Home Credit Default Risk (HCDR)

The course project is based on the Home Credit Default Risk (HCDR) Kaggle Competition. The goal of this project is to predict whether or not a client will repay a loan. In order to make sure that people who struggle to get loans due to insufficient or non-existent credit histories have a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

Some of the challenges

1. Dataset size
    - (688 meg uncompressed) with millions of rows of data
    - 2.71 Gig of data uncompressed

- Dealing with missing data
- Imbalanced datasets
- Summarizing transaction data

Notebook Index:

Phase 1

Kaggle API setup

Kaggle is a Data Science Competition Platform which shares a lot of datasets. In the past, it was troublesome to submit your result as your have to go through the console in your browser and drag your files there. Now you can interact with Kaggle via the command line. E.g.,

```
! kaggle competitions files home-credit-default-risk
```

It is quite easy to setup, it takes me less than 15 minutes to finish a submission.

1. Install library

- Create a API Token (edit your profile on Kaggle.com); this produces `kaggle.json` file
- Put your JSON `kaggle.json` in the right place
- Access competition files; make submissions via the command (see examples below)
- Submit result

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

In [1]:
```
# !pip install kaggle
```

In [2]:
```
!pwd
```

```
/geode2/home/u060/nmalpani/Carbonate/miniconda3/bin
```

In [3]:
```
!mkdir ~/.kaggle
!cp /root/shared/Downloads/kaggle.json ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json
```

```
mkdir: cannot create directory ‘/N/u/nmalpani/Carbonate/.kaggle’: File exists
cp: cannot stat ‘/root/shared/Downloads/kaggle.json’: Permission denied
chmod: cannot access ‘/N/u/nmalpani/Carbonate/.kaggle/kaggle.json’: No such file
or directory
```

In [4]:
```
! kaggle competitions files home-credit-default-risk
```

```
/bin/bash: kaggle: command not found
```

Dataset and how to download

Back ground Home Credit Group

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

Home Credit Group

Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

Background on the dataset

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

The company operates in 14 countries (including United States, Russia, Kazahstan, Belarus, China, India) and focuses on lending primarily to people with little or no credit history which will either not obtain loans or became victims of untrustworthly lenders.

Home Credit group has over 29 million customers, total assests of 21 billions Euro, over 160 millions loans, with the majority in Asia and and almost half of them in China (as of 19-05-2018).

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

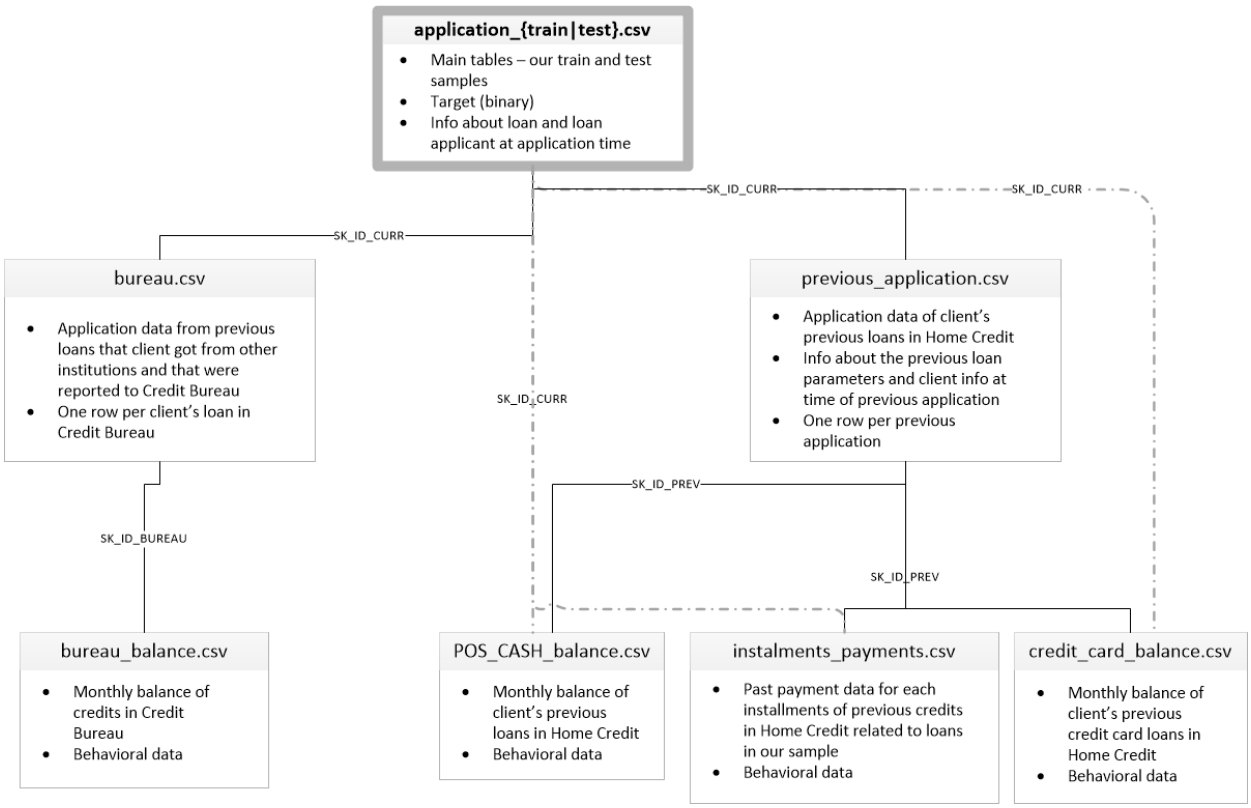
Data files overview

There are 7 different sources of data:

- **application_train/application_test:** the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating **0: the loan was repaid** or **1: the loan was not repaid**. The target variable defines if the client had payment difficulties meaning he/she had late payment more than X days on at least one of the first Y installments of the loan. Such case is marked as 1 while other all other cases as 0.
- **bureau:** data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance:** monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.

- **previous_application:** previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE:** monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment:** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

In [5]:
```
# ![alt](home_credit.png "Home credit")
```



Downloading the files via Kaggle API

Create a base directory:

```
DATA_DIR = "../../../Data/home-credit-default-risk"   #same level as course repo in the data directory
```

Please download the project data files and data dictionary and unzip them using either of the following approaches:

1. Click on the `Download` button on the following Data Webpage and unzip the zip file to the `BASE_DIR`
2. If you plan to use the Kaggle API, please use the following steps.

In [6]:
```
DATA_DIR = "Data/home-credit-default-risk"   #same level as course repo in the d
#DATA_DIR = os.path.join('./ddddd/')
!mkdir $DATA_DIR
```

```
mkdir: cannot create directory ‘Data/home-credit-default-risk’: File exists
```

In [7]:
```
!ls -l $DATA_DIR
```

```
total 2622080
-rw-rw-r-- 1 nmalpani nmalpani  26567651 Dec 11  2019 application_test.csv
-rw-rw-r-- 1 nmalpani nmalpani 166133370 Dec 11  2019 application_train.csv
-rw-rw-r-- 1 nmalpani nmalpani 375592889 Dec 11  2019 bureau_balance.csv
-rw-rw-r-- 1 nmalpani nmalpani 170016717 Dec 11  2019 bureau.csv
-rw-rw-r-- 1 nmalpani nmalpani 424582605 Dec 11  2019 credit_card_balance.csv
-rw-rw-r-- 1 nmalpani nmalpani     37383 Dec 11  2019 HomeCredit_columns_descrip
tion.csv
-rw-rw-r-- 1 nmalpani nmalpani 723118349 Dec 11  2019 installments_payments.csv
-rw-rw-r-- 1 nmalpani nmalpani 392703158 Dec 11  2019 POS_CASH_balance.csv
-rw-rw-r-- 1 nmalpani nmalpani 404973293 Dec 11  2019 previous_application.csv
-rw-rw-r-- 1 nmalpani nmalpani    536202 Dec 11  2019 sample_submission.csv
```

In [8]:
```
# ! kaggle competitions download home-credit-default-risk -p $DATA_DIR
```

Imports

In [9]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import zipfile
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [10]:
```
%matplotlib inline
```

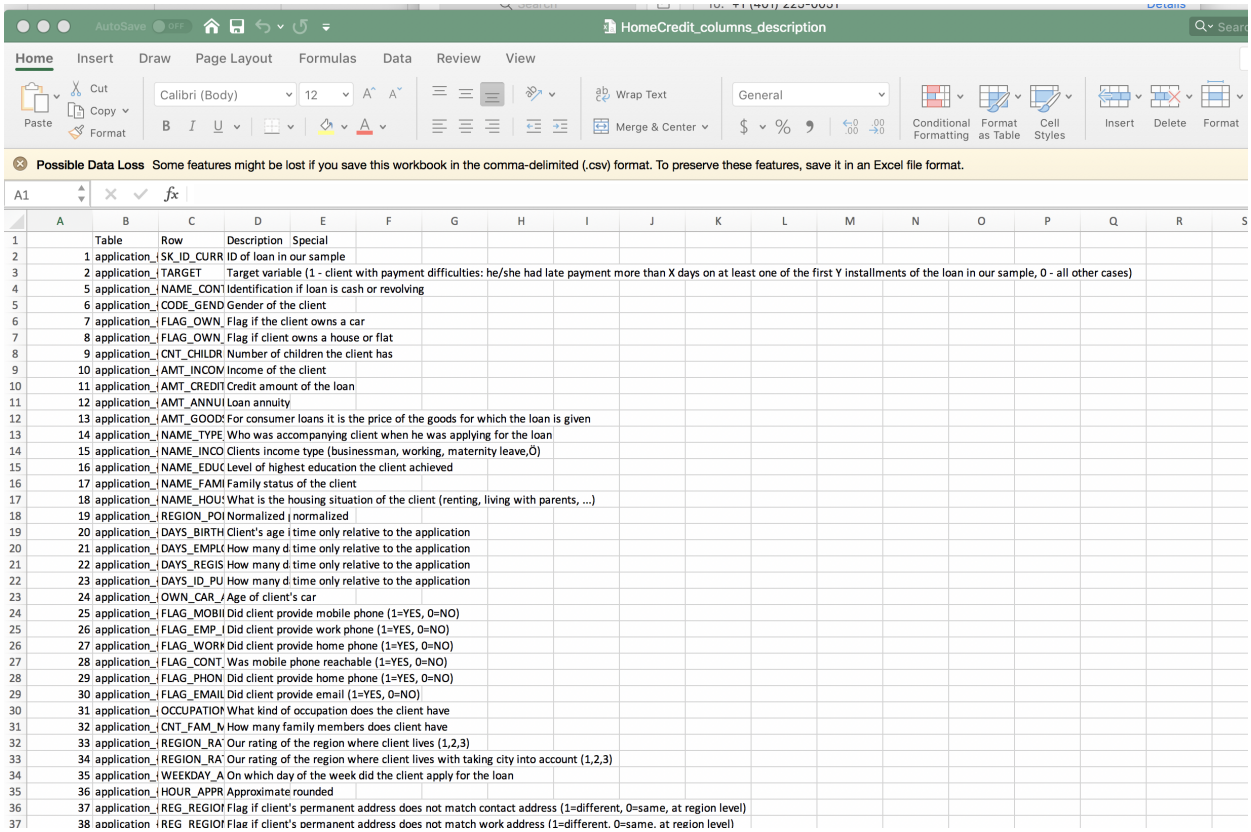
In [11]:

```
unzippingReq = False
if unzippingReq: #please modify this code
    zip_ref = zipfile.ZipFile('application_train.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('application_test.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('bureau_balance.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('bureau.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('credit_card_balance.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('installments_payments.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('POS_CASH_balance.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('previous_application.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
```

Data files overview

Data Dictionary

As part of the data download comes a Data Dictionary. It named `HomeCredit_columns_description.csv`



