Customer Churn analysis in the telecom industry

Created BY,

Rudraneel Chakraborty

Table of Content

PROJECT OBJECTIVE

The proposed Application is used to develop model for customer churn analysis in telecom industry. With the sample data we will make the machine learn.

With the rapid development of communication technology, the field of telecommunication faces complex challenges due to the number of vibrant competitive service providers. Customer Churn is the major issue that faces by the Telecommunication industries in the world. Churn is the activity of customers leaving the company and discarding the services offered by it, due to the dissatisfaction with the provided services. The main areas of this research contend with the ability to identify potential churn customers, cluster customers with similar consumption behaviour and mine the relevant patterns embedded in the collected data.

To minimize the customer churn, prediction activity to be an important part of the telecommunication industry's vital decision making and strategic planning process.

PROJECT SCOPE

The model designed will be helping the companies to escape of customer churn problem with respect to the input features in the model. It is for that; telecommunication companies realize that to keep existing customers is getting more important and agrees that churn analysis is one of the important data mining application areas. Churn Analysis is applied to research why customers switch service provider.

Further it also helps to determining the reasons, figuring out what types of customers are lost, this research will benefit for the company in the future as well.

HARDWARE & SOFTWARE RERUIREMENTS

We need these basic hardware & software requirements to run this application in a desired machine:

- 4 GB RAM
- 32-bit or 64-bit architecture machine
- Licensed free Anaconda software
- Minimum 3 GB disk space to download & install
- Need either Windows or Linux or MAC operating system
- Need the python version 3.5 or higher

DATA DESCRIPTION

We are working on the Dataset downloaded from the KAGGLE.COM website on customer churn analyse document.

Under python to do DATA VISULIZATION, ANALYSIS & CLEANING we use the following modules:

- **NUMPY** or numeric python is basically built to work on numerical data.
- **PANDAS** is basically built on NUMPY to do certain jobs on the basis of numeric data.
- **MATPLOTLIB.PYPLOT** to implement the graphs such as Bar plot, Histogram, Box plot, Count plot, Bell curve to analyse our data.
- **SEABORN** is used to do critical statistical analysis in very simple & effective way, which gives better results comparing to MATPLOTLIB.

Variables are two types:

- Univariate Analysis
- Bivariate Analysis

Data Cleaning is one of the important jobs. It is said that most of the time is being given behind cleaning the data. It includes handling of NaN values, outliers, threshold values, normalization of Data. If these are not perfectly handled then the experience given to the machine will be of no use.

CONTENT OF DATASET

The dataset contains several parameters which are considered important during analysing the model. The parameters are shown in the data which is given below:

	CustomerID	Churn	MonthlyRevenue	MonthlyMinutes	TotalRecurringCharge	DirectorAssistedCalls	OverageMinutes	RoamingCalls	PercChangeMinutes	Perc
0	3000002	Yes	24.00	219.0	22.0	0.25	0.0	0.0	-157.0	
1	3000010	Yes	16.99	10.0	17.0	0.00	0.0	0.0	-4.0	
2	3000014	No	38.00	8.0	38.0	0.00	0.0	0.0	-2.0	
3	3000022	No	82.28	1312.0	75.0	1.24	0.0	0.0	157.0	
4	3000026	Yes	17.14	0.0	17.0	0.00	0.0	0.0	0.0	
5	3000030	No	38.05	682.0	52.0	0.25	0.0	0.0	148.0	
6	3000038	No	31.66	26.0	30.0	0.25	0.0	0.0	60.0	
7	3000042	No	62.13	98.0	66.0	2.48	0.0	0.0	24.0	
8	3000046	No	35.30	24.0	35.0	0.00	0.0	0.0	20.0	
9	3000050	No	81.00	1056.0	75.0	0.00	0.0	0.0	43.0	

10 rows × 58 columns

df.head(10): It shows the us first 10 instances of the sample data of the model

Here we are seeing an instance of the sample data being provided to us.

df.shape : It gives us the total number of rows followed by column.

```
(51047, 58)
```

df.info(): It gives us the type of data specifically each column having.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51047 entries, 0 to 51046
Data columns (total 58 columns):
CustomerID                  51047 non-null int64
Churn                       51047 non-null object
MonthlyRevenue              50891 non-null float64
MonthlyMinutes              50891 non-null float64
TotalRecurringCharge        50891 non-null float64
```

```
DirectorAssistedCalls        50891 non-null float64
OverageMinutes               50891 non-null float64
RoamingCalls                 50891 non-null float64
PercChangeMinutes            50680 non-null float64
PercChangeRevenues           50680 non-null float64
DroppedCalls                 51047 non-null float64
BlockedCalls                 51047 non-null float64
UnansweredCalls              51047 non-null float64
CustomerCareCalls            51047 non-null float64
ThreewayCalls                51047 non-null float64
ReceivedCalls                51047 non-null float64
OutboundCalls                51047 non-null float64
InboundCalls                 51047 non-null float64
PeakCallsInOut               51047 non-null float64
OffPeakCallsInOut            51047 non-null float64
DroppedBlockedCalls          51047 non-null float64
CallForwardingCalls          51047 non-null float64
CallWaitingCalls             51047 non-null float64
MonthsInService              51047 non-null int64
UniqueSubs                   51047 non-null int64
ActiveSubs                   51047 non-null int64
ServiceArea                  51023 non-null object
Handsets                     51046 non-null float64
HandsetModels                51046 non-null float64
CurrentEquipmentDays         51046 non-null float64
AgeHH1                       50138 non-null float64
AgeHH2                       50138 non-null float64
ChildrenInHH                 51047 non-null object
HandsetRefurbished           51047 non-null object
HandsetWebCapable            51047 non-null object
TruckOwner                   51047 non-null object
RVOwner                      51047 non-null object
Homeownership                51047 non-null object
BuysViaMailOrder             51047 non-null object
RespondsToMailOffers         51047 non-null object
OptOutMailings               51047 non-null object
NonUSTravel                  51047 non-null object
OwnsComputer                 51047 non-null object
HasCreditCard                51047 non-null object
RetentionCalls               51047 non-null int64
RetentionOffersAccepted      51047 non-null int64
NewCellphoneUser             51047 non-null object
NotNewCellphoneUser          51047 non-null object
ReferralsMadeBySubscriber    51047 non-null int64
IncomeGroup                  51047 non-null int64
OwnsMotorcycle               51047 non-null object
AdjustmentsToCreditRating    51047 non-null int64
HandsetPrice                 51047 non-null object
MadeCallToRetentionTeam      51047 non-null object
```

```
CreditRating                51047 non-null object
PrizmCode                   51047 non-null object
Occupation                  51047 non-null object
MaritalStatus               51047 non-null object
dtypes: float64(26), int64(9), object(23)
memory usage: 22.6+ MB
```

df.isnull().sum(): It gives us the number of null values in each column

```
CustomerID                    0
Churn                         0
MonthlyRevenue              156
MonthlyMinutes              156
TotalRecurringCharge        156
DirectorAssistedCalls       156
OverageMinutes              156
RoamingCalls                156
PercChangeMinutes           367
PercChangeRevenues          367
DroppedCalls                  0
BlockedCalls                  0
UnansweredCalls               0
CustomerCareCalls             0
ThreewayCalls                 0
ReceivedCalls                 0
OutboundCalls                 0
InboundCalls                  0
PeakCallsInOut                0
OffPeakCallsInOut             0
DroppedBlockedCalls           0
CallForwardingCalls           0
CallWaitingCalls              0
MonthsInService               0
UniqueSubs                    0
ActiveSubs                    0
ServiceArea                  24
Handsets                      1
HandsetModels                 1
CurrentEquipmentDays          1
AgeHH1                      909
AgeHH2                      909
ChildrenInHH                  0
HandsetRefurbished            0
HandsetWebCapable             0
```

```
TruckOwner                    0
RVOwner                       0
Homeownership                 0
BuysViaMailOrder              0
RespondsToMailOffers          0
OptOutMailings                0
NonUSTravel                   0
OwnsComputer                  0
HasCreditCard                 0
RetentionCalls                0
RetentionOffersAccepted       0
NewCellphoneUser              0
NotNewCellphoneUser           0
ReferralsMadeBySubscriber     0
IncomeGroup                   0
OwnsMotorcycle                0
AdjustmentsToCreditRating     0
HandsetPrice                  0
MadeCallToRetentionTeam       0
CreditRating                  0
PrizmCode                     0
Occupation                    0
MaritalStatus                 0
dtype: int64
```

As seen in the above sample data, predicted output column “Churn” that is the predicted Y is having value which is of nominal type.

Finding Missing Value:

(df['MonthlyRevenue']).isnull().sum()

From this above code we can find the missing values from a given dataset of any column.

```
(df['MonthlyRevenue']).isnull().sum()
```

```
156
```

MISSING VALUE HANDLING:

We treat the missing value by method of **forwardfill** and the syntax is

```
#Missing Value Handeling
df['MonthlyRevenue'].fillna(method='ffill',inplace=True)
df['MonthlyMinutes'].fillna(method='ffill',inplace=True)
df['TotalRecurringCharge'].fillna(method='ffill',inplace=True)
df['DirectorAssistedCalls'].fillna(method='ffill',inplace=True)
df['OverageMinutes'].fillna(method='ffill',inplace=True)
df['PercChangeMinutes'].fillna(method='ffill',inplace=True)
df['PercChangeRevenues'].fillna(method='ffill',inplace=True)
df['RoamingCalls'].fillna(method='ffill',inplace=True)
df['ServiceArea'].fillna(method='ffill',inplace=True)
df['Handsets'].fillna(method='ffill',inplace=True)
df['HandsetModels'].fillna(method='ffill',inplace=True)
df['CurrentEquipmentDays'].fillna(method='ffill',inplace=True)
df['AgeHH1'].fillna(method='ffill',inplace=True)
df['AgeHH2'].fillna(method='ffill',inplace=True)
```

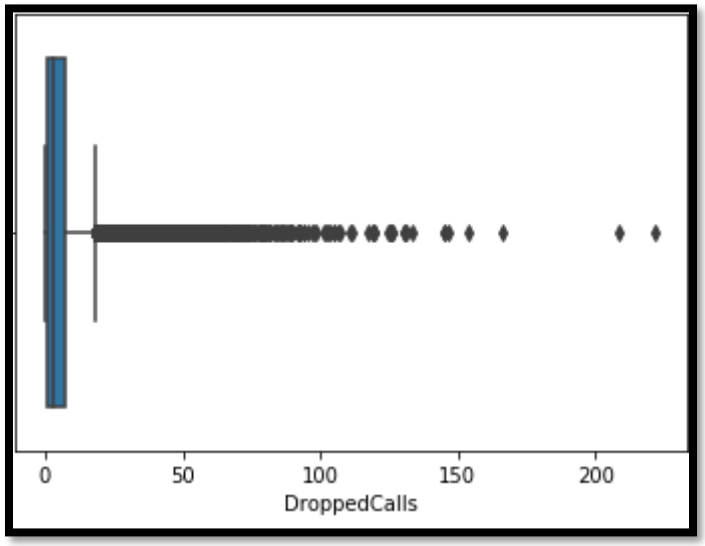
Calculation of Outliers:

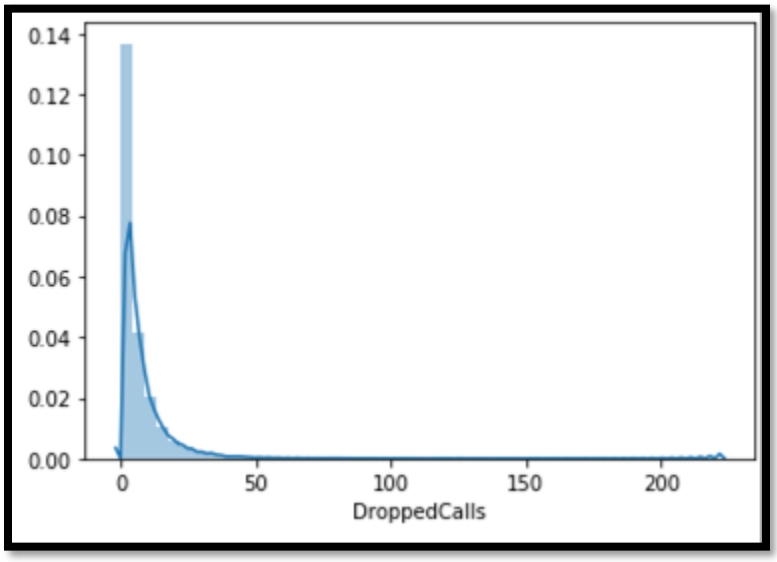
```
sorted(df['DroppedCalls'])

q1, q3= np.percentile(df['DroppedCalls'],[25,75])

iqr = q3 - q1

lower_bound = q1 -(1.5 * iqr)

upper_bound = q3 +(1.5 * iqr)
```

```
print(lower_bound,upper_bound)

tempdf=df.loc[(df['DroppedCalls']<lower_bound) |
(df['DroppedCalls']>upper_bound)]

print(len(tempdf['DroppedCalls']))

x=len(tempdf['DroppedCalls'])

y=len(df['DroppedCalls'])

print((x/y)*100)
```

OUTPUT:

```
-9.8 18.2

3712

7.271729974337375
```





We can see 3712 outliers out of 51047 instances; thus, it contributes 7.27172% of the whole sample data provided.

OUTLIER PERCENTAGE:

Variable Name	Outlier Percentage
MonthlyRevenue	5.90
MonthlyMinutes	5.05
TotalRecurringCharge	1.61
DirectorAssistedCalls	10.88
OverageMinutes	11.48
RoamingCalls	17.35
PercChangeMinutes	13.40
PercChangeRevenues	26.09
DroppedCalls	7.27
BlockedCalls	10.81
UnansweredCalls	7.11
CustomerCareCalls	13.17
ThreewayCalls	9.05
ReceivedCalls	7.13
OutboundCalls	6.54
InboundCalls	9.74
PeakCallsInOut	5.49
OffPeakCallsInOut	7.09
DroppedBlockedCalls	7.71
CallForwardingCalls	0.45
CallWaitingCalls	14.59
MonthsInService	2.38

UniqueSubs	3.67
ActiveSubs	1.19
Handsets	0
HandsetModels	0
CurrentEquipmentDays	0
AgeHH1	5.84
AgeHH2	**4.35**
ChildrenInHH	0
HandsetRefurbished	0
HandsetWebCapable	0
TruckOwner	0
RVOwner	0
Homeownership	0
BuysViaMailOrder	0
RespondsToMailOffers	0
OptOutMailings	0
NonUSTravel	0
OwnsComputer	0
HasCreditCard	0
RetentionCalls	3.42
RetentionOffersAccepted	1.72
NewCellphoneUser	0
NotNewCellphoneUser	0
ReferralsMadeBySubscriber	4.67
IncomeGroup	0
OwnsMotorcycle	0
AdjustmentsToCreditRating	0
HandsetPrice	
MadeCallToRetentionTeam	0
CreditRating	0
PrizmCode	1.95
Occupation	0
MaritalStatus	0

OUTLIER HANDLING:

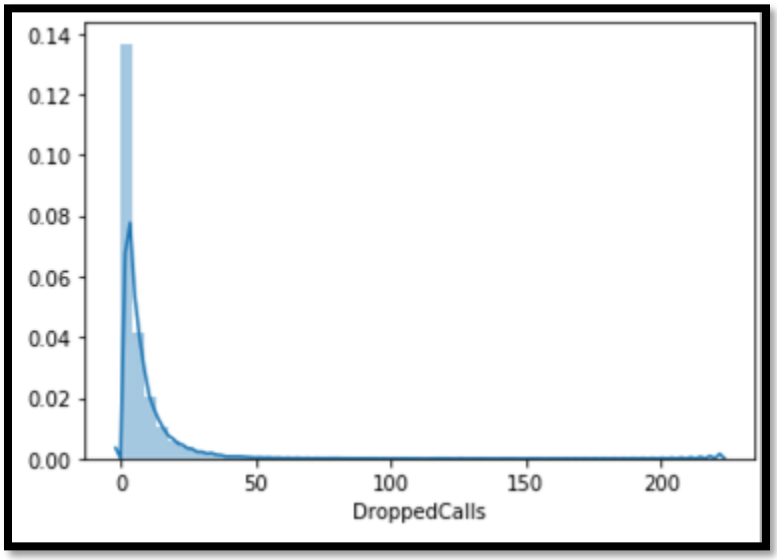
```
#Outlier Handling
outlier=[]
z=[]
for col in df:
    sorted(df[col])
    q1, q3= np.percentile(df[col],[25,75])
    iqr = q3 - q1
    lower_bound = q1 -(1.5 * iqr)
    upper_bound = q3 +(1.5 * iqr)
    #print(lower_bound,upper_bound)
    tempdf=df.loc[(df[col]<lower_bound) | (df[col]>upper_bound)]
    outlier.append(len(tempdf[col]))
    #print(tempdf.head(10))
    x=len(tempdf[col])
    y=len(df[col])
    z=((x/y))
    if z>0 and z<.05:
        df = df[~((df[col] < (lower_bound)) |(df[col] > (upper_bound)))]
    else:
        new=(np.cbrt(df[col]))
        #print(type(new))
        df1=pd.DataFrame(new)
        #print(df1.head())
        df.drop([col],axis=1,inplace=True)
        df=pd.concat((df,df1),axis=1)
print(outlier)
```

```
[0, 3012, 2582, 825, 6608, 5693, 9785, 6660, 13174, 3760, 5258, 3423, 6510, 4413, 3450, 3191, 4837, 2508, 3055, 3374, 195, 652
2, 978, 1660, 0, 2841, 1110, 973, 10383, 3856, 7872, 3380, 0, 0, 651, 0, 1324, 0, 4921, 0, 0, 0]
```

How to calculate skewness:

df['DroppedCalls'].skew()

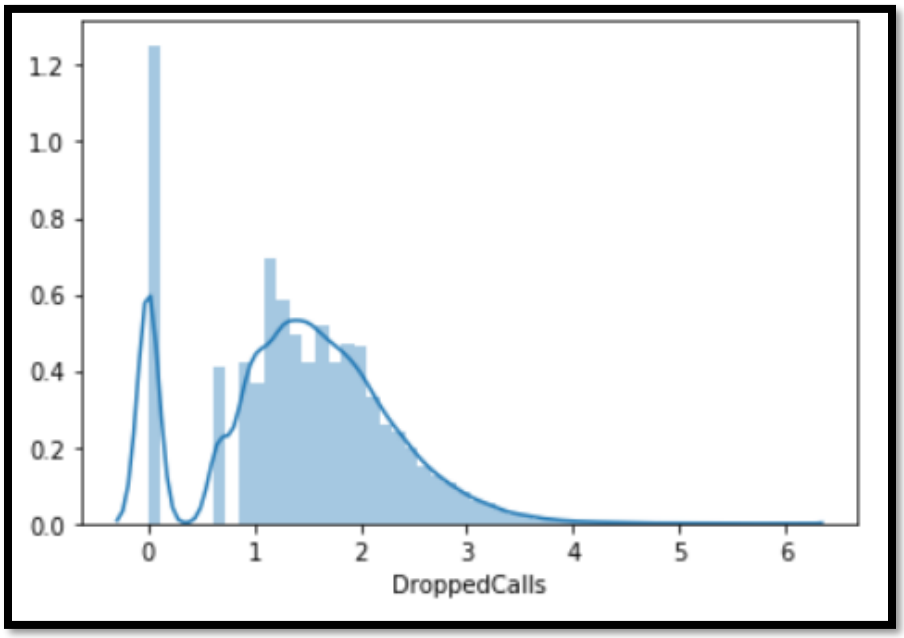
Output: 4.54



How to reduce skewness:

```
s_cbrt=np.cbrt(df['DroppedCalls'])
print(s_cbrt.skew())
print(sns.distplot(s_cbrt,kde=True))
```

Output: 0.07

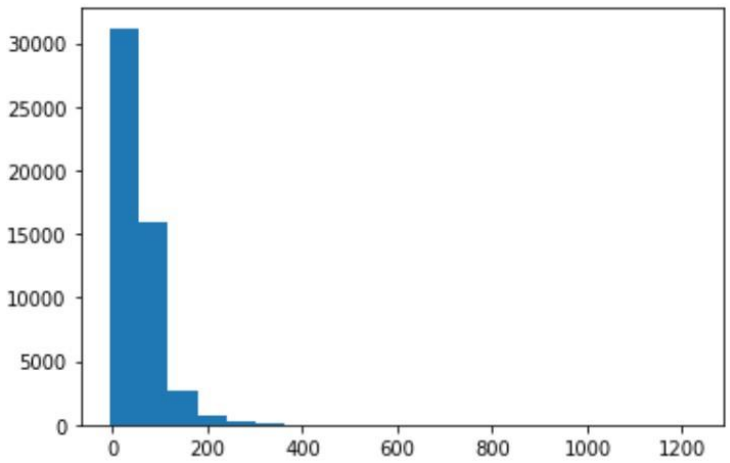


Variable Name	Skewness Value	After Reducing Skewness
MonthlyRevenue	4.17	0.79
MonthlyMinutes	2.19	0.60
TotalRecurringCharge	1.62	0.028
DirectorAssistedCalls	13.50	1.80
OverageMinutes	8.10	1.77
RoamingCalls	57.53	2.43
PercChangeMinutes	-0.32	
PercChangeRevenues	7.85	0.38

DroppedCalls	4.55	0.07
BlockedCalls	9.79	0.83
UnansweredCalls	4.38	0.013
CustomerCareCalls	14.24	1.03
ThreewayCalls	17.55	1.67
ReceivedCalls	3.12	0.16
OutboundCalls	3.53	0.022
InboundCalls	5.93	0.55
PeakCallsInOut	3.32	0.63
OffPeakCallsInOut	3.49	0.94
DroppedBlockedCalls	5.52	1.22
CallForwardingCalls	91.63	24.27
CallWaitingCalls	11.12	2.38
MonthsInService	1.05	0.53
UniqueSubs	79.63	1.71
ActiveSubs	10.64	1.81
CurrentEquipmentDays	1.08	0.02
AgeHH1	0.58	0.08
AgeHH2	0.58	-0.01
ReferralsMadeBySubscriber	36.73	4.92
Income Group	-0.17	-0.69

Difference between Skewness

Before



After

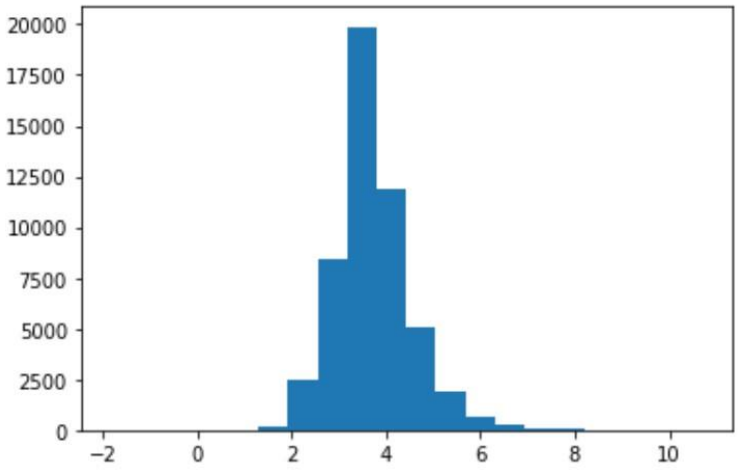
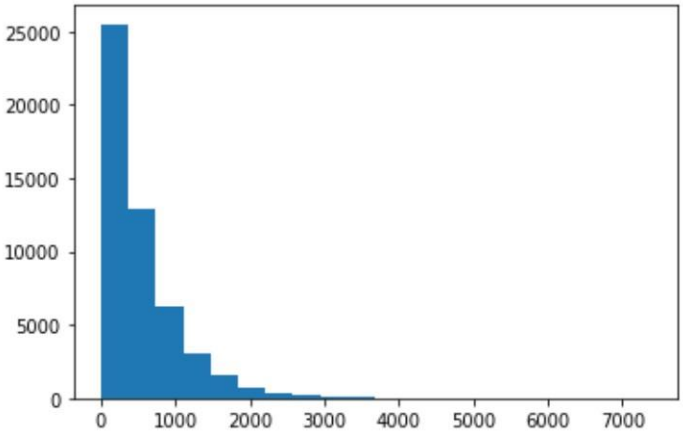


Fig:- MonthlyRevenue

Before



After

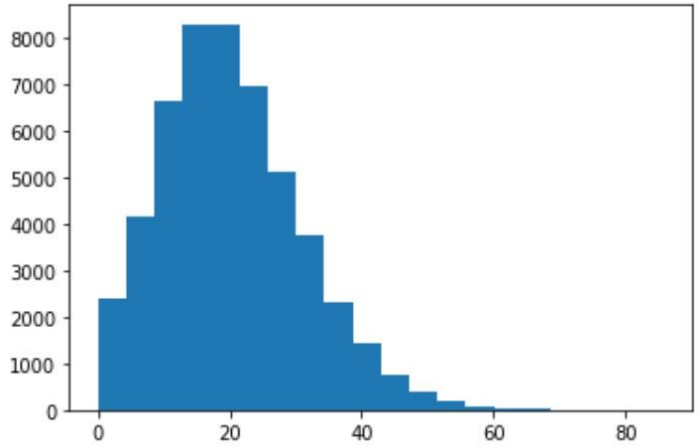
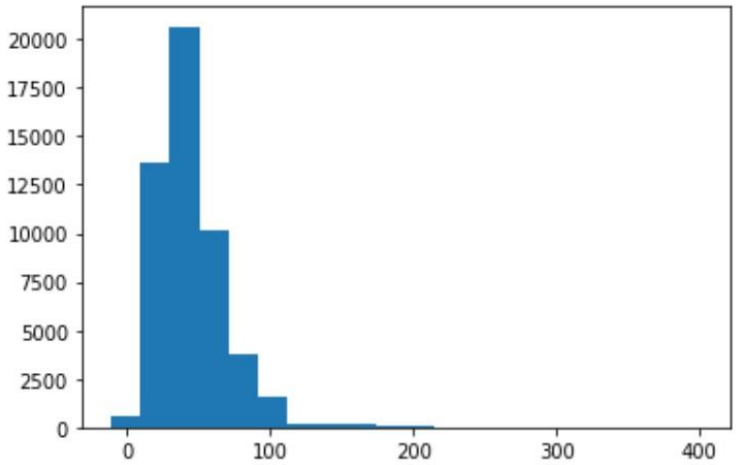


Fig:- MonthlyMinutes

Before



After

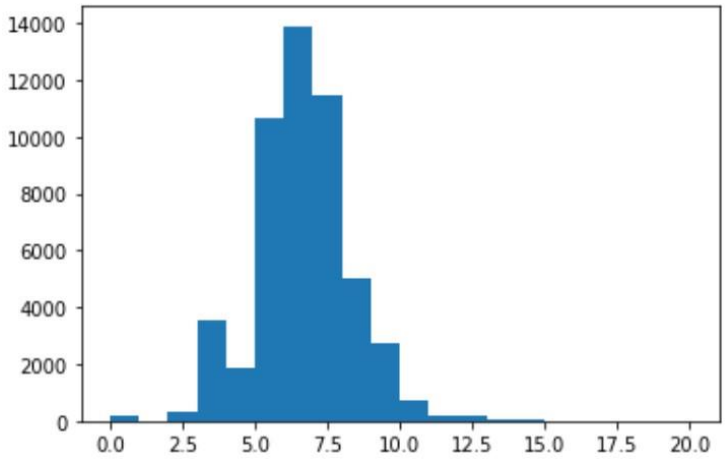
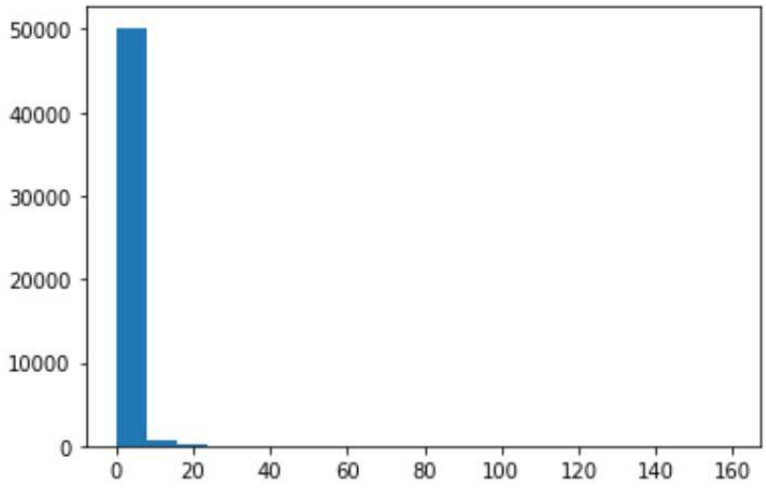


Fig:- TotalRecurringCharge

Before



After

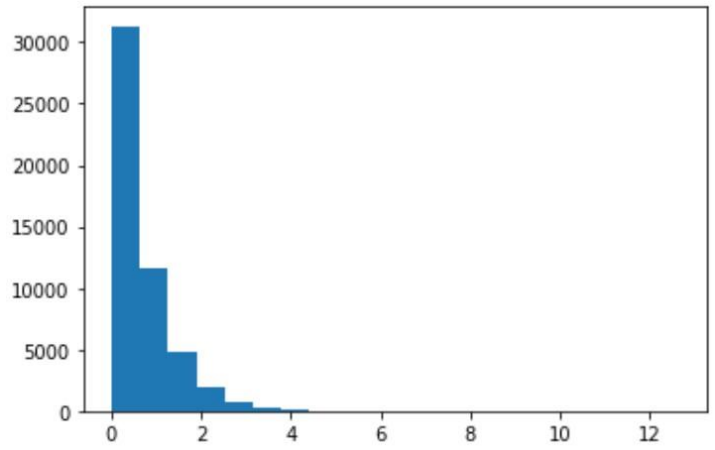
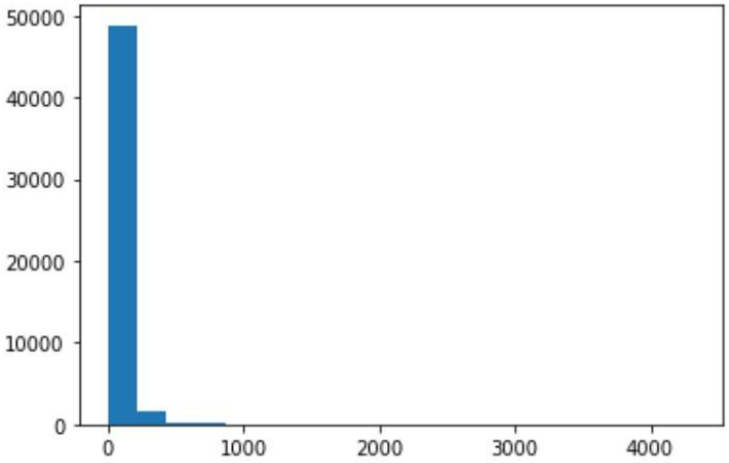


Fig:- DirectorAssistedCalls

Before



After

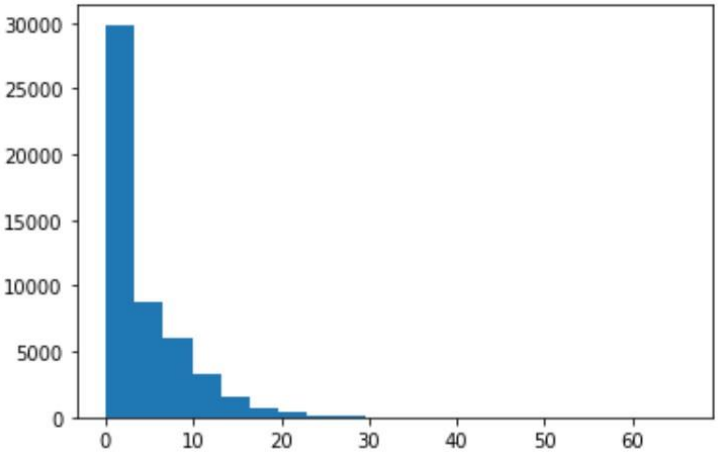


Fig:- OverageMinutes

Before **After**

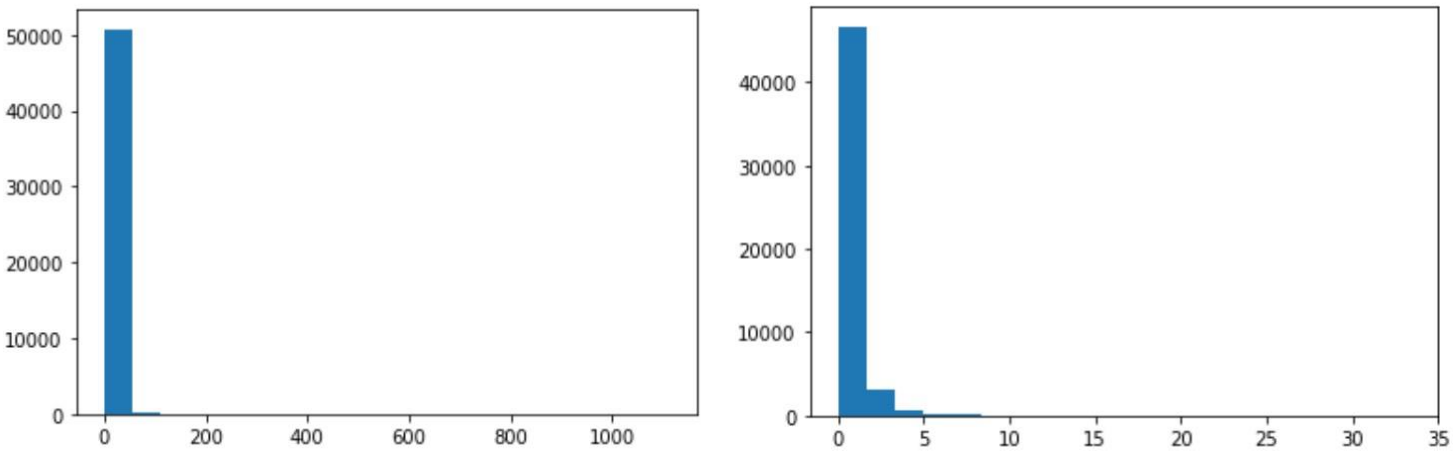


Fig:- RoamingCalls

Before **After**

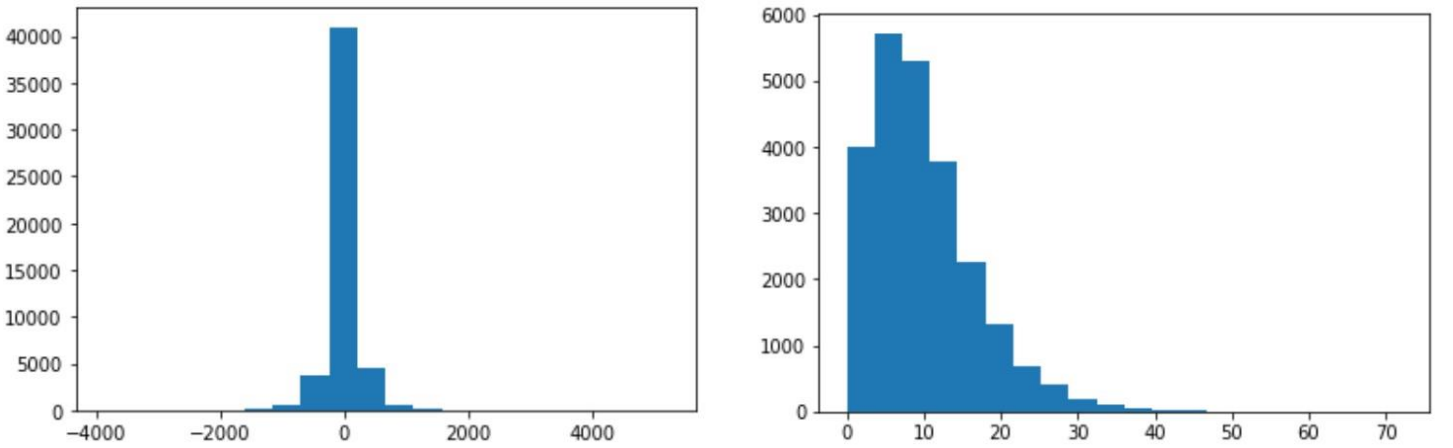


Fig:- PercChangeMinutes

Before **After**

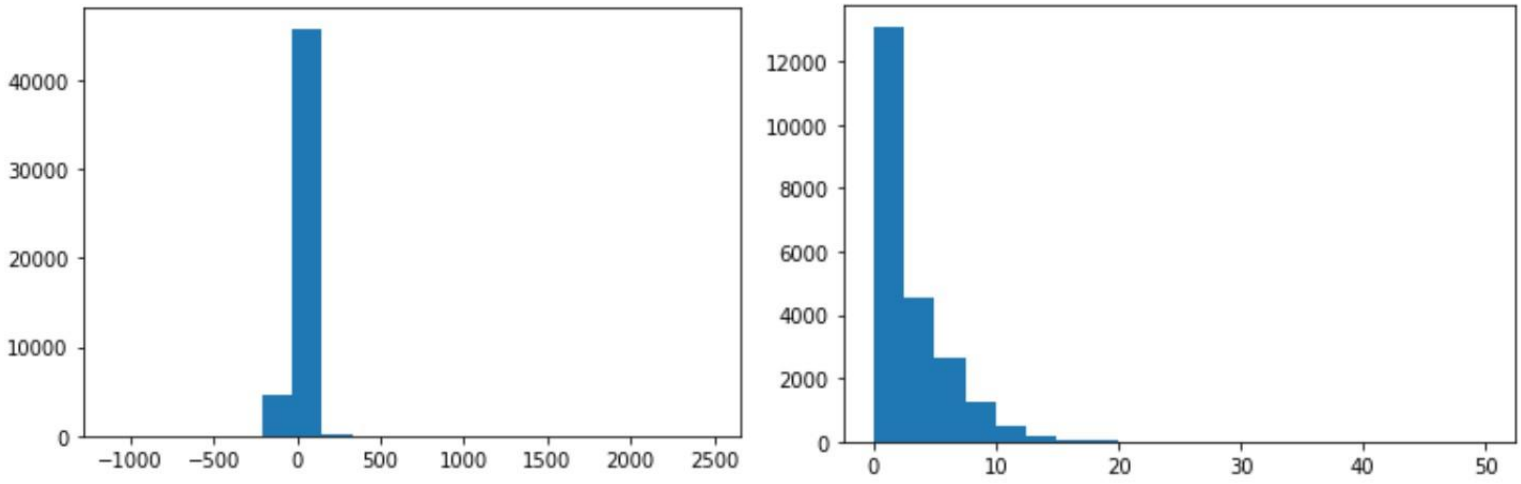


Fig:- PercChangeRevenues

Before **After**

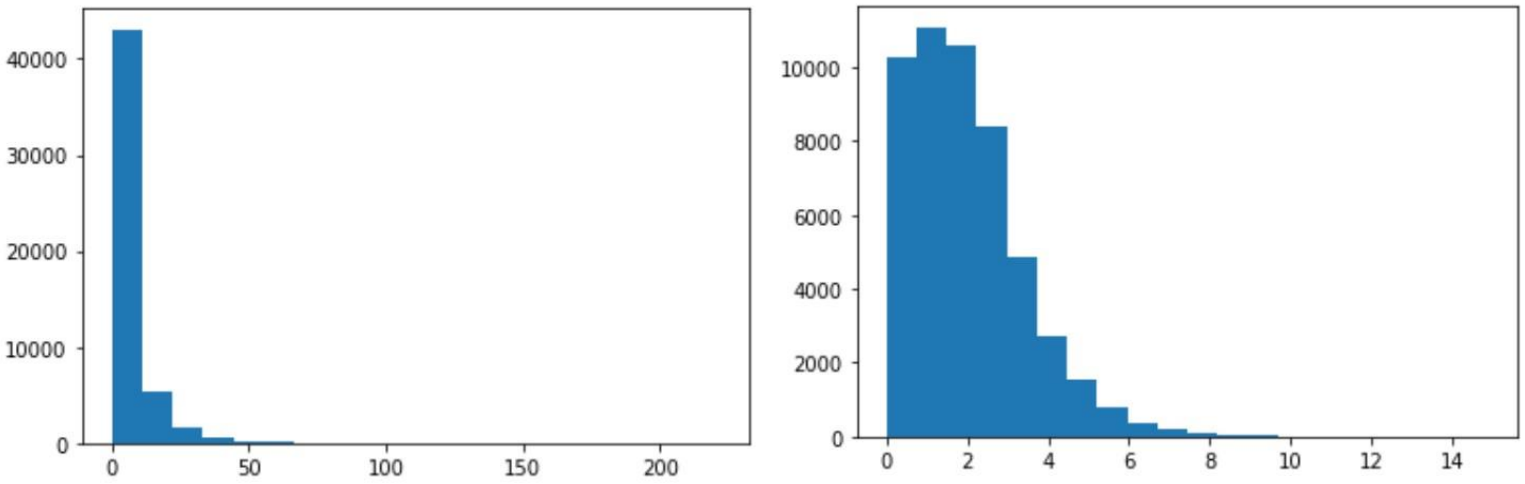
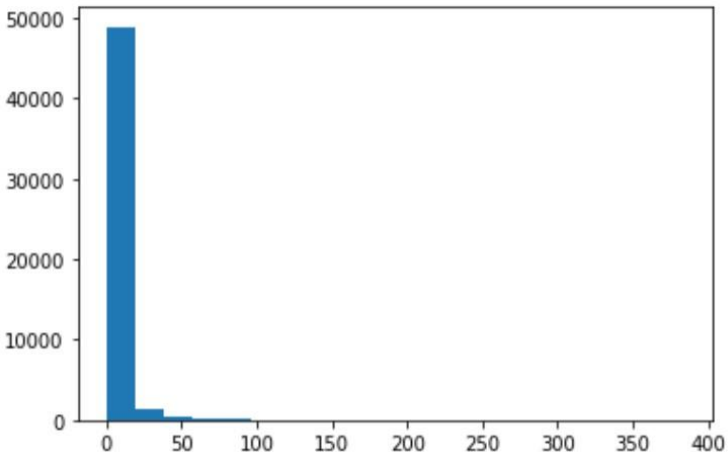


Fig:- DroppedCalls

Before



After

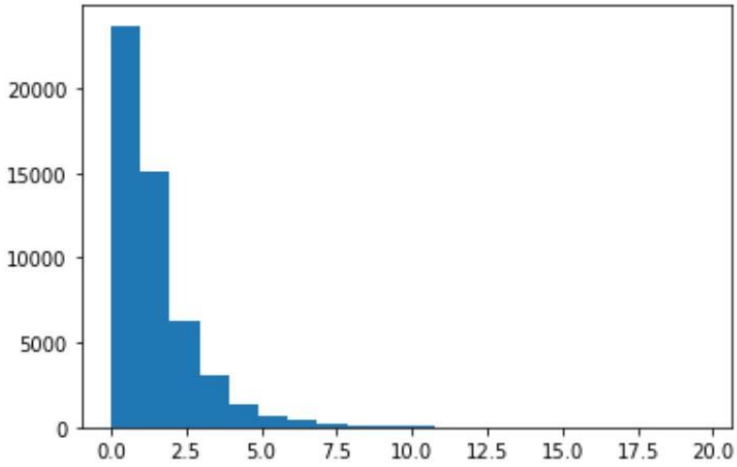
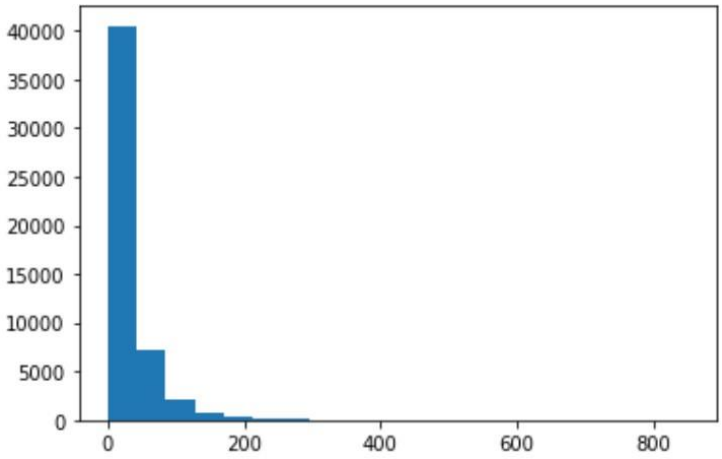


Fig:- BlockedCalls

Before



After

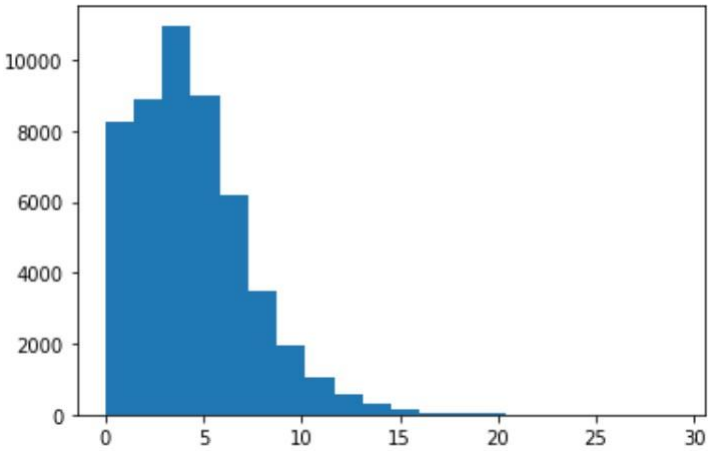
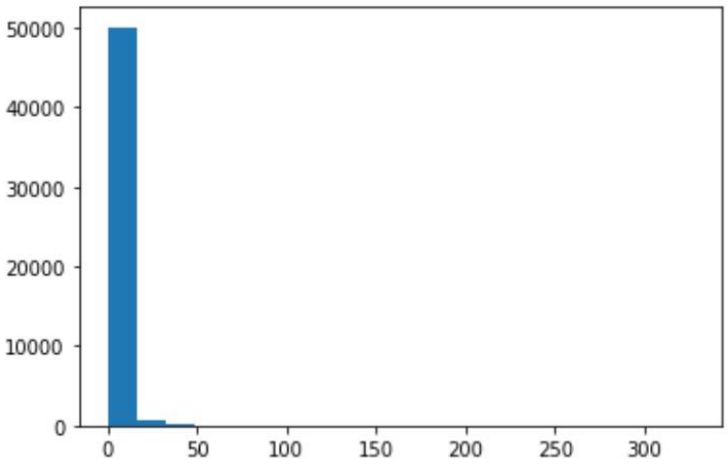


Fig:- UnansweredCalls

Before



After

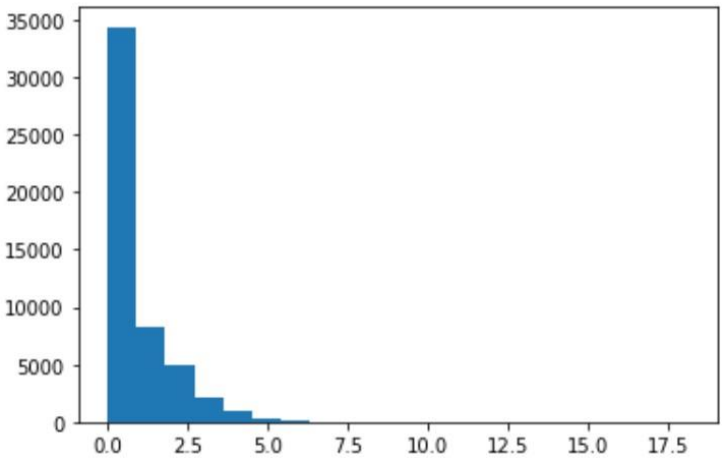
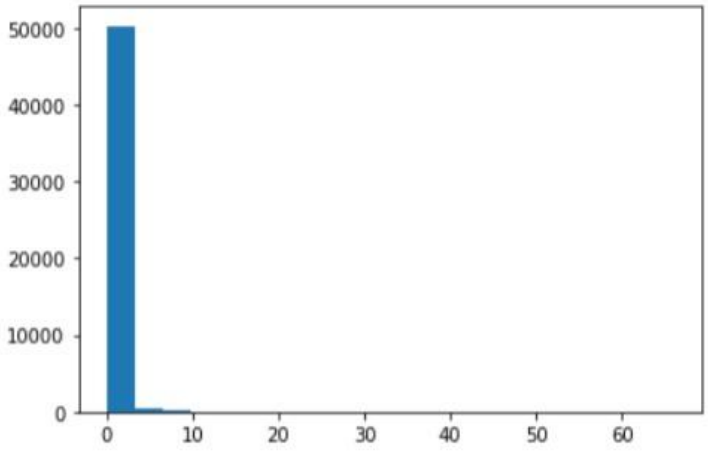


Fig:-CustomerCareCalls

Before



After

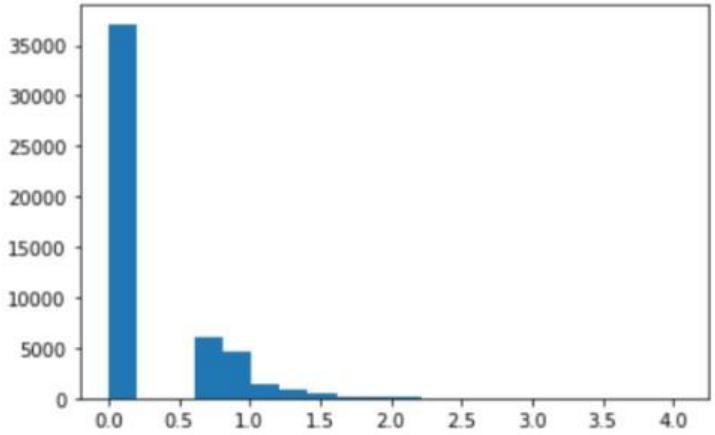
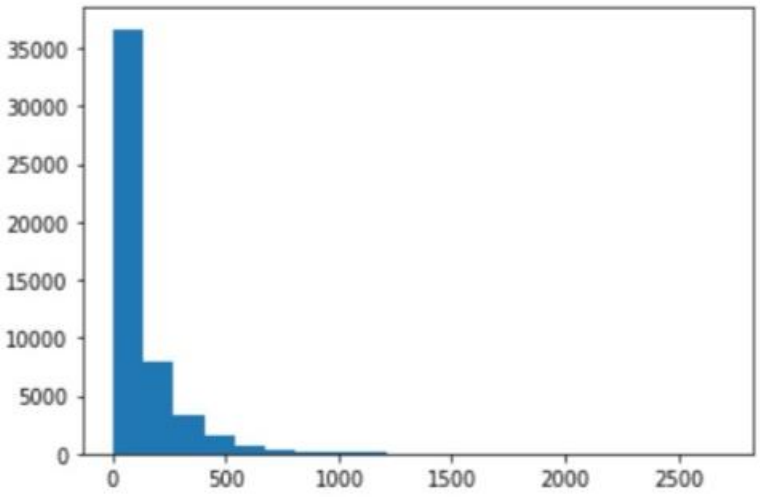


FIG.: THREE WAYS CALL

Before



After

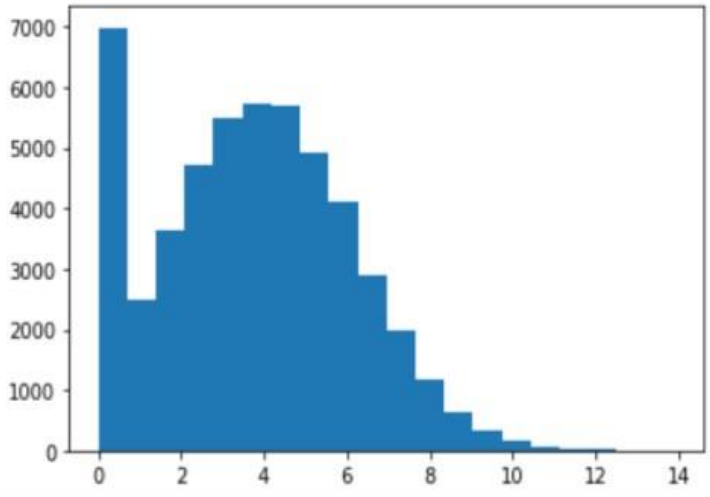
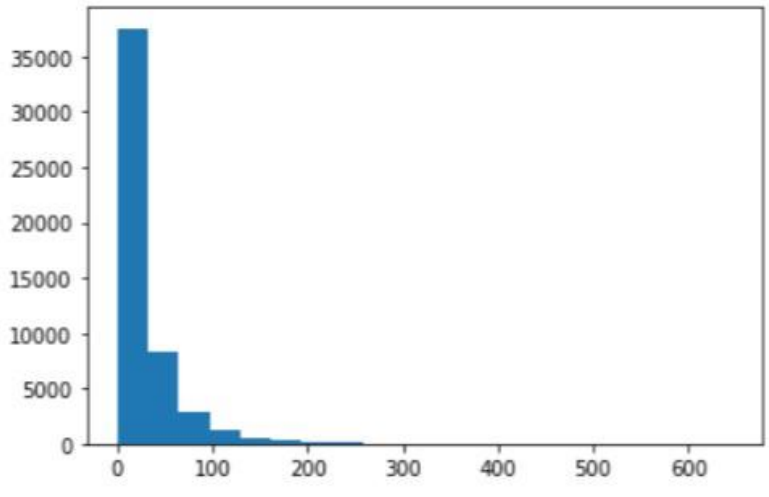


FIG.: RECEIVED CALLS

Before



After

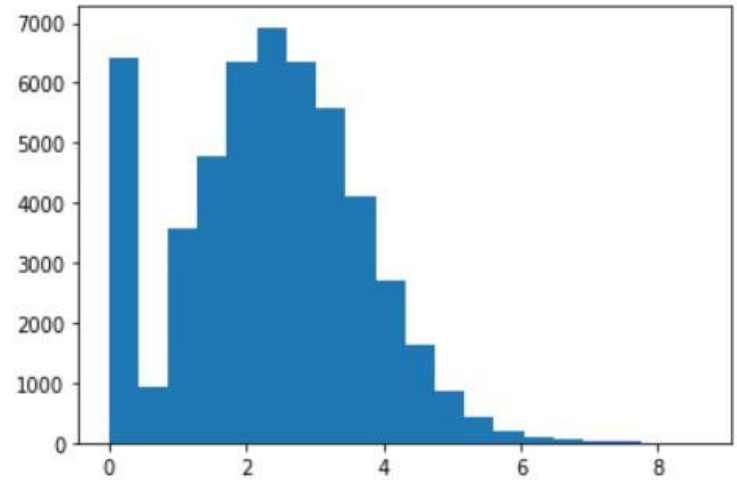


FIG. : OUT BOUND CALLS

Before **After**

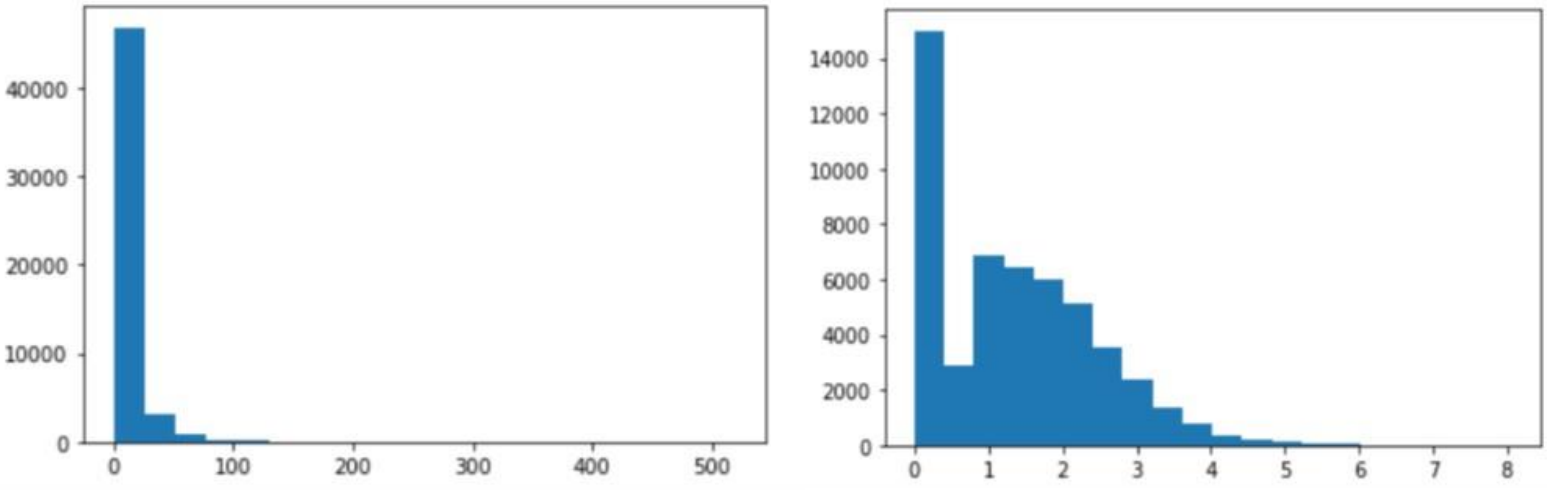


FIG : IN BOUND CALLS

Before **After**

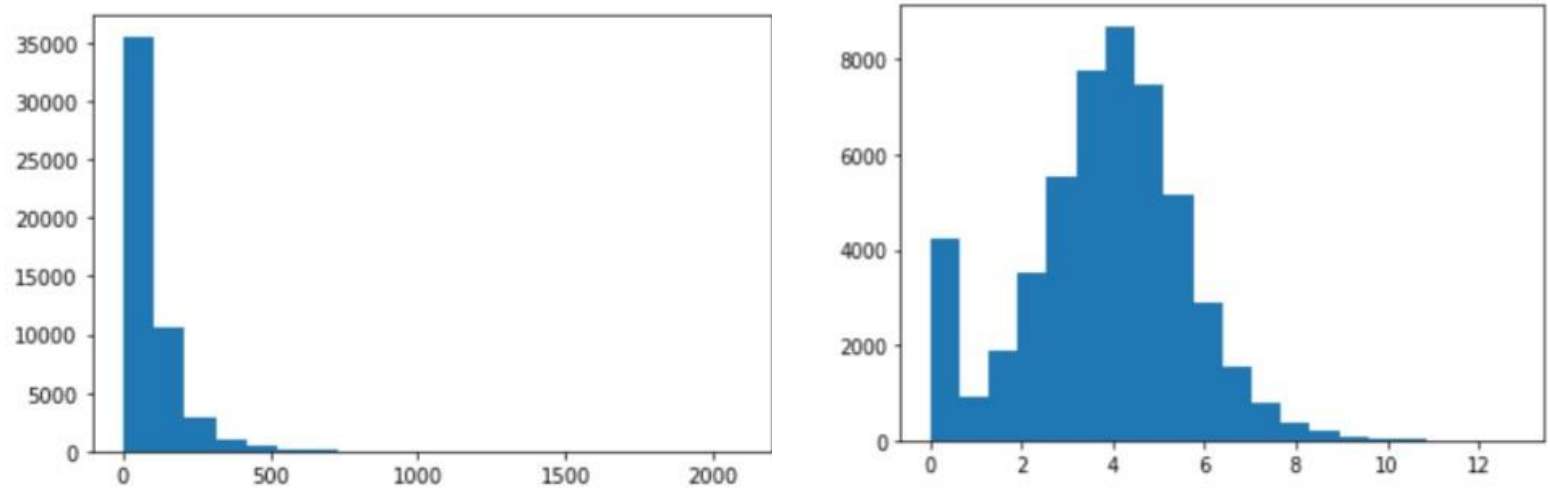


FIG : PEAK CALLS IN-OUT

Before **After**

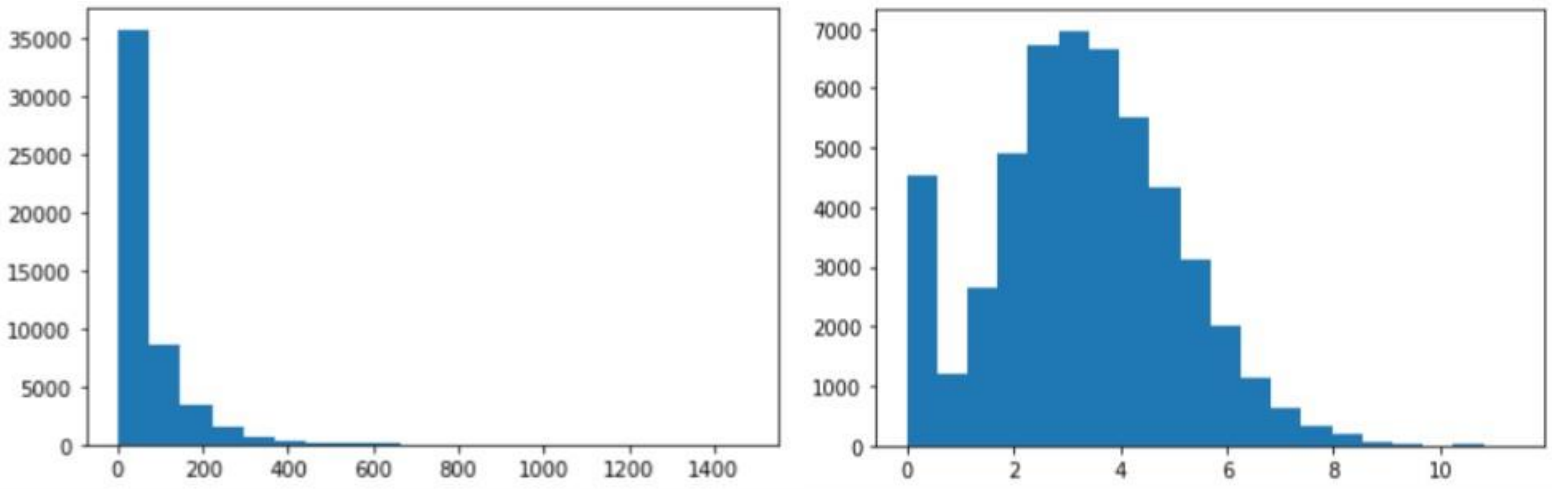


FIG : OFF PEAK CALLS IN-OUT

Before **After**

