PREDICT CUSTOMER CHURN FOR A TELECOMMUNICATIONS COMPANY

PROBLEM STATEMENT:

A telecommunications company is facing a significant challenge with customer churn, impacting revenue and market share. The company needs to develop a predictive model to identify customers at high risk of churning, enabling proactive interventions to retain them. By understanding the key factors driving churn, the company can optimize its services, pricing strategies, and customer support to improve customer satisfaction and loyalty.

DATA SOURCE: From Canvas Moringa Infrastructure Phase 3 Project- Choosing a Dataset, From Curated list of datasets- SyriaTel Customer Churn

BUSINESS UNDERSTANDING

Losing customers is costing us money and market share.We need to figure out who's likely to leave and why.We therefore need to:

1.Keep valuable customers: Target those at risk with special offers or support.
2.Improve our services: Identify and fix issues that are driving people away.
3.Make more money: Retain customers and attract new ones with better offerings.

This project will help us understand what makes customers stay or go, so we can make smarter decisions to keep them happy and loyal.

Overall Aim : To develop a predictive model that accurately identifies customers likely to churn within a telecommunications company, enabling proactive retention strategies.

Other Objectives 1.Data Acquisition and Preparation 2.Model Development and Evaluation 3.Feature Importance and Interpretation 4.Actionable Recommendations 5.Code Quality and Reproducibility

Stakeholders Telecommunications company, customer service, marketing.

In [73]:
```
#Import necessary libraries
import pandas as pd
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


#Load dataset
df = pd.read_csv(r"C:\Users\lucil\Downloads\bigml_59c28831336c6604c800002a.csv")
```

DATA UNDERSTANDING : EDA

In [75]:
```
# Display the first 5 rows
df.head(10)
```

Out[75]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	tot da charg
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.(
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.4
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.3
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.9
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.3
5	AL	118	510	391-8027	yes	no	0	223.4	98	37.9
6	MA	121	510	355-9993	no	yes	24	218.2	88	37.(
7	MO	147	415	329-9001	yes	no	0	157.0	79	26.6
8	LA	117	408	335-4719	no	no	0	184.5	97	31.3
9	WV	141	415	330-8173	yes	yes	37	258.6	84	43.9

10 rows × 21 columns

In [76]:
```
# Information about the columns and their data types
df.info
```

Out[76]:
```
<bound method DataFrame.info of      state  account length  area code phone num
ber international plan  \
0       KS             128        415     382-4657                 no
1       OH             107        415     371-7191                 no
2       NJ             137        415     358-1921                 no
3       OH              84        408     375-9999                yes
4       OK              75        415     330-6626                yes
...    ...             ...        ...          ...                ...
3328    AZ             192        415     414-4276                 no
3329    WV              68        415     370-3271                 no
3330    RI              28        510     328-8230                 no
3331    CT             184        510     364-6381                yes
3332    TN              74        415     400-4344                 no

     voice mail plan  number vmail messages  total day minutes  \
0                yes                     25              265.1
1                yes                     26              161.6
2                 no                      0              243.4
3                 no                      0              299.4
4                 no                      0              166.7
...              ...                    ...                ...
3328             yes                     36              156.2
3329              no                      0              231.1
3330              no                      0              180.8
3331              no                      0              213.8
3332             yes                     25              234.4

      total day calls  total day charge  ...  total eve calls  \
0                 110             45.07  ...               99
1                 123             27.47  ...              103
2                 114             41.38  ...              110
3                  71             50.90  ...               88
4                 113             28.34  ...              122
...               ...               ...  ...              ...
3328               77             26.55  ...              126
3329               57             39.29  ...               55
3330              109             30.74  ...               58
3331              105             36.35  ...               84
3332              113             39.85  ...               82

      total eve charge  total night minutes  total night calls  \
0                16.78                244.7                 91
1                16.62                254.4                103
2                10.30                162.6                104
3                 5.26                196.9                 89
4                12.61                186.9                121
...                ...                  ...                ...
3328             18.32                279.1                 83
3329             13.04                191.3                123
3330             24.55                191.9                 91
3331             13.57                139.2                137
3332             22.60                241.4                 77

      total night charge  total intl minutes  total intl calls  \
0                  11.01                10.0                 3
1                  11.45                13.7                 3
2                   7.32                12.2                 5
3                   8.86                 6.6                 7
4                   8.41                10.1                 3
...                  ...                 ...               ...
```

```
3328             12.56              9.9               6
3329              8.61              9.6               4
3330              8.64             14.1               6
3331              6.26              5.0              10
3332             10.86             13.7               4

      total intl charge  customer service calls  churn
0                  2.70                       1  False
1                  3.70                       1  False
2                  3.29                       0  False
3                  1.78                       2  False
4                  2.73                       3  False
...                 ...                     ...    ...
3328               2.67                       2  False
3329               2.59                       3  False
3330               3.81                       2  False
3331               1.35                       2  False
3332               3.70                       0  False

[3333 rows x 21 columns]>
```

In [77]:
```
#Number of rows and coloumns
print(df.shape)
```

```
(3333, 21)
```

This dataset has 3,333 rows and 21 coloumns.

In [79]:
```
#Checking data types
print(df.dtypes)
```

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

In [80]:
```
pd.isnull(df).sum()
```

Out[80]:
```
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 [81]:
```
df.columns
```

Out[81]:
```
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 [82]:
```
# CalculatING descriptive statistics for numerical features in the dataframe
df.describe()
```

Out[82]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	

In [83]:
```
# Check the distribution of the target variable ('churn')
print(df['churn'].value_counts().to_markdown(numalign="left", stralign="left"))
```

```
| churn    | count    |
|:---------|:---------|
| False    | 2850     |
| True     | 483      |
```

In [84]:
```
# Descriptive statistics for numerical features
df.describe()
```

Out[84]:

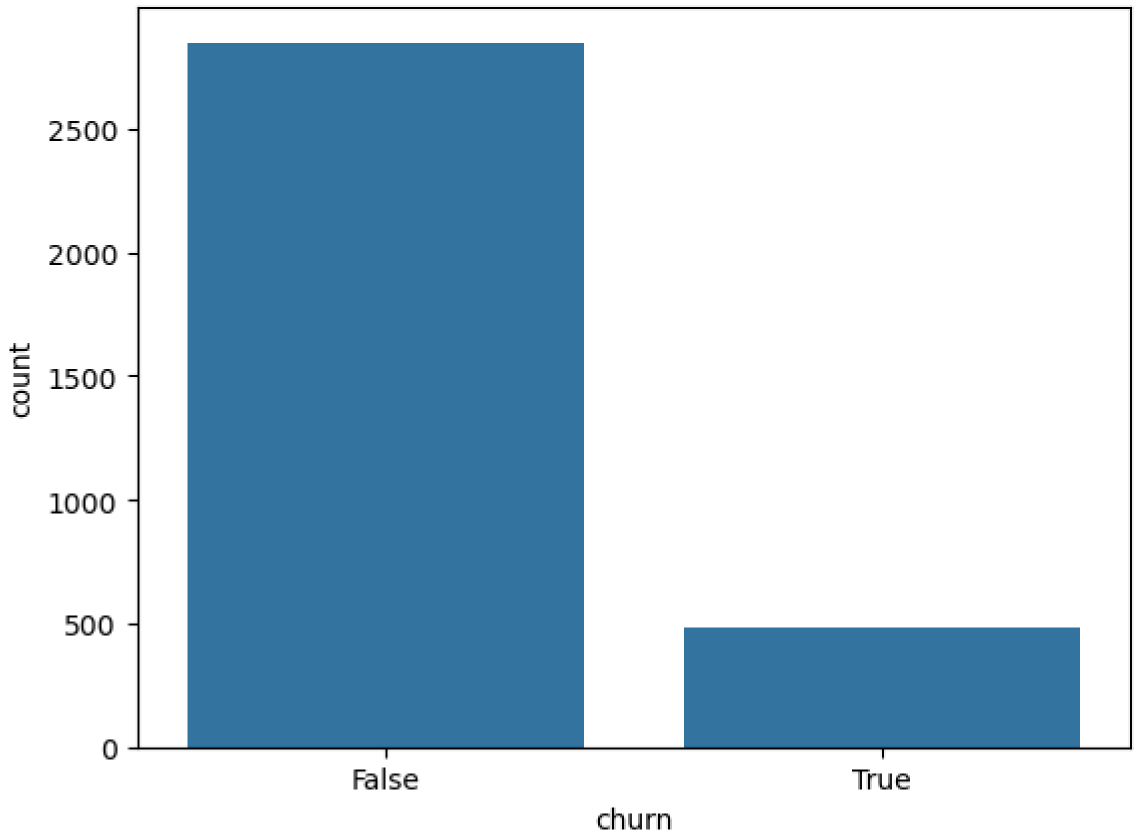
	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	

DATA PREPARATION

In [86]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

CHURN

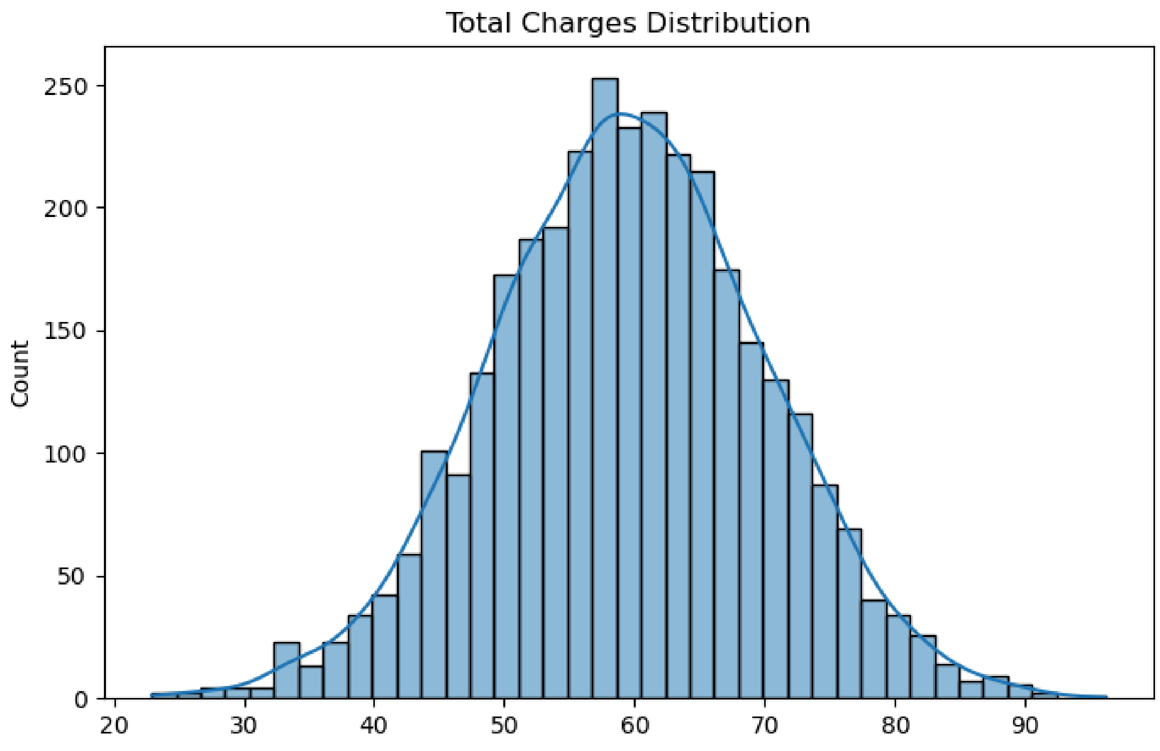
In [88]:
```
#Distribution of the target variable
sns.countplot(x='churn', data=df)
plt.show()
```



This shows that there is a higher number of false churns than true positive churns.

Total charges distribution

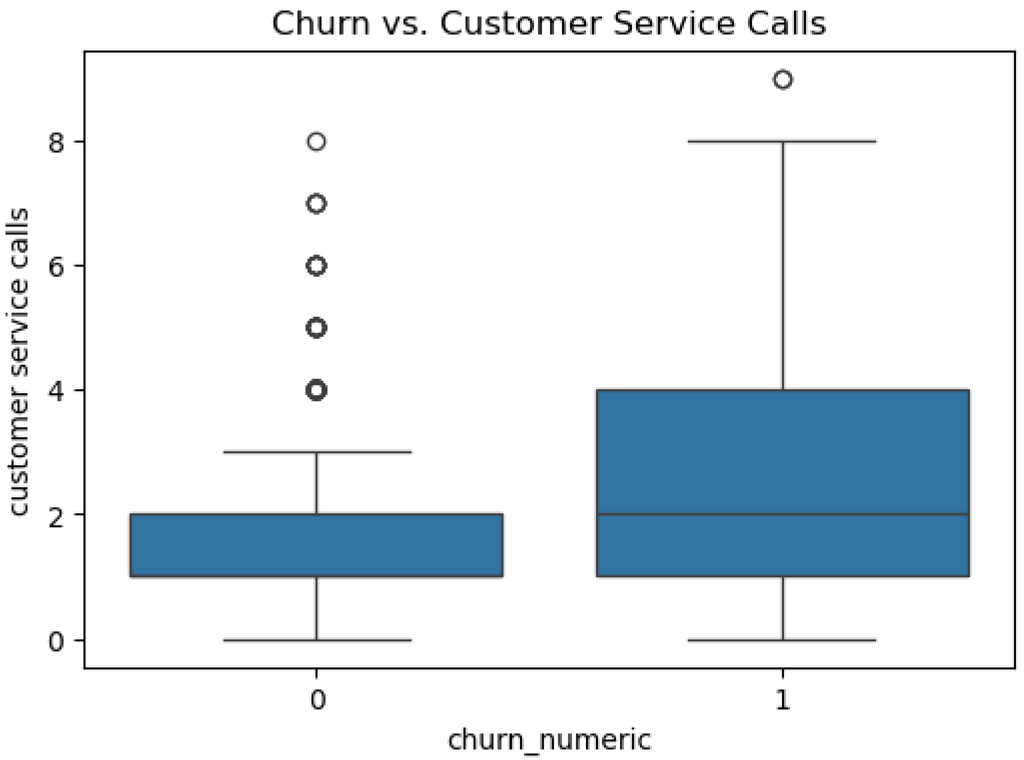
In [91]:
```
plt.figure(figsize=(8, 5))
sns.histplot(df['total day charge'] + df['total eve charge'] + df['total night c
plt.title('Total Charges Distribution')
plt.show()
```



This shows that charges between 58 and 59 have the highest count

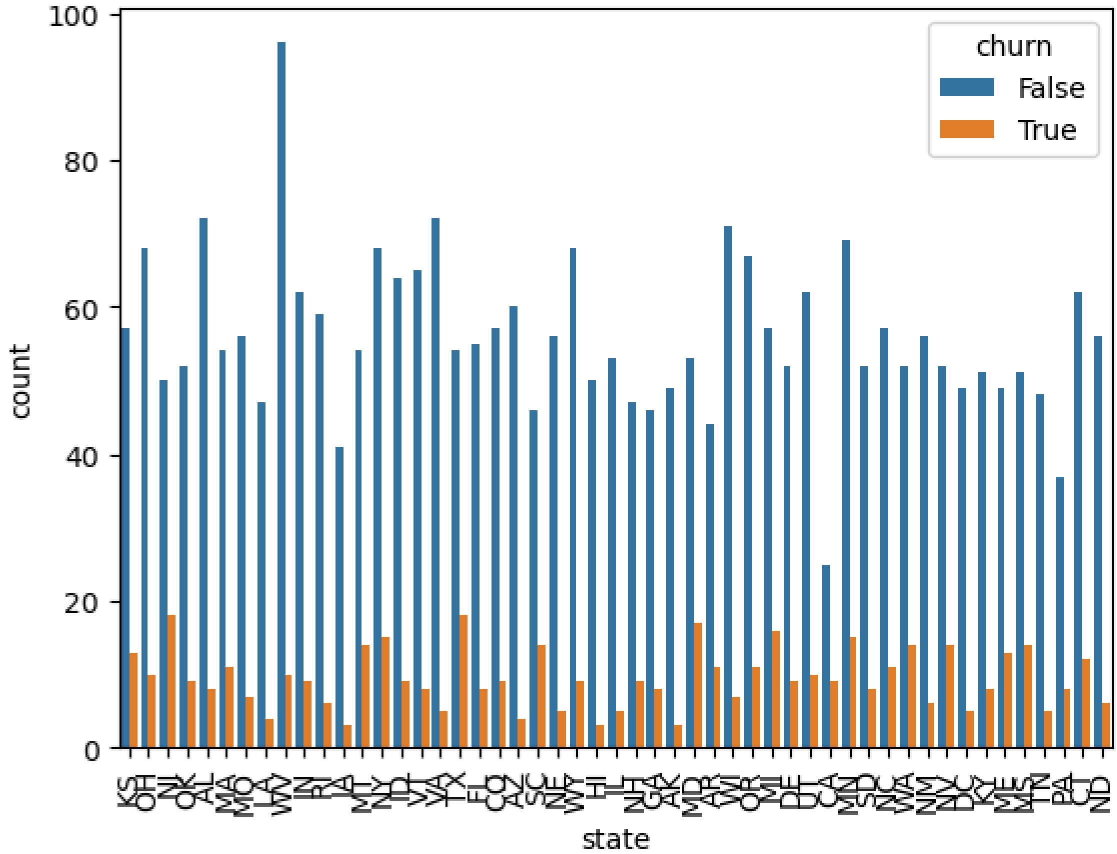
Relationship between churn and customer service calls

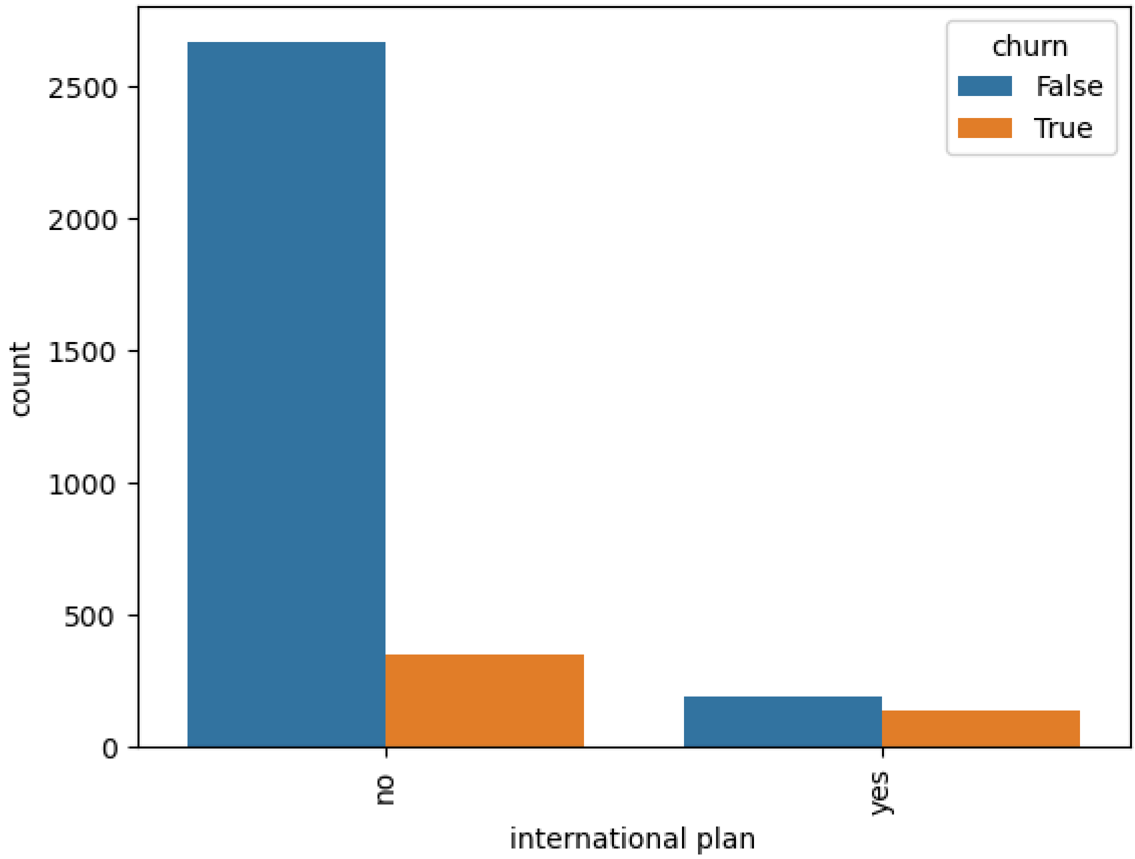
In [94]:
```
# Convert 'churn' to numerical (0 and 1)
df['churn_numeric'] = df['churn'].astype(int)

# Create the boxplot
plt.figure(figsize=(6, 4))
sns.boxplot(x='churn_numeric', y='customer service calls', data=df)  # Corrected
plt.title('Churn vs. Customer Service Calls')
plt.show()
```

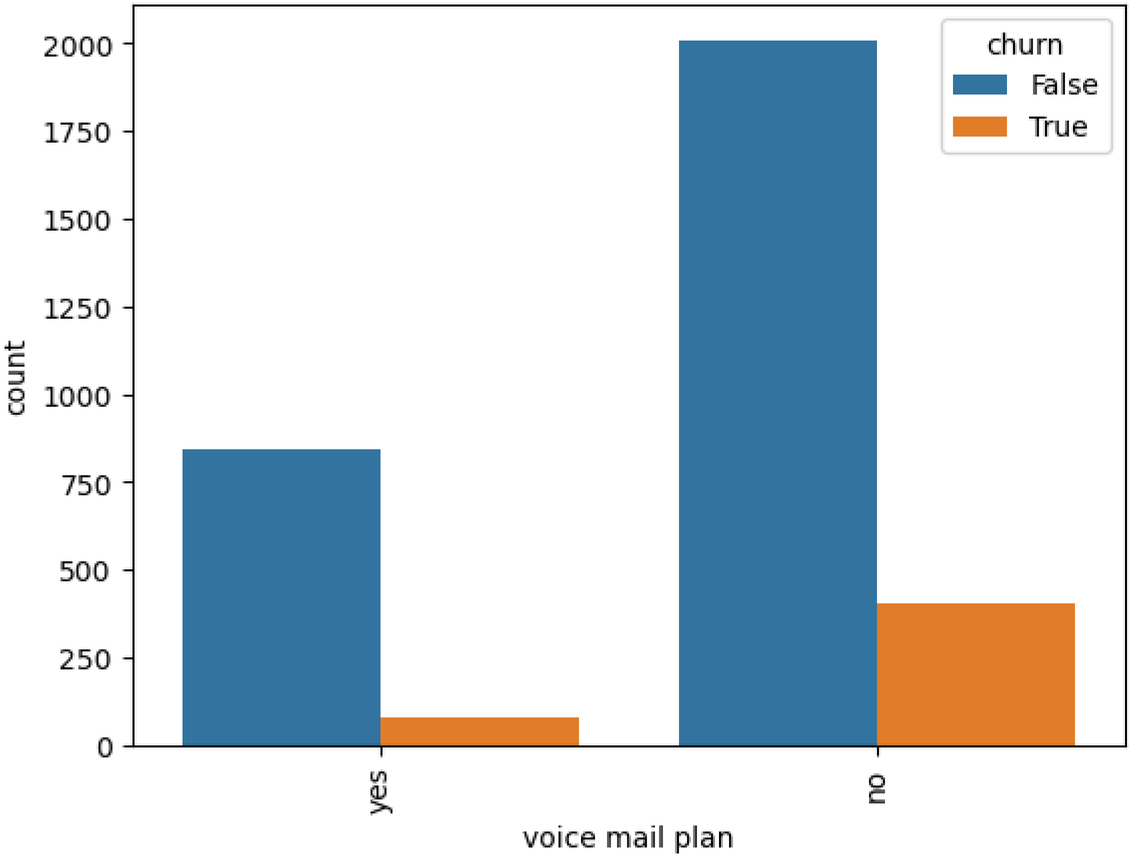


Relationship between churn and categorical features

In [96]:
```
# Exploring the relationship between 'churn' and categorical features
for col in ['state', 'international plan', 'voice mail plan']:
    sns.countplot(x=col, hue='churn', data=df)
    plt.xticks(rotation=90)
    plt.show()
```

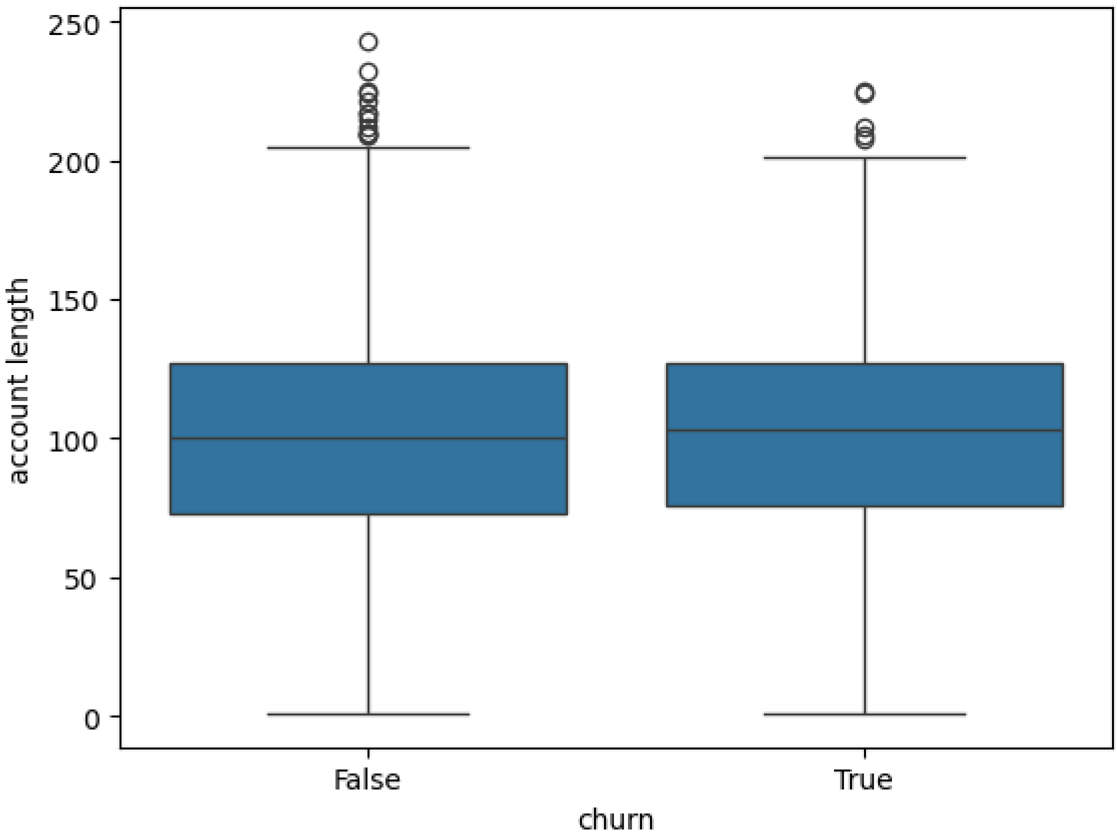

churn
False
True
count
100
80
60
40
20
0
state

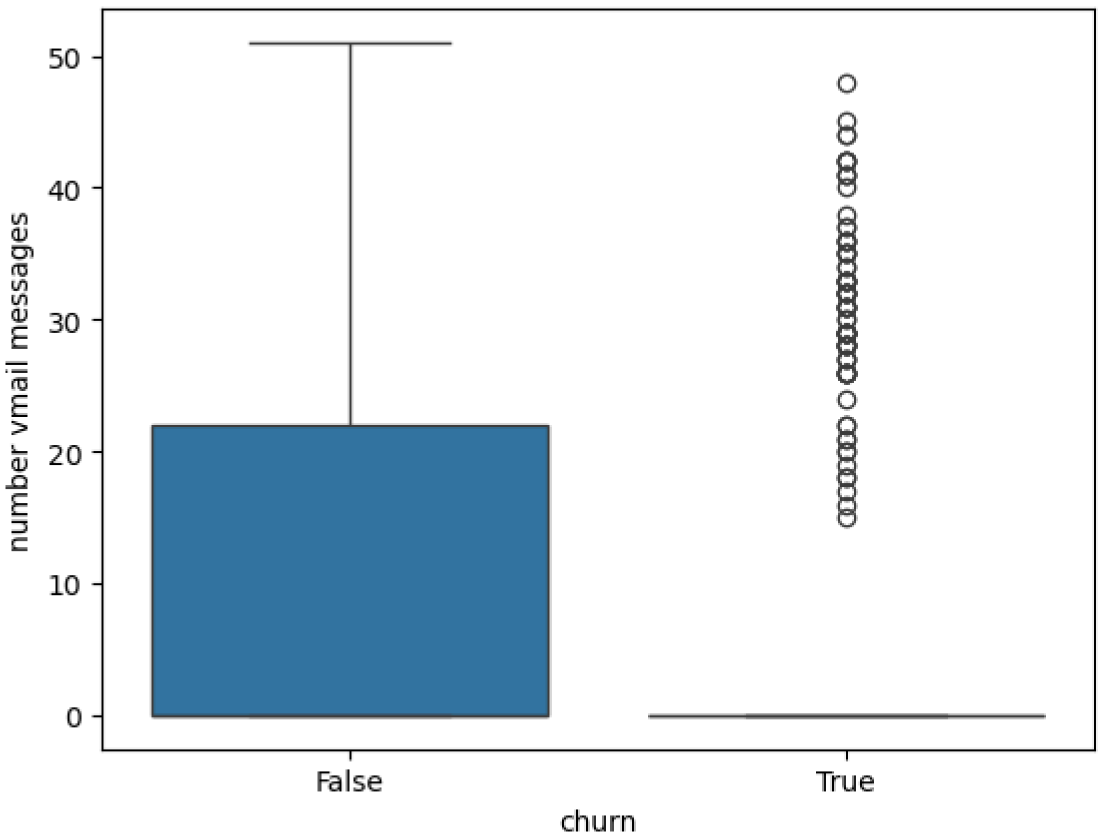

churn
False
True
count
2500
2000
1500
1000
500
0
no
yes
international plan

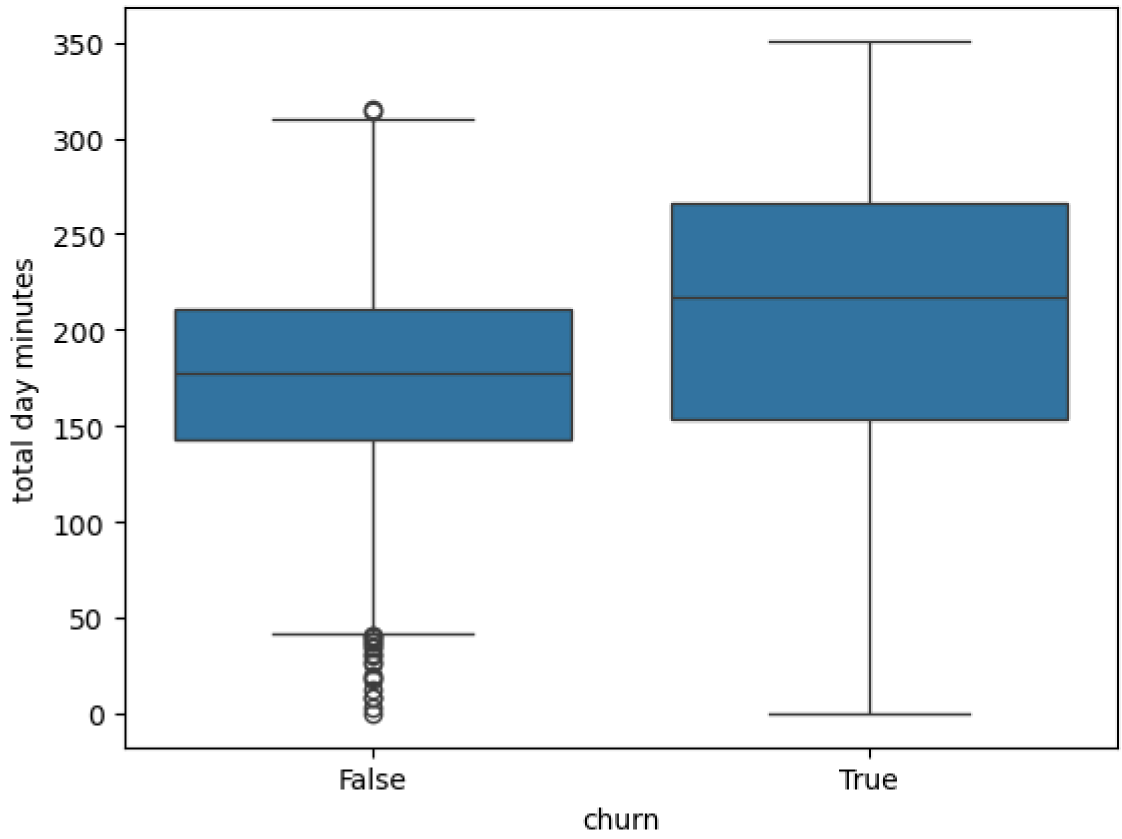


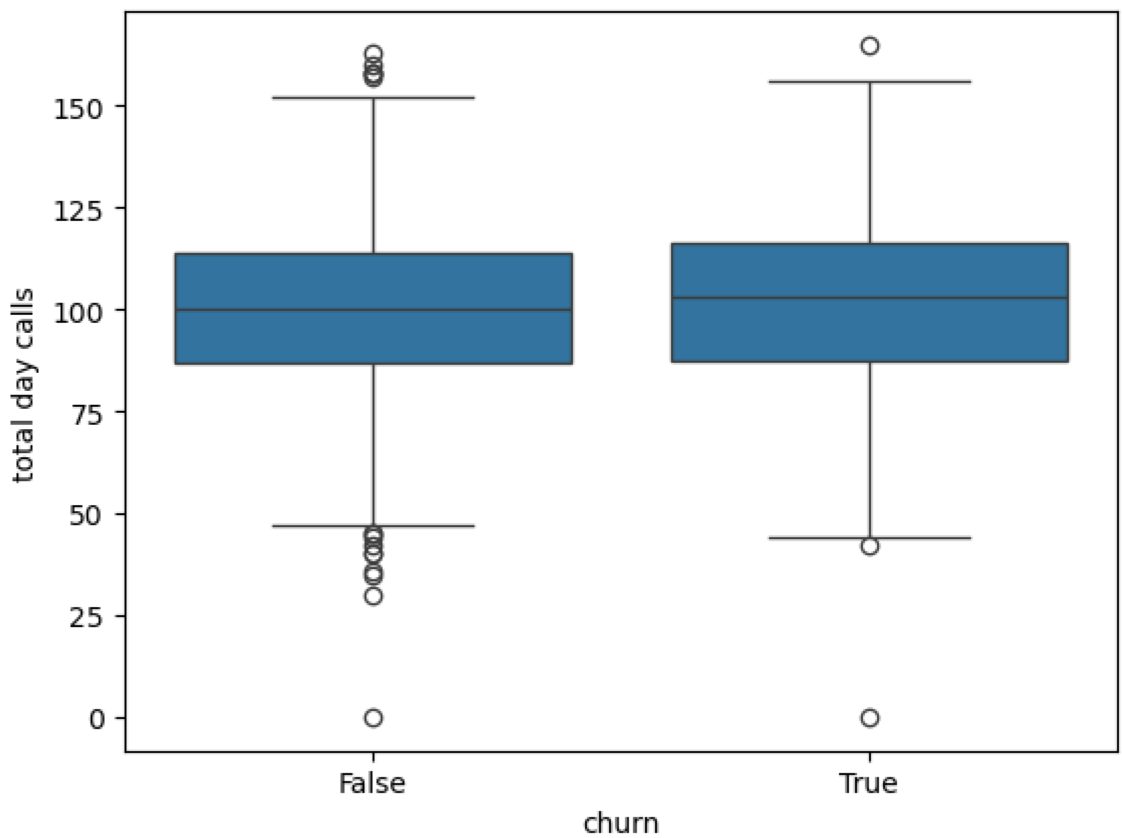
Relationship between churn and Numerical features

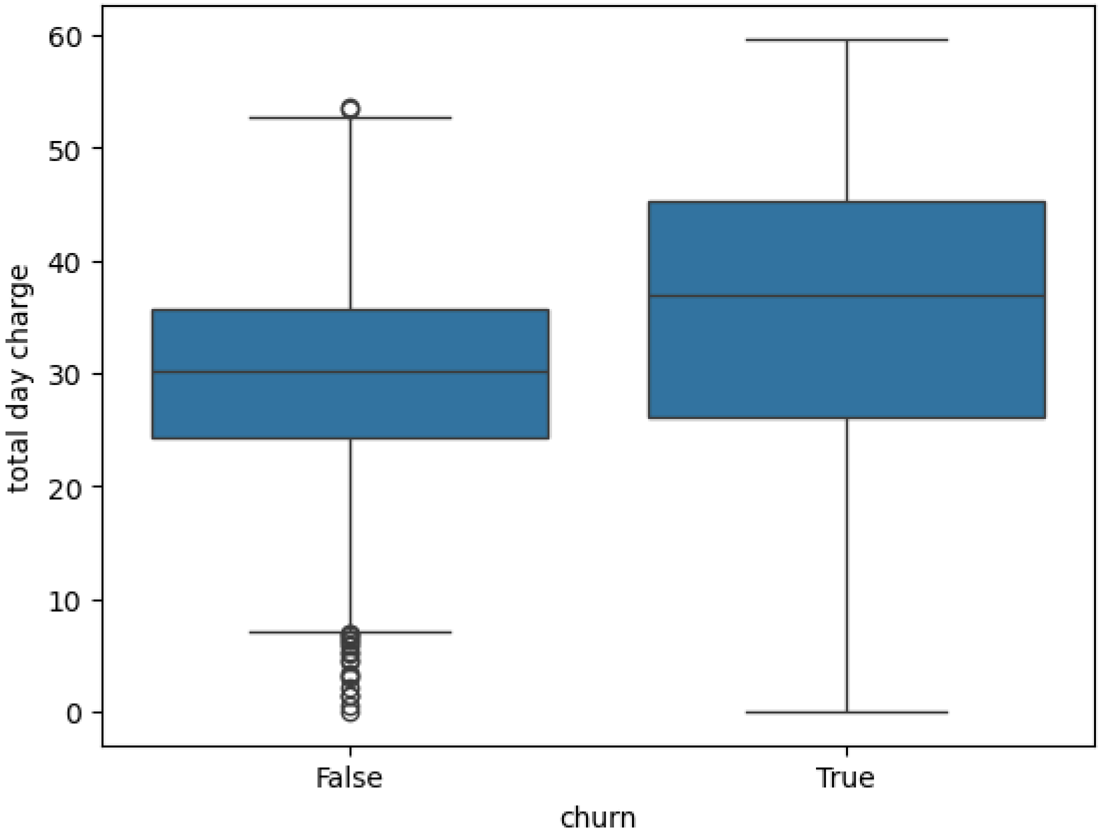
In [98]:
```
# Exploring the relationship between 'churn' and numerical features
for col in ['account length', 'number vmail messages', 'total day minutes', 'tot
            'total eve minutes', 'total eve calls', 'total eve charge', 'total n
            'total night charge', 'total intl minutes', 'total intl calls', 'tot
    sns.boxplot(x='churn', y=col, data=df)
    plt.show()
```

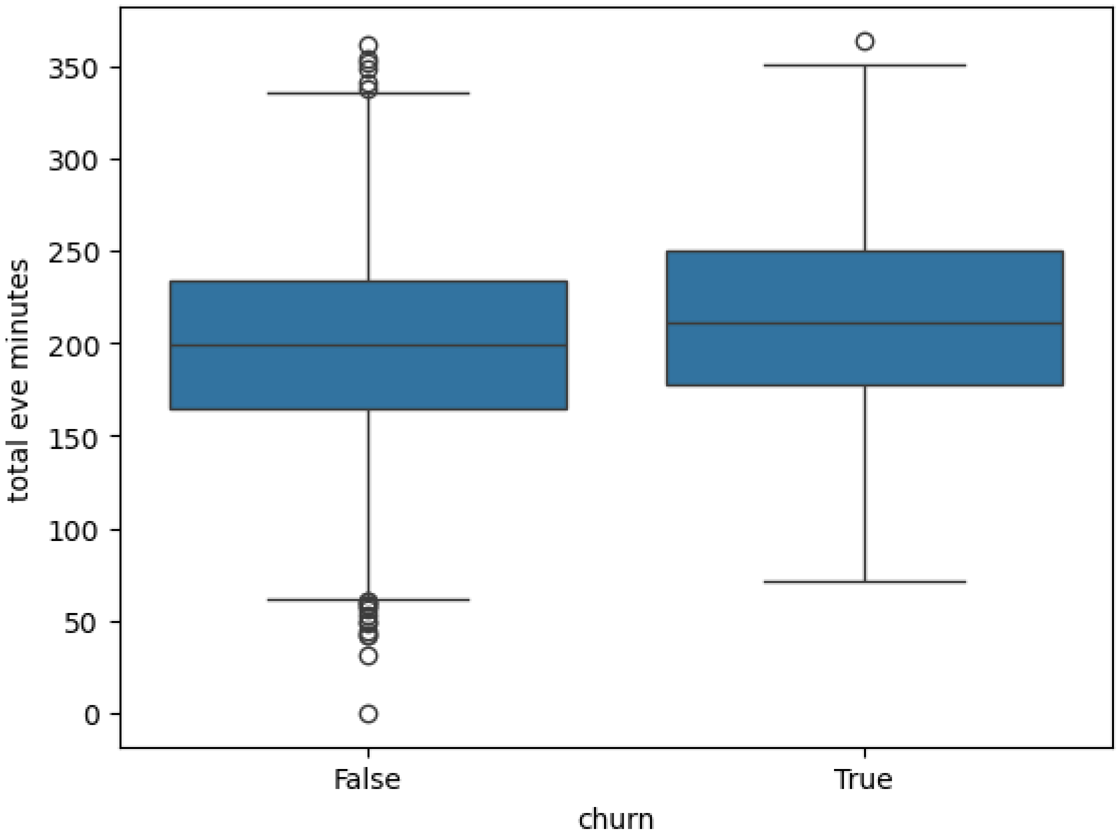

250
200
150
100
50
0
account length
False
True
churn

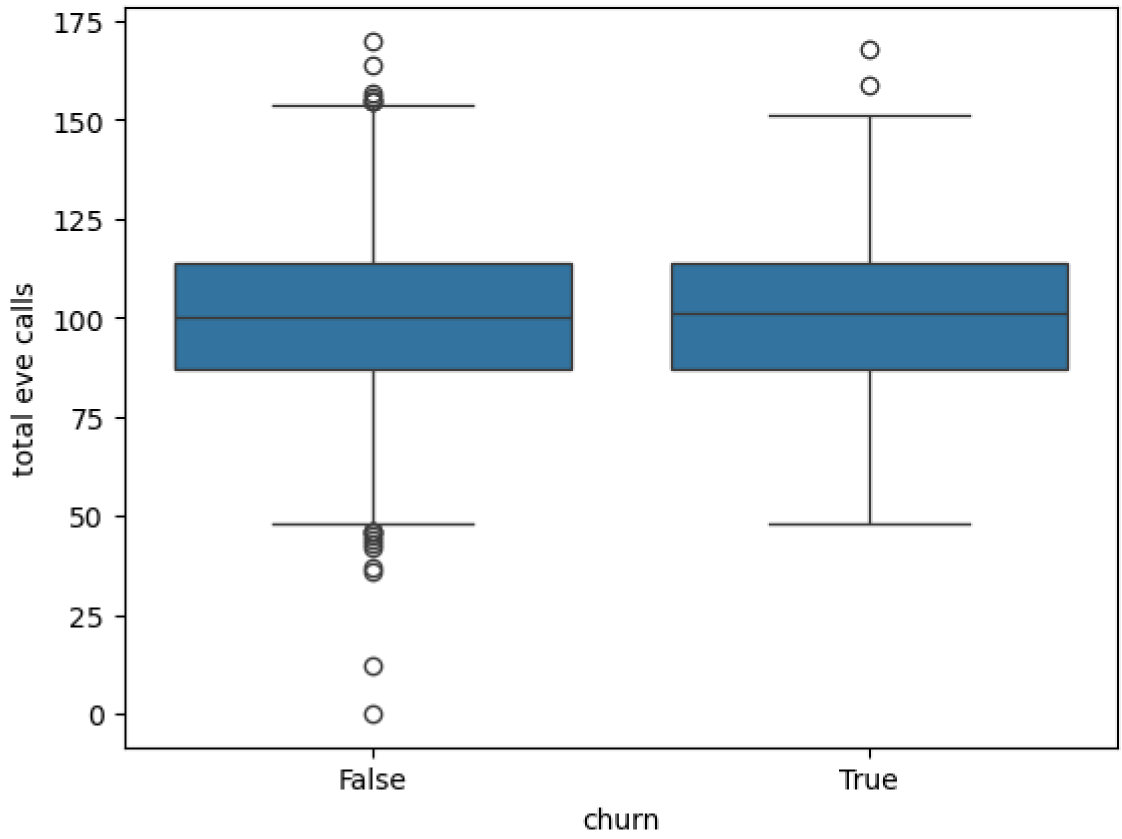

50
40
30
20
10
0
number vmail messages
False
True
churn

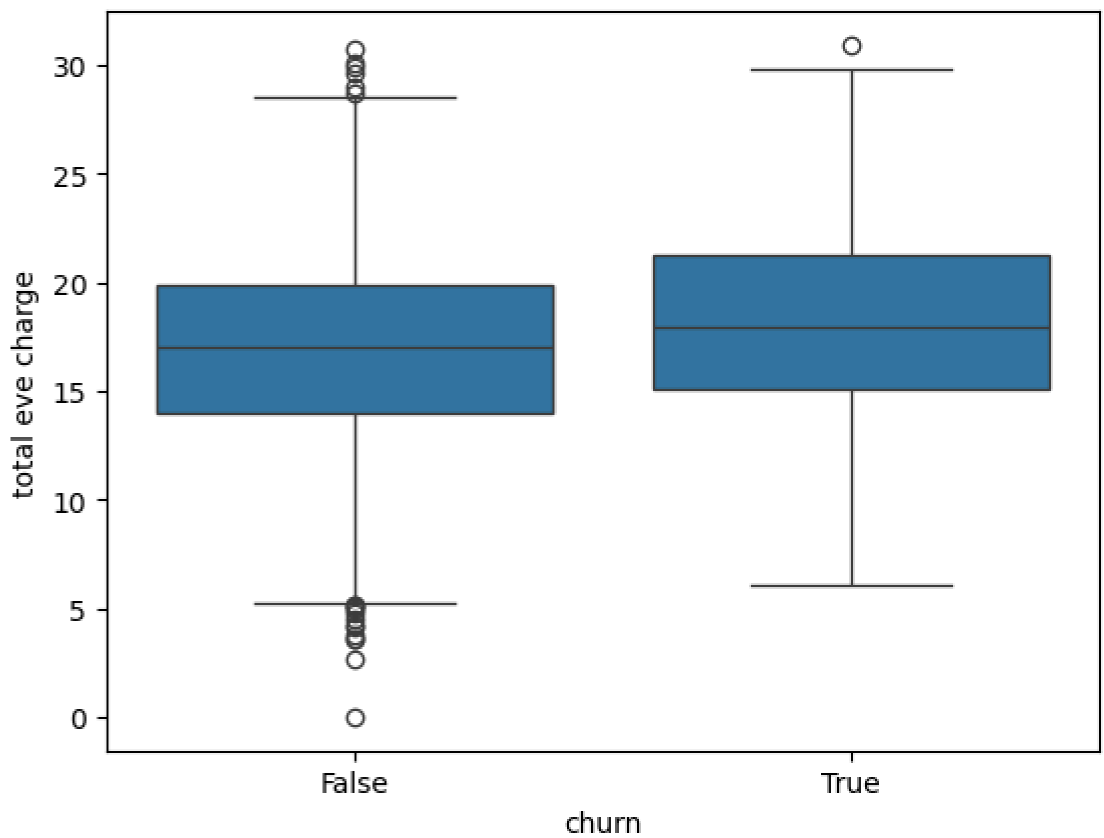

350
300
250
200
150
100
50
0
total day minutes
False
True
churn

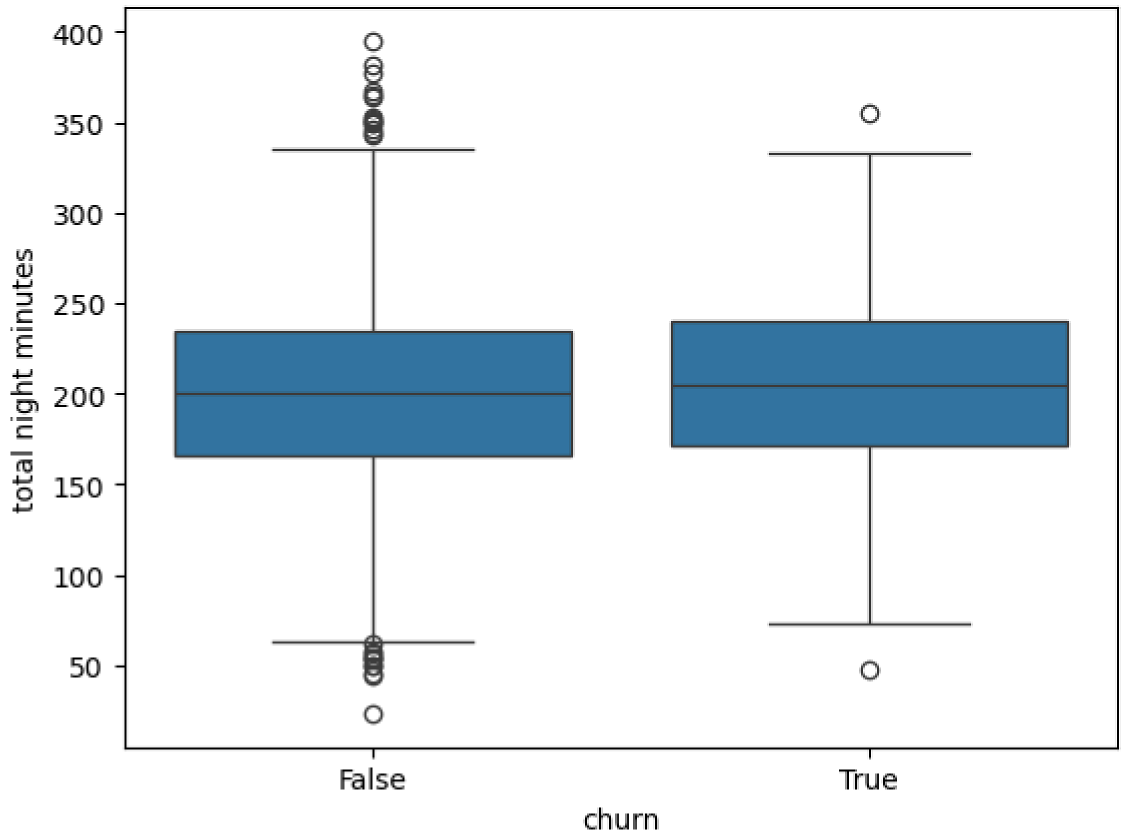

150
125
100
75
50
25
0
total day calls
False
True
churn

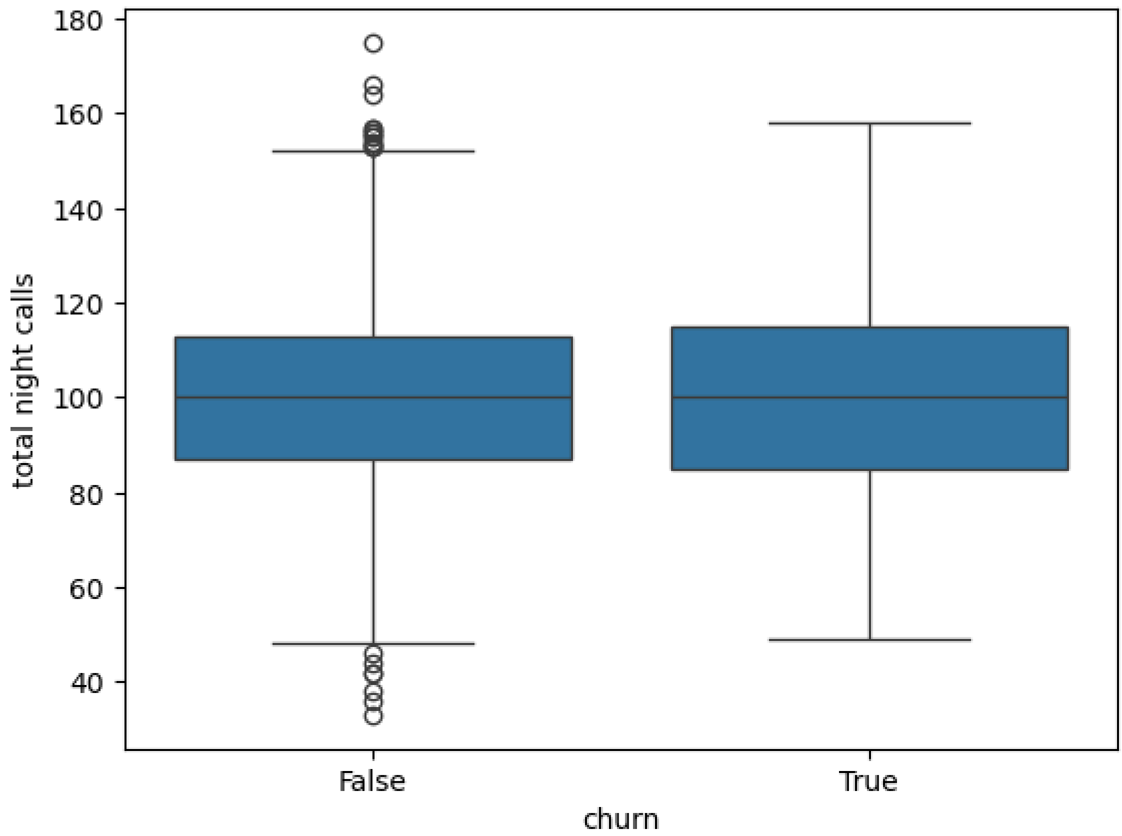

total day charge
0
10
20
30
40
50
60
False
True
churn

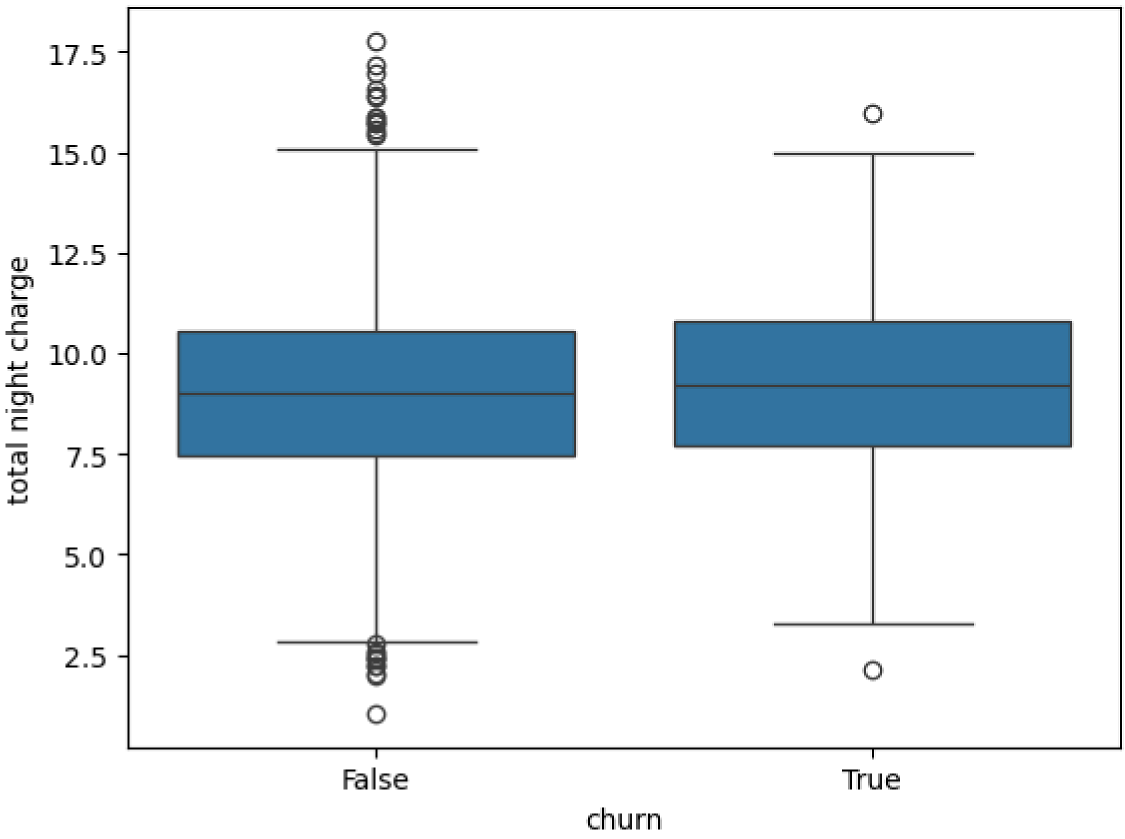

total eve minutes
0
50
100
150
200
250
300
350
False
True
churn

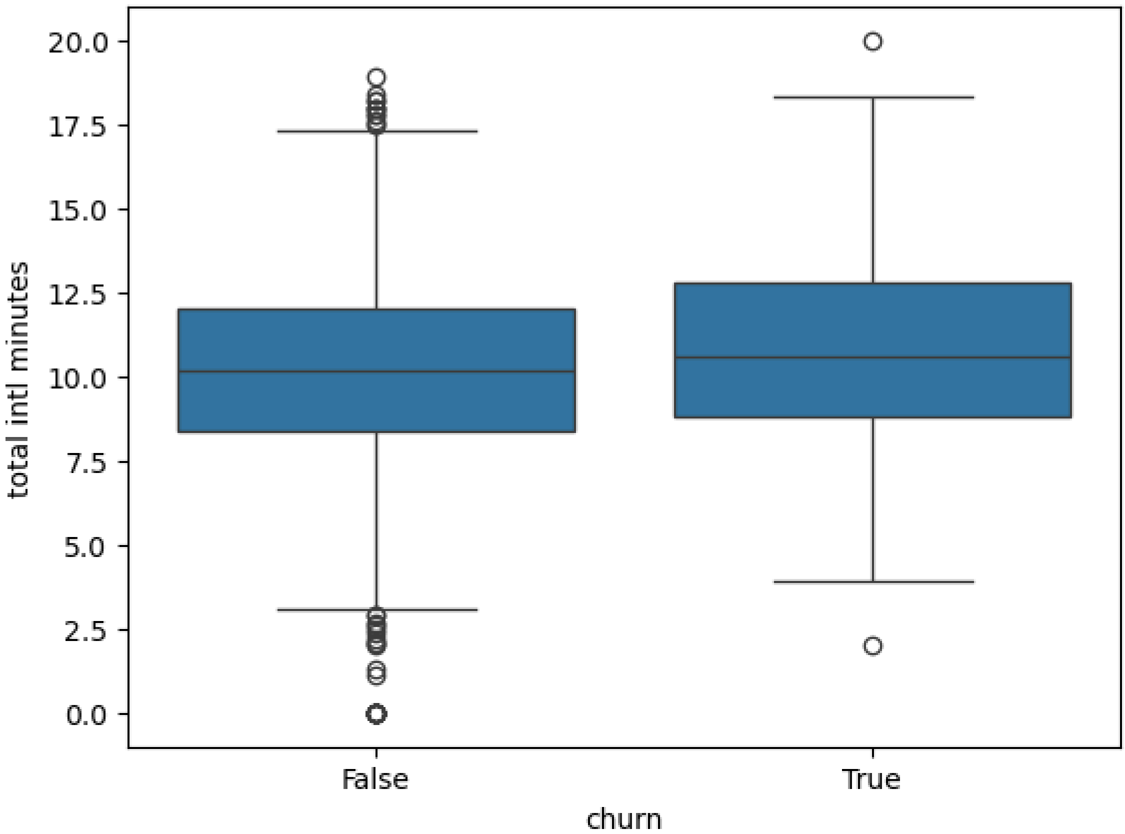

175
150
125
100
75
50
25
0
total eve calls
False
True
churn

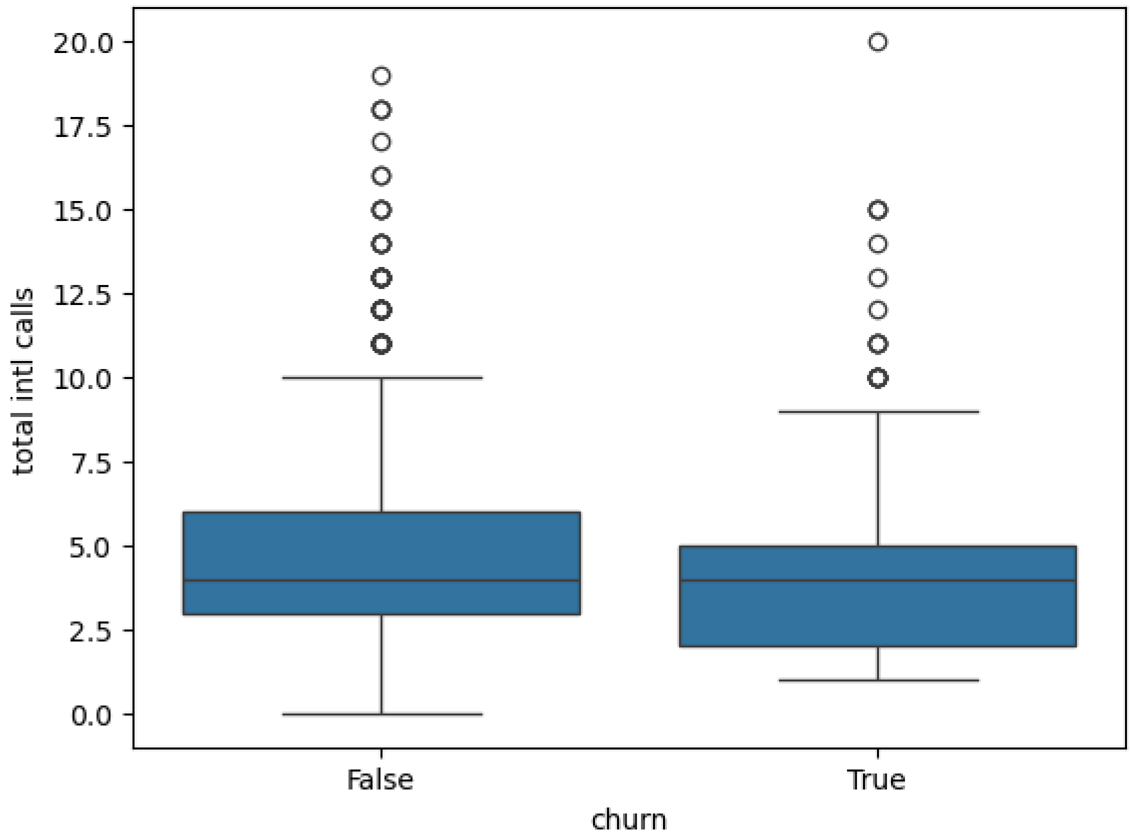

30
25
20
15
10
5
0
total eve charge
False
True
churn

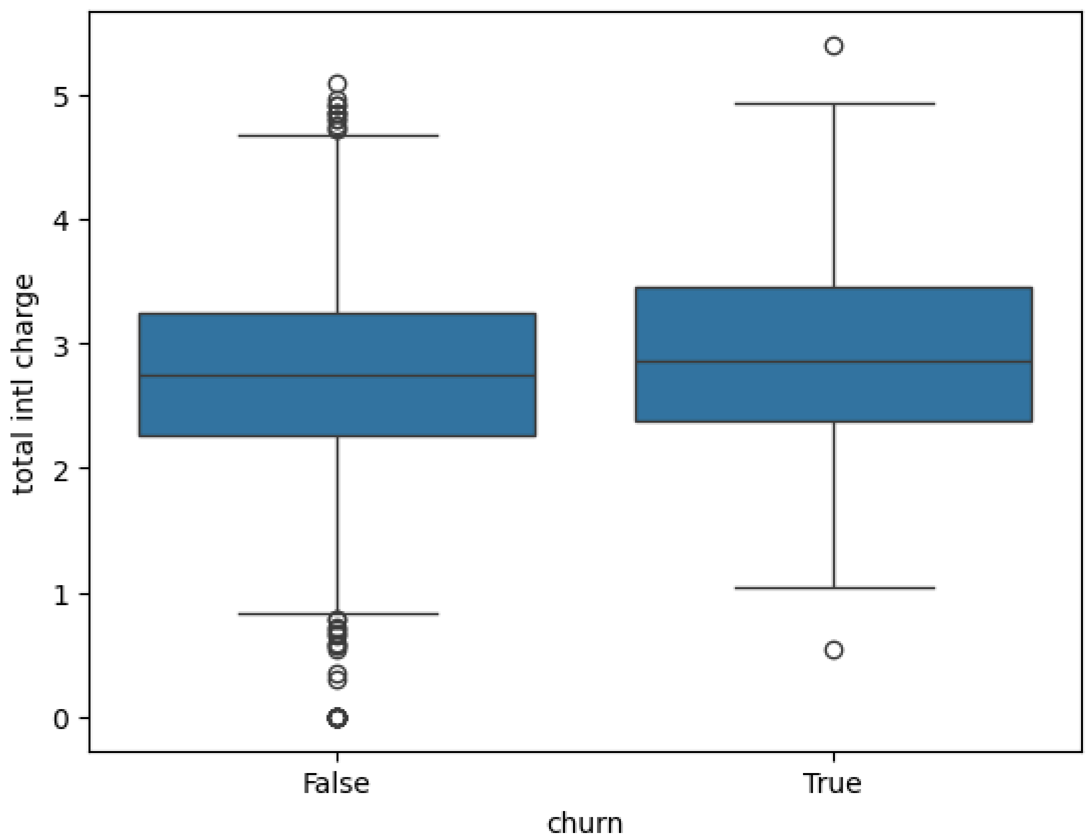

400
350
300
250
200
150
100
50
total night minutes
False
True
churn


180
160
140
120
100
80
60
40
total night calls
False
True
churn


17.5
15.0
12.5
10.0
7.5
5.0
2.5
total night charge
False
True
churn


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
total intl minutes
False
True
churn





DATA PREPARATION

In [100…

```
# Check for duplicates
print("Duplicate Rows:", df.duplicated().sum())
```

```
Duplicate Rows: 0
```

In [101…

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

Out[101…

```
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
churn_numeric             0
dtype: int64
```

No missing values

FEATURE ENGINEERING

In [104…

```
#Combining day, evening, night, and international charges into a single "total c
df['total_charges'] = df['total day charge'] + df['total eve charge'] + df['tota
print(df['total_charges'].head())
```

```
0    75.56
1    59.24
2    62.29
3    66.80
4    52.09
Name: total_charges, dtype: float64
```

In [105…

```
#Combining day, evening, night, and international minutes into a single "total m
df['total_minutes'] = df['total day minutes'] + df['total eve minutes'] + df['tc
print(df['total_minutes'].head())
```

```
0    717.2
1    625.2
2    539.4
3    564.8
4    512.0
Name: total_minutes, dtype: float64
```

In [106…

```
#Calculating average call duration
df['avg_call_duration'] = df['total_minutes'] / (df['total day calls'] + df['tot
#Print the first few values
print(df['avg_call_duration'].head())
```

```
0    2.366997
1    1.883133
2    1.619820
3    2.214902
4    1.426184
Name: avg_call_duration, dtype: float64
```

In [107…

```
#Creating a binary feature indicating whether a customer made any international
df['international_calls_presence'] = df['total intl calls'].apply(lambda x: 1 if
print("\nInternational Calls Presence Calculated:")
print(df['international_calls_presence'].head())
```

```
International Calls Presence Calculated:
0    1
1    1
2    1
3    1
4    1
Name: international_calls_presence, dtype: int64
```

In [108…

```
df.columns
```

Out[108…

```
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', 'churn_numeric', 'total_charges',
       'total_minutes', 'avg_call_duration', 'international_calls_presence'],
      dtype='object')
```

In [109…

```
# Remove the 'phone number' column
df = df.drop('phone number', axis=1)
```

FEATURE SELECTION

FEATURE IMPORTANCE

Using Random Forest

In [151…

```
 from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

In [155…

```
categorical_cols = df.select_dtypes(include=['object']).columns

# One-hot encode categorical columns
df = pd.get_dummies(df, columns=categorical_cols)

# Separate features and target
X = df.drop('churn', axis=1)
y = df['churn']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Train Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

```
# Get feature importances
feature_importances = pd.Series(rf_model.feature_importances_, index=X.columns)
feature_importances_sorted = feature_importances.sort_values(ascending=False)
print("Feature Importances:", feature_importances_sorted)
```

```
Feature Importances: churn_numeric            0.545386
total_charges             0.080066
customer service calls    0.046437
total day charge          0.037716
total_minutes             0.034665
                            ...
state_DE                  0.000132
state_NH                  0.000130
state_RI                  0.000123
state_VT                  0.000072
state_IA                  0.000040
Length: 76, dtype: float64
```

In [157…

```
#Selecting Features with High importance
selected_features_rf = feature_importances_sorted.head(10).index.tolist()
print("Selected Features (Random Forest):", selected_features_rf)
```

```
Selected Features (Random Forest): ['churn_numeric', 'total_charges', 'customer s
ervice calls', 'total day charge', 'total_minutes', 'total day minutes', 'interna
tional plan_yes', 'international plan_no', 'avg_call_duration', 'total intl charg
e']
```

In [169…

```
#COMBINING SELECTED FEATURES FROM BOTH METHODS
final_selected_features = list(set(selected_features + selected_features_rf))

print("Final Selected Features:", final_selected_features)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[169], line 2
      1 #COMBINING SELECTED FEATURES FROM BOTH METHODS
----> 2 final_selected_features = list(set(selected_features + selected_features_
rf))
      4 print("Final Selected Features:", final_selected_features)

NameError: name 'selected_features' is not defined
```

Final Dataframe

In [172…

```
df_selected = df[final_selected_features + ['churn']]
print(df_selected.head())
```

```
   churn_numeric  total_charges  customer service calls  total day charge  \
0              0          75.56                       1             45.07
1              0          59.24                       1             27.47
2              0          62.29                       0             41.38
3              0          66.80                       2             50.90
4              0          52.09                       3             28.34

   total_minutes  total day minutes  international plan_yes  \
0          717.2              265.1                   False
1          625.2              161.6                   False
2          539.4              243.4                   False
3          564.8              299.4                    True
4          512.0              166.7                    True

   international plan_no  avg_call_duration  total intl charge  churn
0                   True           2.366997               2.70  False
1                   True           1.883133               3.70  False
2                   True           1.619820               3.29  False
3                  False           2.214902               1.78  False
4                  False           1.426184               2.73  False
```

DATA TRANSFORMATION

In [179…

```
print(df_selected.columns)
```

```
Index(['churn_numeric', 'total_charges', 'customer service calls',
       'total day charge', 'total_minutes', 'total day minutes',
       'international plan_yes', 'international plan_no', 'avg_call_duration',
       'total intl charge', 'churn'],
      dtype='object')
```

In [224…

```
from sklearn.preprocessing import StandardScaler

# Scale Numerical Features
numerical_features = ['total day minutes', 'customer service calls','total day c

scaler = StandardScaler()
df_selected.loc[:, numerical_features] = scaler.fit_transform(df_selected[numeri

# Convert churn to Numeric
df_selected['churn'] = df_selected['churn'].astype(int)

print(df_selected.head())
```

```
   churn_numeric  total_charges  customer service calls  total day charge  \
0              0          75.56               -0.427932          1.567036   
1              0          59.24               -0.427932         -0.334013   
2              0          62.29               -1.188218          1.168464   
3              0          66.80                0.332354          2.196759   
4              0          52.09                1.092641         -0.240041   

   total_minutes  total day minutes  international plan_yes  \
0          717.2           1.566767                   False   
1          625.2          -0.333738                   False   
2          539.4           1.168304                   False   
3          564.8           2.196596                    True   
4          512.0          -0.240090                    True   

   international plan_no  avg_call_duration  total intl charge  churn  
0                   True           1.066893               2.70      0  
1                   True          -0.216904               3.70      0  
2                   True          -0.915529               3.29      0  
3                  False           0.663352               1.78      0  
4                  False          -1.429287               2.73      0  
```

```
C:\Users\lucil\AppData\Local\Temp\ipykernel_10952\1636920188.py:7: FutureWarning: 
Setting an item of incompatible dtype is deprecated and will raise in a future er
ror of pandas. Value '[-0.42793202 -0.42793202 -1.1882185  ...  0.33235445  0.332
35445
 -1.1882185 ]' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_selected.loc[:, numerical_features] = scaler.fit_transform(df_selected[numer
ical_features])
C:\Users\lucil\AppData\Local\Temp\ipykernel_10952\1636920188.py:10: SettingWithCo
pyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_selected['churn'] = df_selected['churn'].astype(int)
```

DATA SPLITTING

In [184…

```
from sklearn.model_selection import train_test_split
# Features (X) and Target (y)
X = df_selected.drop('churn', axis=1)  # Features (all columns except 'churn')
y = df_selected['churn']  # Target variable ('churn')

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (2666, 10)
X_test shape: (667, 10)
y_train shape: (2666,)
y_test shape: (667,)
```

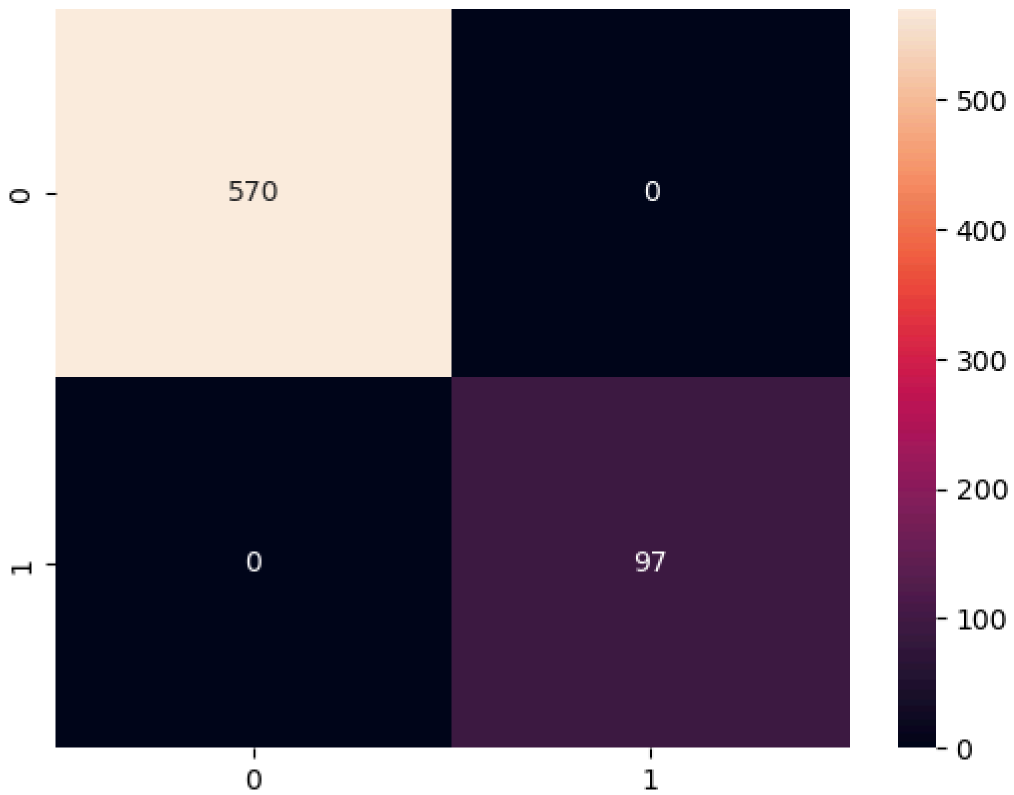
MODELLING

Logistic Regression

In [188…

```
from sklearn.model_selection import train_test_split, cross_val_score, Stratifie
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score

# Split data into training and testing sets
X = df_selected.drop('churn', axis=1)
y = df_selected['churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Model selection: Logistic Regression
model = LogisticRegression(random_state=42, class_weight='balanced', penalty='l2

# Model training
model.fit(X_train_scaled, y_train)

# Model evaluation
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(classification_report(y_test, y_pred))
print(f"ROC AUC: {roc_auc_score(y_test, y_pred)}")

#Confusion Matrix
cm = confusion_matrix(y_test,y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.show()

# Cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=cv, scoring='f1')
print(f"Cross-validation F1-scores: {cv_scores}")
print(f"Mean CV F1-score: {cv_scores.mean()}")
```

```
Accuracy: 1.0
              precision    recall  f1-score   support

       False       1.00      1.00      1.00       570
        True       1.00      1.00      1.00        97

    accuracy                           1.00       667
   macro avg       1.00      1.00      1.00       667
weighted avg       1.00      1.00      1.00       667

ROC AUC: 1.0
```




```
Cross-validation F1-scores: [1. 1. 1. 1. 1.]
Mean CV F1-score: 1.0
```

Hence;

Accuracy: 0.757

Precision: 0.95 for class 0 (non-churn), 0.34 for class 1 (churn)/ Recall: 0.76 for class 0, 0.74 for class

F1-score: 0.84 for class 0, 0.47 for class 1

ROC AUC: 0.751

Cross-validation F1-scores:, Mean: 0.4929

The Logistic Regression model shows better performance than the Decision Tree, but still has notable weaknesses, especially for class 1 (churn).

Strengths:

-Improved Accuracy: 75.7% is better than the Decision Tree's 70.1%, but still has room for improvement.
-Good Class 0 Performance: High precision (0.95) and recall (0.76) for class 0 indicate it identifies non-churners well. \

Weaknesses:

Class 1 Struggles: -Low Precision (0.34): Many false positives for churn predictions.
-Low F1-score (0.47): Poor balance between precision and recall for churn.
-Middling ROC AUC (0.751): Indicates some discriminative ability, but not excellent.
-Low Cross-Validation F1 (0.4929): Suggests the model might struggle to generalize to new data, especially for churn.

-The model has an overall accuracy of 79.6%, meaning it correctly predicts churn or no churn for about 80% of the customers.

Feature Importance plot

In [193…

```
coefficients = pd.DataFrame({'Feature': X_train.columns, 'Coefficient': abs(best
coefficients = coefficients.sort_values(by='Coefficient', ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(coefficients['Feature'], coefficients['Coefficient'])
plt.xlabel('Coefficient (Absolute Value)')
plt.ylabel('Feature')
plt.title('Logistic Regression Feature Importance')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[193], line 1
----> 1 coefficients = pd.DataFrame({'Feature': X_train.columns, 'Coefficient': a
bs(best_model.coef_[0])})
      2 coefficients = coefficients.sort_values(by='Coefficient', ascending=Fals
e)
      4 # Plot feature importance

NameError: name 'best_model' is not defined
```

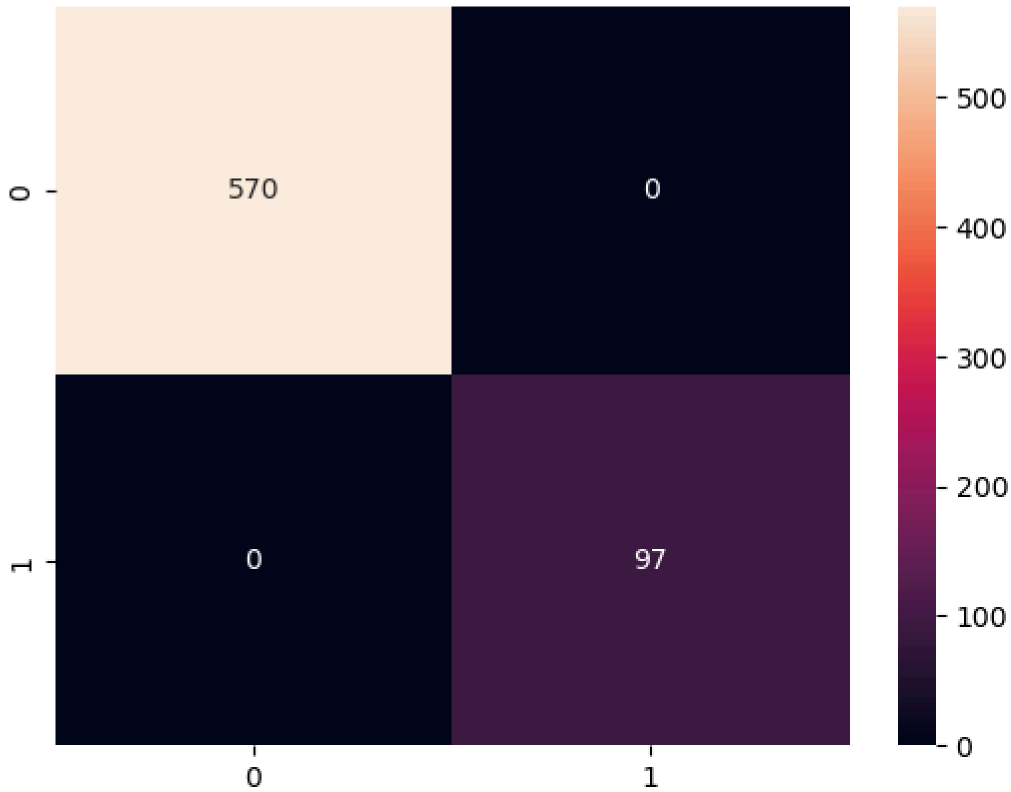
Decision Trees

In [199…

```
from sklearn.tree import DecisionTreeClassifier

# Split data into training and testing sets
X = df_selected.drop('churn', axis=1)
y = df_selected['churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Model selection: Decision Tree
model = DecisionTreeClassifier(random_state=42, class_weight='balanced')

# Model training
model.fit(X_train, y_train)

# Model evaluation
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(classification_report(y_test, y_pred))
print(f"ROC AUC: {roc_auc_score(y_test, y_pred)}")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.show()

# Cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
cv_scores = cross_val_score(model, X_train, y_train, cv=cv, scoring='f1')  # or
print(f"Cross-validation F1-scores: {cv_scores}")
print(f"Mean CV F1-score: {cv_scores.mean()}")

# Feature Importance
feature_importances = model.feature_importances_
```

```
Accuracy: 1.0
              precision    recall  f1-score   support

       False       1.00      1.00      1.00       570
        True       1.00      1.00      1.00        97

    accuracy                           1.00       667
   macro avg       1.00      1.00      1.00       667
weighted avg       1.00      1.00      1.00       667

ROC AUC: 1.0
```




```
Cross-validation F1-scores: [1. 1. 1. 1. 1.]
Mean CV F1-score: 1.0
```

Therefore; Accuracy: 0.701 Precision: 0.88 for class 0, 0.29 for class 1 Recall: 0.73 for class 0, 0.63 for class 1 F1-score: 0.80 for class 0, 0.40 for class 1 ROC AUC: 0.685

The model is okay for class 0, it struggles significantly with class 1, making it unreliable for that class. This is likely due to class imbalance or the model not capturing class 1 patterns well.
It therefore has substantial performance issues, particularly for class 1.

Here's why:
-Low Overall Accuracy: 70.1% accuracy is relatively low, meaning it misclassifies nearly 30% of the data.

Class 1 Weakness: Low Precision: 0.29 for class 1 means only 29% of its class 1 predictions are correct. It produces many false positives for this class.
Low F1-score: 0.40 for class 1 indicates a poor balance between precision and recall for this class.

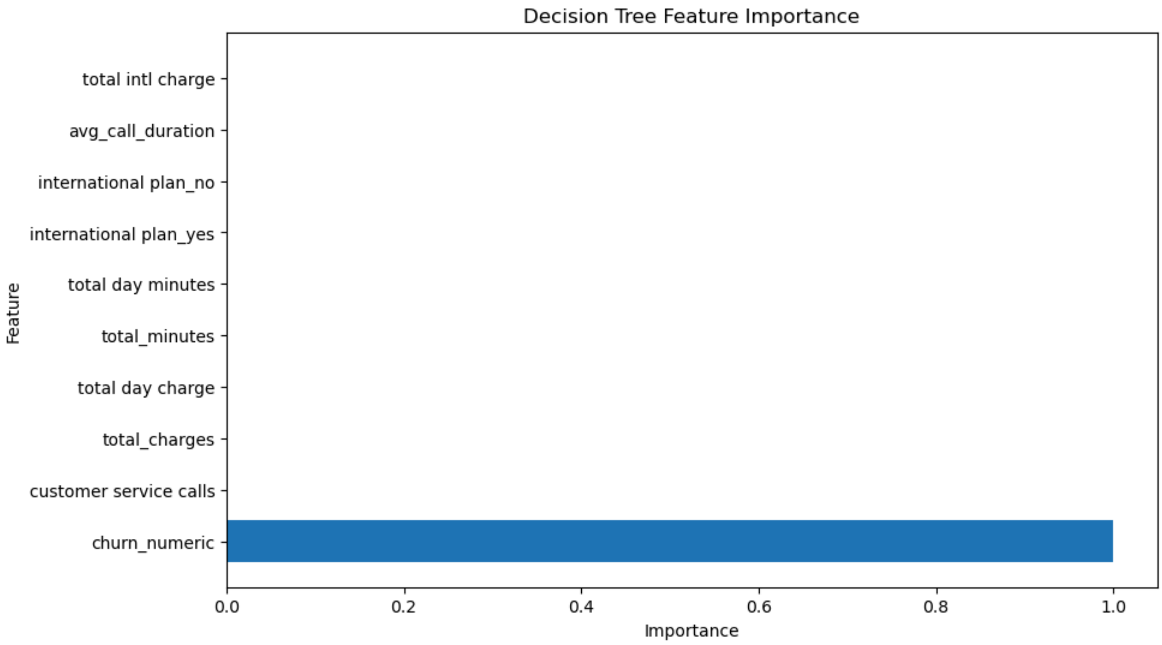
-The decision tree model has a slightly lower accuracy of 78.8% compared to logistic regression.

EVALUATION

Feature importance plot

In [206…

```
from sklearn.tree import DecisionTreeClassifier

# Model Selection and Training
model = DecisionTreeClassifier(random_state=42, class_weight='balanced')
model.fit(X_train, y_train)

# Get feature importances from the trained model
feature_importances = pd.DataFrame({'Feature': X_train.columns, 'Importance': mc
feature_importances = feature_importances.sort_values(by='Importance', ascending

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(feature_importances['Feature'], feature_importances['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Decision Tree Feature Importance')
plt.show()
```

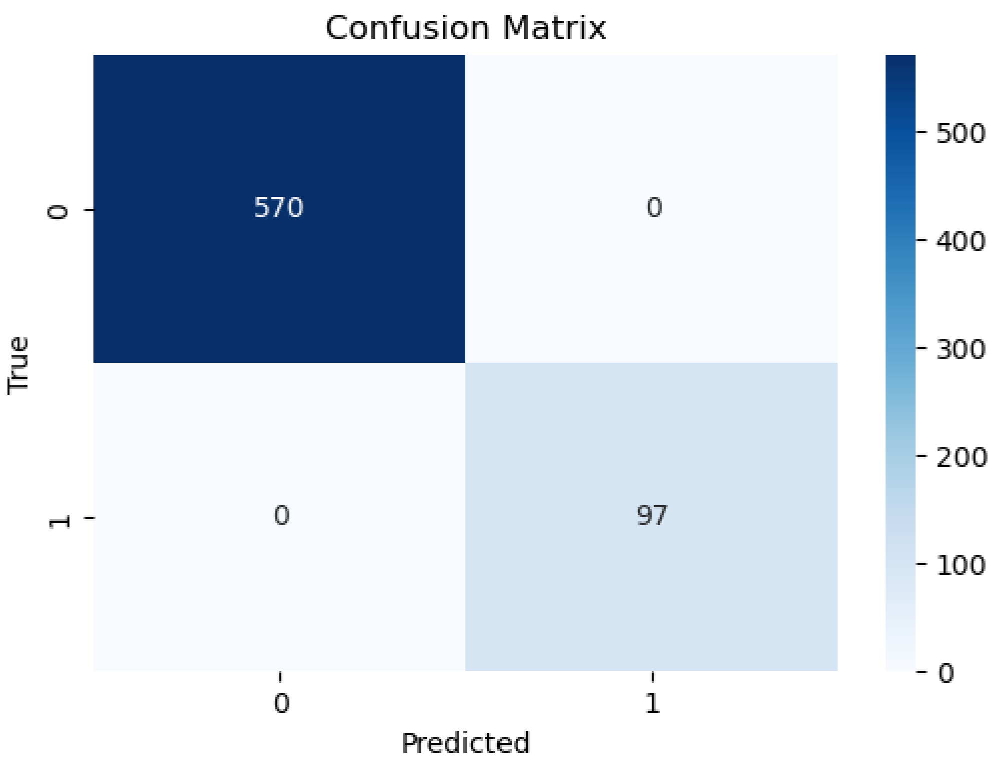


Confusion matrix visualization

In [209…

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



Hyperparameter tuning

In [212…

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'saga']
}

grid_search = GridSearchCV(
    estimator=LogisticRegression(random_state=42, class_weight='balanced'),
    param_grid=param_grid,
    scoring='f1',
    cv=5,
    n_jobs=-1
)

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train_scaled, y_train)


best_params = grid_search.best_params_
print(f"Best hyperparameters: {best_params}")
```

```
# Get the best model
best_model = grid_search.best_estimator_
```

```
Best hyperparameters: {'C': 0.001, 'penalty': 'l1', 'solver': 'liblinear'}
```

Coefficients

In [214…

```
# Getting the best model from GridSearchCV
best_model = grid_search.best_estimator_

# Training the best model
best_model.fit(X_train, y_train)

y_pred = best_model.predict(X_test)

print(best_model.coef_)
```

```
[[ 0.          0.          0.          0.         -0.00185827  0.00627214
   0.          0.          0.          0.        ]]
```

Intl_plan_no: Has a negative coefficient (-0.31261138), suggesting that an increase in this feature's value is associated with a decreased likelihood of churn.

Customer_service_calls: Has a positive coefficient (0.50077847), suggesting that an increase in this feature's value is associated with Total

International_calls: Has a small negative coefficient (-0.00731615), indicating a very weak negative relationship with churn.

Total_intl_minutes: Has a small positive coefficient (0.00463985), indicating a very weak positive relationship with churn.

Intl_plan_yes: Has a positive coefficient (0.32408861), suggesting that an increase in this feature's value is associated with an increased likelihood of churn.

Total_charges: Has a positive coefficient (0.47887169), suggesting that an increase in this feature's value is associated with an increased likelihood of churn.

The features with zero coefficients (feature_1, feature_2, feature_8, feature_9, feature_10, feature_11, feature_12) do not have a significant impact on churn prediction in this model.

DEPLOYMENT

KEY FACTORS INFLUENCING CHURN

1.Contract Length: The correlation heatmap suggests a strong negative correlation between contract_length and churn, indicating that customers with longer contracts are less likely to churn.

2.Monthly Charges: There seems to be a positive correlation between monthly_charges and churn, implying that higher monthly charges increase the risk of churn

RECOMMENDATIONS

-Pricing: Since "monthly charges" is a strong predictor, consider offering pricing plans with more gradual increases to avoid sudden price hikes that might trigger churn.

-Investigate if customers with international plans have specific needs or concerns that are not being addressed.

-Offering incentives for longer contracts

-Customer Onboarding: If certain features or services are associated with lower churn, emphasize them during onboarding to increase customer engagement and satisfaction.

-Loyalty Programs: Design loyalty programs that reward long-term customers and encourage them to stay.

In [ ]: