```
# Importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Optional: To display plots inline in notebooks
%matplotlib inline



# Load the dataset (replace 'application_data.csv' with your dataset path)
application_data = pd.read_csv('/content/application_data.csv')

# Display the first few rows of the dataset
application_data.head()
```

	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	

5 rows × 122 columns

```
# Checking the basic info of the dataset
application_data.info()

# Summary statistics of the dataset
application_data.describe()

# Checking the number of unique values for each column
application_data.nunique()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

	0
SK_ID_CURR	307511
TARGET	2
NAME_CONTRACT_TYPE	2
CODE_GENDER	3
FLAG_OWN_CAR	2
...	...
AMT_REQ_CREDIT_BUREAU_DAY	9
AMT_REQ_CREDIT_BUREAU_WEEK	9
AMT_REQ_CREDIT_BUREAU_MON	24
AMT_REQ_CREDIT_BUREAU_QRT	11
AMT_REQ_CREDIT_BUREAU_YEAR	25

122 rows × 1 columns

dtype: int64

```
# Checking for missing values
missing_values = application_data.isnull().sum()
missing_values_percentage = (missing_values / len(application_data)) * 100

# Display missing values and their percentages
missing_data_df = pd.DataFrame({
    'Total_Missing': missing_values,
    'Percentage_Missing': missing_values_percentage
})
```

```
print(missing_data_df.sort_values(by='Percentage_Missing', ascending=False))
```

```
                          Total_Missing  Percentage_Missing
COMMONAREA_MEDI                  214865           69.872297
COMMONAREA_AVG                   214865           69.872297
COMMONAREA_MODE                  214865           69.872297
NONLIVINGAPARTMENTS_MODE         213514           69.432963
NONLIVINGAPARTMENTS_AVG          213514           69.432963
...                                 ...                 ...
NAME_HOUSING_TYPE                    0            0.000000
NAME_FAMILY_STATUS                   0            0.000000
NAME_EDUCATION_TYPE                  0            0.000000
NAME_INCOME_TYPE                     0            0.000000
SK_ID_CURR                           0            0.000000

[122 rows x 2 columns]
```

```
# Display the percentage of missing values
missing_data_df.sort_values(by='Percentage_Missing', ascending=False)
```

	Total_Missing	Percentage_Missing
COMMONAREA_MEDI	214865	69.872297
COMMONAREA_AVG	214865	69.872297
COMMONAREA_MODE	214865	69.872297
NONLIVINGAPARTMENTS_MODE	213514	69.432963
NONLIVINGAPARTMENTS_AVG	213514	69.432963
...	...	...
NAME_HOUSING_TYPE	0	0.000000
NAME_FAMILY_STATUS	0	0.000000
NAME_EDUCATION_TYPE	0	0.000000
NAME_INCOME_TYPE	0	0.000000
SK_ID_CURR	0	0.000000

122 rows × 2 columns

```
# Dropping columns with missing values > 47%
columns_to_drop = missing_data_df[missing_data_df['Percentage_Missing'] > 47].index
application_data_cleaned = application_data.drop(columns=columns_to_drop)

print(f"Columns dropped: {columns_to_drop}")
```

```
Columns dropped: 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')
```

```
# Handling missing values in 'OCCUPATION_TYPE' (as it's categorical)
application_data_cleaned['OCCUPATION_TYPE'].fillna('Unknown', inplace=True)

# Check if the missing values are filled
application_data_cleaned['OCCUPATION_TYPE'].isnull().sum()
```

```
0
```

```
# Imputing missing values for 'EXT_SOURCE_3' using the median
application_data_cleaned['EXT_SOURCE_3'].fillna(application_data_cleaned['EXT_SOURCE_3'].median(), inplace=True)

# Verify missing values have been handled
```

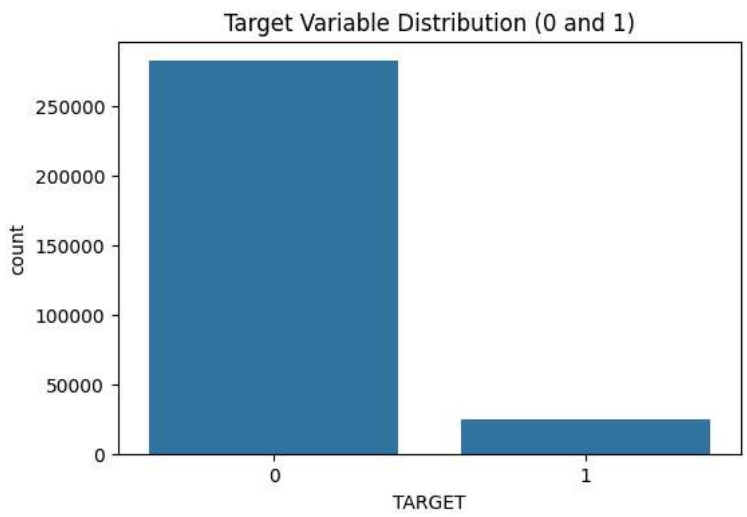
```
application_data_cleaned['EXT_SOURCE_3'].isnull().sum()
```

```
0
```

```
# Binning 'AMT_CREDIT' into categories
application_data_cleaned['AMT_CREDIT_BIN'] = pd.qcut(application_data_cleaned['AMT_CREDIT'], 4, labels=['Low', 'Medium', 'High', 'Very Hi

# Verify binning
application_data_cleaned[['AMT_CREDIT', 'AMT_CREDIT_BIN']].head()
```

	AMT_CREDIT	AMT_CREDIT_BIN
0	406597.5	Medium
1	1293502.5	Very High
2	135000.0	Low
3	312682.5	Medium
4	513000.0	Medium

```
# Checking imbalance in the target variable
target_count = application_data_cleaned['TARGET'].value_counts()

# Plotting the target variable distribution
plt.figure(figsize=(6,4))
sns.countplot(x='TARGET', data=application_data_cleaned)
plt.title('Target Variable Distribution (0 and 1)')
plt.show()

# Display counts
print(target_count)
```



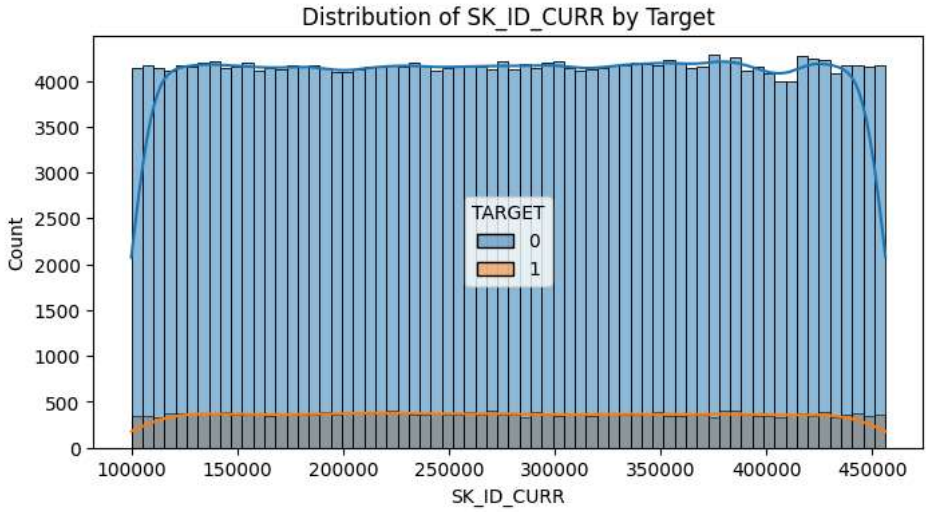

```
TARGET
0    282686
1     24825
Name: count, dtype: int64
```

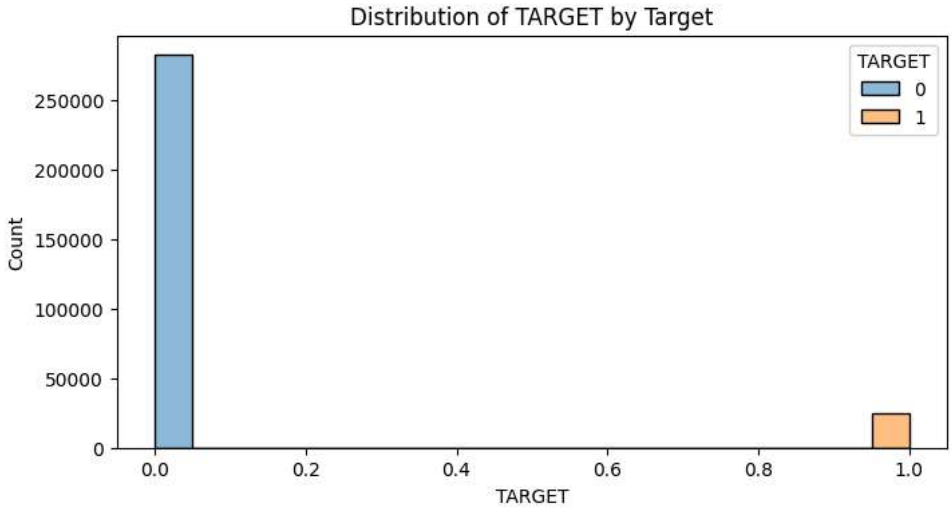
```
# Splitting the dataset into two based on the target variable
data_target_0 = application_data_cleaned[application_data_cleaned['TARGET'] == 0]
data_target_1 = application_data_cleaned[application_data_cleaned['TARGET'] == 1]

# Verify sizes
print(f"Data with TARGET = 0: {len(data_target_0)}")
print(f"Data with TARGET = 1: {len(data_target_1)}")
```

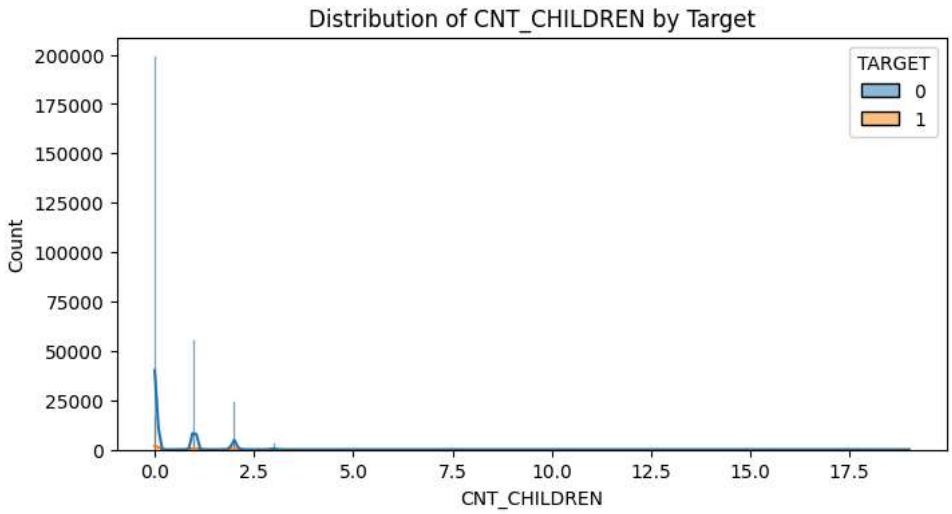
```
Data with TARGET = 0: 282686
Data with TARGET = 1: 24825
```

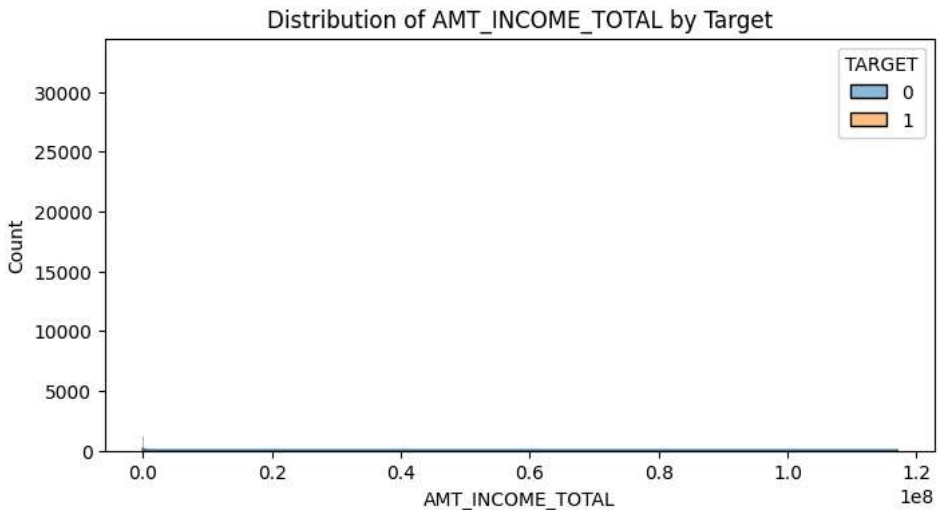
```
# Visualizing numeric columns based on target
numeric_columns = application_data_cleaned.select_dtypes(include=['int64', 'float64']).columns

# Plotting histograms for each numeric column
for column in numeric_columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(data=application_data_cleaned, x=column, hue='TARGET', kde=True)
```

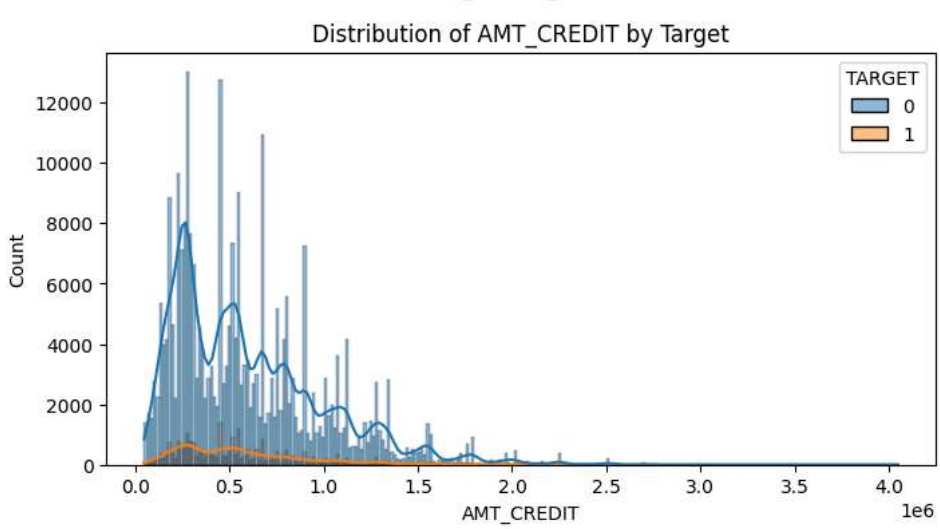
```
        plt.title(f'Distribution of {column} by Target')
        plt.show()
```

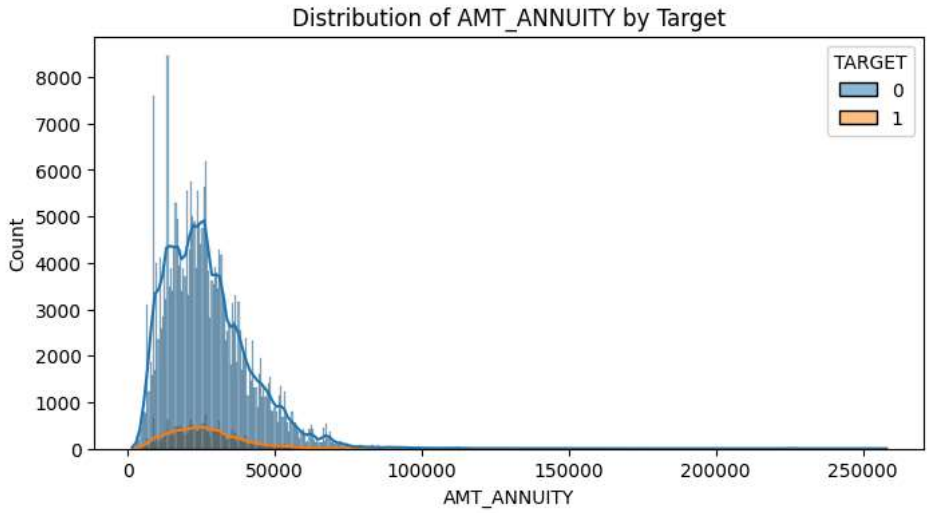

Distribution of SK_ID_CURR by Target
Count
SK_ID_CURR
TARGET
0
1

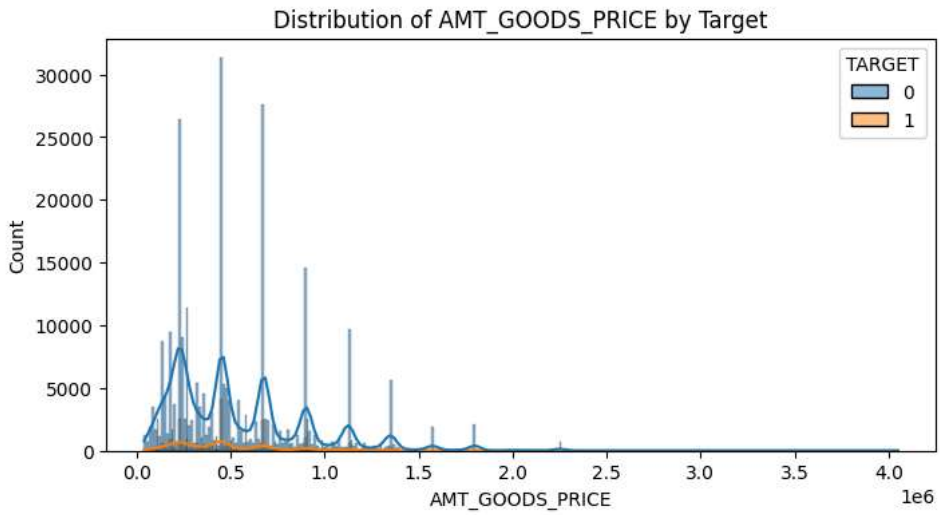

Distribution of TARGET by Target
Count
TARGET
TARGET
0
1

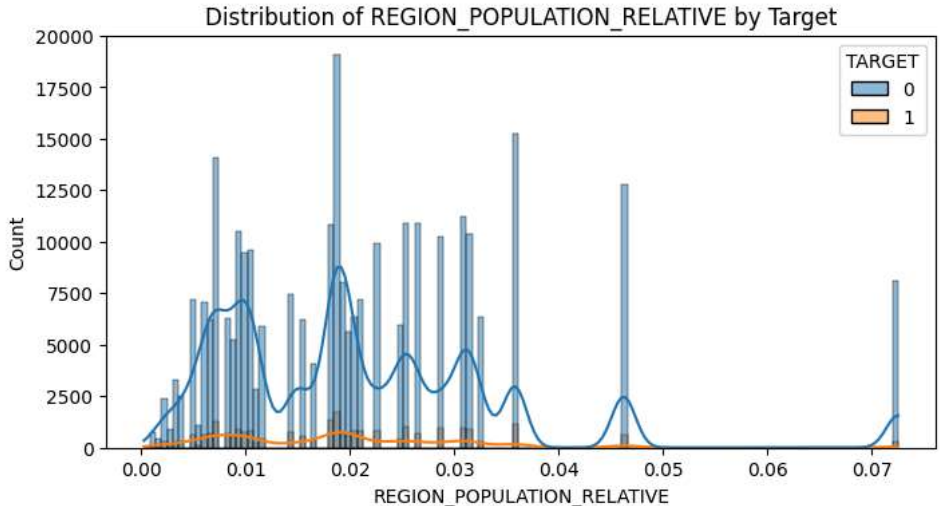

Distribution of CNT_CHILDREN by Target
Count
CNT_CHILDREN
TARGET
0
1

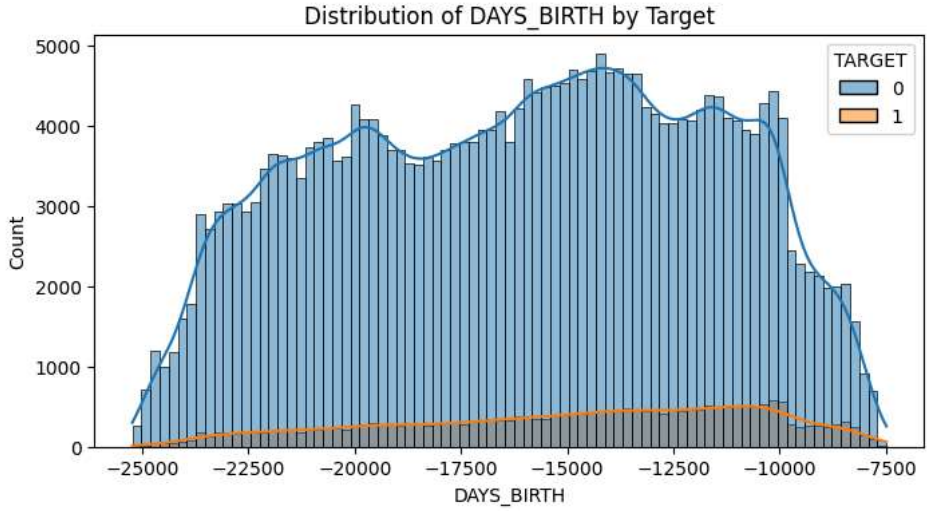

Distribution of AMT_INCOME_TOTAL by Target
Count
AMT_INCOME_TOTAL
1e8
TARGET
0
1

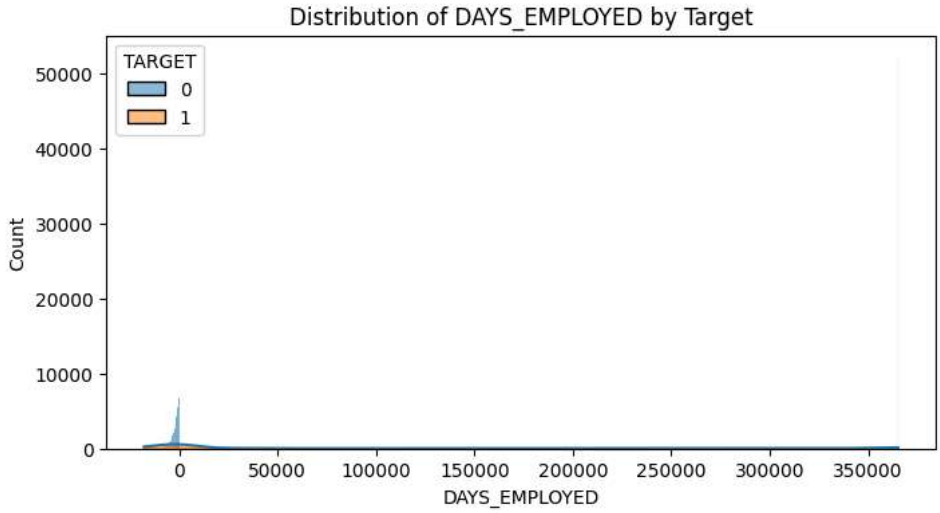

Distribution of AMT_CREDIT by Target
TARGET
0
1
Count
AMT_CREDIT
1e6

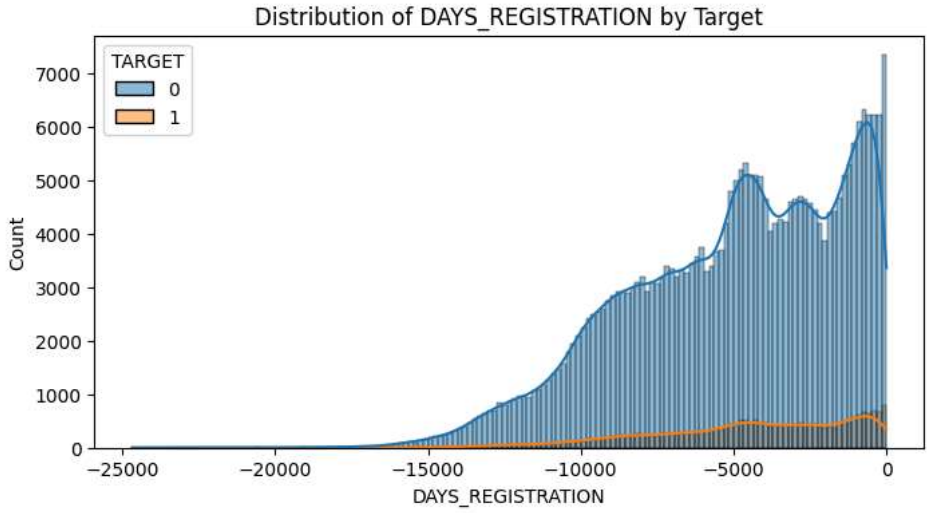

Distribution of AMT_ANNUITY by Target
TARGET
0
1
Count
AMT_ANNUITY

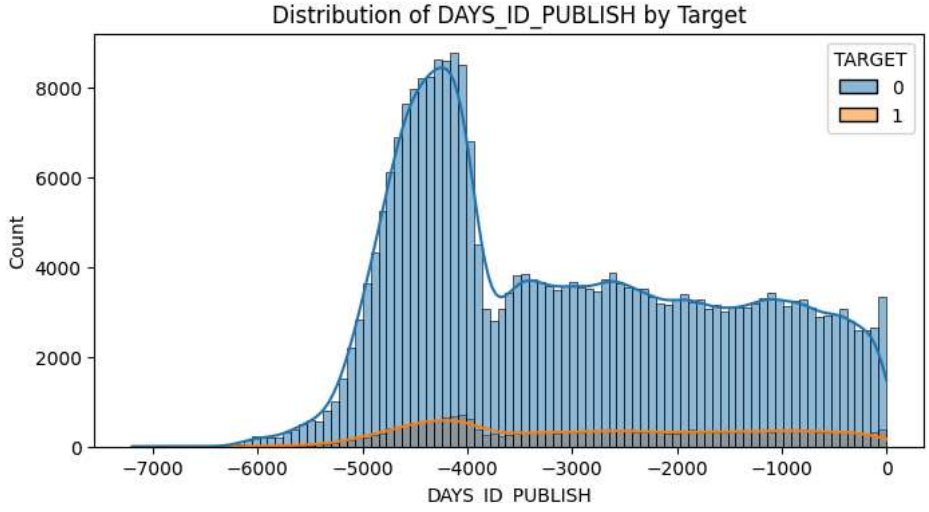

Distribution of AMT_GOODS_PRICE by Target
TARGET
0
1
Count
AMT_GOODS_PRICE
1e6


Distribution of REGION_POPULATION_RELATIVE by Target
TARGET
0
1
Count
REGION_POPULATION_RELATIVE


Distribution of DAYS_BIRTH by Target
TARGET
0
1
Count
DAYS_BIRTH


Distribution of DAYS_EMPLOYED by Target
TARGET
0
1
Count
DAYS_EMPLOYED


Distribution of DAYS_REGISTRATION by Target
TARGET
0
1
Count
DAYS_REGISTRATION


Distribution of DAYS_ID_PUBLISH by Target
TARGET
0
1
Count
DAYS_ID_PUBLISH