# Final Project Submission

Please fill out:

Student name:

***Alvin Kimathi***

- Student pace: Hybrid Full-Time
- Scheduled project review date/time: 7th June 2024
- Instructor name: Maryann Mwikali
- Blog post URL:https://github.com/Kimathi26/Dsc-Phase-3-Project

# BUSINESS PROBLEM

SyriaTel, a tele-communications company in the Middle East has brought to our attention(Anonymous Analysts) the upward rate in which its clients are stopping usage of their product. The are looking to get ahead of the customer churn as it is easier to retain and get a return client than it is to onboard a new client.The company therefore wants to take pro-active measures to identify those likely to churn and retain them while improving customer satisfaction and reducing revenue loss.

## Objectives

- Identify Key factors leading to churn
- Develop a predictive model that accurately predicts whether a customer will churn
- After visualizations propose necessary interventions

# 1: DATA UNDERSTANDING

We obtained our data set from Kaggle, an online hosting service which provided us with the necessary dataset to analyze this field.

### 1.1: Import Necessary Libraries

In [ ]:

```
# Import the necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Preprocessing and model building

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.compose import ColumnTransformer
```

```python
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

#For Models

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier


#For Evaluation

from sklearn.metrics import roc_auc_score, confusion_matrix, classification_report, acc
```

## 1.2: Load our Dataset

In [ ]:

```python
data = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
data.head()
```

Out[ ]:

| | state | account length | area code | phone number | international plan | voice mail plan | number vmail messages | total day minutes | total day calls | total day charge | ... | total eve calls | to cha |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | KS | 128 | 415 | 382-4657 | no | yes | 25 | 265.1 | 110 | 45.07 | ... | 99 | 16 |
| **1** | OH | 107 | 415 | 371-7191 | no | yes | 26 | 161.6 | 123 | 27.47 | ... | 103 | 16 |
| **2** | NJ | 137 | 415 | 358-1921 | no | no | 0 | 243.4 | 114 | 41.38 | ... | 110 | 10 |
| **3** | OH | 84 | 408 | 375-9999 | yes | no | 0 | 299.4 | 71 | 50.90 | ... | 88 | 5 |
| **4** | OK | 75 | 415 | 330-6626 | yes | no | 0 | 166.7 | 113 | 28.34 | ... | 122 | 12 |

5 rows × 21 columns

## 1.3: Explore and Understand our Data set

In [ ]:

```python
print(data.info())
print(data.describe())
```

```
<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
None
       account length    area code  number vmail messages  total day minutes  \
count     3333.000000  3333.000000            3333.000000        3333.000000
mean       101.064806   437.182418               8.099010         179.775098
std         39.822106    42.371290              13.688365          54.467389
min          1.000000   408.000000               0.000000           0.000000
25%         74.000000   408.000000               0.000000         143.700000
50%        101.000000   415.000000               0.000000         179.400000
75%        127.000000   510.000000              20.000000         216.400000
max        243.000000   510.000000              51.000000         350.800000

       total day calls  total day charge  total eve minutes  total eve calls  \
count      3333.000000       3333.000000        3333.000000      3333.000000
mean        100.435644         30.562307         200.980348       100.114311
std          20.069084          9.259435          50.713844        19.922625
min           0.000000          0.000000           0.000000         0.000000
25%          87.000000         24.430000         166.600000        87.000000
50%         101.000000         30.500000         201.400000       100.000000
75%         114.000000         36.790000         235.300000       114.000000
max         165.000000         59.640000         363.700000       170.000000

       total eve charge  total night minutes  total night calls  \
count       3333.000000          3333.000000        3333.000000
mean          17.083540           200.872037         100.107711
std            4.310668            50.573847          19.568609
min            0.000000            23.200000          33.000000
25%           14.160000           167.000000          87.000000
50%           17.120000           201.200000         100.000000
75%           20.000000           235.300000         113.000000
max           30.910000           395.000000         175.000000

       total night charge  total intl minutes  total intl calls  \
count         3333.000000         3333.000000       3333.000000
mean             9.039325           10.237294          4.479448
std              2.275873            2.791840          2.461214
min              1.040000            0.000000          0.000000
25%              7.520000            8.500000          3.000000
50%              9.050000           10.300000          4.000000
75%             10.590000           12.100000          6.000000
max             17.770000           20.000000         20.000000

       total intl charge  customer service calls
count        3333.000000             3333.000000
mean            2.764581                1.562856
std             0.753773                1.315491
min             0.000000                0.000000
25%             2.300000                1.000000
50%             2.780000                1.000000
```

```
75%            3.270000              2.000000
max            5.400000              9.000000
```

# 2: DATA CLEANING AND PREPROCESSING

## 2.1: Handle any Missing Values

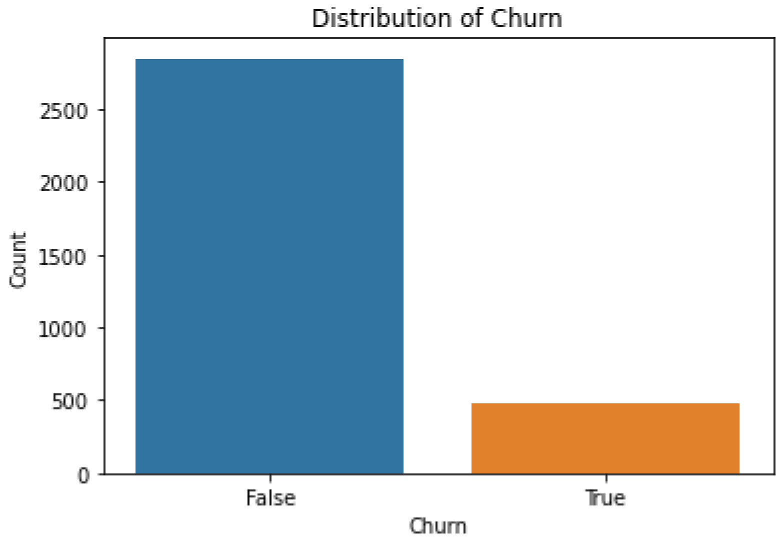
In [ ]:
```
#Check for any missing values
print("\nMissing Values in Data:")
print(data.isnull().sum())
```

```
Missing Values in Data:
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
```

# 3: EXPLORATIVE DATA ANALYSIS

## 3.1: Distribution of Churn

In [ ]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x='churn', data=data)
plt.title('Distribution of Churn')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()
#Percentage of True/False in the distribution
churn_percentage = data['churn'].value_counts(normalize=True)* 100
churn_percentage
```



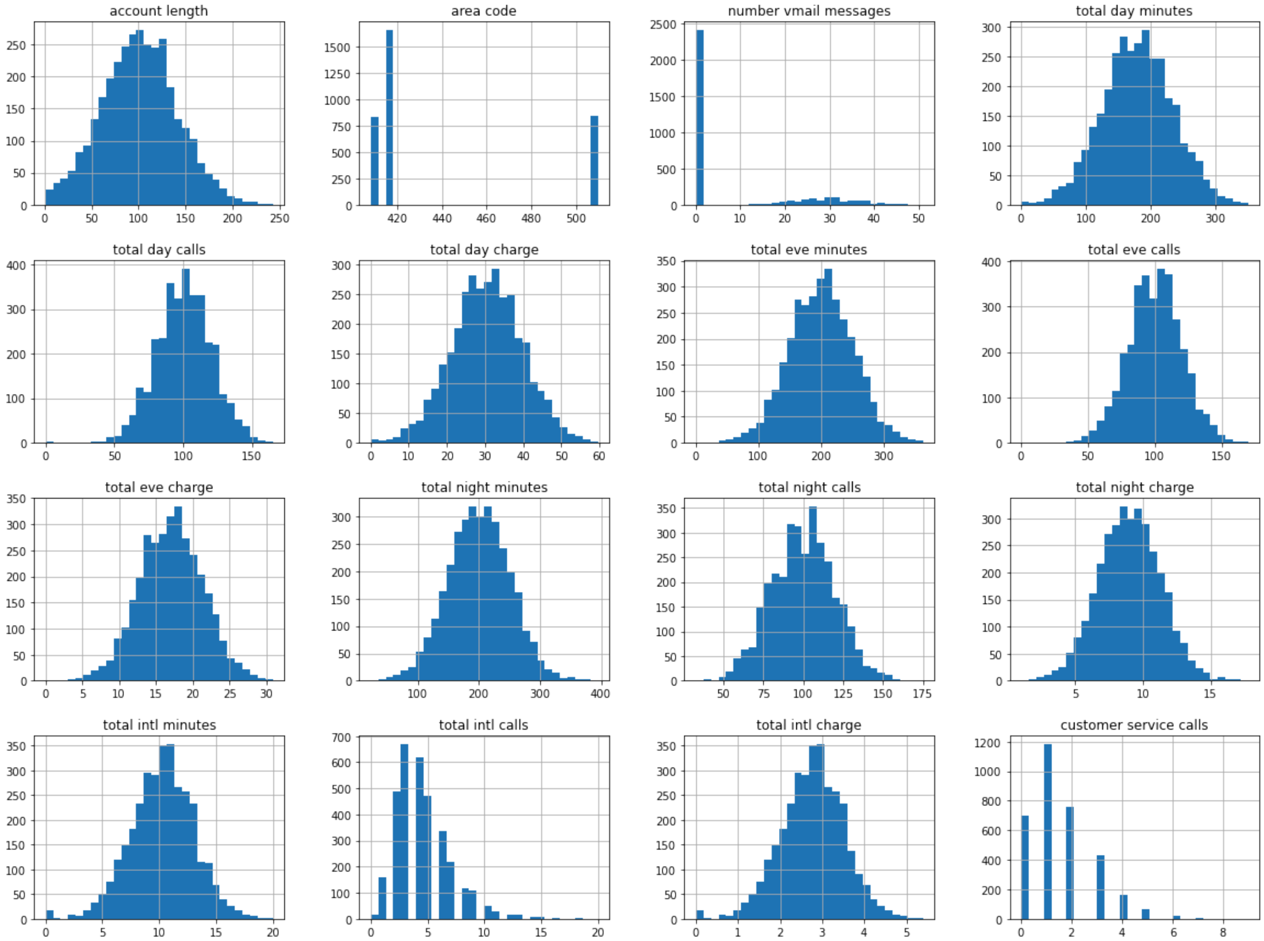

Out[ ]:
```
False    85.508551
True     14.491449
Name: churn, dtype: float64
```

From this, we note that approximately 85% of the customers are still using the service while 14% have actively stopped using the service.

## 3.2: Univariate Analysis

### 3.2.1: Distribution of Numerical features

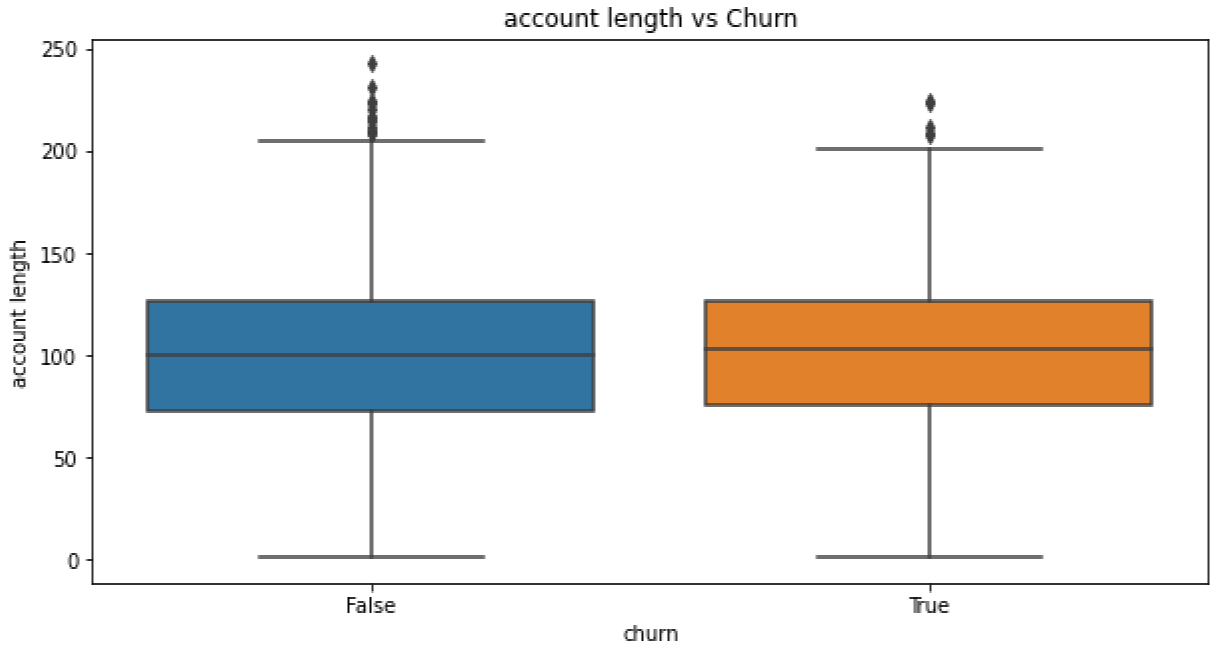
In [ ]:
```
#Histograms for the numeric variables excluding boolean colums
#Define numeric features
numeric_features = data.select_dtypes(include=['float64', 'int64']).columns
data[numeric_features].hist(bins=30, figsize=(20, 15))
plt.show()
```

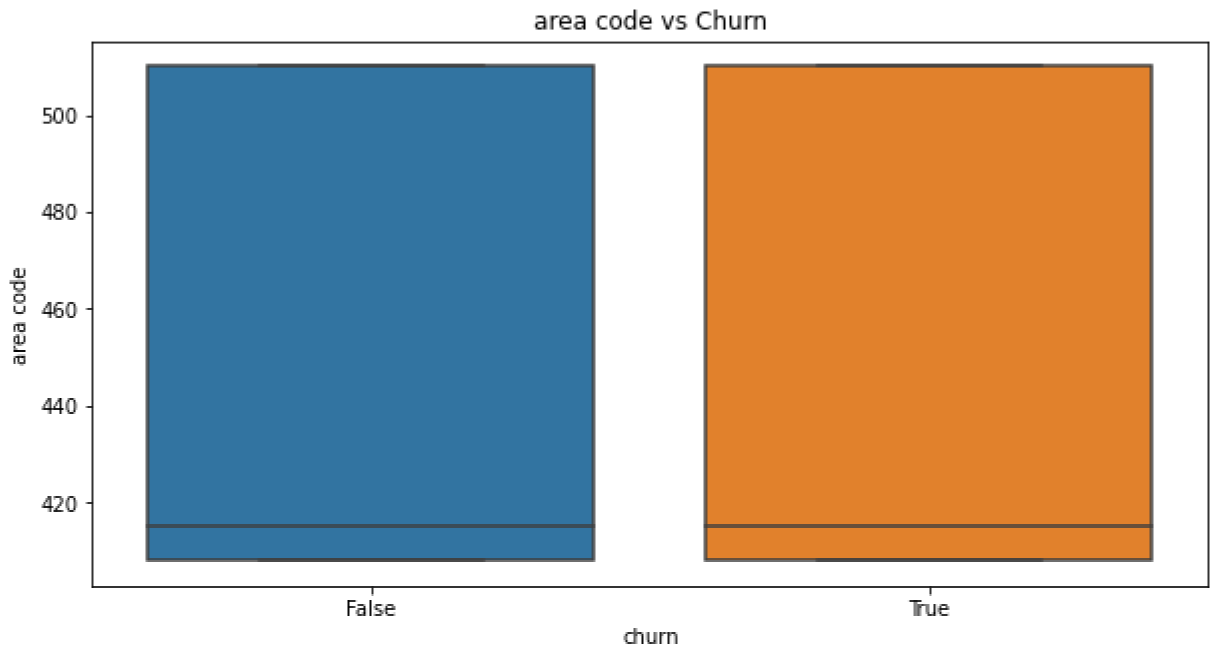


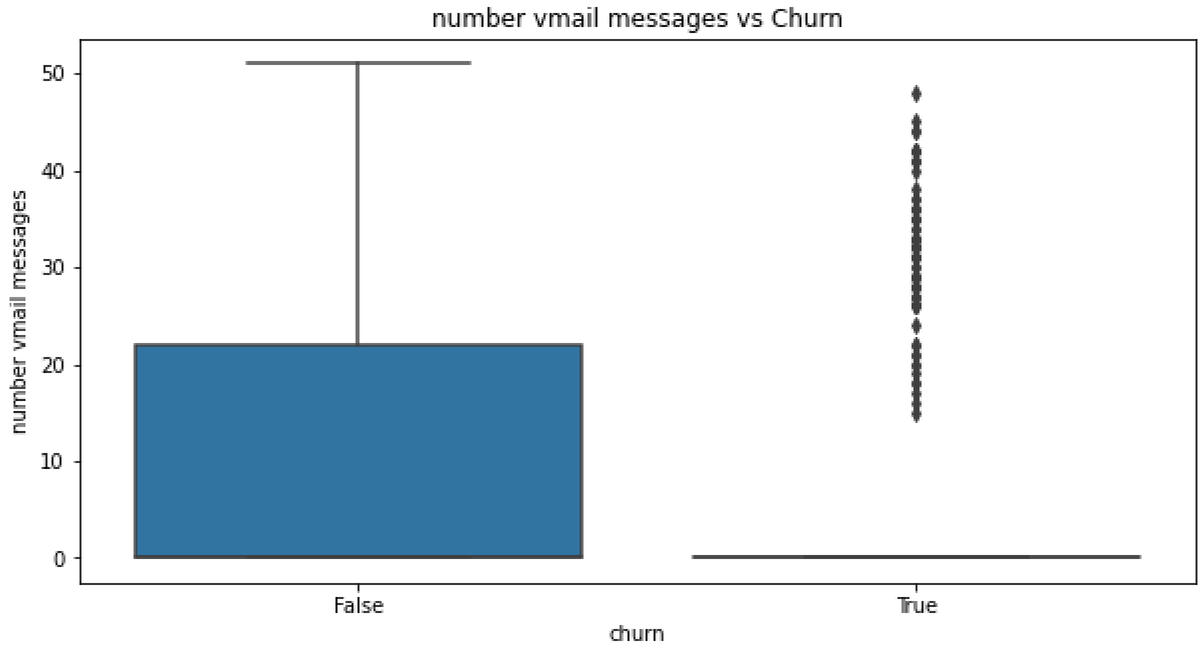

# 3.3: BiVariate Analysis

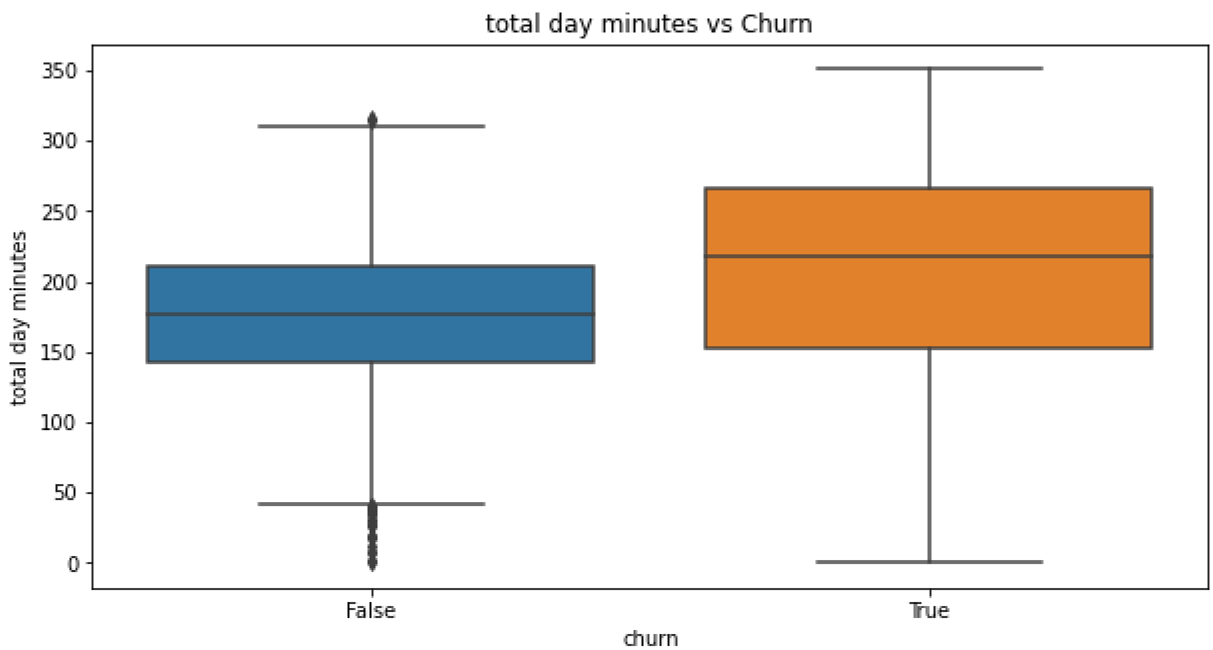
## 3.3.1: Box Plots for Numeric Features vs Churn

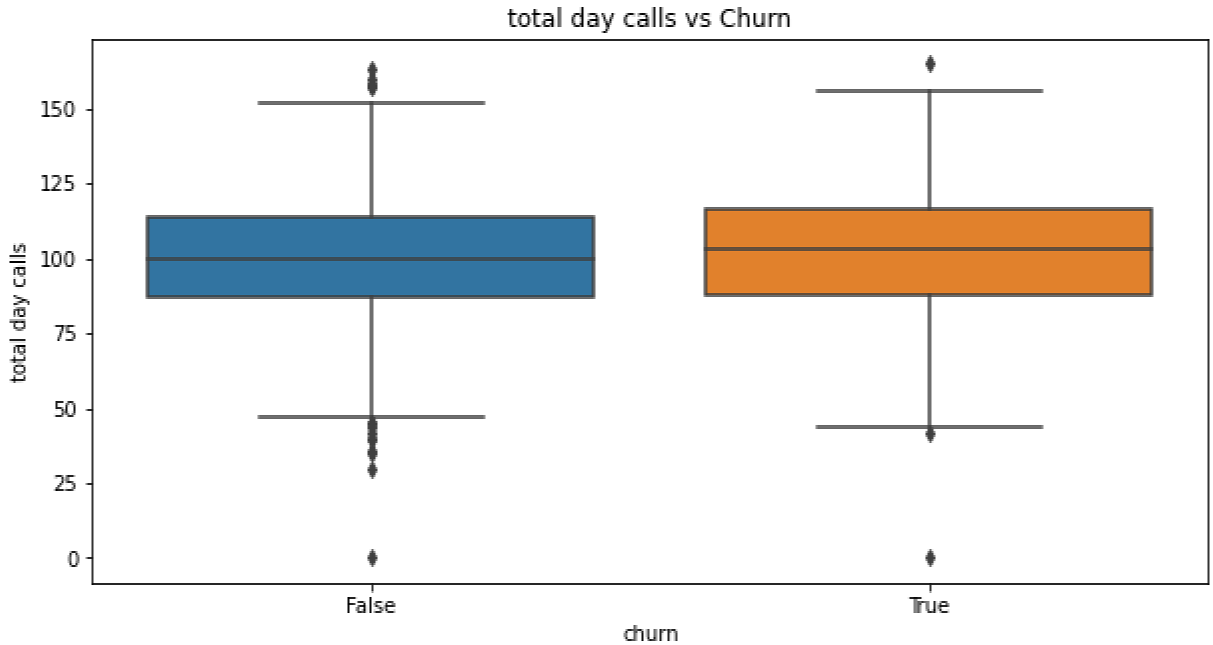
In [ ]:
```
#exclude the 'churn' column
numeric_features = [feature for feature in numeric_features if feature != 'churn']

# Plot box plots for numerical features against Churn
for feature in numeric_features:
    plt.figure(figsize=(10, 5))
    sns.boxplot(x='churn', y=feature, data=data)
    plt.title(f'{feature} vs Churn')
    plt.show()
```

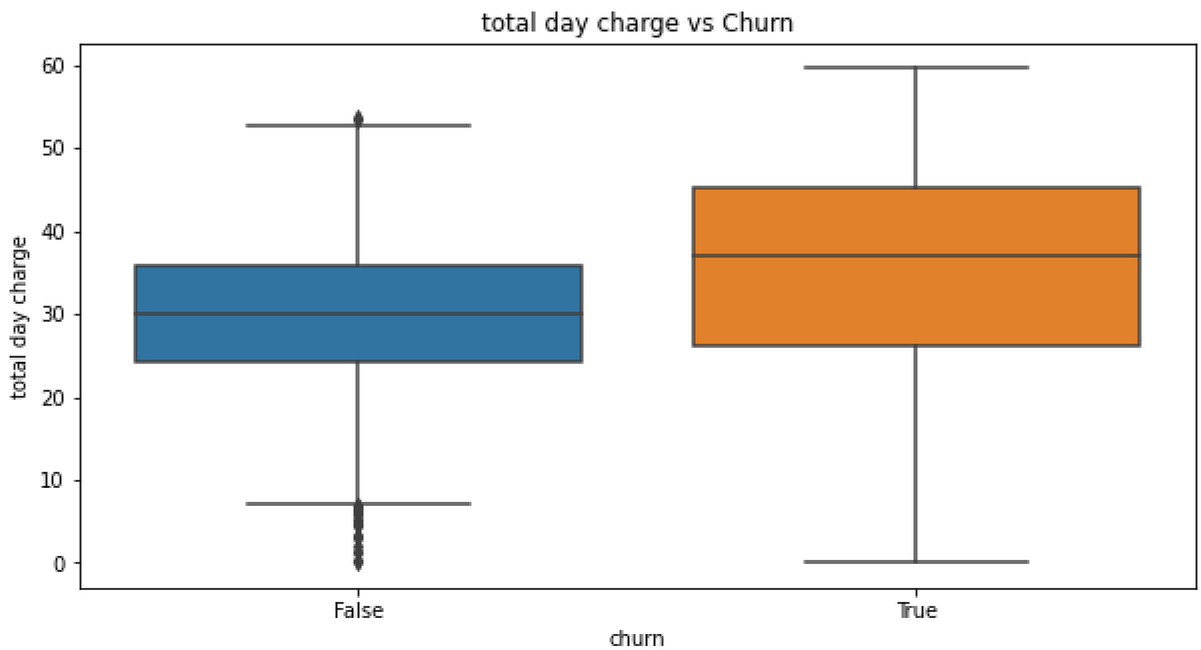

account length vs Churn
250
200
150
100
50
0
account length
False
True
churn

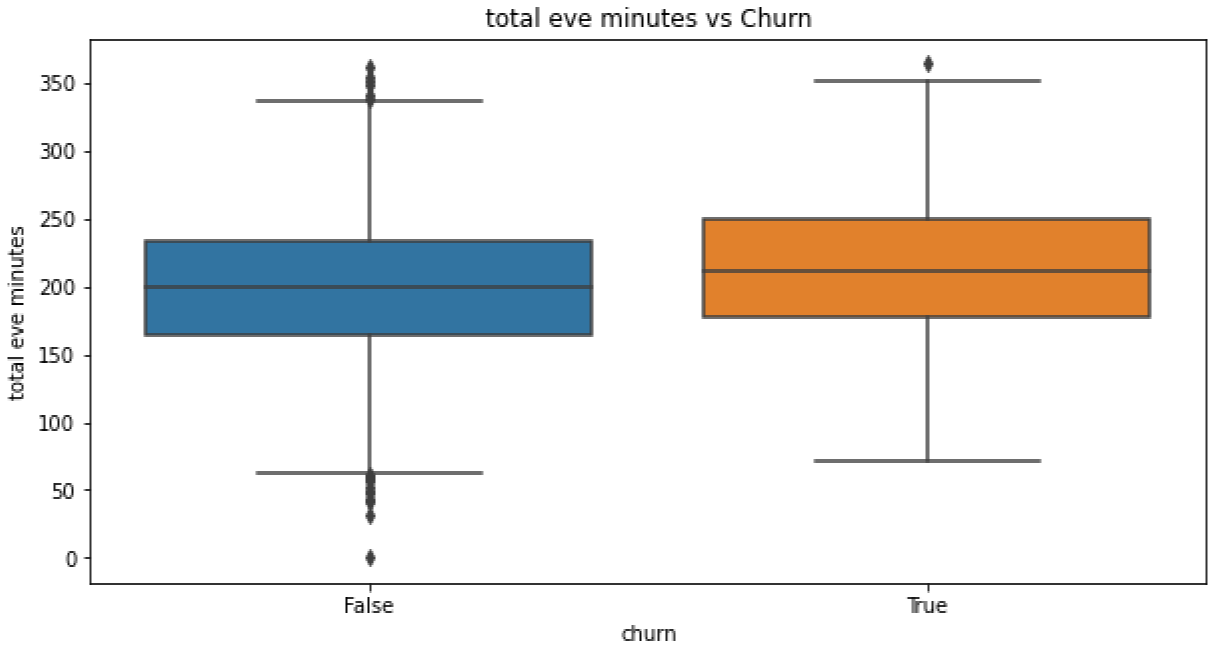

area code vs Churn
500
480
460
440
420
area code
False
True
churn

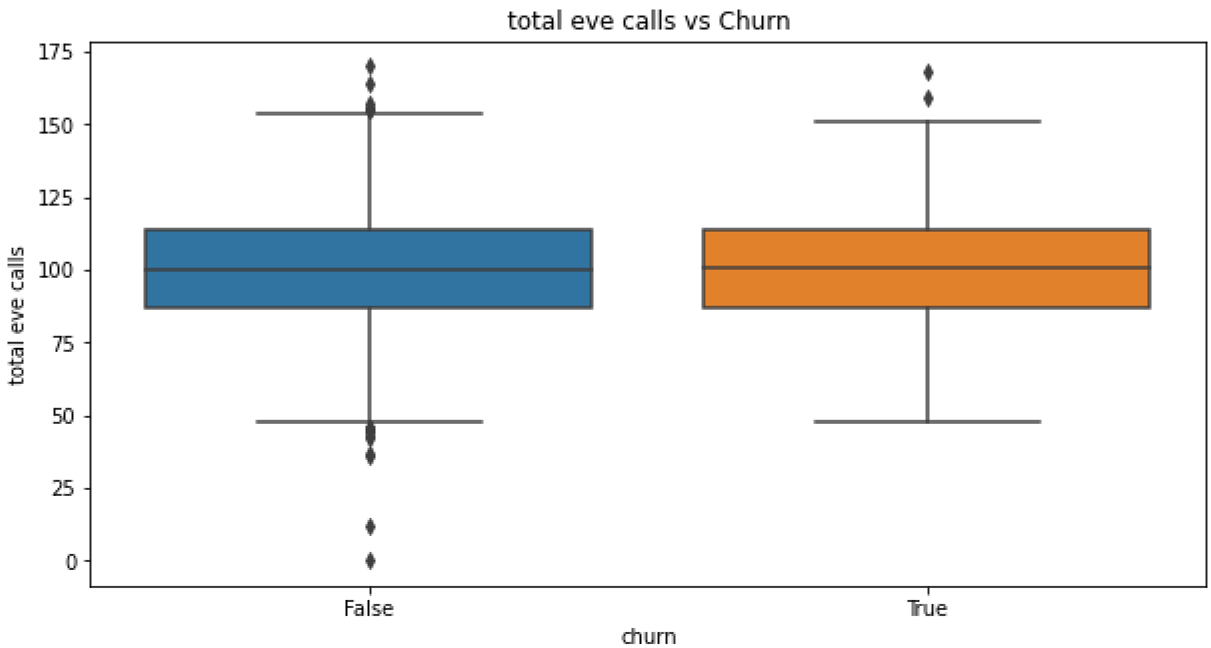

number vmail messages vs Churn
number vmail messages
50
40
30
20
10
0
False
True
churn

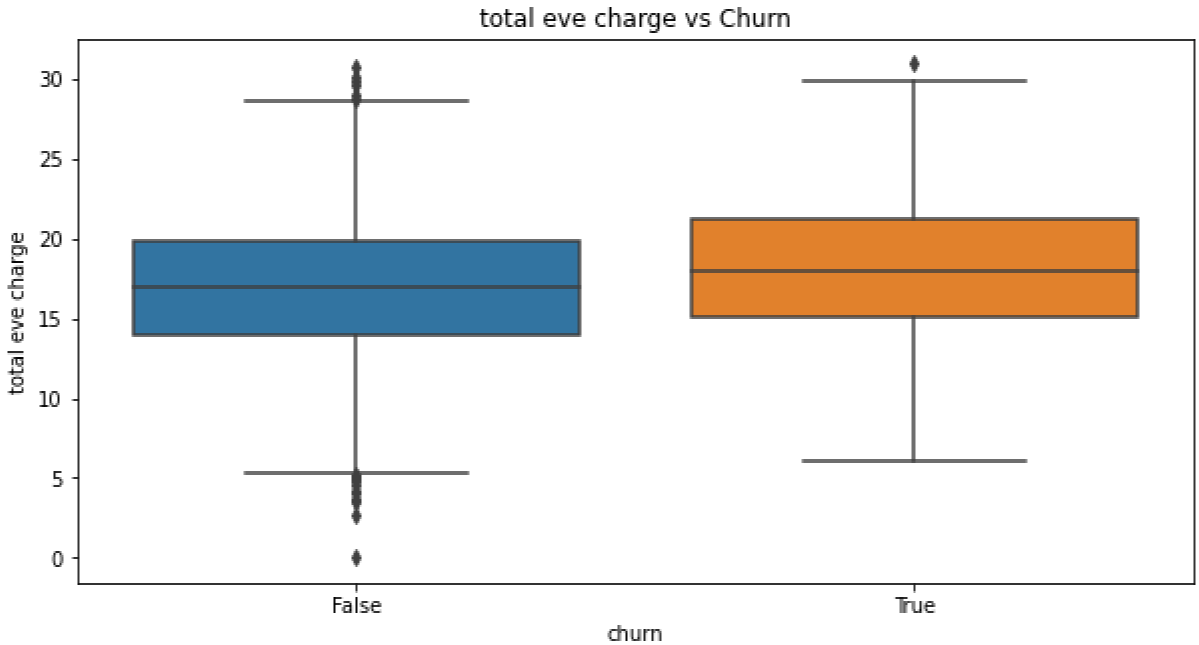

total day minutes vs Churn
total day minutes
350
300
250
200
150
100
50
0
False
True
churn

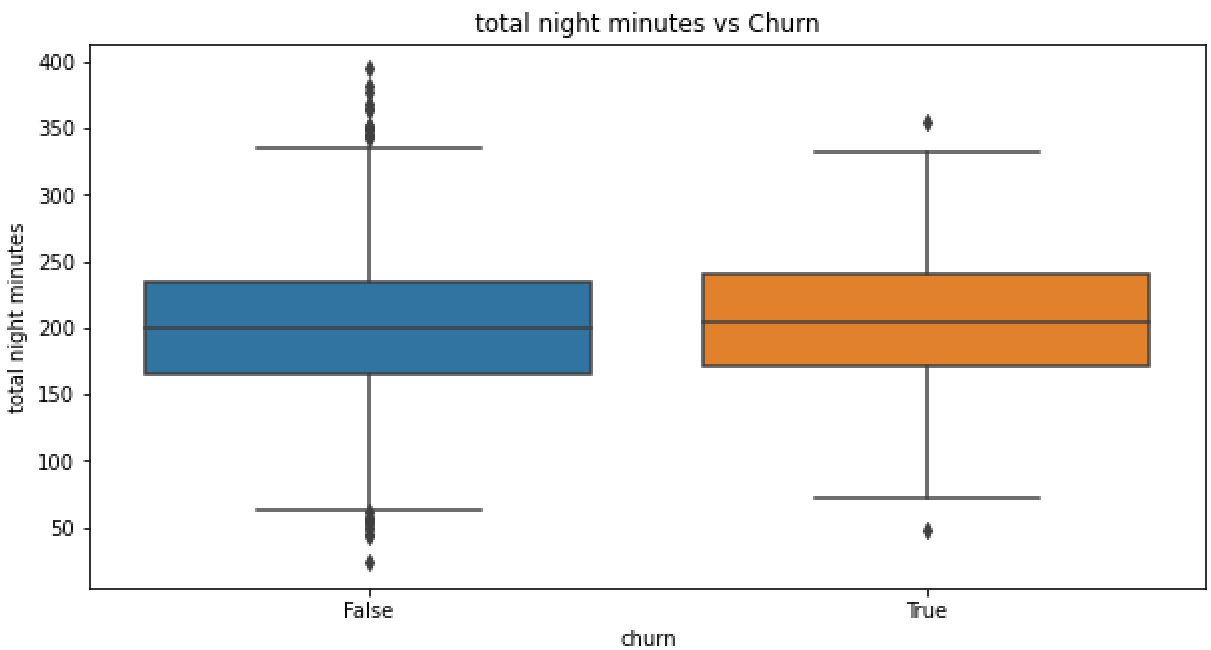

total day calls vs Churn
total day calls
150
125
100
75
50
25
0
False
True
churn

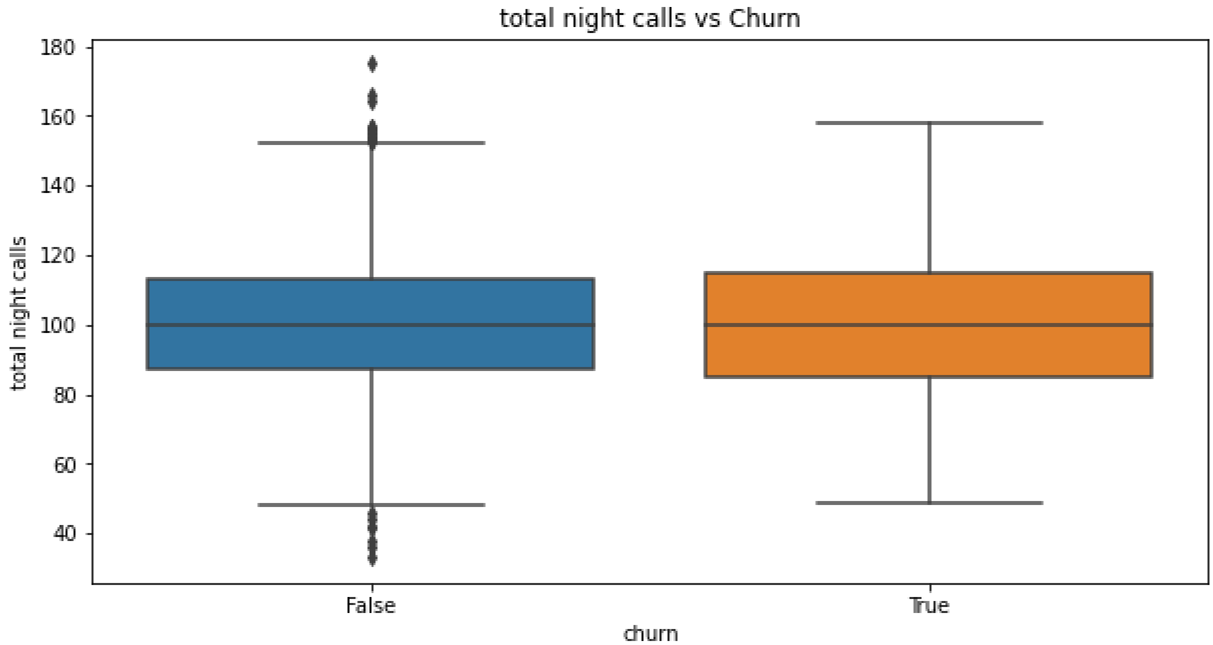

total day charge vs Churn
total day charge
60
50
40
30
20
10
0
False
True
churn

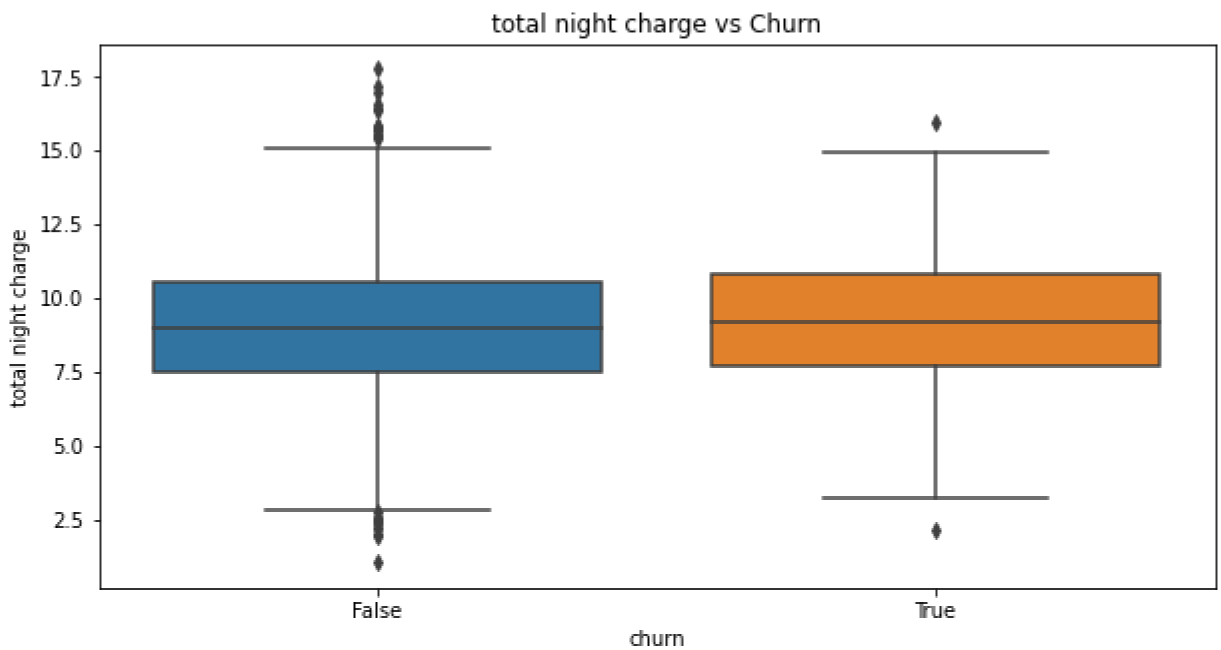

total eve minutes vs Churn
total eve minutes
350
300
250
200
150
100
50
0
False
True
churn

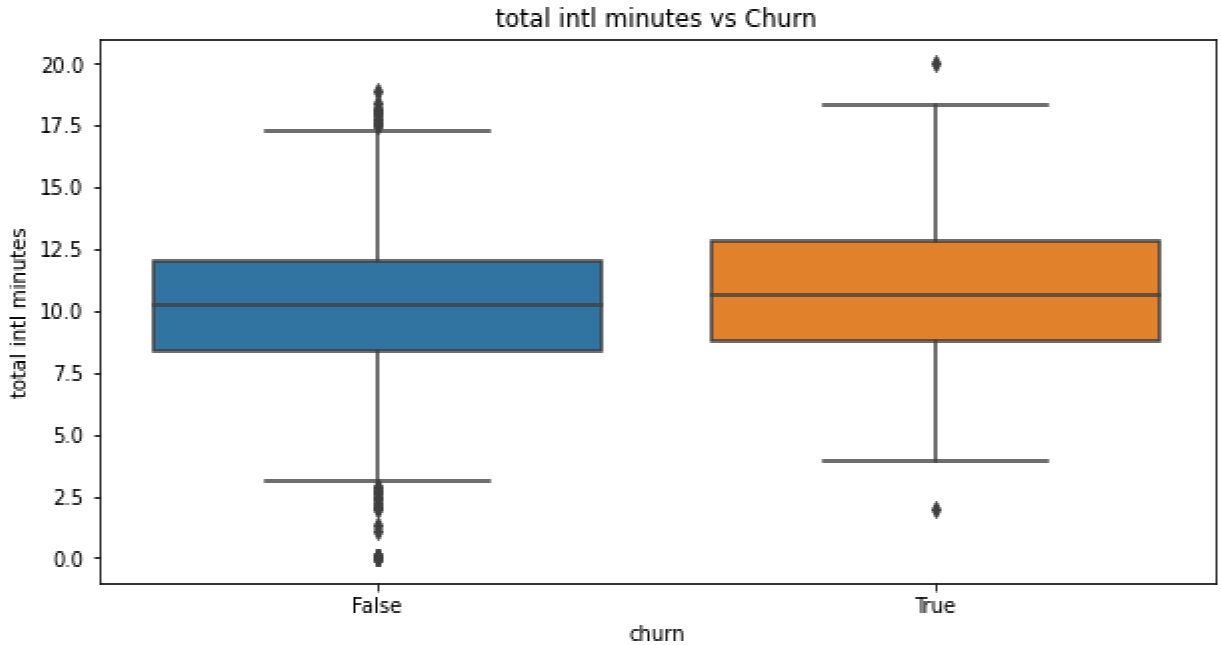

total eve calls vs Churn
total eve calls
175
150
125
100
75
50
25
0
False
True
churn

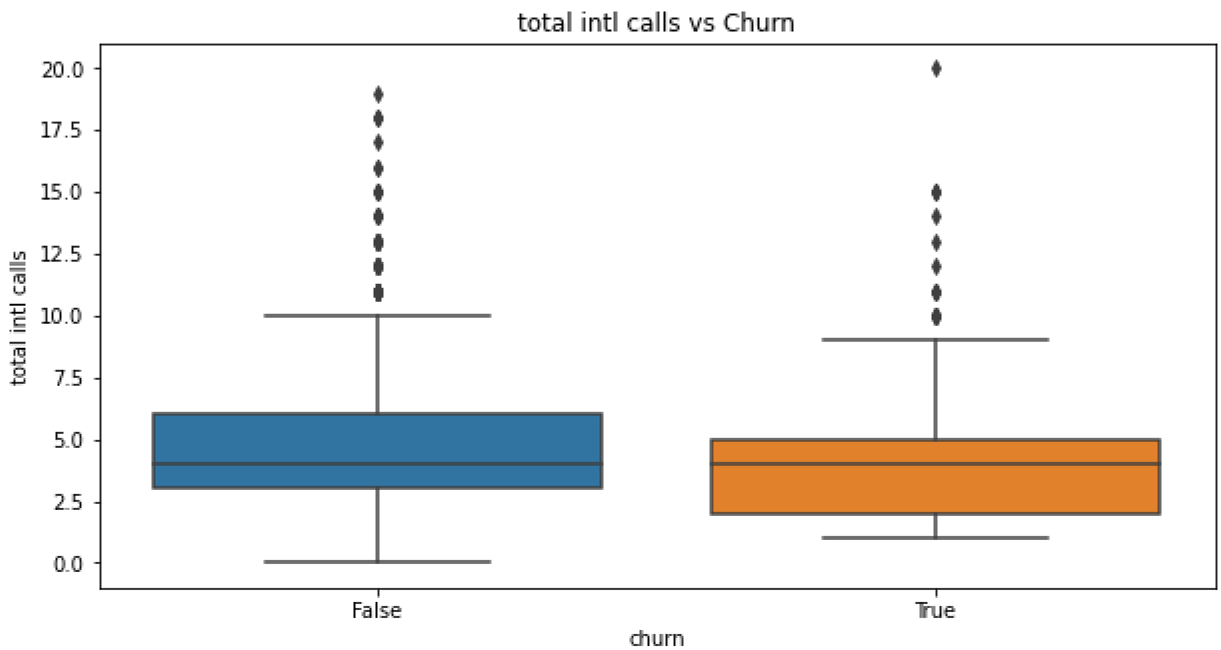

total eve charge vs Churn
30
25
20
15
10
5
0
total eve charge
False
True
churn

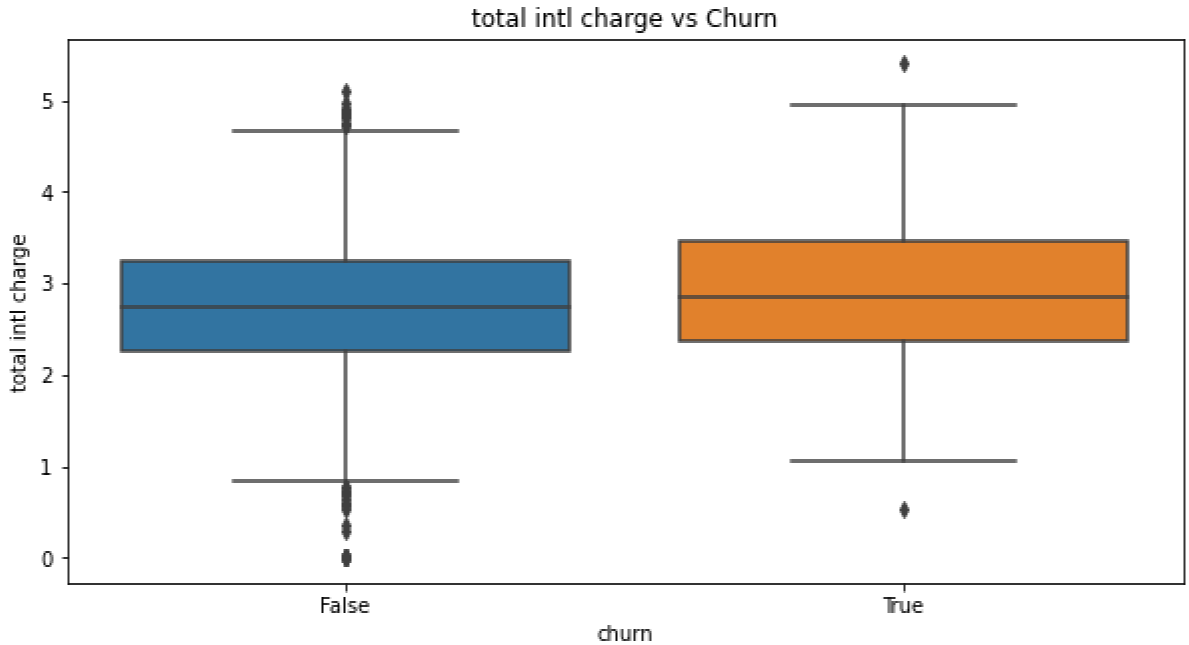

total night minutes vs Churn
400
350
300
250
200
150
100
50
total night minutes
False
True
churn

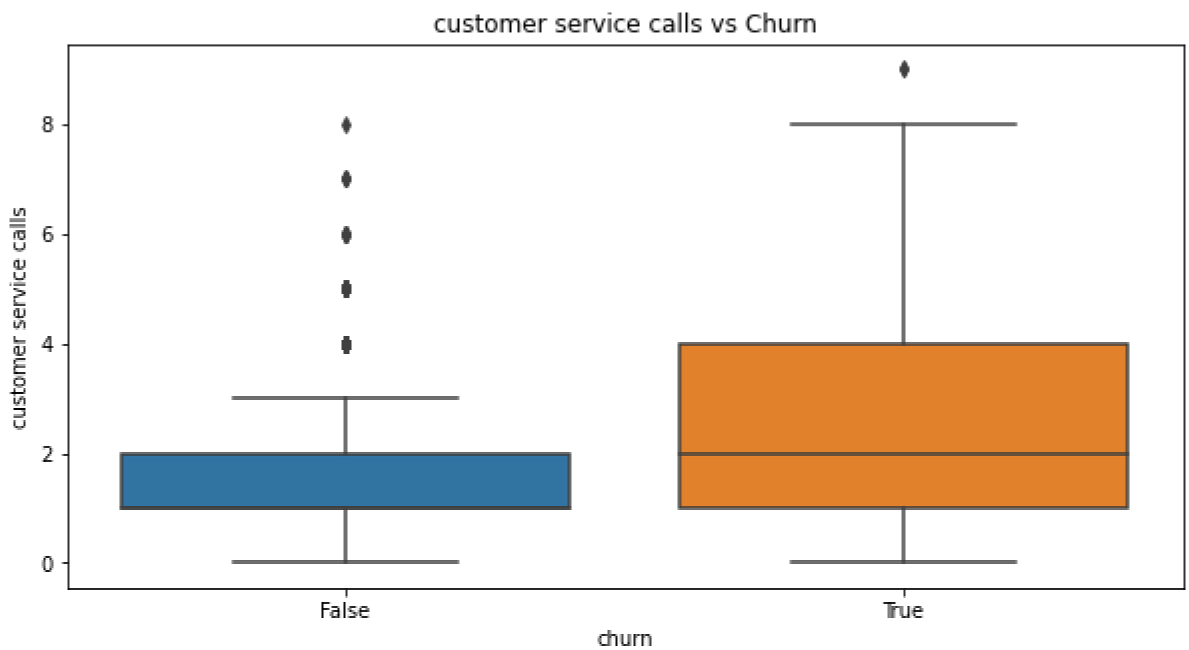

total night calls vs Churn
total night calls
180
160
140
120
100
80
60
40
False
True
churn


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


total intl minutes vs 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

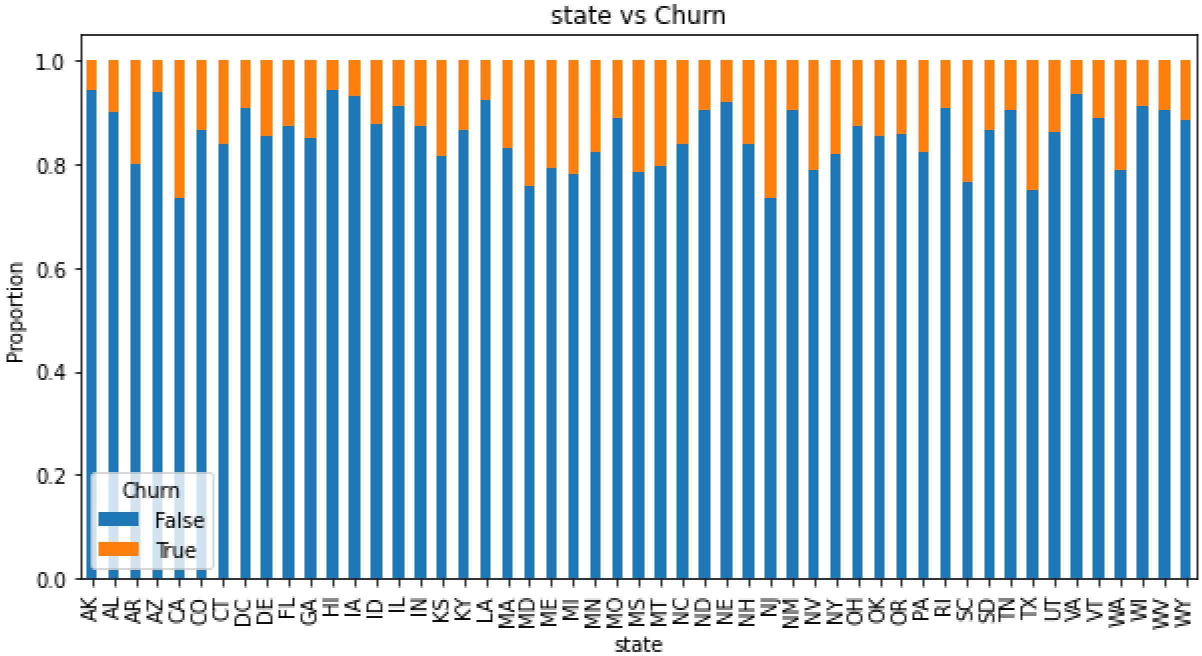

total intl calls vs Churn
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
total intl calls
False
True
churn

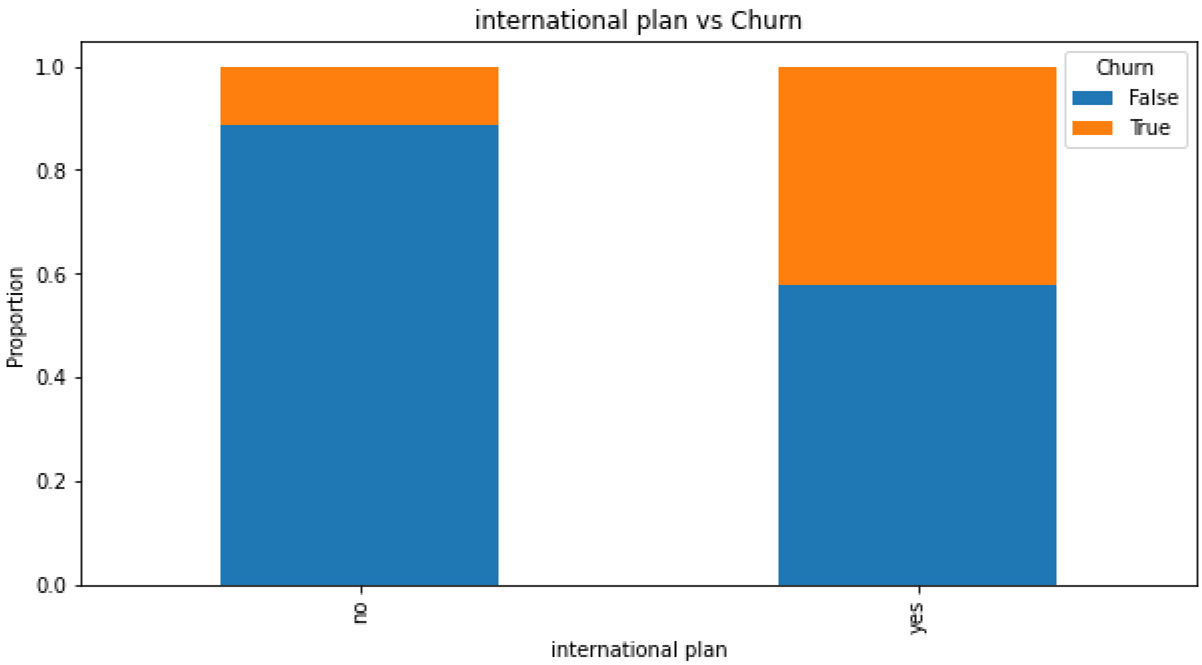


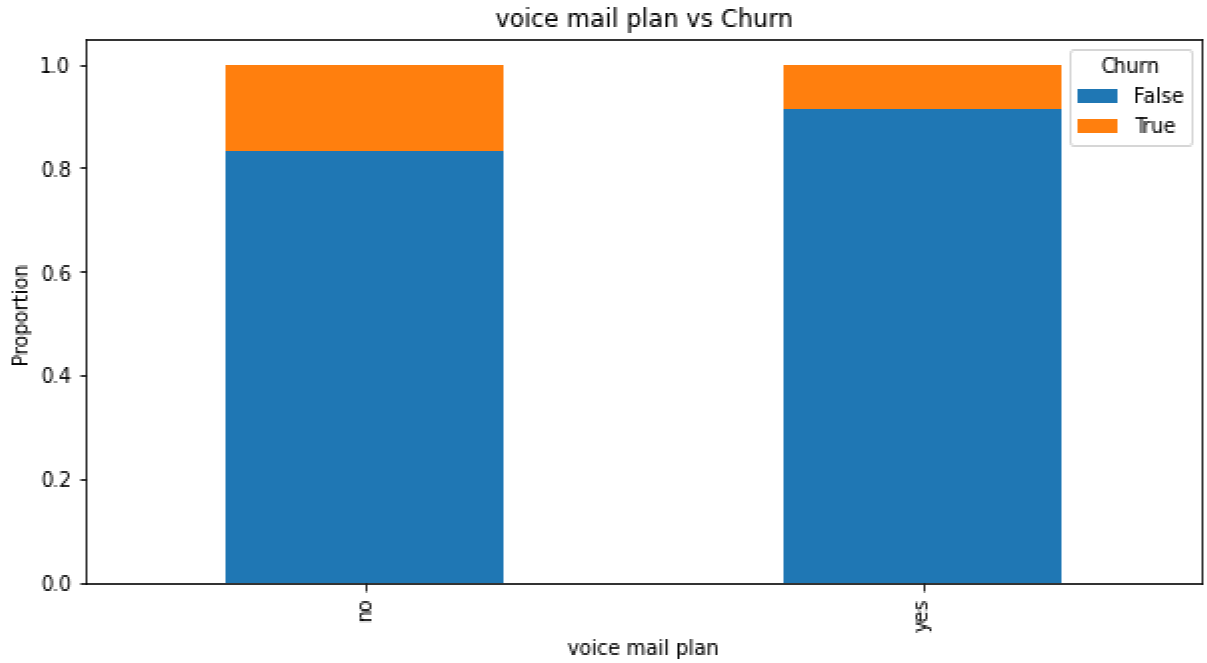




### 3.3.2: Count plots of Categorical features to Churn

In [ ]:
```
# Identify categorical features and exclude the 'phone number' column
categorical_features = data.select_dtypes(include=['object']).columns
categorical_features = [feature for feature in categorical_features if feature != 'phon

# Plot cross-tabulations for categorical features against the target variable
for feature in categorical_features:
    cross_tab = pd.crosstab(data[feature], data['churn'], normalize='index')
    cross_tab.plot(kind='bar', stacked=True, figsize=(10, 5))
    plt.title(f'{feature} vs Churn')
    plt.ylabel('Proportion')
    plt.xlabel(feature)
    plt.legend(title='Churn')
    plt.show()
```


state vs Churn
Proportion
0.0
0.2
0.4
0.6
0.8
1.0
Churn
False
True
AK AL AR AZ CA CO CT DC DE FL GA HI IA ID IL IN KS KY LA MA MD ME MI MN MO MS MT NC ND NE NH NJ NM NV NY OH OK OR PA RI SC SD TN TX UT VA VT WA WI WV WY
state

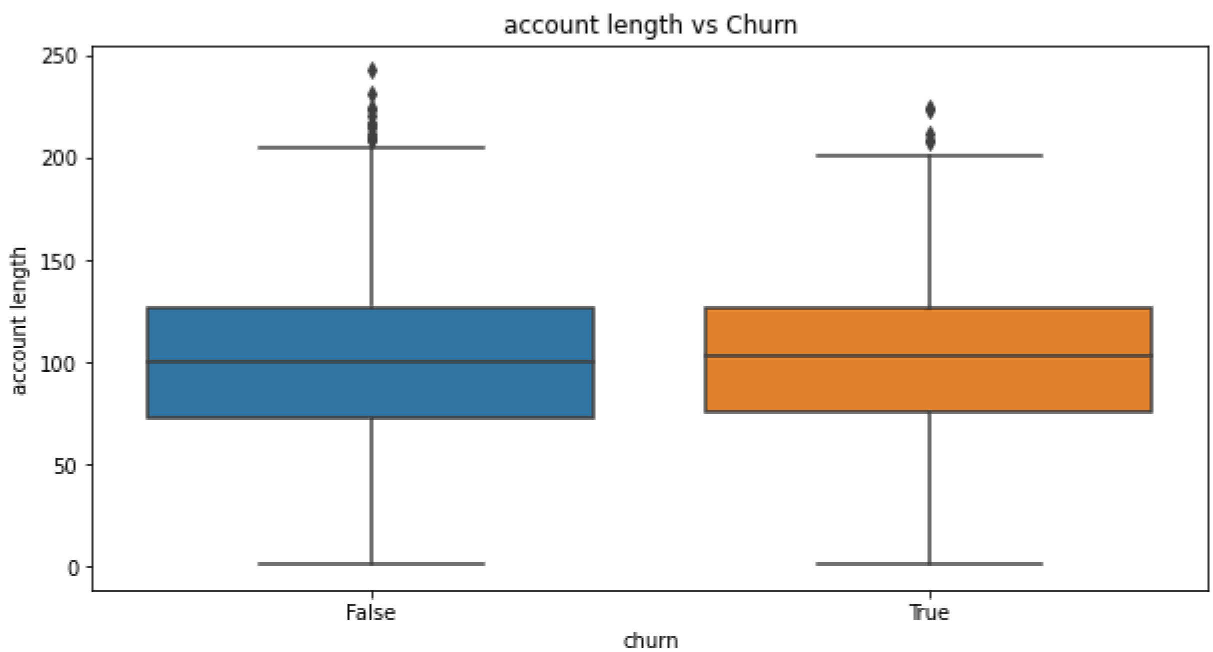

international plan vs Churn
Proportion
0.0
0.2
0.4
0.6
0.8
1.0
Churn
False
True
no
yes
international plan

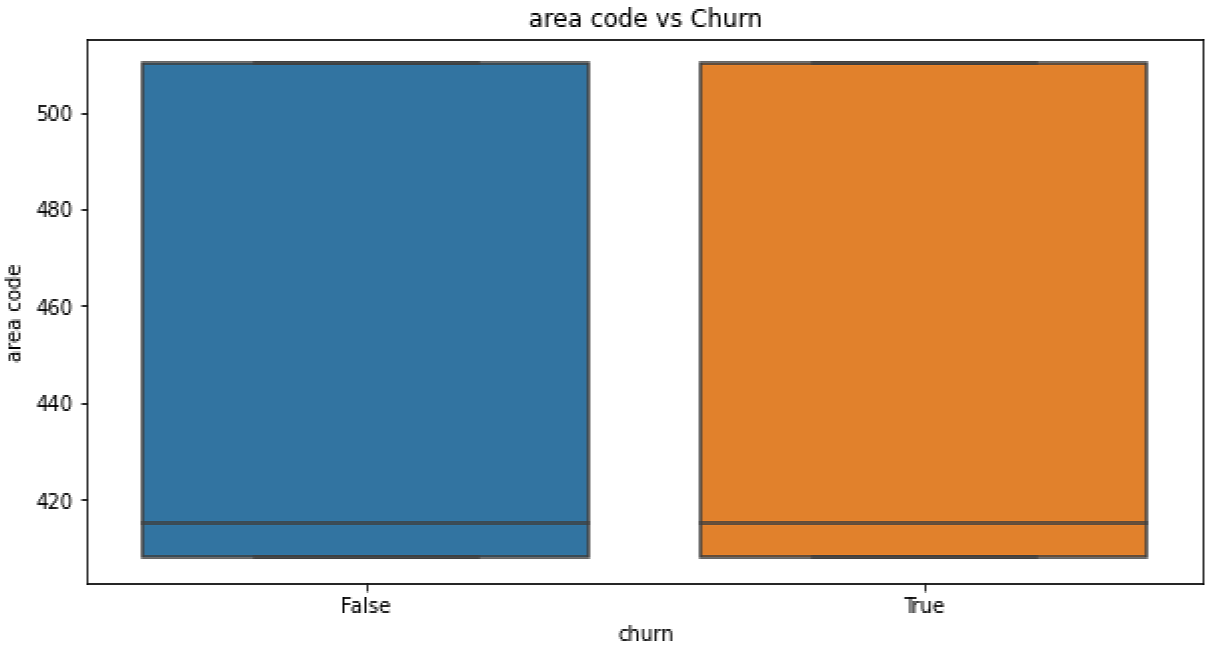


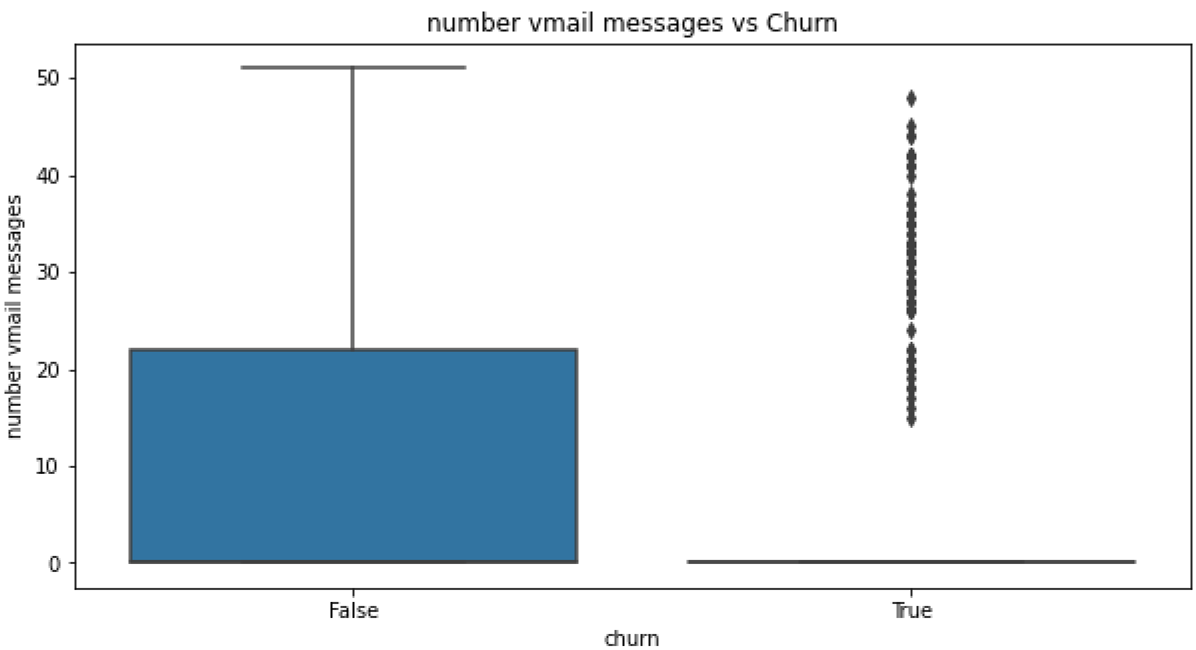

From this, one outstanding observation is that customers with an international plan are more likely to churn than those without.....probably due to higher costs if the plan is not enough to justify the cost and maybe for a short-term need as well.

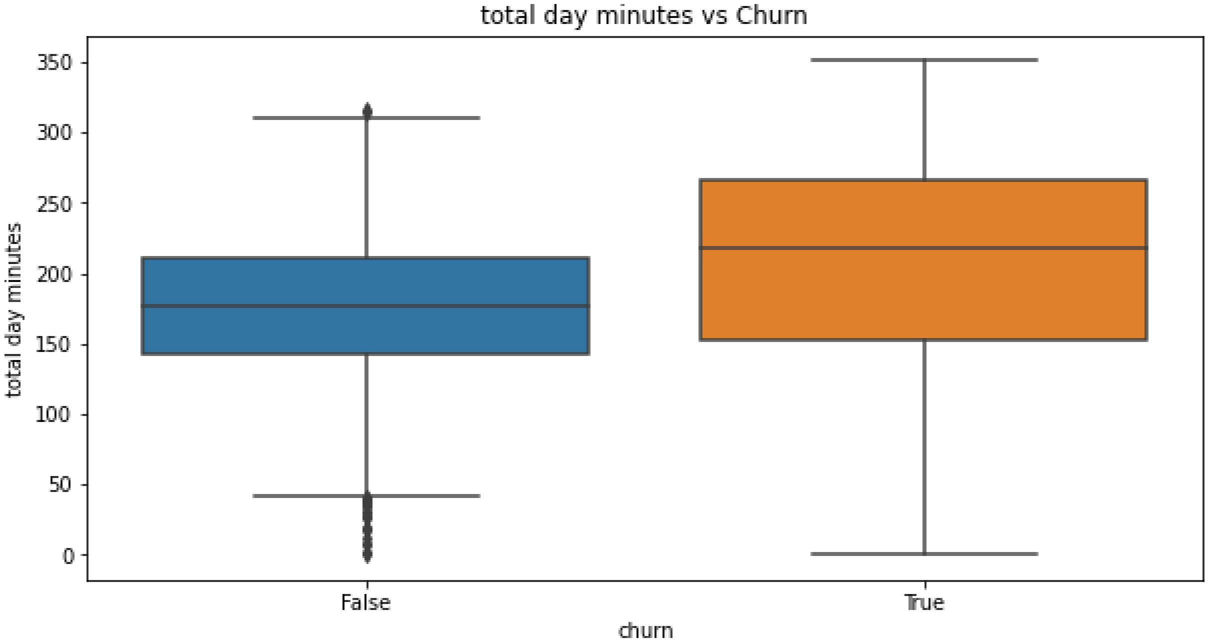
### 3.3.3: Relationship between Numerical features and Churn

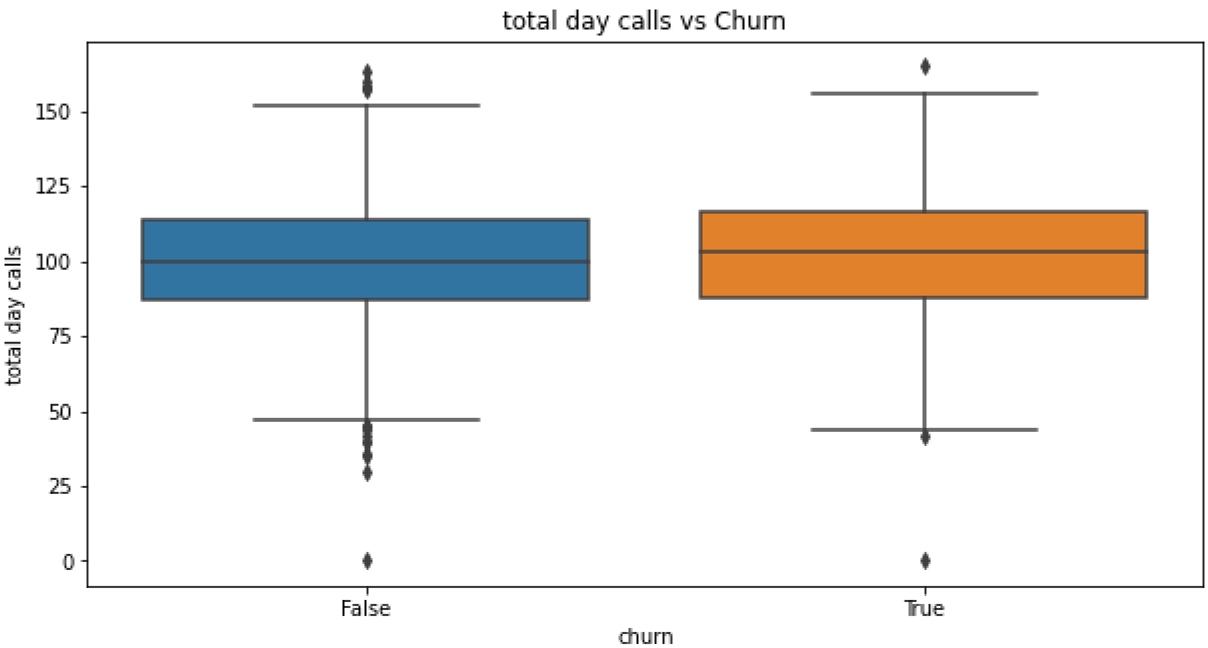
In [ ]:
```python
# Box plots for numeric features against the target variable(Churn)

for feature in numeric_features:
    plt.figure(figsize=(10, 5))
    sns.boxplot(x='churn', y=feature, data=data)
    plt.title(f'{feature} vs Churn')
    plt.show()
```

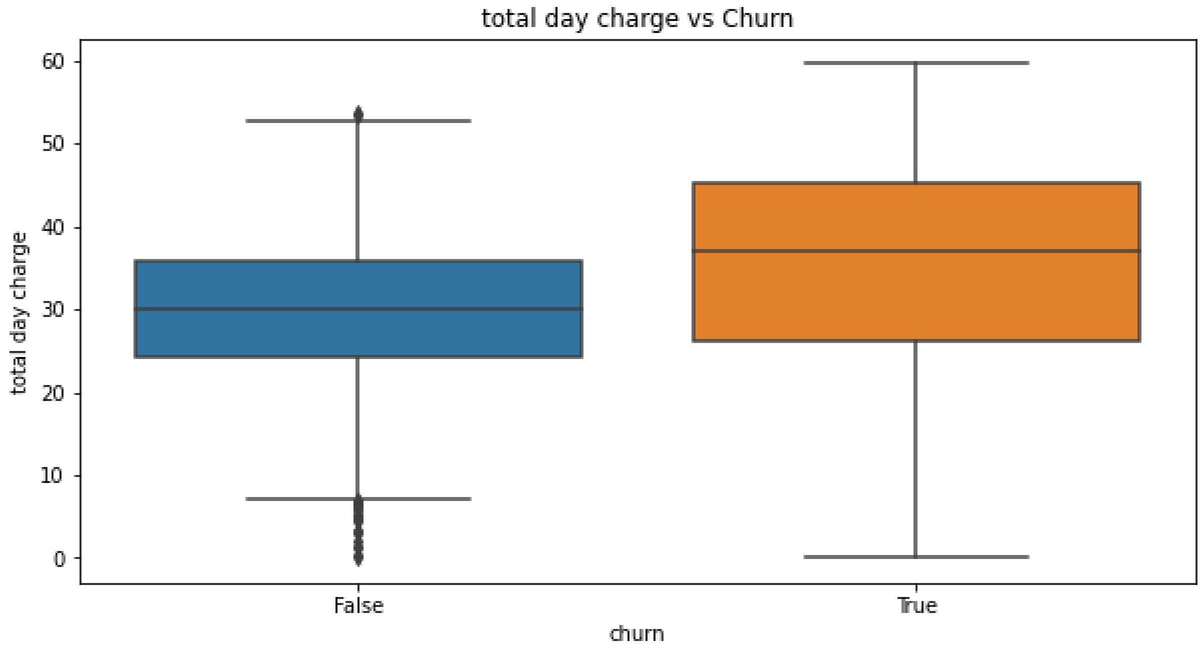


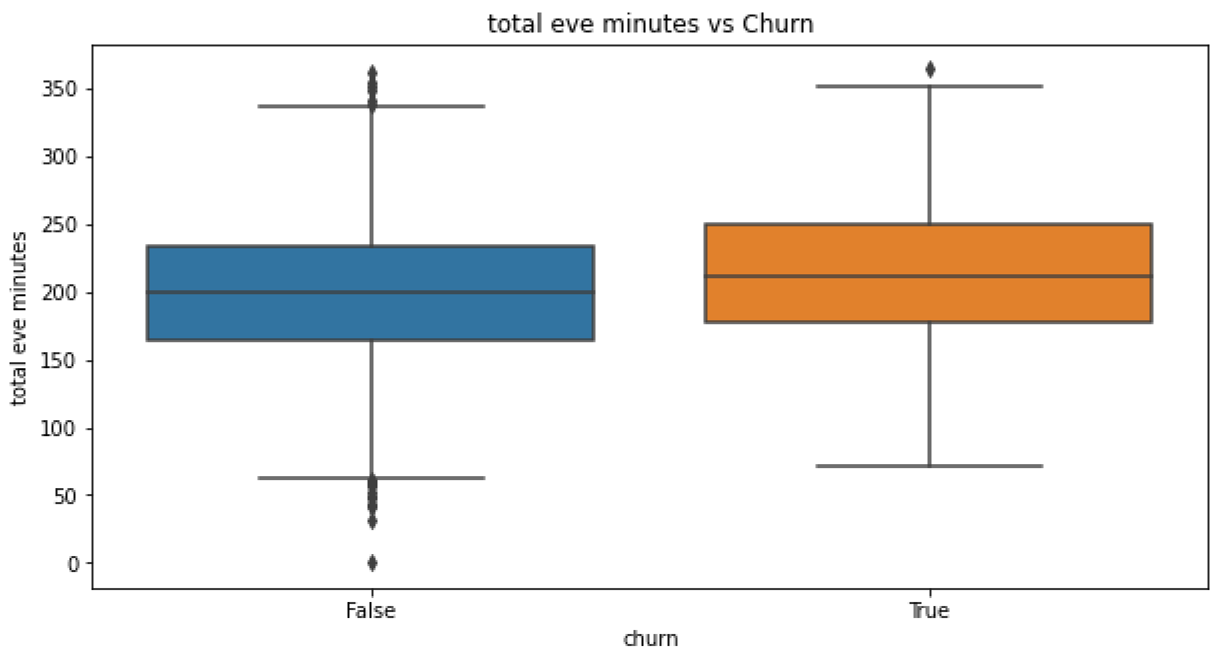

area code vs Churn
500
480
460
440
420
area code
False
True
churn

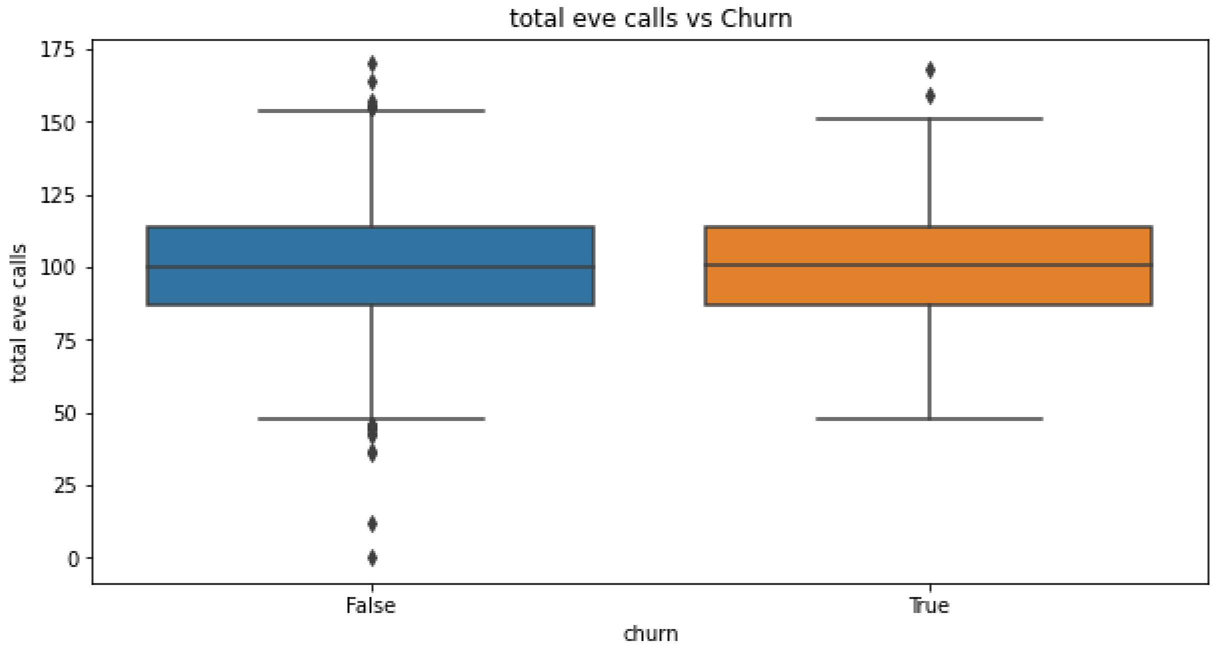

number vmail messages vs Churn
50
40
30
20
10
0
number vmail messages
False
True
churn

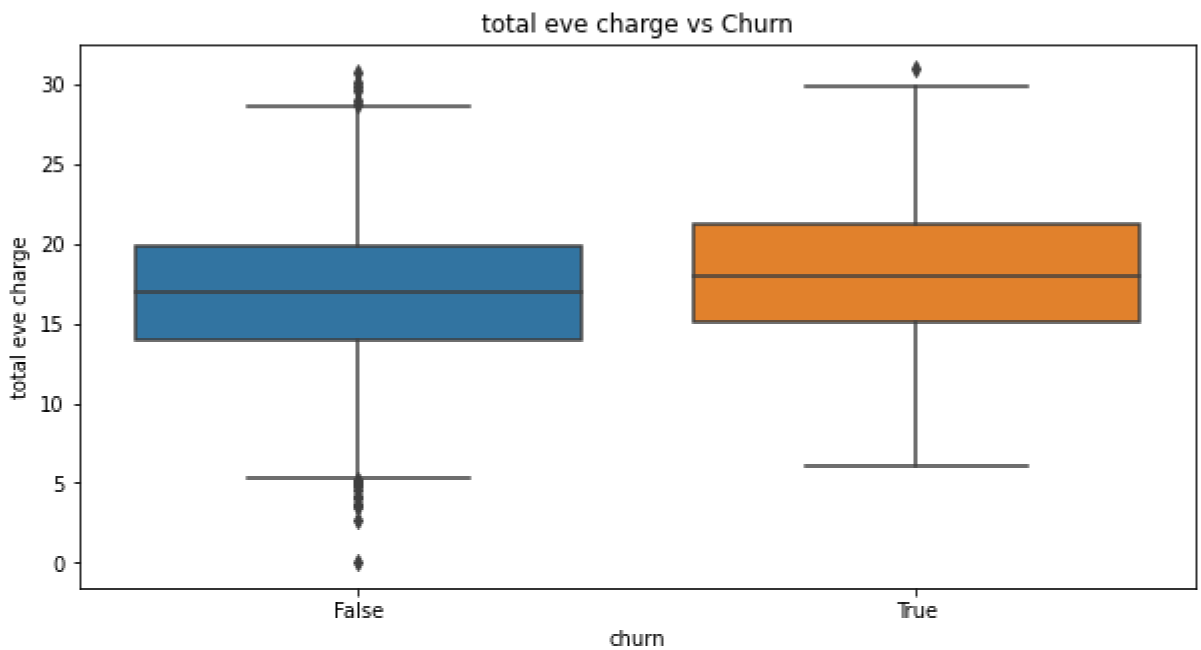

total day minutes vs Churn
total day minutes
churn
False
True
0
50
100
150
200
250
300
350

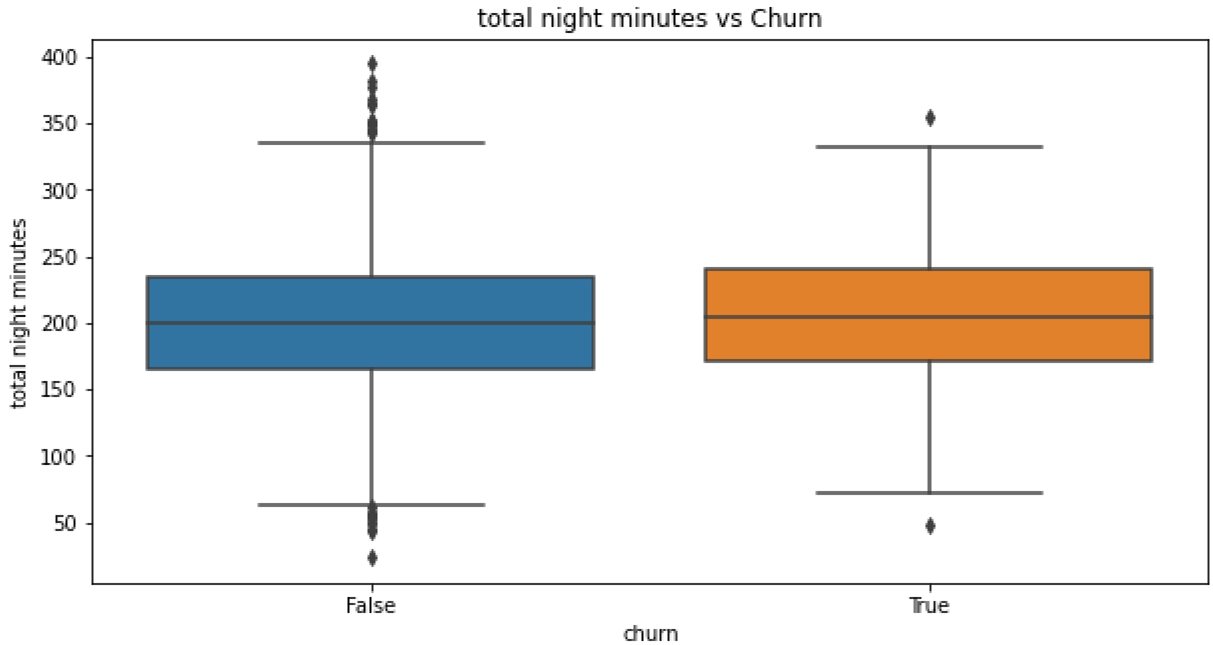

total day calls vs Churn
total day calls
churn
False
True
0
25
50
75
100
125
150

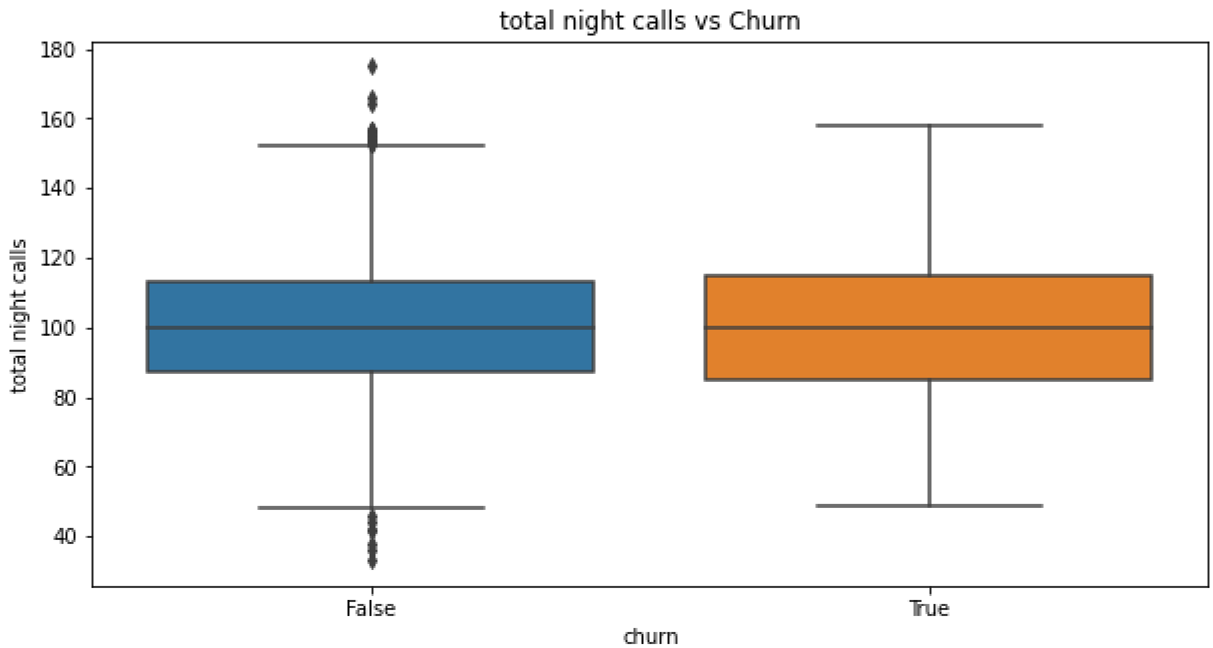

total day charge vs Churn
total day charge
0
10
20
30
40
50
60
False
True
churn

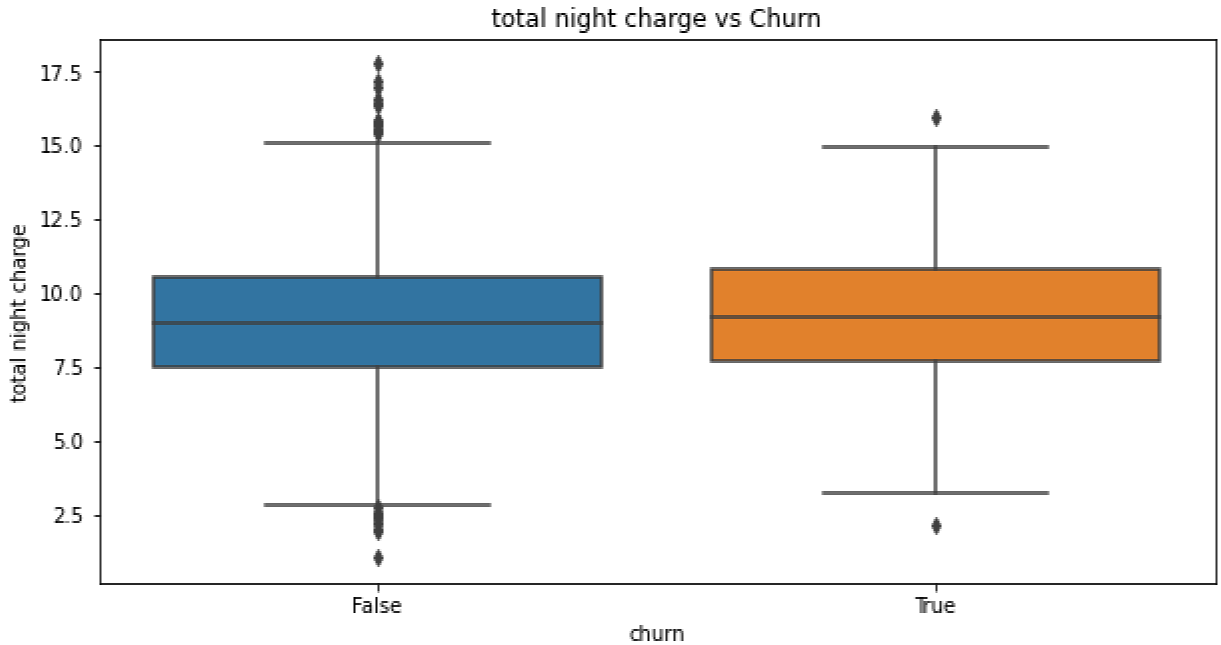

total eve minutes vs Churn
total eve minutes
0
50
100
150
200
250
300
350
False
True
churn

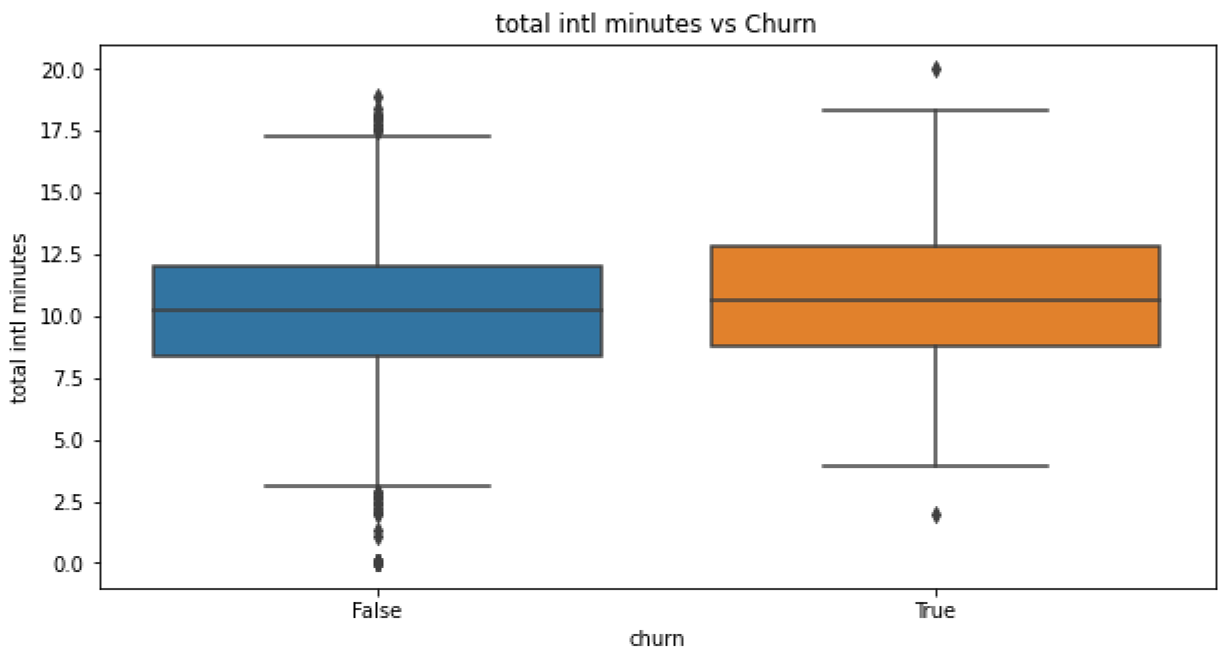

total eve calls vs Churn
175
150
125
100
75
50
25
0
total eve calls
False
True
churn

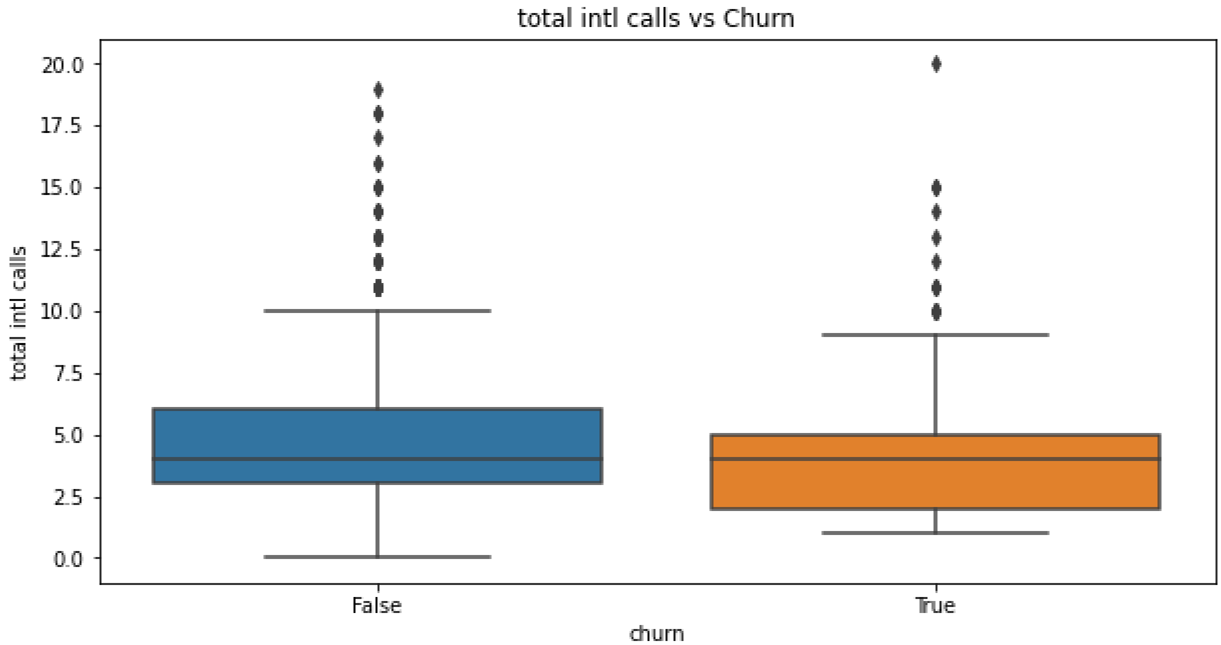

total eve charge vs Churn
30
25
20
15
10
5
0
total eve charge
False
True
churn

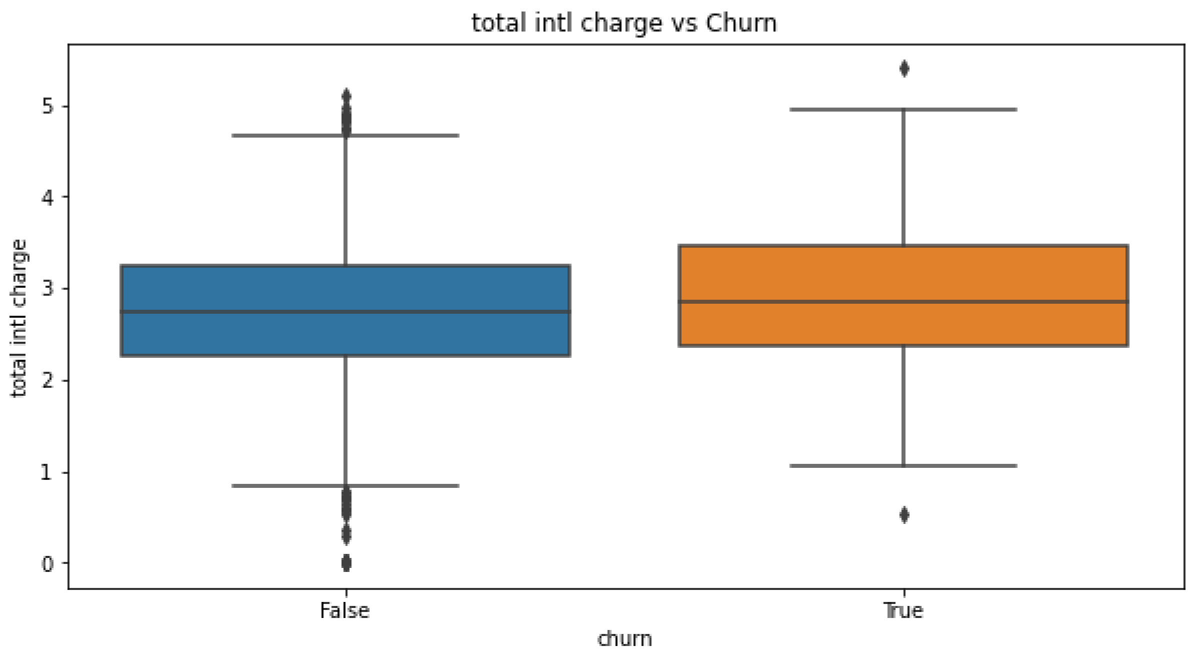

total night minutes vs Churn
400
350
300
250
200
150
100
50
total night minutes
False
True
churn

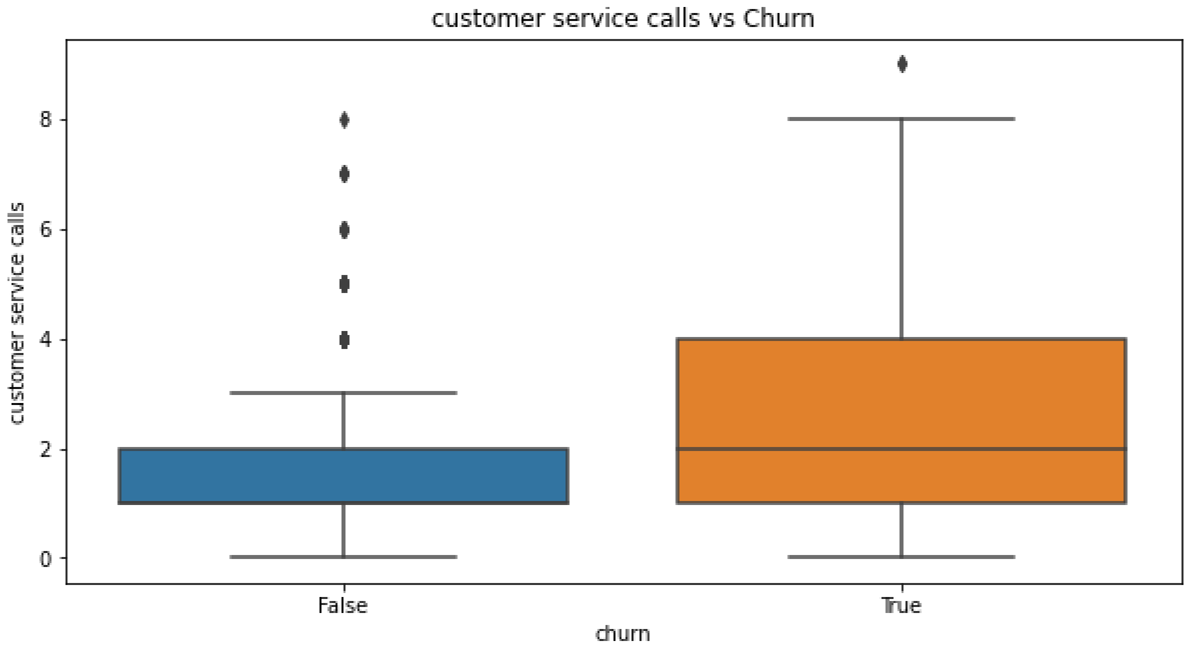

total night calls vs Churn
180
160
140
120
100
80
60
40
total night calls
False
True
churn


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


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


total intl calls vs Churn
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
total intl calls
False
True
churn

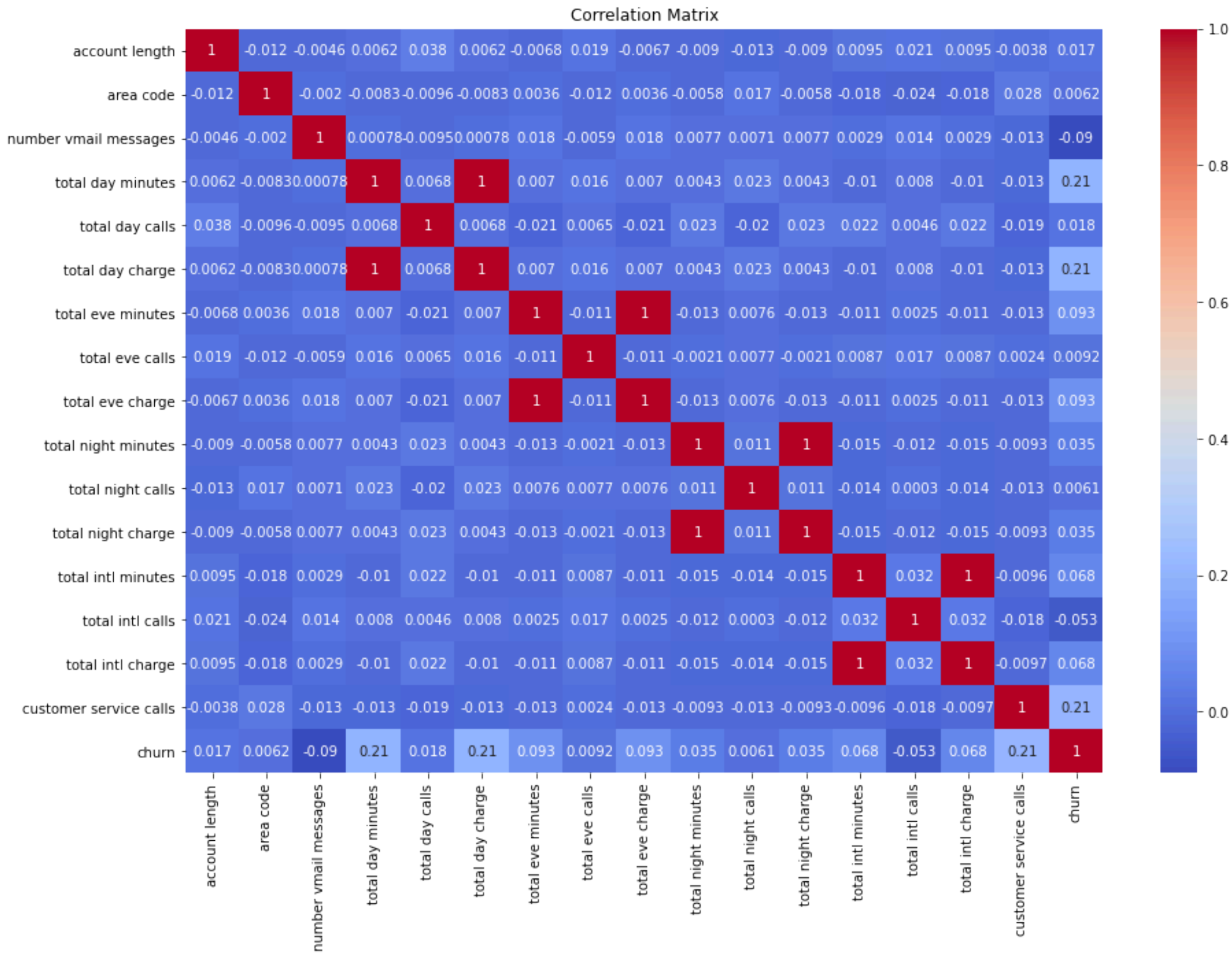

total intl charge vs Churn
5
4
3
2
1
0
total intl charge
False
True
churn




# 3.4: MultiVariate Analysis

## 3.4.1: Correlation Matrix

In [ ]:
```
# Correlation matrix
correlation_matrix = data.corr()
plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```




From this correlation heatmap we can deduce the following:

- Positive correlation between customer service calls and churn(0.21) means customers who call the service are slightly more likely to churn
- Positive correlation between total day minutes and churn(0.21) means customers who talk more during the day are also slightly likely to churn
- Positive correlation between total day charge and churn(0.21) means rates from the service provider might slightly influence the customer to churn as they continue usage. Let us also remeber that correlation does not entirely equal causation as there might be other factors at play.

## 3.5: Encoding Categorical variables

In [ ]:
```
# Identify binary categorical features for Label Encoding
binary_features = ['voice mail plan', 'international plan']

# Identify multi-class categorical features for One-Hot Encoding
multi_class_features = [feature for feature in categorical_features if feature not in b

# Apply Label Encoding to binary features
label_encoders = {}
for feature in binary_features:
    le = LabelEncoder()
    data[feature] = le.fit_transform(data[feature])
    label_encoders[feature] = le
```

```
# Apply One-Hot Encoding to multi-class features
data = pd.get_dummies(data, columns=multi_class_features, drop_first=True)

# Verify the transformations
print(data.head())
```

```
   account length  area code phone number  international plan  \
0             128        415     382-4657                   0
1             107        415     371-7191                   0
2             137        415     358-1921                   0
3              84        408     375-9999                   1
4              75        415     330-6626                   1

   voice mail plan  number vmail messages  total day minutes  total day calls  \
0                1                     25              265.1              110
1                1                     26              161.6              123
2                0                      0              243.4              114
3                0                      0              299.4               71
4                0                      0              166.7              113

   total day charge  total eve minutes  ...  state_SD  state_TN  state_TX  \
0             45.07              197.4  ...         0         0         0
1             27.47              195.5  ...         0         0         0
2             41.38              121.2  ...         0         0         0
3             50.90               61.9  ...         0         0         0
4             28.34              148.3  ...         0         0         0

   state_UT  state_VA  state_VT  state_WA  state_WI  state_WV  state_WY
0         0         0         0         0         0         0         0
1         0         0         0         0         0         0         0
2         0         0         0         0         0         0         0
3         0         0         0         0         0         0         0
4         0         0         0         0         0         0         0

[5 rows x 70 columns]
```

## 3.6: Train-Test Split Test

Train our data from the data set to be used.

In [ ]:
```
# Define features and target
X = data.drop(columns=['churn','phone number'])
y = data['churn']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

### 3.7: Scaling our Data

To bring out improved metrics in our model building and prevent leakage as well

In [ ]:
```
# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the test data
X_test_scaled = scaler.transform(X_test)
```

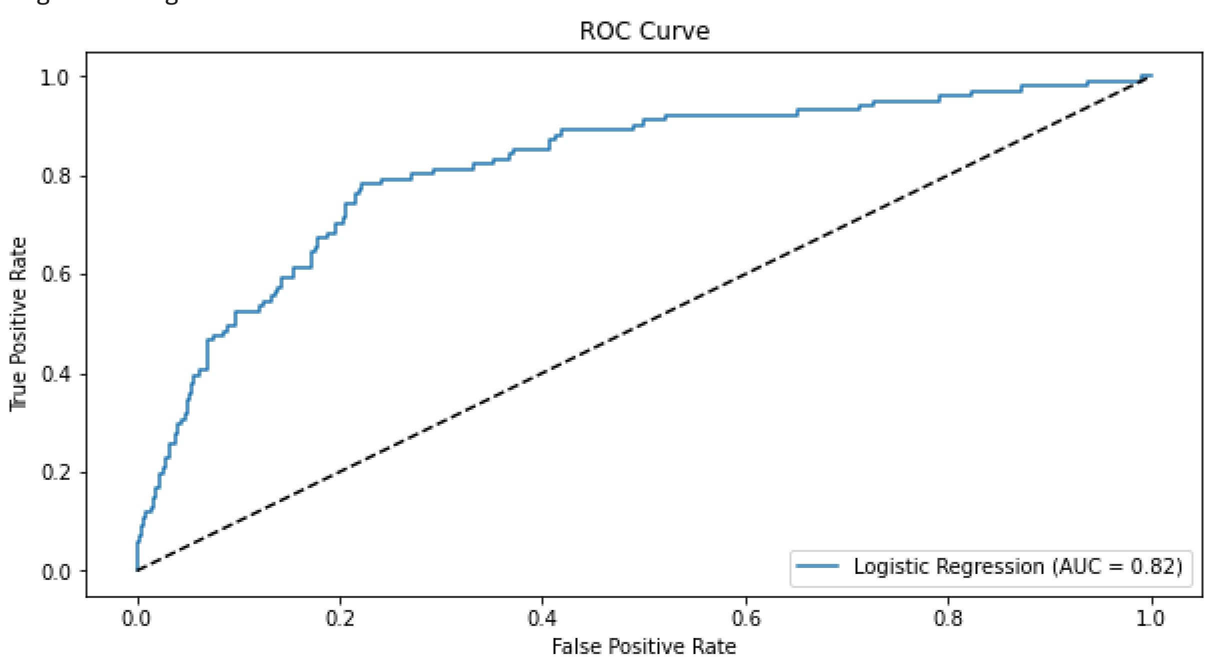
# 4: Model Building

## 4.1: Model 1 (Baseline Model-Logistic Regression)

In [ ]:
```
Model_1 = LogisticRegression(max_iter=1000)
Model_1.fit(X_train_scaled, y_train)

y_pred = Model_1.predict(X_test_scaled)
y_pred_prob = Model_1.predict_proba(X_test_scaled)[:, 1]
```

In [ ]:
```
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred))
print("Logistic Regression Classification Report:\n", classification_report(y_test, y_p
print("Logistic Regression ROC AUC Score:", roc_auc_score(y_test, y_pred_prob))

# Plot ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
plt.figure(figsize=(10, 5))
plt.plot(fpr, tpr, label='Logistic Regression (AUC = %0.2f)' % roc_auc_score(y_test, y_
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

```
Logistic Regression Accuracy: 0.8575712143928036
Logistic Regression Classification Report:
               precision    recall  f1-score   support

       False       0.87      0.97      0.92       566
        True       0.58      0.21      0.31       101

    accuracy                           0.86       667
   macro avg       0.73      0.59      0.61       667
weighted avg       0.83      0.86      0.83       667

Logistic Regression ROC AUC Score: 0.8164293461148235
```

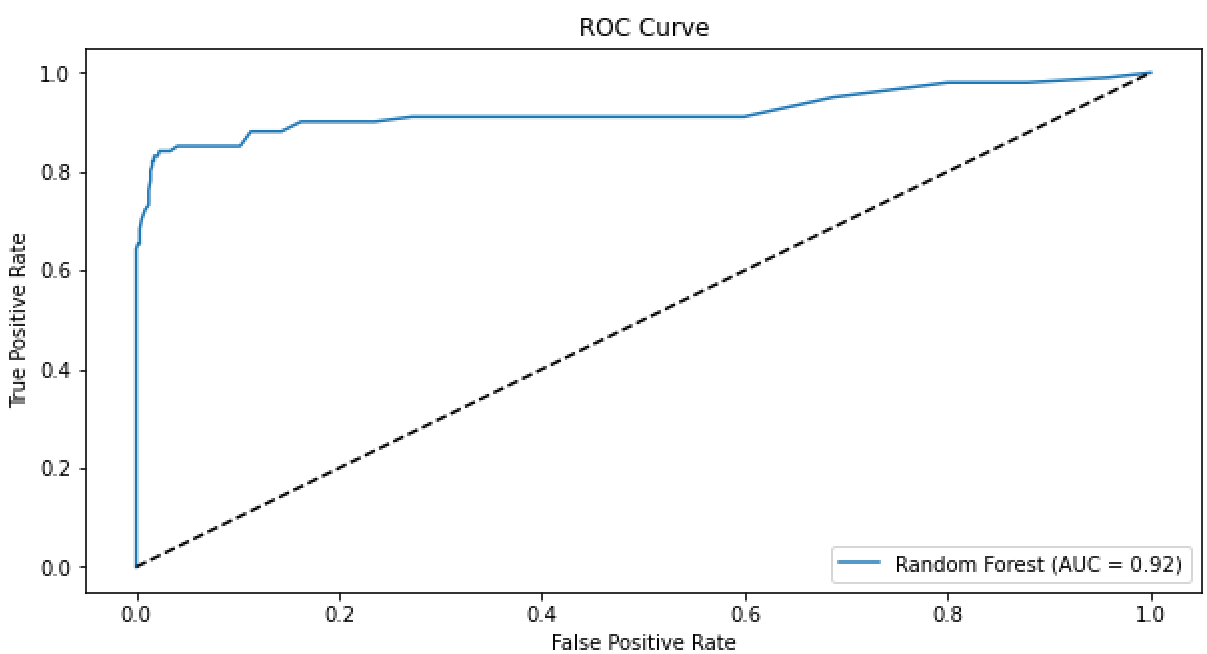


## 4.2: Model 2 (Random Forest Regression Model)

In [ ]:

```
# Train the random forest model
Model_2 = RandomForestClassifier(n_estimators=100, random_state=42)
Model_2.fit(X_train_scaled, y_train)

# Make predictions
y_pred_rf = Model_2.predict(X_test_scaled)
y_pred_prob_rf = Model_2.predict_proba(X_test_scaled)[:, 1]

# Evaluate the model
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Random Forest Classification Report:\n", classification_report(y_test, y_pred_rf
print("Random Forest ROC AUC Score:", roc_auc_score(y_test, y_pred_prob_rf))

# Plot ROC curve
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_prob_rf)
plt.figure(figsize=(10, 5))
plt.plot(fpr_rf, tpr_rf, label='Random Forest (AUC = %0.2f)' % roc_auc_score(y_test, y_
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

```
Random Forest Accuracy: 0.9400299850074962
Random Forest Classification Report:
               precision    recall  f1-score   support

       False       0.93      1.00      0.97       566
        True       1.00      0.60      0.75       101

    accuracy                           0.94       667
   macro avg       0.97      0.80      0.86       667
weighted avg       0.94      0.94      0.93       667

Random Forest ROC AUC Score: 0.9224976384564252
```

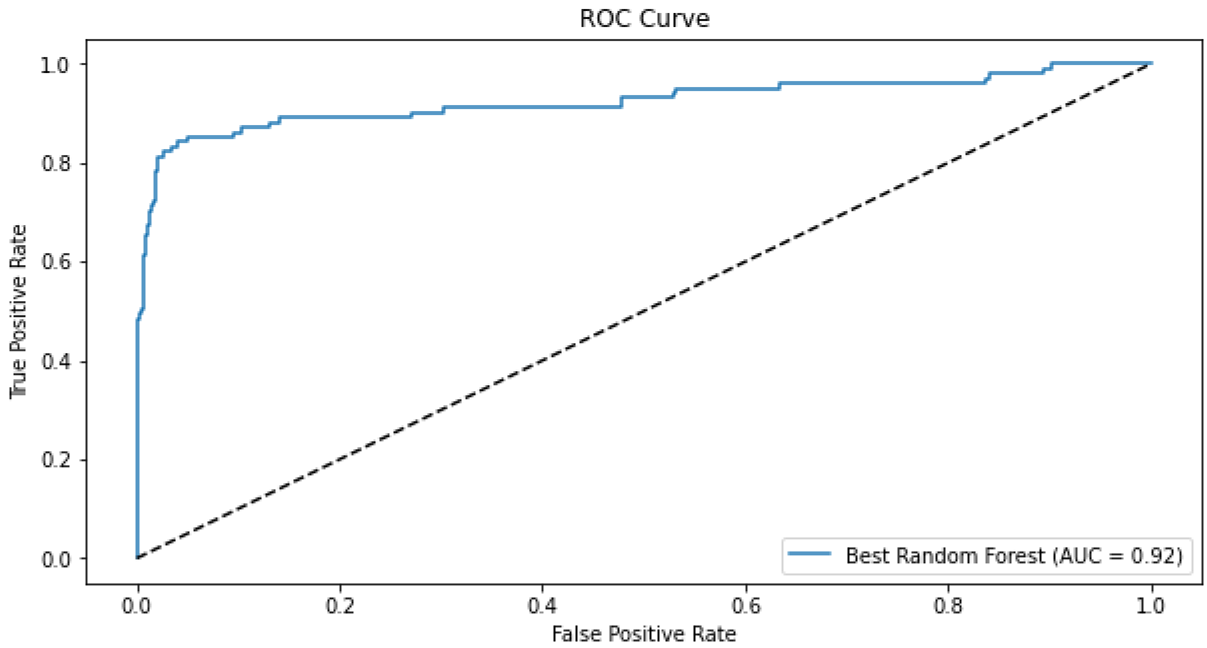


# 4.3: Model 3 (Random Forest Model with HyperParameters Tuning)

In this Particular instance,Grid Search

In [ ]:
```
# Define parameter grid for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42), param_gri

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

# Get the best model from grid search
best_rf_model = grid_search.best_estimator_

# Make predictions
y_pred_best_rf = best_rf_model.predict(X_test_scaled)
y_pred_prob_best_rf = best_rf_model.predict_proba(X_test_scaled)[:, 1]

# Evaluate the best model
print("Best Random Forest Accuracy:", accuracy_score(y_test, y_pred_best_rf))
print("Best Random Forest Classification Report:\n", classification_report(y_test, y_pr
print("Best Random Forest ROC AUC Score:", roc_auc_score(y_test, y_pred_prob_best_rf))

# Plot ROC curve
fpr_best_rf, tpr_best_rf, _ = roc_curve(y_test, y_pred_prob_best_rf)
plt.figure(figsize=(10, 5))
plt.plot(fpr_best_rf, tpr_best_rf, label='Best Random Forest (AUC = %0.2f)' % roc_auc_s
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   26.1s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 324 out of 324 | elapsed:  2.9min finished
Best Random Forest Accuracy: 0.9370314842578711
Best Random Forest Classification Report:
               precision    recall  f1-score   support

       False       0.94      0.99      0.96       566
        True       0.93      0.63      0.75       101

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

Best Random Forest ROC AUC Score: 0.9235384669208971
```

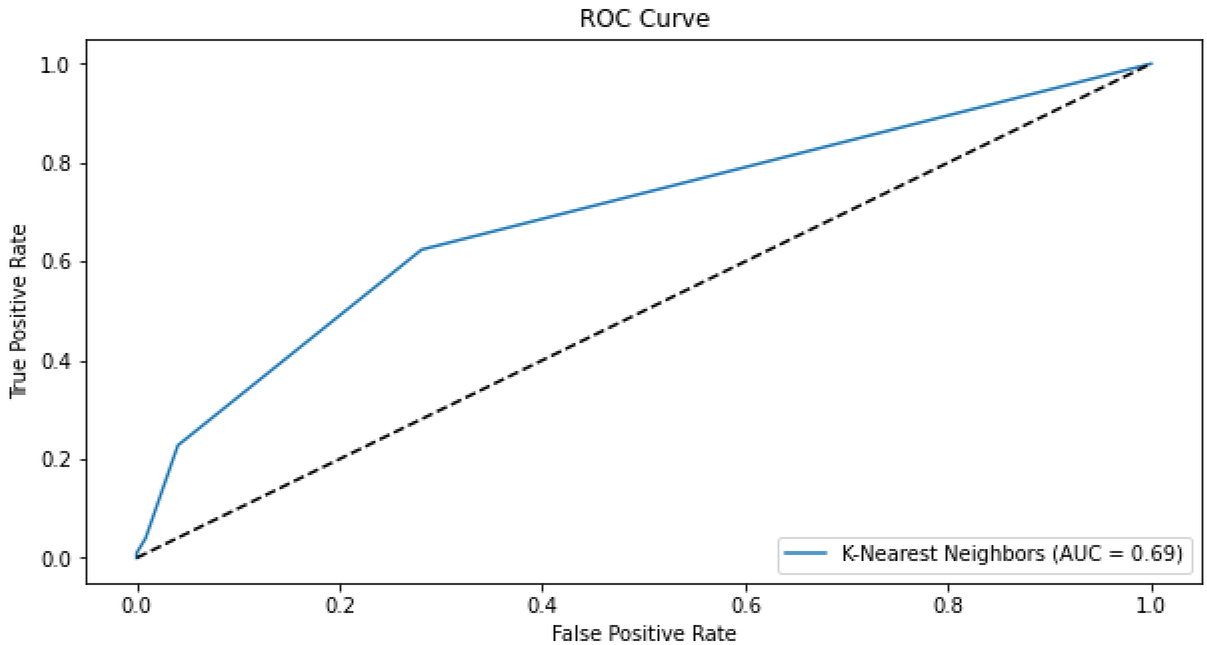



## 4.4: Model 4 K-Nearest Neighbors Model

In [ ]:

```
# Train the K-Nearest Neighbors model
Model_4 = KNeighborsClassifier(n_neighbors=5)
Model_4.fit(X_train_scaled, y_train)

# Make predictions
y_pred_knn = Model_4.predict(X_test_scaled)
y_pred_prob_knn = Model_4.predict_proba(X_test_scaled)[:, 1]

# Evaluate the model
print("K-Nearest Neighbors Accuracy:", accuracy_score(y_test, y_pred_knn))
print("K-Nearest Neighbors Classification Report:\n", classification_report(y_test, y_p
print("K-Nearest Neighbors ROC AUC Score:", roc_auc_score(y_test, y_pred_prob_knn))

# Plot ROC curve
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_pred_prob_knn)
plt.figure(figsize=(10, 5))
plt.plot(fpr_knn, tpr_knn, label='K-Nearest Neighbors (AUC = %0.2f)' % roc_auc_score(y_
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

```
K-Nearest Neighbors Accuracy: 0.8470764617691154
K-Nearest Neighbors Classification Report:
               precision    recall  f1-score   support

       False       0.85      0.99      0.92       566
        True       0.44      0.04      0.07       101

    accuracy                           0.85       667
   macro avg       0.65      0.52      0.49       667
weighted avg       0.79      0.85      0.79       667

K-Nearest Neighbors ROC AUC Score: 0.6905765664905713
```




## 5: Evaluation

### 5.1: Model 1(Baseline Model)

From this Logistic Regression model, we can deduce;

- Imbalanced Dataset: The dataset is imbalanced, with 566 non-churning and 101 churning customers. This imbalance affects the performance metrics, particularly for the minority class (churning customers).
- High Precision but Low Recall for Churn: The model has a reasonable precision for predicting churn (58%), meaning that when it predicts churn, it is correct more than half the time. However, it struggles with recall (21%), meaning it misses a significant number of actual churning customers.
- Strong Performance for Non-Churn: The model performs very well for non-churning customers, with high precision (87%) and recall (97%), indicating it is very reliable in identifying customers who will not churn.
- Weighted Metrics: The weighted averages suggest the model has good overall performance, but the poor recall for churn indicates a need for improvement if predicting churn accurately is critical

### 5.2: Model 2 (Random Forest Model)

From this Random forest Classification model, we can deduce:

- The ROC AUC score of 0.92 indicates a very good ability to distinguish between churning and non-churning customers.
- Improved Performance: The Random Forest model shows significantly improved performance over the logistic regression model, particularly in precision and F1-score for the True (churning) class.

- High Precision for Churn: The precision for the True class is perfect (1.00), meaning there are no false positives. Every customer predicted to churn actually churns.
- Balanced Recall: While the recall for the True class (0.60) is not perfect, it is much better than in the logistic regression model, indicating the Random Forest model is better at identifying actual churners.
- Overall Strong Metrics: The high accuracy, precision, recall, and F1-scores, along with the ROC AUC score, indicate that the Random Forest model performs very well across all key metrics.

### 5.3 Model 3(Tuned Random Forest Model)

For this Tuned Random Forest Model, we can deduce that:

- The ROC AUC score of 0.92 indicates a very good ability to differentiate between churning and non-churning customers.
- High Precision and Recall for Non-Churn: The model performs exceptionally well in predicting non-churning customers, with very high precision (0.94) and recall (0.99).
- Improved Performance for Churn: The precision for predicting churn is very high (0.93), meaning the model makes very few false positive errors. However, the recall (0.63) is still somewhat limited, indicating that some churning customers are not being identified.
- Balanced Metrics: The F1-scores and weighted averages suggest the model has a good balance of precision and recall for both classes, indicating robust overall performance.

### 5.4 Model 4(K-Nearest Neighbors Model)

For this KNN model, we can deduce that:

- High Recall for Non-Churn: The model performs very well in predicting non-churning customers, with high precision (0.85) and very high recall (0.99).
- Poor Performance for Churn: The model struggles significantly with predicting churning customers, with very low precision (0.44) and extremely low recall (0.04). This results in a very low F1-score (0.07) for the churn class.
- Imbalance Impact: The performance metrics indicate that the model is heavily biased towards the majority class (non-churning customers) and performs poorly on the minority class (churning customers).
- Low ROC AUC: The ROC AUC score of 0.69 shows the model's limited effectiveness in distinguishing between churners and non-churners.

## 5.5 Model Recommendation

Putting into consideration all the perfomance metrics for the models above, we can conclude that our best fit Model for predicting whether a customer will churn or not is MODEL 3. It has the highest ROC AUC Score demonstrating excellent discrimination capaility with the highest F1 score indicating a balanced performance for precision and recall making it the preferred Model we recommend for use by SyriaTel.

# 6: RECOMMENDATION

So having analyzed the dataset provided for Syriatel customers, we recommend the following:

1. Target High-Risk Customers Focus retention efforts on customers identified as high-risk by the predictive model Recommendation: Implement a priority system for customer service where high-risk customers receive faster and more personalized support. This can be in the form of a dedicated hotline or a special customer service team.

2. Optimize Service Plans High total day charges are associated with higher churn rates. Review and optimize the pricing and features of your day-time call plans. Offer customized plans with lower day-time charges for high-usage customers. Consider promotional offers such as temporary discounts or bonus minutes.

3. Improve Customer Service Quality A high number of customer service calls correlate with higher churn. Maybe the customers are dissatisfied with the cutomer service Address the root causes of frequent customer service calls. Conduct training sessions for customer service representatives to enhance their problem-solving skills. Implement a feedback system to identify common issues and resolve them proactively.

4. Promote Retention Programs Customers without international plans and voice mail plans show different churn behaviors. Promote the voice mail plan more aggressively, especially to customers without it, as it might be linked to higher retention. Offer free trials or discounts on the voice mail plan to new customers or those who have recently had issues resolved through customer service and i guarantee this will help wholely with customer retention.

5. Enhance Customer Engagement Regularly engage with customers to gather feedback and address their concerns before they consider churning. Implement a customer engagement program where regular check-ins are scheduled. Use email or SMS surveys to gather feedback and act on it promptly.

6. Personalized Marketing Campaigns Certain demographics and usage patterns are linked to higher churn. Use the insights from the predictive model to tailor marketing campaigns. Develop personalized marketing campaigns targeting specific segments identified as high-risk. Offer them incentives like loyalty points, discounts, or free upgrades.

7. Continuous Monitoring and Adaptation Monitor the effectiveness of the implemented retention strategies and adapt them based on real-time data. Set up a dashboard to track key performance indicators (KPIs) such as churn rate, customer satisfaction, and the effectiveness of retention campaigns. Use this data to continuously refine your strategies.