# Project Title: *Home Credit Default Risk Classification using Machine Learning Techniques*

## Team Information:

Group members:

1. Shoukath Ali (shshaik@iu.edu)
2. Bhargavi Vasudev Jahagirdar (bjahagir@iu.edu)
3. Palavi Dhanaji Patil (palpatil2iu.edu)
4. Saransh Kamlesh Singh (singsara@iu.edu)

Team Photo:


## Phase Leadership Plan

| Project Phase | Phase Description | Phase leader |
|---|---|---|
| Phase 0 | Team creation in Canvas and Pick Project | Bhargavi Vasudev Jahagirdar |
| Phase 1 | Project Proposal | Bhargavi Vasudev Jahagirdar |
| Phase 2 | EDA and baseline pipeline | Palavi Dhanaji Patil |
| Phase 3 | Final Project HCDR - feature engineering + hyperparameter tuning | Saransh Singh |
| Phase 4 | Final Submission: Final Project HCDR | Shoukath Ali |

## Credit Assignment Plan

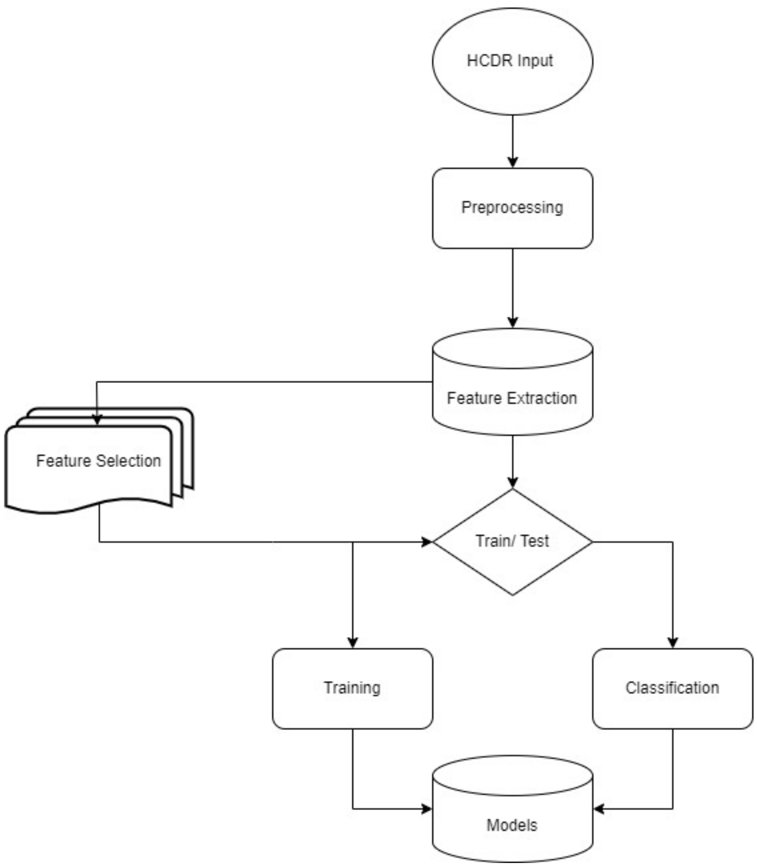
| Project Phase | Phase Description | Phase Task | Owner |
|---|---|---|---|
| Phase 0 | Team creation in Canvas and Pick Project | Team Creation | All |
| | | Team Introduction | All |
| | | Topic Selection | All |
| | | Skillset matrix discussion | All |
| | | Dataset walkthrough | All |
| | | Researching the models for the given dataset | All |
| | | Project meetings | All |
| Phase 1 | Project Proposal | Proposal Documentation | All |
| | | Gantt chart | Bhargavi Jahagirdar |
| | | Phase Leader Plan | Bhargavi Jahagirdar |
| | | Block diagram | Bhargavi Jahagirdar |
| | | Support Vector Machine Research | Shoukath Ali |
| | | Random Forest Research | Saransh Singh, Palavi Patil |
| | | Logistic Regression Research | Palavi Patil, Saransh Singh |
| | | Neural Networks Research | Shoukath Ali, Bhargavi Jahagirdar |
| Phase 2 | EDA and baseline pipeline | EDA - application_train.csv, bureau.csv, bureau_balance.csv, pos_cash_balance.csv, credit_cash.csv, installment_payment.csv, previous_application.csv | Bhargavi Jahagirdar, Palavi Patil, Saransh Singh |
| | | Feature Selection | All |
| | | Preprocessing | All |
| | | Pipelines | Shoukath Ali |
| | | Presentation | Saransh Singh |
| | | Project report | Bhargavi Jahagirdar, Palavi Patil |
| | | Kaggle Submission | Palavi Patil |
| Phase 3 | Feature engineering + Hyperparameter tuning | Feature Engineering | Saransh Singh |
| | | Hyperparameter tuning | Saransh Singh |
| | | Additional Feature Engineering | Palavi Patil, Bhargavi Jahagirdar |
| | | Model training: Random Forest, SVM, Logistic Regression | Saransh Singh, Palavi Patil, Bhargavi Jahagirdar |
| | | Deploying the final model: Random Forest, SVM, Logistic Regression | Saransh Singh, Palavi Patil, Bhargavi Jahagirdar |
| | | Initial Neural Networks Implementation | Shoukath Ali |
| | | Project report | Palavi Patil, Bhargavi Jahagirdar |
| | | Presentation | Saransh Singh |
| Phase 4 | Final Project HCDR / CaDoD - PyTorch Deep Learning | Improving the exisiting models | All |
| | | Implementation of Multi Layer Perceptron model (MLP) | All |
| | | Kaggle Submission | Shoukath Ali |
| | | Project Report | All |
| | | Presentation | All |

## Abstract

The given problem is the Home Credit Default Risk Prediction presented by Kaggle. The team aims to provide suitable Machine Learning models that can accurately predict the results for the given problem statement. For this binary classification problem, we used a Random Forest Classifier,Logistic Regression, and Neural Networks. We performed the project in 4 phases. The previous phases involved creating the project proposal, performing EDA, creating baseline pipelines, feature engineering, and hyperparameter tuning. The current phase involves the implementation of Neural Networks and using Neural Network architectures alongside the implementation of Random Forest Classifier, and Logistic Regression. Accuracy for each of them is91.95% and 91.7%, respectively. We obtained the best results through the Random Forest model. We submitted the below models to the Kaggle competition and achieved a public score of 73.409% using Neural Networks. We used two architectures for Neural Networks and achieved the accuracy of 91 % and 91 % for each of them

## Data Description

| Data File Name | Description |
|---|---|
| application_{train\|test}.csv | Main table for training, includes static data for loan applications. Each row represents a loan in the sample. |
| bureau.csv | Contains information about clients' previous credits reported to Credit Bureau for those with a loan in the sample. Multiple rows for each loan corresponding to the client's previous credits. |
| bureau_balance.csv | Monthly balances of previous credits in Credit Bureau, with one row per month for each previous credit, resulting in many rows. |
| POS_CASH_balance.csv | Monthly balance snapshots of previous POS and cash loans with Home Credit, featuring one row per month for each previous credit related to loans in the sample. |
| credit_card_balance.csv | Monthly balance snapshots of previous credit cards with Home Credit, with one row per month for each previous credit card related to loans in the sample. |
| previous_application.csv | Contains information about all previous loan applications for Home Credit clients in the sample, with one row for each previous application. |
| installments_payments.csv | Provides repayment history for previously disbursed credits from Home Credit related to the loans in the sample. Each row corresponds to a payment made or a missed payment. |
| HomeCredit_columns_description.csv | Contains column descriptions for the various data files. |




Machine Learning Models and Pipelines for the Dataset

4.1 SVM Metrics used 1) Classification metrics (F1 score, recall, precision) 2) accuracy_score

The problem statement's output is to predict the binary outcomes (classification problem). SVM is capable of improving the accuracy of the ML model by considering the proper loss function(Margin) or best parameters. Required Pipelines Standardize, Normalize, Impute (in case of missing data), stratified K fold ( for skewed data ), feature engineering, Grid Search CV (hyperparameter tuning), SVM, and Model evaluation. We will build the final pipelines using multiple sub-pipelines. For Example, numerical_pipeline = Pipeline([ ('imputer', SimpleImputer(strategy='mean')), ('scaler', StandardScaler()), ('pca', PCA(n_components=0.95)) ])

categorical_pipeline = Pipeline([ ('imputer', SimpleImputer(strategy='most_frequent')), ('one_hot', OneHotEncoder()) ])

preprocessor = ColumnTransformer([ ('num', numerical_pipeline, numerical_cols), ('cat', categorical_pipeline, categorical_cols)])

4.2 Random Forest

For the Home Credit Default Risk dataset, where the goal is typically to predict the loan repayment abilities of borrowers, a Random Forest model can be an excellent choice due to its versatility, robustness to overfitting, and ability to handle imbalanced datasets. Here's a proposal outline that answers your questions within a tight character limit:

Metrics used 1) AUC-ROC: Ideal for classification on imbalanced datasets, measures the ability to distinguish between classes.

2) F1-Score: Balances precision and recall, useful when false negatives and false positives are crucial.

3) Accuracy: Provides a quick understanding of overall performance, though less informative on its own for imbalanced data. Why is your model best for the problem statement Random Forest is robust to overfitting and excellent for handling complex datasets with many features, like Home Credit. It's also good for imbalanced classes and does not require scaling of data.

Required Pipelines 1) Data Preprocessing Pipeline: To handle missing values, encode categorical variables, and potentially scale features if necessary.

2) Model Training Pipeline: For fitting the Random Forest model with cross-validation to avoid overfitting.

3) Evaluation Pipeline: To apply the chosen metrics and validate the model's performance. Other pipelines:

4) Feature Engineering Pipeline: To create new features that can provide additional insights for the Random Forest algorithm.

5) Model Optimization Pipeline: Using grid search or random search to fine-tune hyperparameters for the Random Forest. For Example, feature_engineering_pipeline = Pipeline([ ('polynomial_features', PolynomialFeatures(degree=2, include_bias=False)), ('feature_selection', SelectFromModel(RandomForestClassifier(n_estimators=100))) ])

4.3 Logistic Regression Metrics used 1) Confusion Matrix: The confusion matrix will provide a detail of the model's predictions, which will include true positives, true negatives, false positives, and false negatives. 2) Accuracy Score: Accuracy will measure the proportion of correct predictions made by the logistic regression model. 3) Precision Score: Precision will measure the accuracy of positive predictions made by the model. It will tell us the proportion of positive predictions that were correct. 4) F1 score: The F1 score will use precision and recall. It balances the trade-off between precision and recall. Why is your model best for the problem statement For the HCDR dataset, we have to classify whether the client can repay the loan money. The classes will be Yes or No. For this, we can use logistic regression as it is specifically designed for binary classification problems. Logistic regression is a relatively simple model, which is computationally efficient and can be trained quickly. Required Pipelines 1) Standardization: This preprocessing step is required to do the scaling of the input features. 2) Imputation: Multiple methods can be used to impute missing data in our dataset. We can replace missing values with the mean. 3) GridSearchCV: This will be used to tune the hyperparameters so we can find the best combination that will optimize our performance. 4) Model pipeline: Here we can combine our model with other pipelines. For Example, preprocessingPipeline = Pipeline([ ('imputer', SimpleImputer(strategy='mean')), ('scaler', StandardScaler()), ]) Above metrics and pipelines can be used to train HCDR dataset on Logistic Regression Model. 4.4 Neural Networks Metrics used 1) Confusion matrix 2) Classification metrics (F1 score, recall, precision) 3) accuracy_score Why is your model best for the problem statement 1) Scalability: Neural networks can be scaled up with more layers and neurons to handle increasingly complex classification tasks. Deep neural networks (deep learning) are particularly effective at capturing hierarchical representations in the data, which can improve classification accuracy. 2) Regularization Techniques: Neural networks offer various regularization techniques to prevent overfitting, including dropout, weight decay, and early stopping, which can help improve generalization on classification tasks. 3) Neural networks can automatically learn relevant features from raw data. Instead of hand-crafting features, as is often done in traditional machine learning, neural networks learn representations from the data during training.

Required Pipelines 1) Standardize 2) Normalize, Impute (in case of missing data) 3) stratified K fold ( for skewed data ), data augmentation/pre-processing 4) Neural Network Model 5) Grid Search CV (hyperparameter tuning), Optimizing/Regularisation (activation function, loss function, epoch, and batch values) 6) Model evaluation.

Other pipelines: we build the final pipeline using all the sub-pipelines, and here, feature engineering is not required as Neural networks are capable of doing feature learning from the data itself.

F1 Score: The F1 score is a metric used in binary classification to measure a balance between precision and recall. It's calculated using the following formula:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

AUC-ROC (Area Under the Receiver Operating Characteristic Curve): The AUC-ROC is a performance measurement for classification problems at various threshold settings. The ROC curve is a plot of the true positive rate (TPR) against the false positive rate (FPR) at various threshold values. The AUC-ROC represents the area under this ROC curve.

$$\mathrm{TPR} = \frac{\text{True Positives}}{\text{True Positives}+\text{False Negatives}}$$

$$\mathrm{FPR} = \frac{\text{False Positives}}{\text{False Positives}+\text{True Negatives}}$$

## Project Tasks

| Project Phase | Phase Description | Project Tasks | Week 11 | Week 12 | Week 13 | Week 14 | Week 15 | Week 16 |
|---|---|---|---|---|---|---|---|---|
| Phase 0 | Team creation in Canvas and Pick Project | Team Creation | | | | | | |
| | | Project Topic Selection | | | | | | |
| Phase 1 | Project Proposal | Proposal Documentation : Project description, data, metrics, baseline models, baseline pipeline, and other planned pipelines | | | | | | |
| | | Project Plan creation | | | | | | |
| | | Phase leader plan creation | | | | | | |
| | | Credit Assignment Plan | | | | | | |
| Phase 2 | EDA and baseline pipeline | Exploratory Data Analysis | | | | | | |
| | | Baseline model creation | | | | | | |
| | | Feature Engineering | | | | | | |
| | | Hyperparameter tuning | | | | | | |
| | | Phase documentation | | | | | | |
| | | Presentation video preparation | | | | | | |
| Phase 3 | Final Project HCDR - feature engineering + hyperparameter tuning | Feature Engineering | | | | | | |
| | | Hyperparameter tuning | | | | | | |
| | | Additional Feature Engineering | | | | | | |
| | | Model training | | | | | | |
| | | Deploying the final model | | | | | | |
| Phase 4 | Final Submission: Final Project HCDR | Neural Networks Implementation | | | | | | |
| | | Model training | | | | | | |
| | | Deploying the final model | | | | | | |
| | | Final project report | | | | | | |
| | | Presentation video preparation | | | | | | |

## Importing Datasets

```
! pip install -q kaggle
```

```
from google.colab import files

files.upload()
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 72 bytes, last modified: 11/14/2023 - 100% done

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"saranshsingh2626","key":"5c9f084b0d4b8f05324de44a8f9c3b97"}'}
```

```
! mkdir ~/.kaggle

! cp kaggle.json ~/.kaggle/
```

```
! chmod 600 ~/.kaggle/kaggle.json
```

```
! kaggle datasets list
```

```
ref                                                          title                                                 size  lastUpda
-----------------------------------------------------------  ---------------------------------------------------  -----  --------
thedrcat/daigt-v2-train-dataset                              DAIGT V2 Train Dataset                                29MB  2023-11-
muhammadbinimran/housing-price-prediction-data               Housing Price Prediction Data                        763KB  2023-11-
carlmcbrideellis/llm-7-prompt-training-dataset               LLM: 7 prompt training dataset                        41MB  2023-11-
thedrcat/daigt-proper-train-dataset                          DAIGT Proper Train Dataset                           119MB  2023-11-
joebeachcapital/30000-spotify-songs                          30000 Spotify Songs                                    3MB  2023-11-
jacksondivakarr/laptop-price-prediction-dataset              Laptop Price Prediction Dataset                      119KB  2023-11-
ddosad/auto-sales-data                                       Automobile Sales data                                 79KB  2023-11-
julnazz/diabetes-health-indicators-dataset                   Diabetes Health Indicators Dataset                     5MB  2023-11-
stealthtechnologies/predict-lifespan-of-a-comet-goldfish     Predict lifespan of a comet goldfish                  25KB  2023-11-
nelgiriyewithana/world-educational-data                      World Educational Data                                 9KB  2023-11-
thedevastator/bank-term-deposit-predictions                  Bank Term Deposit Predictions                        541KB  2023-11-
maso0dahmed/video-games-data                                 Video Games Data                                       5MB  2023-11-
alejopaullier/daigt-external-dataset                         DAIGT | External Dataset                               3MB  2023-10-
```

```
nelgiriyewithana/australian-vehicle-prices            Australian Vehicle Prices                           582KB  2023-11-
prasad22/healthcare-dataset                           🩺Healthcare Dataset 🧪                              483KB  2023-
adampq/linkedin-jobs-machine-learning-data-set        LinkedIn Job Postings - Machine Learning Data Set   38MB  2023-11-
jacksondivakarr/online-shopping-dataset               🛒 Online Shopping Dataset 📊📉📈                       5MB  202
asimislam/30-yrs-stock-market-data                    30 yrs Stock Market Data                            882KB  2023-11-
imtkaggleteam/life-expectancy                         Life Expectancy                                     730KB  2023-11-
sujaykapadnis/products-datasets                       Detailed Products Datasets                          100KB  2023-11-
```

```
#Downloading the dataset from Kaggle
! kaggle competitions download -c home-credit-default-risk
```

```
Downloading home-credit-default-risk.zip to /content
 98% 673M/688M [00:08<00:00, 137MB/s]
100% 688M/688M [00:08<00:00, 88.4MB/s]
```

```
#Creating a directory for the adding the dataset
! mkdir dataset
```

```
# unzip the downloaded dataset file in the dataset directory
! unzip home-credit-default-risk.zip -d dataset
```

```
Archive:  home-credit-default-risk.zip
  inflating: dataset/HomeCredit_columns_description.csv
  inflating: dataset/POS_CASH_balance.csv
  inflating: dataset/application_test.csv
  inflating: dataset/application_train.csv
  inflating: dataset/bureau.csv
  inflating: dataset/bureau_balance.csv
  inflating: dataset/credit_card_balance.csv
  inflating: dataset/installments_payments.csv
  inflating: dataset/previous_application.csv
  inflating: dataset/sample_submission.csv
```

```
#Loading the data files in dataframes
import numpy as np
import pandas as pd

df_application_train = pd.read_csv('dataset/application_train.csv')
df_application_test = pd.read_csv('dataset/application_test.csv')
df_bureau = pd.read_csv('dataset/bureau.csv')
df_bureau_balance = pd.read_csv('dataset/bureau_balance.csv')
df_pos_cash_balance = pd.read_csv('dataset/POS_CASH_balance.csv')
df_credit_card_balance = pd.read_csv('dataset/credit_card_balance.csv')
df_previous_application = pd.read_csv('dataset/previous_application.csv')
df_installments_payments = pd.read_csv('dataset/installments_payments.csv')
```