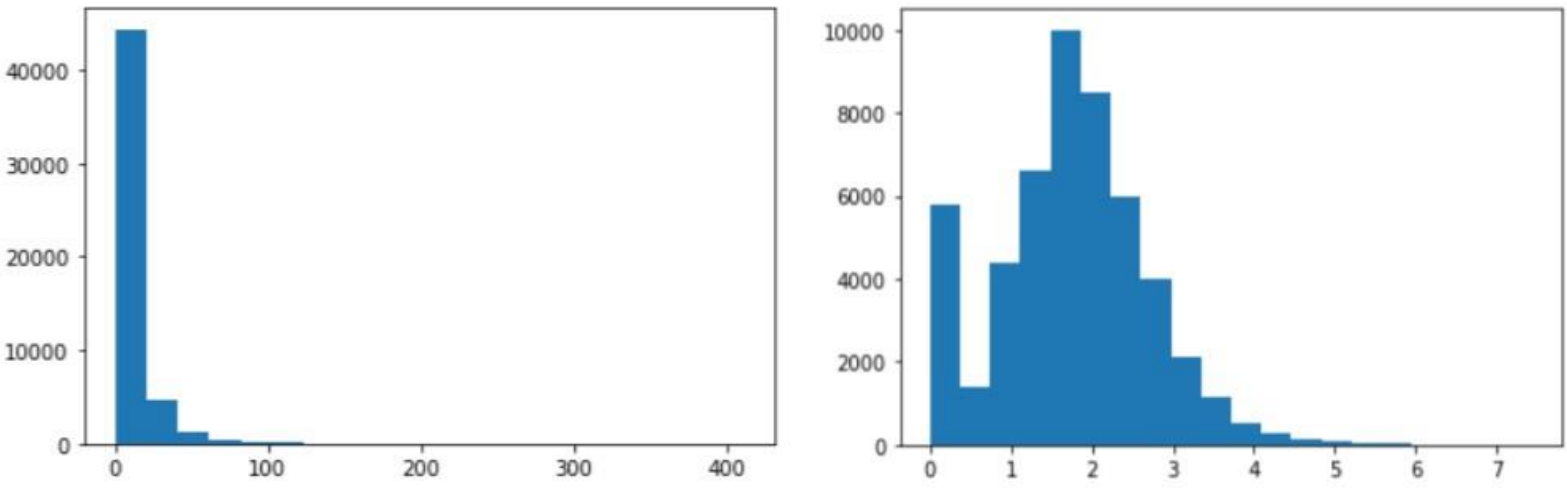


FIG : DROPPED BLOCKED CALLS

Before **After**

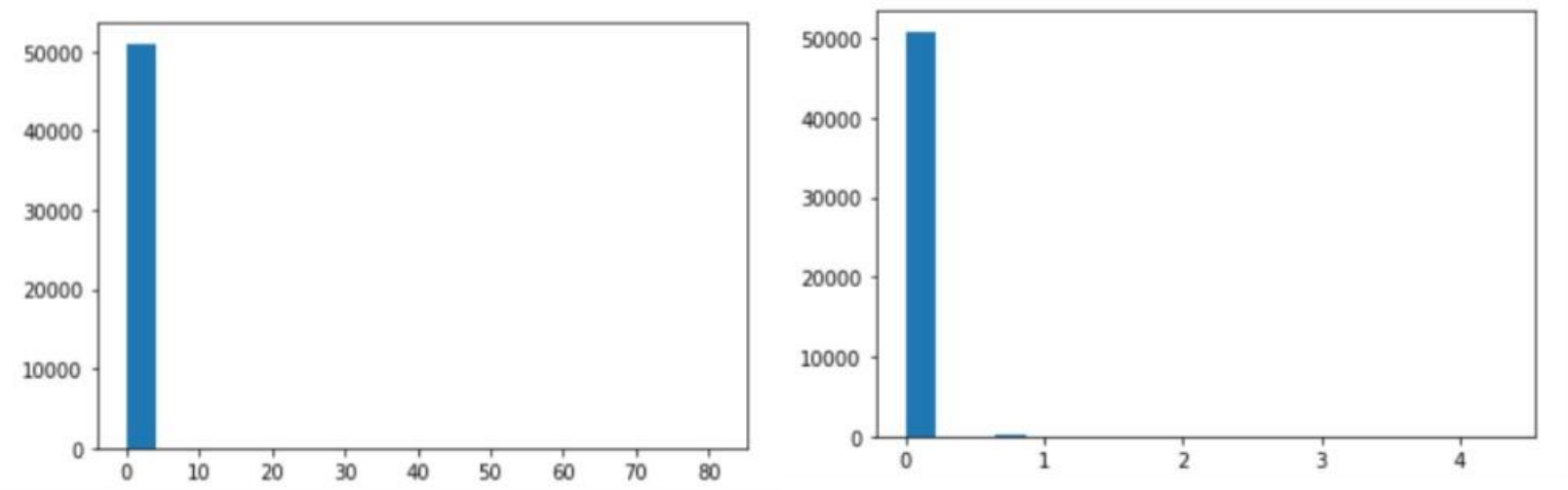


FIG : CALL FORWARDING CALLS

Before **After**

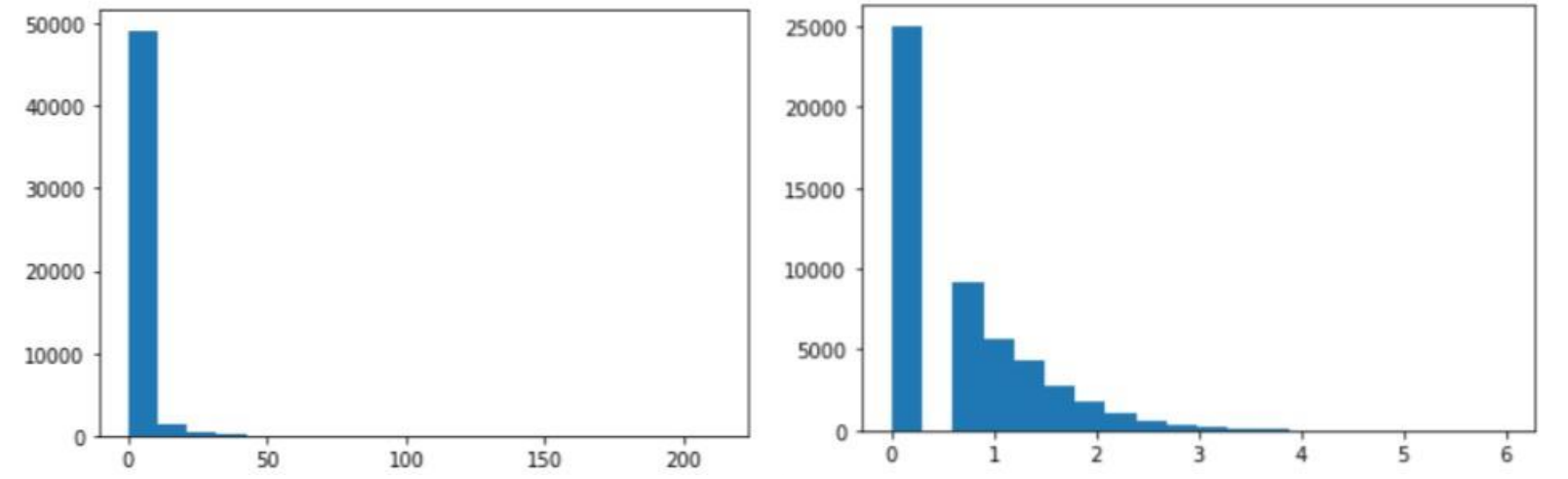


FIG : CALL WAITING CALLS

Before **After**

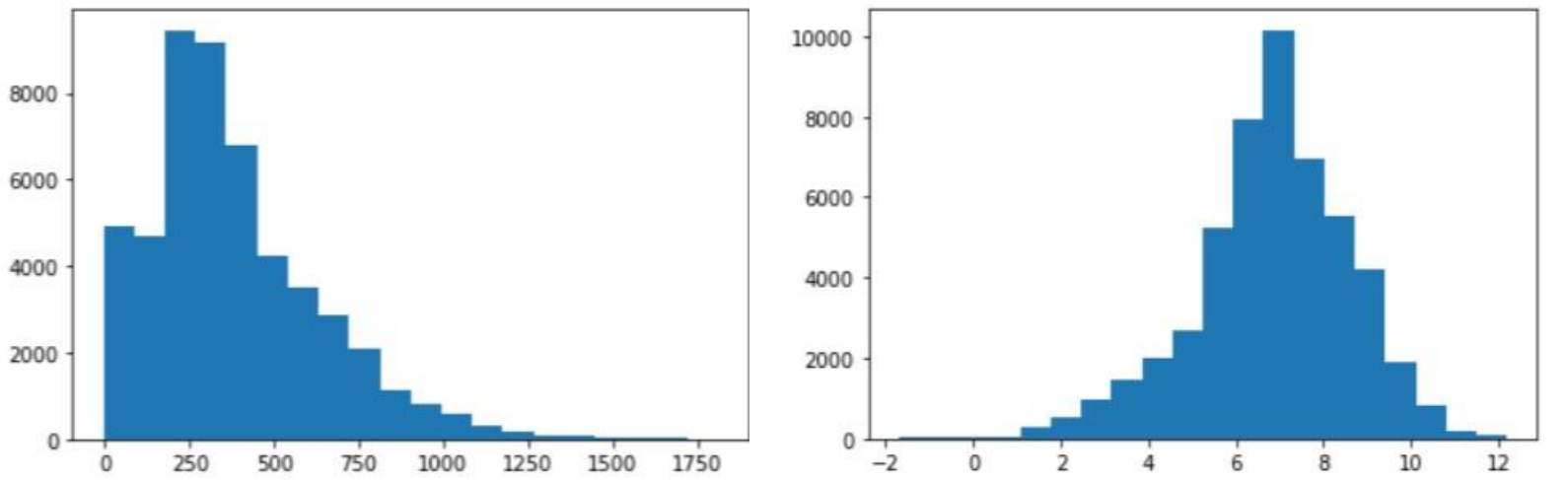


FIG : CURRENT EQUIPMENT DAYS

EDA (EXPLORATORY DATA ANALYSIS)

Variable Identification:

The original dataset downloaded contains the following –

INPUT VARIABLE	TYPE	IMPORTANCE
CustomerID	Continuous	Demographic
MonthlyRevenue	Continuous	Important
MonthlyMinutes	Continuous	Important
TotalRecurringCharge	Continuous	Important
DirectorAssistedCalls	Continuous	Important
OverageMinutes	Continuous	Important
RoamingCalls	Continuous	Important
PercChangeMinutes	Continuous	Important
PercChangeRevenues	Continuous	Important

DroppedCalls	Continuous	Important
BlockedCalls	Continuous	Important
UnansweredCalls	Continuous	Important
CustomerCareCalls	Continuous	Important
ThreewayCalls	Continuous	Important
ReceivedCalls	Continuous	Important
OutboundCalls	Continuous	Important
InboundCalls	Continuous	Important
PeakCallsInOut	Continuous	Important
OffPeakCallsInOut	Continuous	Important
DroppedBlockedCalls	Continuous	Important
CallForwardingCalls	Continuous	Important
CallWaitingCalls	Continuous	Important
MonthsInService	Discrete	Important
UniqueSubs	Discrete	Important
ActiveSubs	Discrete	Important
ServiceArea	Nominal	Demographic
Handsets	Discrete	Important
HandsetModels	Discrete	Demographic
CurrentEquipmentDays	Continuous	Important
AgeHH1	Continuous	Demographic
AgeHH2	Continuous	Demographic
ChildrenInHH	Nominal	Demographic
HandsetRefurbished	Nominal	Demographic
HandsetWebCapable	Nominal	Important
TruckOwner	Nominal	Demographic
RVOwner	Nominal	Demographic
Homeownership	Nominal	Demographic
BuysViaMailOrder	Nominal	Demographic
RespondsToMailOffers	Nominal	Demographic

OptOutMailings	Nominal	Demographic
NonUSTravel	Nominal	Demographic
OwnsComputer	Nominal	Demographic
HasCreditCard	Nominal	Important
RetentionCalls	Discrete	Important
RetentionOffersAccepted	Discrete	Important
NewCellphoneUser	Nominal	Important
NotNewCellphoneUser	Nominal	Demographic
ReferralsMadeBySubscriber	Discrete	Important
IncomeGroup	Discrete	Important
OwnsMotorcycle	Nominal	Demographic
AdjustmentsToCreditRating	Discrete	Important
HandsetPrice	Continuous	Demographic
MadeCallToRetentionTeam	Nominal	Important
CreditRating	Discrete	Important
PrizmCode	Discrete	Important
Occupation	Discrete	Demographic
MaritalStatus	Nominal	Important

OUTPUT VARIABLE:

The output variable of the model is '**Churn'** and it's a Nominal variable.

Giving values of output(Churn) a numerical representation:

Churn
Yes
Yes
No
No
Yes

Churn
1
1
0
0
1

Here we can see that in the 1st column filled with “Yes” & ”No” In the 2nd column filled with “0” & ” 1” It is easy to understand that the “0” & ”1” is the nuumerical representation of the “Yes” & ”No” .

```
le_Gender=LabelEncoder()

df['Churn']=le_Gender.fit_transform(df['Churn'])
#print(df.head())
df['ChildrenInHH']=le_Gender.fit_transform(df['ChildrenInHH'])
df['HandsetRefurbished']=le_Gender.fit_transform(df['HandsetRefurbished'])
df['HandsetWebCapable']=le_Gender.fit_transform(df['HandsetWebCapable'])
df['TruckOwner']=le_Gender.fit_transform(df['TruckOwner'])
df['Homeownership']=le_Gender.fit_transform(df['Homeownership'])
df['BuysViaMailOrder']=le_Gender.fit_transform(df['BuysViaMailOrder'])
df['RespondsToMailOffers']=le_Gender.fit_transform(df['RespondsToMailOffers'])
df['OptOutMailings']=le_Gender.fit_transform(df['OptOutMailings'])
df['NonUSTravel']=le_Gender.fit_transform(df['NonUSTravel'])
df['OwnsComputer']=le_Gender.fit_transform(df['OwnsComputer'])
df['HasCreditCard']=le_Gender.fit_transform(df['HasCreditCard'])
df['NewCellphoneUser']=le_Gender.fit_transform(df['NewCellphoneUser'])
df['NotNewCellphoneUser']=le_Gender.fit_transform(df['NotNewCellphoneUser'])
df['ReferralsMadeBySubscriber']=le_Gender.fit_transform(df['ReferralsMadeBySubscriber'])
df['OwnsMotorcycle']=le_Gender.fit_transform(df['OwnsMotorcycle'])
df['MaritalStatus']=le_Gender.fit_transform(df['MaritalStatus'])
df['MadeCallToRetentionTeam']=le_Gender.fit_transform(df['MadeCallToRetentionTeam'])
df['RVOwner']=le_Gender.fit_transform(df['RVOwner'])
```

From the above syntax we convert the 1st **“Churn”** column to the 2nd one for representing the **Nominal** form.

BIVARIATE ANALYSIS

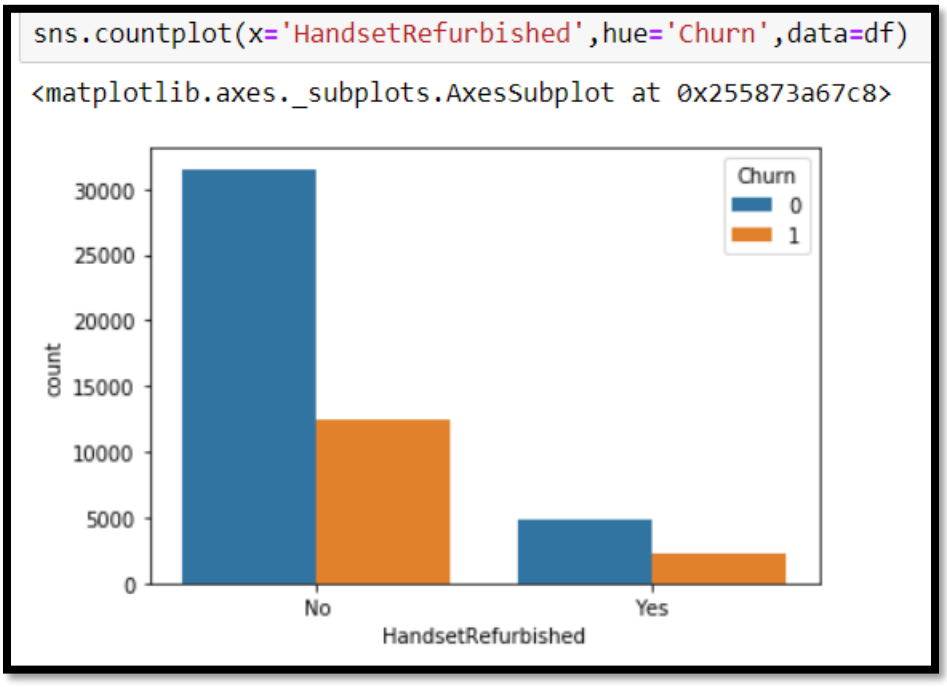
This feature is done to understand the variation of one parameter with respect to other.

It also helps us to find out the mutual dependencies of the columns.

- **Bivariate analysis of HandsetRefurbished column with respect to the output column Churn:-**

```
sns.countplot(x='HandsetRefurbished',hue='Churn',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x255873a67c8>
```

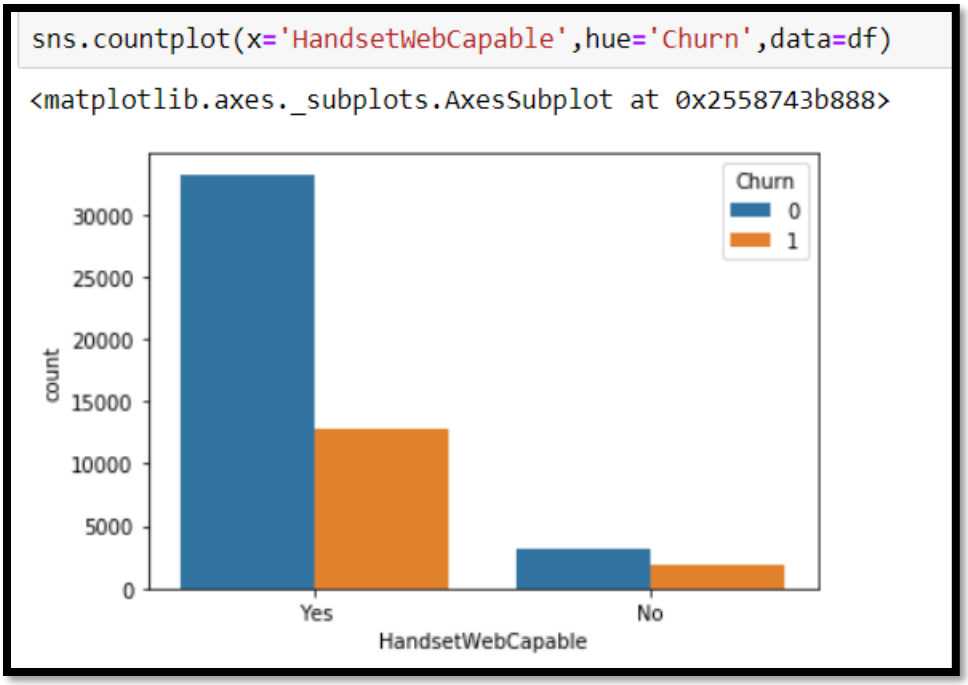


Refurbished Handset users are less chance to leave the company.

- **Bivariate analysis of 'HandsetWebCapable' column with respect to the output column ‘Churn’:-**

```
sns.countplot(x='HandsetWebCapable',hue='Churn',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2558743b888>
```

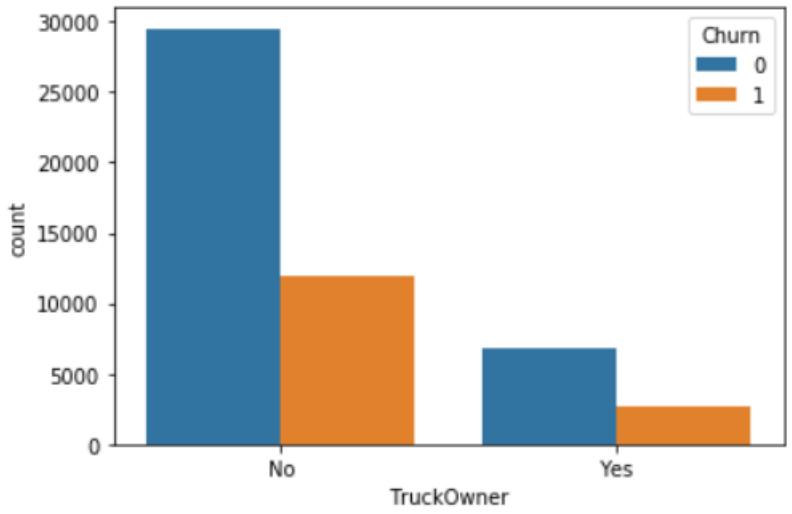


Web capable handset users have more chance to leave the company.

- **Bivariate analysis of ' TruckOwner' column with respect to the output column ‘Churn’:-**

```
sns.countplot(x='TruckOwner',hue='Churn',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x25587aa66c8>
```

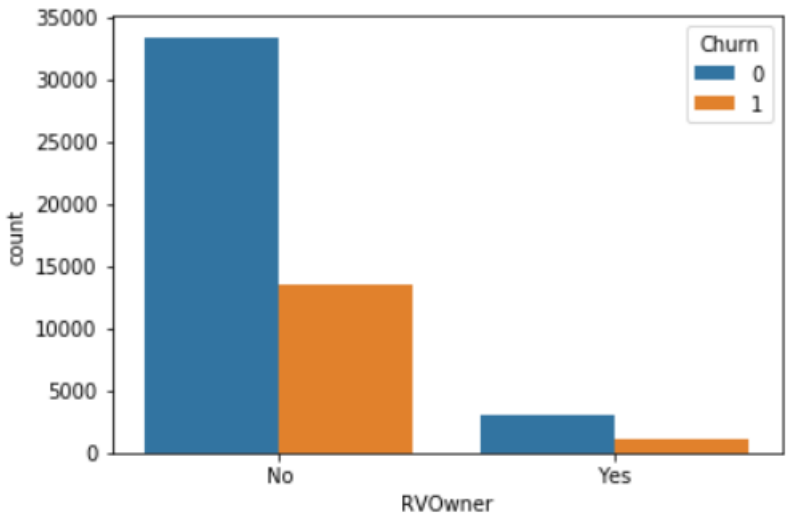


Truck owners have less chance to leave the company.

- **<u>Bivariate analysis of ' RVOwner' column with respect to the output column ‘Churn’:-</u>**

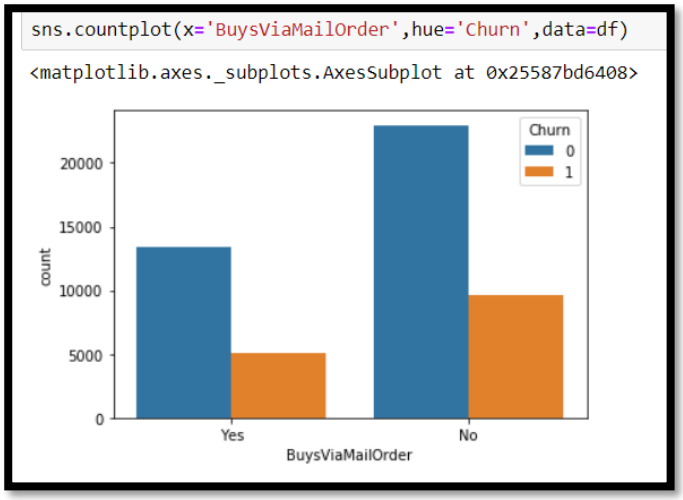
```
sns.countplot(x='RVOwner',hue='Churn',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x25587b06a48>
```



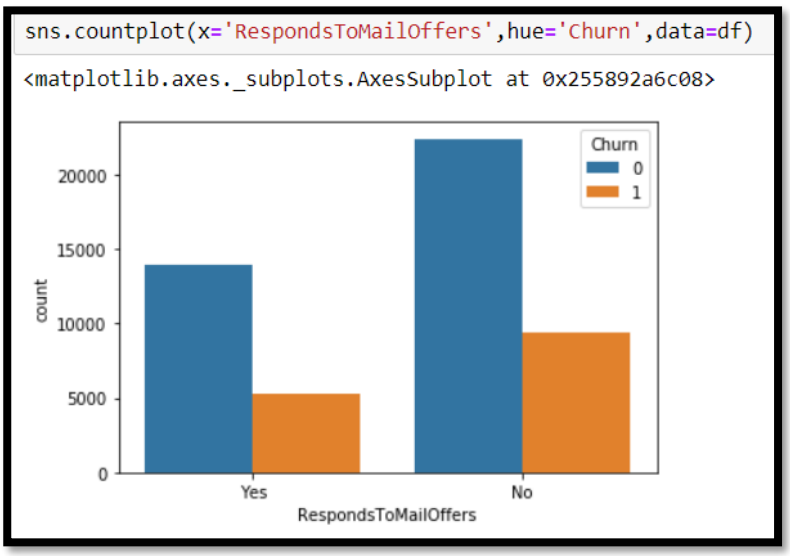
RV owners have less chance to leave the company.

- **Bivariate analysis of ' BuysViaMailOrder' column with respect to the output column 'Churn':-**



Customers who do not buys something via mail has more chance to stay at the company.

- **Bivariate analysis of 'RespondsToMailOffers' column with respect to the output column 'Churn':-**

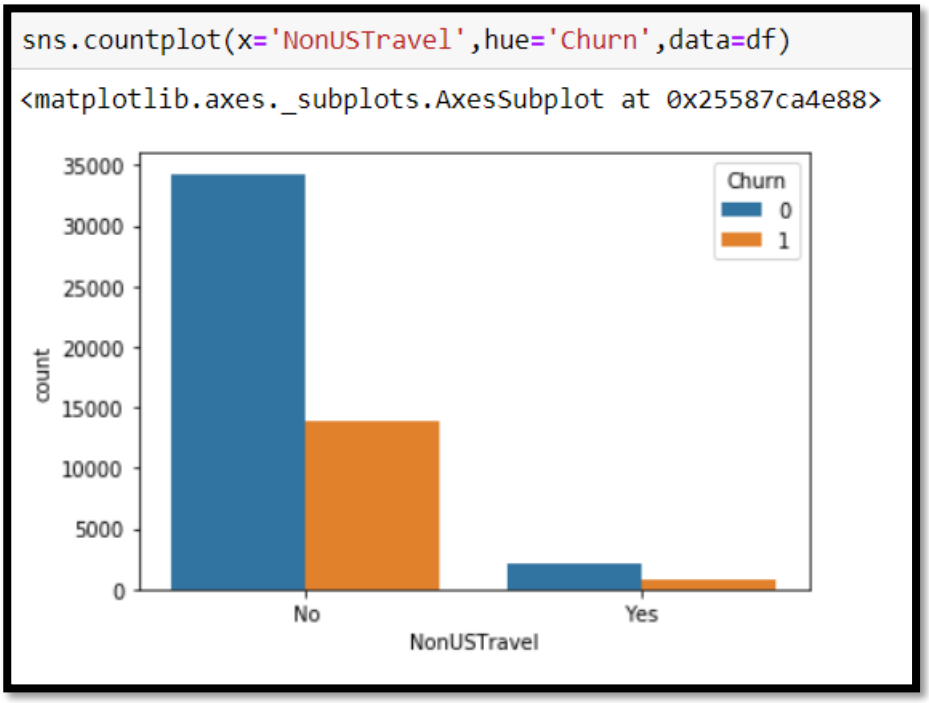


Customers who do not Response any mail offers have more chance to stay at the company.

- **Bivariate analysis of 'NonUSTravel' column with respect to the output column 'Churn':**

```
sns.countplot(x='NonUSTravel',hue='Churn',data=df)
```

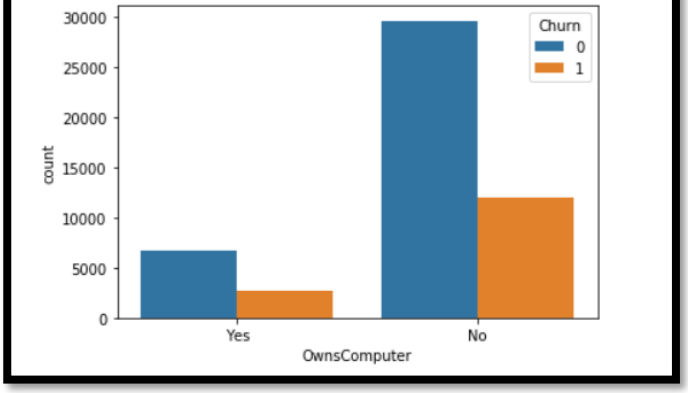
```
<matplotlib.axes._subplots.AxesSubplot at 0x25587ca4e88>
```



Customer who travels US has more chance to stay at the company.

- **Bivariate analysis of 'OwnsComputer' column with respect to the output column 'Churn':**

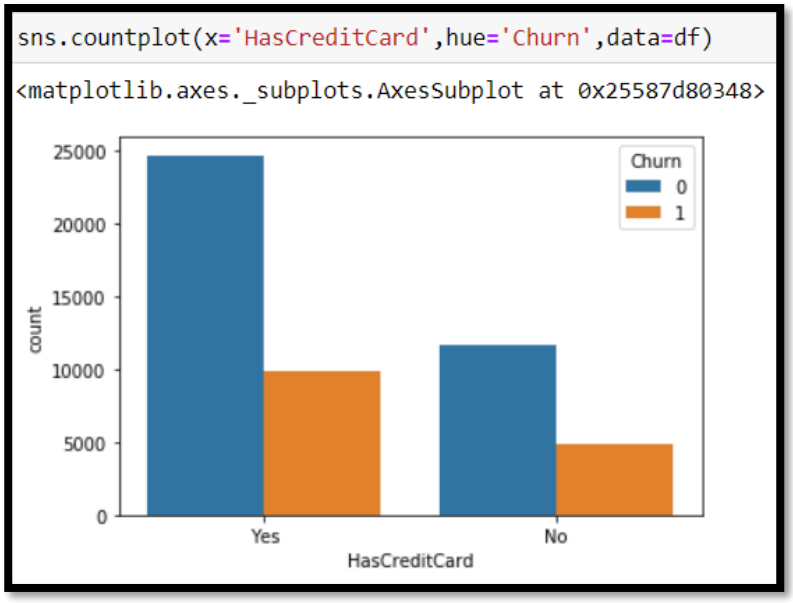
```
sns.countplot(x='OwnsComputer',hue='Churn',data=df)
<matplotlib.axes._subplots.AxesSubplot at 0x25587d12248>
```



Customer does not owns computer has more chance to stay at the company.

- **Bivariate analysis of 'HasCreditCard' column with respect to the output column 'Churn':**

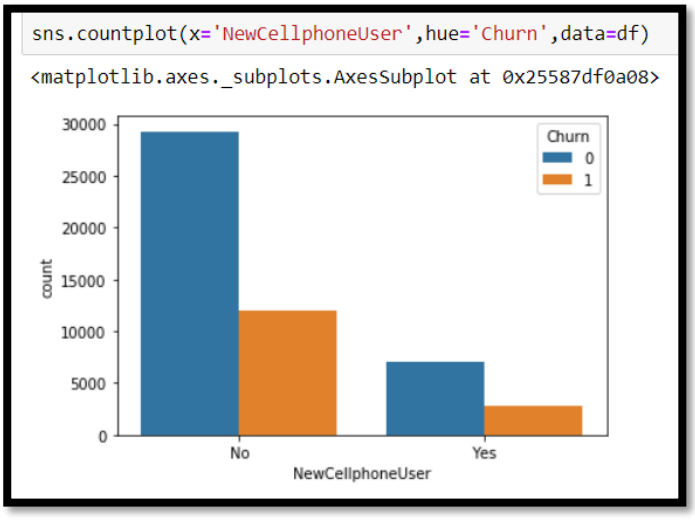
```
sns.countplot(x='HasCreditCard',hue='Churn',data=df)
<matplotlib.axes._subplots.AxesSubplot at 0x25587d80348>
```



Customer who use credit card has more chance to stay at the company.

- **Bivariate analysis of 'NewCellphoneUser' column with respect to the output column 'Churn':**

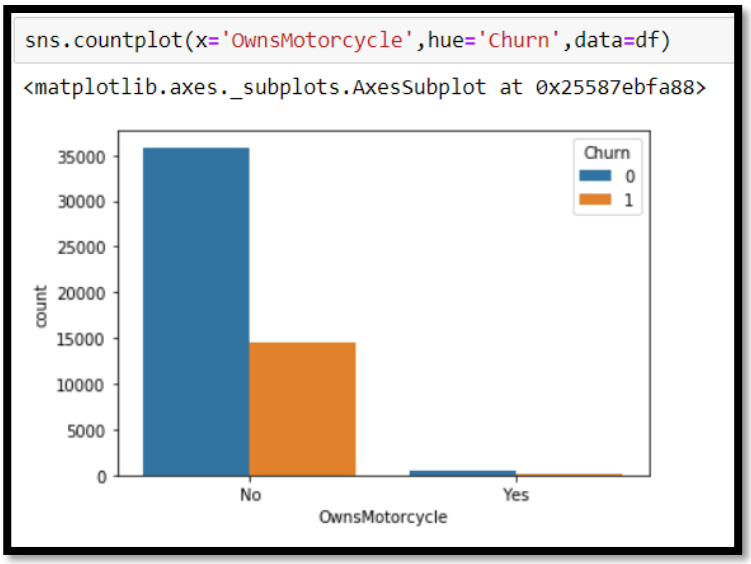
```
sns.countplot(x='NewCellphoneUser',hue='Churn',data=df)
<matplotlib.axes._subplots.AxesSubplot at 0x25587df0a08>
```



New cell phone users have less chance to leave the company.

- **<u>Bivariate analysis of 'OwnsMotorcycle' column with respect to the output column 'Churn':</u>**

```
sns.countplot(x='OwnsMotorcycle',hue='Churn',data=df)
<matplotlib.axes._subplots.AxesSubplot at 0x25587ebfa88>
```

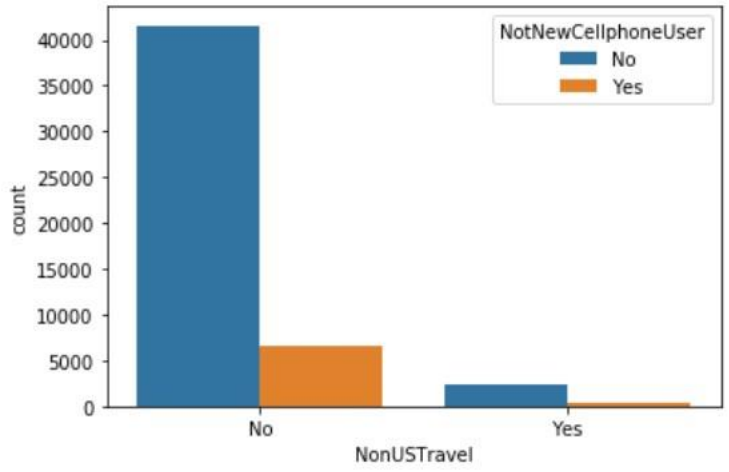


Motorcycle owners have less chance to leave the company.

- **<u>Bivariate analysis of 'NonUSTravel' column with respect to the output column 'NotNewCellphoneUser':</u>**

```
sns.countplot(x='NonUSTravel',hue='NotNewCellphoneUser',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x14520bd60c8>
```

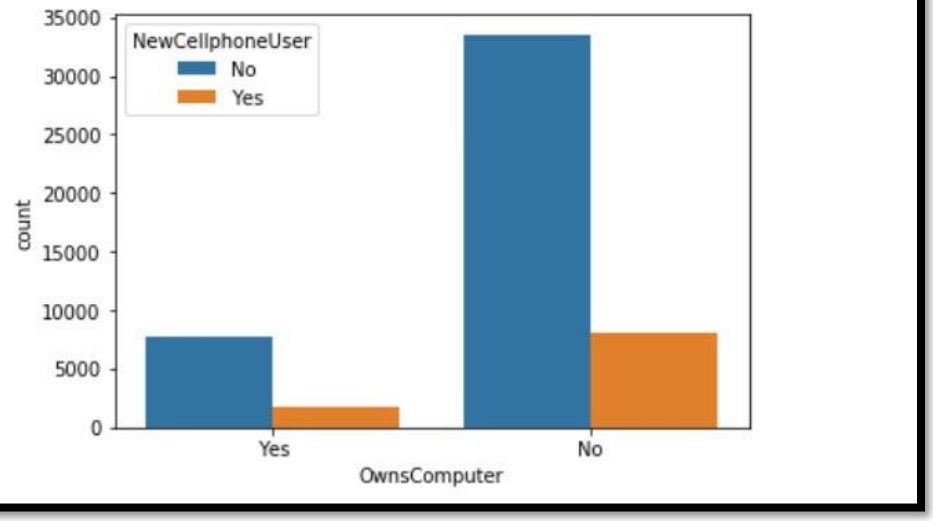


US travelers are new cellphone user.

- **<u>Bivariate analysis of 'OwnsComputer' column with respect to the output column 'NewCellphoneUser':</u>**

```
sns.countplot(x='OwnsComputer',hue='NewCellphoneUser',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x145227bc308>
```

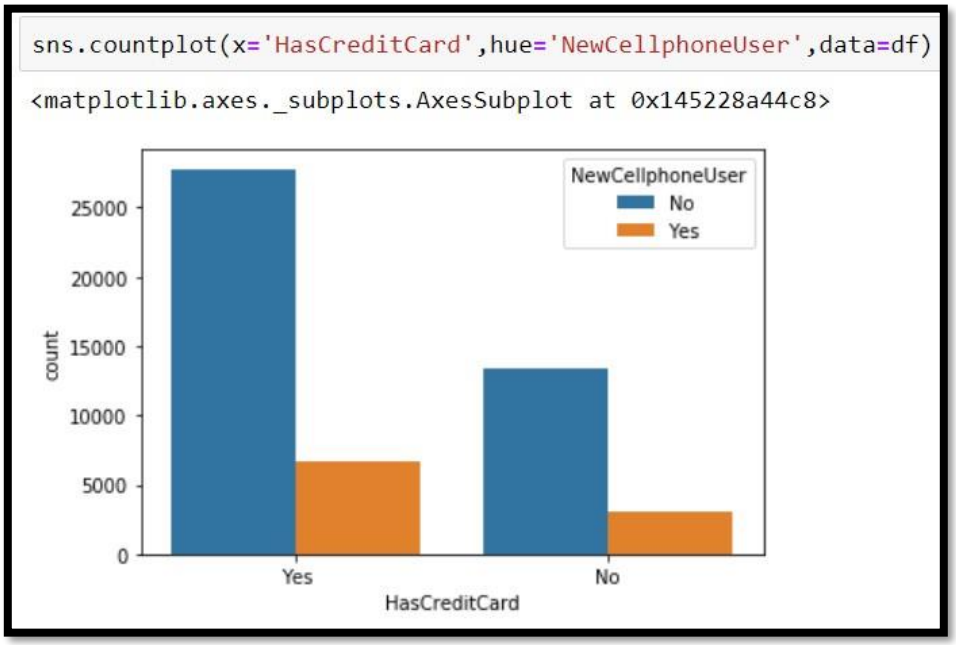


Old cellphone users are not owns computer than the new cell phone user.

- **Bivariate analysis of 'HasCreditCard' column with respect to the output column 'NewCellphoneUser':**

```
sns.countplot(x='HasCreditCard',hue='NewCellphoneUser',data=df)
```

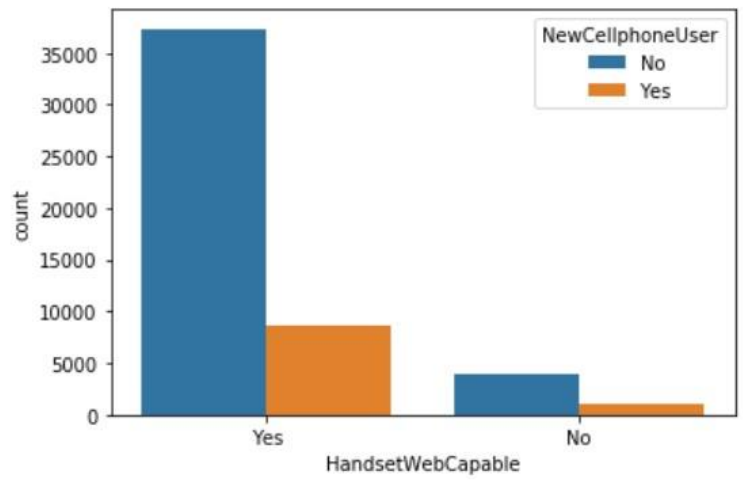
```
<matplotlib.axes._subplots.AxesSubplot at 0x145228a44c8>
```



More old cellphone users has credit card than the new cellphone user.

- **Bivariate analysis of 'HandsetWebCapable' column with respect to the output column 'NewCellphoneUser':**

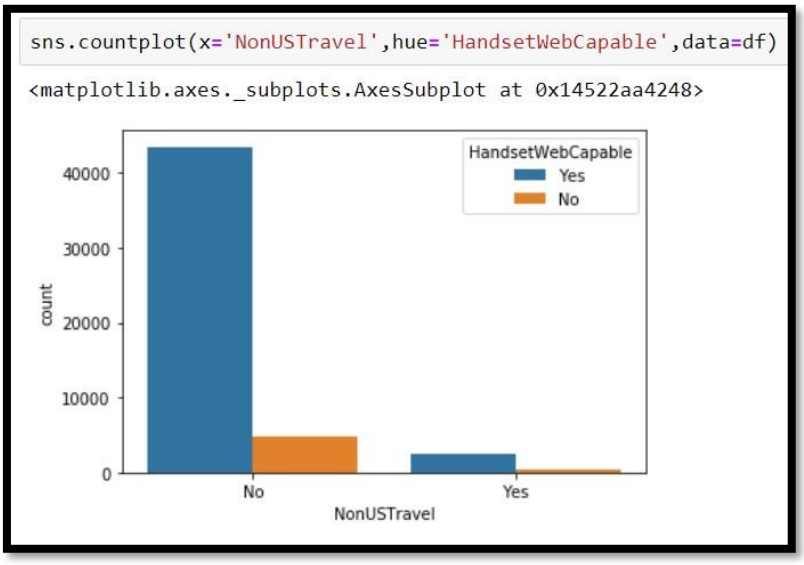
```
sns.countplot(x='HandsetWebCapable',hue='NewCellphoneUser',data=df)
<matplotlib.axes._subplots.AxesSubplot at 0x145229630c8>
```



New cell phone users handsets are webcapable.

- **<u>Bivariate analysis of 'NonUSTravel' column with respect to the output column 'HandsetWebCapable':</u>**

```
sns.countplot(x='NonUSTravel',hue='HandsetWebCapable',data=df)
<matplotlib.axes._subplots.AxesSubplot at 0x14522aa4248>
```

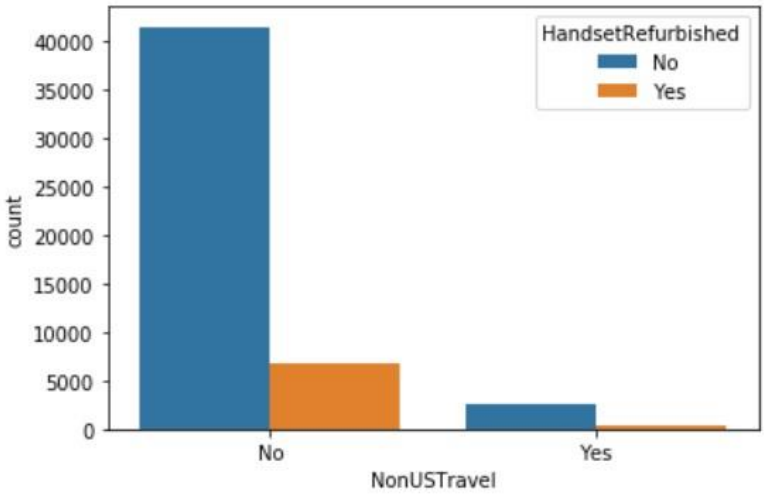


US Traveller handsets are webcapable.

- **Bivariate analysis of 'NonUSTravel' column with respect to the output column 'HandsetRefurbished':**

```
sns.countplot(x='NonUSTravel',hue='HandsetRefurbished',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x27026fa7d48>
```



US traveller used Refurbished Handset.

Checking of class imbalance of Churn column

```
y=0

n=0

for i in df["Churn"]:

   if i==1:

      y+=1

   else:

      n+=1
```

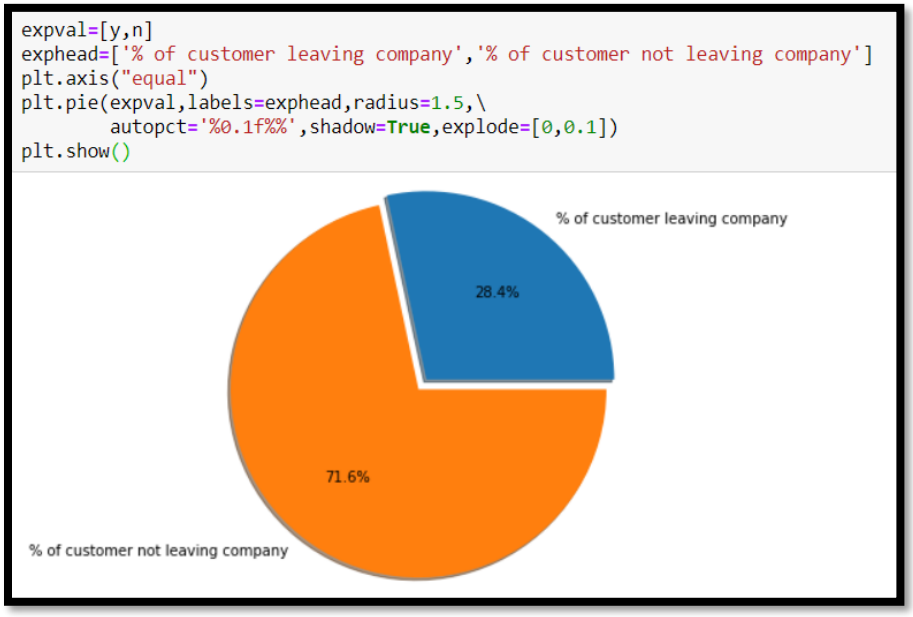
```
print("percentage of customer leaving company:",int((y/37483)*100),"%")

print("percentage of customer not leaving company:",int((n/37483)*100),"%")
```

OUTPUT:

```
percentage of customer leaving company: 28.4 %

percentage of customer not leaving company: 71.6 %
```

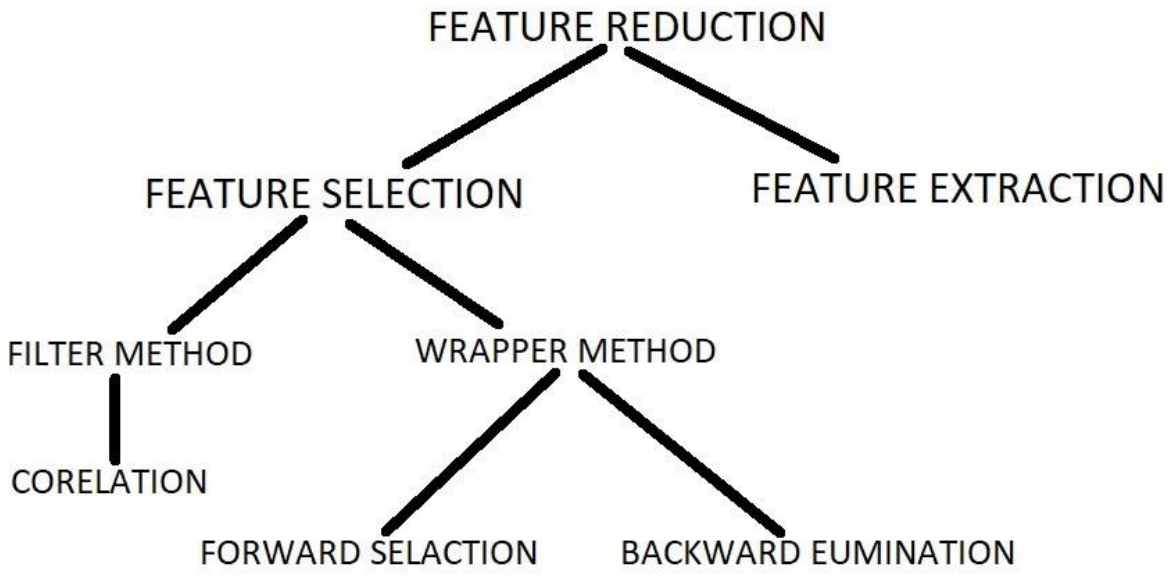
As it is a categorical variable, thus we will draw a pie plot to see that how much contributes to class 0 and class 1.

```
expval=[y,n]
exphead=['% of customer leaving company','% of customer not leaving company']
plt.axis("equal")
plt.pie(expval,labels=exphead,radius=1.5,\
        autopct='%0.1f%%',shadow=True,explode=[0,0.1])
plt.show()
```



FEATURES SELECTION

- Redundant Features: Strong relationship between two input features.



In Forward Selection, features are one by one added in input feature space and as a result of which different subsets are formed. The subset having high performance in chosen at last.

In backward selection process, features are one by one eliminated from input feature sapce and subset with high performance is selectedas same as the forward selection at last.

Here, in the underlined screenshot, correlation is implemented via f_classif and chi2 as follows:

CODE:

```
#Feature Selection 2
feature=[]
x=df.drop("Churn",axis=1)
y=df['Churn']
bestfeatures=SelectKBest(score_func=f_classif,k=12)
fit=bestfeatures.fit_transform(x,y)
cols=x.columns.values[bestfeatures.get_support()]
feature.append(cols)
print(feature)
```

OUTPUT:

```
[array(['TotalRecurringCharge', 'PeakCallsInOut', 'MonthsInService',
       'CurrentEquipmentDays', 'MonthlyMinutes', 'OverageMinutes',
       'PercChangeMinutes', 'UnansweredCalls', 'ReceivedCalls',
       'OutboundCalls', 'InboundCalls', 'OffPeakCallsInOut',
       'CallWaitingCalls', 'ActiveSubs', 'HandsetWebCapable',
       'BuysViaMailOrder', 'RespondsToMailOffers', 'UniqueSubs',
       'HandsetModels', 'CustomerCareCalls'], dtype=object)]
```

MODEL BUILDING

Before Model Building we drop some column. That's are 'Occupation','PrizmCode','CreditRating','HandsetPrice','ServiceArea','CustomerID','AgeHH1','AgeHH2','NonUSTravel','OwnsComputer','NewCellphoneUser','OwnsMotorcycle','AdjustmentsToCreditRating','HandsetRefurbished','Homeownership','ReferralsMadeBySubscriber'.

CODE:

```
df=df.drop(['Occupation','PrizmCode','CreditRating','HandsetPrice','ServiceArea','CustomerID','AgeHH1','AgeHH2','HandsetRefurbished','Homeownership','ReferralsMadeBySubscriber','NonUSTravel','OwnsComputer','NewCellphoneUser','OwnsMotorcycle','AdjustmentsToCreditRating'],axis=1)
```

Logistic Regression:

Logistic regression is a fundamental classification technique. It belongs to the group of linear classifiers and is somewhat similar to polynomial and linear regression. Logistic regression is fast and relatively uncomplicated, and it's convenient for you to interpret the results. Although it's essentially a method for binary classification, it can also be applied to multiclass problems.

CODE:

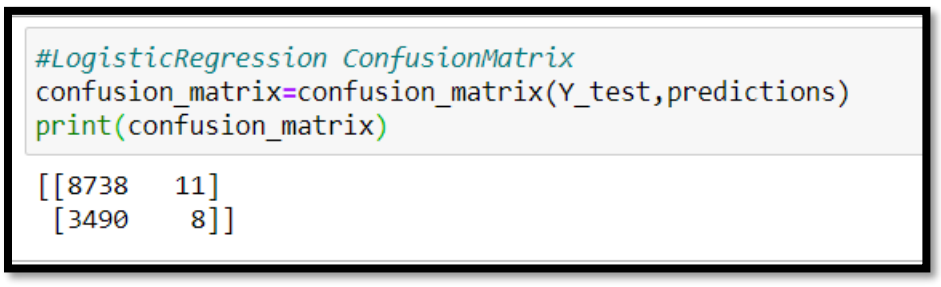
```
from sklearn.model_selection import train_test_split
for col in feature:
   X=df[col]
Y=df['Churn']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test,Y_train, Y_test=train_test_split(X,Y,test_size=0.3,random_state=1)
from sklearn.linear_model import LogisticRegression
logmodel=LogisticRegression()
logmodel.fit(X_train,Y_train)
predictions=logmodel.predict(X_test)
print(predictions)
from sklearn.metrics import accuracy_score
accuracy_score(Y_test,predictions)
```

OUTPUT:

```
[0. 0. 0. ... 0. 0. 1.]


0.7240956969053646
```

CONFUSION MATRIX:



```
#LogisticRegression ConfusionMatrix
confusion_matrix=confusion_matrix(Y_test,predictions)
print(confusion_matrix)
```

```
[[8738   11]
 [3490    8]]
```

MODEL TYPE	ACCURACY	RECALL	SPECIFICITY	F1 SCORE
Logistic Regression	0.724	0.00287	0.0023	0.0045

Decision Tree:

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

CODE:

```
from sklearn.tree import DecisionTreeClassifier
for col in feature:
   X=df[col]
Y=df['Churn']
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=100
)
classifier_entropy=DecisionTreeClassifier(criterion='entropy',random_state=100,m
ax_depth=3)
#Create The Model
classifier_entropy.fit(X_train,Y_train)
y_pred=classifier_entropy.predict(X_test)
print("Accuracy is",accuracy_score(Y_test,y_pred)*100)
```

OUTPUT:

```
Accuracy is 72.61370131460765
```

CONFUSION MATRIX:

```
#DecisionTree confusion matrix
confusion_matrix=confusion_matrix(Y_test,y_pred)
print('confusion_matrix:\n',confusion_matrix)

confusion_matrix:
 [[8785   85]
 [3269  108]]
```

MODEL TYPE	ACCURACY	RECALL	SPECIFICITY	F1 SCORE
Decision Tree	0.726	0.032	0.032	0.0605

k-nearest neighbors algorithm:

In pattern recognition, the k-nearest neighbors algorithm is a non-parametric method proposed by Thomas Cover used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space.

CODE:

```
for col in feature:
    X=df[col]
Y=df['Churn']
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=42)
classifier=KNeighborsClassifier(n_neighbors=19,metric='euclidean')
classifier.fit(X_train,Y_train)
```

```
y_pred=classifier.predict(X_test)
accuracy_score(Y_test, y_pred)
```

OUTPUT:

```
0.7072752510818976
```

CONFUSION MATRIX:

```
#KNN confusion matrix
confusion_matrix=confusion_matrix(Y_test,y_pred)
print('confusion_matrix:\n',confusion_matrix)
```

```
confusion_matrix:
 [[8376  373]
 [3212  286]]
```

MODEL TYPE	ACCURACY	RECALL	SPECIFICITY	F1 SCORE
KNN	0.71	0.0817	0.081	0.137

Naive Bayes:

CODE:

```
from sklearn.naive_bayes import GaussianNB
for col in feature:
    X=df[col]
Y=df['Churn']
```

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=100
)

model=GaussianNB()

model.fit(X_train,Y_train)

predictions=model.predict(X_test)

accuracy_score(Y_test,predictions)
```

OUTPUT :

```
0.680656487303013
```

CONFUSION MATRIX:

```
#NaiveBayes confusion matrix
confusion_matrix=confusion_matrix(Y_test,predictions)
print('confusion_matrix:\n',confusion_matrix)

confusion_matrix:
 [[7668 1202]
 [2709  668]]
```

MODEL TYPE	ACCURACY	RECALL	SPECIFICITY	F1 SCORE
Naïve Bayes	0.681	0.197	0.198	0.254

RANDOM FOREST:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

CODE:

```
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier
#for col in feature:
X=df.drop('Churn',axis=1)
Y=df['Churn']
num_trees=100
max_features=12
kfold=model_selection.KFold(n_splits=10,random_state=7)
model=RandomForestClassifier(n_estimators=num_trees,
max_features=max_features)
results=model_selection.cross_val_score(model,X,Y,cv=kfold)
print(results.mean())
```

OUTPUT:

```
0.7206971809715671
```

BASE MODEL:

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a test set X_test, y_test. Note that the word

“experiment” is not intended to denote academic use only, because even in commercial settings machine learning usually starts out experimentally.

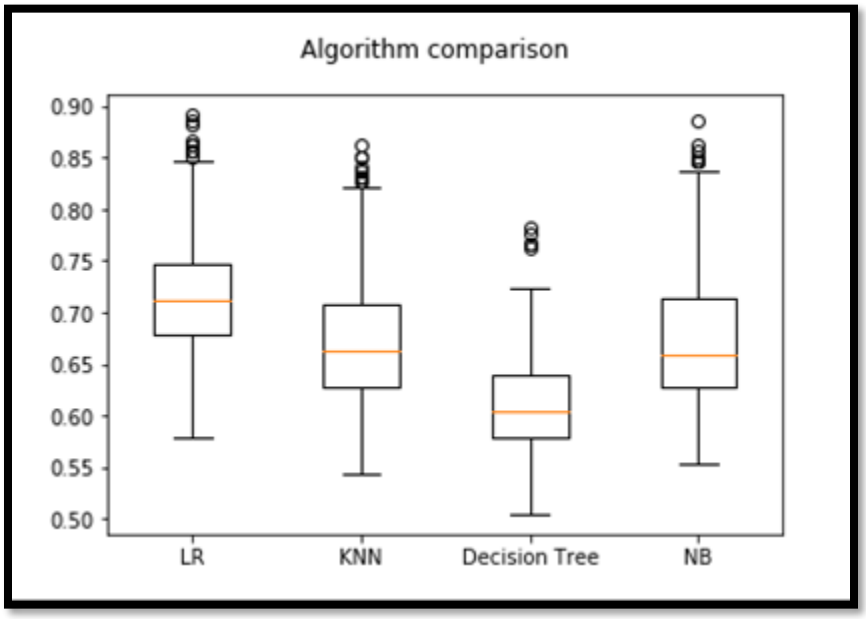
CODE:

```
import pandas
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

for col in feature:
   X=df[col]
Y=df['Churn']

seed=7
models=[]
models.append(('LR',LogisticRegression()))
models.append(('KNN',KNeighborsClassifier()))
models.append(('Decision Tree',DecisionTreeClassifier()))
models.append(('NB',GaussianNB()))
results=[]
names=[]
scoring='accuracy'
for name,model in models:
   kfold=model_selection.KFold(n_splits=int(np.sqrt(len(df))),random_state=seed)

cv_results=model_selection.cross_val_score(model,X,Y,cv=kfold,scoring=scoring)
```

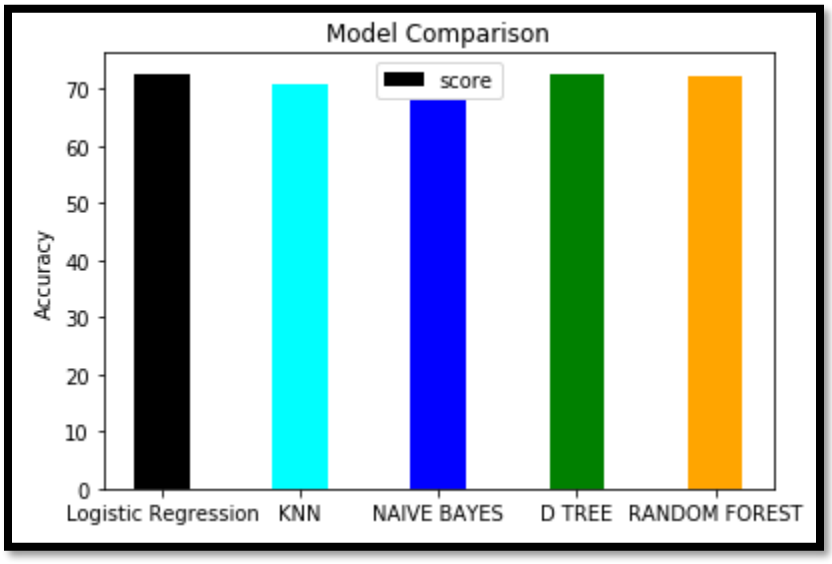
```
    results.append(cv_results)
    names.append(name)
    msg="%s: %f" % (name,cv_results.mean())
    print(msg)

fig=plt.figure()
fig.suptitle('Algorithm comparison')
ax=fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

OUTPUT:

```
LR: 0.716848
KNN: 0.672274
Decision Tree: 0.612086
NB: 0.675703
```



MODEL COMPARISON:



MODEL TYPE	ACCURACY	RECALL	SPECIFICITY	F1 SCORE	PRECISION
Decision Tree	0.726	0.032	0.032	0.0605	0.560
Logistic Regression	0.724	0.00287	0.0023	0.0045	0.579
Naïve Bayes	0.681	0.197	0.198	0.254	0.357
KNN	0.71	0.0817	0.081	0.137	0.434

CONCLUSION

We have tried Logistic Regression, Decision Tree, Naïve Bayes, KNN, Random Forest. We have found more accuracy in the Decision Tree model using these features-

(‘TotalRecurringCharge', 'CurrentEquipmentDays', 'MonthlyMinutes', 'PercChangeMinutes', 'UnansweredCalls', 'ReceivedCalls','OutboundCalls', 'InboundCalls', 'OffPeakCallsInOut','HandsetWebCapable', 'UniqueSubs', 'CustomerCareCalls’).

So we proposed that to use Decision tree model for this project.

FUTURE SCOPE

Inclusion of more data samples to third data source might increase confidence in quantitative results. Though the problem of customer churn was addressed by many researchers in numerous ways, still there is no standard model which addresses the issues of global telecom service providers accurately. There is lot of scope for development of such a model, which could take the above-mentioned factors and many more into consideration.