**Churn reduction**

**Chintan Shah**

**29 November 2018**

Contents

[**Abstract** 3](#_Toc531251471)

[**Introduction** 3](#_Toc531251472)

[**Exploratory data analysis** 3](#_Toc531251473)

[**Describe the data** 4](#_Toc531251474)

[**Null- Values** 5](#_Toc531251475)

[**Churn Counts** 5](#_Toc531251476)

[**Churn ratio by categorical predictors** 7](#_Toc531251477)

[**Explore distributions by continuous predictors** 8](#_Toc531251478)

[**Feature Selection** 8](#_Toc531251479)

[**Encoding dependent variable** 10](#_Toc531251480)

[**Fitting a Model** 11](#_Toc531251481)

[**Evaluating Our Model** 12](#_Toc531251482)

[**Confusion matrix – Random Forest classifier** 13](#_Toc531251483)

[**Confusion matrix – Naïve Bayes classifier** 14](#_Toc531251484)

[**Remarks** 15](#_Toc531251485)

[**Python Code** 15](#_Toc531251486)

[**R Code** 21](#_Toc531251487)

# **Abstract**

The telecom market is saturated and customer growth rates are low. They key focus of market players therefore is on retention and churn control. This project explores the churn dataset to identify the key drivers of churn and builds the best predictive model to predict churn. A strategy to reduce churn is presented and the proposal is evaluated against the model.

# **Introduction**

A common problem across businesses in many industries is that of customer churn. Businesses often have to invest substantial amounts attracting new clients, so every time a client leaves it represents a significant investment lost. Both time and effort then need to be channelled into replacing them. Being able to predict when a client is likely to leave and offer them incentives to stay can offer huge savings to a business. This is the essence of customer churn prediction; how can we quantify if and when a customer is likely to churn?

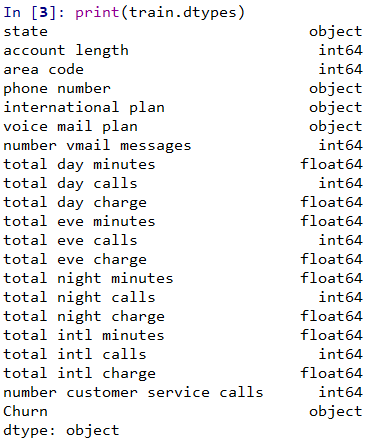
# **Exploratory data analysis**

The churn dataset is split into churnTrain (3333 obs.) and churnTest (1667 obs.) of 20 predictor variables and 1 response variable (churn = yes/no). The proportion of churned customers (churn = yes) is evenly distributed across the 2 sets. Train will be used for data exploration and model building, while Test will be used to measure model performance

Now we can perform some basic exploratory analysis to get a better understanding of what is in our data.

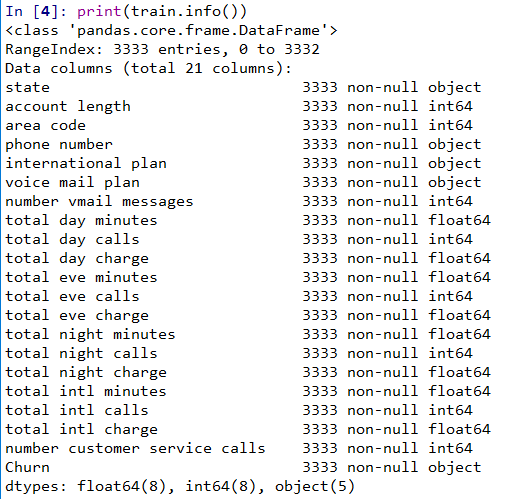
* How much data we have
* If there are any missing values
* What data type each column is
* The distribution of data in each column

