Introduction to the project

Project: Personal Loan Risk Analysis **Inter Name:** Liudmila Stolbetslaia

Understanding Credit Risk

Credit risk is the likelihood of financial loss that can occur if a borrower fails to repay a loan. It refers to the risk that a lender may not receive the owed principal and interest. Lenders can reduce credit risk by evaluating factors related to a borrower's creditworthiness, such as their existing debt and income.

When lenders provide mortgages, credit cards, or other types of loans, there is always the risk that the borrower may not repay the loan.

- Credit risk refers to the possibility of a lender losing money when lending funds to a borrower.
- Consumer credit risk can be evaluated using the five Cs: credit history, repayment capacity, capital, loan conditions, and the associated collateral.
- Borrowers considered to be higher credit risks are often charged higher interest rates on loans.
- Your credit score is one of the factors lenders use to determine the likelihood of you defaulting on a loan.

Bussines Understanding

Nowadays loand are widly used for differnt puposes. It is important to identify the capability of the aplicant to avoid thhe futer resk of finacila lose. Once the company recive the application every singl candiadte will be going throug the data analysis, which includes EDA, feature enginering and ML alpplication. It helps to identyfy if the applicant is alagible to hav it. The approval will be based on the aplicants profile.

If the aplicant is likely to repay the loan, than it is no risk for the company to provide the loan. If not, then approvinf the loan my lead to financial loss, so the candiadte beter to be rejected.

When a client applies for a loan, four possible decisions can be made by the client or the company:

- Approved: The company has approved the loan application.
- Cancelled: The client cancelled the application at some point during the approval process. This could be due to a change of mind or, in some cases, because the client was offered a loan with less favourable terms due to higher risk.
- Refused: The company rejected the loan application, typically because the client did not meet the necessary requirements.
- Unused Offer: The loan was cancelled by the client at a different stage of the process, even though it had been initially offered.

Project Objectives: To analyse financial data to identify potential credit risks by examining customer behaviour, application trends, and relevant financial indicators.

It will provide with undersatnding of driving factors behinf the loan defult. The analysis can be utilised by the banking service for risk assesmnt.

Goals of the progect:

- Identify key risk factors
- Minimise financial lose of the company
- Enhanced Risk Managemen
- Analyse customer behaviur
- Improve customer profile understanding

The data provided in 3 files as explained below:

- 'application_data.csv' contains all the information of the client at the time of application. The data is about whether a client has payment difficulties.
- 'previous_application.csv' contains information about the client's previous loan data. It contains the data whether the previous application had been Approved, Cancelled, Refused or Unused offer.
- 'columns_description.csv' is data dictionary which describes the meaning of the variables.

Tools: Python, EDA, Data Visualization, Feature Engineering, Statistical Analysis

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import LinearSegmentedColormap
```

```
#reach to the Google drive
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

PR Application

```
pr_application_data = pd.read_csv('/content/drive/MyDrive/Oeson/Python/Personla Loan/previous_application.csv')
```

Aplication

```
application_data = pd.read_csv('/content/drive/MyDrive/Oeson/Python/Personla Loan/application_data.csv')
application_data
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
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

307511 rows × 122 columns

```
application_data.info()
```

```
<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
```

```
application_data.describe()
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELA
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	307511.00
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	0.02
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	0.01
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	0.00
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	0.01
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	0.01
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	0.02
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.07

8 rows × 106 columns

```
list(application_data.columns)
```

```
       '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',
       'OBS_30_CNT_SOCIAL_CIRCLE',
       'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE',
       'DEF_60_CNT_SOCIAL_CIRCLE',
       'DAYS_LAST_PHONE_CHANGE',
       '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',
```

```
# List of columns to keep
columns_to_keep = [
    'SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
    'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
    'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
    'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
    'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'FLAG_MOBIL',
    'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_EMAIL', 'REGION_RATING_CLIENT',
    'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
    'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
    'REG_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
    "DAYS_LAST_PHONE_CHANGE"
]

# Filter the DataFrame to only keep the specified columns
application_data = application_data.filter(columns_to_keep)

# Optional: Check the updated DataFrame columns
print(application_data.columns)
```

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE',
       'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
       'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH',
       'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'FLAG_MOBIL', 'FLAG_EMP_PHONE',
```

```
       'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_EMAIL',
       'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE',
       'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
       'DAYS_LAST_PHONE_CHANGE'],
      dtype='object')
```

```
application_data
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
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

307511 rows × 41 columns

```
application_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 41 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   SK_ID_CURR                   307511 non-null  int64  
 1   TARGET                       307511 non-null  int64  
 2   NAME_CONTRACT_TYPE           307511 non-null  object 
 3   CODE_GENDER                  307511 non-null  object 
 4   FLAG_OWN_CAR                 307511 non-null  object 
 5   FLAG_OWN_REALTY              307511 non-null  object 
 6   CNT_CHILDREN                 307511 non-null  int64  
 7   AMT_INCOME_TOTAL             307511 non-null  float64
 8   AMT_CREDIT                   307511 non-null  float64
 9   AMT_ANNUITY                  307499 non-null  float64
 10  AMT_GOODS_PRICE              307233 non-null  float64
 11  NAME_TYPE_SUITE              306219 non-null  object 
 12  NAME_INCOME_TYPE             307511 non-null  object 
 13  NAME_EDUCATION_TYPE          307511 non-null  object 
 14  NAME_FAMILY_STATUS           307511 non-null  object 
 15  NAME_HOUSING_TYPE            307511 non-null  object 
 16  REGION_POPULATION_RELATIVE   307511 non-null  float64
 17  DAYS_BIRTH                   307511 non-null  int64  
 18  DAYS_EMPLOYED                307511 non-null  int64  
 19  DAYS_REGISTRATION            307511 non-null  float64
 20  DAYS_ID_PUBLISH              307511 non-null  int64  
 21  OCCUPATION_TYPE              211120 non-null  object 
 22  CNT_FAM_MEMBERS              307509 non-null  float64
 23  FLAG_MOBIL                   307511 non-null  int64  
 24  FLAG_EMP_PHONE               307511 non-null  int64  
 25  FLAG_WORK_PHONE              307511 non-null  int64  
 26  FLAG_CONT_MOBILE             307511 non-null  int64  
 27  FLAG_EMAIL                   307511 non-null  int64  
 28  REGION_RATING_CLIENT         307511 non-null  int64  
 29  REGION_RATING_CLIENT_W_CITY  307511 non-null  int64  
 30  WEEKDAY_APPR_PROCESS_START   307511 non-null  object 
 31  HOUR_APPR_PROCESS_START      307511 non-null  int64  
 32  REG_REGION_NOT_LIVE_REGION   307511 non-null  int64  
 33  REG_REGION_NOT_WORK_REGION   307511 non-null  int64  
 34  REG_CITY_NOT_LIVE_CITY       307511 non-null  int64  
```

```
 35  REG_CITY_NOT_WORK_CITY       307511 non-null  int64
 36  ORGANIZATION_TYPE            307511 non-null  object
 37  EXT_SOURCE_1                 134133 non-null  float64
 38  EXT_SOURCE_2                 306851 non-null  float64
 39  EXT_SOURCE_3                 246546 non-null  float64
 40  DAYS_LAST_PHONE_CHANGE       307510 non-null  float64
dtypes: float64(11), int64(18), object(12)
memory usage: 96.2+ MB
```

```
application_data.describe().T
```

	count	mean	std	min	25%	50%	75%	
SK_ID_CURR	307511.0	278180.518577	102790.175348	1.000020e+05	189145.500000	278202.000000	367142.500000	4.562550
TARGET	307511.0	0.080729	0.272419	0.000000e+00	0.000000	0.000000	0.000000	1.000000
CNT_CHILDREN	307511.0	0.417052	0.722121	0.000000e+00	0.000000	0.000000	1.000000	1.900000
AMT_INCOME_TOTAL	307511.0	168797.919297	237123.146279	2.565000e+04	112500.000000	147150.000000	202500.000000	1.170000
AMT_CREDIT	307511.0	599025.999706	402490.776996	4.500000e+04	270000.000000	513531.000000	808650.000000	4.050000
AMT_ANNUITY	307499.0	27108.573909	14493.737315	1.615500e+03	16524.000000	24903.000000	34596.000000	2.580255
AMT_GOODS_PRICE	307233.0	538396.207429	369446.460540	4.050000e+04	238500.000000	450000.000000	679500.000000	4.050000
REGION_POPULATION_RELATIVE	307511.0	0.020868	0.013831	2.900000e-04	0.010006	0.018850	0.028663	7.250800
DAYS_BIRTH	307511.0	-16036.995067	4363.988632	-2.522900e+04	-19682.000000	-15750.000000	-12413.000000	-7.489000
DAYS_EMPLOYED	307511.0	63815.045904	141275.766519	-1.791200e+04	-2760.000000	-1213.000000	-289.000000	3.652430
DAYS_REGISTRATION	307511.0	-4986.120328	3522.886321	-2.467200e+04	-7479.500000	-4504.000000	-2010.000000	0.000000
DAYS_ID_PUBLISH	307511.0	-2994.202373	1509.450419	-7.197000e+03	-4299.000000	-3254.000000	-1720.000000	0.000000
CNT_FAM_MEMBERS	307509.0	2.152665	0.910682	1.000000e+00	2.000000	2.000000	3.000000	2.000000
FLAG_MOBIL	307511.0	0.999997	0.001803	0.000000e+00	1.000000	1.000000	1.000000	1.000000
FLAG_EMP_PHONE	307511.0	0.819889	0.384280	0.000000e+00	1.000000	1.000000	1.000000	1.000000
FLAG_WORK_PHONE	307511.0	0.199368	0.399526	0.000000e+00	0.000000	0.000000	0.000000	1.000000
FLAG_CONT_MOBILE	307511.0	0.998133	0.043164	0.000000e+00	1.000000	1.000000	1.000000	1.000000
FLAG_EMAIL	307511.0	0.056720	0.231307	0.000000e+00	0.000000	0.000000	0.000000	1.000000
REGION_RATING_CLIENT	307511.0	2.052463	0.509034	1.000000e+00	2.000000	2.000000	2.000000	3.000000
REGION_RATING_CLIENT_W_CITY	307511.0	2.031521	0.502737	1.000000e+00	2.000000	2.000000	2.000000	3.000000
HOUR_APPR_PROCESS_START	307511.0	12.063419	3.265832	0.000000e+00	10.000000	12.000000	14.000000	2.300000
REG_REGION_NOT_LIVE_REGION	307511.0	0.015144	0.122126	0.000000e+00	0.000000	0.000000	0.000000	1.000000
REG_REGION_NOT_WORK_REGION	307511.0	0.050769	0.219526	0.000000e+00	0.000000	0.000000	0.000000	1.000000
REG_CITY_NOT_LIVE_CITY	307511.0	0.078173	0.268444	0.000000e+00	0.000000	0.000000	0.000000	1.000000
REG_CITY_NOT_WORK_CITY	307511.0	0.230454	0.421124	0.000000e+00	0.000000	0.000000	0.000000	1.000000
EXT_SOURCE_1	134133.0	0.502130	0.211062	1.456813e-02	0.334007	0.505998	0.675053	9.626928
EXT_SOURCE_2	306851.0	0.514393	0.191060	8.173617e-08	0.392457	0.565961	0.663617	8.549997
EXT_SOURCE_3	246546.0	0.510853	0.194844	5.272652e-04	0.370650	0.535276	0.669057	8.960095
DAYS_LAST_PHONE_CHANGE	307510.0	-962.858788	826.808487	-4.292000e+03	-1570.000000	-757.000000	-274.000000	0.000000

```
application_data.isnull().sum()
```

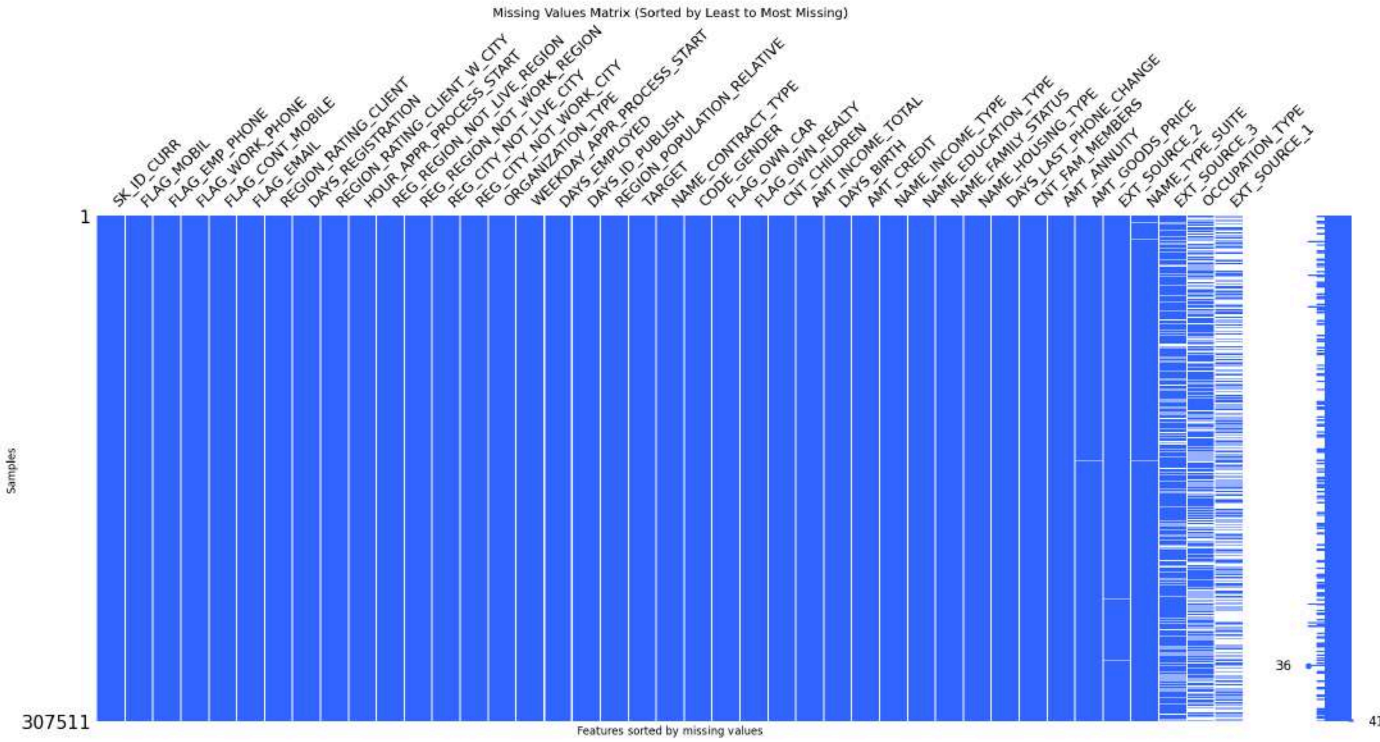
	0
SK_ID_CURR	0
TARGET	0
NAME_CONTRACT_TYPE	0
CODE_GENDER	0
FLAG_OWN_CAR	0
FLAG_OWN_REALTY	0
CNT_CHILDREN	0
AMT_INCOME_TOTAL	0
AMT_CREDIT	0
AMT_ANNUITY	12
AMT_GOODS_PRICE	278
NAME_TYPE_SUITE	1292
NAME_INCOME_TYPE	0
NAME_EDUCATION_TYPE	0
NAME_FAMILY_STATUS	0
NAME_HOUSING_TYPE	0
REGION_POPULATION_RELATIVE	0
DAYS_BIRTH	0
DAYS_EMPLOYED	0
DAYS_REGISTRATION	0
DAYS_ID_PUBLISH	0
OCCUPATION_TYPE	96391
CNT_FAM_MEMBERS	2
FLAG_MOBIL	0
FLAG_EMP_PHONE	0
FLAG_WORK_PHONE	0
FLAG_CONT_MOBILE	0
FLAG_EMAIL	0
REGION_RATING_CLIENT	0
REGION_RATING_CLIENT_W_CITY	0
WEEKDAY_APPR_PROCESS_START	0
HOUR_APPR_PROCESS_START	0
REG_REGION_NOT_LIVE_REGION	0
REG_REGION_NOT_WORK_REGION	0
REG_CITY_NOT_LIVE_CITY	0
REG_CITY_NOT_WORK_CITY	0
ORGANIZATION_TYPE	0
EXT_SOURCE_1	173378
EXT_SOURCE_2	660
EXT_SOURCE_3	60965
DAYS_LAST_PHONE_CHANGE	1

dtype: int64

```
import missingno as msno
```

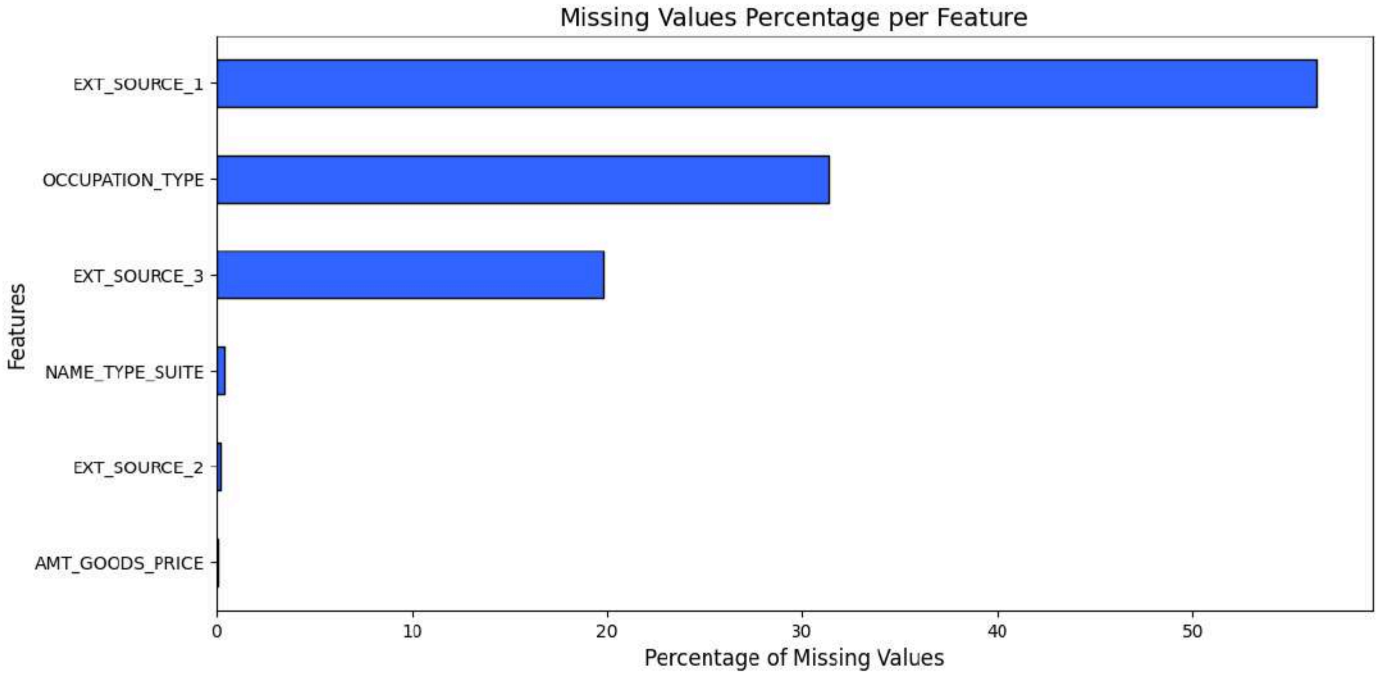
```
# Sort columns by missing values in ascending order
sorted_columns = application_data.isnull().sum().sort_values().index
sorted_data = application_data[sorted_columns]

# Plot missing values matrix with sorted features
plt.figure(figsize=(12, 6))
msno.matrix(sorted_data, color=(0.2, 0.4, 1))  # Blue color (RGB values)
plt.xlabel("Features sorted by missing values", fontsize=12)
plt.ylabel("Samples", fontsize=12)
plt.title("Missing Values Matrix (Sorted by Least to Most Missing)", fontsize=14)
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



Start coding or generate with AI.

```
missing_percent = (application_data.isnull().sum() * 100 / len(application_data)).round(2)

# Sort values in descending order (most missing first)
missing_percent = missing_percent[missing_percent > 0].sort_values(ascending=False)

# Plot the missing values as a horizontal bar chart
plt.figure(figsize=(12, 6))
missing_percent.plot(kind="barh", color=(0.2, 0.4, 1), edgecolor="black")

# Add labels and title
plt.xlabel("Percentage of Missing Values", fontsize=12)
plt.ylabel("Features", fontsize=12)
plt.title("Missing Values Percentage per Feature", fontsize=14)
plt.gca().invert_yaxis()  # Invert y-axis for better readability

# Show the plot
plt.show()
```



Null

I would like to identify the % of null values in the dataset.

```
#find the percentage of null values in each column, to determine what needs to be done as part of clean
(application_data.isnull().sum() * 100 / len(application_data)).round(2)
```

	0
SK_ID_CURR	0.00
TARGET	0.00
NAME_CONTRACT_TYPE	0.00
CODE_GENDER	0.00
FLAG_OWN_CAR	0.00
FLAG_OWN_REALTY	0.00
CNT_CHILDREN	0.00
AMT_INCOME_TOTAL	0.00
AMT_CREDIT	0.00
AMT_ANNUITY	0.00
AMT_GOODS_PRICE	0.09
NAME_TYPE_SUITE	0.42
NAME_INCOME_TYPE	0.00
NAME_EDUCATION_TYPE	0.00
NAME_FAMILY_STATUS	0.00
NAME_HOUSING_TYPE	0.00
REGION_POPULATION_RELATIVE	0.00
DAYS_BIRTH	0.00
DAYS_EMPLOYED	0.00
DAYS_REGISTRATION	0.00
DAYS_ID_PUBLISH	0.00
OCCUPATION_TYPE	31.35
CNT_FAM_MEMBERS	0.00
FLAG_MOBIL	0.00
FLAG_EMP_PHONE	0.00
FLAG_WORK_PHONE	0.00
FLAG_CONT_MOBILE	0.00
FLAG_EMAIL	0.00
REGION_RATING_CLIENT	0.00
REGION_RATING_CLIENT_W_CITY	0.00
WEEKDAY_APPR_PROCESS_START	0.00
HOUR_APPR_PROCESS_START	0.00
REG_REGION_NOT_LIVE_REGION	0.00
REG_REGION_NOT_WORK_REGION	0.00
REG_CITY_NOT_LIVE_CITY	0.00
REG_CITY_NOT_WORK_CITY	0.00
ORGANIZATION_TYPE	0.00
EXT_SOURCE_1	56.38
EXT_SOURCE_2	0.21
EXT_SOURCE_3	19.83
DAYS_LAST_PHONE_CHANGE	0.00

dtype: float64

```
application_data.shape
```

```
(307511, 41)
```

The chart displys top 5 features with null values in the datset.

When dealing with missing data, it's important to assess the extent of missingness in each column. Columns with a high percentage of missing values (e.g., over 50%) might be candidates for removal, as imputing them could introduce significant bias. I will remove EXT_SOURCE_2.

NAN Values.

