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. We are trying to predict the likelihood of a customer churn based on a user's communication usage, plans, and other related factors. SyriaTel can use these findings to improve the services to better keep customers and maintain greater profit for the company.

1.2 Data Understanding

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

In [1]:
```
# Import necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from statsmodels.formula.api import ols
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
import scipy.stats as stats
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.metrics import mean_absolute_error

import xgboost as xgb

from sklearn.metrics import roc_curve, auc

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_sco
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB,Complement
from sklearn.feature_selection import SelectKBest
import joblib

from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_sco
from sklearn.metrics import confusion_matrix,recall_score,precision_recall_curve
from sklearn.metrics import precision_recall_fscore_support,f1_score,fbeta_score
from sklearn.metrics import classification_report, plot_roc_curve, plot_confusior
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score, confusion_matrix, classification_repo
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier

from xgboost import XGBClassifier
```

executed in 1.97s, finished 00:56:16 2021-04-23

In [2]:
```
df = pd.read_csv('data/telecoms.csv')
df.head()
```
executed in 47ms, finished 00:56:16 2021-04-23

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

5 rows × 21 columns

1.3 Data Preparation

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

In [3]:
```
display(df.info())
display(df.describe())
```

executed in 77ms, finished 00:56:16 2021-04-23

```
<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	total eve minutes
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000

In [4]:
```
# Adjust column names with '_'

df.columns = df.columns.str.replace(' ','_')
df.info()
```

executed in 12ms, finished 00:56:16 2021-04-23

```
<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 [5]:
```
# Find missing values columns
df.isna().sum()
```
executed in 15ms, finished 00:56:16 2021-04-23

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
```

- No missing values found.

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

In [6]:
```
# Drop phone_number
df.drop('phone_number', axis=1, inplace=True)
```
executed in 14ms, finished 00:56:16 2021-04-23

In [7]:
```
# Values for area_code

df.area_code.value_counts()
```
executed in 15ms, finished 00:56:16 2021-04-23

Out[7]:
```
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 [8]:
```
# Drop area_code column

df.drop('area_code', axis = 1, inplace=True)
```
executed in 14ms, finished 00:56:16 2021-04-23

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

In [9]:
```
# Change'churn' from bool to int

df['churn'] *= 1

# Change plans to int

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

df[['churn', 'international_plan', 'voice_mail_plan']].astype(int)
```
executed in 29ms, finished 00:56:16 2021-04-23

Out[9]:

	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 [10]:
```
df.head()
```
executed in 29ms, finished 00:56:16 2021-04-23

Out[10]:

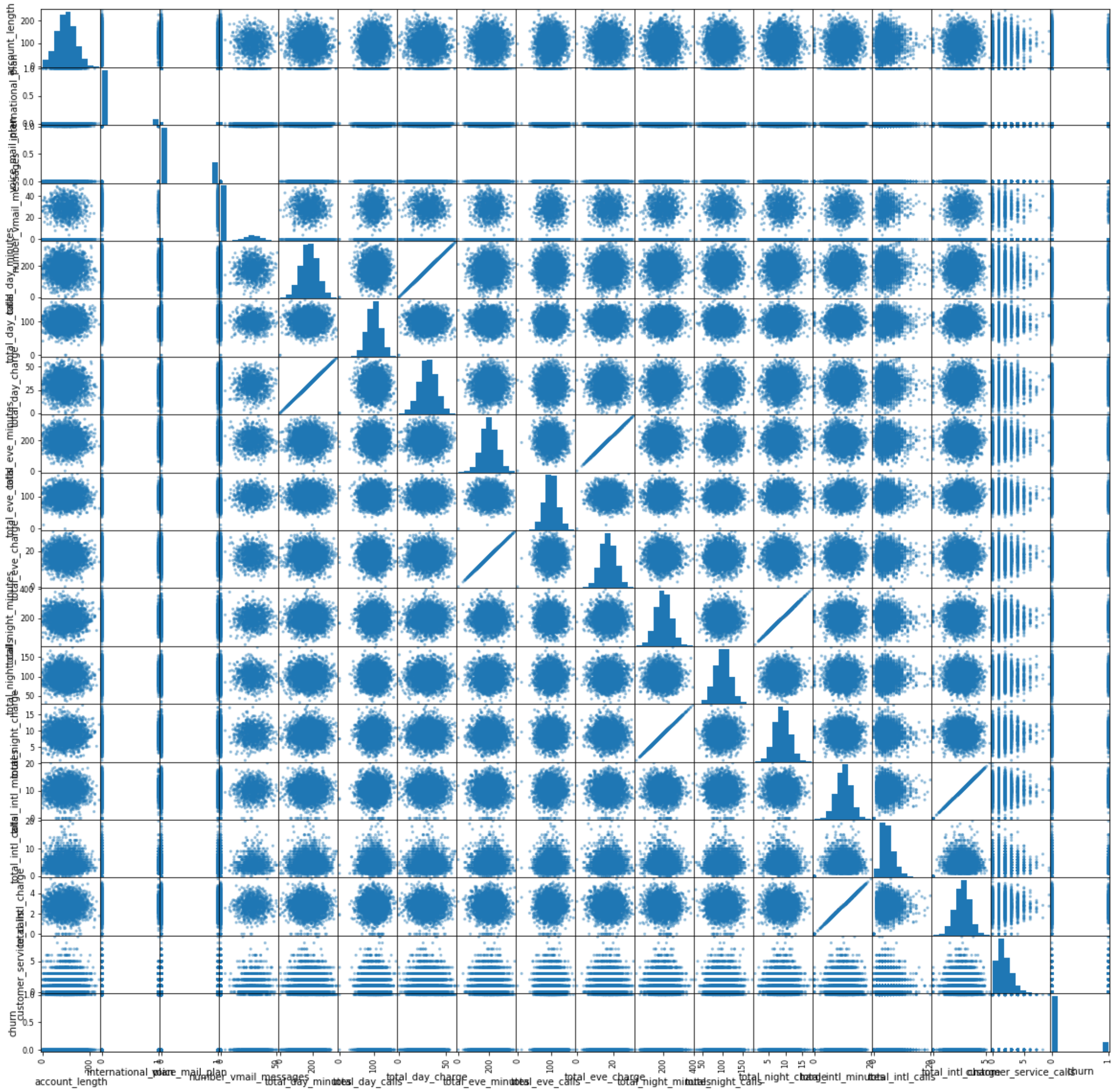
	state	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_mi
0	KS	128	0	1	25	
1	OH	107	0	1	26	
2	NJ	137	0	0	0	
3	OH	84	1	0	0	
4	OK	75	1	0	0	

1.4 Exploratory Data Analysis

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

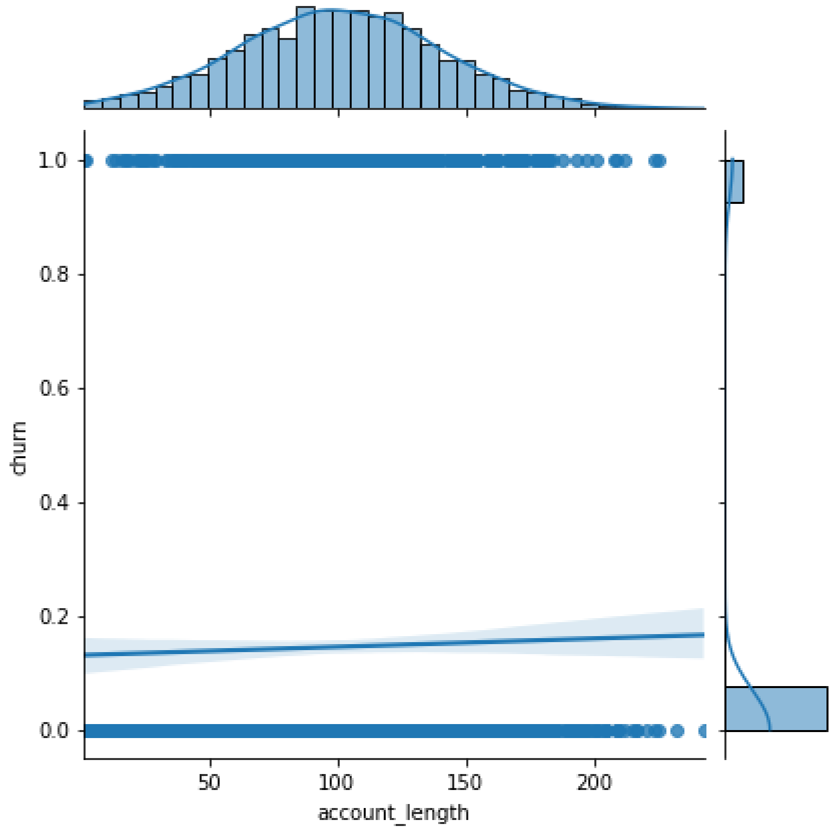
In [11]:
```
# we plot a scatter matrix to inspect distributions of predictors

pd.plotting.scatter_matrix(df, figsize=[20,20]);
plt.show()
```

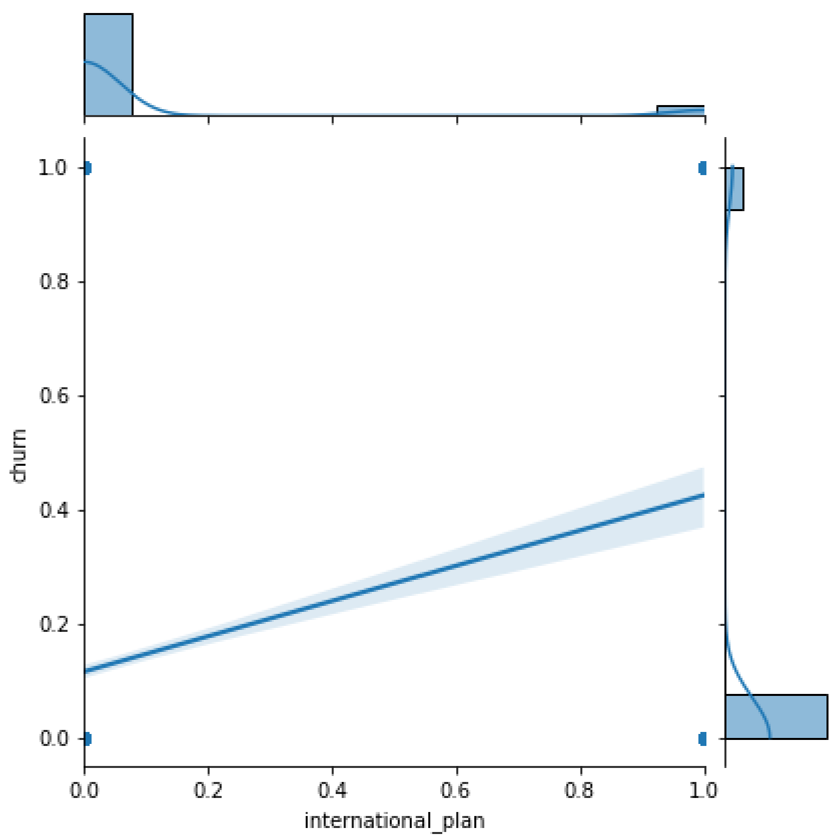
executed in 16.2s, finished 00:56:33 2021-04-23

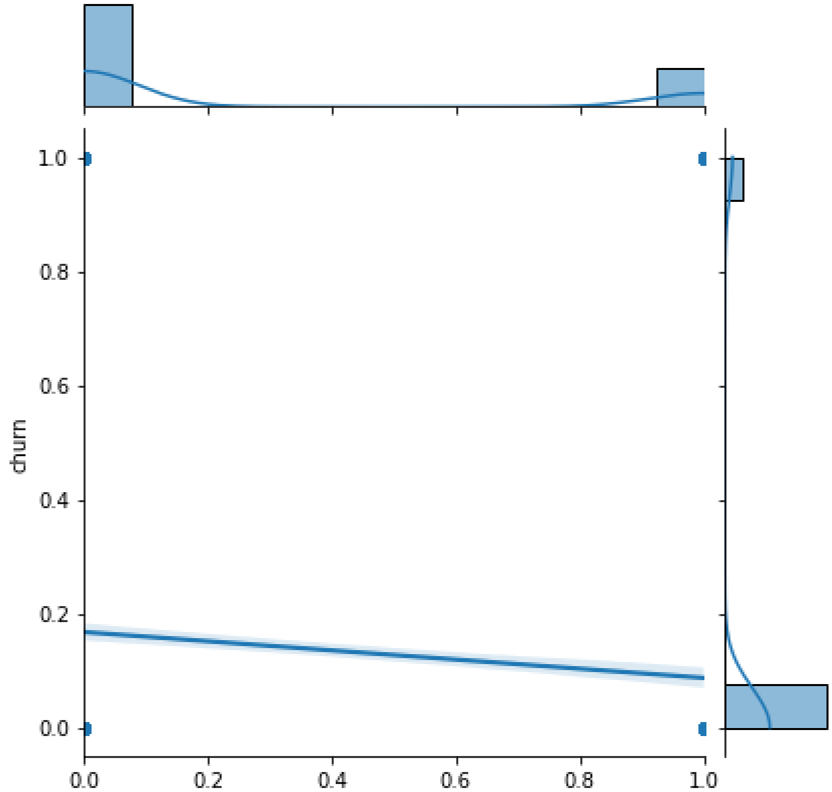


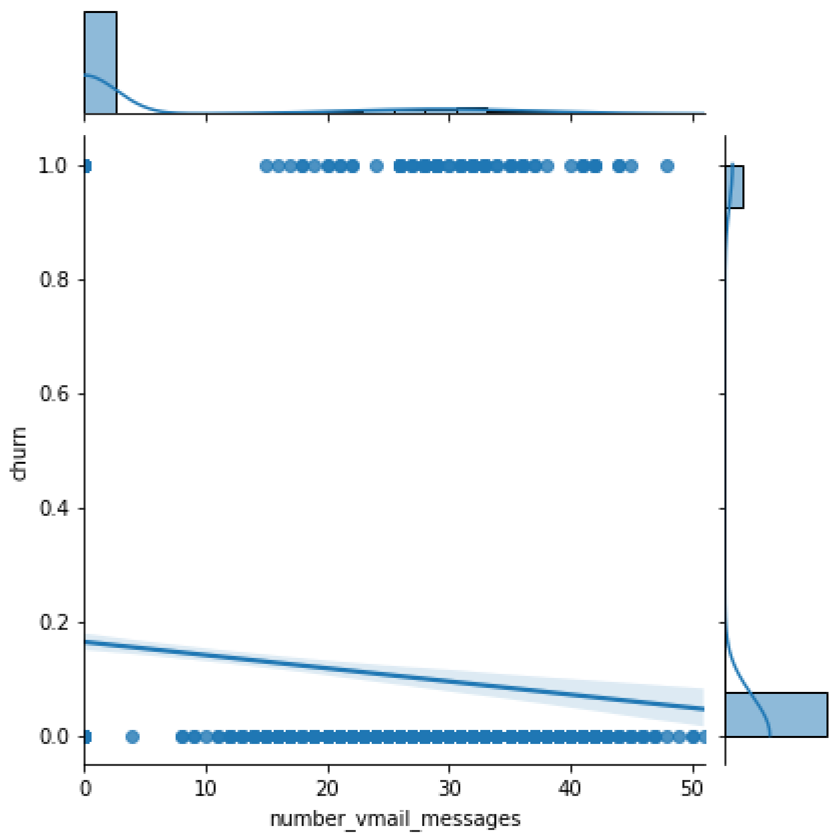
In [12]:
```
for col in df.columns[1:]:
    sns.jointplot(x=col, y='churn', data=df, kind='reg');
```

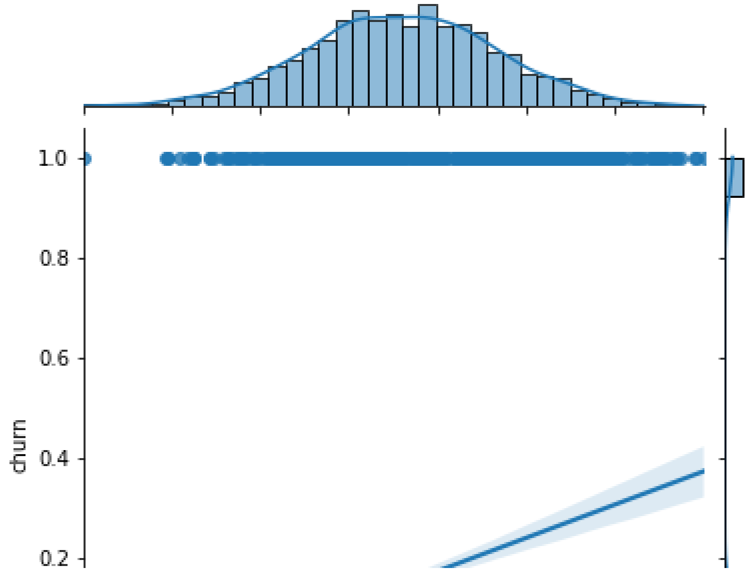
executed in 11.6s, finished 00:56:44 2021-04-23