HomeCredit_columns_description

Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

	A	B	C	D	E
1		Table	Row	Description	Special
2	1	application_	SK_ID_CURR	ID of loan in our sample	
3	2	application_	TARGET	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)	
4	5	application_	NAME_CONT	Identification if loan is cash or revolving	
5	6	application_	CODE_GEND	Gender of the client	
6	7	application_	FLAG_OWN_	Flag if the client owns a car	
7	8	application_	FLAG_OWN_	Flag if client owns a house or flat	
8	9	application_	CNT_CHILDR	Number of children the client has	
9	10	application_	AMT_INCOM	Income of the client	
10	11	application_	AMT_CREDIT	Credit amount of the loan	
11	12	application_	AMT_ANNUI	Loan annuity	
12	13	application_	AMT_GOODS	For consumer loans it is the price of the goods for which the loan is given	
13	14	application_	NAME_TYPE	Who was accompanying client when he was applying for the loan	
14	15	application_	NAME_INCO	Clients income type (businessman, working, maternity leave,Ó)	
15	16	application_	NAME_EDUC	Level of highest education the client achieved	
16	17	application_	NAME_FAMI	Family status of the client	
17	18	application_	NAME_HOUS	What is the housing situation of the client (renting, living with parents, ...)	
18	19	application_	REGION_POI	Normalized p	normalized
19	20	application_	DAYS_BIRTH	Client's age i	time only relative to the application
20	21	application_	DAYS_EMPL(	How many d	time only relative to the application
21	22	application_	DAYS_REGIS	How many d	time only relative to the application
22	23	application_	DAYS_ID_PU	How many d	time only relative to the application
23	24	application_	OWN_CAR_A	Age of client's car	
24	25	application_	FLAG_MOBIL	Did client provide mobile phone (1=YES, 0=NO)	
25	26	application_	FLAG_EMP_I	Did client provide work phone (1=YES, 0=NO)	
26	27	application_	FLAG_WORK	Did client provide home phone (1=YES, 0=NO)	
27	28	application_	FLAG_CONT_	Was mobile phone reachable (1=YES, 0=NO)	
28	29	application_	FLAG_PHON	Did client provide home phone (1=YES, 0=NO)	
29	30	application_	FLAG_EMAIL	Did client provide email (1=YES, 0=NO)	
30	31	application_	OCCUPATION	What kind of occupation does the client have	
31	32	application_	CNT_FAM_M	How many family members does client have	
32	33	application_	REGION_RA	Our rating of the region where client lives (1,2,3)	
33	34	application_	REGION_RA	Our rating of the region where client lives with taking city into account (1,2,3)	
34	35	application_	WEEKDAY_A	On which day of the week did the client apply for the loan	
35	36	application_	HOUR_APPR	Approximate rounded	
36	37	application_	REG_REGIO	Flag if client's permanent address does not match contact address (1=different, 0=same, at region level)	
37	38	application_	REG_REGIO	Flag if client's permanent address does not match work address (1=different, 0=same, at region level)	

Application train

In [12]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import zipfile
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import warnings
warnings.filterwarnings('ignore')

def load_data(in_path, name):
    df = pd.read_csv(in_path)
    print(f"{name}: shape is {df.shape}")
    print(df.info())
    display(df.head(5))
    return df

datasets={}  # lets store the datasets in a dictionary so we can keep track of t
```

```
ds_name = 'application_train'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)

datasets['application_train'].shape
```

```
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_
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	

5 rows × 122 columns

Out[12]:
```
(307511, 122)
```

Application test

- **application_train/application_test:** the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating **0: the loan was repaid** or **1: the loan was not repaid**. The target variable defines if the client had payment difficulties meaning he/she had late payment more than X days on at least one of the first Y installments of the loan. Such case is marked as 1 while other all other cases as 0.

