BANK LOAN CASE STUDY

PYTHON CODE

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

```
df_1=pd.read_csv('D:\\data ANALYTICS AND SCIENCE\\TRAINITY ASSIGN\\application_data.csv')
df_1
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_
0	100002	1	Cash loans	M	N	Y	0	202500.0	
1	100003	0	Cash loans	F	N	N	0	270000.0	1
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	
4	100007	0	Cash loans	M	N	Y	0	121500.0	
...	...	...	...	...	...	...	...	...	
307506	456251	0	Cash loans	M	N	N	0	157500.0	
307507	456252	0	Cash loans	F	N	Y	0	72000.0	
307508	456253	0	Cash loans	F	N	Y	0	153000.0	
307509	456254	1	Cash loans	F	N	Y	0	171000.0	
307510	456255	0	Cash loans	F	N	N	0	157500.0	

307511 rows × 122 columns

```python
pd.set_option('display.max_columns', None)
df_1
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_
0	100002	1	Cash loans	M	N	Y	0	202500.0	
1	100003	0	Cash loans	F	N	N	0	270000.0	1
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	
4	100007	0	Cash loans	M	N	Y	0	121500.0	
...	...	...	...	...	...	...	...	...	
307506	456251	0	Cash loans	M	N	N	0	157500.0	
307507	456252	0	Cash loans	F	N	Y	0	72000.0	
307508	456253	0	Cash loans	F	N	Y	0	153000.0	
307509	456254	1	Cash loans	F	N	Y	0	171000.0	
307510	456255	0	Cash loans	F	N	N	0	157500.0	

307511 rows × 122 columns

```
df_2=pd.read_csv('D:\\data ANALYTICS AND SCIENCE\\TRAINITY ASSIGN\\previous_application.csv')
df_2
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	17145.0
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	607500.0
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	112500.0
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	450000.0
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	337500.0
...	...	...	...	...	...	...	...	...
1670209	2300464	352015	Consumer loans	14704.290	267295.5	311400.0	0.0	267295.5
1670210	2357031	334635	Consumer loans	6622.020	87750.0	64291.5	29250.0	87750.0
1670211	2659632	249544	Consumer loans	11520.855	105237.0	102523.5	10525.5	105237.0
1670212	2785582	400317	Cash loans	18821.520	180000.0	191880.0	NaN	180000.0
1670213	2418762	261212	Cash loans	16431.300	360000.0	360000.0	NaN	360000.0

1670214 rows × 37 columns

```python
pd.set_option('display.max_columns', None)
df_2
```

2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	112500.
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	450000.
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	337500.
...	...	...	...	...	...	...	...	.
1670209	2300464	352015	Consumer loans	14704.290	267295.5	311400.0	0.0	267295.
1670210	2357031	334635	Consumer loans	6622.020	87750.0	64291.5	29250.0	87750.
1670211	2659632	249544	Consumer loans	11520.855	105237.0	102523.5	10525.5	105237.
1670212	2785582	400317	Cash loans	18821.520	180000.0	191880.0	NaN	180000.
1670213	2418762	261212	Cash loans	16431.300	360000.0	360000.0	NaN	360000.

1670214 rows × 37 columns

```
#converting column names to a list for easy data understanding
column_names = list(df_1.columns)
print(column_names)
```

```
['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TO
TAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMIL
Y_STATUS', 'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBL
ISH', 'OWN_CAR_AGE', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL', 'OCCUPA
TION_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_C
ITY', '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', 'ENT
RANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_A
VG', 'NONLIVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMONAR
EA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LI
VINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATAT
ION_MEDI', 'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LAND
AREA_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_DOC
UMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_1
6', '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', 'AM
T_REQ_CREDIT_BUREAU_YEAR']
```

```
column_names = list(df_2.columns)
print(column_names)
```

```
['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY', 'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOO
DS_PRICE', 'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START', 'FLAG_LAST_APPL_PER_CONTRACT', 'NFLAG_LAST_APPL_IN_DAY', 'R
ATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY', 'RATE_INTEREST_PRIVILEGED', 'NAME_CASH_LOAN_PURPOSE', 'NAME_CONTRACT_STATUS', 'DAYS
_DECISION', 'NAME_PAYMENT_TYPE', 'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE', 'NAME_GOODS_CATEGORY', 'NAME_PORT
FOLIO', 'NAME_PRODUCT_TYPE', 'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY', 'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PR
ODUCT_COMBINATION', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DUE', 'DAYS_TERMINATION',
'NFLAG_INSURED_ON_APPROVAL']
```

cleaning data in application csv

```
# transforming the missing data into percentage
100* df_1.isnull().sum() / len(df_1)
```

```
SK_ID_CURR                     0.000000
TARGET                         0.000000
NAME_CONTRACT_TYPE             0.000000
CODE_GENDER                    0.000000
FLAG_OWN_CAR                   0.000000
                                  ...
AMT_REQ_CREDIT_BUREAU_DAY     13.501631
AMT_REQ_CREDIT_BUREAU_WEEK    13.501631
AMT_REQ_CREDIT_BUREAU_MON     13.501631
AMT_REQ_CREDIT_BUREAU_QRT     13.501631
AMT_REQ_CREDIT_BUREAU_YEAR    13.501631
Length: 122, dtype: float64
```

```python
# sorting data in terms of % missing data
def percent_missing(df_1):
    percent_nan = 100* df_1.isnull().sum() / len(df_1)
    percent_nan = percent_nan[percent_nan>0].sort_values()
    return percent_nan
percent_nan = percent_missing(df_1)
percent_nan.tail(30)
```

```
ELEVATORS_AVG               53.295980
ELEVATORS_MEDI              53.295980
ELEVATORS_MODE              53.295980
NONLIVINGAREA_MEDI          55.179164
NONLIVINGAREA_MODE          55.179164
NONLIVINGAREA_AVG           55.179164
EXT_SOURCE_1                56.381073
BASEMENTAREA_MEDI           58.515956
BASEMENTAREA_AVG            58.515956
BASEMENTAREA_MODE           58.515956
LANDAREA_MODE               59.376738
LANDAREA_MEDI               59.376738
LANDAREA_AVG                59.376738
OWN_CAR_AGE                 65.990810
YEARS_BUILD_MODE            66.497784
YEARS_BUILD_MEDI            66.497784
YEARS_BUILD_AVG             66.497784
FLOORSMIN_MODE              67.848630
FLOORSMIN_MEDI              67.848630
FLOORSMIN_AVG               67.848630
LIVINGAPARTMENTS_MODE       68.354953
LIVINGAPARTMENTS_MEDI       68.354953
LIVINGAPARTMENTS_AVG        68.354953
FONDKAPREMONT_MODE          68.386172
NONLIVINGAPARTMENTS_MEDI    69.432963
NONLIVINGAPARTMENTS_MODE    69.432963
NONLIVINGAPARTMENTS_AVG     69.432963
COMMONAREA_MODE             69.872297
COMMONAREA_AVG              69.872297
COMMONAREA_MEDI             69.872297
dtype: float64
```

**since no columns are having percent missing value gretaer than 90 % , I dont need to drop any column

```
df_1.head(4)
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREI
0	100002	1	Cash loans	M	N	Y	0	202500.0	40659
1	100003	0	Cash loans	F	N	N	0	270000.0	129350
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	13500
3	100006	0	Cash loans	F	N	Y	0	135000.0	31268

```
#converting all numeric columns NaN value to 0
numeric_col_fill = ['SK_ID_CURR', 'TARGET' , 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
df_1[numeric_col_fill] = df_1[numeric_col_fill].fillna(0)
df_1[numeric_col_fill]
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE	D.
0	100002	1	0	202500.0	406597.5	24700.5	351000.0	0.018801	
1	100003	0	0	270000.0	1293502.5	35698.5	1129500.0	0.003541	
2	100004	0	0	67500.0	135000.0	6750.0	135000.0	0.010032	
3	100006	0	0	135000.0	312682.5	29686.5	297000.0	0.008019	
4	100007	0	0	121500.0	513000.0	21865.5	513000.0	0.028663	
...	...	...	...	...	...	...	...	...	
307506	456251	0	0	157500.0	254700.0	27558.0	225000.0	0.032561	
307507	456252	0	0	72000.0	269550.0	12001.5	225000.0	0.025164	
307508	456253	0	0	153000.0	677664.0	29979.0	585000.0	0.005002	
307509	456254	1	0	171000.0	370107.0	20205.0	319500.0	0.005313	
307510	456255	0	0	157500.0	675000.0	49117.5	675000.0	0.046220	

307511 rows × 106 columns

```python
# sorting data in terms of % missing data
def percent_missing(df_1):
    percent_nan = 100* df_1.isnull().sum() / len(df_1)
    percent_nan = percent_nan[percent_nan>0].sort_values()
    return percent_nan
percent_nan = percent_missing(df_1)
percent_nan
```

```
NAME_TYPE_SUITE         0.420148
OCCUPATION_TYPE        31.345545
EMERGENCYSTATE_MODE    47.398304
HOUSETYPE_MODE         50.176091
WALLSMATERIAL_MODE     50.840783
FONDKAPREMONT_MODE     68.386172
dtype: float64
```

**all the numeric column missing values are converted to 0 .only string columns are left with Nan values.replace those by "none" string in place of NaN

```
#converting all string columns NaN or null value to "NONE" string
string_col_fill = ['NAME_CONTRACT_TYPE','CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY','NAME_TYPE_SUITE','NAME_INCOME_TYPE','NA
df_1[string_col_fill] = df_1[string_col_fill].fillna('None')
df_1[string_col_fill]
```

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

307511 rows × 16 columns

```
# sorting data in terms of % missing data
def percent_missing(df_1):
    percent_nan = 100* df_1.isnull().sum() / len(df_1)
    percent_nan = percent_nan[percent_nan>0].sort_values()
    return percent_nan
percent_nan = percent_missing(df_1)
percent_nan
#all NaN are removed
```

```
Series([], dtype: float64)
```

outliers in application csv

```
#chceking for outliers in application data
ser = pd.Series(df_1['AMT_INCOME_TOTAL'])
print(ser.describe())
```

```
count    3.075110e+05
mean     1.687979e+05
std      2.371231e+05
min      2.565000e+04
25%      1.125000e+05
50%      1.471500e+05
75%      2.025000e+05
max      1.170000e+08
Name: AMT_INCOME_TOTAL, dtype: float64
```

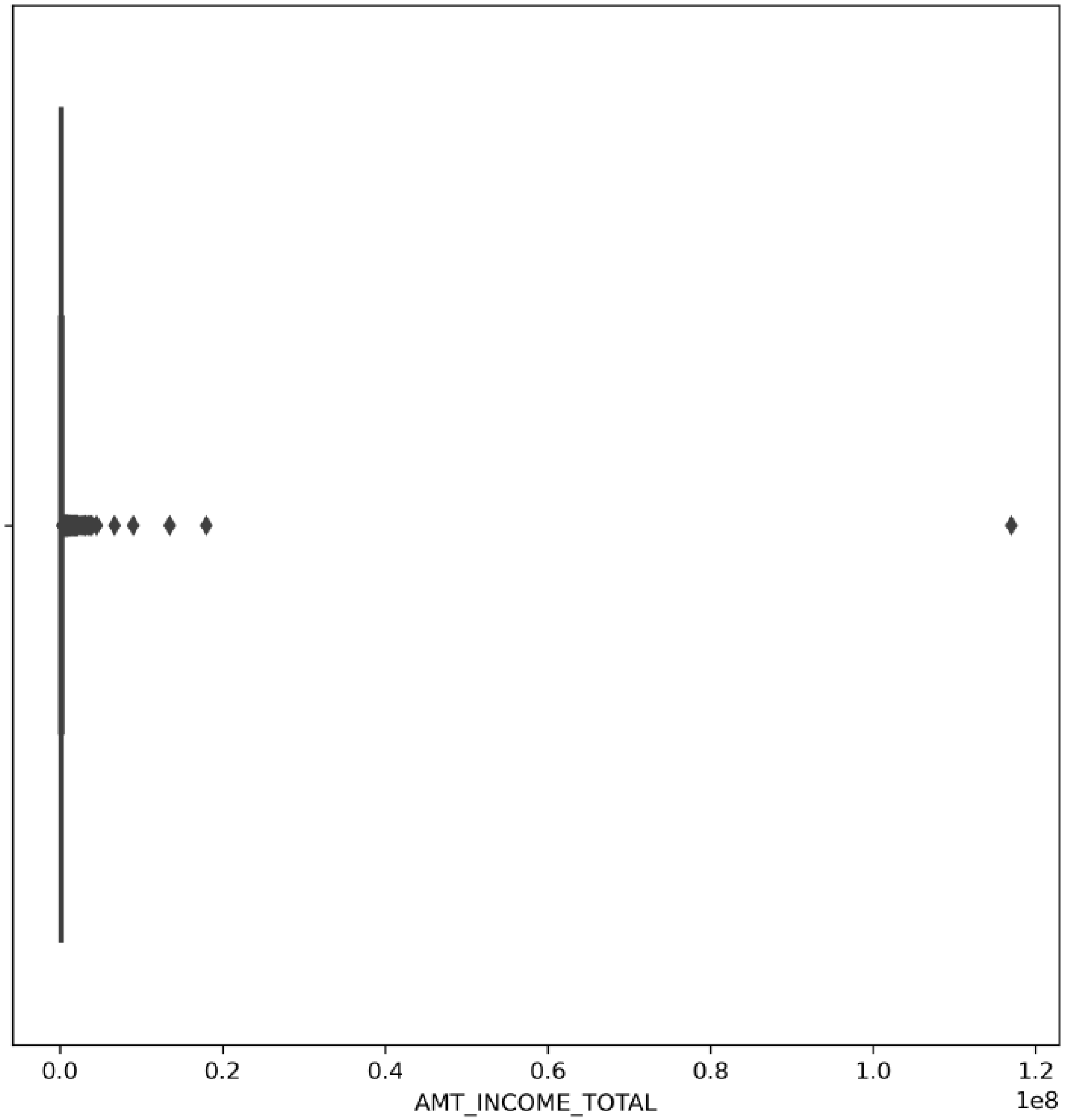
```
IQR=2.025000e+05-1.125000e+05
lower_limit=1.125000e+05-1.5*IQR
lower_limit
```

```
-22500.0
```

```
upper_limit= 2.025000e+05+1.5*IQR
upper_limit
```

```
337500.0
```

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8,8),dpi=300)
sns.boxplot(x='AMT_INCOME_TOTAL',data=df_1,orient='h')
plt.show()
```



**in boxplot we can see there is one point which is above upper limit.but since this a loan case study , greater the income value better it is for bank in terms of risk perspective.

```
ser = pd.Series(df_1['AMT_CREDIT'])
print(ser.describe())
```

```
count    3.075110e+05
mean     5.990260e+05
std      4.024908e+05
min      4.500000e+04
25%      2.700000e+05
50%      5.135310e+05
75%      8.086500e+05
max      4.050000e+06
Name: AMT_CREDIT, dtype: float64
```

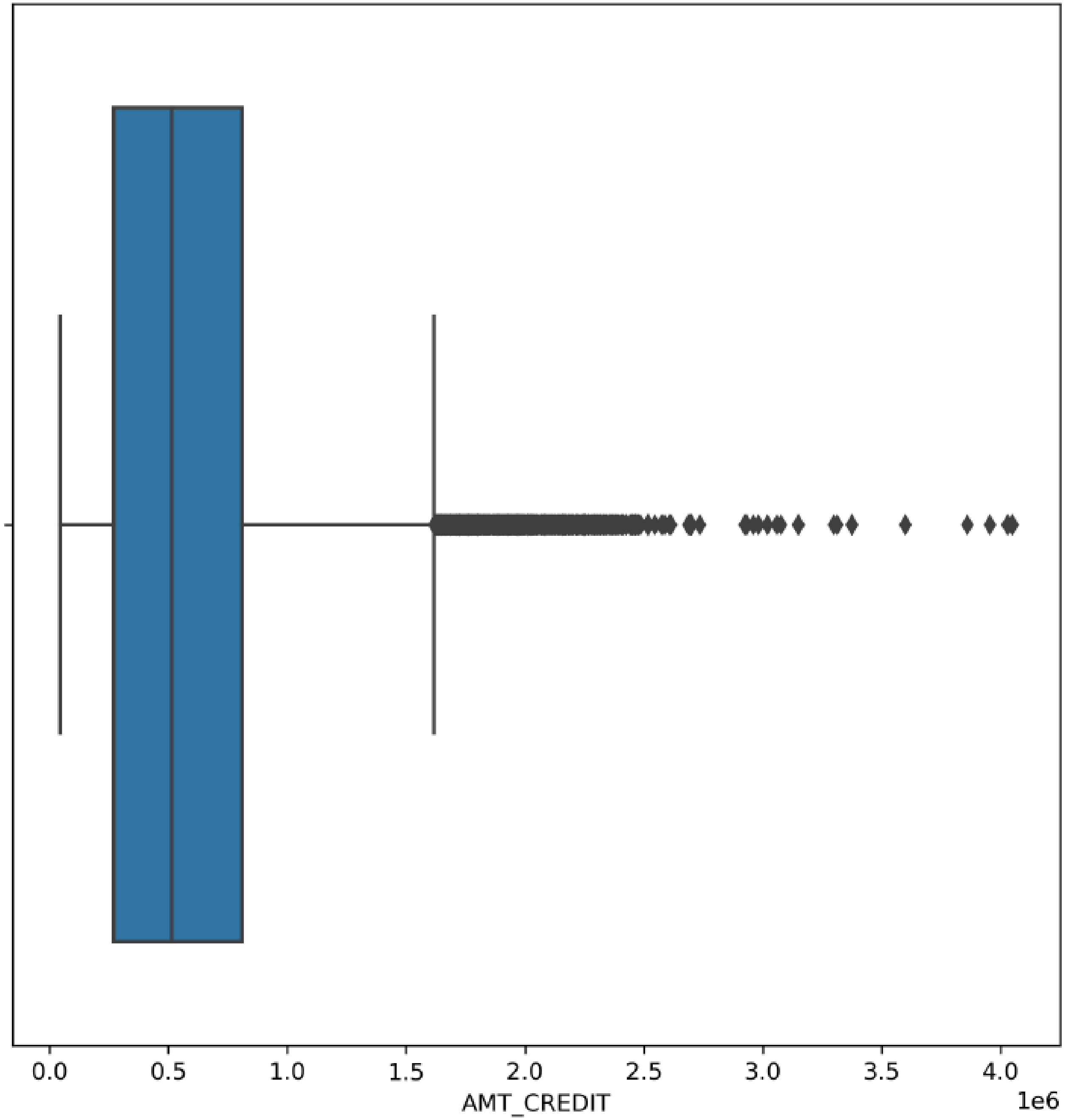
```
IQR=8.086500e+05-2.700000e+05
lower_limit=2.700000e+05-1.5*IQR
lower_limit
```

```
-537975.0
```

```
upper_limit=8.086500e+05+1.5*IQR
upper_limit
```

```
1616625.0
```

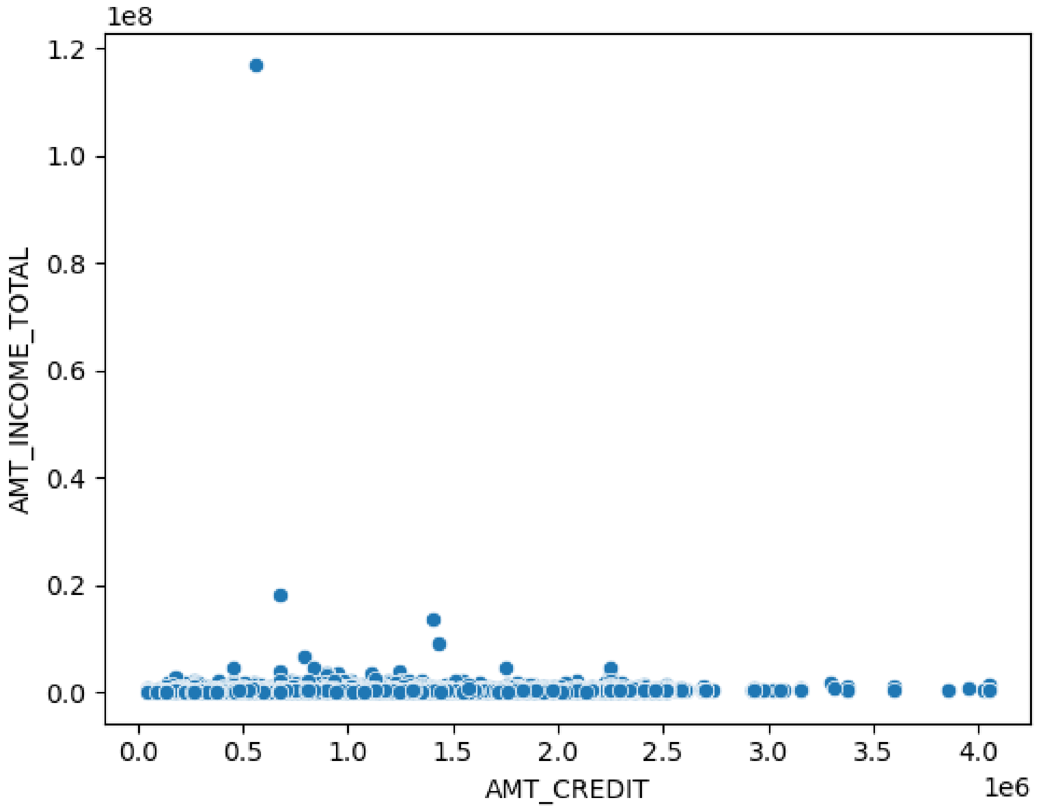
```python
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8,8),dpi=300)
sns.boxplot(x='AMT_CREDIT',data=df_1,orient='h')
plt.show()
```



```
ser[ser >upper_limit]
```

```
60        1663987.5
135       1755000.0
189       2250000.0
235       1710000.0
314       1800000.0
            ...
307216    1827549.0
307252    1724220.0
307401    1718473.5
307422    1971072.0
307476    1762110.0
Name: AMT_CREDIT, Length: 6562, dtype: float64
```

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

# Select only the columns of interest
df_subset = df_1[['AMT_INCOME_TOTAL', 'AMT_CREDIT']]

# Create the scatter plot
sns.scatterplot(data=df_subset, x='AMT_CREDIT', y='AMT_INCOME_TOTAL')

# Show the plot
plt.show()
```

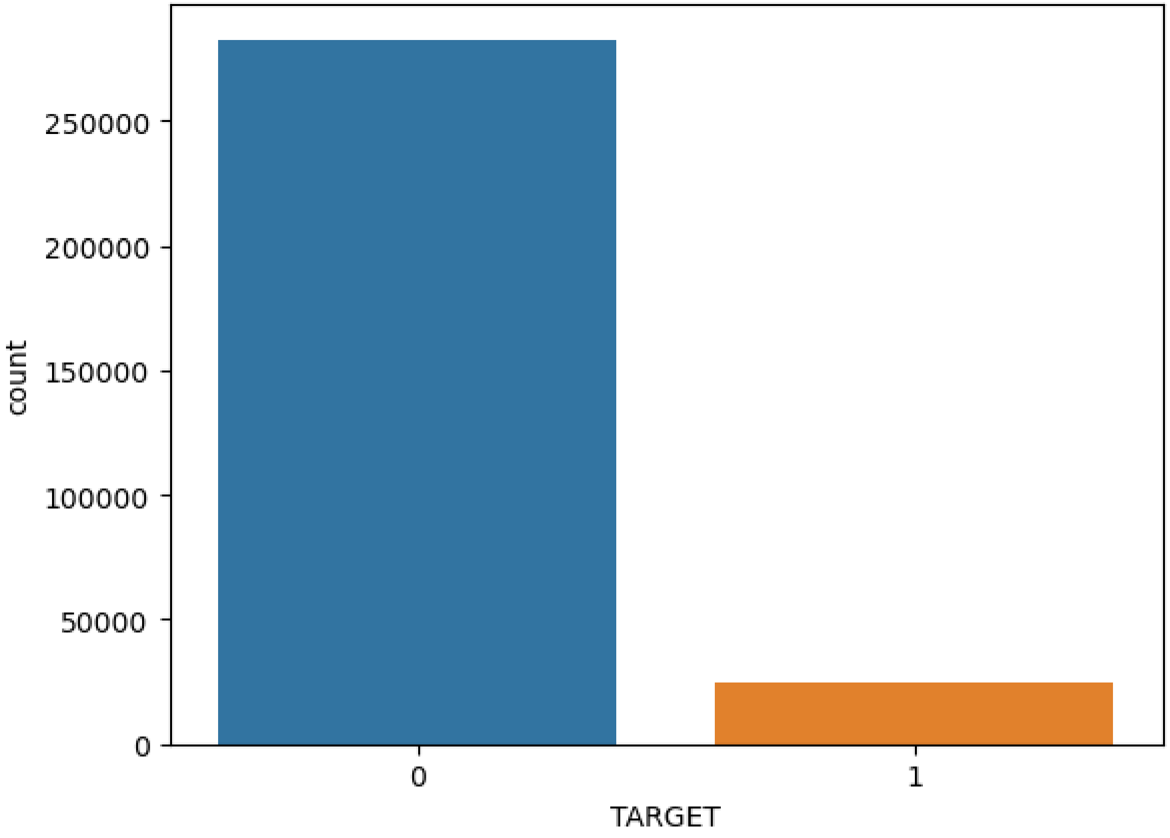


**on reading both box plot and scatter plot we can say , with low income level amount credit > 2.5 * 10 E6 will increase the risk of default.

Identify if there is data imbalance in the data. Find the ratio of data imbalance

```
#showing clear data imbalance
import seaborn as sns

sns.countplot(x='TARGET', data=df_1)
```

```
<AxesSubplot:xlabel='TARGET', ylabel='count'>
```



```
ratio = df_1['TARGET'].value_counts()[1] / df_1['TARGET'].value_counts()[0]
print("Ratio of data imbalance: ", ratio)
```

```
Ratio of data imbalance:  0.08781828601345662
```

**Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample, 0 - all other cases).The target variable represents whether a client has faced payment difficulties on their loan or not. If a client has had a late payment for more than X days on at least one of the first Y installments of the loan, then their target value is 1, which means they have payment difficulties. Otherwise, their target value is 0, which means they do not have payment difficulties. A data imbalance ratio of 0.08781828601345662 indicates that the proportion of records in the dataset with a positive class label (clients with payment difficulties) is significantly smaller than the proportion of records with a negative class label (all other cases). This is an indication of class imbalance in the data, which can pose a challenge for predictive modeling, as the model may be biased towards the majority class and have poor performance on the minority class. It is important to address this imbalance in the data by using techniques such as oversampling, undersampling, or class weighting to ensure that the model is trained on a balanced dataset.

cleaning data in previous_application csv

```python
# transforming the missing data into percentage
100* df_2.isnull().sum() / len(df_2)
```

```
SK_ID_PREV                      0.000000
SK_ID_CURR                      0.000000
NAME_CONTRACT_TYPE              0.000000
AMT_ANNUITY                    22.286665
AMT_APPLICATION                 0.000000
AMT_CREDIT                      0.000060
AMT_DOWN_PAYMENT               53.636480
AMT_GOODS_PRICE                23.081773
WEEKDAY_APPR_PROCESS_START      0.000000
HOUR_APPR_PROCESS_START         0.000000
FLAG_LAST_APPL_PER_CONTRACT     0.000000
NFLAG_LAST_APPL_IN_DAY          0.000000
RATE_DOWN_PAYMENT              53.636480
RATE_INTEREST_PRIMARY          99.643698
RATE_INTEREST_PRIVILEGED       99.643698
NAME_CASH_LOAN_PURPOSE          0.000000
NAME_CONTRACT_STATUS            0.000000
DAYS_DECISION                   0.000000
NAME_PAYMENT_TYPE               0.000000
CODE_REJECT_REASON              0.000000
NAME_TYPE_SUITE                49.119754
NAME_CLIENT_TYPE                0.000000
NAME_GOODS_CATEGORY             0.000000
NAME_PORTFOLIO                  0.000000
NAME_PRODUCT_TYPE               0.000000
CHANNEL_TYPE                    0.000000
SELLERPLACE_AREA                0.000000
NAME_SELLER_INDUSTRY            0.000000
CNT_PAYMENT                    22.286366
NAME_YIELD_GROUP                0.000000
PRODUCT_COMBINATION             0.020716
DAYS_FIRST_DRAWING             40.298129
DAYS_FIRST_DUE                 40.298129
DAYS_LAST_DUE_1ST_VERSION      40.298129
DAYS_LAST_DUE                  40.298129
DAYS_TERMINATION               40.298129
NFLAG_INSURED_ON_APPROVAL      40.298129
dtype: float64
```

```python
# sorting data in terms of % missing data
def percent_missing(df_2):
    percent_nan = 100* df_2.isnull().sum() / len(df_2)
    percent_nan = percent_nan[percent_nan>0].sort_values()
    return percent_nan
percent_nan = percent_missing(df_2)
percent_nan
```

```
AMT_CREDIT                     0.000060
PRODUCT_COMBINATION            0.020716
CNT_PAYMENT                   22.286366
AMT_ANNUITY                   22.286665
AMT_GOODS_PRICE               23.081773
DAYS_FIRST_DRAWING            40.298129
DAYS_FIRST_DUE                40.298129
DAYS_LAST_DUE_1ST_VERSION     40.298129
DAYS_LAST_DUE                 40.298129
DAYS_TERMINATION              40.298129
NFLAG_INSURED_ON_APPROVAL     40.298129
NAME_TYPE_SUITE               49.119754
AMT_DOWN_PAYMENT              53.636480
RATE_DOWN_PAYMENT             53.636480
RATE_INTEREST_PRIMARY         99.643698
RATE_INTEREST_PRIVILEGED      99.643698
dtype: float64
```

**we can drop features like RATE_INTEREST_PRIMARY , RATE_INTEREST_PRIVILEGED which have 99% null values

```python
df_2 = df_2.drop(['RATE_INTEREST_PRIMARY', 'RATE_INTEREST_PRIVILEGED'], axis=1)
df_2
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	17145.0
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	607500.0
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	112500.0
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	450000.0
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	337500.0
...	...	...	...	...	...	...	...	...
1670209	2300464	352015	Consumer loans	14704.290	267295.5	311400.0	0.0	267295.5
1670210	2357031	334635	Consumer loans	6622.020	87750.0	64291.5	29250.0	87750.0
1670211	2659632	249544	Consumer loans	11520.855	105237.0	102523.5	10525.5	105237.0
1670212	2785582	400317	Cash loans	18821.520	180000.0	191880.0	NaN	180000.0
1670213	2418762	261212	Cash loans	16431.300	360000.0	360000.0	NaN	360000.0

1670214 rows × 35 columns

**all the numeric column missing values will be converted to 0

```
#converting all numeric columns NaN value to 0
numeric_col_fill_2 = ['AMT_CREDIT','AMT_ANNUITY','AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE', 'RATE_DOWN_PAYMENT','CNT_PAYMENT','DAYS_F
df_2[numeric_col_fill_2] = df_2[numeric_col_fill_2].fillna(0)
df_2[numeric_col_fill_2]
```

	AMT_CREDIT	AMT_ANNUITY	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	RATE_DOWN_PAYMENT	CNT_PAYMENT	DAYS_FIRST_DRAWING	DAYS_FIR
0	17145.0	1730.430	0.0	17145.0	0.000000	12.0	365243.0	
1	679671.0	25188.615	0.0	607500.0	0.000000	36.0	365243.0	
2	136444.5	15060.735	0.0	112500.0	0.000000	12.0	365243.0	
3	470790.0	47041.335	0.0	450000.0	0.000000	12.0	365243.0	
4	404055.0	31924.395	0.0	337500.0	0.000000	24.0	0.0	
...	...	...	...	...	...	...	...	
1670209	311400.0	14704.290	0.0	267295.5	0.000000	30.0	365243.0	
1670210	64291.5	6622.020	29250.0	87750.0	0.340554	12.0	365243.0	
1670211	102523.5	11520.855	10525.5	105237.0	0.101401	10.0	365243.0	
1670212	191880.0	18821.520	0.0	180000.0	0.000000	12.0	365243.0	
1670213	360000.0	16431.300	0.0	360000.0	0.000000	48.0	365243.0	

1670214 rows × 12 columns

**all string column missing values will be converted "None"

```python
#converting all string columns NaN or null value to "NONE" string
string_col_fill_2 = ['NAME_TYPE_SUITE']
df_2[string_col_fill_2] = df_2[string_col_fill_2].fillna('None')
df_2[string_col_fill_2]
```

	NAME_TYPE_SUITE
0	None
1	Unaccompanied
2	Spouse, partner
3	None
4	None
...	...
1670209	None
1670210	Unaccompanied
1670211	Spouse, partner
1670212	Family
1670213	Family

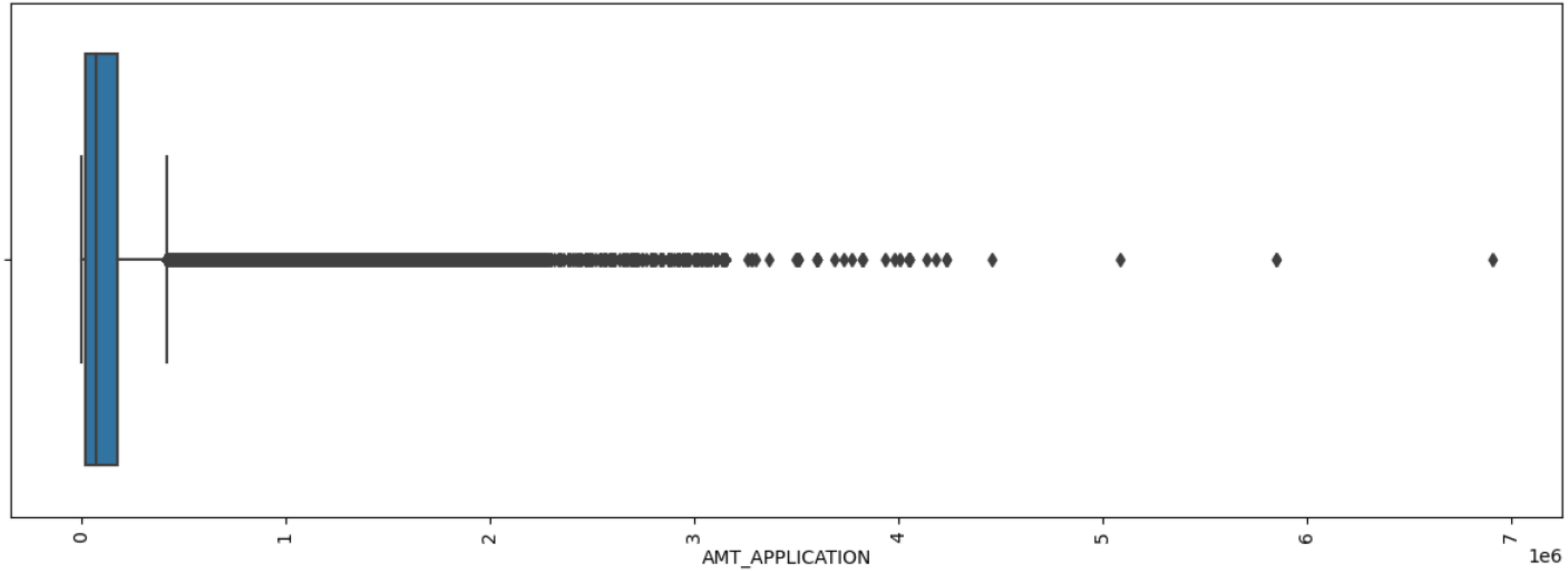
1670214 rows × 1 columns

**all ata of previous application csv cleaned

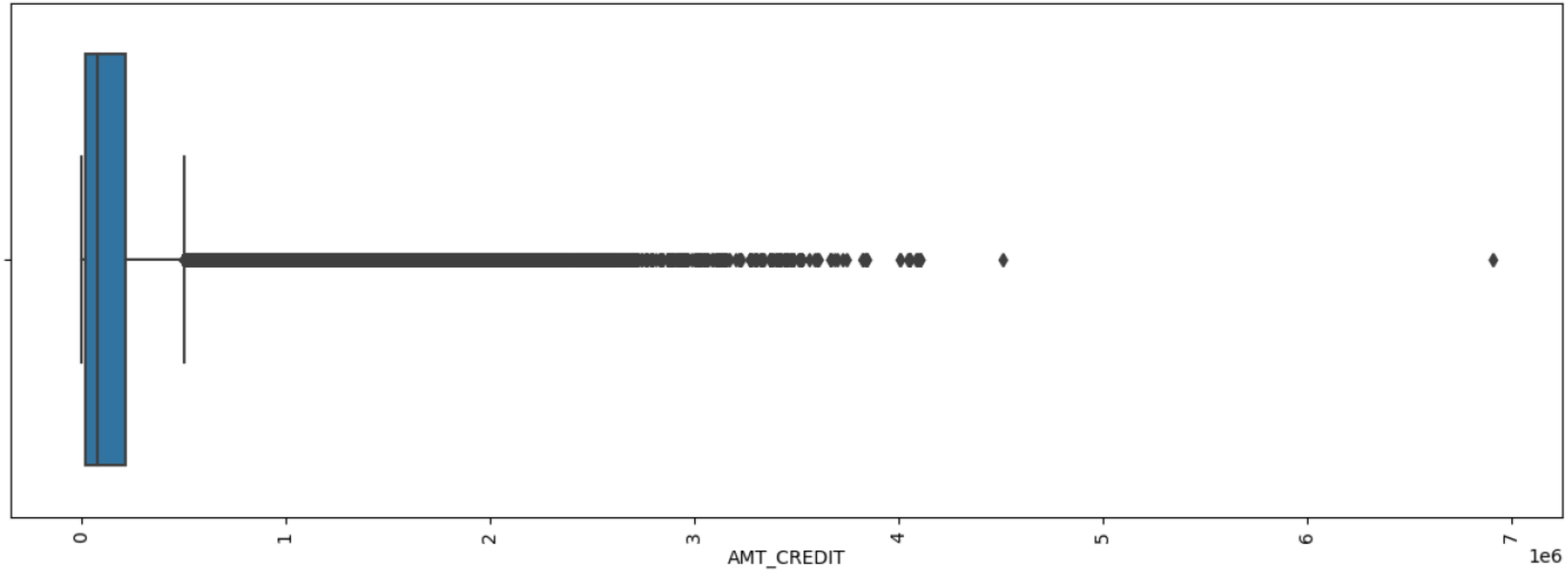
outliers in previous application csv###

```
prev_box = ['AMT_APPLICATION', 'AMT_CREDIT', 'AMT_GOODS_PRICE','CNT_PAYMENT']
for i in df_2[prev_box]:
    plt.figure(1,figsize=(15,5))
    sns.boxplot(df_2[i])
    plt.xticks(rotation = 90,fontsize =10)
    plt.show()
```

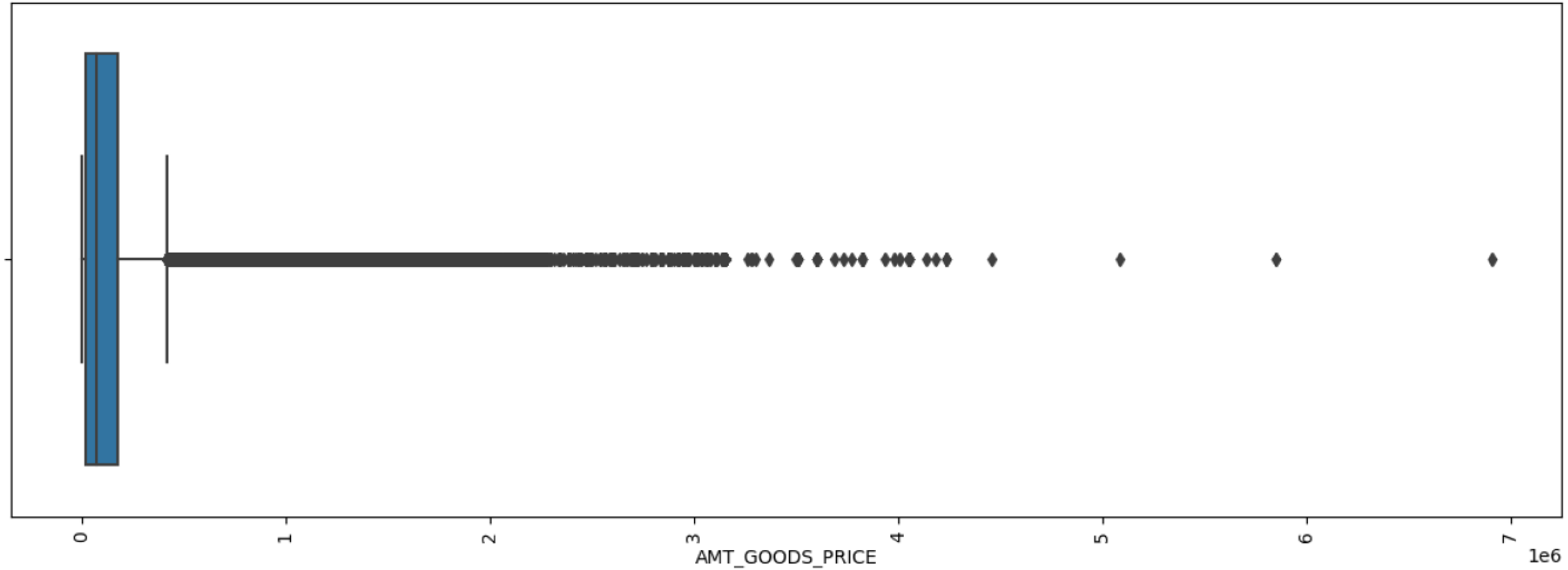
```
C:\Users\Sayak23\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



```
C:\Users\Sayak23\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

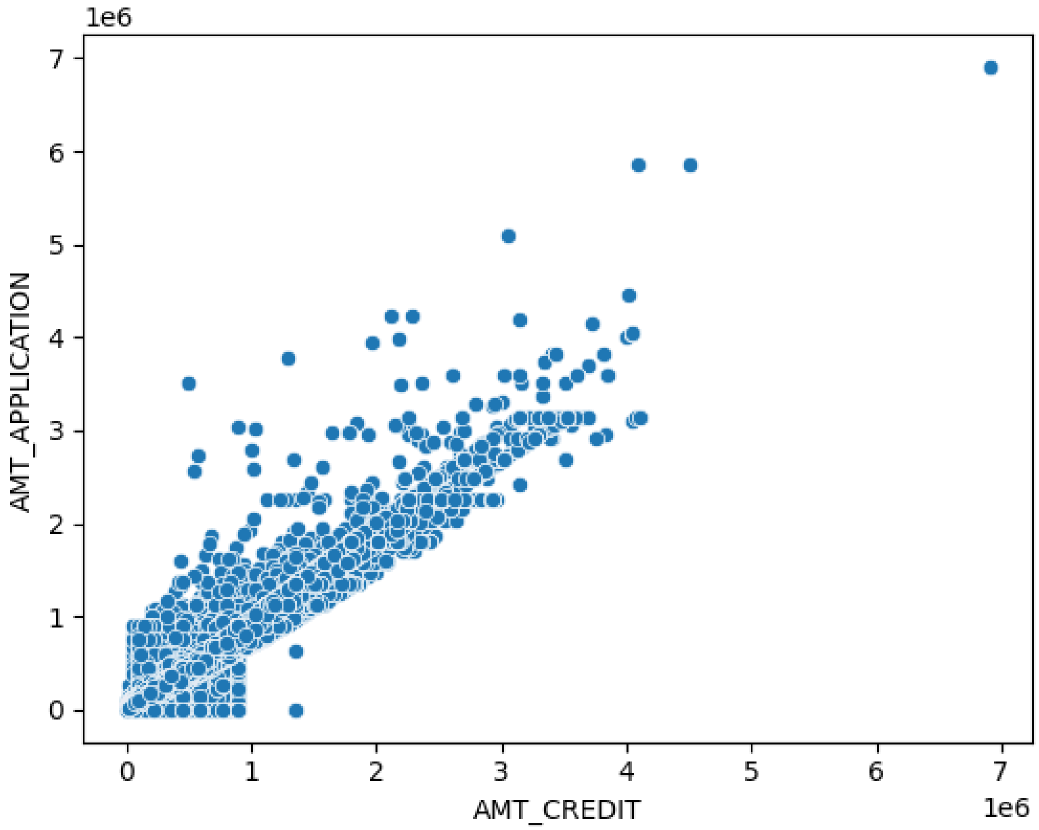


```
C:\Users\Sayak23\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



```
C:\Users\Sayak23\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Select only the columns of interest
df_subset = df_2[['AMT_APPLICATION', 'AMT_CREDIT']]

# Create the scatter plot
sns.scatterplot(data=df_subset, x='AMT_CREDIT', y='AMT_APPLICATION')

# Show the plot
plt.show()
```

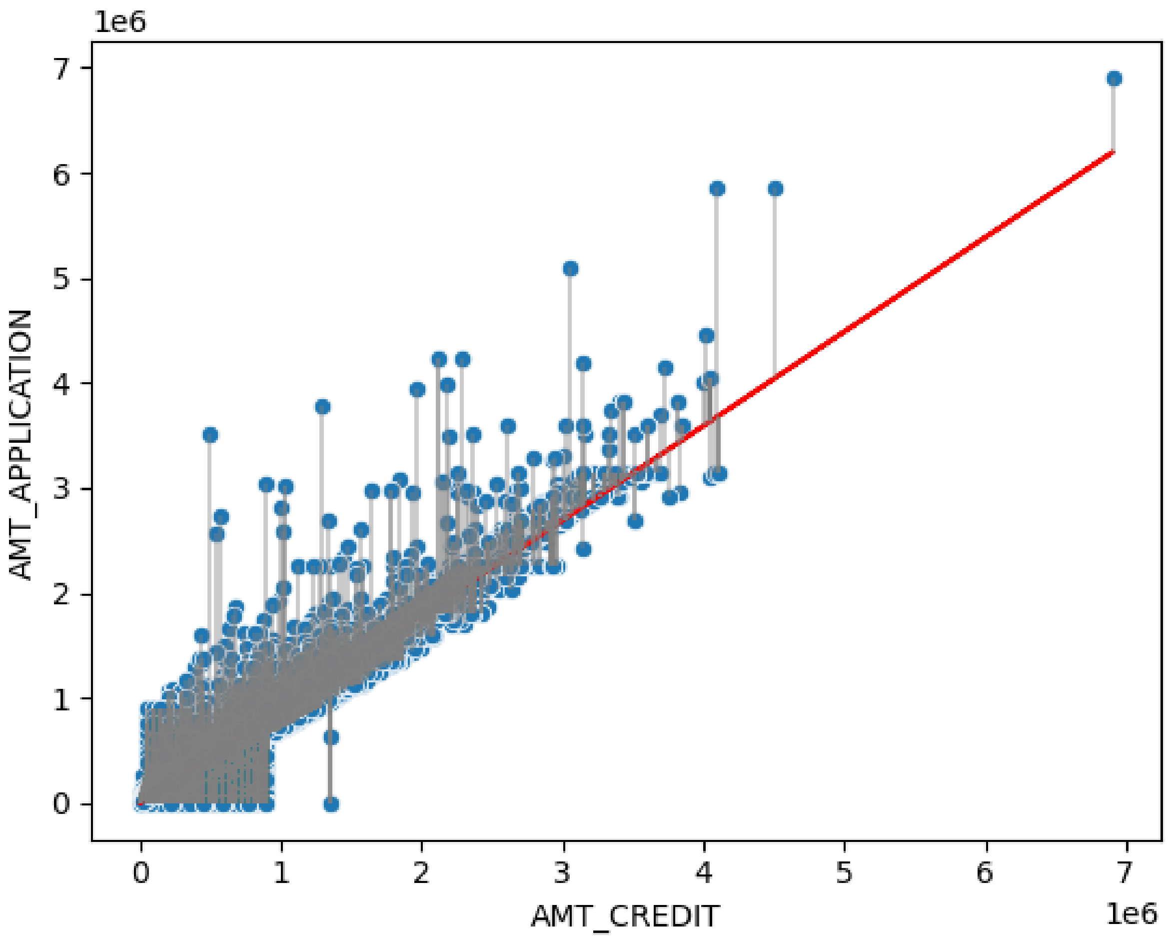


**on checking the amount which was applied and amount which was credited we can get a clear picture for our future loan risk except for some special cases which may arise due to uncontrolled factors.

```
df_subset.isna().sum()
```

```
AMT_APPLICATION    0
AMT_CREDIT         0
dtype: int64
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression

# Select only the columns of interest
N= df_subset[['AMT_APPLICATION', 'AMT_CREDIT']]

# Create the scatter plot
sns.scatterplot(data=N, x='AMT_CREDIT', y='AMT_APPLICATION')

# Fit a linear regression model to the data
X = df_subset['AMT_CREDIT'].values.reshape(-1,1)
y = df_subset['AMT_APPLICATION'].values.reshape(-1,1)
reg = LinearRegression().fit(X, y)

# Add the trend line to the plot
plt.plot(X, reg.predict(X), color='red')

# Calculate and plot the residuals
y_pred = reg.predict(X)
residuals = y - y_pred
plt.vlines(X, y_pred, y, color='gray', alpha=0.4)

# Show the plot
plt.show()
```



**The red trend line represents the linear regression model fit to the data, showing the overall trend of the relationship between AMT_CREDIT and AMT_APPLICATION. The gray vertical lines represent the residuals, or the difference between the predicted values and the actual values of AMT_APPLICATION. These lines show how far each data point deviates from the trend line, giving an indication of how well the linear regression model fits the data. Here we can see a narrow spread of residuals which suggests a good fit.we can predict any future value based on the coefficients of this linear regresssion graph.we can also check errors so that we can minimise it.

```python
from sklearn.metrics import mean_absolute_error

# Fit a linear regression model to the data
X = df_subset['AMT_CREDIT'].values.reshape(-1,1)
y = df_subset['AMT_APPLICATION'].values.reshape(-1,1)
reg = LinearRegression().fit(X, y)

# Predict the values for X
y_pred = reg.predict(X)

# Calculate the mean absolute error
mae = mean_absolute_error(y, y_pred)

print('Mean absolute error:', mae)
```

```
Mean absolute error: 22838.33344974068
```

**we can check the performnace by using other model. I can perform elastic net regression AND Random Forest as a part of supervised learning.

```python
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Select the columns of interest
df_subset = df_2[['AMT_APPLICATION', 'AMT_CREDIT']]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df_subset[['AMT_CREDIT']], df_subset[['AMT_APPLICATION']], test_size=0.3, rar

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and fit the Elastic Net regression model
elastic_net = ElasticNet(alpha=0.5, l1_ratio=0.5)
elastic_net.fit(X_train, y_train)

# Make predictions on the test set and calculate the mean absolute error
y_pred = elastic_net.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)

print("Mean Absolute Error:", mae)
```

```
Mean Absolute Error: 45354.142475194654
```

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

# Select the columns of interest
X = df_subset.drop('AMT_APPLICATION', axis=1)
y = df_subset['AMT_APPLICATION']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the Random Forest regressor object
rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model on the training data
rf.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = rf.predict(X_test)

# Calculate the mean absolute error
mae = mean_absolute_error(y_test, y_pred)

# Print the MAE
print("Random Forest MAE:", mae)
```

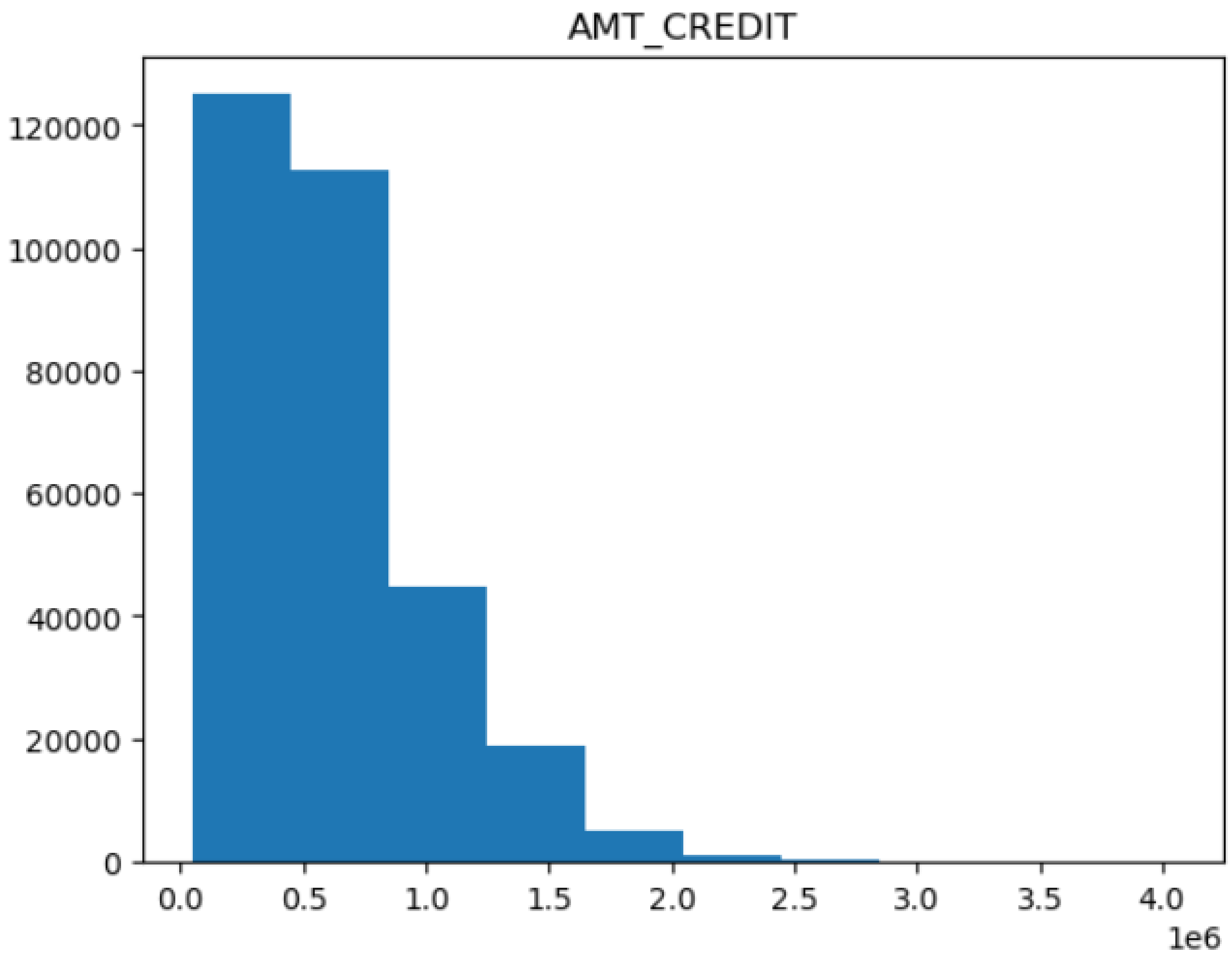
```
Random Forest MAE: 16663.66178272552
```

**therefore we can see random forest is performing better since MAE is coming less

UNIVARIATE ANALYSIS

```python
import pandas as pd
# Univariate Analysis
for column in ['AMT_CREDIT']:
    print(column)
    print('Mean: ', df_1['AMT_CREDIT'].mean())
    print('Median: ', df_1['AMT_CREDIT'].median())
    print('Mode: ', df_1['AMT_CREDIT'].mode())
    print('Standard Deviation: ', df_1['AMT_CREDIT'].std())
    print('Minimum: ', df_1['AMT_CREDIT'].min())
    print('Maximum: ', df_1['AMT_CREDIT'].max())
    print('25th percentile: ', df_1['AMT_CREDIT'].quantile(0.25))
    print('50th percentile: ', df_1['AMT_CREDIT'].quantile(0.5))
    print('75th percentile: ', df_1['AMT_CREDIT'].quantile(0.75))
    print('')

    # Create histogram
    df_1.hist(column='AMT_CREDIT', bins=10, grid=False)
```

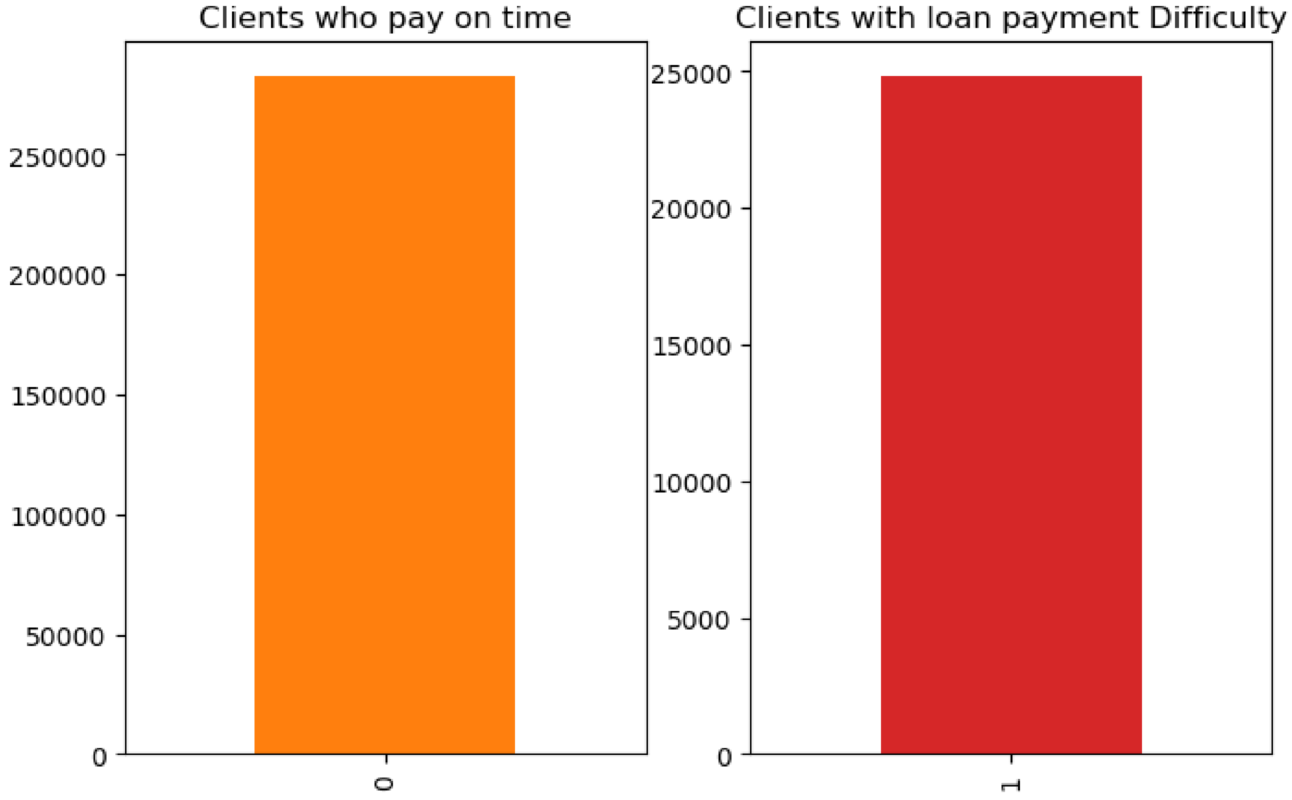
```
AMT_CREDIT
Mean:  599025.9997057016
Median:  513531.0
Mode:  0    450000.0
Name: AMT_CREDIT, dtype: float64
Standard Deviation:  402490.776995946
Minimum:  45000.0
Maximum:  4050000.0
25th percentile:  270000.0
50th percentile:  513531.0
75th percentile:  808650.0
```



**THIS is a left-skewed distribution or a left-tailed distribution.This means that there are more data points on the left side of the distribution.In a left-skewed distribution, the mean is typically less than the median.

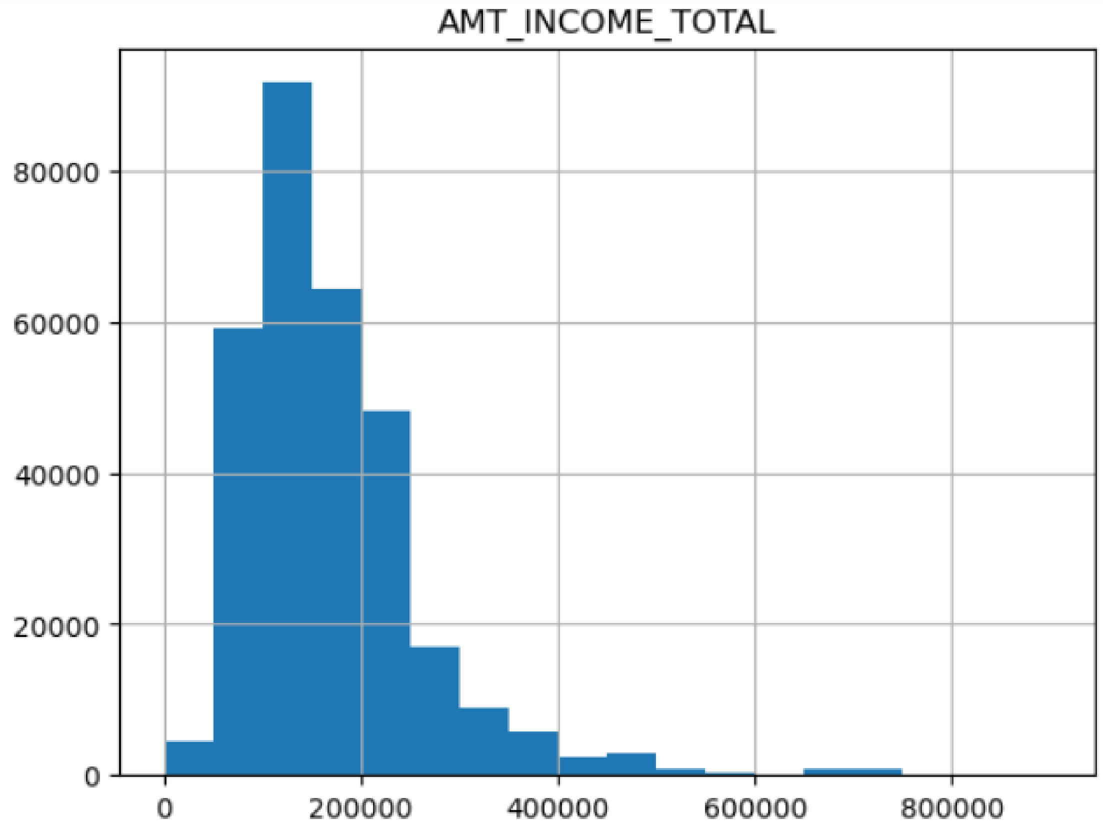
```
# SAVing THE DATASET FOR TARGET VALUES EQUAL TO 0 AND 1 IN SEPERATE VARIABLES
zero = df_1[df_1['TARGET'].isin([0])]
one = df_1[df_1['TARGET'].isin([1])]

plt.figure(figsize=(8,5))
plt.subplot(121); zero['TARGET'].value_counts().plot(kind='bar', color = ['C1']); plt.title("Clients who pay on time")
plt.subplot(122); one['TARGET'].value_counts().plot(kind='bar', color = ['C3']); plt.title("Clients with loan payment Difficulty'
```

```
Text(0.5, 1.0, 'Clients with loan payment Difficulty')
```



**The code is creating two new dataframes 'zero' and 'one' containing the rows from the original dataframe 'df_1' where the value in the 'TARGET' column is 0 and 1, respectively. Then, it is creating a bar plot using matplotlib to display the count of clients who pay on time and clients with loan payment difficulty separately, using the 'TARGET' column of each of these dataframes. The subplot function is used to display both the plots side by side. The first subplot displays the count of clients who pay on time and the second subplot displays the count of clients with loan payment difficulty.

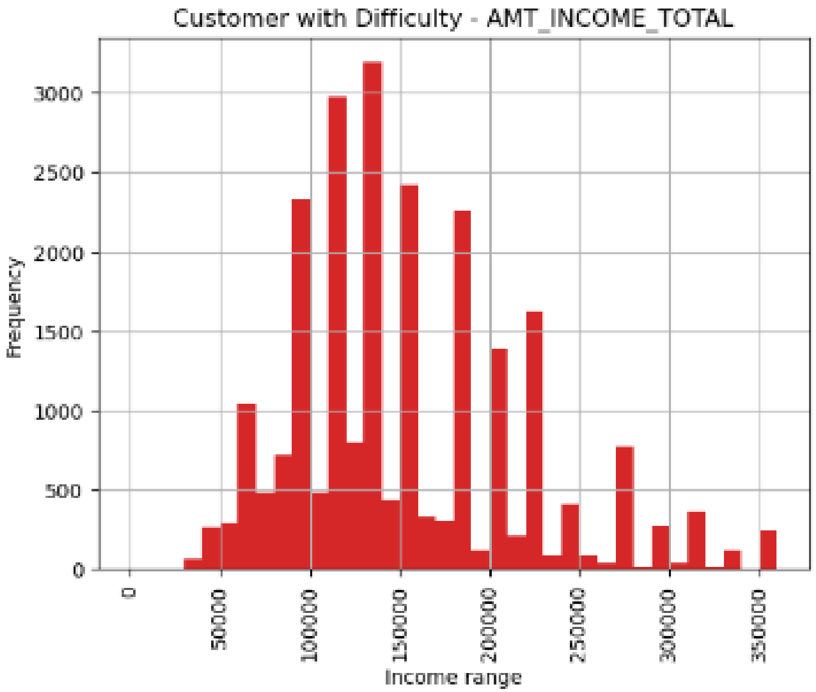
```
# SHOW THE DISTRIBUTION OF AMT_INCOME_TOTAL
amount_Income = df_1[['AMT_INCOME_TOTAL']]

# DEFINE THE BINS
bins = [0, 50000, 100000, 150000,200000, 250000, 300000, 350000, 400000, 450000, 500000, 550000,600000,650000,750000,800000,8500

# PLOT A HISTOGRAM TO SEE THE DISTRIBUTION OF INCOME
amount_Income.hist(bins= bins, range=[2.565000e+04,1.170000e+08])

plt.show()
amount_Income.describe()
```

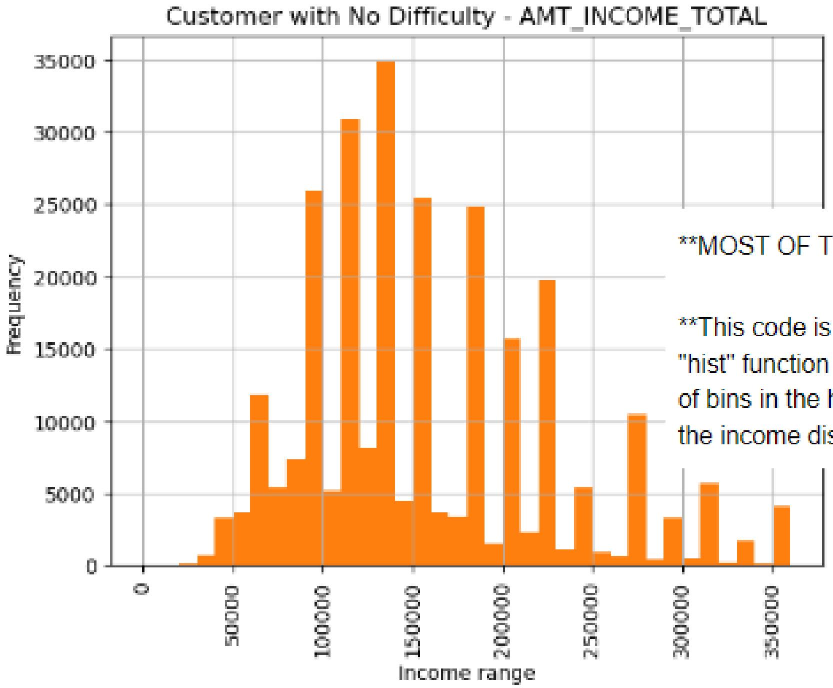


	AMT_INCOME_TOTAL
count	3.075110e+05
mean	1.687979e+05
std	2.371231e+05
min	2.565000e+04
25%	1.125000e+05
50%	1.471500e+05
75%	2.025000e+05
max	1.170000e+08

Segmented univariate analysis

```
# 3.A. UNIVARIATE ANALYSIS - AMT_INCOME_TOTAL
AMT_INCOME_TOTAL_one=one[['AMT_INCOME_TOTAL']]
AMT_INCOME_TOTAL_zero = zero[['AMT_INCOME_TOTAL']]

min = AMT_INCOME_TOTAL_one.describe().min()
max = AMT_INCOME_TOTAL_one.describe().max()
range1=[min['AMT_INCOME_TOTAL'], max['AMT_INCOME_TOTAL']]
bins = [0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000, 110000, 120000, 130000, 140000, 150000, 160000,
AMT_INCOME_TOTAL_one.hist(bins=bins, range=range1, color = ['C3'])
plt.title("Customer with Difficulty - AMT_INCOME_TOTAL")
plt.xlabel("Income range")
plt.ylabel("Frequency")
plt.xticks(rotation = 90, fontsize=10)

min = AMT_INCOME_TOTAL_zero.describe().min()
max = AMT_INCOME_TOTAL_zero.describe().max()
range2=[min['AMT_INCOME_TOTAL'], max['AMT_INCOME_TOTAL']]
AMT_INCOME_TOTAL_zero.hist(bins=bins, range=range2, color = ['C1'])
plt.title("Customer with No Difficulty - AMT_INCOME_TOTAL")
plt.xlabel("Income range")
plt.ylabel("Frequency")
plt.xticks(rotation = 90, fontsize=10)

plt.show()
```

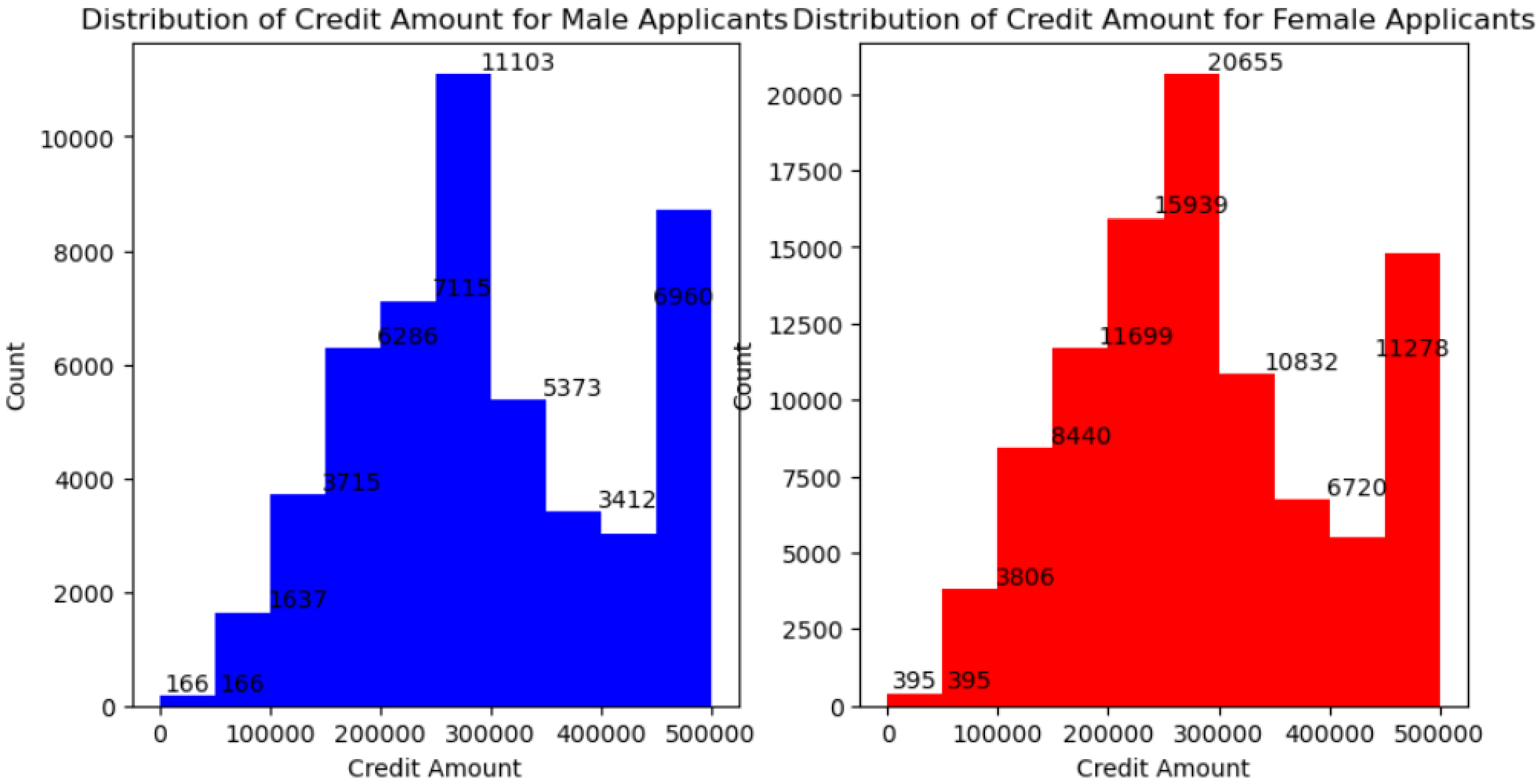




**MOST OF THE LOAN DEFAULTS IS FOR CLIENTS WHOSE INCOME IS BETWEEN 80000 TO 190000

**This code is plotting separate histograms for the income distribution of loan applicants who have defaulted on their loans and those who have not, using the "hist" function in matplotlib. The histograms will show the count of loan applicants within a particular income range. The "bins" argument specifies the number of bins in the histogram and the "color" argument specifies the color of the histogram bars. The resulting plots can help identify any significant differences in the income distribution of loan applicants between those who have defaulted on their loans and those who have not.

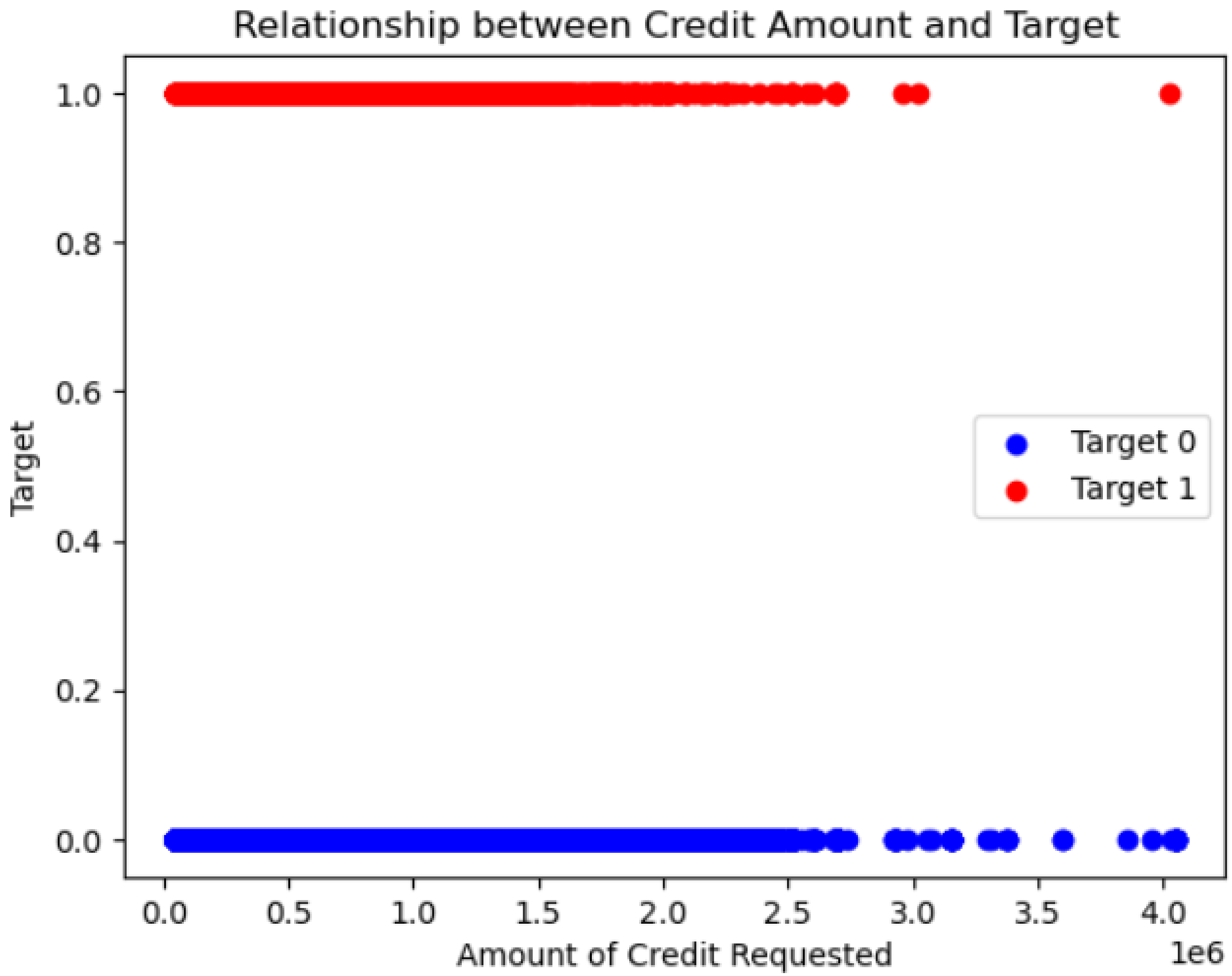
```python
# MALE vs FEMALE applicant
male_applicants = df_1[df_1['CODE_GENDER'] == 'M']
female_applicants = df_1[df_1['CODE_GENDER'] == 'F']
bins = [0, 50000, 100000, 150000, 200000, 250000, 300000, 350000, 400000, 450000, 500000]
range_male = [male_applicants['AMT_CREDIT'].min(), male_applicants['AMT_CREDIT'].max()]
range_female = [female_applicants['AMT_CREDIT'].min(), female_applicants['AMT_CREDIT'].max()]

plt.figure(figsize=(10,5))
plt.subplot(121)
plt.hist(male_applicants['AMT_CREDIT'], bins=bins, range=range_male, color='blue')
plt.xlabel('Credit Amount')
plt.ylabel('Count')
plt.title('Distribution of Credit Amount for Male Applicants')

for i in range(len(bins)-1):
    plt.text((bins[i]+bins[i+1])/2, male_applicants['AMT_CREDIT'].value_counts(bins=bins)[bins[i]], male_applicants['AMT_CREDIT']

plt.subplot(122)
plt.hist(female_applicants['AMT_CREDIT'], bins=bins, range=range_female, color='red')
plt.xlabel('Credit Amount')
plt.ylabel('Count')
plt.title('Distribution of Credit Amount for Female Applicants')

for i in range(len(bins)-1):
    plt.text((bins[i]+bins[i+1])/2, female_applicants['AMT_CREDIT'].value_counts(bins=bins)[bins[i]], female_applicants['AMT_CRED

plt.show()
```




```
**The above code will give us two histograms side-by-side, one showing the distribution of credit amounts for male applicants
and the other showing the distribution for female applicants. We can use this to visually compare the credit amounts requested
by men and women and see if there are any significant differences.
```

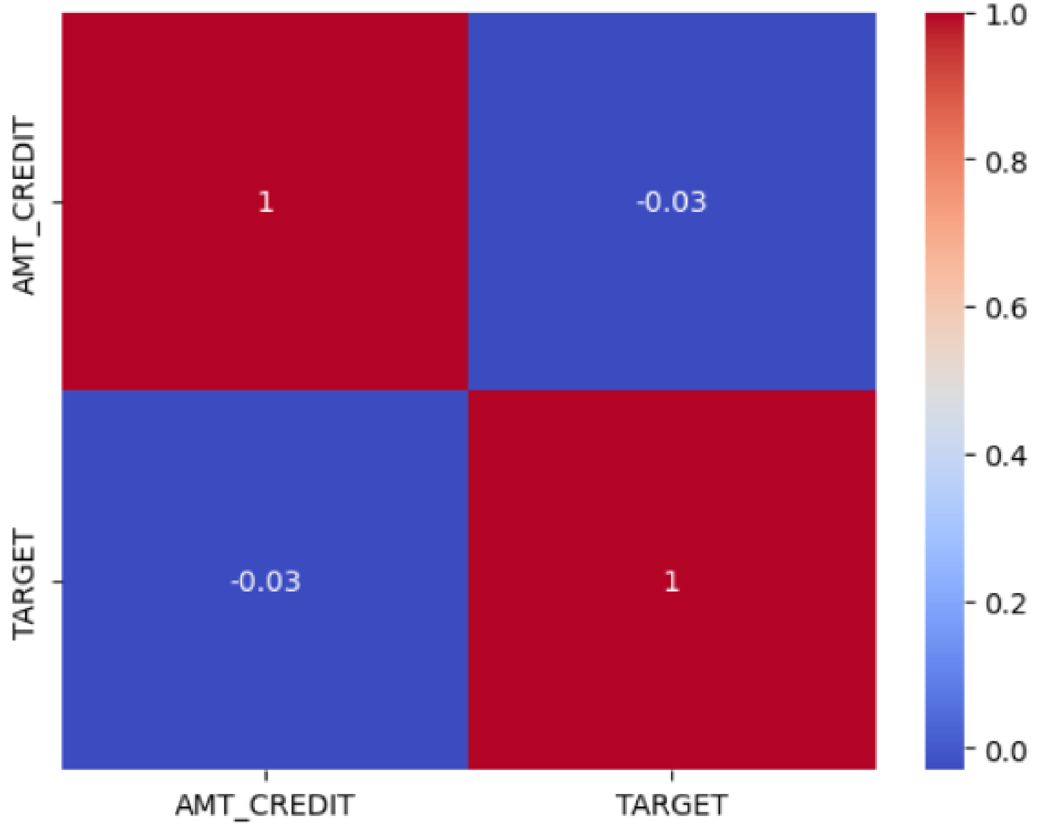
bivariate analysis

```
# credit vs target
import matplotlib.pyplot as plt

# Get data for target = 0
target_0 = df_1[df_1['TARGET'] == 0]
x_0 = target_0['AMT_CREDIT']
y_0 = target_0['TARGET']

# Get data for target = 1
target_1 = df_1[df_1['TARGET'] == 1]
x_1 = target_1['AMT_CREDIT']
y_1 = target_1['TARGET']

# Create scatter plot
plt.scatter(x_0, y_0, color='blue', label='Target 0')
plt.scatter(x_1, y_1, color='red', label='Target 1')
plt.xlabel('Amount of Credit Requested')
plt.ylabel('Target')
plt.title('Relationship between Credit Amount and Target')
plt.legend()
plt.show()
```



**By visualizing the graph we can gain insights into how credit amount affects the likelihood of a loan being repaid.The target variable is represented by different colors, where blue dots represent loans that were repaid on time (target=0) and orange dots represent loans that were not repaid on time (target=1).There are more blue dots (repaid loans) than orange dots (defaulted loans) across most credit amounts.This suggests that the majority of loans are repaid on time regardless of the credit amount.However, we can also see that as the credit amount increases,the proportion of orange dots (defaulted loans)DECREASES as well.This indicates that as the credit amount gets larger,there is a LOWER risk of default.In conclusion, this graph highlights the relationship between credit amount and loan repayment status.

```python
import seaborn as sns

sns.heatmap(df_1[['AMT_CREDIT', 'TARGET']].corr(), annot=True, cmap='coolwarm')
plt.show()
```




```
**we are seeing a heatmap that shows the correlation between credit amount and loan repayment.
**In the heatmap, the value of 1 represents a perfect positive correlation between AMT_CREDIT and TARGET. This means that as the credit amount increases, the likelihood of a loan being repaid also increases. On the other hand, the value of -0.03 indicates a weak negative correlation between the two variables. This means that there is a small negative relationship between the credit amount and the likelihood of a loan being repaid. However, this negative relationship is not strong enough to draw any significant conclusions.Overall, the heat map suggests that there is a positive correlation between the credit amount and the likelihood of a loan being repaid, but this relationship is not strong enough to be considered significant. Other factors, such as income and employment status, may also play a role in determining the likelihood of loan repayment.
```

```python
#top 10 correlation for the Client with payment difficulties and all other cases (Target variable)
# Segregate data frames based on 'TARGET' variable
df_payment_difficulty = df_1[df_1['TARGET'] == 1]
df_no_payment_difficulty = df_1[df_1['TARGET'] == 0]

# Calculate correlation matrix for clients with payment difficulties
corr_payment_difficulty = df_payment_difficulty.corr()

# Identify top 10 correlations for clients with payment difficulties
top_corr_payment_difficulty = corr_payment_difficulty.unstack().sort_values(ascending=False).drop_duplicates()[1:11]

# Calculate correlation matrix for clients without payment difficulties
corr_no_payment_difficulty = df_no_payment_difficulty.corr()

# Identify top 10 correlations for clients without payment difficulties
top_corr_no_payment_difficulty = corr_no_payment_difficulty.unstack().sort_values(ascending=False).drop_duplicates()[1:11]

# Analyze insights from the top correlations
print('Top correlations for clients with payment difficulties:')
print(top_corr_payment_difficulty)
print('\n')
print('Top correlations for clients without payment difficulties:')
print(top_corr_no_payment_difficulty)
```

```
Top correlations for clients with payment difficulties:
YEARS_BEGINEXPLUATATION_MEDI  YEARS_BEGINEXPLUATATION_AVG     0.999964
YEARS_BUILD_MEDI              YEARS_BUILD_AVG                 0.999939
YEARS_BEGINEXPLUATATION_MODE  YEARS_BEGINEXPLUATATION_AVG     0.999792
                              YEARS_BEGINEXPLUATATION_MEDI    0.999774
YEARS_BUILD_MODE              YEARS_BUILD_MEDI                0.999676
                              YEARS_BUILD_AVG                 0.999632
FLOORSMIN_MEDI                FLOORSMIN_AVG                   0.999119
BASEMENTAREA_AVG              BASEMENTAREA_MEDI               0.999011
FLOORSMAX_MEDI                FLOORSMAX_AVG                   0.998769
LIVINGAPARTMENTS_AVG          LIVINGAPARTMENTS_MEDI           0.998711
dtype: float64
```

```
Top correlations for clients without payment difficulties:
YEARS_BEGINEXPLUATATION_MEDI  YEARS_BEGINEXPLUATATION_AVG     0.999952
YEARS_BUILD_MEDI              YEARS_BUILD_AVG                 0.999948
YEARS_BEGINEXPLUATATION_MODE  YEARS_BEGINEXPLUATATION_AVG     0.999736
YEARS_BEGINEXPLUATATION_MEDI  YEARS_BEGINEXPLUATATION_MODE    0.999667
YEARS_BUILD_MEDI              YEARS_BUILD_MODE                0.999661
YEARS_BUILD_AVG               YEARS_BUILD_MODE                0.999643
FLOORSMIN_MEDI                FLOORSMIN_AVG                   0.998826
FLOORSMAX_MEDI                FLOORSMAX_AVG                   0.998646
ENTRANCES_AVG                 ENTRANCES_MEDI                  0.998526
OBS_60_CNT_SOCIAL_CIRCLE      OBS_30_CNT_SOCIAL_CIRCLE        0.998510
dtype: float64
```

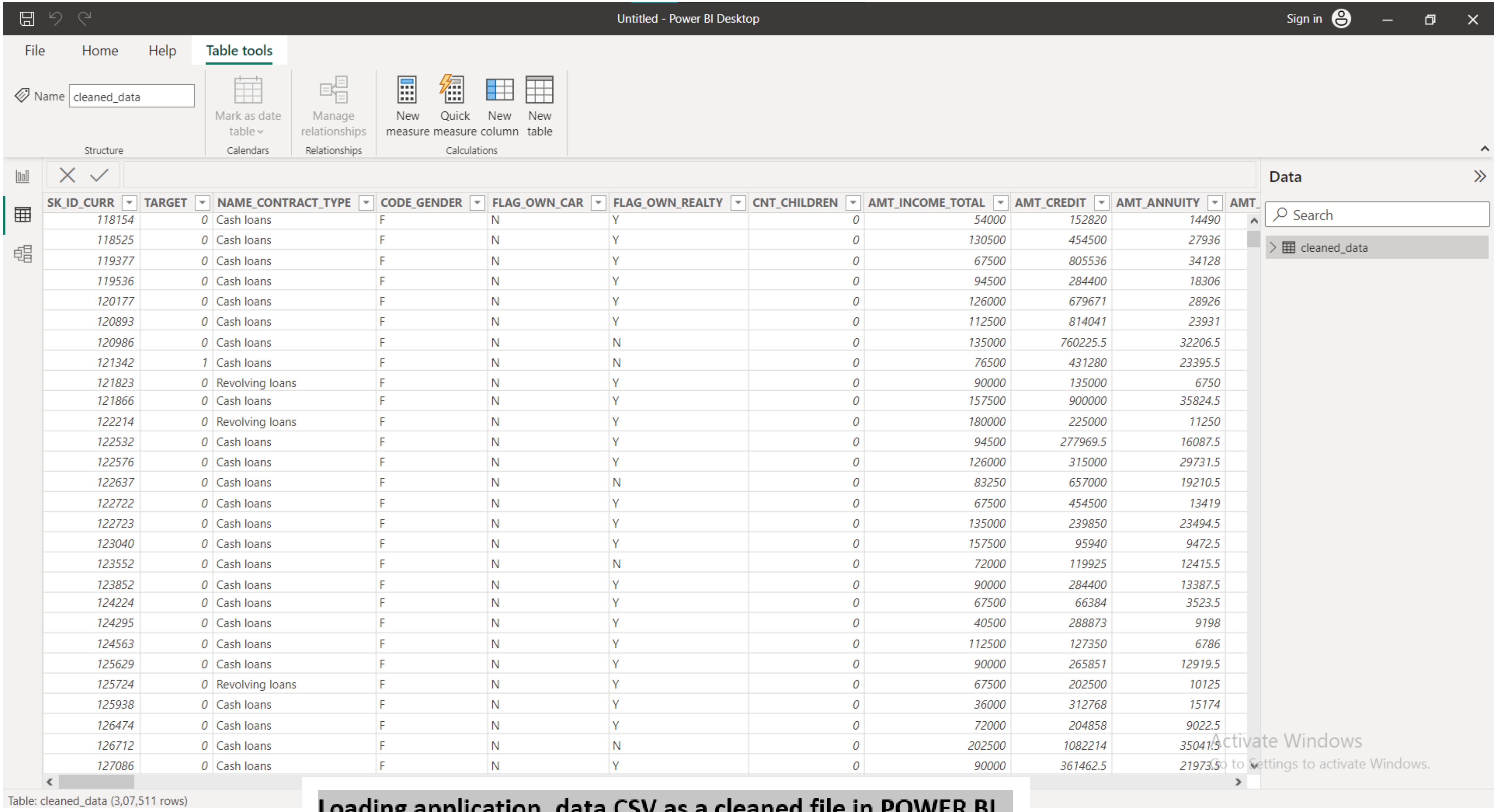
**The variables with the highest correlations are related to the age and construction of the property, followed by variables related to floors, living space, and basement area. It is interesting to note that the variables related to social circles have a high correlation for clients without payment difficulties, but not for those with payment difficulties.

```python
import pandas as pd

# Assume that 'df' is your cleaned dataframe
cleaned_df = df_1

# Saving the dataframe to a CSV file called 'cleaned_data.csv'
cleaned_df.to_csv('cleaned_data.csv', index=False)
```

```python
import os
print(os.getcwd())
```

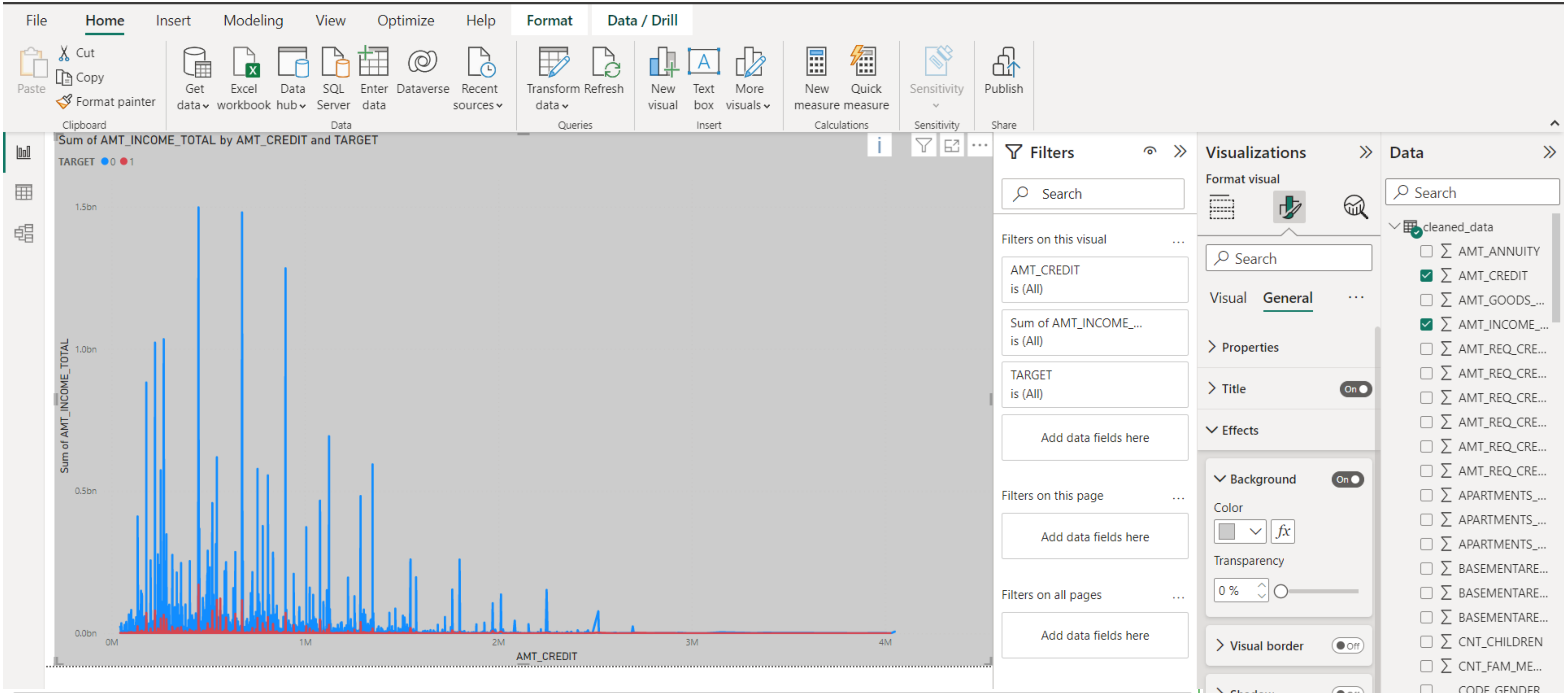
```
C:\Users\Sayak23
```

**now i will install this data in power bi for data visualisation

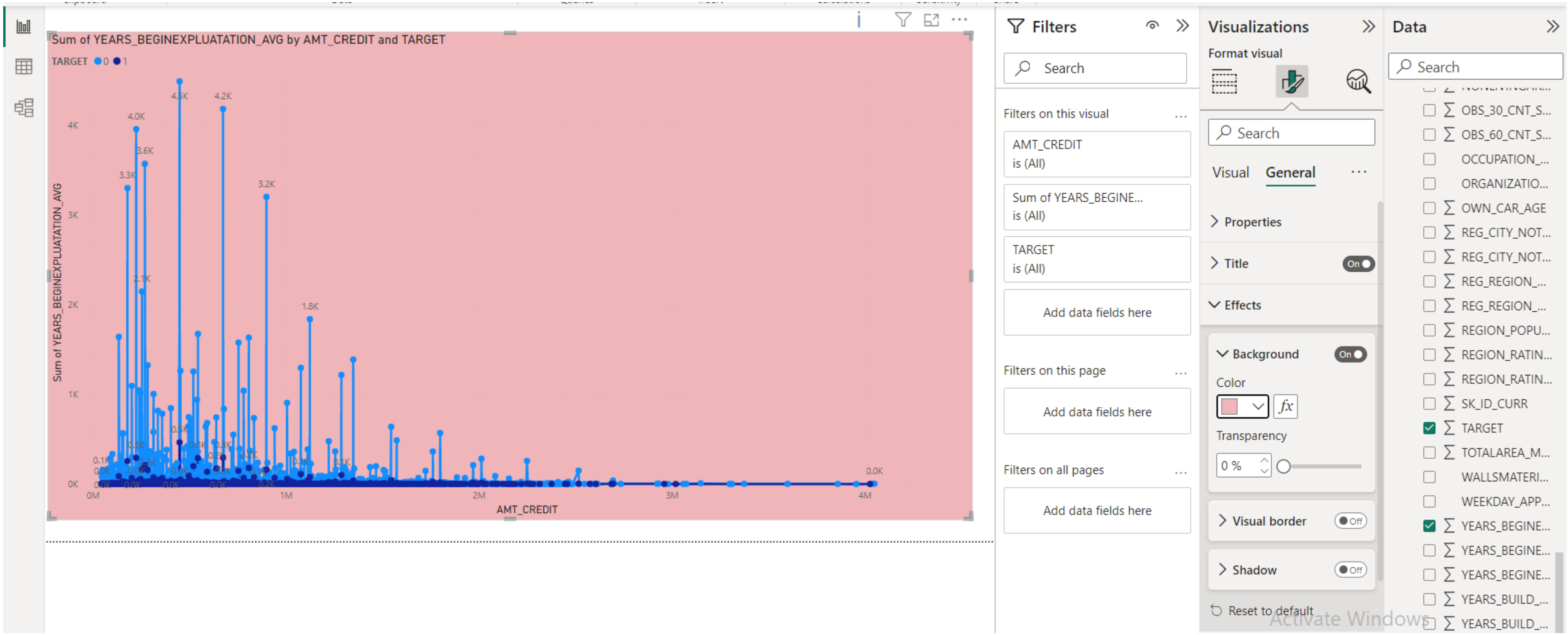


SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY
118154	0	Cash loans	F	N	Y	0	54000	152820	14490
118525	0	Cash loans	F	N	Y	0	130500	454500	27936
119377	0	Cash loans	F	N	Y	0	67500	805536	34128
119536	0	Cash loans	F	N	Y	0	94500	284400	18306
120177	0	Cash loans	F	N	Y	0	126000	679671	28926
120893	0	Cash loans	F	N	Y	0	112500	814041	23931
120986	0	Cash loans	F	N	N	0	135000	760225.5	32206.5
121342	1	Cash loans	F	N	N	0	76500	431280	23395.5
121823	0	Revolving loans	F	N	Y	0	90000	135000	6750
121866	0	Cash loans	F	N	Y	0	157500	900000	35824.5
122214	0	Revolving loans	F	N	Y	0	180000	225000	11250
122532	0	Cash loans	F	N	Y	0	94500	277969.5	16087.5
122576	0	Cash loans	F	N	Y	0	126000	315000	29731.5
122637	0	Cash loans	F	N	N	0	83250	657000	19210.5
122722	0	Cash loans	F	N	Y	0	67500	454500	13419
122723	0	Cash loans	F	N	Y	0	135000	239850	23494.5
123040	0	Cash loans	F	N	Y	0	157500	95940	9472.5
123552	0	Cash loans	F	N	N	0	72000	119925	12415.5
123852	0	Cash loans	F	N	Y	0	90000	284400	13387.5
124224	0	Cash loans	F	N	Y	0	67500	66384	3523.5
124295	0	Cash loans	F	N	Y	0	40500	288873	9198
124563	0	Cash loans	F	N	Y	0	112500	127350	6786
125629	0	Cash loans	F	N	Y	0	90000	265851	12919.5
125724	0	Revolving loans	F	N	Y	0	67500	202500	10125
125938	0	Cash loans	F	N	Y	0	36000	312768	15174
126474	0	Cash loans	F	N	Y	0	72000	204858	9022.5
126712	0	Cash loans	F	N	N	0	202500	1082214	35041.5
127086	0	Cash loans	F	N	Y	0	90000	361462.5	21973.5

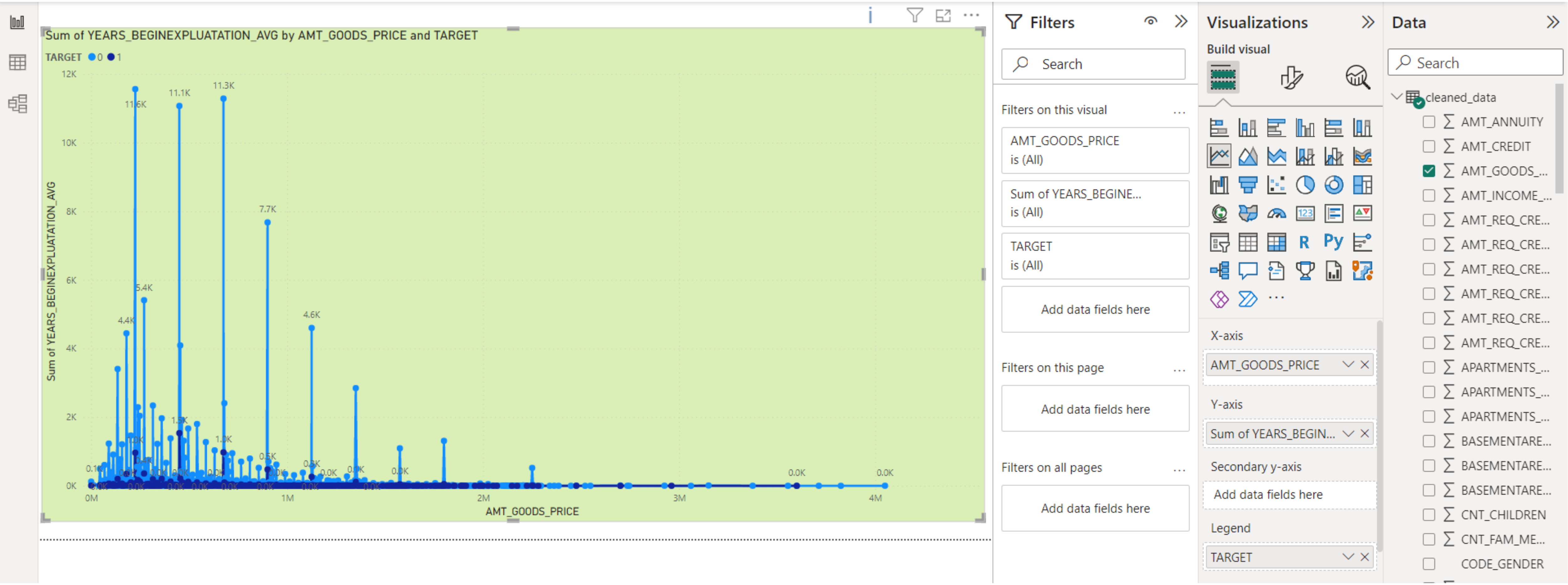
Loading application_data CSV as a cleaned file in POWER BI



**Each point on the LINE GRAPH represents a loan application, and the location of the point on the X and Y axis represents the amount of credit requested and the sum of income, respectively.A graph with "AMT_CREDIT" on the X-axis, "SUM of AMT_INCOME" on the Y-axis, and "TARGET" in the legend would likely indicate the relationship between credit amount, total income, and loan repayment capability. The legend would show the different categories of the target variable (likely indicating whether the loan was repaid or not).The graph would allow you to see if there is any correlation between credit amount, total income, and loan repayment capability. You could also look for any patterns or trends among the different categories of the target variable. For example, you could check if there is a particular range of credit amount or total income that is more likely to result in loan repayment or default. Additionally, you could use the graph to identify any outliers or unusual patterns in the data.



**the resulting graph will show the relationship between the average years of operation of the property and the amount of credit requested by the borrower for both the target categories.The graph can be used to identify any correlation between the sum of YEARS_BEGINEXPLUATATION_AVG and AMT_CREDIT for each TARGET category. If there is a visible trend, it may indicate that the amount of credit requested by the borrower is related to the age of the property.



**it will indicate the relationship between the average years of the beginning of the exploitation of the building and the price of the goods associated with the loan. The 'TARGET' legend will help to differentiate between the defaulters and non-defaulters.The scatter plot will show how the sum of the average years of the beginning of the exploitation of the building varies with the price of the goods for both defaulters and non-defaulters. It will help to identify any trends or patterns in the data and to see whether there is any correlation between the two variables. A positive correlation would indicate that as the sum of the average years of the beginning of the exploitation of the building increases, so does the price of the goods associated with the loan. A negative correlation would indicate the opposite, i.e., as the sum of the average years of the beginning of the exploitation of the building decreases, the price of the goods associated with the loan increases.