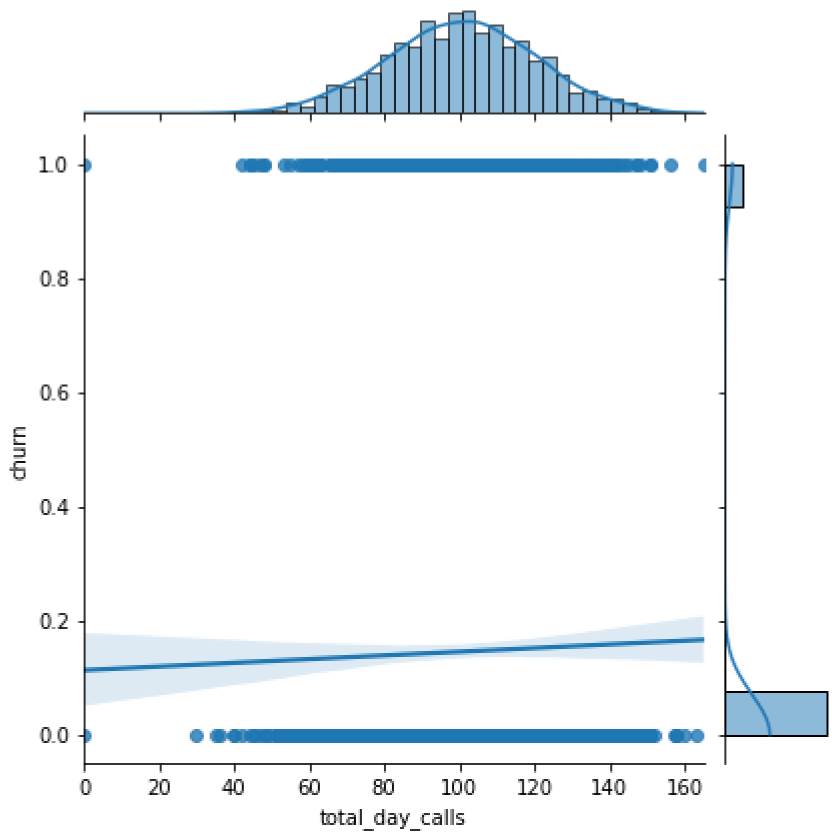


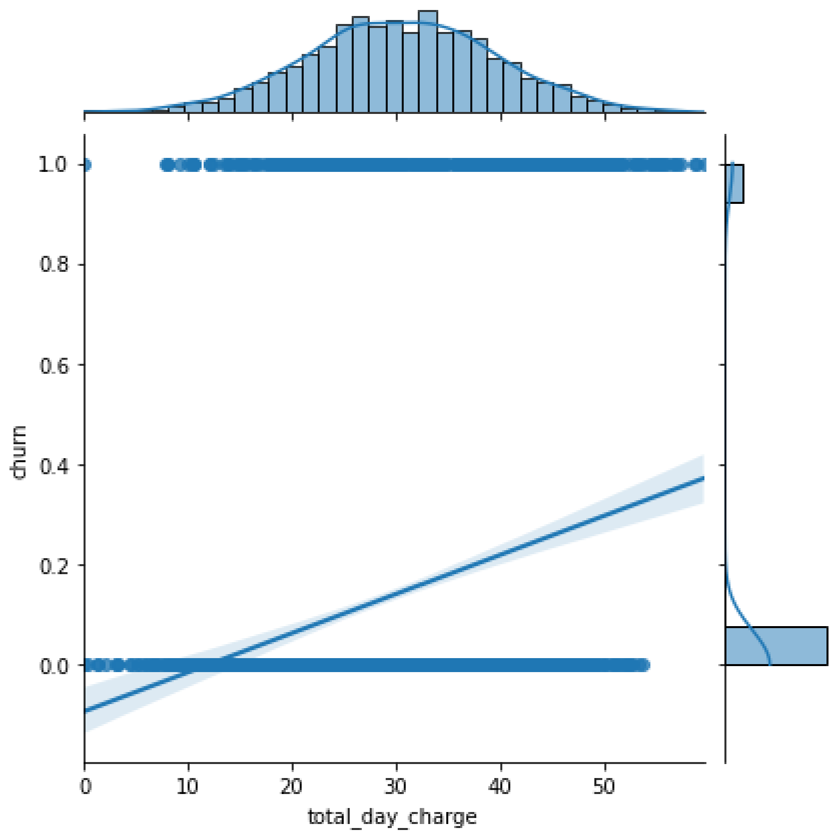


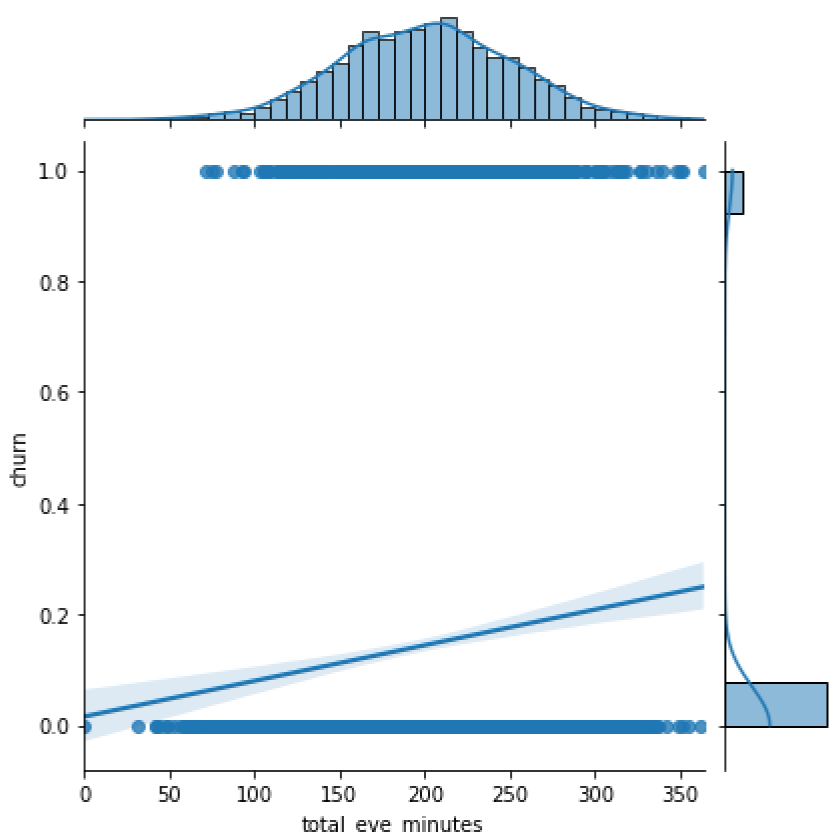

churn
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

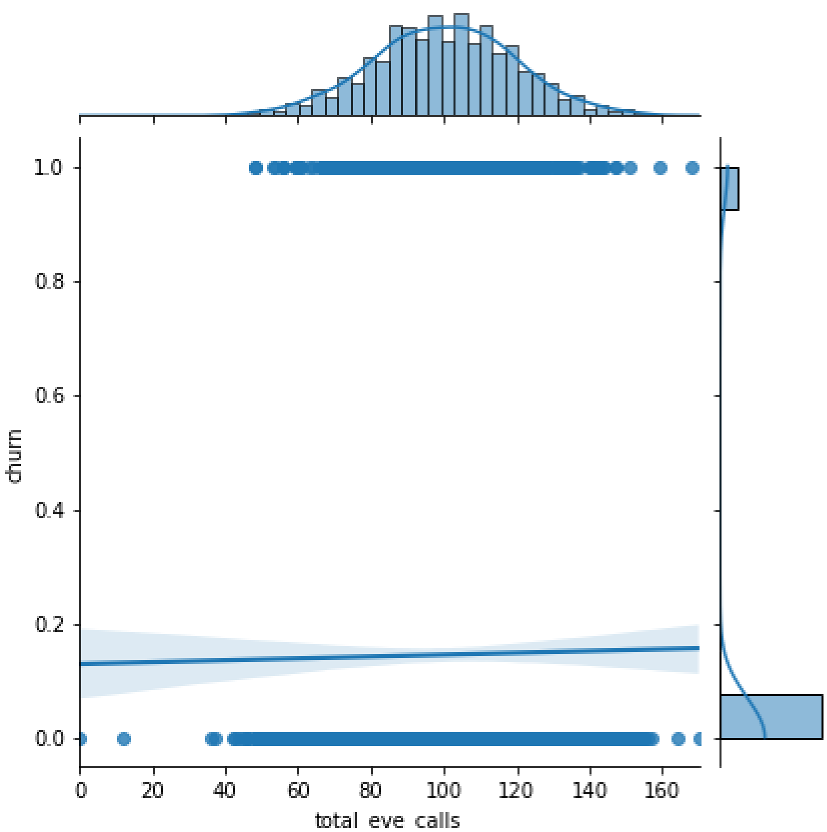

churn
number_vmail_messages
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

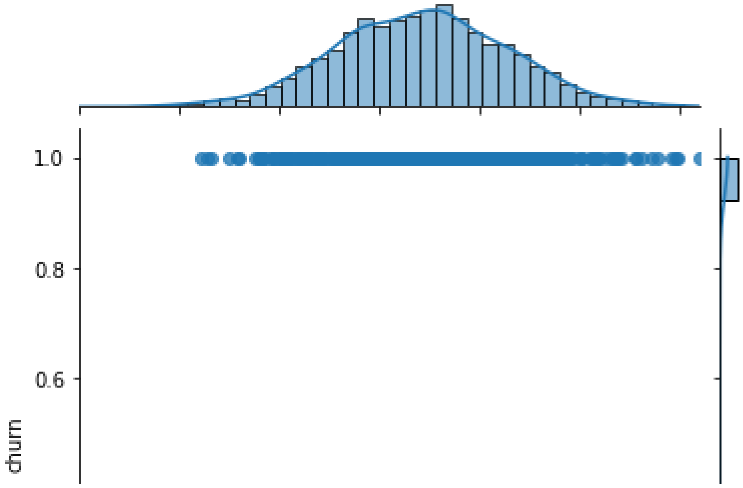

churn
1.0
0.8
0.6
0.4
0.2

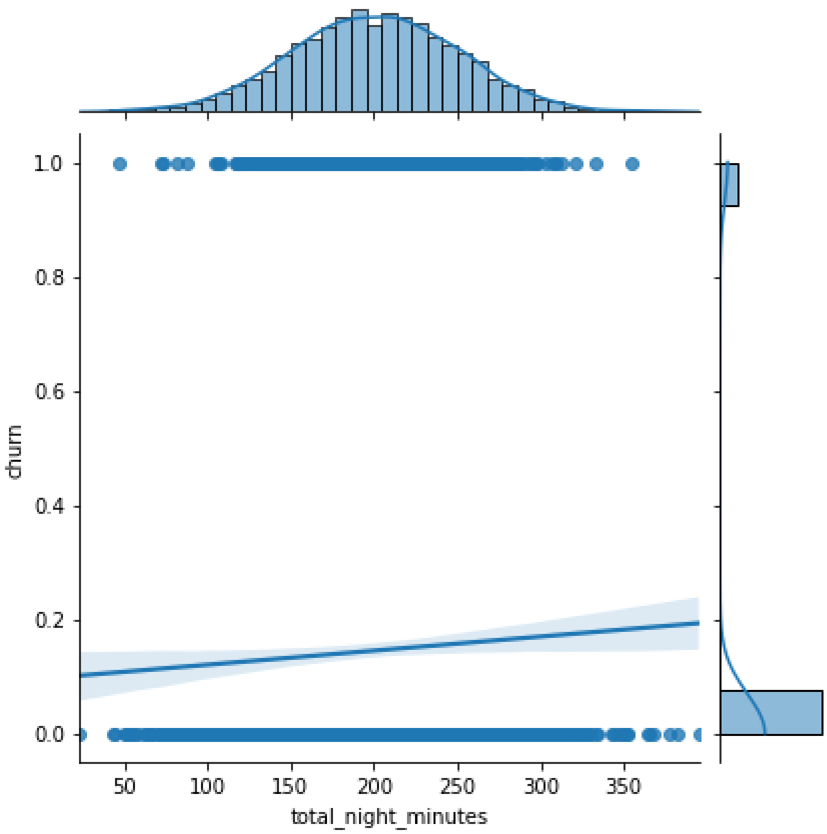

churn
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
140
160
total_day_calls

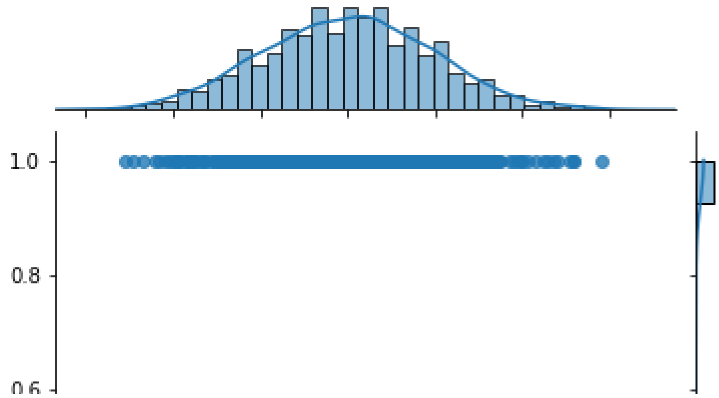

1.0
0.8
0.6
0.4
0.2
0.0
churn
0
10
20
30
40
50
total_day_charge

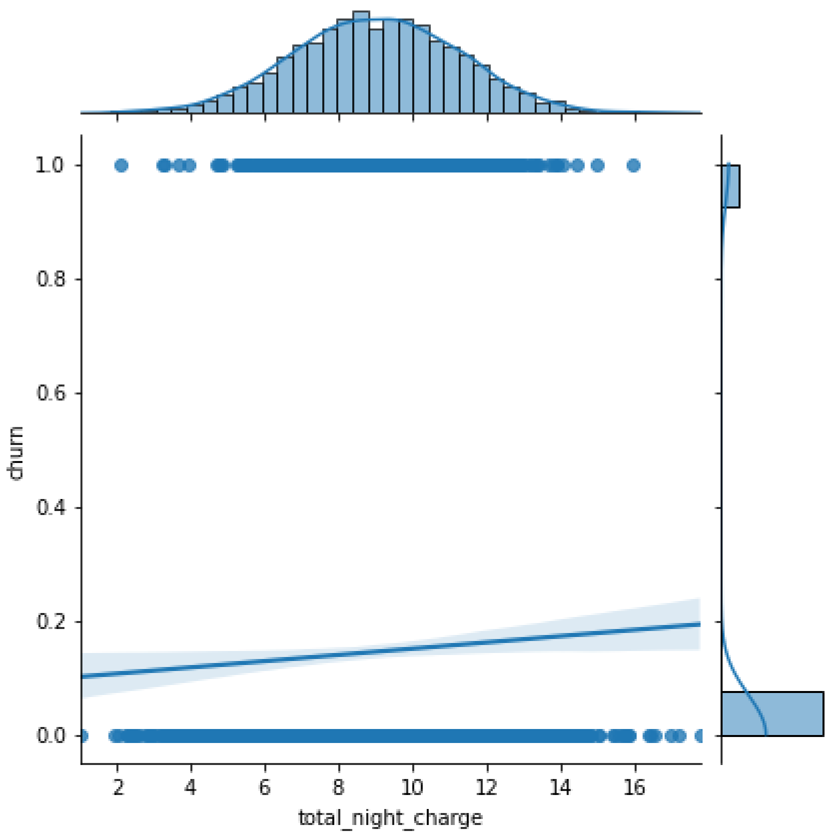

1.0
0.8
0.6
0.4
0.2
0.0
churn
0
50
100
150
200
250
300
350
total_eve_minutes

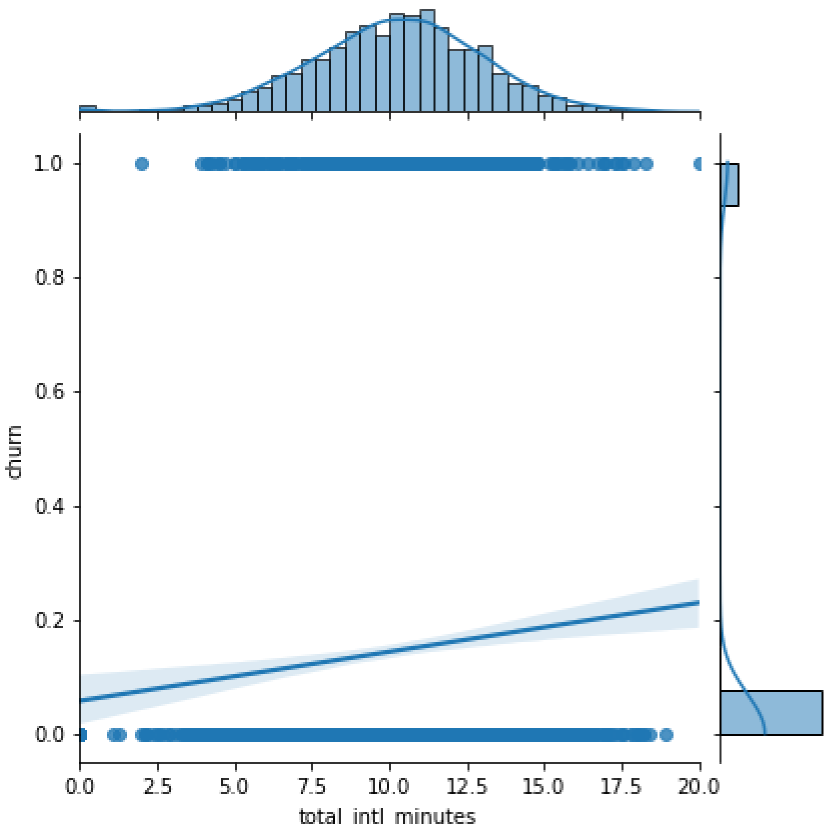

churn
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
140
160
total_eve_calls

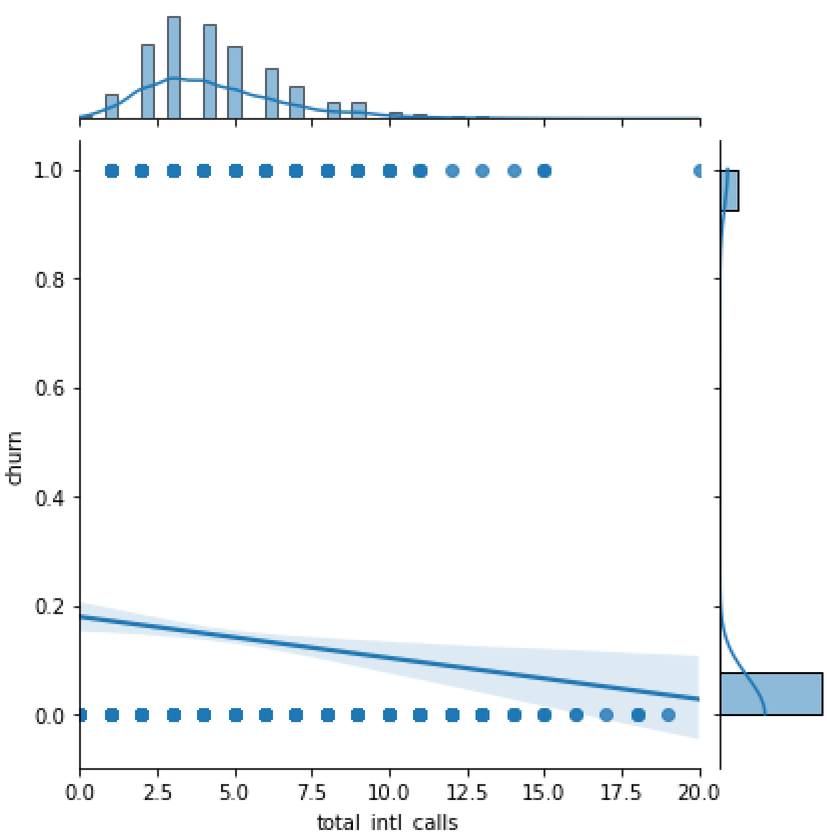

1.0
0.8
0.6
churn

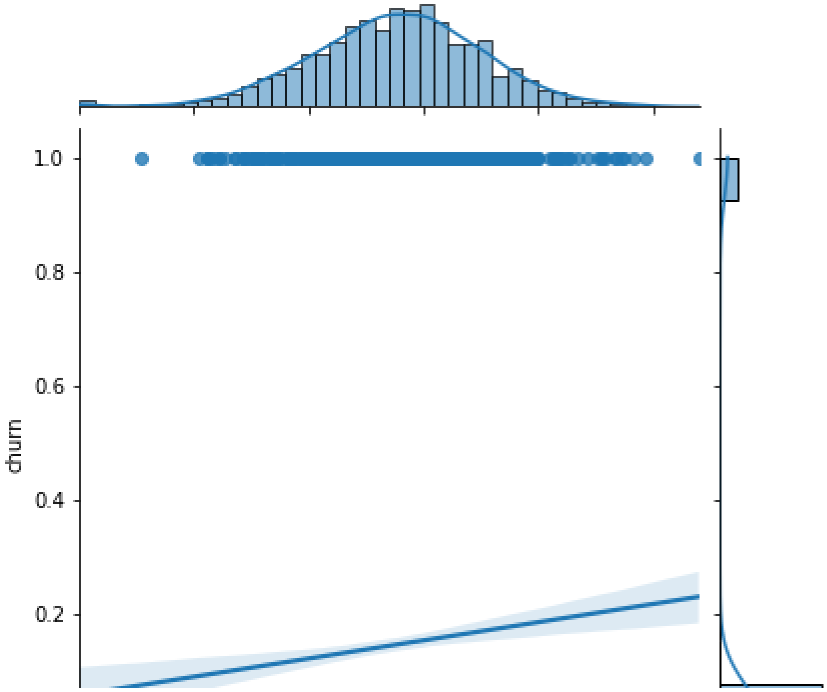

1.0
0.8
0.6
0.4
0.2
0.0
churn
50
100
150
200
250
300
350
total_night_minutes

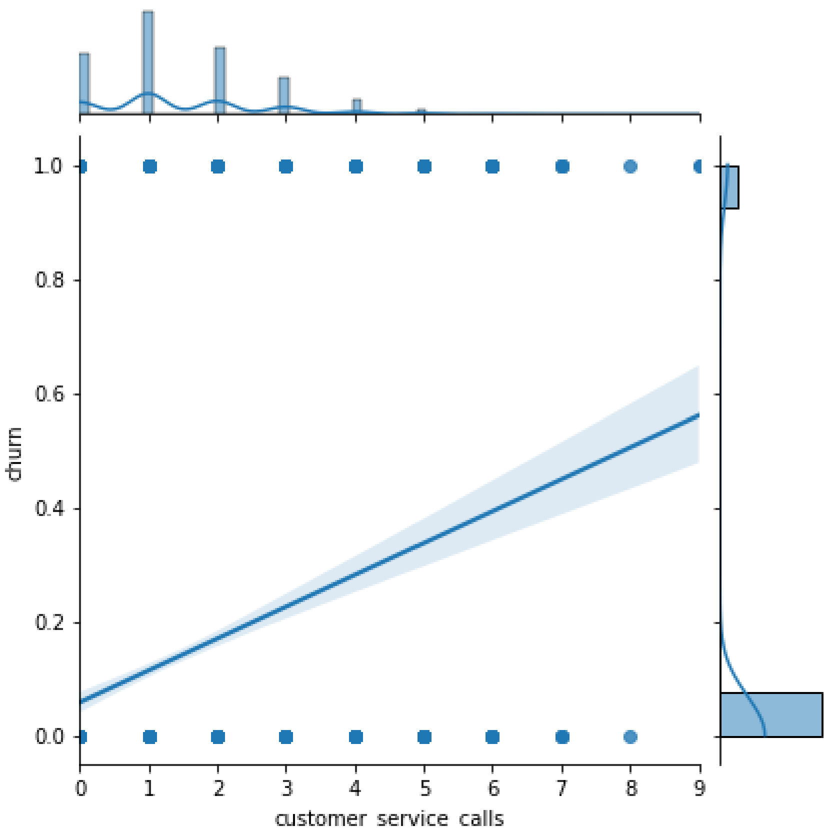

1.0
0.8

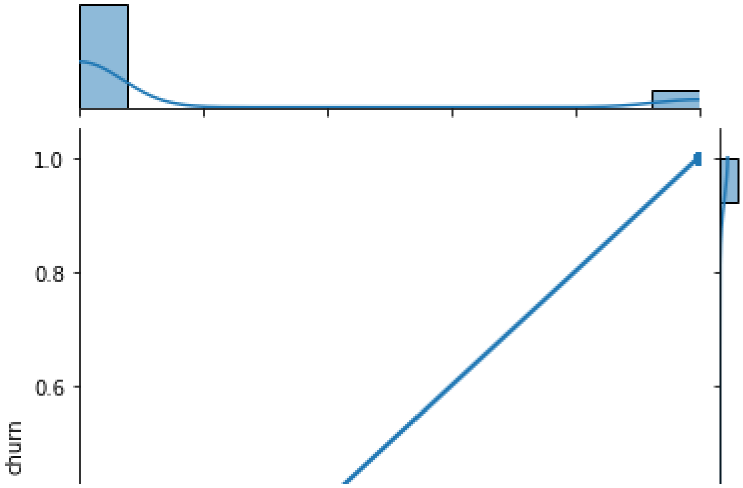

1.0
0.8
0.6
0.4
0.2
0.0
churn
2
4
6
8
10
12
14
16
total_night_charge


churn
1.0
0.8
0.6
0.4
0.2
0.0
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
total_intl_minutes


churn
1.0
0.8
0.6
0.4
0.2
0.0
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
total_intl_calls


churn
1.0
0.8
0.6
0.4
0.2

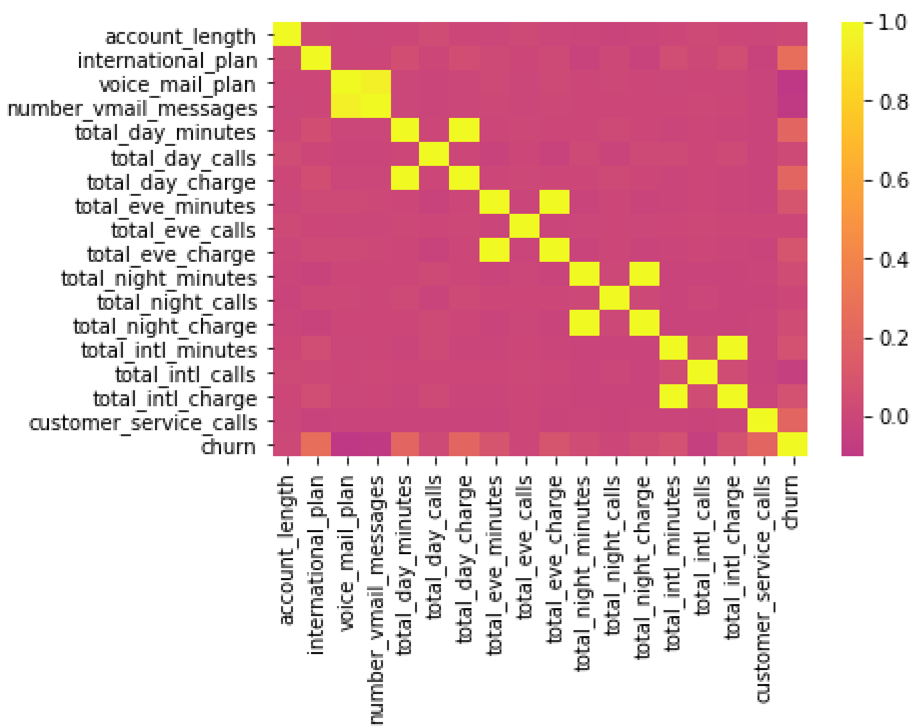

churn
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8
9
customer_service_calls



In [13]:

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

executed in 486ms, finished 00:56:45 2021-04-23



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

In [14]:
```
# 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 00:56:45 2021-04-23

Out[14]:

pairs	cc
(total_day_minutes, total_day_charge)	1.000000
(total_eve_minutes, total_eve_charge)	1.000000
(total_night_charge, total_night_minutes)	0.999999
(total_intl_charge, total_intl_minutes)	0.999993
(number_vmail_messages, voice_mail_plan)	0.956927

- Let's get rid of the factors related to minutes and keep the charge factors. Price is probably 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 [15]:
```
# Dropping total_day_minutes, total_eve_minutes, total_night_minutes, total_intl_

df.drop(['total_day_minutes','total_eve_minutes','total_night_minutes','total_int
```

executed in 15ms, finished 00:56:45 2021-04-23

In [16]:
```
df.head()
```
executed in 14ms, finished 00:56:45 2021-04-23

Out[16]:

	state	account_length	international_plan	voice_mail_plan	total_day_calls	total_day_charge	tota
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	

1.5 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.

1.5.1 Logistic Regression (Baseline)

In [17]:
```
df_dummy = pd.get_dummies(df)
df_dummy.head()
```
executed in 31ms, finished 00:56:45 2021-04-23

Out[17]:

	account_length	international_plan	voice_mail_plan	total_day_calls	total_day_charge	total_eve_c
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 × 64 columns

In [18]:
```
# 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, rando
```
executed in 14ms, finished 00:56:45 2021-04-23

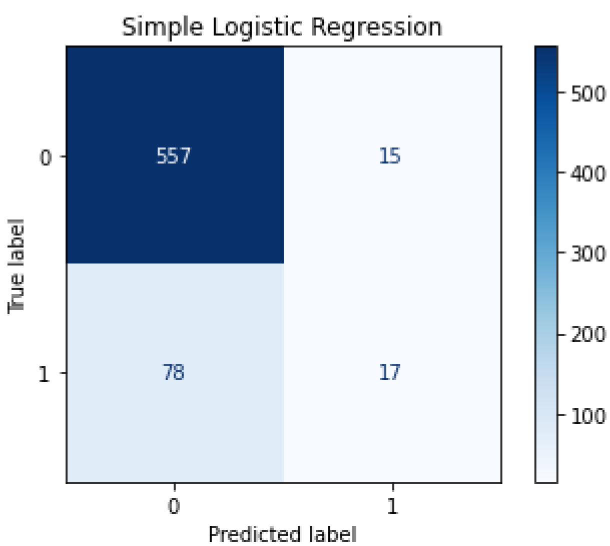
In [19]:
```
# initial regression
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear')
model_log = logreg.fit(X_train, y_train)
```
executed in 60ms, finished 00:56:45 2021-04-23

In [20]:
```
# Prediction
y_hat_train = logreg.predict(X_train)
y_hat_test = logreg.predict(X_test)
```
executed in 26ms, finished 00:56:45 2021-04-23

In [21]:
```
display(model_log.score(X_train, y_train))
display(model_log.score(X_test, y_test))
```
executed in 14ms, finished 00:56:45 2021-04-23

```
0.8690922730682671

0.8605697151424287
```

In [22]:
```
# 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 00:56:45 2021-04-23



In [23]:
```
# we compute our validation metric, recall

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

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

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

print('Training F1-Score: ', f1_score(y_train, y_hat_train))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test))
```

executed in 27ms, finished 00:56:45 2021-04-23

```
Training Precision:  0.6211180124223602
Testing Precision:  0.53125



Training Recall:  0.25773195876288657
Testing Recall:  0.17894736842105263



Training Accuracy:  0.8690922730682671
Testing Accuracy:  0.8605697151424287



Training F1-Score:  0.36429872495446264
Testing F1-Score:  0.2677165354330709
```

- Accuracy is 86%. The accuracy can possibly be higher on for a different model.
- Testing and testing score can be closer.

1.5.2 Random Forest

In [24]:
```
# New dataframe for random forest model
df_rf = pd.get_dummies(df)
```

executed in 14ms, finished 00:56:45 2021-04-23

In [25]:
```
# 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 = 0.25, rando
```

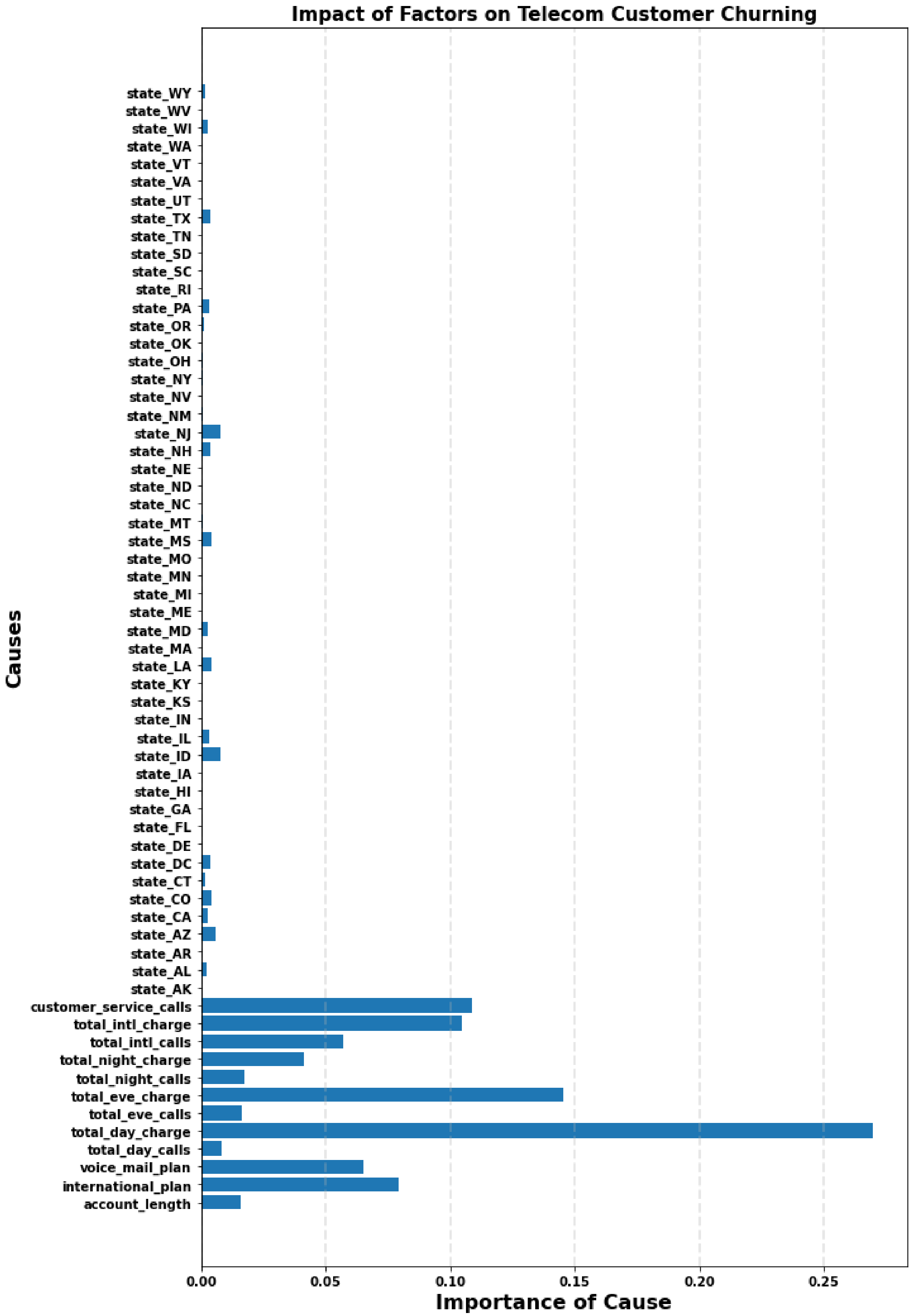
executed in 13ms, finished 00:56:45 2021-04-23

In [26]:
```
# Set up initial forest
tree_clf = DecisionTreeClassifier()
tree_clf.fit(X_train, y_train)
```
executed in 45ms, finished 00:56:45 2021-04-23

Out[26]:
```
DecisionTreeClassifier()
```

In [27]:
```
# Plotting feature importance of models
def plot_feature_importances(model):
    n_features = X_train.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.columns.values, fontsize=10, fontwe
    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, font

plot_feature_importances(tree_clf)
```

executed in 1.45s, finished 00:56:47 2021-04-23


Impact of Factors on Telecom Customer Churning
Causes
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
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
Importance of Cause

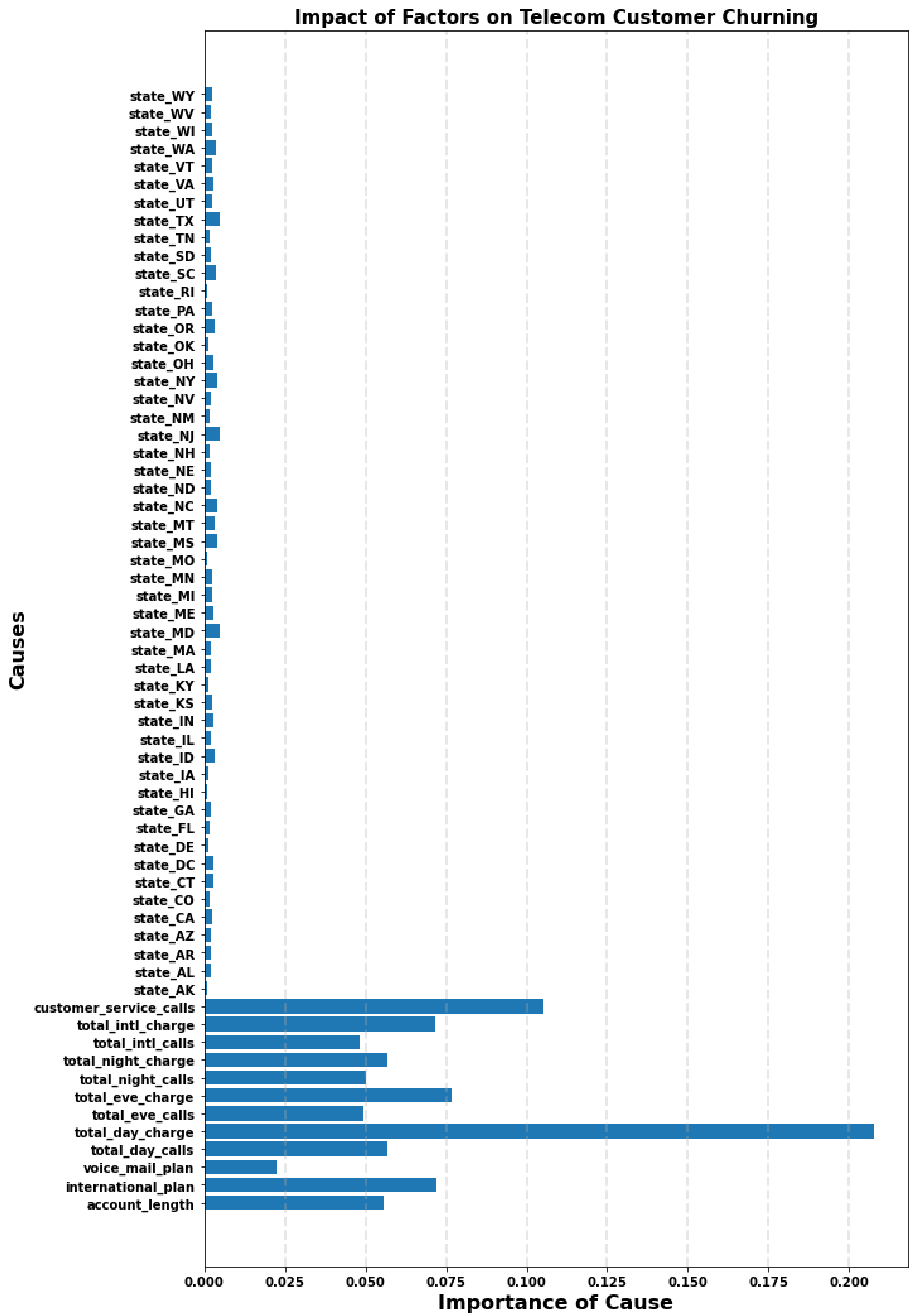
In [28]:
```
# Test set predictions
pred = tree_clf.predict(X_test)

# Confusion matrix and classification report
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print("Testing Accuracy for Decision Tree Classifier: {:.4}%".format(accuracy_sc
```

executed in 14ms, finished 00:56:47 2021-04-23

```
[[683  32]
 [ 31  88]]
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       715
           1       0.73      0.74      0.74       119

    accuracy                           0.92       834
   macro avg       0.84      0.85      0.85       834
weighted avg       0.92      0.92      0.92       834

Testing Accuracy for Decision Tree Classifier: 92.45%
```

In [29]:
```
forest = RandomForestClassifier()
forest.fit(X_train, np.ravel(y_train))
plot_feature_importances(forest)
```

executed in 1.84s, finished 00:56:48 2021-04-23



- The individual states do not seem to have a large influence on the customer churn
- **Customer_service_calls_** and **total_day_charge** appears to have the largest influence on churn

In [30]:
```
param_grid = {
    'max_depth': [2, 5, 10, 25],
    'min_samples_split': [2, 5, 10, 20]
}

gs_tree = GridSearchCV(forest, param_grid, cv=3)
gs_tree.fit(X_train, np.ravel(y_train))

gs_tree.best_params_
```

executed in 10.6s, finished 00:56:59 2021-04-23

Out[30]:
```
{'max_depth': 25, 'min_samples_split': 5}
```

In [31]:
```
forest = RandomForestClassifier(max_depth=5, min_samples_split=2)
forest.fit(X_train, np.ravel(y_train))
preds = forest.predict(X_test)
print(confusion_matrix(y_test, preds))
print(classification_report(y_test, preds))
print("Testing Accuracy for Random Forest Classifier: {:.4}%".format(accuracy_sco
```

executed in 233ms, finished 00:56:59 2021-04-23

```
[[715   0]
 [119   0]]
              precision    recall  f1-score   support

           0       0.86      1.00      0.92       715
           1       0.00      0.00      0.00       119

    accuracy                           0.86       834
   macro avg       0.43      0.50      0.46       834
weighted avg       0.73      0.86      0.79       834

Testing Accuracy for Random Forest Classifier: 85.73%

C:\Users\leebr\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_clas
sification.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division
` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

- Accuracy is still not very high (85%)
- Should try another model

1.5.3 XG Boost

In [32]:
```
# New dataframe for XG Boost model
df_xgb = pd.get_dummies(df)
```

executed in 15ms, finished 00:56:59 2021-04-23

In [33]:
```
# Create X,y and train/test
X = df_xgb.drop(columns=['churn'], axis=1)
y = df_xgb['churn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, rando
```

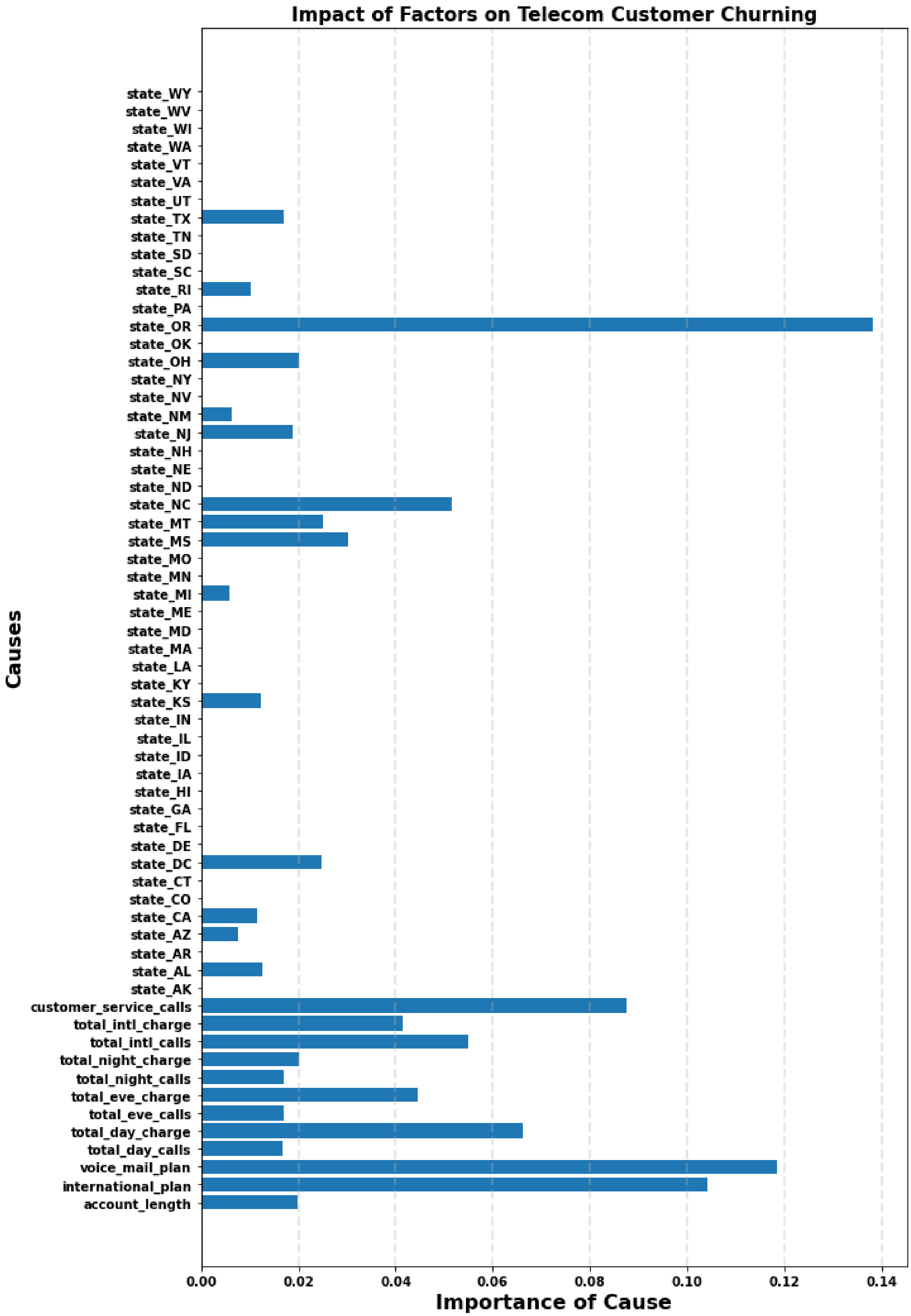
executed in 15ms, finished 00:56:59 2021-04-23

In [34]:
```
# Instantiate XGBClassifier
xgb = XGBClassifier()

# Fit XGBClassifier
xgb.fit(X_train, np.ravel(y_train))

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

# Accuracy of training and test sets
training_accuracy = accuracy_score(y_train, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

executed in 514ms, finished 00:57:00 2021-04-23

```
Training Accuracy: 100.0%
Validation accuracy: 96.04%
```

In [35]:
```
plot_feature_importances(xgb)
```
executed in 1.57s, finished 00:57:01 2021-04-23



In [36]:
```
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='accuracy')
grid_clf.fit(X_train, np.ravel(y_train))

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)
test_preds = grid_clf.predict(X_test)
training_accuracy = accuracy_score(y_train, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print('')
print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

executed in 7.09s, finished 00:57:08 2021-04-23

```
Grid Search found the following optimal parameters:
learning_rate: 0.1
max_depth: 6
min_child_weight: 1
n_estimators: 100
subsample: 0.7

Training Accuracy: 98.44%
Validation accuracy: 96.76%
```

In [37]:
```
print(confusion_matrix(y_test, test_preds))
print(classification_report(y_test, test_preds))
print("Testing Accuracy for XGBoost: {:.4}%".format(accuracy_score(y_test, test_p
```

executed in 30ms, finished 00:57:08 2021-04-23

```
[[712   3]
 [ 24  95]]
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       715
           1       0.97      0.80      0.88       119

    accuracy                           0.97       834
   macro avg       0.97      0.90      0.93       834
weighted avg       0.97      0.97      0.97       834

Testing Accuracy for XGBoost: 96.76%
```

- XG Boost has the highest testing accuracy from the other models.
- We shall use this for our predictive analysis

1.5.4 Final Model Selection

In [38]:
```
df_final = pd.DataFrame(xgb.feature_importances_, X.columns, columns = ['coeffici
df_final.head()
```
executed in 76ms, finished 00:57:09 2021-04-23

Out[38]:

	coefficient
account_length	0.019876
international_plan	0.103945
voice_mail_plan	0.118447
total_day_calls	0.016703
total_day_charge	0.066109

In [39]:
```
df_final_detail = df_final[:12]
display(df_final_detail.head())
df_final_state = df_final[12:]
display(df_final_state.head())
```
executed in 14ms, finished 00:57:09 2021-04-23

	coefficient
account_length	0.019876
international_plan	0.103945
voice_mail_plan	0.118447
total_day_calls	0.016703
total_day_charge	0.066109

	coefficient
state_AK	0.000000
state_AL	0.012708
state_AR	0.000000
state_AZ	0.007599
state_CA	0.011509

In [40]:
```
# 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()
```

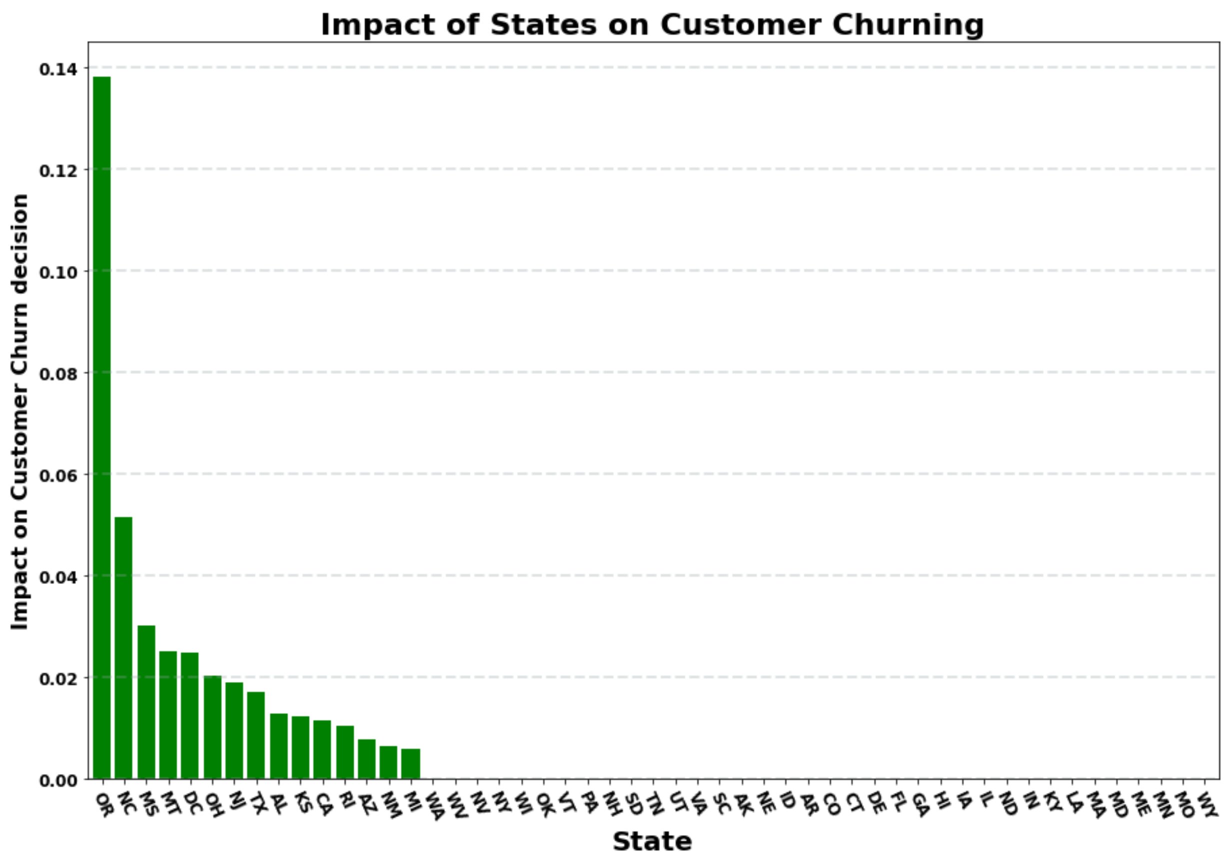
executed in 15ms, finished 00:57:09 2021-04-23

Out[40]:

	coefficient
AK	0.000000
AL	0.012708
AR	0.000000
AZ	0.007599
CA	0.011509

In [41]:
```python
# Create a barplot of state's impact on churn
ax = df_final_state.sort_values(by=['coefficient'], ascending=False).plot(kind='b
ax.set_title("Impact of States on Customer Churning", fontsize = 22, fontweight =
ax.set_xlabel("State", fontsize=20, fontweight='bold')
ax.set_ylabel("Impact on Customer Churn decision", fontsize=16, fontweight='bold'
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 1.06s, finished 00:57:10 2021-04-23



- There seems to be a large churn rate in **state_OR**. This may need to be investigated further.
- No apparent correlation can be found with states with high impact

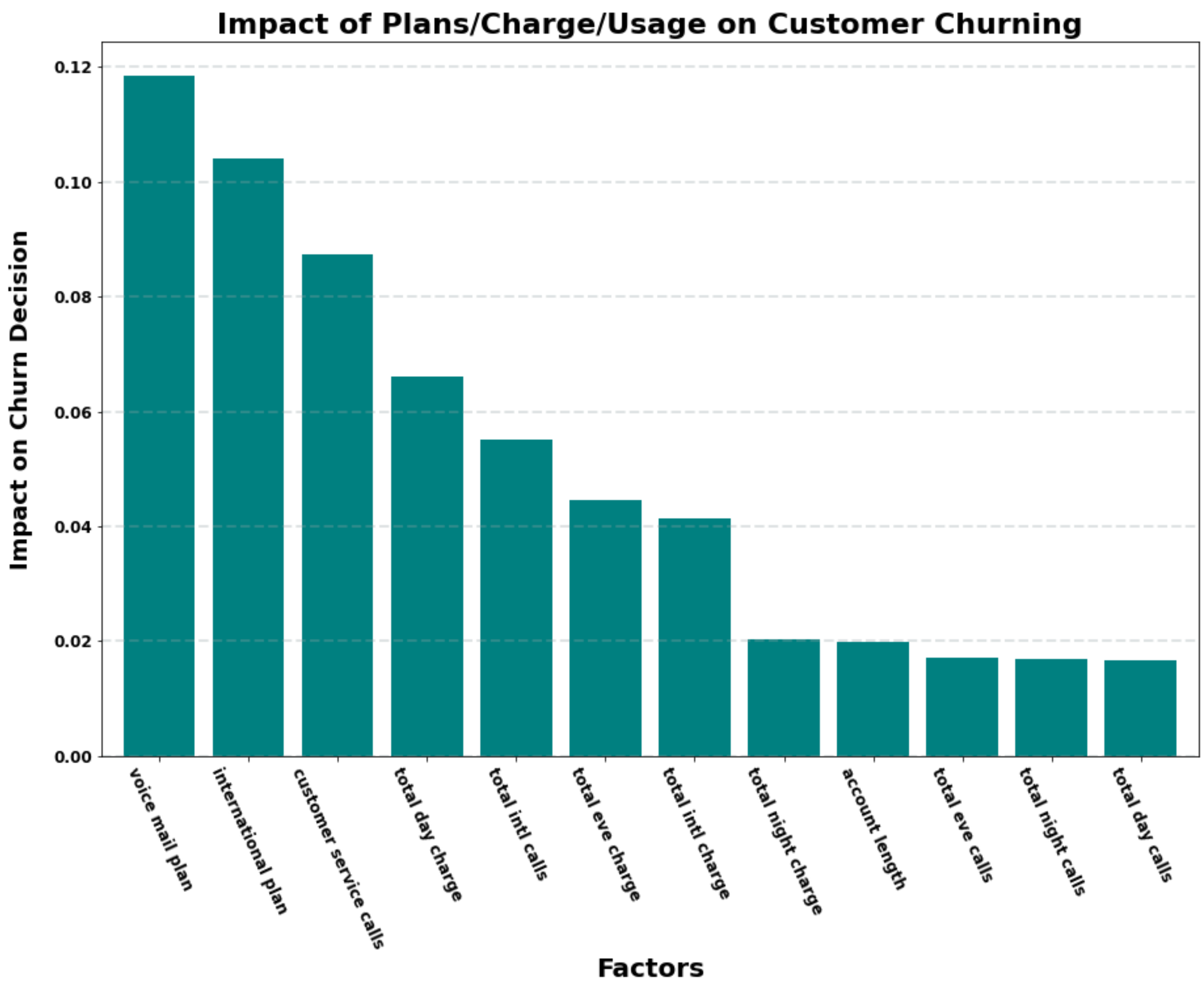
In [42]:
```
# Renaming detail indexes
detail_list = df_final_detail.index.values.tolist()
detail_list = [i.replace('_',' ') for i in detail_list]
df_final_detail.index = detail_list
df_final_detail.head()
```
executed in 14ms, finished 00:57:10 2021-04-23

Out[42]:

	coefficient
account length	0.019876
international plan	0.103945
voice mail plan	0.118447
total day calls	0.016703
total day charge	0.066109

In [43]:
```
# Create a barplot of customer plans/usage's impact on churn
ax = df_final_detail.sort_values(by=['coefficient'], ascending=False).plot(kind='
ax.set_title("Impact of Plans/Charge/Usage on Customer Churning", fontsize = 22,
ax.set_xlabel("Factors", fontsize=20, fontweight='bold')
ax.set_ylabel("Impact on Churn Decision", fontsize=18, fontweight='bold', labelpa
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 251ms, finished 00:57:10 2021-04-23



- **voice_mail_plan** and **international_plan** are the largest factors to customer churn. There may be some issues with how both plans are being handled.
- **customer_service_calls** come close in influence to the plans. There may need to be an improvement to the customer service section of the company.

1.6 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.

1.7 Next Steps

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

- **Locate what factors are causing a larger churn 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**