```
application_data.describe()
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELA
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	307511.00
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	0.02
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	0.01
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	0.00
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	0.01
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	0.01
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	0.02
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.07

8 rows × 29 columns

```
application_data["OCCUPATION_TYPE"].unique()
```

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

```
# Forward fill missing values in 'OCCUPATION_TYPE' column in place
application_data['OCCUPATION_TYPE'].ffill(inplace=True)

# Print the updated DataFrame
application_data.head()
```

```
<ipython-input-141-feb2c434f840>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  application_data['OCCUPATION_TYPE'].ffill(inplace=True)
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_AI
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	2
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	3
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	2
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	2

5 rows × 41 columns

```
application_data['NAME_TYPE_SUITE'].unique()
```

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

```
#Here "Unaccompanied" data has the highest mode.We can fill missing values  with Unaccompanied
```

```
application_data["NAME_TYPE_SUITE"].fillna(application_data["NAME_TYPE_SUITE"].mode()[0],inplace=True)
```

```
<ipython-input-143-4e5a1f17f59f>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  application_data["NAME_TYPE_SUITE"].fillna(application_data["NAME_TYPE_SUITE"].mode()[0],inplace=True)
```

```
# Calculate the mean of 'EXT_SOURCE_3', excluding missing values
mean_value = application_data['EXT_SOURCE_3'].mean()

# Fill missing values with the calculated mean
application_data['EXT_SOURCE_3'].fillna(mean_value, inplace=True)

print(application_data['EXT_SOURCE_3'])
```

```
0         0.139376
1         0.510853
2         0.729567
3         0.510853
4         0.510853
            ...
307506    0.510853
307507    0.510853
307508    0.218859
307509    0.661024
307510    0.113922
Name: EXT_SOURCE_3, Length: 307511, dtype: float64
<ipython-input-144-5f57734ddb06>:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  application_data['EXT_SOURCE_3'].fillna(mean_value, inplace=True)
```

```
# Calculate the mean of 'EXT_SOURCE_2, excluding missing values
mean_value = application_data['EXT_SOURCE_2'].mean()

# Fill missing values with the calculated mean
application_data['EXT_SOURCE_2'].fillna(mean_value, inplace=True)

print(application_data['EXT_SOURCE_2'])
```

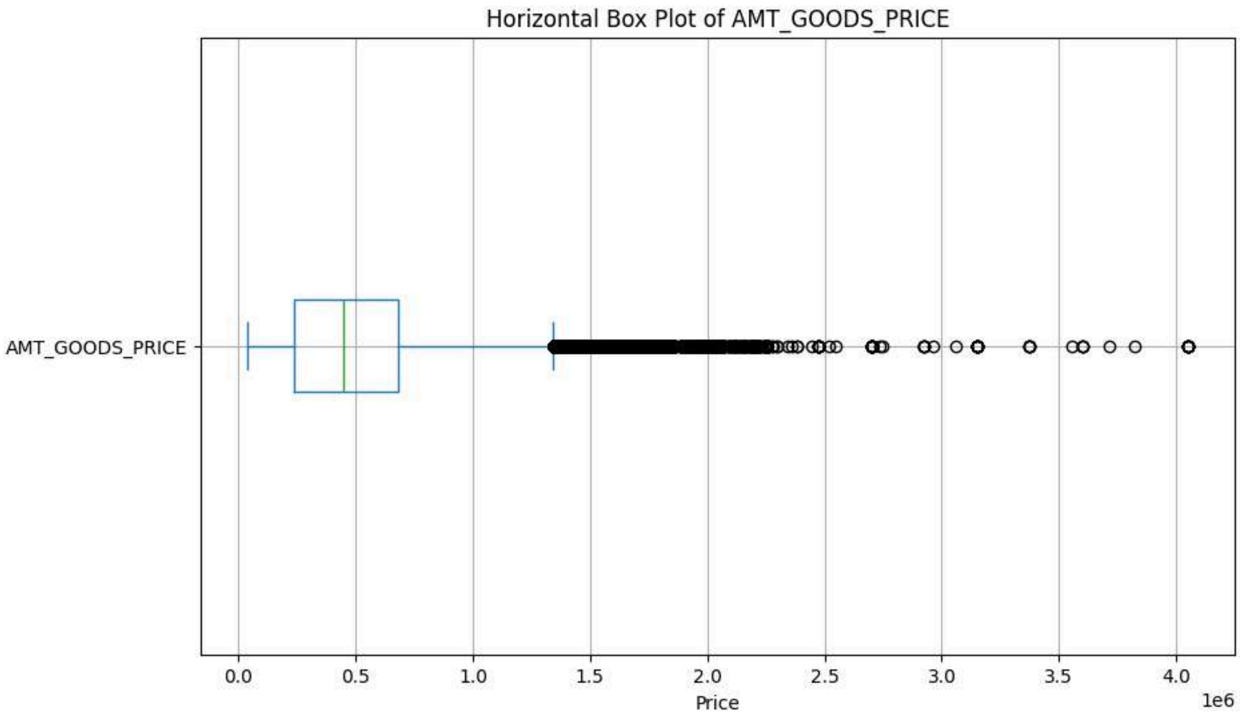
```
0         0.262949
1         0.622246
2         0.555912
3         0.650442
4         0.322738
            ...
307506    0.681632
307507    0.115992
307508    0.535722
307509    0.514163
307510    0.708569
Name: EXT_SOURCE_2, Length: 307511, dtype: float64
<ipython-input-145-4c40e94be3cf>:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  application_data['EXT_SOURCE_2'].fillna(mean_value, inplace=True)
```

```
application_data['AMT_GOODS_PRICE'].describe()
```

	AMT_GOODS_PRICE
count	3.072330e+05
mean	5.383962e+05
std	3.694465e+05
min	4.050000e+04
25%	2.385000e+05
50%	4.500000e+05
75%	6.795000e+05
max	4.050000e+06

dtype: float64

```
import pandas as pd
import matplotlib.pyplot as plt

# Generate a horizontal box plot for the 'AMT_GOODS_PRICE' column
plt.figure(figsize=(10, 6))
application_data['AMT_GOODS_PRICE'].plot.box(vert=False)
plt.title('Horizontal Box Plot of AMT_GOODS_PRICE')
plt.xlabel('Price')
plt.grid(True)
plt.show()
```




```
# Calculate the median of the 'AMT_GOODS_PRICE' column
median_value = application_data['AMT_GOODS_PRICE'].median()

# Replace missing values in 'AMT_GOODS_PRICE' with the median
application_data['AMT_GOODS_PRICE'].fillna(median_value, inplace=True)
```

```
application_data.isnull().sum()
```

	0
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	0
NAME_TYPE_SUITE	0
NAME_INCOME_TYPE	0
NAME_EDUCATION_TYPE	0
NAME_FAMILY_STATUS	0
NAME_HOUSING_TYPE	0
REGION_POPULATION_RELATIVE	0
DAYS_BIRTH	0
DAYS_EMPLOYED	0
DAYS_REGISTRATION	0
DAYS_ID_PUBLISH	0
OCCUPATION_TYPE	0
CNT_FAM_MEMBERS	2
FLAG_MOBIL	0
FLAG_EMP_PHONE	0
FLAG_WORK_PHONE	0
FLAG_CONT_MOBILE	0
FLAG_EMAIL	0
REGION_RATING_CLIENT	0
REGION_RATING_CLIENT_W_CITY	0
WEEKDAY_APPR_PROCESS_START	0
HOUR_APPR_PROCESS_START	0
REG_REGION_NOT_LIVE_REGION	0
REG_REGION_NOT_WORK_REGION	0
REG_CITY_NOT_LIVE_CITY	0
REG_CITY_NOT_WORK_CITY	0
ORGANIZATION_TYPE	0
EXT_SOURCE_1	173378
EXT_SOURCE_2	0
EXT_SOURCE_3	0
DAYS_LAST_PHONE_CHANGE	1

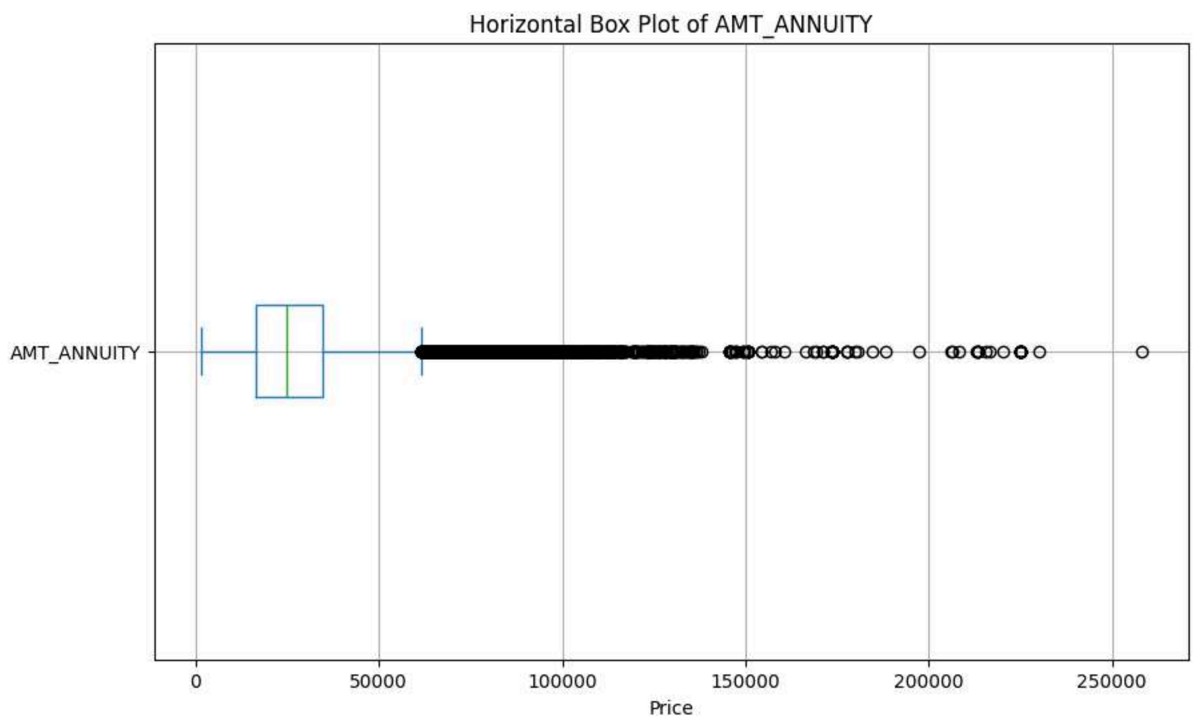
dtype: int64

```
application_data['AMT_ANNUITY'].describe()
```

	AMT_ANNUITY
count	307499.000000
mean	27108.573909
std	14493.737315
min	1615.500000
25%	16524.000000
50%	24903.000000
75%	34596.000000
max	258025.500000

dtype: float64

```
# Generate a horizontal box plot for the 'AMT_ANNUITY'' column
plt.figure(figsize=(10, 6))
application_data['AMT_ANNUITY'].plot.box(vert=False)
plt.title('Horizontal Box Plot of AMT_ANNUITY')
plt.xlabel('Price')
plt.grid(True)
plt.show()
```




```
mean_value = application_data['AMT_ANNUITY'].mean()

# Replace missing values with the mean
application_data['AMT_ANNUITY'].fillna(mean_value, inplace=True)
```

```
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	AMT_AI
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	2
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	3
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	2
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	2

5 rows × 41 columns

```
# Forward fill missing values in 'CNT_FAM_MEMBERS' column
application_data['CNT_FAM_MEMBERS'].fillna(method='ffill', inplace=True)
application_data.head()
```

```
<ipython-input-154-c9109c7a579b>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  application_data['CNT_FAM_MEMBERS'].fillna(method='ffill', inplace=True)
<ipython-input-154-c9109c7a579b>:2: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use obj
  application_data['CNT_FAM_MEMBERS'].fillna(method='ffill', inplace=True)
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_AI
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	2
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	3
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	2
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	2

5 rows × 41 columns

```
application_data['DAYS_LAST_PHONE_CHANGE'].fillna((application_data['DAYS_LAST_PHONE_CHANGE'].mean()), inplace=True)
```

```
<ipython-input-155-645ed047a4c5>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  application_data['DAYS_LAST_PHONE_CHANGE'].fillna((application_data['DAYS_LAST_PHONE_CHANGE'].mean()), inplace=True)
```

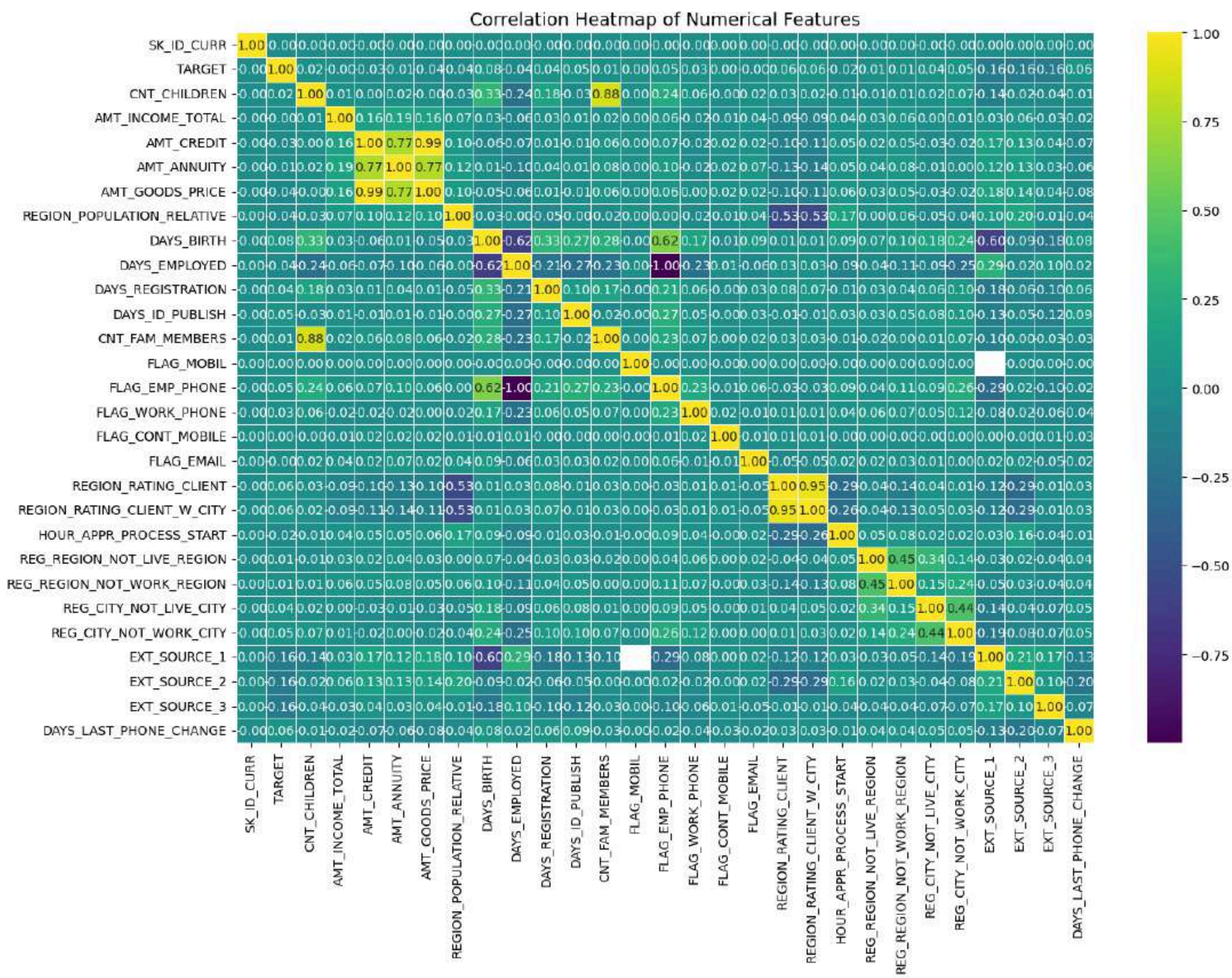
Feature selection technique. Correlation Matrix

When analyzing data, it's important to understand that traditional correlation matrices mainly identify linear relationships between variables. However, many real-world datasets have non-linear associations that linear correlation measures might miss.

A correlation matrix can effectively highlight linear redundancies, it may not detect non-linear dependencies. To comprehensively identify and address all forms of redundancy, it's advisable to complement the correlation matrix with additional methods that capture non-linear relationships, ensuring a more nuanced understanding of variable interactions.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Select only numerical columns
numeric_data = application_data.select_dtypes(include=['number'])

# Compute the correlation matrix
corr_matrix = numeric_data.corr()

# Set up the matplotlib figure with increased size
plt.figure(figsize=(15, 10))
```

```
# Create the heatmap
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="viridis", linewidths=0.5, cbar=True)

# Add title
plt.title("Correlation Heatmap of Numerical Features", fontsize=14)

# Show the plot
plt.show()
```



AMT_ANNUITY and AMT_CREDIT - re highly correlated, with a correlation coefficient of 0.77. This strong correlation is expected, as the monthly payment (AMT_ANNUITY) is directly influenced by the total credit amount (AMT_CREDIT), the interest rate, and the loan term.

A 99% correlation between AMT_GOODS_PRICE and AMT_CREDIT indicates that the total credit amount is almost entirely determined by the price of the goods being financed. This strong relationship suggests that the bank sets the loan amount based almost entirely on the price of the items being purchased. Given this near-perfect correlation, it's advisable to remove one of these variables from your model to reduce redundancy and potential multicollinearity

A high correlation of 88% between CNT_FAM_MEMBERS (number of family members) and CNT_CHILDREN (number of children) indicates that these two variables are closely related. This strong relationship suggests that the total number of family members is largely determined by the number of children in the family. Given this high correlation, it's advisable to consider removing one of these variables from your model to reduce redundancy and potential multicollinearity.

In summary, addressing multicollinearity by removing one of the correlated variables can enhance your model's performance and stability.

I will remove some clumns form the dataset such us EXT_SOURCE_1, AMT_ANNUITY, AMT_GOODS_PRICE, CNT_CHILDREN

```
# List of columns to drop
columns_to_drop = ['EXT_SOURCE_1', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'CNT_CHILDREN', 'HOUR_APPR_PROCESS_START']

# Drop the specified columns from the DataFrame
application_data = application_data.drop(columns=columns_to_drop)

# Display the first few rows of the modified DataFrame to confirm the changes
application_data.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	AMT_INCOME_TOTAL	AMT_CREDIT	NAME_TYPE_SUITE	NAI
0	100002	1	Cash loans	M	N	Y	202500.0	406597.5	Unaccompanied	
1	100003	0	Cash loans	F	N	N	270000.0	1293502.5	Family	
2	100004	0	Revolving loans	M	Y	Y	67500.0	135000.0	Unaccompanied	
3	100006	0	Cash loans	F	N	Y	135000.0	312682.5	Unaccompanied	
4	100007	0	Cash loans	M	N	Y	121500.0	513000.0	Unaccompanied	

5 rows × 36 columns

```
application_data.shape
```

```
(307511, 36)
```

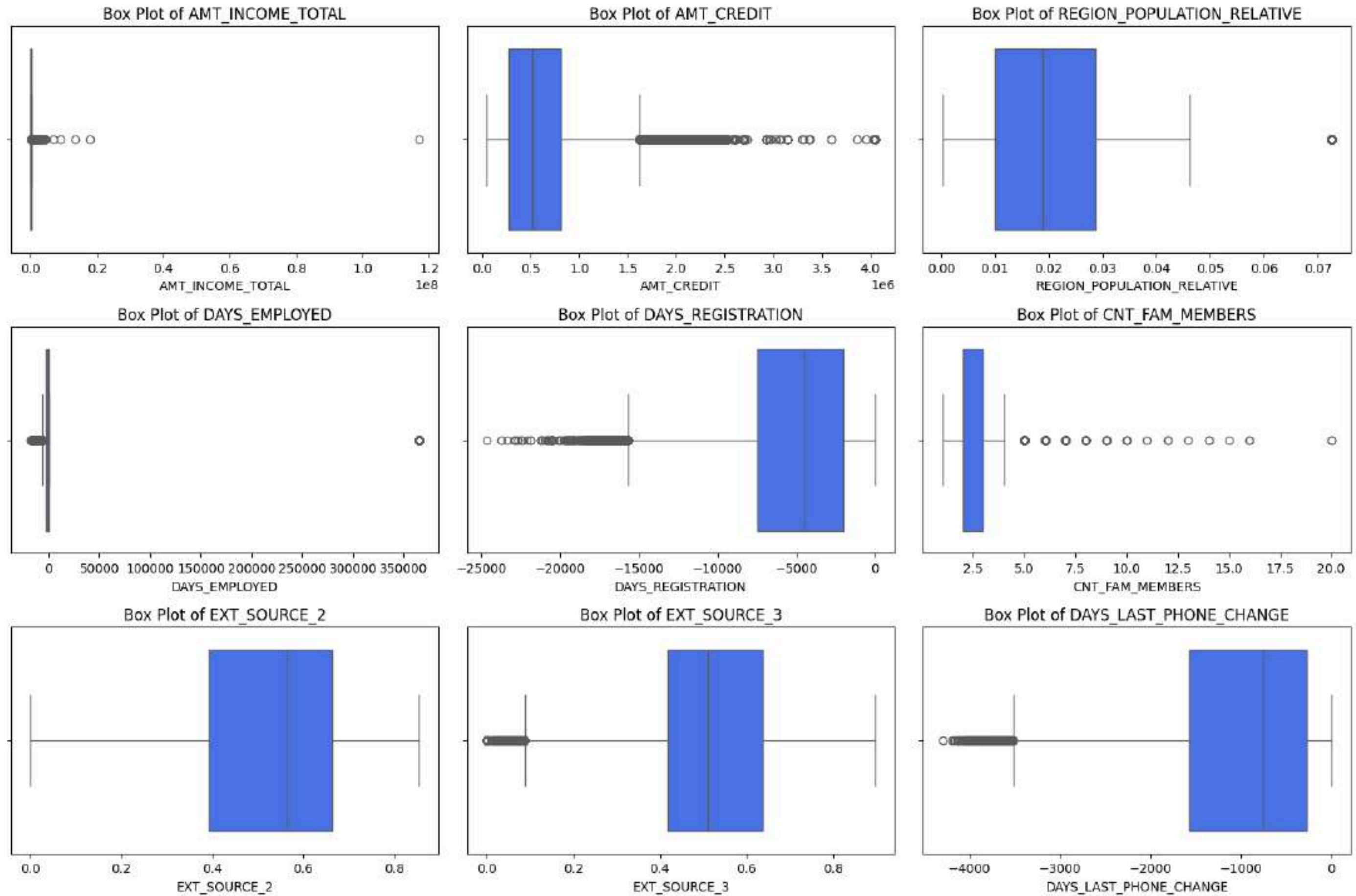
The final result we have as a tible with 37 columns.

Outliers could see that some columns have outliers. Let's find out about other features in the dataframe.

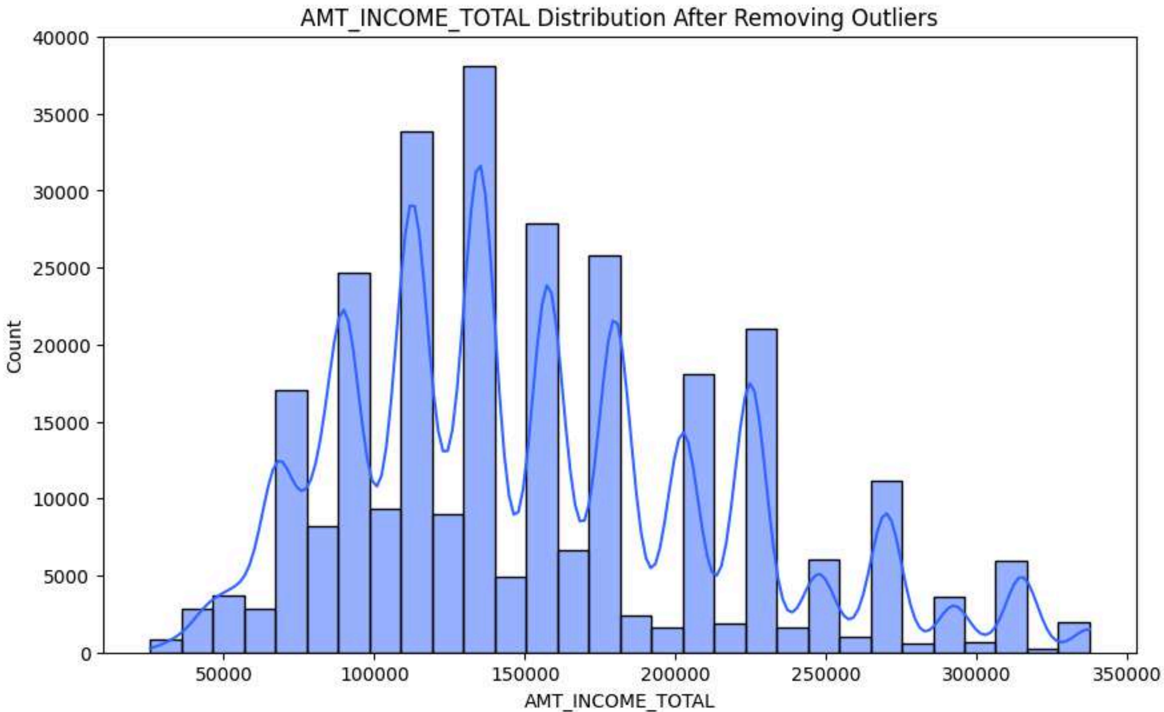
```
application_data.describe().T
```

	count	mean	std	min	25%	50%	75%	
SK_ID_CURR	307511.0	278180.518577	102790.175348	1.000020e+05	189145.500000	278202.000000	367142.500000	4.562550
TARGET	307511.0	0.080729	0.272419	0.000000e+00	0.000000	0.000000	0.000000	1.000000
AMT_INCOME_TOTAL	307511.0	168797.919297	237123.146279	2.565000e+04	112500.000000	147150.000000	202500.000000	1.170000
AMT_CREDIT	307511.0	599025.999706	402490.776996	4.500000e+04	270000.000000	513531.000000	808650.000000	4.050000
REGION_POPULATION_RELATIVE	307511.0	0.020868	0.013831	2.900000e-04	0.010006	0.018850	0.028663	7.25080C
DAYS_BIRTH	307511.0	-16036.995067	4363.988632	-2.522900e+04	-19682.000000	-15750.000000	-12413.000000	-7.489000
DAYS_EMPLOYED	307511.0	63815.045904	141275.766519	-1.791200e+04	-2760.000000	-1213.000000	-289.000000	3.652430
DAYS_REGISTRATION	307511.0	-4986.120328	3522.886321	-2.467200e+04	-7479.500000	-4504.000000	-2010.000000	0.000000
DAYS_ID_PUBLISH	307511.0	-2994.202373	1509.450419	-7.197000e+03	-4299.000000	-3254.000000	-1720.000000	0.000000
CNT_FAM_MEMBERS	307511.0	2.152664	0.910679	1.000000e+00	2.000000	2.000000	3.000000	2.000000
FLAG_MOBIL	307511.0	0.999997	0.001803	0.000000e+00	1.000000	1.000000	1.000000	1.000000
FLAG_EMP_PHONE	307511.0	0.819889	0.384280	0.000000e+00	1.000000	1.000000	1.000000	1.000000
FLAG_WORK_PHONE	307511.0	0.199368	0.399526	0.000000e+00	0.000000	0.000000	0.000000	1.000000
FLAG_CONT_MOBILE	307511.0	0.998133	0.043164	0.000000e+00	1.000000	1.000000	1.000000	1.000000
FLAG_EMAIL	307511.0	0.056720	0.231307	0.000000e+00	0.000000	0.000000	0.000000	1.000000
REGION_RATING_CLIENT	307511.0	2.052463	0.509034	1.000000e+00	2.000000	2.000000	2.000000	3.000000
REGION_RATING_CLIENT_W_CITY	307511.0	2.031521	0.502737	1.000000e+00	2.000000	2.000000	2.000000	3.000000
REG_REGION_NOT_LIVE_REGION	307511.0	0.015144	0.122126	0.000000e+00	0.000000	0.000000	0.000000	1.000000
REG_REGION_NOT_WORK_REGION	307511.0	0.050769	0.219526	0.000000e+00	0.000000	0.000000	0.000000	1.000000
REG_CITY_NOT_LIVE_CITY	307511.0	0.078173	0.268444	0.000000e+00	0.000000	0.000000	0.000000	1.000000
REG_CITY_NOT_WORK_CITY	307511.0	0.230454	0.421124	0.000000e+00	0.000000	0.000000	0.000000	1.000000
EXT_SOURCE_2	307511.0	0.514393	0.190855	8.173617e-08	0.392974	0.565467	0.663422	8.549997
EXT_SOURCE_3	307511.0	0.510853	0.174464	5.272652e-04	0.417100	0.510853	0.636376	8.960095
DAYS_LAST_PHONE_CHANGE	307511.0	-962.858788	826.807143	-4.292000e+03	-1570.000000	-757.000000	-274.000000	0.000000

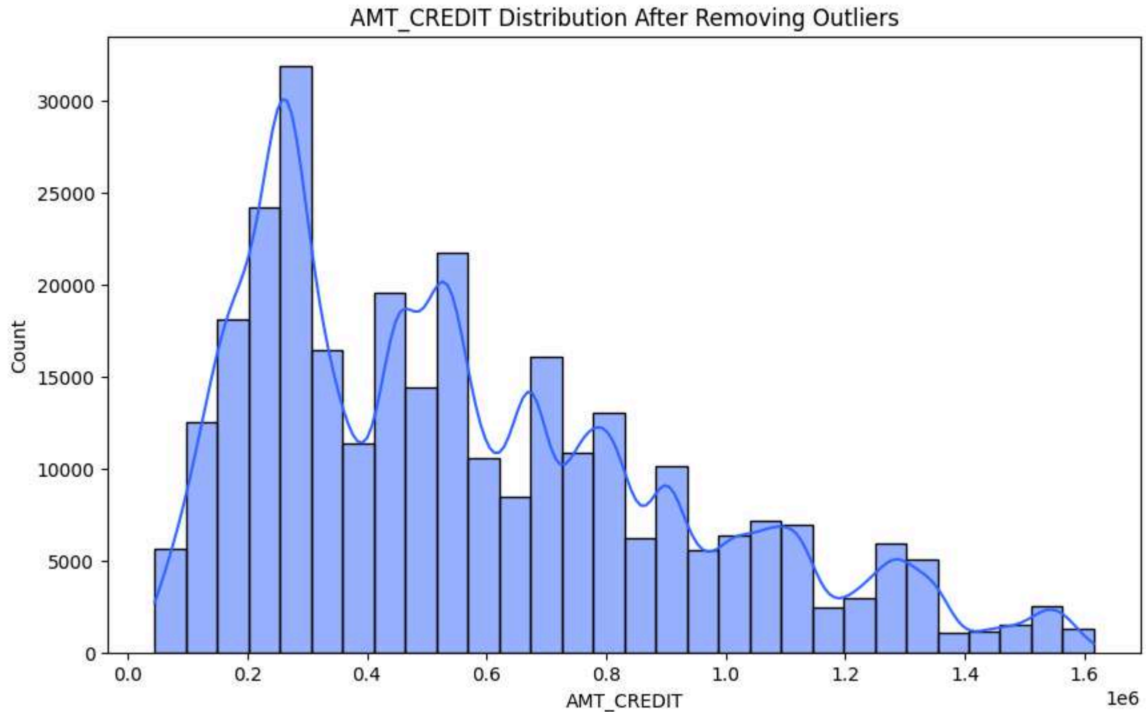
```
# List of features to plot
features = [
    'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'REGION_POPULATION_RELATIVE',
    'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'CNT_FAM_MEMBERS',
    'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_LAST_PHONE_CHANGE'
]

# Set the size of the figure
plt.figure(figsize=(15, 10))

# Create a box plot for each feature
for i, feature in enumerate(features, 1):
    plt.subplot(3, 3, i)  # 3 rows, 3 columns, position i
    sns.boxplot(x=application_data[feature], color=(0.2, 0.4, 1))
    plt.title(f'Box Plot of {feature}')
    plt.xlabel(feature)

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



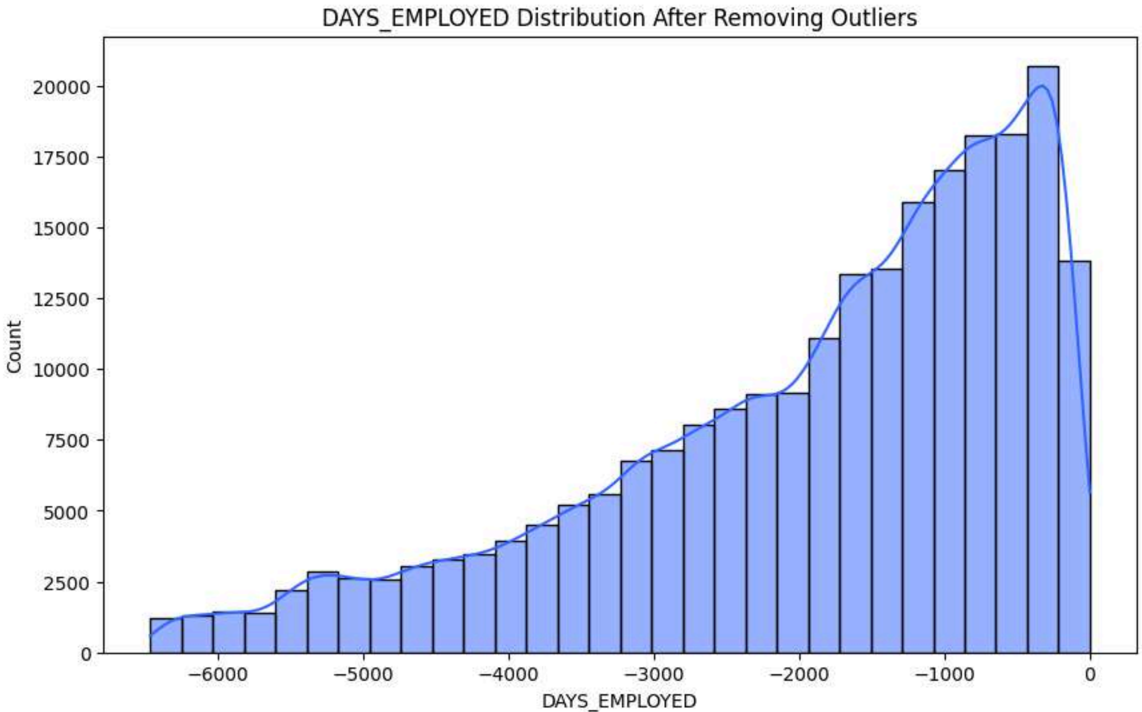

```
# Calculate Q1, Q3, and IQR
Q1 = application_data["AMT_INCOME_TOTAL"].quantile(0.25)
Q3 = application_data["AMT_INCOME_TOTAL"].quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers outside IQR range
application_data_cleaned = application_data[
    (application_data["AMT_INCOME_TOTAL"] >= lower_bound) &
    (application_data["AMT_INCOME_TOTAL"] <= upper_bound)
]

# Plot the cleaned data
f, ax = plt.subplots(figsize=(10,6))
sns.histplot(application_data_cleaned["AMT_INCOME_TOTAL"], kde=True, color=(0.2, 0.4, 1), bins=30)
plt.title("AMT_INCOME_TOTAL Distribution After Removing Outliers")
plt.show()
```




```
#q1=application_data["AMT_INCOME_TOTAL"].quantile(0.95)
#application_data["AMT_INCOME_TOTAL"] =  application_data.AMT_INCOME_TOTAL.apply(lambda x: q1 if x>q1 else x)

#f, ax = plt.subplots(figsize=(10,6))
#outlier_plot_1 = sns.distplot(application_data["AMT_INCOME_TOTAL"], color=(0.2, 0.4, 1))


# Calculate Q1, Q3, and IQR
Q1 = application_data["AMT_CREDIT"].quantile(0.25)
Q3 = application_data["AMT_CREDIT"].quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR


#Remove outliers outside IQR range
application_data_cleaned = application_data[
    (application_data["AMT_CREDIT"] >= lower_bound) &
    (application_data["AMT_CREDIT"] <= upper_bound)
]

# Plot the cleaned data
f, ax = plt.subplots(figsize=(10,6))
sns.histplot(application_data_cleaned["AMT_CREDIT"], kde=True, color=(0.2, 0.4, 1), bins=30)
plt.title("AMT_CREDIT Distribution After Removing Outliers")
plt.show()
```




```
#q2=application_data["AMT_CREDIT"].quantile(0.75)
#application_data["AMT_CREDIT"] =  application_data.AMT_CREDIT.apply(lambda x: q2 if x>q2 else x)

#f, ax = plt.subplots(figsize=(10,6))
#outlier_plot_2 = sns.distplot(application_data["AMT_CREDIT"],color=(0.2, 0.4, 1))


# Calculate Q1, Q3, and IQR
Q1 = application_data["DAYS_EMPLOYED"].quantile(0.25)
Q3 = application_data["DAYS_EMPLOYED"].quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers outside IQR range
application_data_cleaned = application_data[
    (application_data["DAYS_EMPLOYED"] >= lower_bound) &
    (application_data["DAYS_EMPLOYED"] <= upper_bound)
]

# Plot the cleaned data
f, ax = plt.subplots(figsize=(10,6))
sns.histplot(application_data_cleaned["DAYS_EMPLOYED"], kde=True, color=(0.2, 0.4, 1), bins=30)
plt.title("DAYS_EMPLOYED Distribution After Removing Outliers")
plt.show()
```



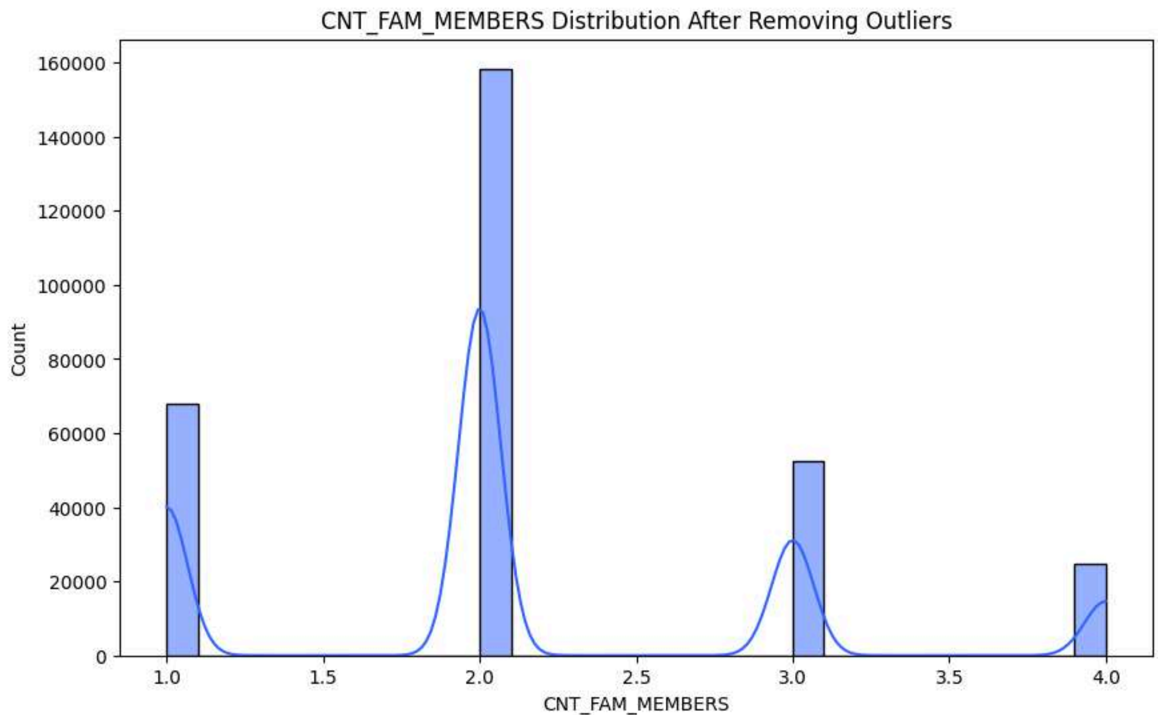

```
application_data['CNT_FAM_MEMBERS'].value_counts()
```

CNT_FAM_MEMBERS	count
2.0	158359
1.0	67847
3.0	52601
4.0	24697
5.0	3478
6.0	408
7.0	81
8.0	20
9.0	6
10.0	3
14.0	2
12.0	2
20.0	2
16.0	2
13.0	1
15.0	1
11.0	1

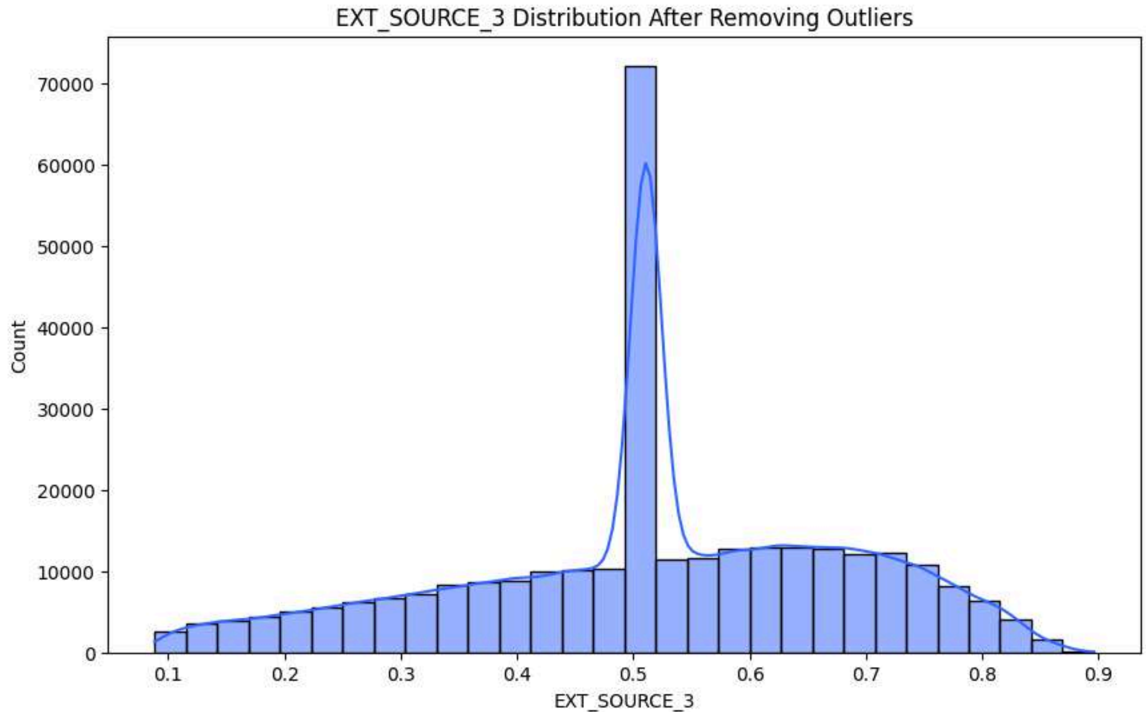
dtype: int64

```
# Calculate Q1, Q3, and IQR
Q1 = application_data["CNT_FAM_MEMBERS"].quantile(0.25)
Q3 = application_data["CNT_FAM_MEMBERS"].quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```
# Remove outliers outside IQR range
application_data_cleaned = application_data[
    (application_data["CNT_FAM_MEMBERS"] >= lower_bound) &
    (application_data["CNT_FAM_MEMBERS"] <= upper_bound)
]

# Plot the cleaned data
f, ax = plt.subplots(figsize=(10,6))
sns.histplot(application_data_cleaned["CNT_FAM_MEMBERS"], kde=True, color=(0.2, 0.4, 1), bins=30)
plt.title("CNT_FAM_MEMBERS Distribution After Removing Outliers")
plt.show()
```




```
# Calculate Q1, Q3, and IQR
Q1 = application_data["EXT_SOURCE_3"].quantile(0.25)
Q3 = application_data["EXT_SOURCE_3"].quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR


# Remove outliers outside IQR range
application_data_cleaned = application_data[
    (application_data["EXT_SOURCE_3"] >= lower_bound) &
    (application_data["EXT_SOURCE_3"] <= upper_bound)
]

# Plot the cleaned data
f, ax = plt.subplots(figsize=(10,6))
sns.histplot(application_data_cleaned["EXT_SOURCE_3"], kde=True, color=(0.2, 0.4, 1), bins=30)
plt.title("EXT_SOURCE_3 Distribution After Removing Outliers")
plt.show()
```




```
application_data.DAYS_LAST_PHONE_CHANGE.value_counts()
```

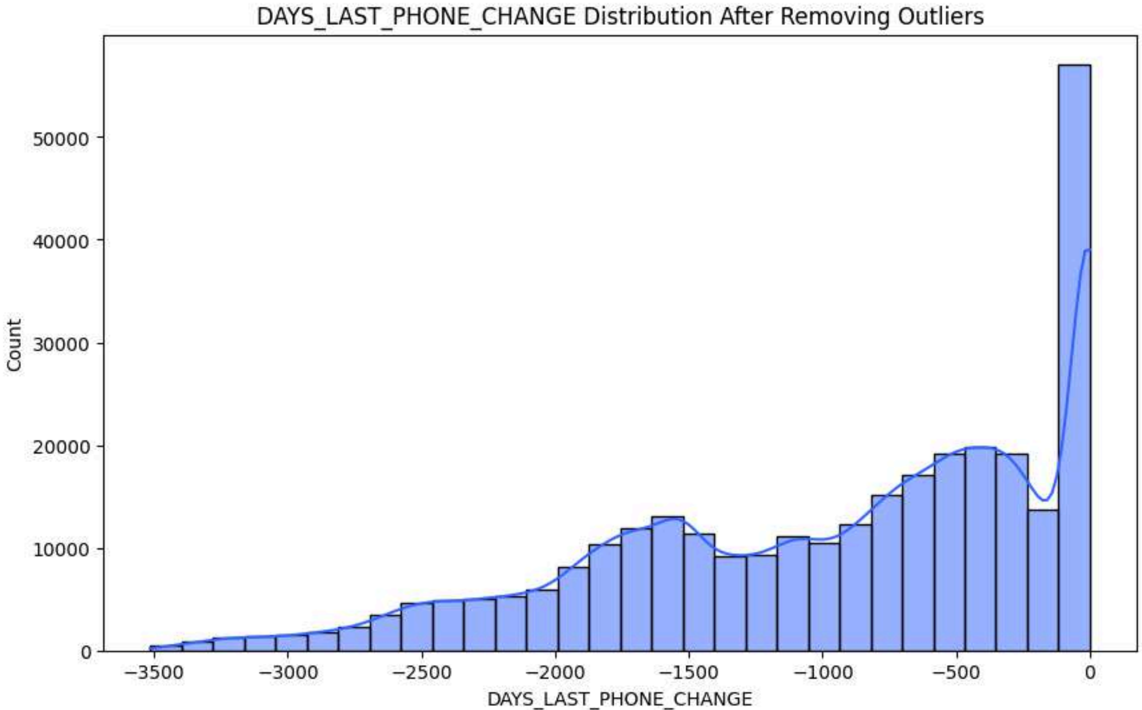
DAYS_LAST_PHONE_CHANGE	count
0.0	37672
-1.0	2812
-2.0	2318
-3.0	1763
-4.0	1285
...	...
-3713.0	1
-3978.0	1
-3983.0	1
-3739.0	1
-3538.0	1

3774 rows × 1 columns

dtype: int64

```
# Calculate Q1, Q3, and IQR
Q1 = application_data["DAYS_LAST_PHONE_CHANGE"].quantile(0.25)
Q3 = application_data["DAYS_LAST_PHONE_CHANGE"].quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers outside IQR range
application_data_cleaned = application_data[
    (application_data["DAYS_LAST_PHONE_CHANGE"] >= lower_bound) &
    (application_data["DAYS_LAST_PHONE_CHANGE"] <= upper_bound)
]

# Plot the cleaned data
f, ax = plt.subplots(figsize=(10,6))
sns.histplot(application_data_cleaned["DAYS_LAST_PHONE_CHANGE"], kde=True, color=(0.2, 0.4, 1), bins=30)
```

```
plt.title("DAYS_LAST_PHONE_CHANGE Distribution After Removing Outliers")
plt.show()
```



I will also convert date of birth to age. It will be more suitable for data analysis.

```
# Converting DAYS_BIRTH to AGE
application_data["AGE"] = application_data.DAYS_BIRTH.apply(lambda x :round(abs(x)/365),0)
application_data["AGE"]
application_data["AGE"] = pd.to_numeric(application_data["AGE"])
```

```
<ipython-input-172-d08887f79b0e>:2: FutureWarning: the convert_dtype parameter is deprecated and will be removed in a future version.  D
  application_data["AGE"] = application_data.DAYS_BIRTH.apply(lambda x :round(abs(x)/365),0)
```

```
application_data.AMT_INCOME_TOTAL.quantile([0.25,0.5,0.85,1])
```

	AMT_INCOME_TOTAL
0.25	112500.0
0.50	147150.0
0.85	234000.0
1.00	117000000.0

dtype: float64

Binning Salary Amount to Categories

Binning, also known as discretization, is the process of converting continuous numerical data into discrete categories or bins. This technique is particularly useful for simplifying models, enhancing interpretability, and managing outliers.

```
# Based on the Quantile Values , segregating the values to its respective categories
def salary_category_func(x):
    if x<337500 and x>=234000:
        return('HIGH')
    elif x<234000 and x>=147150:
        return('MODERATE')
    elif x<147150 and x>=112500:
        return('LOW')
    else:
        return("EXTREMLY LOW")
application_data["SALARY_CATEGORY"] = application_data.AMT_INCOME_TOTAL.apply(salary_category_func)
application_data["SALARY_CATEGORY"]
```

	SALARY_CATEGORY
0	MODERATE
1	HIGH
2	EXTREMLY LOW
3	LOW
4	LOW
...	...
307506	MODERATE
307507	EXTREMLY LOW
307508	MODERATE
307509	MODERATE
307510	MODERATE

307511 rows × 1 columns

dtype: object

```
application_data['SALARY_CATEGORY'].value_counts()
```

SALARY_CATEGORY	count
MODERATE	106966
EXTREMLY LOW	85384
LOW	84194
HIGH	30967

dtype: int64

```
application_data.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	AMT_INCOME_TOTAL	AMT_CREDIT	NAME_TYPE_SUITE	NAI
0	100002	1	Cash loans	M	N	Y	202500.0	406597.5	Unaccompanied	
1	100003	0	Cash loans	F	N	N	270000.0	1293502.5	Family	
2	100004	0	Revolving loans	M	Y	Y	67500.0	135000.0	Unaccompanied	
3	100006	0	Cash loans	F	N	Y	135000.0	312682.5	Unaccompanied	
4	100007	0	Cash loans	M	N	Y	121500.0	513000.0	Unaccompanied	

5 rows × 38 columns

We have 38 columns to work with from the previous application data. Based on the column descriptions and data feature importance techniques, I have removed columns that do not play an important role in the application and provide redundant data. Data selection plays a crucial role in data prediction when we apply machine learning modeling. The selection of features could change based on the machine learning performance. Null values have been identified, visualized, and replaced. I have removed the outliers from several features such as

`AMT_INCOME_TOTAL`, `AMT_CREDIT`, `DAYS_EMPLOYED`, `DAYS_REGISTRATION`, `CNT_FAM_MEMBERS`, `EXT_SOURCE_3`, and `DAYS_LAST_PHONE_CHANGE`. The target variable was visualized with a box plot, and we can see the clear difference between approved and denied applications.It is aslo says that we are dealing with imballanced dataset.

Peviuos Applicatiom

Accessing previouse application table. It containes data from the priviuse application. This data is useful in loan aplication as it shows if the prevouse applications were susessful.

To effectively analyse loan applications, it's essential to integrate current application data with previous application records. The SK_ID_CURR identifier serves as a unique key for each applicant, enabling us to merge these datasets accurately.

Combining Current and Previous Application Data Using SK_ID_CURR:

By merging the current applications with their corresponding previous applications on the SK_ID_CURR key, we can gain a comprehensive view of each applicant's history. This holistic perspective is invaluable for assessing creditworthiness and predicting loan repayment capabilities.

```
pr_application_data.head()
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WEEKDAY_APPR_
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

```
list(pr_application_data.columns)
```

```
['SK_ID_PREV',
 'SK_ID_CURR',
 'NAME_CONTRACT_TYPE',
 'AMT_ANNUITY',
 'AMT_APPLICATION',
 'AMT_CREDIT',
 'AMT_DOWN_PAYMENT',
 'AMT_GOODS_PRICE',
 'WEEKDAY_APPR_PROCESS_START',
 'HOUR_APPR_PROCESS_START',
 'FLAG_LAST_APPL_PER_CONTRACT',
 'NFLAG_LAST_APPL_IN_DAY',
 'RATE_DOWN_PAYMENT',
 'RATE_INTEREST_PRIMARY',
 'RATE_INTEREST_PRIVILEGED',
 'NAME_CASH_LOAN_PURPOSE',
 'NAME_CONTRACT_STATUS',
 'DAYS_DECISION',
 'NAME_PAYMENT_TYPE',
 'CODE_REJECT_REASON',
 'NAME_TYPE_SUITE',
 'NAME_CLIENT_TYPE',
 'NAME_GOODS_CATEGORY',
 'NAME_PORTFOLIO',
 'NAME_PRODUCT_TYPE',
 'CHANNEL_TYPE',
 'SELLERPLACE_AREA',
 'NAME_SELLER_INDUSTRY',
 'CNT_PAYMENT',
 'NAME_YIELD_GROUP',
 'PRODUCT_COMBINATION',
 'DAYS_FIRST_DRAWING',
 'DAYS_FIRST_DUE',
 'DAYS_LAST_DUE_1ST_VERSION',
 'DAYS_LAST_DUE',
 'DAYS_TERMINATION',
 'NFLAG_INSURED_ON_APPROVAL']
```

```
pr_application_data.shape
```

```
pr_application_data.shape
```

```
(1670214, 37)
```

```
pr_application_data.dtypes
```

	0
SK_ID_PREV	int64
SK_ID_CURR	int64
NAME_CONTRACT_TYPE	object
AMT_ANNUITY	float64
AMT_APPLICATION	float64
AMT_CREDIT	float64
AMT_DOWN_PAYMENT	float64
AMT_GOODS_PRICE	float64
WEEKDAY_APPR_PROCESS_START	object
HOUR_APPR_PROCESS_START	int64
FLAG_LAST_APPL_PER_CONTRACT	object
NFLAG_LAST_APPL_IN_DAY	int64
RATE_DOWN_PAYMENT	float64
RATE_INTEREST_PRIMARY	float64
RATE_INTEREST_PRIVILEGED	float64
NAME_CASH_LOAN_PURPOSE	object
NAME_CONTRACT_STATUS	object
DAYS_DECISION	int64
NAME_PAYMENT_TYPE	object
CODE_REJECT_REASON	object
NAME_TYPE_SUITE	object
NAME_CLIENT_TYPE	object
NAME_GOODS_CATEGORY	object
NAME_PORTFOLIO	object
NAME_PRODUCT_TYPE	object
CHANNEL_TYPE	object
SELLERPLACE_AREA	int64
NAME_SELLER_INDUSTRY	object
CNT_PAYMENT	float64
NAME_YIELD_GROUP	object
PRODUCT_COMBINATION	object
DAYS_FIRST_DRAWING	float64
DAYS_FIRST_DUE	float64
DAYS_LAST_DUE_1ST_VERSION	float64
DAYS_LAST_DUE	float64
DAYS_TERMINATION	float64
NFLAG_INSURED_ON_APPROVAL	float64

dtype: object

```
pr_application_data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
SK_ID_PREV	1670214.0	1.923089e+06	532597.958696	1.000001e+06	1.461857e+06	1.923110e+06	2.384280e+06	2845382.000
SK_ID_CURR	1670214.0	2.783572e+05	102814.823849	1.000010e+05	1.893290e+05	2.787145e+05	3.675140e+05	456255.000
AMT_ANNUITY	1297979.0	1.595512e+04	14782.137335	0.000000e+00	6.321780e+03	1.125000e+04	2.065842e+04	418058.145
AMT_APPLICATION	1670214.0	1.752339e+05	292779.762387	0.000000e+00	1.872000e+04	7.104600e+04	1.803600e+05	6905160.000
AMT_CREDIT	1670213.0	1.961140e+05	318574.616546	0.000000e+00	2.416050e+04	8.054100e+04	2.164185e+05	6905160.000
AMT_DOWN_PAYMENT	774370.0	6.697402e+03	20921.495410	-9.000000e-01	0.000000e+00	1.638000e+03	7.740000e+03	3060045.000
AMT_GOODS_PRICE	1284699.0	2.278473e+05	315396.557937	0.000000e+00	5.084100e+04	1.123200e+05	2.340000e+05	6905160.000
HOUR_APPR_PROCESS_START	1670214.0	1.248418e+01	3.334028	0.000000e+00	1.000000e+01	1.200000e+01	1.500000e+01	23.000
NFLAG_LAST_APPL_IN_DAY	1670214.0	9.964675e-01	0.059330	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000
RATE_DOWN_PAYMENT	774370.0	7.963682e-02	0.107823	-1.497876e-05	0.000000e+00	5.160508e-02	1.089091e-01	1.000
RATE_INTEREST_PRIMARY	5951.0	1.883569e-01	0.087671	3.478125e-02	1.607163e-01	1.891222e-01	1.933299e-01	1.000
RATE_INTEREST_PRIVILEGED	5951.0	7.735025e-01	0.100879	3.731501e-01	7.156448e-01	8.350951e-01	8.525370e-01	1.000
DAYS_DECISION	1670214.0	-8.806797e+02	779.099667	-2.922000e+03	-1.300000e+03	-5.810000e+02	-2.800000e+02	-1.000
SELLERPLACE_AREA	1670214.0	3.139511e+02	7127.443459	-1.000000e+00	-1.000000e+00	3.000000e+00	8.200000e+01	4000000.000
CNT_PAYMENT	1297984.0	1.605408e+01	14.567288	0.000000e+00	6.000000e+00	1.200000e+01	2.400000e+01	84.000
DAYS_FIRST_DRAWING	997149.0	3.422099e+05	88916.115834	-2.922000e+03	3.652430e+05	3.652430e+05	3.652430e+05	365243.000
DAYS_FIRST_DUE	997149.0	1.382627e+04	72444.869708	-2.892000e+03	-1.628000e+03	-8.310000e+02	-4.110000e+02	365243.000
DAYS_LAST_DUE_1ST_VERSION	997149.0	3.376777e+04	106857.034789	-2.801000e+03	-1.242000e+03	-3.610000e+02	1.290000e+02	365243.000
DAYS_LAST_DUE	997149.0	7.658240e+04	149647.415123	-2.889000e+03	-1.314000e+03	-5.370000e+02	-7.400000e+01	365243.000
DAYS_TERMINATION	997149.0	8.199234e+04	153303.516729	-2.874000e+03	-1.270000e+03	-4.990000e+02	-4.400000e+01	365243.000
NFLAG_INSURED_ON_APPROVAL	997149.0	3.325702e-01	0.471134	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000

```
pr_application_data.isnull().sum()
```

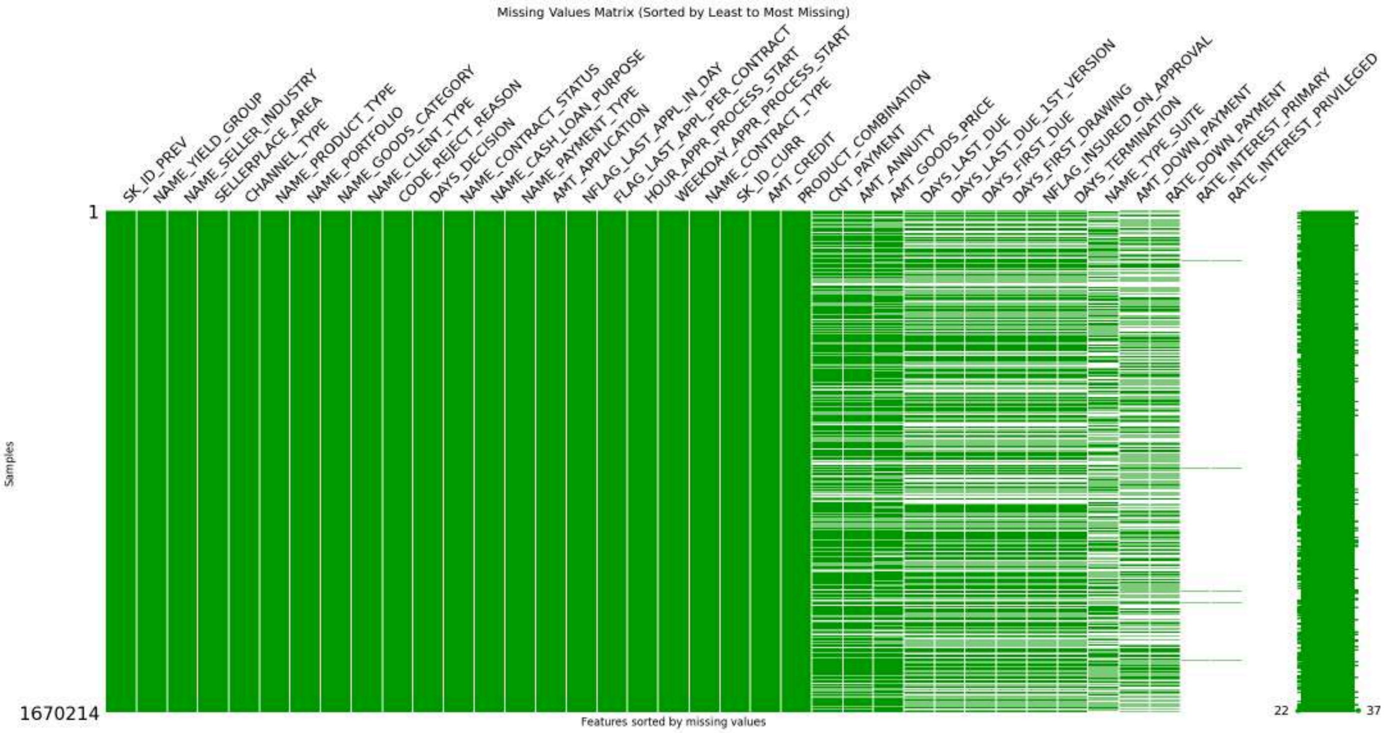
	0
SK_ID_PREV	0
SK_ID_CURR	0
NAME_CONTRACT_TYPE	0
AMT_ANNUITY	372235
AMT_APPLICATION	0
AMT_CREDIT	1
AMT_DOWN_PAYMENT	895844
AMT_GOODS_PRICE	385515
WEEKDAY_APPR_PROCESS_START	0
HOUR_APPR_PROCESS_START	0
FLAG_LAST_APPL_PER_CONTRACT	0
NFLAG_LAST_APPL_IN_DAY	0
RATE_DOWN_PAYMENT	895844
RATE_INTEREST_PRIMARY	1664263
RATE_INTEREST_PRIVILEGED	1664263
NAME_CASH_LOAN_PURPOSE	0
NAME_CONTRACT_STATUS	0
DAYS_DECISION	0
NAME_PAYMENT_TYPE	0
CODE_REJECT_REASON	0
NAME_TYPE_SUITE	820405
NAME_CLIENT_TYPE	0
NAME_GOODS_CATEGORY	0
NAME_PORTFOLIO	0
NAME_PRODUCT_TYPE	0
CHANNEL_TYPE	0
SELLERPLACE_AREA	0
NAME_SELLER_INDUSTRY	0
CNT_PAYMENT	372230
NAME_YIELD_GROUP	0
PRODUCT_COMBINATION	346
DAYS_FIRST_DRAWING	673065
DAYS_FIRST_DUE	673065
DAYS_LAST_DUE_1ST_VERSION	673065
DAYS_LAST_DUE	673065
DAYS_TERMINATION	673065
NFLAG_INSURED_ON_APPROVAL	673065

dtype: int64

```
# Sort columns by missing values in ascending order
sorted_columns = pr_application_data.isnull().sum().sort_values().index
sorted_data = pr_application_data[sorted_columns]

# Plot missing values matrix
plt.figure(figsize=(12, 6))
msno.matrix(sorted_data, color=(0, 0.6, 0))

# Set labels and title
plt.xlabel("Features sorted by missing values", fontsize=12)
```

```
plt.ylabel("Samples", fontsize=12)
plt.title("Missing Values Matrix (Sorted by Least to Most Missing)", fontsize=14)

# Show the plot
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



Start coding or generate with AI.

```
#find the percentage of null values in each column, to determine what needs to be done as part of clean
(pr_application_data.isnull().sum() * 100 / len(pr_application_data)).round(2)
```

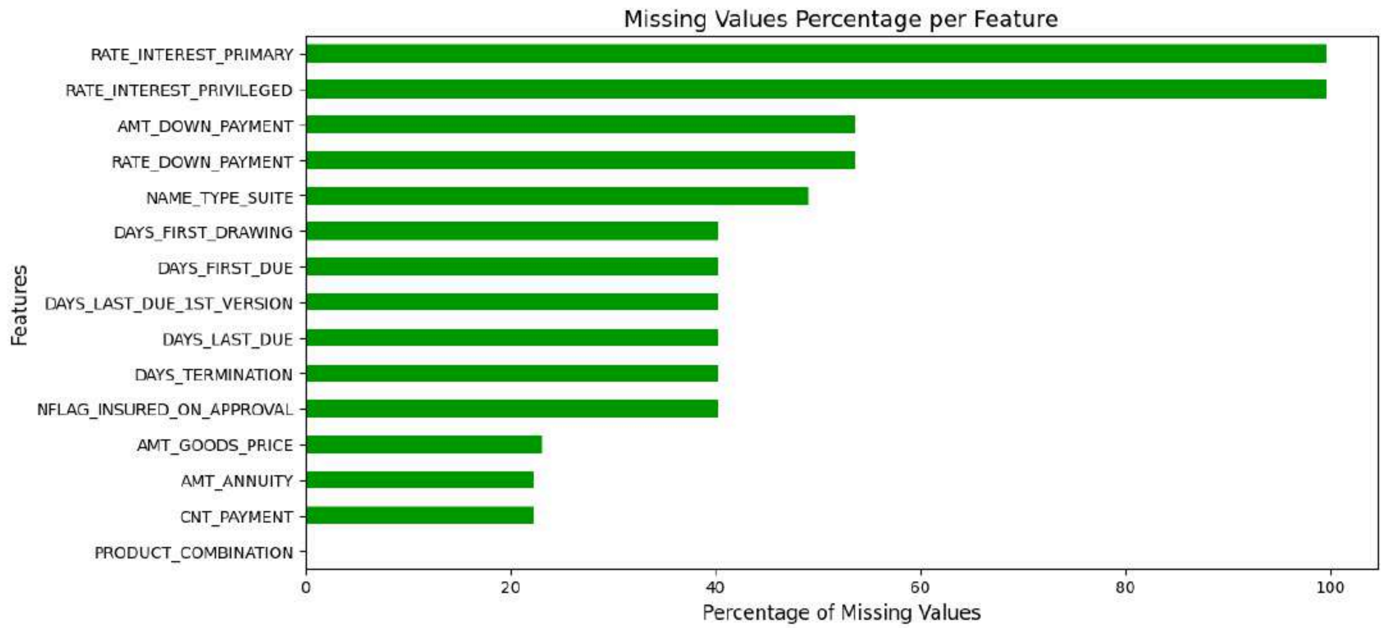
	0
SK_ID_PREV	0.00
SK_ID_CURR	0.00
NAME_CONTRACT_TYPE	0.00
AMT_ANNUITY	22.29
AMT_APPLICATION	0.00
AMT_CREDIT	0.00
AMT_DOWN_PAYMENT	53.64
AMT_GOODS_PRICE	23.08
WEEKDAY_APPR_PROCESS_START	0.00
HOUR_APPR_PROCESS_START	0.00
FLAG_LAST_APPL_PER_CONTRACT	0.00
NFLAG_LAST_APPL_IN_DAY	0.00
RATE_DOWN_PAYMENT	53.64
RATE_INTEREST_PRIMARY	99.64
RATE_INTEREST_PRIVILEGED	99.64
NAME_CASH_LOAN_PURPOSE	0.00
NAME_CONTRACT_STATUS	0.00
DAYS_DECISION	0.00
NAME_PAYMENT_TYPE	0.00
CODE_REJECT_REASON	0.00
NAME_TYPE_SUITE	49.12
NAME_CLIENT_TYPE	0.00
NAME_GOODS_CATEGORY	0.00
NAME_PORTFOLIO	0.00
NAME_PRODUCT_TYPE	0.00
CHANNEL_TYPE	0.00
SELLERPLACE_AREA	0.00
NAME_SELLER_INDUSTRY	0.00
CNT_PAYMENT	22.29
NAME_YIELD_GROUP	0.00
PRODUCT_COMBINATION	0.02
DAYS_FIRST_DRAWING	40.30
DAYS_FIRST_DUE	40.30
DAYS_LAST_DUE_1ST_VERSION	40.30
DAYS_LAST_DUE	40.30
DAYS_TERMINATION	40.30
NFLAG_INSURED_ON_APPROVAL	40.30

dtype: float64

Start coding or generate with AI.

```
missing_percent = (pr_application_data.isnull().sum() * 100 / len(pr_application_data)).round(2)

# Sort values in descending order (most missing first)
missing_percent = missing_percent[missing_percent > 0].sort_values(ascending=False)

# Plot the missing values as a horizontal bar chart
plt.figure(figsize=(12, 6))
missing_percent.plot(kind="barh", color=(0, 0.6, 0))
```

```
# Add labels and title
plt.xlabel("Percentage of Missing Values", fontsize=12)
plt.ylabel("Features", fontsize=12)
plt.title("Missing Values Percentage per Feature", fontsize=14)
plt.gca().invert_yaxis()  # Invert y-axis for better readability

# Show the plot
plt.show()
```



Double-click (or enter) to edit

```
pr_application_data['NAME_TYPE_SUITE'].ffill(inplace=True)
pr_application_data['PRODUCT_COMBINATION'].ffill(inplace=True)
pr_application_data['NFLAG_INSURED_ON_APPROVAL'].ffill(inplace=True)
pr_application_data['AMT_CREDIT'].ffill(inplace=True)
```

```
<ipython-input-186-30f54df68a22>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  pr_application_data['NAME_TYPE_SUITE'].ffill(inplace=True)
<ipython-input-186-30f54df68a22>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  pr_application_data['PRODUCT_COMBINATION'].ffill(inplace=True)
<ipython-input-186-30f54df68a22>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  pr_application_data['NFLAG_INSURED_ON_APPROVAL'].ffill(inplace=True)
<ipython-input-186-30f54df68a22>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  pr_application_data['AMT_CREDIT'].ffill(inplace=True)
```

```
pr_application_data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
SK_ID_PREV	1670214.0	1.923089e+06	532597.958696	1.000001e+06	1.461857e+06	1.923110e+06	2.384280e+06	2845382.000
SK_ID_CURR	1670214.0	2.783572e+05	102814.823849	1.000010e+05	1.893290e+05	2.787145e+05	3.675140e+05	456255.000
AMT_ANNUITY	1297979.0	1.595512e+04	14782.137335	0.000000e+00	6.321780e+03	1.125000e+04	2.065842e+04	418058.145
AMT_APPLICATION	1670214.0	1.752339e+05	292779.762387	0.000000e+00	1.872000e+04	7.104600e+04	1.803600e+05	6905160.000
AMT_CREDIT	1670214.0	1.961139e+05	318574.544121	0.000000e+00	2.416050e+04	8.054100e+04	2.164185e+05	6905160.000
AMT_DOWN_PAYMENT	774370.0	6.697402e+03	20921.495410	-9.000000e-01	0.000000e+00	1.638000e+03	7.740000e+03	3060045.000
AMT_GOODS_PRICE	1284699.0	2.278473e+05	315396.557937	0.000000e+00	5.084100e+04	1.123200e+05	2.340000e+05	6905160.000
HOUR_APPR_PROCESS_START	1670214.0	1.248418e+01	3.334028	0.000000e+00	1.000000e+01	1.200000e+01	1.500000e+01	23.000
NFLAG_LAST_APPL_IN_DAY	1670214.0	9.964675e-01	0.059330	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000
RATE_DOWN_PAYMENT	774370.0	7.963682e-02	0.107823	-1.497876e-05	0.000000e+00	5.160508e-02	1.089091e-01	1.000
RATE_INTEREST_PRIMARY	5951.0	1.883569e-01	0.087671	3.478125e-02	1.607163e-01	1.891222e-01	1.933299e-01	1.000
RATE_INTEREST_PRIVILEGED	5951.0	7.735025e-01	0.100879	3.731501e-01	7.156448e-01	8.350951e-01	8.525370e-01	1.000
DAYS_DECISION	1670214.0	-8.806797e+02	779.099667	-2.922000e+03	-1.300000e+03	-5.810000e+02	-2.800000e+02	-1.000
SELLERPLACE_AREA	1670214.0	3.139511e+02	7127.443459	-1.000000e+00	-1.000000e+00	3.000000e+00	8.200000e+01	4000000.000
CNT_PAYMENT	1297984.0	1.605408e+01	14.567288	0.000000e+00	6.000000e+00	1.200000e+01	2.400000e+01	84.000
DAYS_FIRST_DRAWING	997149.0	3.422099e+05	88916.115834	-2.922000e+03	3.652430e+05	3.652430e+05	3.652430e+05	365243.000
DAYS_FIRST_DUE	997149.0	1.382627e+04	72444.869708	-2.892000e+03	-1.628000e+03	-8.310000e+02	-4.110000e+02	365243.000
DAYS_LAST_DUE_1ST_VERSION	997149.0	3.376777e+04	106857.034789	-2.801000e+03	-1.242000e+03	-3.610000e+02	1.290000e+02	365243.000
DAYS_LAST_DUE	997149.0	7.658240e+04	149647.415123	-2.889000e+03	-1.314000e+03	-5.370000e+02	-7.400000e+01	365243.000
DAYS_TERMINATION	997149.0	8.199234e+04	153303.516729	-2.874000e+03	-1.270000e+03	-4.990000e+02	-4.400000e+01	365243.000
NFLAG_INSURED_ON_APPROVAL	1670214.0	3.846124e-01	0.486504	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000

```
# List of columns to process
columns_of_interest = ['DAYS_FIRST_DRAWING', 'DAYS_LAST_DUE_1ST_VERSION', 'AMT_ANNUITY','CNT_PAYMENT']

# Replace NaN values with the mean of each column
pr_application_data[columns_of_interest] = pr_application_data[columns_of_interest].fillna(pr_application_data[columns_of_interest].mean())

pr_application_data.head()
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WEEKDAY_APPR_
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

```
# List of columns to process
columns_of_interest_1 = ['AMT_DOWN_PAYMENT', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'AMT_GOODS_PRICE', 'RATE_DOWN_PAYMENT']

# Replace NaN values with the median of each specified column
pr_application_data[columns_of_interest_1] = pr_application_data[columns_of_interest_1].fillna(pr_application_data[columns_of_interest_1].me

pr_application_data.head()
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WEEKDAY_APPR_
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	1638.0	607500.0	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	1638.0	112500.0	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	1638.0	450000.0	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	1638.0	337500.0	

5 rows × 37 columns

```
pr_application_data.isnull().sum()
```

	0
SK_ID_PREV	0
SK_ID_CURR	0
NAME_CONTRACT_TYPE	0
AMT_ANNUITY	0
AMT_APPLICATION	0
AMT_CREDIT	0
AMT_DOWN_PAYMENT	0
AMT_GOODS_PRICE	0
WEEKDAY_APPR_PROCESS_START	0
HOUR_APPR_PROCESS_START	0
FLAG_LAST_APPL_PER_CONTRACT	0
NFLAG_LAST_APPL_IN_DAY	0
RATE_DOWN_PAYMENT	0
RATE_INTEREST_PRIMARY	1664263
RATE_INTEREST_PRIVILEGED	1664263
NAME_CASH_LOAN_PURPOSE	0
NAME_CONTRACT_STATUS	0
DAYS_DECISION	0
NAME_PAYMENT_TYPE	0
CODE_REJECT_REASON	0
NAME_TYPE_SUITE	1
NAME_CLIENT_TYPE	0
NAME_GOODS_CATEGORY	0
NAME_PORTFOLIO	0
NAME_PRODUCT_TYPE	0
CHANNEL_TYPE	0
SELLERPLACE_AREA	0
NAME_SELLER_INDUSTRY	0
CNT_PAYMENT	0
NAME_YIELD_GROUP	0
PRODUCT_COMBINATION	0
DAYS_FIRST_DRAWING	0
DAYS_FIRST_DUE	0
DAYS_LAST_DUE_1ST_VERSION	0
DAYS_LAST_DUE	0
DAYS_TERMINATION	0
NFLAG_INSURED_ON_APPROVAL	0

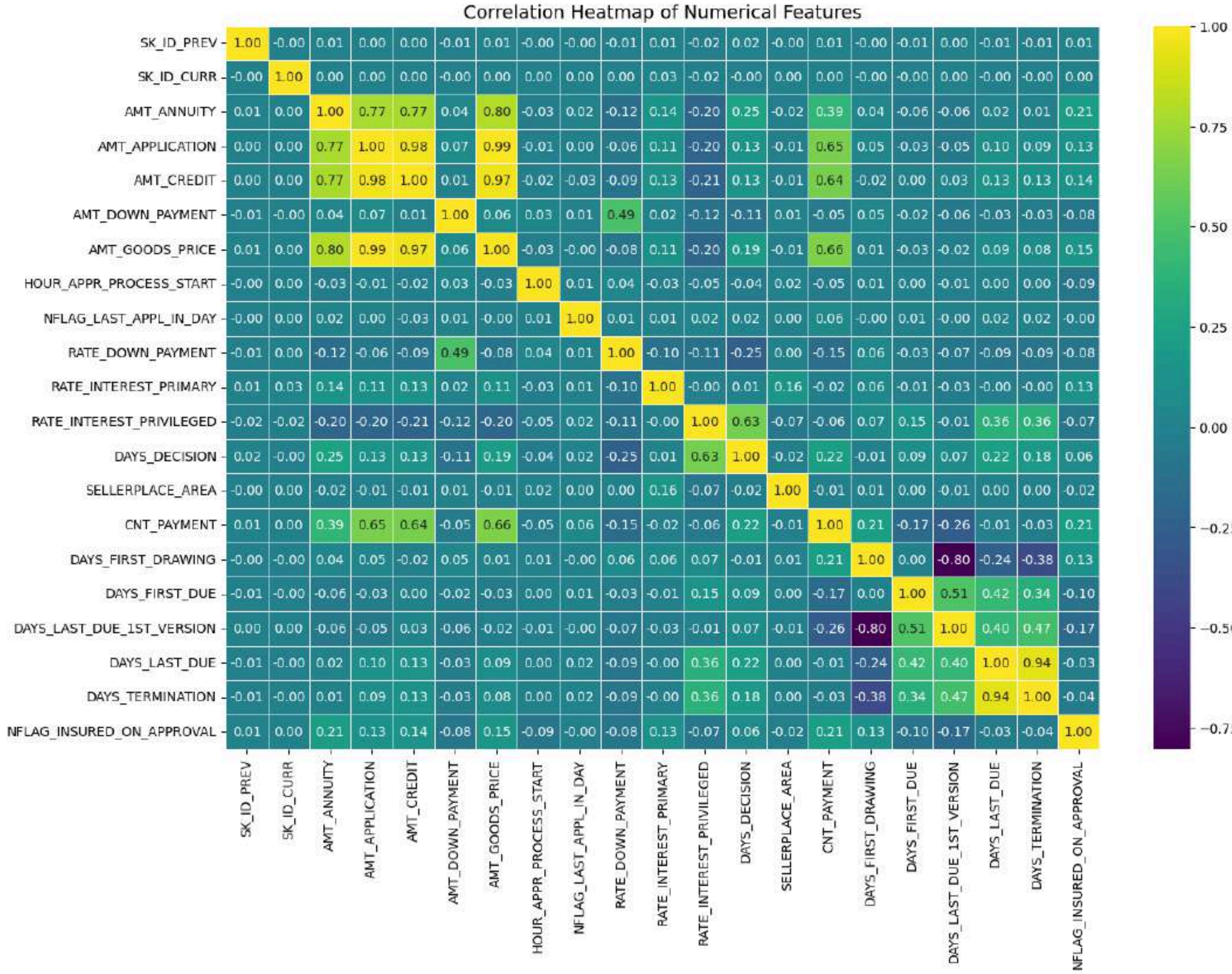
dtype: int64

```
# Fill missing values with the most frequent value (mode)
most_frequent_value = pr_application_data['NAME_TYPE_SUITE'].mode()[0]
pr_application_data['NAME_TYPE_SUITE'].fillna(most_frequent_value, inplace=True)

# Verify that NaN values are filled
print(pr_application_data['NAME_TYPE_SUITE'].value_counts(dropna=False))
```

```
NAME_TYPE_SUITE
Unaccompanied      1043188
Family              392701
Spouse, partner     123750
```

```
Children            57499
Other_B             32233
Other_A             16566
Group of people      4277
Name: count, dtype: int64
<ipython-input-191-1ac1b1f00ca2>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  pr_application_data['NAME_TYPE_SUITE'].fillna(most_frequent_value, inplace=True)
```

```
# Select only numerical columns
numeric_data = pr_application_data.select_dtypes(include=['number'])

# Compute the correlation matrix
corr_matrix = numeric_data.corr()

# Set up the matplotlib figure with increased size
plt.figure(figsize=(15, 10))

# Create the heatmap
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="viridis", linewidths=0.5, cbar=True)

# Add title
plt.title("Correlation Heatmap of Numerical Features", fontsize=14)

# Show the plot
plt.show()
```




```
pr_application_data.drop(columns=["AMT_ANNUITY", "AMT_GOODS_PRICE", "RATE_INTEREST_PRIMARY", "RATE_INTEREST_PRIVILEGED", "DAYS_TERMINATION"]
```

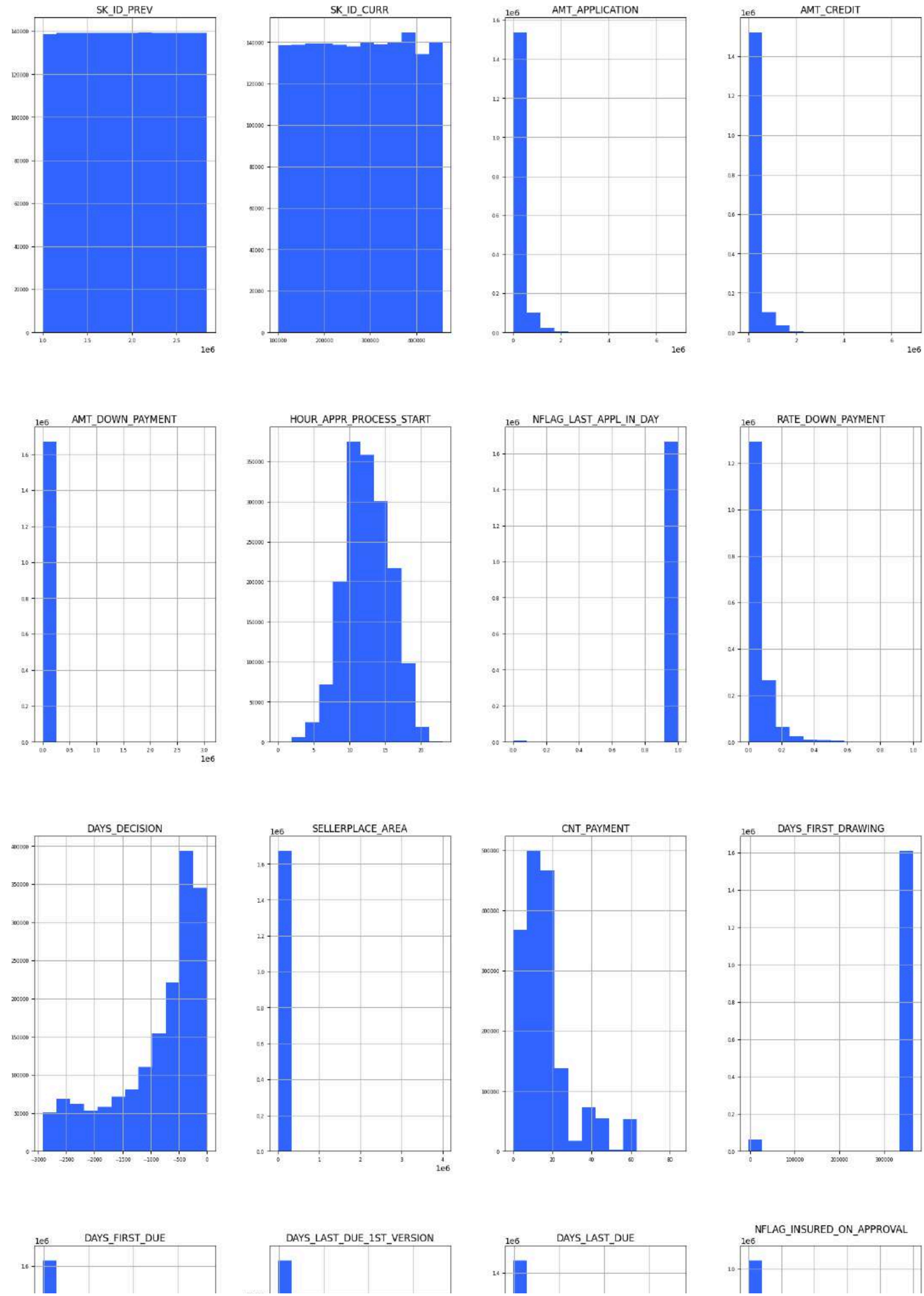
Double-click (or enter) to edit

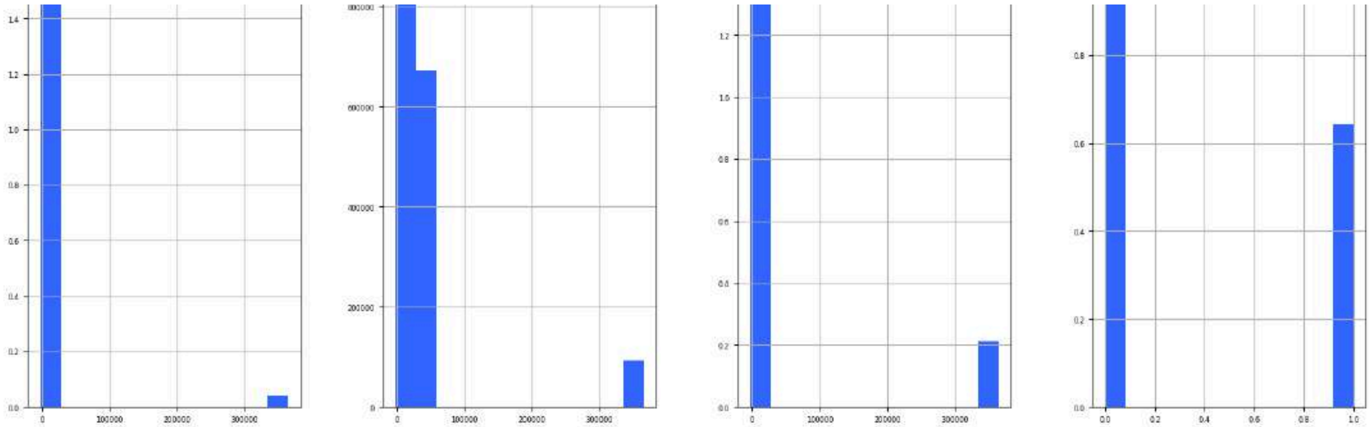
```
pr_application_data.isnull().sum()
```

	0
SK_ID_PREV	0
SK_ID_CURR	0
NAME_CONTRACT_TYPE	0
AMT_APPLICATION	0
AMT_CREDIT	0
AMT_DOWN_PAYMENT	0
WEEKDAY_APPR_PROCESS_START	0
HOUR_APPR_PROCESS_START	0
FLAG_LAST_APPL_PER_CONTRACT	0
NFLAG_LAST_APPL_IN_DAY	0
RATE_DOWN_PAYMENT	0
NAME_CASH_LOAN_PURPOSE	0
NAME_CONTRACT_STATUS	0
DAYS_DECISION	0
NAME_PAYMENT_TYPE	0
CODE_REJECT_REASON	0
NAME_TYPE_SUITE	0
NAME_CLIENT_TYPE	0
NAME_GOODS_CATEGORY	0
NAME_PORTFOLIO	0
NAME_PRODUCT_TYPE	0
CHANNEL_TYPE	0
SELLERPLACE_AREA	0
NAME_SELLER_INDUSTRY	0
CNT_PAYMENT	0
NAME_YIELD_GROUP	0
PRODUCT_COMBINATION	0
DAYS_FIRST_DRAWING	0
DAYS_FIRST_DUE	0
DAYS_LAST_DUE_1ST_VERSION	0
DAYS_LAST_DUE	0
NFLAG_INSURED_ON_APPROVAL	0

dtype: int64

```
pr_application_data.hist(figsize=(20,35),bins=12,xlabelsize=6,ylabelsize=6, color=(0.2, 0.4, 1));
```


SK_ID_PREV
SK_ID_CURR
AMT_APPLICATION
AMT_CREDIT
AMT_DOWN_PAYMENT
HOUR_APPR_PROCESS_START
NFLAG_LAST_APPL_IN_DAY
RATE_DOWN_PAYMENT
DAYS_DECISION
SELLERPLACE_AREA
CNT_PAYMENT
DAYS_FIRST_DRAWING
DAYS_FIRST_DUE
DAYS_LAST_DUE_1ST_VERSION
DAYS_LAST_DUE
NFLAG_INSURED_ON_APPROVAL



Joining the data sets.

Since I am analyzing the financial risk of applicants, the best choice of join depends on how I want to handle missing values.

A Left Join (how="left") is the recommended approach if I want to retain all applicants while incorporating risk-related data when available.

This ensures that I keep all records from the main application_data, which contains the financial details of applicants. If an applicant has no previous application data, their corresponding values from pr_application_data will appear as NaN.

This approach allows me to analyze financial risk for all applicants, even those without prior application history, ensuring a more comprehensive evaluation.

By using a left join, I can maintain the integrity of my dataset while enriching it with relevant financial risk factors.

```
# Perform a left join on the 'SK_ID_CURR' column
merged_data = pd.merge(application_data, pr_application_data, on='SK_ID_CURR', how='left')

# Show the first few rows of the merged data
merged_data.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE_x	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	AMT_INCOME_TOTAL	AMT_CREDIT_x	NAME_TYPE_SUITE_
0	100002	1	Cash loans	M	N	Y	202500.0	406597.5	Unaccompani
1	100003	0	Cash loans	F	N	N	270000.0	1293502.5	Fam
2	100003	0	Cash loans	F	N	N	270000.0	1293502.5	Fam
3	100003	0	Cash loans	F	N	N	270000.0	1293502.5	Fam
4	100004	0	Revolving loans	M	Y	Y	67500.0	135000.0	Unaccompani

5 rows × 69 columns

```
# Check for duplicate columns (with '_x' and '_y' suffixes)
print(merged_data.columns)
```

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE_x', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'AMT_INCOME_TOTAL', 'AMT_CREDIT_x',
       'NAME_TYPE_SUITE_x', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE',
       'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH',
       'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'FLAG_MOBIL', 'FLAG_EMP_PHONE',
       'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_EMAIL',
       'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY',
       'WEEKDAY_APPR_PROCESS_START_x', 'REG_REGION_NOT_LIVE_REGION',
       'REG_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE', 'EXT_SOURCE_2',
       'EXT_SOURCE_3', 'DAYS_LAST_PHONE_CHANGE', 'AGE', 'SALARY_CATEGORY',
       'SK_ID_PREV', 'NAME_CONTRACT_TYPE_y', 'AMT_APPLICATION', 'AMT_CREDIT_y',
       'AMT_DOWN_PAYMENT', 'WEEKDAY_APPR_PROCESS_START_y',
       'HOUR_APPR_PROCESS_START', 'FLAG_LAST_APPL_PER_CONTRACT',
       'NFLAG_LAST_APPL_IN_DAY', 'RATE_DOWN_PAYMENT', 'NAME_CASH_LOAN_PURPOSE',
       'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CODE_REJECT_REASON', 'NAME_TYPE_SUITE_y', 'NAME_CLIENT_TYPE',
       'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE',
       'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY',
       'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
       'DAYS_LAST_DUE', 'NFLAG_INSURED_ON_APPROVAL'],
      dtype='object')
```

```
# Check for missing values in the merged dataset
merged_data.isnull().sum()
```

	0
SK_ID_CURR	0
TARGET	0
NAME_CONTRACT_TYPE_x	0
CODE_GENDER	0
FLAG_OWN_CAR	0
...	...
DAYS_FIRST_DRAWING	16454
DAYS_FIRST_DUE	16454
DAYS_LAST_DUE_1ST_VERSION	16454
DAYS_LAST_DUE	16454
NFLAG_INSURED_ON_APPROVAL	16454

69 rows × 1 columns

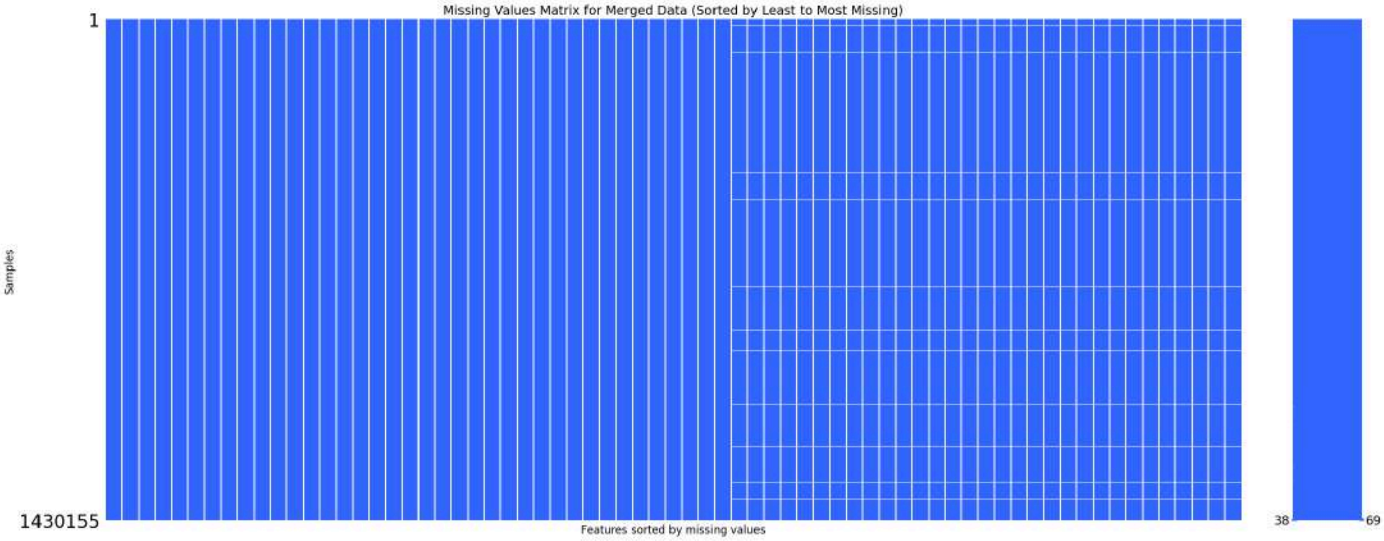
dtype: int64

```
# Display missing values for all columns in the dataset
missing_values = merged_data.isnull().sum()

# To display the missing values count for all columns
print(missing_values)

# If you want to see only columns with missing values:
missing_values = missing_values[missing_values > 0]
print(missing_values.head(-3))
```

```
SK_ID_CURR                   0
TARGET                       0
NAME_CONTRACT_TYPE_x         0
CODE_GENDER                  0
FLAG_OWN_CAR                 0
                            ..
DAYS_LAST_DUE                0
NFLAG_INSURED_ON_APPROVAL    0
Income_Category              0
Age_Group                    0
Employment Duration Group    9
Length: 72, dtype: int64
Series([], dtype: int64)
```

```
# Sort columns by missing values in ascending order
sorted_columns = merged_data.isnull().sum().sort_values().index
sorted_data = merged_data[sorted_columns]

# Plot missing values matrix with sorted features
plt.figure(figsize=(12, 6))
msno.matrix(sorted_data, color=(0.2, 0.4, 1))  # Blue color (RGB values)
plt.xlabel("Features sorted by missing values", fontsize=12)
plt.ylabel("Samples", fontsize=12)
plt.title("Missing Values Matrix for Merged Data (Sorted by Least to Most Missing)", fontsize=14)
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```




```
# Calculate the percentage of missing values per column
missing_percentage = (merged_data.isnull().sum() * 100 / len(merged_data)).round(2)

# Sort values in descending order (most missing first)
missing_percentage = missing_percentage.sort_values(ascending=False)

# Display the sorted missing percentage
print(missing_percentage.head(5))
```

```
SK_ID_CURR            0.0
TARGET                0.0
CODE_REJECT_REASON    0.0
NAME_PAYMENT_TYPE     0.0
DAYS_DECISION         0.0
dtype: float64
```

```
# Define a threshold for missing values (e.g., 30%)
threshold = 0.3 * len(merged_data)

# Drop columns where missing values exceed threshold
merged_data.dropna(thresh=threshold, axis=1, inplace=True)

# Fill remaining missing values
for col in merged_data.columns:
    if merged_data[col].dtype == 'object':  # Categorical
        merged_data[col].fillna(merged_data[col].mode()[0], inplace=True)
    else:  # Numerical
        merged_data[col].fillna(merged_data[col].median(), inplace=True)
```

```
<ipython-input-202-5d4b16ce38fb>:12: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assig
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  merged_data[col].fillna(merged_data[col].median(), inplace=True)
<ipython-input-202-5d4b16ce38fb>:10: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assig
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  merged_data[col].fillna(merged_data[col].mode()[0], inplace=True)
```

```
# Calculate the percentage of missing values per column
missing_percentage = (merged_data.isnull().sum() * 100 / len(merged_data)).round(2)

# Sort values in descending order (most missing first)
missing_percentage = missing_percentage.sort_values(ascending=False)

# Display the sorted missing percentage
print(missing_percentage.head(5))
```

```
SK_ID_CURR            0.0
TARGET                0.0
CODE_REJECT_REASON    0.0
NAME_PAYMENT_TYPE     0.0
DAYS_DECISION         0.0
dtype: float64
```

```
# Check for duplicate columns (with '_x' and '_y' suffixes)
print(merged_data.columns)
```

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE_x', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'AMT_INCOME_TOTAL', 'AMT_CREDIT_x',
       'NAME_TYPE_SUITE_x', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE',
       'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH',
       'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'FLAG_MOBIL', 'FLAG_EMP_PHONE',
       'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_EMAIL',
       'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY',
       'WEEKDAY_APPR_PROCESS_START_x', 'REG_REGION_NOT_LIVE_REGION',
       'REG_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE', 'EXT_SOURCE_2',
       'EXT_SOURCE_3', 'DAYS_LAST_PHONE_CHANGE', 'AGE', 'SALARY_CATEGORY',
       'SK_ID_PREV', 'NAME_CONTRACT_TYPE_y', 'AMT_APPLICATION', 'AMT_CREDIT_y',
       'AMT_DOWN_PAYMENT', 'WEEKDAY_APPR_PROCESS_START_y',
       'HOUR_APPR_PROCESS_START', 'FLAG_LAST_APPL_PER_CONTRACT',
       'NFLAG_LAST_APPL_IN_DAY', 'RATE_DOWN_PAYMENT', 'NAME_CASH_LOAN_PURPOSE',
       'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CODE_REJECT_REASON', 'NAME_TYPE_SUITE_y', 'NAME_CLIENT_TYPE',
       'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE',
       'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY',
       'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
       'DAYS_LAST_DUE', 'NFLAG_INSURED_ON_APPROVAL'],
      dtype='object')
```

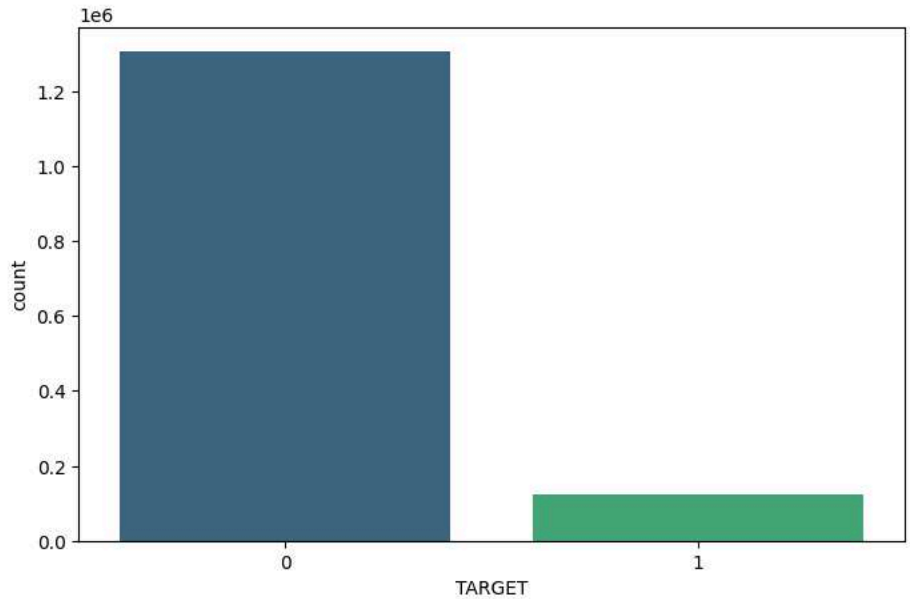
Visualisaton

Visialisation "Target"

This tells us the proportion of clients who defaulted (1) versus those who repaid (0). Insight: If the dataset is imbalanced (e.g., very few defaults), we may need special modeling techniques such as smote to handle the imbalance.

```
# Create a bar chart with blue and orange bars
plt.figure(figsize=(8, 5))
sns.countplot(x=merged_data['TARGET'], palette='viridis')

# Set the title and labels
```

```
<ipython-input-205-4cb37be1e9a9>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.countplot(x=merged_data['TARGET'], palette='viridis')
<Axes: xlabel='TARGET', ylabel='count'>
```



Visualisations insights says that we have more applications, which were repaid rather than defaulted.

I will explore more features to find meaningful insights based on customers characteristics and financial behaviour.

AMT_INCOME_TOTAL

Since I am working with the dataset merged_data and want to visualise the relationship between AMT_INCOME_TOTAL and TARGET. Income is a major factor in a client's ability to repay a loan. If applicants have low income but take high loans, there could be a higher risk of default.

```
merged_data['AMT_INCOME_TOTAL'] = merged_data['AMT_INCOME_TOTAL'].astype(int)
```

```
merged_data['AMT_INCOME_TOTAL'].describe()
```

	AMT_INCOME_TOTAL
count	1.430155e+06
mean	1.736036e+05
std	1.983303e+05
min	2.565000e+04
25%	1.125000e+05
50%	1.575000e+05
75%	2.115000e+05
max	1.170000e+08

dtype: float64

Start coding or generate with AI.

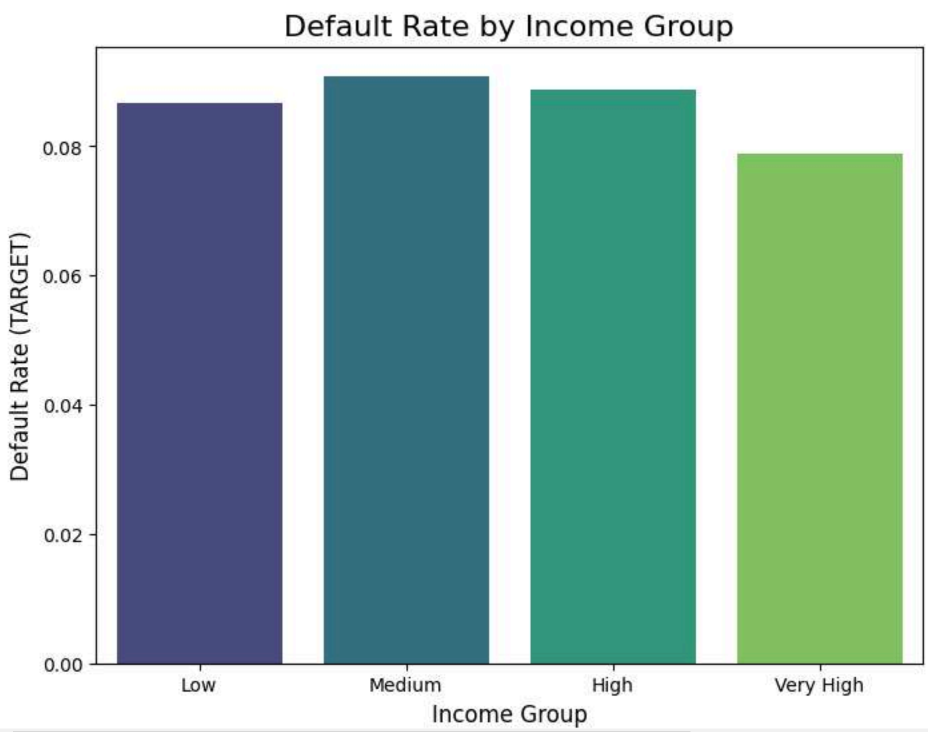
```
 # Define the bins based on descriptive statistics
bins = [0, 112500, 157500, 211500, float('inf')]  # Add upper range for outliers
labels = ['Low', 'Medium', 'High', 'Very High']

# Create a new column 'Income_Category'
merged_data['Income_Category'] = pd.cut(merged_data['AMT_INCOME_TOTAL'], bins=bins, labels=labels)

# Check the first few rows to see the result
merged_data[['AMT_INCOME_TOTAL', 'Income_Category']].head(5)
```

	AMT_INCOME_TOTAL	Income_Category
0	202500	High
1	270000	Very High
2	270000	Very High
3	270000	Very High
4	67500	Low

Comparing the amount of credit requested to the applicant's income helps identify if clients borrow within their means. If low-income clients take out large loans, that could signal potential financial distress

```
# Group the data by Income Group and calculate the mean of TARGET (default rate)
income_default_rate = merged_data.groupby('Income_Category')['TARGET'].mean().reset_index()

# Create the bar plot
plt.figure(figsize=(8, 6))
sns.barplot(x='Income_Category', y='TARGET', data=income_default_rate, palette='viridis')

# Set plot labels and title
plt.title('Default Rate by Income Group', fontsize=16)
plt.xlabel('Income Group', fontsize=12)
plt.ylabel('Default Rate (TARGET)', fontsize=12)

# Show the plot
plt.show()
```

```
<ipython-input-209-7bd1778f8b05>:2: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future v
  income_default_rate = merged_data.groupby('Income_Category')['TARGET'].mean().reset_index()
<ipython-input-209-7bd1778f8b05>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.barplot(x='Income_Category', y='TARGET', data=income_default_rate, palette='viridis')
```



Income gorupping does not provede clear result as we can see that aplication can be still rejected based on othere varibales.

```
Start coding or generate with AI.
```

```
# Convert DAYS_BIRTH to age (divide by -365 because the value is in days and negative)
merged_data['AGE'] = -merged_data['DAYS_BIRTH'] // 365
```

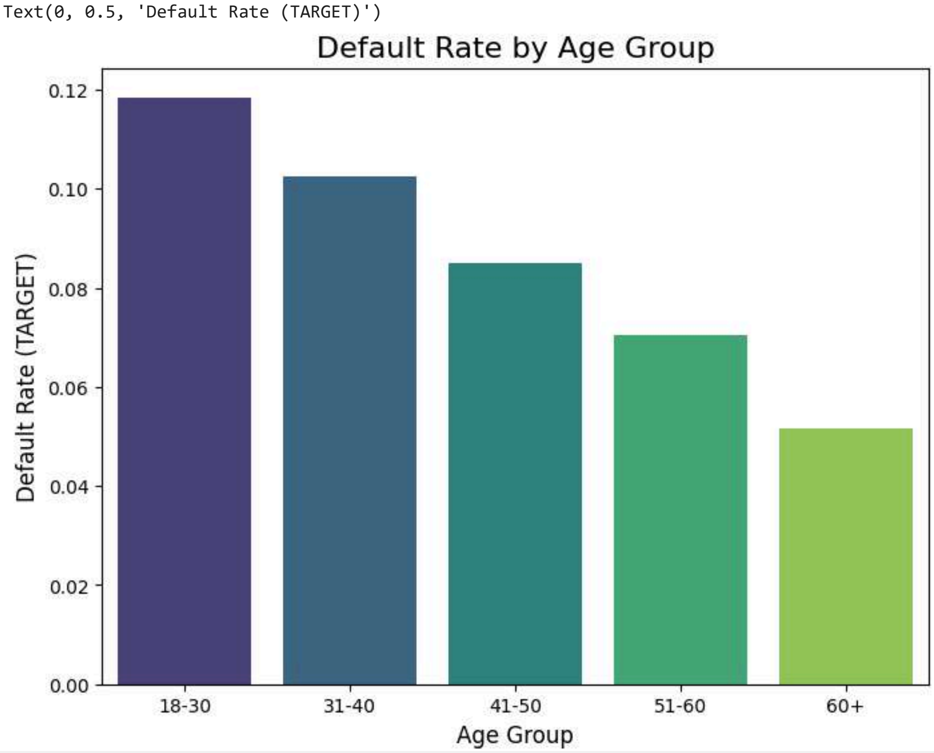
```
merged_data['AGE'].head(5)
```

	AGE
0	25
1	45
2	45
3	45
4	52

dtype: int64

```
bins = [18, 30, 40, 50, 60, 100]  # Example age bins
labels = ['18-30', '31-40', '41-50', '51-60', '60+']  # Age group labels
merged_data['Age_Group'] = pd.cut(merged_data['AGE'], bins=bins, labels=labels, right=False)

# Group by Age Group and calculate the mean of TARGET (default rate)
age_default_rate = merged_data.groupby('Age_Group')['TARGET'].mean().reset_index()

# Create the bar plot
plt.figure(figsize=(8, 6))
sns.barplot(x='Age_Group', y='TARGET', data=age_default_rate, palette='viridis')

# Set plot labels and title
plt.title('Default Rate by Age Group', fontsize=16)
plt.xlabel('Age Group', fontsize=12)
plt.ylabel('Default Rate (TARGET)', fontsize=12)

# Show the plot
```

```
<ipython-input-212-f6b8dc1121c9>:6: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future v
  age_default_rate = merged_data.groupby('Age_Group')['TARGET'].mean().reset_index()
<ipython-input-212-f6b8dc1121c9>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.barplot(x='Age_Group', y='TARGET', data=age_default_rate, palette='viridis')
Text(0, 0.5, 'Default Rate (TARGET)')
```



Start coding or generate with AI.

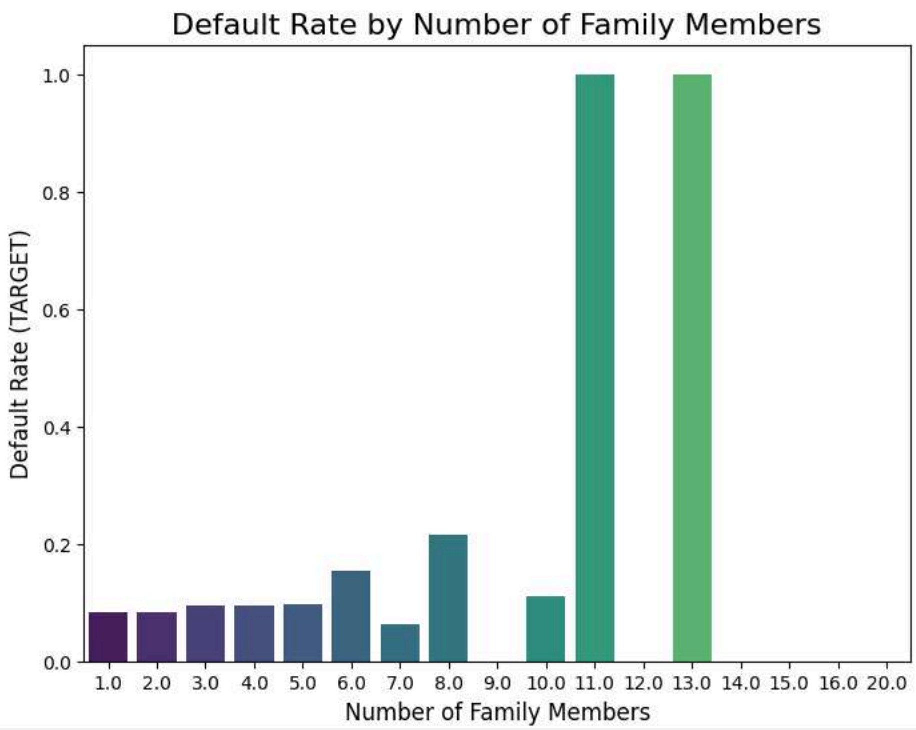
So the highest default rate is observed in the age group below 25 years, based on the DAYS_BIRTH vs. TARGET visualization. This is an important insight because it suggests that younger clients (those under 25) may present a higher credit risk. It is clear to see that people of older age more likely to be financialy stable and repay the loan. They might have hifher income compary to people younger age.

Family Members

This is important for risk assessment because the number of family members can be an indicator of an applicant's financial responsibilities. A larger family typically means higher living expenses, which may reduce the applicant's ability to repay the loan. By factoring in the number of family members, financial institutions can better assess the applicant's capacity to meet repayment obligations and identify higher-risk applicants, helping them make more informed lending decisions

```
# Group by CNT_FAM_MEMBERS and calculate the mean of TARGET (default rate)
family_default_rate = merged_data.groupby('CNT_FAM_MEMBERS')['TARGET'].mean().reset_index()

# Create the bar plot
plt.figure(figsize=(8, 6))
sns.barplot(x='CNT_FAM_MEMBERS', y='TARGET', data=family_default_rate, palette='viridis')

# Set plot labels and title
plt.title('Default Rate by Number of Family Members', fontsize=16)
plt.xlabel('Number of Family Members', fontsize=12)
plt.ylabel('Default Rate (TARGET)', fontsize=12)
```

```
<ipython-input-213-10bb64b39224>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.barplot(x='CNT_FAM_MEMBERS', y='TARGET', data=family_default_rate, palette='viridis')
Text(0, 0.5, 'Default Rate (TARGET)')
```



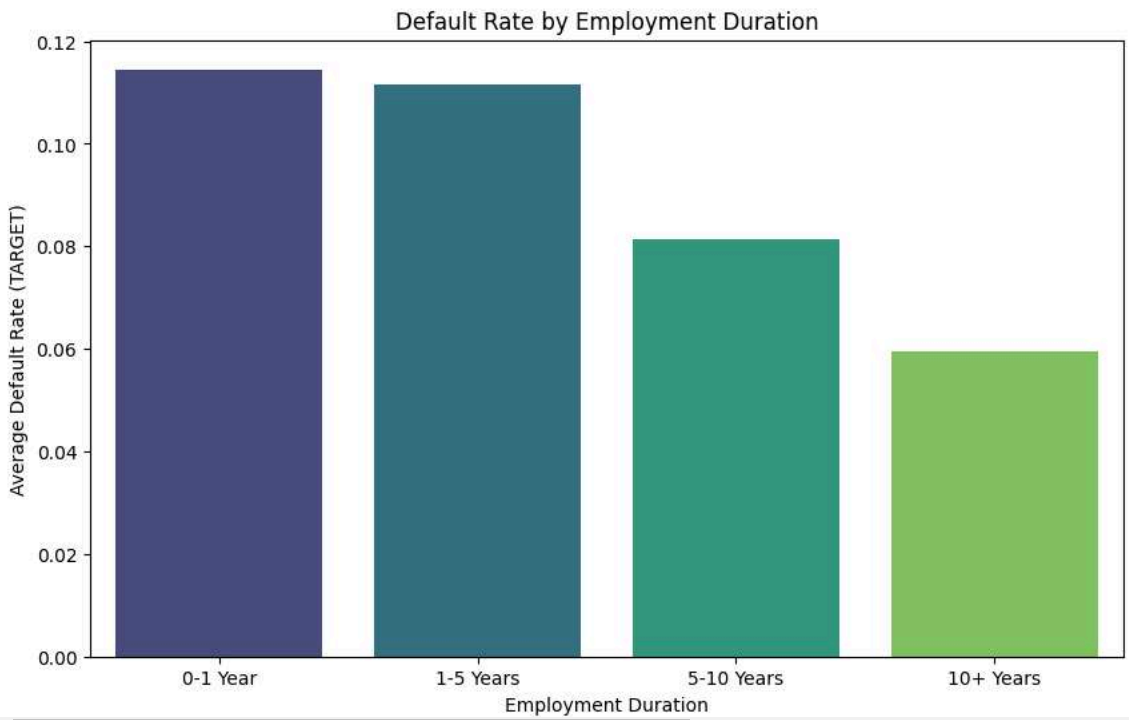
The chart indicates that applicants with more than five family members are more likely to be refused a loan. This could be due to the financial burden of supporting a larger family. In many cases, there may be two working adults, but as the number of family members increases, the available income for repaying the loan may decrease, leading to a higher risk of default.

Days Employed

The feature DAYS_EMPLOYED is important for risk assessment because it indicates the stability and reliability of an applicant's employment history. A longer period of employment suggests a stable source of income, which can improve the applicant's ability to repay the loan. On the other hand, a short or inconsistent employment history could signal instability, which increases the risk of default. This feature helps lenders assess the applicant's financial stability and predict their likelihood of repayment, making it a crucial factor in evaluating loan risk

```
# Convert negative values in 'DAYS_EMPLOYED' to positive (since it's the number of days since employment started)
merged_data['DAYS_EMPLOYED'] = merged_data['DAYS_EMPLOYED'].apply(lambda x: abs(x))


# Create bins for 'DAYS_EMPLOYED'
bins = [0, 365, 365*5, 365*10, float('inf')]  # 0-1 year, 1-5 years, 5-10 years, 10+ years
labels = ['0-1 Year', '1-5 Years', '5-10 Years', '10+ Years']
merged_data['Employment Duration Group'] = pd.cut(merged_data['DAYS_EMPLOYED'], bins=bins, labels=labels)

# Calculate the mean default rate (TARGET) for each employment duration category
default_rate_by_employment = merged_data.groupby('Employment Duration Group')['TARGET'].mean().reset_index()

# Create bar plot
plt.figure(figsize=(10,6))
sns.barplot(x='Employment Duration Group', y='TARGET', data=default_rate_by_employment, palette='viridis')
plt.title('Default Rate by Employment Duration')
plt.xlabel('Employment Duration')
plt.ylabel('Average Default Rate (TARGET)')
plt.show()
```

```
<ipython-input-216-1be33c754372>:7: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future v
  default_rate_by_employment = merged_data.groupby('Employment Duration Group')['TARGET'].mean().reset_index()
<ipython-input-216-1be33c754372>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.barplot(x='Employment Duration Group', y='TARGET', data=default_rate_by_employment, palette='viridis')
```



Applicants with a shorter employment history are more likely to be denied a loan compared to those with a stable job history. A stable and longer work tenure signals financial reliability and a consistent income source, which increases the likelihood of timely loan repayment. In contrast, a shorter employment history may raise concerns about the applicant's financial stability and ability to sustain regular payments, making them a higher risk for default. This makes DAYS_EMPLOYED a crucial factor in assessing loan risk

Education

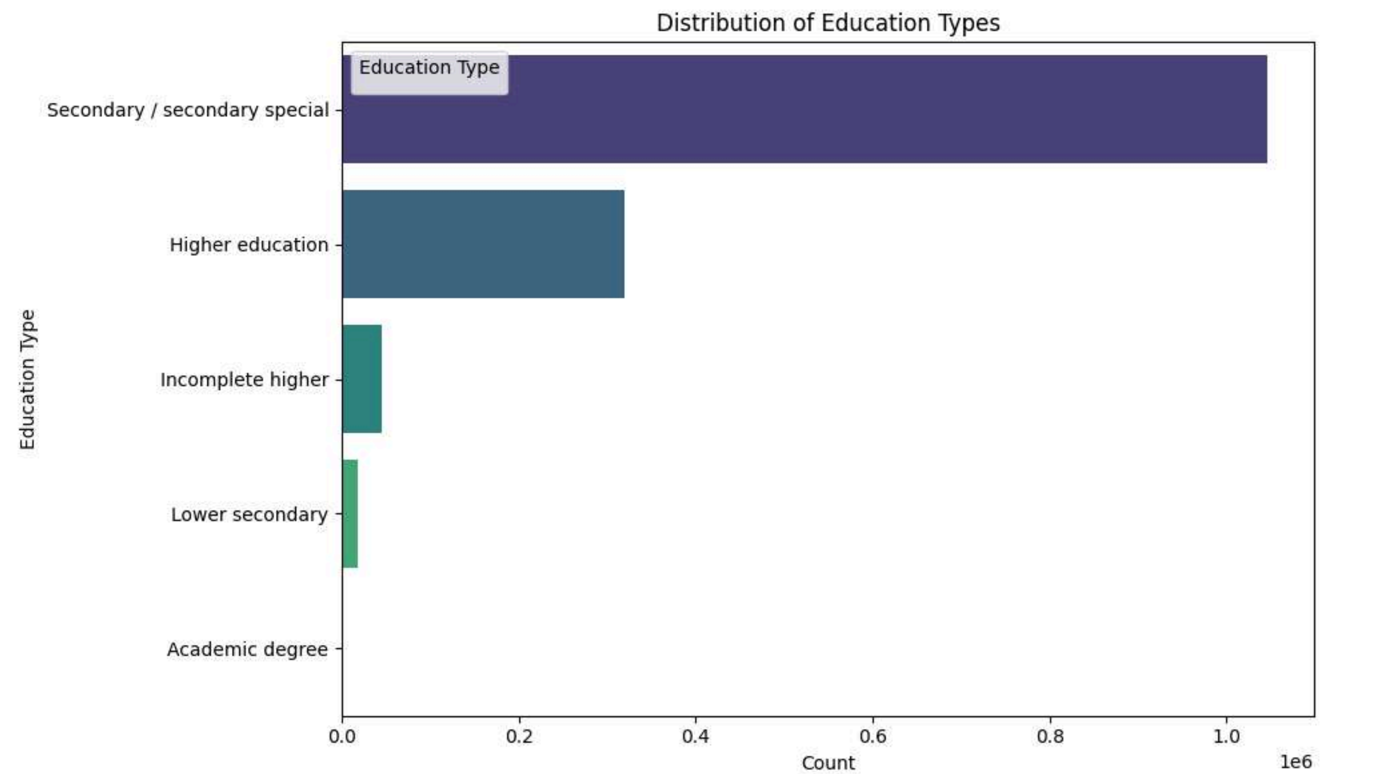
Education plays a significant role in shaping individuals' financial behavior, stability, and access to resources. People with higher education tend to have higher incomes and better financial literacy, which can influence their ability to manage loans, investments, and other financial decisions. In many cases, financial institutions may consider education as a factor when assessing loan applications, as educated individuals might be perceived as more likely to repay loans due to their financial stability.

```
# Calculate the value counts for each education type
education_counts = merged_data['NAME_EDUCATION_TYPE'].value_counts()
print (education_counts)
```

```
NAME_EDUCATION_TYPE
Secondary / secondary special    1046822
Higher education                  319692
Incomplete higher                  45751
Lower secondary                    17300
Academic degree                      590
Name: count, dtype: int64
```

```
# Calculate the value counts for each education type
education_counts = merged_data['NAME_EDUCATION_TYPE'].value_counts()

# Create a horizontal bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x=education_counts.values,
            y=education_counts.index,
            palette='viridis')  # Using Seaborn's Set3 palette for colors

# Customize the plot
plt.title('Distribution of Education Types')
plt.xlabel('Count')
plt.ylabel('Education Type')

# Add a legend
handles, labels = plt.gca().get_legend_handles_labels()
plt.legend(handles, labels, title="Education Type", loc="upper left")

# Show the plot
plt.tight_layout()
plt.show()
```

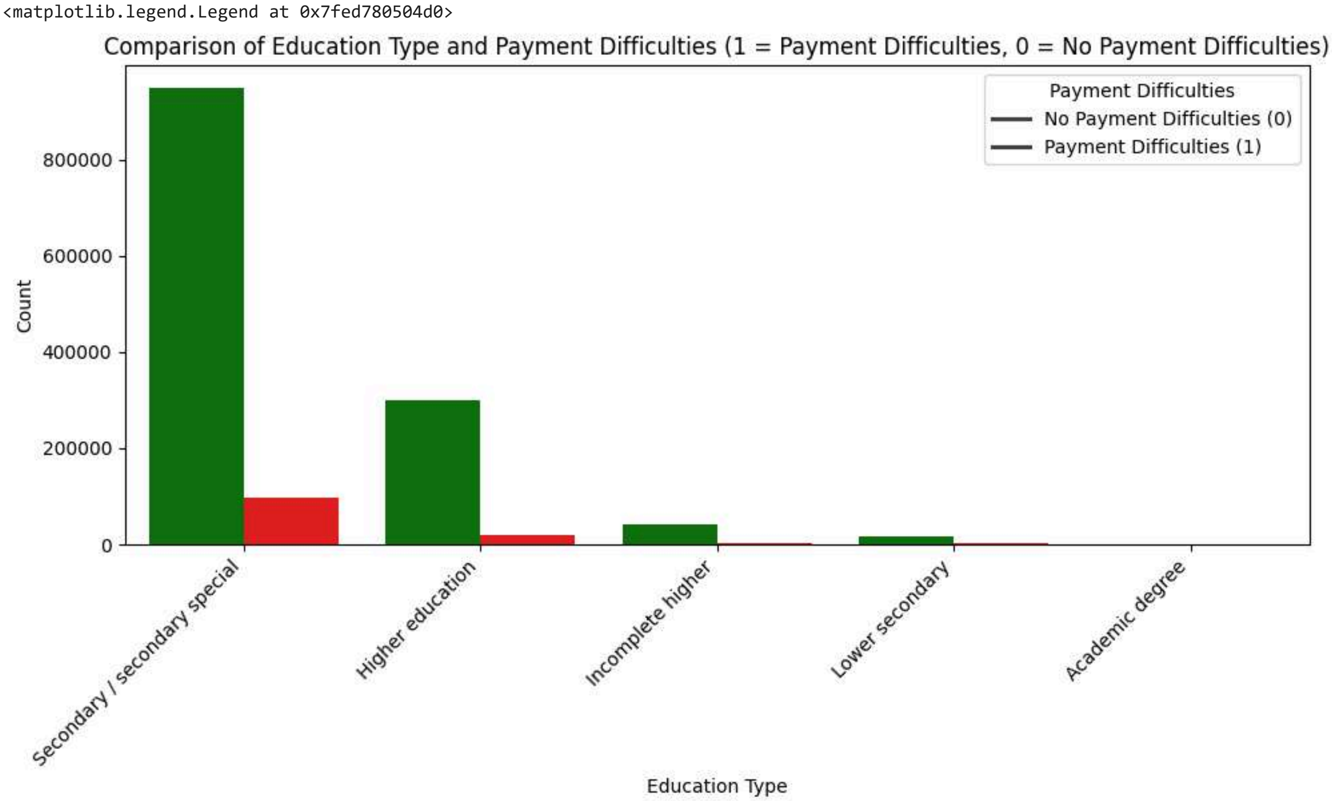
```
<Figure size 1000x600 with 0 Axes>
<ipython-input-237-4b065ea15fe6>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend

  sns.barplot(x=education_counts.values,
<Axes: ylabel='NAME_EDUCATION_TYPE'>
Text(0.5, 1.0, 'Distribution of Education Types')
Text(0.5, 0, 'Count')
Text(0, 0.5, 'Education Type')
<matplotlib.legend.Legend at 0x7fed772efb50>
```




```
# Define the specific order of education types
education_order = ['Secondary / secondary special',
                   'Higher education',
                   'Incomplete higher',
```

```
                'Lower secondary',
                'Academic degree']

# Count the occurrences of TARGET for each education type
education_target_counts = merged_data.groupby(['NAME_EDUCATION_TYPE', 'TARGET']).size().reset_index(name='count')

# Define custom colors for payment difficulties (1) and no payment difficulties (0)
custom_palette = {0: 'green', 1: 'red'}

# Create a bar plot
plt.figure(figsize=(10, 6))

# Use the custom palette
sns.barplot(data=education_target_counts,
            x='NAME_EDUCATION_TYPE',
            y='count',
            hue='TARGET',
            palette=custom_palette,
            order=education_order)  # Set the order of education types

# Customize the plot
plt.title('Comparison of Education Type and Payment Difficulties (1 = Payment Difficulties, 0 = No Payment Difficulties)')
plt.xlabel('Education Type')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')

# Update legend with the colors
plt.legend(title='Payment Difficulties',
           labels=['No Payment Difficulties (0)', 'Payment Difficulties (1)'],
           loc='upper right')

# Show the plot
plt.tight_layout()
plt.show()
```

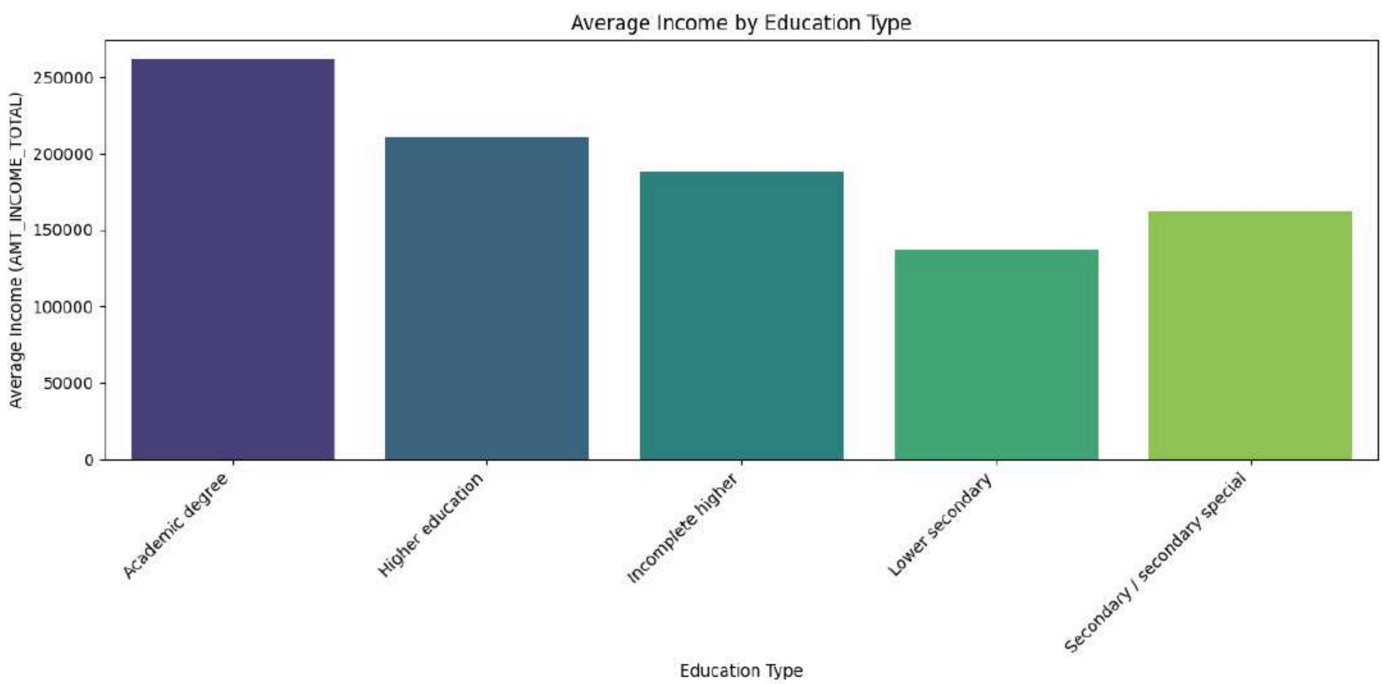
```
<Figure size 1000x600 with 0 Axes>
<Axes: xlabel='NAME_EDUCATION_TYPE', ylabel='count'>
Text(0.5, 1.0, 'Comparison of Education Type and Payment Difficulties (1 = Payment Difficulties, 0 = No Payment Difficulties)')
Text(0.5, 0, 'Education Type')
Text(0, 0.5, 'Count')
([0, 1, 2, 3, 4],
 [Text(0, 0, 'Secondary / secondary special'),
  Text(1, 0, 'Higher education'),
  Text(2, 0, 'Incomplete higher'),
  Text(3, 0, 'Lower secondary'),
  Text(4, 0, 'Academic degree')])
<matplotlib.legend.Legend at 0x7fed780504d0>
```



The NAME_EDUCATION_TYPE feature is important for risk assessment because it shows that while individuals with Secondary / Secondary Special and Higher Education tend to apply more, they also face a higher rejection rate. In contrast, applicants with Incomplete Higher, Lower Secondary, or Academic Degree may have less applications. This suggests that other factors, like income and employment history, play a role in loan approval, and understanding this relationship can help refine risk models and improve the loan evaluation process.

```
# 1. Average Income by Education Type (Bar Chart)
# Grouping by education type and calculating the mean of AMT_INCOME_TOTAL
education_income = merged_data.groupby('NAME_EDUCATION_TYPE')['AMT_INCOME_TOTAL'].mean().reset_index()

# Create the bar plot for average income by education type
plt.figure(figsize=(12, 6))
sns.barplot(x='NAME_EDUCATION_TYPE', y='AMT_INCOME_TOTAL', data=education_income, palette='viridis')
plt.title('Average Income by Education Type')
plt.xlabel('Education Type')
plt.ylabel('Average Income (AMT_INCOME_TOTAL)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
<ipython-input-239-58855bc462d1>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.barplot(x='NAME_EDUCATION_TYPE', y='AMT_INCOME_TOTAL', data=education_income, palette='viridis')
<Axes: xlabel='NAME_EDUCATION_TYPE', ylabel='AMT_INCOME_TOTAL'>
Text(0.5, 1.0, 'Average Income by Education Type')
Text(0.5, 0, 'Education Type')
Text(0, 0.5, 'Average Income (AMT_INCOME_TOTAL)')
([0, 1, 2, 3, 4],
 [Text(0, 0, 'Academic degree'),
  Text(1, 0, 'Higher education'),
  Text(2, 0, 'Incomplete higher'),
  Text(3, 0, 'Lower secondary'),
  Text(4, 0, 'Secondary / secondary special')])
```



Education and Income: There's a clear relationship between education level and income, with those having higher education or academic degrees generally having higher incomes. This could reduce their need for loans.

Loan Approval: Despite higher incomes, people with higher education or secondary special education still tend to have higher loan approval rates, possibly due to their financial aspirations, career growth, or additional financial needs.

CONTRACT STATUS

The NAME_CONTRACT_STATUS feature is crucial for risk assessment as it directly reflects the status of a loan application. This variable helps to categorize whether an application was approved, rejected, or terminated, offering a clear indication of an applicant's relationship with credit institutions. The status of a contract provides insight into how likely applicants are to successfully obtain loans, based on historical trends. For instance, applicants whose previous contracts were terminated or rejected may be considered high-risk by lenders due to their history of default or failure to meet requirements.

By incorporating NAME_CONTRACT_STATUS, financial institutions can better predict an applicant's future behaviour, as previous loan outcomes are often strong indicators of future repayment ability. This makes it a valuable tool for assessing and managing credit risk, allowing lenders to refine their models and ensure more accurate decision-making. The default rate across different loan statuses will give insights into how the previous loan status relates to the likelihood of default.

For example, if a contract status like "approved" has a higher default rate, it suggests that customers with previous approved loans are more likely to default.

[240]
```
merged_data.NAME_CONTRACT_STATUS.value_counts()
```

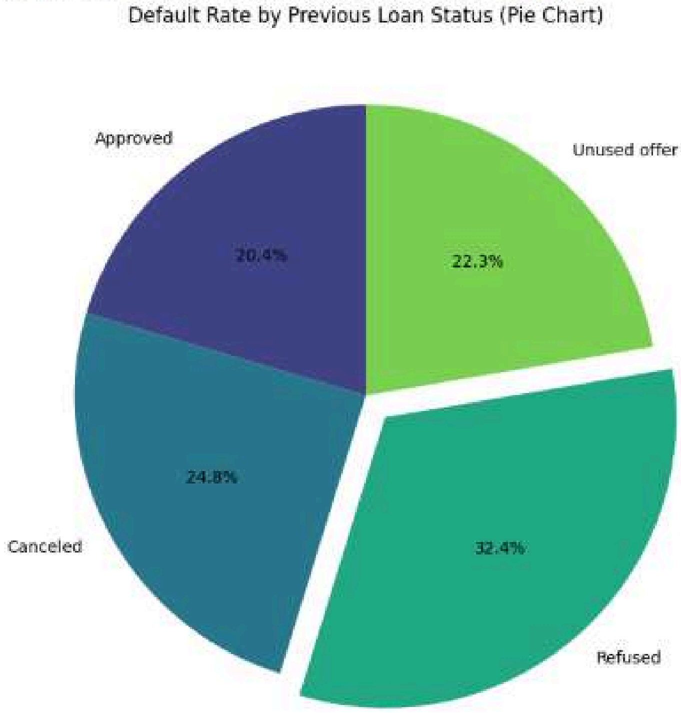
	count
NAME_CONTRACT_STATUS	
Approved	902553
Canceled	259441
Refused	245390
Unused offer	22771

dtype: int64

[241]
```
# Group by contract status and calculate the mean of the TARGET variable (default rate)
contract_status_default_rate = merged_data.groupby('NAME_CONTRACT_STATUS')['TARGET'].mean().reset_index()

# Calculate the explode parameter to highlight the largest slice
explode = [0.1 if val == contract_status_default_rate['TARGET'].max() else 0 for val in contract_status_default_rate['TARGET']]

# Plot a pie chart
plt.figure(figsize=(8, 8))
plt.pie(contract_status_default_rate['TARGET'],
        labels=contract_status_default_rate['NAME_CONTRACT_STATUS'],
        autopct='%1.1f%%', startangle=90,
        colors=sns.color_palette('viridis', len(contract_status_default_rate)),
        explode=explode)  # The explode argument to pull out the largest slice

# Add title
plt.title('Default Rate by Previous Loan Status (Pie Chart)')

# Show the plot
plt.show()
```

```
<Figure size 800x800 with 0 Axes>
([<matplotlib.patches.Wedge at 0x7fed77f8d4d0>,
  <matplotlib.patches.Wedge at 0x7fed70724d50>,
  <matplotlib.patches.Wedge at 0x7fed84575bd0>,
  <matplotlib.patches.Wedge at 0x7fed77796d10>],
 [Text(-0.6588072653334797, 0.8808932893057038, 'Approved'),
  Text(-0.9691056331099662, -0.5204174015869683, 'Canceled'),
  Text(0.7916762259387308, -0.9018030568164026, 'Refused'),
  Text(0.7094836619359246, 0.840614616483672, 'Unused offer')],
 [Text(-0.35934941745462523, 0.48048724871220194, '20.4%'),
  Text(-0.528603072605436, -0.2838640372292554, '24.8%'),
```

```
 Text(0.7916762259387308, -0.9018030508164026, 'Refused'),
 Text(0.7094836619359246, 0.840614616483672, 'Unused offer')],
[Text(-0.35934941745462523, 0.48048724871220194, '20.4%'),
 Text(-0.528603072605436, -0.2838640372292554, '24.8%'),
 Text(0.4618111317975929, -0.5260517831429015, '32.4%'),
 Text(0.3869910883286861, 0.4585170635365483, '22.3%')])
Text(0.5, 1.0, 'Default Rate by Previous Loan Status (Pie Chart)')
```



Refused Applicants as Potential Risk: Applicants with refused applications often display a higher default rate, making them an important group to monitor closely. The refusal of an application could signal potential underlying issues with their creditworthiness, such as a poor financial history or inability to meet the bank's criteria. This makes them a high-risk group for future defaults, and thus, they require more attention and thorough evaluation in future credit assessments. By identifying these individuals early, banks can mitigate risk and implement precautionary measures to manage their exposure.

Rejection reason

The CODE_REJECT_REASON feature is crucial for risk assessment because it provides insight into the specific reasons why applicants' loan applications were declined. By analyzing these rejection codes, financial institutions can identify patterns or recurring issues that may indicate potential risk factors for defaults.

```
[242] # Step 1: Drop missing values in relevant columns
      merged_data = merged_data.dropna(subset=['CODE_REJECT_REASON', 'TARGET'])

      # Step 2: Calculate Default Rate per Rejection Reason and Sort
      reject_pivot = merged_data.groupby('CODE_REJECT_REASON')['TARGET'].mean().to_frame()
      reject_pivot = reject_pivot.sort_values(by='TARGET', ascending=False)  # Sorting by highest default rate

      # Step 3: Plot the Heatmap
      plt.figure(figsize=(10, 6))
      sns.heatmap(reject_pivot, annot=True, fmt=".2%", cmap="viridis", linewidths=0.5, cbar=True)
```

```python
# Step 4: Customize Labels
plt.title("Impact of Past Rejections on Default Rate")
plt.xlabel("Default Rate (%)")
plt.ylabel("Rejection Reason")

plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
<Axes: ylabel='CODE_REJECT_REASON'>
Text(0.5, 1.0, 'Impact of Past Rejections on Default Rate')
Text(0.5, 36.72222222222221, 'Default Rate (%)')
Text(95.72222222222221, 0.5, 'Rejection Reason')
```

SCOFR → "Score Fraud Risk", where an applicant's profile was flagged for potential fraudulent behavior.

LIMIT → "Exceeds Credit Limit", where the requested loan amount was beyond what was deemed acceptable.

HC → "High Credit Risk", meaning applicants were rejected due to their creditworthiness.

XNA → "No Information Available", where there was insufficient data to assess the application.

SCO →"Low Credit Score", indicating rejection due to a poor credit rating.

VERIF → "Verification Issues", meaning the application was rejected due to document mismatches, unverified income, or identity concerns.

CLIENT → "Client-Specific Rejection", possibly due to missing documents, incomplete applications, or policy violations.

XAP → "Application Approved but Not Used". This could indicate applicants who were approved but chose not to take the loan.

SYSTEM → "System-Based Rejection", possibly due to automated rules, missing data, or internal system checks.

Loan Purpose

The feature NAME_CASH_LOAN_PURPOSE is crucial for risk assessment because it indicates the reason for which an applicant is requesting the loan. Different loan purposes can reveal insights into the financial behavior of the applicant and their likelihood of repaying the loan. Here's why this feature is important for risk assessment, especially when comparing it with the TARGET (default rate):

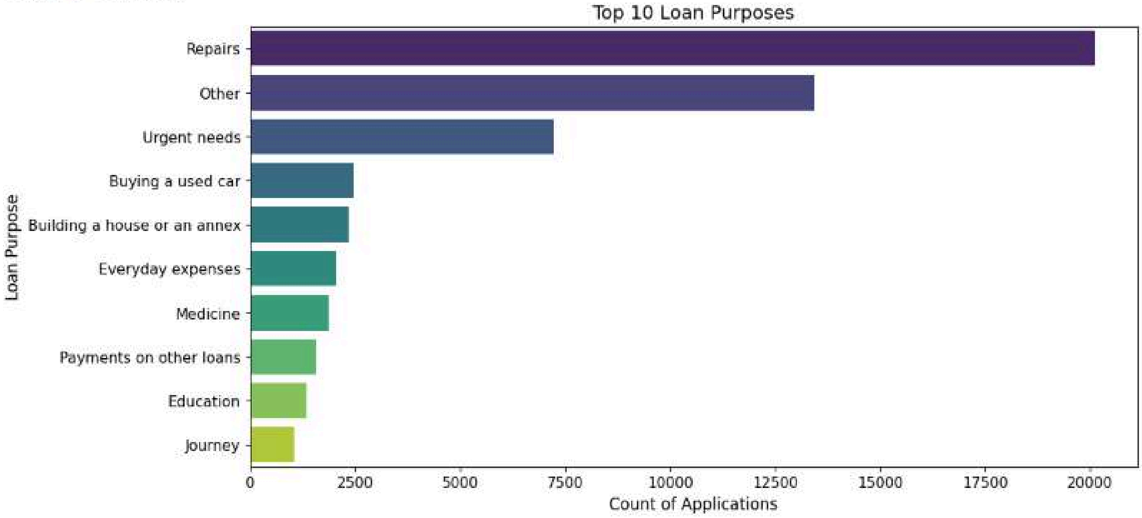
Loan Purpose and Repayment Behavior:

Applicants who take loans for investment purposes (e.g., business, education) may be viewed as lower risk, as they may have the potential to generate income or value from the loan. Conversely, loans for personal consumption (e.g., home renovations, buying goods) could suggest that applicants are less likely to generate an immediate return on investment, making them higher risk.

```
[243] # Exclude 'XAP' and 'XNA' categories from the dataset
      filtered_data = merged_data[~merged_data['NAME_CASH_LOAN_PURPOSE'].isin(['XAP', 'XNA'])]

      # Get the top 5 loan purposes by count after excluding XAP and XNA
      top_10_loan_purposes = filtered_data['NAME_CASH_LOAN_PURPOSE'].value_counts().head(10)

      # Plot the distribution of the top 10 loan purposes horizontally
      plt.figure(figsize=(12, 6))
      sns.barplot(x=top_10_loan_purposes.values, y=top_10_loan_purposes.index, palette='viridis')

      # Title and labels
      plt.title("Top 10 Loan Purposes", fontsize=14)
      plt.xlabel("Count of Applications", fontsize=12)
      plt.ylabel("Loan Purpose", fontsize=12)
      plt.xticks(fontsize=11)
      plt.yticks(fontsize=11)
      plt.show()
```

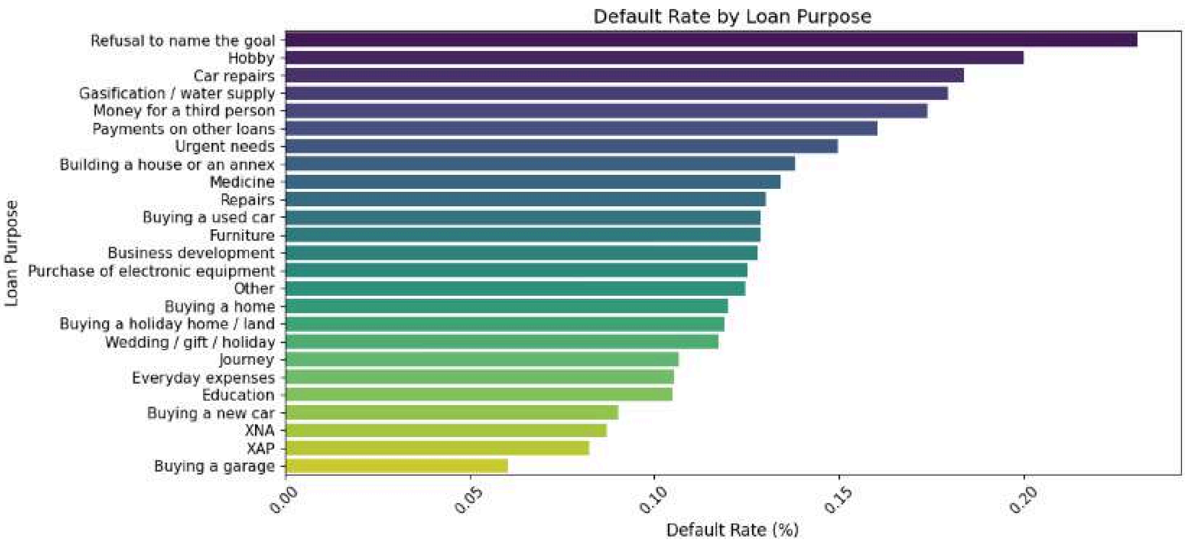
```
<Figure size 1200x600 with 0 Axes>
<ipython-input-243-3f8359c6ad10>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `

  sns.barplot(x=top_10_loan_purposes.values, y=top_10_loan_purposes.index, palette='viridis')
<Axes: ylabel='NAME_CASH_LOAN_PURPOSE'>
Text(0.5, 1.0, 'Top 10 Loan Purposes')
Text(0.5, 0, 'Count of Applications')
Text(0, 0.5, 'Loan Purpose')
(array([    0.,  2500.,  5000.,  7500., 10000., 12500., 15000., 17500.,
        20000., 22500.]),
 [Text(0.0, 0, '0'),
  Text(2500.0, 0, '2500'),
  Text(5000.0, 0, '5000'),
  Text(7500.0, 0, '7500'),
  Text(10000.0, 0, '10000'),
  Text(12500.0, 0, '12500'),
  Text(15000.0, 0, '15000'),
  Text(17500.0, 0, '17500'),
  Text(20000.0, 0, '20000'),
  Text(22500.0, 0, '22500')])
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [Text(0, 0, 'Repairs'),
  Text(0, 1, 'Other'),
  Text(0, 2, 'Urgent needs'),
  Text(0, 3, 'Buying a used car'),
  Text(0, 4, 'Building a house or an annex'),
  Text(0, 5, 'Everyday expenses'),
  Text(0, 6, 'Medicine'),
  Text(0, 7, 'Payments on other loans'),
  Text(0, 8, 'Education'),
  Text(0, 9, 'Journey')])
```

```
Text(0, 9, 'Journey')])
```




```
[244] # Calculate the default rate by loan purpose
loan_purpose_default_rate = merged_data.groupby('NAME_CASH_LOAN_PURPOSE')['TARGET'].mean().reset_index()

# Sort by default rate in descending order
loan_purpose_default_rate = loan_purpose_default_rate.sort_values('TARGET', ascending=False)

# Plot the bar chart
plt.figure(figsize=(12, 6))
sns.barplot(x='TARGET', y='NAME_CASH_LOAN_PURPOSE', data=loan_purpose_default_rate, palette='viridis')

# Title and labels
plt.title("Default Rate by Loan Purpose", fontsize=14)
plt.xlabel("Default Rate (%)", fontsize=12)
plt.ylabel("Loan Purpose", fontsize=12)
plt.xticks(rotation=45, fontsize=11)
plt.yticks(fontsize=11)
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
<ipython-input-244-3f6fed5db76e>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same ef

  sns.barplot(x='TARGET', y='NAME_CASH_LOAN_PURPOSE', data=loan_purpose_default_rate, palette='viridis')
<Axes: xlabel='TARGET', ylabel='NAME_CASH_LOAN_PURPOSE'>
Text(0.5, 1.0, 'Default Rate by Loan Purpose')
Text(0.5, 0, 'Default Rate (%)')
Text(0, 0.5, 'Loan Purpose')
(array([0.  , 0.05, 0.1 , 0.15, 0.2 , 0.25]),
 [Text(0.0, 0, '0.00'),
  Text(0.05, 0, '0.05'),
  Text(0.1, 0, '0.10'),
  Text(0.15000000000000002, 0, '0.15'),
  Text(0.2, 0, '0.20'),
```



Insights: The chart highlights the most popular loan application reasons. When examining default rates by loan purpose, we can see that applicants who don't specify a reason or choose "Hobby" are more likely to default. This suggests that vague or less serious reasons for a loan application could be seen as a red flag by lenders.

Interestingly, it's difficult to understand why "Car repairs" would have a higher default rate compared to "Buying a new car" or "Buying a used car", which have lower default rates. This could indicate that, while the intent for a new or used car might be clearer and more justified, applicants for car repairs may be perceived as having less stable financial behavior, or perhaps the loan amount required for repairs might be lower but not well-supported by their financial situation.

Conclusion

In this analysis, we have explored several key features that contribute to assessing the risk of loan applicants, vs TARGET being our initial focus.

Income Analysis (AMT_INCOME_TOTAL vs. TARGET): We started by examining AMT_INCOME_TOTAL, which plays a central role in assessing an applicant's ability to repay a loan. As expected, higher incomes are correlated with lower default rates. However, this analysis also highlighted that income alone isn't always a reliable indicator, as applicants with varying income levels (low, medium, high, and very high) can still default, showing that other factors must be considered.

Age and Employment History: We found that younger applicants and those with shorter employment histories are more likely to present higher credit risks. Those with stable, long-term employment tend to have a higher chance of repayment, making them more favorable candidates for loan approval.

Family Size (CNT_FAM_MEMBERS): Larger families seem to correlate with higher default rates. A growing family size could result in greater financial strain, reducing the ability to repay loans, especially when the number of earners is lower.

Education: Education level plays an important role in shaping financial stability and behavior. Higher levels of education generally correlate with better financial management skills and income potential. However, we observed that applicants with secondary or higher education levels have both high approval and rejection rates, indicating that other financial factors also heavily influence their loan application success.

Loan Purpose and Default Rates: The NAME_CASH_LOAN_PURPOSE feature helped identify that the reason behind loan requests also contributes to risk assessment. Applicants seeking loans for personal consumption or discretionary spending appear riskier compared to those seeking loans for investments, which may indicate more stable financial intentions.

Rejection Reasons (CODE_REJECT_REASON): Understanding why an application was rejected provides key insights into risk factors. Applicants rejected due to issues like low credit scores, high credit risk, or fraud suspicion tend to have a higher default rate, indicating they are higher-risk candidates.

By analyzing these features and their relationships with TARGET (default rates), we can make more informed, data-driven decisions in the lending process. The AMT_INCOME_TOTAL vs. TARGET analysis was crucial in highlighting that income is an important factor, but it alone is not enough to predict default risk. In conclusion, combining multiple features such as age, family size, education, loan purpose, and rejection reasons provides a holistic approach to credit risk assessment, helping financial institutions optimize their lending strategies and reduce potential losses from defaults.