Home Credit Default Risk (HCDR)

```
# git clone https://github.com/lakshmi-gadhari/hcdr.git
# git clone -b main --single-branch https://github.com/lakshmi-gadhari/hcdr.git
```

The Home Credit Default Risk (HCDR) Kaggle Competition is a machine learning challenge hosted by Home Credit, a company that provides loans to individuals with little or no credit history. The goal of this project is to predict whether or not a client will repay a loan, in order to help ensure that those who struggle to get loan due to insufficient or non-existent credit histories have a positive loan experience. Home Credit makes use of a variety of alternative data--including data and transactional information--to predict their clients' repayment abilities.

Some of the challenges

- Large data
- Imbalanced classes
- Lots of missing values
- 1-2TB of data unprocessed
- Feature engineering
- Feature selection
- Hyperparameter tuning
- Scoring/ranking/prediction bias

Kaggle API setup

Kaggle provides a command line utilities which allows use of datasets, in the past, it was troublesome to submit your result as it involves navigating through the website to upload the submission file. However, this has been solved with Kaggle like the command line. Fig 1.

```
# Ubuntu / Linux / Colab / Cloud / Terminal / Command Line
pip install kaggle
```

It is quite easy to setup, it takes no more than 5 minutes to finish a submission.

1. Install library
2. Create a API Token (will save a file with extension `kaggle.json`), this advises `kaggle.json`. Fig 2.
3. Put your JSON `kaggle.json` in the right place
4. Run a few commands (`competition list`, `competition download`, `competition submit` commands examples below)
5. Submit result

For more detailed information on setting the Kaggle API see here and here.

```
# Create a directory, kaggle
mkdir ~/.kaggle
# Move API token to new directory
mv ~/kaggle.json ~/.kaggle/
# Ensure the API token is secure
chmod 600 ~/.kaggle/kaggle.json
# Install the latest version of the Kaggle API
pip install --upgrade --force-reinstall --no-deps kaggle
# Verify installation
kaggle --version
# Download competition data
kaggle competitions download -c home-credit-default-risk
# Download specific dataset
kaggle datasets download -d dataset-owner/dataset-name
# Download and unzip automatically
kaggle competitions download -c home-credit-default-risk -p ./data --unzip
# Submit to competition
kaggle competitions submit -c home-credit-default-risk -f submission.csv -m "Model v1"
# Check your submissions
kaggle competitions submissions home-credit-default-risk
# Check leaderboard
kaggle competitions leaderboard home-credit-default-risk --show
```

Team and project meta information.

Project Title : Home Credit Default Risk

Group Number : 27

Team Member : Vinay Reddy Polagani - vpolagani@sdsu.edu

- GitHub: `Rachaneh Sahasra` / `vinayreddy-sdsu`
- Phone Number : 619-431-7103
- Name: Rachaneh Sahasra - rsahasra@sdsu.edu

Dataset and how to download

Back ground Home Credit Group

Many people struggle to get loan due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by predatory lenders.

Home Credit Group

Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order make this possible, they utilize various alternative data--including telecommunication and transactional information--to assess their clients' repayment abilities. Although they are willing to lend to clients who have little or no credit history, they want to ensure that clients capable of repayment are not rejected and that their loans are given with a positive impact, thereby, and repayment calendar that will empower their clients to be successful.

Background on the dataset

Home Credit is a non-bank financial institution, founded in 1997 in the Czech Republic.

The company operates in 14 countries, including United States, Russia, Kazakhstan, Belarus, China, India and Vietnam as lending primarily to people with little or no credit histories which will either not obtain loans or become victims of unsecured lenders.

Home Credit has also expressed interest in unsecured lenders, total assets of 2 Billions Euro, over 160 Millions loans, with the majority in Asia and almost half of that in China (as of 31 December 2018).

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kaggle hackers to see if they can further improve the scorecards of their data. Doing so will ensure their clients capable of repayment are not rejected and that Home Credit can provide a positive, rewarding, and repayment calendar that will empower their clients be successful.

Data files overview

There are 7 different sources of data:

- **application_{train|test}.csv**: This is the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by feature SK_ID_CURR. The training application data comes with the TARGET indicating 0: the loan was repaid or, 1: the loan was not repaid. The target variable defines if the client had payment difficulties on the loan for several of the first few installments of the loan. This case is identified when there is a payment made more than the first installment of the loan. In all other cases, the target is set to 0.
- **bureau**.csv: Data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but many loans in the application data can have multiple previous loans.
- **bureau_balance**.csv: Monthly data about the previous credits in bureau. Each row or one month of a previous credit, and a single previous credit can have many rows in bureau_balance, one for each month of history.
- **previous_application**.csv: Previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have many previous application records in previous_application, one for each prior application, or none if this is the first application.
- **POS_CASH_BALANCE**: Monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous point of sale or cash loan can have many rows.
- **credit_card_balance**: Monthly balance data of previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows in credit_card_balance.
- **installments_payment**: Payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

```
# Clone the repo
git clone https://github.com/lakshmi-gadhari/hcdr.git
# Navigate to the data directory
cd hcdr/data/raw
# Verify files are present
ls -lh *.csv
# Count lines in all files
wc -l *.csv
# Create a backup (optional)
mkdir backup
cp *.csv backup/
# Compress for storage (optional)
gzip *.csv
# Later, decompress when needed
gzip -d *.csv.gz
```

```
# Install required packages
pip install pandas numpy matplotlib seaborn
# Start Jupyter
jupyter notebook
# Or for JupyterLab
jupyter lab
```

```
# Basic imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Load application_train.csv
df = pd.read_csv('application_train.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum() / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts())
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
# Load application_train.csv
df = pd.read_csv('application_train.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum() / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts())
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (307511, 122)
Memory usage: 286.23102188 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_SUM
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2 MB
Target distribution:
TARGET
0 282686
1 24825
Name: count, dtype: int64
Target percentage:
TARGET
0 0.919271
1 0.080729
Name: proportion, dtype: float64
Columns with missing values: 67
COMMONAREA_MEDI 214865
COMMONAREA_AVG 214865
COMMONAREA_MODE 214865
NONLIVINGAPARTMENTS_MEDI 213514
NONLIVINGAPARTMENTS_AVG 213514
NONLIVINGAPARTMENTS_MODE 213514
FONDKAPREMONT_MODE 210295
LIVINGAPARTMENTS_MEDI 210199
LIVINGAPARTMENTS_AVG 210199
LIVINGAPARTMENTS_MODE 210199
FLOORSMIN_MEDI 208642
FLOORSMIN_AVG 208642
FLOORSMIN_MODE 208642
YEARS_BUILD_MEDI 204488
YEARS_BUILD_AVG 204488
YEARS_BUILD_MODE 204488
OWN_CAR_AGE 202929
LANDAREA_MEDI 182590
LANDAREA_AVG 182590
LANDAREA_MODE 182590
```

Data files overview

Data Dictionary

As part of the data download comes a Data Dictionary. It's named `HOME_CREDIT_DATA_DESCRIPTION.csv`.