In [13]:
```
ds_name = 'application_test'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
```

```
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
0	100001	Cash loans	F	N	Y
1	100005	Cash loans	M	N	Y
2	100013	Cash loans	M	Y	Y
3	100028	Cash loans	F	N	Y

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY
4	100038	Cash loans	M	Y	N

5 rows × 121 columns

The application dataset has the most information about the client: Gender, income, family status, education ...

The Other datasets

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

In [14]:
```
%%time
ds_names = ("application_train", "application_test", "bureau","bureau_balance","
            "previous_application","POS_CASH_balance")

for ds_name in ds_names:
    datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_n
```

```
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_
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	

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
0	100001	Cash loans	F	N	Y
1	100005	Cash loans	M	N	Y
2	100013	Cash loans	M	Y	Y
3	100028	Cash loans	F	N	Y
4	100038	Cash loans	M	Y	N

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_DA
0	215354	5714462	Closed	currency 1	-497	
1	215354	5714463	Active	currency 1	-208	

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DA
2	215354	5714464	Active	currency 1	-203	
3	215354	5714465	Active	currency 1	-203	
4	215354	5714466	Active	currency 1	-629	

```
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
0	2562384	378907	-6	56.970	135000
1	2582071	363914	-1	63975.555	45000
2	1740877	371185	-7	31815.225	450000
3	1389973	337855	-4	236572.110	225000
4	1891521	126868	-1	453919.455	450000

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_IN
0	1054186	161674	1.0	6	
1	1330831	151639	0.0	34	
2	2085231	193053	2.0	1	
3	2452527	199697	1.0	3	
4	2714724	167756	1.0	2	

```
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
None
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_
0	2030495	271877	Consumer loans	1730.430	17145.0	
1	2802425	108129	Cash loans	25188.615	607500.0	6
2	2523466	122040	Cash loans	15060.735	112500.0	1
3	2819243	176158	Cash loans	47041.335	450000.0	4
4	1784265	202054	Cash loans	31924.395	337500.0	4

5 rows × 37 columns

```
POS_CASH_balance: shape is (10001358, 8)
<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
None
```

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

```
CPU times: user 33.8 s, sys: 15.5 s, total: 49.2 s
Wall time: 49.3 s
```

In [15]:
```
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,}, {datasets[
```

```
dataset application_train       : [    307,511, 122]
dataset application_test        : [     48,744, 121]
dataset bureau                  : [  1,716,428, 17]
dataset bureau_balance          : [ 27,299,925, 3]
dataset credit_card_balance     : [  3,840,312, 23]
dataset installments_payments   : [ 13,605,401, 8]
dataset previous_application    : [  1,670,214, 37]
dataset POS_CASH_balance        : [ 10,001,358, 8]
```

Exploratory Data Analysis

Summary of Application train

In [16]:
```
datasets["application_train"].info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 122 columns):
 #    Column                       Dtype
---   ------                       -----
 0    SK_ID_CURR                   int64
 1    TARGET                       int64
 2    NAME_CONTRACT_TYPE           object
 3    CODE_GENDER                  object
 4    FLAG_OWN_CAR                 object
 5    FLAG_OWN_REALTY              object
 6    CNT_CHILDREN                 int64
 7    AMT_INCOME_TOTAL             float64
 8    AMT_CREDIT                   float64
 9    AMT_ANNUITY                  float64
 10   AMT_GOODS_PRICE              float64
 11   NAME_TYPE_SUITE              object
 12   NAME_INCOME_TYPE             object
 13   NAME_EDUCATION_TYPE          object
 14   NAME_FAMILY_STATUS           object
 15   NAME_HOUSING_TYPE            object
 16   REGION_POPULATION_RELATIVE   float64
```

17	DAYS_BIRTH	int64
18	DAYS_EMPLOYED	int64
19	DAYS_REGISTRATION	float64
20	DAYS_ID_PUBLISH	int64
21	OWN_CAR_AGE	float64
22	FLAG_MOBIL	int64
23	FLAG_EMP_PHONE	int64
24	FLAG_WORK_PHONE	int64
25	FLAG_CONT_MOBILE	int64
26	FLAG_PHONE	int64
27	FLAG_EMAIL	int64
28	OCCUPATION_TYPE	object
29	CNT_FAM_MEMBERS	float64
30	REGION_RATING_CLIENT	int64
31	REGION_RATING_CLIENT_W_CITY	int64
32	WEEKDAY_APPR_PROCESS_START	object
33	HOUR_APPR_PROCESS_START	int64
34	REG_REGION_NOT_LIVE_REGION	int64
35	REG_REGION_NOT_WORK_REGION	int64
36	LIVE_REGION_NOT_WORK_REGION	int64
37	REG_CITY_NOT_LIVE_CITY	int64
38	REG_CITY_NOT_WORK_CITY	int64
39	LIVE_CITY_NOT_WORK_CITY	int64
40	ORGANIZATION_TYPE	object
41	EXT_SOURCE_1	float64
42	EXT_SOURCE_2	float64
43	EXT_SOURCE_3	float64
44	APARTMENTS_AVG	float64
45	BASEMENTAREA_AVG	float64
46	YEARS_BEGINEXPLUATATION_AVG	float64
47	YEARS_BUILD_AVG	float64
48	COMMONAREA_AVG	float64
49	ELEVATORS_AVG	float64
50	ENTRANCES_AVG	float64
51	FLOORSMAX_AVG	float64
52	FLOORSMIN_AVG	float64
53	LANDAREA_AVG	float64
54	LIVINGAPARTMENTS_AVG	float64
55	LIVINGAREA_AVG	float64
56	NONLIVINGAPARTMENTS_AVG	float64
57	NONLIVINGAREA_AVG	float64
58	APARTMENTS_MODE	float64
59	BASEMENTAREA_MODE	float64
60	YEARS_BEGINEXPLUATATION_MODE	float64
61	YEARS_BUILD_MODE	float64
62	COMMONAREA_MODE	float64
63	ELEVATORS_MODE	float64
64	ENTRANCES_MODE	float64
65	FLOORSMAX_MODE	float64
66	FLOORSMIN_MODE	float64
67	LANDAREA_MODE	float64
68	LIVINGAPARTMENTS_MODE	float64
69	LIVINGAREA_MODE	float64
70	NONLIVINGAPARTMENTS_MODE	float64
71	NONLIVINGAREA_MODE	float64
72	APARTMENTS_MEDI	float64
73	BASEMENTAREA_MEDI	float64
74	YEARS_BEGINEXPLUATATION_MEDI	float64
75	YEARS_BUILD_MEDI	float64
76	COMMONAREA_MEDI	float64

```
 77   ELEVATORS_MEDI               float64
 78   ENTRANCES_MEDI               float64
 79   FLOORSMAX_MEDI               float64
 80   FLOORSMIN_MEDI               float64
 81   LANDAREA_MEDI                float64
 82   LIVINGAPARTMENTS_MEDI        float64
 83   LIVINGAREA_MEDI              float64
 84   NONLIVINGAPARTMENTS_MEDI     float64
 85   NONLIVINGAREA_MEDI           float64
 86   FONDKAPREMONT_MODE           object
 87   HOUSETYPE_MODE               object
 88   TOTALAREA_MODE               float64
 89   WALLSMATERIAL_MODE           object
 90   EMERGENCYSTATE_MODE          object
 91   OBS_30_CNT_SOCIAL_CIRCLE     float64
 92   DEF_30_CNT_SOCIAL_CIRCLE     float64
 93   OBS_60_CNT_SOCIAL_CIRCLE     float64
 94   DEF_60_CNT_SOCIAL_CIRCLE     float64
 95   DAYS_LAST_PHONE_CHANGE       float64
 96   FLAG_DOCUMENT_2              int64
 97   FLAG_DOCUMENT_3              int64
 98   FLAG_DOCUMENT_4              int64
 99   FLAG_DOCUMENT_5              int64
 100  FLAG_DOCUMENT_6              int64
 101  FLAG_DOCUMENT_7              int64
 102  FLAG_DOCUMENT_8              int64
 103  FLAG_DOCUMENT_9              int64
 104  FLAG_DOCUMENT_10             int64
 105  FLAG_DOCUMENT_11             int64
 106  FLAG_DOCUMENT_12             int64
 107  FLAG_DOCUMENT_13             int64
 108  FLAG_DOCUMENT_14             int64
 109  FLAG_DOCUMENT_15             int64
 110  FLAG_DOCUMENT_16             int64
 111  FLAG_DOCUMENT_17             int64
 112  FLAG_DOCUMENT_18             int64
 113  FLAG_DOCUMENT_19             int64
 114  FLAG_DOCUMENT_20             int64
 115  FLAG_DOCUMENT_21             int64
 116  AMT_REQ_CREDIT_BUREAU_HOUR   float64
 117  AMT_REQ_CREDIT_BUREAU_DAY    float64
 118  AMT_REQ_CREDIT_BUREAU_WEEK   float64
 119  AMT_REQ_CREDIT_BUREAU_MON    float64
 120  AMT_REQ_CREDIT_BUREAU_QRT    float64
 121  AMT_REQ_CREDIT_BUREAU_YEAR   float64
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