# **Describe the data**



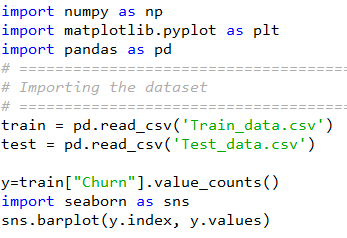
# **Null- Values**

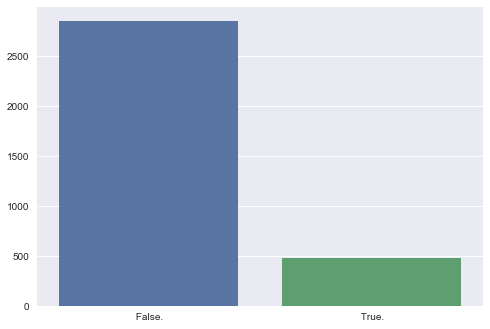
**Checking if dataset has null values**

****

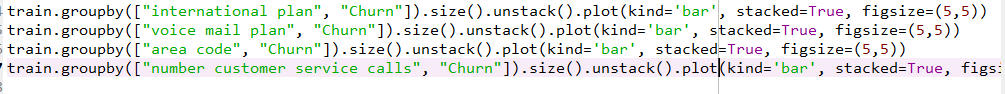
# **Churn Counts**

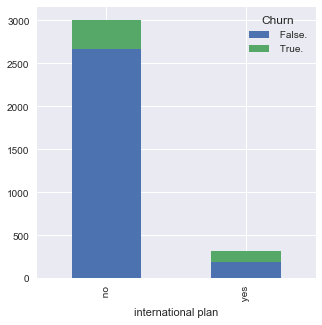
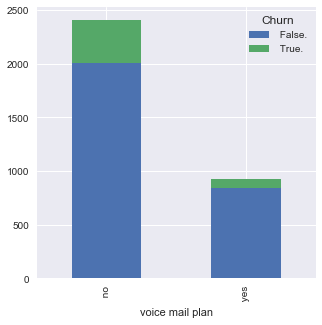
The churn variable has values of yes or no. We have presented here with bar plot

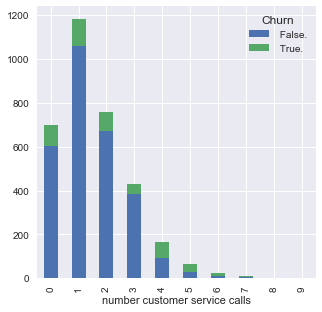
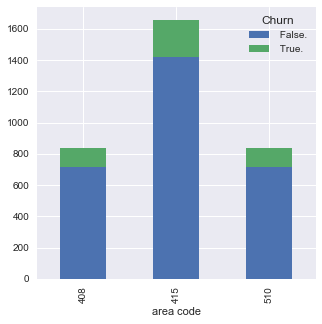




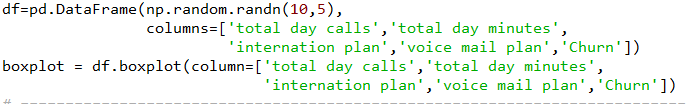
# **Churn ratio by categorical predictors**

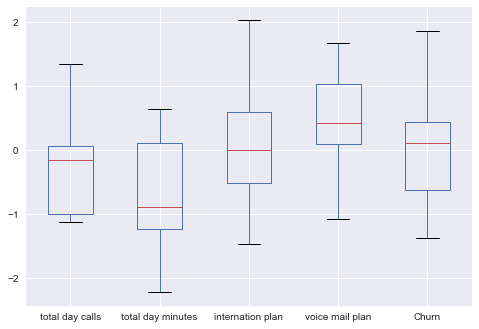




# **Explore distributions by continuous predictors**





The continuous predictors include total minutes, number of calls & charges across day, evening, night and international calls. As call minutes and charges were strongly correlated.

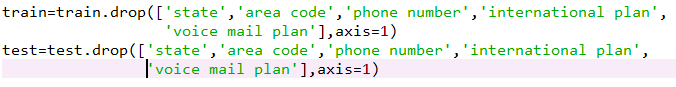
# **Feature Selection**

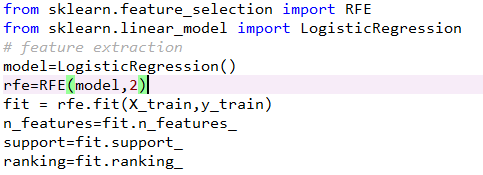
After cleaning and inspecting our data we might come to the conclusion that certain columns are not going to be useful for prediction. In this example we will not be using the phone-number of the client or geographical information about the client because our assumption is that this shouldn't affect churn.

In addition, we have used, recursive feature elimination technique to select best features to train our model.

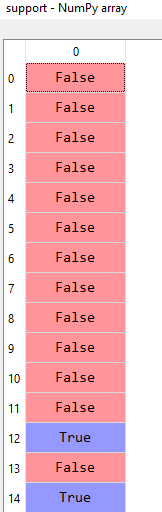
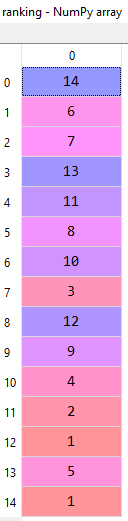
Datasets used to train classification and regression algorithms are high dimensional in nature — this means that they contain many features or attributes. In textual datasets each feature is a word and as you can imagine the vocabulary used in the dataset can be very large. Not all features however, contribute to the prediction variable. Removing features of low importance can improve accuracy, and reduce both model complexity and overfitting. Training time can also be reduced for very large datasets.

Recursive Feature Elimination (RFE) as its title suggests recursively removes features, builds a model using the remaining attributes and calculates model accuracy. RFE is able to work out the combination of attributes that contribute to the prediction on the target variable (or class). Scikit Learn does most of the heavy lifting just import RFE from [sklearn.feature\_selection](http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html" \l "sklearn.feature_selection.RFE" \t "_blank) and pass any classifier model to the RFE() method with the number of features to select. Using familiar Scikit Learn syntax, the .fit() method must then be called.





After performing REF, we have received few variable, i.e. support and ranking. The support variable shows value of true and false whereas ranking provide the number to each feature. Therefore, as per requirement, the model selected features as per number



# **Encoding dependent variable**

Categorical data are variables that contain label values rather than numeric values.

The number of possible values is often limited to a fixed set.

Categorical variables are often called [nominal](https://en.wikipedia.org/wiki/Nominal_category).

Some examples include:

* A “pet” variable with the values: “dog” and “cat“.
* A “color” variable with the values: “red“, “green” and “blue“.

Each value represents a different category.

Some algorithms can work with categorical data directly.

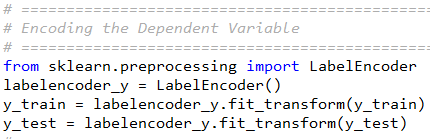
For example, a decision tree can be learned directly from categorical data with no data transform required (this depends on the specific implementation).

Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.

In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves.

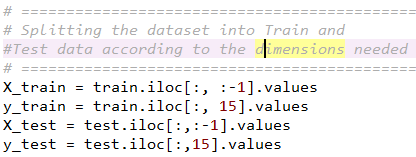
This means that categorical data must be converted to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application.

As in dataset, we are not using all variables. Only we are using ‘Churn’ variable, which has value of yes and no. Therefore, we have to convert into numeric form by using LabelEncoder class of Sklearn.

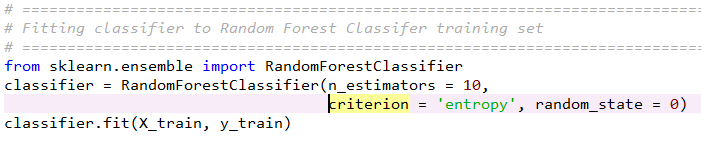


# **Fitting a Model**

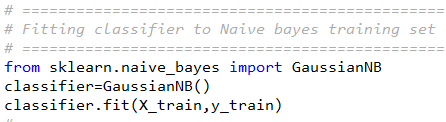
At this point we can construct our model. The first thing to do is split our dataset into training and test sets.



Once we have obtained our split we can use the [RandomForestClassifier()](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) from the sklearn library as our model. We initialise our model, fit it to our dataset using the [fit()](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier.fit) method, then simply make our predictions using the [predict()](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier.predict)method.



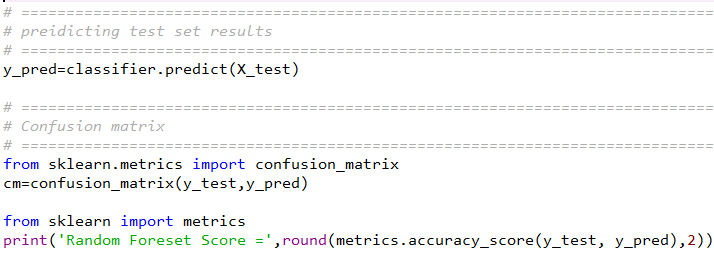
Given the ease of setting up a basic model, a common approach is to initialize and train a variety of different models and pick the most performant one as a starting point. For example, we might also choose to run a naïve Bayes classifier.

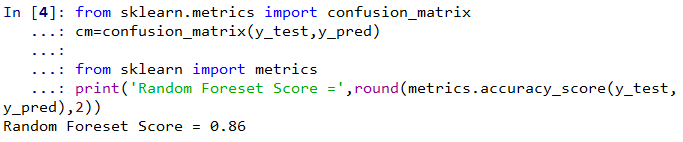


# **Evaluating Our Model**

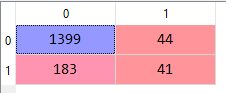
If we display the results we can see we have a list of booleans (0's and 1's) representing whether or not our model thinks a customer has churned or not. Now we can compare this to whether they actually churned to evaluate our model. We could also compute the actual probabilities of a customer churning using [predict](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier.predict_proba) rather than just simple yes / no. We could then use these probabilities as a threshold for driving business decisions around which customers we need to target for retention, and how strong an incentive we need to offer them.

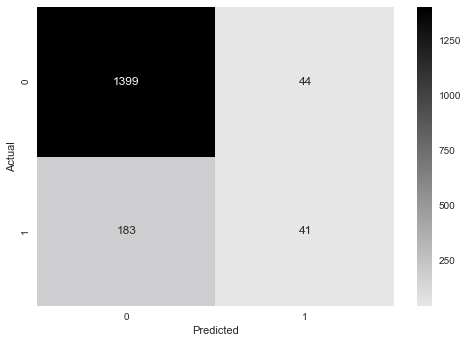
We can achieve the comparison mentioned above by using the [.score()](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier.score) method, and displaying that we can see that we have achieved an accuracy of over 90%, which is not bad for our first attempt.



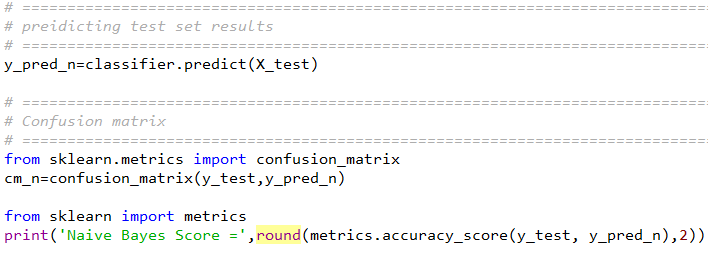


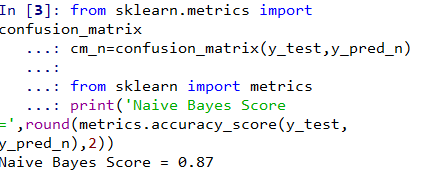
# **Confusion matrix – Random Forest classifier**



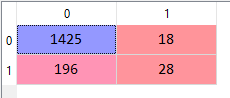


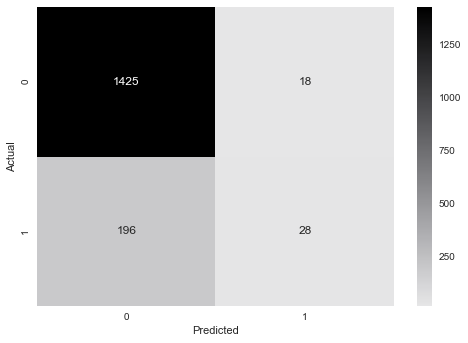
**Naïve Bayes Classifier**





# **Confusion matrix – Naïve Bayes classifier**





# **Remarks**

At per model performance, we can say that naïve bayes classifier shown better result in compare to random forest classifier

# **Python Code**

# -\*- coding: utf-8 -\*-

"""

Created on Fri Nov 23 10:53:01 2018

@author: Chintan

"""

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# =============================================================================

# Importing the dataset

# =============================================================================

train = pd.read\_csv('Train\_data.csv')

test = pd.read\_csv('Test\_data.csv')

# =============================================================================

# Describing the data

# =============================================================================

train.describe()

print(train.dtypes)

# =============================================================================

# Checking if dataset has null values

# =============================================================================

print(train.info())

# =============================================================================

# Information on churn#

# =============================================================================

y=train["Churn"].value\_counts()

import seaborn as sns

sns.barplot(y.index, y.values)

# =============================================================================

# Descriptive Analysis

# =============================================================================

df=pd.DataFrame(np.random.randn(10,5),

columns=['total day calls','total day minutes',

'internation plan','voice mail plan','Churn'])

boxplot = df.boxplot(column=['total day calls','total day minutes',

'internation plan','voice mail plan','Churn'])

# =============================================================================

# Churn By Internation plan

# =============================================================================

train.groupby(["international plan", "Churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(5,5))

train.groupby(["voice mail plan", "Churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(5,5))

train.groupby(["area code", "Churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(5,5))

train.groupby(["number customer service calls", "Churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(5,5))

# ===========================================================================

# Churn By State

# =============================================================================

train.groupby(["state","international plan", "Churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(10,10))

# =============================================================================

# Dropping unwanted columns from train & testtdataset

# =============================================================================

train=train.drop(['state','area code','phone number','international plan',

'voice mail plan'],axis=1)

test=test.drop(['state','area code','phone number','international plan',

'voice mail plan'],axis=1)

# =============================================================================

# Splitting the dataset into Train and

#Test data according to the dimensions needed

# =============================================================================

X\_train = train.iloc[:, :-1].values

y\_train = train.iloc[:, 15].values

X\_test = test.iloc[:,:-1].values

y\_test = test.iloc[:,15].values

# =============================================================================

# Encoding the Dependent Variable

# =============================================================================

from sklearn.preprocessing import LabelEncoder

labelencoder\_y = LabelEncoder()

y\_train = labelencoder\_y.fit\_transform(y\_train)

y\_test = labelencoder\_y.fit\_transform(y\_test)

#==============================================================================

# Feature ranking with recursive feature elimination

#==============================================================================

from sklearn.feature\_selection import RFE

from sklearn.linear\_model import LogisticRegression

# feature extraction

model=LogisticRegression()

rfe=RFE(model,2)

fit = rfe.fit(X\_train,y\_train)

n\_features=fit.n\_features\_

support=fit.support\_

ranking=fit.ranking\_

X\_train=X\_train[:,[12,14]]

X\_test=X\_test[:,[12,14]]

# =============================================================================

# Feature Scaling

# =============================================================================

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# =============================================================================

# Fitting classifier to Random Forest Classifer training set

# =============================================================================

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators = 10,

criterion = 'entropy', random\_state = 0)

classifier.fit(X\_train, y\_train)

# =============================================================================

# preidicting test set results

# =============================================================================

y\_pred=classifier.predict(X\_test)

print(y\_pred)

# =============================================================================

# Confusion matrix

# =============================================================================

from sklearn.metrics import confusion\_matrix

cm=confusion\_matrix(y\_test,y\_pred)

from sklearn import metrics

print('Random Foreset Score =',round(metrics.accuracy\_score(y\_test, y\_pred),2))

conf = (metrics.confusion\_matrix(y\_test, y\_pred))

cmap = sns.cubehelix\_palette(50, hue=0.05, rot=0, light=0.9, dark=0, as\_cmap=True)

sns.heatmap(conf,cmap = cmap,xticklabels=['0','1'],yticklabels=['0','1'],annot=True, fmt="d",)

plt.xlabel('Predicted')

plt.ylabel('Actual')

# =============================================================================

# Fitting classifier to Naive bayes training set

# =============================================================================

from sklearn.naive\_bayes import GaussianNB

classifier1=GaussianNB()

classifier1.fit(X\_train,y\_train)

# =============================================================================

# preidicting test set results

# =============================================================================

y\_pred\_n=classifier1.predict(X\_test)

# =============================================================================

# Confusion matrix

# =============================================================================

from sklearn.metrics import confusion\_matrix

cm\_n=confusion\_matrix(y\_test,y\_pred\_n)

from sklearn import metrics

print('Naive Bayes Score =',round(metrics.accuracy\_score(y\_test, y\_pred\_n),2))

conf = (metrics.confusion\_matrix(y\_test, y\_pred\_n))

cmap = sns.cubehelix\_palette(50, hue=0.05, rot=0, light=0.9, dark=0, as\_cmap=True)

sns.heatmap(conf,cmap = cmap,xticklabels=['0','1'],yticklabels=['0','1'],annot=True, fmt="d",)

plt.xlabel('Predicted')

plt.ylabel('Actual')

# Applying k-Fold Cross Validation

from sklearn.model\_selection import cross\_val\_score

accuracies = cross\_val\_score(estimator = classifier, X = X\_train, y = y\_train, cv = 10)

accuracies.mean()

accuracies.std()

# **R Code**

rm(list=ls(all=T))

setwd("E:/Subject/edwisor/project 2")

library(ggplot2)

library(gridExtra)

library(dplyr)

library(corrplot)

library(pROC)

library(C50)

library(caret)

library(rpart)

train = read.csv('Train\_data.csv')

test = read.csv('Test\_data.csv')

#Exploratory data analysis

var1 =ggplot(train, aes(area.code, fill = Churn)) + geom\_bar(position = "fill") + labs(x = "Area code", y = "") + theme(legend.position = "none")

var2 =ggplot(train, aes(international.plan, fill = Churn)) + geom\_bar(position = "fill") + labs(x = "International?", y = "") + theme(legend.position = "none")

var3 = ggplot(train, aes(voice.mail.plan, fill = Churn)) + geom\_bar(position = "fill") + labs(x = "Voicemail?", y = "") + theme(legend.position = "none")

var4 = ggplot(train, aes(number.customer.service.calls, fill = Churn)) + geom\_bar(position = "fill") + labs(x = "Customer calls", y = "") + theme(legend.position = "none")

grid.arrange(var1, var2, var3, var4, ncol = 4, nrow = 1, top = "Churn & Non-Churn Chart")

#Explore distributions by continuous predictors

daymin = ggplot(train, aes(Churn, total.day.minutes, fill = Churn)) + geom\_boxplot(alpha = 0.8) + theme(legend.position = "null")

evemin = ggplot(train, aes(Churn, total.eve.minutes, fill = Churn)) + geom\_boxplot(alpha = 0.8) + theme(legend.position = "null")

nitmin = ggplot(train, aes(Churn, total.night.minutes, fill = Churn)) + geom\_boxplot(alpha = 0.8) + theme(legend.position = "null")

intmin =- ggplot(train, aes(Churn, total.intl.minutes, fill = Churn)) + geom\_boxplot(alpha = 0.8) + theme(legend.position = "null")

daycal = ggplot(train, aes(Churn, total.day.calls, fill = Churn)) + geom\_boxplot(alpha = 0.8) + theme(legend.position = "null")

evecal = ggplot(train, aes(Churn, total.eve.calls, fill = Churn)) + geom\_boxplot(alpha = 0.8) + theme(legend.position = "null")

nitcal = ggplot(train, aes(Churn, total.night.calls, fill = Churn)) + geom\_boxplot(alpha = 0.8) + theme(legend.position = "null")

intcal = ggplot(train, aes(Churn, total.intl.calls, fill = Churn)) + geom\_boxplot(alpha = 0.8) + theme(legend.position = "null")

grid.arrange(daymin, evemin, nitmin, intmin,

daycal, evecal, nitcal, intcal,

ncol = 4, nrow = 2)

#Find Missing values

anyNA(train)

#Check for collinearity.

corrplot(cor(train[sapply(train, is.numeric)]))

#Remove unnecessary features, which not required for train

train$state = NULL

train$area.code = NULL

train$phone.number = NULL

train$international.plan = NULL

train$voice.mail.plan = NULL

test$state = NULL

test$area.code = NULL

test$phone.number = NULL

test$international.plan = NULL

test$voice.mail.plan = NULL

#Encode dependent variable

train$Churn = factor(train$Churn,

levels = c(' False.', ' True.'),

labels = c(0, 1))

test$Churn = factor(test$Churn,

levels = c(' False.', ' True.'),

labels = c(0, 1))

# Fitting Random Forest Classification to the Training set

# install.packages('randomForest')

library(randomForest)

set.seed(123)

classifier = randomForest(x = train[-1],

y = train$Churn,

ntree = 500)

y\_pred = predict(classifier, newdata = test[-1])

cm = table(test[, 16], y\_pred)

# Fitting Naive Bayes Classification to the Training set

#install.packages('e1071')

library(e1071)

classifier\_n = naiveBayes(x = train[-1],

y = train$Churn)

# Predicting the Test set results

y\_pred\_n = predict(classifier\_n, newdata = test[-1])

# Making the Confusion Matrix

cm\_n = table(test[, 16], y\_pred\_n)