EDA Banking Credit Risk Analytics Case Study - Swapnil Agade

Introduction This is the banking credit risk analysis case study where in the bank wants to understand why are the major reason for the defaults in customer paying their Loans. Basically this study is to understanding of risk analytics in banking and financial services and understand how data is used to minimise the risk of losing money while lending to customers.

Business Objectives This case study aims to identify patterns which indicate if a client has difficulty paying their installments which may be used for taking actions such as denying the loan, reducing the amount of loan, lending (to risky applicants) at a higher interest rate, etc. This will ensure that the consumers capable of repaying the loan are not rejected. Identification of such applicants using EDA is the aim of this case study.

In other words, the company wants to understand the driving factors (or driver variables) behind loan default, i.e. the variables which are strong indicators of default. The company can utilise this knowledge for its portfolio and risk assessment.

==>Business Understanding

There are three data sets

1)Application Data, 2)Previous Application Data, and 3)Columns Description

The Application data set contains all the information of the client at the time of application. The data is about whether a client has payment difficulties.

The Previous Application data set contains information about the client's previous loan data. It contains the data whether the previous application had been Approved, Cancelled, Refused or Unused offer.

The Columns Description is data dictionary which describes the meaning of the variables.

==>Expected Outcomes

Present the overall approach of the analysis in a presentation. Mention the problem statement and the analysis approach briefly.

Identify the missing data and use appropriate method to deal with it. (Remove columns/or replace it with an appropriate value)

Identify if there are outliers in the dataset. Also, mention why do you think it is an outlier. Again, remember that for this exercise, it is not necessary to remove any data points.

Identify if there is data imbalance in the data. Find the ratio of data imbalance.

Explain the results of univariate, segmented univariate, bivariate analysis, etc. in business terms.

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

```
#Importing all the required libraries
import numpy as np, pandas as pd
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline
from matplotlib import cm
from matplotlib.colors import ListedColormap,LinearSegmentedColormap
```

```
#Importing warnings to ignore
import warnings
warnings.filterwarnings("ignore")
```

```
# Reading datasets
# Setting display options to view all the columns
app_data = pd.read_csv("application_data.csv")
pre_data = pd.read_csv("previous_application.csv")
pd.set_option("display.max_columns",200)
pd.set_option("display.max_rows", 200)
```

```
# Checking first 5 data points from application dataset to have a view of data
app_data.sample(5)
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AI
56225	165150	0	Cash loans	M	Y	Y	0	157500.0	288873.0	22950.0	
142708	265473	0	Cash loans	F	N	N	0	76500.0	251280.0	13761.0	
260841	401860	0	Cash loans	F	N	N	0	94500.0	253737.0	13018.5	
176555	304589	0	Cash loans	M	Y	Y	0	112500.0	135000.0	14175.0	
129387	250072	0	Revolving loans	F	N	Y	1	135000.0	247500.0	12375.0	

```
#Exploring the application data set to understand the data types
app_data.info(verbose=True, show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 122 columns):
 #    Column                        Non-Null Count   Dtype
---   ------                        --------------   -----
 0    SK_ID_CURR                    307511 non-null  int64
 1    TARGET                        307511 non-null  int64
 2    NAME_CONTRACT_TYPE            307511 non-null  object
 3    CODE_GENDER                   307511 non-null  object
 4    FLAG_OWN_CAR                  307511 non-null  object
 5    FLAG_OWN_REALTY               307511 non-null  object
 6    CNT_CHILDREN                  307511 non-null  int64
 7    AMT_INCOME_TOTAL              307511 non-null  float64
 8    AMT_CREDIT                    307511 non-null  float64
 9    AMT_ANNUITY                   307499 non-null  float64
 10   AMT_GOODS_PRICE               307233 non-null  float64
 11   NAME_TYPE_SUITE               306219 non-null  object
 12   NAME_INCOME_TYPE              307511 non-null  object
 13   NAME_EDUCATION_TYPE           307511 non-null  object
 14   NAME_FAMILY_STATUS            307511 non-null  object
 15   NAME_HOUSING_TYPE             307511 non-null  object
 16   REGION_POPULATION_RELATIVE    307511 non-null  float64
 17   DAYS_BIRTH                    307511 non-null  int64
 18   DAYS_EMPLOYED                 307511 non-null  int64
 19   DAYS_REGISTRATION             307511 non-null  float64
 20   DAYS_ID_PUBLISH               307511 non-null  int64
 21   OWN_CAR_AGE                   104582 non-null  float64
 22   FLAG_MOBIL                    307511 non-null  int64
 23   FLAG_EMP_PHONE                307511 non-null  int64
 24   FLAG_WORK_PHONE               307511 non-null  int64
 25   FLAG_CONT_MOBILE              307511 non-null  int64
 26   FLAG_PHONE                    307511 non-null  int64
 27   FLAG_EMAIL                    307511 non-null  int64
 28   OCCUPATION_TYPE               211120 non-null  object
 29   CNT_FAM_MEMBERS               307509 non-null  float64
 30   REGION_RATING_CLIENT          307511 non-null  int64
 31   REGION_RATING_CLIENT_W_CITY   307511 non-null  int64
 32   WEEKDAY_APPR_PROCESS_START    307511 non-null  object
 33   HOUR_APPR_PROCESS_START       307511 non-null  int64
 34   REG_REGION_NOT_LIVE_REGION    307511 non-null  int64
 35   REG_REGION_NOT_WORK_REGION    307511 non-null  int64
 36   LIVE_REGION_NOT_WORK_REGION   307511 non-null  int64
 37   REG_CITY_NOT_LIVE_CITY        307511 non-null  int64
 38   REG_CITY_NOT_WORK_CITY        307511 non-null  int64
 39   LIVE_CITY_NOT_WORK_CITY       307511 non-null  int64
 40   ORGANIZATION_TYPE             307511 non-null  object
 41   EXT_SOURCE_1                  134133 non-null  float64
 42   EXT_SOURCE_2                  306851 non-null  float64
 43   EXT_SOURCE_3                  246546 non-null  float64
 44   APARTMENTS_AVG                151450 non-null  float64
 45   BASEMENTAREA_AVG              127568 non-null  float64
 46   YEARS_BEGINEXPLUATATION_AVG   157504 non-null  float64
 47   YEARS_BUILD_AVG               103023 non-null  float64
 48   COMMONAREA_AVG                92646 non-null   float64
 49   ELEVATORS_AVG                 143620 non-null  float64
 50   ENTRANCES_AVG                 152683 non-null  float64
 51   FLOORSMAX_AVG                 154491 non-null  float64
 52   FLOORSMIN_AVG                 98869 non-null   float64
```

```
# Shape of application dataset
app_data.shape
```

```
(307511, 122)
```

```
# Application data Description
app_data.describe()
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE	DAYS_
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	307511.000000	307511.0
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	0.020868	-16036.9
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	0.013831	4363.9
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	0.000290	-25229.0
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	0.010006	-19682.0
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	0.018850	-15750.0
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	0.028663	-12413.0
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.072508	-7489.0

```
#Checking Null values in the dataset and sorting in descending value to get better understanding of Null values
(app_data.isnull().sum()/len(app_data)*100).sort_values(ascending=False)
```

```
COMMONAREA_MEDI                69.872297
COMMONAREA_AVG                 69.872297
COMMONAREA_MODE                69.872297
NONLIVINGAPARTMENTS_MODE       69.432963
NONLIVINGAPARTMENTS_AVG        69.432963
NONLIVINGAPARTMENTS_MEDI       69.432963
FONDKAPREMONT_MODE             68.386172
```

```
LIVINGAPARTMENTS_MODE          68.354953
LIVINGAPARTMENTS_AVG           68.354953
LIVINGAPARTMENTS_MEDI          68.354953
FLOORSMIN_AVG                  67.848630
FLOORSMIN_MODE                 67.848630
FLOORSMIN_MEDI                 67.848630
YEARS_BUILD_MEDI               66.497784
YEARS_BUILD_MODE               66.497784
YEARS_BUILD_AVG                66.497784
OWN_CAR_AGE                    65.990810
LANDAREA_MEDI                  59.376738
LANDAREA_MODE                  59.376738
LANDAREA_AVG                   59.376738
BASEMENTAREA_MEDI              58.515956
BASEMENTAREA_AVG               58.515956
BASEMENTAREA_MODE              58.515956
EXT_SOURCE_1                   56.381073
NONLIVINGAREA_MODE             55.179164
NONLIVINGAREA_AVG              55.179164
NONLIVINGAREA_MEDI             55.179164
ELEVATORS_MEDI                 53.295980
ELEVATORS_AVG                  53.295980
ELEVATORS_MODE                 53.295980
WALLSMATERIAL_MODE             50.840783
APARTMENTS_MEDI                50.749729
APARTMENTS_AVG                 50.749729
APARTMENTS_MODE                50.749729
ENTRANCES_MEDI                 50.348768
ENTRANCES_AVG                  50.348768
ENTRANCES_MODE                 50.348768
LIVINGAREA_AVG                 50.193326
LIVINGAREA_MODE                50.193326
LIVINGAREA_MEDI                50.193326
HOUSETYPE_MODE                 50.176091
FLOORSMAX_MODE                 49.760822
FLOORSMAX_MEDI                 49.760822
FLOORSMAX_AVG                  49.760822
YEARS_BEGINEXPLUATATION_MODE   48.781019
YEARS_BEGINEXPLUATATION_MEDI   48.781019
YEARS_BEGINEXPLUATATION_AVG    48.781019
TOTALAREA_MODE                 48.268517
EMERGENCYSTATE_MODE            47.398304
OCCUPATION_TYPE                31.345545
EXT_SOURCE_3                   19.825307
AMT_REQ_CREDIT_BUREAU_HOUR     13.501631
AMT_REQ_CREDIT_BUREAU_DAY      13.501631
AMT_REQ_CREDIT_BUREAU_WEEK     13.501631
AMT_REQ_CREDIT_BUREAU_MON      13.501631
AMT_REQ_CREDIT_BUREAU_QRT      13.501631
AMT_REQ_CREDIT_BUREAU_YEAR     13.501631
NAME_TYPE_SUITE                 0.420148
```

Dealing with Null Values

Dropping the columns

The popular understanding that if a column has more than 50% of the data as null, we can delete that column. In this dataset, we can observe that the data with missing columns below 50% but above 40% belongs to the same kind of data (i.e mean,median,mode) of the building where the client live. So we can safely drop even these columns. Hence droping columns with more than 40% null values

```
appdata_nullvalues = (app_data.isnull().sum()/len(app_data)*100).sort_values(ascending=False)
appdata_nullvalmorethan40pct = appdata_nullvalues[appdata_nullvalues > 40].index
```

```
# Checking the shape of the dataframe of columns above 40% null values
appdata_nullvalmorethan40pct.shape
```

```
(49,)
```

```
# Droping the columns with more than 40% null values
app_data.drop(labels=appdata_nullvalmorethan40pct,axis=1,inplace=True)
```

```
# Checking the shape of the application dataset after droping the columns with above 40% null values
app_data.shape
```

```
(307511, 73)
```

```
app_data.isnull().sum().sort_values(ascending=False)
```

```
OCCUPATION_TYPE               96391
EXT_SOURCE_3                  60965
AMT_REQ_CREDIT_BUREAU_YEAR    41519
AMT_REQ_CREDIT_BUREAU_QRT     41519
AMT_REQ_CREDIT_BUREAU_MON     41519
AMT_REQ_CREDIT_BUREAU_WEEK    41519
AMT_REQ_CREDIT_BUREAU_DAY     41519
AMT_REQ_CREDIT_BUREAU_HOUR    41519
NAME_TYPE_SUITE                1292
OBS_30_CNT_SOCIAL_CIRCLE       1021
DEF_30_CNT_SOCIAL_CIRCLE       1021
OBS_60_CNT_SOCIAL_CIRCLE       1021
DEF_60_CNT_SOCIAL_CIRCLE       1021
EXT_SOURCE_2                    660
AMT_GOODS_PRICE                 278
AMT_ANNUITY                      12
CNT_FAM_MEMBERS                   2
```

```
DAYS_LAST_PHONE_CHANGE          1
FLAG_DOCUMENT_17                0
FLAG_DOCUMENT_18                0
FLAG_DOCUMENT_21                0
FLAG_DOCUMENT_20                0
FLAG_DOCUMENT_19                0
FLAG_DOCUMENT_2                 0
FLAG_DOCUMENT_3                 0
FLAG_DOCUMENT_4                 0
FLAG_DOCUMENT_5                 0
FLAG_DOCUMENT_16                0
FLAG_DOCUMENT_6                 0
FLAG_DOCUMENT_7                 0
FLAG_DOCUMENT_8                 0
FLAG_DOCUMENT_9                 0
FLAG_DOCUMENT_10                0
FLAG_DOCUMENT_11                0
ORGANIZATION_TYPE               0
FLAG_DOCUMENT_13                0
FLAG_DOCUMENT_14                0
FLAG_DOCUMENT_15                0
FLAG_DOCUMENT_12                0
SK_ID_CURR                      0
LIVE_CITY_NOT_WORK_CITY         0
DAYS_REGISTRATION               0
NAME_CONTRACT_TYPE              0
CODE_GENDER                     0
FLAG_OWN_CAR                    0
FLAG_OWN_REALTY                 0
CNT_CHILDREN                    0
AMT_INCOME_TOTAL                0
AMT_CREDIT                      0
NAME_INCOME_TYPE                0
NAME_EDUCATION_TYPE             0
NAME_FAMILY_STATUS              0
NAME_HOUSING_TYPE               0
REGION_POPULATION_RELATIVE      0
DAYS_BIRTH                      0
DAYS_EMPLOYED                   0
DAYS_ID_PUBLISH                 0
```

```
app_data.sample(5)
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	A
79301	191932	0	Cash loans	F	N	Y	0	225000.0	1120275.0	47596.5	
285873	431091	0	Cash loans	F	N	Y	0	135000.0	1453500.0	38470.5	
251942	391518	0	Cash loans	F	N	Y	0	99000.0	254700.0	14350.5	
187018	316805	0	Cash loans	M	N	Y	0	126000.0	277969.5	18706.5	
286275	431555	0	Cash loans	F	N	N	0	67500.0	178290.0	10233.0	

Now we can deal with other columns with less than 40% null values

OCCUPATION_TYPE column

Occupation_type column is a categorical column stating the Occupation of the customers and an important data point for the analysis to be done. Hence records cannot be dropped.

```
app_data.OCCUPATION_TYPE.value_counts(normalize=True,dropna=False)*100
```

```
NaN                      31.345545
Laborers                 17.946025
Sales staff              10.439301
Core staff                8.965533
Managers                  6.949670
Drivers                   6.049540
High skill tech staff     3.700681
Accountants               3.191105
Medicine staff            2.776161
Security staff            2.185613
Cooking staff             1.933589
Cleaning staff            1.513117
Private service staff     0.862408
Low-skill Laborers        0.680626
Waiters/barmen staff      0.438358
Secretaries               0.424375
Realty agents             0.244219
HR staff                  0.183083
IT staff                  0.171051
Name: OCCUPATION_TYPE, dtype: float64
```

As the null values comprises of over 30% of the data, we can impute the missing values. In this case as the data is categorical, we can create a separate category of Not Known for null values

```
app_data.OCCUPATION_TYPE.fillna("Not Known",inplace=True)
app_data.OCCUPATION_TYPE.value_counts(normalize=True,dropna=False)*100
```

```
Not Known                31.345545
Laborers                 17.946025
Sales staff              10.439301
Core staff                8.965533
Managers                  6.949670
Drivers                   6.049540
High skill tech staff     3.700681
Accountants               3.191105
Medicine staff            2.776161
Security staff            2.185613
Cooking staff             1.933589
Cleaning staff            1.513117
Private service staff     0.862408
Low-skill Laborers        0.680626
Waiters/barmen staff      0.438358
Secretaries               0.424375
Realty agents             0.244219
HR staff                  0.183083
IT staff                  0.171051
Name: OCCUPATION_TYPE, dtype: float64
```

EXT_SOURCE_3 and EXT_SOURCE_2 columns

Both are continuous variable columns, so we can impute the NaN values with either mean or median values.

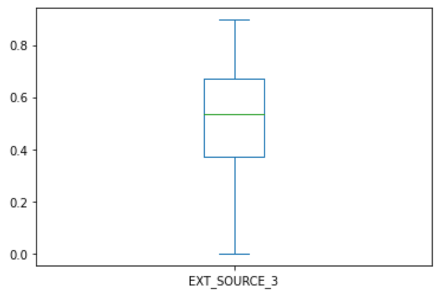
```
app_data.EXT_SOURCE_3.describe()
```

```
count    246546.000000
mean          0.510853
std           0.194844
min           0.000527
25%           0.370650
50%           0.535276
75%           0.669057
max           0.896010
Name: EXT_SOURCE_3, dtype: float64
```

```
# Checking for the null values in EXT_SOURCE_3
app_data.EXT_SOURCE_3.value_counts(normalize=True,dropna=False)*100
```

```
NaN         19.825307
0.746300     0.474780
0.713631     0.427627
0.694093     0.414945
0.670652     0.387303
               ...
0.028674     0.000325
0.025272     0.000325
0.021492     0.000325
0.014556     0.000325
0.043227     0.000325
Name: EXT_SOURCE_3, Length: 815, dtype: float64
```

```
#Checking for Outliers
app_data.EXT_SOURCE_3.plot.box()
plt.show()
```



Since the mean and median almost the same and no outliers, we can impute the mean value for the null values for EXT_SOURCE_3

```
# Imputating the null values with mean
app_data.EXT_SOURCE_3.fillna(app_data.EXT_SOURCE_3.mean(),inplace=True)
```

```
# Re-checking to ensure that there are no null values
app_data.EXT_SOURCE_3.isnull().sum()
```

```
0
```

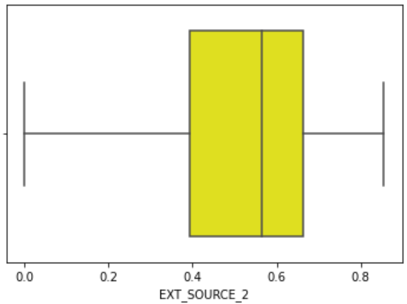
```
app_data.EXT_SOURCE_2.describe()
```

```
count    3.068510e+05
mean     5.143927e-01
std      1.910602e-01
min      8.173617e-08
25%      3.924574e-01
50%      5.659614e-01
75%      6.636171e-01
max      8.549997e-01
Name: EXT_SOURCE_2, dtype: float64
```

```
# Checking for the null values in EXT_SOURCE_2
app_data.EXT_SOURCE_2.value_counts(normalize=True,dropna=False)*100
```

```
0.285898    0.234463
NaN         0.214626
0.262258    0.135605
0.265256    0.111541
0.159679    0.104712
               ...
0.004725    0.000325
0.257313    0.000325
0.282030    0.000325
0.181540    0.000325
0.267834    0.000325
Name: EXT_SOURCE_2, Length: 119832, dtype: float64
```

```
# Checking for Outliers as EXT_SOURCE_2 is continuous variable
sns.boxplot(app_data.EXT_SOURCE_2,color="Yellow")
plt.show()
```



Since the mean and median are almost the same and the data is continuous variable, we can impute the mean/median values for the null values for EXT_SOURCE_2

```
# Imputing the null values with median values for EXT_SOURCE_2
app_data.EXT_SOURCE_2.fillna(app_data.EXT_SOURCE_2.median(),inplace=True)
```

```
# Re-checking to ensure that there are no null values
app_data.EXT_SOURCE_2.isnull().sum()
```

```
0
```

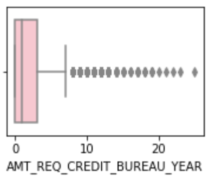
AMT_REQ_CREDIT_BUREAU YEAR, QRT, MON, WEEK, DAY, HOUR Columns

The data is these columns consists the number of Credit Bureau queries about the client before the loan application captured in different time frames

```
app_data.AMT_REQ_CREDIT_BUREAU_YEAR.describe()
```

```
count    265992.000000
mean          1.899974
std           1.869295
min           0.000000
25%           0.000000
50%           1.000000
75%           3.000000
max          25.000000
Name: AMT_REQ_CREDIT_BUREAU_YEAR, dtype: float64
```

```
# Checking for Outliers with Box plot
plt.figure(figsize=(3,2))
sns.boxplot(app_data.AMT_REQ_CREDIT_BUREAU_YEAR,color="Pink")
plt.show()
```

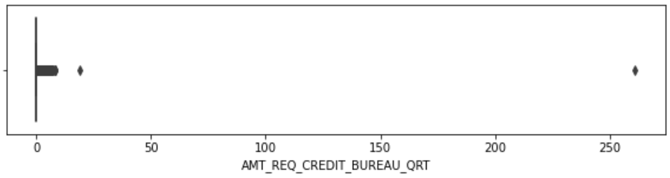


The above column has outliers, so we can impute the NaN values with median

```
app_data.AMT_REQ_CREDIT_BUREAU_QRT.describe()
```

```
count    265992.000000
mean          0.265474
std           0.794056
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max         261.000000
Name: AMT_REQ_CREDIT_BUREAU_QRT, dtype: float64
```

```
# Checking for Outliers with use of boxplot
plt.figure(figsize=(10,2))
sns.boxplot(app_data.AMT_REQ_CREDIT_BUREAU_QRT,color="Brown")
plt.show()
```

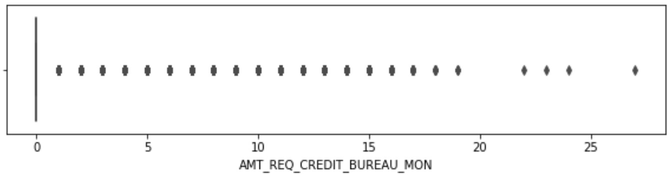


The above column has outliers, hence we can impute the median values in place of NaN values

```
app_data.AMT_REQ_CREDIT_BUREAU_MON.describe()
```

```
count    265992.000000
mean          0.267395
std           0.916002
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max          27.000000
Name: AMT_REQ_CREDIT_BUREAU_MON, dtype: float64
```

```
# Checking for Outliers with Boxplot
plt.figure(figsize=(10,2))
sns.boxplot(app_data.AMT_REQ_CREDIT_BUREAU_MON,color="Yellow")
plt.show()
```

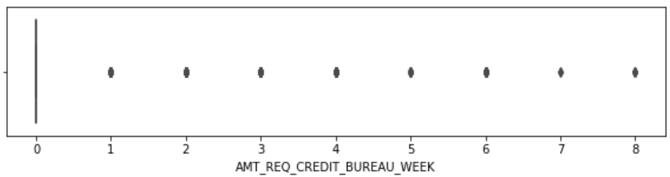


The above column has outliers, hence we can impute the median values in place of NaN values

```
app_data.AMT_REQ_CREDIT_BUREAU_WEEK.describe()
```

```
count    265992.000000
mean          0.034362
std           0.204685
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           8.000000
Name: AMT_REQ_CREDIT_BUREAU_WEEK, dtype: float64
```

```
# Checking for outliers with boxplot
plt.figure(figsize=(10,2))
sns.boxplot(app_data.AMT_REQ_CREDIT_BUREAU_WEEK,color="Yellow")
plt.show()
```

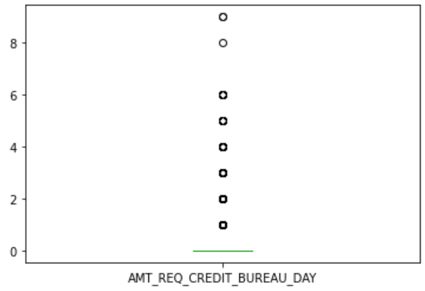


The above column has outliers, hence we can impute the median values in place of NaN values

```
app_data.AMT_REQ_CREDIT_BUREAU_DAY.describe()
```

```
count    265992.000000
mean          0.007000
std           0.110757
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           9.000000
Name: AMT_REQ_CREDIT_BUREAU_DAY, dtype: float64
```

```
# Checking for the outliers in the column with boxplot
app_data.AMT_REQ_CREDIT_BUREAU_DAY.plot.box()
plt.show()
```

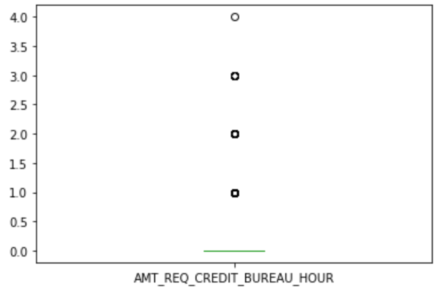


The above column has outliers, hence we can impute the median values in place of NaN values

```
app_data.AMT_REQ_CREDIT_BUREAU_HOUR.describe()
```

```
count    265992.000000
mean          0.006402
std           0.083849
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           4.000000
Name: AMT_REQ_CREDIT_BUREAU_HOUR, dtype: float64
```

```
#Checking for Outliers in the data column
app_data.AMT_REQ_CREDIT_BUREAU_HOUR.plot.box()
plt.show()
```



Since the data is continuous and haveing outliers, we can fill missing NaN with Median value for these columns

```
# Filling NaN values with median values for the above columns
app_data.AMT_REQ_CREDIT_BUREAU_YEAR.fillna(app_data.AMT_REQ_CREDIT_BUREAU_YEAR.median(),inplace=True)
app_data.AMT_REQ_CREDIT_BUREAU_QRT.fillna(app_data.AMT_REQ_CREDIT_BUREAU_QRT.median(),inplace=True)
app_data.AMT_REQ_CREDIT_BUREAU_MON.fillna(app_data.AMT_REQ_CREDIT_BUREAU_MON.median(),inplace=True)
app_data.AMT_REQ_CREDIT_BUREAU_WEEK.fillna(app_data.AMT_REQ_CREDIT_BUREAU_WEEK.median(),inplace=True)
app_data.AMT_REQ_CREDIT_BUREAU_DAY.fillna(app_data.AMT_REQ_CREDIT_BUREAU_DAY.median(),inplace=True)
app_data.AMT_REQ_CREDIT_BUREAU_HOUR.fillna(app_data.AMT_REQ_CREDIT_BUREAU_HOUR.median(),inplace=True)
```

```
# Checking the above code has worked and also to check columns with remaining NaN values
app_data.isnull().sum().sort_values(ascending=False)
```

```
NAME_TYPE_SUITE               1292
OBS_30_CNT_SOCIAL_CIRCLE      1021
DEF_30_CNT_SOCIAL_CIRCLE      1021
OBS_60_CNT_SOCIAL_CIRCLE      1021
DEF_60_CNT_SOCIAL_CIRCLE      1021
AMT_GOODS_PRICE                278
AMT_ANNUITY                     12
CNT_FAM_MEMBERS                  2
DAYS_LAST_PHONE_CHANGE           1
FLAG_DOCUMENT_4                  0
FLAG_DOCUMENT_8                  0
FLAG_DOCUMENT_7                  0
FLAG_DOCUMENT_6                  0
```

```
FLAG_DOCUMENT_5                 0
SK_ID_CURR                      0
FLAG_DOCUMENT_3                 0
FLAG_DOCUMENT_10                0
FLAG_DOCUMENT_2                 0
EXT_SOURCE_3                    0
FLAG_DOCUMENT_9                 0
FLAG_DOCUMENT_12                0
FLAG_DOCUMENT_11                0
FLAG_DOCUMENT_20                0
AMT_REQ_CREDIT_BUREAU_QRT       0
AMT_REQ_CREDIT_BUREAU_MON       0
AMT_REQ_CREDIT_BUREAU_WEEK      0
AMT_REQ_CREDIT_BUREAU_DAY       0
AMT_REQ_CREDIT_BUREAU_HOUR      0
FLAG_DOCUMENT_21                0
FLAG_DOCUMENT_19                0
ORGANIZATION_TYPE               0
FLAG_DOCUMENT_18                0
FLAG_DOCUMENT_17                0
FLAG_DOCUMENT_16                0
FLAG_DOCUMENT_15                0
FLAG_DOCUMENT_14                0
FLAG_DOCUMENT_13                0
EXT_SOURCE_2                    0
REG_CITY_NOT_LIVE_CITY          0
LIVE_CITY_NOT_WORK_CITY         0
NAME_INCOME_TYPE                0
DAYS_EMPLOYED                   0
DAYS_BIRTH                      0
REGION_POPULATION_RELATIVE      0
NAME_HOUSING_TYPE               0
NAME_FAMILY_STATUS              0
NAME_EDUCATION_TYPE             0
AMT_CREDIT                      0
REG_CITY_NOT_WORK_CITY          0
AMT_INCOME_TOTAL                0
CNT_CHILDREN                    0
FLAG_OWN_REALTY                 0
FLAG_OWN_CAR                    0
CODE_GENDER                     0
NAME_CONTRACT_TYPE              0
DAYS_REGISTRATION               0
DAYS_ID_PUBLISH                 0
FLAG_MOBIL                      0
```

NAME_TYPE_SUITE Column

This column captures the data of persons accompanying the customer while applying for the loan

```
# Checking values counts of the column
app_data.NAME_TYPE_SUITE.value_counts()
```

```
Unaccompanied      248526
Family              40149
Spouse, partner     11370
Children             3267
Other_B              1770
Other_A               866
Group of people       271
Name: NAME_TYPE_SUITE, dtype: int64
```

Since the column is categorical, and NaN values are less than 0.5%, we can add these values to the largest category which is Unaccompanied

```
# Since the majority is "Unaccompanied", we can fill the missing values with it
app_data.NAME_TYPE_SUITE.fillna("Unaccompanied",inplace=True)
```

```
app_data.isnull().sum().sort_values(ascending=False)
```

```
OBS_30_CNT_SOCIAL_CIRCLE       1021
DEF_30_CNT_SOCIAL_CIRCLE       1021
OBS_60_CNT_SOCIAL_CIRCLE       1021
DEF_60_CNT_SOCIAL_CIRCLE       1021
AMT_GOODS_PRICE                 278
AMT_ANNUITY                      12
CNT_FAM_MEMBERS                   2
DAYS_LAST_PHONE_CHANGE            1
FLAG_DOCUMENT_4                   0
FLAG_DOCUMENT_8                   0
FLAG_DOCUMENT_7                   0
FLAG_DOCUMENT_6                   0
FLAG_DOCUMENT_5                   0
SK_ID_CURR                        0
FLAG_DOCUMENT_3                   0
FLAG_DOCUMENT_2                   0
FLAG_DOCUMENT_10                  0
EXT_SOURCE_3                      0
EXT_SOURCE_2                      0
FLAG_DOCUMENT_9                   0
FLAG_DOCUMENT_11                  0
LIVE_CITY_NOT_WORK_CITY           0
FLAG_DOCUMENT_20                  0
AMT_REQ_CREDIT_BUREAU_QRT         0
```

```
AMT_REQ_CREDIT_BUREAU_MON        0
AMT_REQ_CREDIT_BUREAU_WEEK       0
AMT_REQ_CREDIT_BUREAU_DAY        0
AMT_REQ_CREDIT_BUREAU_HOUR       0
FLAG_DOCUMENT_21                 0
FLAG_DOCUMENT_19                 0
FLAG_DOCUMENT_12                 0
FLAG_DOCUMENT_18                 0
FLAG_DOCUMENT_17                 0
FLAG_DOCUMENT_16                 0
FLAG_DOCUMENT_15                 0
FLAG_DOCUMENT_14                 0
FLAG_DOCUMENT_13                 0
ORGANIZATION_TYPE                0
REG_CITY_NOT_LIVE_CITY           0
REG_CITY_NOT_WORK_CITY           0
NAME_TYPE_SUITE                  0
DAYS_BIRTH                       0
REGION_POPULATION_RELATIVE       0
NAME_HOUSING_TYPE                0
NAME_FAMILY_STATUS               0
NAME_EDUCATION_TYPE              0
NAME_INCOME_TYPE                 0
AMT_CREDIT                       0
TARGET                           0
AMT_INCOME_TOTAL                 0
CNT_CHILDREN                     0
FLAG_OWN_REALTY                  0
FLAG_OWN_CAR                     0
CODE_GENDER                      0
NAME_CONTRACT_TYPE               0
DAYS_EMPLOYED                    0
DAYS_REGISTRATION                0
```

OBS_30_CNT_SOCIAL_CIRCLE, DEF_30_CNT_SOCIAL_CIRCLE, OBS_60_CNT_SOCIAL_CIRCLE, DEF_60_CNT_SOCIAL_CIRCLE Columns

To start we can describe the above columns as they are similar in nature

```
app_data.OBS_30_CNT_SOCIAL_CIRCLE.describe()
```

```
count    306490.000000
mean          1.422245
std           2.400989
min           0.000000
25%           0.000000
50%           0.000000
75%           2.000000
max         348.000000
Name: OBS_30_CNT_SOCIAL_CIRCLE, dtype: float64
```

```
app_data.DEF_30_CNT_SOCIAL_CIRCLE.describe()
```

```
count    306490.000000
mean          0.143421
std           0.446698
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max          34.000000
Name: DEF_30_CNT_SOCIAL_CIRCLE, dtype: float64
```

```
app_data.OBS_60_CNT_SOCIAL_CIRCLE.describe()
```

```
count    306490.000000
mean          1.405292
std           2.379803
min           0.000000
25%           0.000000
50%           0.000000
75%           2.000000
max         344.000000
Name: OBS_60_CNT_SOCIAL_CIRCLE, dtype: float64
```

```
app_data.DEF_60_CNT_SOCIAL_CIRCLE.describe()
```

```
count    306490.000000
mean          0.100049
std           0.362291
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max          24.000000
Name: DEF_60_CNT_SOCIAL_CIRCLE, dtype: float64
```

Since the data is numerical, we can fill missing NaN with Median value for these columns

```
app_data.isnull().sum().sort_values(ascending=False)
```

```
OBS_30_CNT_SOCIAL_CIRCLE       1021
DEF_30_CNT_SOCIAL_CIRCLE       1021
```

```
OBS_60_CNT_SOCIAL_CIRCLE       1021
DEF_60_CNT_SOCIAL_CIRCLE       1021
AMT_GOODS_PRICE                 278
AMT_ANNUITY                      12
CNT_FAM_MEMBERS                   2
DAYS_LAST_PHONE_CHANGE            1
FLAG_DOCUMENT_4                   0
FLAG_DOCUMENT_8                   0
FLAG_DOCUMENT_7                   0
FLAG_DOCUMENT_6                   0
FLAG_DOCUMENT_5                   0
SK_ID_CURR                        0
FLAG_DOCUMENT_3                   0
FLAG_DOCUMENT_2                   0
FLAG_DOCUMENT_10                  0
EXT_SOURCE_3                      0
EXT_SOURCE_2                      0
FLAG_DOCUMENT_9                   0
FLAG_DOCUMENT_11                  0
LIVE_CITY_NOT_WORK_CITY           0
FLAG_DOCUMENT_20                  0
AMT_REQ_CREDIT_BUREAU_QRT         0
AMT_REQ_CREDIT_BUREAU_MON         0
AMT_REQ_CREDIT_BUREAU_WEEK        0
AMT_REQ_CREDIT_BUREAU_DAY         0
AMT_REQ_CREDIT_BUREAU_HOUR        0
FLAG_DOCUMENT_21                  0
FLAG_DOCUMENT_19                  0
FLAG_DOCUMENT_12                  0
FLAG_DOCUMENT_18                  0
FLAG_DOCUMENT_17                  0
FLAG_DOCUMENT_16                  0
FLAG_DOCUMENT_15                  0
FLAG_DOCUMENT_14                  0
FLAG_DOCUMENT_13                  0
ORGANIZATION_TYPE                 0
REG_CITY_NOT_LIVE_CITY            0
REG_CITY_NOT_WORK_CITY            0
NAME_TYPE_SUITE                   0
DAYS_BIRTH                        0
REGION_POPULATION_RELATIVE        0
NAME_HOUSING_TYPE                 0
NAME_FAMILY_STATUS                0
NAME_EDUCATION_TYPE               0
NAME_INCOME_TYPE                  0
AMT_CREDIT                        0
TARGET                            0
AMT_INCOME_TOTAL                  0
CNT_CHILDREN                      0
FLAG_OWN_REALTY                   0
FLAG_OWN_CAR                      0
CODE_GENDER                       0
NAME_CONTRACT_TYPE                0
DAYS_EMPLOYED                     0
DAYS_REGISTRATION                 0
DAYS_ID_PUBLISH                   0
```

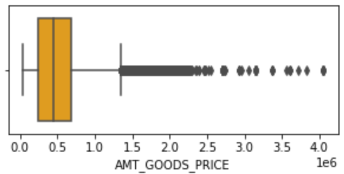
AMT_GOODS_PRICE Column

This column captures the price of the good for which the loan is applied for

```
# Checking the AMT_GOODS_PRICE column
app_data.AMT_GOODS_PRICE.describe()
```

```
count    3.072330e+05
mean     5.383962e+05
std      3.694465e+05
min      4.050000e+04
25%      2.385000e+05
50%      4.500000e+05
75%      6.795000e+05
max      4.050000e+06
Name: AMT_GOODS_PRICE, dtype: float64
```

```
# Checking for the outliers
plt.figure(figsize=(5,2))
sns.boxplot(app_data.AMT_GOODS_PRICE, color="Orange")
plt.show()
```



This data column has outliers, so we can impute the NaN values with median

```
# Filling Missing values with median of the column since it is a continuous data variable
app_data.AMT_GOODS_PRICE.fillna(app_data.AMT_GOODS_PRICE.median(),inplace=True)
```

```
# Checking for any NaN in the column
app_data.AMT_GOODS_PRICE.isnull().sum()
```

```
0
```

```
# Checking quatiles of the column to deal with outliers
app_data.AMT_GOODS_PRICE.quantile([0,0.2,0.4,0.6,0.8,1.0])
```

```
0.0      40500.0
0.2     225000.0
0.4     378000.0
0.6     522000.0
0.8     814500.0
1.0    4050000.0
Name: AMT_GOODS_PRICE, dtype: float64
```

```
# Creating a new dataframe with labels as per the quantiles which will be simpler to do analysis
app_data["AMOUNT_GOODSPRICE"] = pd.qcut(app_data.AMT_GOODS_PRICE,q=[0,0.2,0.4,0.6,0.8,1.0],labels=["Very Low","Low","Medium","High","Very High"])
app_data.head(1)
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	

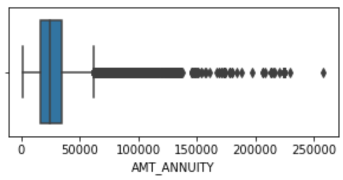
AMT_ANNUITY Column

This column captures the Loan Annuity

```
# Checking the column
app_data.AMT_ANNUITY.describe()
```

```
count    307499.000000
mean      27108.573909
std       14493.737315
min        1615.500000
25%       16524.000000
50%       24903.000000
75%       34596.000000
max      258025.500000
Name: AMT_ANNUITY, dtype: float64
```

```
# Checking for the outliers
plt.figure(figsize=(5,2))
sns.boxplot(app_data.AMT_ANNUITY)
plt.show()
```



Clearly the data column has outliers

```
# Since the missing values are very miniscule in count, we can either remove it or we can replace the data with median value
app_data.AMT_ANNUITY.fillna(app_data.AMT_ANNUITY.median(),inplace=True)
```

```
app_data.AMT_ANNUITY.isnull().sum()
```

```
0
```

```
app_data.AMT_ANNUITY.quantile([0,0.2,0.4,0.6,0.8,1.0])
```

```
0.0      1615.5
0.2     14701.5
0.4     21870.0
0.6     28062.0
0.8     37516.5
1.0    258025.5
Name: AMT_ANNUITY, dtype: float64
```

```
# Creating a new dataframe with labels as per the quantiles which will be simpler to do analysis
app_data["AMOUNT_ANNUITY"] = pd.qcut(app_data.AMT_ANNUITY,q=[0,0.2,0.4,0.6,0.8,1.0],labels=["Very Low","Low","Medium","High","Very High",])
```

CNT_FAM_MEMBERS column

This is column captures the data of no.of family members of the client

```
# Since only two record of the column is having NaN, checking those datapoints
app_data[app_data.CNT_FAM_MEMBERS.isnull()]
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AI
41982	148605	0	Revolving loans	M	N	Y	0	450000.0	675000.0	33750.0	
187348	317181	0	Revolving loans	F	N	Y	0	202500.0	585000.0	29250.0	

```
# Checking the median of the column
app_data.CNT_FAM_MEMBERS.median()
```

```
2.0
```

```
# As only two record have NaN values in this column, we can either delete the rows or we can replace the missing values with Median, it being a numeric col
# Replacing the NaN values with median
app_data.CNT_FAM_MEMBERS.fillna(app_data.CNT_FAM_MEMBERS.median(),inplace=True)
```

```
# Checking whether the above code has made the correct changes
app_data.iloc[[41982,187348]]
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AI
41982	148605	0	Revolving loans	M	N	Y	0	450000.0	675000.0	33750.0	
187348	317181	0	Revolving loans	F	N	Y	0	202500.0	585000.0	29250.0	

```
#Converting Family Members data type to integer from float as No of family members can not be in decimals
app_data.CNT_FAM_MEMBERS = app_data.CNT_FAM_MEMBERS.astype(int)
```

```
# Confirming to check the data type of the column after assigning it to int
app_data.CNT_FAM_MEMBERS.unique()
```

```
array([ 1,  2,  3,  4,  5,  6,  9,  7,  8, 10, 13, 14, 12, 20, 15, 16, 11])
```

DAYS_LAST_PHONE_CHANGE column

This column captures the data of days since last phone number was changed before the customer applied for the loan

```
# Checking the null values for the column
app_data[app_data.DAYS_LAST_PHONE_CHANGE.isnull()]
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AM
15709	118330	0	Cash loans	M	Y	Y	0	126000.0	278613.0	25911.0	

```
# Replacing null value with median
app_data.DAYS_LAST_PHONE_CHANGE.fillna(app_data.DAYS_LAST_PHONE_CHANGE.median(),inplace=True)
```

```
# Checking whether the changes to the null value made
app_data.iloc[[15709]]
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AM
15709	118330	0	Cash loans	M	Y	Y	0	126000.0	278613.0	25911.0	

```
# Changing data type of the column from float to int
app_data.DAYS_LAST_PHONE_CHANGE = app_data.DAYS_LAST_PHONE_CHANGE.astype(int)
```

```
# Confirming that all the null values has been dealt with
app_data.isnull().sum()
```

```
SK_ID_CURR                0
TARGET                    0
NAME_CONTRACT_TYPE        0
CODE_GENDER               0
FLAG_OWN_CAR              0
FLAG_OWN_REALTY           0
CNT_CHILDREN              0
AMT_INCOME_TOTAL          0
AMT_CREDIT                0
AMT_ANNUITY               0
AMT_GOODS_PRICE           0
NAME_TYPE_SUITE           0
NAME_INCOME_TYPE          0
NAME_EDUCATION_TYPE       0
NAME_FAMILY_STATUS        0
NAME_HOUSING_TYPE         0
```

```
REGION_POPULATION_RELATIVE        0
DAYS_BIRTH                        0
DAYS_EMPLOYED                     0
DAYS_REGISTRATION                 0
DAYS_ID_PUBLISH                   0
FLAG_MOBIL                        0
FLAG_EMP_PHONE                    0
FLAG_WORK_PHONE                   0
FLAG_CONT_MOBILE                  0
FLAG_PHONE                        0
FLAG_EMAIL                        0
OCCUPATION_TYPE                   0
CNT_FAM_MEMBERS                   0
REGION_RATING_CLIENT              0
REGION_RATING_CLIENT_W_CITY       0
WEEKDAY_APPR_PROCESS_START        0
HOUR_APPR_PROCESS_START           0
REG_REGION_NOT_LIVE_REGION        0
REG_REGION_NOT_WORK_REGION        0
LIVE_REGION_NOT_WORK_REGION       0
REG_CITY_NOT_LIVE_CITY            0
REG_CITY_NOT_WORK_CITY            0
LIVE_CITY_NOT_WORK_CITY           0
ORGANIZATION_TYPE                 0
EXT_SOURCE_2                      0
EXT_SOURCE_3                      0
OBS_30_CNT_SOCIAL_CIRCLE       1021
DEF_30_CNT_SOCIAL_CIRCLE       1021
OBS_60_CNT_SOCIAL_CIRCLE       1021
DEF_60_CNT_SOCIAL_CIRCLE       1021
DAYS_LAST_PHONE_CHANGE            0
FLAG_DOCUMENT_2                   0
FLAG_DOCUMENT_3                   0
FLAG_DOCUMENT_4                   0
FLAG_DOCUMENT_5                   0
FLAG_DOCUMENT_6                   0
FLAG_DOCUMENT_7                   0
FLAG_DOCUMENT_8                   0
FLAG_DOCUMENT_9                   0
FLAG_DOCUMENT_10                  0
FLAG_DOCUMENT_11                  0
FLAG_DOCUMENT_12                  0
```

Dealing with Negative values

```
app_data.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	

DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_ID_PUBLISH, DAYS_LAST_PHONE_CHANGE columns

It has been seen that these columns have negative values as the days calculated are in reverse i.e from the day of application. For analysis purpose, we need to change the data to absolute values and also adding a column of AGE by imputing the DAYS_BIRTH column

```
app_data.DAYS_BIRTH.describe()
```

```
count    307511.000000
mean     -16036.995067
std        4363.988632
min      -25229.000000
25%      -19682.000000
50%      -15750.000000
75%      -12413.000000
max       -7489.000000
Name: DAYS_BIRTH, dtype: float64
```

```
# Changing negative values to absolute values
# Adding Age column for better analysis
app_data.DAYS_BIRTH = abs(app_data.DAYS_BIRTH)
app_data["AGE"] = round(app_data.DAYS_BIRTH/365)
```

```
# Checking whether the Column "AGE" is created
app_data.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	

```
#Creating a separate dataframe with AGE_GROUP
app_data["AGE_GROUP"] = pd.cut(app_data.AGE,[0,30,40,50,60,100], labels=["<30","30-40","40-50","50-60","60+"] )
```

```
app_data.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	

```
app_data.DAYS_EMPLOYED.describe()
```

```
count    307511.000000
mean      63815.045904
std      141275.766519
min      -17912.000000
25%       -2760.000000
50%       -1213.000000
75%        -289.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

```
# Changing negative values to absolute values
app_data.DAYS_EMPLOYED = abs(app_data.DAYS_EMPLOYED)
```

```
app_data.DAYS_REGISTRATION.describe()
```

```
count    307511.000000
mean      -4986.120328
std        3522.886321
min      -24672.000000
25%       -7479.500000
50%       -4504.000000
75%       -2010.000000
max           0.000000
Name: DAYS_REGISTRATION, dtype: float64
```

```
# Changing negative values to absolute values
app_data.DAYS_REGISTRATION = abs(app_data.DAYS_REGISTRATION)
```

```
app_data.DAYS_ID_PUBLISH.describe()
```

```
count    307511.000000
mean      -2994.202373
std        1509.450419
min       -7197.000000
25%       -4299.000000
50%       -3254.000000
75%       -1720.000000
max           0.000000
Name: DAYS_ID_PUBLISH, dtype: float64
```

```
# Changing negative values to absolute values
app_data.DAYS_ID_PUBLISH = abs(app_data.DAYS_ID_PUBLISH)
```

```
app_data.DAYS_LAST_PHONE_CHANGE.describe()
```

```
count    307511.000000
mean       -962.858119
std         826.807226
min       -4292.000000
25%       -1570.000000
50%        -757.000000
75%        -274.000000
max           0.000000
Name: DAYS_LAST_PHONE_CHANGE, dtype: float64
```

```
# Changing negative values to absolute values
app_data.DAYS_LAST_PHONE_CHANGE = abs(app_data.DAYS_LAST_PHONE_CHANGE)
```

```
# Checking if the negative values has been converted
app_data.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	

Treating Outliers and any other anamolies in the dataset

```
# Getting list of columns which have dtype = object. We can check these categorical columns for any anamolies by reviewing their unique categories
list(set(app_data.columns) - set(app_data.describe()))
```

```
['AMOUNT_ANNUITY',
 'NAME_CONTRACT_TYPE',
 'NAME_HOUSING_TYPE',
 'CODE_GENDER',
 'NAME_INCOME_TYPE',
 'FLAG_OWN_CAR',
 'NAME_FAMILY_STATUS',
 'FLAG_OWN_REALTY',
 'NAME_EDUCATION_TYPE',
 'NAME_TYPE_SUITE',
 'ORGANIZATION_TYPE',
 'AGE_GROUP',
 'WEEKDAY_APPR_PROCESS_START',
 'AMOUNT_GOODSPRICE',
 'OCCUPATION_TYPE']
```

```
app_data.NAME_HOUSING_TYPE.unique() # Seems no anamolies
```

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

```
app_data.NAME_INCOME_TYPE.unique() #There is one category of Maternity leave which can not be a Income Type.
```

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

```
#Checking value counts of Income Types
app_data.NAME_INCOME_TYPE.value_counts() # Only 5 records have Maternity leave as category, we can put these records in Working as it is the largest catego
```

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

```
app_data.NAME_INCOME_TYPE.replace(["Maternity leave"],"Working", inplace=True)
```

```
# Confirming the replacement
app_data.NAME_INCOME_TYPE.value_counts() # The 5 records has been updated to Working
```

```
Working                 158779
Commercial associate     71617
Pensioner                55362
State servant            21703
Unemployed                  22
Student                     18
Businessman                 10
Name: NAME_INCOME_TYPE, dtype: int64
```

```
app_data.WEEKDAY_APPR_PROCESS_START.unique()# The weekdays are correct
```

```
array(['WEDNESDAY', 'MONDAY', 'THURSDAY', 'SUNDAY', 'SATURDAY', 'FRIDAY',
       'TUESDAY'], dtype=object)
```

```
app_data.NAME_TYPE_SUITE.unique() # Seems good
```

```
array(['Unaccompanied', 'Family', 'Spouse, partner', 'Children',
       'Other_A', 'Other_B', 'Group of people'], dtype=object)
```

```
app_data.NAME_FAMILY_STATUS.unique() # Seems good
```

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

```
# Checking unique variables in CODE_GENDER column
app_data.CODE_GENDER.unique()
```

```
array(['M', 'F', 'XNA'], dtype=object)
```

```
app_data.CODE_GENDER.value_counts()
```

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

```
#As only 4 records have XNA variable, we can replace it with its Mode, i.e F
app_data.CODE_GENDER.replace(["XNA"],app_data.CODE_GENDER.mode(),inplace=True)
```

```
#Confirmin if the replacement has been done
app_data.CODE_GENDER.value_counts()
```

```
F    202452
M    105059
Name: CODE_GENDER, dtype: int64
```

```
app_data.OCCUPATION_TYPE.unique() # Seems good
```

```
array(['Laborers', 'Core staff', 'Accountants', 'Managers', 'Not Known',
       'Drivers', 'Sales staff', 'Cleaning staff', 'Cooking staff',
       'Private service staff', 'Medicine staff', 'Security staff',
       'High skill tech staff', 'Waiters/barmen staff',
       'Low-skill Laborers', 'Realty agents', 'Secretaries', 'IT staff',
       'HR staff'], dtype=object)
```

```
app_data.ORGANIZATION_TYPE.unique()# Seems there is a category of XNA which needs to be dealt with
```

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

```
# Checking the percentage of the data in various categories in the data column
app_data.ORGANIZATION_TYPE.value_counts(normalize=True)*100
```

```
Business Entity Type 3    22.110429
XNA                       18.007161
Self-employed             12.491260
Other                      5.425172
Medicine                   3.639870
Business Entity Type 2     3.431747
Government                 3.383294
School                     2.891929
Trade: type 7              2.546576
Kindergarten               2.237318
Construction               2.185613
Business Entity Type 1     1.945947
Transport: type 4          1.755384
Trade: type 3              1.135569
Industry: type 9           1.095245
Industry: type 3           1.065978
Security                   1.055897
Housing                    0.961917
Industry: type 11          0.879318
Military                   0.856555
Bank                       0.815255
Agriculture                0.798020
Police                     0.761274
Transport: type 2          0.716722
Postal                     0.701438
Security Ministries        0.641928
Trade: type 2              0.617864
Restaurant                 0.588922
Services                   0.512177
University                 0.431529
Industry: type 7           0.425025
Transport: type 3          0.386002
```

```
Industry: type 1          0.337874
Hotel                     0.314135
Electricity               0.308932
Industry: type 4          0.285193
Trade: type 6             0.205196
Industry: type 5          0.194790
Insurance                 0.194139
Telecom                   0.187636
Emergency                 0.182107
Industry: type 2          0.148938
Advertising               0.139507
Realtor                   0.128776
Culture                   0.123248
Industry: type 12         0.119996
Trade: type 1             0.113167
Mobile                    0.103086
Legal Services            0.099183
Cleaning                  0.084550
Transport: type 1         0.065364
Industry: type 6          0.036421
Industry: type 10         0.035446
Religion                  0.027641
Industry: type 13         0.021788
Trade: type 4             0.020812
Trade: type 5             0.015934
Industry: type 8          0.007805
```

As over 18% of records are being marked XNA in ORGANIZATION_TYPE column, we can drop off the rows. But since we are not required to delete the records, we can either move this records by adding it to its mode value category or rename XNA as Unknown for better understanding. Adding the data to mode value category can influence the analysis due to its proportion, hence renaming XNA to Unknown

```
app_data.ORGANIZATION_TYPE.replace(["XNA"],"Unknown",inplace=True)
```

```
# Confirming the change
app_data.ORGANIZATION_TYPE.value_counts(normalize=True)*100
```

```
Business Entity Type 3    22.110429
Unknown                   18.007161
Self-employed             12.491260
Other                      5.425172
Medicine                   3.639870
Business Entity Type 2     3.431747
Government                 3.383294
School                     2.891929
Trade: type 7              2.546576
Kindergarten               2.237318
Construction               2.185613
Business Entity Type 1     1.945947
Transport: type 4          1.755384
Trade: type 3              1.135569
Industry: type 9           1.095245
Industry: type 3           1.065978
Security                   1.055897
Housing                    0.961917
Industry: type 11          0.879318
Military                   0.856555
Bank                       0.815255
Agriculture                0.798020
Police                     0.761274
Transport: type 2          0.716722
Postal                     0.701438
Security Ministries        0.641928
Trade: type 2              0.617864
Restaurant                 0.588922
Services                   0.512177
University                 0.431529
Industry: type 7           0.425025
Transport: type 3          0.386002
Industry: type 1           0.337874
Hotel                      0.314135
Electricity                0.308932
Industry: type 4           0.285193
Trade: type 6              0.205196
Industry: type 5           0.194790
Insurance                  0.194139
Telecom                    0.187636
Emergency                  0.182107
Industry: type 2           0.148938
Advertising                0.139507
Realtor                    0.128776
Culture                    0.123248
Industry: type 12          0.119996
Trade: type 1              0.113167
Mobile                     0.103086
Legal Services             0.099183
Cleaning                   0.084550
Transport: type 1          0.065364
Industry: type 6           0.036421
Industry: type 10          0.035446
Religion                   0.027641
Industry: type 13          0.021788
Trade: type 4              0.020812
Trade: type 5              0.015934
Industry: type 8           0.007805
```

```
app_data.FLAG_OWN_REALTY.unique() # Seems good
```

```
array(['Y', 'N'], dtype=object)
```

```
app_data.NAME_EDUCATION_TYPE.unique()# Seems good
```

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

```
app_data.FLAG_OWN_CAR.unique()# Seems good
```

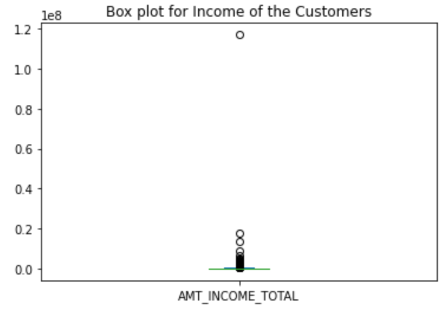
```
array(['N', 'Y'], dtype=object)
```

```
app_data.NAME_CONTRACT_TYPE.unique()# Seems good
```

```
array(['Cash loans', 'Revolving loans'], dtype=object)
```

AMT_INCOME_TOTAL column

```
# Checking for outliers
app_data.AMT_INCOME_TOTAL.plot.box()
plt.title("Box plot for Income of the Customers")
plt.show()
```




```
# Checking for the Quantile values of the data column
app_data.AMT_INCOME_TOTAL.quantile([0,0.2,0.4,0.6,0.8,1.0])
```

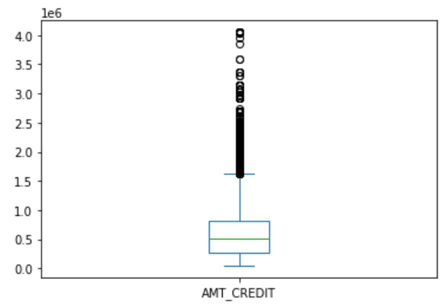
```
0.0        25650.0
0.2        99000.0
0.4       135000.0
0.6       162000.0
0.8       225000.0
1.0    117000000.0
Name: AMT_INCOME_TOTAL, dtype: float64
```

```
# Creating a Categorical datafrom of Customer Income for better analysis and deal with Outliers
app_data["CUST_INCOME"] = pd.qcut(app_data.AMT_INCOME_TOTAL,q=[0,0.2,0.4,0.6,0.8,1.0],labels=["Very Low","Low","Medium","High","Very High"])
app_data.head(1)
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	

AMT_CREDIT column

```
# Checking for Outliers
app_data.AMT_CREDIT.plot.box()
plt.show()
```




```
# Checking Quantile values for the Loan Amount
app_data.AMT_CREDIT.quantile([0,0.2,0.4,0.6,0.8,1.0])
```

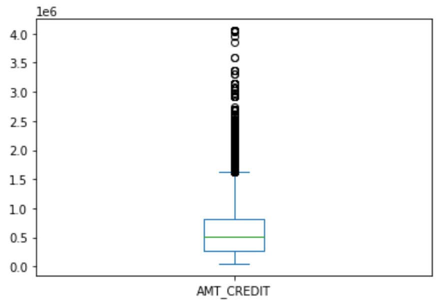
```
0.0      45000.0
0.2     254700.0
0.4     432000.0
0.6     604152.0
0.8     900000.0
1.0    4050000.0
Name: AMT_CREDIT, dtype: float64
```

```
# Creating a Categorical datafrom of Customer Income for better analysis and deal with Outliers
app_data["CUST_INCOME"] = pd.qcut(app_data.AMT_INCOME_TOTAL,q=[0,0.2,0.4,0.6,0.8,1.0],labels=["Very Low","Low","Medium","High","Very High"])
app_data.head(1)
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	

AMT_CREDIT columns

```
# Plotting Boxplot for the data column to check outlier
app_data.AMT_CREDIT.plot.box()
plt.show()
```




```
# Clearly the data column has outliers, we can deal with it by creating a new categorical column which will make the analysing more simpler
app_data["LOAN_AMT"] = pd.qcut(app_data.AMT_CREDIT,q=[0,0.2,0.4,0.6,0.8,1.0],labels=["Very Low","Low","Medium","High","Very High"])
app_data.head(1)
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	

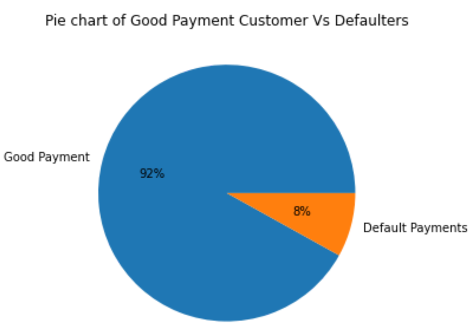
Data Analysis for Application Data

Now that we have clean the data, dealt with NaN values and Outliers, we can start the analysis of the Application Data

```
# Checking the data imbalance of the Target Column
app_data.TARGET.value_counts(normalize=True)*100
```

```
0    91.927118
1     8.072882
Name: TARGET, dtype: float64
```

```
plt.figure(figsize=(10,5))
plt.pie(app_data.TARGET.value_counts(normalize=True)*100,labels = ["Good Payment", "Default Payments"],autopct='%.f%%')
plt.title("Pie chart of Good Payment Customer Vs Defaulters")
plt.show()
```



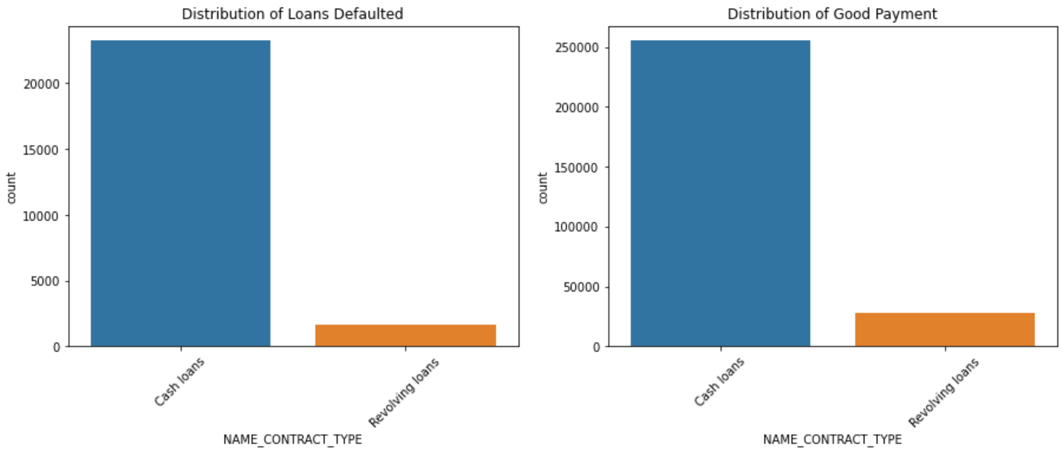
So, only 8% of the Target customers have the payment related issue, rest have been 92% are regular in their dues

```
# To make analysis more simple, we can create two dataframes; one with payment difficulties and other wiht no difficulty.
appdata_payment_issue = app_data[app_data.TARGET==1] # Dataframe for defaulters
appdata_good_payment = app_data[app_data.TARGET==0] # Dataframe for no payment issue
```

```
# Checking whether the data has been separted proportionately in the two dataframes
print("Customers with Payment Defaults (in %) - ",(appdata_payment_issue.value_counts().sum()/app_data.value_counts().sum())*100)
print("Customers with good payments (in %) - ",(appdata_good_payment.value_counts().sum()/app_data.value_counts().sum())*100)
```

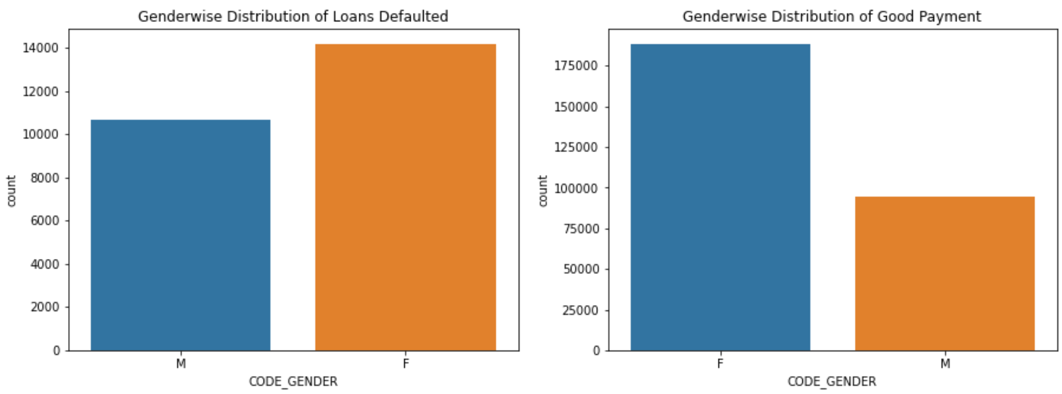
```
Customers with Payment Defaults (in %) -  8.088028973212829
Customers with good payments (in %) -  91.91197102678717
```

```
# Lets analyse whether type of loans has any relation to the payments of the loans
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
sns.countplot(x="NAME_CONTRACT_TYPE", data=appdata_payment_issue)
plt.title("Distribution of Loans Defaulted")
plt.xticks(rotation=45)

plt.subplot(1,2,2)
sns.countplot(x="NAME_CONTRACT_TYPE", data=appdata_good_payment)
plt.title("Distribution of Good Payment")
plt.xticks(rotation=45)

plt.show()
```



It can be observed that the Cash Loans which have fixed term and payment are mostly defaulted. Also from both the plots we can infer that Cash Loans are more preferred by banks than revolving (credit limit) loans

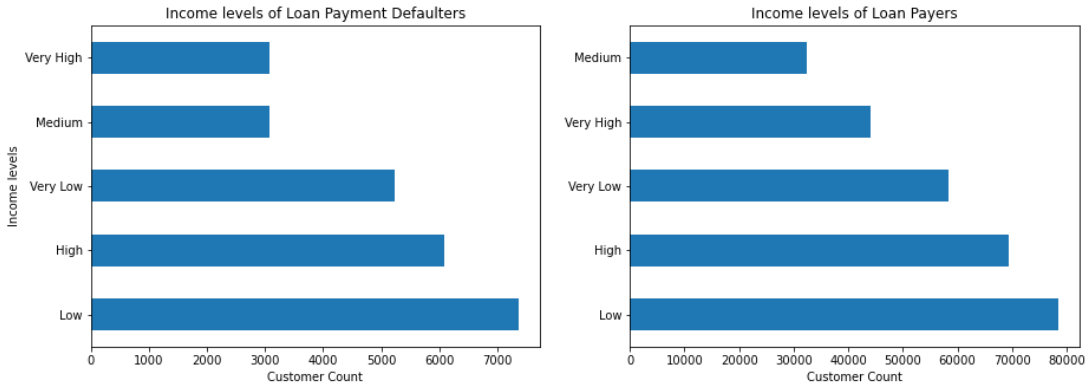
```
# Analysing whether Gender has any relation with the defaults in payments
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
sns.countplot(x="CODE_GENDER", data=appdata_payment_issue)
plt.title("Genderwise Distribution of Loans Defaulted")

plt.subplot(1,2,2)
sns.countplot(x="CODE_GENDER", data=appdata_good_payment)
plt.title("Genderwise Distribution of Good Payment")

plt.show()
```



It can observed that Female gender is more prone to default the payments but this scenario can be because the loans are been given to more number of females as compared to males

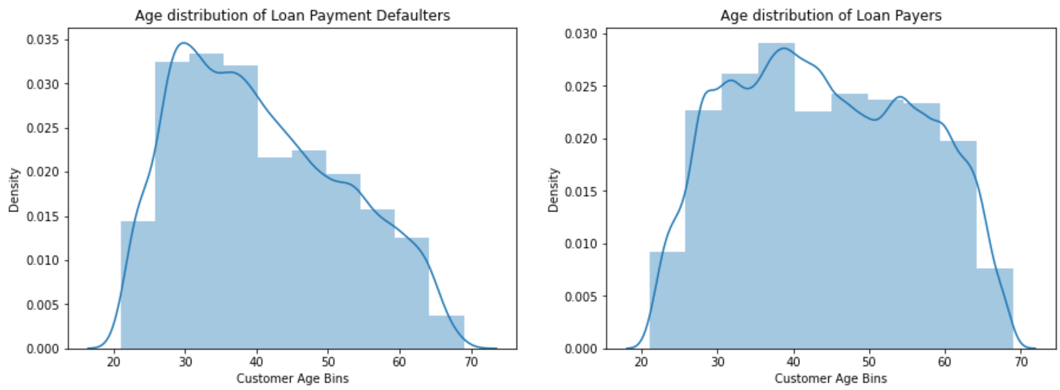
```
# Bar plot to understand Payment defaulter's and the Total Income
plt.figure(figsize= (15,5))
plt.subplot(1,2,1)
```

```
appdata_payment_issue.CUST_INCOME.value_counts().plot.barh()
plt.title("Income levels of Loan Payment Defaulters")
plt.xlabel("Customer Count")
plt.ylabel("Income levels")

plt.subplot(1,2,2)
appdata_good_payment.CUST_INCOME.value_counts().plot.barh()
plt.title("Income levels of Loan Payers")
plt.xlabel("Customer Count")


plt.show()
```

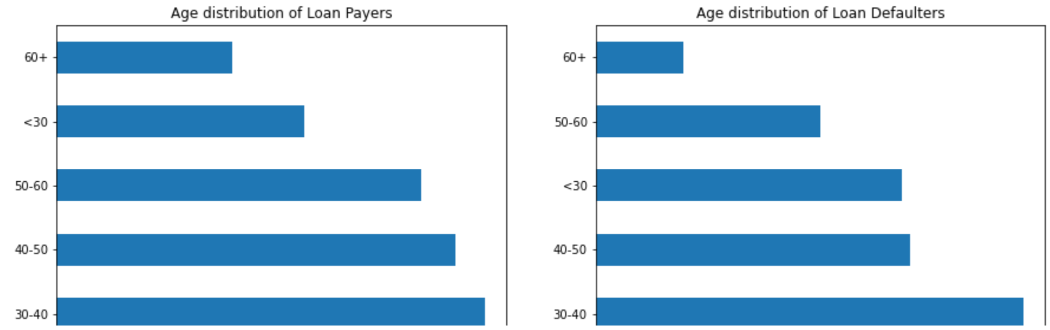


We can observe that Very Low, Low and High level of Income Customers are major defaulters. It is quite understandable that Low levels of Income can be a reason for defaults, but we need to dig deeper to understand the reason behind payment defaults by High Income Customers

```
# Dist plot to understand Payment defaulter's and their Age
plt.figure(figsize= (15,5))
plt.subplot(1,2,1)
sns.distplot(appdata_payment_issue.AGE,bins=10)
plt.title("Age distribution of Loan Payment Defaulters")
plt.xlabel("Customer Age Bins")


plt.subplot(1,2,2)
sns.distplot(appdata_good_payment.AGE,bins=10)
plt.title("Age distribution of Loan Payers")
plt.xlabel("Customer Age Bins")

plt.show()
```

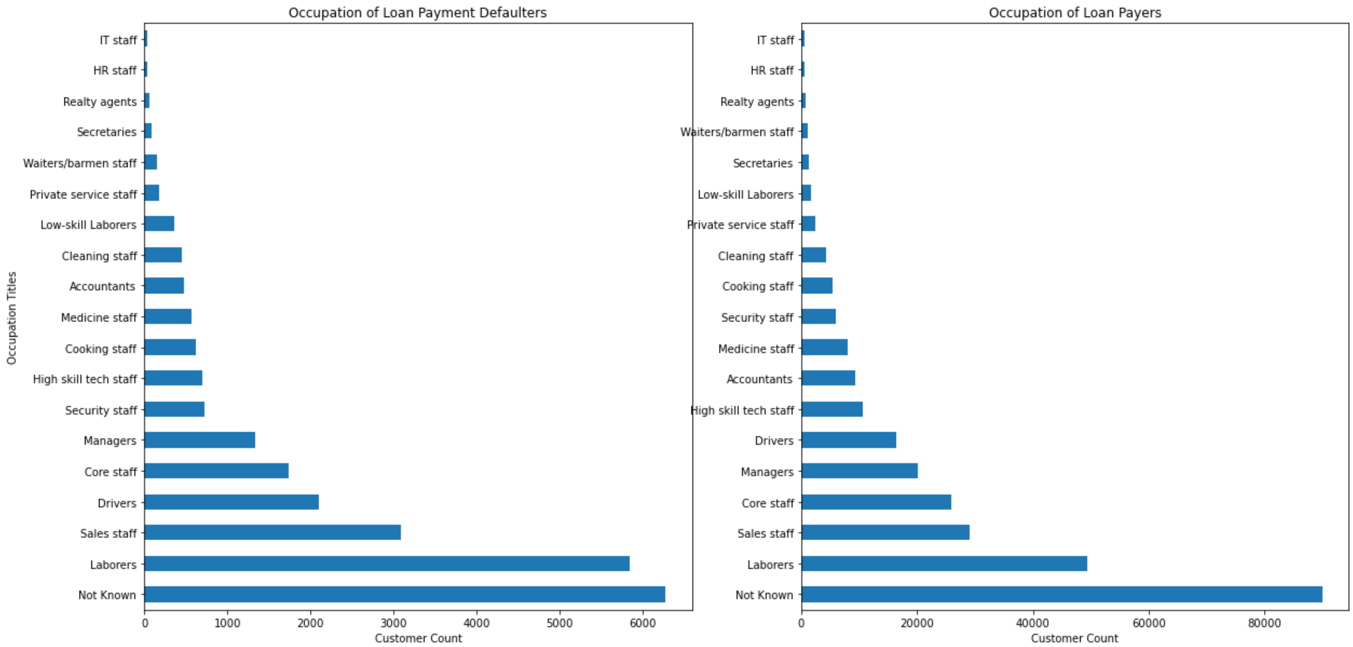



```
# Dist plot to understand Payment defaulter's and their Age GROUP
plt.figure(figsize= (15,5))
plt.subplot(1,2,1)

appdata_good_payment.AGE_GROUP.value_counts(normalize=True).plot.barh()
plt.title("Age distribution of Loan Payers")
plt.xlabel("Customer Age Bins")


plt.subplot(1,2,2)
appdata_payment_issue.AGE_GROUP.value_counts(normalize=True).plot.barh()
plt.title("Age distribution of Loan Defaulters")
plt.xlabel("Customer Age Bins")

plt.show()
```



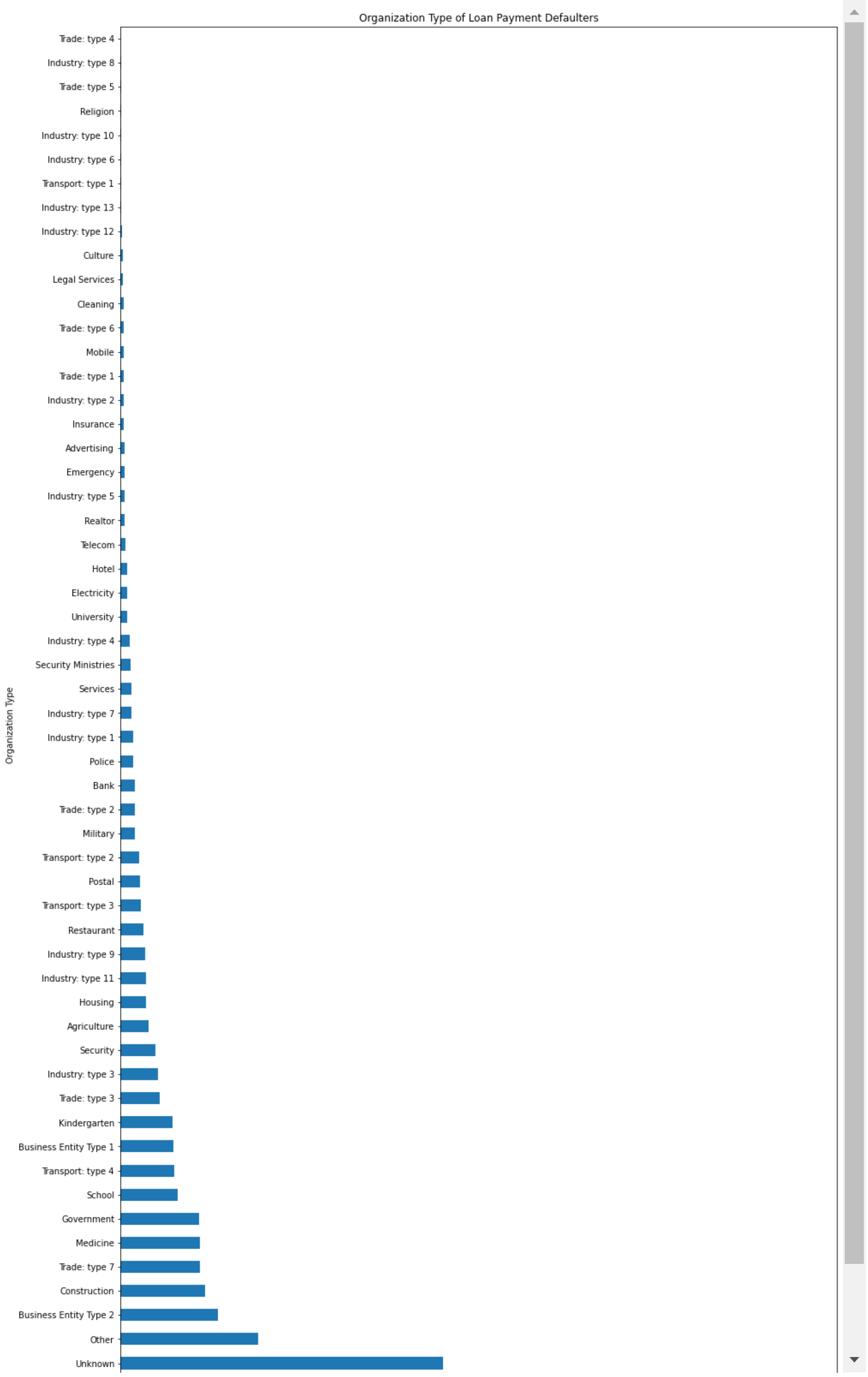
It can be observed from the distplot that age group of 30-50 are the mostly the defaulters. But with customers who pay loan on this age distribution is well proportioned

```
# Bar plot to understand Payment defaulter's and the Occupation
plt.figure(figsize= (20,10))
plt.subplot(1,2,1)
appdata_payment_issue.OCCUPATION_TYPE.value_counts().plot.barh()
plt.title("Occupation of Loan Payment Defaulters")
plt.xlabel("Customer Count")
plt.ylabel("Occupation Titles")

plt.subplot(1,2,2)
appdata_good_payment.OCCUPATION_TYPE.value_counts().plot.barh()
plt.title("Occupation of Loan Payers")
plt.xlabel("Customer Count")
plt.show()
```



We can observe that Laborers, Sales Staff, Drivers mostly the Very Low and Low Income people are major defaulters. But if we look at the Customers with no defaults, the same category of the Occupation Type customers are also major on time payers

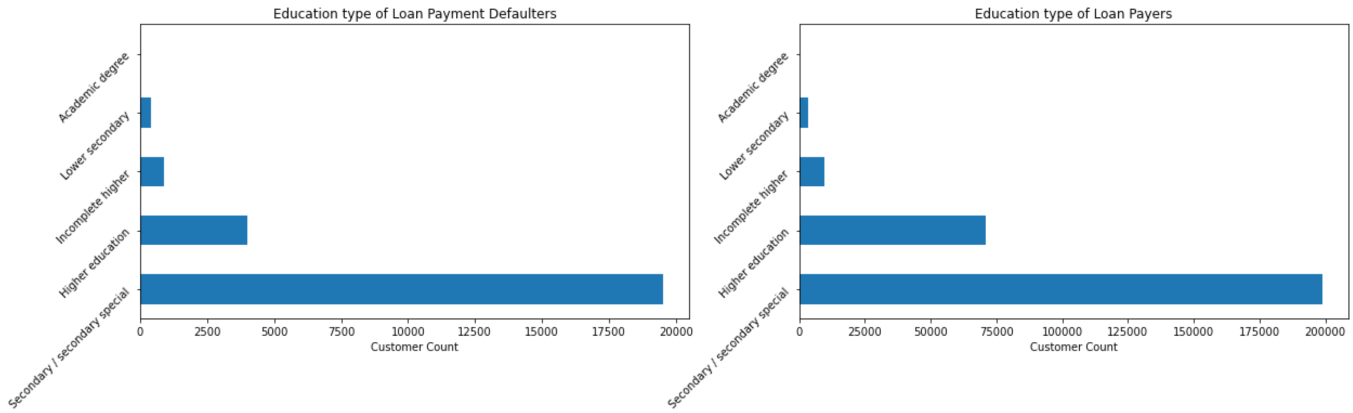
```
# Bar plot to understand Payment defaulter's and the Organization
plt.figure(figsize= (15,30))
appdata_payment_issue.ORGANIZATION_TYPE.value_counts().plot.barh()
plt.title("Organization Type of Loan Payment Defaulters")
plt.xlabel("Customer Count")
plt.ylabel("Organization Type")
plt.show()
```



We can infer that Business Entity Type - 3, Self Employed are major defaulters

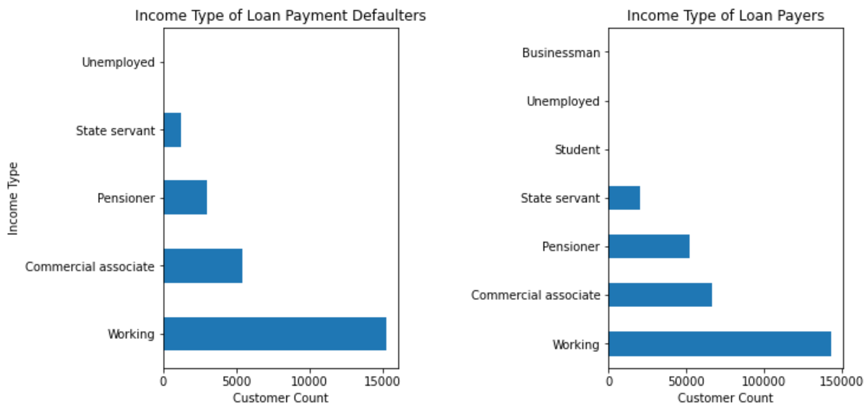
```
# Lets check whether what Education levels are more defaulters
# Bar plot to understand Payment defaulter's and the Education Type
plt.figure(figsize= (20,5))
plt.subplot(1,2,1)
appdata_payment_issue.NAME_EDUCATION_TYPE.value_counts().plot.barh()
plt.title("Education type of Loan Payment Defaulters")
plt.xlabel("Customer Count")

plt.yticks(rotation=45)
```

```
plt.subplot(1,2,2)
appdata_good_payment.NAME_EDUCATION_TYPE.value_counts().plot.barh()
plt.title("Education type of Loan Payers")
plt.xlabel("Customer Count")
plt.yticks(rotation=45)

plt.show()
```



We can infer that maximum defaulters are with secondary level education

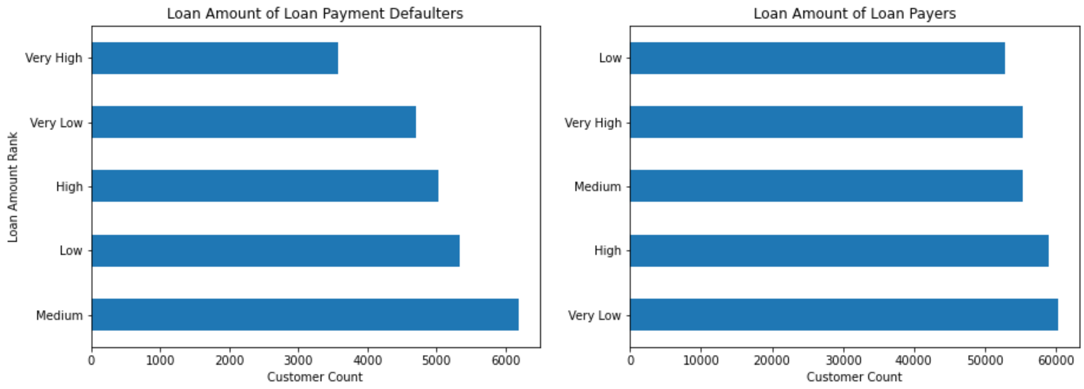
```
# Bar plot to understand Payment defaulter's and the Income Type
plt.figure(figsize= (10,5))
plt.subplot(1,2,1)
appdata_payment_issue.NAME_INCOME_TYPE.value_counts().plot.barh()
plt.title("Income Type of Loan Payment Defaulters")
plt.xlabel("Customer Count")
plt.ylabel("Income Type")

plt.subplot(1,2,2)
appdata_good_payment.NAME_INCOME_TYPE.value_counts().plot.barh()
plt.title("Income Type of Loan Payers")
plt.xlabel("Customer Count")

plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.9,
                    hspace=0.4)

plt.show()
```



It is a strange result that Working customers are mostly the defaulters. We need to further dig into this aspect to understand why a stable income customer is defaulter as against unemployed or Pensioners, who have low income/no income and yet are less in defaults

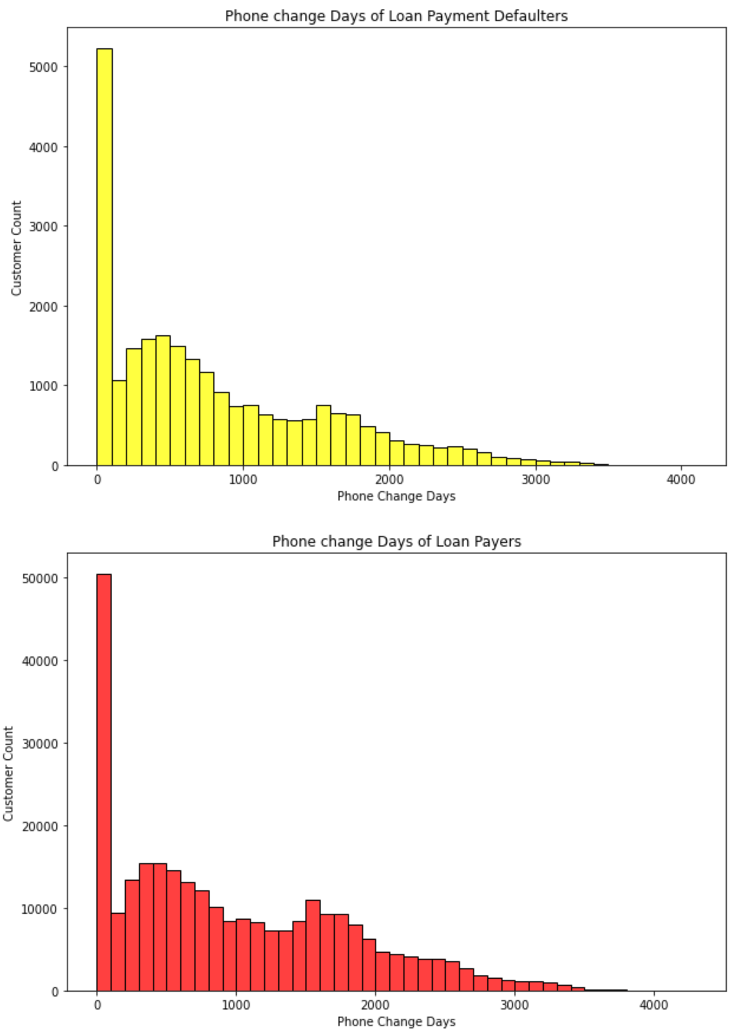
```
# Let's check the which Loan Amount category are majorly defaulted
plt.figure(figsize= (15,5))
plt.subplot(1,2,1)
appdata_payment_issue.LOAN_AMT.value_counts().plot.barh()
plt.title("Loan Amount of Loan Payment Defaulters")
plt.xlabel("Customer Count")
plt.ylabel("Loan Amount Rank")

plt.subplot(1,2,2)
appdata_good_payment.LOAN_AMT.value_counts().plot.barh()
```

```
plt.title("Loan Amount of Loan Payers")
plt.xlabel("Customer Count")

plt.show()
```



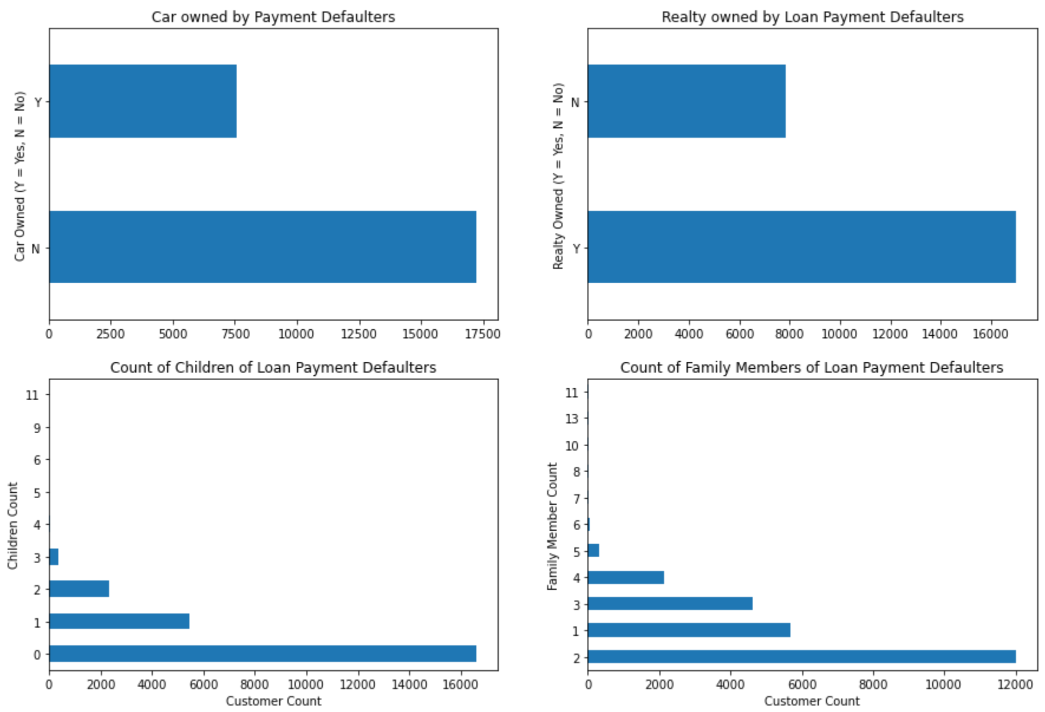
It can be observed that the customers with Medium and Low Loan Amount have defaulted more. Looking at the same scenario of Loan Payers, the loan of all the types are paid consistently

```
# Checking whether the defaulters do change their Phone Numbers
plt.figure(figsize= (10,15))
plt.subplot(2,1,1)
sns.histplot(x="DAYS_LAST_PHONE_CHANGE", data = appdata_payment_issue,binwidth=100,color="Yellow")
plt.title("Phone change Days of Loan Payment Defaulters")
plt.xlabel("Phone Change Days")
plt.ylabel("Customer Count")

plt.subplot(2,1,2)
sns.histplot(x="DAYS_LAST_PHONE_CHANGE", data = appdata_good_payment,binwidth=100,color="Red")
plt.title("Phone change Days of Loan Payers")
plt.xlabel("Phone Change Days")
plt.ylabel("Customer Count")

plt.show()
```



It can be said that the major of the defaulters have changed their Mobile Numbers 100 days prior to Application. But so does the Loan Payers have

```
# Checking with Subplots
plt.figure(figsize= (15,10))
plt.subplot(2,2,1)
appdata_payment_issue.FLAG_OWN_CAR.value_counts().plot.barh()
plt.title("Car owned by Payment Defaulters")

plt.ylabel("Car Owned (Y = Yes, N = No)")

plt.subplot(2,2,2)
appdata_payment_issue.FLAG_OWN_REALTY.value_counts().plot.barh()
plt.title("Realty owned by Loan Payment Defaulters")

plt.ylabel("Realty Owned (Y = Yes, N = No)")

plt.subplot(2,2,3)
appdata_payment_issue.CNT_CHILDREN.value_counts().plot.barh()
plt.title("Count of Children of Loan Payment Defaulters")
plt.xlabel("Customer Count")
plt.ylabel("Children Count")

plt.subplot(2,2,4)
appdata_payment_issue.CNT_FAM_MEMBERS.value_counts().plot.barh()
plt.title("Count of Family Members of Loan Payment Defaulters")
plt.xlabel("Customer Count")
plt.ylabel("Family Member Count")

plt.show()
```



We can infer that

Most of the defaulters owne realty but not car Most of the defaulters have no children and have only 2 members in the family i.e the family is small This clearly strengthen that since younger age group i.e 27-45 years of customers are defaulters, they are Low Income Customers with lesser no. of family members, belong to lower education levels, and Occupation type. Also most of the loans defaulted is for smaller amounts

Lets check the Previous Application CSV

```
# We have already uploaded the csv as pre_data
pre_data.head()
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WEEKDAY_APPR_PROCESS_START
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	17145.0	SATURDAY
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	607500.0	THURSDAY

```
# Let's check the data file
pre_data.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
```

```
pre_data.shape
```

```
(1670214, 37)
```

Dealing with Null Values in the data

```
# Checking the Null values in percentages in the data
(pre_data.isnull().sum()/len(pre_data)*100).sort_values(ascending=False)
```

```
RATE_INTEREST_PRIVILEGED       99.643698
RATE_INTEREST_PRIMARY          99.643698
AMT_DOWN_PAYMENT               53.636480
RATE_DOWN_PAYMENT              53.636480
NAME_TYPE_SUITE                49.119754
NFLAG_INSURED_ON_APPROVAL      40.298129
DAYS_TERMINATION               40.298129
DAYS_LAST_DUE                  40.298129
DAYS_LAST_DUE_1ST_VERSION      40.298129
DAYS_FIRST_DUE                 40.298129
DAYS_FIRST_DRAWING             40.298129
AMT_GOODS_PRICE                23.081773
AMT_ANNUITY                    22.286665
CNT_PAYMENT                    22.286366
PRODUCT_COMBINATION             0.020716
AMT_CREDIT                      0.000060
NAME_YIELD_GROUP                0.000000
NAME_PORTFOLIO                  0.000000
NAME_SELLER_INDUSTRY            0.000000
SELLERPLACE_AREA                0.000000
CHANNEL_TYPE                    0.000000
NAME_PRODUCT_TYPE               0.000000
SK_ID_PREV                      0.000000
NAME_GOODS_CATEGORY             0.000000
NAME_CLIENT_TYPE                0.000000
CODE_REJECT_REASON              0.000000
SK_ID_CURR                      0.000000
DAYS_DECISION                   0.000000
NAME_CONTRACT_STATUS            0.000000
NAME_CASH_LOAN_PURPOSE          0.000000
NFLAG_LAST_APPL_IN_DAY          0.000000
FLAG_LAST_APPL_PER_CONTRACT     0.000000
HOUR_APPR_PROCESS_START         0.000000
```

```
WEEKDAY_APPR_PROCESS_START      0.000000
AMT_APPLICATION                 0.000000
NAME_CONTRACT_TYPE              0.000000
NAME_PAYMENT_TYPE               0.000000
dtype: float64
```

```
#Assigning columns with more than 40% null values to a separate variable
predata_nullvalues = (pre_data.isnull().sum()/len(pre_data)*100).sort_values(ascending=False)
predata_nullvalmorethan40pct = predata_nullvalues[predata_nullvalues > 40].index
```

```
#Checking the shape of the null value variable
predata_nullvalmorethan40pct.shape
```

```
(11,)
```

```
predata_nullvalmorethan40pct
```

```
Index(['RATE_INTEREST_PRIVILEGED', 'RATE_INTEREST_PRIMARY', 'AMT_DOWN_PAYMENT',
       'RATE_DOWN_PAYMENT', 'NAME_TYPE_SUITE', 'NFLAG_INSURED_ON_APPROVAL',
       'DAYS_TERMINATION', 'DAYS_LAST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
       'DAYS_FIRST_DUE', 'DAYS_FIRST_DRAWING'],
      dtype='object')
```

```
# Droping the columns with more than 40% null values
pre_data.drop(labels=predata_nullvalmorethan40pct,axis=1,inplace=True)
```

```
# Checking shape of the dataframe after the dropped columns
pre_data.shape
```

```
(1670214, 26)
```

```
#Checking remaining Columns with Null Values
pre_data.isnull().sum().sort_values(ascending=False)
```

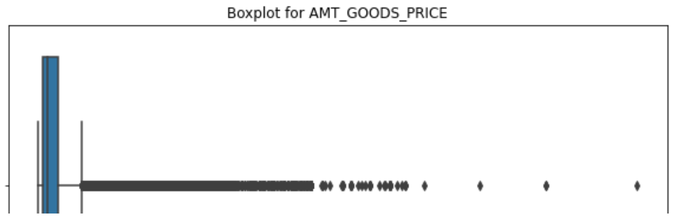
```
AMT_GOODS_PRICE                385515
AMT_ANNUITY                    372235
CNT_PAYMENT                    372230
PRODUCT_COMBINATION               346
AMT_CREDIT                          1
CODE_REJECT_REASON                  0
NAME_YIELD_GROUP                    0
NAME_SELLER_INDUSTRY                0
SELLERPLACE_AREA                    0
CHANNEL_TYPE                        0
NAME_PRODUCT_TYPE                   0
NAME_PORTFOLIO                      0
NAME_GOODS_CATEGORY                 0
NAME_CLIENT_TYPE                    0
SK_ID_PREV                          0
NAME_PAYMENT_TYPE                   0
SK_ID_CURR                          0
NAME_CONTRACT_STATUS                0
NAME_CASH_LOAN_PURPOSE              0
NFLAG_LAST_APPL_IN_DAY              0
FLAG_LAST_APPL_PER_CONTRACT         0
HOUR_APPR_PROCESS_START             0
WEEKDAY_APPR_PROCESS_START          0
AMT_APPLICATION                     0
NAME_CONTRACT_TYPE                  0
DAYS_DECISION                       0
dtype: int64
```

AMT_GOODS_PRICE column

```
pre_data.AMT_GOODS_PRICE.describe()
```

```
count    1.284699e+06
mean     2.278473e+05
std      3.153966e+05
min      0.000000e+00
25%      5.084100e+04
50%      1.123200e+05
75%      2.340000e+05
max      6.905160e+06
Name: AMT_GOODS_PRICE, dtype: float64
```

```
#Ploting the column
plt.figure(figsize= (10,5))
sns.boxplot(x="AMT_GOODS_PRICE",data=pre_data)
plt.title("Boxplot for AMT_GOODS_PRICE")
plt.show()
```



We can see that there are Outliers, we can fill the NaN values with median in this case. Also we can create a separate categorical column to avoid skewness in the analysis

```
# Fill Null values with median
pre_data.AMT_GOODS_PRICE.fillna(pre_data.AMT_GOODS_PRICE.median(), inplace=True)
```

```
# Checking if the column has any null values after the above code
pre_data.AMT_GOODS_PRICE.isnull().sum()
```

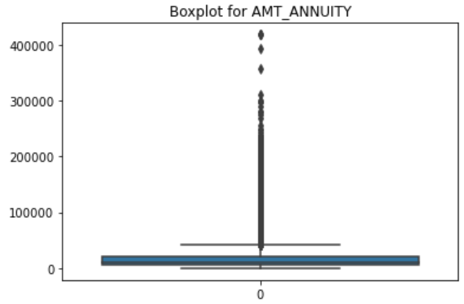
```
0
```

```
# Creating a separate Dataframe
pre_data["AMOUNT_GOODSPRICE"] = pd.qcut(pre_data.AMT_GOODS_PRICE,q=[0,0.2,0.4,0.6,0.8,1.0],labels=["Very Low","Low","Medium","High"],duplicates="drop")
pre_data.head(1)
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	WEEKDAY_APPR_PROCESS_START	HOUR_APPR_PROCES
0	2030495	271877	Consumer loans	1730.43	17145.0	17145.0	17145.0	SATURDAY	

AMT_ANNUITY column

```
# Checking for Outliers thorugh Boxplot
sns.boxplot(data=pre_data.AMT_ANNUITY)
plt.title("Boxplot for AMT_ANNUITY")
plt.show()
```



This column has many outliers, we can replace NaN values with median of the data series and for Outliers, we can create a separate data column with categories

```
# Filling the Null values with median value
pre_data.AMT_ANNUITY = pre_data.AMT_ANNUITY.fillna(pre_data.AMT_ANNUITY.median())
```

```
# Checking to ensure that there are no null values remaining
pre_data.AMT_ANNUITY.isnull().sum()
```

```
0
```

```
# Creating a separate categorical column of AMT_ANNUITY
pre_data.AMOUNT_ANNUITY = pd.qcut(pre_data.AMT_ANNUITY,q=[0,0.2,0.4,0.6,0.8,1],labels=["Very Low","Low","Medium","High"],duplicates="drop")
pre_data.head(1)
```

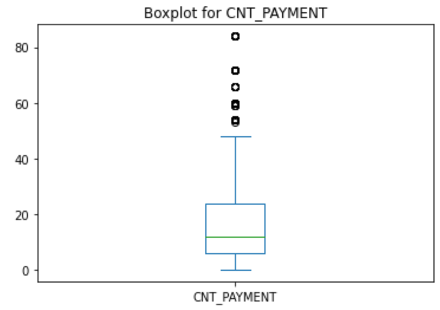

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	WEEKDAY_APPR_PROCESS_START	HOUR_APPR_PROCES
0	2030495	271877	Consumer loans	1730.43	17145.0	17145.0	17145.0	SATURDAY	

CNT_PAYMENT column

```
#Checking the Null count
pre_data.CNT_PAYMENT.isnull().value_counts(normalize=True)
```

```
False    0.777136
True     0.222864
Name: CNT_PAYMENT, dtype: float64
```

```
# Checking for Outliers with boxplot
pre_data.CNT_PAYMENT.plot.box()
plt.title("Boxplot for CNT_PAYMENT")
plt.show()
```



This data column has outliers so we can safely replace median values with the missing values, also we are going to keep the outliers in this data column as it is

```
# Filling null values with median as there are outliers present in the data
pre_data.CNT_PAYMENT.fillna(pre_data.CNT_PAYMENT.median(),inplace=True)
```

PRODUCT_COMBINATION column

```
# Checking the total null values
pre_data.PRODUCT_COMBINATION.isnull().sum()
```

```
346
```

```
# Value counts of the categories
pre_data.PRODUCT_COMBINATION.value_counts()
```

```
Cash                              285990
POS household with interest       263622
POS mobile with interest          220670
Cash X-Sell: middle               143883
Cash X-Sell: low                  130248
Card Street                       112582
POS industry with interest         98833
POS household without interest     82908
Card X-Sell                        80582
Cash Street: high                  59639
Cash X-Sell: high                  59301
Cash Street: middle                34658
Cash Street: low                   33834
POS mobile without interest        24082
POS other with interest            23879
POS industry without interest      12602
POS others without interest         2555
Name: PRODUCT_COMBINATION, dtype: int64
```

```
# As this column is categorical, we can fill null values with the mode i.e Cash
pre_data.PRODUCT_COMBINATION.fillna("Cash",inplace=True)
```

```
# Checking the null values in the data column
pre_data.PRODUCT_COMBINATION.isnull().sum()
```

```
0
```

AMT_CREDIT column

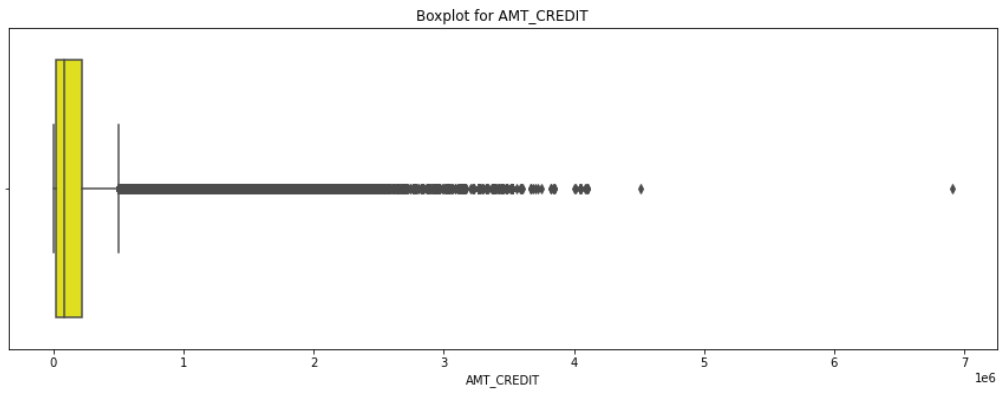
Only one value of it is null values, we can replace it with mean value and check for outliers

```
# Replace the null value with mean
pre_data.AMT_CREDIT.fillna(pre_data.AMT_CREDIT.mean(),inplace=True)
```

```
# Checking for null values after running the code to confirm the change
pre_data.AMT_CREDIT.isnull().sum()
```

```
0
```

```
# Checking the Outliers
plt.figure(figsize=(15,5))
sns.boxplot(x="AMT_CREDIT",data=pre_data,color="Yellow")
plt.title("Boxplot for AMT_CREDIT")
plt.show()
```



There are quite number of outliers in this dataframe, we either the delete the datapoints or we can create a separate categorical dataframe with its quantile values

```
# Creating a separate dataframe
pre_data["CREDIT_AMOUNT"] = pd.qcut(x=pre_data.AMT_CREDIT,q=[0,0.2,0.4,0.6,0.8,1],labels=["Low","Medium","High","Very High"],duplicates="drop")
pre_data.head(1)
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	WEEKDAY_APPR_PROCESS_START	HOUR_APPR_PROCES
0	2030495	271877	Consumer loans	1730.43	17145.0	17145.0	17145.0	SATURDAY	

```
# Ensuring that all Null values are dealt with
pre_data.isnull().sum()
```

```
SK_ID_PREV                     0
SK_ID_CURR                     0
NAME_CONTRACT_TYPE             0
AMT_ANNUITY                    0
AMT_APPLICATION                0
AMT_CREDIT                     0
AMT_GOODS_PRICE                0
WEEKDAY_APPR_PROCESS_START     0
HOUR_APPR_PROCESS_START        0
FLAG_LAST_APPL_PER_CONTRACT    0
NFLAG_LAST_APPL_IN_DAY         0
NAME_CASH_LOAN_PURPOSE         0
NAME_CONTRACT_STATUS           0
DAYS_DECISION                  0
NAME_PAYMENT_TYPE              0
CODE_REJECT_REASON             0
NAME_CLIENT_TYPE               0
NAME_GOODS_CATEGORY            0
NAME_PORTFOLIO                 0
NAME_PRODUCT_TYPE              0
CHANNEL_TYPE                   0
SELLERPLACE_AREA               0
NAME_SELLER_INDUSTRY           0
CNT_PAYMENT                    0
NAME_YIELD_GROUP               0
PRODUCT_COMBINATION            0
AMOUNT_GOODSPRICE              0
CREDIT_AMOUNT                  0
dtype: int64
```

```
# Checking the categorical columns
list(set(pre_data.columns) - set(pre_data.describe()))
```

```
['NAME_CASH_LOAN_PURPOSE',
 'FLAG_LAST_APPL_PER_CONTRACT',
 'NAME_CONTRACT_TYPE',
 'NAME_SELLER_INDUSTRY',
 'CREDIT_AMOUNT',
 'NAME_PRODUCT_TYPE',
 'NAME_GOODS_CATEGORY',
 'PRODUCT_COMBINATION',
 'NAME_PORTFOLIO',
 'NAME_CLIENT_TYPE',
 'NAME_CONTRACT_STATUS',
 'CHANNEL_TYPE',
 'WEEKDAY_APPR_PROCESS_START',
 'NAME_YIELD_GROUP',
 'CODE_REJECT_REASON',
 'AMOUNT_GOODSPRICE',
 'NAME_PAYMENT_TYPE']
```

NAME_PORTFOLIO column

```
# Checking the Null values in the column
pre_data.NAME_PORTFOLIO.value_counts(normalize=True)*100
```

```
POS      41.372603
Cash     27.634962
XNA      22.286366
Cards     8.680624
Cars      0.025446
Name: NAME_PORTFOLIO, dtype: float64
```

This column has over 22% data as XNA. We could replace it with mode but that will heavily skew our analysis towards POS. We can delete the datapoints with XNA but it is not required as per the Case Study requirement. So we will keep the data as it is but will change its category to Unknown for better readibility. Also the it seems that there has been error while capturing the the data as we have category as Cars. It seems that the data should have been captured in Cards. So we are making these changes in this column

```
pre_data.NAME_PORTFOLIO.replace(["XNA","Cars"],["Unknown","Cards"],inplace=True)
```

```
# Confirming the changes has been done
pre_data.NAME_PORTFOLIO.value_counts(normalize=True)*100
```

```
POS        41.372603
Cash       27.634962
Unknown    22.286366
Cards       8.706070
Name: NAME_PORTFOLIO, dtype: float64
```

NAME_CLIENT_TYPE column

```
# Checking for Null values
pre_data.NAME_CLIENT_TYPE.value_counts()
# We can see that the small number of values are Null, we can safely replace it with mode i.e Repeater
```

```
Repeater     1231261
New           301363
Refreshed     135649
XNA             1941
Name: NAME_CLIENT_TYPE, dtype: int64
```

```
# Confirming the changes
pre_data.NAME_CLIENT_TYPE.value_counts()
```

```
Repeater     1233202
New           301363
Refreshed     135649
Name: NAME_CLIENT_TYPE, dtype: int64
```

NAME_YIELD_GROUP column

```
pre_data.NAME_YIELD_GROUP.value_counts()
```

```
XNA           517215
middle        385532
high          353331
low_normal    322095
low_action     92041
Name: NAME_YIELD_GROUP, dtype: int64
```

Since XNA is the largest category, we can simply rename it to Unknown for better readibility

```
pre_data.NAME_YIELD_GROUP.replace(["XNA"],"Unknown",inplace=True)
```

```
pre_data.NAME_YIELD_GROUP.value_counts()
```

```
Unknown       517215
middle        385532
high          353331
low_normal    322095
low_action     92041
Name: NAME_YIELD_GROUP, dtype: int64
```

```
pre_data.FLAG_LAST_APPL_PER_CONTRACT.unique() # Seems correct
```

```
array(['Y', 'N'], dtype=object)
```

NAME_GOODS_CATEGORY column

```
# Checking for Null values in this categorical column
pre_data.NAME_GOODS_CATEGORY.value_counts()
```

```
XNA                         950809
Mobile                      224708
Consumer Electronics        121576
Computers                   105769
Audio/Video                  99441
Furniture                    53656
Photo / Cinema Equipment     25021
Construction Materials       24995
Clothing and Accessories     23554
Auto Accessories              7381
Jewelry                       6290
Homewares                     5023
Medical Supplies              3843
Vehicles                      3370
Sport and Leisure             2981
Gardening                     2668
Other                         2554
Office Appliances             2333
Tourism                       1659
Medicine                      1550
Direct Sales                   446
Fitness                        209
Additional Service             128
Education                      107
Weapon                          77
Insurance                       64
Animals                          1
House Construction               1
Name: NAME_GOODS_CATEGORY, dtype: int64
```

```
# Replacing the null values to Unknown as it is the largest proportion of this data column is Null and it cannot be deleted
pre_data.NAME_GOODS_CATEGORY.replace(["XNA"],"Unknown",inplace=True)
```

```
pre_data.NAME_GOODS_CATEGORY.unique() # XNA has been replaced
```

```
array(['Mobile', 'Unknown', 'Consumer Electronics',
       'Construction Materials', 'Auto Accessories',
       'Photo / Cinema Equipment', 'Computers', 'Audio/Video', 'Medicine',
       'Clothing and Accessories', 'Furniture', 'Sport and Leisure',
       'Homewares', 'Gardening', 'Jewelry', 'Vehicles', 'Education',
       'Medical Supplies', 'Other', 'Direct Sales', 'Office Appliances',
       'Fitness', 'Tourism', 'Insurance', 'Additional Service', 'Weapon',
       'Animals', 'House Construction'], dtype=object)
```

NAME_PRODUCT_TYPE column

```
# Checking this column for null values
pre_data.NAME_PRODUCT_TYPE.value_counts(normalize=True)*100
```

```
XNA        63.684414
x-sell     27.319074
walk-in     8.996512
Name: NAME_PRODUCT_TYPE, dtype: float64
```

```
# This column has 63% data as Null, we will keep the data with replacing XNA with Not Known
pre_data.NAME_PRODUCT_TYPE.replace(["XNA"],"Not Known",inplace=True)
```

```
# Checking unique values to ascertain the replacement has been done
pre_data.NAME_PRODUCT_TYPE.unique()
```

```
array(['Not Known', 'x-sell', 'walk-in'], dtype=object)
```

NAME_PAYMENT_TYPE column

```
pre_data.NAME_PAYMENT_TYPE.value_counts(normalize=True)*100
```

```
Cash through the bank                        61.881412
XNA                                          37.563091
Non-cash from your account                    0.490536
Cashless from the account of the employer     0.064962
Name: NAME_PAYMENT_TYPE, dtype: float64
```

```
# As done above, we will replace the XNA with Not Known
pre_data.NAME_PAYMENT_TYPE.replace(["XNA"],"Not Known",inplace=True)
```

```
pre_data.NAME_PAYMENT_TYPE.unique() # Checking that the null values are replaced
```

```
array(['Cash through the bank', 'Not Known', 'Non-cash from your account',
       'Cashless from the account of the employer'], dtype=object)
```

NAME_SELLER_INDUSTRY column

```
# Checking for null values in the column
pre_data.NAME_SELLER_INDUSTRY.value_counts(normalize=True)*100
```

```
XNA                     51.234153
Consumer electronics    23.845148
Connectivity            16.526565
Furniture                3.463568
Construction             1.783065
Clothing                 1.433888
Industry                 1.149194
Auto technology          0.298764
Jewelry                  0.162195
MLM partners             0.072745
Tourism                  0.030715
Name: NAME_SELLER_INDUSTRY, dtype: float64
```

```
# Since the null values are 51% of the data column, we will replace it with Unknown for better readibility
pre_data.NAME_SELLER_INDUSTRY.replace(["XNA"],"Unknown",inplace=True)
```

```
pre_data.NAME_SELLER_INDUSTRY.value_counts(normalize=True)*100
# Checking the replacement
```

```
Unknown                 51.234153
Consumer electronics    23.845148
Connectivity            16.526565
Furniture                3.463568
Construction             1.783065
Clothing                 1.433888
Industry                 1.149194
Auto technology          0.298764
Jewelry                  0.162195
MLM partners             0.072745
Tourism                  0.030715
Name: NAME_SELLER_INDUSTRY, dtype: float64
```

```
pre_data.CHANNEL_TYPE.unique() # No null values
```

```
array(['Country-wide', 'Contact center', 'Credit and cash offices',
       'Stone', 'Regional / Local', 'AP+ (Cash loan)',
       'Channel of corporate sales', 'Car dealer'], dtype=object)
```

NAME_CONTRACT_TYPE column

```
pre_data.NAME_CONTRACT_TYPE.value_counts()
```

```
Cash loans         747553
Consumer loans     729151
Revolving loans    193164
XNA                   346
Name: NAME_CONTRACT_TYPE, dtype: int64
```

```
# Since the XNA values are miniscule we can merge it with the mode of the column
pre_data.NAME_CONTRACT_TYPE.replace(["XNA"],pre_data.NAME_CONTRACT_TYPE.mode(),inplace=True)
```

```
pre_data.NAME_CONTRACT_TYPE.unique() # XNA replaced
```

```
array(['Consumer loans', 'Cash loans', 'Revolving loans'], dtype=object)
```

The columns NAME_CASH_LOAN_PURPOSE and CODE_REJECT_REASON has XNA and XAP as major values. We are keeping it as it isThe columns NAME_CASH_LOAN_PURPOSE and CODE_REJECT_REASON has XNA and XAP as major values. We are keeping it as it is

```
pre_data.NAME_CASH_LOAN_PURPOSE.value_counts(normalize=True)*100
```

```
XAP                                 55.242083
XNA                                 40.588691
Repairs                              1.422872
Other                                0.934491
Urgent needs                         0.503648
Buying a used car                    0.172912
Building a house or an annex         0.161237
Everyday expenses                    0.144652
Medicine                             0.130163
Payments on other loans              0.115614
Education                            0.094180
Journey                              0.074182
Purchase of electronic equipment     0.063525
Buying a new car                     0.060591
Wedding / gift / holiday             0.057597
Buying a home                        0.051790
Car repairs                          0.047718
Furniture                            0.044845
Buying a holiday home / land         0.031912
Business development                 0.025506
Gasification / water supply          0.017962
Buying a garage                      0.008143
Hobby                                0.003293
Money for a third person             0.001497
```

```
Refusal to name the goal            0.000898
Name: NAME_CASH_LOAN_PURPOSE, dtype: float64
```

```
pre_data.CODE_REJECT_REASON.value_counts(normalize=True)*100
```

```
XAP       81.013152
HC        10.491530
LIMIT      3.333705
SCO        2.243245
CLIENT     1.582791
SCOFR      0.767027
XNA        0.313972
VERIF      0.211650
SYSTEM     0.042929
Name: CODE_REJECT_REASON, dtype: float64
```

```
pre_data.WEEKDAY_APPR_PROCESS_START.unique() # Seems fine, no Null values
```

```
array(['SATURDAY', 'THURSDAY', 'TUESDAY', 'MONDAY', 'FRIDAY', 'SUNDAY',
       'WEDNESDAY'], dtype=object)
```

```
# Converting the negative values to absolute
pre_data.DAYS_DECISION = abs(pre_data.DAYS_DECISION)
```

```
pre_data.SELLERPLACE_AREA = abs(pre_data.SELLERPLACE_AREA )
```

```
pre_data.head(3) # Checking whether the above codes have run properly
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	WEEKDAY_APPR_PROCESS_START	HOUR_APPR_PROCES
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	17145.0	SATURDAY	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	607500.0	THURSDAY	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	112500.0	TUESDAY	

Merging both the data csv files for final analysis

```
# Merging the two csv files
Merged_Data = pd.merge(app_data,pre_data,on="SK_ID_CURR")
```

```
Merged_Data.head()
```

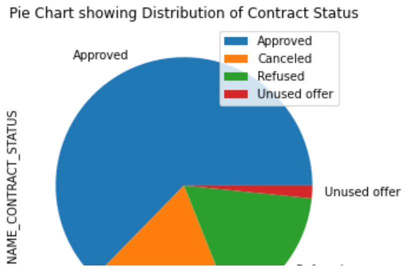
	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5
2	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5
3	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5
4	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0

```
# Creating two separate variables for Defaulters and Customers with no payment issue
MergeData_Defaulters = Merged_Data[Merged_Data.TARGET==1]
MergeData_Noissue = Merged_Data[Merged_Data.TARGET==0]
```

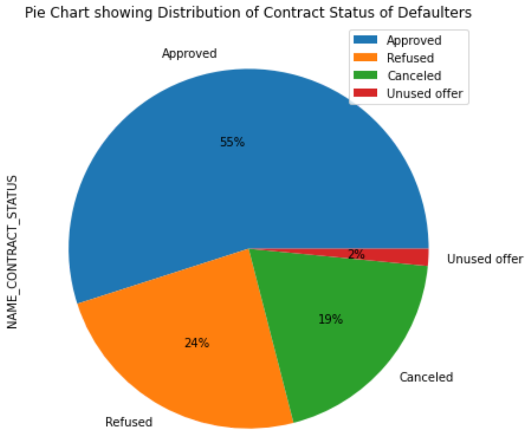
```
# Checking for the Contract Status of the merged data set
Merged_Data.NAME_CONTRACT_STATUS.value_counts(normalize=True)*100
```

```
Approved        62.679378
Canceled        18.351900
Refused         17.357984
Unused offer     1.610737
Name: NAME_CONTRACT_STATUS, dtype: float64
```

```
# Checking the Contract Status of the Merged Data in a piechart
plt.figure(figsize=(15,5))
Merged_Data.NAME_CONTRACT_STATUS.value_counts(normalize=True).plot.pie()
plt.legend()
plt.title("Pie Chart showing Distribution of Contract Status")
plt.show()
```

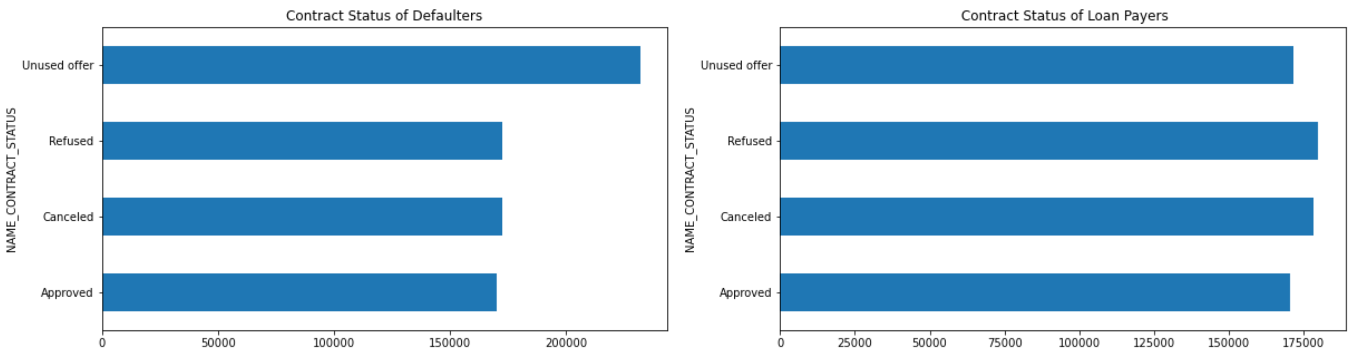



```
# Checking the Contract Status of the Merged Data set of the Defaulters vs Loan Payers
plt.figure(figsize=(15,10))
plt.subplot(1,2,1)
MergeData_Defaulters.NAME_CONTRACT_STATUS.value_counts(normalize=True).plot.pie(autopct='%.f%%')
plt.legend()
plt.title("Pie Chart showing Distribution of Contract Status of Defaulters")

plt.show()
```



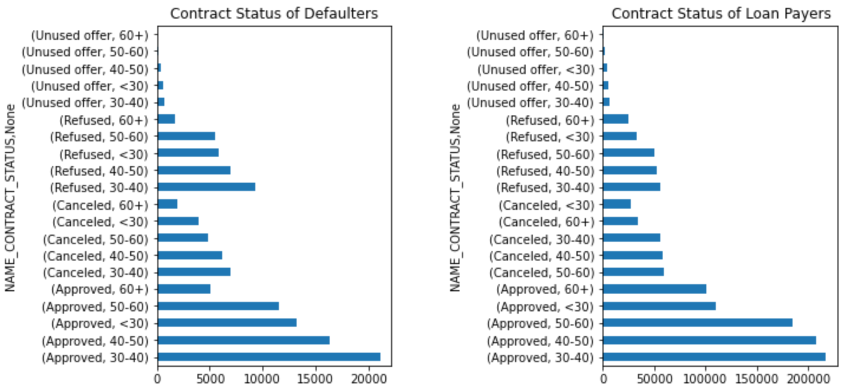
It can be seen that around 24% defaulters were refused the loan in the previous application which were given the loan. This should have been considered while giving the loan

```
# Checking Contract Status with respect to Income of the Customers
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
MergeData_Defaulters.groupby("NAME_CONTRACT_STATUS")["AMT_INCOME_TOTAL"].mean().plot.barh()
plt.title("Contract Status of Defaulters")

plt.subplot(1,2,2)
MergeData_Noissue.groupby("NAME_CONTRACT_STATUS")["AMT_INCOME_TOTAL"].mean().plot.barh()
plt.title(" Contract Status of Loan Payers")

plt.show()
```

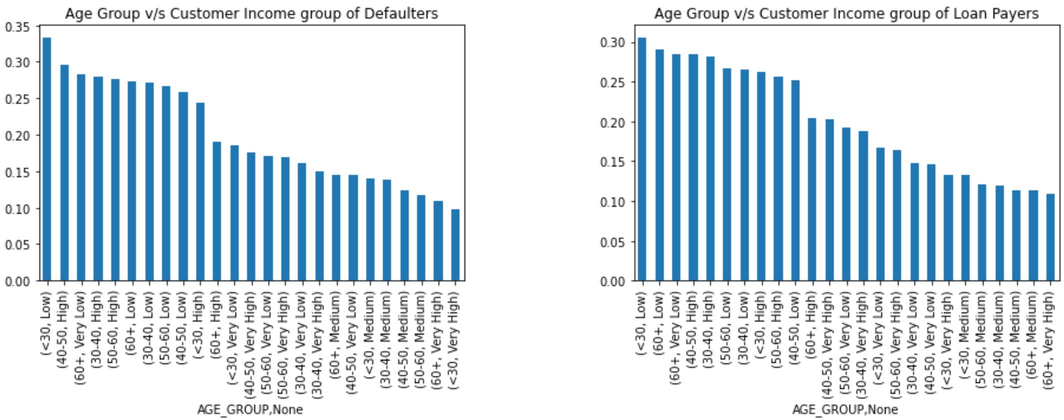



```
# Checking Contract Status with respect to Age of the Customers
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
MergeData_Defaulters.groupby("NAME_CONTRACT_STATUS")["AGE_GROUP"].value_counts().plot.barh()
plt.title("Contract Status of Defaulters")

plt.subplot(1,2,2)
MergeData_Noissue.groupby("NAME_CONTRACT_STATUS")["AGE_GROUP"].value_counts().plot.barh()
plt.title(" Contract Status of Loan Payers")

# Adjusting the spacing within the subplots
plt.subplots_adjust(left=0.1,
```

```
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.9,
                    hspace=0.4)

plt.show()
```

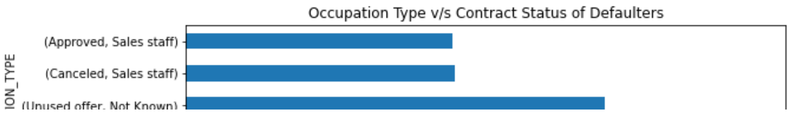



```
# Checking Age Group vs Cust Income of the Customers
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
MergeData_Defaulters.groupby("AGE_GROUP")["CUST_INCOME"].value_counts(normalize=True).sort_values(ascending=False).plot.bar()
plt.title("Age Group v/s Customer Income group of Defaulters")


plt.subplot(1,2,2)
MergeData_Noissue.groupby("AGE_GROUP")["CUST_INCOME"].value_counts(normalize=True).sort_values(ascending=False).plot.bar()
plt.title("Age Group v/s Customer Income group of Loan Payers")

# Adjusting the spacing within the subplots
plt.subplots_adjust(left=0.1,
                    bottom=0.3,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.6)

plt.show()
```

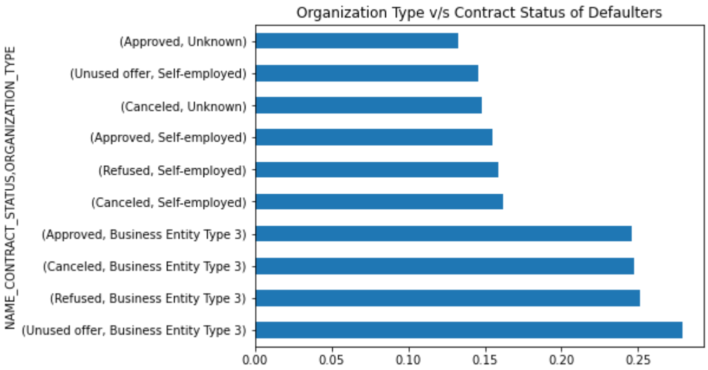


It can be seen that majority of the defaulters are from low income

```
# Defaulters with respect to OCCUPATION_TYPE and NAME_CONTRACT_STATUS

plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
MergeData_Defaulters.groupby("NAME_CONTRACT_STATUS")["OCCUPATION_TYPE"].value_counts(normalize=True).sort_values(ascending=False).head(10).plot.barh()
plt.title("Occupation Type v/s Contract Status of Defaulters")

plt.show()
```



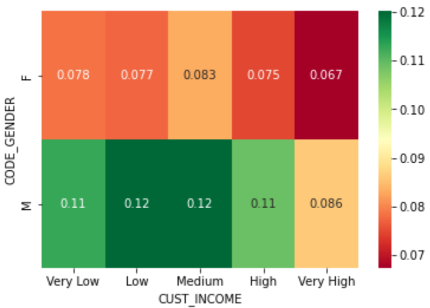

```
# Defaulters ORGANIATION V/s NAME_CONTRACT_STATUS
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
MergeData_Defaulters.groupby("NAME_CONTRACT_STATUS")["ORGANIZATION_TYPE"].value_counts(normalize=True).sort_values(ascending=False).head(10).plot.barh()
plt.title("Organization Type v/s Contract Status of Defaulters")

plt.show()
```



- Most of the defaulters are the ones with Business Entity Type - 3 and Self-Employed.

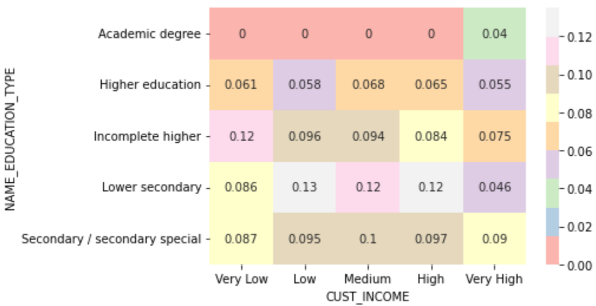
Multivariate Analysis Correlations

```
# Gender vs Customer Income vs Defaulters
Gender_Income = pd.pivot_table(data=Merged_Data, index='CODE_GENDER', columns='CUST_INCOME', values='TARGET')


sns.heatmap(Gender_Income, annot=True, cmap="RdYlGn")
plt.show()
```



- We can see that there is strong correlation between Females and Medium Income group, Males and Very High Income group among Defaulters

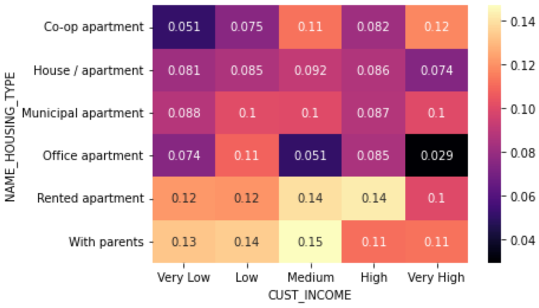
```
# Education vs Customer Income vs Defaulters
Edu_Income = pd.pivot_table(data=Merged_Data, index='NAME_EDUCATION_TYPE', columns='CUST_INCOME', values='TARGET')


sns.heatmap(Edu_Income, annot=True, cmap="Pastel1")
plt.show()
```



- Lower Education levels and Lower Salary levels show strong corelation while defaulting loans

```
House_Income = pd.pivot_table(data=Merged_Data, index='NAME_HOUSING_TYPE', columns='CUST_INCOME', values='TARGET')
```

```
sns.heatmap(House_Income, annot=True, cmap="magma")
plt.show()
```



We can observe that there is strong correlation with all income levels and customers Owning House have defaulted while customers staying with parents or on rent have very weak correlation with defaults

Observations

Following observations can be made from datasets provided and analysed

Majorly customers defaulted the cash loans.

Females were mostly the defaulters but also they were major customers who paid the Loans on time The Secondary level education customers were the defaulters Working class customers were major defaulters but also major customers paying the loan As the no. of family members and children increased customers were more inclined to pay loans on time Low Salary was the main cause for the defaults People who were self employed or who worked for Business Entity Type 3 were major defaulters People who worked as Laborers were most defaulters