Telecom Churn Case Study

By –

0.Importing Packages and the data

```
[ ]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score
import warnings
warnings.filterwarnings('ignore')
```

Important Packages needed for the models total_rech_data_amt_X = av_rech_amt_data_X * total_rech_data_X Where X is the month (6,7,8,9)

In [ ]:
```
telecom['total_rech_data_amt_6'] = telecom['av_rech_amt_data_6'] * telecom['total_rech_data_6']
telecom['total_rech_data_amt_7'] = telecom['av_rech_amt_data_7'] * telecom['total_rech_data_7']
telecom['total_rech_data_amt_8'] = telecom['av_rech_amt_data_8'] * telecom['total_rech_data_8']
telecom['total_rech_data_amt_9'] = telecom['av_rech_amt_data_9'] * telecom['total_rech_data_9']
```

Drop the columns total_rech_data_X and av_rech_amt_data_X

total_rech_data_amt_X = av_rech_amt_data_X * total_rech_data_X Where X is the month (6,7,8,9)

70th Percentile of average recharge amount in first 2 months

```
In [ ]: telecom_av_rech_6_7 = (telecom['total_rech_amt_6'].fillna(0)
        + telecom['total_rech_amt_7'].fillna(0)
        + telecom['total_rech_data_amt_6'].fillna(0)
        + telecom['total_rech_data_amt_7'].fillna(0))/2
```

```
In [ ]: telecom_av_rech_6_7
```

```
Out[ ]: 0         559.0
        1         306.0
        2         241.5
        3         270.0
        4         301.0
                  ...
        99994      85.0
        99995     110.0
        99996      98.5
        99997    1602.0
        99998     432.0
        Length: 99999, dtype: float64
```

```
In [ ]: # 70th percentile value
        percentile_70_6_7 = np.percentile(telecom_av_rech_6_7, 70.0)
        print("70th percentile - ", percentile_70_6_7)

        70th percentile -  478.0
```

```
In [ ]: # fitler the given data set based on 70th percentile
        telecom_hv_cust = telecom[telecom_av_rech_6_7 >= percentile_70_6_7]
```

```
In [ ]: telecom_hv_cust.info()

        <class 'pandas.core.frame.DataFrame'>
        Int64Index: 30001 entries, 0 to 99997
        Columns: 222 entries, mobile_number to total_rech_data_amt_9
        dtypes: float64(175), int64(35), object(12)
        memory usage: 51.0+ MB
```

Figuring out 70th percentile value

```
In [ ]: # Define 'churn' as given in the problem statement
        telecom_hv_cust['churn'] = np.where(telecom_hv_cust[['total_ic_mou_9','total_og_mou_9','vol_2g_mb_9','vol_3g_mb_9']].sum(axis=1) == 0, 1,(
        telecom_hv_cust.head()
```

_9	arpu_6	...	aon	aug_vbc_3g	jul_vbc_3g	jun_vbc_3g	sep_vbc_3g	total_rech_data_amt_6	total_rech_data_amt_7	total_rech_data_amt_8	total_rech_data_amt_9	churn
14	197.385	...	968	30.40	0.00	101.20	3.58	252.0	252.0	252.0	NaN	1
14	1069.180	...	802	57.74	19.38	18.74	0.00	NaN	NaN	NaN	NaN	1
14	378.721	...	315	21.03	910.65	122.16	0.00	NaN	354.0	207.0	NaN	0
14	514.453	...	720	0.00	0.00	0.00	0.00	NaN	NaN	NaN	NaN	0
14	74.350	...	604	40.45	51.86	0.00	0.00	NaN	712.0	540.0	252.0	0

Defining Churn

Imbalance ratio in churn

```
In [ ]: # lets find out imbalance in churn
        telecom_hv_cust['churn'].value_counts()/len(telecom_hv_cust)*100

Out[ ]: 0    91.863605
        1     8.136395
        Name: churn, dtype: float64
```

Imbalance ratio observations :

- There is very high imbalance in the data.
- Imbalance ration is approx 92:8
- We will do imbalance treatment later

For data preparation :

Step I -Drop columns with only one unique values

Step II -Check null values

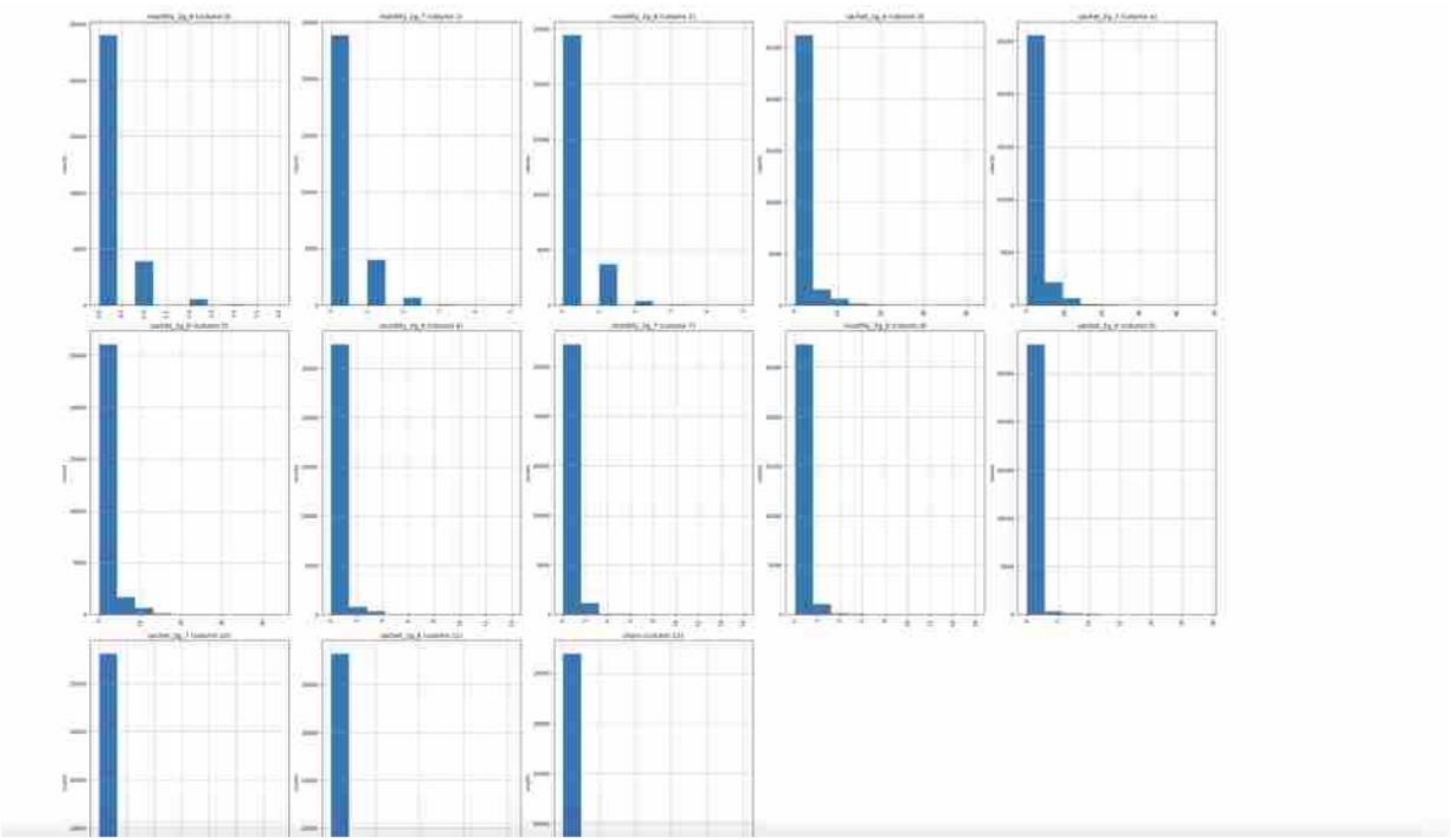
Step III -Check columns with object datatype

Step IV -Drop highly correlated columns

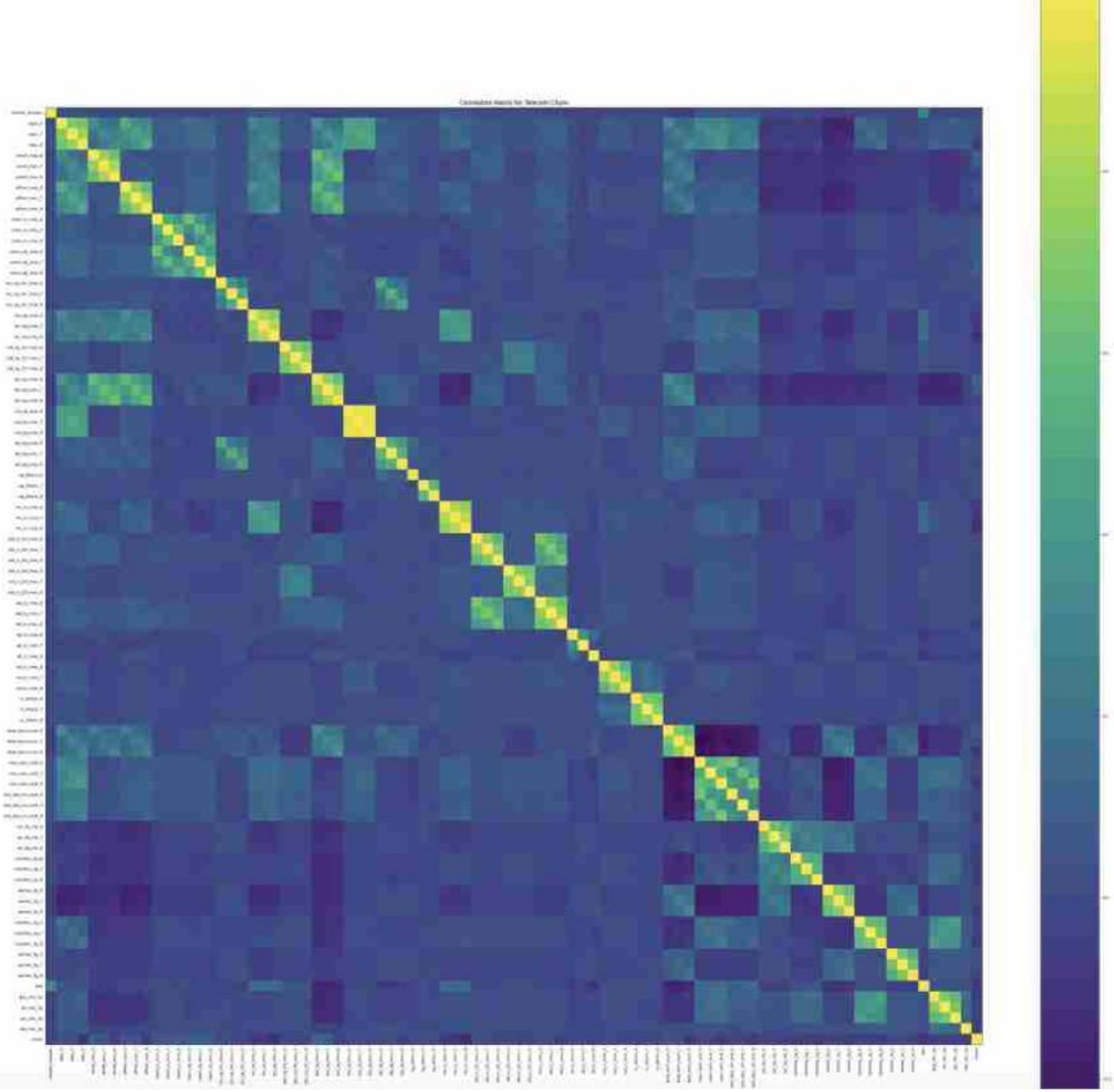
Step V -Delete columns of 9th month

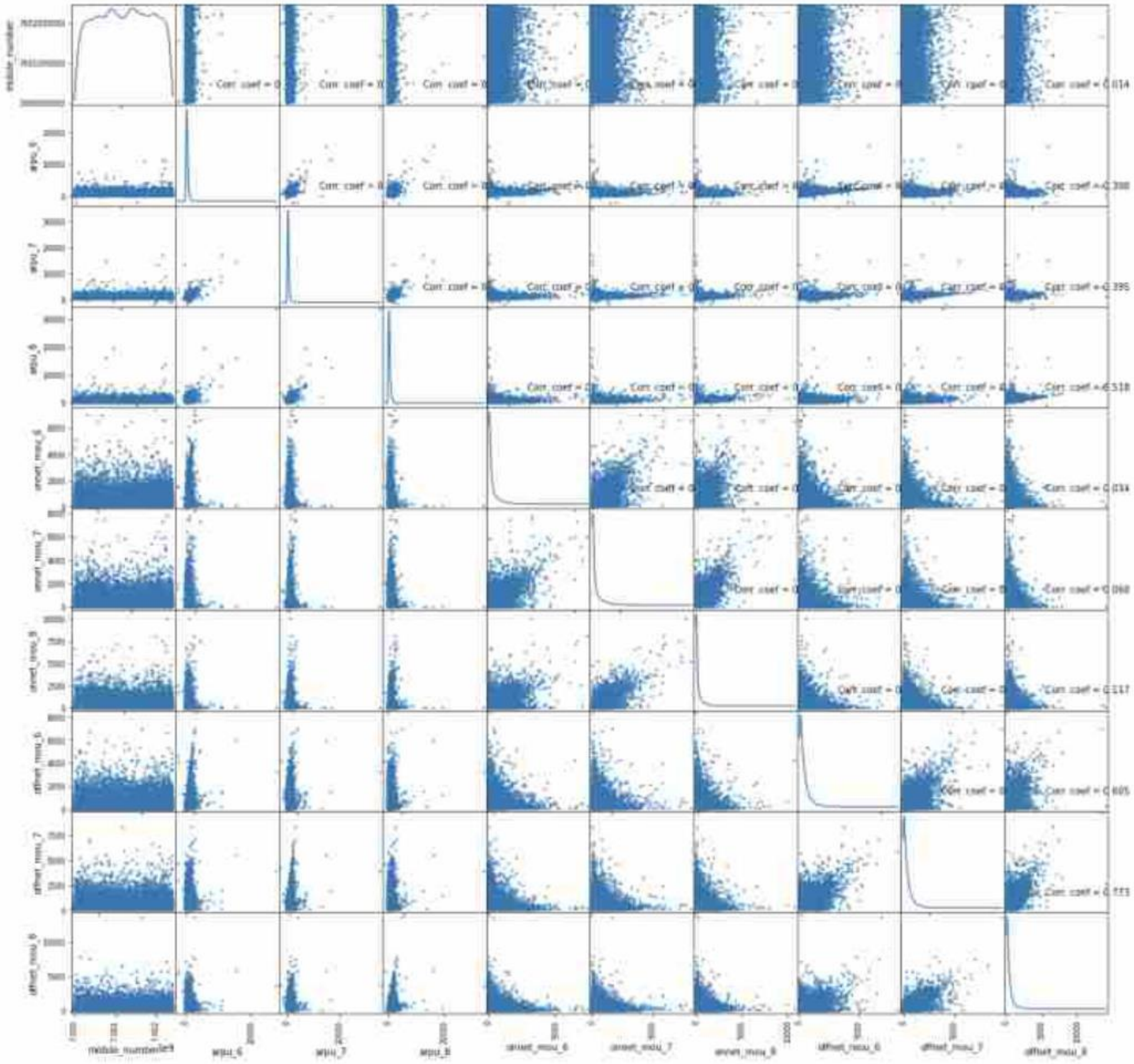
Observations :

There are columns that can be converted to datetime
There are few columns with ~4% null values



Plotting the graphs


Correlation Matrix for Telecom Churn


mobile_number
arpu_6
arpu_7
arpu_8
onnet_mou_6
onnet_mou_7
onnet_mou_8
offnet_mou_6
offnet_mou_7
offnet_mou_8

```
In [ ]: # Create list of columns belonging to 6th and 7th months
        col_list = telecom_hv_cust.filter(regex='_6|_7').columns.str[:-2]
        col_list.unique()

        print (telecom_hv_cust.shape)

        (28504, 88)

In [ ]: # Calculate the average
        for col in col_list.unique():
            avg_col_name = "avg_"+col+"_av67" # Name of the new columns
            col_6 = col+"_6"
            col_7 = col+"_7"
            telecom_hv_cust[avg_col_name] = (telecom_hv_cust[col_6]  + telecom_hv_cust[col_7])/ 2

In [ ]: # Shape before dropping columns
        telecom_hv_cust.shape

Out[ ]: (28504, 115)

In [ ]: # Drop the original columns whose average was calculated
        col_to_drop = telecom_hv_cust.filter(regex='_6|_7').columns
        telecom_hv_cust.drop(col_to_drop, axis=1, inplace=True)

        # Shape after dropping columns
        telecom_hv_cust.shape

Out[ ]: (28504, 61)
```

Creating columns that are average of 6th and 7th months

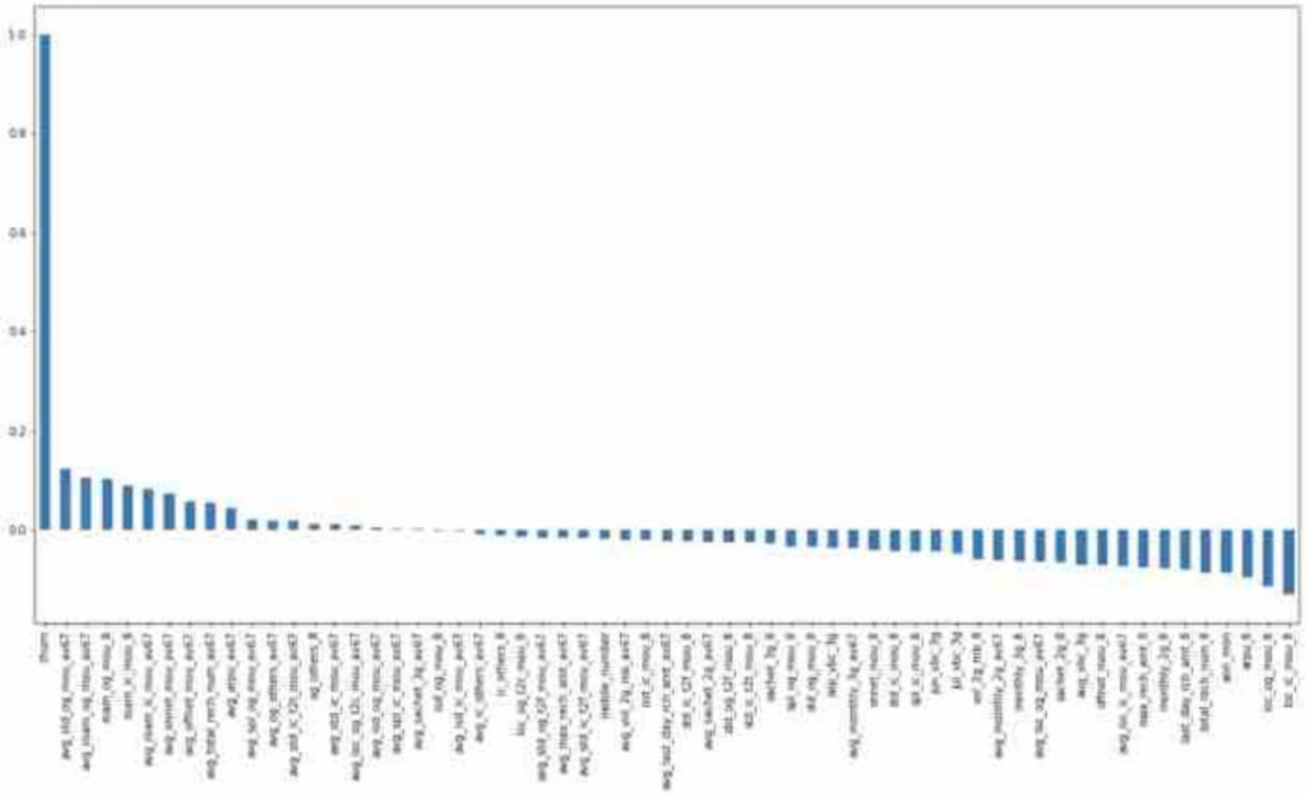
```
In [ ]: # lets now convert AON (Age on Network) in months
        telecom_hv_cust['aon_mon'] = telecom_hv_cust['aon']/30
        telecom_hv_cust.drop('aon', axis=1, inplace=True)
        telecom_hv_cust['aon_mon'].head()

Out[ ]: 7     26.733333
        8     10.500000
        21    24.000000
        23    20.133333
        33    44.266667
        Name: aon_mon, dtype: float64
```

Creating Age on network

7.Bivariate analysis on 'churn'

In [ ]:
```
# Correlation of churn with other columns
plt.figure(figsize=(20,10))
telecom_hv_cust.corr()['churn'].sort_values(ascending = False).plot(kind='bar')
```

Out[ ]:
```
<AxesSubplot:>
```

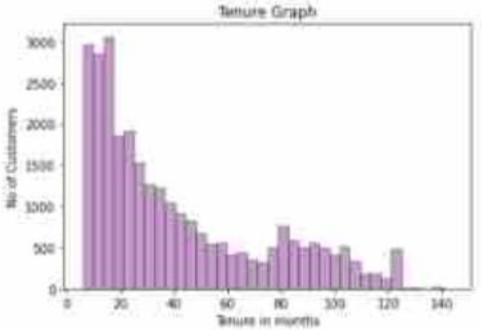


OBSERVARIONS

•Avg Outgoing Calls & calls on roaming for 6 & 7th months are positively correlated with churn.

•Avg Revenue, No. Of Recharge for 8th month has negative correlation with churn.

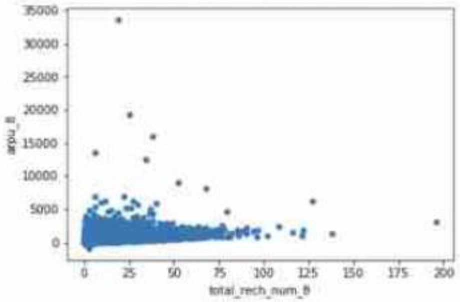
```
In [ ]: # Plot distribution of AON (in months)

        ax = sns.distplot(telecom_hv_cust['aon_mon'], hist=True, kde=False,
                    bins=int(180/5), color = 'purple',
                    hist_kws={'edgecolor':'black'},
                    kde_kws={'linewidth': 10})

        ax.set_ylabel('No of Customers')
        ax.set_xlabel('Tenure in months')
        ax.set_title('Tenure Graph')

Out[ ]: Text(0.5, 1.0, 'Tenure Graph')
```



Plot Distribution of AON

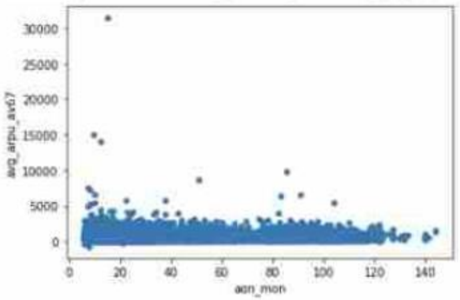
```
In [ ]: # lets now draw a scatter plot between total recharge and avg revenue for the 8th month
        telecom_hv_cust[['total_rech_num_8', 'arpu_8']].plot.scatter(x = 'total_rech_num_8',
                                                                    y='arpu_8')
```

```
Out[ ]: <AxesSubplot:xlabel='total_rech_num_8', ylabel='arpu_8'>
```



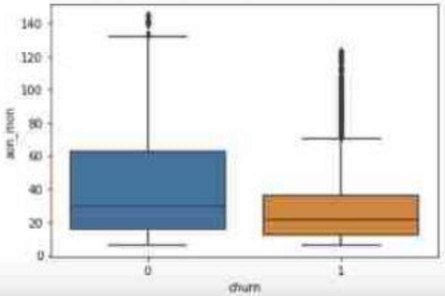

```
In [ ]: # plot between tenure and revenue
        telecom_hv_cust[['aon_mon', 'avg_arpu_av67']].plot.scatter(x = 'aon_mon',
                                                                   y='avg_arpu_av67')
```

```
Out[ ]: <AxesSubplot:xlabel='aon_mon', ylabel='avg_arpu_av67'>
```




```
In [ ]: # AON vs Churn
        sns.boxplot(x = telecom_hv_cust.churn, y = telecom_hv_cust.aon_mon)
```

```
Out[ ]: <AxesSubplot:xlabel='churn', ylabel='aon_mon'>
```



Tenured customers do no churn and they keep availing telecom services.

MODEL 1: SVM Model

In [ ]:
```
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=99)

lr = LogisticRegression()

lr.svm = SVC(kernel='linear')
lr.svm.fit(X_train,y_train)
preds = lr.svm.predict(X_test)
metrics.accuracy_score(y_test, preds)*100
```

Out[ ]:
```
94.19400105244694
```

OBSERVATIONS

- Linear SVM gave us accuracy of 94% on test data

MODEL 2: Logistic regression model using RFE supported columns

In [ ]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state=1)
lr.fit(X_rfe, y_rfe)
```

Out[ ]:
```
▾        LogisticRegression
LogisticRegression(random_state=1)
```

In [ ]:
```
X_test_rfe = pd.DataFrame(data=X_test).iloc[:, rfe.support_]

y_pred = lr.predict(X_test_rfe)

from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(confusion_matrix)
print('Accuracy on the test data:',lr.score(X_test_rfe, y_test))
```

```
Confusion Matrix:
[[4151 1219]
 [  60  271]]
Accuracy on the test data: 0.7756533941413787
```

In [ ]:
```
# Classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

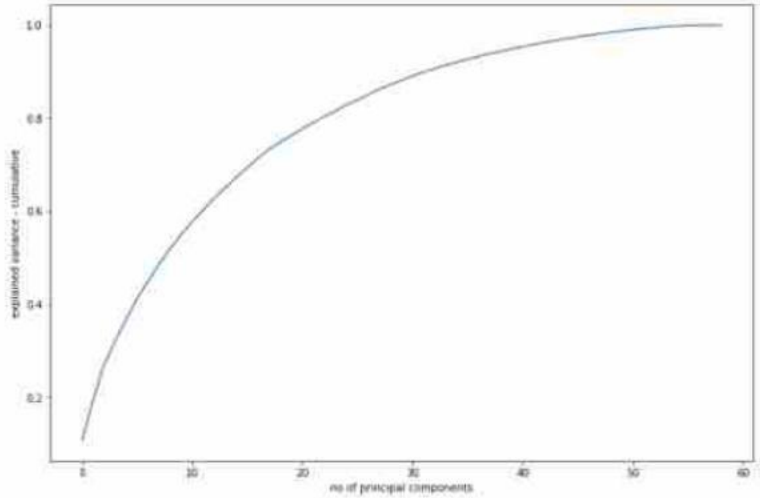
```
              precision    recall  f1-score   support

           0       0.99      0.77      0.87      5370
           1       0.18      0.82      0.30       331

    accuracy                           0.78      5701
   macro avg       0.58      0.80      0.58      5701
weighted avg       0.94      0.78      0.83      5701
```

OBSERVATIONS

- Model Accuracy is approx 78%
- Confusion matix shows high false positive rate, which is not good.

Model 3 - PCA

```
In [ ]: # scree plot to check the variance explained by different PCAs
        fig = plt.figure(figsize = (12,8))
        plt.plot(np.cumsum(pca.explained_variance_ratio_))
        plt.xlabel('no of principal components')
        plt.ylabel('explained variance - cumulative')
        plt.show()
```



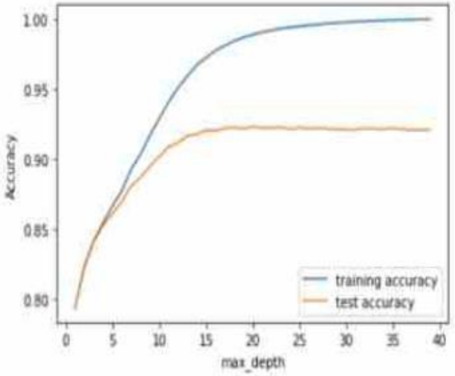

```
In [ ]: np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)

Out[ ]: array([ 10.84,  19.41,  27.12,  32.09,  36.83,  41.27,  44.97,  48.61,
                51.98,  55.  ,  57.75,  60.31,  62.76,  65.07,  67.26,  69.39,
                71.45,  73.32,  74.89,  76.33,  77.74,  79.09,  80.36,  81.59,
                82.79,  83.96,  85.1 ,  86.2 ,  87.22,  88.13,  89.03,  89.91,
                90.7 ,  91.38,  92.05,  92.7 ,  93.28,  93.84,  94.39,  94.93,
                95.39,  95.85,  96.3 ,  96.72,  97.1 ,  97.47,  97.82,  98.15,
                98.47,  98.75,  99.02,  99.27,  99.5 ,  99.68,  99.84,  99.97,
               100.02, 100.02, 100.02])
```

OBSERVATIONS

- 33 columns explains 90% of the variance, lets apply PCA with 33 components

Model 4 – Decision Tree

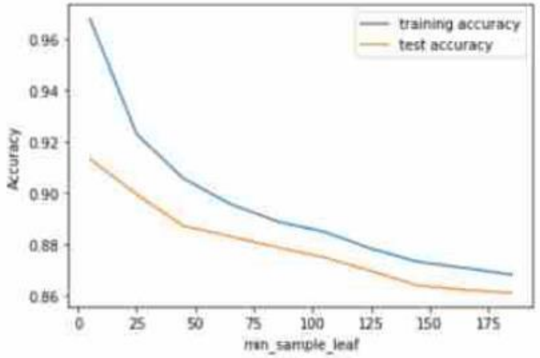
```
In [ ]: # plotting accuracies with max_depth
plt.figure()
plt.plot(score["param_max_depth"],
         score["mean_train_score"],
         label="training accuracy")
plt.plot(score["param_max_depth"],
         score["mean_test_score"],
         label="test accuracy")
plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



OBSERVATIONS

- max_depth of 10 seems to be the optimal one

In [ ]:
```
# plotting accuracies with min_sample_leaf
plt.figure()
plt.plot(score["param_min_samples_leaf"],
         score["mean_train_score"],
         label="training accuracy")
plt.plot(score["param_min_samples_leaf"],
         score["mean_test_score"],
         label="test accuracy")
plt.xlabel("min_sample_leaf")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



OBSERVATIONS

- 25 min_sample_leaf 25 seems to be the optimal one

In [ ]:
```
# accuracy score
print ('Accuracy Score for Decision Tree Final Model :',clf_gini.score(X_test,y_test))
```

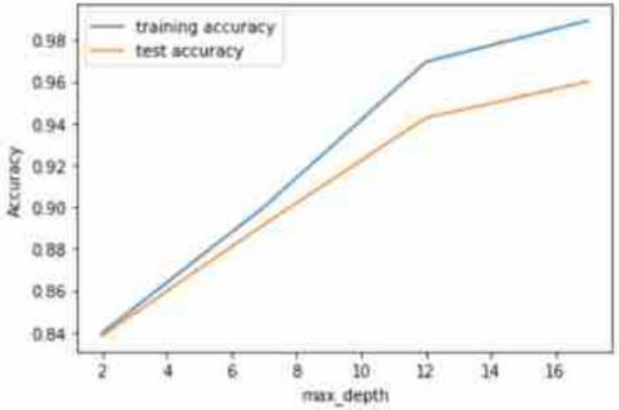
```
Accuracy Score for Decision Tree Final Model : 0.8585126286248831
```

Conclusion from the Decision Tree model

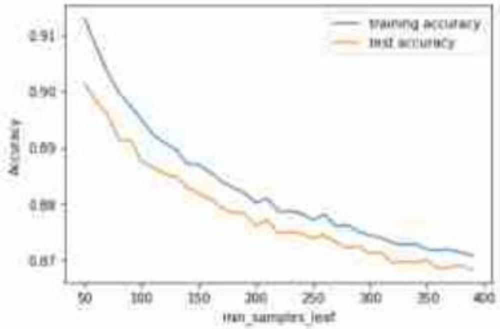
- 85% accuracy on the test dataset
- lots of false positives in the confusion matrix

MODEL 5: Random Forest

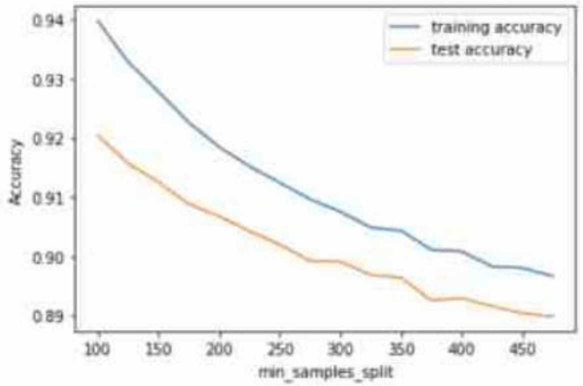
In [ ]:
```
# plotting accuracies with max_depth
plt.figure()
plt.plot(scores["param_max_depth"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_max_depth"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



In [ ]:
```
# scores of GridSearch CV
scores = rf.cv_results_

# plotting accuracies with min_samples_leaf
plt.figure()
plt.plot(scores["param_min_samples_leaf"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_min_samples_leaf"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("min_samples_leaf")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



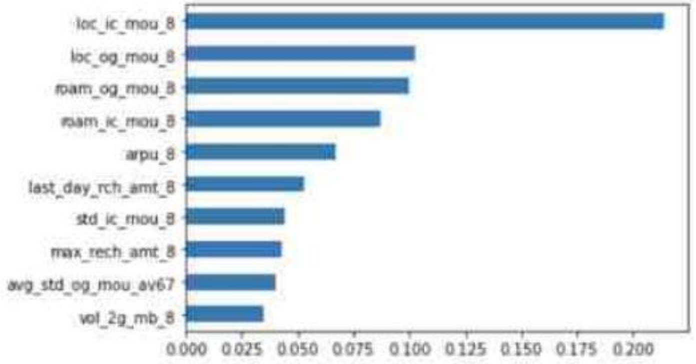
In [ ]:
```
# scores of GridSearch CV
scores = rf.cv_results_

# plotting accuracies with min_samples_split
plt.figure()
plt.plot(scores["param_min_samples_split"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_min_samples_split"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("min_samples_split")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



In [129…

```
# list of important features
X = df
features = X.columns.values
X = pd.DataFrame(scaler.transform(X))
X.columns = features

importances = model_rf.feature_importances_
weights = pd.Series(importances,
                 index=X.columns.values)
weights.sort_values()[-10:].plot(kind = 'barh')
```

Out[129]: <AxesSubplot:>



Conclusions from Random Forest

The top 3 features to predict churn are:

1. Local Incoming for Month 8
2. Average Revenue Per Customer for Month 8
3. Max Recharge Amount for Month 8

Overall Conclusion:

1. The final model is Random Forest as it gives the best accuracy & prediction on unseen data.
2. Std Outgoing Calls and Revenue Per Customer are strong indicators of Churn.
3. Local Incoming and Outgoing Calls for 8th Month and avg revenue in 8th Month are the most important columns to predict churn.
4. customers with tenure less than 4 yrs are more likely to churn.
5. Max Recharge Amount is a strong feature to predict churn.
6. Random Forest produced the best prediction results followed by SVM.