CREDIT EXPLORATORY DATA ANALYSIS

BY,

R.KEERTHIKA DEVI

INTRODUCTION

- This assignment aims to give you an idea of applying EDA in a real business scenario.
- In this assignment, apart from applying the techniques that you have learnt in the EDA module, you will also develop a basic understanding of risk analytics in banking and financial services and understand how data is used to minimize the risk of losing money while lending to customers.

PROBLEM STATEMENT

- This case study aims to identify patterns which indicate if a client has difficulty paying their instalments which may be used for taking action such as denying the loan, reducing the amount of loan, lending at a higher interest rate, etc. This will ensure that the consumers capable of repaying the loan are not rejected. Identification of such applicants using EDA is the aim of this case study.

BUSINESS OBJECTIVES

- This case study aims to identify patterns which indicate if a client has difficulty paying their installments which may be used for taking actions such as denying the loan, reducing the amount of loan, lending (to risky applicants) at a higher interest rate, etc. This will ensure that the consumers capable of repaying the loan are not rejected. Identification of such applicants using EDA is the aim of this case study.
- In other words, the company wants to understand the driving factors (or driver variables) behind loan default, i.e. the variables which are strong indicators of default. The company can utilise this knowledge for its portfolio and risk assessment.
- To develop your understanding of the domain, you are advised to independently research a little about risk analytics - understanding the types of variables and their significance should be enough).

BANK LOAN CASE STUDY

- This case study aims to identify patterns which indicate if a client has difficulty paying their instalments which may be used for taking action such as denying the loan, reducing the amount of loan, lending at the higher interest rate, etc. This will ensure that the consumers capable of repaying the loan are not rejected. Identification of such applicants using EDA is the aim of this case study.

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [475]:
```python
df.isnull().sum()
```

Out[475]:
```
SK_ID_CURR                        0
TARGET                            0
NAME_CONTRACT_TYPE                0
CODE_GENDER                       0
FLAG_OWN_CAR                      0
FLAG_OWN_REALTY                   0
CNT_CHILDREN                      0
AMT_INCOME_TOTAL                  0
AMT_CREDIT                        0
AMT_ANNUITY                      12
AMT_GOODS_PRICE                 278
NAME_TYPE_SUITE                1292
NAME_INCOME_TYPE                  0
NAME_EDUCATION_TYPE               0
NAME_FAMILY_STATUS                0
NAME_HOUSING_TYPE                 0
REGION_POPULATION_RELATIVE        0
DAYS_BIRTH                        0
DAYS_EMPLOYED                     0
DAYS_REGISTRATION                 0
```

In [476]:
```python
df.isnull().mean()
```

Out[476]:
```
SK_ID_CURR                     0.000000
TARGET                         0.000000
NAME_CONTRACT_TYPE             0.000000
CODE_GENDER                    0.000000
FLAG_OWN_CAR                   0.000000
FLAG_OWN_REALTY                0.000000
CNT_CHILDREN                   0.000000
AMT_INCOME_TOTAL               0.000000
AMT_CREDIT                     0.000000
AMT_ANNUITY                    0.000039
AMT_GOODS_PRICE                0.000904
NAME_TYPE_SUITE                0.004201
NAME_INCOME_TYPE               0.000000
NAME_EDUCATION_TYPE            0.000000
NAME_FAMILY_STATUS             0.000000
NAME_HOUSING_TYPE              0.000000
```

PERCENTAGE OF NULL VALUES IN EACH COLUMN

- percentage of data will lead to inGoing through the industry standards, columns with more than 5% of missing values can be dropped
- As if we have more than 50% of data missing in a column, imputing such a big correct results, hence they should be dropped.

```
In [481]: drop_colm=Temp(df)[Temp(df)>50]
          drop_colm

Out[481]: COMMONAREA_MEDI                  69.872
          COMMONAREA_AVG                   69.872
          COMMONAREA_MODE                  69.872
          NONLIVINGAPARTMENTS_MODE         69.433
          NONLIVINGAPARTMENTS_AVG          69.433
          NONLIVINGAPARTMENTS_MEDI         69.433
          FONDKAPREMONT_MODE               68.386
          LIVINGAPARTMENTS_MODE            68.355
          LIVINGAPARTMENTS_AVG             68.355
          LIVINGAPARTMENTS_MEDI            68.355
          FLOORSMIN_AVG                    67.849
          FLOORSMIN_MODE                   67.849
          FLOORSMIN_MEDI                   67.849
          YEARS_BUILD_MEDI                 66.498
          YEARS_BUILD_MODE                 66.498
          YEARS_BUILD_AVG                  66.498
          OWN_CAR_AGE                      65.991
          LANDAREA_MEDI                    59.377
          LANDAREA_MODE                    59.377
```

```
In [482]: drop_colm.index

Out[482]: Index(['COMMONAREA_MEDI', 'COMMONAREA_AVG', 'COMMONAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVIN
          GAPARTMENTS_MEDI', 'FONDKAPREMONT_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAPARTMENTS_AVG', 'LIVINGAPARTMENTS_MEDI', 'FLOORSMIN_A
          VG', 'FLOORSMIN_MODE', 'FLOORSMIN_MEDI', 'YEARS_BUILD_MEDI', 'YEARS_BUILD_MODE', 'YEARS_BUILD_AVG', 'OWN_CAR_AGE', 'LANDAREA_ME
          DI', 'LANDAREA_MODE', 'LANDAREA_AVG', 'BASEMENTAREA_MEDI', 'BASEMENTAREA_AVG', 'BASEMENTAREA_MODE', 'EXT_SOURCE_1', 'NONLIVINGA
          REA_MODE', 'NONLIVINGAREA_AVG', 'NONLIVINGAREA_MEDI', 'ELEVATORS_MEDI', 'ELEVATORS_AVG', 'ELEVATORS_MODE', 'WALLSMATERIAL_MOD
          E', 'APARTMENTS_MEDI', 'APARTMENTS_AVG', 'APARTMENTS_MODE', 'ENTRANCES_MEDI', 'ENTRANCES_AVG', 'ENTRANCES_MODE', 'LIVINGAREA_AV
          G', 'LIVINGAREA_MODE', 'LIVINGAREA_MEDI', 'HOUSETYPE_MODE'], dtype='object')

In [483]: df.drop(columns=drop_colm.index,inplace=True)

In [484]: df.shape

Out[484]: (307511, 81)
```

IMPUTING OR REPLACING NULL VALUES BASED ON PERCENT NULL VALUE

- Here, we are only imputing the values for some of the columns. Based on the data gives, we can impute values with either mode or mean or median, depending upon the columns, their data and more such factors.

Replacing Null Values

```
In [543]: AMT_REQ_CREDIT=['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 [544]: df[AMT_REQ_CREDIT].median()

Out[544]: AMT_REQ_CREDIT_BUREAU_HOUR     0.0
          AMT_REQ_CREDIT_BUREAU_DAY      0.0
          AMT_REQ_CREDIT_BUREAU_WEEK     0.0
          AMT_REQ_CREDIT_BUREAU_MON      0.0
          AMT_REQ_CREDIT_BUREAU_QRT      0.0
          AMT_REQ_CREDIT_BUREAU_YEAR     1.0
          dtype: float64

In [545]: df.fillna(df[AMT_REQ_CREDIT].median(),inplace=True)

In [546]: df[AMT_REQ_CREDIT].isnull().sum()

Out[546]: AMT_REQ_CREDIT_BUREAU_HOUR     0
          AMT_REQ_CREDIT_BUREAU_DAY      0
          AMT_REQ_CREDIT_BUREAU_WEEK     0
          AMT_REQ_CREDIT_BUREAU_MON      0
          AMT_REQ_CREDIT_BUREAU_QRT      0
          AMT_REQ_CREDIT_BUREAU_YEAR     0
          dtype: int64
```

COLLECTING COLUMNS STARTING WITH 'DAYS'

- If you see the data carefully, you will find that though these are days, it contains negative values which is not valid. So let's make changes accordingly.
- As you can see all the columns starts with DAYS, let's make a list of columns we want to change for ease of change.

Out[80]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIV
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307511.000000	3.075110e+05	307511.0000
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.487841	5.383163e+05	0.0208
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.461065	3.692890e+05	0.0138
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	0.0002
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	0.0100
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	0.0188
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	0.0286
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.0725

Collecting columns starting with 'DAYS' in the list 'day_column'

In [213]:

```
day_column=[i for i in df if i.startswith('DAYS')]
day_column
```

Out[213]:

```
['DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'DAYS_LAST_PHONE_CHANGE']
```

In [214]:

```
df1[day_column]=abs(df[day_column])
```

In [215]:

```
print(df['DAYS_BIRTH'].unique())
print(df['DAYS_EMPLOYED'].unique())
print(df['DAYS_REGISTRATION'].unique())
print(df['DAYS_ID_PUBLISH'].unique())
print(df['DAYS_LAST_PHONE_CHANGE'].unique())
```

```
[ 9461 16765 19046 ...  7951  7857 25061]
[  637  1188   225 ... 12971 11084  8694]
[ 3648.  1186.  4260. ... 16396. 14558. 14798.]
```

APPLYING ABS() FUNCTION TO COLUMNS STARTING WITH 'DAYS' TO CONVERT THE NEGATIVE VALUES TO POSITIVE

In [214]:
```
df1[day_column]=abs(df[day_column])
```

In [215]:
```
print(df['DAYS_BIRTH'].unique())
print(df['DAYS_EMPLOYED'].unique())
print(df['DAYS_REGISTRATION'].unique())
print(df['DAYS_ID_PUBLISH'].unique())
print(df['DAYS_LAST_PHONE_CHANGE'].unique())
```

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

In [199]:
```
df[["DAYS_BIRTH","DAYS_EMPLOYED",
   "DAYS_REGISTRATION","DAYS_ID_PUBLISH",
    "DAYS_LAST_PHONE_CHANGE"]]=abs(df[["DAYS_BIRTH","DAYS_EMPLOYED",
    "DAYS_REGISTRATION","DAYS_ID_PUBLISH","DAYS_LAST_PHONE_CHANGE"]])
```

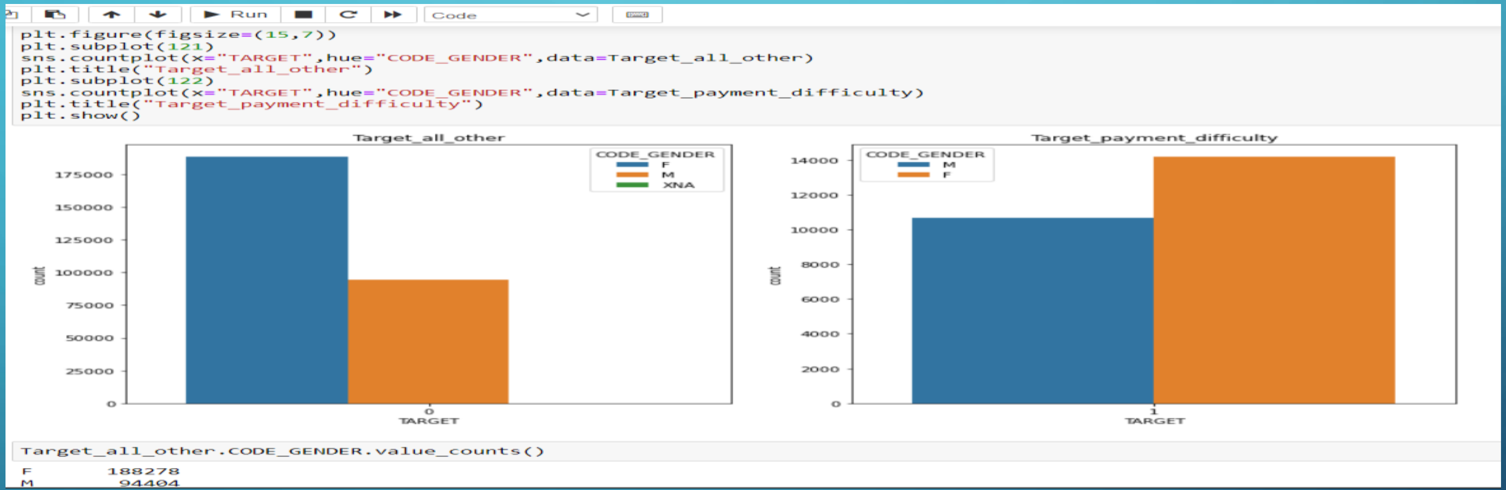
In [200]:
```
df[["DAYS_BIRTH","DAYS_EMPLOYED",
   "DAYS_REGISTRATION","DAYS_ID_PUBLISH",
    "DAYS_LAST_PHONE_CHANGE"]].describe()
```

Out[200]:

	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION	DAYS_ID_PUBLISH	DAYS_LAST_PHONE_CHANGE
count	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000
mean	16036.995067	67724.742149	4986.120328	2994.202373	962.858119
std	4363.988632	139443.751806	3522.886321	1509.450419	826.807226
min	7489.000000	0.000000	0.000000	0.000000	0.000000
25%	12413.000000	933.000000	2010.000000	1720.000000	274.000000
50%	15750.000000	2219.000000	4504.000000	3254.000000	757.000000
75%	19682.000000	5707.000000	7479.500000	4299.000000	1570.000000
max	25229.000000	365243.000000	24672.000000	7197.000000	4292.000000

DATA ANALYSIS: CODE_GENDER:

- As shown in the below screenshot when we plot the Male and Female gender ratio for total number of rows and against Target variable we found different ratios.



- As shown in the above screenshot, we have plotted CODE_GENDER values against total number of rows and Target variable.
- We found that the number of female clients is almost double the number of male clients.

UNIVARIATE ANALYSIS: NUMERICAL DATA:

Univariate Analysis

- Categorical Data and Numerical Data

In [125]:
```
df.head()
```

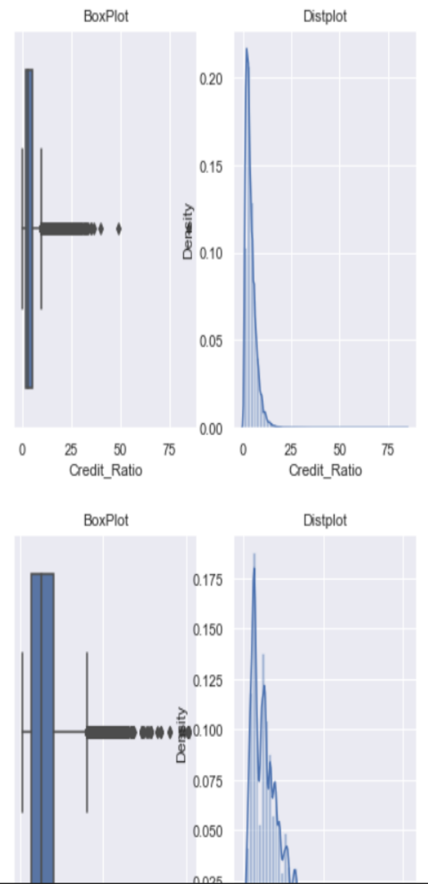
Out[125]:

	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
4	100007	0	Cash loans	M	N	Y	0	121500.0	51300

In [126]:
```
Numerical=['Credit_Ratio','ATM_CREDIT_1','ATM_INCOME','EMPLOYED_DAYS','AGE',
           'AMT_GOODS_PRICE','AMT_ANNUITY','CNT_FAM_MEMBERS']
```

In [127]:
```
def Uni_Numerical(dataframe, column):
    sns.set(style='darkgrid')
    plt.figure(figsize=(10,5))

    plt.subplot(1,3,1)
    sns.boxplot(data=dataframe, x=column, orient='v').set(title='BoxPlot')

    plt.subplot(1,3,2)
```

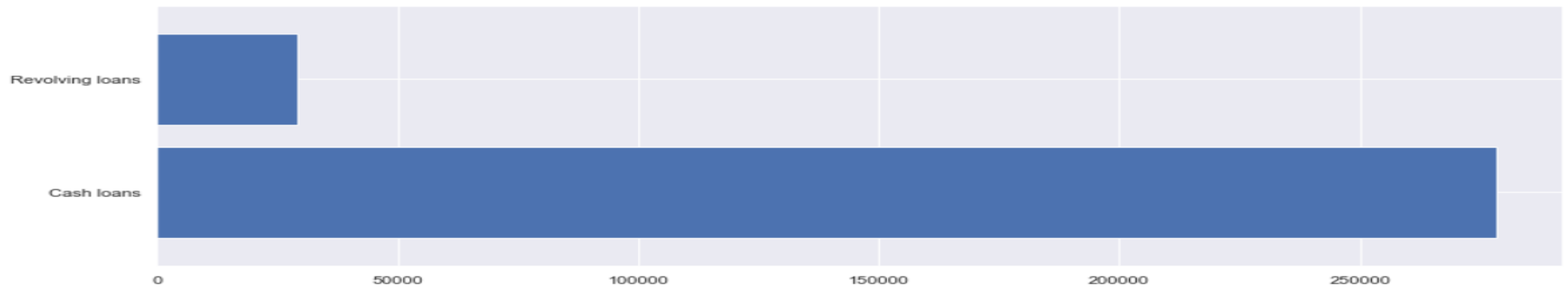
In [128]:
```
for i in Numerical:
    Uni_Numerical(df,i)
```



UNIVARIATE ANALYSIS: CATEGORICAL DATA:

- Plotting the various categorical column to check the customer with payment difficulties and customer with no payment difficulties by using Target column.

```
In [129]: Categorical_col=df.select_dtypes(include='object').columns
          Categorical_col

Out[129]: Index(['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDU
          CATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYPE'],
          dtype='object')

In [130]: Categorical=['NAME_CONTRACT_TYPE','FLAG_OWN_CAR','FLAG_OWN_REALTY',"NAME_TYPE_SUITE",
                       'NAME_INCOME_TYPE','NAME_EDUCATION_TYPE','NAME_FAMILY_STATUS',
                       'NAME_HOUSING_TYPE','OCCUPATION_TYPE','AGE_RANGE','EMPLOYED_DAYS_RANGE',
                       'ATM_CREDIT_1_RANGE','ATM_INCOME_RANGE']

In [131]: def Uni_Categorical(dataframe, column):
              sns.set(style='darkgrid')
              plt.figure(figsize=[15,5])
              dataframe[column].value_counts().plot.barh(width=0.8)
              plt.show()

In [132]: for i in Categorical:
              Uni_Categorical(df,i)
```

