Importing required data

```
import numpy as np
import pandas as pd
import os
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split
import statsmodels
import statsmodels.api as sm
from scipy.stats import kurtosis
import scipy
```

importing datasets uploaded on the google drive

```
#for importing datasets uploaded on the google drive
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

view the folder import from the google drive

```
#to view the folder import from the google drive
!ls "/content/drive/MyDrive/AAS module end exam/"
```

```
'AAS Module Test.docx'     'EDA ZOMATO.ipynb'
 application_data.csv       previous_application.csv
 columns_description.csv   'Problem Statement.docx'
 Covid_Analysis.ipynb      'Summary Report_Vineeta_Aman.pdf'
'EDA STEPS.docx'
```

assigning path variable to path of the data sets

```
#assigning path variable to path of the data sets
path1=  "/content/drive/MyDrive/AAS module end exam/application_data.csv"
path2=  "/content/drive/MyDrive/AAS module end exam/previous_application.csv"
```

+ Code

Importing Data from CSV

```
[ ] # importing data set 'application_data' from csv from the drive
    appl_data = pd.read_csv(path1)
```

```
[ ] # importing "previous_application" data set from csv the drive
    pa_df=pd.read_csv(path2)
```

```
# importing "columns_description" data set from csv
col_df=pd.read_csv("columns_description.csv", encoding='latin1')
```

Viewing the Dataset

checking first five rows and columns of the application_data dataset

```
#checking first five rows and columns of the application_data dataset
appl_data.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	...
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	...
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	...
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	...
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	...
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	...

5 rows × 122 columns

checking first five rows and columns of the previous_application dataset

```
#checking first five rows and columns of the previous_application dataset
pa_df.head()
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WEEKDAY_APPR_PROCESS_START	H
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	17145.0	SATURDAY	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	607500.0	THURSDAY	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	112500.0	TUESDAY	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	450000.0	MONDAY	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	337500.0	THURSDAY	

5 rows × 37 columns

checking first five rows and columns of the columns_description dataset

```
#checking first five rows and columns of the columns_description dataset
col_df.head()
```

	Unnamed: 0	Table	Row	Description	Special
0	1	application_data	SK_ID_CURR	ID of loan in our sample	NaN
1	2	application_data	TARGET	Target variable (1 - client with payment diffi...	NaN
2	5	application_data	NAME_CONTRACT_TYPE	Identification if loan is cash or revolving	NaN
3	6	application_data	CODE_GENDER	Gender of the client	NaN
4	7	application_data	FLAG_OWN_CAR	Flag if the client owns a car	NaN

checking the shape of the dataset "application_data"

```
#checking the shape of the dataset "application_data"
appl_data.shape
```

```
(307511, 122)
```

checking the shape of the dataset "previous_application"

```
#checking the shape of the dataset "previous_application"
pa_df.shape
```

```
(1670214, 37)
```

checking the shape of the dataset "columns_description"

```
#checking the shape of the dataset "columns_description"
col_df.shape
```

```
(160, 5)
```

checking the dimension of the "application_data" data set -- We have a two dimension dataset

```
#checking the dimension of the "application_data" data set -- We have a two dimension dataset
appl_data.ndim
```

```
2
```

checking the dimension of the "previous_application" data set -- We have a two dimension dataset

```
#checking the dimension of the "previous_application" data set -- We have a two dimension dataset
pa_df.ndim
```

```
2
```

checking last five rows and columns of the raw dataset--"application_data"

```
#checking last five rows and columns of the raw dataset--"application_data"
appl_data.tail()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	...	FI
307506	456251	0	Cash loans	M	N	N	0	157500.0	254700.0	27558.0	...	
307507	456252	0	Cash loans	F	N	Y	0	72000.0	269550.0	12001.5	...	
307508	456253	0	Cash loans	F	N	Y	0	153000.0	677664.0	29979.0	...	
307509	456254	1	Cash loans	F	N	Y	0	171000.0	370107.0	20205.0	...	
307510	456255	0	Cash loans	F	N	N	0	157500.0	675000.0	49117.5	...	

5 rows × 122 columns

checking last five rows and columns of the raw dataset--"previous_data"

```
#checking last five rows and columns of the raw dataset--"previous_data"
pa_df.tail()
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WEEKDAY_APPR_PROCESS_START	HC
1670209	2300464	352015	Consumer loans	14704.290	267295.5	311400.0	0.0	267295.5	WEDNESDAY	
1670210	2357031	334635	Consumer loans	6622.020	87750.0	64291.5	29250.0	87750.0	TUESDAY	
1670211	2659632	249544	Consumer loans	11520.855	105237.0	102523.5	10525.5	105237.0	MONDAY	
1670212	2785582	400317	Cash loans	18821.520	180000.0	191880.0	NaN	180000.0	WEDNESDAY	
1670213	2418762	261212	Cash loans	16431.300	360000.0	360000.0	NaN	360000.0	SUNDAY	

5 rows × 37 columns

checking the data types of columns of data set "application_data"

```
#checking the data types of columns of data set "application_data"
appl_data.dtypes
```

```
SK_ID_CURR                      int64
TARGET                          int64
NAME_CONTRACT_TYPE             object
CODE_GENDER                    object
FLAG_OWN_CAR                   object
                                ...
AMT_REQ_CREDIT_BUREAU_DAY     float64
AMT_REQ_CREDIT_BUREAU_WEEK    float64
AMT_REQ_CREDIT_BUREAU_MON     float64
AMT_REQ_CREDIT_BUREAU_QRT     float64
AMT_REQ_CREDIT_BUREAU_YEAR    float64
Length: 122, dtype: object
```

checking the data types of columns of data set "previous_application"

```
#checking the data types of columns of data set "previous_application"
pa_df.dtypes
```

```
SK_ID_PREV                      int64
SK_ID_CURR                      int64
NAME_CONTRACT_TYPE             object
AMT_ANNUITY                   float64
AMT_APPLICATION               float64
AMT_CREDIT                    float64
AMT_DOWN_PAYMENT              float64
AMT_GOODS_PRICE               float64
WEEKDAY_APPR_PROCESS_START     object
HOUR_APPR_PROCESS_START         int64
FLAG_LAST_APPL_PER_CONTRACT    object
NFLAG_LAST_APPL_IN_DAY          int64
```

'''it provides purely descriptive information about the dataset.
This information includes statistics that summarize the central tendency of the variable,

their dispersion, the presence of empty values and their shape'''

```
'''it provides purely descriptive information about the dataset.
This information includes statistics that summarize the central tendency of the variable,
 their dispersion, the presence of empty values and their shape'''
appl_data.info()
```

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

```
pa_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 16  NAME_CONTRACT_STATUS         1670214 non-null  object
 17  DAYS_DECISION                1670214 non-null  int64
 18  NAME_PAYMENT_TYPE            1670214 non-null  object
 19  CODE_REJECT_REASON           1670214 non-null  object
 20  NAME_TYPE_SUITE              849809 non-null   object
```

This shows the statistical summary of all numeric-typed (int, float) columns

```python
# This shows the statistical summary of all numeric-typed (int, float) columns
# Describe
appl_data.describe()
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE	
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	307511.000000	3
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	0.020868	-
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	0.013831	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	0.000290	-
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	0.010006	-
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	0.018850	-
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	0.028663	-
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.072508	

8 rows × 106 columns

```python
pa_df.describe()
```

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	HOUR_APPR_PROCESS_START	NFLAG_LAST_
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	7.743700e+05	1.284699e+06	1.670214e+06	
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	6.697402e+03	2.278473e+05	1.248418e+01	
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	2.092150e+04	3.153966e+05	3.334028e+00	
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	-9.000000e-01	0.000000e+00	0.000000e+00	
25%	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	0.000000e+00	5.084100e+04	1.000000e+01	
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	1.638000e+03	1.123200e+05	1.200000e+01	
75%	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	7.740000e+03	2.340000e+05	1.500000e+01	
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	3.060045e+06	6.905160e+06	2.300000e+01	

8 rows × 21 columns

checking all the columns including those that are of type object

```python
# checking all the columns including those that are of type object
appl_data.describe(include='all')
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
count	307511.000000	307511.000000	307511	307511	307511	307511	307511.000000	3.075110e+05	3.075110e+05
unique	NaN	NaN	2	3	2	2	NaN	NaN	NaN
top	NaN	NaN	Cash loans	F	N	Y	NaN	NaN	NaN
freq	NaN	NaN	278232	202448	202924	213312	NaN	NaN	NaN
mean	278180.518577	0.080729	NaN	NaN	NaN	NaN	0.417052	1.687979e+05	5.990260e+05
std	102790.175348	0.272419	NaN	NaN	NaN	NaN	0.722121	2.371231e+05	4.024908e+05
min	100002.000000	0.000000	NaN	NaN	NaN	NaN	0.000000	2.565000e+04	4.500000e+04
25%	189145.500000	0.000000	NaN	NaN	NaN	NaN	0.000000	1.125000e+05	2.700000e+05
50%	278202.000000	0.000000	NaN	NaN	NaN	NaN	0.000000	1.471500e+05	5.135310e+05
75%	367142.500000	0.000000	NaN	NaN	NaN	NaN	1.000000	2.025000e+05	8.086500e+05
max	456255.000000	1.000000	NaN	NaN	NaN	NaN	19.000000	1.170000e+08	4.050000e+06

11 rows × 122 columns

checking all the columns including those that are of type object

```
# checking all the columns including those that are of type object
pa_df.describe(include='all')
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WE
count	1.670214e+06	1.670214e+06	1670214	1.297979e+06	1.670214e+06	1.670213e+06	7.743700e+05	1.284699e+06	
unique	NaN	NaN	4	NaN	NaN	NaN	NaN	NaN	
top	NaN	NaN	Cash loans	NaN	NaN	NaN	NaN	NaN	
freq	NaN	NaN	747553	NaN	NaN	NaN	NaN	NaN	
mean	1.923089e+06	2.783572e+05	NaN	1.595512e+04	1.752339e+05	1.961140e+05	6.697402e+03	2.278473e+05	
std	5.325980e+05	1.028148e+05	NaN	1.478214e+04	2.927798e+05	3.185746e+05	2.092150e+04	3.153966e+05	
min	1.000001e+06	1.000010e+05	NaN	0.000000e+00	0.000000e+00	0.000000e+00	-9.000000e-01	0.000000e+00	
25%	1.461857e+06	1.893290e+05	NaN	6.321780e+03	1.872000e+04	2.416050e+04	0.000000e+00	5.084100e+04	
50%	1.923110e+06	2.787145e+05	NaN	1.125000e+04	7.104600e+04	8.054100e+04	1.638000e+03	1.123200e+05	
75%	2.384280e+06	3.675140e+05	NaN	2.065842e+04	1.803600e+05	2.164185e+05	7.740000e+03	2.340000e+05	

Data Preparation

```
appl_data.columns
```

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=122)
```

As data is having huge number of lines, we are considering applicable columns only by looking at data set

creating a new data set using the columns of appl_data

```python
#creating a new data set using the columns of appl_data

new_df=appl_data[['SK_ID_CURR', 'AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY',
             'AMT_GOODS_PRICE','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
             'NAME_FAMILY_STATUS','NAME_HOUSING_TYPE','DAYS_BIRTH',
             'DAYS_EMPLOYED','DAYS_REGISTRATION','DAYS_ID_PUBLISH',
             'FLAG_MOBIL','FLAG_EMAIL','OCCUPATION_TYPE','CNT_FAM_MEMBERS',
             'REGION_RATING_CLIENT','REGION_RATING_CLIENT_W_CITY',
             'REG_REGION_NOT_LIVE_REGION','REG_REGION_NOT_WORK_REGION',
             'ORGANIZATION_TYPE','EXT_SOURCE_1','EXT_SOURCE_2','EXT_SOURCE_3',
             'APARTMENTS_AVG','BASEMENTAREA_AVG','DAYS_LAST_PHONE_CHANGE']]
```

```python
newtest_df= pa_df[['SK_ID_PREV','SK_ID_CURR','NAME_CONTRACT_TYPE','AMT_ANNUITY',
                   'AMT_APPLICATION','AMT_CREDIT','AMT_DOWN_PAYMENT','AMT_GOODS_PRICE',
                   'NFLAG_LAST_APPL_IN_DAY','RATE_DOWN_PAYMENT','RATE_INTEREST_PRIMARY',
                   'RATE_INTEREST_PRIVILEGED','NAME_CASH_LOAN_PURPOSE','NAME_CONTRACT_STATUS',
                   'NAME_PAYMENT_TYPE','CODE_REJECT_REASON','NAME_CLIENT_TYPE','NAME_GOODS_CATEGORY',
                   'NAME_PORTFOLIO','CHANNEL_TYPE','NAME_SELLER_INDUSTRY','NAME_YIELD_GROUP',
                   'DAYS_FIRST_DUE','DAYS_LAST_DUE','DAYS_TERMINATION','NFLAG_INSURED_ON_APPROVAL']]
```

creating a new data set using the columns of pa_df

```
#creating a new data set using the columns of pa_df

list2= ['PREV_CustID',
'CURR_CustID',
'CONTRACT_TYPE',
'AMT_ANNUITY',
'AMT_APPLICATION',
'AMT_CREDIT',
'AMT_DOWN_PAYMENT',
'AMT_GOODS_PRICE',
'LASTAPPL_PerDAY',
'RATE_DOWN_PAYMENT',
'RATE_INTEREST_PRIMARY',
'RATE_INTEREST_PRIVILEGED',
'CASH_LOAN_PURPOSE',
'CONTRACT_STATUS',
'PAYMENT_TYPE',
'CODE_REJECT_REASON',
'CLIENT_TYPE',
'GOODS_CATEGORY',
'PORTFOLIO',
'CHANNEL_TYPE',
'SELLER_INDUSTRY',
'YIELD_GROUP',
'FIRST_DUEDay',
```

new dataframe created for pa_df -- "previous_application" with change in column names

```
#new dataframe created for pa_df -- "previous_application" with change in column names
newtest_df.columns= list2
```

checking if the list2 co

```
#checking if the list2 co
newtest_df.head()
```

	PREV_CustID	CURR_CustID	CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	LASTAPPL_PerDAY	RATE_DOWN_PA
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	17145.0	1	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	607500.0	1	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	112500.0	1	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	450000.0	1	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	337500.0	1	

5 rows × 26 columns

checking the new dataframe of appl_data -- "application_data"

```
# checking the new dataframe of appl_data -- "application_data"
new_df.head()
```

	SK_ID_CURR	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMILY_STATUS	NAME_
0	100002	202500.0	406597.5	24700.5	351000.0	Working	Secondary / secondary special	Single / not married	Hou
1	100003	270000.0	1293502.5	35698.5	1129500.0	State servant	Higher education	Married	Hou
2	100004	67500.0	135000.0	6750.0	135000.0	Working	Secondary / secondary special	Single / not married	Hou
3	100006	135000.0	312682.5	29686.5	297000.0	Working	Secondary / secondary special	Civil marriage	Hou
4	100007	121500.0	513000.0	21865.5	513000.0	Working	Secondary / secondary special	Single / not married	Hou

5 rows × 28 columns

checking shape of new data of appl_data , we have now reduced our no of columns from 122 to 28

```
#checking shape of new data of appl_data , we have now reduced our no of columns from 122 to 28
new_df.shape
```

```
(307511, 28)
```

checking names of columns of new data

```
#checking names of columns of new data
new_df.columns
```

```
Index(['SK_ID_CURR', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
       'AMT_GOODS_PRICE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'FLAG_MOBIL',
       'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS',
       'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'ORGANIZATION_TYPE', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
       'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'DAYS_LAST_PHONE_CHANGE'],
      dtype='object')
```

renaming the column names

creating a list to rename the columns

```
#creating a list to rename the columns

list=['Cust_ID', 'INCOME', 'Loan_AMT', 'ANNUITY',
      'GOODS_PRICE', 'INCOME_TYPE', 'EDUCATION',
      'FAMILY_STATUS', 'curr_HOUSING_TYPE', 'age',
      'Work_Exp', 'REGISTRATION_change', 'DAYS_ID_PUBLISH', 'MOBIL_given',
      'EMAIL_given', 'OCCUPATION_TYPE', 'Family_MEMBERS_no',
      'REGION_CLIENT', 'REGION_CLIENT_CITY',
      'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
      'ORGANIZATION_TYPE', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
      'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'DAYS_LAST_PHONE_CHANGE']
```

conactenating the list to new_df

```
# conactenating the list to new_df
new_df.columns=list
```

checking if the column names have changed

```python
#checking if the column names have changed

new_df.columns
```

```
Index(['Cust_ID', 'INCOME', 'Loan_AMT', 'ANNUITY', 'GOODS_PRICE',
       'INCOME_TYPE', 'EDUCATION', 'FAMILY_STATUS', 'curr_HOUSING_TYPE', 'age',
       'Work_Exp', 'REGISTRATION_change', 'DAYS_ID_PUBLISH', 'MOBIL_given',
       'EMAIL_given', 'OCCUPATION_TYPE', 'Family_MEMBERS_no', 'REGION_CLIENT',
       'REGION_CLIENT_CITY', 'REG_REGION_NOT_LIVE_REGION',
       'REG_REGION_NOT_WORK_REGION', 'ORGANIZATION_TYPE', 'EXT_SOURCE_1',
       'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG',
       'DAYS_LAST_PHONE_CHANGE'],
      dtype='object')
```

checking for columns with null values

checking how many null values are present in each of the columns
creating a function to find null values for the dataframe appl_data --> application_data

```python
#checking how many null values are present in each of the columns
#creating a function to find null values for the dataframe appl_data --> application_data
def null_values(appl_data):
    return round((appl_data.isnull().sum()*100/len(appl_data)).sort_values(ascending = False),2)
```

displaying the column names and displaying the percentage of columns having null values

```python
#displaying the column names and displaying the percentage of columns having null values
null_values(new_df)
```

```
BASEMENTAREA_AVG              58.52
EXT_SOURCE_1                  56.38
APARTMENTS_AVG                50.75
OCCUPATION_TYPE               31.35
EXT_SOURCE_3                  19.83
EXT_SOURCE_2                   0.21
GOODS_PRICE                    0.09
ANNUITY                        0.00
Family_MEMBERS_no              0.00
DAYS_LAST_PHONE_CHANGE         0.00
ORGANIZATION_TYPE              0.00
REG_REGION_NOT_WORK_REGION     0.00
REG_REGION_NOT_LIVE_REGION     0.00
REGION_CLIENT_CITY             0.00
REGION_CLIENT                  0.00
Cust_ID                        0.00
INCOME                         0.00
MOBIL_given                    0.00
DAYS_ID_PUBLISH                0.00
REGISTRATION_change            0.00
Work_Exp                       0.00
```

creating a function to find null values for the dataframe appl_data --> previous_data

```
#creating a function to find null values for the dataframe appl_data --> previous_data
def null_values(pa_df):
    return round((pa_df.isnull().sum()*100/len(pa_df)).sort_values(ascending = False),2)
```

displaying the column names and displaying the percentage of columns having null values in newtest_df

```
#displaying the column names and displaying the percentage of columns having null values in newtest_df
null_values(newtest_df)
```

```
RATE_INTEREST_PRIMARY       99.64
RATE_INTEREST_PRIVILEGED    99.64
AMT_DOWN_PAYMENT            53.64
RATE_DOWN_PAYMENT           53.64
INSURED_ON_APPROVAL         40.30
DAYS_TERMINATION            40.30
LAST_DUEDay                 40.30
FIRST_DUEDay                40.30
AMT_GOODS_PRICE             23.08
AMT_ANNUITY                 22.29
AMT_CREDIT                   0.00
GOODS_CATEGORY               0.00
YIELD_GROUP                  0.00
SELLER_INDUSTRY              0.00
CHANNEL_TYPE                 0.00
PORTFOLIO                    0.00
PREV_CustID                  0.00
CLIENT_TYPE                  0.00
CODE_REJECT_REASON           0.00
PAYMENT_TYPE                 0.00
CURR_CustID                  0.00
CASH_LOAN_PURPOSE            0.00
LASTAPPL_PerDAY              0.00
AMT_APPLICATION              0.00
```

these are the columns having maximum null values

1. RATE_INTEREST_PRIMARY 99.64
2. RATE_INTEREST_PRIVILEGED 99.64
3. AMT_DOWN_PAYMENT 53.64
4. RATE_DOWN_PAYMENT 53.64
5. INSURED_ON_APPROVAL 40.30
6. DAYS_TERMINATION 40.30
7. LAST_DUEDay 40.30
8. FIRST_DUEDay 40.30
9. AMT_GOODS_PRICE 23.08
10. AMT_ANNUITY 22.29

number of null values per column

```python
#number of null values per column
print("missing values : ",newtest_df.isna().sum().sort_values(ascending = False))
```

```
missing values :  RATE_INTEREST_PRIMARY       1664263
RATE_INTEREST_PRIVILEGED    1664263
AMT_DOWN_PAYMENT             895844
RATE_DOWN_PAYMENT            895844
INSURED_ON_APPROVAL          673065
DAYS_TERMINATION             673065
LAST_DUEDay                  673065
FIRST_DUEDay                 673065
AMT_GOODS_PRICE              385515
AMT_ANNUITY                  372235
AMT_CREDIT                        1
GOODS_CATEGORY                    0
YIELD_GROUP                       0
SELLER_INDUSTRY                   0
CHANNEL_TYPE                      0
PORTFOLIO                         0
PREV_CustID                       0
```

```python
newtest_df.describe(include='all')
```

	PREV_CustID	CURR_CustID	CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	LASTAPPL_PerDAY
count	1.670214e+06	1.670214e+06	1670214	1.297979e+06	1.670214e+06	1.670213e+06	7.743700e+05	1.284699e+06	1.670214e+06
unique	NaN	NaN	4	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	Cash loans	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	747553	NaN	NaN	NaN	NaN	NaN	NaN
mean	1.923089e+06	2.783572e+05	NaN	1.595512e+04	1.752339e+05	1.961140e+05	6.697402e+03	2.278473e+05	9.964675e-01
std	5.325980e+05	1.028148e+05	NaN	1.478214e+04	2.927798e+05	3.185746e+05	2.092150e+04	3.153966e+05	5.932963e-02
min	1.000001e+06	1.000010e+05	NaN	0.000000e+00	0.000000e+00	0.000000e+00	-9.000000e-01	0.000000e+00	0.000000e+00
25%	1.461857e+06	1.893290e+05	NaN	6.321780e+03	1.872000e+04	2.416050e+04	0.000000e+00	5.084100e+04	1.000000e+00
50%	1.923110e+06	2.787145e+05	NaN	1.125000e+04	7.104600e+04	8.054100e+04	1.638000e+03	1.123200e+05	1.000000e+00
75%	2.384280e+06	3.675140e+05	NaN	2.065842e+04	1.803600e+05	2.164185e+05	7.740000e+03	2.340000e+05	1.000000e+00

checking unique values for categorical columns and visualizing data

importing visualization libraries

```
#importing vizualisation libraries
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
```

unique value CONTRACT_TYPE

```
#unique value CONTRACT_TYPE
newtest_df.CONTRACT_TYPE.unique()
```

```
array(['Consumer loans', 'Cash loans', 'Revolving loans', 'XNA'],
      dtype=object)
```

count of unique value CONTRACT_TYPE

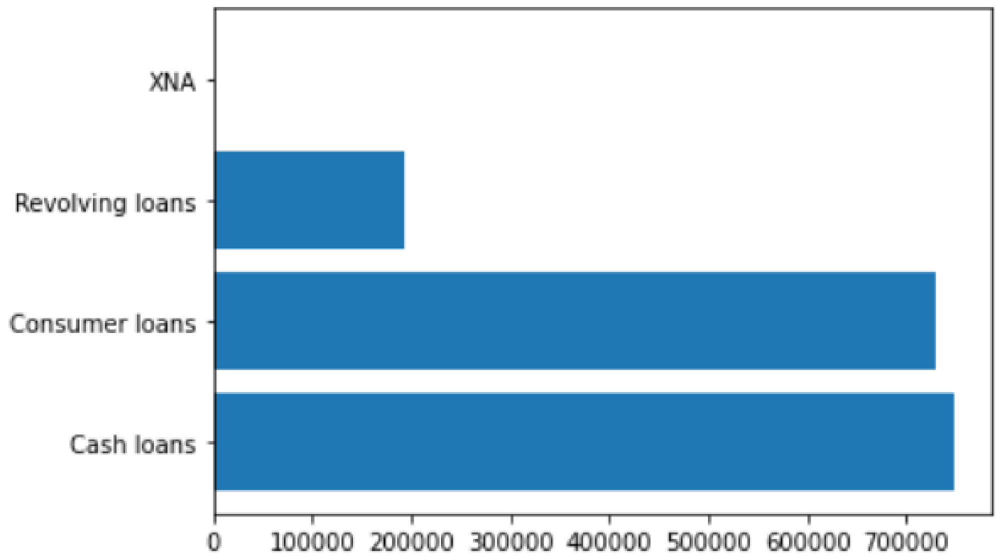
```
#count of unique value CONTRACT_TYPE
ct=newtest_df.CONTRACT_TYPE.value_counts()
ct
```

```
Cash loans         747553
Consumer loans     729151
Revolving loans    193164
XNA                   346
Name: CONTRACT_TYPE, dtype: int64
```

plotting bar chart for CONTRACT_TYPE categorical data

```
#plotting bar chart for CONTRACT_TYPE categorical data
plt.barh(ct.index, ct)
```

```
<BarContainer object of 4 artists>
```

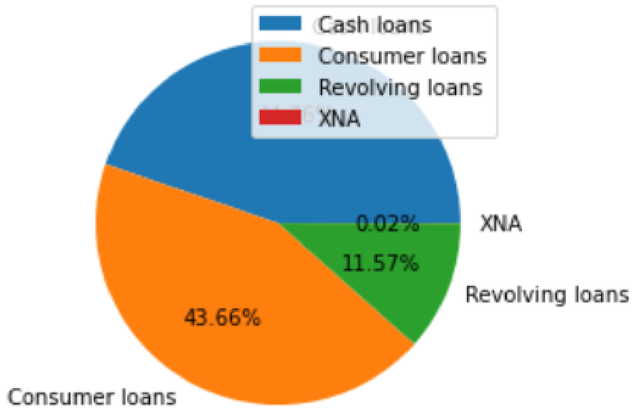


Conclusion:

plot clearly shows that Customer taking cash loans and customer loans are more than compared to those taking revolving loans

plotting pie chart for CONTRACT_TYPE categorical data

```
#plotting pie chart for CONTRACT_TYPE categorical data
plt.pie(x=ct, labels=ct.index, autopct='%1.2f%%')
plt.legend()
plt.show()
```



Conclusion:

Client taking cash loan more

unique value CLIENT_TYPE

```
#unique value CLIENT_TYPE
newtest_df.CLIENT_TYPE.unique()
```

```
array(['Repeater', 'New', 'Refreshed', 'XNA'], dtype=object)
```

count of unique value CLIENT_TYPE

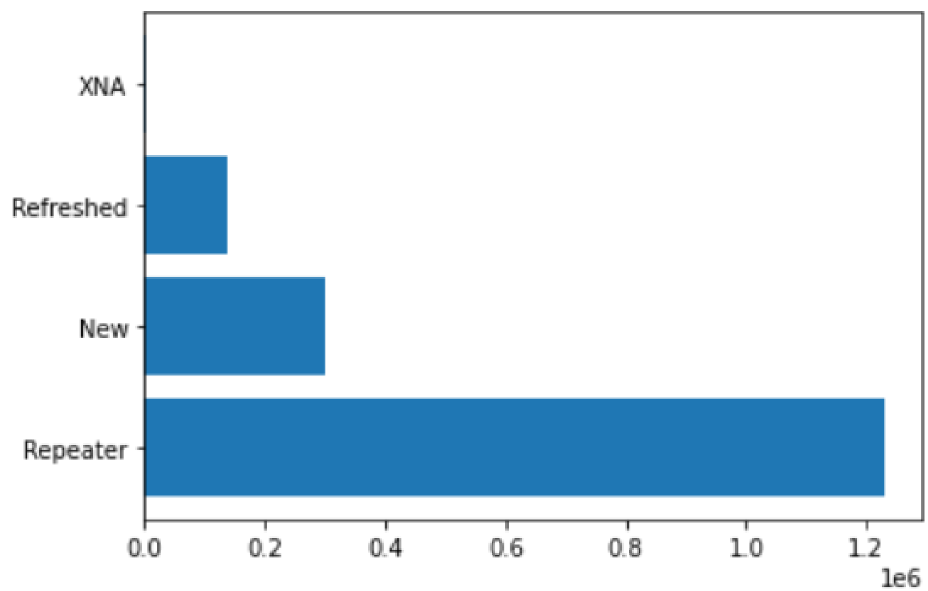
```
#count of unique value CLIENT_TYPE
clt= newtest_df.CLIENT_TYPE.value_counts()
clt
```

```
Repeater     1231261
New           301363
Refreshed     135649
XNA             1941
Name: CLIENT_TYPE, dtype: int64
```

plotting bar chart for categorical data

```
#plotting bar chart for categorical data
plt.barh(clt.index, clt)
```

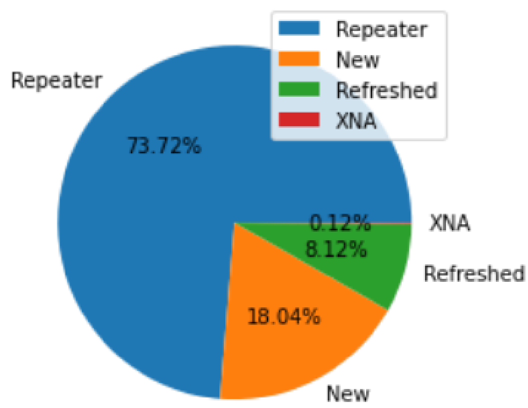
```
<BarContainer object of 4 artists>
```



Conclusion:

From bar graph is concluded that clients who are repeater are maximum

```
plt.pie(x=clt, labels=clt.index, autopct='%1.2f%%')
plt.legend()
plt.show()
```



Conclusion:

From pie graph is concluded that clients who are repeater are maximum

unique value GOODS_CATEGORY

```
#unique value GOODS_CATEGORY
newtest_df.GOODS_CATEGORY.unique()
```

```
array(['Mobile', 'XNA', 'Consumer Electronics', 'Construction Materials',
       'Auto Accessories', 'Photo / Cinema Equipment', 'Computers',
       'Audio/Video', 'Medicine', 'Clothing and Accessories', 'Furniture',
       'Sport and Leisure', 'Homewares', 'Gardening', 'Jewelry',
       'Vehicles', 'Education', 'Medical Supplies', 'Other',
       'Direct Sales', 'Office Appliances', 'Fitness', 'Tourism',
       'Insurance', 'Additional Service', 'Weapon', 'Animals',
       'House Construction'], dtype=object)
```

count of unique value GOODS_CATEGORY

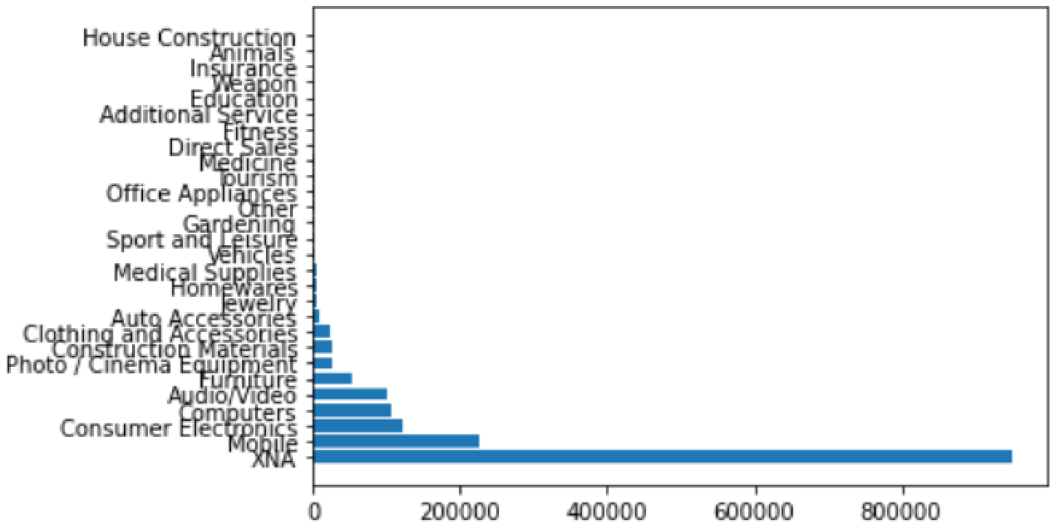
```
#count of unique value GOODS_CATEGORY
gt=newtest_df.GOODS_CATEGORY.value_counts()
gt
```

```
XNA                           950809
Mobile                        224708
Consumer Electronics          121576
Computers                     105769
Audio/Video                    99441
Furniture                      53656
Photo / Cinema Equipment       25021
Construction Materials         24995
Clothing and Accessories       23554
Auto Accessories                7381
Jewelry                         6290
Homewares                       5023
Medical Supplies                3843
Vehicles                        3370
Sport and Leisure               2981
Gardening                       2668
Other                           2554
Office Appliances               2333
Tourism                         1659
Medicine                        1550
```

plotting bar chart for GOODS_CATEGORY categorical data

```
#plotting bar chart for GOODS_CATEGORY categorical data
plt.barh(gt.index, gt)
```

```
<BarContainer object of 28 artists>
```



Conclusion:

It can be concluded that clients who are taking loan for goods category of XNA are maximum plotting pie chart for GOODS_CATEGORY categorical data

```
#plotting pie chart for GOODS_CATEGORY categorical data
plt.pie(x=gt, labels=gt.index, autopct='%1.2f%%')
plt.legend()
plt.show()
```

Conclusion:

People taking loans for electronics equipment are more as compared to people taking loans for house construction or insurance.

unique value PORTFOLIO

```
#unique value PORTFOLIO
newtest_df.PORTFOLIO.unique()
```

```
array(['POS', 'Cash', 'XNA', 'Cards', 'Cars'], dtype=object)
```

count of unique value PORTFOLIO

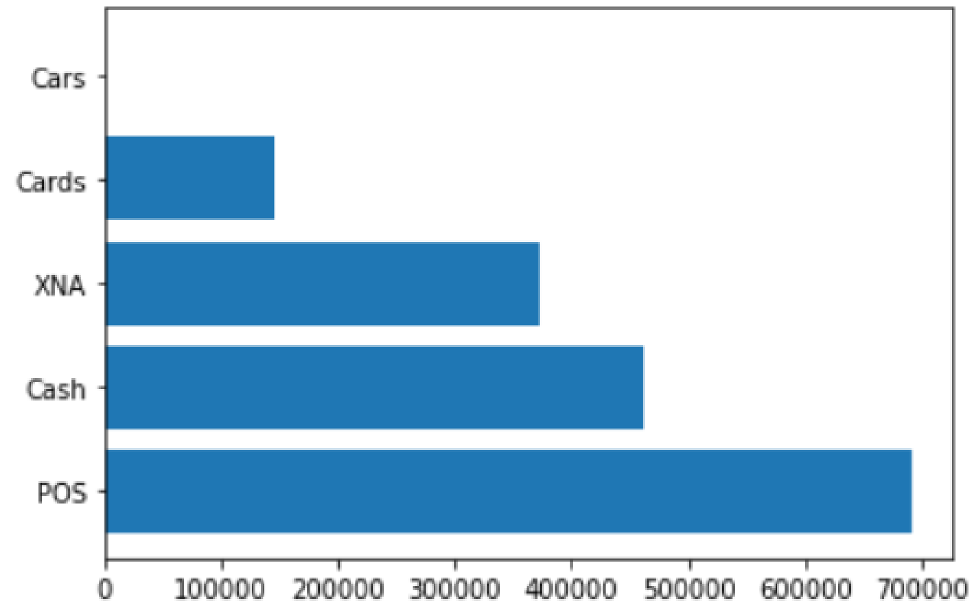
```
#count of unique value PORTFOLIO
pf=newtest_df.PORTFOLIO.value_counts()
pf
```

```
POS      691011
Cash     461563
XNA      372230
Cards    144985
Cars        425
Name: PORTFOLIO, dtype: int64
```

plotting bar chart for PORTFOLIO categorical data

```
#plotting bar chart for PORTFOLIO categorical data
plt.barh(pf.index, pf)
```

```
<BarContainer object of 5 artists>
```



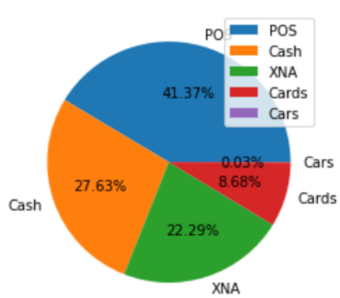
Conclusion:

It is concluded that POS (point-of-sale) type loan category are maximum

POS financing is **a broad term that describes methods for giving shoppers flexible, pay-over-time installment options**.

plotting pie chart for PORTFOLIO categorical data

```
#plotting pie chart for PORTFOLIO categorical data
plt.pie(x=pf, labels=pf.index, autopct='%1.2f%%')
plt.legend()
plt.show()
```



Conclusion:

unique value CHANNEL_TYPE

```
#unique value CHANNEL_TYPE
newtest_df.CHANNEL_TYPE.unique()
```

```
array(['Country-wide', 'Contact center', 'Credit and cash offices',
       'Stone', 'Regional / Local', 'AP+ (Cash loan)',
       'Channel of corporate sales', 'Car dealer'], dtype=object)
```

count of unique value CHANNEL_TYPE

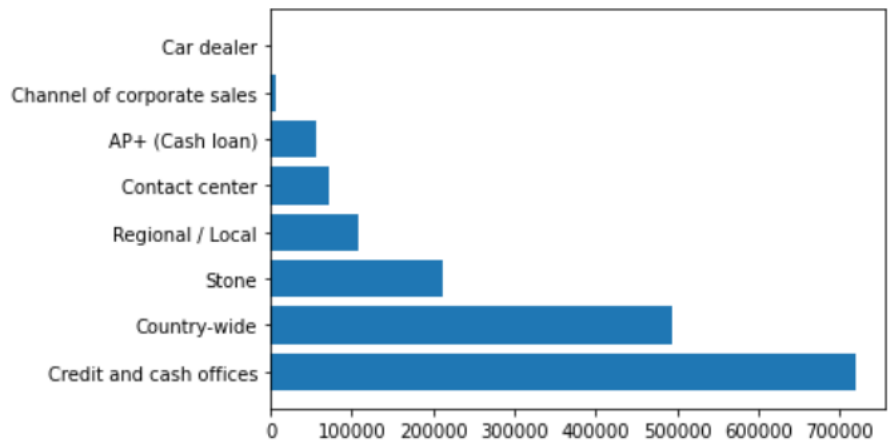
```
#count of unique value CHANNEL_TYPE
cht=newtest_df.CHANNEL_TYPE.value_counts()
cht
```

```
Credit and cash offices       719968
Country-wide                  494690
Stone                         212083
Regional / Local              108528
Contact center                 71297
AP+ (Cash loan)                57046
Channel of corporate sales      6150
Car dealer                       452
Name: CHANNEL_TYPE, dtype: int64
```

plotting bar chart for CHANNEL_TYPE categorical data

```
#plotting bar chart for CHANNEL_TYPE categorical data
plt.barh(cht.index, cht)
```

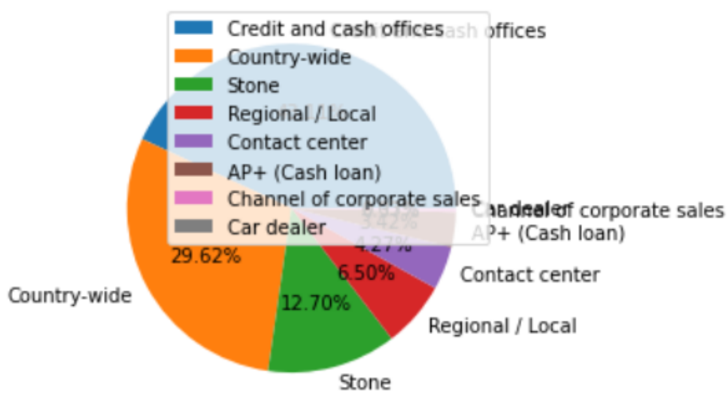
```
<BarContainer object of 8 artists>
```



Conclusion:

plotting pie chart for CHANNEL_TYPE categorical data

```
#plotting pie chart for CHANNEL_TYPE categorical data
plt.pie(x=cht, labels=cht.index, autopct='%1.2f%%')
plt.legend()
plt.show()
```



unique value SELLER_INDUSTRY

```
#unique value SELLER_INDUSTRY
newtest_df.SELLER_INDUSTRY.unique()
```

```
array(['Connectivity', 'XNA', 'Consumer electronics', 'Industry',
       'Clothing', 'Furniture', 'Construction', 'Jewelry',
       'Auto technology', 'MLM partners', 'Tourism'], dtype=object)
```

count of unique value SELLER_INDUSTRY

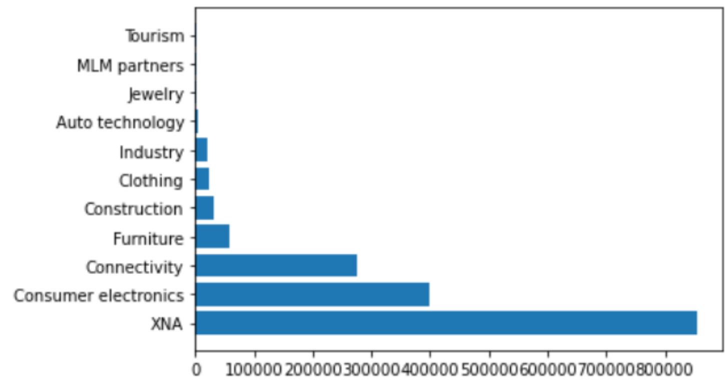
```
#count of unique value SELLER_INDUSTRY
si=newtest_df.SELLER_INDUSTRY.value_counts()
si
```

```
XNA                     855720
Consumer electronics    398265
Connectivity            276029
Furniture                57849
Construction             29781
Clothing                 23949
Industry                 19194
Auto technology           4990
Jewelry                   2709
MLM partners              1215
Tourism                    513
Name: SELLER_INDUSTRY, dtype: int64
```

plotting bar chart for SELLER_INDUSTRY categorical data

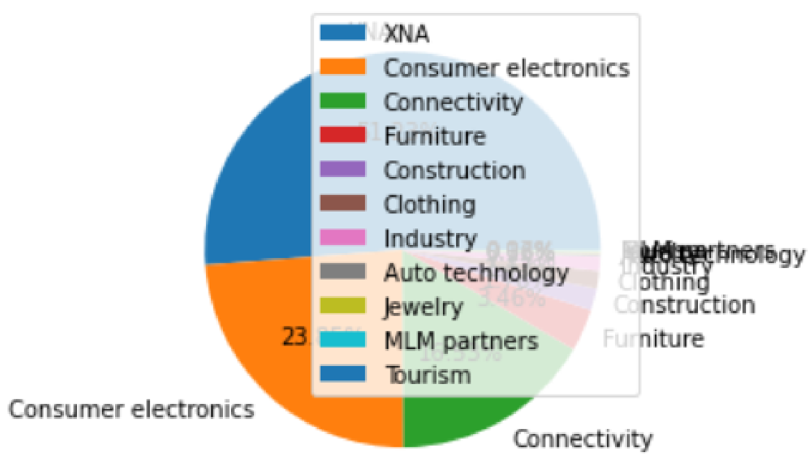
```
#plotting bar chart for SELLER_INDUSTRY categorical data
plt.barh(si.index, si)
```

```
<BarContainer object of 11 artists>
```



plotting pie chart for SELLER_INDUSTRY categorical data

```
#plotting pie chart for SELLER_INDUSTRY categorical data
plt.pie(x=si, labels=si.index, autopct='%1.2f%%')
plt.legend()
plt.show()
```



Conclusion:

unique value YIELD_GROUP

```
#unique value YIELD_GROUP
newtest_df.YIELD_GROUP.unique()
```

```
array(['middle', 'low_action', 'high', 'low_normal', 'XNA'], dtype=object)
```

count of unique value YIELD_GROUP

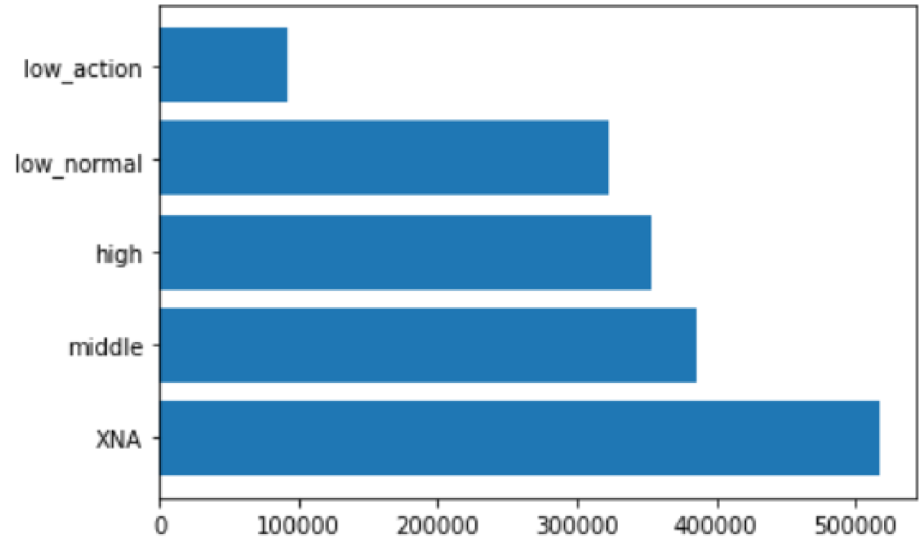
```
#count of unique value YIELD_GROUP
yg=newtest_df.YIELD_GROUP.value_counts()
yg
```

```
XNA           517215
middle        385532
high          353331
low_normal    322095
low_action     92041
Name: YIELD_GROUP, dtype: int64
```

plotting bar chart for YIELD_GROUP categorical data

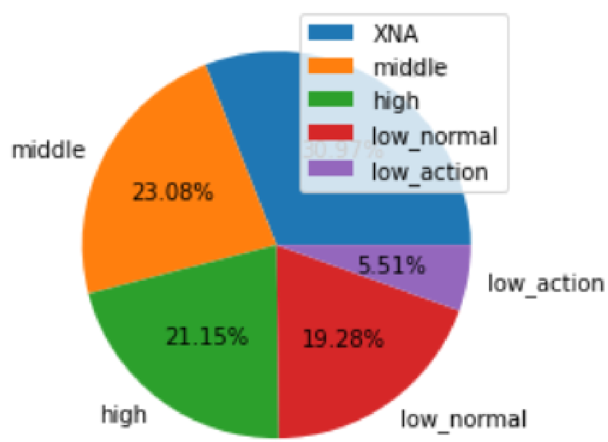
```python
#plotting bar chart for YIELD_GROUP categorical data
plt.barh(yg.index, yg)
```

```
<BarContainer object of 5 artists>
```



plotting pie chart for YIELD_GROUP categorical data

```python
#plotting pie chart for YIELD_GROUP categorical data
plt.pie(x=yg, labels=yg.index, autopct='%1.2f%%')
plt.legend()
plt.show()
```



Conclusion:

--> for categorical data

checking the data types of new test data to find out the continuous categorical data to find the correlation

```
#checking the data types of newtest data to find out the continuous categorical data to find the correlation
newtest_df.dtypes
```

```
PREV_CustID                 int64
CURR_CustID                 int64
CONTRACT_TYPE              object
AMT_ANNUITY               float64
AMT_APPLICATION           float64
AMT_CREDIT                float64
AMT_DOWN_PAYMENT          float64
AMT_GOODS_PRICE           float64
LASTAPPL_PerDAY             int64
RATE_DOWN_PAYMENT         float64
RATE_INTEREST_PRIMARY     float64
RATE_INTEREST_PRIVILEGED  float64
CASH_LOAN_PURPOSE          object
CONTRACT_STATUS            object
PAYMENT_TYPE               object
CODE_REJECT_REASON         object
CLIENT_TYPE                object
GOODS_CATEGORY             object
PORTFOLIO                  object
CHANNEL_TYPE               object
SELLER_INDUSTRY            object
YIELD_GROUP                object
```

taking only continuous data into account

```
#taking only continuous data into account
cat2_df=newtest_df[['CURR_CustID','AMT_ANNUITY', 'AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE',
                    'FIRST_DUEDay','LAST_DUEDay','DAYS_TERMINATION','INSURED_ON_APPROVAL']]
```

```
cat2_df.head()
```

	CURR_CustID	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	FIRST_DUEDay	LAST_DUEDay	DAYS_TERMINATION	INSURED_ON_APPROVAL
0	271877	1730.430	17145.0	17145.0	17145.0	-42.0	-42.0	-37.0	0.0
1	108129	25188.615	607500.0	679671.0	607500.0	-134.0	365243.0	365243.0	1.0
2	122040	15060.735	112500.0	136444.5	112500.0	-271.0	365243.0	365243.0	1.0
3	176158	47041.335	450000.0	470790.0	450000.0	-482.0	-182.0	-177.0	1.0
4	202054	31924.395	337500.0	404055.0	337500.0	NaN	NaN	NaN	NaN

Checking Corrilation Between columns of numeric continues data

```
#Checking Corrilation Between columns of numeric continues data
cat2_df.corr()
```

	CURR_CustID	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	FIRST_DUEDay	LAST_DUEDay	DAYS_TERMINATION	INS
CURR_CustID	1.000000	0.000577	0.000280	0.000195	0.000369	-0.000757	-0.000318	-0.000020	
AMT_ANNUITY	0.000577	1.000000	0.808872	0.816429	0.820895	-0.053295	0.082659	0.068022	
AMT_APPLICATION	0.000280	0.808872	1.000000	0.975824	0.999884	-0.049532	0.172627	0.148618	
AMT_CREDIT	0.000195	0.816429	0.975824	1.000000	0.993087	0.002881	0.224829	0.214320	
AMT_GOODS_PRICE	0.000369	0.820895	0.999884	0.993087	1.000000	-0.021062	0.211696	0.209296	
FIRST_DUEDay	-0.000757	-0.053295	-0.049532	0.002881	-0.021062	1.000000	0.401838	0.323608	
LAST_DUEDay	-0.000318	0.082659	0.172627	0.224829	0.211696	0.401838	1.000000	0.927990	
DAYS_TERMINATION	-0.000020	0.068022	0.148618	0.214320	0.209296	0.323608	0.927990	1.000000	
INSURED_ON_APPROVAL	0.000876	0.283080	0.259219	0.263932	0.243400	-0.119048	0.012560	-0.003065	

dropping the columns having null vvalues and creating new dataframe

```
#dropping the columns having null vvalues and creating new dataframe
cat2_df[['CURR_CustID','AMT_ANNUITY', 'AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE']]
```

	CURR_CustID	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE
0	271877	1730.430	17145.0	17145.0	17145.0
1	108129	25188.615	607500.0	679671.0	607500.0
2	122040	15060.735	112500.0	136444.5	112500.0
3	176158	47041.335	450000.0	470790.0	450000.0
4	202054	31924.395	337500.0	404055.0	337500.0
...	...	...	...	...	...
1670209	352015	14704.290	267295.5	311400.0	267295.5
1670210	334635	6622.020	87750.0	64291.5	87750.0
1670211	249544	11520.855	105237.0	102523.5	105237.0
1670212	400317	18821.520	180000.0	191880.0	180000.0

```
cat3_df=cat2_df[['CURR_CustID','AMT_ANNUITY', 'AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE']].corr()
cat3_df
```

	CURR_CustID	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE
CURR_CustID	1.000000	0.000577	0.000280	0.000195	0.000369
AMT_ANNUITY	0.000577	1.000000	0.808872	0.816429	0.820895
AMT_APPLICATION	0.000280	0.808872	1.000000	0.975824	0.999884
AMT_CREDIT	0.000195	0.816429	0.975824	1.000000	0.993087
AMT_GOODS_PRICE	0.000369	0.820895	0.999884	0.993087	1.000000

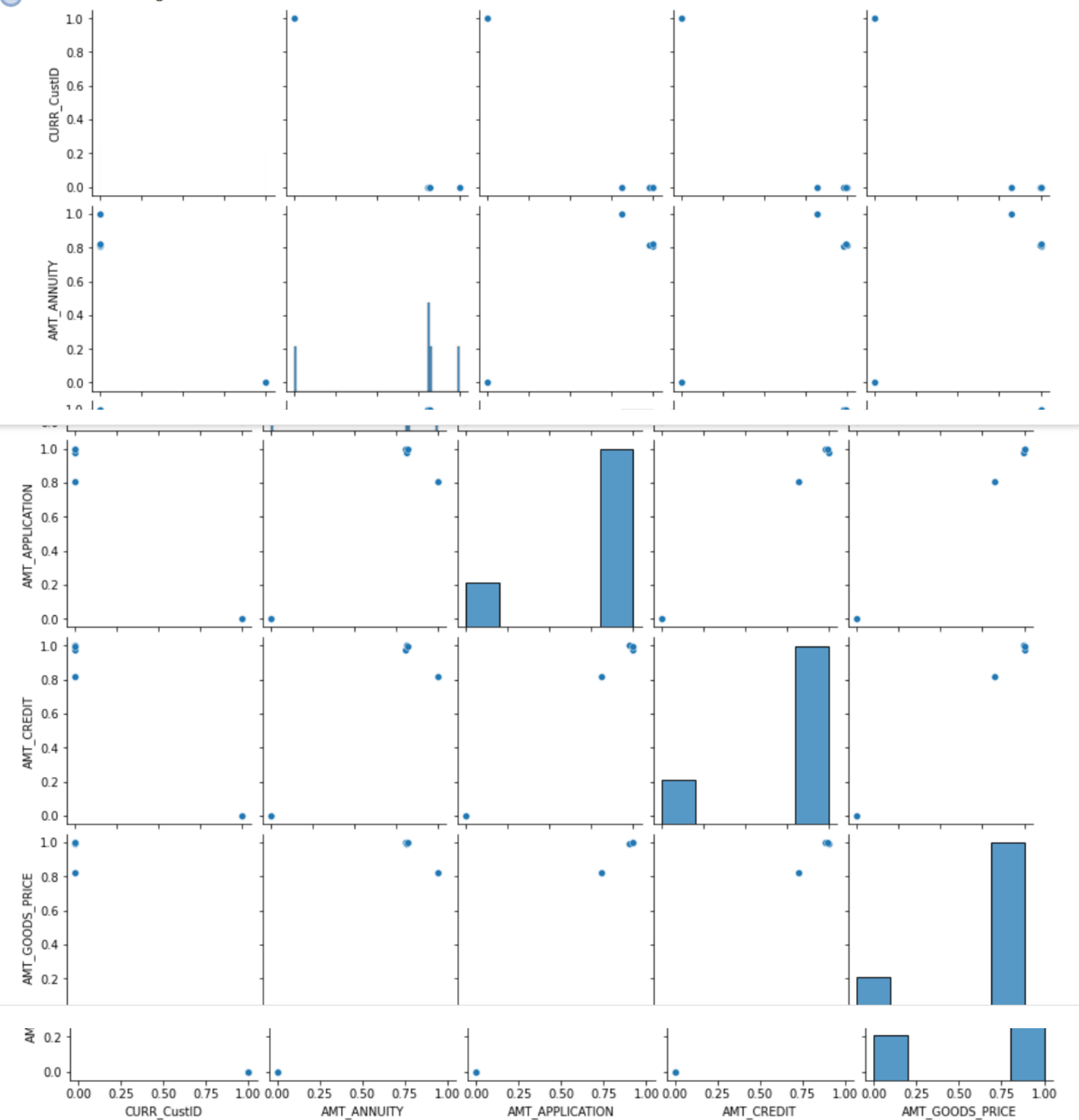
Conclusion-->

1. AMT_ANNUITY is strongly correlated with
- AMT_APPLICATION by corelation coefficient 0.808872
- AMT_CREDIT by corelation coefficient 0.816429
- AMT_GOODS_PRICE by corelation coefficient 0.820895
2. AMT_APPLICATION is strongly correlated with
- AMT_CREDIT by corelation coefficient 0.975824
- AMT_GOODS_PRICE by corelation coefficient 0.999884
3. AMT_CREDIT is strongly correlated with
- AMT_GOODS_PRICE by corelation coefficient 0.993087

ploting pairplot for continues data to check the retaion between columns

creating pairplot graph

```
<seaborn.axisgrid.PairGrid at 0x7f4aa5bddfa0>
```



Conclusion:

Handling Outliers

Checking outliers for INCOME

```
[ ] #Checking outliers for INCOME
    cat3_df.AMT_APPLICATION.describe()
```

```
count    5.000000
mean     0.756972
std      0.430477
min      0.000280
25%      0.808872
50%      0.975824
75%      0.999884
max      1.000000
Name: AMT_APPLICATION, dtype: float64
```

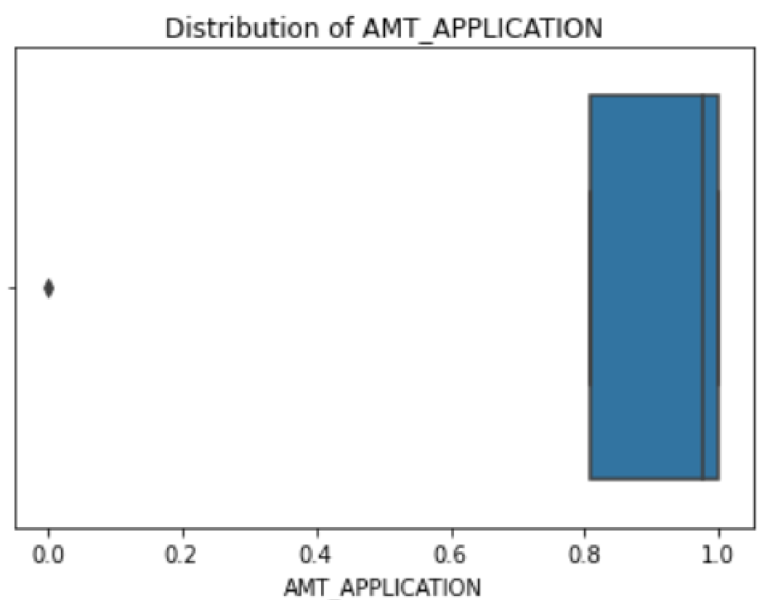
```
sns.boxplot(cat3_df.AMT_APPLICATION)
plt.title('Distribution of AMT_APPLICATION')
plt.show()
```

```
#Checking outliers for INCOME
cat3_df.AMT_APPLICATION.describe()
```

```
count    5.000000
mean     0.756972
std      0.430477
min      0.000280
25%      0.808872
50%      0.975824
75%      0.999884
max      1.000000
Name: AMT_APPLICATION, dtype: float64
```

```
sns.boxplot(cat3_df.AMT_APPLICATION)
plt.title('Distribution of AMT_APPLICATION')
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: F
  warnings.warn(
```



Conclusion:

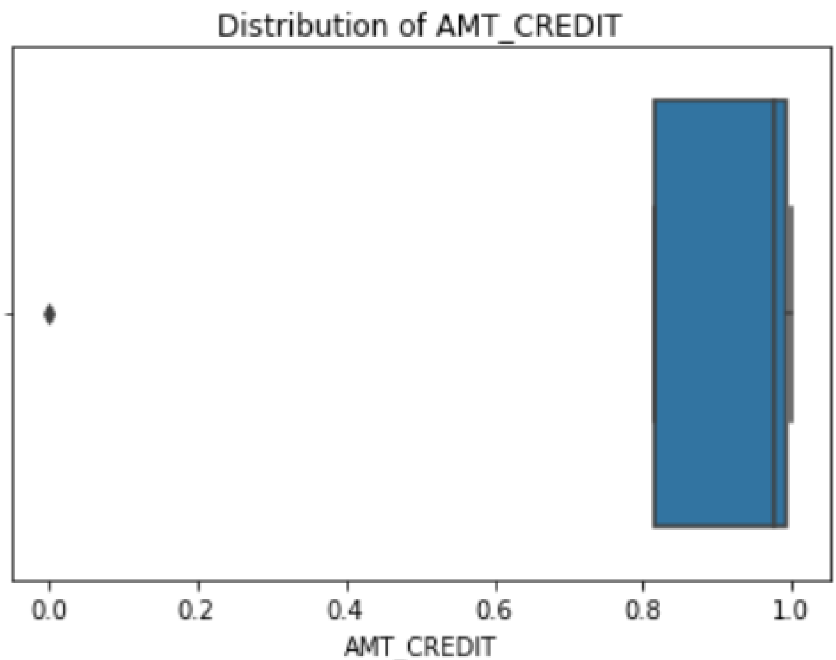
AMT_CREDIT
Checking outliers for AMT_CREDIT

```
#Checking outliers for AMT_CREDIT
cat3_df.AMT_CREDIT.describe()
```

```
count    5.000000
mean     0.757107
std      0.429813
min      0.000195
25%      0.816429
50%      0.975824
75%      0.993087
max      1.000000
Name: AMT_CREDIT, dtype: float64
```

```
sns.boxplot(cat3_df.AMT_CREDIT)
plt.title('Distribution of AMT_CREDIT')
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.p
  warnings.warn(
```



Conclusion:

AMT_GOODS_PRICE

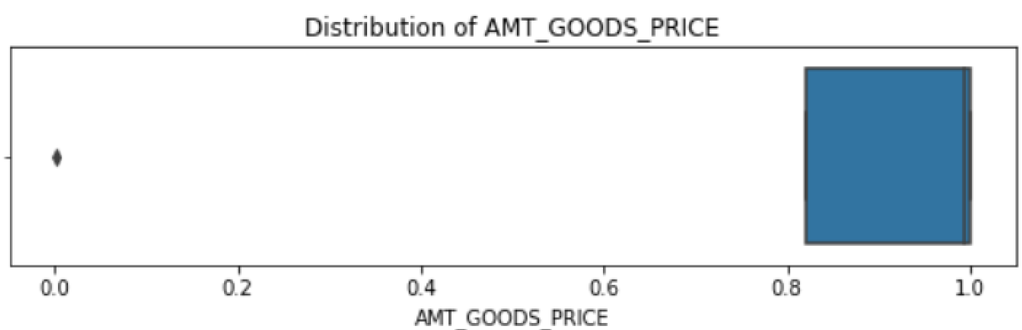
Checking outliers for AMT_GOODS_PRICE

```
#Checking outliers for AMT_GOODS_PRICE
cat3_df.AMT_GOODS_PRICE.describe()
```

```
count    5.000000
mean     0.762847
std      0.433065
min      0.000369
25%      0.820895
50%      0.993087
75%      0.999884
max      1.000000
Name: AMT_GOODS_PRICE, dtype: float64
```

```
plt.figure(figsize=(9,2))
sns.boxplot(cat3_df.AMT_GOODS_PRICE)
plt.title('Distribution of AMT_GOODS_PRICE')
plt.show()
```

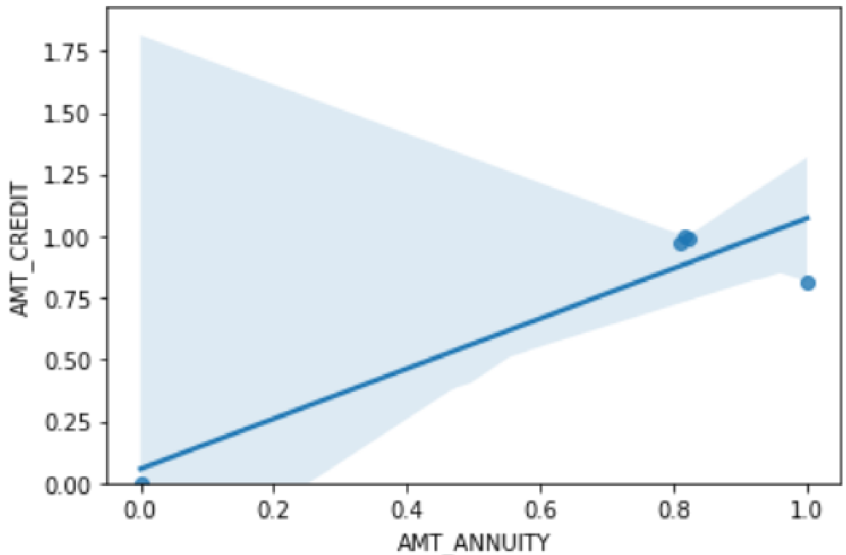
```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWa
  warnings.warn(
```



Conclusion:

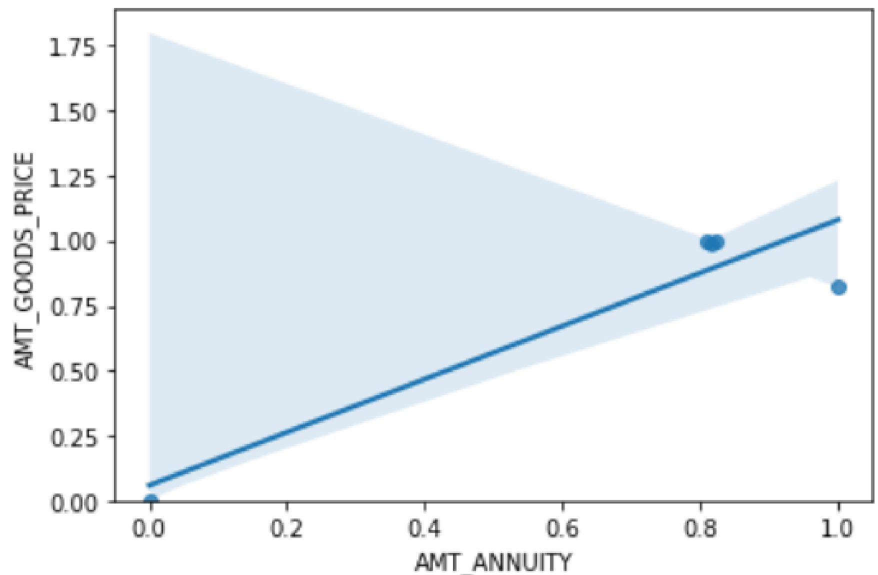
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

```
sns.regplot(x="AMT_ANNUITY", y="AMT_CREDIT", data=cat3_df)
plt.ylim(0,)
plt.show()
```



Conclusion:

```python
sns.regplot(x="AMT_ANNUITY", y="AMT_GOODS_PRICE", data=cat3_df)
plt.ylim(0,)
plt.show()
```



Conclusion:

Filtering data on bases of Loan amount for top 10 loan amounts

Q. Filterring data on bases of Loan amount for top 10 loan amounts

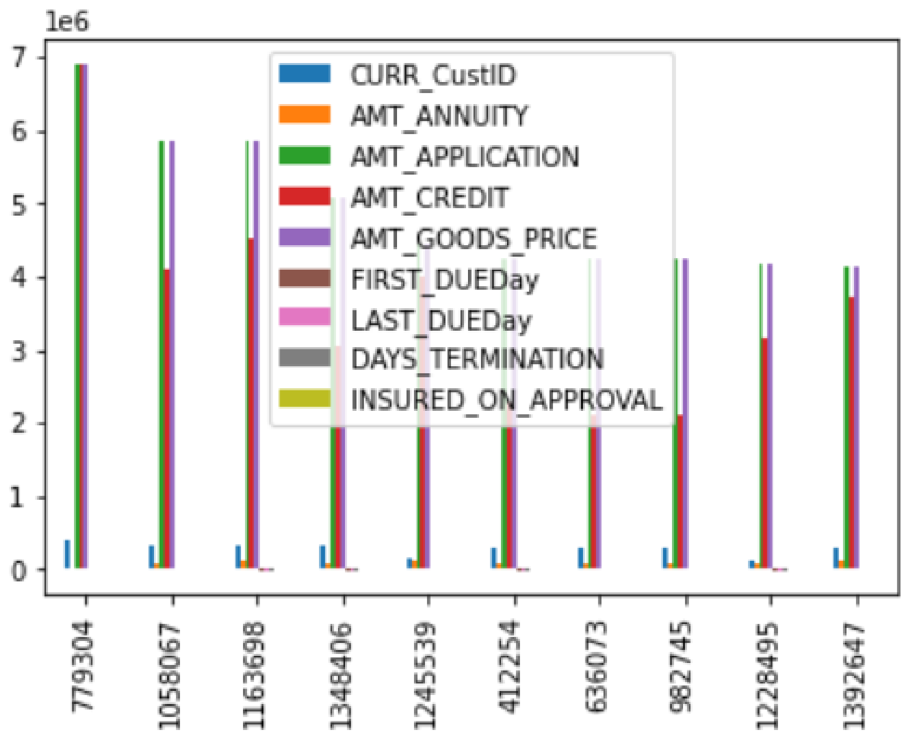
```python
df_amt=cat2_df.nlargest(10,columns='AMT_APPLICATION')
df_amt.head()
```

	CURR_CustID	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	FIRST_DUEDay	LAST_DUEDay	DAYS_TERMINATION	INSURED_ON_APPROVAL
779304	412009	NaN	6905160.0	6905160.0	6905160.0	NaN	NaN	NaN	NaN
1058067	346243	103498.650	5850000.0	4095000.0	5850000.0	NaN	NaN	NaN	NaN
1163698	346243	113979.690	5850000.0	4509688.5	5850000.0	-2443.0	-1513.0	-1505.0	1.0
1348406	324681	83707.830	5085000.0	3051000.0	5085000.0	-2598.0	-2598.0	-2591.0	0.0
1245539	173326	119443.005	4455000.0	4009500.0	4455000.0	NaN	NaN	NaN	NaN

Plotting graph for approved loan amount

```
#Plotting graph for approved loan amount
df_amt.plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4a9ef86d00>
```



conclusion: