Credit_Amount Case Study

Exploratory Data Analysis

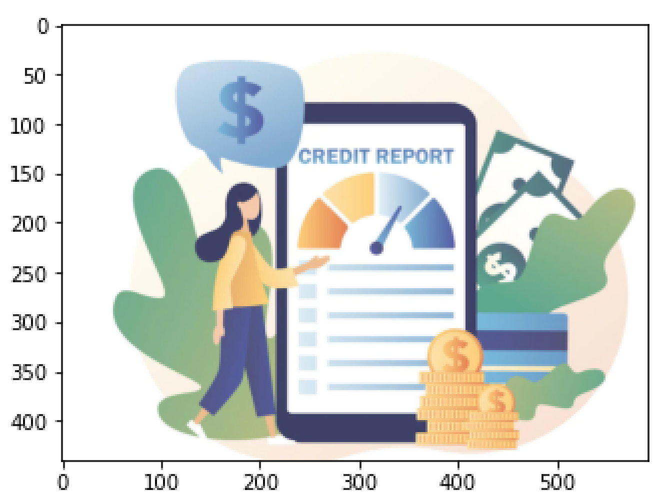
Introduction

> We have loan applications data for about 307k applications. The goal of this case is to perform Risk Analytics with the help of data wrangling and visualisation libraries of Python. The end goal is to derive important insights for the bank to identify the characteristics for bad loan applications. (Bad loans are loans which are delayed/not paid.)

Objectives

- Identify what are some common characteristics of bad loan applications
- Identify if there are any patterns related to applicants with loan difficulties
- Identify the driving factors or strong indicators of a bad loan application

In [282]:
```
img= mpimg.imread("cover-image.PNG")
plt.imshow(img)
plt.show()
```



Project Files

data_dictionary

https://drive.google.com/file/d/1Tsu3qdkakVsLCtzpNzBU5ZJt8aqdMbFt/view?usp=share_link (https://drive.google.com/file/d/1Tsu3qdkakVsLCtzpNzBU5ZJt8aqdMbFt/view?usp=share_link)

credit_data

https://drive.google.com/file/d/1OTM7_A5YkPKjBiniCfzBIq2YVviFxjuc/view?usp=sharing (https://drive.google.com/file/d/1OTM7_A5YkPKjBiniCfzBIq2YVviFxjuc/view?usp=sharing)

Libraries

In [283]:
```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

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

In [284]:
```
pd.options.display.max_rows = 4000
pd.options.display.max_columns = 1000
```

Read Files

In [285]:
```
credit_data= pd.read_csv("application_data.csv")
credit_data
```

Out[285]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_O
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	
...	...	...	...	...	...	
307506	456251	0	Cash loans	M	N	

1. Data Wrangling

In [286]:
```
credit_data.head(20)
```

Out[286]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	
5	100008	0	Cash loans	M	N	
6	100009	0	Cash loans	F	Y	

By observing the data we find that it is a mix of Qualitative and Quantitative variable.

1.1 Inspecting Data

In [287]:

```
# checking the shape of the data

print(f"Number of rows are {credit_data.shape[0]}")
print(f"Number of columns are {credit_data.shape[1]}")
```

```
Number of rows are 307511
Number of columns are 122
```

In [288]:

```
# checking 5 point summary

credit_data.describe()
```

Out[288]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_A
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.(
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.!
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.'
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.!
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.(
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.(
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.(
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.!

we can see there are some invalid **-ve** values in the column (DAYS_BIRTH)

In [289]:

```
# List all Null values

credit_data.isnull().sum().sort_values(ascending=False)
```

Out[289]:

```
COMMONAREA_MEDI             214865
COMMONAREA_AVG              214865
COMMONAREA_MODE             214865
NONLIVINGAPARTMENTS_MODE    213514
NONLIVINGAPARTMENTS_AVG     213514
NONLIVINGAPARTMENTS_MEDI    213514
FONDKAPREMONT_MODE          210295
LIVINGAPARTMENTS_MODE       210199
LIVINGAPARTMENTS_AVG        210199
LIVINGAPARTMENTS_MEDI       210199
FLOORSMIN_AVG               208642
FLOORSMIN_MODE              208642
FLOORSMIN_MEDI              208642
YEARS_BUILD_MEDI            204488
YEARS_BUILD_MODE            204488
YEARS_BUILD_AVG             204488
OWN_CAR_AGE                 202929
LANDAREA_MEDI               182590
LANDAREA_MODE               182590
```

In [290]:
```python
# percentage of Null values

null_per= credit_data.isnull().sum().sort_values(ascending=False)/len(credit_
null_per
```

Out[290]:
```
COMMONAREA_MEDI               69.872297
COMMONAREA_AVG                69.872297
COMMONAREA_MODE               69.872297
NONLIVINGAPARTMENTS_MODE      69.432963
NONLIVINGAPARTMENTS_AVG       69.432963
NONLIVINGAPARTMENTS_MEDI      69.432963
FONDKAPREMONT_MODE            68.386172
LIVINGAPARTMENTS_MODE         68.354953
LIVINGAPARTMENTS_AVG          68.354953
LIVINGAPARTMENTS_MEDI         68.354953
FLOORSMIN_AVG                 67.848630
FLOORSMIN_MODE                67.848630
FLOORSMIN_MEDI                67.848630
YEARS_BUILD_MEDI              66.497784
YEARS_BUILD_MODE              66.497784
YEARS_BUILD_AVG               66.497784
OWN_CAR_AGE                   65.990810
LANDAREA_MEDI                 59.376738
LANDAREA_MODE                 59.376738
LANDAREA_AVG                  59.376738
```

we will not consider the columns having **Null values >45%**

In [291]:
```python
# filter top 60 columns with max null values

null_per.head(60)
```

Out[291]:
```
COMMONAREA_MEDI               69.872297
COMMONAREA_AVG                69.872297
COMMONAREA_MODE               69.872297
NONLIVINGAPARTMENTS_MODE      69.432963
NONLIVINGAPARTMENTS_AVG       69.432963
NONLIVINGAPARTMENTS_MEDI      69.432963
FONDKAPREMONT_MODE            68.386172
LIVINGAPARTMENTS_MODE         68.354953
LIVINGAPARTMENTS_AVG          68.354953
LIVINGAPARTMENTS_MEDI         68.354953
FLOORSMIN_AVG                 67.848630
FLOORSMIN_MODE                67.848630
FLOORSMIN_MEDI                67.848630
YEARS_BUILD_MEDI              66.497784
YEARS_BUILD_MODE              66.497784
YEARS_BUILD_AVG               66.497784
OWN_CAR_AGE                   65.990810
LANDAREA_MEDI                 59.376738
LANDAREA_MODE                 59.376738
LANDAREA_AVG                  59.376738
```

1.2 Data Cleaning

Identifying and removing columns having Null values> 45%

In [292]:
```
null_col = credit_data.isnull().sum().sort_values(ascending=False)
len(null_col)
```

Out[292]: 122

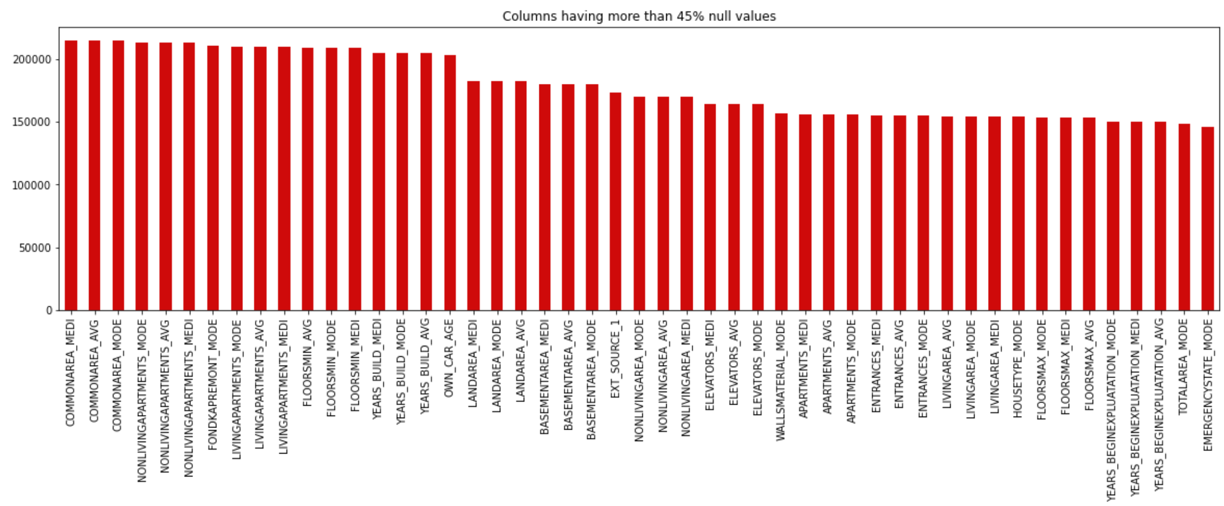
In [293]:
```
null_col= null_col[null_col.values>(0.45*len(credit_data))]
```

In [294]:
```
len(null_col)
```

Out[294]: 49

So there are total **49** Null Columns having **Null values >45%**

In [295]:
```
plt.figure(figsize=(20,5))
null_col.plot(kind='bar', color="#CF0A0A")

plt.title("Columns having more than 45% null values")
plt.show()
```



removing of columns with null values >45%

In [296]:
```
# function to remove null values

def remove_null_cols(data):
    perc=0.45
    df = data.copy()
    shape_before = df.shape
    remove_cols = (df.isnull().sum()/len(df))
    remove_cols = list(remove_cols[remove_cols.values>=perc].index)
    df.drop(labels = remove_cols,axis =1,inplace=True)
    print("Number of Columns dropped\t: ",len(remove_cols))
    print("\nOld dataset rows,columns",shape_before,"\nNew dataset rows,colum
    return df
```

In [297]:
```
# after removal of all null columns now we have credit_data1 for further anal

credit_data1 = remove_null_cols(credit_data)
```

```
Number of Columns dropped       :  49

Old dataset rows,columns (307511, 122)
New dataset rows,columns (307511, 73)
```

In [298]:
```
# checking percent of null values in new dataset

null_per1= credit_data1.isnull().sum().sort_values(ascending=False)/len(credi
null_per1.head(30)
```

Out[298]:
```
OCCUPATION_TYPE               31.345545
EXT_SOURCE_3                  19.825307
AMT_REQ_CREDIT_BUREAU_YEAR    13.501631
AMT_REQ_CREDIT_BUREAU_QRT     13.501631
AMT_REQ_CREDIT_BUREAU_MON     13.501631
AMT_REQ_CREDIT_BUREAU_WEEK    13.501631
AMT_REQ_CREDIT_BUREAU_DAY     13.501631
AMT_REQ_CREDIT_BUREAU_HOUR    13.501631
NAME_TYPE_SUITE                0.420148
OBS_30_CNT_SOCIAL_CIRCLE       0.332021
DEF_30_CNT_SOCIAL_CIRCLE       0.332021
OBS_60_CNT_SOCIAL_CIRCLE       0.332021
DEF_60_CNT_SOCIAL_CIRCLE       0.332021
EXT_SOURCE_2                   0.214626
AMT_GOODS_PRICE                0.090403
AMT_ANNUITY                    0.003902
CNT_FAM_MEMBERS                0.000650
DAYS_LAST_PHONE_CHANGE         0.000325
FLAG_DOCUMENT_17               0.000000
```

now we can verified that our modified dataframe - "credit_data1" has no cols more than 31% null values.

1.3 Imputing Missing Values

Below listed columns showns the same significance as they represent number of queries made to Credit Beureau

AMT_REQ_CREDIT_BUREAU_YEAR

AMT_REQ_CREDIT_BUREAU_MON

AMT_REQ_CREDIT_BUREAU_WEEK

AMT_REQ_CREDIT_BUREAU_DAY

AMT_REQ_CREDIT_BUREAU_HOUR

AMT_REQ_CREDIT_BUREAU_QRT

In [299]:
```
impute_list= ["AMT_REQ_CREDIT_BUREAU_YEAR",
"AMT_REQ_CREDIT_BUREAU_MON",
"AMT_REQ_CREDIT_BUREAU_WEEK",
"AMT_REQ_CREDIT_BUREAU_DAY",
"AMT_REQ_CREDIT_BUREAU_HOUR",
"AMT_REQ_CREDIT_BUREAU_QRT"]
```

In [300]:
```
#loop is creted to get MODE at once

for i in impute_list:
    print(f" Mode of {i} is {credit_data1[i].mode()}")
    print("=="*40)
```

```
 Mode of AMT_REQ_CREDIT_BUREAU_YEAR is 0    0.0
Name: AMT_REQ_CREDIT_BUREAU_YEAR, dtype: float64
============================================================================
====
 Mode of AMT_REQ_CREDIT_BUREAU_MON is 0    0.0
Name: AMT_REQ_CREDIT_BUREAU_MON, dtype: float64
============================================================================
====
 Mode of AMT_REQ_CREDIT_BUREAU_WEEK is 0    0.0
Name: AMT_REQ_CREDIT_BUREAU_WEEK, dtype: float64
============================================================================
====
 Mode of AMT_REQ_CREDIT_BUREAU_DAY is 0    0.0
Name: AMT_REQ_CREDIT_BUREAU_DAY, dtype: float64
============================================================================
====
 Mode of AMT_REQ_CREDIT_BUREAU_HOUR is 0    0.0
Name: AMT_REQ_CREDIT_BUREAU_HOUR, dtype: float64
============================================================================
====
 Mode of AMT_REQ_CREDIT_BUREAU_QRT is 0    0.0
Name: AMT_REQ_CREDIT_BUREAU_QRT, dtype: float64
============================================================================
====
```

we found that the mode of above all columns is 0, Therefore we replace null values of all these columns with MODE

In [301]:
```python
# verfying count of null before imputation

for i in impute_list:
    print(f"Number of null values in {i} is {credit_data1[i].isnull().sum()}"
```

```
Number of null values in AMT_REQ_CREDIT_BUREAU_YEAR is 41519
Number of null values in AMT_REQ_CREDIT_BUREAU_MON is 41519
Number of null values in AMT_REQ_CREDIT_BUREAU_WEEK is 41519
Number of null values in AMT_REQ_CREDIT_BUREAU_DAY is 41519
Number of null values in AMT_REQ_CREDIT_BUREAU_HOUR is 41519
Number of null values in AMT_REQ_CREDIT_BUREAU_QRT is 41519
```

In [302]:
```python
# creating copy of credit_data1

credit_data2= credit_data1.copy()
```

In [303]:
```python
#Imputing null values with 0's

for i in impute_list:
    credit_data2[i]= credit_data1[i].fillna(credit_data1[i].mode()[0])
```

In [304]:
```python
# Verfying count of Nulls after imputation

for i in impute_list:
    print(f"Number of null values in {i} is {credit_data2[i].isnull().sum()}"
```

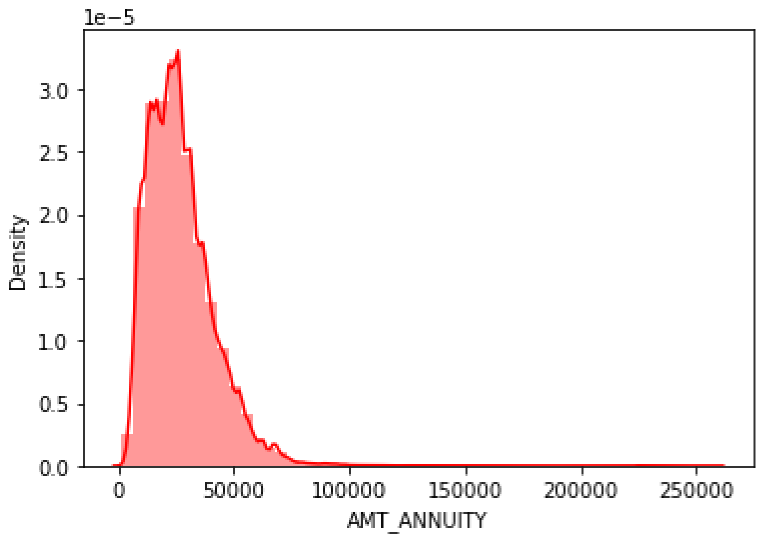
```
Number of null values in AMT_REQ_CREDIT_BUREAU_YEAR is 0
Number of null values in AMT_REQ_CREDIT_BUREAU_MON is 0
Number of null values in AMT_REQ_CREDIT_BUREAU_WEEK is 0
Number of null values in AMT_REQ_CREDIT_BUREAU_DAY is 0
Number of null values in AMT_REQ_CREDIT_BUREAU_HOUR is 0
Number of null values in AMT_REQ_CREDIT_BUREAU_QRT is 0
```

AMOUNT_ANNUITY

In [305]:
```python
credit_data1['AMT_ANNUITY'].describe()
```

Out[305]:
```
count    307499.000000
mean      27108.573909
std       14493.737315
min        1615.500000
25%       16524.000000
50%       24903.000000
75%       34596.000000
max      258025.500000
Name: AMT_ANNUITY, dtype: float64
```

In [306]:
```
sns.distplot(credit_data1['AMT_ANNUITY'], color="red")
plt.show()
print(f"Skewness is {credit_data1['AMT_ANNUITY'].skew()}")
```




```
Skewness is 1.5797773638612507
```

as the data is highly skewed, therefore we use MEDIAN to replace NULL VALUES

In [307]:
```
credit_data1['AMT_ANNUITY'].median()
```

Out[307]:
```
24903.0
```

In [308]:
```
# imputing AMT_ANNUITY Null values with median

credit_data2['AMT_ANNUITY']= credit_data1['AMT_ANNUITY'].fillna(credit_data1[
```

In [309]:
```
#Verfying count of Nulls after imputation

credit_data2['AMT_ANNUITY'].isnull().sum()
```

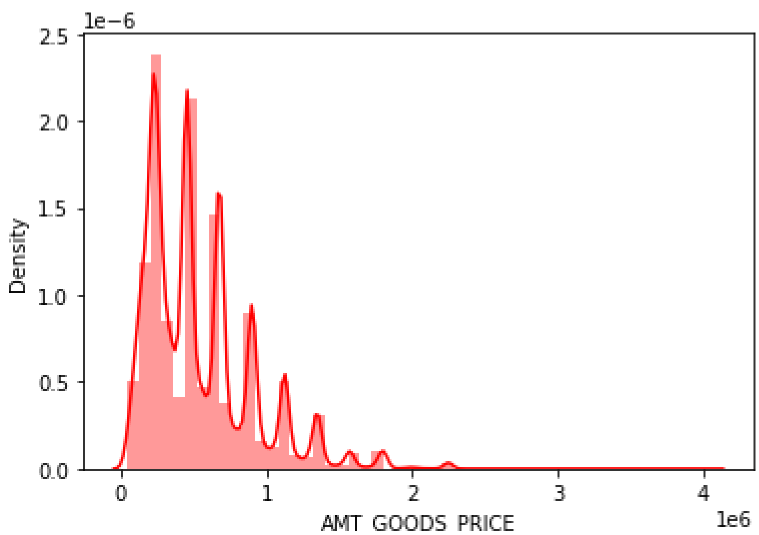
Out[309]:
```
0
```

In [310]:
```python
# checking null values

credit_data2.isnull().sum().sort_values(ascending=False)
```

Out[310]:
```
OCCUPATION_TYPE               96391
EXT_SOURCE_3                  60965
NAME_TYPE_SUITE                1292
OBS_30_CNT_SOCIAL_CIRCLE       1021
DEF_30_CNT_SOCIAL_CIRCLE       1021
OBS_60_CNT_SOCIAL_CIRCLE       1021
DEF_60_CNT_SOCIAL_CIRCLE       1021
EXT_SOURCE_2                    660
AMT_GOODS_PRICE                 278
CNT_FAM_MEMBERS                   2
DAYS_LAST_PHONE_CHANGE            1
FLAG_DOCUMENT_3                   0
FLAG_DOCUMENT_10                  0
FLAG_DOCUMENT_2                   0
FLAG_DOCUMENT_4                   0
FLAG_DOCUMENT_5                   0
FLAG_DOCUMENT_6                   0
FLAG_DOCUMENT_7                   0
FLAG_DOCUMENT_8                   0
```

AMT_GOODS_PRICE

In [311]:
```python
sns.distplot(credit_data1['AMT_GOODS_PRICE'], color="red")
plt.show()

print(f"Skewness is {credit_data1['AMT_GOODS_PRICE'].skew()}")
```




```
Skewness is 1.3490003414747445
```

As skewness of the data is high therefore we treat its **null values with mode**.

In [312]:
```python
# Insert NULL VALUES of AMT_GOODS_PRICE with mode=0

credit_data2['AMT_GOODS_PRICE']= credit_data1['AMT_GOODS_PRICE'].fillna(0)
```

In [313]:
```python
# Checking number of NULL VALUES

credit_data2['AMT_GOODS_PRICE'].isnull().sum()
```

Out[313]:
```
0
```

1.4 Fixing Erroneous Data

As we alredy know using describe function that we need to treat **-ve values** of days columns here we can modify the -ve values with **|-ve| absolute values** assuming -ve sign was a technical dafault during data feed

In [314]:
```python
# checking columns name

credit_data1.columns.sort_values()
```

Out[314]:
```
Index(['AMT_ANNUITY', 'AMT_CREDIT', 'AMT_GOODS_PRICE', 'AMT_INCOME_TOTAL',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_YEAR',
       'CNT_CHILDREN', 'CNT_FAM_MEMBERS', 'CODE_GENDER', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'DAYS_ID_PUBLISH', 'DAYS_LAST_PHONE_CHANGE',
       'DAYS_REGISTRATION', 'DEF_30_CNT_SOCIAL_CIRCLE',
       'DEF_60_CNT_SOCIAL_CIRCLE', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
       'FLAG_CONT_MOBILE', '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_2',
       'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21', 'FLAG_DOCUMENT_3',
       'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6',
       'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9', 'FLAG_EMAI
L',
       'FLAG_EMP_PHONE', 'FLAG_MOBIL', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
       'FLAG_PHONE', 'FLAG_WORK_PHONE', 'HOUR_APPR_PROCESS_START',
       'LIVE_CITY_NOT_WORK_CITY', 'LIVE_REGION_NOT_WORK_REGION',
```

In [315]:
```python
# creating array day_list for DAYS_COLUMNS

erroneous_cols= ['DAYS_BIRTH','DAYS_EMPLOYED', 'DAYS_ID_PUBLISH', 'DAYS_LAST_
erroneous_cols
```

Out[315]:
```
['DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_ID_PUBLISH',
 'DAYS_LAST_PHONE_CHANGE',
 'DAYS_REGISTRATION']
```

So that further operations on these columns can be done easily

-- in this we can create function to columns whose name start with DAY

```
erroneous_cols = [cols for cols in credit_data_2 if cols.startswith('DAYS')]
```

In [316]:
```
# checking if Days Columns contain -ve values or not

for i in erroneous_cols:
    print(credit_data1[i].unique())
    print("=="*40)
```

```
[ -9461 -16765 -19046 ...  -7951  -7857 -25061]
================================================================================
====
[  -637  -1188   -225 ... -12971 -11084  -8694]
================================================================================
====
[-2120  -291 -2531 ... -6194 -5854 -6211]
================================================================================
====
[-1134.  -828.  -815. ... -3988. -3899. -3538.]
================================================================================
====
[ -3648.  -1186.  -4260. ... -16396. -14558. -14798.]
================================================================================
====
```

In [317]:
```
# changing values to +ve using np.abs() function

credit_data2[erroneous_cols] = np.abs(credit_data2[erroneous_cols])
```

In [318]:
```
credit_data2.describe()
```

Out[318]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_A
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307511.(
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.4
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.4
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.5
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.(
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.(
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.(
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.5

now we can check all the values in day column converted into +ve values

Replacing XNAs for CODE_GENDER

In [319]:
```
#checking column for XNA values

credit_data2['CODE_GENDER'].value_counts()
```

Out[319]:
```
F      202448
M      105059
XNA         4
Name: CODE_GENDER, dtype: int64
```

here XNA value are very less therefor we replace it with "F"

In [320]:
```
credit_data2.loc[credit_data2['CODE_GENDER'] == 'XNA', "CODE_GENDER"] = "F"
```

In [321]:
```
credit_data2['CODE_GENDER'].value_counts()
```

Out[321]:
```
F    202452
M    105059
Name: CODE_GENDER, dtype: int64
```

Replacing XNAs for ORGANIZATION_TYPE

In [322]:
```
credit_data2['ORGANIZATION_TYPE'].value_counts()
```

Out[322]:
```
Business Entity Type 3    67992
XNA                       55374
Self-employed             38412
Other                     16683
Medicine                  11193
Business Entity Type 2    10553
Government                10404
School                     8893
Trade: type 7              7831
Kindergarten               6880
Construction               6721
Business Entity Type 1     5984
Transport: type 4          5398
Trade: type 3              3492
Industry: type 9           3368
Industry: type 3           3278
Security                   3247
Housing                    2958
Industry: type 11          2704
```

XNAs for ORGANIZATION_TYPE have 2nd highest count in the data. We must be very careful in imputing such a high number of XNAs with any value.

Since it is a categorical variable, and there won't be any aggregrate functions performed on this data, we don't necessarily need whole of the value to be imputed.

Thus, changing all XNAs with NULLs to protect the originality of data.

In [323]:
```
# replacing XNA with NAN using np.NAN

credit_data2['ORGANIZATION_TYPE'] = credit_data2['ORGANIZATION_TYPE'].replace
```

In [324]:
```
# checking

credit_data2['ORGANIZATION_TYPE'].value_counts()
```

Out[324]:
```
Business Entity Type 3    67992
Self-employed             38412
Other                     16683
Medicine                  11193
Business Entity Type 2    10553
Government                10404
School                     8893
Trade: type 7              7831
Kindergarten               6880
Construction               6721
Business Entity Type 1     5984
Transport: type 4          5398
Trade: type 3              3492
Industry: type 9           3368
Industry: type 3           3278
Security                   3247
Housing                    2958
Industry: type 11          2704
Military                   2634
```

We confirmed that there is no XNA now in the field

1.5 Adding new columns by Binning Continuous Variables

It is always a good practice to identify core or highly significant continuous fields in the data and then bin them into specific categories. It allows for an additional categorical analysis for such fields. We'll observe the use case of same later in this EDA exercise. For now, let's bin some of the continuous variables into 5 bins each as below -

In [325]:
```
credit_data2['AMT_INCOME_TOTAL'].describe()
```

Out[325]:
```
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
```

Binning AMT_INCOME_TOTAL

In [326]:
```
# using pd.qcut function we create an new column AMT_INCOME_RANGE that bin AM

credit_data2['AMT_INCOME_RANGE'] = pd.qcut(credit_data2['AMT_INCOME_TOTAL'],
                                           q=[0, 0.2, 0.5, 0.8, 0.95, 1],
                                           labels = ['Very_Low', 'Low', 'Medi

credit_data2['AMT_INCOME_RANGE'].head()
```

Out[326]:
```
0      Medium
1        High
2    Very_Low
3         Low
4         Low
Name: AMT_INCOME_RANGE, dtype: category
Categories (5, object): ['Very_Low' < 'Low' < 'Medium' < 'High' < 'Very_Hig
h']
```

In [327]:
```
credit_data2['AMT_INCOME_RANGE'].value_counts()
```

Out[327]:
```
Medium       106633
Low           90089
Very_Low      63671
High          33083
Very_High     14035
Name: AMT_INCOME_RANGE, dtype: int64
```

Binning AMT_CREDIT

In [328]:
```
# using pd.qcut bin the AMT_CREDIT into 5 categories

credit_data2['AMT_CREDIT_RANGE'] = pd.qcut(credit_data2['AMT_CREDIT'],
                                          q= [0, 0.2, 0.5, 0.8, 0.95, 1],
                                          labels=['VERY_LOW', 'LOW', "MEDIUM"

credit_data2['AMT_CREDIT_RANGE'].head()
```

Out[328]:
```
0         LOW
1        HIGH
2    VERY_LOW
3         LOW
4         LOW
Name: AMT_CREDIT_RANGE, dtype: category
Categories (5, object): ['VERY_LOW' < 'LOW' < 'MEDIUM' < 'HIGH' < 'VERY_HIG
H']
```

In [329]:
```
credit_data2['AMT_CREDIT_RANGE'].value_counts()
```

Out[329]:
```
MEDIUM       94750
LOW          88924
VERY_LOW     64925
HIGH         44878
VERY_HIGH    14034
Name: AMT_CREDIT_RANGE, dtype: int64
```

Binning DAYS_BIRTH

In [335]:
```
credit_data2["DAYS_BIRTH"]
```

Out[335]:
```
0          9461
1         16765
2         19046
3         19005
4         19932
          ...
307506     9327
307507    20775
307508    14966
307509    11961
307510    16856
Name: DAYS_BIRTH, Length: 307511, dtype: int64
```

In [343]:
```
# converting days into years as DAYS_BIRTH contains number of Days
# so dividing it by 365 we get number of year.
# convert it to int as it gives float values.

credit_data2['DAYS_BIRTH']= (credit_data2['DAYS_BIRTH']/365).astype(int)
```

In [340]:
```
credit_data2['DAYS_BIRTH_RANGE']= pd.cut(credit_data2['DAYS_BIRTH'],
                                      bins=[19,25,35,60,100],
                                      labels=['Very_Young','Young', 'Middle_A

credit_data2['DAYS_BIRTH_RANGE']
```

Out[340]:
```
0         Very_Young
1         Middle_Age
2         Middle_Age
3         Middle_Age
4         Middle_Age
             ...
307506    Very_Young
307507    Middle_Age
307508    Middle_Age
307509         Young
307510    Middle_Age
Name: DAYS_BIRTH_RANGE, Length: 307511, dtype: category
Categories (4, object): ['Very_Young' < 'Young' < 'Middle_Age' < 'Senior_Cit
izen']
```

In [341]:
```
credit_data2['DAYS_BIRTH_RANGE'].value_counts()
```

Out[341]:
```
Middle_Age        185900
Young              75925
Senior_Citizen     29368
Very_Young         16318
Name: DAYS_BIRTH_RANGE, dtype: int64
```

1.6 Splitting Data based on "Target"

Splitting data into 2 subsets based on Target Variable- Defaulter Data and Non-Defaulter Data.

This will help us with the comparison among 2 groups later

In [346]:
```
# 0 = More values means non_defaulters by default

credit_data2.TARGET.value_counts()
```

Out[346]:
```
0    282686
1     24825
Name: TARGET, dtype: int64
```

In [348]:
```
# SPlitting data as per TARGET into deafulter and non-defaulter datasets

defaulters = credit_data2[credit_data2.TARGET == 1]
non_defaulters = credit_data2[credit_data2.TARGET == 0]
```

In [353]:
```
# printing number of defaulters and non_defaulters

print(f"Number of Defaultes are {defaulters.shape[0]}")
print(f"Number of Non-Defaultes {non_defaulters.shape[0]}")
```

```
Number of Defaultes are 24825
Number of Non-Defaultes 282686
```

In [363]:
```
#printing their percentages

print(f"Percentage of Defaultes are {round(defaulters.shape[0]*100/credit_dat
print(f"Percentage of Non Defaultes are {round(non_defaulters.shape[0]*100/cr
```

```
Percentage of Defaultes are 8.07
Percentage of Non Defaultes are 91.93
```

2. Univariate Analysis

3. Bivariate Analysis

4. Final Insights

-- in upcoming post

In [ ]: