SYRIATEL CUSTOMER CHURN PROJECT

Presented by: Samuel Mbugua Igecha

Business Understanding

SyriaTel are a telecommunications company based in the USA who are looking to better understand the reasons behind their customer churn so they formulate strategies to better counter it. Attracting new customers can be very expensive as seems that being proactive by retaining your customers is the key to effective churn management in a telecom company

1.1 Objectives

1.To build machine learning models that will predict how likely a customer will churn.

2.To find the best machine learning model for the correct classification of churned/retained customers.

3.To determine which features affect the customer churn rate thereby giving necessary recommendations.

Data Understanding

Data containing 3333 rows and 21 columns has been provided which is more than sufficient and robust enough to be used for our analysis.That might however change if the need for more data arises.

state: 2-letter code of the US state of customer residence

account length: Number of months the customer has been with the current telcom provider

area code: 3 digit area code.

international plan: (yes/no). The customer has international plan.

voice mail plan: (yes/no). The customer has voice mail plan.

number vmail messages: Number of Voice Mail Messages

total day minutes: Total minutes of day calls.

total day calls: Total number of day calls.

total day charge: Total charge of day calls

total eve minutes: Total minutes of evening calls.

total eve calls: Total number of evening calls.

total eve charge: Total charge of evening calls.

total night minutes: Total minutes of night calls.

total night calls: Total number of night calls.

total night charge: Total charge of night calls.

total intl minutes: Total minutes of international calls.

total intl calls: Total number of international calls.

total intl charge: Total charge of international calls

customer service calls: number of calls made to customer service

churn: Customer Churn, True means churned customer, False means retained customer

Data Preparation

In [1]:
```
# Importing all the important libraries
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, f1_score,precision_score, recall_score, ro
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDi
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier


pd.set_option('display.max_columns', None)
```

In [2]:
```
df = pd.read_csv('Data/bigml_59c28831336c6604c800002a.csv')
df.head()
```

Out[2]:

	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	c
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	197.4	99	
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	195.5	103	
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	121.2	110	
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	61.9	88	
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	148.3	122	

1.1 Data Cleaning

In [3]:
```
df.shape
```

Out[3]: (3333, 21)

In [4]:
```
# Checking for duplicates
df.duplicated().sum()
```

Out[4]: 0

In [5]:
```
# checking for missing data
df.isna().sum()
```

Out[5]:
```
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 [6]:
```
df.columns
```

Out[6]:
```
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 [7]:
```
# renaming column titles by removing spaces
df.columns = df.columns.str.replace(' ', '_')
```

In [8]:
```
df.columns
```

Out[8]:
```
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')
```

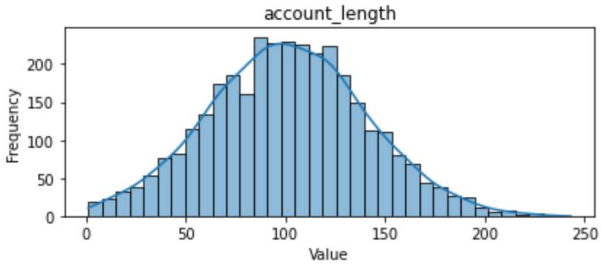
1.2 Exploratory Data Analysis

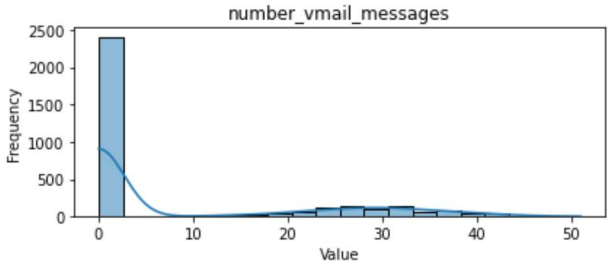
In [9]:
```
df.describe()
```

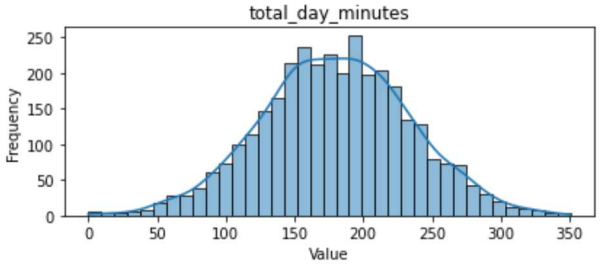
Out[9]:

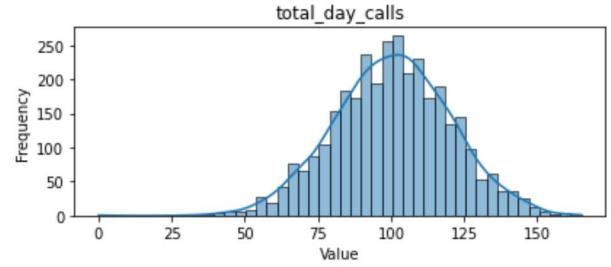
	account_length	area_code	number_vmail_messages	total_day_minutes	total_day_calls	total_day_
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.

