Business Understanding

Problem Statement

Developing a classifier to predict whether a customer will soon stop doing business with SyriaTel, a telecommunications company. This binary classification task aims to identify patterns in customer behavior and demographic information that may indicate a likelyhood to churn. The ultimate goal is to assist SyriaTel in reducing the financial impact of customer churn by implementing proactive retention strategies. We further aim to create a reliable churn rate prediction model by thoroughly examining important features based on the historical company's data

Column Name Description

- **state:** the state the user lives in
- **account length:** the number of days the user has this account
- **area code:** the code of the area the user lives in
- **phone number:** the phone number of the user
- **international plan:** true if the user has the international plan, otherwise false
- **voice mail plan:** true if the user has the voice mail plan, otherwise false
- **number vmail messages:** the number of voice mail messages the user has sent
- **total day minutes:** total number of minutes the user has been in calls during the day
- **total day calls:** total number of calls the user has done during the day
- **total day charge:** total amount of money the user was charged by the Telecom company for calls during the day
- **total eve minutes:** total number of minutes the user has been in calls during the evening
- **total eve calls:** total number of calls the user has done during the evening
- **total eve charge:** total amount of money the user was charged by the Telecom company for calls during the evening
- **total night minutes:** total number of minutes the user has been in calls during the night
- **total night calls:** total number of calls the user has done during the night
- **total night charge:** total amount of money the user was charged by the Telecom company for calls during the night
- **total intl minutes:** total number of minutes the user has been in international calls
- **total intl calls:** total number of international calls the user has done
- **total intl charge:** total amount of money the user was charged by the Telecom company for international calls
- **customer service calls:** number of customer service calls the user has done
- **churn:** true if the user terminated the contract, otherwise false

Research questions

1. what are the indicators that show likely to churn?

2. Which state has the highest churn rate

3. Does the type of customer service affect churn?

4. What is the correlation between minutes and churn?

5. Does the presence of an international calls affect the churn rate?

6. Does the account length affect the churn?

7. Does charge affect churn?

Data Understanding

In [1]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\tabit\anaconda3\lib\sit
e-packages (2.0.3)Note: you may need to restart the kernel to use updated
packages.

Requirement already satisfied: numpy in c:\users\tabit\anaconda3\lib\site-
packages (from xgboost) (1.24.3)
Requirement already satisfied: scipy in c:\users\tabit\anaconda3\lib\site-
packages (from xgboost) (1.11.1)
```

In [2]:
```
# importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
# Load the data
df= pd.read_csv('customer_churn.csv')
```

In [4]:

```
# checking the first 10 rows
df.head(10)
```

Out[4]:

	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	AL	118	510	391-8027	yes	no	0	223.4	98	37.98	...
6	MA	121	510	355-9993	no	yes	24	218.2	88	37.09	...
7	MO	147	415	329-9001	yes	no	0	157.0	79	26.69	...
8	LA	117	408	335-4719	no	no	0	184.5	97	31.37	...
9	WV	141	415	330-8173	yes	yes	37	258.6	84	43.96	...

10 rows × 21 columns

In [5]:
```
# checking the last 10 rows
df.tail(10)
```

Out[5]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge
3323	IN	117	415	362-5899	no	no	0	118.4	126	20.13
3324	WV	159	415	377-1164	no	no	0	169.8	114	28.87
3325	OH	78	408	368-8555	no	no	0	193.4	99	32.88
3326	OH	96	415	347-6812	no	no	0	106.6	128	18.12
3327	SC	79	415	348-3830	no	no	0	134.7	98	22.90
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

10 rows × 21 columns

In [6]:
```
# checking the shape of the data
df.shape
```

Out[6]: (3333, 21)

In [7]:
```
# description of the data
df.describe()
```

Out[7]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	tot: mi
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.0(
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.9
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.7
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.0(
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.6(
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.4(
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.3(
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.7(

In [8]:
```
# Information about the data
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
```

Data Preparation

In [9]:
```
# Convert column names to lowercase and replace empty spaces with under
df.columns = df.columns.str.lower().str.replace(' ', '_')
```

In [10]:
```
# Replace false/true with 0,1
df['churn'] = df['churn'].replace({False:0, True:1})
df.sample(3)
```

Out[10]:

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	n
2214	CT	90	415	347-6994	no	no	
718	AK	127	408	383-9255	no	no	
3162	UT	81	415	355-6422	no	no	

3 rows × 21 columns

In [11]:
```
# checking for null values
df.isnull().sum()
```

Out[11]:
```
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 [12]:
```
#checking for duplicates
df.duplicated().sum()
```

Out[12]:
```
0
```

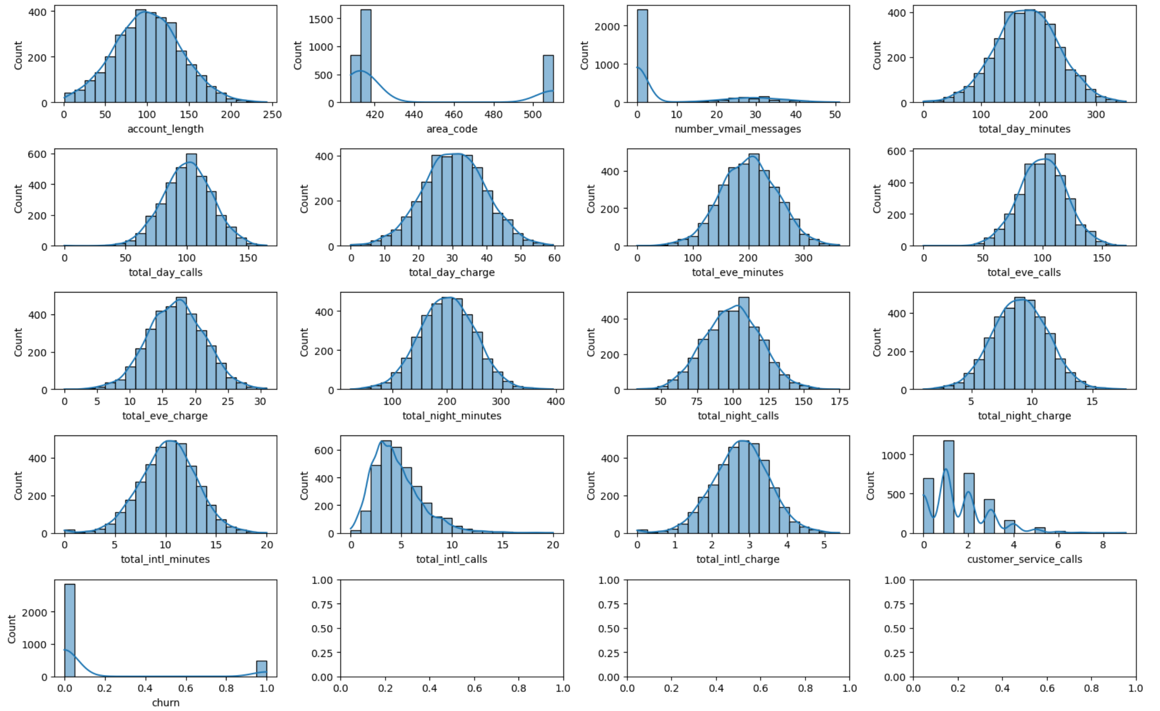
EDA

Distribution of Numeric Features in the Dataset

In [13]:
```
# checking for data types
df2 = df.select_dtypes(include=['float64','int64'])
```

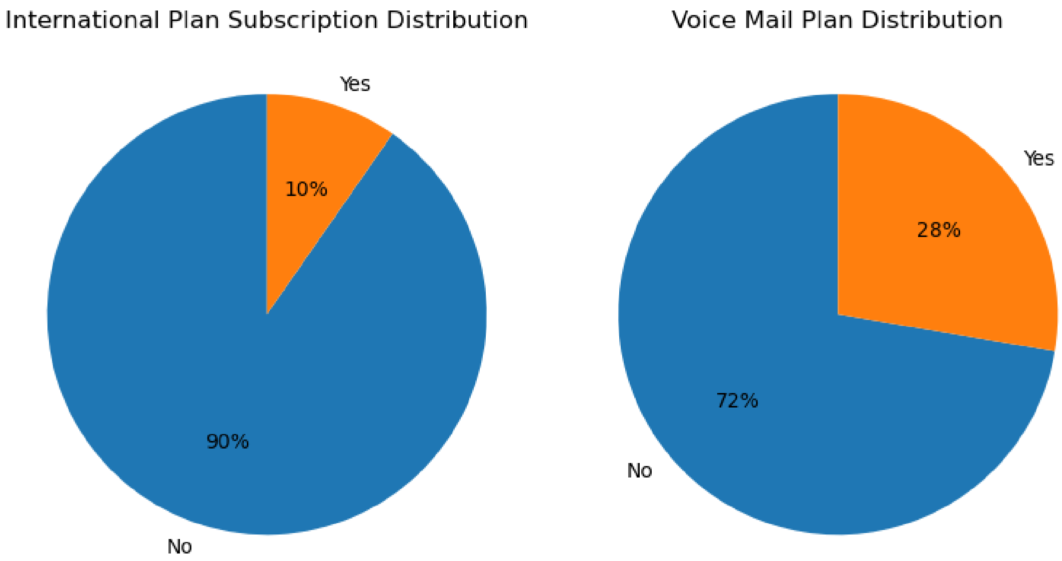
In [14]:

```
fig, axes = plt.subplots(nrows=5, ncols=4, figsize=(16, 10))

# Flatten the axes for easy iteration
axes = axes.flatten()

# Iterate through numeric columns and plot histograms
for i, feature in enumerate(df2):
    sns.histplot(df[feature], ax=axes[i], kde=True, bins=20)
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Count')

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



Subsription Plan Distribution

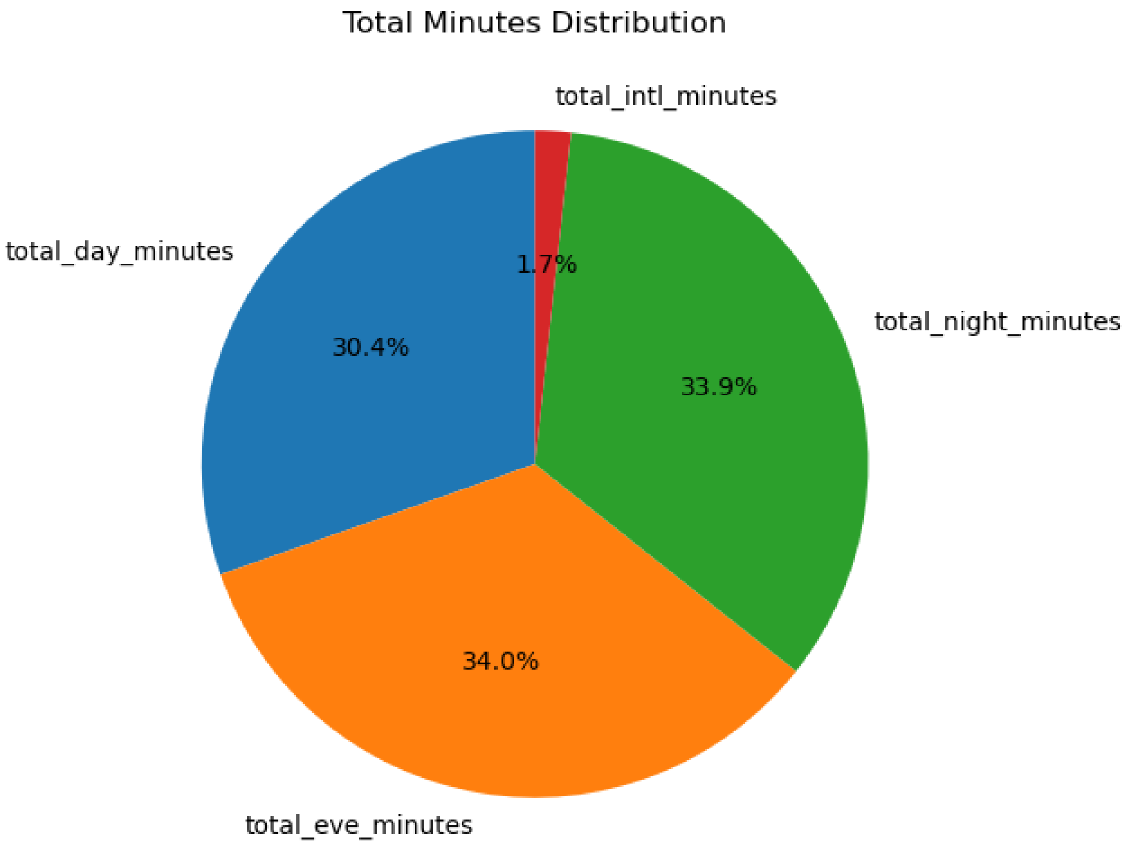
In [15]:
```
plt.figure(figsize=(8, 6))

# Plot the first pie chart
plt.subplot(1, 2, 1)
ax1 = df['international_plan'].value_counts()
plt.pie(ax1, labels=['No', 'Yes'], autopct='%.0f%%', startangle=90)
plt.title('International Plan Subscription Distribution')

# Plot the second pie chart
plt.subplot(1, 2, 2)
ax2 = df['voice_mail_plan'].value_counts()
plt.pie(ax2, labels=['No', 'Yes'], autopct='%.0f%%', startangle=90)
plt.title('Voice Mail Plan Distribution')

# Adjust layout for better spacing
plt.tight_layout()

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



Minutes Distribution

In [16]:
```
#function to create a pie chart
def pie (title, x_axis, labels):
    gsdfgdf
    plt.figure(figsize=(6, 6))
```

In [17]:
```python
col_sum = ['total_day_minutes', 'total_eve_minutes', 'total_night_minut

# Calculate the sum for each column
sums = df[col_sum].sum()

# Plot a pie chart
plt.figure(figsize=(6, 6))
plt.pie(sums, labels=sums.index, autopct='%1.1f%%', startangle=90)
plt.title('Total Minutes Distribution')
plt.show()
```

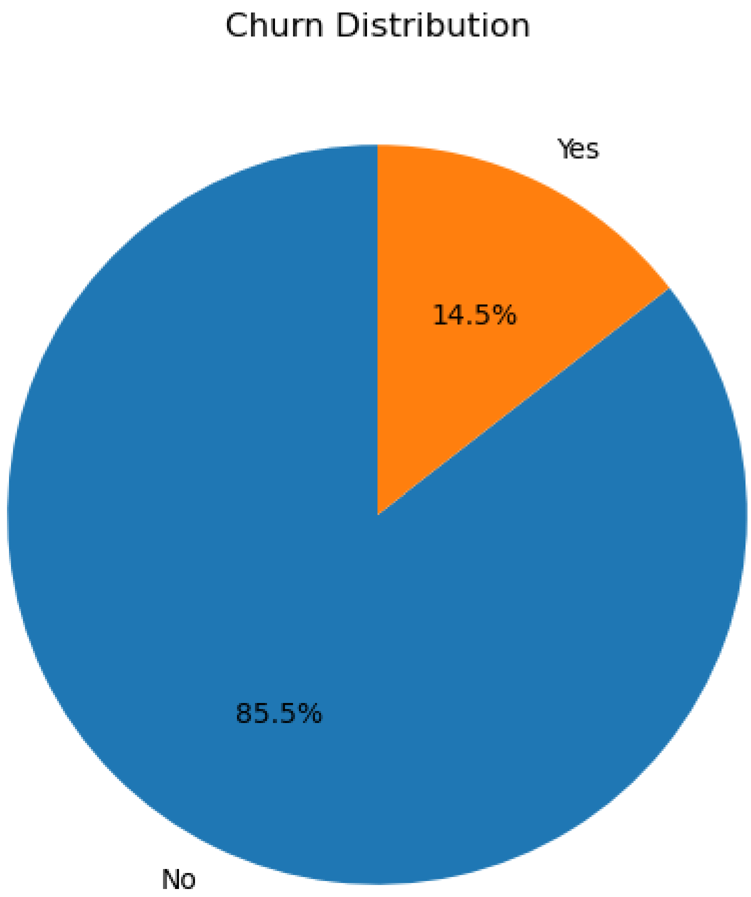


- This shows the distribution of total minutes across different call categories.
- Each slice of the pie represents the proportion of total minutes for a specific call category with total evening minutes being the highest.

Churn Distribution

In [18]:

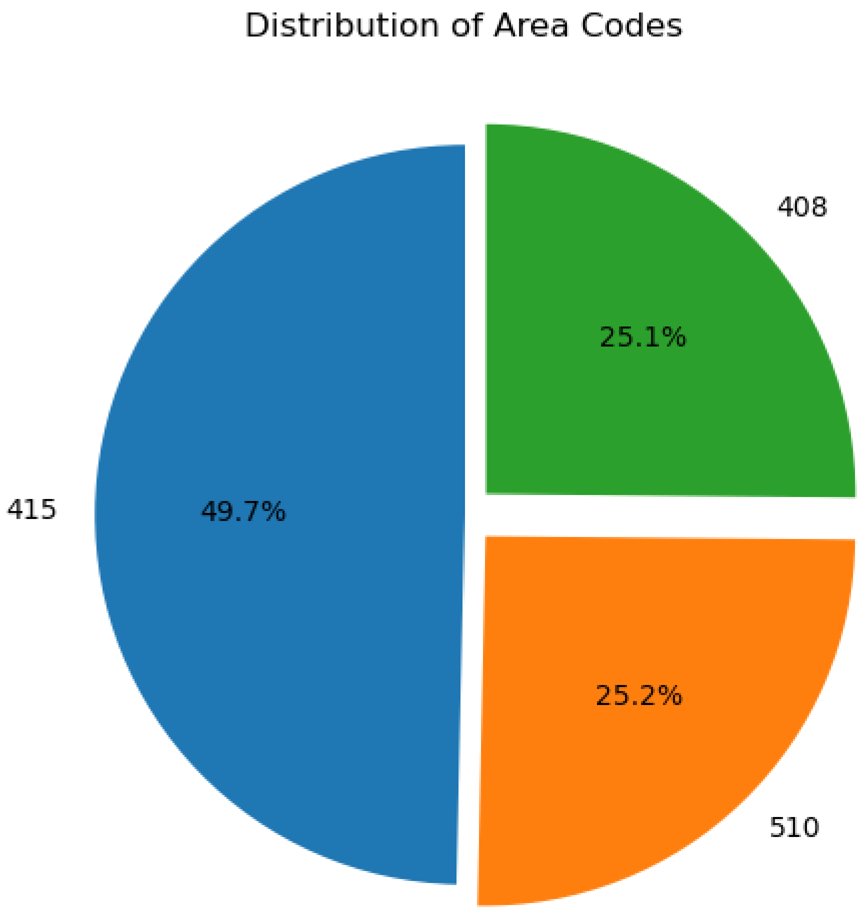
```
churn_counts = df['churn'].value_counts()
plt.figure(figsize=(6, 6))
plt.pie(churn_counts, labels=['No', 'Yes'], autopct='%1.1f%%', startang
plt.title('Churn Distribution')
plt.show()
```



- The pie chart demonstrates that a smaller percentage of customers are churning (leaving) compared to those who are staying. This insight is valuable for businesses to understand the current state of customer retention and can inform strategies aimed at reducing churn and enhancing customer satisfaction.

Area Code Distribution

In [19]:
```
area_code_counts = df['area_code'].value_counts()

# Plot a pie chart
plt.figure(figsize=(6, 6))
plt.pie(area_code_counts, labels=area_code_counts.index, autopct='%1.1f
plt.title('Distribution of Area Codes')
plt.show()
```



Churn Distribution by State

In [20]:
```
from IPython.display import display

# Dictionary mapping state initials to full names
state_mapping = {
    'AL': 'Alabama',
    'AK': 'Alaska',
    'AZ': 'Arizona',
    'AR': 'Arkansas',
    'CA': 'California',
    'CO': 'Colorado',
    'CT': 'Connecticut',
    'DE': 'Delaware',
    'FL': 'Florida',
    'GA': 'Georgia',
    'HI': 'Hawaii',
    'ID': 'Idaho',
    'IL': 'Illinois',
    'IN': 'Indiana',
    'IA': 'Iowa',
    'KS': 'Kansas',
    'KY': 'Kentucky',
    'LA': 'Louisiana',
    'ME': 'Maine',
    'MD': 'Maryland',
    'MA': 'Massachusetts',
    'MI': 'Michigan',
    'MN': 'Minnesota',
    'MS': 'Mississippi',
    'MO': 'Missouri',
    'MT': 'Montana',
    'NE': 'Nebraska',
    'NV': 'Nevada',
    'NH': 'New Hampshire',
    'NJ': 'New Jersey',
    'NM': 'New Mexico',
    'NY': 'New York',
    'NC': 'North Carolina',
    'ND': 'North Dakota',
    'OH': 'Ohio',
    'OK': 'Oklahoma',
    'OR': 'Oregon',
    'PA': 'Pennsylvania',
    'RI': 'Rhode Island',
    'SC': 'South Carolina',
    'SD': 'South Dakota',
    'TN': 'Tennessee',
    'TX': 'Texas',
    'UT': 'Utah',
    'VT': 'Vermont',
    'VA': 'Virginia',
    'WA': 'Washington',
    'WV': 'West Virginia',
    'WI': 'Wisconsin',
    'WY': 'Wyoming'
}


grouped_df = df.groupby(['state', 'churn']).size().unstack()  # Group b
grouped_df.index = grouped_df.index.map(state_mapping)  # Map state ini
grouped_df['Total'] = grouped_df.sum(axis=1)  # Calculate total number

```

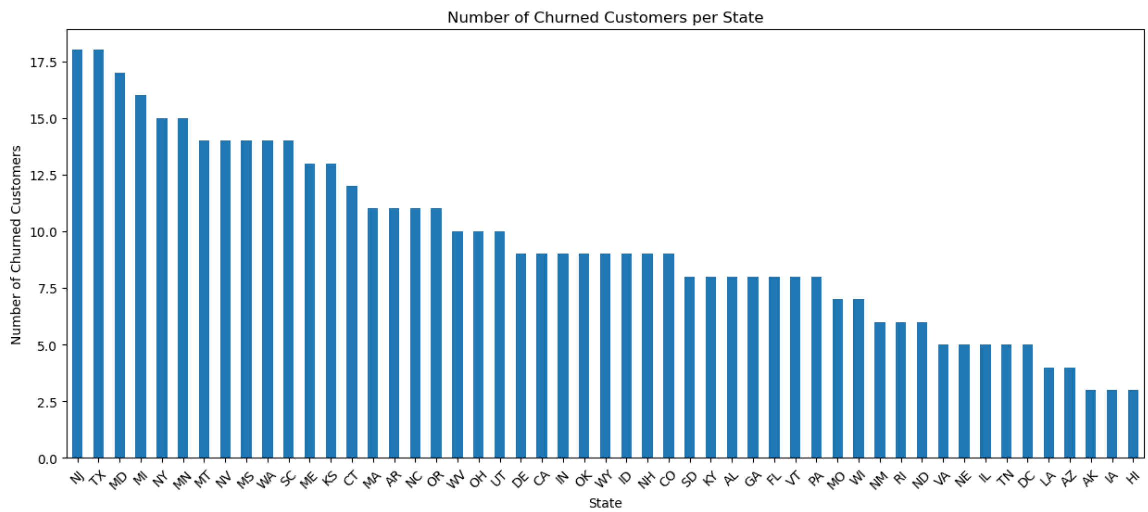
```
# Display the DataFrame as a table
display(grouped_df)
```

churn	0	1	Total
state			
Alaska	49	3	52
Alabama	72	8	80
Arkansas	44	11	55
Arizona	60	4	64
California	25	9	34
Colorado	57	9	66
Connecticut	62	12	74
NaN	49	5	54
Delaware	52	9	61
Florida	55	8	63
Georgia	46	8	54
Hawaii	50	3	53
Iowa	41	3	44
Idaho	64	9	73
Illinois	53	5	58
Indiana	62	9	71
Kansas	57	13	70
Kentucky	51	8	59
Louisiana	47	4	51
Massachusetts	54	11	65
Maryland	53	17	70
Maine	49	13	62
Michigan	57	16	73
Minnesota	69	15	84
Missouri	56	7	63
Mississippi	51	14	65
Montana	54	14	68
North Carolina	57	11	68
North Dakota	56	6	62
Nebraska	56	5	61
New Hampshire	47	9	56
New Jersey	50	18	68
New Mexico	56	6	62
Nevada	52	14	66
New York	68	15	83
Ohio	68	10	78
Oklahoma	52	9	61
Oregon	67	11	78

churn	0	1	Total
state			
Pennsylvania	37	8	45
Rhode Island	59	6	65
South Carolina	46	14	60
South Dakota	52	8	60
Tennessee	48	5	53
Texas	54	18	72
Utah	62	10	72
Virginia	72	5	77
Vermont	65	8	73
Washington	52	14	66
Wisconsin	71	7	78
West Virginia	96	10	106
Wyoming	68	9	77

In [21]:

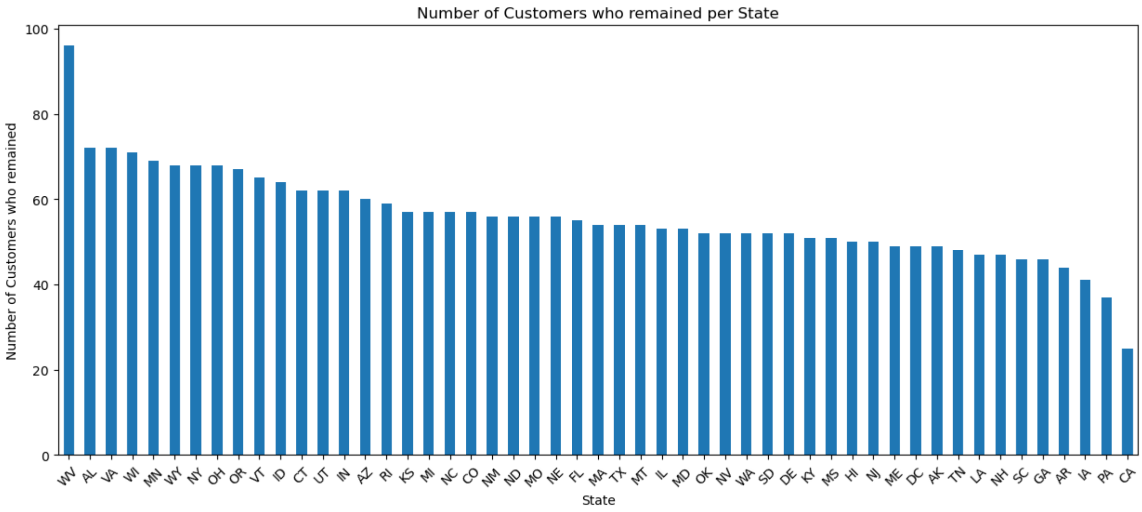
```
churned_df = df[df['churn'] == 1]  # Filter DataFrame to include only c
state_churned_count = churned_df['state'].value_counts()  # Count the n

# Plotting the number of churned customers per state
plt.figure(figsize=(15, 6))
state_churned_count.plot(kind='bar')
plt.title('Number of Churned Customers per State')
plt.xlabel('State')
plt.ylabel('Number of Churned Customers')
plt.xticks(rotation=45)
plt.show()
```



- The highest state that did churn was New Jersey state

In [22]:

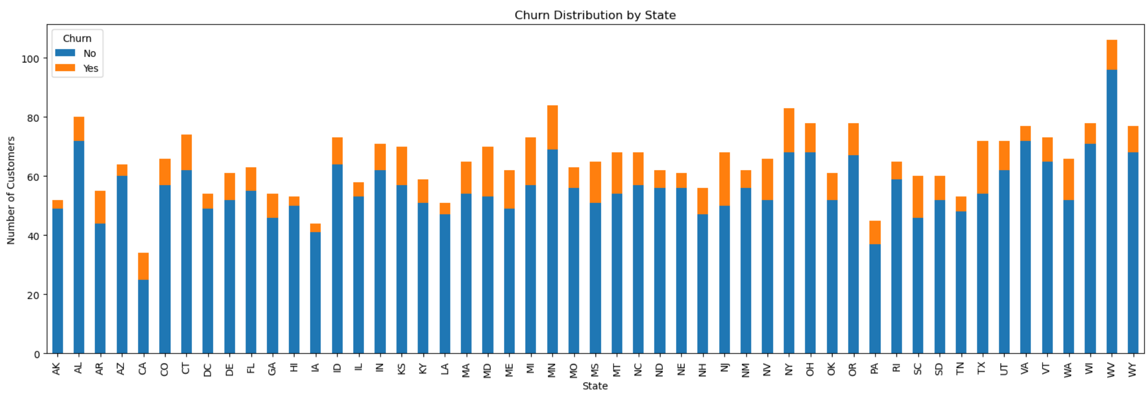
```
churned_df = df[df['churn'] == 0]  # Filter DataFrame to include only c
state_churned_count = churned_df['state'].value_counts()  # Count the n

# Plotting the number of churned customers per state
plt.figure(figsize=(15, 6))
state_churned_count.plot(kind='bar')
plt.title('Number of Customers who remained per State')
plt.xlabel('State')
plt.ylabel('Number of Customers who remained')
plt.xticks(rotation=45)
plt.show()
```



- The highest number of Customers who remained are recorded in West Virginia

In [23]:

```
# Plot churn distribution by state
df.groupby(['state', 'churn']).size().unstack().plot(kind='bar', figsiz

plt.title('Churn Distribution by State')
plt.xlabel('State')
plt.ylabel('Number of Customers')
plt.legend(['No', 'Yes'], title='Churn')
plt.show()
```

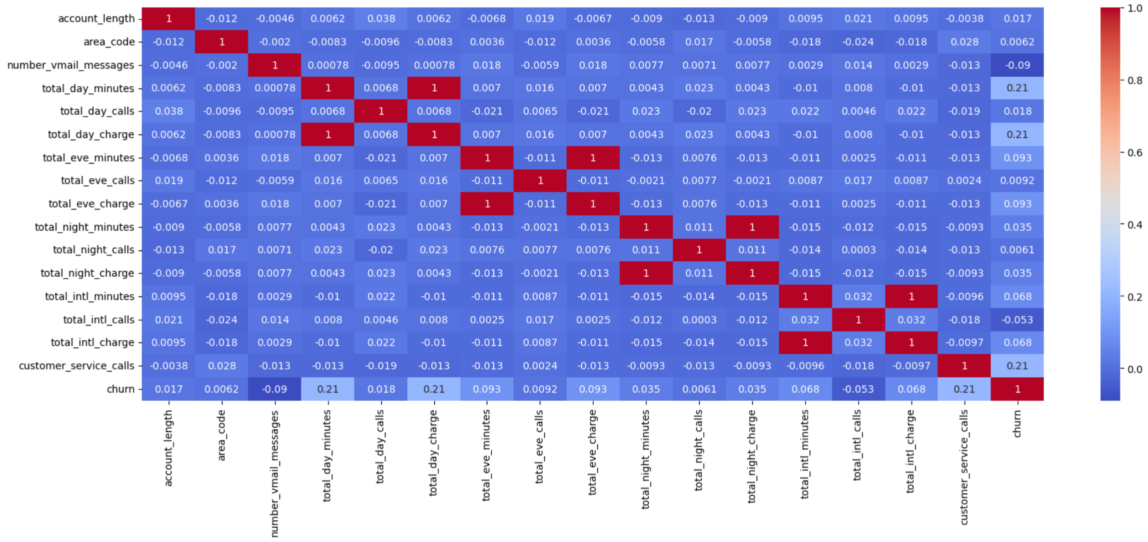


- This shows that West Virginia has thr highest number of customers, with a high retention

Correlation of the features

In [24]:

```python
df2 = df.select_dtypes(include=['float64','int64'])
correlation_matrix = df2.corr()

plt.figure(figsize=(20,7))
sns.heatmap(correlation_matrix, annot = True, cmap='coolwarm')
plt.show()
```



In [25]:

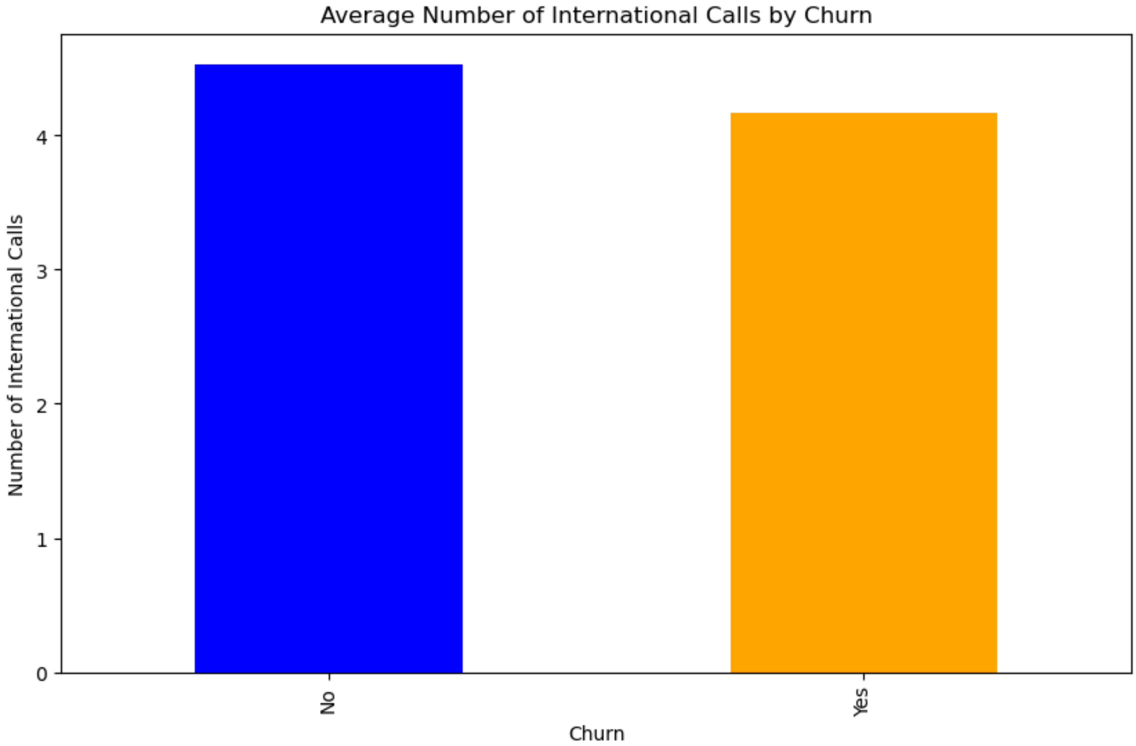
```python
corr_matrix = df2.corr()
churn_corr = corr_matrix['churn'].sort_values(ascending=False)
churn_corr
```

Out[25]:

```
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
```

In [26]:

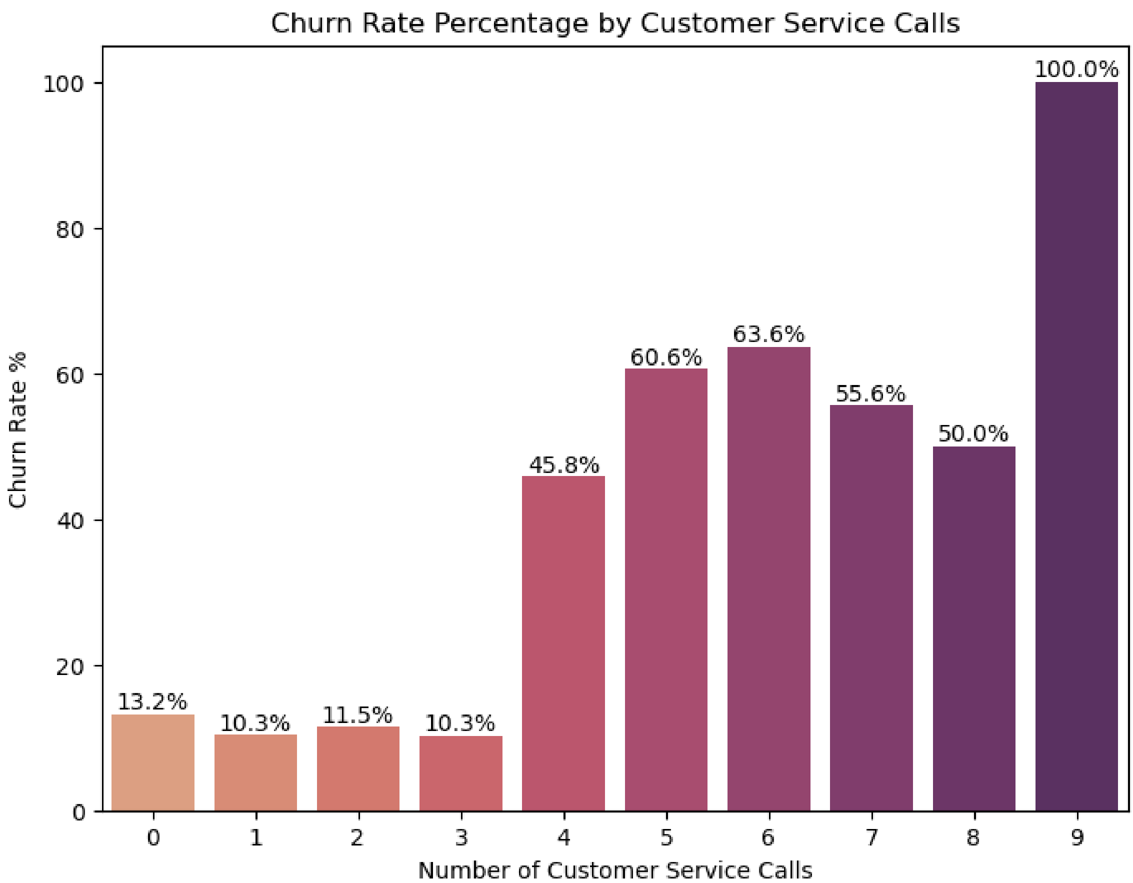
```
# Calculate average number of international calls for each churn status
total_intl_calls = df.groupby('churn')['total_intl_calls'].mean()

plt.figure(figsize=(10, 6))  # Create a new figure with specific size
total_intl_calls.plot(kind='bar', color=['blue', 'orange'])  # Create a
plt.title('Average Number of International Calls by Churn')  # Add titl
plt.xlabel('Churn')  # Add label for x-axis
plt.ylabel('Number of International Calls')  # Add label for y-axis


plt.xticks([0, 1], ['No', 'Yes'])
plt.show()  # Display the plot

```



Churn rate by Customer Service Calls

In [27]:

```
# Calculate churn rate percentage for each number of customer service c
churn_rate = df.groupby('customer_service_calls')['churn'].mean() * 100

# Plotting a bar plot
plt.figure(figsize=(8, 6))
ax = sns.barplot(x=churn_rate.index, y=churn_rate.values, palette='flar
ax.bar_label(ax.containers[0], fmt='%.1f%%', label_type='edge') # Add %


# Adding title and labels
plt.title('Churn Rate Percentage by Customer Service Calls')
plt.xlabel('Number of Customer Service Calls')
plt.ylabel('Churn Rate %')

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



There is a positive correlation between the number of customer service calls and the likelihood of churn.

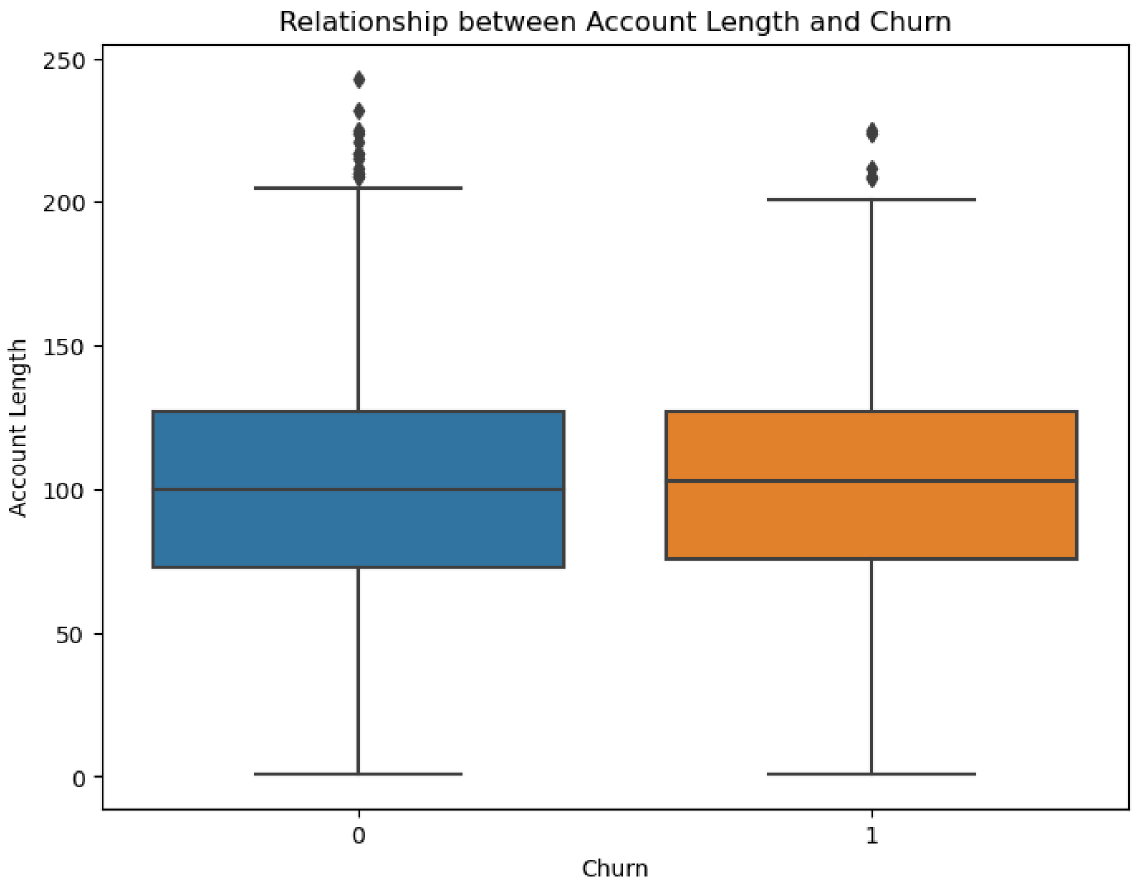
- As the number of customer service calls increases, customers are more likely to churn, or stop using the service. However, there is a disparity in the trend. Specifically, there is a noticeable increase in churn at the 6th customer service call.
- This observation indicates that there might be a particular threshold or point where additional customer service interactions start to have a negative impact on customer retention, potentially leading to a higher churn rate.

- While there is a general trend of increasing churn with more customer service calls, it highlights a potential anomaly or critical point at the 6th customer service call where churn rate spikes, suggesting that this specific interaction might be a key factor in

Relationship between Account Length and Churn

In [28]:

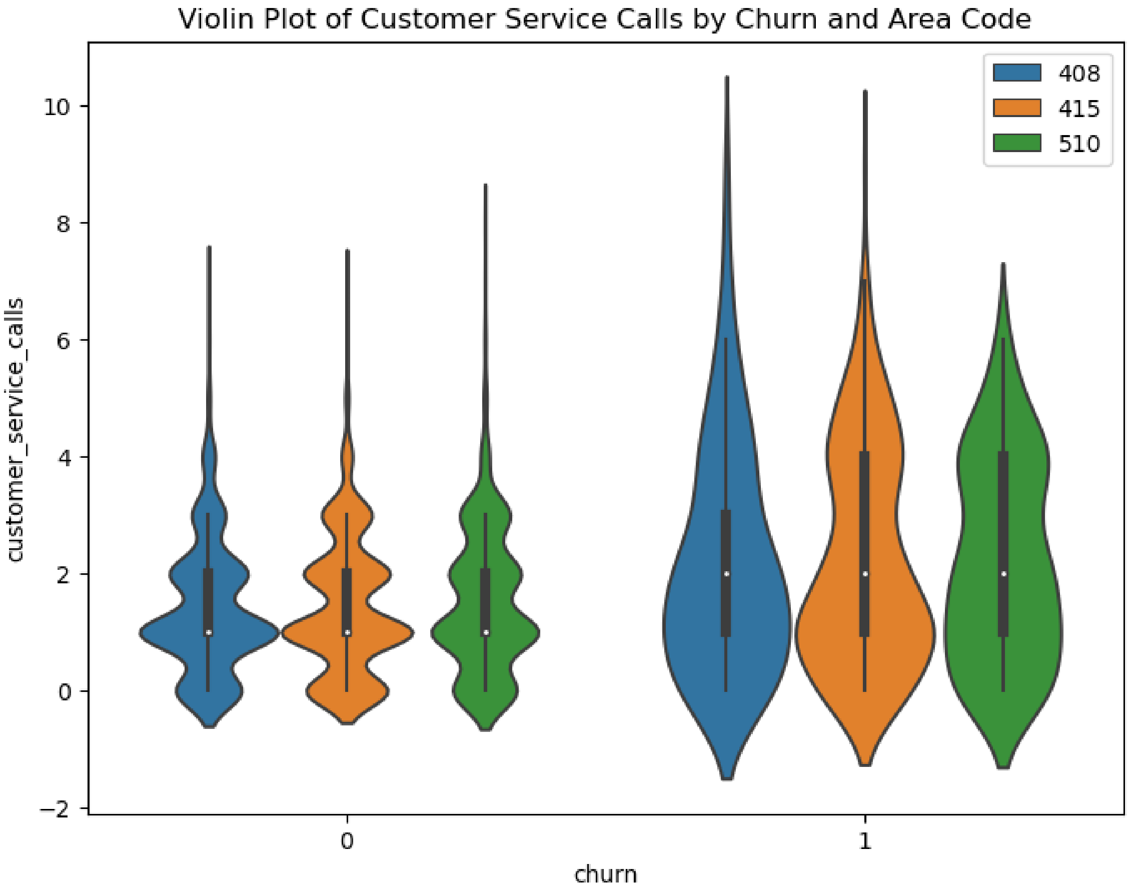
```
# Box plot
plt.figure(figsize=(8, 6))  # Create a new figure with specific size
sns.boxplot(x='churn', y='account_length', data=df)  # Create a box plo
plt.title('Relationship between Account Length and Churn')  # Add title
plt.xlabel('Churn')  # Add label for x-axis
plt.ylabel('Account Length')  # Add label for y-axis
plt.show()  # Display the plot
```



- There's a noticeable difference in median account length between the two groups, it indicates that account length is a factor influencing churn behavior.

Violin plots of the different area codes by customer service calls by churn

In [29]:
```
plt.figure(figsize=(8, 6))
sns.violinplot(data=df, x='churn', y='customer_service_calls', hue='are
plt.legend(loc='upper right')
plt.title('Violin Plot of Customer Service Calls by Churn and Area Code
plt.show()
```



- The plot shows the density of data points at different values of customer service calls for each combination of churn status and area code.

In [30]:
```
churn_corr
```

Out[30]:
```
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
```

We can drop one of the columns with a 1 correlation (total day minutes and total day charge), (total eve minutes and total eve charge), (total night minutes and total night charge), (total intl minutes and total intl charge) and phone number.

- total day charge
- total eve charge
- total night charge
- total intl charge

In [31]:
```
# drop the columns
cols_drop = ['total_day_charge', 'total_eve_charge', 'total_night_charg
df3 = df.drop(cols_drop, axis=1)
df3.head(2)
```

Out[31]:

	state	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messa
0	KS	128	415	no	yes	
1	OH	107	415	no	yes	

In [32]:
```
df3.dtypes
```

Out[32]:
```
state                      object
account_length              int64
area_code                   int64
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                       int64
dtype: object
```

In [33]:
```
df3['churn'].unique()
```

Out[33]:
```
array([0, 1], dtype=int64)
```

In [34]:
```
# Dummy variables
df4 = pd.get_dummies(df3, columns=['state', 'international_plan', 'voic
```

In [35]:
```
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 65 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   account_length          3333 non-null   int64  
 1   area_code               3333 non-null   int64  
 2   number_vmail_messages   3333 non-null   int64  
 3   total_day_minutes       3333 non-null   float64
 4   total_day_calls         3333 non-null   int64  
 5   total_eve_minutes       3333 non-null   float64
 6   total_eve_calls         3333 non-null   int64  
 7   total_night_minutes     3333 non-null   float64
 8   total_night_calls       3333 non-null   int64  
 9   total_intl_minutes      3333 non-null   float64
 10  total_intl_calls        3333 non-null   int64  
 11  customer_service_calls  3333 non-null   int64  
 12  churn                   3333 non-null   int64  
 13  state_AL                3333 non-null   bool   
 14  state_AR                3333 non-null   bool   
 15  state_AZ                3333 non-null   bool   
 16  state_CA                3333 non-null   bool   
 17  state_CO                3333 non-null   bool   
 18  state_CT                3333 non-null   bool   
 19  state_DC                3333 non-null   bool   
 20  state_DE                3333 non-null   bool   
 21  state_FL                3333 non-null   bool   
 22  state_GA                3333 non-null   bool   
 23  state_HI                3333 non-null   bool   
 24  state_IA                3333 non-null   bool   
 25  state_ID                3333 non-null   bool   
 26  state_IL                3333 non-null   bool   
 27  state_IN                3333 non-null   bool   
 28  state_KS                3333 non-null   bool   
 29  state_KY                3333 non-null   bool   
 30  state_LA                3333 non-null   bool   
 31  state_MA                3333 non-null   bool   
 32  state_MD                3333 non-null   bool   
 33  state_ME                3333 non-null   bool   
 34  state_MI                3333 non-null   bool   
 35  state_MN                3333 non-null   bool   
 36  state_MO                3333 non-null   bool   
 37  state_MS                3333 non-null   bool   
 38  state_MT                3333 non-null   bool   
 39  state_NC                3333 non-null   bool   
 40  state_ND                3333 non-null   bool   
 41  state_NE                3333 non-null   bool   
 42  state_NH                3333 non-null   bool   
 43  state_NJ                3333 non-null   bool   
 44  state_NM                3333 non-null   bool   
 45  state_NV                3333 non-null   bool   
 46  state_NY                3333 non-null   bool   
 47  state_OH                3333 non-null   bool   
 48  state_OK                3333 non-null   bool   
 49  state_OR                3333 non-null   bool   
 50  state_PA                3333 non-null   bool   
 51  state_RI                3333 non-null   bool   
 52  state_SC                3333 non-null   bool   
 53  state_SD                3333 non-null   bool   
 54  state_TN                3333 non-null   bool   
 55  state_TX                3333 non-null   bool   
```

```
 56   state_UT                3333 non-null   bool
 57   state_VA                3333 non-null   bool
 58   state_VT                3333 non-null   bool
 59   state_WA                3333 non-null   bool
 60   state_WI                3333 non-null   bool
 61   state_WV                3333 non-null   bool
 62   state_WY                3333 non-null   bool
 63   international_plan_yes  3333 non-null   bool
 64   voice_mail_plan_yes     3333 non-null   bool
dtypes: bool(52), float64(4), int64(9)
memory usage: 507.9 KB
```

Data Modelling

We start with a Logistic Regression for our baseline model

1. Logistic Regression Model

In [36]:

```
# Y Target Variable
y = df4['churn']
X = df4.drop('churn', axis = 1)
```

In [37]:

```
# Create Scaler Object to standardize
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# fit and transform
X_scaled = pd.DataFrame(scaler.fit_transform(X))

X_scaled.head()
```

Out[37]:

	0	1	2	3	4	5	6	7	8
0	0.524793	0.068627	0.490196	0.755701	0.666667	0.542755	0.582353	0.595750	0.408451
1	0.438017	0.068627	0.509804	0.460661	0.745455	0.537531	0.605882	0.621840	0.492958
2	0.561983	0.068627	0.000000	0.693843	0.690909	0.333242	0.647059	0.374933	0.500000
3	0.342975	0.000000	0.000000	0.853478	0.430303	0.170195	0.517647	0.467187	0.394366
4	0.305785	0.068627	0.000000	0.475200	0.684848	0.407754	0.717647	0.440290	0.619718

5 rows × 64 columns

In [38]:
```
# perform train test split
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

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

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

Out[38]:
```
LogisticRegression(C=1000000000000.0, fit_intercept=False, solver='libline
ar')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [39]:
```
# Generate predictions
y_hat_train = logreg.predict(X_train)
y_hat_test = logreg.predict(X_test)
```

In [40]:
```
# Calulate the performance metrics
from sklearn.metrics import accuracy_score, precision_score, recall_sco

# Accuracy
accuracy = accuracy_score(y_test, y_hat_test)
print(f"Accuracy: {accuracy}")

# Precision
precision = precision_score(y_test, y_hat_test)
print(f"Precision: {precision}")

# Recall
recall = recall_score(y_test, y_hat_test)
print(f"Recall: {recall}")

# F1-score
f1 = f1_score(y_test, y_hat_test)
print(f"F1-score: {f1}")

# False positive rate(fpr) and true positive rate(tpr)
fpr, tpr, thresholds = roc_curve(y_test, y_hat_test)

# calculate the AUC
auc = auc(fpr, tpr)
print(f"AUC: {auc}")
```

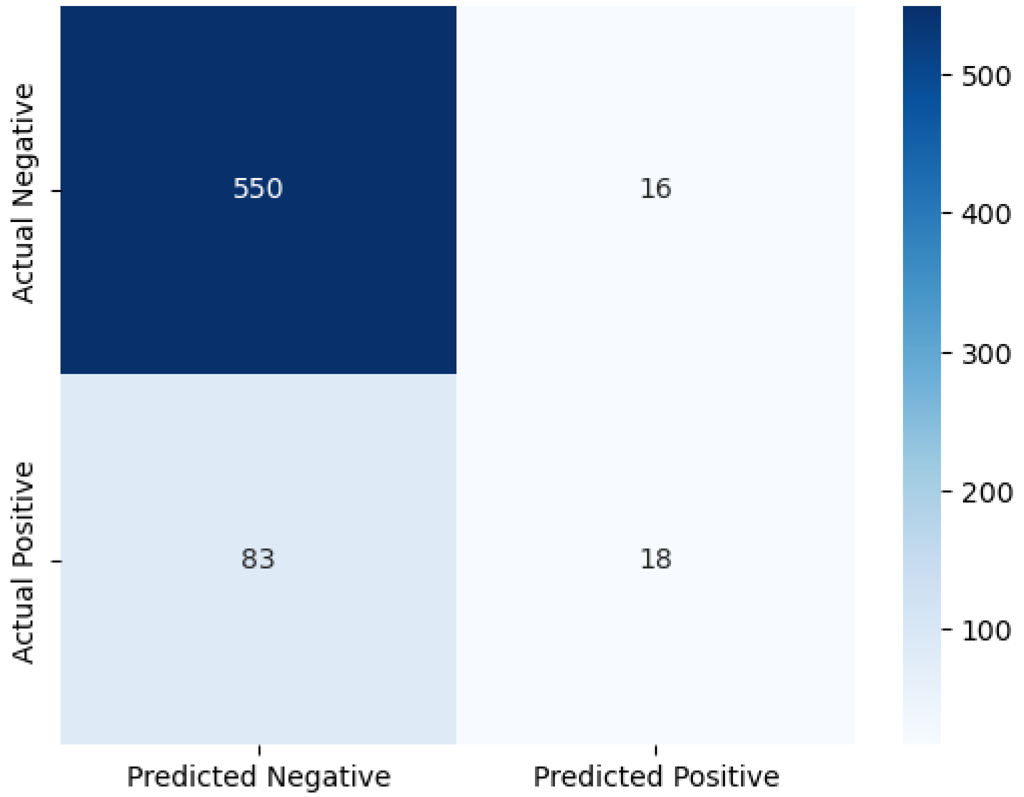
```
Accuracy: 0.8515742128935532
Precision: 0.5294117647058824
Recall: 0.1782178217821782
F1-score: 0.26666666666666666
AUC: 0.5749746352727145
```

- Accuracy: Our classifier shows that our model 85% accurate
- Presicion: Out of all the instances our model predicted as positive, approximately 52.94% were actually positive.

- Recall: Being approximately 17.82%, means that out of all the actual positive instances, our model identified approximately 17.82% correctly.
- F1 score provides a balance between precision and recall. In our case, the F1-score is approximately 26.67%.
- AUC is approximately 0.575, suggesting that our model's ability to distinguish between positive and negative classes is more or less the same as random guessing.

In [41]:
```
# Build a confusion matrix
from sklearn.metrics import confusion_matrix

# Confusion matrix
cm = confusion_matrix(y_test, y_hat_test)

#  Visualize
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
```

Out[41]: <Axes: >



- The model predicted 18 True Positives, 551 True Negatives, 15 False Positives and 83 False Negatives.
- The model predicted 18 customers would churn and they did.
- The model predicted that 551 customers would not churn and they didn't
- The model predicted that 15 customers would churn but they didn't
- The model incorrectly predicted that 83 customers would not churn but they actually churned

To check if the model is imbalanced

In [42]:
```
# Calculate class distribution
class_counts = df4['churn'].value_counts()
majority_class = class_counts.idxmax()
majority_count = class_counts.max()

# Check for imbalance. Threshold for imbalance --> 0.8
if majority_count / len(df4) > 0.8:
    print('df4 is imbalanced.')
else:
    print('df4 is balanced.')
```

```
df4 is imbalanced.
```

- Due to the imbalance of the data we use Random Forest since it handles imbalance better compared to Logistic Regression models

2. Random Forest

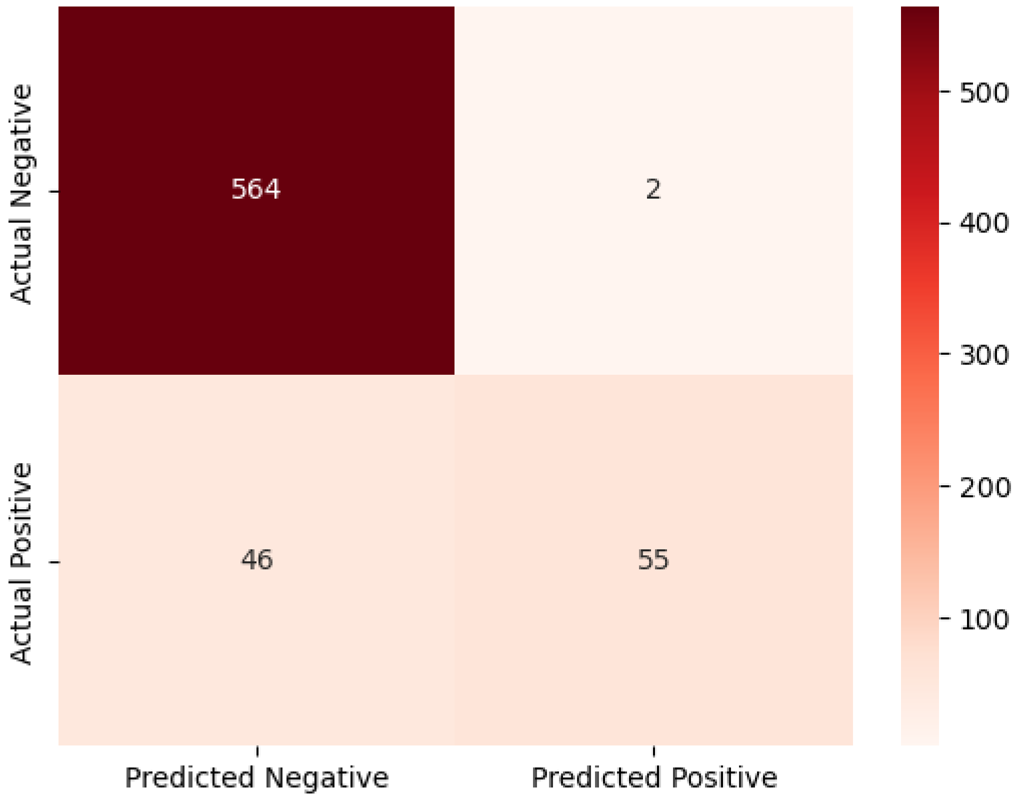
In [43]:
```
# Create and fit a random forest classifier
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(random_state=0)
clf = clf.fit(X_train, y_train)
```

In [44]:
```
from sklearn.metrics import f1_score, precision_recall_curve, roc_auc_s
from sklearn.metrics import roc_curve, auc

# Predictions on the testing data
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Calculate F1-score
f1 = f1_score(y_test, y_pred)
print(f"F1-score: {f1}")

# Calculate precision and recall for different thresholds
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

# Calculate ROC AUC
roc_auc = roc_auc_score(y_test, y_pred)

print(f"ROC AUC: {roc_auc}")
```

```
Accuracy: 0.9280359820089955
F1-score: 0.6962025316455697
ROC AUC: 0.7705104432704755
```

- F1-score being 77% is moderately good performance. AUC of 77% is okay for the imbalanced data. This gives a generally good performance of the model.
- Having an accuracy of 92.9%, performs better than that of Logistic Regression Model, 85.3%

In [45]:
```
from sklearn.metrics import confusion_matrix

cm1 = confusion_matrix(y_test, y_pred)
# Visualize the confusion matrix
sns.heatmap(cm1, annot=True, fmt="d",  cmap='Reds',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.show()
```



- The model predicted 55 True Positives, 564 True Negatives, 2 False Positives and 46 False Negatives.
- The model predicted 55 customers would churn and they did.
- The model predicted that 564 customers would not churn and they didn't
- The model predicted that 2 customers would churn but they didn't
- The model incorrectly predicted that 46 customers would not churn but they actually churned

Random Forest has performed better that the logistic Regression model.

Hyperparameter tuning

In [46]:
```
# Define hyperparameter grid
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [100, 125, 150],
    'max_depth': [5, 10, 15, 20, 25],
    'min_samples_leaf': [1, 2],
    'min_samples_split': [2, 5],
    'criterion': ['gini', 'entropy'],
}

# Create Random Forest model
clf = RandomForestClassifier(random_state=0)

# Initialize GridSearchCV with early stopping and smaller training frac
grid_search = GridSearchCV(clf, param_grid, scoring='f1', cv=3, n_jobs=

# Fit the model on a smaller portion of training data (replace 0.8 with
grid_search.fit(X_train[:int(0.8 * len(X_train))], y_train[:int(0.8 * l

 # Get best parameters and model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

 # Evaluate on the testing set
y_pred = best_model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"F1-score: {f1}")
print(f"Best Hyperparameters: {best_params}")
```

```
Accuracy: 0.9295352323838081
F1-score: 0.6967741935483871
Best Hyperparameters: {'criterion': 'gini', 'max_depth': 25, 'min_samples_
leaf': 1, 'min_samples_split': 5, 'n_estimators': 150}
```

- about 92.9% of the insttances was correctly predicted
- 69.6% F1-score shows a relatively good precision and recall

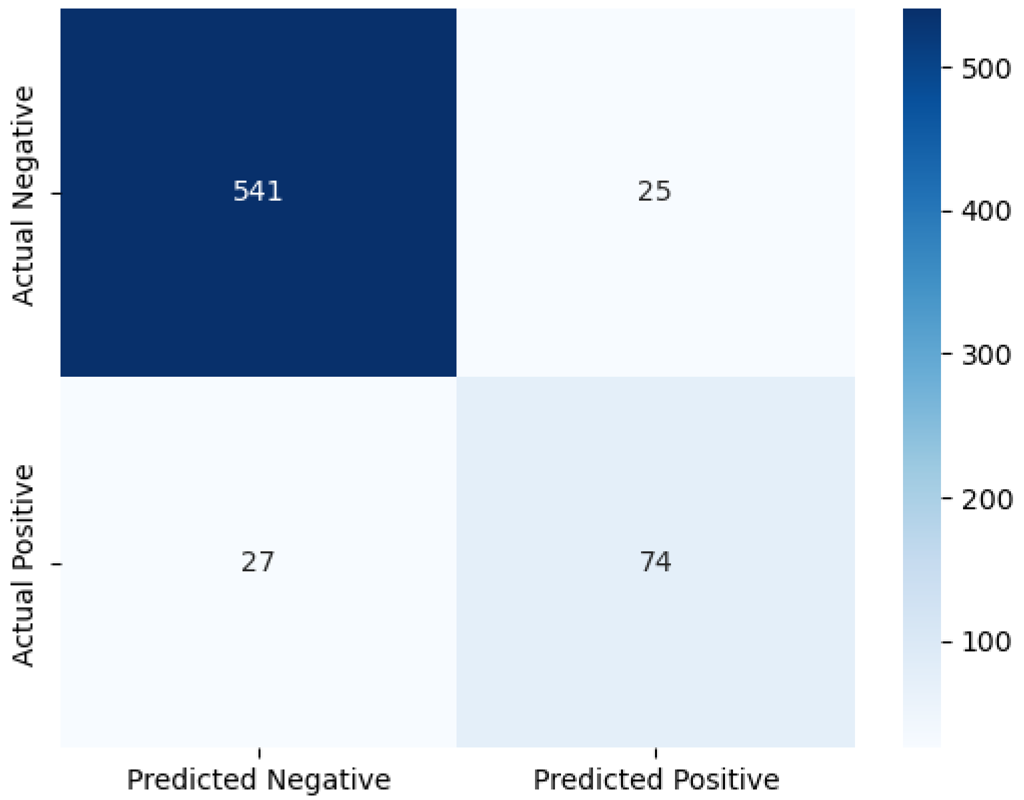
3. DecisionTrees

In [47]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [48]:
```
#Instantiate DecisionTreeClassifier
dt_clf = DecisionTreeClassifier(random_state=42)
```

In [49]:
```
#Fit on the training data
dt_clf.fit(X_train, y_train)

#predict on the test set
dt_y_pred = dt_clf.predict(X_test)
```

In [50]:
```
cm3 = confusion_matrix(y_test, dt_y_pred)
# Visualize the confusion matrix
sns.heatmap(cm3, annot=True, fmt="d",  cmap='Blues',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.show()
```



- We can see that the model made correct predictions for 541 negative cases and 74 positive cases. However, it incorrectly predicted 25 positive cases as negative and 27 negative cases as positive.

In [51]:
```
# Calculate accuracy
accuracy = accuracy_score(y_test, dt_y_pred)
print(f"Accuracy: {accuracy}")

# Calculate F1-score
f1 = f1_score(y_test, dt_y_pred)
print(f"F1-score: {f1}")

# Calculate precision and recall for different thresholds
precision, recall, thresholds = precision_recall_curve(y_test, dt_y_pre

# Calculate ROC AUC
roc_auc = roc_auc_score(y_test, dt_y_pred)

print(f"ROC AUC: {roc_auc}")
```

```
Accuracy: 0.9220389805097451
F1-score: 0.74
ROC AUC: 0.844251828009656
```

4. XGBoost

In [52]:
```
from xgboost import XGBClassifier
```

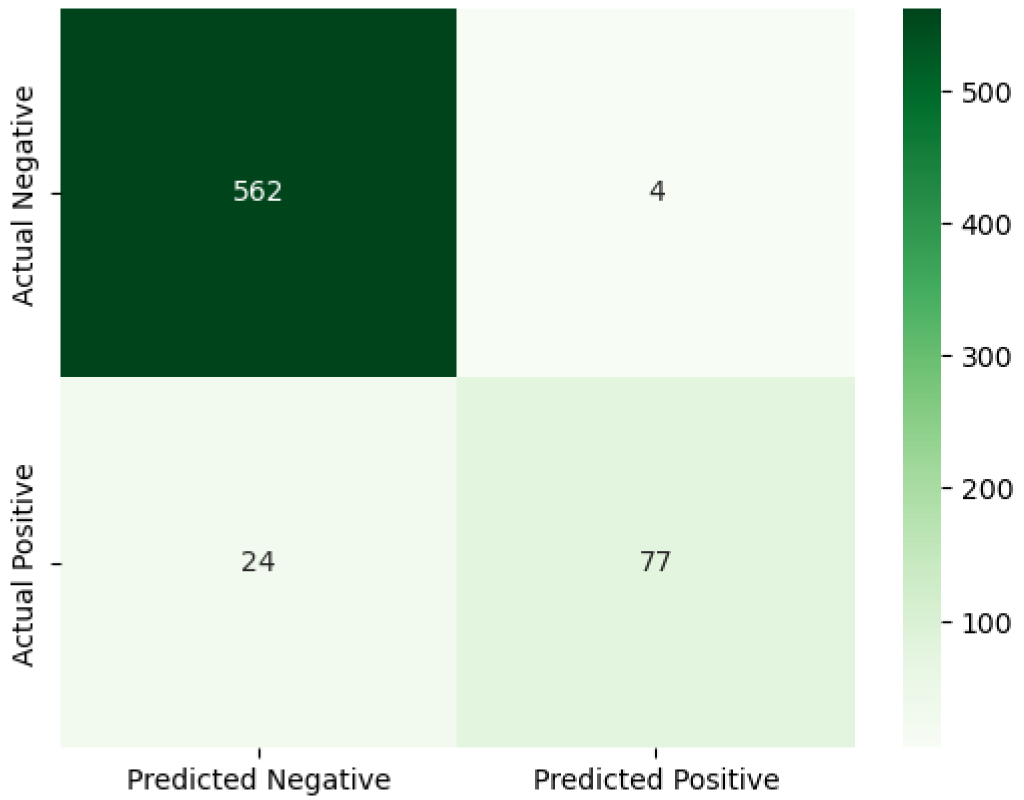
In [53]:
```
#instantiate XGBClassifier
xg_clf = XGBClassifier(random_state=42)

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

Out[53]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=No
ne,
              enable_categorical=False, eval_metric=None, feature_types=No
ne,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=No
ne,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=Non
e,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=42, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [54]:
```
#predict on the test data
y_pred = xg_clf.predict(X_test)
```

In [55]:
```
cm2 = confusion_matrix(y_test, y_pred)
# Visualize the confusion matrix
sns.heatmap(cm2, annot=True, fmt="d",  cmap='Greens',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.show()
```



- The matrix shows that the model correctly identified 562 negative cases (True Negatives) and 77 positive cases (True Positives). 4 negative cases were incorrectly predicted as positive (False Positives), and 24 positive cases were incorrectly predicted as negative (False Negatives).

In [56]:
```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Calculate F1-score
f1 = f1_score(y_test, y_pred)
print(f"F1-score: {f1}")

# Calculate precision and recall for different thresholds
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

# Calculate ROC AUC
roc_auc = roc_auc_score(y_test, y_pred)

print(f"ROC AUC: {roc_auc}")
```

```
Accuracy: 0.9580209895052474
F1-score: 0.8461538461538463
ROC AUC: 0.8776545499072875
```

Show the AUC values of Logistic Regression, Random Forest models and XGBoost

In [57]:
```
# Make predictions on the test data
y_pred_lr = logreg.predict(X_test)

# Calculate probability predictions for the positive class
# Assuming positive class is at index 1
lr_predictions_proba = logreg.predict_proba(X_test)[:, 1]
```

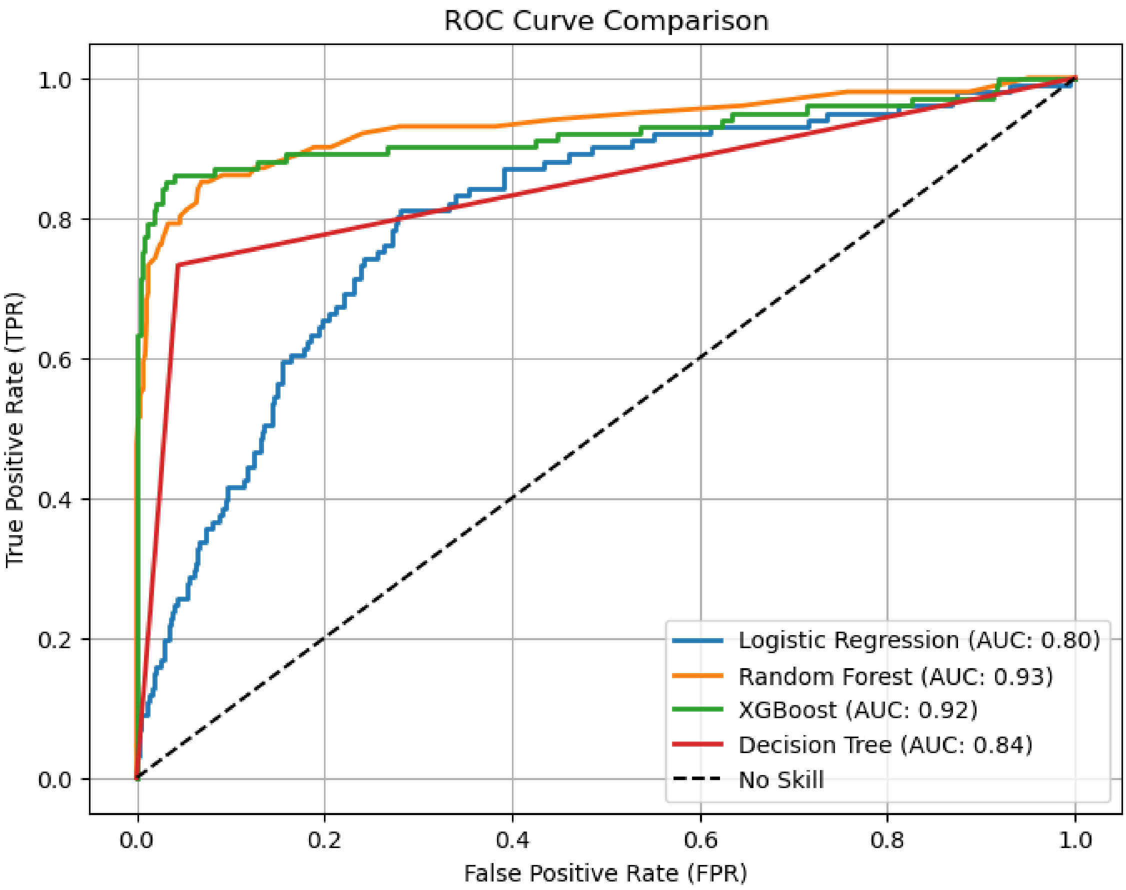
In [58]:
```
# Calculate probability predictions for the positive class
clf.fit(X_train, y_train)
rf_predictions_proba = clf.predict_proba(X_test)[:, 1]
```

In [59]:
```
# Calculate probability predictions for the positive class for theXGBoo
xg_clf.fit(X_train, y_train)
xg_predictions_proba = xg_clf.predict_proba(X_test)[:, 1]
```

In [60]:
```
# Calculate probability predictions for the positive class for decision
dt_clf.fit(X_train, y_train)
dt_predictions_proba = dt_clf.predict_proba(X_test)[:, 1]
```

In [61]:

```
# ROC curve calculation and plot
fpr_lr, tpr_lr, _ = roc_curve(y_test, lr_predictions_proba)
roc_auc_lr = auc(fpr_lr, tpr_lr)

fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_predictions_proba)
roc_auc_rf = auc(fpr_rf, tpr_rf)

fpr_xg, tpr_xg, _ = roc_curve(y_test, xg_predictions_proba)
roc_auc_xg = auc(fpr_xg, tpr_xg)

fpr_dt, tpr_dt, _ = roc_curve(y_test, dt_predictions_proba)
roc_auc_dt = auc(fpr_dt, tpr_dt)

plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, label='Logistic Regression (AUC: {:.2f})'.form
         linewidth=2)
plt.plot(fpr_rf, tpr_rf, label='Random Forest (AUC: {:.2f})'.format(roc
         linewidth=2)
plt.plot(fpr_xg, tpr_xg, label='XGBoost (AUC: {:.2f})'.format(roc_auc_x
         linewidth=2)
plt.plot(fpr_dt, tpr_dt, label='Decision Tree (AUC: {:.2f})'.format(roc
         linewidth=2)

plt.plot([0, 1], [0, 1], linestyle='--', color='black', label='No Skill
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve Comparison')
plt.legend()
plt.grid(True)
plt.show()
```



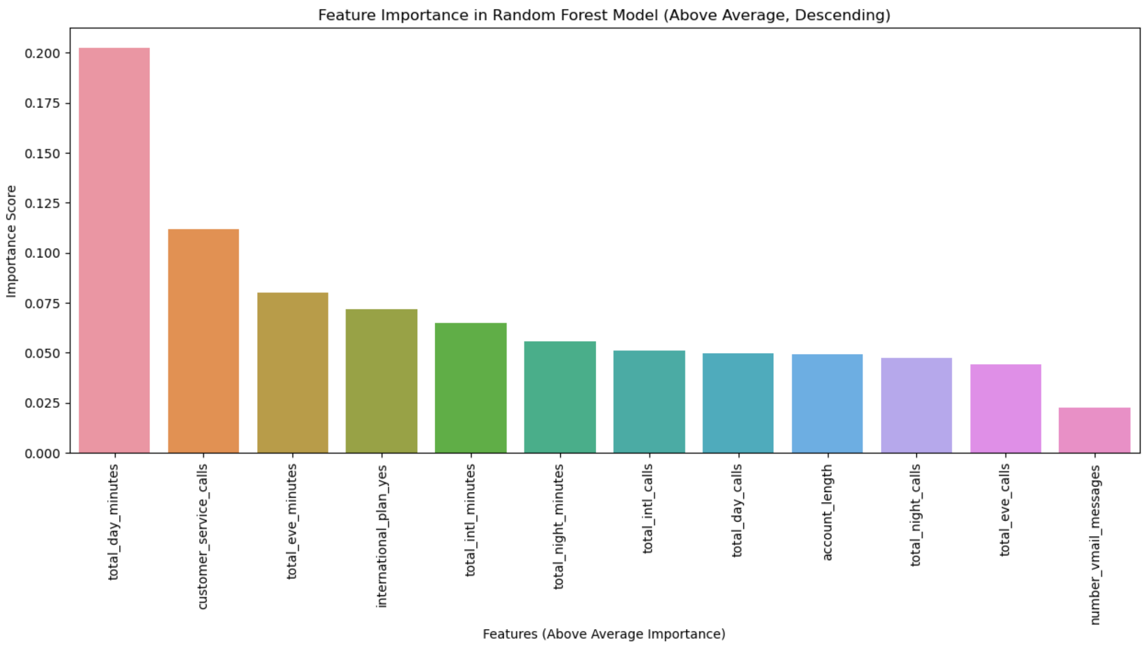
To check for important features

We use a simple bar graph to see the top features that can help determine churn rate

In [62]:
```
# first drop the 'state' columns
feature_names = df4.drop(columns=[col for col in df4.columns if 'state'
feature_names.columns
```

Out[62]:
```
Index(['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', 'total_intl_calls', 'customer_service_calls',
       'churn', 'international_plan_yes', 'voice_mail_plan_yes'],
      dtype='object')
```

In [63]:
```
# Set a larger figure size (adjust as needed)
feature_importances = clf.feature_importances_
feature_names = X_train.columns

# Calculate average importance score
avg_importance = feature_importances.mean()

# Filter features with above-average importance
important_features = feature_names[feature_importances > avg_importance
important_importances = feature_importances[feature_importances > avg_i

# Sort features and importances in descending order
sorted_idx = important_importances.argsort()[::-1]  # Reverse order for
sorted_features = important_features[sorted_idx]
sorted_importances = important_importances[sorted_idx]

# Create the bar plot
plt.figure(figsize=(15, 6))
sns.barplot(x=sorted_features, y=sorted_importances)
plt.xlabel('Features (Above Average Importance)')
plt.ylabel('Importance Score')
plt.title('Feature Importance in Random Forest Model (Above Average, De

# Rotate feature names for better readability
plt.xticks(rotation=90)

plt.show()
```



- From the graph, we can conclude the top features are:
  - Total day minutes
  - Customer service calls
  - Total eve minutes
  - Internal plan (Those that had subscribed)
  - Total minutes

Concusion

- The churn prediction analysis conducted for SyriaTel aimed to develop a classifier to identify customers likely to terminate their services. Through comprehensive data exploration, preparation, and modeling, several key findings emerged:
- Model Performance: Random Forest emerged as the most effective model for churn prediction, outperforming Logistic Regression, Decision Trees, and XGBoost. It exhibited superior accuracy and predictive power, making it the preferred choice for SyriaTel's churn prediction system.
- Key Predictive Features: Total day minutes, customer service calls, and subscription to the international plan were identified as crucial indicators of churn. These insights provide valuable guidance for SyriaTel in devising proactive retention strategies targeted at high-risk customers.### Conclusion

Random Forest model appears to be the best model to predict the customers likely to churn.

Recommendation

Based on the plot, some recommendations would be:

- Enhance Call Quality: Invest in infrastructure and technology to improve call quality, ensuring a better customer experience.
- Customer Service Improvement: Focus on enhancing customer service by reducing response times, increasing efficiency in issue resolution, and offering personalized support.
- Tailored Plans for International Subscribers: Design attractive plans and offers specifically targeted at international subscribers to increase satisfaction and reduce churn.
- Proactive Retention Strategies: Implement proactive measures such as targeted promotions, loyalty rewards, and personalized communication to retain at-risk customers.
- Regular Analysis: Continuously monitor customer behavior and churn patterns, regularly updating models and strategies to adapt to changing market dynamics.