Churn Reduction

Mohit Sharma

24 Jun 2018

Contents

Chapter 1

Introduction

1.1 Problem Statement

The Problem statement is related to predicting Churning of customer. The objective of this Case is to predict customer behavior, whether a customer moves out of business or not based on customer features. Acquiring new customers can be expensive than retaining present one. With the help of this model we will predict customers who can move out (churn) in future so that company can focus on these customers to retain them.

So, our main focus would be on making correct prediction for customer who can churn in future. Ultimately we have to do churn reduction.

1.2 Data

Our model will be classification, which will classify whether a customer will churn (Churn = True) or not (Churn = False) based on his/her given observation.

Table 1.1: Customer Data (columns 1-7)

state	account length	area code	phone number	international plan	voice mail plan	number vmail messages
KS	128	415	382-4657	no	yes	25
OH	107	415	371-7191	no	yes	26
NJ	137	415	358-1921	no	no	0
OH	84	408	375-9999	yes	no	0
OK	75	415	330-6626	yes	no	0

Table 1.2: Customer Data (columns 8-14)

total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes
265.1	110	45.07	197.4	99	16.78	244.7
161.6	123	27.47	195.5	103	16.62	254.4
243.4	114	41.38	121.2	110	10.3	162.6
299.4	71	50.9	61.9	88	5.26	196.9
166.7	113	28.34	148.3	122	12.61	186.9

Table 1.3: Customer Data (columns 15-21)

total night calls	total night charge	total intl minutes	total intl calls	total intl charge	number customer service calls	Churn
91	11.01	10	3	2.7	1	False.
103	11.45	13.7	3	3.7	1	False.
104	7.32	12.2	5	3.29	0	False.
89	8.86	6.6	7	1.78	2	False.
121	8.41	10.1	3	2.73	3	False.

We have total 20 independent variables (IV) in blue shades and 1 dependent variable (DV) in orange shade in below table. Total number of observations in training dataset is 3333.

Table 1.4: Column names of Dataset

Independent Variables	state	total eve minutes
	account length	total eve calls
	area code	total eve charge
	phone number	total night minutes
	international plan	total night calls
	voice mail plan	total night charge
	number vmail messages	total intl minutes
	total day minutes	total intl calls
	total day calls	total intl charge
	total day charge	number customer service calls
Dependent variable	Churn	

* intl stands for international, vmail stands for voice mail, eve stands for evening and rest have their usual meaning.

Our dataset is customer details of a Phone connection providing company and details are given for customers, whether they took voice mail plan or not, whether they took international plan or not, how much they did call, how much they were charged in day time, evening time, night time and during international calls. In churn columns we have false and true. False means that customer did not move out and true means that customers moves out from the company business. So, our main goal would be on getting correct prediction for customer whether he/she will churn out or not for test dataset which contain same features.

Chapter 2

Methodology

2.1 Data Preprocessing: (Exploratory Data Analysis)

While building a model, there is a famous quote "**Garbage in Garbage out**". If we have our best model and we feed our data to that model, then it is not guaranteed that model will perform its best. As our data may have lots of noisy data (Garbage) and model will also follow noisy data and thus can produce wrong result because of that noisy data. We cannot remove noise/error completely from our data but we can reduce it with the help of EDA (Exploratory Data Analysis).

EDA involves getting summary of data with numerical statistics and Graphical visualization.

2.1.1 Understanding the Data:

Datatype: First we will look at the data type of our variables. Below is the list:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
state                            3333 non-null object
account length                   3333 non-null int64
area code                        3333 non-null int64
phone number                     3333 non-null object
international plan               3333 non-null object
voice mail plan                  3333 non-null object
number vmail messages            3333 non-null int64
total day minutes                3333 non-null float64
total day calls                  3333 non-null int64
total day charge                 3333 non-null float64
total eve minutes                3333 non-null float64
total eve calls                  3333 non-null int64
total eve charge                 3333 non-null float64
total night minutes              3333 non-null float64
total night calls                3333 non-null int64
total night charge               3333 non-null float64
total intl minutes               3333 non-null float64
total intl calls                 3333 non-null int64
total intl charge                3333 non-null float64
number customer service calls    3333 non-null int64
Churn                            3333 non-null object
dtypes: float64(8), int64(8), object(5)
memory usage: 546.9+ KB
```

** Python code to generate above result

Here we have 5 categorical variables (showing with object data type) and 16 numerical variables (float and int data type). Area code is category with int64 data type.

Let us first analyze our numerical data. For analyzing we would see statistical summary of our numerical data in below tables:

Statistics of numerical data:

Table 2.1: Numerical data Statistics (columns 1-6)

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge
count	3333	3333	3333	3333	3333	3333
mean	101.064806	437.18242	8.09901	179.7751	100.43564	30.56231
std	39.822106	42.37129	13.688365	54.467389	20.069084	9.259435
min	1	408	0	0	0	0
25%	74	408	0	143.7	87	24.43
50%	101	415	0	179.4	101	30.5
75%	127	510	20	216.4	114	36.79
max	243	510	51	350.8	165	59.64

Table 2.2: Numerical data Statistics (columns 7-13)

	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes
count	3333	3333	3333	3333	3333	3333	3333
mean	200.98035	100.11431	17.08354	200.87204	100.10771	9.039325	10.23729
std	50.713844	19.922625	4.310668	50.573847	19.568609	2.275873	2.79184
min	0	0	0	23.2	33	1.04	0
25%	166.6	87	14.16	167	87	7.52	8.5
50%	201.4	100	17.12	201.2	100	9.05	10.3
75%	235.3	114	20	235.3	113	10.59	12.1
max	363.7	170	30.91	395	175	17.77	20

Table 2.3: Numerical data Statistics (columns 14-16)

	total intl calls	total intl charge	number customer service calls
count	3333	3333	3333
mean	4.47945	2.76458	1.562856
std	2.46121	0.75377	1.315491
min	0	0	0
25%	3	2.3	1
50%	4	2.78	1
75%	6	3.27	2
max	20	5.4	9

`** Python code to generate above result`

Analysis:

From above table we can see that minimum value among all numerical columns is 0 and maximum is 395. It means our data is in same range for all our numerical columns. Our machine algorithms, which are based on distance between two points, are get affected by large difference is range of values as higher values dominate the lower values while calculating distance. But in our dataset that is not a problem. Mean of eight columns values are in hundreds and rest having mean less than hundred.

Checking categorical data:

Finding unique values in each category:

```
state
51
area code
3
phone number
3333
international plan
2
voice mail plan
2
Churn
2
```

** Python code to generate above result

Counting of each unique values in categorical variables area code, international plan, voice mail plan and churn

```
area code
415    1655
510     840
408     838
Name: area code, dtype: int64
=================================
international plan
 no     3010
 yes     323
Name: international plan, dtype: int64
=================================
voice mail plan
 no     2411
 yes     922
Name: voice mail plan, dtype: int64
=================================
Churn
 False.    2850
 True.      483
Name: Churn, dtype: int64
```

** Python code to generate above result

Getting percentage of each target class in column Churn

```
Churn
False.    0.855086
True.     0.144914
Name: Churn, dtype: float64
```

** Python code to generate above result

Analysis:

So, we have 51 unique values in state, so it would be less effective to feed it into model, also while making dummy variables we would maximize the dimension of dataset and would be fall in curse of dimensionality. So it is better to drop state column. For state alternative we have area code and it has only three categories so it would be good to take consideration of area code rather than state. For area code and other categories we will do statistical test in feature selection section and will see if they are important or not.

Also we have less observation for our churn true class only 14.5% observations are related to this class. Generally, if minor class is less than 5% then there is serious issue of target imbalance. In our case we have 14.5% data but we may have issue of target imbalance. Will took this in consideration and will try to improve our model by removing target imbalance at the end. Phone numbers have all different values, which is obvious so we would drop this variable as it would not give any improvement to our model. Rest categorical variable has only two categories, so we would use them if they are important to our model as per statistical test in feature selection section.

2.1.2 Missing value analysis:

In our dataset we are lucky that we don't have any missing values. In case we have missing values then we should impute it using different method mean, median, KNN, linear regression etc.

Checking missing values for each column in our dataset:

```
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
number customer service calls     0
Churn                             0
dtype: int64
```

** Python code to generate above result

<u>Analysis:</u>

We don't have any missing values in our dataset.

2.1.3 Outlier Analysis:

Outlier detection and treatment is always a tricky part especially when our dataset is small. The box plot method detects outlier if any value is present greater than (**Q3 + (1.5 * IQR)**) or less than (**Q1 – (1.5 * IQR)**)

Q1 > 25% of data are less than or equal to this value
Q2 or Median -> 50% of data are less than or equal to this value
Q3 > 75% of data are less than or equal to this value
IQR(Inter Quartile Range) = Q3 – Q1

So, Boxplot method find approx. 1 % of data as outliers. It looks fine if we think only 1% of data we are treating as outlier and no impact would be after removal of outlier. Then we could be wrong.

Outlier should be treated in well manner as after removing outlier, we may be in a situation in which we lost important information which was required to correct prediction. For a small dataset, if we found outlier with the help of boxplot method, then should we assume it as a outlier? Let us try to find out in our case.

First we need to look at nature of outliers, is it really a outlier, outlier which occur due to human error measurement?

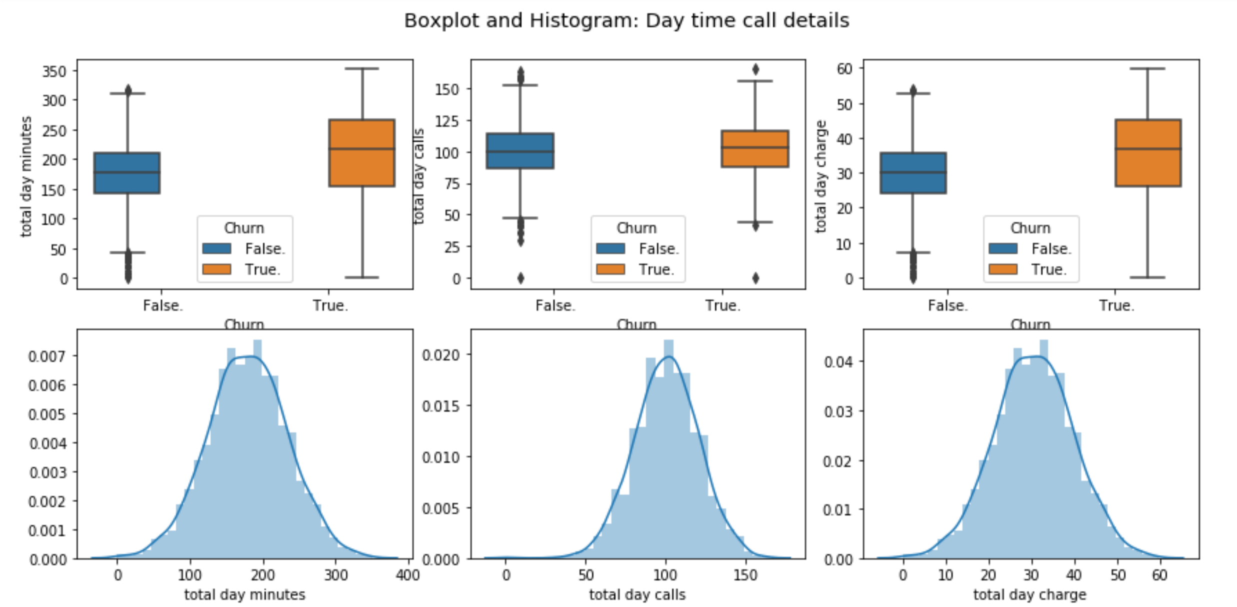
We have small dataset and there is more possibility of a customer is consuming high data, he/she is doing lots of calls than other customer and just because it has high values than others, boxplot will consider it as outlier but in actual it is information for us.

After removing outliers we may have even more small dataset, and by nature of our dataset, it doesn't feel like there could be more chances of human error while recording dataset. It is detail of customers of a company rather than a survey.

So, we will experiment with our models and will feed them different data, one with whole dataset and one after treating outliers. After that we would look at performance of our model, as we don't want to lose any information by treating it as outlier. Further, tree based algorithm are insensitive to outliers as compared to other models. So, our tree based model could give us better result without losing any information.

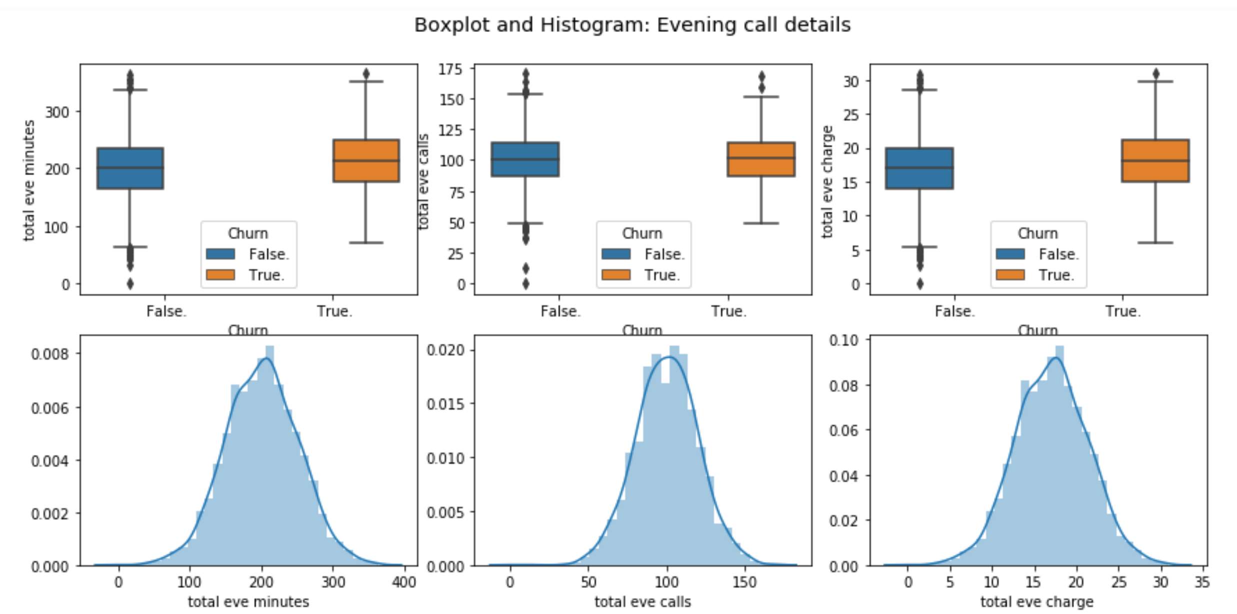
Let us now try to find out outliers and distribution of each variable:

Boxplot and distribution of total day minutes, calls and charge is plotted in below figures.

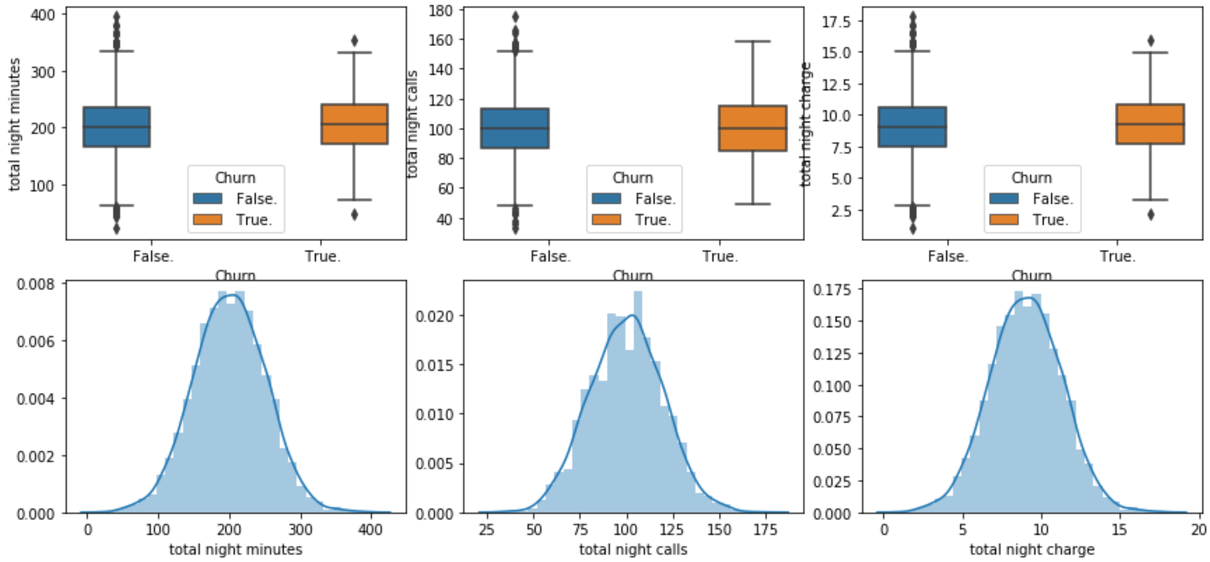


** [Python code](#) to generate above result

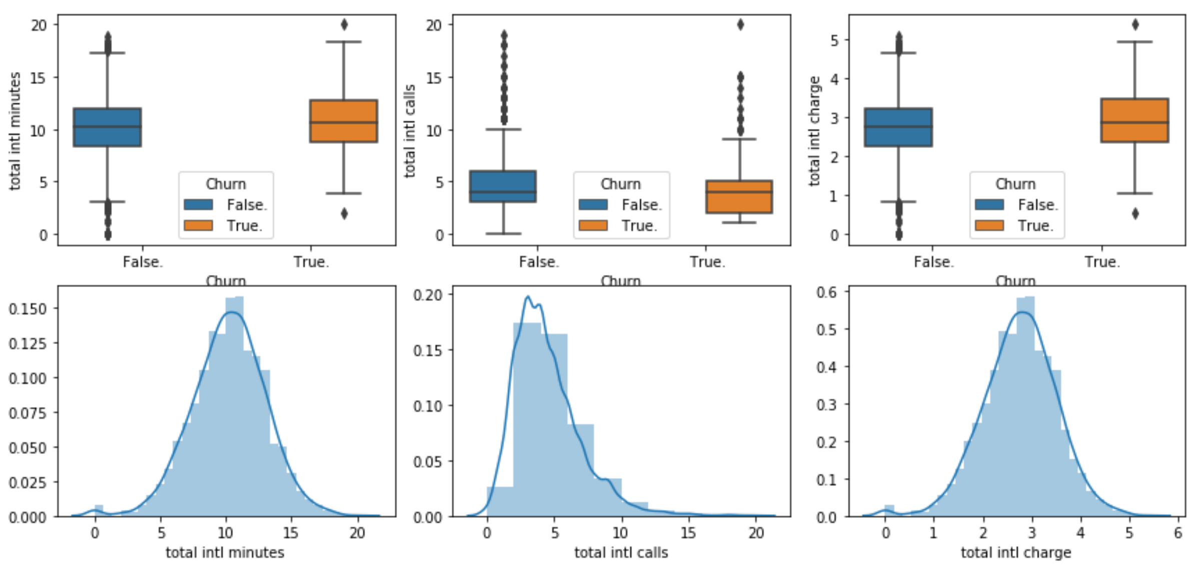
Same distribution and boxplot for Evening, night and international call detail



Boxplot and Histogram: Night call details

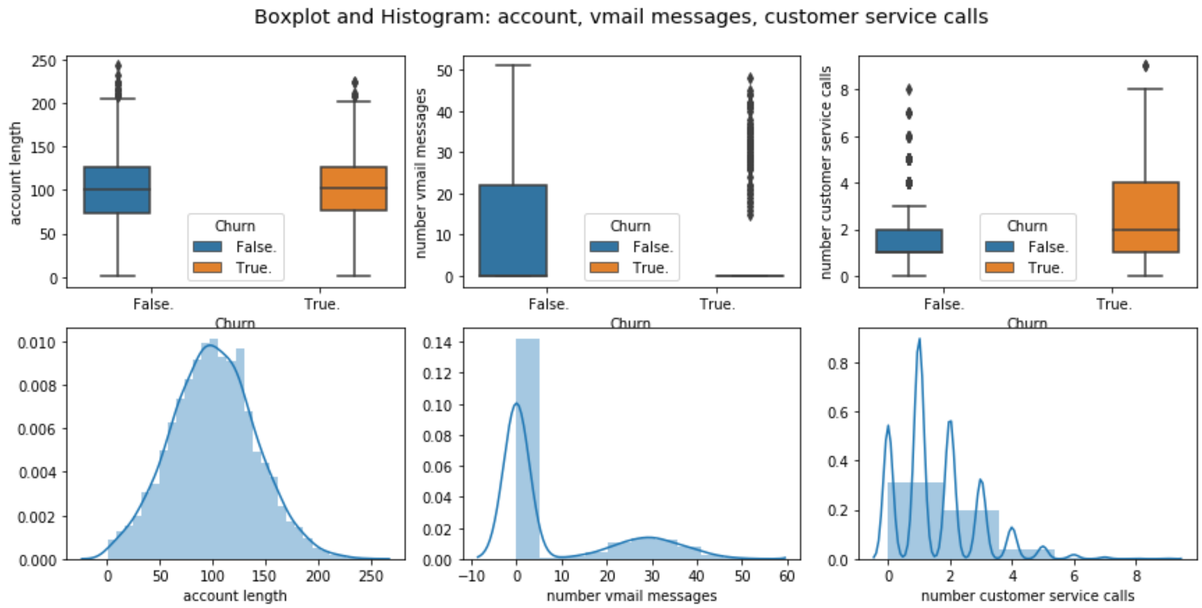


Boxplot and Histogram: International call Details



** [Python code](#) to generate above result

Distribution and boxplot for account length, service call and vmail messages data:



** [Python code](#) to generate above result

Analysis:

For three columns total day minutes, total eve minutes and total intl minutes, we can see that outliers are present at both side and distribution is almost normal. Outliers may be showing just because, for low minutes and high minutes, we have less observations and maximum data is clustered at intermediate values.

Another observation we get from above figures, distribution is exact same for minutes and charges for day, evening and international call. That is obvious charges would be some multiple of minutes. So, We will look at this thing at feature selection section for case of mulitcollinearity.

For 'vmail messages' we have bimodal distribution, which is because most customers have 0 values so there is a strong peak at 0 and normal distribution for values other than zero. In 'number of customer service calls' also we have lots of zeroes.

So, we would now remove outliers and would make another copy of dataframe. So, that we could feed our model with different dataset one with outliers and one without outliers as here in our dataset outliers seems to be information of high and low usages customers.

Removal of outliers as per boxplot method:

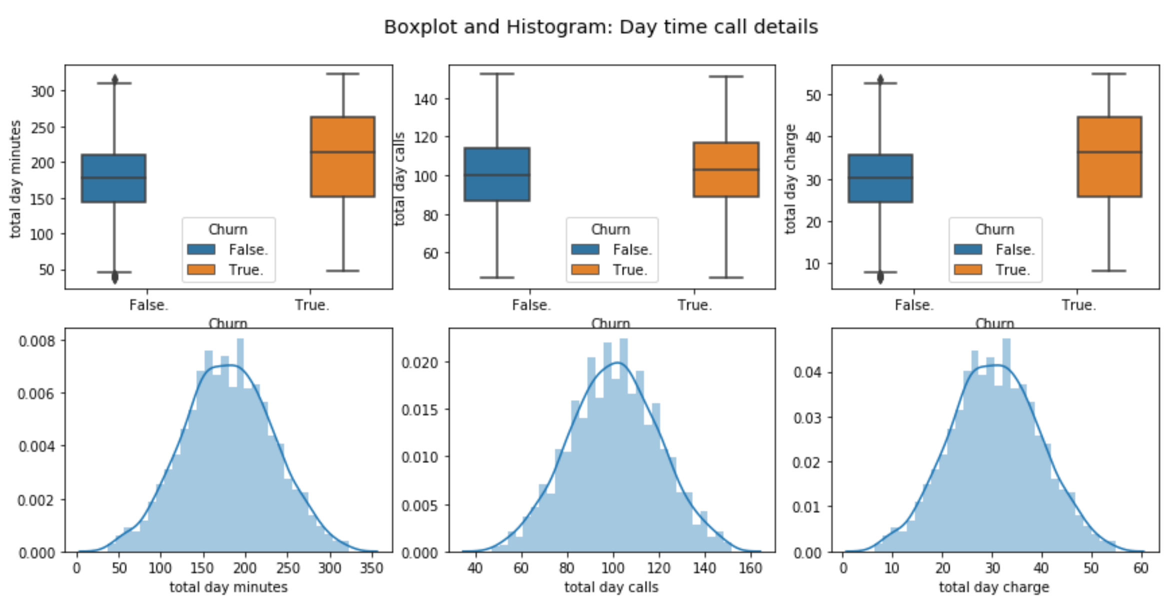
We would remove outliers from following features:

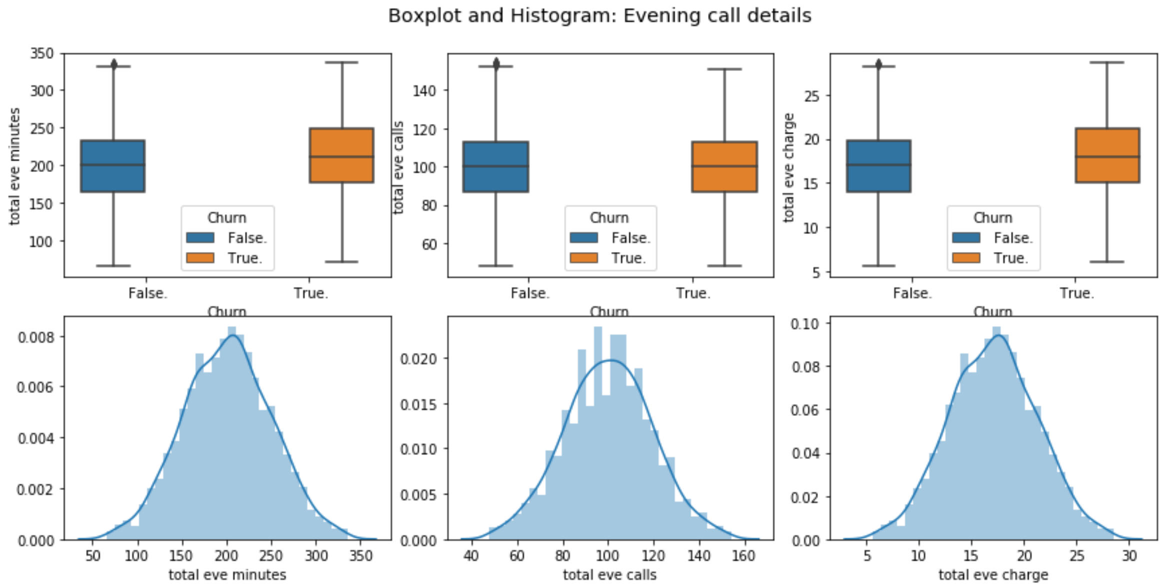
'account length', 'total day minutes', 'total day calls', 'total day charge', 'total eve minutes', 'total eve calls', 'total eve charge', 'total night minutes', 'total night calls', 'total night charge', 'total intl minutes', 'total intl charge', 'total intl calls'

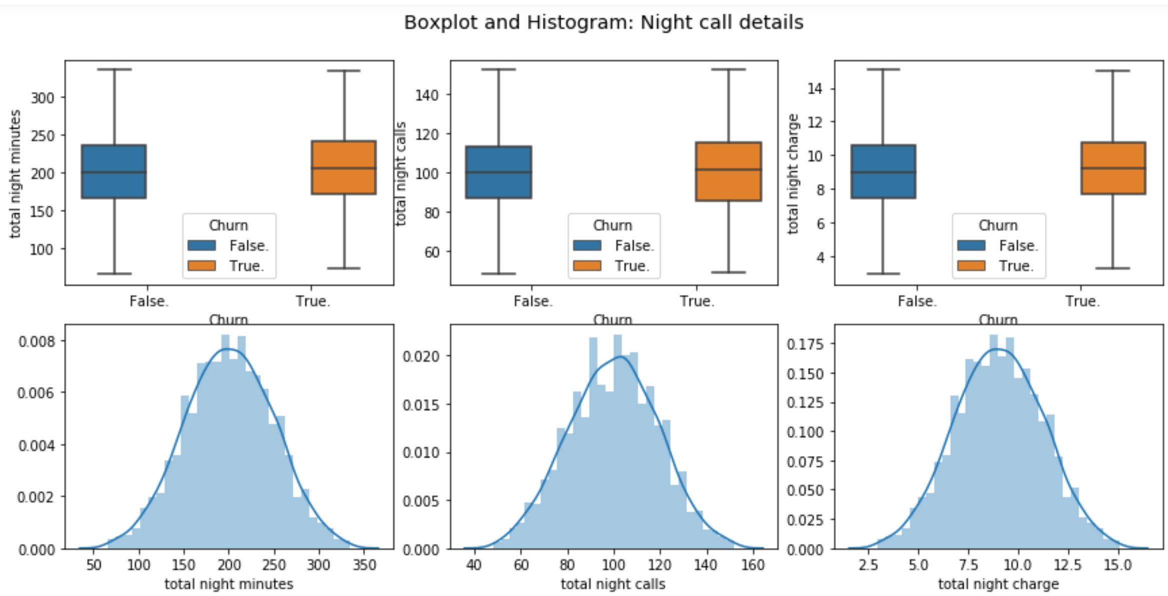
We are not going to outlier removal on number of customer service calls and number of vmail message as both column has more number of zeros and median is more centered towards zero that is why showing far point as outliers.

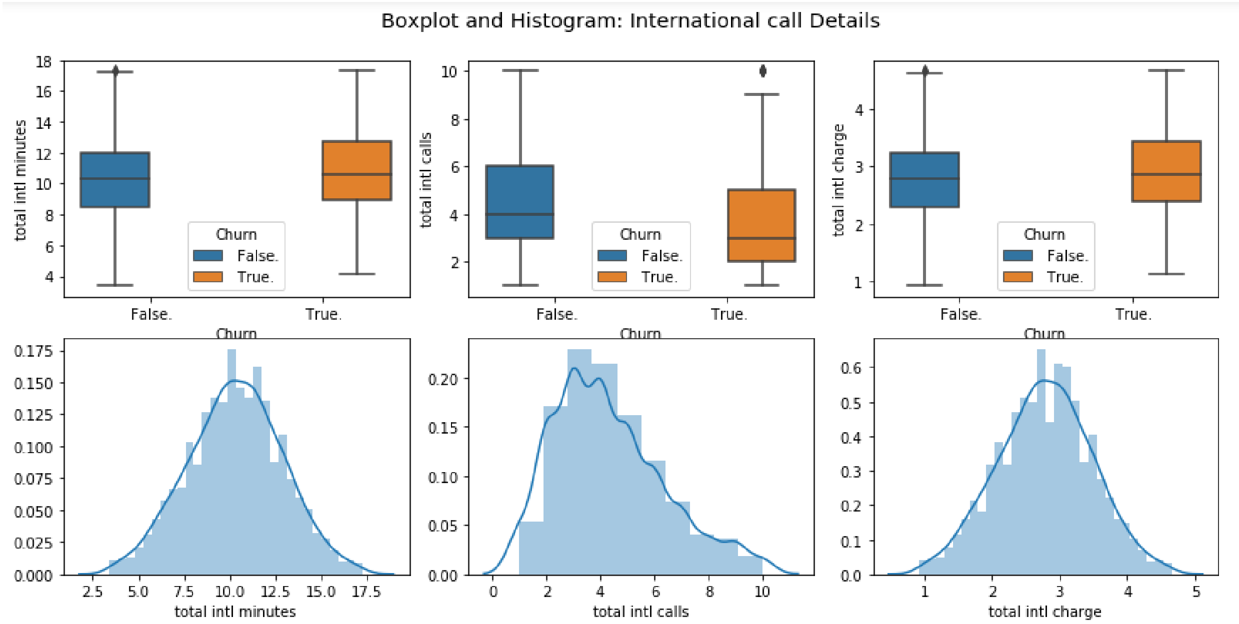
** Python code to remove outlier

After removal of outliers, distribution is as follow:




Boxplot and Histogram: Evening call details
total eve minutes
total eve calls
total eve charge
Churn
False.
True.
total eve minutes
total eve calls
total eve charge


Boxplot and Histogram: Night call details
total night minutes
total night calls
total night charge
Churn
False.
True.
total night minutes
total night calls
total night charge



** [Python code](#) to generate above result

<u>Analysis:</u>

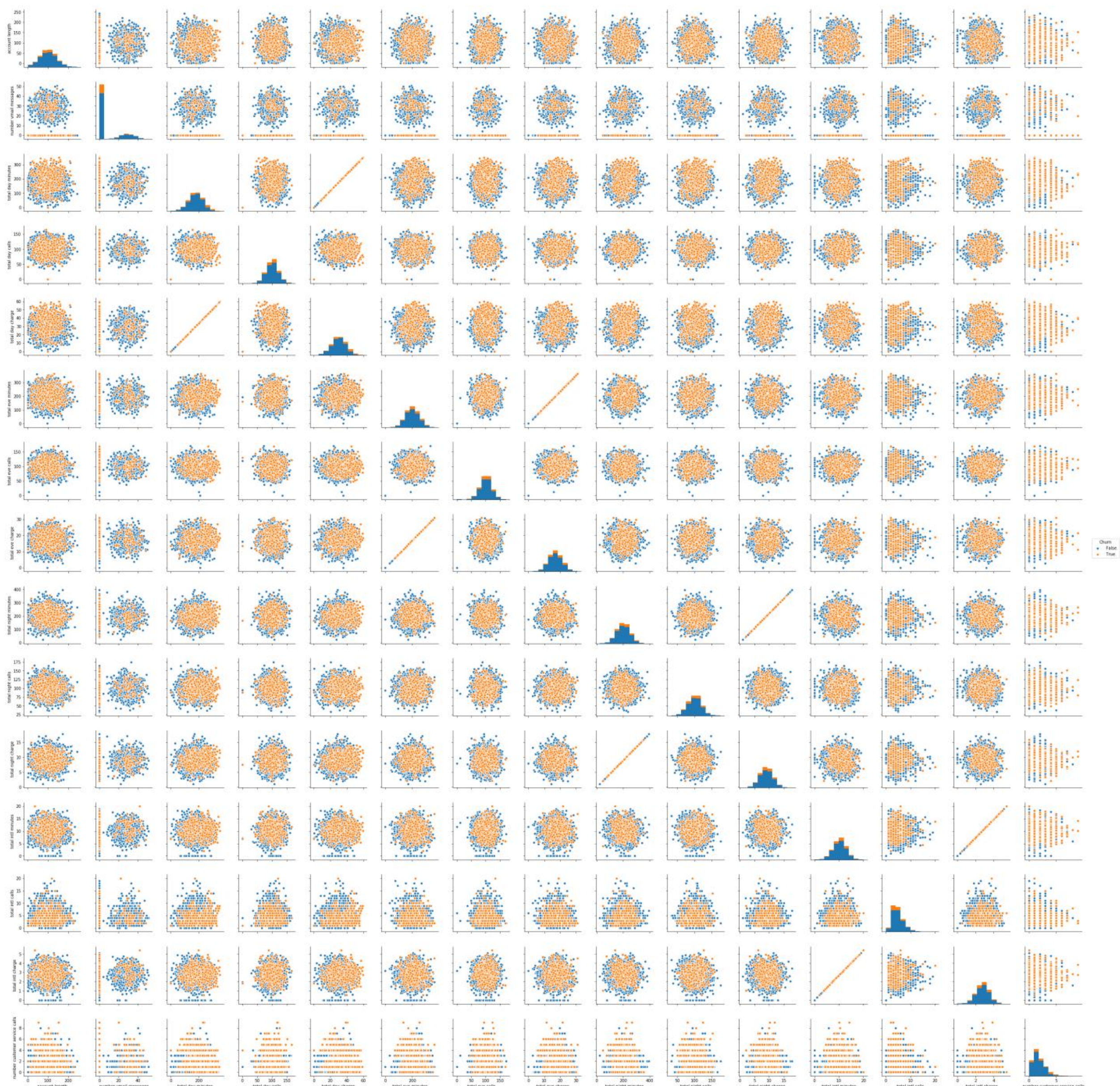
Now, it is showing less or no outliers. After removal of outlier still it is showing outliers, that is because outliers are based on distance from median and after removing outliers median shifted towards center thus distance for other points increase from median and showing points at end points as outlier. But the outliers are very less, so we will not going to do further outlier removal process.

2.1.4 Feature selection:

For selecting features, first we will look at correlation between independent variable. If two variable carrying same information then we would drop one of those as, model performance decreases for multicollinearity.

<u>Let us first look at numerical columns:</u>

We will check scatterplot of all numerical variables with each other:



[Link for Correlation plot image](#)

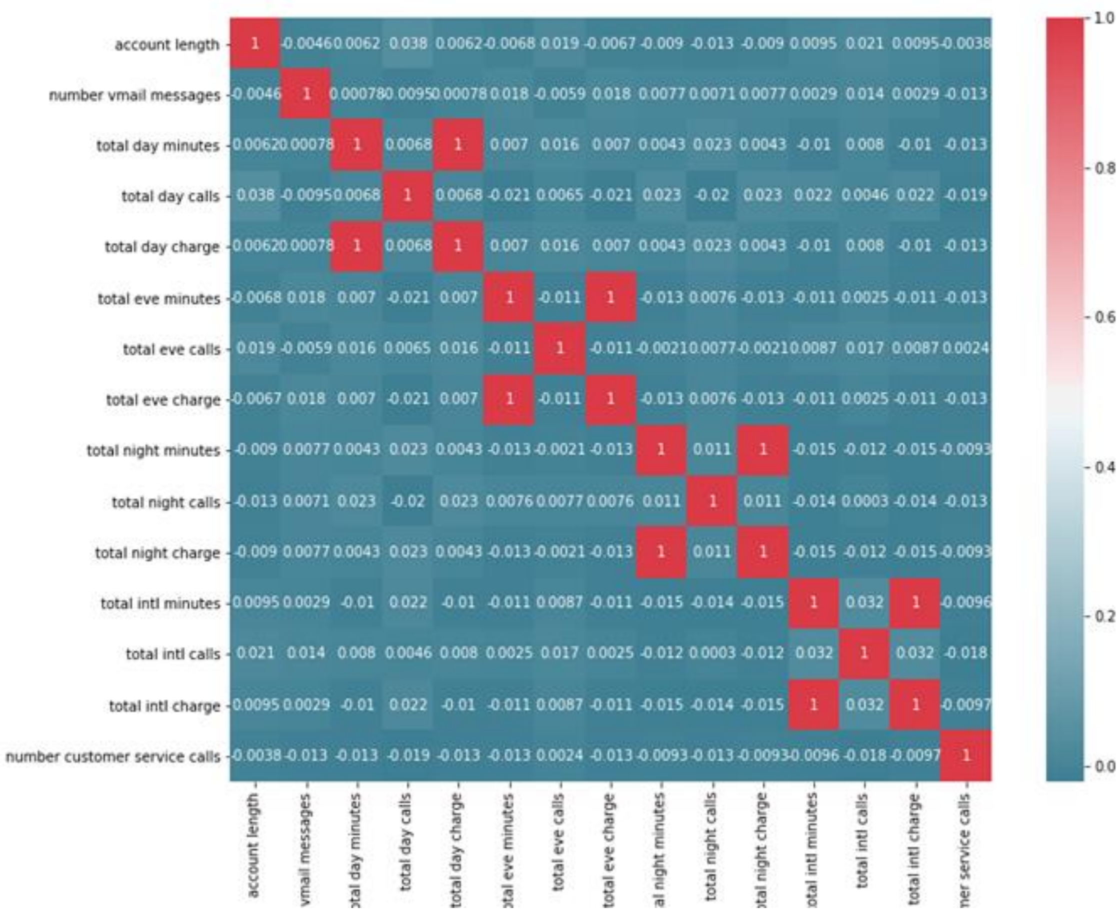
** [Python code](#) to generate above result

As we can see in scatterplot there is strong collinearity between few columns, there is straight line in scatterplot for four pair of columns. Image is in compact form, column names are not visible so link to image is given above to open and zoom it to view column names in detail. Except these four pair, there is not serious multicollinearity between our numerical variables.

Columns which are showing multicollinearity :

- Total day Minutes and total day charges
- Total eve minutes and total eve charges
- Total night minutes and total night charges
- Total intl minutes and total intl charges

Let us analyze the same thing with correlation value in heatmap.



** [Python code](#) to generate above result

Analysis:

From scatterplot and heatmap plot, we can observe that there is exact linear relationship between charges and minute for each column of day, evening, night and international call. So, both charges and minute columns containing same information. So we would remove one of those for all case.

We would remove one of columns from minutes and change for all case as per importance of features.

Let us analyze for categorical variables also:

We have four categorical variable state, area code, voice mail plan and international plan.

Let us first check these categorical variables that how much Churn variable is dependent on other categorical variables. For this we would do chi-square (test of independence) test between Churn column and these four columns:

```
Chi-square - test of independence
=================================
p-value between Churn and state
0.002296221552011188
----------------------------
p-value between Churn and area code
0.9150556960243712
----------------------------
p-value between Churn and international plan
2.4931077033159556e-50
----------------------------
p-value between Churn and voice mail plan
5.15063965903898e-09
----------------------------
```

** Python code to generate above result

Analysis:

From chi-sq test of independence, we have very less p-value than 0.05 for voice mail plan and international plan with Churn variable, so for both these case we have enough evidence to reject null hypothesis and accepting alternate hypothesis, that 'Churn' prediction is dependent on 'voice mail plan' and 'international plan' .

But for area code we have large p value saying that, Churn and 'area code' are independent as we failed to reject null hypothesis. So we would drop 'area code' from our dataset.

For state, we can accept if we took critical value at 0.05 p-value, but we have total of 51 categories in state and while creating dummy variables we would have 50 columns and there would be curse of dimensionality. So we would drop state from our dataset.

Let us check whether 'voice mail plan' and 'international plan' are independent or not?

For this also we use chi-sq test of independence test between them.

```
p-value between international plan  and voice mail plan
0.7784680822485827
----------------------------
```

** Python code to generate above result

Analysis:

Here, we have enough p-value for which we failed to reject the null hypothesis. So **'voice mail plan' and 'international plan' are independent each other and does not have multicollinearity.**

So, we would put both columns in our final dataset.

Now, let us check importance of our numerical variable in Churn prediction:

Table 2.1 :- Important Features

	Feature	importance
0	total day charge	0.132741
1	number customer service calls	0.128364
2	total day minutes	0.128222
3	international plan	0.074672
4	total eve minutes	0.060331
5	total eve charge	0.059080
6	total intl calls	0.056556
7	total intl minutes	0.048008
8	total intl charge	0.047555
9	total night minutes	0.040644
10	total night charge	0.039843
11	total day calls	0.038501
12	account length	0.036908
13	total night calls	0.036007
14	total eve calls	0.035044
15	voice mail plan	0.019351
16	number vmail messages	0.018169

** Python code to generate above result

Let us check VIF value for numerical columns:

const	**142.9**
account length	**1.0**
number vmail messages	**1.0**
total day minutes	**10474222.2**
total day calls	**1.0**
total day charge	**10474226.8**
total eve minutes	**2236930.8**
total eve calls	**1.0**
total eve charge	**2236932.0**
total night minutes	**638715.2**
total night calls	**1.0**
total night charge	**638713.8**
total intl minutes	**69016.5**
total intl calls	**1.0**
total intl charge	**69017.2**
number customer service calls	**1.0**
dtype: float64	

** Python code to generate above result

Analysis:

So, we have important feature in descending order. So, we would remove columns 'total day minutes', 'total eve charge', 'total night charge', 'total intl charge' as these are multicollinear with other columns. As we have less features so we will not remove any other column based on above table. We have VIF values more than 10 indicating multicollinearity.

Let us check again VIF value after removal of mulitcollinear columns.

VIF should be less than 10 for multicollinearity.

const	**140.5**
account length	**1.0**
number vmail messages	**1.0**
total day calls	**1.0**
total day charge	**1.0**
total eve minutes	**1.0**
total eve calls	**1.0**
total night minutes	**1.0**
total night calls	**1.0**
total intl minutes	**1.0**
total intl calls	**1.0**
number customer service calls	**1.0**
dtype: float64	

** Python code to generate above result

Const is not part of our dataset, it is added for calculation of VIF. So as per VIF values, now we don't have multicollinearity in our dataset.

2.1.5 Feature Scaling:

Our dataset has all values in almost same range, so we don't require feature scaling for this dataset. Feature scaling is important for those variables for which their values are too high and too low. Algorithm which uses distance method, are affected by out of range values. Higher value dominates the lesser values in calculating distance. But in our case min value is 0 and maximum is 351.

So, Feature scaling is not requiring for this case.

2.1.6 Data After EDA

Now, let us look at final dataset. All steps involved in data preprocessing can be find here.

As we are going to take consideration of two dataset one with outliers and other without outliers. Our dataset is small so there would be no issue for memory consumption and performance.

Dataset with outliers: **churn_data_df**

(Dropped columns 'state', 'area code', 'phone number', 'total day minutes', 'total eve charge', 'total night charge', 'total intl charge'. Categorical values changed to numeric levels of category**)**

```
churn_data_df.head()
```

	account length	international plan	voice mail plan	number vmail messages	total day calls	total day charge	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	total intl calls	number customer service calls	Churn
0	128	0	1	25	110	45.07	197.4	99	244.7	91	10.0	3	1	0
1	107	0	1	26	123	27.47	195.5	103	254.4	103	13.7	3	1	0
2	137	0	0	0	114	41.38	121.2	110	162.6	104	12.2	5	0	0
3	84	1	0	0	71	50.90	61.9	88	196.9	89	6.6	7	2	0
4	75	1	0	0	113	28.34	148.3	122	186.9	121	10.1	3	3	0

Dataset without outliers: **churn_data_df_wo**

(Dropped columns 'state', 'area code', 'phone number', 'total day minutes', 'total eve charge', 'total night charge', 'total intl charge' and removal of outliers as per boxplot method. Categorical values changed to numeric levels of category**)**

```
churn_data_df_wo.head()
```

	account length	international plan	voice mail plan	number vmail messages	total day calls	total day charge	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	total intl calls	number customer service calls	Churn
0	128	0	1	25	110	45.07	197.4	99	244.7	91	10.0	3	1	0
1	107	0	1	26	123	27.47	195.5	103	254.4	103	13.7	3	1	0
2	137	0	0	0	114	41.38	121.2	110	162.6	104	12.2	5	0	0
4	75	1	0	0	113	28.34	148.3	122	186.9	121	10.1	3	3	0
5	118	1	0	0	98	37.98	220.6	101	203.9	118	6.3	6	0	0

Shape of both dataset

```
print(churn_data_df.shape)
print(churn_data_df_wo.shape)
```

```
(3333, 14)
(3057, 14)
```

2.2 Modeling

We will now build our models. Before proceeding please look at below key terms to avoid any confusion in next steps.

- churn_data_df: training dataset containing all observations.
- churn_data_df: training dataset , containing observation which left after outlier removal
- X_train : containing independent variables of churn_data_df
- y_train: containing dependent variable (Churn) of churn_data_df
- X_train_wo : containing independent variables of churn_data_df_wo (without outliers)
- y_train_wo : containing dependent variable (Churn) of churn_data_df_wo
- X_resampled : dataset after applying SMOTE + Tomek oversampling process on churn_data_df. It is only containing independent variables.
- y_resampled : dataset after applying SMOTE + Tomek oversampling process on churn_data_df. It is only containing dependent variable (Churn).
- X_resampled_wo : dataset after applying SMOTE + Tomek oversampling process on churn_data_df_wo. It is only containing independent variables.
- y_resampled_wo : dataset after applying SMOTE + Tomek oversampling process on churn_data_df_wo. It is only containing dependent variable (Churn).

2.2.1 K-fold CV and GridsearchCV

Before building models on our dataset, we would like to explore two things:

- K-fold cross validation
- GridSearchCV

K-fold Cross Validation:

K-fold cross validation is used to check performance of model which is checked on K different test dataset. Let us assume, we have built a model and we are checking performance of our model on a test data and our model show accuracy of 95% and now we will check our model on different test data and now accuracy is 80%. So what should we consider for deciding model performance? So in this, K-fold cross validation helps, it would divide our training data in k sets and will build a model using k-1 training set and one left set would be used to test our model performance. In this way it would build k times model and each time there would be different test dataset to check performance and at the end all k model's accuracy mean value would be considered as model accuracy.

So, we would use K-fold cross validation technique to get performance of our model.

GridsearchCV : (Hyperparameter tuning)

Hyperparameter are the parameters which we pass as argument to our building function, like kernel, criterion, n_estimators etc. So to get best values of these gridserchcv is used. In this technique, we make list of these different parameters and then gridsearchcv build model for every combination of these parameters and then check crossvalidation score and based on score it gives the best combination of hyperparameters.

And then we can build our model with the values of hyperparameter given by GridSearchCV.

This is called performance tuning and we would use this to tune our model.

2.2.2 Building models

Models and performance of models:

We will build one by one all models and will check performance of our model and then at the will decide for which model we should go.

Logistic Regression:

Performance of Logistic Regression model (K-fold CV and score on test dataset), while model is trained on churn_data_df i.e. with outliers.

```
K-fold cross validation score of model for k = 10 is :
0.860186833540127
====================================
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.89      0.98      0.93      1443
          1       0.61      0.18      0.28       224

avg / total       0.85      0.87      0.84      1667

====== Confusion matrix =======
[[1417   26]
 [ 184   40]]
```

** [Python code] to generate above result

Performance of Logistic Regression model (K-fold CV and score on test dataset), while model is trained on churn_data_df_wo i.e. without outliers.

```
K-fold cross validation score of model for k = 10 is :
0.8609883496304487
====================================
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.89      0.98      0.93      1443
          1       0.61      0.18      0.28       224

avg / total       0.85      0.87      0.84      1667

====== Confusion matrix =======
[[1417   26]
 [ 183   41]]
```

** [Python code] to generate above result

Analysis:

We got almost same performance for both dataset i.e. 86% K- fold accuracy. Logistic regression gives better result for linearly separable data. Let us try other model also and at the end we would decide our model based on our performance. 183 observations are predicted as churning False and in actual these observation were Churning as True.

KNN(K – Nearest Neighbors)

Performance of KNN model (K-fold CV and score on test dataset), while model is trained on churn_data_df i.e. with outliers.

```
K-fold cross validation score of model for k = 10 is :
0.8517874161586738
====================================
Model performance on test dataset
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.87      0.98      0.92      1443
          1       0.40      0.08      0.13       224

avg / total       0.81      0.86      0.82      1667

====== Confusion matrix =======
[[1418   25]
 [ 207   17]]
```

** Python code to generate above result

Performance of KNN model (K-fold CV and score on test dataset), while model is trained on churn_data_df_wo i.e. without outliers.

```
K-fold cross validation score of model for k = 10 is :
0.8524680174129067
====================================
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.87      0.98      0.92      1443
          1       0.38      0.07      0.11       224

avg / total       0.80      0.86      0.81      1667

====== Confusion matrix =======
[[1418   25]
 [ 209   15]]
```

** Python code to generate above result

Analysis:

We got almost same performance for both dataset i.e. 85% K-fold accuracy and slightly less accurate than logistic regression. 207 observations are predicted as churning False and in actual these observations having Churning as True.

Naïve Bayes

Performance of Naïve Bayes model (K-fold CV and score on test dataset), while model is trained on churn_data_df i.e. with outliers.

```
K-fold cross validation score of model for k = 10 is :
0.8514835194475913
====================================
Model performance on test dataset
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.91      0.93      0.92      1443
          1       0.47      0.40      0.43       224

avg / total       0.85      0.86      0.85      1667

====== Confusion matrix =======
[[1342  101]
 [ 135   89]]
```

** Python code to generate above result

Performance of Naïve Bayes model (K-fold CV and score on test dataset), while model is trained on churn_data_df_wo i.e. without outliers.

```
K-fold cross validation score of model for k = 10 is :
0.8501772631944859
====================================
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.91      0.93      0.92      1443
          1       0.47      0.39      0.43       224

avg / total       0.85      0.86      0.85      1667

====== Confusion matrix =======
[[1343  100]
 [ 136   88]]
```

** Python code to generate above result

Analysis:

We got almost same performance for both dataset i.e. 85% K-fold accuracy and slightly less accurate than logistic regression. 135 observations are predicted as churning False and in actual these observations having Churning as True.

Decision tree:

Performance of Decision Tree model (K-fold CV and score on test dataset), while model is trained on churn_data_df i.e. with outliers.

```
K-fold cross validation score of model for k = 10 is :
0.9225899552246858
====================================
Model performance on test dataset
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.95      0.97      0.96      1443
          1       0.78      0.71      0.74       224

avg / total       0.93      0.93      0.93      1667

====== Confusion matrix =======
[[1399   44]
 [  66  158]]
```

** Python code to generate above result

Performance of Decision Tree model (K-fold CV and score on test dataset), while model is trained on churn_data_df_wo i.e. without outliers.

```
K-fold cross validation score of model for k = 10 is :
0.9196283631370294
====================================
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.95      0.94      0.95      1443
          1       0.67      0.71      0.69       224

avg / total       0.92      0.91      0.91      1667

====== Confusion matrix =======
[[1363   80]
 [  65  159]]
```

** Python code to generate above result

Analysis:

We got 92% K-fold accuracy for dataset with outliers and 91% K-fold accuracy for dataset without outliers. We got slightly higher performance for dataset with outliers than dataset without outliers. 65 observations are predicted as churning False and in actual these observations having Churning as True.

So, our decision to not drop outliers is good and Decision tree outperform than other models which we looked before.

Random forest:

Performance of Random Forest model (K-fold CV and score on test dataset), while model is trained on churn_data_df i.e. with outliers.

```
K-fold cross validation score of model for k = 10 is :
0.9342890794986604
====================================
Model performance on test dataset
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.95      1.00      0.97      1443
          1       0.96      0.63      0.76       224

avg / total       0.95      0.95      0.94      1667

====== Confusion matrix =======
[[1437    6]
 [  82  142]]
```

** Python code to generate above result

Performance of Random Forest model (K-fold CV and score on test dataset), while model is trained on churn_data_df_wo i.e. without outliers.

```
K-fold cross validation score of model for k = 10 is :
0.9327357086061344
====================================
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.94      0.99      0.97      1443
          1       0.93      0.61      0.74       224

avg / total       0.94      0.94      0.94      1667

====== Confusion matrix =======
[[1432   11]
 [  87  137]]
```

** Python code to generate above result

Analysis:

We got slightly high performance for dataset with outliers than dataset without outliers. So, again our decision to not to drop outliers is good. And Random forest classifier outperforms than all other models. 82 observations are predicted as churning False and in actual these observations having Churning as True.

2.2.3 Hyperparameter tuning:

Hyperparameter tuning is used to find optimum values of arguments used in building models like n_estimators, max_depth, kernel etc. so that we could gain better result with these tuned parameter. So we will do hyperparameter tuning for two models which gave us accuracy more than 90% i.e. Decision Tree Classifier and Random Forest.

Decision Tree Model Hyperparameter tuning:

Let us tune decision tree for following parameters on dataset with outliers i.e. churn_data_df

```
# hyperparameter tuning for Decision tree classifier
from sklearn.model_selection import GridSearchCV
churn_classifier = DecisionTreeClassifier(random_state=1)
params = [{'criterion':['entropy', 'gini'],
          'max_depth': [6, 8, 10, 12, 20], 'class_weight':['balanced', {0:0.45, 1:0.55},
                                                          {0:0.55, 1:0.45}, {0:0.40, 1:0.60}],
           'random_state' :[1]}]
grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                          scoring = 'f1', cv = 10, n_jobs=-1)
```

```
# tuning Decision Tree for dataset with outlier i.e. churn_data_df
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

```
{'class_weight': {0: 0.55, 1: 0.45},
 'criterion': 'entropy',
 'max_depth': 8,
 'random_state': 1}
```

** Python code to generate above result

Now, building DecisionTreeclassifier with parameter suggested by GridSearchCV for dataset churn_data_df.

```
K-fold cross validation score of model for k = 10 is :
0.9441953929977883
====================================
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.95      0.99      0.97      1443
          1       0.91      0.66      0.76       224

avg / total       0.94      0.95      0.94      1667

====== Confusion matrix =======
[[1429   14]
 [  77  147]]
```

** Python code to generate above result

Let us now tune decision tree for following parameters on dataset without outliers i.e. churn_data_df_wo

```
# hyperparameter tuning for Decision tree classifier for dataset without outliers i.e. churn_dat
from sklearn.model_selection import GridSearchCV
churn_classifier = DecisionTreeClassifier(random_state=1)
params = [{'criterion':['entropy', 'gini'],
          'max_depth': [6, 8, 10, 12], 'class_weight':['balanced', {0:0.45, 1:0.55},
                                                        {0:0.55, 1:0.45}, {0:0.40, 1:0.60}],
           'random_state' :[1]}]
grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                          scoring = 'f1', cv = 10, n_jobs=-1)
grid_search = grid_search.fit(X_train_wo, y_train_wo)
grid_search.best_params_
```

```
{'class_weight': {0: 0.45, 1: 0.55},
 'criterion': 'gini',
 'max_depth': 8,
 'random_state': 1}
```

** Python code to generate above result

Now, building DecisionTreeclassifier with parameter suggested by GridSearchCV for dataset churn_data_df_wo i.e. without outliers.

```
K-fold cross validation score of model for k = 10 is :
0.9402918196326562
====================================
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.95      0.98      0.96      1443
          1       0.83      0.65      0.73       224

avg / total       0.93      0.94      0.93      1667

====== Confusion matrix =======
[[1414   29]
 [  78  146]]
```

** Python code to generate above result

Analysis:

We have improvement in our Decision Tree model after parameter tuning. For dataset with outliers we improved model from 92.26% to 94.42% K-fold accuracy and for dataset without outliers we improved our model from 91.95% to 94.03% K-fold accuracy.

Random Forest Model Hyperparameter tuning:

Let us tune Random Forest for following parameters on dataset with outliers. Churn_data_df

```
# Grid search for finding best parameter for random_forest on churn_data_df dataset
churn_classifier = RandomForestClassifier(random_state=1)
params = [{'criterion':['entropy', 'gini'],'n_estimators':[800, 1000],
          'max_depth': [8, 10, 12], 'class_weight':['balanced', {0:0.45, 1:0.55},
                                                      {0:0.55, 1:0.45}],
           'random_state' :[1]}]
grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                           scoring = 'f1', cv = 10, n_jobs=-1)
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

```
{'class_weight': 'balanced',
 'criterion': 'entropy',
 'max_depth': 10,
 'n_estimators': 1000,
 'random_state': 1}
```

** Python code to generate above result

Let us now build again Random Forest model with parameter suggested by GridsearchCV, on dataset with outlier i.e. churn_data_df

```
K-fold cross validation score of model for k = 10 is :
0.9526020032008056
===================================
Model performance on test dataset
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.96      0.99      0.98      1443
          1       0.93      0.74      0.82       224

avg / total       0.96      0.96      0.96      1667

====== Confusion matrix =======
[[1430   13]
 [  58  166]]
```

** Python code to generate above result

Let us tune Random Forest for following parameters on dataset without outliers.

```
# tuning on chrun_data_df_wo dataset for random forest model
churn_classifier = RandomForestClassifier(random_state=1)
params = [{'criterion':['entropy', 'gini'],'n_estimators':[600, 800, 1000],
          'max_depth': [8, 10, 12, 14], 'class_weight':['balanced', {0:0.45, 1:0.55}],
           'random_state' :[1]}]
grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                          scoring = 'f1', cv = 10, n_jobs=-1)
grid_search = grid_search.fit(X_train_wo, y_train_wo)
grid_search.best_params_
```

```
{'class_weight': 'balanced',
 'criterion': 'entropy',
 'max_depth': 10,
 'n_estimators': 1000,
 'random_state': 1}
```

** Python code to generate above result

Let us now build again Random Forest model with parameter suggested by GridsearchCV, on dataset without outlier i.e. churn_data_df_wo

```
K-fold cross validation score of model for k = 10 is :
0.9524252440406705
====================================
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.96      0.99      0.98      1443
          1       0.94      0.74      0.83       224

avg / total       0.96      0.96      0.96      1667

====== Confusion matrix =======
[[1432   11]
 [  58  166]]
```

** Python code to generate above result

Analysis:

After Hyperparameter tuning our model performance increased from 93% to 95%. And now our model is predicting 58 observations as churning false while these were having churning as true. Which is lowest among all models. So hyperparameter tuning helped us in getting good result.

2.2.4 SMOTE + Tomek (Oversampling)

As we see, we improved our model with tuning of hyperparameter. Overall accuracy increased but we can see from final confusion matrix we have more false positive than false negative. (assuming positive class to churn false i.e. 0)

False positive:- Incorrect classified as class 0 (churning false) , in actual they belongs to class 1 (Churning True).

So, from business point of view, we would be more interested in person who may churn. So that, we can give extra attention to those customers for retaining them with our business.

As, for class 0 i.e churning false, we are getting more accuracy than class 1 i.e. churning true. And we have only 14% data for churning true class. So, it seems our model is overfitting to class 0 i.e. churning false.

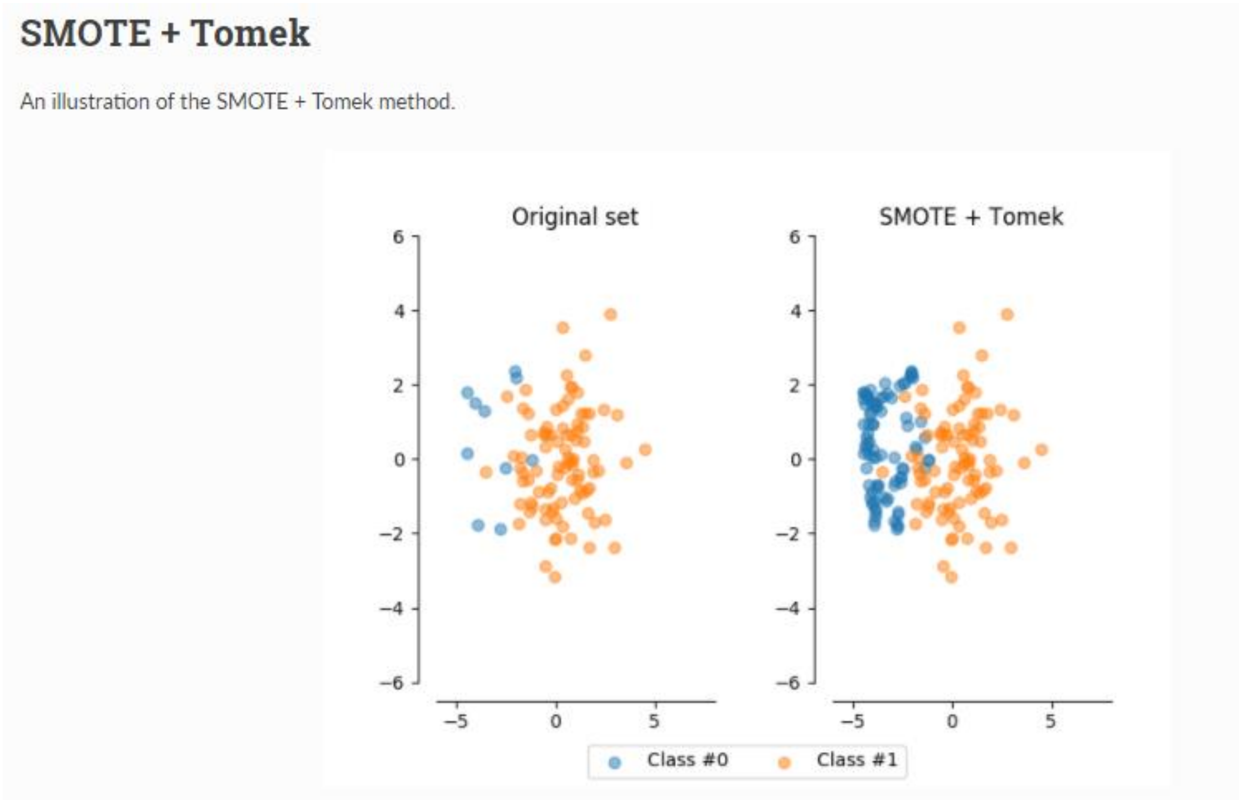
To resolve this issue let us make target in balanced way. For this we would use oversampling technique of SMOTE (Synthetic minority over-sampling technique) and to remove noisy over data. Both SMOTE + tomek will make oversampled data without noise.

SMOTE generate artificial data for minority class, for each observation for minority class k nearest neighbors are identified and then randomly few neighbors are selected and then artificial observation are generated and spread along the line joining observation and nearest neighbors.

Tomek (T–link) use distance method between points and based on distance between good examples identifies each observation as data, noise and at boundary points. With the help of Tomek we can remove noise.

We will use SMOTE + Tomek algorithm for balancing target class.

Below picture is from sklearn official documentation for SMOTE + Tomek



- We will implement smote + tomek in both dataset i.e. churn_data_df and churn_data_df_wo
- We will use Random Forest only and will tune its hyperparameter.

** Python code for resampling using smotetomek

Shape of dataset after SMOTE + Tomek (oversampling) for churn_data_df.

```
print(X_resampled.shape)
print(y_resampled.shape)
print("class proportion")
print(pd.Series(y_resampled).value_counts(normalize = True))
```

```
(5662, 13)
(5662,)
class proportion
1    0.5
0    0.5
dtype: float64
```

Shape of dataset after SMOTE + Tomek (oversampling) for churn_data_df_wo

```
print(X_resampled_wo.shape)
print(y_resampled_wo.shape)
print("class proportion")
print(pd.Series(y_resampled_wo).value_counts(normalize = True))
```

```
(5202, 13)
(5202,)
class proportion
1    0.5
0    0.5
dtype: float64
```

Random Forest Hyperparameter tuning on oversampled dataset

Now, Tuning Random Forest Model for first resampled Data i.e. with outliers

```
# Tuning Random Forest model for resampled data from churn_data_df
churn_classifier = RandomForestClassifier(random_state=1)
params = [{'criterion':['entropy', 'gini'],'n_estimators':[600, 800, 1000],
           'max_depth': [20, 22, 24, 26], 'class_weight':['balanced', {0:0.55, 1:0.45},
                                                          {0:0.45, 1:0.55}],
            'random_state' :[1]}]
grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                           scoring = 'f1', cv = 10, n_jobs=-1)
grid_search = grid_search.fit(X_resampled, y_resampled)
grid_search.best_params_
```

```
{'class_weight': 'balanced',
 'criterion': 'entropy',
 'max_depth': 24,
 'n_estimators': 1000,
 'random_state': 1}
```

** Python code to generate above result

Building Random Forest with parameter suggested by GridsearchCV for oversampled dataset (churn_data_df)

```
K-fold cross validation score of model for k = 10 is :
0.9579853680386204
====================================
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.97      0.98      0.98      1443
          1       0.86      0.81      0.83       224

avg / total       0.96      0.96      0.96      1667

====== Confusion matrix =======
[[1413   30]
 [  42  182]]
```

** Python code to generate above result

Now, Tuning Random Forest Model for second resampled Data i.e. without outliers

```
churn_classifier = RandomForestClassifier(random_state=1)
params = [{'criterion':['entropy','gini'],'n_estimators':[600, 800, 1000],
          'max_depth': [24, 26, 28], 'class_weight':['balanced', {0:0.45, 1:0.55},
                                                      {0:0.55, 1:0.45}],
           'random_state' :[1]}]
grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                           scoring = 'f1', cv = 10, n_jobs=-1)
grid_search = grid_search.fit(X_resampled_wo, y_resampled_wo)
grid_search.best_params_
```

```
{'class_weight': {0: 0.55, 1: 0.45},
 'criterion': 'entropy',
 'max_depth': 26,
 'n_estimators': 800,
 'random_state': 1}
```

** Python code to generate above result

Modeling Random Forest with parameter suggested by GridsearchCV for resampled dataset without outliers.

```
K-fold cross validation score of model for k = 10 is :
0.9623445328617743
====================================
====== Classification Report =======
             precision    recall  f1-score   support

          0       0.97      0.98      0.97      1443
          1       0.85      0.78      0.81       224

avg / total       0.95      0.95      0.95      1667

====== Confusion matrix =======
[[1412   31]
 [  49  175]]
```

** Python code to generate above result

Analysis:

After oversampling data, we improved our model and moreover now our model is predicting true churning more than previous model and dataset.

Before SMOTE, Number of cases when Random Forest detected churning as false (however actual is true): 58

After SMOTE, Number of cases when Random Forest detected churning as false (however actual is true) : 42

Decrease in false detection for true churning = ((58-42)/58) *100 = 27.58%

Final accuracy on test dataset:

```
               churn_prediction
                   0    1
Actual class 0    1413   30
Actual class 1    42    182

Accuracy = Correct Prediction / total observation
         = 95.68%

False Positive Rate (Assuming class 0 i.e. churn = False as
positive class)
FPR = FP / (FP + TN)
    = 42 / (42 + 182)
    = 18.75%

Model K-fold accuracy for K = 10:95.80%
```

Chapter 3

Conclusion

3.1 Final Model and Training Dataset

From the above models we selected below dataset and model for predicting our test dataset. As below model giving us less error in predicting true churning which was our main motive to reduce churning.

Dataset:

- First take whole training dataset.
- Drop columns ‘area code’, ‘state’, ‘phone number’, ‘total day minutes’, ‘total eve charge’, ‘total night charge’, ‘total intl charge’.
- Change ‘international plan’, ‘voice mail plan’ and ‘Churn’ columns to category and then to levels of category (0 and 1)
- Do same thing with test dataset
- Apply SMOTE + Tomek to balancing the target variable on training dataset.

Model:

- Use random Forest model and train using dataset which we prepared with above steps.
- Do hyperparameter tuning.
- And then build model using tuned hyperparameter.
- Our model is ready to predict !!!!!!!!!

3.2 End Notes

- Result shown in this report are from Python notebook.
- We feed our model with two different dataset. If we would have large dataset then we would use only single dataset after removing outliers.
- R-code file link is in Appendix B. Result from R code would not be exact same for building models as implementation of function at backend is different for R and Python. But information would be almost same for both Python and R.

Complete Python code

```
# importing Basic required library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Reading train and test file
churn_data_df = pd.read_csv("Train_data.csv")
test_data_df = pd.read_csv("Test_data.csv")
############################################
#                                          #
#     2.1 Exploratory Data Analysis        #
#                                          #
############################################
###################################
#  2.1.1 understanding the data   #
###################################

# checking dimension of data
print(churn_data_df.shape)
print(test_data_df.shape)
# looking at few observation
churn_data_df.head()
# all columns of dataset
churn_data_df.columns

# Checking datatypes and information of dataset  -> See Result
churn_data_df.info()

# Checking numerical statistics of continuous variable  -> See Result
churn_data_df.describe()

# Extracting each category with object datatype and adding area code, as area code is
# category values in numerical form
cat_columns = list(churn_data_df.columns[churn_data_df.dtypes == 'object'])
cat_columns.insert(2, 'area code')
cat_columns
# changing to categorical variable to category datatype
churn_data_df[cat_columns] = churn_data_df[cat_columns].apply(pd.Categorical)
test_data_df[cat_columns] = test_data_df[cat_columns].apply(pd.Categorical)
# checking total unique values in each categorical variable  -> See Result
churn_data_df[cat_columns].nunique()

# counting of each unique values in last three category  -> See Output
churn_data_df[cat_columns[3:6]].apply(pd.Series.value_counts)

# alternate solution to getting counting in one go
print("value counts of categorical column")
print()
for i in cat_columns[2:6]:
    print(i)
    print(churn_data_df[i].value_counts())
    print("=================================")
```

```
# getting percentage of target variable Churn in training dataset  -> See Output
churn_data_df['Churn'].value_counts(normalize = True)

###################################
#  2.1.2 Missing value analysis   #
###################################
# checking for missing value in each columns  -> See Output
churn_data_df.isnull().sum()

###################################
#  2.1.3 outlier analysis         #
###################################

# defining function to plot historgram and box plot of numerical variable
def hist_and_box_plot(col1, col2, col3, data, bin1=30, bin2=30, bin3=30, sup =""):
    fig, ax = plt.subplots(nrows = 2, ncols = 3, figsize= (12,6))
    super_title = fig.suptitle("Boxplot and Histogram: "+sup, fontsize='x-large')
    plt.tight_layout()
    sns.boxplot(y = col1, x = 'Churn', data = data, ax = ax[0][0], hue = 'Churn')
    sns.boxplot(y = col2, x = 'Churn', data = data, ax = ax[0][1], hue = 'Churn')
    sns.boxplot(y = col3, x = 'Churn', data = data, ax = ax[0][2], hue = 'Churn')
    sns.distplot(data[col1], ax = ax[1][0], bins = bin1)
    sns.distplot(data[col2], ax = ax[1][1], bins = bin2)
    sns.distplot(data[col3], ax = ax[1][2], bins = bin3)
    fig.subplots_adjust(top = 0.90)
    plt.show()

# plotting histogram and boxplot for day calls, minute and charges  -> See Output
hist_and_box_plot('total day minutes', 'total day calls', 'total day charge',
                  data = churn_data_df, sup = "Day time call details")

# plotting histogram and boxplot for evening calls, minute and charges  -> See Output
hist_and_box_plot('total eve minutes', 'total eve calls', 'total eve charge',
                  data = churn_data_df, sup = "Evening call details")

# plotting histogram and boxplot for night calls, minute and charges
hist_and_box_plot('total night minutes', 'total night calls', 'total night charge',
                  data = churn_data_df, sup = "Night call details")

# plotting histogram and boxplot for international calls, minute and charges
hist_and_box_plot('total intl minutes', 'total intl calls', 'total intl charge',
                  data = churn_data_df, bin2=10,sup="International call Details")

# plot for account length , vmail messages and customer service calls  -> See Output
hist_and_box_plot('account length','number vmail messages','number customer service calls'
,
                  data = churn_data_df, bin2 = 10, bin3 = 5,
                  sup = "account, vmail messages, customer service calls")

#####################
# outlier removing  #
#####################
# making another dataset which will not contain outlier stated by boxplot
# as we dont want to loose information already we have small dataset,
# so will create two dataset
# further reason explained in Project report
```

```
# churn_data_df_wo will be our second dataset without outliers
churn_data_df_wo = churn_data_df
# getting all numeric columns
numeric_columns = list(churn_data_df.columns[churn_data_df.dtypes != 'category'])
# removing numeric columns for which we will not do outlier removal process
numeric_columns.remove('number vmail messages')
numeric_columns.remove('number customer service calls')

# removing outliers with boxplot method i.e. points which lie below 1.5*IQR distance
# and above 1.5*IQR distance from median  -> See Output
for i in numeric_columns:
     q75, q25 = np.percentile(churn_data_df_wo.loc[:,i], [75 ,25])
     iqr = q75 - q25
     min = q25 - (iqr*1.5)
     max = q75 + (iqr*1.5)
     churn_data_df_wo = churn_data_df_wo.drop(
             churn_data_df_wo[churn_data_df_wo.loc[:,i] < min].index)
     churn_data_df_wo = churn_data_df_wo.drop(
             churn_data_df_wo[churn_data_df_wo.loc[:,i] > max].index)

# plotting histogram and boxplot for day calls, minute and charges for churn_data_df_wo
# -> See Output
hist_and_box_plot('total day minutes', 'total day calls', 'total day charge',
                  data = churn_data_df_wo, sup = "Day time call details")

#plotting histogram and boxplot for evening calls, minute and charges for churn_data_df_wo

hist_and_box_plot('total eve minutes', 'total eve calls', 'total eve charge',
                  data = churn_data_df_wo, sup = "Evening call details")

# plotting histogram and boxplot for night calls, minute and charges for churn_data_df_wo

hist_and_box_plot('total night minutes', 'total night calls', 'total night charge',
                  data = churn_data_df_wo, sup = "Night call details")

#plotting histogram and boxplot for international detail for churn_data_df_wo
hist_and_box_plot('total intl minutes', 'total intl calls', 'total intl charge',
                  data = churn_data_df_wo, bin2=10, sup="International call Details")

###################################
#  2.1.4 Feature Selection        #
###################################

# Correlation plot between numerical values  -> See Output
numeric_columns = list(churn_data_df.columns[churn_data_df.dtypes != 'category'])
sns.pairplot(data = churn_data_df, x_vars= numeric_columns, y_vars= numeric_columns,
             hue = 'Churn')

# heat map plot between numerical values  -> See Output
fig = plt.figure(figsize = (14,10))
corr = churn_data_df[numeric_columns].corr()
sns.heatmap(corr, mask = np.zeros_like(corr, dtype = np.bool), square = True,
            annot= True, cmap = sns.diverging_palette(220, 10, as_cmap= True))
plt.title("HeatMap between numerical columns of churn dataset")
```

```python
# checking dependency between churn and independent variable (category)  -> See Output
cat_var = ['state', 'area code', 'international plan', 'voice mail plan']
from scipy.stats import chi2_contingency
print("Chi-square - test of independence")
print("=================================")
for i in cat_var:
    chi2, p, dof, ex = chi2_contingency(pd.crosstab(churn_data_df['Churn'],
                                                    churn_data_df[i]))
    print("p-value between Churn and {}".format(i))
    print(p)
    print('----------------------------')

# checking independency between independent variables  -> See Output
chi2, p, dof, ex = chi2_contingency(pd.crosstab(churn_data_df['international plan'],
                                                churn_data_df['voice mail plan']))
print("p-value between international plan  and voice mail plan")
print(p)
print('----------------------------')

# Dropping state, area code and phone number as they are not giving infomation
churn_data_df = churn_data_df.drop(columns=['state', 'area code', 'phone number'])
churn_data_df_wo = churn_data_df_wo.drop(columns=['state','area code','phone number'])
test_data_df = test_data_df.drop(columns=['state', 'area code', 'phone number'])
# changing categories to levels (0 and 1)
cat_columns = churn_data_df.columns[churn_data_df.dtypes == 'category']
for i in cat_columns:
    churn_data_df[i] = churn_data_df[i].cat.codes
    churn_data_df_wo[i] = churn_data_df_wo[i].cat.codes
    test_data_df[i] = test_data_df[i].cat.codes

# checking importance of feature  -> See Output
from sklearn.ensemble import ExtraTreesClassifier
cls = ExtraTreesClassifier(n_estimators=200)
X = churn_data_df.drop(columns=['Churn'])
y = churn_data_df['Churn']
cls.fit(X, y)
imp_feat = pd.DataFrame({'Feature': churn_data_df.drop(columns=["Churn"]).columns,
                         'importance':cls.feature_importances_})
imp_feat.sort_values(by = 'importance', ascending=False).reset_index(drop = True)

# Checking VIF values of numeric columns  -> See Output
from statsmodels.stats.outliers_influence import variance_inflation_factor as vf
from statsmodels.tools.tools import add_constant
numeric_df = add_constant(churn_data_df[numeric_columns])
vif = pd.Series([vf(numeric_df.values, i) for i in range(numeric_df.shape[1])],
                 index = numeric_df.columns)
vif.round(1)

# Deleting multicollinear columns
churn_data_df=churn_data_df.drop(columns=['total day minutes','total eve charge',
                                          'total night charge', 'total intl charge'])
churn_data_df_wo=churn_data_df_wo.drop(columns=['total day minutes','total eve charge',
                                                'total night charge',
                                                'total intl charge'])
test_data_df = test_data_df.drop(columns=['total day minutes','total eve charge',
                                          'total night charge', 'total intl charge'])
```

```python
# checking again VIF after removal of multicollinear columns  -> See Output
numeric_columns = list(churn_data_df.columns[3:13])
numeric_columns.insert(0, 'account length')
numeric_df = add_constant(churn_data_df[numeric_columns])
vif = pd.Series([vf(numeric_df.values, i) for i in range(numeric_df.shape[1])],
                 index = numeric_df.columns)
vif.round(1)

# splitting in X and y for train and test
# X_train -> whole datset
# X_train_wo -> dataset after removal of outliers
X_train = churn_data_df.drop('Churn', axis = 1)
y_train = churn_data_df['Churn']
X_train_wo = churn_data_df_wo.drop('Churn', axis =1)
y_train_wo = churn_data_df_wo['Churn']
X_test = test_data_df.drop('Churn', axis = 1)
y_test = test_data_df['Churn']

############################################
#                                          #
#                                          #
#   2.2.2 Building Classification models   #
#                                          #
#                                          #
############################################

# making general function to fit and predict result (Confusion Matrix)
# and performance (K-fold CV) and to not to repeat code everytime
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
def fit_predict_show_performance(classifier, X_train, y_train):
    '''''
    this function will fit on data passed in argument then it will predict on
    X_test datasetand then will calculate the 10 fold CV accuracy score and then will
    generate classification report and confusion matrix based on prediction and y_test
    it will only print result, to get all calculated result, uncomment last line and
    call it like below example:
    y_pred, cr, cm = fit_predict_show_performance(churn_classifier, X_train, y_train)
    '''
    # fitting model
    classifier.fit(X_train, y_train)
    churn_prediction = classifier.predict(X_test)
    # getting K-fold CV scores for K = 10
    ten_performances = cross_val_score(estimator=classifier,X=X_train,y=y_train,cv=10)
    k_fold_performance = ten_performances.mean()
    print("K-fold cross validation score of model for k = 10 is :")
    print(k_fold_performance)
    print("====================================")
    print("====== Classification Report ======= ")
    cr = classification_report(y_test,churn_prediction)
    print(cr)
    print("====== Confusion matrix ======= ")
    cm = confusion_matrix(y_test,churn_prediction)
    print(cm)
    #return [churn_prediction, cr, cm]
```

```
#########################
#  Logistic Regression  #  -> See Output
#########################

# Building Logistic Regression for churn_data_df i.e. with outliers
from sklearn.linear_model import LogisticRegression
churn_classifier = LogisticRegression()
fit_predict_show_performance(churn_classifier, X_train, y_train)

# Building Logistic Regression for churn_data_df_wo i.e. without outliers  -> See Output
churn_classifier = LogisticRegression()
fit_predict_show_performance(churn_classifier, X_train_wo, y_train_wo)


#########################
#       KNN             #
#########################
# knn for churn_data_df i.e. dataset with outliers  -> See Output
from sklearn.neighbors import KNeighborsClassifier
churn_classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski',p =2)
fit_predict_show_performance(churn_classifier, X_train, y_train)

# knn for churn_data_df_wo i.e. dataset without outliers  -> See Output
churn_classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski',p =2)
fit_predict_show_performance(churn_classifier, X_train_wo, y_train_wo)

#########################
#     Naive Bayes       #
#########################
# Naive bayes with outlier i.e. churn_data_df  -> See Output
from sklearn.naive_bayes import GaussianNB
churn_classifier = GaussianNB()
fit_predict_show_performance(churn_classifier, X_train, y_train)

# Naive bayes without outlier i.e. churn_data_df_wo  -> See Output
churn_classifier = GaussianNB()
fit_predict_show_performance(churn_classifier, X_train_wo, y_train_wo)

#########################
#    Decision Tree      #
#########################
# Decision tree classifier for churn_data_df with outliers  -> See Output
from sklearn.tree import DecisionTreeClassifier
churn_classifier = DecisionTreeClassifier(criterion = 'entropy', random_state=1)
fit_predict_show_performance(churn_classifier, X_train, y_train)

# Decision tree classifier for churn_data_df_wo without outliers  -> See Output
churn_classifier = DecisionTreeClassifier(criterion = 'entropy', random_state=1)
fit_predict_show_performance(churn_classifier, X_train_wo, y_train_wo)
```

```
#########################
#    Random Forest      #
#########################
# Random forest model on churn_data_df i.e. with outliers  -> See Output
from sklearn.ensemble import RandomForestClassifier
churn_classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy',
                                          random_state=1)
fit_predict_show_performance(churn_classifier, X_train, y_train)

# Random forest model on churn_data_df_wo i.e. without outliers  -> See Output
churn_classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy',
                                         random_state=1)
fit_predict_show_performance(churn_classifier, X_train_wo, y_train_wo)

############################################
#                                          #
#                                          #
#        Hyperparameter tuning             #
#                                          #
#                                          #
############################################
#########################################
#                                       #
# tuning decision tree for both dataset #
# churn_data_df and churn_data_df_wo    #
#                                       #
#########################################
# hyperparameter tuning for Decision tree classifier  -> see output
from sklearn.model_selection import GridSearchCV
churn_classifier = DecisionTreeClassifier(random_state=1)
params = [{'criterion':['entropy', 'gini'],
          'max_depth':[6,8,10,12,20],'class_weight':['balanced',{0:0.45, 1:0.55},
                      {0:0.55,1:0.45},{0:0.40,1:0.60}],'random_state' :[1]}]
grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                          scoring = 'f1', cv = 10, n_jobs=-1)

# tuning Decision Tree for dataset with outlier i.e. churn_data_df
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_

#Decision tree classifier for churn_data_df i.e. with outliers after tuning parameter
#from sklearn.tree import DecisionTreeClassifier  -> see output
churn_classifier = DecisionTreeClassifier(criterion = 'entropy',
                                          class_weight={0:0.55, 1:0.45},max_depth=8,
                                          random_state=1)
fit_predict_show_performance(churn_classifier, X_train, y_train)

# hyperparameter tuning for Decision tree for dataset without outliers  -> see output
from sklearn.model_selection import GridSearchCV
churn_classifier = DecisionTreeClassifier(random_state=1)
params = [{'criterion':['entropy', 'gini'],
          'max_depth': [6, 8, 10, 12], 'class_weight':['balanced', {0:0.45, 1:0.55},
                       {0:0.55, 1:0.45}, {0:0.40, 1:0.60}], 'random_state' :[1]}]
grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                          scoring = 'f1', cv = 10, n_jobs=-1)
grid_search = grid_search.fit(X_train_wo, y_train_wo)
grid_search.best_params_
```

```
# Decision tree classifier for churn_data_df_wo without outliers  -> see output
churn_classifier=DecisionTreeClassifier(criterion = 'gini', max_depth = 8,
                                        class_weight={0:0.45,1:0.55},random_state=1)
fit_predict_show_performance(churn_classifier, X_train_wo, y_train_wo)

############ Hyperparameter tuning for Random Forest #############  -> see output
# Grid search for finding best parameter for random_forest on churn_data_df dataset
churn_classifier = RandomForestClassifier(random_state=1)
params=[{'criterion':['entropy', 'gini'],'n_estimators':[800,1000],
         'max_depth': [8, 10, 12], 'class_weight':['balanced', {0:0.45, 1:0.55},
                      {0:0.55, 1:0.45}],'random_state' :[1]}]
grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                          scoring = 'f1', cv = 10, n_jobs=-1)
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_

# tuned randomforest model on chrun_data_df  -> see output
churn_classifier = RandomForestClassifier(n_estimators = 1000, criterion = 'entropy',
                                          class_weight='balanced',max_depth=10,
                                          random_state=1)
fit_predict_show_performance(churn_classifier, X_train, y_train)

# tuning on chrun_data_df_wo dataset for random forest model  -> see output
churn_classifier = RandomForestClassifier(random_state=1)
params = [{'criterion':['entropy', 'gini'],'n_estimators':[600, 800, 1000],
          'max_depth': [8, 10, 12, 14], 'class_weight':['balanced', {0:0.45, 1:0.55}],
           'random_state' :[1]}]
grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                          scoring = 'f1', cv = 10, n_jobs=-1)
grid_search = grid_search.fit(X_train_wo, y_train_wo)
grid_search.best_params_

# tuned Random forest model on churn_data_df_wo i.e. without outliers  -> see output
churn_classifier = RandomForestClassifier(n_estimators = 1000, criterion = 'entropy',
                                          max_depth=10,class_weight='balanced',
                                          random_state=1)
fit_predict_show_performance(churn_classifier, X_train_wo, y_train_wo)

############################################
#                                          #
#    SMOTE + Tomek (Oversampling)          #
#         Balancing Target                 #
#                                          #
############################################

# resmapling data from churn_data_df i.e. with outliers
from imblearn.combine import SMOTETomek
smt = SMOTETomek()
X_resampled, y_resampled = smt.fit_sample(X_train, y_train)

# checking shape of data after resampling
print(X_resampled.shape)
print(y_resampled.shape)
print("class proportion")
print(pd.Series(y_resampled).value_counts(normalize = True))
```

```
# Tuning Random Forest model for resampled data from churn_data_df  -> see output
churn_classifier = RandomForestClassifier(random_state=1)
params = [{'criterion':['entropy', 'gini'],'n_estimators':[600, 800, 1000],
          'max_depth': [20, 22, 24, 26], 'random_state' :[1],
          'class_weight':['balanced', {0:0.55, 1:0.45},{0:0.45, 1:0.55}]}]

grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                          scoring = 'f1', cv = 10, n_jobs=-1)
grid_search = grid_search.fit(X_resampled, y_resampled)
grid_search.best_params_

# building  Random Forest model on tuned hyperparameter  -> see output
churn_classifier = RandomForestClassifier(n_estimators = 1000, criterion = 'entropy',
                                          class_weight='balanced',max_depth=24,
                                          random_state=1)
fit_predict_show_performance(churn_classifier, X_resampled, y_resampled)

# resampling data for dataset churn_data_df_wo i.e. without outliers
smt = SMOTETomek()
X_resampled_wo, y_resampled_wo = smt.fit_sample(X_train_wo, y_train_wo)

# checking shape of data
print(X_resampled_wo.shape)
print(y_resampled_wo.shape)
print("class proportion")
print(pd.Series(y_resampled_wo).value_counts(normalize = True))

# tuning Random forest model for resampled data without outliers  -> see output
churn_classifier = RandomForestClassifier(random_state=1)
params = [{'criterion':['entropy','gini'],'n_estimators':[600, 800, 1000],
          'max_depth': [24, 26, 28], 'random_state' :[1],
          'class_weight':['balanced', {0:0.45, 1:0.55},{0:0.55, 1:0.45}]}]
grid_search = GridSearchCV(estimator=churn_classifier, param_grid=params,
                          scoring = 'f1', cv = 10, n_jobs=-1)
grid_search = grid_search.fit(X_resampled_wo, y_resampled_wo)
grid_search.best_params_

# building Random Forest model on tuned parameter  -> see output
churn_classifier = RandomForestClassifier(n_estimators = 800, criterion = 'entropy',
                                          class_weight={0:0.55, 1:0.45},
                                          max_depth=26,random_state=1)
fit_predict_show_performance(churn_classifier, X_resampled_wo, y_resampled_wo)
```

Full R code Link

R Code

References:

https://www.analyticsvidhya.com/blog/2016/03/practical-guide-deal-imbalanced-classification-problems/

http://contrib.scikit-learn.org/imbalanced-learn/stable/auto_examples/combine/plot_smote_tomek.html