Exploratory Data Analysis (EDA)

Introduction

This case study aims to give us an idea of applying EDA in a real business scenario. In this case study, apart from applying the techniques that we have learnt in the EDA module, we 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.

Importing required libraries:

```
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 Data from CSV

```
[ ]  df=pd.read_csv("application_data (1).csv")
```

```
[ ]  pa_df=pd.read_csv("previous_application.csv")
```

```
[ ]  col_df=pd.read_csv("columns_description.csv", encoding='latin1')
```

Viewing the Dataset

1.Viewing the Dataset of application_data

```
#checking first five rows and columns of the application_data dataset
df.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

2.Viewing the Dataset of previous_application

```
#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	HOUR_A
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

3.Viewing the Dataset of Columns_Description

```
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 dataset

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

```
(307511, 122)
```

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

```
(1670214, 37)
```

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

```
(160, 5)
```

Checking the dimension of 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
pa_df.ndim
```

```
2
```

Checking the datatype of dataset

```
#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
```

```python
#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
RATE_DOWN_PAYMENT             float64
RATE_INTEREST_PRIMARY         float64
RATE_INTEREST_PRIVILEGED      float64
NAME_CASH_LOAN_PURPOSE         object
NAME_CONTRACT_STATUS           object
DAYS_DECISION                   int64
NAME_PAYMENT_TYPE              object
CODE_REJECT_REASON             object
NAME_TYPE_SUITE                object
NAME_CLIENT_TYPE               object
```

Statistical summary of all numeric-typed (int, float) columns(from describe command)

```
df.describe()
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE	D
count	31081.000000	31081.000000	31081.000000	3.108100e+04	3.108100e+04	31081.000000	3.105500e+04	31081.000000	31
mean	118092.231524	0.080274	0.415881	1.720948e+05	6.001104e+05	27150.678115	5.394902e+05	0.020760	-16
std	10423.676420	0.271721	0.722285	6.700652e+05	4.030235e+05	14675.416544	3.704613e+05	0.013759	4
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	2052.000000	4.500000e+04	0.000533	-25
25%	109063.000000	0.000000	0.000000	1.125000e+05	2.700000e+05	16452.000000	2.385000e+05	0.010006	-19
50%	118135.000000	0.000000	0.000000	1.467000e+05	5.172660e+05	24939.000000	4.500000e+05	0.018850	-15
75%	127119.000000	0.000000	1.000000	2.025000e+05	8.086500e+05	34681.500000	6.795000e+05	0.028663	-12
max	136075.000000	1.000000	9.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.072508	-7

8 rows × 106 columns

```
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	NF
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

Data Preparation

Creating a new dataframe using the columns of df

```
new_df=df[['SK_ID_CURR', 'AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY','AMT_GOODS_PRICE','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE','NAME_FAMILY_S
```

```
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_HOUSIN
0	100002	202500.0	406597.5	24700.5	351000.0	Working	Secondary / secondary special	Single / not married	House / apa
1	100003	270000.0	1293502.5	35698.5	1129500.0	State servant	Higher education	Married	House / apa
2	100004	67500.0	135000.0	6750.0	135000.0	Working	Secondary / secondary special	Single / not married	House / apa
3	100006	135000.0	312682.5	29686.5	297000.0	Working	Secondary / secondary special	Civil marriage	House / apa
4	100007	121500.0	513000.0	21865.5	513000.0	Working	Secondary / secondary special	Single / not married	House / apa

5 rows × 28 columns

Checking the shape of new dataframe

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

```
(307511, 28)
```

Checking the names of columns of new dataframe

```
#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

```
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',
       'Perman_add_NOT_cont_REGION', 'perman_add_NOT_WORK_add',
       'ORGANIZATION_TYPE', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
       'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'LAST_PHONE_CHANGE']
```

```
]  new_df.columns=list
```

```
]  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', 'Perman_add_NOT_cont_REGION',
       'perman_add_NOT_WORK_add', 'ORGANIZATION_TYPE', 'EXT_SOURCE_1',
       'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG',
       'LAST_PHONE_CHANGE'],
      dtype='object')
```

Checking for columns with null values

Using a for loop in Python, we can quickly figure out the number of missing values in each column. As mentioned above, "True" represents a missing value and "False" means the value is present in the dataset. In the body of the for loop the method ".value_counts()" counts the number of "True" values.

```
def null_values(new_df):
    return round((new_df.isnull().sum()*100/len(new_df)).sort_values(ascending = False),2)
```

Displaying columns with Percentage of null values

```
#displaying columns with Percentage of 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
LAST_PHONE_CHANGE              0.00
ORGANIZATION_TYPE              0.00
perman_add_NOT_WORK_add        0.00
Perman_add_NOT_cont_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
age                            0.00
curr_HOUSING_TYPE              0.00
```

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

```
missing values :  BASEMENTAREA_AVG            179943
EXT_SOURCE_1                 173378
APARTMENTS_AVG               156061
OCCUPATION_TYPE               96391
EXT_SOURCE_3                  60965
EXT_SOURCE_2                    660
GOODS_PRICE                     278
ANNUITY                          12
Family_MEMBERS_no                 2
LAST_PHONE_CHANGE                 1
ORGANIZATION_TYPE                 0
perman_add_NOT_WORK_add           0
Perman_add_NOT_cont_REGION        0
REGION_CLIENT_CITY                0
REGION_CLIENT                     0
Cust_ID                           0
INCOME                            0
MOBIL_given                       0
DAYS_ID_PUBLISH                   0
REGISTRATION_change               0
Work_Exp                          0
age                               0
curr_HOUSING_TYPE                 0
```

Summary:

Columns having maximum null values

BASEMENTAREA_AVG 58.52 EXT_SOURCE_1 56.38

APARTMENTS_AVG 50.75

OCCUPATION_TYPE 31.35

EXT_SOURCE_3 19.83

these are the columns having maximum null values

Replacing null values with NAN

We can deal with missing data by the following ways:

1.Drop data
   a. Drop the whole row
   b. Drop the whole column

2.Replace data
   a. Replace it by mean
   b. Replace it by frequency
   c. Replace it based on other functions

Whole columns should be dropped only if most entries in the column are empty. In our dataset, none of the columns are empty enough to drop entirely.
We have some freedom in choosing which method to replace data; however, some methods may seem more reasonable than others.

```
#Replacing null values with NAN for all the columns
new_df=new_df.replace(np.nan, 'NAN')
```

```
#columns after replacing with Nan
print("missing values : ",new_df.isna().sum().sort_values(ascending = False))
```

```
missing values :  Cust_ID                         0
INCOME                      0
BASEMENTAREA_AVG            0
APARTMENTS_AVG              0
EXT_SOURCE_3                0
EXT_SOURCE_2                0
EXT_SOURCE_1                0
ORGANIZATION_TYPE           0
perman_add_NOT_WORK_add     0
Perman_add_NOT_cont_REGION  0
REGION_CLIENT_CITY          0
REGION_CLIENT               0
Family_MEMBERS_no           0
OCCUPATION_TYPE             0
EMAIL_given                 0
MOBIL_given                 0
```

We replaced all empty columns with NAN.

Understanding of the variables Categorical variables

```
#taking info to check Column name,  Non-Null Count,   Dtype and sape of data (307511X28 )
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 28 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   Cust_ID                       307511 non-null  int64
 1   INCOME                        307511 non-null  float64
 2   Loan_AMT                      307511 non-null  float64
 3   ANNUITY                       307511 non-null  object
 4   GOODS_PRICE                   307511 non-null  object
 5   INCOME_TYPE                   307511 non-null  object
 6   EDUCATION                     307511 non-null  object
 7   FAMILY_STATUS                 307511 non-null  object
 8   curr_HOUSING_TYPE             307511 non-null  object
 9   age                           307511 non-null  int64
 10  Work_Exp                      307511 non-null  int64
 11  REGISTRATION_change           307511 non-null  float64
 12  DAYS_ID_PUBLISH               307511 non-null  int64
 13  MOBIL_given                   307511 non-null  int64
 14  EMAIL_given                   307511 non-null  int64
 15  OCCUPATION_TYPE               307511 non-null  object
```

Checking statistical summary of all the numeric columns

```
#statisticsal summary of all the numric columns
new_df.describe()
```

	Cust_ID	INCOME	Loan_AMT	age	Work_Exp	REGISTRATION_change	DAYS_ID_PUBLISH	MOBIL_given	EMAIL_given
count	307511.000000	3.075110e+05	3.075110e+05	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000
mean	278180.518577	1.687979e+05	5.990260e+05	-16036.995067	63815.045904	-4986.120328	-2994.202373	0.999997	0.056720
std	102790.175348	2.371231e+05	4.024908e+05	4363.988632	141275.766519	3522.886321	1509.450419	0.001803	0.231307
min	100002.000000	2.565000e+04	4.500000e+04	-25229.000000	-17912.000000	-24672.000000	-7197.000000	0.000000	0.000000
25%	189145.500000	1.125000e+05	2.700000e+05	-19682.000000	-2760.000000	-7479.500000	-4299.000000	1.000000	0.000000
50%	278202.000000	1.471500e+05	5.135310e+05	-15750.000000	-1213.000000	-4504.000000	-3254.000000	1.000000	0.000000
75%	367142.500000	2.025000e+05	8.086500e+05	-12413.000000	-289.000000	-2010.000000	-1720.000000	1.000000	0.000000
max	456255.000000	1.170000e+08	4.050000e+06	-7489.000000	365243.000000	0.000000	0.000000	1.000000	1.000000

Checking correlation summary of all the numeric columns

```
#checking corrilation between columns of numeric data
new_df.corr()
```

	Cust_ID	INCOME	Loan_AMT	age	Work_Exp	REGISTRATION_change	DAYS_ID_PUBLISH	MOBIL_given	EMAIL_given
Cust_ID	1.000000	-0.001820	-0.000343	-0.001500	0.001366	-0.000973	-0.000384	0.002804	0.000281
INCOME	-0.001820	1.000000	0.156870	0.027261	-0.064223	0.027805	0.008506	0.000325	0.038378
Loan_AMT	-0.000343	0.156870	1.000000	-0.055436	-0.066838	0.009621	-0.006575	0.001436	0.016632
age	-0.001500	0.027261	-0.055436	1.000000	-0.615864	0.331912	0.272691	-0.003084	0.088208
Work_Exp	0.001366	-0.064223	-0.066838	-0.615864	1.000000	-0.210242	-0.272378	0.000818	-0.062112
REGISTRATION_change	-0.000973	0.027805	0.009621	0.331912	-0.210242	1.000000	0.101896	-0.000100	0.034388
DAYS_ID_PUBLISH	-0.000384	0.008506	-0.006575	0.272691	-0.272378	0.101896	1.000000	-0.002293	0.027505
MOBIL_given	0.002804	0.000325	0.001436	-0.003084	0.000818	-0.000100	-0.002293	1.000000	0.000442
EMAIL_given	0.000281	0.038378	0.016632	0.088208	-0.062112	0.034388	0.027505	0.000442	1.000000
REGION_CLIENT	-0.001075	-0.085465	-0.101776	0.009361	0.032750	0.080210	-0.005103	0.000186	-0.052063
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Checking statistical summary of all the non-numeric columns

```python
#statisticsal summary of all the non numric columns
new_df.describe(include='all')
```

	Cust_ID	INCOME	Loan_AMT	ANNUITY	GOODS_PRICE	INCOME_TYPE	EDUCATION	FAMILY_STATUS	curr_HOUSING_TYPE	age
count	307511.000000	3.075110e+05	3.075110e+05	307511.0	307511.0	307511	307511	307511	307511	307511.000000
unique	NaN	NaN	NaN	13673.0	1003.0	8	5	6	6	NaN
top	NaN	NaN	NaN	9000.0	450000.0	Working	Secondary / secondary special	Married	House / apartment	NaN
freq	NaN	NaN	NaN	6385.0	26022.0	158774	218391	196432	272868	NaN
mean	278180.518577	1.687979e+05	5.990260e+05	NaN	NaN	NaN	NaN	NaN	NaN	-16036.995067
std	102790.175348	2.371231e+05	4.024908e+05	NaN	NaN	NaN	NaN	NaN	NaN	4363.988632
min	100002.000000	2.565000e+04	4.500000e+04	NaN	NaN	NaN	NaN	NaN	NaN	-25229.000000
25%	189145.500000	1.125000e+05	2.700000e+05	NaN	NaN	NaN	NaN	NaN	NaN	-19682.000000
50%	278202.000000	1.471500e+05	5.135310e+05	NaN	NaN	NaN	NaN	NaN	NaN	-15750.000000
75%	367142.500000	2.025000e+05	8.086500e+05	NaN	NaN	NaN	NaN	NaN	NaN	12413.000000

Checking unique values for categorical columns and Visualizing data

Importing libraries

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

Unique income type

```python
#unique value INCOME_TYPE
new_df.INCOME_TYPE.unique()
```

```
array(['Working', 'State servant', 'Commercial associate', 'Pensioner',
       'Unemployed', 'Student', 'Businessman', 'Maternity leave'],
      dtype=object)
```

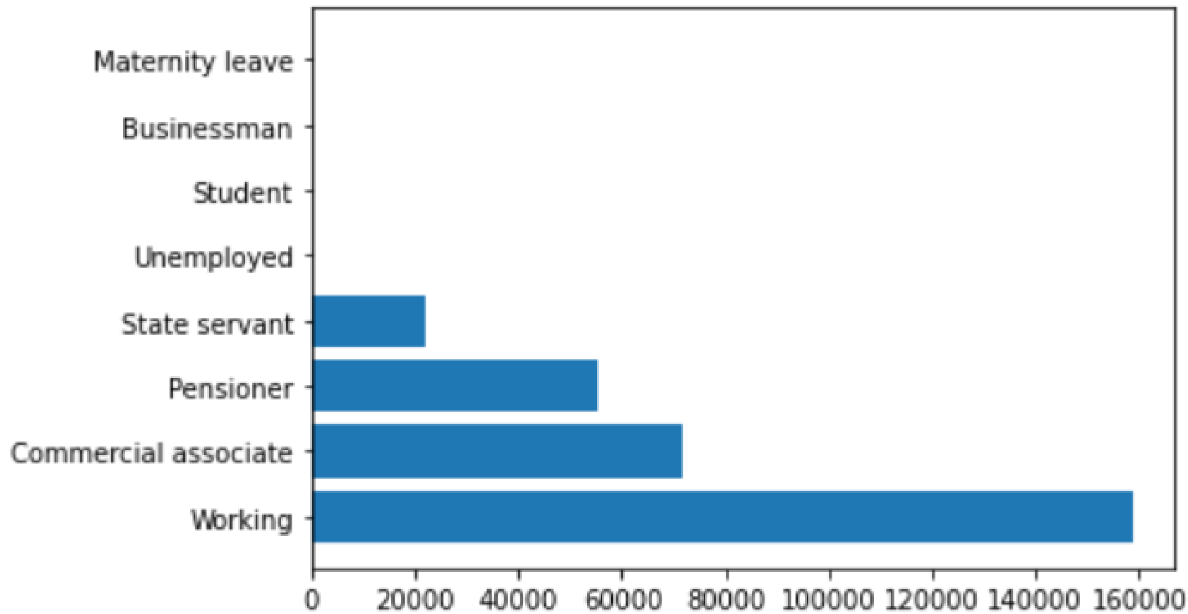
Counting and plotting unique income type

```python
a=new_df.INCOME_TYPE.value_counts()
a
```

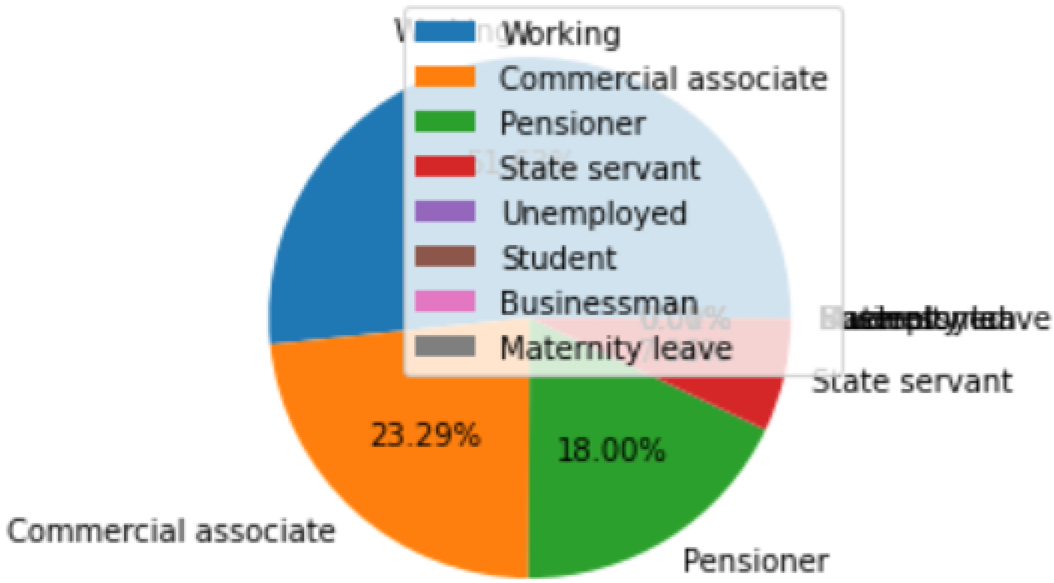
```
Working                 158774
Commercial associate     71617
Pensioner                55362
State servant            21703
Unemployed                  22
Student                     18
Businessman                 10
Maternity leave              5
Name: INCOME_TYPE, dtype: int64
```

```
plt.barh(a.index, a)
```

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



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



It can be concluded that clients with income type working class are maximum.

Unique education type

```
new_df.EDUCATION.unique()
```

```
array(['Secondary / secondary special', 'Higher education',
       'Incomplete higher', 'Lower secondary', 'Academic degree'],
      dtype=object)
```

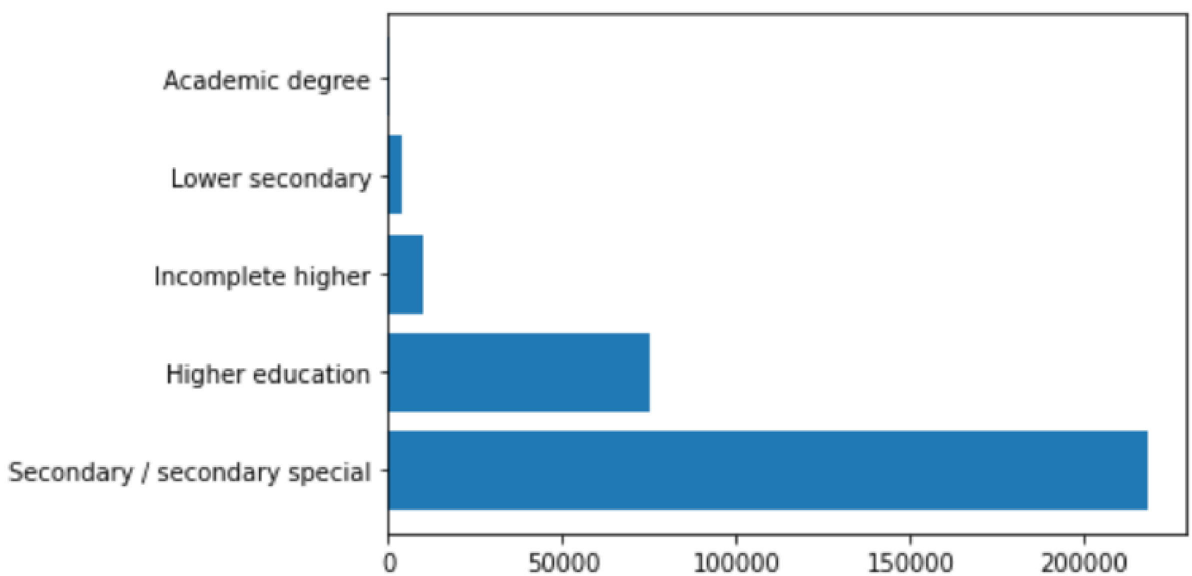
Counting and plotting unique education type

```
b=new_df.EDUCATION.value_counts()
b
```

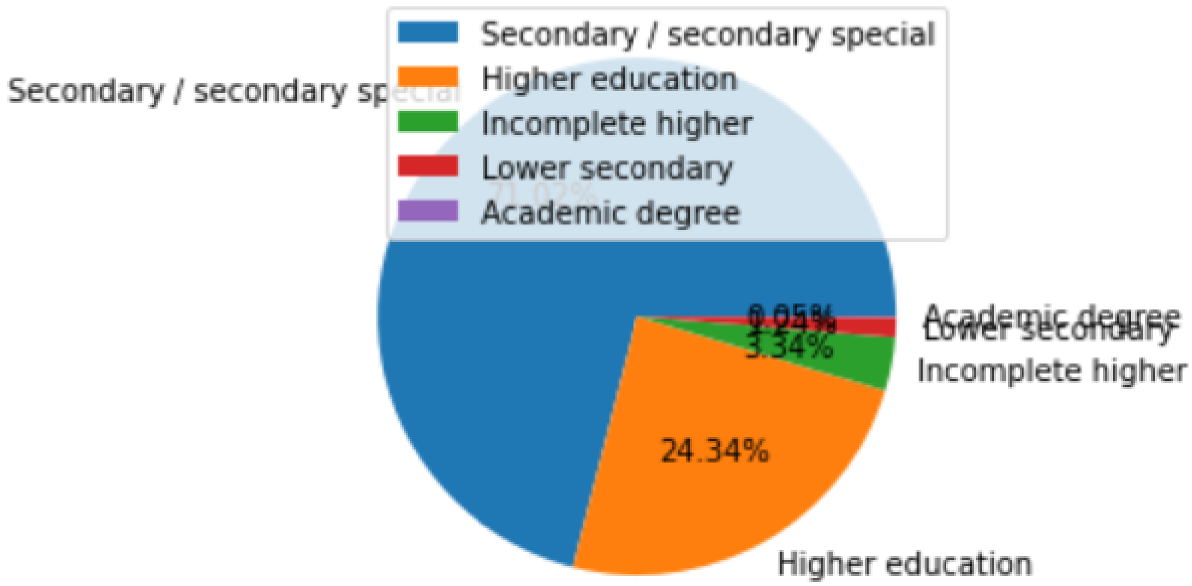
```
Secondary / secondary special    218391
Higher education                  74863
Incomplete higher                 10277
Lower secondary                    3816
Academic degree                     164
Name: EDUCATION, dtype: int64
```

```
plt.barh(b.index, b)
```

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



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



It can be concluded that clients with the education of secondary/secondary special are maximum.

Unique family status

```
new_df.FAMILY_STATUS.unique()
```

```
array(['Single / not married', 'Married', 'Civil marriage', 'Widow',
       'Separated', 'Unknown'], dtype=object)
```

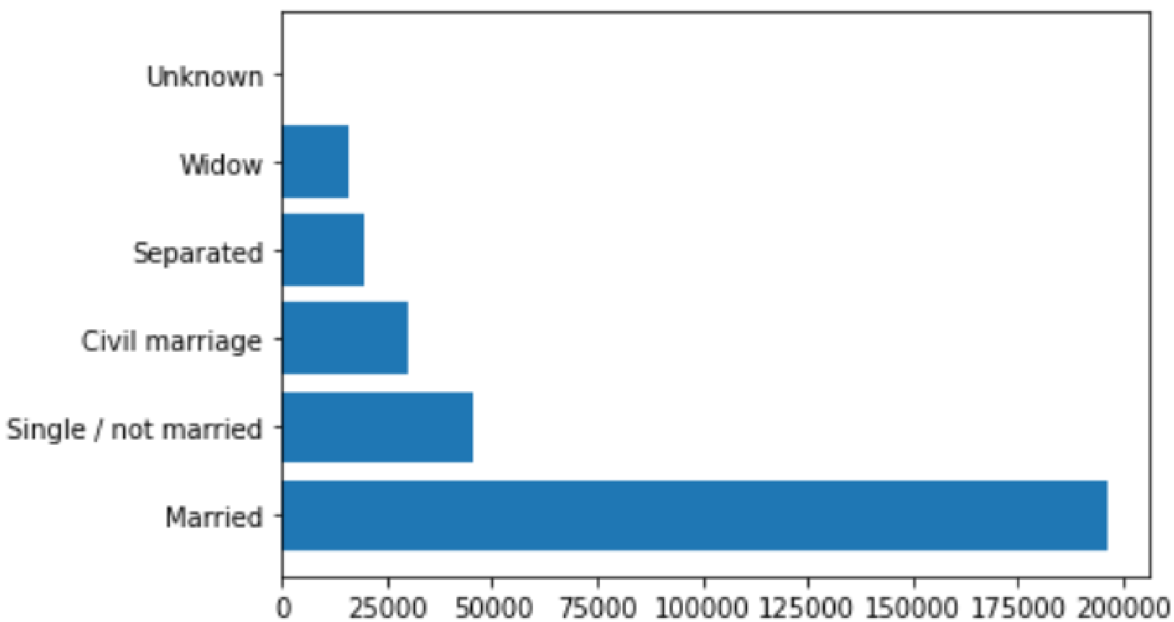
Counting and plotting unique family status

```
c=new_df.FAMILY_STATUS.value_counts()
c
```

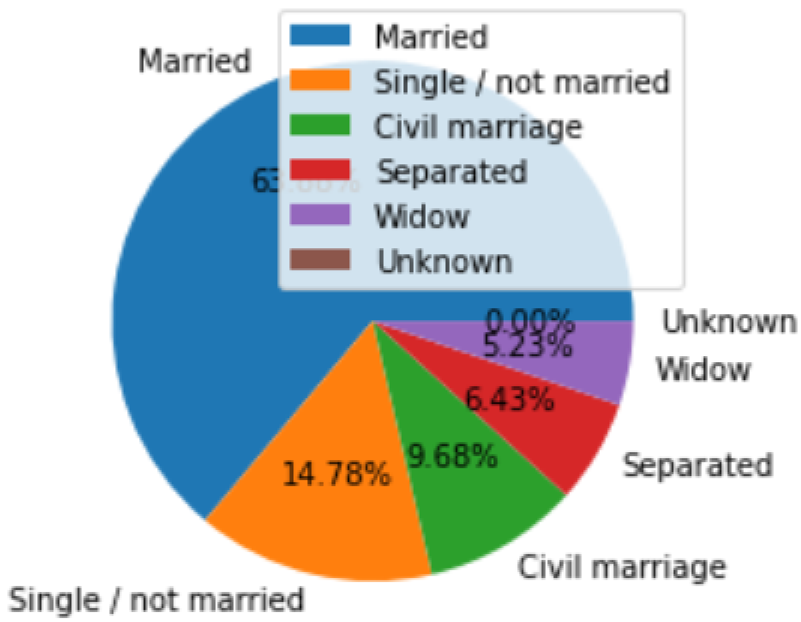
```
Married                    196432
Single / not married        45444
Civil marriage              29775
Separated                   19770
Widow                       16088
Unknown                         2
Name: FAMILY_STATUS, dtype: int64
```

```
plt.barh(c.index, c)
```

```
<BarContainer object of 6 artists>
```



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



It can be concluded that clients with married family status are maximum.

Unique housing type

```
new_df.curr_HOUSING_TYPE.unique()
```

```
array(['House / apartment', 'Rented apartment', 'With parents',
       'Municipal apartment', 'Office apartment', 'Co-op apartment'],
      dtype=object)
```

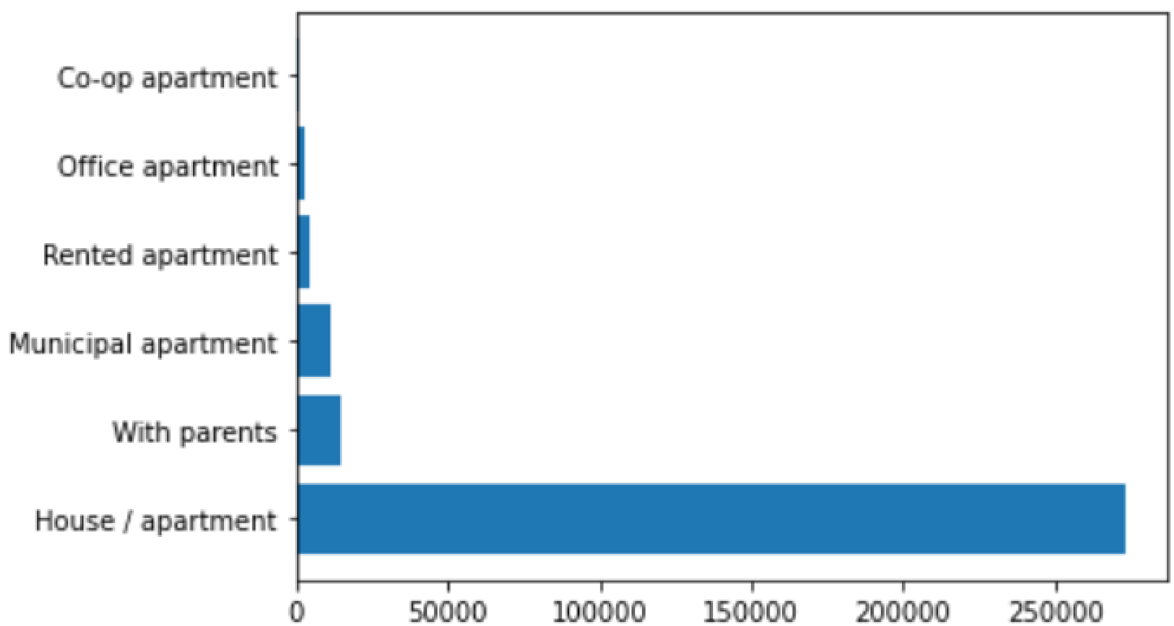
Counting and plotting unique housing type

```
d=new_df.curr_HOUSING_TYPE.value_counts()
d
```

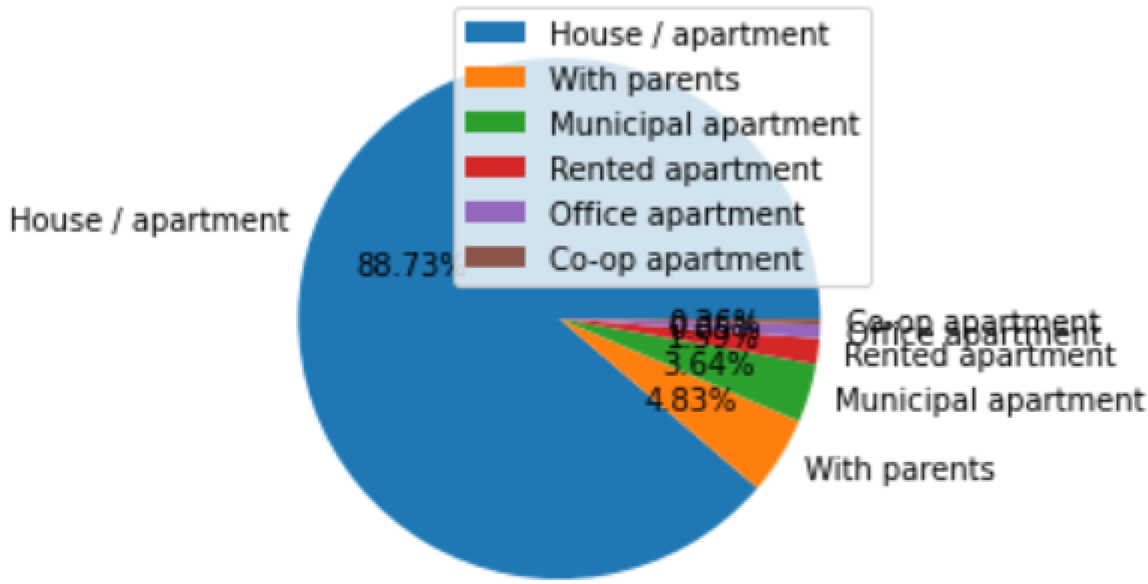
```
House / apartment      272868
With parents            14840
Municipal apartment     11183
Rented apartment         4881
Office apartment         2617
Co-op apartment          1122
Name: curr_HOUSING_TYPE, dtype: int64
```

```
plt.barh(d.index, d)
```

```
<BarContainer object of 6 artists>
```



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



It can be concluded that clients living in housing type of house/apartment are maximum.

Unique organization type

```
new_df.ORGANIZATION_TYPE.unique()
```

```
array(['Business Entity Type 3', 'School', 'Government', 'Religion',
       'Other', 'XNA', 'Electricity', 'Medicine',
       'Business Entity Type 2', 'Self-employed', 'Transport: type 2',
       'Construction', 'Housing', 'Kindergarten', 'Trade: type 7',
       'Industry: type 11', 'Military', 'Services', 'Security Ministries',
       'Transport: type 4', 'Industry: type 1', 'Emergency', 'Security',
       'Trade: type 2', 'University', 'Transport: type 3', 'Police',
       'Business Entity Type 1', 'Postal', 'Industry: type 4',
       'Agriculture', 'Restaurant', 'Culture', 'Hotel',
       'Industry: type 7', 'Trade: type 3', 'Industry: type 3', 'Bank',
       'Industry: type 9', 'Insurance', 'Trade: type 6',
       'Industry: type 2', 'Transport: type 1', 'Industry: type 12',
       'Mobile', 'Trade: type 1', 'Industry: type 5', 'Industry: type 10',
       'Legal Services', 'Advertising', 'Trade: type 5', 'Cleaning',
       'Industry: type 13', 'Trade: type 4', 'Telecom',
       'Industry: type 8', 'Realtor', 'Industry: type 6'], dtype=object)
```

Counting unique organization type

```
new_df.ORGANIZATION_TYPE.value_counts()
```

```
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
Military                   2634
Bank                       2507
Agriculture                2454
Police                     2341
Transport: type 2          2204
Postal                     2157
```

For categorical data(Analyzing and visualizing)

Checking datatypes of columns for finding continuous data type.

```
new_df.dtypes
```

```
Cust_ID                        int64
INCOME                       float64
Loan_AMT                     float64
ANNUITY                       object
GOODS_PRICE                   object
INCOME_TYPE                   object
EDUCATION                     object
FAMILY_STATUS                 object
curr_HOUSING_TYPE             object
age                            int64
Work_Exp                       int64
REGISTRATION_change          float64
DAYS_ID_PUBLISH                int64
MOBIL_given                    int64
EMAIL_given                    int64
OCCUPATION_TYPE               object
Family_MEMBERS_no             object
REGION_CLIENT                  int64
REGION_CLIENT_CITY             int64
Perman_add_NOT_cont_REGION     int64
perman_add_NOT_WORK_add        int64
ORGANIZATION_TYPE             object
EXT_SOURCE_1                  object
EXT_SOURCE_2                  object
EXT_SOURCE_3                  object
```

```
#taking only continues data into account
cat_df=new_df[['Cust_ID','INCOME','Loan_AMT','ANNUITY','GOODS_PRICE']]
```

```
cat_df.head()
```

	Cust_ID	INCOME	Loan_AMT	ANNUITY	GOODS_PRICE
0	100002	202500.0	406597.5	24700.5	351000.0
1	100003	270000.0	1293502.5	35698.5	1129500.0
2	100004	67500.0	135000.0	6750.0	135000.0
3	100006	135000.0	312682.5	29686.5	297000.0
4	100007	121500.0	513000.0	21865.5	513000.0

```
#Checking Corrilation Between columns of numric continues data
cat_df.corr()
```

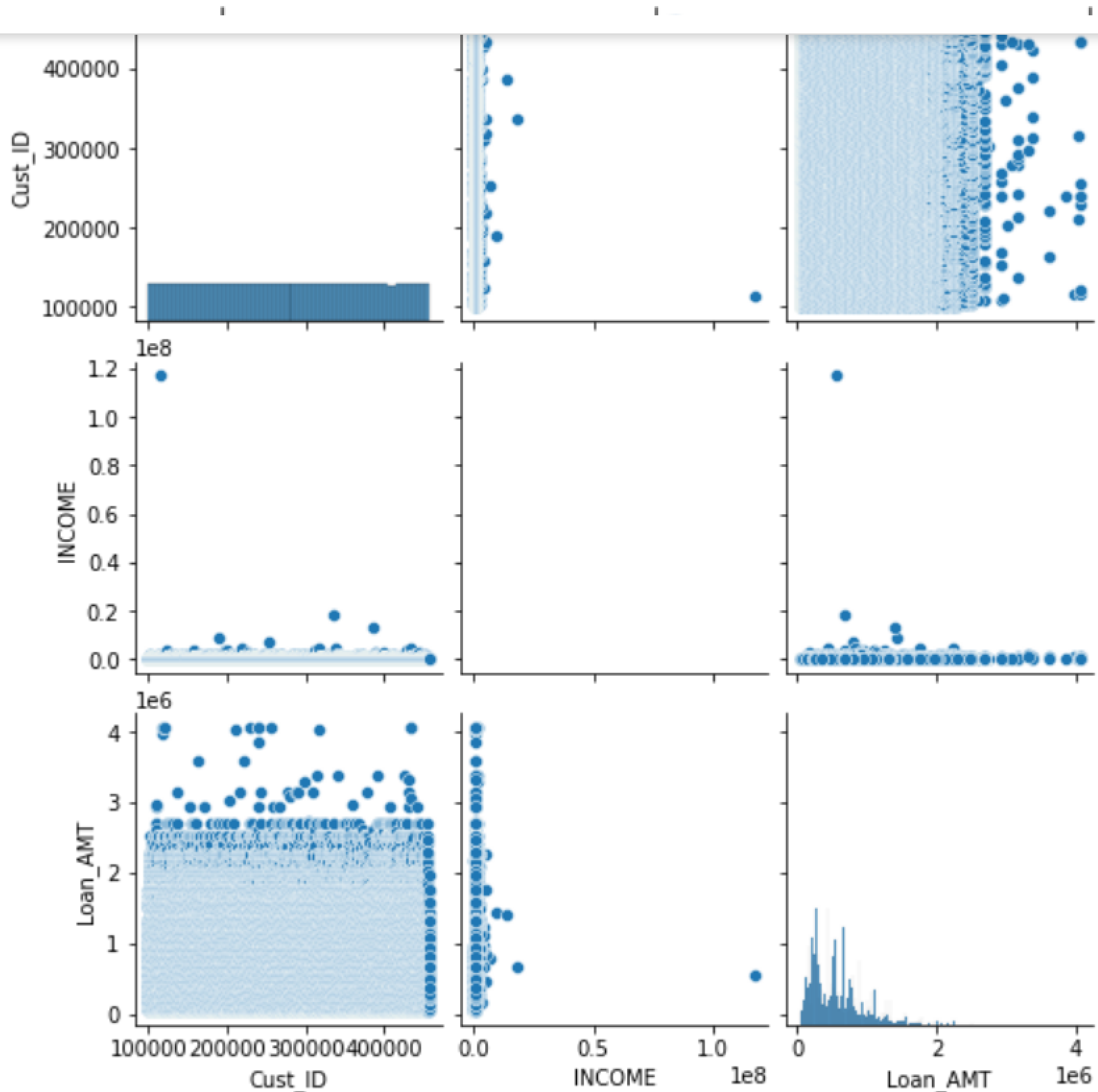
	Cust_ID	INCOME	Loan_AMT
Cust_ID	1.000000	-0.00182	-0.000343
INCOME	-0.001820	1.00000	0.156870
Loan_AMT	-0.000343	0.15687	1.000000

Plotting pairplot for continuous data to check the relation between columns

Showing relationship

```
sns.pairplot(data=cat_df)
```

<seaborn.axisgrid.PairGrid at 0x7fd9ac019520>



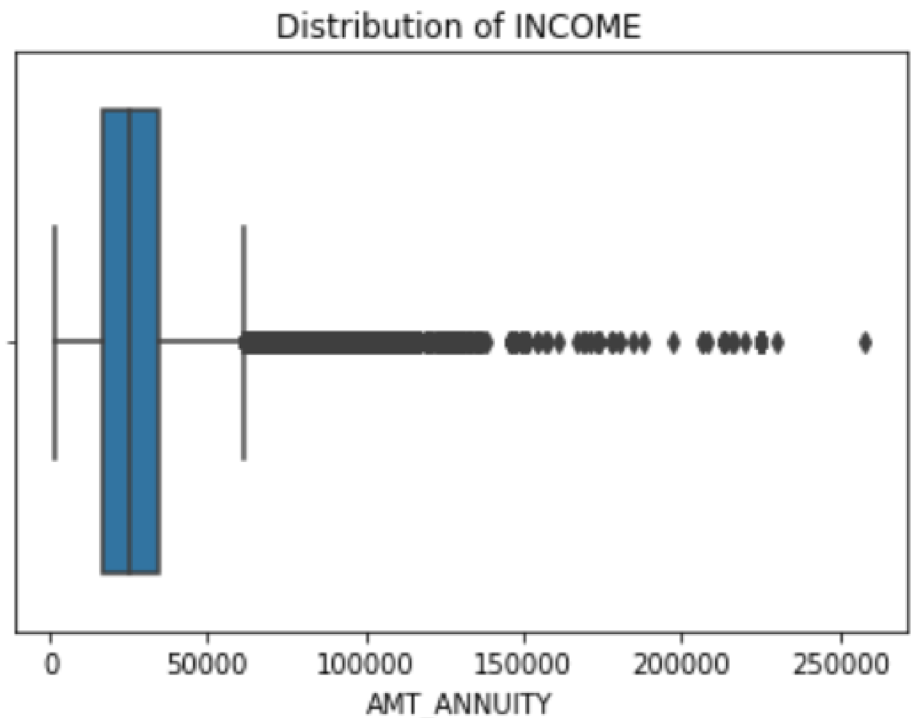
Handling Outliers

```
#Checking outliers for INCOME
cat_df.INCOME.describe()
```

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

```
sns.boxplot(cat_df.INCOME)
plt.title('Distribution of INCOME')
plt.show()
```

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

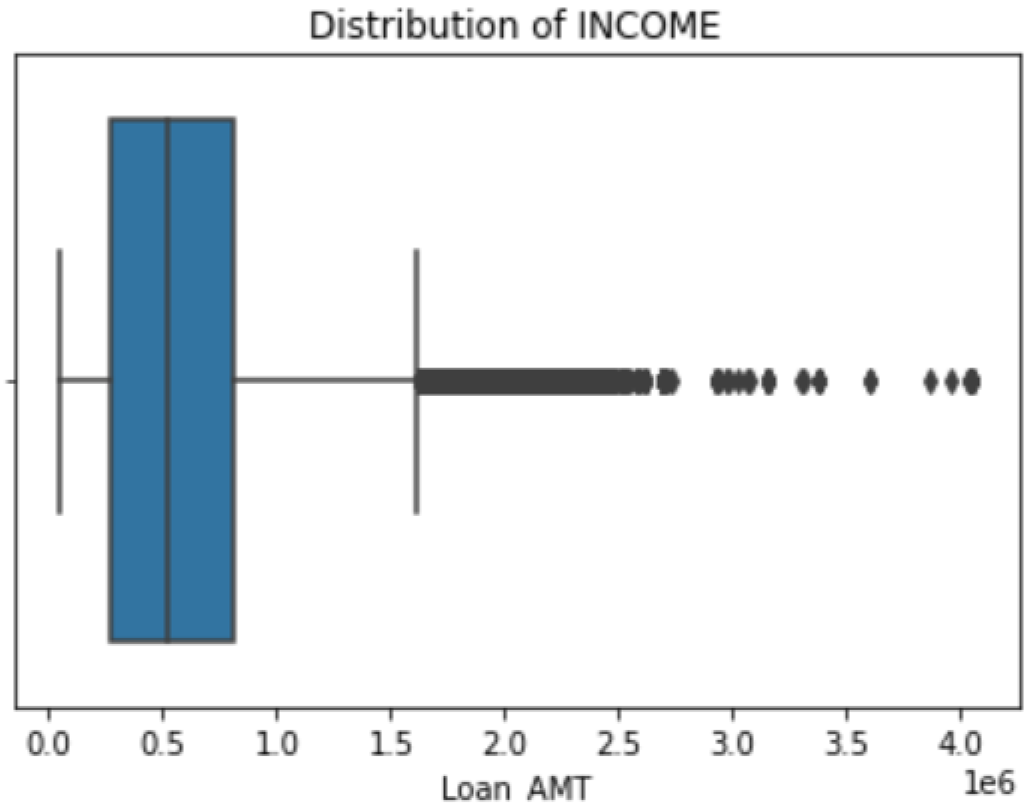


```
#Checking outliers for Loan Amount
cat_df.Loan_AMT.describe()
```

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

```
sns.boxplot(cat_df.Loan_AMT)
plt.title('Distribution of INCOME')
plt.show()
```

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



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

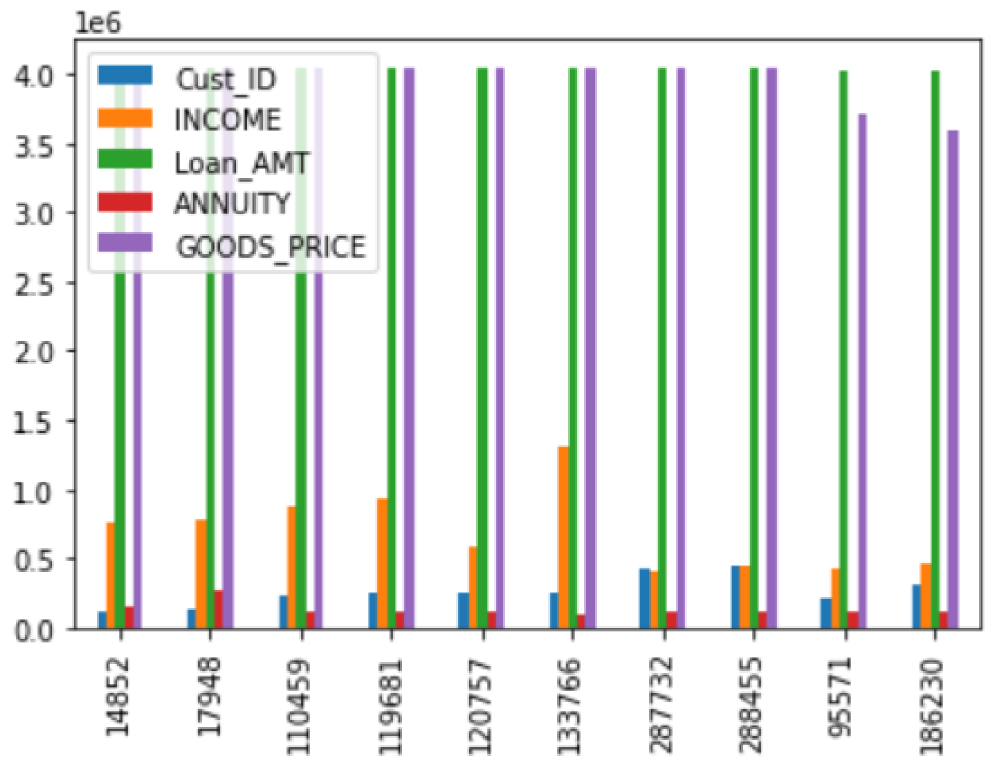
```
#Filterring data on bases of Laon amount for top 10 loan amounts
df_amt=cat_df.nlargest(10,columns='Loan_AMT')
df_amt.head()
```

	Cust_ID	INCOME	Loan_AMT	ANNUITY	GOODS_PRICE
14852	117337	760846.5	4050000.0	146002.5	4050000.0
17948	120926	783000.0	4050000.0	258025.5	4050000.0
110459	228135	864900.0	4050000.0	102384.0	4050000.0
119681	238782	931365.0	4050000.0	102514.5	4050000.0
120757	240007	587250.0	4050000.0	106969.5	4050000.0

Plotting graph for approved loan amount(analyzing and visualizing data)

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9747a82b0>
```



Checking outliers for ANNUITY

```
#Checking outliers for ANNUITY
cat_df.ANNUITY.describe()
```

```
count     307511.0
unique     13673.0
top         9000.0
freq        6385.0
Name: ANNUITY, dtype: float64
```

Checking shape

```
cat_df.shape
```

```
(307511, 5)
```

Dropping annuity

```
cat_df.ANNUITY.dropna()
```

```
print("missing values from Annuty : ",cat_df.ANNUITY.isna().sum())
```

```
missing values from Annuty :  0
```

Checking outliers for GOODS_PRICE

```
#Checking outliers for GOODS_PRICE
cat_df.GOODS_PRICE.describe()
```

```
count     307511.0
unique      1003.0
top       450000.0
freq       26022.0
Name: GOODS_PRICE, dtype: float64
```

Find the top 10 correlation for the Client with payment difficulties and all other cases (Target variable).

```
new_df.TARGET.unique()
```

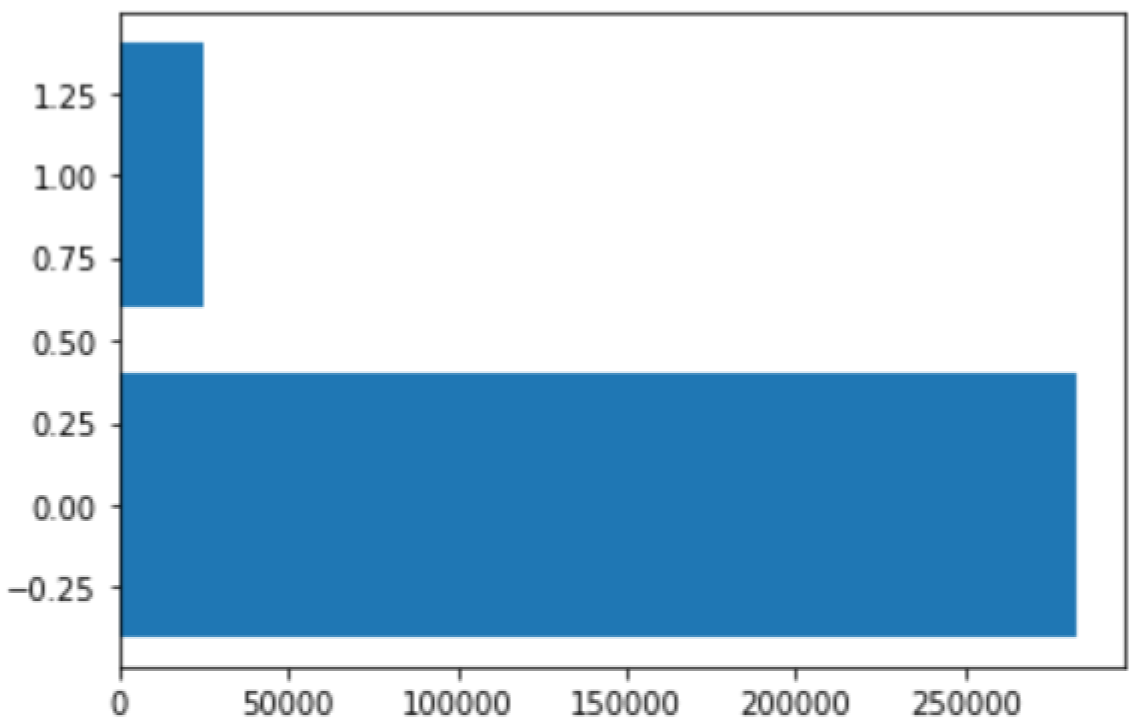
```
array([1, 0])
```

```
r=new_df.TARGET.value_counts()
r
```

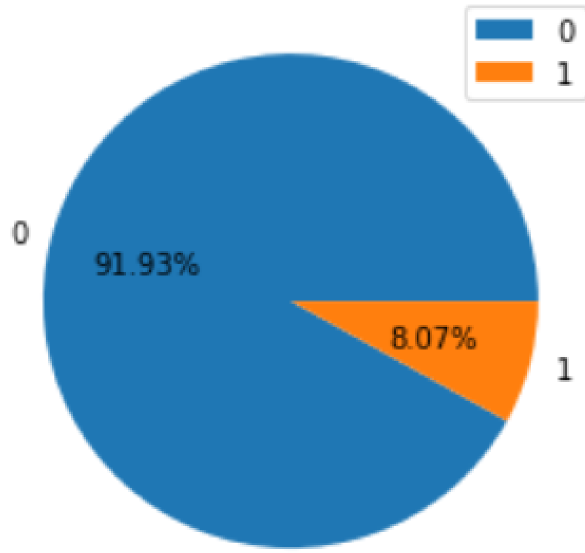
```
0    282686
1     24825
Name: TARGET, dtype: int64
```

```
plt.barh(r.index, r)
```

```
<BarContainer object of 2 artists>
```



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



Conclusion:: 8.07% of people have payment difficulty

Binary Logistic Regression(taking into account Target as response variable)

```python
dff=new_df[['Cust_ID','TARGET','INCOME','Loan_AMT','ANNUITY','GOODS_PRICE']]
```

General Regression:: at least 1 catagorical predictor(we create dummy var) and respoce is continues(we have target variable as catogorical data)

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import sklearn
from sklearn.model_selection import train_test_split
```

```
[ ] mydf=dff[['TARGET','INCOME','Loan_AMT','ANNUITY','GOODS_PRICE']]
```

+ Code + Text

```
[ ] #x_train, x_test, y_train, y_test = train_test_split(mydf.drop('TARGET', axis=1), mydf.TARGET, test_size=0.2)
```

Checking for Target var vs Income

```
x = mydf[['INCOME']]
y=mydf[['TARGET']]
```

```
x=sm.add_constant(x)
```

```
/usr/local/lib/python3.8/dist-p
  x = pd.concat(x[::order], 1)
```

```
x
```

	const	INCOME
0	1.0	202500.0
1	1.0	270000.0
2	1.0	67500.0
3	1.0	135000.0
4	1.0	121500.0
...	...	...
307506	1.0	157500.0
307507	1.0	72000.0
307508	1.0	153000.0
307509	1.0	171000.0
307510	1.0	157500.0

307511 rows × 2 columns

```
mod = sm.Logit(y,x).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.280512
         Iterations 7
```

```
print(mod.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                 TARGET   No. Observations:               307511
Model:                          Logit   Df Residuals:                   307509
Method:                           MLE   Df Model:                            1
Date:                Sat, 31 Dec 2022   Pseudo R-squ.:               0.0001219
Time:                        11:29:51   Log-Likelihood:                -86260.
converged:                       True   LL-Null:                       -86271.
Covariance Type:            nonrobust   LLR p-value:                 4.514e-06
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -2.3800      0.013   -176.495      0.000      -2.406      -2.354
INCOME     -3.142e-07   7.08e-08     -4.435      0.000   -4.53e-07   -1.75e-07
==============================================================================
```

p=value is <0.05 so Ho is rejected and it means that Income affect the person will default or not on loan repayment

Checking for Target Var vs Loan_AMT

```
x = mydf[['Loan_AMT']]
y=mydf[['TARGET']]
```

```
[ ]  x=sm.add_constant(x)
```

```
/usr/local/lib/python3.8/dist
  x = pd.concat(x[::order], 1
```

```
x
```

	const	Loan_AMT
0	1.0	406597.5
1	1.0	1293502.5
2	1.0	135000.0
3	1.0	312682.5
4	1.0	513000.0
...	...	...
307506	1.0	254700.0
307507	1.0	269550.0
307508	1.0	677664.0
307509	1.0	370107.0
307510	1.0	675000.0

```
mod1 = sm.Logit(y,x).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.280066
         Iterations 7
```

```
print(mod1.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                 TARGET   No. Observations:               307511
Model:                          Logit   Df Residuals:                   307509
Method:                           MLE   Df Model:                            1
Date:                Sat, 31 Dec 2022   Pseudo R-squ.:                0.001712
Time:                        12:23:30   Log-Likelihood:                -86123.
converged:                       True   LL-Null:                       -86271.
Covariance Type:            nonrobust   LLR p-value:                 3.428e-66
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -2.2617      0.012   -190.937      0.000      -2.285      -2.239
Loan_AMT   -2.946e-07   1.75e-08    -16.825      0.000   -3.29e-07    -2.6e-07
==============================================================================
```

p=value is <0.05 so Ho is rejected and it means that Income affect the person will default or not on loan repayment