Telecom Churn Prediction for SyriaTel: Identifying Patterns for Customer Retention¶

Overview

This project focuses on helping SyriaTel predict and reduce customer churn, where customers stop using their services. Churn prediction is crucial for businesses to prevent revenue loss.

Business Understanding

Introduction

This project centers on SyriaTel, a prominent telecommunications company, seeking to predict and reduce customer churn. Customer churn, defined as customers discontinuing services, poses a significant challenge to businesses. Our focus is to leverage data analysis and machine learning techniques to unveil patterns indicative of potential customer exits. By identifying these patterns, SyriaTel can proactively implement measures to retain customers and mitigate revenue loss. The project involves thorough data analysis, the development of predictive models, and the formulation of actionable recommendations to enhance customer retention strategies. The overarching goal is to augment SyriaTel's understanding of customer behaviors, ultimately contributing to more effective retention initiatives.

Main Objective

To apply classification modeling techniques to analyze customer churn data for Syria Tel, aiming to identify and quantify the influential factors contributing to customer churn.

Subjective Objectives

1.Explore the Data for Classification Explore the dataset to understand the relationships between different variables and the target variable (customer churn)

2.Develop accurate machine learning models for predicting customer churn based on historical data.

3.Identify and analyze patterns and trends in customer behavior that contribute to churn.

4.Evaluate model performance and fine-tune hyperparameters for optimal predictive accuracy.

5.Provide actionable insights and recommendations based on churn predictions to aid SyriaTel in implementing effective customer retention strategies.

Data Understanding

In the data understanding phase, a thorough examination of the Telecom Churn dataset was conducted, focusing on customer-related features and the target variable, churn. Key activities included identifying data types, addressing missing values, and generating descriptive statistics for numeric variables. Exploratory Data Analysis (EDA) provided valuable visual insights, especially in understanding data distributions and relationships. Categorical variables were examined for potential one-hot encoding, and target variable exploration shed light on the distribution of churn instances. Correlation analysis contributed to identifying potential multicollinearity. Integration of domain knowledge enhanced the understanding of variable significance. The comprehensive assessment of data quality laid a solid foundation for subsequent data preprocessing and model development stages.

```
import pandas as pd
from pandas_profiling import ProfileReport
import numpy as np
%matplotlib inline
import warnings
from sklearn.model_selection import train_test_split #for splitting our data into training and testing sets
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
from xgboost import XGBClassifier
from sklearn.metrics import mean_squared_error, r2_score #Evaluate model performance
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE # appling SMOTE to training data
import statsmodels.api as sm
```

```
df = pd.read_csv("bigml_59c28831336c6604c800002a.csv")
df
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	tota 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
...	...	...	...	...	...	...	...	...	...	
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.5
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.2
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.7
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.3
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.8

3333 rows × 21 columns

```
# Defining a function to get information on the dataset

def dataframe_preview(df):
    #To get the shape of the dataframe
    print("The shape of the dataframe:")
    print(df.shape)
    print('\n')
    #To get the info of the dataframe
    print("The data in the dataframe:")
    print(df.info())
    print('\n')

dataframe_preview(df)
```

```
The shape of the dataframe:
(3333, 21)
```

```
The data in the dataframe:
<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
```

Data Explotation

```
#  Creating a class for exploring and analyzing Pandas DataFrames.
class DataExplorer:
    def __init__(self, data):
        """
        Initialize the DataExplorer class with a dataset.
        :param data: Pandas DataFrame
        """
        self.data = data

    def check_head(self, rows=5):
        """
        Display the first few rows of the dataset.
        :return: Pandas DataFrame showing the head of the dataset
        """
        return self.data.head(rows)

    def check_info(self):
        """
        Display information about the dataset (data types, memory usage, etc.).
        """
        return self.data.info()

    def check_describe(self):
        """
        Generate descriptive statistics of the dataset (count, mean, etc.).
        :return: Pandas DataFrame with descriptive statistics
        """
        return self.data.describe()
```

```
explorer = DataExplorer(df)

# Check the head of the dataset
print("Head of the dataset:")
print(explorer.check_head())

# Check dataset info
print("\nDataset info:")
explorer.check_info()

# Check dataset description
print("\nDataset statistics:")
print(explorer.check_describe())
```

```
Head of the dataset:
  state  account length  area code phone number 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

  voice mail plan  number vmail messages  total day minutes  total day calls  \
0             yes                     25              265.1              110
1             yes                     26              161.6              123
2              no                      0              243.4              114
3              no                      0              299.4               71
4              no                      0              166.7              113

   total day charge  ...  total eve calls  total eve charge  \
0             45.07  ...               99             16.78
1             27.47  ...              103             16.62
2             41.38  ...              110             10.30
3             50.90  ...               88              5.26
4             28.34  ...              122             12.61

   total night minutes  total night calls  total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   total intl minutes  total intl calls  total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   customer service calls  churn
0                       1  False
1                       1  False
2                       0  False
3                       2  False
4                       3  False

[5 rows x 21 columns]

Dataset 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
```

```
# statistics summary of Categorical columns
df.describe(include='O')
```

	state	phone number	international plan	voice mail plan
count	3333	3333	3333	3333

```
# use data profiling to preview our data
profile = ProfileReport(df)
profile
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	21
Number of observations	3333
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	524.2 KiB
Average record size in memory	161.0 B

Variable types

Text	2
Numeric	15
Categorical	1
Boolean	3

Alerts

`number vmail messages` is highly overall correlated with `voice mail plan` **High correlation**

Data Cleaning

```
# Check for null values
pd.DataFrame(df.isna().sum()).T
```

	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	0	0	0	0	0	0	0	0	0	0

1 rows × 21 columns

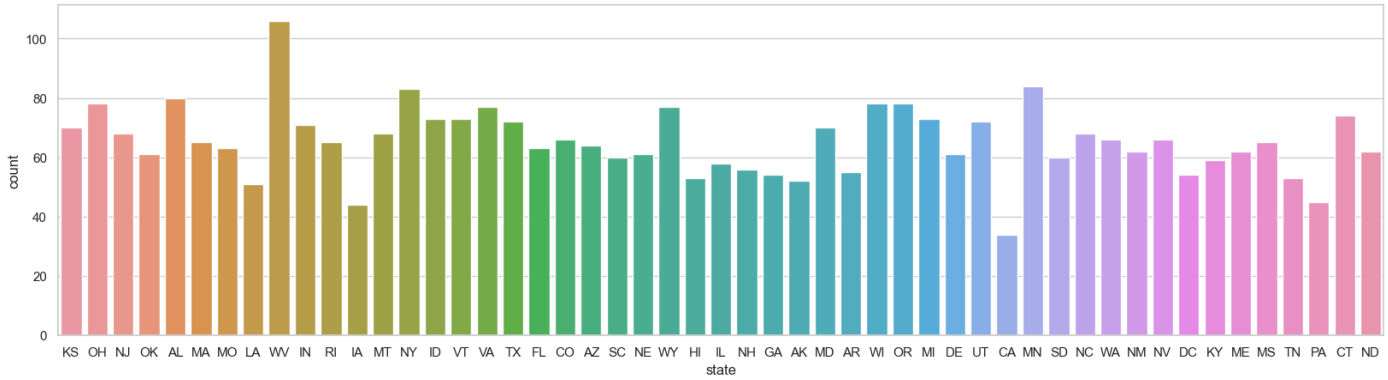
```
# Drop unnecessary columns
df = df.drop(['phone number'], axis=1)
```

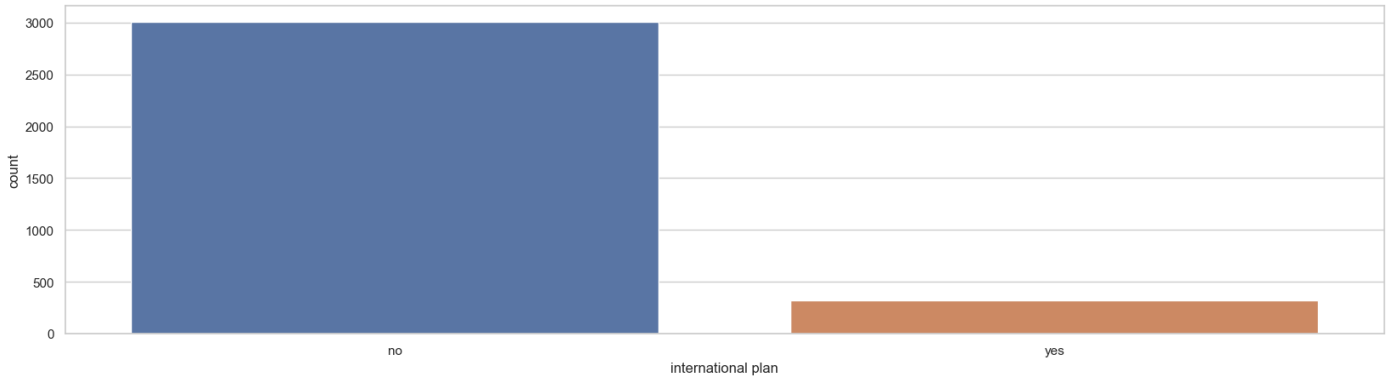
Explarotary Analysis

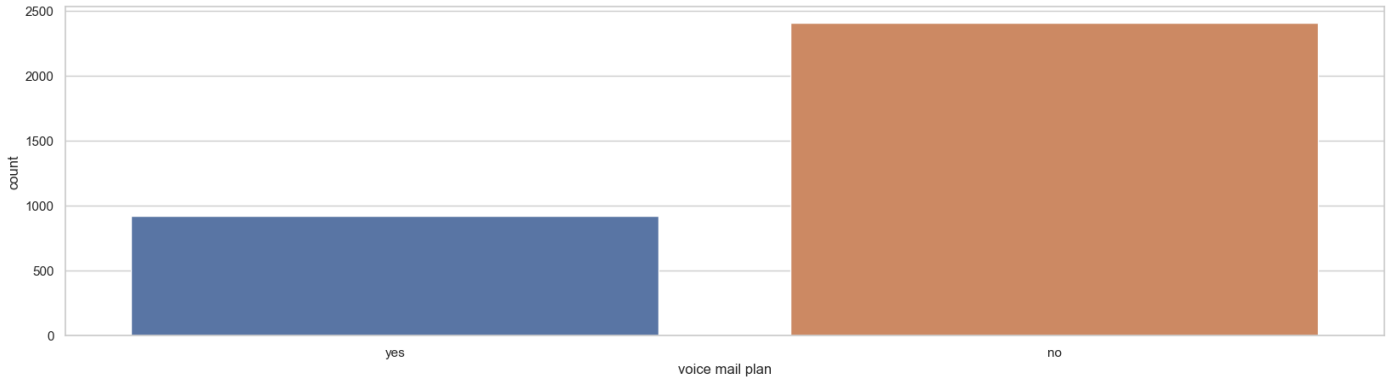
```
# For Categorical variables
categorical = [ cat for cat in df.columns if df[cat].dtypes=='O']
print('List of categorical variables {}'.format(categorical))
```

```
List of categorical variables ['state', 'international plan', 'voice mail plan']
```

```
# plotting categorical variables
for feature in categorical:
    sns.set(style = 'whitegrid')
    plt.figure(figsize=(20,5))
    total = len(df)
    ax = sns.countplot(x = df[feature], data = df)
    plt.show()
```

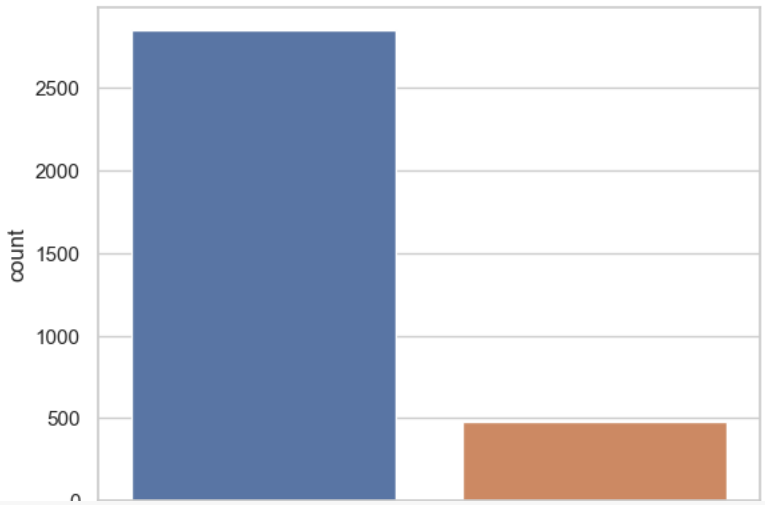







```
# visualize distribution of churn
print(df['churn'].value_counts())
sns.countplot(data=df,x='churn')
plt.show()
```

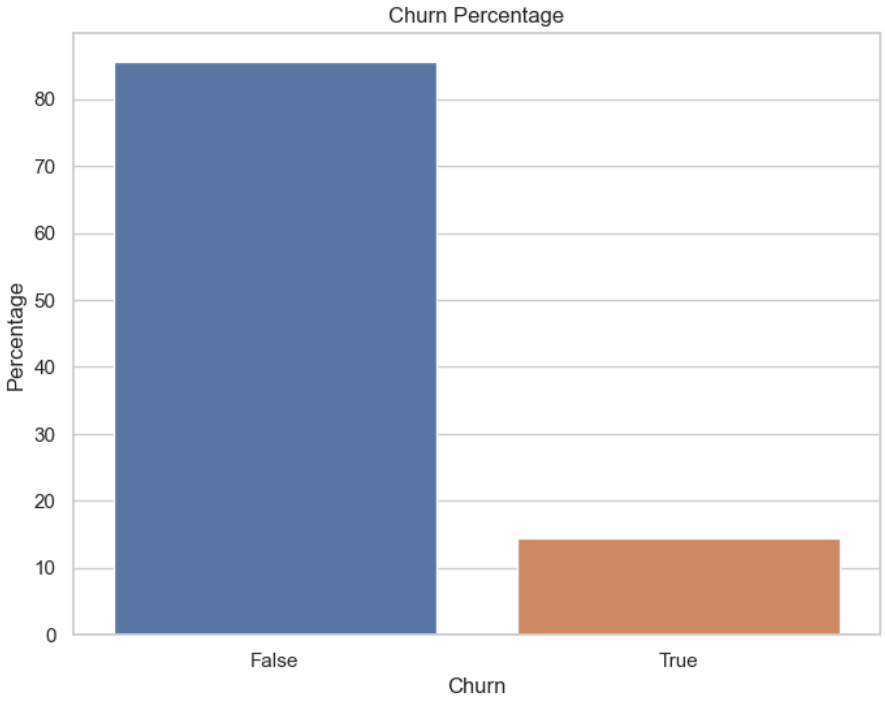
```
churn
False    2850
True      483
Name: count, dtype: int64
```




```
# Plotting the percentage of customer churn
churn_perc = pd.DataFrame(df['churn'].value_counts(normalize=True))
churn_perc
```

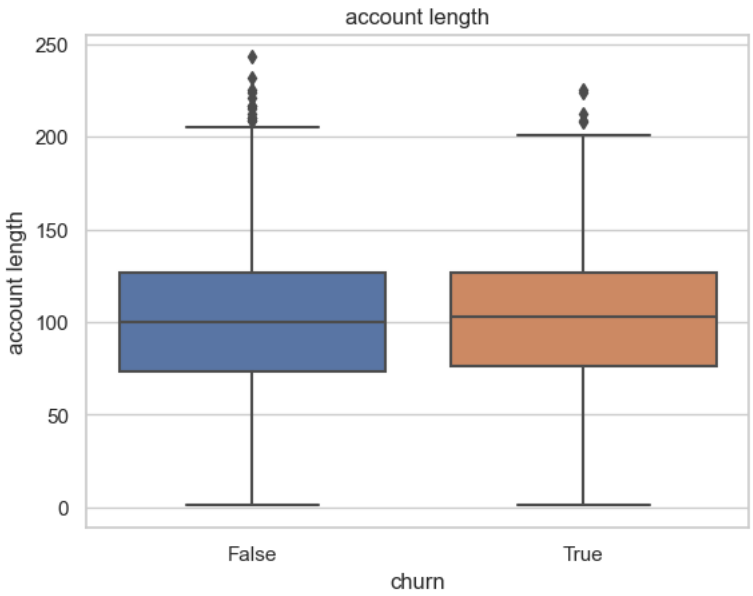
churn	proportion
False	0.855086
True	0.144914

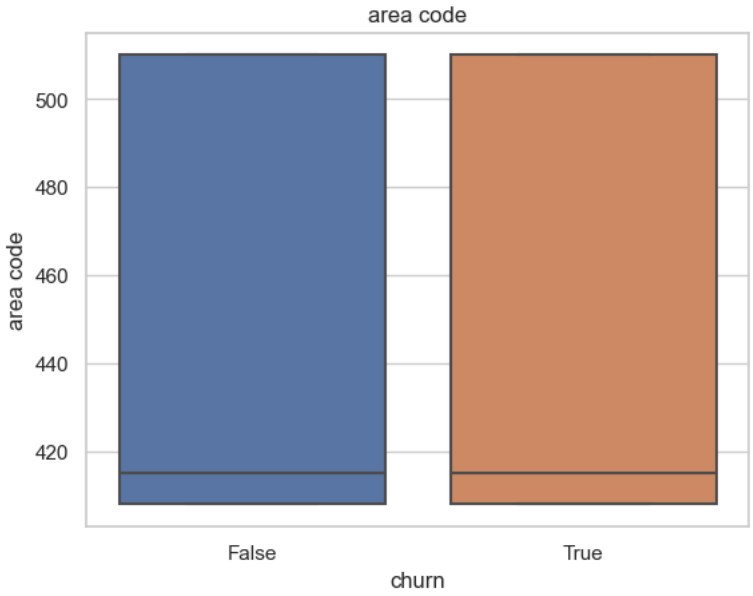
```
churn_perc = df.groupby('churn').size() / len(df) * 100
churn_perc = churn_perc.reset_index(name='percentage')
plt.figure(figsize=(8, 6))
sns.barplot(x='churn', y='percentage', data=churn_perc)
plt.xlabel('Churn')
plt.ylabel('Percentage')
plt.title('Churn Percentage')
plt.show()
```

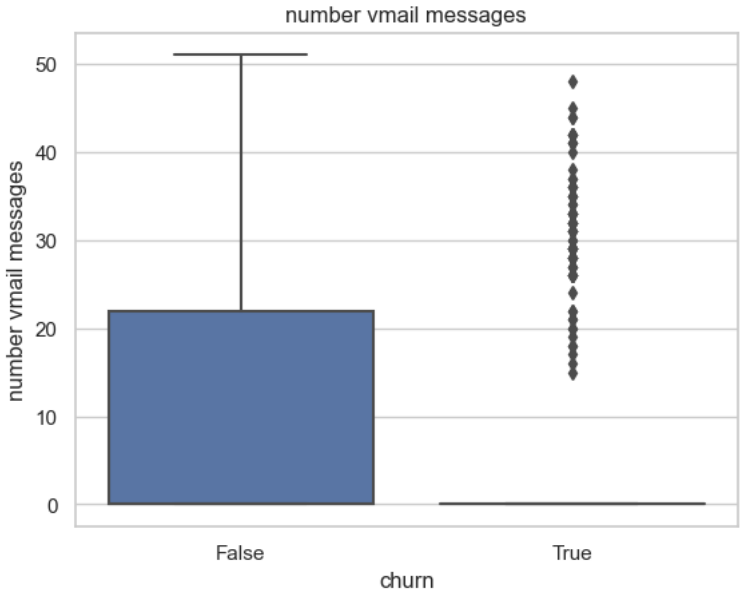


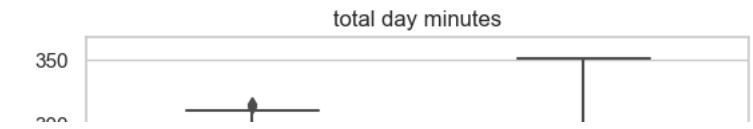
Based on the visualization, it appears that the majority of customers, accounting for 85%, have been retained, while approximately 15% are inclined towards churning

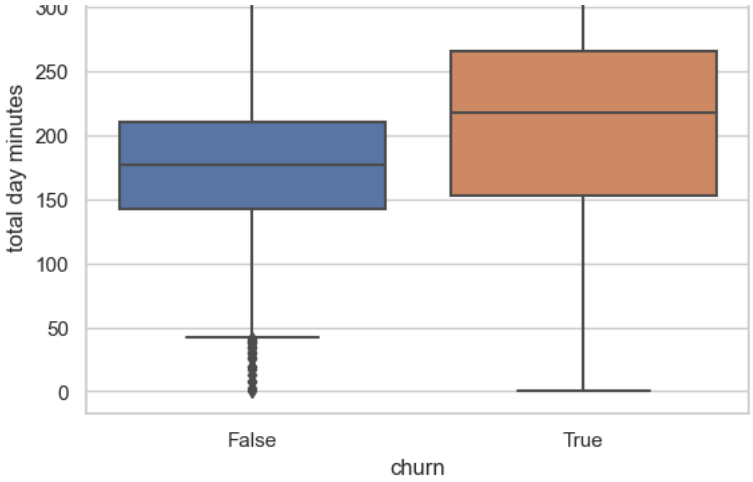
```
# Checking Outliers
numerical_variables = [
    'account length', 'area code', 'number vmail messages', 'total day minutes',
    'total day calls', 'total day charge', 'total eve minutes', 'total eve calls',
    'total eve charge', 'total night minutes', 'total night calls', 'total night charge',
    'total intl minutes', 'total intl calls', 'total intl charge', 'customer service calls'
]
for feature in numerical_variables:
    if feature != 'churn':
        sns.boxplot(x ='churn', y = feature, data = df)
        plt.title(feature)
        plt.show()
```

