Problem Statement

In this case study, we focus to perform EDA using visualizations and statistics summaries on loan application data. We need to find out what factors affect an applicant to be a defaulter or non-defaulter.

Approach

For EDA we will try to follow the bolow steps: 1) Import Modules

2) Read the dataset

3) Data Cleaning

```
On previous_application dataframe

    a) Missing value handling
        i) identifying missing data
        ii) Dropping missing data
    b) Outlier Analysis
On application_data dataframe

    a) Missing value handling
        i) identifying missing data
        ii) Dropping missing data
c) Data Imbalance
```

4) Analysis of different variables in segments :-

```
a) Segment 1

b) Segment 2

c) Segment 3

d) Segment 4

e) Segment 5
```

5) Correlation

```
a) Top 5 correlation for defaulters

b) Top 5 correlations for non-defaulters
```

6) Summary

1) Import Modules

Let's import libraries for EDA

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Import our dataset as datframes using pandas library

2) Read Datasets

```
# reading application_data.csv
application_df = pd.read_csv('/content/sample_data/project 6 file 3.csv')
# reading previous_application.csv
prev_ap_df = pd.read_csv('/content/previous_application (2).csv')
```

```
print(application_df.shape)
print(prev_ap_df.shape)
```

```
(49999, 37)
(49999, 37)
```

Lets check all features and it's datatypes

```
application_df.info()
```

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

```
prev_ap_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Data columns (total 37 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   SK_ID_PREV                   49999 non-null  int64
 1   SK_ID_CURR                   49999 non-null  int64
 2   NAME_CONTRACT_TYPE           49999 non-null  object
 3   AMT_ANNUITY                  39407 non-null  float64
```

```
 4   AMT_APPLICATION              49999 non-null  float64
 5   AMT_CREDIT                   49999 non-null  float64
 6   AMT_DOWN_PAYMENT             24801 non-null  float64
 7   AMT_GOODS_PRICE              39255 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   49999 non-null  object
 9   HOUR_APPR_PROCESS_START      49999 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  49999 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       49999 non-null  int64
 12  RATE_DOWN_PAYMENT            24801 non-null  float64
 13  RATE_INTEREST_PRIMARY        165 non-null    float64
 14  RATE_INTEREST_PRIVILEGED     165 non-null    float64
 15  NAME_CASH_LOAN_PURPOSE       49999 non-null  object
 16  NAME_CONTRACT_STATUS         49999 non-null  object
 17  DAYS_DECISION                49999 non-null  int64
 18  NAME_PAYMENT_TYPE            49999 non-null  object
 19  CODE_REJECT_REASON           49999 non-null  object
 20  NAME_TYPE_SUITE              25756 non-null  object
 21  NAME_CLIENT_TYPE             49999 non-null  object
 22  NAME_GOODS_CATEGORY          49999 non-null  object
 23  NAME_PORTFOLIO               49999 non-null  object
 24  NAME_PRODUCT_TYPE            49999 non-null  object
 25  CHANNEL_TYPE                 49999 non-null  object
 26  SELLERPLACE_AREA             49999 non-null  int64
 27  NAME_SELLER_INDUSTRY         49999 non-null  object
 28  CNT_PAYMENT                  39407 non-null  float64
 29  NAME_YIELD_GROUP             49999 non-null  object
 30  PRODUCT_COMBINATION          49991 non-null  object
 31  DAYS_FIRST_DRAWING           30839 non-null  float64
 32  DAYS_FIRST_DUE               30839 non-null  float64
 33  DAYS_LAST_DUE_1ST_VERSION    30839 non-null  float64
 34  DAYS_LAST_DUE                30839 non-null  float64
 35  DAYS_TERMINATION             30839 non-null  float64
 36  NFLAG_INSURED_ON_APPROVAL    30839 non-null  float64
dtypes: float64(15), int64(6), object(16)
memory usage: 14.1+ MB
```

Insights

1. prev_ap_df contains 37 features and 1670214 rows.

   Out of which 15 are float64, 6 are int64 and 16 are object datatype.

2. application_df contains 121 features, 1 target variable, and 307511 rows.

   Out of which 65 are float64, 41 are integer, 16 are object datatype

```
# We will try to store lists of columns that are common in both dataframes and additional in any of them
additional_feat = []
common_feat = []

for col in application_df.columns:
    if col in prev_ap_df.columns:
        common_feat.append(col)
    else:
        additional_feat.append(col)

print(len(additional_feat))
print(len(common_feat))
print(common_feat)
```

```
0
37
['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY', 'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE'
```

Insights

1. There are 8 columns that are present in both dataframes
2. SK_ID_CURR is the unique identifyer in both dataframe that we will use to merge both dataframes.

3) Data Cleaning

Firstly I will try to perform data cleaning on the 'prev_ap_df'

a) Missing value handling

i) Identifying Missing Data

```
# define a function that has one arguement as dataframe
def missingdata_percentage(df):
    # create an empty dataframe with two columns
    missing = pd.DataFrame(columns=['category','percentage'])
    # iterate through all the columns of dataframe
    for col in df.columns:
        # a conditional statement that only passes those columns that have at least 1 missing value
        if df[col].isna().values.any():
            # calulate the percentage of null values in that particular column
            percentage = 100*df[col].isna().sum()/df.shape[0]
            # append that column into to empty dataframe we created earlier
            missing = missing.append({'category' : col, 'percentage' : percentage}, ignore_index=True)
    # return the dataframe we created
    return missing


def missingdata_percentage(df):
    missing_percent = (df.isnull().sum() / len(df)) * 100
    missing_df = pd.DataFrame({
        'Column': missing_percent.index,
        'Missing Percentage': missing_percent.values
    })
    return missing_df[missing_df['Missing Percentage'] > 0].sort_values(by='Missing Percentage', ascending=False)


missingdata_prev = missingdata_percentage(prev_ap_df)


missingdata_prev.sort_values('Missing Percentage', ascending=False)
```

	Column	Missing Percentage
13	RATE_INTEREST_PRIMARY	99.669993
14	RATE_INTEREST_PRIVILEGED	99.669993
6	AMT_DOWN_PAYMENT	50.397008
12	RATE_DOWN_PAYMENT	50.397008
20	NAME_TYPE_SUITE	48.486970
31	DAYS_FIRST_DRAWING	38.320766
32	DAYS_FIRST_DUE	38.320766
35	DAYS_TERMINATION	38.320766
36	NFLAG_INSURED_ON_APPROVAL	38.320766
34	DAYS_LAST_DUE	38.320766
33	DAYS_LAST_DUE_1ST_VERSION	38.320766
7	AMT_GOODS_PRICE	21.488430
3	AMT_ANNUITY	21.184424
28	CNT_PAYMENT	21.184424
30	PRODUCT_COMBINATION	0.016000

Insights

There are 16 features in prev_app_df that have missing values.

- Permanently dropping the features (RATE_INTEREST_PRIMARY and RATE_INTEREST_PRIVILEGED) as 99% data is missing.
- Dropping rows containing missing values for the features(AMT_CREDIT and PRODUCT_COMBINATION) for very low % of missing data.
- Dropping entries would not cause impact the analysis as percentage of missing value is very low (~2%).

ii) Dropping missing data

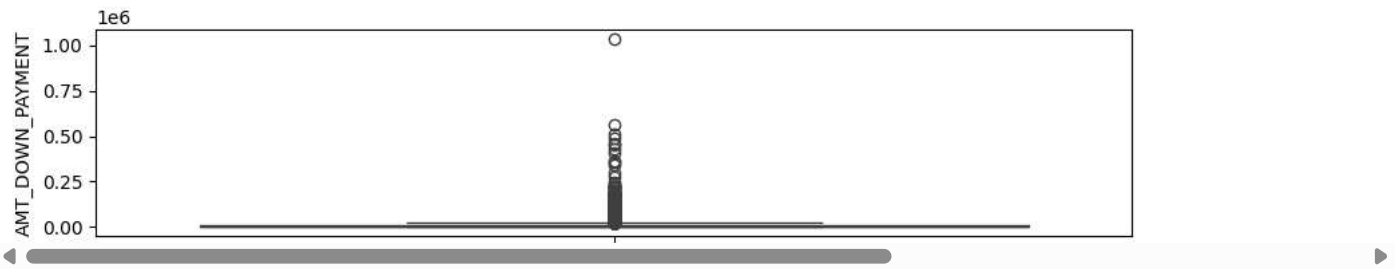
```
prev_ap_df.drop(['RATE_INTEREST_PRIMARY','RATE_INTEREST_PRIVILEGED'], axis=1, inplace=True)
prev_ap_df.dropna(subset=['AMT_CREDIT','PRODUCT_COMBINATION'], inplace=True)

#Checking the remaining columns
prev_ap_df
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WEEKDAY_
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	17145.0	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	607500.0	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	112500.0	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	450000.0	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	337500.0	
...	...	...	...	...	...	...	...	...	
49994	1171956	339569	Cash loans	NaN	0.0	0.0	NaN	NaN	
49995	1904808	363980	Cash loans	NaN	0.0	0.0	NaN	NaN	
49996	2331005	231295	Cash loans	22176.405	180000.0	216418.5	NaN	180000.0	
49997	1960897	346691	Cash loans	NaN	0.0	0.0	NaN	NaN	
49998	1979352	363244	Cash loans	24909.390	360000.0	409896.0	NaN	360000.0	

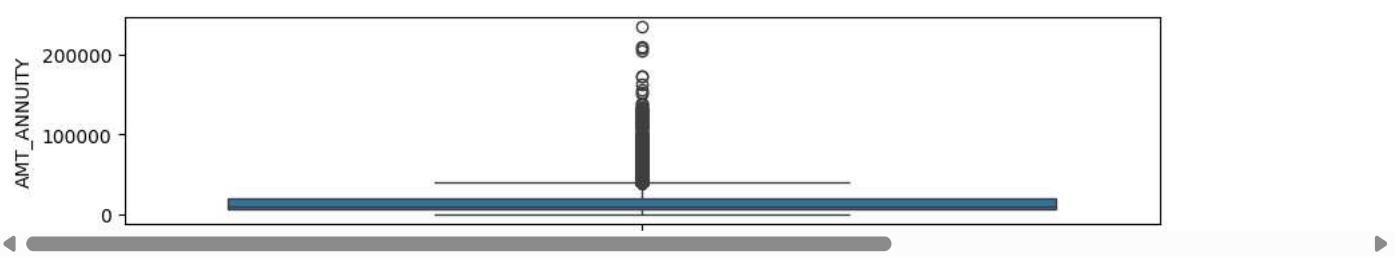
49991 rows × 35 columns

B) Outliers Analysis

```
plt.figure(figsize=(10,2))
sns.boxplot(prev_ap_df['AMT_DOWN_PAYMENT'])
plt.show()
```

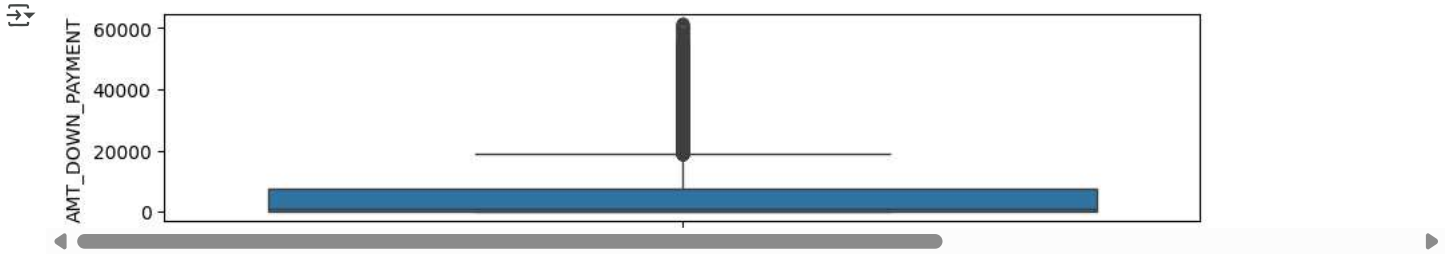



```
plt.figure(figsize=(10,2))
sns.boxplot(prev_ap_df['AMT_ANNUITY'])
plt.show()
```

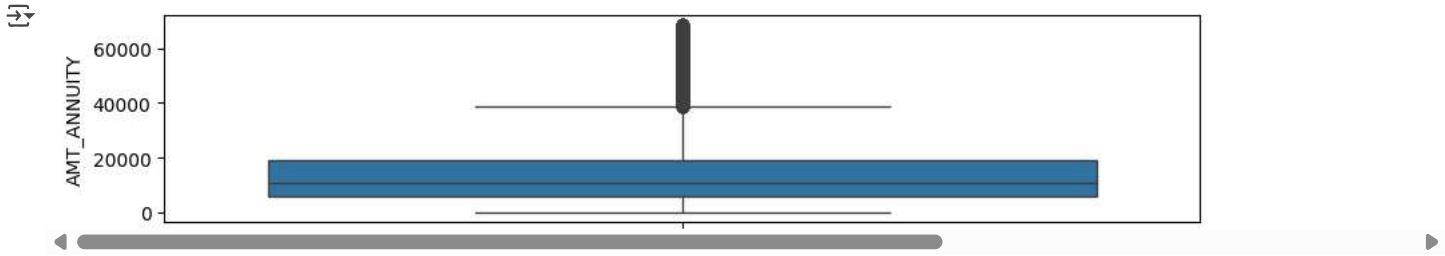



```
import pandas as pd

def remove_outliers(df, column):
    """
    Removes outliers outside of the 99th percentile in a Pandas DataFrame for a given column.

    Parameters:
        df (Pandas DataFrame): The DataFrame containing the data.
```

```
        column (str): The name of the column to remove outliers for.

    Returns:
        The DataFrame with outliers removed.
    """

    # calculate the 99th percentile
    perc_99 = df[column].quantile(0.99)


    # remove outliers outside the IQR
    df = df[~(df[column] > perc_99)]

    return df


prev_ap_df = remove_outliers(prev_ap_df, 'AMT_DOWN_PAYMENT')


plt.figure(figsize=(10,2))
sns.boxplot(prev_ap_df['AMT_DOWN_PAYMENT'])
plt.show()
```




```
prev_ap_df = remove_outliers(prev_ap_df, 'AMT_ANNUITY')


plt.figure(figsize=(10,2))
sns.boxplot(prev_ap_df['AMT_ANNUITY'])
plt.show()
```



Now, I will try to perform data cleaning on the 'application_df'

a) Missing value handling

i) Identifying Missing Data

```
# We will use missingdata_percentage() function that we created earlier
missingdata_application_df = missingdata_percentage(application_df)


def missingdata_percentage(df):
    missing_percent = (df.isnull().sum() / len(df)) * 100
    missing_df = pd.DataFrame({
        'Column': missing_percent.index,
        'Missing Percentage': missing_percent.values
    })
    return missing_df[missing_df['Missing Percentage'] > 0]


missingdata_application_df = missingdata_percentage(application_df)


missingdata_application_df = missingdata_application_df.sort_values('Missing Percentage', ascending=False).reset_index(drop=True)
```

```
missingdata_application_df[missingdata_application_df['percentage']<1]
```

	category	percentage
0	AMT_ANNUITY	0.003902
1	AMT_GOODS_PRICE	0.090403
2	NAME_TYPE_SUITE	0.420148
5	CNT_FAM_MEMBERS	0.000650
7	EXT_SOURCE_2	0.214626
56	OBS_30_CNT_SOCIAL_CIRCLE	0.332021
57	DEF_30_CNT_SOCIAL_CIRCLE	0.332021
58	OBS_60_CNT_SOCIAL_CIRCLE	0.332021
59	DEF_60_CNT_SOCIAL_CIRCLE	0.332021
60	DAYS_LAST_PHONE_CHANGE	0.000325

Insights

There are 16 features in prev_app_df that have missing values.

- Dropping rows containing missing values for the features for <1% of missing data.
- Dropping entries would not cause impact the analysis as percentage of missing value is very low (~2%).

ii) Dropping missing data

```
application_df.dropna(subset=['AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'CNT_FAM_MEMBERS', 'EXT_SOURCE_2', 'OBS_30_CNT_SOCIAL_CIR

#Checking the remaining columns
application_df
```

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

304531 rows × 122 columns

c) Checking Data Imbalance

for checking imbalance in data we need to merge both dataframes and clean it jut as we have cleaned both the datasets separately

```
# Merging only required columns of application_data with previous_application_data

prev_ap_merged = pd.merge(application_df[['SK_ID_CURR', 'TARGET']], prev_ap_df, how='left', on=['SK_ID_CURR'])
print(prev_ap_merged.shape)
```

```
(1404179, 36)
```

```
prev_ap_merged = remove_outliers(prev_ap_merged, 'AMT_DOWN_PAYMENT')
prev_ap_merged = remove_outliers(prev_ap_merged, 'AMT_ANNUITY')
print(prev_ap_merged.shape)
```

```
(1388267, 36)
```

```
prev_ap_merged.describe()
```

	SK_ID_CURR	TARGET	SK_ID_PREV	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	HOUR_APPR
count	1.388267e+06	1.388267e+06	1.371700e+06	1.066537e+06	1.371700e+06	1.371700e+06	646928.000000	1.054234e+06	
mean	2.784320e+05	8.702505e-02	1.922904e+06	1.452031e+04	1.564674e+05	1.775130e+05	5071.408513	2.036190e+05	
std	1.027883e+05	2.818719e-01	5.326533e+05	1.156755e+04	2.481789e+05	2.741488e+05	7987.331198	2.656373e+05	
min	1.000020e+05	0.000000e+00	1.000001e+06	0.000000e+00	0.000000e+00	0.000000e+00	-0.900000	0.000000e+00	
25%	1.893400e+05	0.000000e+00	1.461701e+06	6.126255e+03	1.818000e+04	2.384100e+04	0.000000	4.855590e+04	
50%	2.789110e+05	0.000000e+00	1.922842e+06	1.087384e+04	6.750000e+04	7.695000e+04	1539.000000	1.043100e+05	
75%	3.674500e+05	0.000000e+00	2.384084e+06	1.939279e+04	1.777500e+05	1.978200e+05	7164.000000	2.250000e+05	
max	4.562550e+05	1.000000e+00	2.845381e+06	5.805932e+04	3.511305e+06	3.511305e+06	45000.000000	3.511305e+06	

```
plt.style.use("ggplot")
plt.title("Status of Previous Loan Application and Payment")
sns.countplot(prev_ap_merged['NAME_CONTRACT_STATUS'], hue=prev_ap_merged['TARGET'])
plt.show()
```


```
# Percentage of previously approved loan applicants that defaulted in current loan

total_approved = prev_ap_merged[prev_ap_merged['NAME_CONTRACT_STATUS'] == "Approved"].shape[0]
default_approved = prev_ap_merged[(prev_ap_merged['TARGET'] == 1) & (prev_ap_merged['NAME_CONTRACT_STATUS'] == "Approved")].shape[0]

print("Percentage of previously approved loan applicants that defaulted in current loan : {}".format(default_approved/total_approved*100))
```

```
Percentage of previously approved loan applicants that defaulted in current loan : 7.678348657899435
```

```
total_refused = prev_ap_merged[prev_ap_merged['NAME_CONTRACT_STATUS'] == "Refused"].shape[0]
nondefault_refused = prev_ap_merged[(prev_ap_merged['TARGET'] == 0) & (prev_ap_merged['NAME_CONTRACT_STATUS'] == "Refused")].shape[0]

print("Percentage of previously refused loan applicants that were able to pay current loan : {}".format(nondefault_refused/total_refused*100
```

```
Percentage of previously refused loan applicants that were able to pay current loan : 87.86236584753729
```

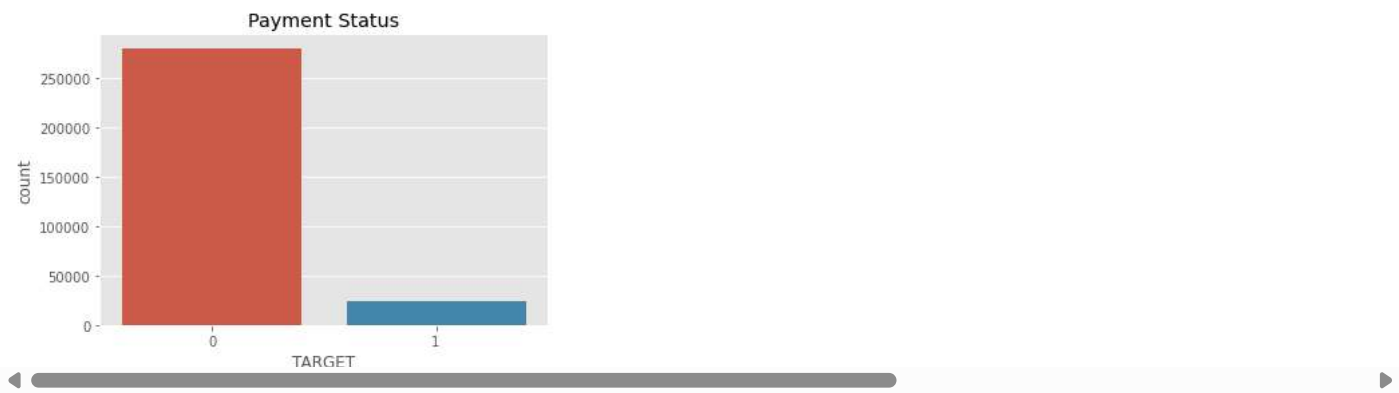
Insights :

The applicants whose previous loans were approved are more likely to pay current loan in time, than the applicants whose previous loans were rejected.

7.6% of the previously approved loan applicants that defaulted in current loan

87.8 % of the previously refused loan applicants that were able to pay current loan

```
plt.title("Payment Status")
sns.countplot(application_df['TARGET'])
plt.show()
```



This data is highly imbalanced as number of defaulter is very less in total population.

```
non_default  = application_df[application_df["TARGET"] == 0]
default  = application_df[application_df["TARGET"] == 1]

print("No. of defaulters: ", default.shape[0])
print("No. of non-defaulters: ", non_default.shape[0])
```

```
No. of defaulters:  24667
No. of non-defaulters:  279864
```

```
print("Percentage of defaulters: ", default.shape[0]*100/(default.shape[0]+non_default.shape[0]))
```

```
Percentage of defaulters:  8.099996387888261
```

Insights :

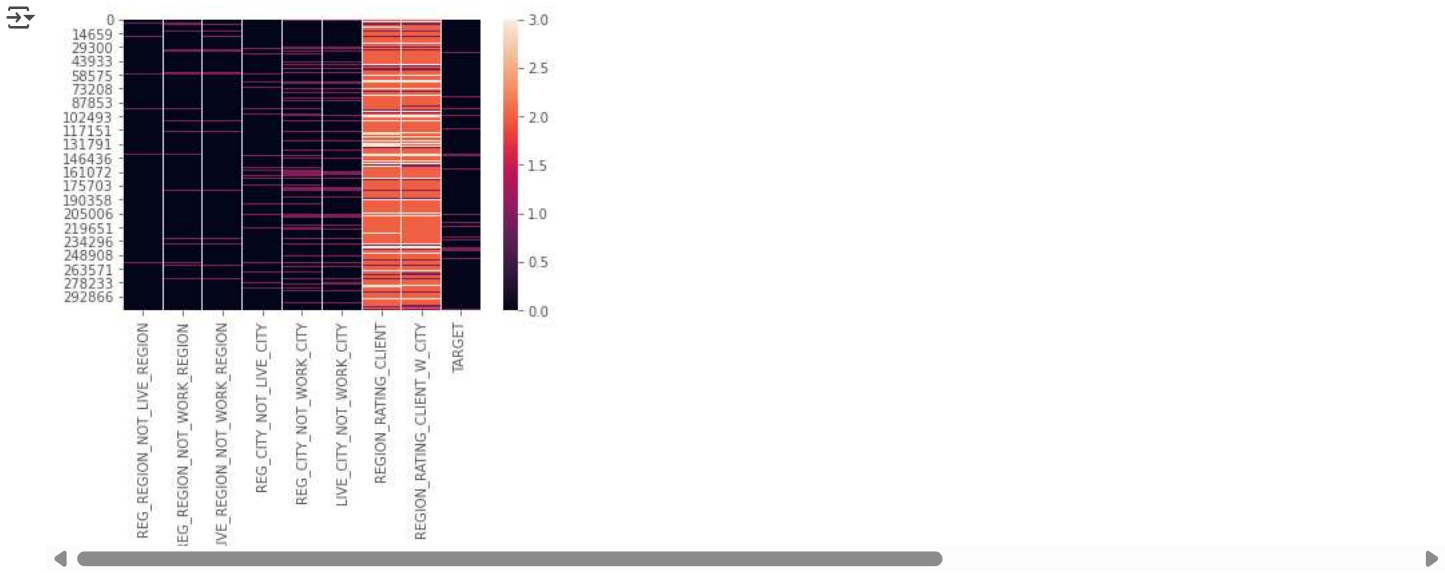
This data is highly imbalanced as number of defaulter is very less in total population. Data Imbalance Ratio

Defaulter : Non-Defaulter = 8 : 92 = 2 : 23

4) Analysis of diffrent variables in segments

Segment 1 :- Region based analysis

```
start_idx = application_df.columns.get_loc('REG_REGION_NOT_LIVE_REGION')
end_idx = application_df.columns.get_loc('LIVE_CITY_NOT_WORK_CITY')

region_df = application_df.iloc[:, start_idx:end_idx+1]


region_df['REGION_RATING_CLIENT'] = application_df['REGION_RATING_CLIENT']
region_df['REGION_RATING_CLIENT_W_CITY'] = application_df['REGION_RATING_CLIENT_W_CITY']
region_df["TARGET"] = application_df["TARGET"]


sns.heatmap(region_df)
plt.show()
```



Insights:

- All the features are labeled as 0 and 1.
- REG_REGION_NOT_LIVE_REGION mostly contains 0, hence it can be removed
- REG_REGION_NOT_WORK_REGION, LIVE_REGION_NOT_WORK_REGION columns are identical, hence one of them can be removed
- REG_CITY_NOT_WORK_CITY, LIVE_CITY_NOT_WORK_CITY columns are identical, hence one of them can be removed.

```
fig=plt.subplots(figsize=(10, 10))

for i, j in enumerate(['REG_REGION_NOT_WORK_REGION','REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY','REGION_RATING_CLIENT', 'REGION_RATIN
    plt.subplot(5, 3, i+1, ylim=(0, 250000))
    plt.subplots_adjust(hspace = 1.0)
    sns.countplot(application_df[j], hue=application_df["TARGET"])
    plt.xticks(rotation=90)
    plt.tight_layout()
```

Insights:

- Defaulter rate is highest when REG_REGION_NOT_WORK_REGION=0 i.e. permanent address and working address is same

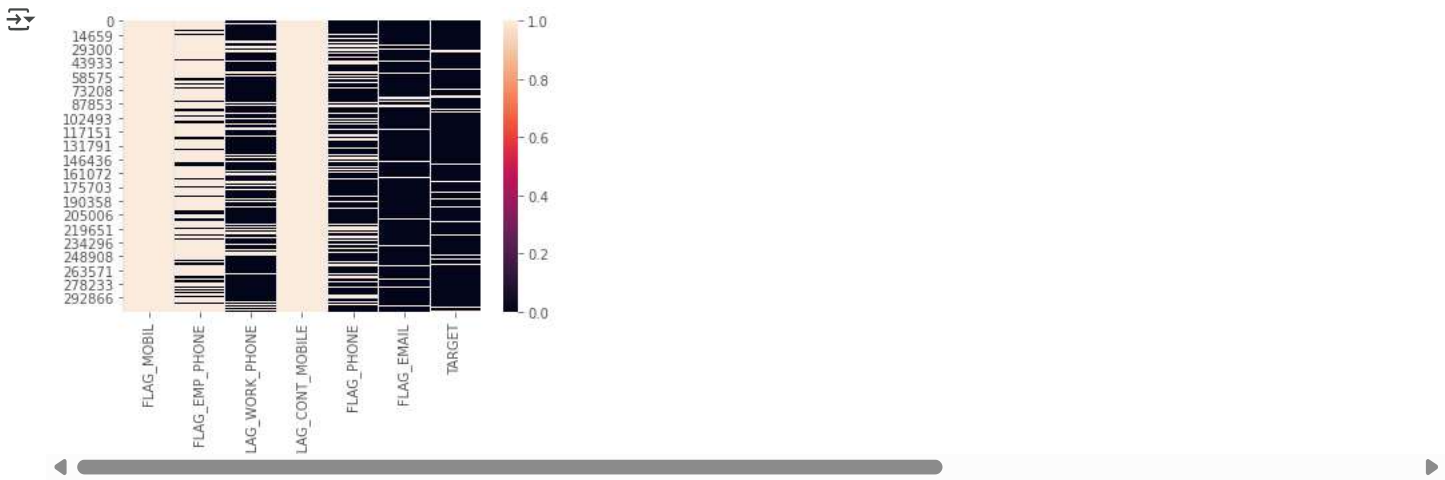
Segment 2 :- Based on contact

```
contact_df = application_df[['FLAG_MOBIL','FLAG_EMP_PHONE','FLAG_WORK_PHONE','FLAG_CONT_MOBILE','FLAG_PHONE','FLAG_EMAIL','DAYS_LAST_PHONE_C
contact_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 304531 entries, 0 to 307510
Data columns (total 8 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   FLAG_MOBIL            304531 non-null  int64
```

```
 1   FLAG_EMP_PHONE          304531 non-null  int64
 2   FLAG_WORK_PHONE         304531 non-null  int64
 3   FLAG_CONT_MOBILE        304531 non-null  int64
 4   FLAG_PHONE              304531 non-null  int64
 5   FLAG_EMAIL              304531 non-null  int64
 6   DAYS_LAST_PHONE_CHANGE  304531 non-null  float64
 7   TARGET                  304531 non-null  int64
dtypes: float64(1), int64(7)
memory usage: 29.0 MB
```

```
plt.figure()
sns.heatmap(contact_df.drop('DAYS_LAST_PHONE_CHANGE', axis=1))
plt.show()
```



Insights:

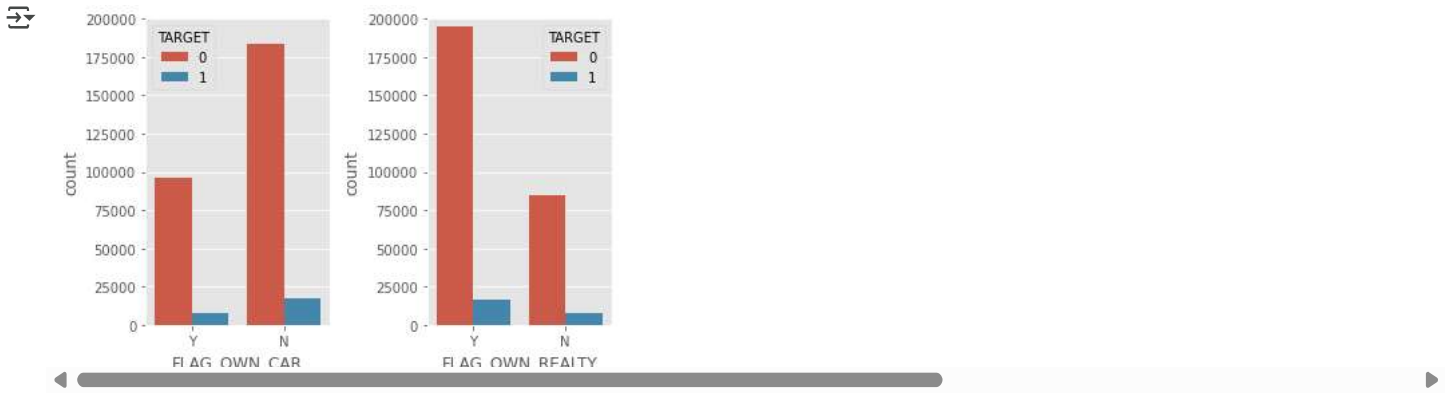
- All the features in contact_df are categorical (0 and 1)
- As there is no similarity of patterns of TARGET value with the features,
- we are assuming the feature are not useful for analysis.
- Hence all of these features can be removed.

Segment 3 :- Based on assets owned

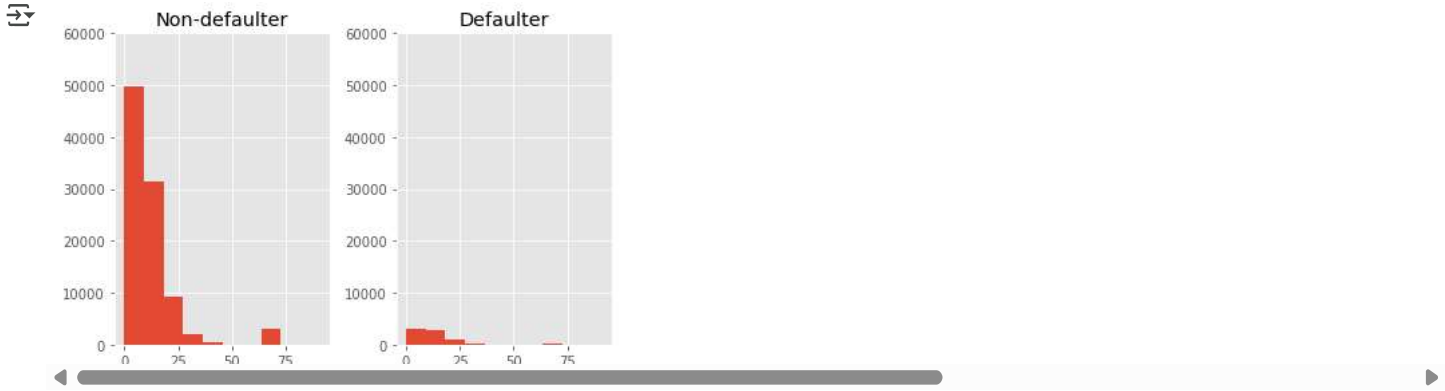
```
application_df[['FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'OWN_CAR_AGE', 'TARGET']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 304531 entries, 0 to 307510
Data columns (total 4 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   FLAG_OWN_CAR    304531 non-null  object
 1   FLAG_OWN_REALTY  304531 non-null  object
 2   OWN_CAR_AGE     103619 non-null  float64
 3   TARGET          304531 non-null  int64
dtypes: float64(1), int64(1), object(2)
memory usage: 19.7+ MB
```

```
fig = plt.figure()

ax1 = fig.add_subplot(1, 2, 1, ylim=(0,200000))
ax2 = fig.add_subplot(1, 2, 2, ylim=(0,200000))

sns.countplot(application_df['FLAG_OWN_CAR'], hue=application_df['TARGET'], order=['Y','N'], ax=ax1)
sns.countplot(application_df['FLAG_OWN_REALTY'], hue=application_df['TARGET'], order=['Y','N'], ax=ax2)

plt.tight_layout()

plt.show()
```




```
fig = plt.figure()

ax1 = fig.add_subplot(1, 2, 1, ylim=(0,60000), title="Non-defaulter")
ax2 = fig.add_subplot(1, 2, 2, ylim=(0,60000), title="Defaulter")

non_default['OWN_CAR_AGE'].hist(bins=10, ax=ax1)
default['OWN_CAR_AGE'].hist(bins=10, ax=ax2)

plt.tight_layout()

plt.show()
```

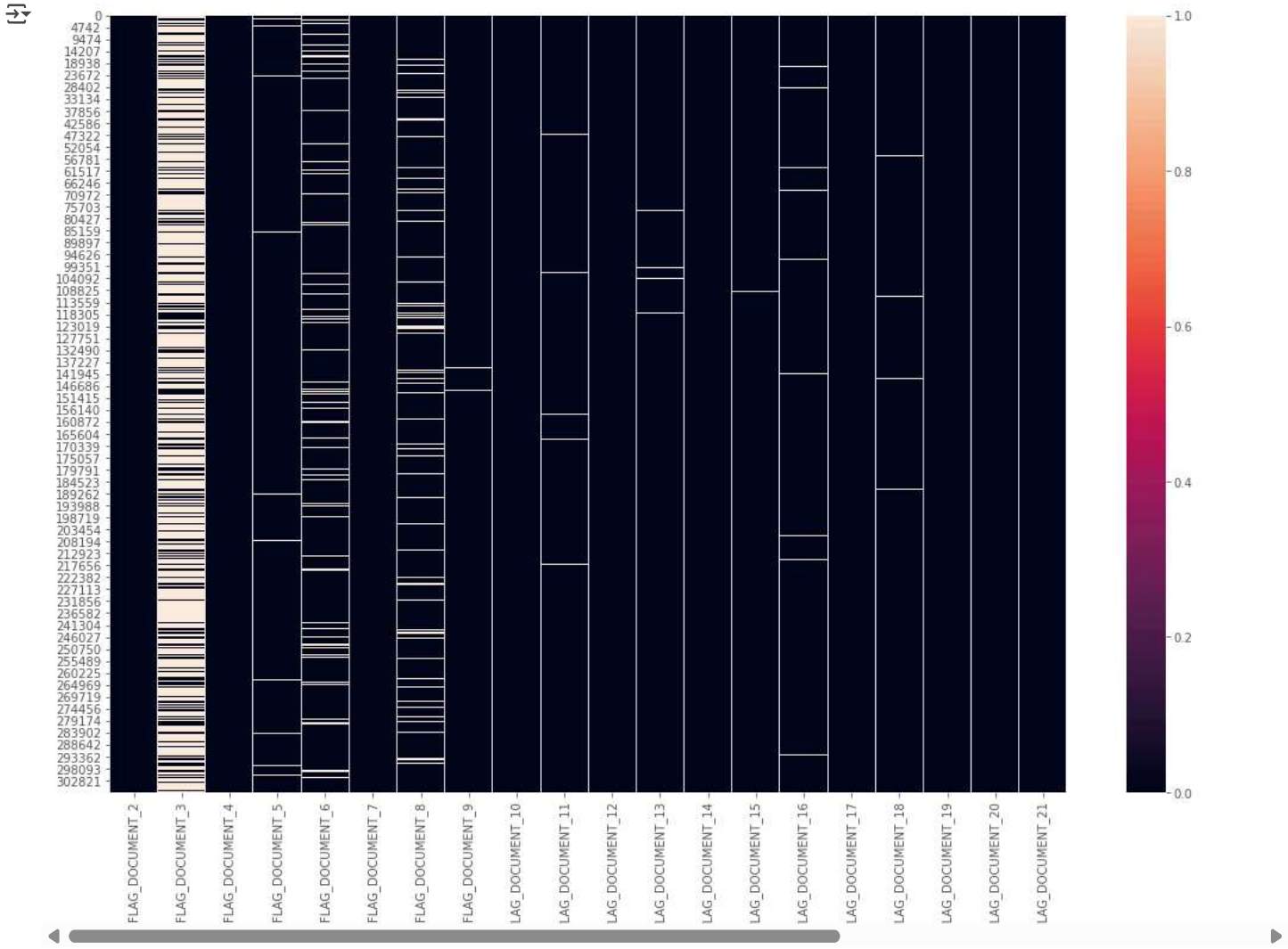


Insights :-

- Most people between the age of 0 to 25 have cars irrespective of weather they are defaulter or not.
- both target values are similar so this feature can also be dropped.
- People not owning reality and car have a slightly higher default rate than the people who own reality and car.

Segement 4 :- Based on documents submitted

```
starting_idx = application_df.columns.get_loc("FLAG_DOCUMENT_2")
ending_idx = application_df.columns.get_loc("FLAG_DOCUMENT_21")+1

plt.figure(figsize=(18,12))
sns.heatmap(application_df.iloc[:,starting_idx:ending_idx])
plt.show()
```

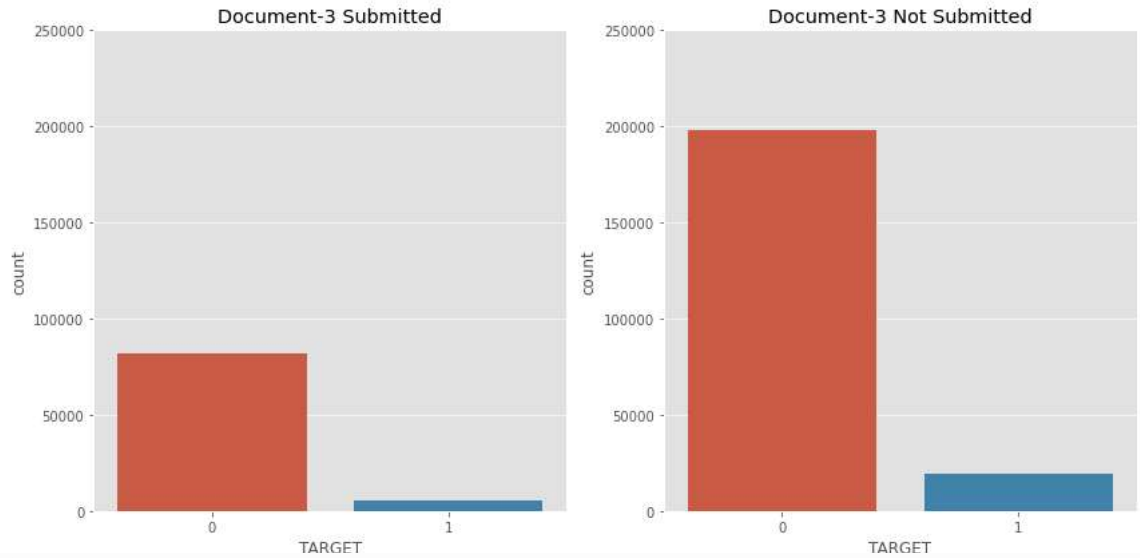


Insights :-

- Document 3 has been submitted by almost every applicant.
- We can asssume all other documents except document 3 will provide help in analyzing the data.

We will now reconfirm the importance of document 3.

```
fig = plt.figure(figsize=(12,6))

ax1 = fig.add_subplot(1, 2, 1, ylim=(0,250000), title="Document-3 Submitted")
ax2 = fig.add_subplot(1, 2, 2, ylim=(0,250000), title="Document-3 Not Submitted")

sns.countplot(application_df[application_df["FLAG_DOCUMENT_3"] == 0]["TARGET"], ax=ax1)
sns.countplot(application_df[application_df["FLAG_DOCUMENT_3"] == 1]["TARGET"], ax=ax2)

plt.tight_layout()

plt.show()
```

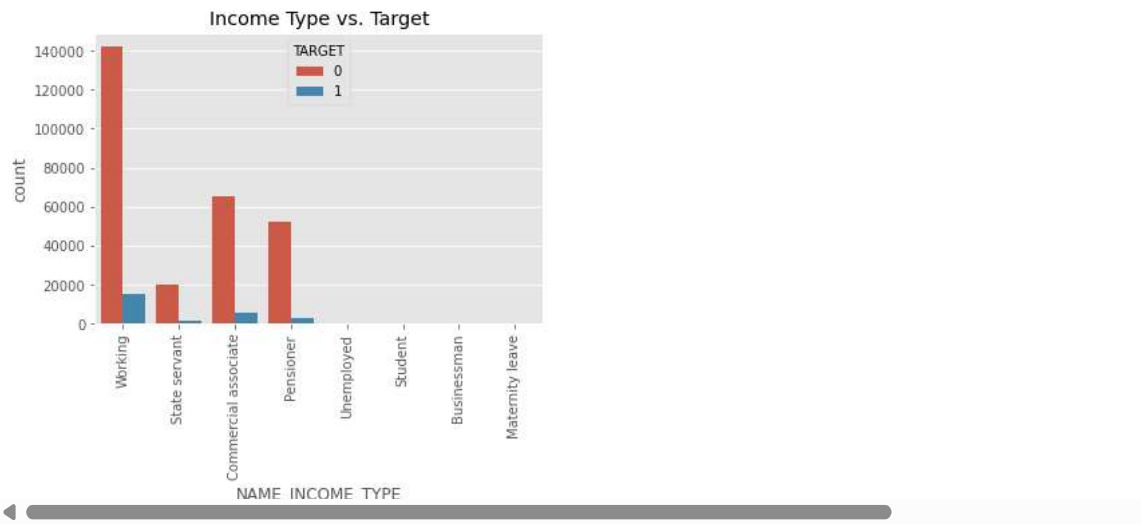


Lookig at the bar graph above, document 3 is shpwing similar trends for both defaulters and non-defaulters.

Hence we can drop this column.

Segment 5 :- Based on education level or occupation

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




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

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


value_wise_defaulter_percentage(application_df,'NAME_INCOME_TYPE')
```

	Value	Percentage of Defaulter
4	Unemployed	42.105263
7	Maternity leave	40.000000
0	Working	9.620506
2	Commercial associate	7.517586
1	State servant	5.761719
3	Pensioner	5.395598
5	Student	0.000000
6	Businessman	0.000000

```
value_wise_defaulter_percentage(application_df,'NAME_EDUCATION_TYPE')
```

	Value	Percentage of Defaulter
3	Lower secondary	10.897098
0	Secondary / secondary special	8.963349
2	Incomplete higher	8.501229
1	Higher education	5.381937
4	Academic degree	1.840491

```
value_wise_defaulter_percentage(application_df, 'OCCUPATION_TYPE')
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-51-c368d444f237> in <module>
----> 1 value_wise_defaulter_percentage(application_df, 'OCCUPATION_TYPE')

<ipython-input-46-83dad9d94a02> in value_wise_defaulter_percentage(df, col)
      5         default_cnt = df[(df[col] == value) & (df.TARGET == 1)].shape[0]
      6         total_cnt = df[df[col] == value].shape[0]
----> 7         new_df = new_df.append({'Value' : value , 'Percentage of Defaulter' : (default_cnt*100/total_cnt)}, ignore_index=True)
      8     return new_df.sort_values(by='Percentage of Defaulter', ascending=False)

ZeroDivisionError: division by zero
```

```
application_df['OCCUPATION_TYPE'].isnull().value_counts()
```

```
False    209096
True      95435
Name: OCCUPATION_TYPE, dtype: int64
```

```
application_df['OCCUPATION_TYPE'].value_counts()
```

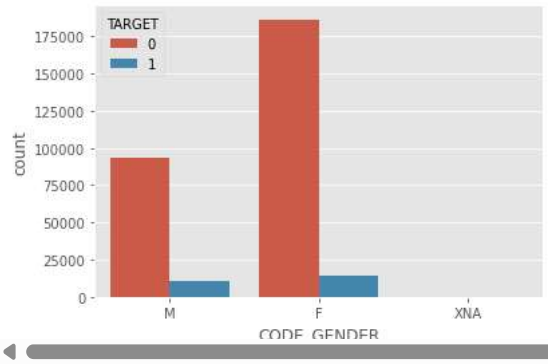
```
Laborers                54730
Sales staff             31790
Core staff              27263
Managers                21114
Drivers                 18456
High skill tech staff   11261
Accountants              9698
Medicine staff           8459
Security staff           6667
Cooking staff            5898
Cleaning staff           4615
Private service staff    2629
Low-skill Laborers       2077
Waiters/barmen staff     1335
Secretaries              1293
Realty agents             742
HR staff                  558
IT staff                  511
Name: OCCUPATION_TYPE, dtype: int64
```

```
# Imputing missing value for OCCUPATION TYPE as "Unknown"

application_df['OCCUPATION_TYPE'].fillna("Unknown", inplace=True)
```

```
value_wise_defaulter_percentage(application_df, 'OCCUPATION_TYPE')
```

	Value	Percentage of Defaulter
14	Low-skill Laborers	17.284545
13	Waiters/barmen staff	11.385768
5	Drivers	11.367577
11	Security staff	10.754462
0	Laborers	10.608441
8	Cooking staff	10.512038
7	Cleaning staff	9.642470
6	Sales staff	9.635105
15	Realty agents	7.951482
16	Secretaries	7.115236
10	Medicine staff	6.691098
17	IT staff	6.653620
9	Private service staff	6.542412
4	Unknown	6.534290
1	Core staff	6.330925
3	Managers	6.251776
12	High skill tech staff	6.180623
18	HR staff	6.093190
2	Accountants	4.866983

```
sns.countplot(application_df['CODE_GENDER'], hue=application_df["TARGET"])
plt.show()
```




```
value_wise_defaulter_percentage(application_df, 'CODE_GENDER')
```

	Value	Percentage of Defaulter
0	M	10.191744
1	F	7.014595
2	XNA	0.000000

Insights :-

- Applicants that are on maternity leave or unemployeed are mst likely to be defaulter.
- Businessmen and studennts have lowest default rate (zero).
- Applicants that have "Lower secondary" education are most likely to be defaulter as compared to others.
- Low skilled labourers have very high rate of defaulters.
- Although there are more female applicants but male aplicants are more likely to be defaulter than famale applicants

5) Correlation

a) Top 5 correlation for defaulters

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

corr_list = defaulter_corr.unstack()

# Listing the correlations in pair sorted in descending order
corr_list.sort_values(ascending=False).drop_duplicates().head(6)
```

```
SK_ID_CURR                SK_ID_CURR                  1.000000
OBS_30_CNT_SOCIAL_CIRCLE  OBS_60_CNT_SOCIAL_CIRCLE    0.998286
BASEMENTAREA_MEDI         BASEMENTAREA_AVG            0.998205
YEARS_BUILD_MEDI          YEARS_BUILD_AVG             0.998090
COMMONAREA_MEDI           COMMONAREA_AVG              0.998085
NONLIVINGAPARTMENTS_AVG   NONLIVINGAPARTMENTS_MEDI    0.998053
dtype: float64
```

Top 10 Correlations for Defaulters

1. (OBS_60_CNT_SOCIAL_CIRCLE, OBS_30_CNT_SOCIAL_CIRCLE)
2. (BASEMENTAREA_MEDI, BASEMENTAREA_AVG)
3. (YEARS_BUILD_MEDI, YEARS_BUILD_AVG)
4. (COMMONAREA_MEDI, COMMONAREA_AVG)
5. (NONLININGAPARTMENTS_AVG, NONLININGAPARTMENTS_MEDI)

b) Top 5 correlation for non-defaulters

```
nondefaulter_corr = non_default.corr()
round(nondefaulter_corr, 2)

nondf_corr_list = nondefaulter_corr.unstack()

# Listing the correlations in pair sorted in descending order
nondf_corr_list.sort_values(ascending=False).drop_duplicates().head(6)
```