Capstone Project: Telecom's Churn Reduction

Supervised by: Prof. Sergei Schreider

Submitted by: Tarang Haria

RUID: 190004326

Email ID: th543@scaretmail.rutgers.edu

INDEX

Contents:

1. INTRODUCTION

1. What is 'Churn Rate'?

The churn rate is the percentage of subscribers to a service who discontinue their subscriptions to the service within a given time period. For a company to expand its clientele, its growth rate, as measured by the number of new customers, must exceed its churn rate. This rate is generally expressed as a percentage.

2. Why is churn so important?

Customer churn (also known as customer attrition, customer turnover or customer defection) is a term used especially in the world of subscription-based businesses to describe loss of customers. For example, if 10 out of 100 subscribers to an Internet service provider (ISP) cancelled their subscriptions, the churn rate for that ISP would be 10%.
Churn is important because it directly affects your service's profitability. It is common to assume that the profitability of a service is directly related to the growth of its customer base. That might lead business owners to imply that in order to grow their customer base, the rate of acquiring new customers must exceed the churn rate.

The objective of this case is to predict the customer behavior. Will be using a public dataset that has customer usage pattern and if the customer has moved or not. Will develop an algorithm that will predict the churn score based on usage pattern.

2. DATA PREPROCESSING

We have cleaned data to fill in the missing values, smooth out the noise and correct inconsistencies in our data.

1. Calculating missing values:

In [380]:
```
missing_train = pd.DataFrame(traindata.isnull().sum())
missing_train
```

Out[380]:

	0
account length	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
number customer service calls	0
Churn	0

In [381]:
```
missing_test = pd.DataFrame(testdata.isnull().sum())
missing_test
```

Out[381]:

	0
account length	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
number customer service calls	0
Churn	0

After analysing missing values for our train and test data, we found that there are no missing values.

2. Storing Numeric Variables:

In order to improve the consistency of our data, we store the variabes in our dataset as numeric.

In [383]:
```
cnames =  ["account length","number vmail messages","total day minutes","total day calls","total day charge","total eve minute
s","total eve calls","total eve charge","total night minutes","total night calls","total night charge","total intl minutes","tot
al intl calls","total intl charge","number customer service calls"]
cnames
```

Out[383]:
```
['account length',
 'number vmail messages',
 'total day minutes',
 'total day calls',
 'total day charge',
 'total eve minutes',
 'total eve calls',
 'total eve charge',
 'total night minutes',
 'total night calls',
 'total night charge',
 'total intl minutes',
 'total intl calls',
 'total intl charge',
 'number customer service calls']
```

3. Detecting and Deleting the Outliers:

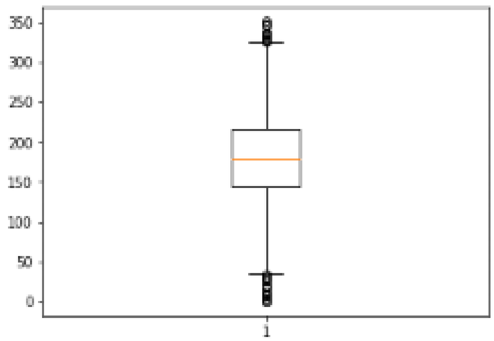
In [384]:
```
for i in cnames:
    print(i)
    q75, q25 = np.percentile(traindata.loc[:,i], [75 ,25])
    iqr = q75 - q25

    min = q25 - (iqr*1.5)
    max = q75 + (iqr*1.5)
    print(min)
    print(max)

    traindata = traindata.drop(traindata[traindata.loc[:,i] < min].index)
    traindata = traindata.drop(traindata[traindata.loc[:,i] > max].index)
```

```
account length
-5.5
206.5
number vmail messages
-30.0
50.0
total day minutes
34.83749999999992
325.1375000000001
total day calls
46.5
154.5
total day charge
6.125
55.125
total eve minutes
64.42499999999995
337.82500000000005
total eve calls
46.5
154.5
total eve charge
5.5550000000000015
28.634999999999998
total night minutes
64.3
337.90000000000003
total night calls
48.0
152.0
total night charge
2.9449999999999985
15.145000000000001
total intl minutes
3.1000000000000005
17.5
total intl calls
-1.5
10.5
total intl charge
0.8949999999999996
4.695
number customer service calls
-0.5
3.5
```

Outlier analysis

In [382]:
```
get_ipython().run_line_magic('matplotlib', 'inline')
plt.boxplot(traindata["total day minutes"])
```

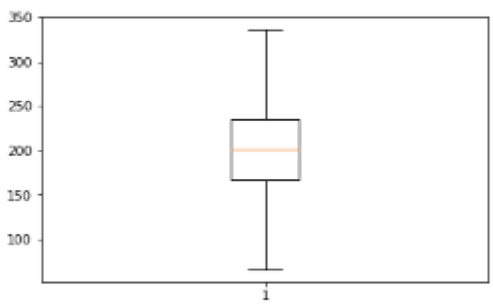
Out[382]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x1e2b9054208>,
  <matplotlib.lines.Line2D at 0x1e2b90546a0>],
 'caps': [<matplotlib.lines.Line2D at 0x1e2b9054ac8>,
  <matplotlib.lines.Line2D at 0x1e2b9054ef0>],
 'boxes': [<matplotlib.lines.Line2D at 0x1e2b90540b8>],
 'medians': [<matplotlib.lines.Line2D at 0x1e2b905d358>],
 'fliers': [<matplotlib.lines.Line2D at 0x1e2b905d780>],
 'means': []}
```



Checking data after boxplot. Here we have deleted all the outliers.

In [385]:
```
plt.boxplot(traindata["total eve minutes"])
traindata.shape
```

Out[385]: (2797, 18)



4.Removing unwanted columns:

After viewing the data, we see that we don't require the variables such as State, Phone Number and Area Code.

Hence we will drop them.

In [377]:
```
traindata =traindata.drop(columns =['phone number','area code','state'])
testdata=testdata.drop(columns = ['phone number','area code','state'])
```

New Train Data

In [378]:
```
traindata.head()
```

Out[378]:

	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge	nu cust se
0	128	no	yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1
1	107	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1
2	137	no	no	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0
3	84	yes	no	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2
4	75	yes	no	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3

New Test Data

In [379]:
```
testdata.head()
```

Out[379]:

	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge	nu cust se
0	101	no	no	0	70.9	123	12.05	211.9	73	18.01	236.0	73	10.62	10.6	3	2.86	3
1	137	no	no	0	223.6	86	38.01	244.8	139	20.81	94.2	81	4.24	9.5	7	2.57	0
2	103	no	yes	29	294.7	95	50.10	237.3	105	20.17	300.3	127	13.51	13.7	6	3.70	1
3	99	no	no	0	216.8	123	36.86	126.4	88	10.74	220.6	82	9.93	15.7	2	4.24	1
4	108	no	no	0	197.4	78	33.56	124.0	101	10.54	204.5	107	9.20	7.7	4	2.08	2

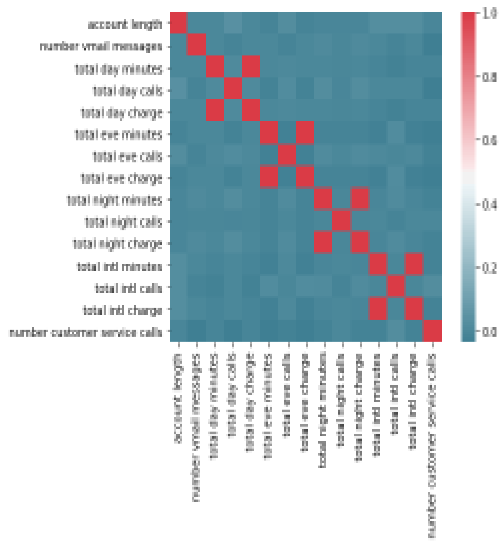
3. FEATURE SELECTION

1. Correlation Analysis:

In [386]:
```
#Correlation plot
corr = traindata.loc[:,cnames]
```

In [387]:
```
#Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(7, 5))
#Generate correlation matrix
corr = corr.corr()
#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```

Out[387]: `<matplotlib.axes._subplots.AxesSubplot at 0x1e2b90e6d68>`



In [388]:
```
#We remove highly coorelated variables we found from correlation test
#a.total day minutes
#b.total night minutes
#c.total eve minutes
#d.total intl minutes
traindata = traindata.drop(['total day minutes','total eve minutes','total intl minutes','total night minutes'],axis=1)
```

In [389]:
```
traindata.head()
```

Out[389]:

	account length	international plan	voice mail plan	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night calls	total night charge	total intl calls	total intl charge	number customer service calls	Churn
0	128	no	yes	25	110	45.07	99	16.78	91	11.01	3	2.70	1	False.
1	107	no	yes	26	123	27.47	103	16.62	103	11.45	3	3.70	1	False.
2	137	no	no	0	114	41.38	110	10.30	104	7.32	5	3.29	0	False.
4	75	yes	no	0	113	28.34	122	12.61	121	8.41	3	2.73	3	False.
5	118	yes	no	0	98	37.98	101	18.75	118	9.18	6	1.70	0	False.

2. Chi-square Test of Independence:

In [390]:
```
#Save categorical variables
categories = ["international plan","voice mail plan"]
#Loop for chi square values
for i in categories:
    print(i)
    chi2, p, dof, ex = chi2_contingency(pd.crosstab(traindata['Churn'],traindata[i]))
    print(p)
```

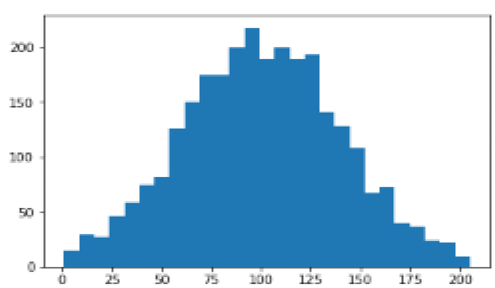
```
international plan
1.6860769270699622e-53
voice mail plan
2.6438944498671704e-07
```

The p value is less than 0.05, so we can use both the categorical variables for our test.

3. Normalization and Sampling:

In [391]:
```
get_ipython().run_line_magic('matplotlib', 'inline')
plt.hist(traindata['account length'], bins='auto')
```

Out[391]:
```
(array([ 15.,  29.,  27.,  46.,  59.,  75.,  82., 126., 150., 175., 175.,
        200., 218., 189., 200., 189., 194., 141., 128., 108.,  67.,  72.,
         40.,  37.,  24.,  22.,   9.]),
 array([  1.        ,   8.55555556,  16.11111111,  23.66666667,
         31.22222222,  38.77777778,  46.33333333,  53.88888889,
         61.44444444,  69.        ,  76.55555556,  84.11111111,
         91.66666667,  99.22222222, 106.77777778, 114.33333333,
        121.88888889, 129.44444444, 137.        , 144.55555556,
        152.11111111, 159.66666667, 167.22222222, 174.77777778,
        182.33333333, 189.88888889, 197.44444444, 205.        ]),
 <a list of 27 Patch objects>)
```



In [392]:
```
cnames1 =  ["account length","total day charge","total day calls","total eve charge","total eve calls","total night charge","total night calls","total intl charge","total intl calls","number customer service calls"]
cnames1
```

Out[392]:
```
['account length',
 'total day charge',
 'total day calls',
 'total eve charge',
 'total eve calls',
 'total night charge',
 'total night calls',
 'total intl charge',
 'total intl calls',
 'number customer service calls']
```

Normalisation

In [393]:
```
high = 1.0
low = 0.0
for i in cnames1:
    mins = np.min(traindata[i], axis=0)
    maxs = np.max(traindata[i], axis=0)
    rng = maxs - mins
    traindata[i] = high - (((high - low) * (maxs - traindata[i])) / rng)
    len(traindata)
traindata.head()
```

Out[393]:

	account length	international plan	voice mail plan	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night calls	total night charge	total intl calls	total intl charge	number customer service calls	Churn
0	0.622549	no	yes	25	0.600000	0.798430	0.481132	0.486710	0.413462	0.665289	0.222222	0.474667	0.333333	False.
1	0.519608	no	yes	26	0.723810	0.434944	0.518868	0.479739	0.528846	0.701653	0.222222	0.741333	0.333333	False.
2	0.666667	no	no	0	0.638095	0.722222	0.584906	0.204357	0.538462	0.360331	0.444444	0.632000	0.000000	False.
4	0.362745	yes	no	0	0.628571	0.452912	0.698113	0.305011	0.701923	0.450413	0.222222	0.482667	1.000000	False.
5	0.573529	yes	no	0	0.485714	0.652003	0.500000	0.572549	0.673077	0.514050	0.555556	0.208000	0.000000	False.

Stratified sampling

In [394]:
```
from sklearn.cross_validation import train_test_split

#Select categorical variable
y = testdata['international plan']

#select subset using stratified Sampling
Rest, testdata = train_test_split(testdata, test_size = 0.5, stratify = y)
```

4. DESCRIPTIVE ANALYSIS

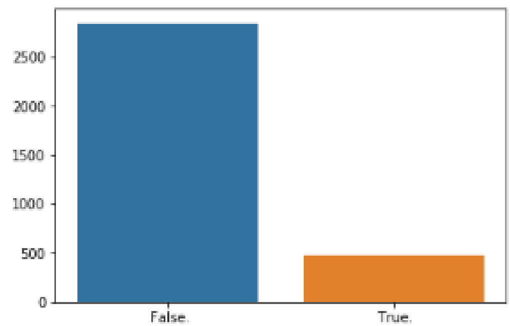
A) Our first analysis is to check the churn ratio, to find the number of users that have moved or not?

Churn Ratio

In [372]:
```
yy = traindata["Churn"].value_counts()
print (yy)
sns.barplot(yy.index, yy.values)
```

```
 False.    2850
 True.      483
Name: Churn, dtype: int64
```

Out[372]: <matplotlib.axes._subplots.AxesSubplot at 0x1e2b8985fd0>



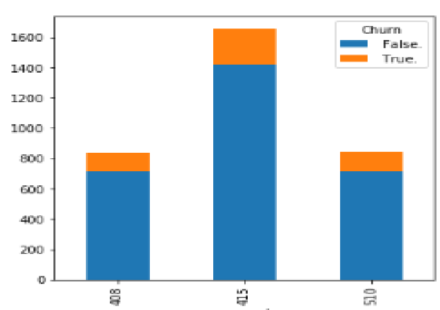
From the above analysis, we found that only 500 users have moved from the subscription plan.

B) Churn by Area Code:

Churn by Area Code

In [374]:
```
traindata.groupby(["area code", "Churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(5,5))
```

Out[374]: <matplotlib.axes._subplots.AxesSubplot at 0x1e2b8e64f60>

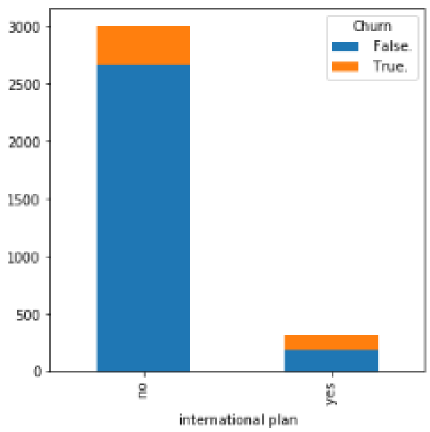


From the above analysis, we found that maximum users from area code 415 moved from the subscription plan.

C)Churn by Customers with International Plan:

Churn By Customers with International plan

```
In [375]: traindata.groupby(["international plan", "Churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(5,5))
Out[375]: <matplotlib.axes._subplots.AxesSubplot at 0x1e2b8f49c50>
```



From the above analysis, we found that very few people have an International plan and half of them have moved from the subscription plan.

5. MACHINE LEARNING MODELS

We are here expected to develop a Model, on our data which will predict whether a customer will Move or no. So here we have to develop model on our train data then implement this model over test data and predict the target variable of test data. The model selection depends upon the dependent variable. The dependent variable can fall in either of the four categories:

1. Nominal
2. Ordinal
3. Interval
4. Ratio If the dependent variable, in our case Move, is Nominal the only predictive analysis that we can perform is Classification, and if the dependent variable is Interval or Ratio the normal method is to do a Regression analysis, or classification after binning. But the dependent variable we are dealing with is Nominal, for which classification model is to be used.

So the classification model we developed here are:

1. C5.0
2. Random Forest.
3. Logistic regression.
4. KNN.
5. Naive Bayes.

For most of model we developed we had followed following steps:

1. Developing model over train data
2. Implementing model on our test data to predict test cases.
3. Building confusion matrix.
4. Calculate the accuracy and FNR.

A) C5.0:

Build model on train data

In [411]:
```
C50model = tree.DecisionTreeClassifier(criterion='entropy').fit(X_train_ind,y_train_dep)
```

Predict test dependent variable

In [412]:
```
C50_predict=C50model.predict(X_test_ind)
```

Generating Confusion Matrix ¶

In [413]:
```
C50matrix = pd.crosstab(y_test_dep, C50_predict)
C50matrix
```

Out[413]:

col_0	0	1
row_0		
0	228	9
1	10	24

Defining TN FN TP FP

In [414]:
```
TP = C50matrix.iloc[0,0]
FN = C50matrix.iloc[1,0]
TN = C50matrix.iloc[1,1]
FP = C50matrix.iloc[0,1]
```

FNR -False Negative Rate

In [415]:
```
(FN*100)/(FN+TP)
```

Out[415]: 4.201680672268908

Accuracy

In [416]:
```
((TP+TN)*100)/(TP+TN+FP+FN)
```

Out[416]: 92.98892988929889

B) Random Forest:

Developing model on test data

In [418]:
```
RF_model = RandomForestClassifier(n_estimators = 40).fit(X_train_ind, y_train_dep)
```

Predicting test dependent variable

In [419]:
```
RF_Predictions = RF_model.predict(X_test_ind)
```

Confusion Matrix

In [420]:
```
RFmatrix = pd.crosstab(y_test_dep, RF_Predictions)
RFmatrix
```

Out[420]:

col_0	0	1
row_0		
0	235	2
1	14	20

Defining TN FN TP FP

In [421]:
```
TP1 = RFmatrix.iloc[0,0]
FN1 = RFmatrix.iloc[1,0]
TN1 = RFmatrix.iloc[1,1]
FP1 = RFmatrix.iloc[0,1]
```

Accuracy

In [422]:
```
((TP1+TN1)*100)/(TP1+TN1+FP1+FN1)
```

Out[422]: 94.09594095940959

FNR

In [423]:
```
(FN1*100)/(FN1+TP1)
```

Out[423]: 5.622489959839357

C) Logistic Regression:

Build Logistic Regression Model

In [429]:
```
logit = sm.Logit(train_logit1['Move'], train_logit1[train_cols]).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.238154
         Iterations 8
```

Predict test data

In [430]:
```
test_logit1['probab'] = logit.predict(test_logit1[train_cols])
test_logit1['ActualVal'] = 1
test_logit1.loc[test_logit1.probab < 0.5, 'ActualVal'] = 0
```

Build Confusion Matrix

In [431]:
```
logitmatrix = pd.crosstab(test_logit1['Move'], test_logit1['ActualVal'])
logitmatrix
```

Out[431]:

ActualVal	0	1
Move		
0.0	232	5
1.0	18	16

Defining TN FN TP FP

In [432]:
```
TP2 = logitmatrix.iloc[0,0]
FN2 = logitmatrix.iloc[1,0]
TN2 = logitmatrix.loc[1,1]
FP2 = logitmatrix.loc[0,1]
```

Accuracy

In [433]:
```
((TP2+TN2)*100)/(TP2+TN2+FP2+FN2)
```

Out[433]: 91.5129151291513

FNR

In [434]:
```
(FN2*100)/(FN2+TP2)
```

Out[434]: 7.2

D) KNN:

Importing libraries for KNN

In [435]:
```
from sklearn.neighbors import KNeighborsClassifier

KNN_model = KNeighborsClassifier(n_neighbors = 5).fit(X_train_ind, y_train_dep)
```

Predict test cases

In [436]:
```
KNN_Predictions = KNN_model.predict(X_test_ind)
```

Confusion Matrix

In [437]:
```
KNNmatrix = pd.crosstab(y_test_dep, KNN_Predictions)
KNNmatrix
```

Out[437]:

col_0	0	1
row_0		
0	232	5
1	26	8

Defining TN FN TP FP

In [438]:
```
TP3 = KNNmatrix.iloc[0,0]
FN3 = KNNmatrix.iloc[1,0]
TN3 = KNNmatrix.iloc[1,1]
FP3 = KNNmatrix.iloc[0,1]
```

Accuracy

In [439]:
```
((TP3+TN3)*100)/(TP3+TN3+FP3+FN3)
```

Out[439]: 88.56088560885608

FNR

In [440]:
```
(FN3*100)/(FN3+TP3)
```

Out[440]: 10.077519379844961

E) Naïve Bayes:

Naive Bayes implementation

In [442]:
```
NBmodel = GaussianNB().fit(X_train_ind, y_train_dep)
```

Predict test cases

In [443]:
```
NBpredictions = NBmodel.predict(X_test_ind)
```

Confusion Matrix

In [444]:
```
NBmatrix = pd.crosstab(y_test_dep, NBpredictions)
NBmatrix
```

Out[444]:

col_0	0	1
row_0		
0	225	12
1	21	13

Defining TN FN TP FP

In [445]:
```
TP4 = NBmatrix.iloc[0,0]
FN4 = NBmatrix.iloc[1,0]
TN4 = NBmatrix.loc[1,1]
FP4 = NBmatrix.loc[0,1]
```

Accuracy

In [446]:
```
((TP4+TN4)*100)/(TP4+TN4+FP4+FN4)
```

Out[446]: 87.82287822878229

FNR

In [448]:
```
(FN4*100)/(FN4+TP4)
```

Out[448]: 8.536585365853659

6.CONCLUSION

For this problem statement we developed 5 models, to predict the target variable. That are:

1. C5.0
2. Random forest
3. Logistic regression
4. KNN
5. Naïve Bayes

- The best False Negative Ratio is achieved with C5.0 and Random Forest which is 4.20 and 5.62 respectively.
- However, the best accuracy is achieved with Random Forest with an accuracy of 94.09 and the second best is C5.0 with an accuracy of 92.98.
- Random Forest and C.50 are the optimal models for our problem.