*Name: Nikhil Zodape*

*Email Id.: Nikszodape@gmail.com*

*Project Name: Bank Loan Case Study*

*Using: Python*

## Problem Statement:

As a data analyst at a finance company specializing in urban lending, the primary challenge is addressing default rates among customers with insufficient credit history. Some applications exploit this gaps, resulting in loan defaults. The objective is to conduct Exploratory Data Analysis (EDA) to uncover patterns in the data, ensuring that deserving application are not unfairly rejected.

When customers apply for loans, four potential outcomes exist: Approval, Cancellation during the approval process, Rejection and Approval with the loan remaining, unused. The overarching goal is to identify patterns that signal a customer's likelihood of struggling with instalment payments. This insight can inform decision such as loan denial, reducing loan amount, or offering loans with higher interest rates to higher-risk application. Ultimately, the company aims to discern the key factors influencing loan default, enabling more information decisions in the loan approval process.

## Task:

1. Identify the missing data in the dataset and decide on an appropriate method to deal with it.
2. Detect and identify outliers in the dataset.
3. Determine if there is data imbalance in the loan application dataset and calculate the ration of data imbalance.
4. Perform univariate analysis to understand the distribution of individual variables, segmented univariate analysis to compare variable distributions for difference scenarios and bivariate analysis to explore relationships between variables and the target variable.
5. Identify the top correlations for each segements.

## Data Understanding:

The dataset contain 3 files:

1. Application_data.csv: It contains all the information of the client at the time of application. The data is about the client is having any payment difficulty.
2. Previous_application.csv: It contain information about clients previous loan data. It contains the data whether the previous application had been Approved, Cancelled, Refused or Unused offer.
3. Columns_description.csv: It is the data dictionary which describe the meaning of the variables.

Data Cleaning:

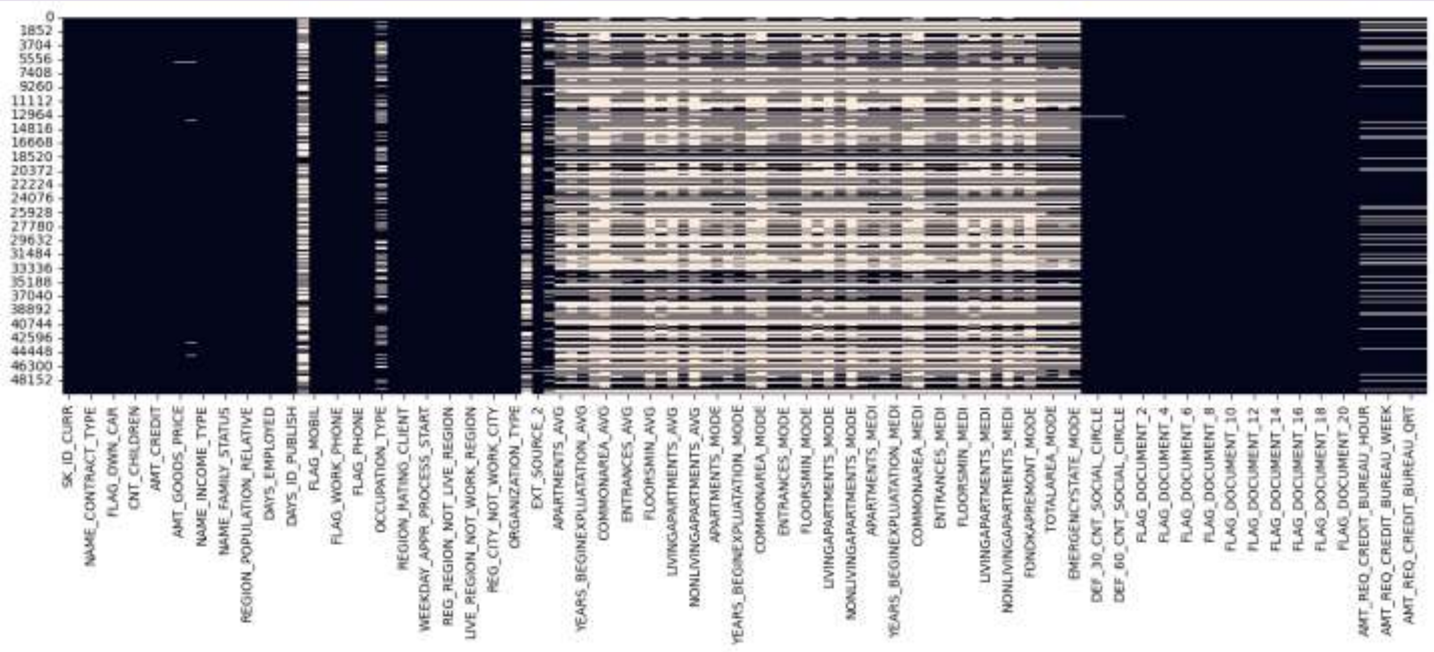
-→ Application data:

The application dataset contains 49999 rows and 122 columns.

Checking the Null values in application dataset.

| | column_name | null_count | null_percentage |
|---|---|---|---|
| 76 | COMMONAREA_MEDI | 34960 | 69.92 |
| 48 | COMMONAREA_AVG | 34960 | 69.92 |
| 62 | COMMONAREA_MODE | 34960 | 69.92 |
| 70 | NONLIVINGAPARTMENTS_MODE | 34714 | 69.43 |
| 56 | NONLIVINGAPARTMENTS_AVG | 34714 | 69.43 |
| 84 | NONLIVINGAPARTMENTS_MEDI | 34714 | 69.43 |
| 68 | LIVINGAPARTMENTS_MODE | 34226 | 68.45 |
| 54 | LIVINGAPARTMENTS_AVG | 34226 | 68.45 |
| 82 | LIVINGAPARTMENTS_MEDI | 34226 | 68.45 |
| 86 | FONDKAPREMONT_MODE | 34191 | 68.38 |
| 52 | FLOORSMIN_AVG | 33894 | 67.79 |
| 66 | FLOORSMIN_MODE | 33894 | 67.79 |
| 80 | FLOORSMIN_MEDI | 33894 | 67.79 |
| 75 | YEARS_BUILD_MEDI | 33239 | 66.48 |
| 61 | YEARS_BUILD_MODE | 33239 | 66.48 |
| 47 | YEARS_BUILD_AVG | 33239 | 66.48 |
| 21 | OWN_CAR_AGE | 32950 | 65.90 |
| 81 | LANDAREA_MEDI | 29721 | 59.44 |
| 67 | LANDAREA_MODE | 29721 | 59.44 |
| 53 | LANDAREA_AVG | 29721 | 59.44 |
| 73 | BASEMENTAREA_MEDI | 29199 | 58.40 |
| 45 | BASEMENTAREA_AVG | 29199 | 58.40 |
| 59 | BASEMENTAREA_MODE | 29199 | 58.40 |
| 41 | EXT_SOURCE_1 | 28172 | 56.35 |
| 71 | NONLIVINGAREA_MODE | 27572 | 55.15 |
| 57 | NONLIVINGAREA_AVG | 27572 | 55.15 |
| 85 | NONLIVINGAREA_MEDI | 27572 | 55.15 |
| 77 | ELEVATORS_MEDI | 26651 | 53.30 |
| 49 | ELEVATORS_AVG | 26651 | 53.30 |
| 63 | ELEVATORS_MODE | 26651 | 53.30 |
| 89 | WALLSMATERIAL_MODE | 25459 | 50.92 |
| 72 | APARTMENTS_MEDI | 25385 | 50.77 |
| 44 | APARTMENTS_AVG | 25385 | 50.77 |
| 58 | APARTMENTS_MODE | 25385 | 50.77 |
| 78 | ENTRANCES_MEDI | 25195 | 50.39 |
| 50 | ENTRANCES_AVG | 25195 | 50.39 |
| 64 | ENTRANCES_MODE | 25195 | 50.39 |
| 55 | LIVINGAREA_AVG | 25137 | 50.28 |
| 69 | LIVINGAREA_MODE | 25137 | 50.28 |
| 83 | LIVINGAREA_MEDI | 25137 | 50.28 |
| 87 | HOUSETYPE_MODE | 25075 | 50.15 |
| 65 | FLOORSMAX_MODE | 24875 | 49.75 |
| 79 | FLOORSMAX_MEDI | 24875 | 49.75 |
| 51 | FLOORSMAX_AVG | 24875 | 49.75 |
| 60 | YEARS_BEGINEXPLUATATION_MODE | 24394 | 48.79 |
| 74 | YEARS_BEGINEXPLUATATION_MEDI | 24394 | 48.79 |
| 46 | YEARS_BEGINEXPLUATATION_AVG | 24394 | 48.79 |
| 88 | TOTALAREA_MODE | 24148 | 48.30 |
| 90 | EMERGENCYSTATE_MODE | 23698 | 47.40 |
| 28 | OCCUPATION_TYPE | 15654 | 31.31 |
| 43 | EXT_SOURCE_3 | 9944 | 19.89 |
| 116 | AMT_REQ_CREDIT_BUREAU_HOUR | 6734 | 13.47 |
| 117 | AMT_REQ_CREDIT_BUREAU_DAY | 6734 | 13.47 |
| 118 | AMT_REQ_CREDIT_BUREAU_WEEK | 6734 | 13.47 |
| 119 | AMT_REQ_CREDIT_BUREAU_MON | 6734 | 13.47 |
| 120 | AMT_REQ_CREDIT_BUREAU_QRT | 6734 | 13.47 |
| 121 | AMT_REQ_CREDIT_BUREAU_YEAR | 6734 | 13.47 |
| 11 | NAME_TYPE_SUITE | 192 | 0.38 |
| 92 | DEF_30_CNT_SOCIAL_CIRCLE | 168 | 0.34 |
| 91 | OBS_30_CNT_SOCIAL_CIRCLE | 168 | 0.34 |
| 93 | OBS_60_CNT_SOCIAL_CIRCLE | 168 | 0.34 |
| 94 | DEF_60_CNT_SOCIAL_CIRCLE | 168 | 0.34 |
| 42 | EXT_SOURCE_2 | 126 | 0.25 |
| 10 | AMT_GOODS_PRICE | 38 | 0.08 |
| 6 | CNT_CHILDREN | 0 | 0.00 |
| 102 | FLAG_DOCUMENT_8 | 0 | 0.00 |
| 2 | NAME_CONTRACT_TYPE | 0 | 0.00 |
| 3 | CODE_GENDER | 0 | 0.00 |
| 4 | FLAG_OWN_CAR | 0 | 0.00 |
| 95 | DAYS_LAST_PHONE_CHANGE | 1 | 0.00 |
| 96 | FLAG_DOCUMENT_2 | 0 | 0.00 |
| 97 | FLAG_DOCUMENT_3 | 0 | 0.00 |
| 98 | FLAG_DOCUMENT_4 | 0 | 0.00 |
| 99 | FLAG_DOCUMENT_5 | 0 | 0.00 |
| 100 | FLAG_DOCUMENT_6 | 0 | 0.00 |

Plotting the heatmap to check the visualization of NaN values in columns.




Removing the NaN values columns whose columns are having NaN values Greater than 40% in it.

Click here to ask Blackbox to help you code faster

```python
# displaying all the collumn who are having more than 40% NaN values in it.
null_value_gre40 = null_percentage[null_percentage > 40].index
null_value_gre40
```

0.0s

```
Index(['OWN_CAR_AGE', 'EXT_SOURCE_1', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG',
       'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BUILD_AVG', 'COMMONAREA_AVG',
       'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_AVG',
       'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG',
       'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG', 'APARTMENTS_MODE',
       'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE',
       'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE',
       'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE',
       'LIVINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE',
       'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI',
       'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI',
       'ENTRANCES_MEDI', 'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI',
       'LIVINGAPARTMENTS_MEDI', 'LIVINGAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI',
       'NONLIVINGAREA_MEDI', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE',
       'TOTALAREA_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE'],
      dtype='object')
```

Also dropping all the Flag Document Columns from the dataset.

Click here to ask Blackbox to help you code faster

```python
# Function to see all the documents columns in one list and to remove them.

def col_name(data):
    columns_to_drop = []
    for column in data.columns:
        if "DOCUMENT" in column:
            columns_to_drop.append(column)

    return columns_to_drop


# Applying the function to the dataset
document_col = col_name(appli_data)
```

0.0s

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

Again Checking all the FlAG Column from the dataset:

```
Click here to ask Blackbox to help you code faster

def flag_name(data):
    # This function takes a DataFrame 'data' as input and identifies columns containing the substring "FLAG".
    # It then returns a list of such columns.
    columns_to_drop = []
    for column in data.columns:
        if "FLAG" in column:
            columns_to_drop.append(column)

    return columns_to_drop

# Call the function with a DataFrame named 'appli_data' to get a list of columns containing "FLAG"
fla = flag_name(appli_data)
print(fla)
print("\n")

# Create a dictionary to store unique values for each identified "FLAG" column
unique_value = {}
for column in fla:
    # For each "FLAG" column, store its unique values in the dictionary
    unique_value[column] = appli_data[column].unique()

# Print the dictionary containing unique values for "FLAG" columns
unique_value


✓ 0.0s
```

```
['FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL']


{'FLAG_OWN_CAR': array(['N', 'Y'], dtype=object),
 'FLAG_OWN_REALTY': array(['Y', 'N'], dtype=object),
 'FLAG_MOBIL': array([1, 0], dtype=int64),
 'FLAG_EMP_PHONE': array([1, 0], dtype=int64),
 'FLAG_WORK_PHONE': array([0, 1], dtype=int64),
 'FLAG_CONT_MOBILE': array([1, 0], dtype=int64),
 'FLAG_PHONE': array([1, 0], dtype=int64),
 'FLAG_EMAIL': array([0, 1], dtype=int64)}
```

Now we dropped all the FLAG Document and Remaining FLAG Columns from the dataset Except FLAG_OWN_CAR and FLAG_OWN_REALTY.

Now we will check all the Unique values of all the object columns to count the frequency of each columns and fill the Nan values as well as XNA and XAP values.

```
Click here to ask Blackbox to help you code faster
# Get the columns with data type 'object'
object_column = appli_data.select_dtypes(include = ['object']).columns

# Create a dictionary to store unique values for each object column
unique_values_dict = {}

# Iterate through each object column and store its unique values.
for column in object_column:
    unique_values_dict[column] = appli_data[column].unique()

for column, values in unique_values_dict.items():
    print(f"Column: {column}")
    print(f"unique_value: {values}")
    print()
✓ 0.0s
```

```
Column: NAME_CONTRACT_TYPE
unique_value: ['Cash loans' 'Revolving loans']

Column: CODE_GENDER
unique_value: ['M' 'F' 'XNA']

Column: FLAG_OWN_CAR
unique_value: ['N' 'Y']

Column: FLAG_OWN_REALTY
unique_value: ['Y' 'N']

Column: NAME_TYPE_SUITE
unique_value: ['Unaccompanied' 'Family' 'Spouse, partner' 'Children' 'Other_A' nan
 'Other_B' 'Group of people']

Column: NAME_INCOME_TYPE
unique_value: ['Working' 'State servant' 'Commercial associate' 'Pensioner' 'Unemployed'
 'Student' 'Businessman' 'Maternity leave']

Column: NAME_EDUCATION_TYPE
unique_value: ['Secondary / secondary special' 'Higher education' 'Incomplete higher'
 'Lower secondary' 'Academic degree']

Column: NAME_FAMILY_STATUS
unique_value: ['Single / not married' 'Married' 'Civil marriage' 'Widow' 'Separated'
 'Unknown']

Column: NAME_HOUSING_TYPE
unique_value: ['House / apartment' 'Rented apartment' 'With parents'
 'Municipal apartment' 'Office apartment' 'Co-op apartment']
```

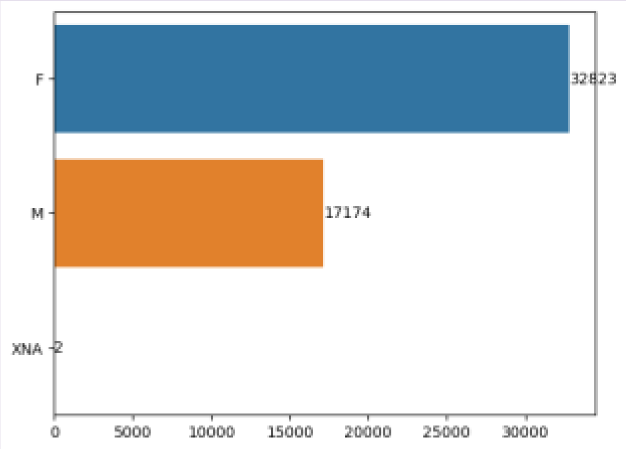
```
Column: OCCUPATION_TYPE
unique_value: ['Laborers' 'Core staff' 'Accountants' 'Managers' nan 'Drivers'
 'Sales staff' 'Cleaning staff' 'Cooking staff' 'Private service staff'
 'Medicine staff' 'Security staff' 'High skill tech staff'
 'Waiters/barmen staff' 'Low-skill Laborers' 'Realty agents' 'Secretaries'
 'IT staff' 'HR staff']

Column: WEEKDAY_APPR_PROCESS_START
unique_value: ['WEDNESDAY' 'MONDAY' 'THURSDAY' 'SUNDAY' 'SATURDAY' 'FRIDAY' 'TUESDAY']

Column: ORGANIZATION_TYPE
unique_value: ['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']
```

Replacing all the XNA and XAP values of all the columns.'

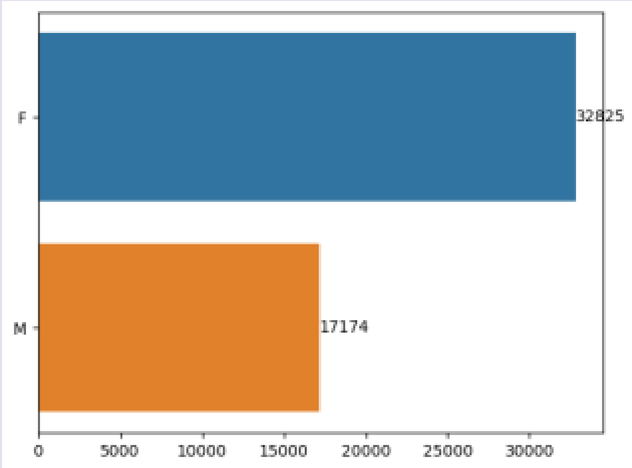
1. CODE_GENDER

```python
Click here to ask Blackbox to help you code faster
# Calculate the count of each unique value in the 'CODE_GENDER' column and store it in 'gender_count'
gender_count = appli_data.CODE_GENDER.value_counts()
print("Before Replacing XNA values: ")
fig = sns.barplot(y = gender_count.index, x = gender_count.values, estimator = 'sum', errorbar = None)
fig.bar_label(fig.containers[0], fontsize=10)
plt.show()

# Replacing the 'XNA' values with the most frequent gender (top most count) in the 'CODE_GENDER' column
# The assumption here is that the most frequent gender is a reasonable replacement for 'XNA'
appli_data['CODE_GENDER'] = appli_data['CODE_GENDER'].replace("XNA", "F")
gen = appli_data.CODE_GENDER.value_counts()

print("\nAfter Replacing the values: ")
fig = sns.barplot(y = gen.index, x = gen.values, estimator = 'sum', errorbar = None)
fig.bar_label(fig.containers[0], fontsize=10)
plt.show()
```

Before Replacing XNA values:




After Replacing the values:




2. ORGANIZATION_TYPE

Click here to ask Blackbox to help you code faster

```python
# Checking the Orginization type counts
org_count = appli_data.ORGANIZATION_TYPE.value_counts()
print(org_count)

# Count of 'XNA' values
xna_count = org_count['XNA']

# Values to fill evenly for 'XNA'
fill_values = [
    'Business Entity Type 3',
    'Self-employed',
    'Other',
    'Medicine',
    'Government',
    'Business Entity Type 2',
    'School',
    'Trade: type 7',
    'Kindergarten'
]

# Create a list of values to replace 'XNA'
replace_values = np.tile(fill_values, int(np.ceil(xna_count / len(fill_values))))

# Trim the list to match the number of 'XNA' values
replace_values = replace_values[:xna_count]

# Replace 'XNA' values with the calculated list of values
appli_data.loc[appli_data['ORGANIZATION_TYPE'] == 'XNA', 'ORGANIZATION_TYPE'] = replace_values


# Checking the updated count
org_count_update = appli_data['ORGANIZATION_TYPE'].value_counts()
print("\n##########################################\n")
print(org_count_update)
```

-→ Handling the NaN values:

```
Click here to ask Blackbox to help you code faster
nan_col = {}
for column in object_column:
    nan_col[column] = appli_data[column].isnull().sum()

nan_col
```

0.0s

```
{'NAME_CONTRACT_TYPE': 0,
 'CODE_GENDER': 0,
 'FLAG_OWN_CAR': 0,
 'FLAG_OWN_REALTY': 0,
 'NAME_TYPE_SUITE': 192,
 'NAME_INCOME_TYPE': 0,
 'NAME_EDUCATION_TYPE': 0,
 'NAME_FAMILY_STATUS': 0,
 'NAME_HOUSING_TYPE': 0,
 'OCCUPATION_TYPE': 15654,
 'WEEKDAY_APPR_PROCESS_START': 0,
 'ORGANIZATION_TYPE': 0}
```

```
Click here to ask Blackbox to help you code faster
# Calculate and print the count of each unique value in the 'NAME_TYPE_SUITE' column
print(appli_data['NAME_TYPE_SUITE'].value_counts(),"\n")

# Fill missing (NaN) values in the 'NAME_TYPE_SUITE' column with the most frequent value (mode)
# The mode is accessed using .mode()[0]
appli_data['NAME_TYPE_SUITE'].fillna(appli_data['NAME_TYPE_SUITE'].mode()[0], inplace=True)

# Check and print the updated count of missing (NaN) values in the 'NAME_TYPE_SUITE' column
nan_count_after_fill = appli_data['NAME_TYPE_SUITE'].isna().sum()
print(nan_count_after_fill)
```

0.0s

```
Unaccompanied      40435
Family              6549
Spouse, partner     1849
Children             542
Other_B              259
Other_A              137
Group of people       36
Name: NAME_TYPE_SUITE, dtype: int64

0
```

```
Click here to ask Blackbox to help you code faster
# Print the count of NaN values in the 'OCCUPATION_TYPE' column
print(f'There are {appli_data.OCCUPATION_TYPE.isnull().sum()} NaN values present in the `Occupation_type`.')

# Print the number of unique non-NaN values in the 'OCCUPATION_TYPE' column
print(f"The unique values present in the `Occupational_type` is: \n{appli_data.OCCUPATION_TYPE.value_counts()}.")

# Get the top 12 most frequent non-NaN values in the 'OCCUPATION_TYPE' column
top_12_frequency = appli_data.OCCUPATION_TYPE.value_counts(ascending=False).nlargest(12).index

# Create a boolean index for NaN values in the 'OCCUPATION_TYPE' column
nan_index = appli_data['OCCUPATION_TYPE'].isna()

# Replace NaN values with a random choice from the top 12 most frequent non-NaN values
appli_data.loc[nan_index, 'OCCUPATION_TYPE'] = np.random.choice(top_12_frequency, nan_index.sum())

# Print the updated count of NaN values in the 'OCCUPATION_TYPE' column
print(f"There are {appli_data['OCCUPATION_TYPE'].isna().sum()} NaN values present in the `Occuption_type`.")
```

0.0s

```
There are 15654 NaN values present in the `Occupation_type`.
The unique values present in the `Occupational_type` is:
Laborers                 8952
Sales staff              5160
Core staff               4434
Managers                 3489
Drivers                  3044
High skill tech staff    1852
Accountants              1621
Medicine staff           1403
Security staff           1140
Cooking staff             963
Cleaning staff            739
Private service staff     447
Low-skill Laborers        357
Waiters/barmen staff      228
Secretaries               212
Realty agents             123
HR staff                  101
IT staff                   80
Name: OCCUPATION_TYPE, dtype: int64.
There are 0 NaN values present in the `Occuption_type`.
```

Checking all the remaining NaN values columns with the dtypes and percentage.

```
Top 16 columns with highest missing values and their data types:
```

| | Missing Values | Data Types | Nan_values_percentage |
|---|---|---|---|
| EXT_SOURCE_3 | 9944 | float64 | 19.888398 |
| AMT_REQ_CREDIT_BUREAU_YEAR | 6734 | float64 | 13.468269 |
| AMT_REQ_CREDIT_BUREAU_QRT | 6734 | float64 | 13.468269 |
| AMT_REQ_CREDIT_BUREAU_MON | 6734 | float64 | 13.468269 |
| AMT_REQ_CREDIT_BUREAU_WEEK | 6734 | float64 | 13.468269 |
| AMT_REQ_CREDIT_BUREAU_DAY | 6734 | float64 | 13.468269 |
| AMT_REQ_CREDIT_BUREAU_HOUR | 6734 | float64 | 13.468269 |
| OBS_60_CNT_SOCIAL_CIRCLE | 168 | float64 | 0.336007 |
| OBS_30_CNT_SOCIAL_CIRCLE | 168 | float64 | 0.336007 |
| DEF_30_CNT_SOCIAL_CIRCLE | 168 | float64 | 0.336007 |
| DEF_60_CNT_SOCIAL_CIRCLE | 168 | float64 | 0.336007 |
| EXT_SOURCE_2 | 126 | float64 | 0.252005 |
| AMT_GOODS_PRICE | 38 | float64 | 0.076002 |
| AMT_ANNUITY | 1 | float64 | 0.002000 |
| DAYS_LAST_PHONE_CHANGE | 1 | float64 | 0.002000 |
| CNT_FAM_MEMBERS | 1 | float64 | 0.002000 |

Replacing the NaN values.

```python
# Print the unique values present in the 'CNT_FAM_MEMBERS' column to check the NaN values
print(f"The unique values present in the 'CNT_FAM_MEMBERS' column are {appli_data.CNT_FAM_MEMBERS.unique()}")

# Calculate and print the count of each unique value in the 'CNT_FAM_MEMBERS' column
value_counts = appli_data.CNT_FAM_MEMBERS.value_counts()
print("────────────────────────────────────────────────────────────")
print(value_counts)

# Fill missing (NaN) values in the 'CNT_FAM_MEMBERS' column with the most frequent value (mode)
# The mode is accessed using .value_counts().index[0]
appli_data['CNT_FAM_MEMBERS'].fillna(value_counts.index[0], inplace=True)

# Check and print the updated count of missing (NaN) values in the 'CNT_FAM_MEMBERS' column
nan_count_after_fill = appli_data.CNT_FAM_MEMBERS.isnull().sum()
print("────────────────────────────────────────────────────────────")
print(f"There are {nan_count_after_fill} values present in the 'CNT_FAM_MEMBERS'.")
print("────────────────────────────────────────────────────────────")
print(appli_data.CNT_FAM_MEMBERS.unique())
```

```
The unique values present in the 'CNT_FAM_MEMBERS' column are [ 1.  2.  3.  4.  5.  6.  9.  7.  8. 10. 13. nan]
────────────────────────────────────────────────────────────
2.0     25807
1.0     10873
3.0      8635
4.0      4000
5.0       592
6.0        68
7.0        12
8.0         6
9.0         2
10.0        2
13.0        1
Name: CNT_FAM_MEMBERS, dtype: int64
────────────────────────────────────────────────────────────
There are 0 values present in the 'CNT_FAM_MEMBERS'.
────────────────────────────────────────────────────────────
[ 1.  2.  3.  4.  5.  6.  9.  7.  8. 10. 13.]
```

Click here to ask Blackbox to help you code faster

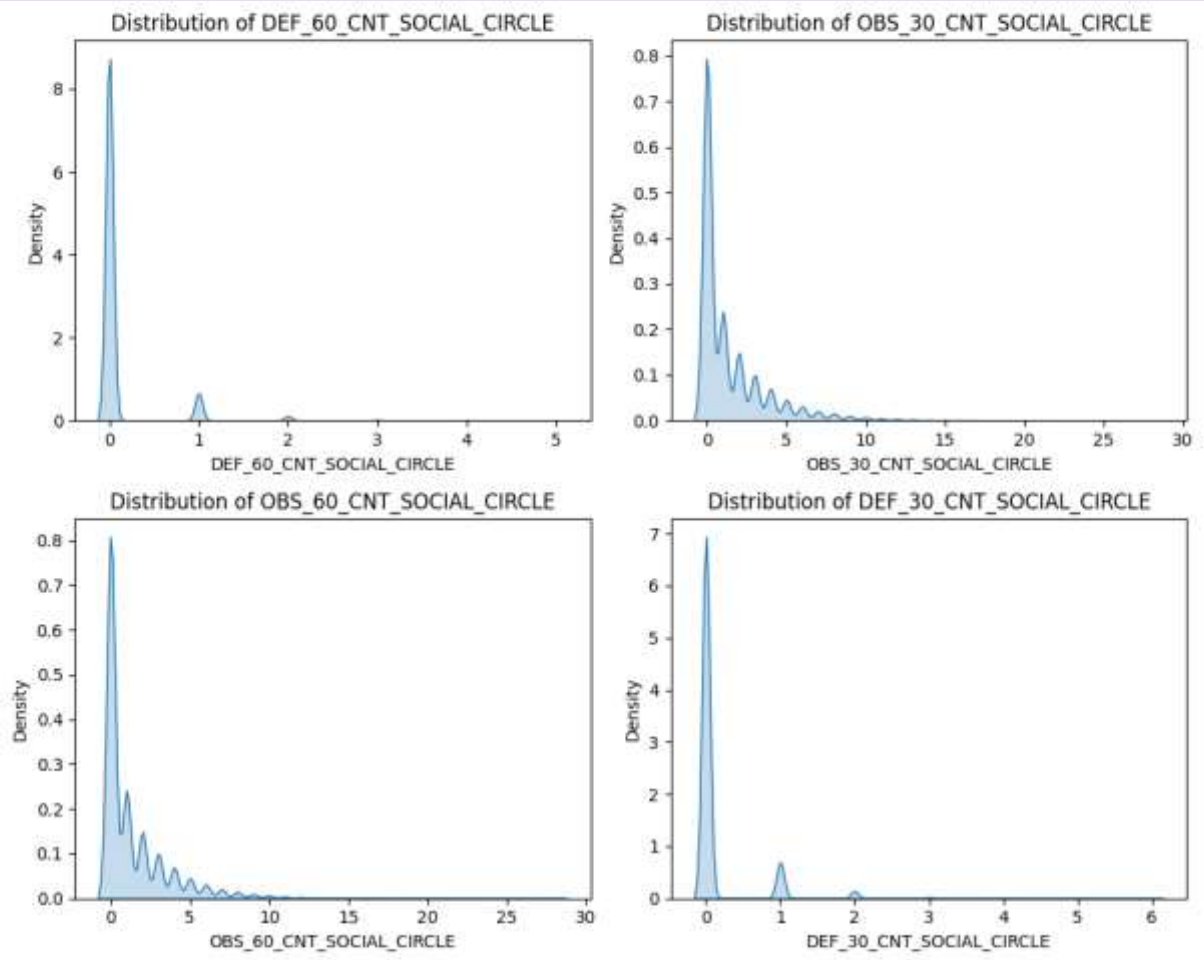
```python
round(appli_data[['AMT_ANNUITY','DAYS_LAST_PHONE_CHANGE','AMT_GOODS_PRICE']].describe(),2)
```

✓ 0.0s

| | AMT_ANNUITY | DAYS_LAST_PHONE_CHANGE | AMT_GOODS_PRICE |
|---|---|---|---|
| count | 49998.00 | 49998.00 | 49961.00 |
| mean | 27107.38 | -964.30 | 539060.04 |
| std | 14562.94 | 829.49 | 369853.25 |
| min | 2052.00 | -4002.00 | 45000.00 |
| 25% | 16456.50 | -1573.00 | 238500.00 |
| 50% | 24939.00 | -755.00 | 450000.00 |
| 75% | 34596.00 | -270.00 | 679500.00 |
| max | 258025.50 | 0.00 | 4050000.00 |

```python
# Fill missing values in the 'AMT_ANNUITY' column with the median value of the column
appli_data['AMT_ANNUITY'].fillna(appli_data['AMT_ANNUITY'].median(), inplace = True)

# Fill missing values in the 'DAYS_LAST_PHONE_CHANGE' column with the mode (most frequent value) of the column
appli_data['DAYS_LAST_PHONE_CHANGE'].fillna(appli_data['DAYS_LAST_PHONE_CHANGE'].mode()[0], inplace = True)

# Fill missing values in the 'AMT_GOODS_PRICE' column with the median value of the column
appli_data['AMT_GOODS_PRICE'].fillna(appli_data['AMT_GOODS_PRICE'].median(), inplace = True)
```

```python
# List of columns to iterate through
columns_to_check = ['DEF_60_CNT_SOCIAL_CIRCLE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE']

# Set up the subplots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))

# Flatten the axes to make it easier to index
axes = axes.flatten()

# Iterate through each specified column in the list 'columns_to_check'
for i, column in enumerate(columns_to_check):
    # Fill missing (NaN) values in the current column with its mode (most frequent value)
    appli_data[column].fillna(appli_data[column].mode()[0], inplace=True)

    # Plot the distribution using Seaborn's distplot
    sns.kdeplot(appli_data[column], ax=axes[i], shade = True)
    axes[i].set_title(f'Distribution of {column}')

# Adjust layout for better visualization
plt.tight_layout()
plt.show()
```



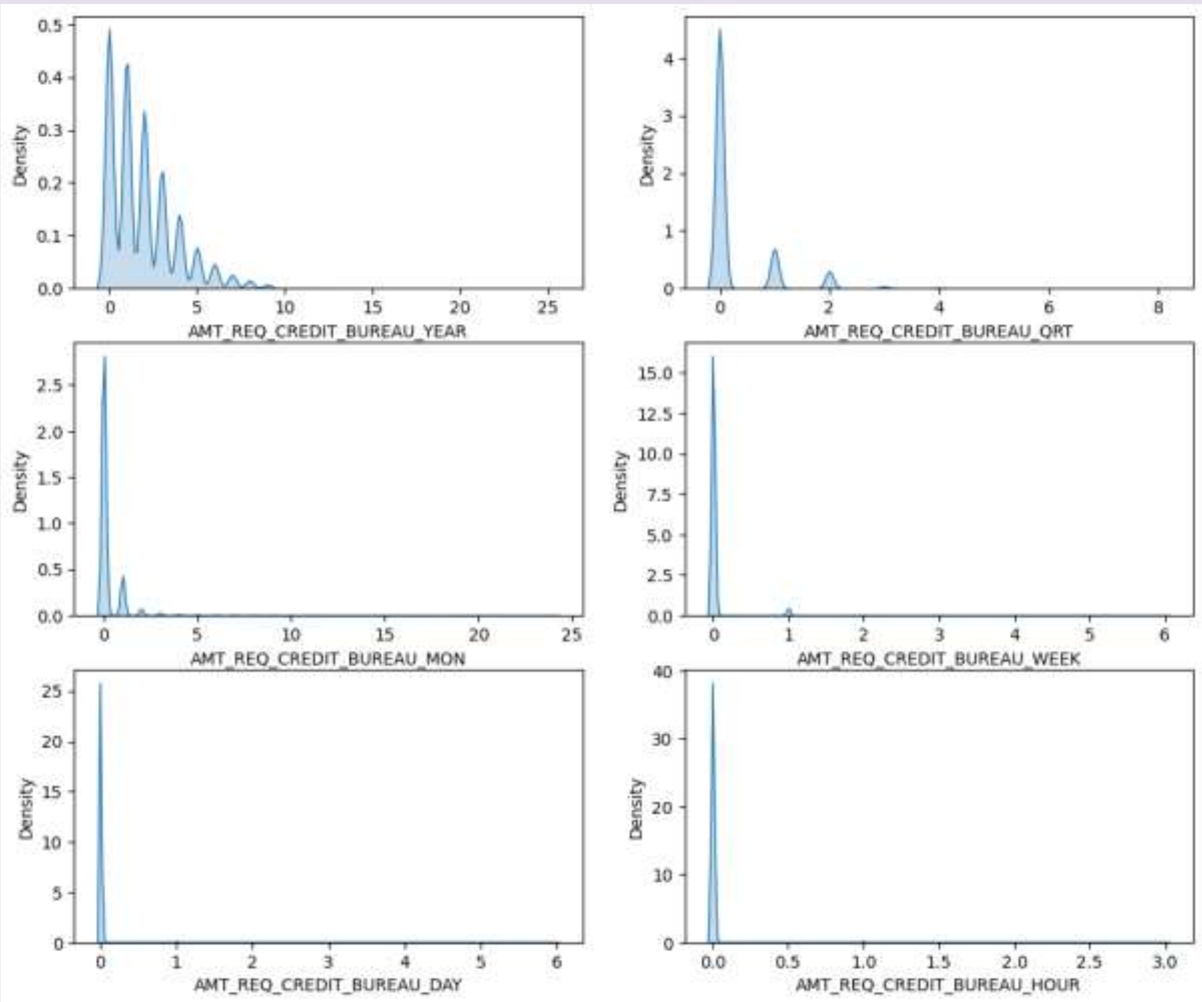
💡 Click here to ask Blackbox to help you code faster

```python
# Iterate through each column and print its value counts
for column in columns_to_check:
    print(appli_data[column].value_counts())
    print("------------------------------------------------")

# Iterate through each specified column in the list 'columns_to_check'
for column in columns_to_check:

    # Fill missing (NaN) values in the current column with its mode (most frequent value)
    appli_data[column].fillna(appli_data[column].mode()[0], inplace=True)

    # Print the count of remaining NaN values after filling for the current column
    print(f'There are {appli_data[column].isna().sum()} NaN values present in {column}.')
    print("------------------------------------------------")

```

✓ 0.0s

💡 Click here to ask Blackbox to help you code faster

```python
plt.figure(figsize=(12, 10))

plt.subplot(3, 2, 1)
sns.kdeplot(appli_data['AMT_REQ_CREDIT_BUREAU_YEAR'], shade = True)

plt.subplot(3, 2, 2)
sns.kdeplot(appli_data['AMT_REQ_CREDIT_BUREAU_QRT'], shade = True)

plt.subplot(3, 2, 3)
sns.kdeplot(appli_data['AMT_REQ_CREDIT_BUREAU_MON'], shade = True)

plt.subplot(3, 2, 4)
sns.kdeplot(appli_data['AMT_REQ_CREDIT_BUREAU_WEEK'], shade = True)

plt.subplot(3, 2, 5)
sns.kdeplot(appli_data['AMT_REQ_CREDIT_BUREAU_DAY'], shade = True)

plt.subplot(3, 2, 6)
sns.kdeplot(appli_data['AMT_REQ_CREDIT_BUREAU_HOUR'], shade = True)

plt.show()
```



```python
💡 Click here to ask Blackbox to help you code faster
col = ['AMT_REQ_CREDIT_BUREAU_YEAR', 'AMT_REQ_CREDIT_BUREAU_QRT',
'AMT_REQ_CREDIT_BUREAU_MON','AMT_REQ_CREDIT_BUREAU_WEEK',
'AMT_REQ_CREDIT_BUREAU_DAY','AMT_REQ_CREDIT_BUREAU_HOUR']

# Iterate through each column and print its value counts amd filling NaN values.
for column in col:
    print(appli_data[column].value_counts())
    print("\n")
    appli_data[column].fillna(appli_data[column].mode()[0], inplace = True)
    print(f'There are {appli_data[column].isna().sum()} NaN values present in {column}.')
```

-→ Previous Application:

The previous application dataset contains 49999 rows and 37 columns.

Checking the NaN values in the form of percentage to remove the columns who are having the NaN values greater than 35%.

| | column_name | null_count | null_percentage |
|---|---|---|---|
| 14 | RATE_INTEREST_PRIVILEGED | 49834 | 99.67 |
| 13 | RATE_INTEREST_PRIMARY | 49834 | 99.67 |
| 12 | RATE_DOWN_PAYMENT | 25198 | 50.40 |
| 6 | AMT_DOWN_PAYMENT | 25198 | 50.40 |
| 20 | NAME_TYPE_SUITE | 24243 | 48.49 |
| 36 | NFLAG_INSURED_ON_APPROVAL | 19160 | 38.32 |
| 31 | DAYS_FIRST_DRAWING | 19160 | 38.32 |
| 32 | DAYS_FIRST_DUE | 19160 | 38.32 |
| 33 | DAYS_LAST_DUE_1ST_VERSION | 19160 | 38.32 |
| 34 | DAYS_LAST_DUE | 19160 | 38.32 |
| 35 | DAYS_TERMINATION | 19160 | 38.32 |
| 7 | AMT_GOODS_PRICE | 10744 | 21.49 |
| 3 | AMT_ANNUITY | 10592 | 21.18 |
| 28 | CNT_PAYMENT | 10592 | 21.18 |
| 30 | PRODUCT_COMBINATION | 8 | 0.02 |
| 25 | CHANNEL_TYPE | 0 | 0.00 |
| 24 | NAME_PRODUCT_TYPE | 0 | 0.00 |
| 29 | NAME_YIELD_GROUP | 0 | 0.00 |
| 26 | SELLERPLACE_AREA | 0 | 0.00 |
| 27 | NAME_SELLER_INDUSTRY | 0 | 0.00 |
| 22 | NAME_GOODS_CATEGORY | 0 | 0.00 |
| 23 | NAME_PORTFOLIO | 0 | 0.00 |
| 0 | SK_ID_PREV | 0 | 0.00 |
| 21 | NAME_CLIENT_TYPE | 0 | 0.00 |
| 19 | CODE_REJECT_REASON | 0 | 0.00 |
| 1 | SK_ID_CURR | 0 | 0.00 |


```python
# displaying all the collumn who are having more than 40% NaN values in it.
null_value_gre40 = null_percentage[null_percentage > 35].index
null_value_gre40
```

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

```python
col_drop = ['NFLAG_LAST_APPL_IN_DAY', 'FLAG_LAST_APPL_PER_CONTRACT', 'HOUR_APPR_PROCESS_START', 'WEEKDAY_APPR_PROCESS_START']

previ_data.drop(columns = col_drop, inplace = True)
```

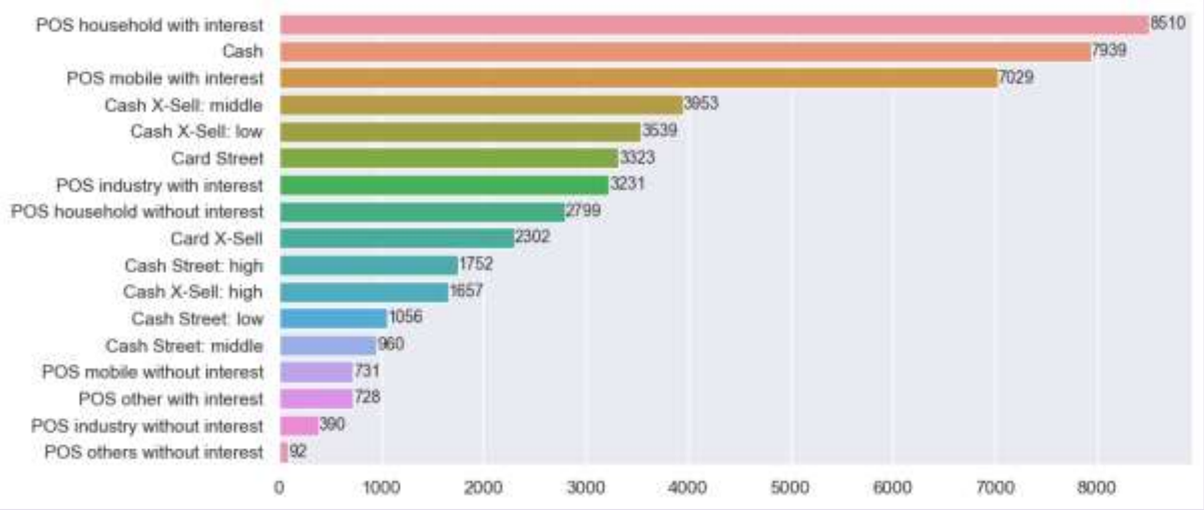
```python
# Display the count of missing values for each column, sorted in descending order,
# and show the top 10 columns with the highest missing values
missing_values_top10 = previ_data.isna().sum().sort_values(ascending=False).nlargest(10)

# Print the results
print(missing_values_top10)
```

```
AMT_GOODS_PRICE         10744
AMT_ANNUITY             10592
CNT_PAYMENT             10592
PRODUCT_COMBINATION         8
NAME_CLIENT_TYPE            0
NAME_YIELD_GROUP            0
NAME_SELLER_INDUSTRY        0
SELLERPLACE_AREA            0
CHANNEL_TYPE                0
NAME_PRODUCT_TYPE           0
dtype: int64
```

➔ Filling NaN Values with top frequency:

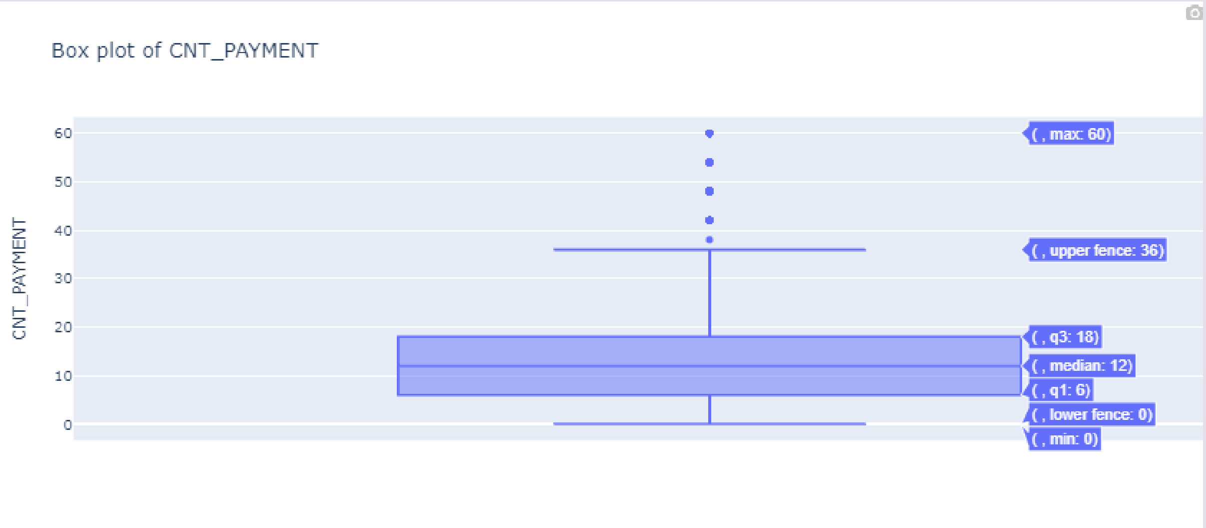
Click here to ask Blackbox to help you code faster

```python
# Count the occurrences of each unique value in the 'PRODUCT_COMBINATION' column
val_count = previ_data.PRODUCT_COMBINATION.value_counts()

# Create a bar plot to visualize the counts of each product combination
plt.figure(figsize=(10, 5))
fig = sns.barplot(x=val_count.values, y=val_count.index, estimator='sum', errorbar=None)

# Add labels to the bars indicating the sum of counts
fig.bar_label(fig.containers[0], fontsize=10)

# Display the bar plot
plt.show()

# Fill missing values in the 'PRODUCT_COMBINATION' column with a specified default value
previ_data.PRODUCT_COMBINATION.fillna("POS household without interest", inplace=True)
```



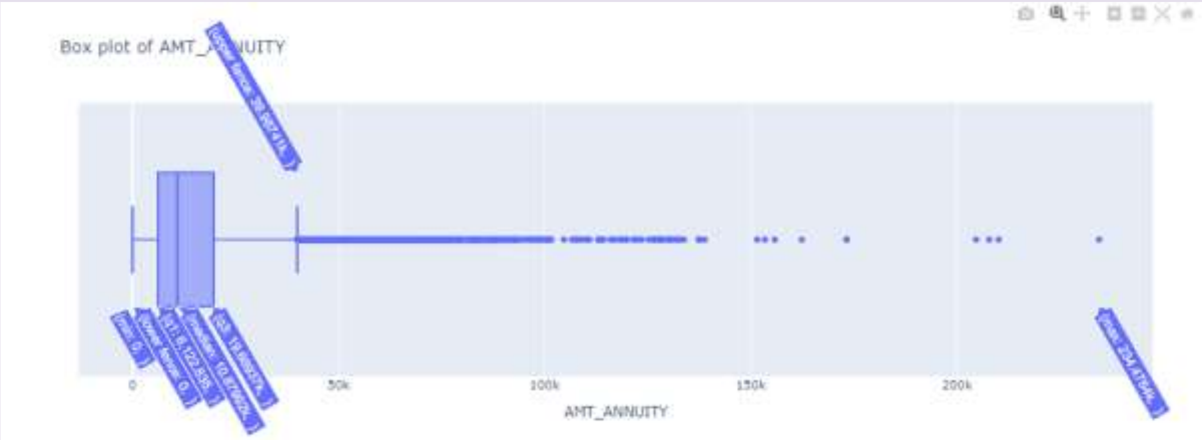

Click here to ask Blackbox to help you code faster

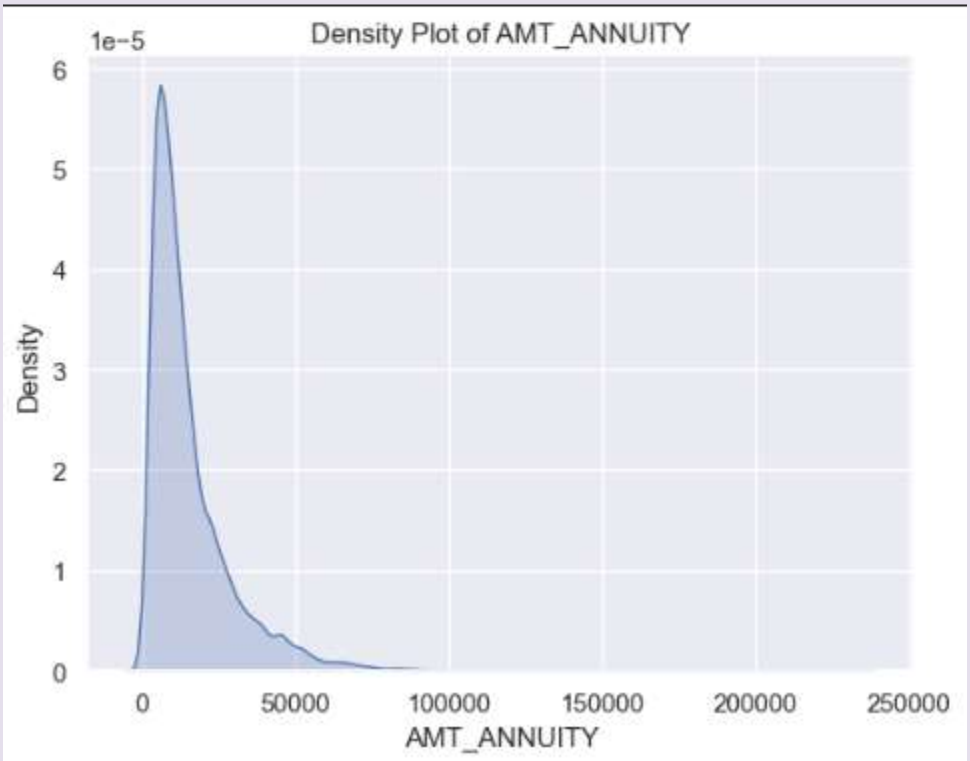
```python
# Create a box plot to visualize the distribution of 'CNT_PAYMENT' column values
fig = px.box(previ_data, y = 'CNT_PAYMENT', title = "Box plot of CNT_PAYMENT")
fig.show()

# Fill missing values in the 'CNT_PAYMENT' column with the median value of the column
previ_data['CNT_PAYMENT'] = previ_data["CNT_PAYMENT"].fillna(previ_data["CNT_PAYMENT"].median())
```



Click here to ask Blackbox to help you code faster

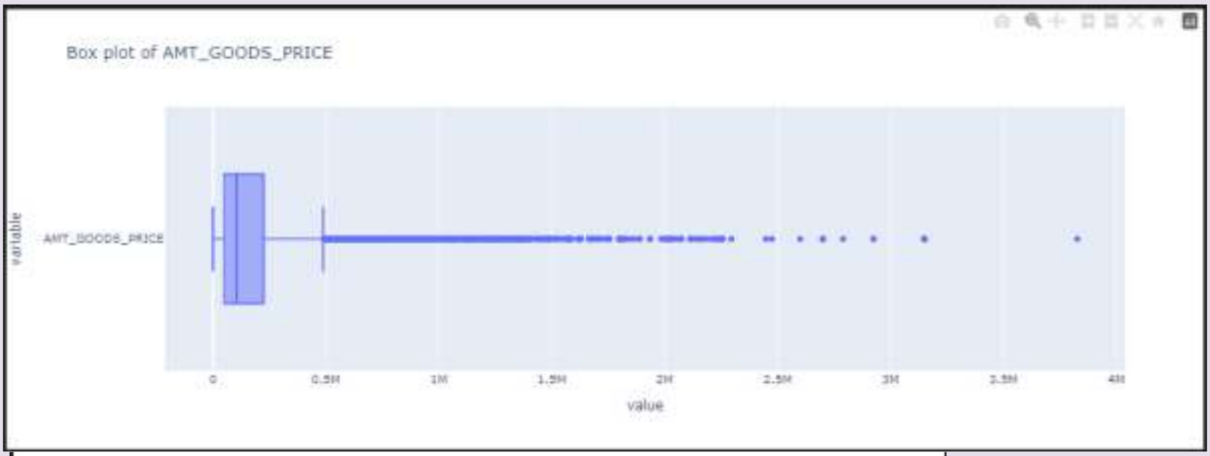
```python
# Create a box plot using Plotly Express for the 'AMT_ANNUITY' column in 'previ_data'
fig = px.box(previ_data, x="AMT_ANNUITY", title="Box plot of AMT_ANNUITY")
fig.show()

# Create a Kernel Density Estimate (KDE) plot using Seaborn for the 'AMT_ANNUITY' column
sns.kdeplot(previ_data['AMT_ANNUITY'], shade=True)

# Add a title to the KDE plot
plt.title("Density Plot of AMT_ANNUITY")
plt.show()

# Fill missing values in the 'AMT_ANNUITY' column with the median value
previ_data['AMT_ANNUITY'].fillna(previ_data['AMT_ANNUITY'].median(), inplace=True)
```

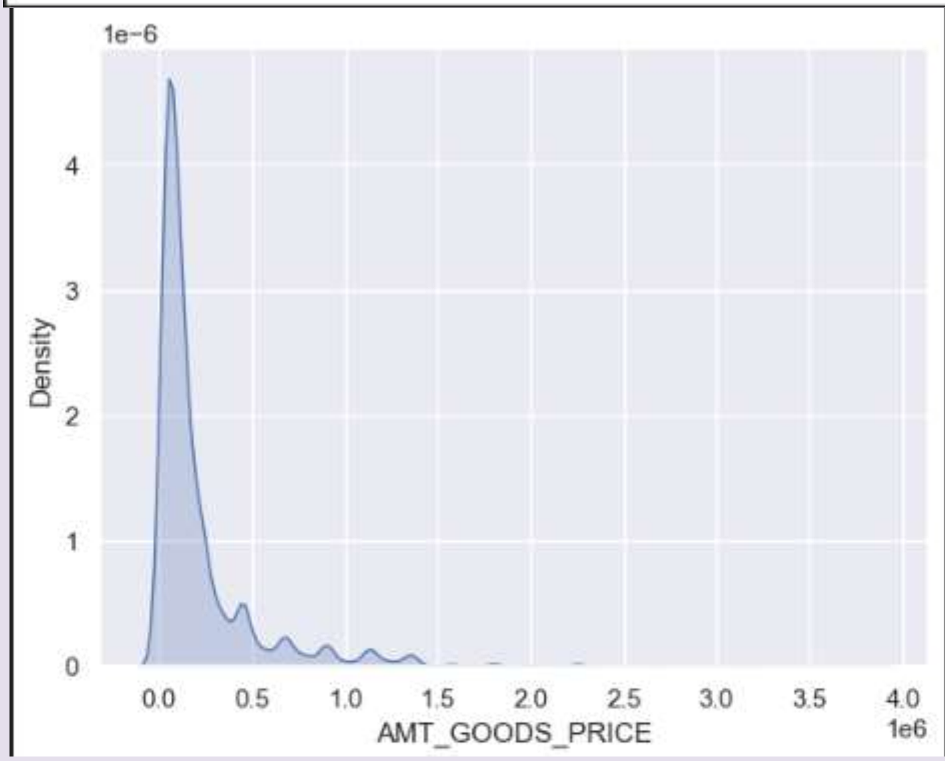


Click here to ask Blackbox to help you code faster

```python
# Create a box plot using Plotly Express for the 'AMT_GOODS_PRICE' column in 'previ_data'
fig = px.box(previ_data, x=['AMT_GOODS_PRICE'], title="Box plot of AMT_GOODS_PRICE")
fig.show()

# Create a Kernel Density Estimate (KDE) plot using Seaborn for the 'AMT_GOODS_PRICE' column
sns.kdeplot(previ_data['AMT_GOODS_PRICE'], shade=True)

# Display the plots
plt.show()

# Fill missing values in the 'AMT_GOODS_PRICE' column with the median value
previ_data['AMT_GOODS_PRICE'].fillna(previ_data['AMT_GOODS_PRICE'].median(), inplace=True)
```



➔ Checking all the colums who are having XNA and XAP in it to fill with top counts.



```python
# List of column with XNA and XAP as unique values.
columns_with_xna_xap = []

# Iterating through each column in dataframe
for column in previ_data.columns:
    # Checking if XNA and XAP is present in the unique values of the column
    if "XNA" in previ_data[column].unique() or "XAP" in previ_data[column].unique():
        columns_with_xna_xap.append(column)
```

```
['NAME_CONTRACT_TYPE',
 'NAME_CASH_LOAN_PURPOSE',
 'NAME_PAYMENT_TYPE',
 'CODE_REJECT_REASON',
 'NAME_CLIENT_TYPE',
 'NAME_GOODS_CATEGORY',
 'NAME_PORTFOLIO',
 'NAME_PRODUCT_TYPE',
 'NAME_SELLER_INDUSTRY',
 'NAME_YIELD_GROUP']
```



```python
# Replace occurrences of "XNA" in the 'NAME_CONTRACT_TYPE' column with the mode (most frequent value)
# Mode is used as a replacement for "XNA" to handle missing or undefined values in a categorical column
previ_data['NAME_CONTRACT_TYPE'].replace("XNA",previ_data['NAME_CONTRACT_TYPE'].mode()[0], inplace = True)

# Replace occurrences of "XNA" in the 'NAME_PAYMENT_TYPE' column with the mode (most frequent value)
# Mode is used as a replacement for "XNA" to handle missing or undefined values in a categorical column
previ_data['NAME_PAYMENT_TYPE'].replace("XNA", previ_data['NAME_PAYMENT_TYPE'].mode()[0], inplace=True)

# Replace occurrences of "XNA" in the 'NAME_CLIENT_TYPE' column with the mode (most frequent value)
# Mode is used as a replacement for "XNA" to handle missing or undefined values in a categorical column
previ_data['NAME_CLIENT_TYPE'].replace("XNA", previ_data['NAME_CLIENT_TYPE'].mode()[0], inplace=True)

# Replace occurrences of "XNA" in the 'NAME_PORTFOLIO' column with the mode (most frequent value)
# Mode is used as a replacement for "XNA" to handle missing or undefined values in a categorical column
previ_data['NAME_PORTFOLIO'].replace("XNA", previ_data['NAME_PORTFOLIO'].mode()[0], inplace=True)

# Replace occurrences of "XNA" in the '"NAME_PRODUCT_TYPE"' column with the mode (most frequent value)
# Mode is used as a replacement for "XNA" to handle missing or undefined values in a categorical column
previ_data["NAME_PRODUCT_TYPE"].replace("XNA", "x-sell", inplace=True)

# Replace occurrences of "XNA" in the 'NAME_YIELD_GROUP' column with the mode (most frequent value)
# Mode is used as a replacement for "XNA" to handle missing or undefined values in a categorical column
previ_data['NAME_YIELD_GROUP'].replace("XNA", 'middle', inplace=True)
```

```python
# Display the counts of unique values in the 'NAME_CASH_LOAN_PURPOSE' column before modification
# Get the values excluding the top two
print(f"'NAME_CASH_LOAN_PURPOSE' column before modification:\n{previ_data['NAME_CASH_LOAN_PURPOSE'].value_counts()}")
print("————————————————————————————————————————")

# Identify the 'XNA' values in the 'NAME_CASH_LOAN_PURPOSE' column
xna_values = previ_data['NAME_CASH_LOAN_PURPOSE'] == 'XNA'

# Divide 'XNA' values into four equal parts
division_size = len(previ_data[xna_values]) // 3
indices = np.where(xna_values)[0]  # Get the indices of 'XNA' values

# Assign parts to 'Repairs', 'Other', 'Urgent needs ' respectively
previ_data.loc[indices[:division_size], 'NAME_CASH_LOAN_PURPOSE'] = 'Repairs'
previ_data.loc[indices[division_size:2*division_size], 'NAME_CASH_LOAN_PURPOSE'] = 'Other'
previ_data.loc[indices[2*division_size:3*division_size], 'NAME_CASH_LOAN_PURPOSE'] = 'Urgent needs'

# Identify the 'XAP' values in the 'NAME_CASH_LOAN_PURPOSE' column
xap_values = previ_data['NAME_CASH_LOAN_PURPOSE'] == 'XAP'

# Divide 'XAP' values into four equal parts
division_size = len(previ_data[xap_values]) // 8
indices = np.where(xap_values)[0]  # Get the indices of 'XAP' values

# Assign parts to 'Repairs', 'Other', 'Urgent needs', 'Buying a used car','Building a house or an annex',
# 'Medicine', 'Payments on other loans', 'Everyday expenses', respectively
previ_data.loc[indices[:division_size], 'NAME_CASH_LOAN_PURPOSE'] = 'Repairs'
previ_data.loc[indices[division_size:2*division_size], 'NAME_CASH_LOAN_PURPOSE'] = 'Other'
previ_data.loc[indices[2*division_size:3*division_size], 'NAME_CASH_LOAN_PURPOSE'] = 'Urgent needs'
previ_data.loc[indices[3*division_size:4*division_size], 'NAME_CASH_LOAN_PURPOSE'] = 'Buying a used car'
previ_data.loc[indices[4*division_size:5*division_size], 'NAME_CASH_LOAN_PURPOSE'] = 'Building a house or an annex'
previ_data.loc[indices[5*division_size:6*division_size], 'NAME_CASH_LOAN_PURPOSE'] = 'Medicine'
previ_data.loc[indices[6*division_size:7*division_size], 'NAME_CASH_LOAN_PURPOSE'] = 'Payments on other loans'
previ_data.loc[indices[7*division_size:8*division_size], 'NAME_CASH_LOAN_PURPOSE'] = 'Everyday expenses'

# Renaming the remaining XNA and XAP to Refusal to name the goal
previ_data['NAME_CASH_LOAN_PURPOSE'].replace(['XNA', 'XAP'], "Refusal to name the goal", inplace = True)

# Display the counts of unique values in the 'NAME_CASH_LOAN_PURPOSE' column after modification
print(f"'NAME_CASH_LOAN_PURPOSE' column after modification:\n{previ_data['NAME_CASH_LOAN_PURPOSE'].value_counts()}")
```

```python
# Display the counts of unique values in the 'CODE_REJECT_REASON' column before modification
print(f"'CODE_REJECT_REASON' column before modification:\n{previ_data['CODE_REJECT_REASON'].value_counts()}")
print("————————————————————————————————————————")

# Replace 'XNA' with 'HC' in the 'CODE_REJECT_REASON' column
previ_data['CODE_REJECT_REASON'].replace('XNA', 'HC', inplace=True)

# Identify the 'XAP' values in the 'CODE_REJECT_REASON' column
xap_values = previ_data['CODE_REJECT_REASON'] == 'XAP'

# Divide 'XAP' values into four equal parts
division_size = len(previ_data[xap_values]) // 4
indices = np.where(xap_values)[0]  # Get the indices of 'XAP' values

# Assign parts to 'HC', 'LIMIT', 'SCO', 'CLIENT' respectively
previ_data.loc[indices[:division_size], 'CODE_REJECT_REASON'] = 'HC'
previ_data.loc[indices[division_size:2*division_size], 'CODE_REJECT_REASON'] = 'LIMIT'
previ_data.loc[indices[2*division_size:3*division_size], 'CODE_REJECT_REASON'] = 'SCO'
previ_data.loc[indices[3*division_size:], 'CODE_REJECT_REASON'] = 'CLIENT'

# Display the counts of unique values in the 'CODE_REJECT_REASON' column after modification
print(f"'CODE_REJECT_REASON' column after modification:\n{previ_data['CODE_REJECT_REASON'].value_counts()}")
```

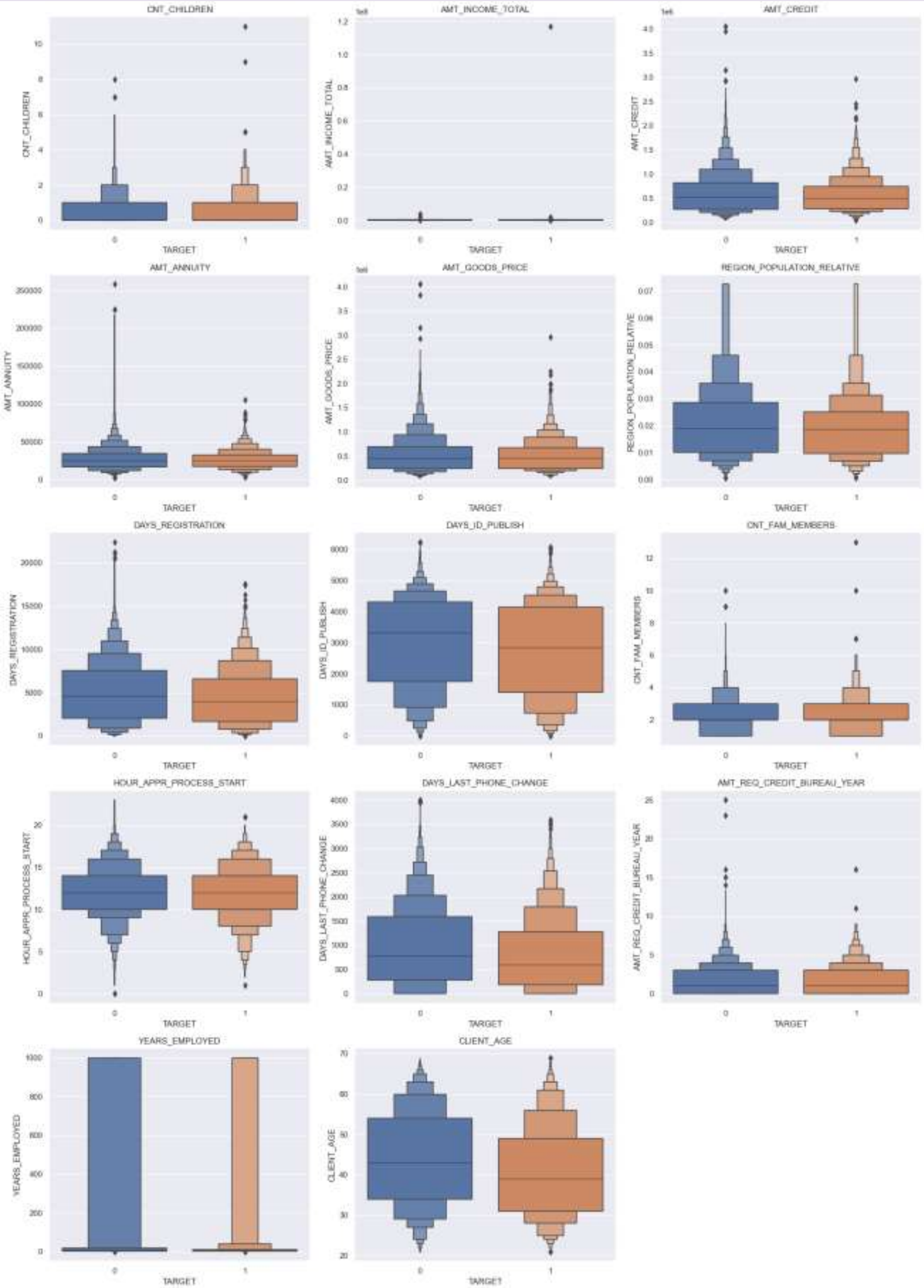
```python
# Display the counts of unique values in the 'NAME_GOODS_CATEGORY' column before modification
print(f"'NAME_GOODS_CATEGORY' column before modification:\n{previ_data['NAME_GOODS_CATEGORY'].value_counts()}")
print("------------------------------------------------------------------------------------")

# Identify the 'XNA' values in the 'NAME_GOODS_CATEGORY' column
xna_values = previ_data['NAME_GOODS_CATEGORY'] == 'XNA'

# Divide 'XNA' values into four equal parts
division_size = len(previ_data[xna_values]) // 5
indices = np.where(xna_values)[0]  # Get the indices of 'XNA' values

# Assign parts to 'Mobile', 'Consumer Electronics', 'Computers', 'Audio/Video', 'Furniture' respectively
previ_data.loc[indices[:division_size], 'NAME_GOODS_CATEGORY'] = 'Mobile'
previ_data.loc[indices[division_size:2*division_size], 'NAME_GOODS_CATEGORY'] = 'Consumer Electronics'
previ_data.loc[indices[2*division_size:3*division_size], 'NAME_GOODS_CATEGORY'] = 'Computers'
previ_data.loc[indices[3*division_size:4*division_size], 'NAME_GOODS_CATEGORY'] = 'Audio/Video'
previ_data.loc[indices[4*division_size:], 'NAME_GOODS_CATEGORY'] = 'Furniture'

# Display the counts of unique values in the 'NAME_GOODS_CATEGORY' column after modification
print(f"'NAME_GOODS_CATEGORY' column after modification:\n{previ_data['NAME_GOODS_CATEGORY'].value_counts()}")
```

```python
# Display the counts of unique values in the 'NAME_SELLER_INDUSTRY' column before modification
print(f"'NAME_SELLER_INDUSTRY' column before modification:\n{previ_data['NAME_SELLER_INDUSTRY'].value_counts()}")
print("------------------------------------------------------------------------------------")

# Identify the 'XNA' values in the 'NAME_SELLER_INDUSTRY' column
xna_values = previ_data['NAME_SELLER_INDUSTRY'] == 'XNA'

# Divide 'XNA' values into four equal parts
division_size = len(previ_data[xna_values]) // 2
indices = np.where(xna_values)[0]  # Get the indices of 'XNA' values

# Assign parts to 'Consumer electronics', 'Connectivity', 'Computers', 'Audio/Video', 'Furniture' respectively
previ_data.loc[indices[:division_size], 'NAME_SELLER_INDUSTRY'] = 'Consumer electronics'
previ_data.loc[indices[division_size:], 'NAME_SELLER_INDUSTRY'] = 'Connectivity'

# Display the counts of unique values in the 'NAME_SELLER_INDUSTRY' column after modification
print(f"'NAME_SELLER_INDUSTRY' column after modification:\n{previ_data['NAME_SELLER_INDUSTRY'].value_counts()}")
```

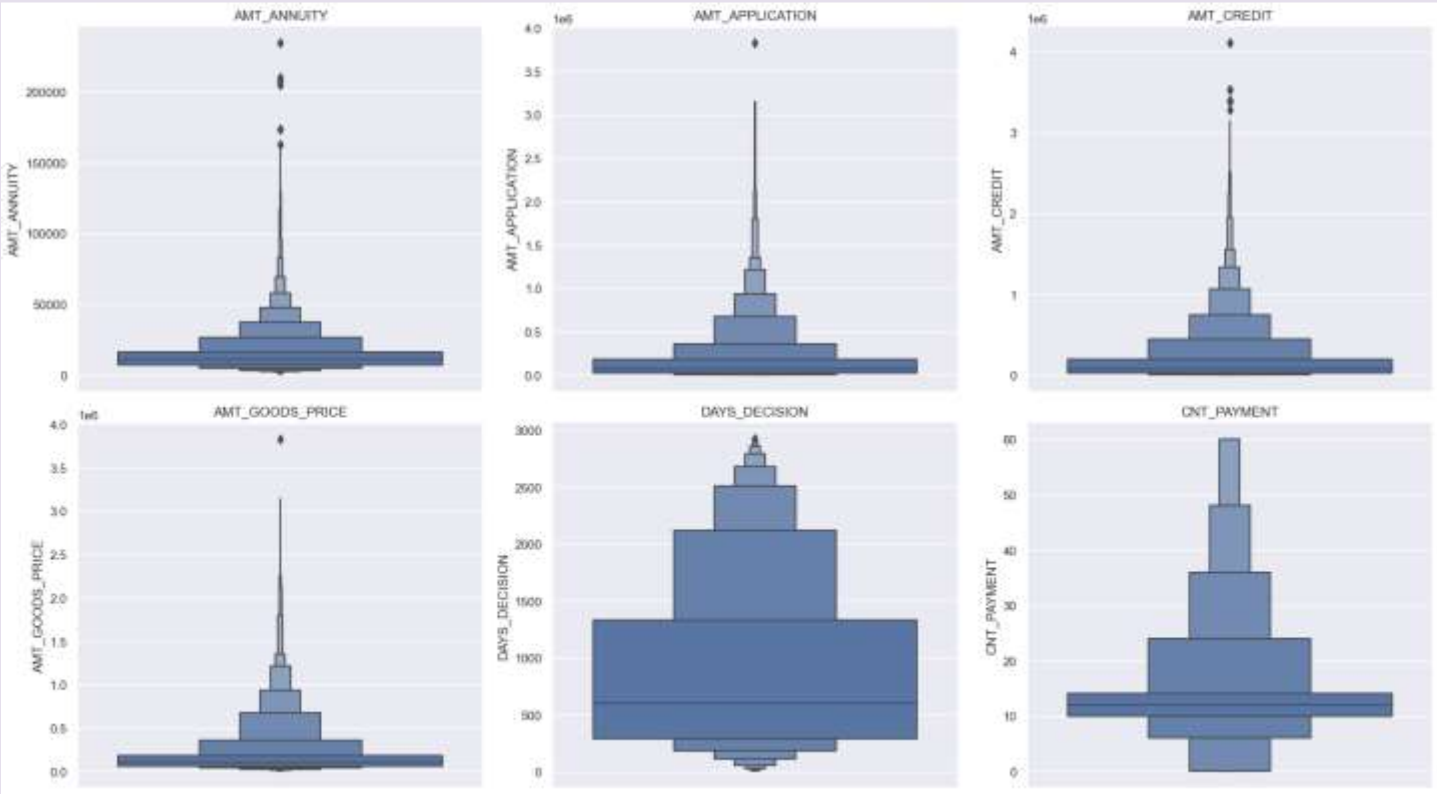
0.0s

## Detecting Outliers:

1. Application data:



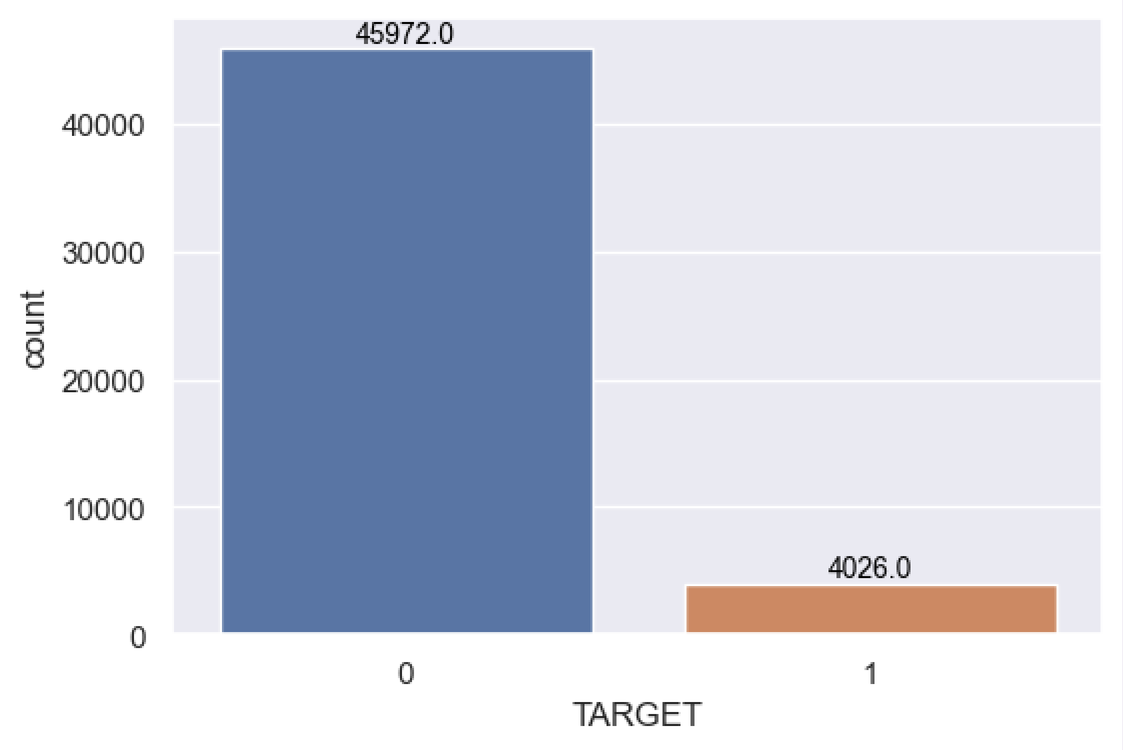
2. Previous Application Data:



## Data Imbalance:

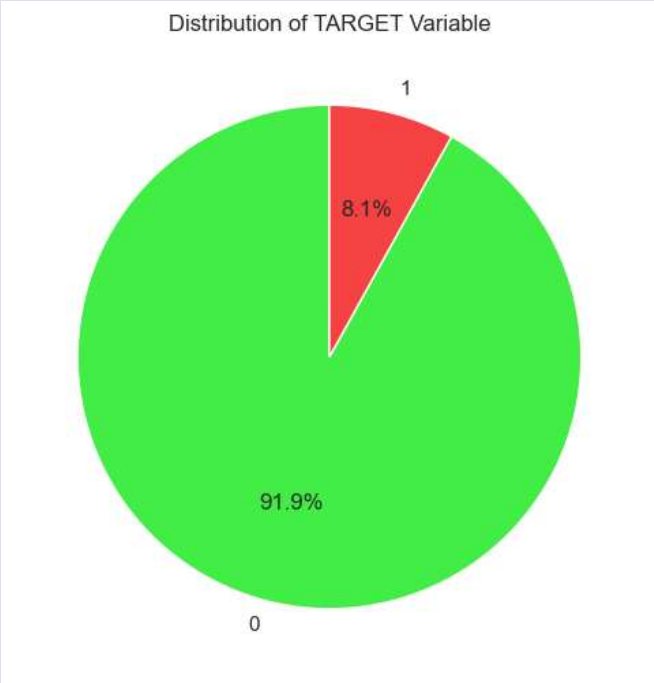
```
💡 Click here to ask Blackbox to help you code faster
# Assuming 'appli_data' is your DataFrame
sns.set(style="darkgrid")
plt.figure(figsize=(6, 4))

# Create a countplot
ax = sns.countplot(x="TARGET", data=appli_data)

# Add counts on top of each bar
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),
                textcoords='offset points')

plt.show()
```

✓ 0.2s




```
💡 Click here to ask Blackbox to help you code faster
# Assuming 'appli_data' is your DataFrame
target_counts = appli_data['TARGET'].value_counts()

# Create a pie chart
plt.figure(figsize=(6, 6))
plt.pie(target_counts, labels=target_counts.index, autopct='%1.1f%%', startangle=90, colors=['#42ed45', '#f54242'])
plt.title('Distribution of TARGET Variable')
plt.show()

```

✓ 0.1s




Click here to ask Blackbox to help you code faster

```
class_counts = appli_data['TARGET'].value_counts()
imbalance_ratio = class_counts[0] / class_counts[1]

"Imbalance Ratio:", round(imbalance_ratio,2)
```

0.0s

```
('Imbalance Ratio:', 11.42)
```

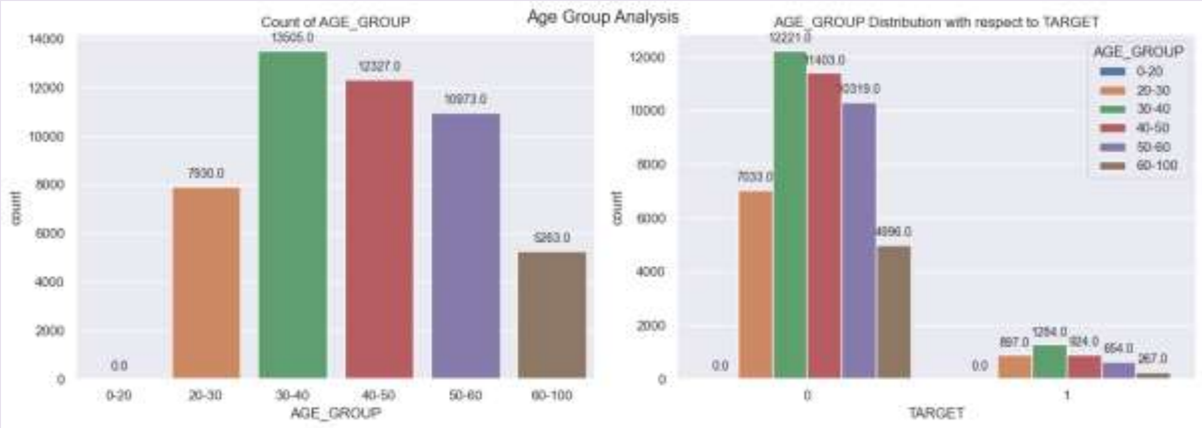
Click here to ask Blackbox to help you code faster

```
app_data_fil = appli_data['TARGET'].value_counts()
class_counts = app_data_fil
imbalance_ratio = class_counts[0] / class_counts[1]

"Imbalance Ratio: 1:{:.0f}".format(imbalance_ratio)
```

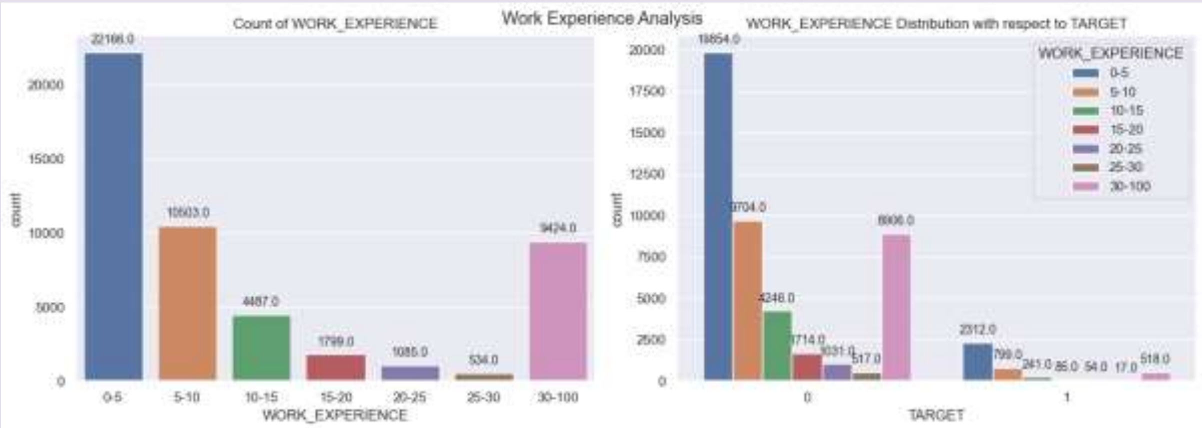
0.0s

```
'Imbalance Ratio: 1:11'
```

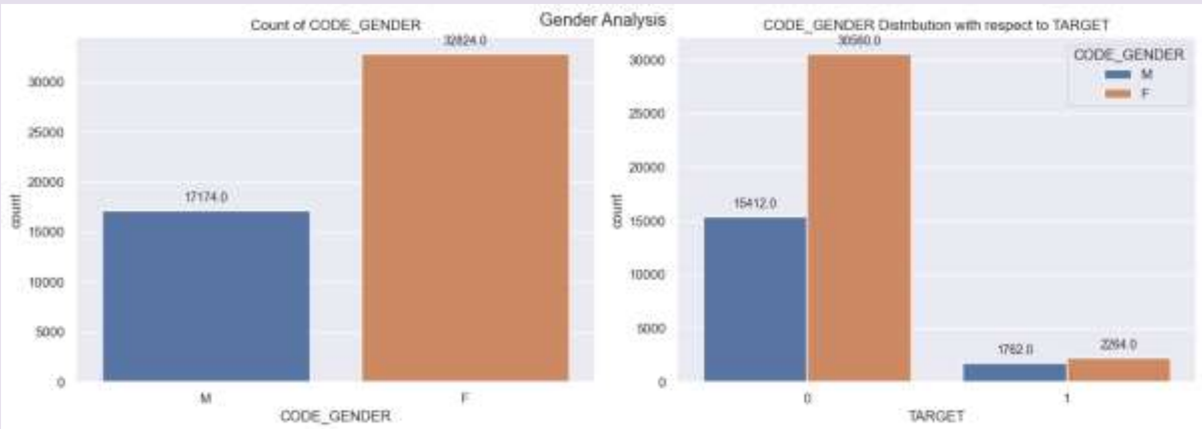
## Univariate Analysis:



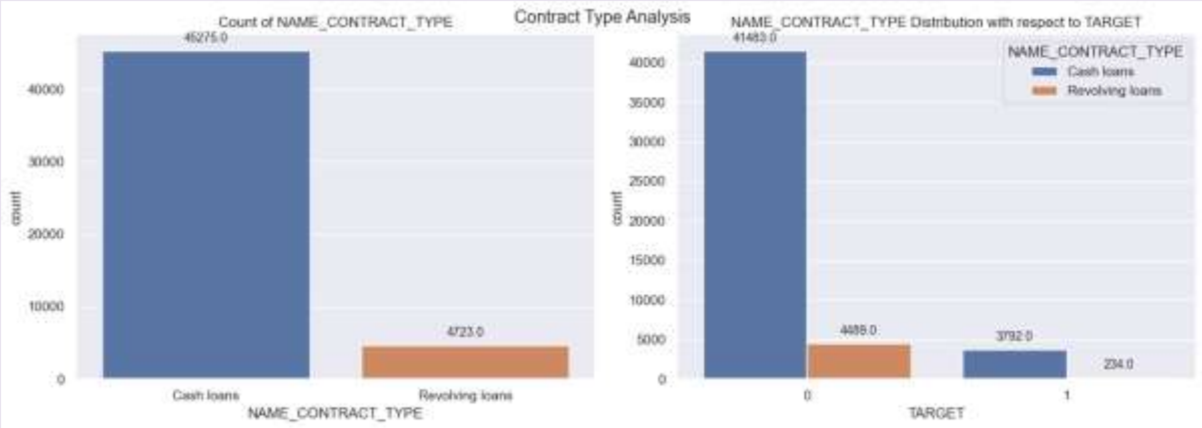

The age group of applicant of age 30-40 and 40-50 and having highest numbers of applicants for loan process.



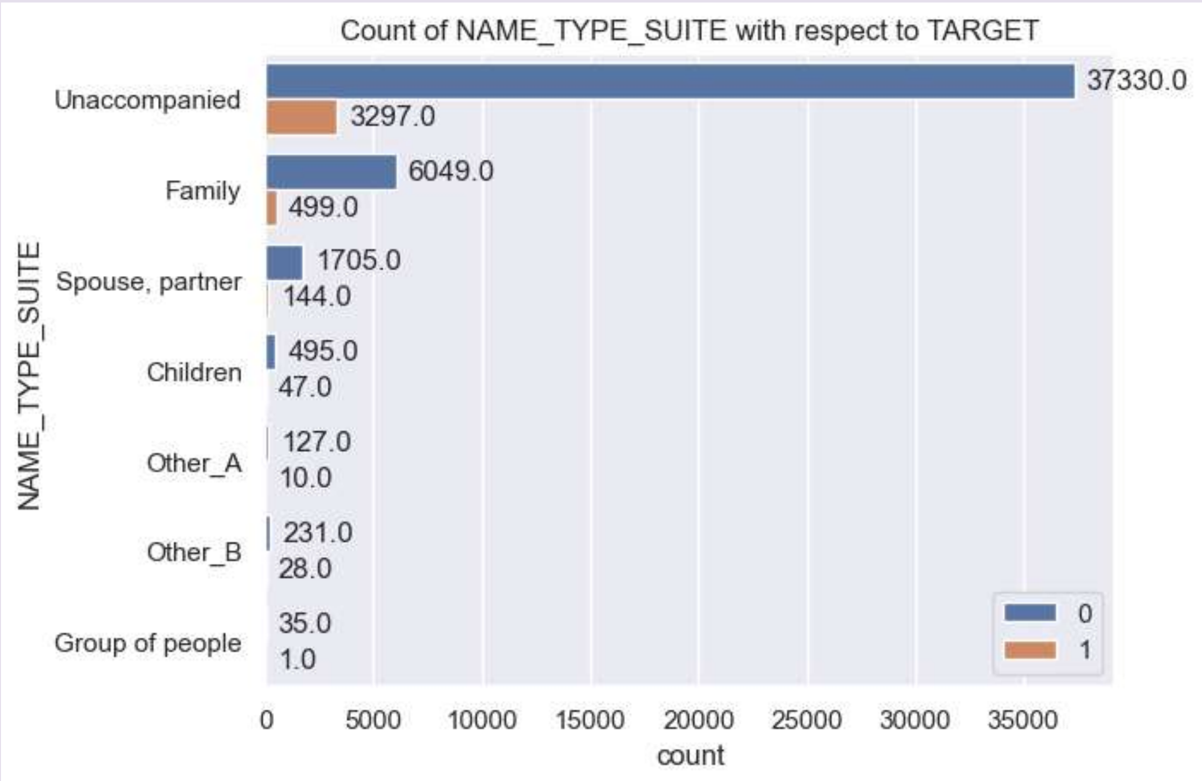

The employees who are having less than 5 years of work experience are applying for the loan and are not having problems while repaying loan.

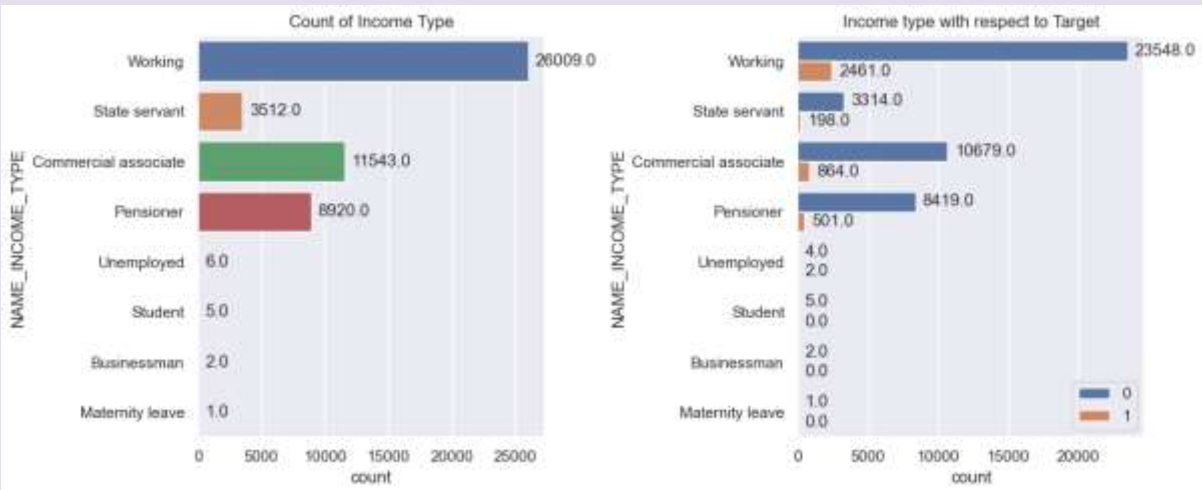


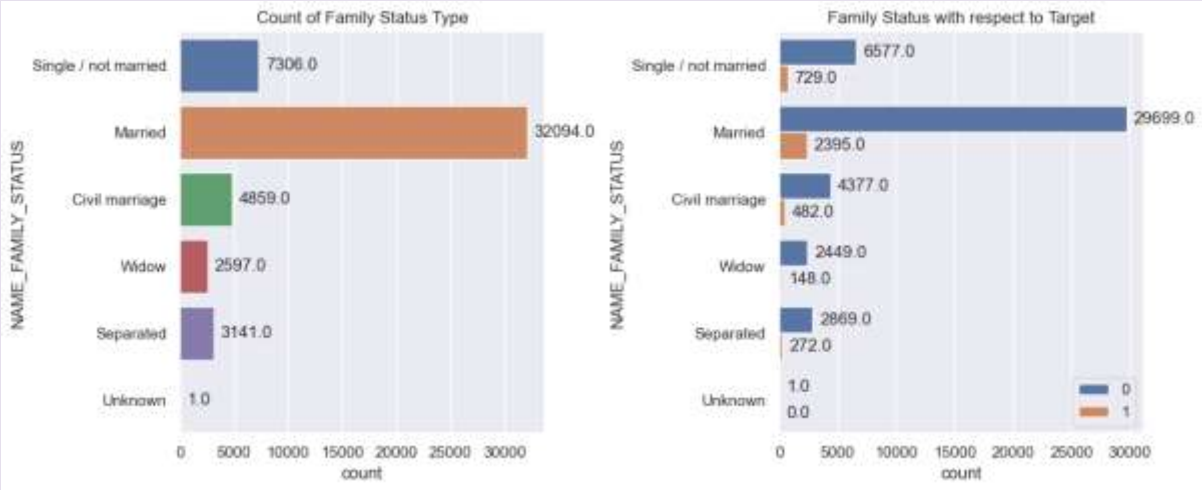

It seems like Female client applied higher than male client for loan.



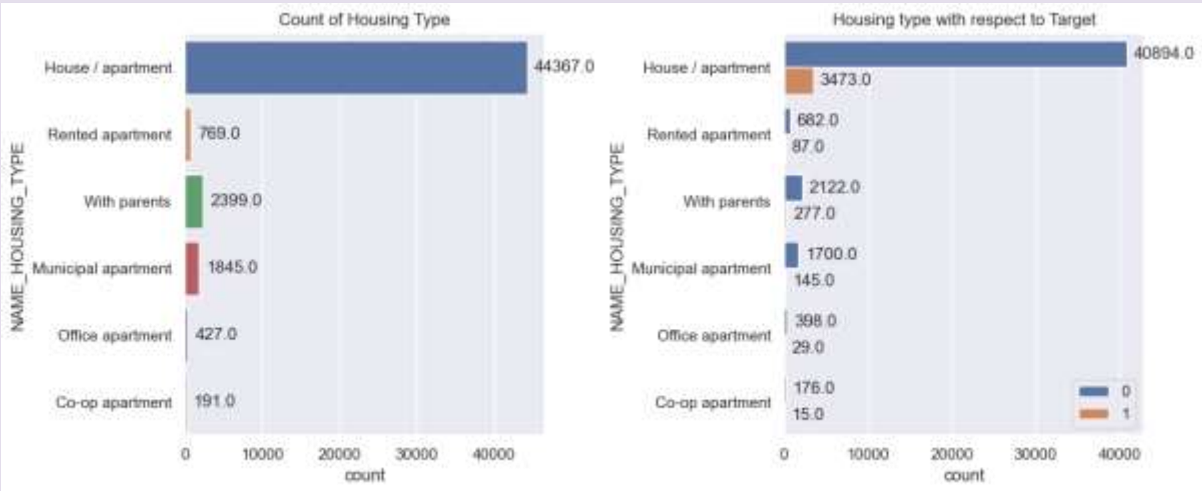

We can see that 90% of peoples are taking Cash loans and 10% are applying for Revolving loans.

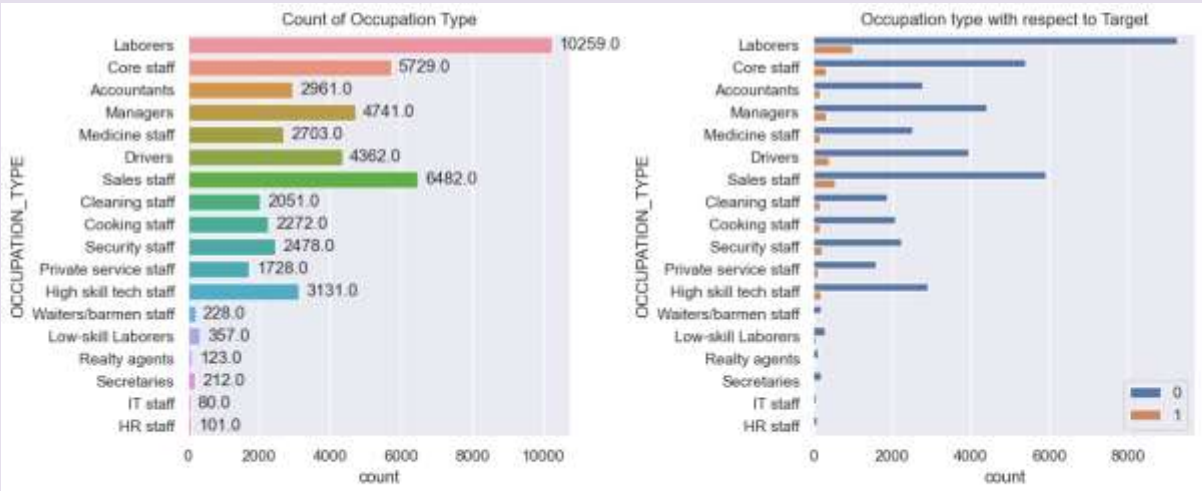


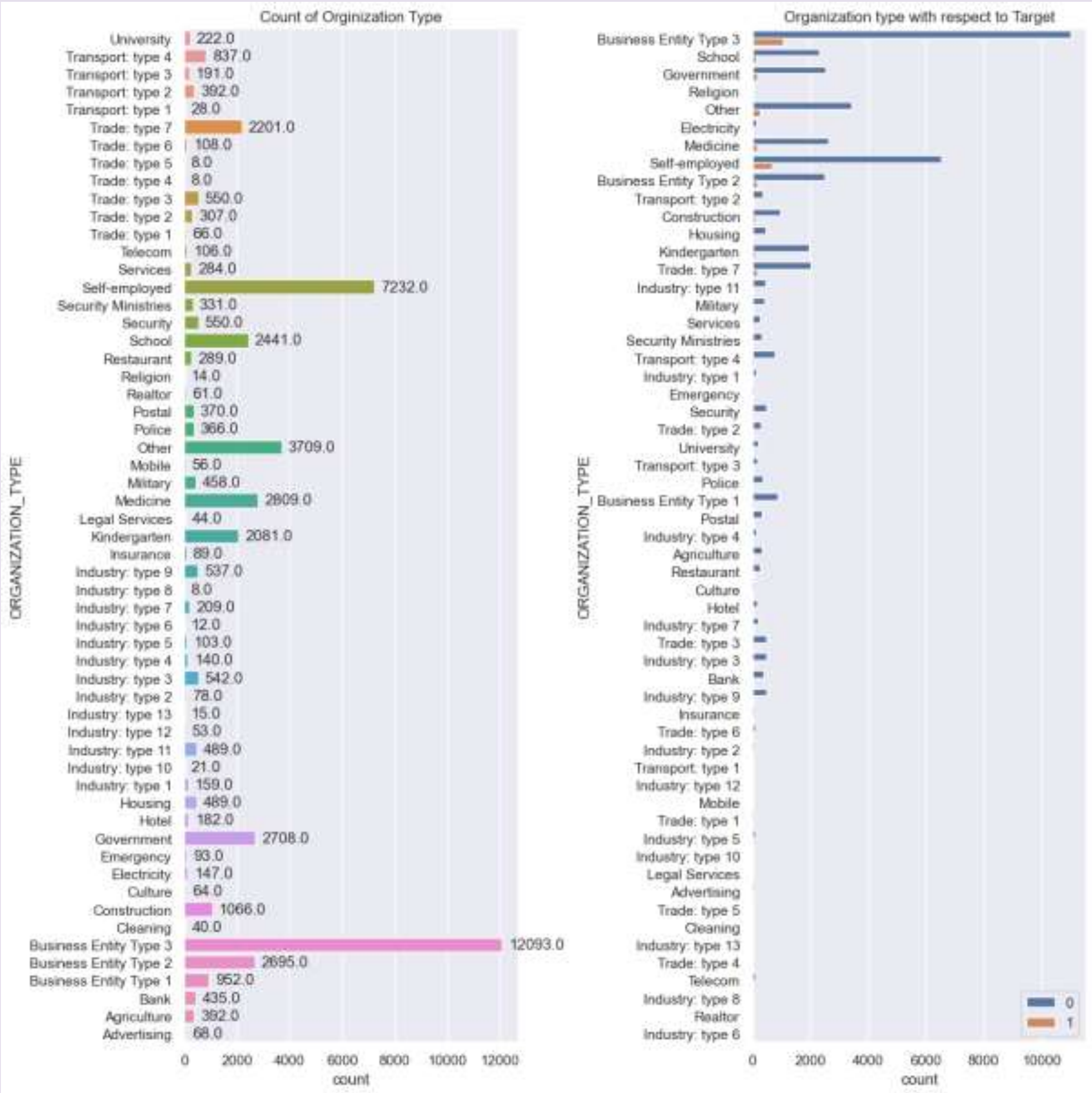



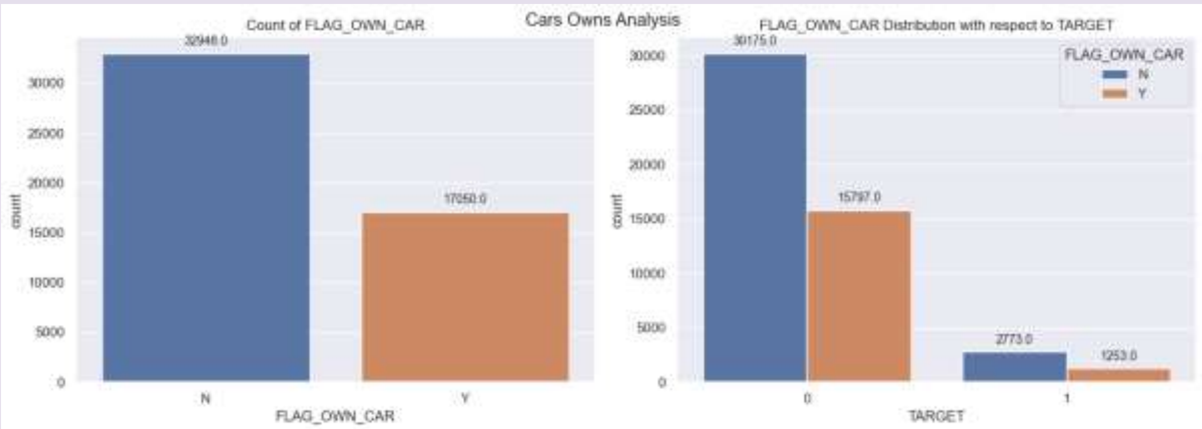


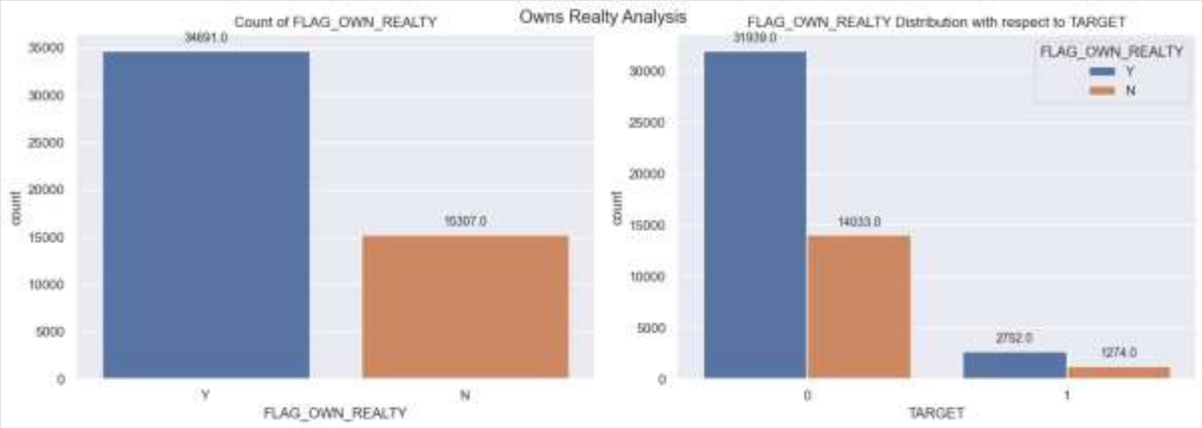

We can see that married peoples are taking more loan than the single ,civil, window and separated people.



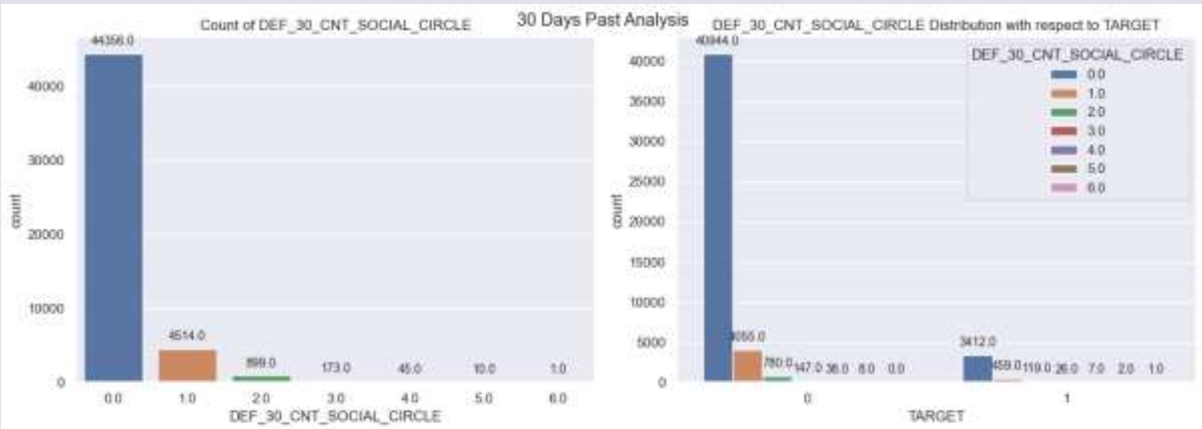

Count of Orginization Type
University 222.0
Transport: type 4 837.0
Transport: type 3 191.0
Transport: type 2 392.0
Transport: type 1 28.0
Trade: type 7 2201.0
Trade: type 6 108.0
Trade: type 5 8.0
Trade: type 4 8.0
Trade: type 3 550.0
Trade: type 2 307.0
Trade: type 1 66.0
Telecom 106.0
Services 284.0
Self-employed 7232.0
Security Ministries 331.0
Security 550.0
School 2441.0
Restaurant 289.0
Religion 14.0
Realtor 61.0
Postal 370.0
Police 366.0
Other 3709.0
Mobile 56.0
Military 458.0
Medicine 2809.0
Legal Services 44.0
Kindergarten 2081.0
Insurance 89.0
Industry: type 9 537.0
Industry: type 8 8.0
Industry: type 7 209.0
Industry: type 6 12.0
Industry: type 5 103.0
Industry: type 4 140.0
Industry: type 3 542.0
Industry: type 2 78.0
Industry: type 13 15.0
Industry: type 12 53.0
Industry: type 11 489.0
Industry: type 10 21.0
Industry: type 1 159.0
Housing 489.0
Hotel 182.0
Government 2708.0
Emergency 93.0
Electricity 147.0
Culture 64.0
Construction 1066.0
Cleaning 40.0
Business Entity Type 3 12093.0
Business Entity Type 2 2695.0
Business Entity Type 1 952.0
Bank 435.0
Agriculture 392.0
Advertising 68.0
ORGANIZATION_TYPE
0 2000 4000 6000 8000 10000 12000
count
Organization type with respect to Target
Business Entity Type 3
School
Government
Religion
Other
Electricity
Medicine
Self-employed
Business Entity Type 2
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
ORGANIZATION_TYPE
0
1
0 2000 4000 6000 8000 10000
count


Cars Owns Analysis
Count of FLAG_OWN_CAR
32946.0
17050.0
count
0 5000 10000 15000 20000 25000 30000
N Y
FLAG_OWN_CAR
FLAG_OWN_CAR Distribution with respect to TARGET
FLAG_OWN_CAR
N
Y
30175.0
15797.0
2773.0
1253.0
count
0 5000 10000 15000 20000 25000 30000
0 1
TARGET

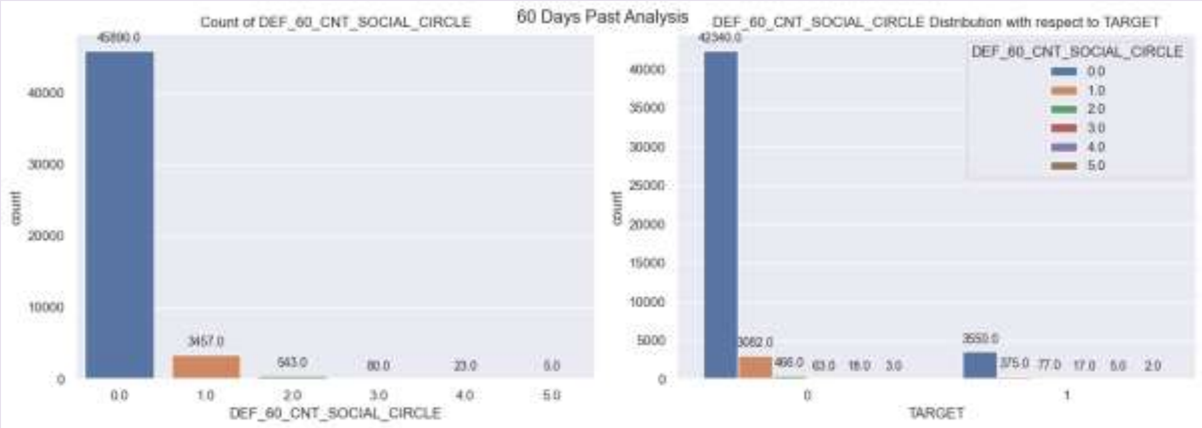


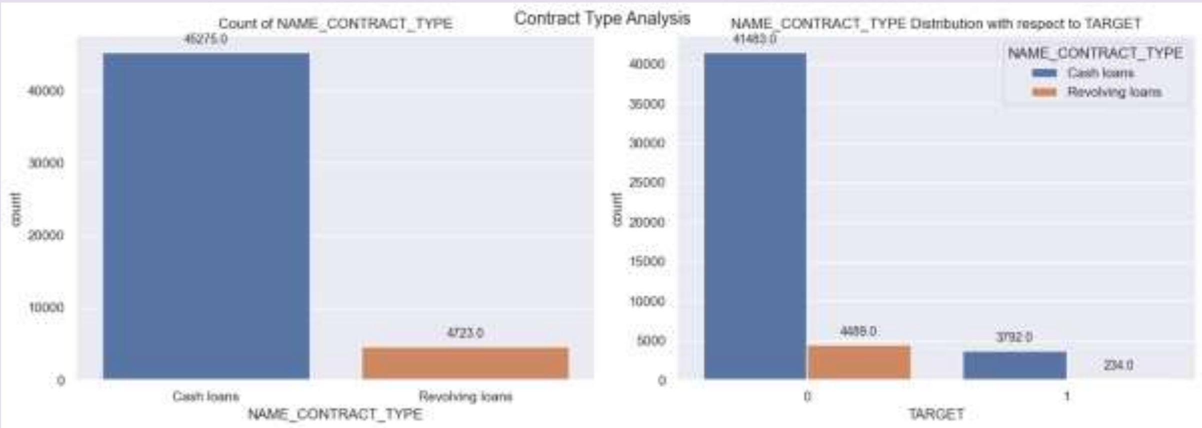
Owns Realty Analysis
Count of FLAG_OWN_REALTY
34691.0
15307.0
Y
N
FLAG_OWN_REALTY
count
FLAG_OWN_REALTY Distribution with respect to TARGET
FLAG_OWN_REALTY
31939.0
14033.0
2752.0
1274.0
TARGET

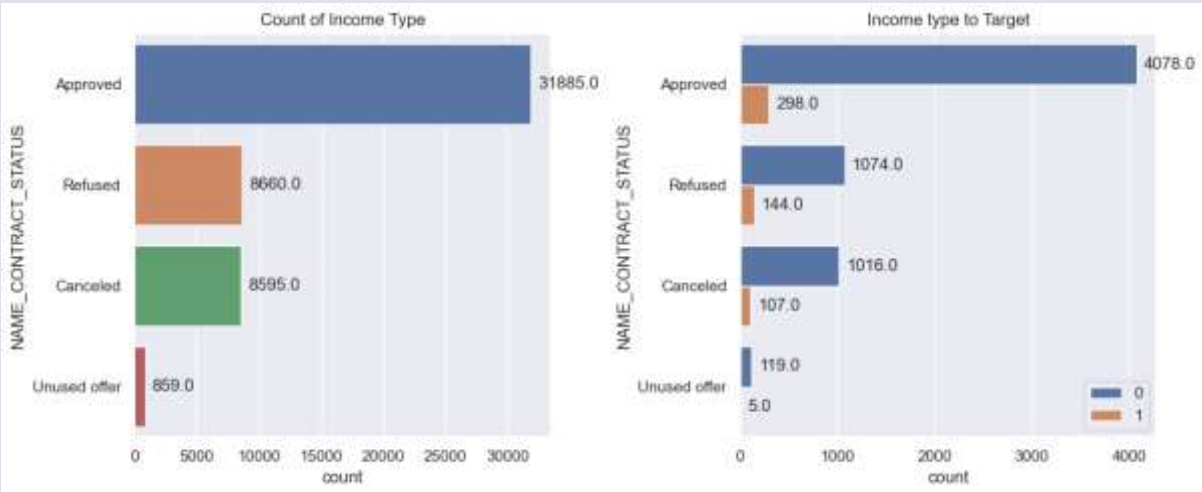

Regional Rating Analysis
Count of REGION_RATING_CLIENT
5226.0
7809.0
REGION_RATING_CLIENT
count
REGION_RATING_CLIENT Distribution with respect to TARGET
REGION_RATING_CLIENT
34042.0
5006.0
6924.0
220.0
2921.0
885.0
TARGET

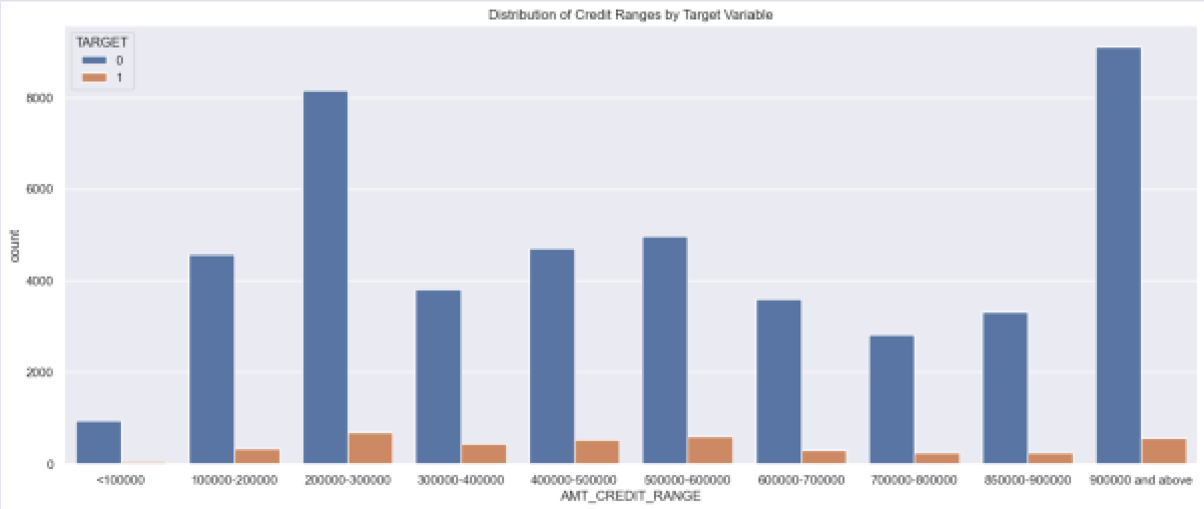
30 Days Past Analysis
Count of DEF_30_CNT_SOCIAL_CIRCLE
44356.0
4514.0
899.0
173.0
45.0
10.0
1.0
DEF_30_CNT_SOCIAL_CIRCLE
count
DEF_30_CNT_SOCIAL_CIRCLE Distribution with respect to TARGET
DEF_30_CNT_SOCIAL_CIRCLE
40944.0
780.0
147.0
36.0
8.0
0.0
3412.0
459.0
119.0
26.0
7.0
2.0
1.0
TARGET


60 Days Past Analysis
Count of DEF_60_CNT_SOCIAL_CIRCLE
DEF_60_CNT_SOCIAL_CIRCLE Distribution with respect to TARGET
DEF_60_CNT_SOCIAL_CIRCLE
count
TARGET
45890.0
3457.0
543.0
80.0
23.0
5.0
42340.0
3082.0
466.0
63.0
18.0
3.0
3550.0
375.0
77.0
17.0
5.0
2.0

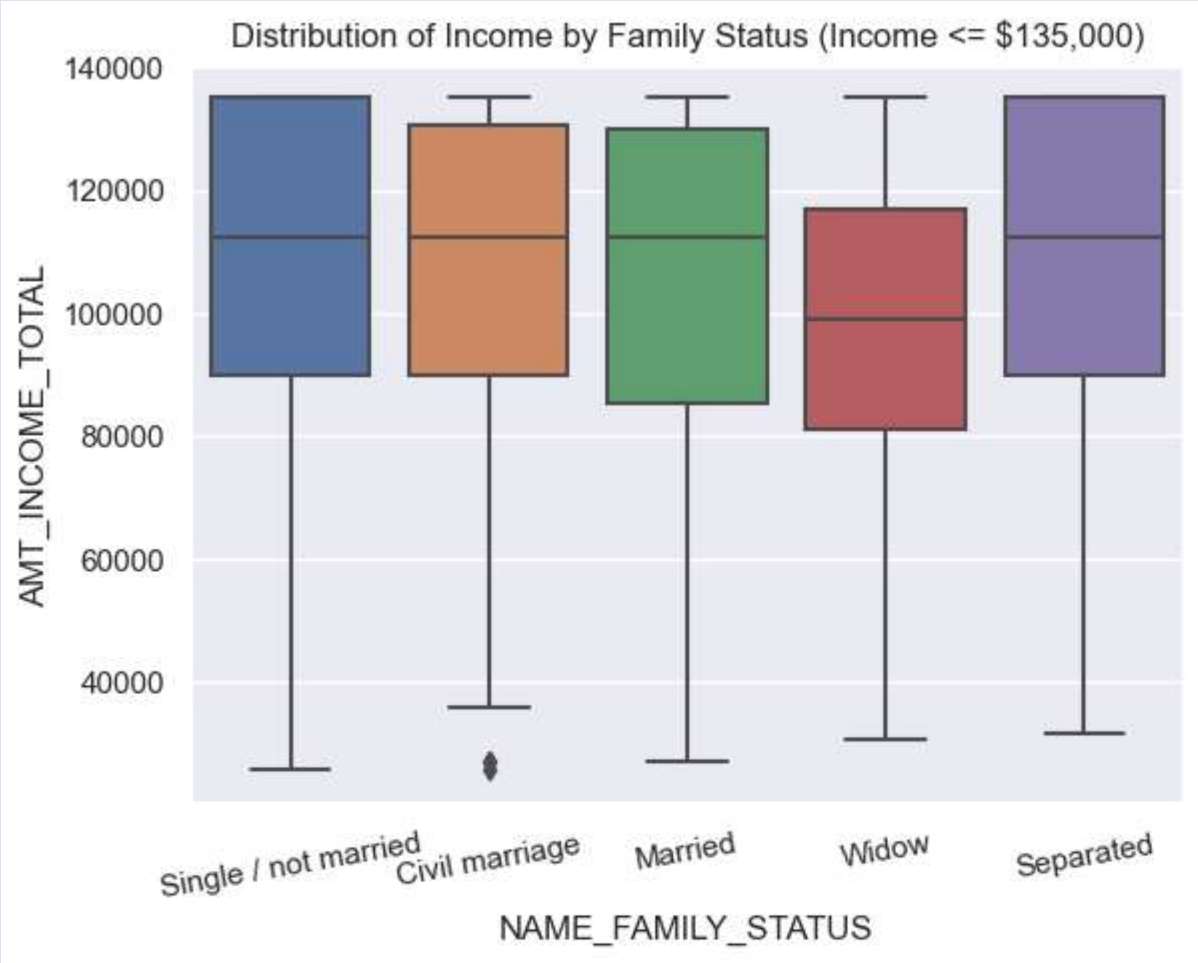

Contract Type Analysis
Count of NAME_CONTRACT_TYPE
NAME_CONTRACT_TYPE Distribution with respect to TARGET
NAME_CONTRACT_TYPE
Cash loans
Revolving loans
count
TARGET
45275.0
4723.0
41483.0
4489.0
3792.0
234.0

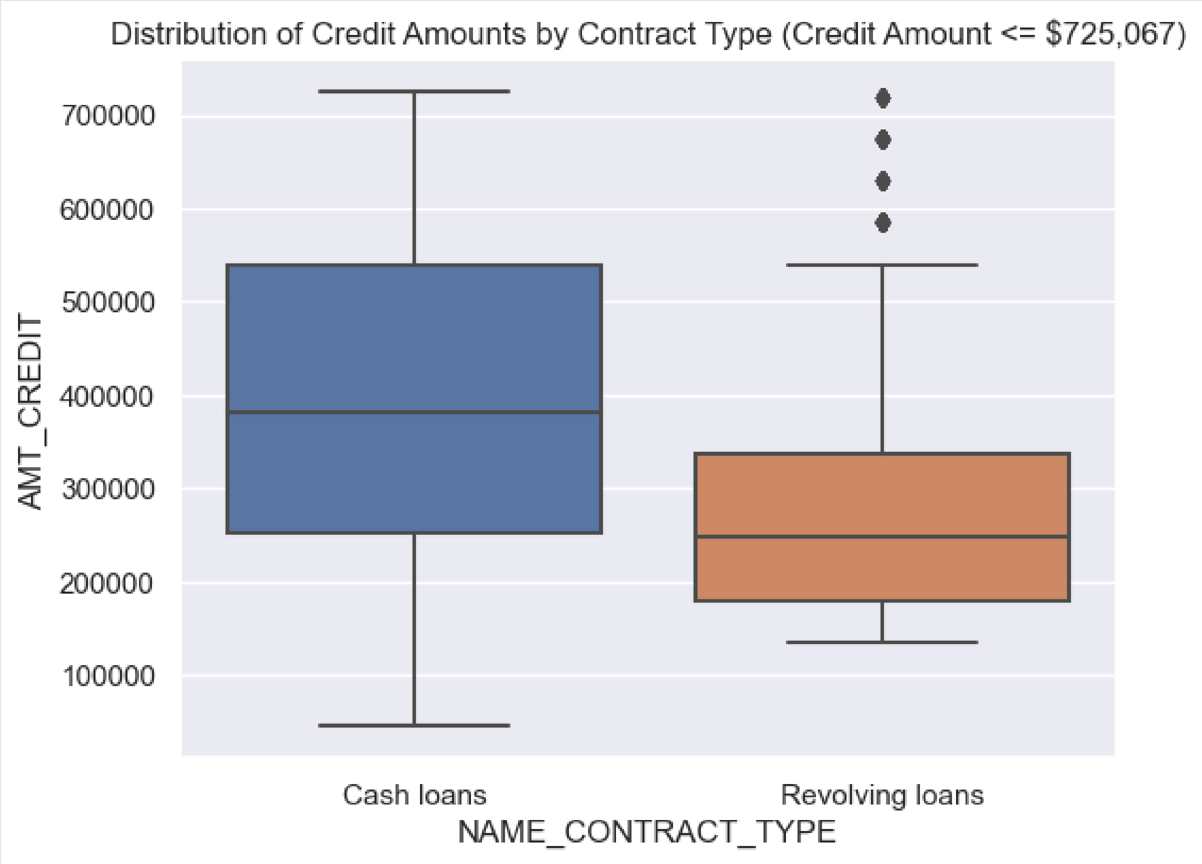

Count of Income Type
Income type to Target
NAME_CONTRACT_STATUS
Approved
Refused
Canceled
Unused offer
count
31885.0
8660.0
8595.0
859.0
4078.0
298.0
1074.0
144.0
1016.0
107.0
119.0
5.0

Distribution of Income Ranges by Target Variable
TARGET
0
1
count
20000
15000
10000
5000
0
<100000
100000-200000
200000-300000
300000-400000
400000-500000
500000 and above
AMT_INCOME_RANGE

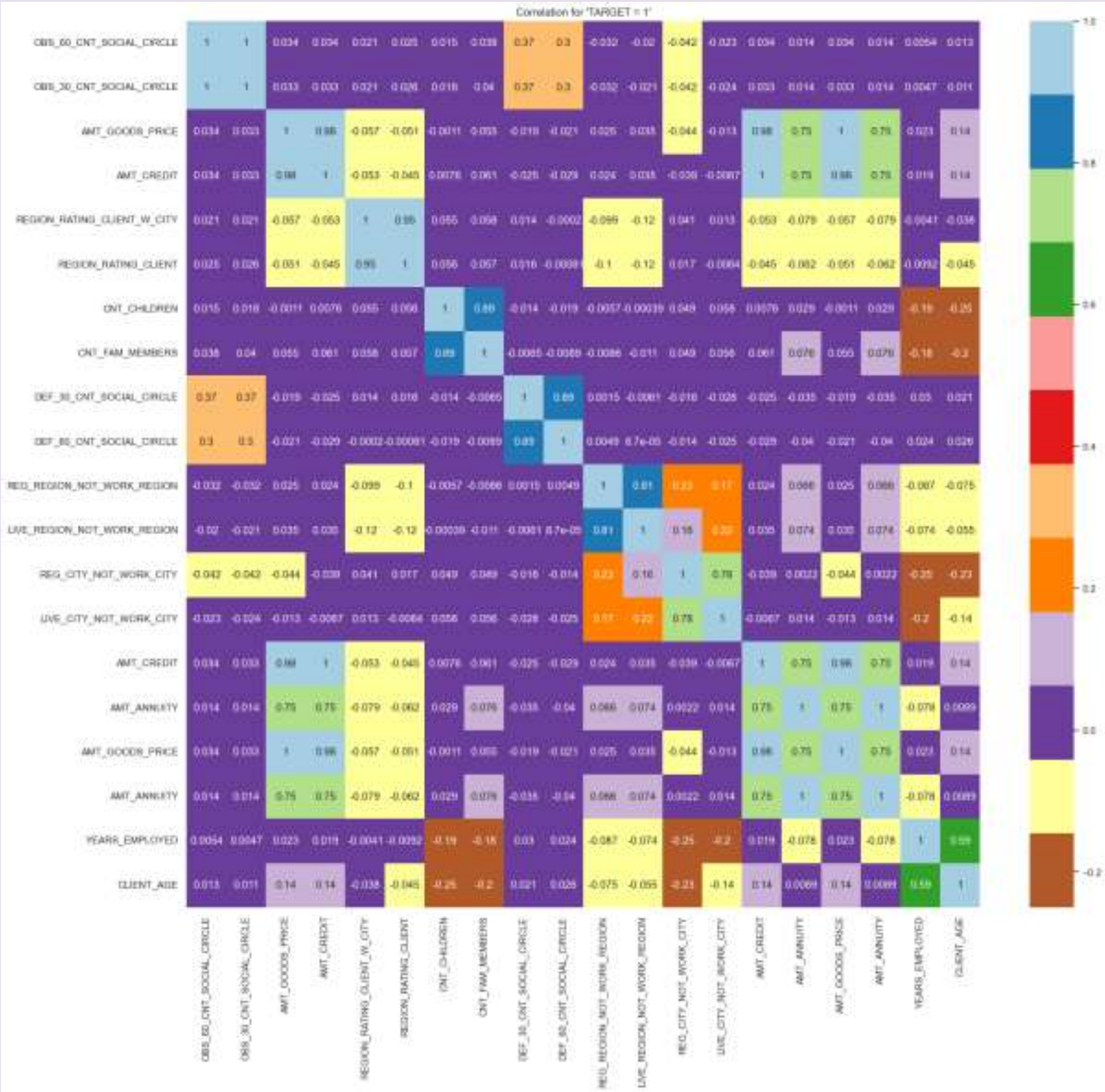
Distribution of Credit Ranges by Target Variable
TARGET
0
1
count
8000
6000
4000
2000
0
<100000
100000-200000
200000-300000
300000-400000
400000-500000
500000-600000
600000-700000
700000-800000
850000-900000
900000 and above
AMT_CREDIT_RANGE


Distribution of Income by Family Status (Income <= $135,000)
140000
120000
100000
80000
60000
40000
AMT_INCOME_TOTAL
Single / not married
Civil marriage
Married
Widow
Separated
NAME_FAMILY_STATUS

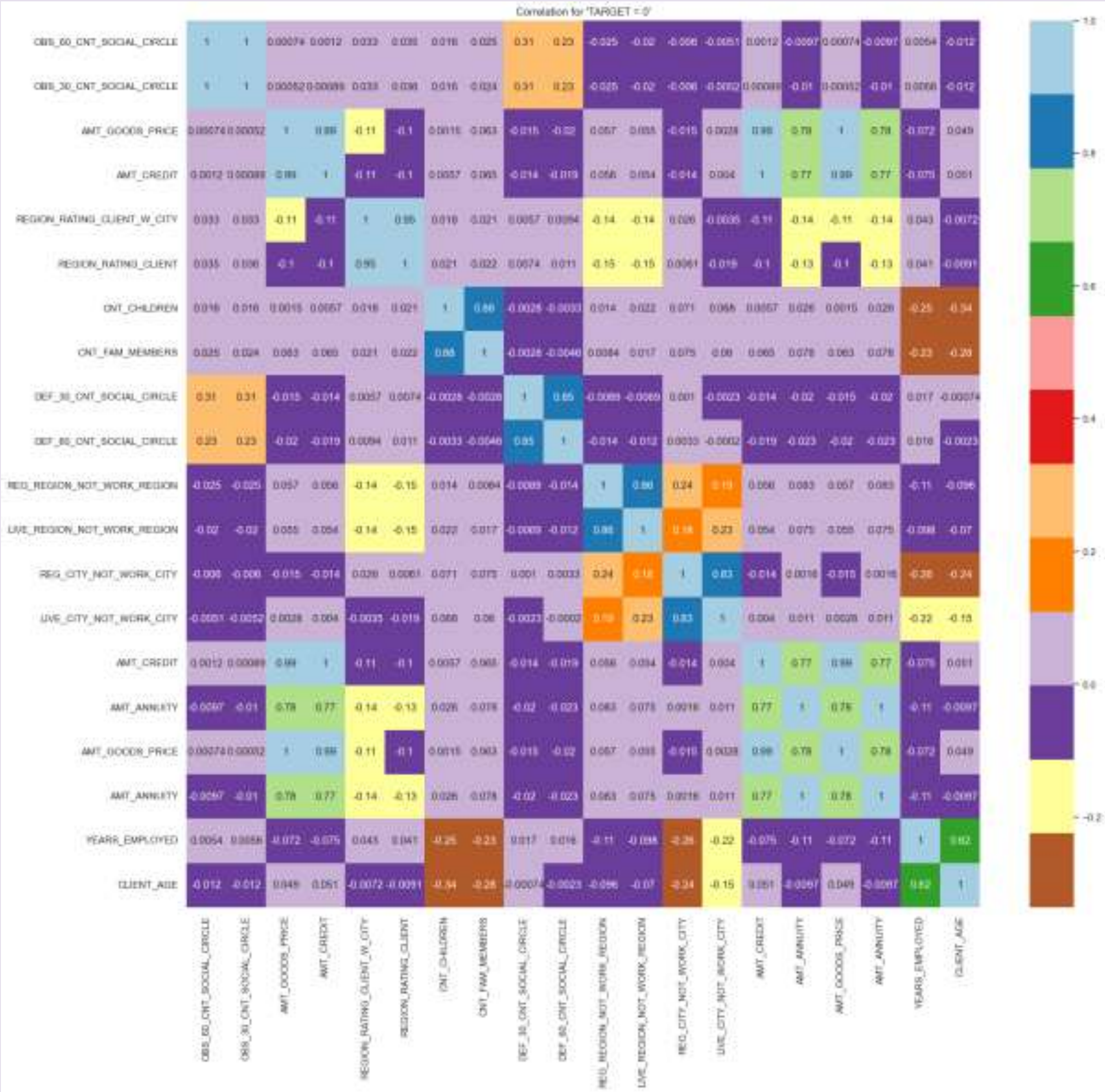

Distribution of Credit Amounts by Contract Type (Credit Amount <= $725,067)
700000
600000
500000
400000
300000
200000
100000
AMT_CREDIT
Cash loans
Revolving loans
NAME_CONTRACT_TYPE

# Correlation Analysis:

Correlation analysis for Target 1



Correlation analysis for Target 0



Link of complete Python workbook:
https://drive.google.com/file/d/1BeSueJIleSqkeTRgK0l28DjjZqMpIhCx/view?usp=drive_link

Video Link:
https://drive.google.com/file/d/1Aq_ILYDxTbOUepryI4ibx6ckQehAppD8/view?usp=drive_link