Customer Churn Risk Prediction

- Introduction
- Data Set
- Projejct Objective
- Project Motivation
- Software Needed
- Loading the Data and Importing Libraries
- Gathering Data
- Exploratory Data Analysis
- Modelling
- Conclusion

Introduction

Your client is a major mobile telecommunication network provider in the US. They are experiencing issues related to customer churn or attrition i.e. customers cancelling their accounts and possibly switching to other competitor services.

Data Set

The data available includes customers' demographic profile, their plan features and usage history along with an indicator whether they actually churned or not. This is provided in the dataset below:

- Churn History Dataset.csv

Use this historical dataset to build/train the churn models. Then evaluate the prediction accuracy of the models on the test dataset below:

- Churn Test Dataset.csv

Project Objective

client is interested in understanding the leading indicators of churn and identifying potential churners ahead of time. This will enable them to take pre-emptive action such as offering better plans and discounts to potential churners and encouraging them to continue their service

Software Needed

Software: Python and Jupyter Notebook

The following packages (libraries) need to be installed:

1. pandas
2. NumPy
3. scikit Learn
4. Logistic Regression
5. Random Forest Classification
6. Desicion Tree Classification
7. GB regressor

Importing Library

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings(action='ignore' , category=FutureWarning )
```

Gathering Data

```
churn_df = pd.read_csv('Churn History Dataset.csv')
churn_test_df = pd.read_csv('Churn Test Dataset.csv')
```

```
churn_df.head()
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	tc cha
0	KS	128	415	382-4657	no	yes	25	265.1	110.0	4

```
churn_df.shape
```

```
(3333, 21)
```

1921

```
churn_df.columns
```

```
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'number customer service calls', 'Y_var'],
      dtype='object')
```

```
cat_cols = ['state' , 'international plan' , 'voice mail plan', 'Y_var']
```

```
churn_df.dtypes
```

```
state                             object
account length                     int64
area code                          int64
phone number                      object
international plan                object
voice mail plan                   object
number vmail messages              int64
total day minutes                float64
total day calls                  float64
total day charge                 float64
total eve minutes                float64
total eve calls                    int64
total eve charge                 float64
total night minutes              float64
total night calls                float64
total night charge               float64
total intl minutes               float64
total intl calls                   int64
total intl charge                float64
number customer service calls      int64
Y_var                             object
dtype: object
```

```
# Check the descriptive statistics of numeric variables
churn_df.describe()
```

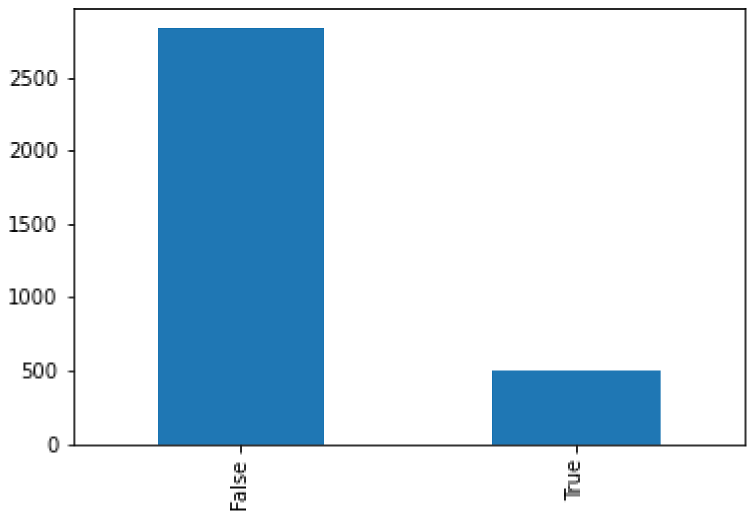
	account length	area code	number vmail messages	total day minutes	total day calls	total day charge
count	3333.000000	3333.000000	3333.000000	3333.000000	3332.000000	3332.000000
mean	101.064806	437.182418	8.099010	190.740294	103.218788	101.768574
std	39.822106	42.371290	13.688365	598.879213	128.891770	4025.094680
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000
75%	127.000000	510.000000	20.000000	216.600000	114.000000	36.825000
max	243.000000	510.000000	51.000000	34545.000000	7100.000000	232343.000000

Removing (. and whitespaces) from Y_var

```
churn_df.Y_var = churn_df.Y_var.str.strip().str.replace('.', '')
```

```
churn_df['Y_var'].value_counts().plot(kind = 'bar')
```

```
<AxesSubplot:>
```




```
churn_df['Y_var'].value_counts()
```

```
False    2829
True      504
Name: Y_var, dtype: int64
```

```
100*churn_df['Y_var'].value_counts()/len(churn_df['Y_var'])
```

```
False    84.878488
True     15.121512
Name: Y_var, dtype: float64
```

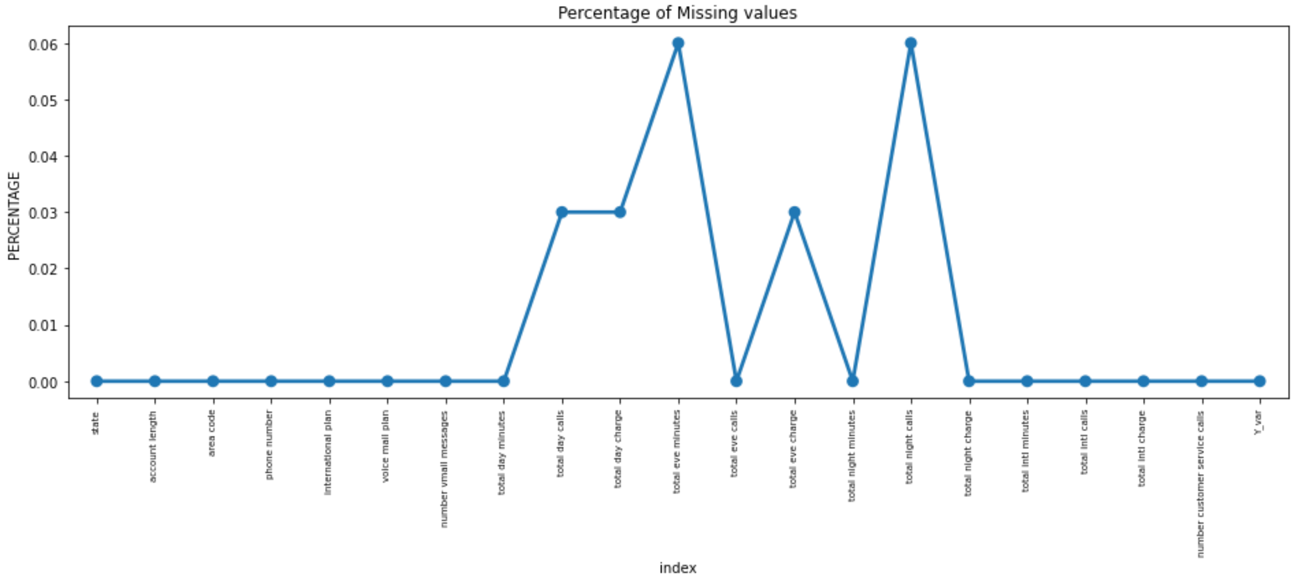
- Data is highly **imbalanced**, ratio =85:15
- So we analyse the data with other features while taking the target values separately to get some insights.

```
# Concise Summary of the dataframe, as we have too many columns, we are using the ve
churn_df.info(verbose = True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   state                           3333 non-null   object 
 1   account length                  3333 non-null   int64  
 2   area code                       3333 non-null   int64  
 3   phone number                    3333 non-null   object 
 4   international plan              3333 non-null   object 
 5   voice mail plan                 3333 non-null   object 
 6   number vmail messages           3333 non-null   int64  
 7   total day minutes               3333 non-null   float64
 8   total day calls                 3332 non-null   float64
 9   total day charge                3332 non-null   float64
 10  total eve minutes               3331 non-null   float64
 11  total eve calls                 3333 non-null   int64  
 12  total eve charge                3332 non-null   float64
 13  total night minutes             3333 non-null   float64
 14  total night calls               3331 non-null   float64
 15  total night charge              3333 non-null   float64
 16  total intl minutes              3333 non-null   float64
 17  total intl calls                3333 non-null   int64  
 18  total intl charge               3333 non-null   float64
 19  number customer service calls   3333 non-null   int64  
 20  Y_var                           3333 non-null   object 
dtypes: float64(10), int64(6), object(5)
memory usage: 546.9+ KB
```

Missing Values

```
missing = pd.DataFrame((churn_df.isnull().sum())*100/churn_df.shape[0]).reset_index(
plt.figure(figsize=(16,5))
ax = sns.pointplot('index',0,data=missing)
plt.xticks(rotation =90,fontsize =7)
plt.title("Percentage of Missing values")
plt.ylabel("PERCENTAGE")
plt.show()
```



there is some missings values in some columns but all the missing values are less than 0.06% so its safe to fill that value with mean or median

Filling missing value

```
churn_df = churn_df.fillna(churn_df.median())
```

Data Cleaning

1. Create a copy of base data for manupulation & processing

```
df = churn_df.copy()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   float64
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
```

```
 11  total eve calls               3333 non-null   int64
 12  total eve charge              3333 non-null   float64
 13  total night minutes           3333 non-null   float64
 14  total night calls             3333 non-null   float64
 15  total night charge            3333 non-null   float64
 16  total intl minutes            3333 non-null   float64
 17  total intl calls              3333 non-null   int64
 18  total intl charge             3333 non-null   float64
 19  number customer service calls 3333 non-null   int64
 20  Y_var                         3333 non-null   object
dtypes: float64(10), int64(6), object(5)
memory usage: 546.9+ KB
```

Checking Duplicate?

```
df.duplicated().sum()
```

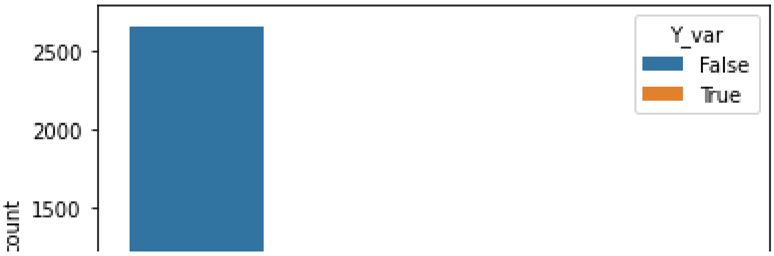
```
0
```

Exploratory Data Analysis

1. Plot distibution of individual predictors by churn

Univariate Analysis

```
for i, predictor in enumerate(df[['international plan', 'voice mail plan']]):
    plt.figure(i)
    sns.countplot(data=df, x=predictor, hue='Y_var')
```



Customer with internation plan activate have high rate of churning

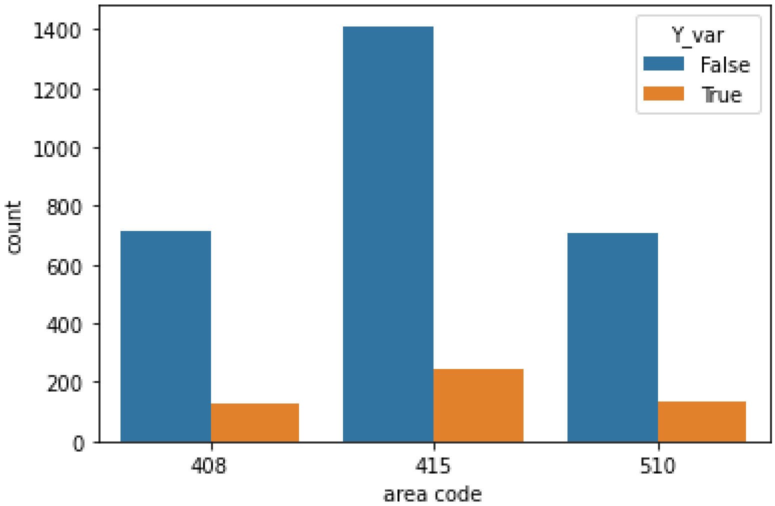
2.Convert the target variable 'Churn' in a binary numeric variable i.e. Yes=1 ; No = 0

```
df['Y_var'] = np.where(df['Y_var']=='True', 1, 0)
```

```
df['phone number'] = df['phone number'].str.replace('-','').astype('int64')
```

```
sns.countplot(x = 'area code' , hue = 'Y_var' , data = churn_df)
```

```
<AxesSubplot:xlabel='area code', ylabel='count'>
```




```
churn_df['area code'].value_counts()
```

```
415    1655
510     840
408     838
Name: area code, dtype: int64
```

Relationship between Total_Charges and (total_Calls , total_Minutes)

Filtering only `day` , `eve` , `night` , `intl calls` , `minutes and charges` columns from df columns

```
day = []
eve = []
night = []
intl = []
total =[]
for i in df.columns:
    if 'day' in i:
        day.append(i)
    elif 'eve' in i:
        eve.append(i)
    elif 'night' in i:
        night.append(i)
    elif 'intl' in i:
        intl.append(i)
for i in df.columns:
    if 'total' in i:
        total.append(i)
```

Outliers Detection

```
Q1 = churn_df[total].quantile(0.25)
Q3 = churn_df[total].quantile(0.75)
IQR = Q3-Q1
upper_limit = Q3 + 1.5 * IQR
lower_limit = Q1 - 1.5 * IQR
```

```
print("Upper limit",upper_limit)
print("Lower limit",lower_limit)
```

```
Upper limit total day minutes      325.950
total day calls        154.500
total day charge        55.405
total eve minutes      338.350
total eve calls        154.500
total eve charge        28.760
total night minutes    337.750
total night calls      152.000
total night charge      15.195
total intl minutes      17.500
total intl calls        10.500
total intl charge        4.725
dtype: float64
Lower limit total day minutes      34.350
total day calls        46.500
total day charge        5.845
total eve minutes      63.550
total eve calls        46.500
total eve charge        5.400
total night minutes    64.550
total night calls      48.000
total night charge      2.915
total intl minutes      3.100
```

```
total intl calls       -1.500
total intl charge       0.845
dtype: float64
```

Removing Outliers

```
df =df[~((df[total] < (Q1 - 1.5 * IQR)) |(df[total] > (Q3 + 1.5 * IQR))).any(axis=1)
```

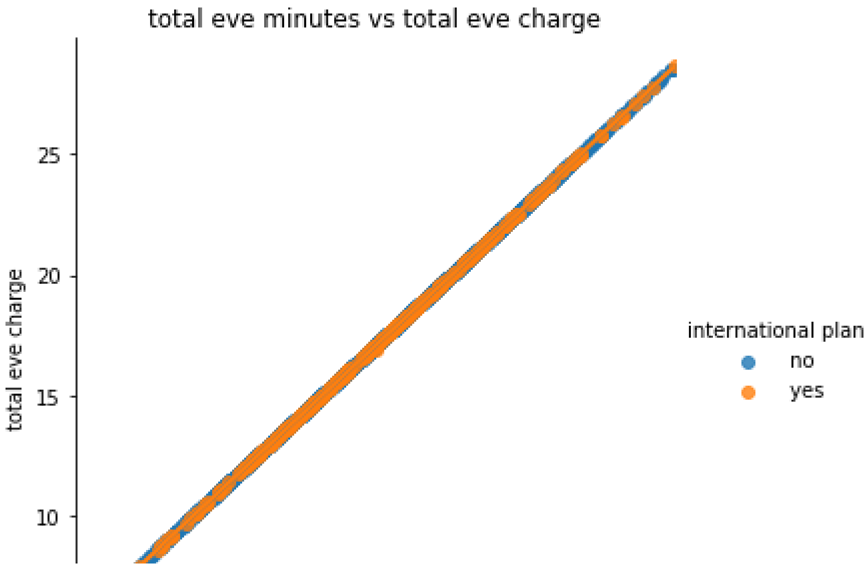
1. Relationship between total eve minutes and eve charge

```
df.head(1)
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	to cha
0	KS	128	415	3824657	no	yes	25	265.1	110.0	4

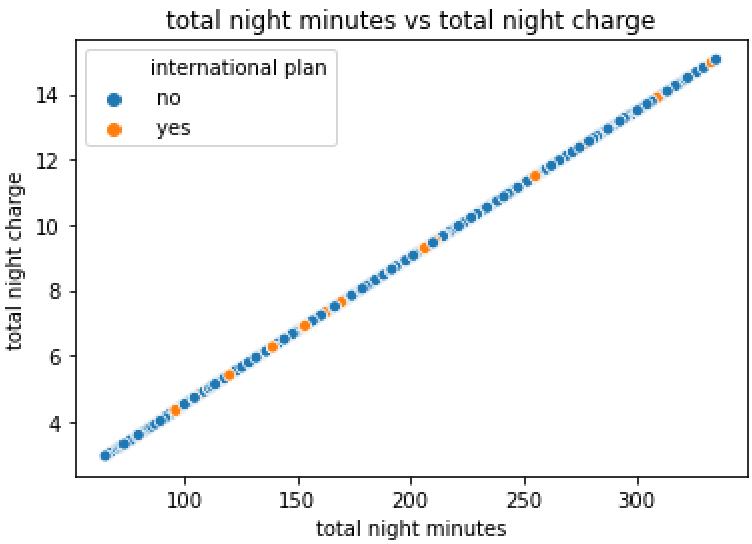
1 rows × 21 columns

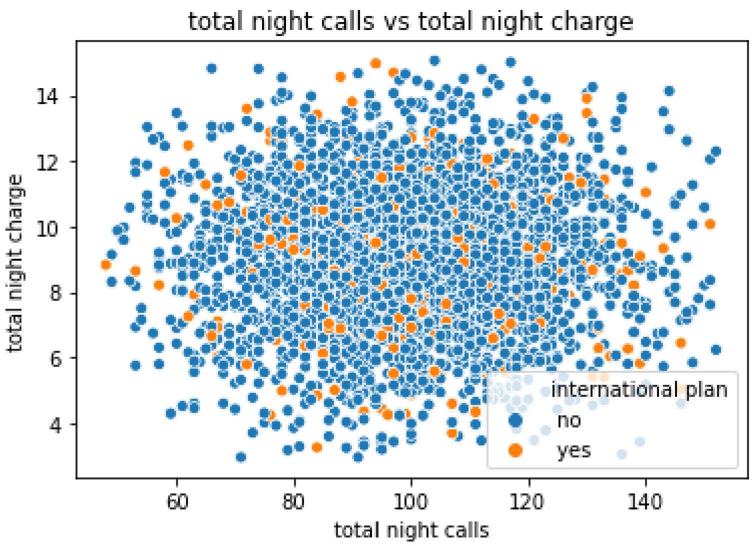
```
for i in eve[0:-1]:
    sns.lmplot(x = i , y = eve[-1] , data = df , hue = 'international plan' )
    plt.title('{} vs {}'.format(i , eve[-1]))
```



2. Relationship between total night minutes and night charge

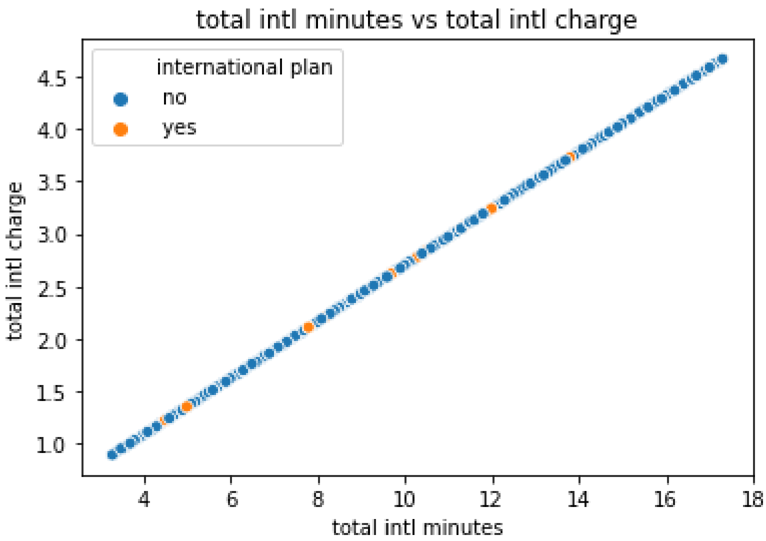
```
for i in night[0:-1]:
    plt.figure()
    sns.scatterplot(x = i , y = night[-1] , data = df , hue = 'international plan')
    plt.title('{} vs {}'.format(i , night[-1]))
```

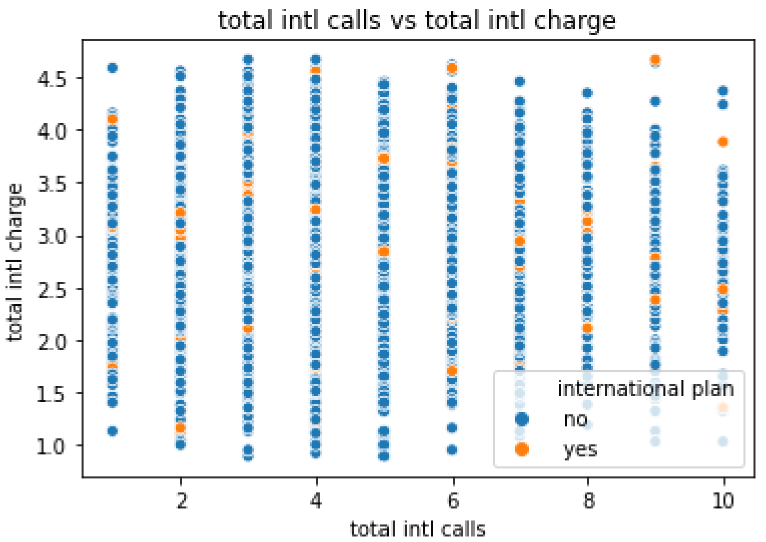




3. Relationship between total intl minutes and intl charge

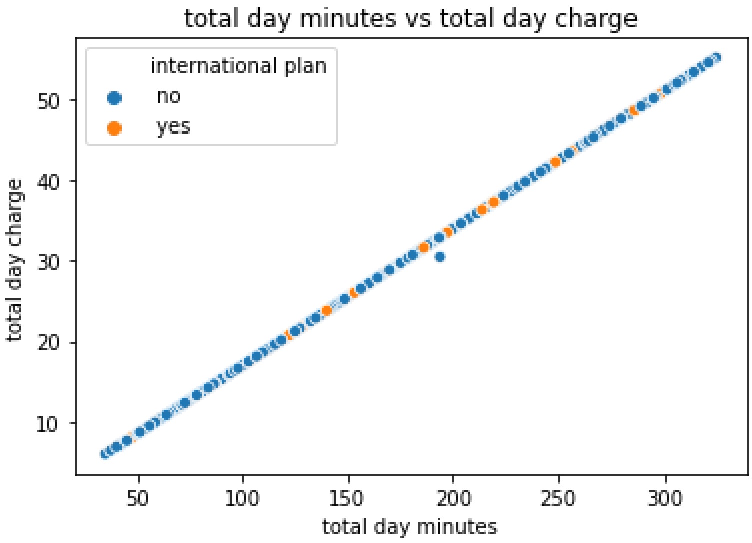
```
for i in intl[0:-1]:
    plt.figure()
    sns.scatterplot(x = i , y = intl[-1] , data = df , hue = 'international plan')
    plt.title('{} vs {}'.format(i , intl[-1]))
```

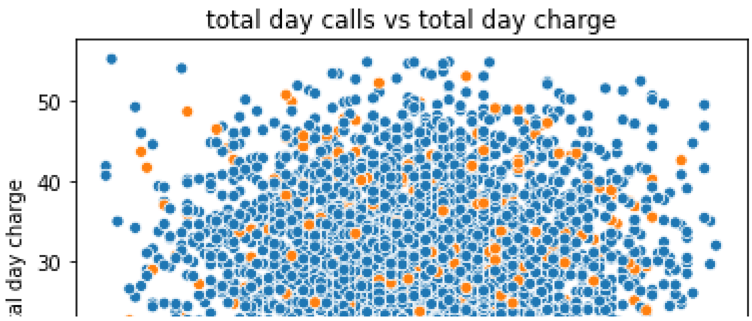




4. Relationship between total day minutes , calls and eve charge

```
for i in day[0:-1]:
    plt.figure()
    sns.scatterplot(x = i , y = day[-1] , data = df , hue = 'international plan')
    plt.title('{} vs {}'.format(i , day[-1]))
```





10. Churn by Daily Charges , Evening Charges , Night Charage , intl Charges

Churned by Internation charges

total day calls

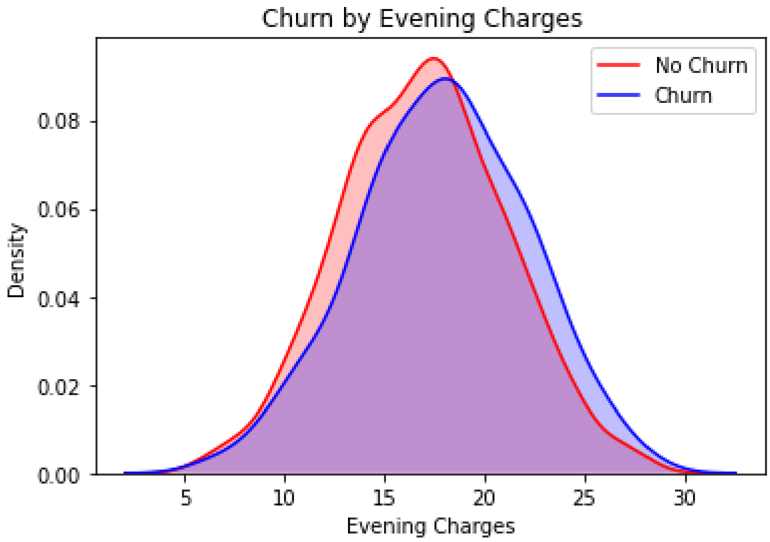
```
Intl_th = sns.kdeplot(df['total intl charge'][(df["Y_var"] == 0) ],
                color="Red", shade = True)
Intl_th = sns.kdeplot(df['total intl charge'][(df["Y_var"] == 1) ],
                ax =Intl_th, color="Blue", shade= True)
Intl_th.legend(["No Churn","Churn"],loc='upper right')
Intl_th.set_ylabel('Density')
Intl_th.set_xlabel('Intl Charges')
Intl_th.set_title('Churn by international charges')
```

- its surprising. even increasing internatinal charges churning decreases.
- i.e intl charges doesn't effect churning.
- int charges are not the cause of churning

Churned by Evening charges

```
Evening_th = sns.kdeplot(df['total eve charge'][(df["Y_var"] == 0) ],
                color="Red", shade = True)
Evening_th = sns.kdeplot(df['total eve charge'][(df["Y_var"] == 1) ],
                ax =Evening_th, color="Blue", shade= True)
Evening_th.legend(["No Churn","Churn"],loc='upper right')
Evening_th.set_ylabel('Density')
Evening_th.set_xlabel('Evening Charges')
Evening_th.set_title('Churn by Evening Charges  ')
```

```
Text(0.5, 1.0, 'Churn by Evening Charges  ')
```

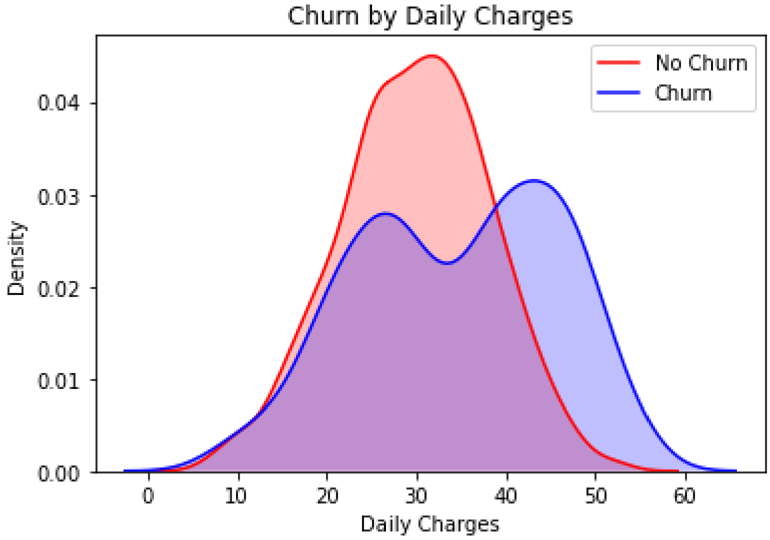


- evening charges are also not effecting churning rate

Churned by daily charges

```
Dth = sns.kdeplot(df['total day charge'][(df["Y_var"] == 0) ],
                color="Red", shade = True)
Dth = sns.kdeplot(df['total day charge'][(df["Y_var"] == 1) ],
                ax =Dth, color="Blue", shade= True)
Dth.legend(["No Churn","Churn"],loc='upper right')
Dth.set_ylabel('Density')
Dth.set_xlabel('Daily Charges')
Dth.set_title(' Churn by Daily Charges ')
```

```
Text(0.5, 1.0, ' Churn by Daily Charges ')
```

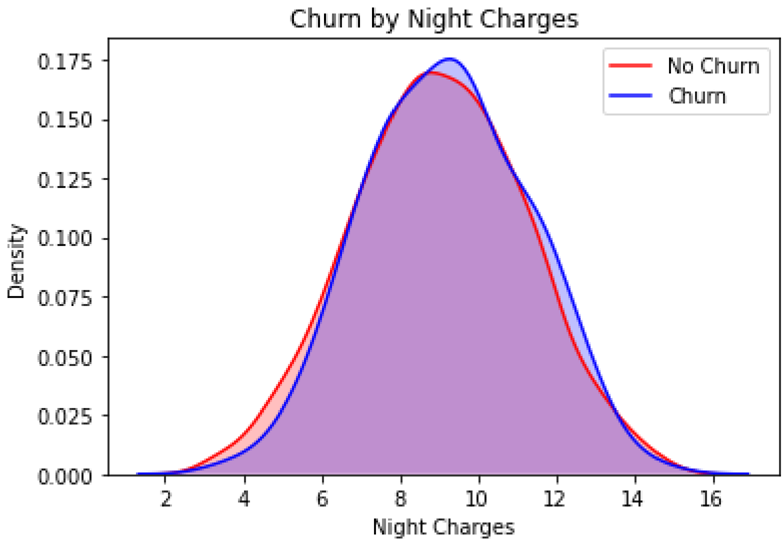


- Increasing in day calling charges churning also increasing
- i.e client should be decreased daily charges of calling so that churning decrease and customer buy another plan

Churned by night charges

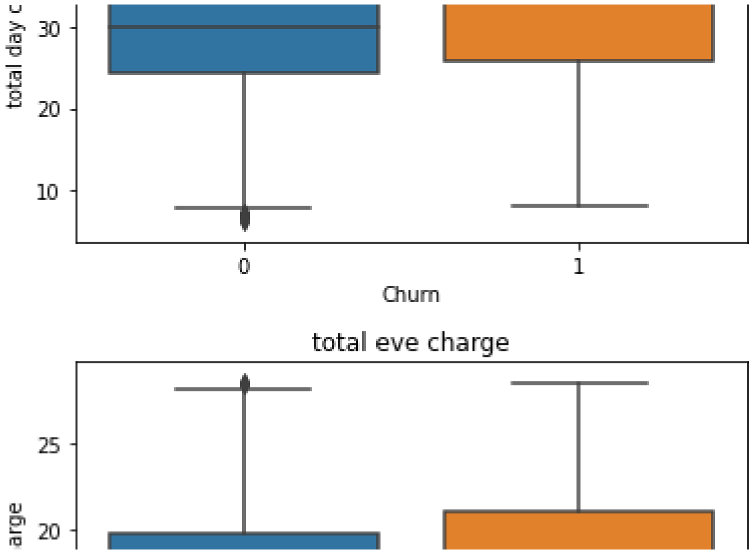
```
night_th = sns.kdeplot(df['total night charge'][(df["Y_var"] == 0) ],
                color="Red", shade = True)
night_th = sns.kdeplot(df['total night charge'][(df["Y_var"] == 1) ],
                ax =night_th, color="Blue", shade= True)
night_th.legend(["No Churn","Churn"],loc='upper right')
night_th.set_ylabel('Density')
night_th.set_xlabel('Night Charges')
night_th.set_title('Churn by Night Charges  ')
```

```
Text(0.5, 1.0, 'Churn by Night Charges  ')
```



night charges doesn't seems effecting churning

```
charge = ['total day charge','total eve charge','total intl charge', 'total night ch
for i in charge:
    plt.figure()
    plt.title('{}'.format(i))
    sns.boxplot(x ='Y_var' , y= i , data = df)
    plt.xlabel("Churn")
    plt.show()
```



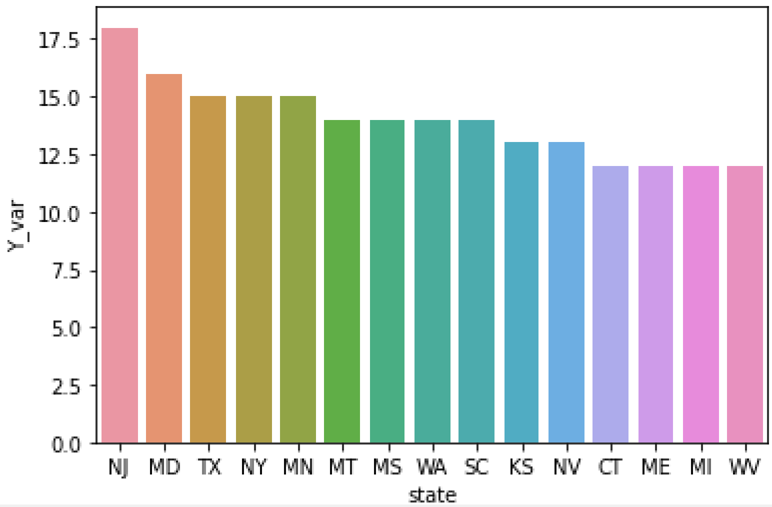
Plot insights:

- Churning customers have higer day charges with a median of ca. 35 USD compared to a median of non-churners of ca. 30 USD.
- Churning customers have higher evening charges with a median of ca.20 USD and much lower interquartile range compared to that of non-churners (median of ca. 17 USD).

Highest Churning state

```
state = df.groupby('state')['Y_var'].sum().sort_values(ascending = False).reset_inde
print('Highest Churne state: {} '.format(state['state'].tolist()))
sns.barplot(x='state' , y='Y_var' , data = state)
```

```
Highest Churne state: ['NJ', 'MD', 'TX', 'NY', 'MN', 'MT', 'MS', 'WA', 'SC', 'KS', 'N
<AxesSubplot:xlabel='state', ylabel='Y_var'>
```



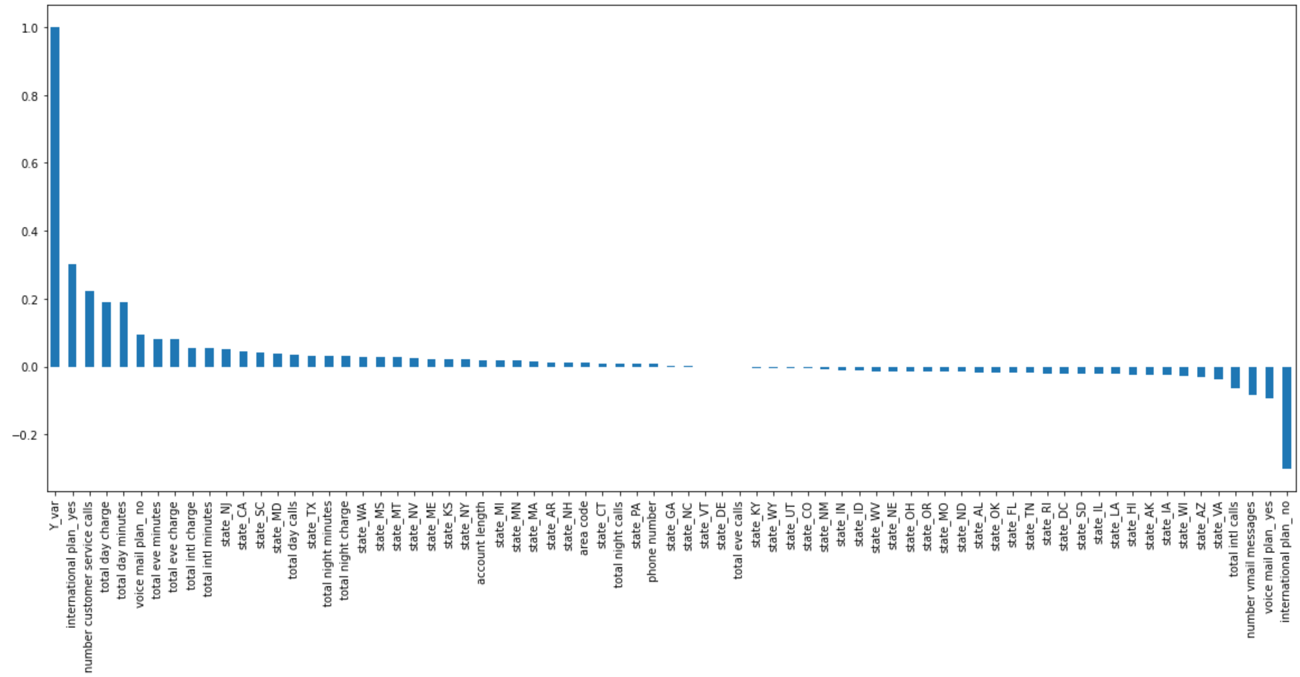
Dummies for categorical columns

```
df = pd.get_dummies(df)
```

Build a correlation of all predictors with 'Churn'

```
plt.figure(figsize=(20,8))
df.corr()['Y_var'].sort_values(ascending = False).plot(kind='bar')
```

```
<AxesSubplot:>
```



Derived Insight:

HIGH Churn seen in case of **internation plan yes**, **No voice plan**, **when number of customer service calls more** and **total eve charges increased**

LOW Churn is seens in case of **Subscriptions without international plan**, **Subscriptions with voice mail plan service** and **number of vmail messages service**

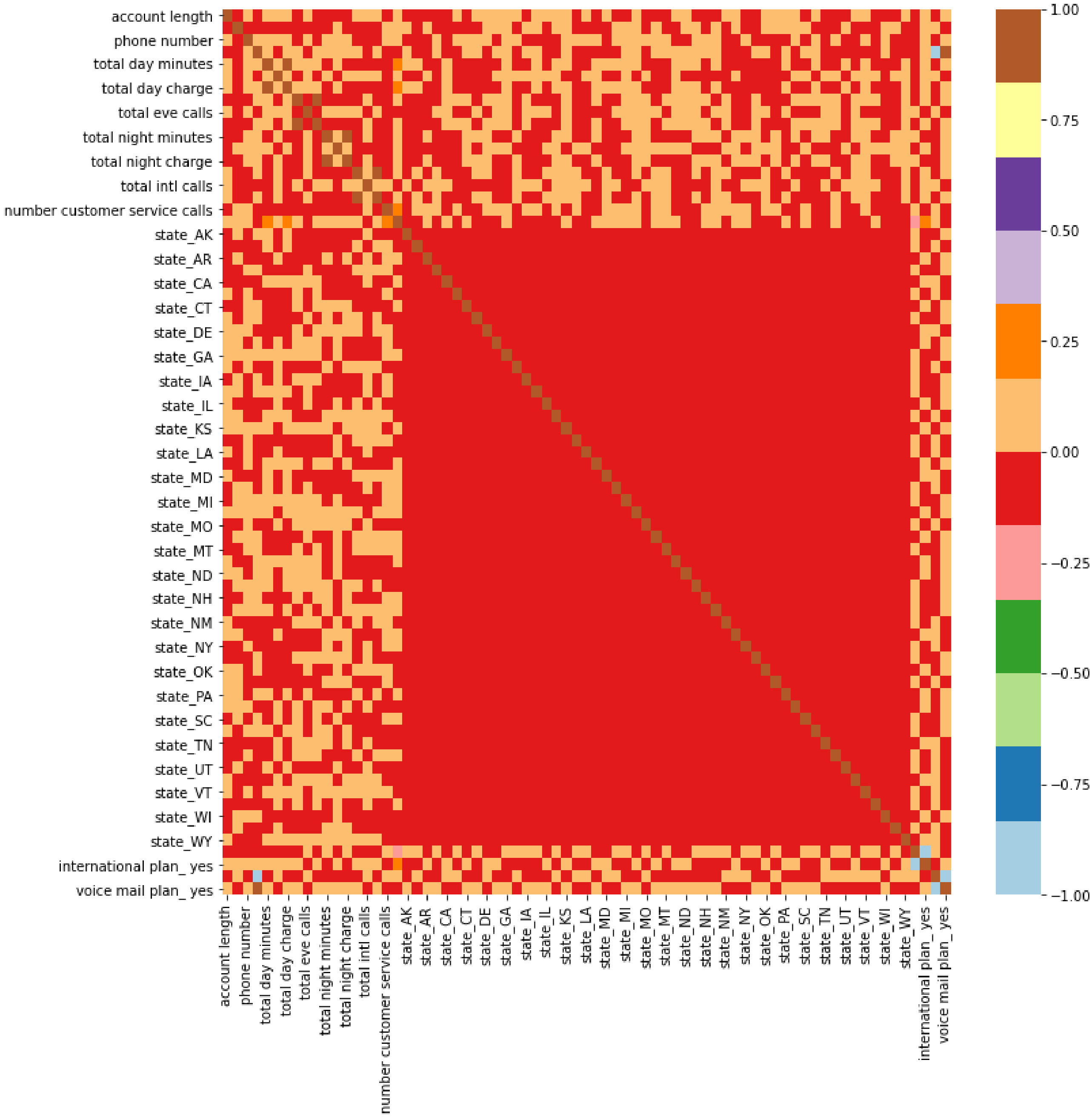
Factors like **total day charge**, **total day calls** and **# state GA , UT , OR , NH , NC , DE** have alomost **NO** impact on Churn

This is also evident from the **Heatmap** below

```
plt.figure(figsize=(12,12))
```

```
sns.heatmap(df.corr(), cmap="Paired")
```

<AxesSubplot:>



CONCLUSION

1. with internatinal plan are the highest churners

2. without voice mail plan are more likely to churn because of no contract terms, as they are free to go customers.
3. state VA and state AZ customers are low churners

```
test_df = churn_test_df.copy()
```

Churn test data

```
test_df['Churn Indicator'] = test_df['Churn Indicator'].str.strip().str.replace('.',
test_df['Churn Indicator'] = np.where(test_df['Churn Indicator']=='True', 1, 0)
test_df['phone number'] = test_df['phone number'].str.replace('-','').astype('int64'
test_df = pd.get_dummies(test_df)
test_df_train = test_df.drop('Churn Indicator' , axis = 1)
```

Model Building

```
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from imblearn.combine import SMOTEENN
```

```
X = df.drop('Y_var' , axis =1)
y = df['Y_var']
```

```
X.head()
```

	account length	area code	phone number	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge
0	128	415	3824657	25	265.1	110.0	45.07	197.4	99	16.78
1	107	415	3717191	26	161.6	123.0	27.47	195.5	103	16.62
2	137	415	3581921	0	243.4	114.0	41.38	121.2	110	10.30
4	75	415	3306626	0	166.7	113.0	28.34	148.3	122	12.61
5	118	510	3918027	0	223.4	98.0	37.98	220.6	101	18.75

5 rows × 72 columns

Train Test Split

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
model_lr= LogisticRegression(random_state=0)
model_lr.fit(X_train, y_train)
```

```
LogisticRegression(random_state=0)
```

```
y_pred = model_lr.predict(X_test)
y_pred[0:10]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
model_lr.score(X_test , y_test)
```

```
0.8583061889250815
```

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.86      1.00      0.92       527
           1       0.00      0.00      0.00        87

    accuracy                           0.86       614
   macro avg       0.43      0.50      0.46       614
weighted avg       0.74      0.86      0.79       614

C:\Users\rakesh.kumar\Miniconda3\envs\clf-env-dev\lib\site-packages\sklearn\metrics\_
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rakesh.kumar\Miniconda3\envs\clf-env-dev\lib\site-packages\sklearn\metrics\_
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rakesh.kumar\Miniconda3\envs\clf-env-dev\lib\site-packages\sklearn\metrics\_
  _warn_prf(average, modifier, msg_start, len(result))
```

```
print('score of churn_test_df with LogisticRegression' , model_lr.score(test_df_trai
print(classification_report(test_df['Churn Indicator'] , model_lr.predict(test_df_tr
```

```
score of churn_test_df with LogisticRegression 0.8564593301435407
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      1432
           1       0.00      0.00      0.00       240
```

```
    accuracy                           0.86      1672
   macro avg       0.43      0.50      0.46      1672
weighted avg       0.73      0.86      0.79      1672

C:\Users\rakesh.kumar\Miniconda3\envs\clf-env-dev\lib\site-packages\sklearn\metrics\_
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rakesh.kumar\Miniconda3\envs\clf-env-dev\lib\site-packages\sklearn\metrics\_
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rakesh.kumar\Miniconda3\envs\clf-env-dev\lib\site-packages\sklearn\metrics\_
  _warn_prf(average, modifier, msg_start, len(result))
```

As you can see that the accuracy is quite low, and as it's an imbalanced dataset, we shouldn't consider Accuracy as our metrics to measure the model, as Accuracy is cursed in imbalanced datasets.

Hence, we need to check recall, precision & f1 score for the minority class, and it's quite evident that the precision, recall & f1 score is too low for Class 1, i.e. churned customers.

Hence, moving ahead to call SMOTEENN (UpSampling + ENN)

```
sm = SMOTEENN()
X_resampled, y_resampled = sm.fit_resample(X, y)
Xr_train,Xr_test,yr_train,yr_test=train_test_split(X_resampled, y_resampled,test_siz
model_lr_smote=LogisticRegression()


model_lr_smote.fit(Xr_train,yr_train)
yr_predict = model_lr_smote.predict(Xr_test)
model_score_r = model_lr_smote.score(Xr_test, yr_test)
print(model_score_r)
print(metrics.classification_report(yr_test, yr_predict))
```

```
0.5885714285714285
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       144
           1       0.59      1.00      0.74       206

    accuracy                           0.59       350
   macro avg       0.29      0.50      0.37       350
weighted avg       0.35      0.59      0.44       350

C:\Users\rakesh.kumar\Miniconda3\envs\clf-env-dev\lib\site-packages\sklearn\metrics\_
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rakesh.kumar\Miniconda3\envs\clf-env-dev\lib\site-packages\sklearn\metrics\_
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rakesh.kumar\Miniconda3\envs\clf-env-dev\lib\site-packages\sklearn\metrics\_
  _warn_prf(average, modifier, msg_start, len(result))
```

Accuracy is very low with logistic Regression . we will try some other classfier

Decision Tree Classifier

[ ] ↳ *11 cells hidden*

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
model_rf = RandomForestClassifier()
model_rf.fit(X_train , y_train)
model_rf.score(X_test , y_test)
```

```
0.9332247557003257
```

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       527
           1       0.83      0.69      0.75        87

    accuracy                           0.94       614
   macro avg       0.89      0.83      0.86       614
weighted avg       0.93      0.94      0.93       614
```

Handling Imblanced data

```
sm = SMOTEENN()
X_resampled, y_resampled = sm.fit_resample(X, y)
Xr_train,Xr_test,yr_train,yr_test=train_test_split(X_resampled, y_resampled,test_siz
model_rf_smote=RandomForestClassifier()
```

```
model_rf_smote.fit(Xr_train,yr_train)
yr_predict = model_rf_smote.predict(Xr_test)
model_score_r = model_rf_smote.score(Xr_test, yr_test)
print(model_score_r)
print(metrics.classification_report(yr_test, yr_predict))
```

```
0.9127906976744186
              precision    recall  f1-score   support

           0       0.93      0.87      0.90       157
           1       0.90      0.95      0.92       187

    accuracy                           0.91       344
   macro avg       0.92      0.91      0.91       344
```

```
weighted avg       0.91      0.91      0.91       344
```

```
print(metrics.confusion_matrix(yr_test, yr_predict))
```

```
[[137  20]
 [ 10 177]]
```

With RF Classifier, also we are able to get very good results, infact better than Decision Tree.

Accuracy is 93% and its also working fine with minority clsss

let's finalise the model which was created by RF Classifier, and save the model so that we can use it in a later stage :)

Pickling the model

```
import pickle
filename = 'model.sav'
pickle.dump(model_rf_smote , open(filename, 'wb'))
load_model = pickle.load(open(filename , 'rb'))
```

```
model_score_r = load_model.score(Xr_test, yr_test)
```

```
print(model_score_r)
```

```
0.9127906976744186
```

Accuracy with churn_train_df

```
print('score of churn_test_df with Random Forest Classifier' ,model_score_r)
print(metrics.classification_report(yr_test, yr_predict))
```

```
score of churn_test_df with Random Forest Classifier 0.9127906976744186
              precision    recall  f1-score   support

           0       0.93      0.87      0.90       157
           1       0.90      0.95      0.92       187

    accuracy                           0.91       344
   macro avg       0.92      0.91      0.91       344
weighted avg       0.91      0.91      0.91       344
```

Accuracy with churn_test_df

```
print('score of churn_test_df with Random Forest Classifier' ,load_model.score(test_
print(classification_report(test_df['Churn Indicator'] ,load_model.predict(test_df_t
```

```
score of churn_test_df with Random Forest Classifier 0.8558612440191388
              precision    recall  f1-score   support

           0       0.95      0.88      0.91      1432
           1       0.50      0.71      0.59       240

    accuracy                           0.86      1672
   macro avg       0.72      0.80      0.75      1672
weighted avg       0.88      0.86      0.87      1672
```

we are getting 85% accuracy score with random forest on our churn_test data . its also giving quite good result with minority class than other model