Home Credit Default Risk (HCDR)- Group no. 25

Phase 4

Team Members:

Chinmay Pathak (crpathak@iu.edu)

Shruti Houji (sghouji@iu.edu)

Dipak Bange (djbange@iu.edu)

Shrirang Mhalgi (srmhalgi@iu.edu) (Phase 4 Leader)

Abstract:

The main aim of this project is to tackle the burden of financial instability and make the job of loan lending organizations easier and fairer. To achieve this, our main task in this phase will be building a neural network pipeline and experimenting with different architectures of it; check if there is any leakage in the pipelines; and find out if we are violating any cardinal sins. We used the PyTorch framework to build a neural network and experimented with different training parameters such as activation functions, iterations, hidden layers, and learning rate. Also, we observed the leakage in the dataset by finding the difference between the categorical features of the training and test datasets. We attempted to balance the dataset where the output (Target=1/0) is equal, however, the accuracy remained the same. We performed hyperparameter tuning for our models, and experimented with different parameters of XGBoost with a total of 240 experiments, LGBM with a total of 180 experiments, and a neural network with a total of 32 experiments. When compared with the previous phase, our accuracy for XGBoost remained the same with 91.96% for the validation dataset, but the LGBM showed a slight improvement of 1%. For neural networks, the public score obtained for the Kaggle submission is 74.6%. The main challenge in this phase we faced was the data imbalance, the hyperparameter tuning for the neural network model to increase the accuracy and the time needed to train the model.

Index

Phase Leadership Plan

Team Member	Shrirang(Phase Leader)	Dipak	Chinmay	Shruti
Phase	Phase 4	Phase 3	Phase 2	Phase 1
	Building Neural Network Model	RFM engineering	Data Collection	Abstract
	Experiment with at least 2 different Network architecture of NN	Feature selection- Dropping redundant features	Exploratory Data Analysis(EDA)	Data Description
	Report neural network architecture in string form	Analysis of Feature Importance	EDA Visualizations	
	Go through your Pipeline and check if there is any leakage	Pipeline updates-Feature selection		Data Analysis
	Violating any cardinal sins of ML	Model Development-Logistic regression, Random Forest,Decision Tree	Merging the CSV files	ML pipeline analysis
	Data Description	Hyperparameter Tuning-Shuffle Split, GridSearch CV, Confusion matrix, ROC curve	Building Machine Learning Pipelines	Credit assignment
	Tasks to be tackled	Model Validation(Comparing results and finding the best model)	Results and Discussion of the results	Overall proposal
	Workflow Diagrams	Video Presentation	Conclusions	EDA
	Hyperparameter Tuning	Team and plan updates	Making Report	
	Graphical Representation of model performance- of Neural Network	Project Description	Abstract	
	Loss function updates	Project Abstract	Video and in class presentation	
	Validation	Conclusions	Team and plan updates	
	Metrics update	Final Report	Project Description	
	Summary		Bibliography	
	Final Report			
	Abstract			
	Video and in class presentation			
	Team and plan updates			
	Project Description			
	Bibliography			
	Results and Discussion of the Results			
	Conclusion			

Credit Assignment Plan

Phase 4

Names	Task	Description
Dipak	Experiment with different network architecture of Neural Network	Experimented on NN on different activations functions and changing the parameters(for eg. no of iterations)
	Checking Pipeline Leakage	Verifying if any test data is present in the train data
	Hyperparameter Tuning	For model XG Boost, Random Forest using GridSearchCV
	Tasks to be tackled for this phase	Building neural network model
	Model Validation	Compared the results of Neural Network architecture built
	Conclusion	
	Creating report in a jupyter notebook file	Explained all the details of the report in the notebook file
	Video Presentation	Recording the video and editing it
Chinmay	Project Abstract	Created using STAR methodology
	Build Neural Network model using PyTorch	Created a code structure for building neural network model
	Data Description	Added description in the report for NN model built
	Metrics Update	Updated the accuracy of different built model
	Hyperparameter Tuning	For model Decision Tree using GridSearchCV
	Model Validation	Compared the results of different models built
	Project Description	Data Description added in the report
Shruti	Credit Assignment Plan	Divided the work among all group members and created a table in the report for it
	Violating any cardinal sins in ML model	Figured out deadly sins of ML model while training
	Workflow Diagram	Created work flow diagram for phase 4
	Loss Function Updates	Updated loss function for neural network
	Hyperparameter Tuning	For model Logistic Regression using GridSearchCV
	Model Validation	Compared the results of different models that were build
	PPT	Created slides for video presenttion
	Project Description	Added workflow diagrams in the report
	Final Report	Added subparts of the report needed to update the results obtained in the python file
Shrirang	Phase Leader Plan	Found out the tasks to be completed for phase 4
	Report neural network architecture in string format	Presented NN architecture in string format
	Graphical Representation of model performance of NN	Plotted graph for analysis
	Hyperparameter Tuning	For model LGBM classifier using GridSearchCV
	Model Validation	Compared the results of different models that were build
	Project Description	Task to be tackled added in the report
	Results and Discussion of Results	interpreted, which means explained, analysed, and compare the results
	Final Report	Added subparts of the report needed to update the results obtained in the python file

Phase 3

Names	Task	Description
Dipak	Phase Leader Plan	Listed down all the work needed for submission for phase 3
	Engineering RFM	Worked on recency metric
	Feature Engineering-Feature selection	Dropping redundant features
	Model Development	Random Forest
	Hyperparameter Tuning	Shuffle Split
	Model Validation	Compared the results of different models built
	Conclusion	Observed conclusions and added in the report
	Modeling Pipelines	-best results (1 to three) for all experiments you conducted with the following details -The families of input features used -For train/valid/test record the following in a Pandas DataFrame:
Chinmay	Project Abstract	Created using STAR methodology
	Engineering RFM	Worked on frequency metric
	Feature Engineering-Feature Selection	Adding new feature
	Model Development	Logistic regression
	Hyperparameter Tuning	GridSearchCV
	Model Validation	Compared the results of different models built
	Project Description	Data Description added in the report
	Modeling Piplelines	- Baseline experiment -Any additional experiments - Final model tuned
	Results and discussion of Results	interpreted, which means explained, analysed, and compare the results
Shruti	Credit Assignment Plan	Divided the work among all group members and created a table in the report for it
	Engineering RFM	Worked on monetory value metric
	Feature Engineering-Feature Selection	Adding new feature
	Hyperparameter Tuning	Confusion Matrix
	Model Validation	Compared the results of different models that were build
	PPT	Created slides for video presenttion
	Project Description	Added workflow diagrams in the report
	Modeling Piplelines	- Loss function used (data loss and regularization parts) in latex - Number of experiments conducted -Experiment table with the following details per experiment:
Shrirang	Model Development	Decsion Tree
	Hyperparameter Tuning	ROC / AUC Curve
	Analysis of Feature Importance	Did analysis of newly added features
	Model Validation	Compared the results of different models that were build
	Project Description	Task to be tackled added in the report
	Selection of the method after feature Engineering	method chosen and why in presentation and results/discussion
	Modeling Piplelines	- A visualization of the modeling pipeline (s) and subpipelines if necessary - Families of input features and count per family - Number of input features - Hyperparameters and settings considered

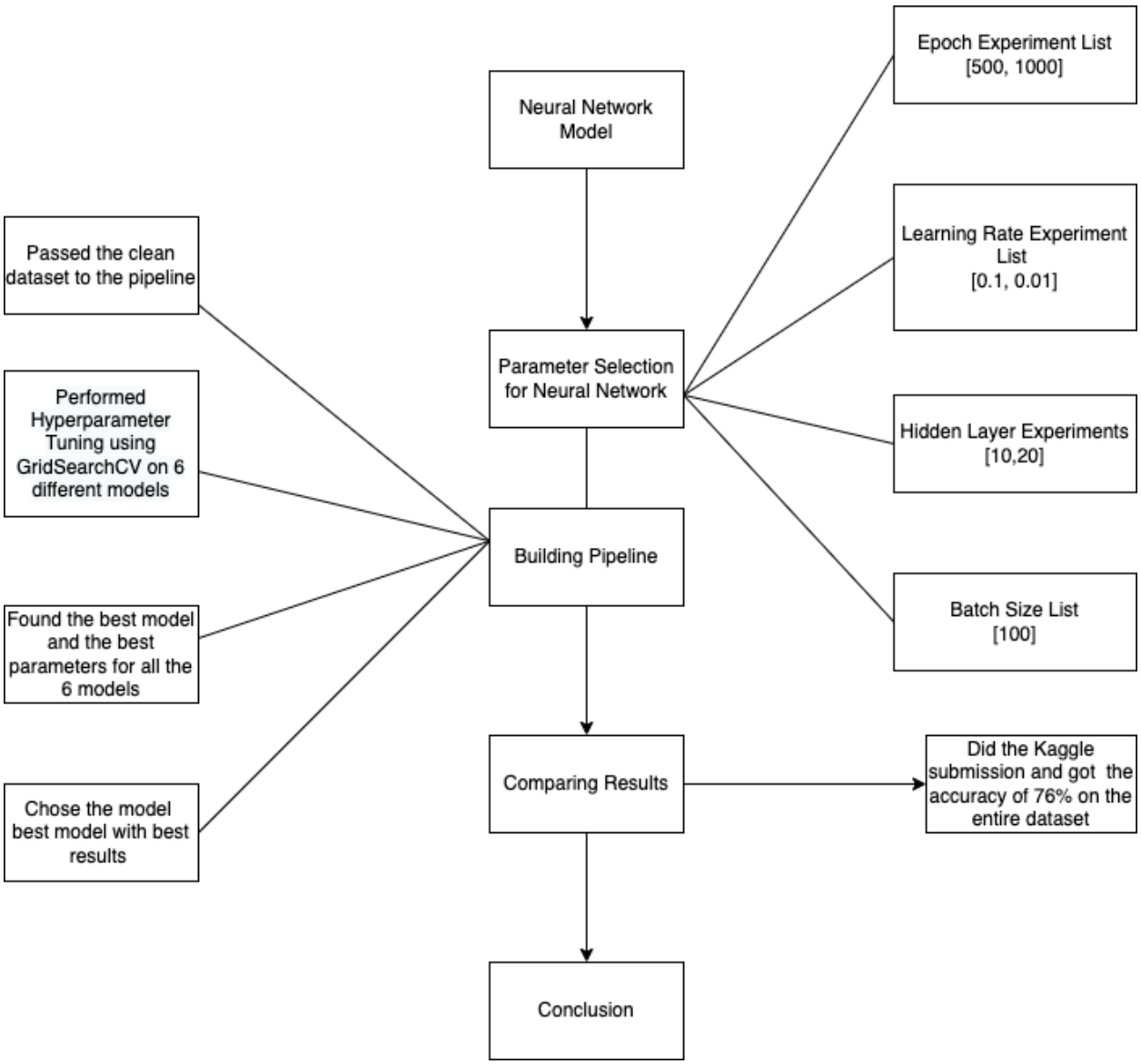
Phase 2

Names	Task	Description
Shruti	Project Abstract	Created using STAR methodology
	Team photo	Through zoom call
	EDA	credit_card_balance.csv, previous_application.csv did EDA on these two csv files
	EDA Visualizations	credit_card_balance.csv, previous_application.csv did EDA visualizations on these two csv files
	Video and class Presentation	Created PPT and recorded a video presentation
	Team and Plan Updates	Created a table for a team plan in the report
Dipak	EDA	application_train.csv, application_test.csv-did EDA on these two csv files
	EDA Visualizations	application_train.csv, application_test.csv-did EDA visualizations on these two csv files
	Modeling pipelines	-A visualization of the modeling pipeline (s) and subpipelines if necessary -Families of input features and count per family -Number of input features -Loss function used (data loss and regularization parts) in latex -Number of experiments conducted
	Results and Discussion of the Results	Interpretated results
Chinmay	Project Description	Created a report for the description of data, tasks to be tackled, provided diagrams to aid understanding the workflow
	EDA	POS_cash_balance.csv, installment_payments.csv-did EDA on these two csv files
	EDA Visualizations	POS_cash_balance.csv, installment_payments.csv-did EDA visualizations on these two csv files
	EDA Report	Analysis of the EDA done and making a final report of it
	Credit Assignment Plan	Divided the work among all group members
Shrirang	EDA	bareau_balance.csv, bareau.csv-did EDA on these two csv files
	EDA Visualizations	bareau_balance.csv, bareau.csv-did EDA visualizations on these two csv files
	Modeling pipelines	-Experiment table with the following details per experiment: - Baseline experiment - The families of input features used -For train/valid/test record the following in a Pandas DataFrame: -Accuracy - AUC/ROC
	Conclusion	restate project focus, restate the hypothesis,summarize,significance of results,future of project, closing thoughts

Phase 1

Names	Task	Description
Shruti	Abstract	Created using STAR methodology
	Team photo	Through zoom call
	Credit Assignment Plan	Divided the tasks equally among all group members
Dipak	Machine Learning Pipelines(block diagram)	Analyzed dataset and created pipelines for feature engineering and data modeling
	Gantt chart	Created timeline for four different phases and assigned work to all members
Chinmay	Data Description	Analyzed the CSV files by downloading the dataset given on Kaggle. Wrote the description by understanding the structure of the dataset
	EDA on Jupyter Notebook	Did the EDA by doing tasks like finding corelation and finding the missing values
Shrirang	Machine Learning and metrics	Identified which models can be applied for classification task
	Analysis of different ML models and accuracy metrics	Went deeply through all the CSV files and studied the relation between the different features

Flow Chart of Phase 4:



HCDR Data Description

The given HCDR dataset consists of the following CSV files; which have the information of all the tables and also about all the columns present in them. These are as follows:

1. **HomeCredit_columns_description.csv:**

This CSV file consists of the information of all the available columns along with all the available in the dataset. There are five columns named Table, Row, Description and Special. The column named “Table” consists of the name of the CSV file. The column named “Row” consists of the names of all the columns which are present in each CSV file. Each row contains the name of a particular column. The column named “Description” consists of the description of the particular column.

2. **POS_Cash_balance.csv:**

This file consists of 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.

3. **Application_Train.csv**:

This file consists of the training data. It has a total of 122 columns present in it. This data will be used for applying the model.

4. **Application_Test.csv:**

This file consists of the testing data. It has a total of 121 columns present in it. This data will be used for testing the trained model.

5. **Bureau.csv:**

All client's previous credits provided by other financial institutions were reported to the Credit Bureau (for clients who have a loan in our sample). The number of rows are equal to the number of credits the client had in the Credit Bureau before the application date.

6. **Bureau_Balance.csv:**

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.

7. **Credit_Card_Balance.csv:**

This file consists of the balance information of all the previous credit cards that the applicant has with Home Credit. This table has each row consisting of the history of previous credits for each month.

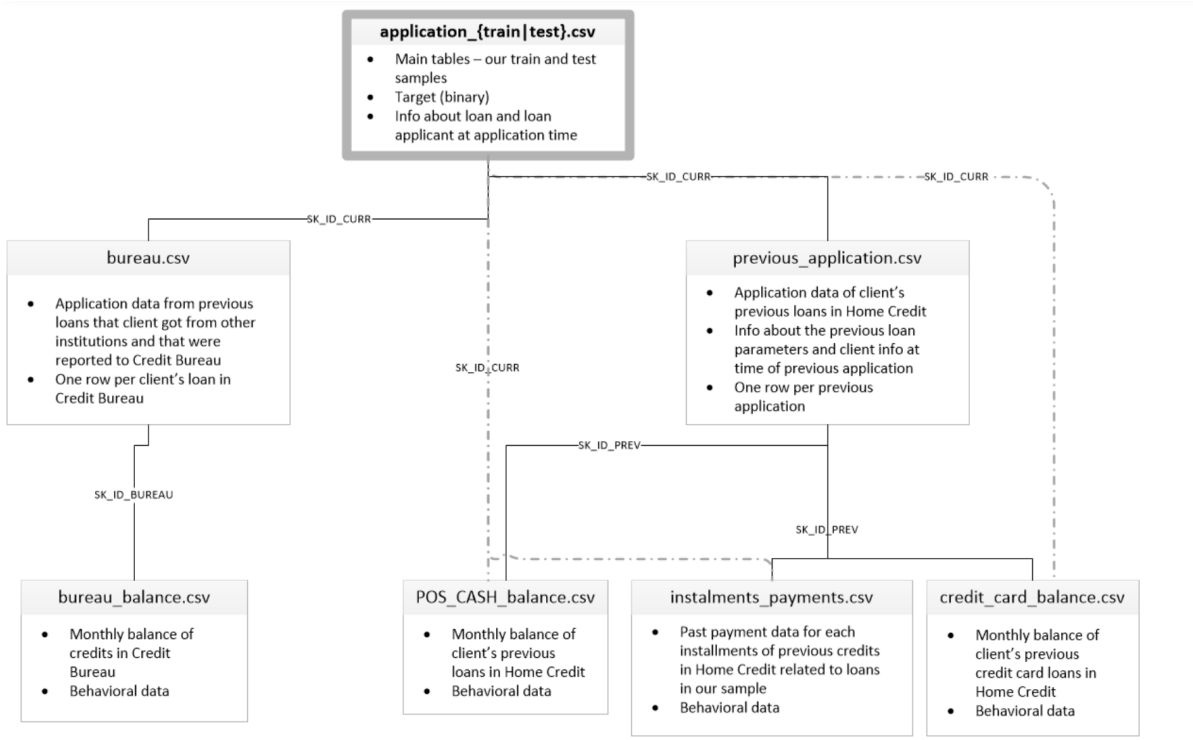
8. Previous_Application.csv:

This table consists of all the applications which have Home Credit loans. The table consists of one row for each previous application of the client.

9. Installments_payments.csv:

This table has the information of the previously disbursed credits in Home Credit related to the loans. Each row has two pieces of information i.e. a row for every payment that was made and also a row for each of the payments that was missed.

All the above mentioned are connected. The connection of these columns is as shown in the figure below.



The metrics of the data are as follows

```
application_train: shape is (307511, 122)
<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
None
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CRED
0	100002	1	Cash loans	M	N	Y	0	202500.0	40659
1	100003	0	Cash loans	F	N	N	0	270000.0	129350
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	13500
3	100006	0	Cash loans	F	N	Y	0	135000.0	31268
4	100007	0	Cash loans	M	N	Y	0	121500.0	51300

5 rows × 122 columns

```
application_test: shape is (48744, 121)
<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
None
```

	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

```
bureau: shape is (1716428, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_CURR              int64
 1   SK_ID_BUREAU            int64
 2   CREDIT_ACTIVE           object
 3   CREDIT_CURRENCY         object
 4   DAYS_CREDIT             int64
 5   CREDIT_DAY_OVERDUE      int64
 6   DAYS_CREDIT_ENDDATE     float64
 7   DAYS_ENDDATE_FACT       float64
 8   AMT_CREDIT_MAX_OVERDUE  float64
 9   CNT_CREDIT_PROLONG      int64
 10  AMT_CREDIT_SUM          float64
 11  AMT_CREDIT_SUM_DEBT     float64
 12  AMT_CREDIT_SUM_LIMIT    float64
 13  AMT_CREDIT_SUM_OVERDUE  float64
 14  CREDIT_TYPE             object
 15  DAYS_CREDIT_UPDATE      int64
 16  AMT_ANNUITY             float64
dtypes: float64(8), int64(6), object(3)
memory usage: 222.6+ MB
None
```

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DAYS_ENDDATE_FAC
0	215354	5714462	Closed	currency 1	-497	0	-153.0	-153.
1	215354	5714463	Active	currency 1	-208	0	1075.0	Nal
2	215354	5714464	Active	currency 1	-203	0	528.0	Nal
3	215354	5714465	Active	currency 1	-203	0	NaN	Nal
4	215354	5714466	Active	currency 1	-629	0	1197.0	Nal

```
bureau_balance: shape is (27299925, 3)
<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
None
```

	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

```
credit_card_balance: shape is (3840312, 23)
<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
None
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AMT_DRAWINGS_CURREN
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	0.
3	1389973	337855	-4	236572.110	225000	2250.0	2250.
4	1891521	126868	-1	453919.455	450000	0.0	11547.

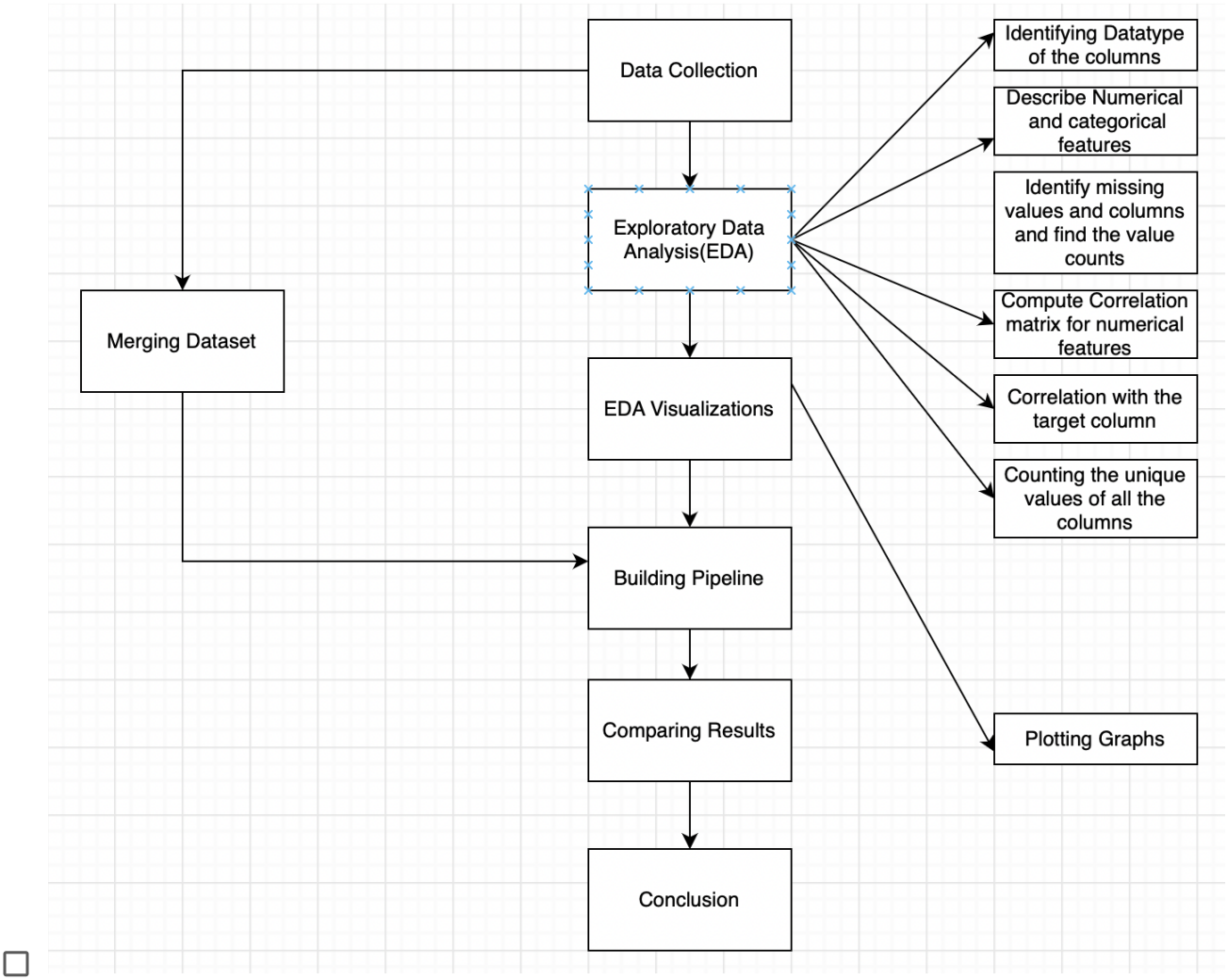
5 rows × 23 columns

```
installments_payments: shape is (13605401, 8)
<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
None
```

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT	A
0	1054186	161674	1.0	6	-1180.0	-1187.0	6948.360	
1	1330831	151639	0.0	34	-2156.0	-2156.0	1716.525	
2	2085231	193053	2.0	1	-63.0	-63.0	25425.000	
3	2452527	199697	1.0	3	-2418.0	-2426.0	24350.130	
4	2714724	167756	1.0	2	-1383.0	-1366.0	2165.040	

```
previous_application: shape is (1670214, 37)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype  
---  ------                       --------------    -----  
 0   SK_ID_PREV                   1670214 non-null  int64  
 1   SK_ID_CURR                   1670214 non-null  int64  
 2   NAME_CONTRACT_TYPE           1670214 non-null  object 
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object 
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64  
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object 
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64  
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object 
 16  NAME_CONTRACT_STATUS         1670214 non-null  object 
 17  DAYS_DECISION                1670214 non-null  int64  
 18  NAME_PAYMENT_TYPE            1670214 non-null  object 
 19  CODE_REJECT_REASON           1670214 non-null  object 
 20  NAME_TYPE_SUITE              849809 non-null   object 
 21  NAME_CLIENT_TYPE             1670214 non-null  object 
 22  NAME_GOODS_CATEGORY          1670214 non-null  object 
 23  NAME_PORTFOLIO               1670214 non-null  object 
 24  NAME_PRODUCT_TYPE            1670214 non-null  object 
 25  CHANNEL_TYPE                 1670214 non-null  object 
 26  SELLERPLACE_AREA             1670214 non-null  int64  
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object 
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object 
 30  PRODUCT_COMBINATION          1669868 non-null  object 
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
```

EDA Steps



Missing Values Analysis (EDA)

The missing values are calculated for all of the above files. The missing values are as follows:

1. **Missing values for Application Train File**: The below table shows the missing values of each of the different columns

Sr. No.	Column Name	Missing Value Count
1	COMMONAREA_MEDI	214865
2	COMMONAREA_AVG	214865
3	COMMONAREA_MODE	214865
4	NONLIVINGAPARTMENTS_MODE	213514
5	NONLIVINGAPARTMENTS_AVG	213514
6	NONLIVINGAPARTMENTS_MEDI	213514
7	FONDKAPREMONT_MODE	210295
8	LIVINGAPARTMENTS_MODE	210199
9	LIVINGAPARTMENTS_MEDI	210199
10	LIVINGAPARTMENTS_AVG	210199
11	FLOORSMIN_AVG	208642
12	FLOORSMIN_MODE	208642
13	FLOORSMIN_MEDI	208642
14	YEARS_BUILD_MEDI	204488
15	YEARS_BUILD_MODE	204488
16	YEARS_BUILD_AVG	204488
17	OWN_CAR_AGE	202929
18	LANDAREA_MEDI	182590
19	LANDAREA_MODE	182590
20	LANDAREA_AVG	182580

From the above table we can observe that the columns COMMONAREA_MEDI, COMMONAREA_AVG, COMMONAREA_MODE all have the missing value count of 214865 which is the highest and the columns LANDAREA_MEDI, LANDAREA_MODE and LANDAREA_AVG have the missing value count 182580 which is the lowest.

2.Missing values for POS_CASH_BALANCE: The below table shows the missing values of each of the different columns.

Sr. No.	Column Name	Missing Value Count
1	CNT_INSTALLMENT_FUTURE	26087
2	CNT_INSTALMENT	26071
3	SK_ID_PREV	0
4	MONTHS_BALANCE	0
5	NAME_CONTRACT_STATUS	0
6	SK_DPD	0
7	SK_DPD_DEF	0

From the above table, we can observe that the column CNT_INSTALLMENT has the highest number of missing values i.e 26087. We can also observe that four columns do not have any missing values i.e. that missing value count is zero.

3. Missing data for Bureau: The below table shows the missing values of each of the different columns.

Sr. No.	Column Name	Missing Value Count
1	AMT_ANNUITY	1226791
2	AMT_CREDIT_MAX_OVERDUE	1124488
3	DAYS_ENDDATE_FACT	633653
4	AMT_CREDIT_SUM_LIMIT	591780
5	AMT_CREDIT_SUM_DEBT	257669
6	DAYS_CREDIT_ENDDATE	105553
7	AMT_CREDIT_SUM	13
8	CREDIT_ACTIVE	0
9	CREDIT_CURRENCY	0
10	DAYS_CREDIT	0

The column AMT_ANNUITY has the highest missing values and the columns CREDIT_ACTIVE ,CREDIT_CURRENCY and DAYS_CEDIT have no missing values.

4. Missing values for Bureau Balance:

Sr. No.	Column Name	Missing Value Count
1	SK_ID_BUREAU	0
2	MONTHS_BALANCE	0
3	STATUS	0

The bureau balance table does not have any missing values.

4. Missing values for Credit Card Balance:

Sr. No.	Column Name	Missing Value Count
1	AMT_PAYMENT_CURRENT	1226791
2	AMT_DRAWINGS_ATM_CURRENT	1124488
3	CNT_DRAWINGS_POS_CURRENT	633653
4	AMT_DRAWINGS_OTHER_CURRENT	591780
5	AMT_DRAWINGS_POS_CURRENT	257669
6	CNT_DRAWINGS_OTHER_CURRENT	105553
7	CNT_DRAWINGS_ATM_CURRENT	13
8	CNT_INSTALLMENT_MATURE_CUM	0
9	AMT_INST_MIN_REGULARITY	0
10	SK_ID_PREV	0

The column AMT_PAYMENT_CURRENT has the highest number of missing values and the columns CNT_INSTALLMENT_MATURE_CUM, AMT_INST_MIN_REGULARITY, SK_ID_PREV does not have any missing values.

5. Missing data for installments payments:

Sr. No.	Column Name	Missing Value Count
1	DAYS_ENTRY_PAYMENT	2905
2	AMT_PAYMENT	2905
3	SK_ID_PREV	0
4	SK_ID_CURR	0
5	NUM_INSTALLMENT_VERSION	0
6	NUM_INSTALLMENT_NUMBER	0
7	DAYS_INSTALMENT	0
8	AMT_INSTALMENT	0

The columns DAYS_ENTRY_PAYMENT and AMT_PAYMENT have the highest number of missing counts and the remaining columns do not have any missing values.

Unique value Analysis

1. **Unique value analysis for POS_CASH_BALANCE:**

```
SK_ID_PREV                936325
SK_ID_CURR                337252
MONTHS_BALANCE                96
CNT_INSTALMENT                73
CNT_INSTALMENT_FUTURE         79
NAME_CONTRACT_STATUS           9
SK_DPD                      3400
SK_DPD_DEF                  2307
dtype: int64
```

2. **Unique value analysis for CREDIT_CARD_BALANCE:**

```
30]: print(datasets["credit_card_balance"]
```

```
SK_ID_PREV                    104307
SK_ID_CURR                    103558
MONTHS_BALANCE                    96
AMT_BALANCE                  1347904
AMT_CREDIT_LIMIT_ACTUAL          181
AMT_DRAWINGS_ATM_CURRENT        2267
AMT_DRAWINGS_CURRENT          187005
AMT_DRAWINGS_OTHER_CURRENT      1832
AMT_DRAWINGS_POS_CURRENT      168748
AMT_INST_MIN_REGULARITY       312266
AMT_PAYMENT_CURRENT           163209
AMT_PAYMENT_TOTAL_CURRENT     182957
AMT_RECEIVABLE_PRINCIPAL     1195839
AMT_RECIVABLE                1338878
AMT_TOTAL_RECEIVABLE         1339008
CNT_DRAWINGS_ATM_CURRENT          44
CNT_DRAWINGS_CURRENT             129
CNT_DRAWINGS_OTHER_CURRENT        11
CNT_DRAWINGS_POS_CURRENT         133
CNT_INSTALMENT_MATURE_CUM        121
NAME_CONTRACT_STATUS               7
SK_DPD                           917
SK_DPD_DEF                       378
dtype: int64
```

3. **Unique value analysis for PREVIOUS_APPLICATION:**

[35]:
```
print(datasets["previous_application"].nunique())
```

```
SK_ID_PREV                     1670214
SK_ID_CURR                      338857
NAME_CONTRACT_TYPE                   4
AMT_ANNUITY                     357959
AMT_APPLICATION                  93885
AMT_CREDIT                       86803
AMT_DOWN_PAYMENT                 29278
AMT_GOODS_PRICE                  93885
WEEKDAY_APPR_PROCESS_START           7
HOUR_APPR_PROCESS_START             24
FLAG_LAST_APPL_PER_CONTRACT          2
NFLAG_LAST_APPL_IN_DAY               2
RATE_DOWN_PAYMENT               207033
RATE_INTEREST_PRIMARY              148
RATE_INTEREST_PRIVILEGED            25
NAME_CASH_LOAN_PURPOSE              25
NAME_CONTRACT_STATUS                 4
DAYS_DECISION                     2922
NAME_PAYMENT_TYPE                    4
CODE_REJECT_REASON                   9
NAME_TYPE_SUITE                      7
NAME_CLIENT_TYPE                     4
NAME_GOODS_CATEGORY                 28
NAME_PORTFOLIO                       5
NAME_PRODUCT_TYPE                    3
CHANNEL_TYPE                         8
SELLERPLACE_AREA                  2097
NAME_SELLER_INDUSTRY                11
CNT_PAYMENT                         49
NAME_YIELD_GROUP                     5
PRODUCT_COMBINATION                 17
DAYS_FIRST_DRAWING                2838
DAYS_FIRST_DUE                    2892
DAYS_LAST_DUE_1ST_VERSION         4605
DAYS_LAST_DUE                     2873
DAYS_TERMINATION                  2830
NFLAG_INSURED_ON_APPROVAL            2
dtype: int64
```

4. **Unique value analysis for APPLICATION_TRAIN:**

```
print(datasets["application_train"].nunique())
```

```
SK_ID_CURR                    307511
TARGET                             2
NAME_CONTRACT_TYPE                 2
CODE_GENDER                        3
FLAG_OWN_CAR                       2
                                 ...
AMT_REQ_CREDIT_BUREAU_DAY          9
AMT_REQ_CREDIT_BUREAU_WEEK         9
AMT_REQ_CREDIT_BUREAU_MON         24
AMT_REQ_CREDIT_BUREAU_QRT         11
AMT_REQ_CREDIT_BUREAU_YEAR        25
Length: 122, dtype: int64
```

5. **Unique value analysis for BUREAU:**

```
print(datasets["bureau"].nunique())
```

```
SK_ID_CURR                 305811
SK_ID_BUREAU              1716428
CREDIT_ACTIVE                   4
CREDIT_CURRENCY                 4
DAYS_CREDIT                  2923
CREDIT_DAY_OVERDUE            942
DAYS_CREDIT_ENDDATE         14096
DAYS_ENDDATE_FACT            2917
AMT_CREDIT_MAX_OVERDUE      68251
CNT_CREDIT_PROLONG             10
AMT_CREDIT_SUM             236708
AMT_CREDIT_SUM_DEBT        226537
AMT_CREDIT_SUM_LIMIT        51726
AMT_CREDIT_SUM_OVERDUE       1616
CREDIT_TYPE                    15
DAYS_CREDIT_UPDATE           2982
AMT_ANNUITY                 40321
dtype: int64
```

+ Code + Markdown

6. **Unique value analysis for Instalment:**

```
SK_ID_PREV                997752
SK_ID_CURR                339587
NUM_INSTALMENT_VERSION        65
NUM_INSTALMENT_NUMBER        277
DAYS_INSTALMENT             2922
DAYS_ENTRY_PAYMENT          3039
AMT_INSTALMENT            902539
AMT_PAYMENT               944235
dtype: int64
```

7. **Unique value analysis for Bureau Balance:**

[36]:
```
print(datasets["bureau_balance"].nunique())
```

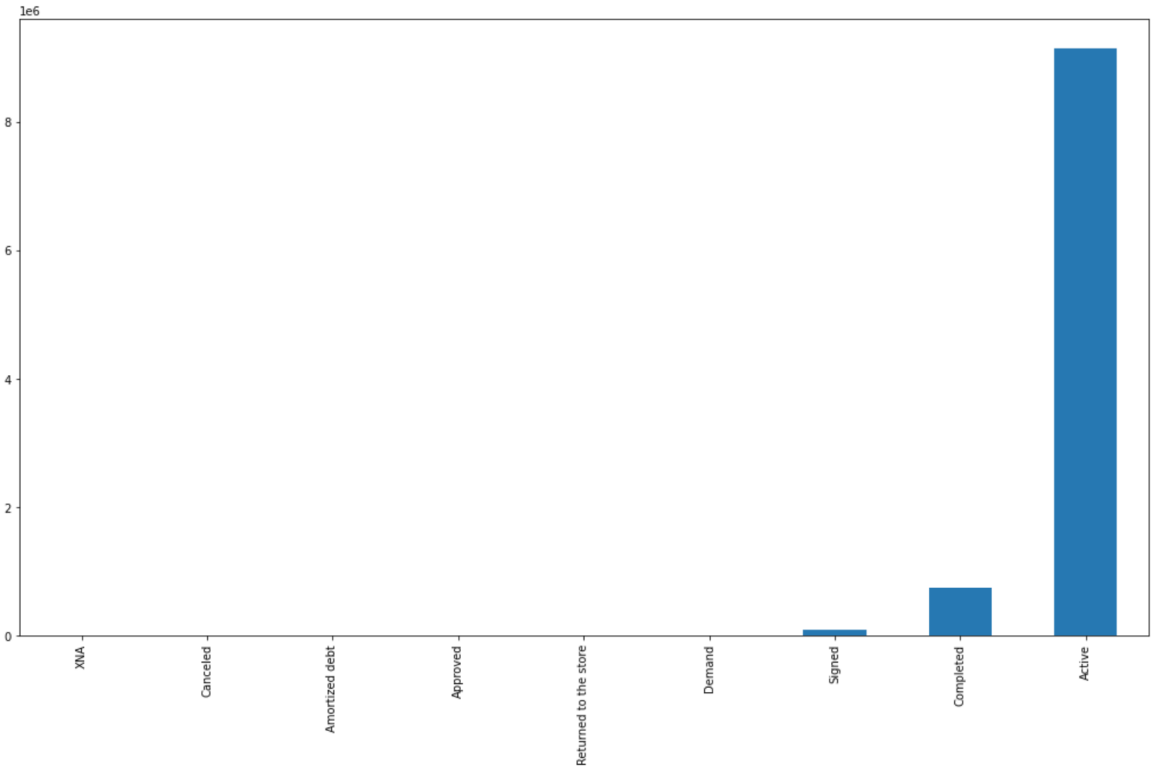
```
SK_ID_BUREAU      817395
MONTHS_BALANCE        97
STATUS                 8
dtype: int64
```

Visual EDA Analysis:

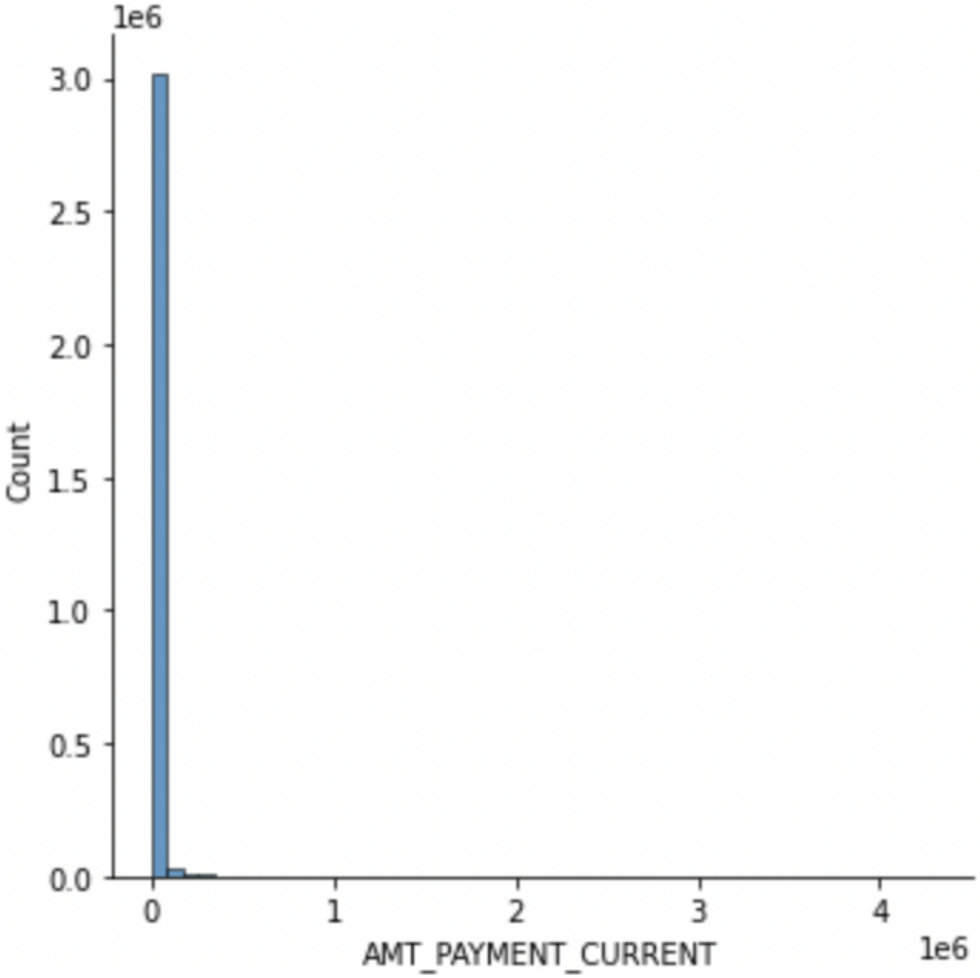
POS_CASH_BALANCE:

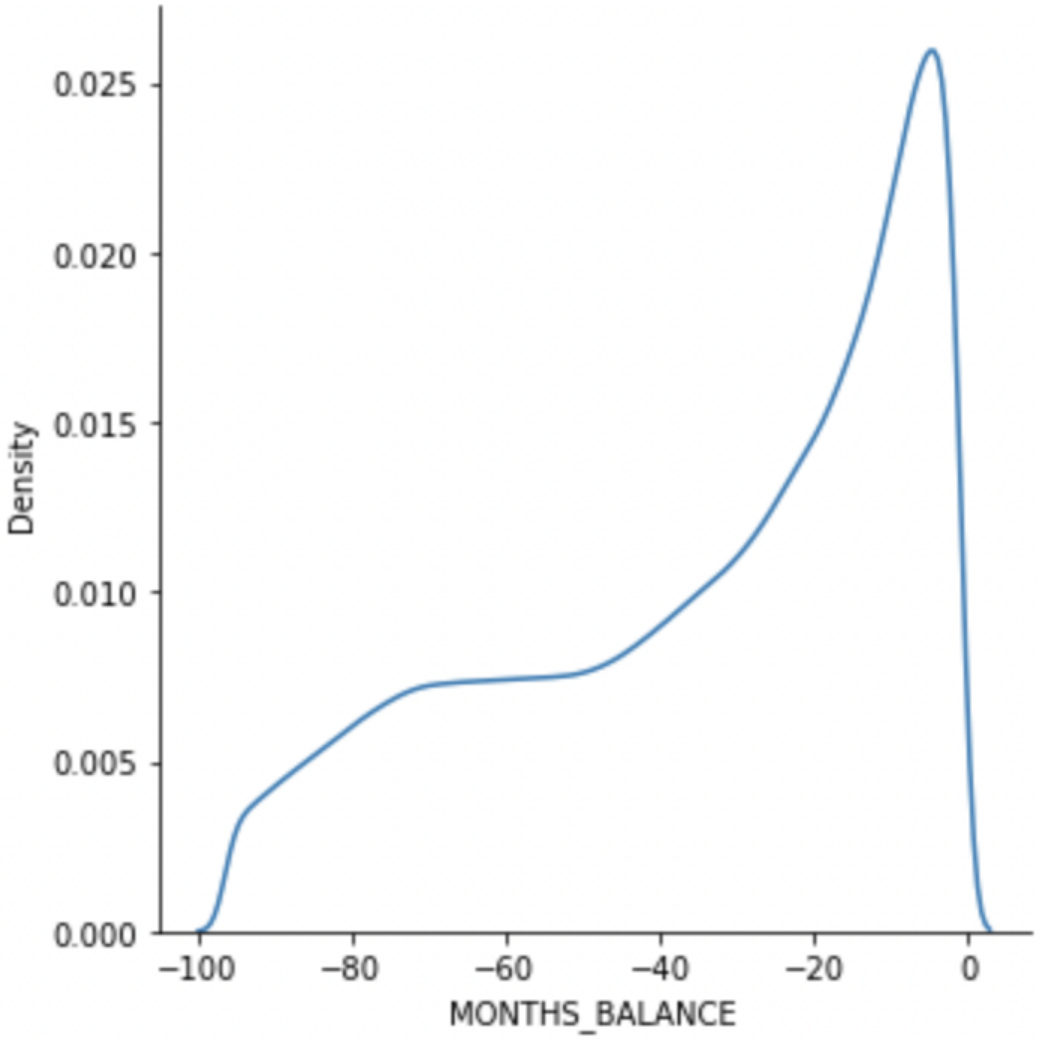
The figure below shows the distribution of value counts of the categorical column NAME_CONTRACT_STATUS.

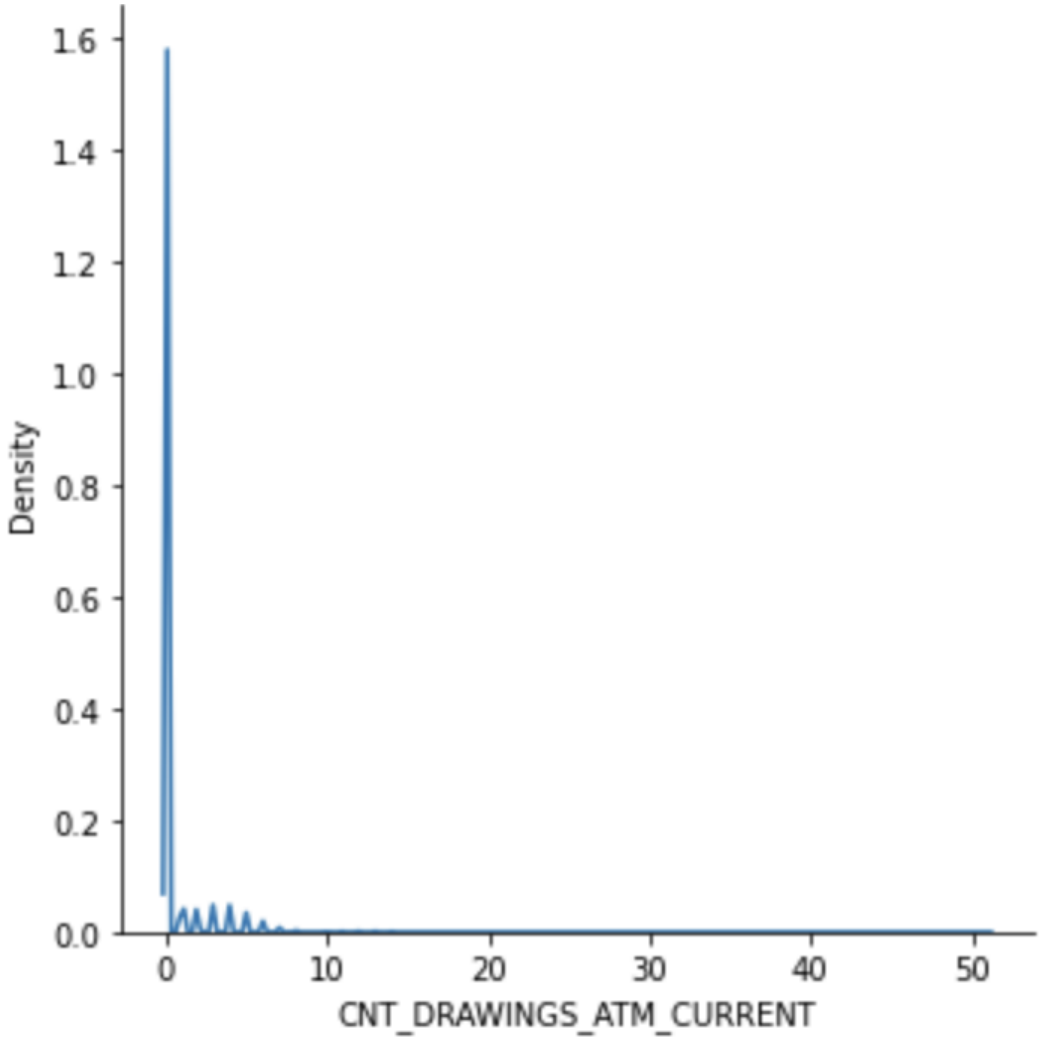
[38... <AxesSubplot:>

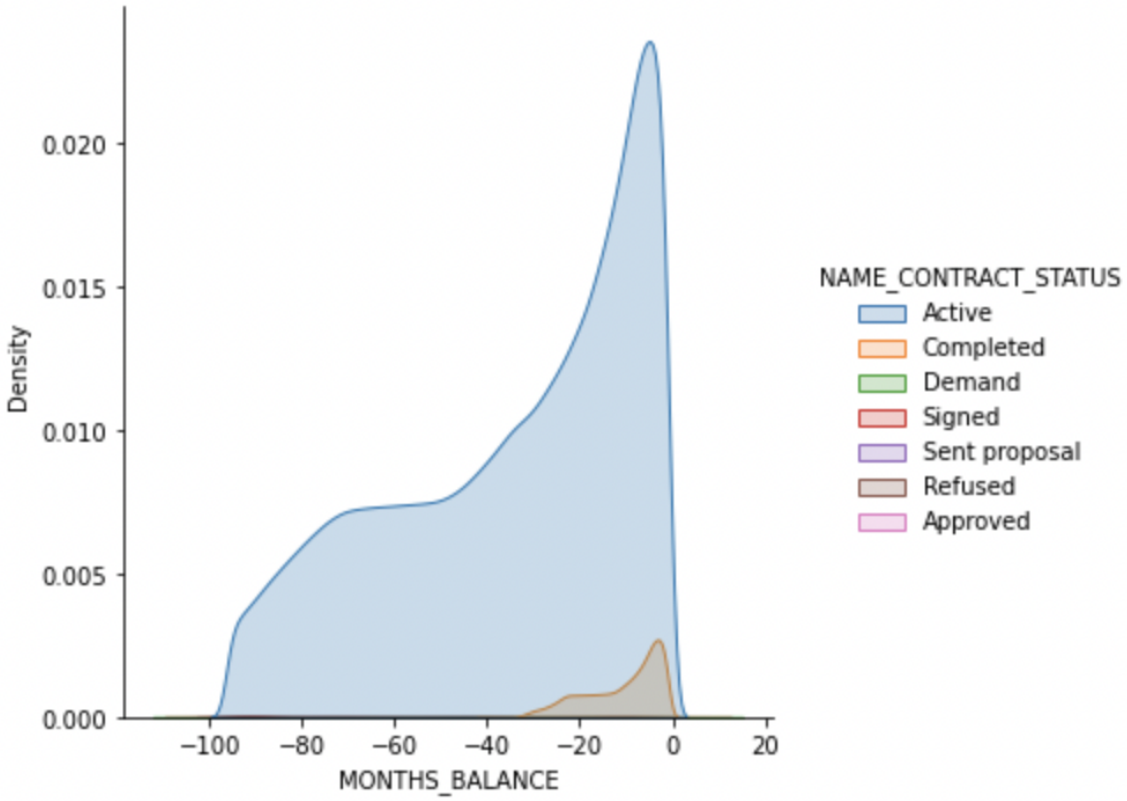


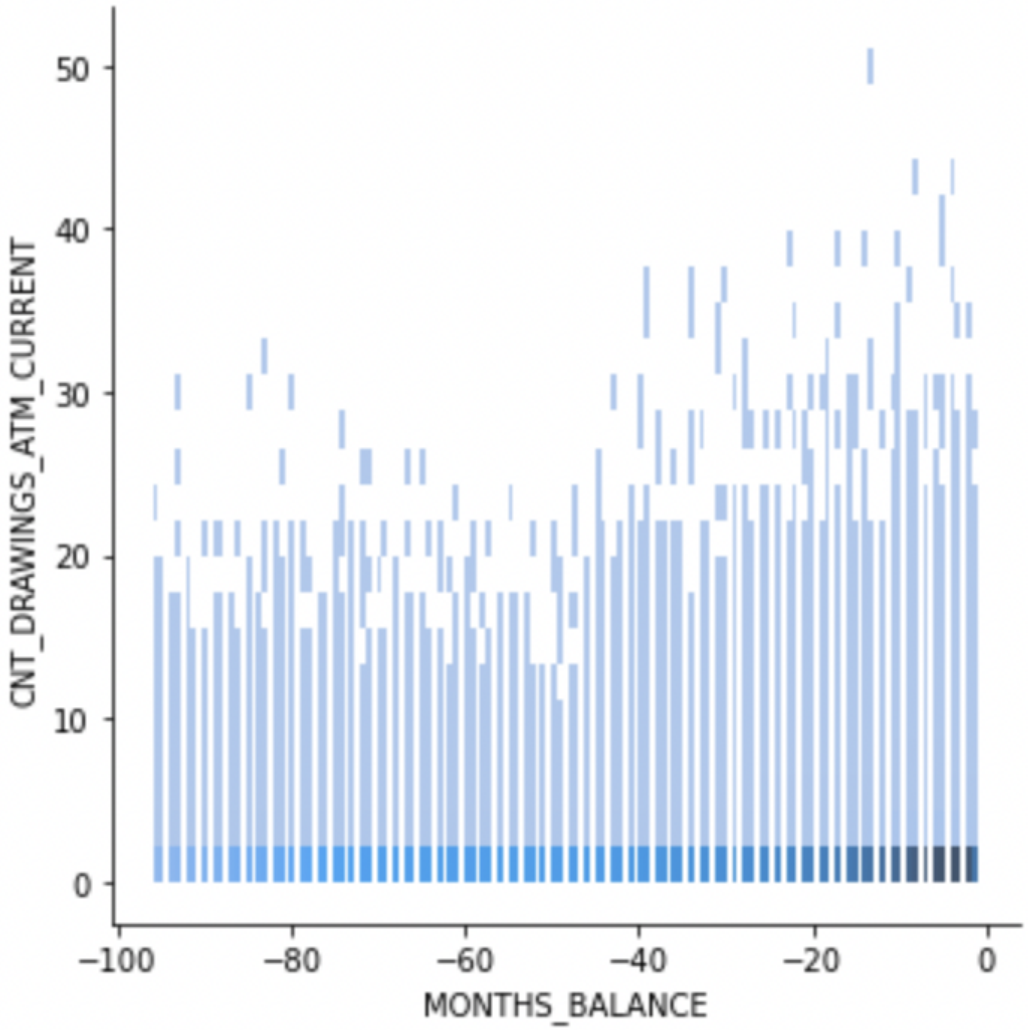
Credit Card Balance:

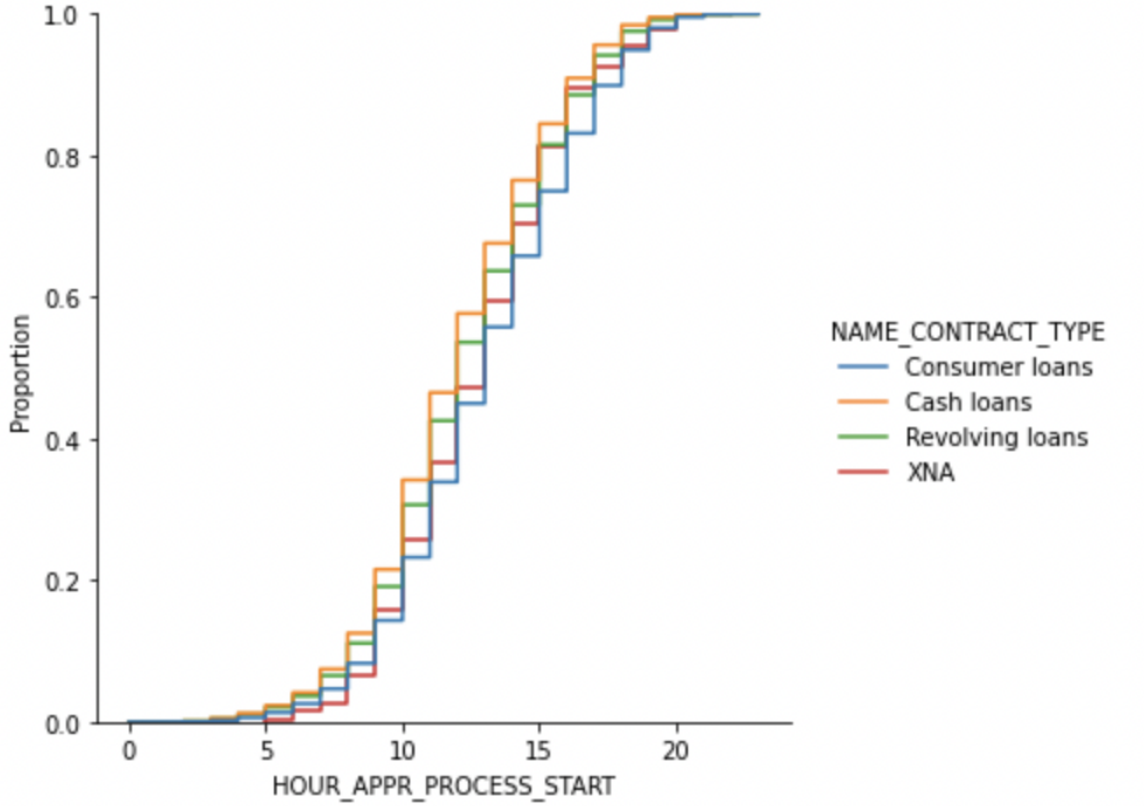

1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Count
0
1
2
3
4
1e6
AMT_PAYMENT_CURRENT

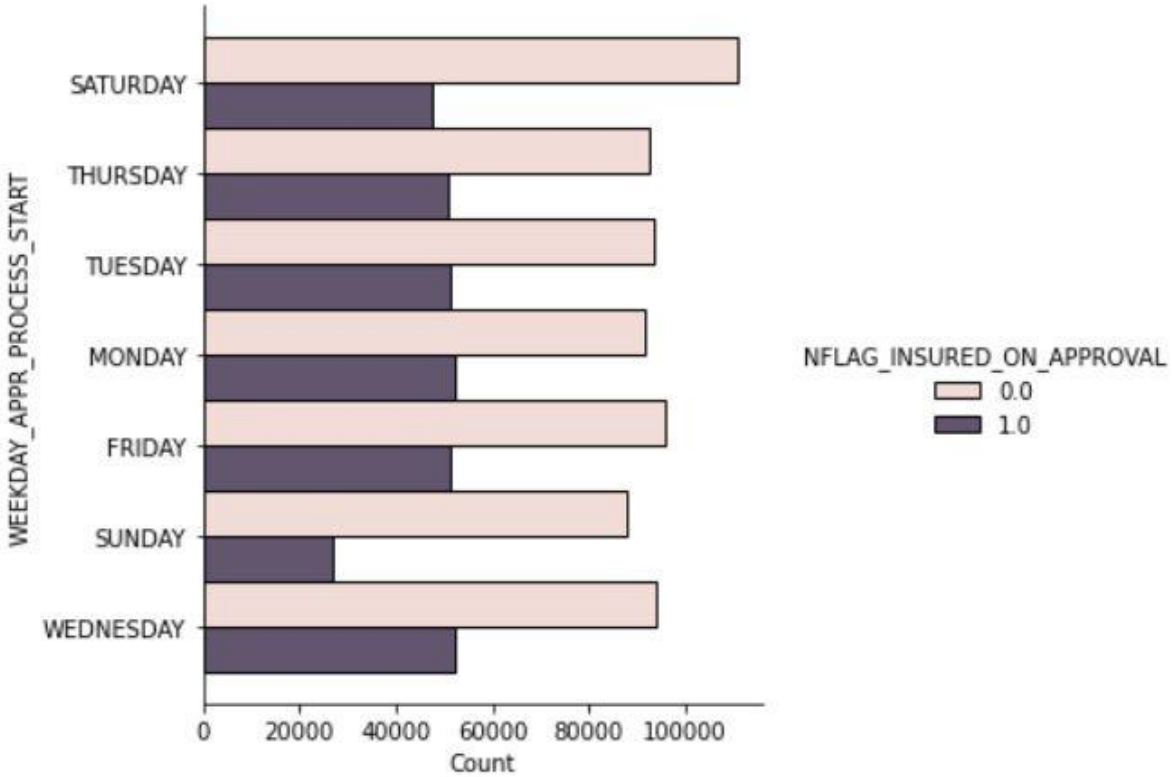

0.025
0.020
0.015
0.010
0.005
0.000
Density
−100
−80
−60
−40
−20
0
MONTHS_BALANCE


1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Density
0
10
20
30
40
50
CNT_DRAWINGS_ATM_CURRENT


Density
0.020
0.015
0.010
0.005
0.000
−100
−80
−60
−40
−20
0
20
MONTHS_BALANCE
NAME_CONTRACT_STATUS
Active
Completed
Demand
Signed
Sent proposal
Refused
Approved


CNT_DRAWINGS_ATM_CURRENT
50
40
30
20
10
0
−100
−80
−60
−40
−20
0
MONTHS_BALANCE


1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0
5
10
15
20
HOUR_APPR_PROCESS_START
NAME_CONTRACT_TYPE
Consumer loans
Cash loans
Revolving loans
XNA


SATURDAY
THURSDAY
TUESDAY
MONDAY
FRIDAY
SUNDAY
WEDNESDAY
WEEKDAY_APPR_PROCESS_START
0
20000
40000
60000
80000
100000
Count
NFLAG_INSURED_ON_APPROVAL
0.0
1.0

4.2. Missing data for application train

[31]:
```
percent = (datasets["application_train"].isnull().sum()/datasets["application_train"].isnull().count()*100).sort_values(asc
sum_missing = datasets["application_train"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Train Missing Count"])
missing_application_train_data.head(20)
```

[31]:

	Percent	Train Missing Count
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865
COMMONAREA_MODE	69.87	214865
NONLIVINGAPARTMENTS_MODE	69.43	213514
NONLIVINGAPARTMENTS_AVG	69.43	213514
NONLIVINGAPARTMENTS_MEDI	69.43	213514
FONDKAPREMONT_MODE	68.39	210295
LIVINGAPARTMENTS_MODE	68.35	210199
LIVINGAPARTMENTS_AVG	68.35	210199
LIVINGAPARTMENTS_MEDI	68.35	210199
FLOORSMIN_AVG	67.85	208642
FLOORSMIN_MODE	67.85	208642
FLOORSMIN_MEDI	67.85	208642
YEARS_BUILD_MEDI	66.50	204488
YEARS_BUILD_MODE	66.50	204488
YEARS_BUILD_AVG	66.50	204488
OWN_CAR_AGE	65.99	202929
LANDAREA_MEDI	59.38	182590
LANDAREA_MODE	59.38	182590
LANDAREA_AVG	59.38	182590

[32]:
```
percent = (datasets["application_test"].isnull().sum()/datasets["application_test"].isnull().count()*100).sort_values(ascen
sum_missing = datasets["application_test"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Count"])
missing_application_train_data.head(20)
```

[32]:

	Percent	Test Missing Count
COMMONAREA_AVG	68.72	33495
COMMONAREA_MODE	68.72	33495
COMMONAREA_MEDI	68.72	33495
NONLIVINGAPARTMENTS_AVG	68.41	33347
NONLIVINGAPARTMENTS_MODE	68.41	33347
NONLIVINGAPARTMENTS_MEDI	68.41	33347
FONDKAPREMONT_MODE	67.28	32797
LIVINGAPARTMENTS_AVG	67.25	32780
LIVINGAPARTMENTS_MODE	67.25	32780
LIVINGAPARTMENTS_MEDI	67.25	32780
FLOORSMIN_MEDI	66.61	32466
FLOORSMIN_AVG	66.61	32466
FLOORSMIN_MODE	66.61	32466
OWN_CAR_AGE	66.29	32312
YEARS_BUILD_AVG	65.28	31818
YEARS_BUILD_MEDI	65.28	31818
YEARS_BUILD_MODE	65.28	31818
LANDAREA_MEDI	57.96	28254
LANDAREA_AVG	57.96	28254
LANDAREA_MODE	57.96	28254

4.4. Correlation with the target column

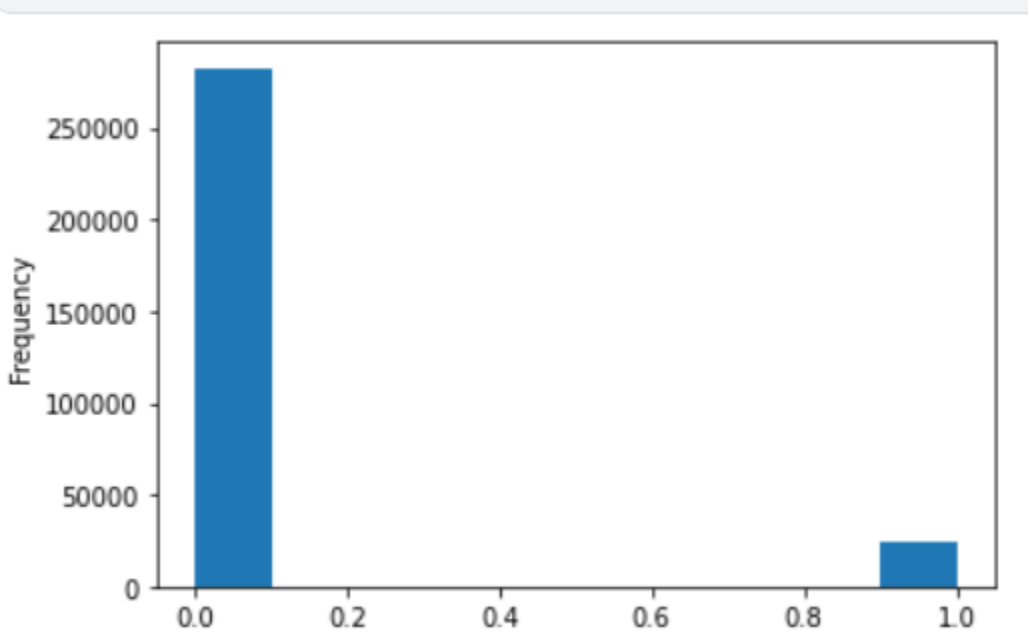
[34]:
```
correlations = datasets["application_train"].corr()['TARGET'].sort_values()
print('Most Positive Correlations:\n', correlations.tail(10))
print('\nMost Negative Correlations:\n', correlations.head(10))
```

```
Most Positive Correlations:
 FLAG_DOCUMENT_3               0.044346
REG_CITY_NOT_LIVE_CITY         0.044395
FLAG_EMP_PHONE                 0.045982
REG_CITY_NOT_WORK_CITY         0.050994
DAYS_ID_PUBLISH                0.051457
DAYS_LAST_PHONE_CHANGE         0.055218
REGION_RATING_CLIENT           0.058899
REGION_RATING_CLIENT_W_CITY    0.060893
DAYS_BIRTH                     0.078239
TARGET                         1.000000
Name: TARGET, dtype: float64

Most Negative Correlations:
 EXT_SOURCE_3                 -0.178919
EXT_SOURCE_2                  -0.160472
EXT_SOURCE_1                  -0.155317
DAYS_EMPLOYED                 -0.044932
FLOORSMAX_AVG                 -0.044003
FLOORSMAX_MEDI                -0.043768
FLOORSMAX_MODE                -0.043226
AMT_GOODS_PRICE               -0.039645
REGION_POPULATION_RELATIVE    -0.037227
ELEVATORS_AVG                 -0.034199
Name: TARGET, dtype: float64
```

