In [1]:
```
#import the librarys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
cd \Users\758449\Downloads
```

```
C:\Users\758449\Downloads
```

In [3]:
```
#read the file (data)
data = pd.read_csv("application_data.csv")

data
```

Out[3]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_
0	100002	1	Cash loans	M	N	Y	0	
1	100003	0	Cash loans	F	N	N	0	
2	100004	0	Revolving loans	M	Y	Y	0	
3	100006	0	Cash loans	F	N	Y	0	
4	100007	0	Cash loans	M	N	Y	0	
...	...	...	...	...	...	...	...	...
307506	456251	0	Cash loans	M	N	N	0	
307507	456252	0	Cash loans	F	N	Y	0	
307508	456253	0	Cash loans	F	N	Y	0	
307509	456254	1	Cash loans	F	N	Y	0	
307510	456255	0	Cash loans	F	N	N	0	

307511 rows × 122 columns

In [4]:
```
#GET SOME Info from dataset like how many numerical features and categorical features and descirption

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

from above dataset we found onething total 122 entries and floats are 65 and int are 41 and object are 16 and data have 307511 rows

In [5]:
```
pd.set_option('float_format', '{:f}'.format)
data.describe()
```

Out[5]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGIO
count	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307499.000000	307233.000000	
mean	278180.518577	0.080729	0.417052	168797.919297	599025.999706	27108.573909	538396.207429	
std	102790.175348	0.272419	0.722121	237123.146279	402490.776996	14493.737315	369446.460540	
min	100002.000000	0.000000	0.000000	25650.000000	45000.000000	1615.500000	40500.000000	
25%	189145.500000	0.000000	0.000000	112500.000000	270000.000000	16524.000000	238500.000000	
50%	278202.000000	0.000000	0.000000	147150.000000	513531.000000	24903.000000	450000.000000	
75%	367142.500000	0.000000	1.000000	202500.000000	808650.000000	34596.000000	679500.000000	
max	456255.000000	1.000000	19.000000	117000000.000000	4050000.000000	258025.500000	4050000.000000	

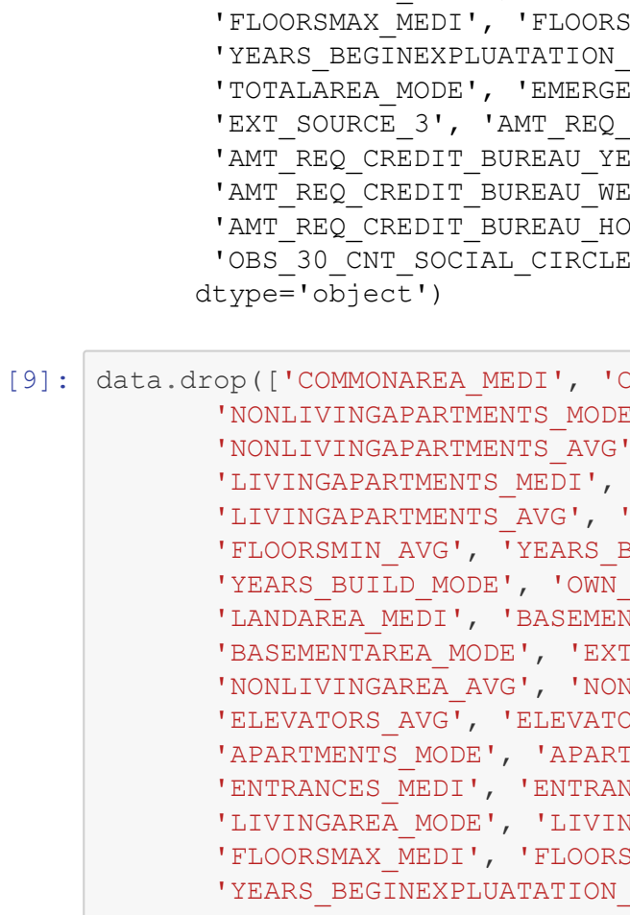
8 rows × 106 columns

In [6]:
```
#handling the missing values(replace with median)
pd.set_option('display.max_columns', None)
data.isnull().sum()
```

Out[6]:
```
SK_ID_CURR                        0
TARGET                            0
NAME_CONTRACT_TYPE                0
CODE_GENDER                       0
FLAG_OWN_CAR                      0
                              ...  
AMT_REQ_CREDIT_BUREAU_DAY     41519
AMT_REQ_CREDIT_BUREAU_WEEK    41519
AMT_REQ_CREDIT_BUREAU_MON     41519
AMT_REQ_CREDIT_BUREAU_QRT     41519
AMT_REQ_CREDIT_BUREAU_YEAR    41519
Length: 122, dtype: int64
```

In [7]:
```
#(100*data.isnull().sum()/len(data).round(2).sort_values(ascending= False)).head(40)
sns.distplot((100*data.isnull().sum()/len(data)).round(2).sort_values(ascending=False).head(60) )
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20e93e91dc0>
```



In [8]:
```
(100*data.isnull().sum()/len(data)).round(2).sort_values(ascending=False).head(60).index
```

Out[8]:
```
Index(['COMMONAREA_MEDI', 'COMMONAREA_AVG', 'COMMONAREA_MODE',
       'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAPARTMENTS_MEDI',
       'NONLIVINGAPARTMENTS_AVG', 'FONDKAPREMONT_MODE',
       'LIVINGAPARTMENTS_MEDI', 'LIVINGAPARTMENTS_MODE',
       'LIVINGAPARTMENTS_AVG', 'FLOORSMIN_MEDI', 'FLOORSMIN_MODE',
       'FLOORSMIN_AVG', 'YEARS_BUILD_MEDI', 'YEARS_BUILD_AVG',
       'YEARS_BUILD_MODE', 'OWN_CAR_AGE', 'LANDAREA_MODE', 'LANDAREA_AVG',
       'LANDAREA_MEDI', 'BASEMENTAREA_MEDI', 'BASEMENTAREA_AVG',
       'BASEMENTAREA_MODE', 'EXT_SOURCE_1', 'NONLIVINGAREA_MEDI',
       'NONLIVINGAREA_AVG', 'NONLIVINGAREA_MODE', 'ELEVATORS_MODE',
       'ELEVATORS_AVG', 'ELEVATORS_MEDI', 'WALLSMATERIAL_MODE',
       'APARTMENTS_MODE', 'APARTMENTS_AVG', 'APARTMENTS_MEDI',
       'ENTRANCES_MEDI', 'ENTRANCES_MODE', 'ENTRANCES_AVG', 'LIVINGAREA_MEDI',
       'LIVINGAREA_MODE', 'LIVINGAREA_AVG', 'HOUSETYPE_MODE', 'FLOORSMAX_MODE',
       'FLOORSMAX_MEDI', 'FLOORSMAX_AVG', 'YEARS_BEGINEXPLUATATION_MEDI',
       'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BEGINEXPLUATATION_MODE',
       'TOTALAREA_MODE', 'EMERGENCYSTATE_MODE', 'OCCUPATION_TYPE',
       'EXT_SOURCE_3', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR', 'AMT_REQ_CREDIT_BUREAU_DAY',
       'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON',
       'AMT_REQ_CREDIT_BUREAU_HOUR', 'NAME_TYPE_SUITE',
       'OBS_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE'],
      dtype='object')
```

In [9]:
```
data.drop(['COMMONAREA_MEDI', 'COMMONAREA_AVG', 'COMMONAREA_MODE',
       'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAPARTMENTS_MEDI',
       'NONLIVINGAPARTMENTS_AVG', 'FONDKAPREMONT_MODE',
       'LIVINGAPARTMENTS_MEDI', 'LIVINGAPARTMENTS_MODE',
       'LIVINGAPARTMENTS_AVG', 'FLOORSMIN_MEDI', 'FLOORSMIN_MODE',
       'FLOORSMIN_AVG', 'YEARS_BUILD_MEDI', 'YEARS_BUILD_AVG',
       'YEARS_BUILD_MODE', 'OWN_CAR_AGE', 'LANDAREA_MODE', 'LANDAREA_AVG',
       'LANDAREA_MEDI', 'BASEMENTAREA_MEDI', 'BASEMENTAREA_AVG',
       'BASEMENTAREA_MODE', 'EXT_SOURCE_1', 'NONLIVINGAREA_MEDI',
       'NONLIVINGAREA_AVG', 'NONLIVINGAREA_MODE', 'ELEVATORS_MODE',
       'ELEVATORS_AVG', 'ELEVATORS_MEDI', 'WALLSMATERIAL_MODE',
       'APARTMENTS_MODE', 'APARTMENTS_AVG', 'APARTMENTS_MEDI',
       'ENTRANCES_MEDI', 'ENTRANCES_MODE', 'ENTRANCES_AVG', 'LIVINGAREA_MEDI',
       'LIVINGAREA_MODE', 'LIVINGAREA_AVG', 'HOUSETYPE_MODE', 'FLOORSMAX_MODE',
       'FLOORSMAX_MEDI', 'FLOORSMAX_AVG', 'YEARS_BEGINEXPLUATATION_MEDI',
       'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BEGINEXPLUATATION_MODE',
       'TOTALAREA_MODE', 'EMERGENCYSTATE_MODE', 'OCCUPATION_TYPE',
       'EXT_SOURCE_3'],axis = 1,inplace = True)
```

In [10]:
```
(100*data.isnull().sum()/len(data)).round(2).sort_values(ascending=False).head(20)
```

Out[10]:
```
AMT_REQ_CREDIT_BUREAU_YEAR   13.500000
AMT_REQ_CREDIT_BUREAU_MON    13.500000
AMT_REQ_CREDIT_BUREAU_WEEK   13.500000
AMT_REQ_CREDIT_BUREAU_DAY    13.500000
AMT_REQ_CREDIT_BUREAU_HOUR   13.500000
AMT_REQ_CREDIT_BUREAU_QRT    13.500000
NAME_TYPE_SUITE               0.420000
DEF_60_CNT_SOCIAL_CIRCLE      0.330000
OBS_60_CNT_SOCIAL_CIRCLE      0.330000
DEF_30_CNT_SOCIAL_CIRCLE      0.330000
OBS_30_CNT_SOCIAL_CIRCLE      0.330000
EXT_SOURCE_2                  0.210000
AMT_GOODS_PRICE               0.090000
DAYS_ID_PUBLISH               0.000000
FLAG_MOBIL                    0.000000
FLAG_WORK_PHONE               0.000000
FLAG_EMP_PHONE                0.000000
DAYS_EMPLOYED                 0.000000
FLAG_CONT_MOBILE              0.000000
FLAG_PHONE                    0.000000
dtype: float64
```

In [11]:
```
data['AMT_REQ_CREDIT_BUREAU_YEAR'].fillna(data.AMT_REQ_CREDIT_BUREAU_YEAR.median(),inplace =True)
data['AMT_REQ_CREDIT_BUREAU_MON'].fillna(data.AMT_REQ_CREDIT_BUREAU_MON.median(),inplace =True)
data['AMT_REQ_CREDIT_BUREAU_DAY'].fillna(data.AMT_REQ_CREDIT_BUREAU_DAY.median(),inplace =True)
data['AMT_REQ_CREDIT_BUREAU_WEEK'].fillna(data.AMT_REQ_CREDIT_BUREAU_WEEK.median(),inplace =True)
data['AMT_REQ_CREDIT_BUREAU_HOUR'].fillna(data.AMT_REQ_CREDIT_BUREAU_HOUR.median(),inplace =True)
data['AMT_REQ_CREDIT_BUREAU_QRT'].fillna(data.AMT_REQ_CREDIT_BUREAU_QRT.median(),inplace =True)
data['DEF_60_CNT_SOCIAL_CIRCLE'].fillna(data.DEF_60_CNT_SOCIAL_CIRCLE.median(),inplace =True)
data['DEF_30_CNT_SOCIAL_CIRCLE'].fillna(data.OBS_30_CNT_SOCIAL_CIRCLE.median(),inplace =True)
data['OBS_60_CNT_SOCIAL_CIRCLE'].fillna(data.OBS_60_CNT_SOCIAL_CIRCLE.median(),inplace =True)
data['OBS_30_CNT_SOCIAL_CIRCLE'].fillna(data.OBS_30_CNT_SOCIAL_CIRCLE.median(),inplace =True)
data['OBS_30_CNT_SOCIAL_CIRCLE'].fillna(data.OBS_30_CNT_SOCIAL_CIRCLE.median(),inplace =True)
data['EXT_SOURCE_2'].fillna(data.EXT_SOURCE_2.median(),inplace =True)
data['AMT_GOODS_PRICE'].fillna(data.AMT_GOODS_PRICE.median(),inplace =True)
data['AMT_ANNUITY'].fillna(data.AMT_ANNUITY.median(),inplace =True)
data['NAME_TYPE_SUITE'].fillna(data.NAME_TYPE_SUITE.mode(),inplace =True)
```

In [12]:
```
data.isnull().sum()
```

Out[12]:
```
SK_ID_CURR                    0
TARGET                        0
NAME_CONTRACT_TYPE            0
CODE_GENDER                   0
FLAG_OWN_CAR                  0
                             ..
AMT_REQ_CREDIT_BUREAU_DAY     0
AMT_REQ_CREDIT_BUREAU_WEEK    0
AMT_REQ_CREDIT_BUREAU_MON     0
AMT_REQ_CREDIT_BUREAU_QRT     0
AMT_REQ_CREDIT_BUREAU_YEAR    0
Length: 71, dtype: int64
```

In [13]:
```
(100*data.isnull().sum()/len(data)).round(2).sort_values(ascending=False).head(20)
```

Out[13]:
```
NAME_TYPE_SUITE               0.420000
AMT_REQ_CREDIT_BUREAU_YEAR    0.000000
DAYS_BIRTH                    0.000000
DAYS_REGISTRATION             0.000000
DAYS_ID_PUBLISH               0.000000
FLAG_MOBIL                    0.000000
FLAG_EMP_PHONE                0.000000
FLAG_WORK_PHONE               0.000000
FLAG_CONT_MOBILE              0.000000
FLAG_PHONE                    0.000000
FLAG_EMAIL                    0.000000
CNT_FAM_MEMBERS               0.000000
REGION_RATING_CLIENT          0.000000
REGION_RATING_CLIENT_W_CITY   0.000000
WEEKDAY_APPR_PROCESS_START    0.000000
HOUR_APPR_PROCESS_START       0.000000
REG_REGION_NOT_LIVE_REGION    0.000000
DAYS_EMPLOYED                 0.000000
REGION_POPULATION_RELATIVE    0.000000
LIVE_REGION_NOT_WORK_REGION   0.000000
dtype: float64
```

In [14]:
```
data['NAME_TYPE_SUITE'].value_counts()
```

Out[14]:
```
Unaccompanied      248526
Family              40149
Spouse, partner     11370
Children             3267
Other_B              1770
Other_A               866
Group of people       271
Name: NAME_TYPE_SUITE, dtype: int64
```

In [15]:
```
data['NAME_TYPE_SUITE'].fillna(data.NAME_TYPE_SUITE.mode(),inplace =True)
```

In [16]:
```
(100*data.isnull().sum()/len(data)).round(2).sort_values(ascending=False).head(20)
```

Out[16]:
```
NAME_TYPE_SUITE               0.420000
AMT_REQ_CREDIT_BUREAU_YEAR    0.000000
DAYS_BIRTH                    0.000000
DAYS_REGISTRATION             0.000000
DAYS_ID_PUBLISH               0.000000
FLAG_MOBIL                    0.000000
FLAG_EMP_PHONE                0.000000
FLAG_WORK_PHONE               0.000000
FLAG_CONT_MOBILE              0.000000
FLAG_PHONE                    0.000000
FLAG_EMAIL                    0.000000
CNT_FAM_MEMBERS               0.000000
REGION_RATING_CLIENT          0.000000
REGION_RATING_CLIENT_W_CITY   0.000000
WEEKDAY_APPR_PROCESS_START    0.000000
HOUR_APPR_PROCESS_START       0.000000
REG_REGION_NOT_LIVE_REGION    0.000000
DAYS_EMPLOYED                 0.000000
REGION_POPULATION_RELATIVE    0.000000
LIVE_REGION_NOT_WORK_REGION   0.000000
dtype: float64
```

In [17]:
```
data.drop(["SK_ID_CURR"],axis =1,inplace = True)
```

In [ ]:

In [18]:
```
#outliers treatment for numerical features(using IQR AND z-score)(num_feat)

data.describe()
```

Out[18]:

	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_
count	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307
mean	0.080729	0.417052	168797.919297	599025.999706	27108.487841	538316.294367	
std	0.272419	0.722121	237123.146279	402490.776996	14493.461065	369288.982246	
min	0.000000	0.000000	25650.000000	45000.000000	1615.500000	40500.000000	
25%	0.000000	0.000000	112500.000000	270000.000000	16524.000000	238500.000000	
50%	0.000000	0.000000	147150.000000	513531.000000	24903.000000	450000.000000	
75%	0.000000	1.000000	202500.000000	808650.000000	34596.000000	679500.000000	
max	1.000000	19.000000	117000000.000000	4050000.000000	258025.500000	4050000.000000	

In [19]:
```
#from sklearn.preprocessing import StandardScaler

#sc = StandardScaler()

#standard_df = sc.fit_transform(data)
#standard_df = pd.DataFrame(standard_df, columns =['x1', 'x2'])
```

In [20]:
```
pd.set_option('float_format', '{:f}'.format)
num_feat=data.select_dtypes (['int64','float64'])
```

In [ ]:

In [21]:
```
num_feat
```

Out[21]:

	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_REL
0	1	0	202500.000000	406597.500000	24700.500000	351000.000000	0.0
1	0	0	270000.000000	1293502.500000	35698.500000	1129500.000000	0.0
2	0	0	67500.000000	135000.000000	6750.000000	135000.000000	0.0
3	0	0	135000.000000	312682.500000	29686.500000	297000.000000	0.0
4	0	0	121500.000000	513000.000000	21865.500000	513000.000000	0.0
...	...	...	...	...	...	...	
307506	0	0	157500.000000	254700.000000	27558.000000	225000.000000	0.0
307507	0	0	72000.000000	269550.000000	12001.500000	225000.000000	0.0
307508	0	0	153000.000000	677664.000000	29979.000000	585000.000000	0.0
307509	1	0	171000.000000	370107.000000	20205.000000	319500.000000	0.0
307510	0	0	157500.000000	675000.000000	49117.500000	675000.000000	0.0

307511 rows × 59 columns

In [ ]:

In [22]:
```
num_feat['DAYS_BIRTH'] = num_feat['DAYS_BIRTH'].abs()
num_feat['DAYS_EMPLOYED'] = num_feat['DAYS_EMPLOYED'].abs()
num_feat['DAYS_REGISTRATION'] = num_feat['DAYS_REGISTRATION'].abs()
num_feat['DAYS_ID_PUBLISH'] = num_feat['DAYS_ID_PUBLISH'].abs()
num_feat['DAYS_LAST_PHONE_CHANGE'] = num_feat['DAYS_LAST_PHONE_CHANGE'].abs()
```

In [23]:
```
num_feat
```

Out[23]:

	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_REL
0	1	0	202500.000000	406597.500000	24700.500000	351000.000000	0.0
1	0	0	270000.000000	1293502.500000	35698.500000	1129500.000000	0.0
2	0	0	67500.000000	135000.000000	6750.000000	135000.000000	0.0
3	0	0	135000.000000	312682.500000	29686.500000	297000.000000	0.0
4	0	0	121500.000000	513000.000000	21865.500000	513000.000000	0.0
...	...	...	...	...	...	...	
307506	0	0	157500.000000	254700.000000	27558.000000	225000.000000	0.0
307507	0	0	72000.000000	269550.000000	12001.500000	225000.000000	0.0
307508	0	0	153000.000000	677664.000000	29979.000000	585000.000000	0.0
307509	1	0	171000.000000	370107.000000	20205.000000	319500.000000	0.0
307510	0	0	157500.000000	675000.000000	49117.500000	675000.000000	0.0

307511 rows × 59 columns

In [ ]:

In [24]:
```
num_feat.drop(['TARGET'],axis = 1,inplace= True)
```

In [25]:
```
num_feat.columns
```

Out[25]:
```
Index(['CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
       'AMT_GOODS_PRICE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'FLAG_MOBIL',
       'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE',
       'FLAG_EMAIL', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
       'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURCE_2',
       'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
       'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3',
       'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6',
       'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9',
       'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12',
       'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15',
       'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',
       'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',
       'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY',
       'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON',
       'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object')
```

In [ ]:

In [ ]:

In [26]:
```
num_feat.drop(["DAYS_LAST_PHONE_CHANGE"],axis = 1,inplace =True)
```

In [27]:
```
num_feat.drop(["DEF_30_CNT_SOCIAL_CIRCLE"],axis = 1,inplace =True)
```

In [28]:
```
num_feat.drop(["FLAG_DOCUMENT_2"],axis = 1,inplace =True)
```

In [29]:
```
#AMT_REQ_CREDIT_BUREAU_HOUR
IQR =num_feat.AMT_REQ_CREDIT_BUREAU_HOUR.quantile(0.75)-num_feat.AMT_REQ_CREDIT_BUREAU_HOUR.quantile(0.
25)
lower_bridge =num_feat['AMT_REQ_CREDIT_BUREAU_HOUR'].quantile(0.25)-(IQR*3)
upper_bridge =num_feat['AMT_REQ_CREDIT_BUREAU_HOUR'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
0.0
0.0
```

In [30]:
```
num_feat.loc[num_feat['AMT_REQ_CREDIT_BUREAU_WEEK']>0,'AMT_REQ_CREDIT_BUREAU_WEEK']=0
num_feat.loc[num_feat['AMT_REQ_CREDIT_BUREAU_DAY']>0,'AMT_REQ_CREDIT_BUREAU_DAY']=0
num_feat.loc[num_feat['AMT_REQ_CREDIT_BUREAU_MON']>0,'AMT_REQ_CREDIT_BUREAU_MON']=0
num_feat.loc[num_feat['AMT_REQ_CREDIT_BUREAU_QRT']>0,'AMT_REQ_CREDIT_BUREAU_QRT']=0
num_feat.loc[num_feat['AMT_REQ_CREDIT_BUREAU_YEAR']>0,'AMT_REQ_CREDIT_BUREAU_YEAR']=0
```

In [ ]:

In [31]:
```
#DAYS_LAST_PHONE_CHANGE
```

In [32]:
```
#num_feat.loc[num_feat['DAYS_LAST_PHONE_CHANGE']>3614,'DAYS_LAST_PHONE_CHANGE']=3614
```

In [ ]:

In [ ]:

In [33]:
```
num_feat.drop(['HOUR_APPR_PROCESS_START'],axis = 1,inplace =True)
```

In [34]:
```
#OBS_30_CNT_SOCIAL_CIRCLE
IQR =num_feat.OBS_30_CNT_SOCIAL_CIRCLE.quantile(0.75)-num_feat.OBS_30_CNT_SOCIAL_CIRCLE.quantile(0.25)
lower_bridge =num_feat['OBS_30_CNT_SOCIAL_CIRCLE'].quantile(0.25)-(IQR*3)
upper_bridge =num_feat['OBS_30_CNT_SOCIAL_CIRCLE'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
-6.0
-4.0
```

In [35]:
```
#num_feat.loc[num_feat['OBS_60_CNT_SOCIAL_CIRCLE']>6,'OBS_60_CNT_SOCIAL_CIRCLE']=6
```

In [36]:
```
#DEF_30_CNT_SOCIAL_CIRCLE
# IQR =num_feat.DEF_30_CNT_SOCIAL_CIRCLE.quantile(0.75)-num_feat.DEF_30_CNT_SOCIAL_CIRCLE.quantile(0.2
5)
# lower_bridge =num_feat['DEF_30_CNT_SOCIAL_CIRCLE'].quantile(0.25)-(IQR*3)
# upper_bridge =num_feat['DEF_30_CNT_SOCIAL_CIRCLE'].quantile(0.75)-(IQR*3)
# print(lower_bridge)
# print(upper_bridge)
```

In [37]:
```
num_feat.loc[num_feat['DEF_60_CNT_SOCIAL_CIRCLE']>0,'DEF_60_CNT_SOCIAL_CIRCLE']=0
```

In [38]:
```
#OBS_60_CNT_SOCIAL_CIRCLE
IQR =num_feat.OBS_60_CNT_SOCIAL_CIRCLE.quantile(0.75)-num_feat.OBS_60_CNT_SOCIAL_CIRCLE.quantile(0.25)
lower_bridge =num_feat['OBS_60_CNT_SOCIAL_CIRCLE'].quantile(0.25)-(IQR*3)
upper_bridge =num_feat['OBS_60_CNT_SOCIAL_CIRCLE'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
-6.0
-4.0
```

In [39]:
```
#CNT_CHILDREN
IQR =num_feat.CNT_CHILDREN.quantile(0.75)-num_feat.CNT_CHILDREN.quantile(0.25)
lower_bridge =num_feat['CNT_CHILDREN'].quantile(0.25)-(IQR*3)
upper_bridge =num_feat['CNT_CHILDREN'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
-3.0
-2.0
```

In [40]:
```
num_feat.loc[num_feat['CNT_CHILDREN']>3,'CNT_CHILDREN']=3
```

In [41]:
```
#AMT_CREDIT
IQR =num_feat.AMT_CREDIT.quantile(0.75)-num_feat.AMT_CREDIT.quantile(0.25)
lower_bridge =num_feat['AMT_CREDIT'].quantile(0.25)-(IQR*3)
upper_bridge =num_feat['AMT_CREDIT'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
-1345950.0
-807300.0
```

In [42]:
```
num_feat.loc[num_feat['AMT_CREDIT']>1345950,'AMT_CREDIT']=1345950
```

In [43]:
```
#AMT_INCOME_TOTAL
IQR =num_feat.AMT_INCOME_TOTAL.quantile(0.75)-num_feat.AMT_INCOME_TOTAL.quantile(0.25)
lower_bridge =num_feat['AMT_INCOME_TOTAL'].quantile(0.25)-(IQR*3)
upper_bridge =num_feat['AMT_INCOME_TOTAL'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
-157500.0
-67500.0
```

In [44]:
```
num_feat.loc[num_feat['AMT_INCOME_TOTAL']>200000,'AMT_INCOME_TOTAL']=200000
```

In [45]:
```
upper_boundries = num_feat['AMT_ANNUITY'].mean()+ 3*num_feat['AMT_ANNUITY'].std()# find the upper bound
ries
lower_boundries = num_feat['AMT_ANNUITY'].mean()- 3*num_feat['AMT_ANNUITY'].std()#finiding the lower bo
undries
print(upper_boundries)
print(lower_boundries)
```

```
70588.8710375585
-16371.895355387784
```

In [46]:
```
#AMT_ANNUITY
IQR =num_feat.AMT_ANNUITY.quantile(0.75)-num_feat.AMT_ANNUITY.quantile(0.25)
lower_bridge =num_feat['AMT_ANNUITY'].quantile(0.25)-(IQR*3)
upper_bridge =num_feat['AMT_ANNUITY'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
-37692.0
-19620.0
```

In [47]:
```
num_feat.loc[num_feat['AMT_ANNUITY']>70588,'AMT_ANNUITY']=70588
```

In [48]:
```
#AMT_GOODS_PRICE
IQR =num_feat.AMT_GOODS_PRICE.quantile(0.75)-num_feat.AMT_GOODS_PRICE.quantile(0.25)
lower_bridge =num_feat['AMT_GOODS_PRICE'].quantile(0.25)-(IQR*3)
upper_bridge =num_feat['AMT_GOODS_PRICE'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
-1084500.0
-643500.0
```

In [49]:
```
num_feat.loc[num_feat['AMT_GOODS_PRICE']>1084500,'AMT_GOODS_PRICE']=1084500
```

In [50]:
```
#DAYS_EMPLOYED
IQR =num_feat.DAYS_EMPLOYED.quantile(0.75)-num_feat.DAYS_EMPLOYED.quantile(0.25)
lower_bridge =num_feat['DAYS_EMPLOYED'].quantile(0.25)-(IQR*3)
upper_bridge =num_feat['DAYS_EMPLOYED'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
-13389.0
-8615.0
```

In [51]:
```
num_feat.loc[num_feat['DAYS_EMPLOYED']>13389,'DAYS_EMPLOYED']=13389
```

In [52]:
```
#DAYS_REGISTRATION
IQR =num_feat.DAYS_REGISTRATION.quantile(0.75)-num_feat.DAYS_REGISTRATION.quantile(0.25)
lower_bridge =num_feat['DAYS_REGISTRATION'].quantile(0.25)-(IQR*3)
upper_bridge =num_feat['DAYS_REGISTRATION'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
-14398.5
-8929.0
```

In [53]:
```
num_feat.loc[num_feat['DAYS_REGISTRATION']>14398,'DAYS_REGISTRATION']=14398
```

In [54]:
```
#CNT_FAM_MEMBERS
IQR =num_feat.CNT_FAM_MEMBERS.quantile(0.75)-num_feat.CNT_FAM_MEMBERS.quantile(0.25)
lower_bridge =num_feat['CNT_FAM_MEMBERS'].quantile(0.25)-(IQR*3)
upper_bridge =num_feat['CNT_FAM_MEMBERS'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
-1.0
0.0
```

In [55]:
```
upper_boundries = num_feat['CNT_FAM_MEMBERS'].mean()+ 3*num_feat['CNT_FAM_MEMBERS'].std()# find the upp
er boundries
lower_boundries = num_feat['CNT_FAM_MEMBERS'].mean()- 3*num_feat['CNT_FAM_MEMBERS'].std()#finiding the
 lower boundries
print(upper_boundries)
print(lower_boundries)
```

```
4.8847101579741015
-0.5793792570898995
```

In [56]:
```
num_feat.loc[num_feat['CNT_FAM_MEMBERS']>3,'CNT_FAM_MEMBERS']=3
```

In [57]:
```
num_feat.drop([])
```

Out[57]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE	D/
0	0	200000.000000	406597.500000	24700.500000	351000.000000	0.018801	
1	0	200000.000000	1293502.500000	35698.500000	1084500.000000	0.003541	
2	0	67500.000000	135000.000000	6750.000000	135000.000000	0.010032	
3	0	135000.000000	312682.500000	29686.500000	297000.000000	0.008019	
4	0	121500.000000	513000.000000	21865.500000	513000.000000	0.028663	
...	...	...	...	...	...	...	
307506	0	157500.000000	254700.000000	27558.000000	225000.000000	0.032561	
307507	0	72000.000000	269550.000000	12001.500000	225000.000000	0.025164	
307508	0	153000.000000	677664.000000	29979.000000	585000.000000	0.005002	
307509	0	171000.000000	370107.000000	20205.000000	319500.000000	0.005313	
307510	0	157500.000000	675000.000000	49117.500000	675000.000000	0.046220	

307511 rows × 54 columns

In [58]:
```
sns.heatmap(num_feat.corr(method= 'pearson'))
```

Out[58]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20e93d96c70>
```

In [59]:
```
def correlation(num_feat,threshold):
    col_corr=set()
    corr_matrix=num_feat.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range (i):
            if abs(corr_matrix.iloc[i,j]>threshold):
                colname =corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr
```

In [60]:
```
corr_features =correlation(num_feat,0.2)
len(set(corr_features))
```

Out[60]:
```
19
```

In [61]:
```
corr_features
```

Out[61]:
```
{'AMT_ANNUITY',
 'AMT_CREDIT',
 'AMT_GOODS_PRICE',
 'CNT_FAM_MEMBERS',
 'DAYS_EMPLOYED',
 'DAYS_ID_PUBLISH',
 'DAYS_REGISTRATION',
 'FLAG_DOCUMENT_3',
 'FLAG_DOCUMENT_6',
 'FLAG_EMP_PHONE',
 'FLAG_PHONE',
 'FLAG_WORK_PHONE',
 'LIVE_CITY_NOT_WORK_CITY',
 'LIVE_REGION_NOT_WORK_REGION',
 'OBS_60_CNT_SOCIAL_CIRCLE',
 'REGION_RATING_CLIENT_W_CITY',
 'REG_CITY_NOT_LIVE_CITY',
 'REG_CITY_NOT_WORK_CITY',
 'REG_REGION_NOT_WORK_REGION'}
```

In [62]:
```
num_feat.describe()
```

Out[62]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE	D/
count	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307
mean	0.414401	145831.230382	582861.869218	26940.161632	511901.974523	0.020868	16
std	0.707106	46518.877135	357882.236012	13689.567715	303278.121247	0.013831	4
min	0.000000	25650.000000	45000.000000	1615.500000	40500.000000	0.000290	7
25%	0.000000	112500.000000	270000.000000	16524.000000	238500.000000	0.010006	12
50%	0.000000	147150.000000	513531.000000	24903.000000	450000.000000	0.018850	15
75%	1.000000	200000.000000	808650.000000	34596.000000	679500.000000	0.028663	19
max	3.000000	200000.000000	1345950.000000	70588.000000	1084500.000000	0.072508	25

In [63]:
```
num_feat = num_feat[['AMT_ANNUITY','AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'AMT_GOODS_PRICE',
 'CNT_FAM_MEMBERS',
 'DAYS_EMPLOYED',
 'DAYS_ID_PUBLISH',
 'DAYS_REGISTRATION']]
```

In [64]:
```
num_feat
```

Out[64]:

	AMT_ANNUITY	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_GOODS_PRICE	CNT_FAM_MEMBERS	DAYS_EMPLOYED	DAYS_ID_PU
0	24700.500000	200000.000000	406597.500000	351000.000000	1.000000	637	
1	35698.500000	200000.000000	1293502.500000	1084500.000000	2.000000	1188	
2	6750.000000	67500.000000	135000.000000	135000.000000	1.000000	225	
3	29686.500000	135000.000000	312682.500000	297000.000000	2.000000	3039	
4	21865.500000	121500.000000	513000.000000	513000.000000	1.000000	3038	
...	...	...	...	...	...	...	
307506	27558.000000	157500.000000	254700.000000	225000.000000	1.000000	236	
307507	12001.500000	72000.000000	269550.000000	225000.000000	1.000000	13389	
307508	29979.000000	153000.000000	677664.000000	585000.000000	1.000000	7921	
307509	20205.000000	171000.000000	370107.000000	319500.000000	2.000000	4786	
307510	49117.500000	157500.000000	675000.000000	675000.000000	2.000000	1262	

307511 rows × 8 columns

In [65]:
```
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.feature_selection import chi2
best_features= SelectKBest(score_func=chi2,k=15)
```

In [66]:
```
# fit = best_features.fit(X,y)
# df_score= pd.DataFrame(fit.scores_)
# df_columns= pd.DataFrame(X.columns)
# featurescores=pd.concat([df_columns,df_score],axis=1,)
# # featurescores.columns=['species','score']
```

In [67]:
```
#num_feat.drop(['AMT_INCOME_TOTAL'],axis =1 ,inplace = True)
```

In [68]:
```
num_feat['AMT_CREDIT'].median()
```

Out[68]:
```
513531.0
```

In [69]:
```
#gaussian distribution
upper_boundries = data['AMT_CREDIT'].mean()+ 3*data['AMT_CREDIT'].std()# find the upper boundries
lower_boundries = data['AMT_CREDIT'].mean()- 3*data['AMT_CREDIT'].std()#finiding the lower boundries
print(upper_boundries)
print(lower_boundries)
```

```
1806498.3306935397
-608446.3312821365
```

In [70]:
```
(num_feat["AMT_CREDIT"]<1345950).value_counts()
```

Out[70]:
```
True     291112
False     16399
Name: AMT_CREDIT, dtype: int64
```

In [71]:
```
IQR =data.AMT_INCOME_TOTAL.quantile(0.75)-data.AMT_INCOME_TOTAL.quantile(0.25)
```

In [72]:
```
lower_bridge =data['AMT_INCOME_TOTAL'].quantile(0.25)-(IQR*3)
upper_bridge =data['AMT_INCOME_TOTAL'].quantile(0.75)-(IQR*3)
print(lower_bridge)
print(upper_bridge)
```

```
-157500.0
-67500.0
```

In [73]:
```
#num_feat.loc[num_feat['AMT_INCOME_TOTAL']>227542,'AMT_INCOME_TOTAL']=227542
```

In [ ]:

In [ ]:

In [74]:
```
# from sklearn.preprocessing import StandardScaler

#sc = StandardScaler()

#standard_df = sc.fit_transform(num_feat)
```

In [ ]:

In [ ]:

In [ ]:

In [75]:
```
# #num_feat = pd.DataFrame(standard_df,columns=['CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT',
#        'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'REGION_POPULATION_RELATIVE',
#        'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH',
#        'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE',
#        'FLAG_PHONE', 'FLAG_EMAIL', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
#        'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START',
#        'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
#        'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
#        'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURCE_2',
#        'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
#        'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
#        'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3',
#        'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6',
#        'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9',
#        'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12',
#        'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15',
#        'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',
#        'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',
#        'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY',
#        'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON',
#        'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR'])
```

In [ ]:

```
In [76]: num_feat
```

Out[76]:

	AMT_ANNUITY	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_GOODS_PRICE	CNT_FAM_MEMBERS	DAYS_EMPLOYED	DAYS_ID_PU
0	24700.500000	200000.000000	406597.500000	351000.000000	1.000000	637	
1	35698.500000	200000.000000	1293502.500000	1084500.000000	2.000000	1188	
2	6750.000000	67500.000000	135000.000000	135000.000000	1.000000	225	
3	29686.500000	135000.000000	312682.500000	297000.000000	2.000000	3039	
4	21865.500000	121500.000000	513000.000000	513000.000000	1.000000	3038	
...	...	...	...	...	...	...	
307506	27558.000000	157500.000000	254700.000000	225000.000000	1.000000	236	
307507	12001.500000	72000.000000	269550.000000	225000.000000	1.000000	13389	
307508	29979.000000	153000.000000	677664.000000	585000.000000	1.000000	7921	
307509	20205.000000	171000.000000	370107.000000	319500.000000	2.000000	4786	
307510	49117.500000	157500.000000	675000.000000	675000.000000	2.000000	1262	

307511 rows × 8 columns

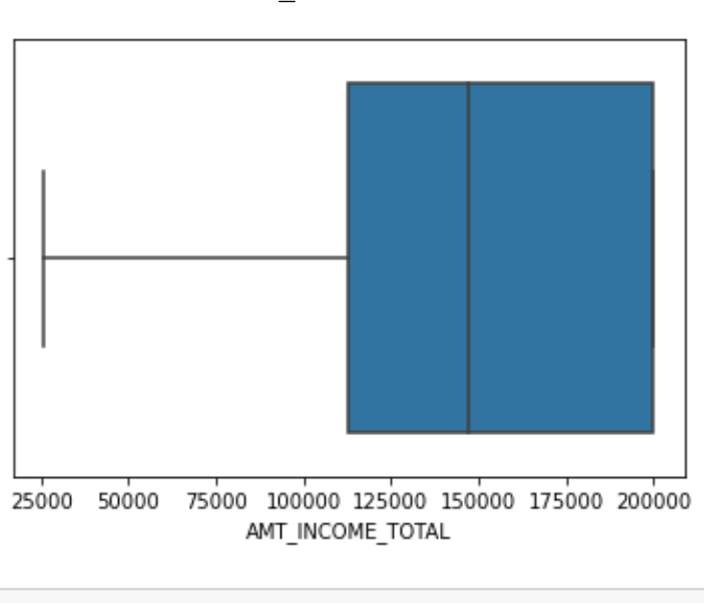
```
In [77]: num_feat.describe()
```

Out[77]:

	AMT_ANNUITY	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_GOODS_PRICE	CNT_FAM_MEMBERS	DAYS_EMPLOYED	DAYS_ID_PU
count	307511.000000	307511.000000	307511.000000	307511.000000	307509.000000	307511.000000	307511.0
mean	26940.161632	145831.230382	582861.869218	511901.974523	2.043765	4363.448085	2994.2
std	13689.567715	46518.877135	357882.236012	303278.121247	0.695068	4724.082848	1509.4
min	1615.500000	25650.000000	45000.000000	40500.000000	1.000000	0.000000	0.0
25%	16524.000000	112500.000000	270000.000000	238500.000000	2.000000	933.000000	1720.0
50%	24903.000000	147150.000000	513531.000000	450000.000000	2.000000	2219.000000	3254.0
75%	34596.000000	200000.000000	808650.000000	679500.000000	3.000000	5707.000000	4299.0
max	70588.000000	200000.000000	1345950.000000	1084500.000000	3.000000	13389.000000	7197.0

```
In [78]: sns.boxplot(num_feat["AMT_INCOME_TOTAL"])

Out[78]: <matplotlib.axes._subplots.AxesSubplot at 0x20e94120ca0>
```



```
In [79]: num_feat.drop(num_feat['AMT_INCOME_TOTAL'].idxmax(),inplace = True)

In [80]: num_feat
```

Out[80]:

	AMT_ANNUITY	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_GOODS_PRICE	CNT_FAM_MEMBERS	DAYS_EMPLOYED	DAYS_ID_PU
1	35698.500000	200000.000000	1293502.500000	1084500.000000	2.000000	1188	
2	6750.000000	67500.000000	135000.000000	135000.000000	1.000000	225	
3	29686.500000	135000.000000	312682.500000	297000.000000	2.000000	3039	
4	21865.500000	121500.000000	513000.000000	513000.000000	1.000000	3038	
5	27517.500000	99000.000000	490495.500000	454500.000000	2.000000	1588	
...	...	...	...	...	...	...	
307506	27558.000000	157500.000000	254700.000000	225000.000000	1.000000	236	
307507	12001.500000	72000.000000	269550.000000	225000.000000	1.000000	13389	
307508	29979.000000	153000.000000	677664.000000	585000.000000	1.000000	7921	
307509	20205.000000	171000.000000	370107.000000	319500.000000	2.000000	4786	
307510	49117.500000	157500.000000	675000.000000	675000.000000	2.000000	1262	

307510 rows × 8 columns

```
In [81]: #encodijng the categorical features using (lebal encoding)(cat_feat)

In [82]: cat_feat=data.select_dtypes (['object'])

In [83]: cat_feat.columns

Out[83]: Index(['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
                'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
                'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'WEEKDAY_APPR_PROCESS_START',
                'ORGANIZATION_TYPE'],
               dtype='object')

In [84]: cat_feat =data[['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
                'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
                'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'WEEKDAY_APPR_PROCESS_START',
                'ORGANIZATION_TYPE','TARGET']]

In [85]: cat_feat
```

Out[85]:

	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	NAME_TYPE_SUITE	NAME_INCOME_TYPE	NA
0	Cash loans	M	N	Y	Unaccompanied	Working	
1	Cash loans	F	N	N	Family	State servant	
2	Revolving loans	M	Y	Y	Unaccompanied	Working	
3	Cash loans	F	N	Y	Unaccompanied	Working	
4	Cash loans	M	N	Y	Unaccompanied	Working	
...	...	...	...	...	...	...	
307506	Cash loans	M	N	N	Unaccompanied	Working	
307507	Cash loans	F	N	Y	Unaccompanied	Pensioner	
307508	Cash loans	F	N	Y	Unaccompanied	Working	
307509	Cash loans	F	N	Y	Unaccompanied	Commercial associate	
307510	Cash loans	F	N	N	Unaccompanied	Commercial associate	

307511 rows × 12 columns

```
In [86]: #cat_feat.drop(['TARGET'],axis =1 ,inplace = True)

In [87]: cat_feat['NAME_TYPE_SUITE'].isnull().sum()

Out[87]: 1292

In [88]: cat_feat['NAME_TYPE_SUITE'].fillna(cat_feat['NAME_TYPE_SUITE'].value_counts().index[0],inplace = True)

In [89]: from sklearn.preprocessing import LabelEncoder
         lb = LabelEncoder()
         cat_feat =cat_feat.apply(LabelEncoder().fit_transform)

In [90]: cat_feat
```

Out[90]:

	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	NAME_TYPE_SUITE	NAME_INCOME_TYPE	NA
0	0	1	0	1	6	7	
1	0	0	0	0	1	4	
2	1	1	1	1	6	7	
3	0	0	0	1	6	7	
4	0	1	0	1	6	7	
...	...	...	...	...	...	...	
307506	0	1	0	0	6	7	
307507	0	0	0	1	6	3	
307508	0	0	0	1	6	7	
307509	0	0	0	1	6	1	
307510	0	0	0	0	6	1	

307511 rows × 12 columns

```
In [91]: num_feat
```

Out[91]:

	AMT_ANNUITY	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_GOODS_PRICE	CNT_FAM_MEMBERS	DAYS_EMPLOYED	DAYS_ID_PU
1	35698.500000	200000.000000	1293502.500000	1084500.000000	2.000000	1188	
2	6750.000000	67500.000000	135000.000000	135000.000000	1.000000	225	
3	29686.500000	135000.000000	312682.500000	297000.000000	2.000000	3039	
4	21865.500000	121500.000000	513000.000000	513000.000000	1.000000	3038	
5	27517.500000	99000.000000	490495.500000	454500.000000	2.000000	1588	
...	...	...	...	...	...	...	
307506	27558.000000	157500.000000	254700.000000	225000.000000	1.000000	236	
307507	12001.500000	72000.000000	269550.000000	225000.000000	1.000000	13389	
307508	29979.000000	153000.000000	677664.000000	585000.000000	1.000000	7921	
307509	20205.000000	171000.000000	370107.000000	319500.000000	2.000000	4786	
307510	49117.500000	157500.000000	675000.000000	675000.000000	2.000000	1262	

307510 rows × 8 columns

```
In [92]: cat_feat
```

Out[92]:

	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	NAME_TYPE_SUITE	NAME_INCOME_TYPE	NA
0	0	1	0	1	6	7	
1	0	0	0	0	1	4	
2	1	1	1	1	6	7	
3	0	0	0	1	6	7	
4	0	1	0	1	6	7	
...	...	...	...	...	...	...	
307506	0	1	0	0	6	7	
307507	0	0	0	1	6	3	
307508	0	0	0	1	6	7	
307509	0	0	0	1	6	1	
307510	0	0	0	0	6	1	

307511 rows × 12 columns

```
In [93]: final_data = pd.concat([num_feat, cat_feat], axis=1, join='inner')

In [94]: final_data
```

Out[94]:

	AMT_ANNUITY	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_GOODS_PRICE	CNT_FAM_MEMBERS	DAYS_EMPLOYED	DAYS_ID_PU
1	35698.500000	200000.000000	1293502.500000	1084500.000000	2.000000	1188	
2	6750.000000	67500.000000	135000.000000	135000.000000	1.000000	225	
3	29686.500000	135000.000000	312682.500000	297000.000000	2.000000	3039	
4	21865.500000	121500.000000	513000.000000	513000.000000	1.000000	3038	
5	27517.500000	99000.000000	490495.500000	454500.000000	2.000000	1588	
...	...	...	...	...	...	...	
307506	27558.000000	157500.000000	254700.000000	225000.000000	1.000000	236	
307507	12001.500000	72000.000000	269550.000000	225000.000000	1.000000	13389	
307508	29979.000000	153000.000000	677664.000000	585000.000000	1.000000	7921	
307509	20205.000000	171000.000000	370107.000000	319500.000000	2.000000	4786	
307510	49117.500000	157500.000000	675000.000000	675000.000000	2.000000	1262	

307510 rows × 20 columns

```
In [95]: final_data.isnull().sum()

Out[95]: AMT_ANNUITY                   0
         AMT_INCOME_TOTAL              0
         AMT_CREDIT                    0
         AMT_GOODS_PRICE               0
         CNT_FAM_MEMBERS               2
         DAYS_EMPLOYED                 0
         DAYS_ID_PUBLISH               0
         DAYS_REGISTRATION             0
         NAME_CONTRACT_TYPE            0
         CODE_GENDER                   0
         FLAG_OWN_CAR                  0
         FLAG_OWN_REALTY               0
         NAME_TYPE_SUITE               0
         NAME_INCOME_TYPE              0
         NAME_EDUCATION_TYPE           0
         NAME_FAMILY_STATUS            0
         NAME_HOUSING_TYPE             0
         WEEKDAY_APPR_PROCESS_START    0
         ORGANIZATION_TYPE             0
         TARGET                        0
         dtype: int64

In [96]: final_data['CNT_FAM_MEMBERS'].fillna(final_data['CNT_FAM_MEMBERS'].median() ,inplace=True)

In [97]: X = final_data.drop(['TARGET'],axis =1)
         y = final_data['TARGET']

In [98]: from  sklearn.model_selection import train_test_split

 In [ ]:

In [99]: X_train, X_test, y_train, y_test = train_test_split(X,y ,test_size=0.33, random_state=42)

In [100]: #X_train = final_data

In [101]: X_train.shape

Out[101]: (206031, 19)

In [102]: from sklearn.ensemble import RandomForestClassifier
          model= RandomForestClassifier()

          model.fit(X_train,y_train)

          y_pred = model.predict(X_test)
          from sklearn.metrics import accuracy_score
          score=accuracy_score(y_test,y_pred)
          print("Your Model Accuracy is", score)

          Your Model Accuracy is 0.9186728288611437

In [103]: from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score, c
          onfusion_matrix, classification_report

          RF_roc=roc_auc_score(y_test, y_pred)
          RF_acc = accuracy_score(y_test, y_pred)
          RF_prec = precision_score(y_test, y_pred)
          RF_rec = recall_score(y_test, y_pred)
          RF_f1 = f1_score(y_test, y_pred)
          RF_cls=classification_report(y_test,y_pred)

In [104]: print("precsion_score is ", RF_prec)
          print("roc_auc_score is ", RF_roc)
          print("recall_score is ", RF_rec)
          print("f1_score is ", RF_f1)

          precsion_score is  0.3333333333333333
          roc_auc_score is  0.5000498648655191
          recall_score is  0.0001211827435773146
          f1_score is  0.00024227740763173837

In [105]: print('classification_report is', RF_cls)

          classification_report is               precision    recall  f1-score   support

                     0       0.92      1.00      0.96     93227
                     1       0.33      0.00      0.00      8252

              accuracy                           0.92    101479
             macro avg       0.63      0.50      0.48    101479
          weighted avg       0.87      0.92      0.88    101479


In [106]: import collections

          from collections import Counter

          from imblearn.over_sampling import  RandomOverSampler

          over_sample = RandomOverSampler(0.85)

          X_train_ns ,y_train_ns =over_sample.fit_sample(X_train,y_train)
          X_test_ns, y_test_ns = over_sample.fit_sample(X_test,y_test)

          print("The number of Target befor fit {}".format(Counter(y)))

          print("The number of Target after fit {}".format(Counter(y_train_ns)))

          The number of Target befor fit Counter({0: 282686, 1: 24824})
          The number of Target after fit Counter({0: 189459, 1: 161040})

In [107]: from imblearn.combine import SMOTETomek

          from imblearn.over_sampling import SMOTE

          import collections

          from collections import Counter

          counter =Counter(y_train)

          print('Befor', counter)

          smt = SMOTE()

          X_train_sm, y_train_sm = smt.fit_sample(X_train,y_train)
          X_test_sm,y_test_sm =smt.fit_sample(X_test,y_test)
          counter1 = Counter(y_train_sm)

          print('after',counter1)

          Befor Counter({0: 189459, 1: 16572})
          after Counter({0: 189459, 1: 189459})

In [108]: model= RandomForestClassifier(n_estimators = 42,criterion='gini')

          new =model.fit(X_train_sm,y_train_sm)

          y_pred1 = model.predict(X_test_sm)

          from sklearn.metrics import accuracy_score

          score=accuracy_score(y_test_sm,y_pred1)

          print("Your Model Accuracy is", score)

          Your Model Accuracy is 0.8774764821350038

In [109]: RF_cls1=classification_report(y_test_sm,y_pred1)

In [110]: print(RF_cls1)

                        precision    recall  f1-score   support

                     0       0.82      0.97      0.89     93227
                     1       0.97      0.78      0.86     93227

              accuracy                           0.88    186454
             macro avg       0.89      0.88      0.88    186454
          weighted avg       0.89      0.88      0.88    186454


In [111]: print(confusion_matrix(y_pred1,y_test_sm))

          [[90784 20402]
           [ 2443 72825]]

In [112]: from sklearn.model_selection import StratifiedKFold

          stfold=StratifiedKFold(n_splits=4)

          model = RandomForestClassifier()

          from sklearn.model_selection import cross_val_score

          score =cross_val_score(model,X_train_sm,y_train_sm,cv =stfold)

In [113]: score

Out[113]: array([0.82094373, 0.96777156, 0.96786623, 0.96689504])

In [114]: RF_roc2=roc_auc_score(y_test_sm, y_pred1)
          RF_acc2= accuracy_score(y_test_sm, y_pred1)
          RF_prec2 = precision_score(y_test_sm, y_pred1)
          RF_rec2 = recall_score(y_test_sm, y_pred1)
          RF_f12 = f1_score(y_test_sm, y_pred1)
          RF_cls2=classification_report(y_test_sm,y_pred1)

In [115]: print("precsion_score is ", RF_prec2)
          print("roc_auc_score is ", RF_roc2)
          print("recall_score is ", RF_rec2)
          print("f1_score is ", RF_f12)

          precsion_score is  0.9675426476058883
          roc_auc_score is  0.877476482135004
          recall_score is  0.7811578190867453
          f1_score is  0.8644173417608831

In [116]: #decsion tree.
          from sklearn.tree import DecisionTreeClassifier
          dtree = DecisionTreeClassifier()

In [117]: tree = dtree.fit(X_train_sm,y_train_sm)
          y_pred = dtree.predict(X_test_sm)
          print("Classification report - \n", classification_report(y_test_sm,y_pred))
          print(confusion_matrix(y_pred,y_test_sm))

          Classification report -
                         precision    recall  f1-score   support

                     0       0.81      0.85      0.83     93227
                     1       0.85      0.80      0.82     93227

              accuracy                           0.83    186454
             macro avg       0.83      0.83      0.83    186454
          weighted avg       0.83      0.83      0.83    186454

          [[79593 18685]
           [13634 74542]]

In [118]: #adabosting
          from sklearn.ensemble import AdaBoostClassifier

In [119]: ab = AdaBoostClassifier(n_estimators=1000, random_state=0)

In [120]: ab.fit(X_train_sm,y_train_sm)
          y_pred2=ab.predict(X_test_sm)
          print("Classification report - \n", classification_report(y_test_sm,y_pred2))
          print(confusion_matrix(y_pred2,y_test_sm))

          Classification report -
                         precision    recall  f1-score   support

                     0       0.85      0.95      0.90     93227
                     1       0.95      0.83      0.89     93227

              accuracy                           0.89    186454
             macro avg       0.90      0.89      0.89    186454
          weighted avg       0.90      0.89      0.89    186454

          [[88881 15552]
           [ 4346 77675]]

In [121]: #gradient boosting
          from sklearn.ensemble import GradientBoostingClassifier

In [122]: GB = GradientBoostingClassifier()

In [123]: GB.fit(X_train_sm,y_train_sm)
          y_pred3=GB.predict(X_test_sm)
          print("Classification report - \n", classification_report(y_test_sm,y_pred3))
          print(confusion_matrix(y_pred3,y_test_sm))

          Classification report -
                         precision    recall  f1-score   support

                     0       0.83      0.95      0.89     93227
                     1       0.94      0.81      0.87     93227

              accuracy                           0.88    186454
             macro avg       0.89      0.88      0.88    186454
          weighted avg       0.89      0.88      0.88    186454

          [[88665 18064]
           [ 4562 75163]]

In [124]: #checking with pycaret

          #from pycaret.classification import *
          #exp_clf = setup(final_data, target = 'TARGET')

In [141]: #exp_clf = setup(final_data, target = 'TARGET',html= False)

In [140]: #best = compare_models()

In [139]: #from sklearn.grid_search import GridSearchCV

In [146]: from sklearn.model_selection import GridSearchCV
          from sklearn.datasets import make_classification
          from sklearn.ensemble import RandomForestClassifier
          # Build a classification task using 3 informative features
          X_train_sm, y_train_sm = make_classification(n_samples=1000,
                                     n_features=20,
                                     n_informative=3,
                                     n_redundant=0,
                                     n_repeated=0,
                                     n_classes=2,
                                     random_state=0,
                                     shuffle=False)


          rfc = RandomForestClassifier(n_jobs=-1,max_features= 'sqrt' ,n_estimators=700, oob_score = True)

          param_grid = {
              'n_estimators': [200, 700],
              'max_features': ['auto', 'sqrt', 'log2']
          }

          CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5,scoring ='accuracy')
          CV_rfc.fit(X_train_sm, y_train_sm)
          print (CV_rfc.best_params_)

          {'max_features': 'auto', 'n_estimators': 700}

In [131]: CV_rfc.best_params_

Out[131]: {'max_features': 'sqrt', 'n_estimators': 700}

In [143]: accuracy=CV_rfc.best_score_

In [144]: accuracy

Out[144]: 0.8710000000000001

 In [ ]:

 In [ ]:

 In [ ]: #scale down the features

 In [ ]: #using train test split ,split the data into training and testing

 In [ ]: #create the logistic regrsion model

 In [ ]: # handle the data imbalancing using randomOversampling and smote for x_train,y_train,x_test,y_test

 In [ ]: #apply the machine learning models

 In [ ]: #features importance

 In [ ]: #usning flask craeate a web application and test
```