# Data Description

## Data File: application_train.csv

**File description:**

- This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET).
- Static data for all applications. One row represents one loan in our data sample.

```
print("File name: application_train.csv")
print("Number of rows, columns:", df_application_train.shape)
print("Number of Missing Values: " + str(df_application_train.isna().sum().sum()))
df_application_train.head(5)
```

```
File name: application_train.csv
Number of rows, columns: (307511, 122)
Number of Missing Values: 9152465
```

| | SK_ID_CURR | TARGET | NAME_CONTRACT_TYPE | CODE_GENDER | FLAG_OWN_CAR | FLAG_OWN_REALTY | CNT_CHILDREN | AMT_INCOME_TOTAL | AMT_CRE |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 100002 | 1 | Cash loans | M | N | Y | 0 | 202500.0 | 4065 |
| 1 | 100003 | 0 | Cash loans | F | N | N | 0 | 270000.0 | 12935 |
| 2 | 100004 | 0 | Revolving loans | M | Y | Y | 0 | 67500.0 | 1350 |

## Data File: application_test.csv

**File description:**

- This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET).
- Static data for all applications. One row represents one loan in our data sample.

```
print("File name: application_test.csv")
print("Number of rows, columns:", df_application_test.shape)
print("Number of Missing Values: " + str(df_application_test.isna().sum().sum()))
df_application_test.head(5)
```

```
File name: application_test.csv
Number of rows, columns: (48744, 121)
Number of Missing Values: 1404419
```

| | SK_ID_CURR | NAME_CONTRACT_TYPE | CODE_GENDER | FLAG_OWN_CAR | FLAG_OWN_REALTY | CNT_CHILDREN | AMT_INCOME_TOTAL | AMT_CREDIT | AMT |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 100001 | Cash loans | F | N | Y | 0 | 135000.0 | 568800.0 | |
| 1 | 100005 | Cash loans | M | N | Y | 0 | 99000.0 | 222768.0 | |
| 2 | 100013 | Cash loans | M | Y | Y | 0 | 202500.0 | 663264.0 | |
| 3 | 100028 | Cash loans | F | N | Y | 2 | 315000.0 | 1575000.0 | |
| 4 | 100038 | Cash loans | M | Y | N | 1 | 180000.0 | 625500.0 | |

5 rows × 121 columns

## Data File: bureau.csv

**File description:**

- All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample).
- For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.

```
print("File name: bureau.csv")
print("Number of rows, columns:", df_bureau.shape)
print("Number of Missing Values: " + str(df_bureau.isna().sum().sum()))
df_bureau.head(5)
```

```
File name: bureau.csv
Number of rows, columns: (1716428, 17)
Number of Missing Values: 3939947
```

| | SK_ID_CURR | SK_ID_BUREAU | CREDIT_ACTIVE | CREDIT_CURRENCY | DAYS_CREDIT | CREDIT_DAY_OVERDUE | DAYS_CREDIT_ENDDATE | DAYS_ENDDAT |
|---|---|---|---|---|---|---|---|---|
| 0 | 215354 | 5714462 | Closed | currency 1 | -497 | 0 | -153.0 | |
| 1 | 215354 | 5714463 | Active | currency 1 | -208 | 0 | 1075.0 | |
| 2 | 215354 | 5714464 | Active | currency 1 | -203 | 0 | 528.0 | |
| 3 | 215354 | 5714465 | Active | currency 1 | -203 | 0 | NaN | |
| 4 | 215354 | 5714466 | Active | currency 1 | -629 | 0 | 1197.0 | |

## Data File: bureau_balance.csv

**File description:**

- Monthly balances of previous credits in Credit Bureau.
- This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (#loans in sample * # of relative previous credits * # of months where we have some history observable for the previous credits) rows.

```
print("File name: bureau_balance.csv")
print("Number of rows, columns:", df_bureau_balance.shape)
print("Number of Missing Values: " + str(df_bureau_balance.isna().sum().sum()))
df_bureau_balance.head(5)
```

```
File name: bureau_balance.csv
Number of rows, columns: (27299925, 3)
Number of Missing Values: 0
```

| | SK_ID_BUREAU | MONTHS_BALANCE | STATUS |
|---|---|---|---|
| **0** | 5715448 | 0 | C |
| **1** | 5715448 | -1 | C |
| **2** | 5715448 | -2 | C |
| **3** | 5715448 | -3 | C |
| **4** | 5715448 | -4 | C |

## Data File: pos_cash_balance.csv

**File description:**

- Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
- This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credits * # of months in which we have some history observable for the previous credits) rows.

```
print("File name: pos_cash_balance.csv")
print("Number of rows, columns:", df_pos_cash_balance.shape)
print("Number of Missing Values: " + str(df_pos_cash_balance.isna().sum().sum()))
df_pos_cash_balance.head(5)
```

```
File name: pos_cash_balance.csv
Number of rows, columns: (10001358, 8)
Number of Missing Values: 52158
```

| | SK_ID_PREV | SK_ID_CURR | MONTHS_BALANCE | CNT_INSTALMENT | CNT_INSTALMENT_FUTURE | NAME_CONTRACT_STATUS | SK_DPD | SK_DPD_DEF |
|---|---|---|---|---|---|---|---|---|
| **0** | 1803195 | 182943 | -31 | 48.0 | 45.0 | Active | 0 | 0 |
| **1** | 1715348 | 367990 | -33 | 36.0 | 35.0 | Active | 0 | 0 |
| **2** | 1784872 | 397406 | -32 | 12.0 | 9.0 | Active | 0 | 0 |
| **3** | 1903291 | 269225 | -35 | 48.0 | 42.0 | Active | 0 | 0 |
| **4** | 2341044 | 334279 | -35 | 36.0 | 35.0 | Active | 0 | 0 |

## Data File: credit_card_balance.csv

**File description:**

- Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
- This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credit cards * # of months where we have some history observable for the previous credit card) rows.

```
print("File name: credit_card_balance.csv")
print("Number of rows, columns:", df_credit_card_balance.shape)
print("Number of Missing Values: " + str(df_credit_card_balance.isna().sum().sum()))
df_credit_card_balance.head(5)
```

```
File name: credit_card_balance.csv
Number of rows, columns: (3840312, 23)
Number of Missing Values: 5877356
```

| | SK_ID_PREV | SK_ID_CURR | MONTHS_BALANCE | AMT_BALANCE | AMT_CREDIT_LIMIT_ACTUAL | AMT_DRAWINGS_ATM_CURRENT | AMT_DRAWINGS_CURRE |
|---|---|---|---|---|---|---|---|
| 0 | 2562384 | 378907 | -6 | 56.970 | 135000 | 0.0 | 877 |
| 1 | 2582071 | 363914 | -1 | 63975.555 | 45000 | 2250.0 | 2250 |
| 2 | 1740877 | 371185 | -7 | 31815.225 | 450000 | 0.0 | ( |
| 3 | 1389973 | 337855 | -4 | 236572.110 | 225000 | 2250.0 | 2250 |
| 4 | 1891521 | 126868 | -1 | 453919.455 | 450000 | 0.0 | 11547 |

5 rows × 23 columns

## Data File: previous_application.csv

**File description:**

- All previous applications for Home Credit loans of clients who have loans in our sample.
- There is one row for each previous application related to loans in our data sample.

```
print("File name: df_previous_application.csv")
print("Number of rows, columns:", df_previous_application.shape)
print("Number of Missing Values: " + str(df_previous_application.isna().sum().sum()))
df_previous_application.head(5)
```

```
File name: df_previous_application.csv
Number of rows, columns: (1670214, 37)
Number of Missing Values: 11109336
```

| | SK_ID_PREV | SK_ID_CURR | NAME_CONTRACT_TYPE | AMT_ANNUITY | AMT_APPLICATION | AMT_CREDIT | AMT_DOWN_PAYMENT | AMT_GOODS_PRICE | WE |
|---|---|---|---|---|---|---|---|---|---|
| 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

## Data File: installments_payments.csv

**File description:**

- Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample.
- There is a) one row for every payment that was made plus b) one row each for missed payment.
- One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.

```
print("File name: installments_payments.csv")
print("Number of rows, columns:", df_installments_payments.shape)
print("Number of Missing Values: " + str(df_installments_payments.isna().sum().sum()))
df_installments_payments.head(5)
```

```
File name: installments_payments.csv
```

## Methods

**Discussion:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **1** | 1330831 | 151639 | 0.0 | 34 | -2156.0 | -2156.0 | 1716.525 |

## Exploratory Data Analysis

### Data File: application_test.csv

**File description:**

- This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET).
- Static data for all applications. One row represents one loan in our data sample.

```
print("File name: application_test.csv")
print("Number of rows, columns:", df_application_test.shape)
print("Number of Missing Values: " + str(df_application_test.isna().sum().sum()))
df_application_test.head(5)
```

```
File name: application_test.csv
Number of rows, columns: (48744, 121)
Number of Missing Values: 1404419
```

| | SK_ID_CURR | NAME_CONTRACT_TYPE | CODE_GENDER | FLAG_OWN_CAR | FLAG_OWN_REALTY | CNT_CHILDREN | AMT_INCOME_TOTAL | AMT_CREDIT | AMT |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 100001 | Cash loans | F | N | Y | 0 | 135000.0 | 568800.0 | |
| **1** | 100005 | Cash loans | M | N | Y | 0 | 99000.0 | 222768.0 | |
| **2** | 100013 | Cash loans | M | Y | Y | 0 | 202500.0 | 663264.0 | |
| **3** | 100028 | Cash loans | F | N | Y | 2 | 315000.0 | 1575000.0 | |
| **4** | 100038 | Cash loans | M | Y | N | 1 | 180000.0 | 625500.0 | |

5 rows × 121 columns

### Data File: bureau.csv

**File description:**

- All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample).
- For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.

```
print("File name: bureau.csv")
print("Number of rows, columns:", df_bureau.shape)
print("Number of Missing Values: " + str(df_bureau.isna().sum().sum()))
df_bureau.head(5)
```

```
File name: bureau.csv
Number of rows, columns: (1716428, 17)
Number of Missing Values: 3939947
```

| | SK_ID_CURR | SK_ID_BUREAU | CREDIT_ACTIVE | CREDIT_CURRENCY | DAYS_CREDIT | CREDIT_DAY_OVERDUE | DAYS_CREDIT_ENDDATE | DAYS_ENDDAT |
|---|---|---|---|---|---|---|---|---|
| **0** | 215354 | 5714462 | Closed | currency 1 | -497 | 0 | -153.0 | |
| **1** | 215354 | 5714463 | Active | currency 1 | -208 | 0 | 1075.0 | |
| **2** | 215354 | 5714464 | Active | currency 1 | -203 | 0 | 528.0 | |
| **3** | 215354 | 5714465 | Active | currency 1 | -203 | 0 | NaN | |
| **4** | 215354 | 5714466 | Active | currency 1 | -629 | 0 | 1197.0 | |

## Data File: bureau_balance.csv

**File description:**

- Monthly balances of previous credits in Credit Bureau.
- This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (#loans in sample * # of relative previous credits * # of months where we have some history observable for the previous credits) rows.

```
print("File name: bureau_balance.csv")
print("Number of rows, columns:", df_bureau_balance.shape)
print("Number of Missing Values: " + str(df_bureau_balance.isna().sum().sum()))
df_bureau_balance.head(5)
```

```
File name: bureau_balance.csv
Number of rows, columns: (27299925, 3)
Number of Missing Values: 0
```

| | SK_ID_BUREAU | MONTHS_BALANCE | STATUS |
|---|---|---|---|
| **0** | 5715448 | 0 | C |
| **1** | 5715448 | -1 | C |
| **2** | 5715448 | -2 | C |
| **3** | 5715448 | -3 | C |
| **4** | 5715448 | -4 | C |

## Data File: pos_cash_balance.csv

**File description:**

- Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
- This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credits * # of months in which we have some history observable for the previous credits) rows.

```
print("File name: pos_cash_balance.csv")
print("Number of rows, columns:", df_pos_cash_balance.shape)
print("Number of Missing Values: " + str(df_pos_cash_balance.isna().sum().sum()))
df_pos_cash_balance.head(5)
```

```
File name: pos_cash_balance.csv
Number of rows, columns: (10001358, 8)
Number of Missing Values: 52158
```

| | SK_ID_PREV | SK_ID_CURR | MONTHS_BALANCE | CNT_INSTALMENT | CNT_INSTALMENT_FUTURE | NAME_CONTRACT_STATUS | SK_DPD | SK_DPD_DEF |
|---|---|---|---|---|---|---|---|---|
| **0** | 1803195 | 182943 | -31 | 48.0 | 45.0 | Active | 0 | 0 |
| **1** | 1715348 | 367990 | -33 | 36.0 | 35.0 | Active | 0 | 0 |
| **2** | 1784872 | 397406 | -32 | 12.0 | 9.0 | Active | 0 | 0 |
| **3** | 1903291 | 269225 | -35 | 48.0 | 42.0 | Active | 0 | 0 |
| **4** | 2341044 | 334279 | -35 | 36.0 | 35.0 | Active | 0 | 0 |

## Data File: credit_card_balance.csv

**File description:**

- Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
- This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credit cards * # of months where we have some history observable for the previous credit card) rows.

```
print("File name: credit_card_balance.csv")
print("Number of rows, columns:", df_credit_card_balance.shape)
print("Number of Missing Values: " + str(df_credit_card_balance.isna().sum().sum()))
df_credit_card_balance.head(5)
```

```
File name: credit_card_balance.csv
Number of rows, columns: (3840312, 23)
Number of Missing Values: 5877356
```

| | SK_ID_PREV | SK_ID_CURR | MONTHS_BALANCE | AMT_BALANCE | AMT_CREDIT_LIMIT_ACTUAL | AMT_DRAWINGS_ATM_CURRENT | AMT_DRAWINGS_CURRE |
|---|---|---|---|---|---|---|---|
| **0** | 2562384 | 378907 | -6 | 56.970 | 135000 | 0.0 | 877 |
| **1** | 2582071 | 363914 | -1 | 63975.555 | 45000 | 2250.0 | 2250 |
| **2** | 1740877 | 371185 | -7 | 31815.225 | 450000 | 0.0 | ( |
| **3** | 1389973 | 337855 | -4 | 236572.110 | 225000 | 2250.0 | 2250 |
| **4** | 1891521 | 126868 | -1 | 453919.455 | 450000 | 0.0 | 11547 |

5 rows × 23 columns

## Data File: previous_application.csv

**File description:**

- All previous applications for Home Credit loans of clients who have loans in our sample.
- There is one row for each previous application related to loans in our data sample.

```
print("File name: df_previous_application.csv")
print("Number of rows, columns:", df_previous_application.shape)
print("Number of Missing Values: " + str(df_previous_application.isna().sum().sum()))
df_previous_application.head(5)
```

```
File name: df_previous_application.csv
Number of rows, columns: (1670214, 37)
Number of Missing Values: 11109336
```

| | SK_ID_PREV | SK_ID_CURR | NAME_CONTRACT_TYPE | AMT_ANNUITY | AMT_APPLICATION | AMT_CREDIT | AMT_DOWN_PAYMENT | AMT_GOODS_PRICE | WE |
|---|---|---|---|---|---|---|---|---|---|
| **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

## Data File: installments_payments.csv

**File description:**

- Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample.
- There is a) one row for every payment that was made plus b) one row each for missed payment.
- One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.

```
print("File name: installments_payments.csv")
print("Number of rows, columns:", df_installments_payments.shape)
print("Number of Missing Values: " + str(df_installments_payments.isna().sum().sum()))
df_installments_payments.head(5)
```