Application Train

1. Distribution of target column



2. Top 20 positive correlations with the target column

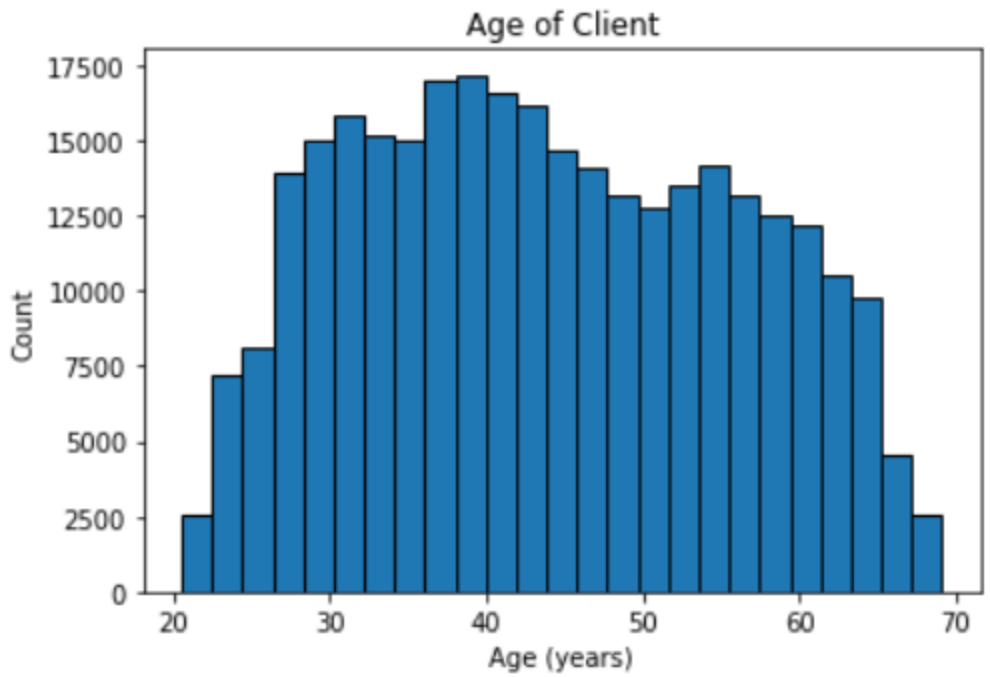
```
Most Positive Correlations:
 OBS_30_CNT_SOCIAL_CIRCLE       0.009131
CNT_FAM_MEMBERS                 0.009308
CNT_CHILDREN                    0.019187
AMT_REQ_CREDIT_BUREAU_YEAR      0.019930
FLAG_WORK_PHONE                 0.028524
DEF_60_CNT_SOCIAL_CIRCLE        0.031276
DEF_30_CNT_SOCIAL_CIRCLE        0.032248
LIVE_CITY_NOT_WORK_CITY         0.032518
OWN_CAR_AGE                     0.037612
DAYS_REGISTRATION               0.041975
FLAG_DOCUMENT_3                 0.044346
REG_CITY_NOT_LIVE_CITY          0.044395
FLAG_EMP_PHONE                  0.045982
REG_CITY_NOT_WORK_CITY          0.050994
DAYS_ID_PUBLISH                 0.051457
DAYS_LAST_PHONE_CHANGE          0.055218
REGION_RATING_CLIENT            0.058899
REGION_RATING_CLIENT_W_CITY     0.060893
DAYS_BIRTH                      0.078239
TARGET                          1.000000
Name: TARGET, dtype: float64
```

Days_birth(age) and region rating client city has the top correlation with the target

3. Most negative correlations with the target column

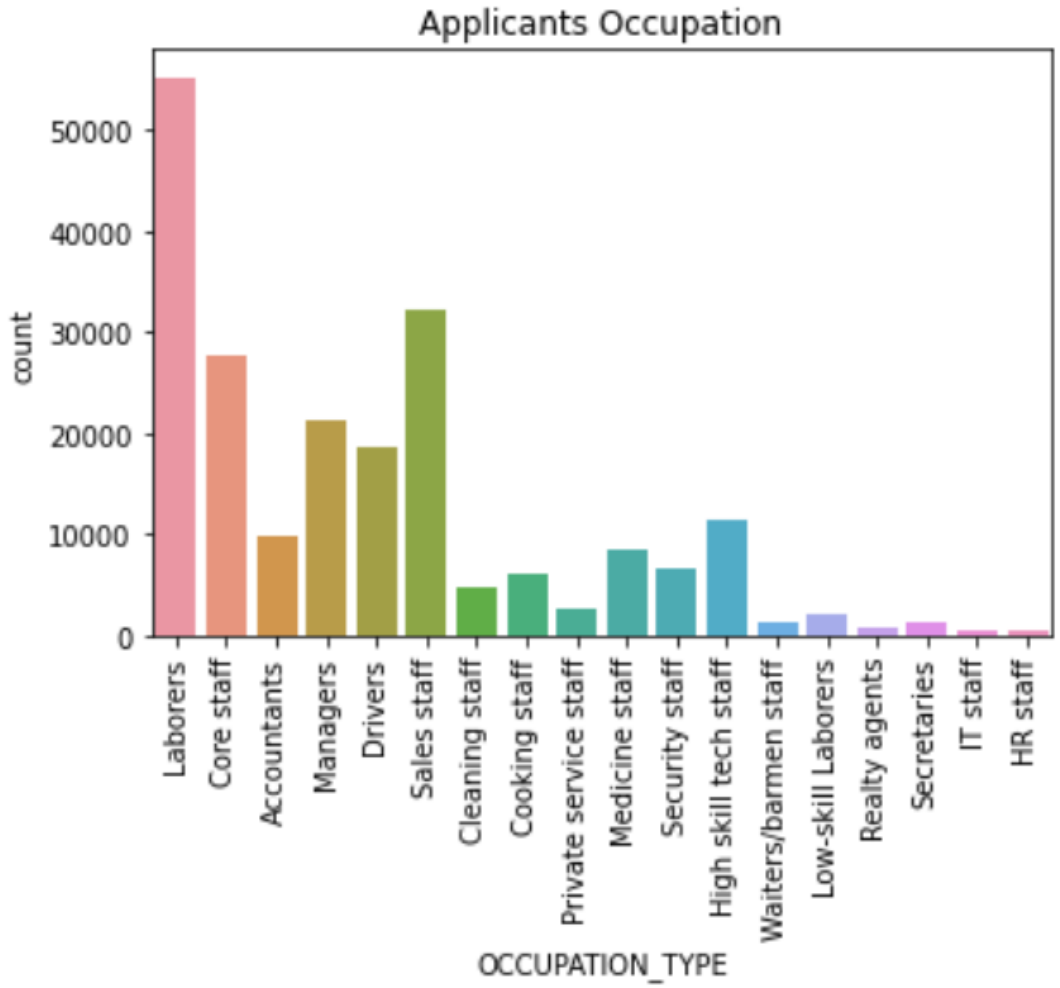
```
Most Negative Correlations:
 EXT_SOURCE_3                 -0.178919
EXT_SOURCE_2                  -0.160472
EXT_SOURCE_1                  -0.155317
DAYS_EMPLOYED                 -0.044932
FLOORSMAX_AVG                 -0.044003
FLOORSMAX_MEDI                -0.043768
FLOORSMAX_MODE                -0.043226
AMT_GOODS_PRICE               -0.039645
REGION_POPULATION_RELATIVE    -0.037227
ELEVATORS_AVG                 -0.034199
ELEVATORS_MEDI                -0.033863
FLOORSMIN_AVG                 -0.033614
FLOORSMIN_MEDI                -0.033394
LIVINGAREA_AVG                -0.032997
LIVINGAREA_MEDI               -0.032739
FLOORSMIN_MODE                -0.032698
TOTALAREA_MODE                -0.032596
ELEVATORS_MODE                -0.032131
LIVINGAREA_MODE               -0.030685
AMT_CREDIT                    -0.030369
APARTMENTS_AVG                -0.029498
APARTMENTS_MEDI               -0.029184
FLAG_DOCUMENT_6               -0.028602
APARTMENTS_MODE               -0.027284
LIVINGAPARTMENTS_AVG          -0.025031
LIVINGAPARTMENTS_MEDI         -0.024621
HOUR_APPR_PROCESS_START       -0.024166
FLAG_PHONE                    -0.023806
LIVINGAPARTMENTS_MODE         -0.023393
BASEMENTAREA_AVG              -0.022746
YEARS_BUILD_MEDI              -0.022326
YEARS_BUILD_AVG               -0.022149
BASEMENTAREA_MEDI             -0.022081
YEARS_BUILD_MODE              -0.022068
BASEMENTAREA_MODE             -0.019952
ENTRANCES_AVG                 -0.019172
ENTRANCES_MEDI                -0.019025
COMMONAREA_MEDI               -0.018573
COMMONAREA_AVG                -0.018550
ENTRANCES_MODE                -0.017387
Name: TARGET, dtype: float64
```

4. Age distribution in the data



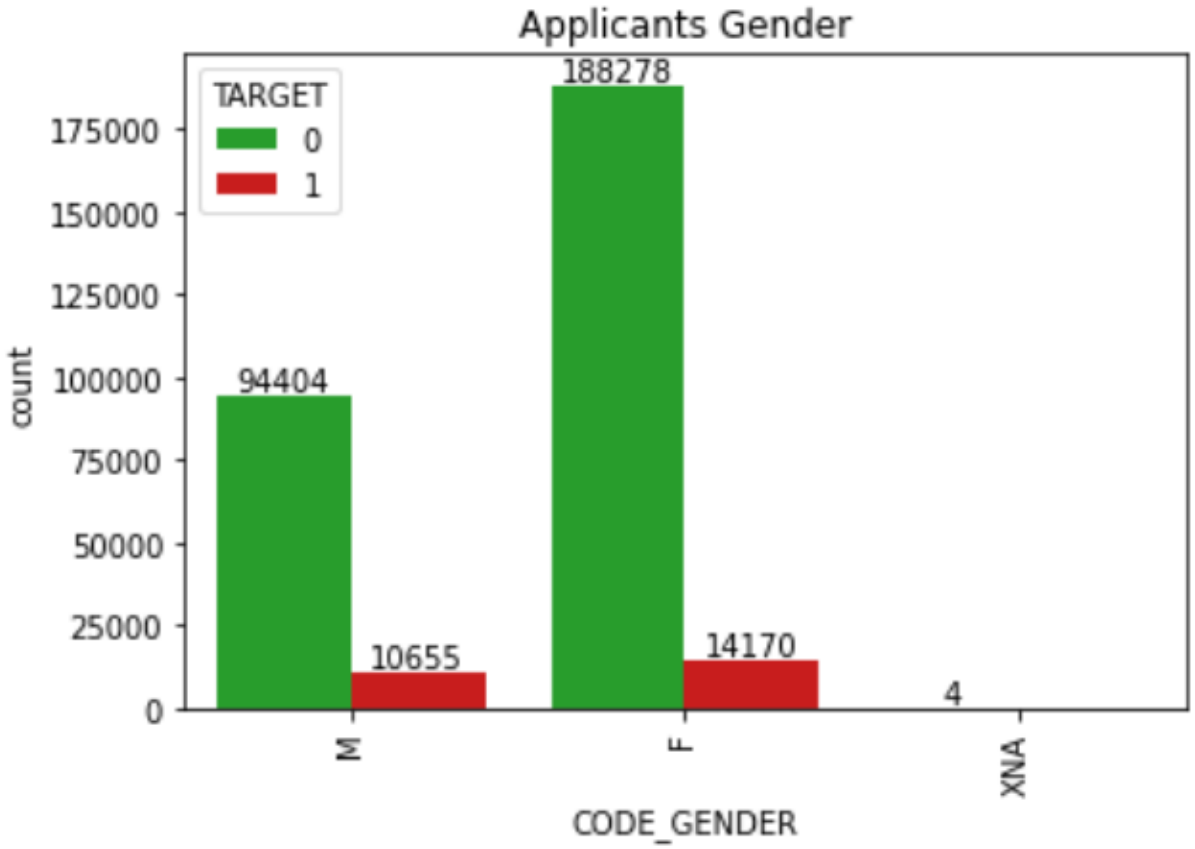
Most of the applicants are between 30 to 50 years old.

5. Applicant's occupation analysis



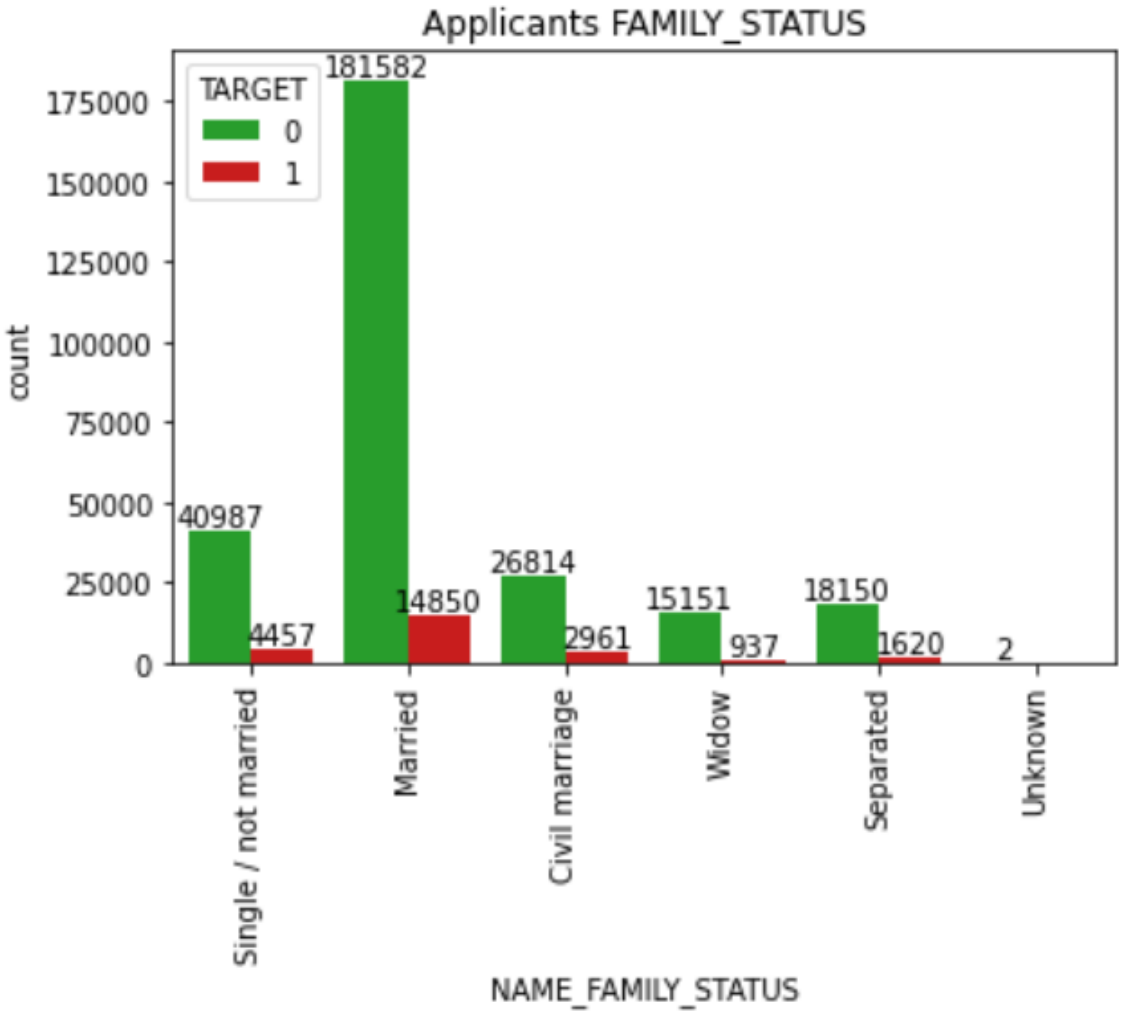
Top Applicants who applied for loan: Laborers - Approx 55 K, Sales Staff - Approx 32 K, Core staff - Approx 28 K.

6. Applicant's gender relation with target column



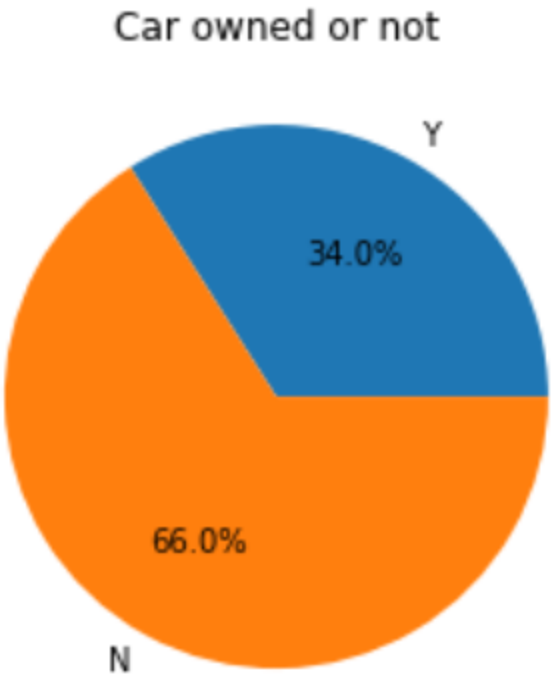
According to applicant data, women have nearly double the number of loan applications than males. Approximately **202,448** loan applications have been submitted overall by women, compared to **105,059 by men**. However, compared to female applicants (approximately 7%), a higher proportion of men (about 10% of the total) had issues repaying the loan or making installments on time.

7. Family status analysis compared to target variable

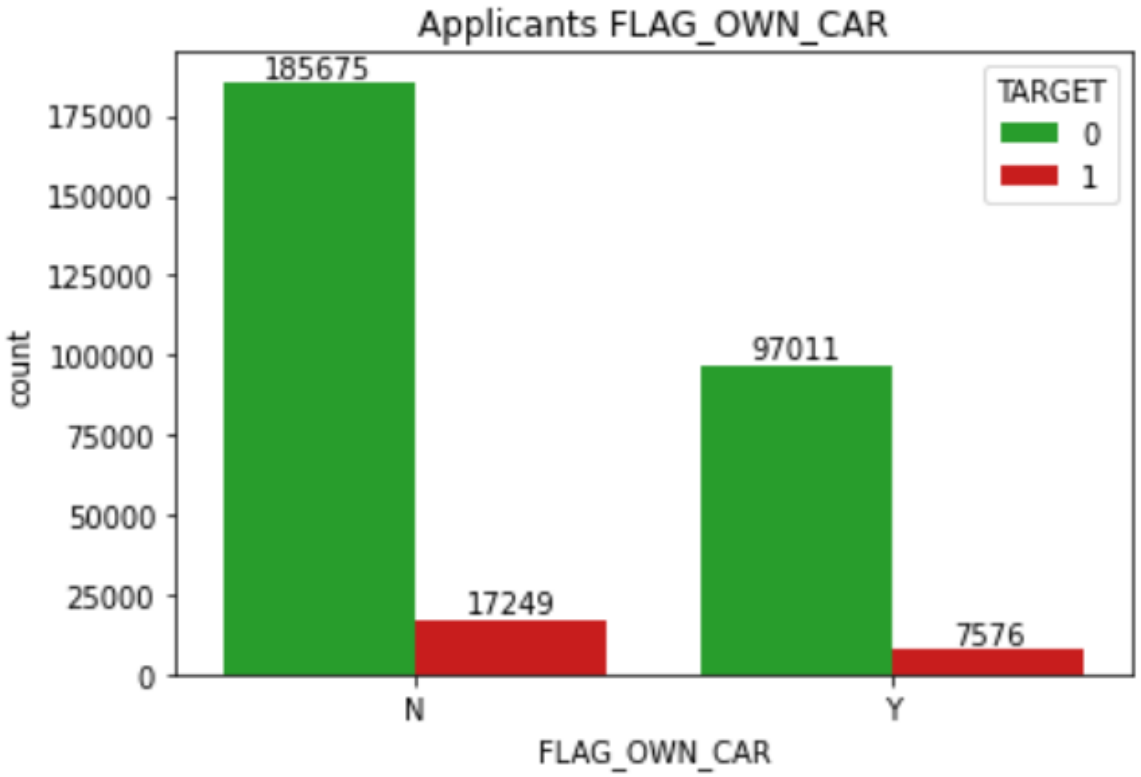


About 196K persons who are married have applied for loans. However, the biggest percentage of loan issues and hurdles (about 10%) are experienced by those who are legally married.

8. Does the applicant own a car?

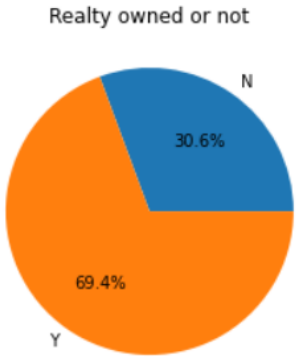


About 66% of the applicant doesn’t own a car.
Car own analysis against target variable:

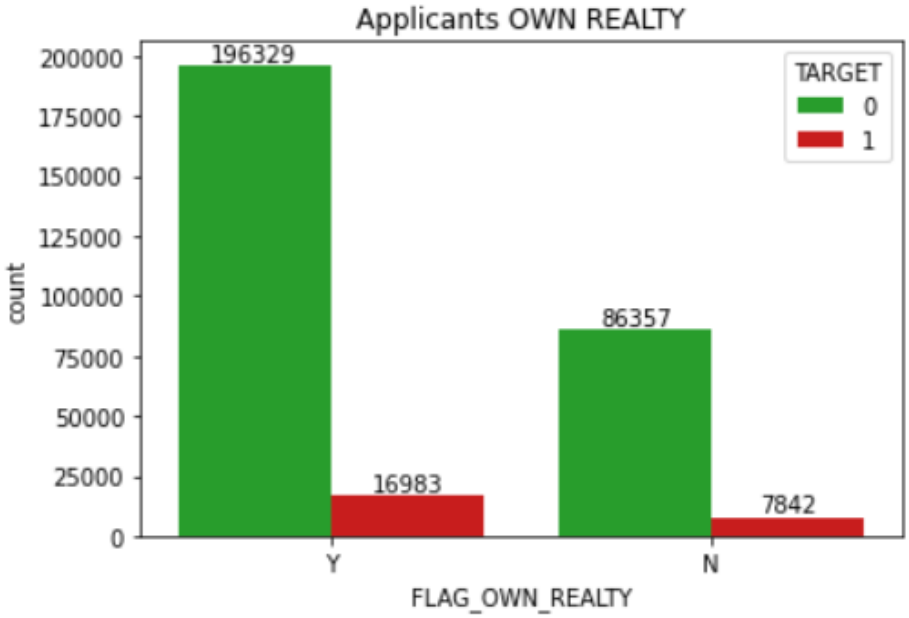


Only "34%" of applicants who have asked for loans in previous years have a car. However, applicants who did not possess a car had a higher percentage of applicants who were having trouble making ends meet.

9. Does the applicant owns any realty?

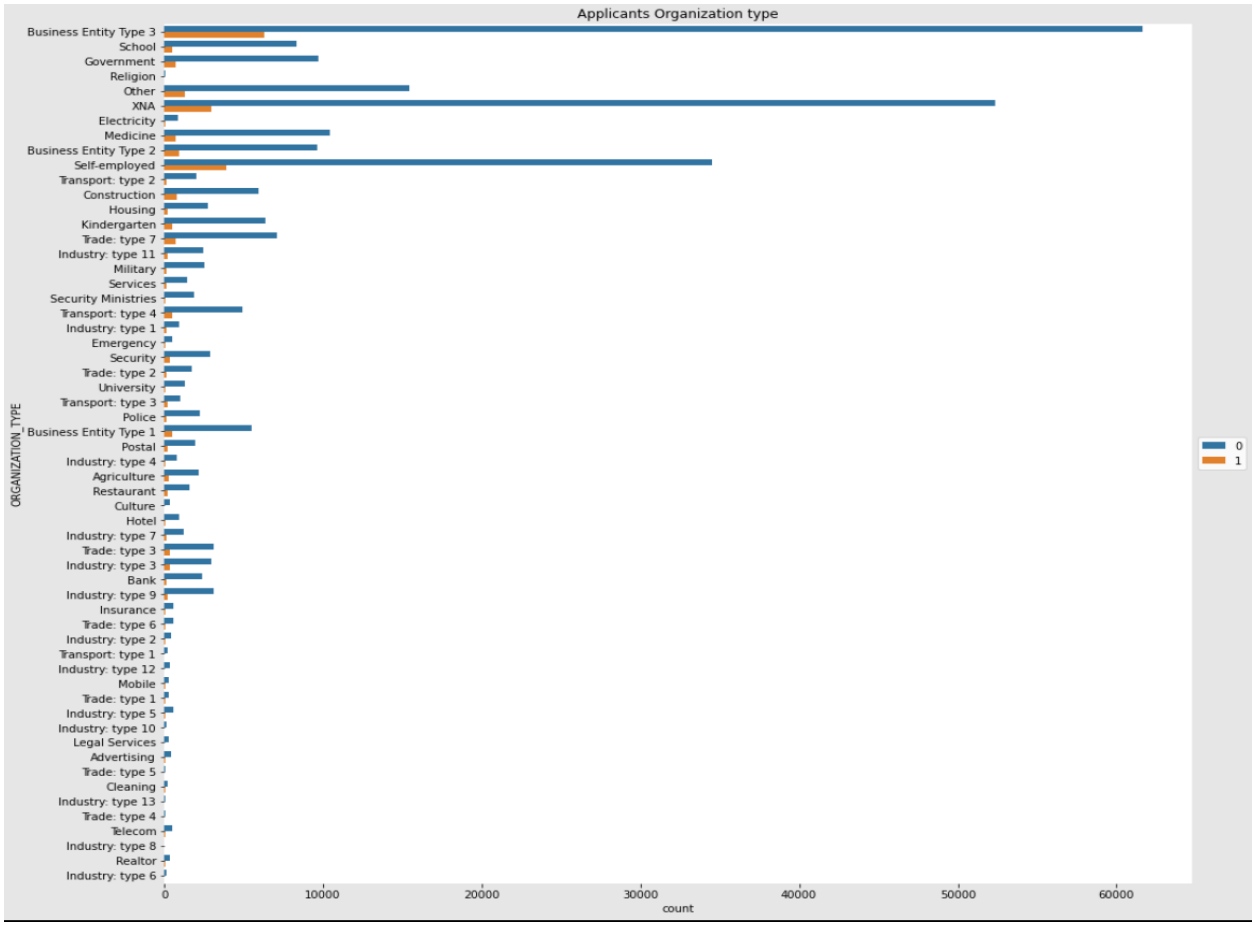


Almost 70% of the applicants own a realty. Check the distribution of it with the target as follow:



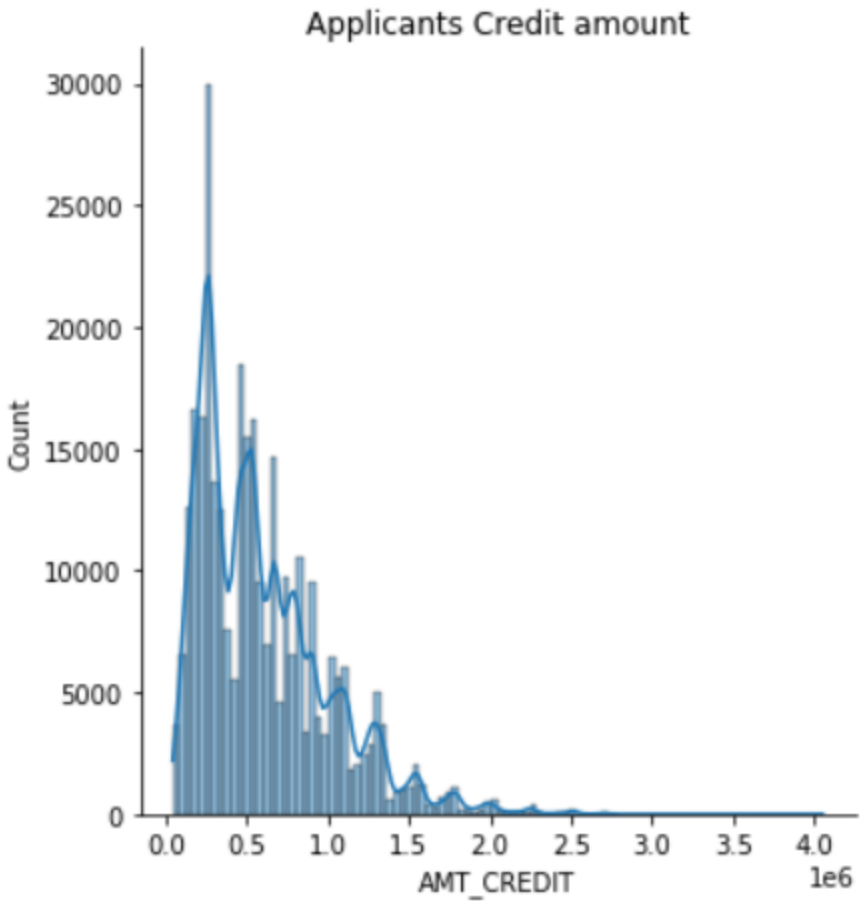
Only "70%" of candidates who have previously applied for the loan own real estate. However, applicants who did not own real estate showed a larger percentage of people having payment issues.

10. Organizations of the applicant

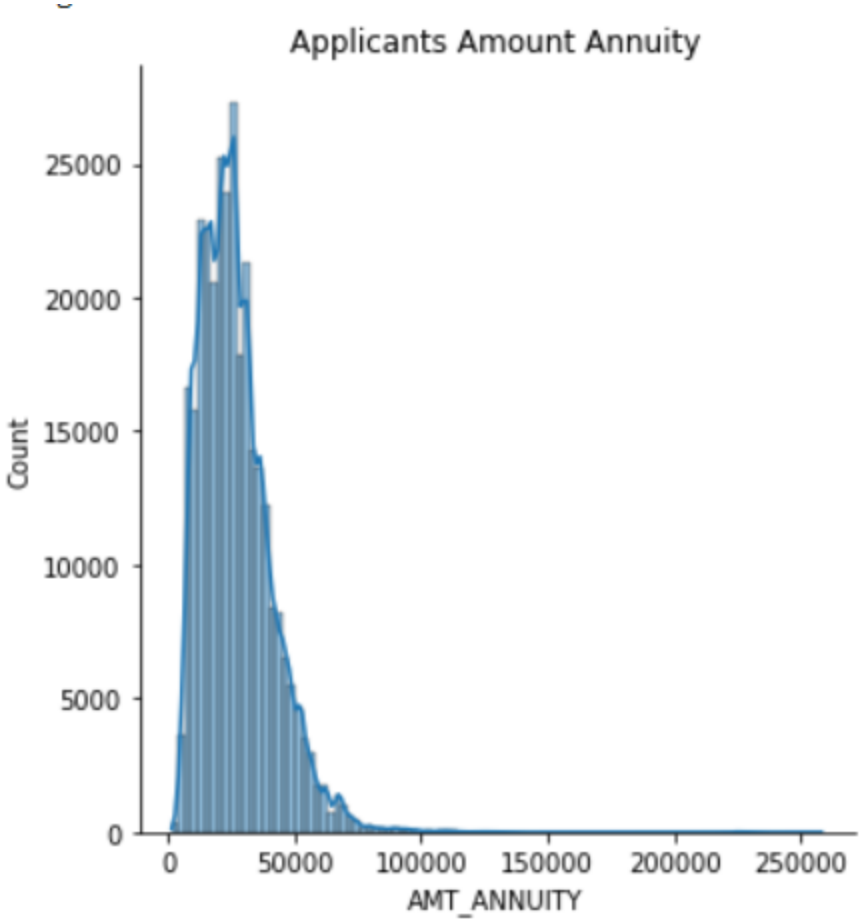


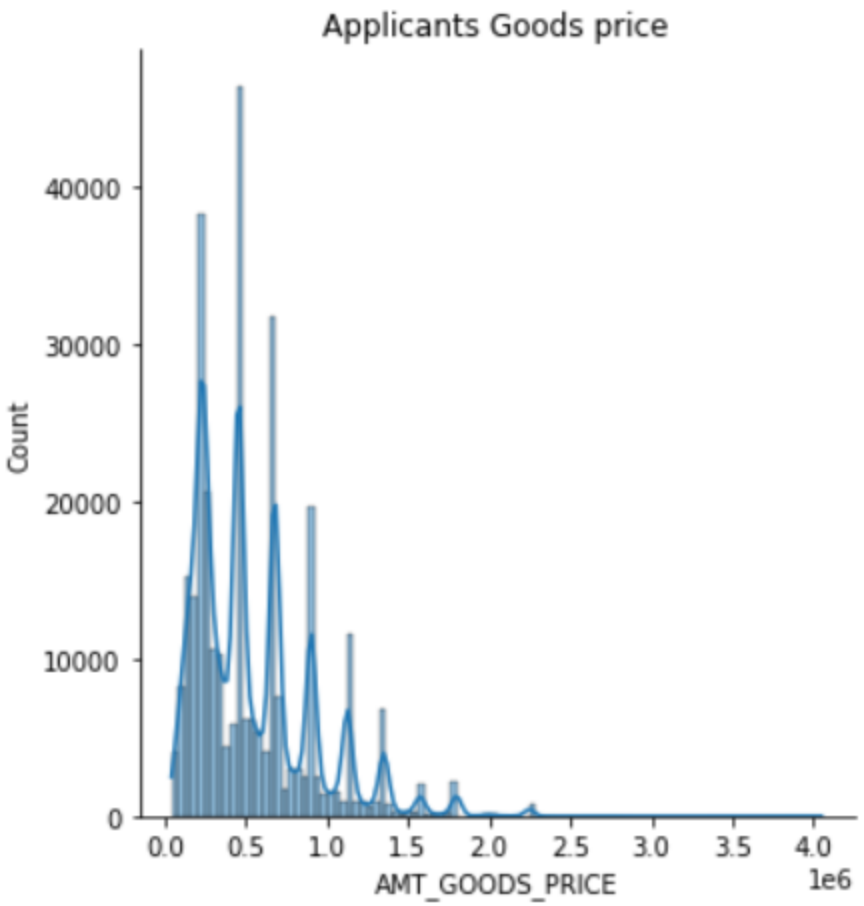
Entity Type 3 type organizations have filed a maximum number of loans equal to approx 67K.

11. Credit amount distribution

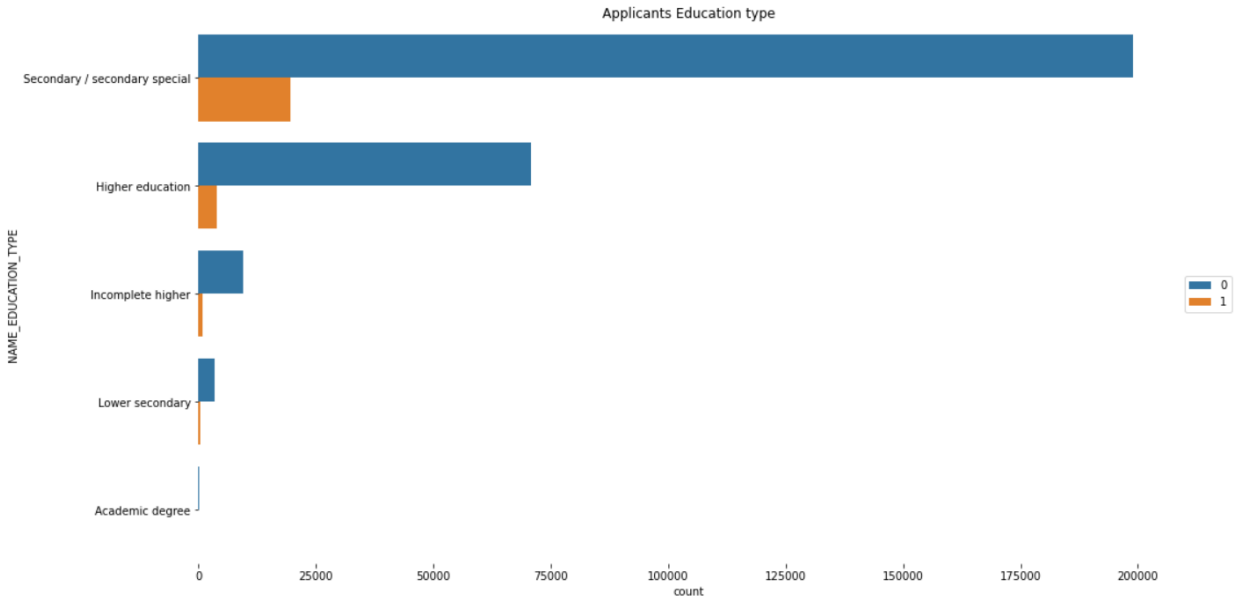


12. Annuity distribution


Applicants Amount Annuity
Count
25000
20000
15000
10000
5000
0
0
50000
100000
150000
200000
250000
AMT_ANNUITY

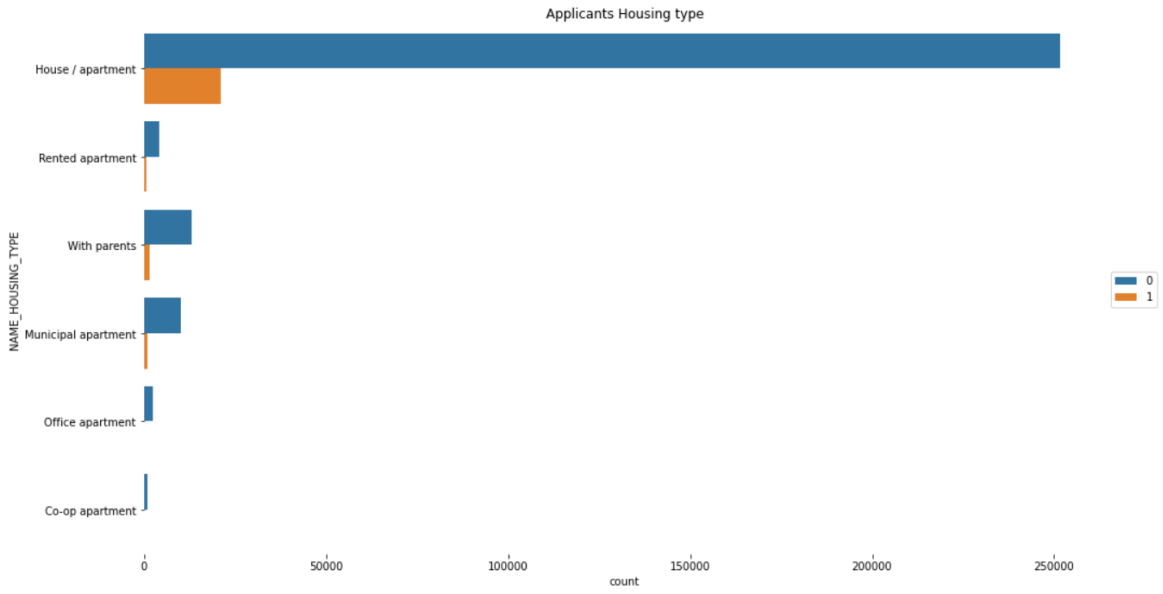


14. Education type



People with secondary education filed the most applications (218K), followed by those with higher education with 75K applications. While it is clear that candidates with lower-level secondary education have the greatest proportion of payment-related issues.

15. Housing type

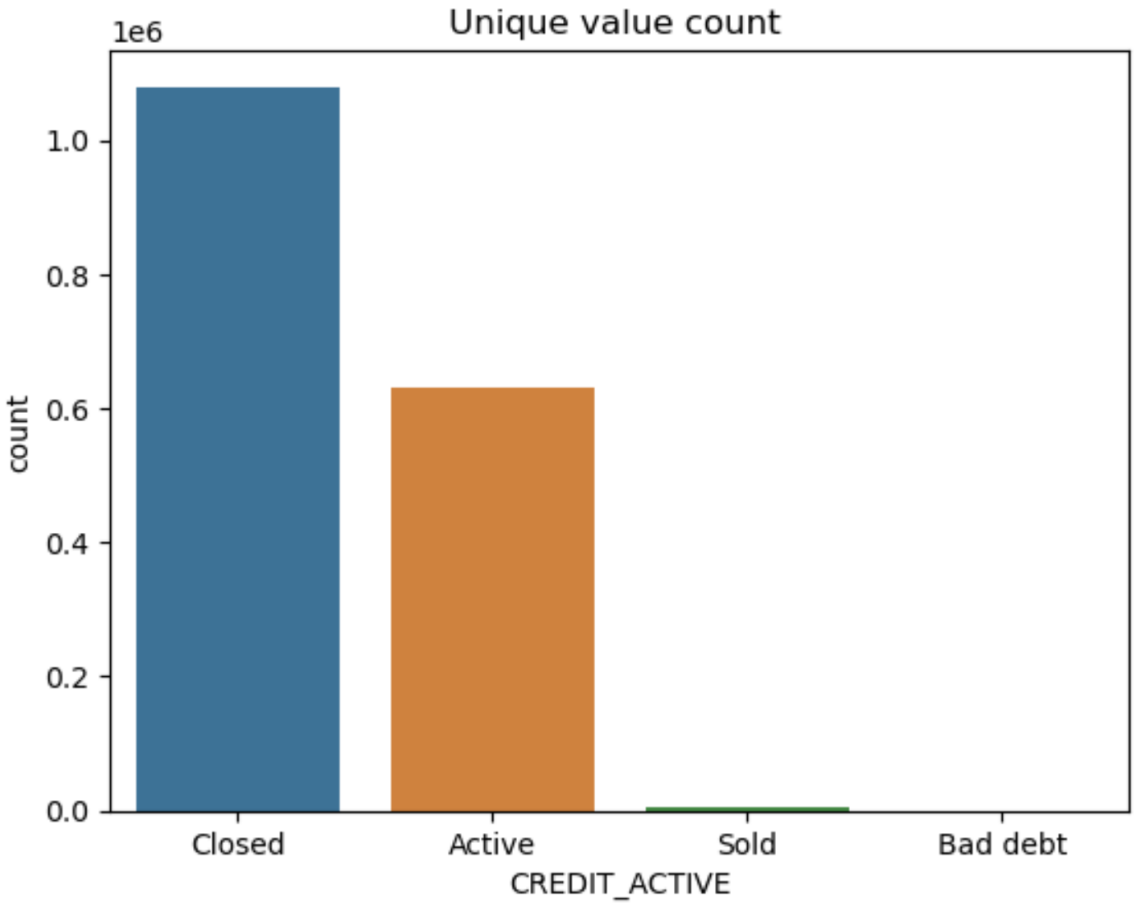


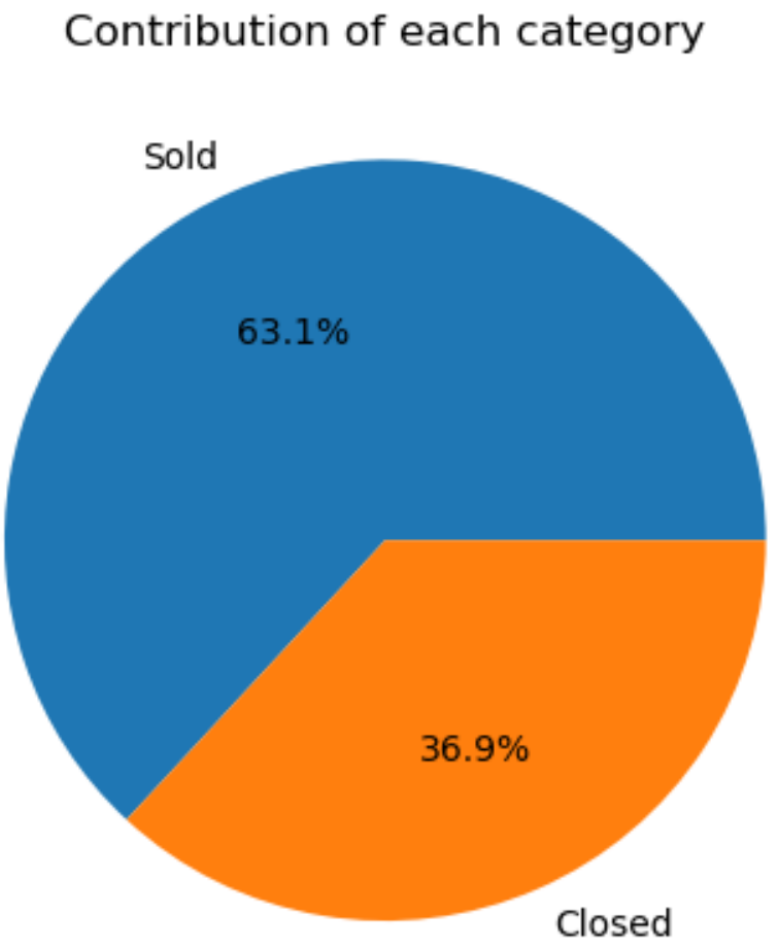
The majority of loan application(roughly 270) come from applicants who live in homes or apartments. Additionally, applicants who live in apartments or with their parents have the greatest proportion of payment-related issues.

Bureau csv

- SK_ID_CURR = ID of loan in our sample - one loan in our sample can have 0,1,2 or more related previous credits in credit bureau
- foreign key used to join application_train.csv
- SK_BUREAU_ID = Recoded ID of previous Credit Bureau credit related to our loan (unique coding for each loan application)
- foreign key used to join credit_bureau.csv
- CREDIT_ACTIVE = Status of the Credit Bureau (CB) reported credits
- Categorical variable containing 4 unique values ('Active', 'Bad debt', 'Closed', and 'Sold)
- There are 0 of missing values for CREDIT_ACTIVE variable
- The different types of categories are {'Sold', 'Closed', 'Bad debt', 'Active'}
    - The count of different categories are :
        - Closed - 1079273
        - Active - 630607
        - Sold - 6527
        - Bad debt - 21

Name: CREDIT_ACTIVE, dtype: in

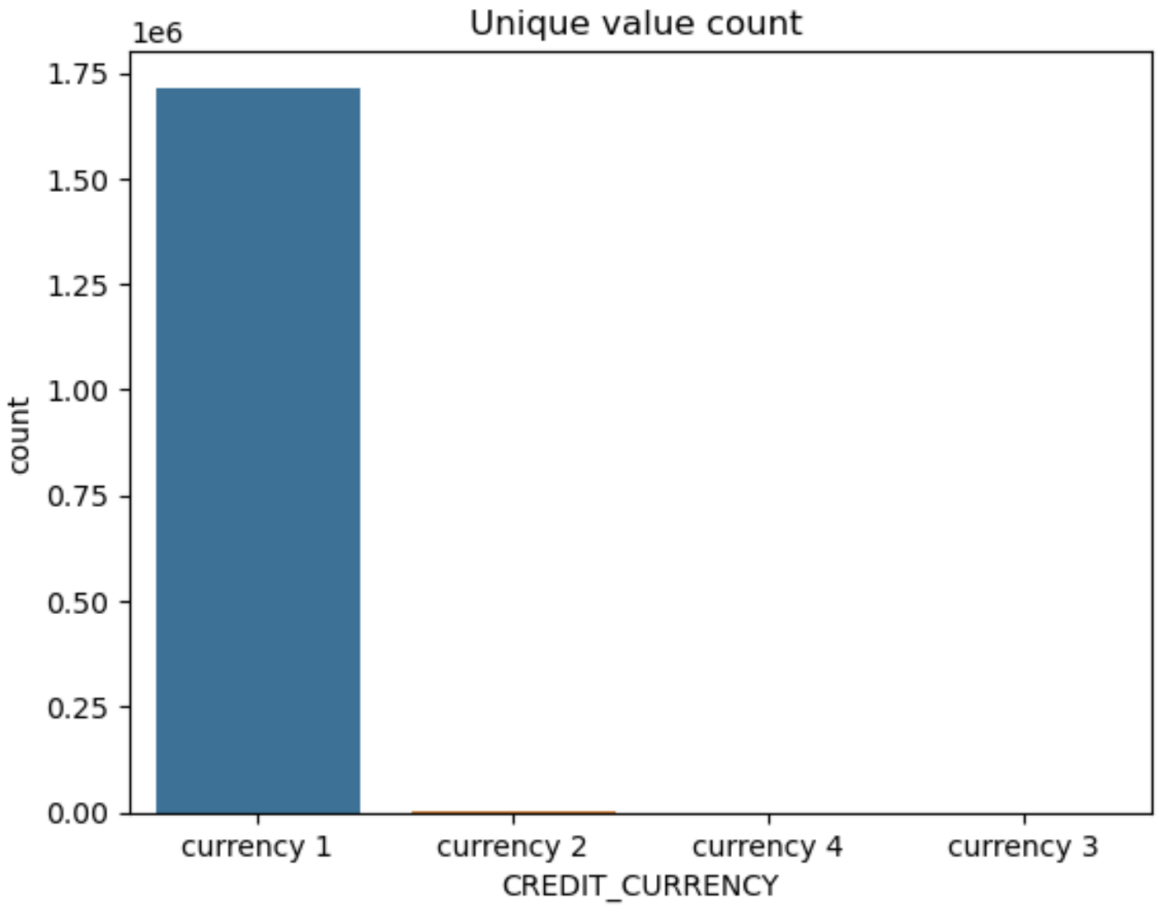

Unique value count
1e6
1.0
0.8
0.6
0.4
0.2
0.0
count
Closed
Active
Sold
Bad debt
CREDIT_ACTIVE

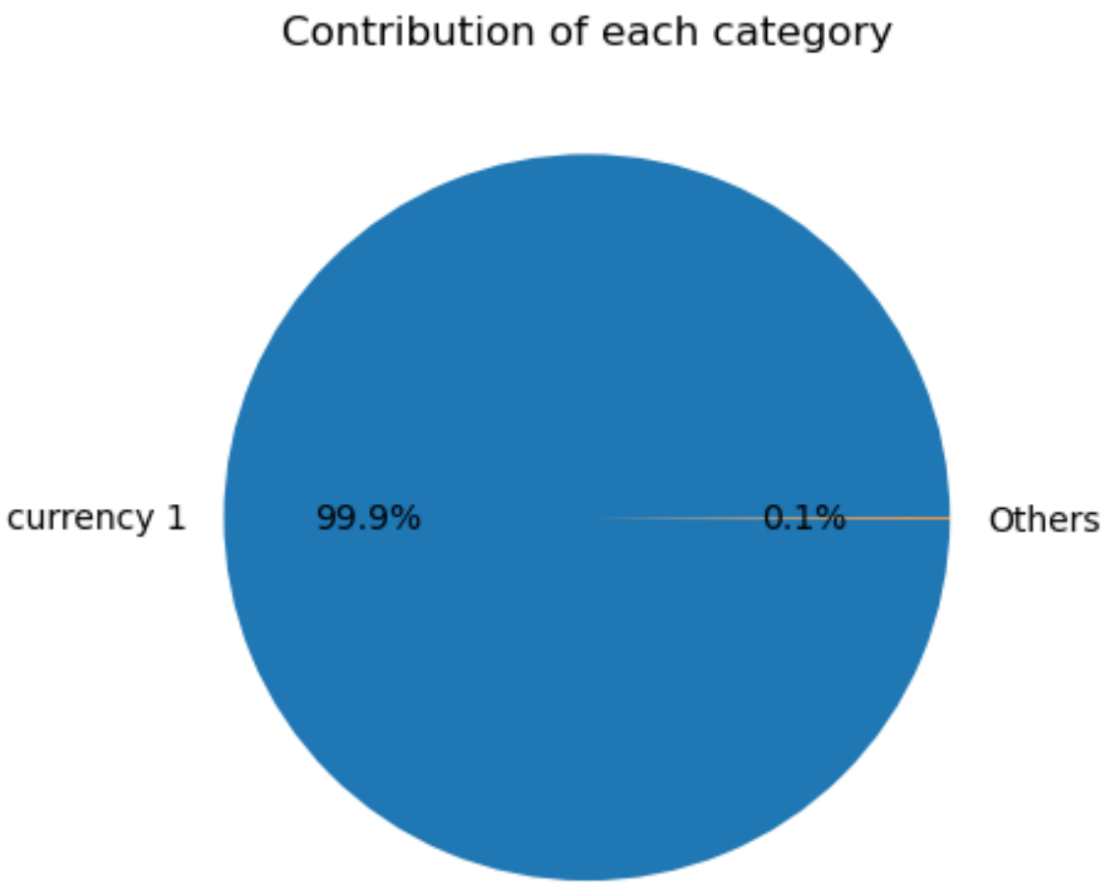


From the above graphs, we can see that there CREDIT_ACTIVE column is a categorical variable having 4 values i.e. Closed, Active, Sold and Bad debt respectively. There are around 1.5 million people for which the credit bureau has reported credit status as Closed which is the highest, followed by 600 thousand Active credit status. Sold and Bad debt credit values have negligible values as compared to Active and Closed categories. 63.1% of the values are contributed by Active status and around 36.9% of the values are contributed by Bad Debt categories

- CREDIT_CURRENCY = Recoded currency of the Credit Bureau credit
- Categorical variable containing 4 types of Currencies ('currency 1', 'currency 2', 'currency 3', and 'currency 4')
- There are 0 of missing values for CREDIT_CURRENCY variable
- The different types of categories are ['currency 1', 'currency 2', 'currency 3', 'currency 4']
    - The count of different categories are :
        - currency 1 - 1715020
        - currency 2 - 1224
        - currency 3 - 174

currency 4 10


Unique value count
1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
count
currency 1
currency 2
currency 4
currency 3
CREDIT_CURRENCY



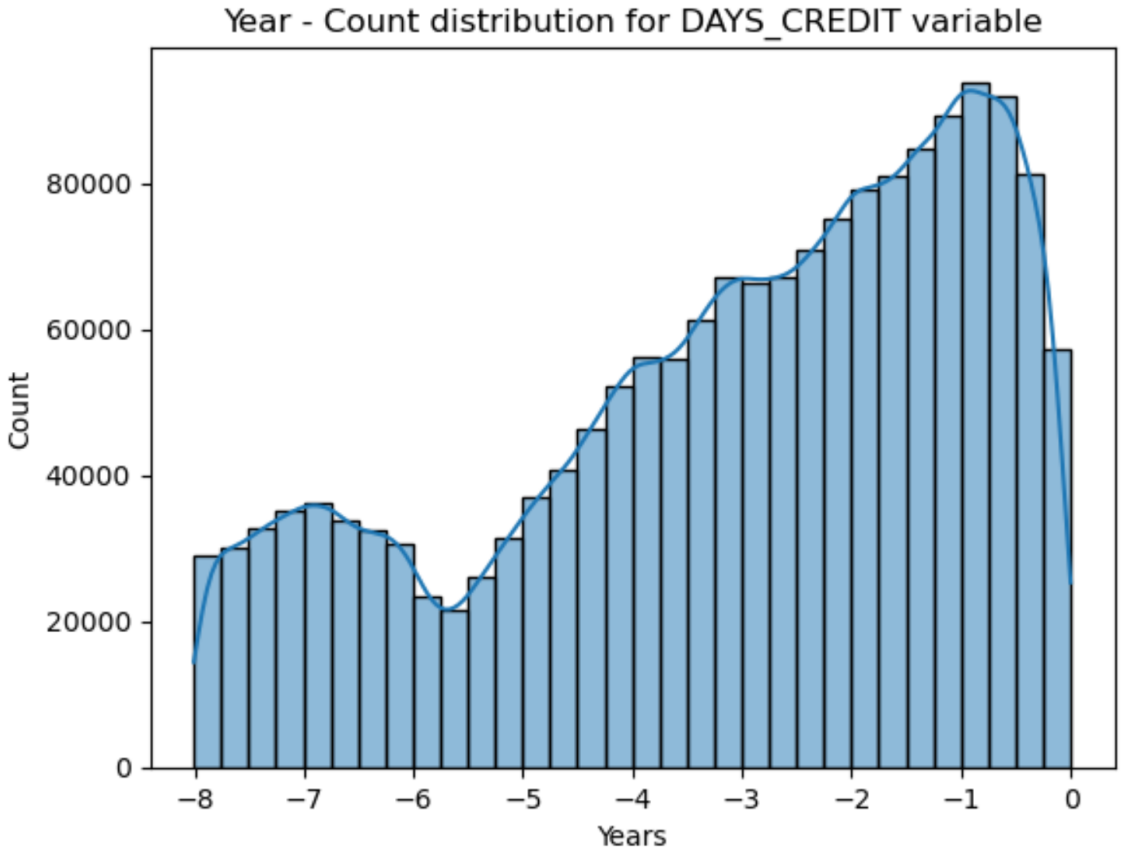
From the above graphs, we can see that the CREDIT_CURRENCY column is a categorical variable having 4 values i.e. 'currency 1', 'currency 2', 'currency 3', and 'currency 4' respectively. There are around 1.7 million people for which the credit bureau has reported credit currency as currency 1 which is the highest, and contributes almost 99.99% of the total reported values

DAYS_CREDIT : How many days before current application did client apply for Credit Bureau credit
Numerical variable containing values from range -2922 to 0
Negative sign only indicates that the applicant has applied prior to the current application date, and has no other significance

Minimum value for DAYS_CREDIT variable is -2922

Maximum value for DAYS_CREDIT variable is 0

There are 0 of missing values for DAYS_CREDIT variable



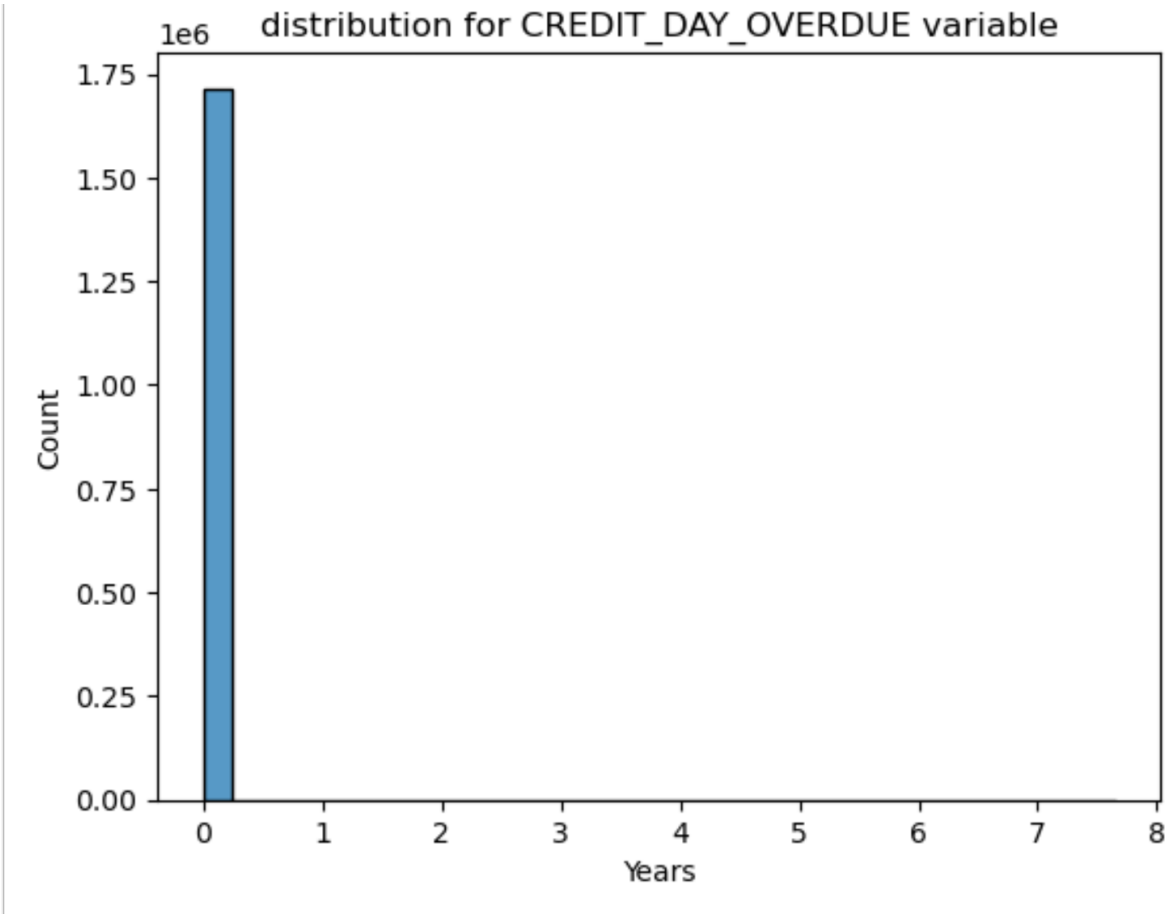
From the above histogram, we can see that most of the people (about 100,000) have applied for bureau credit 1 year prior to their application. The general trend for applying for bureau credit increases from -5 years and has maximum peak values in the range -2.5 years to -0.5 years

CREDIT_DAY_OVERDUE : Number of days past due on CB credit at the time of application for related loan in our sample
Numerical variable containing values from range 0 to 2792

Minimum value for CREDIT_DAY_OVERDUE variable is 0

Maximum value for CREDIT_DAY_OVERDUE variable is 2792

There are 0 of missing values for CREDIT_DAY_OVERDUE variable



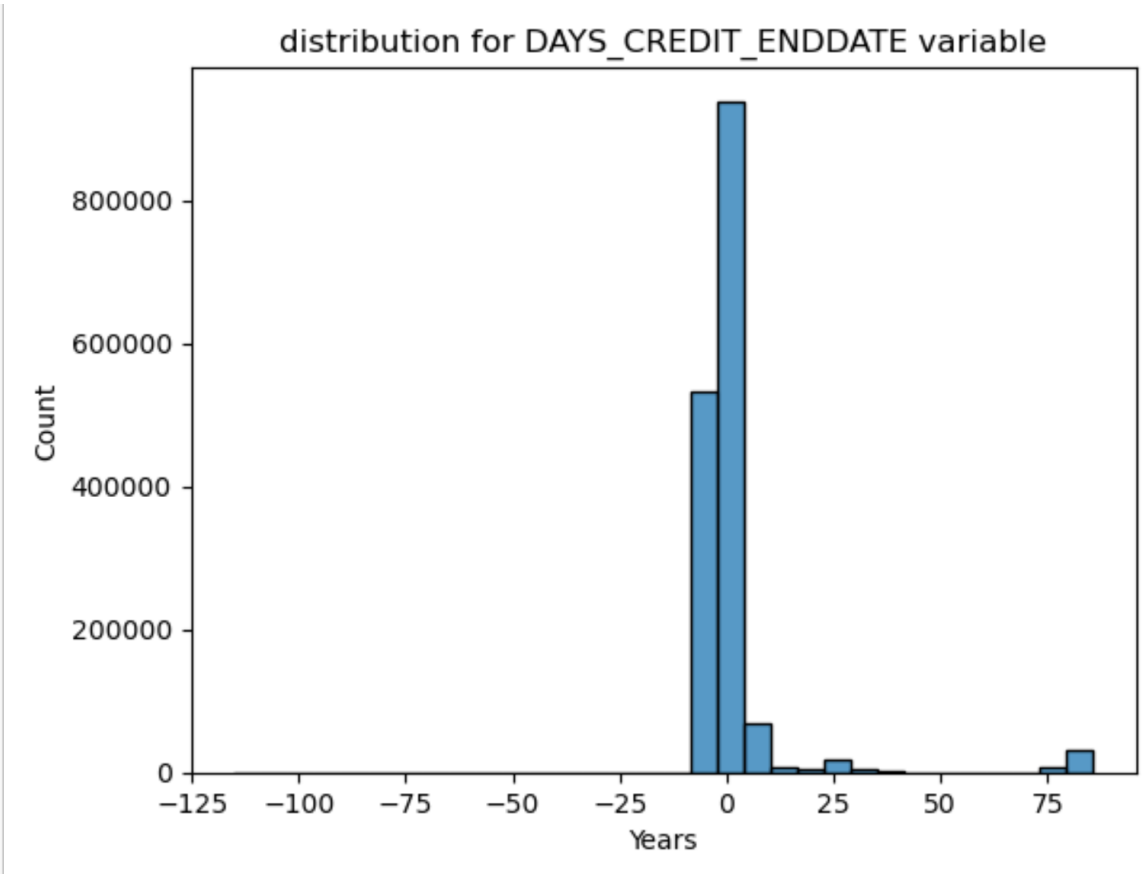
From the above histogram, we can see that almost all of the people (about 99%) have credit overdue for 3 months

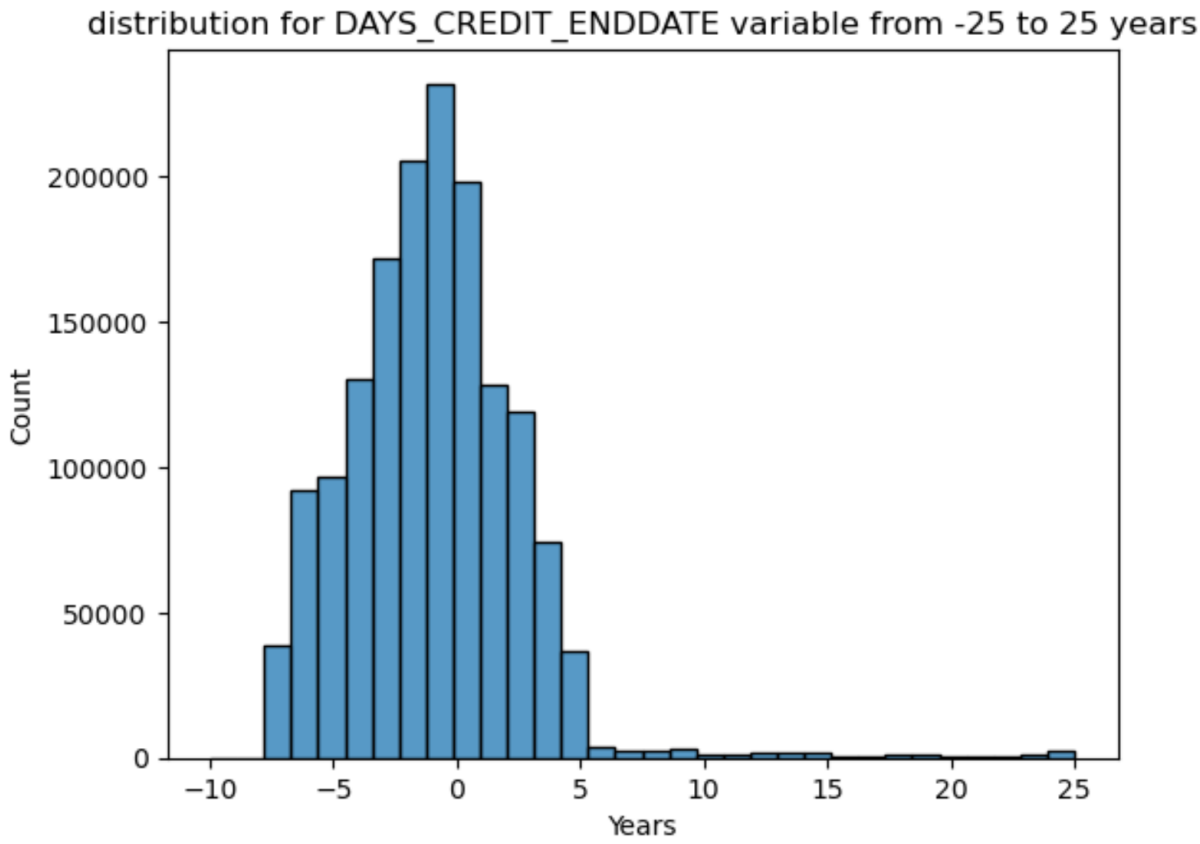
DAYS_CREDIT_ENDDATE : Remaining duration of CB credit (in days) at the time of application in Home Credit
Numerical variable containing values from range 0 to 2792

Minimum value for DAYS_CREDIT_ENDDATE variable is -42060.0 days

Maximum value for DAYS_CREDIT_ENDDATE variable is 31199.0 days

There are 105553 of missing values for DAYS_CREDIT_ENDDATE variable


distribution for DAYS_CREDIT_ENDDATE variable
800000
600000
400000
200000
0
Count
−125
−100
−75
−50
−25
0
25
50
75
Years

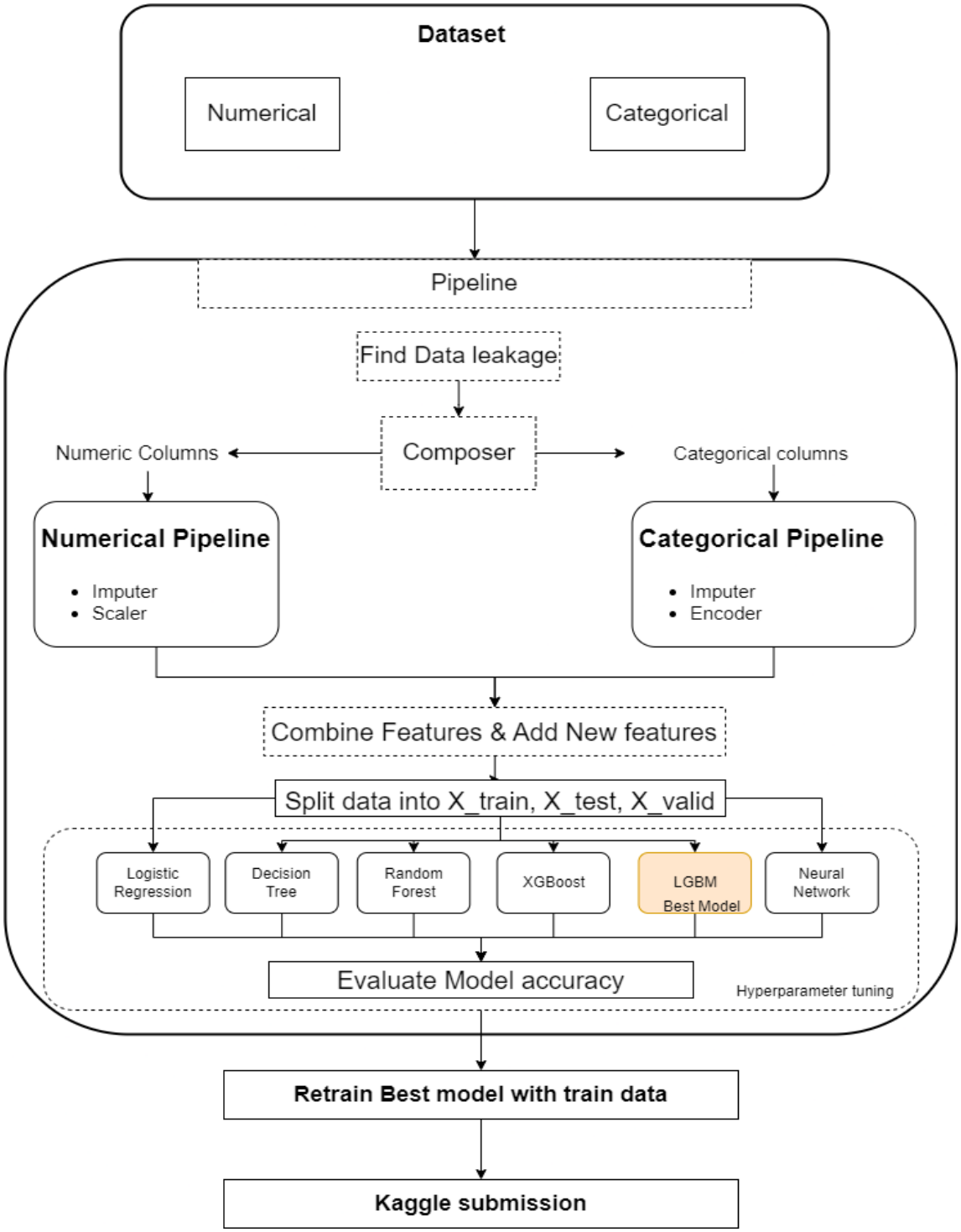


From the above histograms, we can see that most of the values for DAYS_CREDIT_ENDDATE fall in the range -10 years to 25 years.

Negative sign indicates that the client has exhausted their credit, 0 indicates client has no credit and a positive amount indicates how many days does the client have until they are out of credit. This time is only relative to the application

Modeling Pipelines

A visualization of the modeling pipeline



Families of input features and count per family

There are 16 categorical features and 174 numerical features in the sample dataset which is obtained after joining all the given csv files data. The unique categories for each of the categorical features are given below.

```
NAME_CONTRACT_TYPE             2
FLAG_OWN_CAR                   2
FLAG_OWN_REALTY                2
EMERGENCYSTATE_MODE            2
CODE_GENDER                    3
HOUSETYPE_MODE                 3
FONDKAPREMONT_MODE             4
NAME_EDUCATION_TYPE            5
NAME_FAMILY_STATUS             6
NAME_HOUSING_TYPE              6
NAME_TYPE_SUITE                7
WEEKDAY_APPR_PROCESS_START     7
WALLSMATERIAL_MODE             7
NAME_INCOME_TYPE               8
OCCUPATION_TYPE               18
ORGANIZATION_TYPE             58
dtype: int64
```

Number of input features

After applying One Hot encoding on the categorical features, there will be additional 124 new features along with the given 16 categories which makes a total of 140 categorical features. Adding the numerical 174 numerical features, we get a total **314 input features** which will be used to train our model.

Loss function used (data loss and regularization parts)

We have used the following loss functions to train our model:

L1 Regularization (Lasso)

$$\sum_{i=1}^{n}(y_i - \sum_{j} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

L2 Regularization (Ridge)

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Elasticnet

$$L_{enet}(\hat{\beta}) = \frac{\sum_{i-1}^{n}(y_i - x_i'\hat{\beta})^2}{2n} + \lambda(\frac{1-\alpha}{2}\sum_{j-1}^{m}\hat{\beta}_j^2 + \alpha\sum_{j-1}^{m}|\hat{\beta}_j|),$$

Loss function for LGBM

$$V_{j|O}(d) = \frac{1}{n_o}\left(\frac{\left(\sum_{\{x_i \in O : x_{ij} \le d\}} g_i\right)^2}{n^j_{l|O}(d)} + \frac{\left(\sum_{\{x_i \in O : x_{ij} > d\}} g_i\right)^2}{n^j_{r|O}(d)}\right)$$

where $n_O = \sum I[xi \in O]$, $n^j_{l|O}(d) = \sum I[xi \in O : xij \le d]$ and $n^j_{r|O}(d) = \sum I[xi \in O : xij > d]$.

Loss function for XGBoost

Real value (label) known from the training data-set

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

Can be seen as f(x + Δx) where x = $\hat{y_i}^{(t-1)}$

Activation Functions for Neural networks

RELU

$$Relu(z) = max(0, z)$$

Sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

LogSigmoid

$$LogSigmoid(z) = \log(\frac{1}{1+e^{-z}})$$

CELU

$$CELU(x) = max(0, x) + min(0, \alpha * (exp(x/\alpha) - 1))$$

Loss Functions for Neural networks

CrossEntropyLoss

$$-(y \log(p) + (1 - y) \log(1 - p))$$

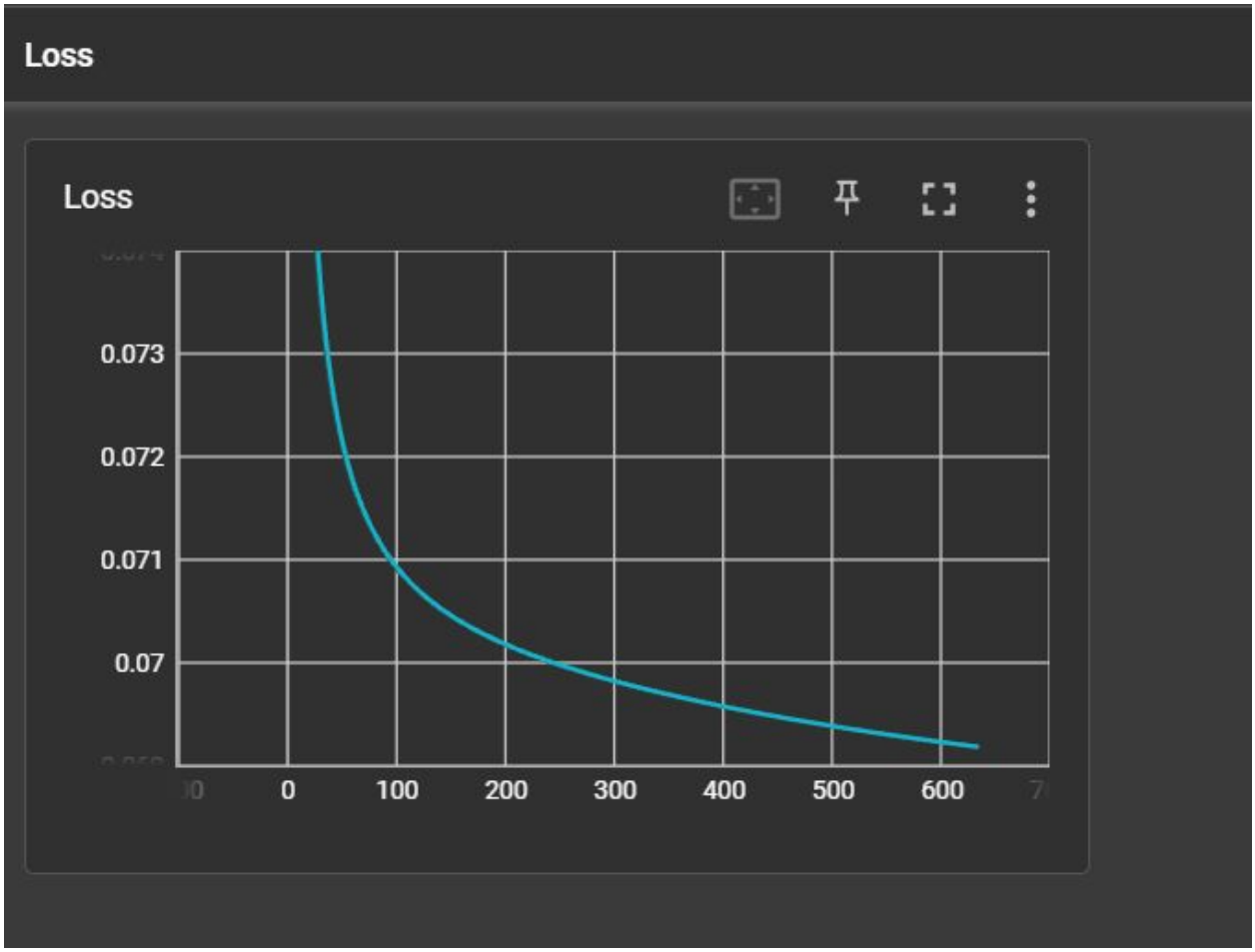
MSELoss

$$\sum_{i=1}^{D} (x_i - y_i)^2$$

Neural network architecture:

In the neural network architecture, 327 features were passed to the input layer. After this, the output of this input layer was fed to the hidden layers. We experimented with the number of hidden layers. Firstly we applied 15 hidden layers and then we tried with 10 and 20 hidden layers. After this, the output of these hidden layers was fed to the activation functions like sigmoid, Tanh, RELU, and CELU. Finally, we calculated the loss by using the loss functions like MSE and cross-entropy loss.

We used Tensorboard to continuously monitor the neural network training. In the Tensorboard we added multiple checks to ensure the model is running smoothly and monitored loss after each epoch as follows.



Tensorboard was very helpful for monitoring training and debugging the issues in the neural network.

The number of experiments conducted

We have done the following experiments and chosen the best parameters for the below mentioned models

1. Logistic Regression: 5 cross-validation folds X 9 candidate parameters + 1 experiment to choose the best model = 46 experiments
2. Decision Tree: 5 cross-validation folds X 12 candidates parameters + 1 experiment to choose the best model = 61 experiments
3. XGBoost = 5 cross-validation folds X 80 candidates parameters + 1 experiment to choose the best model = 241 experiments
4. LGBM = 5 cross-validation folds X 37 candidates parameters + 1 experiment to choose the best model = 181 experiments
5. Random forest = 5 cross-validation folds X 4 candidate parameters + 1 experiment to choose the best model = 21 experiments
6. Neural network = 32 candidate parameters + 1 experiment to choose the best model = 33 experiments

Total number of experiments conducted = 46 + 61 + 241 + 181 + 21 + 33 = 583 experiments

Experiment table with the following details per experiment

Baseline experiment

1. **Logistic Regression:**
   The parameters used for logistic regression are as follows:
   - **C:** Inverse of regularization strength; must be a positive float. Smaller values specify stronger regularization.
   - **Penalty:** The type of penalty applied: Ridge, Lasso, or elasticnet

   We have used the following parameters in our experiment:
   - 'lr__C' : [10, 100],
   - 'lr__penalty' : ['l1', 'l2', 'elasticnet']
2. **Decision Tree:**
   The parameters used for the decision tree are as follows:
   - **max_depth:** The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

   We have used the following parameters in our experiment:
   - 'dt__max_depth' : [10, 15, 20]

The families of input features used

We have used the numerical and categorical features to train the model.

Train/valid/test record Accuracy using ROC AUC as follow

ML models accuracy

```
experimentLog
```

	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Best Params
0	Baseline logistic_regression with 334 inputs	{'lr__C': [1, 0.1, 0.01], 'lr__penalty': ['l1'...	91.95%	91.96%	91.84%	0.087670	0.042359	{'memory': None, 'steps': [('lr', LogisticRegr...
1	Baseline logistic_regression with 334 inputs	{'lr__C': [1, 0.1, 0.01], 'lr__penalty': ['l1'...	91.95%	91.96%	91.84%	0.082613	0.044088	{'memory': None, 'steps': [('lr', LogisticRegr...
2	Baseline logistic_regression with 334 inputs	{'lr__C': [1, 0.1, 0.01], 'lr__penalty': ['l1'...	91.95%	91.96%	91.84%	0.072386	0.035532	{'memory': None, 'steps': [('lr', LogisticRegr...
3	Baseline decision_tree with 334 inputs	{'dt__max_depth': [10, 15, 20], 'dt__splitter'...	92.38%	91.67%	91.58%	0.147465	0.065180	{'memory': None, 'steps': [('dt', DecisionTree...
4	Baseline random_forest with 334 inputs	{'rf__max_depth': [5, 10], 'rf__n_estimators':...	91.98%	91.97%	91.84%	2.507257	1.020463	{'memory': None, 'steps': [('rf', RandomForest...
5	Baseline xgboost with 334 inputs	{'xgb__min_child_weight': [19.0], 'xgb__object...	91.99%	91.98%	91.86%	0.046852	0.023269	{'memory': None, 'steps': [('xgb', XGBClassifi...
6	**Baseline lightgbm with 334 inputs**	**{'LGBMClassifier__boosting_type': ['gbdt'], 'L...**	**92.28%**	**92.00%**	**91.89%**	**0.061664**	**0.028334**	**{'memory': None, 'steps': [('LGBMClassifier', ...**

The LGBM Classifier is chosen as the best model as the test accuracy observed is 91.89 which is the highest of all the models trained.

Neural networks results using ROC AUC for different parameters

```
nn_experiment_log = pd.DataFrame.from_records(model_params_list)
nn_experiment_log.sort_values(by='ROC_AUC_VALID', ascending=False)
```

	act_fn	epoch	hidden_layers	batch_size	loss_fn	learning_rate	Model	ROC_AUC_TRAIN	ROC_AUC_TEST	ROC_AUC_VALID
26	CELU(alpha=1.0)	1000	10	100	MSELoss()	0.10	[Linear(in_features=327, out_features=10, bias...	0.758773	0.753406	0.747729
30	CELU(alpha=1.0)	1000	20	100	MSELoss()	0.10	[Linear(in_features=327, out_features=20, bias...	0.758366	0.752126	0.747559
2	ReLU()	1000	10	100	MSELoss()	0.10	[Linear(in_features=327, out_features=10, bias...	0.753172	0.744697	0.739545
6	ReLU()	1000	20	100	MSELoss()	0.10	[Linear(in_features=327, out_features=20, bias...	0.753764	0.745593	0.735856
31	CELU(alpha=1.0)	1000	20	100	MSELoss()	0.01	[Linear(in_features=327, out_features=20, bias...	0.737027	0.735019	0.729359
27	CELU(alpha=1.0)	1000	10	100	MSELoss()	0.01	[Linear(in_features=327, out_features=10, bias...	0.737097	0.733433	0.729082
18	Sigmoid()	1000	10	100	MSELoss()	0.10	[Linear(in_features=327, out_features=10, bias...	0.721396	0.725235	0.712639
22	Sigmoid()	1000	20	100	MSELoss()	0.10	[Linear(in_features=327, out_features=20, bias...	0.717058	0.719671	0.707915
3	ReLU()	1000	10	100	MSELoss()	0.01	[Linear(in_features=327, out_features=10, bias...	0.703757	0.703854	0.698070
7	ReLU()	1000	20	100	MSELoss()	0.01	[Linear(in_features=327, out_features=20, bias...	0.696450	0.698363	0.687185
10	LogSigmoid()	1000	10	100	MSELoss()	0.10	[Linear(in_features=327, out_features=10, bias...	0.683227	0.689331	0.673940
14	LogSigmoid()	1000	20	100	MSELoss()	0.10	[Linear(in_features=327, out_features=20, bias...	0.665940	0.675992	0.661024
23	Sigmoid()	1000	20	100	MSELoss()	0.01	[Linear(in_features=327, out_features=20, bias...	0.650914	0.650846	0.653459
11	LogSigmoid()	1000	10	100	MSELoss()	0.01	[Linear(in_features=327, out_features=10, bias...	0.625054	0.636910	0.624666
19	Sigmoid()	1000	10	100	MSELoss()	0.01	[Linear(in_features=327, out_features=10, bias...	0.607068	0.606676	0.602565
29	CELU(alpha=1.0)	1000	20	100	CrossEntropyLoss()	0.01	[Linear(in_features=327, out_features=20, bias...	0.575723	0.576083	0.578610
5	ReLU()	1000	20	100	CrossEntropyLoss()	0.01	[Linear(in_features=327, out_features=20, bias...	0.561847	0.565762	0.568244
17	Sigmoid()	1000	10	100	CrossEntropyLoss()	0.01	[Linear(in_features=327, out_features=10, bias...	0.580325	0.577450	0.566948
15	LogSigmoid()	1000	20	100	MSELoss()	0.01	[Linear(in_features=327, out_features=20, bias...	0.548312	0.547595	0.540337
25	CELU(alpha=1.0)	1000	10	100	CrossEntropyLoss()	0.01	[Linear(in_features=327, out_features=10, bias...	0.546295	0.550186	0.540192

Best Neural Network:

327 → 10 → CELU → 1 → MSELoss

Feature Engineering

New features

We have added new features to our dataset from the existing features from application_train.csv

1. NEW_CAR_TO_BIRTH_GAP = DAYS_BIRTH - OWN_CAR_AGE

This feature calculates the difference between the columns days birth and own_car age

2. NEW_EMPLOY_TO_BIRTH_GAP = DAYS_BIRTH - DAYS_EMPLOYED

This feature calculates the difference between the columns days birth and days employed

3. YEARS_BIRTH = DAYS_BIRTH / 365.25

Here, the birth year is calculated by dividing the days birth column by 365.25

4. Credit/Income = AMT_CREDIT / AMT_INCOME_TOTAL

This feature calculated by dividing amount credited by total amount of income

5. Annuity/Income = AMT_ANNUITY / AMT_INCOME_TOTAL

This feature is calculated by dividing the annuity column by total amount of income

6. Employed/Birth = DAYS_EMPLOYED / DAYS_BIRTH

This feature calculates the percentage of days employed by dividing the days_employed column b days_birth

7. Flag_Greater_32 = YEARS_BIRTH > 32 or not

This feature returns the value 1 if the year of birth is greater than 32 else returns 0

8. Flag_Employment_Greater_5 = DAYS_EMPLOYED / -365.25 is greater than 5 or not

This feature checks whether the total years of employment is greater than 5 or not. If the total employment year is greater than 5, then it returns 1 else returns 0

9. Flag_Income_Greater_Credit = AMT_INCOME_TOTAL > AMT_CREDIT

This feature compares the values between the two columns total amount of income and total amount of credit(loan taken by the customer)

10. previous_loan_counts = count(SK_ID_BUREAU) in previous applications.csv

This feature counts the applications that the user has done before their loan was approved.

11. Polynomial Features: Here we have created polynomial features for columns having top positive and negative correlations where we added new features (columns) where values were calculated by squaring the values in X, e.g. X^2. or X^3

    1. EXT_SOURCE_3_power_2

2. EXT_SOURCE_2_power_2
3. EXT_SOURCE_3_power_3
4. EXT_SOURCE_2_power_3
5. EXT_SOURCE_1_power_2
6. EXT_SOURCE_1_power_3
7. DAYS_BIRTH_power_2
8. Flag_Greater_32
9. Flag_Employment_Greater_5
10. Flag_Income_Greater_Credit
11. Credit/Income
12. Annuity/Income
13. Employed/Birth
14. REGION_RATING_CLIENT_power_3
15. REGION_RATING_CLIENT_power_2
16. REGION_RATING_CLIENT_W_CITY_power_3
17. REGION_RATING_CLIENT_W_CITY_power_2
18. DAYS_BIRTH_power_3
19. PREVIOUS_APPLICATIONS_COUNT

Correlation of new features with target variable

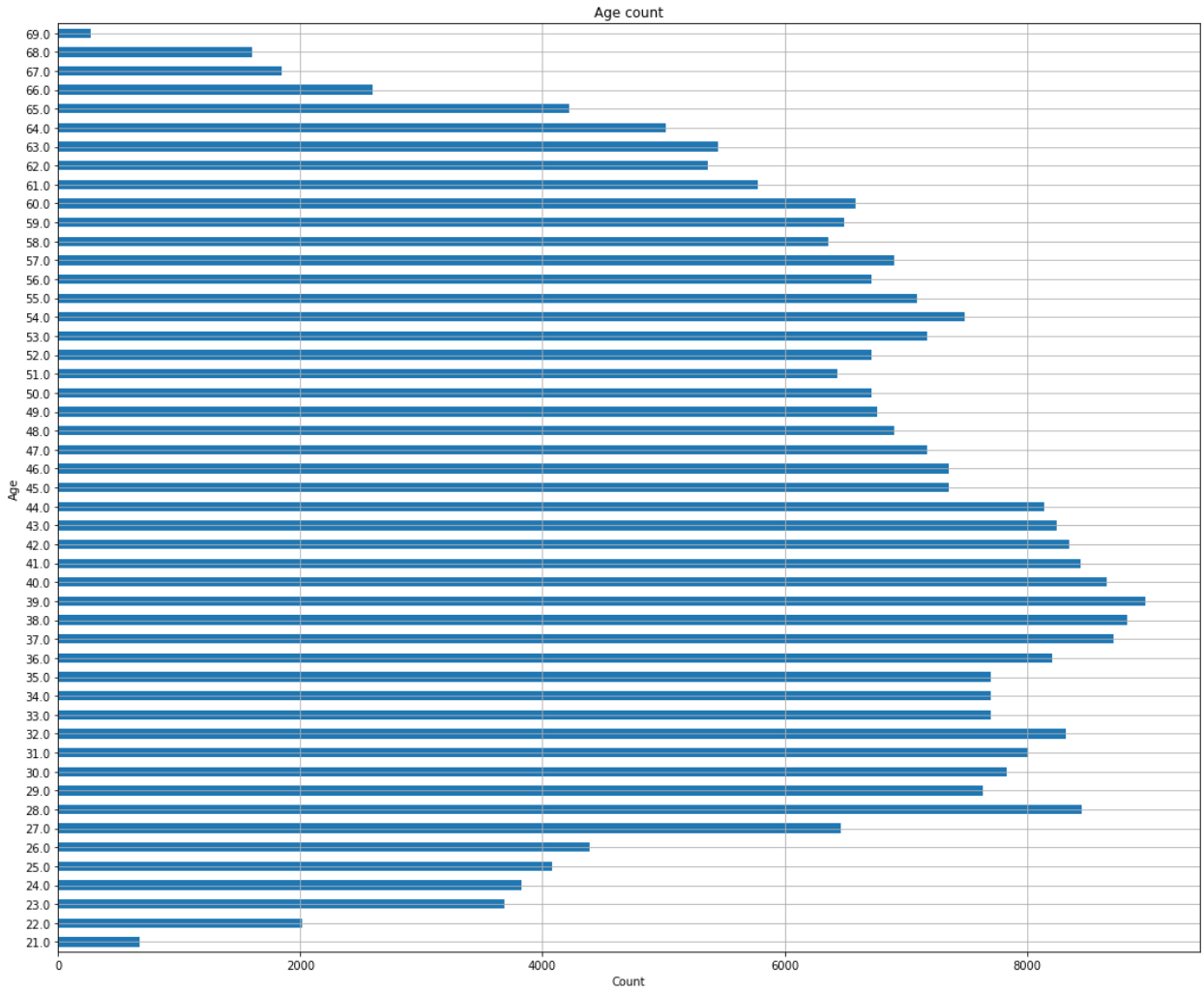
1. **New features correlation with TARGET**

```
EXT_SOURCE_3_power_2                  -0.160346
EXT_SOURCE_2_power_2                  -0.149669
EXT_SOURCE_3_power_3                  -0.143904
EXT_SOURCE_2_power_3                  -0.140392
EXT_SOURCE_1_power_2                  -0.139878
EXT_SOURCE_1_power_3                  -0.126739
DAYS_BIRTH_power_2                    -0.076672
Flag_Greater_32                       -0.058933
Flag_Employment_Greater_5             -0.047424
Flag_Income_Greater_Credit            -0.012294
Credit/Income                         -0.007727
Annuity/Income                         0.014265
Employed/Birth                         0.042206
REGION_RATING_CLIENT_power_3           0.055685
REGION_RATING_CLIENT_power_2           0.057930
REGION_RATING_CLIENT_W_CITY_power_3    0.058058
REGION_RATING_CLIENT_W_CITY_power_2    0.060256
DAYS_BIRTH_power_3                     0.074273
TARGET                                 1.000000
Name: TARGET, dtype: float64
```

2. **New features correlation with TARGET**

	NEW_CAR_TO_BIRTH_GAP	NEW_EMPLOY_TO_BIRTH_GAP	TARGET
NEW_CAR_TO_BIRTH_GAP	1.000000	0.534538	0.056881
NEW_EMPLOY_TO_BIRTH_GAP	0.534538	1.000000	0.046451
TARGET	0.056881	0.046451	1.000000

3. Age distribution



Data leakage

Data leakage is when information from outside the training dataset is used to create the model. This additional information can allow the model to learn or know something that it

otherwise would not know and in turn invalidate the estimated performance of the mode being constructed.

To check the data leakage, following steps are followed:

1. Check if any ‘SK_ID_CURR’ is common between test and train data.

```
# check if any common SK_ID_CURR
new_df = datasets['application_train'].merge(datasets['application_test'], how='inner', on='SK_ID_CURR')
new_df.shape
```

```
(0, 422)
```

2. Check any data points are same between test and train(excluding ‘SK_ID_CURR’)

```
# check if any other columns are common other than TARGET and SK_ID_CURR
columns = list(columns)
x_train_new_df = datasets['application_train'][columns]
x_test_new_df = datasets['application_test'][columns]
print(x_train_new_df.shape)
new_df = x_train_new_df.merge(x_test_new_df, how='inner', on=columns)
new_df.shape
```

```
(307511, 210)
(0, 210)
```

3. See if any categorical values are not in test data those are considered in the train data as we are creating dummy features using One Hot Encoding.

```
for column in cat_features:
    train_cats = set(datasets['application_train'][column].value_counts().index)
    test_cats = set(datasets['application_test'][column].value_counts().index)
    if len(train_cats.difference(test_cats)) > 0:
        print(f"for {column} difference between train and cat is {train_cats.difference(test_cats)}")
```

```
for CODE_GENDER difference between train and cat is {'XNA'}
for NAME_INCOME_TYPE difference between train and cat is {'Maternity leave'}
for NAME_FAMILY_STATUS difference between train and cat is {'Unknown'}
[ColumnTransformer] ..... (2 of 2) Processing cat, total= 1.7min
```

3 columns has 1 each categorical value in extra in the training dataset as shown above.

Hyperparameter Tuning

Hyperparameter tuning techniques

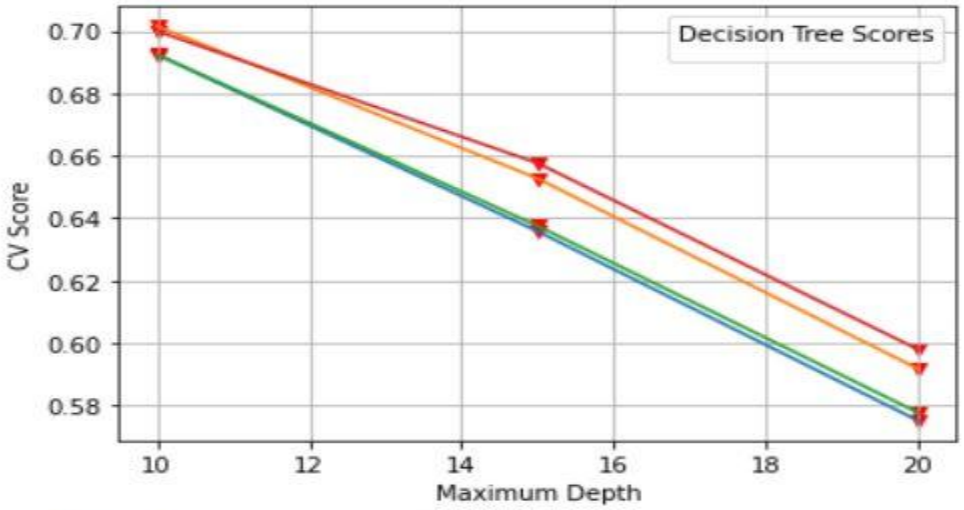
1. **GridSearchCV** - GridSearchCV is used for finding the best parameter values from a given set of parameters for a model in a grid and extracts the best parameter values to find the predictions.

Here are the screenshots of the best parameters we got for each model after hyperparameter tuning is given below:

```
[('lr', LogisticRegression(C=1, solver='saga'))]
[('lr', LogisticRegression(C=1, solver='saga'))]
[('lr', LogisticRegression(C=1, solver='saga'))]
[('dt', DecisionTreeClassifier(max_depth=10, splitter='random'))]
[('rf', RandomForestClassifier(max_depth=10))]
[('xgb', XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.4,
              enable_categorical=False, eta=0.025, eval_metric='auc',
              gamma=0.65, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.0250000004,
              max_delta_step=1.8, max_depth=7, min_child_weight=19.0,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=96, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=0.8,
              tree_method='exact', validate_parameters=1, verbosity=None))]
[('LGBMClassifier', LGBMClassifier())]
```

5:29 PM

Hyperparameter Tuning for Decision Tree Classifier model: Here we have plotted a graph for CV score versus maximum depth




```
depth:  10,  score: 0.6924,  label:best__2
depth:  10,  score: 0.7015,  label:random__2
depth:  10,  score: 0.6925,  label:best__4
depth:  10,  score: 0.6999,  label:random__4
depth:  15,  score: 0.6358,  label:best__2
depth:  15,  score: 0.6527,  label:random__2
depth:  15,  score: 0.6375,  label:best__4
depth:  15,  score: 0.6576,  label:random__4
depth:  20,  score: 0.5751,  label:best__2
depth:  20,  score: 0.5916,  label:random__2
depth:  20,  score: 0.5776,  label:best__4
depth:  20,  score: 0.5979,  label:random__4
```

The best parameter for Decision Tree Classifier is
{ max_depth = 10 and splitter = 'Random' }

As compared to the baseline model, we observed an decrease in accuracy after increasing the max depth because the splitter parameter was added.

2. **ROC / AUC** - We are experimenting with different model parameters and using ***roc_auc*** to calculate accuracy
   Model Parameters used for the model are:

   1) Logistic Regression:
   Accuracy:
   Train Accuracy = 91.95%
   Validation Accuracy = 91.96%
   Test Accuracy = 91.84%

   Best Logistic Parameters :

```
logit_params = {'lr__C' : 1,
         'lr__penalty' : 'l1'
        }
```

2) Decision Tree:

Accuracy:

Train Accuracy = 92.38%

Validation Accuracy = 91.67%

Test Accuracy = 91.58%

Best Decision Tree Parameters :

```
decision_tree_params = {
   'dt__max_depth' : 10,
   'dt__splitter': ["random"],
   'dt__min_samples_split': [2] }
```

3) Random Forest:

Accuracy:

Train Accuracy = 91.98%

Validation Accuracy = 91.97%

Test Accuracy = 91.84%

Best Random Forest Parameters :

```
random_forest_param = {
   'rf__max_depth' : [10],
   'rf__n_estimators' : [100],
   'rf__criterion' : ['gini']
}
```

4) XGBoost

Accuracy:

Train Accuracy = 91.99%

Validation Accuracy = 91.98%

Test Accuracy = 91.86%

Best XGBoost Parameters :

```
xgboost_params = {
   'xgb__min_child_weight': [19.0],
   'xgb__objective': ['binary:logistic'],
   'xgb__max_depth': [7],
   'xgb__eta': [0.025],
   'xgb__eval_metric': ['auc'],
   'xgb__max_delta_step': [1.8],
   'xgb__colsample_bytree':[0.4],
   'xgb__subsample': [0.8],
   'xgb__gamma': [0.65] }
```

Screenshot of XGBoost best parameters after gridsearch:

```
Pipeline(steps=[('xgb',
                 XGBClassifier(base_score=0.5, booster='gbtree',
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=0.1, enable_categorical=False,
                               eta=0.025, eval_metric='auc', gamma=0.65,
                               gpu_id=-1, importance_type=None,
                               interaction_constraints='',
                               learning_rate=0.0250000004, max_delta_step=1.8,
                               max_depth=8, min_child_weight=25.0, missing=nan,
                               monotone_constraints='()', n_estimators=100,
                               n_jobs=96, num_parallel_tree=1, predictor='auto',
                               random_state=0, reg_alpha=0, reg_lambda=1,
                               scale_pos_weight=1, subsample=0.8,
                               tree_method='exact', validate_parameters=1,
                               verbosity=None))])
```

5) LGBM Classifier

Accuracy:

Train Accuracy = 92.28%
Validation Accuracy = 92.00%
Test Accuracy = 91.89%

Best LGBM Parameters :

```
lightgbm_params={
   'LGBMClassifier__boosting_type':['gbdt''],
   'LGBMClassifier__max_depth':[-1],
   'LGBMClassifier__n_estimators':[100]
}
```

From above all the model parameters, we have found the best parameters for each training model

Here, below is the accuracy screenshot for all the models which are trained:

```
experimentLog
```

	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Best Params
0	Baseline logistic_regression with 334 inputs	{'lr__C': [1, 0.1, 0.01], 'lr__penalty': ['l1'...	91.95%	91.96%	91.84%	0.082689	0.029659	{'memory': None, 'steps': [('lr', LogisticRegr...
1	Baseline decision_tree with 334 inputs	{'dt__max_depth': [10, 15, 20], 'dt__splitter'...	92.33%	91.63%	91.47%	0.162742	0.066262	{'memory': None, 'steps': [('dt', DecisionTree...
2	Baseline random_forest with 334 inputs	{'rf__max_depth': [5, 10], 'rf__n_estimators':...	91.97%	91.96%	91.84%	2.469823	1.019843	{'memory': None, 'steps': [('rf', RandomForest...
3	Baseline xgboost with 334 inputs	{'xgb__min_child_weight': [19.0, 25.0], 'xgb__...	91.94%	91.96%	91.84%	0.094252	0.024308	{'memory': None, 'steps': [('xgb', XGBClassifi...
4	Baseline lightgbm with 334 inputs	{'LGBMClassifier__boosting_type': ['gbdt', 'go...	92.28%	91.99%	91.89%	0.066475	0.023907	{'memory': None, 'steps': [('LGBMClassifier', ...

6) Neural network

Accuracy:

Train Accuracy = 75.6%
Validation Accuracy = 74.7%
Test Accuracy = 75.3%

Best Neural network Parameters :
Activation function: CELU
Epoch: 1000
Hidden_layers: 10
Batch_size: 100
Loss function: MSELoss

	act_fn	epoch	hidden_layers	batch_size	loss_fn	learning_rate	Model	ROC_AUC_TRAIN	ROC_AUC_TEST	ROC_AUC_VALID
26	CELU(alpha=1.0)	1000	10	100	MSELoss()	0.10	[Linear(in_features=327, out_features=10, bias...	0.758773	0.753406	0.747729
30	CELU(alpha=1.0)	1000	20	100	MSELoss()	0.10	[Linear(in_features=327, out_features=20, bias...	0.758366	0.752126	0.747559
2	ReLU()	1000	10	100	MSELoss()	0.10	[Linear(in_features=327, out_features=10, bias...	0.753172	0.744697	0.739545
6	ReLU()	1000	20	100	MSELoss()	0.10	[Linear(in_features=327, out_features=20, bias...	0.753764	0.745593	0.735856
31	CELU(alpha=1.0)	1000	20	100	MSELoss()	0.01	[Linear(in_features=327, out_features=20, bias...	0.737027	0.735019	0.729359
27	CELU(alpha=1.0)	1000	10	100	MSELoss()	0.01	[Linear(in_features=327, out_features=10, bias...	0.737097	0.733433	0.729082
18	Sigmoid()	1000	10	100	MSELoss()	0.10	[Linear(in_features=327, out_features=10, bias...	0.721396	0.725235	0.712639
22	Sigmoid()	1000	20	100	MSELoss()	0.10	[Linear(in_features=327, out_features=20, bias...	0.717058	0.719671	0.707915
3	ReLU()	1000	10	100	MSELoss()	0.01	[Linear(in_features=327, out_features=10, bias...	0.703757	0.703854	0.698070
7	ReLU()	1000	20	100	MSELoss()	0.01	[Linear(in_features=327, out_features=20, bias...	0.696450	0.698363	0.687185
10	LogSigmoid()	1000	10	100	MSELoss()	0.10	[Linear(in_features=327, out_features=10, bias...	0.683227	0.689331	0.673940
14	LogSigmoid()	1000	20	100	MSELoss()	0.10	[Linear(in_features=327, out_features=20, bias...	0.665940	0.675992	0.661024
23	Sigmoid()	1000	20	100	MSELoss()	0.01	[Linear(in_features=327, out_features=20, bias...	0.650914	0.650846	0.653459
11	LogSigmoid()	1000	10	100	MSELoss()	0.01	[Linear(in_features=327, out_features=10, bias...	0.625054	0.636910	0.624666
19	Sigmoid()	1000	10	100	MSELoss()	0.01	[Linear(in_features=327, out_features=10, bias...	0.607068	0.606676	0.602565
29	CELU(alpha=1.0)	1000	20	100	CrossEntropyLoss()	0.01	[Linear(in_features=327, out_features=20, bias...	0.575723	0.576083	0.578610
5	ReLU()	1000	20	100	CrossEntropyLoss()	0.01	[Linear(in_features=327, out_features=20, bias...	0.561847	0.565762	0.568244

Impact of newly added features on the model

Here we have calculated the correlation of all the newly added features to find the importance of each feature that will affect the model accuracy

	feature	importance
177	Credit/Income	0.001121
178	Annuity/Income	0.001636
179	Employed/Birth	0.001568
180	Flag_Greater_32	0.000284
181	Flag_Employment_Greater_5	0.000268
182	Flag_Income_Greater_Credit	0.000028
183	DAYS_BIRTH_power_2	0.001738
184	DAYS_BIRTH_power_3	0.001747
185	REGION_RATING_CLIENT_W_CITY_power_2	0.000503
186	REGION_RATING_CLIENT_W_CITY_power_3	0.000454
187	REGION_RATING_CLIENT_power_2	0.000307
188	REGION_RATING_CLIENT_power_3	0.000399
189	EXT_SOURCE_3_power_2	0.009591
190	EXT_SOURCE_3_power_3	0.011495
191	EXT_SOURCE_2_power_2	0.009941
192	EXT_SOURCE_2_power_3	0.009492
193	EXT_SOURCE_1_power_2	0.003049
194	EXT_SOURCE_1_power_3	0.003620

We can observe that polynomial features impact the most as input variables to a power can help to better expose the important relationships between input variables and the target variable. Also, features like EXT_SOURCE_3_power_2 , EXT_SOURCE_2_power_2, EXT_SOURCE_2_power_3 have more impact on the trained model.

Feature Selection

We have included all the features except the ones which have correlation values as 0 which are less important because the classification result does not affect when these features are included in the dataset while training the model.
Zero importance features as below:

- FLAG_MOBIL
- FLAG_DOCUMENT_4
- FLAG_DOCUMENT_7
- FLAG_DOCUMENT_10
- FLAG_DOCUMENT_12
- FLAG_DOCUMENT_13
- FLAG_DOCUMENT_21

Comparing the results using the above techniques and selecting the best model from them.

Results

- After performing a baseline submission, we were getting an accuracy of 56% on our Kaggle submission.
- We performed feature engineering manually, and added **19 new features** and found out their correlation with the TARGET variable.
- Also, One Hot Encoding was performed, on the categorical features and ended up with **139 new features.**
- By performing feature selection using Random Forest algorithm, **we removed 11 feature columns that had 0 importance.**

- <u>Model Observations and Results</u>**:**
  As compared to the baseline model, we observed an decrease in accuracy after increasing the max depth because the splitter parameter was added.

- Here is the screenshot of the experiment log which shows the roc_auc accuracies before feature engineering for Logistic Regression, Decision Tree and Random Forest classifier models:

```
experimentLog
```

	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Best Params
0	Baseline logistic_regression with 334 inputs	{'lr__C': [1, 0.1, 0.01], 'lr__penalty': ['l1'...	91.95%	91.96%	91.84%	0.082689	0.029659	{'memory': None, 'steps': [('lr', LogisticRegr...
1	Baseline decision_tree with 334 inputs	{'dt__max_depth': [10, 15, 20], 'dt__splitter'...	92.33%	91.63%	91.47%	0.162742	0.066262	{'memory': None, 'steps': [('dt', DecisionTree...
2	Baseline random_forest with 334 inputs	{'rf__max_depth': [5, 10], 'rf__n_estimators':...	91.97%	91.96%	91.84%	2.469823	1.019843	{'memory': None, 'steps': [('rf', RandomForest...
3	Baseline xgboost with 334 inputs	{'xgb__min_child_weight': [19.0, 25.0], 'xgb__...	91.94%	91.96%	91.84%	0.094252	0.024308	{'memory': None, 'steps': [('xgb', XGBClassifi...
4	Baseline lightgbm with 334 inputs	{'LGBMClassifier__boosting_type': ['gbdt', 'go...	92.28%	91.99%	91.89%	0.066475	0.023907	{'memory': None, 'steps': [('LGBMClassifier', ...

[39]:

```
for best_param in experimentLog['Best Params'].tolist():
    print(best_param['steps'])
```

```
[('lr', LogisticRegression(C=1, solver='saga'))]
[('dt', DecisionTreeClassifier(max_depth=10, splitter='random'))]
[('rf', RandomForestClassifier(max_depth=10))]
[('xgb', XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.1,
              enable_categorical=False, eta=0.025, eval_metric='auc',
              gamma=0.65, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.0250000004,
              max_delta_step=1.8, max_depth=8, min_child_weight=25.0,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=96, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=0.8,
              tree_method='exact', validate_parameters=1, verbosity=None))]
[('LGBMClassifier', LGBMClassifier(max_depth=8, random_state=0))]
```

- After performing feature engineering, hyperparameter tuning and adding new models i.e XGBoost and LGBM, the screenshot of our experiment log is given below.

```
1 experimentLog
```

	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Best Params
0	Baseline logistic_regression with 334 inputs	{'lr__C': [1, 0.1, 0.01], 'lr__penalty': ['l1'...	91.95%	91.96%	91.84%	0.087670	0.042359	{'memory': None, 'steps': [('lr', LogisticRegr...
1	Baseline logistic_regression with 334 inputs	{'lr__C': [1, 0.1, 0.01], 'lr__penalty': ['l1'...	91.95%	91.96%	91.84%	0.082613	0.044088	{'memory': None, 'steps': [('lr', LogisticRegr...
2	Baseline logistic_regression with 334 inputs	{'lr__C': [1, 0.1, 0.01], 'lr__penalty': ['l1'...	91.95%	91.96%	91.84%	0.072386	0.035532	{'memory': None, 'steps': [('lr', LogisticRegr...
3	Baseline decision_tree with 334 inputs	{'dt__max_depth': [10, 15, 20], 'dt__splitter'...	92.38%	91.67%	91.58%	0.147465	0.065180	{'memory': None, 'steps': [('dt', DecisionTree...
4	Baseline random_forest with 334 inputs	{'rf__max_depth': [5, 10], 'rf__n_estimators':...	91.98%	91.97%	91.84%	2.507257	1.020463	{'memory': None, 'steps': [('rf', RandomForest...
5	Baseline xgboost with 334 inputs	{'xgb__min_child_weight': [19.0], 'xgb__object...	91.99%	91.98%	91.86%	0.046852	0.023269	{'memory': None, 'steps': [('xgb', XGBClassifi...
6	Baseline lightgbm with 334 inputs	{'LGBMClassifier__boosting_type': ['gbdt'], 'L...	92.28%	92.00%	91.89%	0.061664	0.028334	{'memory': None, 'steps': [('LGBMClassifier', ...

- Data leakage analysis found 3 categorical features which are not in the test data. However, after removing the identified categorical variables model accuracy had no effect(as seen in the Kaggle submission below).
- Now common data points were found between test and train datasets.

The highest training test and validation accuracy were obtained on the **LGBM (Light Gradient Boost Model) which was 92.28, 92.00, 91.86 for train validation and test model,** and hence we chose the same to perform our **Kaggle submission which gave us an amazing result of 76.6%** *(a slight improvement from phase 3).* We also submitted the prediction of the neural network and got an accuracy of **74.6% on Kaggle**. Here is the screenshot of the latest submission for your reference.

Submission and Description	Private Score	Public Score	Selected
sub_nn.csv Complete (after deadline) · 16h ago · nn-1	0.73842	0.74611	☐
submission.csv Complete (after deadline) · 19h ago · After data leakage changes	0.76456	0.76619	☐
submission.csv Complete (after deadline) · 1d ago · After training XGBoost for 6 hrs	0.76456	0.76619	☐
submission (1).csv Complete (after deadline) · 5d ago	0.76501	0.76475	☐
submission.csv Complete (after deadline) · 5d ago · phase 3 sub 1	0.50624	0.50256	☐
submission.csv Complete (after deadline) · 11d ago · phase 2	0.54766	0.56682	☐

Conclusion

The ability to repay loans is essential in today's world. We created a machine learning model to predict whether or not the loan borrower will be able to repay the loan in order to solve this problem. For this, we conducted multiple experiments to select the best possible parameters for machine learning and neural network models. To select the best model, we performed data cleaning and Exploratory Data Analysis on original data as well as on new features by performing manual feature engineering and using polynomial features. We also ensured that there was no data leakage in the training dataset. Then we performed experiments using multiple models and found that the Light gradient boost model(LGBM) gave the best accuracy of 92% when we split the train data into train, test, and validation sets. After doing the Kaggle Submission, we achieved a 76% public score. We also trained a neural network model which gave us a Kaggle public score of 74%. The LGBM is clearly outperforming the neural network. In the future, we will try to streamline the process and experiment with new machine learning models and different neural network architectures to make an attempt to increase the accuracy of the dataset. By doing this, we will improvise a loan-lending process that will empower the client to predict the defaulter more accurately.

Bibliography

1. Pandas - https://pandas.pydata.org/
2. Sklearn - https://scikit-learn.org/stable/index.html
3. Logistic Regression - https://en.wikipedia.org/wiki/Logistic_regression
4. Seaborn - https://seaborn.pydata.org
5. Decision Tree - https://en.wikipedia.org/wiki/Decision_tree

6. Normal Distribution - https://en.wikipedia.org/wiki/Normal_distribution
7. Kaggle - https://www.kaggle.com/competitions/home-credit-default-risk
8. Correlation Analysis - https://dergipark.org.tr/en/pub/jader/issue/49634/555979
9. AUC/ROC -https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5?gi=f2d6c64870f2
10. XGBoost Model - https://xgboost.readthedocs.io/en/stable/python/python_api.html
11. Random Forest - https://www.scirp.org/journal/paperinformation.aspx?paperid=65359
12. LGBM - https://lightgbm.readthedocs.io/en/v3.3.2/
13. Polynomial features - https://machinelearningmastery.com/polynomial-features-transforms-for-machine-learning/
14. CELU: https://towardsdatascience.com/google-continuously-differentiable-exponential-linear-units-with-interactive-code-manual-back-fcbe7f84e79
15. Neural Networks: *https://drive.google.com/file/d/1Od9rgGkOaqLUJSrGKREPz3n5-R3gLuj6/view?usp=sharing*
16. Tensorboard: https://www.tensorflow.org/tensorboard