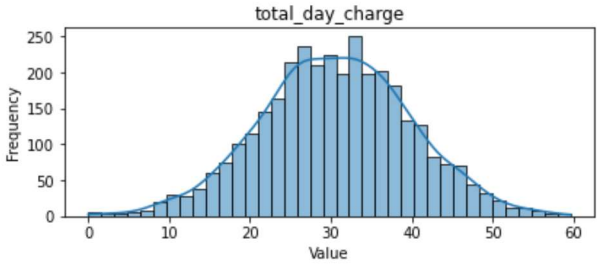
In [10]:
```
columns_to_plot = ['account_length', 'number_vmail_messages',
       'total_day_minutes', 'total_day_calls', 'total_day_charge',
       'total_eve_minutes', 'total_eve_calls', 'total_eve_charge',
       'total_night_minutes', 'total_night_calls', 'total_night_charge',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge',
       'customer_service_calls']

num_plots = len(columns_to_plot)
num_rows = math.ceil(num_plots / 2)
num_cols = 2

# Create the figure and axes for the subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 21))

# Flatten the axes array
axes = axes.flatten()

# Loop through the specified columns and plot histograms using seaborn
for i, column in enumerate(columns_to_plot):
    ax = axes[i]  # Select the current axis
    sns.histplot(data=df, x=column, kde=True, ax=ax)
    ax.set_title(column)
    ax.set_xlabel('Value')
    ax.set_ylabel('Frequency')

# Remove any unused subplots
for j in range(num_plots, num_rows * num_cols):
    fig.delaxes(axes[j])

# Adjust the spacing between subplots
plt.tight_layout()

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

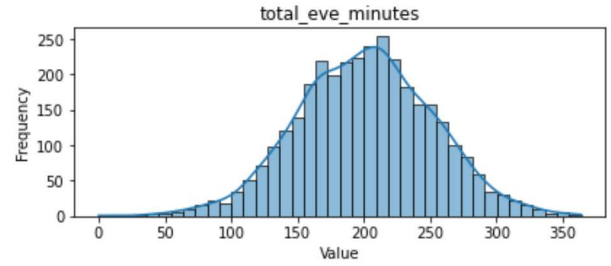

account_length
Frequency
Value

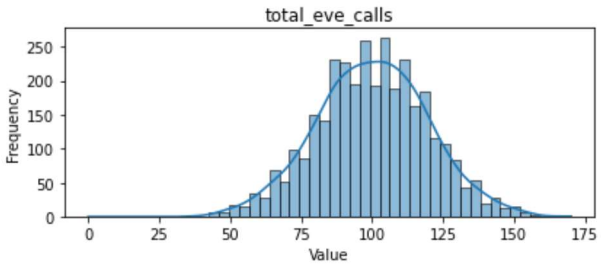

number_vmail_messages
Frequency
Value

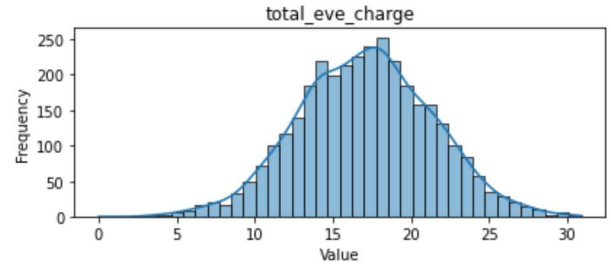

total_day_minutes
Frequency
Value

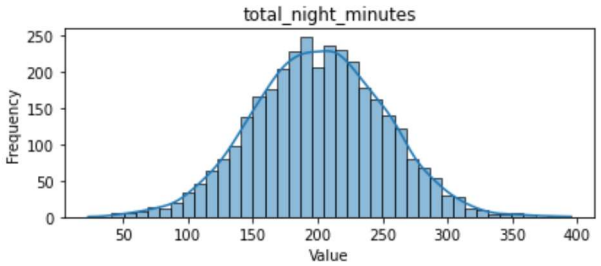

total_day_calls
Frequency
Value

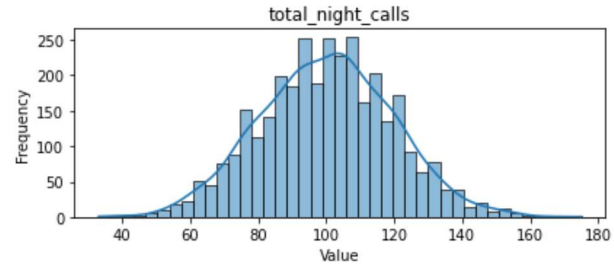

total_day_charge
Frequency
Value

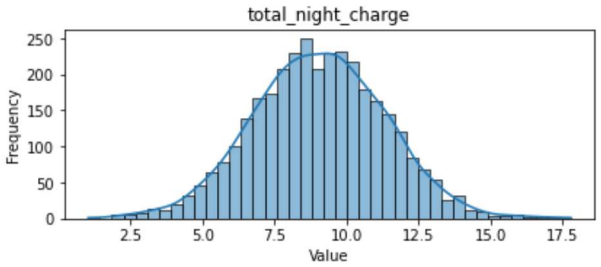

total_eve_minutes
Frequency
Value

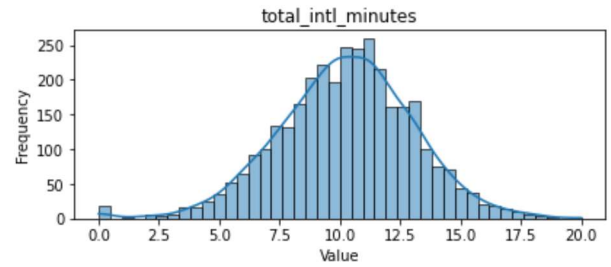

total_eve_calls
Frequency
Value

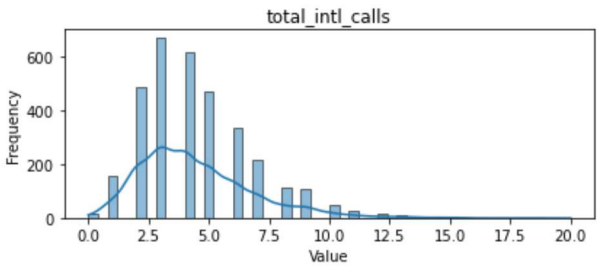

total_eve_charge
Frequency
Value

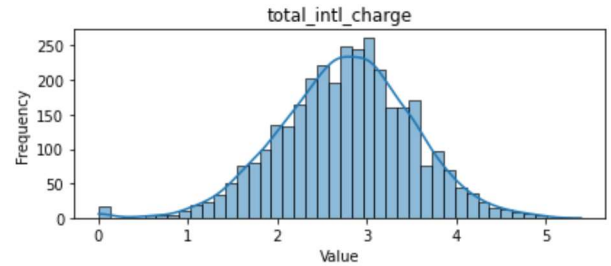

total_night_minutes
Frequency
Value

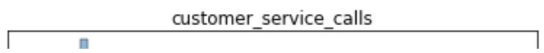

total_night_calls
Frequency
Value

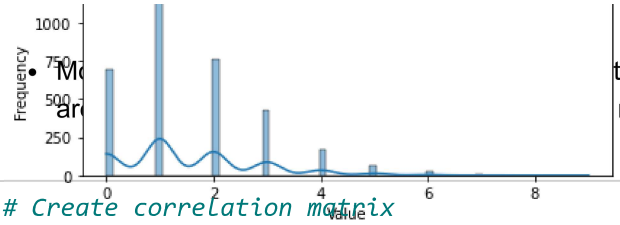

total_night_charge
Frequency
Value


total_intl_minutes
Frequency
Value


total_intl_calls
Frequency
Value

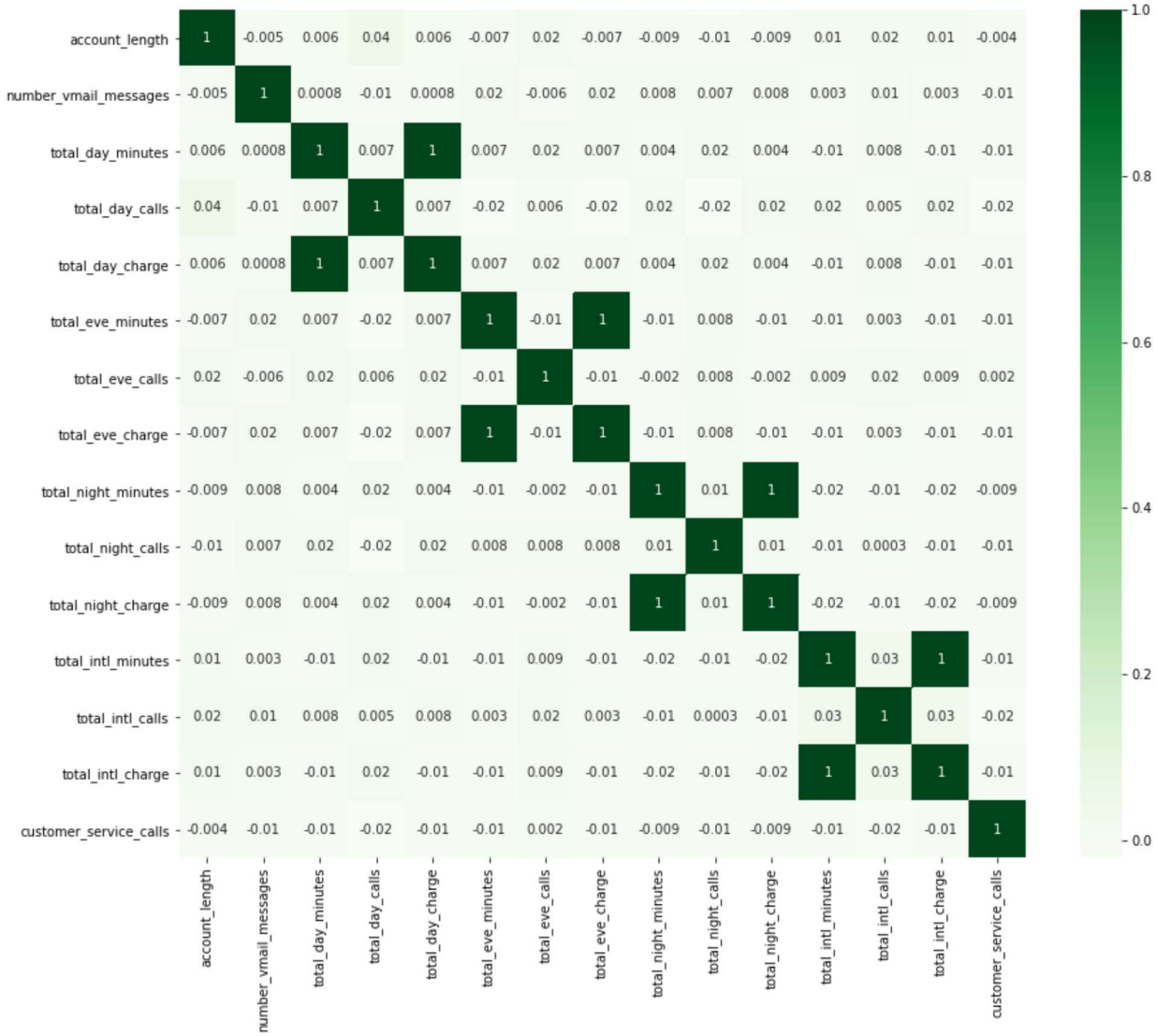

total_intl_charge
Frequency
Value


customer_service_calls



- M ted. Customers service Calls and international calls ar right skewed.

In [11]:
```
# Create correlation matrix
corr_mat = df[columns_to_plot].corr()
mask = np.triu(np.ones_like(corr_mat, dtype=bool))
plt.subplots(figsize=(15,12))
sns.heatmap(corr_mat, annot=True, cmap='Greens', square=True,fmt='.0g');
plt.xticks(rotation=90);
plt.yticks(rotation=0);
```



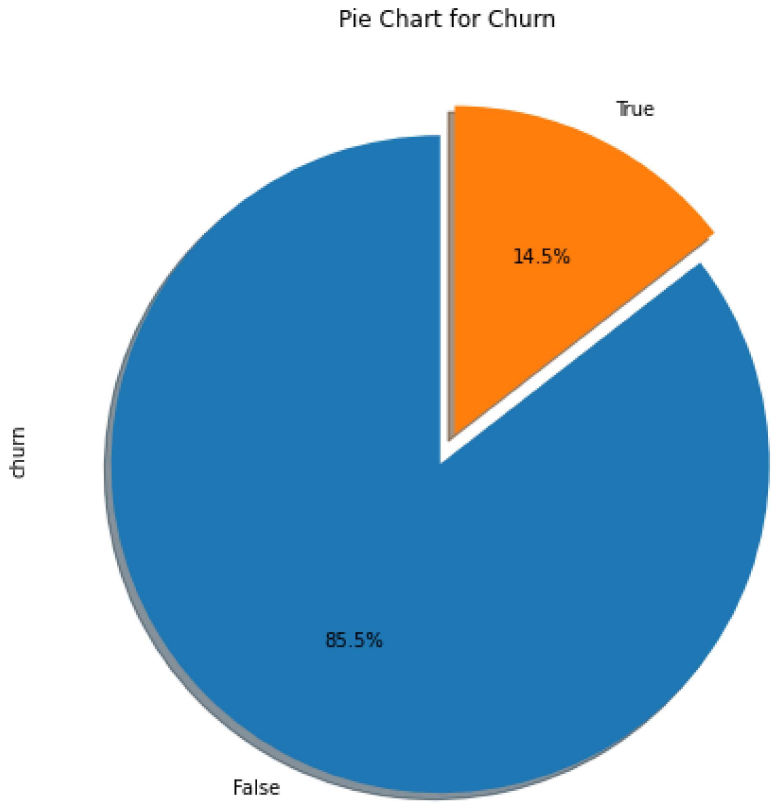
Most of the features in the above heatmap aren't correlated with the exception of;

- total_day_charge & total_day_minutes
- total_eve_charge and total_eve_minutes
- total_night_charge and total_night_minutes
- total_int_charge and total_int_minutes_features

In [12]:
```
df.churn.value_counts()
```

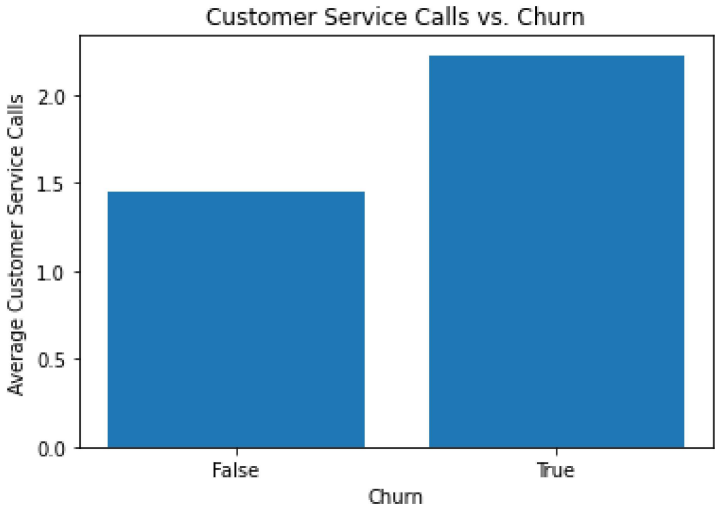
Out[12]:
```
False    2850
True      483
Name: churn, dtype: int64
```

In [13]:
```
# Visualization of target variable/Churn rate
df['churn'].value_counts().plot.pie(explode=[0.05,0.05], autopct='%1.1f%%',  startangl
plt.title('Pie Chart for Churn')
plt.show()
```



- Based off our dataset, SyriaTel experienced a Churnrate of 14.5%
- The class imbalance will need to be addressed however for our model

In [14]:
```
grouped_data = df.groupby('churn')['customer_service_calls'].mean()

# Plot the bar graph
plt.bar(grouped_data.index.astype(str), grouped_data)
plt.xlabel('Churn')
plt.ylabel('Average Customer Service Calls')
plt.title('Customer Service Calls vs. Churn')
plt.show()
```

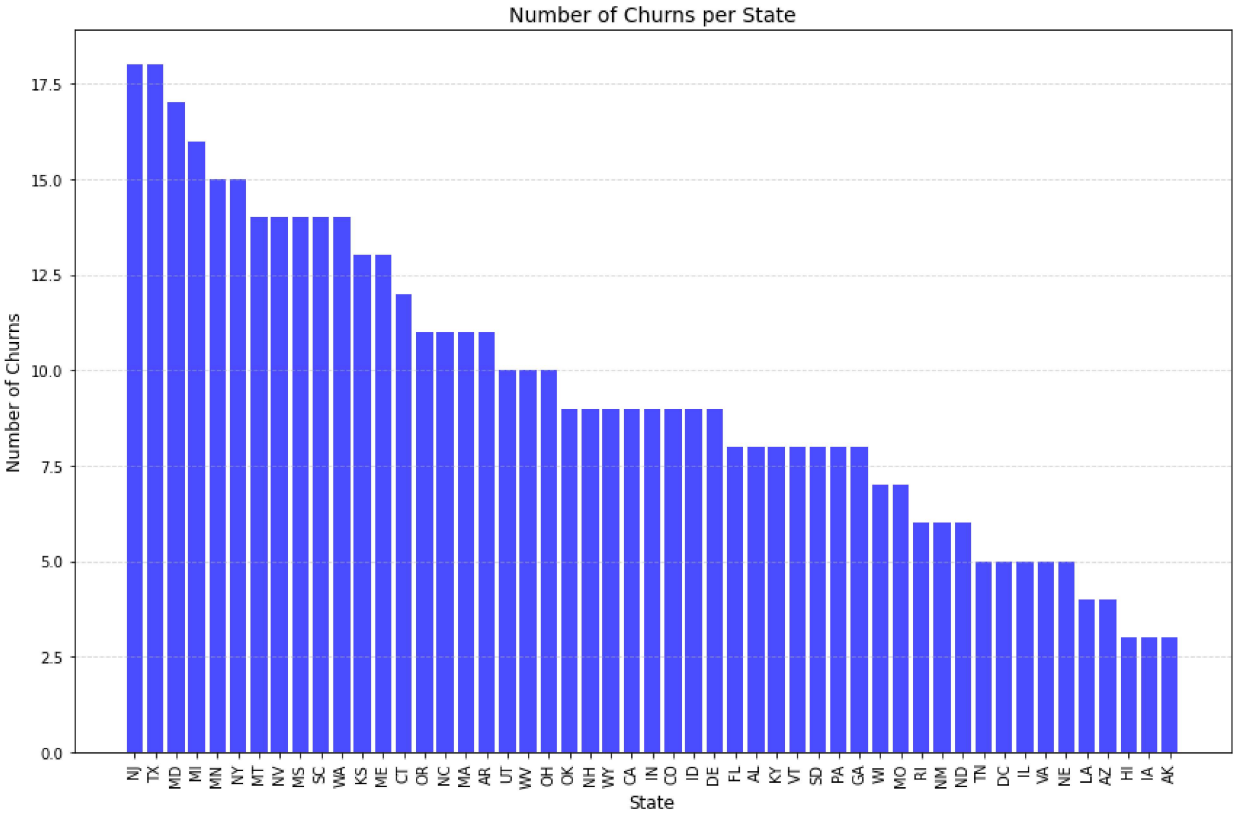


- Churners seem to make more customer service calls on average than non-churners based off the barplot

In [15]:
```
# customers are from all 51 states
df['state'].nunique()
```

Out[15]: 51

In [16]:
```
# Calculate the number of churns per state
churn_count = df['churn'].groupby(df['state']).sum().reset_index()

# Sort the data by churn count in descending order
churn_count = churn_count.sort_values(by='churn', ascending=False)

# Create a bar plot with increased size and aesthetics
plt.figure(figsize=(12, 8))
plt.bar(churn_count['state'], churn_count['churn'], color='blue', alpha=0.7)

# Customization options
plt.xlabel('State', fontsize=12)
plt.ylabel('Number of Churns', fontsize=12)
plt.title('Number of Churns per State', fontsize=14)
plt.xticks(rotation=90, fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.5)

# Adjust the layout and save the plot
plt.tight_layout()
plt.savefig('churns_per_state.png', dpi=300)  # Save the plot as an image
plt.show()
```



- Some states had more customers churning than others. NJ(NewJersey) & TX(Texas) had the most churn. There could be a variety of factors for this, it could be phone network coverage. It's not clear, location might not even a play a role in customer churn as we look to explore the rest of other features in the dataset

In [17]:
```
df.area_code.value_counts()
```

Out[17]:
```
415    1655
510     840
408     838
Name: area_code, dtype: int64
```

There's only 3 area codes and you'd expect each state to be zoned with its own area code. The area codes are 415, 510 & 408 and according to the website allareacodes, they are in CA(Carlifornia). We'll assume the states are the true location of each individual customer. So we'll drop the area code column including phone_number as they won't be needed for our modelling.

In [18]:
```
#dropping the area code & phone number columns
df = df.drop(columns=['area_code', 'phone_number'] , axis=1)
```

In [19]:
```
df['international_plan'] = df['international_plan'].replace(('yes', 'no'), (1, 0))
df['international_plan'] = df['international_plan'].astype('int')
df['voice_mail_plan'] = df['voice_mail_plan'].replace(('yes', 'no'), (1, 0))
df['voice_mail_plan'] = df['voice_mail_plan'].astype('int')
df['churn'] = df['churn'].replace((True, False), (1, 0))
df['churn'] = df['churn'].astype('int')
```

1.3 Data Preprocessing

There's some interesting trends in the data but we don't need to create dummy variables for each of the 51 states so its best we do some feature engineering by creating a new category "region" where the states will be allocated into US regions. Just like how Kenya is divided into regions like Northern, Central, Western etc, it's the same case in the USA. To do this i'll get some new data from the US Census Bureau website then merge it to our current one.

In [20]:
```
# import data for states and their regions
df_regions = pd.read_csv('Data/state-geocodes.csv')
df_regions.head()
```

Out[20]:

	Region	RegionName	Division	State (FIPS)	StateName	StateCode
0	3	South	6	1	Alabama	AL
1	4	West	9	2	Alaska	AK
2	4	West	8	4	Arizona	AZ
3	3	South	7	5	Arkansas	AR
4	4	West	9	6	California	CA

In [21]:
```
# Drop unnecessary columns and rename columns
df_regions = df_regions.drop(columns=['Region', 'Division', 'State (FIPS)', 'StateName
df_regions = df_regions.rename({'StateCode': 'state', 'RegionName': 'region'}, axis=1)
df_regions['state'] = df_regions['state'].astype('category')
df_regions.head()
```

Out[21]:

	region	state
0	South	AL
1	West	AK
2	West	AZ
3	South	AR
4	West	CA

In [22]:
```
# Merge region data with original data
df = df.merge(df_regions, on='state', how='left')
df.head()
```

Out[22]:

	state	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	t
0	KS	128	0	1	25	265.1	
1	OH	107	0	1	26	161.6	
2	NJ	137	0	0	0	243.4	
3	OH	84	1	0	0	299.4	
4	OK	75	1	0	0	166.7	

In [23]:
```
df.region.value_counts()
```

Out[23]:
```
South        1109
West          831
Midwest       802
Northeast     591
Name: region, dtype: int64
```

In [24]:
```
# We will now drop the state column and create dummy variables from region
df = pd.get_dummies(df, columns=['region'], drop_first=True)
df = df.drop(['state'], axis=1)
df.head()
```

Out[24]:

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_da
0	128	0	1	25	265.1	
1	107	0	1	26	161.6	
2	137	0	0	0	243.4	
3	84	1	0	0	299.4	
4	75	1	0	0	166.7	

In [25]:
```
# Split our dataset
y = df['churn']
X = df.drop('churn', axis=1)
```

In [26]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0
```

In [27]:
```
# Standardize the datac so that its on the same scale
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
scaled_df_train = pd.DataFrame(X_train_scaled, columns=X_train.columns)
scaled_df_train.head()
```

Out[27]:

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_da
0	0.341991	0.0	0.0	0.000000	0.580476	0
1	0.268398	0.0	0.0	0.000000	0.378559	0
2	0.497835	0.0	1.0	0.235294	0.634515	0
3	0.303030	0.0	0.0	0.000000	0.802731	0
4	0.515152	0.0	1.0	0.843137	0.509297	0

In [28]:
```
# Previous original class distribution
print('Original class distribution: \n')
print(y.value_counts())
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)
# Preview synthetic sample class distribution
print('----------------------------------------')
print('Synthetic sample class distribution: \n')
print(pd.Series(y_train_resampled).value_counts())
```

```
Original class distribution:

0    2850
1     483
Name: churn, dtype: int64
----------------------------------------
Synthetic sample class distribution:

1    1993
0    1993
Name: churn, dtype: int64
```

- SMOTE (Synthetic Minority Oversampling Technique) was used to address the class imbalance.

Modelling

1.1 Logistic Regression(Baseline Model)

In [29]:
```
lr = LogisticRegression()
lr.fit(X_train_resampled,y_train_resampled)
y_pred_lr = lr.predict(X_test_scaled)
print('Accuracy on training set:',lr.score(X_train_resampled,y_train_resampled))
print('Accuracy on test set:',lr.score(X_test_scaled,y_test))
```

```
Accuracy on training set: 0.7709483191169092
Accuracy on test set: 0.765
```

- The Model has an accuracy of 76.5% on the test and 77% on the training set. This means that it's neither overfitting nor underfitting which is good. However, it is important to consider other evaluation metrics and perform a more comprehensive analysis.

In [30]:
```
print (classification_report(y_test, y_pred_lr))
```

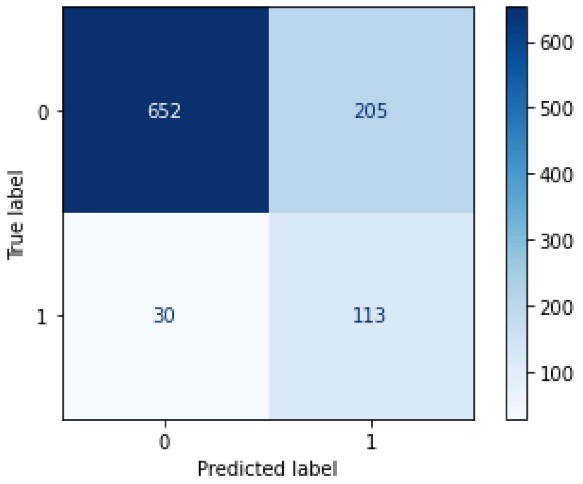
```
              precision    recall  f1-score   support

           0       0.96      0.76      0.85       857
           1       0.36      0.79      0.49       143

    accuracy                           0.77      1000
   macro avg       0.66      0.78      0.67      1000
weighted avg       0.87      0.77      0.80      1000
```

In [31]:
```
print('Accuracy score for testing set: ',round(accuracy_score(y_test,y_pred_lr),5))
print('F1 score for testing set: ',round(f1_score(y_test,y_pred_lr),5))
print('Recall score for testing set: ',round(recall_score(y_test,y_pred_lr),5))
print('Precision score for testing set: ',round(precision_score(y_test,y_pred_lr),5))
```

```
Accuracy score for testing set:  0.765
F1 score for testing set:  0.49024
Recall score for testing set:  0.79021
Precision score for testing set:  0.35535
```

In [32]:
```
cnf_matrix = confusion_matrix(y_test, y_pred_lr)

disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=lr.classes_)
disp.plot(cmap=plt.cm.Blues)
```

Out[32]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f62a9a8a90>
```



- Model accuracy is 76.5%, which isn't bad. F1 score is only 49% which means the test will only be accurate half the times it is ran. However, low precision score of 35.5% suggests that there may be a higher number of false positives as evidenced in our confusion matrix, while the recall score indicates that the model is able to capture a good proportion of true positives.
- The model shows room for improvement, particularly in terms of precision. I am considering exploring different models & ensemble methods like random forests XGBoost to improve its performance.

1.2 Decision Tree

In [33]:
```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train_resampled,y_train_resampled)

y_pred_dt = decision_tree.predict(X_test_scaled)

print('Accuracy on training set:',decision_tree.score(X_train_resampled,y_train_resampl
print('Accuracy on test set:',decision_tree.score(X_test_scaled,y_test))
```

```
Accuracy on training set: 1.0
Accuracy on test set: 0.903
```

- The Model has an accuracy of 90.3% on the test and 100% on the training set. It performs better than the baseline model but there could be potential overfitting, but let's look at other evaluation metrics.

In [34]:
```
print (classification_report(y_test, y_pred_dt))
```

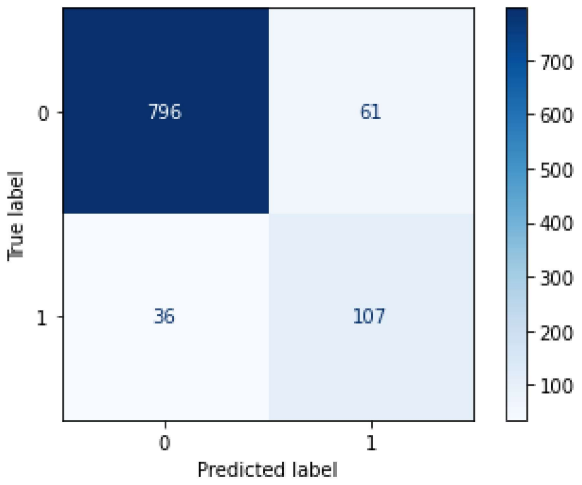
```
              precision    recall  f1-score   support

           0       0.96      0.93      0.94       857
           1       0.64      0.75      0.69       143

    accuracy                           0.90      1000
   macro avg       0.80      0.84      0.82      1000
weighted avg       0.91      0.90      0.91      1000
```

In [35]:
```
print('Accuracy score for testing set: ',round(accuracy_score(y_test,y_pred_dt),5))
print('F1 score for testing set: ',round(f1_score(y_test,y_pred_dt),5))
print('Recall score for testing set: ',round(recall_score(y_test,y_pred_dt),5))
print('Precision score for testing set: ',round(precision_score(y_test,y_pred_dt),5))
```

```
Accuracy score for testing set:  0.903
F1 score for testing set:  0.6881
Recall score for testing set:  0.74825
Precision score for testing set:  0.6369
```

In [36]:
```
cnf_matrix = confusion_matrix(y_test, y_pred_dt)

disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=decision_tre
disp.plot(cmap=plt.cm.Blues)
```

Out[36]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f62abb0640>
```



- Based on these metrics, the decision tree model shows reasonably good performance. However, there is room for improvement, especially in terms of precision. i'll try different algorithms to enhance its performance further.

1.3 Random Forest Classifier

In [37]:
```
forest = RandomForestClassifier()

forest.fit(X_train_resampled,y_train_resampled)
y_pred_forest = forest.predict(X_test_scaled)

print('Accuracy on training set:',forest.score(X_train_resampled,y_train_resampled))
print('Accuracy on test set:',forest.score(X_test_scaled,y_test))
```

```
Accuracy on training set: 1.0
Accuracy on test set: 0.947
```

- The Model has an accuracy of 100% on the training set and 94.7% on the training set. A perfect accuracy on the training set may indicate a potential risk of overfitting. Overall, achieving high accuracy on both the training and test sets is a positive outcome. However, it's best i consider other evaluation metrics and perform further analysis to gain a better understanding of the model's performance. Fine tuning the model might also be required.

In [38]:
```
print(classification_report(y_test, y_pred_forest))
```

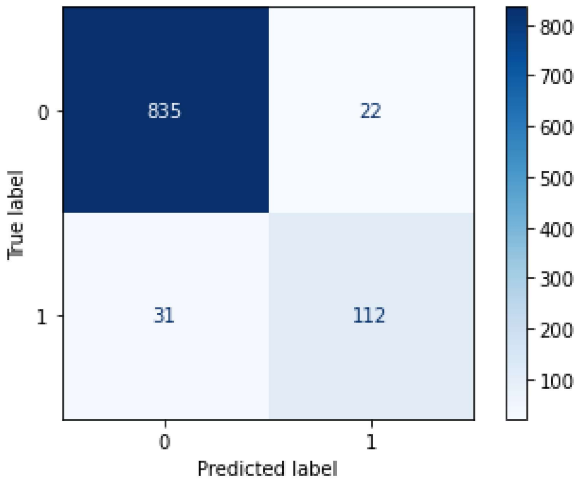
```
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       857
           1       0.84      0.78      0.81       143

    accuracy                           0.95      1000
   macro avg       0.90      0.88      0.89      1000
weighted avg       0.95      0.95      0.95      1000
```

In [39]:
```
print('Accuracy score for testing set: ',round(accuracy_score(y_test,y_pred_forest),5)
print('F1 score for testing set: ',round(f1_score(y_test,y_pred_forest),5))
print('Recall score for testing set: ',round(recall_score(y_test,y_pred_forest),5))
print('Precision score for testing set: ',round(precision_score(y_test,y_pred_forest),
```

```
Accuracy score for testing set:  0.947
F1 score for testing set:  0.80866
Recall score for testing set:  0.78322
Precision score for testing set:  0.83582
```

In [40]:
```
cnf_matrix = confusion_matrix(y_test, y_pred_forest)

disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=forest.class
disp.plot(cmap=plt.cm.Blues)
```

Out[40]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f62ac58520>
```



- Model accuracy score is 94.7% which indicates that the model is performing well on unseen data while F1 score is 80.8%. Both scores are much higher for this model compared to our baseline logistic regression model, which is good news. This model is also performing better than the decision tree
- Both the recall score (78.3%) and precision score (83.5%) are high, suggesting that the model is able to effectively identify positive instances and minimize false positives.

1.4 XGBOOST

In [41]:
```
xgb = XGBClassifier()

xgb.fit(X_train_resampled, y_train_resampled)
y_pred_xgb = xgb.predict(X_test_scaled)
print('Accuracy on training set:',xgb.score(X_train_resampled,y_train_resampled))
print('Accuracy on test set:',xgb.score(X_test_scaled,y_test))
```

```
Accuracy on training set: 1.0
Accuracy on test set: 0.949
```

- It seems like the XGBoost model is performing exceptionally well with high accuracy on both the training and test sets. An accuracy of 100% on the training set indicates that the model perfectly fits the training data, while an accuracy of 94.9% on the test set suggests that it generalizes well to unseen data.

In [42]:
```
print(classification_report(y_test, y_pred_xgb))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       857
           1       0.88      0.75      0.81       143

    accuracy                           0.95      1000
   macro avg       0.92      0.87      0.89      1000
weighted avg       0.95      0.95      0.95      1000
```

- Overall, the XGBoost model demonstrates strong performance on the testing set with high precision, recall, and F1 score. It achieves an accuracy of 95%. However, the recall for class 1 is lower compared to class 0. I may need to fine-tune the model to achieve a better balance between these metrics. To fine-tune the XGBoost model while using less computing power, i will consider RandomSearchCV instead of GridSearchCV

In [43]:
```
# Define the parameter grid
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 200, 300],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'gamma': [0, 1, 2]
}

# Create the RandomizedSearchCV object
random_search = RandomizedSearchCV(estimator=xgb, param_distributions=param_grid, cv=3

# Fit the RandomizedSearchCV object to the training data
random_search.fit(X_train_resampled, y_train_resampled)

# Print the best parameters and best score
print("Best Parameters: ", random_search.best_params_)
print("Best Score: ", random_search.best_score_)
```

```
Best Parameters:  {'subsample': 0.8, 'n_estimators': 100, 'max_depth': 5, 'learning_r
ate': 0.1, 'gamma': 0, 'colsample_bytree': 0.8}
Best Score:  0.9533429051042394
```

In [44]:
```
y_pred = random_search.best_estimator_.predict(X_test_scaled)
recall = recall_score(y_test, y_pred)

print('Recall score for testing set: ',round(recall_score(y_test,y_pred),5))
```

```
Recall score for testing set:  0.76224
```

- There's an improvement in our recall score from 75% to now 76.2% after finetuning the model. i'll fine tune it further to see if there's any significant improvement

In [45]:
```
# Define the refined parameter grid
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 200, 300],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'gamma': [0, 1, 2]
}


# Create the RandomizedSearchCV object with increased iterations
random_search = RandomizedSearchCV(estimator=xgb, param_distributions=param_grid, cv=3

# Fit the RandomizedSearchCV object to the training data
random_search.fit(X_train_resampled, y_train_resampled)

# Print the best parameters and best score
print("Best Parameters: ", random_search.best_params_)
print("Best Score: ", random_search.best_score_)
```

```
Best Parameters:  {'subsample': 1.0, 'n_estimators': 100, 'max_depth': 7, 'learning_r
ate': 0.1, 'gamma': 1, 'colsample_bytree': 0.8}
Best Score:  0.9583597746894273
```

In [46]:
```
y_pred = random_search.best_estimator_.predict(X_test_scaled)
recall = recall_score(y_test, y_pred)

print('Recall score for testing set: ',round(recall_score(y_test,y_pred),5))
```

```
Recall score for testing set:  0.78322
```

- There's now an improvement in our recall score from 76.2% to now 78.3% after finetuning the model once again. Now i will consider additional steps such as cross-validation to further enhance the model's performance.

In [47]:
```
from sklearn.model_selection import cross_validate

scoring = ['accuracy', 'recall']
results = cross_validate(xgb, X_train_resampled, y_train_resampled, cv=5, scoring=scor

# Print the recall scores
print("Recall Scores: ", results['test_recall'])
print("Mean Recall: ", results['test_recall'].mean())
print("Standard Deviation: ", results['test_recall'].std())
```

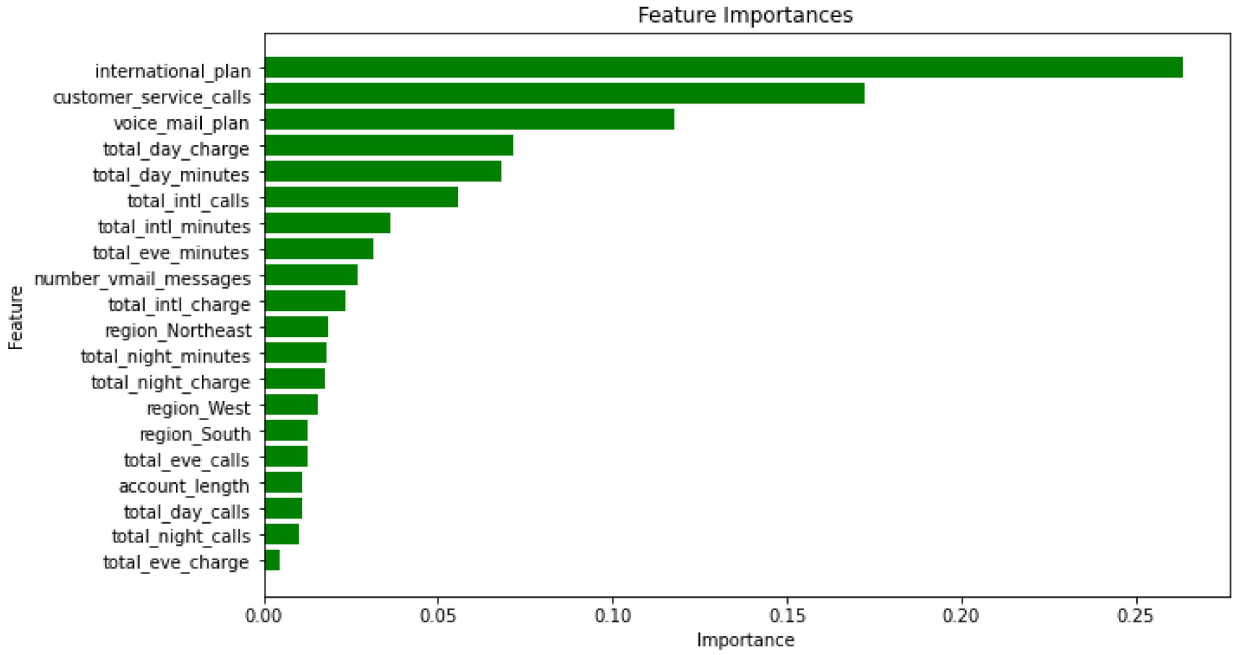
```
Recall Scores:  [0.7518797  0.99246231 0.98743719 0.98245614 0.98746867]
Mean Recall:  0.9403408017531264
Standard Deviation:  0.0942836637103103
```

- Based on the recall scores obtained from cross-validation, it appears that the XGBoost model has a relatively high average recall score of approximately 94.3%. This suggests that the model is performing well.
- The individual recall scores indicate that the model's performance may vary slightly across different subsets of the data. The standard deviation of 0.0942 further confirms this.

Final Model Evaluation

- XGBoost has so far been the best model out of the bunch after alot of finetuning and cross-validation has gone into it inorder to improve on its overall performance. Alot of focus went into improving the recall score. Lastly, I will look at the feature importance of my model.

In [48]:
```
importance_df = pd.DataFrame({'feature': X_train.columns, 'importance': xgb.feature_im
importance_df = importance_df.sort_values('importance', ascending=False)

fig, ax = plt.subplots(figsize=(10, 6))
ax.barh(importance_df['feature'], importance_df['importance'], color='green')
ax.invert_yaxis()  # Invert the y-axis to show the most important features at the top
ax.set_xlabel('Importance')
ax.set_ylabel('Feature')
ax.set_title('Feature Importances')
plt.show()
```



Conclusion and recommendations

Based on the feature importance chart, the three most important features for predicting customer churn are total_day_charge, international_plan, and customer_service_calls. This suggests that customers who have an international plan are more likely to churn, as are those who make day calls. In addition, customers who have made a larger number of customer service calls are also more likely to churn.

With this information, the telephone company can take steps to reduce churn and retain customers. For example, they could investigate whether their international plans are meeting the needs of their customers or if there are alternative plans that may be more suitable. The company could also evaluate the importance of lowering day charges for customers, and consider the impact on churn rates if such plans are not offered. Finally, they may want to improve on their customer service to decrease the likelihood of churn.

Through various research, it has emerged that churn in the telecom industry is most often due to bad customer service. Customers canceled their contracts for the following reasons:

- companies wasted their time meaning it took forever to have their issues resolved.
- they had to call more than once.
- untrained or incompetent agents.
- inferior self-service options.

It seems that being proactive is the key to effective churn management in telecom