P. H. D

Churn Classification

By

Karthik Reddy Mathuru

1404

Batch-31

Problem Statement:

To predict the telecom customers who are likely to exit the contract and also to generate patterns of Churn and non-churn to assist the management to take appropriate decisions to limit churn.

Most telecom companies suffer from voluntary churn. Churn rate has strong impact on the life time value of the customer because it affects the length of service and the future revenue of the company. It is estimated that 75 percent of the 17 to 20 million subscribers signing up with a new wireless carrier every year are coming from another wireless provider, which means they are churners.

State of the art:

Telecom companies are battling to attract each other's customers while retaining their own. Thus, Customer churn reduction is the central concern of most telecom companies as switching costs to the customer are low and acquisition cost to the company is high. Churn reduces profitability as it means potential loss of future revenue and also losing the invested costs of acquisition.

Ways to Reduce Customer Churn:

- Consistently Exceed Customers' Expectations

  The most fundamental way to decrease your churn rate is by keeping your customers happy. While you definitely want to avoid letting them down, you have to look for areas to go over and above your customer's expectations and delight them. Failing to deliver on a promise is one of the fastest ways to lose a customer, and many companies say that dissatisfaction and unmet expectations are among the top reasons for client churn.

- Provide Awesome Customer Service

  This one should go without saying, but if you've ever spent half an hour listening to hold muzak waiting for a disinterested, incompetent customer service rep to "assist you," you'll know that some companies simply don't put enough effort into customer service.

- **Create Switching Costs:**

  Switching costs are any cost that a customer incurs by trading one product or service for another. Higher switching costs naturally reduce churn by reducing the likelihood that a customer will switch to a substitute product instead of returning to your brand.

Methods:

Logistic Regression:

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).

- We use ROC (Receiver Operating Characteristics) curve to set a threshold value which minimizes the false positive rate and maximizes the true positive rate.

Decision Trees:

Decision Tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets using greedy approach.

- Gini Index – (Classification and Regression Trees)
- Entropy – (C 5.0)

The above gives the impurity of the node. By these values we choose the node.

Random Forests:

Random 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 (classification) or mean prediction (regression) of the individual trees.

- It performs bagging technique (divides dataset into samples with replacement and build a decision tree for each sample)
- Random forest takes random number of features while performing bagging.
- It reduces the variance, there is less chance of overfitting.

Xg- Boost:

XGBoost is an open-source software library which provides the gradient boosting framework for C++, Java, Python, R and Julia. It works on Linux, Windows and macOS. From the project description, it aims to provide a "Scalable, Portable and Distributed Gradient Boosting (GBM, GBRT, GBDT) Library".

- XGBoost is extreme gradient boosting technique
- It is a boosting technique (builds a decision tree and add more weight to wrongly classified sample, again built a tree on it and so on till n trees)
- It reduces both bias and variance.

SVM:

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. It finds a max margin hyper plane two separate classes.

- In SVM, we have many kernel – radial, polynomial, linear etc.
- It performs quadratic optimization and kernel trick
- It also handles dual problem.
- It can handle class imbalance.

Stacking:

Stacking (also called meta ensembling) is a model ensembling technique used to combine information from multiple predictive models to generate a new model. Often times the stacked model (also called 2nd-level model) will outperform each of the individual models due its smoothing nature and ability to highlight each base model where it performs best and discredit each base model where it performs poorly.

MLP:

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable. Multilayer perceptron's are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer

Data:

The dataset depicts the details of the telecom customer, like, services that each customer has signed for, customer account information, and their demographic info etc.

- Attributes – 25 (Numerical – 3, Categorical – 20, Date – 2)
- Records – 5925

Attribute Information:

Demographics Data:

- HouseholdID: Each Household id
- Country: Country (For this attribute, missing values are denoted as "?")
- State: State (For this attribute, missing values are denoted as "?")
- Retired: Whether retired
- HasPartner: Demographic information - whether the customer has partner (1-Yes; 2-No)
- HasDependents: Demographic information - whether the customer has dependents (1-Yes; 2-No)
- Education: Education qualification
- Gender: Demographic information – gender

Account Information:

- CustomerID: CustomerID
- Base Charges: Customer account information (Charges for Base plan)
- DOC: Date of data collection
- Total Charges: Customer account information (Total). (For this attribute, missing values are denoted as "MISSINGVAL" also)

- DOE: Date of entry as customer
- Electronic Billing: Customer account information - whether electronic billing
- Contract Type: Contract type (For this attribute, missing values are denoted as “NA”)
- Payment Method: payment method

Data of ServicedOptedFor:

- CustomerID: CustomerID
- TypeOfService: Service signed for
- SeviceDetails:

Churn Data:

- CustomerID: Customer ID
- Churn: Whether the customer churns (Target)

Pre-processing:

- The datasets have the details of the customer - services that each customer has signed for, customer account information, and their demographic info.
- Data is aggregated from four datasets. Customer ID is the primary key
- Data in services have long format, it is made to wide format using pivot table(python).
- There are 42 missing values in total. We have two columns which have no variance (Country, State) which has 13 missing values, these columns are removed.
- Missing values are very less. So, omitted the records with NA.

Feature Engineering:

- DOC (Date of Collection)
- DOE (Date of Entry)

From above two columns we get the age (Life time) of the customer by difference between them.

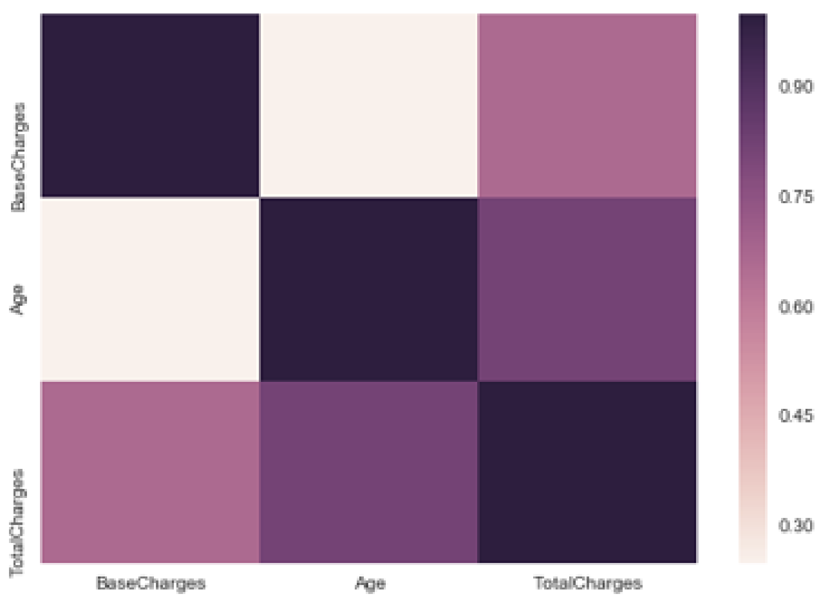
AGE = DOC – DOE

Standardization:

Age, Base Charges, Total Charges are the numeric attributes in data with different scale. We use standardization techniques to make them in same scale.

Correlation:

By the plot, we know that there is a high positive correlation between Age (attribute is derived from DOC and DOE) and Total charges.

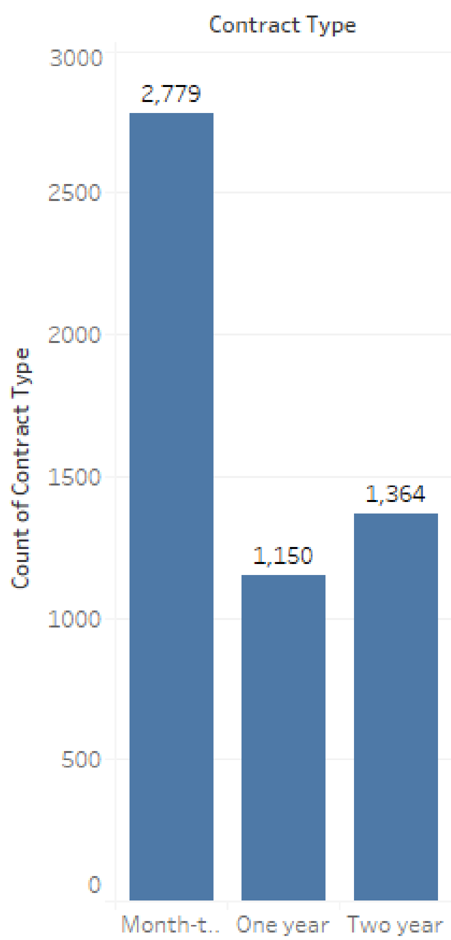


Analysis:

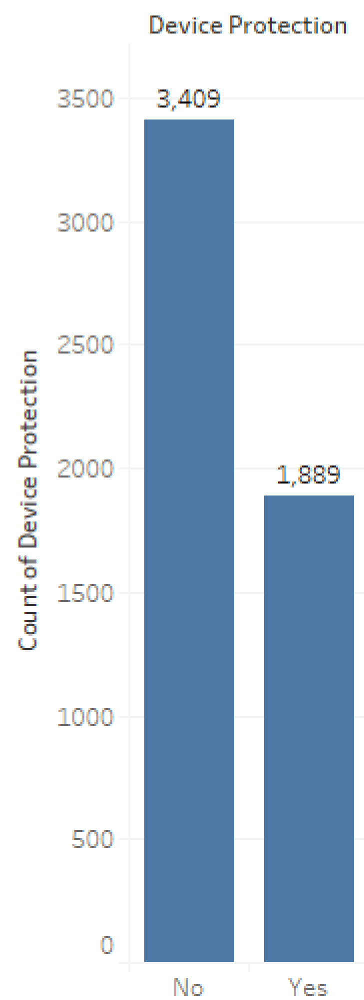
➢ Uni – Variate:

Contract Type	Count of Contract Type
One year	1,150
Two years	1,364
Month-to-month	2,779

Device Protection	Count of Device Protection
Yes	1,889
No	3,409

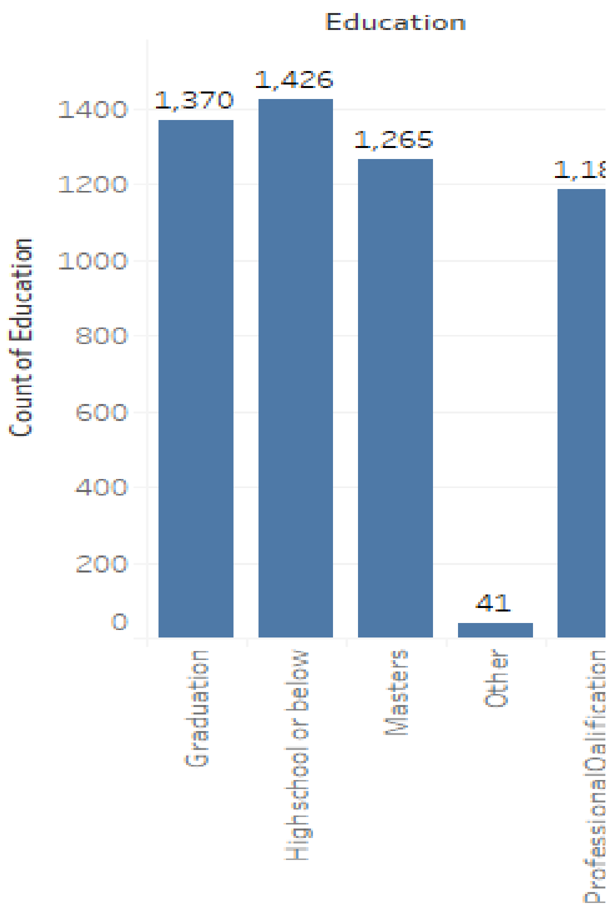


Count of Contract Type for each Contract Type. The marks are labeled by count of Contract Type. The view is filtered on Contract Type, which excludes NA.



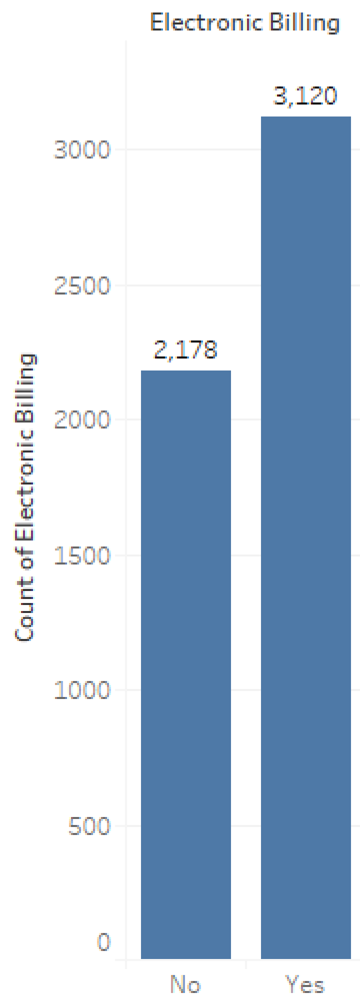
Count of Device Protection for each Device Protection. The marks are labeled by count of Device Protection.

Education	Count of Education
Other	41
Professional Qualification	1,186
Masters	1,265
Graduation	1,370
Highschool or below	1,426



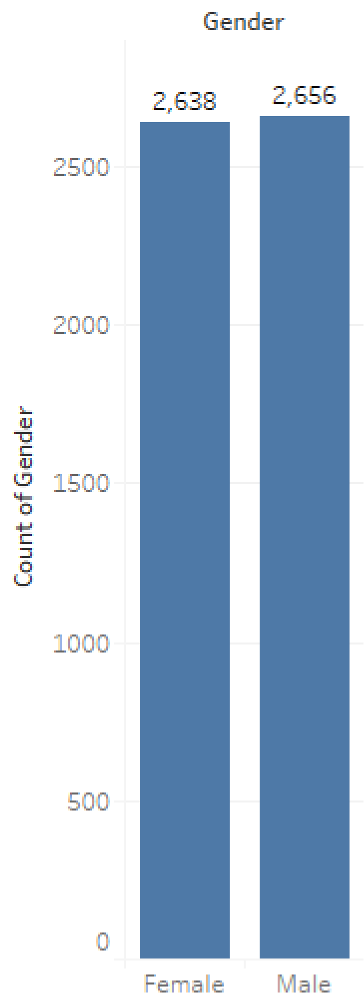
Count of Education for each Education. The marks are labeled by count of Education. The view is filtered on Education, which excludes Null.

Electronic Billing	Count of Electronic Billing
No	2,178
Yes	3,120



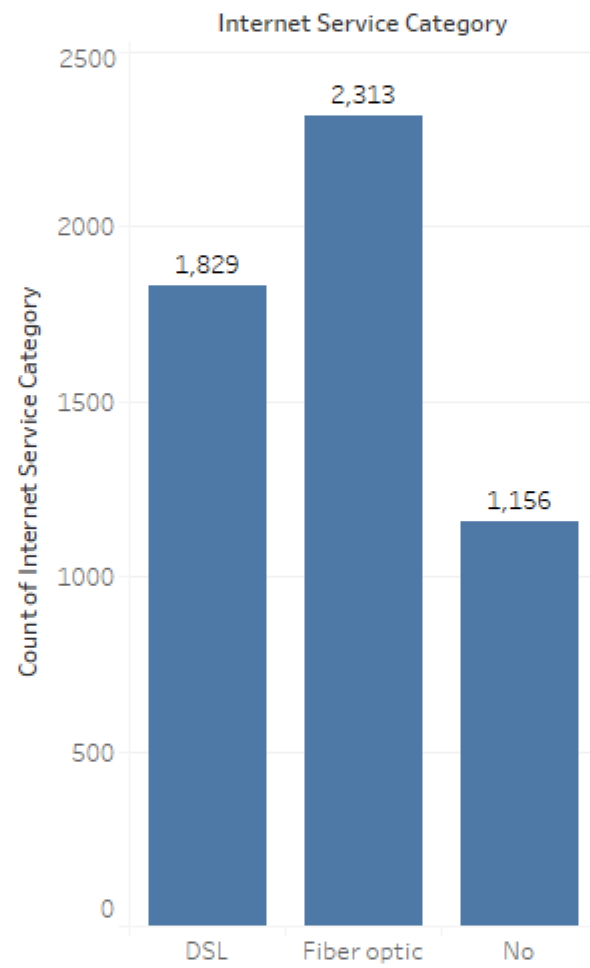
Count of Electronic Billing for each Electronic Billing. The marks are labeled by count of Electronic Billing.

Gender	Count of Gender
Female	2,638
Male	2,656



Count of Gender for each Gender. The marks are labeled by count of Gender. The view is filtered on Gender, which excludes Null.

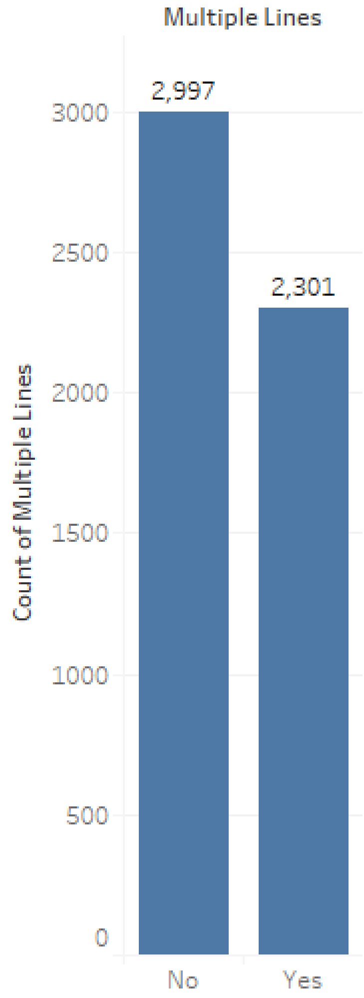
Internet Service Category	Count of Internet Service Category
No	1,156
DSL	1,829
Fiber optic	2,313



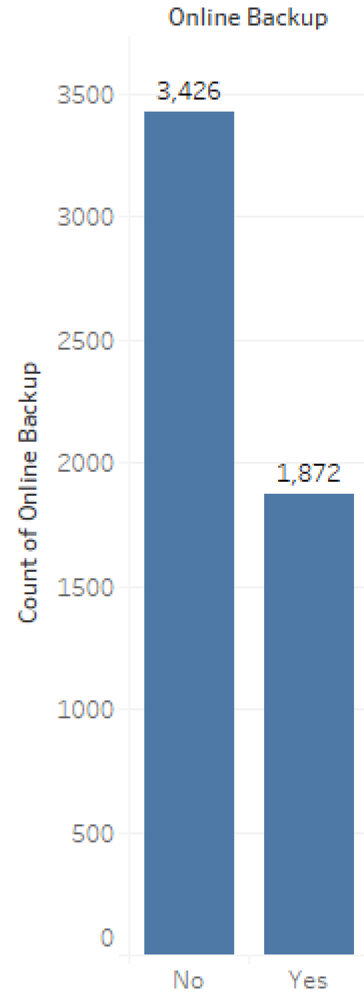
Count of Internet Service Category for each Internet Service Category. The marks are labeled by count of Internet Service Category.

Multiple Lines	Count of Multiple Lines
Yes	2,301
No	2,997

Online Backup	Count of Online Backup
Yes	1,872
No	3,426



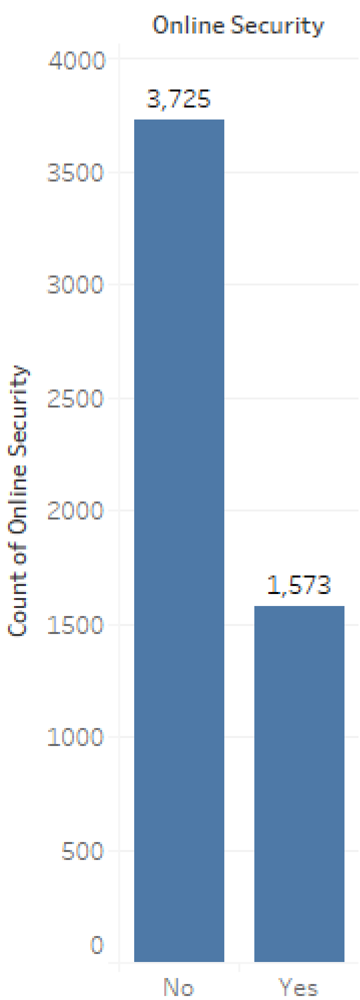
Count of Multiple Lines for each Multiple Lines. The marks are labeled by count of Multiple Lines.



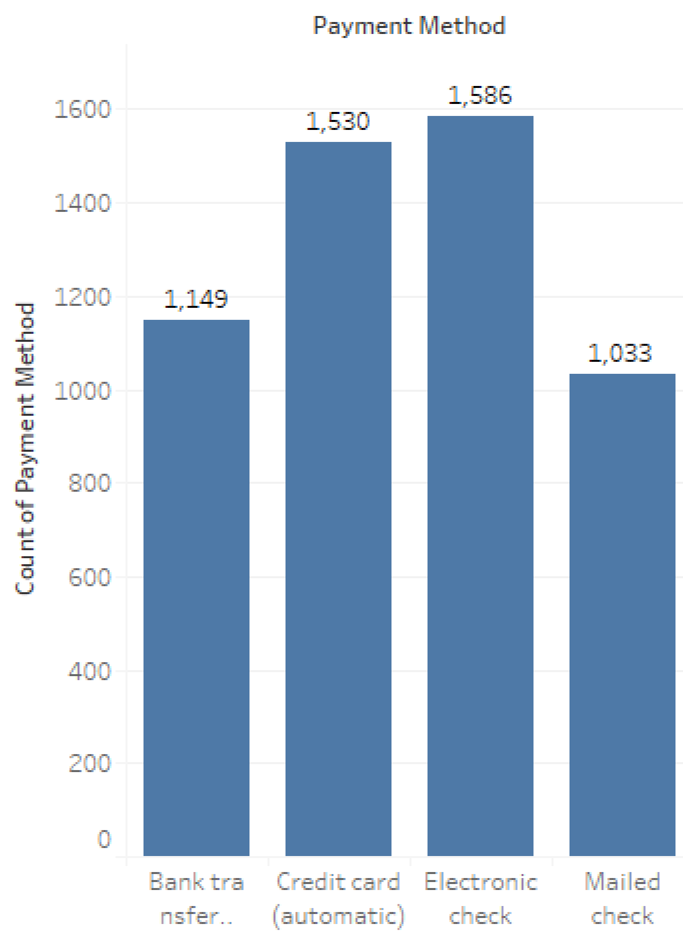
Count of Online Backup for each Online Backup. The marks are labeled by count of Online Backup.

Online Security	Count of Online Security
Yes	1,573
No	3,725

Payment Method	Count of Payment Method
Mailed check	1,033
Bank transfer (automatic)	1,149
Credit card (automatic)	1,530
Electronic check	1,586



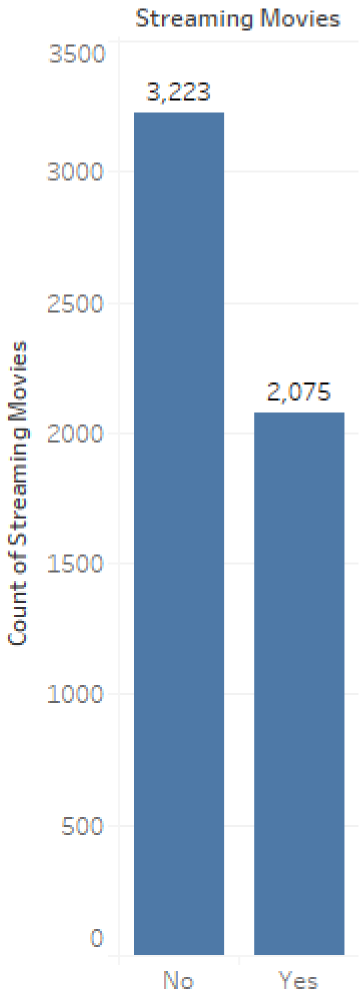
Count of Online Security for each Online Security. The marks are labeled by count of Online Security.



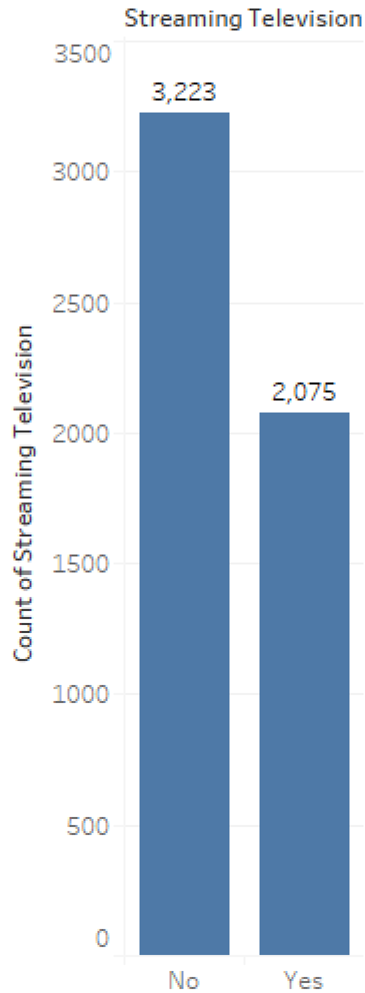
Count of Payment Method for each Payment Method. The marks are labeled by count of Payment Method.

Streaming Movies	Count of Streaming Movies
Yes	2,075
No	3,223

Streaming Television	Count of Streaming Television
Yes	2,075
No	3,223



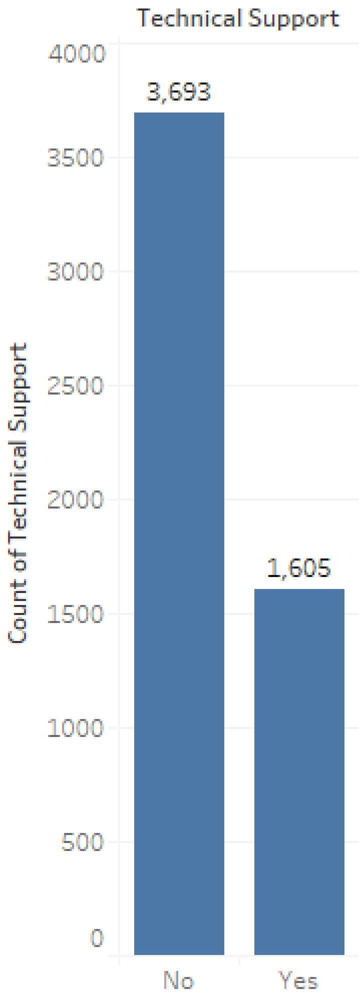
Count of Streaming Movies for each Streaming Movies. The marks are labeled by count of Streaming Movies.



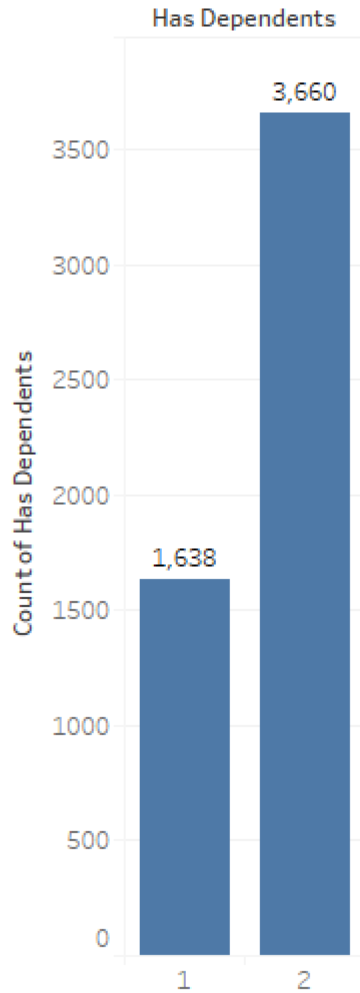
Count of Streaming Television for each Streaming Television. The marks are labeled by count of Streaming Television.

Technical Support	Count of Technical Support
Yes	1,605
No	3,693

Has Dependents	Count of Has Dependents
1	1,638
2	3,660



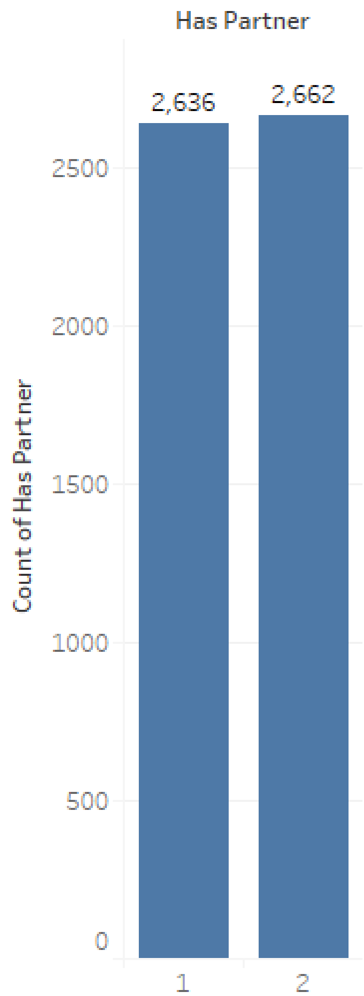
Count of Technical Support for each Technical Support. The marks are labeled by count of Technical Support.



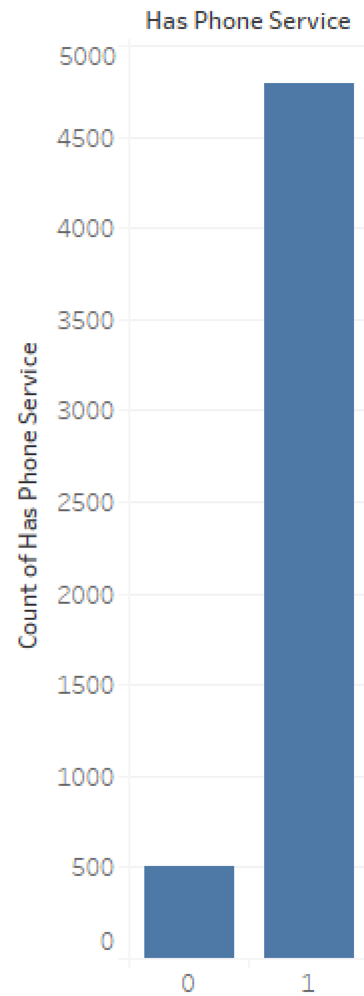
Count of Has Dependents for each Has Dependents. The marks are labeled by count of Has Dependents.

