In [1]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Let's start by importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
%matplotlib inline
```

In [3]:
```
plt.style.use('ggplot')
```

Importing the Data

In [4]:
```
application_df = pd.read_csv(r"C:\Users\Arindham Krishna\OneDrive\Desktop\Projects for Portfolio\Exploratory Data Analysis\Loan Applications
prev_ap_df = pd.read_csv(r"C:\Users\Arindham Krishna\OneDrive\Desktop\Projects for Portfolio\Exploratory Data Analysis\Loan Applications Data
```

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

Out[5]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANI
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	(
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	2

5 rows × 122 columns

In [6]:
```
prev_ap_df.head()
```

Out[6]:

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

5 rows × 37 columns

In [7]:
```
application_df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 122 columns):
 #    Column                       Dtype
---   ------                       -----
 0    SK_ID_CURR                   int64
 1    TARGET                       int64
 2    NAME_CONTRACT_TYPE           object
 3    CODE_GENDER                  object
 4    FLAG_OWN_CAR                 object
 5    FLAG_OWN_REALTY              object
 6    CNT_CHILDREN                 int64
 7    AMT_INCOME_TOTAL             float64
 8    AMT_CREDIT                   float64
 9    AMT_ANNUITY                  float64
 10   AMT_GOODS_PRICE              float64
 11   NAME_TYPE_SUITE              object
 12   NAME_INCOME_TYPE             object
 13   NAME_EDUCATION_TYPE          object
 14   NAME_FAMILY_STATUS           object
 15   NAME_HOUSING_TYPE            object
 16   REGION_POPULATION_RELATIVE   float64
 17   DAYS_BIRTH                   int64
 18   DAYS_EMPLOYED                int64
 19   DAYS_REGISTRATION            float64
 20   DAYS_ID_PUBLISH              int64
 21   OWN_CAR_AGE                  float64
 22   FLAG_MOBIL                   int64
 23   FLAG_EMP_PHONE               int64
 24   FLAG_WORK_PHONE              int64
 25   FLAG_CONT_MOBILE             int64
 26   FLAG_PHONE                   int64
 27   FLAG_EMAIL                   int64
 28   OCCUPATION_TYPE              object
 29   CNT_FAM_MEMBERS              float64
```

```
 30   REGION_RATING_CLIENT          int64
 31   REGION_RATING_CLIENT_W_CITY   int64
 32   WEEKDAY_APPR_PROCESS_START    object
 33   HOUR_APPR_PROCESS_START       int64
 34   REG_REGION_NOT_LIVE_REGION    int64
 35   REG_REGION_NOT_WORK_REGION    int64
 36   LIVE_REGION_NOT_WORK_REGION   int64
 37   REG_CITY_NOT_LIVE_CITY        int64
 38   REG_CITY_NOT_WORK_CITY        int64
 39   LIVE_CITY_NOT_WORK_CITY       int64
 40   ORGANIZATION_TYPE             object
 41   EXT_SOURCE_1                  float64
 42   EXT_SOURCE_2                  float64
 43   EXT_SOURCE_3                  float64
 44   APARTMENTS_AVG                float64
 45   BASEMENTAREA_AVG              float64
 46   YEARS_BEGINEXPLUATATION_AVG   float64
 47   YEARS_BUILD_AVG               float64
 48   COMMONAREA_AVG                float64
 49   ELEVATORS_AVG                 float64
 50   ENTRANCES_AVG                 float64
 51   FLOORSMAX_AVG                 float64
 52   FLOORSMIN_AVG                 float64
 53   LANDAREA_AVG                  float64
 54   LIVINGAPARTMENTS_AVG          float64
 55   LIVINGAREA_AVG                float64
 56   NONLIVINGAPARTMENTS_AVG       float64
 57   NONLIVINGAREA_AVG             float64
 58   APARTMENTS_MODE               float64
 59   BASEMENTAREA_MODE             float64
 60   YEARS_BEGINEXPLUATATION_MODE  float64
 61   YEARS_BUILD_MODE              float64
 62   COMMONAREA_MODE               float64
 63   ELEVATORS_MODE                float64
 64   ENTRANCES_MODE                float64
 65   FLOORSMAX_MODE                float64
 66   FLOORSMIN_MODE                float64
 67   LANDAREA_MODE                 float64
 68   LIVINGAPARTMENTS_MODE         float64
 69   LIVINGAREA_MODE               float64
 70   NONLIVINGAPARTMENTS_MODE      float64
 71   NONLIVINGAREA_MODE            float64
 72   APARTMENTS_MEDI               float64
 73   BASEMENTAREA_MEDI             float64
 74   YEARS_BEGINEXPLUATATION_MEDI  float64
 75   YEARS_BUILD_MEDI              float64
 76   COMMONAREA_MEDI               float64
 77   ELEVATORS_MEDI                float64
 78   ENTRANCES_MEDI                float64
 79   FLOORSMAX_MEDI                float64
 80   FLOORSMIN_MEDI                float64
 81   LANDAREA_MEDI                 float64
 82   LIVINGAPARTMENTS_MEDI         float64
 83   LIVINGAREA_MEDI               float64
 84   NONLIVINGAPARTMENTS_MEDI      float64
 85   NONLIVINGAREA_MEDI            float64
 86   FONDKAPREMONT_MODE            object
 87   HOUSETYPE_MODE                object
 88   TOTALAREA_MODE                float64
 89   WALLSMATERIAL_MODE            object
 90   EMERGENCYSTATE_MODE           object
 91   OBS_30_CNT_SOCIAL_CIRCLE      float64
 92   DEF_30_CNT_SOCIAL_CIRCLE      float64
 93   OBS_60_CNT_SOCIAL_CIRCLE      float64
 94   DEF_60_CNT_SOCIAL_CIRCLE      float64
 95   DAYS_LAST_PHONE_CHANGE        float64
 96   FLAG_DOCUMENT_2               int64
 97   FLAG_DOCUMENT_3               int64
 98   FLAG_DOCUMENT_4               int64
 99   FLAG_DOCUMENT_5               int64
 100  FLAG_DOCUMENT_6               int64
 101  FLAG_DOCUMENT_7               int64
 102  FLAG_DOCUMENT_8               int64
 103  FLAG_DOCUMENT_9               int64
 104  FLAG_DOCUMENT_10              int64
 105  FLAG_DOCUMENT_11              int64
 106  FLAG_DOCUMENT_12              int64
 107  FLAG_DOCUMENT_13              int64
 108  FLAG_DOCUMENT_14              int64
 109  FLAG_DOCUMENT_15              int64
 110  FLAG_DOCUMENT_16              int64
 111  FLAG_DOCUMENT_17              int64
 112  FLAG_DOCUMENT_18              int64
 113  FLAG_DOCUMENT_19              int64
 114  FLAG_DOCUMENT_20              int64
 115  FLAG_DOCUMENT_21              int64
 116  AMT_REQ_CREDIT_BUREAU_HOUR    float64
 117  AMT_REQ_CREDIT_BUREAU_DAY     float64
 118  AMT_REQ_CREDIT_BUREAU_WEEK    float64
 119  AMT_REQ_CREDIT_BUREAU_MON     float64
 120  AMT_REQ_CREDIT_BUREAU_QRT     float64
 121  AMT_REQ_CREDIT_BUREAU_YEAR    float64
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

In [8]:

```
prev_ap_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
```

```
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype  
---  ------                       --------------    -----  
 0   SK_ID_PREV                   1670214 non-null  int64  
 1   SK_ID_CURR                   1670214 non-null  int64  
 2   NAME_CONTRACT_TYPE           1670214 non-null  object 
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object 
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64  
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object 
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64  
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object 
 16  NAME_CONTRACT_STATUS         1670214 non-null  object 
 17  DAYS_DECISION                1670214 non-null  int64  
 18  NAME_PAYMENT_TYPE            1670214 non-null  object 
 19  CODE_REJECT_REASON           1670214 non-null  object 
 20  NAME_TYPE_SUITE              849809 non-null   object 
 21  NAME_CLIENT_TYPE             1670214 non-null  object 
 22  NAME_GOODS_CATEGORY          1670214 non-null  object 
 23  NAME_PORTFOLIO               1670214 non-null  object 
 24  NAME_PRODUCT_TYPE            1670214 non-null  object 
 25  CHANNEL_TYPE                 1670214 non-null  object 
 26  SELLERPLACE_AREA             1670214 non-null  int64  
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object 
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object 
 30  PRODUCT_COMBINATION          1669868 non-null  object 
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
```

We can see that when we use .info(), it will give us the detailed information about the dataset. For example, application_df.info() gives an output where we can see the number of records it contains and the number of features it has. Application data set has 307511 entries and 122 columns. Again, you can also check the amount of memory the dataset is cosuming. Application dataset is consuming 286.2+ MB and the same above line you can see data types and their counts.

Same goes for the previous application dataset, prev_ap_df.info()

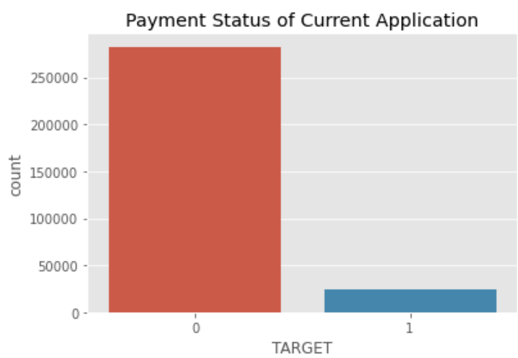
In [9]:
```
# Another way to check the number of columns a dataset has, is to go ahead with the .shape command.
print("shape of application dataset",application_df.shape)
print("shape of prev application dataset",prev_ap_df.shape)
```

```
shape of application dataset (307511, 122)
shape of prev application dataset (1670214, 37)
```

Now, as we know that we need to check the current defaulters from the loan application and also we need to analyse what are the chances that a prospective client might turn out to be a defaulter so that we can avoid the losses.

Let's first look at how current defaulter to creditor count looks like from the application dataset.

In [10]:
```
plt.title("Payment Status of Current Application")
sns.countplot(application_df['TARGET'])
plt.show()
```



(1 = Defaulter, 0 = Creditor)From the above countplot, we can see that the defaulter count is comparitively very minimal to the creditors count. This defines that our dataset is an imbalaned data.

What is Imbalanced Data?

So, while doing any analysis or majorly to do any prediction we always have a target feature. Depending on that target feature we will be able to make the predictions. Now, lets say if the target feature is imbalanced like in this case then the algorithms or classifiers will only pick up the majority values and the minority values will be ignored. In our case, if we give the same data set to any classifier then there are chances that defaulters records will be ignored and only creditors records will be considered and doing so will give us inaccurate predictions.

Another example other than this is if we consider disease prediction data and in there if 95 patiens are without disease and 5 are with disease then chances are more that the classifier will ignore the minority records.

Checking the ratio

In [11]:
```
creditor = application_df[application_df['TARGET']==0]
defaulter = application_df[application_df["TARGET"]==1]

print("Number of Creditors", creditor.shape[0])
print("Number of Defaulters", defaulter.shape[0])
```

```
Number of Creditors 282686
Number of Defaulters 24825
```

Here, we can see 282,686 (two hundred eighty thousand applications have paid their installments timely) 24,825(twenty five thousand applications are defaulters)

In [12]:
```
#Lets chcek the percentage of defaulters
print("Percentage of defaulters:", round(defaulter.shape[0]*100/(creditor.shape[0]+defaulter.shape[0]),2))
```

```
Percentage of defaulters: 8.07
```

Almost 8 percent of applications are into the defualters list. Rest 92 percent are creditors.

Defaulter : Creditor = 8:92

Now lets start with our analysis and find out which clients can be a defaulter and which clients are good prospects and not to loose them by not providing loan.

Let's start with analysing all the features we have got.

In [13]:
```
#This will print out all the features, in list. We can also use just list(application_df.columns)
print(list(application_df.columns))
```

```
['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CRE
DIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYP
E', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OWN_CAR_AGE', 'FLAG_MOBIL', 'FLAG_E
MP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START', 'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_RE
GION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE', 'EX
T_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BUILD_AVG', 'COMMONA
REA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONL
IVINGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMO
NAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MO
DE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_M
EDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'L
IVINGAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'TOTALAREA_MODE', 'WALLSMATERIAL_M
ODE', 'EMERGENCYSTATE_MODE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7',
'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FL
AG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21', 'AMT
_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QR
T', 'AMT_REQ_CREDIT_BUREAU_YEAR']
```

Incestigation 1: Documents and its Impact on Target

In [14]:
```
application_documents_df = application_df[['FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5',
                                           'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9',
                                           'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13',
                                           'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17',
                                           'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',]]
```

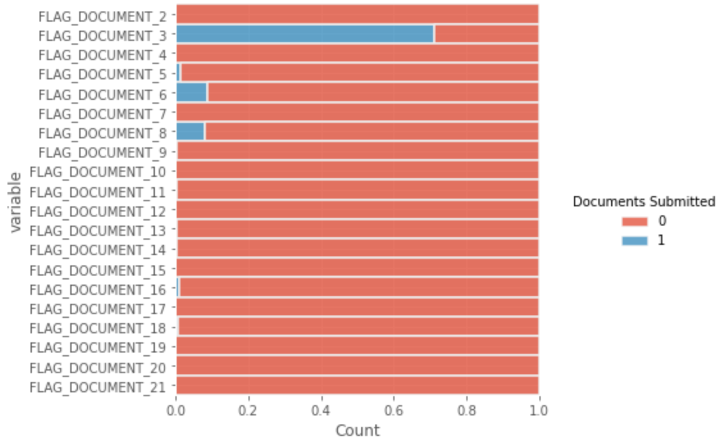
In [15]:
```
application_documents_df.head()
```

Out[15]:

	FLAG_DOCUMENT_2	FLAG_DOCUMENT_3	FLAG_DOCUMENT_4	FLAG_DOCUMENT_5	FLAG_DOCUMENT_6	FLAG_DOCUMENT_7	FLAG_DOCUMENT_8	FLAG_DOCUMENT
0	0	1	0	0	0	0	0	
1	0	1	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	1	0	0	0	0	0	
4	0	0	0	0	0	0	1	

In [16]:
```
plt.figure(figsize=(10,6))
sns.displot(
    data=application_documents_df.melt(value_name="Documents Submitted"),
    y="variable",
    hue="Documents Submitted",
    multiple="fill",
    aspect=1.25
)
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```



(0 = Not Submitted, 1 = Submitted) From the above plot, can see that most of the applications have not submitted all the documents except the Document_3

It's obvious that if these documents were not submitted then they will not make any imapct on our Target.

However, we will check the correlation between document_3 and Target.

Correlation Matrix Between Document 3 and Target

In [17]:
```
corr = application_df[["TARGET","FLAG_DOCUMENT_3"]].corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[17]:

	TARGET	FLAG_DOCUMENT_3
TARGET	1.000000	0.044346
FLAG_DOCUMENT_3	0.044346	1.000000

As we know the correlation values range between -1 to 1 and any values nearer or equal to -1 determines a negative correlation, any value nearer to 0 determines no correlation and any value near to 1 or equal to 1 determines that there is correlation.

From the above matrix, we can see that the correlation values are nearer to 0 and hence document_3 submission does not impact the target value by any chance. We can also go ahead and drop document_3 feature along with other documents.

Investigation 2: Clients House Details vs Target

Lets see if the information provided to us about the size and other details of clients stay has by any chance impact on the target columns.

It's convenient to print .columns() as you can copy the column names easily.

In [18]:
```
application_houseinfo_df = application_df[['APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG',
                                            'YEARS_BUILD_AVG', 'COMMONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG',
                                            'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG',
                                            'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG',
                                            'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE',
                                            'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE',
                                            'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE',
                                            'LIVINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE',
                                            'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI',
                                            'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI',
                                            'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI',
                                            'LIVINGAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI',
                                            'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'TOTALAREA_MODE', 'WALLSMATERIAL_MODE',
                                            'EMERGENCYSTATE_MODE']]
```

In [19]:
```
application_houseinfo_df.head()
```

Out[19]:

	APARTMENTS_AVG	BASEMENTAREA_AVG	YEARS_BEGINEXPLUATATION_AVG	YEARS_BUILD_AVG	COMMONAREA_AVG	ELEVATORS_AVG	ENTRANCES_AVG	FLOORSM/
0	0.0247	0.0369	0.9722	0.6192	0.0143	0.00	0.0690	
1	0.0959	0.0529	0.9851	0.7960	0.0605	0.08	0.0345	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

5 rows × 47 columns

Looks like these features can contain null records, lets investigate for that.

In [20]:
```
print(application_houseinfo_df.isna().sum())
#APARTMENTS_AVG has 156,061 missing records and on a glance, we can see that almost all these features has high missing values.
```

```
APARTMENTS_AVG                  156061
BASEMENTAREA_AVG                179943
YEARS_BEGINEXPLUATATION_AVG     150007
YEARS_BUILD_AVG                 204488
COMMONAREA_AVG                  214865
```

```
ELEVATORS_AVG                   163891
ENTRANCES_AVG                   154828
FLOORSMAX_AVG                   153020
FLOORSMIN_AVG                   208642
LANDAREA_AVG                    182590
LIVINGAPARTMENTS_AVG            210199
LIVINGAREA_AVG                  154350
NONLIVINGAPARTMENTS_AVG         213514
NONLIVINGAREA_AVG               169682
APARTMENTS_MODE                 156061
BASEMENTAREA_MODE               179943
YEARS_BEGINEXPLUATATION_MODE    150007
YEARS_BUILD_MODE                204488
COMMONAREA_MODE                 214865
ELEVATORS_MODE                  163891
ENTRANCES_MODE                  154828
FLOORSMAX_MODE                  153020
FLOORSMIN_MODE                  208642
LANDAREA_MODE                   182590
LIVINGAPARTMENTS_MODE           210199
LIVINGAREA_MODE                 154350
NONLIVINGAPARTMENTS_MODE        213514
NONLIVINGAREA_MODE              169682
APARTMENTS_MEDI                 156061
BASEMENTAREA_MEDI               179943
YEARS_BEGINEXPLUATATION_MEDI    150007
YEARS_BUILD_MEDI                204488
COMMONAREA_MEDI                 214865
ELEVATORS_MEDI                  163891
ENTRANCES_MEDI                  154828
FLOORSMAX_MEDI                  153020
FLOORSMIN_MEDI                  208642
LANDAREA_MEDI                   182590
LIVINGAPARTMENTS_MEDI           210199
LIVINGAREA_MEDI                 154350
NONLIVINGAPARTMENTS_MEDI        213514
NONLIVINGAREA_MEDI              169682
FONDKAPREMONT_MODE              210295
HOUSETYPE_MODE                  154297
TOTALAREA_MODE                  148431
WALLSMATERIAL_MODE              156341
EMERGENCYSTATE_MODE             145755
dtype: int64
```

For more better understanding lets calculate missing values percentage in these particualr 47 columns.

In [21]:

```python
# print(round((application_houseinfo_df.isnull().sum()*100/application_df.shape[0]),2))
#At a glance looks like all columns have more than 45% of missing values. Let's sort them to get aware about the range.
houseinfo_missingdata = round((application_houseinfo_df.isnull().sum()*100/application_df.shape[0]),2)
print(houseinfo_missingdata.sort_values())

#As said that after sorting we can figure the range and here we can see that (47-70%) of data is missing.
#Hence, its wise to drop these records because we have records of around three hundred thousand and we will still be left be
# fair amount of records to do the analysis.
```

```
EMERGENCYSTATE_MODE             47.40
TOTALAREA_MODE                  48.27
YEARS_BEGINEXPLUATATION_AVG     48.78
YEARS_BEGINEXPLUATATION_MEDI    48.78
YEARS_BEGINEXPLUATATION_MODE    48.78
FLOORSMAX_MEDI                  49.76
FLOORSMAX_AVG                   49.76
FLOORSMAX_MODE                  49.76
HOUSETYPE_MODE                  50.18
LIVINGAREA_MEDI                 50.19
LIVINGAREA_AVG                  50.19
LIVINGAREA_MODE                 50.19
ENTRANCES_AVG                   50.35
ENTRANCES_MEDI                  50.35
ENTRANCES_MODE                  50.35
APARTMENTS_MEDI                 50.75
APARTMENTS_AVG                  50.75
APARTMENTS_MODE                 50.75
WALLSMATERIAL_MODE              50.84
ELEVATORS_MODE                  53.30
ELEVATORS_AVG                   53.30
ELEVATORS_MEDI                  53.30
NONLIVINGAREA_AVG               55.18
NONLIVINGAREA_MEDI              55.18
NONLIVINGAREA_MODE              55.18
BASEMENTAREA_AVG                58.52
BASEMENTAREA_MEDI               58.52
BASEMENTAREA_MODE               58.52
LANDAREA_MODE                   59.38
LANDAREA_MEDI                   59.38
LANDAREA_AVG                    59.38
YEARS_BUILD_AVG                 66.50
YEARS_BUILD_MODE                66.50
YEARS_BUILD_MEDI                66.50
FLOORSMIN_AVG                   67.85
FLOORSMIN_MEDI                  67.85
FLOORSMIN_MODE                  67.85
LIVINGAPARTMENTS_AVG            68.35
LIVINGAPARTMENTS_MEDI           68.35
LIVINGAPARTMENTS_MODE           68.35
FONDKAPREMONT_MODE              68.39
NONLIVINGAPARTMENTS_MODE        69.43
NONLIVINGAPARTMENTS_AVG         69.43
NONLIVINGAPARTMENTS_MEDI        69.43
COMMONAREA_AVG                  69.87
```

```
COMMONAREA_MODE              69.87
COMMONAREA_MEDI              69.87
dtype: float64
```

In [22]:

```
#Dropping the redundant features.
application_df.drop(['APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG',
                                             'YEARS_BUILD_AVG', 'COMMONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG',
                                             'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG',
                                             'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG',
                                             'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE',
                                             'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE',
                                             'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE',
                                             'LIVINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE',
                                             'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI',
                                             'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI',
                                             'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI',
                                             'LIVINGAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI',
                                             'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'TOTALAREA_MODE', 'WALLSMATERIAL_MODE',
                                             'EMERGENCYSTATE_MODE','FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5',
                                             'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9',
                                             'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13',
                                             'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17',
                                             'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',], axis=1)
```

Out[22]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AM
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	
...	...	...	...	...	...	...	...	...	...	
307506	456251	0	Cash loans	M	N	N	0	157500.0	254700.0	
307507	456252	0	Cash loans	F	N	Y	0	72000.0	269550.0	
307508	456253	0	Cash loans	F	N	Y	0	153000.0	677664.0	
307509	456254	1	Cash loans	F	N	Y	0	171000.0	370107.0	
307510	456255	0	Cash loans	F	N	N	0	157500.0	675000.0	

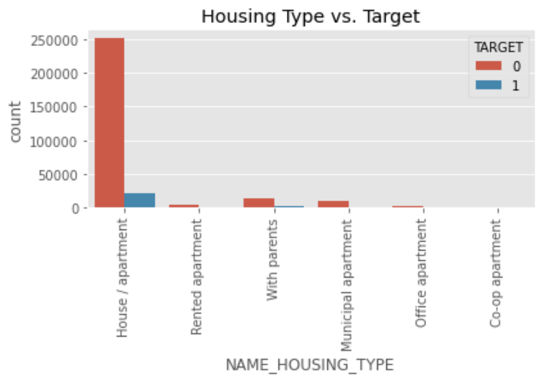
307511 rows × 55 columns

The application data had 122 columns initially but after finding out no correlation of particular columns with target we have dropped them and now we are left with 307511 rows × 55 columns

House Type vs Target

In [23]:

```
plt.figure()
sns.countplot(application_df["NAME_HOUSING_TYPE"], hue=application_df["TARGET"])
plt.xticks(rotation=90)
plt.tight_layout()
plt.title("Housing Type vs. Target")
plt.show()
```



In [24]:

```
#### Defining a function so that we get percentage of defaulters for that particular column. #####
# Function_name : value_wise_defaulter_percentage
# Usage : Returns % of defaulters for every unique value of a column(Categorical)
# Arguments : dataframe, column
# Returns : a dataframe containing unique values of a caterory and % of defaulters

def value_wise_defaulter_percentage(df, col):
    new_df = pd.DataFrame(columns=['Value', 'Percentage of Defaulter'])

    for value in df[col].unique():
        default_cnt = df[(df[col] == value) & (df.TARGET == 1)].shape[0]
        total_cnt = df[df[col] == value].shape[0]
        new_df = new_df.append({'Value' : value , 'Percentage of Defaulter' : (default_cnt*100/total_cnt)}, ignore_index=True)
    return new_df.sort_values(by='Percentage of Defaulter', ascending=False)
```

In [25]:
```
value_wise_defaulter_percentage(application_df, "NAME_HOUSING_TYPE")
```

Out[25]:

	Value	Percentage of Defaulter
1	Rented apartment	12.313051
2	With parents	11.698113
3	Municipal apartment	8.539748
5	Co-op apartment	7.932264
0	House / apartment	7.795711
4	Office apartment	6.572411

It can be seent that clients living in Rented apartment or living with parents have higher chances of being a defaulter.

Marital Status vs Target

In [26]:
```
value_wise_defaulter_percentage(application_df,"NAME_FAMILY_STATUS")
```

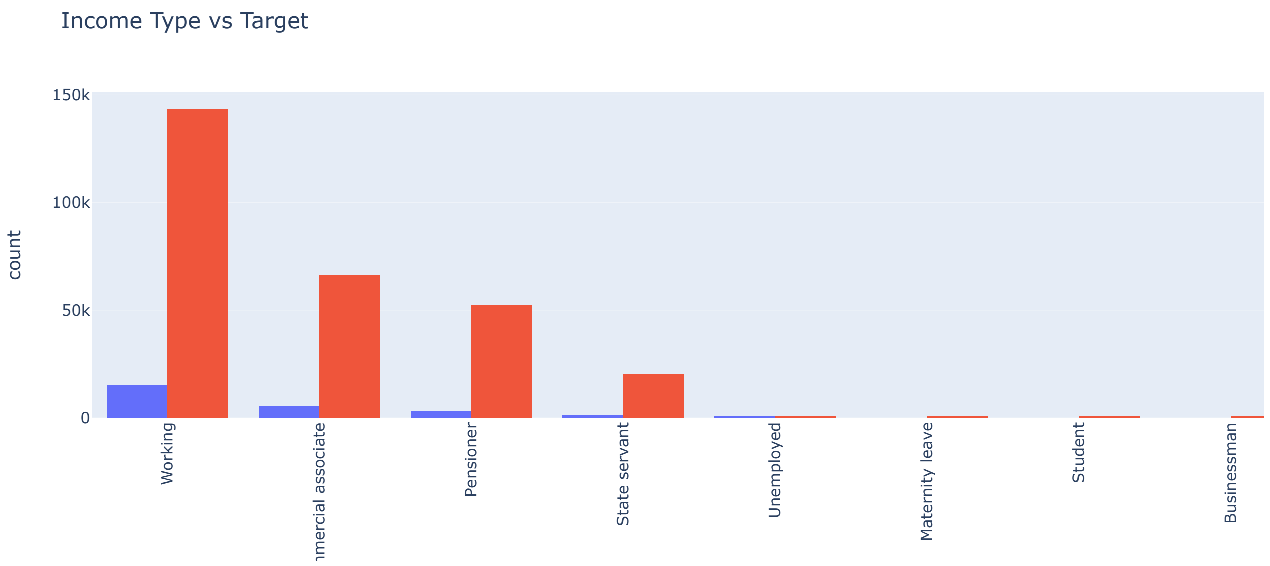
Out[26]:

	Value	Percentage of Defaulter
2	Civil marriage	9.944584
0	Single / not married	9.807675
4	Separated	8.194234
1	Married	7.559868
3	Widow	5.824217
5	Unknown	0.000000

We can see that applicants with Civil Marriage and Signle Staus have higher possibility of being a defaulter.

Income Type vs Target and Education Type vs Target

In [27]:
```
fig = px.histogram(application_df, x="NAME_INCOME_TYPE", color="TARGET",title= "Income Type vs Target",barmode='group')
fig.update_xaxes(tickangle = -90)
```



In [28]:
```
value_wise_defaulter_percentage(application_df,"NAME_INCOME_TYPE")
```

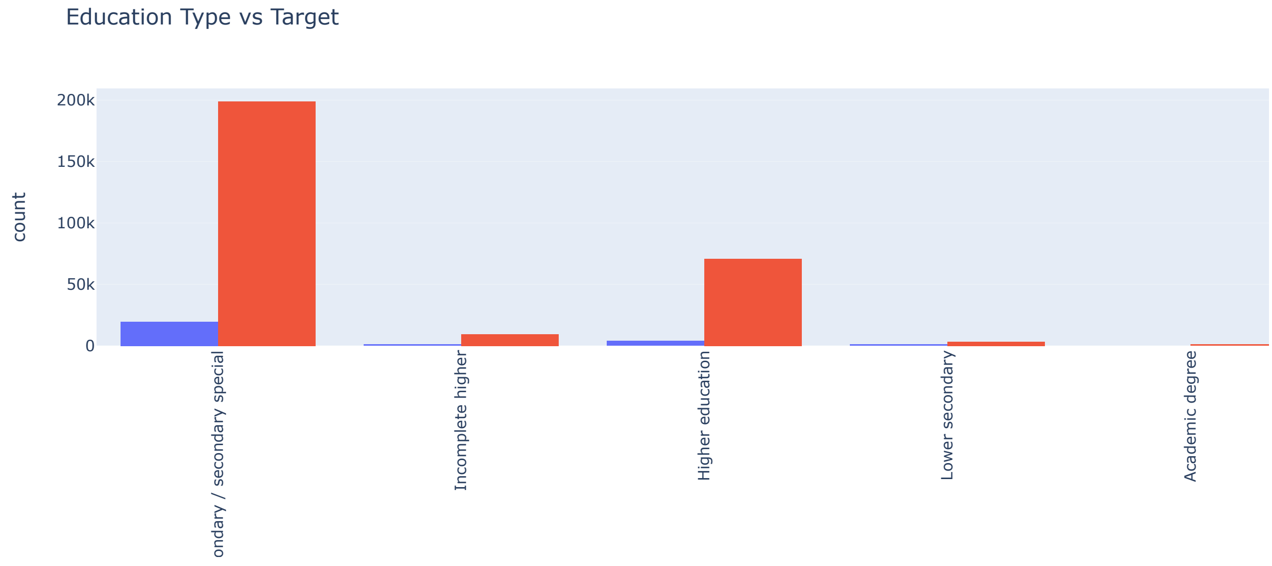
Out[28]:

	Value	Percentage of Defaulter
7	Maternity leave	40.000000
4	Unemployed	36.363636
0	Working	9.588472
2	Commercial associate	7.484257
1	State servant	5.754965
3	Pensioner	5.386366
5	Student	0.000000
6	Businessman	0.000000

Observation:

Applicants in their Maternity Leave and Applicants who are unemployed have very high chance that they can be defaulter. It should be avoided or cross checked with other parameters before sanctioning the loan.

In [29]:
```
fig = px.histogram(application_df, x="NAME_EDUCATION_TYPE", color="TARGET",title= "Education Type vs Target",barmode='group')
fig.update_xaxes(tickangle = -90)
```



In [30]:
```
value_wise_defaulter_percentage(application_df,"NAME_EDUCATION_TYPE")
```

Out[30]:

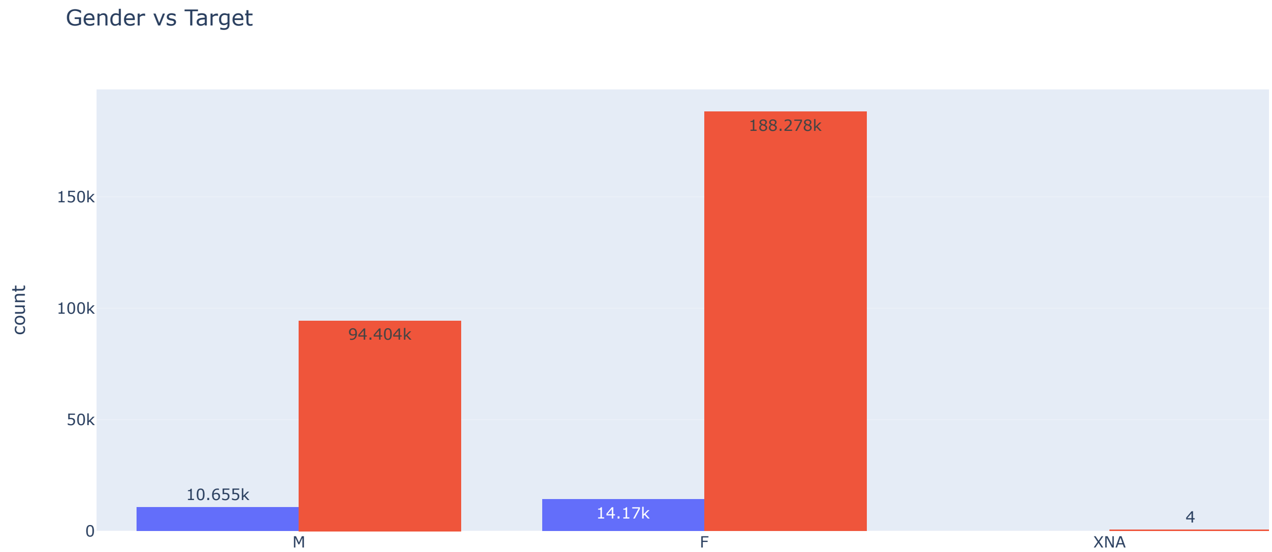
	Value	Percentage of Defaulter
3	Lower secondary	10.927673
0	Secondary / secondary special	8.939929
2	Incomplete higher	8.484966
1	Higher education	5.355115
4	Academic degree	1.829268

Observation:

Applicants with not proper education background can have the chances of not repaying the loan. Verify the education background before sanctioning the loan.

Gender, Age, Income vs Target

In [31]:
```
px.histogram(application_df, x="CODE_GENDER", color="TARGET",title= "Gender vs Target",barmode='group',text_auto=True)
```



In [32]:

```
print(application_df["CODE_GENDER"].value_counts())
```

```
F      202448
M      105059
XNA         4
Name: CODE_GENDER, dtype: int64
```

We have more Female applicants than Male and also we have more Females with loan defaulter cases.

Age

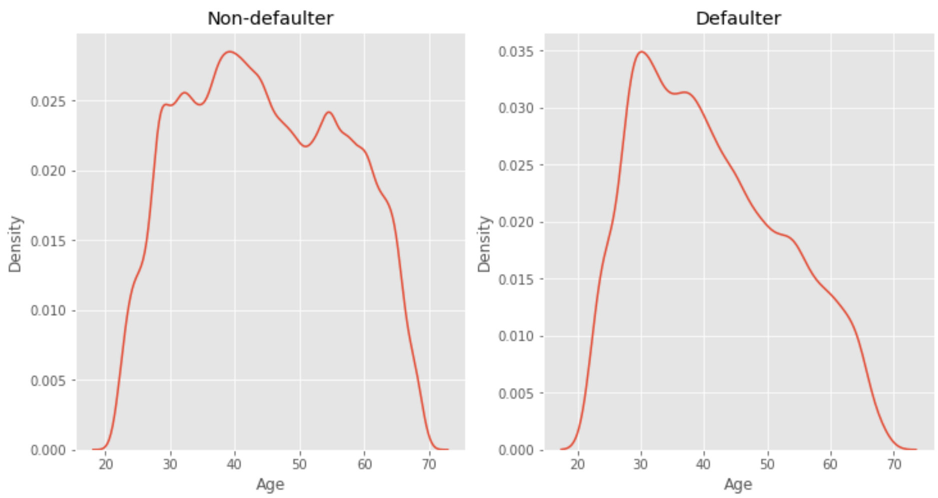
In our dataset we have age given in number of days with. Convert it by dividing with 365 or 365.25(more accurate) and if dividing with 365 then later use the abs() function to make the age positive.

In [33]:
```
# application_df["Age"] = application_df.DAYS_BIRTH//(-365.25)
# application_df[["Age"]].describe()

#or

application_df["Age"] = application_df.DAYS_BIRTH//(365.25)
application_df["Age"] = abs(application_df["Age"])
application_df["Age"].describe()
```

Out[33]:
```
count    307511.000000
mean         44.404607
std          11.945771
min          21.000000
25%          34.000000
50%          44.000000
75%          54.000000
max          70.000000
Name: Age, dtype: float64
```

We have got minimum age of applicant as 21 and maximum age of applicant that has applied is 70.

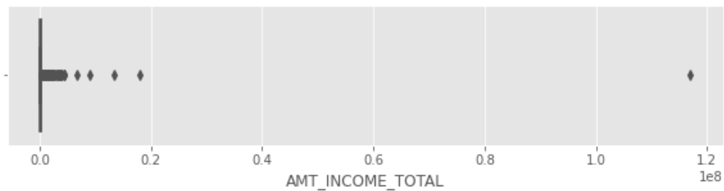
In [34]:
```
fig = plt.figure(figsize=(12,6))

ax1 = fig.add_subplot(1, 2, 1, title="Non-defaulter")
ax2 = fig.add_subplot(1, 2, 2, title="Defaulter")

sns.kdeplot(application_df[application_df["TARGET"] == 0]['Age'], ax=ax1)
sns.kdeplot(application_df[application_df["TARGET"] == 1]['Age'], ax=ax2)

plt.show()
```



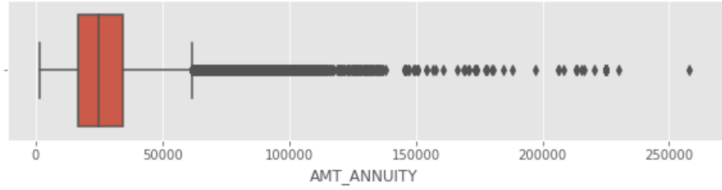
Applicants in their 30's have highest cases of default and as the age goes 40 above then the default case has seen a decrease.

Income and Annuity

In [35]:
```
plt.figure(figsize=(10,2))
sns.boxplot(application_df['AMT_INCOME_TOTAL'])
plt.show()
```



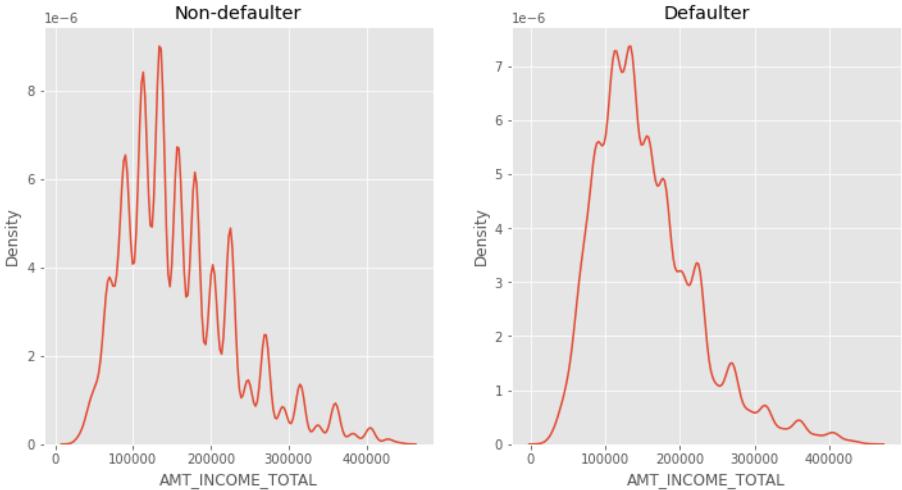
In [36]:
```
plt.figure(figsize=(10,2))
sns.boxplot(application_df['AMT_ANNUITY'])
plt.show()
```



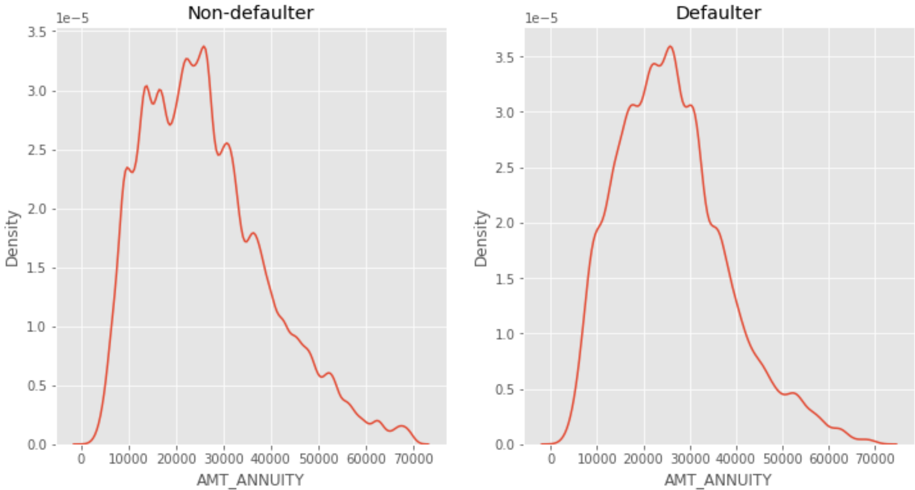
In both box plots we can see that their are outliers, These outliers can be valid too but again this will impact the other records Lets remove these outliers and then plot a graph.

In [37]:
```
application_df = application_df[application_df['AMT_ANNUITY'] < np.nanpercentile(application_df['AMT_ANNUITY'], 99)]
application_df = application_df[application_df['AMT_INCOME_TOTAL'] < np.nanpercentile(application_df['AMT_INCOME_TOTAL'], 99)]
```

In [38]:
```
fig = plt.figure(figsize=(12,6))

ax1 = fig.add_subplot(1, 2, 1, title="Non-defaulter")
ax2 = fig.add_subplot(1, 2, 2, title="Defaulter")

sns.kdeplot(application_df[application_df["TARGET"] == 0]['AMT_INCOME_TOTAL'], ax=ax1)
sns.kdeplot(application_df[application_df["TARGET"] == 1]['AMT_INCOME_TOTAL'], ax=ax2)

plt.show()
```



In [39]:
```
fig = plt.figure(figsize=(12,6))

ax1 = fig.add_subplot(1, 2, 1, title="Non-defaulter")
ax2 = fig.add_subplot(1, 2, 2, title="Defaulter")

sns.kdeplot(application_df[application_df["TARGET"] == 0]['AMT_ANNUITY'], ax=ax1)
sns.kdeplot(application_df[application_df["TARGET"] == 1]['AMT_ANNUITY'], ax=ax2)

plt.show()
```



Again, as we see that range between 100,000 and 150,000 as annual income have high chance of non repayment whereas the cases are less as annual income increases.

Amount Annuity(Monthy Installments). We see no differnece in the distribution for non defaulters and defaulters.

Top Features with high correlation for Defaulter value

In [40]:
```
default = application_df[application_df["TARGET"]==1]
default.drop(["SK_ID_CURR"],axis=1)
```

Out[40]:

	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY
0	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5
26	1	Cash loans	F	N	Y	0	112500.0	979992.0	27076.5
40	1	Cash loans	M	N	Y	0	202500.0	1193580.0	35028.0
42	1	Cash loans	F	N	N	0	135000.0	288873.0	16258.5
81	1	Cash loans	F	N	Y	0	81000.0	252000.0	14593.5
...	...	...	...	...	...	...	...	...	...
307448	1	Cash loans	M	N	N	1	207000.0	450000.0	32746.5
307475	1	Cash loans	F	N	N	1	144000.0	1303200.0	46809.0
307481	1	Cash loans	M	N	Y	0	225000.0	297000.0	19975.5
307489	1	Cash loans	F	N	Y	0	225000.0	521280.0	23089.5
307509	1	Cash loans	F	N	Y	0	171000.0	370107.0	20205.0

24414 rows × 122 columns

In [41]:

```
defaulter_corr = default.corr()
round(defaulter_corr, 2)

corr_list = defaulter_corr.unstack()
```

In [42]:

```
# Listing the correlations in pair sorted in descending order
corr_list.sort_values(ascending=False).drop_duplicates().head(11)
```

Out[42]:

```
SK_ID_CURR                SK_ID_CURR                1.000000
OBS_30_CNT_SOCIAL_CIRCLE  OBS_60_CNT_SOCIAL_CIRCLE  0.998262
BASEMENTAREA_MEDI         BASEMENTAREA_AVG          0.998251
YEARS_BUILD_AVG           YEARS_BUILD_MEDI          0.998129
COMMONAREA_AVG            COMMONAREA_MEDI           0.998041
FLOORSMIN_AVG             FLOORSMIN_MEDI            0.997963
NONLIVINGAPARTMENTS_MEDI  NONLIVINGAPARTMENTS_AVG   0.997934
LIVINGAPARTMENTS_MEDI     LIVINGAPARTMENTS_AVG      0.997701
NONLIVINGAPARTMENTS_MODE  NONLIVINGAPARTMENTS_MEDI  0.997359
FLOORSMAX_MEDI            FLOORSMAX_AVG             0.997304
ENTRANCES_AVG             ENTRANCES_MEDI            0.996697
dtype: float64
```

In [46]:

```
pip install nbconvert[webpdf]
jupyter nbconvert mynotebook.ipynb --to pdf
```

```
  File "<ipython-input-46-4c687abb0bb1>", line 1
    pip install nbconvert[webpdf]
        ^
SyntaxError: invalid syntax
```

In [44]:

```
jupyter nbconvert --to webpdf --allow-chromium-download your-notebook-file.ipynb
```

```
  File "<ipython-input-44-1be2664eae28>", line 1
    jupyter nbconvert --to webpdf --allow-chromium-download your-notebook-file.ipynb
            ^
SyntaxError: invalid syntax
```

In [45]:

```
jupyter nbconvert mynotebook.ipynb --to pdf
```

```
  File "<ipython-input-45-381043bd2394>", line 1
    jupyter nbconvert mynotebook.ipynb --to pdf
            ^
SyntaxError: invalid syntax
```

In [ ]: