Bank Loan Case Study
Final Project - 2

PROJECT DESCRIPTION:

In this project i.e. Bank Loan Case Study, I provided support in the effort to perform an exploratory data analysis (EDA) for a bank. In order to conduct the analysis and identify the answer to the problem statement, we will take into consideration the various forms of EDA (univariate, bivariate, and multivariate analysis) in this section., and by giving the data to the development teams. I am utilising a variety of Python queries during this process to obtain the relevant data.

APPROACH:

I initially examined the objective to find the relevant data that the team needed and then I imported the data into Jupyter Notebook and using different queries to find necessary solutions for the bank. By using the below Steps for EDA

- Understanding the datasets provided by the bank.
- Checking for missing data.
- Using the appropriate approach (mean, median or mode) to impute the missing data. Dropping columns with maximum null values.
- Identifying the outliers in the dataset.
- Drawing data summary such as plotting graphs, heatmaps, etc.
- Providing insights for the bank based on my understandings.

TECH-STACK USED:

I carried out the project using **Jupyter Notebook.**

INSIGHTS:

- I identify and clarify the main objective and by using Jupyter Notebook, I learn more about real-time analytics and obtain insights by applying various python queries on this project. I identified the bank database info that was provided. I gained various insights into the real time bank loan case analysis.

RESULTS:

By the completion of the project, I had improved my Excel skills, learned how to work with real-world data and experienced how to approach and resolve problems.

Based on the data provided, I was able to achieve a number of results which are listed below.

1. Understanding the datasets.

Importing and checking the datasets provided by the bank.

```
app_data = pd.read_csv('application_data.csv')
```

```
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
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	

5 rows × 122 columns

```
prev_app_data = pd.read_csv('previous_application.csv')
```

```
prev_app_data.head()
```

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

5 rows × 37 columns

2. Cleaning the data:

Counting the number of missing values in each column of the application_data.csv dataset.

 # calculate the percentage of missing values in each column

missing_data = app_data.isnull().mean() * 100

 # filter the columns with more than 50% missing values

columns_to_drop = missing_data[missing_data > 50].sort_values(ascending=False)

Some columns contains missing data more then 50% therefore I decided to drop these columns.

```
print(columns_to_drop)
```

```
COMMONAREA_AVG               69.872297
COMMONAREA_MEDI              69.872297
COMMONAREA_MODE              69.872297
NONLIVINGAPARTMENTS_MEDI     69.432963
NONLIVINGAPARTMENTS_AVG      69.432963
NONLIVINGAPARTMENTS_MODE     69.432963
FONDKAPREMONT_MODE           68.386172
LIVINGAPARTMENTS_MEDI        68.354953
LIVINGAPARTMENTS_AVG         68.354953
LIVINGAPARTMENTS_MODE        68.354953
FLOORSMIN_MODE               67.848630
FLOORSMIN_MEDI               67.848630
FLOORSMIN_AVG                67.848630
YEARS_BUILD_MODE             66.497784
YEARS_BUILD_AVG              66.497784
YEARS_BUILD_MEDI             66.497784
OWN_CAR_AGE                  65.990810
LANDAREA_AVG                 59.376738
LANDAREA_MEDI                59.376738
```

```
# Dropping the columns with huge missing data
app_data.drop(columns= columns_to_drop.index ,inplace=True)
```

Result: Columns reduced from 122 numbers to 81 numbers.

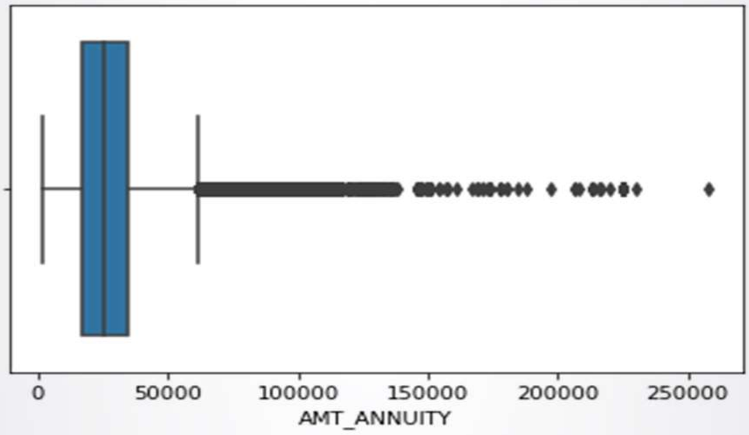
- Checking the missing values in data frame.

```
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                      12
AMT_GOODS_PRICE                 278
NAME_TYPE_SUITE                1292
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
```

- Checking the outliers by plotting a Box plot for AMT_ANNUITY .

```
sns.boxplot(app_data.AMT_ANNUITY)
plt.show()
```



.

AMT_ANNUITY have a lot of outliers therefore we fill the missing values using median

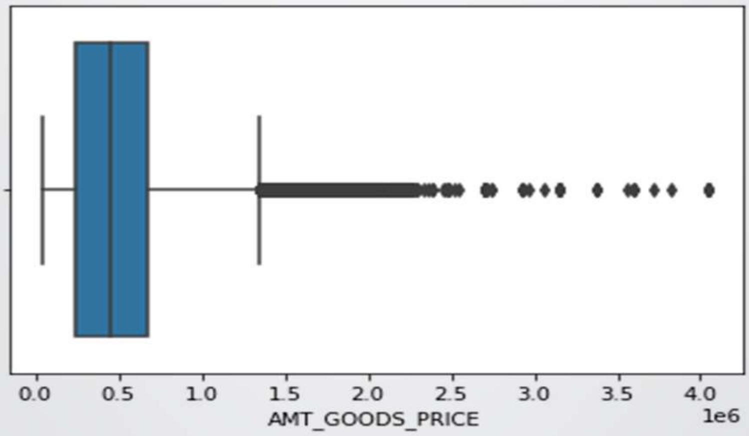
```
median_value = app_data['AMT_ANNUITY'].median()
median_value
```

Result: 24903.0

```
app_data['AMT_ANNUITY'] = app_data['AMT_ANNUITY'].fillna(median_value)
```

- Checking the outliers by plotting a Box plot for AMT_GOODS_PRICE

```
sns.boxplot(app_data.AMT_GOODS_PRICE)
plt.show()
```



AMT_GOODS_PRICE have a lot of outliers so we fill the missing values using median.

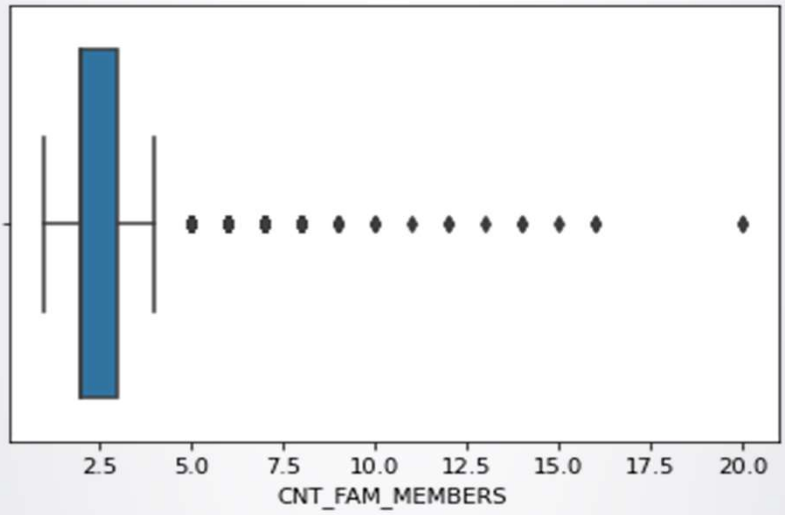
```
median_value = app_data['AMT_GOODS_PRICE'].median()
median_value
```

Result: 450000.0

```
app_data['AMT_GOODS_PRICE’] =
app_data['AMT_GOODS_PRICE'].fillna(median_value)
```

- NAME_TYPE_SUITE is a categorical value, using mode ('unaccompanied') to fill the null values.
- Checking the outliers by plotting a Box plot for CNT_FAM_MEMBERS.

```
sns.boxplot(app_data.CNT_FAM_MEMBERS)
plt.show()
```



.

CNT_FAM_MEMBERS have a lot of outliers therefore we fill the missing values using median

```
median_value = app_data['CNT_FAM_MEMBERS '].median()
median_value
```

Result: 2.0

```
app_data['CNT_FAM_MEMBERS’]=
app_data['CNT_FAM_MEMBERS'].fillna(median_value)
```

- Checking percentages of each category for OCCUPATION_TYPE

```
app_data['OCCUPATION_TYPE'].value_counts(normalize=True)*100
```

```
Laborers                 26.139636
Sales staff              15.205570
Core staff               13.058924
Managers                 10.122679
Drivers                   8.811576
High skill tech staff     5.390299
Accountants               4.648067
Medicine staff            4.043672
Security staff            3.183498
Cooking staff             2.816408
```

```
Cleaning staff            2.203960
Private service staff     1.256158
Low-skill Laborers        0.991379
Waiters/barmen staff      0.638499
Secretaries               0.618132
Realty agents             0.355722
HR staff                  0.266673
IT staff                  0.249147
Name: OCCUPATION_TYPE, dtype: float64
```

- Finding null values and filling the null value with 'unknown’.

```
app_data['OCCUPATION_TYPE'].isnull().sum()
```

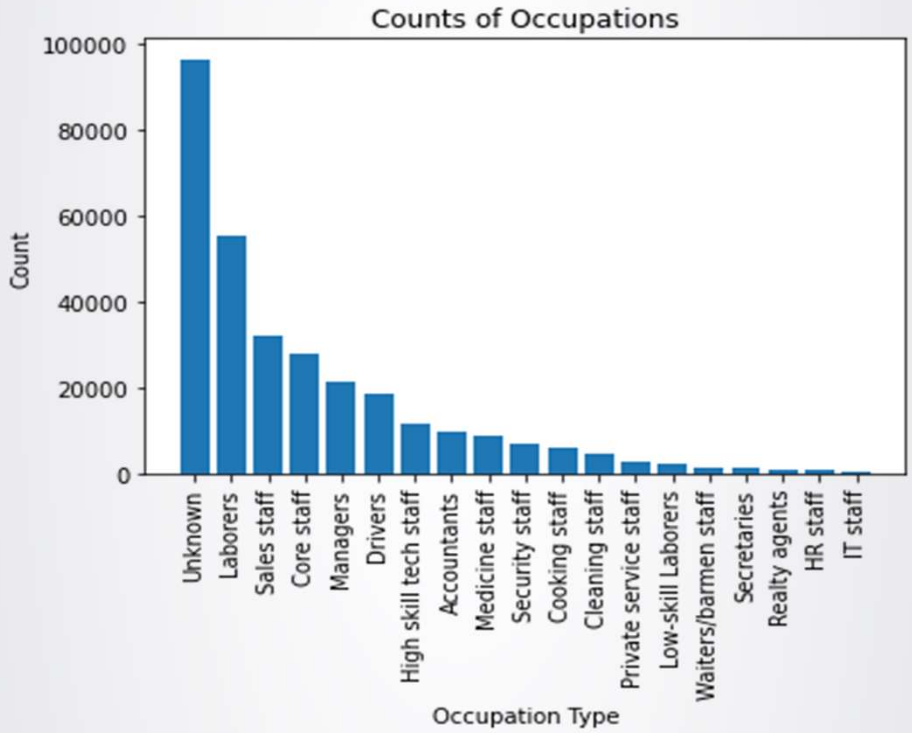
Result: 96391

```
app_data['OCCUPATION_TYPE']=
app_data['OCCUPATION_TYPE'].fillna('Unknown’)
app_data['OCCUPATION_TYPE'].value_counts(normalize=True)*100
```

```
Unknown                  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
```

Plotting a chart of the different categories of OCCUPATION_TYPE

```
occupation_counts = app_data['OCCUPATION_TYPE'].value_counts()
# Create a bar chart
fig, ax = plt.subplots()
ax.bar(occupation_counts.index, occupation_counts.values)
# Set chart title and axis labels
ax.set_title('Counts of Occupations')
ax.set_xlabel('Occupation Type')
ax.set_ylabel('Count')
# Rotate x-axis labels for better visibility
plt.xticks(rotation=90)
plt.show()
```



- **Checking client's SOCIAL_CIRCLE**

app_data[['OBS_30_CNT_SOCIAL_CIRCLE','DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE','DEF_60_CNT_SOCIAL_CIRCLE']].describe()

	OBS_30_CNT_SOCIAL_CIRCLE	DEF_30_CNT_SOCIAL_CIRCLE	OBS_60_CNT_SOCIAL_CIRCLE	DEF_60_CNT_SOCIAL_CIRCLE
count	306490.000000	306490.000000	306490.000000	306490.000000
mean	1.422245	0.143421	1.405292	0.100049
std	2.400989	0.446698	2.379803	0.362291
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	2.000000	0.000000	2.000000	0.000000
max	348.000000	34.000000	344.000000	24.000000

Replacing the missing values with median values for the social surroundings

#Making a list of all variables pertaining to client's social surroundings

SOCIAL_CIRCLE =

['OBS_30_CNT_SOCIAL_CIRCLE','DEF_30_CNT_SOCIAL_CIRCLE',

'OBS_60_CNT_SOCIAL_CIRCLE','DEF_60_CNT_SOCIAL_CIRCLE’]

#Replacing the missing values

app_data.fillna(app_data[SOCIAL_CIRCLE].median(),inplace = True)

- **For EXT_SOURCE**

```
pd.set_option('display.max_rows', 10)
#Since EXT_SOURCE_1 is already dropped
app_data[['EXT_SOURCE_2','EXT_SOURCE_3']]
```

	EXT_SOURCE_2	EXT_SOURCE_3
0	0.262949	0.139376
1	0.622246	NaN
2	0.555912	0.729567
3	0.650442	NaN
4	0.322738	NaN
...	...	...
307506	0.681632	NaN
307507	0.115992	NaN
307508	0.535722	0.218859
307509	0.514163	0.661024
307510	0.708569	0.113922

307511 rows × 2 columns

```
app_data[['EXT_SOURCE_2','EXT_SOURCE_3']].describe()
```

	EXT_SOURCE_2	EXT_SOURCE_3
count	3.068510e+05	246546.000000
mean	5.143927e-01	0.510853
std	1.910602e-01	0.194844
min	8.173617e-08	0.000527
25%	3.924574e-01	0.370650
50%	5.659614e-01	0.535276
75%	6.636171e-01	0.669057
max	8.549997e-01	0.896010

```
#Replacing the missing values with median values for EXT_SOURCE_2
median_value = app_data['EXT_SOURCE_2'].median()
median_value
```

Result: 0.5659614260608526

```
app_data['EXT_SOURCE_2'] = app_data['EXT_SOURCE_2'].fillna(median_value)
```

#Replacing the missing values with median values for EXT_SOURCE_2

median_value = app_data['EXT_SOURCE_3'].median()

median_value

Result: 0.5352762504724826

:app_data['EXT_SOURCE_3'] = app_data['EXT_SOURCE_3'].fillna(median_value)

- **Statistics for DAYS_LAST_PHONE_CHANGE**

app_data.DAYS_LAST_PHONE_CHANGE.describe()