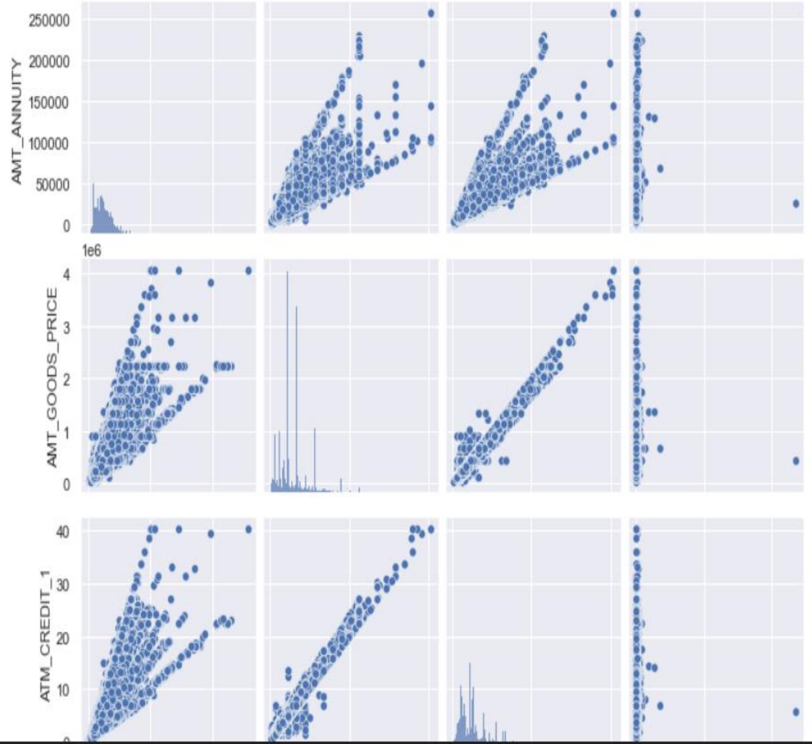


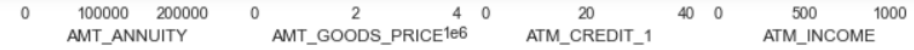
BIVARIATE ANALYSIS:

Bivariate Analysis

Target

In [133]:
```
sns.pairplot(data=df,vars=['AMT_ANNUITY','AMT_GOODS_PRICE','ATM_CREDIT_1','ATM_INCOME'])
plt.show()
```





In [134]:
```
sns.scatterplot(x=Target_payment_difficulty.AMT_ANNUITY,
                y=Target_payment_difficulty.AMT_GOODS_PRICE,
               data=Target_payment_difficulty, hue='TARGET')
```

Out[134]: <AxesSubplot:xlabel='AMT_ANNUITY', ylabel='AMT_GOODS_PRICE'>

In [135]:
```
plt.figure(figsize=[10,6])
sns.set(style='darkgrid')
sns.barplot(x=df.CODE_GENDER, y=df.AGE)
plt.show()
```

BIVARIATE ANALYSIS:
TARGET PAYMENT DIFFICULTY:

- Plotting a heatmap to understand the correlation between the target and other variable for customer with payment difficulties and other customers.

PREVIOUS APPLICATION DATA STRUCTURE:

Previous application

- Data Structure

In [140]:
```
df1=pd.read_csv("previous_application.csv")
df1.head()
```

Out[140]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WEEKI
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	

In [141]:
```
df1.shape
```

Out[141]:
```
(1670214, 37)
```

In [142]:
```
print(df1.info('all'))
```

```
 21  NAME_CLIENT_TYPE          1670214 non-null  object
 22  NAME_GOODS_CATEGORY       1670214 non-null  object
 23  NAME_PORTFOLIO            1670214 non-null  object
 24  NAME_PRODUCT_TYPE         1670214 non-null  object
 25  CHANNEL_TYPE              1670214 non-null  object
 26  SELLERPLACE_AREA          1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY      1670214 non-null  object
 28  CNT_PAYMENT               1297984 non-null  float64
 29  NAME_YIELD_GROUP          1670214 non-null  object
 30  PRODUCT_COMBINATION       1669868 non-null  object
```

Dropping Column

[146]:
```
Temp(df1)
```

[146]:
```
RATE_INTEREST_PRIVILEGED     99.644
RATE_INTEREST_PRIMARY        99.644
AMT_DOWN_PAYMENT             53.636
RATE_DOWN_PAYMENT            53.636
NAME_TYPE_SUITE              49.120
NFLAG_INSURED_ON_APPROVAL    40.298
DAYS_TERMINATION             40.298
DAYS_LAST_DUE                40.298
DAYS_LAST_DUE_1ST_VERSION    40.298
DAYS_FIRST_DUE               40.298
DAYS_FIRST_DRAWING           40.298
AMT_GOODS_PRICE              23.082
AMT_ANNUITY                  22.287
CNT_PAYMENT                  22.286
PRODUCT_COMBINATION           0.021
AMT_CREDIT                    0.000
NAME_YIELD_GROUP              0.000
NAME_PORTFOLIO                0.000
NAME_SELLER_INDUSTRY          0.000
```

[147]:
```
drop_colm1=Temp(df1)[Temp(df1)>50]
drop_colm1
```

[147]:
```
RATE_INTEREST_PRIVILEGED    99.644
RATE_INTEREST_PRIMARY       99.644
AMT_DOWN_PAYMENT            53.636
RATE_DOWN_PAYMENT           53.636
dtype: float64
```

[148]:
```
drop_colm1.index
```

[148]:
```
Index(['RATE_INTEREST_PRIVILEGED', 'RATE_INTEREST_PRIMARY', 'AMT_DOWN_PAYMENT', 'RATE_DOWN_PAYMENT'], dtype='object')
```

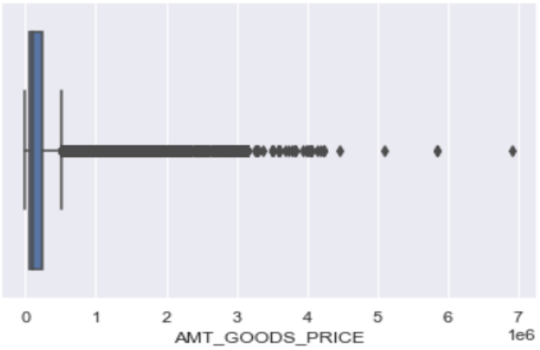
FILLING NULL VALUES

- Filling Null values

```
198]: df1.NAME_TYPE_SUITE.value_counts()

198]: Unaccompanied      1329375
      Family              213263
      Spouse, partner      67069
      Children             31566
      Other_B              17624
      Other_A               9077
      Group of people       2240
      Name: NAME_TYPE_SUITE, dtype: int64

153]: df1.NAME_TYPE_SUITE.isnull().sum()

153]: 820405

154]: df1['NAME_TYPE_SUITE']=df1.NAME_TYPE_SUITE.fillna('Unaccompanied')

156]: df1.AMT_GOODS_PRICE.describe()

156]: count    1.284699e+06
      mean     2.278473e+05
      std      3.153966e+05
      min      0.000000e+00
      25%      5.084100e+04
      50%      1.123200e+05
      75%      2.340000e+05
      max      6.905160e+06
      Name: AMT_GOODS_PRICE, dtype: float64

157]: sns.boxplot(df1.AMT_GOODS_PRICE)
      plt.show()
```



```
      25%      5.084100e+04
      50%      1.123200e+05
      75%      2.340000e+05
      max      6.905160e+06
      Name: AMT_GOODS_PRICE, dtype: float64

In [157]: sns.boxplot(df1.AMT_GOODS_PRICE)
          plt.show()
```




```
In [158]: df1.AMT_GOODS_PRICE.median()

Out[158]: 112320.0

In [159]: df1["AMT_GOODS_PRICE"]=df1.AMT_GOODS_PRICE.fillna(df1['AMT_GOODS_PRICE']==df1['AMT_CREDIT'])

In [160]: df1.AMT_GOODS_PRICE.isnull().sum()

Out[160]: 0

In [161]: df1.AMT_CREDIT.isnull().sum()

Out[161]: 1
```

BINNING OF CONTINUOUS VARIABLES:

Binning of continuous Variables

[179]:
```
df1.describe()
```

[179]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	HOUR_APPR_PROCESS_START	NFLAG_LAST_APPL_IN_DAY	DAYS_DECI:
count	1.670214e+06	1.670214e+06	1.670214e+06	1.670214e+06	1.670213e+06	1.670214e+06	1.670214e+06	1.670214
mean	1.923089e+06	2.783572e+05	1.490651e+04	1.752339e+05	1.961140e+05	1.248418e+01	9.964675e-01	-8.806797
std	5.325980e+05	1.028148e+05	1.317751e+04	2.927798e+05	3.185746e+05	3.334028e+00	5.932963e-02	7.790997
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	-2.922000
25%	1.461857e+06	1.893290e+05	7.547096e+03	1.872000e+04	2.416050e+04	1.000000e+01	1.000000e+00	-1.300000
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	1.200000e+01	1.000000e+00	-5.810000
75%	2.384280e+06	3.675140e+05	1.682403e+04	1.803600e+05	2.164185e+05	1.500000e+01	1.000000e+00	-2.800000
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	2.300000e+01	1.000000e+00	-1.000000