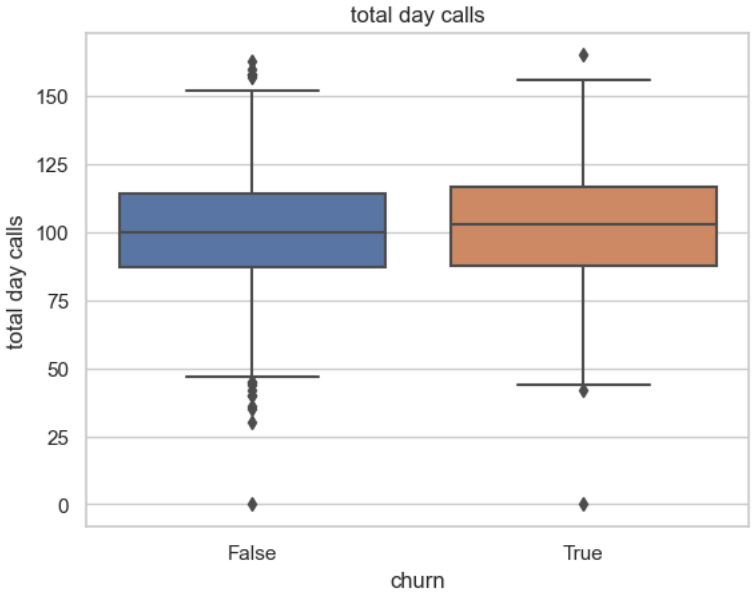

account length
250
200
150
100
50
0
account length
False
True
churn

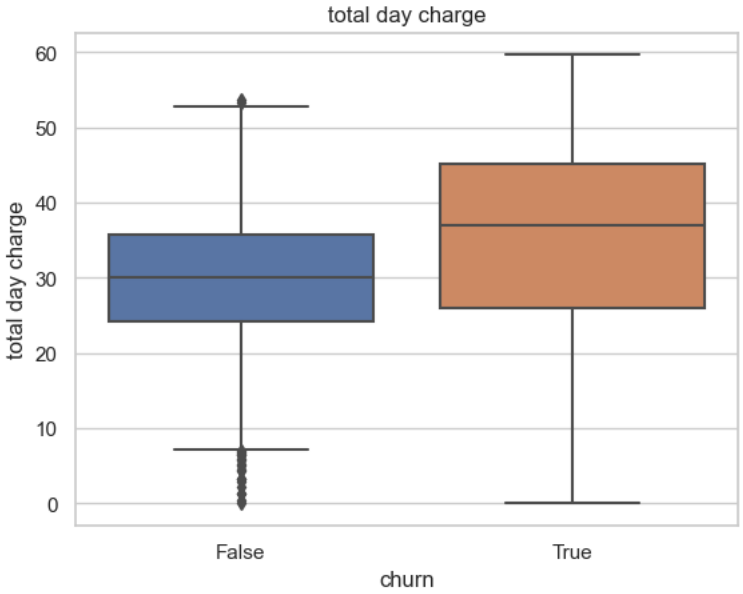

area code
500
480
460
440
420
area code
False
True
churn

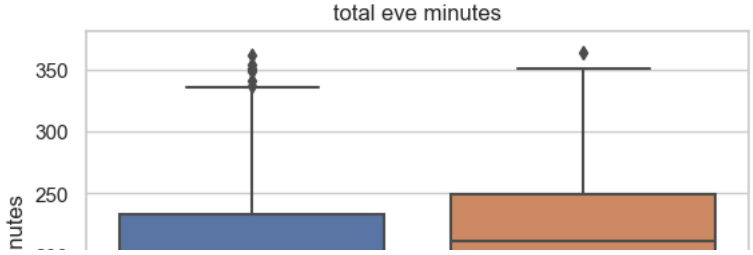

number vmail messages
50
40
30
20
10
0
number vmail messages
False
True
churn

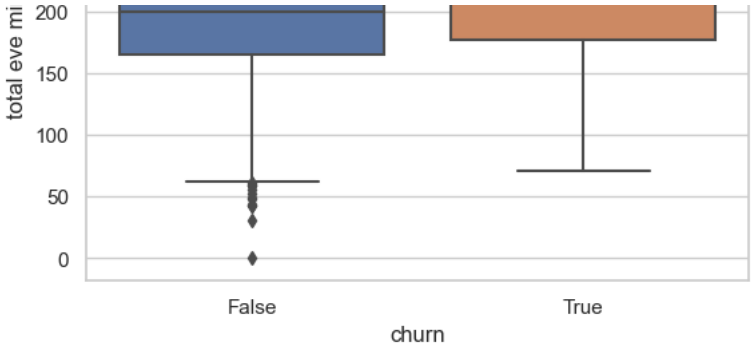

total day minutes
350

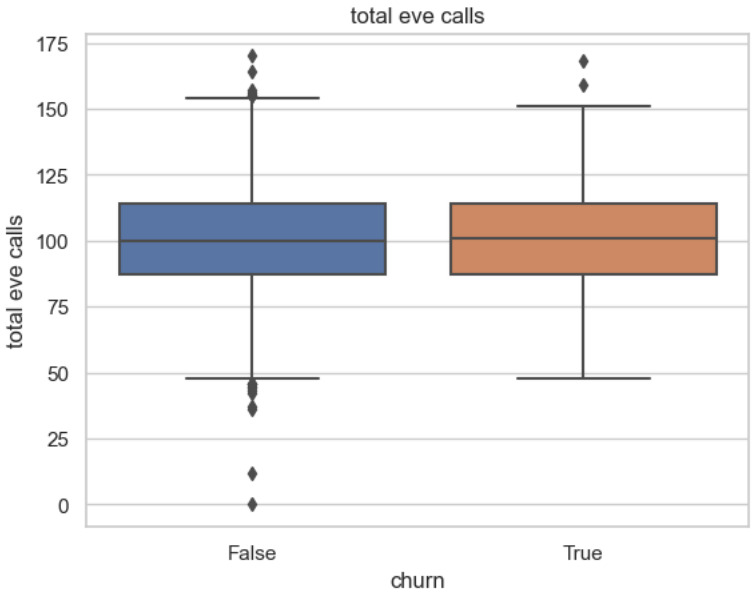

total day minutes
300
250
200
150
100
50
0
False
True
churn

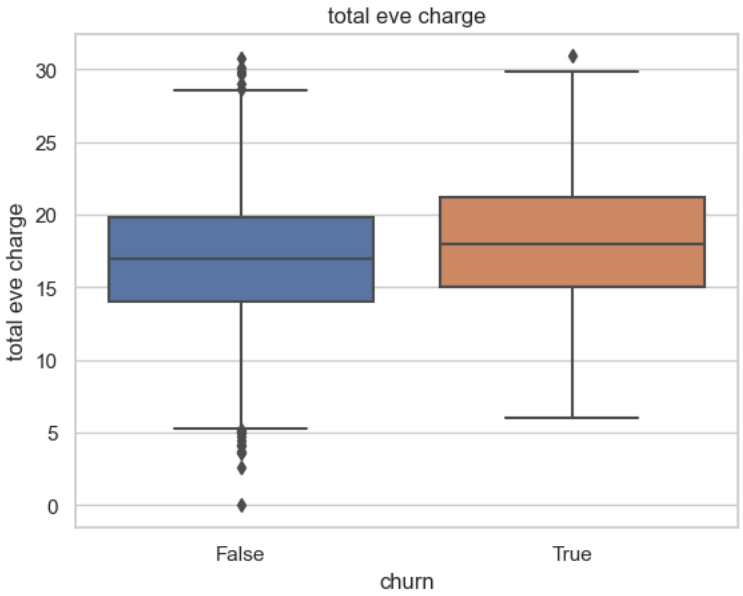

total day calls
total day calls
150
125
100
75
50
25
0
False
True
churn

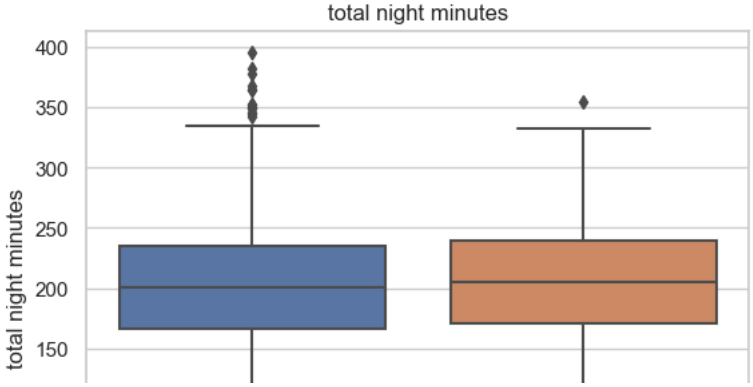

total day charge
total day charge
60
50
40
30
20
10
0
False
True
churn

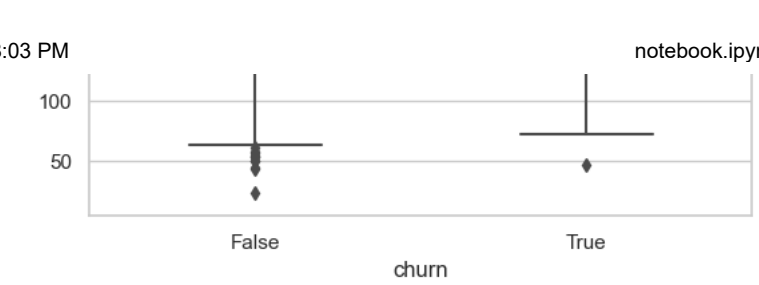

total eve minutes
350
300
250

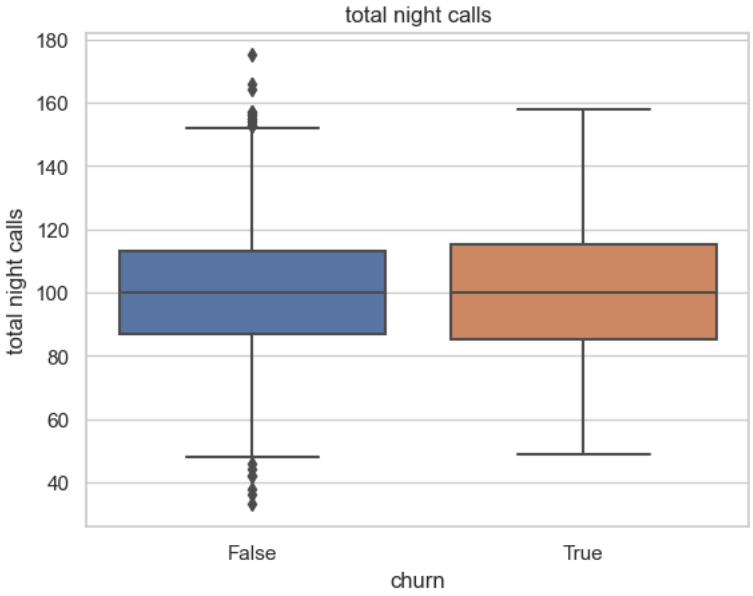

total eve mi
200
150
100
50
0
False
True
churn

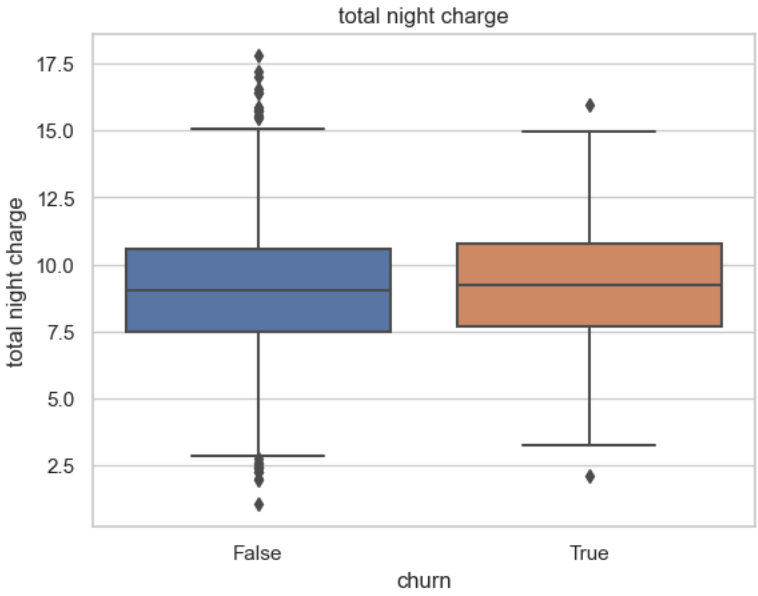

total eve calls
total eve calls
175
150
125
100
75
50
25
0
False
True
churn

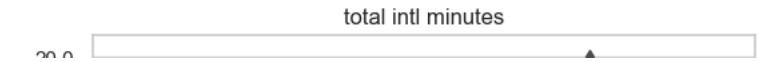

total eve charge
total eve charge
30
25
20
15
10
5
0
False
True
churn


total night minutes
total night minutes
400
350
300
250
200
150


100
50
False
True
churn

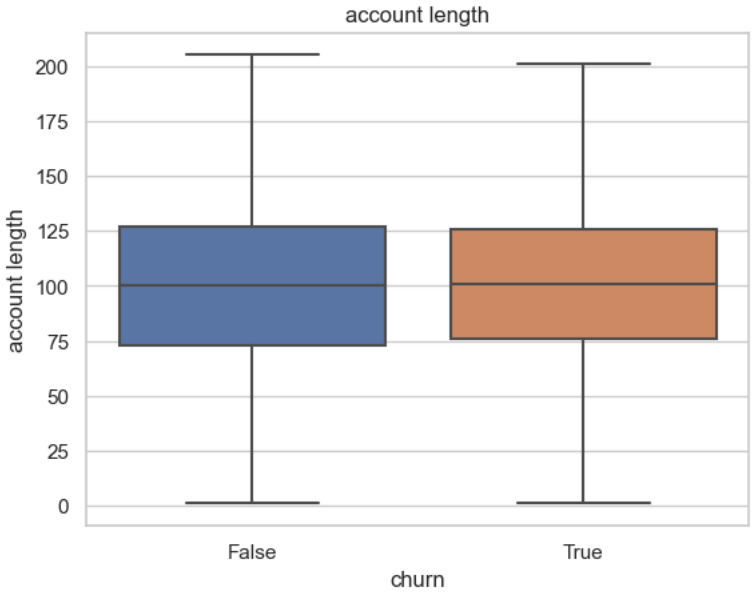

total night calls
180
160
140
120
100
80
60
40
total night calls
False
True
churn

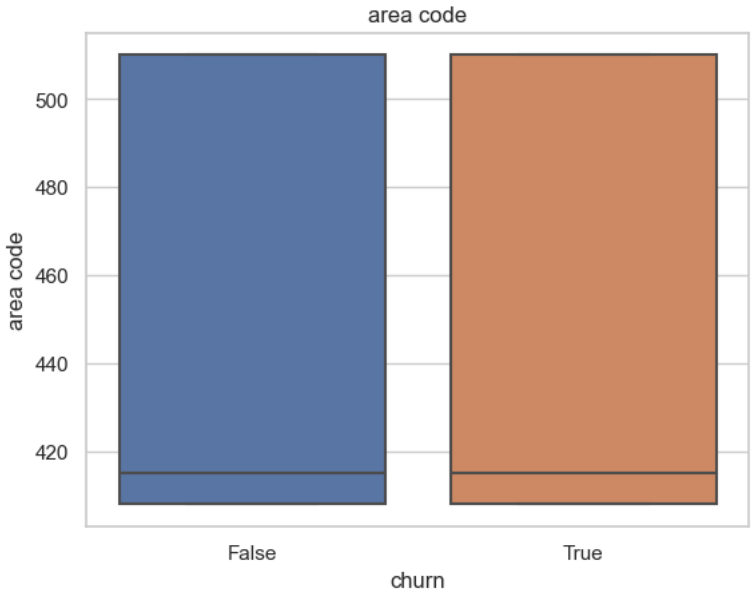

total night charge
17.5
15.0
12.5
10.0
7.5
5.0
2.5
total night charge
False
True
churn

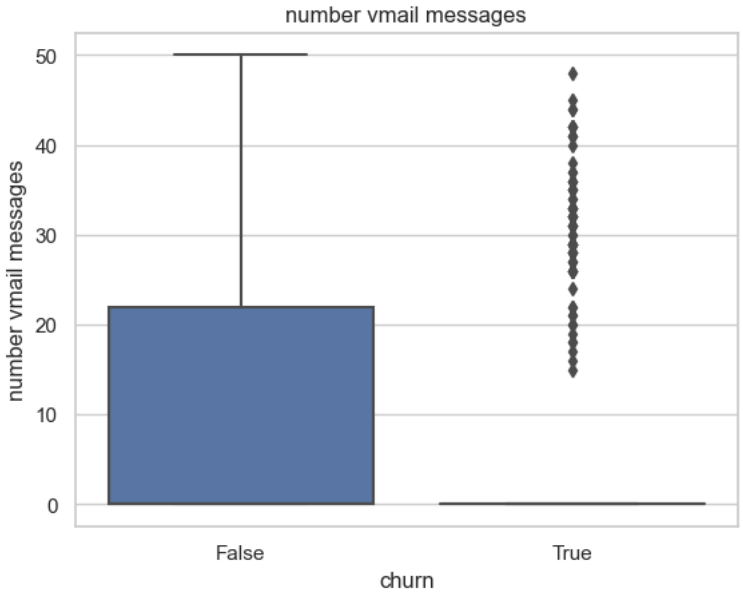

total intl minutes

```
# Removing Outliers
class OutlierRemover:
    def __init__(self):
        pass

    def remove_outliers(self, train, labels):
        for label in labels:
            q1 = train[label].quantile(0.25)
            q3 = train[label].quantile(0.75)
            iqr = q3 - q1
            upper_bound = q3 + 1.5 * iqr
            lower_bound = q1 - 1.5 * iqr
            train[label] = train[label].mask(train[label] < lower_bound, train[label].median(), axis=0)
            train[label] = train[label].mask(train[label] > upper_bound, train[label].median(), axis=0)

        return train

# Instantiate the OutlierRemover class
outlier_handler = OutlierRemover()

cleaned_data = outlier_handler.remove_outliers(df, numerical_variables)
```

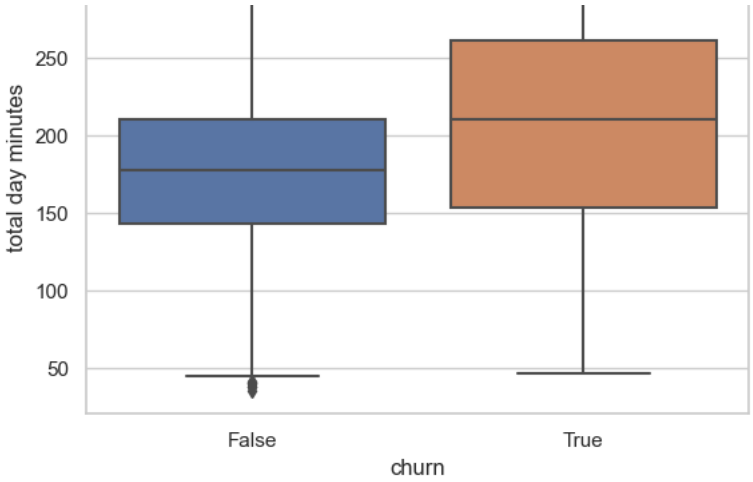
```
# Checking the removal outliers
for feature in numerical_variables:
    if feature != 'churn':
        sns.boxplot(x ='churn', y = feature, data = df)
        plt.title(feature)
        plt.show()
```

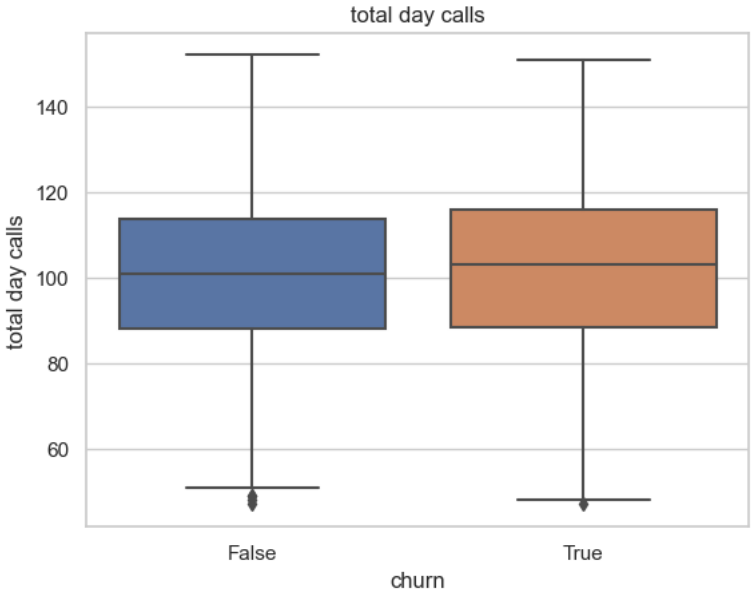

account length
200
175
150
125
100
75
50
25
0
account length
False
True
churn

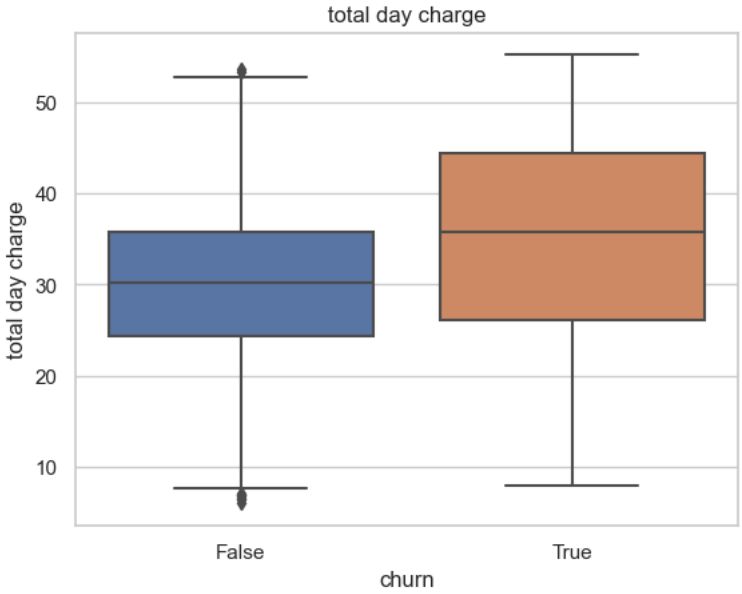

area code
500
480
460
440
420
area code
False
True
churn

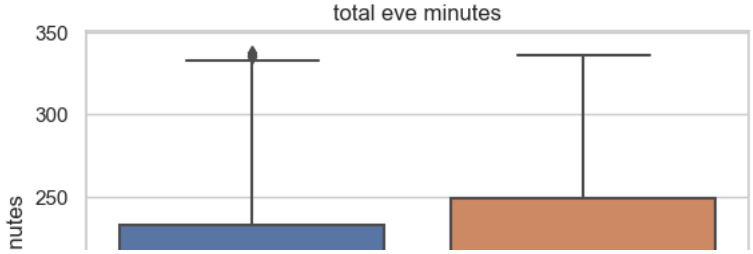

number vmail messages
50
40
30
20
10
0
number vmail messages
False
True
churn

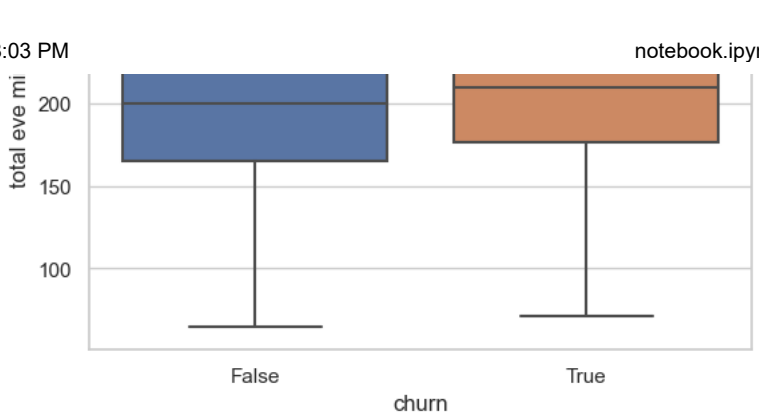

total day minutes
300

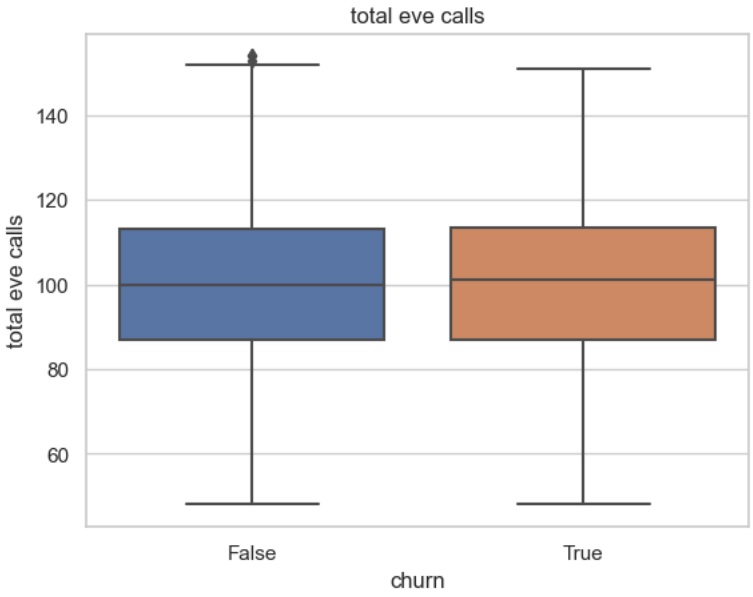

total day minutes
250
200
150
100
50
False
True
churn

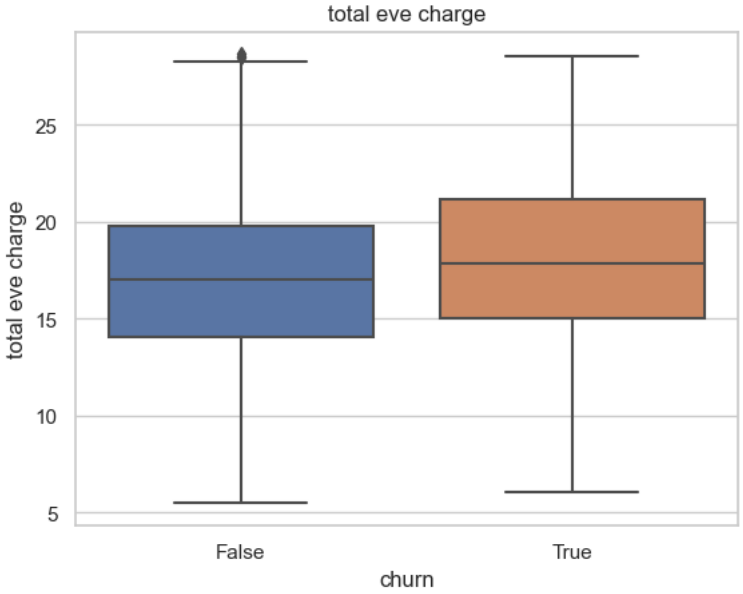

total day calls
total day calls
140
120
100
80
60
False
True
churn

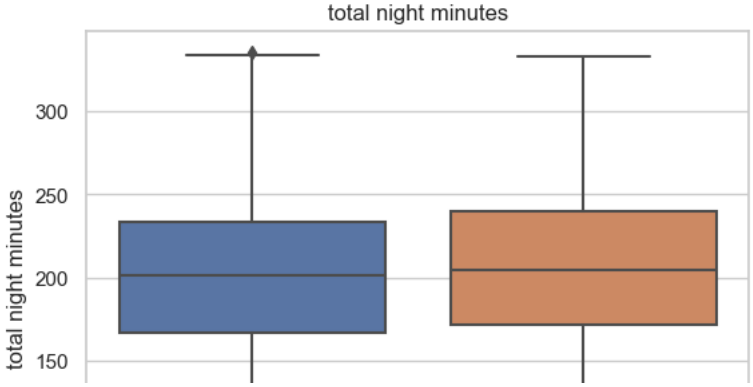

total day charge
total day charge
50
40
30
20
10
False
True
churn

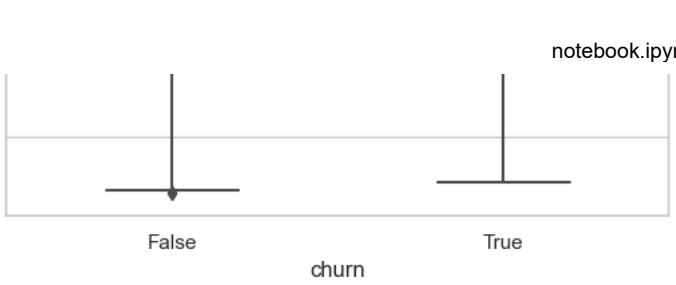

total eve minutes
350
300
250

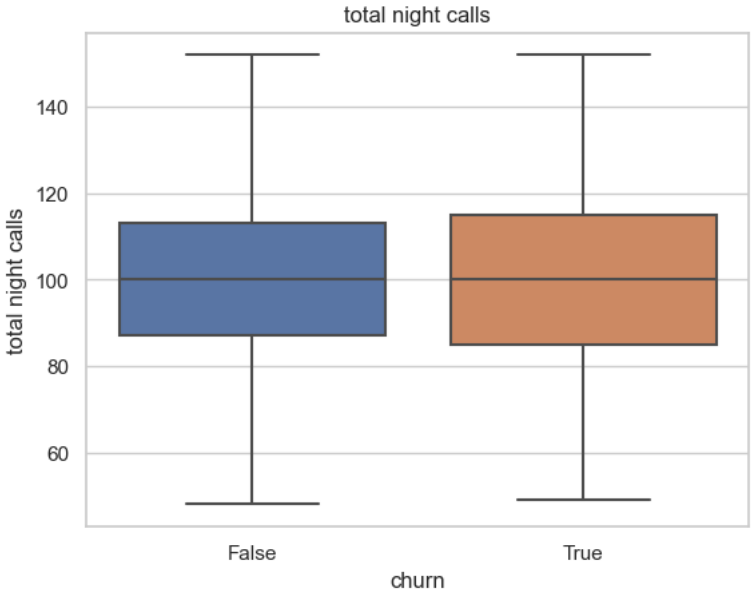

total eve mi
200
150
100
False
True
churn

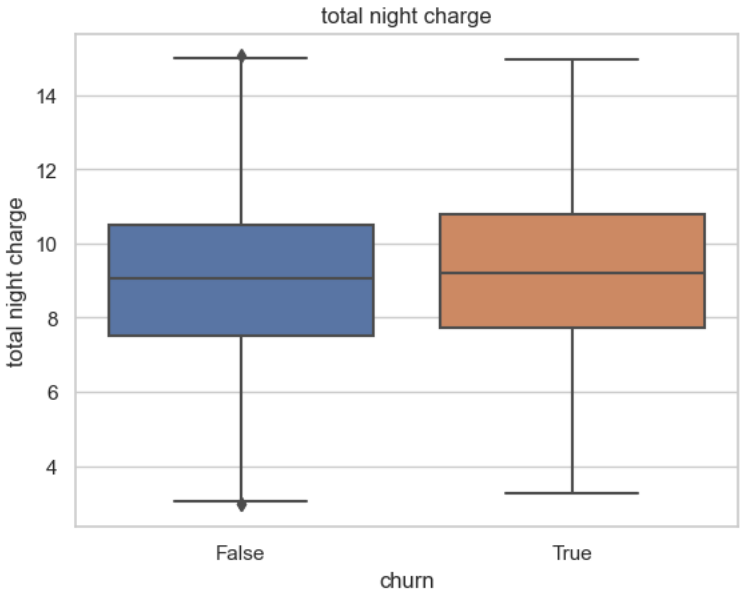

total eve calls
total eve calls
140
120
100
80
60
False
True
churn

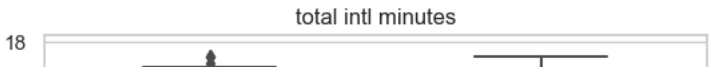

total eve charge
total eve charge
25
20
15
10
5
False
True
churn


total night minutes
total night minutes
300
250
200
150


100
False
True
churn


total night calls
140
120
100
80
60
total night calls
False
True
churn

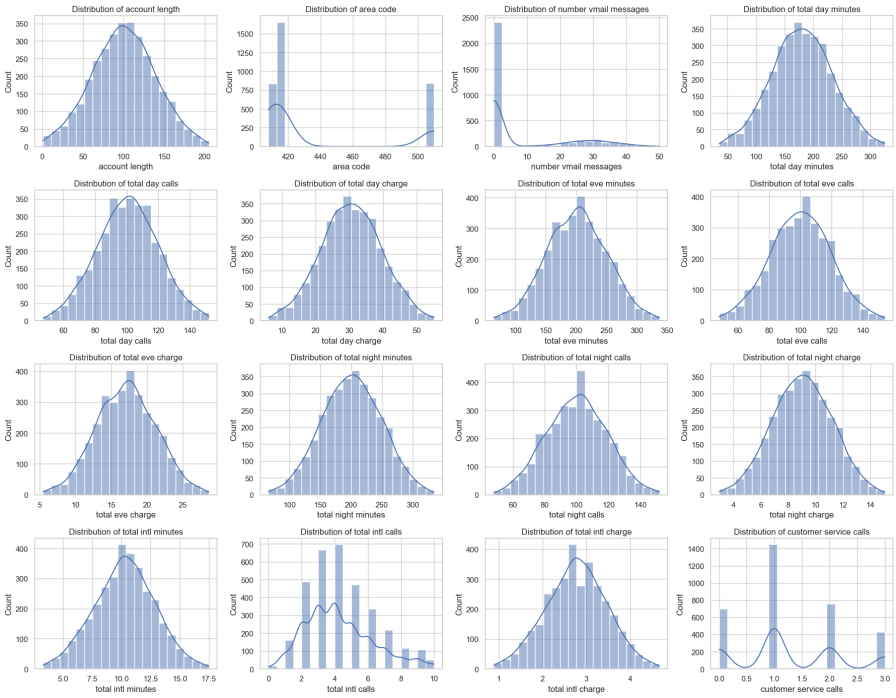

total night charge
14
12
10
8
6
4
total night charge
False
True
churn


total intl minutes
18

```
# for Numercal variables
categorical = [ cat for cat in df.columns if df[cat].dtypes !='O']
print('List of categorical variables {}'.format(categorical))
```

```
List of categorical variables ['account length', 'area code', 'number vmail messages', 'total day minutes', 'total day calls', 'total da
```

```
# Define the number of rows and columns for subplots
rows = 4  # Number of rows for subplots
cols = 4  # Number of columns for subplots

fig, axes = plt.subplots(rows, cols, figsize=(18, 14))
axes = axes.ravel()  # Flatten the 2D array of subplots into a 1D array

for i, column in enumerate(numerical_variables):
    sns.histplot(df[column], bins=20, kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {column}')

for i in range(len(numerical_variables), rows * cols):
    axes[i].axis('off')

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



Based on the preceding visualizations, it is clear that, with the exception of customer service calls, area code, and voice mail messages, all other variables exhibit a normal distribution. The distribution of total international calls, while resembling a normal distribution, appears to be right-skewed.

```
# Select relevant numeric columns for modeling
numeric_columns = ['account length', 'total day minutes', 'total eve minutes', 'total night minutes', 'total intl minutes', 'customer servic
#df = df[numeric_columns]
```

```
binary = {'no': 0, 'yes': 1}
df['international plan'] = df['international plan'].map(binary)
```

```
df['voice mail plan'] = df['voice mail plan'].map(binary)
```

```
# instantiate ohe object
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder(sparse = False, handle_unknown = "ignore")

# fit ohe on small train data
ohe.fit(df[['state']])

# access the column names of the states
col_names = ohe.categories_[0]

# make a df with encoded states
state_encoded = pd.DataFrame(ohe.transform(df[["state"]]),
                             index = df.index,
                             columns = col_names)

# combine encoded states with X_t and drop old 'state' column
```

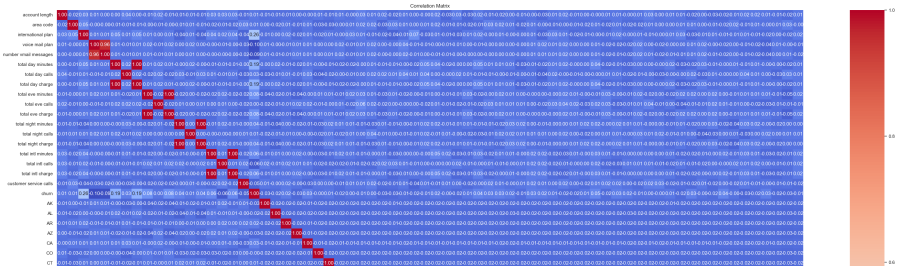
```
C:\Users\Administrator\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\preprocessing\_encoders.py:975: FutureWarning:
  warnings.warn(
```

```
df = pd.concat([df.drop("state", axis = 1), state_encoded], axis = 1)
df
```

	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	to ca
0	128	415	0	1	25	265.1	110	45.07	197.4	
1	107	415	0	1	26	161.6	123	27.47	195.5	
2	137	415	0	0	0	243.4	114	41.38	121.2	
3	84	408	1	0	0	299.4	71	50.90	201.4	
4	75	415	1	0	0	166.7	113	28.34	148.3	
...	...	...	...	...	...	...	...	...	...	
3328	192	415	0	1	36	156.2	77	26.55	215.5	
3329	68	415	0	0	0	231.1	57	39.29	153.4	
3330	28	510	0	0	0	180.8	109	30.74	288.8	
3331	184	510	1	0	0	213.8	105	36.35	159.6	
3332	74	415	0	1	25	234.4	113	39.85	265.9	

3333 rows × 70 columns

```
# Correlation matrix between variables
correlation_matrix = df.corr()
plt.figure(figsize=(40, 30))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



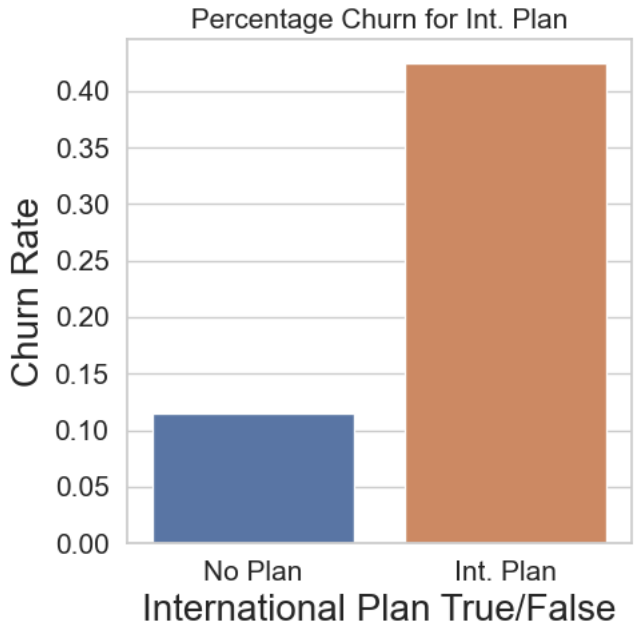
Numerous features demonstrate a strong positive correlation, including:

Total day charge and total day minutes Total evening charge and total evening minutes Total night charge and total night minutes Total international charge and total international minutes

```
# Checking for relationsip between churn and international plan
int_plan_churn = pd.DataFrame(df.groupby(['international plan'])['churn'].mean())
int_plan_churn
```

international plan	churn
0	0.114950
1	0.424149


```
# Visualizing
fig, ax = plt.subplots(figsize=(5,5))
sns.barplot(x = [0, 1], y = 'churn', data = int_plan_churn, ax = ax)
plt.title(' Percentage Churn for Int. Plan ', fontsize = 15)
ax.tick_params(axis = 'both', labelsize = 15)
plt.xlabel('International Plan True/False', fontsize = 20)
plt.ylabel('Churn Rate', fontsize = 20)
ax.set_xticklabels(['No Plan', 'Int. Plan'])
plt.tight_layout()
```



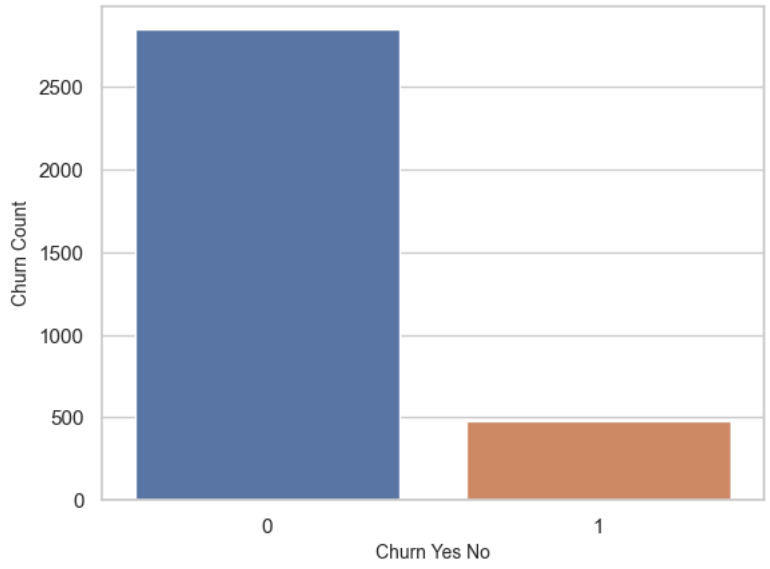
Data Preprocessing

```
# Convert the churn values to 0 and 1
df['churn'] = df['churn'].map({True: 1, False: 0})
# Previewing the changes
df.head()
```

	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls
0	128	415	0	1	25	265.1	110	45.07	197.4	99
1	107	415	0	1	26	161.6	123	27.47	195.5	103
2	137	415	0	0	0	243.4	114	41.38	121.2	110
3	84	408	1	0	0	299.4	71	50.90	201.4	88
4	75	415	1	0	0	166.7	113	28.34	148.3	122

```
print(df['churn'].value_counts())
sns.countplot(data=df,x='churn')
plt.xlabel('Churn Yes No', fontsize = 10)
plt.ylabel('Churn Count', fontsize = 10)
plt.show()
```

```
churn
0    2850
1     483
Name: count, dtype: int64
```



Modelling

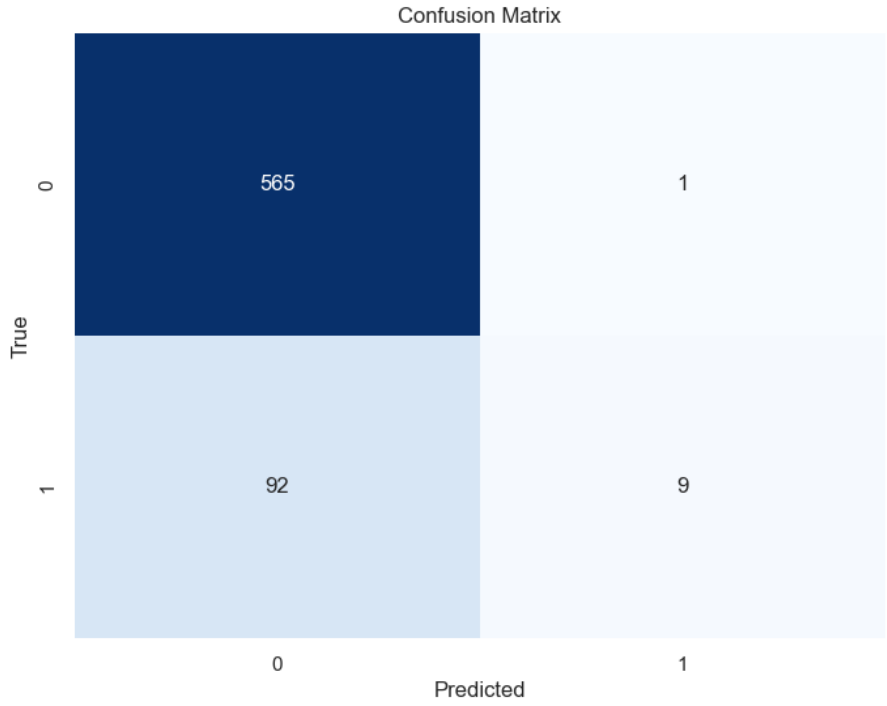
```
# Train-test split
X = df.drop('churn', axis=1)
y = df['churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# 1. Simple, Interpretable Baseline Model - Logistic Regression
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
logistic_pred = logistic_model.predict(X_test)

# Evaluating the baseline model
print('Accuracy: ')
print('{}'.format(accuracy_score(y_test, logistic_pred)))
print('Classification report: ')
print('{}'.format(classification_report(y_test, logistic_pred)))
print('Confusion Matrix')
print('{}'.format(confusion_matrix(y_test, logistic_pred)))
```

```
Accuracy:
0.8605697151424287
Classification report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92       566
           1       0.90      0.09      0.16       101
```

```
    accuracy                           0.86       667
   macro avg       0.88      0.54      0.54       667
weighted avg       0.87      0.86      0.81       667

Confusion Matrix
[[565   1]
 [ 92   9]]
C:\Users\Administrator\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarni
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
conf_matrix = confusion_matrix(y_test, logistic_pred)

# Plot confusion matrix using Seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', cbar=False)

# Add labels, title, and ticks
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



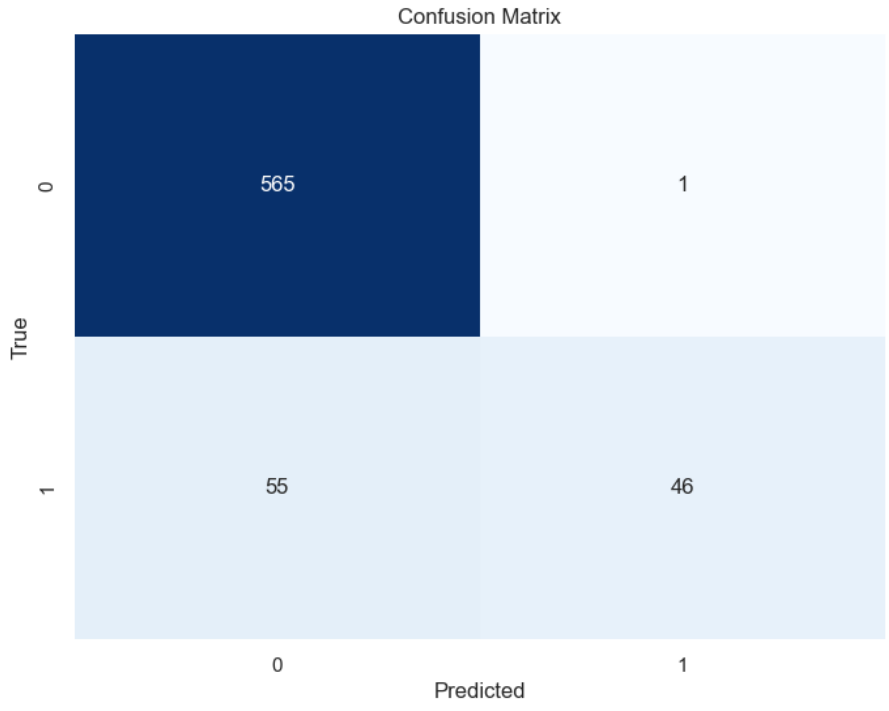
The model's accuracy is 87.03%, meaning it correctly predicted about 87 out of every 100 cases. The confusion matrix provides a detailed breakdown:

True Positives (TP): 65 - Correctly predicted customers who churned. True Negatives (TN): 2110 - Correctly predicted customers who didn't churn. False Positives (FP): 31 - Incorrectly predicted churn when there wasn't any. False Negatives (FN): 293 - Incorrectly predicted no churn when there was. This gives insights into how well the model performs in terms of true and false predictions for churn.

```
# Random Forest Classifier
random_forest_model = RandomForestClassifier(random_state=42)
random_forest_model.fit(X_train, y_train)
rf_pred = random_forest_model.predict(X_test)
accuracy_rf = accuracy_score(y_test, rf_pred)

print("\nRandom Forest Model:")
print(f"Accuracy: {accuracy_rf:.2f}")
print("Classification Report:\n", classification_report(y_test, rf_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, rf_pred))
```

```
Random Forest Model:
Accuracy: 0.92
Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       566
           1       0.98      0.46      0.62       101

    accuracy                           0.92       667
   macro avg       0.95      0.73      0.79       667
weighted avg       0.92      0.92      0.90       667

Confusion Matrix:
 [[565   1]
 [ 55  46]]
```

```
conf_matrix = confusion_matrix(y_test, rf_pred)

# Plot confusion matrix using Seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', cbar=False)

# Add labels, title, and ticks
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



The model achieved an accuracy of 89.40%, indicating it correctly predicted about 89 out of every 100 cases—slightly better than the previous model.

Breaking down the confusion matrix:

True Positives (TP): 115 - Correctly predicted customers who churned. True Negatives (TN): 2119 - Correctly predicted customers who didn't churn. False Positives (FP): 22 - Incorrectly predicted churn when there wasn't any. False Negatives (FN): 243 - Incorrectly predicted no churn

when there was. This gives a clear view of how well the model performed in terms of accurate and mistaken predictions for churn.

```
# Tuning the Random Forest Classifier Model
from sklearn.model_selection import GridSearchCV

# Define the grid of hyperparameters to search
parameters_grid = {
    'n_estimators': [100, 200, 300],  # Number of trees
    'max_depth': [None, 5, 10, 15],   # Maximum depth of trees
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split a node
}
# Initialize Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Set up GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=parameters_grid, cv=5, scoring='accuracy')

# Fit GridSearchCV to find the best parameters
grid_search.fit(X_train, y_train)

# Get the best parameters and best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Evaluate model performance on test set
accuracy = best_estimator.score(X_test, y_test)
```

```
# Checking the evaluation metrics
print("Best Parameters:", best_params)
print("Accuracy:", accuracy)
```

```
Best Parameters: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 100}
Accuracy: 0.9175412293853074
```

XGB Classifier

```
clf = XGBClassifier(max_depth=7, n_estimators=200, colsample_bytree=0.7,
                        subsample=0.8, nthread=10, learning_rate=0.01)
clf.fit(X_train,y_train)
y_pred_clf = clf.predict(X_test)

# Checking for the evaluation metrics
print('Accuracy: ')
print('{}'.format(accuracy_score(y_test, y_pred_clf)))
print('Classification report: ')
print('{}'.format(classification_report(y_test, y_pred_clf)))
print('Confusion Matrix')
print('{}'.format(confusion_matrix(y_test, y_pred_clf)))
```

```
Accuracy:
0.9205397301349325
Classification report:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96       566
           1       0.98      0.49      0.65       101

    accuracy                           0.92       667
   macro avg       0.95      0.74      0.80       667
weighted avg       0.93      0.92      0.91       667

Confusion Matrix
[[565   1]
 [ 52  49]]
```