Loading Data

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df = pd.read_csv("application_record.csv")
df.head()
```

Out[2]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INC
0	5008804	M	Y	Y	0	
1	5008805	M	Y	Y	0	
2	5008806	M	Y	Y	0	
3	5008808	F	N	Y	0	
4	5008809	F	N	Y	0	

In [3]:

```
df.shape
```

Out[3]:

(438557, 18)

In [4]:

```
df.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
```

In [5]:

```
credit_df = pd.read_csv("credit_record.csv")
credit_df.head()
```

Out[5]:

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C

In [6]:

```
credit_df.shape
```

Out[6]:

```
(1048575, 3)
```

In [7]:

```
credit_df.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
```

Exploratory Data Analysis (EDA)

On File - Application Record.csv

In [8]:

```
df.describe()
```

Out[8]:

	ID	CNT_CHILDREN	AMT_INCOME_TOTAL	DAYS_BIRTH	DAYS_EMPLOYED
count	4.385570e+05	438557.000000	4.385570e+05	438557.000000	438557.000000
mean	6.022176e+06	0.427390	1.875243e+05	-15997.904649	60563.675328
std	5.716370e+05	0.724882	1.100869e+05	4185.030007	138767.799647
min	5.008804e+06	0.000000	2.610000e+04	-25201.000000	-17531.000000
25%	5.609375e+06	0.000000	1.215000e+05	-19483.000000	-3103.000000
50%	6.047745e+06	0.000000	1.607805e+05	-15630.000000	-1467.000000
75%	6.456971e+06	1.000000	2.250000e+05	-12514.000000	-371.000000
max	7.999952e+06	19.000000	6.750000e+06	-7489.000000	365243.000000

In [9]:

```
df.isnull().sum()
```

Out[9]:

```
ID                      0
CODE_GENDER             0
FLAG_OWN_CAR            0
FLAG_OWN_REALTY         0
CNT_CHILDREN            0
AMT_INCOME_TOTAL        0
NAME_INCOME_TYPE        0
NAME_EDUCATION_TYPE     0
NAME_FAMILY_STATUS      0
NAME_HOUSING_TYPE       0
DAYS_BIRTH              0
DAYS_EMPLOYED           0
FLAG_MOBIL              0
FLAG_WORK_PHONE         0
FLAG_PHONE              0
FLAG_EMAIL              0
OCCUPATION_TYPE    134203
CNT_FAM_MEMBERS         0
dtype: int64
```

In [10]:

```
# dropping occupation type which has many null values
df.drop('OCCUPATION_TYPE', axis=1, inplace=True)
```

In [11]:

```
# Checking duplicates in 'ID' column
len(df['ID']) - len(df['ID'].unique())
```

Out[11]:

```
47
```

In [12]:

```
# Dropping duplicate entries from ID column
df = df.drop_duplicates('ID', keep='last')
```

In [13]:

```
# Checking Non-Numerical Columns
cat_columns = df.columns[(df.dtypes =='object').values].tolist()
cat_columns
```

Out[13]:

```
['CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE']
```

In [14]:

```
# Checking Numerical Columns
df.columns[(df.dtypes !='object').values].tolist()
```

Out[14]:

```
['ID',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'FLAG_MOBIL',
 'FLAG_WORK_PHONE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'CNT_FAM_MEMBERS']
```

In [15]:

```
# Checking unique values from Categorical Columns

for i in df.columns[(df.dtypes =='object').values].tolist():
    print(i,'\n')
    print(df[i].value_counts())
    print('-----------------------------------------------')
```

```
CODE_GENDER

F    294412
M    144098
Name: CODE_GENDER, dtype: int64
-----------------------------------------------
FLAG_OWN_CAR

N    275428
Y    163082
Name: FLAG_OWN_CAR, dtype: int64
-----------------------------------------------
FLAG_OWN_REALTY

Y    304043
N    134467
Name: FLAG_OWN_REALTY, dtype: int64
-----------------------------------------------
NAME_INCOME_TYPE

Working                 226087
Commercial associate    100739
Pensioner                75483
State servant            36184
Student                     17
Name: NAME_INCOME_TYPE, dtype: int64
-----------------------------------------------
NAME_EDUCATION_TYPE

Secondary / secondary special    301789
Higher education                 117509
Incomplete higher                 14849
Lower secondary                    4051
Academic degree                     312
Name: NAME_EDUCATION_TYPE, dtype: int64
-----------------------------------------------
NAME_FAMILY_STATUS

Married                 299798
Single / not married     55268
Civil marriage           36524
Separated                27249
Widow                    19671
Name: NAME_FAMILY_STATUS, dtype: int64
-----------------------------------------------
NAME_HOUSING_TYPE

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
-----------------------------------------------
```

In [16]:

```python
# Checking unique values from Numerical Columns
df['CNT_CHILDREN'].value_counts()
```

Out[16]:

```
0     304038
1      88518
2      39879
3       5430
4        486
5        133
7          9
9          5
6          4
12         4
14         3
19         1
Name: CNT_CHILDREN, dtype: int64
```

In [17]:

```python
# Checking Min , Max values from 'DAYS_BIRTH' column
print('Min DAYS_BIRTH :', df['DAYS_BIRTH'].min(),'\nMax DAYS_BIRTH :', df['DAYS_BIRTH'].max())
```

```
Min DAYS_BIRTH : -25201
Max DAYS_BIRTH : -7489
```

In [18]:

```python
# Converting 'DAYS_BIRTH' values from Day to Years
df['DAYS_BIRTH'] = round(df['DAYS_BIRTH']/-365,0)
df.rename(columns={'DAYS_BIRTH':'AGE_YEARS'}, inplace=True)
# Checking unique values greater than 0
df[df['DAYS_EMPLOYED']>0]['DAYS_EMPLOYED'].unique()
```

Out[18]:

```
array([365243])
```

In [19]:

```python
# As mentioned in document, if 'DAYS_EMPLOYED' is positive no, it means person currently unemployed,
df['DAYS_EMPLOYED'].replace(365243, 0, inplace=True)
```

In [20]:

```python
# Converting 'DAYS_EMPLOYED' values from Day to Years
df['DAYS_EMPLOYED'] = abs(round(df['DAYS_EMPLOYED']/-365,0))
df.rename(columns={'DAYS_EMPLOYED':'YEARS_EMPLOYED'}, inplace=True)
```

In [21]:

```
df['FLAG_MOBIL'].value_counts()
```

Out[21]:

```
1    438510
Name: FLAG_MOBIL, dtype: int64
```

In [22]:

```
# As all the values in column are 1, hence dropping column
df.drop('FLAG_MOBIL', axis=1, inplace=True)
```

In [23]:

```
df['FLAG_WORK_PHONE'].value_counts()
```

Out[23]:

```
0    348118
1     90392
Name: FLAG_WORK_PHONE, dtype: int64
```

In [24]:

```
# This column only contains 0 & 1 values for Mobile no submitted, hence dropping column
df.drop('FLAG_WORK_PHONE', axis=1, inplace=True)
```

In [25]:

```
df['FLAG_PHONE'].value_counts()
```

Out[25]:

```
0    312323
1    126187
Name: FLAG_PHONE, dtype: int64
```

In [26]:

```
# This column only contains 0 & 1 values for Phone no submitted, hence dropping column
df.drop('FLAG_PHONE', axis=1, inplace=True)
```

In [27]:

```
df['FLAG_EMAIL'].value_counts()
```

Out[27]:

```
0    391062
1     47448
Name: FLAG_EMAIL, dtype: int64
```

In [28]:

```
# This column only contains 0 & 1 values for Email submitted, hence dropping column
df.drop('FLAG_EMAIL', axis=1, inplace=True)
```

In [29]:

```
df['CNT_FAM_MEMBERS'].value_counts()
```

Out[29]:

```
2.0     233867
1.0      84483
3.0      77119
4.0      37351
5.0       5081
6.0        459
7.0        124
9.0          9
11.0         5
8.0          4
14.0         4
15.0         3
20.0         1
Name: CNT_FAM_MEMBERS, dtype: int64
```

In [30]:

```
df.head()
```

Out[30]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INC
0	5008804	M	Y	Y	0	
1	5008805	M	Y	Y	0	
2	5008806	M	Y	Y	0	
3	5008808	F	N	Y	0	
4	5008809	F	N	Y	0	

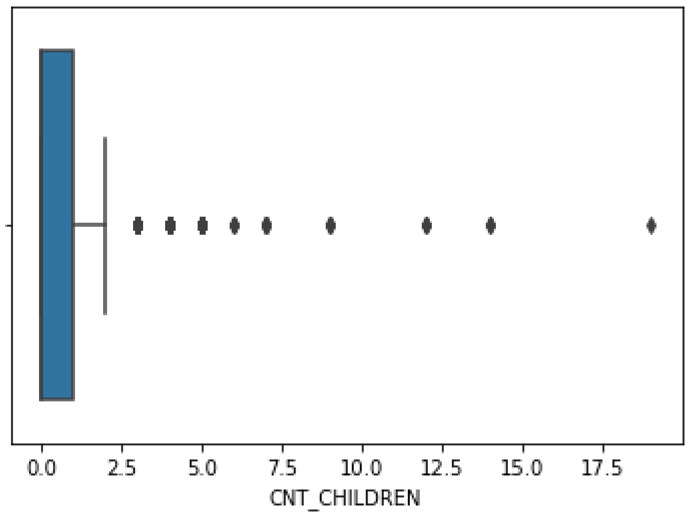
Visualization

In [31]:

```
#create plot to detect outliers
sns.boxplot(df['CNT_CHILDREN'])
```

Out[31]:

<AxesSubplot:xlabel='CNT_CHILDREN'>

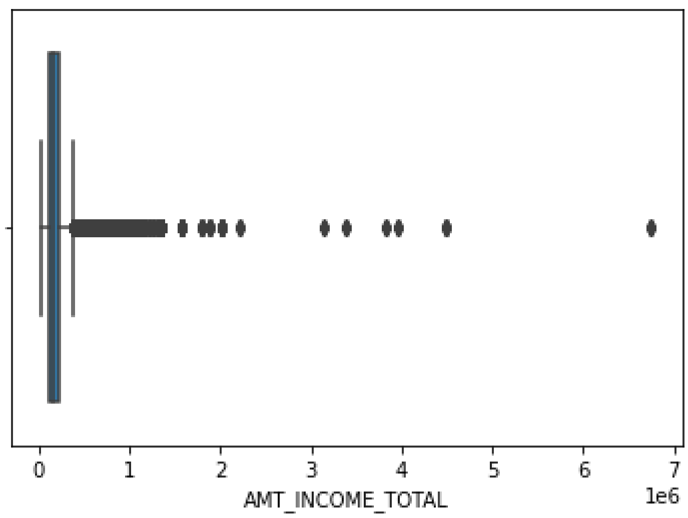


In [32]:

```
sns.boxplot(df['AMT_INCOME_TOTAL'])
```

Out[32]:

<AxesSubplot:xlabel='AMT_INCOME_TOTAL'>

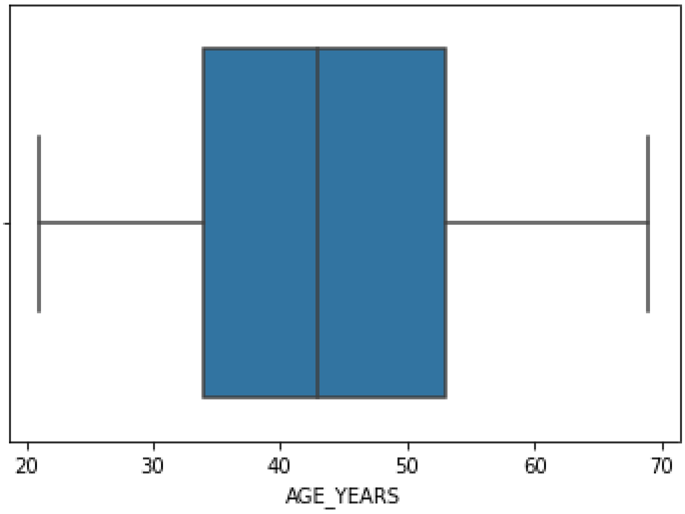


In [33]:

```
sns.boxplot(df['AGE_YEARS'])
```

Out[33]:

<AxesSubplot:xlabel='AGE_YEARS'>

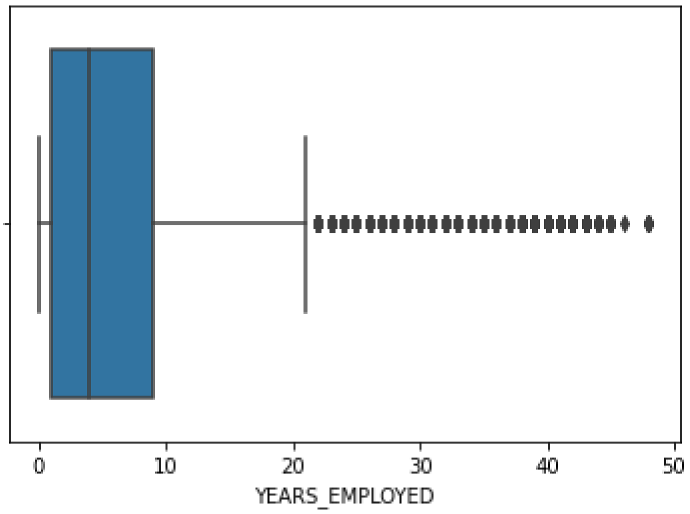


In [34]:

```
sns.boxplot(df['YEARS_EMPLOYED'])
```

Out[34]:

<AxesSubplot:xlabel='YEARS_EMPLOYED'>

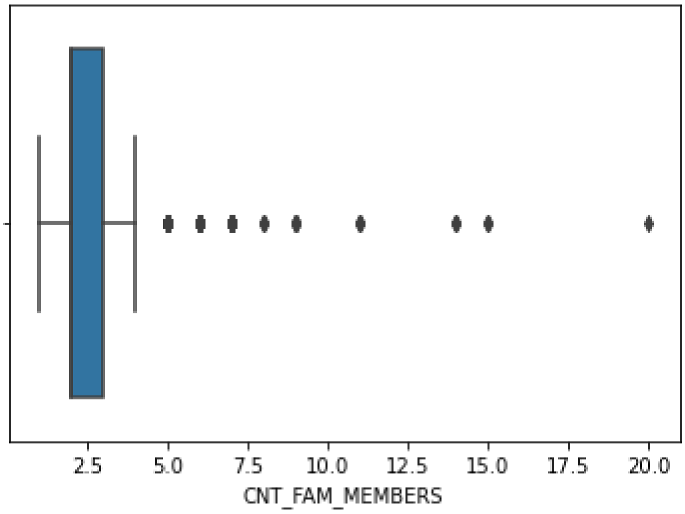


In [35]:

```
sns.boxplot(df['CNT_FAM_MEMBERS'])
```

Out[35]:

```
<AxesSubplot:xlabel='CNT_FAM_MEMBERS'>
```



Data cleansing (Removing Outliers)

In [36]:

```
high_bound = df['CNT_CHILDREN'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = df['CNT_CHILDREN'].quantile(0.001)
print('low_bound :', low_bound)
```

```
high_bound : 4.0
low_bound : 0.0
```

In [37]:

```
df = df[(df['CNT_CHILDREN']>=low_bound) & (df['CNT_CHILDREN']<=high_bound)]
```

In [38]:

```
high_bound = df['AMT_INCOME_TOTAL'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = df['AMT_INCOME_TOTAL'].quantile(0.001)
print('low_bound :', low_bound)
```

```
high_bound : 990000.0
low_bound : 36000.0
```

In [39]:

```
df = df[(df['AMT_INCOME_TOTAL']>=low_bound) & (df['AMT_INCOME_TOTAL']<=high_bound)]
```

In [40]:

```
high_bound = df['YEARS_EMPLOYED'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = df['YEARS_EMPLOYED'].quantile(0.001)
print('low_bound :', low_bound)
```

```
high_bound : 40.0
low_bound : 0.0
```

In [41]:

```
df = df[(df['YEARS_EMPLOYED']>=low_bound) & (df['YEARS_EMPLOYED']<=high_bound)]
```

In [42]:

```
high_bound = df['CNT_FAM_MEMBERS'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = df['CNT_FAM_MEMBERS'].quantile(0.001)
print('low_bound :', low_bound)
```

```
high_bound : 6.0
low_bound : 1.0
```

In [43]:

```
df = df[(df['CNT_FAM_MEMBERS']>=low_bound) & (df['CNT_FAM_MEMBERS']<=high_bound)]
```

In [44]:

```
df.head()
```

Out[44]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INC
0	5008804	M	Y	Y	0	
1	5008805	M	Y	Y	0	
2	5008806	M	Y	Y	0	
3	5008808	F	N	Y	0	
4	5008809	F	N	Y	0	

On File - Credit Record.csv

In [45]:

```
credit_df.head()
```

Out[45]:

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C

In [46]:

```
df.isnull().sum()
```

Out[46]:

```
ID                     0
CODE_GENDER            0
FLAG_OWN_CAR           0
FLAG_OWN_REALTY        0
CNT_CHILDREN           0
AMT_INCOME_TOTAL       0
NAME_INCOME_TYPE       0
NAME_EDUCATION_TYPE    0
NAME_FAMILY_STATUS     0
NAME_HOUSING_TYPE      0
AGE_YEARS              0
YEARS_EMPLOYED         0
CNT_FAM_MEMBERS        0
dtype: int64
```

In [47]:

```
credit_df['STATUS'].value_counts()
```

Out[47]:

```
C    442031
0    383120
X    209230
1     11090
5      1693
2       868
3       320
4       223
Name: STATUS, dtype: int64
```

In [48]:

```
# categorizing 'STATUS' column to binary classification   0 : Good Client and 1 : bad client
credit_df['STATUS'].replace(['C', 'X'],0, inplace=True)
```

In [49]:

```
credit_df['STATUS'].replace(['2','3','4','5'],1, inplace=True)
```

In [50]:

```
credit_df['STATUS'] = credit_df['STATUS'].astype('int')
```

In [51]:

```
credit_df.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  int64
dtypes: int64(3)
memory usage: 24.0 MB
```

In [52]:

```
credit_df['STATUS'].value_counts(normalize=True)*100
```

Out[52]:

```
0    98.646353
1     1.353647
Name: STATUS, dtype: float64
```

In [53]:

```
credit_df_trans = credit_df.groupby('ID').agg(max).reset_index()
```

In [54]:

```
credit_df_trans.drop('MONTHS_BALANCE', axis=1, inplace=True)
credit_df_trans.head()
```

Out[54]:

	ID	STATUS
0	5001711	0
1	5001712	0
2	5001713	0
3	5001714	0
4	5001715	0

In [55]:

```
credit_df_trans['STATUS'].value_counts(normalize=True)*100
```

Out[55]:

```
0    88.365771
1    11.634229
Name: STATUS, dtype: float64
```

Merging Dataframes

In [56]:

```
# merging the two datasets based on 'ID'
data = pd.merge(df, credit_df_trans, on='ID', how='inner')
data.head()
```

Out[56]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INC
0	5008804	M	Y	Y	0	
1	5008805	M	Y	Y	0	
2	5008806	M	Y	Y	0	
3	5008808	F	N	Y	0	
4	5008809	F	N	Y	0	

In [57]:

```
data.shape
```

Out[57]:

```
(36326, 14)
```

In [58]:

```
data.drop('ID', axis=1, inplace=True)
```

In [59]:

```
# checking if there are still duplicate rows in Final Dataframe
len(data) - len(data.drop_duplicates())
```

Out[59]:

```
25268
```

In [60]:

```
# Dropping duplicate records
data = data.drop_duplicates()
data.reset_index(drop=True ,inplace=True)
```

In [61]:

```
data.shape
```

Out[61]:

```
(11058, 13)
```

In [62]:

```
data.isnull().sum()
```

Out[62]:

```
CODE_GENDER            0
FLAG_OWN_CAR           0
FLAG_OWN_REALTY        0
CNT_CHILDREN           0
AMT_INCOME_TOTAL       0
NAME_INCOME_TYPE       0
NAME_EDUCATION_TYPE    0
NAME_FAMILY_STATUS     0
NAME_HOUSING_TYPE      0
AGE_YEARS              0
YEARS_EMPLOYED         0
CNT_FAM_MEMBERS        0
STATUS                 0
dtype: int64
```

In [63]:

```
data['STATUS'].value_counts(normalize=True)*100
```

Out[63]:

```
0    78.513294
1    21.486706
Name: STATUS, dtype: float64
```

Feature Engineering

In [64]:

```
data.head()
```

Out[64]:

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOT/
0	M	Y	Y	0	427500
1	M	Y	Y	0	112500
2	F	N	Y	0	270000
3	F	N	Y	0	283500
4	M	Y	Y	0	270000

In [65]:

```
cat_columns = data.columns[(data.dtypes =='object').values].tolist()
cat_columns
```

Out[65]:

```
['CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE']
```

In [66]:

```
#Converting all Non-Numerical Columns to Numerical
from sklearn.preprocessing import LabelEncoder

for col in cat_columns:
        globals()['LE_{}'.format(col)] = LabelEncoder()
        data[col] = globals()['LE_{}'.format(col)].fit_transform(data[col])
data.head()
```

Out[66]:

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOT
0	1	1	1	0	42750(
1	1	1	1	0	11250(
2	0	0	1	0	27000(
3	0	0	1	0	28350(
4	1	1	1	0	27000(

In [67]:

```
for col in cat_columns:
    print(col , "  : ", globals()['LE_{}'.format(col)].classes_)
```

```
CODE_GENDER   :  ['F' 'M']
FLAG_OWN_CAR   :  ['N' 'Y']
FLAG_OWN_REALTY   :  ['N' 'Y']
NAME_INCOME_TYPE   :  ['Commercial associate' 'Pensioner' 'State servant' 'Student'
 'Working']
NAME_EDUCATION_TYPE   :  ['Academic degree' 'Higher education' 'Incomplete higher'
 'Lower secondary' 'Secondary / secondary special']
NAME_FAMILY_STATUS   :  ['Civil marriage' 'Married' 'Separated' 'Single / not marrie
d' 'Widow']
NAME_HOUSING_TYPE   :  ['Co-op apartment' 'House / apartment' 'Municipal apartment'
 'Office apartment' 'Rented apartment' 'With parents']
```

In [68]:

```
data.corr()
```

Out[68]:

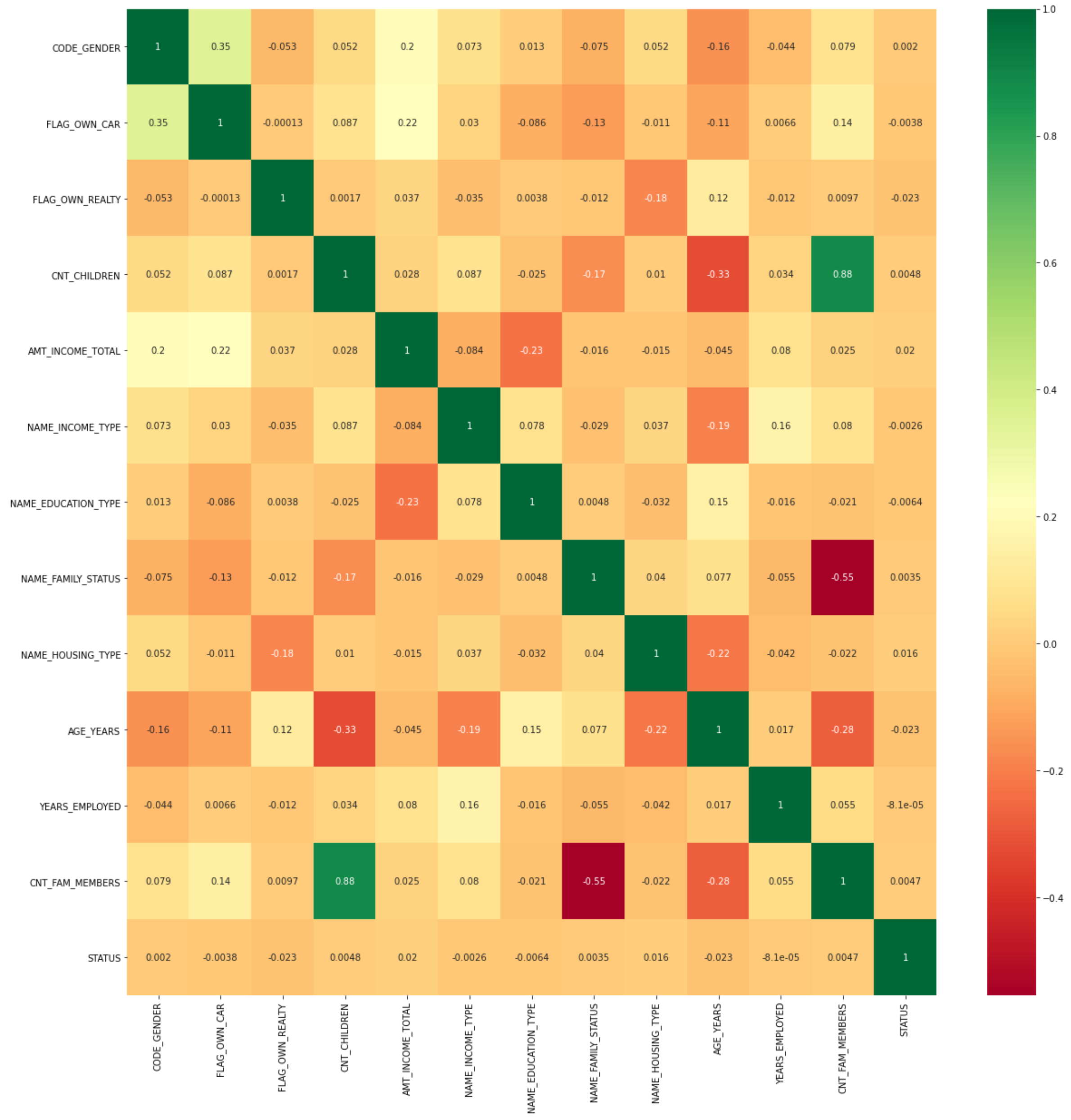
	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDR
CODE_GENDER	1.000000	0.348307	-0.052647	0.052
FLAG_OWN_CAR	0.348307	1.000000	-0.000126	0.087
FLAG_OWN_REALTY	-0.052647	-0.000126	1.000000	0.001
CNT_CHILDREN	0.052351	0.087407	0.001740	1.000
AMT_INCOME_TOTAL	0.199358	0.218026	0.036549	0.027
NAME_INCOME_TYPE	0.072725	0.030008	-0.034673	0.086
NAME_EDUCATION_TYPE	0.013080	-0.085706	0.003771	-0.025
NAME_FAMILY_STATUS	-0.075315	-0.125707	-0.011871	-0.166
NAME_HOUSING_TYPE	0.052185	-0.011203	-0.178070	0.010
AGE_YEARS	-0.157507	-0.106634	0.121351	-0.326
YEARS_EMPLOYED	-0.043657	0.006570	-0.011665	0.033
CNT_FAM_MEMBERS	0.078853	0.138978	0.009698	0.884
STATUS	0.002039	-0.003826	-0.022887	0.004

In [69]:

```
top_corr_features = data.corr().index
plt.figure(figsize=(20, 20))
sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

Out[69]:

<AxesSubplot:>



In [70]:

```
features = data.drop(['STATUS'], axis=1)
label = data['STATUS']
```

In [71]:

```
features.head()
```

Out[71]:

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOT/
0	1	1	1	0	427500
1	1	1	1	0	112500
2	0	0	1	0	270000
3	0	0	1	0	283500
4	1	1	1	0	270000

In [73]:

```
label.head()
```

Out[73]:

```
0    1
1    0
2    0
3    0
4    0
Name: STATUS, dtype: int64
```

Machine Learning Model

In [74]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(features,
                                                    label,
                                                    test_size=0.2,
                                                    random_state = 10)
```

In [75]:

```
# Logistic Regression

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

log_model = LogisticRegression()
log_model.fit(x_train, y_train)

print('Logistic Model Accuracy : ', log_model.score(x_test, y_test)*100, '%')

prediction = log_model.predict(x_test)
print('\nConfusion matrix :')
print(confusion_matrix(y_test, prediction))

print('\nClassification report:')
print(classification_report(y_test, prediction))
```

```
Logistic Model Accuracy :  78.84267631103074 %

Confusion matrix :
[[1744    0]
 [ 468    0]]

Classification report:
              precision    recall  f1-score   support

           0       0.79      1.00      0.88      1744
           1       0.00      0.00      0.00       468

    accuracy                           0.79      2212
   macro avg       0.39      0.50      0.44      2212
weighted avg       0.62      0.79      0.70      2212
```

Balancing dataset

In [77]:

```
# scaling all features
from sklearn.preprocessing import MinMaxScaler
MMS = MinMaxScaler()
x_train_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)
```

In [78]:

```
# adding samples to minority class using SMOTE
from imblearn.over_sampling import SMOTE
oversample = SMOTE()

x_train_oversam, y_train_oversam = oversample.fit_resample(x_train_scaled, y_train)
x_test_oversam, y_test_oversam = oversample.fit_resample(x_test_scaled, y_test)
```

In [79]:

```
# Original majority and minority class
y_train.value_counts(normalize=True)*100
```

Out[79]:

```
0    78.430929
1    21.569071
Name: STATUS, dtype: float64
```

In [80]:

```
# after using SMOTE
y_train_oversam.value_counts(normalize=True)*100
```

Out[80]:

```
0    50.0
1    50.0
Name: STATUS, dtype: float64
```

Machine Learning Model after Balancing

In [81]:

```
# Logistic Regression

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

log_model = LogisticRegression()
log_model.fit(x_train_oversam, y_train_oversam)

print('Logistic Model Accuracy : ', log_model.score(x_test_oversam, y_test_oversam)*100, '%')

prediction = log_model.predict(x_test_oversam)
print('\nConfusion matrix :')
print(confusion_matrix(y_test_oversam, prediction))

print('\nClassification report:')
print(classification_report(y_test_oversam, prediction))
```

```
Logistic Model Accuracy :  49.856651376146786 %

Confusion matrix :
[[968 776]
 [973 771]]

Classification report:
              precision    recall  f1-score   support

           0       0.50      0.56      0.53      1744
           1       0.50      0.44      0.47      1744

    accuracy                           0.50      3488
   macro avg       0.50      0.50      0.50      3488
weighted avg       0.50      0.50      0.50      3488
```

The effect of the over-sampled (SMOTE) model is not as good as that of the previous one, so this model is not used here and the previous model is still used