In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import TomekLinks
#from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier



applications = pd.read_csv("C:/BI/CIND 820/Files/application_record.csv", encoding = 'u
tf-8')
credit_record = pd.read_csv("C:/BI/CIND 820/Files/credit_record.csv", encoding = 'utf-
8')

applications.head()
```

```
C:\Users\sarah\Anaconda3\lib\site-packages\pandas\compat\_optional.py:138:
UserWarning: Pandas requires version '2.7.0' or newer of 'numexpr' (versio
n '2.6.9' currently installed).
  warnings.warn(msg, UserWarning)
```

Out[1]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_I
0	5008804	M	Y	Y	0	
1	5008805	M	Y	Y	0	
2	5008806	M	Y	Y	0	
3	5008808	F	N	Y	0	
4	5008809	F	N	Y	0	

In [104]:

```
applications.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438557 entries, 0 to 438556
Data columns (total 18 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   ID                   438557 non-null  int64  
 1   CODE_GENDER          438557 non-null  object 
 2   FLAG_OWN_CAR         438557 non-null  object 
 3   FLAG_OWN_REALTY      438557 non-null  object 
 4   CNT_CHILDREN         438557 non-null  int64  
 5   AMT_INCOME_TOTAL     438557 non-null  float64
 6   NAME_INCOME_TYPE     438557 non-null  object 
 7   NAME_EDUCATION_TYPE  438557 non-null  object 
 8   NAME_FAMILY_STATUS   438557 non-null  object 
 9   NAME_HOUSING_TYPE    438557 non-null  object 
 10  DAYS_BIRTH           438557 non-null  int64  
 11  DAYS_EMPLOYED        438557 non-null  int64  
 12  FLAG_MOBIL           438557 non-null  int64  
 13  FLAG_WORK_PHONE      438557 non-null  int64  
 14  FLAG_PHONE           438557 non-null  int64  
 15  FLAG_EMAIL           438557 non-null  int64  
 16  OCCUPATION_TYPE      304354 non-null  object 
 17  CNT_FAM_MEMBERS      438557 non-null  float64
dtypes: float64(2), int64(8), object(8)
memory usage: 60.2+ MB
```

In [19]:

```
applications.describe()
```

Out[19]:

	ID	CNT_CHILDREN	AMT_INCOME_TOTAL	DAYS_BIRTH	DAYS_EMPLOYEI
count	4.385570e+05	438557.000000	4.385570e+05	438557.000000	438557.00000
mean	6.022176e+06	0.427390	1.875243e+05	-15997.904649	60563.67532
std	5.716370e+05	0.724882	1.100869e+05	4185.030007	138767.79964
min	5.008804e+06	0.000000	2.610000e+04	-25201.000000	-17531.00000
25%	5.609375e+06	0.000000	1.215000e+05	-19483.000000	-3103.00000
50%	6.047745e+06	0.000000	1.607805e+05	-15630.000000	-1467.00000
75%	6.456971e+06	1.000000	2.250000e+05	-12514.000000	-371.00000
max	7.999952e+06	19.000000	6.750000e+06	-7489.000000	365243.00000

In [4]:

```
credit_record.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 3 columns):
 #   Column          Non-Null Count    Dtype 
---  ------          --------------    ----- 
 0   ID              1048575 non-null  int64 
 1   MONTHS_BALANCE  1048575 non-null  int64 
 2   STATUS          1048575 non-null  object
dtypes: int64(2), object(1)
memory usage: 24.0+ MB
```

In [5]:

```
credit_record.describe()
```

Out[5]:

	ID	MONTHS_BALANCE
count	1.048575e+06	1.048575e+06
mean	5.068286e+06	-1.913700e+01
std	4.615058e+04	1.402350e+01
min	5.001711e+06	-6.000000e+01
25%	5.023644e+06	-2.900000e+01
50%	5.062104e+06	-1.700000e+01
75%	5.113856e+06	-7.000000e+00
max	5.150487e+06	0.000000e+00

In [105]:

```
applications.isnull().sum()
```

Out[105]:

```
ID                          0
CODE_GENDER                 0
FLAG_OWN_CAR                0
FLAG_OWN_REALTY             0
CNT_CHILDREN                0
AMT_INCOME_TOTAL            0
NAME_INCOME_TYPE            0
NAME_EDUCATION_TYPE         0
NAME_FAMILY_STATUS          0
NAME_HOUSING_TYPE           0
DAYS_BIRTH                  0
DAYS_EMPLOYED               0
FLAG_MOBIL                  0
FLAG_WORK_PHONE             0
FLAG_PHONE                  0
FLAG_EMAIL                  0
OCCUPATION_TYPE        134203
CNT_FAM_MEMBERS             0
dtype: int64
```

In [21]:

```
applications.FLAG_MOBIL.value_counts()
```

Out[21]:

```
1    438557
Name: FLAG_MOBIL, dtype: int64
```

In [22]:

```
credit_record.head(10)
```

Out[22]:

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C
5	5001712	-1	C
6	5001712	-2	C
7	5001712	-3	C
8	5001712	-4	C
9	5001712	-5	C

In [2]:

```
credit_record.STATUS.value_counts()
```

Out[2]:

```
C    442031
0    383120
X    209230
1     11090
5      1693
2       868
3       320
4       223
Name: STATUS, dtype: int64
```

In [3]:

```
#Convert status values to binary (2,3,4,5 --> 1 'Bad'; Else 0 'Good')

credit_record['STATUS'] = np.where((credit_record['STATUS'] == '2') |
                                   (credit_record['STATUS'] == '3' )|
                                   (credit_record['STATUS'] == '4' )|
                                   (credit_record['STATUS'] == '5'), 1, 0)
```

In [107]:

```
credit_record.head(10)
```

Out[107]:

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	0
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	0
5	5001712	-1	0
6	5001712	-2	0
7	5001712	-3	0
8	5001712	-4	0
9	5001712	-5	0

In [4]:

```
credit_record.STATUS.value_counts()
```

Out[4]:

```
0    1045471
1       3104
Name: STATUS, dtype: int64
```

In [ ]:

In [108]:

```
applications.FLAG_MOBIL.value_counts()
```

Out[108]:

```
1    438557
Name: FLAG_MOBIL, dtype: int64
```

In [ ]:

In [5]:

```
#Drop unwanted data
applications.drop( columns = ['FLAG_MOBIL'],inplace=True)
applications.dropna(subset=['OCCUPATION_TYPE'],inplace=True)
applications.drop_duplicates(subset=applications.columns[1:],inplace=True)
```

In [110]:

```
applications.describe()
```

Out[110]:

	ID	CNT_CHILDREN	AMT_INCOME_TOTAL	DAYS_BIRTH	DAYS_EMPLOYEI
count	6.260800e+04	62608.000000	6.260800e+04	62608.000000	62608.00000(
mean	5.909607e+06	0.507172	1.854948e+05	-14681.916416	-2427.51932
std	5.162990e+05	0.767497	1.075516e+05	3533.115374	2305.17068
min	5.008806e+06	0.000000	2.700000e+04	-24770.000000	-17531.00000(
25%	5.471167e+06	0.000000	1.215000e+05	-17368.000000	-3250.00000(
50%	5.956173e+06	0.000000	1.575000e+05	-14474.000000	-1723.50000(
75%	6.288576e+06	1.000000	2.250000e+05	-11784.000000	-817.00000(
max	7.995770e+06	19.000000	6.750000e+06	-7489.000000	-12.00000(

In [6]:

```
applications.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 62608 entries, 2 to 438553
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   62608 non-null  int64  
 1   CODE_GENDER          62608 non-null  object 
 2   FLAG_OWN_CAR         62608 non-null  object 
 3   FLAG_OWN_REALTY      62608 non-null  object 
 4   CNT_CHILDREN         62608 non-null  int64  
 5   AMT_INCOME_TOTAL     62608 non-null  float64
 6   NAME_INCOME_TYPE     62608 non-null  object 
 7   NAME_EDUCATION_TYPE  62608 non-null  object 
 8   NAME_FAMILY_STATUS   62608 non-null  object 
 9   NAME_HOUSING_TYPE    62608 non-null  object 
 10  DAYS_BIRTH           62608 non-null  int64  
 11  DAYS_EMPLOYED        62608 non-null  int64  
 12  FLAG_WORK_PHONE      62608 non-null  int64  
 13  FLAG_PHONE           62608 non-null  int64  
 14  FLAG_EMAIL           62608 non-null  int64  
 15  OCCUPATION_TYPE      62608 non-null  object 
 16  CNT_FAM_MEMBERS      62608 non-null  float64
dtypes: float64(2), int64(7), object(8)
memory usage: 8.6+ MB
```

In [ ]:

```

```

In [7]:

```
#creating a DF with the most recent month in each status for all applications

credit_classified = pd.DataFrame(pd.unique(credit_record.ID),columns = ['ID'])


credit_classified['Max_Mnth_Good'] = [max(credit_record[(credit_record.ID == i) & (credit_record.STATUS == 0)].MONTHS_BALANCE) for i in credit_classified.ID]
credit_classified['Max_Mnth_Bad'] = [max(credit_record[(credit_record.ID == i) & (credit_record.STATUS == 1)].MONTHS_BALANCE ,default=1) for i in credit_classified.ID]
```

In [112]:

```

```

In [8]:

```
#creating a DF with the most recent month in each status for all applications

credit_classified['Status'] = ["Good" if (credit_classified.Max_Mnth_Good.iloc[i] > credit_classified.Max_Mnth_Bad.iloc[i]) or (credit_classified.Max_Mnth_Bad.iloc[i] == 1) else "Bad" for i in range(len(credit_classified.ID))]
```

In [9]:

```
credit_classified.Status.value_counts()
```

Out[9]:

```
Good    45873
Bad       112
Name: Status, dtype: int64
```

In [116]:

```
credit_classified
```

Out[116]:

	ID	Max_Mnth_Good	Max_Mnth_Bad	Status
0	5001711	0	1	Good
1	5001712	0	1	Good
2	5001713	0	1	Good
3	5001714	0	1	Good
4	5001715	0	1	Good
...	...	...	...	...
45980	5150482	-11	1	Good
45981	5150483	0	1	Good
45982	5150484	0	1	Good
45983	5150485	0	1	Good
45984	5150487	0	1	Good

45985 rows × 4 columns

In [117]:

```
credit_classified[credit_classified["Status"]=="Bad"]
```

Out[117]:

	ID	Max_Mnth_Good	Max_Mnth_Bad	Status
2450	5004891	-1	0	Bad
2695	5005205	-3	0	Bad
3795	5009524	-2	0	Bad
3970	5009744	-15	0	Bad
3974	5009749	-11	0	Bad
...	...	...	...	...
45103	5149188	-30	-19	Bad
45105	5149190	-10	0	Bad
45107	5149192	-54	-43	Bad
45621	5149828	-3	0	Bad
45802	5150049	-1	0	Bad

112 rows × 4 columns

In [118]:

```
#Merging all data
```

In [14]:

```
merged_data = pd.merge(applications, credit_classified, how = "inner" , on='ID')
```

In [15]:

```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6715 entries, 0 to 6714
Data columns (total 20 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   6715 non-null   int64  
 1   CODE_GENDER          6715 non-null   object 
 2   FLAG_OWN_CAR         6715 non-null   object 
 3   FLAG_OWN_REALTY      6715 non-null   object 
 4   CNT_CHILDREN         6715 non-null   int64  
 5   AMT_INCOME_TOTAL     6715 non-null   float64
 6   NAME_INCOME_TYPE     6715 non-null   object 
 7   NAME_EDUCATION_TYPE  6715 non-null   object 
 8   NAME_FAMILY_STATUS   6715 non-null   object 
 9   NAME_HOUSING_TYPE    6715 non-null   object 
 10  DAYS_BIRTH           6715 non-null   int64  
 11  DAYS_EMPLOYED        6715 non-null   int64  
 12  FLAG_WORK_PHONE      6715 non-null   int64  
 13  FLAG_PHONE           6715 non-null   int64  
 14  FLAG_EMAIL           6715 non-null   int64  
 15  OCCUPATION_TYPE      6715 non-null   object 
 16  CNT_FAM_MEMBERS      6715 non-null   float64
 17  Max_Mnth_Good        6715 non-null   int64  
 18  Max_Mnth_Bad         6715 non-null   int64  
 19  Status               6715 non-null   object 
dtypes: float64(2), int64(9), object(9)
memory usage: 1.1+ MB
```

In [16]:

```
merged_data.describe()
```

Out[16]:

	ID	CNT_CHILDREN	AMT_INCOME_TOTAL	DAYS_BIRTH	DAYS_EMPLOYE
count	6.715000e+03	6715.000000	6.715000e+03	6715.000000	6715.00000
mean	5.076510e+06	0.508116	1.896064e+05	-14769.037081	-2485.38629
std	4.091949e+04	0.819438	1.022247e+05	3529.228015	2299.57327
min	5.008806e+06	0.000000	2.700000e+04	-24611.000000	-15713.00000
25%	5.036962e+06	0.000000	1.260000e+05	-17448.000000	-3350.50000
50%	5.078898e+06	0.000000	1.665000e+05	-14548.000000	-1788.00000
75%	5.113032e+06	1.000000	2.250000e+05	-11919.500000	-859.00000
max	5.150467e+06	19.000000	1.575000e+06	-7489.000000	-17.00000

In [19]:

```
#Handling Outliers
#the function to define the whiskers
def drop_outlier(x):
    q75,q25 = np.percentile(merged_data[x],[75,25])
    intr_qr = q75-q25
    mx = q75+(1.5*intr_qr)
    mn = q25-(1.5*intr_qr)
    return mx,mn
```

In [106]:

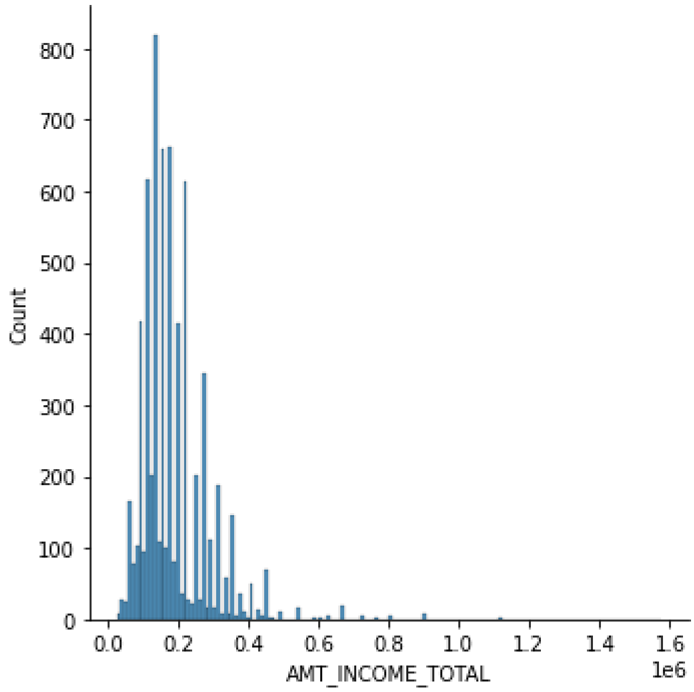
```
#pip install seaborn==0.11.0
```

In [17]:

```
sns.displot(merged_data, x="AMT_INCOME_TOTAL")
```

Out[17]:

```
<seaborn.axisgrid.FacetGrid at 0x1e9c80694a8>
```

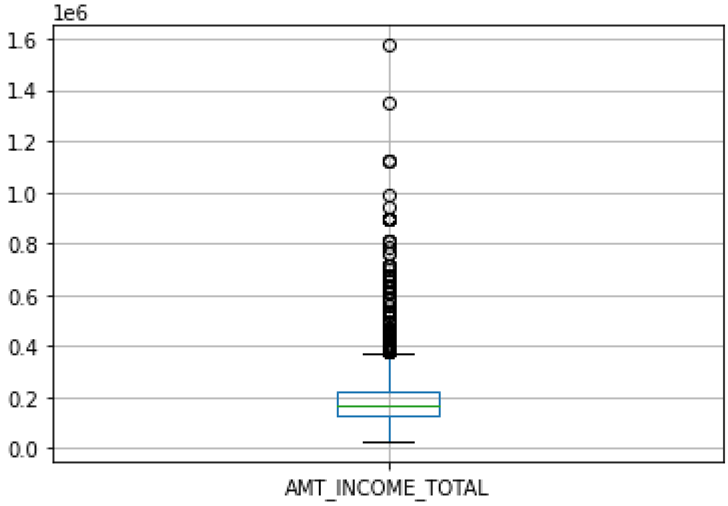


In [78]:

```
merged_data.boxplot('AMT_INCOME_TOTAL')
```

Out[78]:

<AxesSubplot:>



In [ ]:

```

```

In [20]:

```
mx,mn = drop_outlier('AMT_INCOME_TOTAL')
mx,mn
```

Out[20]:

(373500.0, -22500.0)

In [21]:

```
median=merged_data['AMT_INCOME_TOTAL'].median()
median
outliers = (merged_data['AMT_INCOME_TOTAL'] > mx) | (merged_data['AMT_INCOME_TOTAL'] < mn)
outliers
merged_data['AMT_INCOME_TOTAL'].mask(outliers, other=median,inplace=True)
merged_data['AMT_INCOME_TOTAL'].head(10)
```

Out[21]:

```
0    112500.0
1    270000.0
2    270000.0
3    135000.0
4    130500.0
5    157500.0
6    270000.0
7    166500.0
8    112500.0
9    135000.0
Name: AMT_INCOME_TOTAL, dtype: float64
```

In [22]:

```
len(merged_data.index)
```

Out[22]:

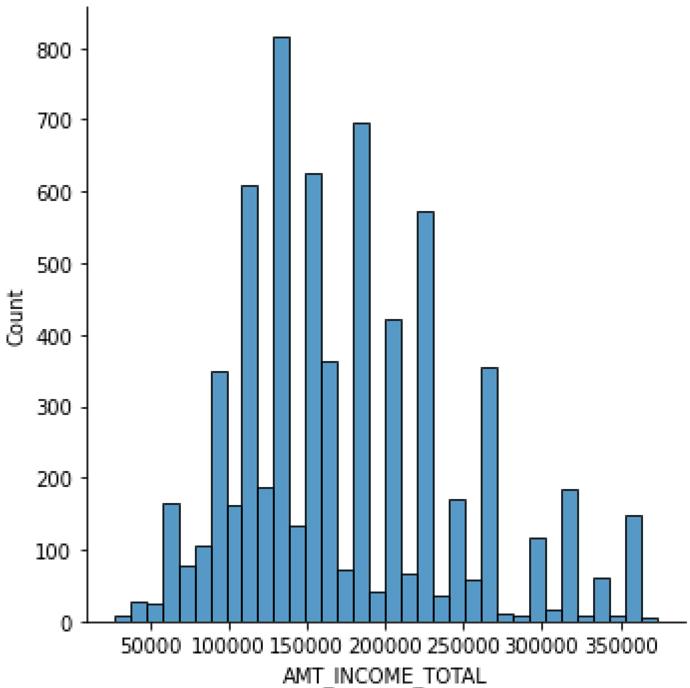
```
6715
```

In [23]:

```
sns.displot(merged_data, x="AMT_INCOME_TOTAL")
```

Out[23]:

```
<seaborn.axisgrid.FacetGrid at 0x1e9c90fe198>
```

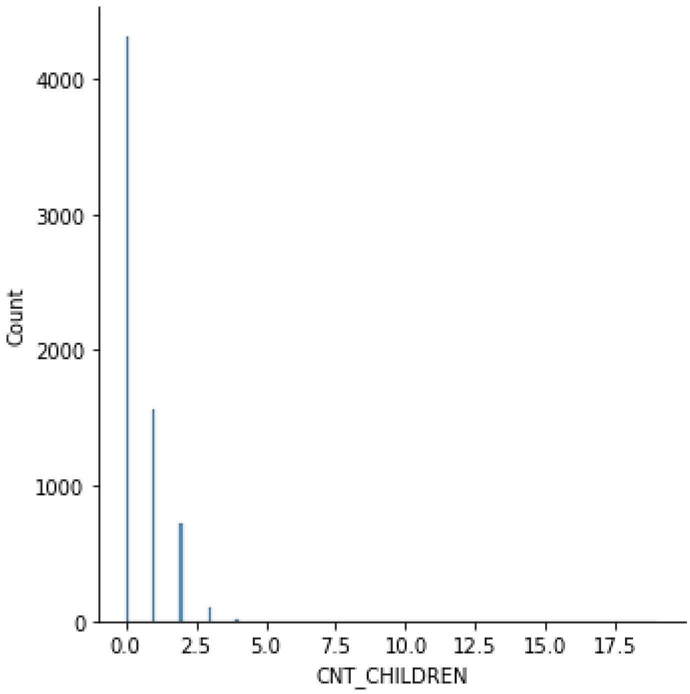


In [24]:

```
sns.displot(merged_data, x="CNT_CHILDREN")
```

Out[24]:

<seaborn.axisgrid.FacetGrid at 0x1e9c9156550>

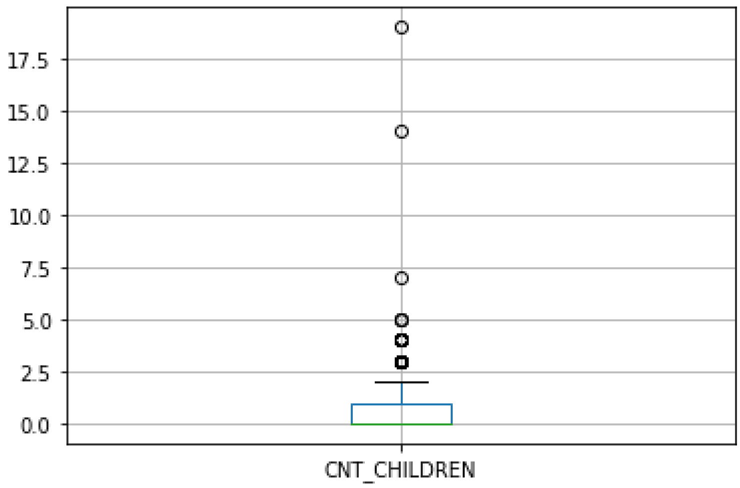


In [25]:

```
merged_data.boxplot('CNT_CHILDREN')
```

Out[25]:

<AxesSubplot:>



In [26]:

```
mx,mn = drop_outlier('CNT_CHILDREN')
mx,mn
```

Out[26]:

(2.5, -1.5)

In [27]:

```
median=merged_data['CNT_CHILDREN'].median()
median
outliers = (merged_data['CNT_CHILDREN'] > mx) | (merged_data['CNT_CHILDREN'] < mn)
outliers
merged_data['CNT_CHILDREN'].mask(outliers, other=median,inplace=True)
merged_data['CNT_CHILDREN'].head(10)
```

Out[27]:

```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    1
8    0
9    2
Name: CNT_CHILDREN, dtype: int64
```

In [28]:

```
#merged_data.drop(merged_data[merged_data.CNT_CHILDREN > 3].index,inplace=True)
len(merged_data.index)
```

Out[28]:

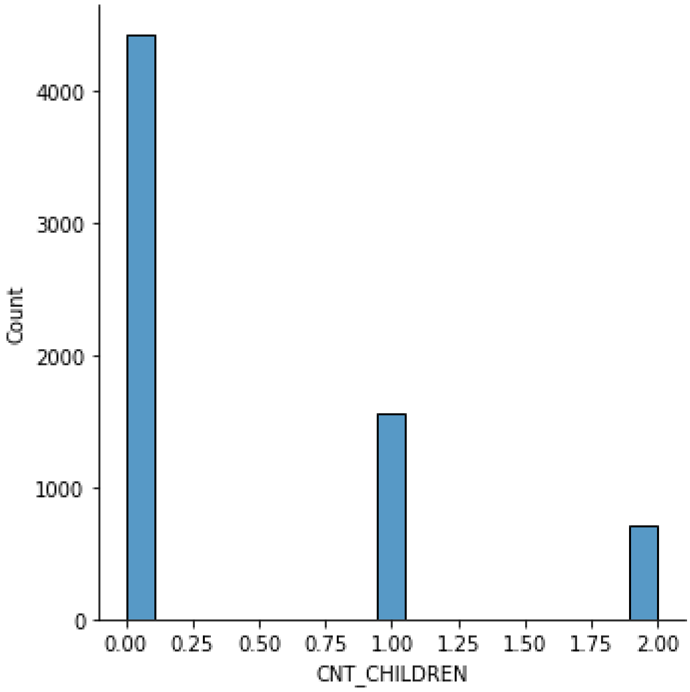
```
6715
```

In [133]:

```
#"Customer Distribution by number of children
sns.displot(merged_data, x="CNT_CHILDREN")
```

Out[133]:

```
<seaborn.axisgrid.FacetGrid at 0x263a5890f28>
```

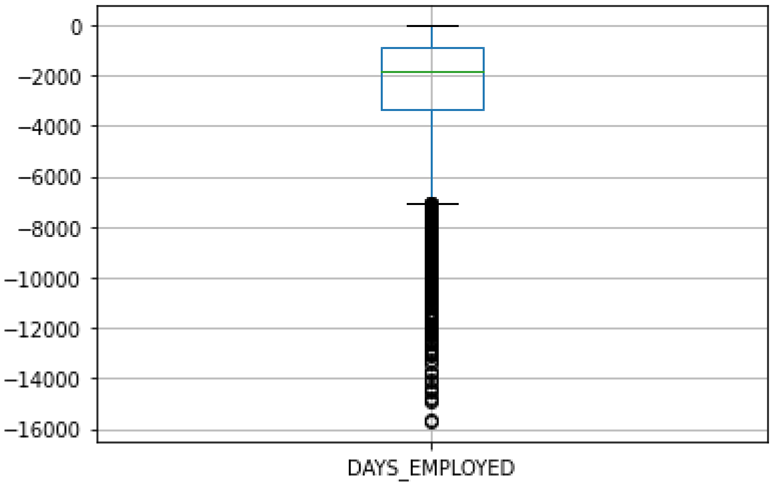


In [29]:

```
merged_data.boxplot('DAYS_EMPLOYED')
```

Out[29]:

```
<AxesSubplot:>
```



In [30]:

```
mx,mn = drop_outlier('DAYS_EMPLOYED')
mx,mn
```

Out[30]:

```
(2878.25, -7087.75)
```

In [ ]:

```

```

In [31]:

```
median=merged_data['DAYS_EMPLOYED'].median()
median
outliers = (merged_data['DAYS_EMPLOYED'] > mx) | (merged_data['DAYS_EMPLOYED'] < mn)
outliers
merged_data['DAYS_EMPLOYED'].mask(outliers, other=median,inplace=True)
merged_data['DAYS_EMPLOYED'].head(10)
```

Out[31]:

```
0   -1134
1   -3051
2    -769
3   -1194
4   -1103
5   -1469
6   -1163
7   -2016
8   -4450
9   -3173
Name: DAYS_EMPLOYED, dtype: int64
```

In [138]:

```
#merged_data.drop(merged_data[merged_data.DAYS_EMPLOYED > mx].index,inplace=True)
#merged_data.drop(merged_data[merged_data.DAYS_EMPLOYED < mn].index,inplace=True)
len(merged_data.index)
```

Out[138]:

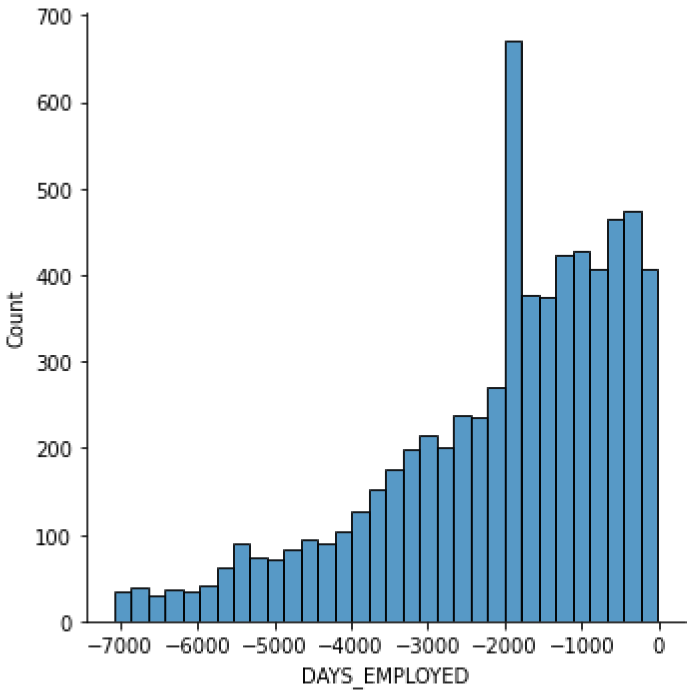
```
6715
```

In [32]:

```
sns.displot(merged_data, x="DAYS_EMPLOYED")
```

Out[32]:

<seaborn.axisgrid.FacetGrid at 0x1e9cfeba320>



In [33]:

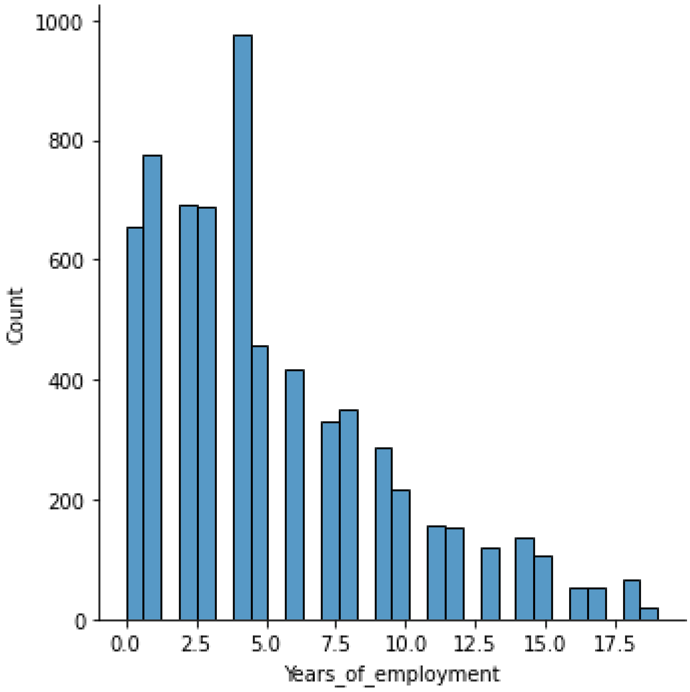
```
merged_data['Years_of_employment']= -(merged_data['DAYS_EMPLOYED'])//365

merged_data.drop( columns = ['DAYS_EMPLOYED'],inplace=True)
```

In [34]:

```
sns.displot(merged_data, x='Years_of_employment')
```

Out[34]:

```
<seaborn.axisgrid.FacetGrid at 0x1e9cffc3e48>
```



In [35]:

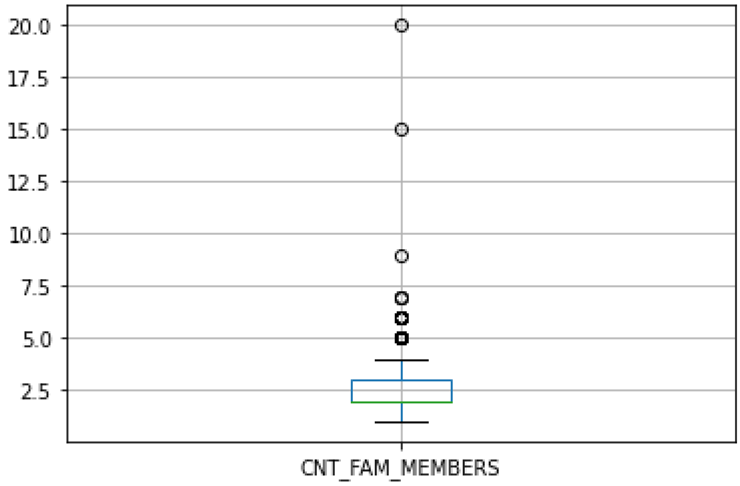
```
merged_data['Age']= -(merged_data['DAYS_BIRTH'])//365

merged_data.drop( columns = ['DAYS_BIRTH'],inplace=True)
```

In [36]:

```
merged_data.boxplot('CNT_FAM_MEMBERS')
```

Out[36]:

<AxesSubplot:>



In [37]:

```
mx,mn = drop_outlier('CNT_FAM_MEMBERS')
mx,mn
```

Out[37]:

(4.5, 0.5)

In [38]:

```
median=merged_data['CNT_FAM_MEMBERS'].median()
median
outliers = (merged_data['CNT_FAM_MEMBERS'] > mx) | (merged_data['CNT_FAM_MEMBERS'] < mn
)
outliers
merged_data['CNT_FAM_MEMBERS'].mask(outliers, other=median,inplace=True)
merged_data['CNT_FAM_MEMBERS'].head(10)
```

Out[38]:

```
0    2.0
1    1.0
2    2.0
3    2.0
4    2.0
5    2.0
6    2.0
7    3.0
8    2.0
9    4.0
Name: CNT_FAM_MEMBERS, dtype: float64
```

In [39]:

```
#merged_data.drop(merged_data[merged_data.CNT_FAM_MEMBERS > 6].index,inplace=True)
len(merged_data.index)
```

Out[39]:

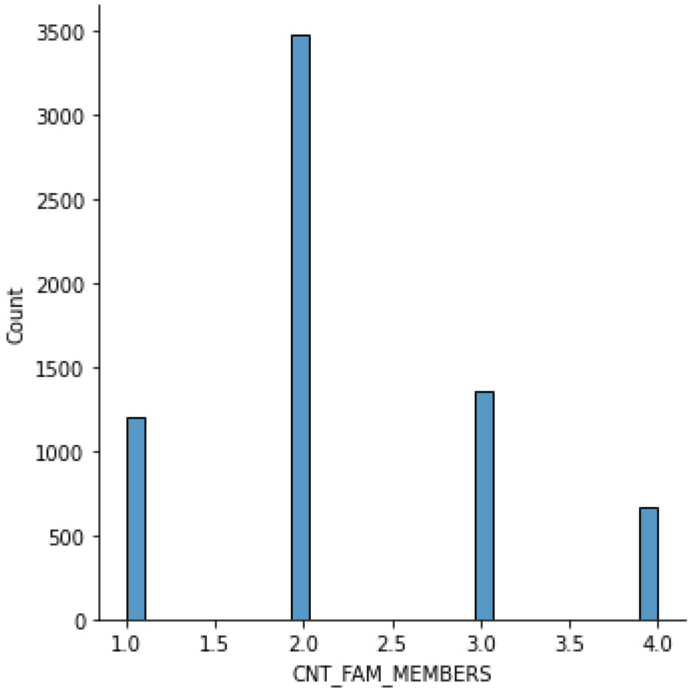
```
6715
```

In [40]:

```
#"Customer Distribution by family members

sns.displot(merged_data, x="CNT_FAM_MEMBERS")
```

Out[40]:

<seaborn.axisgrid.FacetGrid at 0x1e9d0177898>



In [41]:

```
merged_data.info()
```

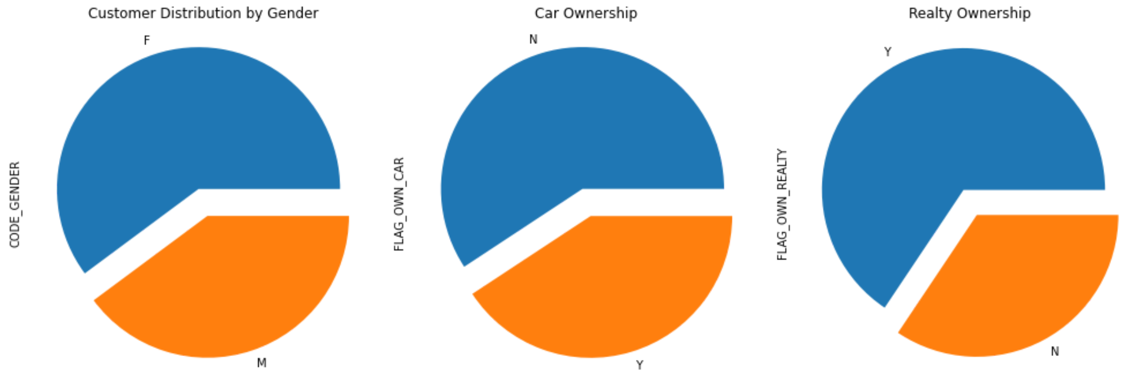
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6715 entries, 0 to 6714
Data columns (total 20 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   6715 non-null   int64  
 1   CODE_GENDER          6715 non-null   object 
 2   FLAG_OWN_CAR         6715 non-null   object 
 3   FLAG_OWN_REALTY      6715 non-null   object 
 4   CNT_CHILDREN         6715 non-null   int64  
 5   AMT_INCOME_TOTAL     6715 non-null   float64
 6   NAME_INCOME_TYPE     6715 non-null   object 
 7   NAME_EDUCATION_TYPE  6715 non-null   object 
 8   NAME_FAMILY_STATUS   6715 non-null   object 
 9   NAME_HOUSING_TYPE    6715 non-null   object 
 10  FLAG_WORK_PHONE      6715 non-null   int64  
 11  FLAG_PHONE           6715 non-null   int64  
 12  FLAG_EMAIL           6715 non-null   int64  
 13  OCCUPATION_TYPE      6715 non-null   object 
 14  CNT_FAM_MEMBERS      6715 non-null   float64
 15  Max_Mnth_Good        6715 non-null   int64  
 16  Max_Mnth_Bad         6715 non-null   int64  
 17  Status               6715 non-null   object 
 18  Years_of_employment  6715 non-null   int64  
 19  Age                  6715 non-null   int64  
dtypes: float64(2), int64(9), object(9)
memory usage: 1.1+ MB
```

In [ ]:

```

```

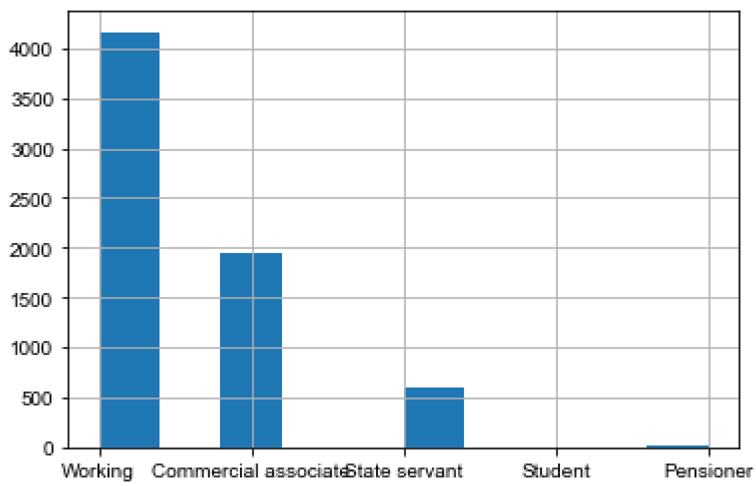
In [42]:

```
fig, axes = plt.subplots(1,3)

g1= merged_data['CODE_GENDER'].value_counts().plot.pie(explode=[0.1,0.1], ax=axes[0])
g1.set_title("Customer Distribution by Gender")

g2= merged_data['FLAG_OWN_CAR'].value_counts().plot.pie(explode=[0.1,0.1], ax=axes[1])
g2.set_title("Car Ownership")

g3= merged_data['FLAG_OWN_REALTY'].value_counts().plot.pie(explode=[0.1,0.1], ax=axes[2
])
g3.set_title("Realty Ownership")

fig.set_size_inches(14,5)

plt.tight_layout()

plt.show()
```



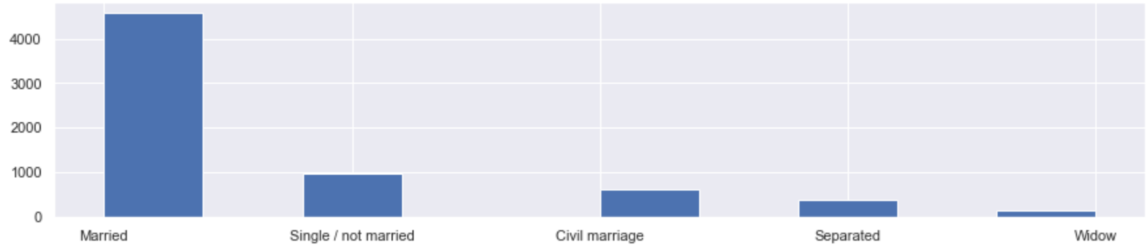
In [43]:

```
#Customer Distribution by Income Type

merged_data['NAME_INCOME_TYPE'].hist()
sns.set(rc={'figure.figsize':(15,3)})
```



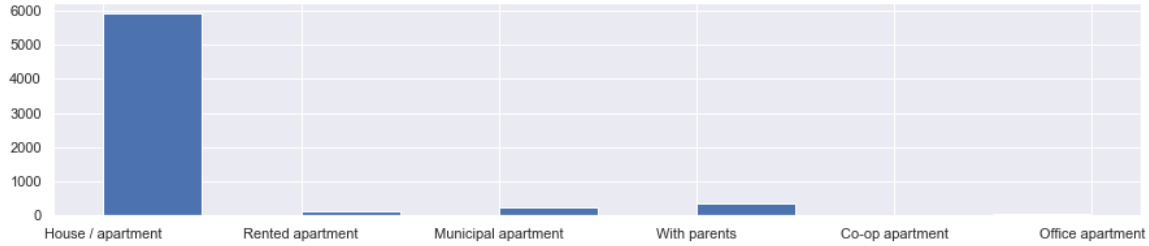
In [44]:

```
#Customer Distribution by family status
merged_data['NAME_FAMILY_STATUS'].hist()
sns.set(rc={'figure.figsize':(15,3)})
```



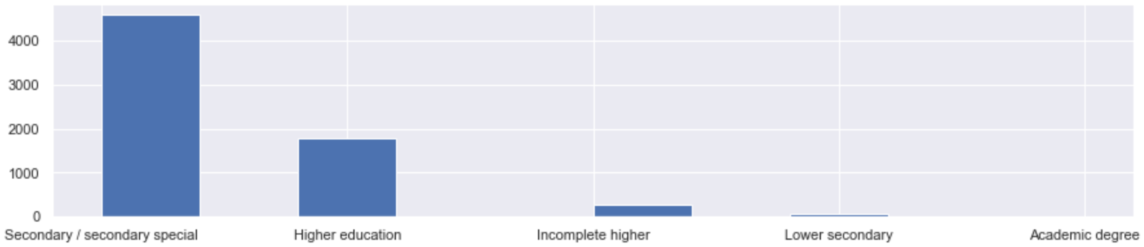
In [45]:

```
#Customer Distribution by Housing type
merged_data['NAME_HOUSING_TYPE'].hist()
sns.set(rc={'figure.figsize':(15,3)})
```



In [46]:

```
#Customer Distribution by Education Type
merged_data['NAME_EDUCATION_TYPE'].hist()
sns.set(rc={'figure.figsize':(15,3)})
```

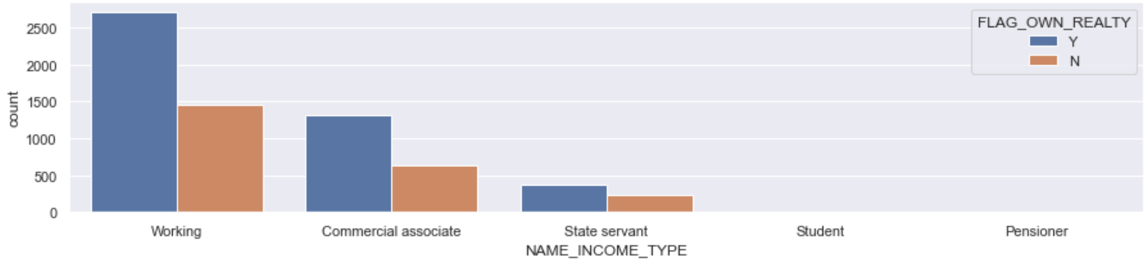


In [47]:

```
#Income type Distribution in realty ownership
from pylab import rcParams
sns.set(rc={'figure.figsize':(15,3)})
sns.countplot(x='NAME_INCOME_TYPE',hue='FLAG_OWN_REALTY',data=merged_data)
```

Out[47]:

```
<AxesSubplot:xlabel='NAME_INCOME_TYPE', ylabel='count'>
```

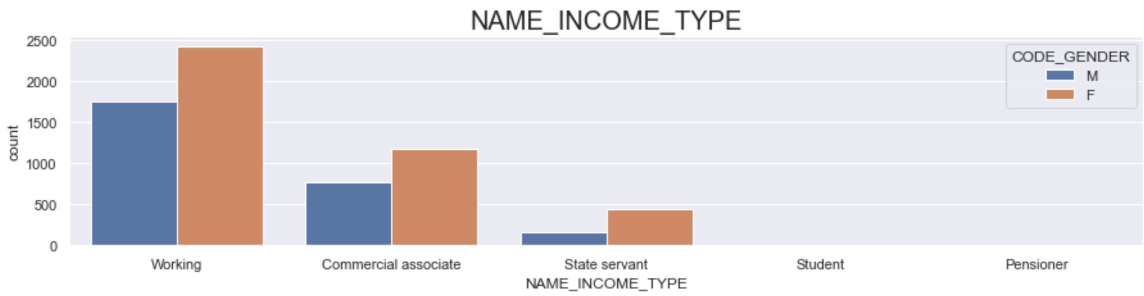


In [48]:

```
#Income type Distribution in gender
sns.set(rc={'figure.figsize':(15,3)})
S=sns.countplot(x='NAME_INCOME_TYPE',hue='CODE_GENDER',data=merged_data)
S.axes.set_title("NAME_INCOME_TYPE",fontsize=20)
```

Out[48]:

```
Text(0.5, 1.0, 'NAME_INCOME_TYPE')
```

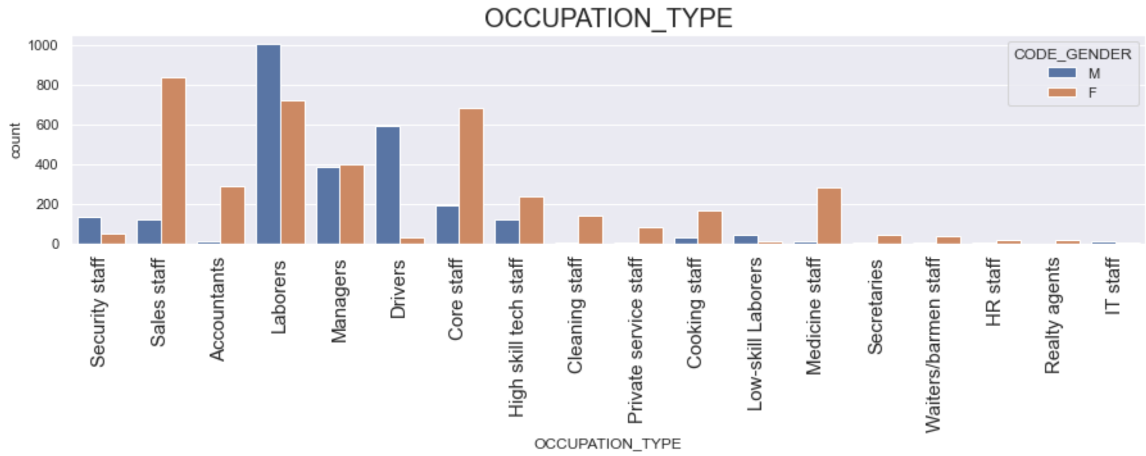


In [49]:

```
sns.set(rc={'figure.figsize':(15,3)})
plt.xticks(fontsize=15,rotation='vertical')
P=sns.countplot(x='OCCUPATION_TYPE',hue='CODE_GENDER',data=merged_data)
P.axes.set_title("OCCUPATION_TYPE",fontsize=20)
```

Out[49]:

```
Text(0.5, 1.0, 'OCCUPATION_TYPE')
```



In [161]:

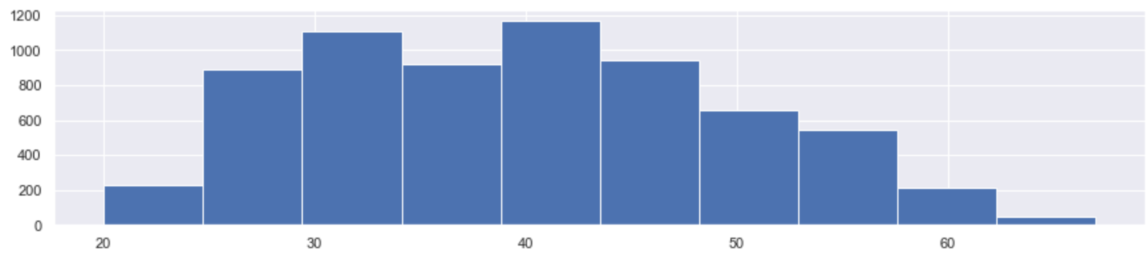
```
#customer distribution by age
sns.set(rc={'figure.figsize':(10,3)})
#merged_data['Age']= -(merged_data['DAYS_BIRTH'])//365
#merged_data['Age']= merged_data['Age'].astype(int)
#print(merged_data['Age'].value_counts(bins=10,normalize=True,sort=False))
#merged_data['Age'].plot(kind='hist',bins=20,density=True)
#plt.show()
```

In [50]:

```
merged_data['Age'].hist()
```

Out[50]:

```
<AxesSubplot:>
```

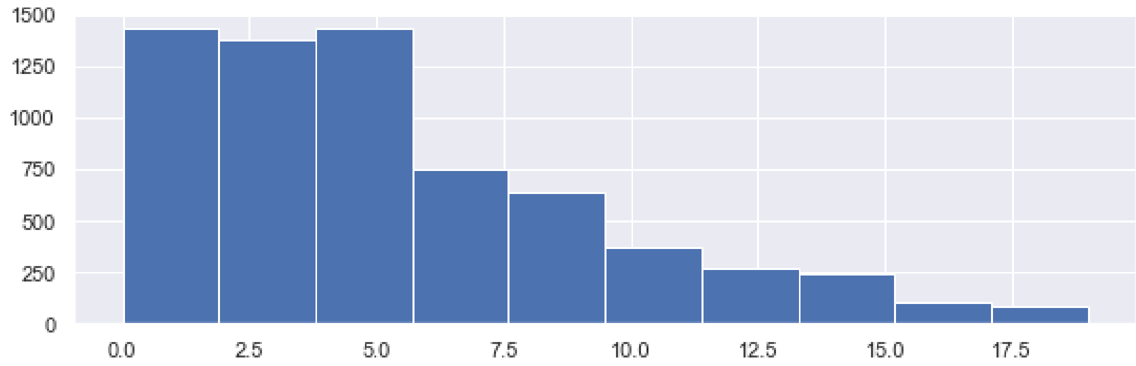


In [51]:

```
#customer distribution by years of employment

sns.set(rc={'figure.figsize':(10,3)})
#merged_data['Years_of_employment']= -(merged_data['DAYS_EMPLOYED'])//365
#merged_data['Years_of_employment']= merged_data['Years_of_employment'].astype(int)
#print(merged_data['Age'].value_counts(bins=10,normalize=True,sort=False))
#merged_data['Years_of_employment'].plot(kind='hist',bins=20,density=True)
#plt.show()
merged_data['Years_of_employment'].hist()
```

Out[51]:

<AxesSubplot:>



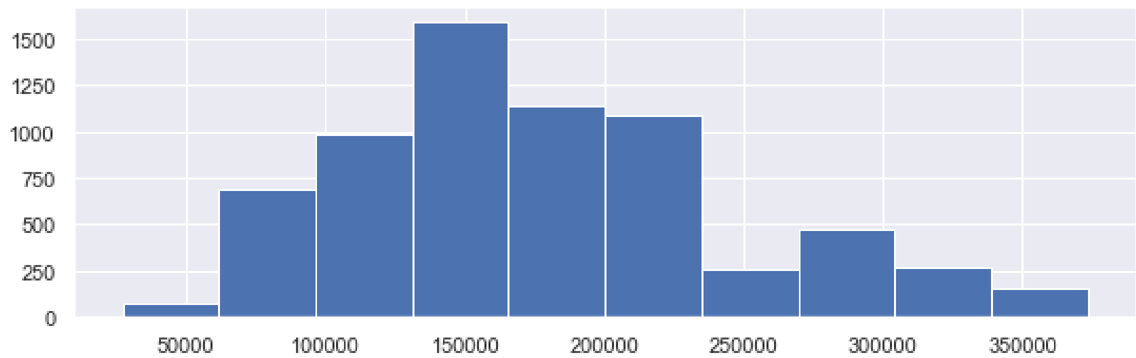
In [ ]:

In [52]:

```
sns.set(rc={'figure.figsize':(10,3)})
#merged_data['AMT_INCOME_TOTAL']=merged_data['AMT_INCOME_TOTAL'].astype(object)
#merged_data['AMT_INCOME_TOTAL'] = merged_data['AMT_INCOME_TOTAL']/10000
#print(merged_data['AMT_INCOME_TOTAL'].value_counts(bins=10,sort=False))
#merged_data['AMT_INCOME_TOTAL'].plot(kind='hist',bins=60,density=True)
#plt.show()

merged_data['AMT_INCOME_TOTAL'].hist()
```

Out[52]:

<AxesSubplot:>

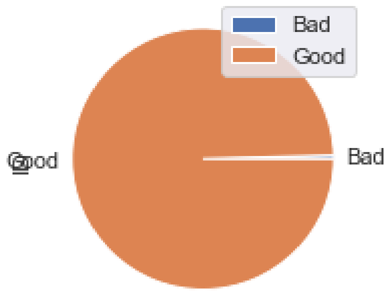


In [53]:

```
#optional

merged_data.groupby(['Status']).count().plot(kind='pie', y='ID')
```

Out[53]:

```
<AxesSubplot:ylabel='ID'>
```

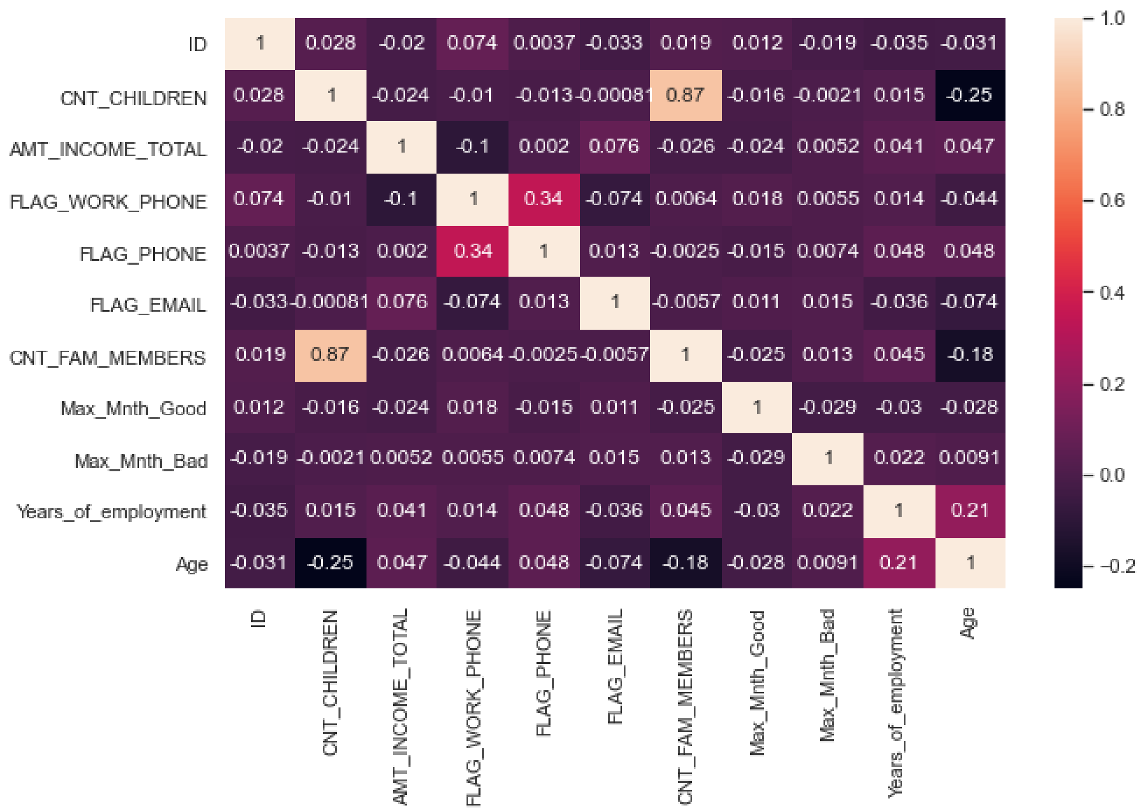


In [54]:

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

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(merged_data.corr(), ax=ax, annot=True)
```

Out[54]:

```
<AxesSubplot:>
```

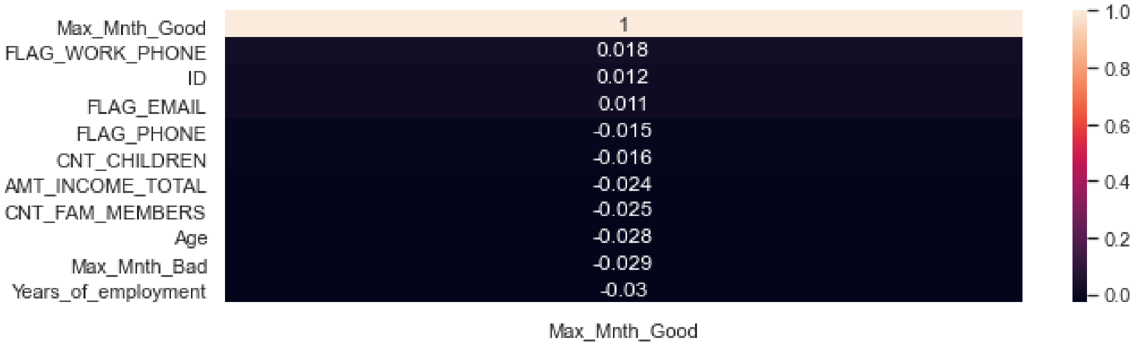


In [55]:

```
#optional

corr = merged_data.corr()[['Max_Mnth_Good']].sort_values(by='Max_Mnth_Good', ascending=
False)

sns.heatmap(corr, annot=True)
```

Out[55]:

<AxesSubplot:>

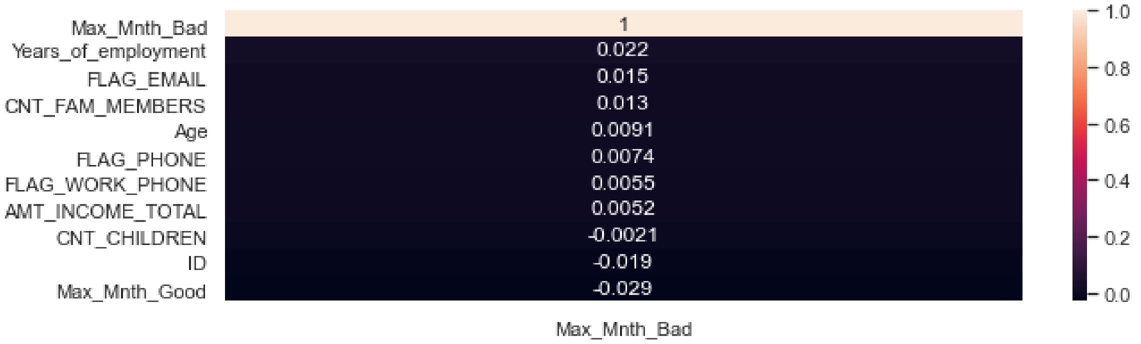


In [56]:

```
#optional

corr = merged_data.corr()[['Max_Mnth_Bad']].sort_values(by='Max_Mnth_Bad', ascending=Fa
lse)

sns.heatmap(corr, annot=True)
```

Out[56]:

<AxesSubplot:>



In [62]:

```
merged_data = merged_data[['ID','CODE_GENDER','FLAG_OWN_CAR','FLAG_OWN_REALTY','CNT_CHI
LDREN','AMT_INCOME_TOTAL','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE','NAME_FAMILY_STATUS'
,'NAME_HOUSING_TYPE','FLAG_WORK_PHONE','FLAG_PHONE','FLAG_EMAIL','OCCUPATION_TYPE','CNT
_FAM_MEMBERS','Years_of_employment','Age','Max_Mnth_Good','Max_Mnth_Bad','Status']]
```

In [65]:

```
merged_data
```

Out[65]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AM
0	5008806	M	Y	Y	0	
1	5008808	F	N	Y	0	
2	5008815	M	Y	Y	0	
3	5008819	M	Y	Y	0	
4	5008825	F	Y	N	0	
...	...	...	...	...	...	
6710	5143578	M	Y	N	0	
6711	5146078	F	N	Y	1	
6712	5148694	F	N	N	0	
6713	5149838	F	N	Y	0	
6714	5150337	M	N	Y	0	

6715 rows × 20 columns

In [66]:

```
xData = merged_data[merged_data.columns[1:-3]]
yData = merged_data[merged_data.columns[-1]]
```

In [68]:

```
yData.value_counts()
```

Out[68]:

```
Good    6688
Bad       27
Name: Status, dtype: int64
```

In [72]:

```
#Encoding categorical values
xData = pd.get_dummies(xData,drop_first=True)
```

In [73]:

```
xData
```

Out[73]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	FLAG_WORK_PHONE	FLAG_PHONE	FLAG_EM
0	0	112500.0	0	0	
1	0	270000.0	0	1	
2	0	270000.0	1	1	
3	0	135000.0	0	0	
4	0	130500.0	0	0	
...	...	...	...	...	
6710	0	157500.0	1	0	
6711	1	108000.0	1	1	
6712	0	180000.0	0	0	
6713	0	157500.0	0	1	
6714	0	112500.0	0	0	

6715 rows × 45 columns

In [74]:

```
#Standardization
std = StandardScaler()
std.fit(xData)
xScal = std.transform(xData)
xScal  = pd.DataFrame(xScal,columns=xData.columns)
xScal.head(10)
```

Out[74]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	FLAG_WORK_PHONE	FLAG_PHONE	FLAG_EMAIL
0	-0.658210	-0.907142	-0.602870	-0.634401	-0.327929
1	-0.658210	1.371612	-0.602870	1.576290	3.049439
2	-0.658210	1.371612	1.658733	1.576290	3.049439
3	-0.658210	-0.581606	-0.602870	-0.634401	-0.327929
4	-0.658210	-0.646713	-0.602870	-0.634401	-0.327929
5	-0.658210	-0.256069	-0.602870	1.576290	-0.327929
6	-0.658210	1.371612	-0.602870	-0.634401	-0.327929
7	0.816066	-0.125855	-0.602870	-0.634401	-0.327929
8	-0.658210	-0.907142	-0.602870	1.576290	-0.327929
9	2.290341	-0.581606	-0.602870	-0.634401	-0.327929

10 rows × 45 columns

In [75]:

```
#Find the Random State with required count of 'Bad' values
rndm_stat = [train_test_split(xData,yData,random_state=x) for x in range(100)]
badCounts = [rndm_stat[i][3].value_counts()['Bad'] for i in range(100)]
bstRndmStat = badCounts.index(4,8)
bstRndmStat
```

Out[75]:

12

In [76]:

```
#Split into Training & Testing
X_train, X_test, y_train, y_test = train_test_split(xData,yData,random_state=bstRndmSta
t)
```

In [77]:

```
#Under-sampling the 'Good' status
tl = TomekLinks(sampling_strategy='majority')
x_tl, y_tl = tl.fit_resample(X_train,y_train)
print('Original dataset shape', y_train.value_counts())
print('Resample dataset shape', y_tl.value_counts())
```

```
Original dataset shape Good    5013
Bad       23
Name: Status, dtype: int64
Resample dataset shape Good    5001
Bad       23
Name: Status, dtype: int64
```

In [78]:

```
#Oversampling the 'Bad' status
smote = SMOTE()
x_smote, y_smote = smote.fit_resample(x_tl, y_tl)
print('Original dataset shape', y_tl.value_counts())
print('Resample dataset shape', y_smote.value_counts())
```

```
Original dataset shape Good    5001
Bad       23
Name: Status, dtype: int64
Resample dataset shape Good    5001
Bad     5001
Name: Status, dtype: int64
```

In [80]:

```
#KNN Classifier
#from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_smote, y_smote)
y_pred = knn.predict(X_test)
print(knn.score(X_test, y_test))
confusion_matrix(y_test, y_pred)
```

```
0.9737939249553306
```

Out[80]:

```
array([[   0,    4],
       [  40, 1635]], dtype=int64)
```

In [90]:

```
#Random Forest Classification

rdf = RandomForestClassifier(max_depth=2, random_state=10)
rdf.fit(x_smote, y_smote)
y_pred = rdf.predict(X_test)
print(rdf.score(X_test, y_test))
confusion_matrix(y_test, y_pred)
```

```
0.8826682549136391
```

Out[90]:

```
array([[   1,    3],
       [ 194, 1481]], dtype=int64)
```

In [82]:

```

```

In [ ]:

```

```

In [ ]:

```

```