BUSINESS UNDERSTANDING

Objective

SyriaTel seeks to identify the clients who are most likely to discontinue using its service. The business can take proactive measures, including improving service, resolving problems, or giving exclusive deals, to keep these clients happy and prevent loss of customers by understanding their needs by making predictions if a customer will stay or move

Why is it important?

The dataset provides key indicators related to customer behavior, which help explain why predicting churn (losing customers) matters:

1. Usage Patterns & Charges:

Customers with low usage (e.g., total minutes, calls, and charges) may not be fully engaged and could be at risk of leaving. High international charges without an international plan might lead to dissatisfaction due to high costs.

2. Customer Complaints:

Customer service calls could indicate frustration. A high number of calls may mean a customer is unhappy and considering leaving.

3. Plans & Features:

Customers without a voice mail plan or international plan might be looking for better alternatives elsewhere. A short account length might mean new customers leave quickly if they’re unhappy.

4. Retention Strategies:

By predicting who is likely to churn, SyriaTel can offer discounts, better plans, or improved customer service before customers decide to leave.

Key Business Questions

1. What is the overall churn rate?
2. Which factors correlate with customer churn?
3. Are customers with specific service plans more likely to churn?
4. How does customer behavior, like usage and charges, impact churn?
5. How does customer service interaction affect churn?
6. Do customers in certain regions or area codes have higher churn rates?

Success criteria

1. Achieve a measurable churn rate that can be tracked over time to monitor improvements.
2. Identify the top 3 to 5 key factors (such as customer service calls, usage patterns, service plans) that strongly correlate with churn.
3. Clearly identify whether customers with certain plans (e.g., international plan, voicemail plan) have a higher churn rate.
4. Establish a direct correlation between usage behavior (e.g., heavy daytime or international calls) and churn.
5. Determine the impact of customer service call frequency on churn.
6. Identify geographic regions or area codes with significantly higher churn.

Data Understanding

Overview

SyriaTel wants to reduce customer churn by identifying key factors that influence whether a customer leaves the service. The dataset includes customer demographics, service usage details, and customer support interactions. Below is a breakdown of how different features in the dataset help answer key business questions related to churn.

1. What is the overall churn rate? To determine the churn rate, we analyze the churn column, where:

True represents customers who have churned. False represents customers who have remained. By calculating the percentage of customers with churn = True, we can determine the overall churn rate, which helps SyriaTel measure how many customers are leaving.

Relevant column(s):

churn 2. Which factors correlate with customer churn? To understand which factors influence churn, we examine the relationship between churn and different numerical and categorical variables. Features like total minutes used, number of calls, total charges, and customer service calls could play a role in predicting churn. Correlation analysis and visualizations will help us identify significant patterns.

Relevant column(s):

account_length (How long a customer has been with SyriaTel) total_day_minutes, total_eve_minutes, total_night_minutes, total_intl_minutes (Total call usage at different times) total_day_charge, total_eve_charge, total_night_charge, total_intl_charge (Total charges for calls) customer_service_calls (Number of times a customer has contacted support) 3. Are customers with specific service plans more likely to churn? SyriaTel offers optional plans, such as an International Plan and a Voicemail Plan. Customers subscribed to these plans may have different churn rates. By comparing churn rates between customers with and without these plans, we can determine whether they influence customer retention.

Relevant column(s):

international_plan (Indicates if a customer has an international calling plan: "yes" or "no") voice_mail_plan (Indicates if a customer has a voicemail plan: "yes" or "no") 4. How does customer behavior, like usage and charges, impact churn? Customer behavior can indicate churn risk. Customers with high usage and charges may feel the service is expensive and switch to competitors, while very low usage may indicate dissatisfaction. By analyzing total call minutes, total charges, and the number of calls made, we can determine if certain usage patterns contribute to churn.

Relevant column(s):

total_day_minutes, total_eve_minutes, total_night_minutes, total_intl_minutes (Minutes used at different times of the day) total_day_calls, total_eve_calls, total_night_calls, total_intl_calls (Number of calls made at different times) total_day_charge, total_eve_charge, total_night_charge, total_intl_charge (Charges incurred for calls) 5. How does customer service interaction affect churn? Frequent customer service interactions may indicate dissatisfaction. Customers who contact customer support multiple times may experience unresolved issues, leading to higher churn rates. By analyzing the number of customer service calls, we can assess whether higher contact rates correlate with increased churn.

Relevant column(s):

customer_service_calls (Number of calls made to customer service) 6. Do customers in certain regions or area codes have higher churn rates? Geographic factors may impact churn. Customers in certain regions might experience poor network coverage, pricing differences, or competition from other providers. By analyzing churn rates across different states and area codes, we can identify regions with higher customer loss.

Relevant column(s):

state (Customer's state, useful for regional churn analysis) area_code (Indicates the area code, which can help group customers by region) Conclusion This dataset provides valuable insights into customer behavior, service usage, and interactions with SyriaTel. By analyzing these variables through Exploratory Data Analysis (EDA), we can uncover patterns that help predict churn and develop strategies for customer retention. Understanding how service plans, customer service interactions, usage behavior, and geographic factors contribute to churn will enable SyriaTel to make data-driven decisions and improve customer satisfaction. customer_service_calls: The number of times a customer calls customer service. Key Uses:

Q5: Does the number of customer service calls predict the likelihood of churn? Data Collection Process To make informed decisions about retaining customers, we need to gather relevant data from the following sources:

SyriaTel Service Data:

This dataset will provide basic information about customer usage, service plans, and churn status. We will extract and analyze data like call minutes, service plan types, and customer churn. Customer Demographics:

This data will include customer information such as region, area code, and any available demographic data that may influence churn. Customer Service Interactions:

We will focus on data related to customer service calls, which could be an indicator of potential churn. Combining the Datasets After collecting data from SyriaTel's customer service records, usage patterns, and demographic data, we will combine these datasets into one unified dataset by merging them on customer_id.

Key Steps in Data Combination: Join Datasets: Merge the datasets on customer_id (primary identifier). SyriaTel Service Data with Customer Service Data using customer_id. Customer Demographics with the combined service data using customer_id. Once the datasets are combined, we will have a comprehensive dataset that includes:

Customer demographic data (e.g., area code). Usage data (e.g., total day minutes, customer service calls). Churn status (the target variable). Data Understanding Summary In this phase, we aim to understand the structure and content of the datasets to ensure we can answer the business questions. Specifically, we will:

Examine usage patterns (e.g., total minutes used, call frequency) to uncover which patterns are indicative of churn. Analyze the impact of service plans (e.g., international and voicemail plans) on customer retention. Explore the relationship between customer service interactions (e.g., number of calls to customer service) and churn. Investigate the influence of demographic factors (e.g., area code) on churn rates. By combining the datasets, we will

Data Collection

The dataset used in this analysis is a single structured dataset containing customer information, service usage statistics, and customer support interactions. It does not require merging multiple datasets.

2. Data Features The dataset includes the following types of data:

A. Customer Demographics & Account Information state – The state where the customer is located. area_code – The customer's area code, indicating the geographical region. phone_number – A unique identifier for each customer (not used for analysis). account_length – The number of days the customer has been with SyriaTel. B. Service Plans & Subscriptions international_plan – Indicates if the customer has an international calling plan (yes or no). voice_mail_plan – Indicates if the customer has a voicemail plan (yes or no). number_vmail_messages – Number of voicemail messages received. C. Call Usage & Charges For each time period (Day, Evening, Night, International):

total_day_minutes, total_eve_minutes, total_night_minutes, total_intl_minutes – Total call minutes used. total_day_calls, total_eve_calls, total_night_calls, total_intl_calls – Total number of calls made. total_day_charge, total_eve_charge, total_night_charge, total_intl_charge – Total charges incurred for calls. D. Customer Service Interactions customer_service_calls – Number of times the customer has contacted customer support.

Importing libraries

In [9]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classificati
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import time
import warnings
warnings.filterwarnings('ignore')
```

Data Cleaning

In [10]:
```
df=pd.read_csv('Syria Tel.csv')
df.head()
```

Out[10]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...

5 rows × 21 columns

In [11]:
```
df.tail()
```

Out[11]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85

5 rows × 21 columns

In [12]:
```
# displaying basic information
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 [13]:
```
# Select numeric columns
numeric_columns = df.select_dtypes(include='number').columns
numeric_columns
# Convert numeric columns to numeric data types, coercing errors to NaN
df[numeric_columns] = df[numeric_columns].apply(pd.to_numeric, errors='coe
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 [14]:
```
# Select categorical columns
categorical_columns = df.select_dtypes(include=['object', 'category']).colu
categorical_columns
# Convert categorical columns to 'object' dtype
df[categorical_columns] = df[categorical_columns].astype('object')
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 [15]:
```
# displaying statistics of numerical columns
df.describe()
```

Out[15]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	tc n
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.

In [16]:
```
#displaying statistics of categorcal colums
df.describe(include='O')
```

Out[16]:

	state	phone number	international plan	voice mail plan
count	3333	3333	3333	3333
unique	51	3333	2	2
top	WV	382-4657	no	no
freq	106	1	3010	2411

In [17]:
```
# Convert all boolean columns to object (string)
df = df.astype({col: "object" for col in df.select_dtypes(include=["bool"])
```

In [18]:
```
# check if churn has been converted to boolean
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   object 
dtypes: float64(8), int64(8), object(5)
memory usage: 546.9+ KB
```

In [19]:
```
#checking for missing values
df.isnull().sum()
```

Out[19]:
```
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
```

In [20]:
```
# checking for duplicates
df.duplicated().sum()
```

Out[20]:
```
0
```

In [21]:
```
# dropping of irrelevant column
df=df.drop('phone number', axis=1)
```

In [22]:
```
# checking for columns
df.columns
```

Out[22]:
```
Index(['state', 'account length', 'area code', 'international plan',
       'voice mail plan', 'number vmail messages', 'total day minutes',
       'total day calls', 'total day charge', 'total eve minutes',
       'total eve calls', 'total eve charge', 'total night minutes',
       'total night calls', 'total night charge', 'total intl minutes',
       'total intl calls', 'total intl charge', 'customer service calls',
       'churn'],
      dtype='object')
```

In [23]:
```
# replacing wide space with (_)
df.columns = df.columns.str.replace(' ', '_')
df.columns
```

Out[23]:
```
Index(['state', 'account_length', 'area_code', 'international_plan',
       'voice_mail_plan', 'number_vmail_messages', 'total_day_minutes',
       'total_day_calls', 'total_day_charge', 'total_eve_minutes',
       'total_eve_calls', 'total_eve_charge', 'total_night_minutes',
       'total_night_calls', 'total_night_charge', 'total_intl_minutes',
       'total_intl_calls', 'total_intl_charge', 'customer_service_calls',
       'churn'],
      dtype='object')
```

In [24]:
```
# Identify columns with unique values
unique_value_columns = [col for col in df.columns if df[col].nunique() == 

# Drop columns with unique values
df = df.drop(columns=unique_value_columns)

# Print the columns that were removed
if unique_value_columns:
    print("Removed columns with unique values:", unique_value_columns)
else:
    print("No columns with unique values were found.")
```

```
No columns with unique values were found.
```

In [25]:
```
# Identify columns with zero variance (constant columns)
zero_variance_columns = [col for col in df.columns if df[col].nunique() ==

# Drop columns with zero variance
df = df.drop(columns=zero_variance_columns)

# Print the columns that were removed
if zero_variance_columns:
    print("Removed columns with zero variance:", zero_variance_columns)
else:
    print("No columns with zero variance were found.")
```

```
No columns with zero variance were found.
```

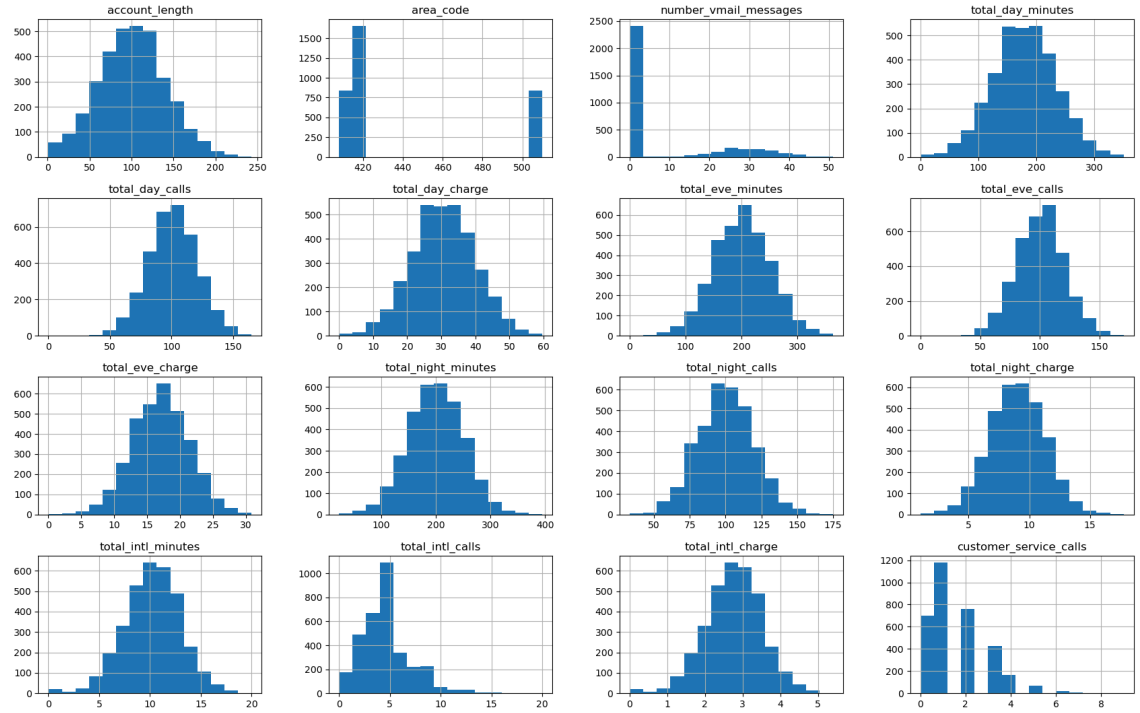
In [26]:
```
# Total rows
total_records = len(df)

# Missing values
missing_values = df.isnull().sum()

# Percent missing values
percent_missing_values = (missing_values/ total_records) * 100

# Use a DataFrame
missing_values_df = pd.DataFrame({
    'missing_values': missing_values,
    'percentage (%)': percent_missing_values
})

print(f"Total records: {total_records}")
# Slice to only the missing values
missing_values_df.iloc[ :]
```

```
Total records: 3333
```

Out[26]:

	missing_values	percentage (%)
state	0	0.0
account_length	0	0.0
area_code	0	0.0
international_plan	0	0.0
voice_mail_plan	0	0.0
number_vmail_messages	0	0.0
total_day_minutes	0	0.0
total_day_calls	0	0.0
total_day_charge	0	0.0
total_eve_minutes	0	0.0
total_eve_calls	0	0.0
total_eve_charge	0	0.0
total_night_minutes	0	0.0
total_night_calls	0	0.0
total_night_charge	0	0.0
total_intl_minutes	0	0.0
total_intl_calls	0	0.0
total_intl_charge	0	0.0
customer_service_calls	0	0.0
churn	0	0.0

In [27]:
```
df.hist(bins=15, figsize=(21,13)); # distribution of columns
```



In [28]:
```
# checking  how many observations and variables are in the datase
print(f'The number of observations are {df.shape[0]} and variables are {df
```

```
The number of observations are 3333 and variables are 20
```

In [200]:
```
from ydata_profiling import ProfileReport
```

In [201]:
```
# Generate a profile report
profile = ProfileReport(df, title="Churn Data Profile Report", explorative=

# Display the profile report in the notebook
profile.to_notebook_iframe()
```

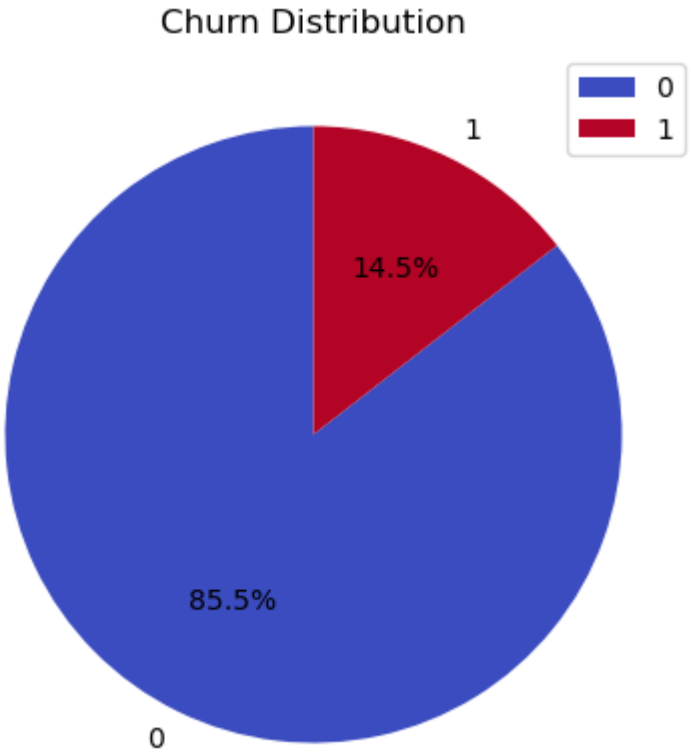
```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Explanatory Data Analysis

Univariate Analysis

Churn Distribution

In [150]:
```
# Count occurrences of each category in 'booking_status'
booking = df['churn'].value_counts()

# Plot as a pie chart
plt.figure(figsize=(5, 5))  # Set figure size
booking.plot(kind='pie', autopct='%1.1f%%', startangle=90, cmap='coolwarm'

# Customize labels and title
plt.ylabel('')  # Remove y-axis label for clarity
plt.title('Churn Distribution')

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

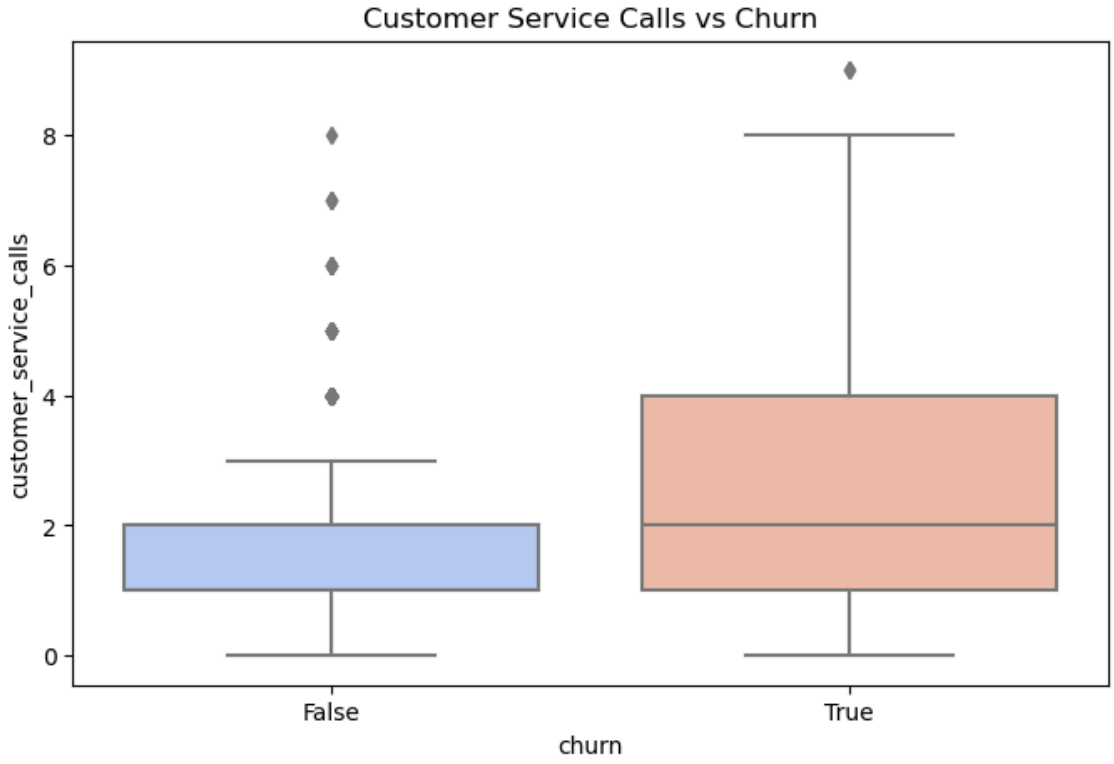


Insight: The number of customers who have stayed (false) 85.5% are more than the number of customers who have churned (True) 14.5% hence showing some class imbalance. Answers question 1

Bivariate Analysis

Churn vs Customer Service

In [32]:
```
plt.figure(figsize=(8,5))
sns.boxplot(x='churn', y='customer_service_calls', data=df, palette="coolwa
plt.title("Customer Service Calls vs Churn")
plt.show()
```



Insight: Number of True is more than False this represents a lot of disatisfaction as number of customers who have churned are more.

Outliers for Churned Customers (churn=True)

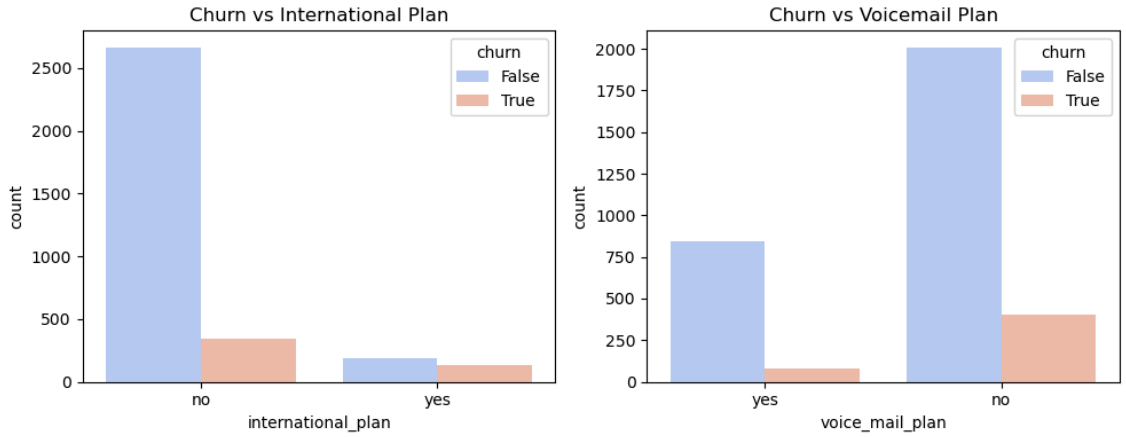
These are customers who called customer service significantly more than the average churned customer. This suggests that frustration with service issues may have pushed them to leave. If many outliers exist, it could mean that a small group of customers experienced repeated problems and escalated their complaints before eventually churning.

Outliers for Non-Churned Customers (churn=False)

These are customers who also made many service calls but did not churn. This could indicate that customer service successfully resolved their issues, or they were loyal despite frequent complaints. If non-churned outliers are close in range to churned ones, it might mean the difference between staying and leaving is how well customer service handled the issue.
Answers question 5.

Churn vs. Service Plans

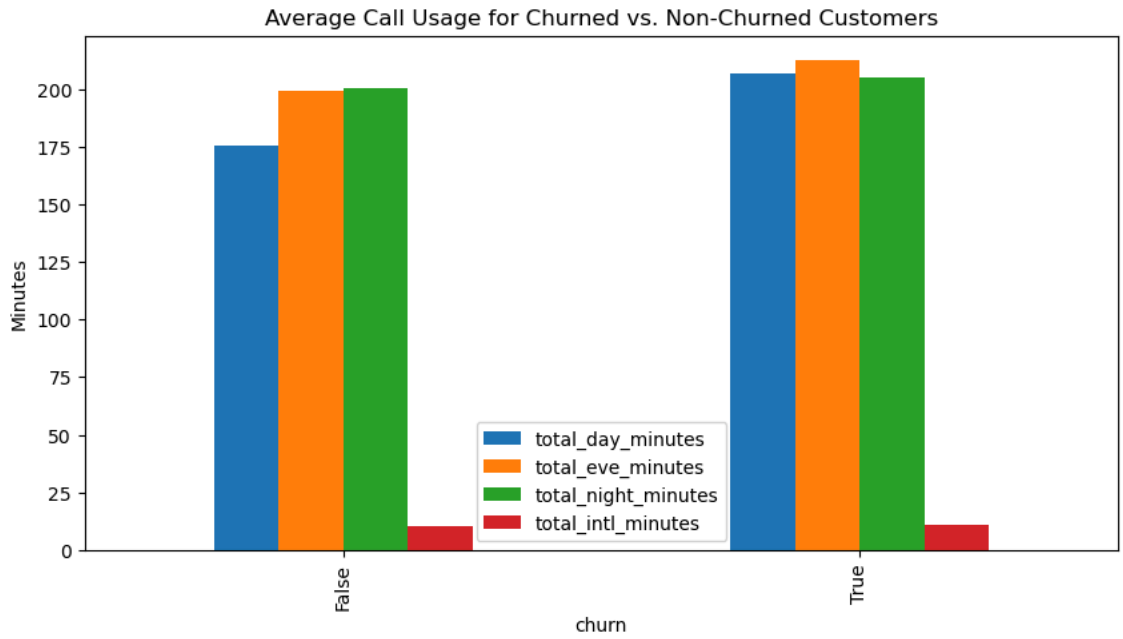
In [33]:
```
# Convert boolean columns to object (string)
df["churn"] = df["churn"].astype(str)
df["international_plan"] = df["international_plan"].astype(str)
df["voice_mail_plan"] = df["voice_mail_plan"].astype(str)


plt.figure(figsize=(10,4))

# Churn vs International Plan
plt.subplot(1,2,1)
sns.countplot(x="international_plan", hue="churn", data=df, palette="coolwa
plt.title("Churn vs International Plan")

# Churn vs Voicemail Plan
plt.subplot(1,2,2)
sns.countplot(x="voice_mail_plan", hue="churn", data=df, palette="coolwarm"
plt.title("Churn vs Voicemail Plan")

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



Insight:Voicemail Plan has a lot of customers who have stayed than international plan.
Answers Question 3

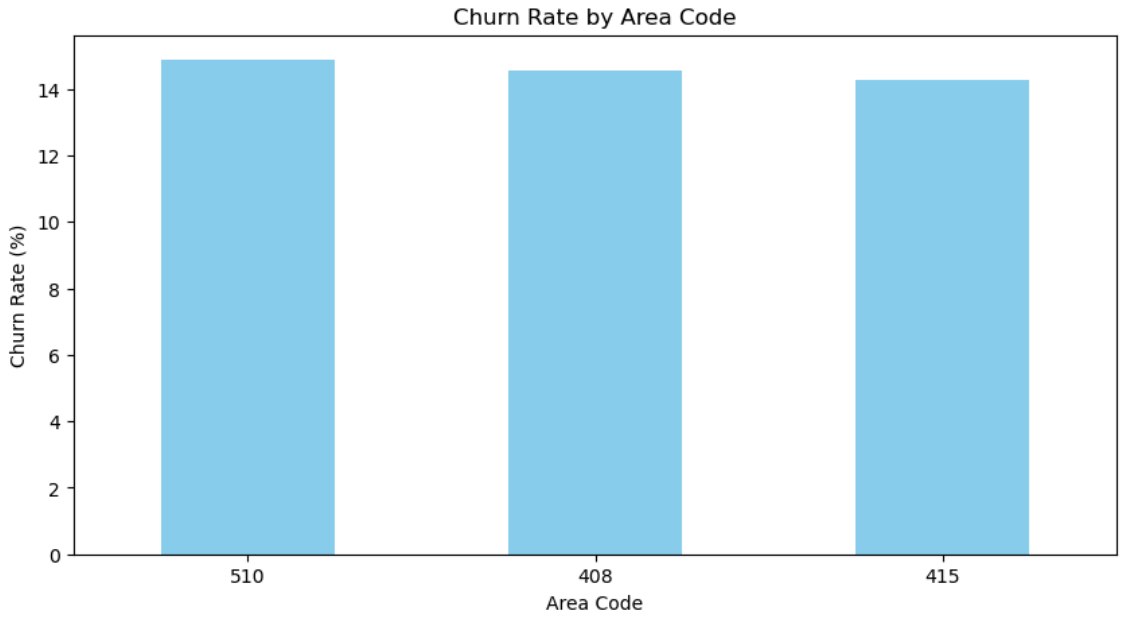
Average Call Usage for Churned vs. Non-Churned Customers

In [34]:
```
usage_features = ['total_day_minutes', 'total_eve_minutes', 'total_night_m
df.groupby("churn")[usage_features].mean().plot(kind="bar", figsize=(10, 5
plt.title("Average Call Usage for Churned vs. Non-Churned Customers")
plt.ylabel("Minutes")
plt.show()
```



Insight: those who have churned are more than those customers who have stayed according the frequency of usage . The more the minutes the more the churn. Answers Question 4

In [35]:
```
df["churn"] = df["churn"].map({'False':0,'True': 1}) #change into numerical

# Compute churn rate for each area code
area_churn = df.groupby("area_code")["churn"].mean() * 100  # Get mean chur

# Plot churn rate by area code
plt.figure(figsize=(10, 5))
area_churn.sort_values(ascending=False).plot(kind="bar", color="skyblue")
plt.title("Churn Rate by Area Code")
plt.ylabel("Churn Rate (%)")
plt.xlabel("Area Code")
plt.xticks(rotation=0)  # Keep area codes readable
plt.show()
```



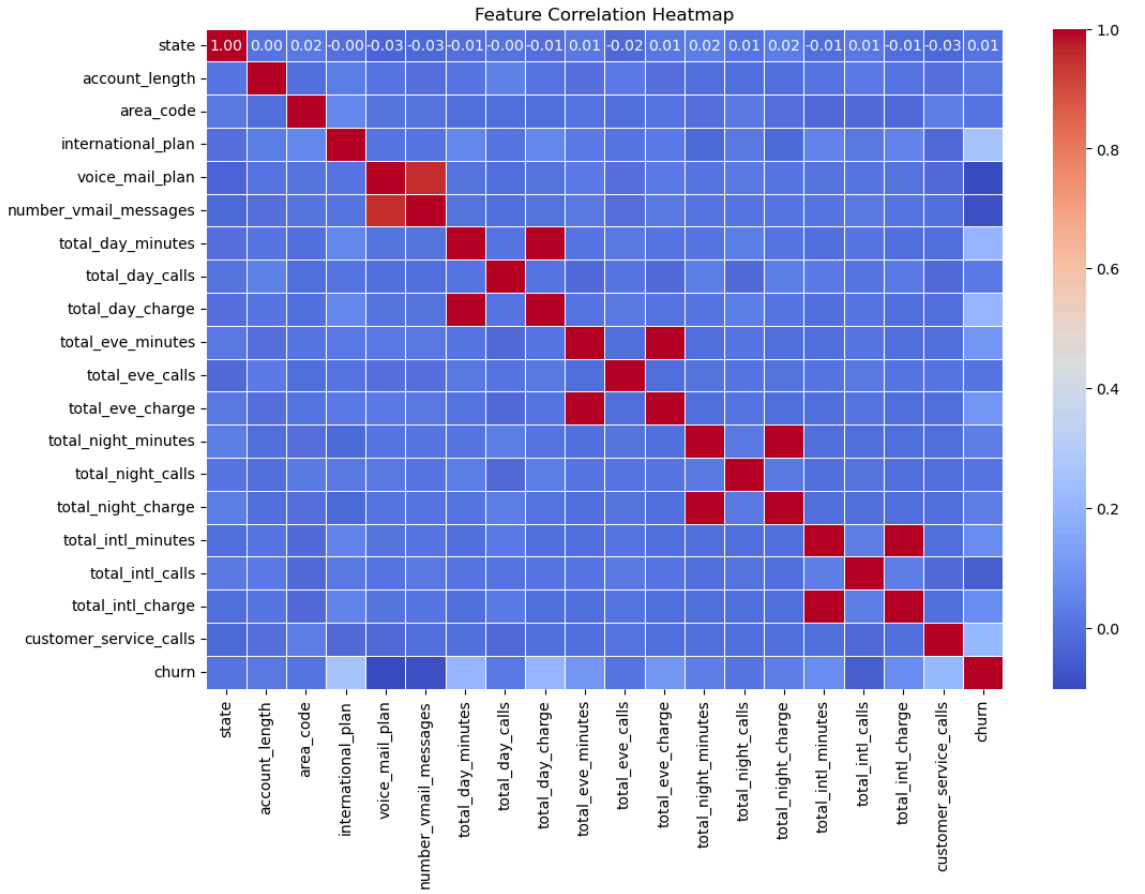
Insight: Area 510 has more churn rate than area 408 and 415. Answers Question 6.

Multivariate Analysis

Correlation Heat Map

In [36]:
```
# changing categorical columns into numerical
df_encoded = df.copy()
encoder = LabelEncoder()

# Apply encoding to categorical columns
for col in ["international_plan", "voice_mail_plan", "churn", 'state']:
    df_encoded[col] = encoder.fit_transform(df_encoded[col])
```

In [37]:
```
# # Compute correlation on numeric data
corr_matrix = df_encoded.corr()

# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths
plt.title("Feature Correlation Heatmap")
plt.show()
```



In [38]:
```
# Compute correlation matrix
corr_matrix = df_encoded.corr()

# Display the correlation values
print(corr_matrix)
```

```
                          state  account_length  area_code  \
state                  1.000000        0.003678   0.015814
account_length         0.003678        1.000000  -0.012463
area_code              0.015814       -0.012463   1.000000
international_plan    -0.004597        0.024735   0.048551
voice_mail_plan       -0.031664        0.002918  -0.000747
number_vmail_messages -0.027762       -0.004628  -0.001994
total_day_minutes     -0.006737        0.006216  -0.008264
total_day_calls       -0.000764        0.038470  -0.009646
total_day_charge      -0.006736        0.006214  -0.008264
total_eve_minutes      0.013682       -0.006757   0.003580
total_eve_calls       -0.016268        0.019260  -0.011886
total_eve_charge       0.013674       -0.006745   0.003607
total_night_minutes    0.024576       -0.008955  -0.005825
total_night_calls      0.007458       -0.013176   0.016522
total_night_charge     0.024572       -0.008960  -0.005845
total_intl_minutes    -0.007834        0.009514  -0.018288
total_intl_calls       0.013967        0.020661  -0.024179
total_intl_charge     -0.007819        0.009546  -0.018395
```

Insight: International charge, total day minutes, total day charge and customer service are higly correlated with churn. Answers Question 2.

Preprocessing

In [39]:
```
# Create a list of columns to encode
categorical_columns = ['international_plan', 'voice_mail_plan']

# Create a copy of the DataFrame with the selected columns
encoded_df = df.copy()

# Create an instance of OneHotEncoder
# sparse=False to produce a dense array and drop='first' to drop the first
encoder = OneHotEncoder(sparse_output=False)

# Iterate through each categorical column

for column in categorical_columns:
    # Fit and transform the selected column
    one_hot_encoded = encoder.fit_transform(encoded_df[[column]])

    # Create a DataFrame with one-hot encoded columns
    one_hot_df = pd.DataFrame(one_hot_encoded, columns=encoder.get_feature_

    # Concatenate the one-hot encoded DataFrame with the original DataFrame
    encoded_df = pd.concat([encoded_df, one_hot_df], axis=1)

    # Drop the original categorical column
    encoded_df = encoded_df.drop([column], axis=1)

# Display the resulting DataFrame
df= encoded_df.copy()

df.head()
```

Out[39]:

	state	account_length	area_code	number_vmail_messages	total_day_minutes	total_day_
0	KS	128	415	25	265.1	
1	OH	107	415	26	161.6	
2	NJ	137	415	0	243.4	
3	OH	84	408	0	299.4	
4	OK	75	415	0	166.7	

5 rows × 22 columns

Feature engineering

In [40]:
```
df.columns
```

Out[40]:
```
Index(['state', 'account_length', 'area_code', 'number_vmail_messages',
       'total_day_minutes', 'total_day_calls', 'total_day_charge',
       'total_eve_minutes', 'total_eve_calls', 'total_eve_charge',
       'total_night_minutes', 'total_night_calls', 'total_night_charge',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge',
       'customer_service_calls', 'churn', 'international_plan_no',
       'international_plan_yes', 'voice_mail_plan_no', 'voice_mail_plan_y
es'],
      dtype='object')
```

In [41]:
```
df["international_plan_total"] = df["international_plan_yes"] + df["interna
df["voice_mail_plan_total"] = df["voice_mail_plan_yes"] + df["voice_mail_pl
df.columns
```

Out[41]:
```
Index(['state', 'account_length', 'area_code', 'number_vmail_messages',
       'total_day_minutes', 'total_day_calls', 'total_day_charge',
       'total_eve_minutes', 'total_eve_calls', 'total_eve_charge',
       'total_night_minutes', 'total_night_calls', 'total_night_charge',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge',
       'customer_service_calls', 'churn', 'international_plan_no',
       'international_plan_yes', 'voice_mail_plan_no', 'voice_mail_plan_y
es',
       'international_plan_total', 'voice_mail_plan_total'],
      dtype='object')
```

In [42]:
```
df = df.drop(columns=['international_plan_no', 'international_plan_yes','vo
df.head()
```

Out[42]:

	state	account_length	area_code	number_vmail_messages	total_day_minutes	total_day_
0	KS	128	415	25	265.1	
1	OH	107	415	26	161.6	
2	NJ	137	415	0	243.4	
3	OH	84	408	0	299.4	
4	OK	75	415	0	166.7	

In [43]:
```
# Convert all numeric columns explicitly
df["total_day_calls"] = pd.to_numeric(df["total_day_calls"], errors="coerce
df["total_eve_calls"] = pd.to_numeric(df["total_eve_calls"], errors="coerce
df["total_night_calls"] = pd.to_numeric(df["total_night_calls"], errors="co
df["total_intl_calls"] = pd.to_numeric(df["total_intl_calls"], errors="coer

df["total_calls"] = (
    df["total_day_calls"] + df["total_eve_calls"] + df["total_night_calls"]
)
```

In [44]:
```
charge_cols = ["total_day_charge", "total_eve_charge", "total_night_charge"

# Convert them to numeric (handling errors safely)
df[charge_cols] = df[charge_cols].apply(pd.to_numeric, errors="coerce")

# Compute total charges
df["total_charges"] = df[charge_cols].sum(axis=1)
```

In [45]:
```
# List of columns to sum
minute_cols = ["total_day_minutes", "total_eve_minutes", "total_night_minut

# Convert them to numeric (handling errors safely)
df[minute_cols] = df[minute_cols].apply(pd.to_numeric, errors="coerce")

# Compute total minutes
df["total_minutes"] = df[minute_cols].sum(axis=1)
```

In [46]:
```
df.columns
```

Out[46]:
```
Index(['state', 'account_length', 'area_code', 'number_vmail_messages',
       'total_day_minutes', 'total_day_calls', 'total_day_charge',
       'total_eve_minutes', 'total_eve_calls', 'total_eve_charge',
       'total_night_minutes', 'total_night_calls', 'total_night_charge',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge',
       'customer_service_calls', 'churn', 'international_plan_total',
       'voice_mail_plan_total', 'total_calls', 'total_charges',
       'total_minutes'],
      dtype='object')
```

In [47]:
```
df = df.drop(columns=['total_day_minutes', 'total_day_calls', 'total_day_ch
       'total_eve_minutes', 'total_eve_calls', 'total_eve_charge',
       'total_night_minutes', 'total_night_calls', 'total_night_charge',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge' ], err
df.columns
```

Out[47]:
```
Index(['state', 'account_length', 'area_code', 'number_vmail_messages',
       'customer_service_calls', 'churn', 'international_plan_total',
       'voice_mail_plan_total', 'total_calls', 'total_charges',
       'total_minutes'],
      dtype='object')
```

In [143]:
```
df.to_csv("updated_Syria_Tel.csv", index=False)
```

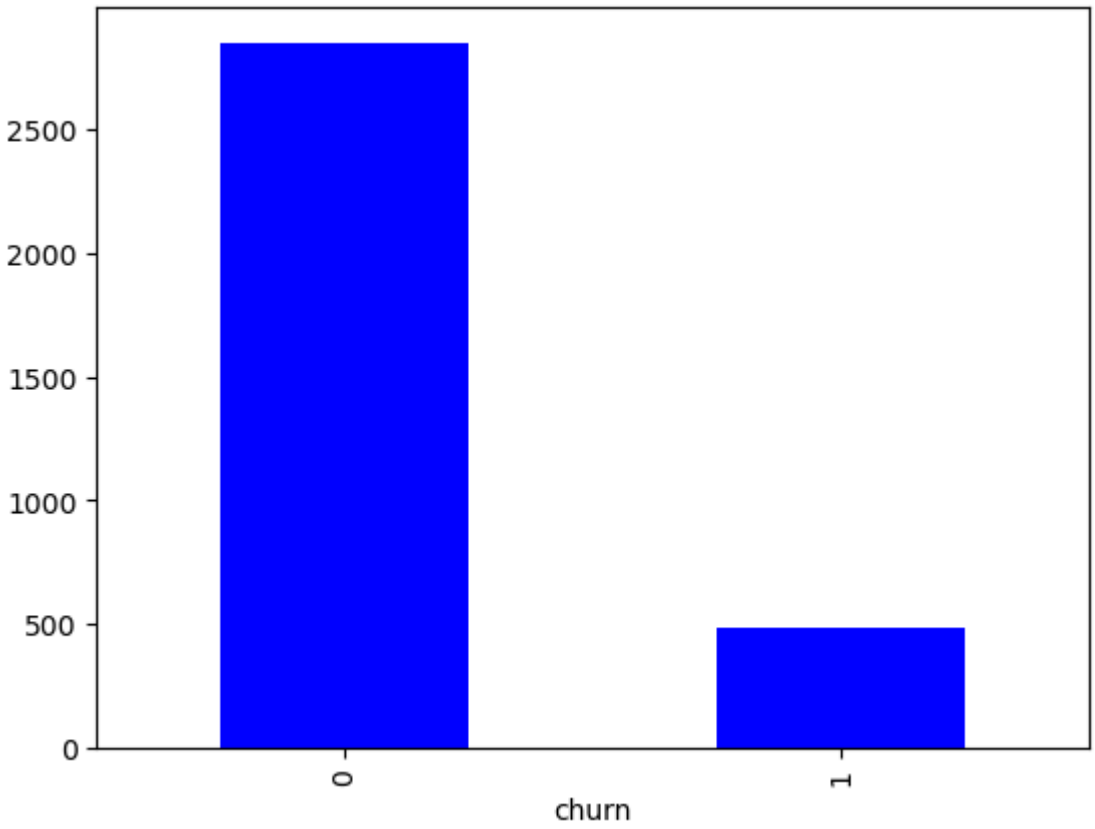
Creating a target variable

In [48]:
```
x= df.drop(['churn', 'state'] ,axis=1)
y = df['churn']
```

Checking Class Imbalance

In [170]:
```
target_value =df['churn'].value_counts()
target_value.plot(kind='bar', color=['blue'])
print(target_value)
```

```
churn
0    2850
1     483
Name: count, dtype: int64
```



In [171]:
```
# traim -test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, ra
```

In [172]:
```
from sklearn.preprocessing import StandardScaler
# scaling

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

Peformance before SMOTE

In [173]:
```
from sklearn.ensemble import GradientBoostingClassifier

log_reg = LogisticRegression()
dt = DecisionTreeClassifier()
rf = RandomForestClassifier()
gbc = GradientBoostingClassifier()


# Make predictions
models = {
    "Logistic Regression": log_reg,
    "Decision Tree": dt,
    "Random Forest": rf,
    "Gadient Boost": gbc
}


# Store model results
results = []

for name, model in models.items():
    # Measure training time
    start_train = time.time()
    model.fit(x_train_scaled, y_train)
    end_train = time.time()
    training_time = end_train - start_train

    # Measure prediction time
    start_pred = time.time()
    y_pred = model.predict(x_test_scaled)
    y_prob = model.predict_proba(x_test_scaled)[:, 1] # For ROC-AUC
    end_pred = time.time()
    prediction_time = end_pred - start_pred

    # Store results
    results.append({
        "Model": name,
        "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),
        "ROC-AUC": roc_auc_score(y_test, y_prob),
        "Training Time (s)": training_time,
        "Prediction Time (s)": prediction_time
    })

# Convert results to DataFrame
df_results = pd.DataFrame(results)
df_results
```

Out[173]:

	Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC	Training Time (s)	Prediction Time (s)
0	Logistic Regression	0.851574	0.450000	0.092784	0.153846	0.753247	0.100053	0.006655
1	Decision Tree	0.892054	0.628866	0.628866	0.628866	0.782854	0.060936	0.003296
2	Random Forest	0.940030	0.983051	0.597938	0.743590	0.774191	1.264563	0.094894
3	Gadient Boost	0.940030	0.983051	0.597938	0.743590	0.805761	1.187917	0.000000

In [174]:
```
# Store results in a list
cm_results = []

# Loop through models
for name, model in models.items():
    y_pred = model.predict(x_test_scaled)  # Get predictions
    cm = confusion_matrix(y_test, y_pred)  # Compute confusion matrix

    # Append results in dictionary form
    cm_results.append({
        "Model": name,
        "TN": cm[0, 0],  # True Negatives
        "FP": cm[0, 1],  # False Positives
        "FN": cm[1, 0],  # False Negatives
        "TP": cm[1, 1]   # True Positives
    })

# Convert list of dictionaries into a DataFrame
cm_df = pd.DataFrame(cm_results)

# Print confusion matrix table
cm_df
```

Out[174]:

	Model	TN	FP	FN	TP
0	Logistic Regression	559	11	88	9
1	Decision Tree	534	36	36	61
2	Random Forest	569	1	39	58
3	Gadient Boost	569	1	39	58

Performance After Using SMOTE to handle class imbalance

In [152]:
```
from imblearn.over_sampling import SMOTE
from collections import Counter

# Initialize SMOTE
smote = SMOTE(random_state=1)

# Apply SMOTE to the training data
x_train_resampled, y_train_resampled = smote.fit_resample(x_train, y_train)

# Print class distribution before and after
print("Before SMOTE:", Counter(y_train))  # Original class distribution
print("After SMOTE:", Counter(y_train_resampled))
```

```
Before SMOTE: Counter({0: 2280, 1: 386})
After SMOTE: Counter({0: 2280, 1: 2280})
```

In [153]:
```
from sklearn.preprocessing import StandardScaler
# scaling

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train_resampled)
x_test_scaled = scaler.transform(x_test)
```

In [163]:
```
log_reg2 = LogisticRegression()
dt2 = DecisionTreeClassifier()
rf2 = RandomForestClassifier()
gbc2=GradientBoostingClassifier()

# Make predictions
models1 = {
    "Logistic Regression": log_reg2,
    "Decision Tree": dt2,
    "Random Forest": rf2,
    "Gradient Boost Classifier" : gbc2
}


# Store model results
results2 = []

for name, model1 in models1.items():
    # Measure training time
    start_train = time.time()
    model1.fit(x_train_scaled, y_train_resampled)
    end_train = time.time()
    training_time = end_train - start_train

    # Measure prediction time
    start_pred = time.time()
    y_pred = model1.predict(x_test_scaled)
    y_prob = model1.predict_proba(x_test_scaled)[:, 1] # For ROC-AUC
    end_pred = time.time()
    prediction_time = end_pred - start_pred

    # Store results
    results2.append({
        "Model": name,
        "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),
        "ROC-AUC": roc_auc_score(y_test, y_prob),
        "Training Time (s)": training_time,
        "Prediction Time (s)": prediction_time
    })

# Convert results to DataFrame
df_results2 = pd.DataFrame(results2)
df_results2
```

Out[163]:

	Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC	Training Time (s)	Prediction Time (s)
0	Logistic Regression	0.680660	0.276923	0.742268	0.403361	0.749286	0.061607	0.000000
1	Decision Tree	0.817091	0.413793	0.618557	0.495868	0.734717	0.042528	0.000000
2	Random Forest	0.914543	0.756410	0.608247	0.674286	0.754015	1.610341	0.067848
3	Gradient Boost Classifier	0.907046	0.705882	0.618557	0.659341	0.776216	1.808963	0.008032

To determine the best model, consider the most relevant metric(s) for your churn prediction problem. Here's how you can evaluate:

Key Insights from the Table

Accuracy: Random Forest has the highest accuracy (0.9145), followed by Gradient Boosting (0.9070).

Precision: Random Forest (0.7564) is the best, meaning it has the highest proportion of correctly predicted churn cases out of all churn predictions.

Recall: Logistic Regression (0.7423) is the highest, meaning it captures the most actual churn cases, but its precision is very low.

F1 Score: Random Forest (0.6743) balances precision and recall well.

ROC-AUC: Gradient Boosting (0.7762) is the best, meaning it better separates churners from non-churners. Training & Prediction Time:

Decision Tree is the fastest to train and predict.

Random Forest & Gradient Boosting are more computationally expensive.

Best Model?

If you prioritize overall performance (Accuracy, F1, ROC-AUC): Random Forest

If you want the best churn separation (ROC-AUC): Gradient Boosting

If speed is a major factor: Decision Tree is a good trade-off.

If recall is crucial (capturing more churners): Logistic Regression, but it has lower precision.

Final Recommendation:

Random Forest is the best overall model due to its strong performance across multiple metrics, particularly accuracy, precision, and F1-score.

Confusion Matrix of each model

In [165]:
```
# Store results in a list
cm_results2 = []

# Loop through models
for name, model1 in models1.items():
    y_pred = model1.predict(x_test_scaled)  # Get predictions
    cm = confusion_matrix(y_test, y_pred)  # Compute confusion matrix

    # Append results in dictionary form
    cm_results2.append({
        "Model": name,
        "TN": cm[0, 0],  # True Negatives
        "FP": cm[0, 1],  # False Positives
        "FN": cm[1, 0],  # False Negatives
        "TP": cm[1, 1]   # True Positives
    })

# Convert list of dictionaries into a DataFrame
cm_df2 = pd.DataFrame(cm_results2)

# Print confusion matrix table
cm_df2
```

Out[165]:

	Model	TN	FP	FN	TP
0	Logistic Regression	382	188	25	72
1	Decision Tree	485	85	37	60
2	Random Forest	551	19	38	59
3	Gradient Boost Classifier	545	25	37	60

Which Model is Best?

If capturing the most churners is the priority: Logistic Regression (Highest TP, lowest FN).

If reducing false alarms (FP) is crucial: Random Forest or Gradient Boosting.

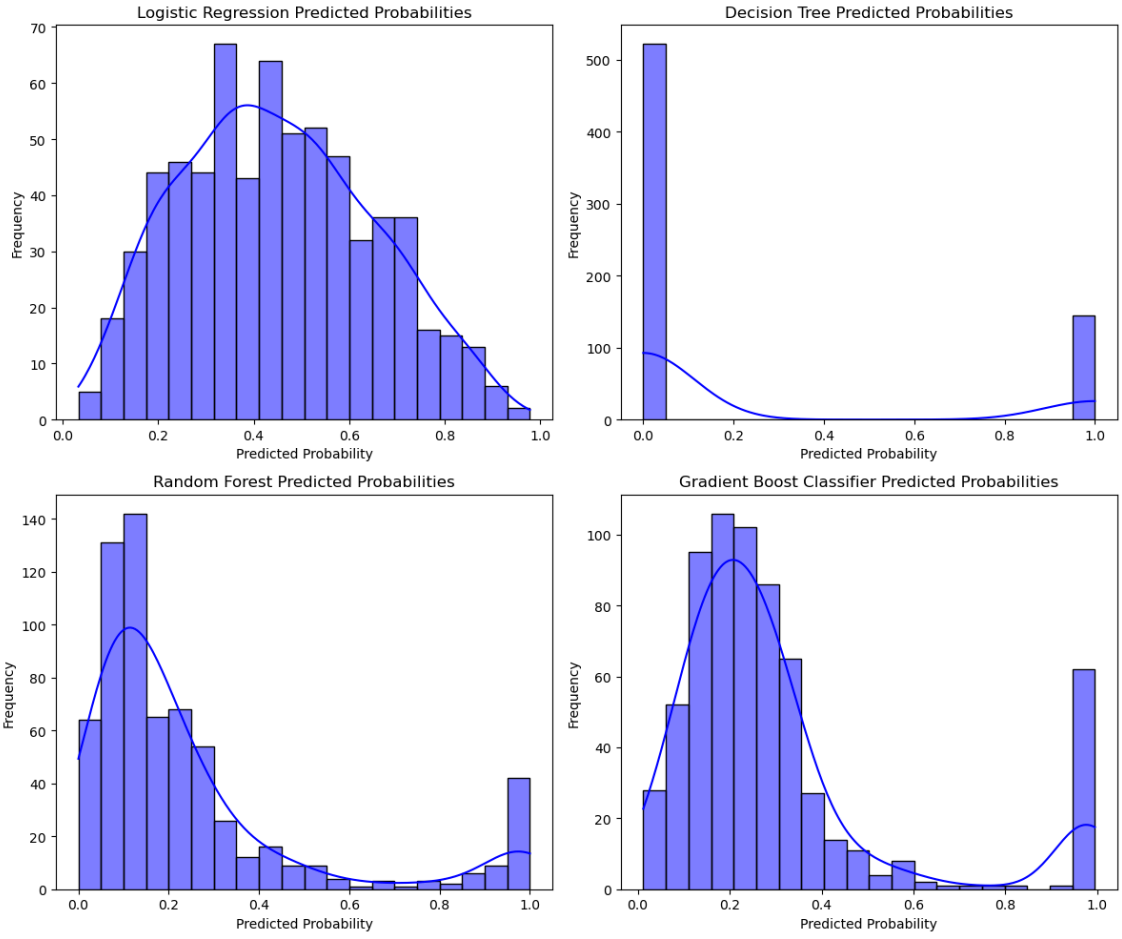
If a balance is needed: Gradient Boosting (Similar to Decision Tree but lower FP).

If accuracy in non-churners is more important: Random Forest (highest TN, lowest FP).

Random Forest or Gradient Boosting for a balanced approach.

In [166]:

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

# Set up the plot grid
fig, axes = plt.subplots(2, 2, figsize=(12, 10))  # Adjust grid size based
axes = axes.flatten()  # Flatten to iterate easily

# Loop through models and plot histograms
for i, (name, model1) in enumerate(models1.items()):
    # Get predicted probabilities
    y_prob = model1.predict_proba(x_test_scaled)[:, 1] # For ROC-AUC

    # Plot histogram
    sns.histplot(y_prob, bins=20, kde=True, color="blue", ax=axes[i])
    axes[i].set_title(f"{name} Predicted Probabilities")
    axes[i].set_xlabel("Predicted Probability")
    axes[i].set_ylabel("Frequency")

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



The histogram shows the distribution of predicted probabilities for the positive class (churn) across different models. Here's how you can interpret it:

Shape and Spread of the Distribution

If the probabilities are skewed toward 0, the model is more confident that most customers will not churn. If the probabilities are skewed toward 1, the model is more confident that most customers will churn. If the probabilities are spread out, the model is more uncertain about predictions. Model Confidence

A bimodal distribution (peaks near both 0 and 1) indicates that the model confidently separates churners from non-churners. A normal or uniform distribution (spread across the middle) suggests that the model is uncertain in distinguishing churners from non-churners. Comparing Models

If one model has probabilities mostly close to 0 and 1, it means it makes stronger classifications and is more confident. If another model has probabilities concentrated around 0.5, it means it is less confident and struggles to differentiate churners from non-churners. Impact on Threshold Selection

In real-world applications, you may adjust the decision threshold (default is 0.5) based on business needs. If a model has many probabilities around 0.5, a small threshold change could significantly impact performance (e.g., setting a lower threshold to catch more potential churners).

Key Takeway

Hyperparameter tuning of each model

In [118]:
```
from sklearn.preprocessing import StandardScaler
# scaling

scaler = StandardScaler()
x_train_scaled= scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

Logistics Regression

In [176]:
```
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
    "C": [0.01, 0.1, 1, 10, 100],  # Regularization strength
    "penalty": ["l1", "l2"],  # L1 = Lasso, L2 = Ridge
    "solver": ["liblinear", "saga"]  # Supports both L1 and L2
}

# Initialize Logistic Regression with class weights
log_reg1 = LogisticRegression(class_weight="balanced", random_state=1)

# Perform GridSearchCV
grid_search1 = GridSearchCV(log_reg1, param_grid, cv=5, scoring="f1", n_jo
grid_search1.fit(x_train_scaled, y_train)

# Get the best parameters
print("Best Parameters:", grid_search1.best_params_)

best_log_reg = grid_search1.best_estimator_
```

```
Best Parameters: {'C': 0.01, 'penalty': 'l1', 'solver': 'saga'}
```

In [191]:
```
y_pred = best_log_reg.predict(x_test_scaled)
y_prob = best_log_reg.predict_proba(x_test)[:, 1]

# Print performance metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score,

print("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("ROC-AUC Score:", roc_auc_score(y_test, y_prob))
print("training data score: " + str(best_log_reg.score(x_train_scaled, y_t
print('test data score ' + str(best_log_reg.score(x_test_scaled, y_test)))
```

```
Accuracy: 0.7061469265367316
Precision: 0.2838427947598253
Recall: 0.6701030927835051
F1 Score: 0.39877300613496935
ROC-AUC Score: 0.7149936697413637
training data score: 0.7145536384096024
test data score 0.7061469265367316
```

Decision Tree

Grid Search Method

In [108]:
```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 5, 10],
    'criterion': ['gini', 'entropy'],
    "class_weight": ["balanced", None]

}

# Run GridSearchCV
dt_model1 = DecisionTreeClassifier(random_state=1)
grid_search_dt = GridSearchCV(dt_model1, param_grid, cv=5, scoring='accura
grid_search_dt.fit(x_train_scaled, y_train)

# Get best parameters
print("Best Parameters:", grid_search_dt.best_params_)

# Best model
best_dt_model = grid_search_dt.best_estimator_
```

```
Best Parameters: {'class_weight': None, 'criterion': 'entropy', 'max_dept
h': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

In [109]:
```
y_pred = best_dt_model.predict(x_test_scaled)
y_prob = best_dt_model.predict_proba(x_test)[:, 1]

# Print performance metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score,

print("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("ROC-AUC Score:", roc_auc_score(y_test, y_prob))
```

```
Accuracy: 0.9415292353823088
Precision: 1.0
Recall: 0.5979381443298969
F1 Score: 0.7483870967741935
ROC-AUC Score: 0.5520618556701031
```

Random Forest

In [112]:

```
# Define the model
rf_model1= RandomForestClassifier(random_state=1)

# Define hyperparameters to tune
param_grid = {
    "n_estimators": [100, 200, 300],  # Number of trees
    "max_depth": [10, 20, 30],  # Tree depth
    "min_samples_split": [2, 5, 10],  # Min samples to split
    "min_samples_leaf": [1, 2, 4],  # Min samples in a leaf
    "class_weight": ["balanced", None]  # Handling class imbalance

}

# GridSearchCV
grid_search_rf = GridSearchCV(
    estimator=rf_model1,
    param_grid=param_grid,
    scoring="accuracy",  # Can use 'roc_auc' for imbalanced data
    cv=5,  # Cross-validation
    n_jobs=-1,  # Use all available processors
    verbose=2
)

# Fit the model
grid_search_rf.fit(x_train_scaled, y_train)

# Best parameters
print("Best Parameters:", grid_search_rf.best_params_)
best_rf_model=grid_search_rf.best_estimator_
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Parameters: {'class_weight': 'balanced', 'max_depth': 10, 'min_sampl
es_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
```

In [194]:
```
y_pred = best_rf_model.predict(x_test_scaled)
y_prob = best_rf_model.predict_proba(x_test_scaled)[:, 1]

# Print performance metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score,

print("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("ROC-AUC Score:", roc_auc_score(y_test, y_prob))
print("training data score: " + str(best_rf_model.score(x_train_scaled, y_t
print('test data score ' + str(best_rf_model.score(x_test_scaled, y_test))
```

```
Accuracy: 0.9400299850074962
Precision: 0.9830508474576272
Recall: 0.5979381443298969
F1 Score: 0.7435897435897435
ROC-AUC Score: 0.766784228612769
training data score: 0.9654913728432108
test data score 0.9400299850074962
```

Gradient Boost CLassifier

In [178]:
```
gbc1 = GradientBoostingClassifier(random_state=1)

# Define the hyperparameter grid
param_grid = {
    "n_estimators": [50, 100, 200],  # Number of trees
    "learning_rate": [0.01, 0.1, 0.2],  # Step size for tree updates
    "max_depth": [3, 5, 7],  # Tree depth
    "min_samples_split": [2, 5, 10],  # Minimum samples to split a node
    "min_samples_leaf": [1, 3, 5],  # Minimum samples per leaf
    "subsample": [0.8, 1.0]  # Fraction of samples used per tree
}

grid_search = GridSearchCV(
    gbc1, param_grid, cv=5, scoring="roc_auc", n_jobs=-1, verbose=1
)

# Fit the model on training data
grid_search.fit(x_train_scaled, y_train)

# Get the best parameters
print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 5 folds for each of 486 candidates, totalling 2430 fits
Best Parameters: {'learning_rate': 0.2, 'max_depth': 7, 'min_samples_lea
f': 1, 'min_samples_split': 2, 'n_estimators': 50, 'subsample': 1.0}
```

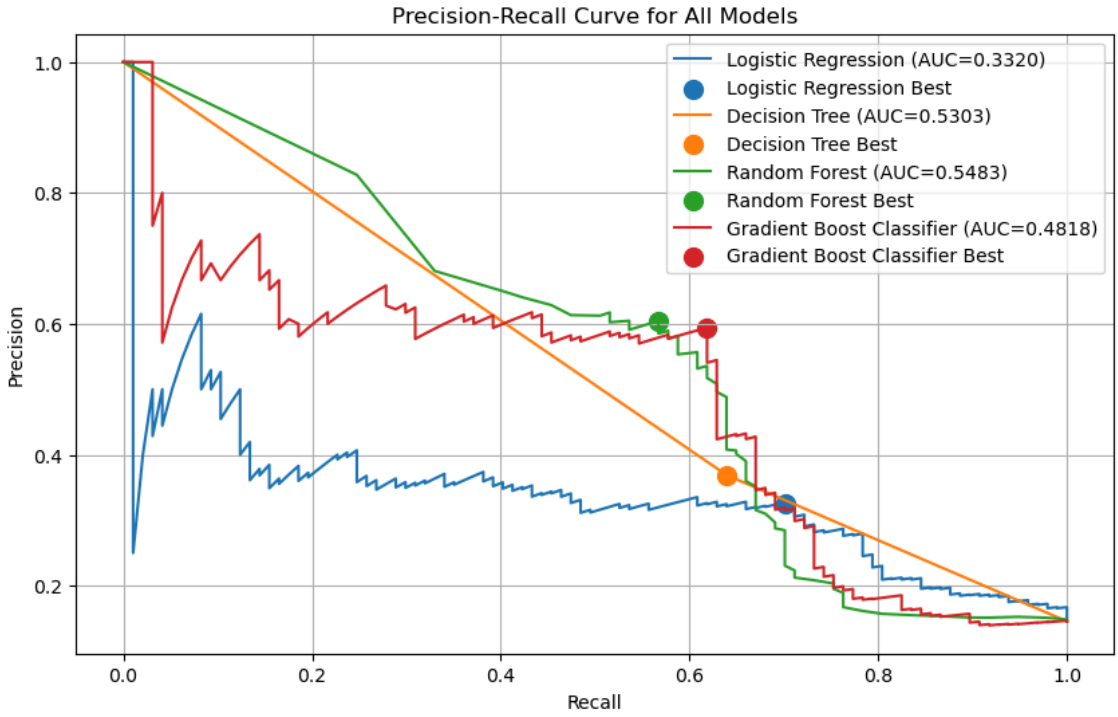
In [193]:
```
best_gbc_model=grid_search.best_estimator_
y_pred = best_gbc_model.predict(x_test_scaled)
y_prob = best_gbc_model.predict_proba(x_test_scaled)[:, 1]

# Print performance metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score,

print("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("ROC-AUC Score:", roc_auc_score(y_test, y_prob))
print("training data score: " + str(best_gbc_model.score(x_train_scaled, y_
print('test data score ' + str(best_gbc_model.score(x_test_scaled, y_test)
```

```
Accuracy: 0.9370314842578711
Precision: 0.9104477611940298
Recall: 0.6288659793814433
F1 Score: 0.7439024390243902
ROC-AUC Score: 0.7852595406040875
training data score: 0.9996249062265566
test data score 0.9370314842578711
```

Insights After Hypertuning: i did not see much of a difference after hypertuning so inferences just remain the same with Gradient Boost Classifier and Random Forest generally performing well

Precision-Recall Curve

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, auc

# Dictionary to store results
pr_results = {}

# Plot PR Curve for each model
plt.figure(figsize=(10, 6))

for name, model1 in models1.items():
    y_prob = model1.predict_proba(x_test_scaled)[:, 1]  # Get predicted pro

    precision, recall, thresholds = precision_recall_curve(y_test, y_prob)
    f1_scores = 2 * (precision * recall) / (precision + recall + 1e-9)  # A
    best_idx = np.argmax(f1_scores)  # Best threshold index

    best_threshold = thresholds[best_idx]
    best_precision = precision[best_idx]
    best_recall = recall[best_idx]

    # Store best results
    pr_results[name] = {
        "Best Threshold": best_threshold,
        "Precision": best_precision,
        "Recall": best_recall
    }

    # Plot curve
    plt.plot(recall, precision, label=f"{name} (AUC={auc(recall, precision)
    plt.scatter(best_recall, best_precision, marker="o", label=f"{name} Bes

plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve for All Models")
plt.legend()
plt.grid()
plt.show()

# Print best thresholds for each model
import pandas as pd
pr_df = pd.DataFrame.from_dict(pr_results, orient='index')
pr_df
```



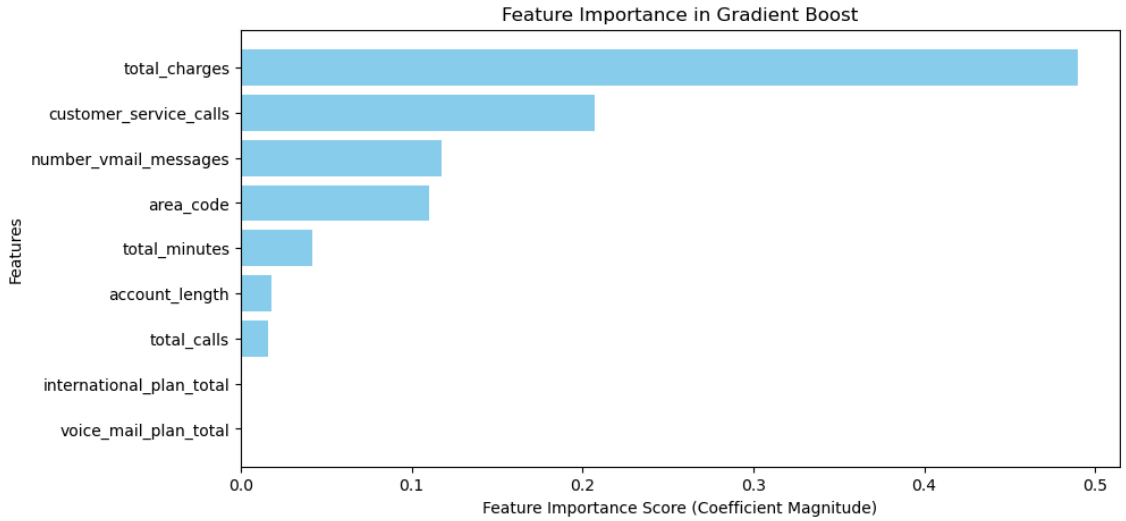
Out[182]:

	Best Threshold	Precision	Recall
Logistic Regression	0.619900	0.325359	0.701031
Decision Tree	1.000000	0.369048	0.639175
Random Forest	0.890000	0.604396	0.567010
Gradient Boost Classifier	0.741479	0.594059	0.618557

Insight :The overall balanced model is Random forest , Though Gradient Boosting seems to be the best with a recall of 0.62 and a precision of 0.59 though logistics regression seems to have High recall ensures you capture most of the actual churners. While the are under the curve is much larger for Gradient Boost and Random forest

Feature Importance

In [186]:
```
# Get feature importance using coefficients
feature_importance = gbc2.feature_importances_# Take absolute values

# Convert to a DataFrame
feature_names = x_train.columns  # Get column names
feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importance
}).sort_values(by='Importance', ascending=False)

# Plot Feature Importances
plt.figure(figsize=(10, 5))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importanc
plt.xlabel("Feature Importance Score (Coefficient Magnitude)")
plt.ylabel("Features")
plt.title("Feature Importance in Gradient Boost")
plt.gca().invert_yaxis()  # Invert y-axis for better readability
plt.show()
```



The top 3 Feature importance show very high correlation with churn here are the insights and recomendations

Total Charges (Billing & Spending Patterns)

Insight:

Customers with higher total charges might be more likely to churn. This could indicate dissatisfaction with pricing or unexpected charges.

Recommendations:

Offer tiered discounts or loyalty rewards for high-spending customers. Analyze billing complaints to identify frequent issues. Introduce flexible payment plans to prevent churn due to affordability concerns.

Customer Service Calls (Support Interaction)

Insight:

High customer service call frequency is strongly linked to churn. Customers who call support multiple times may have unresolved issues.

Recommendations:

Improve first-call resolution by training support staff to handle issues more effectively. Identify high-risk customers early and provide proactive support before they churn. Monitor call sentiment using AI to detect frustration and intervene before cancellation.

Number of Voicemail Messages (Usage Behavior)

Insight:

A lower number of voicemail messages may indicate low engagement with the service. Customers who don’t use voicemail may be switching to other communication apps.

Recommendations:

Promote voicemail features or integrate with popular messaging apps. Segment customers with low voicemail usage and offer alternative services. Survey customers to understand their communication preferences.

Final Takeaways

Billing issues drive churn → Provide flexible plans & loyalty discounts. Support interactions predict dissatisfaction → Improve service resolution & proactive support. Changing