Importing Libraries and Dataset

In [3]:
```
# importing the required libraries/Modules

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

In [4]:
```
application_data = pd.read_csv("application_data.csv")
previous_application = pd.read_csv("previous_application.csv")
```

```
# Dataset Explore
```

In [5]:
```
application_data.head()
```

Out[5]:

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

5 rows × 122 columns

In [6]:
```
application_data.tail()
```

Out[6]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLA
49994	157871	0	Cash loans	F	N	
49995	157872	0	Cash loans	M	N	
49996	157873	0	Cash loans	M	N	
49997	157874	0	Cash loans	F	N	
49998	157875	0	Cash loans	F	N	

5 rows × 122 columns

In [7]:
```
1 application_data.columns
```

Out[7]:
```
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)
```

In [8]:
```
1 application_data.shape
```

Out[8]:
```
(49999, 122)
```

In [9]:
```
1 application_data.describe()
```

Out[9]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_
count	49999.000000	49999.000000	49999.000000	4.999900e+04	4.999900e+04	499
mean	129013.210584	0.080522	0.419848	1.707676e+05	5.997006e+05	271
std	16690.512048	0.272102	0.724039	5.318191e+05	4.024154e+05	145
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	20
25%	114570.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	164
50%	129076.000000	0.000000	0.000000	1.458000e+05	5.147775e+05	249
75%	143438.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	345
max	157875.000000	1.000000	11.000000	1.170000e+08	4.050000e+06	2580

8 rows × 106 columns

In [10]:
```
1 application_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(64), int64(42), object(16)
memory usage: 46.5+ MB
```

In [11]:
```
1 previous_application.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Data columns (total 37 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   SK_ID_PREV                   49999 non-null  int64  
 1   SK_ID_CURR                   49999 non-null  int64  
 2   NAME_CONTRACT_TYPE           49999 non-null  object 
 3   AMT_ANNUITY                  39407 non-null  float64
 4   AMT_APPLICATION              49999 non-null  float64
 5   AMT_CREDIT                   49999 non-null  float64
 6   AMT_DOWN_PAYMENT             24801 non-null  float64
 7   AMT_GOODS_PRICE              39255 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   49999 non-null  object 
 9   HOUR_APPR_PROCESS_START      49999 non-null  int64  
 10  FLAG_LAST_APPL_PER_CONTRACT  49999 non-null  object 
 11  NFLAG_LAST_APPL_IN_DAY       49999 non-null  int64  
 12  RATE_DOWN_PAYMENT            24801 non-null  float64
 13  RATE_INTEREST_PRIMARY        165 non-null    float64
 14  RATE_INTEREST_PRIVILEGED     165 non-null    float64
 15  NAME_CASH_LOAN_PURPOSE       49999 non-null  object 
 16  NAME_CONTRACT_STATUS         49999 non-null  object 
 17  DAYS_DECISION                49999 non-null  int64  
 18  NAME_PAYMENT_TYPE            49999 non-null  object 
 19  CODE_REJECT_REASON           49999 non-null  object 
 20  NAME_TYPE_SUITE              25756 non-null  object 
 21  NAME_CLIENT_TYPE             49999 non-null  object 
 22  NAME_GOODS_CATEGORY          49999 non-null  object 
 23  NAME_PORTFOLIO               49999 non-null  object 
 24  NAME_PRODUCT_TYPE            49999 non-null  object 
 25  CHANNEL_TYPE                 49999 non-null  object 
 26  SELLERPLACE_AREA             49999 non-null  int64  
 27  NAME_SELLER_INDUSTRY         49999 non-null  object 
 28  CNT_PAYMENT                  39407 non-null  float64
 29  NAME_YIELD_GROUP             49999 non-null  object 
 30  PRODUCT_COMBINATION          49991 non-null  object 
 31  DAYS_FIRST_DRAWING           30839 non-null  float64
 32  DAYS_FIRST_DUE               30839 non-null  float64
 33  DAYS_LAST_DUE_1ST_VERSION    30839 non-null  float64
 34  DAYS_LAST_DUE                30839 non-null  float64
 35  DAYS_TERMINATION             30839 non-null  float64
 36  NFLAG_INSURED_ON_APPROVAL    30839 non-null  float64
dtypes: float64(15), int64(6), object(16)
memory usage: 14.1+ MB
```

In [12]:
```
previous_application.describe()
```

Out[12]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DO
count	4.999900e+04	49999.000000	39407.000000	4.999900e+04	4.999900e+04	
mean	1.922254e+06	278983.187604	15482.596847	1.688925e+05	1.885429e+05	
std	5.351980e+05	102780.124434	14530.971854	2.822035e+05	3.084736e+05	
min	1.000001e+06	100007.000000	0.000000	0.000000e+00	0.000000e+00	
25%	1.457920e+06	189919.500000	6122.835000	2.204550e+04	2.605500e+04	
50%	1.920889e+06	279264.000000	10879.920000	7.155000e+04	7.890750e+04	
75%	2.388632e+06	368527.500000	19669.140000	1.800000e+05	1.981058e+05	
max	2.845367e+06	456254.000000	234478.395000	3.826372e+06	4.104351e+06	

8 rows × 21 columns

In [13]:
```
previous_application.head()
```

Out[13]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AM
0	2030495	271877	Consumer loans	1730.430	17145.0	
1	2802425	108129	Cash loans	25188.615	607500.0	
2	2523466	122040	Cash loans	15060.735	112500.0	
3	2819243	176158	Cash loans	47041.335	450000.0	
4	1784265	202054	Cash loans	31924.395	337500.0	

5 rows × 37 columns

In [14]:
```
previous_application.tail()
```

Out[14]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION
49994	1171956	339569	Cash loans	NaN	0.0
49995	1904808	363980	Cash loans	NaN	0.0
49996	2331005	231295	Cash loans	22176.405	180000.0
49997	1960897	346691	Cash loans	NaN	0.0
49998	1979352	363244	Cash loans	24909.390	360000.0

5 rows × 37 columns

In [15]:
```
previous_application.shape
```

Out[15]:
```
(49999, 37)
```

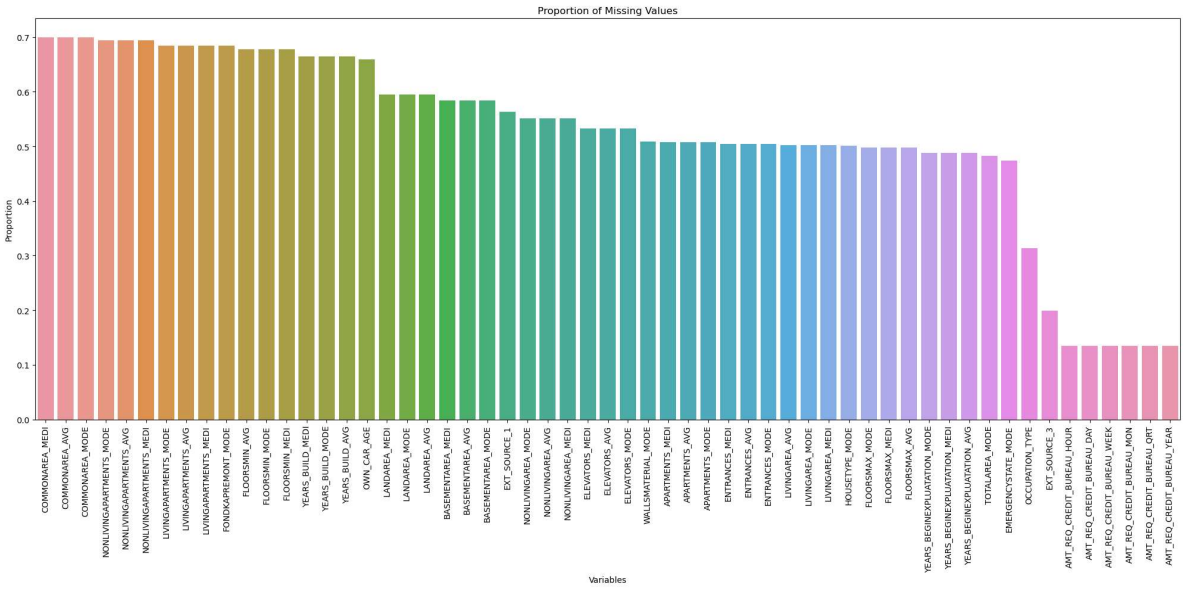
A. Identify Missing Data and Deal with it Appropriately

In [16]:
```
a = application_data.isnull().sum()
a
```

Out[16]:
```
SK_ID_CURR                      0
TARGET                          0
NAME_CONTRACT_TYPE              0
CODE_GENDER                     0
FLAG_OWN_CAR                    0
                             ...
AMT_REQ_CREDIT_BUREAU_DAY    6734
AMT_REQ_CREDIT_BUREAU_WEEK   6734
AMT_REQ_CREDIT_BUREAU_MON    6734
AMT_REQ_CREDIT_BUREAU_QRT    6734
AMT_REQ_CREDIT_BUREAU_YEAR   6734
Length: 122, dtype: int64
```

Creating bar chart to visualize the proportion of missing values for each variable.

In [17]:
```
# Calculate the proportion of missing values for columns having missing va
mean_val = application_data.isnull().mean()

# Sorting the mean_val for better visualization
mean_val_sorted = mean_val.sort_values(ascending=False)

missing_var_mean = mean_val_sorted[mean_val_sorted>0.1]

# Create a bar chart using seaborn
plt.figure(figsize=(20, 10))
sns.barplot(x=missing_var_mean.index, y=missing_var_mean.values)
plt.title('Proportion of Missing Values')
plt.xlabel('Variables')
plt.ylabel('Proportion')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

```



In [18]:
```
a = a[a>(len(application_data)*0.5)]
print(a)
len(a)
```

```
OWN_CAR_AGE                 32950
EXT_SOURCE_1                28172
APARTMENTS_AVG              25385
BASEMENTAREA_AVG            29199
YEARS_BUILD_AVG             33239
COMMONAREA_AVG              34960
ELEVATORS_AVG               26651
ENTRANCES_AVG               25195
FLOORSMIN_AVG               33894
LANDAREA_AVG                29721
LIVINGAPARTMENTS_AVG        34226
LIVINGAREA_AVG              25137
NONLIVINGAPARTMENTS_AVG     34714
NONLIVINGAREA_AVG           27572
APARTMENTS_MODE             25385
BASEMENTAREA_MODE           29199
YEARS_BUILD_MODE            33239
COMMONAREA_MODE             34960
ELEVATORS_MODE              26651
ENTRANCES_MODE              25195
FLOORSMIN_MODE              33894
LANDAREA_MODE               29721
LIVINGAPARTMENTS_MODE       34226
LIVINGAREA_MODE             25137
NONLIVINGAPARTMENTS_MODE    34714
NONLIVINGAREA_MODE          27572
APARTMENTS_MEDI             25385
BASEMENTAREA_MEDI           29199
YEARS_BUILD_MEDI            33239
COMMONAREA_MEDI             34960
ELEVATORS_MEDI              26651
ENTRANCES_MEDI              25195
FLOORSMIN_MEDI              33894
LANDAREA_MEDI               29721
LIVINGAPARTMENTS_MEDI       34226
LIVINGAREA_MEDI             25137
NONLIVINGAPARTMENTS_MEDI    34714
NONLIVINGAREA_MEDI          27572
FONDKAPREMONT_MODE          34191
HOUSETYPE_MODE              25075
WALLSMATERIAL_MODE          25459
dtype: int64
```

Out[18]: 41

In [19]:
```
# Here we need OWN_CAR_AGE and EXT_SOURCE_1 columns thus removing that ent

a.pop("OWN_CAR_AGE")
a.pop("EXT_SOURCE_1")
```

Out[19]: 28172

This columns have more than 50% missing values

Discription of this columns :- Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor

We don't need this information for our analysis thus removing those columns

In [20]:
```
# drop those columns from the application_data

application_data = application_data.drop(columns=a.index)
```

In [21]:
```
# Shape of the dataset after dropping those columns

application_data.shape
```

Out[21]: (49999, 83)

In [22]:
```
application_data.head()
```

Out[22]:

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

5 rows × 83 columns

In [23]:
```
# Let's see now which columns have null values so that we can work on ther

a = application_data.isnull().sum()
a = a[a>0]
print(a)
len(a)
```

```
AMT_ANNUITY                      1
AMT_GOODS_PRICE                 38
NAME_TYPE_SUITE                192
OWN_CAR_AGE                  32950
OCCUPATION_TYPE              15654
CNT_FAM_MEMBERS                  1
EXT_SOURCE_1                 28172
EXT_SOURCE_2                   126
EXT_SOURCE_3                  9944
YEARS_BEGINEXPLUATATION_AVG  24394
FLOORSMAX_AVG                24875
YEARS_BEGINEXPLUATATION_MODE 24394
FLOORSMAX_MODE               24875
YEARS_BEGINEXPLUATATION_MEDI 24394
FLOORSMAX_MEDI               24875
TOTALAREA_MODE               24148
EMERGENCYSTATE_MODE          23698
OBS_30_CNT_SOCIAL_CIRCLE       168
DEF_30_CNT_SOCIAL_CIRCLE       168
OBS_60_CNT_SOCIAL_CIRCLE       168
DEF_60_CNT_SOCIAL_CIRCLE       168
DAYS_LAST_PHONE_CHANGE           1
AMT_REQ_CREDIT_BUREAU_HOUR    6734
AMT_REQ_CREDIT_BUREAU_DAY     6734
AMT_REQ_CREDIT_BUREAU_WEEK    6734
AMT_REQ_CREDIT_BUREAU_MON     6734
AMT_REQ_CREDIT_BUREAU_QRT     6734
AMT_REQ_CREDIT_BUREAU_YEAR    6734
dtype: int64
```

Out[23]: 28

Observation

1) AMT_REQ_CREDIT_BUREAU_HOUR/DAY/WEEK/MON/QRT/YEAR has the Number of enquiries to Credit Bureau about the client before application thus null values in this columns mean that there are 0 enquires thus filling null values in this columns with 0.

2) NAME_TYPE_SUITE has null values where it is not provided by applicant but if it is not porvided then it should be "Unaccompanied".

3) OCCUPATION_TYPE has null values where it is not provied by applicant let's replace it with "Unknown".

4) CNT_FAM_MEMBERS has 1 null value also its family_details are unknown so it is safe to replce its value to 0

In [24]:
```
# Filling those null values with the obersved value

keys = ['AMT_REQ_CREDIT_BUREAU_HOUR', "AMT_REQ_CREDIT_BUREAU_DAY", "AMT_RE
values = [0,0,0,0,0,0,"Unaccompanied", "Unknown", 0]

filldict = dict(zip(keys, values))

application_data.fillna(value=filldict, inplace=True)
```

In [25]:
```
# Removing the Normalized information about building where the client live

application_data = application_data.drop(columns=["YEARS_BEGINEXPLUATATION
```

Handling Duplicate values

In [26]:
```
# Finding duplicated values

application_data.duplicated().sum()
```

Out[26]: 0

Changing the columns having neagtive values.

In [27]:
```
# This values might be entered negative accendentally .
# We can easily see that age can't have negative value.

application_data[['DAYS_BIRTH' ,'DAYS_EMPLOYED' ,'DAYS_REGISTRATION' ,'DA
```

Out[27]:

	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION	DAYS_ID_PUBLISH
0	-9461	-637	-3648	-2120
1	-16765	-1188	-1186	-291
2	-19046	-225	-4260	-2531
3	-19005	-3039	-9833	-2437
4	-19932	-3038	-4311	-3458
...	...	...	...	...
49994	-10667	-285	-2521	-3333
49995	-20211	-4651	-11281	-3722
49996	-10280	-1158	-8620	-2604
49997	-23485	-2181	-2662	-4200
49998	-19251	365243	-12934	-2783

49999 rows × 4 columns

In [28]:
```
# Using absolute fucntion to convert negative values to positive

application_data[['DAYS_BIRTH' ,'DAYS_EMPLOYED' ,'DAYS_REGISTRATION' ,'DA

application_data[['DAYS_BIRTH' ,'DAYS_EMPLOYED' ,'DAYS_REGISTRATION' ,'DA
```

Out[28]:

	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION	DAYS_ID_PUBLISH
0	9461	637	3648	2120
1	16765	1188	1186	291
2	19046	225	4260	2531
3	19005	3039	9833	2437
4	19932	3038	4311	3458
...	...	...	...	...
49994	10667	285	2521	3333
49995	20211	4651	11281	3722
49996	10280	1158	8620	2604
49997	23485	2181	2662	4200
49998	19251	365243	12934	2783

49999 rows × 4 columns

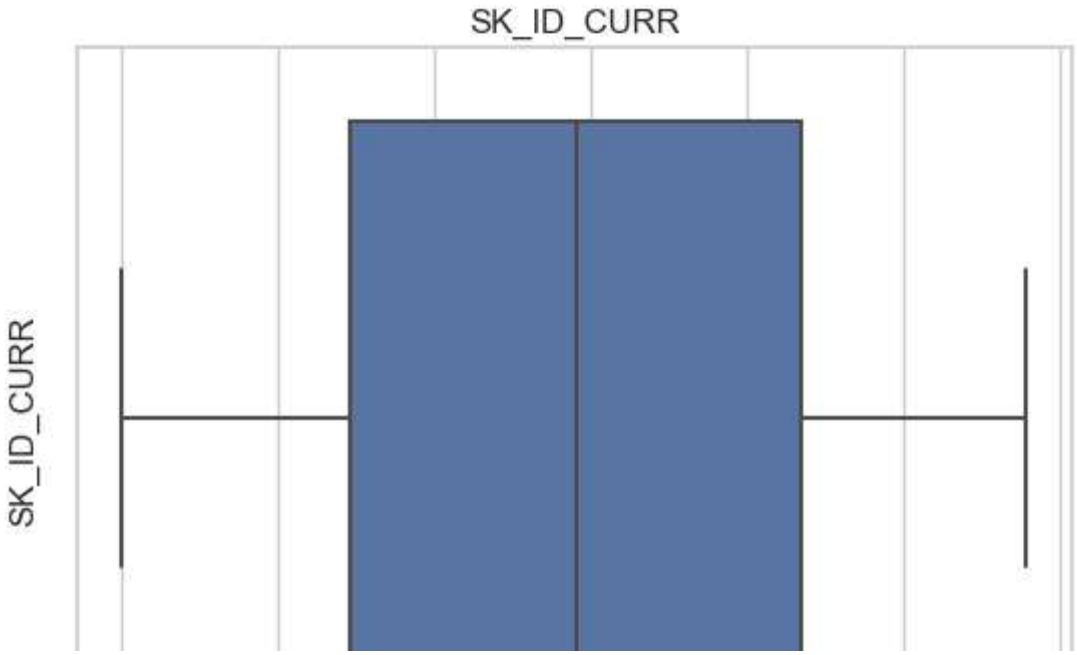
So we Identifed Missing values and Handled them.

B. Identify Outliers in the Datase

In [29]:
```
# Select numerical columns for analysis of Outliers
numerical_columns = application_data.select_dtypes(include=[np.number]).c
```

In [30]:

```
sns.set_theme(style="whitegrid")

# Created and empty dictuonary to store outliers as values and variable na
dictoutlier = {}


for i in numerical_columns:

    # Created an empty list to store outliers
    outliers = []

    # to Calculate IQR of data i
    q1 = np.percentile(application_data[i], 25)
    q3 = np.percentile(application_data[i], 75)
    iqr = q3 - q1

    # Define lower and upper bounds for outliers
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr


    # find and store outliers in the variable i
    outliers = [x for x in application_data[i] if x < lower_bound or x > u
    dictoutlier[i] = outliers

    # Plot the box plot with variable name as title and outliers marked ma
    sns.boxplot(x = application_data[i], flierprops=dict(marker='o', marke
    plt.title(i)
    plt.ylabel(i)
    plt.show()
```



In [31]:
```
# let's see how many outliers each variable have

for key, value in dictoutlier.items():
    if len(value) > 0:
        print(f"Length of {key}: {len(value)}")
```

```
Length of TARGET: 4026
Length of CNT_CHILDREN: 723
Length of AMT_INCOME_TOTAL: 2295
Length of AMT_CREDIT: 1063
Length of REGION_POPULATION_RELATIVE: 1329
Length of DAYS_EMPLOYED: 9082
Length of DAYS_REGISTRATION: 96
Length of FLAG_MOBIL: 1
Length of FLAG_EMP_PHONE: 8926
Length of FLAG_WORK_PHONE: 9963
Length of FLAG_CONT_MOBILE: 101
Length of FLAG_EMAIL: 2783
Length of CNT_FAM_MEMBERS: 684
Length of REGION_RATING_CLIENT: 13035
Length of REGION_RATING_CLIENT_W_CITY: 12658
Length of HOUR_APPR_PROCESS_START: 353
Length of REG_REGION_NOT_LIVE_REGION: 750
Length of REG_REGION_NOT_WORK_REGION: 2496
Length of LIVE_REGION_NOT_WORK_REGION: 1982
Length of REG_CITY_NOT_LIVE_CITY: 3998
Length of REG_CITY_NOT_WORK_CITY: 11608
Length of LIVE_CITY_NOT_WORK_CITY: 8985
Length of FLAG_DOCUMENT_2: 2
Length of FLAG_DOCUMENT_4: 9
Length of FLAG_DOCUMENT_5: 785
Length of FLAG_DOCUMENT_6: 4335
Length of FLAG_DOCUMENT_7: 11
Length of FLAG_DOCUMENT_8: 4038
Length of FLAG_DOCUMENT_9: 184
Length of FLAG_DOCUMENT_10: 1
Length of FLAG_DOCUMENT_11: 213
Length of FLAG_DOCUMENT_13: 161
Length of FLAG_DOCUMENT_14: 158
Length of FLAG_DOCUMENT_15: 41
Length of FLAG_DOCUMENT_16: 501
Length of FLAG_DOCUMENT_17: 15
Length of FLAG_DOCUMENT_18: 425
Length of FLAG_DOCUMENT_19: 35
Length of FLAG_DOCUMENT_20: 26
Length of FLAG_DOCUMENT_21: 19
Length of AMT_REQ_CREDIT_BUREAU_HOUR: 295
Length of AMT_REQ_CREDIT_BUREAU_DAY: 272
Length of AMT_REQ_CREDIT_BUREAU_WEEK: 1314
Length of AMT_REQ_CREDIT_BUREAU_MON: 7140
Length of AMT_REQ_CREDIT_BUREAU_QRT: 8134
Length of AMT_REQ_CREDIT_BUREAU_YEAR: 552
```

In [32]:
```
# let's see how many unique outliers each variable have

for key, value in dictoutlier.items():
    if len(value) > 0:
        print(f"Length of {key}: {len(set(value))}")
```

```
Length of TARGET: 1
Length of CNT_CHILDREN: 8
Length of AMT_INCOME_TOTAL: 120
Length of AMT_CREDIT: 262
Length of REGION_POPULATION_RELATIVE: 1
Length of DAYS_EMPLOYED: 151
Length of DAYS_REGISTRATION: 96
Length of FLAG_MOBIL: 1
Length of FLAG_EMP_PHONE: 1
Length of FLAG_WORK_PHONE: 1
Length of FLAG_CONT_MOBILE: 1
Length of FLAG_EMAIL: 1
Length of CNT_FAM_MEMBERS: 8
Length of REGION_RATING_CLIENT: 2
Length of REGION_RATING_CLIENT_W_CITY: 2
Length of HOUR_APPR_PROCESS_START: 7
Length of REG_REGION_NOT_LIVE_REGION: 1
Length of REG_REGION_NOT_WORK_REGION: 1
Length of LIVE_REGION_NOT_WORK_REGION: 1
Length of REG_CITY_NOT_LIVE_CITY: 1
Length of REG_CITY_NOT_WORK_CITY: 1
Length of LIVE_CITY_NOT_WORK_CITY: 1
Length of FLAG_DOCUMENT_2: 1
Length of FLAG_DOCUMENT_4: 1
Length of FLAG_DOCUMENT_5: 1
Length of FLAG_DOCUMENT_6: 1
Length of FLAG_DOCUMENT_7: 1
Length of FLAG_DOCUMENT_8: 1
Length of FLAG_DOCUMENT_9: 1
Length of FLAG_DOCUMENT_10: 1
Length of FLAG_DOCUMENT_11: 1
Length of FLAG_DOCUMENT_13: 1
Length of FLAG_DOCUMENT_14: 1
Length of FLAG_DOCUMENT_15: 1
Length of FLAG_DOCUMENT_16: 1
Length of FLAG_DOCUMENT_17: 1
Length of FLAG_DOCUMENT_18: 1
Length of FLAG_DOCUMENT_19: 1
Length of FLAG_DOCUMENT_20: 1
Length of FLAG_DOCUMENT_21: 1
Length of AMT_REQ_CREDIT_BUREAU_HOUR: 3
Length of AMT_REQ_CREDIT_BUREAU_DAY: 6
Length of AMT_REQ_CREDIT_BUREAU_WEEK: 6
Length of AMT_REQ_CREDIT_BUREAU_MON: 19
Length of AMT_REQ_CREDIT_BUREAU_QRT: 8
Length of AMT_REQ_CREDIT_BUREAU_YEAR: 11
```

So here, outliers for each variable is found and stored in a dictionary named dictoutliers. We can use this result in our futhur study.

In [ ]:
```
1
```

In [ ]:
```
1
```

C. Analyze Data Imbalance:

In [33]:
```
1 class_frequencies = application_data["NAME_CONTRACT_TYPE"].value_counts()
```

In [34]:
```
1 print(class_frequencies)
2 type(class_frequencies)
3 class_frequencies
```

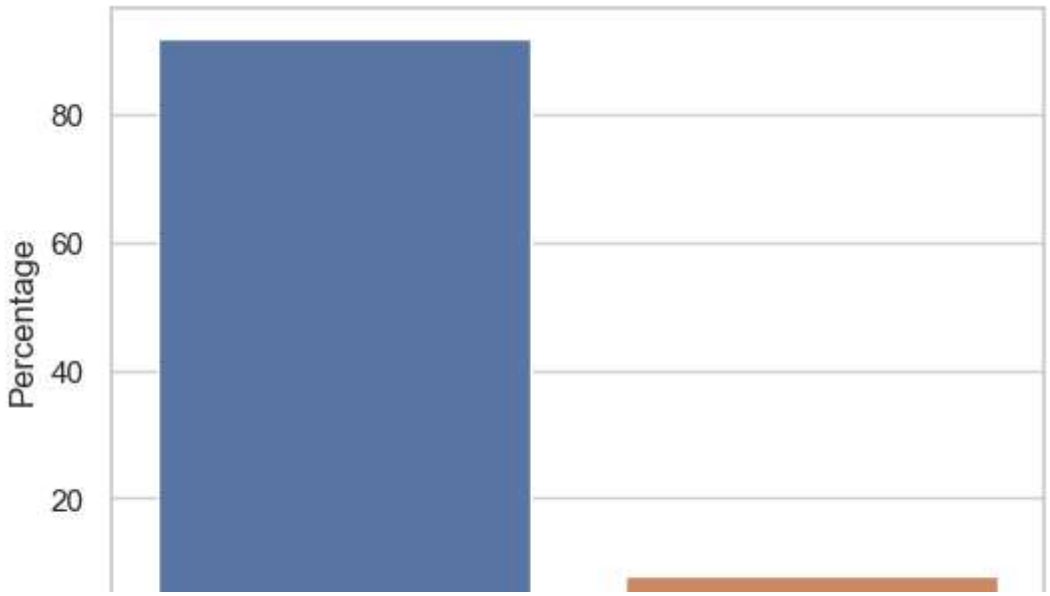
```
Cash loans         45276
Revolving loans     4723
Name: NAME_CONTRACT_TYPE, dtype: int64
```

Out[34]:
```
Cash loans         45276
Revolving loans     4723
Name: NAME_CONTRACT_TYPE, dtype: int64
```

In [35]:

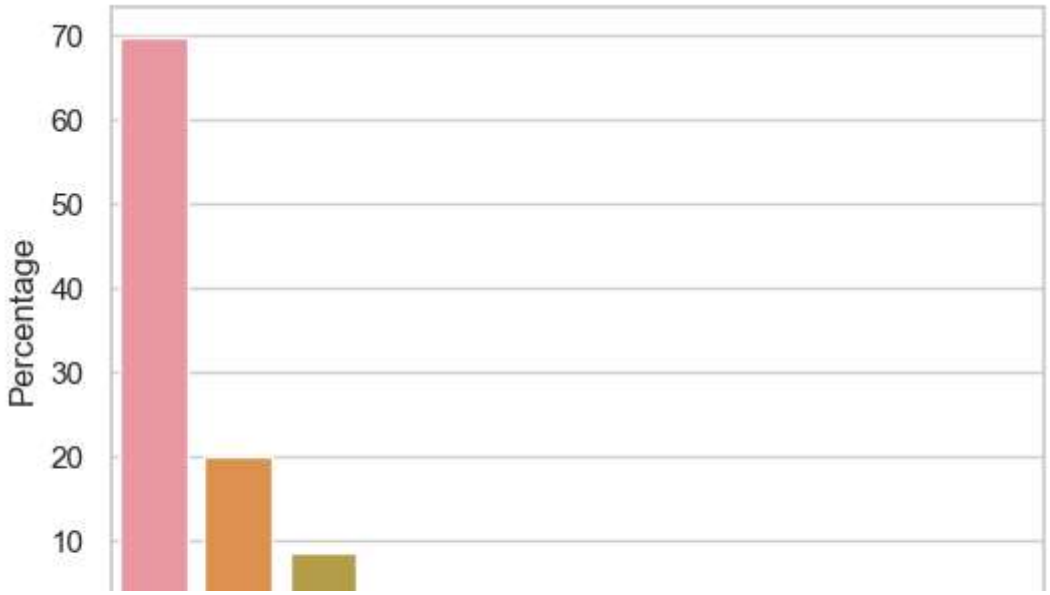
```
for i in application_data.columns:
    # Calculate the class frequencies
    class_frequencies = application_data[i].value_counts()

    # Create a bar chart to visualize Data imbalance.
    # Using if condition to give only those we are categorical variables.

    if len(class_frequencies)<=3:

        # printing the percentage of each value_count in list
        percentage = np.around((class_frequencies.values)/len(application_
        print(percentage)

        # Create a bar chart to visualize Data imbalance.
        plt.figure(figsize=(6, 4))
        sns.barplot(x = class_frequencies.index, y = (class_frequencies.va
        plt.xlabel(i)
        plt.ylabel("Percentage")
        plt.show()


```

```
[92.  8.]
```



In [36]:
```
for i in application_data.columns:
    # Calculate the class frequencies
    class_frequencies = application_data[i].value_counts()

    # Create a bar chart to visualize Data imbalance.
    # Using if condition to give only those we are categorical variables.

    if len(class_frequencies)<=15 and len(class_frequencies)>3:

        # printing the percentage of each value_count in list
        percentage = np.around((class_frequencies.values)/len(application_
        print(percentage)

        # Create a bar chart to visualize Data imbalance.
        plt.figure(figsize=(6, 4))
        sns.barplot(x = class_frequencies.index, y = (class_frequencies.va
        plt.xlabel(i)
        plt.ylabel("Percentage")
        plt.xticks(rotation=75)
        plt.show()
```

```
[70. 20.  9.  1.  0.  0.  0.  0.  0.  0.  0.]
```



Inference/Observation from the analysis of Data Imbalance

In [ ]:
```
1
```

In [37]:
```
# target variable shows us that 92% percent people have the value 1 i.e. c
# NAME_CONTRACT_TYPE has the 91% of loans of Cash Loans type.
# CODE_GENDER data imbalance shows us that there are 66% female applicatns
# FLAG_OWN_CAR data imbalance shows us that 66% applicants don't own car.
# FLAG_OWN_REALTY data imbalance shows us that 69% applicants own house.
# CNT_CHILDREN data imbalance shows us that 70% applicants don't have chil
# NAME_TYPE_SUITE data imbalance shows us that 81% of the applicants are u
# NAME_INCOME_TYPE data imbalance shows us that 52% applicants are working
# NAME_FAMILY_STATUS data imbalance shows us that 64% applicants are marri
# NAME_HOUSINGIN_TYPE data imbalance shows us that 52% applicants are live
# CNT_FAM_MEMBERES data imbalance shows us that 52% applicants have 2 fami
# and 22%, 17% applicants have 1 & 3 family members respectively
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

D. Perform Univariate, Segmented Univariate, and Bivariate Analysis

Univariate, Segmented Univariate, and Bivariate Analysis

In [38]:
```
1 application_data.columns
```

Out[38]:
```
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', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE',
       'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
       'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OWN_CAR_AG
E',
       'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE',
       'FLAG_PHONE', 'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS',
       'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY',
       'ORGANIZATION_TYPE', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
       'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
       'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3',
       'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6',
       'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9',
       'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12',
       'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15',
       'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',
       'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_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')
```

Univariate Analysis

Univariate analysis focuses on examining the distribution and summary statistics of individual variables.

In [39]:
```
1 # Calculated Summary Stats for all numeric variables using the describe fu
2
3 summary_stats = application_data.describe()
```

In [40]:
```
# createing the seperate table using the summary_stats for each variable

from tabulate import tabulate
# create separate tables for each variable
for col in summary_stats.columns:
    col_table = summary_stats[[col]].transpose()

    # Converted the table to a string using tabulate
    table_str = tabulate(col_table, tablefmt='grid')
    print(f"Summary Statistics for {col}:\n{table_str}\n")
    plt.show()
```

```
Summary Statistics for SK_ID_CURR:
+------------+-------+--------+---------+--------+--------+--------+------
--+---------+
| SK_ID_CURR | 49999 | 129013 | 16690.5 | 100002 | 114570 | 129076 | 14343
8 | 157875 |
+------------+-------+--------+---------+--------+--------+--------+------
--+---------+

Summary Statistics for TARGET:
+--------+-------+-----------+----------+---+---+---+---+---+
| TARGET | 49999 | 0.0805216 | 0.272102 | 0 | 0 | 0 | 0 | 1 |
+--------+-------+-----------+----------+---+---+---+---+---+

Summary Statistics for CNT_CHILDREN:
+--------------+-------+----------+----------+---+---+---+---+----+
| CNT_CHILDREN | 49999 | 0.419848 | 0.724039 | 0 | 0 | 0 | 1 | 11 |
+--------------+-------+----------+----------+---+---+---+---+----+

Summary Statistics for AMT_INCOME_TOTAL:
```

In [ ]:
```
1
```

Segmented Univariate Analysis:

In [41]:
```
for i in application_data.columns:
    # Calculate the class frequencies
    class_frequencies = application_data[i].value_counts()

    # Create a pie chart to performt Univariate Analysis.
    # Using if condition to give only those we are categorical variables.

    if len(class_frequencies)<=10:

        print(class_frequencies)
        # printing the percentage of each value_count in list
        percentage = np.around((class_frequencies.values)/len(application_
        print(percentage)

        # Create a bar chart to visualize Data imbalance.
        plt.figure(figsize=(6, 6))
        #sns.pie(x = class_frequencies.index, y = (class_frequencies.value
        sns.set_palette("Set2")  # You can choose a different color palet
        plt.pie((class_frequencies.values)/len(application_data[i])*100, 
        plt.xlabel(i)
        plt.ylabel("Percentage")
        plt.show()

```

```
0    45973
1     4026
Name: TARGET, dtype: int64
[92.  8.]
```

Inferences

target variable shows us that 92% percent people have the value 1 i.e. client with payment difficulties

NAME_CONTRACT_TYPE has the 91% of loans of Cash Loans type.

CODE_GENDER univariate analysis shows us that there are 66% female applicatns and 34% male applicants

FLAG_OWN_CAR univariate analysis shows us that 66% applicants don't own car.

FLAG_OWN_REALTY univariate analysis shows us that 69% applicants own house.

CNT_CHILDREN univariate analysis shows us that 70% applicants don't have childrens.

NAME_TYPE_SUITE univariate analysis shows us that 81% of the applicants are unaccompined

NAME_INCOME_TYPE univariate analysis shows us that 52% applicants are working.

NAME_FAMILY_STATUS univariate analysis shows us that 64% applicants are married.

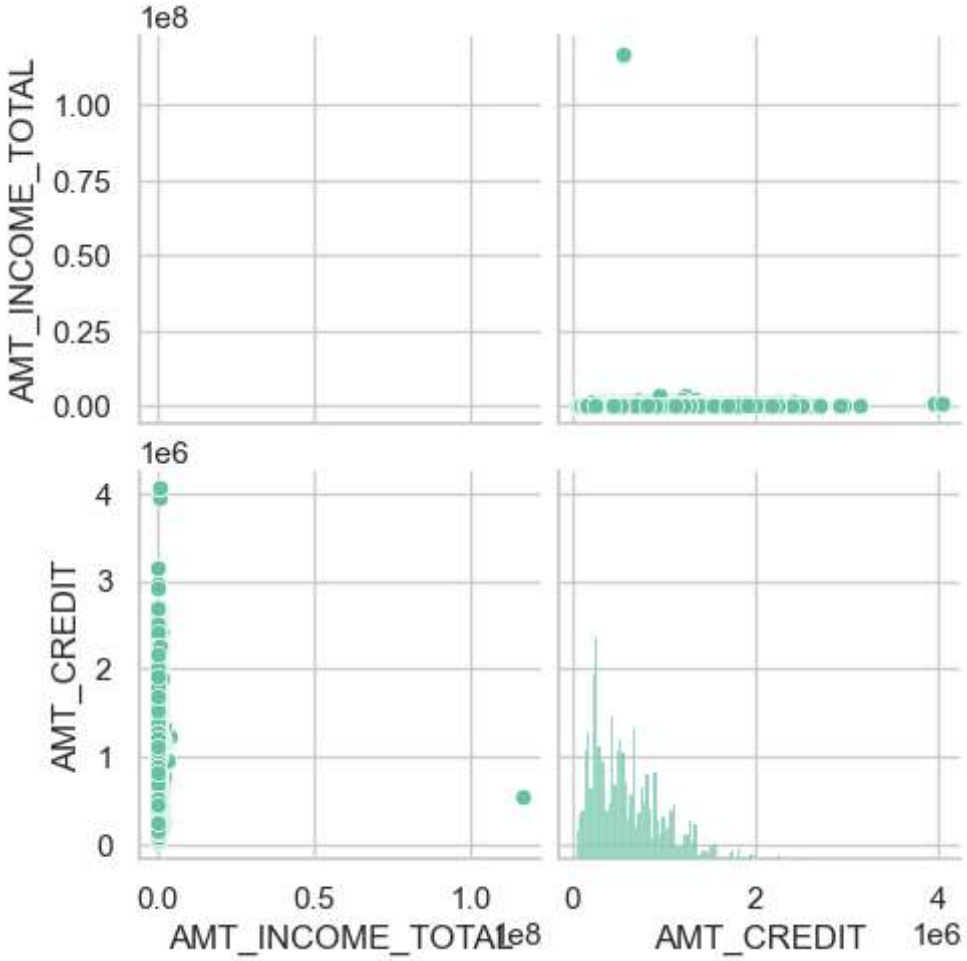
NAME_HOUSINGIN_TYPE univariate analysis shows us that 52% applicants are live in House/Apartment.

CNT_FAM_MEMBERES univariate analysis shows us that 52% applicants have 2 family membersand 22%, 17% applicants have 1 & 3 family members respectively

In [ ]:
```
1
```

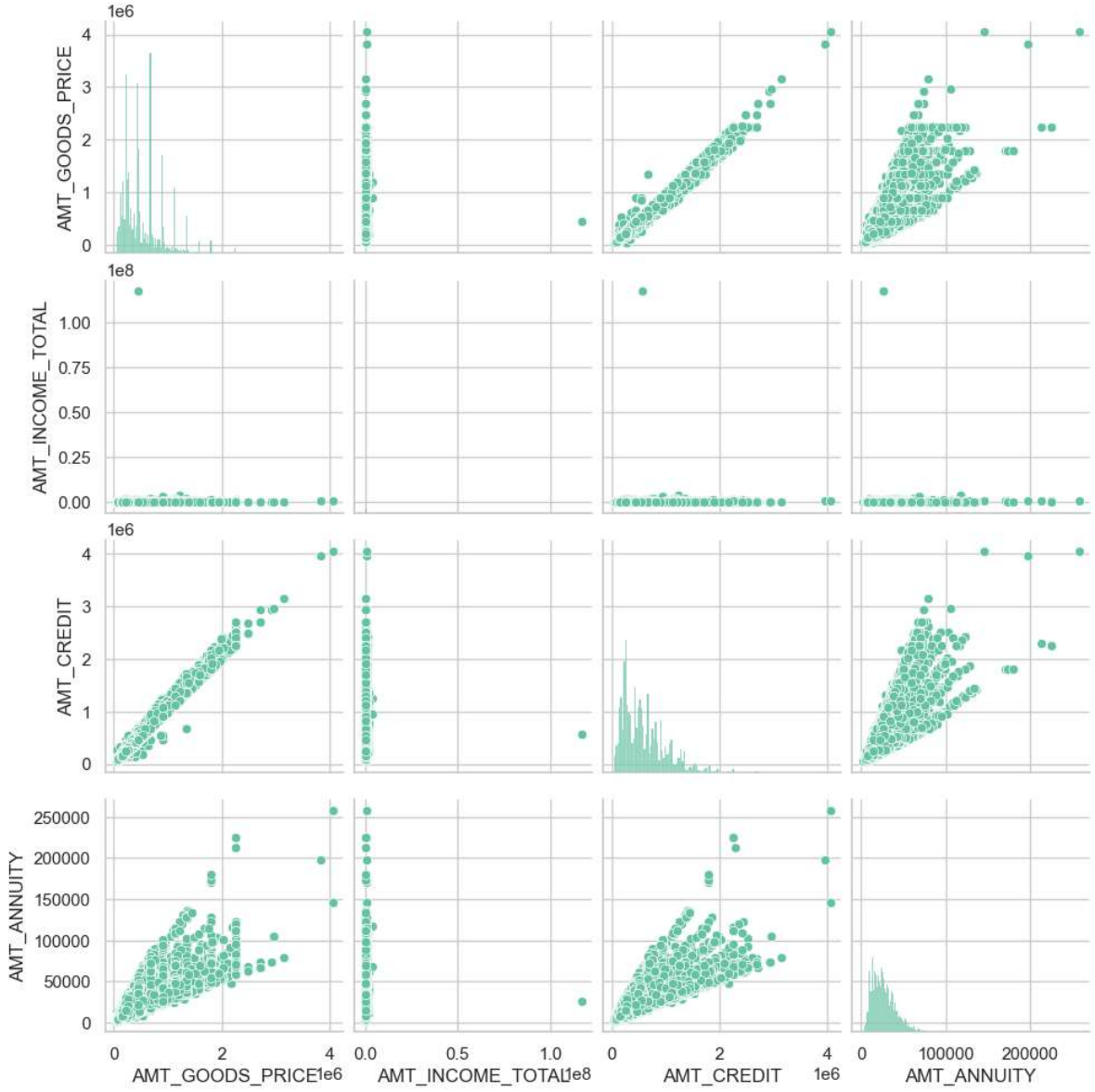
In [43]:

```
from scipy.stats import chi2_contingency

# Bivariate Analysis: Numeric vs. Numeric Variables
numeric_vars = ['AMT_INCOME_TOTAL', 'AMT_CREDIT']
numeric_df = application_data[numeric_vars]

# Calculate correlation coefficient
correlation_matrix = numeric_df.corr()

# Create scatter plots
sns.pairplot(numeric_df)
plt.show()
```



In [44]:
```
plt.figure(figsize=[20,8])
sns.pairplot(application_data[['AMT_GOODS_PRICE','AMT_INCOME_TOTAL','AMT_
plt.show()
```

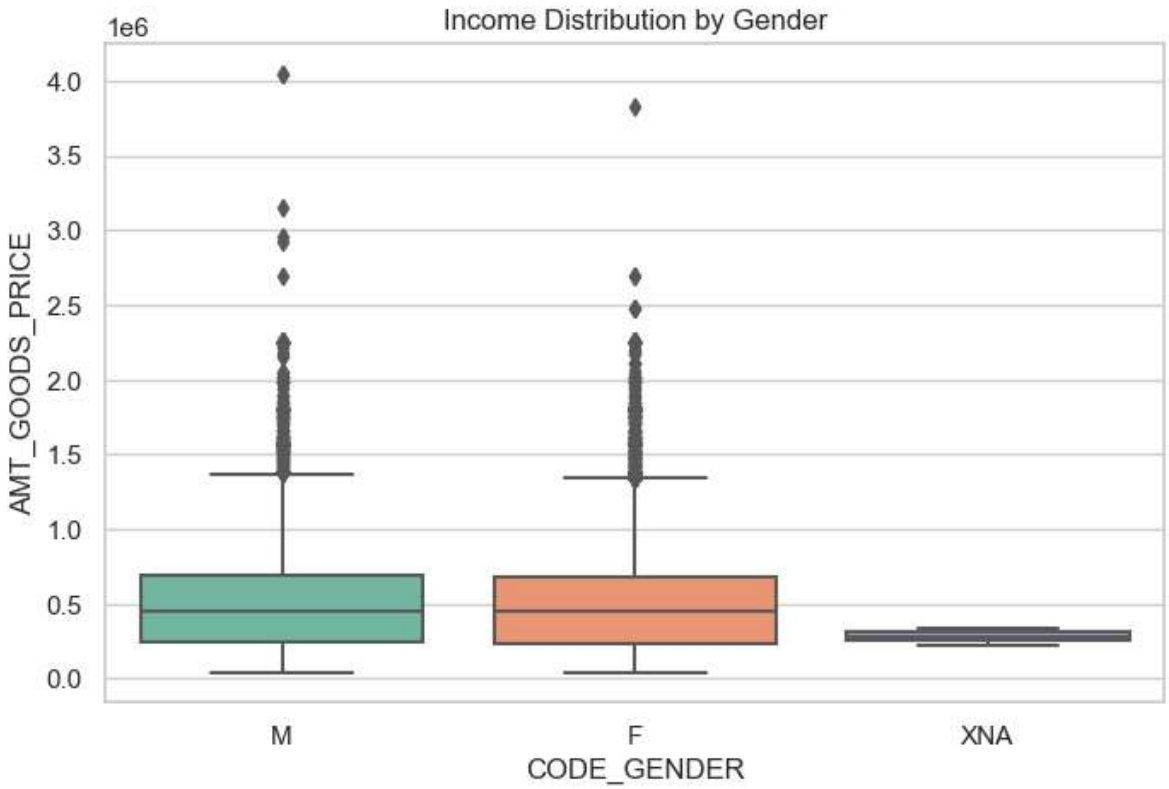
```
<Figure size 2000x800 with 0 Axes>
```

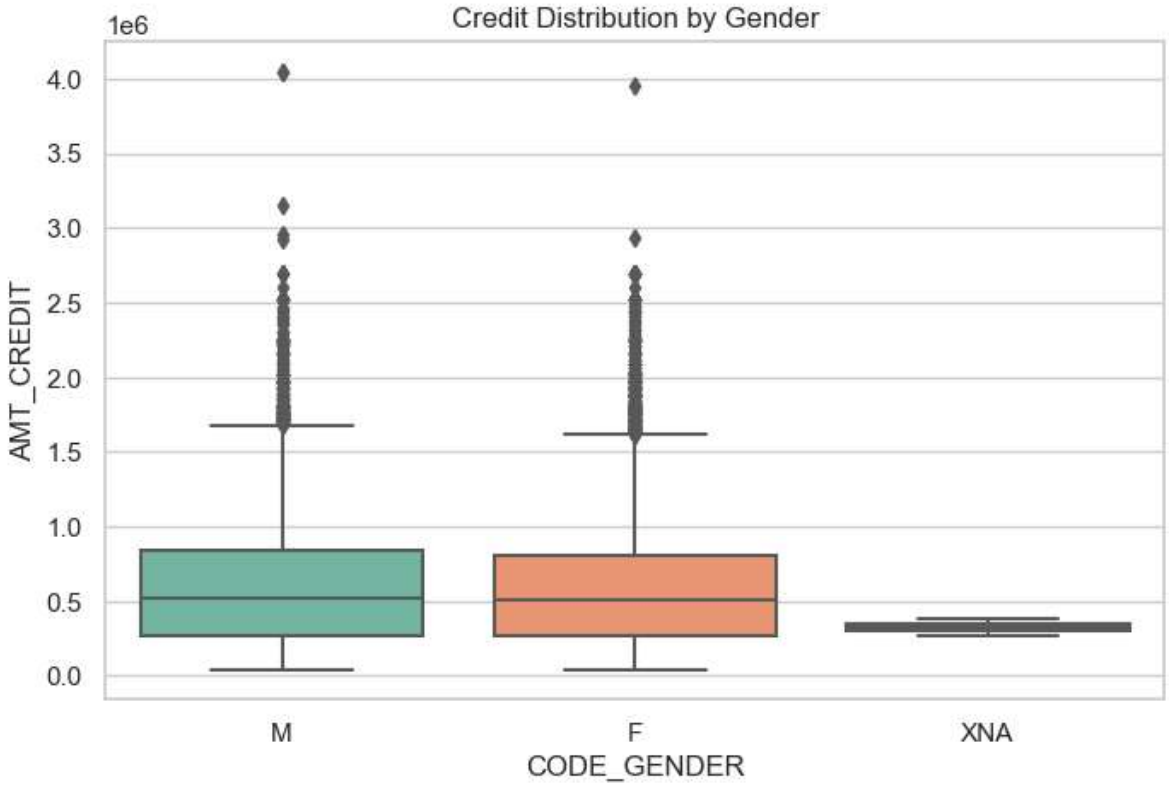


Inference

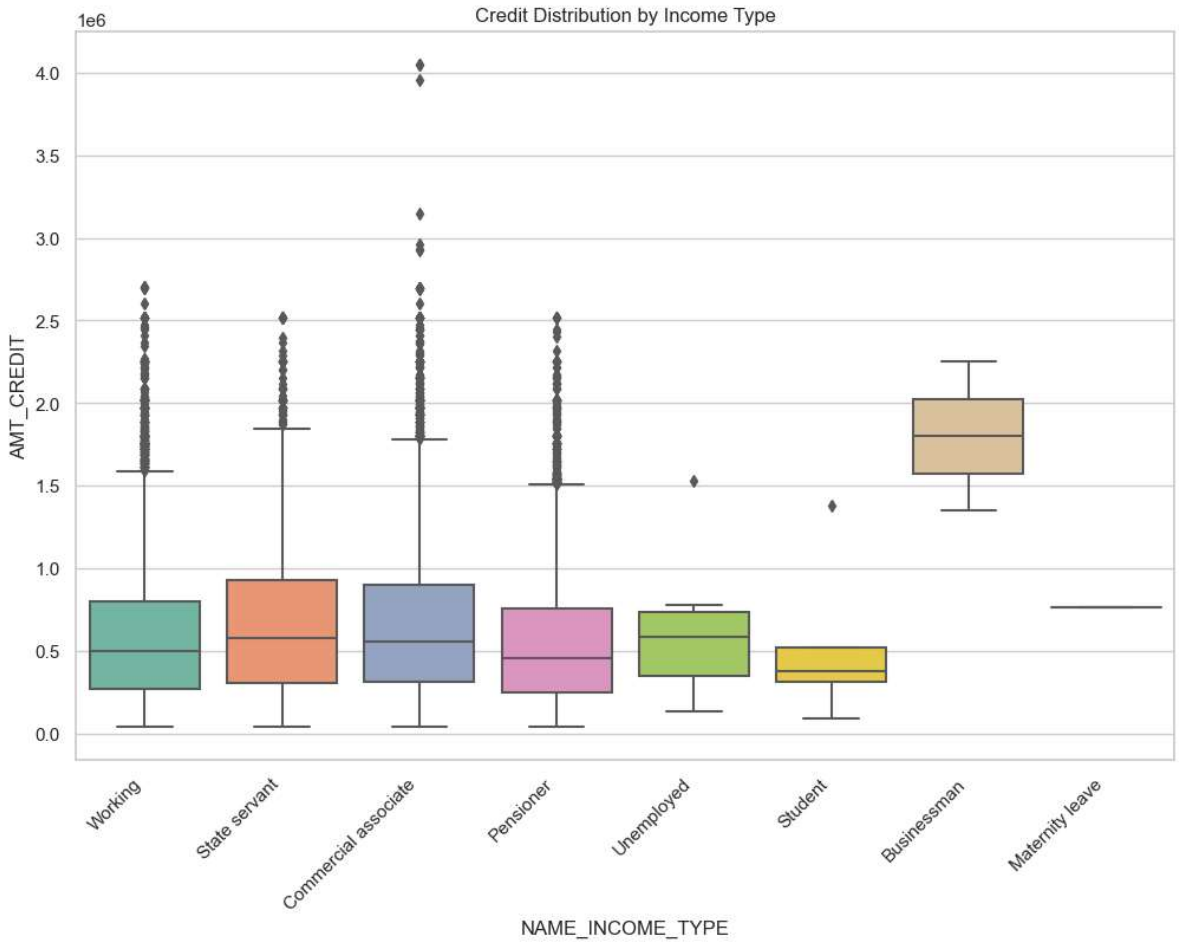
The variables AMT_GOODS_PRICE, AMT_ANNUITY, and AMT_CREDIT show a good positive correlation, which is expected due to the higher cost of goods leading to larger loan amounts and subsequent annuity payments.

In [45]:
```
# To find Income Distribution by Gender
plt.figure(figsize=(8, 5))
sns.boxplot(x='CODE_GENDER', y='AMT_GOODS_PRICE', data=application_data)
plt.title('Income Distribution by Gender')
plt.show()

# To find Credit Distribution by Gender
plt.figure(figsize=(8, 5))
sns.boxplot(x='CODE_GENDER', y='AMT_CREDIT', data=application_data)
plt.title('Credit Distribution by Gender')
plt.show()

```




Credit Distribution by Gender
1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
AMT_CREDIT
M
F
XNA
CODE_GENDER

In [46]:
```
# To find Credit Distribution by Income Type

plt.figure(figsize=(12, 8))
sns.boxplot(x='NAME_INCOME_TYPE', y='AMT_CREDIT', data=application_data)
plt.xticks(rotation=45, ha='right')
plt.title('Credit Distribution by Income Type')
plt.show()

```



Inference

_Bivariate analysis between AMT_GOODS_PRICE and AMT_CREDIT through box plots segmented by CODE_GENDER shows that both male and female applicants tend to receive similar loan amounts.

_Bivariate analysis of AMT_CREDIT through box plots segmented by NAME_INCOME_TYPE shows that Businessman get more Credit amount than that of other income types.

In [ ]:
```
1
```

E. Identify Top Correlations for Different Scenarios:

Top Correlations for Different Scenarios

In [85]:
```
# Select numerical columns for Univariate Analysis

numerical_columns = application_data.select_dtypes(include=[np.number])

numerical_columns = numerical_columns.drop(columns=["FLAG_DOCUMENT_3","FL
```

In [88]:
```
print(numerical_columns.shape)
numerical_columns.columns

```

```
(49999, 43)
```

Out[88]:
```
Index(['SK_ID_CURR', 'TARGET', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
       'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OWN_CAR_AGE', 'FLAG_MOBIL',
       'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE',
       'FLAG_EMAIL', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
       'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURCE_1',
       'EXT_SOURCE_2', 'EXT_SOURCE_3', 'OBS_30_CNT_SOCIAL_CIRCLE',
       'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE',
       'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE',
       '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')
```

In [93]:
```
# Generate heatmap

correlation = numerical_columns.corr()
print(correlation)
correlation.to_clipboard(index=True)
```

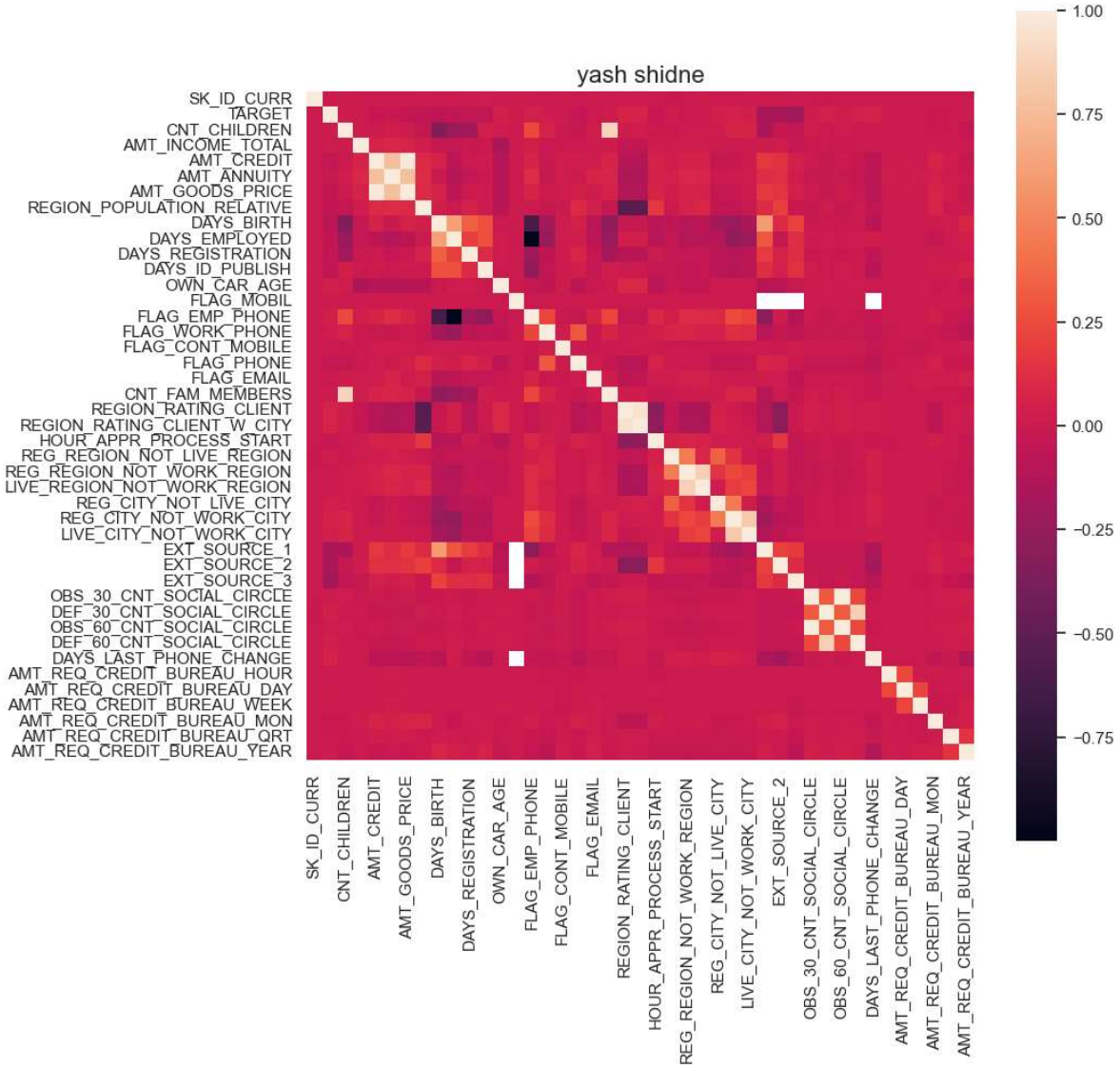
```
                            SK_ID_CURR    TARGET  CNT_CHILDREN  \
SK_ID_CURR                    1.000000  0.003295      0.005538
TARGET                        0.003295  1.000000      0.026364
CNT_CHILDREN                  0.005538  0.026364      1.000000
AMT_INCOME_TOTAL             -0.003014  0.010894      0.009589
AMT_CREDIT                   -0.000732 -0.032428      0.004972
AMT_ANNUITY                  -0.002084 -0.012399      0.026179
AMT_GOODS_PRICE              -0.000743 -0.041307      0.000253
REGION_POPULATION_RELATIVE    0.001979 -0.040799     -0.025556
DAYS_BIRTH                   -0.001324 -0.076788     -0.329264
DAYS_EMPLOYED                -0.004393 -0.042472     -0.241540
DAYS_REGISTRATION             0.003709 -0.042343     -0.181217
DAYS_ID_PUBLISH               0.008738 -0.046927      0.032116
OWN_CAR_AGE                   0.002485  0.039534      0.017437
FLAG_MOBIL                    0.002863  0.001323      0.002593
FLAG_EMP_PHONE                0.004449  0.041408      0.240678
FLAG_WORK_PHONE              -0.003127  0.021302      0.055881
FLAG_CONT_MOBILE              0.001012  0.006766     -0.002827
FLAG_PHONE                   -0.004005 -0.032679     -0.030654
```

In [90]:
```
print(correlation["AMT_ANNUITY"].sort_values(ascending = False), "\n")
```

```
AMT_ANNUITY                  1.000000
AMT_GOODS_PRICE              0.774434
AMT_CREDIT                   0.769499
EXT_SOURCE_2                 0.128928
EXT_SOURCE_1                 0.115437
REGION_POPULATION_RELATIVE   0.115112
FLAG_EMP_PHONE               0.109570
AMT_INCOME_TOTAL             0.083009
REG_REGION_NOT_WORK_REGION   0.081295
CNT_FAM_MEMBERS              0.077353
LIVE_REGION_NOT_WORK_REGION  0.074849
FLAG_EMAIL                   0.065896
HOUR_APPR_PROCESS_START      0.053275
REG_REGION_NOT_LIVE_REGION   0.044803
AMT_REQ_CREDIT_BUREAU_MON    0.039915
CNT_CHILDREN                 0.026179
EXT_SOURCE_3                 0.023602
FLAG_CONT_MOBILE             0.023111
AMT_REQ_CREDIT_BUREAU_WEEK   0.019918
AMT_REQ_CREDIT_BUREAU_HOUR   0.012014
LIVE_CITY_NOT_WORK_CITY      0.010940
AMT_REQ_CREDIT_BUREAU_QRT    0.009306
AMT_REQ_CREDIT_BUREAU_DAY    0.007011
FLAG_PHONE                   0.005127
REG_CITY_NOT_WORK_CITY       0.001061
FLAG_MOBIL                   0.000367
AMT_REQ_CREDIT_BUREAU_YEAR   0.000027
SK_ID_CURR                  -0.002084
DAYS_ID_PUBLISH             -0.006716
REG_CITY_NOT_LIVE_CITY      -0.006721
DAYS_BIRTH                  -0.007712
OBS_60_CNT_SOCIAL_CIRCLE    -0.009012
OBS_30_CNT_SOCIAL_CIRCLE    -0.009325
TARGET                      -0.012399
FLAG_WORK_PHONE             -0.020965
DEF_30_CNT_SOCIAL_CIRCLE    -0.021819
DEF_60_CNT_SOCIAL_CIRCLE    -0.025276
DAYS_REGISTRATION           -0.033219
DAYS_LAST_PHONE_CHANGE      -0.067257
OWN_CAR_AGE                 -0.096866
DAYS_EMPLOYED               -0.108710
REGION_RATING_CLIENT        -0.125803
REGION_RATING_CLIENT_W_CITY -0.139322
Name: AMT_ANNUITY, dtype: float64
```

In [91]:
```
1
2 f, ax = plt.subplots(figsize=(10, 10))
3 plt.title("yash shidne", y =1, size = 16)
4
5 sns.heatmap(correlation, square = True, vmax = 1, linewidths = 0.000001)
6
```

Out[91]: <AxesSubplot:title={'center':'yash shidne'}>



In [103]:
```
# Get the upper triangle of the correlation matrix so that there is no sar
upper_triangle_values = correlation.where(np.triu(np.ones(correlation.shap

# Find the top 5 highest correlations
top_positive_correlations = upper_triangle_values.unstack().sort_values(as

# Find the top 5 lowest correlations
top_negative_correlations = upper_triangle_values.unstack().sort_values(as

print("Top 5 highest correlations:")
print(top_positive_correlations)

print("\nTop 5 lowest correlations:")
print(top_negative_correlations)

```

```
Top 5 highest correlations:
OBS_60_CNT_SOCIAL_CIRCLE     OBS_30_CNT_SOCIAL_CIRCLE      0.998331
AMT_GOODS_PRICE              AMT_CREDIT                    0.986944
REGION_RATING_CLIENT_W_CITY  REGION_RATING_CLIENT          0.950710
CNT_FAM_MEMBERS              CNT_CHILDREN                  0.880430
LIVE_REGION_NOT_WORK_REGION  REG_REGION_NOT_WORK_REGION    0.857142
dtype: float64

Top 5 lowest correlations:
FLAG_EMP_PHONE               DAYS_EMPLOYED                -0.999746
                             DAYS_BIRTH                   -0.617703
REGION_RATING_CLIENT         REGION_POPULATION_RELATIVE   -0.532667
REGION_RATING_CLIENT_W_CITY  REGION_POPULATION_RELATIVE   -0.530439
DAYS_BIRTH                   CNT_CHILDREN                 -0.329264
dtype: float64

C:\Users\Yash\AppData\Local\Temp\ipykernel_8388\2900829348.py:2: DeprecationW
arning: `np.bool` is a deprecated alias for the builtin `bool`. To silence th
is warning, use `bool` by itself. Doing this will not modify any behavior and
is safe. If you specifically wanted the numpy scalar type, use `np.bool_` her
e.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations)
  upper_triangle_values = correlation.where(np.triu(np.ones(correlation.shap
e), k=1).astype(np.bool))
```

In [ ]:
```

```