ActivityBalanceChurnDataset

May 23, 2022

[26]:
```
import pandas as pd
```

[27]:
```
data = pd.read_csv('https://raw.githubusercontent.com/fenago/datasets/main/
↪churn.csv',sep=',')
```

[28]:
```
data.sample(5)
```

[28]:

	churn	accountlength	internationalplan	voicemailplan	\
3021	No	57	no	no	
535	No	194	no	no	
3941	No	113	no	no	
3275	No	120	no	yes	
752	No	22	no	yes	

	numbervmailmessages	totaldayminutes	totaldaycalls	totaldaycharge	\
3021	0	85.9	92	14.60	
535	0	48.4	101	8.23	
3941	0	193.1	93	32.83	
3275	27	128.5	115	21.85	
752	23	182.1	94	30.96	

	totaleveminutes	totalevecalls	totalevecharge	totalnightminutes	\
3021	193.9	127	16.48	231.5	
535	281.1	138	23.89	218.5	
3941	206.4	85	17.54	215.9	
3275	163.7	91	13.91	242.9	
752	164.6	59	13.99	128.8	

	totalnightcalls	totalnightcharge	totalintlminutes	totalintlcalls	\
3021	93	10.42	10.1	2	
535	87	9.83	18.2	1	
3941	102	9.72	11.1	2	
3275	121	10.93	0.0	0	
752	102	5.80	12.7	4	

	totalintlcharge	numbercustomerservicecalls
3021	2.73	0

```
535              4.91                          1
3941             3.00                          1
3275             0.00                          1
752              3.43                          3
```

[29]:
```
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
```

[30]:
```
data.columns
```

[30]:
```
Index(['churn', 'accountlength', 'internationalplan', 'voicemailplan',
       'numbervmailmessages', 'totaldayminutes', 'totaldaycalls',
       'totaldaycharge', 'totaleveminutes', 'totalevecalls', 'totalevecharge',
       'totalnightminutes', 'totalnightcalls', 'totalnightcharge',
       'totalintlminutes', 'totalintlcalls', 'totalintlcharge',
       'numbercustomerservicecalls'],
      dtype='object')
```

[31]:
```
data1 = data.drop(['churn', 'internationalplan', 'voicemailplan'], axis=1)
```

[32]:
```
data1.sample(5)
```

[32]:

	accountlength	numbervmailmessages	totaldayminutes	totaldaycalls	\
4671	67	0	166.6	102	
2636	104	0	200.2	92	
4931	129	23	210.7	119	
3446	118	0	256.5	115	
4623	74	0	207.1	79	

	totaldaycharge	totaleveminutes	totalevecalls	totalevecharge	\
4671	28.32	226.3	110	19.24	
2636	34.03	118.7	87	10.09	
4931	35.82	137.4	51	11.68	
3446	43.61	135.3	79	11.50	
4623	35.21	182.0	100	15.47	

	totalnightminutes	totalnightcalls	totalnightcharge	totalintlminutes	\
4671	147.2	121	6.62	9.1	
2636	236.6	65	10.65	6.0	
4931	248.2	92	11.17	7.9	
3446	208.3	131	9.37	7.5	
4623	233.7	73	10.52	7.4	

	totalintlcalls	totalintlcharge	numbercustomerservicecalls
4671	6	2.46	3
2636	6	1.62	2
4931	6	2.13	2

```
3446                5              2.03                           0
4623                8              2.00                           1
```

[34]:
```
# Converting each of the columns to scaled version
for x in data1.columns:
    data1[x] = min_max_scaler.fit_transform(data1[x].values.reshape(-1,1))
```

[35]:
```
data1.sample(5)
```

[35]:
```
      accountlength  numbervmailmessages  totaldayminutes  totaldaycalls  \
2721       0.202479             0.000000         0.840114       0.769697
1499       0.351240             0.000000         0.388051       0.630303
4239       0.446281             0.769231         0.283642       0.703030
2967       0.611570             0.384615         0.420484       0.800000
4333       0.685950             0.000000         0.371266       0.400000

      totaldaycharge  totaleveminutes  totalevecalls  totalevecharge  \
2721        0.840027         0.350289       0.588235        0.350372
1499        0.388052         0.556778       0.647059        0.556778
4239        0.283635         0.385757       0.688235        0.385959
2967        0.420515         0.761067       0.552941        0.761242
4333        0.371319         0.222436       0.600000        0.222582

      totalnightminutes  totalnightcalls  totalnightcharge  totalintlminutes  \
2721           0.422278         0.600000          0.422622             0.480
1499           0.584051         0.491429          0.584131             0.575
4239           0.560253         0.434286          0.560495             0.695
2967           0.379494         0.628571          0.379854             0.510
4333           0.561013         0.662857          0.561058             0.560

      totalintlcalls  totalintlcharge  numbercustomerservicecalls
2721            0.30         0.479630                    0.111111
1499            0.05         0.575926                    0.333333
4239            0.25         0.694444                    0.333333
2967            0.30         0.509259                    0.000000
4333            0.25         0.559259                    0.111111
```

[37]:
```
data2 = pd.get_dummies(data[['internationalplan', 'voicemailplan']])
```

[38]:
```
data2.head()
```

[38]:
```
   internationalplan_no  internationalplan_yes  voicemailplan_no  \
0                     1                      0                 0
1                     1                      0                 0
2                     1                      0                 1
3                     0                      1                 1
4                     0                      1                 1
```

```
   voicemailplan_yes
0                  1
1                  1
2                  0
3                  0
4                  0
```

[41]:
```
X = pd.concat([data1, data2], axis=1)
```

[43]:
```
X.head()
```

[43]:
```
   accountlength  numbervmailmessages  totaldayminutes  totaldaycalls  \
0       0.524793             0.480769         0.754196       0.666667
1       0.438017             0.500000         0.459744       0.745455
2       0.561983             0.000000         0.692461       0.690909
3       0.342975             0.000000         0.851778       0.430303
4       0.305785             0.000000         0.474253       0.684848

   totaldaycharge  totaleveminutes  totalevecalls  totalevecharge  \
0        0.754183         0.542755       0.582353        0.542866
1        0.459672         0.537531       0.605882        0.537690
2        0.692436         0.333242       0.647059        0.333225
3        0.851740         0.170195       0.517647        0.170171
4        0.474230         0.407754       0.717647        0.407959

   totalnightminutes  totalnightcalls  totalnightcharge  totalintlminutes  \
0           0.619494         0.520000          0.619584             0.500
1           0.644051         0.588571          0.644344             0.685
2           0.411646         0.594286          0.411930             0.610
3           0.498481         0.508571          0.498593             0.330
4           0.473165         0.691429          0.473270             0.505

   totalintlcalls  totalintlcharge  numbercustomerservicecalls  \
0            0.15         0.500000                    0.111111
1            0.15         0.685185                    0.111111
2            0.25         0.609259                    0.000000
3            0.35         0.329630                    0.222222
4            0.15         0.505556                    0.333333

   internationalplan_no  internationalplan_yes  voicemailplan_no  \
0                     1                      0                 0
1                     1                      0                 0
2                     1                      0                 1
3                     0                      1                 1
4                     0                      1                 1
```

```
   voicemailplan_yes
0                  1
1                  1
2                  0
3                  0
4                  0
```

[48]:
```
Y = data['churn']
Y.head()
# print(Y['churn'])
```

[48]:
```
0    No
1    No
2    No
3    No
4    No
Name: churn, dtype: object
```

[49]:
```
Y=Y.apply(lambda x: 1 if x=='Yes' else 0)
```

[50]:
```
print(Y)
```

```
0       0
1       0
2       0
3       0
4       0
       ..
4995    0
4996    1
4997    0
4998    0
4999    0
Name: churn, Length: 5000, dtype: int64
```

[51]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,␣
 ↪stratify=Y, random_state=123)
# Defining the LogisticRegression function
churnModel = LogisticRegression()
churnModel.fit(X_train, y_train)
```

[51]:
```
LogisticRegression()
```

[52]:
```
pred = churnModel.predict(X_test)
print('Accuracy of Logistic regression model prediction on test set: {:.2f}'.
 ↪format(churnModel.score(X_test, y_test)))
```

```
Accuracy of Logistic regression model prediction on test set: 0.86
```

[53]:
```
# Confusion Matrix for the model
from sklearn.metrics import confusion_matrix
confusionMatrix = confusion_matrix(y_test, pred)
print(confusionMatrix)
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
[[1254   34]
 [ 176   36]]
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      1288
           1       0.51      0.17      0.26       212

    accuracy                           0.86      1500
   macro avg       0.70      0.57      0.59      1500
weighted avg       0.83      0.86      0.83      1500
```

[56]:
```
print('Percentage of churn = yes :',(y_train[y_train==1].value_counts()/
 ↪len(y_train) ) * 100)
print('Percentage of churn = no :',(y_train[y_train==0].value_counts()/
 ↪len(y_train) ) * 100)
```

```
Percentage of churn = yes : 1    14.142857
Name: churn, dtype: float64
Percentage of churn = no : 0    85.857143
Name: churn, dtype: float64
```

[57]:
```
from sklearn.model_selection import train_test_split
# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,␣
 ↪random_state=123)
```

[58]:
```
trainData = pd.concat([X_train,y_train],axis=1)
trainData.head()
```

[58]:

	accountlength	numbervmailmessages	totaldayminutes	totaldaycalls	\
4036	0.256198	0.500000	0.609388	0.484848	
2883	0.504132	0.000000	0.595733	0.296970	
4162	0.012397	0.000000	0.482788	0.581818	

```
4640       0.450413             0.000000         0.714936       0.551515
2430       0.491736             0.769231         0.364438       0.600000

      totaldaycharge  totaleveminutes  totalevecalls  totalevecharge  \
4036        0.609270         0.695628       0.894118        0.695891
2883        0.595716         0.652736       0.688235        0.652863
4162        0.482764         0.362387       0.552941        0.362342
4640        0.714859         0.569700       0.558824        0.569719
2430        0.364458         0.681056       0.458824        0.681009

      totalnightminutes  totalnightcalls  totalnightcharge  totalintlminutes  \
4036           0.395949         0.622857          0.396173             0.515
2883           0.605570         0.560000          0.605515             0.490
4162           0.620506         0.491429          0.620709             0.710
4640           0.630380         0.400000          0.630838             0.645
2430           0.505570         0.691429          0.505909             0.780

      totalintlcalls  totalintlcharge  numbercustomerservicecalls  \
4036            0.10         0.514815                    0.222222
2883            0.55         0.490741                    0.111111
4162            0.20         0.709259                    0.000000
4640            0.10         0.644444                    0.111111
2430            0.15         0.779630                    0.000000

      internationalplan_no  internationalplan_yes  voicemailplan_no  \
4036                     1                      0                 0
2883                     1                      0                 1
4162                     0                      1                 1
4640                     1                      0                 1
2430                     1                      0                 0

      voicemailplan_yes  churn
4036                  1      0
2883                  0      0
4162                  0      1
4640                  0      1
2430                  1      0
```

[60]:
```
ind = trainData[trainData['churn']==1].index
print(len(ind))
```

```
490
```

[61]:
```
# Seperate the minority classes
minData = trainData.loc[ind]
print(minData.shape)
```

```
(490, 20)
```

[62]:
```
ind1 = trainData[trainData['churn']==0].index
print(len(ind1))
```

```
3010
```

[63]:
```
majData = trainData.loc[ind1]
print(majData.shape)
majData.head()
```

```
(3010, 20)
```

[63]:

	accountlength	numbervmailmessages	totaldayminutes	totaldaycalls	\
4036	0.256198	0.500000	0.609388	0.484848	
2883	0.504132	0.000000	0.595733	0.296970	
2430	0.491736	0.769231	0.364438	0.600000	
449	0.322314	0.403846	0.751920	0.478788	
4179	0.578512	0.000000	0.613940	0.478788	

	totaldaycharge	totaleveminutes	totalevecalls	totalevecharge	\
4036	0.609270	0.695628	0.894118	0.695891	
2883	0.595716	0.652736	0.688235	0.652863	
2430	0.364458	0.681056	0.458824	0.681009	
449	0.751841	0.557602	0.694118	0.557748	
4179	0.613956	0.309871	0.500000	0.309932	

	totalnightminutes	totalnightcalls	totalnightcharge	totalintlminutes	\
4036	0.395949	0.622857	0.396173	0.515	
2883	0.605570	0.560000	0.605515	0.490	
2430	0.505570	0.691429	0.505909	0.780	
449	0.438987	0.525714	0.438942	0.315	
4179	0.561519	0.622857	0.561621	0.280	

	totalintlcalls	totalintlcharge	numbercustomerservicecalls	\
4036	0.10	0.514815	0.222222	
2883	0.55	0.490741	0.111111	
2430	0.15	0.779630	0.000000	
449	0.15	0.314815	0.444444	
4179	0.20	0.279630	0.222222	

	internationalplan_no	internationalplan_yes	voicemailplan_no	\
4036	1	0	0	
2883	1	0	1	
2430	1	0	0	
449	1	0	0	
4179	1	0	1	

	voicemailplan_yes	churn

```
4036                 1      0
2883                 0      0
2430                 1      0
449                  1      0
4179                 0      0
```

[64]:
```
majSample = majData.sample(n=len(ind),random_state = 123)
```

[65]:
```
print(majSample.shape)
majSample.head()
```

```
(490, 20)
```

[65]:

	accountlength	numbervmailmessages	totaldayminutes	totaldaycalls	\
1807	0.450413	0.000000	0.557895	0.624242	
4578	0.475207	0.000000	0.244097	0.533333	
355	0.123967	0.000000	0.472546	0.636364	
23	0.454545	0.000000	0.314083	0.624242	
1541	0.194215	0.692308	0.656899	0.557576	

	totaldaycharge	totaleveminutes	totalevecalls	totalevecharge	\
1807	0.557898	0.549079	0.723529	0.549013	
4578	0.244143	0.318394	0.658824	0.318344	
355	0.472557	0.218037	0.547059	0.218052	
23	0.314090	0.377509	0.600000	0.377548	
1541	0.656794	0.460819	0.711765	0.461016	

	totalnightminutes	totalnightcalls	totalnightcharge	totalintlminutes	\
1807	0.344051	0.405714	0.344401	0.645	
4578	0.495949	0.520000	0.496342	0.550	
355	0.541013	0.560000	0.541362	0.635	
23	0.480000	0.600000	0.480023	0.385	
1541	0.683544	0.497143	0.683737	0.380	

	totalintlcalls	totalintlcharge	numbercustomerservicecalls	\
1807	0.05	0.644444	0.333333	
4578	0.10	0.550000	0.111111	
355	0.10	0.635185	0.111111	
23	0.30	0.385185	0.222222	
1541	0.20	0.379630	0.333333	

	internationalplan_no	internationalplan_yes	voicemailplan_no	\
1807	1	0	1	
4578	1	0	1	
355	1	0	1	
23	1	0	1	
1541	1	0	0	

```
      voicemailplan_yes  churn
1807                  0      0
4578                  0      0
355                   0      0
23                    0      0
1541                  1      0
```

[66]:
```
# Concatinating both data sets and then shuffling the data set
balData = pd.concat([minData,majSample],axis = 0)
print('balanced data set shape',balData.shape)
# Shuffling the data set
from sklearn.utils import shuffle
balData = shuffle(balData)
balData.head()
```

```
balanced data set shape (980, 20)
```

[66]:
```
      accountlength  numbervmailmessages  totaldayminutes  totaldaycalls  \
4378       0.483471             0.000000         0.520910       0.775758
4030       0.438017             0.000000         0.741394       0.490909
2474       0.326446             0.423077         0.558748       0.696970
3373       0.797521             0.000000         0.734851       0.460606
2987       0.537190             0.000000         0.347937       0.503030

      totaldaycharge  totaleveminutes  totalevecalls  totalevecharge  \
4378        0.520917         0.662634       0.547059        0.662892
4030        0.741299         0.676657       0.682353        0.676804
2474        0.558735         0.413253       0.641176        0.413458
3373        0.734772         0.841353       0.517647        0.841475
2987        0.347892         0.326643       0.552941        0.326755

      totalnightminutes  totalnightcalls  totalnightcharge  totalintlminutes  \
4378           0.534430         0.617143          0.534609             0.625
4030           0.615949         0.668571          0.616207             0.640
2474           0.446076         0.428571          0.446258             0.465
3373           0.612152         0.605714          0.612268             0.680
2987           0.374430         0.542857          0.374789             0.685

      totalintlcalls  totalintlcharge  numbercustomerservicecalls  \
4378            0.20         0.625926                    0.000000
4030            0.45         0.640741                    0.000000
2474            0.05         0.464815                    0.000000
3373            0.20         0.679630                    0.111111
2987            0.15         0.685185                    0.333333

      internationalplan_no  internationalplan_yes  voicemailplan_no  \
```

```
4378                    1                    0                    1
4030                    1                    0                    1
2474                    1                    0                    0
3373                    1                    0                    1
2987                    0                    1                    1

      voicemailplan_yes  churn
4378                  0      0
4030                  0      1
2474                  1      0
3373                  0      1
2987                  0      1
```

[68]:
```
balData.shape
```

[68]:
```
(980, 20)
```

[72]:
```
# Making the new X_train and y_train
X_trainNew = balData.iloc[:,0:19]
X_trainNew.head()
y_trainNew = balData['churn']
y_trainNew.head()
```

[72]:
```
4378    0
4030    1
2474    0
3373    1
2987    1
Name: churn, dtype: int64
```

[74]:
```
# Defining the LogisticRegression function
churnModel2 = LogisticRegression()
churnModel2.fit(X_trainNew, y_trainNew)
# Predicting on the test
pred = churnModel2.predict(X_test)
print('Accuracy of Logistic regression model prediction on test set for␣
 ↪balanced data set: {:.2f}'.format(churnModel2.score(X_test, y_test)))
```

```
Accuracy of Logistic regression model prediction on test set for balanced data
set: 0.79
```

[75]:
```
# Confusion Matrix for the model
from sklearn.metrics import confusion_matrix
confusionMatrix = confusion_matrix(y_test, pred)
print(confusionMatrix)
```

```
[[1032  251]
 [  57  160]]
```

[77]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.80      0.87      1283
           1       0.39      0.74      0.51       217

    accuracy                           0.79      1500
   macro avg       0.67      0.77      0.69      1500
weighted avg       0.87      0.79      0.82      1500
```

[78]:
```
#SMOTE
# Splitting the data into train and test sets
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=0)
print("Before OverSampling count of yes: {}".format(sum(y_train==1)))
print("Before OverSampling count of no: {} \n".format(sum(y_train==0)))
```

```
Before OverSampling count of yes: 505
Before OverSampling count of no: 2995
```

[79]:
```
import smote_variants as sv
import numpy as np
# Instantiating the SMOTE class
oversampler= sv.SMOTE()
```

[80]:
```
# Creating new training set
X_train_us, y_train_us = oversampler.sample(np.array(X_train), np.
 ↪array(y_train))
```

```
2022-05-23 14:53:01,528:INFO:SMOTE: Running sampling via ('SMOTE',
"{'proportion': 1.0, 'n_neighbors': 5, 'n_jobs': 1, 'random_state': None}")
```

[82]:
```
# Shape after oversampling
print('After OverSampling, the shape of train_X: {}'.format(X_train_us.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_us.
 ↪shape))
print("After OverSampling, counts of label 'Yes': {}".
 ↪format(sum(y_train_us==1)))
print("After OverSampling, counts of label 'No': {}".format(sum(y_train_us==0)))
```

```
After OverSampling, the shape of train_X: (5990, 19)
After OverSampling, the shape of train_y: (5990,)
```

```
After OverSampling, counts of label 'Yes': 2995
After OverSampling, counts of label 'No': 2995
```

[84]:
```
# Training the model with Logistic regression model
# Defining the LogisticRegression function
churnModel3 = LogisticRegression()
churnModel3.fit(X_train_us, y_train_us)
# Predicting on the test set
pred = churnModel3.predict(X_test)
# Printing accuracy
print('Accuracy of Logistic regression model prediction on test set for Smote␣
 ↪balanced data set: {:.2f}'.format(churnModel3.score(X_test, y_test)))
# Confusion Matrix for the model
from sklearn.metrics import confusion_matrix
confusionMatrix = confusion_matrix(y_test, pred)
print(confusionMatrix)
# Classification report for the model
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
Accuracy of Logistic regression model prediction on test set for Smote balanced
data set: 0.76
[[984 314]
 [ 45 157]]
              precision    recall  f1-score   support

           0       0.96      0.76      0.85      1298
           1       0.33      0.78      0.47       202

    accuracy                           0.76      1500
   macro avg       0.64      0.77      0.66      1500
weighted avg       0.87      0.76      0.79      1500
```

[ ]:
```
# Best Method: 1st Logistic Regression Model (ChurnModel)
# If the goal is to identiy "Yes" the first would be the most accurate at 51%␣
 ↪precision
# If we are aiming for general accuracy, the first would also be the most␣
 ↪accurate
```