Application train

```
# Load application_train.csv
df = pd.read_csv('application_train.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum() / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (307511, 122)
Memory usage: 286.23102188 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_SUM
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2 MB
Target distribution:
TARGET
0 282686
1 24825
Name: count, dtype: int64
Target percentage:
TARGET
0 0.919271
1 0.080729
Name: proportion, dtype: float64
Columns with missing values: 67
COMMONAREA_MEDI 214865
COMMONAREA_AVG 214865
COMMONAREA_MODE 214865
NONLIVINGAPARTMENTS_MEDI 213514
NONLIVINGAPARTMENTS_AVG 213514
NONLIVINGAPARTMENTS_MODE 213514
FONDKAPREMONT_MODE 210295
LIVINGAPARTMENTS_MEDI 210199
LIVINGAPARTMENTS_AVG 210199
LIVINGAPARTMENTS_MODE 210199
FLOORSMIN_MEDI 208642
FLOORSMIN_AVG 208642
FLOORSMIN_MODE 208642
YEARS_BUILD_MEDI 204488
YEARS_BUILD_AVG 204488
YEARS_BUILD_MODE 204488
OWN_CAR_AGE 202929
LANDAREA_MEDI 182590
LANDAREA_AVG 182590
LANDAREA_MODE 182590
```

Application test

```
# Load application_test.csv
df = pd.read_csv('application_test.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (48744, 121)
Memory usage: 45.1 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_SUM
dtypes: float64(65), int64(40), object(16)
memory usage: 45.1 MB
Target distribution:
TARGET
0 48744
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 64
COMMONAREA_MEDI 33495
COMMONAREA_AVG 33495
COMMONAREA_MODE 33495
NONLIVINGAPARTMENTS_MEDI 33347
NONLIVINGAPARTMENTS_AVG 33347
NONLIVINGAPARTMENTS_MODE 33347
FONDKAPREMONT_MODE 32797
LIVINGAPARTMENTS_MEDI 32780
LIVINGAPARTMENTS_AVG 32780
LIVINGAPARTMENTS_MODE 32780
FLOORSMIN_MEDI 32466
FLOORSMIN_AVG 32466
FLOORSMIN_MODE 32466
YEARS_BUILD_MEDI 32312
YEARS_BUILD_AVG 32312
YEARS_BUILD_MODE 32312
OWN_CAR_AGE 32312
LANDAREA_MEDI 29726
LANDAREA_AVG 29726
LANDAREA_MODE 29726
```

The Other datasets

- **bureau**.csv: Data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but many loans in the application data can have multiple previous credits.
- **bureau_balance**.csv: Monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have many rows in bureau_balance, one for each month of history.
- **previous_application**.csv: Previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one and only one associated loan in POS_CASH_balance, credit_card_balance, and installments_payments.
- **POS_CASH_balance**.csv: Monthly balance data of previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous point of sale or cash loan can have many rows.
- **credit_card_balance**.csv: Monthly balance data of previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows in credit_card_balance.
- **installments_payments**.csv: Payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

```
# Load bureau.csv
df = pd.read_csv('bureau.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (1716428, 17)
Memory usage: 223.4 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Columns: 17 entries, SK_ID_CURR to AMT_ANNUITY
dtypes: float64(8), int64(6), object(3)
memory usage: 223.4 MB
Target distribution:
TARGET
0 1716428
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 5
AMT_ANNUITY 1226791
AMT_CREDIT_MAX_OVERDUE 1124488
DAYS_ENDDATE_FACT 633653
AMT_CREDIT_SUM_LIMIT 591780
AMT_CREDIT_SUM_DEBT 257669
```

```
# Load bureau_balance.csv
df = pd.read_csv('bureau_balance.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (27299925, 3)
Memory usage: 1635.1 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Columns: 3 entries, SK_ID_BUREAU to STATUS
dtypes: int64(2), object(1)
memory usage: 1635.1 MB
Target distribution:
TARGET
0 27299925
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 0
```

```
# Load previous_application.csv
df = pd.read_csv('previous_application.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (1670214, 37)
Memory usage: 471.4 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Columns: 37 entries, SK_ID_PREV to NFLAG_INSURED_ON_APPROVAL
dtypes: float64(15), int64(6), object(16)
memory usage: 471.4 MB
Target distribution:
TARGET
0 1670214
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 16
RATE_INTEREST_PRIVILEGED 1664263
RATE_INTEREST_PRIMARY 1664263
AMT_DOWN_PAYMENT 895844
RATE_DOWN_PAYMENT 895844
AMT_GOODS_PRICE 385515
PRODUCT_COMBINATION 346011
NAME_TYPE_SUITE 820405
CNT_PAYMENT 372230
NAME_PRODUCT_TYPE 517215
CHANNEL_TYPE 517215
SELLERPLACE_AREA 517215
NAME_SELLER_INDUSTRY 517215
NAME_YIELD_GROUP 517215
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
```

```
# Load POS_CASH_balance.csv
df = pd.read_csv('POS_CASH_balance.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (10001358, 8)
Memory usage: 76.3 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 10001358 entries, 0 to 10001357
Columns: 8 entries, SK_ID_PREV to SK_DPD_DEF
dtypes: int64(8)
memory usage: 76.3 MB
Target distribution:
TARGET
0 10001358
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 0
```

```
# Load credit_card_balance.csv
df = pd.read_csv('credit_card_balance.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (3840312, 23)
Memory usage: 673.2 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Columns: 23 entries, SK_ID_PREV to SK_DPD_DEF
dtypes: int64(19), object(4)
memory usage: 673.2 MB
Target distribution:
TARGET
0 3840312
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 5
AMT_DRAWINGS_ATM_CURRENT 749816
AMT_DRAWINGS_OTHER_CURRENT 749816
AMT_DRAWINGS_POS_CURRENT 749816
AMT_INSTALMENT_MINIMUM 305236
AMT_PAYMENT_TOTAL_CURRENT 767988
```

```
# Load installments_payments.csv
df = pd.read_csv('installments_payments.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (13605401, 8)
Memory usage: 103.8 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Columns: 8 entries, SK_ID_PREV to AMT_PAYMENT
dtypes: int64(8)
memory usage: 103.8 MB
Target distribution:
TARGET
0 13605401
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 2
DAYS_ENTRY_PAYMENT 2905
AMT_PAYMENT 2905
```

Exploratory Data Analysis

Summary of Application train

```
# Load application_train.csv
df = pd.read_csv('application_train.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (307511, 122)
Memory usage: 286.23102188 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_SUM
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2 MB
Target distribution:
TARGET
0 282686
1 24825
Name: count, dtype: int64
Target percentage:
TARGET
0 0.919271
1 0.080729
Name: proportion, dtype: float64
Columns with missing values: 67
COMMONAREA_MEDI 214865
COMMONAREA_AVG 214865
COMMONAREA_MODE 214865
NONLIVINGAPARTMENTS_MEDI 213514
NONLIVINGAPARTMENTS_AVG 213514
NONLIVINGAPARTMENTS_MODE 213514
FONDKAPREMONT_MODE 210295
LIVINGAPARTMENTS_MEDI 210199
LIVINGAPARTMENTS_AVG 210199
LIVINGAPARTMENTS_MODE 210199
FLOORSMIN_MEDI 208642
FLOORSMIN_AVG 208642
FLOORSMIN_MODE 208642
YEARS_BUILD_MEDI 204488
YEARS_BUILD_AVG 204488
YEARS_BUILD_MODE 204488
OWN_CAR_AGE 202929
LANDAREA_MEDI 182590
LANDAREA_AVG 182590
LANDAREA_MODE 182590
```

Summary of Application test

```
# Load application_test.csv
df = pd.read_csv('application_test.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (48744, 121)
Memory usage: 45.1 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_SUM
dtypes: float64(65), int64(40), object(16)
memory usage: 45.1 MB
Target distribution:
TARGET
0 48744
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 64
COMMONAREA_MEDI 33495
COMMONAREA_AVG 33495
COMMONAREA_MODE 33495
NONLIVINGAPARTMENTS_MEDI 33347
NONLIVINGAPARTMENTS_AVG 33347
NONLIVINGAPARTMENTS_MODE 33347
FONDKAPREMONT_MODE 32797
LIVINGAPARTMENTS_MEDI 32780
LIVINGAPARTMENTS_AVG 32780
LIVINGAPARTMENTS_MODE 32780
FLOORSMIN_MEDI 32466
FLOORSMIN_AVG 32466
FLOORSMIN_MODE 32466
YEARS_BUILD_MEDI 32312
YEARS_BUILD_AVG 32312
YEARS_BUILD_MODE 32312
OWN_CAR_AGE 32312
LANDAREA_MEDI 29726
LANDAREA_AVG 29726
LANDAREA_MODE 29726
```

```
# Load bureau.csv
df = pd.read_csv('bureau.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (1716428, 17)
Memory usage: 223.4 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Columns: 17 entries, SK_ID_CURR to AMT_ANNUITY
dtypes: float64(8), int64(6), object(3)
memory usage: 223.4 MB
Target distribution:
TARGET
0 1716428
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 5
AMT_ANNUITY 1226791
AMT_CREDIT_MAX_OVERDUE 1124488
DAYS_ENDDATE_FACT 633653
AMT_CREDIT_SUM_LIMIT 591780
AMT_CREDIT_SUM_DEBT 257669
```

```
# Load bureau_balance.csv
df = pd.read_csv('bureau_balance.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (27299925, 3)
Memory usage: 1635.1 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Columns: 3 entries, SK_ID_BUREAU to STATUS
dtypes: int64(2), object(1)
memory usage: 1635.1 MB
Target distribution:
TARGET
0 27299925
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 0
```

```
# Load previous_application.csv
df = pd.read_csv('previous_application.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (1670214, 37)
Memory usage: 471.4 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Columns: 37 entries, SK_ID_PREV to NFLAG_INSURED_ON_APPROVAL
dtypes: float64(15), int64(6), object(16)
memory usage: 471.4 MB
Target distribution:
TARGET
0 1670214
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 16
RATE_INTEREST_PRIVILEGED 1664263
RATE_INTEREST_PRIMARY 1664263
AMT_DOWN_PAYMENT 895844
RATE_DOWN_PAYMENT 895844
AMT_GOODS_PRICE 385515
PRODUCT_COMBINATION 346011
NAME_TYPE_SUITE 820405
CNT_PAYMENT 372230
NAME_PRODUCT_TYPE 517215
CHANNEL_TYPE 517215
SELLERPLACE_AREA 517215
NAME_SELLER_INDUSTRY 517215
NAME_YIELD_GROUP 517215
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
```

```
# Load POS_CASH_balance.csv
df = pd.read_csv('POS_CASH_balance.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (10001358, 8)
Memory usage: 76.3 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 10001358 entries, 0 to 10001357
Columns: 8 entries, SK_ID_PREV to SK_DPD_DEF
dtypes: int64(8)
memory usage: 76.3 MB
Target distribution:
TARGET
0 10001358
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 0
```

```
# Load credit_card_balance.csv
df = pd.read_csv('credit_card_balance.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (3840312, 23)
Memory usage: 673.2 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Columns: 23 entries, SK_ID_PREV to SK_DPD_DEF
dtypes: int64(19), object(4)
memory usage: 673.2 MB
Target distribution:
TARGET
0 3840312
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 5
AMT_DRAWINGS_ATM_CURRENT 749816
AMT_DRAWINGS_OTHER_CURRENT 749816
AMT_DRAWINGS_POS_CURRENT 749816
AMT_INSTALMENT_MINIMUM 305236
AMT_PAYMENT_TOTAL_CURRENT 767988
```

```
# Load installments_payments.csv
df = pd.read_csv('installments_payments.csv')
# Basic exploration
print("Shape:", df.shape)
print("Memory usage:", df.memory_usage().sum / 1024**2, "MB")
df.info()
df.head()
df.describe()
# Check target distribution
print("Target distribution:")
print(df['TARGET'].value_counts)
print("Target percentage:")
print(df['TARGET'].value_counts(normalize=True))
# Check for missing values
missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print("Columns with missing values:", len(missing))
print(missing.head(20))
```

```
Shape: (13605401, 8)
Memory usage: 103.8 MB
<class'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Columns: 8 entries, SK_ID_PREV to AMT_PAYMENT
dtypes: int64(8)
memory usage: 103.8 MB
Target distribution:
TARGET
0 13605401
Name: count, dtype: int64
Target percentage:
TARGET
0 1.0
Name: proportion, dtype: float64
Columns with missing values: 2
DAYS_ENTRY_PAYMENT 2905
AMT_PAYMENT 2905
```

Exploratory Data Analysis

Summary of Application train

```
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	AMO
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	3075
mean	278180.518577	0.080729	0.417052	1.687979e+05	599026.000000	27108.573909	538396.207429	0.00
std	102790.175348	0.272419	0.722269	2.371231e+05	402490.776996	14493.737315	369446.460540	0.04
min	100002.000000	0.000000	0.000000	2.565000e+04	45000.000000	1615.500000	40500.000000	0.00
25%	189145.500000	0.000000	0.000000	1.125000e+05	270000.000000	16524.000000	238500.000000	0.00
50%	278202.000000	0.000000	0.000000	1.471500e+05	513531.000000	24903.000000	450000.000000	0.00
75%	367142.500000	0.000000	1.000000	2.025000e+05	808650.000000	34596.000000	679500.000000	0.00
max	456255.000000	1.000000	19.000000	117000000.000000	4050000.000000	258025.500000	4050000.000000	1.00

8 rows × 106 columns

```
data_train.describe()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCO
count	307511	307511	307511	307511	307511	307511	307511	307511
2	100002	1	Cash loans	M	N	Y	0	202500
3	100003	0	Revolving loans	F	Y	N	1	135000
top	100004	0	Revolving loans	F	N	Y	0	112500
freq	2	282686	307021	202448	202924	213312	215371	26841

4 rows × 122 columns

```
data_test.describe()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL
count	48744	48744	48744	48744	48744	48744	48744	48744
2	100001	0	Cash loans	F	N	Y	0	135000
top	100038	0	Cash loans	F	N	Y	0	112500
freq	2	48744	48744	32678	32678	32678	32678	8781

4 rows × 121 columns

```
data_train.shape, data_test.shape
```

```
((307511, 122), (48744, 121))
```

```
data_train.dtypes.value_counts
```

```
float64    65
int64      41
object     16
dtype: int64
```

```
data_test.dtypes.value_counts
```

```
float64    64
int64      40
object     16
dtype: int64
```

```
data_train.isnull.sum.sort_values(ascending=False)[:20]
```

```
COMMONAREA_MEDI           214865
COMMONAREA_AVG            214865
COMMONAREA_MODE           214865
NONLIVINGAPARTMENTS_MEDI  213514
NONLIVINGAPARTMENTS_AVG   213514
NONLIVINGAPARTMENTS_MODE  213514
FONDKAPREMONT_MODE        210295
LIVINGAPARTMENTS_MEDI     210199
LIVINGAPARTMENTS_AVG      210199
LIVINGAPARTMENTS_MODE     210199
FLOORSMIN_MEDI            208642
FLOORSMIN_AVG             208642
FLOORSMIN_MODE            208642
YEARS_BUILD_MEDI          204488
YEARS_BUILD_AVG           204488
YEARS_BUILD_MODE          204488
OWN_CAR_AGE               202929
LANDAREA_MEDI             182590
LANDAREA_AVG              182590
LANDAREA_MODE             182590
dtype: int64
```

Missing data for application-train

```
def missing_values_table(df):
    mis_val = df.isnull.sum
    mis_val_percent = 100 * df.isnull.sum / len(df)
    mis_val_table = pd.concat(mis_val, mis_val_percent)
    mis_val_table_ren_columns = mis_val_table.rename(
    {0 : 'Missing Values', 1 : '% of Total Values'})
    mis_val_table_ren_columns = mis_val_table_ren_columns[
        mis_val_table_ren_columns.iloc[:,1] != 0].sort_values('% of Total Values', ascending=False).round(1)
    print("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
        "There are " + str(mis_val_table_ren_columns.shape[0]) +
          " columns that have missing values.")
    return mis_val_table_ren_columns
```

```
missing_table_train = missing_values_table(data_train)
```

```
Your selected dataframe has 122 columns.
There are 67 columns that have missing values.
```

```
missing_table_train.head(20)
```

	Missing Values	% of Total Values
COMMONAREA_MEDI	214865	69.9
COMMONAREA_AVG	214865	69.9
COMMONAREA_MODE	214865	69.9
NONLIVINGAPARTMENTS_MEDI	213514	69.4
NONLIVINGAPARTMENTS_AVG	213514	69.4
NONLIVINGAPARTMENTS_MODE	213514	69.4
FONDKAPREMONT_MODE	210295	68.4
LIVINGAPARTMENTS_MEDI	210199	68.3
LIVINGAPARTMENTS_AVG	210199	68.3
LIVINGAPARTMENTS_MODE	210199	68.3
FLOORSMIN_MEDI	208642	67.8
FLOORSMIN_AVG	208642	67.8
FLOORSMIN_MODE	208642	67.8
YEARS_BUILD_MEDI	204488	66.5
YEARS_BUILD_AVG	204488	66.5
YEARS_BUILD_MODE	204488	66.5
OWN_CAR_AGE	202929	66.0
LANDAREA_MEDI	182590	59.4
LANDAREA_AVG	182590	59.4
LANDAREA_MODE	182590	59.4

Distribution of the target column

```
plot_target_hist(data_train, 'TARGET', 'Distribution of the target column')
```

Correlation with the target column

```
correlations = data_train.corr['TARGET'].sort_values(ascending=False)
print('Most Positive Correlations:', correlations[:15],
      'Most Negative Correlations:', correlations[-15:])
```

```
Most Positive Correlations: DAYS_BIRTH                    0.078239
REGION_RATING_CLIENT                0.058899
REGION_RATING_CLIENT_W_CITY         0.057686
DAYS_LAST_PHONE_CHANGE              0.055461
DAYS_ID_PUBLISH                     0.051457
REG_CITY_NOT_WORK_CITY              0.050994
FLAG_DOCUMENT_3                     0.044346
DAYS_REGISTRATION                   0.041975
OWN_CAR_AGE                         0.037612
LIVE_CITY_NOT_WORK_CITY             0.032518
DEF_30_CNT_SOCIAL_CIRCLE            0.032248
DEF_60_CNT_SOCIAL_CIRCLE            0.031276
FLAG_DOCUMENT_6                     0.028881
REG_CITY_NOT_LIVE_CITY              0.027139
FLAG_EMAIL                          0.027100
Name: TARGET, dtype: float64
Most Negative Correlations: EXT_SOURCE_3                   -0.178919
EXT_SOURCE_2                   -0.160472
EXT_SOURCE_1                   -0.155317
NONLIVINGAREA_AVG              -0.074957
NONLIVINGAREA_MEDI             -0.074314
NONLIVINGAREA_MODE             -0.072364
ELEVATORS_AVG                  -0.034199
ELEVATORS_MEDI                 -0.033863
ELEVATORS_MODE                 -0.032131
AMT_GOODS_PRICE                -0.039645
FLOOR                          -0.044003
EMERGENCYSTATE_MODE            -0.045982
HOUSETYPE_MODE                 -0.047848
AGE_RANGE                      -0.057490
Name: TARGET, dtype: float64
```


Applicants Age

```
plot_distribution('Applicants Age', data_train['DAYS_BIRTH'] / -365, '', 'Years')
```


Applicants occupations

```
plot_stats('OCCUPATION_TYPE', data_train)
```

```
# Age
plt.rcParams['figure.dpi'] = 100
print('Oldest Applicant:',
      int(np.max(np.abs(df.loc[df['DAYS_BIRTH'] < 0, 'DAYS_BIRTH'] / 365.25))),
      'years')
plt.show()
```



```
# Income
plt.rcParams['figure.dpi'] = 100
sns.boxplot(x='TARGET', y='AMT_INCOME_TOTAL', data=df[df.AMT_INCOME_TOTAL < 500000])
plt.title('Income by Loan Status')
plt.show()
```



```
# Credit
plt.rcParams['figure.dpi'] = 100
sns.boxplot(x='TARGET', y='AMT_CREDIT', data=df[df.AMT_CREDIT < 2000000])
plt.title('Credit by Loan Status')
plt.show()
```



```
# Contract Type
plt.rcParams['figure.dpi'] = 100
sns.countplot(x='NAME_CONTRACT_TYPE', hue='TARGET', data=df)
plt.title('Contract Type by Loan Outcome')
plt.show()
```


Observation: From the above chart we can see that the amount borrower type of 18 and 40 has maximum number who are at risk

```
# Gender
plt.rcParams['figure.dpi'] = 100
sns.countplot(x='CODE_GENDER', hue='TARGET', data=df[df.CODE_GENDER != 'XNA'])
plt.title('Gender by Loan Outcome')
plt.show()
```



```
# Education
plt.rcParams['figure.dpi'] = 100
sns.countplot(x='NAME_EDUCATION_TYPE', hue='TARGET', data=df)
plt.title('Education by Loan Outcome')
plt.xticks(rotation=45)
plt.show()
```



```
# Housing
plt.rcParams['figure.dpi'] = 100
sns.countplot(x='NAME_HOUSING_TYPE', hue='TARGET', data=df)
plt.title('Housing Type by Loan Outcome')
plt.xticks(rotation=45)
plt.show()
```



```
# Occupation
plt.rcParams['figure.dpi'] = 100
sns.countplot(x='OCCUPATION_TYPE', hue='TARGET', data=df[df.OCCUPATION_TYPE != 'NaN'])
plt.title('Occupation by Loan Outcome')
plt.xticks(rotation=45)
plt.show()
```



```
# Contract Type
plt.rcParams['figure.dpi'] = 100
sns.countplot(x='NAME_CONTRACT_TYPE', hue='TARGET', data=df)
plt.title('Contract Type by Loan Outcome')
plt.show()
```



```
# Gender
plt.rcppi = 100
sns.countplot(x='CODE_GENDER', hue='TARGET', data=df[df.CODE_GENDER != 'XNA'])
plt.title('Gender by Loan Outcome')
plt.show()
```



```
# Education
plt.rcppi = 100
sns.countplot(x='NAME_EDUCATION_TYPE', hue='TARGET', data=df)
plt.title('Education by Loan Outcome')
plt.xticks(rotation=45)
plt.show()
```



```
# Housing
plt.rcppi = 100
sns.countplot(x='NAME_HOUSING_TYPE', hue='TARGET', data=df)
plt.title('Housing Type by Loan Outcome')
plt.xticks(rotation=45)
plt.show()
```



```
# Occupation
plt.rcppi = 100
sns.countplot(x='OCCUPATION_TYPE', hue='TARGET', data=df[df.OCCUPATION_TYPE != 'NaN'])
plt.title('Occupation by Loan Outcome')
plt.xticks(rotation=45)
plt.show()
```


Summary of bureau

```
df.shape (307511, 122)
df.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: bool(3), category(16), datetime64[ns, UTC](1), float64(65), int64(41), object(16)
memory usage: 1.0 GB
```

```
DESCRIBE "home_credit_default_risk".."application_train";
```

```
Connected to home_credit_default_risk
Processing Command(s) at line 1
Release 18.0.0.0.0 - Production on Fri Jun 13 20:20:42 2025
Version 18.0.0.0.0
Copyright (c) 1982, 2022, Oracle and/or affiliates.
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL>
Disconnected from home_credit_default_risk
```

```
DESCRIBE "home_credit_default_risk".."bureau";
```

```
Connected to home_credit_default_risk
Processing Command(s) at line 1

SQL_ID_BUCKET  BUCKETS_BALANCE  SUM(CB
-------------- --------------- -------
1              1               1
2              2               2
3              3               3
4              4               4
5              5               5

SQL>
Disconnected from home_credit_default_risk
```

```
DESCRIBE "home_credit_default_risk".."bureau_balance";
```

```
Connected to home_credit_default_risk
Processing Command(s) at line 1

SQL_ID_BUCKET  BUCKETS_BALANCE  SUM(CB
-------------- --------------- -------
1              1               1
2              2               2
3              3               3
4              4               4
5              5               5

SQL>
Disconnected from home_credit_default_risk
```

```
DESCRIBE "home_credit_default_risk".."previous_application";
```

```
Connected to home_credit_default_risk
Processing Command(s) at line 1
Warning: Command processing failed.
ORA-00942: table or view does not exist

SQL_ID_BUCKET  BUCKETS_BALANCE  SUM(CB
-------------- --------------- -------
1              1               1
2              2               2
3              3               3
4              4               4
5              5               5

SQL>
Disconnected from home_credit_default_risk
```

```
DESCRIBE "home_credit_default_risk".."POS_CASH_BALANCE";
```

```
Connected to home_credit_default_risk
Processing Command(s) at line 1
Warning: Command processing failed.
ORA-00942: table or view does not exist

SQL_ID_BUCKET  BUCKETS_BALANCE  SUM(CB
-------------- --------------- -------
1              1               1
2              2               2
3              3               3
4              4               4
5              5               5

SQL>
Disconnected from home_credit_default_risk
```

```
DESCRIBE "home_credit_default_risk".."credit_card_balance";
```

```
Connected to home_credit_default_risk
Processing Command(s) at line 1
Warning: Command processing failed.
ORA-00942: table or view does not exist

SQL_ID_BUCKET  BUCKETS_BALANCE  SUM(CB
-------------- --------------- -------
1              1               1
2              2               2
3              3               3
4              4               4
5              5               5

SQL>
Disconnected from home_credit_default_risk
```

```
DESCRIBE "home_credit_default_risk".."installments_payments";
```

```
Connected to home_credit_default_risk
Processing Command(s) at line 1
Warning: Command processing failed.
ORA-00942: table or view does not exist

SQL_ID_BUCKET  BUCKETS_BALANCE  SUM(CB
-------------- --------------- -------
1              1               1
2              2               2
3              3               3
4              4               4
5              5               5

SQL>
Disconnected from home_credit_default_risk
```

Summary of Previous_application

```
DESCRIBE "home_credit_default_risk".."previous_application";
```

```
Connected to home_credit_default_risk
Processing Command(s) at line 1
Warning: Command processing failed.
ORA-00942: table or view does not exist

Name                          Null?    Type
----------------------------- -------- ----
SK_ID_PREV                    NOT NULL NUMBER(10)
SK_ID_CURR                    NOT NULL NUMBER(10)
NAME_CONTRACT_TYPE            NOT NULL VARCHAR2(255 CHAR)
AMT_ANNUITY                   NOT NULL NUMBER(19,4)
AMT_APPLICATION               NOT NULL NUMBER(19,4)
AMT_CREDIT                    NOT NULL NUMBER(19,4)
AMT_DOWN_PAYMENT              NOT NULL NUMBER(19,4)
AMT_GOODS_PRICE               NOT NULL NUMBER(19,4)
WEEKDAY_APPR_PROCESS_START    NOT NULL VARCHAR2(255 CHAR)
HOUR_APPR_PROCESS_START       NOT NULL NUMBER(10)
FLAG_LAST_APPL_PER_CONTRACT   NOT NULL VARCHAR2(255 CHAR)
NFLAG_LAST_APPL_IN_DAY        NOT NULL NUMBER(10)
NAME_CASH_LOAN_PURPOSE        NOT NULL VARCHAR2(255 CHAR)
NAME_CONTRACT_STATUS          NOT NULL VARCHAR2(255 CHAR)
DAYS_DECISION                 NOT NULL NUMBER(10)
NAME_PAYMENT_TYPE             NOT NULL VARCHAR2(255 CHAR)
CODE_REJECT_REASON            NOT NULL VARCHAR2(255 CHAR)
NAME_TYPE_SUITE               NOT NULL VARCHAR2(255 CHAR)
NAME_CLIENT_TYPE              NOT NULL VARCHAR2(255 CHAR)
NAME_GOODS_CATEGORY           NOT NULL VARCHAR2(255 CHAR)
NAME_PORTFOLIO                NOT NULL VARCHAR2(255 CHAR)
NAME_PRODUCT_TYPE             NOT NULL VARCHAR2(255 CHAR)
CHANNEL_TYPE                  NOT NULL VARCHAR2(255 CHAR)
SELLERPLACE_AREA              NOT NULL NUMBER(10)
NAME_SELLER_INDUSTRY          NOT NULL VARCHAR2(255 CHAR)
CNT_PAYMENT                   NOT NULL NUMBER(19,4)
NAME_YIELD_GROUP              NOT NULL VARCHAR2(255 CHAR)
PRODUCT_COMBINATION           NOT NULL VARCHAR2(255 CHAR)
DAYS_FIRST_DRAWING            NOT NULL NUMBER(19,4)
DAYS_FIRST_DUE                NOT NULL NUMBER(19,4)
DAYS_LAST_DUE_1ST_VERSION     NOT NULL NUMBER(19,4)
DAYS_LAST_DUE                 NOT NULL NUMBER(19,4)
DAYS_TERMINATION              NOT NULL NUMBER(19,4)
NFLAG_INSURED_ON_APPROVAL     NOT NULL NUMBER(19,4)

SQL>
Disconnected from home_credit_default_risk
```

```
DESCRIBE "home_credit_default_risk".."previous_application";
```

```
Connected to home_credit_default_risk
Processing Command(s) at line 1

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  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
----------- ----------- ------------------- ------------ ---------------- ----------- ------------------ ------------------ ---------------------------- ------------------------ ---------------------------- ------------------------ ------------------------ ---------------------- -------------- -------------------- -------------------- ---------------- ------------------- ------------------------ ---------------- ------------------- ------------ ------------------- ------------------------ ------------ ------------------- ---------------------- ---------------- ---------------- ---------------------------- ---------------- ---------------- ----------------------------
2030495     271877      Consumer loans     1730.43      18076.95         17976.15    2000.025           18076.95           SATURDAY                         15                        Y                            1                        1                        XAP                    Approved               -73               Cash through the bank  XAP                  Other_B           Mobile            POS               XNA                    Stone                  12           low_normal        Cash X Sales: low    365.243             -73              -43                        300              -77              0
2030496     099881      Cash loans         7875.42      60639.36         68054.34    6890.475           60639.36           THURSDAY                         11                        N                            1                        0                        XAP                    Approved               -544              Cash through the bank  XAP                  Other_B           Middle            POS               XNA                    Stone                  34           low_normal        Cash X Sales: low    365.243             -544             -494                       -488             -488             0
2030497     099882      Cash loans         4643.76      28098.18         28098.18    0.0                4500.0             WEDNESDAY                        11                        N                            1                        0                        XAP                    Approved               -690              Cash through the bank  XAP                  Other_B           Middle            POS               XNA                    Stone                  12           low_normal        Cash X Sales: low    365.243             -690             -360                       -360             -360             0

3 rows 1-3 of 1,670,214

SQL_ID_PREV  SQL
```

previous applications for the submission file

The period is the maple underneath the item had previous applications in the `previous_application.csv`. ~ 47% not on file, 54% people had had previous applications.

```
%matplotlib inline

import matplotlib.pyplot as plt

ipw.sep_columns(df[target], ipw.sep(df[target], 'previous_application'))
```

```
[ipw.sep_columns(df.between('Previous_application', 'FLAG_OWN_CAR'), ipw.sep(df.between('Previous_application', 'FLAG_OWN_CAR'), 'previous_application'))]
```

```
df.shape
```

```
(307511, 122)
```

```
print("There was", len(df.shape[0]), "previous applications")
print("Only just", 1.0*df.shape[0]/307511, "previous applications")
print("Meaning that", 1.0-1.0*df.shape[0]/307511, "had no previous applications")
```

```
There was 307511 previous applications
Only just 1.0 previous applications
Meaning that 0.0 had no previous applications
```

```
ipw.sep(df[target].sort_values(), 'prev', 10, rot=90, figsize=(20,5))
```

```
<matplotlib.axes._subplots.AxesSubplot>
```

Histogram of Number of previous applications for an ID

```
ipw.sep(df[target].sort_values(), 'prev', 10, rot=90, figsize=(20,5))
```

```
<matplotlib.axes._subplots.AxesSubplot>
```

Can we differentiate applications by low, medium and high previous apps?

```
> 0 apps, all of people, 100% are new
> 1 apps, 0.7% of people, 99.3% had prev
> 2 apps, 0.3% of people, 99.7% had prev
```

```
df[df.SK_ID_PREV == df[df.SK_ID_PREV.isnull()].SK_ID_PREV].shape
df[df.SK_ID_PREV == df[df.SK_ID_PREV.isnull()].SK_ID_PREV].sample(1).T.sort_values(0, ascending=False)[0:20]
df[df.SK_ID_PREV == df[df.SK_ID_PREV.isnull()].SK_ID_PREV].sample(1).T.sort_values(0, ascending=False).shape
df[df.SK_ID_PREV == df[df.SK_ID_PREV.isnull()].SK_ID_PREV].sample(1).T.sort_values(0, ascending=False).tail(20)
df[df.SK_ID_PREV == df[df.SK_ID_PREV.isnull()].SK_ID_PREV].sample(1).T.sort_values(0, ascending=False).tail(20).shape
```

```
(307511, 122)
SK_ID_PREV                  NaN
SK_ID_CURR                  NaN
NAME_CONTRACT_TYPE          NaN
AMT_ANNUITY                 NaN
AMT_APPLICATION             NaN
AMT_CREDIT                  NaN
AMT_DOWN_PAYMENT            NaN
AMT_GOODS_PRICE             NaN
WEEKDAY_APPR_PROCESS_START  NaN
HOUR_APPR_PROCESS_START     NaN
FLAG_LAST_APPLICATION_PER   NaN
NFLAG_LAST_APPLICATION_IN   NaN
NAME_CASH_LOAN_PURPOSE      NaN
NAME_CONTRACT_STATUS        NaN
DAYS_DECISION               NaN
NAME_PAYMENT_TYPE           NaN
CODE_REJECT_REASON          NaN
NAME_TYPE_SUITE             NaN
NAME_CLIENT_TYPE            NaN
NAME_GOODS_CATEGORY         NaN
Name: 0, dtype: object
(122, 1)
SK_ID_PREV                  NaN
SK_ID_CURR                  NaN
NAME_CONTRACT_TYPE          NaN
AMT_ANNUITY                 NaN
AMT_APPLICATION             NaN
AMT_CREDIT                  NaN
AMT_DOWN_PAYMENT            NaN
AMT_GOODS_PRICE             NaN
WEEKDAY_APPR_PROCESS_START  NaN
HOUR_APPR_PROCESS_START     NaN
FLAG_LAST_APPLICATION_PER   NaN
NFLAG_LAST_APPLICATION_IN   NaN
NAME_CASH_LOAN_PURPOSE      NaN
NAME_CONTRACT_STATUS        NaN
DAYS_DECISION               NaN
NAME_PAYMENT_TYPE           NaN
CODE_REJECT_REASON          NaN
NAME_TYPE_SUITE             NaN
NAME_CLIENT_TYPE            NaN
NAME_GOODS_CATEGORY         NaN
Name: 0, dtype: object
(122, 1)
```

Feature Engineering and Selection

```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import numpy as np, pandas as pd, sklearn, matplotlib, scipy, gc, ipw, lightgbm as lgb
from sklearn.model_selection import train_test_split, KFold, StratifiredKFold
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, MaxAbsScaler
from sklearn.metrics import roc_auc_score, roc_curve
from scipy.stats import ttest_ind, chi2_contingency
import matplotlib.pyplot as plt, seaborn as sns, os, sys, gzip, zipfile, pickle, time, shutil, StringIO
from PIL import Image, ImageFont, ImageDraw
from IPLib import DB, Util, Display, KaggleWeb, KaggleArticle, ML
lr = ML.LogisticRegression(n_jobs=-1)
rf = ML.RandomForestClassifier(n_jobs=-1)
gb = ML.GradientBoostingClassifier(n_estimators=100)
np.set_printoptions(threshold=sys.maxsize)
pd.pd.set_option('display.max_rows', None)
pd.pd.set_option('display.max_columns', None)
pd.pd.set_option('display.width', None)
pd.pd.set_option('display.max_colwidth', -1)
```

```
The autoreload extension already loaded. To reload it, use:
  %reload_ext autoreload
```

```
df_app = ipw.read_csv('data/application_train.csv')
df_app.shape
```

```
(307511, 122)
```

```
df_app.sample(1).T
```

```
Feature Engineering on Bureau Balance
```

```
dataset='bureau_balance'
```

```
[nthreads, 4]
```

```
df_bureau = dataset='bureau_balance'
```

```
df_bureau.shape
```

```
(27299925, 3)
```

```
df_bureau.columns
```

```
Index(['SK_ID_BUREAU', 'MONTHS_BALANCE', 'STATUS'], dtype='object')
```

```
df_bureau.sample(1).T
df_bureau.dtypes
df_bureau.isnull().sum
df_bureau.describe(include=[object])
df_bureau.describe(exclude=[object])
```

```
SK_ID_BUREAU    int64
MONTHS_BALANCE  int64
STATUS          object
dtype: object
SK_ID_BUREAU    0
MONTHS_BALANCE  0
STATUS          0
dtype: int64
       SK_ID_BUREAU  MONTHS_BALANCE
count  27299925      27299925
mean   6036297       -30.670616
std    492351        23.811330
min    5001709       -96.000000
25%    5537996       -46.000000
50%    5996556       -25.000000
75%    6425941       -7.000000
max    6842888       0.000000
       STATUS
count  27299925
unique 6
top    C
freq   10731808
```

```
df_bureau[df.SK_ID_BUREAU == df_bureau[df_bureau.isnull().sum(axis=1)>0].SK_ID_BUREAU].sample(1).T.sort_values(0, ascending=False)[0:20]
df_bureau[df.SK_ID_BUREAU == df_bureau[df_bureau.isnull().sum(axis=1)>0].SK_ID_BUREAU].sample(1).T.sort_values(0, ascending=False).shape
df_bureau[df.SK_ID_BUREAU == df_bureau[df_bureau.isnull().sum(axis=1)>0].SK_ID_BUREAU].sample(1).T.sort_values(0, ascending=False).tail(20)
df_bureau[df.SK_ID_BUREAU == df_bureau[df_bureau.isnull().sum(axis=1)>0].SK_ID_BUREAU].sample(1).T.sort_values(0, ascending=False).tail(20).shape
```

```
Series([], dtype: object)
(0, 1)
Series([], dtype: object)
(0, 1)
```

```
[0 rows x 0 columns]
```

```
[0 rows x 0 columns]
```

```
[0 rows x 0 columns]
```

```
[INFO] Loaded: (1,048,575, 7)
[INFO] Dropped: ['id', 'Unnamed: 0']
[INFO] Shape after drop: (1,048,575, 5)
[INFO] Target distribution:
Target
0    0.999312
1    0.000688
Name: proportion, dtype: float64
```

```
[INFO] Numeric features (5): ['Amount', 'Time', 'V1', 'V2', 'V3']
[INFO] Missing values:
Series([], dtype: int64)
[INFO] Duplicates: 1,081
[INFO] Outlier % (IQR method):
Amount    14.0
Time       0.0
V1         5.1
V2         4.6
V3         2.1
dtype: float64
```

```
[INFO] Saved: eda_distributions.png
[INFO] Saved: eda_correlation.png
[INFO] Saved: eda_boilerplate.pkl
```

```
[INFO] Loaded: (1,048,575, 5)
[INFO] Dropped: ['Time']
[INFO] Shape after drop: (1,048,575, 4)
[INFO] Target distribution:
Target
0    0.999312
1    0.000688
Name: proportion, dtype: float64
[INFO] Numeric features (4): ['Amount', 'V1', 'V2', 'V3']
[INFO] Missing values:
Series([], dtype: int64)
[INFO] Duplicates: 1,081
[INFO] Outlier % (IQR method):
Amount    14.0
V1         5.1
V2         4.6
V3         2.1
dtype: float64
[INFO] Saved: eda_distributions.png
[INFO] Saved: eda_correlation.png
[INFO] Saved: eda_boilerplate.pkl
```

```
[INFO] Loaded: (1,048,575, 4
```

Log function

$$LogLoss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} x_{ij} * \ log(p_{ij})$$

Gini log loss

$$Gini(t) = 1 - \sum_{i=1}^{j} P(i|t)^2$$

Experimental results

```
Training
```

	avg_gini	Train Acc	Test Acc	Train AUC	Test AUC	Train F1	Test F1
0	Logistic Regression	0.5342	0.8121	0.8000	0.8000	0.3871	0.4211
1	Random Forest Model	0.5130	0.8175	0.8175	0.8075	0.3900	0.3846
2	XG Boost Model	0.5431	0.8250	0.8375	0.7750	0.4000	0.3636

Discussion of Results

In the approach, the models were overfitting and gave an accuracy of 1. We improved this, and the models performed better compared to phase 1

Logistic Regression results

- Train Accuracy: 81.21%
- Test ROC_AUC: 0.80

Random Forest results

- Train accuracy: 81%
- Test ROC_AUC: 0.80

XGBoost results

- Train accuracy: 83.25%
- Test ROC_AUC: 0.75

In this final phase, we have also implemented the AUC model using Pycaret. And obtained the following scores

- Test accuracy: 0.8
- Test AUC: 0.8
- Test Gini: 0.6

Conclusion: XGBoost performed better among all models.Kaggle submission was made on XGboost and gain a score 0.7747

Conclusion

This project's main goal is to develop a machine-learning model that can predict whether or not a loan applicant will be able to repay the loan. Without any statistical analysis, loans disparaging applicants with no credit history or default history get approved. The XGB classifier outperformed the other models in this study, indicating that it is the most appropriate algorithm for the job. The XGB classifier's capacity to manage complex data links and to learn from faulty mistakes is the reason for its success. The importance of addressing the problem utilizing statistical analysis resulting from numerous aspects taken into account.

In the previous phase of the project the models were overfitting (the data and hence gave an accuracy score of 1.0. I corrected this by implementing cross-validation and hyperparameter tuning. The models performed better in this phase. The Logistic Regression model had a Receiver Operating Characteristic Area Under Curve (ROC AUC) of 0.80 on the test data, and its F1 score indicated that the model's recall and precision were well balanced, with a score of 0.42. The Random Forest model had a ROC AUC of 0.80 and an F1 score of 0.38, indicating that it was not as strong at balancing recall and precision as the Logistic Regression model. The XGBoost model had the highest ROC AUC of 0.75 and an F1 score of 0.36. We have also made a kaggle submission and obtained a score of 0.7747.

Kaggle Submission

Leaderboard

Submitted notebook

sakthivelsel Last submitted 2 days ago
Submitted by Sivaji Rajan Sakthivel 2 minutes ago

Score: 0.7747
Public score: 0.7499

View & rate leaderboard

View Notebook

Refresh

References

Some of the material in this notebook has been adapted from here

TODO: Predicting Loan Repayment with Automated Feature Engineering in Featuretools

Guide Notebook

- Kaggle Competition: Home Credit Default Risk
  - https://github.com/WinnerDeveloper/home-credit-default-risk/blob/master/01.Exploratory%20Data%20Analysis.ipynb
  - https://www.kaggle.com/code/jtrotman/home-credit-default-risk-feature-tools/notebook
- Source engineering paper: https://dataiku-dss-solutions.s3.amazonaws.com/FEATURETOOL_DSS_EN.pdf
- https://www.analyticsvidhya.com/blog/2017/09/automated-feature-engineering/