Has Partner	Count of Has Partner
1	2,636
2	2,662

Has Phone Service	Count of Has Phone Service
0	506
1	4,792

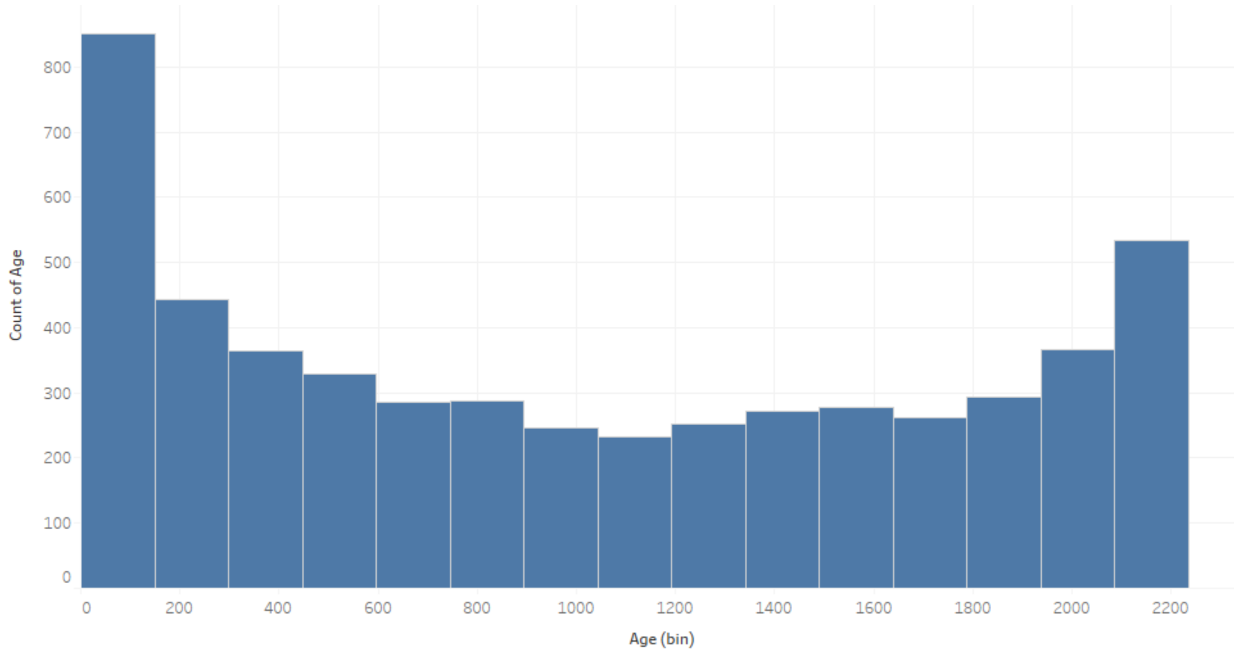


Count of Has Partner for each Has Partner. The marks are labeled by count of Has Partner.



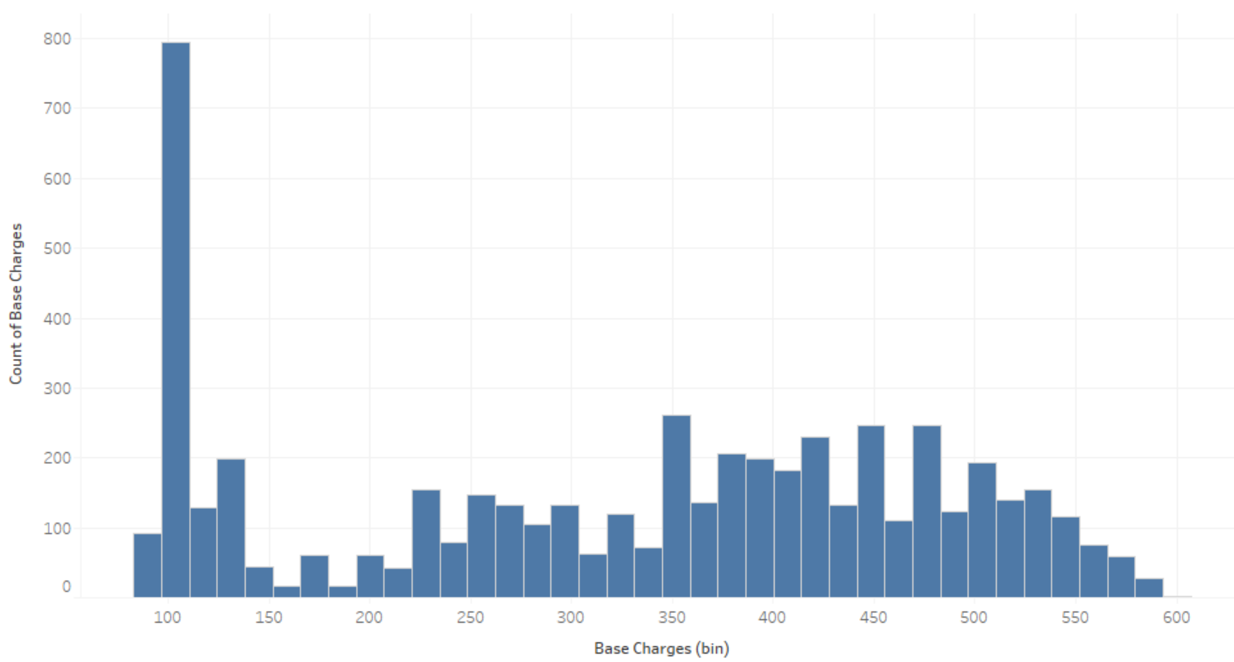
Count of Has Phone Service for each Has Phone Service.

Age:



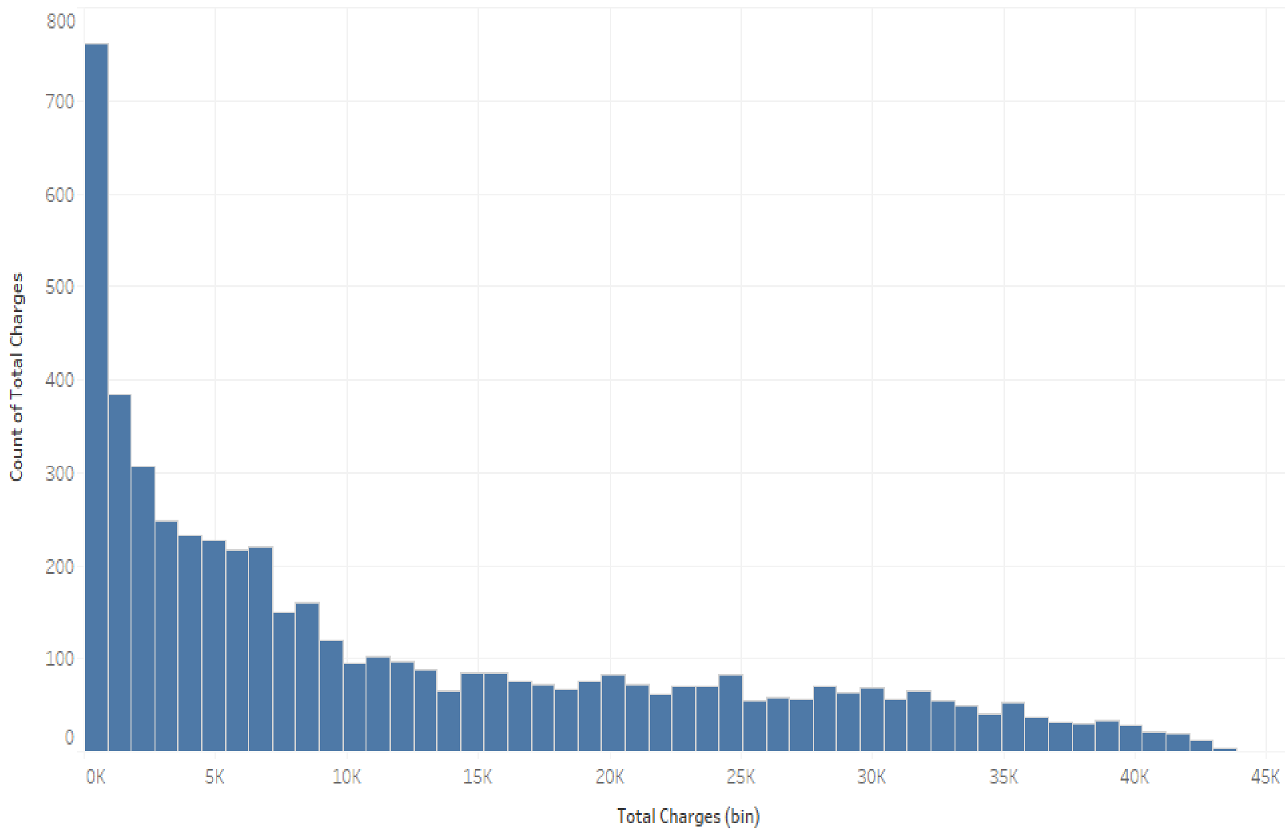
The trend of count of Age for Age (bin).

Base Charges:



The trend of count of Base Charges for Base Charges (bin).

Total Charges:



The trend of count of Total Charges for Total Charges (bin).

Result:

Error metric is Recall

- Recall = True Positive/ Total Actual Positive
- We should predict telecom customers who are likely to churn. Recall of class ‘Yes’ gives us customers who are likely to churn.

➢ Decision Tree:

○ Gini index

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.83	0.83	790
1	0.48	0.48	0.48	264
avg / total	0.74	0.74	0.74	1054

Accuracy 0.740037950664

○ Entropy:

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.83	0.83	790
1	0.49	0.50	0.49	264
avg / total	0.75	0.74	0.74	1054

Accuracy : 0.743833017078

➢ Random Forest:

Classifaction Report :

	precision	recall	f1-score	support
0	0.91	0.76	0.83	790
1	0.52	0.77	0.62	264
avg / total	0.81	0.76	0.78	1054

Accuracy : 0.76375711575

Xg Boost

Classifaction Report :

	precision	recall	f1-score	support
0	0.85	0.91	0.88	790
1	0.66	0.51	0.58	264
avg / total	0.80	0.81	0.80	1054

Accuracy : 0.812144212524

SVM

Classifaction Report :

	precision	recall	f1-score	support
0	0.89	0.82	0.85	790
1	0.56	0.70	0.63	264
avg / total	0.81	0.79	0.80	1054

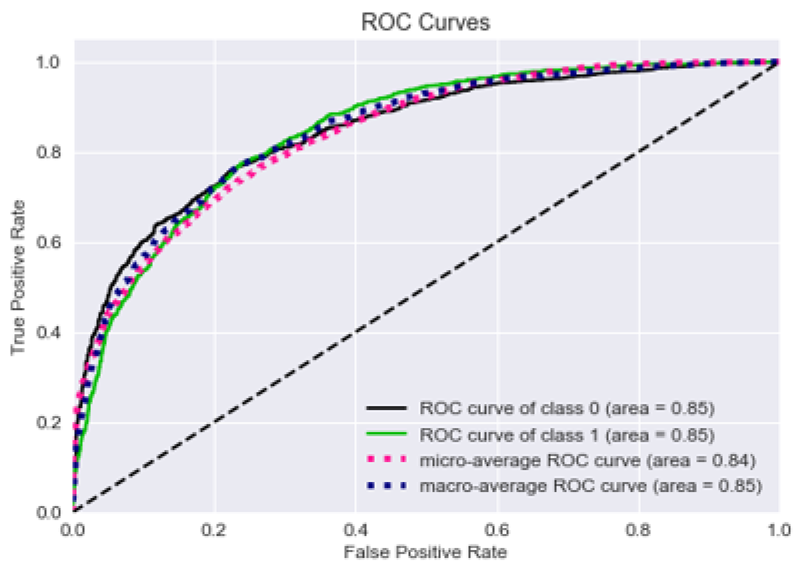
Accuracy : 0.789373814042

Logistic regression

Classifaction Report :

	precision	recall	f1-score	support
0	0.90	0.75	0.82	790
1	0.51	0.76	0.61	264
avg / total	0.80	0.75	0.77	1054

Accuracy : 0.753320683112



➢ Stacking:

```
Classifaction Report :

             precision    recall  f1-score   support

          0       0.88      0.84      0.86       790
          1       0.59      0.67      0.63       264

avg / total       0.81      0.80      0.80      1054


Accuracy :  0.799810246679
```

➢ MLP:

```
Classifaction Report :

             precision    recall  f1-score   support

          0       0.84      0.87      0.86       790
          1       0.57      0.52      0.54       264

avg / total       0.78      0.78      0.78      1054

Accuracy : 0.781783681214
```

Patterns:

Decision trees are used to extract the hidden patterns, decision trees give rules which are supported by confidence, lift. C5.0 is used to extract the patterns.

Appendices:

```
# In[1]:


# import modules

import pandas as pd

import numpy as np


# #### Loading the data


# In[2]:


# Train

train = pd.read_csv('TrainData\Train.csv')

accinfo = pd.read_csv('TrainData\Train_Accountinfo.csv',na_values=['NA','MISSINGVAL'])

demographics = pd.read_csv('TrainData\Train_Demographics.csv',na_values='?')

services = pd.read_csv('TrainData\Train_ServicesOptedFor.csv')



# In[3]:


# Test

test = pd.read_csv('TestData\Test.csv')

taccinfo = pd.read_csv('TestData\Test_Accountinfo.csv',na_values=['NA','MISSINGVAL'])
```

```
tdemographics = pd.read_csv('TestData\Test_Demographics.csv',na_values='?')
tservices = pd.read_csv('TestData\Test_ServicesOptedFor.csv')


# #### Dimensions

# In[4]:

# Shapes of the train data
print(train.shape)
print(accinfo.shape)
print(demographics.shape)
print(services.shape)


# In[5]:

# Shapes of test data
print(test.shape)
print(taccinfo.shape)
print(tdemographics.shape)
print(tservices.shape)


# From the data, we know that we have CustomerID as PrimaryKey

# In[6]:
```

```
# unique columns in datasets
print(len(train.CustomerID.unique()))
print(len(accinfo.CustomerID.unique()))
print(len(demographics.HouseholdID.unique()))
print(len(services.CustomerID.unique()))


# ###### Merging the data frames by CustomerID

# In[7]:

# Train
n1 = pd.merge(train,accinfo,on='CustomerID')


# In[8]:

# We have HouseholdID in demographics which is CustomerID so renaming the column.
demographics.columns = ['CustomerID', 'Country', 'State', 'Retired', 'HasPartner',
    'HasDependents', 'Education', 'Gender']


# In[9]:

new = pd.merge(n1,demographics,on='CustomerID')


# In[10]:
```

```
# Dimensions
new.shape



# In[11]:


# Test
tdemographics.columns = ['CustomerID', 'Country', 'State', 'Retired', 'HasPartner',
       'HasDependents', 'Education', 'Gender']



# In[12]:


tnew = pd.merge(taccinfo,tdemographics,on='CustomerID')



# In[13]:


tnew.shape



# ** Services ** have shape (15921, 3)

# In[14]:


# count - categories
services.SeviceDetails.value_counts()
```

```
# In[15]:


# Changing the attribute types to category
services.SeviceDetails = services.SeviceDetails.astype('category')



# In[16]:


# Test
tservices.SeviceDetails = tservices.SeviceDetails.astype('category')



# * Dataframe is in long format, we should make it into wide foramt

# In[17]:


# Label Encoding
services.SeviceDetails = services.SeviceDetails.cat.codes
services.SeviceDetails.value_counts()



# In[18]:


tservices.SeviceDetails = tservices.SeviceDetails.cat.codes
tservices.SeviceDetails.value_counts()
```

```
# In[19]:

# creating a multilevel index with values as serviceDetails
n = services.pivot_table(index="CustomerID", columns="TypeOfService",values = "SeviceDetails")


# In[20]:

n.head()


# In[21]:

ser = np.matrix(n)


# In[22]:

ser = pd.DataFrame(ser,columns=n.columns)
ser.head()


# In[23]:

tn = tservices.pivot_table(index="CustomerID", columns="TypeOfService",values = "SeviceDetails")
```

```
# In[24]:


tser = np.matrix(tn)



# In[25]:


tser = pd.DataFrame(tser,columns=tn.columns)
tser.head()



# In[26]:


# concatinating the data frames
final = pd.concat([new,ser],1) # train
tfinal = pd.concat([tnew,tser],1) # test



# In[27]:


final.head() # view



# In[28]:


tfinal.head()
```

```
# #### Missing Values


# In[29]:


print(final.isnull().sum())
print('\n Total:\t\t', final.isnull().any(1).sum())



# In[30]:


# Drop the columns with no variance and unique values
final = final.drop(['CustomerID','Country','State'],1)
tfinal = tfinal.drop(['CustomerID','Country','State'],1)



# In[31]:


print(final.isnull().sum())
print('\n Total:\t\t', final.isnull().any(1).sum())



# In[32]:


# Missing values are missing at random, so there is no pattern in it.
final[final.isnull().any(1)]



# In[33]:
```

```
final = final.dropna() # Drop missing values


# In[34]:


final.to_csv('f.csv',index=False)
final = pd.read_csv('f.csv')


# In[35]:


final.head()


# #### Type Casting

# In[36]:


num = [ 'BaseCharges',  'TotalCharges' ]
date = ['DOC', 'DOE']
cat = ['Churn','ElectronicBilling','ContractType', 'PaymentMethod',
      'Retired', 'HasPartner', 'HasDependents', 'Education',
       'Gender','DeviceProtection', 'HasPhoneService',
      'InternetServiceCategory', 'MultipleLines', 'OnlineBackup',
      'OnlineSecurity', 'StreamingMovies', 'StreamingTelevision',
      'TechnicalSupport']
```

```
# ##### Categorical

# In[37]:

# Changing attributes to category
for i in cat:
    final[i] = final[i].astype('category')



# In[38]:

# Label Encoding for categorical attributes
for i in cat:
    final[i] = final[i].cat.codes
for i in cat:
    final[i] = final[i].astype('category')



# In[39]:

# Test
for i in cat[1:]:
    tfinal[i] = tfinal[i].astype('category')



# In[40]:
```

```
for i in cat[1:]:
    tfinal[i] = tfinal[i].cat.codes
for i in cat[1:]:
    tfinal[i] = tfinal[i].astype('category')


# In[41]:


final.to_csv('fin.csv')# write data


# In[42]:


final.head()


# ### UniVariate Analysis:

# In[43]:


import matplotlib.pyplot as plt
get_ipython().magic('matplotlib inline')


# In[44]:


# Plots for categorical attributes
for i in cat:
```

```
    pd.crosstab(final[i],columns='count').plot(kind = 'bar')
    plt.savefig(i)


# ### Numerical

# In[45]:

for i in num:
    final[i] = final[i].astype('int')
# Base Charges plot
plt.figure()
final.BaseCharges.hist(grid = False)
plt.xlabel('BaseCharges')
plt.ylabel('frequency')
plt.savefig('base_hist.png')


# In[46]:

# Total Charges plot
plt.figure()
final.TotalCharges.hist()
plt.xlabel('TotalCharges')
plt.ylabel('frequency')
plt.savefig('tot_hist.png')
```

```
# In[47]:


# Box plots
for i in num:
    final.boxplot(i)
    plt.show()
    plt.savefig(i)



# ### Date


# In[48]:


import datetime



# In[49]:


# Date time format
final.DOC = pd.to_datetime(final.DOC)
tfinal.DOC = pd.to_datetime(tfinal.DOC)



# In[50]:


final.DOE = pd.to_datetime(final.DOE)
tfinal.DOE = pd.to_datetime(tfinal.DOE)
```

```
# #### Feature Engineering

# In[51]:

# Age is the lifetime of the customer
final['Age'] = final.DOC - final.DOE
tfinal['Age'] = tfinal.DOC - tfinal.DOE


# In[52]:

# Converting to int, models does not accept timedelta
final.Age = (final.Age / np.timedelta64(1, 'D')).astype(int)
tfinal.Age = (tfinal.Age / np.timedelta64(1, 'D')).astype(int)


# In[53]:

tfinal.head()


# ** correlation **

# In[54]:

import seaborn as sns
sns.heatmap(final[['BaseCharges','Age','TotalCharges']].corr())
```

```
# #### Standardization

# In[55]:

# Numerical variables
num_var = final[['BaseCharges','Age' ]]
num_var.shape
tnum_var = tfinal[['BaseCharges', 'Age' ]]
tnum_var.shape


# In[56]:

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()


# In[57]:

num_scaled = pd.DataFrame(scaler.fit_transform(num_var),columns=num_var.columns)
num_scaled.head()


# In[58]:

tnum_var.isnull().sum()
```

```
# In[59]:

tnum_scaled = pd.DataFrame(scaler.fit_transform(tnum_var),columns=tnum_var.columns)


# In[60]:

tnum_scaled.shape


# In[61]:

# Dropping all the columns as we already have them
final = final.drop(['TotalCharges','BaseCharges','Age'],1)


# In[62]:

tfinal = tfinal.drop(['TotalCharges','BaseCharges','Age'],1)


# In[63]:

final = pd.concat([num_scaled,final],1) # concatenation
```

```
# In[64]:

final.head()

# In[65]:

tfinal = pd.concat([tnum_scaled,tfinal],1)

# In[66]:

tfinal.head()

# In[67]:

final.columns # Columns

# In[68]:

final.shape #dimensions

# In[69]:

final.to_csv('tab.csv')
```

```
# In[70]:

te_x  = tfinal.drop(['DOC','DOE'],1) # derived new column from DOC and DOE, so we drop them

# In[71]:

te_x.head()

# In[72]:

y = final.Churn # Target
x = final.drop(['Churn','DOC','DOE'],1)

# In[73]:

x.head()

# In[74]:

# Percentage of target variable
print('0 :',(y.value_counts()[0]/y.value_counts().sum())*100)
print('1 :',(y.value_counts()[1]/y.value_counts().sum())*100)
```

Train Test Split

In[75]:

```
from sklearn.model_selection import train_test_split
```

In[76]:

```
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size = 0.2,stratify = y)
```

Model Building:

Decision tree :

#

Decision Tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets using greedy approach.

In[77]:

```
from sklearn.tree import DecisionTreeClassifier

from sklearn import metrics
```

```
# In[78]:


# gini index - CART(Classification and Refression Tree)
tree = DecisionTreeClassifier(class_weight='balanced')



# In[79]:


tree.fit(xtrain,ytrain) # Train the model



# In[80]:


tree.score(xtrain,ytrain) #Accuracy



# In[81]:


print('\n Classification Report:\n\n',metrics.classification_report(ytest,tree.predict(xtest)))
print('Accuracy',metrics.accuracy_score(ytest,tree.predict(xtest)))



# In[82]:


# Entropy - C5.0
entropy = DecisionTreeClassifier(criterion='entropy',class_weight='balanced') # entropy
```

In[83]:

```
entropy.fit(xtrain,ytrain) #Train the model
```

In[84]:

```
entropy.score(xtest,ytest) # accuracy
```

In[85]:

```
print('\n Classification Report:\n\n',metrics.classification_report(ytest,entropy.predict(xtest)))

print('Accuracy :',metrics.accuracy_score(ytest,entropy.predict(xtest)))
```

Random Forest:

#

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

In[86]:

```
from sklearn.ensemble import RandomForestClassifier
```

In[87]:

```
r = RandomForestClassifier(n_estimators=100,n_jobs=-1,oob_score=True,
                        class_weight='balanced',max_depth=14,random_state = 20,max_leaf_nodes = 29,min_samples_leaf = 6)



# In[88]:


rf_exp = r.fit(xtrain,ytrain)



# In[89]:


r.score(xtrain,ytrain)



# In[90]:


print('\n Classifaction Report : \n\n',metrics.classification_report(ytest,r.predict(xtest)))
print('Accuracy :',metrics.accuracy_score(ytest,r.predict(xtest)))



# In[91]:


# Tuning
from sklearn.grid_search import RandomizedSearchCV
```

```
# In[92]:

param_grid = dict(max_depth=list(range(1, 20)),min_samples_leaf = list(range(1,12)),
          max_leaf_nodes=list(range(2,30)))


# In[93]:

grid = RandomizedSearchCV(r, param_grid, cv=10, scoring='accuracy',n_jobs= -1,n_iter=20)


# In[94]:

grid.fit(xtrain,ytrain)


# In[95]:

grid.best_params_


# ** Random Forest : Feature Importance **

# In[96]:

sorted(list(zip(r.feature_importances_,xtrain.columns)))
```

```
# In[97]:


### Drop the columns which are not important
rxtrain = xtrain.drop(['Retired', 'HasPartner','Gender',
      'HasPhoneService','DeviceProtection','MultipleLines',
      'HasDependents','StreamingTelevision'],1)
rxtest = xtest.drop(['Retired', 'HasPartner','Gender',
      'HasPhoneService','DeviceProtection','MultipleLines',
      'HasDependents','StreamingTelevision'],1)


# In[98]:


rxtrain.columns


# In[99]:


rte_x = te_x.drop(['Retired', 'HasPartner','Gender',
      'HasPhoneService','DeviceProtection','MultipleLines',
      'HasDependents','StreamingTelevision'],1)# Test


# In[100]:


rte_x.columns
```

```
# ** Xg Boost: **

# In[101]:

from xgboost import XGBClassifier


# In[102]:

xg = XGBClassifier(n_jobs= -1,n_estimators= 500,booster='gblinear',max_delta_step= 2)


# In[103]:

xg.fit(np.matrix(xtrain),ytrain)


# In[104]:

xg.score(np.matrix(xtrain),ytrain) # Predictions


# In[105]:

print('\n Classifaction Report : \n\n',metrics.classification_report(ytest,xg.predict(np.matrix(xtest)))) # Predictions
print('Accuracy :',metrics.accuracy_score(ytest,xg.predict(np.matrix(xtest))))
```

** SVM (Support Vector Machine):**

In[106]:

```
from sklearn.svm import LinearSVC
s = LinearSVC(class_weight='balanced',C = 10)
```

In[107]:

```
s.fit(xtrain,ytrain)
```

In[108]:

```
print('\n Classifaction Report : \n\n',metrics.classification_report(ytest,s.predict(xtest))) # Predictions
print('Accuracy :',metrics.accuracy_score(ytest,s.predict(xtest)))
```

**Logistic Regression : **

In[109]:

```
from sklearn.linear_model import LogisticRegressionCV
logreg = LogisticRegressionCV(class_weight='balanced',random_state=20,cv =
10,scoring='accuracy',Cs=100)
```

```
# In[110]:


logreg.fit(xtrain,ytrain)



# In[111]:


import scikitplot as skplt
skplt.metrics.plot_roc_curve(ytrain,logreg.predict_proba(xtrain))



# In[112]:


print('\n Classifaction Report : \n\n',metrics.classification_report(ytest,logreg.predict(xtest)))
print('Accuracy :',metrics.accuracy_score(ytest,logreg.predict(xtest)))



# #### Stacking

# In[113]:


# Train Predictions
sx = s.predict(np.matrix(xtrain))
sl = logreg.predict(xtrain)
sr = r.predict(xtrain)
```

```
# In[114]:


# val Predictions
stx = s.predict(np.matrix(xtest))
strr = r.predict(xtest)
stl = logreg.predict(xtest)



# In[115]:


# test Predictions
stxt = s.predict(np.matrix(te_x))
strrt = r.predict(te_x)
stlt = logreg.predict(te_x)



# In[116]:


# Train
stack_train = pd.concat([pd.DataFrame(sx),pd.DataFrame(sr),pd.DataFrame(sl)],1)



# In[117]:


# val
stack_test = pd.concat([pd.DataFrame(stx),pd.DataFrame(strr),pd.DataFrame(stl)],1)
```

```
# In[118]:


# test
stack_t = pd.concat([pd.DataFrame(stxt),pd.DataFrame(strrt),pd.DataFrame(stlt)],1)



# In[119]:


stack_train.shape



# In[120]:


stack = RandomForestClassifier(max_depth=13)



# In[121]:


stack.fit(stack_train,ytrain)



# In[122]:


print('\n Classifaction Report : \n\n',metrics.classification_report(ytest,stack.predict(stack_test)))
print('\n Accuracy : ',metrics.accuracy_score(ytest,stack.predict(stack_test)))



# #### MLP
```

```
# In[123]:


from sklearn.neural_network import MLPClassifier


# In[124]:


m = MLPClassifier(learning_rate='adaptive',hidden_layer_sizes=(75,75))


# In[125]:


m.fit(xtrain,ytrain)


# In[126]:


# Classification report and accuracy
print('\n Classifaction Report : \n\n',metrics.classification_report(ytest,m.predict(np.matrix(xtest))))
print('Accuracy :',metrics.accuracy_score(ytest,m.predict(np.matrix(xtest))))


# ** Predictions On Test **

# In[127]:


logreg.fit(x,y)
```

```
# In[128]:

p = logreg.predict(te_x)


# In[129]:

labels = ['No','Yes']
churn=[]
for i in p:
    churn.append(labels[i])


# In[130]:

churn = pd.DataFrame(churn,columns=['Churn'])


# In[131]:

preds = pd.concat([test,churn],1) # CustomerID and Churn


# In[132]:

preds.to_csv('predictions.csv',index=False) # write to predictions.csv
```

** Patterns **

In[133]:

```
exp = tree.fit(xtrain,ytrain)  # gini
exp1 = entropy.fit(xtrain,ytrain) # Entropy
```

In[134]:

```
from sklearn.tree import export_graphviz
export_graphviz(exp,'patern.dot',class_names=['No','Yes'],feature_names=xtrain.columns)
export_graphviz(exp1,'entropy.dot',class_names=['No','Yes'],feature_names=xtrain.columns)
```