In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
%matplotlib inline
```

Loading 'application_data'

In [2]:
```
df1 = pd.read_csv("application_data.csv")
df1.head(10)
```

Out[2]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTA
0	100002	1	Cash loans	M	N	Y	0	202500
1	100003	0	Cash loans	F	N	N	0	270000
2	100004	0	Revolving loans	M	Y	Y	0	67500
3	100006	0	Cash loans	F	N	Y	0	135000
4	100007	0	Cash loans	M	N	Y	0	121500
5	100008	0	Cash loans	M	N	Y	0	99000
6	100009	0	Cash loans	F	Y	Y	1	171000
7	100010	0	Cash loans	M	Y	Y	0	360000
8	100011	0	Cash loans	F	N	Y	0	112500
9	100012	0	Revolving loans	M	N	Y	0	135000

10 rows × 122 columns

In [3]:
```
df1.shape
```

Out[3]:
```
(49999, 122)
```

In [4]:
```
df1.describe()
```

Out[4]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULAT
count	49999.000000	49999.000000	49999.000000	4.999900e+04	4.999900e+04	49998.000000	4.996100e+04	
mean	129013.210584	0.080522	0.419848	1.707676e+05	5.997006e+05	27107.377355	5.390600e+05	
std	16690.512048	0.272102	0.724039	5.318191e+05	4.024154e+05	14562.944435	3.698533e+05	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	2052.000000	4.500000e+04	
25%	114570.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16456.500000	2.385000e+05	
50%	129076.000000	0.000000	0.000000	1.458000e+05	5.147775e+05	24939.000000	4.500000e+05	
75%	143438.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	
max	157875.000000	1.000000	11.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	

8 rows × 106 columns

A. Identify Missing Data and Deal with it Appropriately

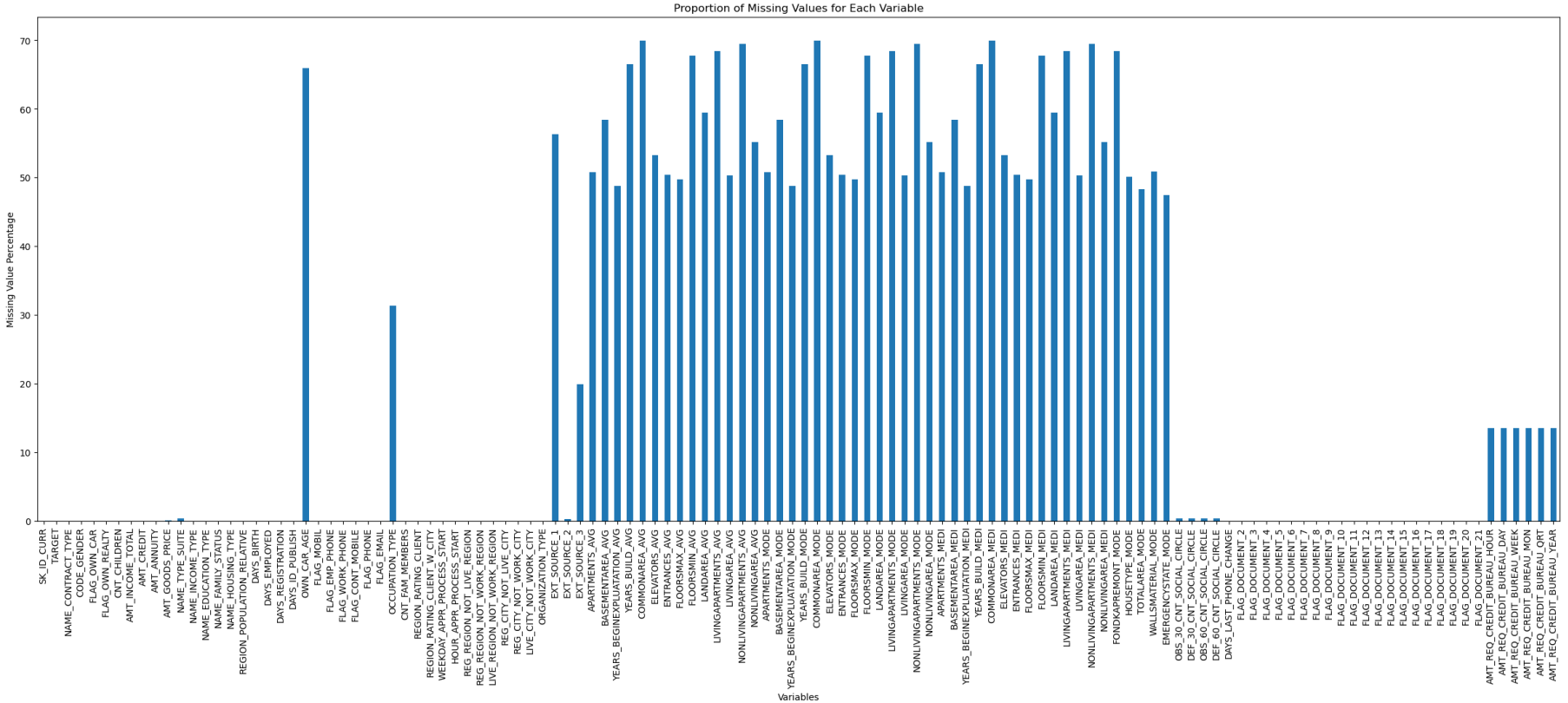
In [5]:
```
df1.isnull().sum().sum() # Missing Vlues in dataset
```

Out[5]:
```
1488212
```

Lets check the missing values for each column by visualization and percentage

In [6]:
```
missing_proportions = (df1.isnull().sum() / len(df1)) * 100
```

In [7]:
```
# Creating bar plot to see the missing values
plt.figure(figsize=(30, 10))
missing_proportions.plot(kind='bar')
plt.xlabel('Variables')
plt.ylabel('Missing Value Percentage')
plt.title('Proportion of Missing Values for Each Variable')
plt.xticks(rotation=90)

plt.show()
```



In [8]:
```
# Calculate the percentage of missing values for each column
missing_percentage = (df1.isnull().sum() / len(df1)) * 100
```

In [9]:
```
# we need column Occupation Type which has 32% missing data
# Create a mask to identify columns with more than 32% missing values
mask = missing_percentage > 32
```

In [10]:
```
# Filter columns with more than 32% missing values
columns_with_more_than_32_percent_missing = missing_percentage[mask]
```

In [11]:
```
print(columns_with_more_than_32_percent_missing)
```

OWN_CAR_AGE	65.901318
EXT_SOURCE_1	56.345127
APARTMENTS_AVG	50.771015
BASEMENTAREA_AVG	58.399168
YEARS_BEGINEXPLUATATION_AVG	48.788976
YEARS_BUILD_AVG	66.479330
COMMONAREA_AVG	69.921398
ELEVATORS_AVG	53.303066
ENTRANCES_AVG	50.391008
FLOORSMAX_AVG	49.750995
FLOORSMIN_AVG	67.789356
LANDAREA_AVG	59.443189
LIVINGAPARTMENTS_AVG	68.453369
LIVINGAREA_AVG	50.275006
NONLIVINGAPARTMENTS_AVG	69.429389
NONLIVINGAREA_AVG	55.145103
APARTMENTS_MODE	50.771015
BASEMENTAREA_MODE	58.399168
YEARS_BEGINEXPLUATATION_MODE	48.788976
YEARS_BUILD_MODE	66.479330
COMMONAREA_MODE	69.921398
ELEVATORS_MODE	53.303066
ENTRANCES_MODE	50.391008
FLOORSMAX_MODE	49.750995
FLOORSMIN_MODE	67.789356
LANDAREA_MODE	59.443189
LIVINGAPARTMENTS_MODE	68.453369
LIVINGAREA_MODE	50.275006
NONLIVINGAPARTMENTS_MODE	69.429389
NONLIVINGAREA_MODE	55.145103
APARTMENTS_MEDI	50.771015
BASEMENTAREA_MEDI	58.399168
YEARS_BEGINEXPLUATATION_MEDI	48.788976
YEARS_BUILD_MEDI	66.479330
COMMONAREA_MEDI	69.921398
ELEVATORS_MEDI	53.303066
ENTRANCES_MEDI	50.391008
FLOORSMAX_MEDI	49.750995
FLOORSMIN_MEDI	67.789356
LANDAREA_MEDI	59.443189
LIVINGAPARTMENTS_MEDI	68.453369

```
LIVINGAREA_MEDI                 50.275006
NONLIVINGAPARTMENTS_MEDI        69.429389
NONLIVINGAREA_MEDI              55.145103
FONDKAPREMONT_MODE              68.383368
HOUSETYPE_MODE                  50.151003
TOTALAREA_MODE                  48.296966
WALLSMATERIAL_MODE              50.919018
EMERGENCYSTATE_MODE             47.396948
dtype: float64
```

In [12]:
```
columns_with_more_than_32_percent_missing.count() # Total count of missing columns
```

Out[12]:
```
49
```

Operations to perform on missing data

1.Remove columns which has more than 32% missing data

2.Replace the missing values with "median" for numeric data and "mode" for categorical data.

*Remove unnecessary columns as well which are no use in analysis

In [13]:
```
# Dropping the columns which have missing values more than 32% as well as unnecessary columns
new_app = pd.DataFrame(df1.drop(['NAME_TYPE_SUITE','DAYS_REGISTRATION','DAYS_ID_PUBLISH','OWN_CAR_AGE','FLAG_MOBIL','F
,'APARTMENTS_AVG','BASEMENTAREA_AVG','YEARS_BEGINEXPLUATATION_AVG','YEARS_BUILD_AVG', 'COMMONAREA_AVG','ELEVATORS_AVG'
 'NONLIVINGAREA_AVG','APARTMENTS_MODE','BASEMENTAREA_MODE','YEARS_BEGINEXPLUATATION_MODE','YEARS_BUILD_MODE','COMMONAR
 'NONLIVINGAPARTMENTS_MEDI','NONLIVINGAREA_MEDI','FONDKAPREMONT_MODE','HOUSETYPE_MODE','TOTALAREA_MODE','WALLSMATERIAL
 'FLAG_DOCUMENT_8','FLAG_DOCUMENT_9','FLAG_DOCUMENT_10','FLAG_DOCUMENT_11','FLAG_DOCUMENT_12','FLAG_DOCUMENT_13','FLAG
```

In [14]:
```
new_app.shape
```

Out[14]:
```
(49999, 22)
```

In [15]:
```
new_app.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Data columns (total 22 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   SK_ID_CURR                   49999 non-null  int64  
 1   TARGET                       49999 non-null  int64  
 2   NAME_CONTRACT_TYPE           49999 non-null  object 
 3   CODE_GENDER                  49999 non-null  object 
 4   FLAG_OWN_CAR                 49999 non-null  object 
 5   FLAG_OWN_REALTY              49999 non-null  object 
 6   CNT_CHILDREN                 49999 non-null  int64  
 7   AMT_INCOME_TOTAL             49999 non-null  float64
 8   AMT_CREDIT                   49999 non-null  float64
 9   AMT_ANNUITY                  49998 non-null  float64
 10  AMT_GOODS_PRICE              49961 non-null  float64
 11  NAME_INCOME_TYPE             49999 non-null  object 
 12  NAME_EDUCATION_TYPE          49999 non-null  object 
 13  NAME_FAMILY_STATUS           49999 non-null  object 
 14  NAME_HOUSING_TYPE            49999 non-null  object 
 15  REGION_POPULATION_RELATIVE   49999 non-null  float64
 16  DAYS_BIRTH                   49999 non-null  int64  
 17  DAYS_EMPLOYED                49999 non-null  int64  
 18  OCCUPATION_TYPE              34345 non-null  object 
 19  CNT_FAM_MEMBERS              49998 non-null  float64
 20  ORGANIZATION_TYPE            49999 non-null  object 
 21  DAYS_LAST_PHONE_CHANGE       49998 non-null  float64
dtypes: float64(7), int64(5), object(10)
memory usage: 8.4+ MB
```

In [16]:
```
new_app.isnull().sum().sum()
```

Out[16]:
```
15695
```

Filling the remaining missing values with 'median' for numeric data and "mode" for categorical data.

In [17]:
```
# Replace missing values in 'Categorical' columns with the mode
new_app['OCCUPATION_TYPE'].fillna(new_app['OCCUPATION_TYPE'].mode()[0], inplace=True)
```

*Replacing the 'XNA' values from ORGANIZATION_TYPE with 'Not Disclosed'

In [23]:
```
new_app.ORGANIZATION_TYPE.replace("XNA","Not Disclosed", inplace=True)
```

In [24]:
```
new_app.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Data columns (total 22 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   SK_ID_CURR                   49999 non-null  int64  
 1   TARGET                       49999 non-null  int64  
 2   NAME_CONTRACT_TYPE           49999 non-null  object 
 3   CODE_GENDER                  49999 non-null  object 
 4   FLAG_OWN_CAR                 49999 non-null  object 
 5   FLAG_OWN_REALTY              49999 non-null  object 
 6   CNT_CHILDREN                 49999 non-null  int64  
 7   AMT_INCOME_TOTAL             49999 non-null  float64
 8   AMT_CREDIT                   49999 non-null  float64
 9   AMT_ANNUITY                  49998 non-null  float64
 10  AMT_GOODS_PRICE              49961 non-null  float64
 11  NAME_INCOME_TYPE             49999 non-null  object 
 12  NAME_EDUCATION_TYPE          49999 non-null  object 
 13  NAME_FAMILY_STATUS           49999 non-null  object 
 14  NAME_HOUSING_TYPE            49999 non-null  object 
 15  REGION_POPULATION_RELATIVE   49999 non-null  float64
 16  DAYS_BIRTH                   49999 non-null  int64  
 17  DAYS_EMPLOYED                49999 non-null  int64  
 18  OCCUPATION_TYPE              49999 non-null  object 
 19  CNT_FAM_MEMBERS              49998 non-null  float64
 20  ORGANIZATION_TYPE            49999 non-null  object 
 21  DAYS_LAST_PHONE_CHANGE       49998 non-null  float64
dtypes: float64(7), int64(5), object(10)
memory usage: 8.4+ MB
```

In [25]:
```
new_app.shape
```

Out[25]:
```
(49999, 22)
```

In [26]:
```
# Filling missing values for numeric columns with median
new_app_missing =['AMT_ANNUITY','AMT_GOODS_PRICE','CNT_FAM_MEMBERS','DAYS_LAST_PHONE_CHANGE']
```

In [27]:
```
for i in new_app_missing:
    new_app[i].fillna(new_app[i].median(), inplace=True)
```

In [28]:
```
new_app.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Data columns (total 22 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   SK_ID_CURR                   49999 non-null  int64  
 1   TARGET                       49999 non-null  int64  
 2   NAME_CONTRACT_TYPE           49999 non-null  object 
 3   CODE_GENDER                  49999 non-null  object 
 4   FLAG_OWN_CAR                 49999 non-null  object 
 5   FLAG_OWN_REALTY              49999 non-null  object 
 6   CNT_CHILDREN                 49999 non-null  int64  
 7   AMT_INCOME_TOTAL             49999 non-null  float64
 8   AMT_CREDIT                   49999 non-null  float64
 9   AMT_ANNUITY                  49999 non-null  float64
 10  AMT_GOODS_PRICE              49999 non-null  float64
 11  NAME_INCOME_TYPE             49999 non-null  object 
 12  NAME_EDUCATION_TYPE          49999 non-null  object 
 13  NAME_FAMILY_STATUS           49999 non-null  object 
 14  NAME_HOUSING_TYPE            49999 non-null  object 
 15  REGION_POPULATION_RELATIVE   49999 non-null  float64
 16  DAYS_BIRTH                   49999 non-null  int64  
 17  DAYS_EMPLOYED                49999 non-null  int64  
 18  OCCUPATION_TYPE              49999 non-null  object 
 19  CNT_FAM_MEMBERS              49999 non-null  float64
 20  ORGANIZATION_TYPE            49999 non-null  object 
 21  DAYS_LAST_PHONE_CHANGE       49999 non-null  float64
dtypes: float64(7), int64(5), object(10)
memory usage: 8.4+ MB
```

Columns DAYS_BIRTH, DAYS_EMPLOYED, DAYS_LAST_PHONE_CHANGE have negative days values, so we will convert them into with positive days & years for analysis and readability

In [29]:
```
new_app['AGE'] = abs(new_app['DAYS_BIRTH']//365) # Divide 'DAYS_BIRTH' by 365 for taking Age
```

In [30]:
```
new_app = new_app.drop(['DAYS_BIRTH'],axis=1) # Drop 'DAYS_BIRTH' column
```

In [31]:
```
absolute_values= ["DAYS_EMPLOYED","DAYS_LAST_PHONE_CHANGE"] # changing negative values into +ve
for i in absolute_values:
    new_app[i]= new_app[i].abs()
```

In [32]:
```
new_app['YEARS_EMPLOYED'] = abs(new_app['DAYS_EMPLOYED']//365) # Divide 'DAYS_EMPLOYED' by 365 for taking employment i
```

In [33]:
```
new_app = new_app.drop(['DAYS_EMPLOYED'],axis=1) # Drop 'DAYS_BIRTH' column
```

In [34]:
```
new_app.head()
```

Out[34]:

'PE	REGION_POPULATION_RELATIVE	OCCUPATION_TYPE	CNT_FAM_MEMBERS	ORGANIZATION_TYPE	DAYS_LAST_PHONE_CHANGE	AGE	YEARS
ıent	0.018801	Laborers	1.0	Business Entity Type 3	1134.0	26	
ıent	0.003541	Core staff	2.0	School	828.0	46	
ıent	0.010032	Laborers	1.0	Government	815.0	53	
ıent	0.008019	Laborers	2.0	Business Entity Type 3	617.0	53	
ıent	0.028663	Core staff	1.0	Religion	1106.0	55	

In [35]:
```
new_app.describe()
```

Out[35]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULAT
count	49999.000000	49999.000000	49999.000000	4.999900e+04	4.999900e+04	49999.000000	4.999900e+04	
mean	129013.210584	0.080522	0.419848	1.707676e+05	5.997006e+05	27107.333987	5.389923e+05	
std	16690.512048	0.272102	0.724039	5.318191e+05	4.024154e+05	14562.802028	3.697208e+05	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	2052.000000	4.500000e+04	
25%	114570.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16456.500000	2.385000e+05	
50%	129076.000000	0.000000	0.000000	1.458000e+05	5.147775e+05	24939.000000	4.500000e+05	
75%	143438.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	
max	157875.000000	1.000000	11.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	

In [183]:
```
new_app.to_excel("applications_Cleaned data.xlsx")
```

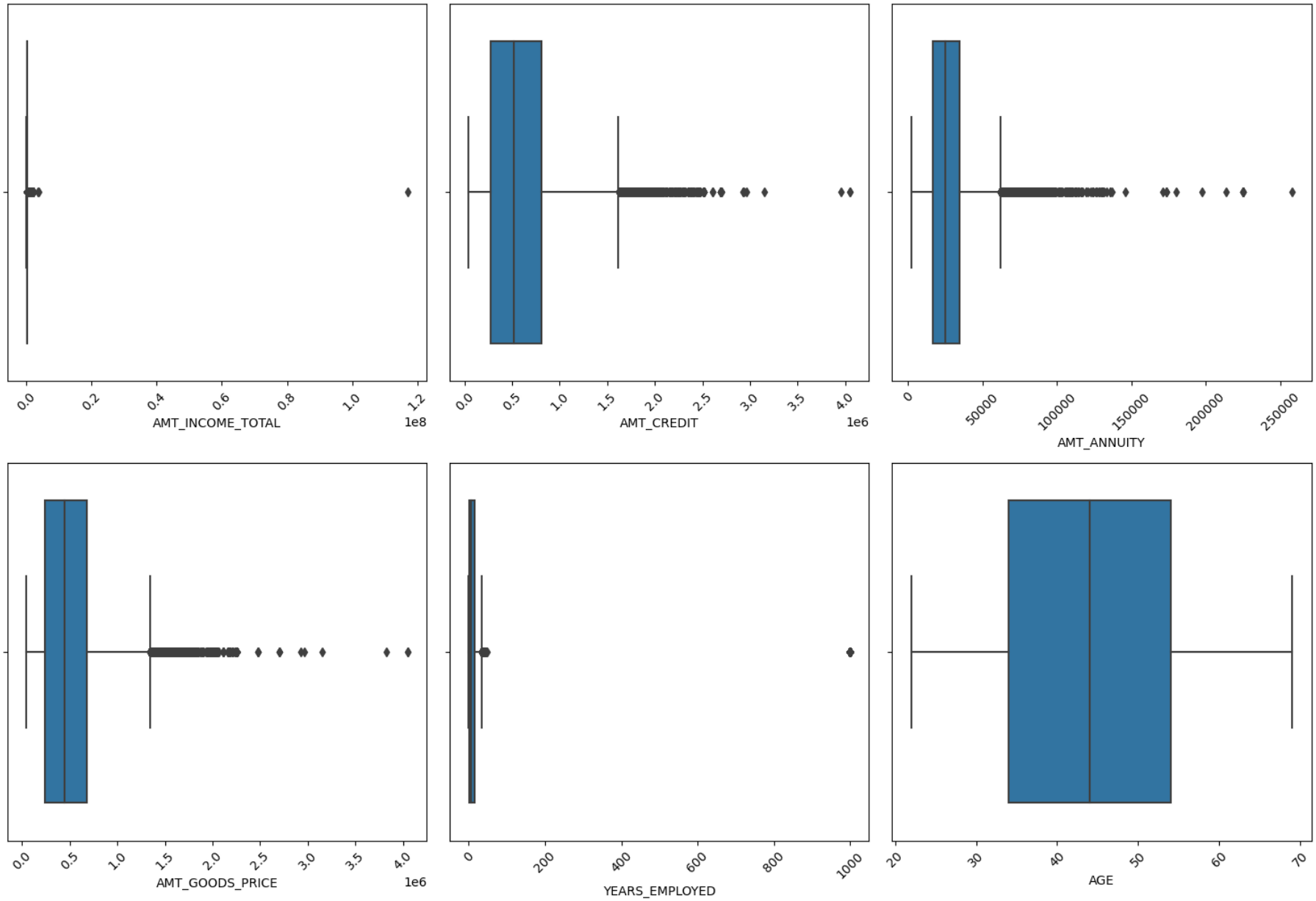
B. Identify Outliers in the Dataset:

We will use the Boxplot visualization to check the outliers for below variables for which we can see some huge numbers in statistical description above

In [36]:
```
columns_to_check = new_app[['AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY','AMT_GOODS_PRICE','YEARS_EMPLOYED',
                            'AGE']]
```

In [37]:

```
plt.figure(figsize=(15, 20))

for i, column_name in enumerate(columns_to_check):
    plt.subplot(4, 3, i + 1)
    sns.boxplot(data=new_app, x=column_name)
    plt.xlabel(column_name)
    plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



1. AMT_ANNUITY, AMT_CREDIT, AMT_GOODS_PRICE have some number of outliers.
2. AMT_INCOME_TOTAL has huge number of outliers which indicate that few of the loan applicants have high income compared to the others.
3. YEARS_EMPLOYED has outlier value around 1000(years) which is impossible and hence this has to be incorrect entry.

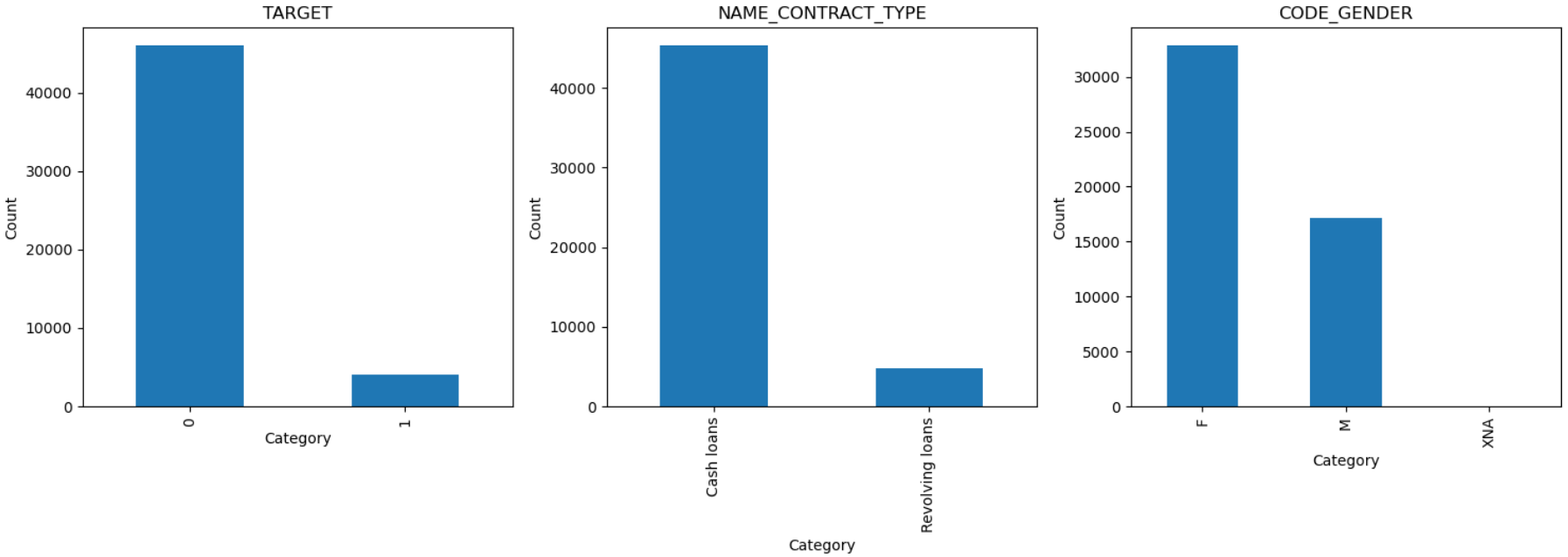
4. AGE column has no outliers.

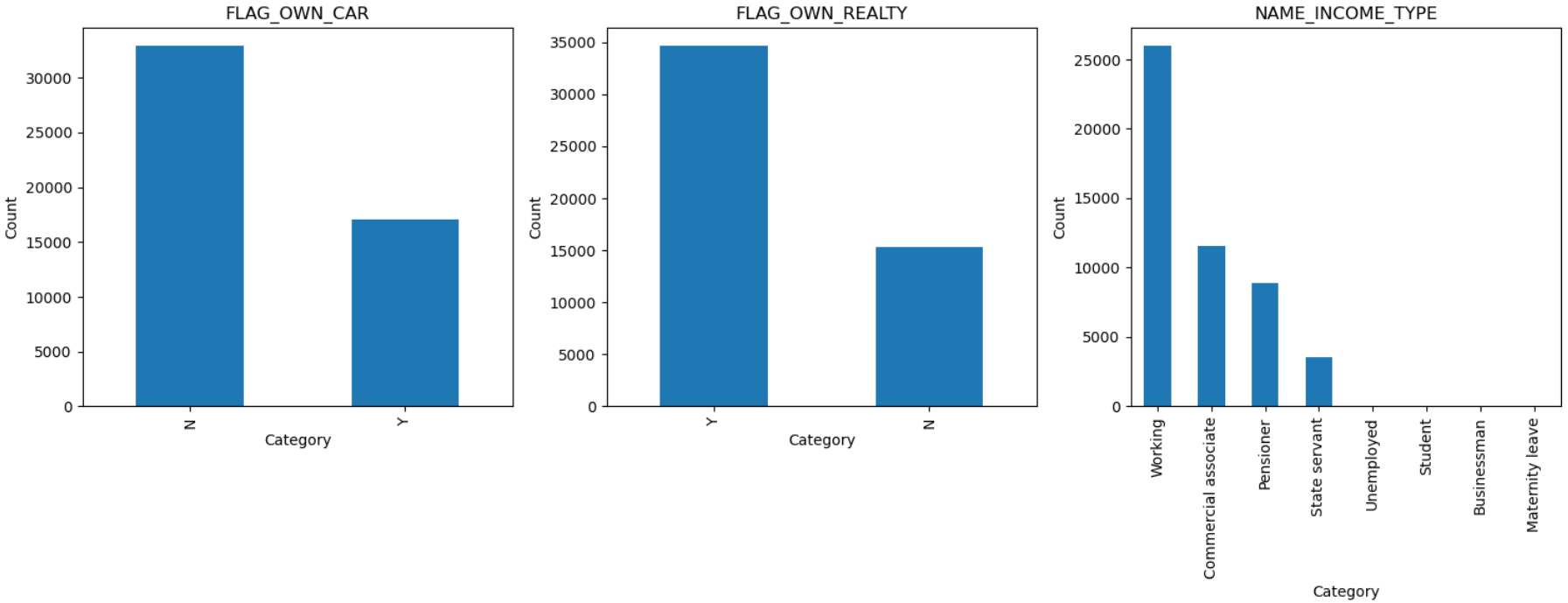
C. Analyze Data Imbalance:

In [38]:
```
class_distribution = new_app['TARGET'].value_counts()
imbalance_ratio = class_distribution.min() / class_distribution.max()

print(f"Imbalance ratio: {imbalance_ratio:.2f}")
```

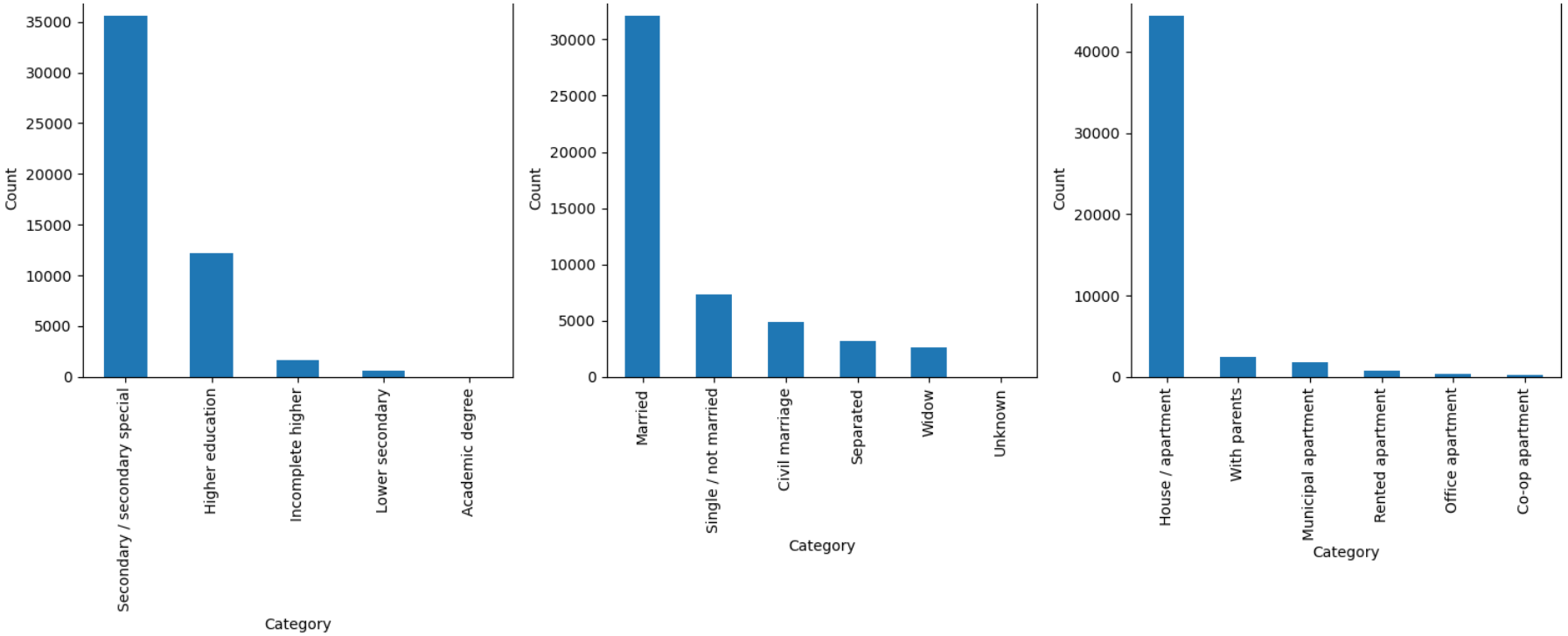
```
Imbalance ratio: 0.09
```

In [39]:
```
variables_to_check = ['TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
                      'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE']

num_plots = len(variables_to_check)
num_rows = (num_plots + 2) // 3
num_cols = 3

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 6 * num_rows))

for i, variable in enumerate(variables_to_check):
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col]

    value_counts = new_app[variable].value_counts()
    value_counts.plot(kind='bar', ax=ax)

    ax.set_title(variable)
    ax.set_xlabel('Category')
    ax.set_ylabel('Count')

for i in range(num_plots, num_rows * num_cols):
    fig.delaxes(axes.flatten()[i])

plt.tight_layout()
plt.show()
```


TARGET
Count
Category
0
1
NAME_CONTRACT_TYPE
Cash loans
Revolving loans
CODE_GENDER
F
M
XNA


FLAG_OWN_CAR
N
Y
FLAG_OWN_REALTY
NAME_INCOME_TYPE
Working
Commercial associate
Pensioner
State servant
Unemployed
Student
Businessman
Maternity leave


NAME_EDUCATION_TYPE
NAME_FAMILY_STATUS
NAME_HOUSING_TYPE



*We have identified data imbalance in several columns:

TARGET: The distribution of defaulters (1) is significantly lower compared to non-defaulters (0). This indicates an imbalance in the target variable, with a majority of clients being non-defaulters.

NAME_CONTRACT_TYPE: There is an imbalance between the number of revolving loans and cash loans. The majority of loans are cash loans, while the number of revolving loans is relatively low.

NAME_EDUCATION_TYPE: The majority of loan applicants have secondary or secondary special education. This indicates an imbalance in the education level of applicants.

NAME_FAMILY_STATUS: Married individuals represent a significant majority among loan applicants. This suggests an imbalance in the marital status of applicants.

NAME_HOUSING_TYPE: The majority of loan applications come from individuals who own a home or apartment. This indicates an imbalance in the type of housing for loan applicants.

Binning of variables AMT_INCOME_TOTAL, AMT_CREDIT, AMT_ANNUITY, AGE for segmented analysis

In [40]:
```
# Binning for AMT_INCOME_TOTAL
income_bins = [0, 25000, 50000, 75000, 100000, float("inf")]
income_labels = ['0-25K', '25K-50K', '50K-75K', '75K-100K', '100K+']
new_app['INCOME_GROUP'] = pd.cut(new_app['AMT_INCOME_TOTAL'], bins=income_bins, labels=income_labels)

# Binning for AMT_CREDIT
credit_bins = [0, 100000, 200000, 300000, 400000, float("inf")]
credit_labels = ['0-100K', '100K-200K', '200K-300K', '300K-400K', '400K+']
new_app['CREDIT_GROUP'] = pd.cut(new_app['AMT_CREDIT'], bins=credit_bins, labels=credit_labels)

# Binning for AMT_ANNUITY
annuity_bins = [0, 5000, 10000, 15000, 20000, float("inf")]
annuity_labels = ['0-5K', '5K-10K', '10K-15K', '15K-20K', '20K+']
new_app['ANNUITY_GROUP'] = pd.cut(new_app['AMT_ANNUITY'], bins=annuity_bins, labels=annuity_labels)

# Binning for AGE
age_bins = [0, 30, 40, 50, 60, float("inf")]
age_labels = ['0-30', '30-40', '40-50', '50-60', '60+']
new_app['AGE_GROUP'] = pd.cut(new_app['AGE'], bins=age_bins, labels=age_labels)
```

In [41]:
```
new_app["YEARS_EMP_GROUP"]= pd.cut(new_app.YEARS_EMPLOYED,bins=[-1,5,10,20,30,40,50,60,1000],labels=["0-5","5-10","10-
```

In [42]:
```
new_app.head()
```

Out[42]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTA
0	100002	1	Cash loans	M	N	Y	0	202500
1	100003	0	Cash loans	F	N	N	0	270000
2	100004	0	Revolving loans	M	Y	Y	0	67500
3	100006	0	Cash loans	F	N	Y	0	135000
4	100007	0	Cash loans	M	N	Y	0	121500

5 rows × 27 columns

D. Perform Univariate, Segmented Univariate, and Bivariate Analysis:

Univariate Analysis - Categorical Ordered & Unordered

We will create Plot Functions so that we can reuse those plots for multiple variables

In [43]:
```
def use_bar_plot(Column,dataframe):
    dataframe[Column].value_counts().plot.bar(color="violet")
    plt.title("Bar Chart of " + Column.title(), fontdict={"fontsize":12,"fontweight":5,"color":"Black"})
    plt.grid(zorder=1, axis="y")
    plt.show()
```

In [44]:
```
def use_barh_plot(Column,dataframe):
    dataframe[Column].value_counts().plot.barh(color="orange")
    plt.title("Bar Chart of " + Column.title(), fontdict={"fontsize":12,"fontweight":5,"color":"Black"})
    plt.grid(zorder=1, axis="x")
    plt.show()
```

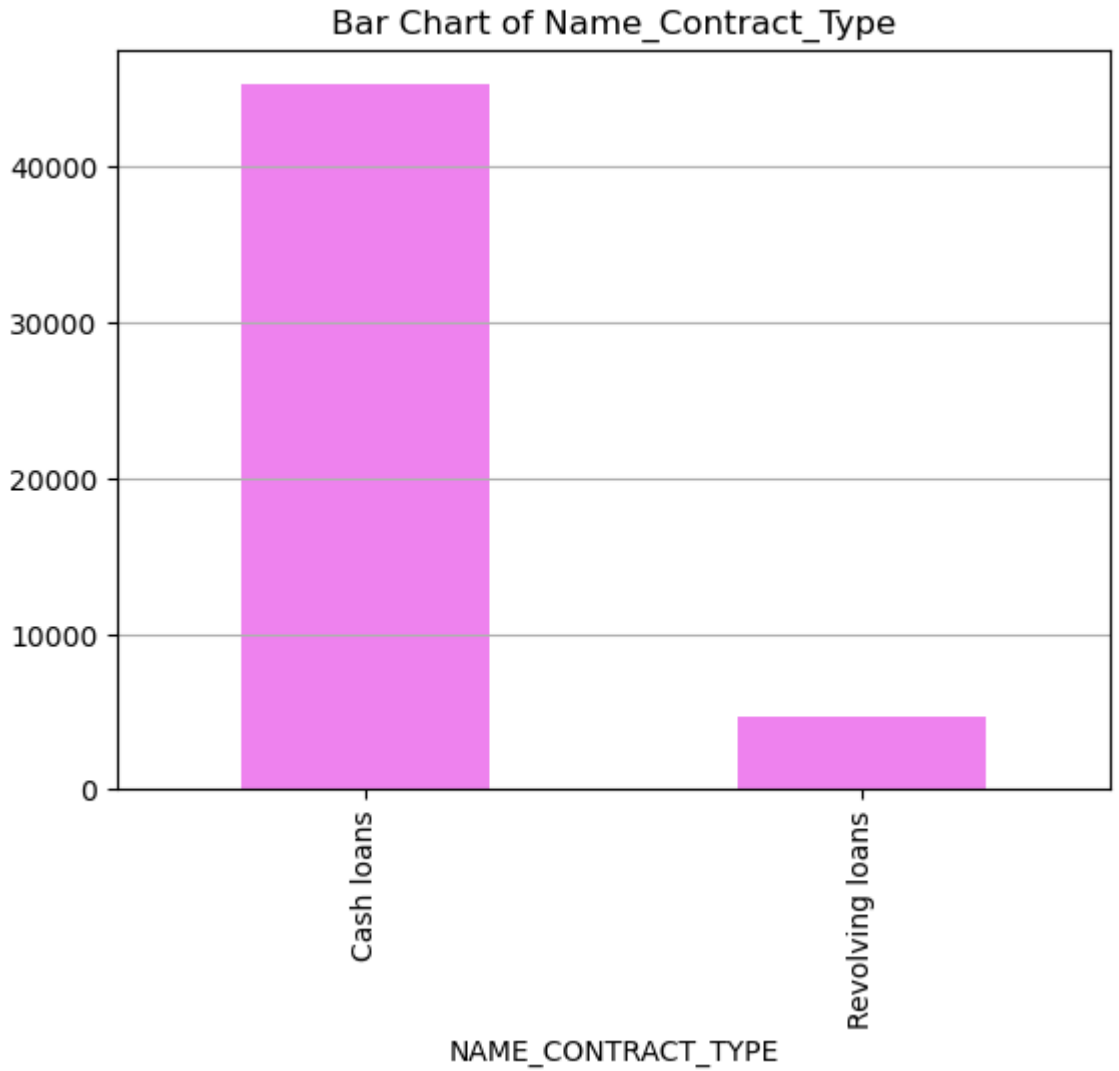
In [45]:
```
def use_pie_plot(Column, dataframe, figsize=(10, 8)):
    plt.figure(figsize=figsize)
    value_counts = dataframe[Column].value_counts()
    value_counts.plot.pie(autopct='%1.1f%%', startangle=90, labels=None)
    plt.title("Pie Chart of " + Column.title())
    plt.legend(labels=value_counts.index, loc='center left', bbox_to_anchor=(1.0, 0.5))
    plt.show()
```

NAME_CONTRACT_TYPE

In [46]:
```
new_app.NAME_CONTRACT_TYPE.value_counts(normalize=True)*100
```

Out[46]:
```
NAME_CONTRACT_TYPE
Cash loans         90.553811
Revolving loans     9.446189
Name: proportion, dtype: float64
```

In [47]:
```
use_bar_plot("NAME_CONTRACT_TYPE",new_app)
```



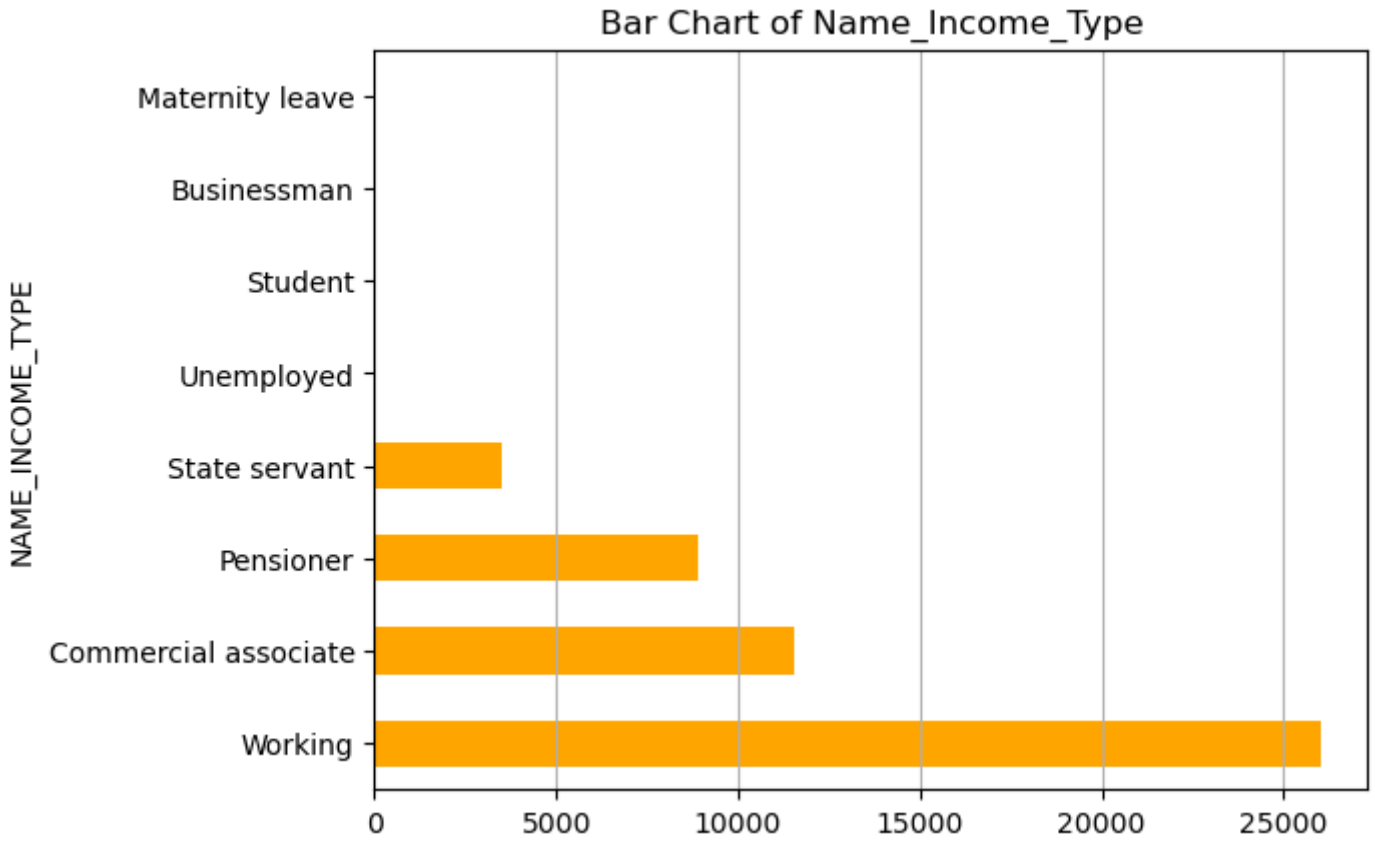
The Contract_type graph shows that Cash loans were taken by 90.55% of the people! and only 9.44% people took Revolving Loans

NAME_INCOME_TYPE

In [48]:
```
new_app.NAME_INCOME_TYPE.value_counts()
```

Out[48]:
```
NAME_INCOME_TYPE
Working                 26010
Commercial associate    11543
Pensioner                8920
State servant            3512
Unemployed                  6
Student                     5
Businessman                 2
Maternity leave             1
Name: count, dtype: int64
```

In [49]:
```
use_barh_plot("NAME_INCOME_TYPE",new_app)
```



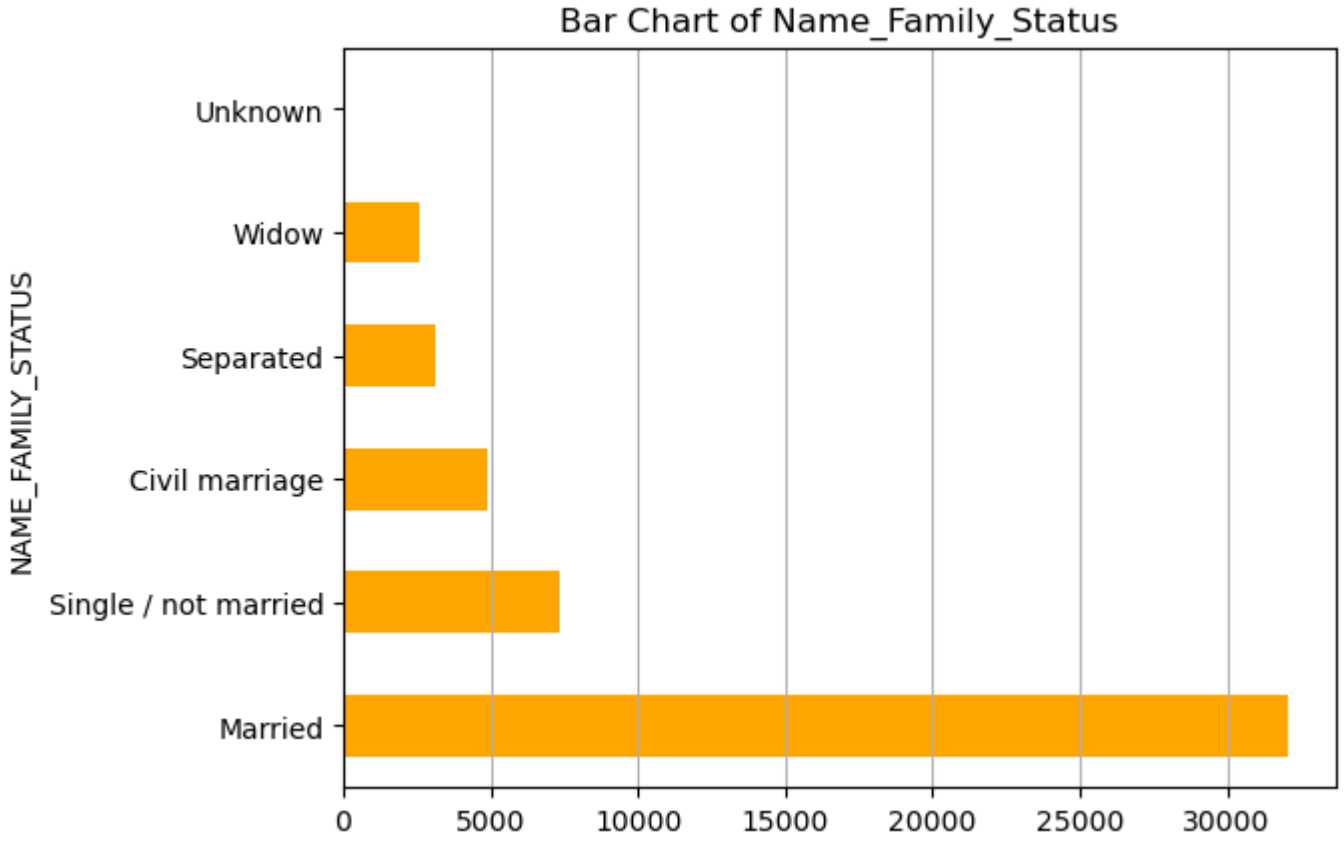
- Income_type graph shows clients income type. The majority of the clients who applied for loans were working class professionals with 26010 applications. Followed by Commercial associate 11543 applications, which is followed by pensioners and State servants.
- Unemployed, Students, Bussinessman, Maternity leave people were the least amount of people who applied for loan

NAME_FAMILY_STATUS

In [50]:
```
new_app.NAME_FAMILY_STATUS.value_counts(normalize=True)*100
```

Out[50]:
```
NAME_FAMILY_STATUS
Married                 64.189284
Single / not married    14.612292
Civil marriage           9.718194
Separated                6.284126
Widow                    5.194104
Unknown                  0.002000
Name: proportion, dtype: float64
```

In [51]:
```
use_barh_plot("NAME_FAMILY_STATUS",new_app)
```



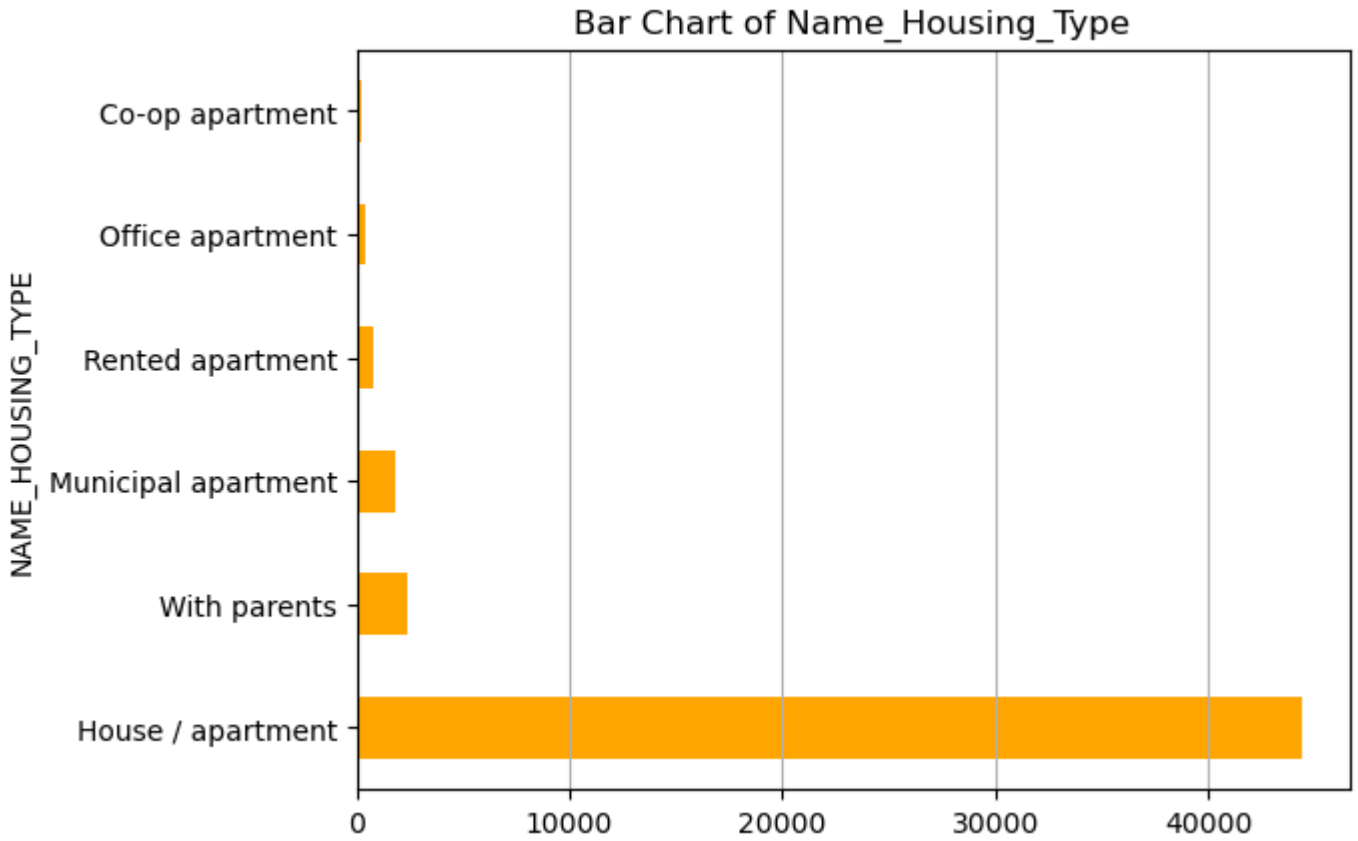
- Married people were the higher percentage of loan applicants with 64.18%. Followed by Single /Not Married people having 14.61%
- Civil marriage, Seperate, Widow were among the lower percentage category who applied for loans

NAME_HOUSING_TYPE

In [52]:
```
new_app.NAME_HOUSING_TYPE.value_counts(normalize=True)*100
```

Out[52]:
```
NAME_HOUSING_TYPE
House / apartment      88.737775
With parents            4.798096
Municipal apartment     3.690074
Rented apartment        1.538031
Office apartment        0.854017
Co-op apartment         0.382008
Name: proportion, dtype: float64
```

In [53]:
```
use_barh_plot("NAME_HOUSING_TYPE",new_app)
```



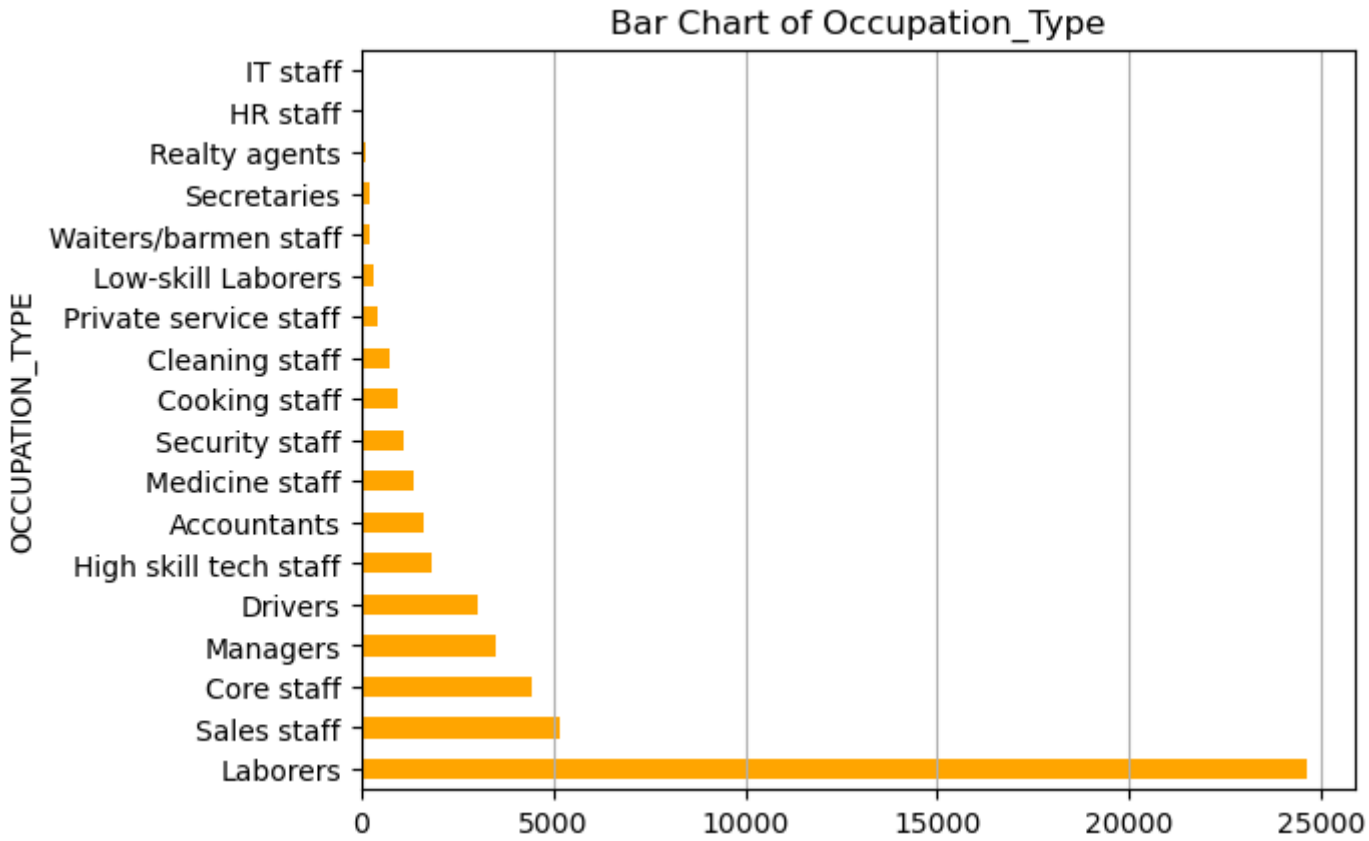
- 88.73% of the applicants were living in a House / Apartment
- 4.79% were living with their parents followed by 3.69% living in Muncipal Apartment
- Rented apartment accounted for 1.538031%, Office apartment accounted for 0.854017% and Co-op apartment accounted for 0.382008

OCCUPATION_TYPE

In [54]:
```
new_app.OCCUPATION_TYPE.value_counts(normalize=True)*100
```

Out[54]:
```
OCCUPATION_TYPE
Laborers                 49.212984
Sales staff              10.320206
Core staff                8.868177
Managers                  6.978140
Drivers                   6.088122
High skill tech staff     3.704074
Accountants               3.242065
Medicine staff            2.806056
Security staff            2.280046
Cooking staff             1.926039
Cleaning staff            1.478030
Private service staff     0.894018
Low-skill Laborers        0.714014
Waiters/barmen staff      0.456009
Secretaries               0.424008
Realty agents             0.246005
HR staff                  0.202004
IT staff                  0.160003
Name: proportion, dtype: float64
```

In [55]:
```
use_barh_plot("OCCUPATION_TYPE",new_app)
```



- We can see from the above percentage value that Laborers were the highest percentage of people who applied for a loan having the value of 49.21%, Followed by Sales staff (10.32%), Core staff (8.86%), Managers (6.97%) etc

ORGANIZATION_TYPE

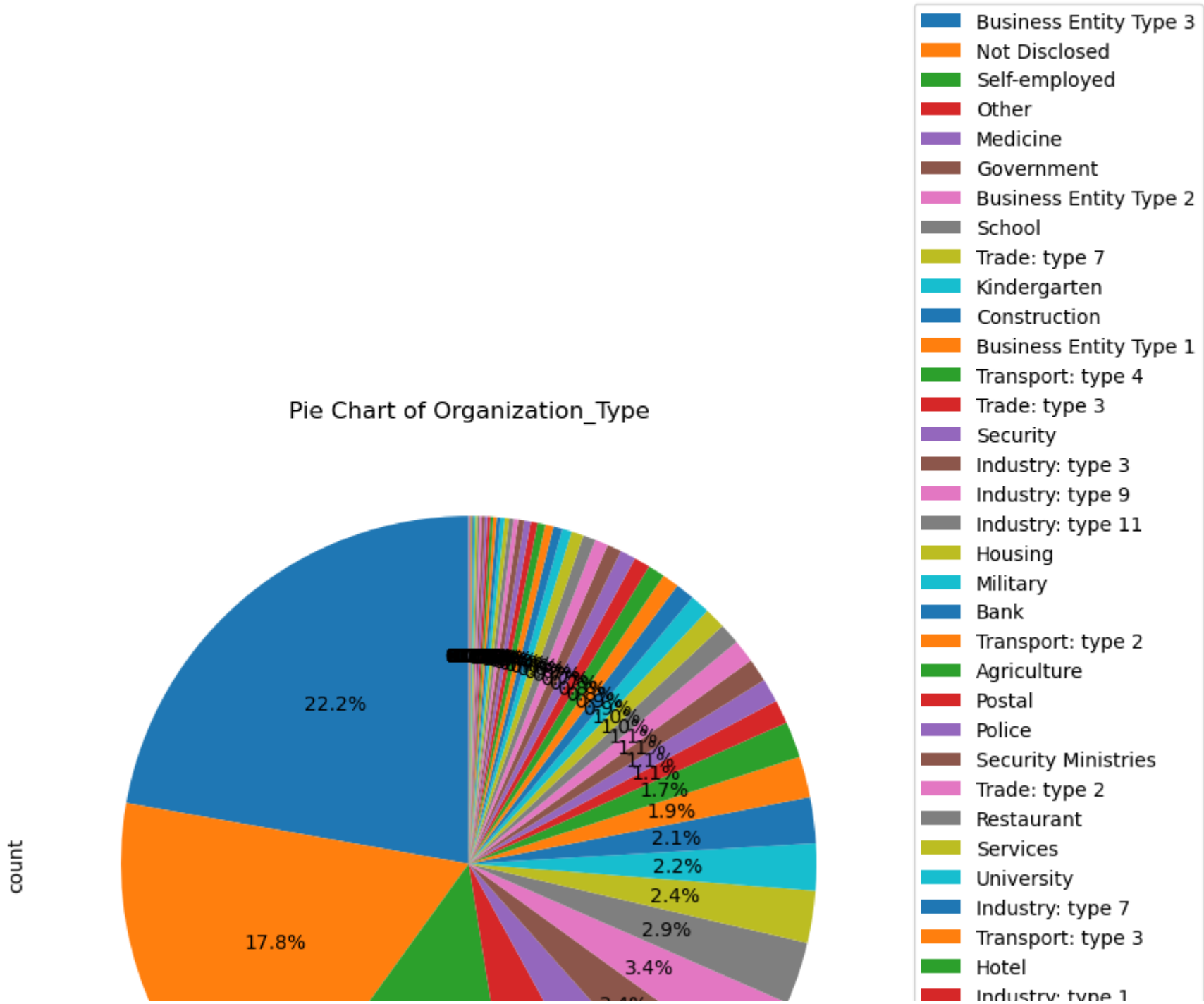
In [56]:
```
new_app.ORGANIZATION_TYPE.value_counts(normalize=True)*100
```

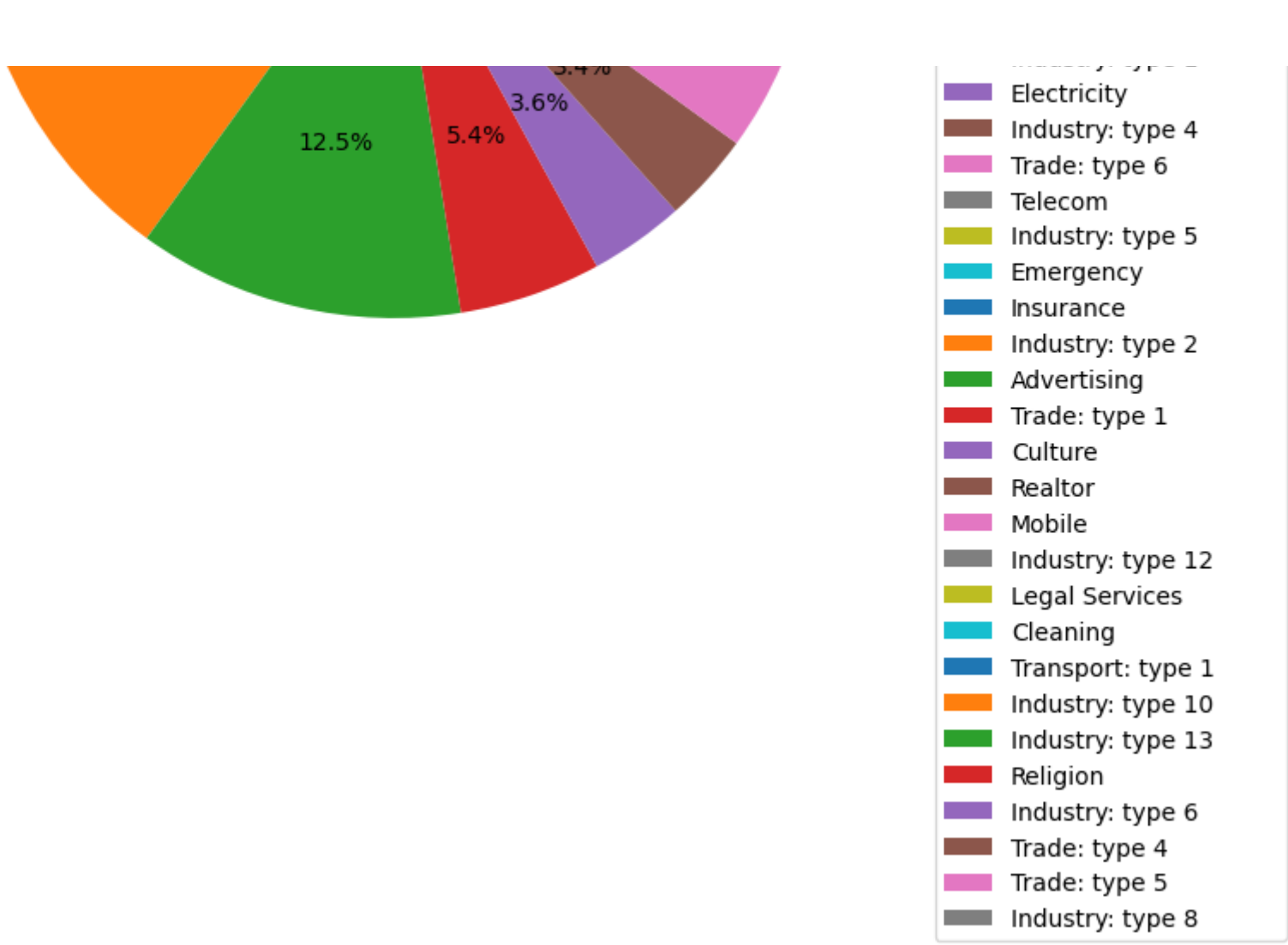
Out[56]:

ORGANIZATION_TYPE	
Business Entity Type 3	22.202444
Not Disclosed	17.848357
Self-employed	12.480250
Other	5.434109
Medicine	3.634073
Government	3.432069
Business Entity Type 2	3.408068
School	2.900058
Trade: type 7	2.420048
Kindergarten	2.180044
Construction	2.132043
Business Entity Type 1	1.906038
Transport: type 4	1.674033
Trade: type 3	1.100022
Security	1.100022
Industry: type 3	1.084022
Industry: type 9	1.074021
Industry: type 11	0.978020
Housing	0.978020
Military	0.916018
Bank	0.870017
Transport: type 2	0.784016
Agriculture	0.784016
Postal	0.740015
Police	0.732015
Security Ministries	0.662013
Trade: type 2	0.614012
Restaurant	0.578012
Services	0.568011
University	0.444009
Industry: type 7	0.418008
Transport: type 3	0.382008
Hotel	0.364007
Industry: type 1	0.318006
Electricity	0.294006
Industry: type 4	0.280006
Trade: type 6	0.216004
Telecom	0.212004
Industry: type 5	0.206004
Emergency	0.186004

```
Insurance                 0.178004
Industry: type 2          0.156003
Advertising               0.136003
Trade: type 1             0.132003
Culture                   0.128003
Realtor                   0.122002
Mobile                    0.112002
Industry: type 12         0.106002
Legal Services            0.088002
Cleaning                  0.080002
Transport: type 1         0.056001
Industry: type 10         0.042001
Industry: type 13         0.030001
Religion                  0.028001
Industry: type 6          0.024000
Trade: type 4             0.016000
Trade: type 5             0.016000
Industry: type 8          0.016000
Name: proportion, dtype: float64
```

In [61]:
```
use_pie_plot("ORGANIZATION_TYPE",new_app)
```


Pie Chart of Organization_Type
count
22.2%
17.8%
1.7%
1.9%
2.1%
2.2%
2.4%
2.9%
3.4%
Business Entity Type 3
Not Disclosed
Self-employed
Other
Medicine
Government
Business Entity Type 2
School
Trade: type 7
Kindergarten
Construction
Business Entity Type 1
Transport: type 4
Trade: type 3
Security
Industry: type 3
Industry: type 9
Industry: type 11
Housing
Military
Bank
Transport: type 2
Agriculture
Postal
Police
Security Ministries
Trade: type 2
Restaurant
Services
University
Industry: type 7
Transport: type 3
Hotel



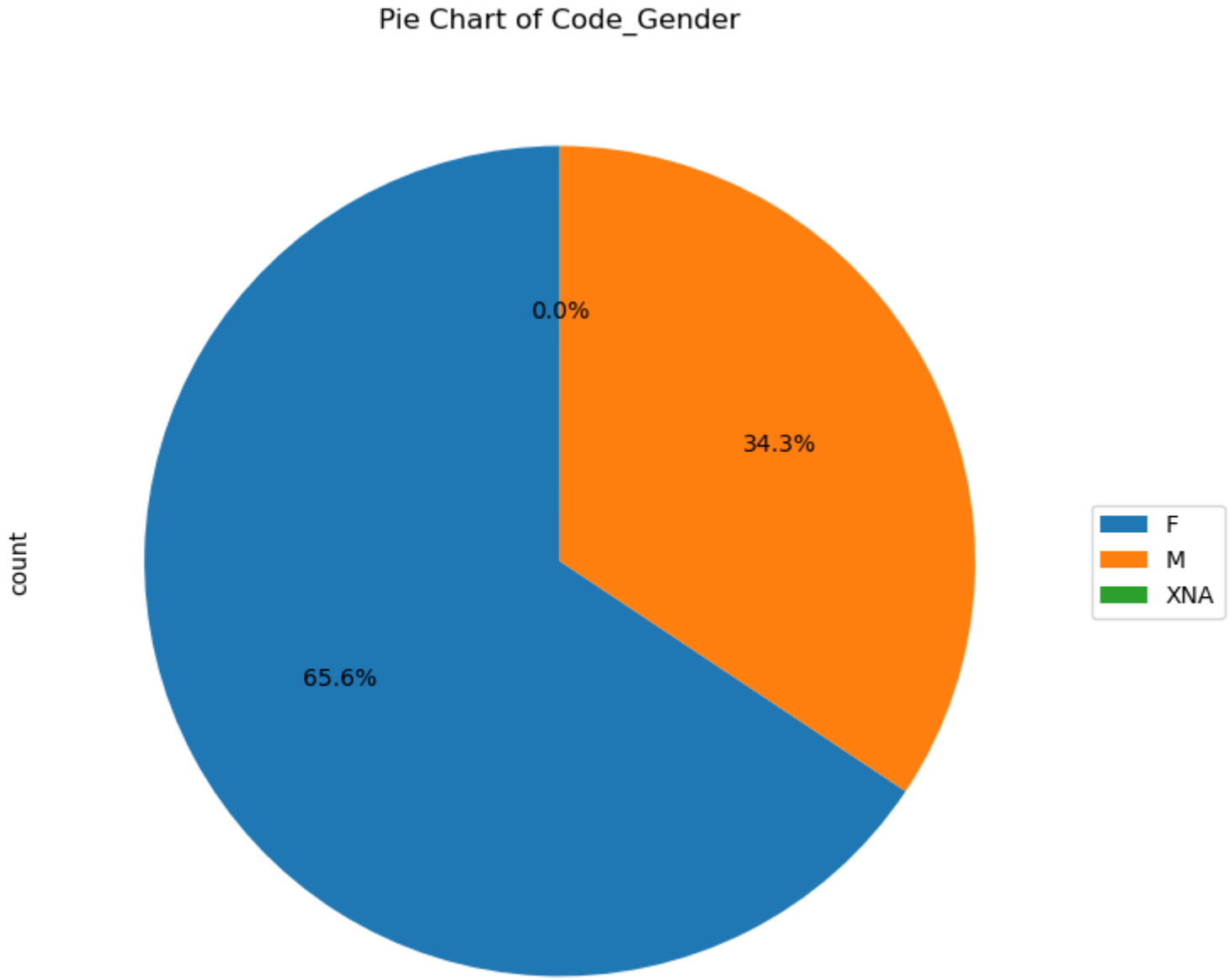
- Business Entity Type3, Self employed, Medicine, Government were the major organizations the loan applicants
- However 17.84% people have not disclosed their Organizations

CODE_GENDER

In [62]:
```
new_app.CODE_GENDER.value_counts(normalize=True)*100
```

Out[62]:
```
CODE_GENDER
F      65.647313
M      34.348687
XNA     0.004000
Name: proportion, dtype: float64
```

In [63]:
```
use_pie_plot("CODE_GENDER",new_app)
```



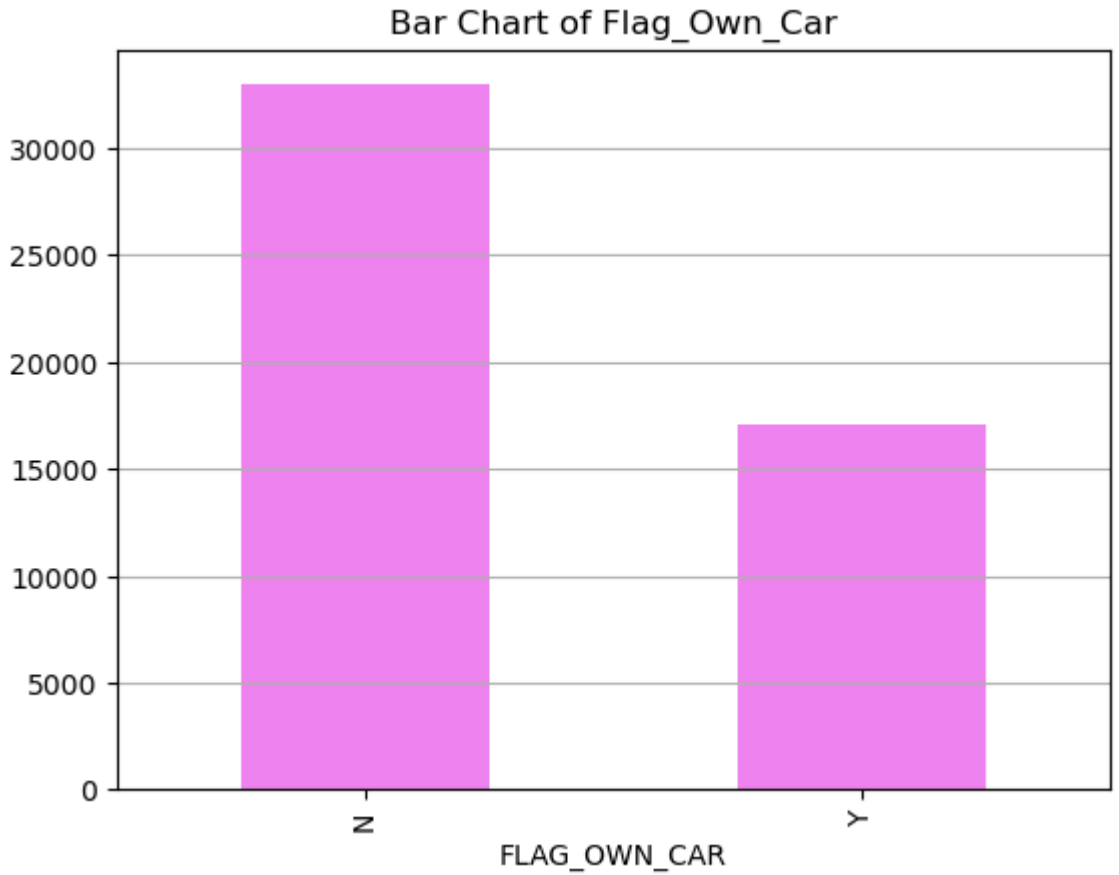
- 65.64% of Females have applied for loan while remaining 34.34% being males

FLAG_OWN_CAR

In [64]:
```
new_app.FLAG_OWN_CAR.value_counts(normalize=True)*100
```

Out[64]:
```
FLAG_OWN_CAR
N    65.899318
Y    34.100682
Name: proportion, dtype: float64
```

In [65]:
```
use_bar_plot("FLAG_OWN_CAR",new_app)
```



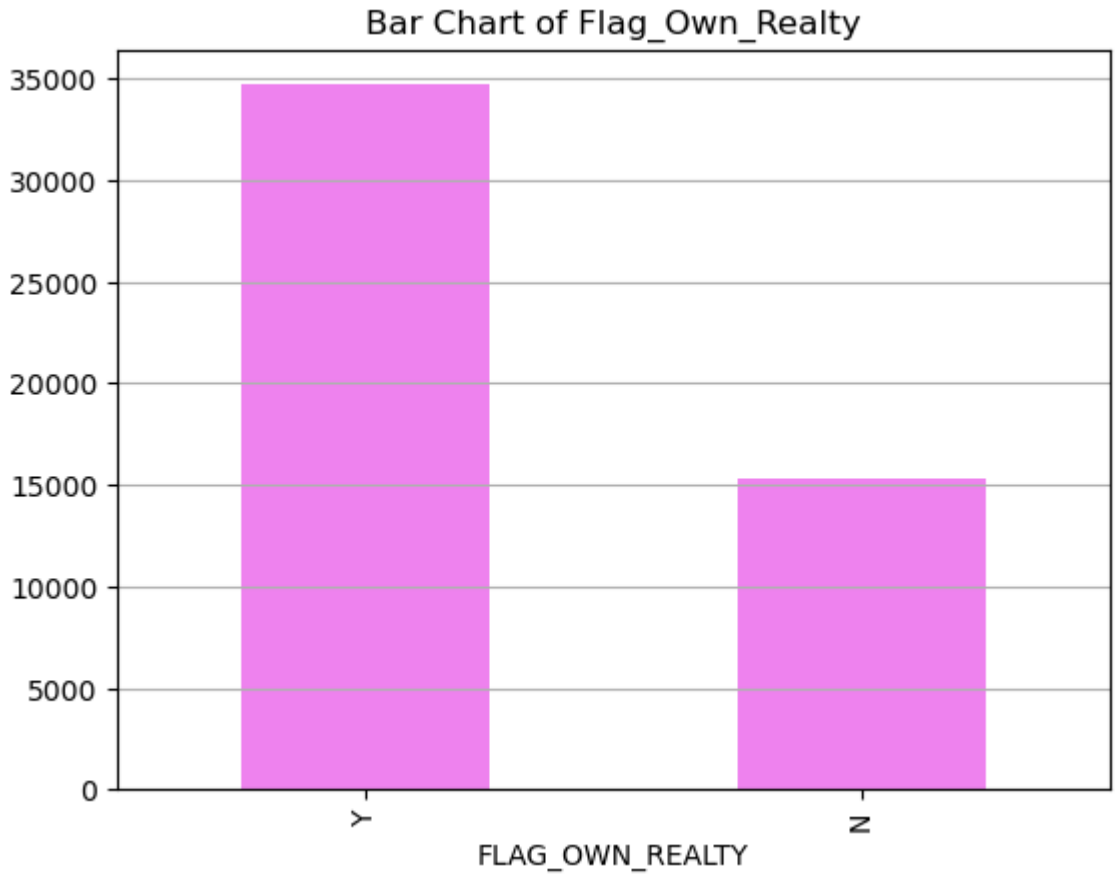
- 65.89% of the people do not own a car and they were the majority who applied for loans
- 34.10% of the people who applied owns a car

FLAG_OWN_REALTY

In [66]:
```
new_app.FLAG_OWN_REALTY.value_counts(normalize=True)*100
```

Out[66]:
```
FLAG_OWN_REALTY
Y    69.383388
N    30.616612
Name: proportion, dtype: float64
```

In [67]:
```
use_bar_plot("FLAG_OWN_REALTY",new_app)
```



- 69.38% people own a Realty(House or flat)
- While 30.61% people dont have own any realty

Segmented Univariate

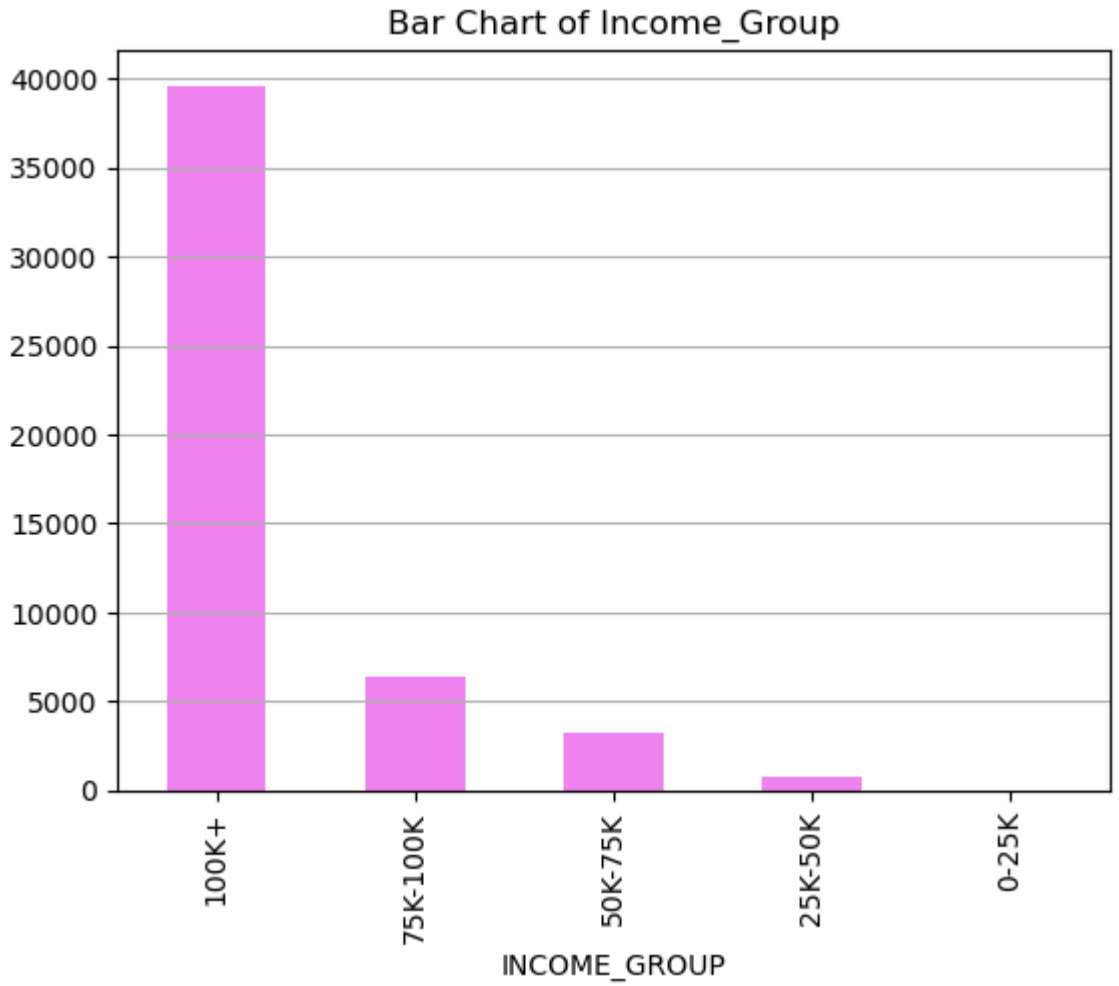
In [68]:
```
new_app.INCOME_GROUP.value_counts(normalize=True)*100
```

Out[68]:
```
INCOME_GROUP
100K+       79.215584
75K-100K    12.724254
50K-75K      6.452129
25K-50K      1.608032
0-25K        0.000000
Name: proportion, dtype: float64
```

In [69]:
```
new_app.INCOME_GROUP.value_counts()
```

Out[69]:
```
INCOME_GROUP
100K+       39607
75K-100K     6362
50K-75K      3226
25K-50K       804
0-25K           0
Name: count, dtype: int64
```

In [70]:
```
use_bar_plot("INCOME_GROUP",new_app)
```

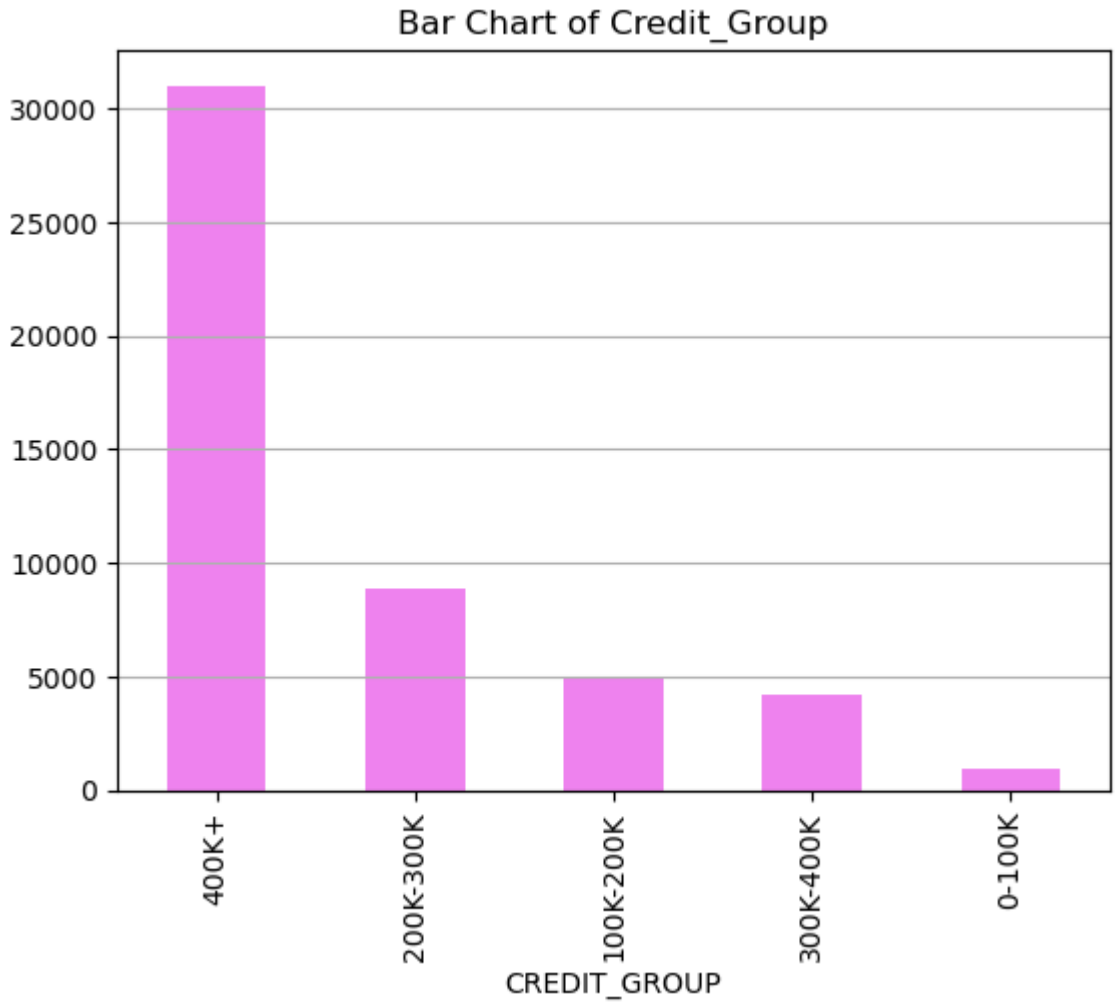


- Majority of the loan appliers were from 100k+ (Very High) wage Income being 79.21% total
- followed by 75k-100k (High class) income being 26.73% & 50k-75k (Medium Class) income which is 6.45%
- Low Class income wage people were 1.60% applied for loan
- It shows loan applied by Very High & High wage people can repay the loan

In [71]:
```
new_app.CREDIT_GROUP.value_counts(normalize=True)*100
```

Out[71]:
```
CREDIT_GROUP
400K+        61.989240
200K-300K    17.698354
100K-200K     9.822196
300K-400K     8.512170
0-100K        1.978040
Name: proportion, dtype: float64
```

In [72]:
```
use_bar_plot("CREDIT_GROUP",new_app)
```

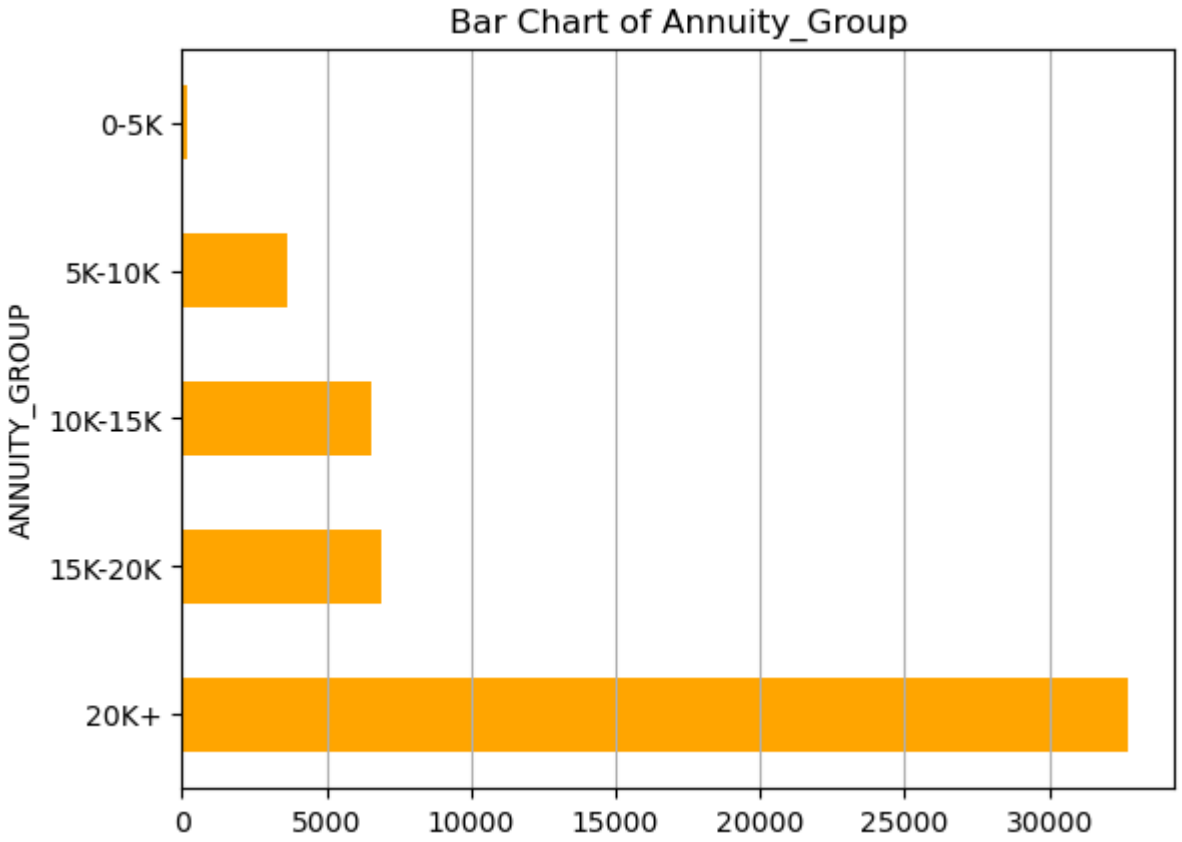


- Mainly the credit amount of loan ranged from 400k+ followed by 200k-300k and 100k-200k

In [73]:
```
new_app.ANNUITY_GROUP.value_counts(normalize=True)*100
```

Out[73]:
```
ANNUITY_GROUP
20K+       65.417308
15K-20K    13.790276
10K-15K    13.128263
5K-10K      7.270145
0-5K        0.394008
Name: proportion, dtype: float64
```

In [74]:
```
use_barh_plot("ANNUITY_GROUP",new_app)
```



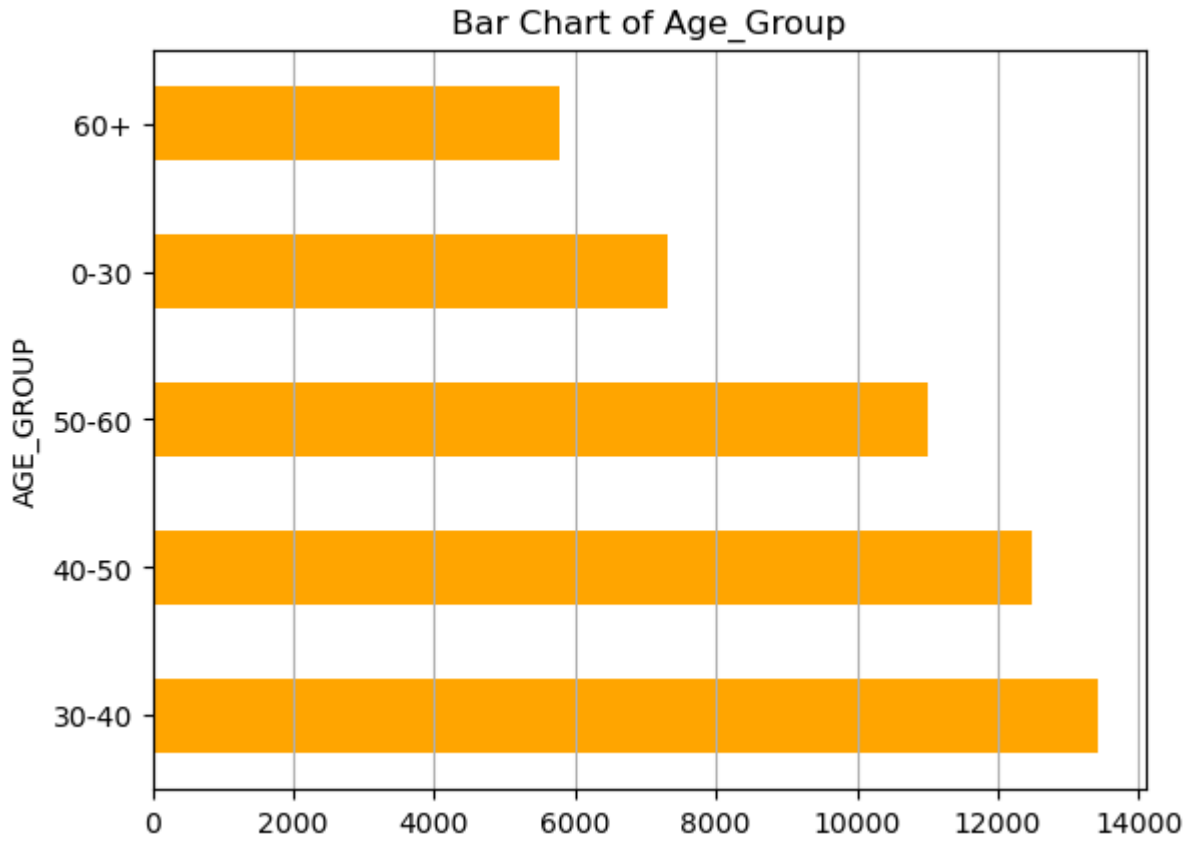
- Applicants took Annuities ranging in 20K which topped the graph with 65.41%

- Followed by 15k-20k (13.79%), 10k-15k (13.12%) etc.

In [75]:
```
new_app.AGE_GROUP.value_counts(normalize=True)*100
```

Out[75]:
```
AGE_GROUP
30-40    26.844537
40-50    24.978500
50-60    22.040441
0-30     14.604292
60+      11.532231
Name: proportion, dtype: float64
```

In [76]:
```
use_barh_plot("AGE_GROUP",new_app)
```

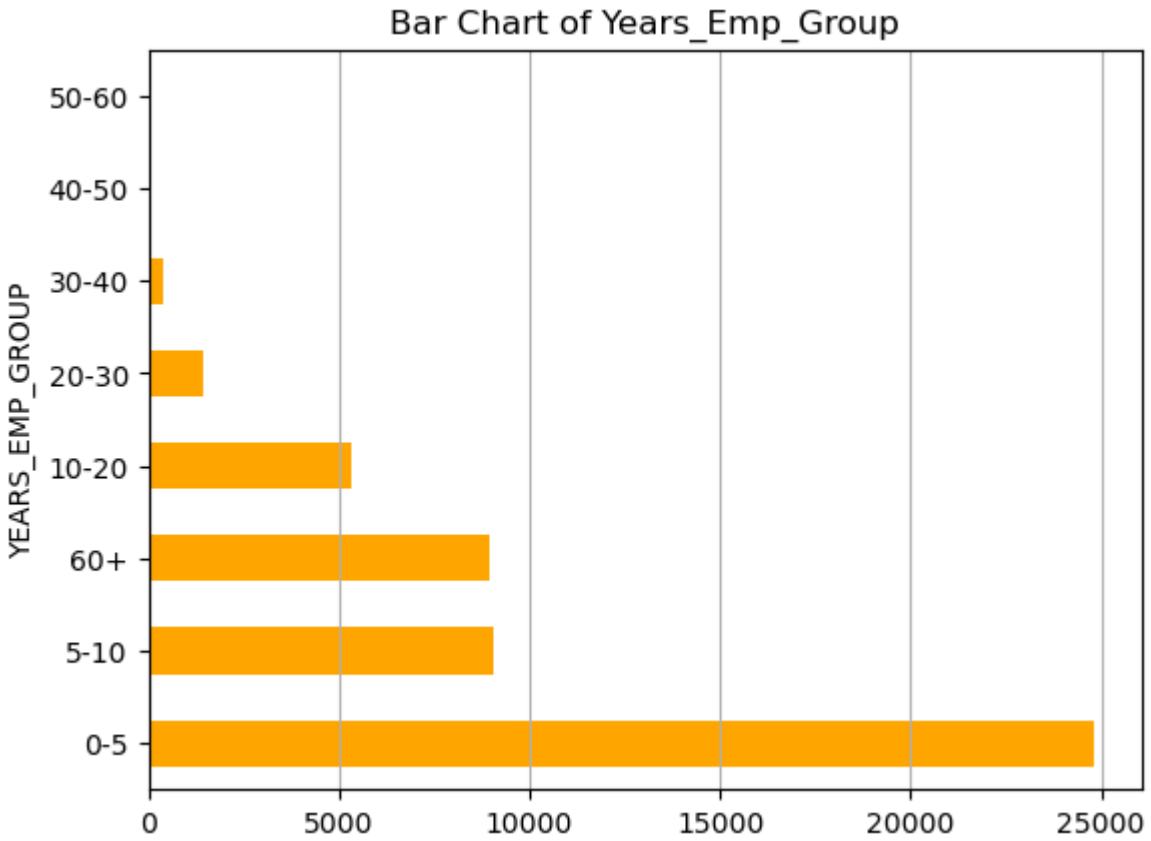


- Majority of the loans applied by younger age people ranging in 30-40 age group which is 26.84%
- followed by mid senior people from 40-50 age group & 50-60 senior age group

In [77]:
```
new_app.YEARS_EMP_GROUP.value_counts(normalize=True)*100
```

Out[77]:
```
YEARS_EMP_GROUP
0-5      49.634993
5-10     18.158363
60+      17.848357
10-20    10.618212
20-30     2.888058
30-40     0.804016
40-50     0.048001
50-60     0.000000
Name: proportion, dtype: float64
```

In [78]:
```
use_barh_plot("YEARS_EMP_GROUP",new_app)
```



- 0-5 years of experience people topped the chart with 49.63%
- with none being from 50-60 years , and some being from 40-50 years

Bivariate Analysis

To do Bivariate analysis we will devide the dataset into two parts i.e.

- Client with payment difficulties (Target=1)

- All other cases (Target=0)

In [158]:
```
# Client with payment difficulties (Target=1)
new_app_target_1 = new_app[new_app['TARGET'] == 1]
new_app_target_1.head()
```

Out[158]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOT
0	100002	1	Cash loans	M	N	Y	0	20250
26	100031	1	Cash loans	F	N	Y	0	11250
40	100047	1	Cash loans	M	N	Y	0	20250
42	100049	1	Cash loans	F	N	N	0	13500
81	100096	1	Cash loans	F	N	Y	0	8100

5 rows × 27 columns

In [159]:
```
# Client with all other cases (Target=0)
new_app_target_0 = new_app[new_app['TARGET'] == 0]
new_app_target_0.head()
```

Out[159]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTA
1	100003	0	Cash loans	F	N	N	0	270000
2	100004	0	Revolving loans	M	Y	Y	0	67500
3	100006	0	Cash loans	F	N	Y	0	135000
4	100007	0	Cash loans	M	N	Y	0	121500
5	100008	0	Cash loans	M	N	Y	0	99000

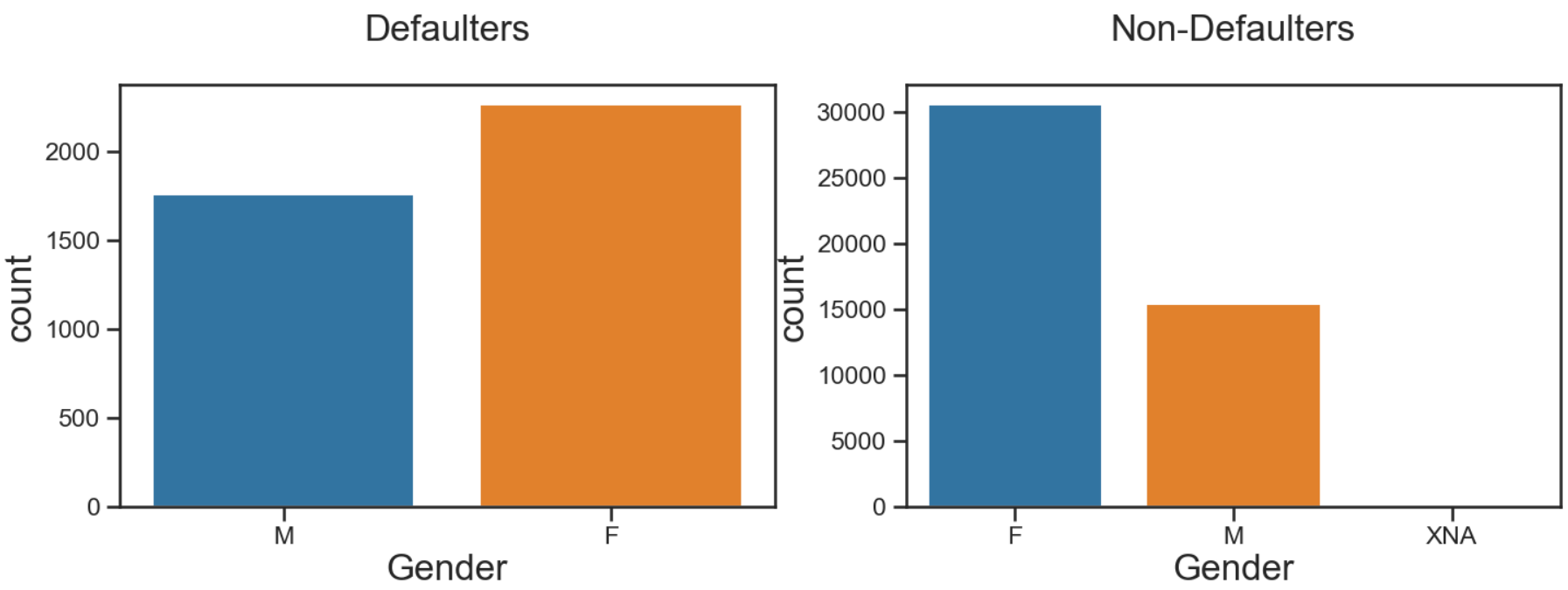
5 rows × 27 columns

Analysis on Categorical variables

Count of defaulters and non-defaulters on the basis of gender

In [163]:
```
# Plotting two plots for delaulters and non defaulters on basis of gender
plt.figure(figsize=(17,5))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'CODE_GENDER',data=new_app_target_1)
plt.title('Defaulters')
ax.set(xlabel='Gender')

plt.subplot(1,2,2)
ax = sns.countplot(x = 'CODE_GENDER',data=new_app_target_0)
plt.title('Non-Defaulters')
ax.set(xlabel='Gender')
```

Out[163]:
```
[Text(0.5, 0, 'Gender')]
```

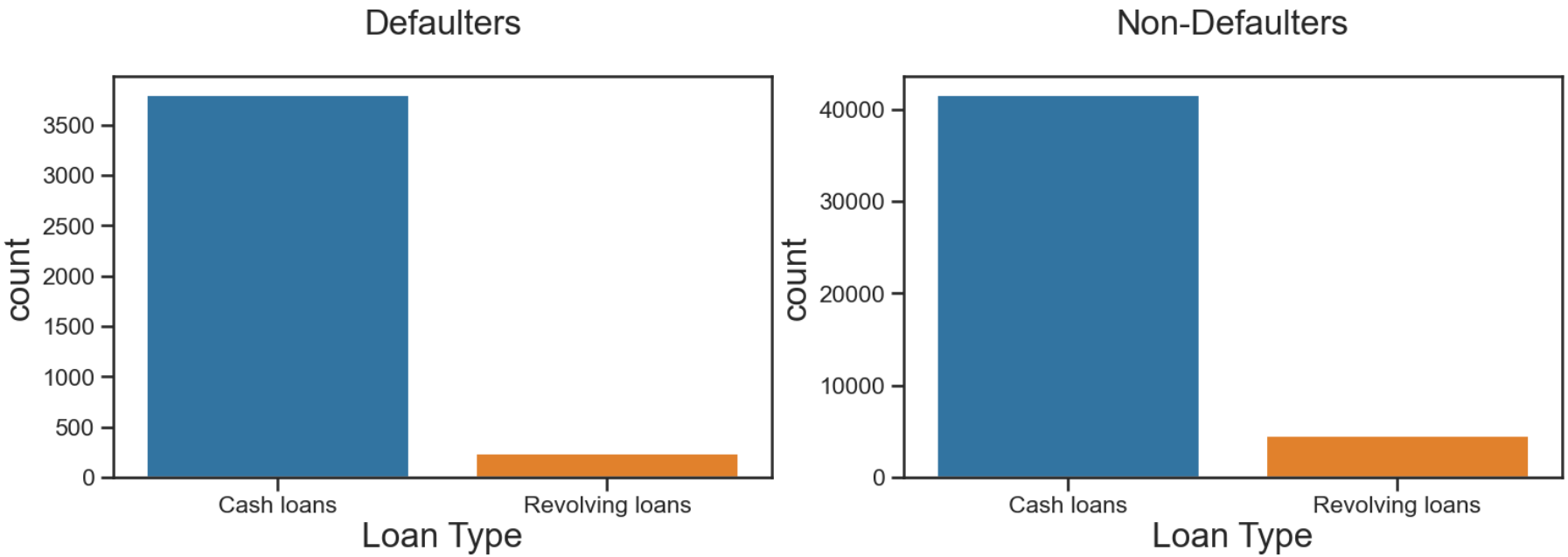


Defaulter Analysis: We observe a slight gender imbalance among defaulters, with a slightly higher number of females compared to males. This suggests that there may be certain factors or variables that contribute to a higher likelihood of defaulting for females.

Non-Defaulter Analysis: Similarly, among non-defaulters, we continue to observe a higher representation of females compared to males. This finding indicates that gender may not be a significant differentiating factor in determining loan repayment behavior, as both genders have a similar

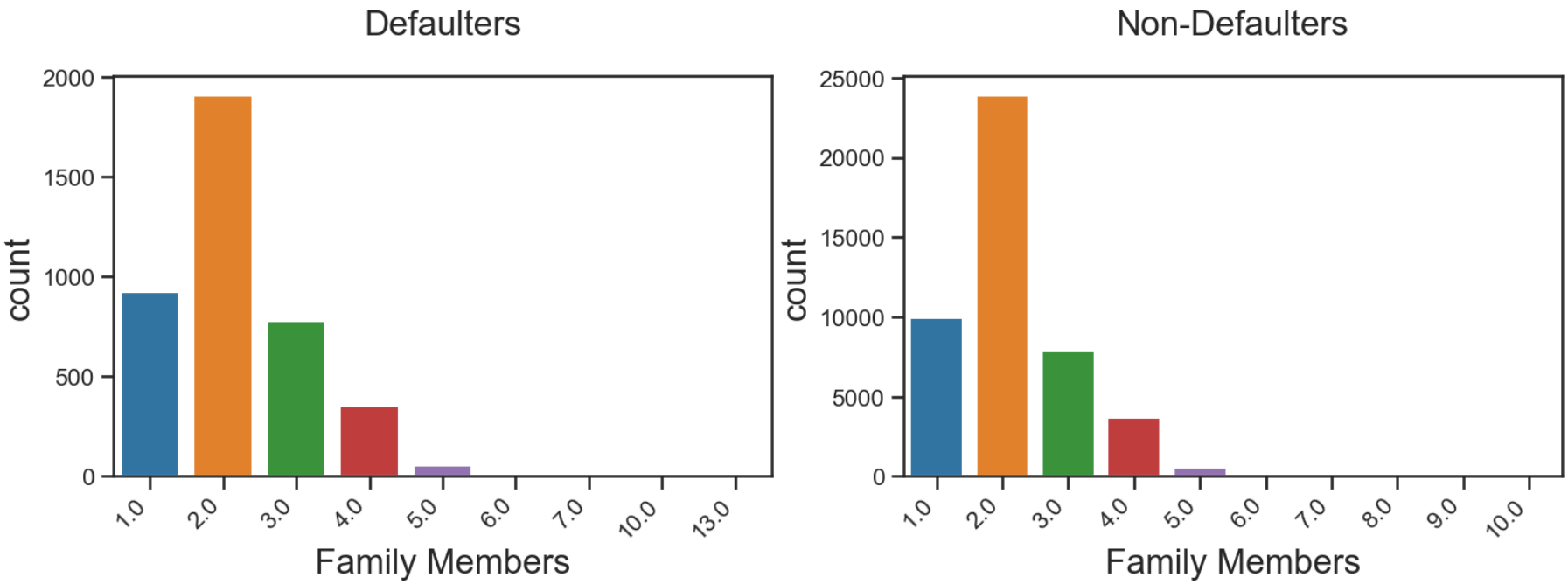
In [165]:
```
# Plotting two plots for delaulters and non defaulters on basis of Contract Type
plt.figure(figsize=(18,5))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'NAME_CONTRACT_TYPE',data=new_app_target_1)
plt.title('Defaulters')
ax.set(xlabel='Loan Type')

plt.subplot(1,2,2)
ax = sns.countplot(x = 'NAME_CONTRACT_TYPE',data=new_app_target_0)
plt.title('Non-Defaulters')
ax.set(xlabel='Loan Type')
```

Out[165]:
```
[Text(0.5, 0, 'Loan Type')]
```

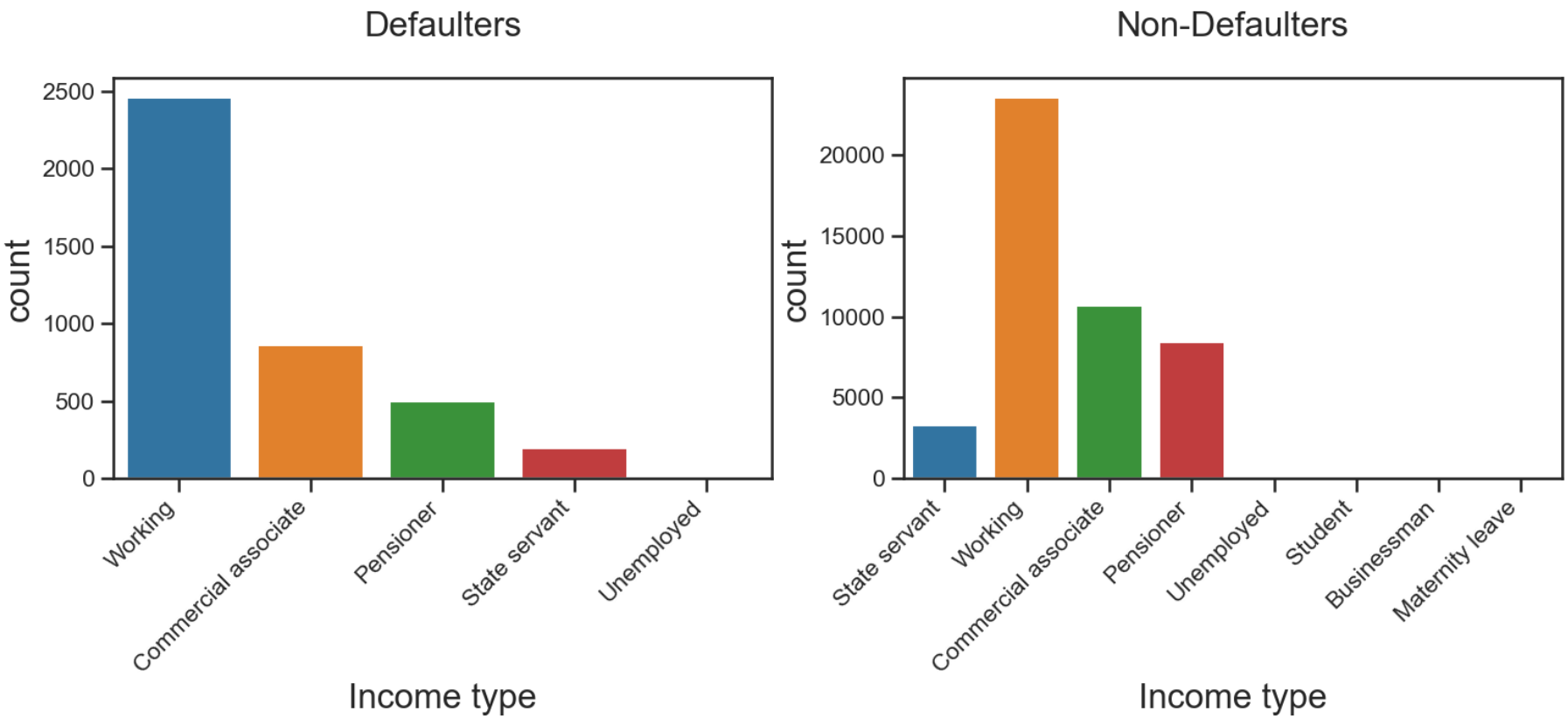


In both the cases of defaulters and non-defaulters, we observe a significant difference in the number of Revolving loans compared to Cash loans. Revolving loans are considerably less prevalent among both defaulters and non-defaulters. This indicates that the majority of loan applications, regardless of the repayment status, are for Cash loans rather than Revolving loans.

In [178]:
```
# Plotting two plots for delaulters and non defaulters on basis of Income Type
plt.figure(figsize=(18,5))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'CNT_FAM_MEMBERS',data=new_app_target_1)
plt.title('Defaulters')
ax.set(xlabel='Family Members')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')


plt.subplot(1,2,2)
ax = sns.countplot(x = 'CNT_FAM_MEMBERS',data=new_app_target_0)
plt.title('Non-Defaulters')
ax.set(xlabel='Family Members')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')
```



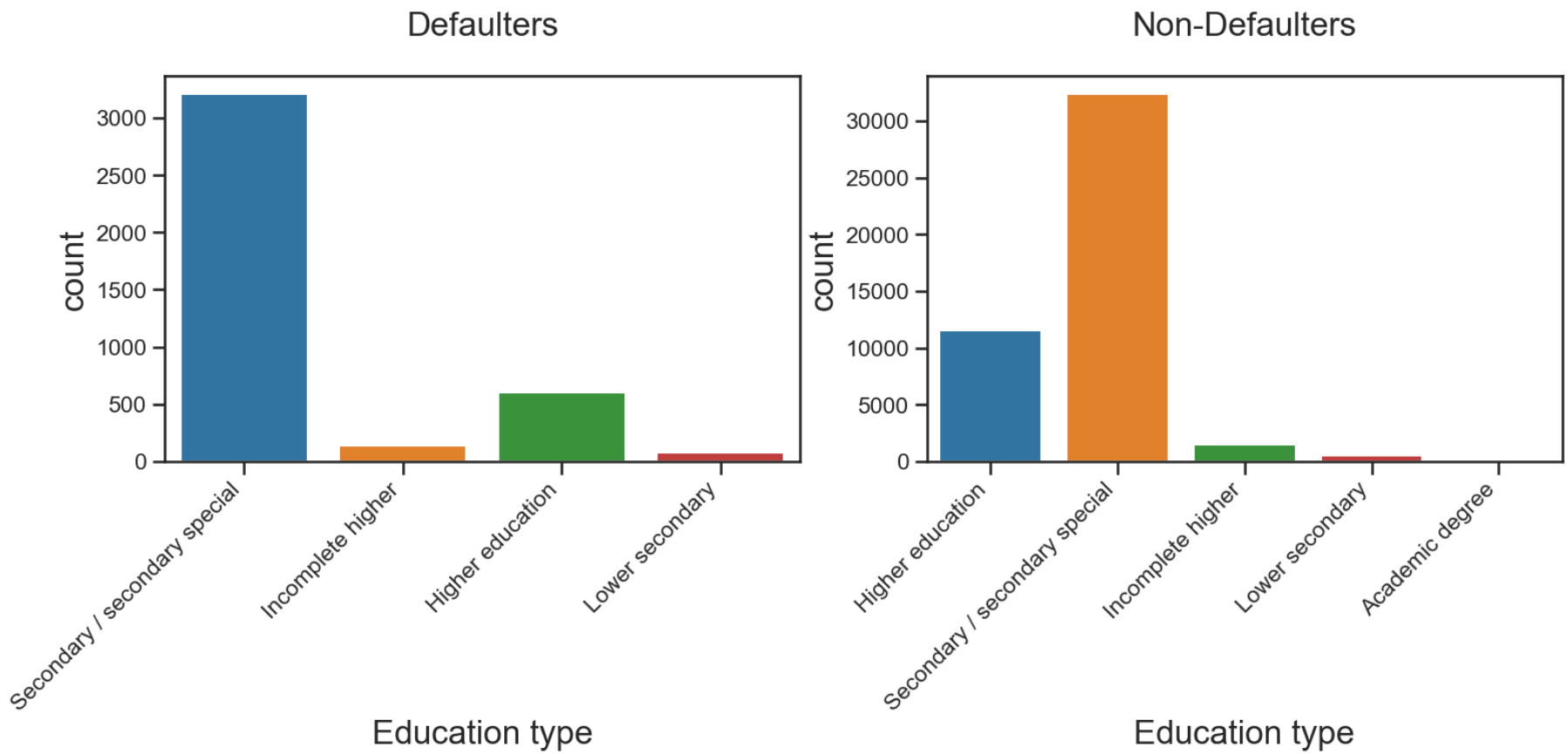
In [182]:
```
# Plotting two plots for delaulters and non defaulters on basis of Income Type
plt.figure(figsize=(18,5))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'NAME_INCOME_TYPE',data=new_app_target_1)
plt.title('Defaulters')
ax.set(xlabel='Income type')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')


plt.subplot(1,2,2)
ax = sns.countplot(x = 'NAME_INCOME_TYPE',data=new_app_target_0)
plt.title('Non-Defaulters')
ax.set(xlabel='Income type')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')
```



Defaulters: Among the defaulters, we observe that individuals belonging to the "Working" profession have the highest number. This suggests that working individuals are more prone to defaulting on their loans compared to individuals in other professions.

Non-defaulters: Similarly, among the non-defaulters, the majority are individuals from the "Working" profession. This indicates that individuals in the working category are more likely to meet their loan repayment obligations and are considered reliable borrowers.

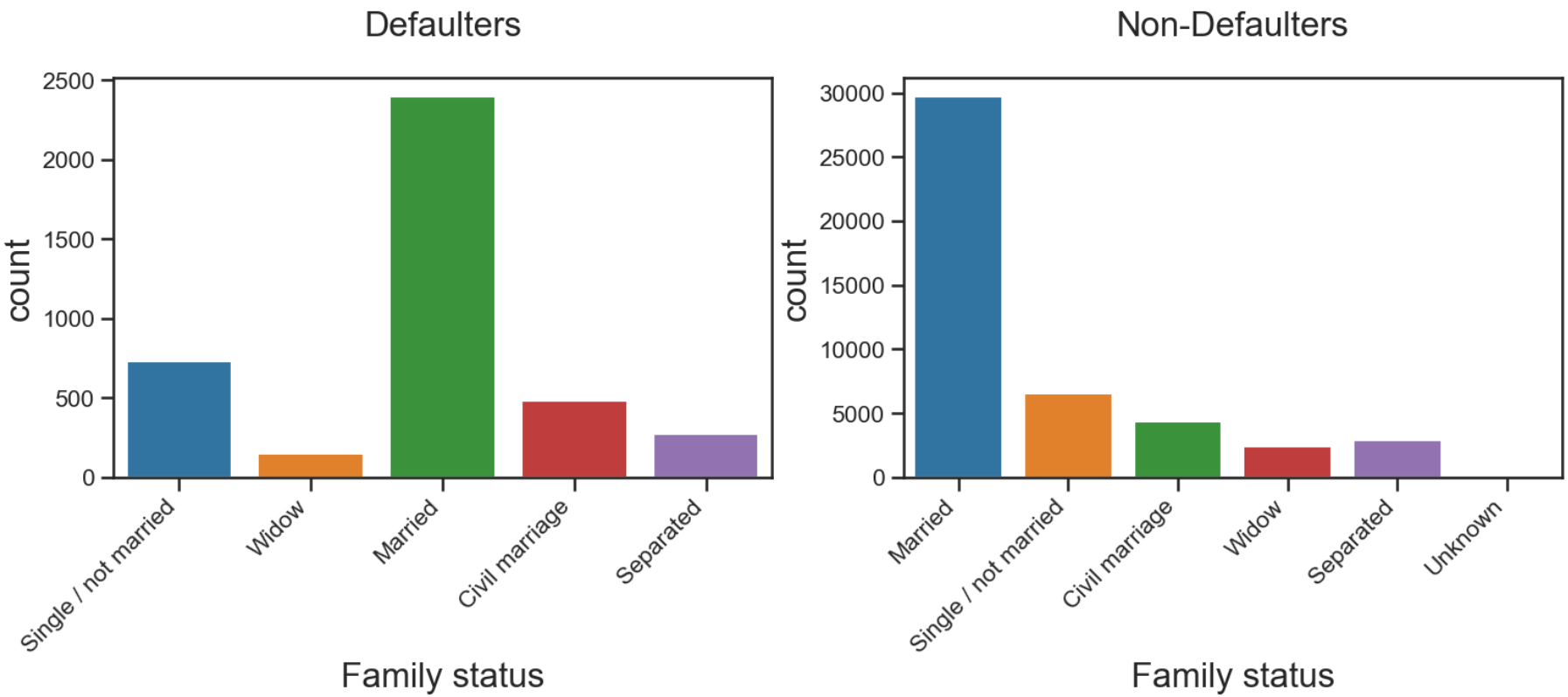
In [167]:
```
# Plotting two plots for delaulters and non defaulters on basis of Education Type
plt.figure(figsize=(18,5))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'NAME_EDUCATION_TYPE',data=new_app_target_1)
plt.title('Defaulters')
ax.set(xlabel='Education type')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')


plt.subplot(1,2,2)
ax = sns.countplot(x = 'NAME_EDUCATION_TYPE',data=new_app_target_0)
plt.title('Non-Defaulters')
ax.set(xlabel='Education type')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')
```



Defaulters: Among the defaulters, individuals with a secondary or secondary special level of education have the highest representation. This indicates that individuals with lower levels of education are more likely to default on their loans compared to those with higher levels of education.

Non-defaulters: Similarly, among the non-defaulters, the majority are individuals with a secondary or secondary special level of education. This suggests that individuals with lower levels of education are also more likely to fulfill their loan repayment obligations and demonstrate responsible borrowing behavior.

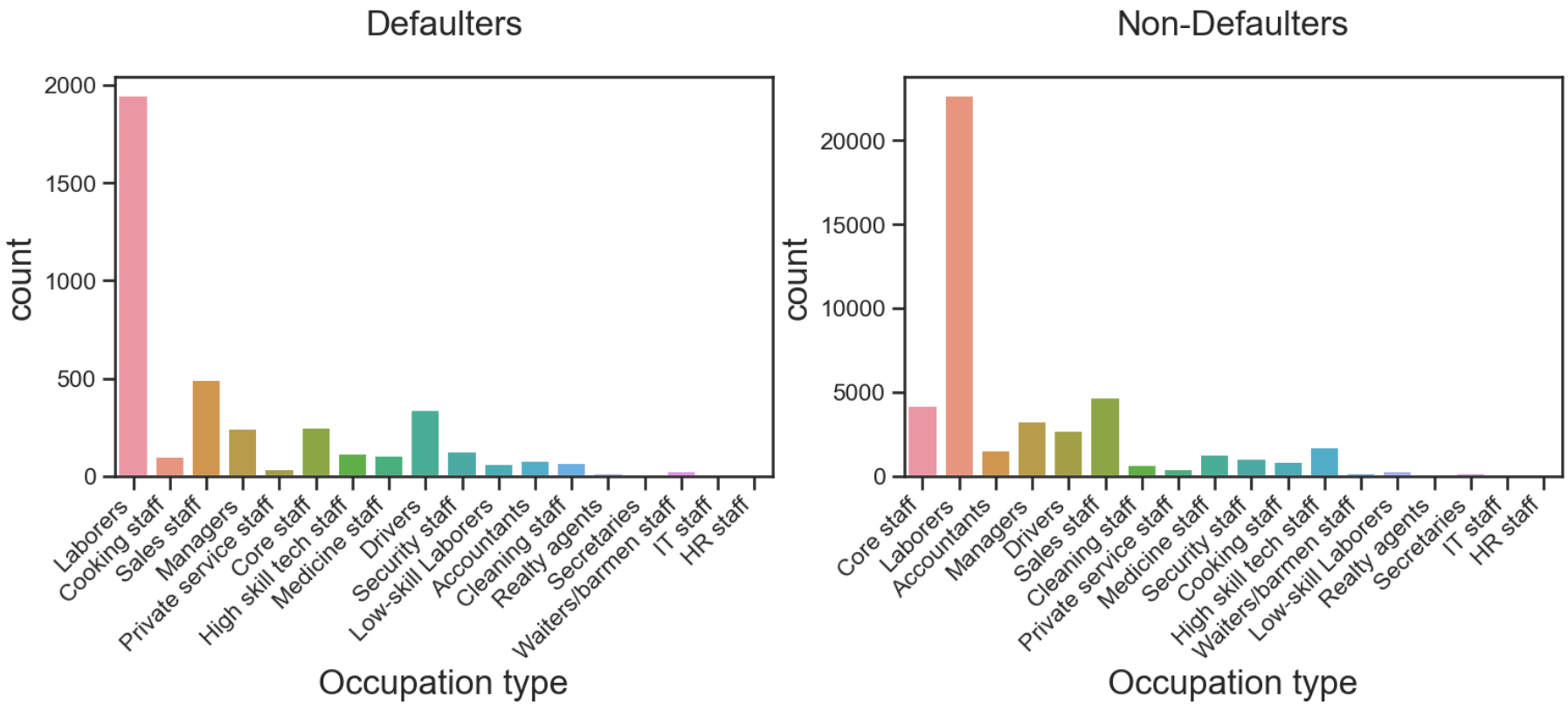
In [168]:
```
# Plotting two plots for delaulters and non defaulters on basis of Family Status
plt.figure(figsize=(18,5))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'NAME_FAMILY_STATUS',data=new_app_target_1)
plt.title('Defaulters')
ax.set(xlabel='Family status')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')


plt.subplot(1,2,2)
ax = sns.countplot(x = 'NAME_FAMILY_STATUS',data=new_app_target_0)
plt.title('Non-Defaulters')
ax.set(xlabel='Family status')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')
```



Defaulters: Among the defaulters, the majority of individuals are married. This suggests that married individuals may have higher financial obligations or face more financial stress, leading to a higher likelihood of defaulting on their loans compared to individuals in other marital statuses.

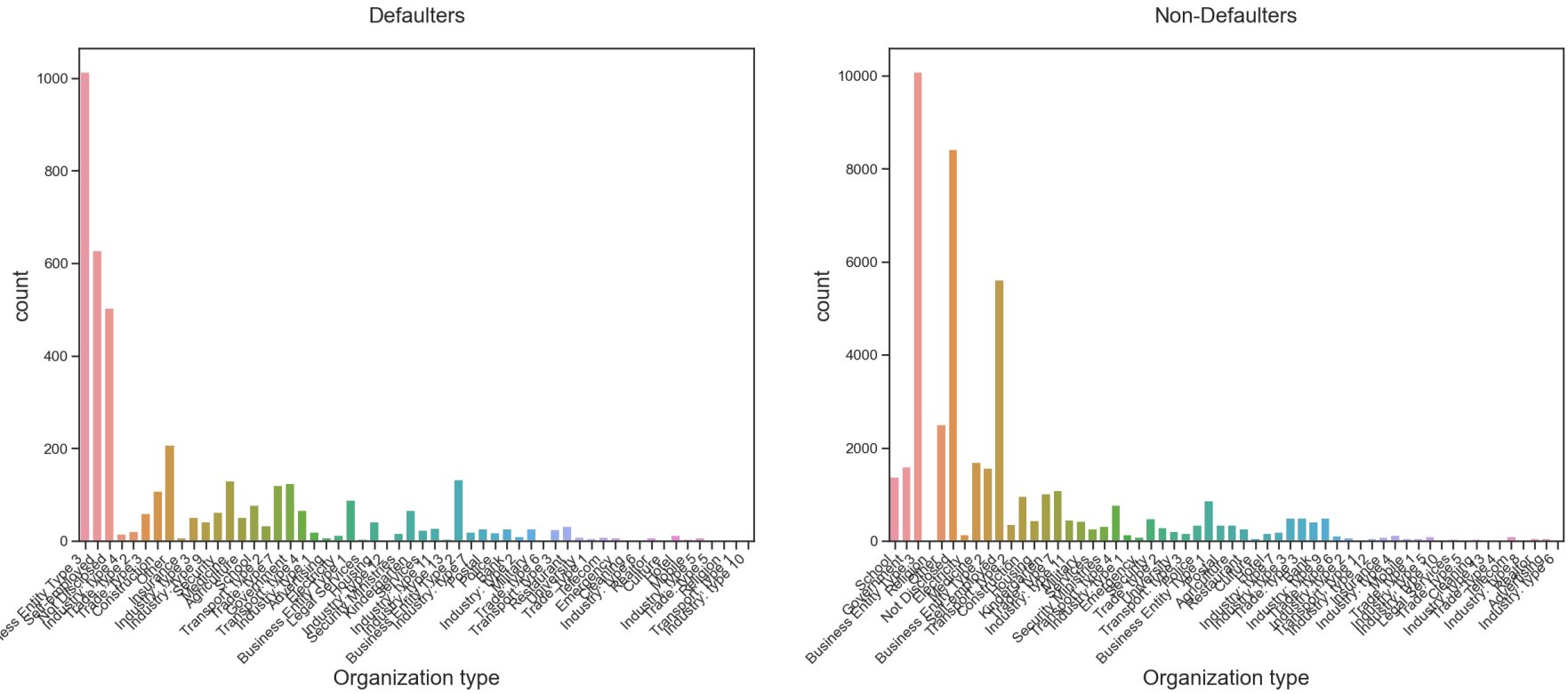
Non-defaulters: Similarly, among the non-defaulters, the majority are also married individuals. This indicates that being married does not necessarily imply a higher risk of loan default. Other factors such as income stability, financial management skills, and responsible borrowing behavior may play a significant role in loan repayment success.

In [169]:
```
# Plotting two plots for delaulters and non defaulters on basis of Occupation Type
plt.figure(figsize=(18,5))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'OCCUPATION_TYPE',data=new_app_target_1)
plt.title('Defaulters')
ax.set(xlabel='Occupation type')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')


plt.subplot(1,2,2)
ax = sns.countplot(x = 'OCCUPATION_TYPE',data=new_app_target_0)
plt.title('Non-Defaulters')
ax.set(xlabel='Occupation type')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')
```



In the above graph Laborers are the higher in both the cases Defaulter & Non-defaulter. While the other occupation types are lower compare to Laborers. So Occupation not playing any curcial role in loan-repayment.

In [170]:
```
# Plotting two plots for delaulters and non defaulters on basis of Organization Type
plt.figure(figsize=(30,10))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'ORGANIZATION_TYPE',data=new_app_target_1)
plt.title('Defaulters')
ax.set(xlabel='Organization type')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')


plt.subplot(1,2,2)
ax = sns.countplot(x = 'ORGANIZATION_TYPE',data=new_app_target_0)
plt.title('Non-Defaulters')
ax.set(xlabel='Organization type')
temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')
```

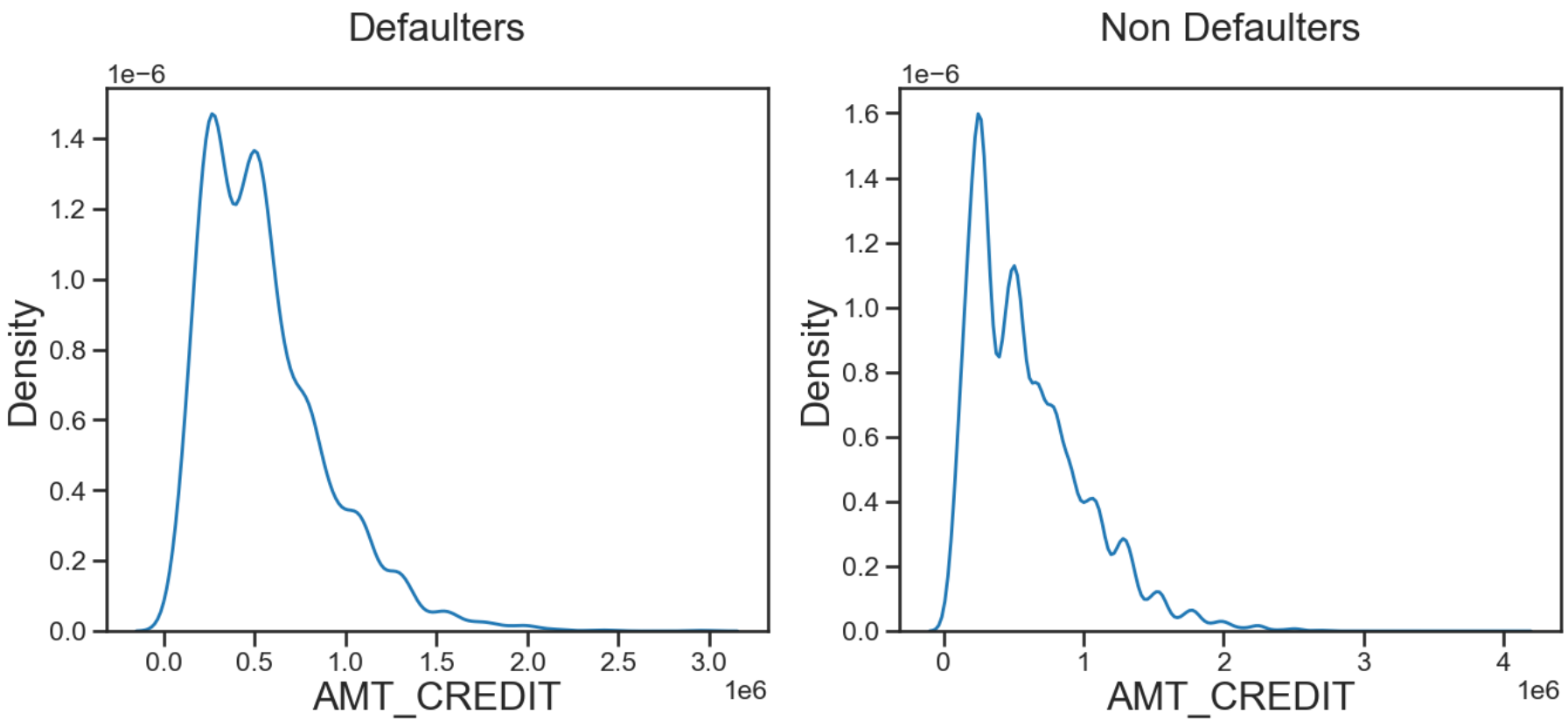


In the both the cases Business Entity Type 3 is higher on both the ends followed by self employed & Not Disclosed Which show same pettern in Default & Non-default

Analysis on Numeric variable

In [171]:
```
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
plt.title('Defaulters')
sns.distplot(new_app_target_1['AMT_CREDIT'],hist=False)

plt.subplot(1,2,2)
plt.title('Non Defaulters')
sns.distplot(new_app_target_0['AMT_CREDIT'],hist=False)
```

Out[171]: <Axes: title={'center': 'Non Defaulters'}, xlabel='AMT_CREDIT', ylabel='Density'>

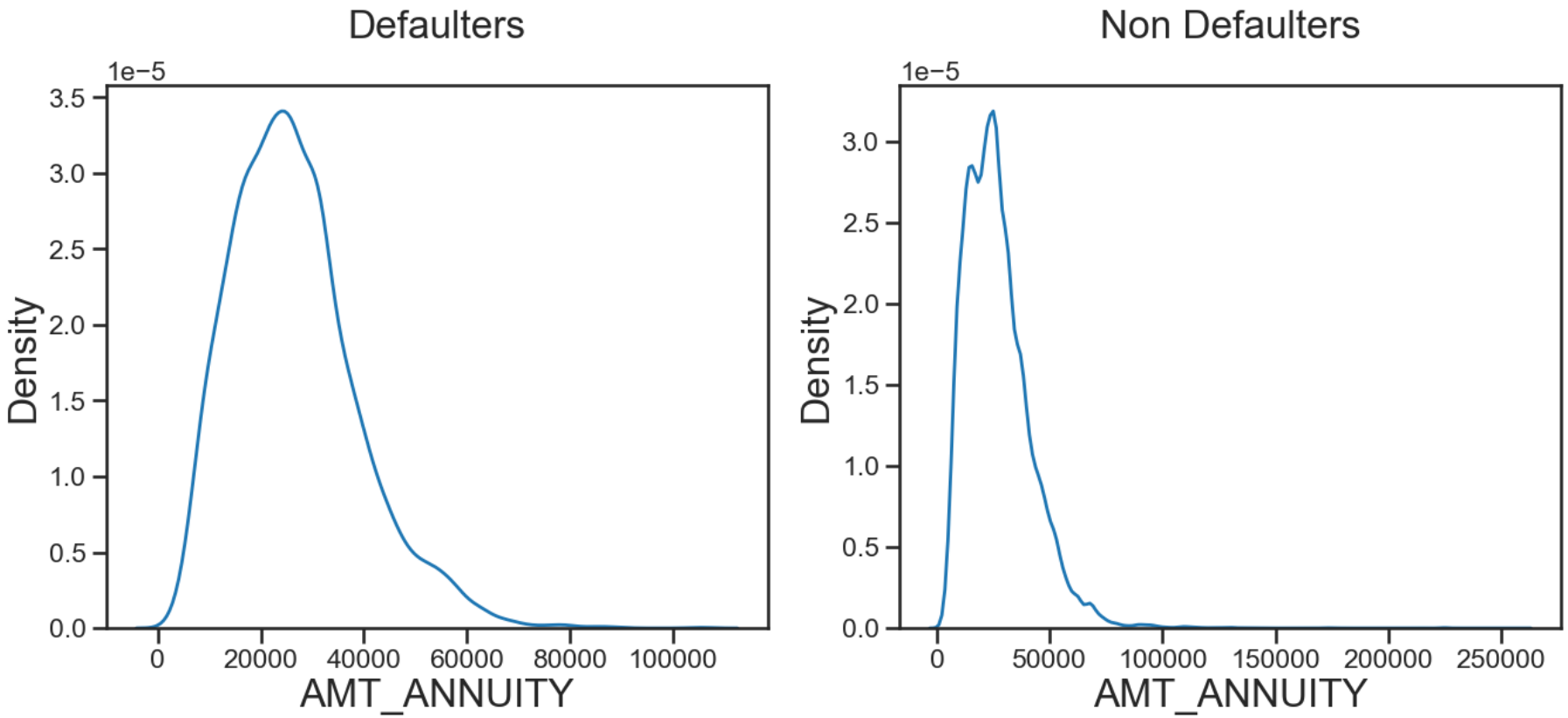


Defaulters: The analysis reveals that there is a clear trend among defaulters regarding the loan amount. As the loan amount decreases, the probability of default increases. There is a spike in default rates for loans with credit amounts up to 500,000. This suggests that borrowers with lower loan amounts may have a higher likelihood of defaulting on their loans, possibly due to financial constraints or difficulties in meeting repayment obligations.

Non-defaulters: On the other hand, for non-defaulters, there is a gradual decrease in the probability of default as the loan amount increases. Borrowers with higher loan amounts tend to have a lower likelihood of defaulting, indicating that they may have a stronger financial capacity to repay larger loans.

In [172]:
```
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
plt.title('Defaulters')
sns.distplot(new_app_target_1['AMT_ANNUITY'],hist=False)

plt.subplot(1,2,2)
plt.title('Non Defaulters')
sns.distplot(new_app_target_0['AMT_ANNUITY'],hist=False)
```

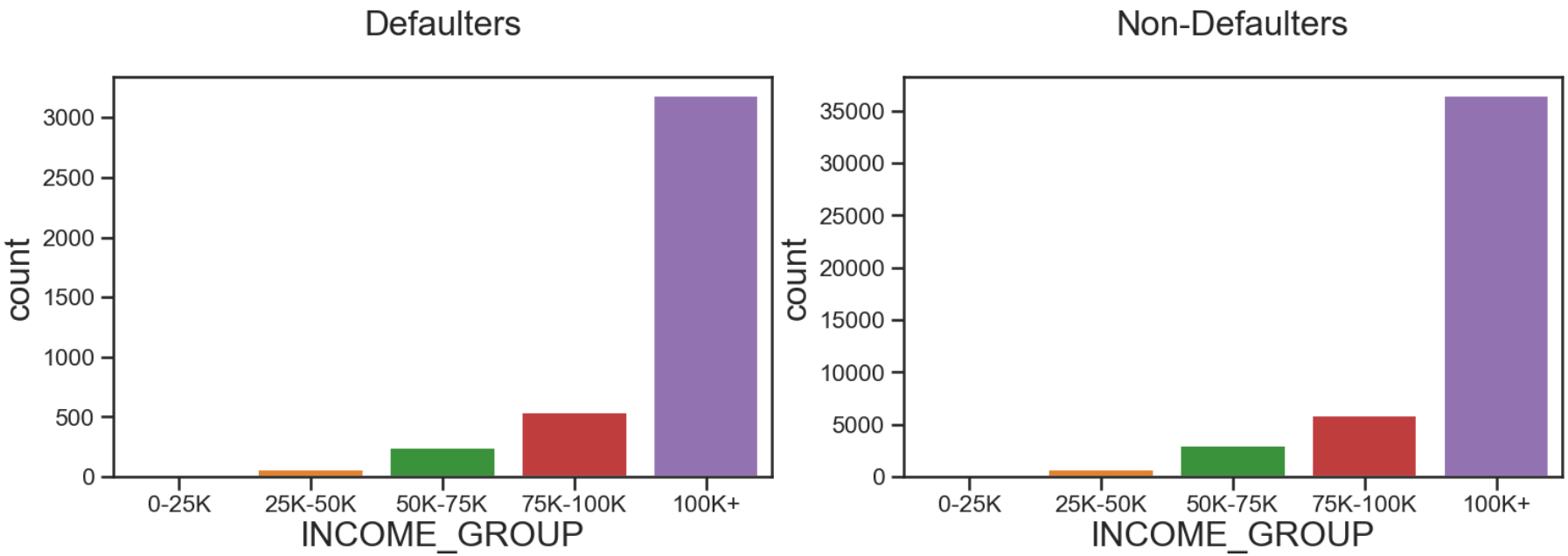
Out[172]: <Axes: title={'center': 'Non Defaulters'}, xlabel='AMT_ANNUITY', ylabel='Density'>



In both cases, whether for defaulters or non-defaulters, the distribution plot of loan annuity shows a concentration of values between 10,000 and 40,000. This indicates that a significant number of loan applicants, regardless of their default status, have monthly loan annuity payments falling within this range.

In [174]:
```
plt.figure(figsize=(18,5))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'INCOME_GROUP',data=new_app_target_1)
plt.title('Defaulters')

plt.subplot(1,2,2)
ax = sns.countplot(x = 'INCOME_GROUP',data=new_app_target_0)
plt.title('Non-Defaulters')
```

Out[174]:
```
Text(0.5, 1.0, 'Non-Defaulters')
```

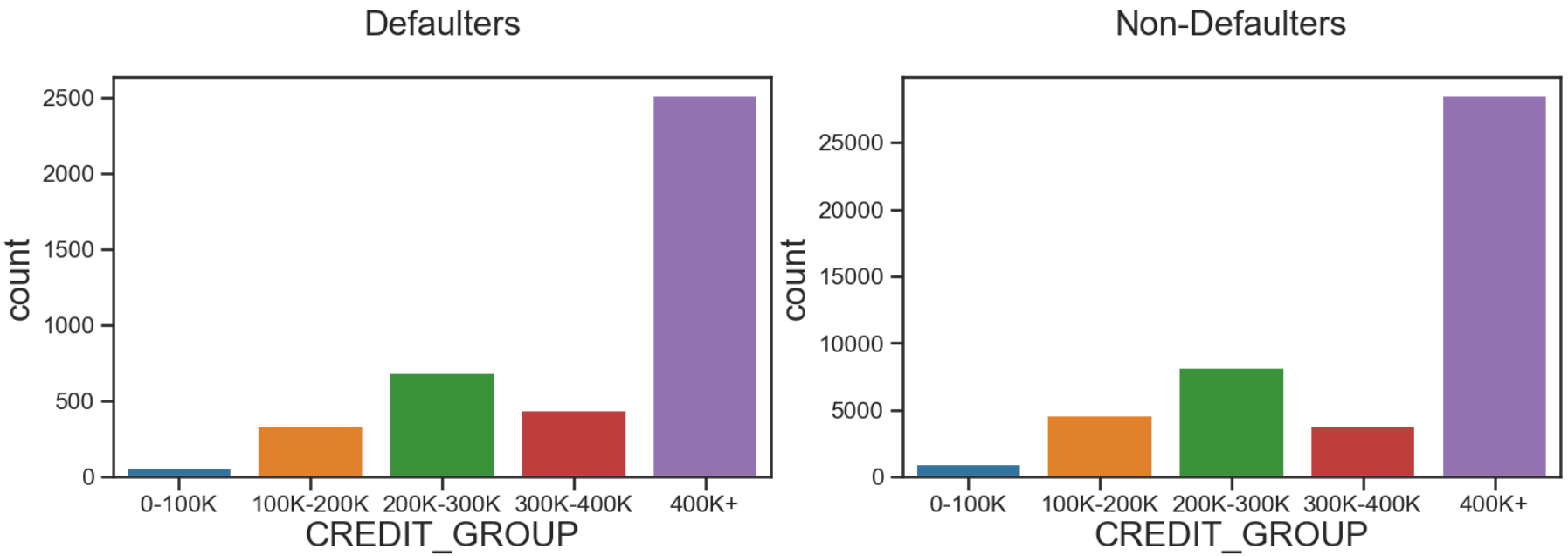


Defaulters: Higher-income individuals surprisingly have a higher default rate compared to medium and low-income groups, challenging the assumption of lower default risk with higher income. In contrast, the count of defaulters is relatively lower in the low-income group.

Non-defaulters: The count of non-defaulters is higher in the high-income group and lower in the low-income group, aligning with the expectation that higher income is associated with a lower likelihood of default.

In [175]:
```
plt.figure(figsize=(18,5))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'CREDIT_GROUP',data=new_app_target_1)
plt.title('Defaulters')

plt.subplot(1,2,2)
ax = sns.countplot(x = 'CREDIT_GROUP',data=new_app_target_0)
plt.title('Non-Defaulters')
```

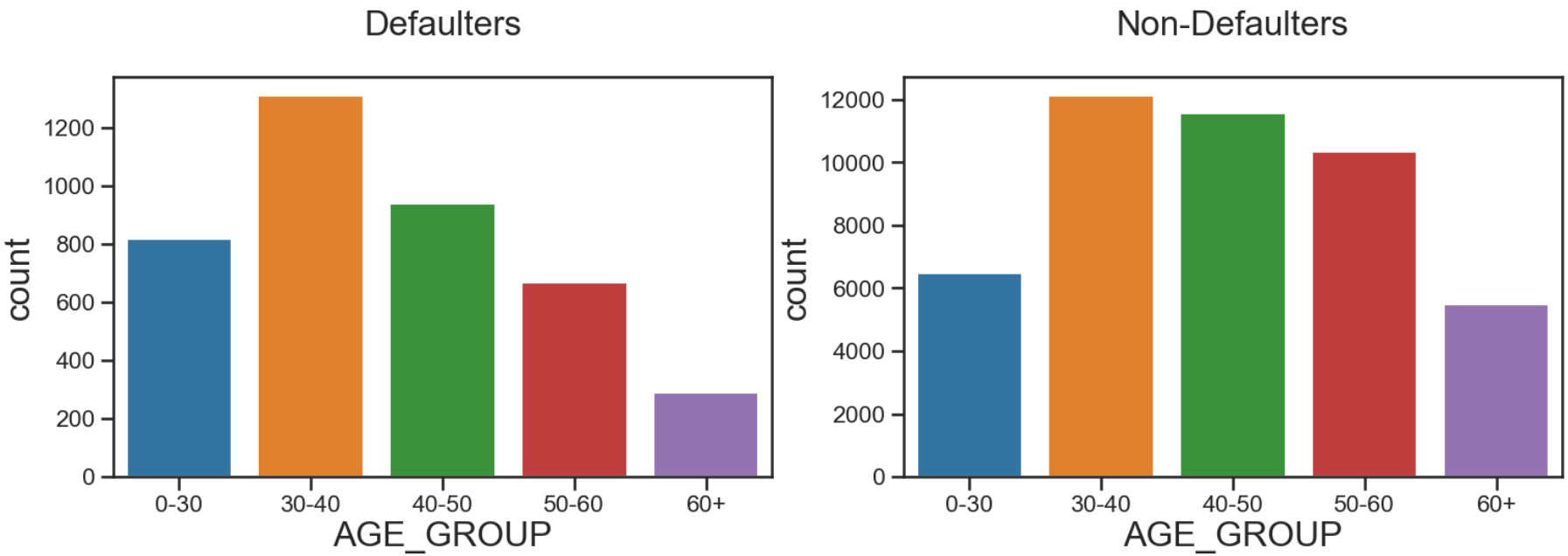
Out[175]:
```
Text(0.5, 1.0, 'Non-Defaulters')
```



Upon analyzing the data, we observe interesting patterns related to the credit amount and the likelihood of defaulting. For defaulters, there is a higher concentration of individuals in the higher credit amount groups, indicating that those with higher credited amounts are more likely to default on their loans. On the other hand, non-defaulters also show a similar trend, with a larger proportion of individuals falling into the higher credit amount groups.

In [176]:
```
plt.figure(figsize=(18,5))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'AGE_GROUP',data=new_app_target_1)
plt.title('Defaulters')

plt.subplot(1,2,2)
ax = sns.countplot(x = 'AGE_GROUP',data=new_app_target_0)
plt.title('Non-Defaulters')
```

Out[176]:
```
Text(0.5, 1.0, 'Non-Defaulters')
```

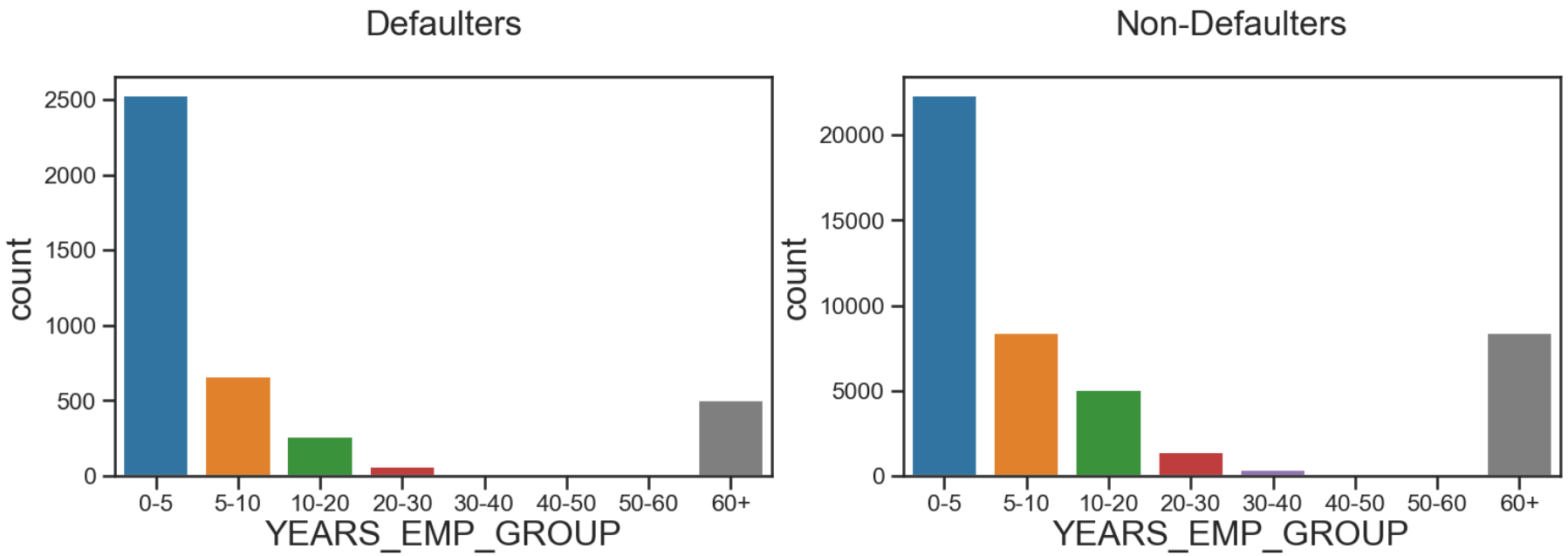


In the case of defaulters, the analysis reveals that young people are more likely to default compared to individuals in the other two age groups. On the other hand, senior citizens are less likely to default than individuals in the younger age groups.

For non-defaulters, there isn't a significant difference in the likelihood of default across the age groups. This suggests that age may not be a strong determining factor for loan repayment behavior among non-defaulters.

In [177]:
```
plt.figure(figsize=(18,5))

plt.subplot(1,2,1)
ax = sns.countplot(x = 'YEARS_EMP_GROUP',data=new_app_target_1)
plt.title('Defaulters')

plt.subplot(1,2,2)
ax = sns.countplot(x = 'YEARS_EMP_GROUP',data=new_app_target_0)
plt.title('Non-Defaulters')
```

Out[177]:
```
Text(0.5, 1.0, 'Non-Defaulters')
```



Defaulters: 0-5 years of experience people surprisingly have a higher default rate compared to other groups, challenging the assumption of lower default risk with 20-30 & 30-40 experience groups. In contrast, the count of defaulters is relatively lower in the low experienced group.

Non-defaulters: The count of 0-5 experience is higher in the group and lower in the 20-30 & 30-40 group, which is similar with default.

E. Identify Top Correlations for Different Scenarios:

Corelation of relevant numerical columns for defaulters and non defaulters

In [98]:
```
# Listing the relevant columns for finding corelation
corr_cols = ['AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY','AMT_GOODS_PRICE','AGE','CNT_FAM_MEMBERS']
```

Corelation of defaulters

In [99]:
```
new_app_target_1 = new_app_target_1[corr_cols]
new_app_target_1.head()
```

Out[99]:

	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	AGE	CNT_FAM_MEMBERS
0	202500.0	406597.5	24700.5	351000.0	26	1.0
26	112500.0	979992.0	27076.5	702000.0	52	1.0
40	202500.0	1193580.0	35028.0	855000.0	48	2.0
42	135000.0	288873.0	16258.5	238500.0	37	2.0
81	81000.0	252000.0	14593.5	252000.0	68	2.0

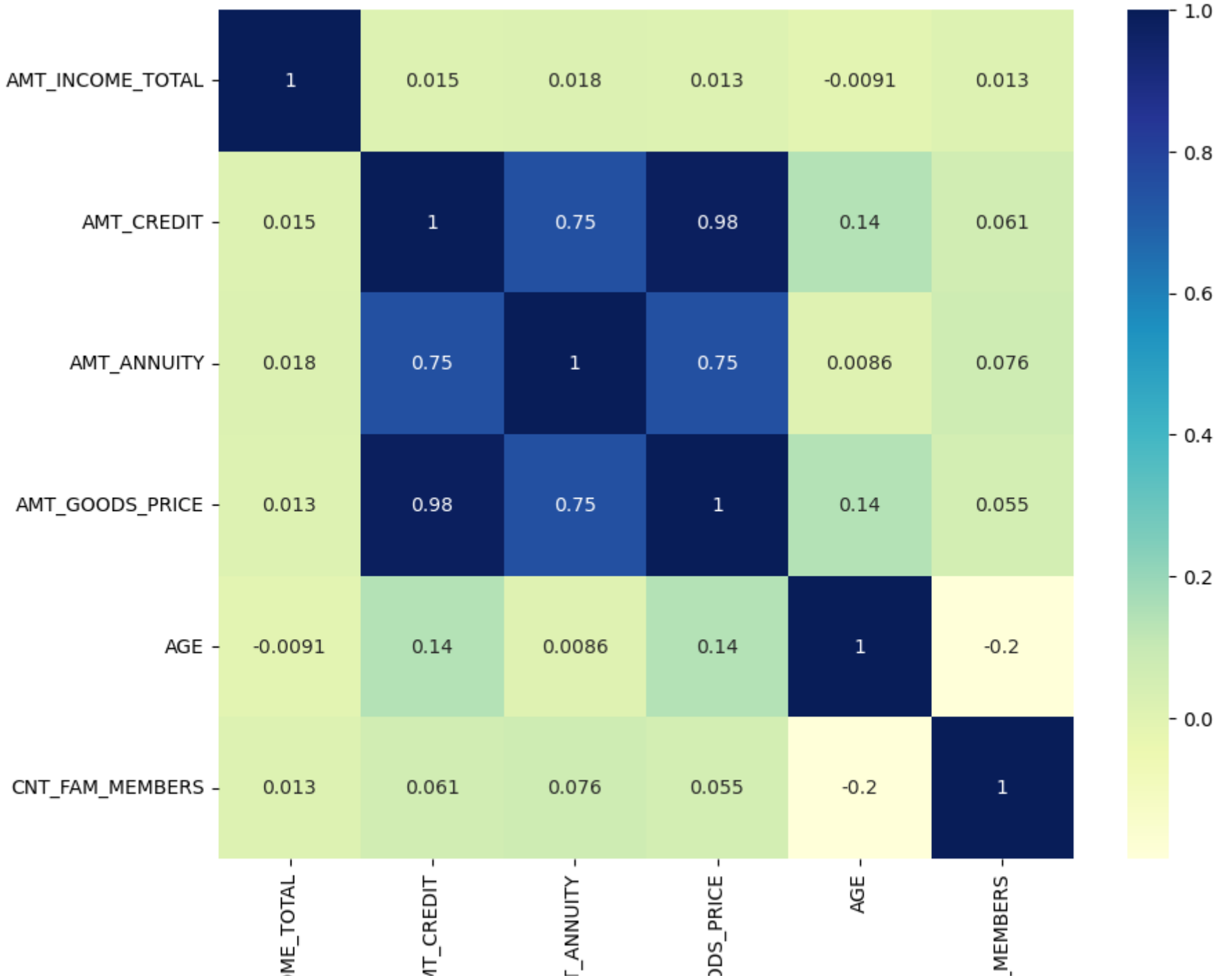
In [100]:
```
new_app_target_1.corr()
```

Out[100]:

	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	AGE	CNT_FAM_MEMBERS
AMT_INCOME_TOTAL	1.000000	0.015271	0.018005	0.013270	-0.009108	0.013122
AMT_CREDIT	0.015271	1.000000	0.749665	0.982268	0.142405	0.061249
AMT_ANNUITY	0.018005	0.749665	1.000000	0.749504	0.008563	0.075838
AMT_GOODS_PRICE	0.013270	0.982268	0.749504	1.000000	0.140872	0.055136
AGE	-0.009108	0.142405	0.008563	0.140872	1.000000	-0.199451
CNT_FAM_MEMBERS	0.013122	0.061249	0.075838	0.055136	-0.199451	1.000000

In [101]:
```
plt.figure(figsize=(10,8))
sns.heatmap(new_app_target_1.corr(),cmap="YlGnBu",annot=True)
```

Out[101]: <Axes: >


AMT_INCOME_TOTAL
AMT_CREDIT
AMT_ANNUITY
AMT_GOODS_PRICE
AGE
CNT_FAM_MEMBERS
1 0.015 0.018 0.013 -0.0091 0.013
0.015 1 0.75 0.98 0.14 0.061
0.018 0.75 1 0.75 0.0086 0.076
0.013 0.98 0.75 1 0.14 0.055
-0.0091 0.14 0.0086 0.14 1 -0.2
0.013 0.061 0.076 0.055 -0.2 1
ME_TOTAL
T_CREDIT
T_ANNUITY
ODS_PRICE
AGE
_MEMBERS
1.0
0.8
0.6
0.4
0.2
0.0

Highly correlated columns for defaulters:

AMT_CREDIT and AMT_ANNUITY: These two variables show a strong positive correlation of 0.75 among defaulters. It suggests that as the loan amount (AMT_CREDIT) increases, the corresponding annuity payments (AMT_ANNUITY) also tend to increase.

AMT_CREDIT and AMT_GOODS_PRICE: Defaulters exhibit a remarkably high correlation of 0.98 between the loan amount (AMT_CREDIT) and the price of the goods (AMT_GOODS_PRICE) being financed. This indicates that the loan amount is closely aligned with the value of the purchased goods.

AMT_ANNUITY and AMT_GOODS_PRICE: There is a moderate positive correlation of 0.75 between the annuity payments (AMT_ANNUITY) and the price of the goods (AMT_GOODS_PRICE) among defaulters. As the price of the goods increases, the corresponding annuity payments also tend to increase.

Corelation of non defaulters

In [102]:
```
new_app_target_0 = new_app_target_0[corr_cols]
new_app_target_0.head()
```

Out[102]:

	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	AGE	CNT_FAM_MEMBERS
1	270000.0	1293502.5	35698.5	1129500.0	46	2.0
2	67500.0	135000.0	6750.0	135000.0	53	1.0
3	135000.0	312682.5	29686.5	297000.0	53	2.0
4	121500.0	513000.0	21865.5	513000.0	55	1.0
5	99000.0	490495.5	27517.5	454500.0	47	2.0

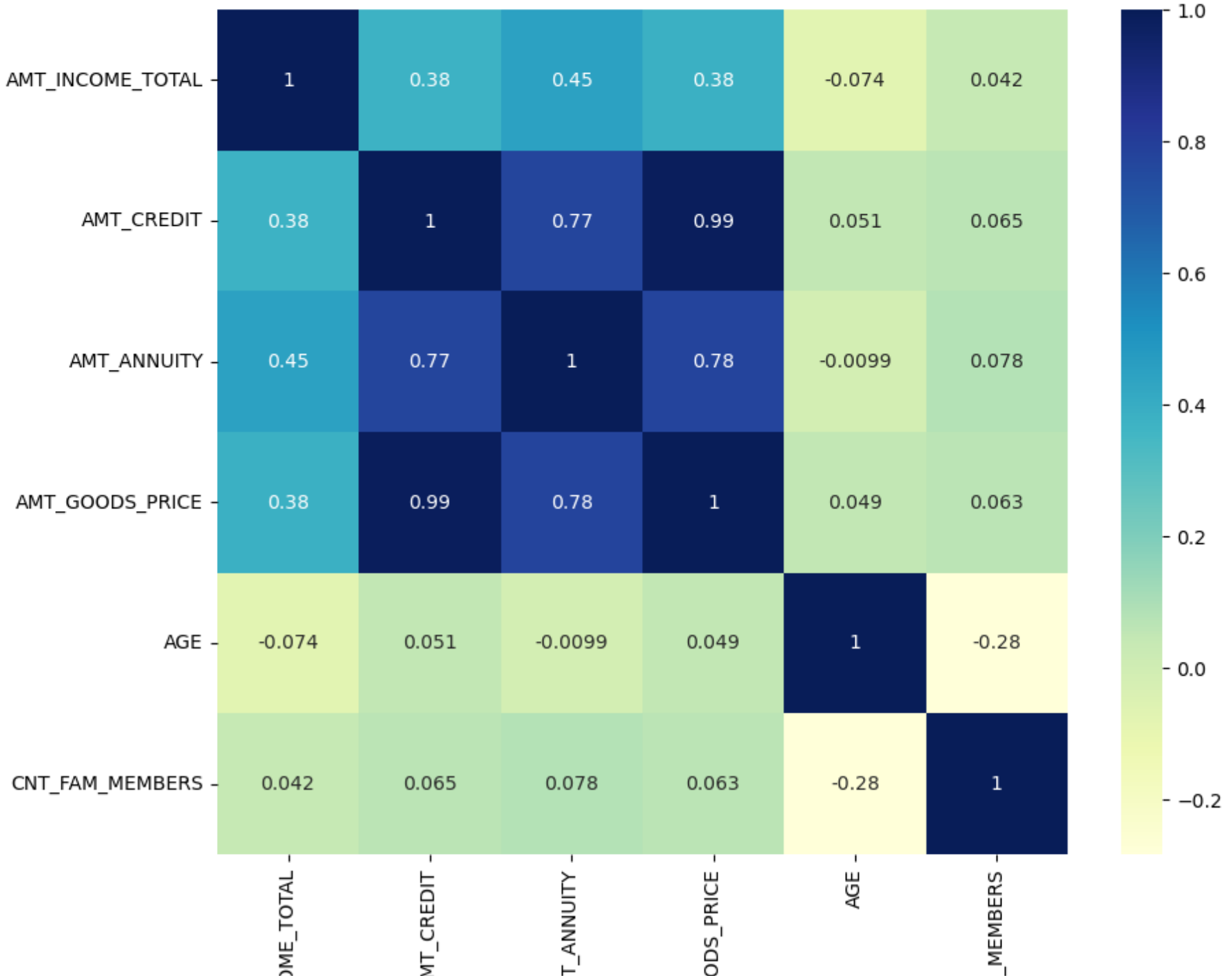
In [103]:
```
new_app_target_0.corr()
```

Out[103]:

	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	AGE	CNT_FAM_MEMBERS
AMT_INCOME_TOTAL	1.000000	0.377966	0.451135	0.384576	-0.073711	0.041599
AMT_CREDIT	0.377966	1.000000	0.770773	0.987000	0.051047	0.064877
AMT_ANNUITY	0.451135	0.770773	1.000000	0.775835	-0.009929	0.077893
AMT_GOODS_PRICE	0.384576	0.987000	0.775835	1.000000	0.048706	0.062892
AGE	-0.073711	0.051047	-0.009929	0.048706	1.000000	-0.284162
CNT_FAM_MEMBERS	0.041599	0.064877	0.077893	0.062892	-0.284162	1.000000

In [104]:
```
plt.figure(figsize=(10,8))
sns.heatmap(new_app_target_0.corr(),cmap="YlGnBu",annot=True)
```

Out[104]: <Axes: >


AMT_INCOME_TOTAL
AMT_CREDIT
AMT_ANNUITY
AMT_GOODS_PRICE
AGE
CNT_FAM_MEMBERS
1
0.38
0.45
0.38
-0.074
0.042
0.38
1
0.77
0.99
0.051
0.065
0.45
0.77
1
0.78
-0.0099
0.078
0.38
0.99
0.78
1
0.049
0.063
-0.074
0.051
-0.0099
0.049
1
-0.28
0.042
0.065
0.078
0.063
-0.28
1
OME_TOTAL
MT_CREDIT
T_ANNUITY
ODS_PRICE
AGE
_MEMBERS
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Highly correlated columns for non-defaulters:

AMT_CREDIT and AMT_ANNUITY: Among non-defaulters, there is a strong positive correlation of 0.77 between the loan amount (AMT_CREDIT) and the corresponding annuity payments (AMT_ANNUITY). This suggests that as the loan amount increases, the associated annuity payments also tend to increase.

AMT_CREDIT and AMT_GOODS_PRICE: Non-defaulters exhibit a high correlation of 0.99 between the loan amount (AMT_CREDIT) and the price of the financed goods (AMT_GOODS_PRICE). This indicates that the loan amount is closely related to the value of the purchased goods.

AMT_ANNUITY and AMT_GOODS_PRICE: There is a moderate positive correlation of 0.78 between the annuity payments (AMT_ANNUITY) and the price of the goods (AMT_GOODS_PRICE) among non-defaulters. As the price of the goods increases, the corresponding annuity payments also tend to increase.

Conclusion:

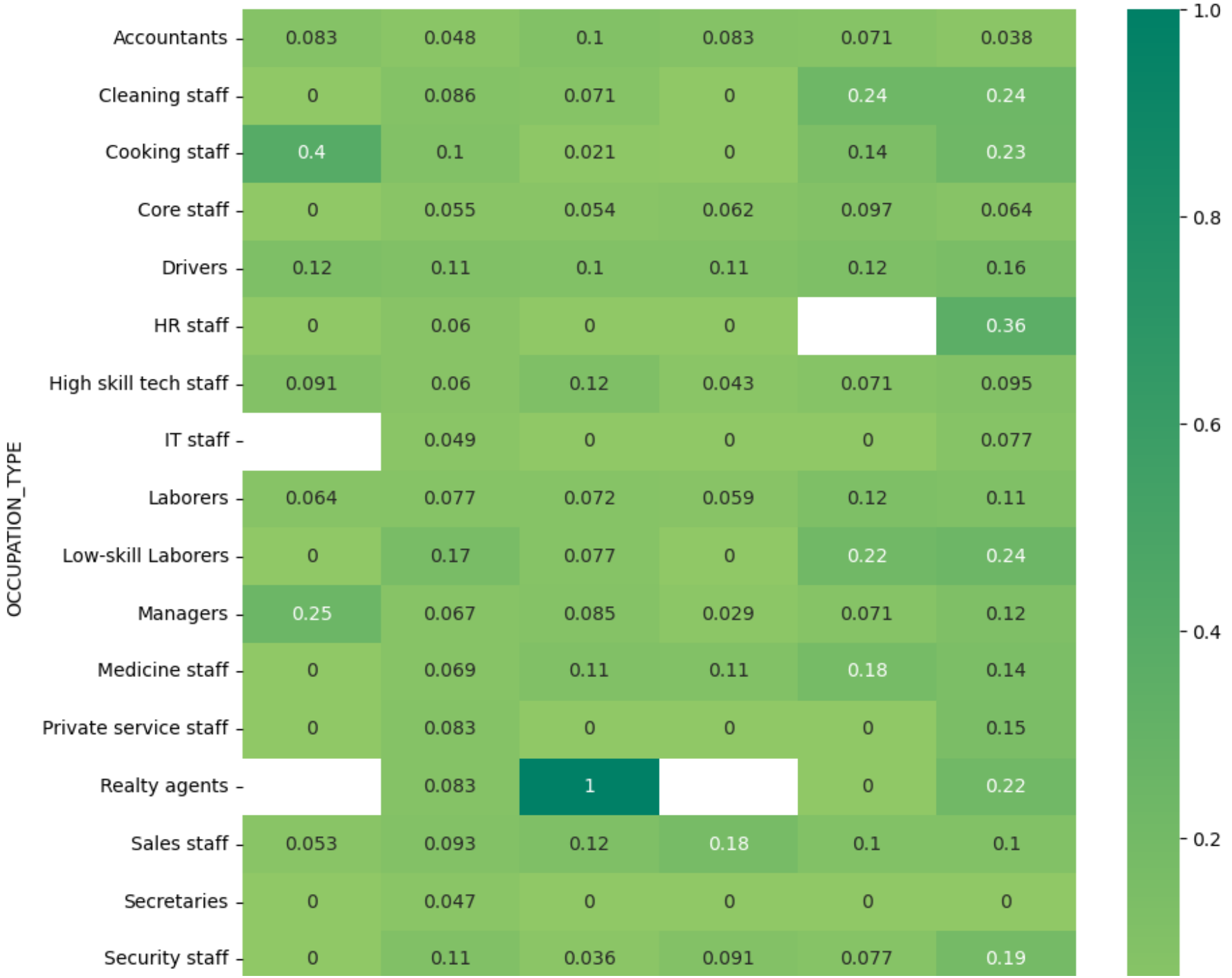
In this analysis of defaulters and non-defaulters, we observed that the same pairs of columns exhibit high correlation for both groups. Specifically, the columns AMT_CREDIT and AMT_ANNUITY, AMT_CREDIT and AMT_GOODS_PRICE, and AMT_ANNUITY and AMT_GOODS_PRICE showed significant correlations in both cases.

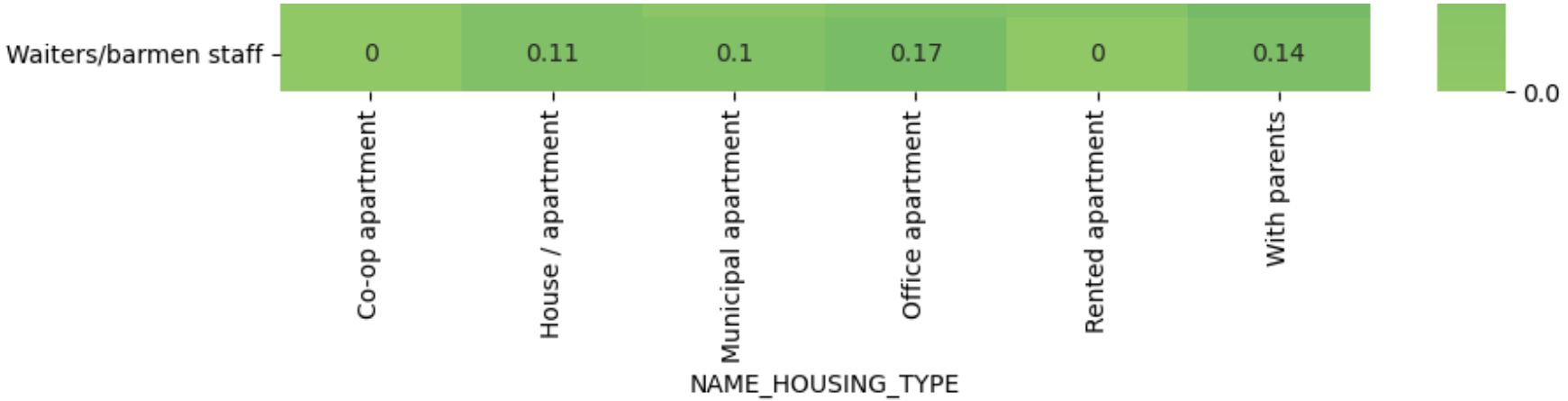
This suggests that these variables have a strong relationship with each other, regardless of the borrower's default status. The loan amount (AMT_CREDIT) appears to be closely related to both the annuity payments (AMT_ANNUITY) and the price of the financed goods (AMT_GOODS_PRICE). These insights highlight the importance of considering these variables together when assessing creditworthiness and making lending decisions.

Corelation of relevant Categorical columns for defaulters and non defaulters

- Correlation of Occupation_type vs Housing_type wrt Target

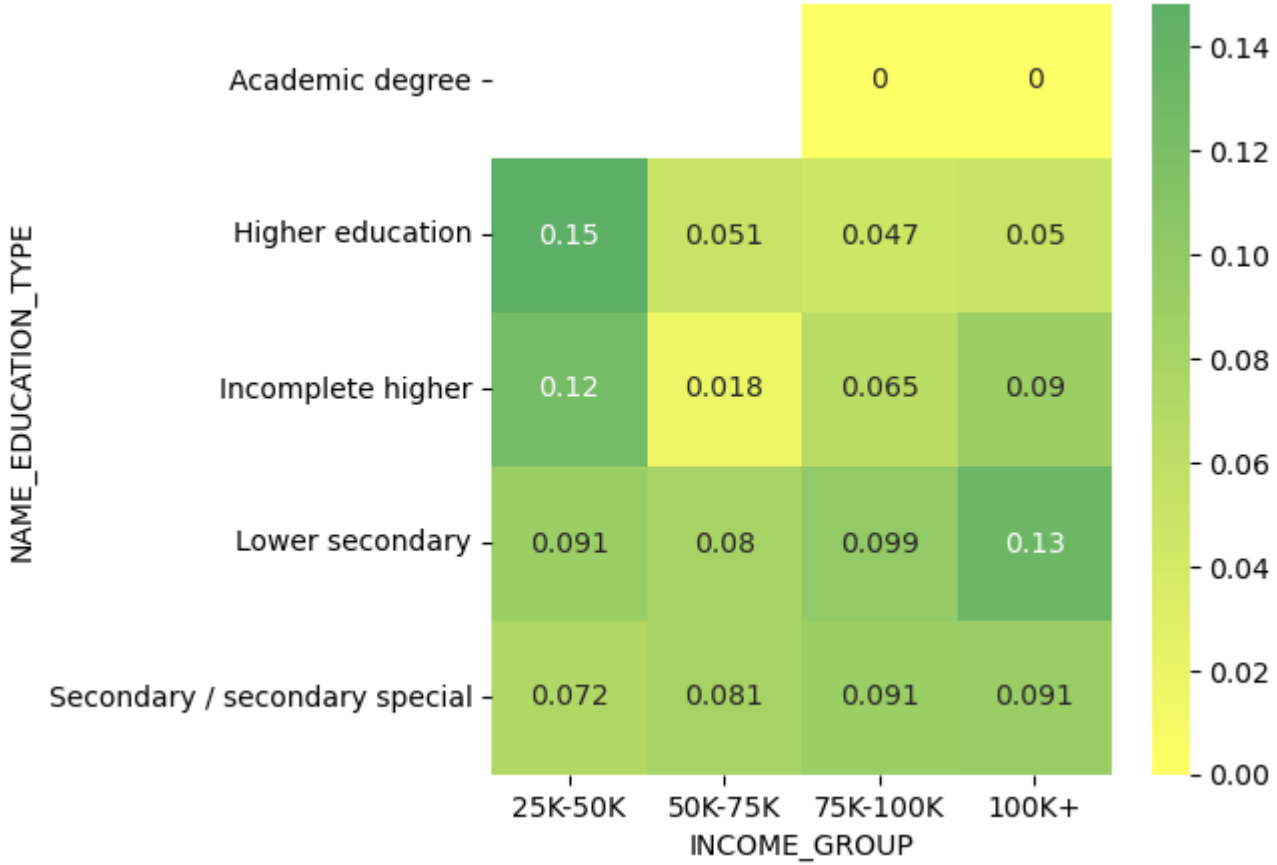
In [105]:
```
plt.figure(figsize=[10,10])
piv = pd.pivot_table(data=new_app, index="OCCUPATION_TYPE", columns="NAME_HOUSING_TYPE", values="TARGET")
sns.heatmap(piv, cmap="summer_r", annot=True, center=0.117)
plt.show()
```


OCCUPATION_TYPE
Accountants 0.083 0.048 0.1 0.083 0.071 0.038
Cleaning staff 0 0.086 0.071 0 0.24 0.24
Cooking staff 0.4 0.1 0.021 0 0.14 0.23
Core staff 0 0.055 0.054 0.062 0.097 0.064
Drivers 0.12 0.11 0.1 0.11 0.12 0.16
HR staff 0 0.06 0 0 0.36
High skill tech staff 0.091 0.06 0.12 0.043 0.071 0.095
IT staff 0.049 0 0 0 0.077
Laborers 0.064 0.077 0.072 0.059 0.12 0.11
Low-skill Laborers 0 0.17 0.077 0 0.22 0.24
Managers 0.25 0.067 0.085 0.029 0.071 0.12
Medicine staff 0 0.069 0.11 0.11 0.18 0.14
Private service staff 0 0.083 0 0 0 0.15
Realty agents 0.083 1 0 0.22
Sales staff 0.053 0.093 0.12 0.18 0.1 0.1
Secretaries 0 0.047 0 0 0 0
Security staff 0 0.11 0.036 0.091 0.077 0.19
1.0
0.8
0.6
0.4
0.2



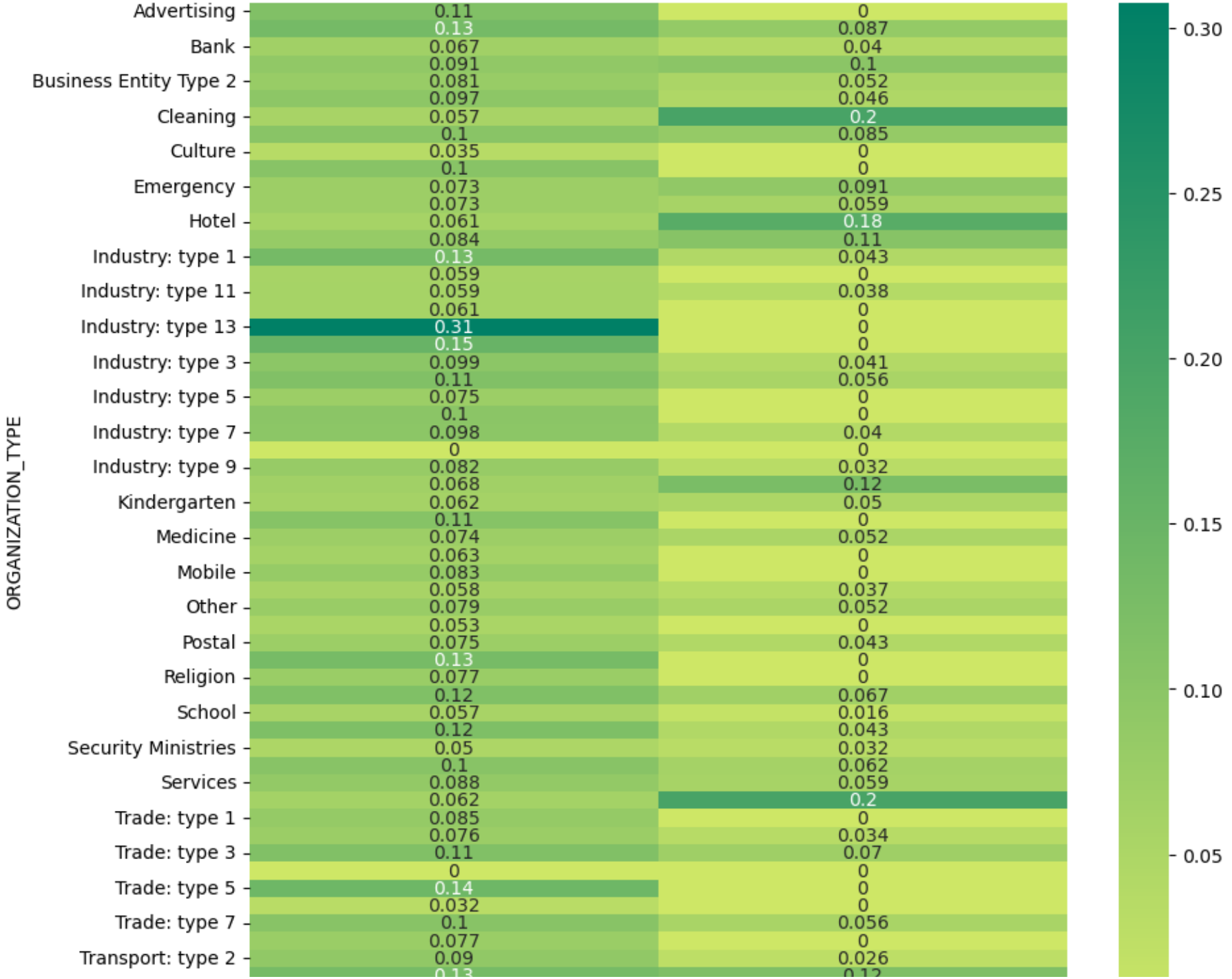
- Correlation of Name_Education_type vs Income_class wrt Target

In [106]:
```
plt.figure(figsize=[5,5])
piv = pd.pivot_table(data=new_app, index="NAME_EDUCATION_TYPE", columns="INCOME_GROUP", values="TARGET")
sns.heatmap(piv, cmap="summer_r", annot=True, center=0.117)
plt.show()
```



- Correlation of Organization_type vs Contract_type wrt Target

In [108]:
```
plt.figure(figsize=[10,10])
piv = pd.pivot_table(data=new_app, index="ORGANIZATION_TYPE", columns="NAME_CONTRACT_TYPE", values="TARGET")
sns.heatmap(piv, cmap="summer_r", annot=True, center=0.117)
plt.show()
```


ORGANIZATION_TYPE
Advertising
Bank
Business Entity Type 2
Cleaning
Culture
Emergency
Hotel
Industry: type 1
Industry: type 11
Industry: type 13
Industry: type 3
Industry: type 5
Industry: type 7
Industry: type 9
Kindergarten
Medicine
Mobile
Other
Postal
Religion
School
Security Ministries
Services
Trade: type 1
Trade: type 3
Trade: type 5
Trade: type 7
Transport: type 2
0.11 0
0.13 0.087
0.067 0.04
0.091 0.1
0.081 0.052
0.097 0.046
0.057 0.2
0.1 0.085
0.035 0
0.1 0
0.073 0.091
0.073 0.059
0.061 0.18
0.084 0.11
0.13 0.043
0.059 0
0.059 0.038
0.061 0
0.31 0
0.15 0
0.099 0.041
0.11 0.056
0.075 0
0.1 0
0.098 0.04
0 0
0.082 0.032
0.068 0.12
0.062 0.05
0.11 0
0.074 0.052
0.063 0
0.083 0
0.058 0.037
0.079 0.052
0.053 0
0.075 0.043
0.13 0
0.077 0
0.12 0.067
0.057 0.016
0.12 0.043
0.05 0.032
0.1 0.062
0.088 0.059
0.062 0.2
0.085 0
0.076 0.034
0.11 0.07
0 0
0.14 0
0.032 0
0.1 0.056
0.077 0
0.09 0.026
0.30
0.25
0.20
0.15
0.10
0.05

Previous Application

Loading previous_application data

In [109]:
```
df2 = pd.read_csv("previous_application.csv")
df2.head()
```

Out[109]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_P
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	17
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	607
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	112
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	450
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	337

5 rows × 37 columns

In [110]:
```
df2.describe()
```

Out[110]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	HOUR_APPR_
count	4.999900e+04	49999.000000	39407.000000	4.999900e+04	4.999900e+04	2.480100e+04	3.925500e+04	
mean	1.922254e+06	278983.187604	15482.596847	1.688925e+05	1.885429e+05	6.557571e+03	2.151414e+05	
std	5.351980e+05	102780.124434	14530.971854	2.822035e+05	3.084736e+05	1.744458e+04	3.024993e+05	
min	1.000001e+06	100007.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
25%	1.457920e+06	189919.500000	6122.835000	2.204550e+04	2.605500e+04	0.000000e+00	4.941000e+04	
50%	1.920889e+06	279264.000000	10879.920000	7.155000e+04	7.890750e+04	1.566000e+03	1.040175e+05	
75%	2.388632e+06	368527.500000	19669.140000	1.800000e+05	1.981058e+05	7.875000e+03	2.250000e+05	
max	2.845367e+06	456254.000000	234478.395000	3.826372e+06	4.104351e+06	1.035000e+06	3.826372e+06	

8 rows × 21 columns

In [111]:
```
df2.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 [112]:
```
df2.shape
```

Out[112]:
```
(49999, 37)
```

In [113]:
```
df2.isnull().sum().sum() # Missing Vlues in dataset
```

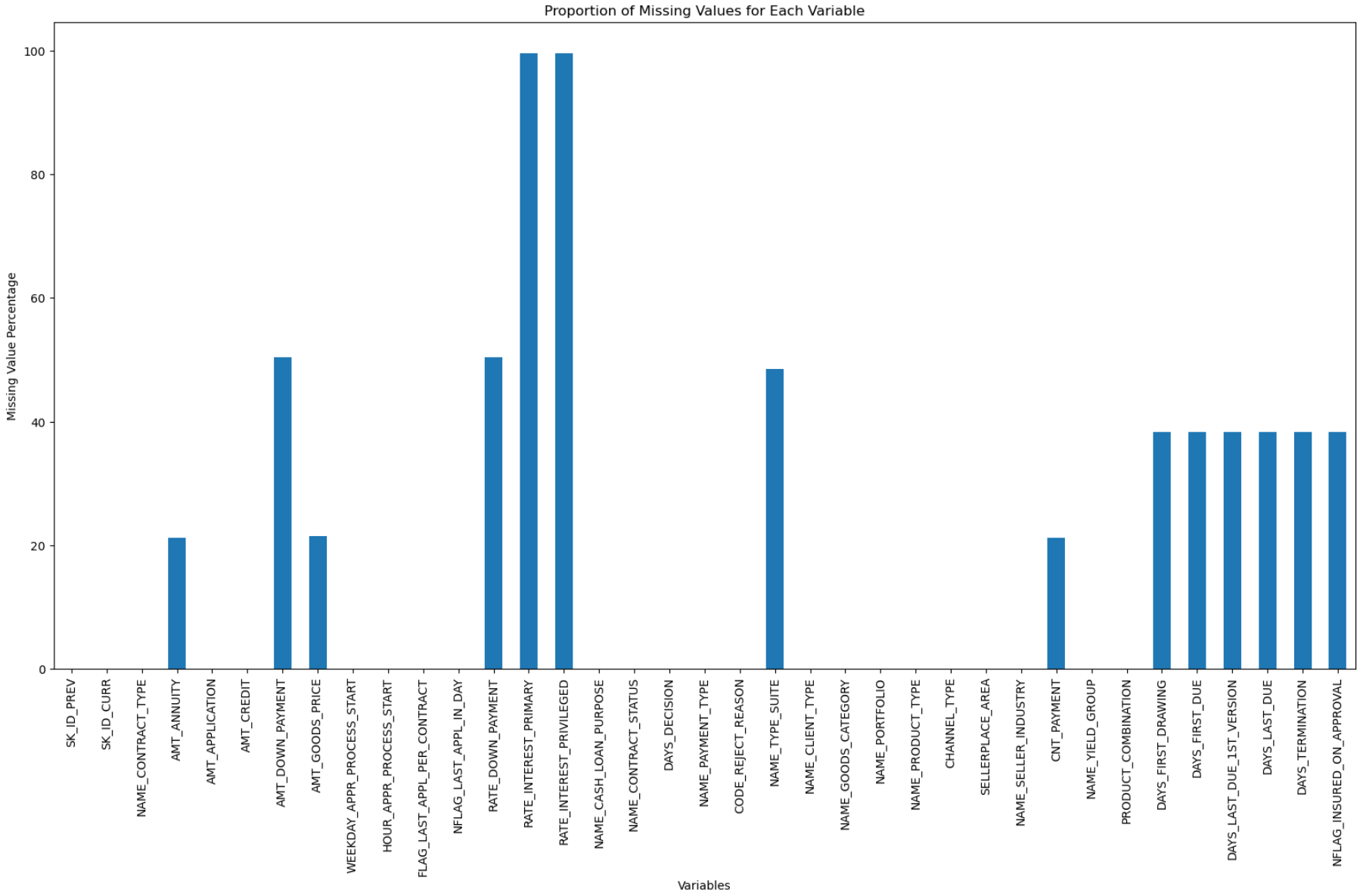
Out[113]:
```
321203
```

Identify Missing Data and Dealing it Appropriately

In [114]:
```
missing_proportions1 = (df2.isnull().sum() / len(df2)) * 100
```

In [115]:

```
plt.figure(figsize=(20, 10))
missing_proportions1.plot(kind='bar')
plt.xlabel('Variables')
plt.ylabel('Missing Value Percentage')
plt.title('Proportion of Missing Values for Each Variable')
plt.xticks(rotation=90)

plt.show()
```



In [116]:
```
mask1 = missing_proportions1 > 10
```

In [117]:
```
columns_with_more_than_10_percent_missing = missing_proportions1[mask1] # Filter columns with more than 10% missing va
```

In [118]:
```
print(columns_with_more_than_10_percent_missing)
```

```
AMT_ANNUITY                 21.184424
AMT_DOWN_PAYMENT            50.397008
AMT_GOODS_PRICE             21.488430
RATE_DOWN_PAYMENT           50.397008
RATE_INTEREST_PRIMARY       99.669993
RATE_INTEREST_PRIVILEGED    99.669993
NAME_TYPE_SUITE             48.486970
CNT_PAYMENT                 21.184424
DAYS_FIRST_DRAWING          38.320766
DAYS_FIRST_DUE              38.320766
DAYS_LAST_DUE_1ST_VERSION   38.320766
DAYS_LAST_DUE               38.320766
DAYS_TERMINATION            38.320766
NFLAG_INSURED_ON_APPROVAL   38.320766
dtype: float64
```

- As per the above missing percentage we will drop the columns with missing values above 30%
- we will drop unneccessary columns
- Also we can see columns with having huge amount of rows filled with "XAP" & "XNA" is nothing but the null values should be droped as well

In [119]:
```
#Dropping columns
pre_app = pd.DataFrame(df2.drop( ["AMT_DOWN_PAYMENT","WEEKDAY_APPR_PROCESS_START","HOUR_APPR_PROCESS_START","FLAG_LAST
          "NFLAG_LAST_APPL_IN_DAY","RATE_DOWN_PAYMENT","RATE_INTEREST_PRIMARY","RATE_INTEREST_PRIVILEGED","DAYS_DECISI
          ,"NAME_PORTFOLIO","NAME_PRODUCT_TYPE","SELLERPLACE_AREA","NAME_SELLER_INDUSTRY","NAME_YIELD_GROUP"
          ,"CODE_REJECT_REASON", "PRODUCT_COMBINATION","DAYS_FIRST_DRAWING","DAYS_FIRST_DUE","DAYS_LAST_DUE_1ST_VERSIO
```

In [120]:
```
pre_app.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   SK_ID_CURR            49999 non-null  int64  
 1   NAME_CONTRACT_TYPE    49999 non-null  object 
 2   AMT_ANNUITY           39407 non-null  float64
 3   AMT_APPLICATION       49999 non-null  float64
 4   AMT_CREDIT            49999 non-null  float64
 5   AMT_GOODS_PRICE       39255 non-null  float64
 6   NAME_CONTRACT_STATUS  49999 non-null  object 
 7   NAME_CLIENT_TYPE      49999 non-null  object 
 8   CHANNEL_TYPE          49999 non-null  object 
 9   CNT_PAYMENT           39407 non-null  float64
dtypes: float64(5), int64(1), object(4)
memory usage: 3.8+ MB
```

In [121]:
```
pre_app.NAME_CONTRACT_STATUS.value_counts()
```

Out[121]:
```
NAME_CONTRACT_STATUS
Approved        31885
Refused          8660
Canceled         8595
Unused offer      859
Name: count, dtype: int64
```

In [122]:
```
pre_app.describe()
```

Out[122]:

	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	CNT_PAYMENT
count	49999.000000	39407.000000	4.999900e+04	4.999900e+04	3.925500e+04	39407.000000
mean	278983.187604	15482.596847	1.688925e+05	1.885429e+05	2.151414e+05	15.555891
std	102780.124434	14530.971854	2.822035e+05	3.084736e+05	3.024993e+05	13.985174
min	100007.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000
25%	189919.500000	6122.835000	2.204550e+04	2.605500e+04	4.941000e+04	6.000000
50%	279264.000000	10879.920000	7.155000e+04	7.890750e+04	1.040175e+05	12.000000
75%	368527.500000	19669.140000	1.800000e+05	1.981058e+05	2.250000e+05	18.000000
max	456254.000000	234478.395000	3.826372e+06	4.104351e+06	3.826372e+06	60.000000

- Filling missing values with Median

In [123]:
```
missing_columns = pre_app[['AMT_ANNUITY','AMT_GOODS_PRICE','CNT_PAYMENT']]
```

In [124]:
```
for column in missing_columns:
    median = pre_app[column].median()
    pre_app[column].fillna(median, inplace=True)
```

In [125]:
```
pre_app.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   SK_ID_CURR            49999 non-null  int64  
 1   NAME_CONTRACT_TYPE    49999 non-null  object 
 2   AMT_ANNUITY           49999 non-null  float64
 3   AMT_APPLICATION       49999 non-null  float64
 4   AMT_CREDIT            49999 non-null  float64
 5   AMT_GOODS_PRICE       49999 non-null  float64
 6   NAME_CONTRACT_STATUS  49999 non-null  object 
 7   NAME_CLIENT_TYPE      49999 non-null  object 
 8   CHANNEL_TYPE          49999 non-null  object 
 9   CNT_PAYMENT           49999 non-null  float64
dtypes: float64(5), int64(1), object(4)
memory usage: 3.8+ MB
```

Now check for XNA values in above categorical variables and fill them with mode

Replacing Contract type XNA values

In [126]:
```
pre_app['NAME_CONTRACT_TYPE'].mode()
```

Out[126]:
```
0    Consumer loans
Name: NAME_CONTRACT_TYPE, dtype: object
```

In [127]:
```
pre_app['NAME_CONTRACT_TYPE'].value_counts()
```

Out[127]:
```
NAME_CONTRACT_TYPE
Consumer loans     23510
Cash loans         20856
Revolving loans     5625
XNA                    8
Name: count, dtype: int64
```

In [128]:
```
pre_app.NAME_CONTRACT_TYPE.replace("XNA","Consumer loans", inplace=True)
```

In [129]:
```
pre_app['NAME_CONTRACT_TYPE'].value_counts()
```

Out[129]:
```
NAME_CONTRACT_TYPE
Consumer loans     23518
Cash loans         20856
Revolving loans     5625
Name: count, dtype: int64
```

Replacing Client type XNA values

In [130]:
```
pre_app['NAME_CLIENT_TYPE'].mode()
```

Out[130]:
```
0    Repeater
Name: NAME_CLIENT_TYPE, dtype: object
```

In [131]:
```
pre_app['NAME_CLIENT_TYPE'].value_counts()
```

Out[131]:
```
NAME_CLIENT_TYPE
Repeater     36167
New           9548
Refreshed     4227
XNA             57
Name: count, dtype: int64
```

In [184]:
```
pre_app.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   SK_ID_CURR            49999 non-null  int64  
 1   NAME_CONTRACT_TYPE    49999 non-null  object 
 2   AMT_ANNUITY           49999 non-null  float64
 3   AMT_APPLICATION       49999 non-null  float64
 4   AMT_CREDIT            49999 non-null  float64
 5   AMT_GOODS_PRICE       49999 non-null  float64
 6   NAME_CONTRACT_STATUS  49999 non-null  object 
 7   NAME_CLIENT_TYPE      49999 non-null  object 
 8   CHANNEL_TYPE          49999 non-null  object 
 9   CNT_PAYMENT           49999 non-null  float64
dtypes: float64(5), int64(1), object(4)
memory usage: 3.8+ MB
```

In [132]:
```
pre_app.NAME_CLIENT_TYPE.replace("XNA","Repeater", inplace=True)
```

In [133]:
```
pre_app['NAME_CLIENT_TYPE'].value_counts()
```

Out[133]:
```
NAME_CLIENT_TYPE
Repeater     36224
New           9548
Refreshed     4227
Name: count, dtype: int64
```

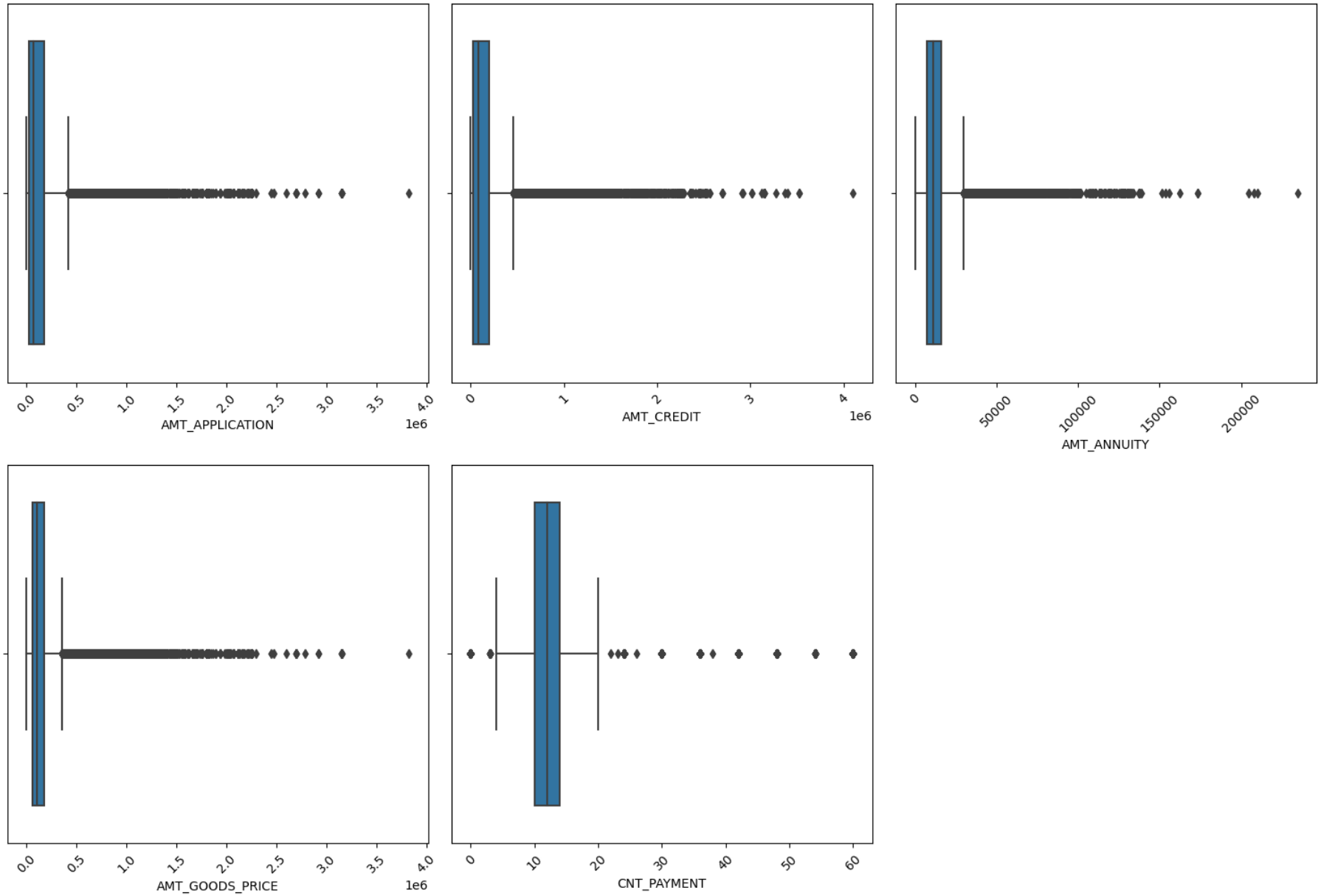
In [185]:
```
pre_app.to_excel("previous_application_Cleaned data.xlsx")
```

Outliers in the Dataset

In [134]:
```
columns_to_check1 = pre_app[['AMT_APPLICATION','AMT_CREDIT','AMT_ANNUITY','AMT_GOODS_PRICE',
                             'CNT_PAYMENT']]
```

In [135]:

```
plt.figure(figsize=(15, 20))

for i, column_name in enumerate(columns_to_check1):
    plt.subplot(4, 3, i + 1)
    sns.boxplot(data=pre_app, x=column_name)
    plt.xlabel(column_name)
    plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



1. AMT_ANNUITY, AMT_CREDIT, AMT_GOODS_PRICE, AMT_APPLICATION have some number of outliers.
2. CNT_PAYMENT has less outliers compare to other variables.

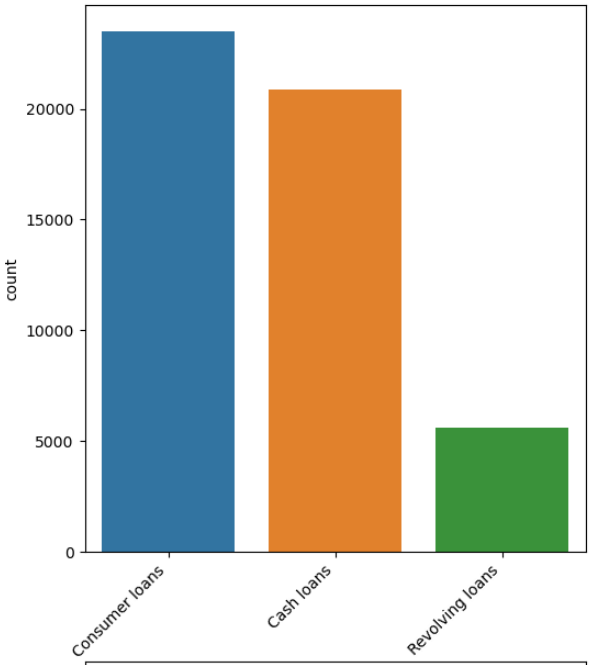
Data Imbalance

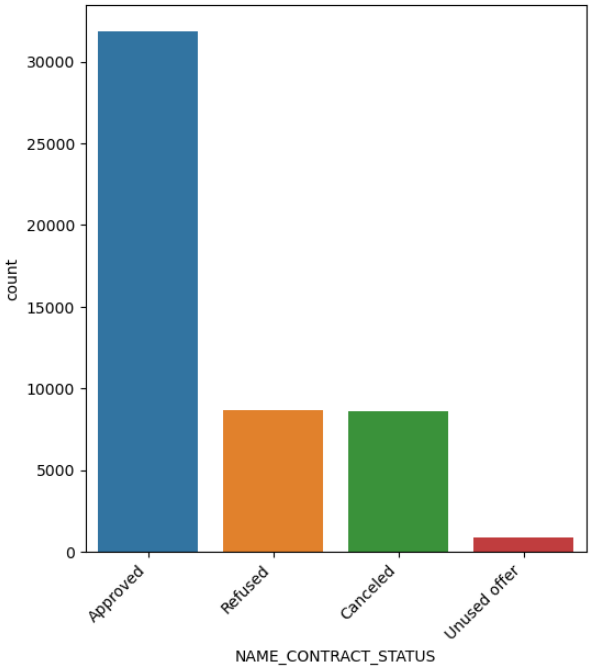
In [136]:
```
class_distribution1 = pre_app['NAME_CONTRACT_TYPE'].value_counts()
imbalance_ratio1 = class_distribution1.min() / class_distribution1.max()

print(f"Imbalance ratio: {imbalance_ratio1:.2f}")
```

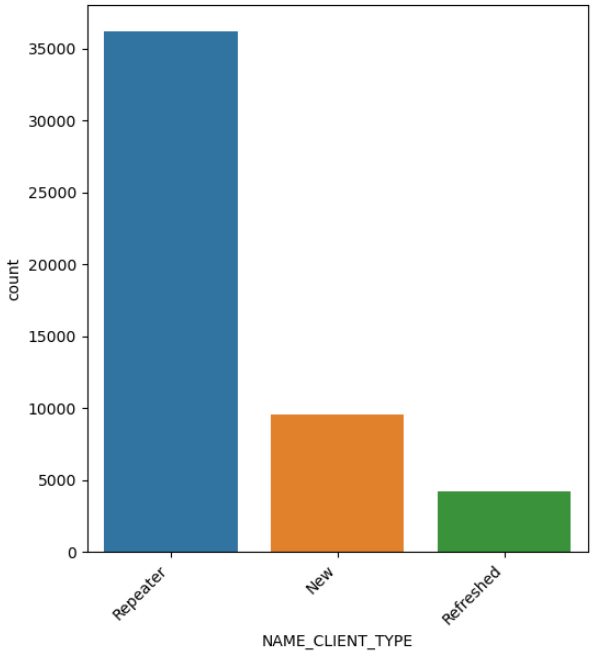
```
Imbalance ratio: 0.24
```

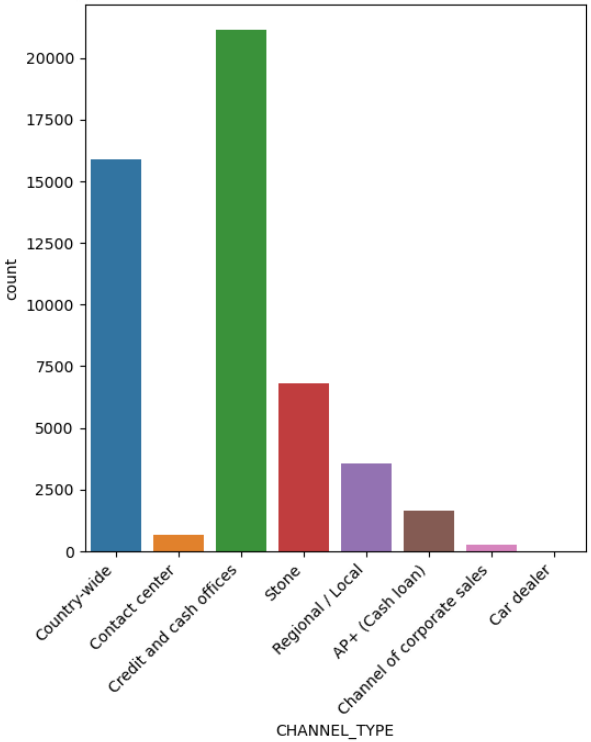
In [137]:

```
# Listing columns for check data imbalance and plotting them
col_list = ['NAME_CONTRACT_TYPE','NAME_CONTRACT_STATUS','NAME_CLIENT_TYPE','CHANNEL_TYPE']
k=0
plt.figure(figsize=(20,22))
for col in col_list:
    k=k+1
    plt.subplot(3, 3,k)
    #df_application_prev[col].value_counts().plot(kind='bar');
    ax = sns.countplot(x = col , data = pre_app )
    temp = ax.set_xticklabels(ax.get_xticklabels(), rotation = 45, horizontalalignment='right')
```


count
0
5000
10000
15000
20000
Consumer loans
Cash loans
Revolving loans


count
0
5000
10000
15000
20000
25000
30000
Approved
Refused
Canceled
Unused offer
NAME_CONTRACT_STATUS


count
0
5000
10000
15000
20000
25000
30000
35000
Repeater
New
Refreshed
NAME_CLIENT_TYPE


count
0
2500
5000
7500
10000
12500
15000
17500
20000
Country-wide
Contact center
Credit and cash offices
Stone
Regional / Local
AP+ (Cash loan)
Channel of corporate sales
Car dealer
CHANNEL_TYPE

*We have identified data imbalance in several columns:

NAME_CONTRACT_TYPE: There is an imbalance between the number of consumer loans, revolving loans and cash loans. The majority of loans are consumer loans, while the number of revolving loans is relatively low.

NAME_CONTRACT_STATUS: The majority of loan were approved in status, while Refused & Cancelled loans were near about same and Unused offers are very low.

NAME_CLIENT_TYPE: Repeater clients are high in numbers. On the other hands New & Refreshed clients are very low compare to Reapeater

CHANNEL_TYPE: Credit & Cash offers are high in numbers followed by country-wide

Univariate and Bivariate Analysis

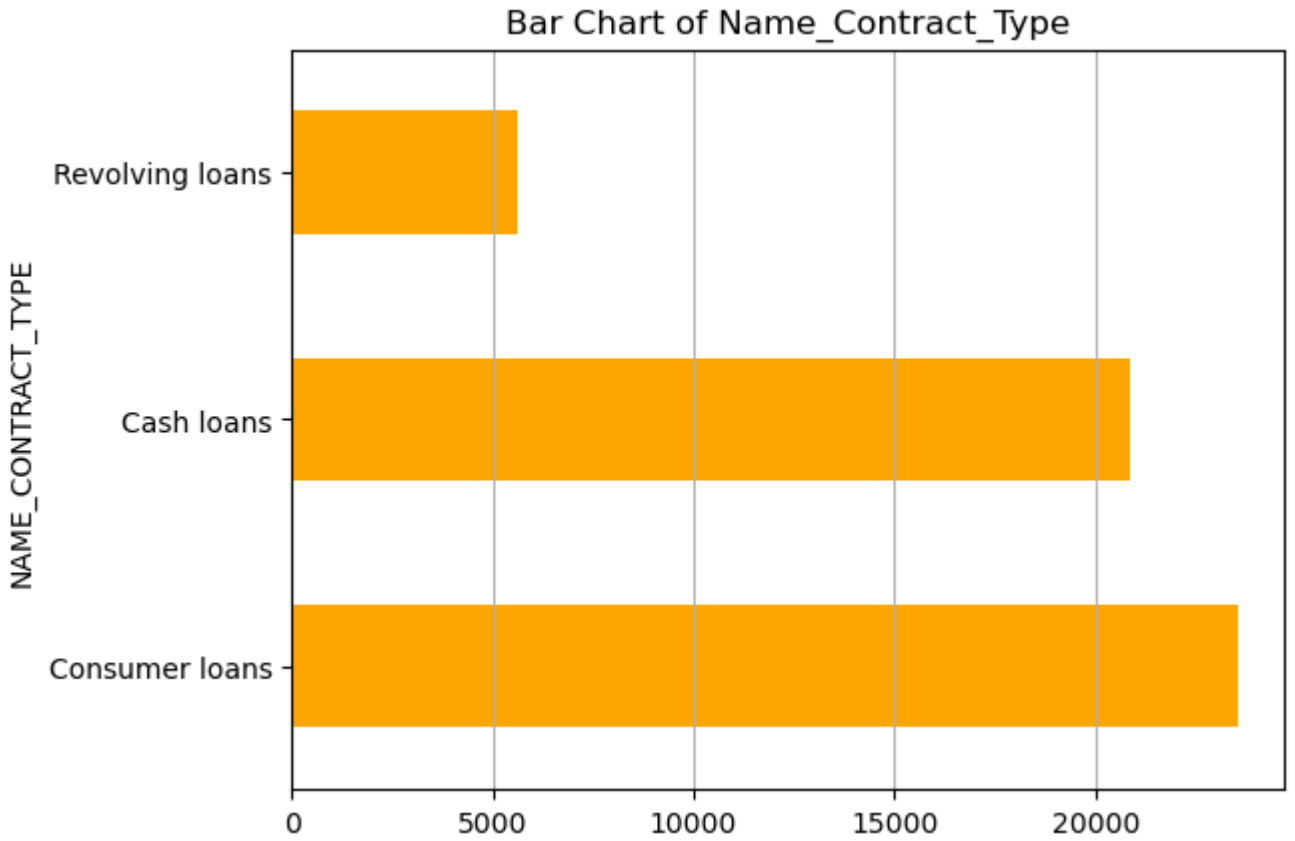
Univariate Analysis

- NAME_CONTRACT_TYPE

In [138]:
```
pre_app.NAME_CONTRACT_TYPE.value_counts()
```

Out[138]:
```
NAME_CONTRACT_TYPE
Consumer loans     23518
Cash loans         20856
Revolving loans     5625
Name: count, dtype: int64
```

In [139]:
```
use_barh_plot("NAME_CONTRACT_TYPE",pre_app)
```



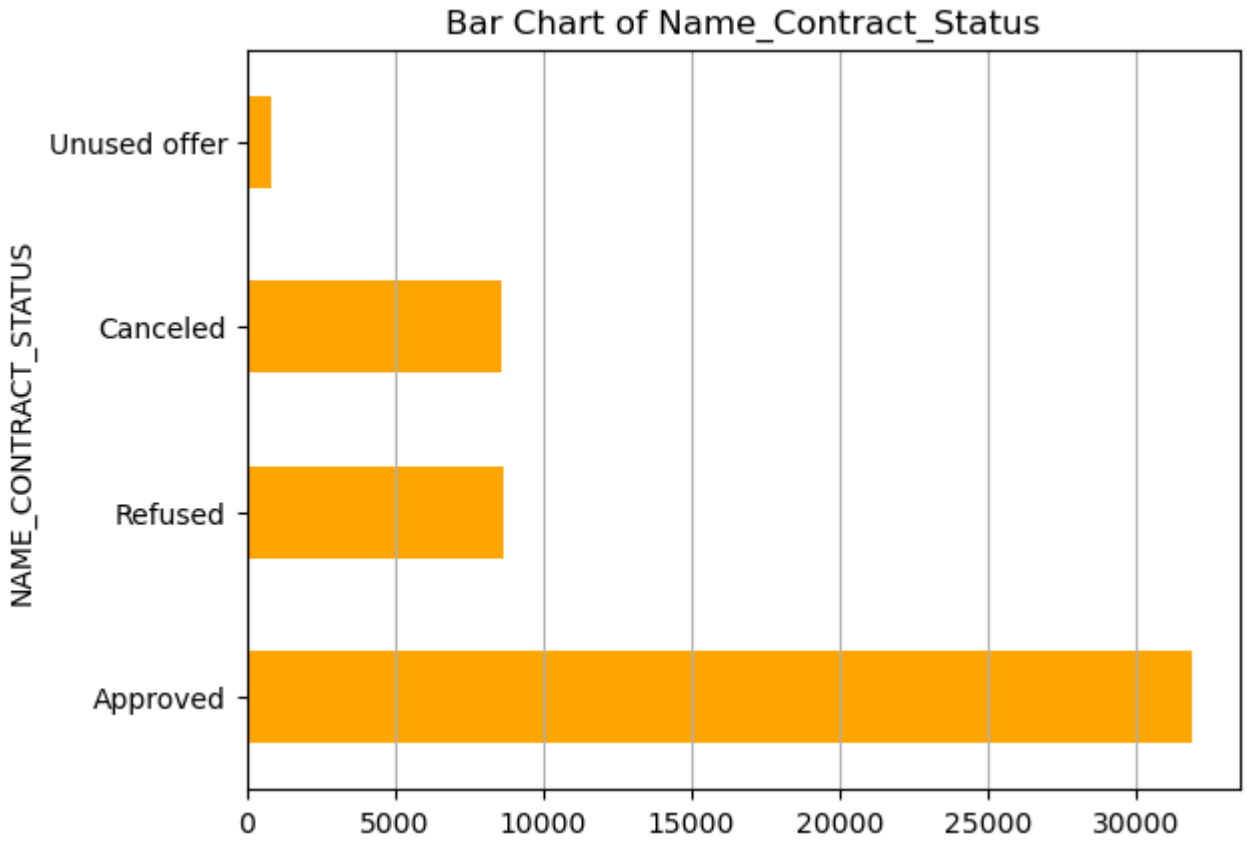
- Among previous customers Consumer loans were the highest applied for loan, followed by Cash loans and then Revolving loans

NAME_CONTRACT_STATUS

In [140]:
```
pre_app.NAME_CONTRACT_STATUS.value_counts(normalize=True)*100
```

Out[140]:
```
NAME_CONTRACT_STATUS
Approved        63.771275
Refused         17.320346
Canceled        17.190344
Unused offer     1.718034
Name: proportion, dtype: float64
```

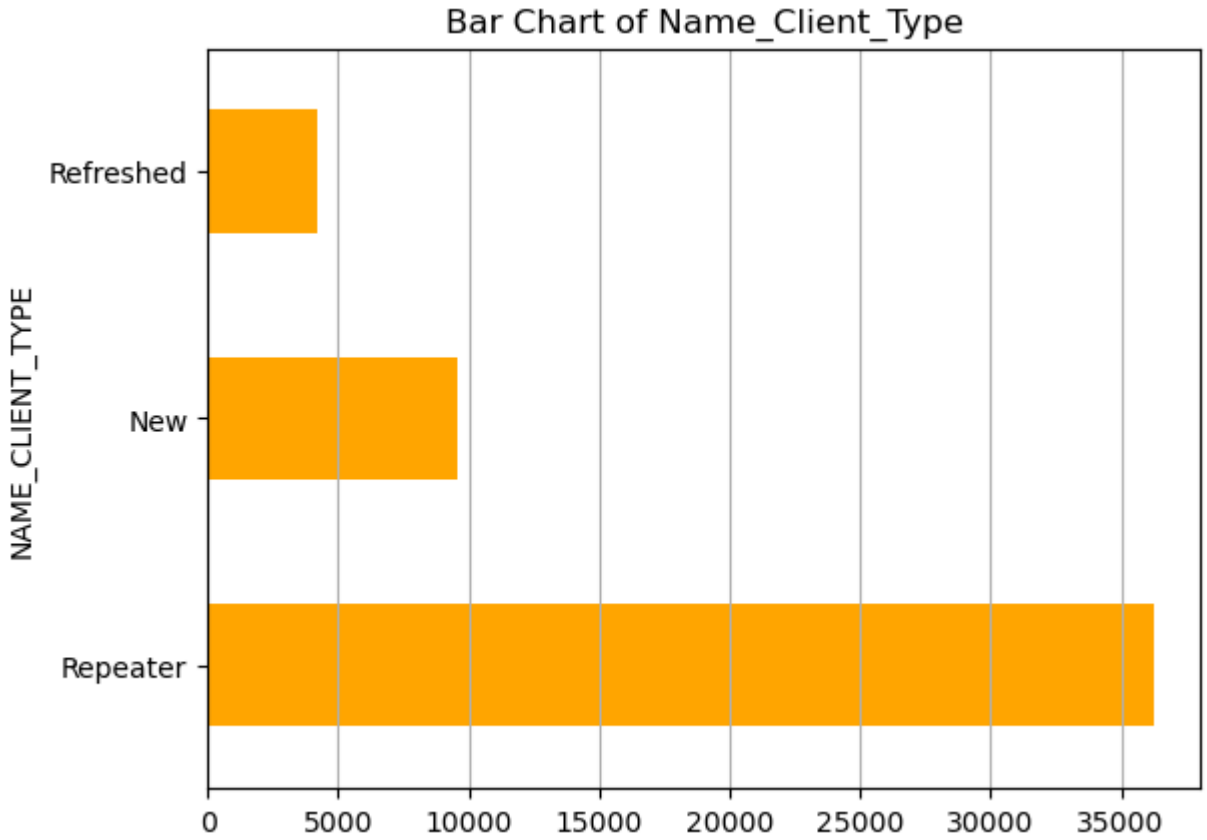
In [141]:
```
use_barh_plot("NAME_CONTRACT_STATUS",pre_app)
```



- Majority of the applicants got approved of getting the loans.

NAME_CLIENT_TYPE

In [142]:
```
use_barh_plot("NAME_CLIENT_TYPE",pre_app)
```

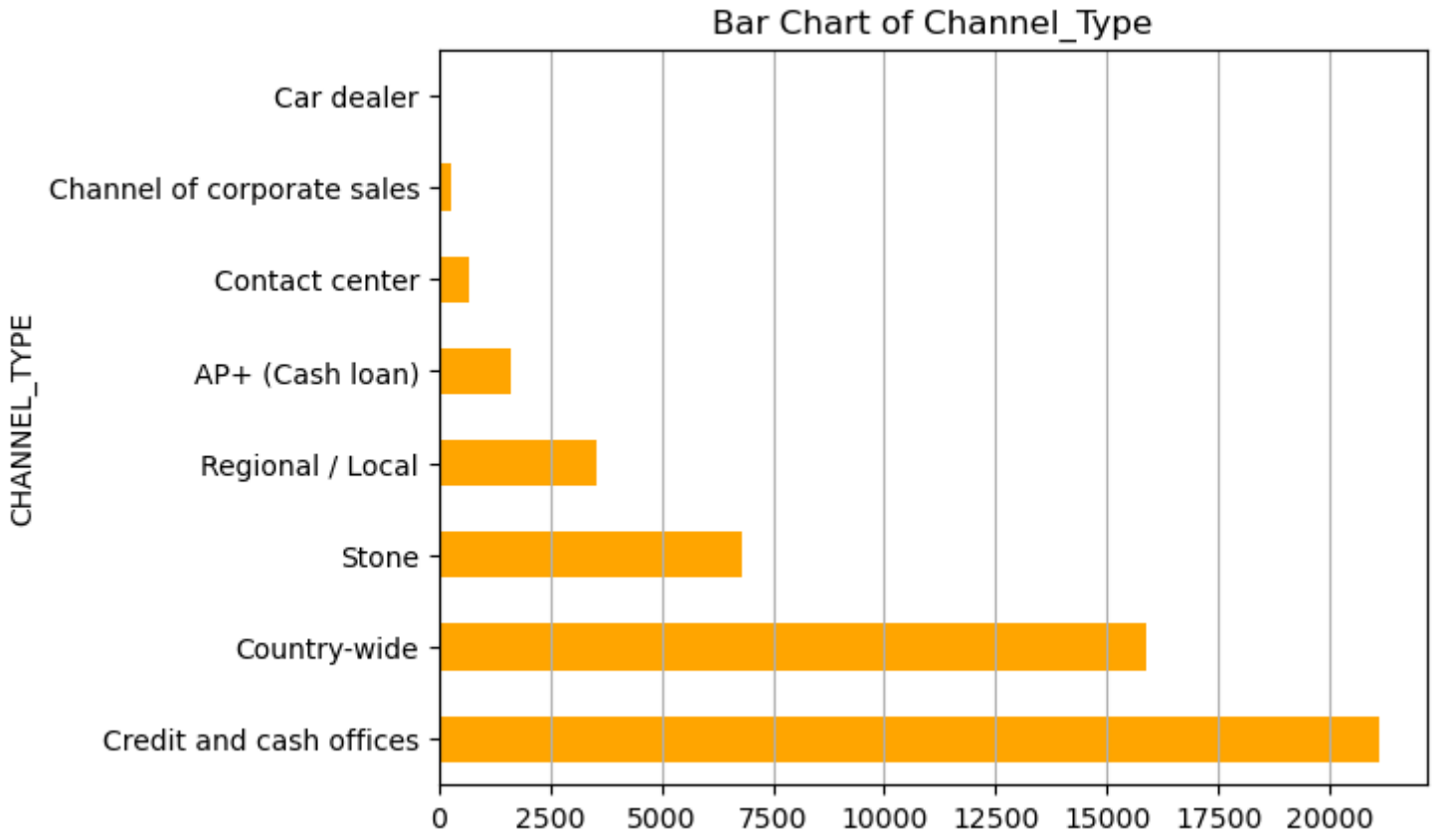


- Repeater clients are more in number who aaplied for loan, while New & Refreshed are very low.

In [143]:
```
pre_app.CHANNEL_TYPE.value_counts()
```

Out[143]:
```
CHANNEL_TYPE
Credit and cash offices       21119
Country-wide                  15900
Stone                          6811
Regional / Local               3563
AP+ (Cash loan)                1638
Contact center                  675
Channel of corporate sales      276
Car dealer                       17
Name: count, dtype: int64
```

In [144]:
```
use_barh_plot("CHANNEL_TYPE",pre_app)
```

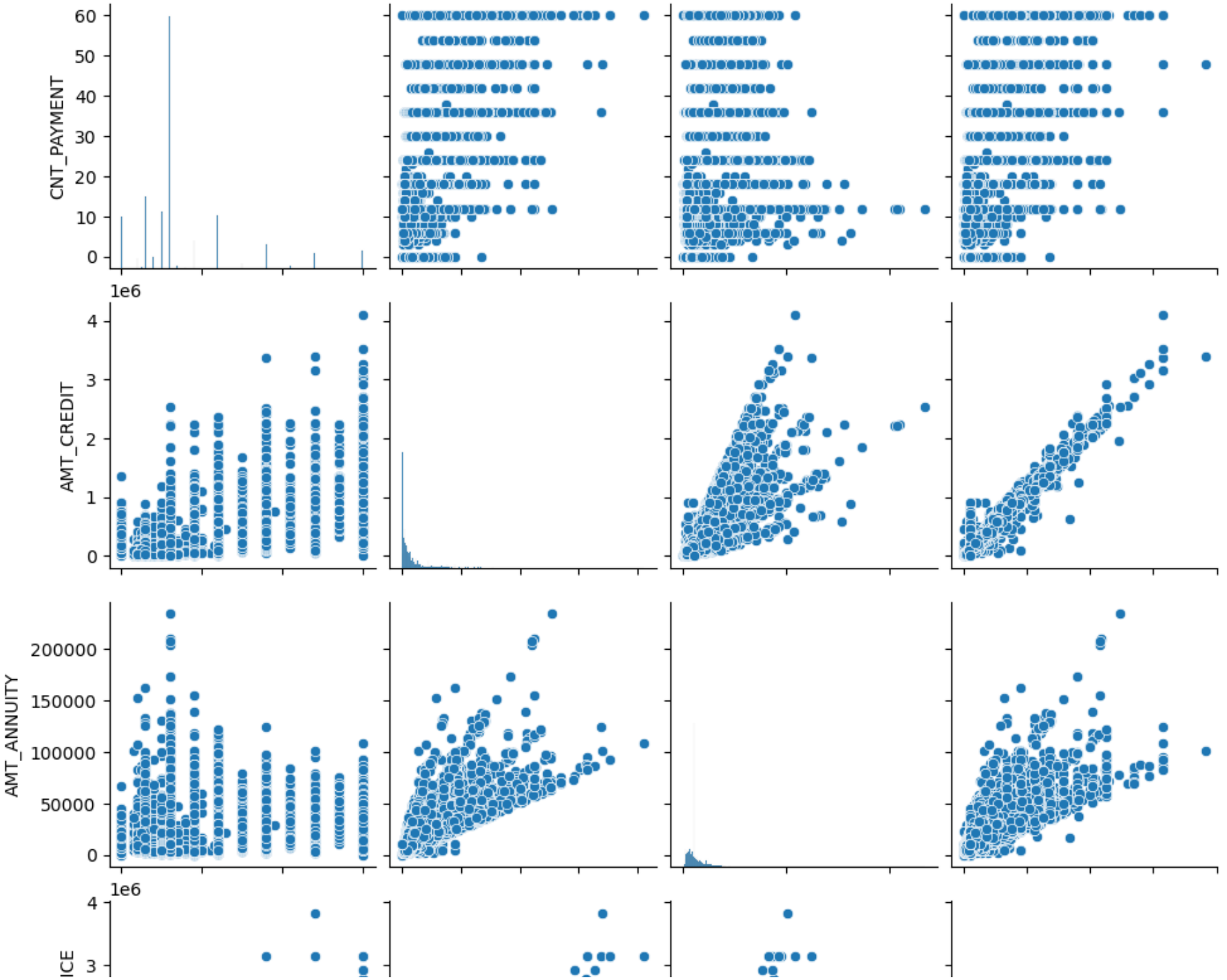


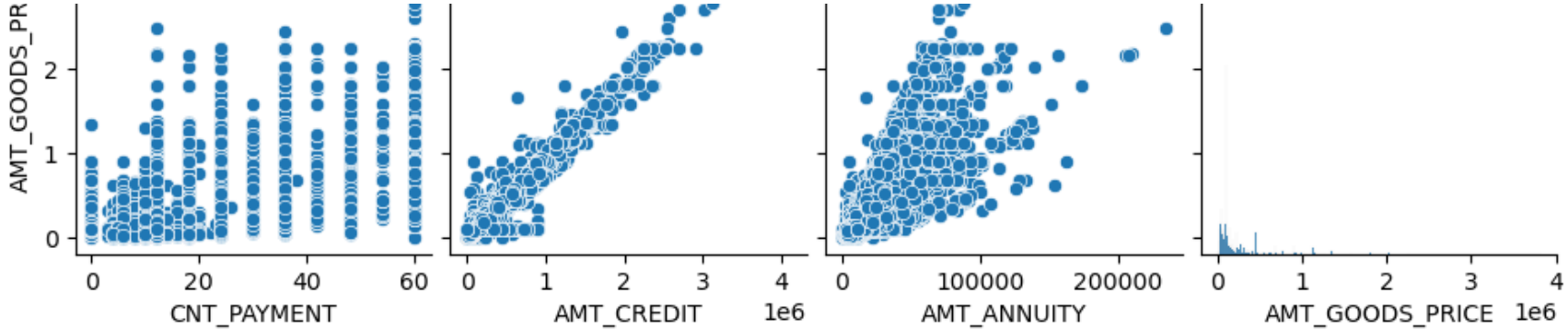
- Received majority of the loan applications from Credit and cash offices

Bivariate Analysis

In [145]:
```
plt.figure(figsize=[5,5])
sns.pairplot(data=pre_app,vars=["CNT_PAYMENT","AMT_CREDIT","AMT_ANNUITY","AMT_GOODS_PRICE"])
plt.show()
```

```
<Figure size 500x500 with 0 Axes>
```


CNT_PAYMENT
AMT_CREDIT
AMT_ANNUITY
ICE
1e6
1e6
0
10
20
30
40
50
60
0
1
2
3
4
0
50000
100000
150000
200000
0
3
4


AMT_GOODS_PR
2
1
0
0
20
40
60
CNT_PAYMENT
0
1
2
3
4
1e6
AMT_CREDIT
0
100000
200000
AMT_ANNUITY
0
1
2
3
4
1e6
AMT_GOODS_PRICE

In [ ]:

Heatmap

In [146]:
```
plt.figure(figsize=[5,5])
sns.heatmap(pre_app[["AMT_APPLICATION","AMT_ANNUITY","AMT_CREDIT","AMT_GOODS_PRICE","CNT_PAYMENT"]].corr(), annot=True
plt.show()
```

- In the Previous application data AMT_APPLICATION have high correlation with AMT_CREDIT & AMT_GOODS_PRICE
- Also AMT_CREDIT have high correlation with AMT_GOODS_PRICE

Merging the previous application data with the current application data

In [147]:
```
merge_df= pd.merge(new_app, pre_app, how='inner', on='SK_ID_CURR')
merge_df.head()
```

Out[147]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TO
0	100007	0	Cash loans	M	N	Y	0	1215
1	100009	0	Cash loans	F	Y	Y	1	1710
2	100012	0	Revolving loans	M	N	Y	0	1350
3	100026	0	Cash loans	F	N	N	1	4500
4	100027	0	Cash loans	F	N	Y	0	832

5 rows × 36 columns

In [148]:
```
merge_df.shape
```

Out[148]:
```
(6841, 36)
```

In [149]:
```
merge_df.info()
```

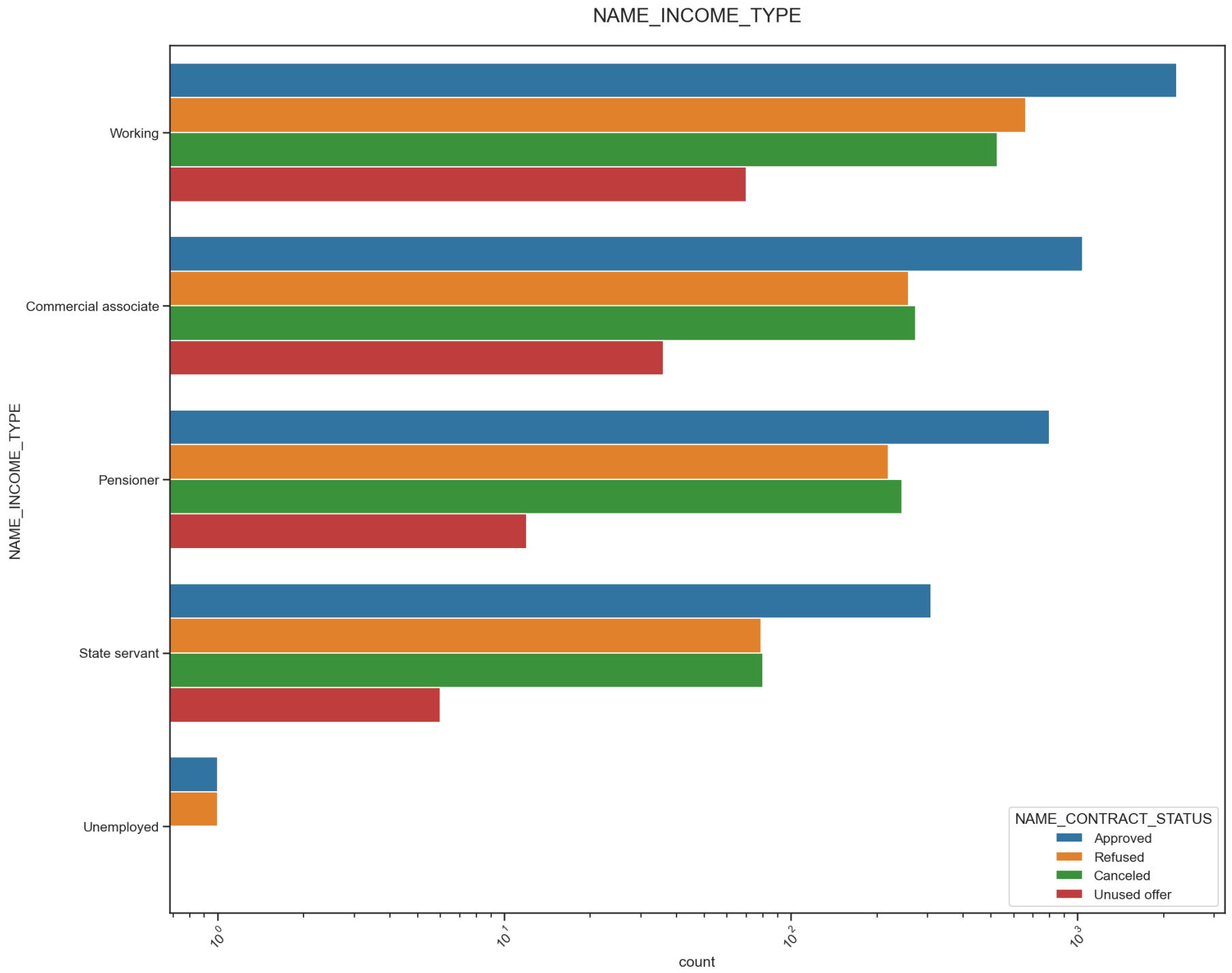
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6841 entries, 0 to 6840
Data columns (total 36 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   SK_ID_CURR                   6841 non-null   int64
 1   TARGET                       6841 non-null   int64
 2   NAME_CONTRACT_TYPE_x         6841 non-null   object
 3   CODE_GENDER                  6841 non-null   object
 4   FLAG_OWN_CAR                 6841 non-null   object
 5   FLAG_OWN_REALTY              6841 non-null   object
 6   CNT_CHILDREN                 6841 non-null   int64
 7   AMT_INCOME_TOTAL             6841 non-null   float64
 8   AMT_CREDIT_x                 6841 non-null   float64
 9   AMT_ANNUITY_x                6841 non-null   float64
 10  AMT_GOODS_PRICE_x            6841 non-null   float64
 11  NAME_INCOME_TYPE             6841 non-null   object
 12  NAME_EDUCATION_TYPE          6841 non-null   object
 13  NAME_FAMILY_STATUS           6841 non-null   object
 14  NAME_HOUSING_TYPE            6841 non-null   object
 15  REGION_POPULATION_RELATIVE   6841 non-null   float64
 16  OCCUPATION_TYPE              6841 non-null   object
 17  CNT_FAM_MEMBERS              6841 non-null   float64
 18  ORGANIZATION_TYPE            6841 non-null   object
 19  DAYS_LAST_PHONE_CHANGE       6841 non-null   float64
 20  AGE                          6841 non-null   int64
 21  YEARS_EMPLOYED               6841 non-null   int64
 22  INCOME_GROUP                 6841 non-null   category
 23  CREDIT_GROUP                 6841 non-null   category
 24  ANNUITY_GROUP                6841 non-null   category
 25  AGE_GROUP                    6841 non-null   category
 26  YEARS_EMP_GROUP              6841 non-null   category
 27  NAME_CONTRACT_TYPE_y         6841 non-null   object
 28  AMT_ANNUITY_y                6841 non-null   float64
 29  AMT_APPLICATION              6841 non-null   float64
 30  AMT_CREDIT_y                 6841 non-null   float64
 31  AMT_GOODS_PRICE_y            6841 non-null   float64
 32  NAME_CONTRACT_STATUS         6841 non-null   object
 33  NAME_CLIENT_TYPE             6841 non-null   object
 34  CHANNEL_TYPE                 6841 non-null   object
 35  CNT_PAYMENT                  6841 non-null   float64
```

```
dtypes: category(5), float64(12), int64(5), object(14)
memory usage: 1.7+ MB
```

Univariate Analysis on Merged data

Distribution of Previous Applciation Contract type & New Applcaition Income type

In [150]:
```
#distribution of NAME_CONTRACT_STATUS

sns.set_style('ticks')
sns.set_context('talk', rc={'lines.linewidth': 2})

plt.figure(figsize=(23,19))
plt.xscale('log')
plt.rcParams["axes.labelsize"] = 24
plt.rcParams['axes.titlesize'] = 24
plt.rcParams['axes.titlepad'] = 29
plt.xticks(rotation=45)


cash_loan_purpose = sns.countplot(data = merge_df, y='NAME_INCOME_TYPE',
                  order=merge_df['NAME_INCOME_TYPE'].value_counts().index,
                  hue = 'NAME_CONTRACT_STATUS')

plt.title('NAME_INCOME_TYPE')
plt.show()
```


NAME_INCOME_TYPE
Working
Commercial associate
Pensioner
State servant
Unemployed
NAME_INCOME_TYPE
count
NAME_CONTRACT_STATUS
Approved
Refused
Canceled
Unused offer

Inferences from above graph:

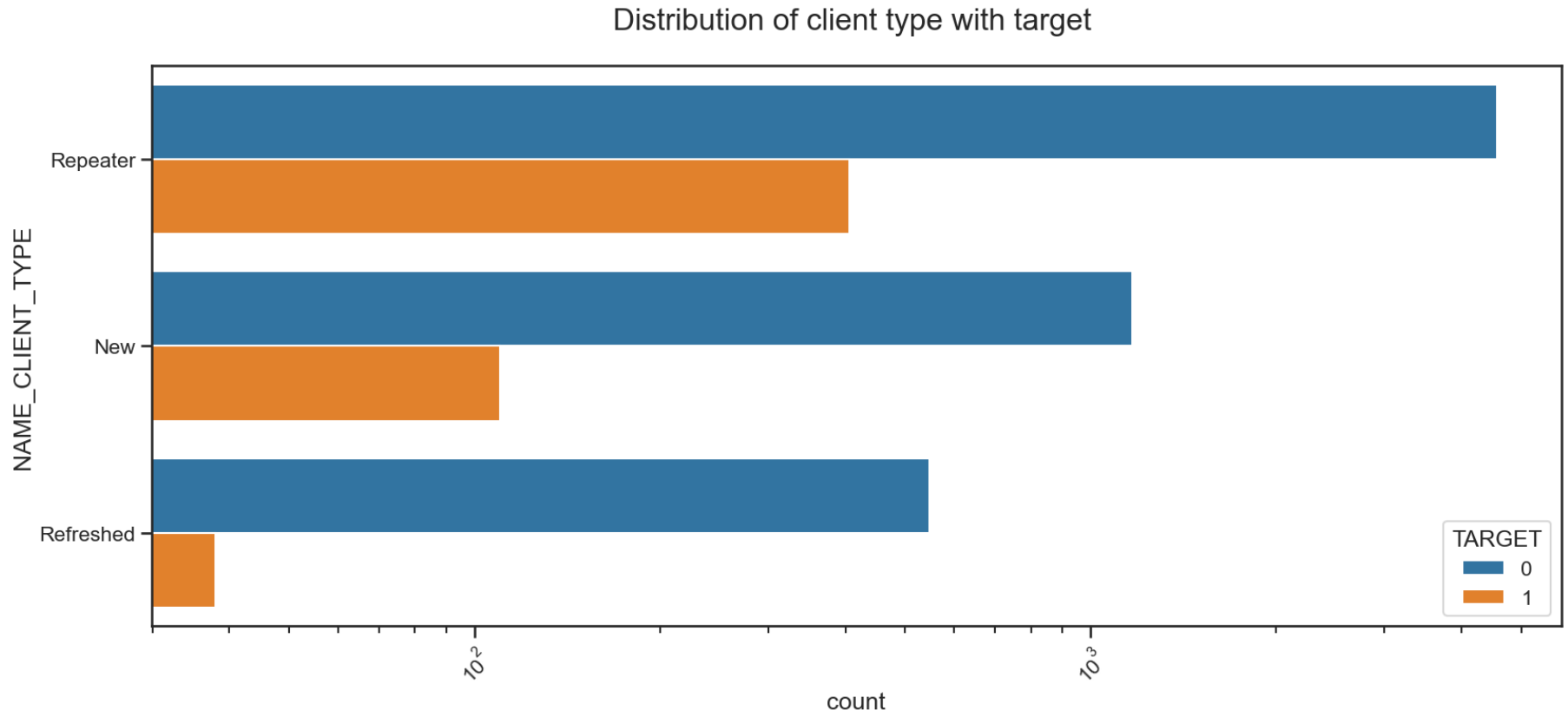
Majority of the loans are approved for Working category. Which shows Working people are large in numbers taking loans. Also the loans Cancelled & Refused for Working category having good proportion compare to Approved.

Commercial associate, Pensioners & Stat servant also got majority of the loans Approved, Compare to other categories in contract status.

The 'Unemployed' people got same proportion of Approved & Refused loans .

Distribution of Previous Applciation Client type with respect to TARGET

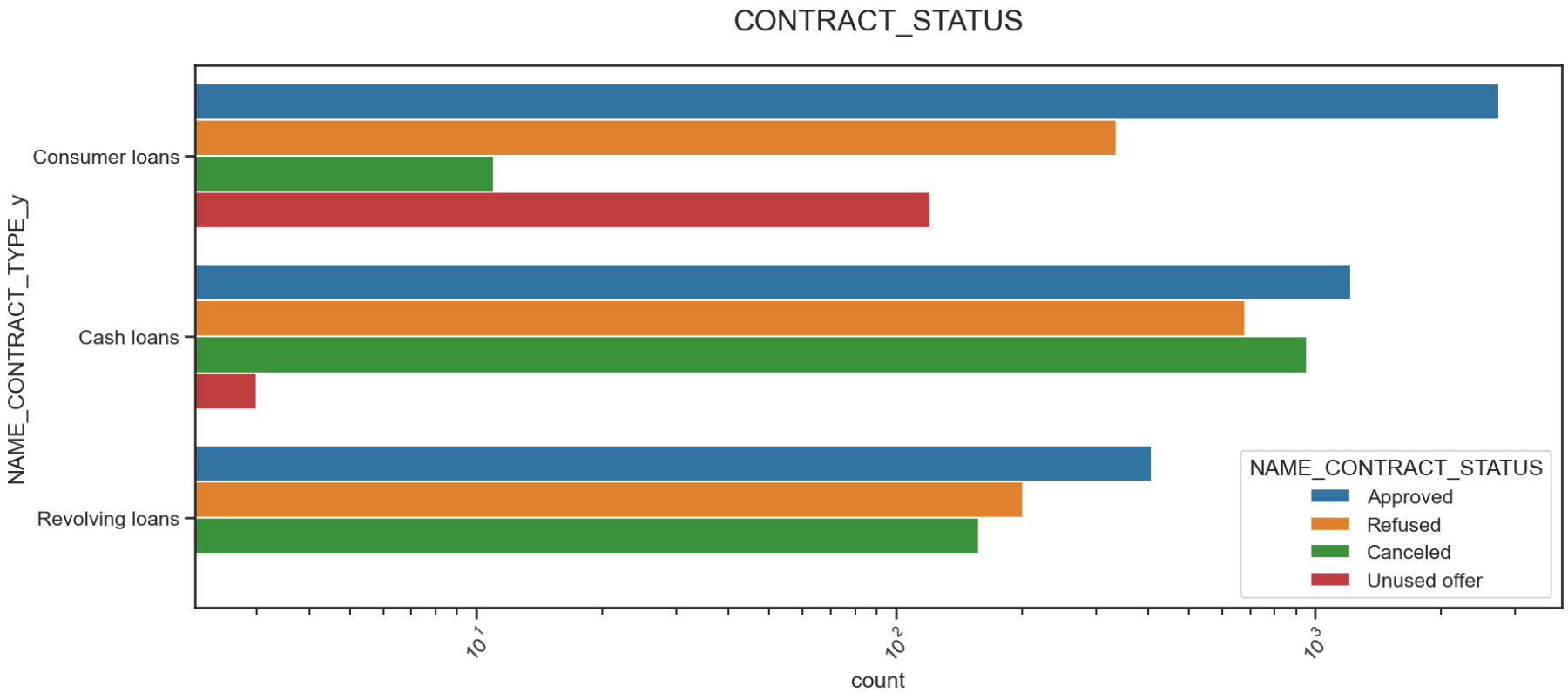
In [151]:
```
# Distribution of loan Status with Target=0, Target=1

sns.set_style('ticks')
sns.set_context('talk')


plt.figure(figsize=(20,8))
plt.rcParams["axes.labelsize"] = 19
plt.rcParams['axes.titlesize'] = 24
plt.rcParams['axes.titlepad'] = 29
plt.xticks(rotation=45)
plt.xscale('log')

plt.title('Distribution of client type with target ')
loan_purpose = sns.countplot(data = merge_df, y= 'NAME_CLIENT_TYPE',
                    order=merge_df['NAME_CLIENT_TYPE'].value_counts().index,
                    hue = 'TARGET')
plt.show()
```



- From the above graph Refreshed clients are less default in paying loans
- New Clients are less deafulter compare to repeater
- But on the other side Repeater clients are higher in paying loan on time.

Checking the Distribution of Previous applications Contract type & Contract status

In [152]:
```
#distribution of Previous Contract type & Contract status

sns.set_style('ticks')
sns.set_context('talk', rc={'lines.linewidth': 2})

plt.figure(figsize=(20,8))
plt.xscale('log')
plt.rcParams["axes.labelsize"] = 24
plt.rcParams['axes.titlesize'] = 24
plt.rcParams['axes.titlepad'] = 29
plt.xticks(rotation=45)


cash_loan_purpose = sns.countplot(data = merge_df, y='NAME_CONTRACT_TYPE_y',
                  order=merge_df['NAME_CONTRACT_TYPE_y'].value_counts().index,
                  hue = 'NAME_CONTRACT_STATUS')

plt.title('CONTRACT_STATUS')
plt.show()
```

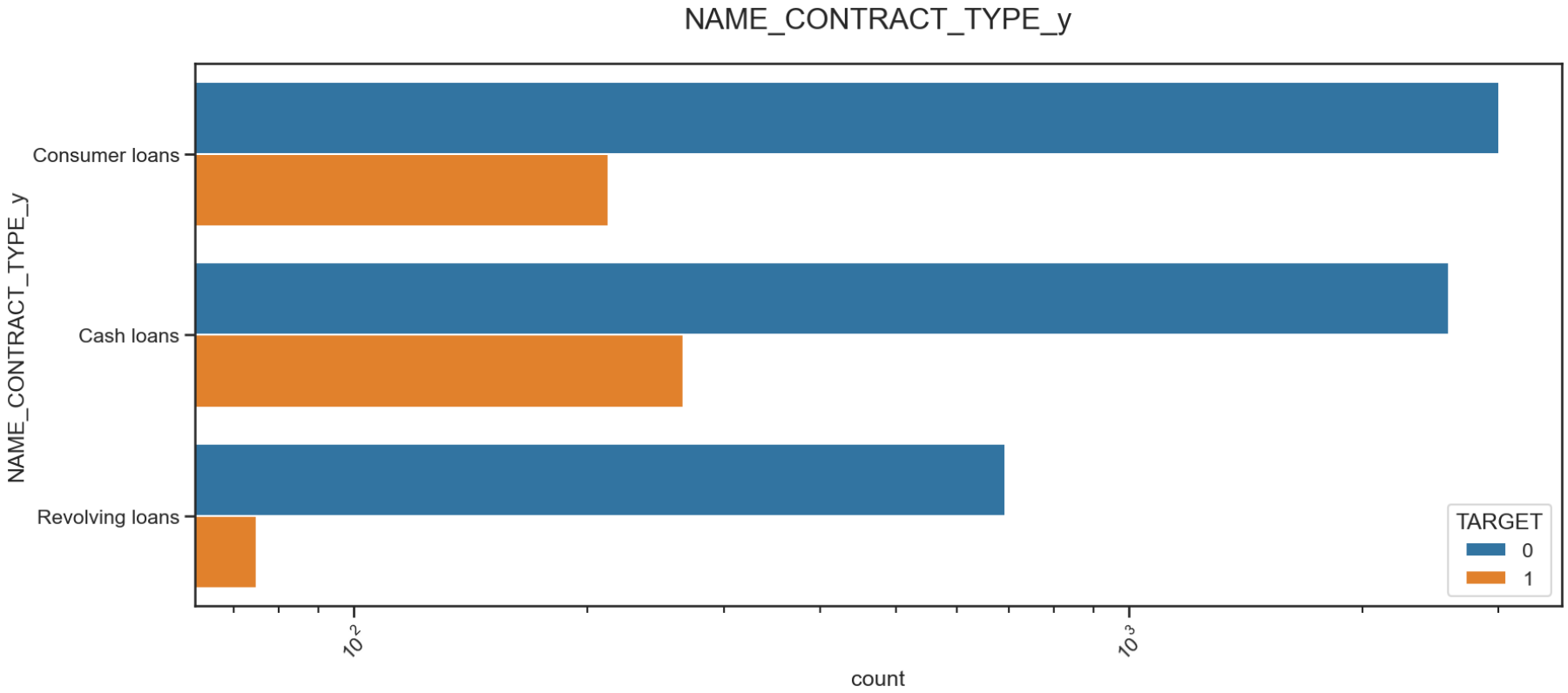


- The above graph shows in previous applications majority of the loans got approved for Consumer loans as well as they have less cancelled loans
- Although consumer loans have large unused offers as well.
- While the Cash loans & Revolving loans have high proprotion of 'Refused' & 'Canceled' compare to Consumer loans
- There is no unused offers in Revolving loans

Distribution of Previous Applciation Contract type with respect to TARGET

In [153]:

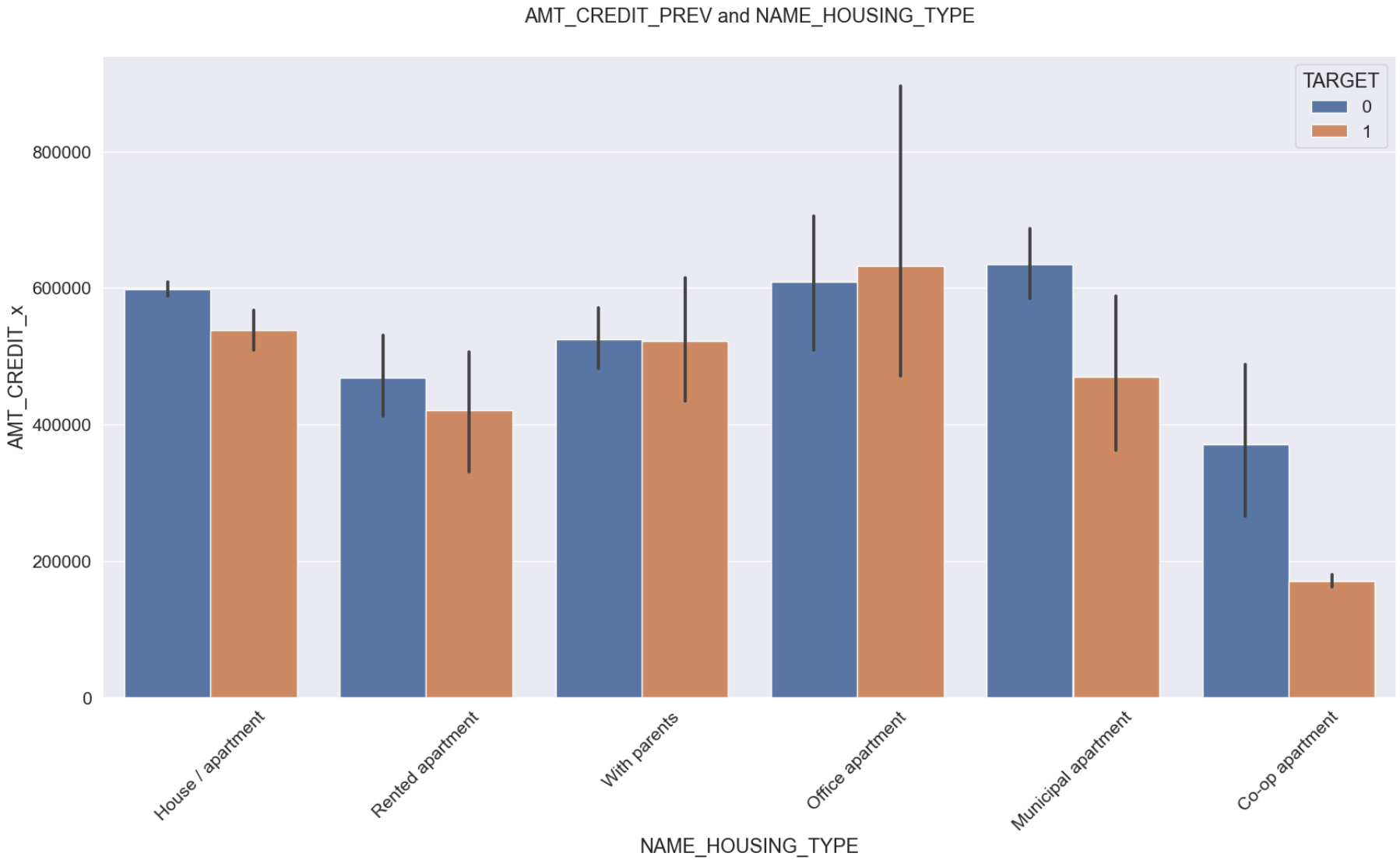
```
sns.set_style('ticks')
sns.set_context('talk', rc={'lines.linewidth': 2})

plt.figure(figsize=(20,8))
plt.xscale('log')
plt.rcParams["axes.labelsize"] = 24
plt.rcParams['axes.titlesize'] = 24
plt.rcParams['axes.titlepad'] = 29
plt.xticks(rotation=45)


cash_loan_purpose = sns.countplot(data = merge_df, y='NAME_CONTRACT_TYPE_y',
                  order=merge_df['NAME_CONTRACT_TYPE_y'].value_counts().index,
                  hue = 'TARGET')

plt.title('NAME_CONTRACT_TYPE_y')
plt.show()
```



- From the previous application in contract type, Consumer loans have less payment defaulters compare to cash loans.
- Also Consumer loans have high non-defaulters as well than cash loans

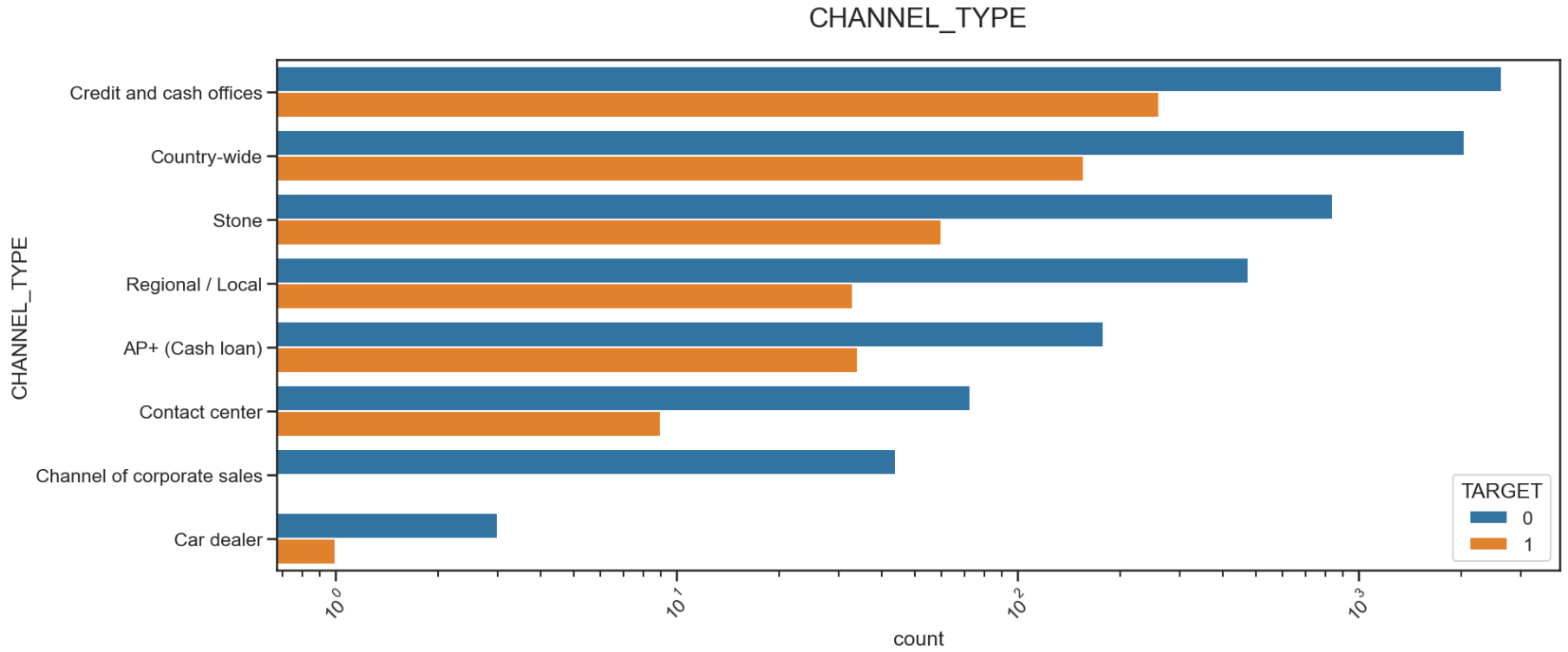
Checking the distribution of AMT_CREDIT from previous data & NAME_HOUSING_TYPE from new applications Wrt TARGET

In [125]:
```
# AMT_CREDIT_x and NAME_HOUSING_TYPE distribution using barplot on log scale
sns.set(font_scale=1.4)
plt.figure(figsize=(20,10))
plt.xticks(rotation=45)

sns.barplot(data = merge_df, y='AMT_CREDIT_x', x='NAME_HOUSING_TYPE', hue='TARGET')

plt.title('AMT_CREDIT_PREV and NAME_HOUSING_TYPE')
plt.show()
```



- The above graph shows people living in Co-op apartment are comparatively less defaulter, while the people living in Office apartment are high defaulter in paying loans

In [155]:
```
sns.set_style('ticks')
sns.set_context('talk', rc={'lines.linewidth': 2})

plt.figure(figsize=(20,8))
plt.xscale('log')
plt.rcParams["axes.labelsize"] = 24
plt.rcParams['axes.titlesize'] = 24
plt.rcParams['axes.titlepad'] = 29
plt.xticks(rotation=45)


cash_loan_purpose = sns.countplot(data = merge_df, y='CHANNEL_TYPE',
                  order=merge_df['CHANNEL_TYPE'].value_counts().index,
                  hue = 'TARGET')

plt.title('CHANNEL_TYPE')
plt.show()
```



- Channel of corporate sales have no defaulters in paying loans. So banks/financial institution can give loans to clients aquired from this channel
- Credit and cash offers have large number of non-defaulters as well as defaulters compare to other channels

In [ ]:

In [ ]: