SyriaTel Customer Churn Prediction and Retention Strategy

Author: John Mugambi

Overview ¶

SyraiTel, a telecomunications company faces challenges in customer retension as some users discontinue their services. By leveraging machine learing techniques, we aim to predict customer churn and provide strategic insights to help SyriaTel improve customer retention.

Business Problem

Customer churn impacts SyriaTel revenue. The company needs an effective way to identify customers who are likely to leave and take proactive measures to retain them. Understanding the factors influencing churn will allow SyriaTel to enhance customer satisfaction, optimize marketing strategies, and improve overall business performance.

Objectives

Develop a machine learning model to predict whether a customer is likely to churn.

Identify key factors that influence customer churn.

Provide actionable insights to SyriaTel for improving customer retention strategies

Business Questions

1. What customer behaviors or attributes are most predictive of churn?
2. How can SyriaTel proactively intervene to retain at-risk customers?
3. Which classification models perform best in predicting customer churn?
4. What strategies can be implemented to enhance customer satisfaction and reduce churn?

Data Understanding

Data Sources and Relevance

The dataset for this project comes from Kaggle and contains customer data related to SyriaTel, a telecommunications company. This dataset is highly relevant to the problem of customer churn prediction. By analyzing customer behaviors and service usage patterns, we can identify key factors influencing churn. Understanding these factors will help SyriaTel develop effective strategies to retain customers, improve satisfaction, and reduce financial losses associated with customer attrition.

In [439]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from scipy.stats import boxcox
```

In [440]:
```
# Load the data
churn_df = pd.read_csv("./data/churn_dataset.csv")
churn_df
```

Out[440]:

	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	...
...	...	...	...	...	...	...	...	...	...	...	...
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	...

3333 rows × 21 columns

In [441]:
```
churn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

In [442]:
```
churn_df.shape
```

Out[442]:
```
(3333, 21)
```

In [443]:
```
churn_df.describe()
```

Out[443]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total ev minute
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.00000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.98034
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.71384
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.60000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.40000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.30000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.70000

In [444]:
```
churn_df['state'].value_counts()
```

Out[444]:
```
WV    106
MN     84
NY     83
AL     80
OR     78
WI     78
OH     78
WY     77
VA     77
CT     74
VT     73
MI     73
ID     73
UT     72
TX     72
IN     71
MD     70
KS     70
MT     68
NC     68
NJ     68
NV     66
CO     66
WA     66
RI     65
MS     65
MA     65
AZ     64
MO     63
FL     63
ME     62
ND     62
NM     62
OK     61
DE     61
NE     61
SD     60
SC     60
KY     59
IL     58
NH     56
AR     55
DC     54
GA     54
TN     53
HI     53
AK     52
LA     51
PA     45
IA     44
CA     34
Name: state, dtype: int64
```

In [445]:
```
churn_df['international plan'].value_counts()
```

Out[445]:
```
no     3010
yes     323
Name: international plan, dtype: int64
```

In [446]:
```
churn_df['voice mail plan'].value_counts()
```

Out[446]:
```
no     2411
yes     922
Name: voice mail plan, dtype: int64
```

Data Preparation

Data Cleaning

In [447]:
```
#Format the column names removing unconventional naming ways
churn_df.columns = churn_df.columns.str.replace(' ','_')
churn_df.columns
```

Out[447]:
```
Index(['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_minutes', 'total_eve_calls', 'total_eve_charge',
       'total_night_minutes', 'total_night_calls', 'total_night_charge',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge',
       'customer_service_calls', 'churn'],
      dtype='object')
```

In [448]:
```
#Count number of missing values in the datafram
churn_df.isnull().sum()
```

Out[448]:
```
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 [449]:
```
# Check for duplicates
churn_df.duplicated().sum()
```

Out[449]:
```
0
```

In [450]:
```
# Converting the categorical columns to categorical types
churn_df["international_plan"] = churn_df["international_plan"].astype("catego
churn_df["voice_mail_plan"] = churn_df["voice_mail_plan"].astype("category")
```

In [451]:
```
# Convert the boolean to int 0/1
#churn_df["churn"] = churn_df["churn"].astype("int")
```

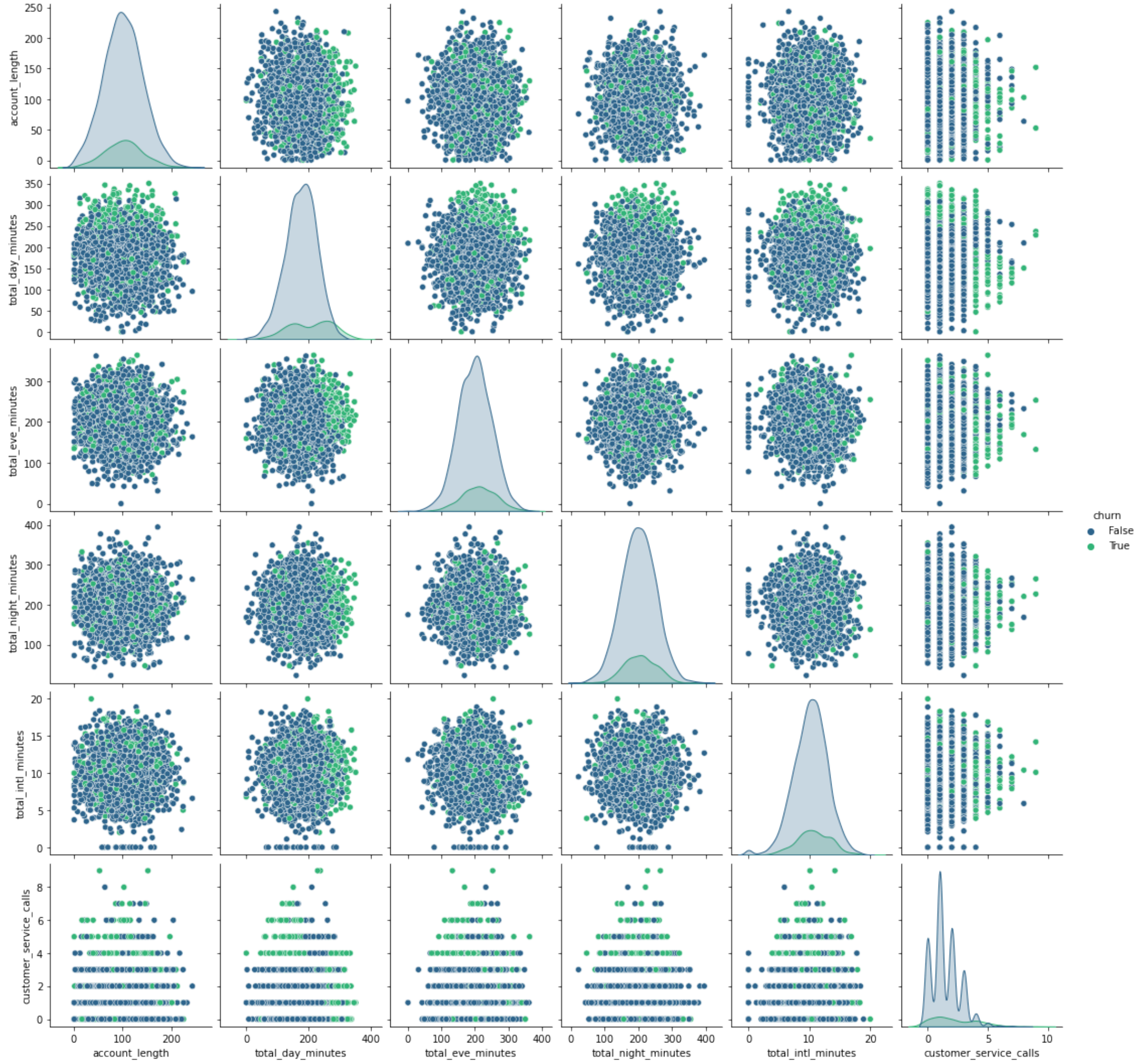
Feature Engineering

In [452]:
```
#Drop the phone number column since it may not provide usefull predictive valu
churn_df.drop(columns=["phone_number"], inplace=True)
```

In [453]:
```
#Create a new total calls feature
churn_df["total_calls"] = (
    churn_df["total_day_calls"]
    + churn_df["total_eve_calls"]
    + churn_df["total_night_calls"]
    + churn_df["total_intl_calls"]
)
```

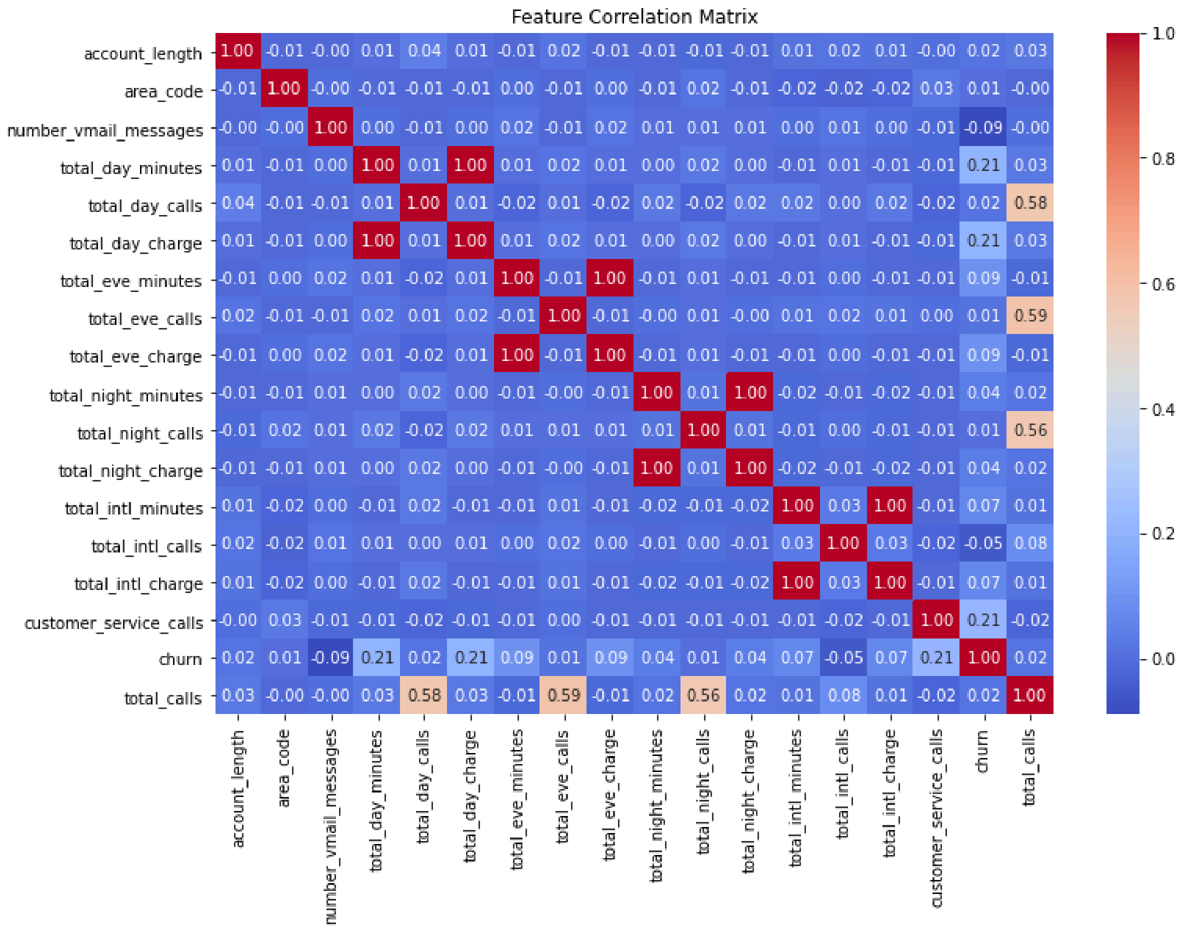
Visualization

In [454]:
```
sns.pairplot(churn_df[['state', 'account_length', 'total_day_minutes', 'total_
                        'total_night_minutes', 'total_intl_minutes', 'customer_
            hue='churn', palette='viridis', diag_kind='kde')
plt.show()
```

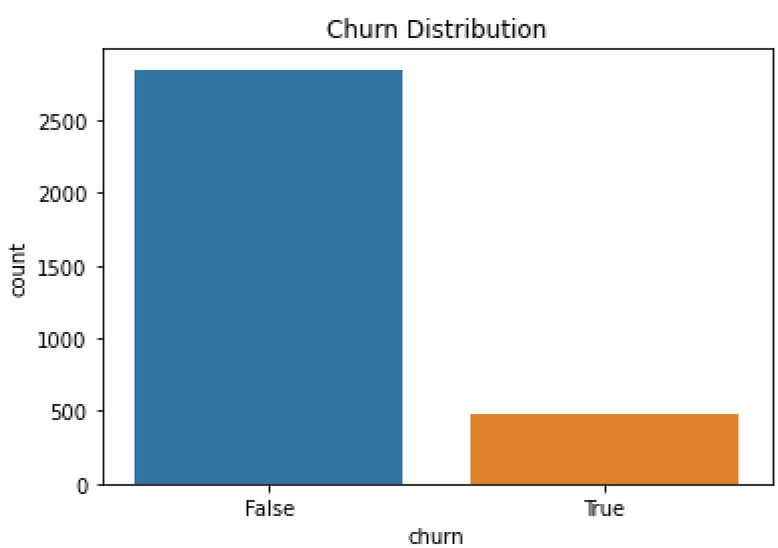


Data Modeling

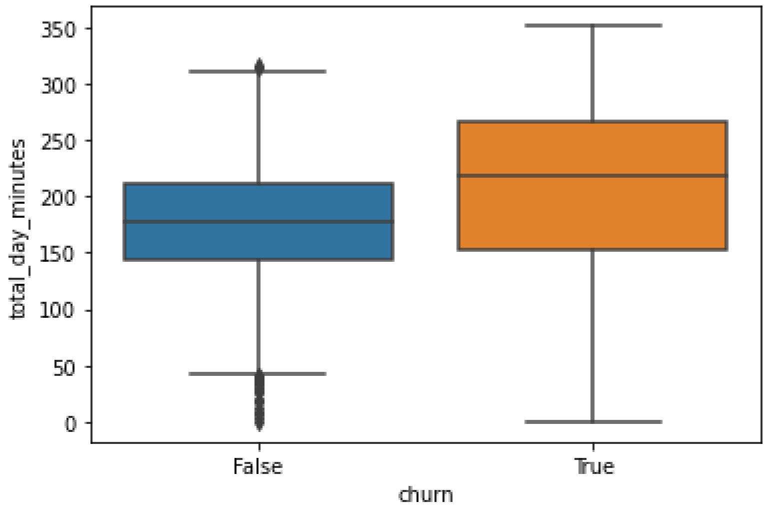
In [455]:
```
#Correlation Analysis
plt.figure(figsize=(12, 8))
sns.heatmap(churn_df.corr(), annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Feature Correlation Matrix")
plt.show()
```



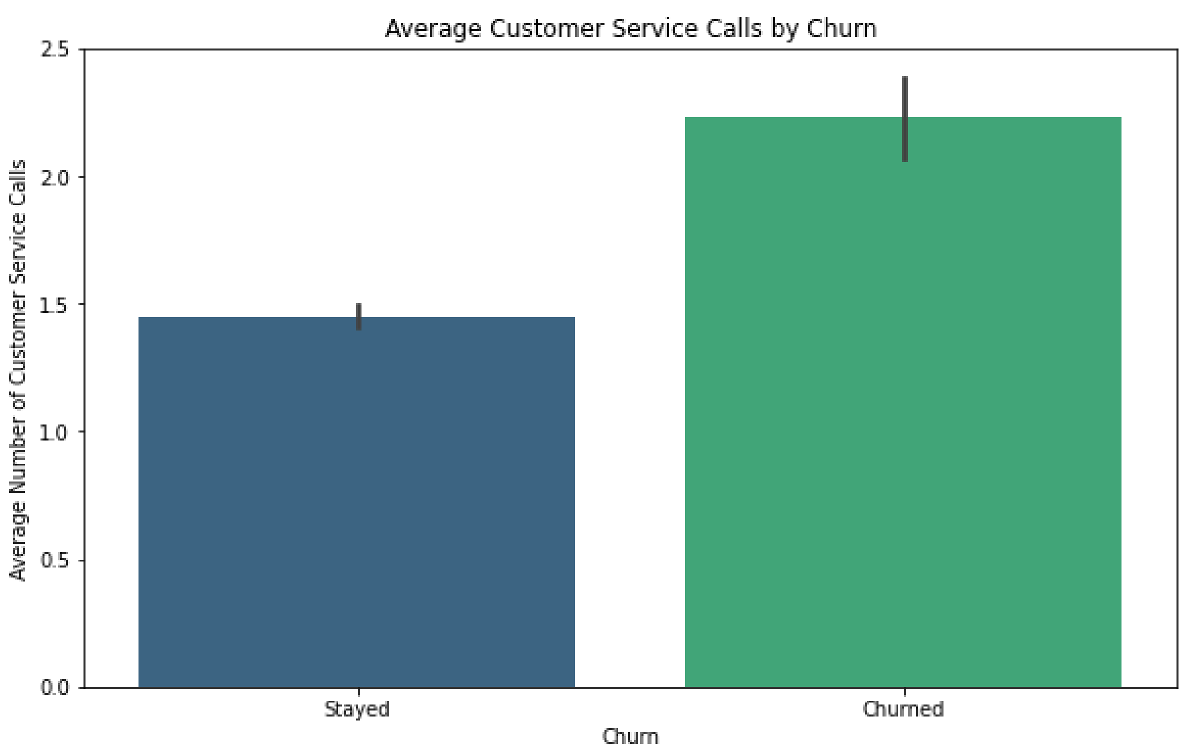
In [456]:
```
#Check the churn distribution
sns.countplot(x=churn_df["churn"])
plt.title("Churn Distribution")
plt.show()
```



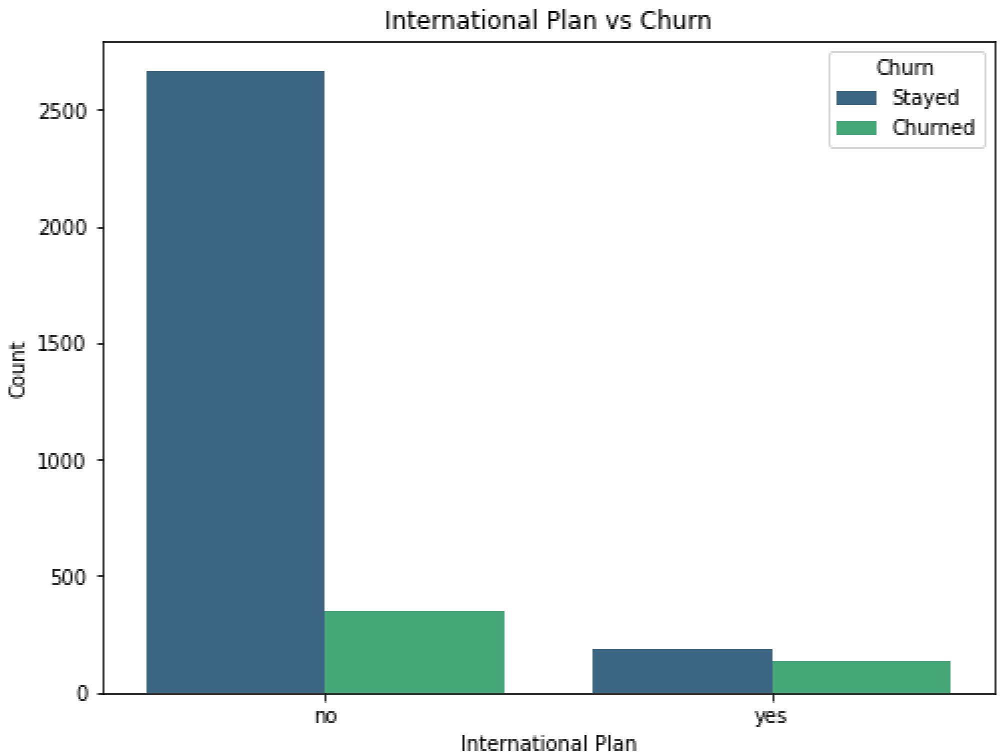
In [457]:
```
#Boxplots for Numerical Variables:
sns.boxplot(x="churn", y="total_day_minutes", data=churn_df)
plt.show()
```



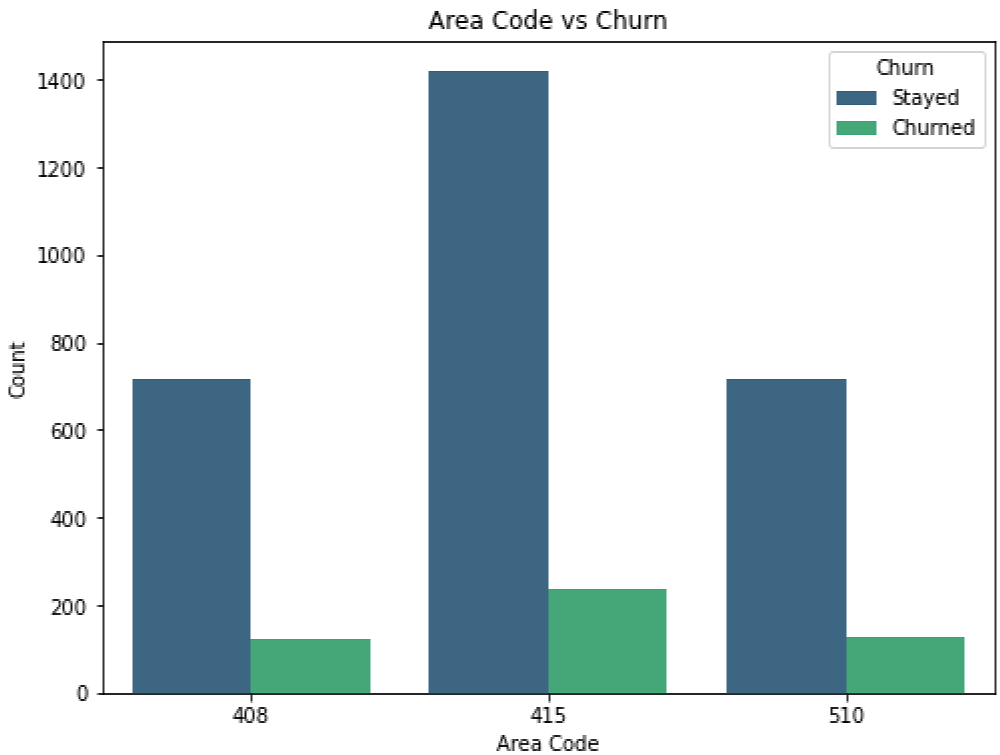
In [458]:
```
#Visualize if customers service calls relate to churn
plt.figure(figsize=(10, 6))
sns.barplot(x='churn', y='customer_service_calls', data=churn_df, palette='vir
plt.title('Average Customer Service Calls by Churn')
plt.xlabel('Churn')
plt.ylabel('Average Number of Customer Service Calls')
plt.xticks([0, 1], ['Stayed', 'Churned'])
plt.show()
```



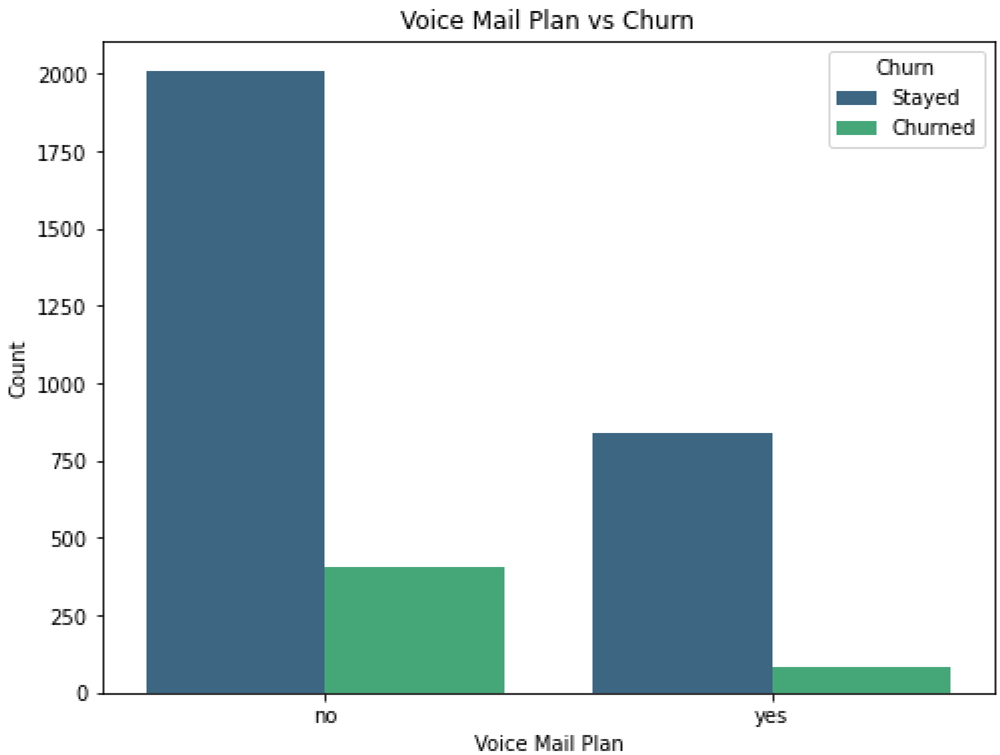
In [459]:
```
plt.figure(figsize=(8, 6))
sns.countplot(x='international_plan', hue='churn', data=churn_df, palette='vir
plt.title('International Plan vs Churn')
plt.xlabel('International Plan')
plt.ylabel('Count')
plt.legend(title='Churn', labels=['Stayed', 'Churned'])
plt.show()
```



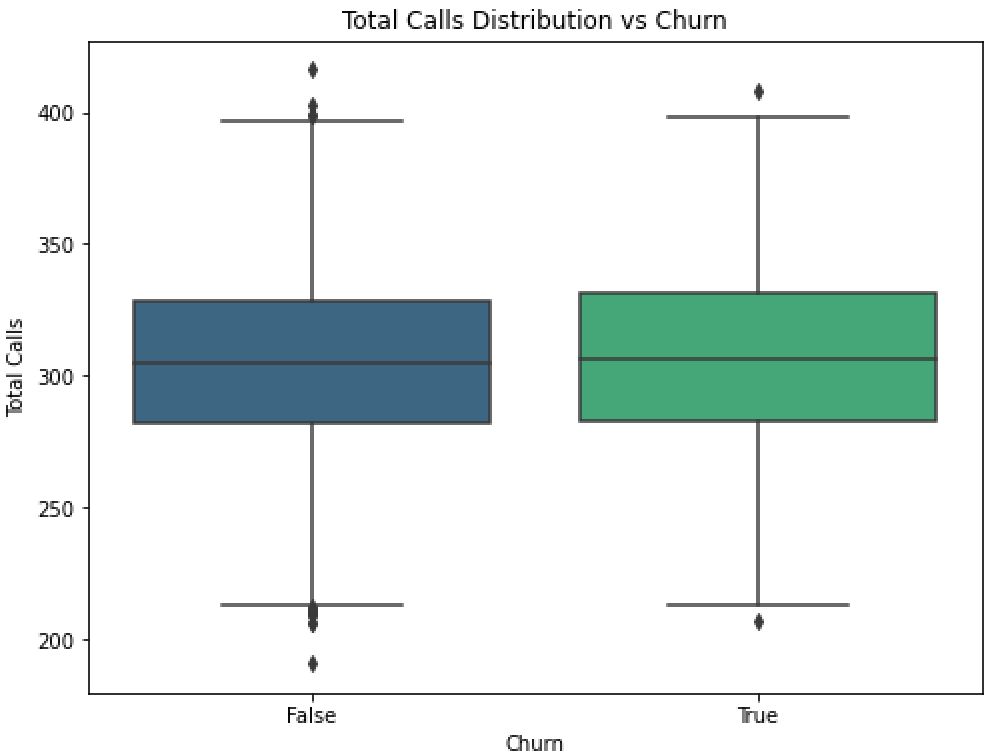
In [460]:
```
plt.figure(figsize=(8, 6))
sns.countplot(x='area_code', hue='churn', data=churn_df, palette='viridis')
plt.title('Area Code vs Churn')
plt.xlabel('Area Code')
plt.ylabel('Count')
plt.legend(title='Churn', labels=['Stayed', 'Churned'])
plt.show()
```



In [461]:
```
plt.figure(figsize=(8, 6))
sns.countplot(x='voice_mail_plan', hue='churn', data=churn_df, palette='viridi
plt.title('Voice Mail Plan vs Churn')
plt.xlabel('Voice Mail Plan')
plt.ylabel('Count')
plt.legend(title='Churn', labels=['Stayed', 'Churned'])
plt.show()
```



In [462]:
```
#total calls distribution
plt.figure(figsize=(8, 6))
sns.boxplot(x='churn', y='total_calls', data=churn_df, palette='viridis')
plt.title('Total Calls Distribution vs Churn')
plt.xlabel('Churn')
plt.ylabel('Total Calls')
plt.show()
```



Preprocessing

In [463]:
```
churn_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   international_plan      3333 non-null   category
 4   voice_mail_plan         3333 non-null   category
 5   number_vmail_messages   3333 non-null   int64   
 6   total_day_minutes       3333 non-null   float64 
 7   total_day_calls         3333 non-null   int64   
 8   total_day_charge        3333 non-null   float64 
 9   total_eve_minutes       3333 non-null   float64 
 10  total_eve_calls         3333 non-null   int64   
 11  total_eve_charge        3333 non-null   float64 
 12  total_night_minutes     3333 non-null   float64 
 13  total_night_calls       3333 non-null   int64   
 14  total_night_charge      3333 non-null   float64 
 15  total_intl_minutes      3333 non-null   float64 
 16  total_intl_calls        3333 non-null   int64   
 17  total_intl_charge       3333 non-null   float64 
 18  customer_service_calls  3333 non-null   int64   
 19  churn                   3333 non-null   bool    
 20  total_calls             3333 non-null   int64   
dtypes: bool(1), category(2), float64(8), int64(9), object(1)
memory usage: 478.8+ KB
```

In [464]:
```
# get the numeric cols and the cat cols
num_original_columns = churn_df.select_dtypes(include=np.number).columns.tolis
categorical_cols = churn_df.select_dtypes(exclude=np.number).columns.tolist()

print("Categorical columns:", categorical_cols)
num_original_columns
```

```
Categorical columns: ['state', 'international_plan', 'voice_mail_plan', 'chur
n']
```

Out[464]:
```
['account_length',
 'area_code',
 'number_vmail_messages',
 'total_day_minutes',
 'total_day_calls',
 'total_day_charge',
 'total_eve_minutes',
 'total_eve_calls',
 'total_eve_charge',
 'total_night_minutes',
 'total_night_calls',
 'total_night_charge',
 'total_intl_minutes',
 'total_intl_calls',
 'total_intl_charge',
 'customer_service_calls',
 'total_calls']
```

Encoding

In [465]:
```
churn_df[categorical_cols].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   state               3333 non-null   object
 1   international_plan  3333 non-null   category
 2   voice_mail_plan     3333 non-null   category
 3   churn               3333 non-null   bool
dtypes: bool(1), category(2), object(1)
memory usage: 36.1+ KB
```

In [466]:
```
for i in categorical_cols:
  print(f'The variable "{i}" has {churn_df[i].nunique()} variables: {churn_df[
```

```
The variable "state" has 51 variables: ['KS' 'OH' 'NJ' 'OK' 'AL' 'MA' 'MO' 'L
A' 'WV' 'IN' 'RI' 'IA' 'MT' 'NY'
 'ID' 'VT' 'VA' 'TX' 'FL' 'CO' 'AZ' 'SC' 'NE' 'WY' 'HI' 'IL' 'NH' 'GA'
 'AK' 'MD' 'AR' 'WI' 'OR' 'MI' 'DE' 'UT' 'CA' 'MN' 'SD' 'NC' 'WA' 'NM'
 'NV' 'DC' 'KY' 'ME' 'MS' 'TN' 'PA' 'CT' 'ND']

The variable "international_plan" has 2 variables: ['no', 'yes']
Categories (2, object): ['no', 'yes']

The variable "voice_mail_plan" has 2 variables: ['yes', 'no']
Categories (2, object): ['yes', 'no']

The variable "churn" has 2 variables: [False  True]
```

In [467]:
```
# Initialize Label Encoder
le = LabelEncoder()
```

In [468]:
```
# Apply Label Encoding to binary categorical columns
churn_df["international_plan"] = le.fit_transform(churn_df["international_plan
churn_df["voice_mail_plan"] = le.fit_transform(churn_df["voice_mail_plan"])
```

In [469]:
```
# One-Hot Encode 'state' column, adn drop it from the datafram
churn_df = pd.get_dummies(churn_df, columns=["state"], drop_first=True)
```

In [470]:
```
churn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 70 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   account_length          3333 non-null   int64  
 1   area_code               3333 non-null   int64  
 2   international_plan      3333 non-null   int32  
 3   voice_mail_plan         3333 non-null   int32  
 4   number_vmail_messages   3333 non-null   int64  
 5   total_day_minutes       3333 non-null   float64
 6   total_day_calls         3333 non-null   int64  
 7   total_day_charge        3333 non-null   float64
 8   total_eve_minutes       3333 non-null   float64
 9   total_eve_calls         3333 non-null   int64  
 10  total_eve_charge        3333 non-null   float64
 11  total_night_minutes     3333 non-null   float64
 12  total_night_calls       3333 non-null   int64  
 13  total_night_charge      3333 non-null   float64
 14  total_intl_minutes      3333 non-null   float64
 15  total_intl_calls        3333 non-null   int64  
 16  total_intl_charge       3333 non-null   float64
 17  customer_service_calls  3333 non-null   int64  
 18  churn                   3333 non-null   bool   
 19  total_calls             3333 non-null   int64  
 20  state_AL                3333 non-null   uint8  
 21  state_AR                3333 non-null   uint8  
 22  state_AZ                3333 non-null   uint8  
 23  state_CA                3333 non-null   uint8  
 24  state_CO                3333 non-null   uint8  
 25  state_CT                3333 non-null   uint8  
 26  state_DC                3333 non-null   uint8  
 27  state_DE                3333 non-null   uint8  
 28  state_FL                3333 non-null   uint8  
 29  state_GA                3333 non-null   uint8  
 30  state_HI                3333 non-null   uint8  
 31  state_IA                3333 non-null   uint8  
 32  state_ID                3333 non-null   uint8  
 33  state_IL                3333 non-null   uint8  
 34  state_IN                3333 non-null   uint8  
 35  state_KS                3333 non-null   uint8  
 36  state_KY                3333 non-null   uint8  
 37  state_LA                3333 non-null   uint8  
 38  state_MA                3333 non-null   uint8  
 39  state_MD                3333 non-null   uint8  
 40  state_ME                3333 non-null   uint8  
 41  state_MI                3333 non-null   uint8  
 42  state_MN                3333 non-null   uint8  
 43  state_MO                3333 non-null   uint8  
 44  state_MS                3333 non-null   uint8  
 45  state_MT                3333 non-null   uint8  
 46  state_NC                3333 non-null   uint8  
 47  state_ND                3333 non-null   uint8  
 48  state_NE                3333 non-null   uint8  
 49  state_NH                3333 non-null   uint8  
 50  state_NJ                3333 non-null   uint8  
 51  state_NM                3333 non-null   uint8  
```

```
 52  state_NV                3333 non-null   uint8
 53  state_NY                3333 non-null   uint8
 54  state_OH                3333 non-null   uint8
 55  state_OK                3333 non-null   uint8
 56  state_OR                3333 non-null   uint8
 57  state_PA                3333 non-null   uint8
 58  state_RI                3333 non-null   uint8
 59  state_SC                3333 non-null   uint8
 60  state_SD                3333 non-null   uint8
 61  state_TN                3333 non-null   uint8
 62  state_TX                3333 non-null   uint8
 63  state_UT                3333 non-null   uint8
 64  state_VA                3333 non-null   uint8
 65  state_VT                3333 non-null   uint8
 66  state_WA                3333 non-null   uint8
 67  state_WI                3333 non-null   uint8
 68  state_WV                3333 non-null   uint8
 69  state_WY                3333 non-null   uint8
dtypes: bool(1), float64(8), int32(2), int64(9), uint8(50)
memory usage: 634.8 KB
```

In [471]:
```
churn_df.head()
```

Out[471]:

	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_
0	128	415	0	1	25	
1	107	415	0	1	26	
2	137	415	0	0	0	
3	84	408	1	0	0	
4	75	415	1	0	0	

5 rows × 70 columns

Scaling

In [472]:
```
scaler = StandardScaler()
```

In [473]:
```
# Select numerical columns without the encoded categorical ones
numerical_cols = churn_df.select_dtypes(include=np.number).columns
```

In [474]:
```
#Check Skewness
for col in num_original_columns:
    skew_value = churn_df[col].skew()
    print(f"Column: {col}, Skewness: {skew_value:.4f}")
```

```
Column: account_length, Skewness: 0.0966
Column: area_code, Skewness: 1.1268
Column: number_vmail_messages, Skewness: 1.2648
Column: total_day_minutes, Skewness: -0.0291
Column: total_day_calls, Skewness: -0.1118
Column: total_day_charge, Skewness: -0.0291
Column: total_eve_minutes, Skewness: -0.0239
Column: total_eve_calls, Skewness: -0.0556
Column: total_eve_charge, Skewness: -0.0239
Column: total_night_minutes, Skewness: 0.0089
Column: total_night_calls, Skewness: 0.0325
Column: total_night_charge, Skewness: 0.0089
Column: total_intl_minutes, Skewness: -0.2451
Column: total_intl_calls, Skewness: 1.3215
Column: total_intl_charge, Skewness: -0.2453
Column: customer_service_calls, Skewness: 1.0914
Column: total_calls, Skewness: -0.0376
```

BAsed on the abovethe following are Highly skewed (consider transformation):

- area_code (1.1268)
- number_vmail_messages (1.2648)
- total_intl_calls (1.3215)
- customer_service_calls (1.0914)
- churn (2.0184)

Moderately skewed:

- total_intl_minutes (-0.2451)
- total_intl_charge (-0.2453)

In [475]:
```
# Box-Cox transformation was chosen for right-skewed
# data like number_vmail_messages,total_intl_calls,customer_service_calls
# i.e values greater that 1

for col in ["number_vmail_messages", "total_intl_calls", "customer_service_cal
    churn_df[col], _ = boxcox(churn_df[col] + 1)
```

In [476]:
```
# # Log transform 'churn' - has binary values
# churn_df["churn"] = np.log1p(churn_df["churn"])
```

In [477]:
```
#Confirm skewness has improved
for col in ["number_vmail_messages", "total_intl_calls", "customer_service_cal
    print(f"Column: {col}, Skewness: {churn_df[col].skew()}")
```

```
Column: number_vmail_messages, Skewness: 1.0002251990902429
Column: total_intl_calls, Skewness: 0.005816369249351684
Column: customer_service_calls, Skewness: -0.013769770295669267
```

In [478]:
```
# Apply scaling
churn_df_scaled = churn_df.copy()
churn_df_scaled[numerical_cols] = scaler.fit_transform(churn_df[numerical_cols
churn_df_scaled.head()
```

Out[478]:

	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_
0	0.676489	-0.523603	-0.327580	1.617086	1.608479	
1	0.149065	-0.523603	-0.327580	1.617086	1.612570	
2	0.902529	-0.523603	-0.327580	-0.618396	-0.618292	
3	-0.428590	-0.688834	3.052685	-0.618396	-0.618292	
4	-0.654629	-0.523603	3.052685	-0.618396	-0.618292	

5 rows × 70 columns

In [479]:
```
churn_df[numerical_cols].describe()
churn_df_scaled['churn']

numerical_cols
```

Out[479]:
```
Index(['account_length', 'area_code', 'international_plan', 'voice_mail_pla
n',
       'number_vmail_messages', 'total_day_minutes', 'total_day_calls',
       'total_day_charge', 'total_eve_minutes', 'total_eve_calls',
       'total_eve_charge', 'total_night_minutes', 'total_night_calls',
       'total_night_charge', 'total_intl_minutes', 'total_intl_calls',
       'total_intl_charge', 'customer_service_calls', 'total_calls',
       'state_AL', 'state_AR', 'state_AZ', 'state_CA', 'state_CO', 'state_C
T',
       'state_DC', 'state_DE', 'state_FL', 'state_GA', 'state_HI', 'state_I
A',
       'state_ID', 'state_IL', 'state_IN', 'state_KS', 'state_KY', 'state_L
A',
       'state_MA', 'state_MD', 'state_ME', 'state_MI', 'state_MN', 'state_M
O',
       'state_MS', 'state_MT', 'state_NC', 'state_ND', 'state_NE', 'state_N
H',
       'state_NJ', 'state_NM', 'state_NV', 'state_NY', 'state_OH', 'state_O
K',
       'state_OR', 'state_PA', 'state_RI', 'state_SC', 'state_SD', 'state_T
N',
       'state_TX', 'state_UT', 'state_VA', 'state_VT', 'state_WA', 'state_W
I',
       'state_WV', 'state_WY'],
      dtype='object')
```

Modeling

In [480]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import statsmodels.api as sm

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,
from sklearn.metrics import mean_squared_error, r2_score
```

In [481]:
```
# Defining dependent and independent variable
X = churn_df[numerical_cols]
y = churn_df['churn']

X = sm.add_constant(X)
model = sm.OLS(y, X).fit()

print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  churn   R-squared:                       0.18
5
Model:                            OLS   Adj. R-squared:                  0.16
8
Method:                 Least Squares   F-statistic:                     10.7
2
Date:                Sat, 08 Mar 2025   Prob (F-statistic):          6.20e-10
0
Time:                        20:25:06   Log-Likelihood:                -908.9
4
No. Observations:                3333   AIC:                             195
8.
Df Residuals:                    3263   BIC:                             238
6.
Df Model:                          69
Covariance Type:            nonrobust
==============================================================================
=============
                            coef    std err          t      P>|t|      [0.02
5      0.975]
------------------------------------------------------------------------------
-------------
const                    -0.3711      0.103     -3.610      0.000      -0.57
3      -0.170
account_length            0.0001      0.000      1.055      0.291      -0.00
0       0.000
area_code             -8.911e-05      0.000     -0.671      0.502      -0.00
0       0.000
international_plan        0.3056      0.019     16.008      0.000       0.26
8       0.343
voice_mail_plan          -1.0226      0.685     -1.494      0.135      -2.36
5       0.320
number_vmail_messages     0.9003      0.654      1.376      0.169      -0.38
2       2.183
total_day_minutes         0.0369      0.333      0.111      0.912      -0.61
7       0.691
total_day_calls          -0.0216      0.008     -2.560      0.010      -0.03
8      -0.005
total_day_charge         -0.2100      1.962     -0.107      0.915      -4.05
6       3.636
total_eve_minutes         0.0842      0.166      0.509      0.611      -0.24
0       0.409
total_eve_calls          -0.0219      0.008     -2.591      0.010      -0.03
8      -0.005
total_eve_charge         -0.9828      1.947     -0.505      0.614      -4.80
1       2.835
total_night_minutes      -0.0544      0.088     -0.615      0.539      -0.22
8       0.119
total_night_calls        -0.0220      0.008     -2.604      0.009      -0.03
9      -0.005
total_night_charge        1.2162      1.966      0.619      0.536      -2.63
9       5.072
total_intl_minutes       -0.4130      0.529     -0.780      0.435      -1.45
1       0.625
```

```
total_intl_calls          -0.1425      0.038     -3.731      0.000      -0.21
7      -0.068
total_intl_charge          1.5610      1.960      0.796      0.426      -2.28
2       5.404
customer_service_calls     0.0993      0.010     10.120      0.000       0.08
0       0.119
total_calls                0.0220      0.008      2.604      0.009       0.00
5       0.038
state_AL                   0.0146      0.057      0.255      0.798      -0.09
8       0.127
state_AR                   0.0837      0.062      1.344      0.179      -0.03
8       0.206
state_AZ                   0.0055      0.060      0.091      0.927      -0.11
2       0.123
state_CA                   0.1738      0.071      2.447      0.014       0.03
5       0.313
state_CO                   0.0473      0.060      0.793      0.428      -0.07
0       0.164
state_CT                   0.0711      0.058      1.220      0.223      -0.04
3       0.185
state_DC                   0.0224      0.063      0.358      0.720      -0.10
0       0.145
state_DE                   0.0324      0.061      0.532      0.594      -0.08
7       0.152
state_FL                   0.0260      0.060      0.430      0.667      -0.09
3       0.145
state_GA                   0.0502      0.063      0.802      0.423      -0.07
3       0.173
state_HI                  -0.0091      0.063     -0.144      0.885      -0.13
2       0.114
state_IA                   0.0204      0.066      0.309      0.757      -0.10
9       0.150
state_ID                   0.0505      0.058      0.865      0.387      -0.06
4       0.165
state_IL                  -0.0245      0.062     -0.397      0.691      -0.14
5       0.096
state_IN                   0.0208      0.059      0.354      0.723      -0.09
5       0.136
state_KS                   0.0707      0.059      1.199      0.230      -0.04
5       0.186
state_KY                   0.0602      0.061      0.984      0.325      -0.06
0       0.180
state_LA                   0.0230      0.063      0.362      0.717      -0.10
1       0.147
state_MA                   0.0832      0.060      1.388      0.165      -0.03
4       0.201
state_MD                   0.1111      0.059      1.883      0.060      -0.00
5       0.227
state_ME                   0.1113      0.061      1.837      0.066      -0.00
7       0.230
state_MI                   0.1156      0.058      1.977      0.048       0.00
1       0.230
state_MN                   0.0856      0.057      1.506      0.132      -0.02
6       0.197
state_MO                   0.0332      0.060      0.551      0.582      -0.08
5       0.152
state_MS                   0.1163      0.060      1.942      0.052      -0.00
```

```
1       0.234
state_MT                  0.1522      0.059      2.568      0.010       0.03
6       0.268
state_NC                  0.0446      0.059      0.751      0.453      -0.07
2       0.161
state_ND                 -0.0138      0.061     -0.227      0.820      -0.13
3       0.105
state_NE                  0.0099      0.061      0.162      0.871      -0.10
9       0.129
state_NH                  0.0884      0.062      1.427      0.154      -0.03
3       0.210
state_NJ                  0.1520      0.059      2.561      0.010       0.03
6       0.268
state_NM                 -0.0017      0.061     -0.027      0.978      -0.12
1       0.117
state_NV                  0.0963      0.060      1.613      0.107      -0.02
1       0.213
state_NY                  0.0890      0.057      1.564      0.118      -0.02
3       0.201
state_OH                  0.0393      0.058      0.682      0.495      -0.07
4       0.152
state_OK                  0.0542      0.061      0.891      0.373      -0.06
5       0.174
state_OR                  0.0474      0.058      0.823      0.411      -0.06
6       0.160
state_PA                  0.0788      0.066      1.202      0.229      -0.05
0       0.207
state_RI                  0.0095      0.060      0.159      0.874      -0.10
8       0.127
state_SC                  0.1841      0.061      3.015      0.003       0.06
4       0.304
state_SD                  0.0445      0.061      0.730      0.465      -0.07
5       0.164
state_TN                  0.0095      0.063      0.151      0.880      -0.11
4       0.133
state_TX                  0.1556      0.059      2.657      0.008       0.04
1       0.270
state_UT                  0.0703      0.059      1.202      0.229      -0.04
4       0.185
state_VA                 -0.0429      0.058     -0.742      0.458      -0.15
6       0.071
state_VT                  0.0078      0.058      0.134      0.894      -0.10
7       0.122
state_WA                  0.1316      0.060      2.204      0.028       0.01
5       0.249
state_WI                  0.0129      0.058      0.223      0.823      -0.10
0       0.126
state_WV                  0.0311      0.055      0.571      0.568      -0.07
6       0.138
state_WY                  0.0165      0.058      0.285      0.775      -0.09
7       0.130
==============================================================================
=
Omnibus:                      844.862   Durbin-Watson:                   1.96
8
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1690.78
8
```

```
Skew:                           1.531   Prob(JB):                        0.0
0
Kurtosis:                       4.673   Cond. No.                    2.43e+0
5
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 2.43e+05. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

In [482]:

```python
# Make predictions
y_pred = model.predict(X)

# Calculate the metrics
rmse = np.sqrt(mean_squared_error(y, y_pred))
r2 = r2_score(y, y_pred)
mae = mean_absolute_error(y, y_pred)

print(f"\nRMSE: {rmse}")
print(f"R-squared: {r2}")
print(f"Mean Absolute Error: {mae}")
```

```
RMSE: 0.3178328250152115
R-squared: 0.1847775467138406
Mean Absolute Error: 0.22132800557877938
```

While the model seems to fit perfectly ($R^2 = 1.0$, RMSE close to zero)

The F-statistic is 7.63e+27, which is extremely high, and the F-statistic= 0.00, indicates that the overall model is statistically significant.

The RMSE value is very low suggesting that the model’s predictions are very close to the actual values.

The skewness value is -0.042, which is very close to 0, therefore meaning the model doesn't have large biases.

Classification

Logistic Regression

In [483]:
```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = churn_df_scaled.drop(columns=['churn'])
y = churn_df_scaled['churn']

# Split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

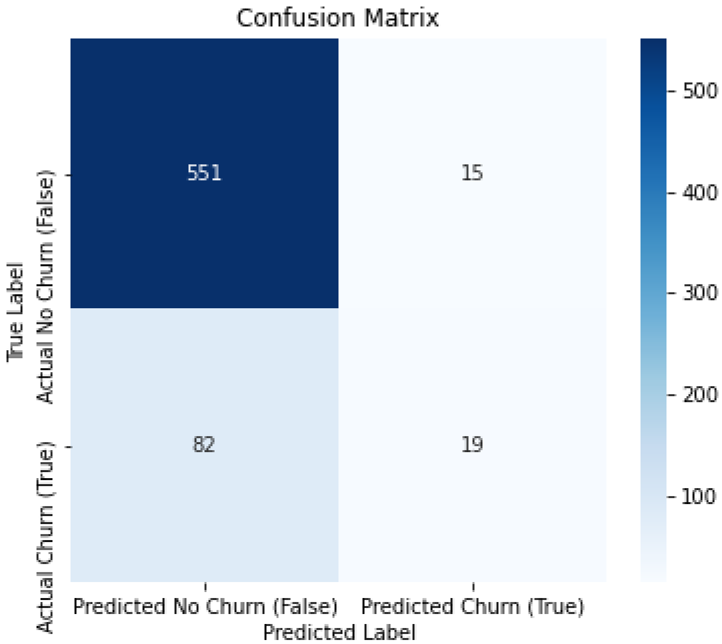
In [484]:
```
print(y.unique())
churn_df_scaled['churn']
```

```
[False  True]
```

Out[484]:
```
0       False
1       False
2       False
3       False
4       False
        ...
3328    False
3329    False
3330    False
3331    False
3332    False
Name: churn, Length: 3333, dtype: bool
```

In [485]:
```python
# Initialize, train logistic regression model
logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train, y_train)

y_pred = logreg_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print('Model 1 - Logistic Regression')
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")

# Classification report
print(classification_report(y_test, y_pred))

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted No Churn (False)', 'Predicted Churn (True)
            yticklabels=['Actual No Churn (False)', 'Actual Churn (True)'])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

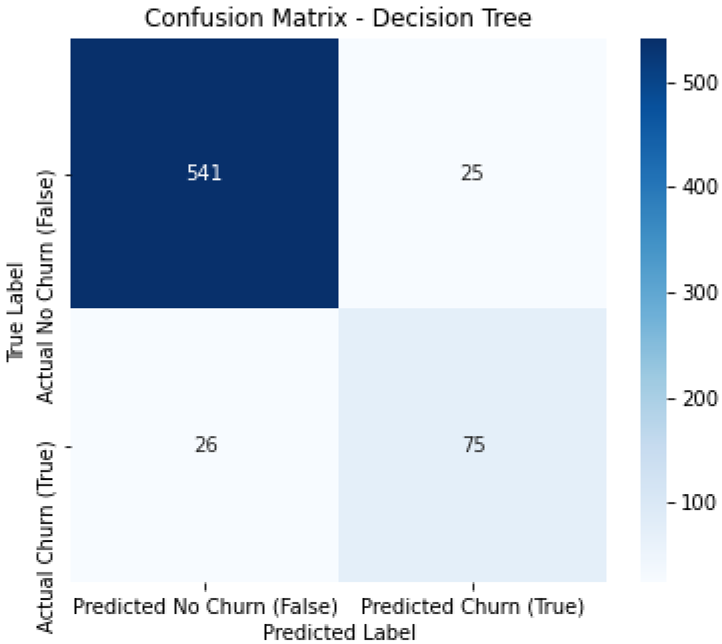
```
Model 1 - Logistic Regression
Accuracy: 0.8545727136431784
Recall: 0.18811881188118812
Precision: 0.5588235294117647
F1-score: 0.2814814814814815
              precision    recall  f1-score   support

       False       0.87      0.97      0.92       566
        True       0.56      0.19      0.28       101

    accuracy                           0.85       667
   macro avg       0.71      0.58      0.60       667
weighted avg       0.82      0.85      0.82       667
```



Recall is quite low at 0.19, indicating that many customers who actually churned were not identified by the model. We will try other classification algorithms that might handle the imbalance better

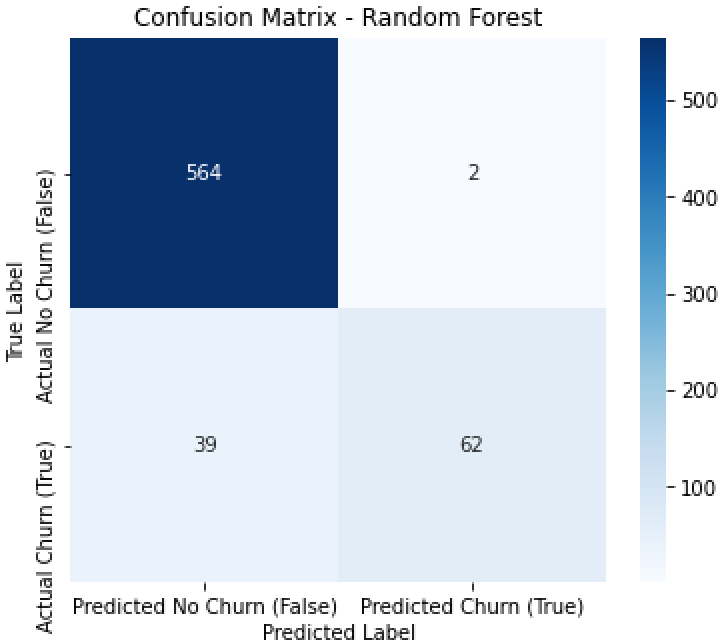
In [486]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_
import seaborn as sns
import matplotlib.pyplot as plt

# Initialize, train Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt = dt_model.predict(X_test)

# Evaluate the model
accuracy_dt = accuracy_score(y_test, y_pred_dt)
recall_dt = recall_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt)
f1_dt = f1_score(y_test, y_pred_dt)

print('Model 2 - Decision Tree Classifier')
print(f"Accuracy: {accuracy_dt}")
print(f"Recall: {recall_dt}")
print(f"Precision: {precision_dt}")
print(f"F1-score: {f1_dt}")

# Classification report
print(classification_report(y_test, y_pred_dt))

# Plot confusion matrix
cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(6,5))
sns.heatmap(cm_dt, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted No Churn (False)', 'Predicted Churn (True)
            yticklabels=['Actual No Churn (False)', 'Actual Churn (True)'])
plt.title("Confusion Matrix - Decision Tree")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Model 2 - Decision Tree Classifier
Accuracy: 0.9235382308845578
Recall: 0.7425742574257426
Precision: 0.75
F1-score: 0.746268656716418
              precision    recall  f1-score   support

       False       0.95      0.96      0.95       566
        True       0.75      0.74      0.75       101

    accuracy                           0.92       667
   macro avg       0.85      0.85      0.85       667
weighted avg       0.92      0.92      0.92       667
```


Confusion Matrix - Decision Tree
541
25
26
75
True Label
Actual No Churn (False)
Actual Churn (True)
Predicted No Churn (False)
Predicted Churn (True)
Predicted Label
500
400
300
200
100

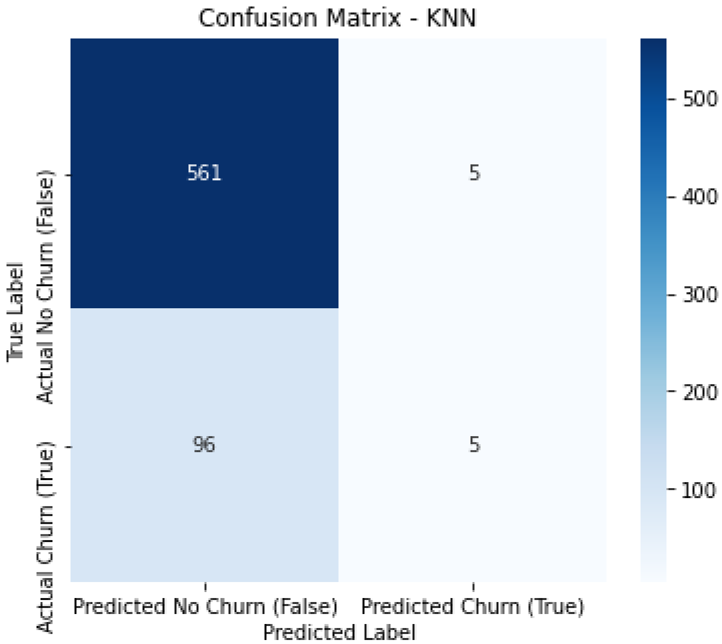
In [487]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize, train Random Forest model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

print('Model 3 - Random Forest Classifier')
print(f"Accuracy: {accuracy_rf}")
print(f"Recall: {recall_rf}")
print(f"Precision: {precision_rf}")
print(f"F1-score: {f1_rf}")

# Classification report
print(classification_report(y_test, y_pred_rf))

# Plot confusion matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(6,5))
sns.heatmap(cm_rf, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted No Churn (False)', 'Predicted Churn (True)
            yticklabels=['Actual No Churn (False)', 'Actual Churn (True)'])
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Model 3 - Random Forest Classifier
Accuracy: 0.9385307346326837
Recall: 0.6138613861386139
Precision: 0.96875
F1-score: 0.7515151515151515
              precision    recall  f1-score   support

       False       0.94      1.00      0.96       566
        True       0.97      0.61      0.75       101

    accuracy                           0.94       667
   macro avg       0.95      0.81      0.86       667
weighted avg       0.94      0.94      0.93       667
```


Confusion Matrix - Random Forest
564
2
39
62
True Label
Actual No Churn (False)
Actual Churn (True)
Predicted No Churn (False)
Predicted Churn (True)
Predicted Label
500
400
300
200
100

In [488]:
```
from sklearn.neighbors import KNeighborsClassifier

# Initialize, train KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

y_pred_knn = knn_model.predict(X_test)

# Evaluate the model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
recall_knn = recall_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)

print('Model 4 - KNN Classifier')
print(f"Accuracy: {accuracy_knn}")
print(f"Recall: {recall_knn}")
print(f"Precision: {precision_knn}")
print(f"F1-score: {f1_knn}")

# Classification report
print(classification_report(y_test, y_pred_knn))

# Plot confusion matrix
cm_knn = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(6,5))
sns.heatmap(cm_knn, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted No Churn (False)', 'Predicted Churn (True)
            yticklabels=['Actual No Churn (False)', 'Actual Churn (True)'])
plt.title("Confusion Matrix - KNN")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

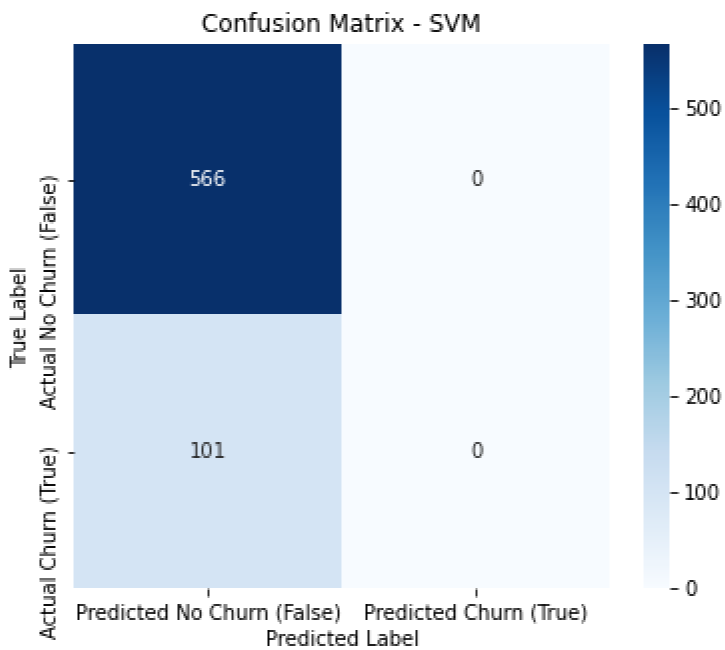
```
Model 4 - KNN Classifier
Accuracy: 0.848575712143928
Recall: 0.04950495049504951
Precision: 0.5
F1-score: 0.09009009009009009
              precision    recall  f1-score   support

       False       0.85      0.99      0.92       566
        True       0.50      0.05      0.09       101

    accuracy                           0.85       667
   macro avg       0.68      0.52      0.50       667
weighted avg       0.80      0.85      0.79       667
```


Confusion Matrix - KNN
True Label
Actual No Churn (False)
Actual Churn (True)
561
5
96
5
Predicted No Churn (False)
Predicted Churn (True)
Predicted Label
500
400
300
200
100

In [489]:
```
from sklearn.svm import SVC

# Initialize, train SVM model
svm_model = SVC(kernel='linear', random_state=42)  # You can also experiment w
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_svm = svm_model.predict(X_test)

# Evaluate the model
accuracy_svm = accuracy_score(y_test, y_pred_svm)
recall_svm = recall_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)

print('Model 5 - SVM Classifier')
print(f"Accuracy: {accuracy_svm}")
print(f"Recall: {recall_svm}")
print(f"Precision: {precision_svm}")
print(f"F1-score: {f1_svm}")

# Classification report
print(classification_report(y_test, y_pred_svm))

# Plot confusion matrix
cm_svm = confusion_matrix(y_test, y_pred_svm)
plt.figure(figsize=(6,5))
sns.heatmap(cm_svm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted No Churn (False)', 'Predicted Churn (True)
            yticklabels=['Actual No Churn (False)', 'Actual Churn (True)'])
plt.title("Confusion Matrix - SVM")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

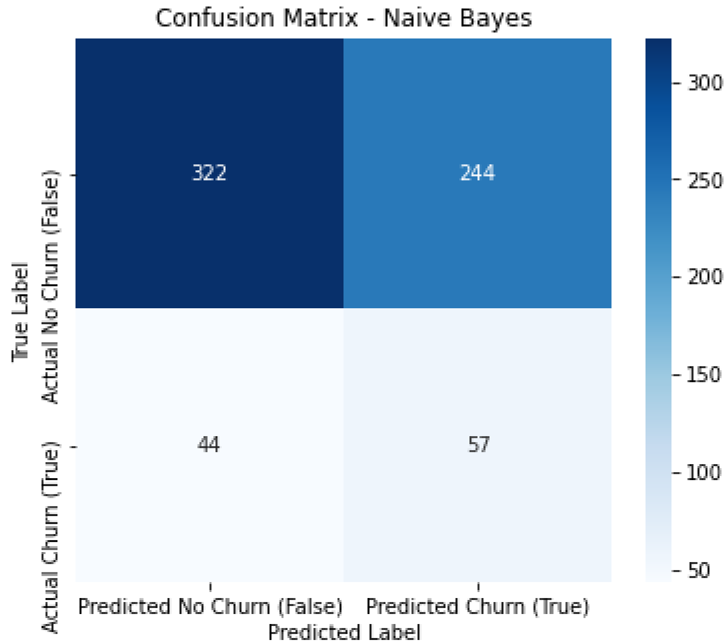
```
C:\Users\jmuriithi\.conda\envs\learn-env\lib\site-packages\sklearn\metrics\_c
lassification.py:1221: UndefinedMetricWarning: Precision is ill-defined and b
eing set to 0.0 due to no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\jmuriithi\.conda\envs\learn-env\lib\site-packages\sklearn\metrics\_c
lassification.py:1221: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Model 5 - SVM Classifier
Accuracy: 0.848575712143928
Recall: 0.0
Precision: 0.0
F1-score: 0.0
              precision    recall  f1-score   support

       False       0.85      1.00      0.92       566
        True       0.00      0.00      0.00       101

    accuracy                           0.85       667
   macro avg       0.42      0.50      0.46       667
weighted avg       0.72      0.85      0.78       667
```



In [490]:
```
from sklearn.naive_bayes import GaussianNB

# Initialize, train Naive Bayes model
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_nb = nb_model.predict(X_test)

# Evaluate the model
accuracy_nb = accuracy_score(y_test, y_pred_nb)
recall_nb = recall_score(y_test, y_pred_nb)
precision_nb = precision_score(y_test, y_pred_nb)
f1_nb = f1_score(y_test, y_pred_nb)

print('Model 6 - Naive Bayes Classifier')
print(f"Accuracy: {accuracy_nb}")
print(f"Recall: {recall_nb}")
print(f"Precision: {precision_nb}")
print(f"F1-score: {f1_nb}")
print(classification_report(y_test, y_pred_nb))

# Plot confusion matrix
cm_nb = confusion_matrix(y_test, y_pred_nb)
plt.figure(figsize=(6,5))
sns.heatmap(cm_nb, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted No Churn (False)', 'Predicted Churn (True)
            yticklabels=['Actual No Churn (False)', 'Actual Churn (True)'])
plt.title("Confusion Matrix - Naive Bayes")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Model 6 - Naive Bayes Classifier
Accuracy: 0.568215892053973
Recall: 0.5643564356435643
Precision: 0.1893687707641196
F1-score: 0.2835820895522388
              precision    recall  f1-score   support

       False       0.88      0.57      0.69       566
        True       0.19      0.56      0.28       101

    accuracy                           0.57       667
   macro avg       0.53      0.57      0.49       667
weighted avg       0.78      0.57      0.63       667
```



Best Performing Model

- Random Forest shows the best balance of accuracy (93.85%), precision (96.88%), and F1-score (75.15%), despite having a slightly lower recall (61.39%) compared to the decision tree.

Second Best Model

- Decision Tree also performs well with high accuracy (92.35%), decent recall (74.26%), and a good F1-score (74.63%). It's a great model for detecting churn.

Weak Models

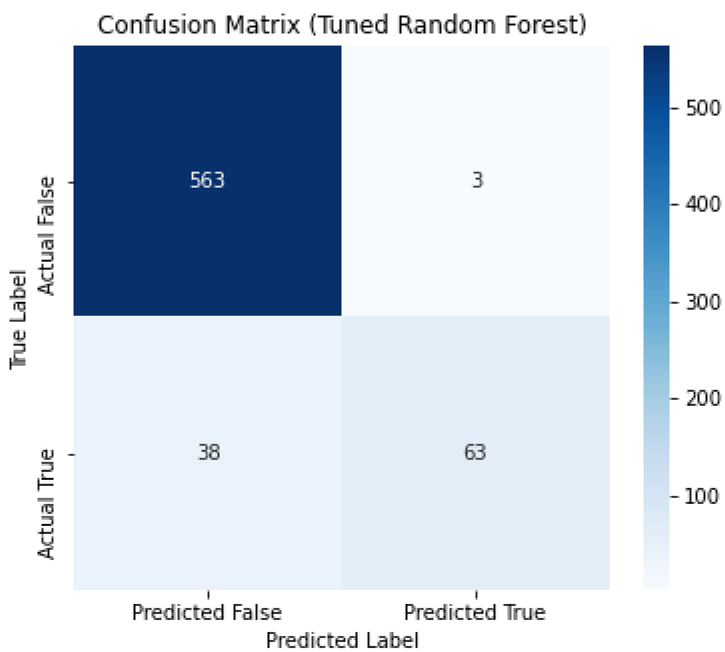
- KNN and SVM fail to predict churn effectively, particularly in terms of recall and precision.

Hyperparameter Tuning

In [491]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_
import matplotlib.pyplot as plt
import seaborn as sns

# Parameter grid for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize the RandomForest model
rf_model = RandomForestClassifier(random_state=42)
# Initialize GridSearchCV
rf_grid = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, verbos

rf_grid.fit(X_train, y_train)
rf_model_tuned = rf_grid.best_estimator_
y_pred = rf_model_tuned.predict(X_test)

# Evaluate the tuned model
rf_accuracy = accuracy_score(y_test, y_pred)
rf_recall = recall_score(y_test, y_pred)
rf_precision = precision_score(y_test, y_pred)
rf_f1 = f1_score(y_test, y_pred)

print(f"Tuned Random Forest - Accuracy: {rf_accuracy}")
print(f"Tuned Random Forest - Recall: {rf_recall}")
print(f"Tuned Random Forest - Precision: {rf_precision}")
print(f"Tuned Random Forest - F1-score: {rf_f1}")
print(classification_report(y_test, y_pred))

# Confusion matrix for the tuned model
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted False', 'Predicted True'],
            yticklabels=['Actual False', 'Actual True'])
plt.title("Confusion Matrix (Tuned Random Forest)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    6.3s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   24.0s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   50.7s
[Parallel(n_jobs=-1)]: Done 405 out of 405 | elapsed:   58.8s finished

Tuned Random Forest - Accuracy: 0.9385307346326837
Tuned Random Forest - Recall: 0.6237623762376238
Tuned Random Forest - Precision: 0.9545454545454546
Tuned Random Forest - F1-score: 0.7544910179640719
              precision    recall  f1-score   support

       False       0.94      0.99      0.96       566
        True       0.95      0.62      0.75       101

    accuracy                           0.94       667
   macro avg       0.95      0.81      0.86       667
weighted avg       0.94      0.94      0.93       667
```



In [492]:

```
# The best parameters after tuning
best_params = rf_grid.best_params_
print("Best Hyperparameters for Random Forest Model:")
print(best_params)
```

```
Best Hyperparameters for Random Forest Model:
{'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimator
s': 100}
```

Model Evaluation

The tuned Random Forest model shows excellent performance with a high accuracy (94%) and very good precision (95%)

Conclusion & Recommendation

The Random Forest Classifier emerged as the best model for churn prediction, achieving the highest accuracy (93.85%) and a good balance between recall (0.62) and precision (0.95) after hyperparameter tuning. The Decision Tree Classifier also performed well with an accuracy of 92%.

Models like KNN, SVM, and Naive Bayes showed poorer performance and are not recommended for this task.

Recommendation:

- Use Random Forest as the primary model due to its balanced performance and high accuracy.
- Consider further tuning.

Based on the data and analysis, one of the key attributes predictive of churn is the number of customer service calls. Customers who have had more interactions with customer service tend to be more likely to churn

Customers who make frequent customer service calls could be facing unresolved issues. By analyzing the types of issues raised, SyriaTel can prioritize improvements in the areas that matter most to customers

In [ ]:

```
```