File name: installments_payments.csv

## **File**: application_train.csv

```
print("Number of Rows: " + str(df_application_train.shape[0]))
print("Number of Columns: " + str(df_application_train.shape[1]))
print("Number of Total Missing Values: " + str(df_application_train.isna().sum().sum()))
print("Data Frame Shape: " + str(df_application_train.shape))
print("Number of Missing Values by Feature: " + str(df_application_train.isna().sum()))
print("Data Types:",df_application_train.dtypes)
print("Data Frame: Data Types", df_application_train.dtypes.value_counts())
print("Summary", df_application_train.describe())
print("Correlation Statistic", df_application_train.corr())
print("Summary Information:", df_application_train.info())
```

```
SK_ID_CURR                                   0.002099
TARGET                                       0.000788
CNT_CHILDREN                                -0.002436
AMT_INCOME_TOTAL                             0.002387
AMT_CREDIT                                  -0.001275
...                                               ...
AMT_REQ_CREDIT_BUREAU_DAY                    0.217412
AMT_REQ_CREDIT_BUREAU_WEEK                   1.000000
AMT_REQ_CREDIT_BUREAU_MON                   -0.014096
AMT_REQ_CREDIT_BUREAU_QRT                   -0.015115
AMT_REQ_CREDIT_BUREAU_YEAR                   0.018917

                            AMT_REQ_CREDIT_BUREAU_MON  \
SK_ID_CURR                                  0.000485
TARGET                                     -0.012462
CNT_CHILDREN                               -0.010808
AMT_INCOME_TOTAL                            0.024700
AMT_CREDIT                                  0.054451
...                                              ...
AMT_REQ_CREDIT_BUREAU_DAY                  -0.005258
AMT_REQ_CREDIT_BUREAU_WEEK                 -0.014096
AMT_REQ_CREDIT_BUREAU_MON                   1.000000
AMT_REQ_CREDIT_BUREAU_QRT                  -0.007789
AMT_REQ_CREDIT_BUREAU_YEAR                 -0.004975

                            AMT_REQ_CREDIT_BUREAU_QRT  \
SK_ID_CURR                                  0.001025
TARGET                                     -0.002022
CNT_CHILDREN                               -0.007836
AMT_INCOME_TOTAL                            0.004859
AMT_CREDIT                                  0.015925
...                                              ...
AMT_REQ_CREDIT_BUREAU_DAY                  -0.004416
AMT_REQ_CREDIT_BUREAU_WEEK                 -0.015115
AMT_REQ_CREDIT_BUREAU_MON                  -0.007789
AMT_REQ_CREDIT_BUREAU_QRT                   1.000000
AMT_REQ_CREDIT_BUREAU_YEAR                  0.076208

                            AMT_REQ_CREDIT_BUREAU_YEAR
SK_ID_CURR                                   0.004659
TARGET                                       0.019930
CNT_CHILDREN                                -0.041550
AMT_INCOME_TOTAL                             0.011690
AMT_CREDIT                                  -0.048448
...                                               ...
AMT_REQ_CREDIT_BUREAU_DAY                   -0.003355
AMT_REQ_CREDIT_BUREAU_WEEK                   0.018917
AMT_REQ_CREDIT_BUREAU_MON                   -0.004975
AMT_REQ_CREDIT_BUREAU_QRT                    0.076208
AMT_REQ_CREDIT_BUREAU_YEAR                   1.000000

[106 rows x 106 columns]
<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
Summary Information: None
```

## **File**: application_test.csv

```
print("Number of Rows: " + str(df_application_test.shape[0]))
print("Number of Columns: " + str(df_application_test.shape[1]))
print("Number of Total Missing Values: " + str(df_application_test.isna().sum().sum()))
print("Data Frame Shape: " + str(df_application_test.shape))
print("Number of Missing Values by Feature: " + str(df_application_test.isna().sum()))
print("Data Types:",df_application_test.dtypes)
print("Data Frame: Data Types", df_application_test.dtypes.value_counts())
print("Summary", df_application_test.describe())
print("Correlation Statistic", df_application_test.corr())
print("Summary Information:", df_application_test.info())
```

```
SK_ID_CURR                                    0.001178
CNT_CHILDREN                                  0.007523
AMT_INCOME_TOTAL                             -0.002867
AMT_CREDIT                                    0.002904
AMT_ANNUITY                                   0.003085
...                                                ...
AMT_REQ_CREDIT_BUREAU_DAY                     0.035567
AMT_REQ_CREDIT_BUREAU_WEEK                    1.000000
AMT_REQ_CREDIT_BUREAU_MON                     0.054291
AMT_REQ_CREDIT_BUREAU_QRT                     0.024957
AMT_REQ_CREDIT_BUREAU_YEAR                   -0.000252

                            AMT_REQ_CREDIT_BUREAU_MON  \
SK_ID_CURR                                   0.000430
CNT_CHILDREN                                -0.008337
AMT_INCOME_TOTAL                             0.008691
AMT_CREDIT                                  -0.000156
AMT_ANNUITY                                  0.005695
...                                               ...
AMT_REQ_CREDIT_BUREAU_DAY                    0.005877
AMT_REQ_CREDIT_BUREAU_WEEK                   0.054291
AMT_REQ_CREDIT_BUREAU_MON                    1.000000
AMT_REQ_CREDIT_BUREAU_QRT                    0.005446
AMT_REQ_CREDIT_BUREAU_YEAR                   0.026118

                            AMT_REQ_CREDIT_BUREAU_QRT  \
SK_ID_CURR                                  -0.002092
CNT_CHILDREN                                 0.029006
AMT_INCOME_TOTAL                             0.007410
AMT_CREDIT                                  -0.007750
AMT_ANNUITY                                  0.012443
...                                               ...
AMT_REQ_CREDIT_BUREAU_DAY                    0.006509
AMT_REQ_CREDIT_BUREAU_WEEK                   0.024957
AMT_REQ_CREDIT_BUREAU_MON                    0.005446
AMT_REQ_CREDIT_BUREAU_QRT                    1.000000
AMT_REQ_CREDIT_BUREAU_YEAR                  -0.013081

                            AMT_REQ_CREDIT_BUREAU_YEAR
SK_ID_CURR                                    0.003457
CNT_CHILDREN                                 -0.039265
AMT_INCOME_TOTAL                              0.003281
AMT_CREDIT                                   -0.034533
AMT_ANNUITY                                  -0.044901
...                                                ...
AMT_REQ_CREDIT_BUREAU_DAY                     0.002002
AMT_REQ_CREDIT_BUREAU_WEEK                   -0.000252
AMT_REQ_CREDIT_BUREAU_MON                     0.026118
AMT_REQ_CREDIT_BUREAU_QRT                    -0.013081
AMT_REQ_CREDIT_BUREAU_YEAR                    1.000000

[105 rows x 105 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
Summary Information: None
```

## **File**: bureau.csv

```
print("Number of Rows: " + str(df_bureau.shape[0]))
print("Number of Columns: " + str(df_bureau.shape[1]))
print("Number of Total Missing Values: " + str(df_bureau.isna().sum().sum()))
print("Data Frame Shape: " + str(df_bureau.shape))
print("Number of Missing Values by Feature: " + str(df_bureau.isna().sum()))
print("Data Types:",df_bureau.dtypes)
print("Data Frame: Data Types", df_bureau.dtypes.value_counts())
print("Summary", df_bureau.describe())
print("Correlation Statistic", df_bureau.corr())
print("Summary Information:", df_bureau.info())
```

```
Number of Rows: 1716428
Number of Columns: 17
Number of Total Missing Values: 3939947
Data Frame Shape: (1716428, 17)
Number of Missing Values by Feature: SK_ID_CURR                      0
SK_ID_BUREAU                    0
CREDIT_ACTIVE                   0
CREDIT_CURRENCY                 0
DAYS_CREDIT                     0
CREDIT_DAY_OVERDUE              0
DAYS_CREDIT_ENDDATE        105553
DAYS_ENDDATE_FACT          633653
AMT_CREDIT_MAX_OVERDUE    1124488
CNT_CREDIT_PROLONG              0
AMT_CREDIT_SUM                 13
AMT_CREDIT_SUM_DEBT        257669
AMT_CREDIT_SUM_LIMIT       591780
AMT_CREDIT_SUM_OVERDUE          0
CREDIT_TYPE                     0
DAYS_CREDIT_UPDATE              0
AMT_ANNUITY               1226791
dtype: int64
Data Types: SK_ID_CURR                  int64
SK_ID_BUREAU                int64
CREDIT_ACTIVE              object
CREDIT_CURRENCY            object
DAYS_CREDIT                 int64
CREDIT_DAY_OVERDUE          int64
DAYS_CREDIT_ENDDATE       float64
DAYS_ENDDATE_FACT         float64
AMT_CREDIT_MAX_OVERDUE    float64
CNT_CREDIT_PROLONG          int64
AMT_CREDIT_SUM            float64
AMT_CREDIT_SUM_DEBT       float64
AMT_CREDIT_SUM_LIMIT      float64
AMT_CREDIT_SUM_OVERDUE    float64
CREDIT_TYPE                object
DAYS_CREDIT_UPDATE          int64
AMT_ANNUITY               float64
dtype: object
Data Frame: Data Types float64    8
int64      6
object     3
dtype: int64
Summary          SK_ID_CURR  SK_ID_BUREAU   DAYS_CREDIT  CREDIT_DAY_OVERDUE  \
count  1.716428e+06  1.716428e+06  1.716428e+06        1.716428e+06
mean   2.782149e+05  5.924434e+06 -1.142108e+03        8.181666e-01
std    1.029386e+05  5.322657e+05  7.951649e+02        3.654443e+01
min    1.000010e+05  5.000000e+06 -2.922000e+03        0.000000e+00
25%    1.888668e+05  5.463954e+06 -1.666000e+03        0.000000e+00
50%    2.780550e+05  5.926304e+06 -9.870000e+02        0.000000e+00
75%    3.674260e+05  6.385681e+06 -4.740000e+02        0.000000e+00
max    4.562550e+05  6.843457e+06  0.000000e+00        2.792000e+03

       DAYS_CREDIT_ENDDATE  DAYS_ENDDATE_FACT  AMT_CREDIT_MAX_OVERDUE  \
count         1.610875e+06       1.082775e+06            5.919400e+05
mean          5.105174e+02      -1.017437e+03            3.825418e+03
std           4.994220e+03       7.140106e+02            2.060316e+05
```

**File**: bureau_balance.csv

```
print("Number of Rows: " + str(df_bureau_balance.shape[0]))
print("Number of Columns: " + str(df_bureau_balance.shape[1]))
print("Number of Total Missing Values: " + str(df_bureau_balance.isna().sum().sum()))
print("Data Frame Shape: " + str(df_bureau_balance.shape))
print("Number of Missing Values by Feature: " + str(df_bureau_balance.isna().sum()))
print("Data Types:",df_bureau_balance.dtypes)
print("Data Frame: Data Types", df_bureau_balance.dtypes.value_counts())
print("Summary", df_bureau_balance.describe())
print("Correlation Statistic", df_bureau_balance.corr())
print("Summary Information:", df_bureau_balance.info())
```

```
Number of Rows: 27299925
Number of Columns: 3
Number of Total Missing Values: 0
Data Frame Shape: (27299925, 3)
Number of Missing Values by Feature: SK_ID_BUREAU      0
MONTHS_BALANCE    0
STATUS            0
dtype: int64
Data Types: SK_ID_BUREAU       int64
MONTHS_BALANCE     int64
STATUS            object
dtype: object
Data Frame: Data Types int64     2
object    1
dtype: int64
Summary        SK_ID_BUREAU  MONTHS_BALANCE
count  2.729992e+07    2.729992e+07
mean   6.036297e+06   -3.074169e+01
std    4.923489e+05    2.386451e+01
min    5.001709e+06   -9.600000e+01
25%    5.730933e+06   -4.600000e+01
50%    6.070821e+06   -2.500000e+01
75%    6.431951e+06   -1.100000e+01
max    6.842888e+06    0.000000e+00
<ipython-input-31-1e0b9f61469f>:9: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a fu
  print("Correlation Statistic", df_bureau_balance.corr())
Correlation Statistic                 SK_ID_BUREAU  MONTHS_BALANCE
SK_ID_BUREAU        1.000000        0.011873
MONTHS_BALANCE      0.011873        1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Data columns (total 3 columns):
 #   Column          Dtype
---  ------          -----
 0   SK_ID_BUREAU    int64
 1   MONTHS_BALANCE  int64
 2   STATUS          object
dtypes: int64(2), object(1)
memory usage: 624.8+ MB
Summary Information: None
```

## **File**: pos_cash_balance.csv

```
print("Number of Rows: " + str(df_pos_cash_balance.shape[0]))
print("Number of Columns: " + str(df_pos_cash_balance.shape[1]))
print("Number of Total Missing Values: " + str(df_pos_cash_balance.isna().sum().sum()))
print("Data Frame Shape: " + str(df_pos_cash_balance.shape))
print("Number of Missing Values by Feature: " + str(df_pos_cash_balance.isna().sum()))
print("Data Types:",df_pos_cash_balance.dtypes)
print("Data Frame: Data Types", df_pos_cash_balance.dtypes.value_counts())
print("Summary", df_pos_cash_balance.describe())
print("Correlation Statistic", df_pos_cash_balance.corr())
print("Summary Information:", df_pos_cash_balance.info())
```

```
       CNT_INSTALMENT_FUTURE        SK_DPD     SK_DPD_DEF
count          9.975271e+06  1.000136e+07  1.000136e+07
mean           1.048384e+01  1.160693e+01  6.544684e-01
std            1.110906e+01  1.327140e+02  3.276249e+01
min            0.000000e+00  0.000000e+00  0.000000e+00
25%            3.000000e+00  0.000000e+00  0.000000e+00
50%            7.000000e+00  0.000000e+00  0.000000e+00
75%            1.400000e+01  0.000000e+00  0.000000e+00
max            8.500000e+01  4.231000e+03  3.595000e+03
<ipython-input-32-cb2b07444f43>:9: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a fu
  print("Correlation Statistic", df_pos_cash_balance.corr())
Correlation Statistic                        SK_ID_PREV  SK_ID_CURR  MONTHS_BALANCE  CNT_INSTALMENT  \
SK_ID_PREV                1.000000   -0.000336        0.001835        0.003820
SK_ID_CURR               -0.000336    1.000000        0.000404        0.000144
MONTHS_BALANCE            0.001835    0.000404        1.000000        0.336163
CNT_INSTALMENT            0.003820    0.000144        0.336163        1.000000
CNT_INSTALMENT_FUTURE     0.003679   -0.000559        0.271595        0.871276
SK_DPD                   -0.000487    0.003118       -0.018939       -0.060803
SK_DPD_DEF                0.004848    0.001948       -0.000381       -0.014154

                       CNT_INSTALMENT_FUTURE    SK_DPD  SK_DPD_DEF
SK_ID_PREV                          0.003679 -0.000487    0.004848
SK_ID_CURR                         -0.000559  0.003118    0.001948
MONTHS_BALANCE                      0.271595 -0.018939   -0.000381
CNT_INSTALMENT                      0.871276 -0.060803   -0.014154
CNT_INSTALMENT_FUTURE               1.000000 -0.082004   -0.017436
SK_DPD                             -0.082004  1.000000    0.245782
SK_DPD_DEF                         -0.017436  0.245782    1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10001358 entries, 0 to 10001357
Data columns (total 8 columns):
 #   Column                 Dtype
---  ------                 -----
 0   SK_ID_PREV             int64
 1   SK_ID_CURR             int64
 2   MONTHS_BALANCE         int64
 3   CNT_INSTALMENT         float64
 4   CNT_INSTALMENT_FUTURE  float64
 5   NAME_CONTRACT_STATUS   object
 6   SK_DPD                 int64
 7   SK_DPD_DEF             int64
dtypes: float64(2), int64(5), object(1)
memory usage: 610.4+ MB
Summary Information: None
```

## **File**: credit_card_balance.csv

```
print("Number of Rows: " + str(df_credit_card_balance.shape[0]))
print("Number of Columns: " + str(df_credit_card_balance.shape[1]))
print("Number of Total Missing Values: " + str(df_credit_card_balance.isna().sum().sum()))
print("Data Frame Shape: " + str(df_credit_card_balance.shape))
print("Number of Missing Values by Feature: " + str(df_credit_card_balance.isna().sum()))
print("Data Types:",df_credit_card_balance.dtypes)
print("Data Frame: Data Types", df_credit_card_balance.dtypes.value_counts())
print("Summary", df_credit_card_balance.describe())
print("Correlation Statistic", df_credit_card_balance.corr())
print("Summary Information:", df_credit_card_balance.info())
```

```
SK_DPD                                      0.059654  1.000000    0.218950
SK_DPD_DEF                                  0.002156  0.218950    1.000000

[22 rows x 22 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column                      Dtype  
---  ------                      -----  
 0   SK_ID_PREV                  int64  
 1   SK_ID_CURR                  int64  
 2   MONTHS_BALANCE              int64  
 3   AMT_BALANCE                 float64
 4   AMT_CREDIT_LIMIT_ACTUAL     int64  
 5   AMT_DRAWINGS_ATM_CURRENT    float64
 6   AMT_DRAWINGS_CURRENT        float64
 7   AMT_DRAWINGS_OTHER_CURRENT  float64
 8   AMT_DRAWINGS_POS_CURRENT    float64
 9   AMT_INST_MIN_REGULARITY     float64
 10  AMT_PAYMENT_CURRENT         float64
 11  AMT_PAYMENT_TOTAL_CURRENT   float64
 12  AMT_RECEIVABLE_PRINCIPAL    float64
 13  AMT_RECIVABLE               float64
 14  AMT_TOTAL_RECEIVABLE        float64
 15  CNT_DRAWINGS_ATM_CURRENT    float64
 16  CNT_DRAWINGS_CURRENT        int64  
 17  CNT_DRAWINGS_OTHER_CURRENT  float64
 18  CNT_DRAWINGS_POS_CURRENT    float64
 19  CNT_INSTALMENT_MATURE_CUM   float64
 20  NAME_CONTRACT_STATUS        object 
 21  SK_DPD                      int64  
 22  SK_DPD_DEF                  int64  
dtypes: float64(15), int64(7), object(1)
memory usage: 673.9+ MB
Summary Information: None
```

