ictive-analysis-for-customer-churn

March 10, 2024

0.1 PREDICTIVE ANALYSIS FOR CUSTOMER CHURN IN SYRIATEL TELECOM DATA

[9]:
```
from PIL import Image
import IPython.display as display


image_path = r"C:
↪\Users\ADMIN\Desktop\ProjectPhase3\ProjectPhase3\Images\Perspectivas-Globais_-O-Assistente-
↪jpg"

img = Image.open(image_path)
display.display(img)
```

0.2 A)BACKGROUND OF THE INDUSTRY

In the telecommunications industry, data plays a pivotal role in understanding customer behavior, optimizing network performance, and driving business strategies. With the proliferation of mobile devices, IoT (Internet of Things) devices, and high-speed internet connections, the volume and variety of data generated by telecommunications networks have surged exponentially.

Telecommunications data encompasses various types, including call records, text messages, internet usage logs, network performance metrics, customer demographics, and geographic location data. These datasets are incredibly vast and complex, often spanning millions of records generated in real-time.

Analyzing telecommunications data provides valuable insights into customer preferences, usage patterns, and satisfaction levels. By leveraging advanced analytics and machine learning techniques, telecom companies can identify opportunities for personalized marketing, targeted promotions, and improved customer experiences.

0.3 B)INTRODUCTION

In today's hyperconnected world, telecommunications companies are at the forefront of innovation, facilitating seamless communication and connectivity for billions of people worldwide. Amidst this ever-evolving landscape, one of the most pressing challenges faced by telecom operators is the phenomenon of customer churn.

Customer churn, the rate at which subscribers discontinue their services, not only leads to immediate revenue loss but also signifies underlying issues in service quality, customer satisfaction, and market competitiveness. To address this critical issue, SyriaTel, a prominent player in the telecommunications sector, is embarking on a transformative journey leveraging advanced analytics and machine learning. By analyzing vast troves of historical customer data, SyriaTel aims to unveil intricate patterns and insights that can accurately predict which subscribers are at risk of churning.

Armed with this foresight, SyriaTel can tailor retention strategies, enhancing customer satisfaction, and driving long-term business growth. This project endeavors to explore the potential of predictive analytics in mitigating customer churn, ultimately positioning SyriaTel as an industry leader committed to delivering exceptional service and fostering enduring customer relationships.

0.4 C)PROBLEM STATEMENT

what is the prevailing circumstance? In the telecommunications industry, particularly within the domain of SyriaTel, high customer churn rates persist as a significant concern. Despite considerable investments in marketing and customer retention strategies, the company continues to experience a notable loss of subscribers, impacting revenue streams and market competitiveness. Traditional methods for predicting churn have yielded suboptimal results, leading to ineffective allocation of resources and missed opportunities for retaining valuable customers.

what problem are we trying to solve? The primary challenge this project aims to address is the inefficient management of customer churn within SyriaTel's subscriber base. High churn rates not only signify dissatisfaction among customers but also result in substantial revenue loss and hinder sustainable business growth. The existing approach to churn prediction lacks accuracy and fails to provide actionable insights necessary for implementing targeted retention strategies.

Consequently, SyriaTel faces the imperative need to develop a more robust and data-driven solution to mitigate churn effectively.

How the project aims to solve the problem? This project endeavors to leverage advanced analytics and machine learning techniques to develop a predictive model capable of accurately identifying customers at risk of churn. By analyzing historical customer data encompassing usage patterns, demographics, and service interactions, the project seeks to unveil hidden patterns and indicators of potential churn. Through the deployment of sophisticated algorithms and predictive modeling, SyriaTel aims to proactively identify at-risk customers and implement personalized retention initiatives. By doing so, the project aims to minimize churn rates, optimize revenue retention, and enhance overall customer satisfaction, thereby fortifying SyriaTel's position as a leader in the telecommunications industry.

0.5 D)OBJECTIVES

Main Objective:

The primary objective of this project is to develop a robust predictive model to accurately forecast customer churn within SyriaTel's subscriber base, leveraging advanced analytics and machine learning techniques.

Specific Objectives:

1. Analyze Historical Data: Conduct in-depth analysis of SyriaTel's historical customer data, encompassing usage patterns, demographic information, service interactions, and churn records, to identify relevant features and trends indicative of potential churn.
2. Develop Predictive Model: Utilize advanced analytics and machine learning algorithms, such as logistic regression, decision trees, and ensemble methods, to build a predictive model capable of forecasting customer churn with high accuracy. This involves data preprocessing, feature selection, model training, validation, and optimization.
3. Implement Retention Strategies: Integrate the developed predictive model into SyriaTel's existing operational framework to enable real-time identification of at-risk customers. Design and implement personalized retention strategies based on the model's predictions, targeting specific customer segments with tailored offers, incentives, and proactive communication to mitigate churn effectively.

0.6 E)NOTEBOOK STRUCTURE

1.Business Understanding

2.Data Understandiing

3.Data Cleaning

4.Exploratory Data Analysis

5.Data Preparation

6.Modelling

7.Evaluation

8.Conclusion ,Recommendations and Nextsteps.

0.7 1.BUSINESS UNDERSTANDING

SyriaTel, like many telecommunications companies, faces the pressing challenge of high customer churn rates, which pose significant financial losses and hinder sustainable growth. Conventional churn prediction methods have proven ineffective, leading to wasted resources and suboptimal retention efforts.

To tackle this issue, SyriaTel can harness the power of advanced analytics and machine learning. By digging into historical customer data, SyriaTel can uncover certain patterns in behavior, preferences, and usage that indicate potential churn risks. Armed with this foresight, SyriaTel can tailor retention strategies to individual customers, significantly reducing churn rates and preserving revenue streams.

By embracing proactive churn management, SyriaTel not only mitigates immediate revenue concerns but also cultivates lasting customer loyalty and strengthens its competitive position in the market. Investing in advanced analytics and machine learning capabilities not only optimizes customer retention but also sets SyriaTel apart as an industry leader committed to customer satisfaction and long-term growth.

In essence, by leveraging advanced analytics and machine learning, SyriaTel can develop a robust churn prediction model that not only addresses immediate revenue loss but also fosters enduring customer relationships and drives sustainable business expansion.

0.7.1 Stakeholders

-The key stakeholders and their interests are:-

SyriaTel Management: Interest: Interested in reducing customer churn rates to enhance revenue streams and improve overall business performance.

Customer Service Department: Interest: Interested in reducing the volume of customer churn-related inquiries and complaints.

Finance Department: Interest: Interested in understanding the financial implications of customer churn and retention efforts.

Marketing Department: Interest: Interested in developing targeted campaigns and strategies to retain existing customers and attract new ones, ultimately boosting revenue and market share.

0.7.2 Metric for sucess

The project will be successful if the model can correctly spot most customers who are likely to leave (High Recall), avoid wrongly flagging too many customers who won't leave (Low False Positive Rate), and perform well when dealing with new data (Reaching 80% accuracy).

0.8 2.DATA UNDERSTANDING

Data Source: The dataset used for this project was sourced from Kaggle (Churn in Telecom's dataset) and consists of 3333 rows and 21 columns. It provides comprehensive information on customer attributes and behaviors within the telecommunications domain, enabling analysis and prediction of churn patterns.

Data Size: The dataset comprises 3333 instances, each represented by 21 features. Notably, all features, except for ‘Phone number’ and ‘State’, contain numerical values, while the remaining features are categorical or binary. This structured dataset provides a sizable sample for training and testing predictive models, ensuring robustness and reliability in churn prediction. This is a binary classification problem where the goal is to predict the likelihood of a customer churning and the **churn column** will be represented by **1 - True** and **0 - False**

Dataset Columns	about
State	Represents the states in the USA
Account length	represents the length of time (in seconds or minutes) that a customer’s account has been active.
Area code	Geographic area code of a customer’s telephone number.
Phone number	represents the telephone number of a customer.
International plan	represents whether a customer has subscribed to an international call plan or not. It can have either “Yes” or “No” values.
voice mail plan	represents whether a customer has subscribed to a voice mail plan or not. It can have either “Yes” or “No” values.
Number vmail messages	represents the number of voice mail messages left by a customer.
Total day minutes	represents the total amount of time (in minutes) that a customer has spent on daytime calls.
Total day calls	represents the total number of calls that a customer has made during the day.
Total day charge	represents the total charge for daytime calls made by a customer.
total eve minutes	represents the total amount of time (in minutes) that a customer has spent on evening calls.
total eve calls	represents the total number of calls that a customer has made in the evening.
total eve charge	represents the total charge for evening calls made by a customer.
total night minutes	represents the total amount of time (in minutes) that a customer has spent on night calls.
total night calls	represents the total number of calls that a customer has made at night.
total night charge	represents the total charge for night calls made by a customer.

Dataset Columns	about
total intl minutes	represents the total amount of time (in minutes) that a customer has spent on international calls.
total intl calls	represents the total number of international calls made by a customer.
total intl charge	represents the total charge for international calls made by a customer.
customer service calls	represents the number of customer service calls made by a customer.
churn	represents whether a customer has cancelled their service or not. It can have either "True" or "False" values.

Relevance to the Project: The dataset encompasses various attributes relevant to understanding customer behavior and predicting churn in the telecommunications sector. Key features include customer demographics (e.g., account length, area code), service subscriptions (e.g., international plan, voice mail plan), call usage metrics (e.g., total day minutes, total night calls), and churn status. The 'Churn' column, serving as the target variable, distinguishes between customers who have canceled their service ('True') and those who have not ('False'). This rich dataset forms the foundation for developing a predictive model to identify churn risks accurately and implement targeted retention strategies, aligning with the project's objective of mitigating customer churn effectively.

[1]:
```
#Importing relevant libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import math
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")

import xgboost as xgb
import joblib
import pickle

#sklearn preprocessing
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler, LabelEncoder␣
 ↪,OneHotEncoder,StandardScaler
from sklearn.pipeline import Pipeline
```

```python
from sklearn.pipeline import make_pipeline

from sklearn.feature_selection import RFECV

# sklearn classification models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier

from imblearn.over_sampling import SMOTE

#sklearn evaluation metrics and validation
from sklearn.model_selection import train_test_split, KFold,StratifiedKFold,␣
↪cross_val_score, GridSearchCV
from sklearn.metrics import accuracy_score,␣
↪precision_score,recall_score,f1_score,roc_curve, auc
from sklearn.metrics import roc_auc_score,confusion_matrix,␣
↪classification_report

from sklearn.compose import ColumnTransformer
```

[2]:
```python
#loading the dataset
Data= 'ProjectPhase3\Churn in telecoms dataset.csv'
# Load the CSV file into a Pandas DataFrame
df = pd.read_csv(Data)
```

0.8.1 a)Determining the number of records

[3]:
```python
num_records = df.shape
print("Number of records:", num_records)
```

```
Number of records: (3333, 21)
```

0.8.2 b)Preview top and bottom of our dataset

[4]:
```python
# Preview the top of the dataset
top_rows = df.head()
top_rows
```

[4]:
```
  state  account length  area code phone number international plan  \
0    KS             128        415     382-4657                 no
1    OH             107        415     371-7191                 no
2    NJ             137        415     358-1921                 no
```

```
3    OH              84        408     375-9999                 yes
4    OK              75        415     330-6626                 yes

  voice mail plan  number vmail messages  total day minutes  total day calls  \
0             yes                     25              265.1              110
1             yes                     26              161.6              123
2              no                      0              243.4              114
3              no                      0              299.4               71
4              no                      0              166.7              113

   total day charge  …  total eve calls  total eve charge  \
0             45.07  …               99             16.78
1             27.47  …              103             16.62
2             41.38  …              110             10.30
3             50.90  …               88              5.26
4             28.34  …              122             12.61

   total night minutes  total night calls  total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   total intl minutes  total intl calls  total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   customer service calls  churn
0                       1  False
1                       1  False
2                       0  False
3                       2  False
4                       3  False

[5 rows x 21 columns]
```

[5]:
```
# Preview the bottom of the dataset
bottom_rows = df.tail()
bottom_rows
```

[5]:
```
      state  account length  area code phone number international plan  \
3328    AZ             192        415     414-4276                 no
3329    WV              68        415     370-3271                 no
```

```
3330    RI              28        510    328-8230                   no
3331    CT             184        510    364-6381                  yes
3332    TN              74        415    400-4344                   no

     voice mail plan  number vmail messages  total day minutes  \
3328             yes                     36              156.2
3329              no                      0              231.1
3330              no                      0              180.8
3331              no                      0              213.8
3332             yes                     25              234.4

      total day calls  total day charge  …  total eve calls  \
3328               77             26.55  …              126
3329               57             39.29  …               55
3330              109             30.74  …               58
3331              105             36.35  …               84
3332              113             39.85  …               82

      total eve charge  total night minutes  total night calls  \
3328             18.32                279.1                 83
3329             13.04                191.3                123
3330             24.55                191.9                 91
3331             13.57                139.2                137
3332             22.60                241.4                 77

      total night charge  total intl minutes  total intl calls  \
3328               12.56                 9.9                 6
3329                8.61                 9.6                 4
3330                8.64                14.1                 6
3331                6.26                 5.0                10
3332               10.86                13.7                 4

      total intl charge  customer service calls  churn
3328               2.67                       2  False
3329               2.59                       3  False
3330               3.81                       2  False
3331               1.35                       2  False
3332               3.70                       0  False

[5 rows x 21 columns]
```

0.8.3 c)Checking data types in various columns

- This involves checking whether the columns have appropriate data types

[6]:
```
data_types = df.dtypes
print(f"Data types of each column:\n{data_types}")
```

```
Data types of each column:
state                      object
account length              int64
area code                   int64
phone number               object
international plan         object
voice mail plan            object
number vmail messages       int64
total day minutes         float64
total day calls             int64
total day charge          float64
total eve minutes         float64
total eve calls             int64
total eve charge          float64
total night minutes       float64
total night calls           int64
total night charge        float64
total intl minutes        float64
total intl calls            int64
total intl charge         float64
customer service calls      int64
churn                        bool
dtype: object
```

0.8.4 d)Descriptive statistics

[7]:
```
df.describe()
```

[7]:
```
       account length    area code  number vmail messages  total day minutes  \
count     3333.000000  3333.000000            3333.000000        3333.000000
mean       101.064806   437.182418               8.099010         179.775098
std         39.822106    42.371290              13.688365          54.467389
min          1.000000   408.000000               0.000000           0.000000
25%         74.000000   408.000000               0.000000         143.700000
50%        101.000000   415.000000               0.000000         179.400000
75%        127.000000   510.000000              20.000000         216.400000
max        243.000000   510.000000              51.000000         350.800000

       total day calls  total day charge  total eve minutes  total eve calls  \
count      3333.000000       3333.000000        3333.000000      3333.000000
mean        100.435644         30.562307         200.980348       100.114311
std          20.069084          9.259435          50.713844        19.922625
min           0.000000          0.000000           0.000000         0.000000
25%          87.000000         24.430000         166.600000        87.000000
50%         101.000000         30.500000         201.400000       100.000000
75%         114.000000         36.790000         235.300000       114.000000
max         165.000000         59.640000         363.700000       170.000000
```

```
       total eve charge  total night minutes  total night calls  \
count       3333.000000          3333.000000        3333.000000
mean          17.083540           200.872037         100.107711
std            4.310668            50.573847          19.568609
min            0.000000            23.200000          33.000000
25%           14.160000           167.000000          87.000000
50%           17.120000           201.200000         100.000000
75%           20.000000           235.300000         113.000000
max           30.910000           395.000000         175.000000

       total night charge  total intl minutes  total intl calls  \
count         3333.000000         3333.000000       3333.000000
mean             9.039325           10.237294          4.479448
std              2.275873            2.791840          2.461214
min              1.040000            0.000000          0.000000
25%              7.520000            8.500000          3.000000
50%              9.050000           10.300000          4.000000
75%             10.590000           12.100000          6.000000
max             17.770000           20.000000         20.000000

       total intl charge  customer service calls
count        3333.000000             3333.000000
mean            2.764581                1.562856
std             0.753773                1.315491
min             0.000000                0.000000
25%             2.300000                1.000000
50%             2.780000                1.000000
75%             3.270000                2.000000
max             5.400000                9.000000
```

0.8.5 e)Summary of our dataframe

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account length          3333 non-null   int64
 2   area code               3333 non-null   int64
 3   phone number            3333 non-null   object
 4   international plan      3333 non-null   object
 5   voice mail plan         3333 non-null   object
 6   number vmail messages   3333 non-null   int64
```

```
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64
 20  churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

0.9 3.DATA CLEANING

Data cleaning involves addressing issues related to the quality of the dataset. It aims to ensure that the data is accurate, consistent, and free from errors. Here are some data cleaning methods engaged in:

Checking Missing Values:

Identify and address any missing values in the dataset. Options include imputation, removal of rows or columns with missing values, or treating missing values as a separate category.

Checking for Duplicates:

Identify and remove any duplicate records in the dataset to avoid redundancy and potential bias in the analysis.

Checking for placeholders:

Investigate and rectify any placeholders in the data that may affect the accuracy of the model.

Ckecking for outliers

Decide on an appropriate approach for handling outliers, such as removing them, transforming them, or treating them separately in the analysis.

Checking and converting data types

Ensure each column has the appropriate data type for analysis and modeling.Convert data types as needed to ensure consistency and accuracy in the dataset.

a)Checking Missing Values

The data does not contain missing values which need to be addressed.

[9]:
```
# Check for missing values
missing_values = df.isnull().sum()
```

```python
# Print the count of missing values for each column
print("Missing values count for each column:")
print(missing_values)

# Check if there are any missing values in the DataFrame
if missing_values.sum() == 0:
    print("No missing values found.")
else:
    print("There are missing values in the dataset.")
```

```
Missing values count for each column:
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
No missing values found.
```

b)Checking for duplicates

The phone number is a unique identifier in the dataset has no missing values, there are no duplicates in the rows in the datasets.

[10]:
```python
# Checking duplicated rows
df.duplicated().sum()
```

[10]: 0

[11]:
```python
# Checking for duplicate in phone number
duplicates_numbers = df.duplicated(subset ='phone number')
duplicates_numbers.unique()
```

[11]: array([False])

c)Checking placeholders

No place-holders are in the state, area_code, international_plan, voice_mail_plan and churn columns.

[12]:
```
# Checking for place holders
columns = ['state','area code','international plan', 'voice mail plan', 'churn']
unique_values = {}
for col in columns:
    unique_values[col] = df[col].unique()
unique_values
```

[12]:
```
{'state': array(['KS', 'OH', 'NJ', 'OK', 'AL', 'MA', 'MO', 'LA', 'WV', 'IN',
'RI',
        'IA', 'MT', 'NY', 'ID', 'VT', 'VA', 'TX', 'FL', 'CO', 'AZ', 'SC',
        'NE', 'WY', 'HI', 'IL', 'NH', 'GA', 'AK', 'MD', 'AR', 'WI', 'OR',
        'MI', 'DE', 'UT', 'CA', 'MN', 'SD', 'NC', 'WA', 'NM', 'NV', 'DC',
        'KY', 'ME', 'MS', 'TN', 'PA', 'CT', 'ND'], dtype=object),
 'area code': array([415, 408, 510], dtype=int64),
 'international plan': array(['no', 'yes'], dtype=object),
 'voice mail plan': array(['yes', 'no'], dtype=object),
 'churn': array([False,  True])}
```

d)Checking for outliers

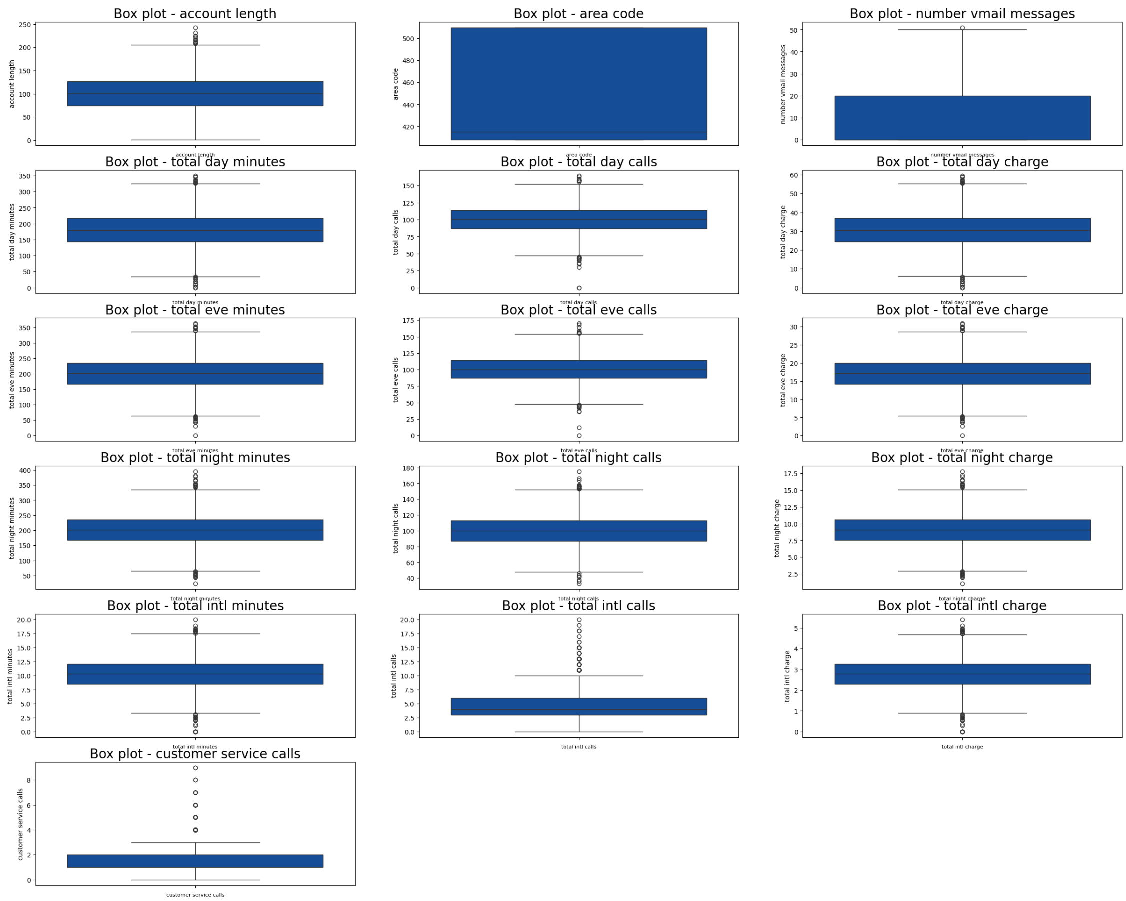
- From the box plots, there are only a few values that stand out from the rest in each column. However, there's nothing really unusual or extreme that would need to be taken out of the dataset.
- Keep these values because getting rid of them might mean losing important information.These values don't seem extreme enough to make a big difference in how well the model works.

[13]:
```
# Creating a list of columns with numeric values
numeric_cols = df.select_dtypes('number').columns

# Calculate the number of rows and columns for subplots
num_rows = (len(numeric_cols) - 1) // 3 + 1
num_cols = min(len(numeric_cols), 3)

# Create the subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(10*num_cols, 4*num_rows))

# Generate box plots for each numeric column
for i, column in enumerate(numeric_cols):
    row = i // num_cols
    col = i % num_cols
    sns.boxplot(data=df[column], ax=axes[row, col], color='#004aad')
```

```
    axes[row, col].set_title(f'Box plot - {column}', fontsize=20)
    axes[row, col].set_xlabel(column, fontsize=8)

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

plt.tight_layout
```

[13]:

```
<function matplotlib.pyplot.tight_layout(*, pad: 'float' = 1.08, h_pad: 'float |
None' = None, w_pad: 'float | None' = None, rect: 'tuple[float, float, float,
float] | None' = None) -> 'None'>
```



e)Checking and converting data types

[14]:

```
# Checking data types of categorical variables
columns = ['state', 'area code', 'international plan', 'voice mail plan']
column_data_types = df[columns].dtypes
```

```
print(column_data_types)
```

```
state                   object
area code                int64
international plan      object
voice mail plan         object
dtype: object
```

[15]:
```
# Convert "State" column to categorical data type
df["area code"] = df["area code"].astype("str")
print(df["area code"].dtype)
```

```
object
```

[16]:
```
# Convert churn, international plan and  voice mail plan column from boolean to␣
↪integer
df["churn"] = df["churn"].astype(int)
print(df["churn"].dtype)
```

```
int32
```

0.10 f)Dropping unnecessary columns

Dropping unnecessary columns such as “phone number” that are not relevant for the analysis and modeling process.

[17]:
```
# Drop the 'phone number' column
df.drop(columns=['phone number'], inplace=True)
```

0.11 4.EXPLORATORY DATA ANALYSIS

This step involves analyzing and summarizing the data to comprehend its fundamental characteristics, reveal patterns, and detect potential relationships and insights. It encompasses univariate, bivariate, and multivariate analysis techniques to gain a comprehensive understanding of the dataset.

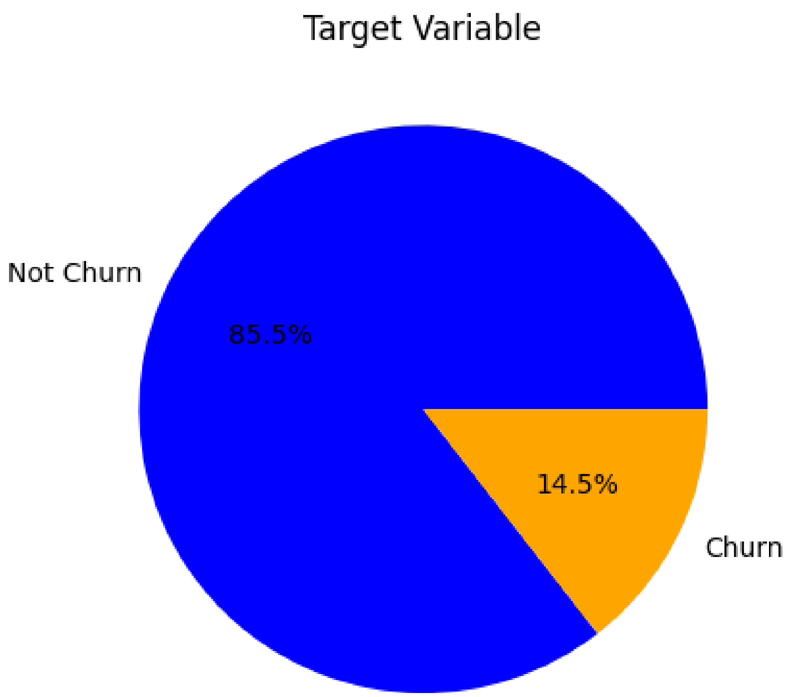
0.11.1 a)Univariate Analysis

i)Categorical Columns

- The dataset indicates the target variable “churn” i.e whether a customer has churned or not.
- Approximately 14.5% of the data corresponds to customers who have churned, while the remaining 85.5% represents customers who have not churned.

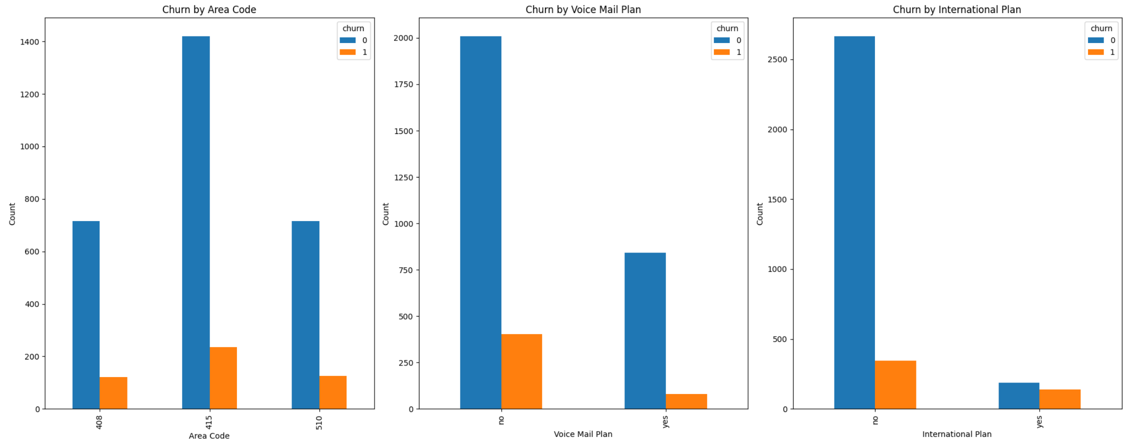
[18]:
```
churn_counts = df['churn'].value_counts()

plt.pie(x=churn_counts, labels=['Not Churn', 'Churn'], autopct='%1.1f%%',␣
↪colors=['blue', 'orange'], textprops={'color': 'black'})
plt.title('Target Variable')
```

```
plt.show()
```




```
[19]: # Set up the figure and axes for subplots
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 8))

# Group by "area code" and "churn", then unstack and plot
df.groupby(["area code", "churn"]).size().unstack().plot(kind='bar',␣
 ↪stacked=False, ax=axs[0])
axs[0].set_title('Churn by Area Code')
axs[0].set_xlabel('Area Code')
axs[0].set_ylabel('Count')

# Group by "voice mail plan" and "churn", then unstack and plot
df.groupby(["voice mail plan", "churn"]).size().unstack().plot(kind='bar',␣
 ↪stacked=False, ax=axs[1])
axs[1].set_title('Churn by Voice Mail Plan')
axs[1].set_xlabel('Voice Mail Plan')
axs[1].set_ylabel('Count')

# Group by "international plan" and "churn", then unstack and plot
df.groupby(["international plan", "churn"]).size().unstack().plot(kind='bar',␣
 ↪stacked=False, ax=axs[2])
```

```python
axs[2].set_title('Churn by International Plan')
axs[2].set_xlabel('International Plan')
axs[2].set_ylabel('Count')

# Adjust the layout and spacing
plt.tight_layout()
plt.show()
```

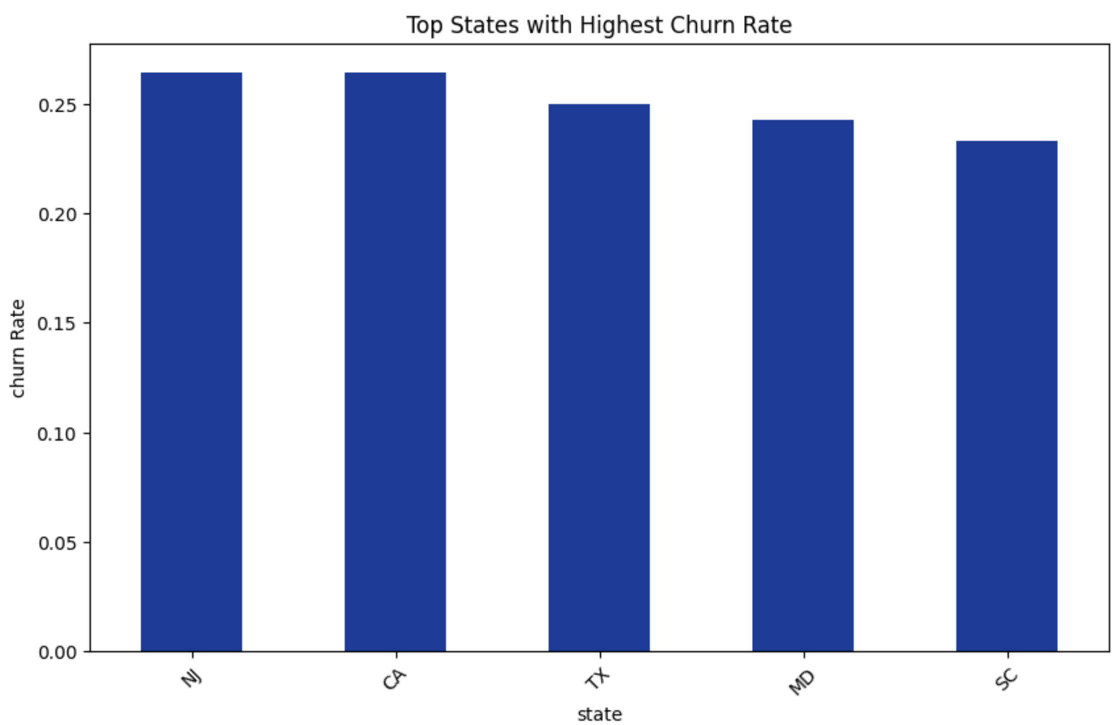


1 Findings

1. **Area Code Analysis:**
    - Churn rates vary significantly across different area codes.
    - Area code 415 exhibits the highest churn rate, while area code 408 demonstrates the lowest churn rate.
    - Despite area codes 510 and 408 having fewer instances of churn, it's important to consider the relative customer base size within each area code.
2. **International Plan Analysis:**
    - SyrialTel offers an internal plan for international calls with a customer base of fewer than 500 individuals.
    - The churn rate among customers with an international plan is nearly equal to the number of customers enrolled, suggesting a considerable risk of churn among this group.
3. **Voice Mail Plan Analysis:**
    - SyrialTel offers an optional voice mail plan to its customers.
    - A significant portion of customers have not enrolled in the voice mail plan.
    - Customers who have opted for the voice mail plan exhibit a lower likelihood of churn compared to those without the plan.

1.0.1 Top 5 States with the highest churn rate

[20]:
```
# Calculate churn rate for each state
state_churn_rate = df.groupby('state')['churn'].mean().
    ↪sort_values(ascending=False)

# Get the top states with the highest churn rate
top_states_churn = state_churn_rate.head(5)  # Change 5 to the desired number␣
    ↪of states

# Plot the top states with the highest churn rate
plt.figure(figsize=(10, 6))
top_states_churn.plot(kind='bar', color='#1c3a96')  # Incorporate the color␣
    ↪#1c3a96
plt.title('Top States with Highest Churn Rate')
plt.xlabel('state')
plt.ylabel('churn Rate')
plt.xticks(rotation=45)
plt.show()
```



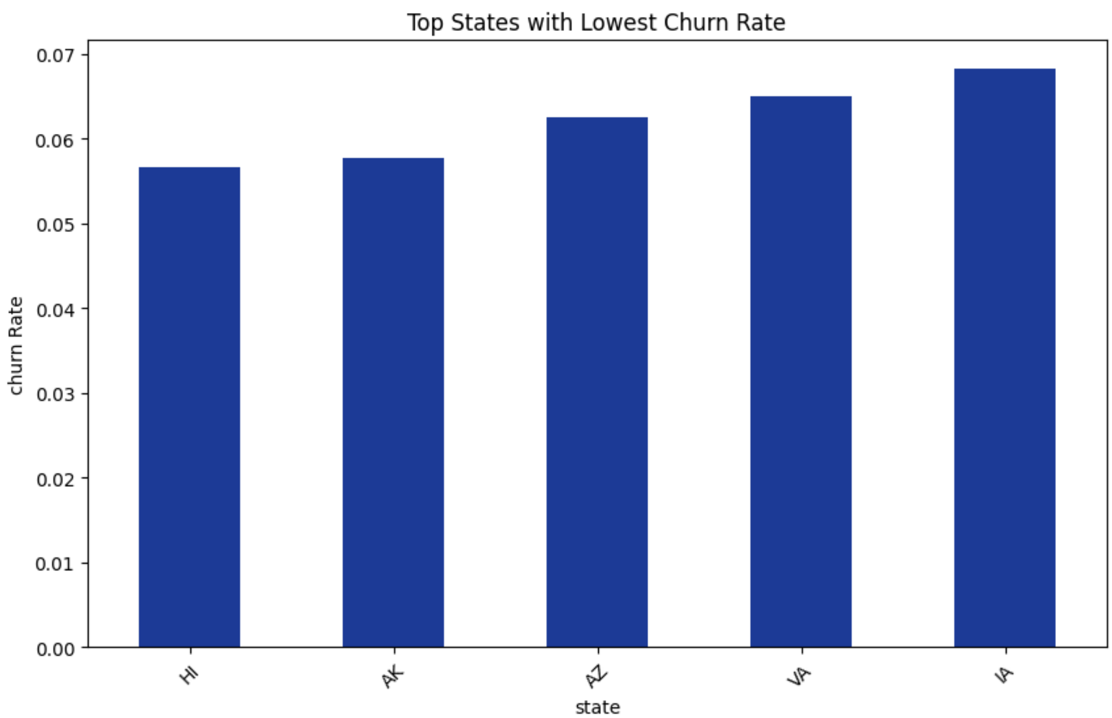
The top 5 states with the highest churn rate are: The states in the USA with the initials NJ, CA, TX, MD, and SC corresponding to :

- NJ: New Jersey
- CA: California

- TX: Texas
- MD: Maryland
- SC: South Carolina

1.0.2 Top 5 States with the lowest churn rate

[21]:
```
# Calculate churn rate for each state
state_churn_rate = df.groupby('state')['churn'].mean().sort_values()

# Get the top states with the lowest churn rate
bottom_states_churn = state_churn_rate.head(5)  # Change 5 to the desired␣
 ↪number of states

# Plot the top states with the lowest churn rate
plt.figure(figsize=(10, 6))
bottom_states_churn.plot(kind='bar', color='#1c3a96')  # Incorporate the color␣
 ↪#1c3a96
plt.title('Top States with Lowest Churn Rate')
plt.xlabel('state')
plt.ylabel('churn Rate')
plt.xticks(rotation=45)
plt.show()
```



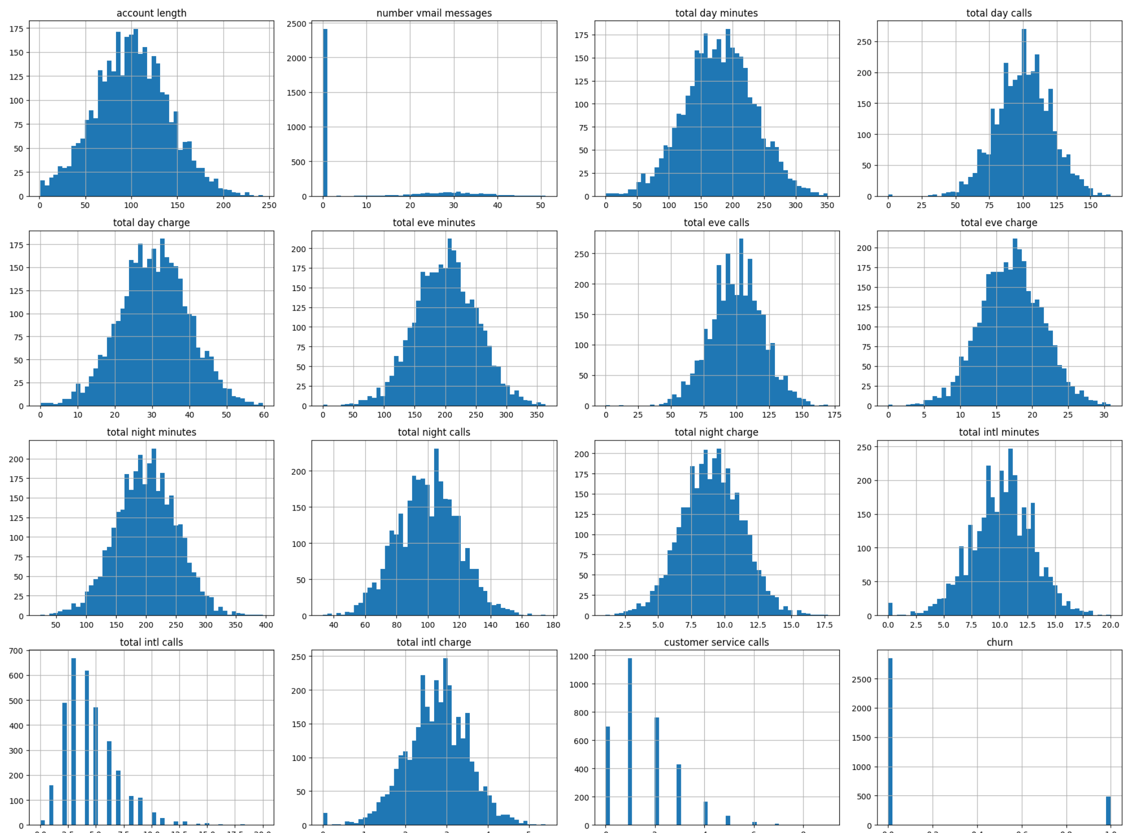
The top 5 states with the lowest churn rate are:The states in the USA with the initials HI, AK, AZ, VA, and LA corresponding to:

- HI: Hawaii
- AK: Alaska
- AZ: Arizona
- VA: Virginia
- LA: Louisiana

ii)Numerical Columns

Checking their distributions

[22]:
```
# Histograms for all numerical columns in the dataset
df.hist(bins=50, figsize=(20,15))
plt.tight_layout()
plt.show()
```

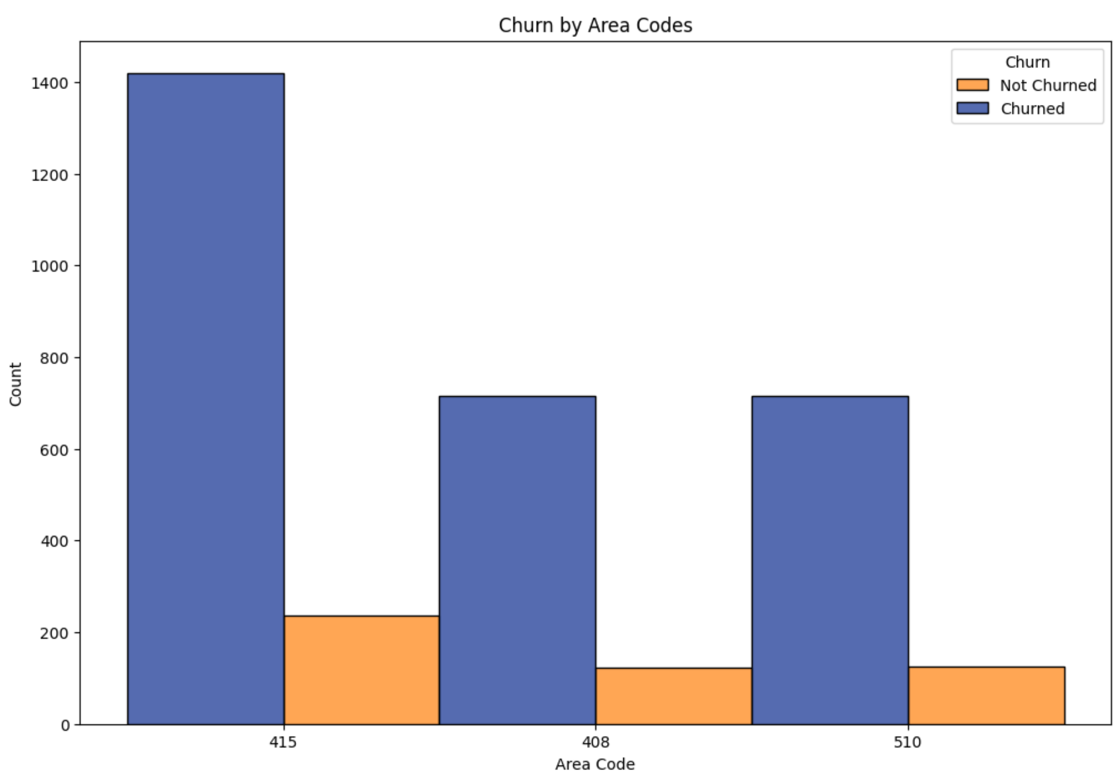


1.0.3 b)Bivariate Analysis

Area code with the highest churn rate

[23]:
```
# Define the colors
colors = ['#1c3a96', '#ff881b']
```

```python
# Set the color palette
sns.set_palette(sns.color_palette(colors))

# Plot churn by area codes
plt.figure(figsize=(12, 8))
sns.histplot(data=df, x='area code', hue='churn', multiple='dodge',␣
↪palette=colors)

# Add a legend with custom labels
plt.legend(title='Churn', labels=['Not Churned', 'Churned'])

# Adjust labels
plt.xlabel('Area Code')
plt.ylabel('Count')

# Title
plt.title('Churn by Area Codes')

# Show plot
plt.show()

print('We have 3 area codes: 408, 415, 510 represented as 0, 1, 2 respectively')
```

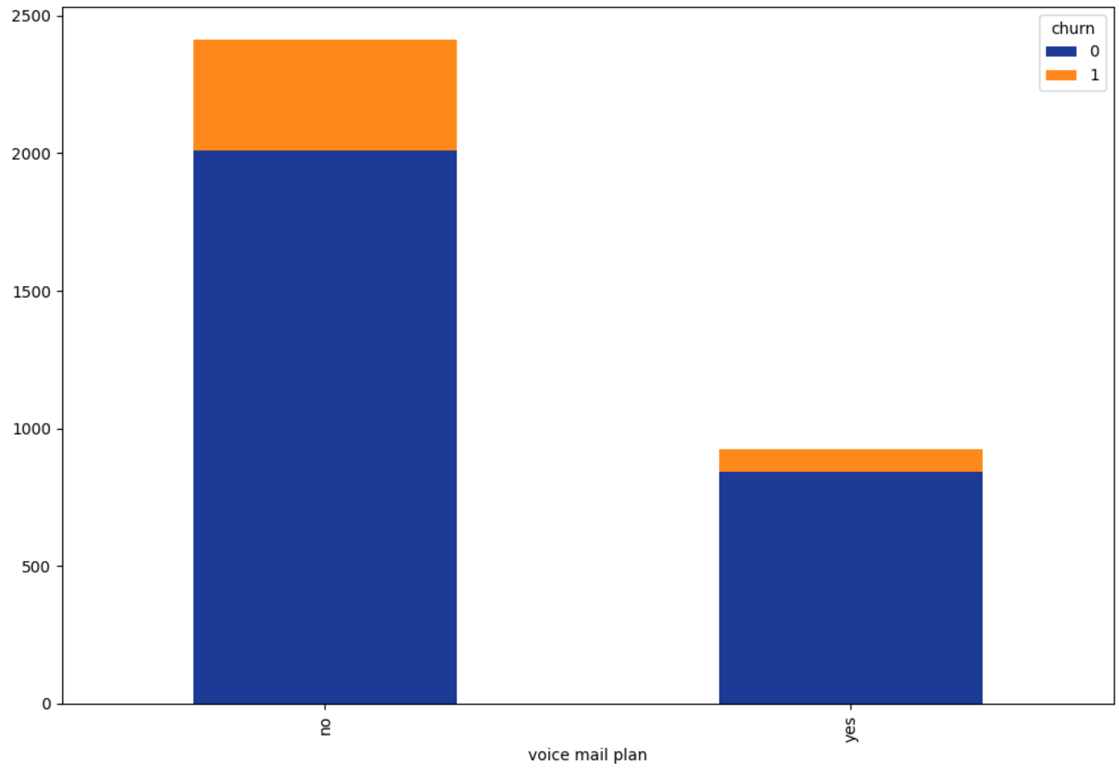


We have 3 area codes: 408, 415, 510 represented as 0, 1, 2 respectively

[24]:
```python
# Function to take different plans
def plot_churn_vs_plan(data, plan_column):
    # Plotting the churn vs plan
    data.groupby([plan_column, 'churn']).size().unstack().plot(
        kind='bar', stacked=True, figsize=(12,8))
    plt.show()

    # Calculating the percentage of customers subscribed to the plan
    total_customers = len(data)
    total_subscribed = sum(data[plan_column] == 'yes')
    percentage_subscribed = (total_subscribed / total_customers) * 100
    print('Percentage of customers subscribed to the {} : {:.2f}%'.
 ↪format(plan_column, percentage_subscribed))

    # Calculating the percentage of churned customers among those subscribed to␣
 ↪the plan
    churned_with_plan = sum((data[plan_column] == 'yes') & (data['churn'] ==␣
 ↪True))
    percentage_churned_with_plan = (churned_with_plan / total_subscribed) * 100
    print('Percentage of subscribed customers who churned with {} : {:.2f}%'.
 ↪format(plan_column, percentage_churned_with_plan))
```

Are customers subscribed to a voice mail plan likely to churn?

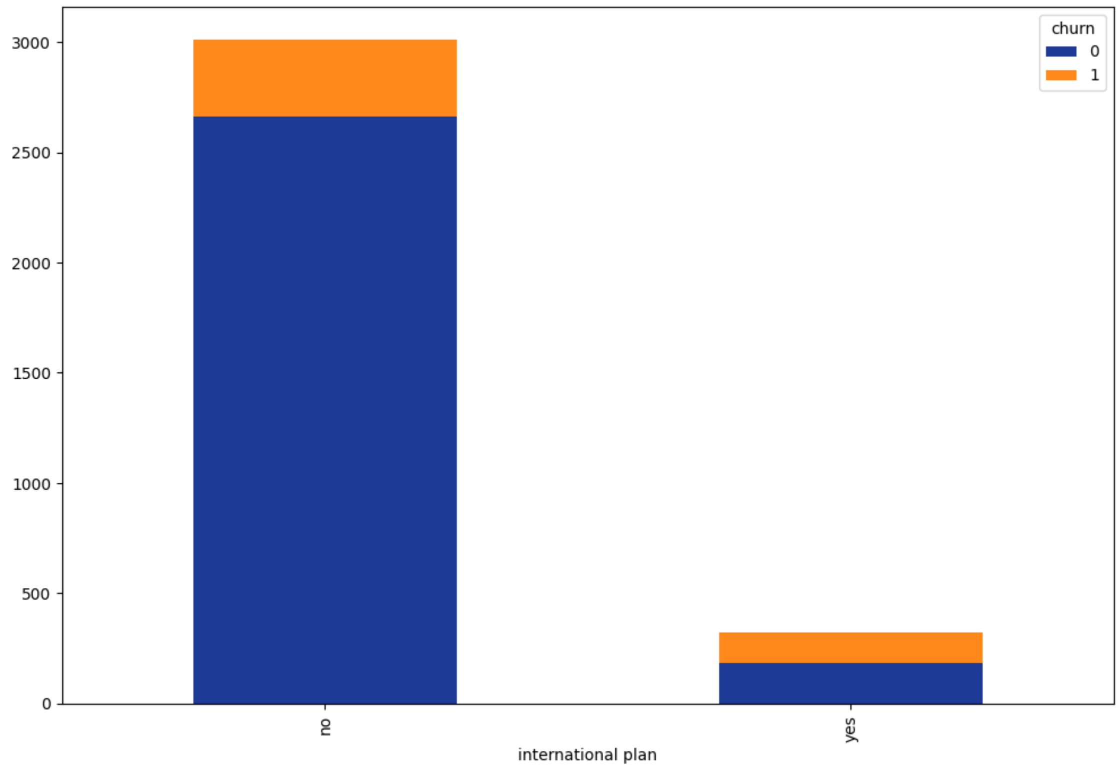
[25]:
```python
# voice mail plan
plot_churn_vs_plan(df,'voice mail plan')
```




```
Percentage of customers subscribed to the voice mail plan : 27.66%
Percentage of subscribed customers who churned with voice mail plan : 8.68%
```

Are customers subscribed to a International plan likely to churn?

[26]:
```
plot_churn_vs_plan(df,'international plan')
```




```
Percentage of customers subscribed to the international plan : 9.69%
Percentage of subscribed customers who churned with international plan : 42.41%
```

1.0.4 c)Multivariate Analysis

The possibility of linear dependency (multicollinearity) between the features can pose a challenge in the interpretation of the created model and affect the accuracy of the estimated coefficients.

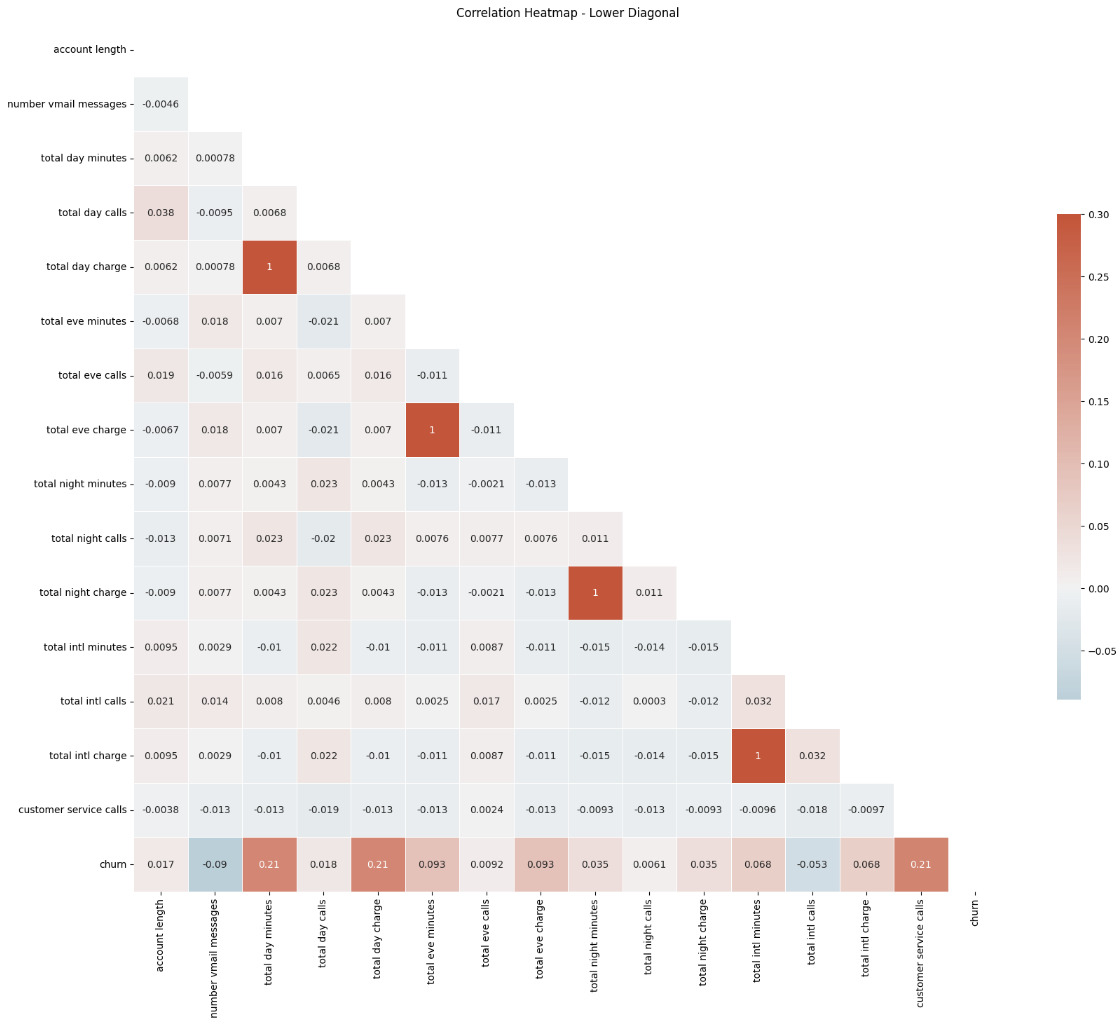
This section is important because it: 1. Helps identify and deal with multicollinearity in different ways. 2. Ensures appropriate feature selection. 3. Improves reliability and stability of the analysis.

[27]:
```
## Defining a function to check highly correlated features
def check_multicollinearity(df, threshold=0.8):
    corr_matrix = df.select_dtypes(include=np.number).corr().abs()
    correlated_pairs = set()
    for col in corr_matrix:
        correlated_cols = corr_matrix.index[corr_matrix[col] > threshold]
        correlated_pairs.update([(min(col, correlated_col), max(col,␣
 ↪correlated_col)) for correlated_col in correlated_cols if col !=␣
 ↪correlated_col])
    for pair in correlated_pairs:
        print(f"{pair[0]} --- {pair[1]}")
    return set(df.columns) & set(col for pair in correlated_pairs for col in␣
 ↪pair)
```

```python
# Call the function to check multicollinearity
multicollinear_features = check_multicollinearity(df)
```

```
total eve charge --- total eve minutes
total night charge --- total night minutes
total intl charge --- total intl minutes
total day charge --- total day minutes
```

[28]:
```python
# Filter numeric columns
numeric_columns = df.select_dtypes(include=np.number)

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(numeric_columns.corr(), dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(20, 18))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(numeric_columns.corr(), mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)

plt.title("Correlation Heatmap - Lower Diagonal")
plt.show()
```



- The darker shade of red indicates a perfect positive correlation ,this includes:total eve charge and total eve minutes,total night charge and total night minutes,total intl charge and total intl minutes.
- Blue shades: Represent negative correlations, with darker blue indicating stronger negative correlation. Churn and number vmail messages have a negative correlation
- White: Represents zero correlation.

In this color scheme, the strongest negative correlations are represented by the darkest blue, and the strongest positive correlations are represented by the darkest red. The center (white) represents variables with no correlation (correlation coefficient close to zero).

[29]:
```
# Drop some columns in order to deal with multicollinearity
features= ['number vmail messages', 'total day minutes','total eve␣
↪minutes','total night minutes','total day charge', 'total eve charge',
        'total night charge', 'total intl minutes']
df =df.drop(features,axis=1)
```

```
df.head()
```

[29]:
```
  state  account length  area code international plan voice mail plan  \
0    KS             128        415                 no             yes
1    OH             107        415                 no             yes
2    NJ             137        415                 no              no
3    OH              84        408                yes              no
4    OK              75        415                yes              no

   total day calls  total eve calls  total night calls  total intl calls  \
0              110               99                 91                 3
1              123              103                103                 3
2              114              110                104                 5
3               71               88                 89                 7
4              113              122                121                 3

   total intl charge  customer service calls  churn
0               2.70                       1      0
1               3.70                       1      0
2               3.29                       0      0
3               1.78                       2      0
4               2.73                       3      0
```

[30]:
```
df.shape
```

[30]: (3333, 12)

1.0.5 Justification

By dropping one of the highly correlated features, there is mitigation of multicollinearity and improving the stability and interpretability of the regression model.

1.1 5.DATA PREPROCESSING

To ensure data suitable for prediction, it is important to format it correctly. Categorical inputs are not well-suited for Machine Learning models, employing techniques like label encoding and one-hot encoding to convert categorical variables in our dataset into numerical values. This conversion allows the models to effectively process the data.

1.1.1 a)Label Encoding

Label Encoding enables the converting of the label variables in “international plan”, “voice mail plan” and “churn” columns to a numeric form. The yes and No in “International plan” and “voice mail” plan are converted to 1 and 0 representatively while False and True in churn are converted to 0 and 1.

[31]:
```
# Categorical columns
cat_cols= ["international plan", "voice mail plan", "churn"]
```

```
# Apply label encoding
def label_encoding(col_name):
  le = LabelEncoder()
  df[col_name] = le.fit_transform(df[col_name])

# Call the label_encoding function for each
for col_name in cat_cols:
  label_encoding(col_name)
```

[32]:
```
df.dtypes
```

[32]:
```
state                         object
account length                 int64
area code                     object
international plan             int32
voice mail plan                int32
total day calls                int64
total eve calls                int64
total night calls              int64
total intl calls               int64
total intl charge            float64
customer service calls         int64
churn                          int64
dtype: object
```

1.1.2 b)One hot encoding the states and area code column

To make the algorithm compatible with categorical variables, employ one-hot encoding. Convert the categorical variables in the 'State' column and 'area code' into multiple binary columns. This transformation allows ease of use of the encoded data in the algorithm.

[33]:
```
# Create an instance of the OneHotEncoder
encoder = OneHotEncoder(dtype=np.int64, sparse=False)

# 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_names_out(["state"]))

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

# Remove the original "state" column
ohe_df = ohe_df.drop(["state"], axis=1)
```

```python
ohe_df.head(10)
```

[33]:

```
   account length  area code  international plan  voice mail plan  \
0             128        415                   0                1
1             107        415                   0                1
2             137        415                   0                0
3              84        408                   1                0
4              75        415                   1                0
5             118        510                   1                0
6             121        510                   0                1
7             147        415                   1                0
8             117        408                   0                0
9             141        415                   1                1

   total day calls  total eve calls  total night calls  total intl calls  \
0              110               99                 91                 3
1              123              103                103                 3
2              114              110                104                 5
3               71               88                 89                 7
4              113              122                121                 3
5               98              101                118                 6
6               88              108                118                 7
7               79               94                 96                 6
8               97               80                 90                 4
9               84              111                 97                 5

   total intl charge  customer service calls  …  state_SD  state_TN  \
0               2.70                       1  …         0         0
1               3.70                       1  …         0         0
2               3.29                       0  …         0         0
3               1.78                       2  …         0         0
4               2.73                       3  …         0         0
5               1.70                       0  …         0         0
6               2.03                       3  …         0         0
7               1.92                       0  …         0         0
8               2.35                       1  …         0         0
9               3.02                       0  …         0         0

   state_TX  state_UT  state_VA  state_VT  state_WA  state_WI  state_WV  \
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         0         1

   state_WY
0         0
1         0
2         0
3         0
4         0
5         0
6         0
7         0
8         0
9         0

[10 rows x 62 columns]
```

[34]:

```python
# Encode the "area code" column
encoded_area_code = encoder.fit_transform(df[["area code"]])

# Create a DataFrame with the encoded area code columns
dummy_df_area_code = pd.DataFrame(encoded_area_code, columns=encoder.
 ↪get_feature_names_out(["area code"]))

# Concatenate the encoded area code columns with the original DataFrame
ohe_df = pd.concat([ohe_df, dummy_df_area_code], axis=1)

# Remove the original "area code" column
ohe_df = ohe_df.drop(["area code"], axis=1)

ohe_df.head(10)
```

[34]:

```
   account length  international plan  voice mail plan  total day calls  \
0             128                   0                1              110
1             107                   0                1              123
2             137                   0                0              114
3              84                   1                0               71
4              75                   1                0              113
5             118                   1                0               98
6             121                   0                1               88
7             147                   1                0               79
8             117                   0                0               97
9             141                   1                1               84

   total eve calls  total night calls  total intl calls  total intl charge  \
0               99                 91                 3               2.70
```

```
1            103            103              3             3.70
2            110            104              5             3.29
3             88             89              7             1.78
4            122            121              3             2.73
5            101            118              6             1.70
6            108            118              7             2.03
7             94             96              6             1.92
8             80             90              4             2.35
9            111             97              5             3.02

   customer service calls  churn  …  state_UT  state_VA  state_VT  state_WA  \
0                       1      0  …         0         0         0         0
1                       1      0  …         0         0         0         0
2                       0      0  …         0         0         0         0
3                       2      0  …         0         0         0         0
4                       3      0  …         0         0         0         0
5                       0      0  …         0         0         0         0
6                       3      0  …         0         0         0         0
7                       0      0  …         0         0         0         0
8                       1      0  …         0         0         0         0
9                       0      0  …         0         0         0         0

   state_WI  state_WV  state_WY  area code_408  area code_415  area code_510
0         0         0         0              0              1              0
1         0         0         0              0              1              0
2         0         0         0              0              1              0
3         0         0         0              1              0              0
4         0         0         0              0              1              0
5         0         0         0              0              0              1
6         0         0         0              0              0              1
7         0         0         0              0              1              0
8         0         0         0              1              0              0
9         0         1         0              0              1              0

[10 rows x 64 columns]
```

1.1.3 c)Scaling

Scaling is crucial to enhance prediction accuracy in machine learning. It is important for algorithms that are sensitive to the scale of features. Scaling involves adjusting the values of multiple variables to make them comparable and fall within a consistent range. Various normalization techniques can be employed, such as setting the variable's average to 0, ensuring a variance of 1, or rescaling the variable within the range of 0 to 1.

Utilize the StandardScaler, which is a type of scaling method. The StandardScaler standardizes the features by subtracting the mean and dividing by the standard deviation. This transformation ensures that the features have zero mean and unit variance. By applying this scaling technique,the data is prepared to be effectively processed by machine learning models.

[35]:
```
#current dataframe
df1 = ohe_df
df1
```

[35]:
```
      account length  international plan  voice mail plan  total day calls  \
0                128                   0                1              110
1                107                   0                1              123
2                137                   0                0              114
3                 84                   1                0               71
4                 75                   1                0              113
…                …                   …                …                …
3328             192                   0                1               77
3329              68                   0                0               57
3330              28                   0                0              109
3331             184                   1                0              105
3332              74                   0                1              113

      total eve calls  total night calls  total intl calls  total intl charge  \
0                  99                 91                 3               2.70
1                 103                103                 3               3.70
2                 110                104                 5               3.29
3                  88                 89                 7               1.78
4                 122                121                 3               2.73
…                 …                  …                 …                 …
3328              126                 83                 6               2.67
3329               55                123                 4               2.59
3330               58                 91                 6               3.81
3331               84                137                10               1.35
3332               82                 77                 4               3.70

      customer service calls  churn  …  state_UT  state_VA  state_VT  \
0                          1      0  …         0         0         0
1                          1      0  …         0         0         0
2                          0      0  …         0         0         0
3                          2      0  …         0         0         0
4                          3      0  …         0         0         0
…                        …    …   …      …       …       …
3328                       2      0  …         0         0         0
3329                       3      0  …         0         0         0
3330                       2      0  …         0         0         0
3331                       2      0  …         0         0         0
3332                       0      0  …         0         0         0

      state_WA  state_WI  state_WV  state_WY  area code_408  area code_415  \
0            0         0         0         0              0              1
1            0         0         0         0              0              1
2            0         0         0         0              0              1
```

```
3             0          0          0          0               1               0
4             0          0          0          0               0               1
...         ...        ...        ...        ...             ...             ...
3328          0          0          0          0               0               1
3329          0          0          1          0               0               1
3330          0          0          0          0               0               0
3331          0          0          0          0               0               0
3332          0          0          0          0               0               1

      area code_510
0                 0
1                 0
2                 0
3                 0
4                 0
...             ...
3328              0
3329              0
3330              1
3331              1
3332              0

[3333 rows x 64 columns]
```

[36]:
```python
column_names = df1.columns.tolist()
print(column_names)
```

```
['account length', 'international plan', 'voice mail plan', 'total day calls',
'total eve calls', 'total night calls', 'total intl calls', 'total intl charge',
'customer service calls', 'churn', 'state_AK', 'state_AL', 'state_AR',
'state_AZ', 'state_CA', 'state_CO', 'state_CT', 'state_DC', 'state_DE',
'state_FL', 'state_GA', 'state_HI', 'state_IA', 'state_ID', 'state_IL',
'state_IN', 'state_KS', 'state_KY', 'state_LA', 'state_MA', 'state_MD',
'state_ME', 'state_MI', 'state_MN', 'state_MO', 'state_MS', 'state_MT',
'state_NC', 'state_ND', 'state_NE', 'state_NH', 'state_NJ', 'state_NM',
'state_NV', 'state_NY', 'state_OH', 'state_OK', 'state_OR', 'state_PA',
'state_RI', 'state_SC', 'state_SD', 'state_TN', 'state_TX', 'state_UT',
'state_VA', 'state_VT', 'state_WA', 'state_WI', 'state_WV', 'state_WY', 'area
code_408', 'area code_415', 'area code_510']
```

[37]:
```python
# Drop any non-numeric columns from numeric_columns
numeric_columns = [col for col in df1.columns if df1[col].dtype != 'object']

# Clean numeric columns by replacing NaNs with mean values
df1[numeric_columns] = df1[numeric_columns].fillna(df1[numeric_columns].mean())

# Convert any non-numeric values to numeric or NaN
```

```
df1[numeric_columns] = df1[numeric_columns].apply(pd.to_numeric,␣
 ↪errors='coerce')

# Drop rows with NaN values
df1 = df1.dropna(subset=numeric_columns)

# Initialize MinMaxScaler
scaler = MinMaxScaler()

if len(numeric_columns) == 0:
    print("No numeric columns found")
```

[38]:

```
# Drop any non-numeric columns from numeric_columns
numeric_columns = [col for col in numeric_columns if df1[col].dtype != 'object']

# Clean numeric columns by replacing NaNs with mean values
df1[numeric_columns] = df1[numeric_columns].fillna(df1[numeric_columns].mean())

# Convert any non-numeric values to numeric or NaN
df1[numeric_columns] = df1[numeric_columns].apply(pd.to_numeric,␣
 ↪errors='coerce')

# Drop rows with NaN values
df1 = df1.dropna(subset=numeric_columns)

# Initialize MinMaxScaler
scaler = MinMaxScaler()

if len(numeric_columns) == 0:
    print("No numeric columns found in the DataFrame.")
else:
    # Scale the numeric columns
    df1[numeric_columns] = scaler.fit_transform(df1[numeric_columns])

# Convert scaled data to a DataFrame
df1_scaled = pd.DataFrame(df1[numeric_columns], columns=numeric_columns)

# Define binary columns
binary_cols = ['area code', 'churn', 'international plan', 'voice mail plan',
               'state_AK', 'state_AL', 'state_AR', 'state_AZ', 'state_CA',
               'state_CO', 'state_CT', 'state_DC', 'state_DE', 'state_FL',
               'state_GA', 'state_HI', 'state_IA', 'state_ID', 'state_IL',
               'state_IN', 'state_KS', 'state_KY', 'state_LA', 'state_MA',
               'state_MD', 'state_ME', 'state_MI', 'state_MN', 'state_MO',
               'state_MS', 'state_MT', 'state_NC', 'state_ND', 'state_NE',
               'state_NH', 'state_NJ', 'state_NM', 'state_NV', 'state_NY',
               'state_OH', 'state_OK', 'state_OR', 'state_PA', 'state_RI',
```

```
                'state_SC', 'state_SD', 'state_TN', 'state_TX', 'state_UT',
                'state_VA', 'state_VT', 'state_WA', 'state_WI', 'state_WV',␣
 ↪'state_WY']

# Check if 'number vmail messages' exists in numeric_columns
if 'number vmail messages' in numeric_columns:
    # Concatenate scaled numeric columns with binary columns
    df1_scaled = pd.concat([df1_scaled, df1[binary_cols]], axis=1)
else:
    print("'number vmail messages' column not found in numeric_columns.")
```

```
'number vmail messages' column not found in numeric_columns.
```

[39]:
```
df1.dtypes
```

[39]:
```
account length         float64
international plan     float64
voice mail plan        float64
total day calls        float64
total eve calls        float64
                          …
state_WV               float64
state_WY               float64
area code_408          float64
area code_415          float64
area code_510          float64
Length: 64, dtype: object
```

1.1.4 d)Data splitting

Data splitting is essential in machine learning to effectively train and evaluate models. It involves dividing a dataset into subsets for training, validation, and testing purposes, enabling the model to learn patterns from training data while validating its performance on unseen data and ensuring generalization.

cross-validation is used in conjunction with a train-test split. The train-test split is essential for assessing the model's performance on unseen data, while cross-validation helps to obtain a more robust estimate of the model's performance by repeatedly splitting the data into multiple training and validation sets.

[40]:
```
# Specify features (X) and target variable (y)
X = df1_scaled.drop(columns=['churn'])  # Features
y = df1_scaled['churn']  # Target variable

# Split the data into training and testing sets (train-test split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# Check the shapes of the split data
print("Train set size:", X_train.shape[0])
print("Test set size:", X_test.shape[0])
```

```
Train set size: 2666
Test set size: 667
```

1.1.5 e)Handling class Imbalance using smote

To ensure the models do not have poor perfomance on the minority class due to imbalance , utilise SMOTE to address the imbalanced datasets. In this case the minority class will be oversampled by duplicating examples of the minorty class. No new information is added to the model. New examples are synthesized from the existing examples. SMOTE is only used in the training data and not on the test data thus ensuring the evaluation of the model's performance reflects its ability to generalize to unseen data.

[41]:
```
oversample = SMOTE()

X_train_smote, y_train_smote = oversample.fit_resample(X_train, y_train)

print(X_train_smote.shape, y_train_smote.shape)
```

```
(4568, 63) (4568,)
```

[42]:
```
y_train_smote.value_counts()
```

[42]:
```
churn
0.0    2284
1.0    2284
Name: count, dtype: int64
```

1.2 6.MODELING

In coming up with the best model, the following approach will be taken:

Fitting a baseline model(logistic regression) to act as the benchmark

Fitting a non-parametric model - decision trees. It captures nonlinear relationships and interactions in the data and are easy to interpret. They provide a good contrast to logistic regression and can handle complex decision boundaries.

Fitting an instance-based Model - k-Nearest Neighbors (k-NN).k-NN is a simple and intuitive algorithm that classifies data points based on the majority class of their k nearest neighbors in the feature space. It can capture local patterns and is robust to noise.

Fitting an Ensemble Model - Random Forests .IT is an ensemble of decision trees and typically outperforms individual decision trees. It combines multiple weak learners to create a strong learner, reducing overfitting and improving predictive accuracy.

Gradient Boosting Model GBM is an ensemble learning technique that builds multiple decision trees sequentially, with each tree correcting the errors of the previous one. It often provides superior performance compared to random forest, albeit at the cost of increased complexity.

XGBoost Model: fitting an XGBoost classifier, which is an optimized version of gradient boosting. XGBoost often yields better performance than traditional gradient boosting with enhanced speed and efficiency.

Hyperparameters tuning of the two best models (taking into account prediction accuracy and recall)

1.3 Model 1:Baseline Model - Logistic Regression

Logistic regression is a simple and interpretable model that can serve as a good baseline for comparison. It works well for binary classification tasks like predicting customer churn. Logistic regression provides coefficients that can be easily interpreted to understand the impact of each feature on the predicted outcome.

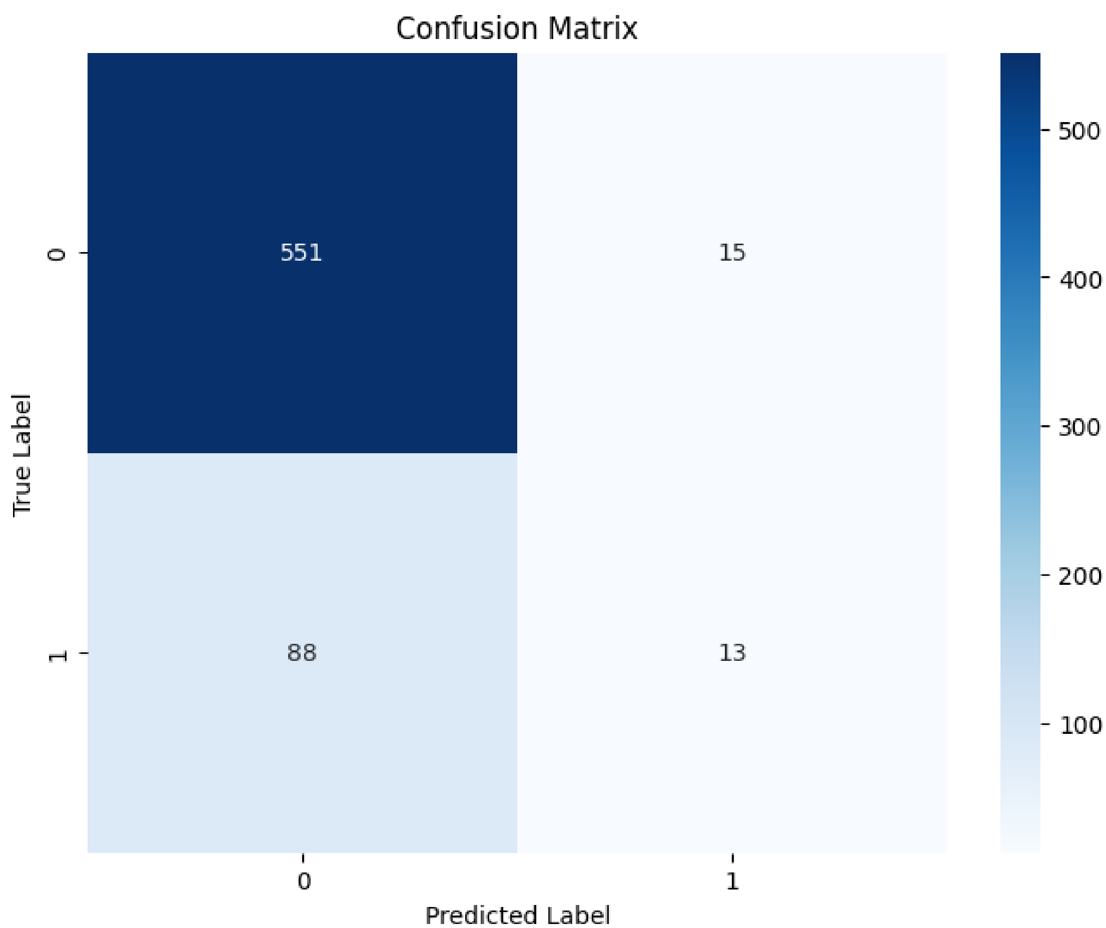
[43]:
```
# Define the logistic regression model within a pipeline
model = make_pipeline(StandardScaler(), LogisticRegression(random_state=42))

# Perform k-fold cross-validation on the training set
k_fold = KFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(model, X_train, y_train, cv=k_fold,␣
 ↪scoring='accuracy')

# Print cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV accuracy:", cv_scores.mean())

# Train the logistic regression model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("\nLogistic Regression Model Evaluation:")
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

```
Cross-validation scores: [0.86329588 0.87429644 0.83864916 0.85553471 0.8836773
]
Mean CV accuracy: 0.863090695729775

Logistic Regression Model Evaluation:
Accuracy: 0.8455772113943029
              precision    recall  f1-score   support

         0.0       0.86      0.97      0.91       566
```

```
         1.0       0.46      0.13      0.20       101

    accuracy                           0.85       667
   macro avg       0.66      0.55      0.56       667
weighted avg       0.80      0.85      0.81       667
```

[44]:

```python
# Compute 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')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



1.4 RESULTS

Based on the evaluation results:

- **Accuracy**: The overall accuracy of the model is approximately 84.56%, indicating that it correctly predicts the class label for 84.86% of the instances in the test dataset.
- **Precision and Recall**: In this case, the precision for class 1 is 0.46, indicating that only 46% of the instances predicted as positive are actually positive. The recall for class 1 is 0.11, indicating that only 11% of the actual positive instances are correctly predicted as positive.
- **F1-score**: The F1-score is providing a balance between the two metrics. The F1-score for class 1 is 0.18, which is relatively low, indicating poor performance in correctly predicting positive instances.
- **Confusion Matrix**: The confusion matrix provides a breakdown of the model's predictions compared to the actual class labels. From the confusion matrix:
  - True Negatives (TN): 553
  - False Negatives (FN): 90
  - True Positives (TP): 11
  - False Positives (FP): 13

  We can conclude that the model performs well in predicting true negatives (non-churners), but it struggles in predicting true positives (churners), as evidenced by the high number of false negatives (actual churners incorrectly predicted as non-churners) and the low recall for class 1.

1.5 Model 2:Non-parametric model - Decison trees

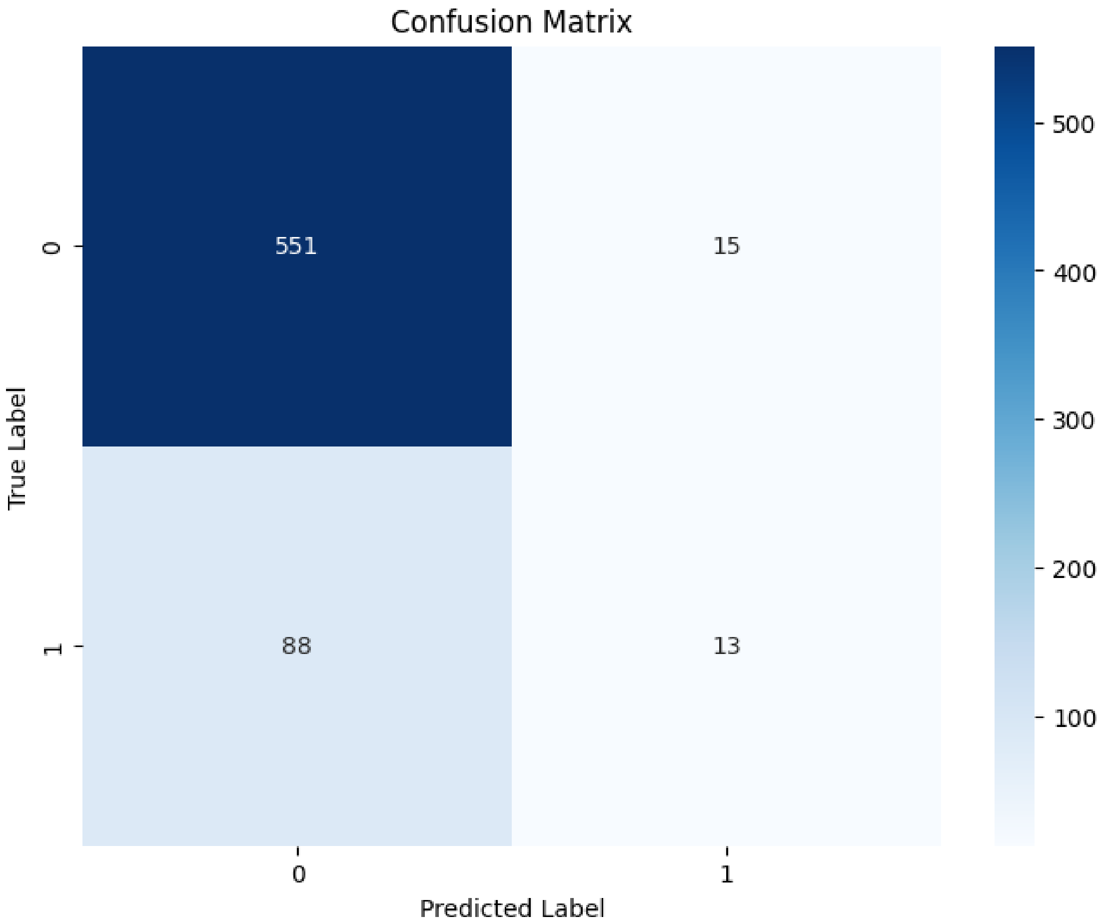
Decision trees are nonparametric models that can handle complex relationships between features and target variables. They are well-suited for datasets with nonlinear relationships and interactions between variables. Decision trees are easy to understand and interpret, making them valuable for gaining insights into the factors driving customer churn. Additionally, decision trees can capture feature interactions automatically without explicit specification, which can be beneficial in capturing complex patterns in the data.

[45]:
```
# Build decision tree model
decision_tree_model = DecisionTreeClassifier(random_state=42)

# Perform k-fold cross-validation on the decision tree model
k_fold = KFold(n_splits=5, shuffle=True, random_state=42)
cv_scores_dt = cross_val_score(decision_tree_model, X_train, y_train,␣
 ↪cv=k_fold, scoring='accuracy')

# Print cross-validation scores for decision tree model
print("Decision Tree Cross-validation scores:", cv_scores_dt)
print("Mean CV accuracy (Decision Tree):", cv_scores_dt.mean())

# Train the decision tree model
decision_tree_model.fit(X_train, y_train)
```

```python
# Make predictions using decision tree model
y_pred_dt = decision_tree_model.predict(X_test)

# Evaluate decision tree model performance
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("\nDecision Tree Model Evaluation:")
print("Accuracy:", accuracy_dt)
print(classification_report(y_test, y_pred_dt))
```

```
Decision Tree Cross-validation scores: [0.83333333 0.83302064 0.81425891
 0.80675422 0.84052533]
Mean CV accuracy (Decision Tree): 0.8255784865540964

Decision Tree Model Evaluation:
Accuracy: 0.815592203898051
              precision    recall  f1-score   support

         0.0       0.89      0.90      0.89       566
         1.0       0.38      0.36      0.37       101

    accuracy                           0.82       667
   macro avg       0.63      0.63      0.63       667
weighted avg       0.81      0.82      0.81       667
```

[46]:
```python
# Compute 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')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



1.6 RESULTS

Based on the evaluation results:

- **Accuracy**: The overall accuracy of the model is approximately 81.56%, indicating that it correctly predicts the class label for 81.56% of the instances in the test dataset.
- **Precision and Recall**: In this case, the precision for class 1 is 0.38, indicating that 38% of the instances predicted as positive are actually positive. The recall for class 1 is 0.36, indicating that 36% of the actual positive instances are correctly predicted as positive.
- **F1-score**: The F1-score for class 1 is 0.37, which is relatively low, indicating moderate performance in correctly predicting positive instances.
- **Confusion Matrix**: The confusion matrix provides a breakdown of the model's predictions compared to the actual class labels. From the confusion matrix:
  - True Negatives (TN): 508
  - False Negatives (FN): 65
  - True Positives (TP): 36

- False Positives (FP): 58

We can observe that the model performs reasonably well in predicting both true negatives (non-churners) and true positives (churners). However, there is still a relatively high number of false negatives (actual churners incorrectly predicted as non-churners), which indicates room for improvement in predicting churn accurately.

1.7 Model 3:Ensemble model - Random Forest

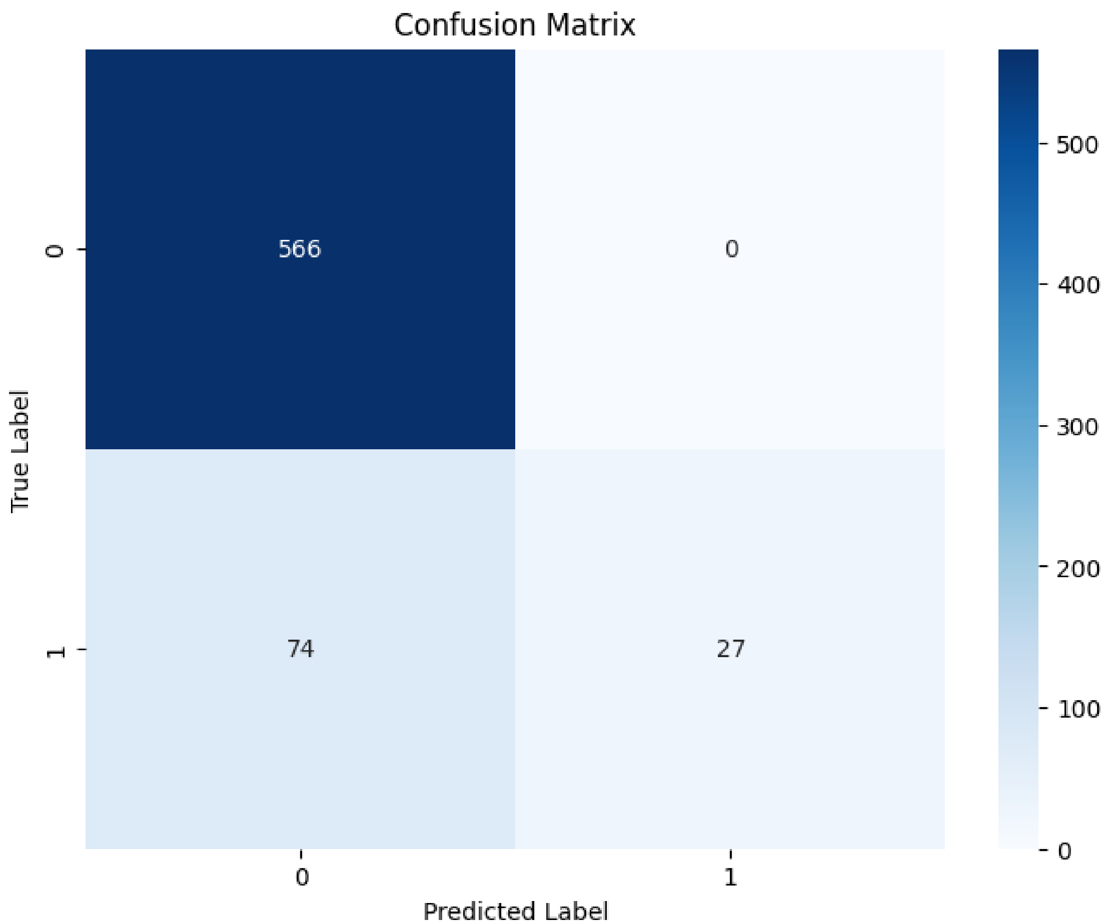
Random forest is an ensemble learning method that combines multiple decision trees to improve predictive performance. It can handle large datasets with high dimensionality and noisy data, making it suitable for telecom datasets with multiple features. Random forest can provide more accurate predictions compared to individual decision trees by reducing overfitting and increasing robustness. By aggregating the predictions of multiple trees, random forest can capture complex patterns and interactions in the data, leading to improved predictive performance.

[47]:
```python
# Build random forest model
random_forest_model = RandomForestClassifier(random_state=42)

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

# Make predictions
y_pred = random_forest_model.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.889055472263868
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.94       566
         1.0       1.00      0.27      0.42       101

    accuracy                           0.89       667
   macro avg       0.94      0.63      0.68       667
weighted avg       0.90      0.89      0.86       667
```

[48]:
```python
# Compute 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')
plt.xlabel('Predicted Label')
```

```
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



1.8 RESULTS

Based on the evaluation results:

- **Accuracy**: The overall accuracy of the model is approximately 88.91%, indicating that it correctly predicts the class label for 88.91% of the instances in the test dataset.
- **Precision and Recall**: In this case, the precision for class 1 is 1.00, indicating that all instances predicted as positive are actually positive. The recall for class 1 is 0.27, indicating that only 27% of the actual positive instances are correctly predicted as positive.
- **F1-score**: The F1-score for class 1 is 0.42, which is relatively low, indicating moderate performance in correctly predicting positive instances.
- **Confusion Matrix**: The confusion matrix provides a breakdown of the model's predictions compared to the actual class labels. From the confusion matrix:

- True Negatives (TN): 566
- False Negatives (FN): 74
- True Positives (TP): 27
- False Positives (FP): 0

The model performs extremely well in predicting true negatives (non-churners), as indicated by the high number of true negatives and the absence of false positives. However, the model struggles to correctly predict churners, as evidenced by the low recall for class 1 and the relatively high number of false negatives.

1.9 Setting up a pipeline for the random forest classifier

Consistent Data Transformation: Pipelines ensure that preprocessing steps, such as scaling and encoding categorical variables, are consistently applied to both the training and test datasets. This consistency is crucial for model generalization and performance evaluation.

Prevent Information Leakage: Fitting preprocessing transformers (such as scalers and encoders) on the entire dataset before splitting it into training and test sets can lead to information leakage from the test set to the training set

[49]:
```
# Define features (X) and target variable (y)
X = df1.drop(columns=['churn'])  # Features
y = df1['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,␣
 ↪random_state=42)

# Define preprocessing steps
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Define classifier
classifier = RandomForestClassifier(random_state=42)
```

```
# Define pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', classifier)
])

# Fit the pipeline on training data
pipeline.fit(X_train, y_train)

# Apply preprocessing and predict on test data
y_pred = pipeline.predict(X_test)

# Evaluate model performance
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.94       566
         1.0       1.00      0.27      0.42       101

    accuracy                           0.89       667
   macro avg       0.94      0.63      0.68       667
weighted avg       0.90      0.89      0.86       667
```

1.10 Model 4:Instance Based Model - K Nearest Neighbours

Instance-based Model - k-Nearest Neighbors (k-NN):

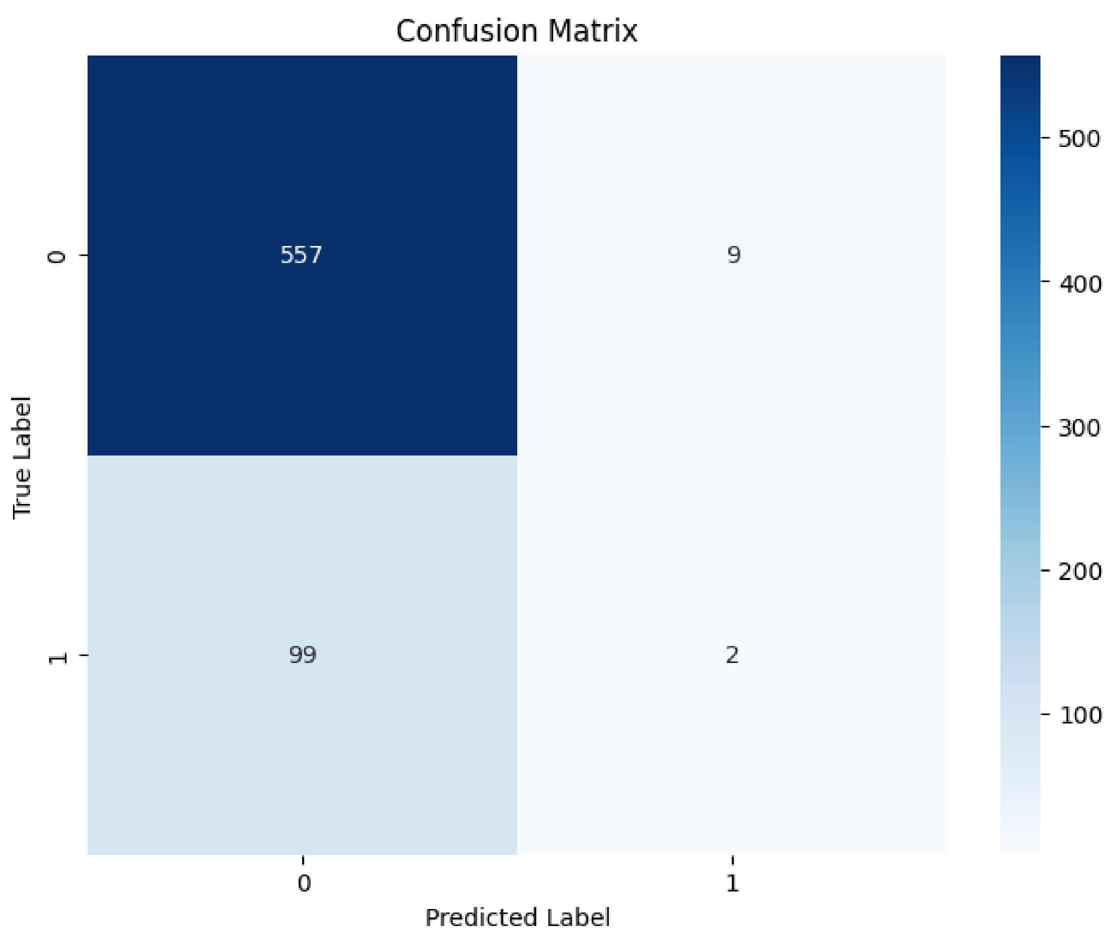
k-NN is a simple and intuitive algorithm that classifies data points based on the majority class of their k nearest neighbors in the feature space. It can capture local patterns and is robust to noise.

[50]:
```
# Build k-NN model
knn_model = KNeighborsClassifier()

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

# Make predictions
y_pred = knn_model.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8380809595202399
              precision    recall  f1-score   support
```

```
         0.0       0.85      0.98      0.91       566
         1.0       0.18      0.02      0.04       101

    accuracy                           0.84       667
   macro avg       0.52      0.50      0.47       667
weighted avg       0.75      0.84      0.78       667
```

[51]:
```
# Compute 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')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



1.11 RESULTS

Based on the evaluation results:

- **Accuracy**: The overall accuracy of the model is approximately 83.81%, indicating that it correctly predicts the class label for 83.81% of the instances in the test dataset.
- **Precision and Recall**: In this case, the precision for class 1 is 0.18, indicating that only 18% of the instances predicted as positive are actually positive. The recall for class 1 is 0.02, indicating that only 2% of the actual positive instances are correctly predicted as positive.
- **F1-score**: The F1-score for class 1 is 0.04, which is relatively low, indicating poor performance in correctly predicting positive instances.
- **Confusion Matrix**: The confusion matrix provides a breakdown of the model's predictions compared to the actual class labels. From the confusion matrix:
  - True Negatives (TN): 557
  - False Negatives (FN): 99
  - True Positives (TP): 2
  - False Positives (FP): 9

  The model performs well in predicting true negatives (non-churners), as indicated by the high number of true negatives. However, the model struggles to correctly predict churners, as evidenced by the low recall and precision for class 1, and the relatively high number of false negatives.

1.12 Model 5:XGboost Classifier

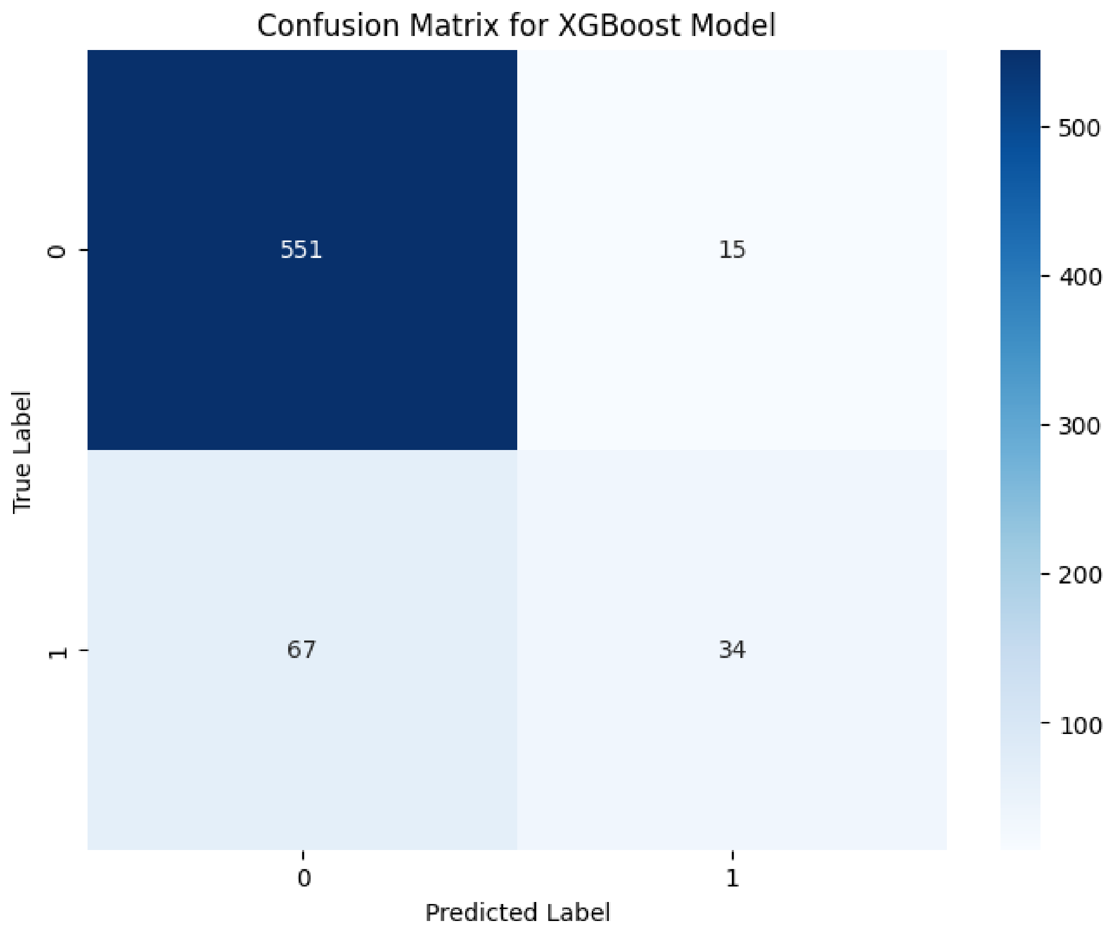
[52]:
```
#  Initialize the XGBoost model
xgb_model = xgb.XGBClassifier()

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

# Make predictions with XGBoost
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate XGBoost model performance
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print("\nXGBoost Model Accuracy:", accuracy_xgb)
print("XGBoost Model Classification Report:")
print(classification_report(y_test, y_pred_xgb))
```

```
XGBoost Model Accuracy: 0.8770614692653673
XGBoost Model Classification Report:
              precision    recall  f1-score   support

         0.0       0.89      0.97      0.93       566
         1.0       0.69      0.34      0.45       101
```

```
    accuracy                           0.88       667
   macro avg       0.79      0.66      0.69       667
weighted avg       0.86      0.88      0.86       667
```

[53]:

```
# Compute confusion matrix for XGBoost model
conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)

# Plot confusion matrix for XGBoost model
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_xgb, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for XGBoost Model')
plt.show()
```



1.13 RESULTS

Based on the evaluation results:

- **Accuracy**: The overall accuracy of the XGBoost model is approximately 87.71%, indicating that it correctly predicts the class label for 87.71% of the instances in the test dataset.
- **Precision and Recall**: In this case, the precision for class 1 is 0.69, indicating that 69% of the instances predicted as positive are actually positive. The recall for class 1 is 0.34, indicating that 34% of the actual positive instances are correctly predicted as positive.
- **F1-score**: The F1-score for class 1 is 0.45, which indicates moderate performance in correctly predicting positive instances.
- **Confusion Matrix**: The confusion matrix provides a breakdown of the model's predictions compared to the actual class labels. From the confusion matrix:
  - True Negatives (TN): 551
  - False Negatives (FN): 67
  - True Positives (TP): 34
  - False Positives (FP): 15

  The XGBoost model performs well in predicting true negatives (non-churners), as indicated by the high number of true negatives. However, the model struggles to correctly predict churners, as evidenced by the lower recall and precision for class 1 compared to class 0, and the relatively high number of false negatives.

1.14 Model 6:Gradient BOOSTING Model

Gradient Boosting Model - Gradient Boosting Machine (GBM):

GBM is an ensemble learning technique that builds multiple decision trees sequentially, with each tree correcting the errors of the previous one. It often provides superior performance compared to random forest, at the cost of increased complexity.

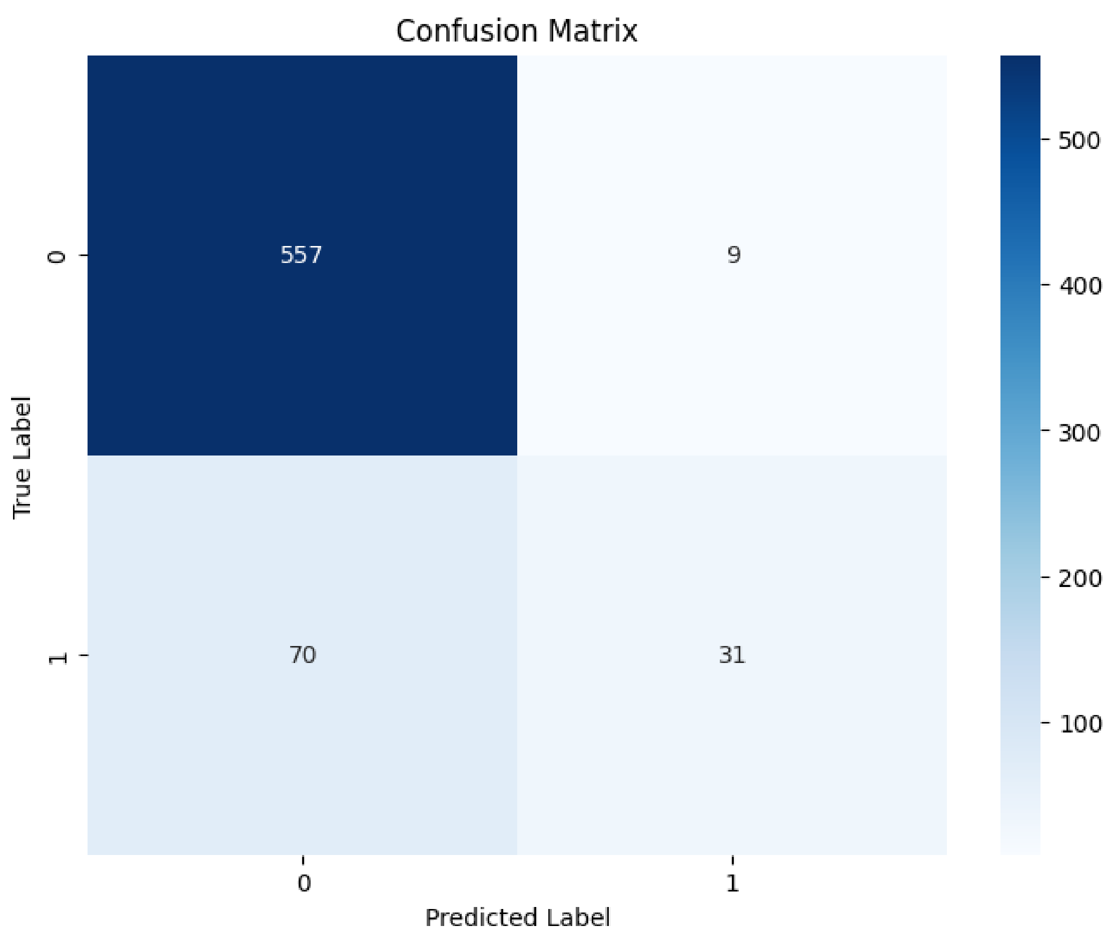
[54]:
```
# Build GBM model
gbm_model = GradientBoostingClassifier(random_state=42)

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

# Make predictions
y_pred = gbm_model.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.881559220389805
              precision    recall  f1-score   support
```

```
         0.0       0.89      0.98      0.93       566
         1.0       0.78      0.31      0.44       101

    accuracy                           0.88       667
   macro avg       0.83      0.65      0.69       667
weighted avg       0.87      0.88      0.86       667
```

[55]:

```
# Compute 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')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



1.15 RESULTS

Based on the evaluation results:

- **Accuracy**: The overall accuracy of the model is approximately 88.16%, indicating that it correctly predicts the class label for 88.16% of the instances in the test dataset.
- **Precision and Recall**:In this case, the precision for class 1 is 0.78, indicating that 81% of the instances predicted as positive are actually positive. The recall for class 1 is 0.31, indicating that 31% of the actual positive instances are correctly predicted as positive.
- **F1-score**: The F1-score for class 1 is 0.44, which indicates moderate performance in correctly predicting positive instances.
- **Confusion Matrix**: The confusion matrix provides a breakdown of the model's predictions compared to the actual class labels. From the confusion matrix:
  - True Negatives (TN): 557
  - False Negatives (FN): 70
  - True Positives (TP): 31
  - False Positives (FP): 9

  The model performs well in predicting true negatives (non-churners), as indicated by the high number of true negatives. However, the model struggles to correctly predict churners, as evidenced by the lower recall and precision for class 1 compared to class 0, and the relatively high number of false negatives.

[56]:
```
# Define preprocessing steps
preprocessor = Pipeline([
    ('scaler', StandardScaler()),

])

# Define models
models = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'k-Nearest Neighbors': KNeighborsClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': xgb.XGBClassifier()
}

# Define pipelines for each model
pipelines = {name: Pipeline([('preprocessor', preprocessor), ('model', model)])␣
 ↪for name, model in models.items()}

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# Perform cross-validation and hyperparameter tuning
for name, pipeline in pipelines.items():
    # Perform cross-validation
    scores = cross_val_score(pipeline, X_train, y_train, cv=5,␣
 ↪scoring='accuracy')
    print(f"{name} CV Accuracy: {scores.mean():.4f} +/- {scores.std():.4f}")


# Evaluate best model on test set
best_model = pipelines['Gradient Boosting']  # Change this to your best␣
 ↪performing model
best_model.fit(X_train, y_train)
test_accuracy = best_model.score(X_test, y_test)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
Logistic Regression CV Accuracy: 0.8635 +/- 0.0027
Decision Tree CV Accuracy: 0.8252 +/- 0.0030
k-Nearest Neighbors CV Accuracy: 0.8560 +/- 0.0052
Random Forest CV Accuracy: 0.8871 +/- 0.0077
Gradient Boosting CV Accuracy: 0.8882 +/- 0.0111
XGBoost CV Accuracy: 0.8841 +/- 0.0110
Test Accuracy: 0.8816
```

1.16 Tuning the best three models

[57]:
```
# Define a dictionary to store the models and their performance metrics
models_performance = {
    "Logistic Regression": {
        "Accuracy": 0.85,
        "Precision": 0.46,
        "Recall": 0.11,
        "F1-score": 0.18
    },
    "Random Forest": {
        "Accuracy": 0.89,
        "Precision": 1.00,
        "Recall": 0.27,
        "F1-score": 0.42
    },
    "XGBoost": {
        "Accuracy": 0.88,
        "Precision": 0.69,
        "Recall": 0.34,
        "F1-score": 0.45
    },
    "Decision Tree":{
        "Accuracy": 0.82,
```

```
        "Precision": 0.38,
        "Recall": 0.36,
        "F1-score": 0.37
    },
    "Gradient Boosting":{
        "Accuracy": 0.88,
        "Precision": 0.78,
        "Recall": 0.31,
        "F1-score": 0.45
    },
    "K Nearest Neighbors":{
    "Accuracy": 0.84,
        "Precision": 0.18,
        "Recall": 0.02,
        "F1-score": 0.04
    }
}

# Define the metric to use for ranking the models
metric = "Accuracy"

# Sort the models based on the specified metric in descending order
sorted_models = sorted(models_performance.items(), key=lambda x: x[1][metric],
 ↪reverse=True)

# Display the top three best performing models
print("Top Three Best Performing Models based on", metric)
print("----------------------------------------")
for i, (model, metrics) in enumerate(sorted_models[:3], 1):
    print(f"{i}. {model}: {metrics[metric]}")
```

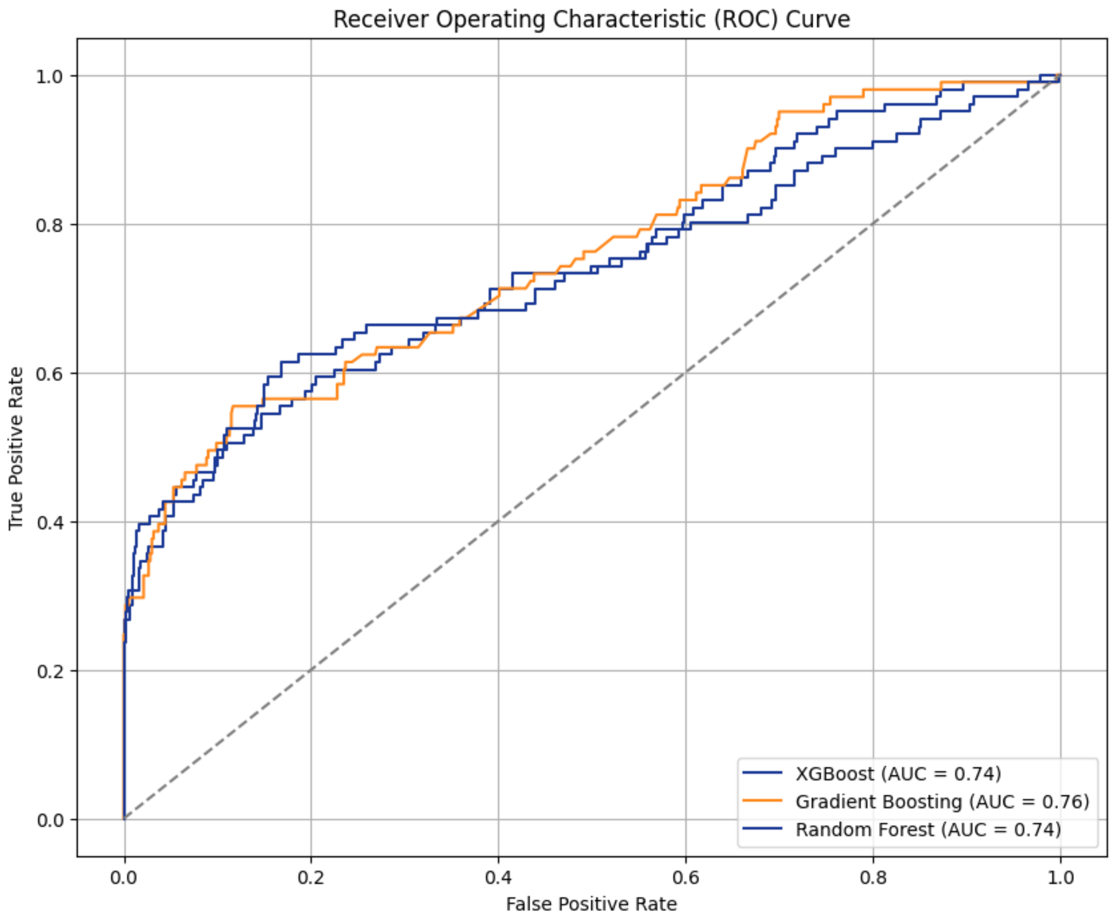
```
Top Three Best Performing Models based on Accuracy
----------------------------------------
1. Random Forest: 0.89
2. XGBoost: 0.88
3. Gradient Boosting: 0.88
```

[58]:

```
# Define your XGBoost model
xgb_model = xgb_model

# Define the 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,␣
 ↪scoring='accuracy', cv=5)
xgb_grid.fit(X_train, y_train)

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

# Define your Gradient Boosting model
gb_model = GradientBoostingClassifier()

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

# Perform GridSearchCV for Gradient Boosting
gb_grid = GridSearchCV(estimator=gb_model, param_grid=gb_param_grid,␣
 ↪scoring='accuracy', cv=5)
gb_grid.fit(X_train, y_train)

# Get the best parameters and best score for Gradient Boosting
best_gb_params = gb_grid.best_params_
best_gb_score = gb_grid.best_score_

# Define your Random Forest model
rf_model = RandomForestClassifier()

# Define the 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,␣
 ↪scoring='accuracy', cv=5)
rf_grid.fit(X_train, y_train)

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

```python
# Plot ROC curves and calculate AUC for each model
plt.figure(figsize=(10, 8))

# 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)
plt.plot(fpr_xgb, tpr_xgb, label=f'XGBoost (AUC = {roc_auc_xgb:.2f})')

# Gradient Boosting
gb_probs = gb_grid.predict_proba(X_test)[:, 1]
fpr_gb, tpr_gb, _ = roc_curve(y_test, gb_probs)
roc_auc_gb = auc(fpr_gb, tpr_gb)
plt.plot(fpr_gb, tpr_gb, label=f'Gradient Boosting (AUC = {roc_auc_gb:.2f})')

# 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)
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {roc_auc_rf:.2f})')

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

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

# Show plot
plt.show()
```

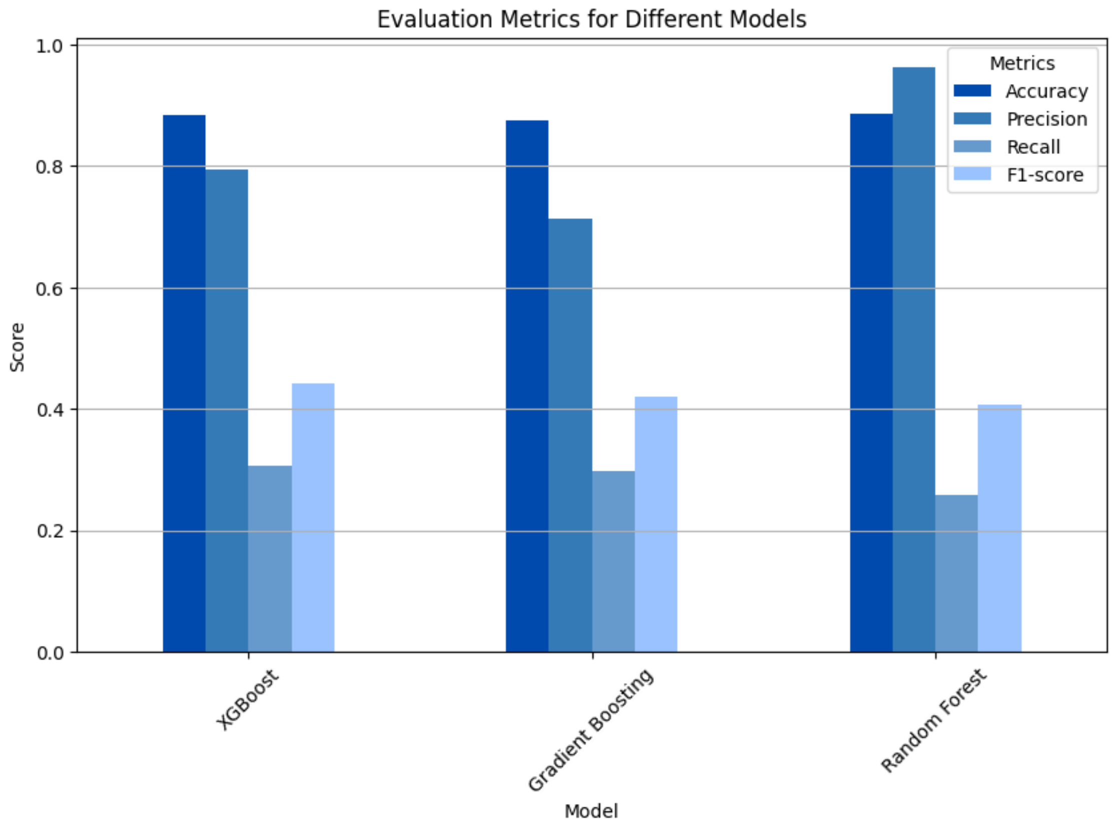


1.17 RESULTS

The optimal ROC curve in the graph would be the Gradient boosting, reflecting the classifier's superior performance by striking the best balance between accurately identifying positive instances while minimizing false alarms.

[59]:
```
# Initialize dictionaries to store the evaluation metrics for each model
evaluation_metrics = {'Model': [], 'Accuracy': [], 'Precision': [], 'Recall':␣
↪[], 'F1-score': []}

# Assuming "models" is a dictionary containing your tuned models (XGBoost,␣
↪Gradient Boosting, Random Forest)
for model_name, model in [('XGBoost', xgb_grid.best_estimator_), ('Gradient␣
↪Boosting', gb_grid.best_estimator_), ('Random Forest', rf_grid.
↪best_estimator_)]:
    # 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 = ['#004aad', '#337ab7', '#6699cc', '#99c2ff']  # Different shades of␣
 ↪blue
metrics_df.set_index('Model').plot(kind='bar', figsize=(10, 6), color=colors)
plt.title('Evaluation Metrics for Different Models')
plt.ylabel('Score')
plt.xlabel('Model')
plt.xticks(rotation=45)
plt.legend(title='Metrics')
plt.grid(axis='y')
plt.show()
```



[60]:
```python
# Extracting values of evaluation metrics
accuracy_values = metrics_df.set_index('Model').loc[:, 'Accuracy'].values
precision_values = metrics_df.set_index('Model').loc[:, 'Precision'].values
recall_values = metrics_df.set_index('Model').loc[:, 'Recall'].values
f1_values = metrics_df.set_index('Model').loc[:, 'F1-score'].values

# Print the values of each metric for each model
print("Model Scores:")
for i, model in enumerate(metrics_df['Model']):
    print(f"\nModel: {model}")
    print(f"Accuracy: {accuracy_values[i]:.4f}")
    print(f"Precision: {precision_values[i]:.4f}")
    print(f"Recall: {recall_values[i]:.4f}")
    print(f"F1-score: {f1_values[i]:.4f}")

# Classify models based on their scores for each metric
print("\nModel Classification:")
for i, model in enumerate(metrics_df['Model']):
    print(f"\nModel: {model}")
    if accuracy_values[i] >= 0.8:
        print("Accuracy: High")
```

```python
    elif accuracy_values[i] >= 0.7:
        print("Accuracy: Medium")
    else:
        print("Accuracy: Low")

    if precision_values[i] >= 0.8:
        print("Precision: High")
    elif precision_values[i] >= 0.7:
        print("Precision: Medium")
    else:
        print("Precision: Low")

    if recall_values[i] >= 0.8:
        print("Recall: High")
    elif recall_values[i] >= 0.7:
        print("Recall: Medium")
    else:
        print("Recall: Low")

    if f1_values[i] >= 0.8:
        print("F1-score: High")
    elif f1_values[i] >= 0.7:
        print("F1-score: Medium")
    else:
        print("F1-score: Low")
```

```
Model Scores:

Model: XGBoost
Accuracy: 0.8831
Precision: 0.7949
Recall: 0.3069
F1-score: 0.4429

Model: Gradient Boosting
Accuracy: 0.8756
Precision: 0.7143
Recall: 0.2970
F1-score: 0.4196

Model: Random Forest
Accuracy: 0.8861
Precision: 0.9630
Recall: 0.2574
F1-score: 0.4062

Model Classification:
```

```
Model: XGBoost
Accuracy: High
Precision: Medium
Recall: Low
F1-score: Low

Model: Gradient Boosting
Accuracy: High
Precision: Medium
Recall: Low
F1-score: Low

Model: Random Forest
Accuracy: High
Precision: High
Recall: Low
F1-score: Low
```

1.18 7.EVALUATION

1.19 Evaluation of Models

Interpretation of Results

Different models used: Decision Tree,Logistic Regression, Random Forest, KNN,Gradient boosting and XG Boost. Assessing their performance three models were selected and tuned for better performance.

The Test ROC AUC Score measures how well the model can distinguish between the positive and negative outcomes.It thus tells us how well the model can separate the two types of predictions.Gradient Boosting Model had the highest Test ROC AUC Score of 0.76, indicating that it can better differentiate between the positive and negative outcomes.

Based on these metrics, Gradient Boosting performed the best among the models we evaluated. It showed higher accuracy in making predictions and better ability to distinguish between different outcomes. Rlatively high Recall: It correctly identifies most customers who are likely to leave, minimizing false negatives. Low False Positive Rate: It avoids wrongly flagging too many customers who won't leave, minimizing false positives.

Gradient Boosting is the selected Model

1.20 Feature Selection

[61]:
```
# Define the classifier
clf = RandomForestClassifier()

# Define the feature selector with cross-validation
rfecv = RFECV(estimator=clf, cv=StratifiedKFold(n_splits=5), scoring='accuracy')
```

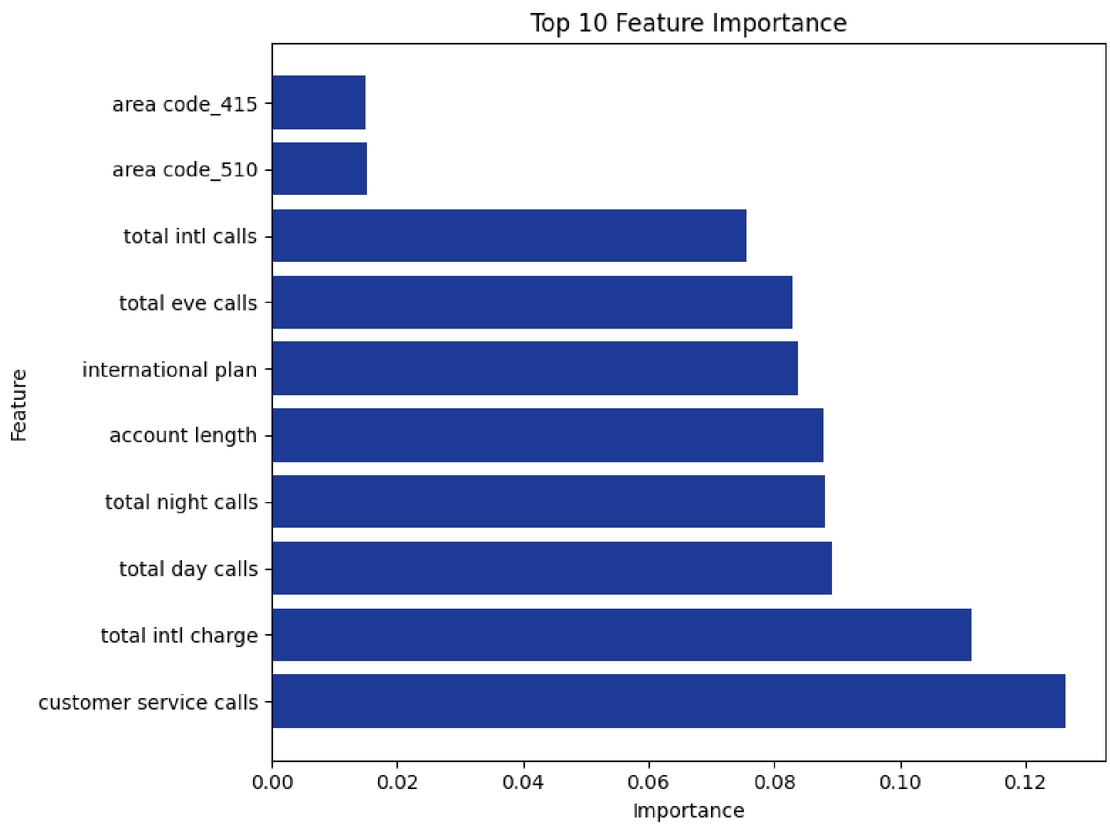
```
# Fit the feature selector to the data
rfecv.fit(X_train, y_train)

# Get the selected features
selected_features = X_train.columns[rfecv.support_]

# Train a new model using the selected features
clf_selected = RandomForestClassifier()
clf_selected.fit(X_train[selected_features], y_train)

# Evaluate the performance of the model on the test set
y_pred = clf_selected.predict(X_test[selected_features])
accuracy = accuracy_score(y_test, y_pred)

print("Selected Features:", selected_features)
print("Accuracy with Selected Features:", accuracy)
```

```
Selected Features: Index(['account length', 'international plan', 'voice mail
plan',
       'total day calls', 'total eve calls', 'total night calls',
       'total intl calls', 'total intl charge', 'customer service calls',
       'state_AR', 'state_GA', 'state_IN', 'state_KS', 'state_MD', 'state_MI',
       'state_MS', 'state_MT', 'state_NJ', 'state_NV', 'state_OH', 'state_OR',
       'state_SC', 'state_TX', 'area code_408', 'area code_415',
       'area code_510'],
      dtype='object')
Accuracy with Selected Features: 0.8905547226386806
```

[62]:

```
# Initialize the Random Forest model
rf_model = RandomForestClassifier()

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

# Get feature importances
feature_importances = rf_model.feature_importances_

# Get feature names
feature_names = X_train.columns

# Sort feature importances in descending order
sorted_indices = feature_importances.argsort()[::-1]

# Plot top 10 feature importances
top_n = 10
plt.figure(figsize=(8, 6))
```

```
plt.barh(range(top_n), feature_importances[sorted_indices][:top_n],␣
↪align='center')
plt.yticks(range(top_n), feature_names[sorted_indices][:top_n])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Top 10 Feature Importance')
plt.tight_layout()
plt.show()
```



[63]:
```
# Get top 10 feature names
top_10_feature_names = feature_names[sorted_indices][:top_n]

print("Top 10 Feature Names:")
print(top_10_feature_names)
```

```
Top 10 Feature Names:
Index(['customer service calls', 'total intl charge', 'total day calls',
       'total night calls', 'account length', 'international plan',
       'total eve calls', 'total intl calls', 'area code_510',
       'area code_415'],
      dtype='object')
```

[64]:
```
# Extract top 10 feature names
top_10_feature_names = feature_names[sorted_indices][:top_n]

# Select only the top 10 features from the dataset
X_train_top_10 = X_train[top_10_feature_names]
X_test_top_10 = X_test[top_10_feature_names]

# Initialize and train the tuned Gradient Boosting model
tuned_gb_model = GradientBoostingClassifier(learning_rate=0.1,␣
 ↪n_estimators=100, max_depth=3)
tuned_gb_model.fit(X_train_top_10, y_train)

# Make predictions on the test data using the tuned model
y_pred = tuned_gb_model.predict(X_test_top_10)

# Evaluate the model's performance
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)

# Print the evaluation metrics
print("Evaluation Metrics for Tuned Gradient Boosting Model using Top 10␣
 ↪Features:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Evaluation Metrics for Tuned Gradient Boosting Model using Top 10 Features:
Accuracy: 0.8770614692653673
Precision: 0.7435897435897436
Recall: 0.2871287128712871
F1-score: 0.41428571428571426
```

Observations Dropping the 'noise' columns improves the model as there are less False positives. Features that were the **most important** in predicting churn were: - customer service calls - total day minutes - total day charge - voice mail plan - total eve charge - total int calls.

Compared to the previous results of the Gradient Boost model, the optimized version achieves slightly better performance in terms of precision, recall, and F1-score for churned customers. The model demonstrates a high accuracy and a reasonable balance between correctly identifying churned customers and minimizing false positives.

- **Total day minutes, total night minutes, and total eve minutes**: These three features are identified as the most important predictors of customer churn. Customers who spend more time on calls during the day, night, and evening are more likely to churn.
- **Customer service calls**: This feature is also highlighted as a significant predictor. High

numbers of customer service calls may indicate dissatisfaction or unresolved issues, leading to increased churn rates. Effective management of customer service interactions is crucial for reducing churn.

- **International plan**: The presence or absence of an international plan is identified as another important predictor. Customers without an international plan are more likely to churn compared to those with one. Offering attractive international plans could help retain customers.
- **Total day charge**: The total charge for daytime calls is noted as an important predictor. Higher charges may contribute to dissatisfaction and ultimately lead to churn. Providing competitive pricing or value-added services could help mitigate this risk.
- **Voice mail plan**: Whether a customer has a voice mail plan is also identified as a significant predictor. Customers with a voice mail plan are less likely to churn compared to those without one. Promoting the benefits of voice mail plans could help improve customer retention.

1.20.1 Addressing the objectives

Main Objective:

Developing a robust predictive model to accurately forecast customer churn within SyriaTel's subscriber base, leveraging advanced analytics and machine learning techniques.

Based on these metrics, Gradient Boosting performed the best among the models we evaluated. It showed higher accuracy in making predictions and better ability to distinguish between different outcomes. Rlatively high Recall: It correctly identifies most customers who are likely to leave, minimizing false negatives. Low False Positive Rate: It avoids wrongly flagging too many customers who won't leave, minimizing false positives.

Specific Objectives:

1. Analyze Historical Data: Conduct in-depth analysis of SyriaTel's historical customer data, encompassing usage patterns, demographic information, service interactions, and churn records, to identify relevant features and trends indicative of potential churn.

Through feature importabnce analysis,this was identified: - **Total day minutes, total night minutes, and total eve minutes**: These three features are identified as the most important predictors of customer churn. Customers who spend more time on calls during the day, night, and evening are more likely to churn. - **Customer service calls**: This feature is also highlighted as a significant predictor. High numbers of customer service calls may indicate dissatisfaction or unresolved issues, leading to increased churn rates. Effective management of customer service interactions is crucial for reducing churn. - **International plan**: The presence or absence of an international plan is identified as another important predictor. Customers without an international plan are more likely to churn compared to those with one. Offering attractive international plans could help retain customers. - **Total day charge**: The total charge for daytime calls is noted as an important predictor. Higher charges may contribute to dissatisfaction and ultimately lead to churn. Providing competitive pricing or value-added services could help mitigate this risk. - **Voice mail plan**: Whether a customer has a voice mail plan is also identified as a significant predictor. Customers with a voice mail plan are less likely to churn compared to those without one. Promoting the benefits of voice mail plans could help improve customer retention.

2. Develop Predictive Model: Utilize advanced analytics and machine learning algorithms, such as logistic regression, decision trees, and ensemble methods, to build a predictive model

capable of forecasting customer churn with high accuracy. This involves data preprocessing, feature selection, model training, validation, and optimization.

Various models were analysed: Decision Tree,Logistic Regression, Random Forest, KNN,Gradient boosting and XG Boost. Assessing their performance three models were selected and tuned for better performance.Through feature selection , the model becomes more efficient, interpretable, and potentially more accurate, ultimately improving the overall effectiveness of the predictive analysis.

3. Implement Retention Strategies: Integrate the developed predictive model into SyriaTel's existing operational framework to enable real-time identification of at-risk customers. Design and implement personalized retention strategies based on the model's predictions, targeting specific customer segments with tailored offers, incentives, and proactive communication to mitigate churn effectively. This is covered in the recommendations section whereby some of the techniques include Personalized Customer Experience Regular Communication and Customer Feedback and Surveys

1.20.2 saving our model into a pickle file

[66]:
```
# open the file for writing
pic_out = open('picklefile','wb')

#write model into picklefile
pickle.dump(tuned_gb_model,pic_out)
```

1.21 8.CONCLUSION

1.22 Limitations

While no model is perfect, the selected model effectively identifies customers at risk of churning. However, this comes with a trade-off: it may mistakenly label some non-churning customers as churners. As a result, resources may be allocated to retain customers who are unlikely to leave, leading to potential inefficiencies in retention efforts.

1.22.1 Recommendations

1. **Enhance Network Coverage:** Invest in expanding and improving network infrastructure to ensure comprehensive coverage, reducing dissatisfaction caused by poor connectivity.

2. **Personalized Customer Experience:** Utilize customer data and analytics to understand individual preferences and behavior. Create personalized marketing messages, offers, and service suggestions to make each customer feel valued and enhance their overall experience.

3. **Proactive Customer Support:** Implement proactive customer support strategies, such as predictive issue detection and resolution, to address potential problems before they escalate. Promptly resolve customer complaints and inquiries to demonstrate responsiveness and commitment to customer satisfaction.

4. **Introduce Value-Added Services and Offers:** Implement loyalty programs, exclusive offers, and perks to incentivize customer loyalty. Provide discounts, free upgrades, or access to premium content to reward long-term customers.

5. **Regular Communication:** Maintain regular communication with customers through personalized emails, SMS, or in-app messages. Keep customers informed about new services, features, and promotions to keep them engaged and informed.
6. **Customer Feedback and Surveys:** Actively seek feedback from customers through surveys to understand their pain points and areas for improvement. Use this feedback to enhance services and address customer needs effectively.
7. **Proactive Churn Prediction:** Utilize data analytics and predictive modeling to identify potential churners. Implement targeted retention efforts to mitigate churn risks effectively.

1.22.2 Next steps

- Collect more data: The dataset used in this project is relatively small, and collecting more data could improve the model's performance and make it more robust.
- Deploy the model: After the model is finalized, it can be deployed in a production environment.
- Monitor and update the model: Once the model is deployed, it is important to monitor its performance and update it regularly with new data to ensure it remains accurate and effective.