credit-card-predictive-analysis

June 11, 2024

`Project-3 Credit Card Prediction Analysis Made by : Priyanshu Kumar`

1 Credit Card Prediction Analysis!

2 Context

Credit score cards are a common risk control method in the financial industry. It uses personal information and data submitted by credit card applicants to predict the probability of future defaults and credit card borrowings. The bank is able to decide whether to issue a credit card to the applicant. Credit scores can objectively quantify the magnitude of risk.

Generally speaking, credit score cards are based on historical data. Once encountering large economic fluctuations. Past models may lose their original predictive power. Logistic model is a common method for credit scoring. Because Logistic is suitable for binary classification tasks and can calculate the coefficients of each feature. In order to facilitate understanding and operation, the score card will multiply the logistic regression coefficient by a certain value (such as 100) and round it.

At present, with the development of machine learning algorithms. More predictive methods such as Boosting, Random Forest, and Support Vector Machines have been introduced into credit card scoring. However, these methods often do not have good transparency. It may be difficult to provide customers and regulators with a reason for rejection or acceptance.

3 Task

Build a machine learning model to predict if an applicant is 'good' or 'bad' client, different from other tasks, the definition of 'good' or 'bad' is not given. You should use some techique, such as vintage analysis to construct you label. Also, unbalance data problem is a big problem in this task.

4 Importing Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

5 Extracting data using two data sources

[2]:
```python
app = pd.read_csv("../input/credit-card-approval-prediction/application_record.
↪csv")
crecord = pd.read_csv("../input/credit-card-approval-prediction/credit_record.
↪csv")
```

- 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

[3]:
```python
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
```

[4]:
```python
crecord.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 3 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   ID              1048575 non-null  int64
 1   MONTHS_BALANCE  1048575 non-null  int64
 2   STATUS          1048575 non-null  object
```

```
dtypes: int64(2), object(1)
memory usage: 24.0+ MB
```

[5]:
```
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.
```

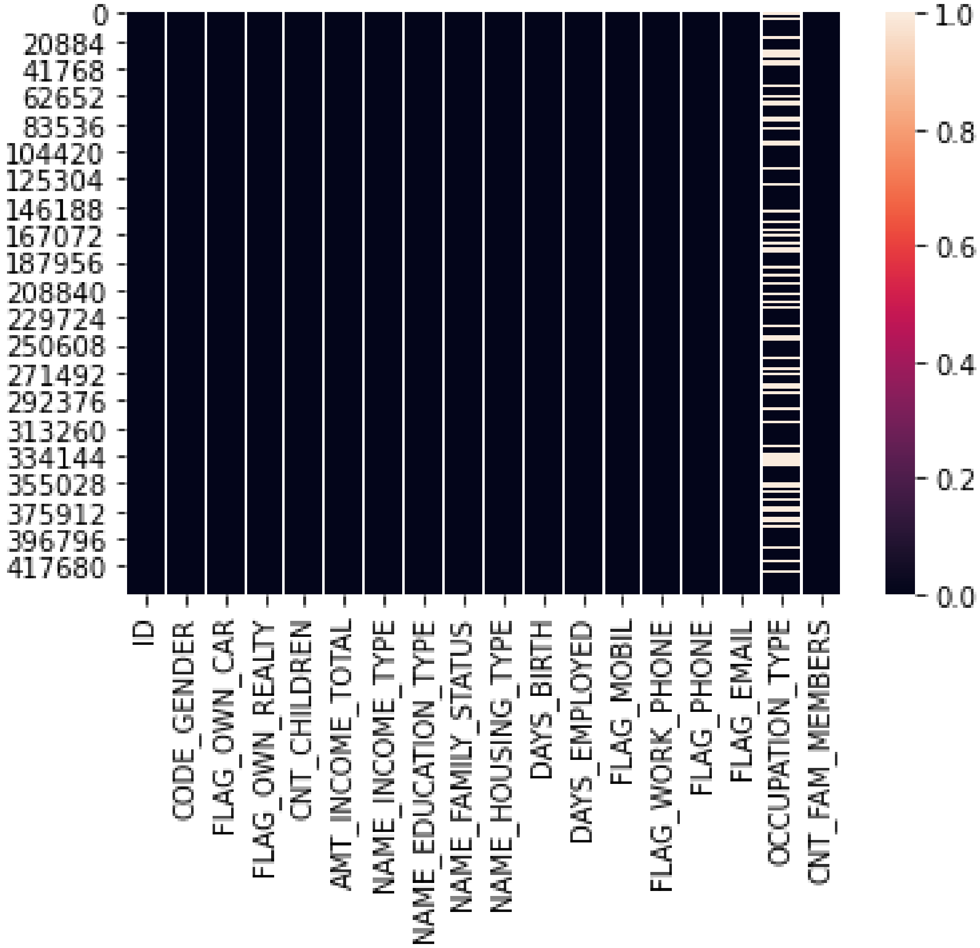
[6]:
```
45985
```

[7]:
```
len(set(crecord['ID']).intersection(set(app['ID']))) # checking to see how many␣
 ↪records match in two datasets
```

[7]:
```
36457
```

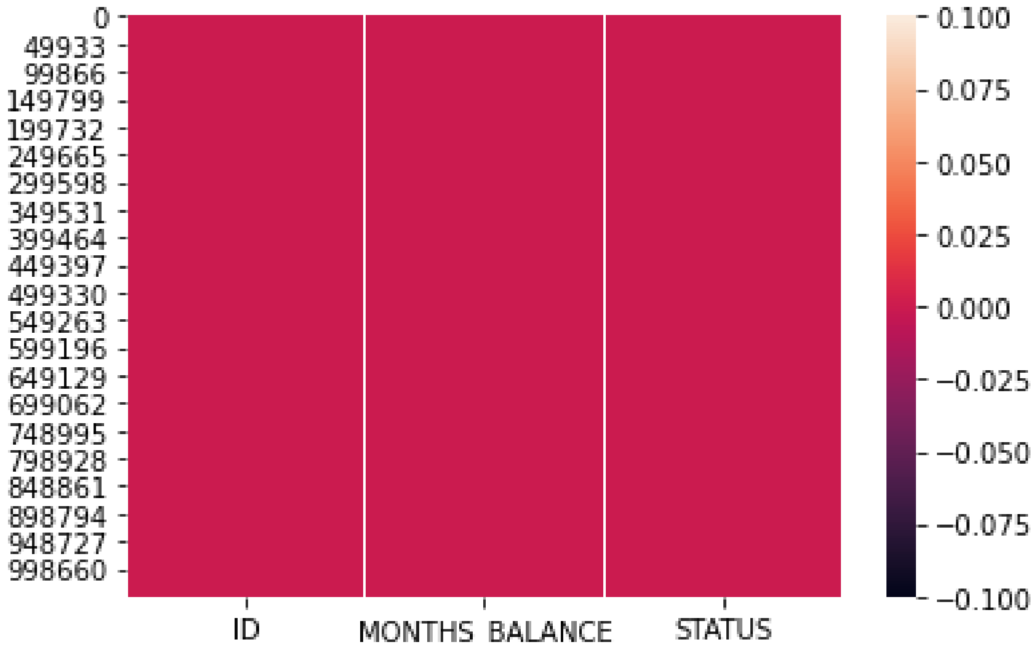
[8]:
```
sns.heatmap(app.isnull()) # checking for null values. Seems like␣
 ↪occupation_type has many
```

[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb19d9592d0>
```



[9]:
```
sns.heatmap(crecord.isnull()) # checking for null values. All good here!
```

[9]: <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
2            FLAG_OWN_CAR
3         FLAG_OWN_REALTY
6        NAME_INCOME_TYPE
7     NAME_EDUCATION_TYPE
8      NAME_FAMILY_STATUS
9       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]:
```python
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]:
```python
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
```

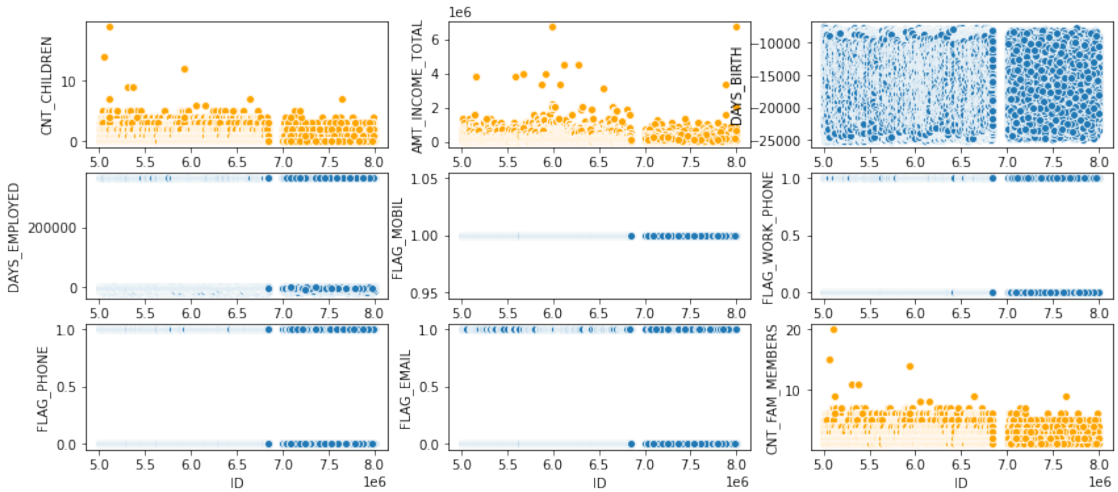
[17]:
```
app[num_type].head()
# We will look at numeric columns and see if there is anything that needs to be␣
↪changed.
```

[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')
```

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb19754c610>
```



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.

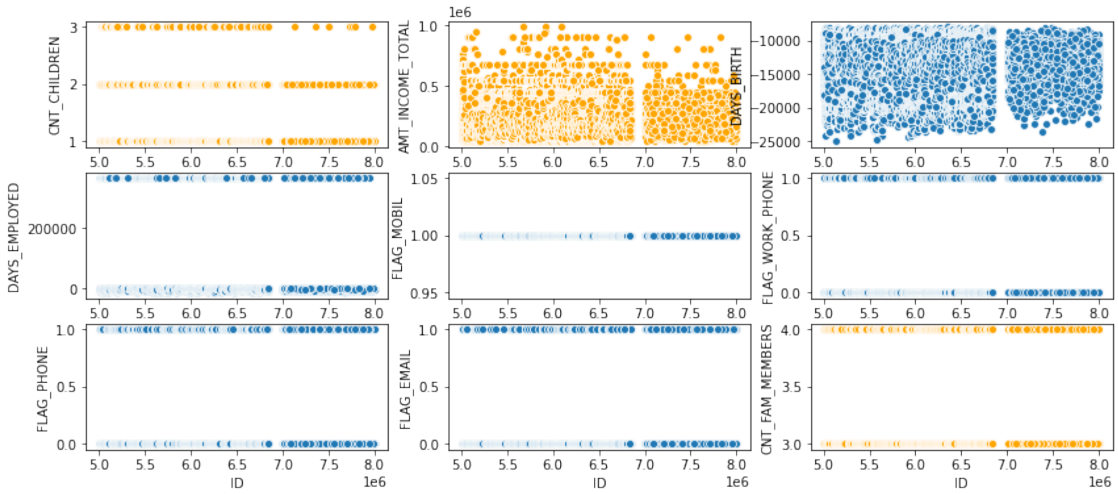
[19]:
```
# FOR CNT_CHILDREN COLUMN
q_hi = app['CNT_CHILDREN'].quantile(0.999)
q_low = app['CNT_CHILDREN'].quantile(0.001)
app = app[(app['CNT_CHILDREN']>q_low) & (app['CNT_CHILDREN']<q_hi)]
```

[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]:
```python
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]:
```python
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 columms.
```

[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
```

[27]:

	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]:
```python
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
```

[28]:
```
         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]:
```python
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
dtypes: float64(2), int64(16)
memory usage: 1.4 MB
```

[30]:
```
X = df.iloc[:,1:-1] # X value contains all the variables except labels
y = df.iloc[:,-1] # these are the labels
```

[31]:
```
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)
```

```
[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.

[41]:
```
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
```