## **File**: previous_application.csv

```
print("Number of Rows: " + str(df_previous_application.shape[0]))
print("Number of Columns: " + str(df_previous_application.shape[1]))
print("Number of Total Missing Values: " + str(df_previous_application.isna().sum().sum()))
print("Data Frame Shape: " + str(df_previous_application.shape))
print("Number of Missing Values by Feature: " + str(df_previous_application.isna().sum()))
print("Data Types:",df_previous_application.dtypes)
print("Data Frame: Data Types", df_previous_application.dtypes.value_counts())
print("Summary", df_previous_application.describe())
print("Correlation Statistic", df_previous_application.corr())
print("Summary Information:", df_previous_application.info())
```

```
Number of Rows: 1670214
Number of Columns: 37
Number of Total Missing Values: 11109336
Data Frame Shape: (1670214, 37)
Number of Missing Values by Feature: 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
Data Types: 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
```

## **File**: installments_payments.csv

```
print("Number of Rows: " + str(df_installments_payments.shape[0]))
print("Number of Columns: " + str(df_installments_payments.shape[1]))
print("Number of Total Missing Values: " + str(df_installments_payments.isna().sum().sum()))
print("Data Frame Shape: " + str(df_installments_payments.shape))
print("Number of Missing Values by Feature: " + str(df_installments_payments.isna().sum()))
print("Data Types:",df_installments_payments.dtypes)
print("Data Frame: Data Types", df_installments_payments.dtypes.value_counts())
print("Summary", df_installments_payments.describe())
print("Correlation Statistic", df_installments_payments.corr())
print("Summary Information:", df_installments_payments.info())
```

```
AMT_PAYMENT                          0.126602        0.937191     1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_PREV              int64
 1   SK_ID_CURR              int64
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   int64
 4   DAYS_INSTALMENT         float64
 5   DAYS_ENTRY_PAYMENT      float64
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
Summary Information: None
```

# Visual Exploratory Data Analysis

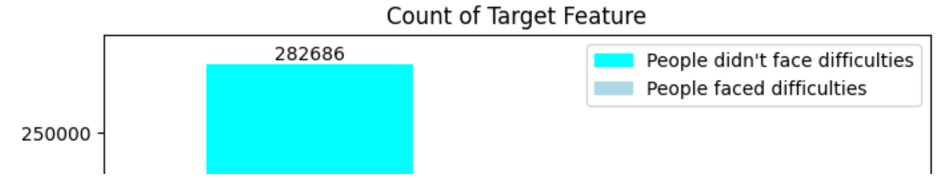
## File: application_train.csv

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

### Feature: Target

**Feature description: Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample, 0 - all other cases)**

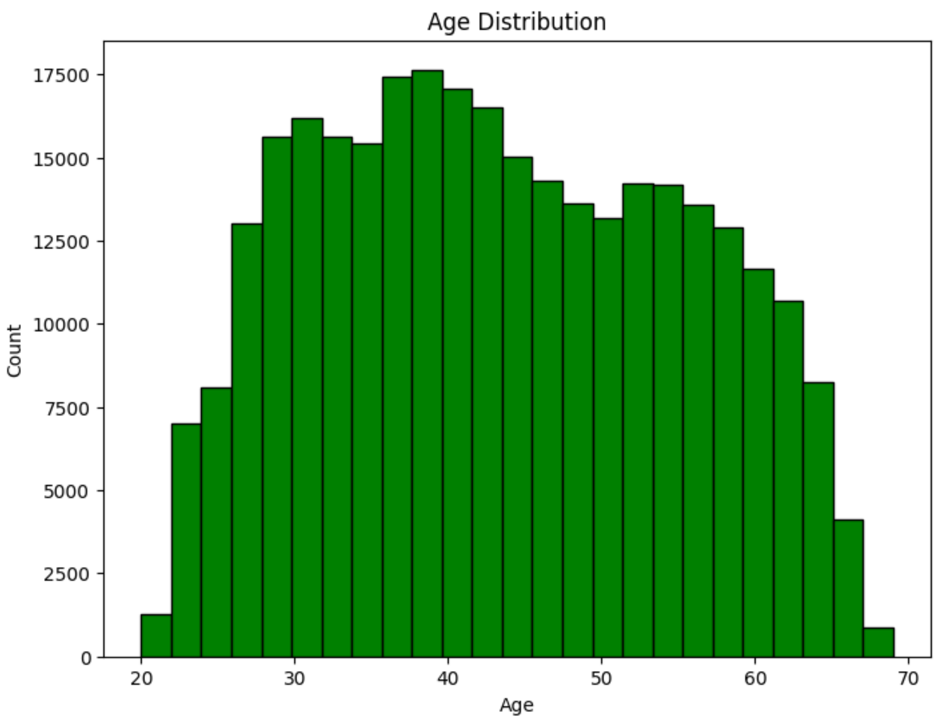
```
import matplotlib.patches as mpatches

fig, ax = plt.subplots(figsize=(8, 6))

barplot = df_application_train['TARGET'].value_counts().plot(kind='bar', color=['cyan', 'lightblue'])

barplot.set_title("Count of Target Feature")
barplot.set_ylabel("Count")
barplot.set_xlabel("Target Values")

legend_labels = ["People didn't face difficulties", "People faced difficulties"]
legend_patches = [mpatches.Patch(color=color, label=label) for color, label in zip(['cyan', 'lightblue'], legend_labels)]
plt.legend(handles=legend_patches, loc='upper right')

for i, count in enumerate(df_application_train['TARGET'].value_counts()):
    plt.text(i, count + 500, str(count), ha='center', va='bottom')

plt.show()
```




We can see from the above that:

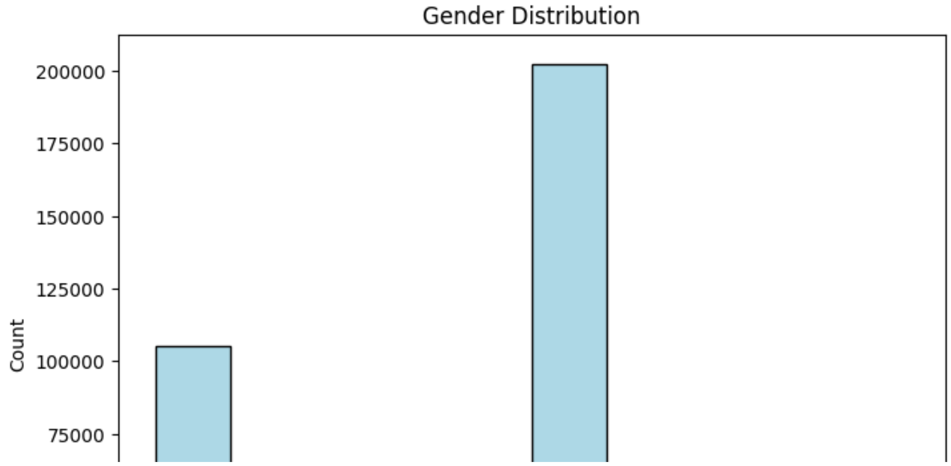
- People who faced difficulties in repaying the loan sum up to **[Class 1]: 24825**
- People who didn't face any difficulties repaying the loan sum up to **[Class 0]: 282686**

```
df_calculated_age = df_application_train['DAYS_BIRTH']//-365
fig, ax = plt.subplots(figsize=(8, 6))

ax.set_title('Age Distribution')
ax.set_ylabel('Count')
ax.set_xlabel('Age')
ax.hist(df_calculated_age, bins=25, color="green", edgecolor="black")

plt.show()
```

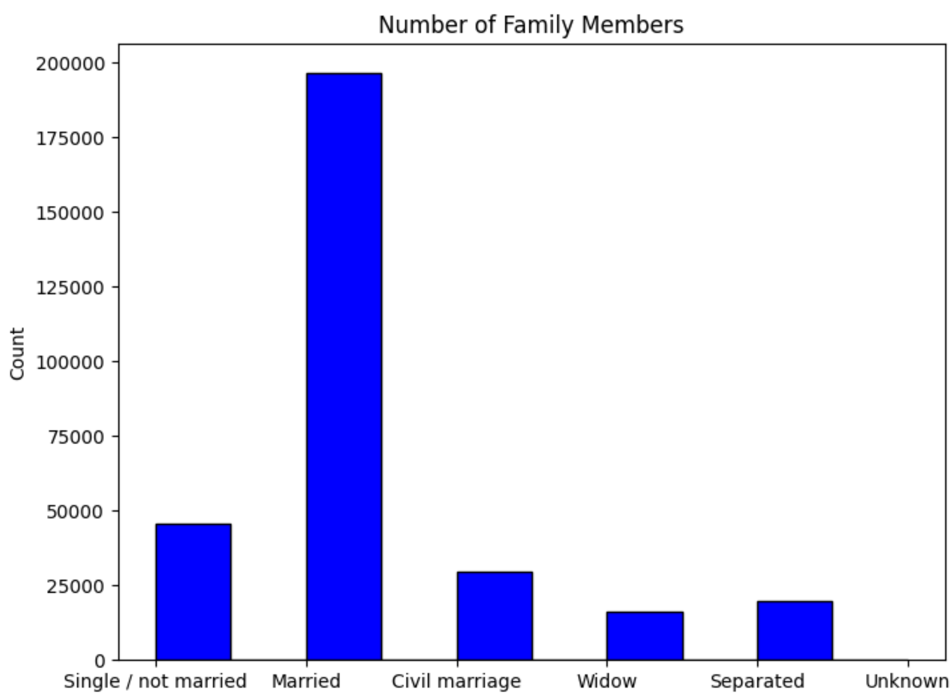



Analysis

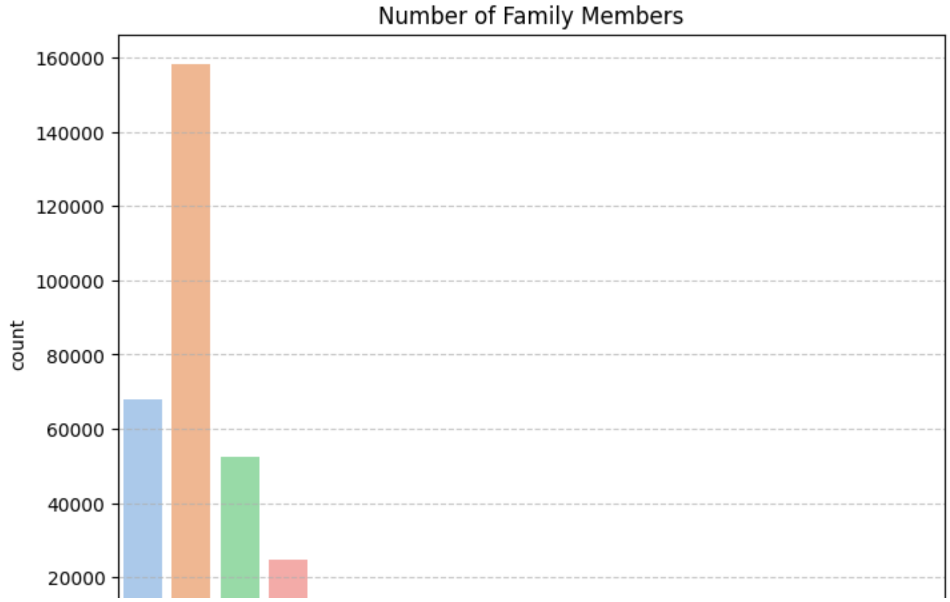
```
fig, ax = plt.subplots(figsize=(8, 6))

ax.set_title('Gender Distribution')
ax.set_ylabel('Count')
ax.set_xlabel('Gender')
ax.hist(df_application_train['CODE_GENDER'], color="lightblue", edgecolor="black")

plt.show()
```



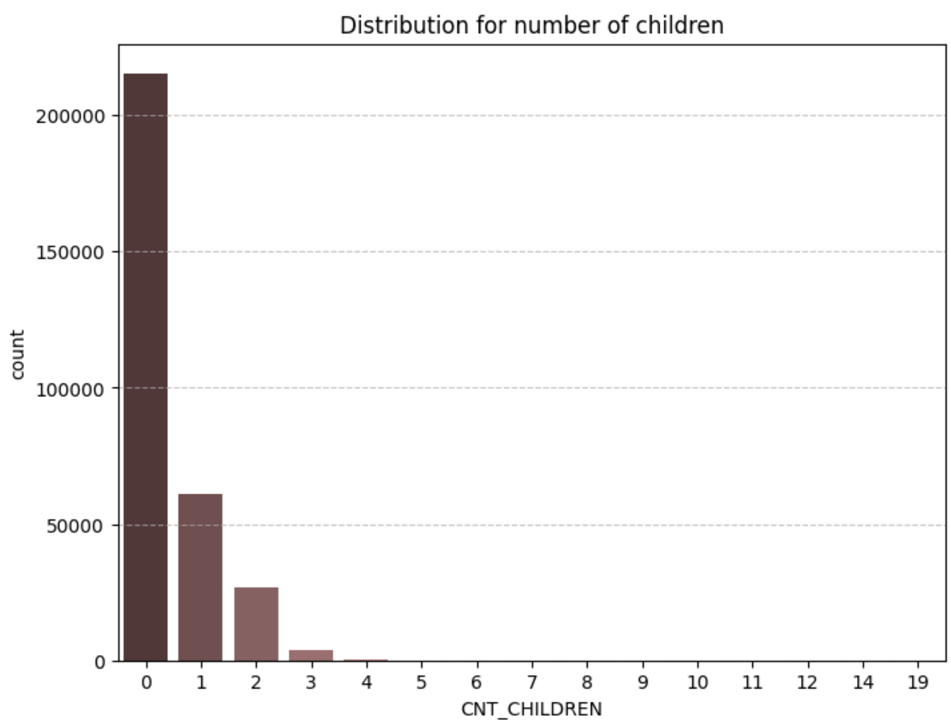

```
import seaborn as sns
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(8, 6))

ax.set_title('Number of Family Members')
ax.set_ylabel('Count')
ax.set_xlabel('')
ax.hist(df_application_train['NAME_FAMILY_STATUS'], color="blue", edgecolor="black")

plt.show()
```




```
fig, ax = plt.subplots(figsize=(8, 6))
ax.set_title('Number of Family Members')
ax.set_ylabel('Count')
ax.set_xlabel('Number of Family Members')

sns.countplot(ax=ax, data=df_application_train, palette="pastel", x="CNT_FAM_MEMBERS")

ax.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()
```



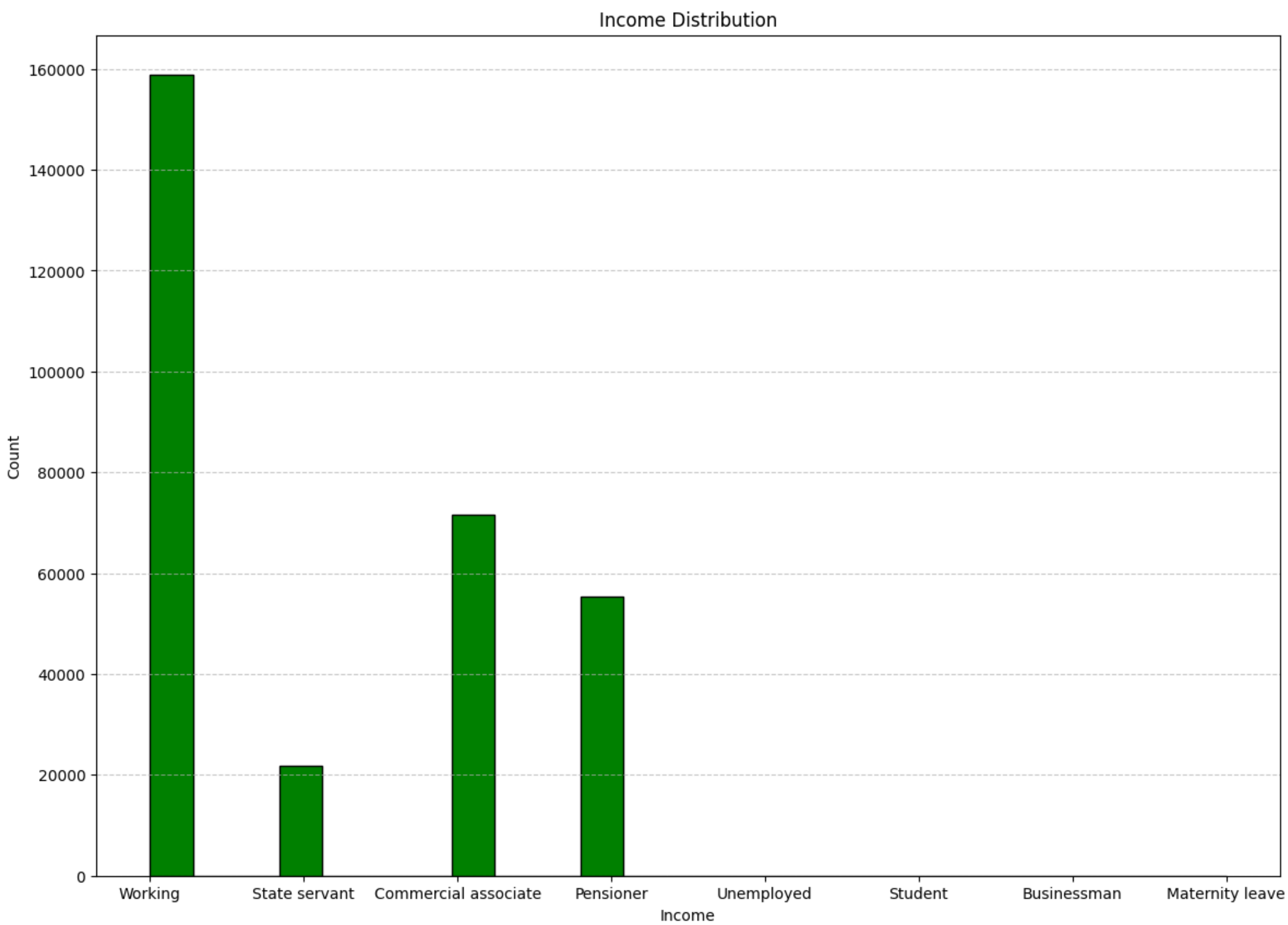

```
fig, ax = plt.subplots(figsize=(8, 6))
ax.set_title('Distribution for number of children')
ax.set_ylabel('Count')
ax.set_xlabel('Number of Children')

sns.countplot(ax=ax, data=df_application_train, palette="pink", x="CNT_CHILDREN")

ax.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()
```




```
education_type_counts = df_application_train['NAME_EDUCATION_TYPE'].value_counts()

print(education_type_counts)
```

```
Secondary / secondary special    218391
Higher education                  74863
Incomplete higher                 10277
Lower secondary                    3816
Academic degree                     164
Name: NAME_EDUCATION_TYPE, dtype: int64
```

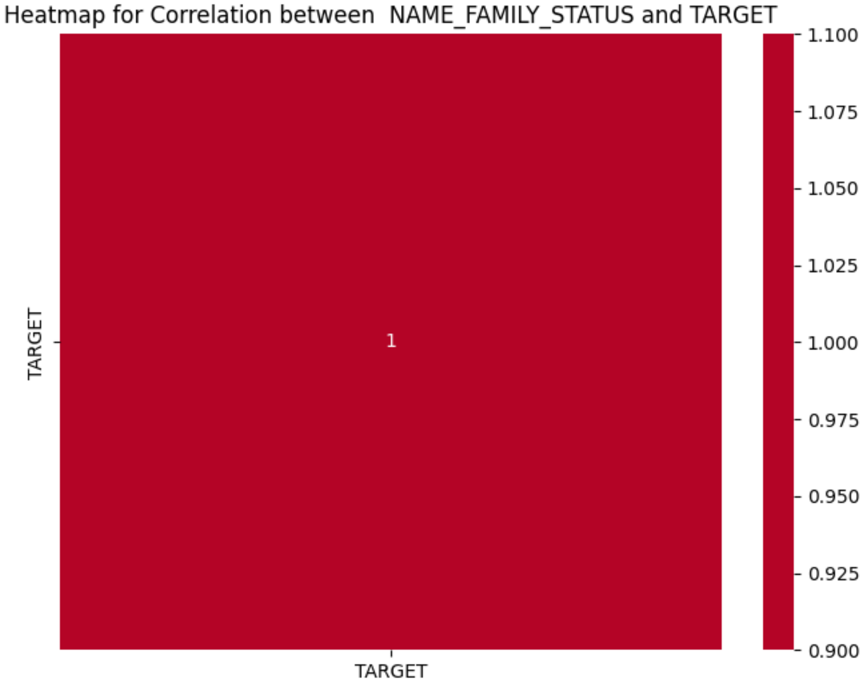
```
fig, ax = plt.subplots(figsize=(14, 10))

ax.set_title('Income Distribution')
ax.set_ylabel('Count')
ax.set_xlabel('Income')
ax.hist(df_application_train['NAME_INCOME_TYPE'], bins=25, color="green", edgecolor="black")
ax.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()
```




## Let's check correlation between features and target variable.

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


features = df_application_train[['NAME_FAMILY_STATUS', 'TARGET']]

# Compute the correlation matrix for the two features
corr_matrix = features.corr()

# Create a heatmap for the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Heatmap for Correlation between  NAME_FAMILY_STATUS and TARGET')
plt.show()
```

```
<ipython-input-45-858e97cceb8f>:8: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a fu
  corr_matrix = features.corr()
```




```
spearman_correlation = df_application_train['NAME_FAMILY_STATUS'].corr(df_application_train['TARGET'], method='spearman')
print(f"The Spearman correlation between NAME_FAMILY_STATUS and TARGET is: {spearman_correlation}")
```

```
The Spearman correlation between NAME_FAMILY_STATUS and TARGET is: -0.001815167766133806
```

```
df_application_train.head()
```

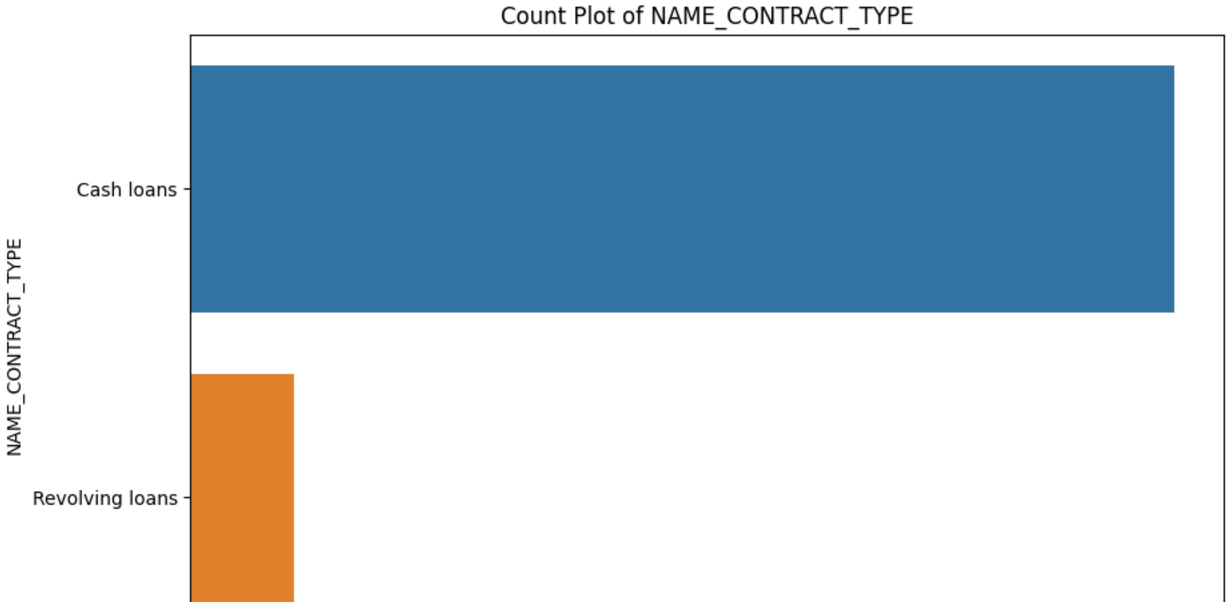
| | SK_ID_CURR | TARGET | NAME_CONTRACT_TYPE | CODE_GENDER | FLAG_OWN_CAR | FLAG_OWN_REALTY | CNT_CHILDREN | AMT_INCOME_TOTAL | AMT_CRE |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 100002 | 1 | Cash loans | M | N | Y | 0 | 202500.0 | 4065 |
| **1** | 100003 | 0 | Cash loans | F | N | N | 0 | 270000.0 | 12935 |
| **2** | 100004 | 0 | Revolving loans | M | Y | Y | 0 | 67500.0 | 1350 |
| **3** | 100006 | 0 | Cash loans | F | N | Y | 0 | 135000.0 | 3126 |
| **4** | 100007 | 0 | Cash loans | M | N | Y | 0 | 121500.0 | 5130 |

5 rows × 122 columns

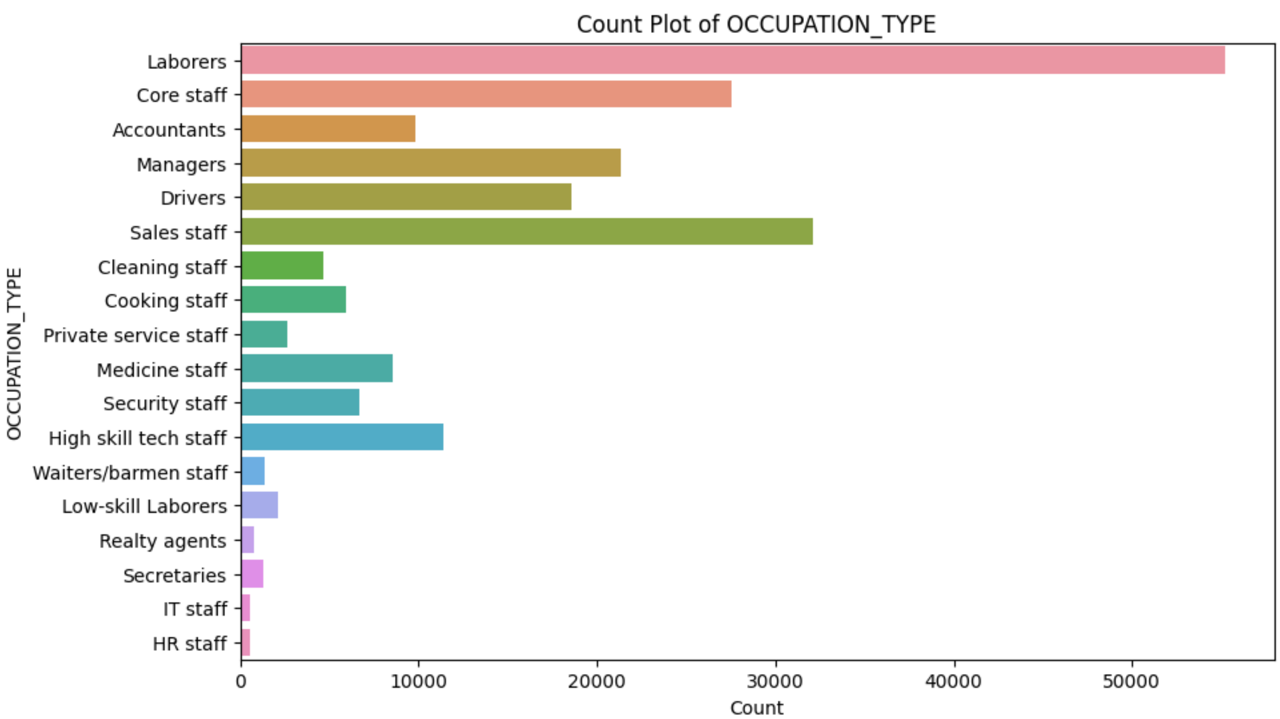
```
column_names=df_application_train.columns.tolist()
print(column_names)
```

```
['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME
```

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

# Count Plot for a categorical variable
plt.figure(figsize=(10, 6))
sns.countplot(data=df_application_train, y='NAME_CONTRACT_TYPE')
plt.title('Count Plot of NAME_CONTRACT_TYPE')
plt.xlabel('Count')
plt.ylabel('NAME_CONTRACT_TYPE')
plt.show()
```

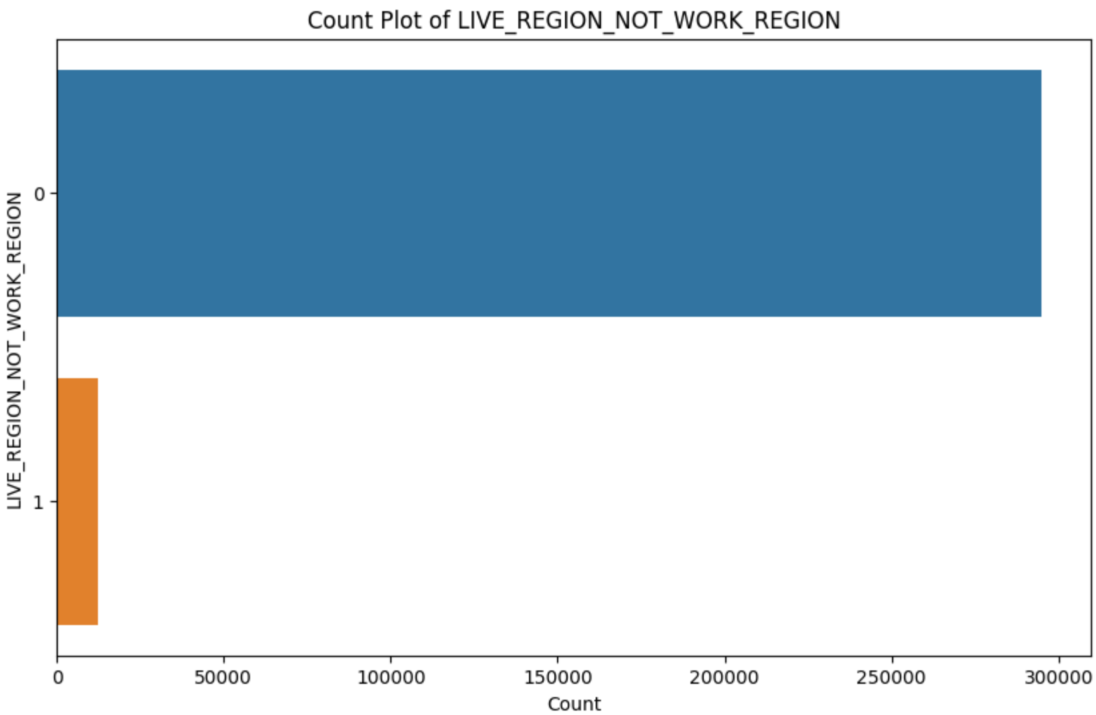



```
import seaborn as sns
import matplotlib.pyplot as plt

# Count Plot for a categorical variable
plt.figure(figsize=(10, 6))
sns.countplot(data=df_application_train, y='OCCUPATION_TYPE')
plt.title('Count Plot of OCCUPATION_TYPE')
plt.xlabel('Count')
plt.ylabel('OCCUPATION_TYPE')
plt.show()
```

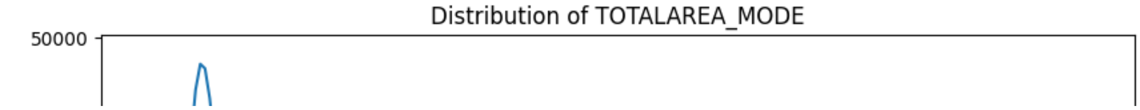



This categorical feature tend to display which occupations are most likely to do loan application. Previously.

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

# Count Plot for a categorical variable
plt.figure(figsize=(10, 6))
sns.countplot(data=df_application_train, y='LIVE_REGION_NOT_WORK_REGION')
plt.title('Count Plot of LIVE_REGION_NOT_WORK_REGION')
plt.xlabel('Count')
plt.ylabel('LIVE_REGION_NOT_WORK_REGION')
plt.show()
```




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

# Histogram of DAYS_FIRST_DUE
plt.figure(figsize=(10, 6))
sns.histplot(df_application_train['TOTALAREA_MODE'], bins=30, kde=True)
plt.title('Distribution of TOTALAREA_MODE')
plt.xlabel('TOTALAREA_MODE')
plt.ylabel('Frequency')
plt.show()
```




Left skewed graph, shows us that total area owned by the applicants which can used as credibility to check whether to grant loan or not

## Handling missing values in application_train

```
missing_values = df_application_train.isnull()
print('Missing Values in entire dataframe',missing_values)
```

```
Missing Values in entire dataframe         SK_ID_CURR  TARGET  NAME_CONTRACT_TYPE  CODE_GENDER  FLAG_OWN_CAR  \
0            False   False               False        False         False
1            False   False               False        False         False
2            False   False               False        False         False
3            False   False               False        False         False
4            False   False               False        False         False
...            ...     ...                 ...          ...           ...
307506       False   False               False        False         False
307507       False   False               False        False         False
307508       False   False               False        False         False
307509       False   False               False        False         False
307510       False   False               False        False         False

        FLAG_OWN_REALTY  CNT_CHILDREN  AMT_INCOME_TOTAL  AMT_CREDIT  \
0                 False         False             False       False
1                 False         False             False       False
2                 False         False             False       False
3                 False         False             False       False
4                 False         False             False       False
...                 ...           ...               ...         ...
307506            False         False             False       False
307507            False         False             False       False
307508            False         False             False       False
307509            False         False             False       False
307510            False         False             False       False

        AMT_ANNUITY  ...  FLAG_DOCUMENT_18  FLAG_DOCUMENT_19  \
0             False  ...             False             False
1             False  ...             False             False
2             False  ...             False             False
3             False  ...             False             False
4             False  ...             False             False
...             ...  ...               ...               ...
307506        False  ...             False             False
307507        False  ...             False             False
307508        False  ...             False             False
307509        False  ...             False             False
307510        False  ...             False             False

        FLAG_DOCUMENT_20  FLAG_DOCUMENT_21  AMT_REQ_CREDIT_BUREAU_HOUR  \
0                  False             False                       False
1                  False             False                       False
2                  False             False                       False
3                  False             False                        True
4                  False             False                       False
...                  ...               ...                         ...
307506             False             False                        True
307507             False             False                        True
307508             False             False                       False
307509             False             False                       False
307510             False             False                       False

        AMT_REQ_CREDIT_BUREAU_DAY  AMT_REQ_CREDIT_BUREAU_WEEK  \
0                           False                       False
1                           False                       False
2                           False                       False
3                            True                        True
4                           False                       False
```

```
missing_values_count = df_application_train.isnull().sum()
print(missing_values_count)
```

```
SK_ID_CURR                     0
TARGET                         0
NAME_CONTRACT_TYPE             0
CODE_GENDER                    0
FLAG_OWN_CAR                   0
                             ...
AMT_REQ_CREDIT_BUREAU_DAY     41519
AMT_REQ_CREDIT_BUREAU_WEEK    41519
AMT_REQ_CREDIT_BUREAU_MON     41519
AMT_REQ_CREDIT_BUREAU_QRT     41519
AMT_REQ_CREDIT_BUREAU_YEAR    41519
Length: 122, dtype: int64
```

```
missing_values_percentage = (df_application_train.isnull().sum() / len(df_application_train)) * 100
###missing values percent per column
print(missing_values_percentage)
```

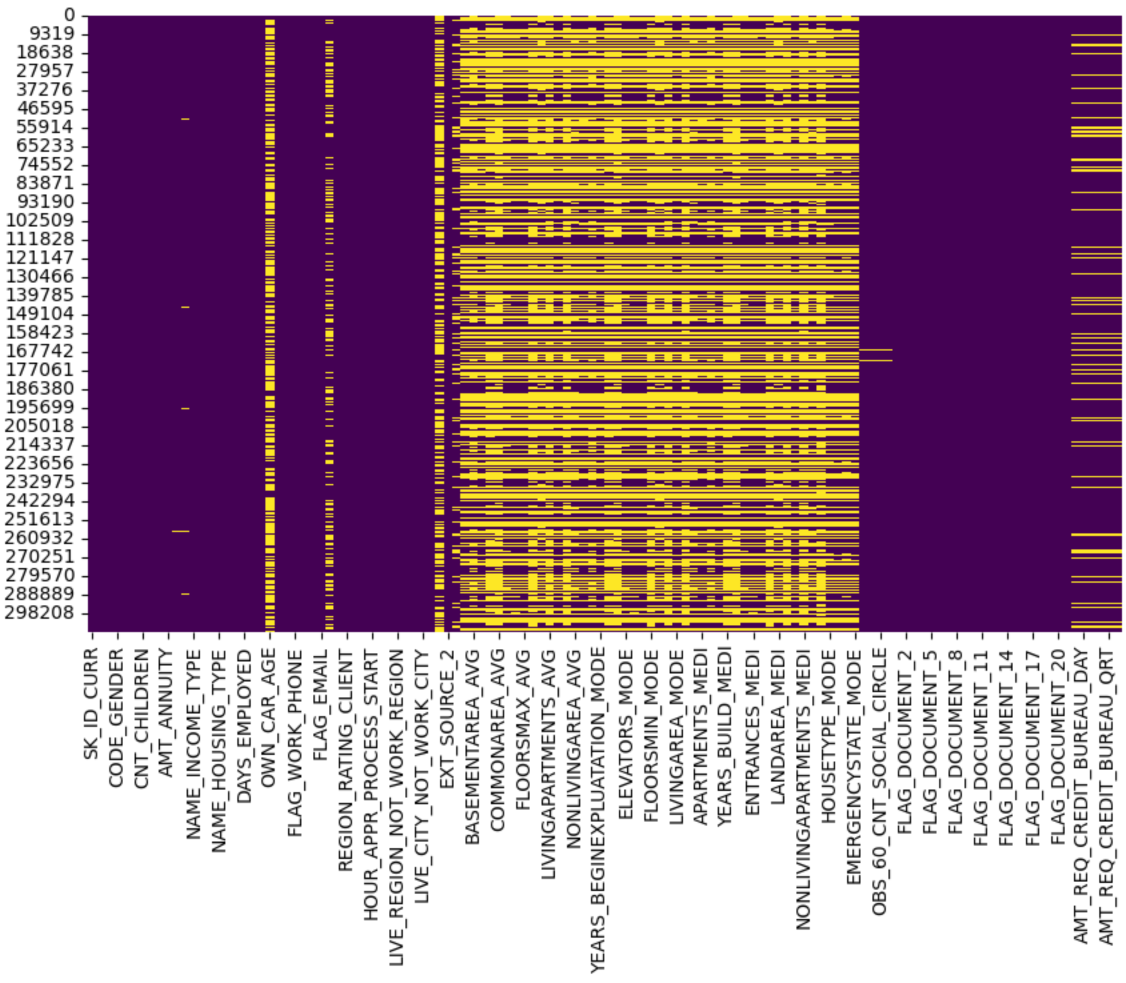
```
SK_ID_CURR                     0.000000
TARGET                         0.000000
NAME_CONTRACT_TYPE             0.000000
CODE_GENDER                    0.000000
FLAG_OWN_CAR                   0.000000
                                 ...
AMT_REQ_CREDIT_BUREAU_DAY     13.501631
AMT_REQ_CREDIT_BUREAU_WEEK    13.501631
AMT_REQ_CREDIT_BUREAU_MON     13.501631
AMT_REQ_CREDIT_BUREAU_QRT     13.501631
AMT_REQ_CREDIT_BUREAU_YEAR    13.501631
Length: 122, dtype: float64
```

```
missing_values_rows = df_application_train[df_application_train.isnull().any(axis=1)]
###filtering dataframe to show only missing values
print(missing_values_rows)
```

```
        SK_ID_CURR  TARGET NAME_CONTRACT_TYPE CODE_GENDER FLAG_OWN_CAR  \
0           100002       1         Cash loans           M            N
1           100003       0         Cash loans           F            N
2           100004       0    Revolving loans           M            Y
3           100006       0         Cash loans           F            N
4           100007       0         Cash loans           M            N
...            ...     ...                ...         ...          ...
307506      456251       0         Cash loans           M            N
307507      456252       0         Cash loans           F            N
307508      456253       0         Cash loans           F            N
307509      456254       1         Cash loans           F            N
307510      456255       0         Cash loans           F            N

       FLAG_OWN_REALTY  CNT_CHILDREN  AMT_INCOME_TOTAL  AMT_CREDIT  \
0                    Y             0          202500.0    406597.5
1                    N             0          270000.0   1293502.5
2                    Y             0           67500.0    135000.0
3                    Y             0          135000.0    312682.5
4                    Y             0          121500.0    513000.0
...                ...           ...               ...         ...
307506               N             0          157500.0    254700.0
307507               Y             0           72000.0    269550.0
307508               Y             0          153000.0    677664.0
307509               Y             0          171000.0    370107.0
307510               N             0          157500.0    675000.0

        AMT_ANNUITY  ...  FLAG_DOCUMENT_18 FLAG_DOCUMENT_19 FLAG_DOCUMENT_20  \
0           24700.5  ...                 0                0                0
1           35698.5  ...                 0                0                0
2            6750.0  ...                 0                0                0
3           29686.5  ...                 0                0                0
4           21865.5  ...                 0                0                0
...             ...  ...               ...              ...              ...
307506      27558.0  ...                 0                0                0
307507      12001.5  ...                 0                0                0
307508      29979.0  ...                 0                0                0
307509      20205.0  ...                 0                0                0
307510      49117.5  ...                 0                0                0

       FLAG_DOCUMENT_21 AMT_REQ_CREDIT_BUREAU_HOUR AMT_REQ_CREDIT_BUREAU_DAY  \
0                     0                        0.0                       0.0
1                     0                        0.0                       0.0
2                     0                        0.0                       0.0
3                     0                        NaN                       NaN
4                     0                        0.0                       0.0
...                 ...                        ...                       ...
307506                0                        NaN                       NaN
307507                0                        NaN                       NaN
```

```
307508              0                    1.0                     0.0
307509              0                    0.0                     0.0
307510              0                    0.0                     0.0

        AMT_REQ_CREDIT_BUREAU_WEEK  AMT_REQ_CREDIT_BUREAU_MON  \
0                              0.0                        0.0
1                              0.0                        0.0
2                              0.0                        0.0
3                              NaN                        NaN
4                              0.0                        0.0
```

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

plt.figure(figsize=(10, 6))
sns.heatmap(df_application_train.isnull(), cbar=False, cmap='viridis')
plt.show()
```



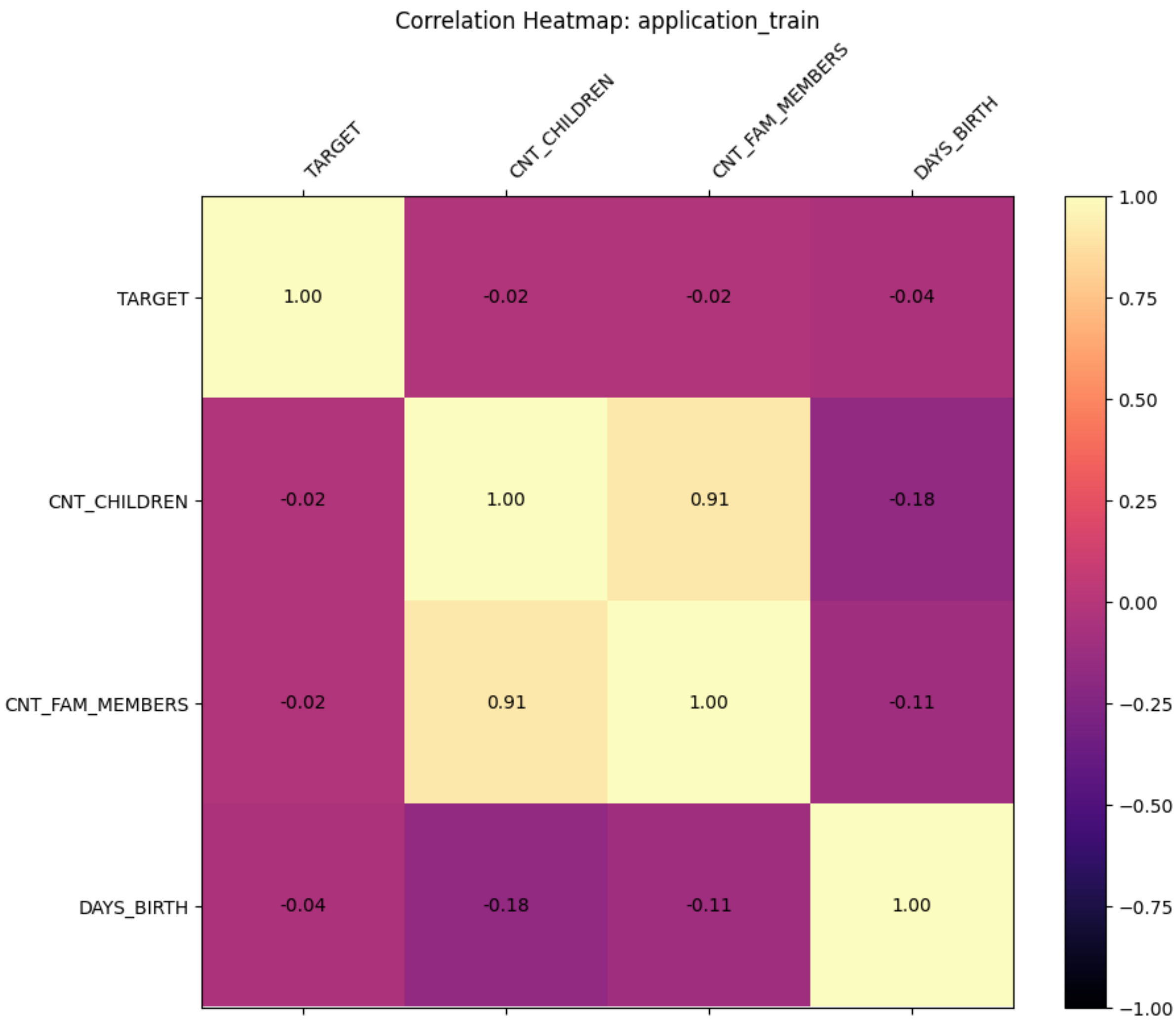

## Heatmap of missing values in application_train

```
missing_values = df_application_train['DAYS_BIRTH'].isnull()
print(missing_values)
```

```
df_application_train.dropna(inplace=True)
```

```
correlation_data = df_application_train[["TARGET", "CNT_CHILDREN", "CNT_FAM_MEMBERS", "DAYS_BIRTH"]]
correlation_data["DAYS_BIRTH"] = abs(correlation_data["DAYS_BIRTH"])
correlation_data = correlation_data.corr()

fig, ax = plt.subplots(figsize=(10, 8))
cax = ax.matshow(correlation_data, cmap="magma", vmin=-1.0, vmax=1.0)
fig.colorbar(cax)

ticks = list(range(len(correlation_data.columns)))
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(correlation_data.columns, rotation=45, ha="left")
ax.set_yticklabels(correlation_data.columns)

for i in range(len(correlation_data.columns)):
    for j in range(len(correlation_data.columns)):
        text = ax.text(j, i, f"{correlation_data.iloc[i, j]:.2f}", ha="center", va="center", color="black")

plt.title("Correlation Heatmap: application_train")
plt.show()
```

```
<ipython-input-60-1d187026b71b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  correlation_data["DAYS_BIRTH"] = abs(correlation_data["DAYS_BIRTH"])
```

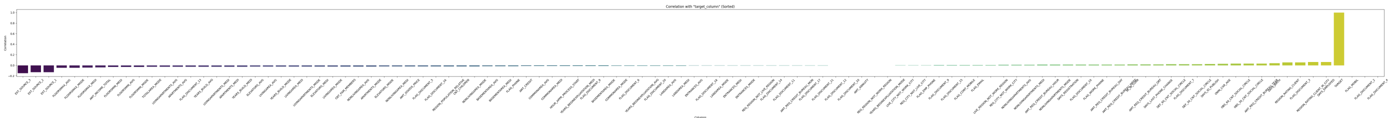



```
datacorr = df_application_train.corrwith(df_application_train['TARGET'])
datacorr
```

```
<ipython-input-61-25930280b08d>:1: FutureWarning: The default value of numeric_only in DataFrame.corrwith is deprecated. In
  datacorr = df_application_train.corrwith(df_application_train['TARGET'])
SK_ID_CURR                    0.015474
TARGET                        1.000000
```

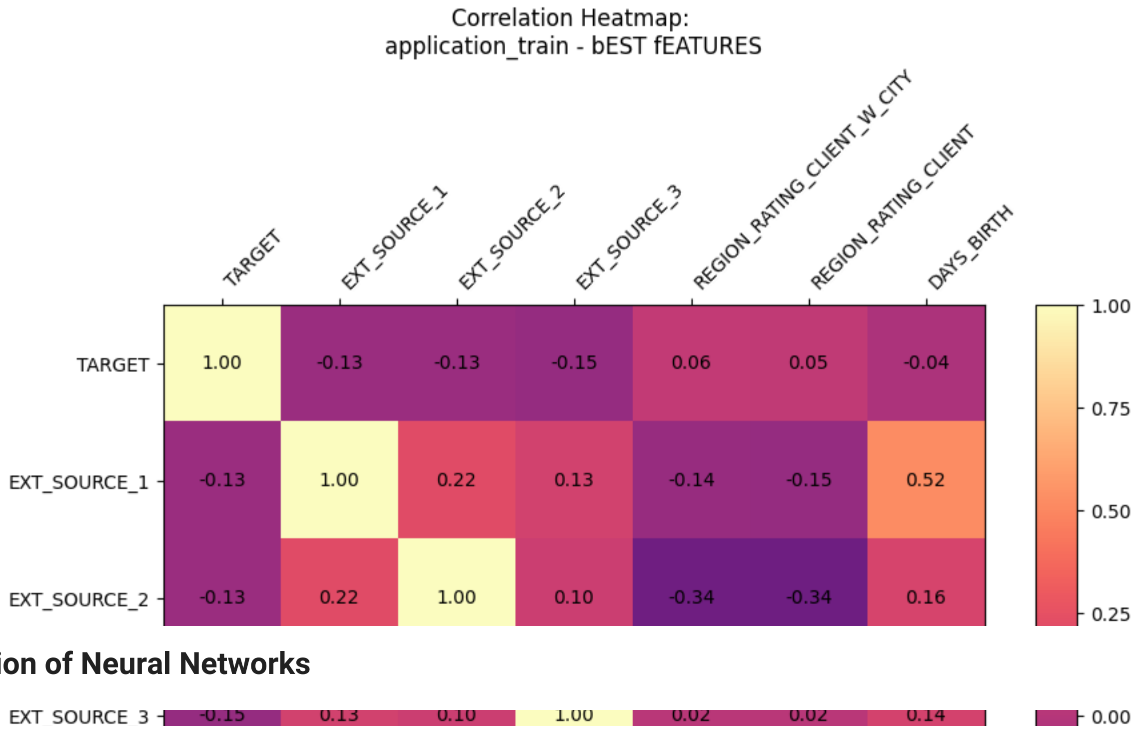
```
CNT_CHILDREN                  -0.019993
AMT_INCOME_TOTAL              -0.039762
AMT_CREDIT                    -0.014634
                                 ...
AMT_REQ_CREDIT_BUREAU_DAY      0.014616
AMT_REQ_CREDIT_BUREAU_WEEK     0.015000
AMT_REQ_CREDIT_BUREAU_MON     -0.004202
AMT_REQ_CREDIT_BUREAU_QRT      0.016465
AMT_REQ_CREDIT_BUREAU_YEAR     0.033832
Length: 106, dtype: float64
```

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

app_sorted = datacorr.sort_values()
plt.figure(figsize=(70, 6))
sns.barplot(x=app_sorted.index, y=app_sorted.values, palette='viridis')

plt.title('Correlation with "target_column" (Sorted)')
plt.xlabel('Columns')
plt.ylabel('Correlation')

plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```



```
correlation_data = df_application_train[['TARGET','EXT_SOURCE_1','EXT_SOURCE_2','EXT_SOURCE_3','REGION_RATING_CLIENT_W_CITY','RI
correlation_data["DAYS_BIRTH"] = abs(correlation_data["DAYS_BIRTH"])
correlation_data = correlation_data.corr()

fig, ax = plt.subplots(figsize=(10, 8))
cax = ax.matshow(correlation_data, cmap="magma", vmin=-1.0, vmax=1.0)
fig.colorbar(cax)

ticks = list(range(len(correlation_data.columns)))
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(correlation_data.columns, rotation=45, ha="left")
ax.set_yticklabels(correlation_data.columns)

for i in range(len(correlation_data.columns)):
    for j in range(len(correlation_data.columns)):
        text = ax.text(j, i, f"{correlation_data.iloc[i, j]:.2f}", ha="center", va="center", color="black")

plt.title("Correlation Heatmap: \napplication_train - bEST fEATURES")
plt.show()
```

```
<ipython-input-63-4111a7d6d40f>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  correlation_data["DAYS_BIRTH"] = abs(correlation_data["DAYS_BIRTH"])
```




## Implementation of Neural Networks

```
import torch
import torch.utils.data
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error,roc_auc_score,roc_curve, auc,f1_score
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
import matplotlib.pyplot as plt
import datetime
import random
import string

from torch.utils.data import Dataset, TensorDataset, DataLoader
from sklearn.feature_selection import VarianceThreshold

torch.manual_seed(0)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# load data
hcdr_application = pd.read_csv("clean_final.csv")


X = hcdr_application.drop('TARGET', axis = 1)
y = hcdr_application.TARGET
print("Shapes:", X.shape, y.shape)

# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle = True)
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=0.15, random_state=42, shuffle=True

y_train = y_train.to_numpy()
y_validation = y_validation.to_numpy()
y_test = y_test.to_numpy()


# convert numpy arrays to tensors
X_train_tensor = torch.from_numpy(np.array(X_train))
X_valid_tensor = torch.from_numpy(np.array(X_validation))
X_test_tensor = torch.from_numpy(np.array(X_test))
y_train_tensor = torch.from_numpy(y_train.astype('int'))
y_valid_tensor = torch.from_numpy(y_validation.astype('int'))
y_test_tensor = torch.from_numpy(y_test.astype('int'))

# create TensorDataset in PyTorch
hcdr_train = torch.utils.data.TensorDataset(X_train_tensor, y_train_tensor)
hcdr_valid = torch.utils.data.TensorDataset(X_valid_tensor, y_valid_tensor)
hcdr_test = torch.utils.data.TensorDataset(X_test_tensor, y_test_tensor)


train_batch_size = 32
valid_test_batch_size = 16
trainloader_hcdr = torch.utils.data.DataLoader(hcdr_train, batch_size=train_batch_size, shuffle=True, num_workers=2)
validloader_hcdr = torch.utils.data.DataLoader(hcdr_valid, batch_size=valid_test_batch_size, shuffle=True, num_workers=2)
testloader_hcdr = torch.utils.data.DataLoader(hcdr_test, batch_size=valid_test_batch_size, shuffle=True, num_workers=2)



def run_hcdr_model(
    hidden_layer_neurons=[32, 16, 8],
    opt=optim.SGD,
    epochs=5,
    learning_rate=1e-3
):

    D_in = X_test.shape[1]  # Input layer neurons depend on the input dataset shape
    D_out = 2  # Output layer neurons here, 2 classes: 0 and 1
```

```
    str_neurons = [str(h) for h in hidden_layer_neurons]
    arch_string = f"{D_in}-{'-'.join(str_neurons)}-{D_out}"

    layers = [
        torch.nn.Linear(D_in, hidden_layer_neurons[0]),  # X.matmul(W1)
        nn.ReLU(),  # ReLU( X.matmul(W1))
    ]

    # Add hidden layers
    for i in range(1, len(hidden_layer_neurons)):
        prev, curr = hidden_layer_neurons[i - 1], hidden_layer_neurons[i]
        layers.append(torch.nn.Linear(prev, curr))
        layers.append(nn.ReLU())
    # layers.append(nn.Sigmoid())


    # Add final layer

    layers.append(nn.Linear(hidden_layer_neurons[-1], D_out)) # Relu( X.matmul(W1)).matmul(W2))


    model = torch.nn.Sequential(*layers)

    model.to(device)


    loss_fn = nn.CrossEntropyLoss()  #for classfication
    optimizer = opt(model.parameters(), lr=learning_rate)

    #summary(model, (4, 20))
    print('-'*50)
    print('Model:')
    print(model)
    print('-'*50)



    loss_history = []
    acc_history = []
    def train_epoch(epoch, model, loss_fn, opt, train_loader):
        running_loss = 0.0
        count = 0
        y_pred = []
        epoch_target = []

        for batch_id, data in enumerate(train_loader):
            inputs, target = data[0].to(device), data[1].to(device)
            # 1:zero the grad, 2:forward pass, 3:calculate loss,  and 4:backprop!
            opt.zero_grad()
            preds = model(inputs.float())

            # compute loss and gradients
            loss = loss_fn(preds, target)

            loss.backward() #calculate nabla_w
            loss_history.append(loss.item())
            opt.step()  #update W
            y_pred.extend(torch.argmax(preds, dim=1).tolist())
            epoch_target.extend(target.tolist())

            running_loss += loss.item()
            count += 1

        loss = np.round(running_loss/count, 3)

        #accuracy
        correct = (np.array(y_pred) == np.array(epoch_target))
        accuracy = correct.sum() / correct.size
        accuracy = np.round(accuracy, 3)
        return loss, accuracy



    def evaluate_model(epoch, model, loss_fn, opt, data_loader, tag = "Test"):
        overall_loss = 0.0
        count = 0
```

```
        y_pred = []
        epoch_target = []
        for i,data in enumerate(data_loader):
            inputs, target = data[0].to(device), data[1].to(device)
            preds = model(inputs.float())

            loss = loss_fn(preds, target)           # compute loss value

            overall_loss += (loss.item())  # compute total loss to save to logs
            y_pred.extend(torch.argmax(preds, dim=1).tolist())
            epoch_target.extend(target.tolist())
            count += 1

        # compute mean loss
        loss = np.round(overall_loss/count, 3)
        #accuracy
        correct = (np.array(y_pred) == np.array(epoch_target))
        accuracy = correct.sum() / correct.size
        accuracy = np.round(accuracy, 3)
        return loss, accuracy



    for epoch in range(epochs):
        # print(f"Epoch {epoch+1}")
        train_loss, train_accuracy = train_epoch(epoch, model, loss_fn, optimizer, trainloader_hcdr)
        valid_loss, valid_accuracy = evaluate_model(epoch, model, loss_fn, optimizer, validloader_hcdr, tag = "Validation")
        print(f"Epoch {epoch+1}: Train Accuracy: {train_accuracy}\t Validation Accuracy: {valid_accuracy}")
    print("-"*50)
    test_loss, test_accuracy = evaluate_model(epoch, model, loss_fn, opt, testloader_hcdr, tag="Test")

    return arch_string, train_accuracy, valid_accuracy, test_accuracy, model
```

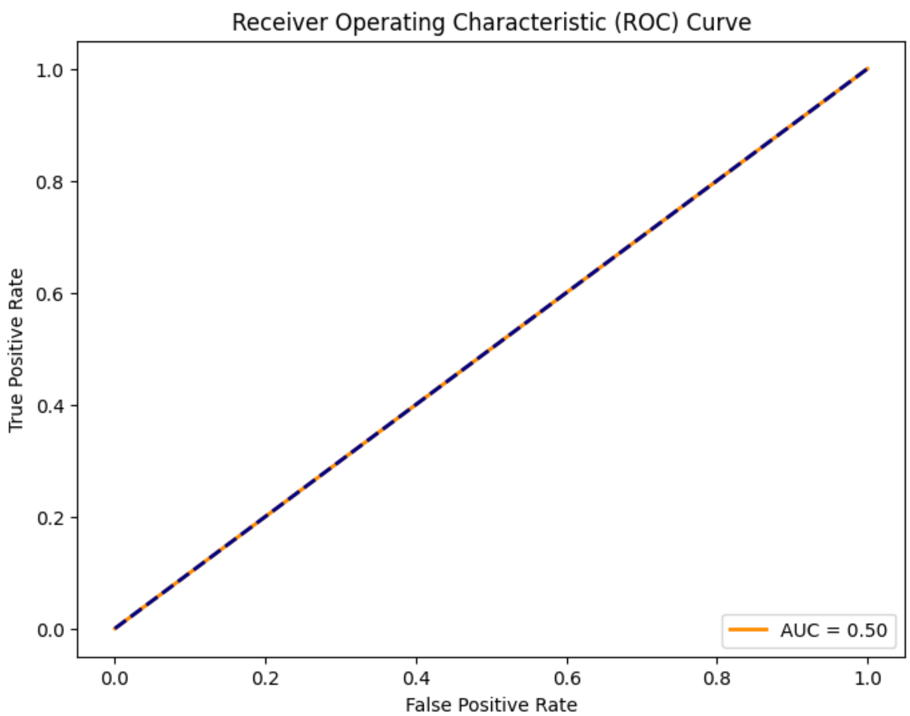
```
 Shapes: (307511, 246) (307511,)
```

```
import pandas as pd
torch.manual_seed(0)

# hidden_layer_neurons = [32,16,8]
hidden_layer_neurons = [300,200,64,8]
opt = optim.Adam   # optim.SGD, Optim.Adam, etc.
epochs = 3
learning_rate = 2e-3


arch_string, train_accuracy, valid_accuracy, test_accuracy,model = run_hcdr_model(
    hidden_layer_neurons,
    opt,
    epochs,
    learning_rate
)


try: Log
except : Log = pd.DataFrame(
    columns=[
        "Architecture string",
        "Optimizer",
        "Epochs",
        "Train accuracy",
        "Valid accuracy",
        "Test accuracy",
    ]
)

Log.loc[len(Log)] = [
    arch_string,
    f"{opt}",
    f"{epochs}",
    f"{train_accuracy * 100}%",
    f"{valid_accuracy * 100}%",
    f"{test_accuracy * 100}%",
]

Log
```

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=246, out_features=300, bias=True)
  (1): ReLU()
  (2): Linear(in_features=300, out_features=200, bias=True)
  (3): ReLU()
  (4): Linear(in_features=200, out_features=64, bias=True)
  (5): ReLU()
  (6): Linear(in_features=64, out_features=8, bias=True)
  (7): ReLU()
  (8): Linear(in_features=8, out_features=2, bias=True)
)
--------------------------------------------------
Epoch 1: Train Accuracy: 0.911   Validation Accuracy: 0.919
Epoch 2: Train Accuracy: 0.919   Validation Accuracy: 0.919
Epoch 3: Train Accuracy: 0.919   Validation Accuracy: 0.919
--------------------------------------------------
```

| | Architecture string | Optimizer | Epochs | Train accuracy | Valid accuracy | Test accuracy |
|---|---|---|---|---|---|---|
| 0 | 246-32-16-8-2 | <class 'torch.optim.adam.Adam'> | 3 | 91.9% | 91.9% | 92.0% |
| 1 | 246-300-200-64-8-2 | <class 'torch.optim.adam.Adam'> | 3 | 91.9% | 91.9% | 92.0% |

```
with torch.no_grad():
    model.eval()
    outputs = model(X_test_tensor.float().cuda())

predicted_probs = outputs[:, 1].cpu().numpy()
predicted_probs[predicted_probs<0] = 0
predicted_probs[predicted_probs>0] = 1
true_labels = y_test_tensor.numpy()


# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(true_labels, predicted_probs)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



```
from sklearn.preprocessing import LabelEncoder
pd.set_option('display.float_format', lambda x: '%.5f' % x)
pd.set_option('mode.chained_assignment', None)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
if torch.cuda.is_available():
  print("GPU - ", torch.cuda.get_device_name(0))

# SET HYPERPARAMETERS
test_size = 0.2
epochs = 12
batch_size = 320
learningrate= 0.000008
hp_emb_drop = 0.04
nnlayers = [800, 350]
hp_ps = [0.001,0.01]

# LOAD DATA
application_train_df = pd.read_csv('dataset/application_train.csv').sample(frac = 1)
application_test_df = pd.read_csv('dataset/application_test.csv')
previous_application_df = pd.read_csv('dataset/previous_application.csv')

application_train_df['CSV_SOURCE'] = 'application_train.csv'
application_test_df['CSV_SOURCE'] = 'application_test.csv'
df = pd.concat([application_train_df, application_test_df])

# PREPARING previous_applications.csv
temp_previous_df = previous_application_df.groupby('SK_ID_CURR', as_index=False).agg({'NAME_CONTRACT_STATUS': lambda x: ','.joi
temp_previous_df['has_only_approved'] = np.where(temp_previous_df['NAME_CONTRACT_STATUS'] == 'Approved', '1', '0')
temp_previous_df['has_been_rejected'] = np.where(temp_previous_df['NAME_CONTRACT_STATUS'].str.contains('Refused'), '1', '0')

# JOIN DATA
df = pd.merge(df, temp_previous_df, on='SK_ID_CURR', how='left')

# Feature engineering
# total_amt_req_credit_bureau
df['total_amt_req_credit_bureau'] = (
  df['AMT_REQ_CREDIT_BUREAU_YEAR'] * 1 +
  df['AMT_REQ_CREDIT_BUREAU_QRT'] * 2 +
  df['AMT_REQ_CREDIT_BUREAU_MON'] * 8 +
  df['AMT_REQ_CREDIT_BUREAU_WEEK'] * 16 +
  df['AMT_REQ_CREDIT_BUREAU_DAY'] * 32 +
  df['AMT_REQ_CREDIT_BUREAU_HOUR'] * 64)
df['total_amt_req_credit_bureau_isnull'] = np.where(df['total_amt_req_credit_bureau'].isnull(), '1', '0')
df['total_amt_req_credit_bureau'].fillna(0, inplace=True)

#has_job
df['has_job'] = np.where(df['NAME_INCOME_TYPE'].isin(['Pensioner', 'Student', 'Unemployed']), '1', '0')

# has_children
df['has_children'] = np.where(df['CNT_CHILDREN'] > 0, '1', '0')

# cluster_days_employed
def cluster_days_employed(x):
    days = x['DAYS_EMPLOYED']
    if days > 0:
      return 'not available'
    else:
      days = abs(days)
      if days < 30:
        return 'less 1 month'
      elif days < 180:
        return 'less 6 months'
      elif days < 365:
        return 'less 1 year'
      elif days < 1095:
        return 'less 3 years'
      elif days < 1825:
        return 'less 5 years'
      elif days < 3600:
        return 'less 10 years'
      elif days < 7200:
        return 'less 20 years'
      elif days >= 7200:
        return 'more 20 years'
      else:
        return 'not available'
```

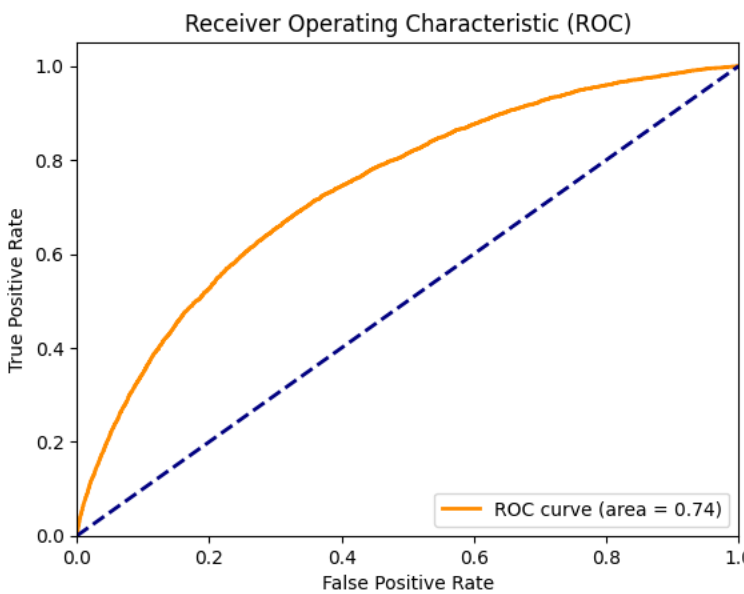
```
    df['cluster_days_employed'] = df.apply(cluster_days_employed, axis=1)

    # custom_ext_source_3
    def cluster_ext_source(x):
        if str(x) == 'nan':
          return 'not available'
        else:
          if x < 0.1:
            return 'less 0.1'
          elif x < 0.2:
            return 'less 0.2'
          elif x < 0.3:
            return 'less 0.3'
          elif x < 0.4:
            return 'less 0.4'
          elif x < 0.5:
            return 'less 0.5'
          elif x < 0.6:
            return 'less 0.6'
          elif x < 0.7:
            return 'less 0.7'
          elif x < 0.8:
            return 'less 0.8'
          elif x < 0.9:
            return 'less 0.9'
          elif x <= 1:
            return 'less 1'
    df['cluster_ext_source_1'] = df['EXT_SOURCE_1'].apply(lambda x: cluster_ext_source(x))
    df['cluster_ext_source_2'] = df['EXT_SOURCE_2'].apply(lambda x: cluster_ext_source(x))
    df['cluster_ext_source_3'] = df['EXT_SOURCE_3'].apply(lambda x: cluster_ext_source(x))

    # house_variables_sum
    house_data = ['APARTMENTS_AVG','APARTMENTS_MEDI','APARTMENTS_MODE','BASEMENTAREA_AVG',
      'BASEMENTAREA_MEDI','BASEMENTAREA_MODE','COMMONAREA_AVG','COMMONAREA_MEDI',
      'COMMONAREA_MODE','ELEVATORS_AVG','ELEVATORS_MEDI','ELEVATORS_MODE','EMERGENCYSTATE_MODE',
      'ENTRANCES_AVG','ENTRANCES_MEDI','ENTRANCES_MODE','FLOORSMAX_AVG','FLOORSMAX_MEDI',
      'FLOORSMAX_MODE','FLOORSMIN_AVG','FLOORSMIN_MEDI','FLOORSMIN_MODE','FONDKAPREMONT_MODE',
      'HOUSETYPE_MODE','LANDAREA_AVG','LANDAREA_MEDI','LANDAREA_MODE','LIVINGAPARTMENTS_AVG',
      'LIVINGAPARTMENTS_MEDI','LIVINGAPARTMENTS_MODE','LIVINGAREA_AVG','LIVINGAREA_MEDI','LIVINGAREA_MODE',
      'NONLIVINGAPARTMENTS_AVG','NONLIVINGAPARTMENTS_MEDI','NONLIVINGAPARTMENTS_MODE','NONLIVINGAREA_AVG',
      'NONLIVINGAREA_MEDI','NONLIVINGAREA_MODE','TOTALAREA_MODE','WALLSMATERIAL_MODE',
      'YEARS_BEGINEXPLUATATION_AVG','YEARS_BEGINEXPLUATATION_MEDI','YEARS_BEGINEXPLUATATION_MODE',
      'YEARS_BUILD_AVG','YEARS_BUILD_MEDI','YEARS_BUILD_MODE']
    df['house_variables_sum'] = df[house_data].sum(axis=1)
    df['house_variables_sum_isnull'] = np.where(df['house_variables_sum'].isnull(), '1', '0')
    df['house_variables_sum'].fillna(value=df['house_variables_sum'].median(), inplace=True)



    num_columns = [
      'AMT_ANNUITY', 'AMT_CREDIT', 'AMT_GOODS_PRICE', 'AMT_INCOME_TOTAL',
      'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_ID_PUBLISH', 'DAYS_REGISTRATION',
      'CNT_CHILDREN', 'CNT_FAM_MEMBERS', 'DAYS_EMPLOYED', 'DAYS_LAST_PHONE_CHANGE',
      'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'total_amt_req_credit_bureau',
      'house_variables_sum']
    cat_columns = [
      'CODE_GENDER', 'CSV_SOURCE', 'FLAG_OWN_CAR', 'NAME_EDUCATION_TYPE', 'FLAG_OWN_REALTY', 'OCCUPATION_TYPE', 'ORGANIZATION_TYPE'
      'NAME_CONTRACT_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'NAME_INCOME_TYPE', 'NAME_TYPE_SUITE',
      'has_only_approved', 'has_been_rejected', 'has_job', 'has_children', 'cluster_days_employed',
      'cluster_ext_source_1', 'cluster_ext_source_2', 'cluster_ext_source_3',
      'total_amt_req_credit_bureau_isnull', 'house_variables_sum_isnull']

    target_column = ['TARGET']
    df = df[num_columns + cat_columns + target_column]

    #   Impute missing values
    for numerical_column in num_columns:
      if df[numerical_column].isnull().values.any():
        df[numerical_column + '_isnull'] = np.where(df[numerical_column].isnull(), '1', '0')
      df[numerical_column].fillna(value=df[numerical_column].median(), inplace=True)

    for categorical_column in cat_columns:
      df[categorical_column].fillna('NULL', inplace=True)

    # Standard
    minmax_scaler = preprocessing.MinMaxScaler()
    df[num_columns] = pd.DataFrame(minmax_scaler.fit_transform(df[num_columns]))
```

```
#label_encoding
cat_columns.remove('CSV_SOURCE')

for column in cat_columns:
  df[column] = LabelEncoder().fit_transform(df[column].astype(str))
  df[column] = df[column].astype('category')

# split dataset
train_df = df[df['CSV_SOURCE'] == 'application_train.csv']
train_output_df = pd.DataFrame(train_df['TARGET'], columns=['TARGET'])

test_df = df[df['CSV_SOURCE'] == 'application_test.csv']

# remove columns
train_df.drop(columns=['CSV_SOURCE', 'TARGET'], axis=0, inplace=True)
test_df.drop(columns=['CSV_SOURCE', 'TARGET'], axis=0, inplace=True)

#validation
x_train, x_valid, y_train, y_valid = train_test_split(train_df, train_output_df, test_size=test_size, random_state=42)


def create_tensors(input_df):
  stack = []
  for column in input_df.columns:
    if input_df.dtypes[column] == np.int64 or input_df.dtypes[column] == np.float64:
      stack.append(input_df[column].astype(np.float64))
    else:
      stack.append(input_df[column].cat.codes.values)
  return torch.tensor(np.stack(stack, 1), dtype=torch.float)

tensor_x_train_cat = create_tensors(x_train[cat_columns]).float().to(device)
tensor_x_train_num = create_tensors(x_train[num_columns]).float().to(device)
tensor_y_train = torch.tensor(y_train.values).flatten().float().to(device)

tensor_x_valid_cat = create_tensors(x_valid[cat_columns]).float().to(device)
tensor_x_valid_num = create_tensors(x_valid[num_columns]).float().to(device)
tensor_y_valid = torch.tensor(y_valid.values).flatten().float().to(device)

tensor_x_test_cat = create_tensors(test_df[cat_columns]).float().to(device)
tensor_x_test_num = create_tensors(test_df[num_columns]).float().to(device)

# CREATE CATEGORICAL EMBEDDING SIZES
cat_columns_size = [len(df[column].cat.categories) for column in cat_columns]
categorical_embedding_sizes = [(col_size, min(50, (col_size + 1) // 2)) for col_size in cat_columns_size]

# DEFINE NEURAL NETWORK MODEL
class Model(nn.Module):
  def __init__(self, embedding_size, input_size, num_numerical_cols, layers, ps):
    super().__init__()

    self.all_embeddings = nn.ModuleList([nn.Embedding(ni, nf) for ni, nf in embedding_size])
    self.emb_drop = nn.Dropout(hp_emb_drop)

    self.bn_cont = nn.BatchNorm1d(num_numerical_cols)

    layer = []
    for i, elem in enumerate(layers):
      layer.append(nn.Linear(input_size, elem))
      layer.append(nn.ReLU(inplace=True))
      layer.append(nn.BatchNorm1d(layers[i]))
      layer.append(nn.Dropout(ps[i]))
      input_size = elem
    layer.append(nn.Linear(layers[-1], 1))

    self.layers = nn.Sequential(*layer)

  def forward(self, x_c, x_n):

    embeddings = [e(x_c[:,i].long()) for i, e in enumerate(self.all_embeddings)]

    x = torch.cat(embeddings, 1)
    x = self.emb_drop(x)

    x_n = self.bn_cont(x_n)

    x = torch.cat([x, x_n], 1)
```

```
        x = self.layers(x)

        return x

    # INSTANCIATE MODEL

    num_numerical_cols = tensor_x_train_num.shape[1]

    num_categorical_cols = sum((nf for ni, nf in categorical_embedding_sizes))
    initial_input_size = num_categorical_cols + num_numerical_cols

    model = Model(categorical_embedding_sizes, initial_input_size, num_numerical_cols, layers=nnlayers, ps=hp_ps)
    sigmoid = nn.Sigmoid()
    loss_function = nn.BCELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learningrate)
    model.to(device)

    # TRAIN NEURAL NETWORK MODEL
    print("TRAINING MODEL...")
    train_tensor_dataset = TensorDataset(tensor_x_train_cat, tensor_x_train_num, tensor_y_train)
    train_loader = DataLoader(dataset=train_tensor_dataset, batch_size=batch_size, shuffle=True)

    model.train()

    tot_y_train_in = []
    tot_y_train_out = []

    for epoch in range(epochs):
      train_losses = []
      for x_cat, x_num, y in train_loader:
        y_train = model(x_cat, x_num)
        single_loss = loss_function(sigmoid(y_train.squeeze()), y)
        single_loss.backward()
        optimizer.step()

        train_losses.append(single_loss.item())
        tot_y_train_in.append(y)
        tot_y_train_out.append(y_train)
      epoch_loss = 1.0 * sum(train_losses) / len(train_losses)
      epoch_auc = roc_auc_score(torch.cat(tot_y_train_in).cpu().numpy(), torch.cat(tot_y_train_out).cpu().detach().numpy())
      tot_y_train_in = []
      tot_y_train_out = []
      print("\tepoch: " + str(epoch) + "\tloss: " + str(epoch_loss) + "\tauc: " + str(epoch_auc))

    # VALIDATE NEURAL NETWORK MODEL
    print("VALIDATING MODEL...")
    validation_tensor_dataset = TensorDataset(tensor_x_valid_cat, tensor_x_valid_num, tensor_y_valid)
    validation_loader = DataLoader(dataset=validation_tensor_dataset, batch_size=batch_size, shuffle=True)

    valid_losses = []

    model.eval()

    tot_y_valid_in = []
    tot_y_valid_out = []

    with torch.no_grad():
      for x_cat, x_num, y in validation_loader:
        y_valid = model(x_cat, x_num)
        validation_loss = loss_function(sigmoid(y_valid.squeeze()), y)
        valid_losses.append(validation_loss.item())

        tot_y_valid_in.append(y_valid)
        tot_y_valid_out.append(y)

      valid_loss = round(1.0 * sum(valid_losses) / len(valid_losses), 5)
      print("\tloss: " + str(valid_loss))
      valid_auc = roc_auc_score(torch.cat(tot_y_valid_out).cpu(), torch.cat(tot_y_valid_in).cpu())
      print("\tauc: " + str(valid_auc))

    # MAKE PREDICTIONS
    print("MAKING PREDICTIONS...")
    with torch.no_grad():
      y_test = model(tensor_x_test_cat, tensor_x_test_num)

    # GENERATE SUBMISSION.csv
    print("GENERATING SUBMISSIONS...")
```

```
nn_prediction_df = pd.DataFrame(y_test.cpu().detach().numpy()).astype("float")
x_scaled = minmax_scaler.fit_transform(nn_prediction_df)
nn_prediction_df = pd.DataFrame(x_scaled)
nn_prediction_df = pd.concat([nn_prediction_df, application_test_df['SK_ID_CURR']], axis=1)
nn_prediction_df.columns = ['TEMP_TARGET', 'SK_ID_CURR']
nn_prediction_df['TARGET'] = nn_prediction_df['TEMP_TARGET']
nn_prediction_df = nn_prediction_df[['SK_ID_CURR', 'TARGET']]
nn_prediction_df.to_csv('submission1.csv', index=False)

print("EXECUTION COMPLETED.")
```

```
GPU -  Tesla T4
<ipython-input-17-87fe9576a828>:121: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only
  df['house_variables_sum'] = df[house_data].sum(axis=1)
TRAINING MODEL...
        epoch: 0        loss: 0.7139497920967668        auc: 0.6309687141142222
        epoch: 1        loss: 0.667470389984364 auc: 0.6886699362857125
        epoch: 2        loss: 0.6343034151626656        auc: 0.7217084690823279
        epoch: 3        loss: 0.574318533523495 auc: 0.7253117017023164
        epoch: 4        loss: 0.5002443059523177        auc: 0.7288723717833956
        epoch: 5        loss: 0.4283307173782269        auc: 0.7368439610165431
        epoch: 6        loss: 0.36568192538886446       auc: 0.7429465721912977
        epoch: 7        loss: 0.31617653191632195       auc: 0.7477063433327675
        epoch: 8        loss: 0.28012465405603687       auc: 0.7544540997999702
        epoch: 9        loss: 0.25712734706922685       auc: 0.7631410760378139
        epoch: 10       loss: 0.2450951326769257        auc: 0.7690683618517538
        epoch: 11       loss: 0.24061422689450887       auc: 0.7762495197384395
VALIDATING MODEL...
        loss: 0.25533
        auc: 0.738632800087059
MAKING PREDICTIONS...
```




```
GENERATING SUBMISSIONS...
EXECUTION COMPLETED.
```

```
model
```

```
Model(
  (all_embeddings): ModuleList(
    (0): Embedding(3, 2)
    (1): Embedding(2, 1)
    (2): Embedding(5, 3)
    (3): Embedding(2, 1)
    (4): Embedding(19, 10)
    (5): Embedding(58, 29)
    (6): Embedding(2, 1)
    (7-8): 2 x Embedding(6, 3)
    (9-10): 2 x Embedding(8, 4)
    (11-12): 2 x Embedding(3, 2)
    (13-14): 2 x Embedding(2, 1)
    (15): Embedding(9, 5)
    (16): Embedding(11, 6)
    (17-18): 2 x Embedding(10, 5)
    (19): Embedding(2, 1)
```

```
      (20): Embedding(1, 1)
    )
    (emb_drop): Dropout(p=0.04, inplace=False)
    (bn_cont): BatchNorm1d(17, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (layers): Sequential(
      (0): Linear(in_features=107, out_features=800, bias=True)
      (1): ReLU(inplace=True)
      (2): BatchNorm1d(800, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (3): Dropout(p=0.001, inplace=False)
      (4): Linear(in_features=800, out_features=350, bias=True)
      (5): ReLU(inplace=True)
      (6): BatchNorm1d(350, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (7): Dropout(p=0.01, inplace=False)
      (8): Linear(in_features=350, out_features=1, bias=True)
    )
  )
```

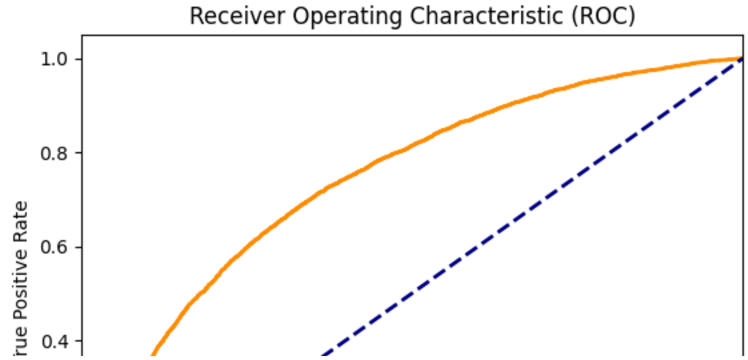
```
layer_sizes = []
for module in model.layers:
    if isinstance(module, nn.Linear):
        layer_sizes.append(module.out_features)

# Create the architecture string
architecture_string = '-'.join(map(str, layer_sizes))
print(f'architecture_string', architecture_string)
```

```
architecture_string 800-350-1
```

```
##score
from sklearn.metrics import accuracy_score, recall_score, precision_score,f1_score

y_true = torch.cat(tot_y_valid_out).cpu().numpy()
y_pred_probs = torch.cat(tot_y_valid_in).cpu().numpy()

y_pred = (y_pred_probs > 0.5).astype(int)

accuracy = accuracy_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
precision_score = precision_score(y_true, y_pred)
f1_score = f1_score(y_true,y_pred)

print(f'Accuracy: {accuracy:.4f}')
print(f'f1_score: {f1_score:.4f}')
print(f'precision: {precision_score:.4f}')
print(f'Recall: {recall:.4f}')
```

```
Accuracy: 0.9194
f1_score: 0.0263
precision: 0.5076
Recall: 0.0135
```

```
# Calculate AUC-ROC and plot the curve
fpr, tpr, _ = roc_curve(torch.cat(tot_y_valid_out).cpu(), torch.cat(tot_y_valid_in).cpu())
roc_auc = auc(fpr, tpr)

plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```




**Discussion:**

Architecture string of Neural Networks - 800-350-1, 246-32-16-8-2

Architecture string used - 800-350-1 Multiple architecttures were used to build a optimized Neural Networks and have choose neural network with accuracy was 91% with auc curve - 0.74, f1_score: 0.0263

## Key Experiments

## Logistic Regression

```
#Importing Libraries
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve
import matplotlib.pyplot as plt
```

```
#Loading dataset
df_clean = pd.read_csv('clean_final.csv')
```

```
df1 = df_clean.copy()
```