Phase3_Project / index.ipynb

Annolyne Readme.md 89eedc0 · 28 minutes ago

2.45 MB

ML-Driven Churn Reduction: Syriatel's Customer Retention Strategy

Author: Annolyne Chemutai

School: Moringa school

Data Science PT

Phase 3 project

Introduction

Business growth and development remains a central motivator in organizational decision-making and policy making. Although every business leader aspires to achieve growth in revenues, clientele, and profitability, they must try as much as possible to avoid making losses.

In recent years, such leaders, as well as business experts, have identified customer satisfaction as an important factor to ensuring such growth and development. Without customers, a business would not make any sales, record any cash inflows in terms of revenues, nor make any profits. This underscores the the need for organizations to implement measures that retain existing customers.

Recent technological advancements have also contributed to an increased business rivalry, especially due to increased startups and entrants. Such competition, coupled with an augmented saturation of markets, means that it has become harder and more expensive for businesses in most sectors to acquire new clients, which means they must shift their focus to cementing relationships with existing customers.

Through this project, we are building a prediction model that identifies patterns in customer churning that would best suit Syriatel Mobile Telecom and in which can be helpful in developing mitigation strategies on customer churn rates. The project is structured as follows:

1. **Business Understanding**
2. **Data Understanding**
3. **Data Preparation**
4. **Exploratory Data Analysis**
5. **Modelling**
6. **Model Evaluation**
7. **Recommendations and Conclusions**

Business Understanding

With an increasing blend of factors such as competition, technological innovations, and globalization in the telecommunication markets, **Syriatel Mobile Telecom** has

emphasized the need to improve customer satisfaction and preserve its clientele. The Syrian telecommunication giant reiterates its commitment to maintaining its market position by establishing `"its reputation by focusing on customer satisfaction and social responsibility."`

Although these efforts have been fruitful over the years, the company needs to increase its commitment to reducing customer churn rates, which might threaten its market position, profitability, and overall growth.

Hence, this project will help **Syriatel Mobile Telecom** identify customers with the highest probabilities of churning, which will be crucial for implementing new policies and business frameworks intended to ensure retention.

Primary stakeholder:

- Syriatel Mobile Telecom

Other Stakeholders:

- Shareholders
- Employees
- Customers

As the principal stakeholder, the company stands to benefit from this model through a reduction in customer churn rates, which has the potential to increase revenues and profits, promote growth, and sustain or even increase its market position. The customers will also benefit through improved telecommunication services and better customer service. As the company continues to grow through increased revenues, profits, customer base, and market share, the shareholders will also get more returns on their investments (ROI), while employees benefit from better remuneration and bonuses.

The project aims to provide value to the different stakeholders by identifying predictable patterns related to customer churn, which can help **SyriaTel** take proactive measures to retain customers and minimize revenue loss.

Research Objectives:

1. To identify the key features that determine if a customer is likely to churn.
2. To determine the most suitable model for predicting customer churn.
3. To establish a customer retention strategy to reduce churn.

Key Research Inquiries:

- What are the primary indicators of customer churn for Syriatel Mobile Telecom?
- Which predictive model demonstrates the highest efficacy in forecasting customer churn?
- What retention strategies could Syriatel Mobile Telecom employ to mitigate

customer attrition rates?

Data Understanding

The Kaggle dataset "Churn in Telecom" contains information about customer activity and subscription cancellations with a telecom company. This dataset aims to facilitate the development of predictive models that can help the telecom business reduce revenue loss from short-term customers.

The dataset comprises 3,333 entries across 21 columns. These columns include information such as state, account length, area code, phone number, international plan, voice mail plan, number of voice mail messages, and various usage metrics for day, evening, night, and international calls. It also includes data on customer service calls and churn status.

In this project phase, we'll familiarize ourselves with the data and identify potential quality issues. We'll also conduct initial exploratory data analysis to uncover preliminary insights.

Summary of Features in the Dataset

- **State:** The customer's state of residence
- **Account Length:** Number of days the customer has held an account
- **Area Code:** Customer's area code
- **Phone Number:** Customer's phone number
- **International Plan:** True if the customer has an international plan, false otherwise
- **Voice Mail Plan:** True if the customer has a voice mail plan, false otherwise
- **Number Vmail Messages:** Number of voicemails sent by the customer
- **Total Day Minutes:** Total minutes of daytime calls
- **Total Day Calls:** Total number of daytime calls
- **Total Day Charge:** Total charge for daytime calls
- **Total Eve Minutes:** Total minutes of evening calls
- **Total Eve Calls:** Total number of evening calls
- **Total Eve Charge:** Total charge for evening calls
- **Total Night Minutes:** Total minutes of nighttime calls
- **Total Night Calls:** Total number of nighttime calls
- **Total Night Charge:** Total charge for nighttime calls
- **Total Intl Minutes:** Total minutes of international calls
- **Total Intl Calls:** Total number of international calls
- **Total Intl Charge:** Total charge for international calls
- **Customer Service Calls:** Number of calls made to customer service
- **Churn:** True if the customer terminated their contract, false otherwise

Data Preparation

In this section, we will prepare our data for exploratory data analysis and modeling. We'll start by importing the necessary libraries and loading the dataset using pandas. Next, we'll preview the data, examining the number of features and records, as well as statistical characteristics. Finally, we'll conduct thorough data preprocessing, which includes checking for and removing any missing values, and transforming the data as needed.

To begin, we'll import all the required libraries for this project and load the data into a pandas DataFrame.

In [48]:
```
# Importing libraries.
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, classification_report, confusion_
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1
from matplotlib import pyplot as plt
%matplotlib inline
plt.style.use('seaborn-darkgrid')
```

In [49]:
```
#Loading the data into a pandas dataframe
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
```

Next, is to analyze the dataset to determine feature count, identify missing values, and pinpoint columns needing transformation for modeling. This step provides crucial insights for subsequent analysis.

In [50]:
```
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
```

In [51]:
```
# checking for the general shape of the df
df.shape
```

Out[51]:
```
(3333, 21)
```

Our dataset contains 3333 records across 21 columns with no null values. Further review is needed to identify potential anomalies. - 4 object columns, 8 integer columns ,8 float columns and 1 boolean column. The target variable is "churn", with remaining columns as features.

In [52]:
```
# checking top 10 rows
df.head(10)
```

Out[52]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	ch
0	KS	128	415	382-4657	no	yes	25	265.1	110	4
1	OH	107	415	371-7191	no	yes	26	161.6	123	2
2	NJ	137	415	358-1921	no	no	0	243.4	114	4
3	OH	84	408	375-9999	yes	no	0	299.4	71	5
4	OK	75	415	330-6626	yes	no	0	166.7	113	2
5	AL	118	510	391-8027	yes	no	0	223.4	98	3
6	MA	121	510	355-9993	no	yes	24	218.2	88	3
7	MO	147	415	329-	yes	no	0	157.0	79	1

7	MO	147	415	9001	yes	no	0	157.0	79	2
8	LA	117	408	335-4719	no	no	0	184.5	97	3
9	WV	141	415	330-8173	yes	yes	37	258.6	84	4

10 rows × 21 columns

In [53]:
```
# Previewing the top 10 rows
df.tail(10)
```

Out[53]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls
3323	IN	117	415	362-5899	no	no	0	118.4	126
3324	WV	159	415	377-1164	no	no	0	169.8	114
3325	OH	78	408	368-8555	no	no	0	193.4	99
3326	OH	96	415	347-6812	no	no	0	106.6	128
3327	SC	79	415	348-3830	no	no	0	134.7	98
3328	AZ	192	415	414-4276	no	yes	36	156.2	77
3329	WV	68	415	370-3271	no	no	0	231.1	57
3330	RI	28	510	328-8230	no	no	0	180.8	109
3331	CT	184	510	364-6381	yes	no	0	213.8	105
3332	TN	74	415	400-4344	no	yes	25	234.4	113

10 rows × 21 columns

Many column names as seen above have multiple words. To improve accessibility, we'll remove whitespaces by replacing them with underscores ('_') in the column names.

In [54]:
```
# Removing whitespaces in column names and replacing with underscores
```

```
df.columns = df.columns.str.replace(' ', '_', regex=False)
```

In [55]:
```
# Previewing the top 10 rows
df.tail(10)
```

Out[55]:

	state	account_length	area_code	phone_number	international_plan	voice_mai
3323	IN	117	415	362-5899	no	
3324	WV	159	415	377-1164	no	
3325	OH	78	408	368-8555	no	
3326	OH	96	415	347-6812	no	
3327	SC	79	415	348-3830	no	
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	

10 rows × 21 columns

In [56]:
```
#Viewing the statistical details such as std, percentile, count, and the mean
df.describe()
```

Out[56]:

	account_length	area_code	number_vmail_messages	total_day_minutes	total
count	3333.000000	3333.000000	3333.000000	3333.000000	33
mean	101.064806	437.182418	8.099010	179.775098	1
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	1
75%	127.000000	510.000000	20.000000	216.400000	1
max	243.000000	510.000000	51.000000	350.800000	1

Data Cleaning

Below cell checks for missing values across all the columns

In [57]:
```
#checking for missing values (nan) in the dataframe
missing_values = df.isnull().sum()
print(missing_values)
```

```
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. However its important to review df further to identify values that are not a representation of the data

In [58]:
```
# checking for value_count for the different state abbreviations
df.state.value_counts()
```

Out[58]:
```
WV    106
MN     84
NY     83
AL     80
OH     78
OR     78
WI     78
VA     77
WY     77
CT     74
MI     73
VT     73
ID     73
UT     72
TX     72
IN     71
KS     70
MD     70
NJ     68
MT     68
NC     68
CO     66
NV     66
WA     66
```

```
MA     65
RI     65
MS     65
AZ     64
FL     63
MO     63
NM     62
ND     62
ME     62
DE     61
OK     61
NE     61
SD     60
SC     60
KY     59
IL     58
NH     56
AR     55
GA     54
DC     54
TN     53
HI     53
AK     52
LA     51
PA     45
IA     44
CA     34
Name: state, dtype: int64
```

I will drop the state column since the area code column provides sufficient geographical information. Multiple subscribers can share an area code, so there's no need to check for duplicates in this column.

In [59]:
```
# dropping the state column
df = df.drop('state', axis=1)
```

In [60]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   account_length          3333 non-null   int64  
 1   area_code               3333 non-null   int64  
 2   phone_number            3333 non-null   object 
 3   international_plan      3333 non-null   object 
 4   voice_mail_plan         3333 non-null   object 
 5   number_vmail_messages   3333 non-null   int64  
 6   total_day_minutes       3333 non-null   float64
 7   total_day_calls         3333 non-null   int64  
 8   total_day_charge        3333 non-null   float64
 9   total_eve_minutes       3333 non-null   float64
 10  total_eve_calls         3333 non-null   int64  
 11  total_eve_charge        3333 non-null   float64
 12  total_night_minutes     3333 non-null   float64
 13  total_night_calls       3333 non-null   int64  
```

```
 14  total_night_charge      3333 non-null   float64
 15  total_intl_minutes      3333 non-null   float64
 16  total_intl_calls        3333 non-null   int64
 17  total_intl_charge       3333 non-null   float64
 18  customer_service_calls  3333 non-null   int64
 19  churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 498.1+ KB
```

Next I will check the Account length Column

In [61]:
```
#checking account length column
df.account_length.value_counts()
```

Out[61]:
```
105    43
87     42
93     40
101    40
90     39
       ..
191     1
199     1
215     1
221     1
2       1
Name: account_length, Length: 212, dtype: int64
```

Also review the *Area Code Column* for the possibilities of unique or missing values

In [62]:
```
df.area_code.unique()
```

Out[62]:
```
array([415, 408, 510], dtype=int64)
```

In [63]:
```
df.area_code.value_counts()
```

Out[63]:
```
415    1655
510     840
408     838
Name: area_code, dtype: int64
```

Reviewing the Phone Number Column

In [64]:
```
df.phone_number
```

Out[64]:
```
0       382-4657
1       371-7191
2       358-1921
3       375-9999
4       330-6626
          ...
3328    414-4276
3329    370-3271
3330    328-8230
3331    364-6381
3332    400-4344
```

```
Name: phone_number, Length: 3333, dtype: object
```

In [65]:
```
df.phone_number.unique
```

Out[65]:
```
<bound method Series.unique of 0       382-4657
1       371-7191
2       358-1921
3       375-9999
4       330-6626
          ...
3328    414-4276
3329    370-3271
3330    328-8230
3331    364-6381
3332    400-4344
Name: phone_number, Length: 3333, dtype: object>
```

The phone_number column is currently an object datatype. We'll convert it to an integer for proper numerical processing. To do this, we need to remove the '-' and convert the dtype to integer

In [66]:
```
# Removing dashes from 'phone_number' and converting to integer
df['phone_number'] = df['phone_number'].str.replace('-', '').astype(int)
```

In [67]:
```
# checking if above conversion is effected
df.phone_number
```

Out[67]:
```
0       3824657
1       3717191
2       3581921
3       3759999
4       3306626
         ...
3328    4144276
3329    3703271
3330    3288230
3331    3646381
3332    4004344
Name: phone_number, Length: 3333, dtype: int32
```

In [68]:
```
# Check for duplicates in the 'phone number' column
duplicates = df.duplicated('phone_number')

# Filter the DataFrame to show only the duplicate rows
duplicate_rows = df[duplicates]
duplicate_rows
```

Out[68]:

account_length	area_code	phone_number	international_plan	voice_mail_plan	nun

No duplicates in the phone number column. phone number is a representation of one customer this means this will form out unique identifier.

In [69]:
```
# Setting 'phone_number' column as the index
df.set_index('phone_number', inplace=True)
```

In [70]:
```
# Previewing the general information of the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3333 entries, 3824657 to 4004344
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   account_length          3333 non-null   int64  
 1   area_code               3333 non-null   int64  
 2   international_plan      3333 non-null   object 
 3   voice_mail_plan         3333 non-null   object 
 4   number_vmail_messages   3333 non-null   int64  
 5   total_day_minutes       3333 non-null   float64
 6   total_day_calls         3333 non-null   int64  
 7   total_day_charge        3333 non-null   float64
 8   total_eve_minutes       3333 non-null   float64
 9   total_eve_calls         3333 non-null   int64  
 10  total_eve_charge        3333 non-null   float64
 11  total_night_minutes     3333 non-null   float64
 12  total_night_calls       3333 non-null   int64  
 13  total_night_charge      3333 non-null   float64
 14  total_intl_minutes      3333 non-null   float64
 15  total_intl_calls        3333 non-null   int64  
 16  total_intl_charge       3333 non-null   float64
 17  customer_service_calls  3333 non-null   int64  
 18  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(2)
memory usage: 498.0+ KB
```

In [71]:
```
# Display the DataFrame to confirm changes
df
```

Out[71]:

phone_number	account_length	area_code	international_plan	voice_mail_plan	numb
3824657	128	415	no	yes	
3717191	107	415	no	yes	
3581921	137	415	no	no	
3759999	84	408	yes	no	
3306626	75	415	yes	no	
...	...	...	...	...	
4144276	192	415	no	yes	
3703271	68	415	no	no	
3288230	28	510	no	no	

3646381	184	510	yes	no
4004344	74	415	no	yes

3333 rows × 19 columns

Lets check on the Churn Column which will be our target variable and check for any anormalies.

In [72]:
```
#Reviewing the churn column
df.churn.value_counts()
```

Out[72]:
```
False    2850
True      483
Name: churn, dtype: int64
```

The data shows 2850 false values (non-churned clients) and 483 true values (churned clients).

Exploratory Data Analysis

In this section, we will conduct a comprehensive exploration of the data through univariate, bivariate, and `multivariate analysis` .

This thorough data exploration aims to identify potential correlations among features and variable distributions, which will prove crucial for feature engineering and modeling.

Univariate Analysis

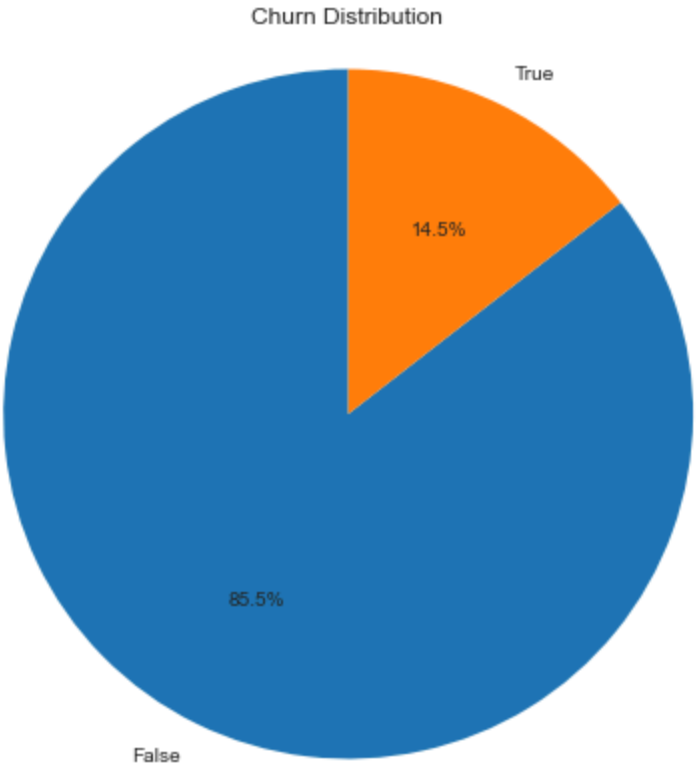
`Univariate data analysis` involves examining a single variable at a time. In our project, we'll use this method to study the distribution of each feature in the dataset. This will help us understand their characteristics and identify potential issues like outliers.

We'll begin with the target variable column, `churn` . This categorical variable uses boolean values (`True` and `False`) to indicate whether a client is likely to churn or not.

To start, we'll visualize the distribution of this column using a pie chart.

In [73]:
```
import matplotlib.pyplot as plt

# Create a pie chart to visualize churn distribution
churn_counts = df['churn'].value_counts()
plt.figure(figsize=(10, 7))
plt.pie(churn_counts, labels=churn_counts.index, autopct='%1.1f%%', startangl
```

```
plt.title('Churn Distribution')
plt.axis('equal')  # Ensures the pie is a circle
plt.show()
```



The dataset contains 3,333 customers, with 483 ending their contracts. This means 14.5% of customers were lost.

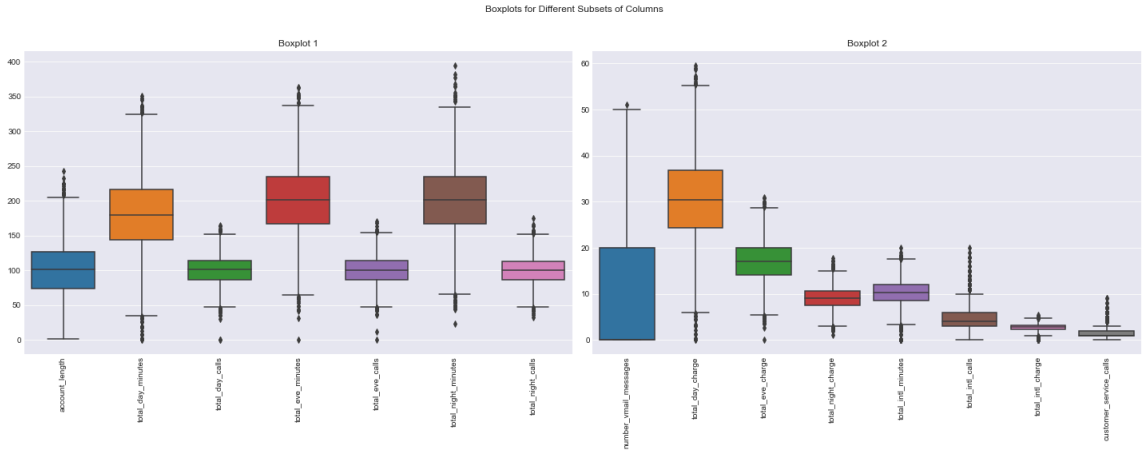
The uneven split between churned and non-churned customers creates a data imbalance. We need to fix this before modeling to avoid inaccurate predictions.

Next lets check for outliers

In [74]:

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

# Define columns for the boxplots
cols = {
    'Boxplot 1': ['account_length', 'total_day_minutes', 'total_day_calls',
                  'total_eve_minutes', 'total_eve_calls', 'total_night_minute
    'Boxplot 2': ['number_vmail_messages', 'total_day_charge', 'total_eve_cha
                  'total_night_charge', 'total_intl_minutes', 'total_intl_cal
}

# Create a figure with two subplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 8))

# Plot each subset of columns in its own boxplot
for ax, (title, columns) in zip(axes, cols.items()):
    sns.boxplot(data=df[columns], ax=ax)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```

```
    ax.set_title(title)

# Set the main title for the figure
fig.suptitle('Boxplots for Different Subsets of Columns')

# Display the plot
plt.tight_layout(rect=[0, 0, 1, 0.96])  # Adjust layout to make room for the
plt.show()
```



In our dataset, we confidently assert that all outliers possess critical information essential for our models. As such, we will retain every data point without exception, ensuring our analysis captures the full spectrum of customer behaviors and patterns.

Bivariate Analysis

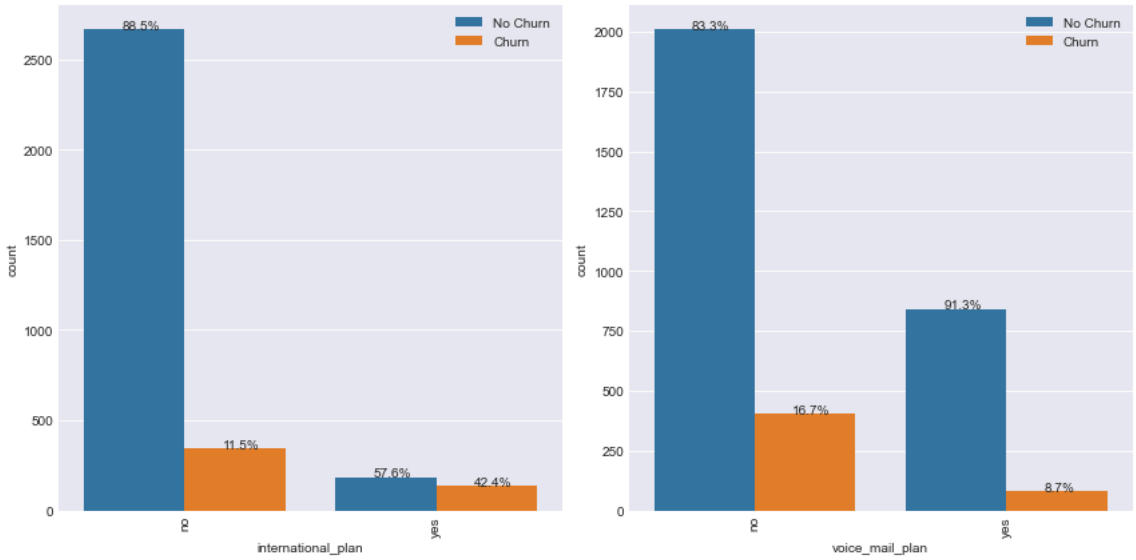
Bivariate analysis examines the relationship between two variables. In our project, we explore how each feature relates to the target variable (customer churn) to understand their connections.

We're analyzing customer churn in relation to `state`, `area code`, `international plan`, and `voice mail plan`. Our goal is to identify potential correlations between these categorical columns and the customer churn rate.

In [75]:

```
categoric_cols = ['international_plan','voice_mail_plan']

fig, axes = plt.subplots(nrows=1, ncols=len(categoric_cols), figsize=(12, 6))

for i, col in enumerate(categoric_cols):
    ax = sns.countplot(x=col, hue="churn", data=df, order=df[col].value_count
    axes[i].set_xticklabels(axes[i].get_xticklabels(), rotation=90)
    handles, labels = axes[i].get_legend_handles_labels()
    axes[i].legend(handles, ['No Churn', 'Churn'], loc="upper right")

    # Calculate the total number of observations within each group
    totals = df.groupby(col)["churn"].count().values

    # Iterate over the rectangles in the plot
    for j, p in enumerate(ax.patches):
        # Calculate the percentage of observations in each group
```

```
        percentage = '{:.1f}%'.format(100 * p.get_height()/totals[j % 2])
        # Add text annotations with the calculated percentages
        x = p.get_x() + p.get_width() / 2 - 0.05
        y = p.get_y() + p.get_height()
        ax.annotate(percentage, (x, y), size=10)

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

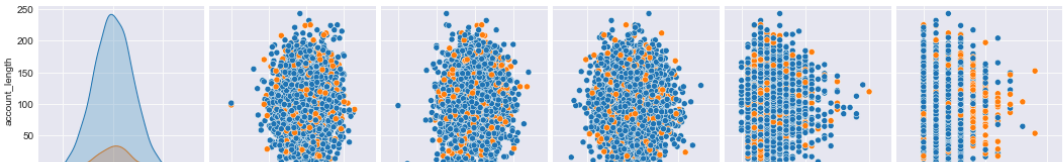


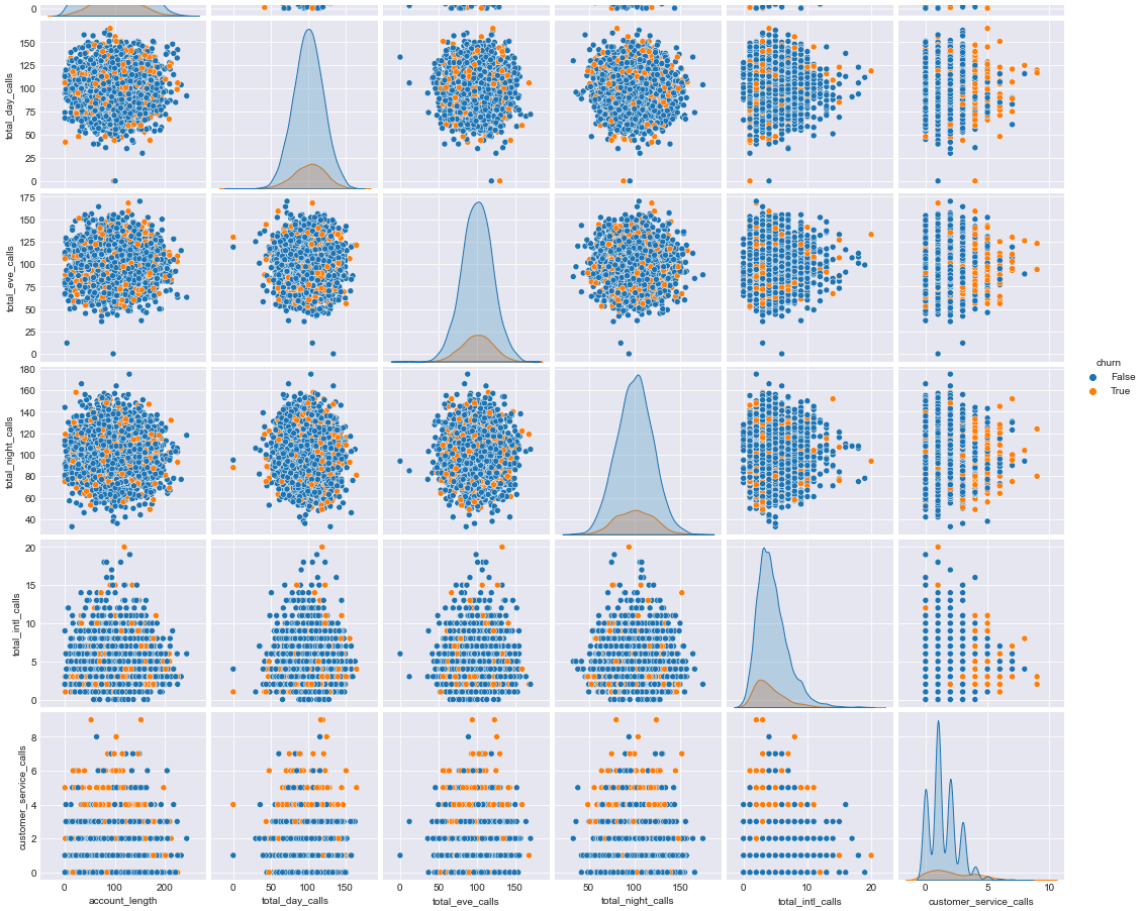
International plan subscribers had higher churn rates (42.4%) than non-subscribers (11.5%), suggesting potential issues with the plan. Conversely, voice mail plan subscribers showed lower churn rates (8.7%) compared to non-subscribers (16.7%), indicating it may reduce churn likelihood.

We then visualize feature correlations with customer churn using pairplots. This helps identify which factors may influence a customer's decision to leave.

In [76]:

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

# Select relevant columns for pairplot
data_temp = df[["account_length", "total_day_calls", "total_eve_calls",
                "total_night_calls", "total_intl_calls", "customer_service_ca

# Create pairplot
sns.pairplot(data_temp, hue="churn", height=2.5)

# Show plot
plt.show()
```





Our analysis reveals a significant correlation between the frequency of customer service calls and churn rates. The data indicates that customers who make more than four service calls are substantially more likely to terminate their service.

Furthermore, the high volume of customer service calls generally indicates dissatisfaction with the provided service. When customers need to make more than four calls, it suggests that their issues are not being resolved efficiently, which consequently increases the probability of service discontinuation..

Multi-variate Analysis

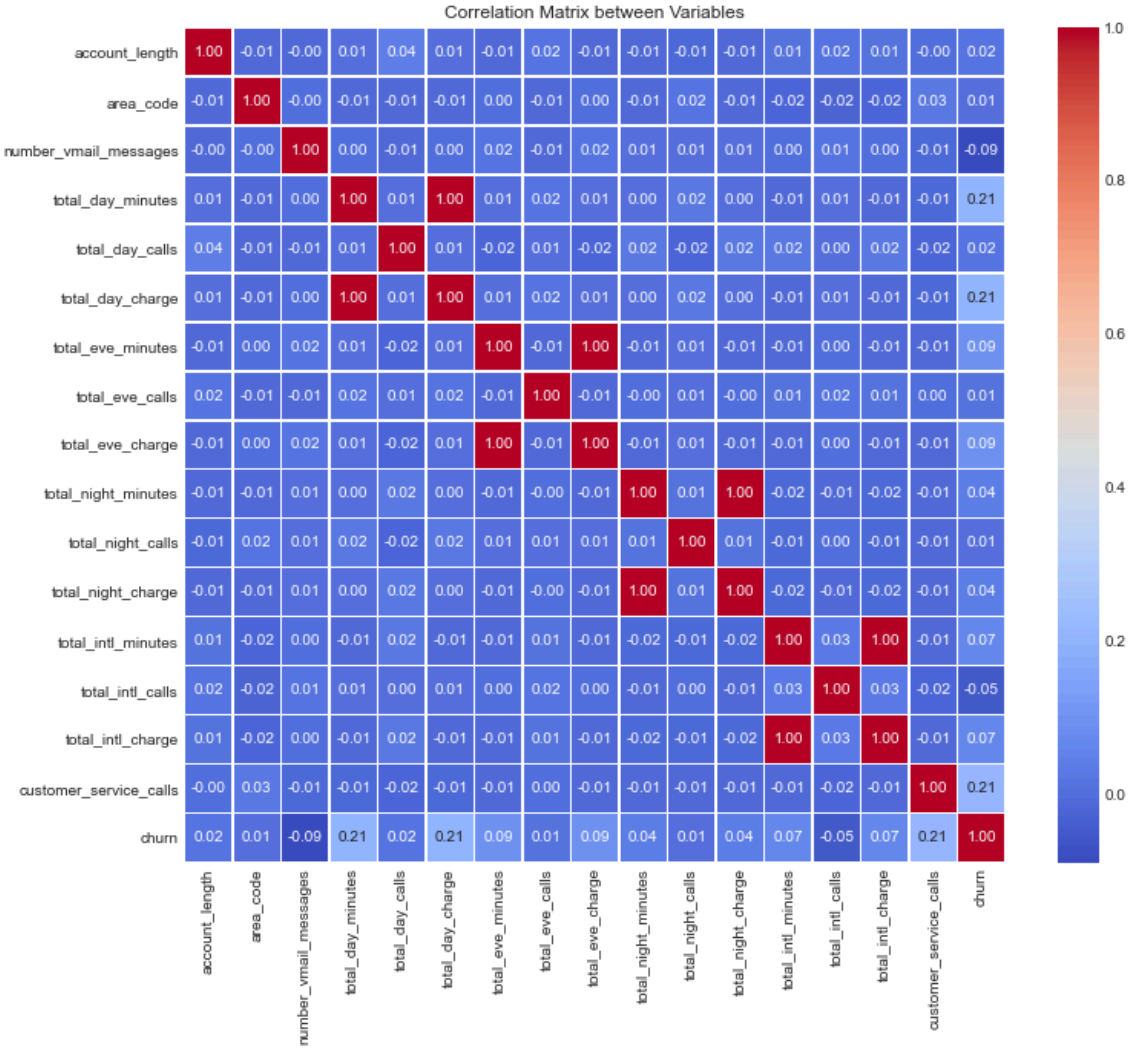
Multivariate analysis explores the interrelationships between multiple variables concurrently. In our research, we examine how different features interact with the target variable (customer churn) to obtain a comprehensive understanding of their combined impact.

To identify correlations between various variables in the dataset, we utilized a correlation matrix. This approach allows us to visualize and quantify the strength of relationships among multiple factors simultaneously.

In [77]:

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

# Calculate the correlation matrix
corr_matrix = df.corr()
```

```
corr_matrix = df.corr()

# Plot the correlation heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0
plt.title('Correlation Matrix between Variables')
plt.show()
```



The correlation matrix indicates a perfect correlation between `total international charge` and `total international minutes`, suggesting multicollinearity. Given their interdependence, it is advisable to utilize only one of these features in the model development process.

Additionally, the analysis reveals that `total minutes`, `total day charge`, and `customer service calls` demonstrate significant correlations with the target variable, warranting particular attention in our modeling approach.

Pre-Data Preprocessing

In this stage we understand that we are dealing with categorical and numeric data columns, some of which must be tranformed into a datatype acceptable by the different machine learning models. We are going to clean, organize, and prepare raw

data for analysis and modeling.

An illustrative example involves the application of one-hot encoding to convert categorical columns with object datatypes into numerical representations. This process typically results in binary values, denoted as 1s and 0s, which enhance the data's compatibility with various analytical models.

We'll split the dataset into training and testing subsets for model development and evaluation. The training set will build models, while the test set will assess their performance. Cross-validation techniques will ensure thorough model evaluation.

1. **Transforming categorical columns to numeric**

In [78]:
```
# Convert 'churn' column to integer type
df['churn'] = df['churn'].astype(int)

# Apply one-hot encoding to categorical columns
df = pd.get_dummies(df, columns=['area_code', 'international_plan', 'voice_ma
```

In [79]:
```
df.head(7)
```

Out[79]:

phone_number	account_length	number_vmail_messages	total_day_minutes	total_day
3824657	128	25	265.1	
3717191	107	26	161.6	
3581921	137	0	243.4	
3759999	84	0	299.4	
3306626	75	0	166.7	
3918027	118	0	223.4	
3559993	121	24	218.2	

7 rows × 23 columns

2. **Isolate the target variable from the feature set**

We proceed to isolate the target variable from the feature set, apply standardization techniques to the features, and implement measures to address the class imbalance present in the target variable.

In [80]:
```
# Separating features (X) from the target variable (y)
```

```
y = df['churn']           # Target variable
X = df.drop('churn', axis=1)  # Features (excluding 'churn')
```

3. **Conduct a Train-test-split**

In [81]:
```
# 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, rand
```

Modelling

In this session, we will create several models, evaluate their performance, and then conduct hyper-parameter tuning to improve them. Our goal is to identify the model and parameters that yield the best results.

We train and evaluate the following models:

- Logistic Regression Model.
- Decision Trees.
- Random Forests.
- K-Nearest Neighbors.

Model 1: Logistic Regression Model

Logistic Regression Model predicts binary outcomes by analyzing relationships between multiple variables. It's commonly used for classification in machine learning, assessing the probability of an instance belonging to a particular category.

In our analysis, we employ logistic regression to model the relationship between our features and the probability of customer churn.

In [84]:
```
# Create a pipeline that includes standardization and logistic regression
model_pipeline = Pipeline(steps=[
    ('preprocessor', ColumnTransformer(
        transformers=[('num', StandardScaler(), X_train.columns)]
    )),
    ('classifier', LogisticRegression())
])

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

# Predict churn for the train and test data
y_train_pred = model_pipeline.predict(X_train)
y_test_pred = model_pipeline.predict(X_test)

# Calculate and print the accuracy scores
print(f"Train Accuracy: {accuracy_score(y_train, y_train_pred):.2f}")
print(f"Test Accuracy: {accuracy_score(y_test, y_test_pred):.2f}")
```

```
# Print the classification report and confusion matrix for test data
print("Classification Report (Test Data):")
print(classification_report(y_test, y_test_pred))
print("Confusion Matrix (Test Data):")
print(confusion_matrix(y_test, y_test_pred))
```

```
Train Accuracy: 0.86
Test Accuracy: 0.86
Classification Report (Test Data):
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       566
           1       0.60      0.18      0.27       101

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

Confusion Matrix (Test Data):
[[554  12]
 [ 83  18]]
```

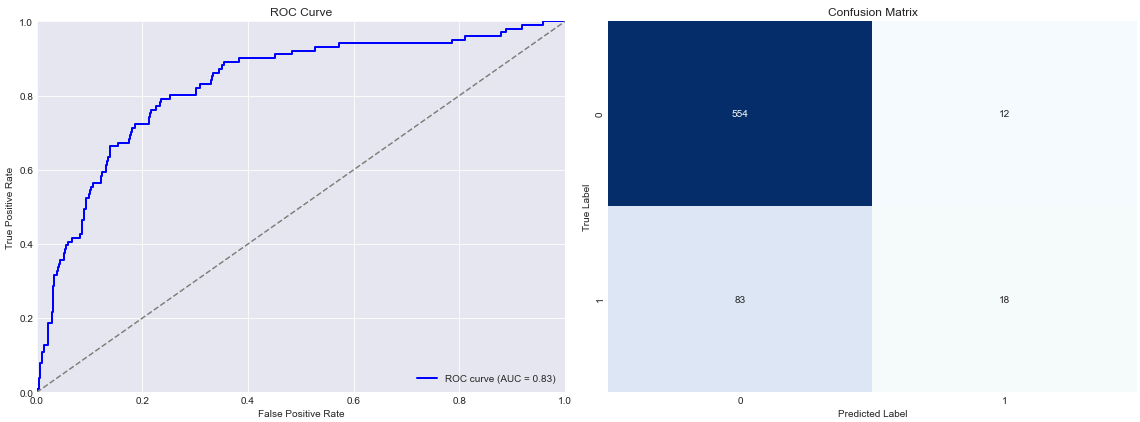
1. **Accuracy:** Our model has similar performance 86% on both the training and test data, which is generally a good sign of generalization.
2. **Model Performance **:The model performs well on non-churn cases (Class 0) with high recall (0.98) and precision (0.87). This means the model is very good at identifying non-churners, but it does a poor job at identifying churners (Class 1).
3. **Precision:** The high recall for Class 0 and low recall for Class 1 suggest that the model tends to predict the majority class more often, which leads to a high accuracy overall but poor performance for the minority class.
4. **F1-Score:** The F1-Score for Class 1 is particularly low (0.27), which is a combined measure of precision and recall. This indicates that improvements are needed for Class 1 prediction.
5. **Class Imbalance:** The poor performance on Class 1 (churn) indicates that our dataset might be imbalanced, with far fewer churn cases compared to non-churn cases. This is reflected in the lower recall (0.18) and precision (0.60) for Class 1. The model is biased towards predicting the majority class (Class 0).

Plot the ROC Curve (Receiver Operating Characteristic curve), the AUC (Area Under the Curve), and Confusion Matrix to visualize the results.

In [85]:

```
# Calculate ROC curve and AUC
y_prob = model_pipeline.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

# Plot ROC curve and confusion matrix
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
```

```
# ROC Curve
axes[0].plot(fpr, tpr, 'b', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
axes[0].plot([0, 1], [0, 1], 'gray', linestyle='--')
axes[0].set_xlim([0.0, 1.0])
axes[0].set_ylim([0.0, 1.0])
axes[0].set_xlabel('False Positive Rate')
axes[0].set_ylabel('True Positive Rate')
axes[0].set_title('ROC Curve')
axes[0].legend(loc="lower right")

# Confusion Matrix
confusion_mat = confusion_matrix(y_test, y_test_pred)
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap='Blues', cbar=False, ax=
axes[1].set_xlabel('Predicted Label')
axes[1].set_ylabel('True Label')
axes[1].set_title('Confusion Matrix')

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



ROC Curve Plot:

The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. The curve starts at (0,0) and ends at (1,1). The closer the ROC curve is to the top-left corner, the better the model's performance and which is the case with our model curve above.

AUC Interpretation: An AUC of 0.83 is generally considered quite good. It means our model has a good trade-off between sensitivity (true positive rate) and specificity (true negative rate).

Confusion Matrix:

The confusion matrix shows how many instances were correctly and incorrectly classified by the model. It is structured as follows:

True Negatives (TN): 554

False Positives (FP): 12

False Negatives (FN): 83

True Positives (TP): 18

The confusion matrix shows that while the model performs well on predicting non-churn cases (high TN, low FP), it struggles with churn cases (low TP, high FN).

Even though the AUC is good, the confusion matrix reveals that the model struggles with predicting churn cases (Class 1). The low number of True Positives and the high number of False Negatives suggest the model is biased towards predicting the majority class (non-churn). For this case we are going to adress the Imbalance in the next cell code

Adrresing Logistic Imbalance

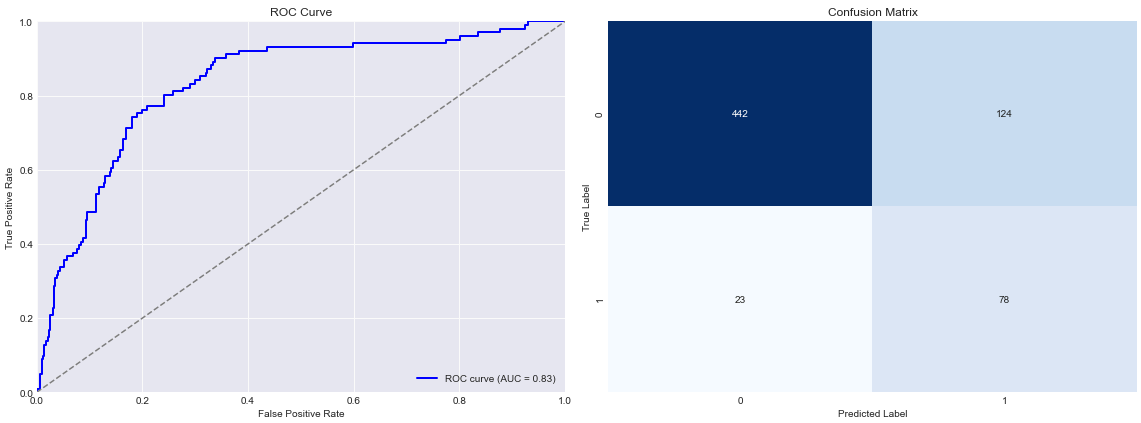
In [87]:
```
 #Define the preprocessing and model pipeline
model_pipeline = Pipeline(steps=[
    ('preprocessor', ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), X.columns)  # Standardize all numerical
        ]
    )),
    ('classifier', LogisticRegression(class_weight='balanced'))  # Initialize
])

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

# Predict on the test data
y_pred = model_pipeline.predict(X_test)

# Print model performance metrics
print(f"Train Accuracy: {model_pipeline.score(X_train, y_train):.2f}")
print(f"Test Accuracy: {model_pipeline.score(X_test, y_test):.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Train Accuracy: 0.77
Test Accuracy: 0.78

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.78      0.86       566
           1       0.39      0.77      0.51       101

    accuracy                           0.78       667
   macro avg       0.67      0.78      0.69       667
weighted avg       0.87      0.78      0.81       667

Confusion Matrix:
[[442 124]
```

```
[ 23  78]]
```

In [88]:

```
# Calculate ROC curve and AUC
y_prob = model_pipeline.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

# Create plots
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# ROC Curve
axes[0].plot(fpr, tpr, 'b', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
axes[0].plot([0, 1], [0, 1], 'gray', linestyle='--')
axes[0].set_xlim([0.0, 1.0])
axes[0].set_ylim([0.0, 1.0])
axes[0].set_xlabel('False Positive Rate')
axes[0].set_ylabel('True Positive Rate')
axes[0].set_title('ROC Curve')
axes[0].legend(loc="lower right")

# Confusion Matrix
confusion_mat = confusion_matrix(y_test, y_pred)
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap='Blues', cbar=False, ax=
axes[1].set_xlabel('Predicted Label')
axes[1].set_ylabel('True Label')
axes[1].set_title('Confusion Matrix')

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



Accuracy: The model's accuracy of 0.77 on training data and 0.78 on test data indicates it performs well overall but with slight overfitting. The performance is consistent across both datasets, which is a good indicator of generalization.

**Precision **: class 0 churn (0.95): The model is very good at identifying non-churn cases when it predicts them while class 1 churn (0.39), the model is less effective at identifying churn cases accurately; it only correctly predicts churn 39% of the time when it predicts it.

**Recall ** :class 0 churn (0.78), The model identifies 78% of actual non-churn cases, whereas class 1 churn (0.77), The model identifies 77% of actual churn cases.

F1-score: The F1-Score for Class 0 (0.86) is high, indicating good performance for non-churn cases. The F1-Score for Class 1 (0.51) is lower, reflecting the model's difficulty in accurately

Confusion Matrix:

True Negatives (442): Correctly predicted non-churn cases.

False Positives (124): Non-churn cases incorrectly predicted as churn.

False Negatives (23): Churn cases incorrectly predicted as non-churn.

True Positives (78): Correctly predicted churn cases.

The confusion matrix shows that the model is quite good at identifying non-churn cases but misses a significant number of churn cases (high False Negatives) and makes some incorrect predictions for non-churn cases (False Positives).

ROC curve (AUC = 0.83): An AUC (Area Under the Curve) value of 0.83 indicates that the model has good discriminative power and is reasonably effective at distinguishing between the two classes.

Conclusion on logistic regression model:

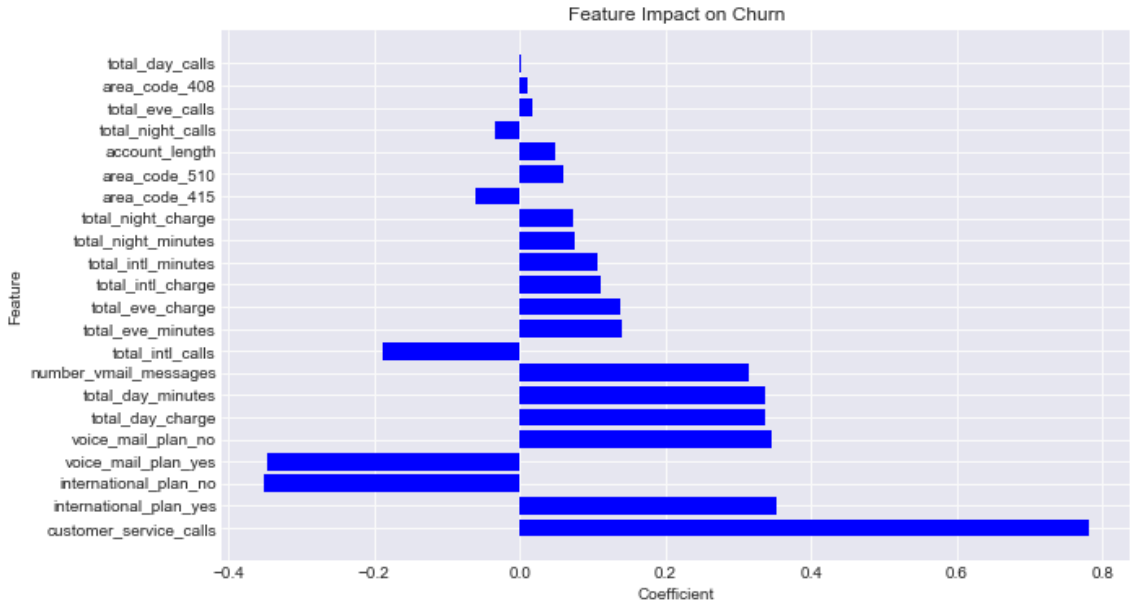
Based on the current results above, the logistic regression model with balanced class weights is a good start but may not be the best fit for your dataset, particularly given its challenges with accurately predicting churn cases

Feature importance in logistic regression model

The coefficients provide insights into the impact of each feature on the likelihood of churn (negative/undesired impact) or not churn (positive/desired impact).

In [90]:

```
# Get the coefficients from the logistic regression model
coefficients = model_pipeline.named_steps['classifier'].coef_[0]

# Create and sort DataFrame of coefficients
coefficients_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': coefficients
}).assign(
    Abs_Coefficient=lambda df: np.abs(df['Coefficient'])
).sort_values(by='Abs_Coefficient', ascending=False)

# Plot the coefficients
plt.figure(figsize=(10, 6))
plt.barh(coefficients_df['Feature'], coefficients_df['Coefficient'], color='b
plt.xlabel('Coefficient')
plt.ylabel('Feature')
plt.title('Feature Impact on Churn')
plt.show()
```



In [91]:

```
# Get the coefficients from the logistic regression model
coefficients = model_pipeline.named_steps['classifier'].coef_[0]

# Create and sort DataFrame of coefficients
coefficients_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': coefficients
}).assign(
    Abs_Coefficient=lambda df: np.abs(df['Coefficient'])
).sort_values(by='Abs_Coefficient', ascending=False)

# Display the sorted coefficients DataFrame
print(coefficients_df)
```

	Feature	Coefficient	Abs_Coefficient
14	customer_service_calls	0.781643	0.781643
19	international_plan_yes	0.351737	0.351737
18	international_plan_no	-0.351737	0.351737
21	voice_mail_plan_yes	-0.346152	0.346152
20	voice_mail_plan_no	0.346152	0.346152
4	total_day_charge	0.337942	0.337942
2	total_day_minutes	0.337680	0.337680
1	number_vmail_messages	0.314547	0.314547
12	total_intl_calls	-0.187955	0.187955
5	total_eve_minutes	0.139153	0.139153
7	total_eve_charge	0.137978	0.137978
13	total_intl_charge	0.110921	0.110921
11	total_intl_minutes	0.106664	0.106664
8	total_night_minutes	0.074404	0.074404
10	total_night_charge	0.072972	0.072972
16	area_code_415	-0.061263	0.061263
17	area_code_510	0.060865	0.060865
0	account_length	0.048723	0.048723
9	total_night_calls	-0.033360	0.033360
6	total_eve_calls	0.017118	0.017118
15	area_code_408	0.009950	0.009950
3	total_day_calls	0.001019	0.001019

High Impact Features: Features like customer_service_calls, international_plan_yes,

and voice_mail_plan_yes have the largest coefficients, indicating they significantly influence churn predictions.

Low Impact Features: Features like account_length, total_night_calls, and total_day_calls have minimal impact, suggesting they are less informative for predicting churn.

To summarize, features with positive coefficients are associated with an increased probability of customer churn, while those with negative coefficients correlate with a reduced likelihood of churn. This insight aids in identifying key factors influencing customer retention.

Model 2: Decision Tree Classifier

Baseline Model

Decision trees are predictive models that use a tree-like structure to make decisions based on input features. They recursively split data, creating a flowchart that leads to final predictions for classification or regression tasks.

We will now proceed to initialize the DecisionTreeClassifier and subsequently train it using our prepared X_train and y_train datasets..

In [92]:
```
# Initialize and train the Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)
```

Model Evaluation

In [93]:
```
# Evaluate the model's performance
y_pred = clf.predict(X_test)  # Ensure predictions are made before evaluation
metrics = {
    'Accuracy': accuracy_score(y_test, y_pred),
    'Precision': precision_score(y_test, y_pred),
    'Recall': recall_score(y_test, y_pred),
    'F1 Score': f1_score(y_test, y_pred)
}

# Print evaluation metrics
for metric, score in metrics.items():
    print(f'{metric}: {score:.2f}')

# Calculate and print train and test scores
print(f'Train Score: {clf.score(X_train, y_train):.2f}')
print(f'Test Score: {clf.score(X_test, y_test):.2f}')
```

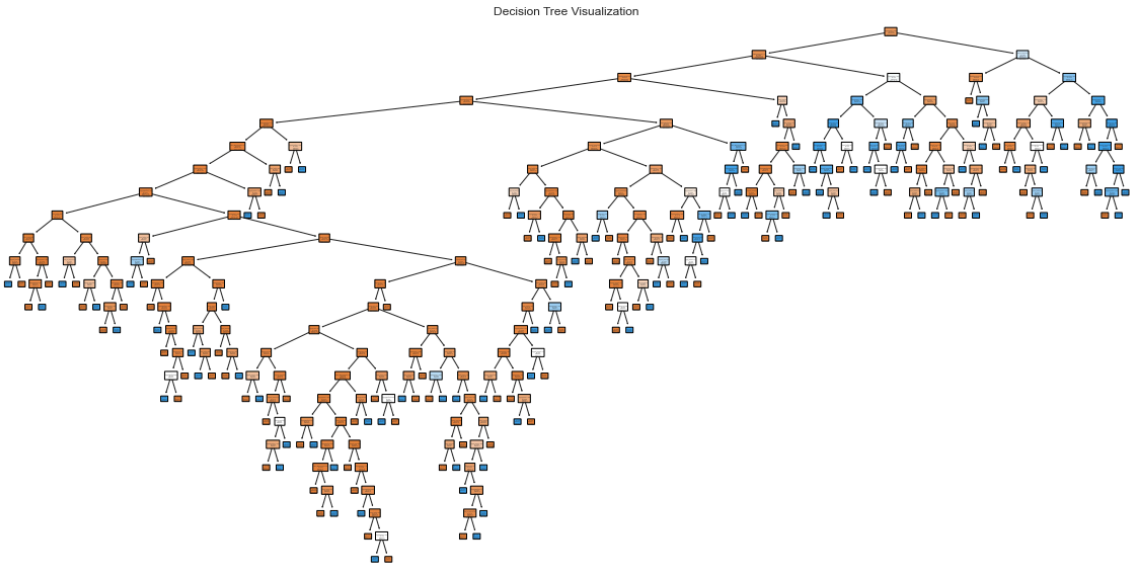
```
Accuracy: 0.92
Precision: 0.72
Recall: 0.75
```

```
F1 Score: 0.73
Train Score: 1.00
Test Score: 0.92
```

Generalization and visualization

In [95]:

```
from sklearn.tree import plot_tree

plt.figure(figsize=(20, 10))
plot_tree(clf, feature_names=X.columns, class_names=['Class 0', 'Class 1'], f
plt.title('Decision Tree Visualization')
plt.show()
```



The above visualizations give a comprehensive view of the model's performance and its decision-making process.

Improving the model using SMOTE

In [96]:

```
# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Train the Decision Tree Classifier on the oversampled data
dt_smote = DecisionTreeClassifier(random_state=42)
dt_smote.fit(X_train_smote, y_train_smote)

# Make predictions on the test set
y_pred_smote = dt_smote.predict(X_test)

# Evaluate the model's performance
accuracy_smote = accuracy_score(y_test, y_pred_smote)
precision_smote = precision_score(y_test, y_pred_smote)
recall_smote = recall_score(y_test, y_pred_smote)
f1_smote = f1_score(y_test, y_pred_smote)

# Print the classification report
print(f'Accuracy: {accuracy_smote:.2f}')
```

```
print(f'Precision: {precision_smote:.2f}')
print(f'Recall: {recall_smote:.2f}')
print(f'F1 Score: {f1_smote:.2f}')
print('Classification Report:')
print(classification_report(y_test, y_pred_smote))
```

```
Accuracy: 0.90
Precision: 0.63
Recall: 0.76
F1 Score: 0.69
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.92      0.94       566
           1       0.63      0.76      0.69       101

    accuracy                           0.90       667
   macro avg       0.79      0.84      0.81       667
weighted avg       0.91      0.90      0.90       667
```

In [97]:

```
# Print the evaluation metrics
print(f"Accuracy: {accuracy_smote:.2f}")
print(f"Precision: {precision_smote:.2f}")
print(f"Recall: {recall_smote:.2f}")
print(f"F1-score: {f1_smote:.2f}")

# Calculate and print train and test scores
print(f'Train Score: {dt_smote.score(X_train_smote, y_train_smote):.2f}')
print(f'Test Score: {dt_smote.score(X_test, y_test):.2f}')
```

```
Accuracy: 0.90
Precision: 0.63
Recall: 0.76
F1-score: 0.69
Train Score: 1.00
Test Score: 0.90
```

Model Performance: The model shows high accuracy (0.90) on test data. However, its lower precision compared to recall suggests it identifies most positive cases but includes some false positives.

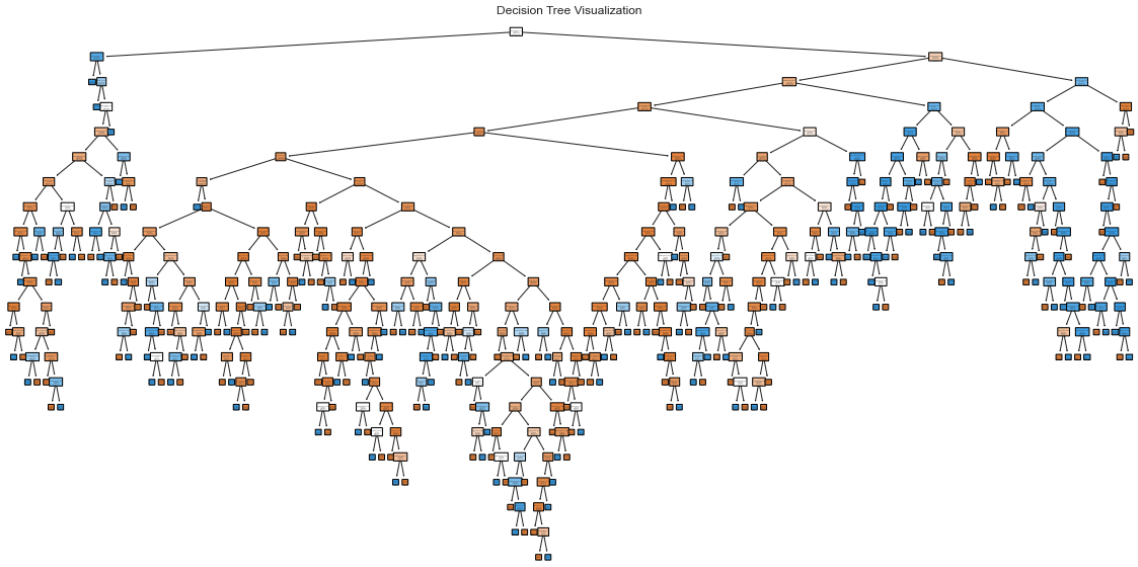
Training vs. Testing: The model achieves perfect accuracy on the training data, potentially indicating overfitting. Nevertheless, its strong performance on the test data suggests reasonable generalization ability.

Generalization and Visualization-SMOTE

In [99]:

```
# Plot the Decision Tree
plt.figure(figsize=(20, 10))
plot_tree(
    dt_smote,
    feature_names=X_test.columns,
    class_names=['0', '1'],
    filled=True,
    rounded=True
)
```

```
plt.title('Decision Tree Visualization')
plt.show()
```



While SMOTE has helped balance the classes, the core performance characteristics of our Decision Tree model remain similar. in this case, the performance metrics (accuracy, precision, recall, F1-score) did not change with SMOTE. Fine-tuning or exploring additional models might provide further improvements.

Improving the model using GridSeachCV

GridSearchCV is a technique used to systematically search for the best combination of hyperparameters for a machine learning model. It improves model performance by testing various parameter configurations and selecting the optimal set based on cross-validated results.

In [110…

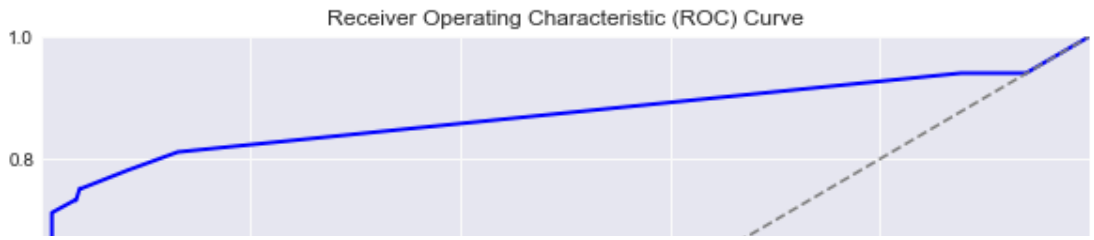
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1
from sklearn.tree import DecisionTreeClassifier

# Define the custom parameter grid
param_grid = {
    'max_depth': [3, 5, 7, 10, 15],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 5, 10]
}

# Initialize the classifier
clf = DecisionTreeClassifier(random_state=42)

# Set up GridSearchCV with the custom parameter grid
grid_search = GridSearchCV(clf, param_grid, scoring='accuracy', cv=5)

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Get the best model from grid search
best_clf = grid_search.best_estimator_
```

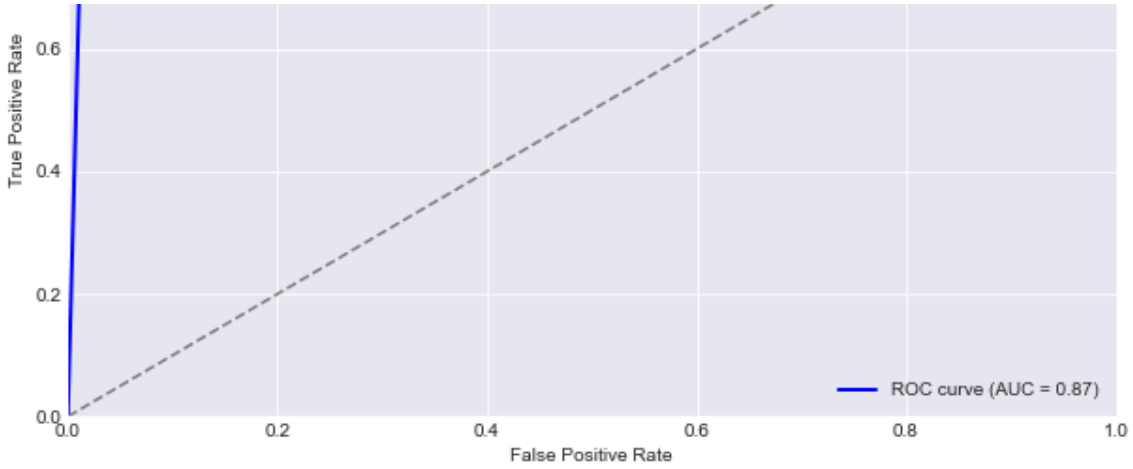
```
# Make predictions on the test set
y_pred = best_clf.predict(X_test)

# Print evaluation metrics
print('Best Parameters:', grid_search.best_params_)
print('Best Cross-Validation Score:', grid_search.best_score_)
print('Test Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred))
print('Recall:', recall_score(y_test, y_pred))
print('F1 Score:', f1_score(y_test, y_pred))

# Print train and test scores
print('Train Score:', best_clf.score(X_train, y_train))
print('Test Score:', best_clf.score(X_test, y_test))
```

```
Best Parameters: {'max_depth': 7, 'min_samples_leaf': 2, 'min_samples_split':
2}
Best Cross-Validation Score: 0.9403608997196281
Test Accuracy: 0.9445277361319341
Precision: 0.9210526315789473
Recall: 0.693069306930693
F1 Score: 0.7909604519774012
Train Score: 0.9643660915228808
Test Score: 0.9445277361319341
```

In [111…

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Fit the model with the best parameters
best_clf = grid_search.best_estimator_

# Predict probabilities for the positive class (1)
y_prob = best_clf.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

# Plot ROC Curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='b', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

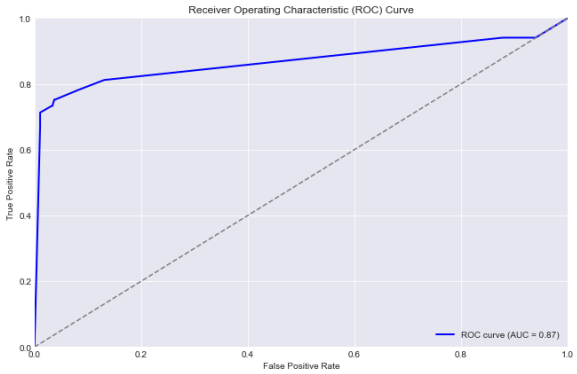


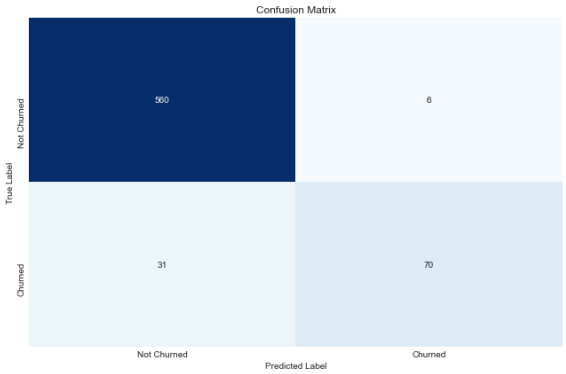


In [116…

```
from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix, Confu
import matplotlib.pyplot as plt
import seaborn as sns

# Fit the model with the best parameters
best_clf = grid_search.best_estimator_

# Predict probabilities for the positive class (1)
y_prob = best_clf.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

# Make predictions for confusion matrix
y_pred = best_clf.predict(X_test)

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Set up the figure with 2 subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

# Plot ROC Curve
ax1.plot(fpr, tpr, color='b', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
ax1.plot([0, 1], [0, 1], color='gray', linestyle='--')
ax1.set_xlim([0.0, 1.0])
ax1.set_ylim([0.0, 1.0])
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('Receiver Operating Characteristic (ROC) Curve')
ax1.legend(loc="lower right")

# Plot Confusion Matrix
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Blues', cbar=False, ax=ax
            xticklabels=['Not Churned', 'Churned'],
            yticklabels=['Not Churned', 'Churned'])
ax2.set_xlabel('Predicted Label')
ax2.set_ylabel('True Label')
ax2.set_title('Confusion Matrix')

# Adjust layout and show plot
plt.tight_layout()
plt.show()
```

```
# Print the confusion matrix values
tn, fp, fn, tp = conf_matrix.ravel()

print("True Negatives (TN):", tn)
print("False Positives (FP):", fp)
print("False Negatives (FN):", fn)
print("True Positives (TP):", tp)
```






```
True Negatives (TN): 560
False Positives (FP): 6
False Negatives (FN): 31
True Positives (TP): 70
```

Model interpretion

Test Accuracy: 94.5%, The high accuracy indicates that the model performs well overall, correctly classifying the majority of test samples.

Precision: 92.1%, This high precision suggests that when the model predicts the positive class (churned), it is correct most of the time. This reduces false positives effectively. the model is very reliable when it predicts that a customer will churn.

Recall: 69.3%, The lower recall indicates that the model misses some actual positive cases (churned). This suggests that while the model is good at avoiding false positives, it could be improved in identifying more positive cases.

F1 Score: 0.79, This score reflects a balance between precision and recall. An F1 score of 0.79 indicates a good balance but highlights room for improvement, especially in recall.

Train 96.4% vs. Test Scores 94.5%: The model generalizes well with a small gap between training and test scores, showing that it is not overfitting and performs consistently on unseen data.

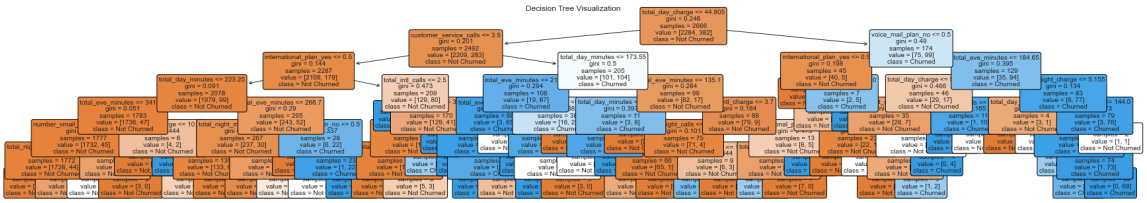
ROC Curve and AUC: An AUC of 0.87 signifies that the model has a strong ability to distinguish between the positive and negative classes. The ROC curve likely shows a good trade-off between sensitivity (true positive rate) and specificity (false positive rate), indicating that the model performs well across different classification thresholds.

Confusion Matrix: True Negatives (TN): The model correctly predicted 560 instances

of "Not Churned." False Positives (FP) on the other hand were only 6 instances where the model incorrectly predicted "Churned" when the true label was "Not Churned." The model is highly accurate in identifying customers who are not likely to churn. The low number of false positives suggests a low rate of false alarms regarding churn.

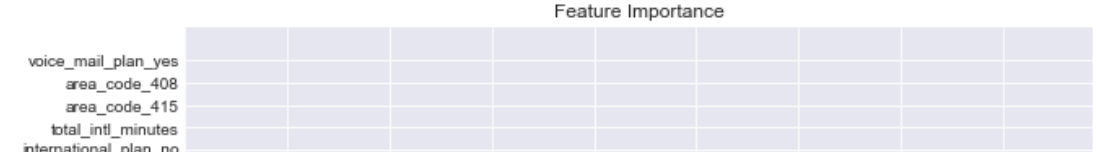
In [124…

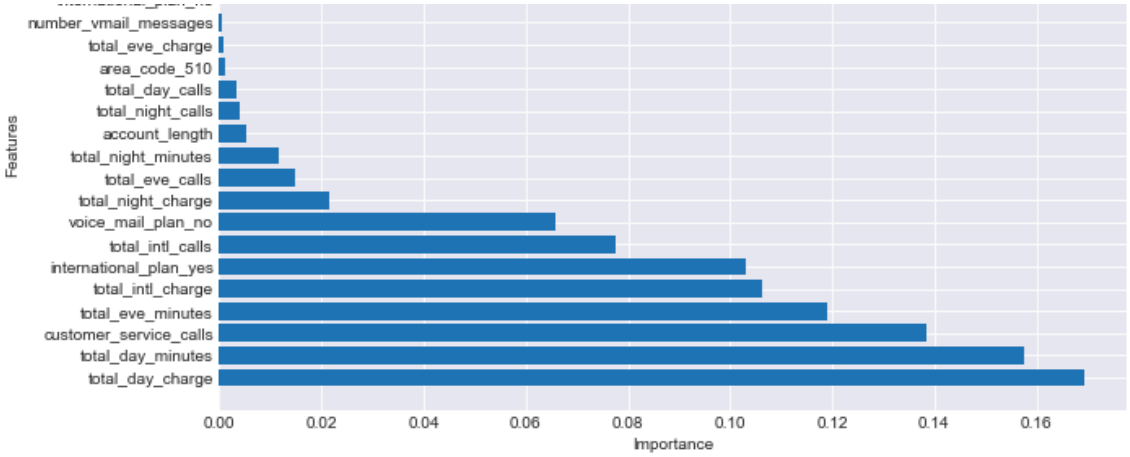
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Plot the decision tree
plt.figure(figsize=(30, 5))
plot_tree(
    best_clf,
    feature_names=X_test.columns,
    class_names=['Not Churned', 'Churned'],
    filled=True,
    rounded=True,
    fontsize=10
)
plt.title('Decision Tree Visualization')
plt.show()
```



In [125…

```
import matplotlib.pyplot as plt

# Get feature importances from the model
importances = clf1.feature_importances_

# Create a DataFrame for feature importances
feature_importance_df = pd.DataFrame({
    'Feature': X_test.columns,
    'Importance': importances
})

# Sort by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', as

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Feature Importance')
plt.show()
```





High Importance Features: Features like total_day_charge, Total_intl_charge are significant predictors of churn, indicating that how much a customer spends on calls during the day is crucial for predicting churn. Customers with higher daytime charges are more likely to churn like wise The total charges for international calls have the second-highest importance. Higher international charges might indicate dissatisfaction or cost-related concerns, leading to a higher likelihood of churn.

Low Importance Features: Features like total_night_calls and area_code_415 have low influence, suggesting they might not be as valuable in predicting churn. You might consider removing them to simplify the model.

Understanding feature importance aids in model refinement and focuses attention on the most influential variables. It guides feature engineering by identifying which attributes merit further development and which may be eliminated or modified.

Decision tree model conclusion

The Decision Tree model is strong and performs well based on the findings above. However, considering potential improvements in recall and avoiding overfitting, exploring ensemble methods

Model 4: RandomForestClassifier

Random Forests is a machine learning algorithm that combines multiple decision trees to create a more robust and accurate model. It improves upon individual decision trees by reducing overfitting and increasing generalization through ensemble learning.

In [126…

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1
from sklearn.preprocessing import StandardScaler

# Initialize and apply StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize and train Random Forest classifier
```

```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train_scaled, y_train)

# Make predictions
y_pred = rf.predict(X_test_scaled)

# Evaluate performance
accuracy_rf = accuracy_score(y_test, y_pred)
precision_rf = precision_score(y_test, y_pred)
recall_rf = recall_score(y_test, y_pred)
f1_rf = f1_score(y_test, y_pred)

# Print metrics
print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1-score:", f1_rf)

# Print train and test scores
print("Train score:", rf.score(X_train_scaled, y_train))
print("Test score:", rf.score(X_test_scaled, y_test))
```

```
Accuracy: 0.9445277361319341
Precision: 0.9210526315789473
Recall: 0.693069306930693
F1-score: 0.7909604519774012
Train score: 1.0
Test score: 0.9445277361319341
```

Accuracy: The model correctly classifies approximately 94.5% of the test data. This is a high accuracy, indicating that the model performs well overall in predicting both churned and not churned cases.

Precision: When the model predicts a positive class (churned), it is correct about 92.1% of the time. High precision means that the model has a low false positive rate, which is crucial when false positives are costly.

Recall: The model identifies about 69.3% of the actual positive cases (churned). While this recall is decent, it indicates that the model misses about 30.7% of actual churned cases, which could be improved

F1-Score: The F1-score is the harmonic mean of precision and recall. A score of 0.7910 shows a good balance between precision and recall, though there is room for improvement, particularly in recall.

Train Score: The model achieves 100% accuracy on the training data, indicating it perfectly fits the training data. This suggests potential overfitting, where the model performs exceptionally well on training data but may not generalize as well to new data.

Test Score: The model's performance on the test data is consistent with the accuracy reported earlier (94.5%), showing that it generalizes well to unseen data.

General Insights: High Accuracy and Test Score: The Random Forest model performs well overall, with high accuracy on both training and test datasets, indicating good generalization.

Precision vs. Recall Trade-Off: While precision is high, recall is lower. This trade-off suggests the model is better at avoiding false positives but could be improved in identifying more actual churn cases.

Potential Overfitting: The perfect training score (1.0) suggests the model may be overfitting the training data. This is common with complex models and may be mitigated by techniques such as cross-validation, pruning, or using a simpler model.

The Random Forest model seems to have performed well in predicting churn and not churned customers. It achieved high accuracy, precision, and recall scores, indicating that it is effective in identifying churned customers while minimizing false positives.

Addressing overfiiting - Using k-fold

In [127…

```
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler

# Initialize the Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Initialize the Standard Scaler
scaler = StandardScaler()

# Scale the features
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Perform k-fold cross-validation
cv_scores = cross_val_score(rf, X_train_scaled, y_train, cv=5, scoring='accur

# Train the model
rf.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = rf.predict(X_test_scaled)

# Evaluate the model
accuracy_rf = np.mean(cv_scores)
precision_rf = precision_score(y_test, y_pred)
recall_rf = recall_score(y_test, y_pred)
f1_rf = f1_score(y_test, y_pred)

# Print the results
print("Cross-Validation Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1-score:", f1_rf)
```

```
# Print train and test scores
train_score = rf.score(X_train_scaled, y_train)
test_score = rf.score(X_test_scaled, y_test)

print("Train score:", train_score)
print("Test score:", test_score)
```

```
Cross-Validation Accuracy: 0.9523655936645797
Precision: 0.9210526315789473
Recall: 0.693069306930693
F1-score: 0.7909604519774012
Train score: 1.0
Test score: 0.9445277361319341
```

Random Forest classifier with reduced n_estimators and limited max_depth

In [128…

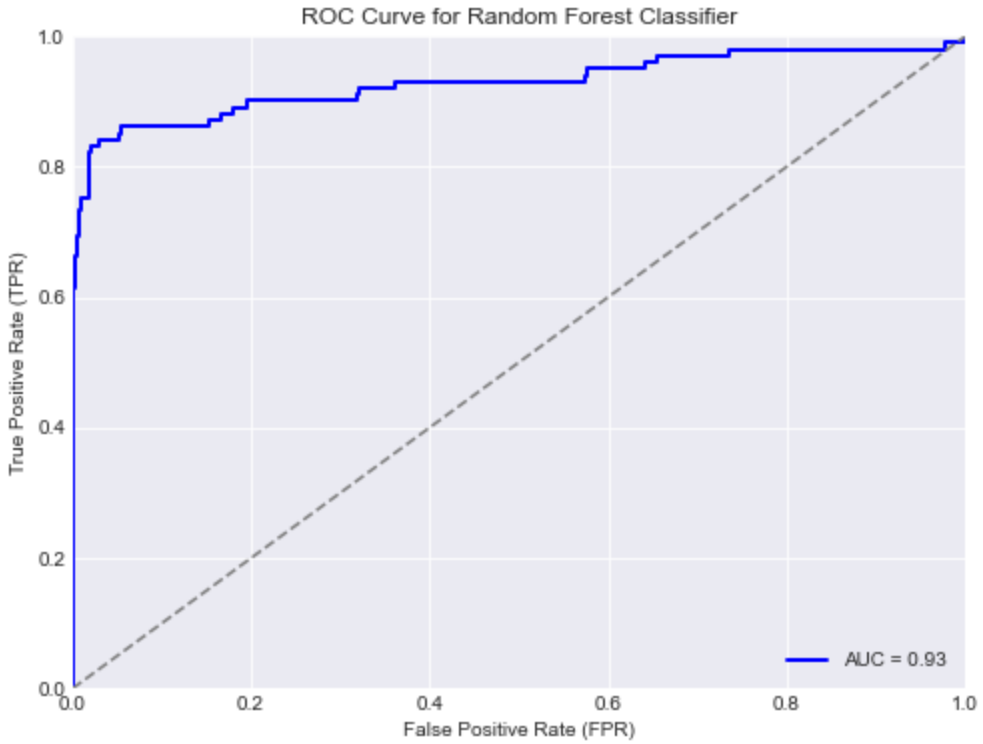
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler

# Initialize the Random Forest classifier
rf = RandomForestClassifier(n_estimators=50, max_depth=10, random_state=42)

# Initialize the scaler
scaler = StandardScaler()

# Scale the features
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Perform 5-fold cross-validation
cv_scores = cross_val_score(rf, X_train_scaled, y_train, cv=5, scoring='accur

# Train the model
rf.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = rf.predict(X_test_scaled)

# Evaluate the model
accuracy_rf = np.mean(cv_scores)
precision_rf = precision_score(y_test, y_pred)
recall_rf = recall_score(y_test, y_pred)
f1_rf = f1_score(y_test, y_pred)

# Print evaluation metrics
print("Cross-Validation Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1-score:", f1_rf)

# Print train and test scores
print("Train score:", rf.score(X_train_scaled, y_train))
print("Test score:", rf.score(X_test_scaled, y_test))
```

```
Cross-validation Accuracy: 0.9437366053041072
Precision: 0.958904109589041
Recall: 0.693069306930693
F1-score: 0.8045977011494252
Train score: 0.977119279819955
Test score: 0.9490254872563718
```

Plot the ROC , AUC curve

In [129…

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Get probability estimates for class 1 (positive class)
y_prob = rf.predict_proba(X_test_scaled)[:, 1]

# Calculate the false positive rate (FPR), true positive rate (TPR), and thre
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Calculate the area under the ROC curve (AUC)
roc_auc = roc_auc_score(y_test, y_prob)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='b', lw=2, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Random Forest Classifier')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



Cross-Validation Accuracy: A score of 0.944 is quite high, suggesting that the model has strong predictive performance and generalizes well on the training data.

Precision: A precision of 0.959 means that when the model predicts a positive outcome, it is correct 95.9% of the time. This is very high, indicating that the model has a strong ability to avoid false positives.

Recall: A recall of 0.693 indicates that the model correctly identifies 69.3% of the true positive cases. While this is good, it is lower than precision, which suggests that the model misses some positive cases (false negatives).

F1-score:A score of 0.805 indicates a strong balance between precision and recall. This reflects a good trade-off between catching most of the positive cases and minimizing false positives.

Train Score: The training accuracy of 0.977 indicates that the model performs exceptionally well on the training data, correctly classifying 97.7% of the samples. This high score suggests the model has learned the training data very well.

Test Score: The test accuracy of 0.949 indicates that the model performs very well on unseen test data, classifying 94.9% of the test samples correctly. This high score suggests that the model generalizes well to new data.

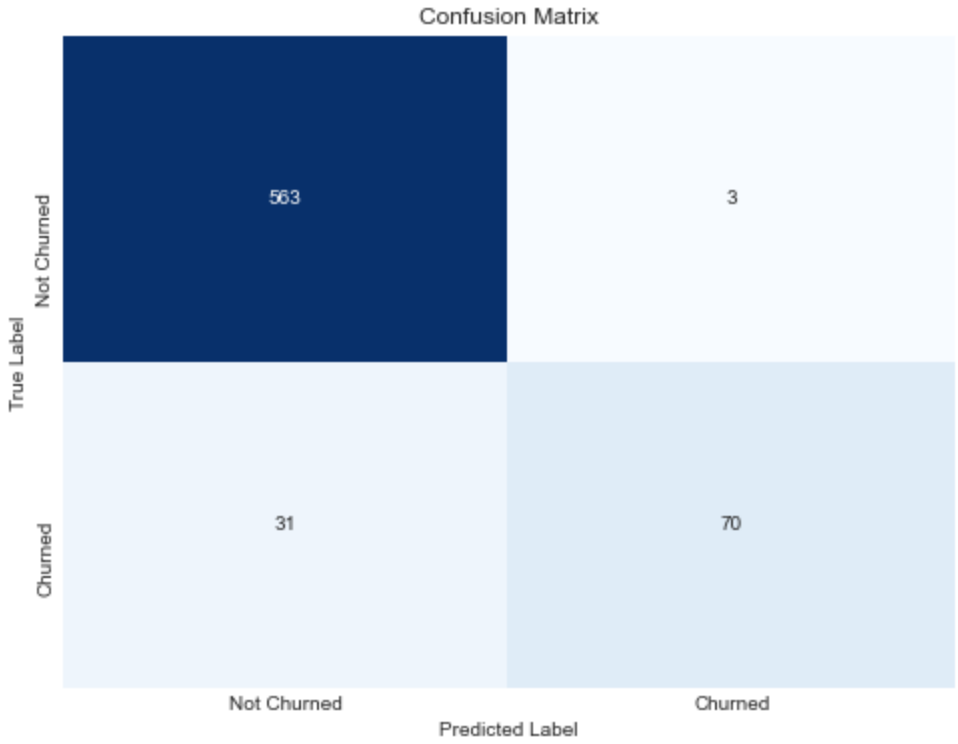
AUC (Area Under the Curve): The AUC of 0.93 for the ROC curve indicates excellent model performance in distinguishing between the positive and negative classes. An AUC close to 1.0 suggests that the model has a high capability to separate the classes.

Overall Performance: The RandomForestClassifier shows very strong performance with high accuracy, precision, and F1-score. The high cross-validation accuracy and test accuracy further confirm that the model generalizes well and is not overfitting the training data.

Confusion Matrix

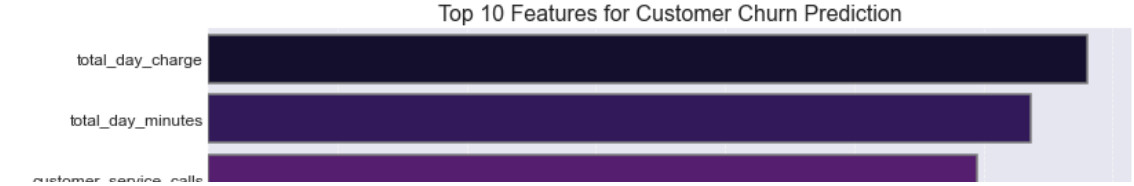
In [130…

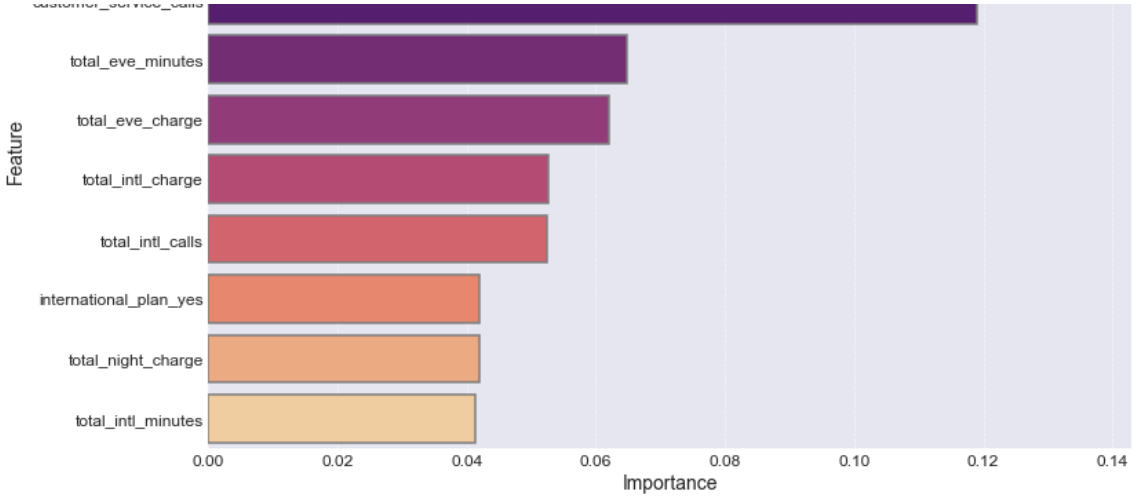
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap for visualization
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Not Churned', 'Churned'],
            yticklabels=['Not Churned', 'Churned'])

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



In [134…

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the Random Forest classifier
rf = RandomForestClassifier(n_estimators=50, random_state=42)
rf.fit(X_train_scaled, y_train)

# Get and sort feature importances
importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': rf.feature_importances_
}).sort_values(by='Importance', ascending=False)

# Plot the top N most important features with a refined color scheme
top_n = 10  # Number of top features to display
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=importance_df.head(top_n),
            palette='magma', edgecolor='grey', linewidth=1.5)
plt.title(f'Top {top_n} Features for Customer Churn Prediction', fontsize=16)
plt.xlabel('Importance', fontsize=14)
plt.ylabel('Feature', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.show()
```





In [135…

```
importance_df
```

Out[135…

	Feature	Importance
4	total_day_charge	0.136149
2	total_day_minutes	0.127265
14	customer_service_calls	0.119014
5	total_eve_minutes	0.064852
7	total_eve_charge	0.062002
13	total_intl_charge	0.052610
12	total_intl_calls	0.052502
19	international_plan_yes	0.042068
10	total_night_charge	0.041968
11	total_intl_minutes	0.041368
18	international_plan_no	0.040335
8	total_night_minutes	0.037998
3	total_day_calls	0.034001
0	account_length	0.029322
9	total_night_calls	0.028145
6	total_eve_calls	0.027829
1	number_vmail_messages	0.019718
21	voice_mail_plan_yes	0.014637
20	voice_mail_plan_no	0.013694
16	area_code_415	0.005654
17	area_code_510	0.005131

15	area_code_408	0.003739

THE BEST FIT MODEL

Based on above 3 models we've looked into, lets identify the best fit by ploting ROC, AUC for all the three

In [144…

```
import pandas as pd

# Create the comparison DataFrame
comparison_frame = pd.DataFrame({
    'Model': ['Logistic Regression', 'Decision Trees Classifier', 'Random For
    'Accuracy (Test Set)': [0.78, 0.95, 0.94],
    'F1 Score (Test Set)': [0.51, 0.81, 0.80],
    'Recall (Test Set)': [0.77, 0.73, 0.69],
    'Precision (Test Set)': [0.39, 0.90, 0.96]
})

# Highlight the maximum values in each column
comparison_frame.style.highlight_max(color='lightgreen', axis=0)
```

Out[144…

	Model	Accuracy (Test Set)	F1 Score (Test Set)	Recall (Test Set)	Precision (Test Set)
0	Logistic Regression	0.780000	0.510000	0.770000	0.390000
1	Decision Trees Classifier	0.950000	0.810000	0.730000	0.900000
2	Random Forest Classifier	0.940000	0.800000	0.690000	0.960000

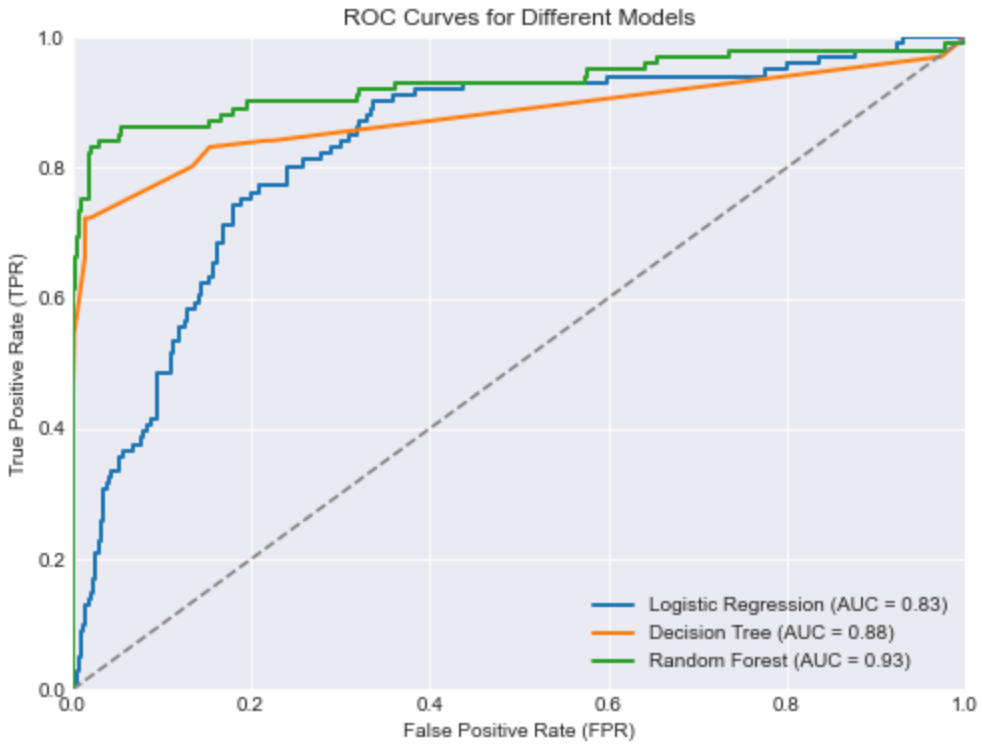
Logististic Regression, Desicion tree and Random Forest Model ROC, AUC

In [159…

```
# Initialize classifiers
classifiers = {
    'Logistic Regression': LogisticRegression(class_weight='balanced'),
    'Decision Tree': DecisionTreeClassifier(max_depth=5, random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=50, max_depth=10, ra
}

# Initialize the scaler and fit-transform the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

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

for name, clf in classifiers.items():
    clf.fit(X_train_scaled, y_train)
    y_prob = clf.predict_proba(X_test_scaled)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
```

```
    roc_auc = roc_auc_score(y_test, y_prob)
    plt.plot(fpr, tpr, lw=2, label=f'{name} (AUC = {roc_auc:.2f})')

 # Print AUC results for each classifier
    print(f'{name} - AUC: {roc_auc:.2f}')

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

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curves for Different Models')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

```
Logistic Regression - AUC: 0.83
Decision Tree - AUC: 0.88
Random Forest - AUC: 0.93
```




```
Random Forest - AUC: 0.93
```

Summary for Customer Churn Prediction

- **Best Model**: The **Random Forest** algorithm demonstrates optimal performance in predicting customer churn. With an impressive AUC of 0.93, it exhibits exceptional accuracy in distinguishing between churned and retained customers, providing a reliable tool for identifying potential churners.

- **Intermediate Model**: The **Decision Tree** model surpasses Logistic Regression, achieving an AUC of 0.88. It effectively captures intricate patterns in customer behavior but requires meticulous fine-tuning to mitigate overfitting risks

behavior but requires meticulous fine-tuning to mitigate overfitting risks.

- **Baseline Model**: While **Logistic Regression** yields satisfactory results, it falls