1 SyriaTel Customer Analysis

By: Brian Lee

1.1 Business problem

SyriaTel telecommunications company has hired us to analyze the causes to customer churn. **Churn** is whether a customer will stop doing business with the company. **The goal is to maintain SyriaTel customers in order to stably increase company profits and to build good customer reputation for future clientele.** We are trying to predict the likelihood of a customer churn based on a user's communication usage, plans, and other related factors.

Business Questions to have in mind:

- Do we need to improve our current customer plans?
- How does customer call usage affect their decisions to stay with SyriaTel?
- How is our response to customer needs?

1.2 Data Understanding

This project uses the SyriaTel customer dataset, which can be found in 'telecoms.csv' in the 'data' folder.

In [2]:

```
# Import necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import statsmodels.api as sm
import scipy.stats as stats
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder

from sklearn.linear_model import LinearRegression
from sklearn import metrics

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.metrics import precision_score, recall_score, accuracy_score, f1
from sklearn.metrics import confusion_matrix,recall_score,precision_recall_cu
from sklearn.metrics import precision_recall_fscore_support,f1_score,fbeta_sc
from sklearn.metrics import classification_report, plot_roc_curve, plot_confu
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.metrics import classification_report

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

import xgboost as xgb
from xgboost import XGBClassifier
```

executed in 7.44s, finished 00:50:32 2021-11-13

In [3]:

```
df = pd.read_csv('data/telecoms.csv')
df.head()
```

executed in 61ms, finished 00:50:32 2021-11-13

Out[3]:

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

5 rows × 21 columns

1.3 Data Preparation

Let's quickly examine the dataset and clean it up for proper analysis and modeling

In [4]:

```
display(df.info())
display(df.describe())
```

executed in 61ms, finished 00:50:32 2021-11-13

```
<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	total day calls	total day charge	tota mi
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.0(
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.98
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.71
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.0(
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.6(
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.4(
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.3(
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.7(

In [5]:

```
# Adjust column names with '_'
df.columns = df.columns.str.replace(' ','_')
df.info()
```

executed in 15ms, finished 00:50:32 2021-11-13

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account_length          3333 non-null   int64  
 2   area_code               3333 non-null   int64  
 3   phone_number            3333 non-null   object 
 4   international_plan      3333 non-null   object 
 5   voice_mail_plan         3333 non-null   object 
 6   number_vmail_messages   3333 non-null   int64  
 7   total_day_minutes       3333 non-null   float64
 8   total_day_calls         3333 non-null   int64  
 9   total_day_charge        3333 non-null   float64
 10  total_eve_minutes       3333 non-null   float64
 11  total_eve_calls         3333 non-null   int64  
 12  total_eve_charge        3333 non-null   float64
 13  total_night_minutes     3333 non-null   float64
 14  total_night_calls       3333 non-null   int64  
 15  total_night_charge      3333 non-null   float64
 16  total_intl_minutes      3333 non-null   float64
 17  total_intl_calls        3333 non-null   int64  
 18  total_intl_charge       3333 non-null   float64
 19  customer_service_calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

In [6]:
```
# Find missing values columns
df.isna().sum()
```
executed in 14ms, finished 00:50:32 2021-11-13

Out[6]:
```
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
```

- No missing values found.

Phone numbers are just unique identifiers. Does not provide additional information. Could drop it

In [7]:
```
# Drop phone_number
df.drop('phone_number', axis=1, inplace=True)
```
executed in 15ms, finished 00:50:32 2021-11-13

In [8]:
```
# Values for area_code
df.area_code.value_counts()
```
executed in 14ms, finished 00:50:32 2021-11-13

Out[8]:
```
415    1655
510     840
408     838
Name: area_code, dtype: int64
```

Taking a look at the area_code values show that there are only 3 area codes (San Francisco area), despite the data being declared for several different states. It will be better to drop the area codes in this case.

In [9]:
```
# Drop area_code column
df.drop('area_code', axis = 1, inplace=True)
```
executed in 14ms, finished 00:50:32 2021-11-13

Let's change the categorical columns to integers for easier analysis

In [10]:
```
# Change churn and plans to int
df['churn'] *= 1

df['international_plan'] = df['international_plan'].apply(lambda x: 1 if x=='
df['voice_mail_plan'] = df['voice_mail_plan'].apply(lambda x: 1 if x=='yes' e

df[['churn', 'international_plan', 'voice_mail_plan']].astype(int)
```
executed in 28ms, finished 00:50:32 2021-11-13

Out[10]:

	churn	international_plan	voice_mail_plan
0	0	0	1
1	0	0	1
2	0	0	0
3	0	1	0
4	0	1	0
...	...	...	...
3328	0	0	1
3329	0	0	0
3330	0	0	0
3331	0	1	0
3332	0	0	1

3333 rows × 3 columns

In [11]:
```
df.head()
```
executed in 30ms, finished 00:50:32 2021-11-13

Out[11]:

	state	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_da
0	KS	128	0	1	25	
1	OH	107	0	1	26	
2	NJ	137	0	0	0	
3	OH	84	1	0	0	
4	OK	75	1	0	0	

1.4 Feature Engineering

Let's make a few features that will be useful to examine on a customer's overall phone usage.

In [12]:

```
# Creating a column for total customer charges by adding all the individual c
df['total_customer_charges'] = df['total_day_charge'] + df['total_eve_charge'
df['total_customer_charges'].head()
```

executed in 14ms, finished 00:50:32 2021-11-13

Out[12]:

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

In [13]:

```
# Doing the same method for total calls and minutes
df['total_customer_calls'] = df['total_day_calls'] + df['total_eve_calls'] +
df['total_customer_minutes'] = df['total_day_minutes'] + df['total_eve_minute
df.head()
```

executed in 31ms, finished 00:50:32 2021-11-13

Out[13]:

	state	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_da
0	KS	128	0	1	25	
1	OH	107	0	1	26	
2	NJ	137	0	0	0	
3	OH	84	1	0	0	
4	OK	75	1	0	0	

5 rows × 22 columns

1.5 Exploratory Data Analysis

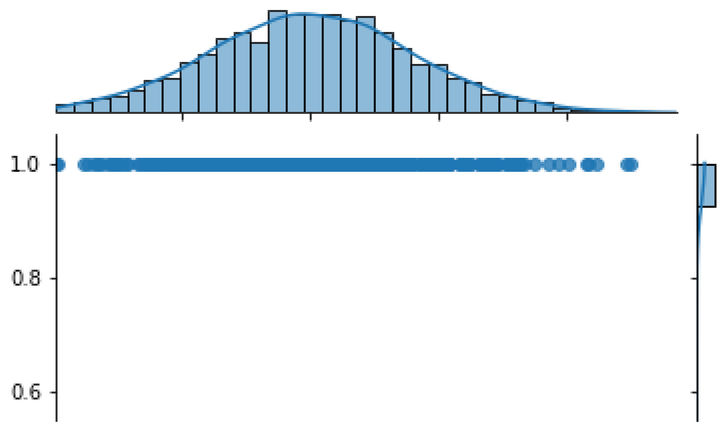
Using the cleaned data, we will examine the distributions of the columns and descriptive statistics for the dataset

In [14]:

```
# Let's take a look at each column in relation with churn
for col in df.columns[1:]:
    sns.jointplot(x=col, y='churn', data=df, kind='reg');
```

executed in 15.6s, finished 00:50:48 2021-11-13

```
C:\Users\leebr\anaconda3\envs\learn-env\lib\site-packages\seaborn\axisgri
d.py:1559: RuntimeWarning: More than 20 figures have been opened. Figures
created through the pyplot interface (`matplotlib.pyplot.figure`) are ret
ained until explicitly closed and may consume too much memory. (To contro
l this warning, see the rcParam `figure.max_open_warning`).
  f = plt.figure(figsize=(height, height))
```



Observations

- Customers with international plan seem to slightly lean towards churning
- Customers with voicemail plan seem to slightly lean towards NOT churning
- More voicemail msgs (suggesting high plan usage) slightly leans towards NOT churning
- As a general trend, the **more** calls/minutes/charges that customers use/receives, the **more** they are inclined to churn from SyriaTel.

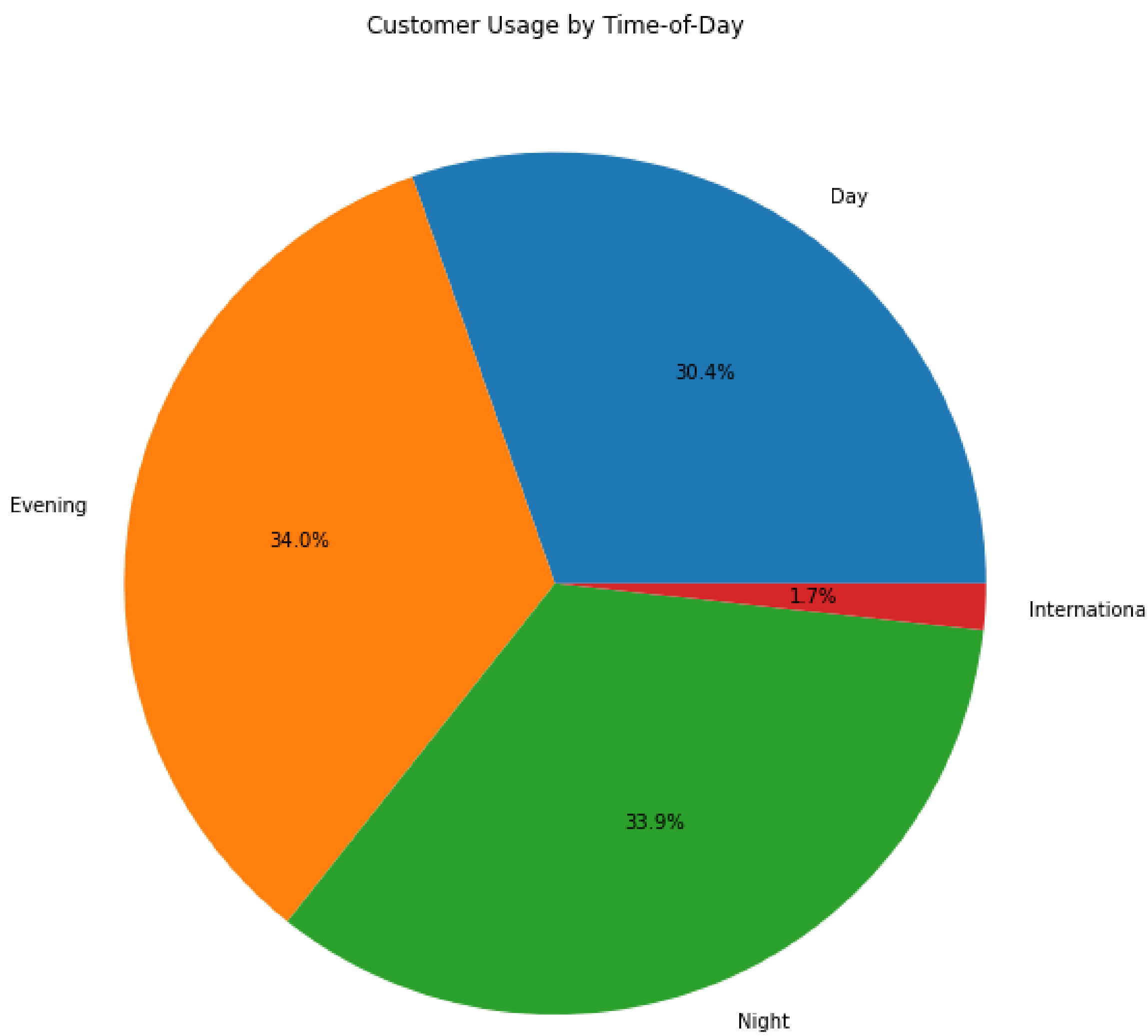
Let's see how total how much time-of-day affects customer usage

- Day: 5 AM to 5 PM
- Evening: 5 PM to 9 PM
- Night: 9 PM to 4 AM

In [15]:

```
# Set minutes data up for pie chart
data_time = df[['total_day_minutes','total_eve_minutes','total_night_minutes'
data_time.rename(index={"total_day_minutes": "Day", "total_eve_minutes": "Eve

# Plot
pie, ax = plt.subplots(figsize=[10,20])
labels = data_time.index
plt.pie(data_time, labels=labels, autopct='%1.1f%%')
plt.title('Customer Usage by Time-of-Day')
plt.show()
```

executed in 125ms, finished 00:50:48 2021-11-13

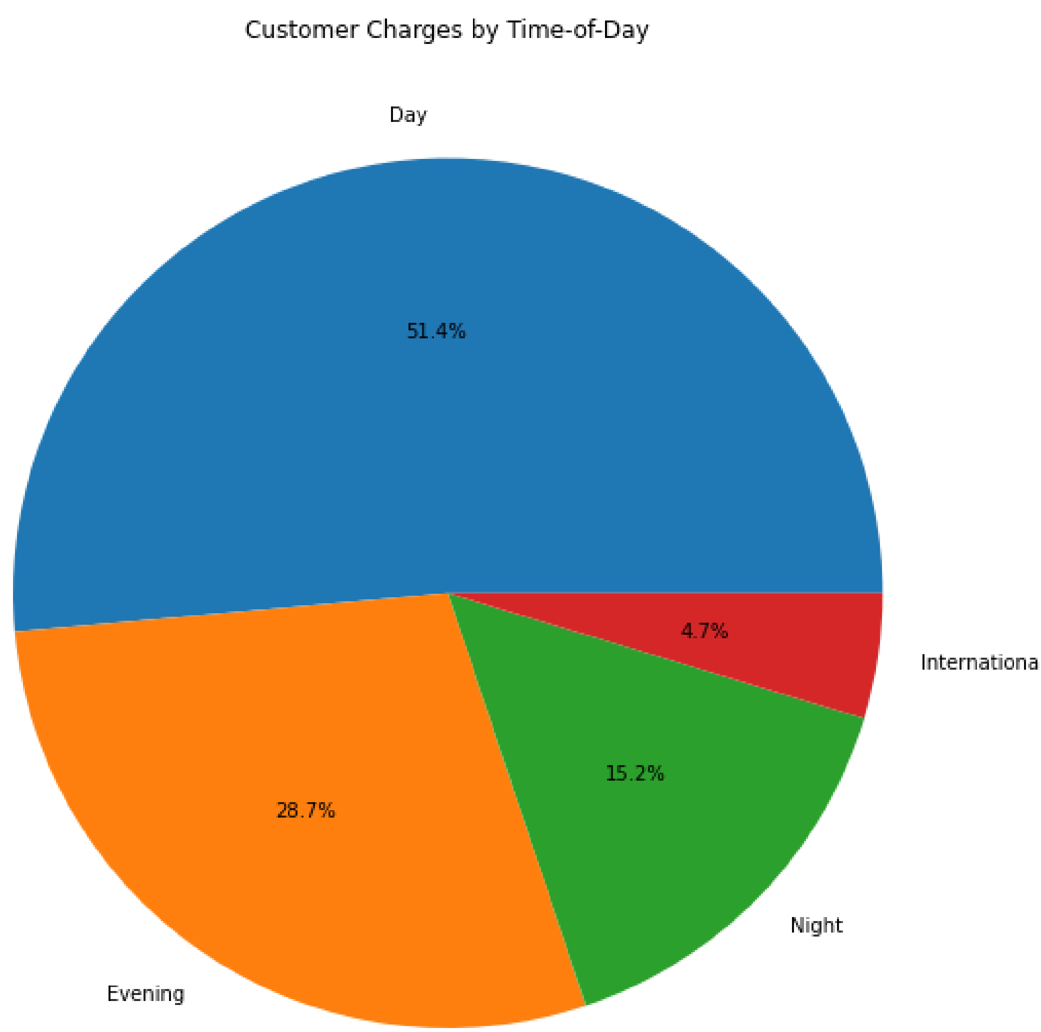


- Surprisingly, each part of the day is equally spent on calls
- Customers spend most of their calls in the latter part of the day (Evening + Night)

In [16]:

```
# Set data up for pie chart
data_charges = df[['total_day_charge','total_eve_charge','total_night_charge'
data_charges.rename(index={"total_day_charge": "Day", "total_eve_charge": "Ev

# Plot
pie, ax = plt.subplots(figsize=[10,20])
labels = data_charges.index
plt.pie(data_charges, labels=labels, autopct='%1.1f%%')
plt.title('Customer Charges by Time-of-Day')
plt.show()
```

executed in 123ms, finished 00:50:48 2021-11-13

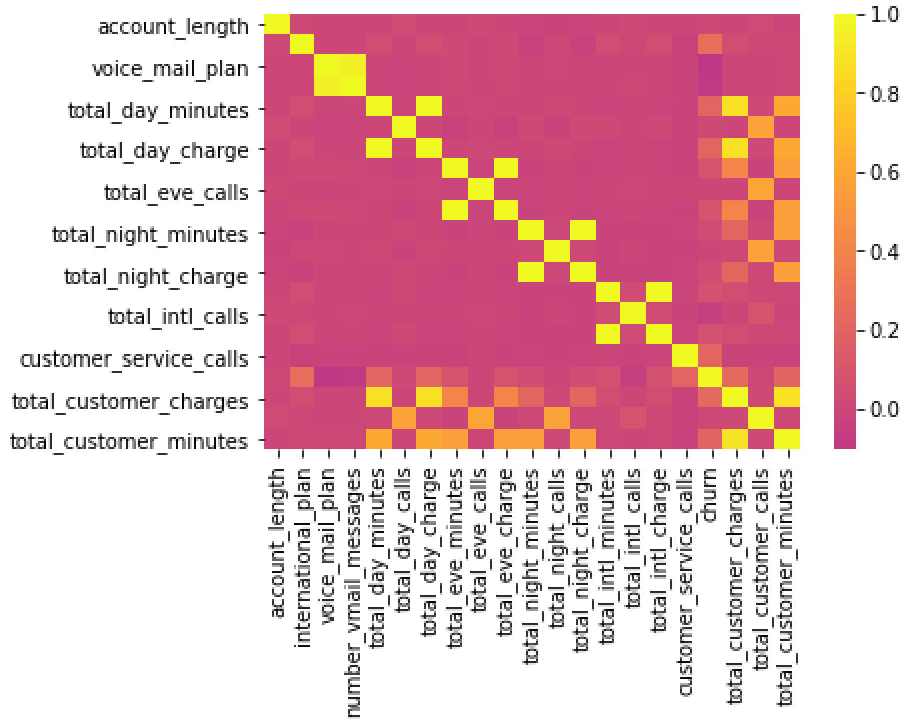


- Despite most usages being at Night and Evening, the total customer charges are the largest for day.
- It may be better to reduce the Day time charges and increase the Evening and Night time charges as it will lead to larger profits.

In [17]:

```
# Heatmap for correlation values
import seaborn as sns
sns.heatmap(df.corr(), cmap='plasma', center=0);
```

executed in 501ms, finished 00:50:49 2021-11-13



There appears to be be some heavy multicollinearity between some factors. Let's identify which ones are causing an issue

In [18]:

```
# Displays if correlation coefficient values is greater than 0.75
df.corr()
abs(df.corr()) > 0.75

# Finds which column pairs have a CC values > 0.75
df_mc = df.corr().abs().stack().reset_index().sort_values(0, ascending=False)

df_mc['pairs'] = list(zip(df_mc.level_0, df_mc.level_1))

df_mc.set_index(['pairs'], inplace = True)

df_mc.drop(columns=['level_1', 'level_0'], inplace = True)

# cc for correlation coefficient
df_mc.columns = ['cc']

df_mc.drop_duplicates(inplace=True)

df_mc[(df_mc.cc>.75) & (df_mc.cc<1)]
```

executed in 30ms, finished 00:50:49 2021-11-13

Out[18]:

pairs	cc
(total_day_charge, total_day_minutes)	1.000000
(total_eve_minutes, total_eve_charge)	1.000000
(total_night_charge, total_night_minutes)	0.999999
(total_intl_minutes, total_intl_charge)	0.999993
(voice_mail_plan, number_vmail_messages)	0.956927
(total_customer_charges, total_customer_minutes)	0.890804
(total_customer_charges, total_day_charge)	0.884757
(total_customer_charges, total_day_minutes)	0.884754

- Let's get rid of the factors related to minutes and keep the charge factors. Price is more important to our overall analysis
- We will also remove number_vmail_messages as the more important factor is that they have a voicemail plan.

In [19]:

```
# Dropping total_day_minutes, total_eve_minutes, total_night_minutes, total_i
df.drop(['total_day_minutes','total_eve_minutes','total_night_minutes','total
```

executed in 15ms, finished 00:50:49 2021-11-13

In [20]:
```
# Final Cleaned data
df.head()
```
executed in 15ms, finished 00:50:49 2021-11-13

Out[20]:

	state	account_length	international_plan	voice_mail_plan	total_day_calls	total_day_charge
0	KS	128	0	1	110	45.07
1	OH	107	0	1	123	27.47
2	NJ	137	0	0	114	41.38
3	OH	84	1	0	71	50.90
4	OK	75	1	0	113	28.34

1.6 Modeling

Now that we have explored the cleaned data, we can finally move on to create models to properly see the effects of each of the factors on telecom customer churning.

1.6.1 Model Training/Test data

In [21]:
```
# Create categories
df_dummy = pd.get_dummies(df)
df_dummy.head()
```
executed in 30ms, finished 00:50:49 2021-11-13

Out[21]:

	account_length	international_plan	voice_mail_plan	total_day_calls	total_day_charge	total_
0	128	0	1	110	45.07	
1	107	0	1	123	27.47	
2	137	0	0	114	41.38	
3	84	1	0	71	50.90	
4	75	1	0	113	28.34	

5 rows × 66 columns

In [22]:
```
# Create X,y and train/test
X = df_dummy.drop(columns=['churn'], axis=1)
y = df_dummy['churn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, r
```
executed in 15ms, finished 00:50:49 2021-11-13

1.6.2 Class Imbalance

In [23]:
```
df['churn'].value_counts()
```
executed in 11ms, finished 00:50:49 2021-11-13

Out[23]:
```
0    2850
1     483
Name: churn, dtype: int64
```

In [24]:
```
# SMOTE
smote = SMOTE()
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)
```
executed in 40ms, finished 00:50:49 2021-11-13

1.6.3 Logistic Regression

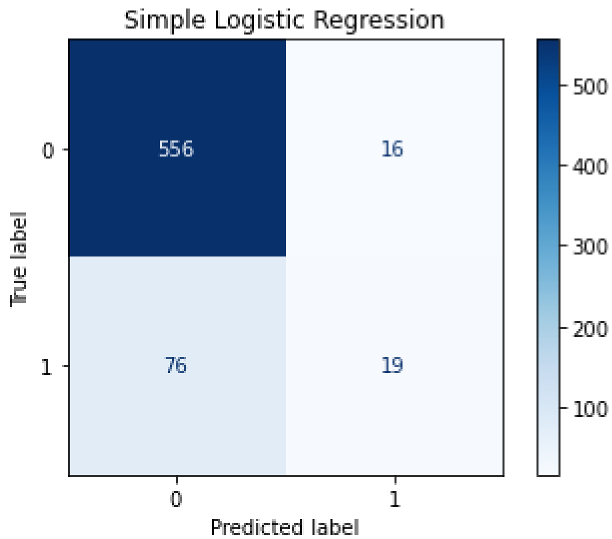
In [25]:
```
# Logistic Regression model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear')
model_log = logreg.fit(X_train_sm, y_train_sm)
```
executed in 74ms, finished 00:50:49 2021-11-13

In [26]:
```
# Prediction
y_hat_train = logreg.predict(X_train_sm)
y_hat_test = logreg.predict(X_test)
```
executed in 12ms, finished 00:50:49 2021-11-13

In [27]:
```
display(model_log.score(X_train_sm, y_train_sm))
display(model_log.score(X_test, y_test))
```
executed in 14ms, finished 00:50:49 2021-11-13

```
0.9220807726075505

0.8620689655172413
```

In [28]:
```
# Plot confusion matrix
plot_confusion_matrix(model_log, X_test, y_test,cmap=plt.cm.Blues)
plt.title('Simple Logistic Regression')
plt.show()
```
executed in 171ms, finished 00:50:49 2021-11-13



Type *Markdown* and LaTeX: α^2

In [29]:
```
# we compute our validation metric, recall

print('Training Precision: ', precision_score(y_train_sm, y_hat_train))
print('Testing Precision: ', precision_score(y_test, y_hat_test))
print('\n\n')

print('Training Recall: ', recall_score(y_train_sm, y_hat_train))
print('Testing Recall: ', recall_score(y_test, y_hat_test))
print('\n\n')

print('Training Accuracy: ', accuracy_score(y_train_sm, y_hat_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test))
print('\n\n')

print('Training F1-Score: ', f1_score(y_train_sm, y_hat_train))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test))
```
executed in 30ms, finished 00:50:49 2021-11-13

```
Training Precision:  0.9674282936315022
Testing Precision:  0.5428571428571428



Training Recall:  0.8735733099209834
Testing Recall:  0.2



Training Accuracy:  0.9220807726075505
Testing Accuracy:  0.8620689655172413



Training F1-Score:  0.9181084198385236
Testing F1-Score:  0.29230769230769227
```

- We want to focus on **Recall** as our main model metric because we want to focus on predicting whether or not a customer is going to churn
- Recall is 87% for Training and 20% for Test. The test score is extremely low. A new model classifier will be needed
- The training score is higher than the recall score, thus is **NOT Overfitting**.

1.6.4 Decision Tree

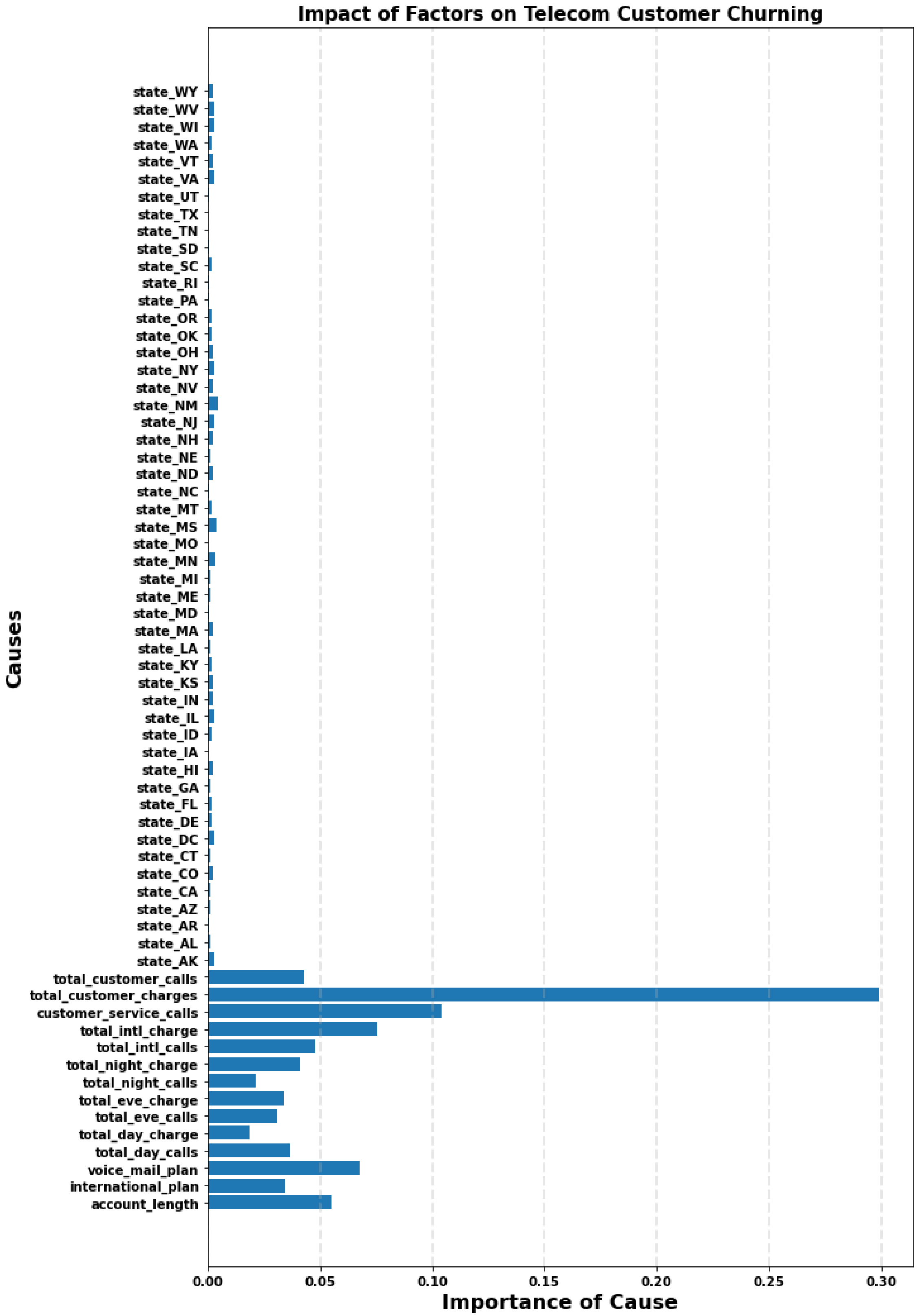
In [30]:
```
# Set up Decision Tree
tree_clf = DecisionTreeClassifier()
tree_clf.fit(X_train_sm, y_train_sm)
```
executed in 61ms, finished 00:50:49 2021-11-13

Out[30]:
```
DecisionTreeClassifier()
```

In [31]:

```
# Plotting feature importance of models
def plot_feature_importances(model):
    n_features = X_train_sm.shape[1]
    plt.figure(figsize=(10,18))
    plt.barh(range(n_features), model.feature_importances_)
    plt.xticks(fontsize=10, fontweight='bold')
    plt.yticks(np.arange(n_features), X_train_sm.columns.values, fontsize=10,
    plt.xlabel('Importance of Cause', fontsize=16, fontweight='bold' )
    plt.ylabel('Causes',fontsize=16, fontweight='bold')
    plt.grid(linestyle='--', linewidth=2, axis='x', alpha=0.3)
    plt.title("Impact of Factors on Telecom Customer Churning", fontsize=15,

plot_feature_importances(tree_clf)
```

executed in 1.81s, finished 00:50:51 2021-11-13


Impact of Factors on Telecom Customer Churning
Causes
Importance of Cause
state_WY
state_WV
state_WI
state_WA
state_VT
state_VA
state_UT
state_TX
state_TN
state_SD
state_SC
state_RI
state_PA
state_OR
state_OK
state_OH
state_NY
state_NV
state_NM
state_NJ
state_NH
state_NE
state_ND
state_NC
state_MT
state_MS
state_MO
state_MN
state_MI
state_ME
state_MD
state_MA
state_LA
state_KY
state_KS
state_IN
state_IL
state_ID
state_IA
state_HI
state_GA
state_FL
state_DE
state_DC
state_CT
state_CO
state_CA
state_AZ
state_AR
state_AL
state_AK
total_customer_calls
total_customer_charges
customer_service_calls
total_intl_charge
total_intl_calls
total_night_charge
total_night_calls
total_eve_charge
total_eve_calls
total_day_charge
total_day_calls
voice_mail_plan
international_plan
account_length
0.00
0.05
0.10
0.15
0.20
0.25
0.30

In [32]:

```
# Test set predictions
val = tree_clf.predict(X_train_sm)
pred = tree_clf.predict(X_test)

# Confusion matrix and classification report
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print('Training Recall: ', recall_score(y_train_sm, val)*100)
print('Testing Recall: ', recall_score(y_test, pred)*100)
```

executed in 30ms, finished 00:50:51 2021-11-13

```
[[499  73]
 [ 12  83]]
              precision    recall  f1-score   support

           0       0.98      0.87      0.92       572
           1       0.53      0.87      0.66        95

    accuracy                           0.87       667
   macro avg       0.75      0.87      0.79       667
weighted avg       0.91      0.87      0.88       667

Training Recall:  100.0
Testing Recall:  87.36842105263159
```

- Not Overfitting

In [33]:

```
# Finding the best parameters
param_grid = {
    'max_depth': [2, 5, 10, 25, 50],
    'min_samples_split': [2, 5, 10, 20]
}

gs_trees = GridSearchCV(tree_clf, param_grid, cv=3, scoring='recall')
gs_trees.fit(X_train_sm, np.ravel(y_train_sm))

gs_trees.best_params_

best_parameters = gs_trees.best_params_

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(best_parameters.keys()):
    print('%s: %r' % (param_name, best_parameters[param_name]))

val = gs_trees.predict(X_train_sm)
pred = gs_trees.predict(X_test)

# Confusion matrix and classification report
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print('Training Recall: ', recall_score(y_train_sm, val)*100)
print('Testing Recall: ', recall_score(y_test, pred)*100)
```

executed in 1.51s, finished 00:50:53 2021-11-13

```
Grid Search found the following optimal parameters:
max_depth: 25
min_samples_split: 2
[[478  94]
 [ 12  83]]
              precision    recall  f1-score   support

           0       0.98      0.84      0.90       572
           1       0.47      0.87      0.61        95

    accuracy                           0.84       667
   macro avg       0.72      0.85      0.76       667
weighted avg       0.90      0.84      0.86       667

Training Recall:  100.0
Testing Recall:  87.36842105263159
```

1.6.5 Random Forest

In [34]:

```
# Random Forest
forest = RandomForestClassifier()
forest.fit(X_train_sm, np.ravel(y_train_sm))
```

executed in 669ms, finished 00:50:53 2021-11-13

Out[34]:

```
RandomForestClassifier()
```

In [35]:
```
# Finding the best parameters
param_grid = {
    'max_depth': [2, 5, 10, 25],
    'min_samples_split': [2, 5, 10, 20]
}

gs_tree = GridSearchCV(forest, param_grid, cv=3, scoring='recall')
gs_tree.fit(X_train_sm, np.ravel(y_train_sm))

gs_tree.best_params_
```
executed in 15.3s, finished 00:51:09 2021-11-13

Out[35]:
```
{'max_depth': 25, 'min_samples_split': 2}
```

In [36]:
```
# Applying best Random Forest parameters
forest = RandomForestClassifier(max_depth=25, min_samples_split=2)
forest.fit(X_train_sm, np.ravel(y_train_sm))

vals = forest.predict(X_train_sm)
preds = forest.predict(X_test)

print(confusion_matrix(y_test, preds))
print(classification_report(y_test, preds))
print('Training Recall: ', recall_score(y_train_sm, vals)*100)
print('Testing Recall: ', recall_score(y_test, preds)*100)
```
executed in 733ms, finished 00:51:09 2021-11-13

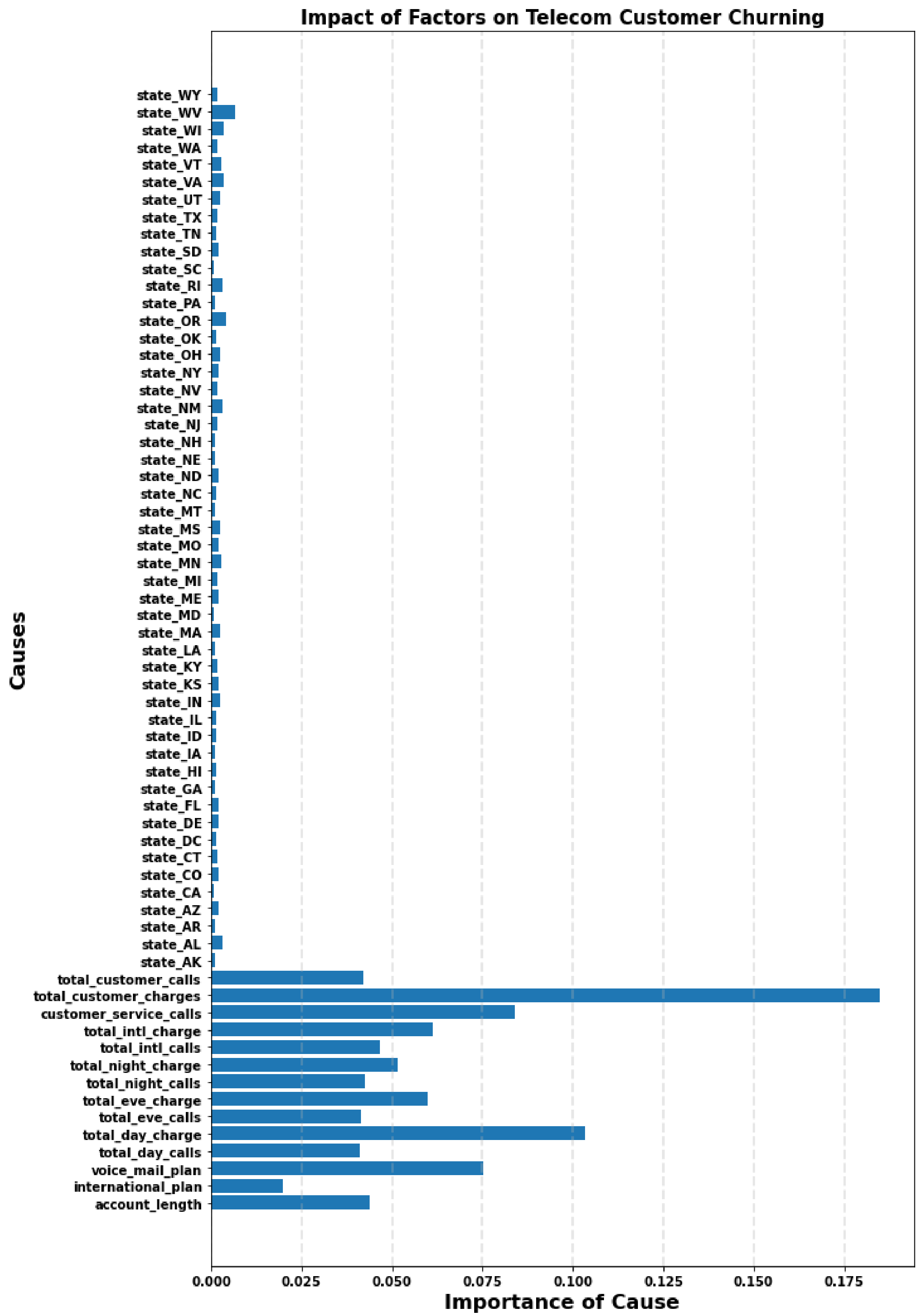
```
[[550  22]
 [ 20  75]]
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       572
           1       0.77      0.79      0.78        95

    accuracy                           0.94       667
   macro avg       0.87      0.88      0.87       667
weighted avg       0.94      0.94      0.94       667

Training Recall:  100.0
Testing Recall:  78.94736842105263
```

- Our Confusion matrix is looking better than before. Predicting more True churns than False churns
- Training Recall score is higher than test. **Not Overfitting**
- Testing Recall score is still low (76%).
- Should keep looking for a better Classifier

In [37]:

```
plot_feature_importances(forest)
```

executed in 1.86s, finished 00:51:11 2021-11-13



- The individual states do not seem to have a large influence on the customer churn
- **Total customer charges_** and **total day charge** appears to have the largest influence on churn

1.6.6 XG Boost

In [38]:

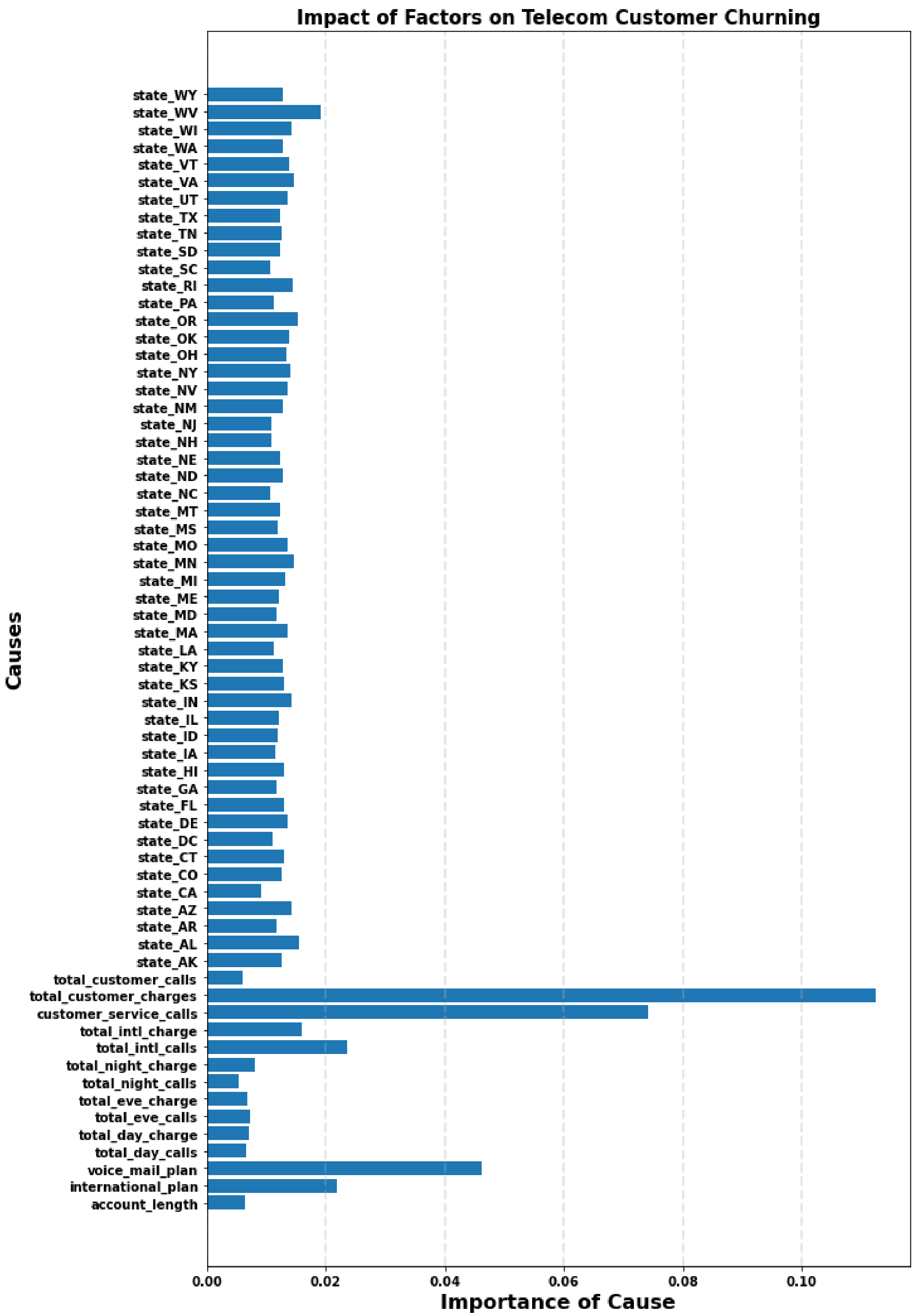
```
# Instantiate XGBClassifier
xgb = XGBClassifier()

# Fit XGBClassifier
xgb.fit(X_train_sm, np.ravel(y_train_sm))

# Predict on training and test sets
training_preds = xgb.predict(X_train_sm)
test_preds = xgb.predict(X_test)

# Accuracy of training and test sets
training_recall = recall_score(y_train_sm, training_preds)
test_recall = recall_score(y_test, test_preds)

print('Training Recall: {:.4}%'.format(training_recall * 100))
print('Test Recall: {:.4}%'.format(test_recall * 100))
```

executed in 1.20s, finished 00:51:12 2021-11-13

```
Training Recall: 99.03%
Test Recall: 86.32%
```

In [39]:
```
plot_feature_importances(xgb)
```

executed in 1.80s, finished 00:51:14 2021-11-13



In [40]:

```
param_grid = {
    'learning_rate': [0.1, 0.2],
    'max_depth': [6],
    'min_child_weight': [1, 2],
    'subsample': [0.5, 0.7],
    'n_estimators': [100],
}

grid_clf = GridSearchCV(xgb, param_grid, scoring='recall')
grid_clf.fit(X_train_sm, np.ravel(y_train_sm))

best_parameters = grid_clf.best_params_

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(best_parameters.keys()):
    print('%s: %r' % (param_name, best_parameters[param_name]))

training_preds = grid_clf.predict(X_train_sm)
test_preds = grid_clf.predict(X_test)

training_recall = recall_score(y_train_sm, training_preds)
test_recall = recall_score(y_test, test_preds)

print('')
print('Training Recall: {:.4}%'.format(training_recall * 100))
print('Test Recall: {:.4}%'.format(test_recall * 100))
```

executed in 10.5s, finished 00:51:25 2021-11-13

```
Grid Search found the following optimal parameters:
learning_rate: 0.2
max_depth: 6
min_child_weight: 1
n_estimators: 100
subsample: 0.7

Training Recall: 97.41%
Test Recall: 84.21%
```

In [41]:
```
print(confusion_matrix(y_test, test_preds))
print(classification_report(y_test, test_preds))
print('')
print('Training Recall: {:.4}%'.format(training_recall * 100))
print('Test Recall: {:.4}%'.format(test_recall * 100))
```
executed in 30ms, finished 00:51:25 2021-11-13

```
[[560  12]
 [ 15  80]]
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       572
           1       0.87      0.84      0.86        95

    accuracy                           0.96       667
   macro avg       0.92      0.91      0.92       667
weighted avg       0.96      0.96      0.96       667


Training Recall: 97.41%
Test Recall: 84.21%
```

- Confusion Matrix shows a great True Churn prediction rate compared to False Churn.
- Higher Training recall than Test recall. **Not Overfitting**
- **XG Boost has the highest testing Recall from the previous models.**
- We shall use this for our predictive analysis

1.6.7 Final Model Selection

In [42]:
```
df_final = pd.DataFrame(xgb.feature_importances_, X.columns, columns = ['coef
df_final.head()
```
executed in 60ms, finished 00:51:25 2021-11-13

Out[42]:

	coefficient
account_length	0.006312
international_plan	0.021781
voice_mail_plan	0.046166
total_day_calls	0.006678
total_day_charge	0.007045

In [43]:
```
df_final_detail = df_final[:14]
display(df_final_detail.head())
df_final_state = df_final[14:]
display(df_final_state.head())
```
executed in 15ms, finished 00:51:25 2021-11-13

	coefficient
account_length	0.006312
international_plan	0.021781
voice_mail_plan	0.046166
total_day_calls	0.006678
total_day_charge	0.007045

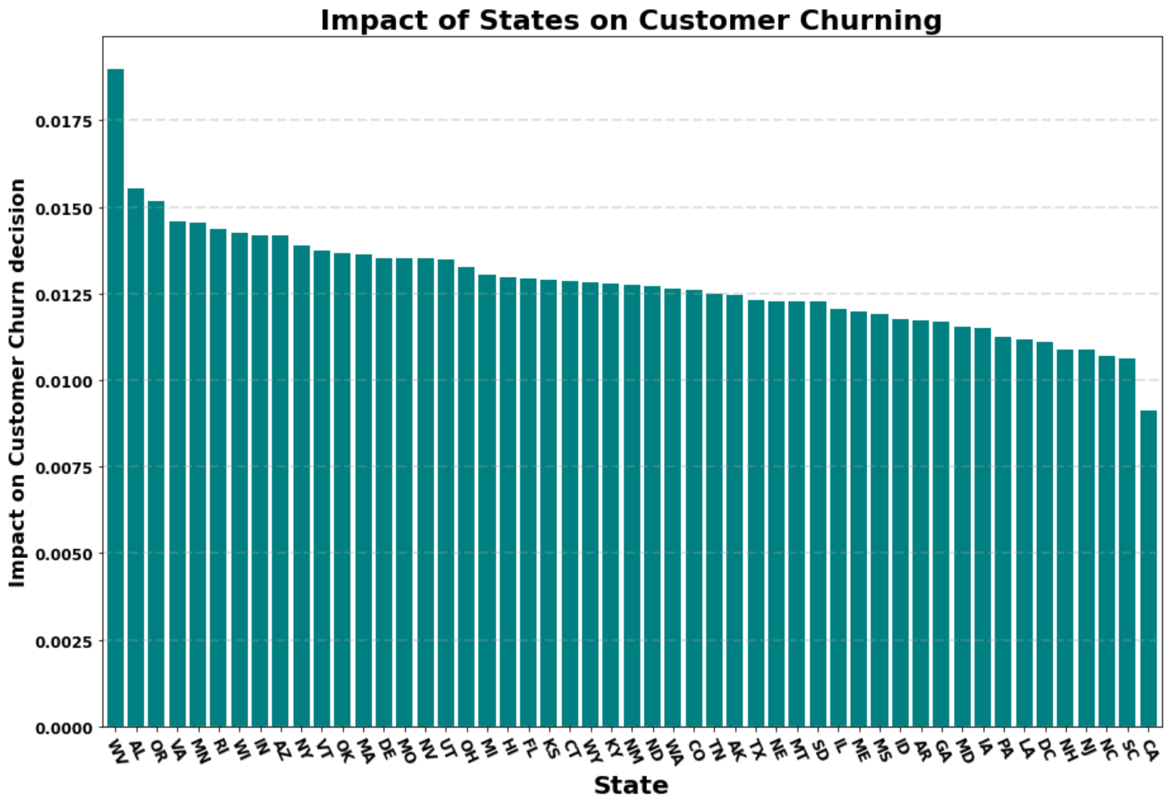
	coefficient
state_AK	0.012476
state_AL	0.015536
state_AR	0.011731
state_AZ	0.014177
state_CA	0.009130

In [44]:
```
# Renaming states indexes
as_list = df_final_state.index.values.tolist()
as_list = [i.replace('state_','') for i in as_list]
df_final_state.index = as_list
df_final_state.head()
```
executed in 14ms, finished 00:51:25 2021-11-13

Out[44]:

	coefficient
AK	0.012476
AL	0.015536
AR	0.011731
AZ	0.014177
CA	0.009130

In [45]:
```
# Create a barplot of state's impact on churn
ax = df_final_state.sort_values(by=['coefficient'], ascending=False).plot(kir
ax.set_title("Impact of States on Customer Churning", fontsize = 22, fontweig
ax.set_xlabel("State", fontsize=20, fontweight='bold')
ax.set_ylabel("Impact on Customer Churn decision", fontsize=16, fontweight='b
ax.get_legend().remove()

plt.xticks(rotation=-65, fontsize=12, fontweight='bold')
plt.yticks(fontsize=12, fontweight='bold')
plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='y', alpha=0.3)
```
executed in 1.26s, finished 00:51:26 2021-11-13

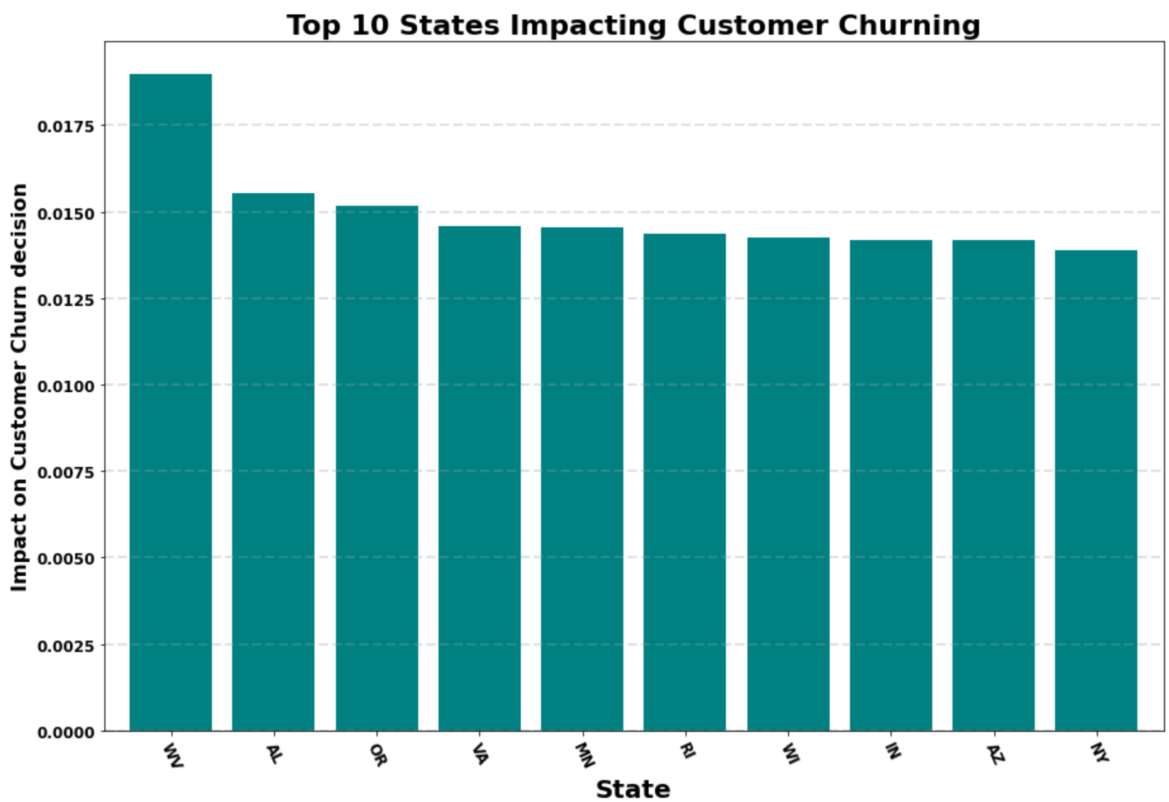


- There seems to be a slightly larger churn rate in **state_WV**. This may need to be investigated further.
- No apparent correlation can be found with states with higher impact

In [46]:

```
# Create a barplot of top 5 state's impact on churn
ax = df_final_state.sort_values(by=['coefficient'], ascending=False).head(10)
ax.set_title("Top 10 States Impacting Customer Churning", fontsize = 22, font
ax.set_xlabel("State", fontsize=20, fontweight='bold')
ax.set_ylabel("Impact on Customer Churn decision", fontsize=16, fontweight='b
ax.get_legend().remove()

plt.xticks(rotation=-65, fontsize=12, fontweight='bold')
plt.yticks(fontsize=12, fontweight='bold')
plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='y', alpha=0.3)
```

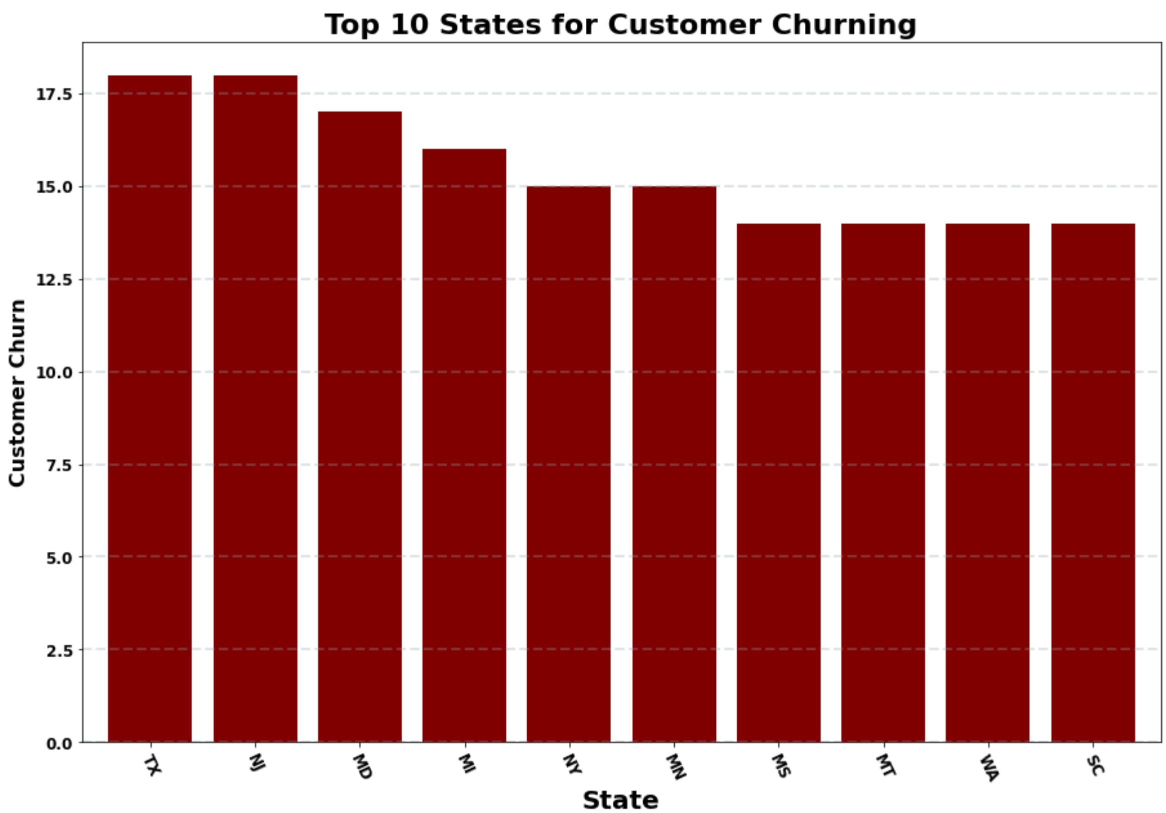
executed in 234ms, finished 00:51:26 2021-11-13



In [47]:

```
# Create a barplot of top 5 state's churn based on original dataset
ax = df.groupby(['state']).sum().churn.sort_values(ascending=False).head(10).
ax.set_title("Top 10 States for Customer Churning", fontsize = 22, fontweight
ax.set_xlabel("State", fontsize=20, fontweight='bold')
ax.set_ylabel("Customer Churn", fontsize=16, fontweight='bold');

plt.xticks(rotation=-65, fontsize=12, fontweight='bold')
plt.yticks(fontsize=12, fontweight='bold')
plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='y', alpha=0.3)
```

executed in 219ms, finished 00:51:27 2021-11-13



- Original Data set results for top churn and model feature importance appear to be showing different results
- Will need further investigation on individual state's impact on churn

In [48]:

```
# Renaming detail indexes
detail_list = df_final_detail.index.values.tolist()
detail_list = [i.replace('_',' ') for i in detail_list]
df_final_detail.index = detail_list
df_final_detail.head()
```

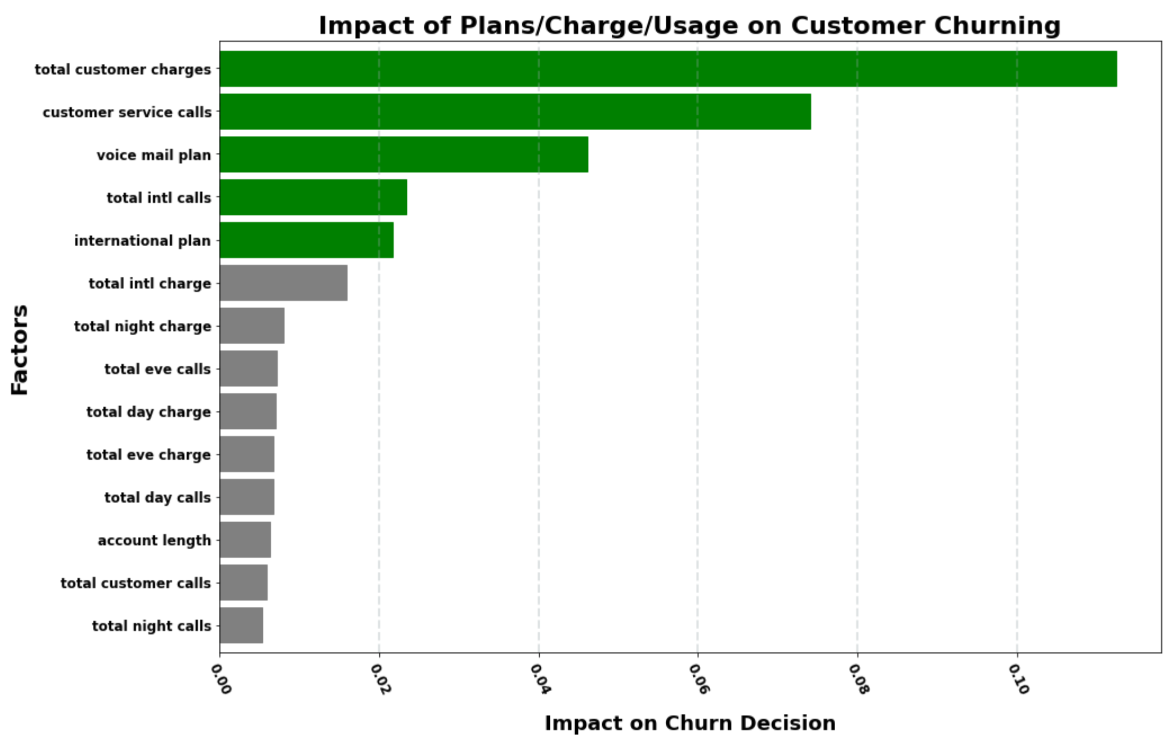
executed in 14ms, finished 00:51:27 2021-11-13

Out[48]:

	coefficient
account length	0.006312
international plan	0.021781
voice mail plan	0.046166
total day calls	0.006678
total day charge	0.007045

In [49]:
```
# Create a barplot of customer plans/usage's impact on churn
ax = df_final_detail.sort_values(by=['coefficient'], ascending=True).plot(kir
ax.set_title("Impact of Plans/Charge/Usage on Customer Churning", fontsize =
ax.set_ylabel("Factors", fontsize=20, fontweight='bold')
ax.set_xlabel("Impact on Churn Decision", fontsize=18, fontweight='bold', lab
ax.get_legend().remove()

for column in ax.patches:
    if column.get_width() > 0.02:
        column.set_color('green')
    else:
        column.set_color('grey')

plt.xticks(rotation=-65, fontsize=12, fontweight='bold')
plt.yticks(fontsize=12, fontweight='bold')
plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='x', alpha=0.3)
```

executed in 480ms, finished 00:51:27 2021-11-13



- **total_customer_charges** appear to be the largest cause of churn. This makes sense seeing that most customers
- **customer_service_calls** come 2nd influence to the plans. There may need to be an improvement to the customer service section of the company.
- **voice_mail_plan** and **international_plan** are the relatively high factors to customer churn. There may be some issues with how both plans are being handled.

1.7 Conclusions

The analysis of the SyriaTel customer churn dataset resulted in the following conclusions:

- There needs to be an improvement to the **Voice mail plan** and **International plan**. The customer churn is heavily affected by the effectiveness of the plans. These plans need to be further examined to entice customers to stay with SyriaTel.
- The **Customer service** department may need examining. We need to check staffing to see what is causing the customers to turn away from SyriaTel as they submit for help through the customer service line.
- We can predict future customer churn with our final model. This should help us mitigate customer losses if we contact the customer earlier for their input.

1.8 Recommendations

What can we do right now?

- **We can adjust the minute-to-charge rates for day-evening-night.** The high total charges appear to be causing the most impact on SyriaTel customer's churning. Despite most of our customers utilizing around 2/3 of their total minutes in the evening and night, the day-time charges overtake the two charges combined. We may be able to get more customer satisfaction from them knowing that their charges in the day-time are being lowered, despite most of their minutes are spent in the evening. This will allow us to maintain profits from call charges while maintaining a lower churn.
- **Increase the staffing of the Customer Service department.** Increased staffing will readily make available more customers to be directly in line with staff. This will increase customer satisfaction. Additionally, more staffing may help us identify the common issues customers are having more quickly, allowing SyriaTel to address specific issues quickly as well. Overall, customer churn should go down.
- **Reduce the charge of international calls/plan** This will reduce customer churn over the factors of the international plan and its charges. It is not worth losing a customer over a specific plan they have chosen. We should keep the customer to profit off of the general uses outside of the international plan.

1.9 Next Steps

Further analysis of the SyriaTel data could yield additional insights to other recommendations

- **Locate what factors are causing a larger churn impact within specific states**
- **Create an alert system that detects when individual customer are in range of possibly churning**
- **Investigate a change over system from international to domestic plans on customer churn**