```
count    307510.000000
mean       -962.858788
std         826.808487
min       -4292.000000
25%       -1570.000000
50%        -757.000000
75%        -274.000000
max           0.000000
Name: DAYS_LAST_PHONE_CHANGE, dtype: float64
```

app_data['DAYS_LAST_PHONE_CHANGE'].value_counts(normalize=True)

```
 0.0       0.122507
-1.0       0.009144
-2.0       0.007538
-3.0       0.005733
-4.0       0.004179
             ...
-4051.0    0.000003
-3593.0    0.000003
-3622.0    0.000003
-3570.0    0.000003
-3538.0    0.000003
Name: DAYS_LAST_PHONE_CHANGE, Length: 3773, dtype: float64
```

using mode value = 0 (most occurring) to fill the missing data

app_data['DAYS_LAST_PHONE_CHANGE'] =

app_data['DAYS_LAST_PHONE_CHANGE'].fillna(0)

For Previous Application data

prev_app_data.count()median_value

```
SK_ID_PREV                   1670214
SK_ID_CURR                   1670214
NAME_CONTRACT_TYPE           1670214
AMT_ANNUITY                  1297979
AMT_APPLICATION              1670214
                              ...
DAYS_FIRST_DUE                997149
DAYS_LAST_DUE_1ST_VERSION     997149
DAYS_LAST_DUE                 997149
DAYS_TERMINATION              997149
NFLAG_INSURED_ON_APPROVAL     997149
Length: 37, dtype: int64
```

prev_app_data.isnull().sum()

- **Checking missing value percentage for previous application data**

missing_percentage = prev_app_data.isnull().sum() / len(prev_app_data) * 100

print(missing_percentage.sort_values(ascending=True))

```
SK_ID_PREV                     0.000000
NAME_YIELD_GROUP               0.000000
NAME_SELLER_INDUSTRY           0.000000
SELLERPLACE_AREA               0.000000
CHANNEL_TYPE                   0.000000
NAME_PRODUCT_TYPE              0.000000
NAME_PORTFOLIO                 0.000000
NAME_GOODS_CATEGORY            0.000000
NAME_CLIENT_TYPE               0.000000
CODE_REJECT_REASON             0.000000
DAYS_DECISION                  0.000000
NAME_CONTRACT_STATUS           0.000000
NAME_CASH_LOAN_PURPOSE         0.000000
NAME_PAYMENT_TYPE              0.000000
AMT_APPLICATION                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
NAME_CONTRACT_TYPE             0.000000
SK_ID_CURR                     0.000000
AMT_CREDIT                     0.000060
PRODUCT_COMBINATION            0.020716
CNT_PAYMENT                   22.286366
AMT_ANNUITY                   22.286665
AMT_GOODS_PRICE               23.081773
DAYS_LAST_DUE                 40.298129
DAYS_LAST_DUE_1ST_VERSION     40.298129
DAYS_FIRST_DUE                40.298129
DAYS_FIRST_DRAWING            40.298129
NFLAG_INSURED_ON_APPROVAL     40.298129
DAYS_TERMINATION              40.298129
NAME_TYPE_SUITE               49.119754
AMT_DOWN_PAYMENT              53.636480
RATE_DOWN_PAYMENT             53.636480
RATE_INTEREST_PRIMARY         99.643698
RATE_INTEREST_PRIVILEGED      99.643698
dtype: float64
```

Dropping columns with missing value more then 50%

col_to_drop2 = missing_percentage[missing_percentage > 50].index.tolist()

print(col_to_drop2)

Result : ['AMT_DOWN_PAYMENT', 'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY', 'RATE_INTEREST_PRIVILEGED']

prev_app_data.drop(columns= col_to_drop2 ,inplace=True)

```
prev_app_data.head()
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE	WEEKDAY_APPR_PROCESS_START
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
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	450000.0	MONDAY
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	337500.0	THURSDAY

5 rows × 33 columns

- **Replacing missing values**

```
prev_app_data.NAME_TYPE_SUITE.value_counts()
```

```
Unaccompanied      508970
Family             213263
Spouse, partner     67069
Children            31566
Other_B             17624
Other_A              9077
Group of people      2240
Name: NAME_TYPE_SUITE, dtype: int64
```

```
prev_app_data['NAME_TYPE_SUITE'] =
prev_app_data['NAME_TYPE_SUITE'].fillna('Unaccompanied’)
prev_app_data.NAME_TYPE_SUITE.value_counts()
```

```
Unaccompanied      1329375
Family              213263
Spouse, partner      67069
Children             31566
Other_B              17624
Other_A               9077
Group of people       2240
Name: NAME_TYPE_SUITE, dtype: int64
```

- **Filling the missing values with mode for PRODUCT_COMBINATION**

```
prev_app_data['PRODUCT_COMBINATION'] =
prev_app_data.PRODUCT_COMBINATION.fillna(prev_app_data.PRODUCT_COMB
INATION.mode()[0])
```

- **Replacing the null values with median**

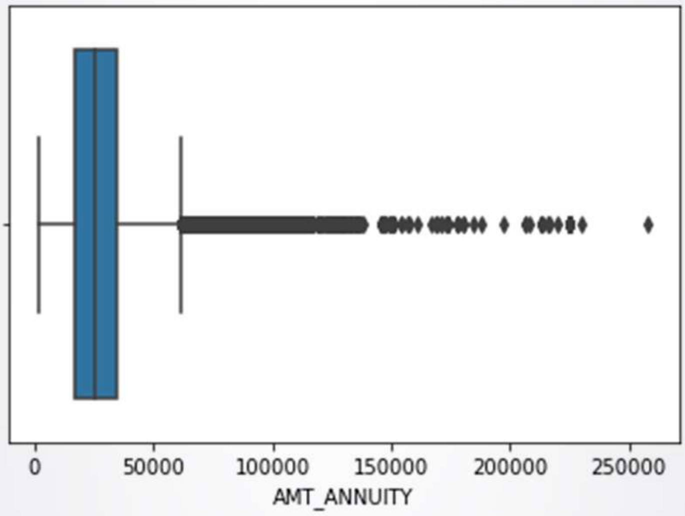
```
prev_app_data["AMT_GOODS_PRICE"].fillna(prev_app_data["AMT_GOODS_PRICE"].me
dian(), inplace=True)
prev_app_data["AMT_ANNUITY"].fillna(prev_app_data["AMT_ANNUITY"].median(),
inplace=True)
prev_app_data["CNT_PAYMENT"].fillna(prev_app_data["CNT_PAYMENT"].median(),
inplace=True)
```

Part 2

- **Identifying outliers**

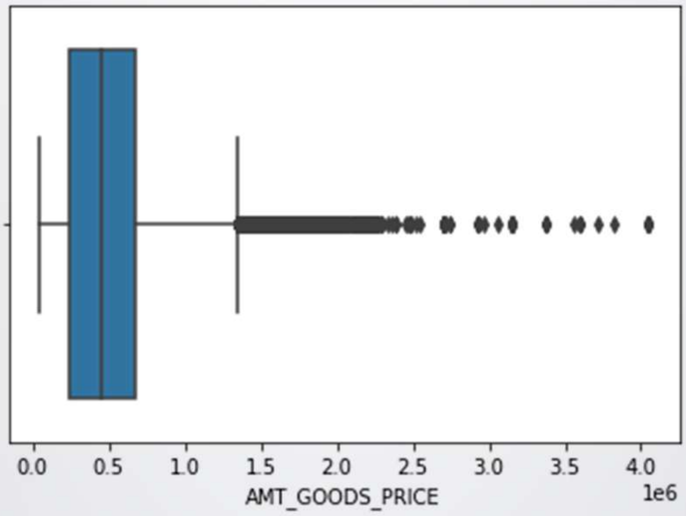
sns.boxplot(app_data.AMT_ANNUITY)

plt.show()



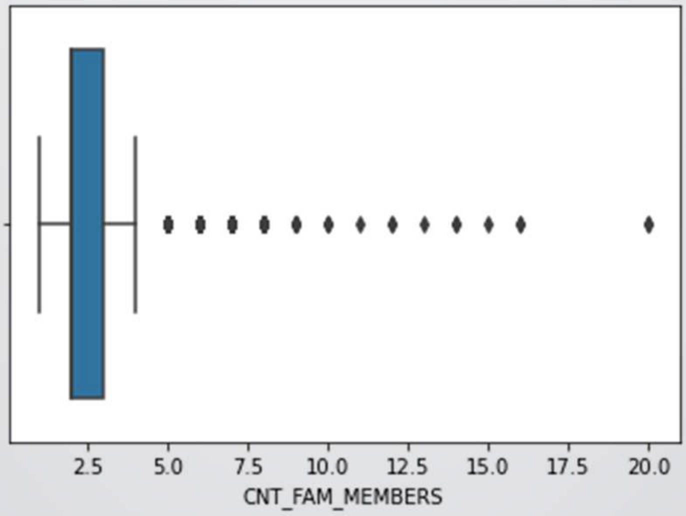
sns.boxplot(app_data.AMT_GOODS_PRICE)

plt.show()

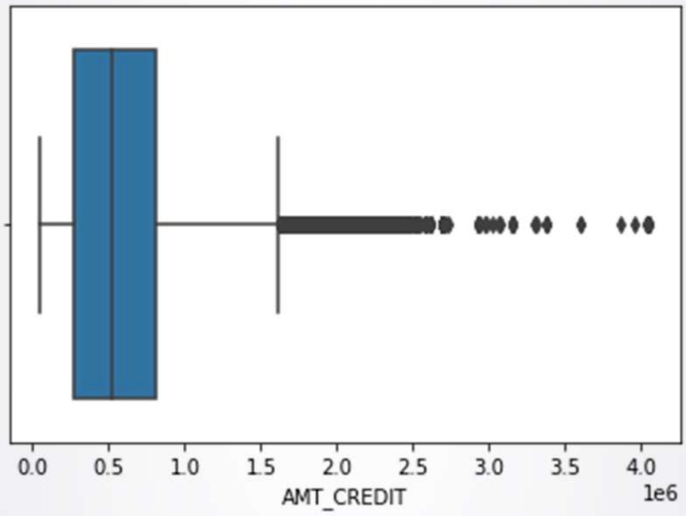


sns.boxplot(app_data.CNT_FAM_MEMBERS)

plt.show()

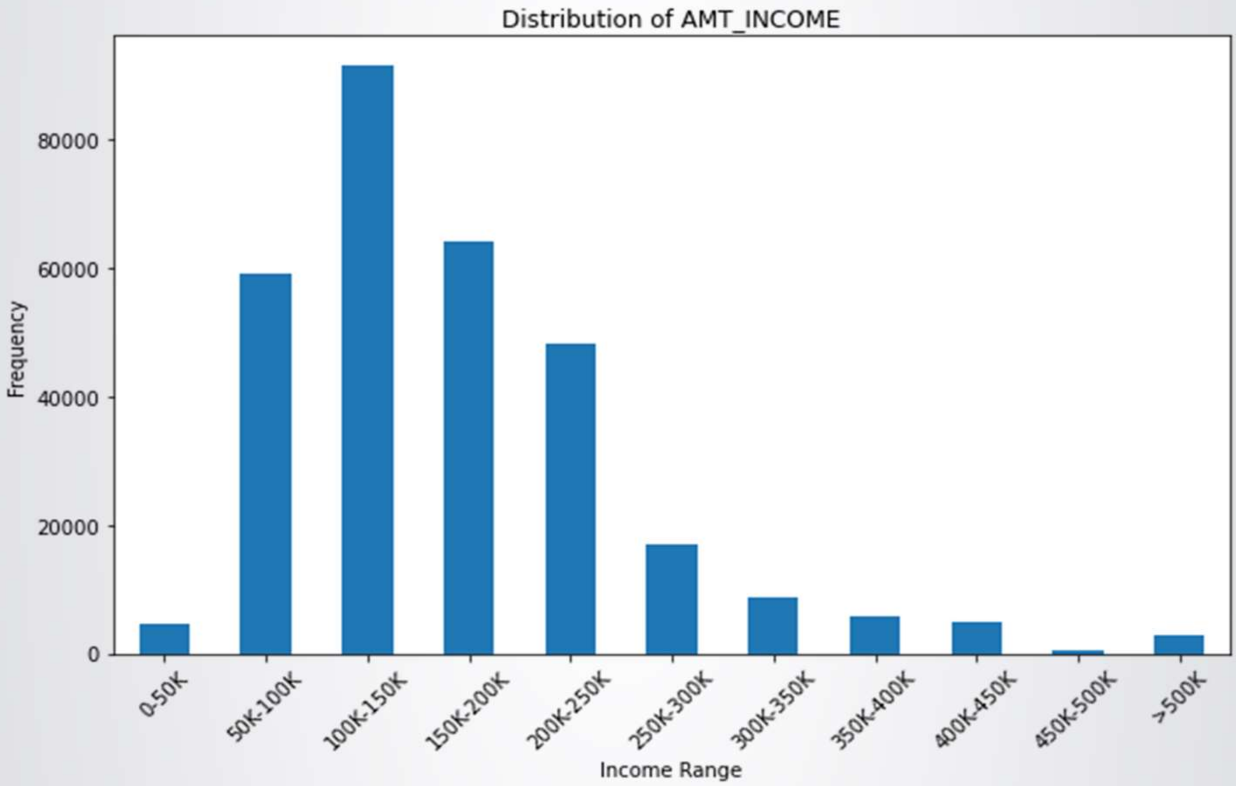


```
sns.boxplot(app_data.AMT_CREDIT)
plt.show()
```



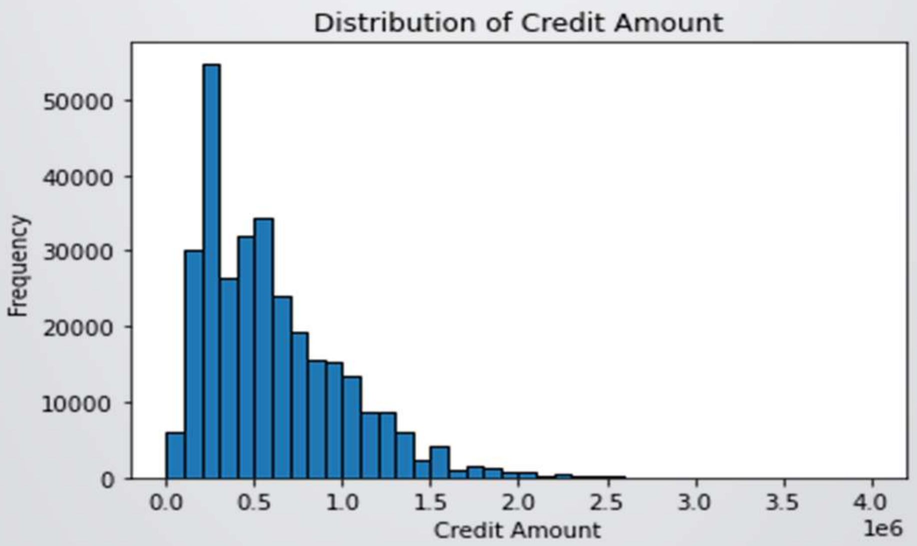
- **Distribution of bin for AMT_INCOME**

```
# Define the bin labels
bin_labels = ['0-50K', '50K-100K', '100K-150K', '150K-200K', '200K-250K', '250K-300K', '300K-350K', '350K-400K', '400K-450K', '450K-500K', '>500K']

# Perform the binning using pd.cut
app_data['Income Bin'] = pd.cut(app_data['AMT_INCOME_TOTAL'], bins=bin_edges, labels=bin_labels)

# Calculate the frequency count for each bin
bin_counts = app_data['Income Bin'].value_counts().sort_index()

# Plot a bar chart of the bin frequencies
bin_counts.plot(kind='bar', figsize=(10, 6))
plt.title('Distribution of AMT_INCOME')
plt.xlabel('Income Range')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()
```



- **Distribution of CREDIT_AMOUNT**

```
# create bins
bins = range(0, int(app_data['AMT_CREDIT'].max()), 100000)
# create histogram
plt.hist(app_data['AMT_CREDIT'], bins=bins, edgecolor='black')
# add labels and title
plt.xlabel('Credit Amount')
plt.ylabel('Frequency')
plt.title('Distribution of Credit Amount')
plt.show()
```



▪ Identifying data imbalance

```
# Count the number of observations for each target class
target_counts = app_data['TARGET'].value_counts()
```

▪ Calculate the ratio of data imbalance

```
imbalance_ratio = target_counts[1] / target_counts[0]
print("Ratio of data imbalance: {:.2f}".format(imbalance_ratio
```

Ratio of data imbalance: 0.09

```
target_counts = app_data['TARGET'].value_counts()
print(target_counts)
```

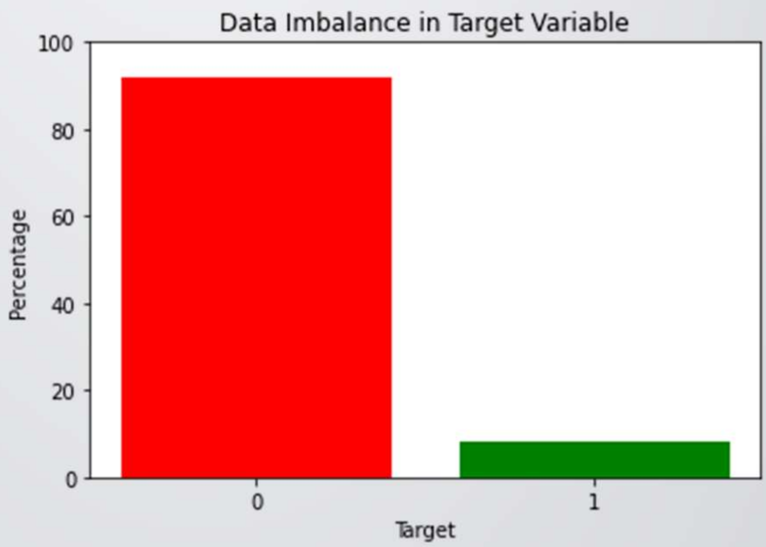
0 - 282686

1 - 24825

Name: TARGET, dtype: int64

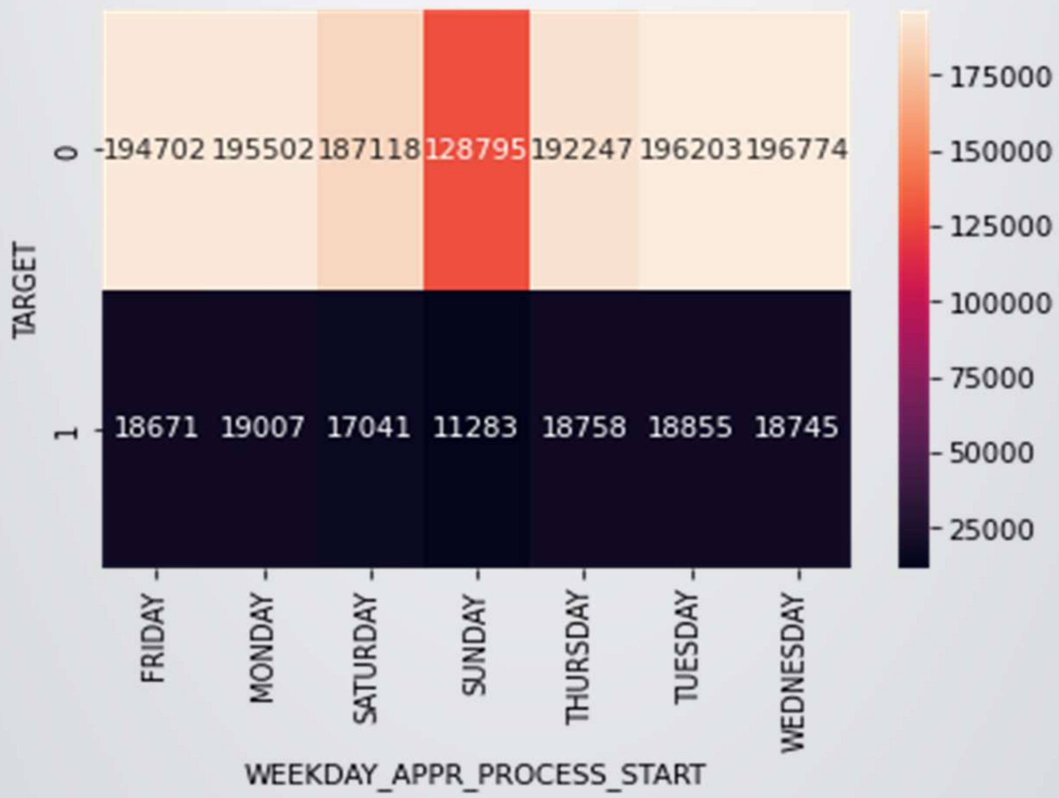
▪ Calculate the ratio of target variable

```
target_ratio = app_data["TARGET"].value_counts(normalize=True)
# plot a bar chart of the ratio
fig, ax = plt.subplots()
ax.bar(target_ratio.index.astype(str), target_ratio.values*100, color=['red', 'green'])
ax.set_title("Data Imbalance in Target Variable")
ax.set_xlabel("Target")
ax.set_ylabel("Percentage")
ax.set_ylim([0,100])
plt.show()
```



- **Univariate Analysis**

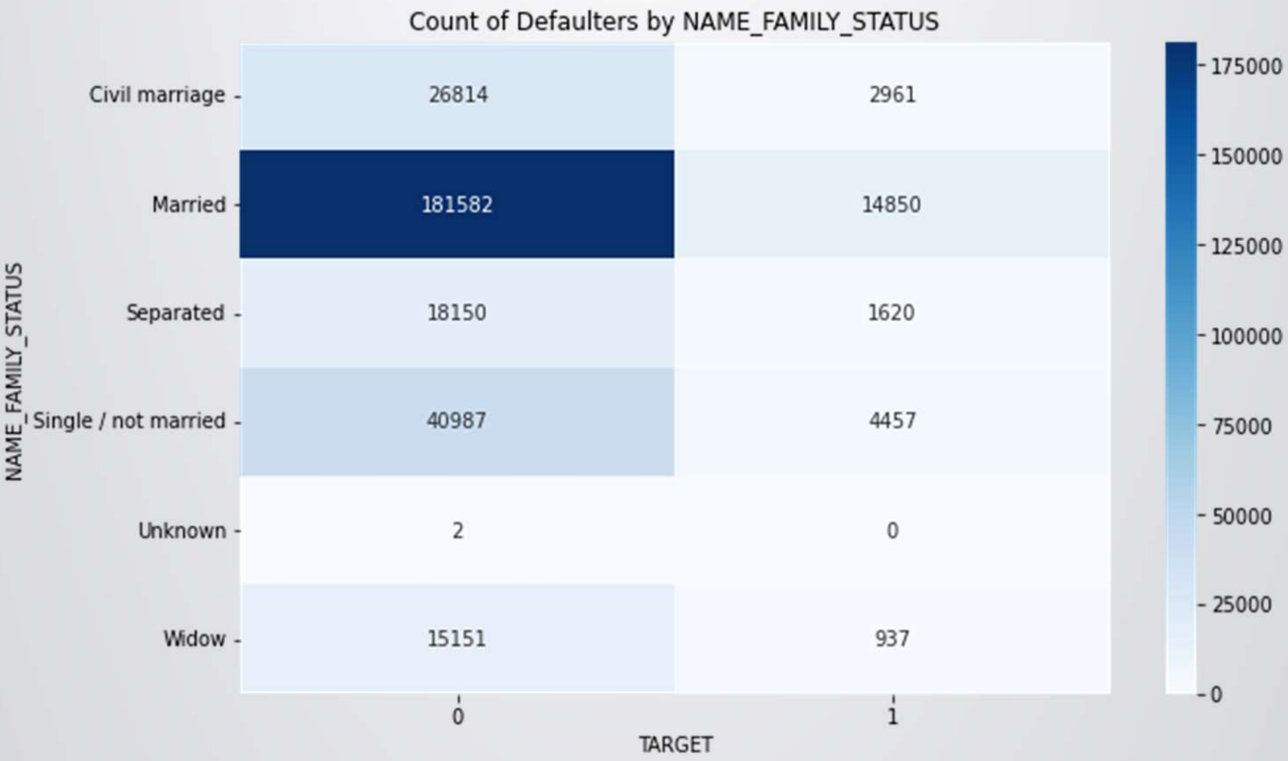
```
# Merge datasets using SK_ID_CURR as the key
merged_data = pd.merge(app_data[['SK_ID_CURR', 'TARGET']],
prev_app_data[['SK_ID_CURR', 'WEEKDAY_APPR_PROCESS_START']],
on='SK_ID_CURR', how='inner')

# Group by weekday and target, and count the number of occurrences in each group
grouped_data = merged_data.groupby(['WEEKDAY_APPR_PROCESS_START',
'TARGET']).size().reset_index(name='count')

# Pivot the table to have weekday as columns and target as index
pivoted_data = grouped_data.pivot(index='TARGET',
columns='WEEKDAY_APPR_PROCESS_START', values='count')

# Plot the heatmap
sns.heatmap(pivoted_data, annot=True, fmt='g')
```



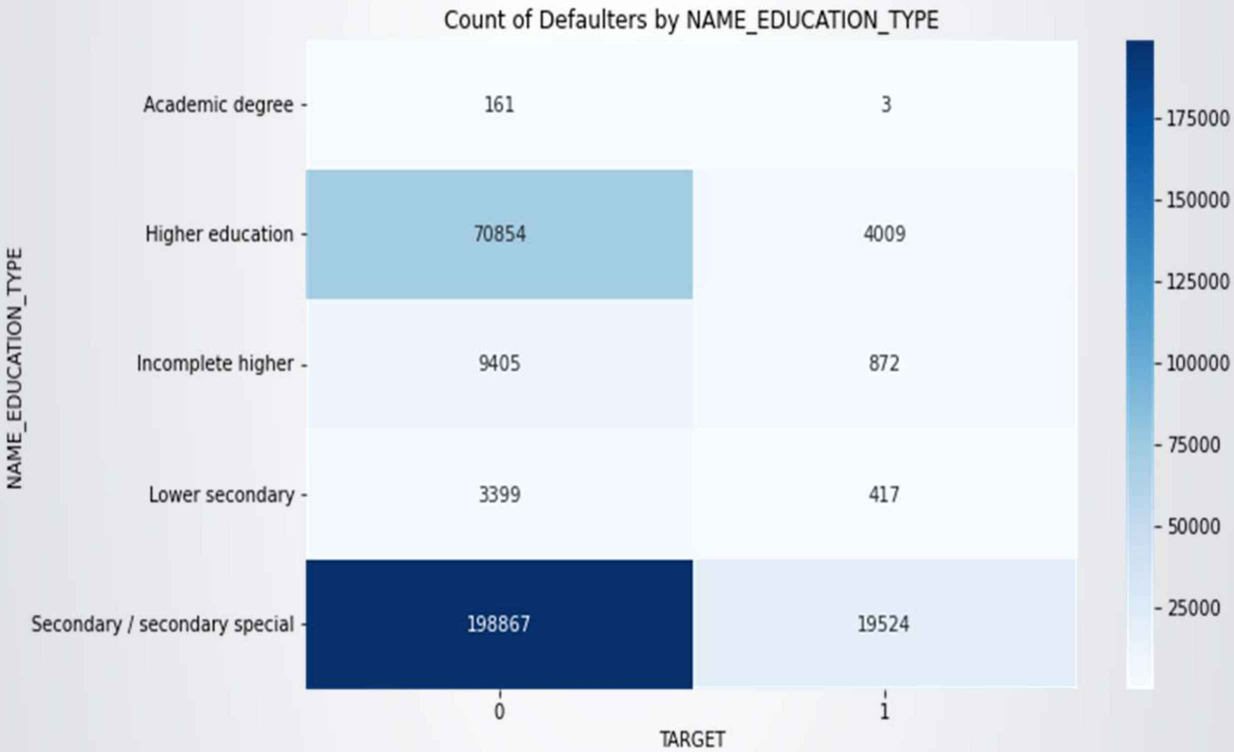
- **Graphical representations of clients data**

```
import seaborn as sns
columns = ['NAME_FAMILY_STATUS', 'NAME_EDUCATION_TYPE',
'NAME_HOUSING_TYPE', 'NAME_INCOME_TYPE', 'OCCUPATION_TYPE']
for col in columns:
    plt.figure(figsize=(10,6))
    plt.title(f"Count of Defaulters by {col}")
    sns.heatmap(app_data.pivot_table(index=col, columns='TARGET',
values='SK_ID_CURR', aggfunc='count', fill_value=0), annot=True, fmt='g',
cmap='Blues')
    plt.show()
```

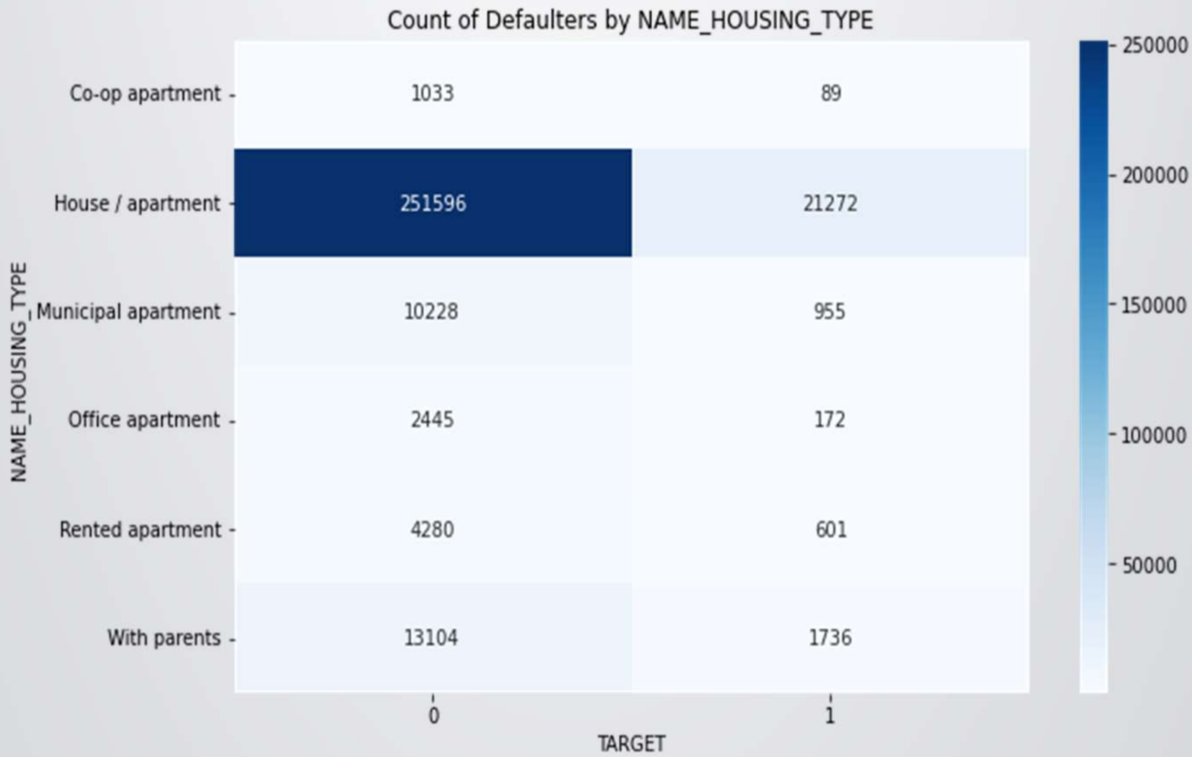
- **Family status**



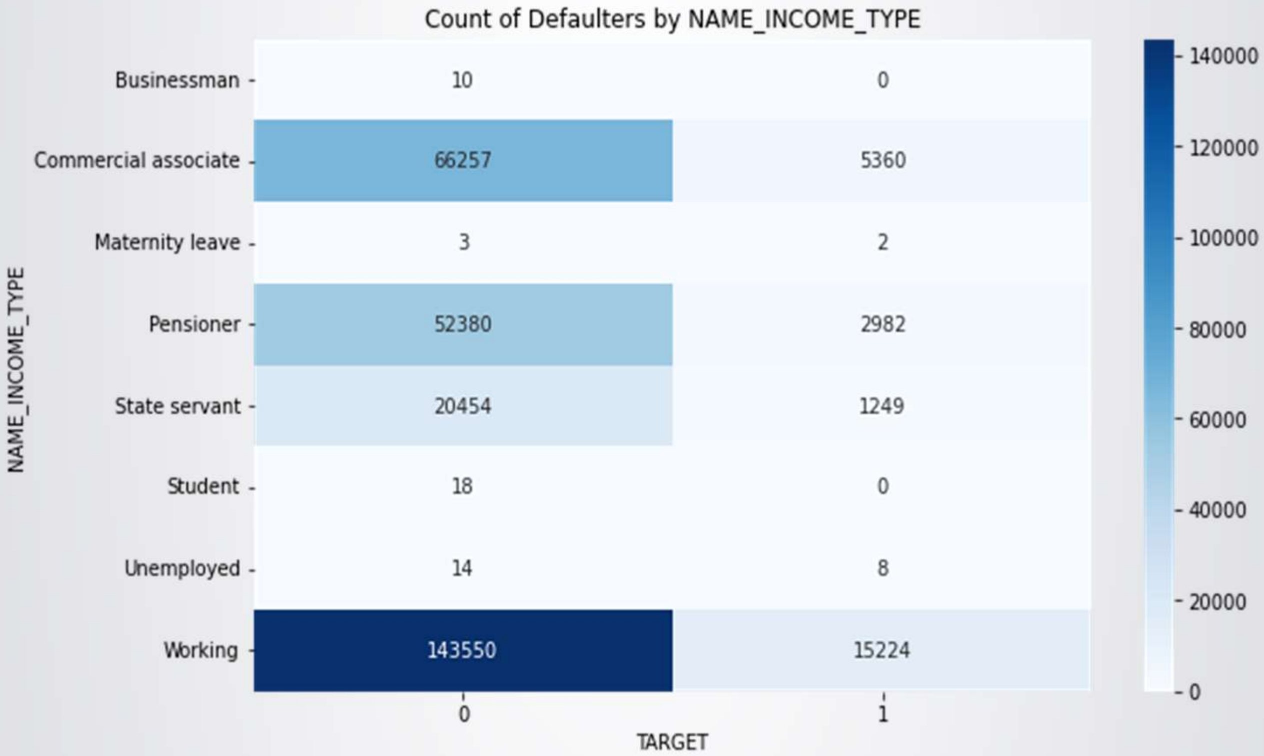
- **Education Type**



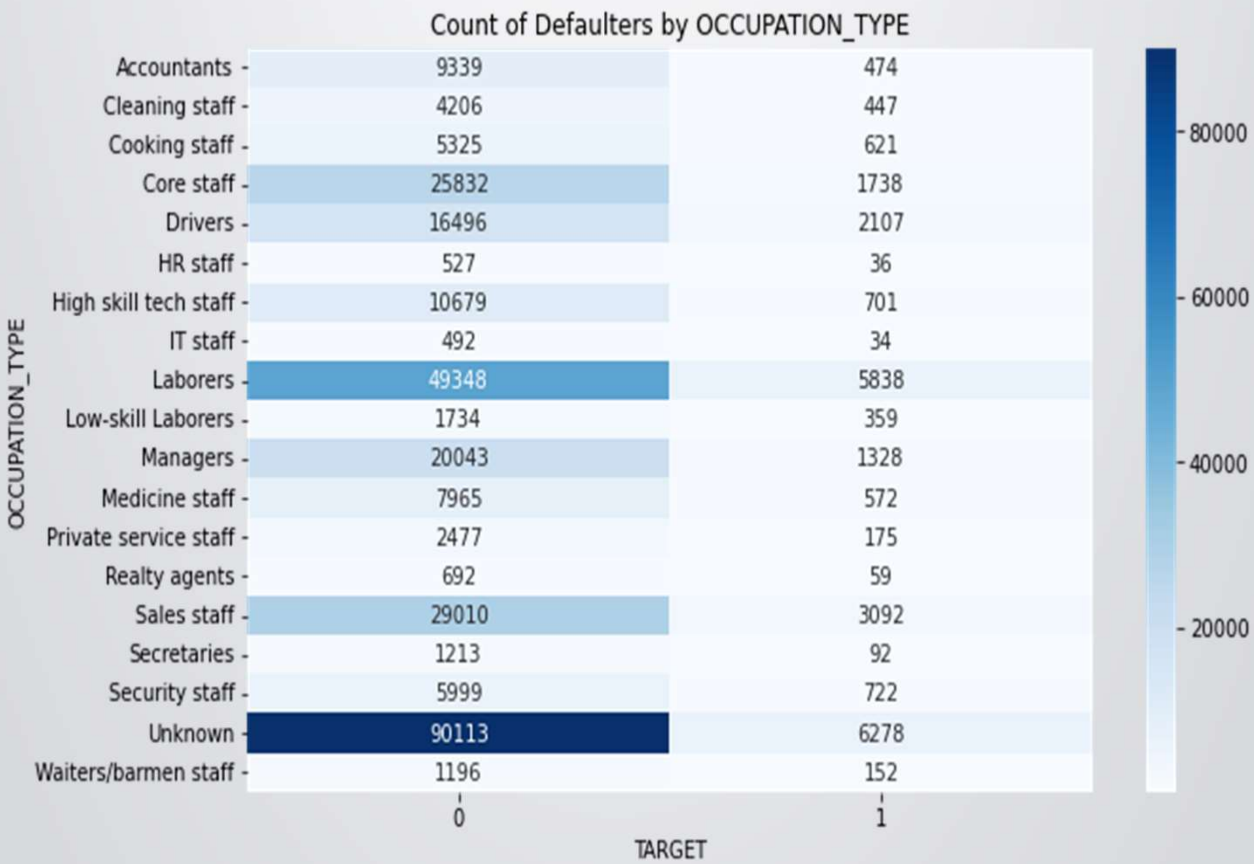
- **Housing Type**



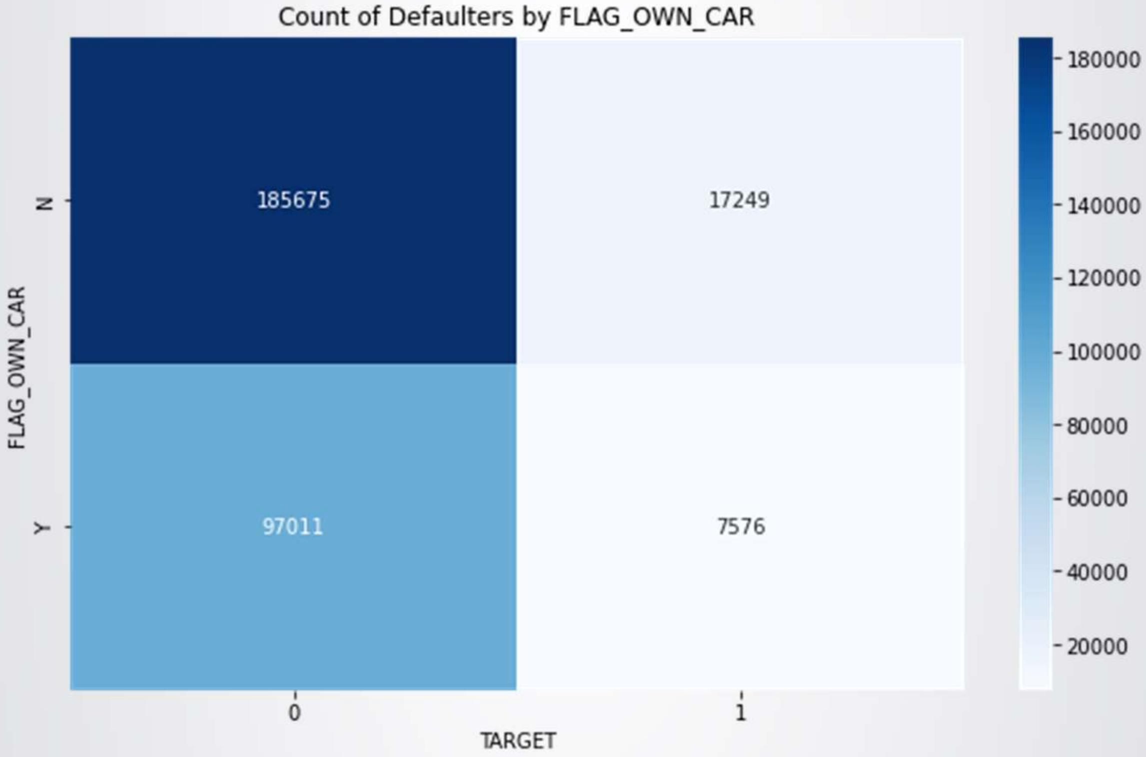
- **Income Type**



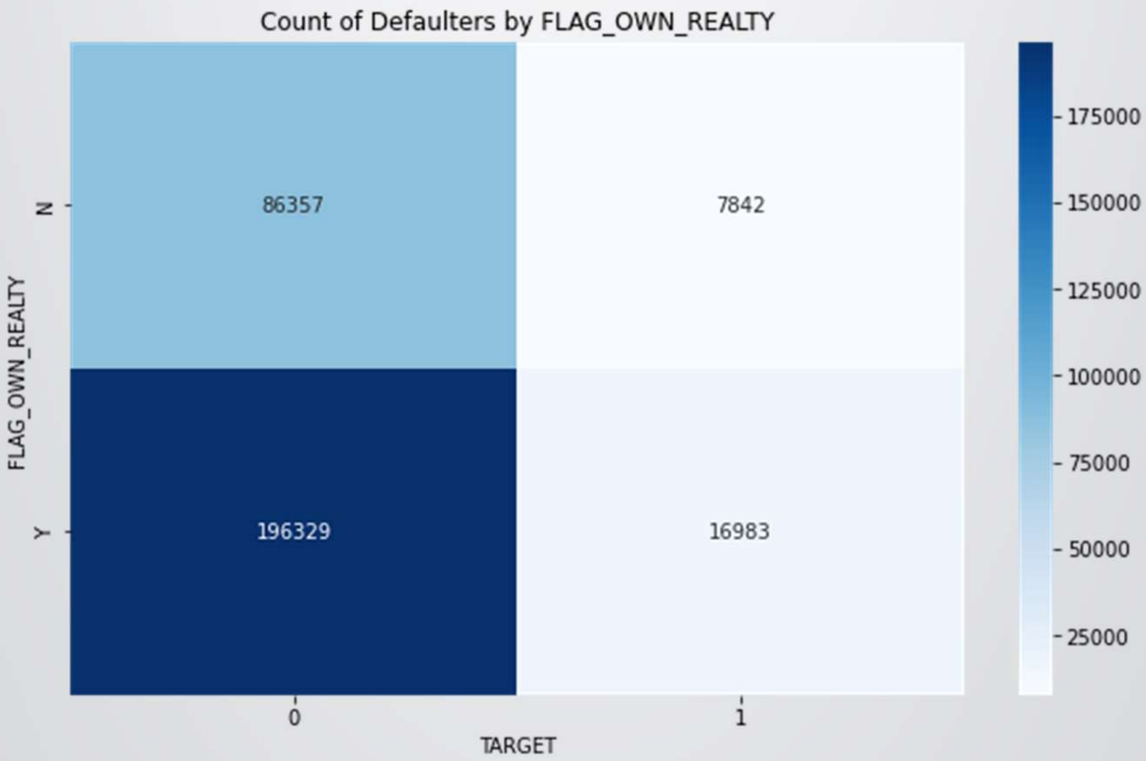
- **Occupation Type**



- **Having cars**



- **Having real estate**



Based on the analysis, we can conclude the following insights:

- Clients who have payment difficulties tend to have lower credit amounts and lower incomes compared to those who do not have payment difficulties.
- Clients who own a car or real estate are less likely to have payment difficulties compared to those who do not own these assets.
- Clients who have a higher family size tend to have a higher chance of payment difficulties.
- Clients who have lower education levels tend to have a higher chance of payment difficulties.
- Clients who are in the "Laborers" or "Low-skill Laborers" occupation tend to have a higher chance of payment difficulties.
- Clients who have a higher number of past loans or applications tend to have a higher chance of payment difficulties.
- Clients who have a lower age tend to have a higher chance of payment difficulties.
- Clients who apply for cash loans or revolving loans tend to have a higher chance of payment difficulties compared to those who apply for consumer loans or loans for specific purposes.

With the information obtained from the provided data, the banks can utilise these insights to evaluate and manage risk when deciding whether to grant loans to clients.