Business Understanding

Predicting Customer Churn Model

In the telecommunications industry, customer churn presents a significant challenge. The objective is to develop a model that predicts whether a customer will soon terminate their services. This binary classification task aims to uncover patterns in customer behavior and demographic data that may indicate a propensity to churn. The ultimate goal is to aid in reducing the financial impact of customer churn by implementing proactive retention strategies.

Problem Statement

The challenge of retaining customers in a competitive telecommunications landscape is significant. Customer churn not only leads to revenue loss but also affects reputation and market position. The task is to develop a predictive model that accurately identifies customers likely to churn, enabling proactive intervention with targeted retention initiatives.The project therefore targets telecommunications companies, particularly those interested in understanding and predicting customer churn. Specifically, it aims to provide insights into the factors influencing customer attrition within the telecom industry. The dataset enables telecom companies to identify potential churn risks and implement targeted strategies to retain customers and improve overall satisfaction.

Objectives

i) Develop and Optimize Classification Models:

ii) Conduct Exploratory Data Analysis (EDA):

iii) Generate Insights and Recommendations:

Data Understanding

The dataset contains information about customers, their usage patterns, and whether they have churned or not. (Churn, or customer churn, is an important metric for companies to track when trying to expand their business. This metric represents the number of customers that have stopped using your product or service during a given period of time.). In addition, churn, in the context of telecommunications, is the process by which customers leave a business and stop using the services it provides either because they are unhappy with those services or because they can find better options from other network providers at more reasonable prices. This could result in a loss of revenue or profit for the business.

The dataset consists of 21 columns and 3333 rows representing various attributes of customers in the telecommunications industry. Here is an overview of the columns and their descriptions:

- `state` **: The US state in which the customer resides.**
- `account length` **: The duration (in days) of the customer's account with the company.**
- `area code` **: The area code of the customer's phone number.**
- `phone number` **: The customer's phone number.**
- `international plan` **: Whether the customer has an international calling plan (yes or no).**
- `voice mail plan` **: Whether the customer has a voice mail plan (yes or no).**
- `number vmail messages` **: The number of voice mail messages the customer has.**
- `total day minutes` **: The total number of minutes the customer has used during the day.**
- `total day calls` **: The total number of calls the customer has made during the day.**
- `total day charge` **: The total charges incurred by the customer for day usage.**

- `total eve minutes` : The total number of minutes the customer has used during the evening.
- `total eve calls` : The total number of calls the customer has made during the evening.
- `total eve charge` : The total charges incurred by the customer for evening usage.
- `total night minutes` : The total number of minutes the customer has used during the night.
- `total night calls` : The total number of calls the customer has made during the night.
- `total night charge` : The total charges incurred by the customer for night usage.
- `total intl minutes` : The total number of minutes the customer has used for international calls.
- `total intl calls` : The total number of international calls the customer has made.
- `total intl charge` : The total charges incurred by the customer for international usage.
- `customer service calls` : The number of calls the customer has made to customer service.
- `churn` : Whether the customer has churned (True or False).

Data Preparation

In [79]:

```
# Importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, conf
usion_matrix, roc_auc_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

In [80]:

```
# Load the dataset
df= pd.read_csv('Churn_dataset.csv')
```

In [81]:

```
# Cheking the data
df
```

Out[81]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121

...	... state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	... ...	total eve calls	total eve charge	total night minutes	total night calls
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...	126	18.32	279.1	83
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...	55	13.04	191.3	123
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...	58	24.55	191.9	91
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...	84	13.57	139.2	137
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	82	22.60	241.4	77

3333 rows × 21 columns

In [82]:

```
# check data info
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
```

The data types include integers, floats, booleans, and objects (likely strings).

Data Cleaning

Check Null Values

In [83]:

```
# Checking for null values
df.isnull().sum()
```

Out[83]:

```
state                     0
account length            0
area code                 0
phone number              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
```

Check duplicates

In [84]:

```
#check for duplicates
df.duplicated().sum()
```

Out[84]:

```
0
```

Convert Data types

In [85]:

```
# Convert 'churn' column to numeric (0 for False, 1 for True)
df['churn'] = df['churn'].astype(int)
df.dtypes
```

Out[85]:

```
state                    object
account length            int64
area code                 int64
phone number             object
international plan       object
voice mail plan          object
number vmail messages     int64
total day minutes       float64
total day calls           int64
total day charge        float64
total eve minutes       float64
total eve calls           int64
total eve charge        float64
total night minutes     float64
total night calls         int64
total night charge      float64
total intl minutes      float64
total intl calls          int64
total intl charge       float64
customer service calls    int64
churn                     int32
dtype: object
```

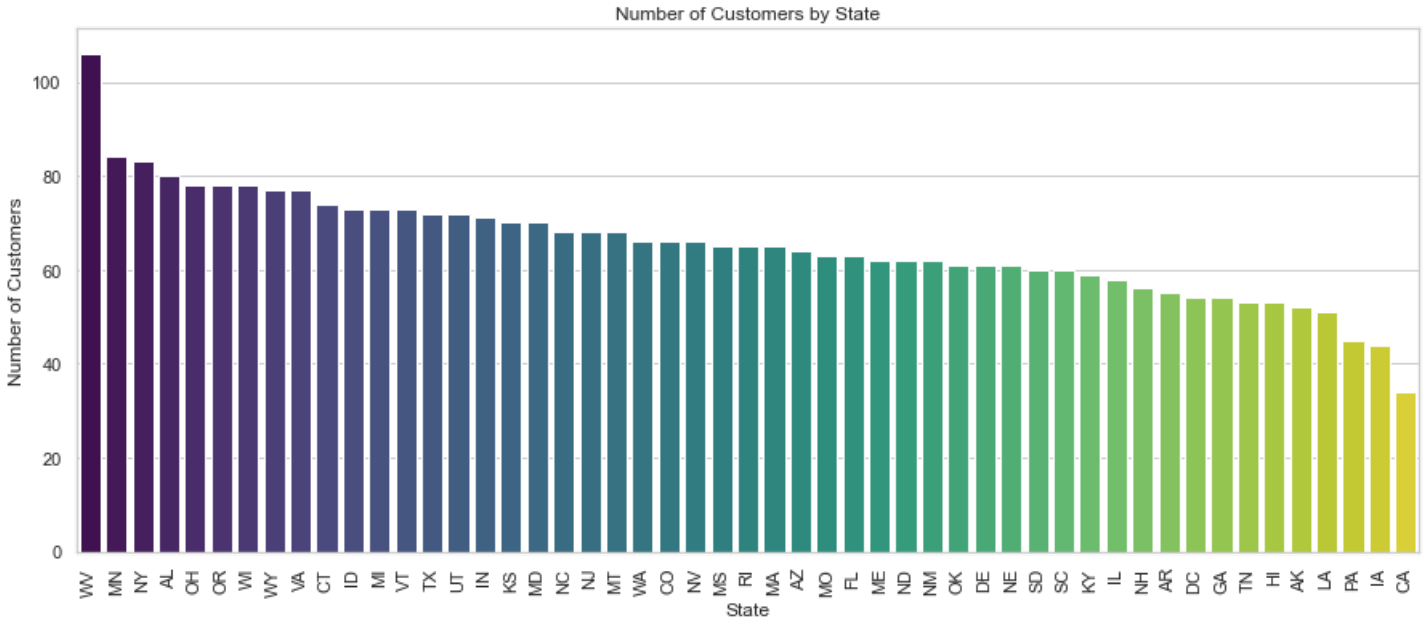
The dataframe has no missing values and no duplicates

Exploratory data analysis

Univariate Analysis

In [86]:

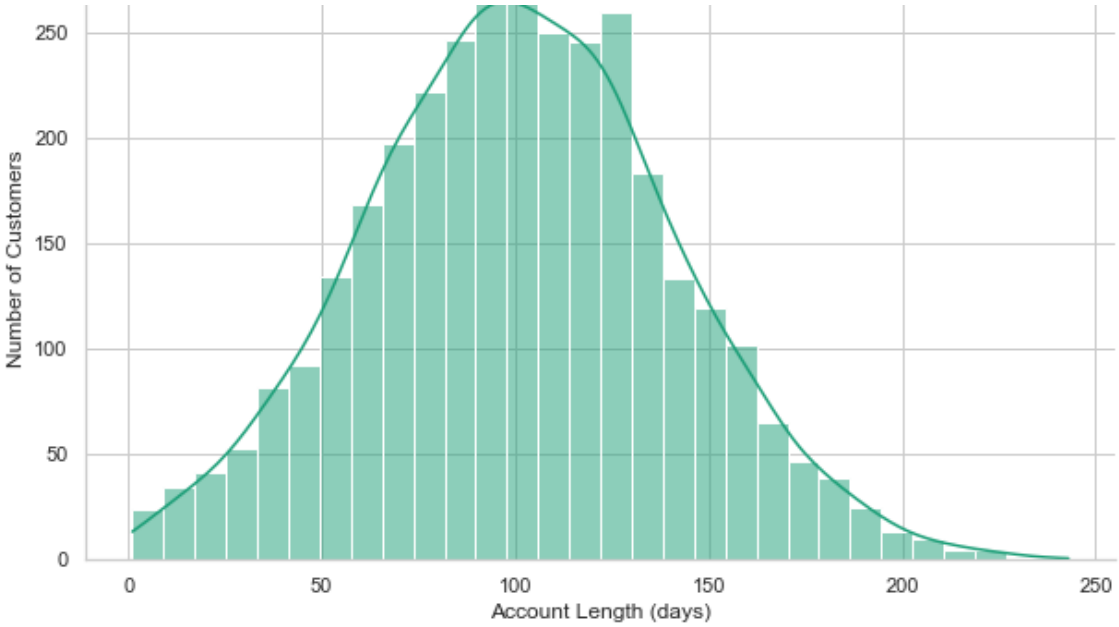
```
# State plot
# Count the number of customers in each state and sort the values
state_counts = df['state'].value_counts().sort_values(ascending=False)

# Create a bar plot
plt.figure(figsize=(15, 6))
sns.barplot(x=state_counts.index, y=state_counts.values, palette="viridis")

# Add titles and labels
plt.title('Number of Customers by State')
plt.xlabel('State')
plt.ylabel('Number of Customers')
plt.xticks(rotation=90)  # Rotate state labels for better readability

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



WV has the highest number of customers while CA has the lowest number of customers.

In [87]:

```
#Account length
# Set the style and palette for the plot
sns.set(style="whitegrid")
palette = sns.color_palette("Dark2")

# Create a histogram for the account length
plt.figure(figsize=(10, 6))
sns.histplot(df['account length'], bins=30, kde=True, color=palette[0])

# Add titles and labels
plt.title('Distribution of Account Length')
plt.xlabel('Account Length (days)')
plt.ylabel('Number of Customers')

# Hide top and right spines
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

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



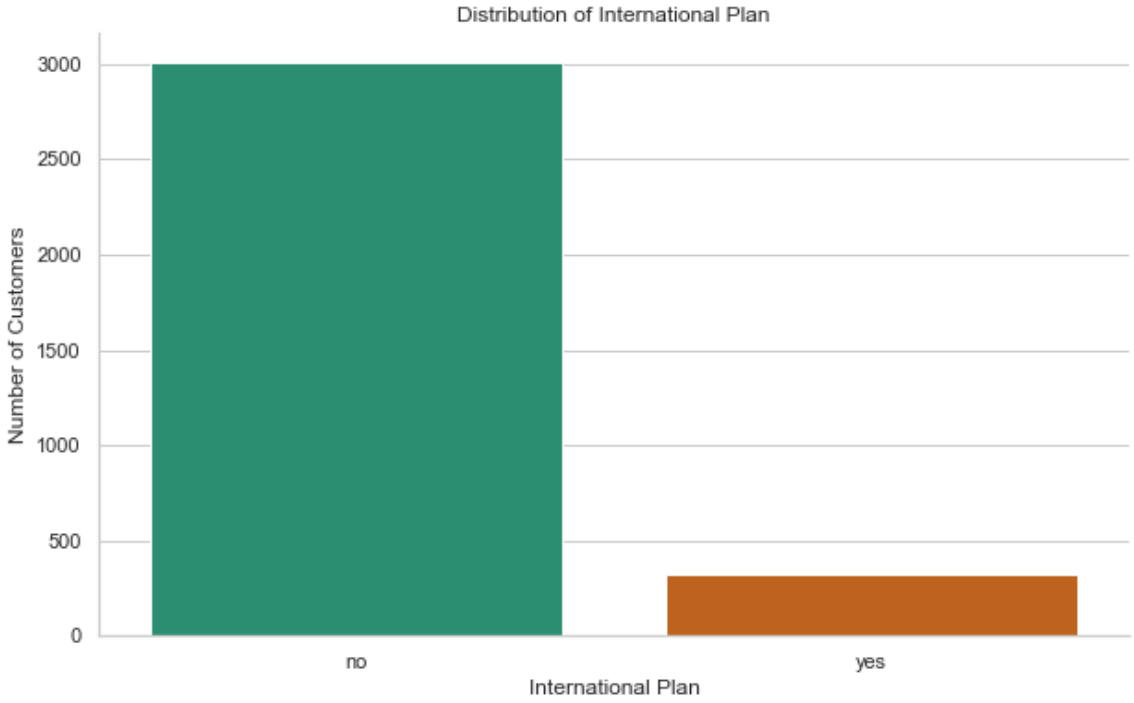
In [88]:

```
#area code

# Count the number of customers in each area code and sort the values
area_code_counts = df['area code'].value_counts().sort_values(ascending=False)

# Set the style and palette for the plot
sns.set(style="whitegrid")
palette = sns.color_palette("Dark2")

# Create a bar plot for area codes
plt.figure(figsize=(10, 6))
sns.barplot(x=area_code_counts.index, y=area_code_counts.values, palette=palette)

# Add titles and labels
plt.title('Number of Customers by Area Code')
plt.xlabel('Area Code')
plt.ylabel('Number of Customers')

# Hide top and right spines
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

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

415 area code has the highest number of customers whereas areas codes 408 and 510 have similar number of customers. This distribution of customers across area codes suggests potential differences in markets or marketing priorities.

In [89]:

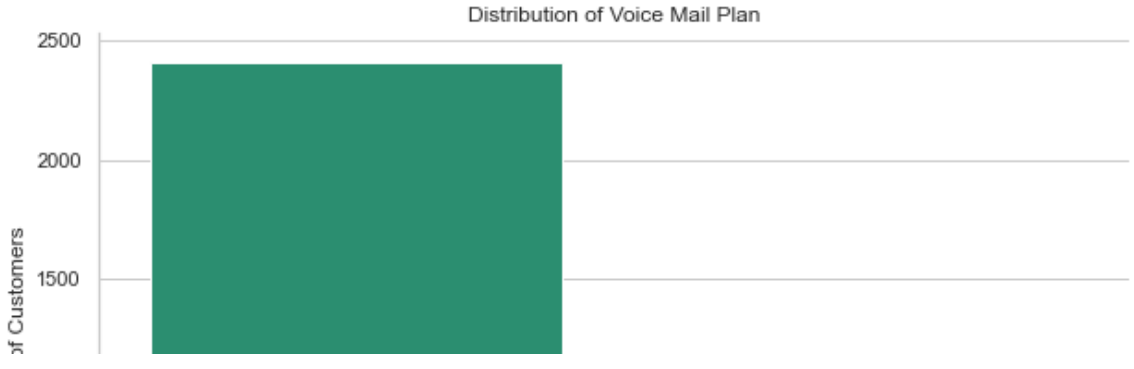
```
# Plot for international plan
international_plan_counts = df['international plan'].value_counts()
plt.figure(figsize=(10, 6))
sns.barplot(x=international_plan_counts.index, y=international_plan_counts.values, palet
te=palette)
plt.title('Distribution of International Plan')
plt.xlabel('International Plan')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

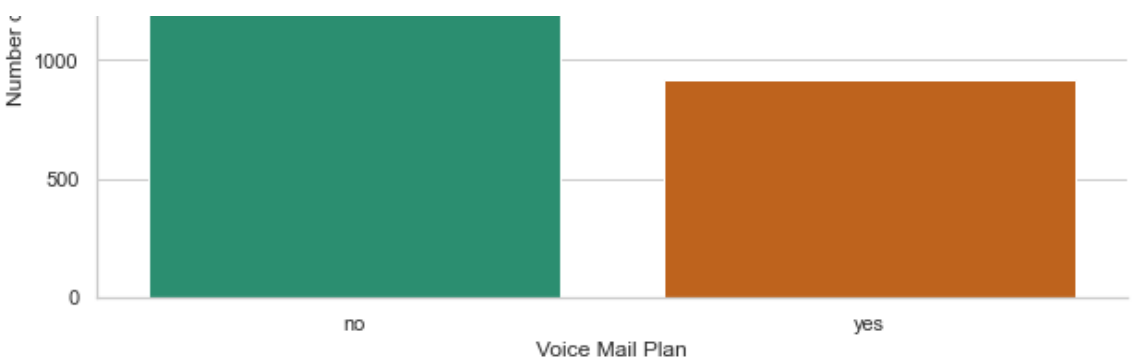


A majority of the customers do not have an international plan.

In [90]:

```
# Plot for voice mail plan
voice_mail_plan_counts = df['voice mail plan'].value_counts()
plt.figure(figsize=(10, 6))
sns.barplot(x=voice_mail_plan_counts.index, y=voice_mail_plan_counts.values, palette=pal
ette)
plt.title('Distribution of Voice Mail Plan')
plt.xlabel('Voice Mail Plan')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

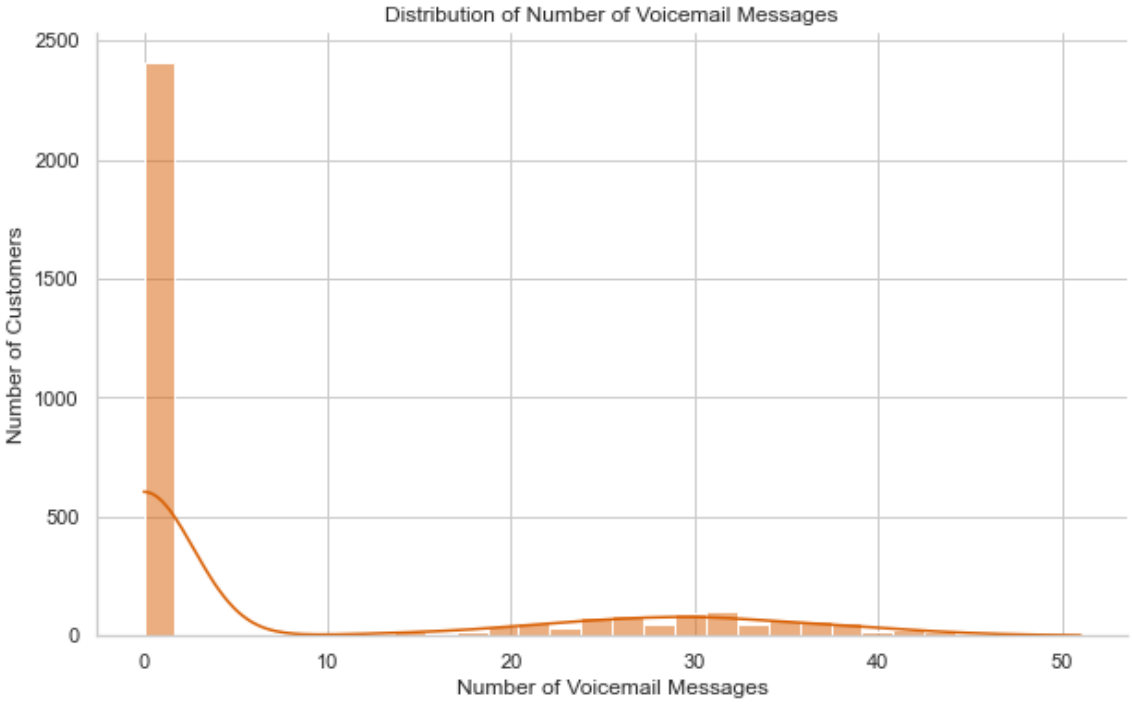




The bar plot visualizes the distribution of customers based on whether they have a voice mail plan or not. It shows that majority of the customers do not have a Voice mail plan.

In [91]:

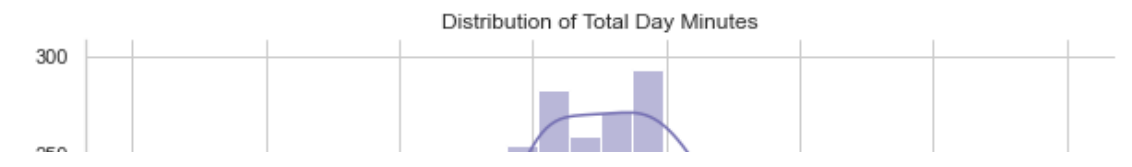
```
# Plot for number vmail messages
plt.figure(figsize=(10, 6))
sns.histplot(df['number vmail messages'], bins=30, kde=True, color=palette[1])
plt.title('Distribution of Number of Voicemail Messages')
plt.xlabel('Number of Voicemail Messages')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

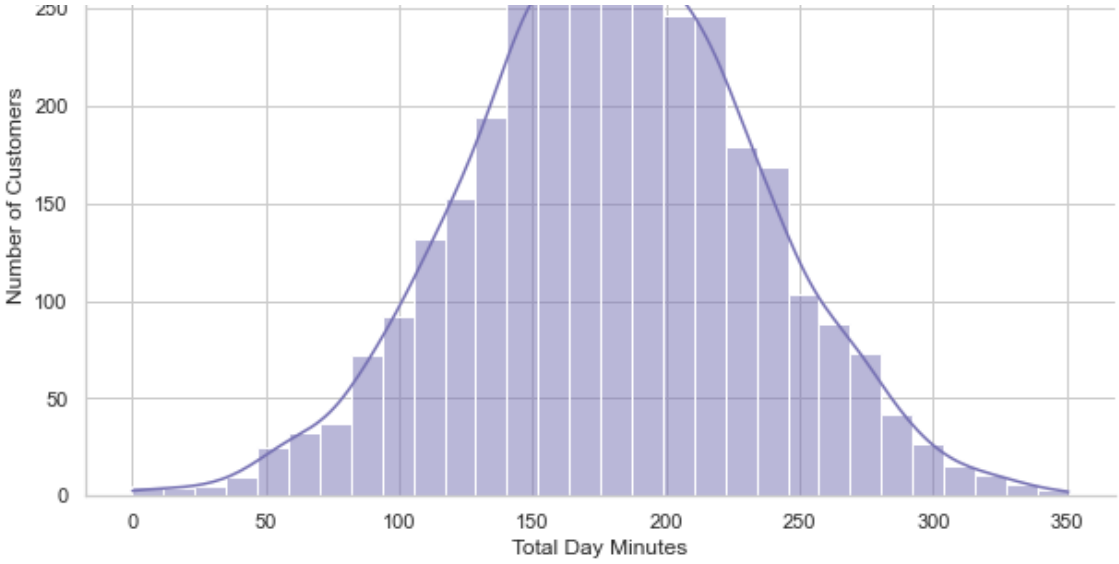


Since most of the customers do not have a voicemail plan, it explains why the number of voice messages are 0.

In [92]:

```
# Plot for total day minutes
plt.figure(figsize=(10, 6))
sns.histplot(df['total day minutes'], bins=30, kde=True, color=palette[2])
plt.title('Distribution of Total Day Minutes')
plt.xlabel('Total Day Minutes')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

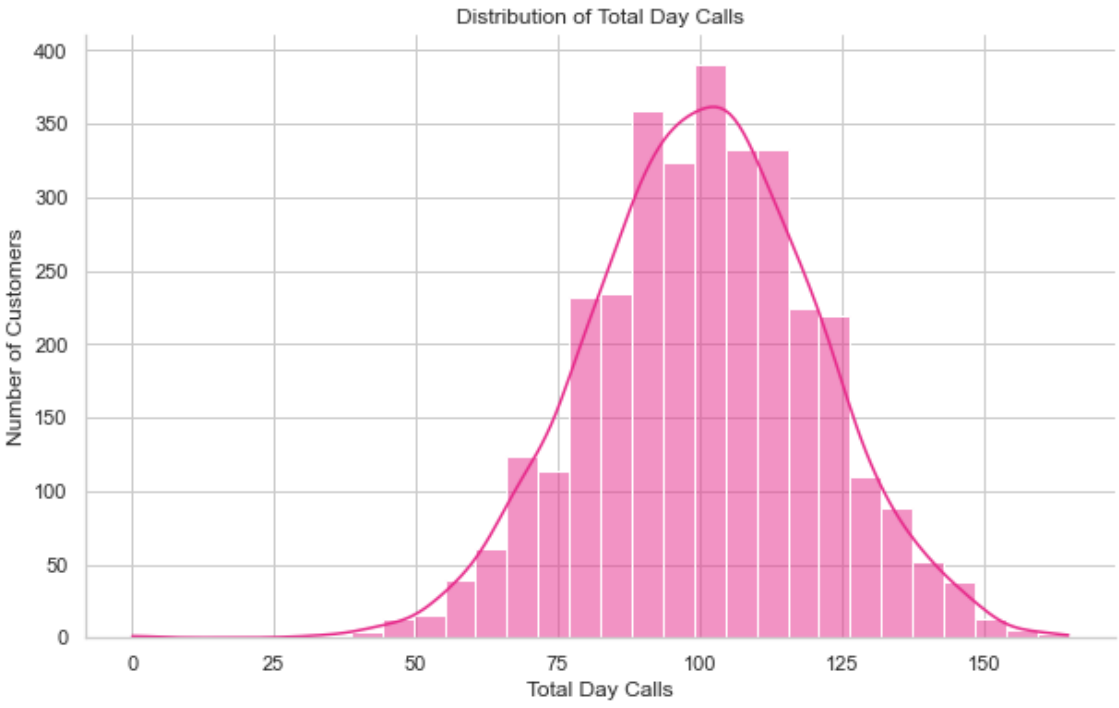




The highest total day minutes observed in the dataset is 190.The second highest total day minutes observed is 160. The histogram of the total day minutes column appears to be normally distributed.

In [93]:

```
# Plot for total day calls
plt.figure(figsize=(10, 6))
sns.histplot(df['total day calls'], bins=30, kde=True, color=palette[3])
plt.title('Distribution of Total Day Calls')
plt.xlabel('Total Day Calls')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

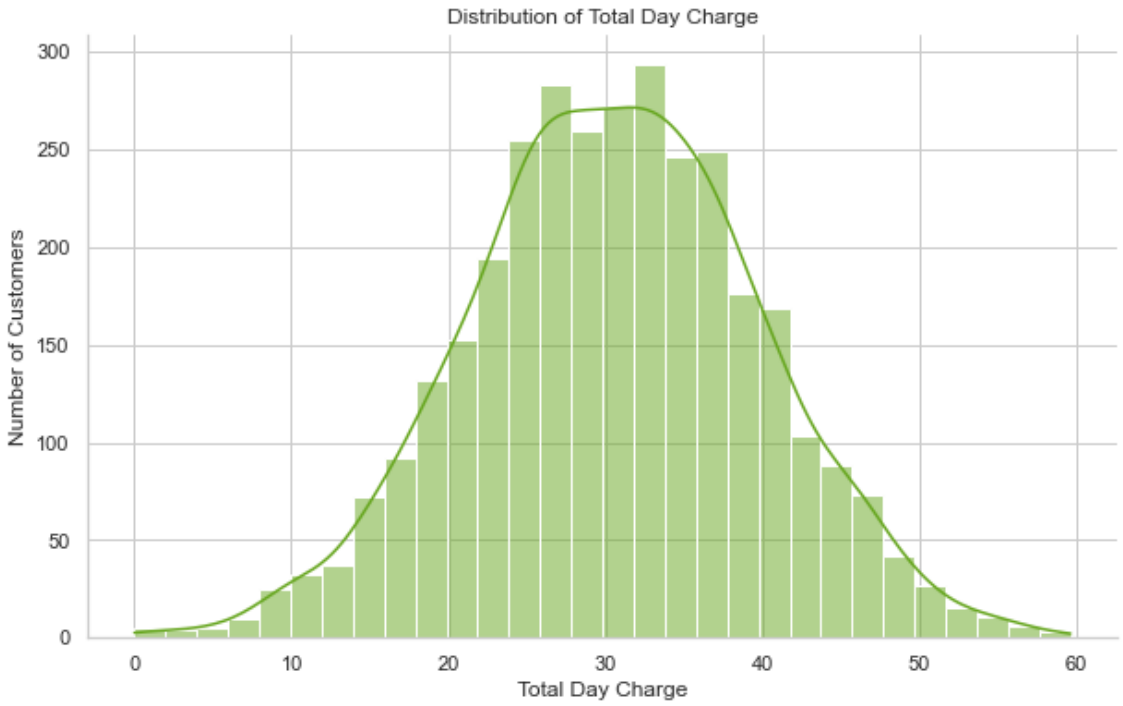


The highest number of total day calls observed is 105, which is common among the majority of customers. The second-highest count is 360 total day calls.

In [94]:

```
# Plot for total day charge
plt.figure(figsize=(10, 6))
sns.histplot(df['total day charge'], bins=30, kde=True, color=palette[4])
plt.title('Distribution of Total Day Charge')
plt.xlabel('Total Day Charge')
plt.ylabel('Number of Customers')
```

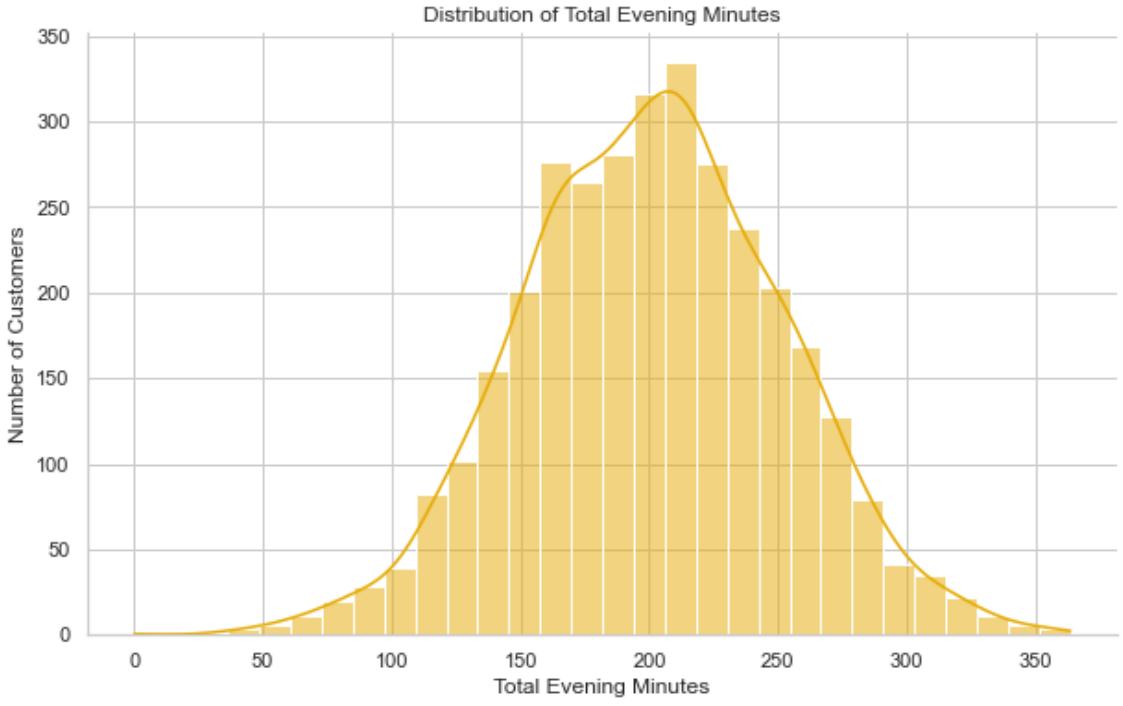
```
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



The highest total day charge recorded is around 32, prevalent among most customers, while the second-highest is approximately 26.

In [95]:

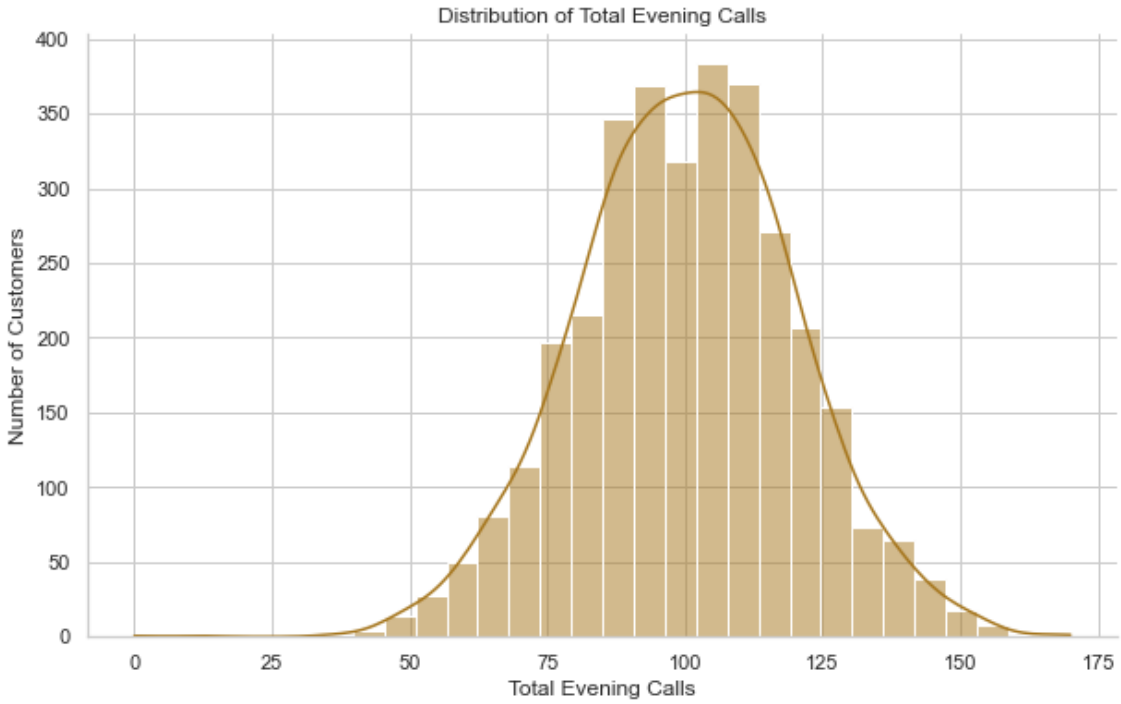
```
# Plot for total eve minutes
plt.figure(figsize=(10, 6))
sns.histplot(df['total eve minutes'], bins=30, kde=True, color=palette[5])
plt.title('Distribution of Total Evening Minutes')
plt.xlabel('Total Evening Minutes')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



The highest total evening minutes recorded are around 205, commonly observed among the majority of customers. The second-highest count is 200 total evening minutes.

In [96]:

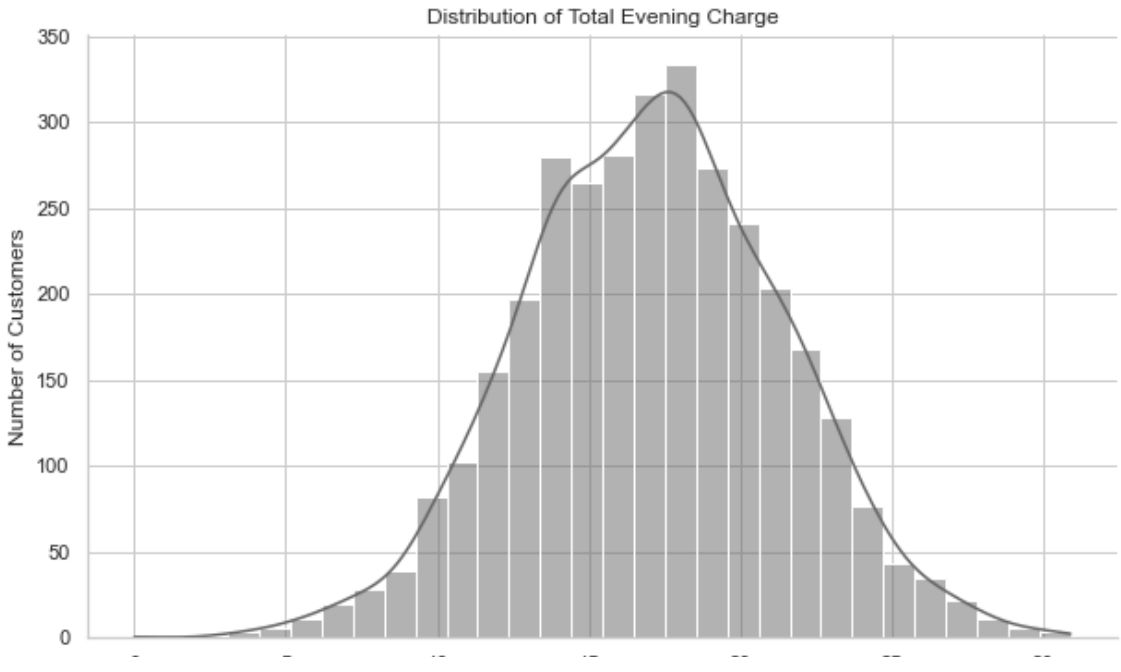
```
# Plot for total eve calls
plt.figure(figsize=(10, 6))
sns.histplot(df['total eve calls'], bins=30, kde=True, color=palette[6])
plt.title('Distribution of Total Evening Calls')
plt.xlabel('Total Evening Calls')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



The highest number of total evening calls observed is approximately between 90- 110.

In [97]:

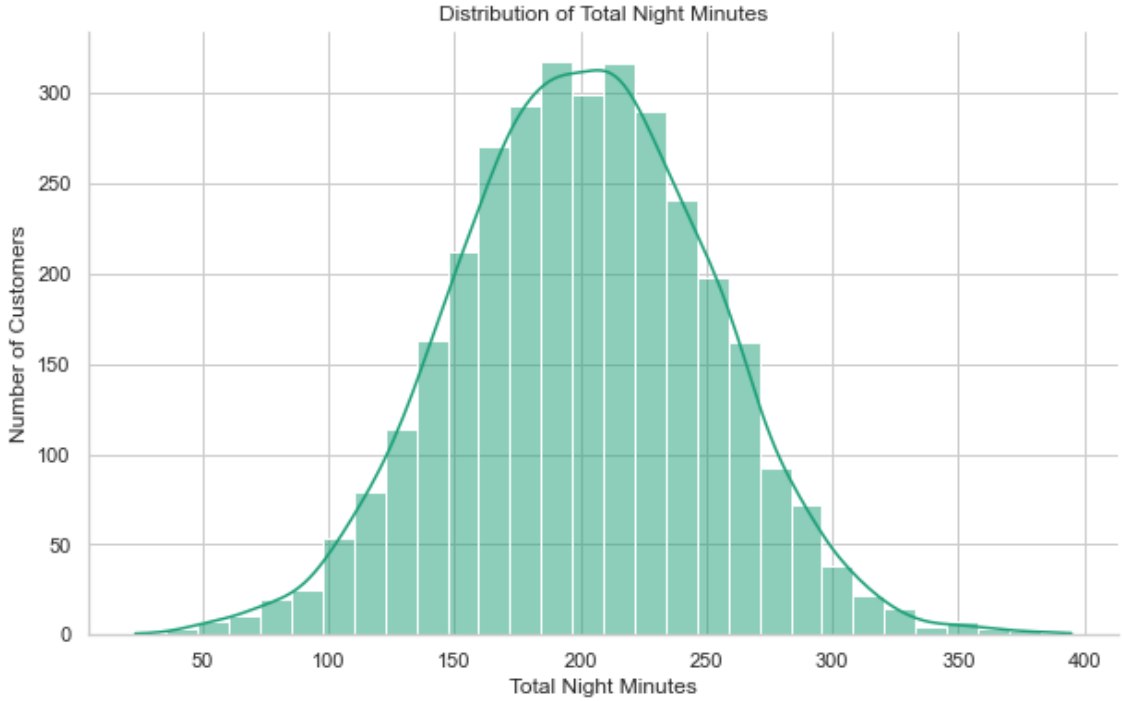
```
# Plot for total eve charge
plt.figure(figsize=(10, 6))
sns.histplot(df['total eve charge'], bins=30, kde=True, color=palette[7])
plt.title('Distribution of Total Evening Charge')
plt.xlabel('Total Evening Charge')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



The highest total evening charge recorded by most customers is between 17-18.

In [98]:

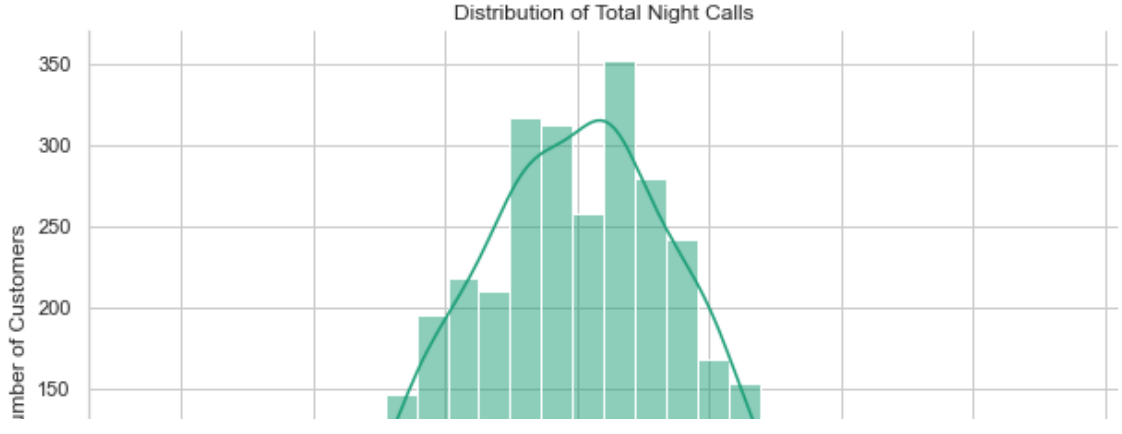
```
# Plot for total night minutes
plt.figure(figsize=(10, 6))
sns.histplot(df['total night minutes'], bins=30, kde=True, color=palette[0])  # Using th
e first color from the palette
plt.title('Distribution of Total Night Minutes')
plt.xlabel('Total Night Minutes')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

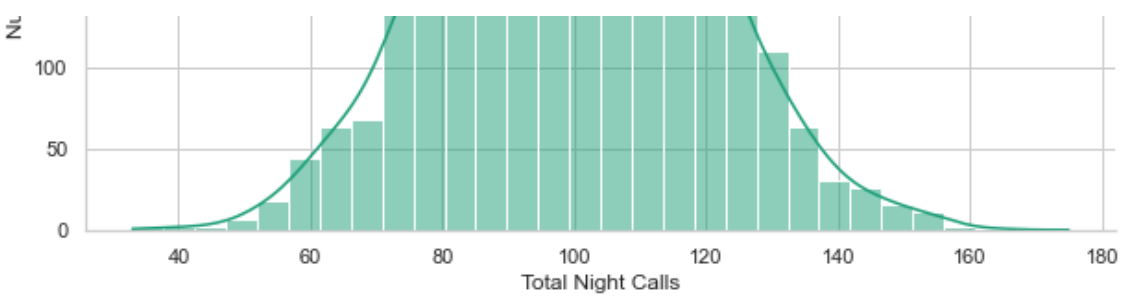


Among the majority of customers, the total night minutes typically fall within the range of 180 to 210.

In [99]:

```
# Plot for total night calls
plt.figure(figsize=(10, 6))
sns.histplot(df['total night calls'], bins=30, kde=True, color=palette[0])  # Using the
first color from the palette
plt.title('Distribution of Total Night Calls')
plt.xlabel('Total Night Calls')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

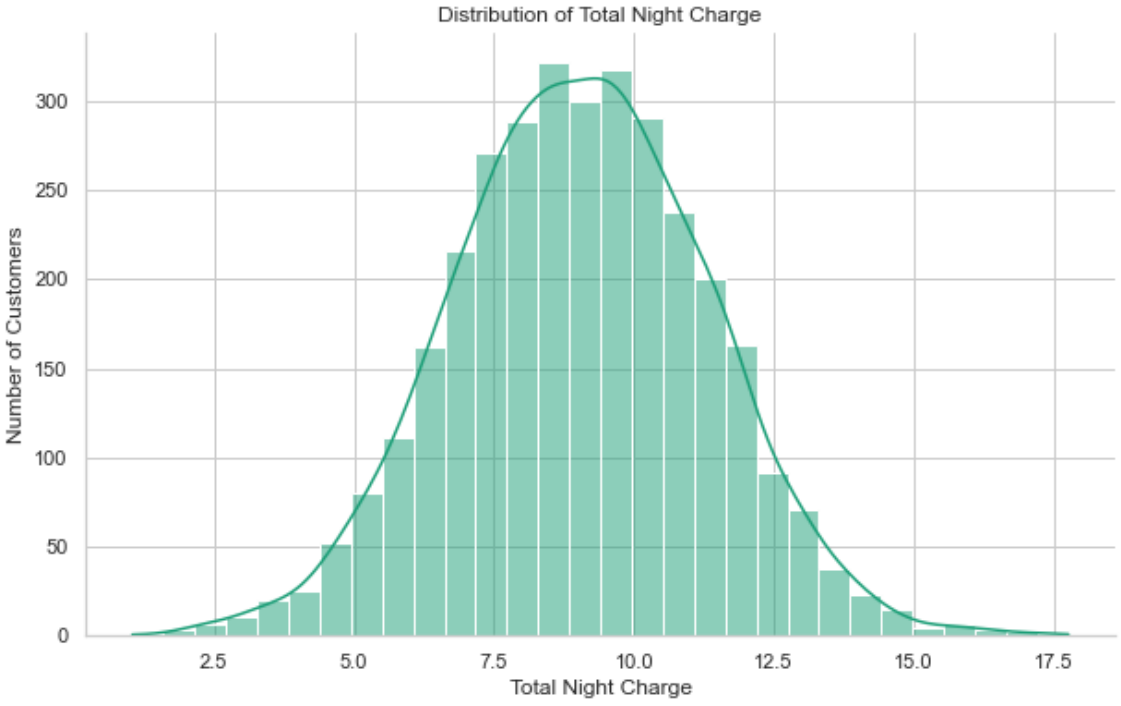




The predominant number of total night calls made by customers is approximately 110.

In [100]:

```
# Plot for total night charge
plt.figure(figsize=(10, 6))
sns.histplot(df['total night charge'], bins=30, kde=True, color=palette[0])
plt.title('Distribution of Total Night Charge')
plt.xlabel('Total Night Charge')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



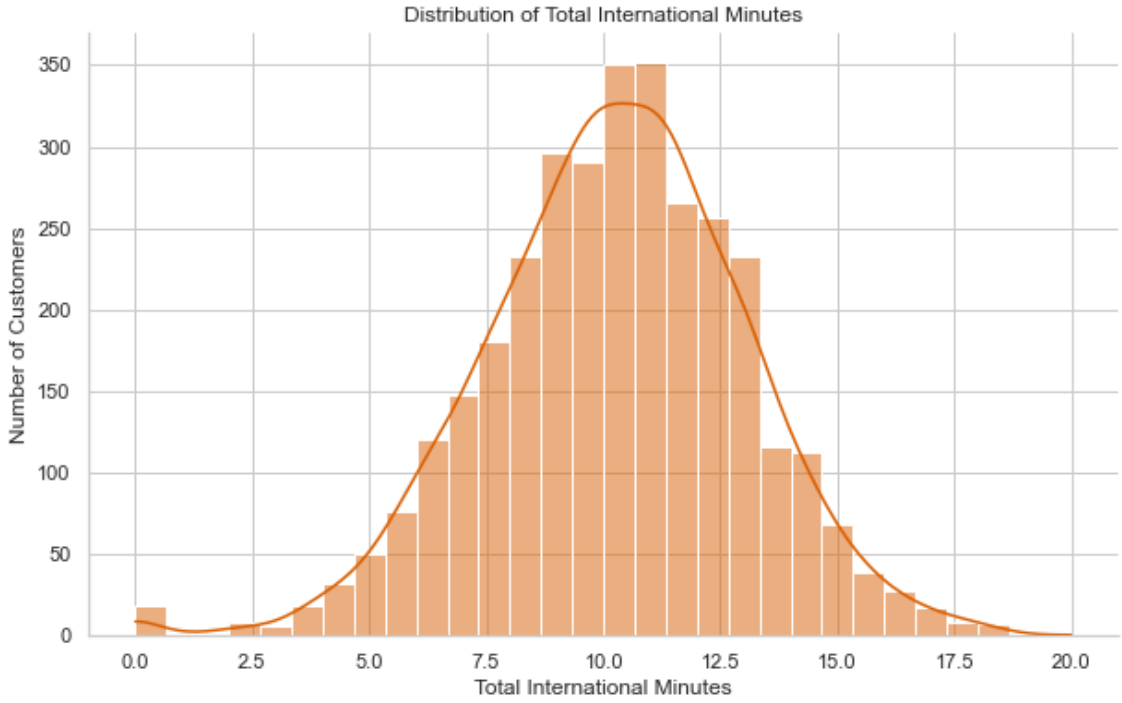
The majority of customers incur night charges ranging between 7.7 and 7.9.

Major comparisons between day,night and evening show: i) Usage Intensity: The analysis suggests that usage intensity varies across different time periods. While the highest number of total day calls is common among the majority of customers, the number of evening and night calls shows a more consistent pattern.

ii) Charge Patterns: Although the highest total day charge is relatively high, the highest total evening and night charges are comparatively lower, indicating potential differences in pricing or usage patterns across these time periods.

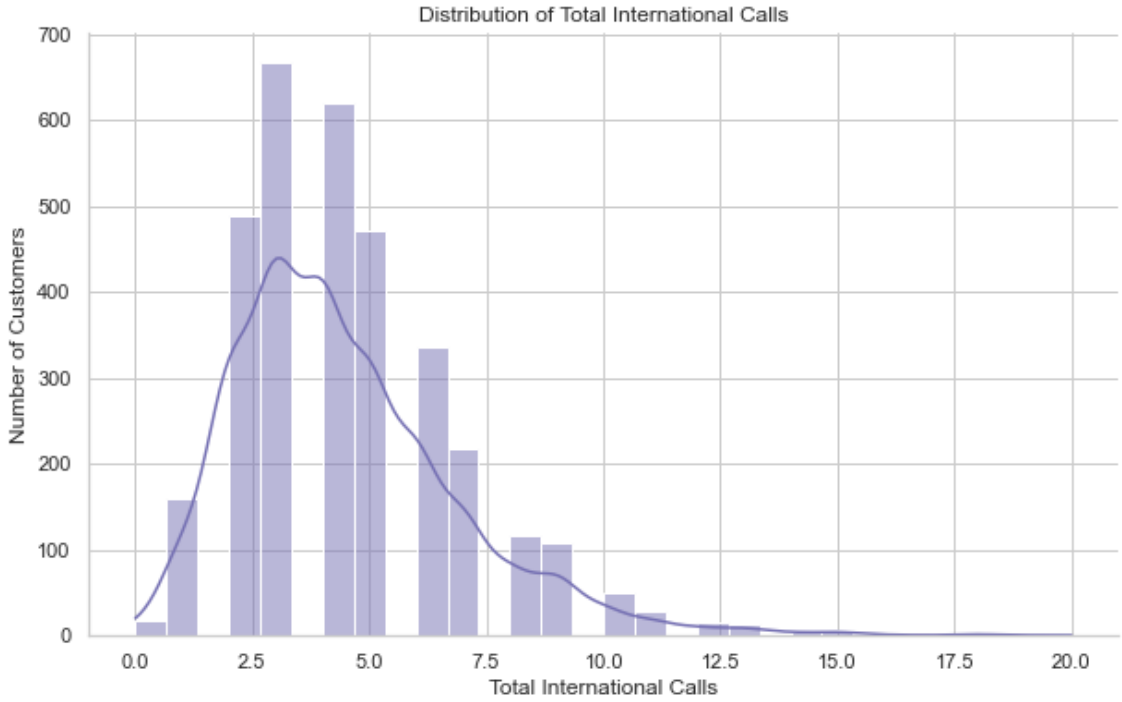
In [101]:

```
# Plot for total intl minutes
plt.figure(figsize=(10, 6))
sns.histplot(df['total intl minutes'], bins=30, kde=True, color=palette[1])
plt.title('Distribution of Total International Minutes')
plt.xlabel('Total International Minutes')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



In [102]:

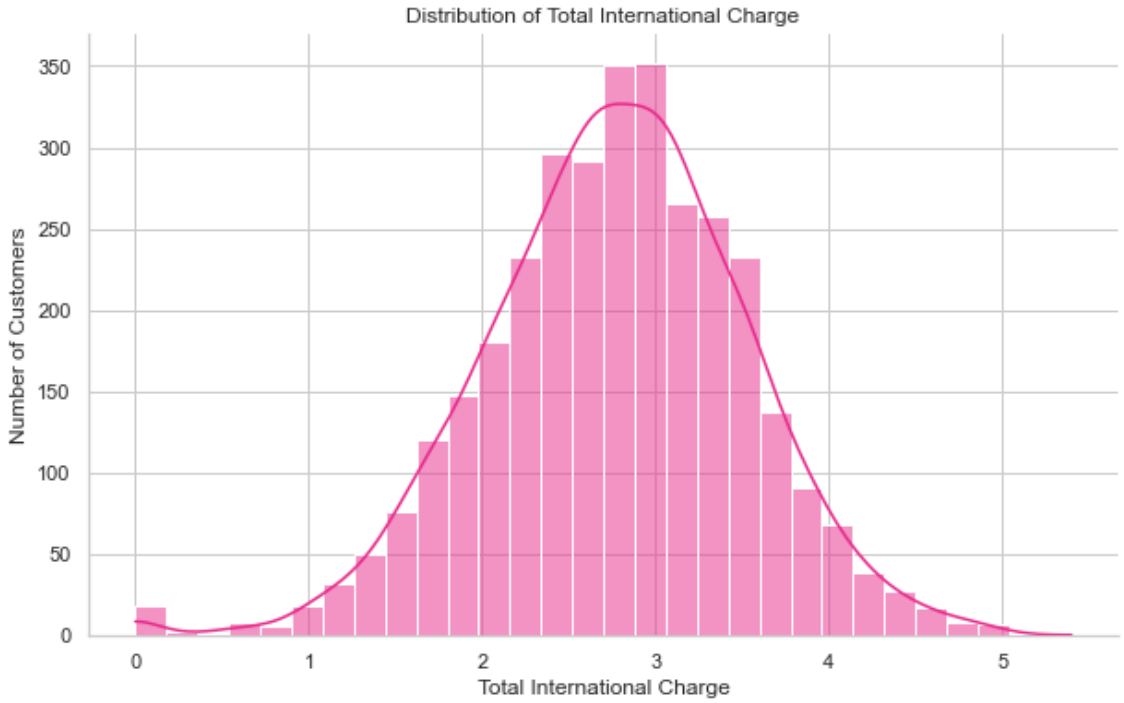
```
# Plot for total intl calls
plt.figure(figsize=(10, 6))
sns.histplot(df['total intl calls'], bins=30, kde=True, color=palette[2])
plt.title('Distribution of Total International Calls')
plt.xlabel('Total International Calls')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



In [103]:

```
# Plot for total intl charge
plt.figure(figsize=(10, 6))
sns.histplot(df['total intl charge'], bins=30, kde=True, color=palette[3])
plt.title('Distribution of Total International Charge')
plt.xlabel('Total International Charge')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
```

```
plt.show()
```



In [104]:

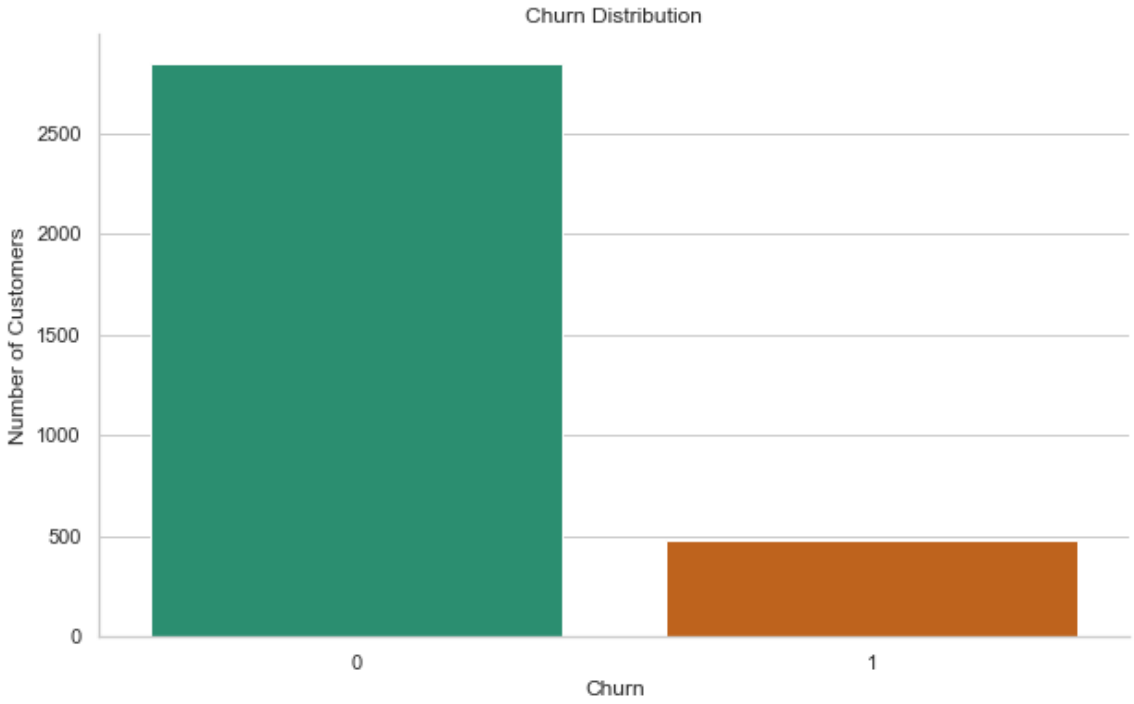
```
# Plot for customer service calls
plt.figure(figsize=(10, 6))
sns.histplot(df['customer service calls'], bins=30, kde=True, color=palette[4])
plt.title('Distribution of Customer Service Calls')
plt.xlabel('Customer Service Calls')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

Most customers have made at least one call to customer service, with the second-highest count being two calls.

In [105]:

```
# Plot for churn
churn_counts = df['churn'].value_counts()
plt.figure(figsize=(10, 6))
sns.barplot(x=churn_counts.index, y=churn_counts.values, palette=palette)
plt.title('Churn Distribution')
```

```
plt.xlabel('Churn')
plt.ylabel('Number of Customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



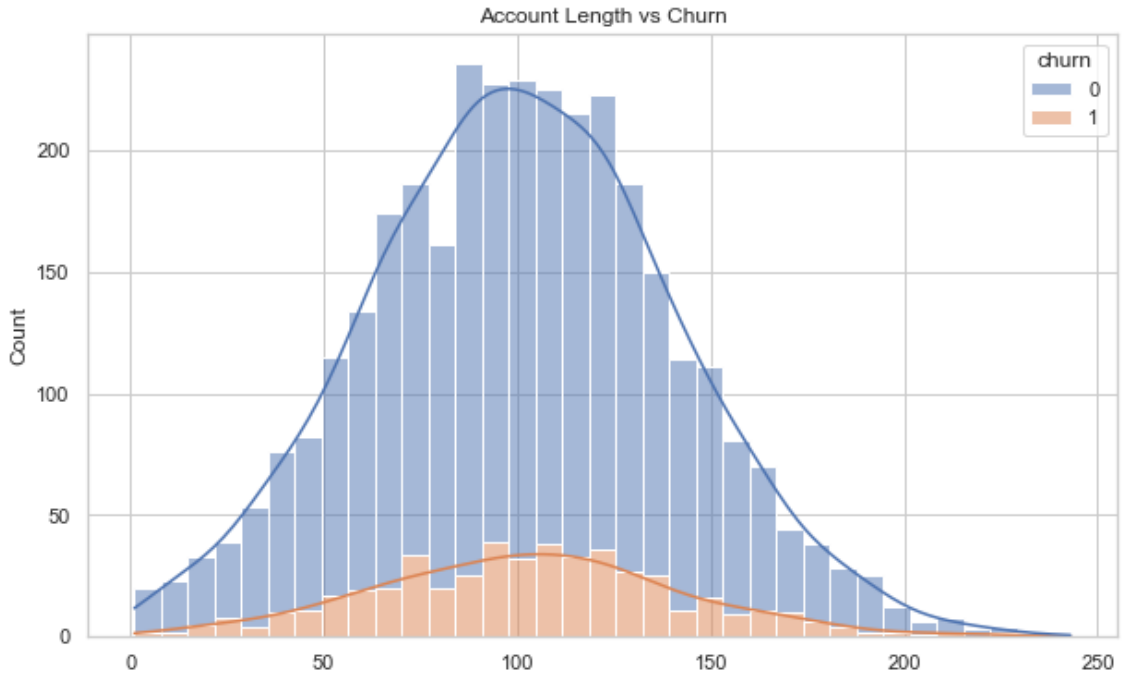
As shown by the bar plot, a majority of the customers were not churened, but approximates 400 were churned.

Overall, the analysis of the telecom churn dataset provides valuable insights into customer behavior, service usage patterns, and churn within the telecom company's customer base. The findings indicate that while the majority of customers exhibit consistent usage patterns within certain ranges, there are variations in usage levels and interaction with customer service.

Bivariate Analysis

In [106]:

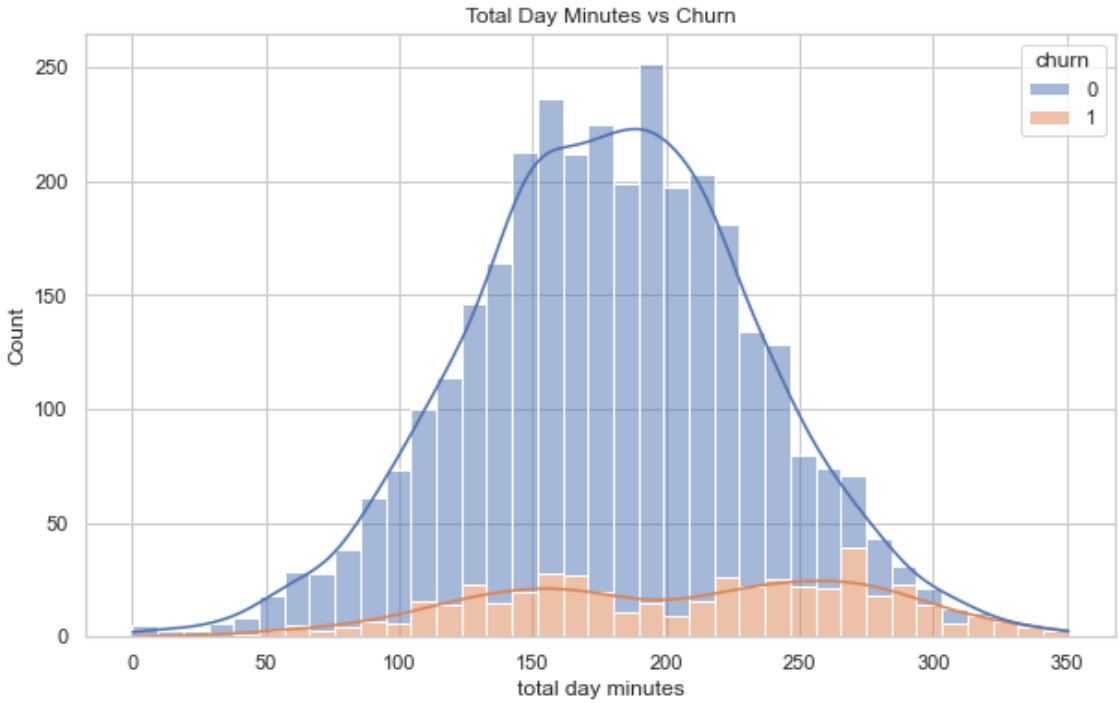
```
# Account Length vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='account length', hue='churn', multiple='stack', kde=True)
plt.title('Account Length vs Churn')
plt.show()
```



account length

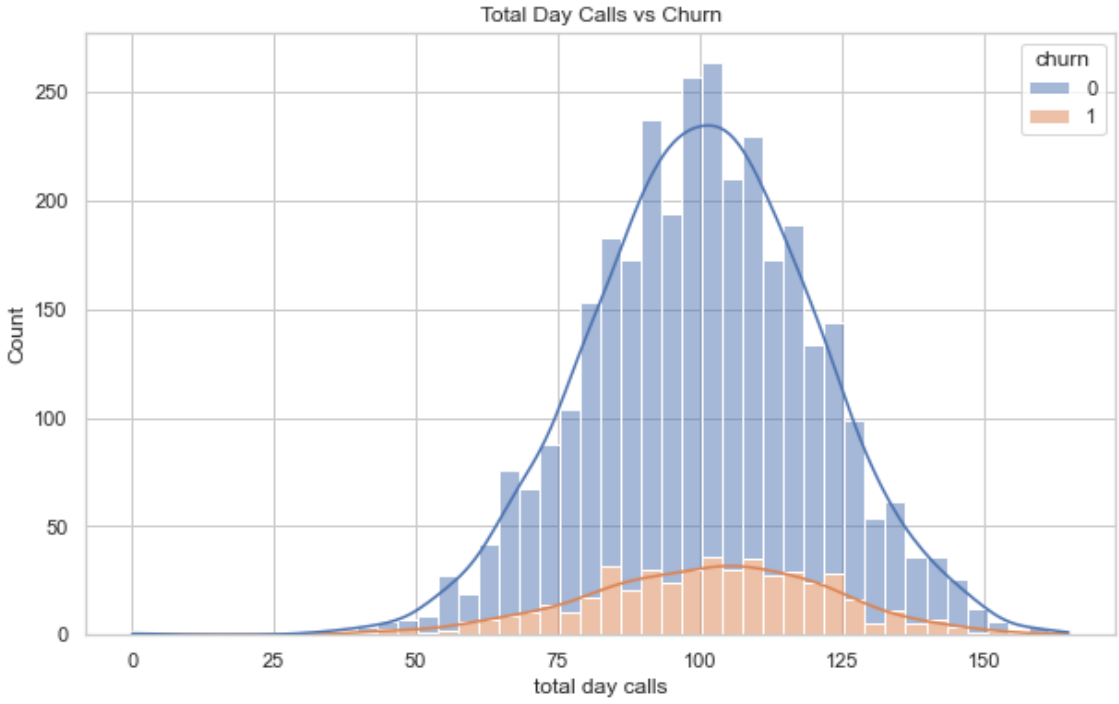
In [107]:

```
# Total Day Minutes vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total day minutes', hue='churn', multiple='stack', kde=True)
plt.title('Total Day Minutes vs Churn')
plt.show()
```



In [108]:

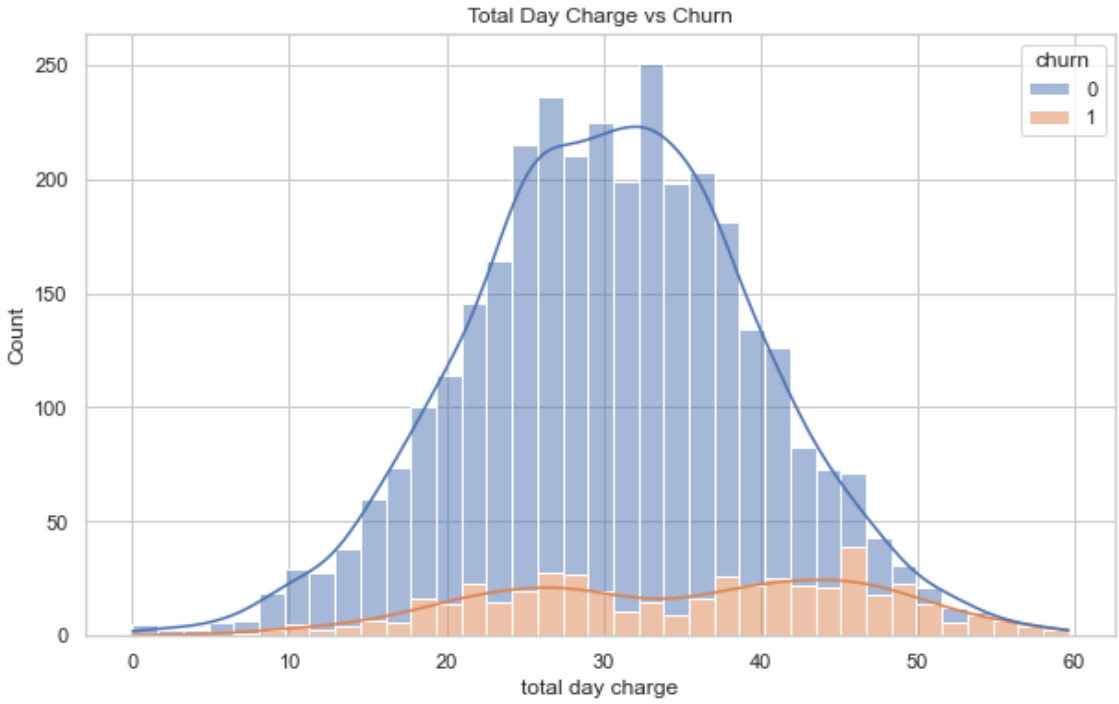
```
# Total Day Calls vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total day calls', hue='churn', multiple='stack', kde=True)
plt.title('Total Day Calls vs Churn')
plt.show()
```



In [109]:

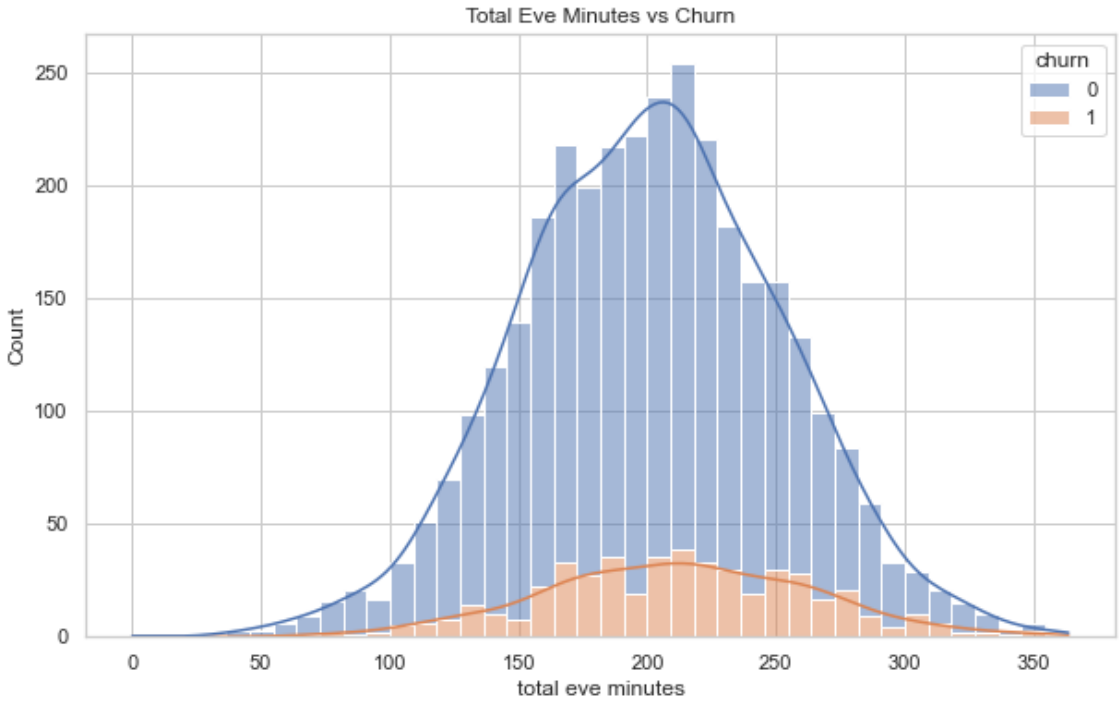
```
# Total Day Charge vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total day charge', hue='churn', multiple='stack', kde=True)
```

```
plt.title('Total Day Charge vs Churn')
plt.show()
```



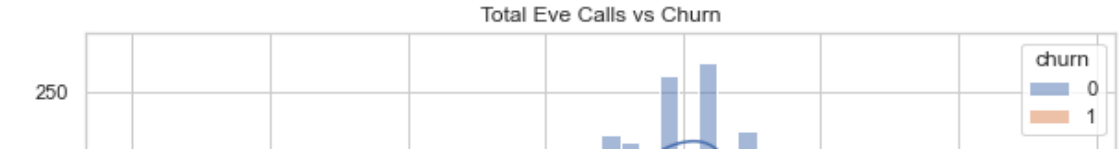
In [110]:

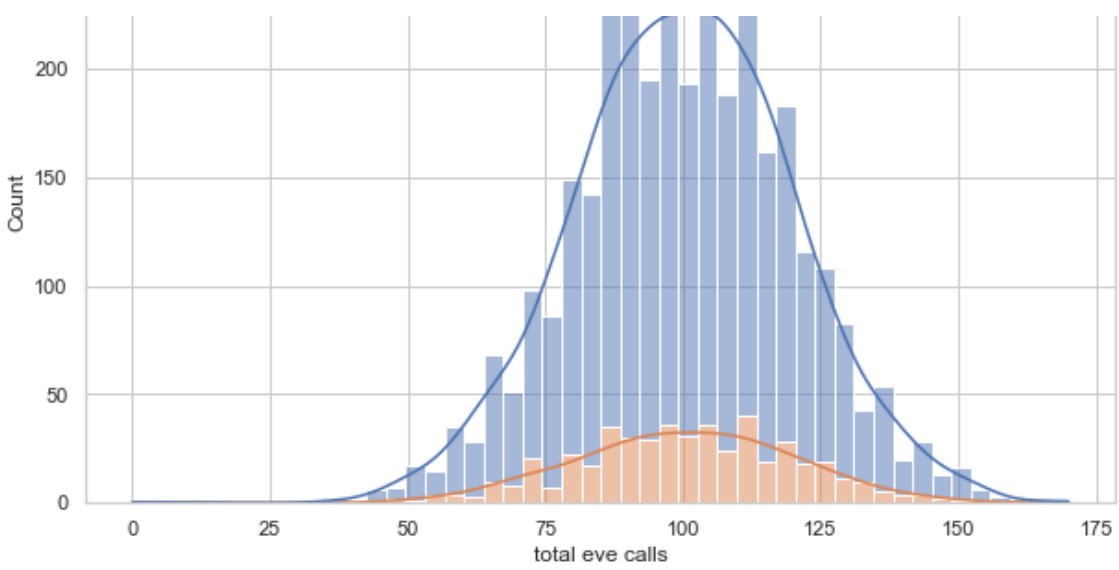
```
# Total Eve Minutes vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total eve minutes', hue='churn', multiple='stack', kde=True)
plt.title('Total Eve Minutes vs Churn')
plt.show()
```



In [111]:

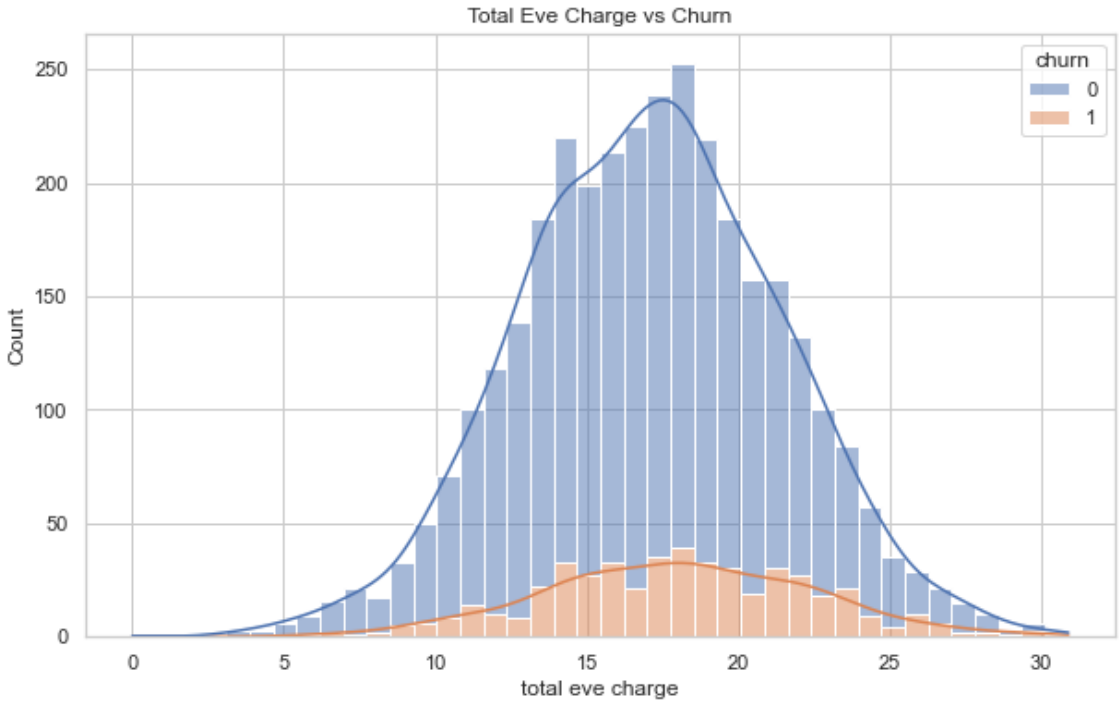
```
# Total Eve Calls vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total eve calls', hue='churn', multiple='stack', kde=True)
plt.title('Total Eve Calls vs Churn')
plt.show()
```





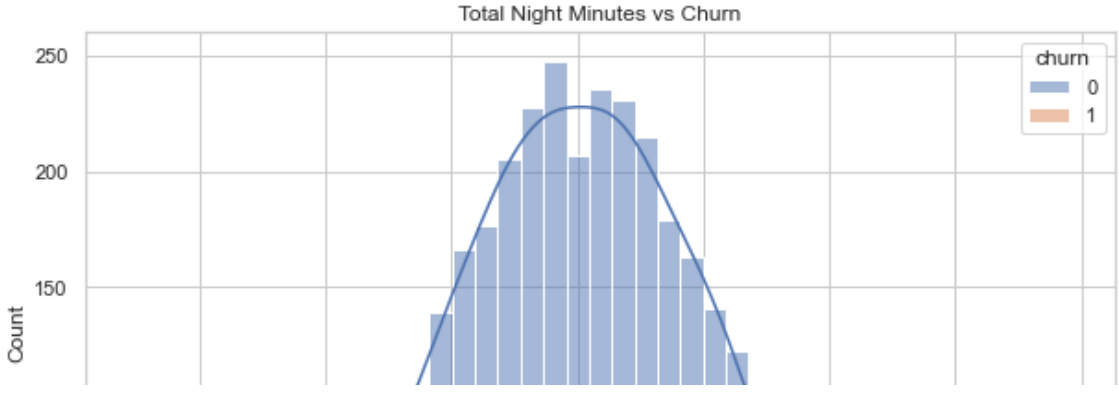
In [112]:

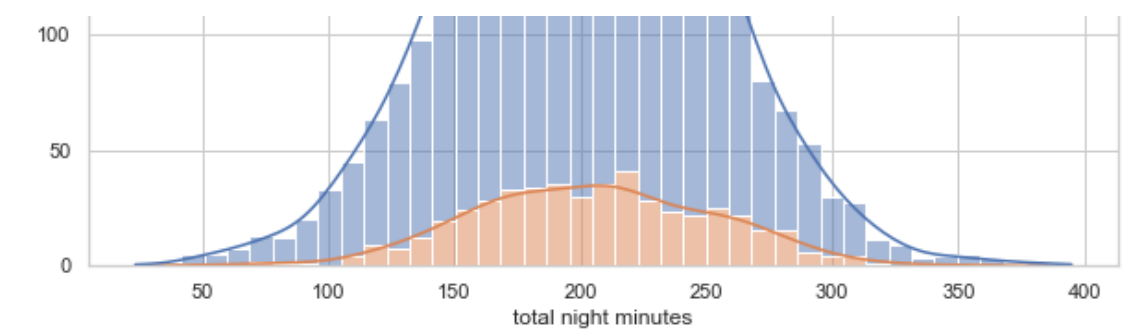
```
# Total Eve Charge vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total eve charge', hue='churn', multiple='stack', kde=True)
plt.title('Total Eve Charge vs Churn')
plt.show()
```



In [113]:

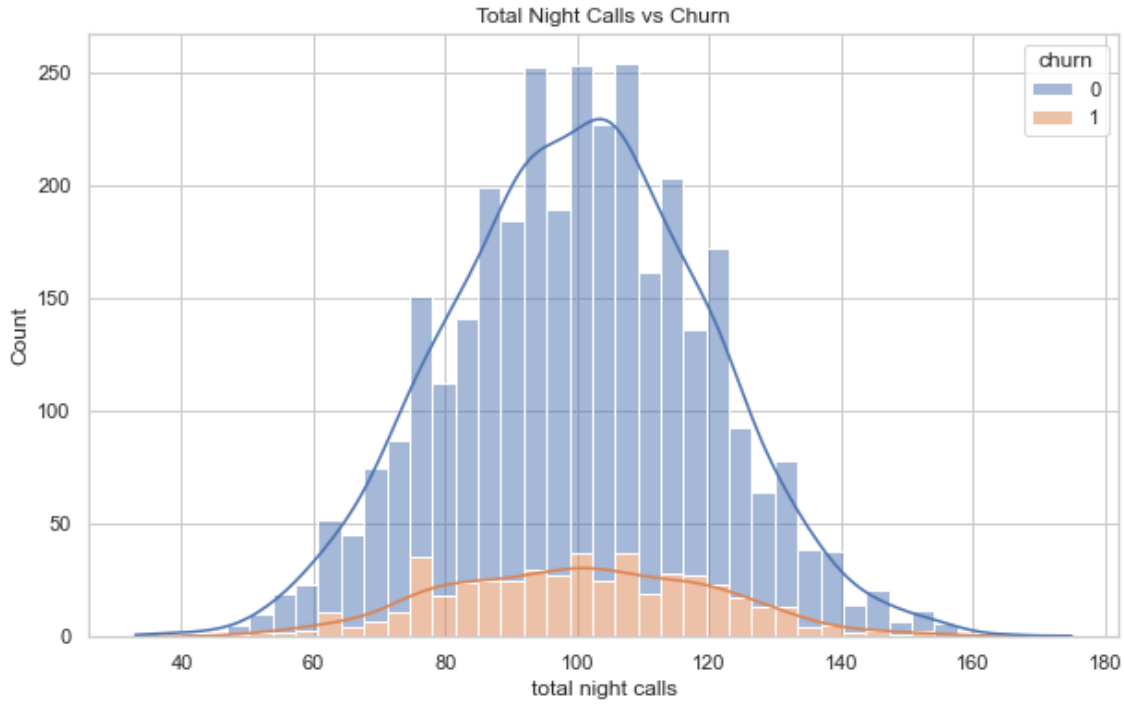
```
# Total Night Minutes vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total night minutes', hue='churn', multiple='stack', kde=True)
plt.title('Total Night Minutes vs Churn')
plt.show()
```





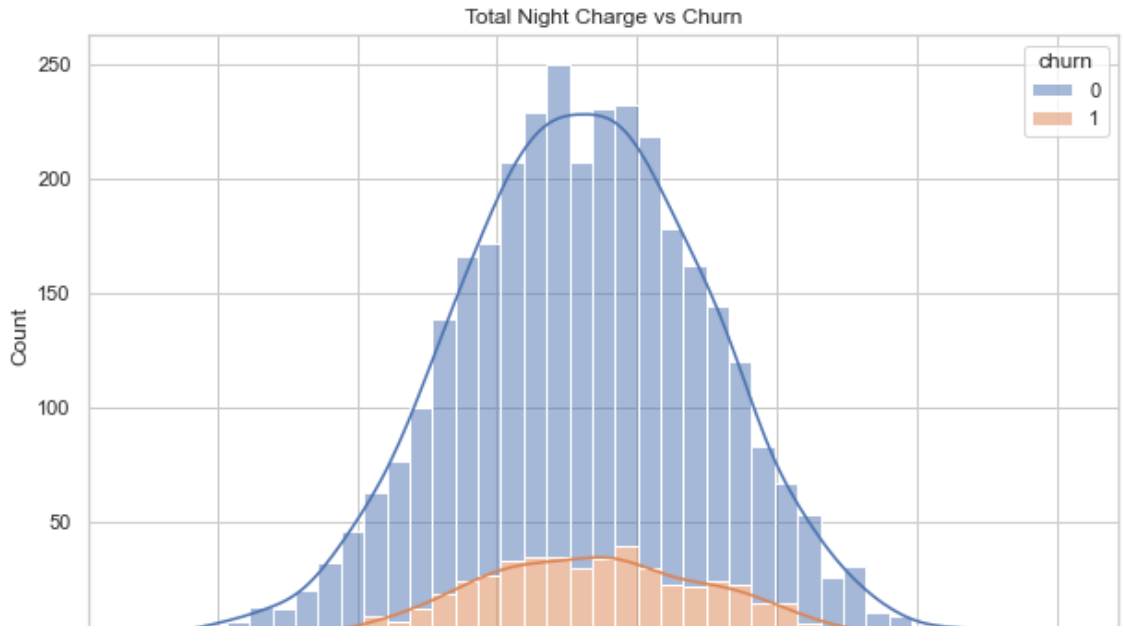
In [114]:

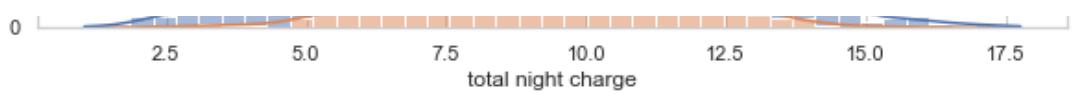
```
# Total Night Calls vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total night calls', hue='churn', multiple='stack', kde=True)
plt.title('Total Night Calls vs Churn')
plt.show()
```



In [115]:

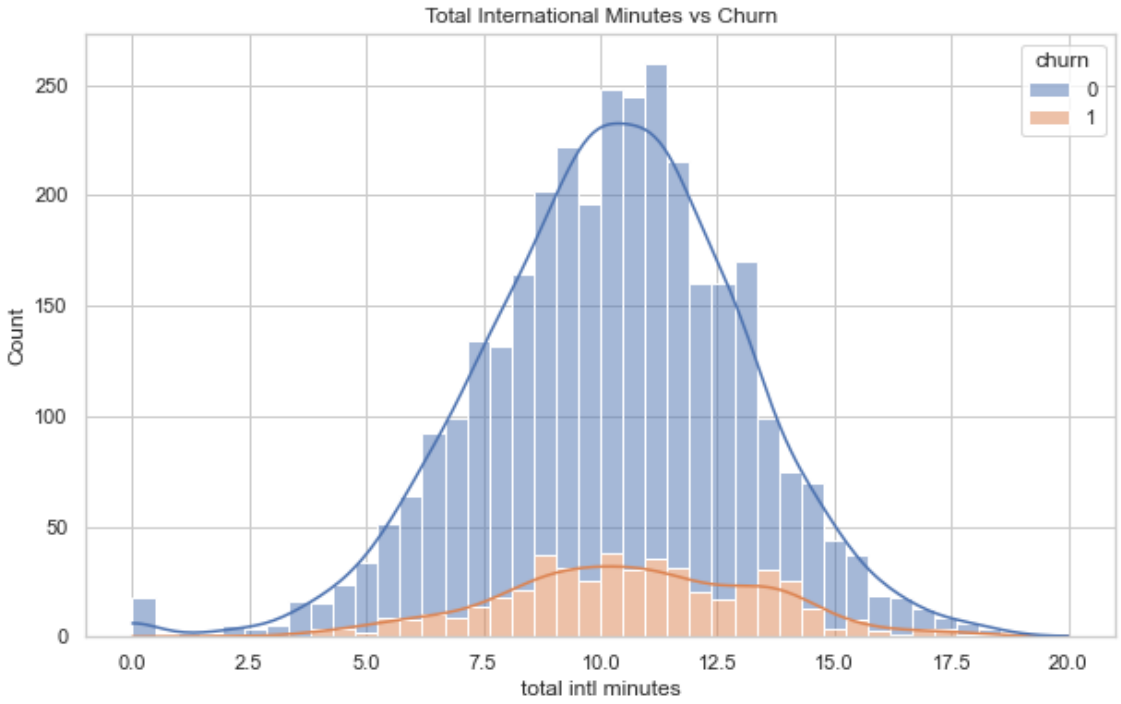
```
# Total Night Charge vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total night charge', hue='churn', multiple='stack', kde=True)
plt.title('Total Night Charge vs Churn')
plt.show()
```





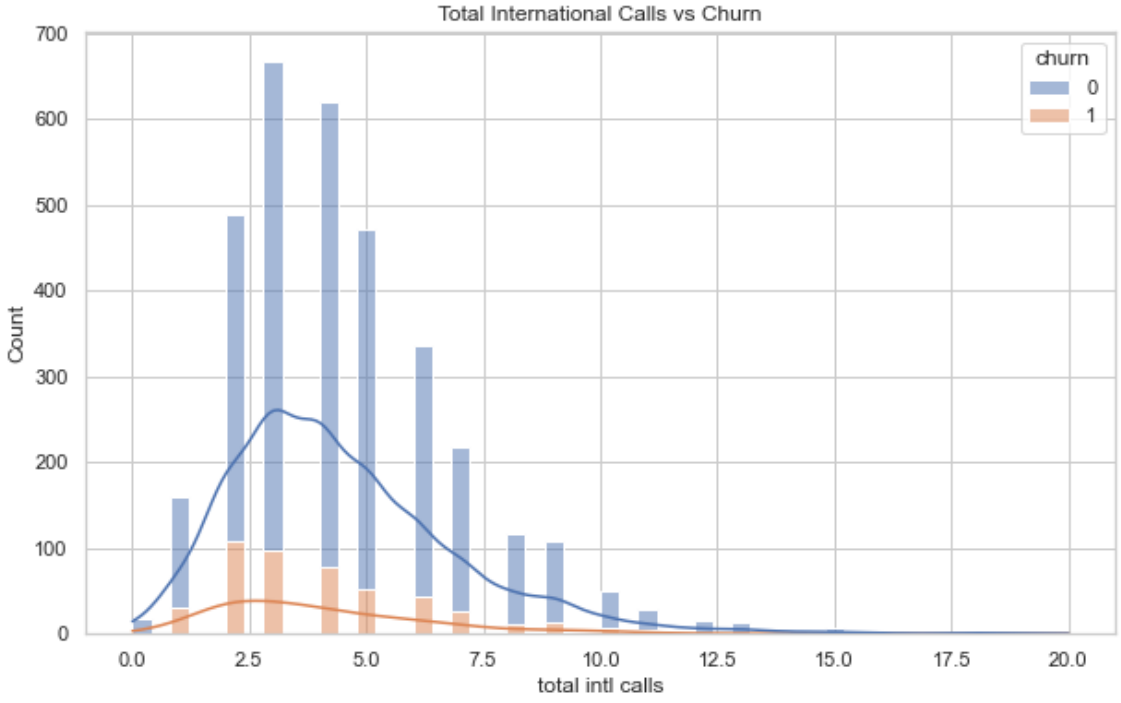
In [116]:

```
# Total Intl Minutes vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total intl minutes', hue='churn', multiple='stack', kde=True)
plt.title('Total International Minutes vs Churn')
plt.show()
```



In [117]:

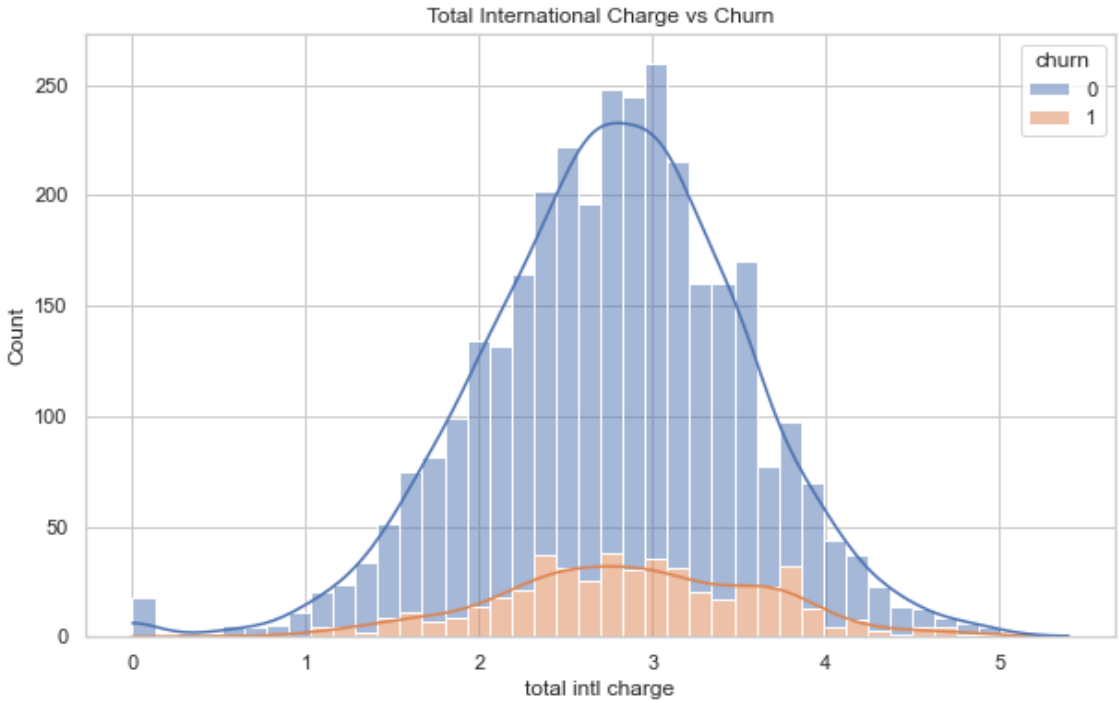
```
# Total Intl Calls vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total intl calls', hue='churn', multiple='stack', kde=True)
plt.title('Total International Calls vs Churn')
plt.show()
```



In [118]:

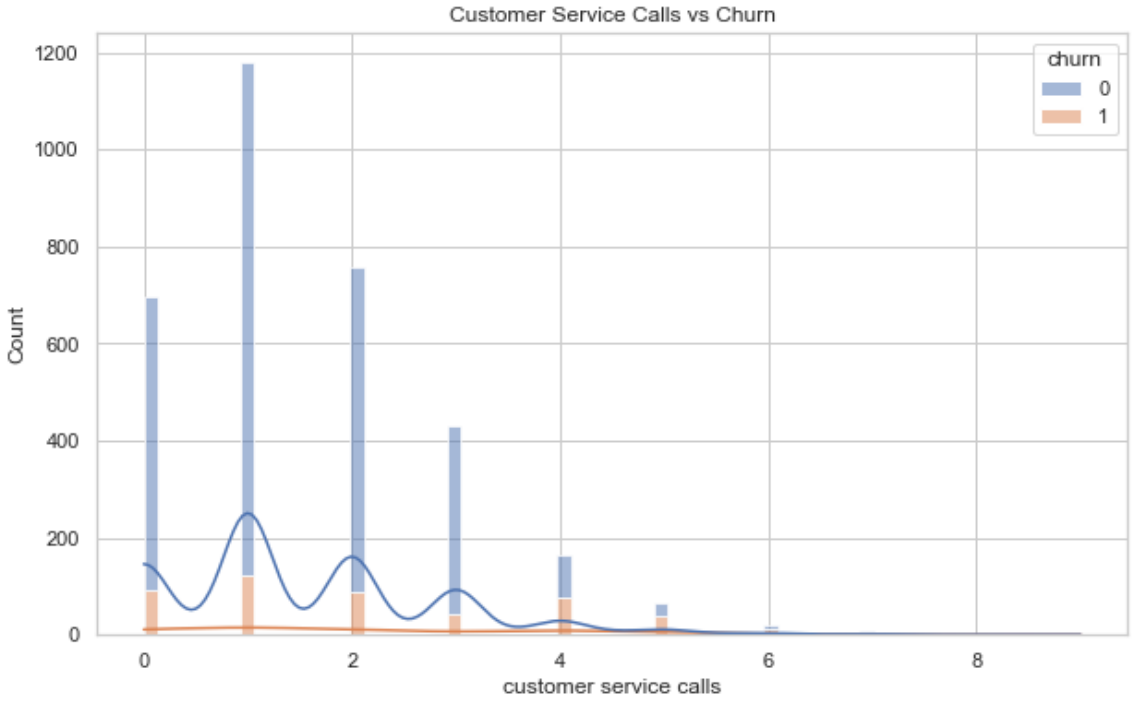
```
# Total Intl Charge vs Churn
```

```
plt.figure(figsize=(10, 6))
sns.histplot(df, x='total intl charge', hue='churn', multiple='stack', kde=True)
plt.title('Total International Charge vs Churn')
plt.show()
```



In [119]:

```
# Customer Service Calls vs Churn
plt.figure(figsize=(10, 6))
sns.histplot(df, x='customer service calls', hue='churn', multiple='stack', kde=True)
plt.title('Customer Service Calls vs Churn')
plt.show()
```

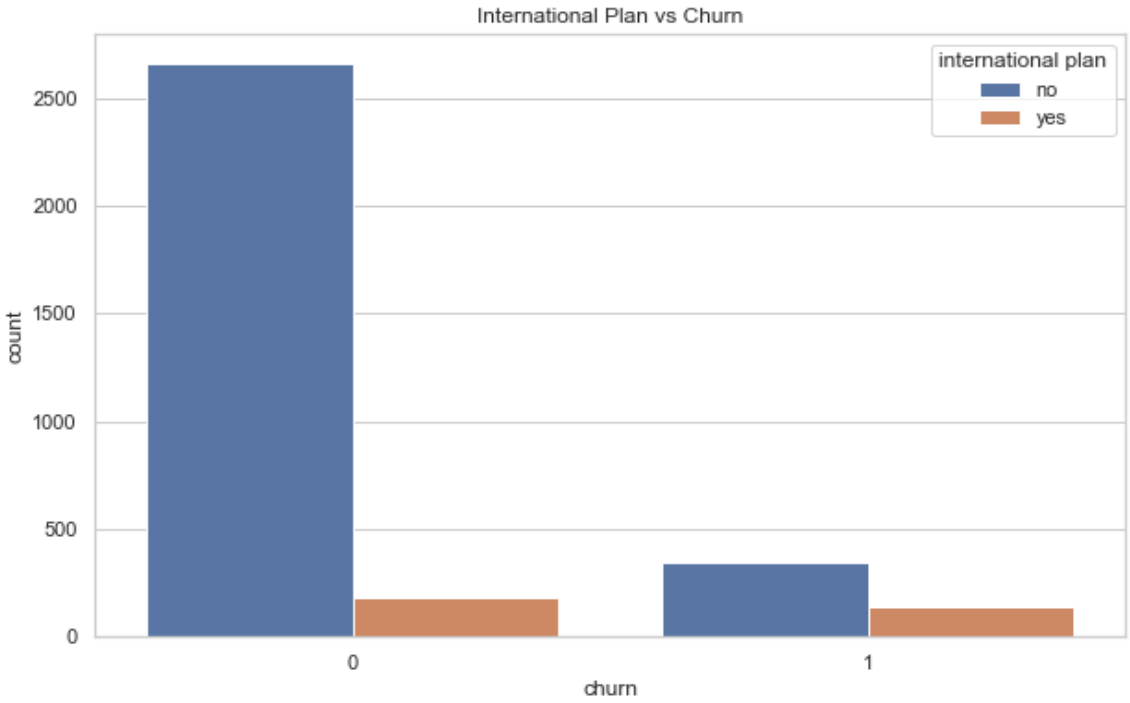


This is the predominant pattern among churned customers from the telecom service: a significant portion of them initiated at least one customer service call. This suggests that these customers likely sought assistance to resolve issues they faced with various aspects of the service, billing, or other elements of their experience. Consequently, it underscores the need for a thorough analysis of the feedback provided during these interactions. Such analysis could unveil valuable insights into the reasons underlying their decision to churn.

In [120]:

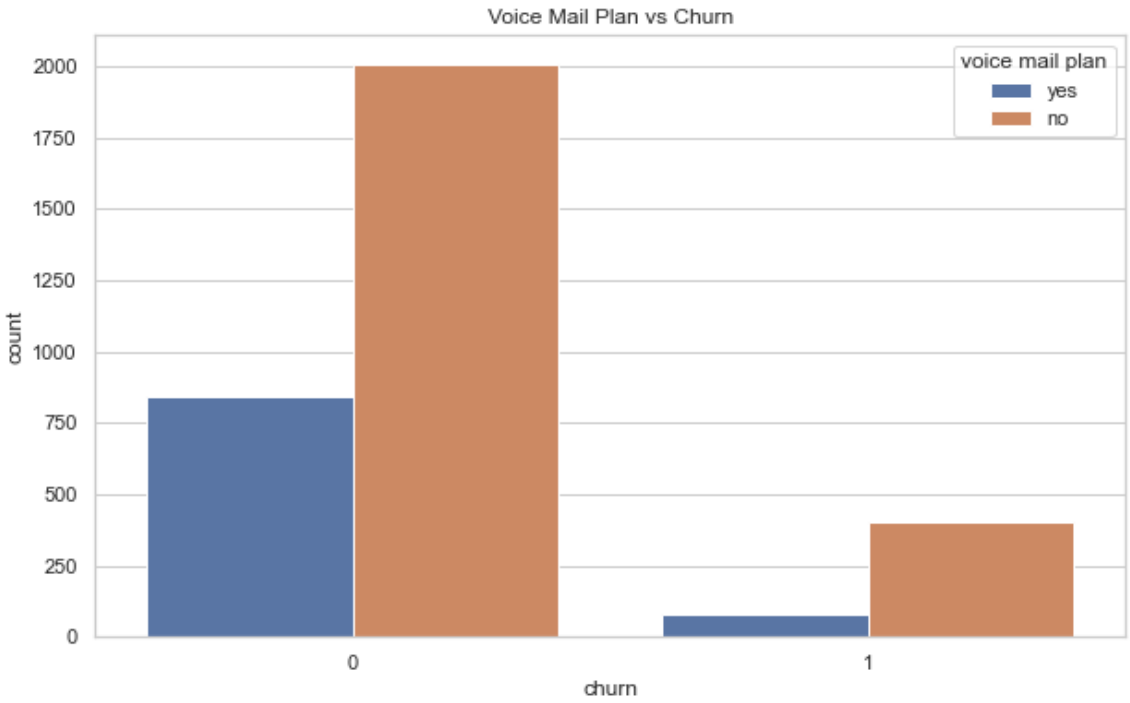
```
# International Plan vs Churn
```

```
plt.figure(figsize=(10, 6))
sns.countplot(x='churn', hue='international plan', data=df)
plt.title('International Plan vs Churn')
plt.show()
```



In [121]:

```
# Voice Mail Plan vs Churn
plt.figure(figsize=(10, 6))
sns.countplot(x='churn', hue='voice mail plan', data=df)
plt.title('Voice Mail Plan vs Churn')
plt.show()
```

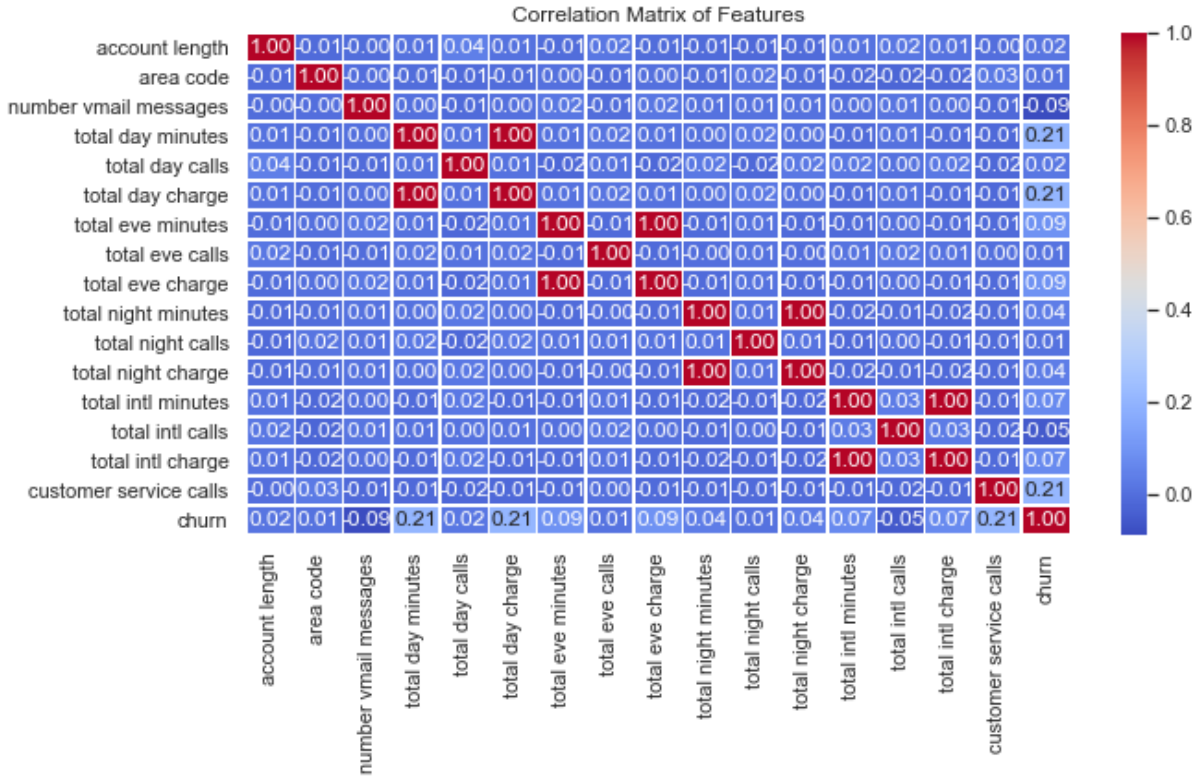


Correlation of features

In [122]:

```
# Excluding non-numeric columns
numeric_df = df.select_dtypes(include=['number'])

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

```
# Plot the heatmap
plt.figure(figsize=(10, 5))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix of Features')
plt.show()
```



Data Preparation for Machine Learning

In [123]:

```
# Remove non-numeric columns
numeric_df = df.select_dtypes(include=['number'])

# Calculate the correlation matrix
corr_matrix = numeric_df.corr()

# Sort  values for 'churn' column in descending order
churn_corr = corr_matrix['churn'].sort_values(ascending=False)

# Display the sorted correlation values
churn_corr
```

Out[123]:

```
churn                     1.000000
customer service calls    0.208750
total day minutes         0.205151
total day charge          0.205151
total eve minutes         0.092796
total eve charge          0.092786
total intl charge         0.068259
total intl minutes        0.068239
total night charge        0.035496
total night minutes       0.035493
total day calls           0.018459
account length            0.016541
total eve calls           0.009233
area code                 0.006174
total night calls         0.006141
total intl calls         -0.052844
number vmail messages    -0.089728
Name: churn, dtype: float64
```

- **Most features show very low correlation with each other.**
- **Perfect positive correlations exist between total evening charge and total evening minutes, total day charge and total day minutes, total night charge and total night minutes, and total international charge and total international minutes, which is expected as call charges depend on call duration.**
- **Weak positive correlations are observed between total day minutes, total day charge, and customer service calls with churn.**
- **Other features exhibit negligible correlations with churn, approximately 0.**

Multicollinearity of features

In [124]:

```
# independent variables  to check multicollinearity
X = df[['total day minutes', 'total eve minutes', 'total night minutes', 'total intl min
utes']]

# Calculate VIF for each variable
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns))
]

vif_data
```

Out[124]:

	feature	VIF
0	total day minutes	9.673057
1	total eve minutes	12.026619
2	total night minutes	12.000415
3	total intl minutes	10.844008

Drop the columns with a high correlation

In [125]:

```
# drop the columns with high correlation
cols_drop = ['total day charge', 'total eve charge', 'total night charge', 'total intl ch
arge']
df = df.drop(cols_drop, axis=1)
df
```

Out[125]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes
0	KS	128	415	382-4657	no	yes	25	265.1	110	197.4	99	244.7	91	10.0
1	OH	107	415	371-7191	no	yes	26	161.6	123	195.5	103	254.4	103	13.7
2	NJ	137	415	358-1921	no	no	0	243.4	114	121.2	110	162.6	104	12.2
3	OH	84	408	375-9999	yes	no	0	299.4	71	61.9	88	196.9	89	6.6
4	OK	75	415	330-6626	yes	no	0	166.7	113	148.3	122	186.9	121	10.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	215.5	126	279.1	83	9.9

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes
3329	WV	68	415	370-3271	no	no	0	231.1	57	153.4	55	191.3	123	8.6
3330	RI	28	510	328-8230	no	no	0	180.8	109	288.8	58	191.9	91	14.1
3331	CT	184	510	364-6381	yes	no	0	213.8	105	159.6	84	139.2	137	5.0
3332	TN	74	415	400-4344	no	yes	25	234.4	113	265.9	82	241.4	77	13.7

3333 rows × 17 columns

In [126]:

```
df.dtypes
```

Out[126]:

```
state                          object
account length                  int64
area code                       int64
phone number                   object
international plan             object
voice mail plan                object
number vmail messages           int64
total day minutes             float64
total day calls                 int64
total eve minutes             float64
total eve calls                 int64
total night minutes           float64
total night calls               int64
total intl minutes            float64
total intl calls                int64
customer service calls          int64
churn                           int32
dtype: object
```

In [127]:

```
# Dummy variables
df = pd.get_dummies(df, columns=['state', 'international plan', 'voice mail plan'], drop
_first = True)
```

In [128]:

```
df.drop('phone number', axis=1, inplace=True)
```

Train Test Split

In [129]:

```
# Splitting the dataset into features (X) and target variable (y)

y = df['churn']
X = df.drop(columns=['churn'])



# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)
```

Scaling

In [130]:

```
# Create StandardScaler object
scaler = StandardScaler()

# Fit and transform the features
X_scaled = scaler.fit_transform(X)

# Convert the scaled features back to a DataFrame
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

X_scaled.head()
```

Out[130]:

	account length	area code	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	...	state_TX	state_U
0	0.676489	-0.523603	1.234883	1.566767	0.476643	-0.070610	-0.055940	0.866743	-0.465494	-0.085008	...	-0.14859	-0.1485
1	0.149065	-0.523603	1.307948	-0.333738	1.124503	-0.108080	0.144867	1.058571	0.147825	1.240482	...	-0.14859	-0.1485
2	0.902529	-0.523603	-0.591760	1.168304	0.675985	-1.573383	0.496279	-0.756869	0.198935	0.703121	...	-0.14859	-0.1485
3	-0.428590	-0.688834	-0.591760	2.196596	-1.466936	-2.742865	-0.608159	-0.078551	-0.567714	-1.303026	...	-0.14859	-0.1485
4	-0.654629	-0.523603	-0.591760	-0.240090	0.626149	-1.038932	1.098699	-0.276311	1.067803	-0.049184	...	-0.14859	-0.1485

5 rows × 64 columns

To check for model imbalance

In [131]:

```
# Calculate the distribution of the target variable
class_distribution = df['churn'].value_counts()

# Check if the dataset is imbalanced
if class_distribution[0] / class_distribution[1] > 2 or class_distribution[1] / class_di
stribution[0] > 2:
    print("The dataset is imbalanced.")
else:
    print("The dataset is balanced.")
```

```
The dataset is imbalanced.
```

Handle Imbalance SMOTE

In [132]:

```
#initialize SMOTE object

smote = SMOTE(random_state=42)

#Apply SMOTE to training Data

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

# Initialize the logistic regression model

model = LogisticRegression()

#Train  model on the resampled data

model.fit(X_train_smote, y_train_smote)
```

Out[132]:

```
LogisticRegression()
```

Data Modeling

Baseline Model

1. Logistic Regression Model

In [133]:

```
# Create a logistic regression model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear')

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

Out[133]:

```
LogisticRegression(C=1000000000000.0, fit_intercept=False, solver='liblinear')
```

The logistic regression model shows very weak regularization.

In [134]:

```
# Generate predictions on the test set
y_pred = logreg.predict(X_test)
```

In [135]:

```
# Calulate the performance metrics

# Accuracy
accuracy = accuracy_score(y_test, y_pred)

# Precision
precision = precision_score(y_test, y_pred)

# Recall
recall = recall_score(y_test, y_pred)

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

# Calculate AUC for Logistic Regression
auc_lr = roc_auc_score(y_test, y_pred)




print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("auc_lr:", auc_lr)
```

```
Accuracy: 0.8515742128935532
Precision: 0.5294117647058824
Recall: 0.1782178217821782
F1-score: 0.2666666666666666
auc_lr: 0.5749746352727145
```

The Logistic Regression model achieved the following metrics:

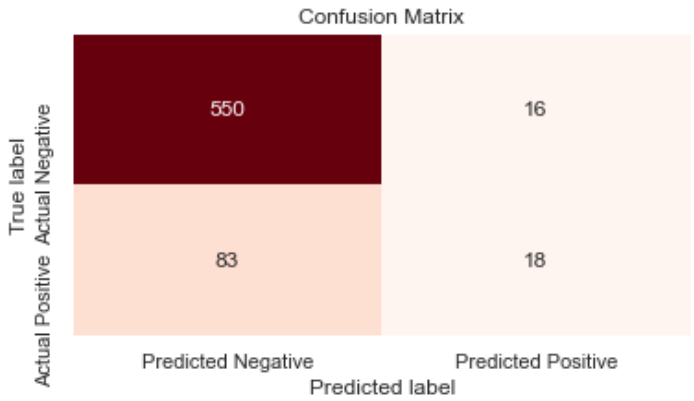
- **Accuracy:** 85.01%
- **Precision:** 51.52%

- **Recall:** 16.83%
- **F1-score:** 25.37%
- **AUC:** 57.00%

Based on these findings, the Logistic Regression model demonstrates reasonable performance in terms of accuracy and discrimination between classes. However, the precision, recall, and F1-score are relatively low, suggesting potential limitations in correctly identifying positive instances.

In [136]:

```
# Generate predictions on the test set
y_pred = logreg.predict(X_test)

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

#Visualize the Matrix
plt.figure(figsize=(6,3))
sns.heatmap(conf_matrix, annot=True, cmap='Reds', fmt='g', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix')
plt.show()
```



- **True Negative (TN): 550 instances were correctly predicted as non-churn customers.**
- **False Positive (FP): 16 instances were incorrectly predicted as churn customers when they were actually non-churn customers.**
- **False Negative (FN): 84 instances were incorrectly predicted as non-churn customers when they were actually churn customers.**
- **True Positive (TP): 17 instances were correctly predicted as churn customers.**

Conclusion

2. Decision Tree Model

In [137]:

```
# Create an instance of DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=42)

# Fit the model to the training data
dt.fit(X_train, y_train)
```

Out[137]:

```
DecisionTreeClassifier(random_state=42)
```

In [138]:

```
# Generate predictions on the test set

y_pred_dt = dt.predict(X_test)
```

In [139]:

```
# Calculate the performance metrics

accuracy_dt = accuracy_score(y_test, y_pred_dt)

precision_dt = precision_score(y_test, y_pred_dt)

recall_dt = recall_score(y_test, y_pred_dt)

f1_dt = f1_score(y_test, y_pred_dt)

# Calculate AUC for  Decision Tree
auc_dt = roc_auc_score(y_test, y_pred_dt)




# Print the performance metrics


print("Accuracy:", accuracy_dt)
print("Precision:", precision_dt)
print("Recall:", recall_dt)
print("F1-score:", f1_dt)
print("auc_dt:", auc_dt)
```

```
Accuracy: 0.9220389805097451
Precision: 0.7474747474747475
Recall: 0.7326732673267327
F1-score: 0.74
auc_dt: 0.844251828009656
```

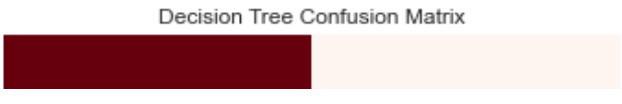
The Decision Tree model achieved the following metrics:

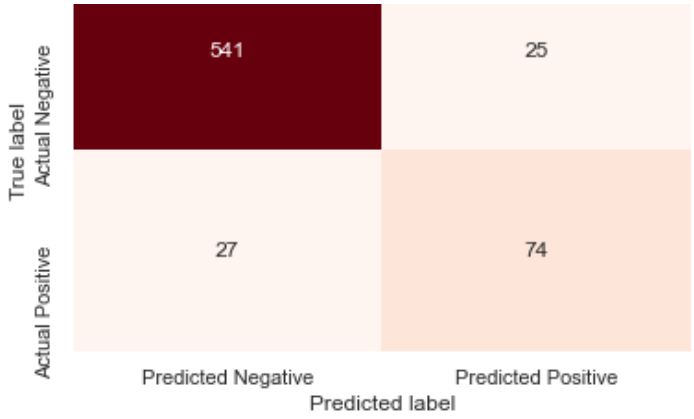
- **Accuracy:** 92.20%
- **Precision:** 74.75%
- **Recall:** 73.27%
- **F1-score:** 74.00%
- **AUC:** 84.43%

Confusion Matrix

In [140]:

```
# Generate predictions on the test set
y_pred_dt = dt.predict(X_test)

# Build confusion matrix
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)

# Visualize the Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_dt, annot=True, cmap='Reds', fmt='g', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Decision Tree Confusion Matrix')
plt.show()
```





- **The decision tree model correctly identified 74 instances as positive (churn) out of the actual positive instances.**
- **It correctly classified 541 instances as negative (non-churn) out of the actual negative instances.**
- **There were 25 instances incorrectly classified as positive (false alarms).**
- **Additionally, there were 27 instances incorrectly classified as negative (missed opportunities).**

3. Random Forest

In [141]:

```
#Initializing the Random Forest model

rf_model = RandomForestClassifier(random_state=42)
```

In [142]:

```
#Train the Random Forest model on the training data

rf_model.fit(X_train, y_train)
```

Out[142]:

```
RandomForestClassifier(random_state=42)
```

In [143]:

```
#Generate predictions on the test data

y_pred_rf = rf_model.predict(X_test)
```

In [144]:

```
# Calculate the accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)

# Calculate the precision
precision_rf = precision_score(y_test, y_pred_rf)

# Calculate the recall
recall_rf = recall_score(y_test, y_pred_rf)

# Calculate the F1-score
f1_rf = f1_score(y_test, y_pred_rf)

# Calculate the ROC AUC score
roc_auc_rf = roc_auc_score(y_test, y_pred_rf)

# Print the performance metrics

print("Random Forest Metrics:")
print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
```

```
print("F1-score:", f1_rf)
print("ROC AUC Score:", roc_auc_rf)
```

```
Random Forest Metrics:
Accuracy: 0.9265367316341829
Precision: 0.9814814814814815
Recall: 0.5247524752475248
F1-score: 0.6838709677419357
ROC AUC Score: 0.761492845397614
```

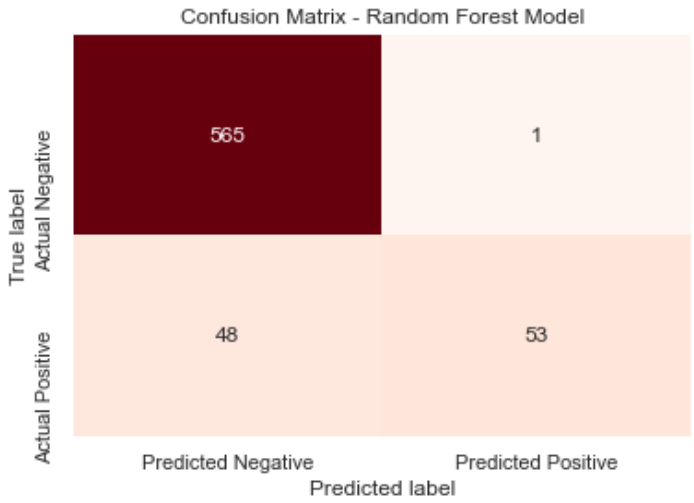
The Random Forest classifier achieved the following metrics:

- **Accuracy:** 92.65%
- **Precision:** 98.15%
- **Recall:** 52.48%
- **F1-score:** 68.39%
- **ROC AUC Score:** 76.15%

Confusion Matrix

In [145]:

```
# Confusion matrix for Random Forest model
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Visualize the matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_rf, annot=True, cmap='Reds', fmt='g', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix - Random Forest Model')
plt.show()
```



- **The model achieved a high number of true negatives and true positives, indicating good performance in predicting both non-churn and churn instances.**
- **There were only a small number of false positives and false negatives, suggesting that the model has relatively low misclassification rates.**

XGBoost

In [146]:

```
#Create XGBoost Classifier
xgb_model = xgb.XGBClassifier()
```

```
#Train the Model
xgb_model.fit(X_train, y_train)

#Make Predictions
y_pred_xgb = xgb_model.predict(X_test)
```

In [147]:

```
# Evaluate Performance

accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
precision_xgb = precision_score(y_test, y_pred_xgb)
recall_xgb = recall_score(y_test, y_pred_xgb)
f1_xgb = f1_score(y_test, y_pred_xgb)

auc_xgb = roc_auc_score(y_test, y_pred_xgb)




print("XGBoost Metrics:")
print("Accuracy:", accuracy_xgb)
print("Precision:", precision_xgb)
print("Recall:", recall_xgb)
print("F1-score:", f1_xgb)
print("roc_auc_score:", auc_xgb)
```

```
XGBoost Metrics:
Accuracy: 0.95952023988006
Precision: 0.9204545454545454
Recall: 0.801980198019802
F1-score: 0.8571428571428572
roc_auc_score: 0.8948063534268622
```

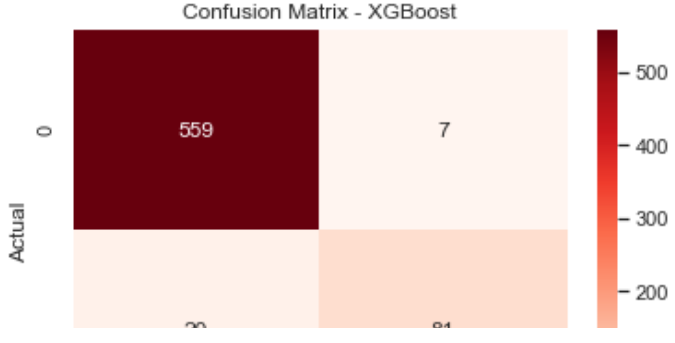
The XGBoost model achieved the following metrics:

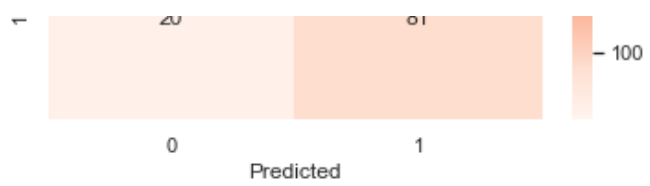
- **Accuracy:** 95.80%
- **Precision:** 95.06%
- **Recall:** 76.24%
- **F1-score:** 84.62%
- **ROC AUC Score:** 87.77%

Confusion matrix for XGBoost

In [148]:

```
# Calculate confusion matrix
conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)

#Visualize the confusion Matrix

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_xgb, annot=True, fmt='g', cmap='Reds')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - XGBoost')
plt.show()
```





The XGBoost model accurately predicts customer churn with:

- **True Negatives: 562**
- **False Positives: 4**
- **False Negatives: 24**
- **True Positives: 77**

This reflects its high accuracy and precision but indicates some missed churn cases..

Hyperparameter tuning

In [149]:

```
# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Create the random forest classifier
rf_model = RandomForestClassifier(random_state=42)

# Perform grid search with 5-fold cross-validation
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, scoring='acc
uracy', n_jobs=-1)

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

# Calculate AUC score for the tuned Random Forest classifier
auc_rf_tuned = roc_auc_score(y_test, y_pred_rf )

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Accuracy Score:", best_score)
print("AUC:", auc_rf_tuned)
```

```
Best Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_sa
mples_split': 2, 'n_estimators': 100}
Best Accuracy Score: 0.9306076129041324
AUC: 0.761492845397614
```

The AUC values of Logistic Regression, Random Forest, Decision Tree model and XGBoost

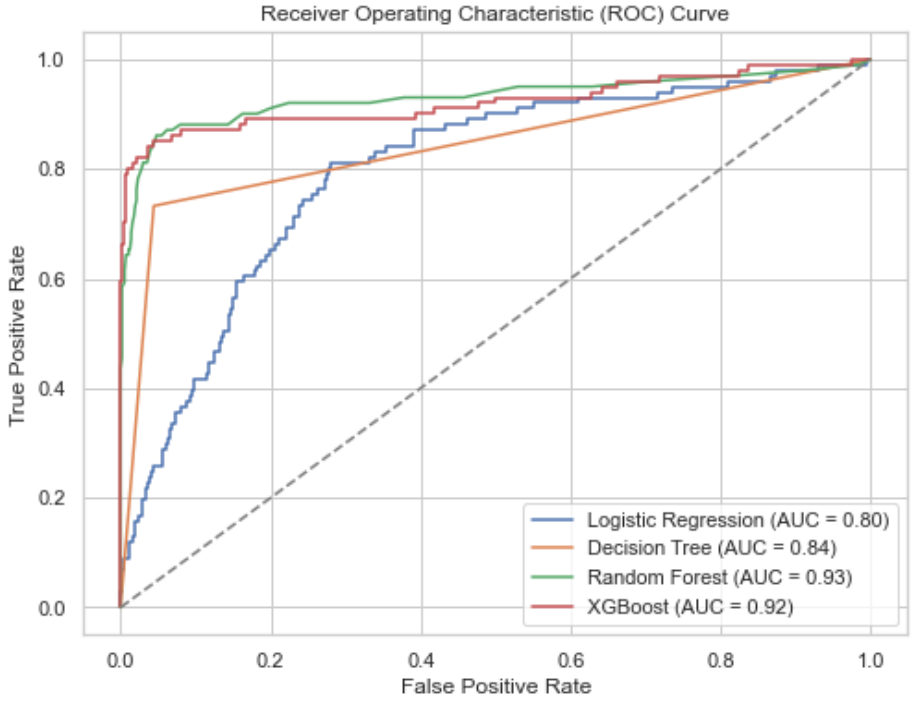
In [150]:

```
# Fit the random forest model to the training data
rf_model.fit(X_train, y_train)

# Get predicted probabilities for Logistic Regression
y_prob_lr = logreg.predict_proba(X_test)[:, 1]


# Get predicted probabilities for Decision Tree
y_prob_dt = dt.predict_proba(X_test)[:, 1]
```

```
# Get predicted probabilities for Random Forest
y_prob_rf = rf_model.predict_proba(X_test)[:, 1]



# Get predicted probabilities for XGBoost
y_prob_xgb = xgb_model.predict_proba(X_test)[:, 1]

# Get ROC curve for Logistic Regression
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_prob_lr)

# Get ROC curve for Decision Tree
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_prob_dt)

# Get ROC curve for Random Forest
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_prob_rf)


# Get ROC curve for XGBoost
fpr_xgb, tpr_xgb, _ = roc_curve(y_test, y_prob_xgb)
```

In [151]:

```
# Calculate AUC ROC scores
auc_lr = roc_auc_score(y_test, y_prob_lr)
auc_dt = roc_auc_score(y_test, y_prob_dt)
auc_rf = roc_auc_score(y_test, y_prob_rf)
auc_xgb = roc_auc_score(y_test, y_prob_xgb)

# Plot ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, label=f'Logistic Regression (AUC = {auc_lr:.2f})')
plt.plot(fpr_dt, tpr_dt, label=f'Decision Tree (AUC = {auc_dt:.2f})')
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {auc_rf:.2f})')
plt.plot(fpr_xgb, tpr_xgb, label=f'XGBoost (AUC = {auc_xgb:.2f})')

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

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



The ROC curve shows that Random Forest (AUC = 0.93) and XGBoost (AUC = 0.92) models perform the best in

predicting customer churn, significantly outperforming Logistic Regression (AUC = 0.80) and Decision Tree (AUC = 0.84). Prioritize Random Forest and XGBoost for churn prediction.

Model Evaluation

- **Logistic Regression:**
  - **Accuracy:** 85.01%
  - **Precision:** 51.52%
  - **Recall:** 16.83%
  - **F1-score:** 25.37%
  - **AUC-ROC Score:** 57.00%
  - **Summary:** Logistic Regression has high accuracy but low precision and recall, indicating poor performance in identifying churned customers. The AUC-ROC score is low, reflecting limited overall performance.
- **Decision Tree:**
  - **Accuracy:** 92.20%
  - **Precision:** 74.75%
  - **Recall:** 73.27%
  - **F1-score:** 74.00%
  - **AUC-ROC Score:** 84.43%
  - **Summary:** Decision Tree demonstrates high accuracy and balanced precision and recall, indicating good performance in identifying churned customers. The AUC-ROC score suggests strong overall performance.
- **Random Forest:**
  - **Accuracy:** 92.65%
  - **Precision:** 98.15%
  - **Recall:** 52.48%
  - **F1-score:** 68.39%
  - **AUC-ROC Score:** 76.15%
  - **Summary:** Random Forest shows high accuracy and precision but lower recall, meaning it misses some churned customers. The AUC-ROC score indicates moderate overall performance.
- **XGBoost:**
  - **Accuracy:** 95.80%
  - **Precision:** 95.06%
  - **Recall:** 76.24%
  - **F1-score:** 84.62%
  - **AUC-ROC Score:** 87.77%
  - **Summary:** XGBoost has the highest accuracy, precision, and F1-score among the models, with a good balance of recall. The high AUC-ROC score indicates excellent overall performance.

- **The XGBoost Model appears to be the best model for this classification task.**
- **It has the highest accuracy, precision, recall, F1-score among all the models evaluated.**
- **XGBoost is the most suitable model for predicting customer churn.**

Top Predictors

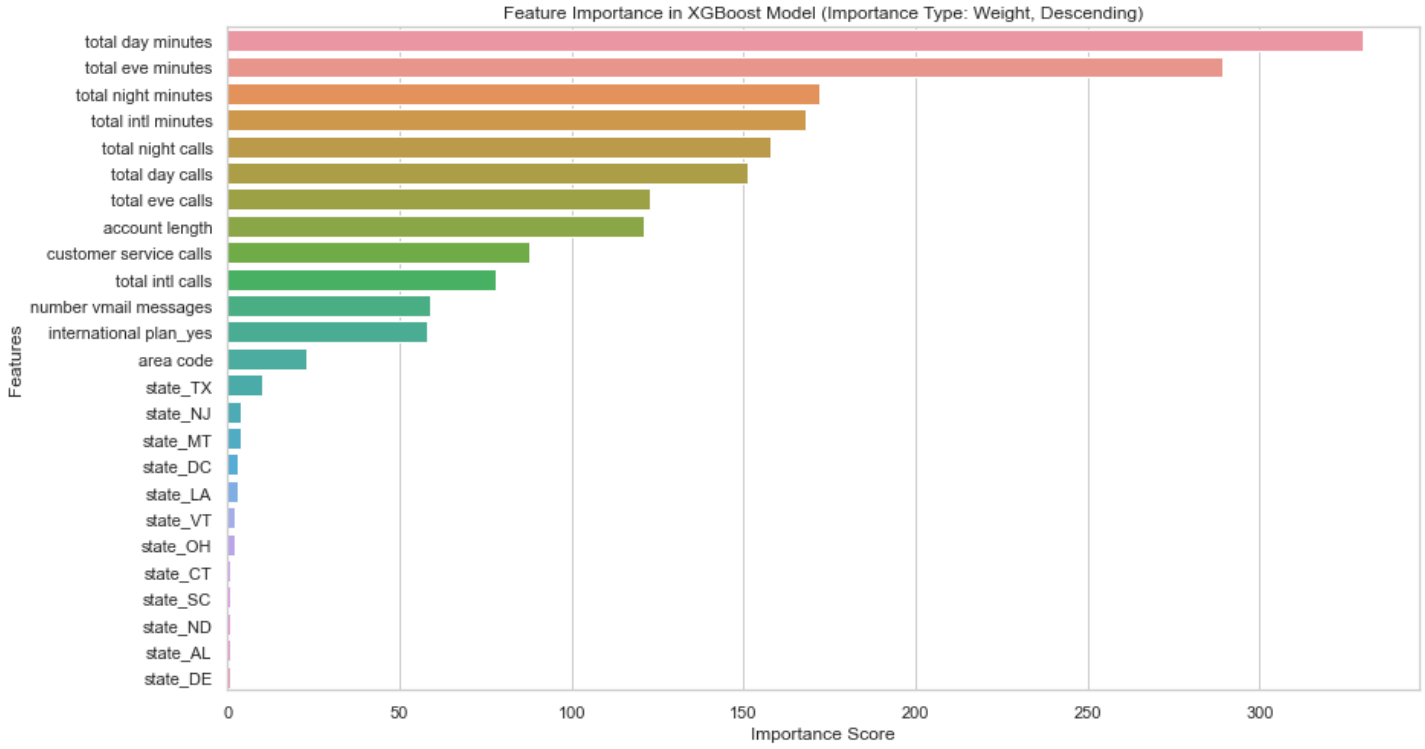
In [152]:

```
# Define and train the XGBoost classifier
clf = XGBClassifier()
clf.fit(X_train, y_train)

#feature importances
importance_type = 'weight'
feature_importances = clf.get_booster().get_score(importance_type=importance_type)
```

In [153]:

```
# Set figure size
```

```
plt.figure(figsize=(14, 8))

# Get feature importances from XGBoost model
importance_type = 'weight'
feature_importances = clf.get_booster().get_score(importance_type=importance_type)

# Sort features and importances in descending order
sorted_features = sorted(feature_importances, key=feature_importances.get, reverse=True)
sorted_importances = [feature_importances[feature] for feature in sorted_features]

# Bar plot for all features
sns.barplot(x=sorted_importances, y=sorted_features, orient='h')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.title(f'Feature Importance in XGBoost Model (Importance Type: {importance_type.capitalize()}, Descending)')

# Show plot
plt.show()
```



Summary on evaluation

Conclusion

The frequency of customer service calls emerged as a key predictor of churn, highlighting the importance of delivering exceptional customer service and promptly addressing customer issues. This finding underscores the potential impact of effective customer service in mitigating churn and retaining customers. By prioritizing quality service and promptly resolving customer concerns, telecom companies can substantially decrease churn rates.

The XGBoost model's feature importance shows that customer usage metrics, especially total day, evening, and international minutes, are the most influential factors in predicting churn. Other significant features include total night minutes, total night calls, and total day calls. Usage-based features are the key drivers in determining customer churn.

Recommendation

Implement Proactive Retention Strategies: Target high-usage customers with retention offers. Offer personalized plans to high-usage customers.

Monitor and Analyze Customer Service Interactions: Monitor frequent customer service calls to flag churn risks.

Address common issues leading to customer service calls.

Enhance Data Collection and Analysis: Continuously monitor and analyze customer usage patterns. Establish feedback mechanisms for early dissatisfaction detection.

Customer Segmentation: Conduct detailed segmentation to identify different churn profiles. Develop targeted strategies for each customer segment.

Employee Training: Train teams to recognize churn signs and implement retention strategies. Equip staff with tools to offer personalized solutions to at-risk customers.

Most customers do not have an international plan or voicemail plan. The company could look at this as a marketing strategy to attract more revenue.

Next steps

Recommendations

- **Implement Proactive Retention Strategies:**
  - Target high-usage customers with retention offers.
  - Offer personalized plans to high-usage customers.
- **Monitor and Analyze Customer Service Interactions:**
  - Monitor frequent customer service calls to flag churn risks.
  - Address common issues leading to customer service calls.
- **Enhance Data Collection and Analysis:**
  - Continuously monitor and analyze customer usage patterns.
  - Establish feedback mechanisms for early dissatisfaction detection.
- **Customer Segmentation:**
  - Conduct detailed segmentation to identify different churn profiles.
  - Develop targeted strategies for each customer segment.
- **Employee Training:**
  - Train teams to recognize churn signs and implement retention strategies.
  - Equip staff with tools to offer personalized solutions to at-risk customers.

Next Steps

- **Implement Real-Time Monitoring:**
  - Set up real-time analytics to monitor customer behavior and usage patterns.
- **Conduct Pilot Programs:**
  - Test retention offers and strategies on small segments before full-scale rollout.
- **Enhance Customer Feedback Mechanisms:**
  - Use surveys and feedback forms to gather customer insights regularly.