[180]:
```
df1['AMT_ANNUITY_AMOUNT']=df1['AMT_ANNUITY']/100000
```

[181]:
```
df1['AMT_APPLICATION_AMOUNT']=df1['AMT_APPLICATION']/100000
```

[182]:
```
df1['AMT_CREDIT_AMOUNT']=df1['AMT_CREDIT']/100000
```

[183]:
```
df1[['AMT_ANNUITY_AMOUNT','AMT_APPLICATION_AMOUNT','AMT_CREDIT_AMOUNT']].describe()
```

[183]:

	AMT_ANNUITY_AMOUNT	AMT_APPLICATION_AMOUNT	AMT_CREDIT_AMOUNT
count	1.670214e+06	1.670214e+06	1.670213e+06
mean	1.490651e-01	1.752339e+00	1.961140e+00
std	1.317751e-01	2.927798e+00	3.185746e+00

APPLYING ABS() FUNCTION TO COLUMNS STARTING WITH 'DAYS' TO CONVERT THE NEGATIVE VALUES TO POSITIVE

- Applying abs() function to columns starting with 'DAYS' to convert the negative values to positive.

In [184]:
```
df1[['DAYS_DECISION','DAYS_FIRST_DRAWING',
     'DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSION',
     'DAYS_LAST_DUE','DAYS_TERMINATION']]=abs(df1[['DAYS_DECISION','DAYS_FIRST_DRAWING',
     'DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSION',
     'DAYS_LAST_DUE','DAYS_TERMINATION']])
```

In [185]:
```
df1[['DAYS_DECISION','DAYS_FIRST_DRAWING',
     'DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSION',
     'DAYS_LAST_DUE','DAYS_TERMINATION']]=round(df1[['DAYS_DECISION','DAYS_FIRST_DRAWING',
     'DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSION',
     'DAYS_LAST_DUE','DAYS_TERMINATION']]/365,2)
```

In [186]:
```
df1[['DAYS_DECISION','DAYS_FIRST_DRAWING',
     'DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSION',
     'DAYS_LAST_DUE','DAYS_TERMINATION']].describe()
```

Out[186]:

	DAYS_DECISION	DAYS_FIRST_DRAWING	DAYS_FIRST_DUE	DAYS_LAST_DUE_1ST_VERSION	DAYS_LAST_DUE	DAYS_TERMINATION
count	1.670214e+06	997149.000000	997149.000000	997149.000000	997149.000000	997149.000000
mean	2.412823e+00	937.921942	43.696681	96.338383	214.117964	228.783904
std	2.134519e+00	242.229786	197.281033	291.524391	407.764320	417.767311
min	0.000000e+00	0.010000	0.010000	0.000000	0.010000	0.010000
25%	7.700000e-01	1000.670000	1.300000	0.700000	1.250000	1.220000
50%	1.590000e+00	1000.670000	2.520000	2.030000	3.160000	3.210000
75%	3.560000e+00	1000.670000	5.000000	4.750000	6.620000	6.850000
max	8.010000e+00	1000.670000	1000.670000	1000.670000	1000.670000	1000.670000

In [187]:
```
df1['AMT_CREDIT_AMOUNT_RANGE']=pd.cut(df['ATM_INCOME'],
                                     bins=[0,1,2,3,4,5,6,7,8,9,10,100],
```

UNIVARIATE ANALYSIS: CATEGORICAL DATA:

Univariate Analysis

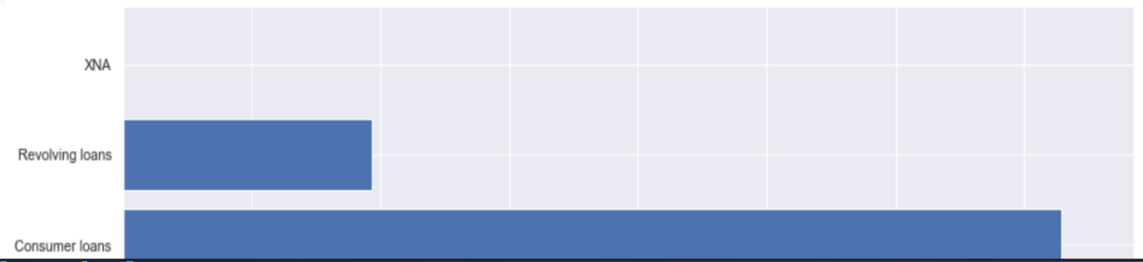
[189]:
```
df1.head()
```

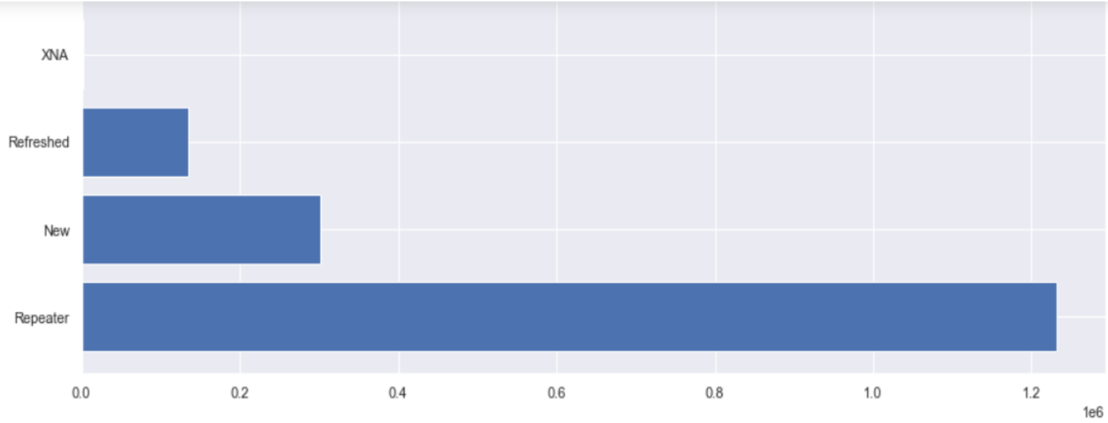
[189]:

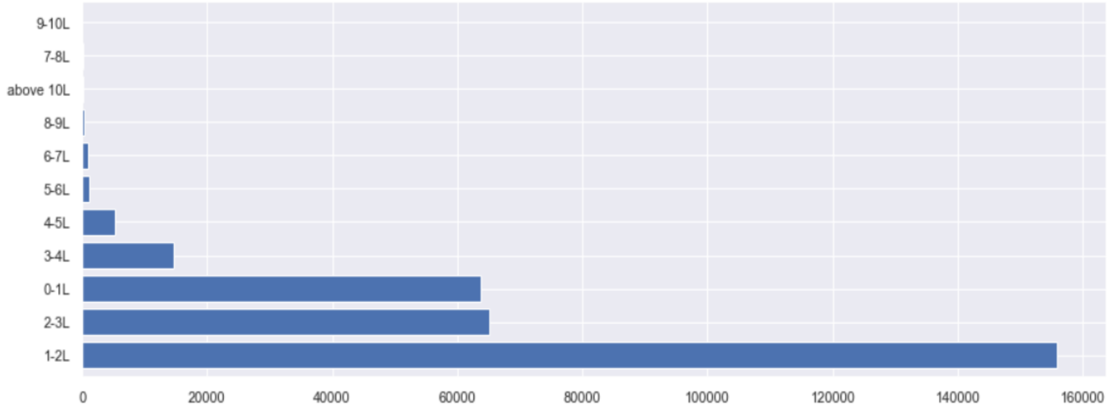
	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	WEEKDAY_APPR_PROCESS_ST
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	17145.0	SATUR
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	607500.0	THURS
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	112500.0	TUES
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	450000.0	MON
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	337500.0	THURS

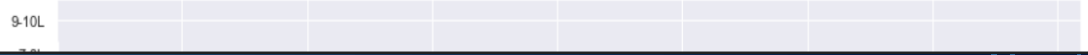
[190]:
```
Categorical_Data=['NAME_CONTRACT_TYPE',
                  'NAME_CONTRACT_STATUS','NAME_PAYMENT_TYPE',
                 'NAME_TYPE_SUITE','NAME_CLIENT_TYPE',
                  'AMT_CREDIT_AMOUNT_RANGE','AMT_APPLICATION_AMOUNT_RANGE']
```

[191]:
```
for i in Categorical_Data:
    Uni_Categorical(df1,i)
```









CONCLUSION:

- Now that we have understood and gained insight into the dataset ie): performed an exploratory Data Analysis, try to use ML algorithms to classify fraudulently. So let's summarize what we have learnt in this case study,
- We have extensively covered pre-processing steps required to analyse data
- We have covered Null value imputation methods
- We have also covered step by step analysing techniques such as Univariate analysis, Bivariate analysis.