[1]:

**import numpy as np import pandas as pd**

**import matplotlib.pyplot as plt import seaborn as sns**

# Credit Card Prediction Analysis

app = pd.read\_csv("../input/credit-card-approval-prediction/application\_record.

↪csv")

crecord = pd.read\_csv("../input/credit-card-approval-prediction/credit\_record.

↪csv")

[3]:

[4]:

* + Using different methods to understand data
  + data is complex and both dataset need some kind of transformation before analysis
  + datasets are indivudally dealt with and then eventually compiled using joins

app.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 438557 entries, 0 to 438556 Data columns (total 18 columns):

# Column Non-Null Count Dtype

|  |  |  |  |
| --- | --- | --- | --- |
| 0 ID | 438557 | non-null | int64 |
| 1 CODE\_GENDER | 438557 | non-null | object |
| 2 FLAG\_OWN\_CAR | 438557 | non-null | object |
| 3 FLAG\_OWN\_REALTY | 438557 | non-null | object |
| 4 CNT\_CHILDREN | 438557 | non-null | int64 |
| 5 AMT\_INCOME\_TOTAL | 438557 | non-null | float64 |
| 6 NAME\_INCOME\_TYPE | 438557 | non-null | object |
| 7 NAME\_EDUCATION\_TYPE | 438557 | non-null | object |
| 8 NAME\_FAMILY\_STATUS | 438557 | non-null | object |
| 9 NAME\_HOUSING\_TYPE | 438557 | non-null | object |
| 10 DAYS\_BIRTH | 438557 | non-null | int64 |
| 11 DAYS\_EMPLOYED | 438557 | non-null | int64 |
| 12 FLAG\_MOBIL | 438557 | non-null | int64 |
| 13 FLAG\_WORK\_PHONE | 438557 | non-null | int64 |
| 14 FLAG\_PHONE | 438557 | non-null | int64 |
| 15 FLAG\_EMAIL | 438557 | non-null | int64 |
| 16 OCCUPATION\_TYPE | 304354 | non-null | object |
| 17 CNT\_FAM\_MEMBERS | 438557 | non-null | float64 |

dtypes: float64(2), int64(8), object(8) memory usage: 60.2+ MB

crecord.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 1048575 entries, 0 to 1048574 Data columns (total 3 columns):

# Column Non-Null Count Dtype

1. ID 1048575 non-null int64
2. MONTHS\_BALANCE 1048575 non-null int64

[5]:

1. STATUS 1048575 non-null object dtypes: int64(2), object(1)

memory usage: 24.0+ MB

app['ID'].nunique() *# the total rows are 438,557. This means it has duplicates*

[5]: 438510

[6]:

crecord['ID'].nunique()

*# this has around 43,000 unique rows as there are repeating entries for*␣

↪*different monthly values and status.*

[6]: 45985

[7]:

len(set(crecord['ID']).intersection(set(app['ID']))) *# checking to see how many*␣

↪*records match in two datasets*

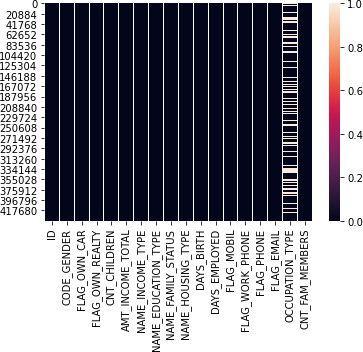
[7]: 36457

[8]:

sns.heatmap(app.isnull()) *# checking for null values. Seems like*␣

↪*occupation\_type has many*

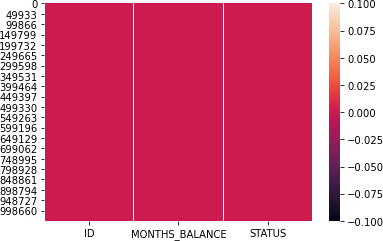
* 1. : <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb19d9592d0>



[9]:

sns.heatmap(crecord.isnull()) *# checking for null values. All good here!*

* 1. : <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb19d743110>



[10]:

app = app.drop\_duplicates('ID', keep='last')

*# we identified that there are some duplicates in this dataset*

*# we will be deleting those duplicates and will keep the last entry of the ID*␣

↪*if its repeated.*

[11]:

app.drop('OCCUPATION\_TYPE', axis=1, inplace=**True**)

*#we identified earlier that occupation\_type has many missing values*

*# we will drop this column*

[12]:

ot = pd.DataFrame(app.dtypes =='object').reset\_index() object\_type = ot[ot[0] == **True**]['index']

object\_type

*#we are filtering the columns that have non numeric values to see if they are*␣

↪*useful*

[12]: 1 CODE\_GENDER

1. FLAG\_OWN\_CAR
2. FLAG\_OWN\_REALTY
3. NAME\_INCOME\_TYPE
4. NAME\_EDUCATION\_TYPE
5. NAME\_FAMILY\_STATUS
6. NAME\_HOUSING\_TYPE Name: index, dtype: object

[13]:

num\_type = pd.DataFrame(app.dtypes != 'object').reset\_index().rename(columns = ␣

↪{0:'yes/no'})

num\_type = num\_type[num\_type['yes/no'] ==**True**]['index']

*#HAVE CREATED SEPARATE LIST FOR NUMERIC TYPE INCASE IT WILL BE NEEDED IN*␣

↪*FURTHER ANALYSIS*

*# IT IS NEEDED IN FURTHER ANALYSIS*

[14]:

a = app[object\_type]['CODE\_GENDER'].value\_counts() b = app[object\_type]['FLAG\_OWN\_CAR'].value\_counts()

c = app[object\_type]['FLAG\_OWN\_REALTY'].value\_counts() d = app[object\_type]['NAME\_INCOME\_TYPE'].value\_counts()

e = app[object\_type]['NAME\_EDUCATION\_TYPE'].value\_counts() f = app[object\_type]['NAME\_FAMILY\_STATUS'].value\_counts() g = app[object\_type]['NAME\_HOUSING\_TYPE'].value\_counts()

print( a,"**\n**",b,'**\n**', c, '**\n**', d, '**\n**', e, '**\n**', f, '**\n**', g)

*#this is just to see what each column is.*

*#It seems that all of them are important since there is very fine classifcation*␣

↪*in each column.*

*# their effectiveness cannot be judged at this moment so we convert all of them*␣

↪*to numeric values.*

F 294412

M 144098

Name: CODE\_GENDER, dtype: int64 N 275428

Y 163082

Name: FLAG\_OWN\_CAR, dtype: int64 Y 304043

N 134467

Name: FLAG\_OWN\_REALTY, dtype: int64 Working 226087

Commercial associate 100739

Pensioner 75483

State servant 36184

Student 17

Name: NAME\_INCOME\_TYPE, dtype: int64 Secondary / secondary special 301789

|  |  |
| --- | --- |
| Higher education | 117509 |
| Incomplete higher | 14849 |
| Lower secondary | 4051 |
| Academic degree 312  Name: NAME\_EDUCATION\_TYPE, dtype: int64 | |
| Married | 299798 |
| Single / not married | 55268 |
| Civil marriage | 36524 |

Separated 27249

Widow 19671

Name: NAME\_FAMILY\_STATUS, dtype: int64

|  |  |
| --- | --- |
| House / apartment | 393788 |
| With parents | 19074 |
| Municipal apartment | 14213 |
| Rented apartment | 5974 |
| Office apartment | 3922 |
| Co-op apartment | 1539 |

Name: NAME\_HOUSING\_TYPE, dtype: int64

[15]:

**from sklearn.preprocessing import** LabelEncoder le = LabelEncoder()

**for** x **in** app:

**if** app[x].dtypes=='object':

app[x] = le.fit\_transform(app[x])

*# we have transformed all the non numeric data columns into data columns*

*# this method applies 0,1.. classification to different value types.*

[16]:

app.head(10)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [16]: | ID | CODE\_GENDER | FLAG\_OWN\_CAR | FLAG\_OWN\_REALTY | CNT\_CHILDREN | \ |
|  | 0 5008804 | 1 | 1 | 1 | 0 |  |
|  | 1 5008805 | 1 | 1 | 1 | 0 |  |
|  | 2 5008806 | 1 | 1 | 1 | 0 |  |
|  | 3 5008808 | 0 | 0 | 1 | 0 |  |
|  | 4 5008809 | 0 | 0 | 1 | 0 |  |
|  | 5 5008810 | 0 | 0 | 1 | 0 |  |
|  | 6 5008811 | 0 | 0 | 1 | 0 |  |
|  | 7 5008812 | 0 | 0 | 1 | 0 |  |
|  | 8 5008813 | 0 | 0 | 1 | 0 |  |
|  | 9 5008814 | 0 | 0 | 1 | 0 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| AMT\_INCOME\_TOTAL | NAME\_INCOME\_TYPE | NAME\_EDUCATION\_TYPE | \ |
| 0 427500.0 | 4 | 1 |  |
| 1 427500.0 | 4 | 1 |  |
| 2 112500.0 | 4 | 4 |  |
| 3 270000.0 | 0 | 4 |  |
| 4 270000.0 | 0 | 4 |  |
| 5 270000.0 | 0 | 4 |  |
| 6 270000.0 | 0 | 4 |  |
| 7 283500.0 | 1 | 1 |  |
| 8 283500.0 | 1 | 1 |  |
| 9 283500.0 | 1 | 1 |  |

NAME\_FAMILY\_STATUS NAME\_HOUSING\_TYPE DAYS\_BIRTH DAYS\_EMPLOYED \ 0 0 4 -12005 -4542

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 0 | 4 | -12005 | -4542 |
| 2 | 1 | 1 | -21474 | -1134 |
| 3 | 3 | 1 | -19110 | -3051 |
| 4 | 3 | 1 | -19110 | -3051 |
| 5 | 3 | 1 | -19110 | -3051 |
| 6 | 3 | 1 | -19110 | -3051 |
| 7 | 2 | 1 | -22464 | 365243 |
| 8 | 2 | 1 | -22464 | 365243 |
| 9 | 2 | 1 | -22464 | 365243 |
| FLAG\_MOBIL | FLAG\_WORK\_PHONE | FLAG\_PHONE | FLAG\_EMAIL | CNT\_FAM\_MEMBERS |
| 0 1 | 1 | 0 | 0 | 2.0 |
| 1 1 | 1 | 0 | 0 | 2.0 |
| 2 1 | 0 | 0 | 0 | 2.0 |
| 3 1 | 0 | 1 | 1 | 1.0 |
| 4 1 | 0 | 1 | 1 | 1.0 |
| 5 1 | 0 | 1 | 1 | 1.0 |
| 6 1 | 0 | 1 | 1 | 1.0 |
| 7 1 | 0 | 0 | 0 | 1.0 |
| 8 1 | 0 | 0 | 0 | 1.0 |
| 9 1 | 0 | 0 | 0 | 1.0 |

[17]:

app[num\_type].head()

*# We will look at numeric columns and see if there is anything that needs to be*␣

↪*changed.*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [17]: | ID | | CNT\_CHILDREN | AMT\_INCOME\_TOTAL | | DAYS\_BIRTH | DAYS\_EMPLOYED \ |
|  | 0 5008804 | | 0 | 427500.0 | | -12005 | -4542 |
|  | 1 5008805 | | 0 | 427500.0 | | -12005 | -4542 |
|  | 2 5008806 | | 0 | 112500.0 | | -21474 | -1134 |
|  | 3 5008808 | | 0 | 270000.0 | | -19110 | -3051 |
|  | 4 5008809 | | 0 | 270000.0 | | -19110 | -3051 |
|  | | FLAG\_MOBIL FLAG\_WORK\_PHONE | | | FLAG\_PHONE | FLAG\_EMAIL | CNT\_FAM\_MEMBERS |
| 0 | | 1 1 | | | 0 | 0 | 2.0 |
| 1 | | 1 1 | | | 0 | 0 | 2.0 |
| 2 | | 1 0 | | | 0 | 0 | 2.0 |
| 3 | | 1 0 | | | 1 | 1 | 1.0 |
| 4 | | 1 0 | | | 1 | 1 | 1.0 |

[18]:

fig, ax= plt.subplots(nrows= 3, ncols = 3, figsize= (14,6))

sns.scatterplot(x='ID', y='CNT\_CHILDREN', data=app, ax=ax[0][0], color=␣

↪'orange')

sns.scatterplot(x='ID', y='AMT\_INCOME\_TOTAL', data=app, ax=ax[0][1],␣

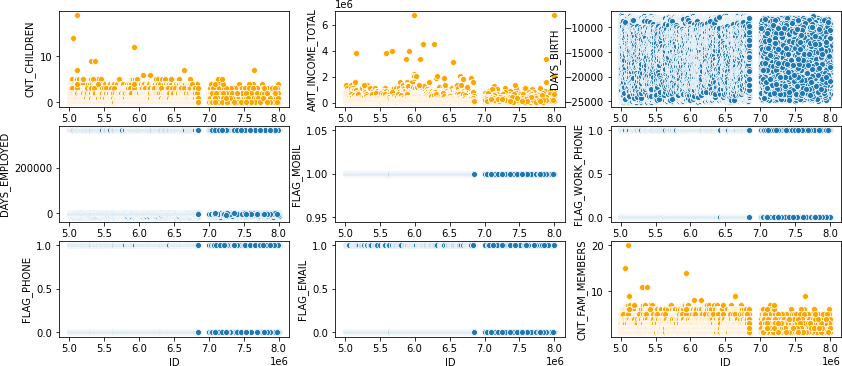
↪color='orange')

sns.scatterplot(x='ID', y='DAYS\_BIRTH', data=app, ax=ax[0][2])

sns.scatterplot(x='ID', y='DAYS\_EMPLOYED', data=app, ax=ax[1][0]) sns.scatterplot(x='ID', y='FLAG\_MOBIL', data=app, ax=ax[1][1]) sns.scatterplot(x='ID', y='FLAG\_WORK\_PHONE', data=app, ax=ax[1][2]) sns.scatterplot(x='ID', y='FLAG\_PHONE', data=app, ax=ax[2][0]) sns.scatterplot(x='ID', y='FLAG\_EMAIL', data=app, ax=ax[2][1]) sns.scatterplot(x='ID', y='CNT\_FAM\_MEMBERS', data=app, ax=ax[2][2], color=␣

↪'orange')

1. : <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb19754c610>



[19]:

There are outliers in 3 columns. 1. CNT\_CHILDREN 2. AMT\_INCOME\_TOTAL 3. CNT\_FAM\_MEMBERS

* + We need to remove these outliers to make sure they do not affect our model results.
  + We will now remove these outliers.

*# FOR CNT\_CHILDREN COLUMN*

q\_hi = app['CNT\_CHILDREN'].quantile(0.999) q\_low = app['CNT\_CHILDREN'].quantile(0.001)

app = app[(app['CNT\_CHILDREN']>q\_low) & (app['CNT\_CHILDREN']<q\_hi)]

[20]:

*# FOR AMT\_INCOME\_TOTAL COLUMN*

q\_hi = app['AMT\_INCOME\_TOTAL'].quantile(0.999) q\_low = app['AMT\_INCOME\_TOTAL'].quantile(0.001)

app= app[(app['AMT\_INCOME\_TOTAL']>q\_low) & (app['AMT\_INCOME\_TOTAL']<q\_hi)]

[21]:

*#FOR CNT\_FAM\_MEMBERS COLUMN*

q\_hi = app['CNT\_FAM\_MEMBERS'].quantile(0.999) q\_low = app['CNT\_FAM\_MEMBERS'].quantile(0.001)

app= app[(app['CNT\_FAM\_MEMBERS']>q\_low) & (app['CNT\_FAM\_MEMBERS']<q\_hi)]

[22]:

fig, ax= plt.subplots(nrows= 3, ncols = 3, figsize= (14,6))

sns.scatterplot(x='ID', y='CNT\_CHILDREN', data=app, ax=ax[0][0], color=␣

↪'orange')

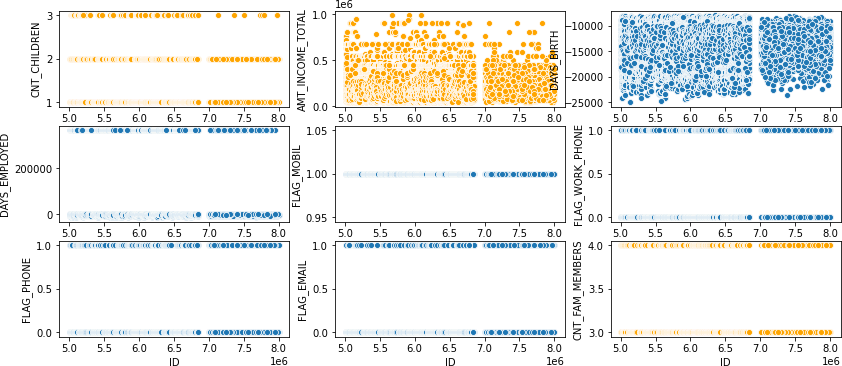
sns.scatterplot(x='ID', y='AMT\_INCOME\_TOTAL', data=app, ax=ax[0][1],␣

↪color='orange')

sns.scatterplot(x='ID', y='DAYS\_BIRTH', data=app, ax=ax[0][2]) sns.scatterplot(x='ID', y='DAYS\_EMPLOYED', data=app, ax=ax[1][0]) sns.scatterplot(x='ID', y='FLAG\_MOBIL', data=app, ax=ax[1][1]) sns.scatterplot(x='ID', y='FLAG\_WORK\_PHONE', data=app, ax=ax[1][2]) sns.scatterplot(x='ID', y='FLAG\_PHONE', data=app, ax=ax[2][0]) sns.scatterplot(x='ID', y='FLAG\_EMAIL', data=app, ax=ax[2][1]) sns.scatterplot(x='ID', y='CNT\_FAM\_MEMBERS', data=app, ax=ax[2][2], color=␣

↪'orange')

[22]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb19378f750>



[23]:

crecord['Months from today'] = crecord['MONTHS\_BALANCE']\*-1

crecord = crecord.sort\_values(['ID','Months from today'], ascending=**True**) crecord.head(10)

*# we calculated months from today column to see how much old is the month*

*# we also sort the data according to ID and Months from today columns.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [23]: | ID | MONTHS\_BALANCE | STATUS | Months from today |
|  | 0 5001711 | 0 | X | 0 |
|  | 1 5001711 | -1 | 0 | 1 |
|  | 2 5001711 | -2 | 0 | 2 |
|  | 3 5001711 | -3 | 0 | 3 |
|  | 4 5001712 | 0 | C | 0 |
|  | 5 5001712 | -1 | C | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 5001712 | -2 | C | 2 |
| 7 5001712 | -3 | C | 3 |
| 8 5001712 | -4 | C | 4 |
| 9 5001712 | -5 | C | 5 |

[24]:

crecord['STATUS'].value\_counts()

*# performed a value count on status to see how many values exist of each type*

[24]: C 442031

0 383120

X 209230

1 11090

5 1693

2 868

3 320

4 223

Name: STATUS, dtype: int64

[25]:

crecord['STATUS'].replace({'C': 0, 'X' : 0}, inplace=**True**) crecord['STATUS'] = crecord['STATUS'].astype('int')

crecord['STATUS'] = crecord['STATUS'].apply(**lambda** x:1 **if** x >= 2 **else** 0)

*# replace the value C and X with 0 as it is the same type*

*# 1,2,3,4,5 are classified as 1 because they are the same type # these will be our labels/prediction results for our model*

[26]:

crecord['STATUS'].value\_counts(normalize=**True**)

*# there is a problem here*

*# the data is oversampled for the labels # 0 are 99%*

*# 1 are only 1% in the whole dataset*

*# we will need to address the oversampling issue in order to make sense of our*␣

↪*analysis*

*# this will be done after when we combine both the datasets # so first we will join the datasets*

[26]: 0 0.99704

1 0.00296

Name: STATUS, dtype: float64

[27]:

crecordgb = crecord.groupby('ID').agg(max).reset\_index() crecordgb.head()

*#we are grouping the data in crecord by ID so that we can join it with app*

1. : ID MONTHS\_BALANCE STATUS Months from today

|  |  |  |  |
| --- | --- | --- | --- |
| 0 5001711 | 0 | 0 | 3 |
| 1 5001712 | 0 | 0 | 18 |
| 2 5001713 | 0 | 0 | 21 |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 5001714 | 0 | 0 | 14 |
| 4 5001715 | 0 | 0 | 59 |

[28]:

df = app.join(crecordgb.set\_index('ID'), on='ID', how='inner') df.drop(['Months from today', 'MONTHS\_BALANCE'], axis=1, inplace=**True**) df.head()

*# no that this is joined, we will solve over sampling issue*

1. : ID CODE\_GENDER FLAG\_OWN\_CAR FLAG\_OWN\_REALTY CNT\_CHILDREN \

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 29 5008838 | 1 | 0 | 1 | 1 |
| 30 5008839 | 1 | 0 | 1 | 1 |
| 31 5008840 | 1 | 0 | 1 | 1 |
| 32 5008841 | 1 | 0 | 1 | 1 |
| 33 5008842 | 1 | 0 | 1 | 1 |

AMT\_INCOME\_TOTAL NAME\_INCOME\_TYPE NAME\_EDUCATION\_TYPE \ 29 405000.0 0 1

30 405000.0 0 1

31 405000.0 0 1

32 405000.0 0 1

33 405000.0 0 1

NAME\_FAMILY\_STATUS NAME\_HOUSING\_TYPE DAYS\_BIRTH DAYS\_EMPLOYED \

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | 1 | 1 | -11842 | -2016 | |
| 30 | 1 | 1 | -11842 | -2016 | |
| 31 | 1 | 1 | -11842 | -2016 | |
| 32 | 1 | 1 | -11842 | -2016 | |
| 33 | 1 | 1 | -11842 | -2016 | |
| FLAG\_MOBIL | FLAG\_WORK\_PHONE | FLAG\_PHONE | FLAG\_EMAIL | CNT\_FAM\_MEMBERS | \ |
| 29 1 | 0 | 0 | 0 | 3.0 |  |
| 30 1 | 0 | 0 | 0 | 3.0 |  |
| 31 1 | 0 | 0 | 0 | 3.0 |  |
| 32 1 | 0 | 0 | 0 | 3.0 |  |
| 33 1 | 0 | 0 | 0 | 3.0 |  |

|  |  |
| --- | --- |
|  | STATUS |
| 29 | 0 |
| 30 | 0 |
| 31 | 0 |
| 32 | 0 |
| 33 | 0 |

df.info() # checking for number of rows. # there are 9516 rows.

[29]:

df.info() *# checking for number of rows. # there are 9516 rows.*

<class 'pandas.core.frame.DataFrame'> Int64Index: 9516 entries, 29 to 434805 Data columns (total 18 columns):

# Column Non-Null Count Dtype

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 ID | 9516 | non-null |  | int64 |
| 1 CODE\_GENDER | 9516 | non-null |  | int64 |
| 2 FLAG\_OWN\_CAR | 9516 | non-null |  | int64 |
| 3 FLAG\_OWN\_REALTY | 9516 | non-null |  | int64 |
| 4 CNT\_CHILDREN | 9516 | non-null |  | int64 |
| 5 AMT\_INCOME\_TOTAL | 9516 | non-null |  | float64 |
| 6 NAME\_INCOME\_TYPE | 9516 | non-null |  | int64 |
| 7 NAME\_EDUCATION\_TYPE | 9516 | non-null |  | int64 |
| 8 NAME\_FAMILY\_STATUS | 9516 | non-null |  | int64 |
| 9 NAME\_HOUSING\_TYPE | 9516 | non-null |  | int64 |
| 10 DAYS\_BIRTH | 9516 | non-null |  | int64 |
| 11 DAYS\_EMPLOYED | 9516 | non-null |  | int64 |
| 12 FLAG\_MOBIL | 9516 | non-null |  | int64 |
| 13 FLAG\_WORK\_PHONE | 9516 | non-null |  | int64 |
| 14 FLAG\_PHONE | 9516 | non-null |  | int64 |
| 15 FLAG\_EMAIL | 9516 | non-null |  | int64 |
| 16 CNT\_FAM\_MEMBERS | 9516 | non-null |  | float64 |
| 17 STATUS | 9516 | non-null |  | int64 |

[30]:

dtypes: float64(2), int64(16) memory usage: 1.4 MB

[31]:

X = df.iloc[:,1:-1] *# X value contains all the variables except labels*

y = df.iloc[:,-1] *# these are the labels*

**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)

*# we create the test train split first*

[32]:

**from sklearn.preprocessing import** MinMaxScaler mms = MinMaxScaler()

X\_scaled = pd.DataFrame(mms.fit\_transform(X\_train), columns=X\_train.columns) X\_test\_scaled = pd.DataFrame(mms.transform(X\_test), columns=X\_test.columns)

*# we have now fit and transform the data into a scaler for accurate reading and*␣

↪*results.*

[33]:

**from imblearn.over\_sampling import** SMOTE oversample = SMOTE()

X\_balanced, y\_balanced = oversample.fit\_resample(X\_scaled, y\_train) X\_test\_balanced, y\_test\_balanced = oversample.fit\_resample(X\_test\_scaled,␣

↪y\_test)

*# we have addressed the issue of oversampling here*

[34]:

y\_train.value\_counts()

[34]: 0 6562

1 99

Name: STATUS, dtype: int64

[35]:

y\_balanced.value\_counts()

[35]: 1 6562

0 6562

Name: STATUS, dtype: int64

[36]:

y\_test.value\_counts()

[36]: 0 2803

1 52

Name: STATUS, dtype: int64

[37]:

y\_test\_balanced.value\_counts()

[37]: 1 2803

0 2803

Name: STATUS, dtype: int64

* We notice in the value counts above that label types are now balanced
* the problem of oversampling is solved now
* we will now implement different models to see which one performs the best

[38]:

**from sklearn.linear\_model import** LogisticRegression **from sklearn.neighbors import** KNeighborsClassifier **from sklearn.svm import** SVC

**from sklearn.tree import** DecisionTreeClassifier **from sklearn.ensemble import** RandomForestClassifier **from xgboost import** XGBClassifier

[39]:

classifiers = {

"LogisticRegression" : LogisticRegression(), "KNeighbors" : KNeighborsClassifier(),

"SVC" : SVC(),

"DecisionTree" : DecisionTreeClassifier(), "RandomForest" : RandomForestClassifier(), "XGBoost" : XGBClassifier()

}

[40]:

train\_scores = [] test\_scores = []

**for** key, classifier **in** classifiers.items(): classifier.fit(X\_balanced, y\_balanced)

train\_score = classifier.score(X\_balanced, y\_balanced) train\_scores.append(train\_score)

test\_score = classifier.score(X\_test\_balanced, y\_test\_balanced) test\_scores.append(test\_score)

print(train\_scores) print(test\_scores)

[41]:

[0.6156659555013715, 0.9849131362389515, 0.9400335263639135, 0.9951234379762267,

0.9951234379762267, 0.9950472416946053]

[0.5651088119871566, 0.7320727791651802, 0.7549054584373885, 0.8241170174812701,

0.7684623617552622, 0.8662147698894042]

* We found out that XGBoost model is performing best on the train set as well as test set with 91% accuracy
* We will be using XGBoost to predict our values.

xgb = XGBClassifier()

model = xgb.fit(X\_balanced, y\_balanced) prediction = xgb.predict(X\_test\_balanced)

[42]:

**from sklearn.metrics import** classification\_report print(classification\_report(y\_test\_balanced, prediction))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.79 | 0.99 | 0.88 | 2803 |
| 1 | 0.99 | 0.74 | 0.85 | 2803 |
| accuracy |  |  | 0.87 | 5606 |
| macro avg | 0.89 | 0.87 | 0.86 | 5606 |
| weighted avg | 0.89 | 0.87 | 0.86 | 5606 |