home_credit_default_analysis_project

September 9, 2018

1 Home Credit Default Risk

1.1 Overview

For financial institutes like banks giving loan to customers is a complicated process. Bank want to ensure that it gives loans to those customers who have low risk. If the customer defaults in repaying loans it will be a loss to Bank. That is why Banks perform extensive credit risk analysis before approving the loan to customer.

In this capstone project I have chosen Kaggle competition challenge "Home Credit Default Analysis" where I also participated in the competition. The URL for the competition is: https://www.kaggle.com/c/home-credit-default-risk

Home Credit is a global financial institute which provides loans to lender. Home Credit operates in 10 countries globally Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities. While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful. I have analysed the data provided by Home Credit data using statistical techniques. Next I have applied Machine Learning model to predict risk of each customer. My ultimate objective is to get the model that has best ROC-AUC score

1.2 Preprocess Datasets:

In this step I have first all the loaded datasets given: * application_train.csv into dataframe application_train_data * application_test.csv into dataframe application_test_data * bureau.csv into into dataframe bureau * bureau_balance.csv into dataframe bureau_balance * POS_CASH_balance.csv into dataframe POS_CASH_balance * credit_card_balance.csv into dataframe credit_card_balance * previous_application.csv into dataframe previous_application * installments_payments.csv into dataframe installments_payments

Next, get separate dataframes from bureau.csv based on values in CREDIT_ACTIVE field which are: * Active as active_bereau_credit dataframe * Closed as closed_bereau_credit dataframe * Sold as sold_bereau_credit dataframe * Bad debt as bad_debt_bereau_credit dataframe

Next, get total count of each type of CREDIT_ACTIVE group by SK_ID_CUR and load into dataframe bureau_credit_count

Next merge dataframe application_train_data with each of the dataframes active_bereau_credit dataframe, closed_bereau_credit dataframe, sold_bereau_credit dataframe, bad_debt_bereau_credit dataframe by performing left join on field SK_ID_CURR and crea a new dataframe application_bureau_train_data

The above step do with application_train_data with the same set of bereau dataframes and create a new dataframe dataframe application_bureau_test_data

Similarly create separate dataframes for each contract type in previous_application (i.e. Cash Loans, Consumer Loans, Revolving Loans, XNA). Create another dataframe which will have count of number of each type of contract_type in previous_application group by SK_ID_CUR. With application_bureau_train_data merge these dataframes by performing left join on SK_ID_CUR and create new dataframe application_bureau_loan_train_data. Similarly, with application_bureau_test_data merge these dataframes by performing left join on SK_ID_CUR and create new dataframe application_bureau_loan_test_data

Note: While modelling I have not used a few datasets like installment_payments, as these seem to me transacational dataset

The dataframe application_bureau_loan_train_data will be used for further analysis. When any transformations or adding/deleting fields is done on application_bureau_loan_train_data, the same will be applied on application_bureau_test_data, as this is used for Kaggle public ranking

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

In [2]:
```
pd.set_option('display.height', 1000)
pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1500)
```

In [3]:
```
# load application_train.csv into application_train_data dataframe
application_train_data = pd.read_csv('all/application_train.csv')
print('Training data shape:',application_train_data.shape)
```

```
Training data shape: (307511, 122)
```

In [4]:
```
application_train_data.head()
```

Out[4]:
```
   SK_ID_CURR  TARGET NAME_CONTRACT_TYPE CODE_GENDER FLAG_OWN_CAR FLAG_OWN_REALTY  CNT_
0      100002       1         Cash loans           M            N               Y
1      100003       0         Cash loans           F            N               N
2      100004       0    Revolving loans           M            Y               Y
3      100006       0         Cash loans           F            N               Y
4      100007       0         Cash loans           M            N               Y

   LANDAREA_MEDI  LIVINGAPARTMENTS_MEDI  LIVINGAREA_MEDI  NONLIVINGAPARTMENTS_MEDI  NON
0         0.0375                 0.0205           0.0193                    0.0000
```

```
1        0.0132                0.0787        0.0558                0.0039
2           NaN                   NaN           NaN                   NaN
3           NaN                   NaN           NaN                   NaN
4           NaN                   NaN           NaN                   NaN
```

In [5]:
```
# Get all the columns of application_train_data
application_train_data.columns.values.tolist()
```

Out[5]:
```
['SK_ID_CURR',
 'TARGET',
 'NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'AMT_ANNUITY',
 'AMT_GOODS_PRICE',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'OWN_CAR_AGE',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'OCCUPATION_TYPE',
 'CNT_FAM_MEMBERS',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'WEEKDAY_APPR_PROCESS_START',
 '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',
```

```
'ORGANIZATION_TYPE',
'EXT_SOURCE_1',
'EXT_SOURCE_2',
'EXT_SOURCE_3',
'APARTMENTS_AVG',
'BASEMENTAREA_AVG',
'YEARS_BEGINEXPLUATATION_AVG',
'YEARS_BUILD_AVG',
'COMMONAREA_AVG',
'ELEVATORS_AVG',
'ENTRANCES_AVG',
'FLOORSMAX_AVG',
'FLOORSMIN_AVG',
'LANDAREA_AVG',
'LIVINGAPARTMENTS_AVG',
'LIVINGAREA_AVG',
'NONLIVINGAPARTMENTS_AVG',
'NONLIVINGAREA_AVG',
'APARTMENTS_MODE',
'BASEMENTAREA_MODE',
'YEARS_BEGINEXPLUATATION_MODE',
'YEARS_BUILD_MODE',
'COMMONAREA_MODE',
'ELEVATORS_MODE',
'ENTRANCES_MODE',
'FLOORSMAX_MODE',
'FLOORSMIN_MODE',
'LANDAREA_MODE',
'LIVINGAPARTMENTS_MODE',
'LIVINGAREA_MODE',
'NONLIVINGAPARTMENTS_MODE',
'NONLIVINGAREA_MODE',
'APARTMENTS_MEDI',
'BASEMENTAREA_MEDI',
'YEARS_BEGINEXPLUATATION_MEDI',
'YEARS_BUILD_MEDI',
'COMMONAREA_MEDI',
'ELEVATORS_MEDI',
'ENTRANCES_MEDI',
'FLOORSMAX_MEDI',
'FLOORSMIN_MEDI',
'LANDAREA_MEDI',
'LIVINGAPARTMENTS_MEDI',
'LIVINGAREA_MEDI',
'NONLIVINGAPARTMENTS_MEDI',
'NONLIVINGAREA_MEDI',
'FONDKAPREMONT_MODE',
'HOUSETYPE_MODE',
```

```
 'TOTALAREA_MODE',
 'WALLSMATERIAL_MODE',
 'EMERGENCYSTATE_MODE',
 '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 [6]:
```
# load application_test.csv into application_test_data dataframe
application_test_data = pd.read_csv('all/application_test.csv')
print('Testing data shape:',application_test_data.shape)
```

```
Testing data shape: (48744, 121)
```

In [7]:
```
application_test_data.head()
```

Out[7]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREI
0	100001	Cash loans	F	N	Y	(
1	100005	Cash loans	M	N	Y	(
2	100013	Cash loans	M	Y	Y	(
3	100028	Cash loans	F	N	Y	:

```
   4      100038         Cash loans           M            Y              N              

      LANDAREA_MEDI  LIVINGAPARTMENTS_MEDI  LIVINGAREA_MEDI  NONLIVINGAPARTMENTS_MEDI  NO
   0            NaN                    NaN           0.0514                       NaN
   1            NaN                    NaN              NaN                       NaN
   2            NaN                    NaN              NaN                       NaN
   3         0.2078                 0.2446           0.3739                    0.0388
   4            NaN                    NaN              NaN                       NaN
```

In [9]:
```python
bureau = pd.read_csv('all/bureau.csv')
print('Bureau data shape:',bureau.shape)
bureau.head()
```

```
Bureau data shape: (1716428, 17)
```

Out[9]:
```
      SK_ID_CURR  SK_ID_BUREAU CREDIT_ACTIVE CREDIT_CURRENCY  DAYS_CREDIT  CREDIT_DAY_OVE
   0      215354       5714462        Closed      currency 1         -497
   1      215354       5714463        Active      currency 1         -208
   2      215354       5714464        Active      currency 1         -203
   3      215354       5714465        Active      currency 1         -203
   4      215354       5714466        Active      currency 1         -629
```

In [10]:
```python
# Get all the types of CREDIT_ACTIVE
bureau['CREDIT_ACTIVE'].unique()
```

Out[10]:
```
array(['Closed', 'Active', 'Sold', 'Bad debt'], dtype=object)
```

In [11]:
```python
# Get count of all the types of CREDIT_ACTIVE
bureau['CREDIT_ACTIVE'].value_counts()
```

Out[11]:
```
Closed      1079273
Active       630607
Sold           6527
Bad debt         21
Name: CREDIT_ACTIVE, dtype: int64
```

In [12]:
```python
np.max(bureau['CNT_CREDIT_PROLONG'])
```

Out[12]:
```
9
```

In [13]:
```python
# Get summary of all the Active credit details
active_bereau_credit = bureau[bureau.CREDIT_ACTIVE=='Active'].groupby(['SK_ID_CURR'],
active_bereau_credit.head()
```

Out[13]:
```
      SK_ID_CURR  SK_ID_BUREAU  DAYS_CREDIT  CREDIT_DAY_OVERDUE  DAYS_CREDIT_ENDDATE  DA
   0      100001      17689905         -928                   0               3091.0
   1      100002      12317812        -1145                   0                780.0
   2      100003       5885880         -606                   0               1216.0
   3      100005      13470403         -199                   0               1446.0
   4      100008       6491434          -78                   0                471.0
```

In [14]: `np.max(active_bereau_credit['CNT_CREDIT_PROLONG'])`

Out[14]: 9

In [15]: `np.min(active_bereau_credit['DAYS_CREDIT_ENDDATE'])`

Out[15]: -83445.0

In [14]: `np.min(active_bereau_credit['DAYS_ENDDATE_FACT'])`

Out[14]: -8664.0

In [16]:
```
# Transform fields of active_bereau_credit
active_bereau_credit['DAYS_CREDIT'] =  active_bereau_credit['DAYS_CREDIT']/365.0
active_bereau_credit['DAYS_CREDIT_ENDDATE'] = active_bereau_credit['DAYS_CREDIT_ENDDA
active_bereau_credit['DAYS_ENDDATE_FACT'] =  active_bereau_credit['DAYS_ENDDATE_FACT']
active_bereau_credit['CREDIT_DAY_OVERDUE'] = active_bereau_credit['CREDIT_DAY_OVERDUE

active_bereau_credit = active_bereau_credit.rename(columns= {'DAYS_CREDIT': 'YEARS_CR
                                                            'DAYS_CREDIT_ENDDATE': '
                                                            'DAYS_ENDDATE_FACT':'YEA
                                                            'CREDIT_DAY_OVERDUE':'CR
                                                            'AMT_CREDIT_SUM': 'AMT_C
                                                            'AMT_CREDIT_SUM_DEBT':'A
                                                            'AMT_CREDIT_SUM_LIMIT':'
                                                            'AMT_CREDIT_MAX_OVERDUE'
                                                            'AMT_CREDIT_SUM_OVERDUE'
                                                            'AMT_ANNUITY':'AMT_ANNUI
                                                            'CNT_CREDIT_PROLONG': 'C
                                                           })

active_bereau_credit.drop(['SK_ID_BUREAU',  'DAYS_CREDIT_UPDATE'], axis=1, inplace=Tr

active_bereau_credit.fillna(0, inplace=True)
active_bereau_credit.head()
```

Out[16]:

	SK_ID_CURR	YEARS_CREDIT_ACTIVE	CREDIT_YEAR_OVERDUE_ACTIVE	YEARS_CREDIT_ENDDATE_
0	100001	-2.542466	0.0	8.
1	100002	-3.136986	0.0	2.
2	100003	-1.660274	0.0	3.
3	100005	-0.545205	0.0	3.
4	100008	-0.213699	0.0	1.

In [17]:
```
# Get summary of all the Closed credit details
closed_bereau_credit = bureau[bureau.CREDIT_ACTIVE=='Closed'].groupby(['SK_ID_CURR'],
closed_bereau_credit.head()
```

Out[17]:

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DA
0	100001	23586526	-4217	0	-2514.0	

1	100002	36908365	-5847	0	-2874.0
2	100003	17657634	-4997	0	-3394.0
3	100004	13658267	-1734	0	-977.0
4	100005	6735200	-373	0	-128.0

In [18]:
```
np.max(closed_bereau_credit['CNT_CREDIT_PROLONG'])
```

Out[18]: 6

In [19]:
```
# Transform fields of closed_bereau_credit
closed_bereau_credit['DAYS_CREDIT'] = closed_bereau_credit['DAYS_CREDIT']/365.0
closed_bereau_credit['DAYS_CREDIT_ENDDATE'] =  closed_bereau_credit['DAYS_CREDIT_ENDDA
closed_bereau_credit['DAYS_ENDDATE_FACT'] =  closed_bereau_credit['DAYS_ENDDATE_FACT']
closed_bereau_credit['CREDIT_DAY_OVERDUE'] = closed_bereau_credit['CREDIT_DAY_OVERDUE

closed_bereau_credit = closed_bereau_credit.rename(columns= {'DAYS_CREDIT': 'YEARS_CRE
                                                          'DAYS_CREDIT_ENDDATE': 'YEAR
                                                          'DAYS_ENDDATE_FACT':'YEARS_E
                                                          'CREDIT_DAY_OVERDUE':'CREDIT
                                                          'AMT_CREDIT_SUM': 'AMT_CREDI
                                                          'AMT_CREDIT_SUM_DEBT':'AMT_C
                                                          'AMT_CREDIT_SUM_LIMIT':'AMT_
                                                          'AMT_CREDIT_MAX_OVERDUE':'AM
                                                          'AMT_CREDIT_SUM_OVERDUE':'AM
                                                          'AMT_ANNUITY':'AMT_ANNUITY_C
                                                          'CNT_CREDIT_PROLONG': 'CNT_C
                                                            })

closed_bereau_credit.drop(['SK_ID_BUREAU',   'DAYS_CREDIT_UPDATE'], axis=1, inplace=Tr
closed_bereau_credit.fillna(0, inplace=True)

closed_bereau_credit.head()
```

Out[19]:

	SK_ID_CURR	YEARS_CREDIT_CLOSED	CREDIT_YEAR_OVERDUE_CLOSED	YEARS_CREDIT_ENDDATE_C
0	100001	-11.553425	0.0	-6.8
1	100002	-16.019178	0.0	-7.8
2	100003	-13.690411	0.0	-9.2
3	100004	-4.750685	0.0	-2.6
4	100005	-1.021918	0.0	-0.3

In [20]:
```
# Get summary of all the Sold credit details
sold_bereau_credit = bureau[bureau.CREDIT_ACTIVE=='Sold'].groupby(['SK_ID_CURR'], as_i
sold_bereau_credit.head()
```

Out[20]:

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DAY
0	100039	5153449	-1206	0	-980.0	
1	100128	5941041	-2641	0	-1987.0	
2	100162	6131361	-1998	0	-1272.0	
3	100170	5915577	-147	0	0.0	
4	100201	5928807	-2270	0	-1907.0	

```
In [21]: np.max(sold_bereau_credit['CNT_CREDIT_PROLONG'])
```

```
Out[21]: 1
```

```
In [22]: # Transform fields of sold_bereau_credit
         sold_bereau_credit['DAYS_CREDIT'] =  sold_bereau_credit['DAYS_CREDIT']/365.0
         sold_bereau_credit['DAYS_CREDIT_ENDDATE'] = sold_bereau_credit['DAYS_CREDIT_ENDDATE']/
         sold_bereau_credit['DAYS_ENDDATE_FACT'] =  sold_bereau_credit['DAYS_ENDDATE_FACT']/365
         sold_bereau_credit['CREDIT_DAY_OVERDUE'] = sold_bereau_credit['CREDIT_DAY_OVERDUE']/36

         sold_bereau_credit = sold_bereau_credit.rename(columns= {'DAYS_CREDIT': 'YEARS_CREDIT_
                                                                  'DAYS_CREDIT_ENDDATE': 'YEAR
                                                                  'DAYS_ENDDATE_FACT':'YEARS_E
                                                                  'CREDIT_DAY_OVERDUE':'CREDIT
                                                                  'AMT_CREDIT_SUM': 'AMT_CREDI
                                                                  'AMT_CREDIT_SUM_DEBT':'AMT_C
                                                                  'AMT_CREDIT_SUM_LIMIT':'AMT_
                                                                  'AMT_CREDIT_MAX_OVERDUE':'AM
                                                                  'AMT_CREDIT_SUM_OVERDUE':'AM
                                                                   'AMT_ANNUITY':'AMT_ANNUITY_
                                                                   'CNT_CREDIT_PROLONG': 'CNT_
                                                                    })

         sold_bereau_credit.drop(['SK_ID_BUREAU', 'DAYS_CREDIT_UPDATE'], axis=1, inplace=True)
         sold_bereau_credit.fillna(0, inplace=True)

         sold_bereau_credit.head()
```

Out[22]:

	SK_ID_CURR	YEARS_CREDIT_SOLD	CREDIT_YEAR_OVERDUE_SOLD	YEARS_CREDIT_ENDDATE_SOLD
0	100039	-3.304110	0.0	-2.684932
1	100128	-7.235616	0.0	-5.443836
2	100162	-5.473973	0.0	-3.484932
3	100170	-0.402740	0.0	0.000000
4	100201	-6.219178	0.0	-5.224658

```
In [23]: # Get summary of all the bad debt credit details
         bad_debt_bereau_credit = bureau[bureau.CREDIT_ACTIVE=='Bad debt'].groupby(['SK_ID_CURR
         bad_debt_bereau_credit.head(10)
```

Out[23]:

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DA
0	158069	6039562	-1683	366	-862.0	
1	163442	5997537	-1502	366	-1292.0	
2	176952	5326184	-2241	1135	-1876.0	
3	186360	5499851	-1218	366	-852.0	
4	207535	5300044	-2834	0	-1724.0	
5	231185	5173404	-2740	1761	-2558.0	
6	232061	6441729	-2899	0	-1773.0	
7	243877	6446445	-2493	366	-898.0	
8	264970	5345303	-2728	0	-2514.0	
9	273612	5309530	-2112	0	-1900.0	

In [24]:
```
np.max(bad_debt_bereau_credit['CNT_CREDIT_PROLONG'])
```

Out[24]: 1

In [25]:
```
# Transform fields of bad_debt_bereau_credit
bad_debt_bereau_credit['DAYS_CREDIT'] =  bad_debt_bereau_credit['DAYS_CREDIT']/365.0
bad_debt_bereau_credit['DAYS_CREDIT_ENDDATE'] =  bad_debt_bereau_credit['DAYS_CREDIT_I
bad_debt_bereau_credit['DAYS_ENDDATE_FACT'] =  bad_debt_bereau_credit['DAYS_ENDDATE_F/
bad_debt_bereau_credit['CREDIT_DAY_OVERDUE'] = bad_debt_bereau_credit['CREDIT_DAY_OVEI

bad_debt_bereau_credit = bad_debt_bereau_credit.rename(columns= {'DAYS_CREDIT': 'YEARS
                                                          'DAYS_CREDIT_ENDDATE': 'YEAI
                                                          'DAYS_ENDDATE_FACT':'YEARS_I
                                                          'CREDIT_DAY_OVERDUE':'CREDI
                                                          'AMT_CREDIT_SUM': 'AMT_CREDI
                                                          'AMT_CREDIT_SUM_DEBT':'AMT_(
                                                          'AMT_CREDIT_SUM_LIMIT':'AMT_
                                                          'AMT_CREDIT_MAX_OVERDUE':'AI
                                                          'AMT_CREDIT_SUM_OVERDUE':'AI
                                                           'AMT_ANNUITY':'AMT_ANNUITY_
                                                            'CNT_CREDIT_PROLONG': 'CN
                                                            })

bad_debt_bereau_credit.drop(['SK_ID_BUREAU', 'DAYS_CREDIT_UPDATE'], axis=1, inplace=T:
bad_debt_bereau_credit.fillna(0, inplace=True)
bad_debt_bereau_credit.head()
```

Out[25]:

	SK_ID_CURR	YEARS_CREDIT_BAD_DEBT	CREDIT_YEAR_OVERDUE_BAD_DEBT	YEARS_CREDIT_ENDD/
0	158069	-4.610959	1.002740	
1	163442	-4.115068	1.002740	
2	176952	-6.139726	3.109589	
3	186360	-3.336986	1.002740	
4	207535	-7.764384	0.000000	

In [26]:
```
# Group count of Active, Bad_debt, Closed, Sold by SK_ID_CURR
bureau_credit_count = bureau.pivot_table(index=['SK_ID_CURR'], columns='CREDIT_ACTIVE
bureau_credit_count = bureau_credit_count.rename(columns= {"Bad debt":"Bad_debt"})
bureau_credit_count.fillna(0, inplace=True)
bureau_credit_count.head()
```

Out[26]:

CREDIT_ACTIVE	SK_ID_CURR	Active	Bad_debt	Closed	Sold
0	100001	3	0	4	0
1	100002	2	0	6	0
2	100003	1	0	3	0
3	100004	0	0	2	0
4	100005	2	0	1	0

In [27]:
```
# Merge application_train_data with all the bereau information and make new dataframe
application_bureau_train_data = pd.merge(application_train_data , active_bereau_credi
```

```
application_bureau_train_data = pd.merge(application_bureau_train_data, closed_bereau_
application_bureau_train_data = pd.merge(application_bureau_train_data, sold_bereau_c
application_bureau_train_data = pd.merge(application_bureau_train_data, bad_debt_bere
application_bureau_train_data = pd.merge(application_bureau_train_data, bureau_credit_
application_bureau_train_data.head()
```

Out[27]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CN
0	100002	1	Cash loans	M	N	Y	
1	100003	0	Cash loans	F	N	N	
2	100004	0	Revolving loans	M	Y	Y	
3	100006	0	Cash loans	F	N	Y	
4	100007	0	Cash loans	M	N	Y	

	LANDAREA_MEDI	LIVINGAPARTMENTS_MEDI	LIVINGAREA_MEDI	NONLIVINGAPARTMENTS_MEDI	N
0	0.0375	0.0205	0.0193	0.0000	
1	0.0132	0.0787	0.0558	0.0039	
2	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	

	YEARS_ENDDATE_FACT_SOLD	AMT_CREDIT_MAX_OVERDUE_SOLD	CNT_CREDIT_PROLONG_SOLD	AMT_
0	NaN	NaN	NaN	
1	NaN	NaN	NaN	
2	NaN	NaN	NaN	
3	NaN	NaN	NaN	
4	NaN	NaN	NaN	

In [28]:

```
# Merge application_train_data with all the bereau information and make new dataframe
application_bureau_test_data = pd.merge(application_test_data , active_bereau_credit,
application_bureau_test_data = pd.merge(application_bureau_test_data, closed_bereau_c
application_bureau_test_data = pd.merge(application_bureau_test_data, sold_bereau_cre
application_bureau_test_data = pd.merge(application_bureau_test_data, bad_debt_bereau
application_bureau_test_data = pd.merge(application_bureau_test_data, bureau_credit_c
application_bureau_test_data.head()
```

Out[28]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDR
0	100001	Cash loans	F	N	Y	
1	100005	Cash loans	M	N	Y	
2	100013	Cash loans	M	Y	Y	
3	100028	Cash loans	F	N	Y	
4	100038	Cash loans	M	Y	N	

	LANDAREA_MEDI	LIVINGAPARTMENTS_MEDI	LIVINGAREA_MEDI	NONLIVINGAPARTMENTS_MEDI	N
0	NaN	NaN	0.0514	NaN	
1	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	
3	0.2078	0.2446	0.3739	0.0388	
4	NaN	NaN	NaN	NaN	

	YEARS_ENDDATE_FACT_SOLD	AMT_CREDIT_MAX_OVERDUE_SOLD	CNT_CREDIT_PROLONG_SOLD	AMT_
0	NaN	NaN	NaN	
1	NaN	NaN	NaN	
2	NaN	NaN	NaN	
3	NaN	NaN	NaN	
4	NaN	NaN	NaN	

In [29]:
```
bureau_balance_data = pd.read_csv('all/bureau_balance.csv')
print('Bureau Balance data shape:',bureau_balance_data.shape)
bureau_balance_data.head(10)
```

Bureau Balance data shape: (27299925, 3)

Out[29]:

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
0	5715448	0	C
1	5715448	-1	C
2	5715448	-2	C
3	5715448	-3	C
4	5715448	-4	C
5	5715448	-5	C
6	5715448	-6	C
7	5715448	-7	C
8	5715448	-8	C
9	5715448	-9	0

In [31]:
```
# Load previous_application.csv into dataframe previous_application
previous_application = pd.read_csv('all/previous_application.csv')
print('Previous Application shape:',previous_application.shape)
previous_application.head()
```

Previous Application shape: (1670214, 37)

Out[31]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.(
1	2802425	108129	Cash loans	25188.615	607500.0	679671.(
2	2523466	122040	Cash loans	15060.735	112500.0	136444.!
3	2819243	176158	Cash loans	47041.335	450000.0	470790.(
4	1784265	202054	Cash loans	31924.395	337500.0	404055.(

In [32]:
```
# Get count of all the loan Contract Types
previous_application['NAME_CONTRACT_TYPE'].value_counts()
```

Out[32]:
```
Cash loans         747553
Consumer loans     729151
Revolving loans    193164
XNA                   346
Name: NAME_CONTRACT_TYPE, dtype: int64
```

In [33]:
```
# Get summary of all the cash loan information
previous_application_cash_loan = previous_application[previous_application.NAME_CONTRA
previous_application_cash_loan = previous_application_cash_loan[['SK_ID_CURR', 'AMT_A
previous_application_cash_loan = previous_application_cash_loan.rename(columns={'AMT_
previous_application_cash_loan.fillna(0)
previous_application_cash_loan.head()
```

Out[33]:

	SK_ID_CURR	PREV_CASH_AMT_ANNUITY	PREV_CASH_AMT_APPLICATION	PREV_CASH_AMT_CREDIT
0	100003	98356.995	900000.0	1035882.0
1	100005	0.000	0.0	0.0
2	100006	96896.610	1818000.0	2063110.5
3	100007	68237.010	855000.0	954553.5
4	100008	25309.575	450000.0	501975.0

In [34]:
```
# Get summary of all the consumer loan information
previous_application_consumer_loan = previous_application[previous_application.NAME_C
previous_application_consumer_loan = previous_application_consumer_loan[['SK_ID_CURR'
previous_application_consumer_loan = previous_application_consumer_loan.rename(column
previous_application_consumer_loan.fillna(0)
previous_application_consumer_loan.head()
```

Out[34]:

	SK_ID_CURR	PREV_CONSUMER_AMT_ANNUITY	PREV_CONSUMER_AMT_APPLICATION	PREV_CONSUME
0	100001	3951.000	24835.5	
1	100002	9251.775	179055.0	
2	100003	71304.975	406309.5	
3	100004	5357.250	24282.0	
4	100005	4813.200	44617.5	

In [35]:
```
# Get summary of all the revolving loan information
previous_application_revolving_loan = previous_application[previous_application.NAME_
previous_application_revolving_loan = previous_application_revolving_loan[['SK_ID_CUR
previous_application_revolving_loan = previous_application_revolving_loan.rename(colu
previous_application_revolving_loan.fillna(0)
previous_application_revolving_loan.head()
```

Out[35]:

	SK_ID_CURR	PREV_REVOVING_AMT_ANNUITY	PREV_REVOLVING_AMT_APPLICATION	PREV_REVOLV
0	100006	13500.0	270000.0	
1	100011	9000.0	0.0	
2	100021	33750.0	0.0	
3	100023	2250.0	45000.0	
4	100028	11250.0	0.0	

In [36]:
```
# Get summary of all the XNA loan information
previous_application_XNA_loan = previous_application[previous_application.NAME_CONTRAC
previous_application_XNA_loan = previous_application_XNA_loan[['SK_ID_CURR', 'AMT_ANNU
previous_application_XNA_loan = previous_application_XNA_loan.rename(columns={'AMT_ANN
previous_application_XNA_loan.fillna(0)
previous_application_XNA_loan.head()
```

Out[36]:

	SK_ID_CURR	PREV_XNA_AMT_ANNUITY	PREV_XNA_AMT_APPLICATION	PREV_XNA_AMT_CREDIT	PI
0	100523	0.0	0.0	0.0	
1	101728	0.0	0.0	0.0	
2	103244	0.0	0.0	0.0	
3	103715	0.0	0.0	0.0	
4	105000	0.0	0.0	0.0	

In [37]:

```
# Group count of Active, Bad_debt, Closed, Sold by SK_ID_CURR
previous_application_loan_count = previous_application.pivot_table(index=['SK_ID_CURR
previous_application_loan_count = previous_application_loan_count.rename(columns= {"Ca
previous_application_loan_count.fillna(0)
previous_application_loan_count.head()
```

Out[37]:

NAME_CONTRACT_TYPE	SK_ID_CURR	CASH_LOANS	CONSUMER_LOANS	REVOLVING_LOANS	XNA
0	100001	0	1	0	0
1	100002	0	1	0	0
2	100003	1	2	0	0
3	100004	0	1	0	0
4	100005	1	1	0	0

In [38]:

```
# Merge all the previous application loan data with train and bereau data to create n
application_bureau_loan_train_data = pd.merge(application_bureau_train_data , previous
application_bureau_loan_train_data = pd.merge(application_bureau_loan_train_data , pre
application_bureau_loan_train_data = pd.merge(application_bureau_loan_train_data , pre
application_bureau_loan_train_data = pd.merge(application_bureau_loan_train_data , pre
application_bureau_loan_train_data = pd.merge(application_bureau_loan_train_data , pre
application_bureau_loan_train_data.head()
```

Out[38]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CN
0	100002	1	Cash loans	M	N	Y	
1	100003	0	Cash loans	F	N	N	
2	100004	0	Revolving loans	M	Y	Y	
3	100006	0	Cash loans	F	N	Y	
4	100007	0	Cash loans	M	N	Y	

	LANDAREA_MEDI	LIVINGAPARTMENTS_MEDI	LIVINGAREA_MEDI	NONLIVINGAPARTMENTS_MEDI	N
0	0.0375	0.0205	0.0193	0.0000	
1	0.0132	0.0787	0.0558	0.0039	
2	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	

	YEARS_ENDDATE_FACT_SOLD	AMT_CREDIT_MAX_OVERDUE_SOLD	CNT_CREDIT_PROLONG_SOLD	AMT_
0	NaN	NaN	NaN	
1	NaN	NaN	NaN	
2	NaN	NaN	NaN	
3	NaN	NaN	NaN	
4	NaN	NaN	NaN	

```
In [39]: # Merge all the previous application loan data with test and bereau data to create ne
         application_bureau_loan_test_data = pd.merge(application_bureau_test_data , previous_a
         application_bureau_loan_test_data = pd.merge(application_bureau_loan_test_data , previ
         application_bureau_loan_test_data = pd.merge(application_bureau_loan_test_data , previ
         application_bureau_loan_test_data = pd.merge(application_bureau_loan_test_data , previ
         application_bureau_loan_test_data = pd.merge(application_bureau_loan_test_data , previ
         application_bureau_loan_test_data.head()
```

Out[39]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDRE
0	100001	Cash loans	F	N	Y	
1	100005	Cash loans	M	N	Y	
2	100013	Cash loans	M	Y	Y	
3	100028	Cash loans	F	N	Y	
4	100038	Cash loans	M	Y	N	

	LANDAREA_MEDI	LIVINGAPARTMENTS_MEDI	LIVINGAREA_MEDI	NONLIVINGAPARTMENTS_MEDI	NO
0	NaN	NaN	0.0514	NaN	
1	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	
3	0.2078	0.2446	0.3739	0.0388	
4	NaN	NaN	NaN	NaN	

	YEARS_ENDDATE_FACT_SOLD	AMT_CREDIT_MAX_OVERDUE_SOLD	CNT_CREDIT_PROLONG_SOLD	AMT_
0	NaN	NaN	NaN	
1	NaN	NaN	NaN	
2	NaN	NaN	NaN	
3	NaN	NaN	NaN	
4	NaN	NaN	NaN	

```
In [40]: pos_cash_balance = pd.read_csv('all/POS_CASH_balance.csv')
         print('Pos cash data shape:',pos_cash_balance.shape)
         pos_cash_balance.head()
```

Pos cash data shape: (10001358, 8)

Out[40]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	NAME_
0	1803195	182943	-31	48.0	45.0	
1	1715348	367990	-33	36.0	35.0	
2	1784872	397406	-32	12.0	9.0	
3	1903291	269225	-35	48.0	42.0	
4	2341044	334279	-35	36.0	35.0	

```
In [41]: credit_card_balance = pd.read_csv('all/credit_card_balance.csv')
         print('Credit card balance data shape:',credit_card_balance.shape)
         credit_card_balance.head()
```

Credit card balance data shape: (3840312, 23)

Out[41]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_I
0	2562384	378907	-6	56.970	135000	
1	2582071	363914	-1	63975.555	45000	
2	1740877	371185	-7	31815.225	450000	
3	1389973	337855	-4	236572.110	225000	
4	1891521	126868	-1	453919.455	450000	

In [42]:
```
installments_payments = pd.read_csv('all/installments_payments.csv')
print('Installments payments data shape:',installments_payments.shape)
installments_payments.head()
```

```
Installments payments data shape: (13605401, 8)
```

Out[42]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALI
0	1054186	161674	1.0	6	-118
1	1330831	151639	0.0	34	-215
2	2085231	193053	2.0	1	-6
3	2452527	199697	1.0	3	-241
4	2714724	167756	1.0	2	-138

1.3 Missing Data Analysis

In this step, we first get which all columns have missing values and then calculate percentage of records which have missing values in each column.

Next find out all the columns whose type is string and fill value 'NA' for all the missing values For remainging misssing values which are numerics, fill value 0

In [43]:
```
application_bureau_loan_train_data.isnull().any()
```

Out[43]:
```
SK_ID_CURR                    False
TARGET                        False
NAME_CONTRACT_TYPE            False
CODE_GENDER                   False
FLAG_OWN_CAR                  False
FLAG_OWN_REALTY               False
CNT_CHILDREN                  False
AMT_INCOME_TOTAL              False
AMT_CREDIT                    False
AMT_ANNUITY                    True
AMT_GOODS_PRICE                True
NAME_TYPE_SUITE                True
NAME_INCOME_TYPE              False
NAME_EDUCATION_TYPE           False
NAME_FAMILY_STATUS            False
NAME_HOUSING_TYPE             False
REGION_POPULATION_RELATIVE    False
DAYS_BIRTH                    False
DAYS_EMPLOYED                 False
```

```
DAYS_REGISTRATION                False
DAYS_ID_PUBLISH                  False
OWN_CAR_AGE                       True
FLAG_MOBIL                       False
FLAG_EMP_PHONE                   False
FLAG_WORK_PHONE                  False
FLAG_CONT_MOBILE                 False
FLAG_PHONE                       False
FLAG_EMAIL                       False
OCCUPATION_TYPE                   True
CNT_FAM_MEMBERS                   True
REGION_RATING_CLIENT             False
REGION_RATING_CLIENT_W_CITY      False
WEEKDAY_APPR_PROCESS_START       False
HOUR_APPR_PROCESS_START          False
REG_REGION_NOT_LIVE_REGION       False
REG_REGION_NOT_WORK_REGION       False
LIVE_REGION_NOT_WORK_REGION      False
REG_CITY_NOT_LIVE_CITY           False
REG_CITY_NOT_WORK_CITY           False
LIVE_CITY_NOT_WORK_CITY          False
ORGANIZATION_TYPE                False
EXT_SOURCE_1                      True
EXT_SOURCE_2                      True
EXT_SOURCE_3                      True
APARTMENTS_AVG                    True
BASEMENTAREA_AVG                  True
YEARS_BEGINEXPLUATATION_AVG       True
YEARS_BUILD_AVG                   True
COMMONAREA_AVG                    True
ELEVATORS_AVG                     True
ENTRANCES_AVG                     True
FLOORSMAX_AVG                     True
FLOORSMIN_AVG                     True
LANDAREA_AVG                      True
LIVINGAPARTMENTS_AVG              True
LIVINGAREA_AVG                    True
NONLIVINGAPARTMENTS_AVG           True
NONLIVINGAREA_AVG                 True
APARTMENTS_MODE                   True
BASEMENTAREA_MODE                 True
YEARS_BEGINEXPLUATATION_MODE      True
YEARS_BUILD_MODE                  True
COMMONAREA_MODE                   True
ELEVATORS_MODE                    True
ENTRANCES_MODE                    True
FLOORSMAX_MODE                    True
FLOORSMIN_MODE                    True
```

LANDAREA_MODE	True
LIVINGAPARTMENTS_MODE	True
LIVINGAREA_MODE	True
NONLIVINGAPARTMENTS_MODE	True
NONLIVINGAREA_MODE	True
APARTMENTS_MEDI	True
BASEMENTAREA_MEDI	True
YEARS_BEGINEXPLUATATION_MEDI	True
YEARS_BUILD_MEDI	True
COMMONAREA_MEDI	True
ELEVATORS_MEDI	True
ENTRANCES_MEDI	True
FLOORSMAX_MEDI	True
FLOORSMIN_MEDI	True
LANDAREA_MEDI	True
LIVINGAPARTMENTS_MEDI	True
LIVINGAREA_MEDI	True
NONLIVINGAPARTMENTS_MEDI	True
NONLIVINGAREA_MEDI	True
FONDKAPREMONT_MODE	True
HOUSETYPE_MODE	True
TOTALAREA_MODE	True
WALLSMATERIAL_MODE	True
EMERGENCYSTATE_MODE	True
OBS_30_CNT_SOCIAL_CIRCLE	True
DEF_30_CNT_SOCIAL_CIRCLE	True
OBS_60_CNT_SOCIAL_CIRCLE	True
DEF_60_CNT_SOCIAL_CIRCLE	True
DAYS_LAST_PHONE_CHANGE	True
FLAG_DOCUMENT_2	False
FLAG_DOCUMENT_3	False
FLAG_DOCUMENT_4	False
FLAG_DOCUMENT_5	False
FLAG_DOCUMENT_6	False
FLAG_DOCUMENT_7	False
FLAG_DOCUMENT_8	False
FLAG_DOCUMENT_9	False
FLAG_DOCUMENT_10	False
FLAG_DOCUMENT_11	False
FLAG_DOCUMENT_12	False
FLAG_DOCUMENT_13	False
FLAG_DOCUMENT_14	False
FLAG_DOCUMENT_15	False
FLAG_DOCUMENT_16	False
FLAG_DOCUMENT_17	False
FLAG_DOCUMENT_18	False
FLAG_DOCUMENT_19	False
FLAG_DOCUMENT_20	False

```
FLAG_DOCUMENT_21                    False
AMT_REQ_CREDIT_BUREAU_HOUR           True
AMT_REQ_CREDIT_BUREAU_DAY            True
AMT_REQ_CREDIT_BUREAU_WEEK           True
AMT_REQ_CREDIT_BUREAU_MON            True
AMT_REQ_CREDIT_BUREAU_QRT            True
AMT_REQ_CREDIT_BUREAU_YEAR           True
YEARS_CREDIT_ACTIVE                  True
CREDIT_YEAR_OVERDUE_ACTIVE           True
YEARS_CREDIT_ENDDATE_ACTIVE          True
YEARS_ENDDATE_FACT_ACTIVE            True
AMT_CREDIT_MAX_OVERDUE_ACTIVE        True
CNT_CREDIT_PROLONG_ACTIVE            True
AMT_CREDIT_SUM_ACTIVE                True
AMT_CREDIT_SUM_DEBT_ACTIVE           True
AMT_CREDIT_SUM_LIMIT_ACTIVE          True
AMT_CREDIT_SUM_OVERDUE_ACTIVE        True
AMT_ANNUITY_ACTIVE                   True
YEARS_CREDIT_CLOSED                  True
CREDIT_YEAR_OVERDUE_CLOSED           True
YEARS_CREDIT_ENDDATE_CLOSED          True
YEARS_ENDDATE_FACT_CLOSED            True
AMT_CREDIT_MAX_OVERDUE_CLOSED        True
CNT_CREDIT_PROLONG_CLOSED            True
AMT_CREDIT_SUM_CLOSED                True
AMT_CREDIT_SUM_DEBT_CLOSED           True
AMT_CREDIT_SUM_LIMIT_CLOSED          True
AMT_CREDIT_SUM_OVERDUE_CLOSED        True
AMT_ANNUITY_CLOSED                   True
YEARS_CREDIT_SOLD                    True
CREDIT_YEAR_OVERDUE_SOLD             True
YEARS_CREDIT_ENDDATE_SOLD            True
YEARS_ENDDATE_FACT_SOLD              True
AMT_CREDIT_MAX_OVERDUE_SOLD          True
CNT_CREDIT_PROLONG_SOLD              True
AMT_CREDIT_SUM_SOLD                  True
AMT_CREDIT_SUM_DEBT_SOLD             True
AMT_CREDIT_SUM_LIMIT_SOLD            True
AMT_CREDIT_SUM_OVERDUE_SOLD          True
AMT_ANNUITY_SOLD                     True
YEARS_CREDIT_BAD_DEBT                True
CREDIT_YEAR_OVERDUE_BAD_DEBT         True
YEARS_CREDIT_ENDDATE_BAD_DEBT        True
YEARS_ENDDATE_FACT_BAD_DEBT          True
AMT_CREDIT_MAX_OVERDUE_BAD_DEBT      True
CNT_CREDIT_PROLONG_BAD_DEBT          True
AMT_CREDIT_SUM_BAD_DEBT              True
AMT_CREDIT_SUM_DEBT_BAD_DEBT         True
```

```
AMT_CREDIT_SUM_LIMIT_BAD_DEBT       True
AMT_CREDIT_SUM_OVERDUE_BAD_DEBT     True
AMT_ANNUITY_BAD_DEBT                True
Active                              True
Bad_debt                            True
Closed                              True
Sold                                True
PREV_CASH_AMT_ANNUITY               True
PREV_CASH_AMT_APPLICATION           True
PREV_CASH_AMT_CREDIT                True
PREV_CASH_AMT_DOWN_PAYMENT          True
PREV_CASH_AMT_GOODS_PRICE           True
PREV_CONSUMER_AMT_ANNUITY           True
PREV_CONSUMER_AMT_APPLICATION       True
PREV_CONSUMER_AMT_CREDIT            True
PREV_CONSUMER_AMT_DOWN_PAYMENT      True
PREV_CONSUMER_AMT_GOODS_PRICE       True
PREV_REVOVING_AMT_ANNUITY           True
PREV_REVOLVING_AMT_APPLICATION      True
PREV_REVOLVING_AMT_CREDIT           True
PREV_REVOVING_AMT_DOWN_PAYMENT      True
PREV_REVOVING_AMT_GOODS_PRICE       True
PREV_XNA_AMT_ANNUITY                True
PREV_XNA_AMT_APPLICATION            True
PREV_XNA_AMT_CREDIT                 True
PREV_XNA_AMT_DOWN_PAYMENT           True
PREV_XNA_AMT_GOODS_PRICE            True
CASH_LOANS                          True
CONSUMER_LOANS                      True
REVOLVING_LOANS                     True
XNA                                 True
dtype: bool
```

In [44]:
```
missing_info = list(application_bureau_loan_train_data.columns[application_bureau_loa
missing_info
```

Out[44]:
```
['AMT_ANNUITY',
 'AMT_GOODS_PRICE',
 'NAME_TYPE_SUITE',
 'OWN_CAR_AGE',
 'OCCUPATION_TYPE',
 'CNT_FAM_MEMBERS',
 'EXT_SOURCE_1',
 'EXT_SOURCE_2',
 'EXT_SOURCE_3',
 'APARTMENTS_AVG',
 'BASEMENTAREA_AVG',
 'YEARS_BEGINEXPLUATATION_AVG',
```

```
'YEARS_BUILD_AVG',
'COMMONAREA_AVG',
'ELEVATORS_AVG',
'ENTRANCES_AVG',
'FLOORSMAX_AVG',
'FLOORSMIN_AVG',
'LANDAREA_AVG',
'LIVINGAPARTMENTS_AVG',
'LIVINGAREA_AVG',
'NONLIVINGAPARTMENTS_AVG',
'NONLIVINGAREA_AVG',
'APARTMENTS_MODE',
'BASEMENTAREA_MODE',
'YEARS_BEGINEXPLUATATION_MODE',
'YEARS_BUILD_MODE',
'COMMONAREA_MODE',
'ELEVATORS_MODE',
'ENTRANCES_MODE',
'FLOORSMAX_MODE',
'FLOORSMIN_MODE',
'LANDAREA_MODE',
'LIVINGAPARTMENTS_MODE',
'LIVINGAREA_MODE',
'NONLIVINGAPARTMENTS_MODE',
'NONLIVINGAREA_MODE',
'APARTMENTS_MEDI',
'BASEMENTAREA_MEDI',
'YEARS_BEGINEXPLUATATION_MEDI',
'YEARS_BUILD_MEDI',
'COMMONAREA_MEDI',
'ELEVATORS_MEDI',
'ENTRANCES_MEDI',
'FLOORSMAX_MEDI',
'FLOORSMIN_MEDI',
'LANDAREA_MEDI',
'LIVINGAPARTMENTS_MEDI',
'LIVINGAREA_MEDI',
'NONLIVINGAPARTMENTS_MEDI',
'NONLIVINGAREA_MEDI',
'FONDKAPREMONT_MODE',
'HOUSETYPE_MODE',
'TOTALAREA_MODE',
'WALLSMATERIAL_MODE',
'EMERGENCYSTATE_MODE',
'OBS_30_CNT_SOCIAL_CIRCLE',
'DEF_30_CNT_SOCIAL_CIRCLE',
'OBS_60_CNT_SOCIAL_CIRCLE',
'DEF_60_CNT_SOCIAL_CIRCLE',
```

```
'DAYS_LAST_PHONE_CHANGE',
'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',
'YEARS_CREDIT_ACTIVE',
'CREDIT_YEAR_OVERDUE_ACTIVE',
'YEARS_CREDIT_ENDDATE_ACTIVE',
'YEARS_ENDDATE_FACT_ACTIVE',
'AMT_CREDIT_MAX_OVERDUE_ACTIVE',
'CNT_CREDIT_PROLONG_ACTIVE',
'AMT_CREDIT_SUM_ACTIVE',
'AMT_CREDIT_SUM_DEBT_ACTIVE',
'AMT_CREDIT_SUM_LIMIT_ACTIVE',
'AMT_CREDIT_SUM_OVERDUE_ACTIVE',
'AMT_ANNUITY_ACTIVE',
'YEARS_CREDIT_CLOSED',
'CREDIT_YEAR_OVERDUE_CLOSED',
'YEARS_CREDIT_ENDDATE_CLOSED',
'YEARS_ENDDATE_FACT_CLOSED',
'AMT_CREDIT_MAX_OVERDUE_CLOSED',
'CNT_CREDIT_PROLONG_CLOSED',
'AMT_CREDIT_SUM_CLOSED',
'AMT_CREDIT_SUM_DEBT_CLOSED',
'AMT_CREDIT_SUM_LIMIT_CLOSED',
'AMT_CREDIT_SUM_OVERDUE_CLOSED',
'AMT_ANNUITY_CLOSED',
'YEARS_CREDIT_SOLD',
'CREDIT_YEAR_OVERDUE_SOLD',
'YEARS_CREDIT_ENDDATE_SOLD',
'YEARS_ENDDATE_FACT_SOLD',
'AMT_CREDIT_MAX_OVERDUE_SOLD',
'CNT_CREDIT_PROLONG_SOLD',
'AMT_CREDIT_SUM_SOLD',
'AMT_CREDIT_SUM_DEBT_SOLD',
'AMT_CREDIT_SUM_LIMIT_SOLD',
'AMT_CREDIT_SUM_OVERDUE_SOLD',
'AMT_ANNUITY_SOLD',
'YEARS_CREDIT_BAD_DEBT',
'CREDIT_YEAR_OVERDUE_BAD_DEBT',
'YEARS_CREDIT_ENDDATE_BAD_DEBT',
'YEARS_ENDDATE_FACT_BAD_DEBT',
'AMT_CREDIT_MAX_OVERDUE_BAD_DEBT',
'CNT_CREDIT_PROLONG_BAD_DEBT',
'AMT_CREDIT_SUM_BAD_DEBT',
'AMT_CREDIT_SUM_DEBT_BAD_DEBT',
```

```
 'AMT_CREDIT_SUM_LIMIT_BAD_DEBT',
 'AMT_CREDIT_SUM_OVERDUE_BAD_DEBT',
 'AMT_ANNUITY_BAD_DEBT',
 'Active',
 'Bad_debt',
 'Closed',
 'Sold',
 'PREV_CASH_AMT_ANNUITY',
 'PREV_CASH_AMT_APPLICATION',
 'PREV_CASH_AMT_CREDIT',
 'PREV_CASH_AMT_DOWN_PAYMENT',
 'PREV_CASH_AMT_GOODS_PRICE',
 'PREV_CONSUMER_AMT_ANNUITY',
 'PREV_CONSUMER_AMT_APPLICATION',
 'PREV_CONSUMER_AMT_CREDIT',
 'PREV_CONSUMER_AMT_DOWN_PAYMENT',
 'PREV_CONSUMER_AMT_GOODS_PRICE',
 'PREV_REVOVING_AMT_ANNUITY',
 'PREV_REVOLVING_AMT_APPLICATION',
 'PREV_REVOLVING_AMT_CREDIT',
 'PREV_REVOVING_AMT_DOWN_PAYMENT',
 'PREV_REVOVING_AMT_GOODS_PRICE',
 'PREV_XNA_AMT_ANNUITY',
 'PREV_XNA_AMT_APPLICATION',
 'PREV_XNA_AMT_CREDIT',
 'PREV_XNA_AMT_DOWN_PAYMENT',
 'PREV_XNA_AMT_GOODS_PRICE',
 'CASH_LOANS',
 'CONSUMER_LOANS',
 'REVOLVING_LOANS',
 'XNA']
```

In [45]:
```
# Find and Display percentage of missing values in each column
for col in missing_info:
    percent_missing = application_bureau_loan_train_data[application_bureau_loan_trai
    print('percent missing for column {}: {:.2f}%'.format(col, round(percent_missing,
```

```
percent missing for column AMT_ANNUITY: 0.00%
percent missing for column AMT_GOODS_PRICE: 0.09%
percent missing for column NAME_TYPE_SUITE: 0.42%
percent missing for column OWN_CAR_AGE: 65.99%
percent missing for column OCCUPATION_TYPE: 31.35%
percent missing for column CNT_FAM_MEMBERS: 0.00%
percent missing for column EXT_SOURCE_1: 56.38%
percent missing for column EXT_SOURCE_2: 0.21%
percent missing for column EXT_SOURCE_3: 19.83%
percent missing for column APARTMENTS_AVG: 50.75%
percent missing for column BASEMENTAREA_AVG: 58.52%
```

```
percent missing for column YEARS_BEGINEXPLUATATION_AVG: 48.78%
percent missing for column YEARS_BUILD_AVG: 66.50%
percent missing for column COMMONAREA_AVG: 69.87%
percent missing for column ELEVATORS_AVG: 53.30%
percent missing for column ENTRANCES_AVG: 50.35%
percent missing for column FLOORSMAX_AVG: 49.76%
percent missing for column FLOORSMIN_AVG: 67.85%
percent missing for column LANDAREA_AVG: 59.38%
percent missing for column LIVINGAPARTMENTS_AVG: 68.35%
percent missing for column LIVINGAREA_AVG: 50.19%
percent missing for column NONLIVINGAPARTMENTS_AVG: 69.43%
percent missing for column NONLIVINGAREA_AVG: 55.18%
percent missing for column APARTMENTS_MODE: 50.75%
percent missing for column BASEMENTAREA_MODE: 58.52%
percent missing for column YEARS_BEGINEXPLUATATION_MODE: 48.78%
percent missing for column YEARS_BUILD_MODE: 66.50%
percent missing for column COMMONAREA_MODE: 69.87%
percent missing for column ELEVATORS_MODE: 53.30%
percent missing for column ENTRANCES_MODE: 50.35%
percent missing for column FLOORSMAX_MODE: 49.76%
percent missing for column FLOORSMIN_MODE: 67.85%
percent missing for column LANDAREA_MODE: 59.38%
percent missing for column LIVINGAPARTMENTS_MODE: 68.35%
percent missing for column LIVINGAREA_MODE: 50.19%
percent missing for column NONLIVINGAPARTMENTS_MODE: 69.43%
percent missing for column NONLIVINGAREA_MODE: 55.18%
percent missing for column APARTMENTS_MEDI: 50.75%
percent missing for column BASEMENTAREA_MEDI: 58.52%
percent missing for column YEARS_BEGINEXPLUATATION_MEDI: 48.78%
percent missing for column YEARS_BUILD_MEDI: 66.50%
percent missing for column COMMONAREA_MEDI: 69.87%
percent missing for column ELEVATORS_MEDI: 53.30%
percent missing for column ENTRANCES_MEDI: 50.35%
percent missing for column FLOORSMAX_MEDI: 49.76%
percent missing for column FLOORSMIN_MEDI: 67.85%
percent missing for column LANDAREA_MEDI: 59.38%
percent missing for column LIVINGAPARTMENTS_MEDI: 68.35%
percent missing for column LIVINGAREA_MEDI: 50.19%
percent missing for column NONLIVINGAPARTMENTS_MEDI: 69.43%
percent missing for column NONLIVINGAREA_MEDI: 55.18%
percent missing for column FONDKAPREMONT_MODE: 68.39%
percent missing for column HOUSETYPE_MODE: 50.18%
percent missing for column TOTALAREA_MODE: 48.27%
percent missing for column WALLSMATERIAL_MODE: 50.84%
percent missing for column EMERGENCYSTATE_MODE: 47.40%
percent missing for column OBS_30_CNT_SOCIAL_CIRCLE: 0.33%
percent missing for column DEF_30_CNT_SOCIAL_CIRCLE: 0.33%
percent missing for column OBS_60_CNT_SOCIAL_CIRCLE: 0.33%
```

```
percent missing for column DEF_60_CNT_SOCIAL_CIRCLE: 0.33%
percent missing for column DAYS_LAST_PHONE_CHANGE: 0.00%
percent missing for column AMT_REQ_CREDIT_BUREAU_HOUR: 13.50%
percent missing for column AMT_REQ_CREDIT_BUREAU_DAY: 13.50%
percent missing for column AMT_REQ_CREDIT_BUREAU_WEEK: 13.50%
percent missing for column AMT_REQ_CREDIT_BUREAU_MON: 13.50%
percent missing for column AMT_REQ_CREDIT_BUREAU_QRT: 13.50%
percent missing for column AMT_REQ_CREDIT_BUREAU_YEAR: 13.50%
percent missing for column YEARS_CREDIT_ACTIVE: 29.38%
percent missing for column CREDIT_YEAR_OVERDUE_ACTIVE: 29.38%
percent missing for column YEARS_CREDIT_ENDDATE_ACTIVE: 29.38%
percent missing for column YEARS_ENDDATE_FACT_ACTIVE: 29.38%
percent missing for column AMT_CREDIT_MAX_OVERDUE_ACTIVE: 29.38%
percent missing for column CNT_CREDIT_PROLONG_ACTIVE: 29.38%
percent missing for column AMT_CREDIT_SUM_ACTIVE: 29.38%
percent missing for column AMT_CREDIT_SUM_DEBT_ACTIVE: 29.38%
percent missing for column AMT_CREDIT_SUM_LIMIT_ACTIVE: 29.38%
percent missing for column AMT_CREDIT_SUM_OVERDUE_ACTIVE: 29.38%
percent missing for column AMT_ANNUITY_ACTIVE: 29.38%
percent missing for column YEARS_CREDIT_CLOSED: 25.15%
percent missing for column CREDIT_YEAR_OVERDUE_CLOSED: 25.15%
percent missing for column YEARS_CREDIT_ENDDATE_CLOSED: 25.15%
percent missing for column YEARS_ENDDATE_FACT_CLOSED: 25.15%
percent missing for column AMT_CREDIT_MAX_OVERDUE_CLOSED: 25.15%
percent missing for column CNT_CREDIT_PROLONG_CLOSED: 25.15%
percent missing for column AMT_CREDIT_SUM_CLOSED: 25.15%
percent missing for column AMT_CREDIT_SUM_DEBT_CLOSED: 25.15%
percent missing for column AMT_CREDIT_SUM_LIMIT_CLOSED: 25.15%
percent missing for column AMT_CREDIT_SUM_OVERDUE_CLOSED: 25.15%
percent missing for column AMT_ANNUITY_CLOSED: 25.15%
percent missing for column YEARS_CREDIT_SOLD: 98.30%
percent missing for column CREDIT_YEAR_OVERDUE_SOLD: 98.30%
percent missing for column YEARS_CREDIT_ENDDATE_SOLD: 98.30%
percent missing for column YEARS_ENDDATE_FACT_SOLD: 98.30%
percent missing for column AMT_CREDIT_MAX_OVERDUE_SOLD: 98.30%
percent missing for column CNT_CREDIT_PROLONG_SOLD: 98.30%
percent missing for column AMT_CREDIT_SUM_SOLD: 98.30%
percent missing for column AMT_CREDIT_SUM_DEBT_SOLD: 98.30%
percent missing for column AMT_CREDIT_SUM_LIMIT_SOLD: 98.30%
percent missing for column AMT_CREDIT_SUM_OVERDUE_SOLD: 98.30%
percent missing for column AMT_ANNUITY_SOLD: 98.30%
percent missing for column YEARS_CREDIT_BAD_DEBT: 99.99%
percent missing for column CREDIT_YEAR_OVERDUE_BAD_DEBT: 99.99%
percent missing for column YEARS_CREDIT_ENDDATE_BAD_DEBT: 99.99%
percent missing for column YEARS_ENDDATE_FACT_BAD_DEBT: 99.99%
percent missing for column AMT_CREDIT_MAX_OVERDUE_BAD_DEBT: 99.99%
percent missing for column CNT_CREDIT_PROLONG_BAD_DEBT: 99.99%
percent missing for column AMT_CREDIT_SUM_BAD_DEBT: 99.99%
```

```
percent missing for column AMT_CREDIT_SUM_DEBT_BAD_DEBT: 99.99%
percent missing for column AMT_CREDIT_SUM_LIMIT_BAD_DEBT: 99.99%
percent missing for column AMT_CREDIT_SUM_OVERDUE_BAD_DEBT: 99.99%
percent missing for column AMT_ANNUITY_BAD_DEBT: 99.99%
percent missing for column Active: 14.31%
percent missing for column Bad_debt: 14.31%
percent missing for column Closed: 14.31%
percent missing for column Sold: 14.31%
percent missing for column PREV_CASH_AMT_ANNUITY: 44.27%
percent missing for column PREV_CASH_AMT_APPLICATION: 44.27%
percent missing for column PREV_CASH_AMT_CREDIT: 44.27%
percent missing for column PREV_CASH_AMT_DOWN_PAYMENT: 44.27%
percent missing for column PREV_CASH_AMT_GOODS_PRICE: 44.27%
percent missing for column PREV_CONSUMER_AMT_ANNUITY: 12.52%
percent missing for column PREV_CONSUMER_AMT_APPLICATION: 12.52%
percent missing for column PREV_CONSUMER_AMT_CREDIT: 12.52%
percent missing for column PREV_CONSUMER_AMT_DOWN_PAYMENT: 12.52%
percent missing for column PREV_CONSUMER_AMT_GOODS_PRICE: 12.52%
percent missing for column PREV_REVOVING_AMT_ANNUITY: 66.16%
percent missing for column PREV_REVOLVING_AMT_APPLICATION: 66.16%
percent missing for column PREV_REVOLVING_AMT_CREDIT: 66.16%
percent missing for column PREV_REVOVING_AMT_DOWN_PAYMENT: 66.16%
percent missing for column PREV_REVOVING_AMT_GOODS_PRICE: 66.16%
percent missing for column PREV_XNA_AMT_ANNUITY: 99.91%
percent missing for column PREV_XNA_AMT_APPLICATION: 99.91%
percent missing for column PREV_XNA_AMT_CREDIT: 99.91%
percent missing for column PREV_XNA_AMT_DOWN_PAYMENT: 99.91%
percent missing for column PREV_XNA_AMT_GOODS_PRICE: 99.91%
percent missing for column CASH_LOANS: 5.35%
percent missing for column CONSUMER_LOANS: 5.35%
percent missing for column REVOLVING_LOANS: 5.35%
percent missing for column XNA: 5.35%
```

In [46]:
```
# For application_bureau_loan_train_data populate all the missing string fields to NA
# Populate all the missing numerical fields to 0
str_cols = application_bureau_loan_train_data.columns[application_bureau_loan_train_da
application_bureau_loan_train_data[str_cols] = application_bureau_loan_train_data[str_
application_bureau_loan_train_data.fillna(0,inplace=True)
```

In [47]:
```
# Confirm if there is no more missing information present
missing_info = list(application_bureau_loan_train_data.columns[application_bureau_loan
missing_info
```

Out[47]: []

In [48]:
```
# For application_bureau_loan_teest_data Populate all the missing object to NA
# Populate all teh missing numerical fields to 0
str_cols = application_bureau_loan_test_data.columns[application_bureau_loan_test_data
```

```
application_bureau_loan_test_data[str_cols] = application_bureau_loan_test_data[str_co
application_bureau_loan_test_data.fillna(0,inplace=True)
```

1.4 Analyze the Data

1.4.1 First try to understand the data by looking a few records

```
In [46]: application_bureau_loan_train_data.head()
```

```
Out[46]:    SK_ID_CURR  TARGET NAME_CONTRACT_TYPE CODE_GENDER FLAG_OWN_CAR FLAG_OWN_REALTY  CN
         0      100002       1         Cash loans           M            N               Y
         1      100003       0         Cash loans           F            N               N
         2      100004       0    Revolving loans           M            Y               Y
         3      100006       0         Cash loans           F            N               Y
         4      100007       0         Cash loans           M            N               Y

            LANDAREA_MEDI  LIVINGAPARTMENTS_MEDI  LIVINGAREA_MEDI  NONLIVINGAPARTMENTS_MEDI  N
         0         0.0375                 0.0205           0.0193                    0.0000
         1         0.0132                 0.0787           0.0558                    0.0039
         2         0.0000                 0.0000           0.0000                    0.0000
         3         0.0000                 0.0000           0.0000                    0.0000
         4         0.0000                 0.0000           0.0000                    0.0000

            YEARS_ENDDATE_FACT_SOLD  AMT_CREDIT_MAX_OVERDUE_SOLD  CNT_CREDIT_PROLONG_SOLD  AMT
         0                      0.0                          0.0                      0.0
         1                      0.0                          0.0                      0.0
         2                      0.0                          0.0                      0.0
         3                      0.0                          0.0                      0.0
         4                      0.0                          0.0                      0.0
```

1.4.2 Get the event rate

Event rate percentage is calculated by dividing number of 1 in TARGET field by total number of records multiplied by 100

```
In [228]: event_rate = (sum(application_bureau_loan_train_data.loc[application_bureau_loan_tra
          print("Event_Rate: " + str(event_rate) + "%")
```

```
Event_Rate: 8.072881945686495%
```

1.4.3 Interpretation:

From the event rate, I can conclude that the class is highly imbalanced

1.4.4 Analyze NAME_CONTRACT_TYPE vs TARGET

- Create count of each type of NAME_CONTRACT_TYPE
- Create a cross-tabulation bar plot between NAME_CONTRACT_TYPE vs TARGET

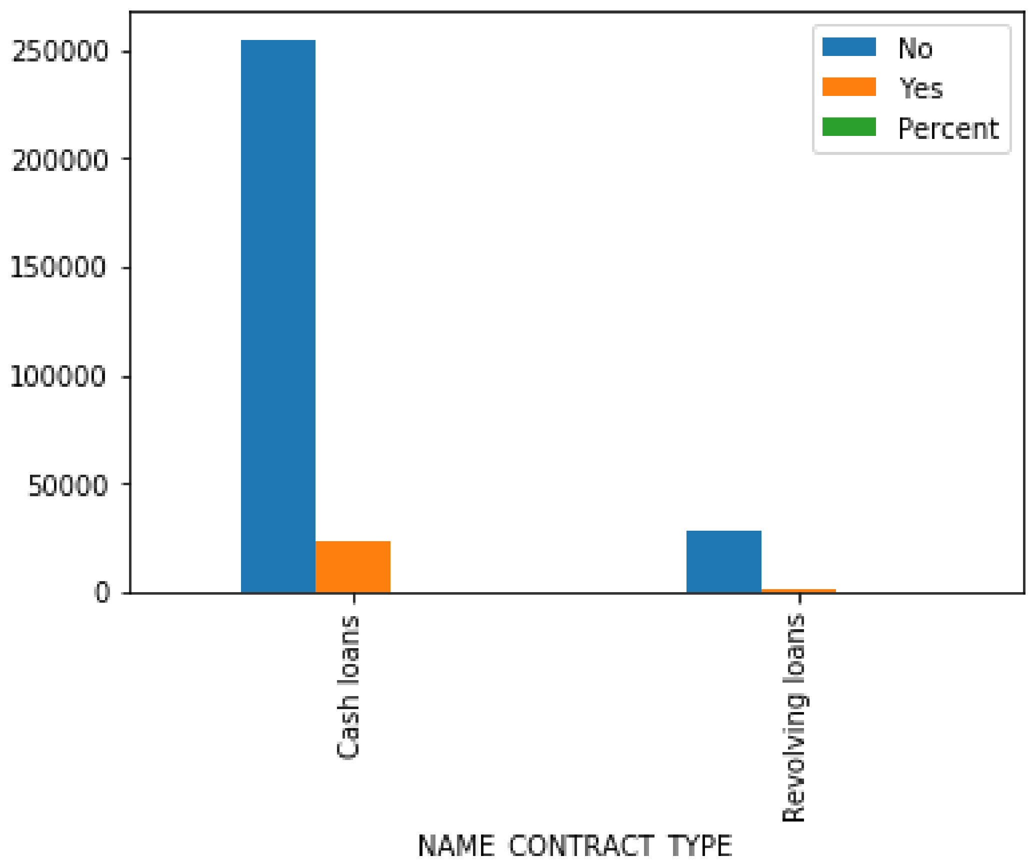
In [48]: application_bureau_loan_train_data['NAME_CONTRACT_TYPE'].value_counts()

```
Out[48]: Cash loans         278232
         Revolving loans     29279
         Name: NAME_CONTRACT_TYPE, dtype: int64
```

```
In [49]: tab = pd.crosstab(index=application_train_data['NAME_CONTRACT_TYPE'],columns=applicat
         tab.columns = ['No','Yes']
         tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
         print(tab)
         tab.plot(kind='bar')
```

```
                        No    Yes   Percent
NAME_CONTRACT_TYPE
Cash loans          255011  23221  8.345913
Revolving loans      27675   1604  5.478329
```

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x1451826d4a8>



1.5 Interpreation:

Higher percentage of cash loans were approved compared to Revolving loans

1.5.1 Analyze CODE_GENDER vs TARGET

- Create count of each type of CODE_GENDER
- Create a cross-tabulation graph between CODE_GENDER vs TARGET

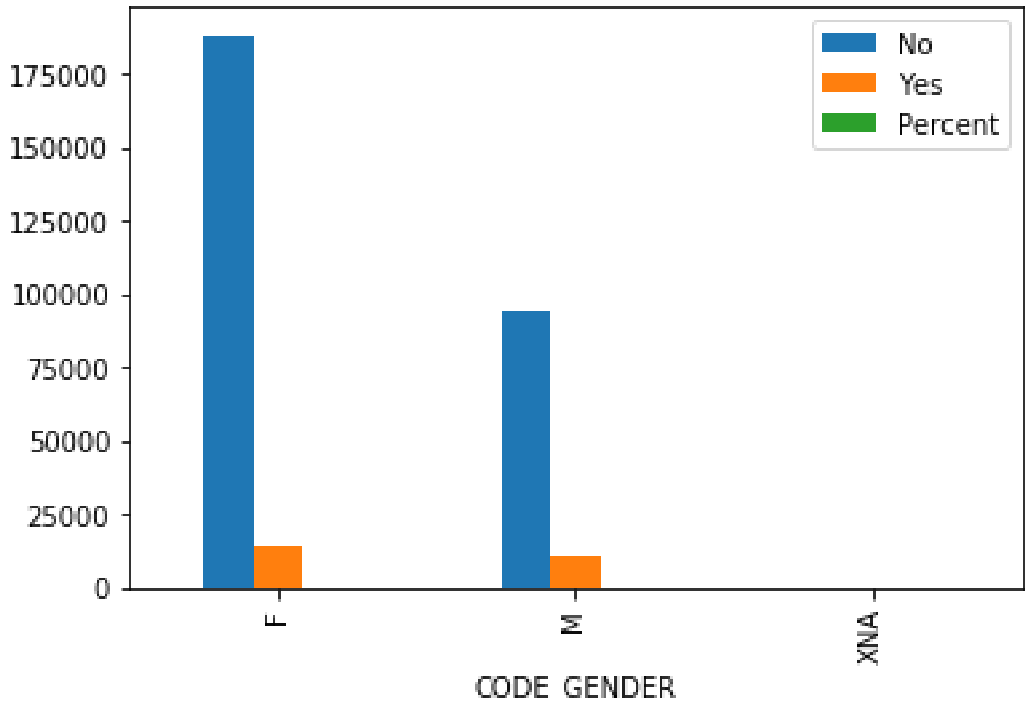
```
In [231]: application_bureau_loan_train_data['CODE_GENDER'].value_counts()
```

```
Out[231]: F      202448
          M      105059
          XNA         4
          Name: CODE_GENDER, dtype: int64
```

```
In [232]: tab = pd.crosstab(index=application_train_data['CODE_GENDER'],columns=application_tra
          tab.columns = ['No','Yes']
          tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
          print(tab)
          tab.plot(kind='bar')
```

```
                 No    Yes    Percent
CODE_GENDER
F            188278  14170   6.999328
M             94404  10655  10.141920
XNA               4      0   0.000000
```

```
Out[232]: <matplotlib.axes._subplots.AxesSubplot at 0x14740518160>
```



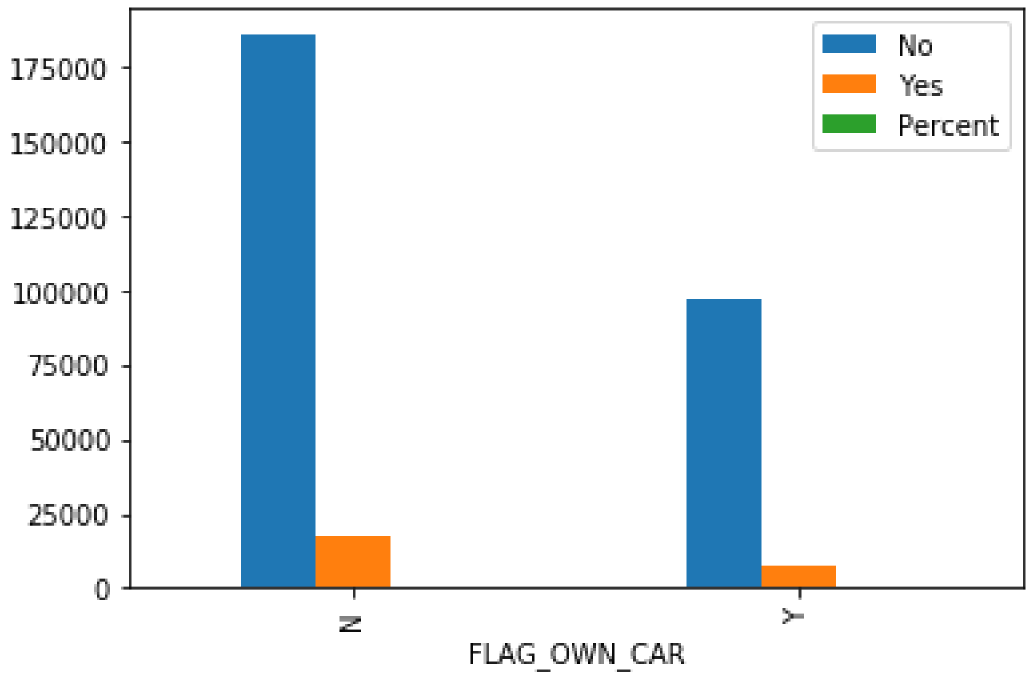
1.5.2 Interpretation:

Almost simlar number of loans have been given to Female and Male although percentage of males given loans is little higher than percentage of females So , I will drop CODE_GENDER as it does not seem to be imporatant field, another reason is I do not want algorithm to have any gender bias to be legally compliant

1.5.3 Analyze FLAG_OWN_CAR vs TARGET

- Create count of each type of FLAG_OWN_CAR
- Create a cross-tabulation graph between FLAG_OWN_CAR vs TARGET

```
In [234]: application_bureau_loan_train_data['FLAG_OWN_CAR'].value_counts()

Out[234]: N    202924
          Y    104587
          Name: FLAG_OWN_CAR, dtype: int64

In [235]: tab = pd.crosstab(index=application_train_data['FLAG_OWN_CAR'],columns=application_t
          tab.columns = ['No','Yes']
          tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
          print(tab)
          tab.plot(kind='bar')

                  No    Yes   Percent
FLAG_OWN_CAR
N             185675  17249  8.500227
Y              97011   7576  7.243730


Out[235]: &lt;matplotlib.axes._subplots.AxesSubplot at 0x147405490b8&gt;
```



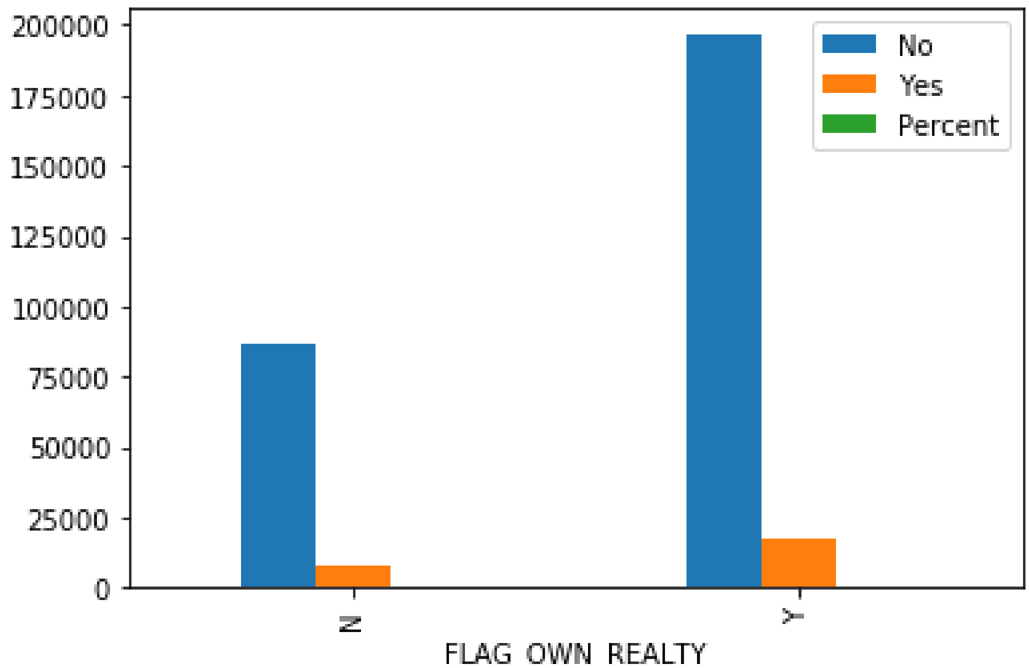

```
In [54]: application_bureau_loan_train_data['FLAG_OWN_REALTY'].value_counts()
```

```
Out[54]: Y    213312
         N     94199
         Name: FLAG_OWN_REALTY, dtype: int64
```

```
In [55]: tab = pd.crosstab(index=application_train_data['FLAG_OWN_REALTY'],columns=application_
         tab.columns = ['No','Yes']
         tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
         print(tab)
         tab.plot(kind='bar')
```

```
                     No    Yes   Percent
FLAG_OWN_REALTY
N                 86357   7842  8.324929
Y                196329  16983  7.961577
```

```
Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x145184ce6a0>
```



1.5.4 Interpreation:

FLAG_OWN_CAR having 'Y' are givne more number of loans than 'N', although percenate of 'Y' given loan are similar to percentage of 'N' given loan

1.5.5 Analyze CNT_CHILDREN vs TARGET

- Get count of each type of CODE_GENDER
- Create a cross-tabulation graph between CODE_GENDER vs TARGET

```
In [236]: application_bureau_loan_train_data['CNT_CHILDREN'].value_counts()
```

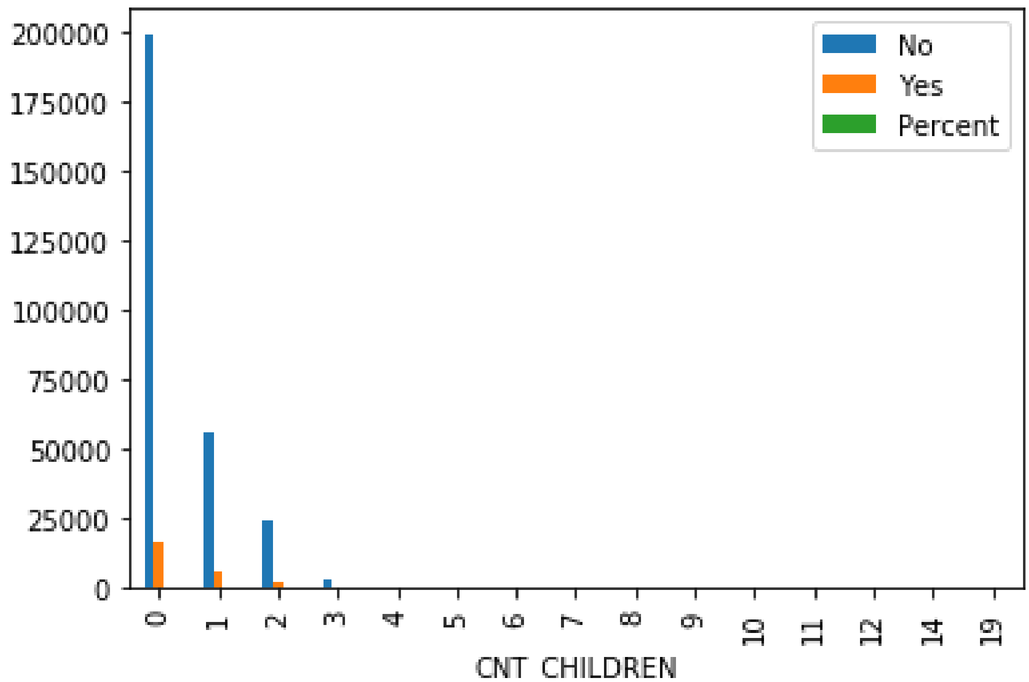
```
Out[236]: 0      215371
          1       61119
          2       26749
          3        3717
          4         429
          5          84
          6          21
          7           7
          14          3
          19          2
          12          2
          10          2
          9           2
```

```
8          2
11         1
Name: CNT_CHILDREN, dtype: int64
```

In [237]:
```
tab = pd.crosstab(index=application_bureau_loan_train_data['CNT_CHILDREN'],columns=ap
tab.columns = ['No','Yes']
tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
print(tab)
tab.plot(kind='bar')
```

```
                  No    Yes     Percent
CNT_CHILDREN
0             198762  16609    7.711809
1              55665   5454    8.923575
2              24416   2333    8.721821
3               3359    358    9.631423
4                374     55   12.820513
5                 77      7    8.333333
6                 15      6   28.571429
7                  7      0    0.000000
8                  2      0    0.000000
9                  0      2  100.000000
10                 2      0    0.000000
11                 0      1  100.000000
12                 2      0    0.000000
14                 3      0    0.000000
19                 2      0    0.000000
```

Out[237]: `<matplotlib.axes._subplots.AxesSubplot at 0x147402df780>`



1.5.6 Interpreation:

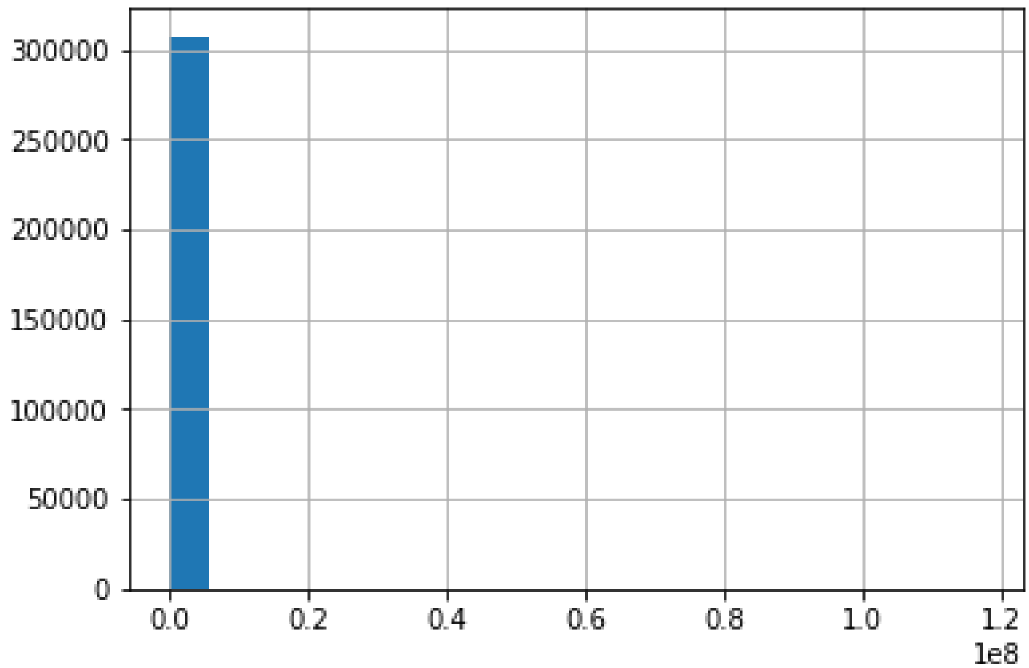
CNT_CHILDREN 0, 1 and 2 have been given more of loans in that order.

1.5.7 Histogram analysis of AMT_INCOME_TOTAL, AMT_CREDIT, AMT_ANNUITY, AMT_GOODS_PRICE

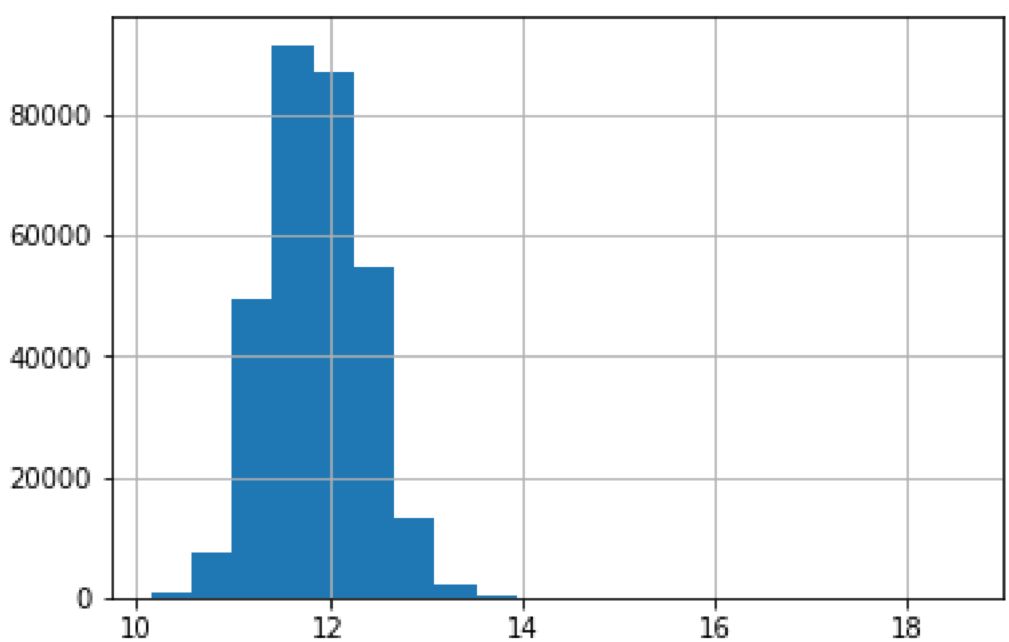
Draw histograms using fields AMT_INCOME_TOTAL, AMT_CREDIT, AMT_ANNUITTY, AMT_GOODS_PRICE

If histograms are skewed or not normal, try log tranformation and check if it becomes normal

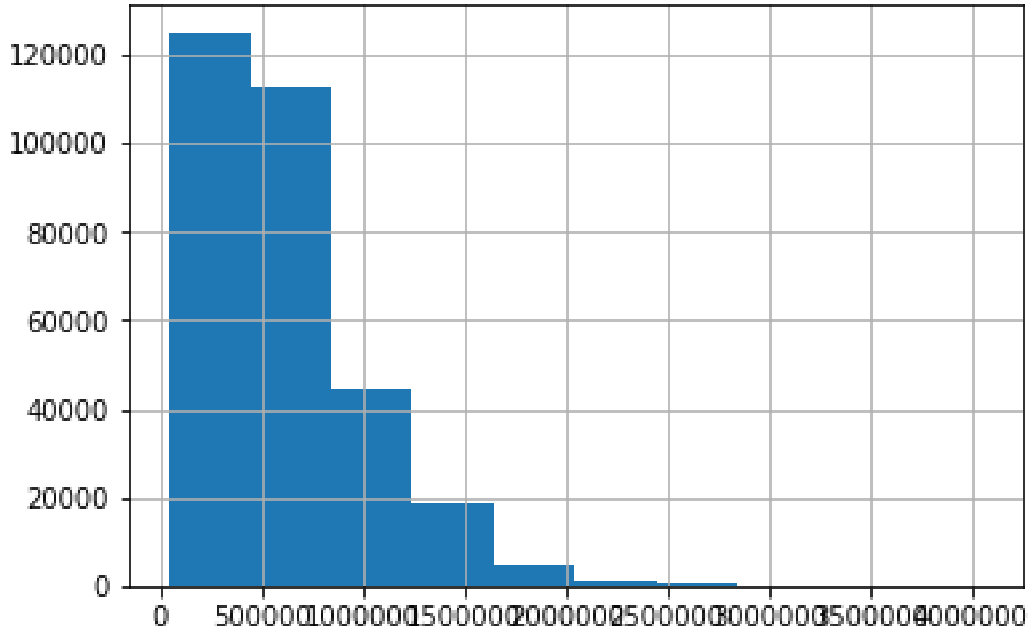
In [238]:
```
application_bureau_loan_train_data['AMT_INCOME_TOTAL'].hist(bins=20)
plt.show()
```



In [239]:
```
np.log(application_bureau_loan_train_data['AMT_INCOME_TOTAL'] + 1).hist(bins=20)
plt.show()
```



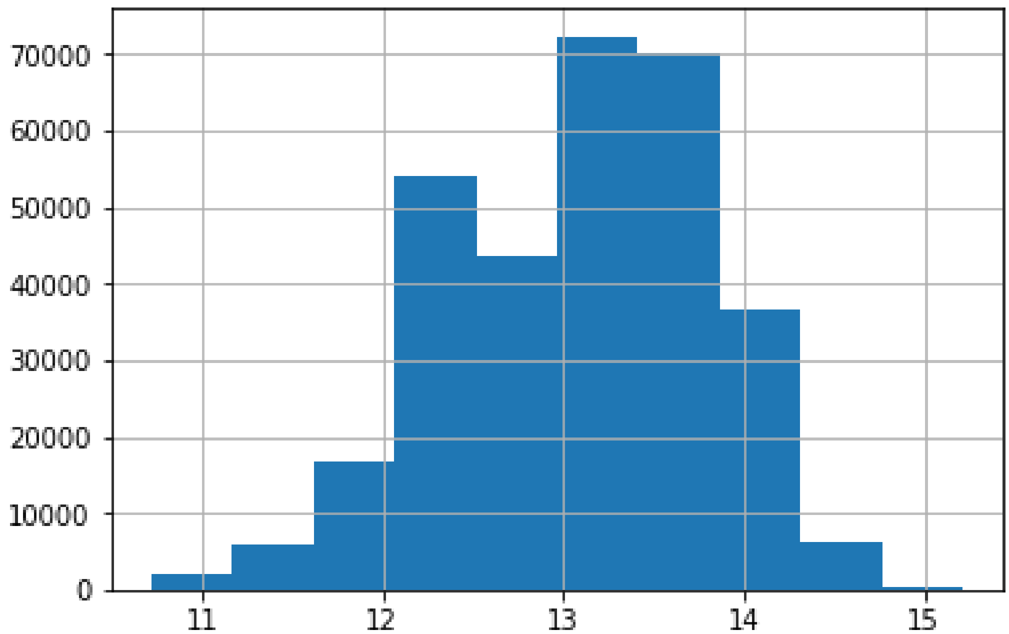
```
In [240]: application_bureau_loan_train_data['AMT_CREDIT'].hist()
          plt.show()
```



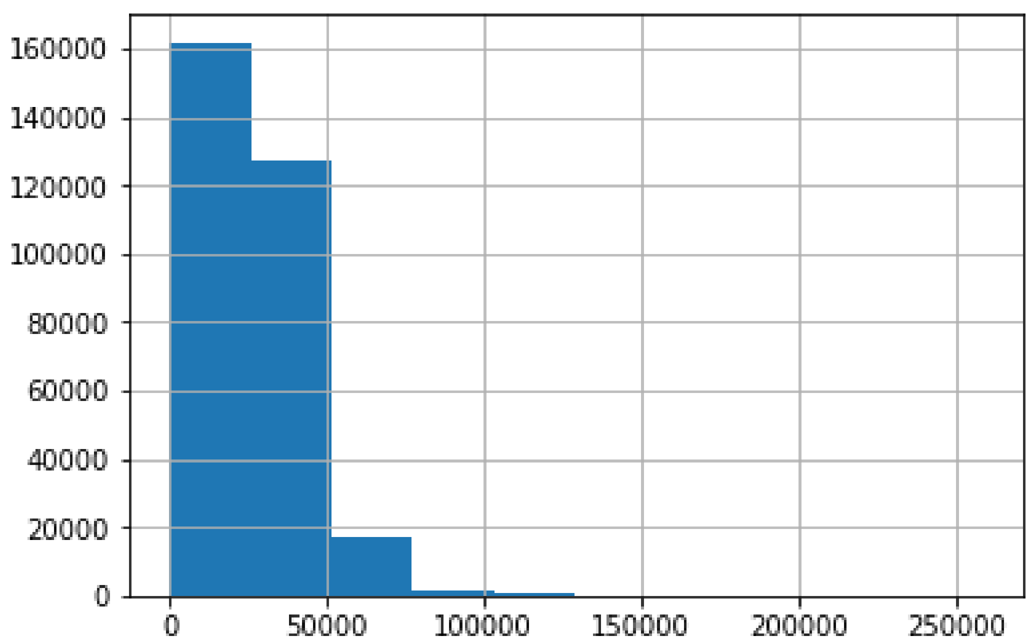
1.6 Interpretation:

AMT_CREDIT is left skewed, we can do log transformation t ocheck if it becomes normal

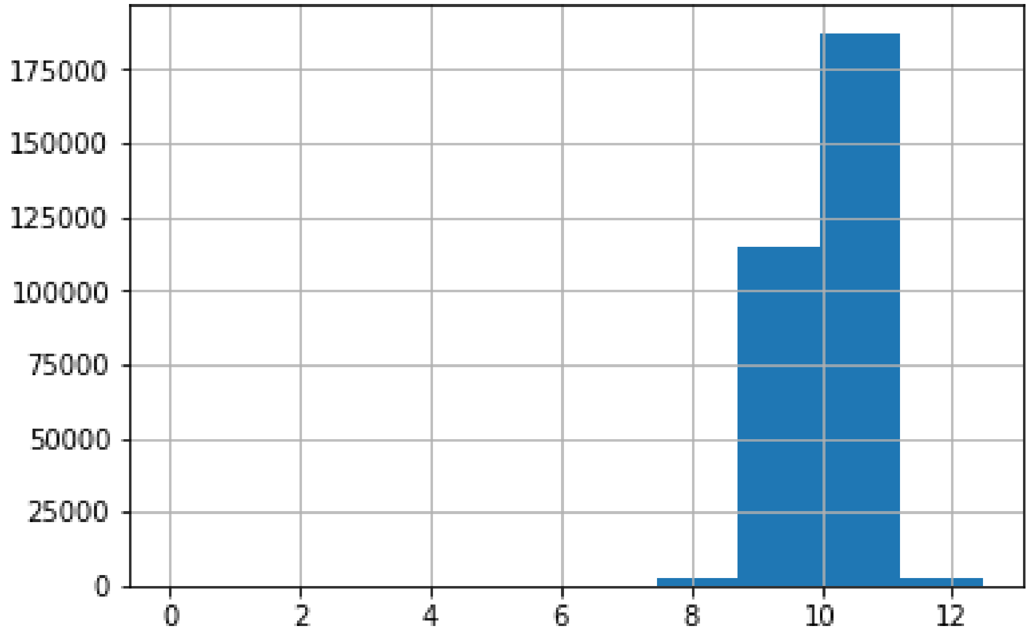
```
In [241]: np.log(application_bureau_loan_train_data['AMT_CREDIT'] + 1).hist()
          plt.show()
```



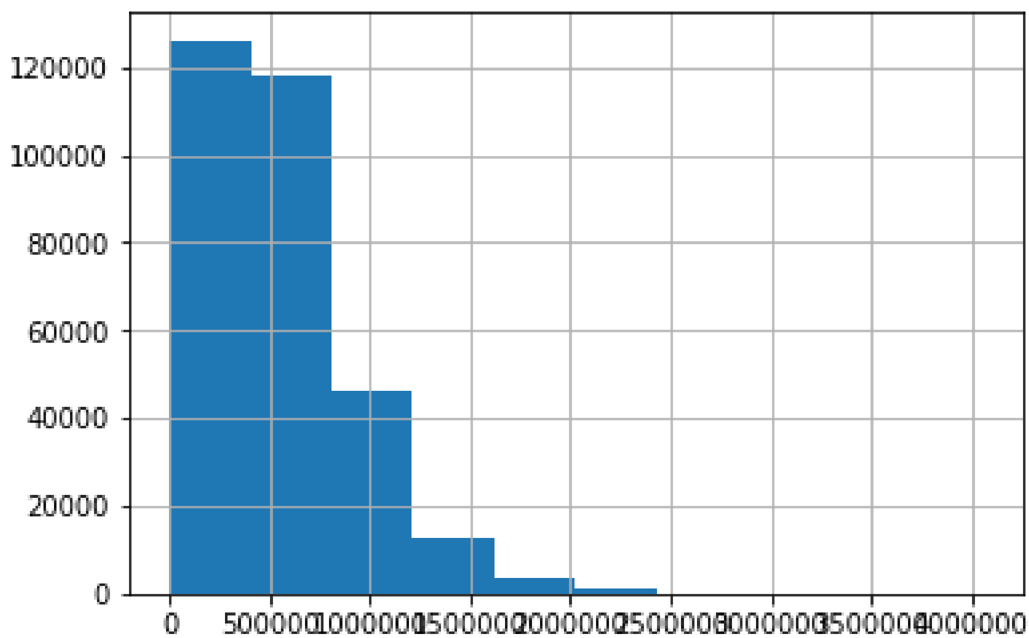
In [242]:
```
application_bureau_loan_train_data['AMT_ANNUITY'].hist()
plt.show()
```



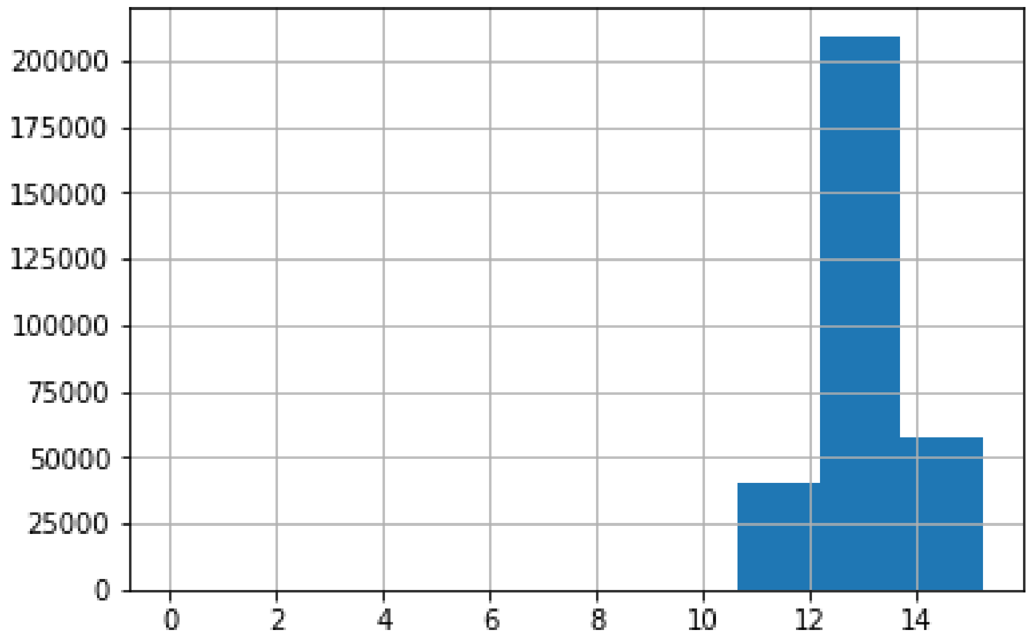
In [243]:
```
np.log(application_bureau_loan_train_data['AMT_ANNUITY'] + 1).hist()
plt.show()
```



In [244]:
```
application_bureau_loan_train_data['AMT_GOODS_PRICE'].hist()
plt.show()
```



```
In [245]: np.log(application_bureau_loan_train_data['AMT_GOODS_PRICE'] + 1).hist()
          plt.show()
```



1.7 Interpretation:

AMT_INCOME_TOTAL, AMT_CREDIT, AMT_ANNUITTY, AMT_GOODS_PRICE are not normal distrbiution as evident from teh histogram. But if we apply log transformation he histogram on thse fields become close to normal.

We will apply log transformation on fields 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE' and create a new dataframe application_bureau_loan_train_data_log

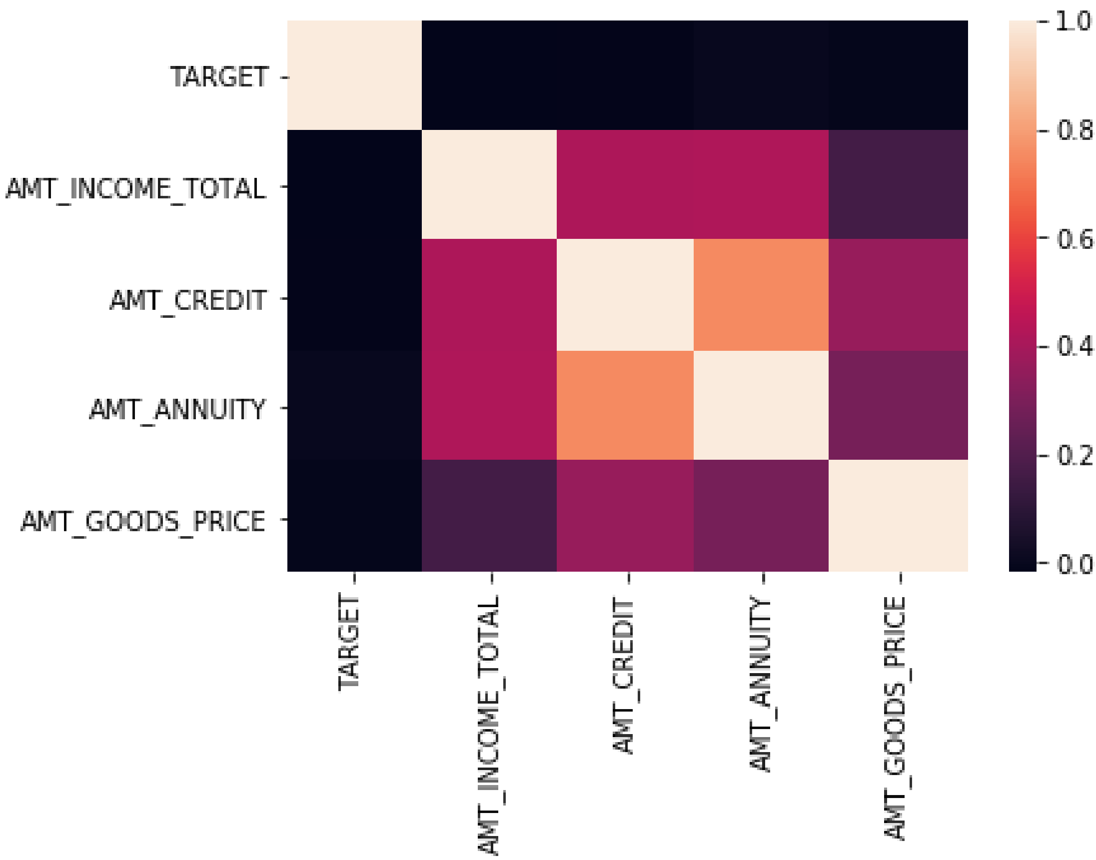
1.7.1 Linear correlation analysis of fields:

TARGET, AMT_INCOME_TOTAL,AMT_CREDIT,AMT_ANNUITY,AMT_GOODS_PRICE * First calculate correlation coefficinets * Draw the heatmap

```
In [253]: cor = application_bureau_loan_train_data_log[['TARGET','AMT_INCOME_TOTAL','AMT_CREDI
          print( "Correlation coefficients are:")
          print(str(cor))

          sns.heatmap(cor)
```

```
Correlation coefficients are:
                    TARGET  AMT_INCOME_TOTAL  AMT_CREDIT  AMT_ANNUITY  AMT_GOODS_PRICE
TARGET            1.000000         -0.017830   -0.010122     0.002893        -0.006468
AMT_INCOME_TOTAL -0.017830          1.000000    0.419369     0.422166         0.161980
AMT_CREDIT       -0.010122          0.419369    1.000000     0.752855         0.367808
AMT_ANNUITY       0.002893          0.422166    0.752855     1.000000         0.290103
AMT_GOODS_PRICE  -0.006468          0.161980    0.367808     0.290103         1.000000
```

Out[253]: <matplotlib.axes._subplots.AxesSubplot at 0x145029bee48>



1.8 Interpretation:

From the heamap and correlation coefficients, found that AMT_ANNUITY is strongly dependent on AMT_CREDIT

Also, AMT_CREDIT, AMT_ANNUITY is dependent on AMT_INCOME_TOTAL Also, AMT_CREDIT is to some extent dependent on AMT_GOODS_PRICE.

I will drop the column AMT_ANNUITY

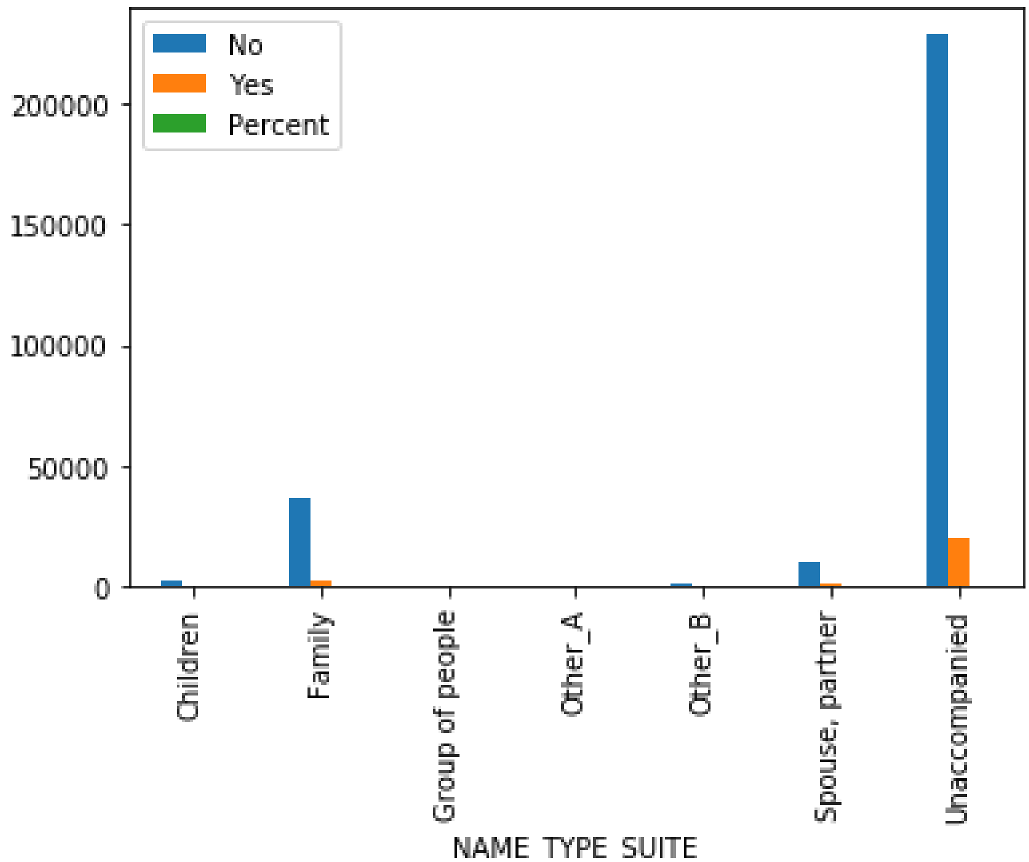
In [255]: application_bureau_loan_train_data_log['NAME_TYPE_SUITE'].value_counts()

```
Out[255]: Unaccompanied      248526
          Family              40149
          Spouse, partner     11370
          Children             3267
          Other_B              1770
          NA                   1292
          Other_A               866
          Group of people       271
          Name: NAME_TYPE_SUITE, dtype: int64
```

```
In [256]: tab = pd.crosstab(index=application_train_data['NAME_TYPE_SUITE'],columns=applicatio
          tab.columns = ['No','Yes']
          tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
          print(tab)
          tab.plot(kind='bar')
```

```
                     No    Yes   Percent
NAME_TYPE_SUITE
Children           3026    241  7.376798
Family            37140   3009  7.494583
Group of people     248     23  8.487085
Other_A             790     76  8.775982
Other_B            1596    174  9.830508
Spouse, partner   10475    895  7.871592
Unaccompanied    228189  20337  8.183047
```

```
Out[256]: <matplotlib.axes._subplots.AxesSubplot at 0x145029bef98>
```



In [257]:
```
application_bureau_loan_train_data_log['NAME_INCOME_TYPE'].value_counts()
```

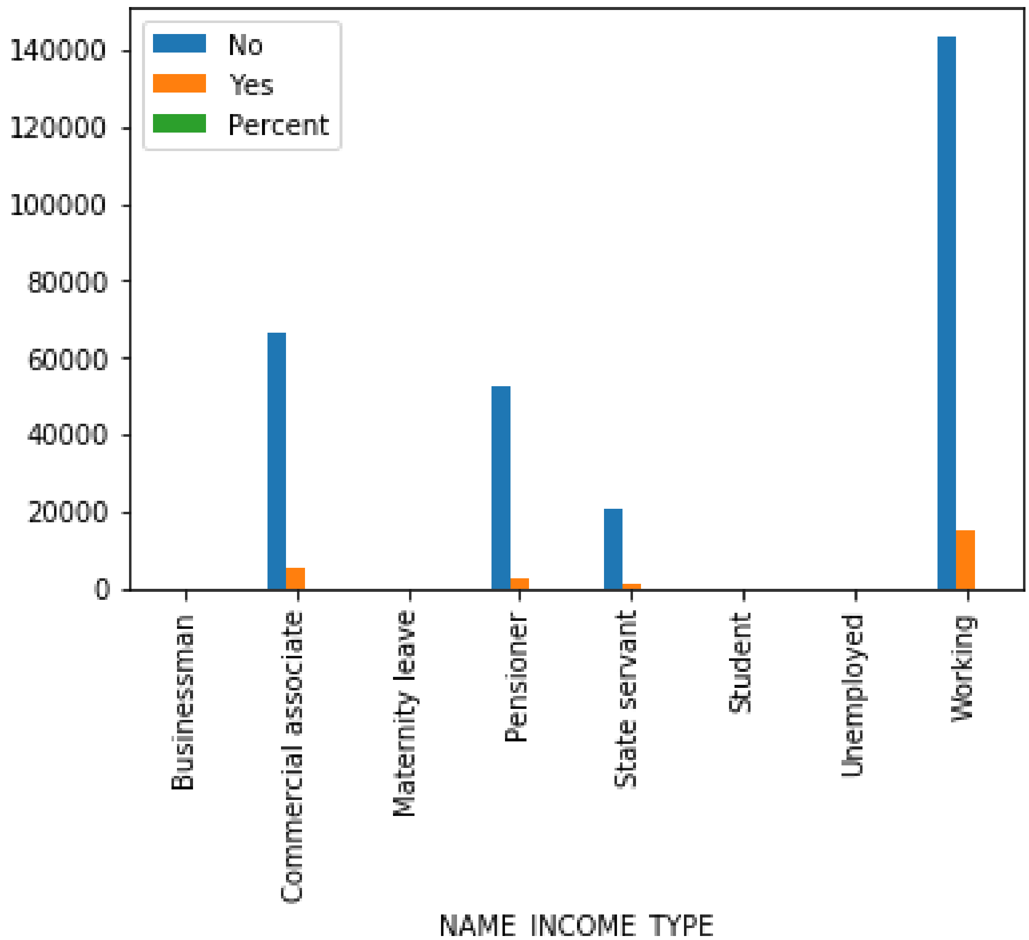
Out[257]:
```
Working                 158774
Commercial associate     71617
Pensioner                55362
State servant            21703
Unemployed                  22
Student                     18
Businessman                 10
Maternity leave              5
Name: NAME_INCOME_TYPE, dtype: int64
```

In [258]:
```
tab = pd.crosstab(index=application_bureau_loan_train_data_log['NAME_INCOME_TYPE'],co
tab.columns = ['No','Yes']
tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
print(tab)
tab.plot(kind='bar')
```

```
                          No     Yes    Percent
NAME_INCOME_TYPE
```

Businessman	10	0	0.000000
Commercial associate	66257	5360	7.484257
Maternity leave	3	2	40.000000
Pensioner	52380	2982	5.386366
State servant	20454	1249	5.754965
Student	18	0	0.000000
Unemployed	14	8	36.363636
Working	143550	15224	9.588472

```
Out[258]: <matplotlib.axes._subplots.AxesSubplot at 0x14500742e48>
```



1.9 Interpretation:

Working has been given more number of loans

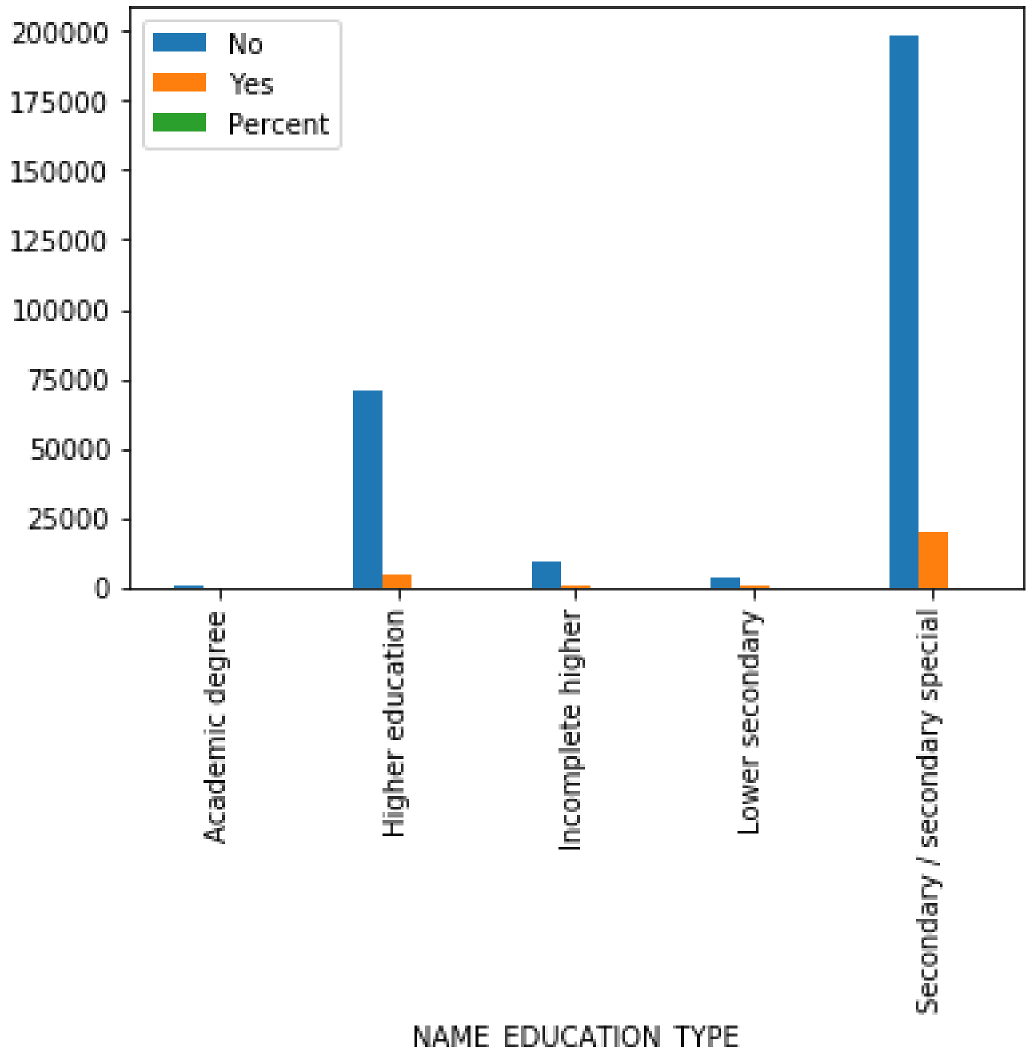
```
In [259]: application_bureau_loan_train_data_log['NAME_EDUCATION_TYPE'].value_counts()
```

```
Out[259]: Secondary / secondary special    218391
          Higher education                  74863
          Incomplete higher                 10277
          Lower secondary                    3816
          Academic degree                     164
          Name: NAME_EDUCATION_TYPE, dtype: int64
```

```
In [260]: tab = pd.crosstab(index=application_bureau_loan_train_data_log['NAME_EDUCATION_TYPE']
          tab.columns = ['No','Yes']
          tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
          print(tab)
          tab.plot(kind='bar')
```

```
                                   No    Yes    Percent
NAME_EDUCATION_TYPE
Academic degree                   161      3   1.829268
Higher education                70854   4009   5.355115
Incomplete higher                9405    872   8.484966
Lower secondary                  3399    417  10.927673
Secondary / secondary special  198867  19524   8.939929
```

```
Out[260]: <matplotlib.axes._subplots.AxesSubplot at 0x1450344cc88>
```



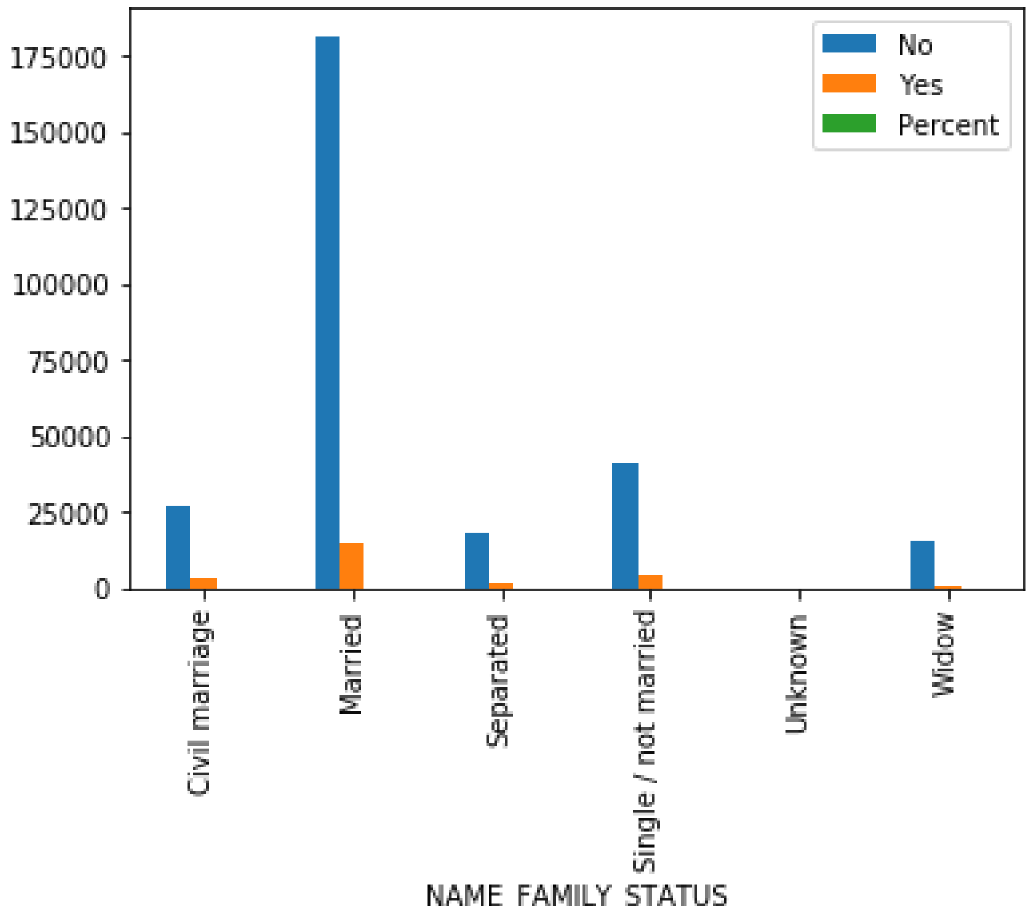
In [261]:
```
application_bureau_loan_train_data_log['NAME_FAMILY_STATUS'].value_counts()
```

Out[261]:
```
Married                 196432
Single / not married     45444
Civil marriage           29775
Separated                19770
Widow                    16088
Unknown                      2
Name: NAME_FAMILY_STATUS, dtype: int64
```

In [262]:
```
tab = pd.crosstab(index=application_bureau_loan_train_data_log['NAME_FAMILY_STATUS']
tab.columns = ['No','Yes']
tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
print(tab)
tab.plot(kind='bar')
```

```
                          No    Yes   Percent
NAME_FAMILY_STATUS
Civil marriage         26814   2961  9.944584
Married               181582  14850  7.559868
Separated              18150   1620  8.194234
Single / not married   40987   4457  9.807675
Unknown                    2      0  0.000000
Widow                  15151    937  5.824217
```

```
Out[262]: <matplotlib.axes._subplots.AxesSubplot at 0x14501e6c3c8>
```



1.10 Interpretation:

Married have been given more number of loans

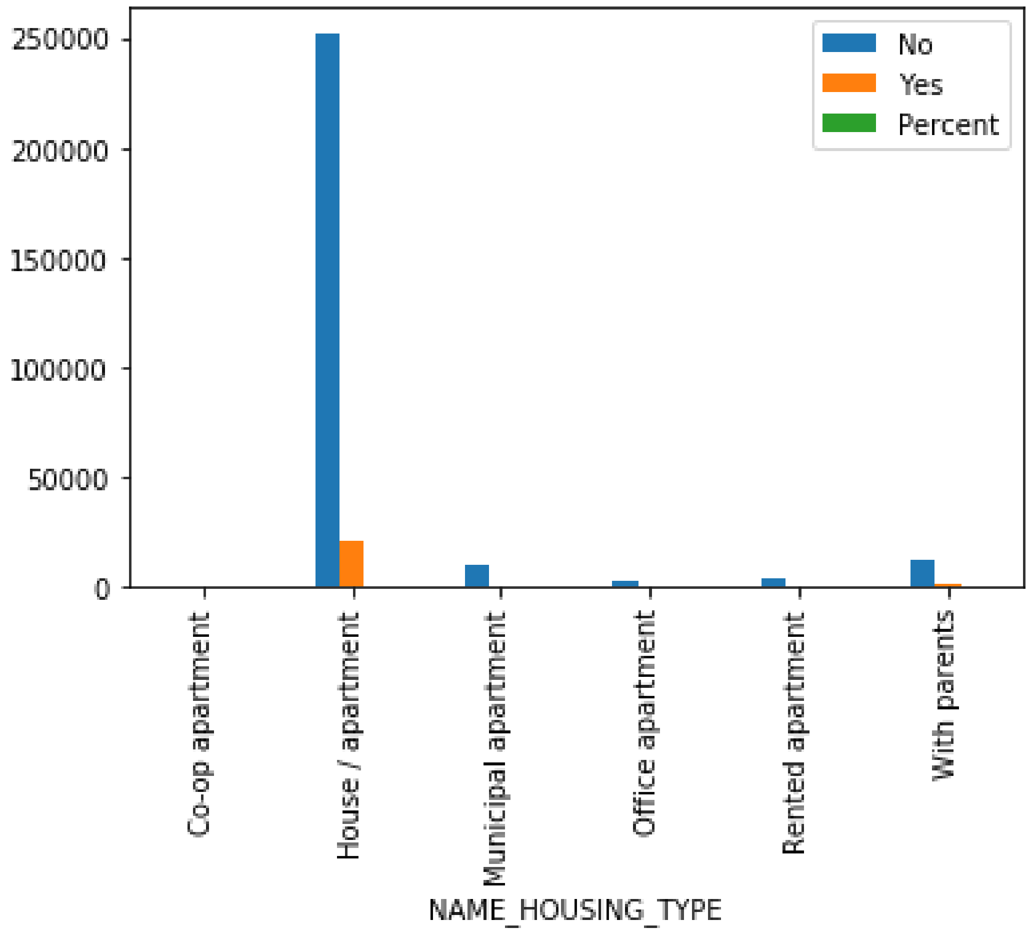
```
In [263]: application_bureau_loan_train_data_log['NAME_HOUSING_TYPE'].value_counts()
```

Out[263]:
```
House / apartment      272868
With parents            14840
Municipal apartment     11183
Rented apartment         4881
Office apartment         2617
Co-op apartment          1122
Name: NAME_HOUSING_TYPE, dtype: int64
```

In [264]:
```
tab = pd.crosstab(index=application_bureau_loan_train_data_log['NAME_HOUSING_TYPE'],c
tab.columns = ['No','Yes']
tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
print(tab)
tab.plot(kind='bar')
```

```
                         No    Yes    Percent
NAME_HOUSING_TYPE
Co-op apartment        1033     89   7.932264
House / apartment    251596  21272   7.795711
Municipal apartment   10228    955   8.539748
Office apartment       2445    172   6.572411
Rented apartment       4280    601  12.313051
With parents          13104   1736  11.698113
```

Out[264]: <matplotlib.axes._subplots.AxesSubplot at 0x1450076b438>



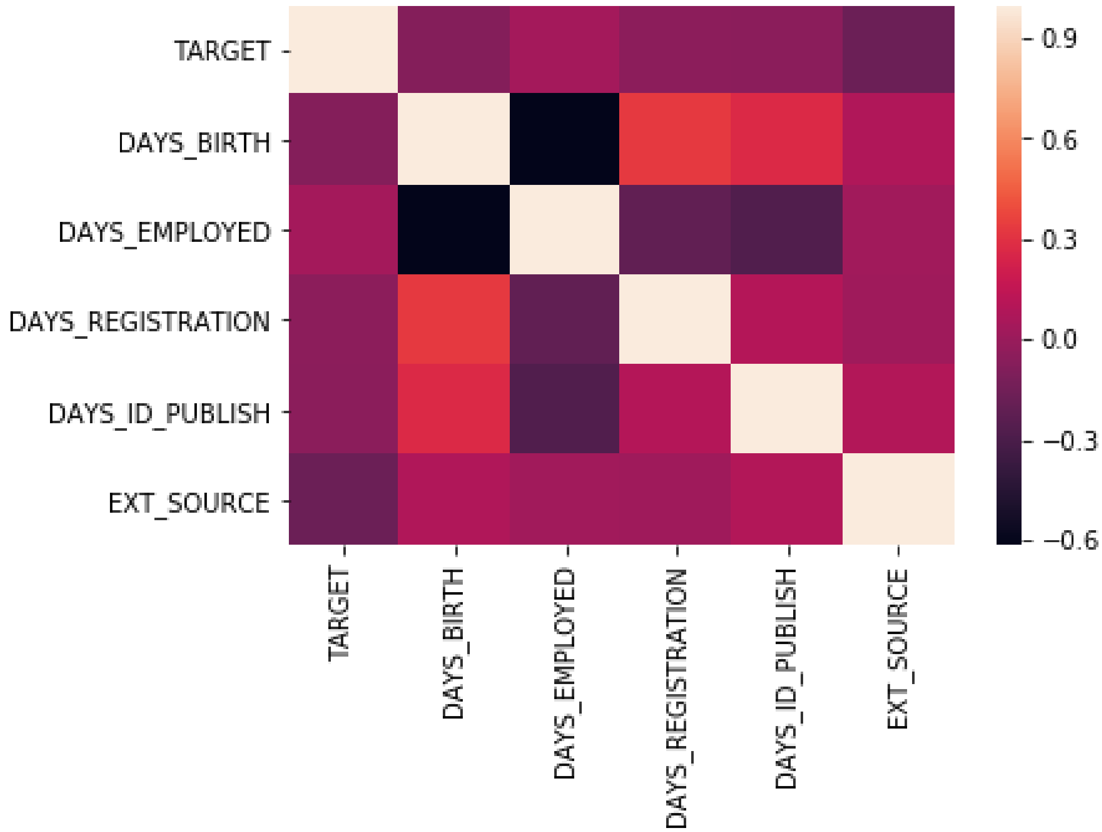
1.11 Interpretation:

House/apartment have been given more loan than any other category

```
In [266]: cor = application_bureau_loan_train_data_log[['TARGET','DAYS_BIRTH', 'DAYS_EMPLOYED'
          print( "Correlation coefficients are:")
          print(str(cor))

          sns.heatmap(cor)
```

```
Correlation coefficients are:
                      TARGET  DAYS_BIRTH  DAYS_EMPLOYED  DAYS_REGISTRATION  DAYS_ID_PUBLISH  EXT
TARGET              1.000000   -0.078239       0.044932          -0.041975        -0.051457   -(
DAYS_BIRTH         -0.078239    1.000000      -0.615864           0.331912         0.272691   (
DAYS_EMPLOYED       0.044932   -0.615864       1.000000          -0.210242        -0.272378   (
DAYS_REGISTRATION -0.041975    0.331912      -0.210242           1.000000         0.101896   (
DAYS_ID_PUBLISH    -0.051457    0.272691      -0.272378           0.101896         1.000000   (
EXT_SOURCE         -0.173322    0.087817       0.030691           0.027263         0.092992   1
```

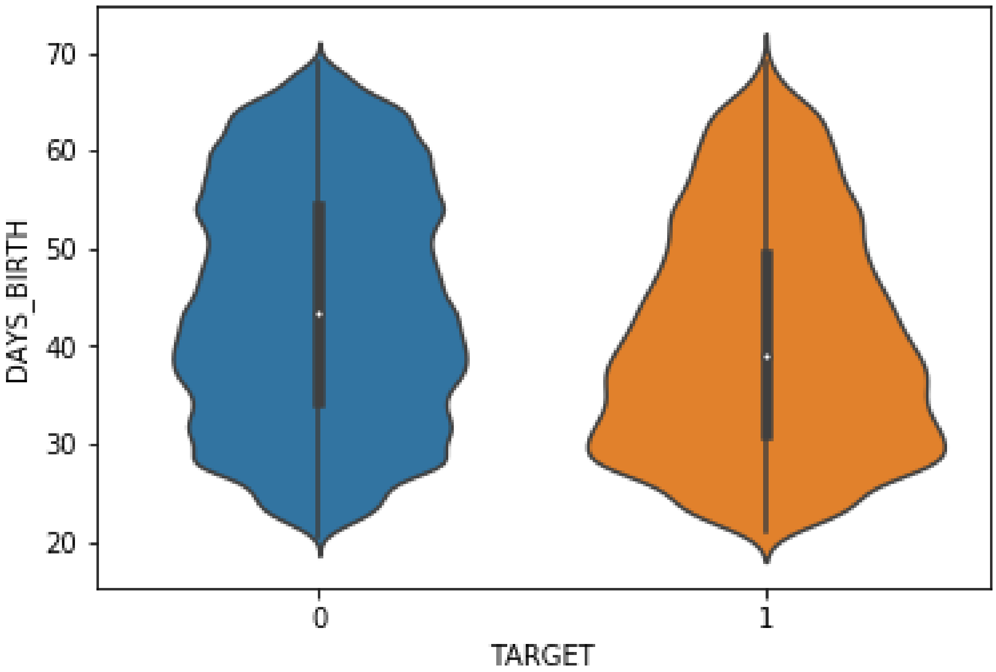
Out[266]: <matplotlib.axes._subplots.AxesSubplot at 0x145007dda58>



1.12 Interpretation:

DAYS_BIRTH has strong linear relationship with DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_ID_PUBLISH has some linear relationship with DAYS_BIRTH TARGET has little relationship with EXT_SOURCE

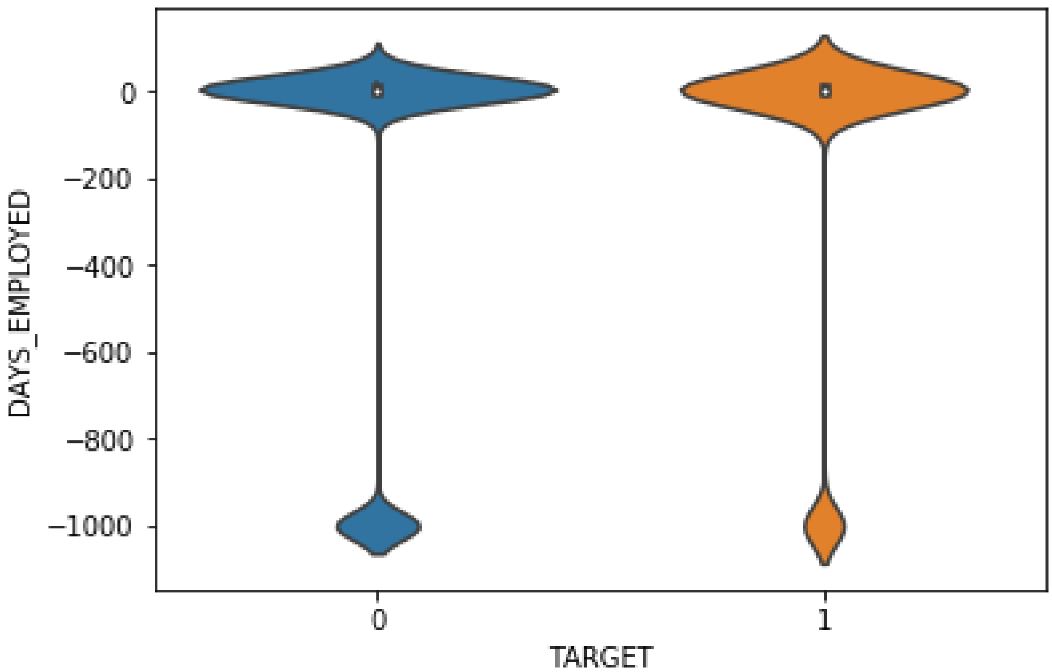
```
In [268]: sns.violinplot(x='TARGET',y='DAYS_BIRTH',data=application_bureau_loan_train_data_log
          plt.show()
```



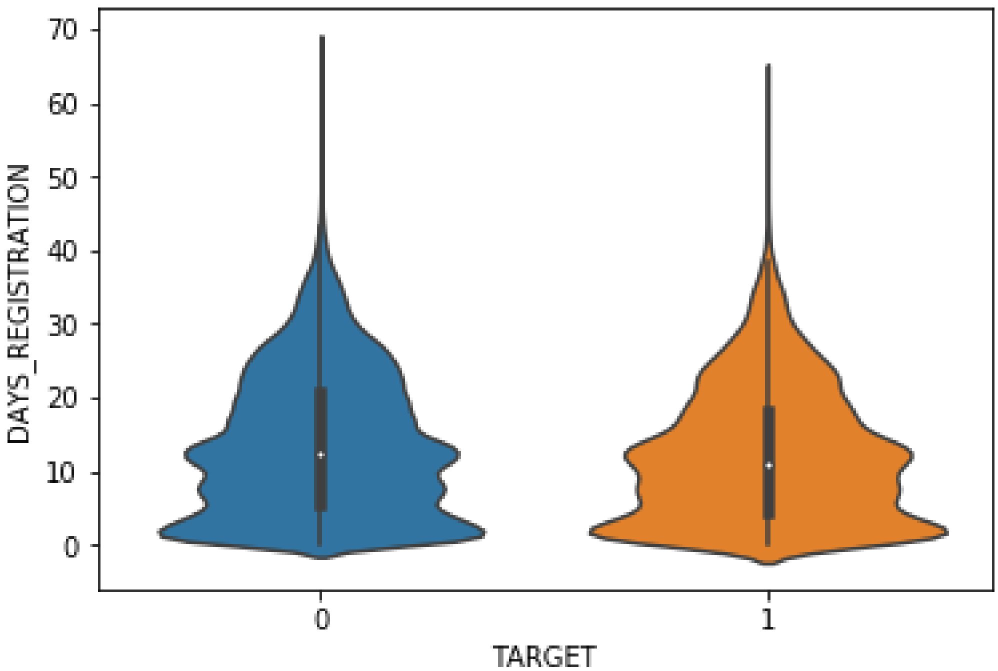
1.13 Interpretation:

Most loans have been given around age 30 after that loan approval rate sequencially decreases

```
In [269]: sns.violinplot(x='TARGET',y='DAYS_EMPLOYED',data=application_bureau_loan_train_data_l
          plt.show()
```



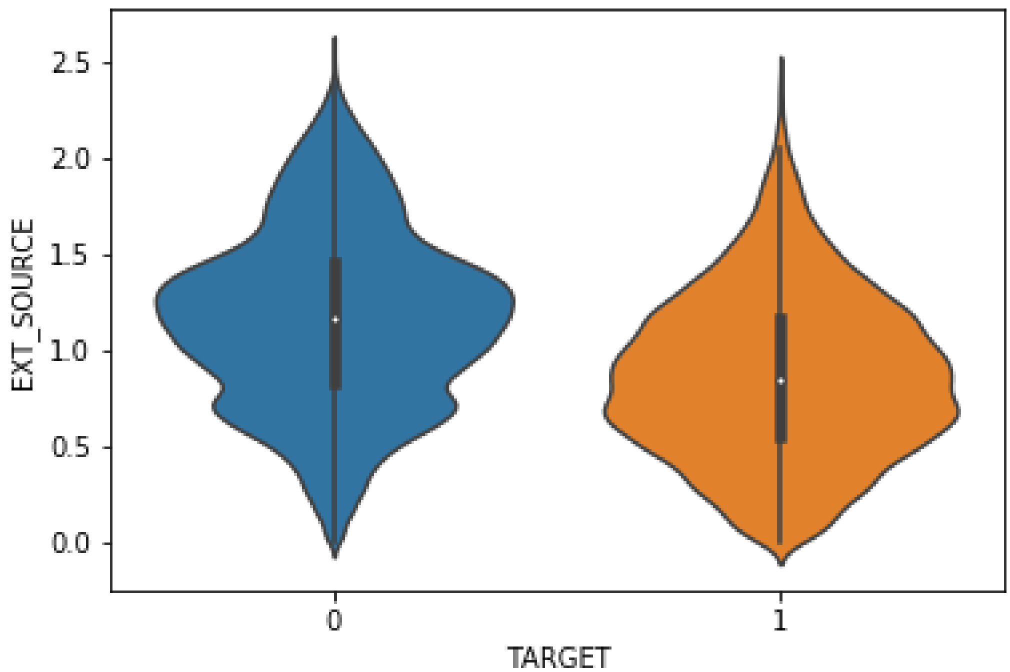
In [270]:
```
sns.violinplot(x='TARGET',y='DAYS_REGISTRATION',data=application_bureau_loan_train_da
plt.show()
```



1.14 Interpretation:

Interpretation: DAYS_REGISTRATION has strong linear relationship with TARGET

```
In [83]: sns.violinplot(x='TARGET',y='EXT_SOURCE',data=application_bureau_loan_train_data_log)
         plt.show()
```



1.15 Interpretation:

Interpretation: EXT_SOURCE has relationship with TARGET

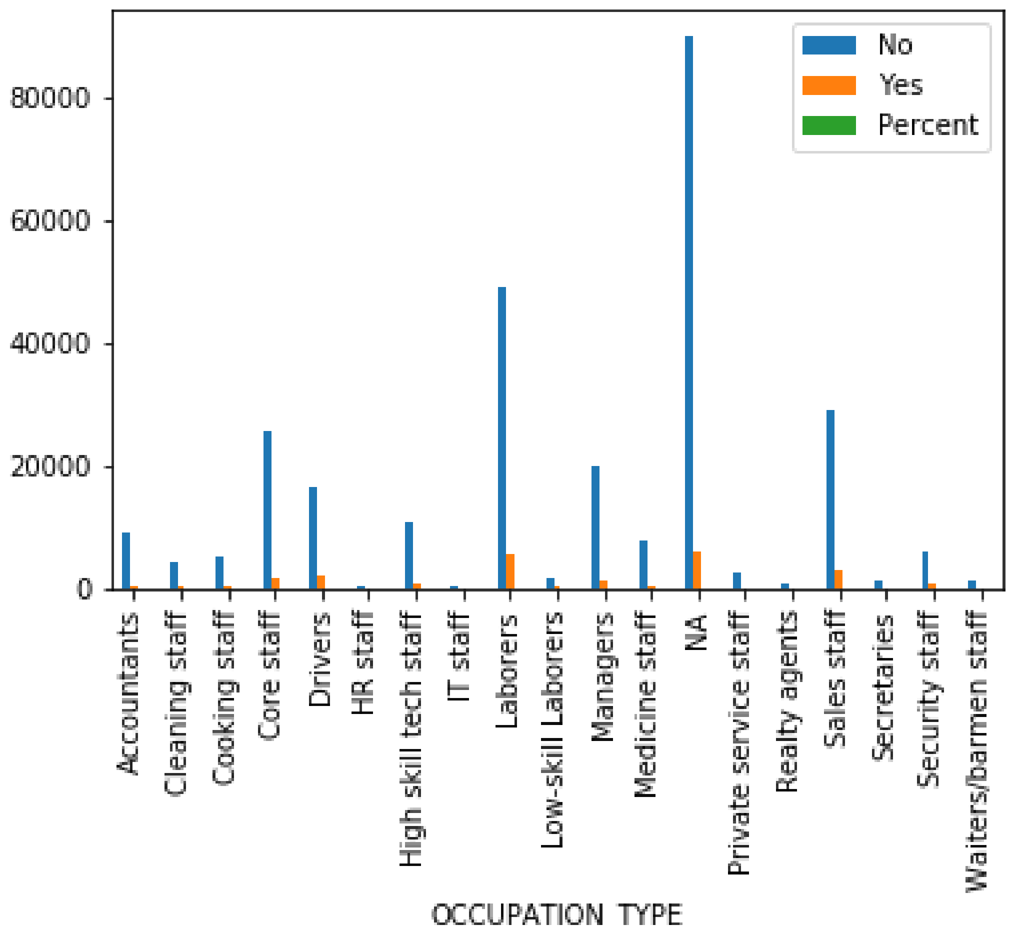
```
In [271]: application_bureau_loan_train_data_log['OCCUPATION_TYPE'].value_counts()

Out[271]: NA                       96391
          Laborers                 55186
          Sales staff              32102
          Core staff               27570
          Managers                 21371
          Drivers                  18603
          High skill tech staff    11380
          Accountants               9813
          Medicine staff            8537
          Security staff            6721
          Cooking staff             5946
          Cleaning staff            4653
```

```
Private service staff     2652
Low-skill Laborers        2093
Waiters/barmen staff      1348
Secretaries               1305
Realty agents              751
HR staff                   563
IT staff                   526
Name: OCCUPATION_TYPE, dtype: int64
```

In [272]:
```
tab = pd.crosstab(index=application_bureau_loan_train_data_log['OCCUPATION_TYPE'],co
tab.columns = ['No','Yes']
tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
print(tab)
tab.plot(kind='bar')
```

```
                          No   Yes    Percent
OCCUPATION_TYPE
Accountants             9339   474   4.830327
Cleaning staff          4206   447   9.606705
Cooking staff           5325   621  10.443996
Core staff             25832  1738   6.303954
Drivers                16496  2107  11.326130
HR staff                 527    36   6.394316
High skill tech staff  10679   701   6.159930
IT staff                 492    34   6.463878
Laborers               49348  5838  10.578770
Low-skill Laborers      1734   359  17.152413
Managers               20043  1328   6.214028
Medicine staff          7965   572   6.700246
NA                     90113  6278   6.513056
Private service staff   2477   175   6.598793
Realty agents            692    59   7.856192
Sales staff            29010  3092   9.631799
Secretaries             1213    92   7.049808
Security staff          5999   722  10.742449
Waiters/barmen staff    1196   152  11.275964
```

Out[272]: <matplotlib.axes._subplots.AxesSubplot at 0x14502abfef0>



1.16 Interpretation:

Laborers and NA have been given more loans

```
In [273]: application_bureau_loan_train_data_log['CNT_FAM_MEMBERS'].value_counts()
```

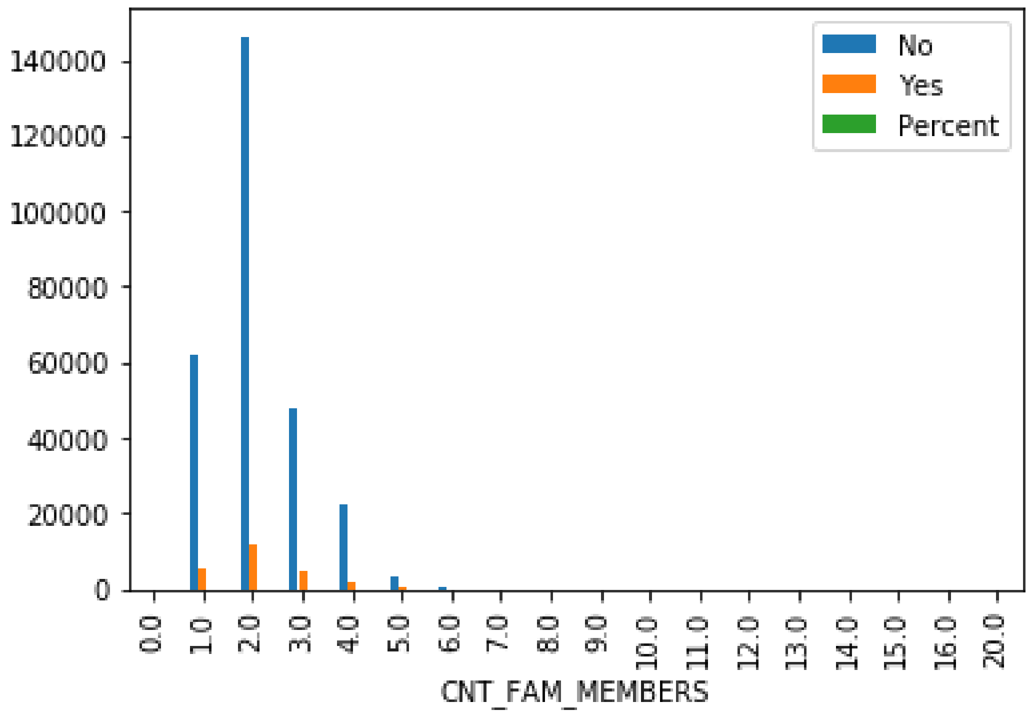
```
Out[273]: 2.0     158357
          1.0      67847
          3.0      52601
          4.0      24697
          5.0       3478
          6.0        408
          7.0         81
          8.0         20
          9.0          6
          10.0         3
          0.0          2
          20.0         2
```

```
16.0          2
12.0          2
14.0          2
15.0          1
13.0          1
11.0          1
Name: CNT_FAM_MEMBERS, dtype: int64
```

In [274]:
```
tab = pd.crosstab(index=application_bureau_loan_train_data_log['CNT_FAM_MEMBERS'],co
tab.columns = ['No','Yes']
tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
print(tab)
tab.plot(kind='bar')
```

```
                     No    Yes     Percent
CNT_FAM_MEMBERS
0.0                   2      0    0.000000
1.0               62172   5675    8.364408
2.0              146348  12009    7.583498
3.0               47993   4608    8.760290
4.0               22561   2136    8.648824
5.0                3151    327    9.401955
6.0                 353     55   13.480392
7.0                  75      6    7.407407
8.0                  14      6   30.000000
9.0                   6      0    0.000000
10.0                  2      1   33.333333
11.0                  0      1  100.000000
12.0                  2      0    0.000000
13.0                  0      1  100.000000
14.0                  2      0    0.000000
15.0                  1      0    0.000000
16.0                  2      0    0.000000
20.0                  2      0    0.000000
```

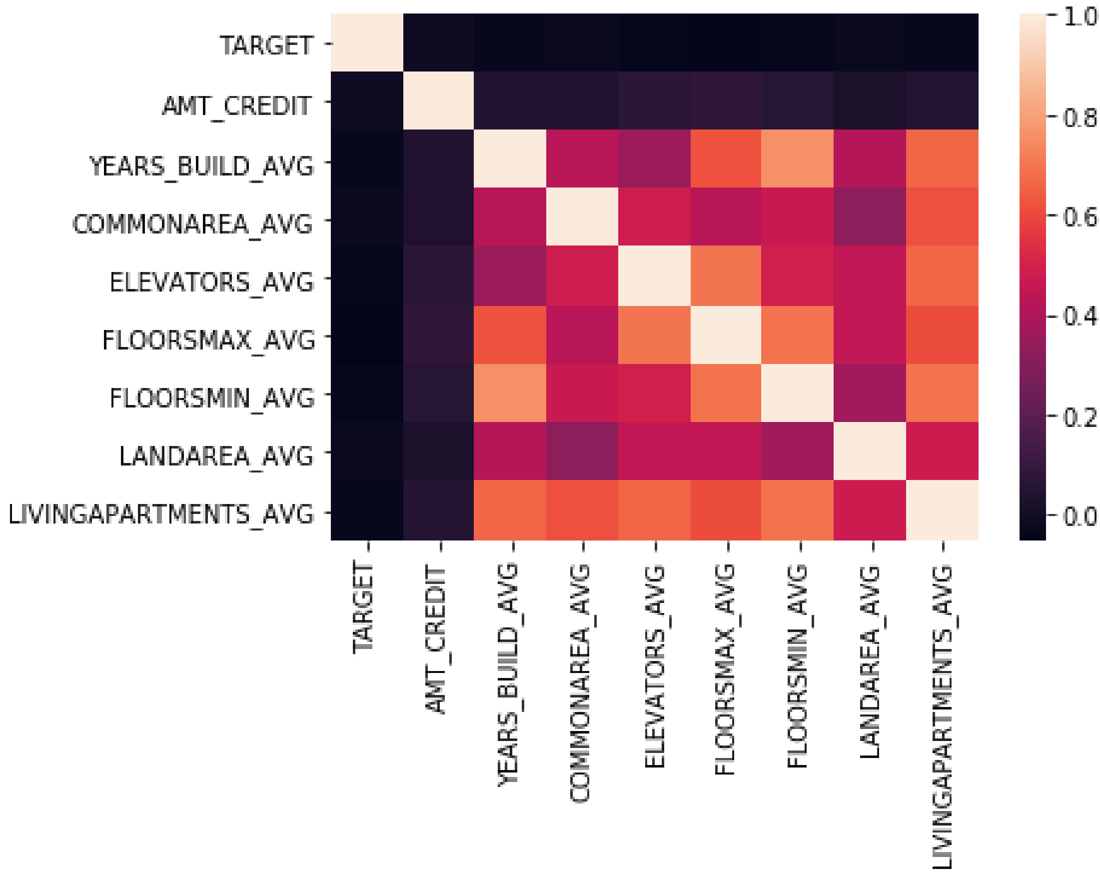
Out[274]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14502a39cc0>
```



In [275]:
```
cor = application_bureau_loan_train_data_log[['TARGET','AMT_CREDIT','YEARS_BUILD_AVG
print( "Correlation coefficients are:")
print(str(cor))

sns.heatmap(cor)
```

```
Correlation coefficients are:
                       TARGET  AMT_CREDIT  YEARS_BUILD_AVG  COMMONAREA_AVG  ELEVATORS_AVG  FL(
TARGET               1.000000   -0.010122        -0.033073       -0.020853      -0.035853
AMT_CREDIT          -0.010122    1.000000         0.044097        0.039327       0.068086
YEARS_BUILD_AVG     -0.033073    0.044097         1.000000        0.428953       0.357608
COMMONAREA_AVG      -0.020853    0.039327         0.428953        1.000000       0.480672
ELEVATORS_AVG       -0.035853    0.068086         0.357608        0.480672       1.000000
FLOORSMAX_AVG       -0.049839    0.081633         0.620825        0.426768       0.695423
FLOORSMIN_AVG       -0.034177    0.058867         0.759495        0.468471       0.492523
LANDAREA_AVG        -0.023152    0.026208         0.417323        0.323363       0.448520
LIVINGAPARTMENTS_AVG -0.029525   0.049818         0.661667        0.616901       0.660955
```

Out[275]: <matplotlib.axes._subplots.AxesSubplot at 0x14507609978>



1.17 Interpretation:

TARGET and CREDIT_AMOUNT has no linear relationship with any other fields, LIVINGAPARTMENTS_AVG has strong linear relationship with YEARS_BUILD_AVG, COMMON_AREA_AVG, ELEVATORS_AVG, FLOOR_MIN_AVG,FLOOR_MAX_AVG,FLOORSMIN_AVG and medium relationship with LANDAREA_AVG.

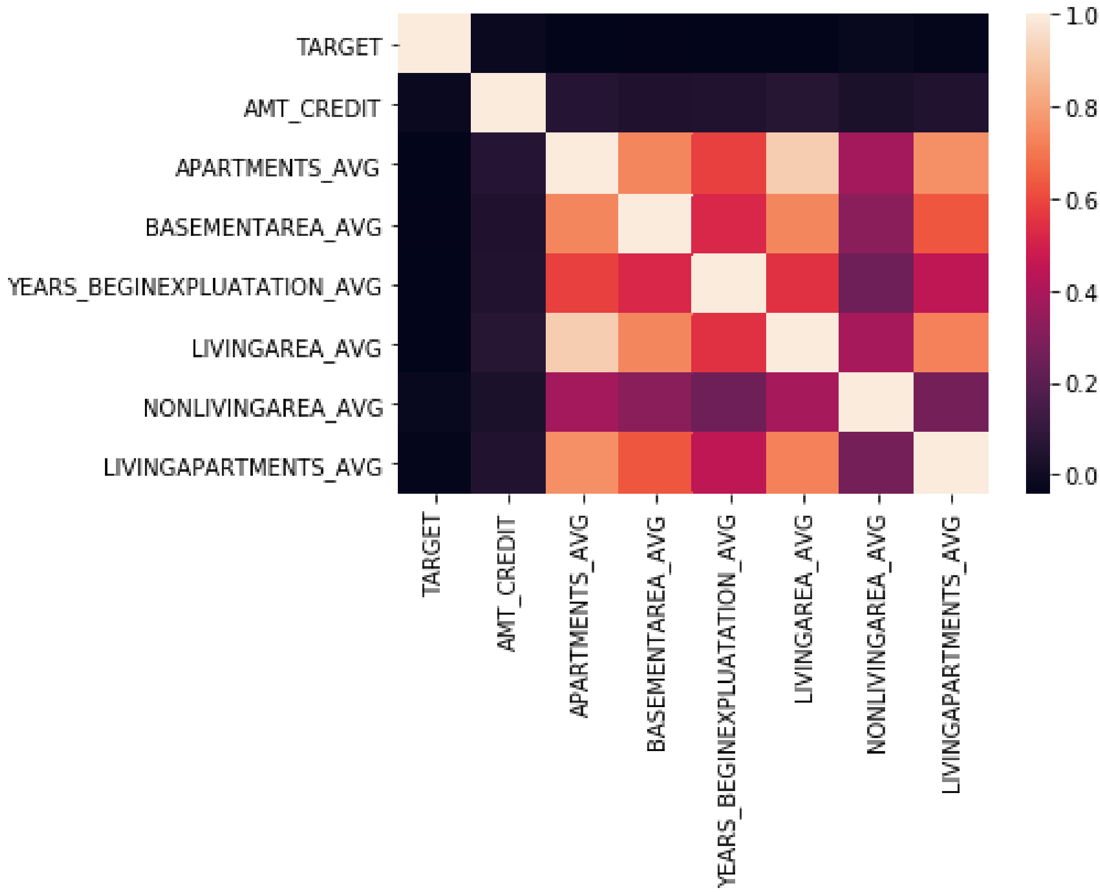
We will drop the columns COMMON_AREA_AVG, ELEVATORS_AVG, FLOOR_MIN_AVG,FLOOR_MAX_AVG,FLOORSMIN_AVG

```
In [276]: cor = application_bureau_loan_train_data_log[['TARGET','AMT_CREDIT', 'APARTMENTS_AVG
          print( "Correlation coefficients are:")
          print(str(cor))

          sns.heatmap(cor)
```

```
Correlation coefficients are:
                               TARGET  AMT_CREDIT  APARTMENTS_AVG  BASEMENTAREA_AVG  YEARS_BEG
TARGET                       1.000000   -0.010122       -0.039924         -0.033759
AMT_CREDIT                  -0.010122    1.000000        0.062026          0.047030
```

```
APARTMENTS_AVG               -0.039924    0.062026        1.000000          0.737313
BASEMENTAREA_AVG             -0.033759    0.047030        0.737313          1.000000
YEARS_BEGINEXPLUATATION_AVG -0.040965    0.049162        0.589120          0.523454
LIVINGAREA_AVG               -0.040301    0.066065        0.916381          0.737142
NONLIVINGAREA_AVG            -0.019446    0.035985        0.381249          0.320550
LIVINGAPARTMENTS_AVG         -0.029525    0.049818        0.759989          0.633589
```

```
Out[276]: <matplotlib.axes._subplots.AxesSubplot at 0x145076cae80>
```



1.18 Interpretation:

LIVINGAPARTMENTS_AVG has strong linear relationship with APARTMENTS_AVG, BASEMENTAREA_AVG, LIVINGAREA_AVG

LIVINGAREA_AVG has very strong linear relationship with APARTMENTS_AVG

We can remove the columns APARTMENTS_AVG, BASEMENTAREA_AVG, LIVINGAREA_AVG

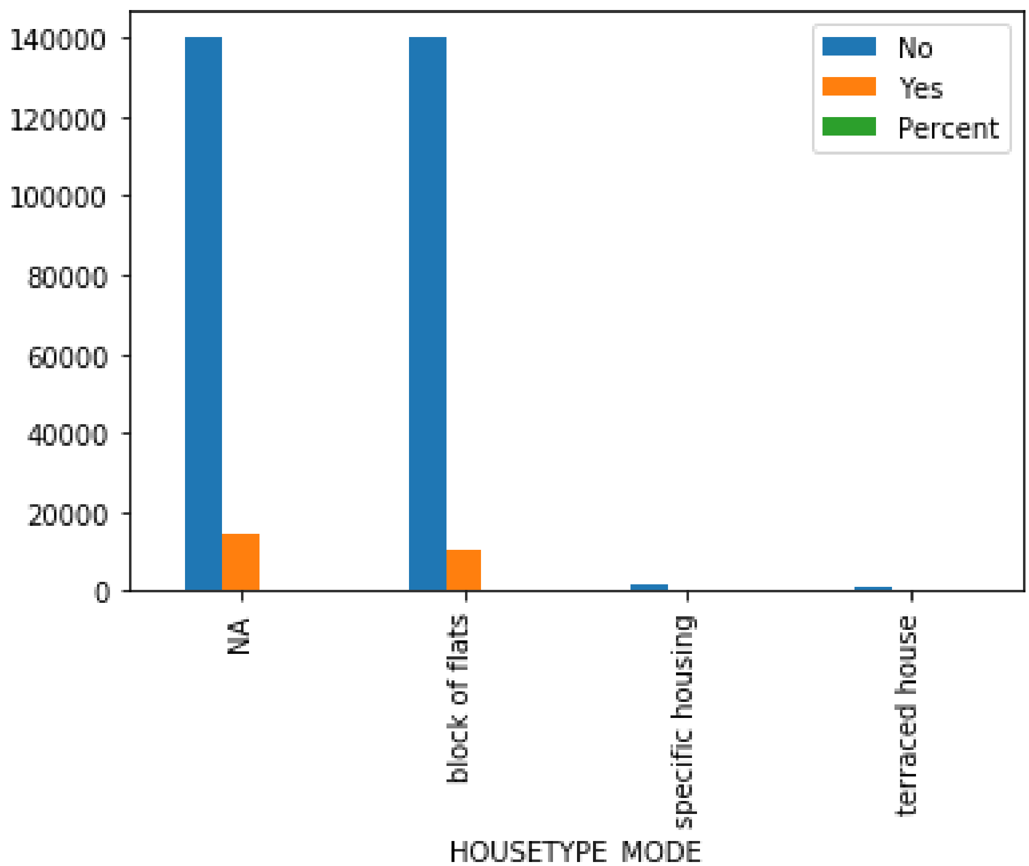
```
In [90]: application_bureau_loan_train_data_log['HOUSETYPE_MODE'].value_counts()
```

```
Out[90]: NA                    154297
         block of flats        150503
         specific housing        1499
         terraced house          1212
         Name: HOUSETYPE_MODE, dtype: int64
```

```
In [277]: tab = pd.crosstab(index=application_bureau_loan_train_data_log['HOUSETYPE_MODE'],colu
          tab.columns = ['No','Yes']
          tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
          print(tab)
          tab.plot(kind='bar')
```

```
                      No    Yes    Percent
HOUSETYPE_MODE
NA                140177  14120   9.151182
block of flats    140053  10450   6.943383
specific housing    1347    152  10.140093
terraced house      1109    103   8.498350
```

```
Out[277]: <matplotlib.axes._subplots.AxesSubplot at 0x14507c41048>
```



In [278]: application_bureau_loan_train_data_log['TOTALAREA_MODE'].value_counts()

```
Out[278]: 0.0000    149013
          0.0570       247
          0.0547       230
          0.0550       227
          0.0548       227
          0.0555       227
          0.0551       225
          0.0554       220
          0.0573       220
          0.0566       219
          0.0556       217
          0.0559       216
          0.0543       214
          0.0529       212
          0.0603       211
          0.0552       208
          0.0541       205
          0.0689       205
          0.0525       204
          0.0064       204
          0.0067       203
          0.0500       203
          0.0502       201
          0.0017       200
          0.0688       199
          0.0066       198
          0.0526       198
          0.0540       197
          0.0536       196
          0.0687       195
          0.0532       195
          0.0569       194
          0.0553       194
          0.0538       194
          0.0063       192
          0.0528       192
          0.0498       192
          0.0607       190
          0.0533       189
          0.0018       189
          0.0686       188
          0.0546       188
          0.0568       188
          0.0558       187
          0.0564       186
          0.0531       185
```

0.0520	185
0.0423	185
0.0549	185
0.0019	184
0.0544	184
0.0530	184
0.0080	184
0.0545	184
0.0557	184
0.0572	183
0.0521	183
0.0503	183
0.0539	182
0.0609	182
0.0696	182
0.0694	182
0.0402	181
0.0542	181
0.0711	181
0.0600	181
0.0563	180
0.0562	180
0.0690	180
0.0535	180
0.0065	179
0.0083	179
0.0604	178
0.0684	178
0.0574	177
0.0702	177
0.0086	177
0.0534	175
0.0079	175
0.0565	174
0.0501	174
0.0610	174
0.0685	174
0.0681	174
0.0523	173
0.0601	173
0.0504	173
0.0571	172
0.0082	172
0.0068	172
0.0567	171
0.0578	171
0.0062	170
0.0605	170

0.0495	170
0.0703	169
0.0561	169
0.0490	169
0.0059	169
0.0524	168
0.0598	168
0.0403	168
0.0699	168
0.0088	168
0.0427	168
0.0491	168
0.0602	168
0.0424	167
0.0612	167
0.0408	166
0.0522	166
0.0057	166
0.0692	165
0.0517	165
0.0069	165
0.0537	165
0.0016	164
0.0512	164
0.0691	163
0.0428	163
0.0506	163
0.0683	163
0.0070	163
0.0098	162
0.0081	162
0.0421	162
0.0707	162
0.0599	161
0.0099	161
0.0672	161
0.0616	161
0.0084	161
0.0432	160
0.0114	160
0.0061	160
0.0405	160
0.0078	160
0.0527	160
0.0493	160
0.0560	159
0.0100	159
0.0575	159

0.0698	158
0.0060	157
0.0715	157
0.0766	157
0.0115	157
0.0497	157
0.0426	156
0.0113	156
0.0089	156
0.0507	155
0.0704	155
0.0492	155
0.0116	155
0.0087	155
0.0695	155
1.0000	155
0.0608	154
0.0594	154
0.0077	154
0.0494	154
0.0023	153
0.0714	153
0.0434	152
0.0645	152
0.0425	152
0.0717	152
0.0097	151
0.0020	151
0.0595	151
0.0422	150
0.0499	150
0.0085	150
0.0596	150
0.0693	150
0.0313	149
0.0753	149
0.0075	149
0.0597	149
0.0743	149
0.0516	148
0.0756	148
0.0058	148
0.0654	148
0.0701	148
0.0121	147
0.0650	147
0.0662	147
0.0396	147

0.0090	147
0.0505	146
0.0101	146
0.0700	145
0.0615	145
0.0111	145
0.0679	145
0.0404	145
0.0406	145
0.0719	145
0.0712	144
0.0395	144
0.0117	143
0.0508	143
0.0591	143
0.0519	143
0.0496	143
0.0510	143
0.0697	142
0.0401	142
0.0708	142
0.0487	142
0.0124	142
0.0489	141
0.0400	141
0.0613	141
0.0074	141
0.0606	140
0.0739	140
0.0579	140
0.0478	139
0.0661	139
0.0675	139
0.0764	139
0.0053	139
0.0509	138
0.0515	138
0.0076	138
0.0706	138
0.0680	137
0.0430	137
0.0107	137
0.0479	137
0.0577	137
0.0646	137
0.0720	137
0.0112	136
0.0482	136

0.0710	136
0.0757	136
0.0765	136
0.0673	135
0.0071	135
0.0713	135
0.0623	135
0.0133	135
0.0581	135
0.0754	135
0.0716	134
0.0410	134
0.0677	134
0.0096	134
0.0103	134
0.0429	134
0.0108	134
0.0024	134
0.0399	134
0.0468	134
0.0072	133
0.0664	133
0.0475	133
0.0407	133
0.0669	133
0.0774	133
0.0657	133
0.0511	133
0.0441	132
0.0670	132
0.0122	132
0.0091	132
0.0022	132
0.0767	132
0.0621	131
0.0593	131
0.0768	131
0.0470	130
0.0576	130
0.0393	130
0.0611	130
0.0021	130
0.0095	130
0.0433	130
0.0663	130
0.0102	130
0.0750	130
0.0477	130

0.0651	130
0.0660	130
0.0092	129
0.0583	129
0.0518	129
0.0025	129
0.0634	129
0.0105	128
0.0129	128
0.0674	128
0.0449	128
0.0723	127
0.0394	127
0.0588	127
0.0435	127
0.0653	127
0.0476	127
0.0724	127
0.0415	127
0.0514	126
0.0436	126
0.0763	126
0.0480	126
0.0440	126
0.0513	126
0.0647	126
0.0614	126
0.0752	125
0.0620	125
0.0416	125
0.0676	124
0.0640	124
0.0488	124
0.0586	124
0.0741	124
0.0667	124
0.0413	124
0.0659	124
0.0409	124
0.0633	124
0.0486	123
0.0652	123
0.0144	123
0.0123	123
0.0484	122
0.0584	122
0.0658	122
0.0104	122

0.0755	122
0.0471	122
0.0148	122
0.0474	122
0.0727	121
0.0131	121
0.0043	121
0.0668	121
0.0110	121
0.0682	121
0.0666	121
0.0134	121
0.0414	121
0.0469	121
0.0589	120
0.0015	120
0.0462	120
0.0120	120
0.0648	120
0.0902	120
0.0590	120
0.0709	120
0.0671	120
0.0047	120
0.0580	120
0.0582	120
0.0200	119
0.0630	119
0.0136	118
0.0149	118
0.0106	118
0.0617	118
0.0417	118
0.0635	118
0.0420	118
0.0198	117
0.0397	117
0.0622	117
0.0897	116
0.0483	116
0.0735	116
0.0056	116
0.0128	116
0.0782	116
0.0439	116
0.0448	115
0.0013	115
0.0452	115

0.0447	115
0.0751	115
0.0141	115
0.0705	115
0.0749	114
0.0729	114
0.0718	114
0.0215	114
0.0626	114
0.0052	114
0.0446	114
0.0746	114
0.0770	113
0.0485	113
0.0073	113
0.0127	113
0.0745	113
0.0014	113
0.0761	113
0.0431	112
0.0398	112
0.0467	112
0.0438	112
0.0730	112
0.0027	112
0.0054	112
0.0391	112
0.0445	112
0.0132	111
0.0463	111
0.0585	111
0.0412	111
0.0411	110
0.0895	110
0.0656	110
0.0728	110
0.0737	110
0.0443	110
0.0629	110
0.0678	110
0.0619	109
0.0587	109
0.0119	109
0.0740	109
0.0042	109
0.0632	109
0.0093	109
0.0734	109

0.0722	109
0.0726	109
0.0130	109
0.0638	108
0.0437	108
0.0029	108
0.0780	108
0.0758	107
0.0655	107
0.0051	107
0.0419	107
0.0802	107
0.0760	107
0.0773	107
0.0143	106
0.0644	106
0.0210	106
0.0725	106
0.0733	106
0.0618	106
0.0118	105
0.0442	105
0.0151	105
0.0150	105
0.0779	105
0.0335	105
0.0665	105
0.0641	105
0.0911	105
0.0026	105
0.0318	105
0.0744	104
0.0460	104
0.0450	104
0.0807	104
0.0135	104
0.0138	104
0.0314	104
0.0202	104
0.0759	104
0.0464	104
0.0592	104
0.0481	103
0.0636	103
0.0721	103
0.0778	103
0.0896	103
0.0736	103

0.0139	102
0.0044	102
0.0762	102
0.0627	102
0.0909	102
0.0227	102
0.0747	101
0.0142	101
0.0321	101
0.0784	101
0.0234	101
0.0444	101
0.0213	101
0.0748	101
0.0769	101
0.0164	101
0.0901	101
0.0899	100
0.0459	100
0.0742	100
0.0055	100
0.0904	100
	...
0.4067	1
0.3368	1
0.6188	1
0.6320	1
0.6867	1
0.3099	1
0.4295	1
0.3680	1
0.2940	1
0.4343	1
0.6767	1
0.7938	1
0.5376	1
0.5027	1
0.9629	1
0.8050	1
0.4309	1
0.4414	1
0.5629	1
0.3617	1
0.9127	1
0.4695	1
0.2791	1
0.3445	1
0.6872	1

0.7334	1
0.7925	1
0.4913	1
0.3963	1
0.9746	1
0.5550	1
0.5788	1
0.6895	1
0.4323	1
0.2583	1
0.5134	1
0.6076	1
0.5025	1
0.5157	1
0.8187	1
0.4824	1
0.3759	1
0.4492	1
0.8674	1
0.7725	1
0.2905	1
0.4261	1
0.4608	1
0.7580	1
0.3986	1
0.3977	1
0.7970	1
0.6308	1
0.9750	1
0.7010	1
0.5067	1
0.3558	1
0.4926	1
0.3131	1
0.3777	1
0.3834	1
0.6102	1
0.6462	1
0.6233	1
0.4506	1
0.3677	1
0.4193	1
0.3526	1
0.2763	1
0.5775	1
0.5594	1
0.3032	1
0.8900	1

0.5469	1
0.6096	1
0.6426	1
0.6097	1
0.4870	1
0.3128	1
0.3774	1
0.4772	1
0.3735	1
0.3928	1
0.6639	1
0.3756	1
0.5071	1
0.4610	1
0.5528	1
0.5357	1
0.5552	1
0.5777	1
0.4601	1
0.8754	1
0.3130	1
0.4376	1
0.6130	1
0.4107	1
0.4790	1
0.4828	1
0.5266	1
0.7510	1
0.8301	1
0.4478	1
0.4490	1
0.6931	1
0.3755	1
0.4200	1
0.5452	1
0.3813	1
0.2710	1
0.7557	1
0.4165	1
0.7349	1
0.4319	1
0.4035	1
0.3327	1
0.5013	1
0.6850	1
0.3838	1
0.5748	1
0.8856	1

0.4743	1
0.4557	1
0.5994	1
0.7953	1
0.4669	1
0.5906	1
0.5756	1
0.3639	1
0.4960	1
0.4689	1
0.4320	1
0.6153	1
0.4543	1
0.4189	1
0.4739	1
0.4388	1
0.6795	1
0.5540	1
0.5336	1
0.6384	1
0.4033	1
0.5545	1
0.4208	1
0.6214	1
0.5685	1
0.5978	1
0.4461	1
0.5827	1
0.5135	1
0.6062	1
0.6293	1
0.4820	1
0.4573	1
0.7674	1
0.4446	1
0.4186	1
0.4589	1
0.3658	1
0.4968	1
0.6491	1
0.7864	1
0.4118	1
0.3894	1
0.3699	1
0.5207	1
0.4598	1
0.4712	1
0.4195	1

0.4771	1
0.6494	1
0.6134	1
0.6192	1
0.4890	1
0.4023	1
0.4215	1
0.5206	1
0.4799	1
0.7980	1
0.6376	1
0.6246	1
0.9532	1
0.4132	1
0.3685	1
0.3945	1
0.6693	1
0.9034	1
0.7791	1
0.4875	1
0.4042	1
0.4630	1
0.7819	1
0.3616	1
0.5536	1
0.5220	1
0.3303	1
0.7712	1
0.8006	1
0.5419	1
0.8178	1
0.6861	1
0.6490	1
0.6995	1
0.4403	1
0.6089	1
0.7384	1
0.3781	1
0.4233	1
0.6391	1
0.6507	1
0.6222	1
0.4333	1
0.6123	1
0.8350	1
0.4119	1
0.7294	1
0.4219	1

0.6747	1
0.5495	1
0.4644	1
0.3782	1
0.4328	1
0.3584	1
0.5566	1
0.4059	1
0.6613	1
0.5095	1
0.6580	1
0.6633	1
0.2947	1
0.4634	1
0.5791	1
0.9749	1
0.3175	1
0.3778	1
0.4661	1
0.5783	1
0.8043	1
0.3893	1
0.6486	1
0.3013	1
0.4076	1
0.3697	1
0.4961	1
0.5350	1
0.4177	1
0.3852	1
0.6896	1
0.6381	1
0.4911	1
0.3860	1
0.3329	1
0.4096	1
0.6858	1
0.5072	1
0.4945	1
0.3589	1
0.5635	1
0.2894	1
0.5214	1
0.5088	1
0.4375	1
0.4436	1
0.3078	1
0.4834	1

0.5145	1
0.3950	1
0.3280	1
0.3980	1
0.3661	1
0.3615	1
0.4080	1
0.4841	1
0.5542	1
0.4511	1
0.2911	1
0.5585	1
0.7013	1
0.4029	1
0.8901	1
0.2997	1
0.3016	1
0.2969	1
0.6804	1
0.6555	1
0.5223	1
0.4088	1
0.4172	1
0.4647	1
0.6313	1
0.4264	1
0.3922	1
0.3916	1
0.4920	1
0.3281	1
0.6126	1
0.3563	1
0.7335	1
0.4093	1
0.4866	1
0.6071	1
0.6300	1
0.4670	1
0.3023	1
0.6985	1
0.6421	1
0.3842	1
0.3803	1
0.4196	1
0.4202	1
0.2945	1
0.3909	1
0.4037	1

0.5643	1
0.3974	1
0.4425	1
0.5312	1
0.5851	1
0.4713	1
0.5966	1
0.3238	1
0.5108	1
0.6680	1
0.6939	1
0.3086	1
0.2739	1
0.2492	1
0.4241	1
0.7945	1
0.5274	1
0.5625	1
0.3594	1
0.4342	1
0.6442	1
0.5740	1
0.4952	1
0.5482	1
0.5427	1
0.3741	1
0.3041	1
0.6458	1
0.6105	1
0.3410	1
0.4590	1
0.6400	1
0.4463	1
0.7086	1
0.6493	1
0.4041	1
0.7512	1
0.3966	1
0.4548	1
0.6044	1
0.4819	1
0.3382	1
0.5144	1
0.5513	1
0.7625	1
0.7830	1
0.4616	1
0.4069	1

0.6159	1
0.5224	1
0.3053	1
0.3176	1
0.5782	1
0.3681	1
0.5151	1
0.3820	1
0.5167	1
0.3065	1
0.8452	1
0.3255	1
0.9920	1
0.3254	1
0.2660	1
0.5195	1
0.7935	1
0.3962	1
0.4112	1
0.5291	1
0.4951	1
0.2837	1
0.5441	1
0.9215	1
0.3495	1
0.8882	1
0.5845	1
0.2587	1
0.3794	1
0.5335	1
0.4044	1
0.4055	1
0.5692	1
0.4010	1
0.3399	1
0.6994	1
0.2229	1
0.4639	1
0.5712	1
0.4406	1
0.5276	1
0.3469	1
0.4977	1
0.5265	1
0.5053	1
0.3141	1
0.5370	1
0.5240	1

0.6740	1
0.4150	1
0.7275	1
0.3524	1
0.6142	1
0.4612	1
0.3148	1
0.5968	1
0.7499	1
0.4943	1
0.8520	1
0.6189	1
0.4272	1
0.5014	1
0.4654	1
0.4804	1
0.6154	1
0.5718	1
0.4625	1
0.4302	1
0.4472	1
0.6559	1
0.5358	1
0.4851	1
0.3486	1
0.5383	1
0.4378	1
0.8713	1
0.2993	1
0.4683	1
0.3321	1
0.6048	1
0.3967	1
0.5991	1
0.3352	1
0.3730	1
0.9973	1
0.7104	1
0.6348	1
0.6305	1
0.4045	1
0.9931	1
0.5488	1
0.4432	1
0.6791	1
0.6509	1
0.3467	1
0.4486	1

```
0.3428        1
0.7343        1
0.4326        1
0.8493        1
0.5734        1
0.3539        1
0.5023        1
0.5361        1
0.8300        1
0.6028        1
0.6645        1
0.4939        1
0.2624        1
0.4180        1
0.3623        1
0.4801        1
0.5693        1
0.4840        1
0.6800        1
0.8357        1
0.3621        1
0.3550        1
0.8712        1
0.9712        1
0.5925        1
0.6967        1
0.4124        1
0.4519        1
0.6006        1
0.5418        1
0.3082        1
0.6424        1
0.4538        1
0.4242        1
0.5391        1
0.6281        1
0.5044        1
0.8569        1
0.8362        1
0.5119        1
0.4823        1
0.3659        1
0.3500        1
Name: TOTALAREA_MODE, Length: 5116, dtype: int64
```

In [93]:
```
application_bureau_loan_train_data_log['FONDKAPREMONT_MODE'].value_counts()
```

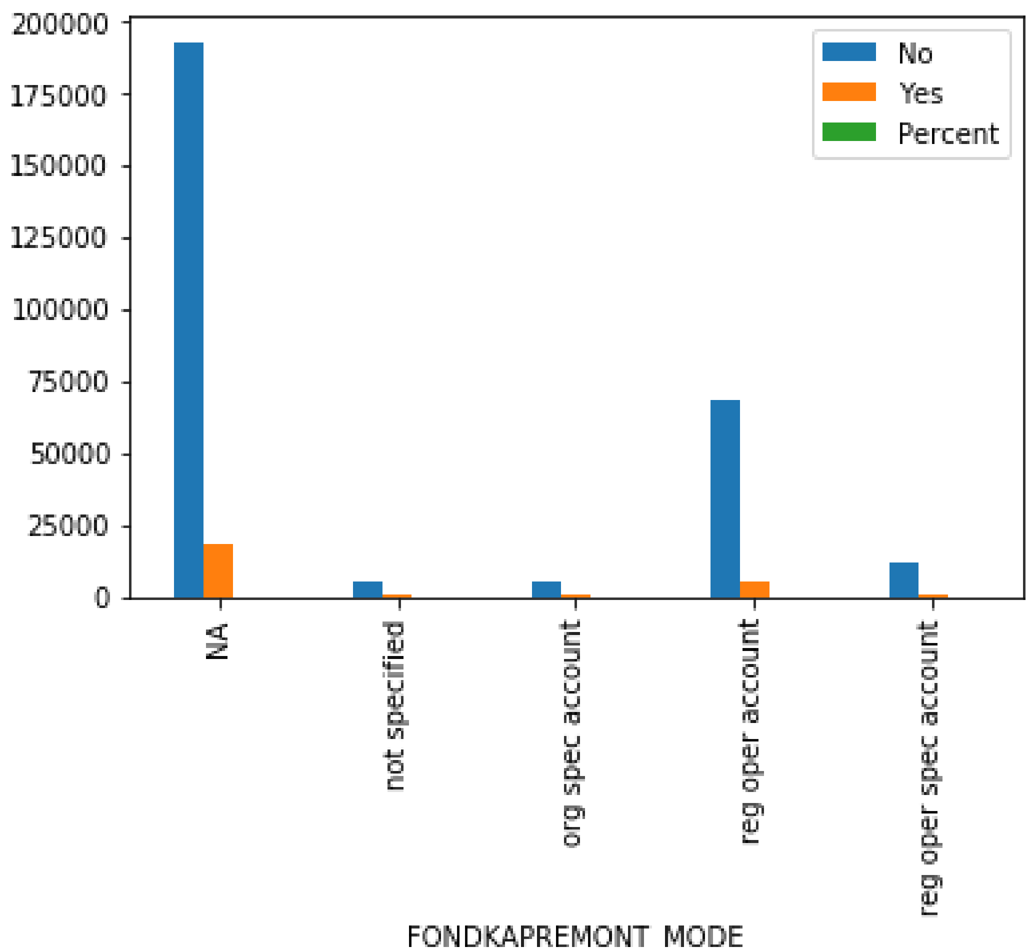
Out[93]:
```
NA                        210295
reg oper account           73830
```

```
reg oper spec account     12080
not specified              5687
org spec account           5619
Name: FONDKAPREMONT_MODE, dtype: int64
```

In [279]:
```
tab = pd.crosstab(index=application_bureau_loan_train_data_log['FONDKAPREMONT_MODE']
tab.columns = ['No','Yes']
tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
print(tab)
tab.plot(kind='bar')
```

```
                           No    Yes   Percent
FONDKAPREMONT_MODE
NA                     192170  18125  8.618845
not specified            5258    429  7.543520
org spec account         5292    327  5.819541
reg oper account        68678   5152  6.978193
reg oper spec account   11288    792  6.556291
```

Out[279]: <matplotlib.axes._subplots.AxesSubplot at 0x14507cda898>



In [280]: application_bureau_loan_train_data_log['WALLSMATERIAL_MODE'].value_counts()

Out[280]:
```
NA                156341
Panel              66040
Stone, brick       64815
Block               9253
Wooden              5362
Mixed               2296
Monolithic          1779
Others              1625
Name: WALLSMATERIAL_MODE, dtype: int64
```

In [281]:
```
tab = pd.crosstab(index=application_bureau_loan_train_data_log['WALLSMATERIAL_MODE']
tab.columns = ['No','Yes']
tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
tab
```

Out[281]:

WALLSMATERIAL_MODE	No	Yes	Percent
Block	8603	650	7.024749
Mixed	2123	173	7.534843
Monolithic	1695	84	4.721754
NA	142070	14271	9.128124
Others	1490	135	8.307692
Panel	61848	4192	6.347668
Stone, brick	60015	4800	7.405693
Wooden	4842	520	9.697874

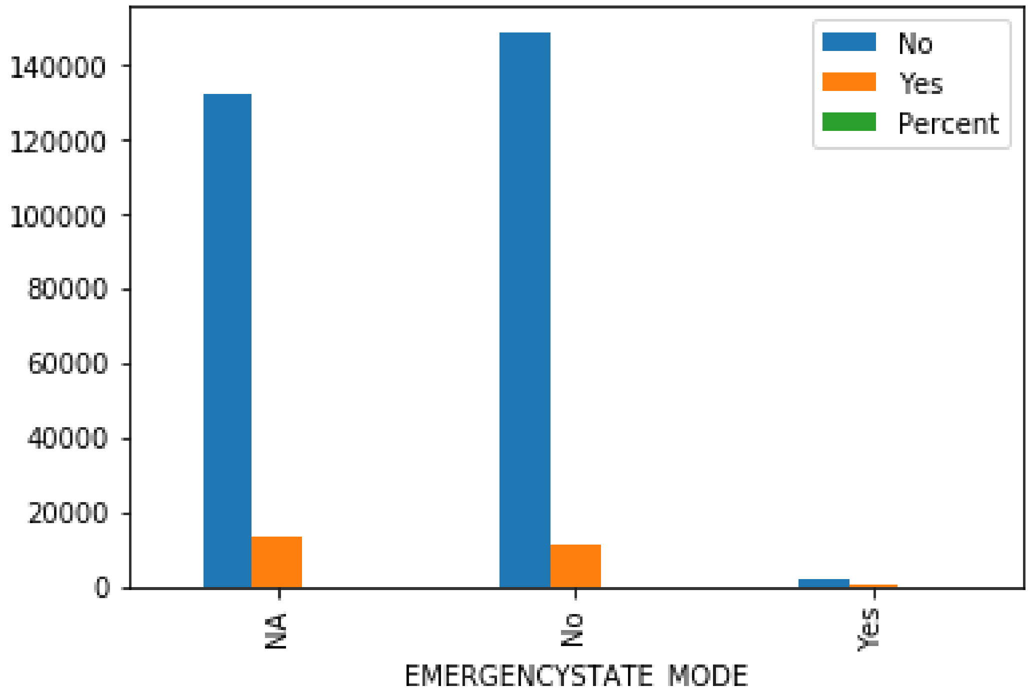
In [97]: application_bureau_loan_train_data_log['EMERGENCYSTATE_MODE'].value_counts()

Out[97]:
```
No     159428
NA     145755
Yes      2328
Name: EMERGENCYSTATE_MODE, dtype: int64
```

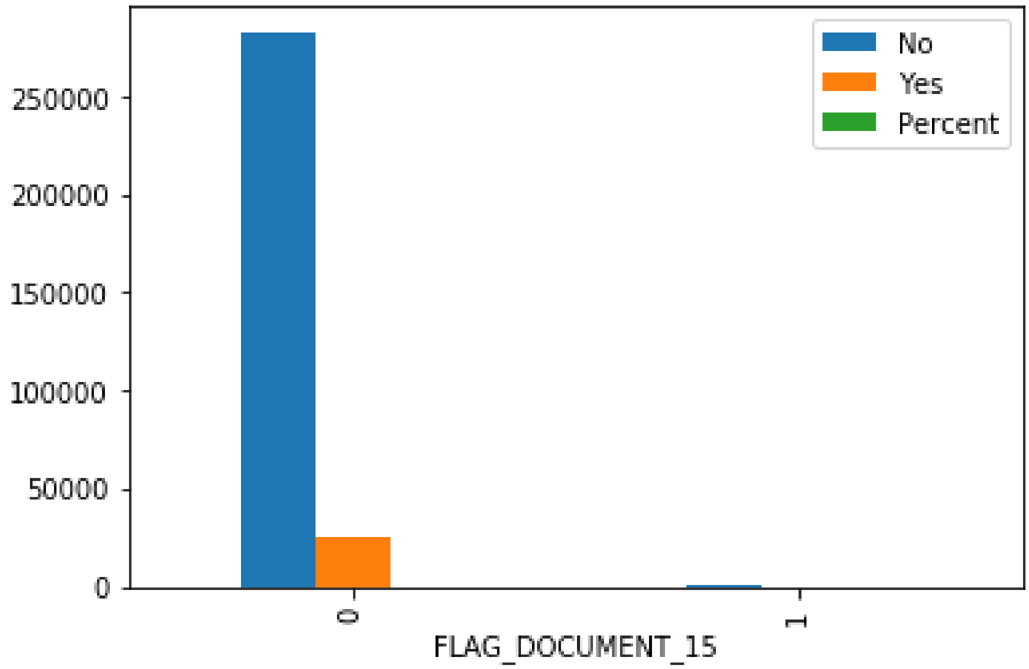
In [98]:
```
tab = pd.crosstab(index=application_bureau_loan_train_data_log['EMERGENCYSTATE_MODE']
tab.columns = ['No','Yes']
tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
print(tab)
tab.plot(kind='bar')
```

```
                         No    Yes   Percent
EMERGENCYSTATE_MODE
NA                   132257  13498  9.260746
No                   148324  11104  6.964900
Yes                    2105    223  9.579038
```

```
Out[98]: <matplotlib.axes._subplots.AxesSubplot at 0x14518cd8e80>
```




```
In [99]: application_bureau_loan_train_data_log['FLAG_DOCUMENT_15'].value_counts()

Out[99]: 0    307139
         1       372
         Name: FLAG_DOCUMENT_15, dtype: int64

In [100]: tab = pd.crosstab(index=application_bureau_loan_train_data_log['FLAG_DOCUMENT_15'],co
          tab.columns = ['No','Yes']
          tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
          print(tab)
          tab.plot(kind='bar')

                      No    Yes   Percent
FLAG_DOCUMENT_15
0                 282325  24814  8.079078
1                    361     11  2.956989


Out[100]: <matplotlib.axes._subplots.AxesSubplot at 0x1451b7e5358>
```



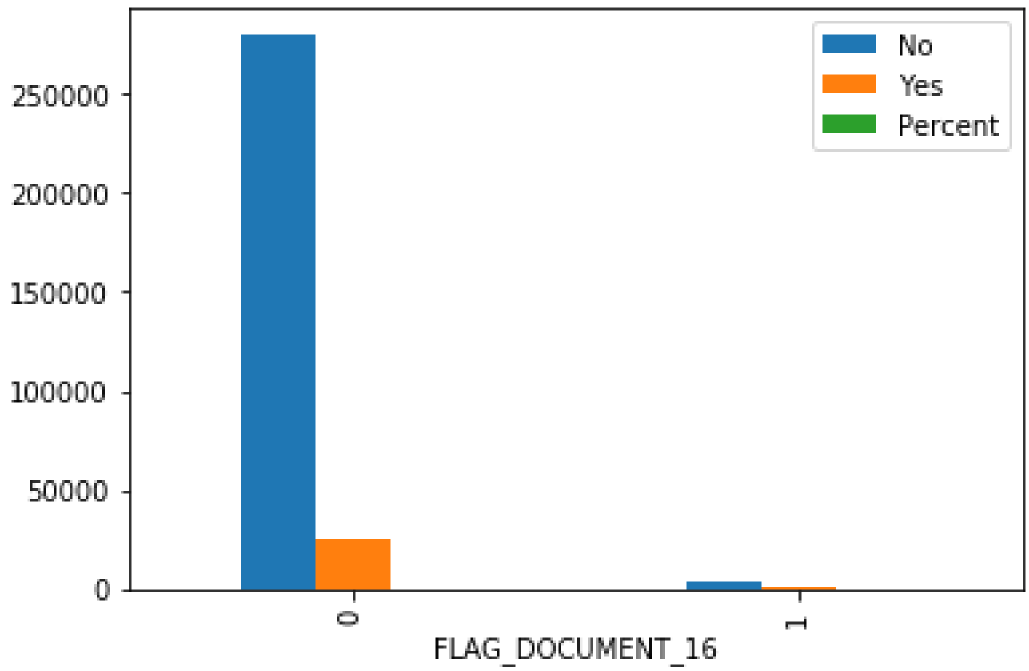

```
In [101]: application_bureau_loan_train_data_log['FLAG_DOCUMENT_16'].value_counts()
```

```
Out[101]: 0    304458
          1      3053
          Name: FLAG_DOCUMENT_16, dtype: int64
```

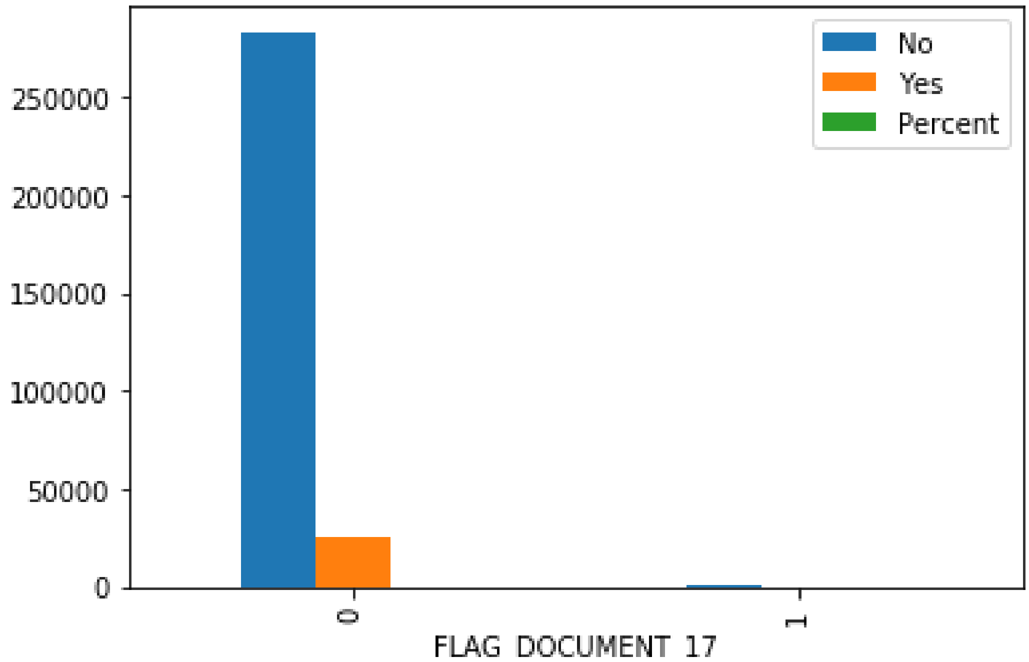
```
In [102]: tab = pd.crosstab(index=application_bureau_loan_train_data_log['FLAG_DOCUMENT_16'],co
          tab.columns = ['No','Yes']
          tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
          print(tab)
          tab.plot(kind='bar')
```

```
                      No    Yes   Percent
FLAG_DOCUMENT_16
0                 279783  24675  8.104566
1                   2903    150  4.913200
```

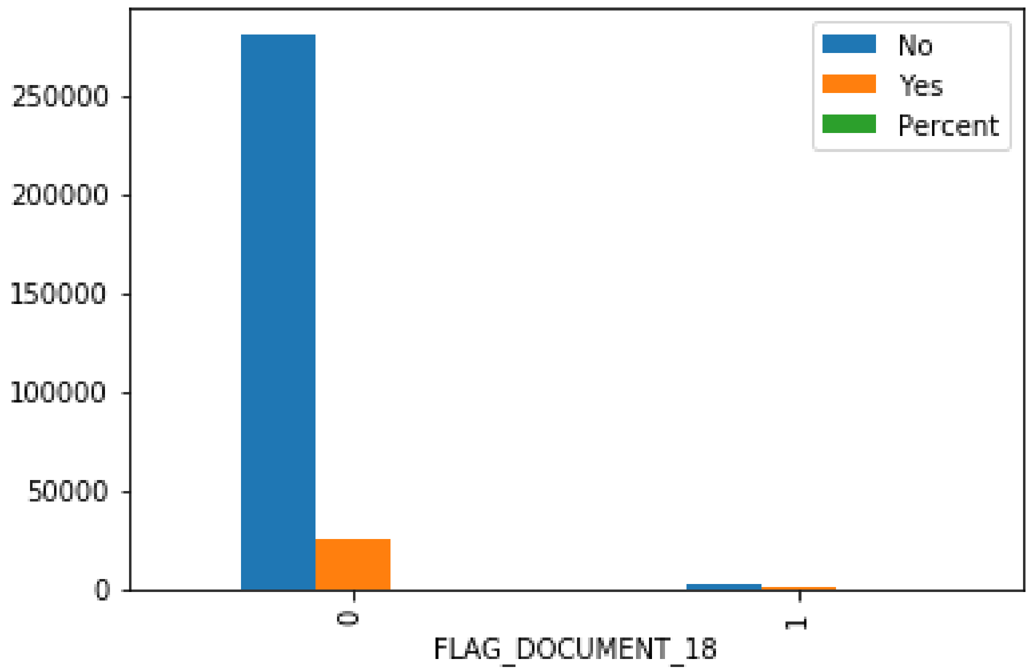
```
Out[102]: <matplotlib.axes._subplots.AxesSubplot at 0x1451b83a400>
```




```
In [103]: application_bureau_loan_train_data_log['FLAG_DOCUMENT_17'].value_counts()

Out[103]: 0    307429
          1        82
          Name: FLAG_DOCUMENT_17, dtype: int64

In [104]: tab = pd.crosstab(index=application_bureau_loan_train_data_log['FLAG_DOCUMENT_17'],co
          tab.columns = ['No','Yes']
          tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
          print(tab)
          tab.plot(kind='bar')

                      No    Yes   Percent
FLAG_DOCUMENT_17
0                 282606  24823  8.074385
1                     80      2  2.439024


Out[104]: <matplotlib.axes._subplots.AxesSubplot at 0x1451b8bfe10>
```



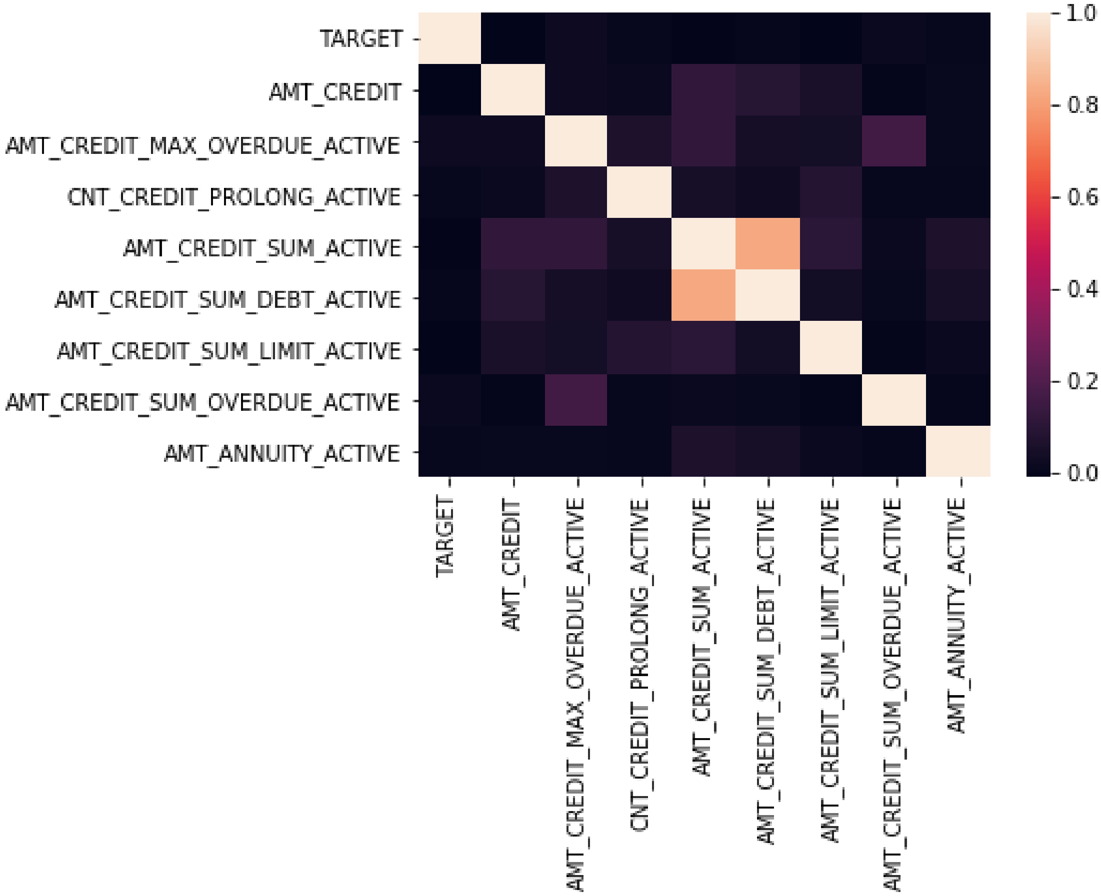

```
In [105]: application_bureau_loan_train_data_log['FLAG_DOCUMENT_18'].value_counts()

Out[105]: 0    305011
          1      2500
          Name: FLAG_DOCUMENT_18, dtype: int64

In [106]: tab = pd.crosstab(index=application_bureau_loan_train_data_log['FLAG_DOCUMENT_18'],co
          tab.columns = ['No','Yes']
          tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
          print(tab)
          tab.plot(kind='bar')

                      No    Yes   Percent
FLAG_DOCUMENT_18
0                 280328  24683  8.092495
1                   2358    142  5.680000


Out[106]: <matplotlib.axes._subplots.AxesSubplot at 0x1451b867dd8>
```




```
In [107]: cor = application_bureau_loan_train_data_log[['TARGET','AMT_CREDIT', 'AMT_CREDIT_MAX_
          'AMT_CREDIT_SUM_LIMIT_ACTIVE','AMT_CREDIT_SUM_OVERDUE_ACTIVE','AMT_ANNUITY_ACTIVE']]
          print( "Correlation coefficients are:")
          print(str(cor))

          sns.heatmap(cor)
```

```
Correlation coefficients are:
                                 TARGET  AMT_CREDIT  AMT_CREDIT_MAX_OVERDUE_ACTIVE  CNT_CREDIT_
TARGET                         1.000000   -0.012181                       0.020019
AMT_CREDIT                    -0.012181    1.000000                       0.021127
AMT_CREDIT_MAX_OVERDUE_ACTIVE  0.020019    0.021127                       1.000000
CNT_CREDIT_PROLONG_ACTIVE      0.006638    0.014217                       0.065078
AMT_CREDIT_SUM_ACTIVE         -0.005999    0.120972                       0.120834
AMT_CREDIT_SUM_DEBT_ACTIVE     0.001960    0.090657                       0.043659
AMT_CREDIT_SUM_LIMIT_ACTIVE   -0.011996    0.056987                       0.039532
AMT_CREDIT_SUM_OVERDUE_ACTIVE  0.012892   -0.001178                       0.163072
AMT_ANNUITY_ACTIVE             0.006538    0.009148                       0.011538
```

```
Out[107]: <matplotlib.axes._subplots.AxesSubplot at 0x1451b96fa58>
```



1.19 Interpretation:

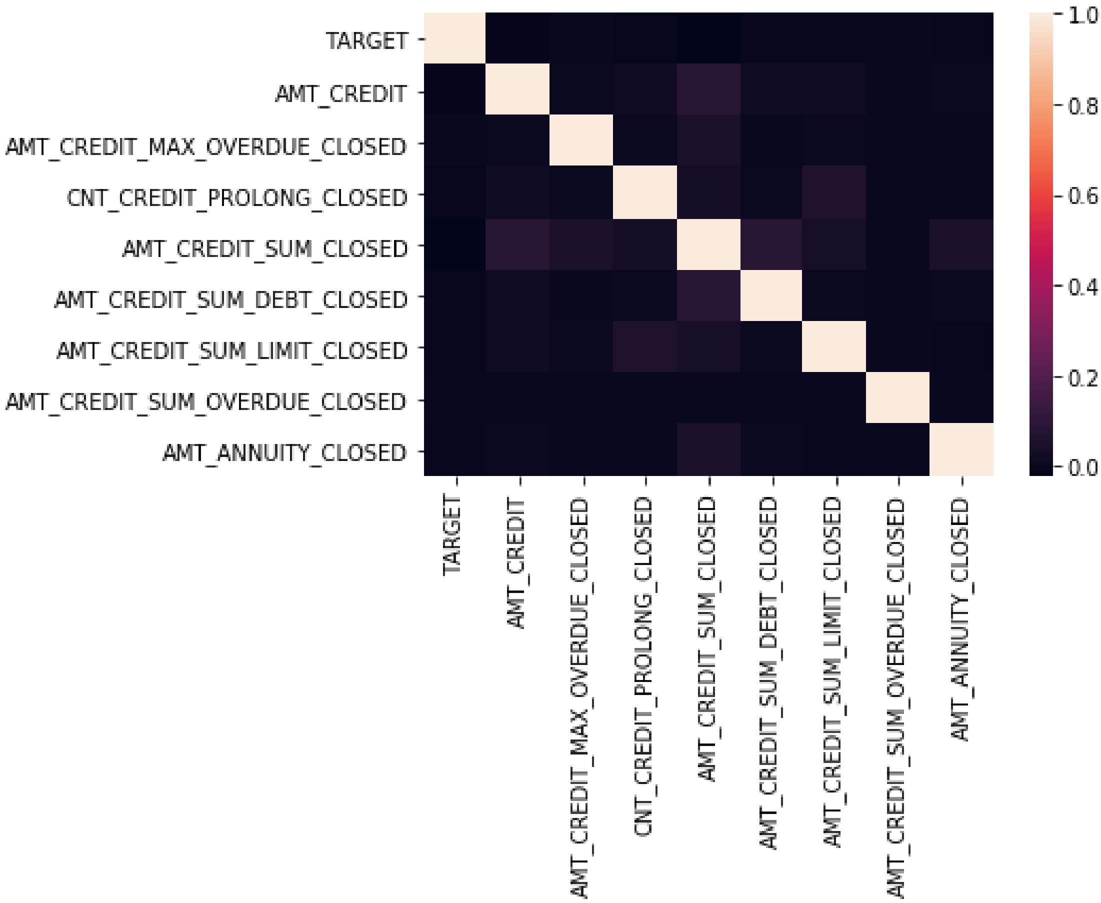
Only AMT_CREDIT_SUM_ACTIVE has very strong linear relationship with AMT_CREDIT_SUM_DEBT_ACTIVE. Other than that there is no almost no linear relationship between other varialbes

```
In [108]: cor = application_bureau_loan_train_data_log[['TARGET','AMT_CREDIT', 'AMT_CREDIT_MAX_
          'AMT_CREDIT_SUM_LIMIT_CLOSED','AMT_CREDIT_SUM_OVERDUE_CLOSED','AMT_ANNUITY_CLOSED']]
          print( "Correlation coefficients are:")
          print(str(cor))

          sns.heatmap(cor)
```

```
Correlation coefficients are:
                                TARGET  AMT_CREDIT  AMT_CREDIT_MAX_OVERDUE_CLOSED  CNT_CREDIT_
TARGET                        1.000000   -0.012181                       0.000128
AMT_CREDIT                   -0.012181    1.000000                       0.006428
AMT_CREDIT_MAX_OVERDUE_CLOSED  0.000128    0.006428                       1.000000
CNT_CREDIT_PROLONG_CLOSED    -0.006475    0.018753                       0.006670
```

```
AMT_CREDIT_SUM_CLOSED          -0.020238    0.085200                        0.053115
AMT_CREDIT_SUM_DEBT_CLOSED     -0.001044    0.019467                        0.000094
AMT_CREDIT_SUM_LIMIT_CLOSED    -0.001038    0.016404                        0.004159
AMT_CREDIT_SUM_OVERDUE_CLOSED -0.000235     0.000228                        0.000100
AMT_ANNUITY_CLOSED             -0.002010    0.006602                        0.001593
```

```
Out[108]: <matplotlib.axes._subplots.AxesSubplot at 0x1451d42b828>
```



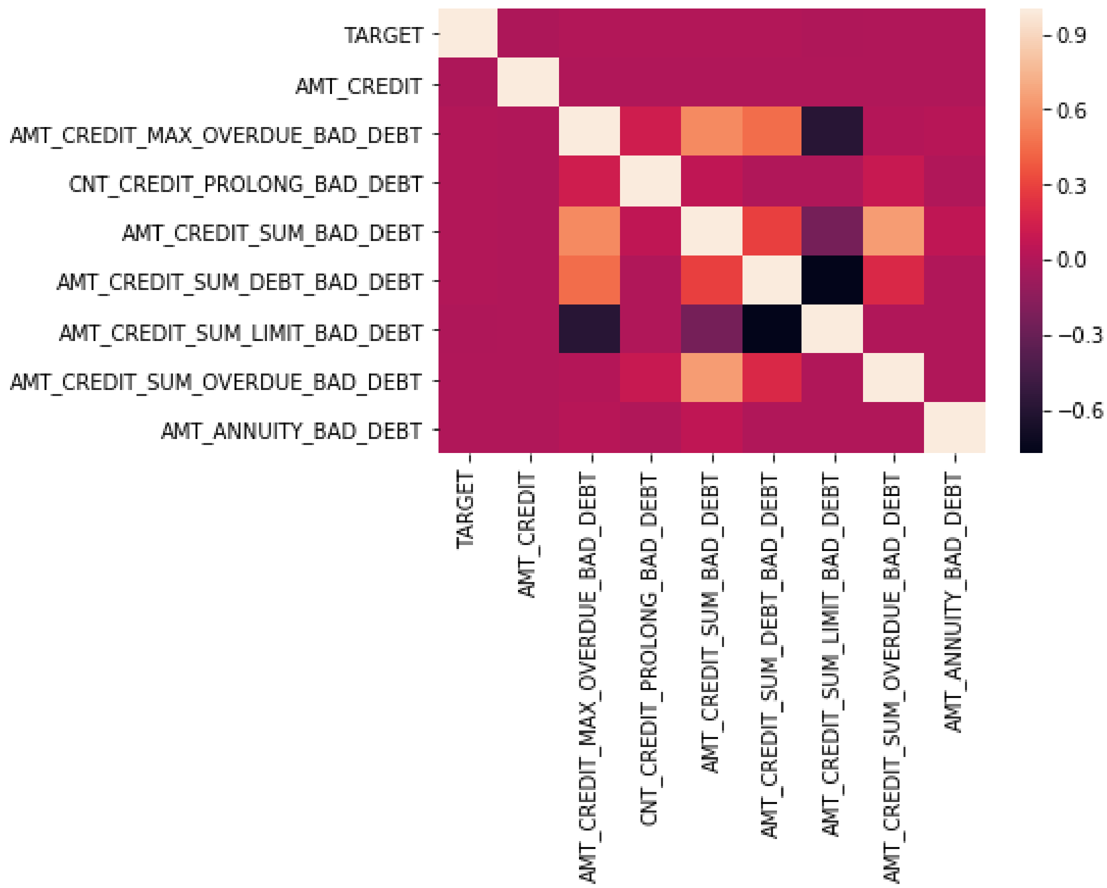
1.20 Interpretation:

There is almost no relationship between any pair of columns

```
In [109]: cor = application_bureau_loan_train_data_log[['TARGET','AMT_CREDIT', 'AMT_CREDIT_MAX_
          'AMT_CREDIT_SUM_LIMIT_BAD_DEBT','AMT_CREDIT_SUM_OVERDUE_BAD_DEBT','AMT_ANNUITY_BAD_D
          print( "Correlation coefficients are:")
          print(str(cor))

          sns.heatmap(cor)
```

```
Correlation coefficients are:
                                 TARGET  AMT_CREDIT  AMT_CREDIT_MAX_OVERDUE_BAD_DEBT  CNT_CRE
TARGET                         1.000000   -0.012181                         0.006409
AMT_CREDIT                    -0.012181    1.000000                        -0.003127
AMT_CREDIT_MAX_OVERDUE_BAD_DEBT  0.006409   -0.003127                         1.000000
CNT_CREDIT_PROLONG_BAD_DEBT    0.006085   -0.001882                         0.132931
AMT_CREDIT_SUM_BAD_DEBT        0.004161   -0.002201                         0.567926
AMT_CREDIT_SUM_DEBT_BAD_DEBT   0.003411   -0.002486                         0.453370
AMT_CREDIT_SUM_LIMIT_BAD_DEBT -0.005121    0.001877                        -0.583099
AMT_CREDIT_SUM_OVERDUE_BAD_DEBT -0.000278    0.000185                         0.012558
AMT_ANNUITY_BAD_DEBT          -0.000534    0.000326                         0.026141
```

```
Out[109]: <matplotlib.axes._subplots.AxesSubplot at 0x1451d4e0940>
```



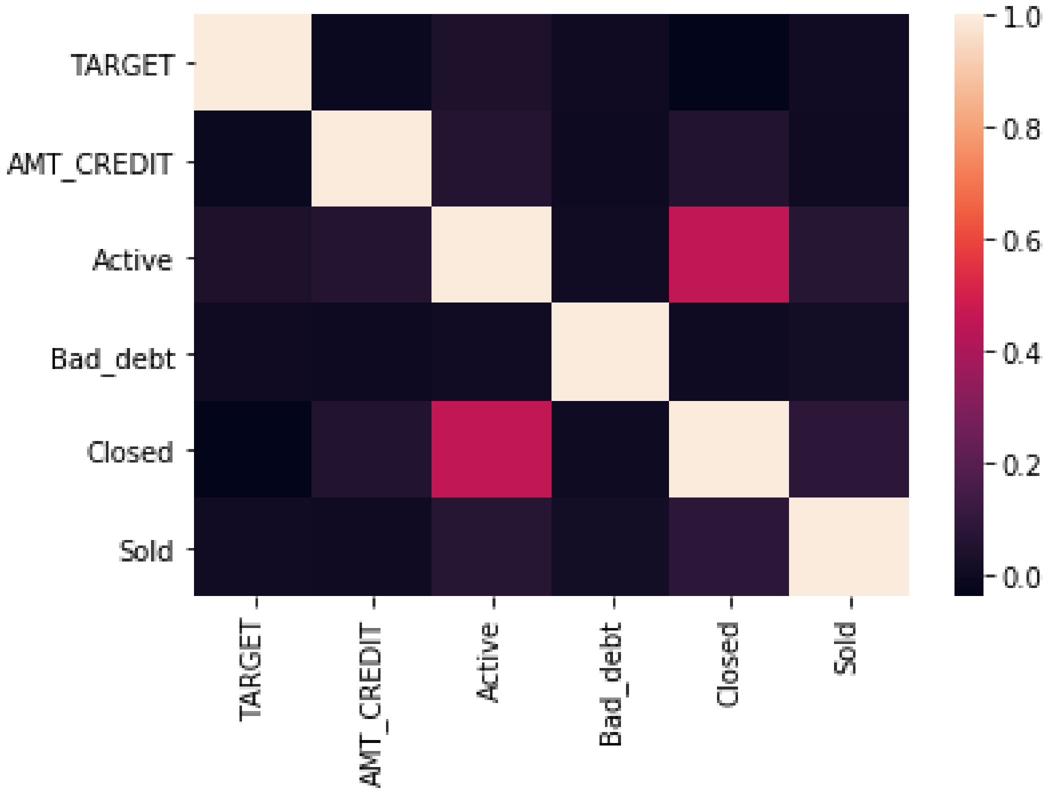
1.21 Interpretation:

AMT_CREDIT_SUM_LIMIT_BAD_DEBT has strong relationship with AMT_CREDIT_SUM_BAD_DEBT

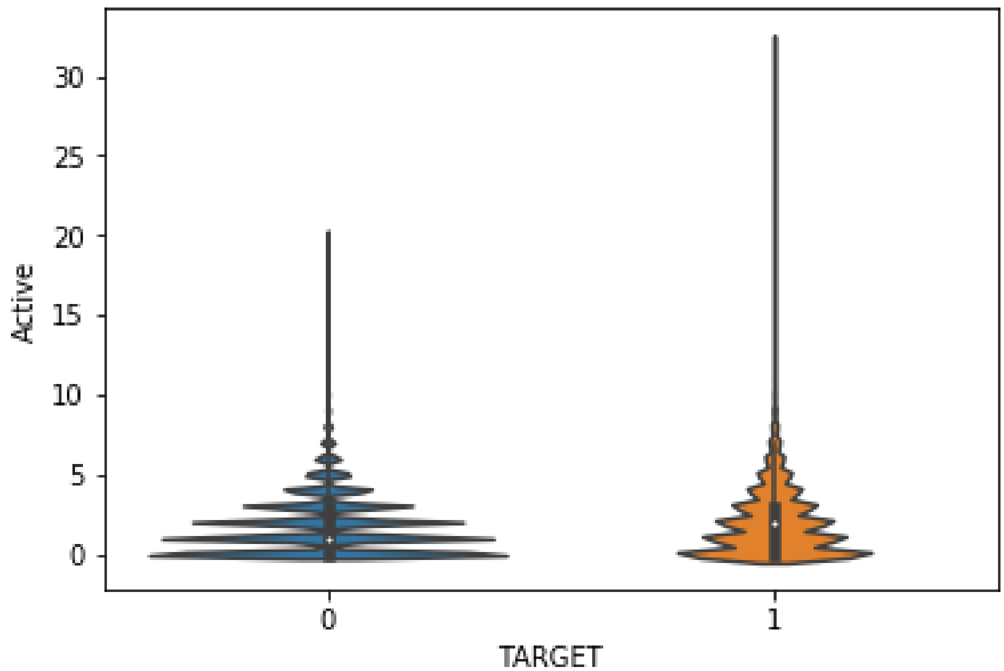
```
In [110]: cor = application_bureau_loan_train_data_log[['TARGET','AMT_CREDIT','Active', 'Bad_de
          print( "Correlation coefficients are:")
          print(str(cor))

          sns.heatmap(cor)
```

```
Correlation coefficients are:
              TARGET  AMT_CREDIT    Active  Bad_debt    Closed      Sold
TARGET      1.000000   -0.012181  0.043569  0.003531 -0.037233  0.009347
AMT_CREDIT -0.012181    1.000000  0.061461 -0.003743  0.056674  0.007242
Active      0.043569    0.061461  1.000000  0.008212  0.455955  0.070171
Bad_debt    0.003531   -0.003743  0.008212  1.000000  0.002678  0.012759
Closed     -0.037233    0.056674  0.455955  0.002678  1.000000  0.084678
Sold        0.009347    0.007242  0.070171  0.012759  0.084678  1.000000
```

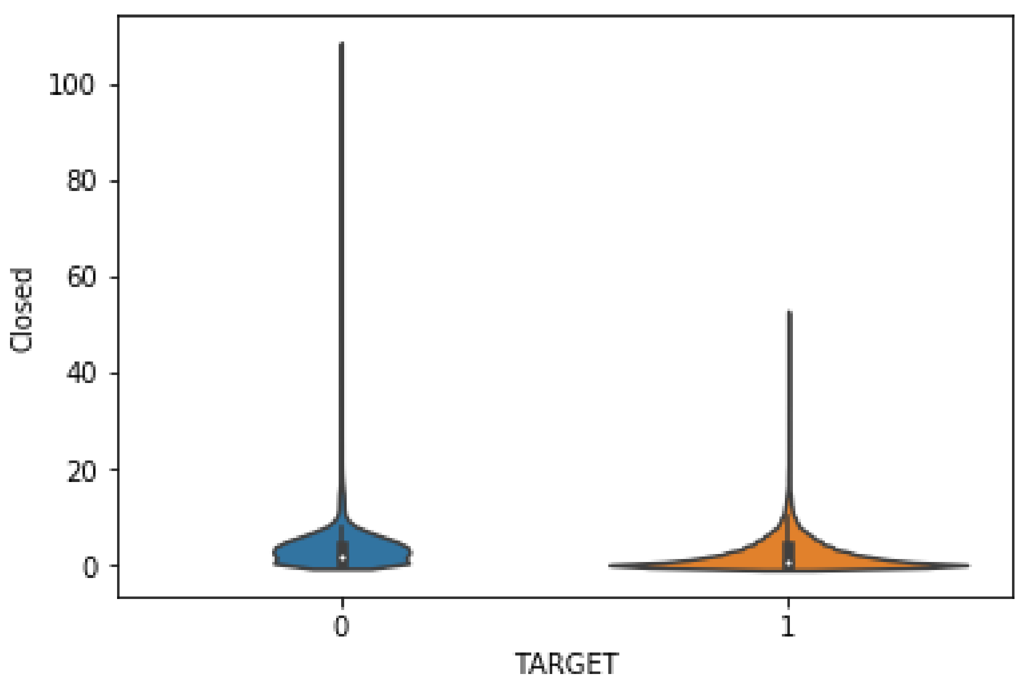
```
Out[110]: <matplotlib.axes._subplots.AxesSubplot at 0x1451d595400>
```



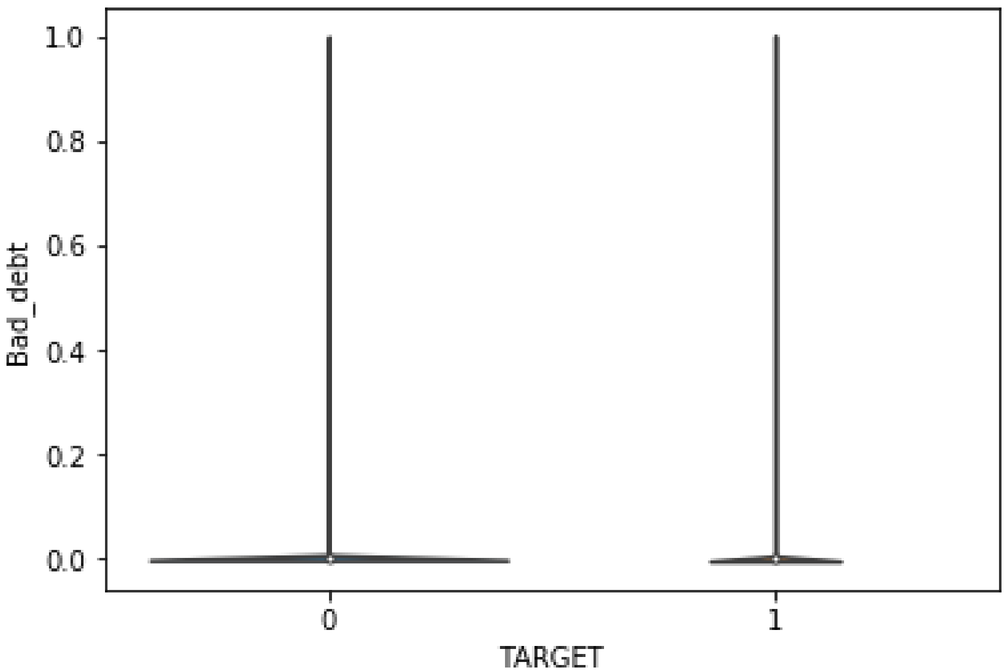

```
In [111]: sns.violinplot(x='TARGET',y='Active',data=application_bureau_loan_train_data_log)
          plt.show()
```



In [112]:
```
sns.violinplot(x='TARGET',y='Closed',data=application_bureau_loan_train_data_log)
plt.show()
```



In [113]:
```
sns.violinplot(x='TARGET',y='Bad_debt',data=application_bureau_loan_train_data_log)
plt.show()
```



In [114]:
```
application_bureau_loan_train_data_log.groupby('TARGET').mean()
```

Out[114]:

TARGET	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_(
0	278244.744536	0.412946	11.911923	13.07269	10.067121	
1	277449.167936	0.463807	11.878753	13.04071	10.069109	

TARGET	DAYS_LAST_PHONE_CHANGE	FLAG_DOCUMENT_2	FLAG_DOCUMENT_3	FLAG_DOCUMENT_4	FI
0	-976.384840	0.000032	0.704060	0.000088	
1	-808.796818	0.000161	0.777925	0.000000	

TARGET	YEARS_ENDDATE_FACT_BAD_DEBT	AMT_CREDIT_MAX_OVERDUE_BAD_DEBT	CNT_CREDIT_PROI
0	-0.000103	2.088477	
1	-0.000071	19.172637	

In [115]:
```
application_bureau_loan_train_data_log.groupby('TARGET').median()
```

Out[115]:

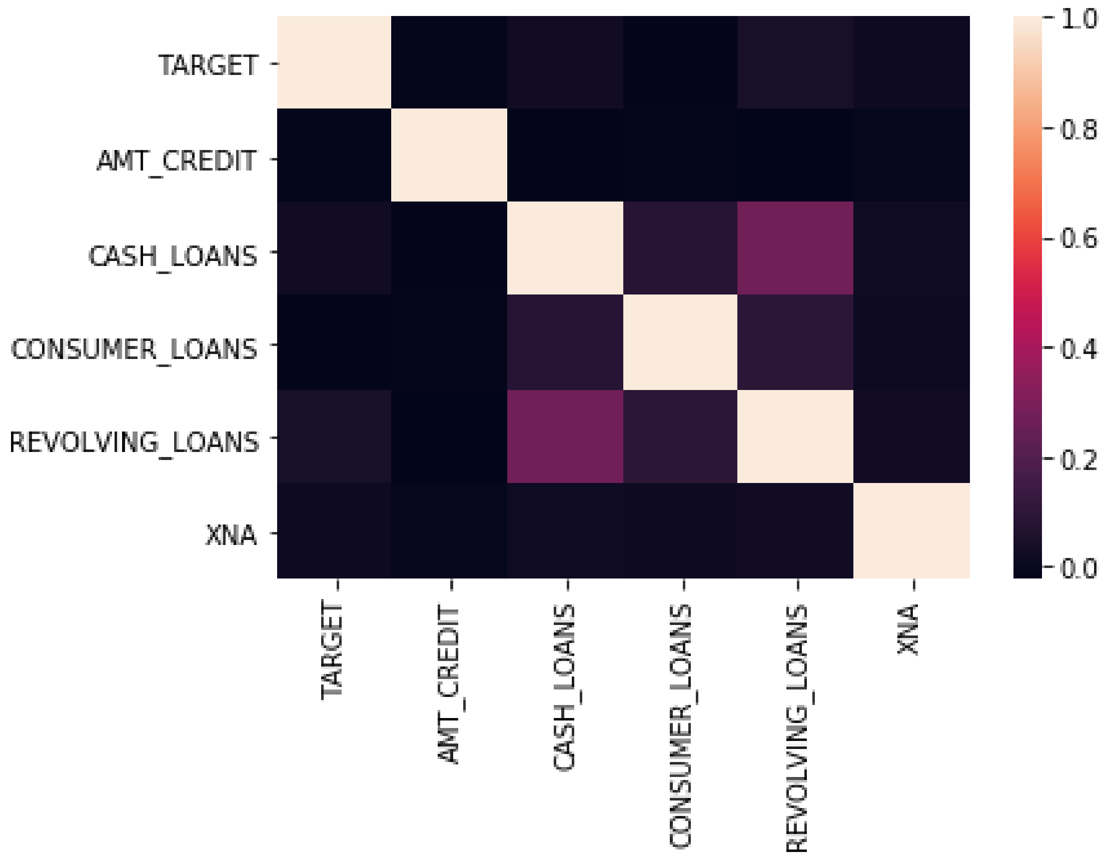
TARGET	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOOI
0	278362.5	0.0	11.908347	13.157323	10.121699	13

```
1         276291.0           0.0         11.813037   13.117393    10.137136        1

        DAYS_LAST_PHONE_CHANGE  FLAG_DOCUMENT_2  FLAG_DOCUMENT_3  FLAG_DOCUMENT_4  F
TARGET
0                       -776.0              0.0              1.0              0.0
1                       -594.0              0.0              1.0              0.0

        YEARS_ENDDATE_FACT_BAD_DEBT  AMT_CREDIT_MAX_OVERDUE_BAD_DEBT  CNT_CREDIT_PRO
TARGET
0                               0.0                              0.0
1                               0.0                              0.0
```

In [116]:
```
cor = application_bureau_loan_train_data_log[['TARGET','AMT_CREDIT','CASH_LOANS', 'C
print( "Correlation coefficients are:")
print(str(cor))

sns.heatmap(cor)
```

```
Correlation coefficients are:
                   TARGET  AMT_CREDIT  CASH_LOANS  CONSUMER_LOANS  REVOLVING_LOANS       XNA
TARGET           1.000000   -0.012181    0.024765       -0.014818         0.046637  0.012869
AMT_CREDIT      -0.012181    1.000000   -0.012592       -0.011646        -0.020183 -0.007261
CASH_LOANS       0.024765   -0.012592    1.000000        0.081205         0.273391  0.019134
CONSUMER_LOANS  -0.014818   -0.011646    0.081205        1.000000         0.099303  0.011974
REVOLVING_LOANS  0.046637   -0.020183    0.273391        0.099303         1.000000  0.024859
XNA              0.012869   -0.007261    0.019134        0.011974         0.024859  1.000000
```

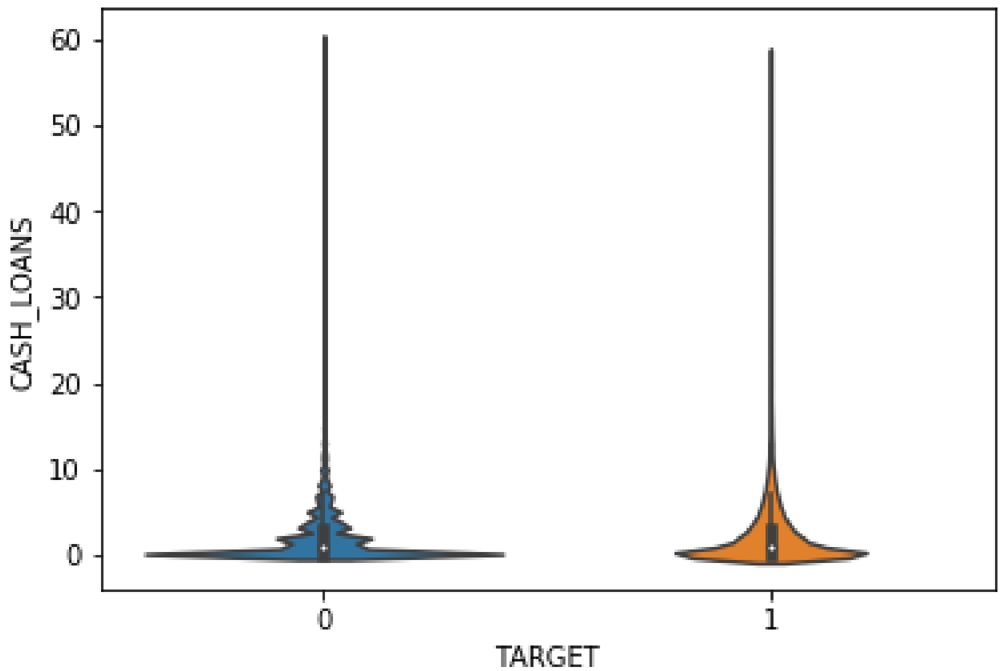
Out[116]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14522529e48>
```



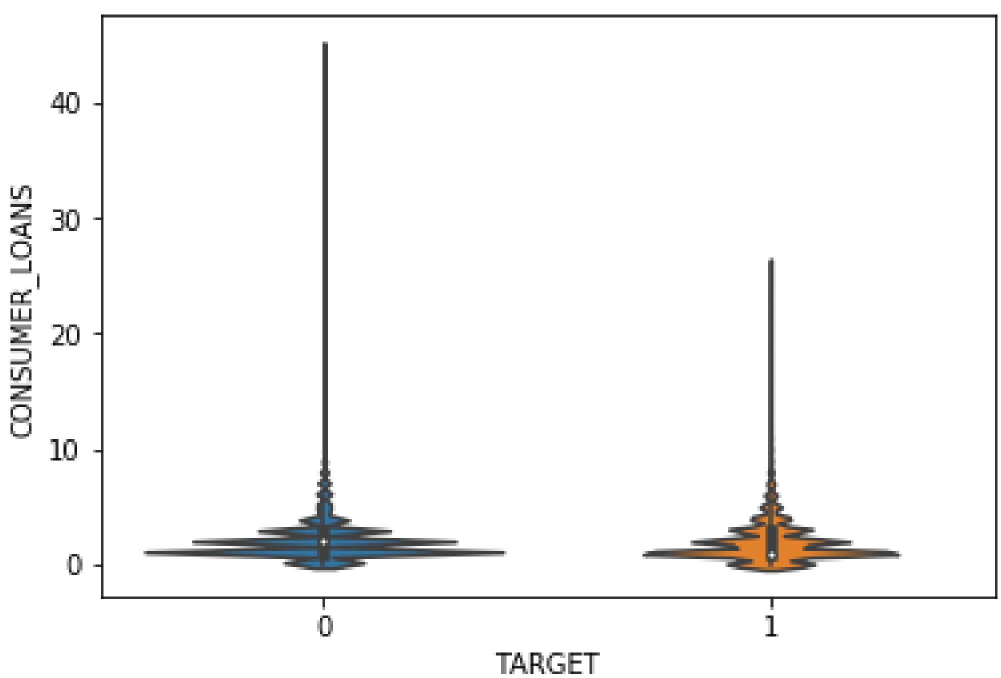
1.22 Interpretation:

CASH_LOANS has linear relationship with REVOLVING_LOANS

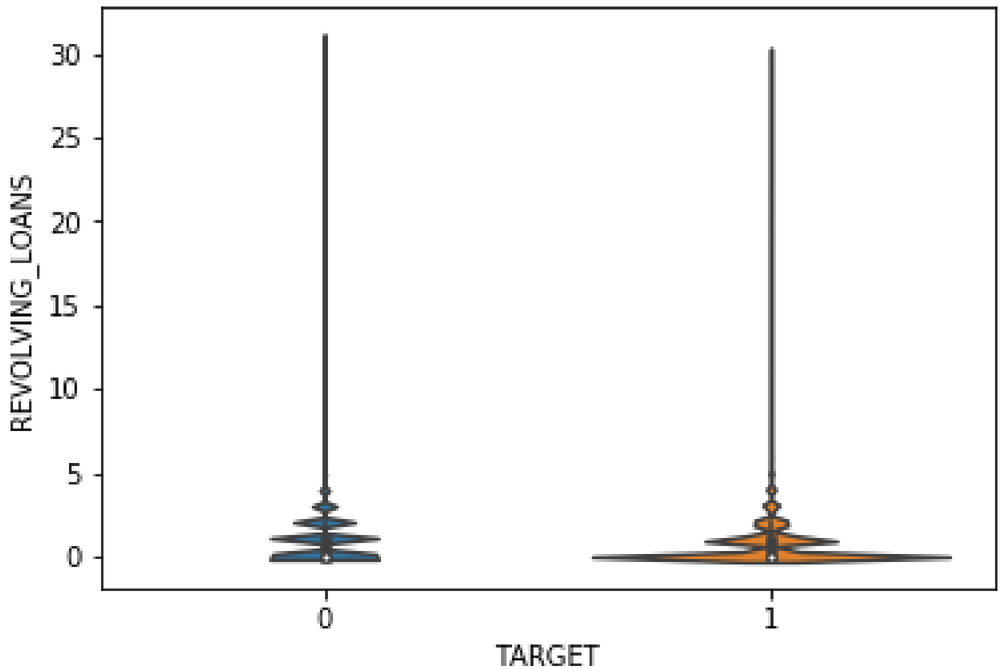
```
In [117]: sns.violinplot(x='TARGET',y='CASH_LOANS',data=application_bureau_loan_train_data_log)
          plt.show()
```



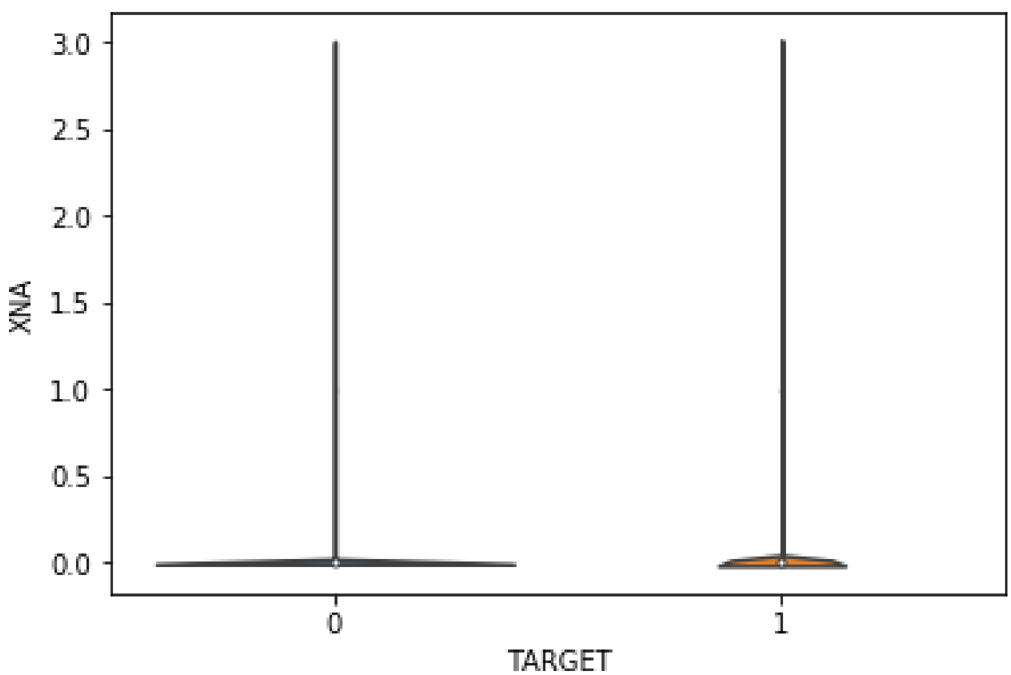

```
In [118]: sns.violinplot(x='TARGET',y='CONSUMER_LOANS',data=application_bureau_loan_train_data_
          plt.show()
```



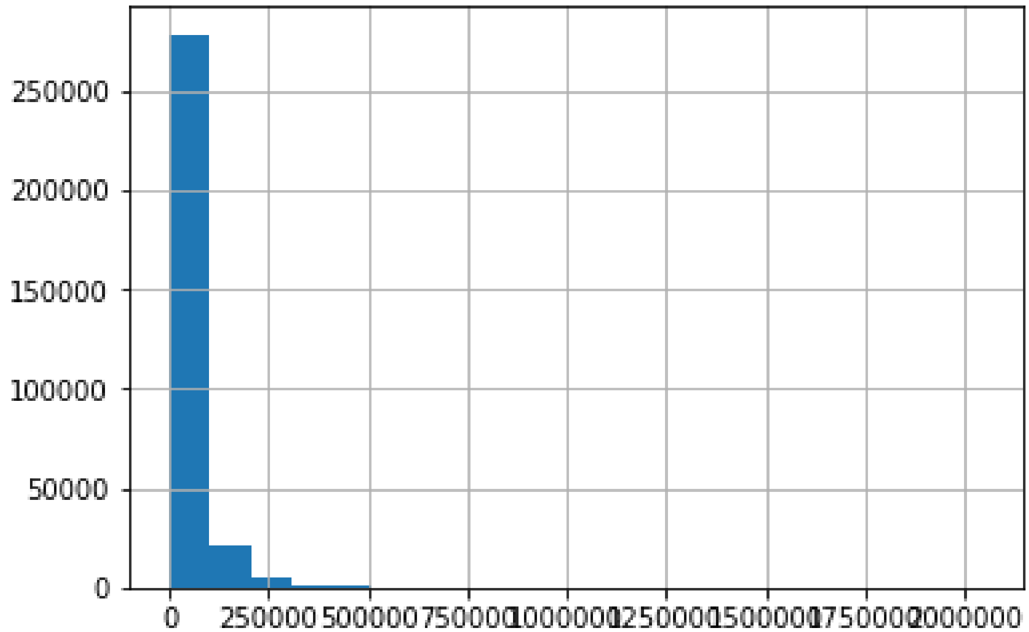
```
In [119]: sns.violinplot(x='TARGET',y='REVOLVING_LOANS',data=application_bureau_loan_train_data
          plt.show()
```



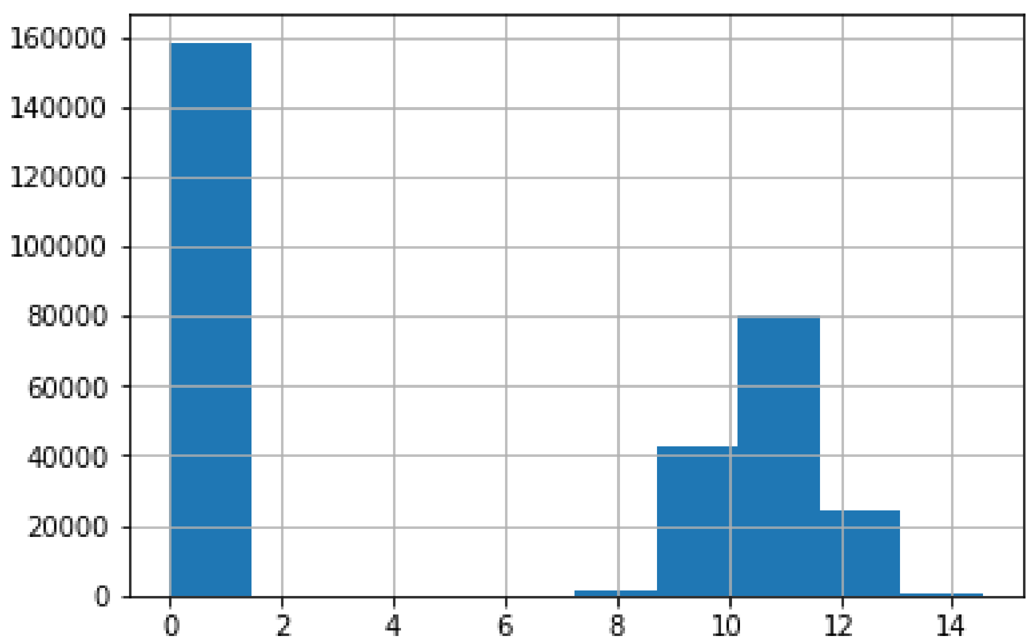

```
In [120]: sns.violinplot(x='TARGET',y='XNA',data=application_bureau_loan_train_data_log)
          plt.show()
```



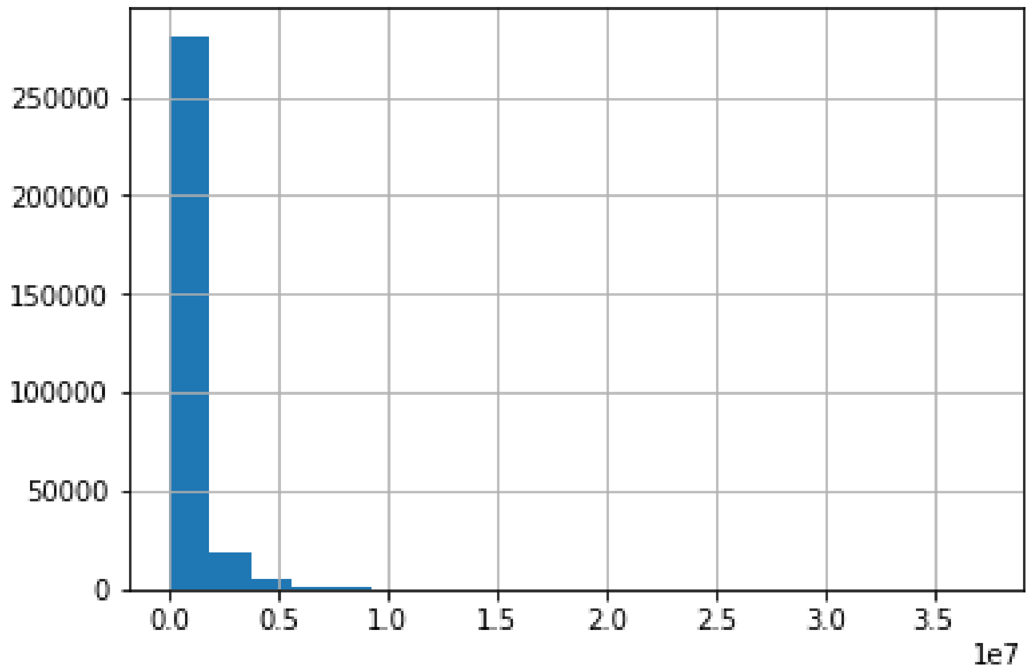
```
In [121]: application_bureau_loan_train_data['PREV_CASH_AMT_ANNUITY'].hist(bins=20)
          plt.show()
```



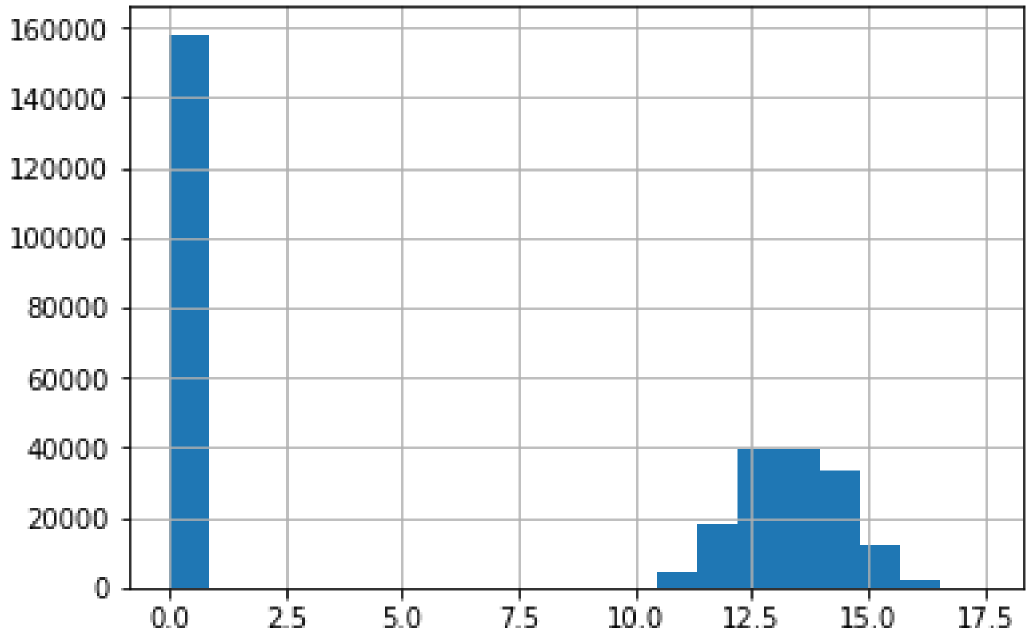

```
In [122]: np.log(application_bureau_loan_train_data['PREV_CASH_AMT_ANNUITY'] + 1).hist()
          plt.show()
```



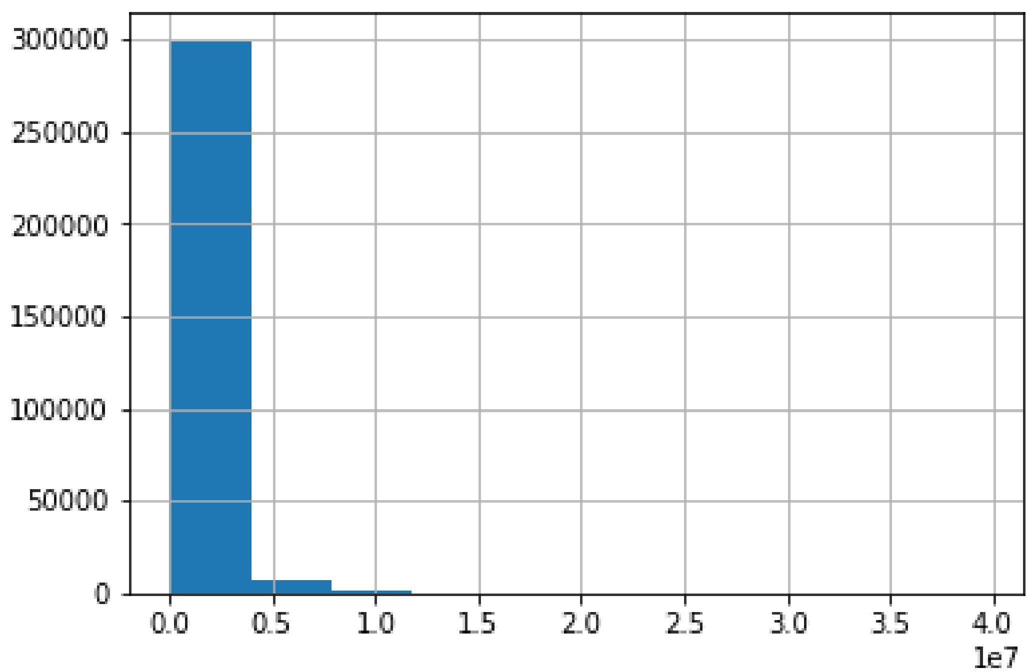
```
In [123]: application_bureau_loan_train_data['PREV_CASH_AMT_APPLICATION'].hist(bins=20)
          plt.show()
```



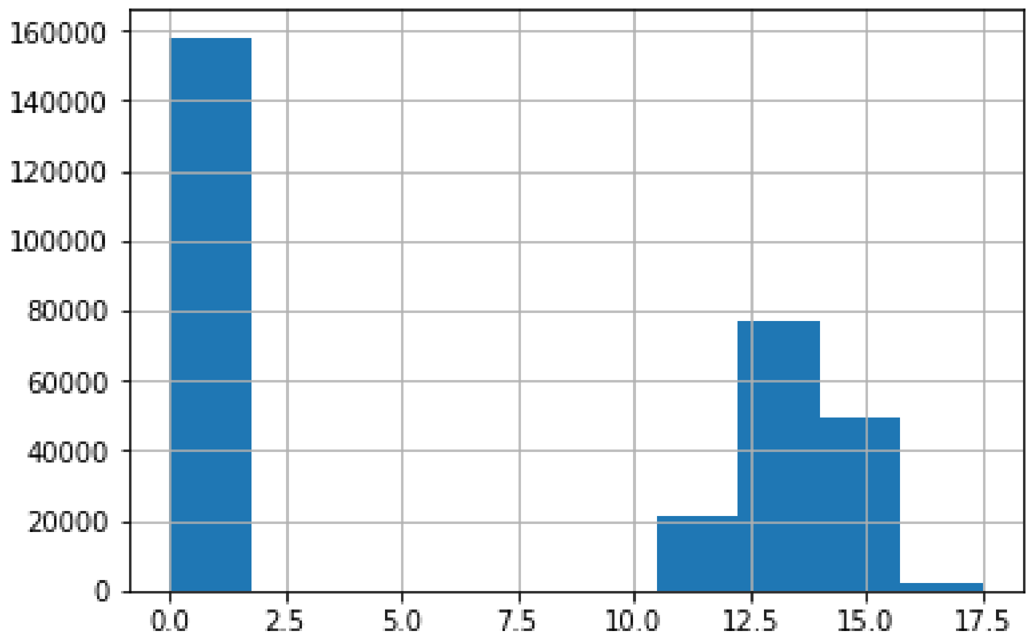

```
In [124]: np.log(application_bureau_loan_train_data['PREV_CASH_AMT_APPLICATION'] + 1).hist(bins
          plt.show()
```



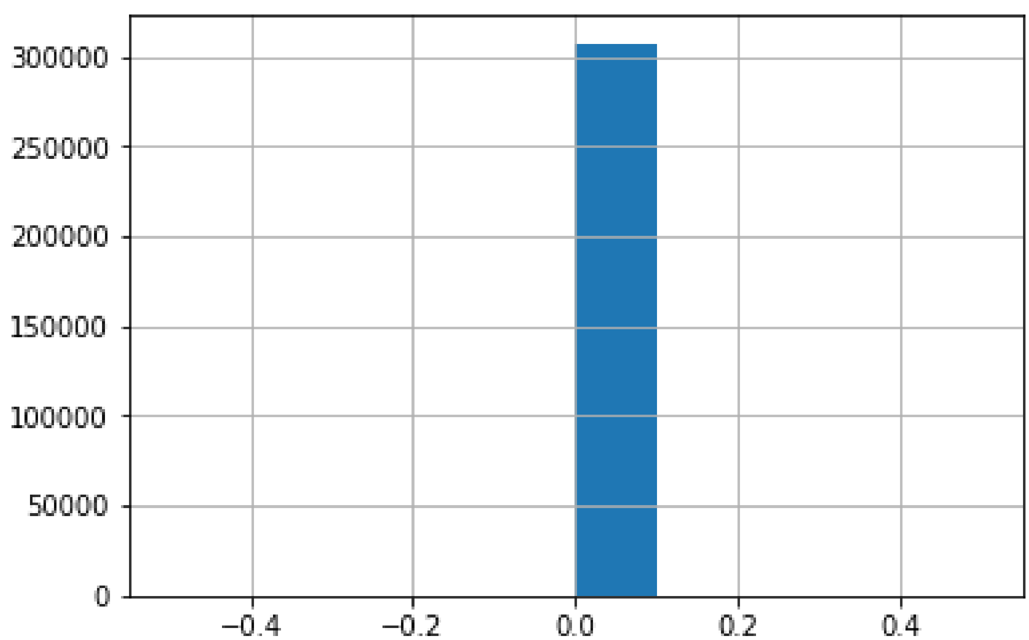

```
In [125]: application_bureau_loan_train_data['PREV_CASH_AMT_CREDIT'].hist()
          plt.show()
```



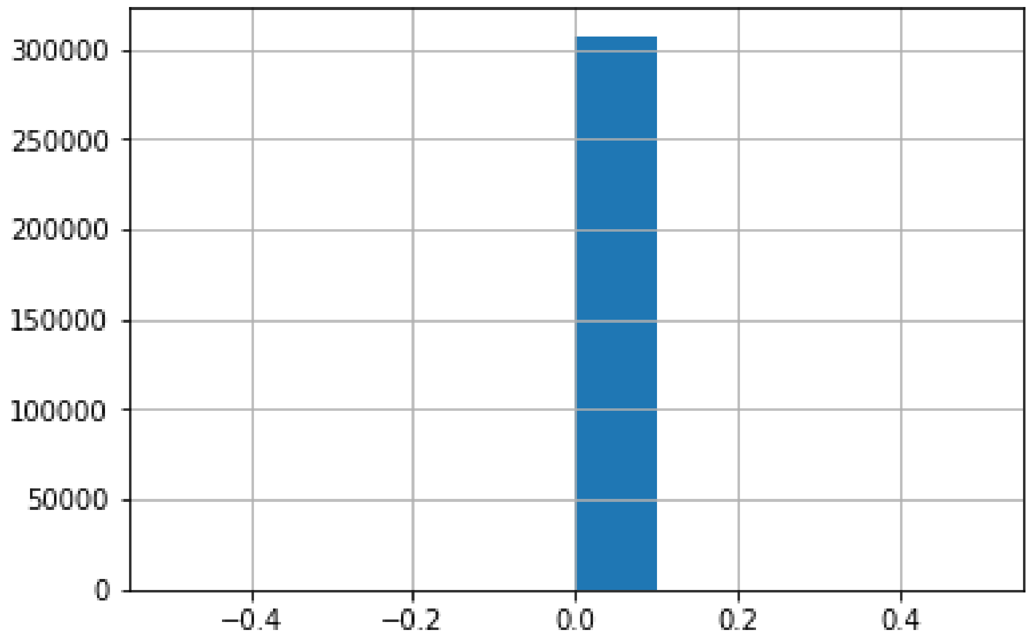
In [126]:
```
np.log(application_bureau_loan_train_data['PREV_CASH_AMT_CREDIT'] +1).hist()
plt.show()
```



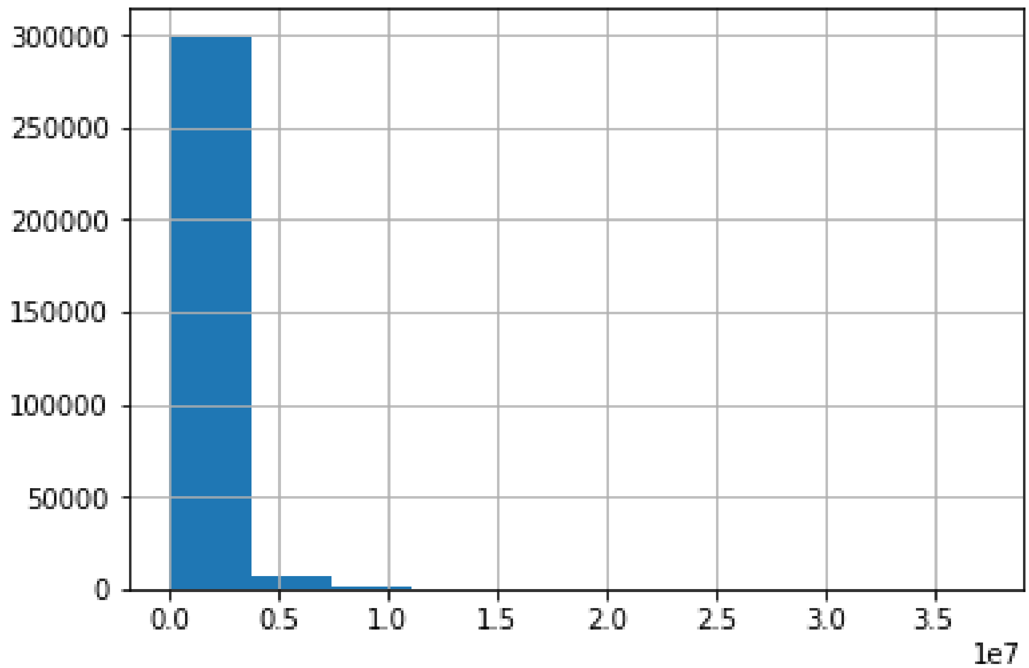
In [127]:
```
application_bureau_loan_train_data['PREV_CASH_AMT_DOWN_PAYMENT'].hist()
plt.show()
```



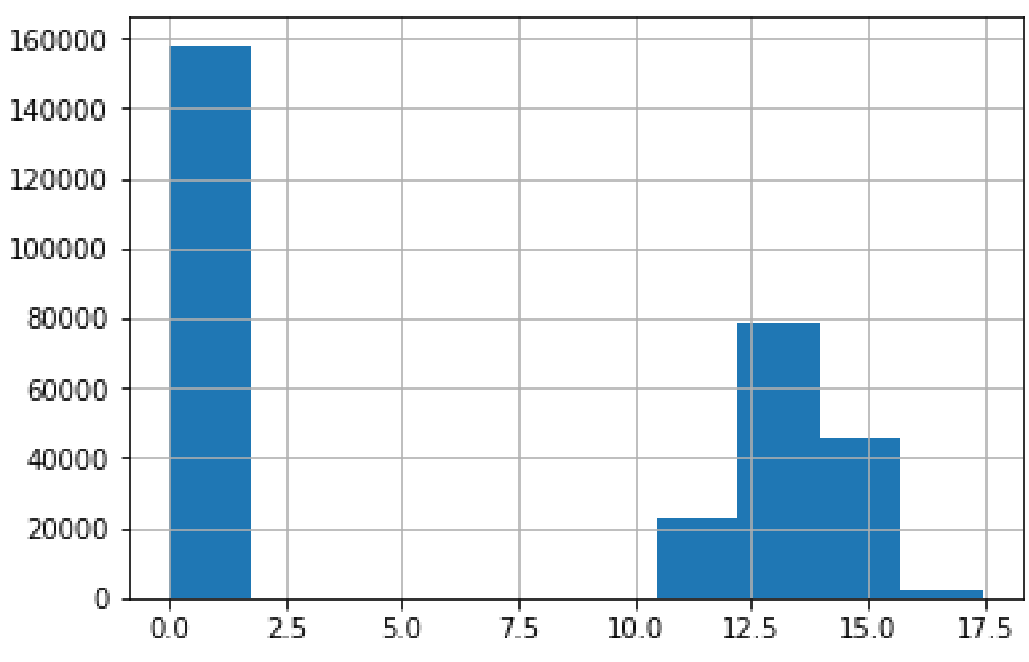
In [128]:
```
np.log(application_bureau_loan_train_data['PREV_CASH_AMT_DOWN_PAYMENT'] + 1).hist()
plt.show()
```



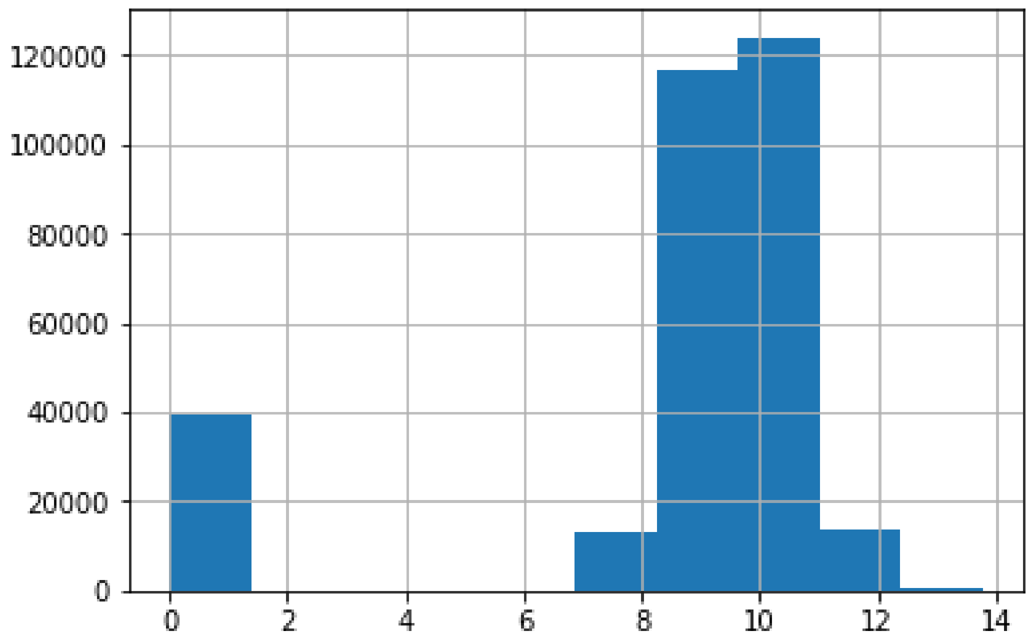
In [129]:
```
application_bureau_loan_train_data['PREV_CASH_AMT_GOODS_PRICE'].hist()
plt.show()
```




```
In [130]: np.log(application_bureau_loan_train_data['PREV_CASH_AMT_GOODS_PRICE'] + 1).hist()
          plt.show()
```



```
In [131]: np.log(application_bureau_loan_train_data['PREV_CONSUMER_AMT_ANNUITY'] + 1).hist()
          plt.show()
```



1.23 Field Transformations

i. Logarithmic Transformation: For highly-skewed feature distributions such as AMT_INCOME_TOTAL', 'AMT_CREDIT', logarithmic transformation is done on the data so that the very large and very small values do not negatively affect the performance of a learning algorithm. Using a logarithmic transformation significantly reduces the range of values caused by outliers. Care must be taken when applying this transformation however: The logarithm of 0 is undefined, so we must translate the values by a small amount above 0 to apply the the logarithm successfully.

ii. Normalizing Numerical Features

In addition to performing transformations on features that are highly skewed, it is often good practice to perform some type of scaling on numerical features. Applying a scaling to the data does not change the shape of each feature's distribution (such as AMT_INCOME_TOTAL', 'AMT_CREDIT' above); however, normalization ensures that each feature is treated equally when applying supervised learners. Note that once scaling is applied, observing the data in its raw form will no longer have the same original meaning, as exampled below.

iii.One hot encoding for categorical features Categorical variables having more than two possible vlaues are encoded using the one-hot encoding scheme. One-hot encoding creates a "dummy" variable for each possible category of each non-numeric feature. For example, assume someFeature has three possible entries: A, B, or C. We then encode this feature into someFeature_A, someFeature_B and someFeature_C.

iv. Label Encoding: Categorical variables having more than two possible are encoded using Label Encode to have values 0 and 1

v. Drop not relevant fields:

Some of the fields are not relevant for this project, this is based on analysis, intuitions and domain knowlege are dropped
Refrences: Udacity my earlier project on Finding donors https://github.com/monimoyd/finding_donors/blob/master/finding_donors.ipynb

In [49]:
```
# Perform log transformation
log_transform_fields = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PR
                        'PREV_CASH_AMT_APPLICATION', 'PREV_CASH_AMT_CREDIT', 'PREV_CAS
                        'PREV_CASH_AMT_GOODS_PRICE', 'PREV_CONSUMER_AMT_ANNUITY', 'PRE
                        'PREV_CONSUMER_AMT_CREDIT', 'PREV_CONSUMER_AMT_DOWN_PAYMENT',
                        'PREV_REVOVING_AMT_ANNUITY', 'PREV_REVOLVING_AMT_APPLICATION',
                        'PREV_REVOVING_AMT_DOWN_PAYMENT', 'PREV_REVOVING_AMT_GOODS_PRI
                        'PREV_XNA_AMT_APPLICATION', 'PREV_XNA_AMT_CREDIT',   'PREV_XNA
                        'PREV_XNA_AMT_GOODS_PRICE']
train_data = pd.DataFrame(data = application_bureau_loan_train_data)
train_data[log_transform_fields] = application_bureau_loan_train_data[log_transform_f

test_data = pd.DataFrame(data = application_bureau_loan_test_data)
test_data[log_transform_fields] = application_bureau_loan_test_data[log_transform_fie
```

In [50]:
```
days_transform_fields = ['DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID
temp1 = pd.DataFrame(data = train_data)
temp1[days_transform_fields] = train_data[days_transform_fields].apply(lambda x: -1.0
train_data = temp1

temp2 = pd.DataFrame(data = test_data)
temp2[days_transform_fields] = test_data[days_transform_fields].apply(lambda x: -1.0
test_data = temp2
```

In [51]:
```
train_data['EXT_SOURCE'] = train_data['EXT_SOURCE_1'] + train_data['EXT_SOURCE_2'] +
test_data['EXT_SOURCE'] = test_data['EXT_SOURCE_1'] + test_data['EXT_SOURCE_2'] +  te
```

In [52]:
```
# Drop fields which are not very relevant
drop_fields = ['CODE_GENDER','WEEKDAY_APPR_PROCESS_START','HOUR_APPR_PROCESS_START','
               'FLAG_EMP_PHONE','FLAG_WORK_PHONE','FLAG_CONT_MOBILE','FLAG_PHONE','FL
               'ELEVATORS_AVG','FLOORSMIN_AVG','FLOORSMAX_AVG','APARTMENTS_AVG', 'BAS
               'APARTMENTS_MODE','BASEMENTAREA_MODE','YEARS_BEGINEXPLUATATION_MODE','
               'ELEVATORS_MODE','ENTRANCES_MODE','FLOORSMAX_MODE','FLOORSMIN_MODE','L
               'LIVINGAREA_MODE','NONLIVINGAPARTMENTS_MODE','NONLIVINGAREA_MODE','APA
               'YEARS_BEGINEXPLUATATION_MEDI','YEARS_BUILD_MEDI','COMMONAREA_MEDI','E
               'FLOORSMAX_MEDI','FLOORSMIN_MEDI','LANDAREA_MEDI','LIVINGAPARTMENTS_ME
               'NONLIVINGAPARTMENTS_MEDI','NONLIVINGAREA_MEDI','OBS_30_CNT_SOCIAL_CIR
               'OBS_60_CNT_SOCIAL_CIRCLE','DEF_60_CNT_SOCIAL_CIRCLE','DAYS_LAST_PHONE
               'AMT_REQ_CREDIT_BUREAU_DAY','AMT_REQ_CREDIT_BUREAU_WEEK','AMT_REQ_CRED
```

```
                'AMT_REQ_CREDIT_BUREAU_YEAR', 'EXT_SOURCE_1','EXT_SOURCE_2','EXT_SOURC

# drop columns from train_data
train_data.drop(drop_fields, axis=1, inplace=True)

# drop columns from test_data
test_data.drop(drop_fields, axis=1, inplace=True)
```

In [53]:
```
from sklearn.preprocessing import MinMaxScaler

# Initialize a scaler, then apply it to the features
scaler = MinMaxScaler() # default=(0, 1)

numerical = ['AMT_INCOME_TOTAL','AMT_CREDIT','AMT_GOODS_PRICE','DAYS_BIRTH','DAYS_EMP
'YEARS_CREDIT_ACTIVE','CREDIT_YEAR_OVERDUE_ACTIVE','YEARS_CREDIT_ENDDATE_ACTIVE','YEA
'AMT_CREDIT_MAX_OVERDUE_ACTIVE','AMT_CREDIT_SUM_ACTIVE','AMT_CREDIT_SUM_DEBT_ACTIVE',
'AMT_CREDIT_SUM_OVERDUE_ACTIVE','AMT_ANNUITY_ACTIVE','YEARS_CREDIT_CLOSED','CREDIT_YE
'YEARS_CREDIT_ENDDATE_CLOSED','YEARS_ENDDATE_FACT_CLOSED','AMT_CREDIT_MAX_OVERDUE_CLO
'AMT_CREDIT_SUM_CLOSED','AMT_CREDIT_SUM_DEBT_CLOSED','AMT_CREDIT_SUM_LIMIT_CLOSED','A
'AMT_ANNUITY_CLOSED','YEARS_CREDIT_SOLD','CREDIT_YEAR_OVERDUE_SOLD','YEARS_CREDIT_END
'AMT_CREDIT_MAX_OVERDUE_SOLD','CNT_CREDIT_PROLONG_SOLD','AMT_CREDIT_SUM_SOLD','AMT_CR
'AMT_CREDIT_SUM_LIMIT_SOLD','AMT_CREDIT_SUM_OVERDUE_SOLD','AMT_ANNUITY_SOLD','YEARS_C
'CREDIT_YEAR_OVERDUE_BAD_DEBT','YEARS_CREDIT_ENDDATE_BAD_DEBT','YEARS_ENDDATE_FACT_BA
'CNT_CREDIT_PROLONG_BAD_DEBT','AMT_CREDIT_SUM_BAD_DEBT','AMT_CREDIT_SUM_DEBT_BAD_DEBT
'AMT_CREDIT_SUM_OVERDUE_BAD_DEBT','AMT_ANNUITY_BAD_DEBT',
'PREV_CASH_AMT_ANNUITY','PREV_CASH_AMT_APPLICATION','PREV_CASH_AMT_CREDIT','PREV_CASH
'PREV_CONSUMER_AMT_ANNUITY','PREV_CONSUMER_AMT_APPLICATION','PREV_CONSUMER_AMT_CREDIT
'PREV_CONSUMER_AMT_GOODS_PRICE','PREV_REVOVING_AMT_ANNUITY','PREV_REVOLVING_AMT_APPLI
'PREV_REVOVING_AMT_DOWN_PAYMENT','PREV_REVOVING_AMT_GOODS_PRICE','PREV_XNA_AMT_ANNUIT
'PREV_XNA_AMT_CREDIT','PREV_XNA_AMT_DOWN_PAYMENT','PREV_XNA_AMT_GOODS_PRICE','EXT_SOU

temp1 = pd.DataFrame(data = train_data)
temp1[numerical] = scaler.fit_transform( train_data[numerical])
train_data = temp1

temp2 = pd.DataFrame(data = test_data)
temp2[numerical] = scaler.fit_transform( test_data[numerical])
test_data = temp2
```

In [54]:
```
from sklearn.preprocessing import LabelEncoder

label_encoder1 = LabelEncoder()
label_encoder2 = LabelEncoder()

label_count1 = 0
for i in train_data:
    if  train_data[i].dtype=='object':
        if len(list(train_data[i].unique())) <=2:
```

```
                label_encoder1.fit(train_data[i])
                train_data[i]=label_encoder1.transform(train_data[i])
            label_count1 +=1
    print('%d columns of train_data are encoded.'%label_count1)

    label_count2 = 0
    for i in test_data:
        if  test_data[i].dtype=='object':
            if len(list(test_data[i].unique())) <=2:
                label_encoder2.fit(test_data[i])
                test_data[i]=label_encoder2.transform(test_data[i])
            label_count2 +=1
    print('%d columns of test_data are encoded.'%label_count2)
```

```
14 columns of train_data are encoded.
14 columns of test_data are encoded.
```

In [55]:
```
#  One-hot encode the 'train_data' data using pandas.get_dummies()
categorical = ['NAME_TYPE_SUITE','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE','NAME_FAMIL
'FONDKAPREMONT_MODE','HOUSETYPE_MODE','WALLSMATERIAL_MODE','EMERGENCYSTATE_MODE']

train_data = pd.get_dummies(data = train_data, columns = categorical)

#  One-hot encode the 'test_data' data using pandas.get_dummies()
categorical = ['NAME_TYPE_SUITE','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE','NAME_FAMIL
'FONDKAPREMONT_MODE','HOUSETYPE_MODE','WALLSMATERIAL_MODE','EMERGENCYSTATE_MODE']

#features_log_minmax_transform = pd.DataFrame(data = features_log_transformed)
test_data = pd.get_dummies(data = test_data, columns = categorical)
```

In [56]:
```
train_data.drop(['NAME_INCOME_TYPE_Maternity leave', 'NAME_FAMILY_STATUS_Unknown'], a
```

In [57]:
```
# Drop the fields TARGET, SK_ID_CURR from train_data to create dataframe train_data_x
train_data_x = train_data.drop(['TARGET', 'SK_ID_CURR'], axis=1)

# Get only filed TARGET to create dataframe train_data_y
train_data_y = train_data['TARGET']
```

In [58]:
```
train_data_x.head()
```

Out[58]:

	NAME_CONTRACT_TYPE	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL
0	0	0	1	0	0.245232
1	0	0	0	0	0.279376
2	1	1	1	0	0.114839
3	0	0	1	0	0.197108
4	0	0	1	0	0.184602

```
   AMT_ANNUITY_CLOSED  YEARS_CREDIT_SOLD  CREDIT_YEAR_OVERDUE_SOLD  YEARS_CREDIT_ENDD
```

```
   0             0.0            1.0                     0.0
   1             0.0            1.0                     0.0
   2             0.0            1.0                     0.0
   3             0.0            1.0                     0.0
   4             0.0            1.0                     0.0

      NAME_INCOME_TYPE_Commercial associate  NAME_INCOME_TYPE_Pensioner  NAME_INCOME_TYPE
   0                                      0                           0
   1                                      0                           0
   2                                      0                           0
   3                                      0                           0
   4                                      0                           0

      ORGANIZATION_TYPE_Business Entity Type 2  ORGANIZATION_TYPE_Business Entity Type 3
   0                                         0                                         1
   1                                         0                                         0
   2                                         0                                         0
   3                                         0                                         1
   4                                         0                                         0

      ORGANIZATION_TYPE_Trade: type 6  ORGANIZATION_TYPE_Trade: type 7  ORGANIZATION_TYPE
   0                                0                                0
   1                                0                                0
   2                                0                                0
   3                                0                                0
   4                                0                                0
```

In [59]:
```
train_data_y.head()
```

Out[59]:
```
0    1
1    0
2    0
3    0
4    0
Name: TARGET, dtype: int64
```

In [60]:
```
# Remove SK_ID_CURR field from test_data to create dataframe test_data_x which will b
test_data_x = test_data.drop([ 'SK_ID_CURR'], axis=1)

# Get SK_ID_CURR from test_data to create dataframe test_data_id which will be used f
test_data_id = test_data['SK_ID_CURR']
```

In [61]:
```
test_data_x.head()
```

Out[61]:
```
   NAME_CONTRACT_TYPE  FLAG_OWN_CAR  FLAG_OWN_REALTY  CNT_CHILDREN  AMT_INCOME_TOTAL
0                   0             0                1             0          0.316124
1                   0             0                1             0          0.255285
2                   0             1                1             0          0.395659
3                   0             0                1             2          0.482328
```

```
4                    0              1                0              1          0.372555

   AMT_ANNUITY_CLOSED  YEARS_CREDIT_SOLD  CREDIT_YEAR_OVERDUE_SOLD  YEARS_CREDIT_ENDDA
0                 0.0                1.0                       0.0                   0
1                 0.0                1.0                       0.0                   0
2                 0.0                1.0                       0.0                   0
3                 0.0                1.0                       0.0                   0
4                 0.0                1.0                       0.0                   0

   NAME_INCOME_TYPE_Commercial associate  NAME_INCOME_TYPE_Pensioner  NAME_INCOME_TYPE
0                                      0                           0
1                                      0                           0
2                                      0                           0
3                                      0                           0
4                                      0                           0

   ORGANIZATION_TYPE_Business Entity Type 2  ORGANIZATION_TYPE_Business Entity Type 3
0                                         0                                         0
1                                         0                                         0
2                                         0                                         0
3                                         0                                         1
4                                         0                                         1

   ORGANIZATION_TYPE_Trade: type 6  ORGANIZATION_TYPE_Trade: type 7  ORGANIZATION_TYPE
0                                0                                0
1                                0                                0
2                                0                                0
3                                0                                0
4                                0                                0
```

In [62]:
```
test_data_id.head()
```

Out[62]:
```
0    100001
1    100005
2    100013
3    100028
4    100038
Name: SK_ID_CURR, dtype: int64
```

In [63]:
```
# Check if there is any field which is there in train_data_x but not in test_data_x a
train_col_set = set(train_data_x.columns.values.tolist())
test_col_set = set(test_data_x.columns.values.tolist())

train_minus_test_list = list(train_col_set - test_col_set)
test_minus_train_list = list(test_col_set - train_col_set)
```

In [72]:
```
train_minus_test_list
```

Out[72]:
```
[]
```

In [73]: test_minus_train_list

Out[73]: []

1.24 Cross Validation

Split the train_data_x and train_data_y into 70% into training dataframes X_train,y_train and 30% to Test dataframes X_test,y_test

```
In [64]: from sklearn.cross_validation import train_test_split

         # Split the 'features' and 'income' data into training and testing sets
         X_train, X_test, y_train, y_test = train_test_split(train_data_x,
                                                             train_data_y,
                                                             test_size = 0.3,
                                                             random_state = 0)

         # Show the results of the split
         print("Training set has {} samples.".format(X_train.shape[0]))
         print("Testing set has {} samples.".format(X_test.shape[0]))
```

```
E:\anaconda\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module w
  "This module will be removed in 0.20.", DeprecationWarning)


Training set has 215257 samples.
Testing set has 92254 samples.
```

```
In [66]: X_train.shape

Out[66]: (215257, 250)

In [67]: y_train.shape

Out[67]: (215257,)

In [68]: X_test.shape

Out[68]: (92254, 250)

In [75]: y_test.shape

Out[75]: (92254,)
```

1.25 Naive Predictor Performace

The purpose of generating a naive predictor is simply to show what a base model without any intelligence would look like. In the real world, ideally your base model would be either the results of a previous model or could be based on a research paper upon which you are looking to improve. When there is no benchmark model set, getting a result better than random choice is a place we could start from.

```
In [69]: TP = np.sum(y_train) # Counting the ones as this is the naive case.

         FP = y_train.count() - TP # Specific to the naive case

         TN = 0 # No predicted negatives in the naive case
         FN = 0 # No predicted negatives in the naive case

         #  Calculate accuracy, precision and recall
         accuracy = float(TP + TN)/float(TP + FP + TN + FN)
         recall = float(TP)/float(TP + FN)
         precision = float(TP)/float(TP + FP)

         #  Calculate f1_score.
         f1_score_value = float(2.0 * accuracy * recall/(accuracy + recall))

         # Print the results
         print("Naive Predictor: [Accuracy score: {:.4f}, F-score: {:.4f}]".format(accuracy, f1
```

```
Naive Predictor: [Accuracy score: 0.0812, F-score: 0.1503]
```

1.26 Apply Supervised Machine Learning Models

The following six supervised learning models that are currently available in scikit-learn are used to train the data:

i. Decision Trees: Decision tree is used for prediction and assessing the relative importance of variables. for the current problem we will need to do prediction for home loan, decision tree can be used

ii. Logistic Regression: logistic regression is a simple model moves with non-linear function hence can work with linearly and non-linearly separable problems

iii. Gaussian Naive Bayes (GaussianNB): Gaussian Naive Bayes is a simple but powerful algorithm for predictive modeling suitable for current problems

iv. Gradient Boosting: Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

v. XGB Boosting: XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

vi. Random Forest: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees

Note: I have first tried first 3 models and next tried last 3 models to take advantage of existing framework created by Udacity

The model which best ROC-AUC score but satisfactory Accuracy score and also capable of generating actual probability using predict_proba will be chosen

Note: I have used some of the works from my previous finding_donors Udacity project: http://localhost:8888/notebooks/finding_donors-master/finding_donors-master/finding_donors.ipynb

References: https://en.wikipedia.org/wiki/Gradient_boosting https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/ https://en.wikipedia.org/wiki/Random_forest

In [ ]:
```
## Metrics used for evaluating models:

ROC-AUC score: An ROC curve (receiver operating characteristic curve) is a graph showi
model at all classification thresholds. This curve plots two parameters: True Positive
True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:
        recall  =  (true positives )/(true positives + false negatives)

An ROC curve plots TPR vs. FPR at different classification thresholds.  AUC stands for
AUC measures the entire two dimensional area underneath the entire ROC curve. AUC range
0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose   predic
This Kaggle competition is judged based on ROC-AUC score, so I will be using this  Sco

However, I will also calculate Accuracy and F1-score for completeness and for Comparin

Accuracy: Accuracy   is a common metric for binary classifiers. It takes into account
         accuracy  =  (true positives + true negatives)/dataset size

Precision:
        precision  =  (true positives )/(true positives + false positive)
Recall:
        recall  =  (true positives )/(true positives + false negatives)

F1-score:
     F1-score  =  2*(precision * recall )/(precision + recall)

After training is done , the main metrics that will be used for selection is ROC-AUC s
training and are calculated for each model.
```

Reference:
https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

In [70]:
```
#  Import two metrics from sklearn - fbeta_score and accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score


def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    '''
    inputs:
       - learner: the learning algorithm to be trained and predicted on
       - sample_size: the size of samples (number) to be drawn from training set
       - X_train: features training set
       - y_train: income training set
       - X_test: features testing set
       - y_test: income testing set
    '''

    results = {}

    # Fit the learner to the training data using slicing with 'sample_size' using .fi
    start = time() # Get start time
    learner = learner
    print( "Doing learner.fit")
    learner.fit(X_train, y_train)
    end = time() # Get end time
    print ("Done learner.fit")

    #  Calculate the training time
    results['train_time'] = end - start
    print( "training time=" + str(results['train_time']))

    # Get the predictions on the test set(X_test),
    #       then get predictions on the first 300 training samples(X_train) using .pr
    start = time() # Get start time
    print ("Doing learner.predict X_test")
    predictions_test = learner.predict(X_test)
    print( "Doing learner.predict X_train 300 samples")
    predictions_train = learner.predict(X_train[:300])
    end = time() # Get end time

    #  Calculate the total prediction time
    results['pred_time'] = end - start
    print("prediction time=" + str(results['pred_time']))
```

```
    #  Compute accuracy on the first 300 training samples which is y_train[:300]
    print( "Calculating accuracy_score")
    results['acc_train'] = accuracy_score(predictions_train, y_train[:300])
    print("accuracy_score on 300 samples of training data=" + str(results['acc_train']

    #  Compute accuracy on test set using accuracy_score()
    results['acc_test'] = accuracy_score(predictions_test, y_test)
    print( "accuracy_score on test data=" + str(results['acc_test']))

    #  Compute F1-score on the the first 300 training samples using f1_score()
    print ("Calculating f1_score")
    results['f_train'] = f1_score( y_train[:300],predictions_train)
    print ("f1_score on 300 samples of training data=" + str(results['f_train']))

    #  Compute F-score on the test set which is y_test
    results['f_test'] = f1_score(y_test, predictions_test)
    print ("f1_score on test data=" + str(results['f_test']))

     #  Compute F1-score on the the first 300 training samples using f1_score()
    print ("Calculating f1_score")
    results['roc_auc_score_train'] = roc_auc_score( y_train[:300],predictions_train)
    print ("roc_auc_score on 300 samples of training data=" + str(results['roc_auc_sco

    results['roc_auc_score_test'] = roc_auc_score( y_test, predictions_test)
    print ("roc_auc_score on test data=" + str(results['roc_auc_score_test']))

    # Success
    print("{} trained on {} samples.".format(learner.__class__.__name__, sample_size)

    # Return the results
    return results
```

In [71]:
```
# Import the three supervised learning models from sklearn
from time import time
from IPython.display import display # Allows the use of display() for DataFrames

# Import supplementary visualization code visuals.py
import visuals as vs
from sklearn import tree
from sklearn.linear_model import LogisticRegression
#from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

# Initialize the three models

clf_A = tree.DecisionTreeClassifier(random_state=0)
clf_B = LogisticRegression(random_state=0)
```

```python
    clf_C = GaussianNB()


    #  Calculate the number of samples for 1%, 10%, and 100% of the training data
    # HINT: samples_100 is the entire training set i.e. len(y_train)
    # HINT: samples_10 is 10% of samples_100 (ensure to set the count of the values to be
    # HINT: samples_1 is 1% of samples_100 (ensure to set the count of the values to be `

    samples_100 = len(y_train)
    print( "samples_100 = " + str(samples_100))
    samples_10 = len(y_train) / 10
    print ("samples_10 = " + str(samples_10))
    samples_1 = len(y_train) / 100
    print ("samples_1=" + str(samples_1))

    # Collect results on the learners)
    results = {}
    for clf in [clf_A, clf_B, clf_C]:
        clf_name = clf.__class__.__name__
        print ("Getting results for clf_name=" + str(clf_name))
        results[clf_name] = {}
        for i, samples in enumerate([samples_1, samples_10, samples_100]):
            print ("Getting results for samples=" + str(samples))
            results[clf_name][i] = \
            train_predict(clf, samples, X_train, y_train, X_test, y_test)
```

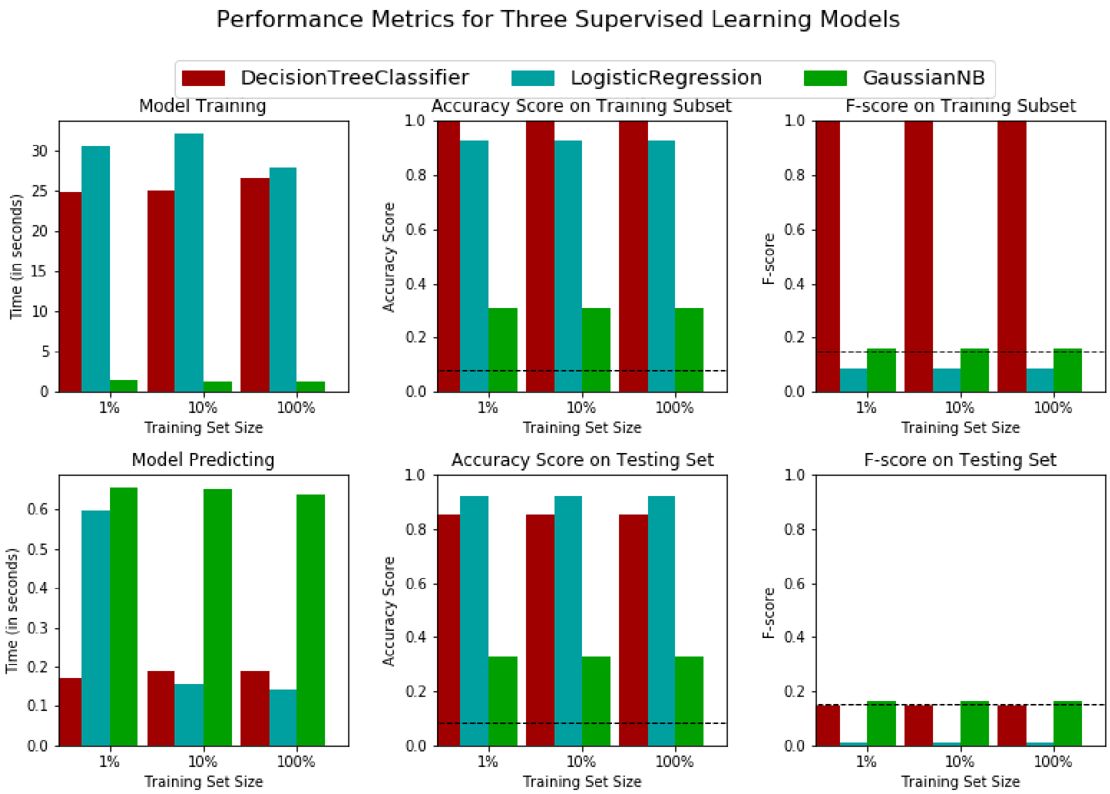
```
samples_100 = 215257
samples_10 = 21525.7
samples_1=2152.57
Getting results for clf_name=DecisionTreeClassifier
Getting results for samples=2152.57
Doing learner.fit
Done learner.fit
training time=24.81396222114563
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.17183756828308105
Calculating accuracy_score
accuracy_score on 300 samples of training data=1.0
accuracy_score on test data=0.8535022871637002
Calculating f1_score
f1_score on 300 samples of training data=1.0
f1_score on test data=0.14919735599622286
Calculating f1_score
roc_auc_score on 300 samples of training data=1.0
roc_auc_score on test data=0.5373841053737395
DecisionTreeClassifier trained on 2152.57 samples.
Getting results for samples=21525.7
```

```
Doing learner.fit
Done learner.fit
training time=24.950313568115234
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.18748211860656738
Calculating accuracy_score
accuracy_score on 300 samples of training data=1.0
accuracy_score on test data=0.8535022871637002
Calculating f1_score
f1_score on 300 samples of training data=1.0
f1_score on test data=0.14919735599622286
Calculating f1_score
roc_auc_score on 300 samples of training data=1.0
roc_auc_score on test data=0.5373841053737395
DecisionTreeClassifier trained on 21525.7 samples.
Getting results for samples=215257
Doing learner.fit
Done learner.fit
training time=26.52279257774353
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.18745732307434082
Calculating accuracy_score
accuracy_score on 300 samples of training data=1.0
accuracy_score on test data=0.8535022871637002
Calculating f1_score
f1_score on 300 samples of training data=1.0
f1_score on test data=0.14919735599622286
Calculating f1_score
roc_auc_score on 300 samples of training data=1.0
roc_auc_score on test data=0.5373841053737395
DecisionTreeClassifier trained on 215257 samples.
Getting results for clf_name=LogisticRegression
Getting results for samples=2152.57
Doing learner.fit
Done learner.fit
training time=30.5544490814209
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.5981895923614502
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.9266666666666666
accuracy_score on test data=0.9204045353047022
Calculating f1_score
f1_score on 300 samples of training data=0.08333333333333333
f1_score on test data=0.012108166285483654
Calculating f1_score
```

```
roc_auc_score on 300 samples of training data=0.5217391304347826
roc_auc_score on test data=0.5027827561901226
LogisticRegression trained on 2152.57 samples.
Getting results for samples=21525.7
Doing learner.fit
Done learner.fit
training time=32.00682187080383
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.15621423721313477
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.9266666666666666
accuracy_score on test data=0.9204045353047022
Calculating f1_score
f1_score on 300 samples of training data=0.08333333333333333
f1_score on test data=0.012108166285483654
Calculating f1_score
roc_auc_score on 300 samples of training data=0.5217391304347826
roc_auc_score on test data=0.5027827561901226
LogisticRegression trained on 21525.7 samples.
Getting results for samples=215257
Doing learner.fit
Done learner.fit
training time=27.85963487625122
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.14059233665466309
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.9266666666666666
accuracy_score on test data=0.9204045353047022
Calculating f1_score
f1_score on 300 samples of training data=0.08333333333333333
f1_score on test data=0.012108166285483654
Calculating f1_score
roc_auc_score on 300 samples of training data=0.5217391304347826
roc_auc_score on test data=0.5027827561901226
LogisticRegression trained on 215257 samples.
Getting results for clf_name=GaussianNB
Getting results for samples=2152.57
Doing learner.fit
Done learner.fit
training time=1.4183473587036133
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.6556863784790039
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.30666666666666664
accuracy_score on test data=0.3256227372254862
```

```
Calculating f1_score
f1_score on 300 samples of training data=0.16129032258064516
f1_score on test data=0.16414982803095443
Calculating f1_score
roc_auc_score on 300 samples of training data=0.5647465076126197
roc_auc_score on test data=0.5570575733653007
GaussianNB trained on 2152.57 samples.
Getting results for samples=21525.7
Doing learner.fit
Done learner.fit
training time=1.2648911476135254
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.6526141166687012
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.30666666666666664
accuracy_score on test data=0.3256227372254862
Calculating f1_score
f1_score on 300 samples of training data=0.16129032258064516
f1_score on test data=0.16414982803095443
Calculating f1_score
roc_auc_score on 300 samples of training data=0.5647465076126197
roc_auc_score on test data=0.5570575733653007
GaussianNB trained on 21525.7 samples.
Getting results for samples=215257
Doing learner.fit
Done learner.fit
training time=1.286242961883545
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.6372771263122559
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.30666666666666664
accuracy_score on test data=0.3256227372254862
Calculating f1_score
f1_score on 300 samples of training data=0.16129032258064516
f1_score on test data=0.16414982803095443
Calculating f1_score
roc_auc_score on 300 samples of training data=0.5647465076126197
roc_auc_score on test data=0.5570575733653007
GaussianNB trained on 215257 samples.
```

In [72]:
```
# Run metrics visualization for the three supervised learning models chosen
vs.evaluate(results, accuracy, f1_score_value)
```




```
In [73]: # Import the three supervised learning models from sklearn
         from time import time
         from IPython.display import display # Allows the use of display() for DataFrames

         # Import supplementary visualization code visuals.py
         import visuals as vs
         from xgboost import XGBClassifier
         from sklearn.ensemble import GradientBoostingClassifier
         from sklearn.ensemble import RandomForestClassifier


         # Initialize the three models

         clf_D = XGBClassifier(random_state=0)
         clf_E = GradientBoostingClassifier(random_state=0)
         clf_F = RandomForestClassifier(n_estimators=30,random_state=0)


         samples_100 = len(y_train)
         print( "samples_100 = " + str(samples_100))
         samples_10 = len(y_train) / 10
         print ("samples_10 = " + str(samples_10))
```

```
samples_1 = len(y_train) / 100
print ("samples_1=" + str(samples_1))

# Collect results on the learners)
results = {}
for clf in [clf_D, clf_E, clf_F]:
    clf_name = clf.__class__.__name__
    print ("Getting results for clf_name=" + str(clf_name))
    results[clf_name] = {}
    for i, samples in enumerate([samples_1, samples_10, samples_100]):
        print ("Getting results for samples=" + str(samples))
        results[clf_name][i] = \
        train_predict(clf, samples, X_train, y_train, X_test, y_test)
```

```
samples_100 = 215257
samples_10 = 21525.7
samples_1=2152.57
Getting results for clf_name=XGBClassifier
Getting results for samples=2152.57
Doing learner.fit
Done learner.fit
training time=127.99821949005127
Doing learner.predict X_test
```

```
E:\anaconda\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The trutl
  if diff:
E:\anaconda\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The trutl
  if diff:
E:\anaconda\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: I
  'precision', 'predicted', average, warn_for)
```

```
Doing learner.predict X_train 300 samples
prediction time=1.2992854118347168
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.9233333333333333
accuracy_score on test data=0.9205888091573265
Calculating f1_score
f1_score on 300 samples of training data=0.0
f1_score on test data=0.005700325732899024
Calculating f1_score
roc_auc_score on 300 samples of training data=0.5
roc_auc_score on test data=0.50138929953749
XGBClassifier trained on 2152.57 samples.
Getting results for samples=21525.7
Doing learner.fit
Done learner.fit
```

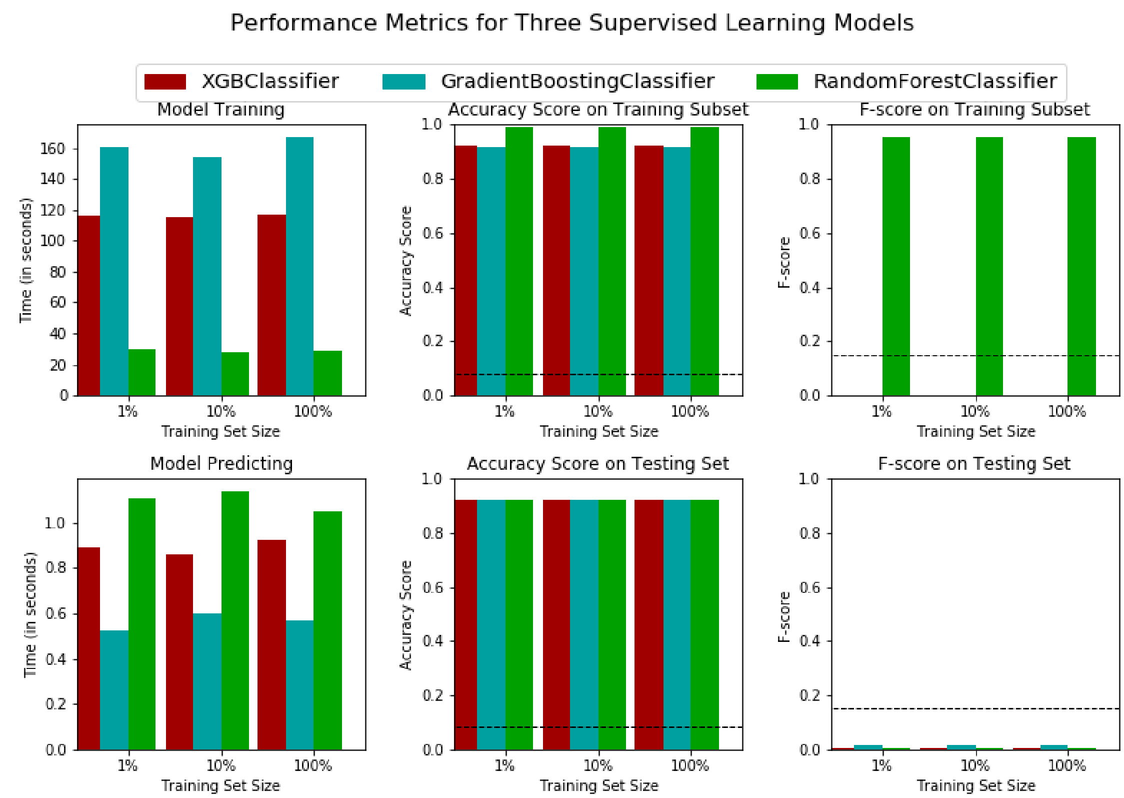
```
training time=125.9269917011261
Doing learner.predict X_test


E:\anaconda\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The trut
  if diff:
E:\anaconda\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The trut
  if diff:
E:\anaconda\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: 
  'precision', 'predicted', average, warn_for)


Doing learner.predict X_train 300 samples
prediction time=0.9042308330535889
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.9233333333333333
accuracy_score on test data=0.9205888091573265
Calculating f1_score
f1_score on 300 samples of training data=0.0
f1_score on test data=0.005700325732899024
Calculating f1_score
roc_auc_score on 300 samples of training data=0.5
roc_auc_score on test data=0.50138929953749
XGBClassifier trained on 21525.7 samples.
Getting results for samples=215257
Doing learner.fit
Done learner.fit
training time=122.79929876327515
Doing learner.predict X_test


E:\anaconda\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The trut
  if diff:
E:\anaconda\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The trut
  if diff:
E:\anaconda\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: 
  'precision', 'predicted', average, warn_for)


Doing learner.predict X_train 300 samples
prediction time=0.8904173374176025
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.9233333333333333
accuracy_score on test data=0.9205888091573265
Calculating f1_score
f1_score on 300 samples of training data=0.0
f1_score on test data=0.005700325732899024
Calculating f1_score
roc_auc_score on 300 samples of training data=0.5
```

```
roc_auc_score on test data=0.50138929953749
XGBClassifier trained on 215257 samples.
Getting results for clf_name=GradientBoostingClassifier
Getting results for samples=2152.57
Doing learner.fit
Done learner.fit
training time=166.84947872161865
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.6240260601043701
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.92
accuracy_score on test data=0.9204804127734298
Calculating f1_score
f1_score on 300 samples of training data=0.0
f1_score on test data=0.0158304266165817
Calculating f1_score
roc_auc_score on 300 samples of training data=0.4981949458483754
roc_auc_score on test data=0.5036952164905554
GradientBoostingClassifier trained on 2152.57 samples.
Getting results for samples=21525.7
Doing learner.fit
Done learner.fit
training time=162.90181159973145
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.5821716785430908
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.92
accuracy_score on test data=0.9204804127734298
Calculating f1_score
f1_score on 300 samples of training data=0.0
f1_score on test data=0.0158304266165817
Calculating f1_score
roc_auc_score on 300 samples of training data=0.4981949458483754
roc_auc_score on test data=0.5036952164905554
GradientBoostingClassifier trained on 21525.7 samples.
Getting results for samples=215257
Doing learner.fit
Done learner.fit
training time=166.7149350643158
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.5310990810394287
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.92
accuracy_score on test data=0.9204804127734298
Calculating f1_score
```

```
f1_score on 300 samples of training data=0.0
f1_score on test data=0.0158304266165817
Calculating f1_score
roc_auc_score on 300 samples of training data=0.4981949458483754
roc_auc_score on test data=0.5036952164905554
GradientBoostingClassifier trained on 215257 samples.
Getting results for clf_name=RandomForestClassifier
Getting results for samples=2152.57
Doing learner.fit
Done learner.fit
training time=27.07782006263733
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=1.0639278888702393
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.9966666666666667
accuracy_score on test data=0.9204587334966505
Calculating f1_score
f1_score on 300 samples of training data=0.9777777777777777
f1_score on test data=0.004341926729986432
Calculating f1_score
roc_auc_score on 300 samples of training data=0.9782608695652174
roc_auc_score on test data=0.5010074819087675
RandomForestClassifier trained on 2152.57 samples.
Getting results for samples=21525.7
Doing learner.fit
Done learner.fit
training time=27.986596822738647
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=1.0768773555755615
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.9966666666666667
accuracy_score on test data=0.9204587334966505
Calculating f1_score
f1_score on 300 samples of training data=0.9777777777777777
f1_score on test data=0.004341926729986432
Calculating f1_score
roc_auc_score on 300 samples of training data=0.9782608695652174
roc_auc_score on test data=0.5010074819087675
RandomForestClassifier trained on 21525.7 samples.
Getting results for samples=215257
Doing learner.fit
Done learner.fit
training time=27.320880889892578
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=1.0689430236816406
```

```
Calculating accuracy_score
accuracy_score on 300 samples of training data=0.9966666666666667
accuracy_score on test data=0.9204587334966505
Calculating f1_score
f1_score on 300 samples of training data=0.9777777777777777
f1_score on test data=0.004341926729986432
Calculating f1_score
roc_auc_score on 300 samples of training data=0.9782608695652174
roc_auc_score on test data=0.5010074819087675
RandomForestClassifier trained on 215257 samples.
```

```
In [92]: # Run metrics visualization for the three supervised learning models chosen
         vs.evaluate(results, accuracy, f1_score_value)
```



1.27 Choosing the model

ROC-AU score, Accuracy Score, F score on test data and 300 training samples alongwith Training Time and Prediction time are listed below -

Metric	Decision Tree	Logistics	GaussianNB	XGBBoost	GradientBoost	RandomF
ROC-AUC (test)	0.537319	0.502448	0.557352	0.501365	0.503695	0.500826

Metric	Decision Tree	Logistics	GaussianNB	XGBBoost	GradientBoost	RandomF
ROC-AUC (300 training)	1.0	0.521739	0.564746	0.5	0.498194	0.956521
Accuracy (test)	0.85338	0.920361	0.325134	0.920545	0.920480	0.920470
Accuracy(300 training)	0.85338	0.926666	0.306667	0.923333 0.92	0.993333	
F1 score (test data)	0.149094	0.010771	0.164253	0.005697	0.015830	0.003531
F1 score(300 training)	1.0	0.083333	0.161290	0.0	0.0	0.954545
Training Time	24.800899	23.82941	1.342295	117.141126	166.722213	28.703508
Prediction Time	0.171837	0.109350	0.623674	0.922175	0.567169	1.050263

From the above result we can see teh ROC-AUC scorewise best model is GaussianNB having score 0.557352. However, we can not take this model as Accuracy is very low (only 0.32) also in sklearn this model does not provide the actual probability.

The next best model is: Decision tree having ROC-AUC score of 0.537319 and accuracy score 0.85338. But I could not consider this model, as Decision Tree model has overfitting issues. Another main reason is that it does not provide the actual probabiluty.

Next best model is: GradientBoost which has good ROC-AUC score of 0.503695 and accuracy score of 0.920480. Although, scores of XGBBoost is comparable with GradientBoost. But I would still go with GradientBoost, becuase it has better F1 score (0.015830) , than XGBBoost (0.005697)

So the finally I have chosen GradientBoost

1.28 Preparing to submit to Kaggle

Following steps are for preparing data for submission to Kaggle

- Use the predict_proba to the the GradientBoost model to get teh probabily
- Get SK_ID_CURR field of test_data and store as test_data_id
- Merge first column of probability and merge with test_data_id
- Save the dataframe to the CSV file
- Submit to Kaggle

```
In [87]: grad_boost_model = clf_E

In [88]: gbc_proba = grad_boost_model.predict_proba(test_data_x)

In [89]: gbc_proba

Out[89]: array([[0.95805732, 0.04194268],
                [0.44370391, 0.55629609],
                [0.83679721, 0.16320279],
                ...,
                [0.95118904, 0.04881096],
                [0.9196496 , 0.0803504 ],
                [0.75743379, 0.24256621]])

In [90]: gbc_proba_frame = pd.DataFrame(list(gbc_proba[:,1]), columns=['TARGET'])

In [91]: test_data_id = test_data[['SK_ID_CURR']]

In [92]: gbc_frame = pd.merge(test_data_id, gbc_proba_frame, left_index=True, right_index=True
```

In [93]: gbc_frame.head()

Out[93]:

	SK_ID_CURR	TARGET
0	100001	0.041943
1	100005	0.556296
2	100013	0.163203
3	100028	0.037223
4	100038	0.167112

In [94]:
```
gbc_frame.to_csv('credit_risk_submission.csv', index=False, header=True)
```

1.29 My score on kaggle (0.55958) is as below:

If image below is not visible, please use the URL:
https://drive.google.com/open?id=10CulFKD6OA21edTiAHSzeI1gbfxClhb9

1.30 My Ranking on Kaggle

As this is a late submission, I would not get the ranking for this submission (but original ranking was when I submitted before deadline was: 7159 on a score of 0.49769). The Highest score in Private Leadership is 0.80570.

It is not clear whether late submission score is based on Private Leadership rank or Public Leadership Rank. If I would have made the same submission before deadline and if late submission is based on Private leadership Rank, my rank in Private leadership would have been 6804 and if it is base on public leadership, my rank in public leadership would have been 6807. Total teams were 7198. Also, this is tthe first time I am participating in Kaggle competetion. Thanks a lot to Udactiy for this

1.31 Further Refinement of Model

The model can be further refined using grid search (GridSearchCV) where parameters with different values are provided and grid search finds the best . I also tried the following code to refine the model but unfortunately in my computer it got hang after multiple attempts.

Another way to refine using Deep Neural Network techniques, which I am still working on it

In [ ]:
```
#  Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score


#  Initialize the classifier
grad_boost_classifier = GradientBoostingClassifier(random_state=0)


parameters= {'n_estimators': [100, 150, 200], 'learning_rate':[ 0.5, 0.2, 0.1], 'max_d


#  Perform grid search on the classifier using 'scorer' as the scoring method using Gr
```

```python
grid_obj = GridSearchCV(grad_boost_classifier, param_grid=parameters,  scoring='roc_auc

#  Fit the grid search object to the training data and find the optimal parameters usi
print("Calling fit on grid_obj")
grid_fit = grid_obj.fit(X_train,y_train)

# Get the estimator
best_clf = grid_fit.best_estimator_

# Make predictions using the unoptimized and model
print("Calling predict")
predictions = (grad_boost_classifier.fit(X_train, y_train)).predict(X_test)
best_predictions = best_clf.predict(X_test)

best_parameters = grid_obj.best_estimator_.get_params()
print("Best parameters are:")
for param_name in sorted(parameters.keys()):
    print('\t%s: %r' % (param_name, best_parameters[param_name]))

# Report the before-and-afterscores
print("Unoptimized model\n------")
print("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, predictio
print("roc-auc score on testing data: {:.4f}".format(roc_auc_score(y_test, predictions
print("\nOptimized Model\n------")
print("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test,
print("Final roc-auc on the testing data: {:.4f}".format(roc_auc_score(y_test, best_pre
```