9610194902548725

Classification Report:
              precision    recall  f1-score   support

       False       0.96      0.99      0.98       566
        True       0.96      0.77      0.86       101

    accuracy                           0.96       667
   macro avg       0.96      0.88      0.92       667
weighted avg       0.96      0.96      0.96       667
```

In [169]:
```
# Generate confusion matrix
conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_xgb, annot=True, fmt="d", cmap="Blues", cbar=False
plt.title("Confusion Matrix - XGBoost")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

This XGBoost model demonstrates high precision, indicating that when it predicts a customer will churn (True), it is correct 96% of the time. However, its recall for churn is lower at 77%

Hyperparameter Tuning

We are going to select the two best performing models and tune them

In [170]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

# Define classifiers
classifiers = {
    "Logistic Regression": LogisticRegression(),
    "Random Forests": RandomForestClassifier(),
    "XGBoost": XGBClassifier()
}

# Define performance metrics
performance_metrics = {}

# Evaluate each classifier using cross-validation
for clf_name, clf in classifiers.items():
    # Perform cross-validation
    cv_precision = cross_val_score(clf, X_train, y_train, cv=5, scoring='p
    cv_recall = cross_val_score(clf, X_train, y_train, cv=5, scoring='reca

    # Compute average precision and recall
    avg_precision = np.mean(cv_precision)
    avg_recall = np.mean(cv_recall)

    # Store the performance metrics in the dictionary
    performance_metrics[clf_name] = {
        "Precision": avg_precision,
        "Recall": avg_recall
    }

# Sort models based on their average precision in descending order
sorted_models = sorted(performance_metrics.items(), key=lambda x: x[1]["Pr

# Select the top two models
best_two_models = sorted_models[:2]

# Print the performance metrics for the top two models
for clf_name, metrics in best_two_models:
    print(f"{clf_name}:")
    print(f"  Precision: {metrics['Precision']:.4f}")
    print(f"  Recall: {metrics['Recall']:.4f}")
    print()

```

```
Random Forests:
  Precision: 0.9716
  Recall: 0.5862

XGBoost:
  Precision: 0.9036
  Recall: 0.7460
```

Tunng Random Forests

In [154]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define Random Forest model
rf_model = RandomForestClassifier()

# Define hyperparameters grid for Random Forest
rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform GridSearchCV for Random Forest
rf_grid = GridSearchCV(estimator=rf_model, param_grid=rf_param_grid, scori
rf_grid.fit(X_train, y_train)

# Get best parameters and best score for Random Forest
best_rf_params = rf_grid.best_params_
best_rf_score = rf_grid.best_score_

# Create new Random Forest model with best parameters
best_rf_model = RandomForestClassifier(**best_rf_params)
best_rf_model.fit(X_train, y_train)

# Evaluate best Random Forest model
best_rf_model_score = best_rf_model.score(X_test, y_test)
```

Tuning XGBoost

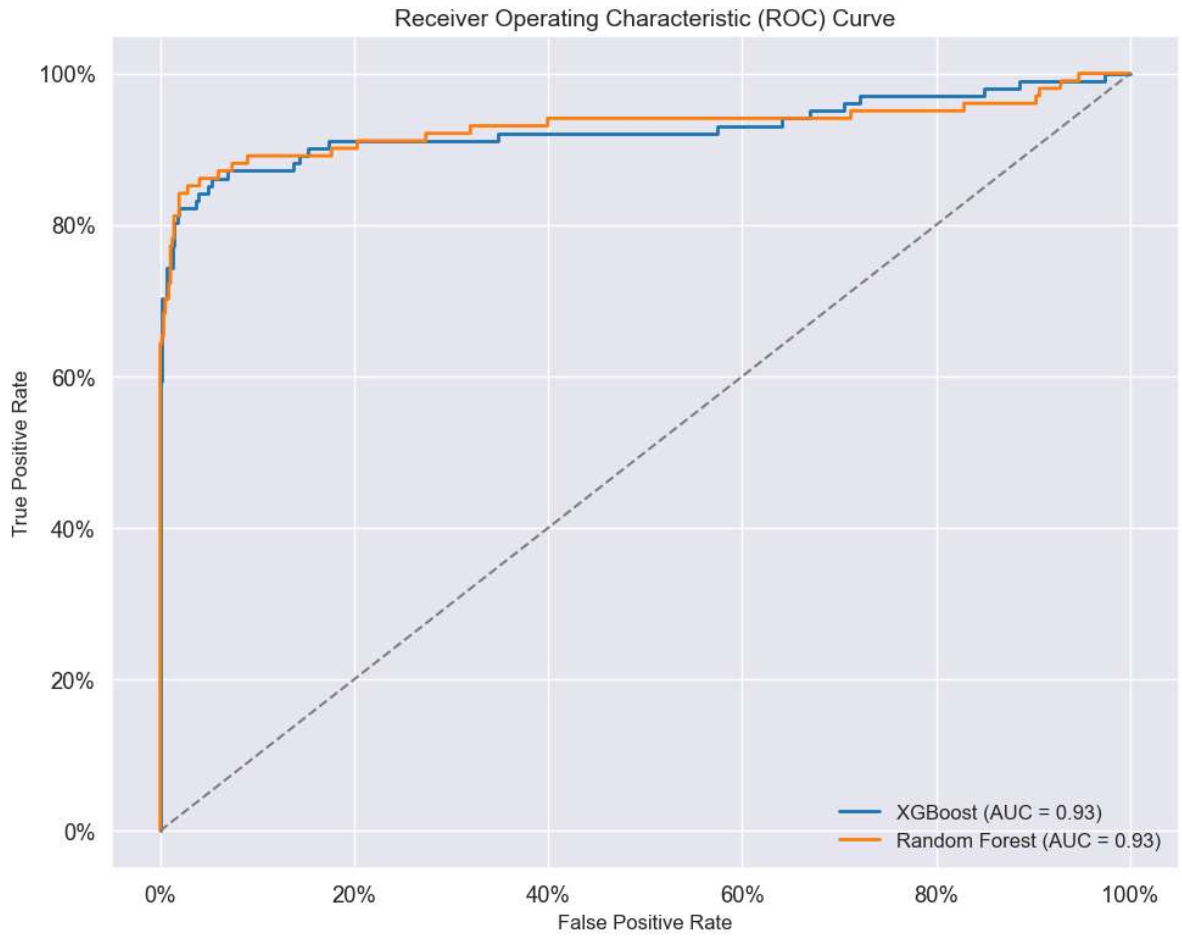
In [155]:

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

# Define XGBoost model
xgb_model = XGBClassifier()

# Define hyperparameters grid for XGBoost
xgb_param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'n_estimators': [100, 200, 300]
}

# Perform GridSearchCV for XGBoost
xgb_grid = GridSearchCV(estimator=xgb_model, param_grid=xgb_param_grid, sc
xgb_grid.fit(X_train, y_train)

# Get best parameters and best score for XGBoost
best_xgb_params = xgb_grid.best_params_
best_xgb_score = xgb_grid.best_score_

# Create new XGBoost model with best parameters
best_xgb_model = XGBClassifier(**best_xgb_params)
best_xgb_model.fit(X_train, y_train)

# Evaluate best XGBoost model
best_xgb_model_score = best_xgb_model.score(X_test, y_test)
```

The below ROC curve helps visualize the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity) across different threshold values, aiding in assessing the performance of a binary classifier and determining the optimal classification threshold. It enhances model interpretability and provides valuable insights into classifier discrimination.

In [160]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from matplotlib.ticker import PercentFormatter

# Get predicted probabilities for XGBoost
xgb_probs = xgb_grid.predict_proba(X_test)[:, 1]
fpr_xgb, tpr_xgb, _ = roc_curve(y_test, xgb_probs)
roc_auc_xgb = auc(fpr_xgb, tpr_xgb)

# Get predicted probabilities for Random Forest
rf_probs = rf_grid.predict_proba(X_test)[:, 1]
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_probs)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Set up the figure and axes
plt.figure(figsize=(10, 8))

# Plot ROC curves with specific colors and linewidths
plt.plot(fpr_xgb, tpr_xgb, color='#1f77b4', lw=2, label=f'XGBoost (AUC = 
plt.plot(fpr_rf, tpr_rf, color='#ff7f0e', lw=2, label=f'Random Forest (AUC

# Plot ROC curve for random classifier (baseline)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

# Customize ticks on x-axis to percentage
plt.gca().xaxis.set_major_formatter(PercentFormatter(1))
plt.gca().yaxis.set_major_formatter(PercentFormatter(1))

# Set plot labels and legend
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('Receiver Operating Characteristic (ROC) Curve', fontsize=14)
plt.legend(loc='lower right', fontsize=12)
plt.grid(True)

# Tight layout
plt.tight_layout()

# Show plot
plt.show()


```

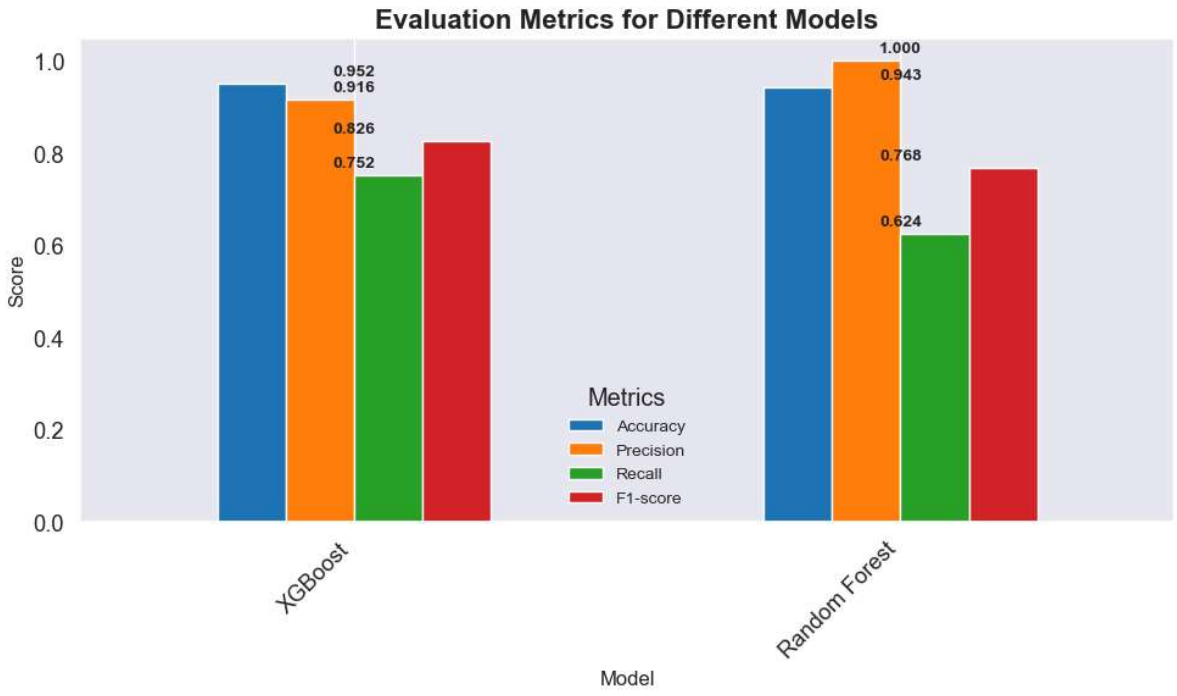


The ROC AUC score of 0.93 for both models demonstrates strong discriminatory power, reducing false positives and false negatives, indicating their ability to accurately capture data patterns and provide well-informed predictions across various threshold values, thereby enhancing their reliability in classification tasks.

Evaluation Metrics

In [162]:

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score,

# Initialize dictionaries to store the evaluation metrics for each model
evaluation_metrics = {'Model': [], 'Accuracy': [], 'Precision': [], 'Recal

# Assuming "models" is a dictionary containing your tuned models (XGBoost,
for model_name, model in [('XGBoost', xgb_grid.best_estimator_), ('Random
    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Store evaluation metrics in the dictionary
    evaluation_metrics['Model'].append(model_name)
    evaluation_metrics['Accuracy'].append(accuracy)
    evaluation_metrics['Precision'].append(precision)
    evaluation_metrics['Recall'].append(recall)
    evaluation_metrics['F1-score'].append(f1)

# Convert the dictionary to a DataFrame
metrics_df = pd.DataFrame(evaluation_metrics)

# Plot bar graphs for each evaluation metric
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']  # Blue, Orange, Gre
metrics_df.set_index('Model').plot(kind='bar', figsize=(10, 6), color=colo
plt.title('Evaluation Metrics for Different Models', fontsize=16, fontweig
plt.ylabel('Score', fontsize=12)
plt.xlabel('Model', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.legend(title='Metrics', fontsize=10)
plt.grid(axis='y')

# Annotate bars with values
for i, model in enumerate(metrics_df['Model']):
    plt.text(i, metrics_df.loc[i, 'Accuracy'] + 0.01, f'{metrics_df.loc[i,
    plt.text(i, metrics_df.loc[i, 'Precision'] + 0.01, f'{metrics_df.loc[i
    plt.text(i, metrics_df.loc[i, 'Recall'] + 0.01, f'{metrics_df.loc[i, '
    plt.text(i, metrics_df.loc[i, 'F1-score'] + 0.01, f'{metrics_df.loc[i,

# Adjust layout
plt.tight_layout()

# Show plot
plt.show()
```



In [163]:
```
1 xgb_model.fit(X_train, y_train)
```

Out[163]:
```
XGBClassifier
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=N
one,
              enable_categorical=False, eval_metric=None, feature_types=N
one,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=N
one,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
```

In [164]:
```
1 rf_model.fit(X_train, y_train)
```

Out[164]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [165]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,

# Define a function to calculate and print evaluation metrics
def print_evaluation_metrics(model_name, y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    print(f"{model_name} Metrics:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")
    print()

# Make predictions using the tuned XGBoost model
y_pred_xgb = xgb_model.predict(X_test)

# Print evaluation metrics for XGBoost
print_evaluation_metrics("XGBoost", y_test, y_pred_xgb)

# Make predictions using the tuned Random Forest model
y_pred_rf = rf_model.predict(X_test)

# Print evaluation metrics for Random Forest
print_evaluation_metrics("Random Forest", y_test, y_pred_rf)

```

```
XGBoost Metrics:
Accuracy: 0.9610
Precision: 0.9630
Recall: 0.7723
F1-score: 0.8571

Random Forest Metrics:
Accuracy: 0.9415
Precision: 1.0000
Recall: 0.6139
F1-score: 0.7607
```

XGBoost vs Random Forest Evaluation

- XGBoost outperformed Random Forest in accuracy, recall, and F1-score.
- XGBoost achieved a higher accuracy (0.9610) than Random Forest (0.9415).
- Random Forest had a perfect precision (1.0000), indicating all positive predictions were correct.
- XGBoost's higher recall (0.7723) indicates its ability to identify more true positives.

Feature selection is crucial in enhancing model performance, reducing overfitting, and improving interpretability. Prioritizing informative features simplifies the model, reduces computational costs, and may improve predictive accuracy.

In [166]:

```
from xgboost import XGBClassifier

# Train XGBoost model
xgb_model.fit(X_train, y_train)

# Access feature importance
feature_importance_xgb = xgb_model.feature_importances_

# Rank features based on importance scores
ranked_features_xgb = sorted(zip(feature_importance_xgb, X.columns), reve

# Select top N features (e.g., top 10)
top_features_xgb = [feature for importance, feature in ranked_features_xg

# Subset data with selected features
X_train_subset_xgb = X_train[top_features_xgb]
X_test_subset_xgb = X_test[top_features_xgb]

# Re-train model with selected features
xgb_model_subset = XGBClassifier()
xgb_model_subset.fit(X_train_subset_xgb, y_train)

# Print selected top features
print("Top 10 Features Selected by XGBoost:")
for rank, feature in enumerate(top_features_xgb, start=1):
    print(f"{rank}. {feature}")

# Evaluate model performance on the test set
y_pred_subset_xgb = xgb_model_subset.predict(X_test_subset_xgb)

# Print evaluation metrics
print("\nXGBoost Model with Selected Features Metrics:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_subset_xgb):.4f}")
print(f"Precision: {precision_score(y_test, y_pred_subset_xgb):.4f}")
print(f"Recall: {recall_score(y_test, y_pred_subset_xgb):.4f}")
print(f"F1-score: {f1_score(y_test, y_pred_subset_xgb):.4f}")

```

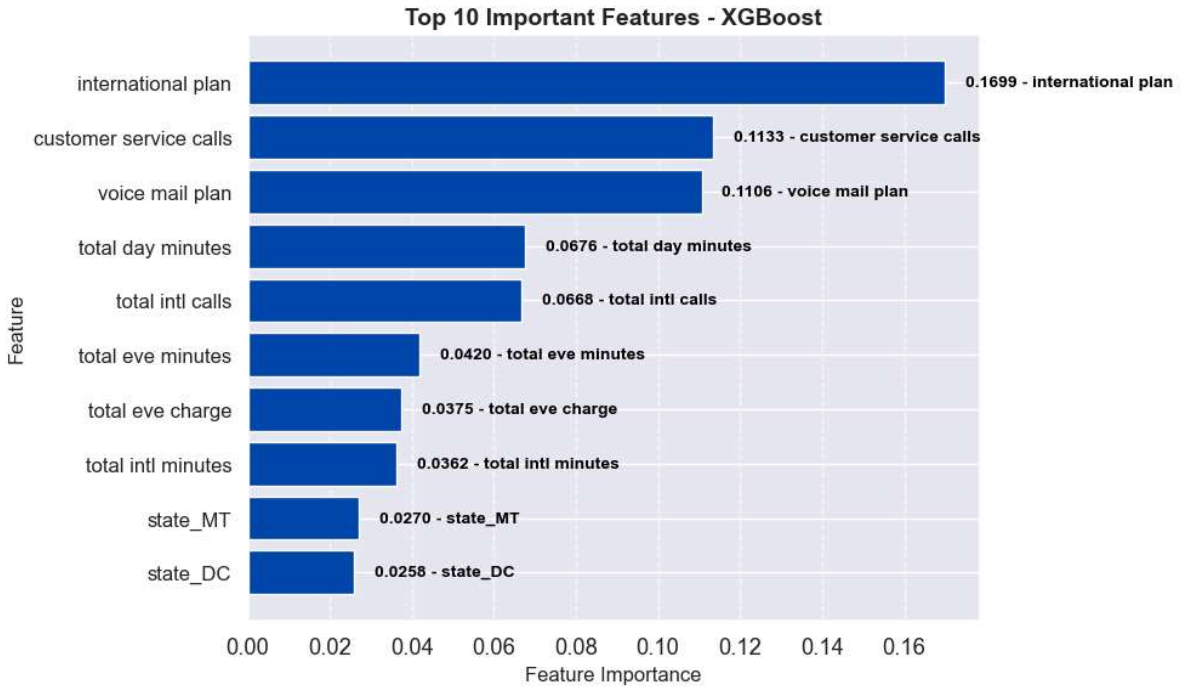
```
Top 10 Features Selected by XGBoost:
1. international plan
2. customer service calls
3. voice mail plan
4. total day minutes
5. total intl calls
6. total eve minutes
7. total eve charge
8. total intl minutes
9. state_MT
10. state_DC

XGBoost Model with Selected Features Metrics:
Accuracy: 0.9535
Precision: 0.9268
Recall: 0.7525
F1-score: 0.8306
```

In [167]:

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,

# Train XGBoost model
xgb_model.fit(X_train, y_train)

# Access feature importance
feature_importance_xgb = xgb_model.feature_importances_

# Rank features based on importance scores
ranked_features_xgb = sorted(zip(feature_importance_xgb, X.columns), rever

# Select top N features (e.g., top 10)
top_features_xgb = [feature for importance, feature in ranked_features_xgb

# Subset data with selected features
X_train_subset_xgb = X_train[top_features_xgb]
X_test_subset_xgb = X_test[top_features_xgb]

# Re-train model with selected features
xgb_model_subset = XGBClassifier()
xgb_model_subset.fit(X_train_subset_xgb, y_train)

# Print selected top features
print("Top 10 Features Selected by XGBoost:")
for rank, (importance, feature) in enumerate(ranked_features_xgb[:10], sta
    print(f"{rank}. {feature}: {importance:.4f}")

# Evaluate model performance on the test set
y_pred_subset_xgb = xgb_model_subset.predict(X_test_subset_xgb)

# Calculate evaluation metrics
accuracy_subset_xgb = accuracy_score(y_test, y_pred_subset_xgb)
precision_subset_xgb = precision_score(y_test, y_pred_subset_xgb)
recall_subset_xgb = recall_score(y_test, y_pred_subset_xgb)
f1_subset_xgb = f1_score(y_test, y_pred_subset_xgb)

# Print evaluation metrics
print("\nXGBoost Model with Selected Features Metrics:")
print(f"Accuracy: {accuracy_subset_xgb:.4f}")
print(f"Precision: {precision_subset_xgb:.4f}")
print(f"Recall: {recall_subset_xgb:.4f}")
print(f"F1-score: {f1_subset_xgb:.4f}")

```

```
Top 10 Features Selected by XGBoost:
1. international plan: 0.1699
2. customer service calls: 0.1133
3. voice mail plan: 0.1106
4. total day minutes: 0.0676
5. total intl calls: 0.0668
6. total eve minutes: 0.0420
7. total eve charge: 0.0375
8. total intl minutes: 0.0362
9. state_MT: 0.0270
10. state_DC: 0.0258

XGBoost Model with Selected Features Metrics:
Accuracy: 0.9535
Precision: 0.9268
Recall: 0.7525
F1-score: 0.8306
```

In [168]:
```
ine a professional color palette
 = '#004AAD'  # Dark blue color
_color = '#6699CC'  # Light blue color

t feature importances
igure(figsize=(10, 6))
= plt.barh(range(len(top_features_xgb)), [importance for importance, _ in ranke
ticks(range(len(top_features_xgb)), [feature for _, feature in ranked_features_
label('Feature Importance', fontsize=12)
label('Feature', fontsize=12)
itle('Top 10 Important Features - XGBoost', fontsize=14, fontweight='bold')

ing annotations
ar, (importance, feature) in zip(bars, ranked_features_xgb[:10]):
ltext(bar.get_width() + 0.005, bar.get_y() + bar.get_height()/2, f'{importanc
    va='center', fontsize=10, fontweight='bold', color='black')

timize plot aesthetics
ca().invert_yaxis()
rid(axis='x', linestyle='--', alpha=0.7)
ight_layout()
how()
```



Conclusion & Recomendations

This project developed machine learning models to predict customer churn for SyriaTel. Using various classification algorithms, such as Logistic Regression, Random Forests and XGBoost, We then identified key predictors of churn and developed accurate predictive models. From the

data we could see that focusing on customer retention strategies, enhancing service offerings, continuous model monitoring and improvement, establishing a customer feedback loop, and investing in data analytics and infrastructure would be the best corse of action.

We Recomend targeted customer retention strategies, especially for high churn risk customers, can help reduce churn rates and improve customer satisfaction. Enhancing service offerings, such as international plans and call durations, can also help reduce dissatisfaction and churn. Continuous monitoring and retraining with updated data are recommended to ensure model effectiveness over time. Including customer feedback in model training and decision-making processes can lead to more accurate predictions and better outcomes.

By implementing these recommendations and leveraging the insights gained from the predictive

In [ ]:
```
1
```

In [ ]:
```
1
```