In [17]:
```
datasets["application_train"].shape
```

Out[17]:
```
(307511, 122)
```

In [18]:
```
datasets["application_train"].describe() #numerical only features
```

Out[18]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	3074

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	165
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	249
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	345
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258

8 rows × 106 columns

In [19]:
```
datasets["application_test"].describe() #numerical only features
```

Out[19]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	

8 rows × 105 columns

In [20]:
```
datasets["application_test"].shape
```

Out[20]:
```
(48744, 121)
```

In [21]:
```
datasets["application_train"].describe(include='all') #look at all categorical a
```

Out[21]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
count	307511.000000	307511.000000	307511	307511	307511
unique	NaN	NaN	2	3	2
top	NaN	NaN	Cash loans	F	N
freq	NaN	NaN	278232	202448	202924
mean	278180.518577	0.080729	NaN	NaN	NaN
std	102790.175348	0.272419	NaN	NaN	NaN
min	100002.000000	0.000000	NaN	NaN	NaN

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
25%	189145.500000	0.000000	NaN	NaN	NaN
50%	278202.000000	0.000000	NaN	NaN	NaN
75%	367142.500000	0.000000	NaN	NaN	NaN
max	456255.000000	1.000000	NaN	NaN	NaN

11 rows × 122 columns

Data Description

- In this dataset we have a total of 122 columns.
- There are a lot of columns which have missing values in the dataset.
- As we can already see that there are so many missing values in the top 10 rows itself.
- We'll have to figure out a way to deal with thid!

In [22]:
```
datasets['application_train'].corrwith(datasets['application_train']["TARGET"]).
```

Out[22]:
```
TARGET                        1.000000
DAYS_BIRTH                    0.078239
REGION_RATING_CLIENT_W_CITY   0.060893
REGION_RATING_CLIENT          0.058899
DAYS_LAST_PHONE_CHANGE        0.055218
                                ...
FLOORSMAX_AVG                -0.044003
DAYS_EMPLOYED                -0.044932
EXT_SOURCE_1                 -0.155317
EXT_SOURCE_2                 -0.160472
EXT_SOURCE_3                 -0.178919
Length: 106, dtype: float64
```

Correlation with the Target Variable

As we an see above, here we have tried to find out the correlation of various indepenndent features in the dataset with the dependent variable or the target variable and tried understanding the feature correlation. We see that there are very few columns which are related to the target variable, thus making it all the. ore difficult to draw our predictions. Let's see how we proceed with it further!

In [23]:
```
percent = (datasets["application_train"].isnull().sum()/datasets["application_tr
sum_missing = datasets["application_train"].isna().sum().sort_values(ascending =
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys
missing_application_train_data.head(20)
```

Out[23]:

	Percent	Train Missing Count
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865

	Percent	Train Missing Count
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

In [24]:
```
datasets['application_train'].isnull().sum()
```

Out[24]:
```
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 Analysis

- There are so many columns in the dataset whoch have large amount of missing values and we need to deal with those in some way. We'll figure it out in the path ahead.

Missingno is a Python library that provides the ability to understand the distribution of missing values through informative visualizations. The visualizations can be in the form of heat maps or

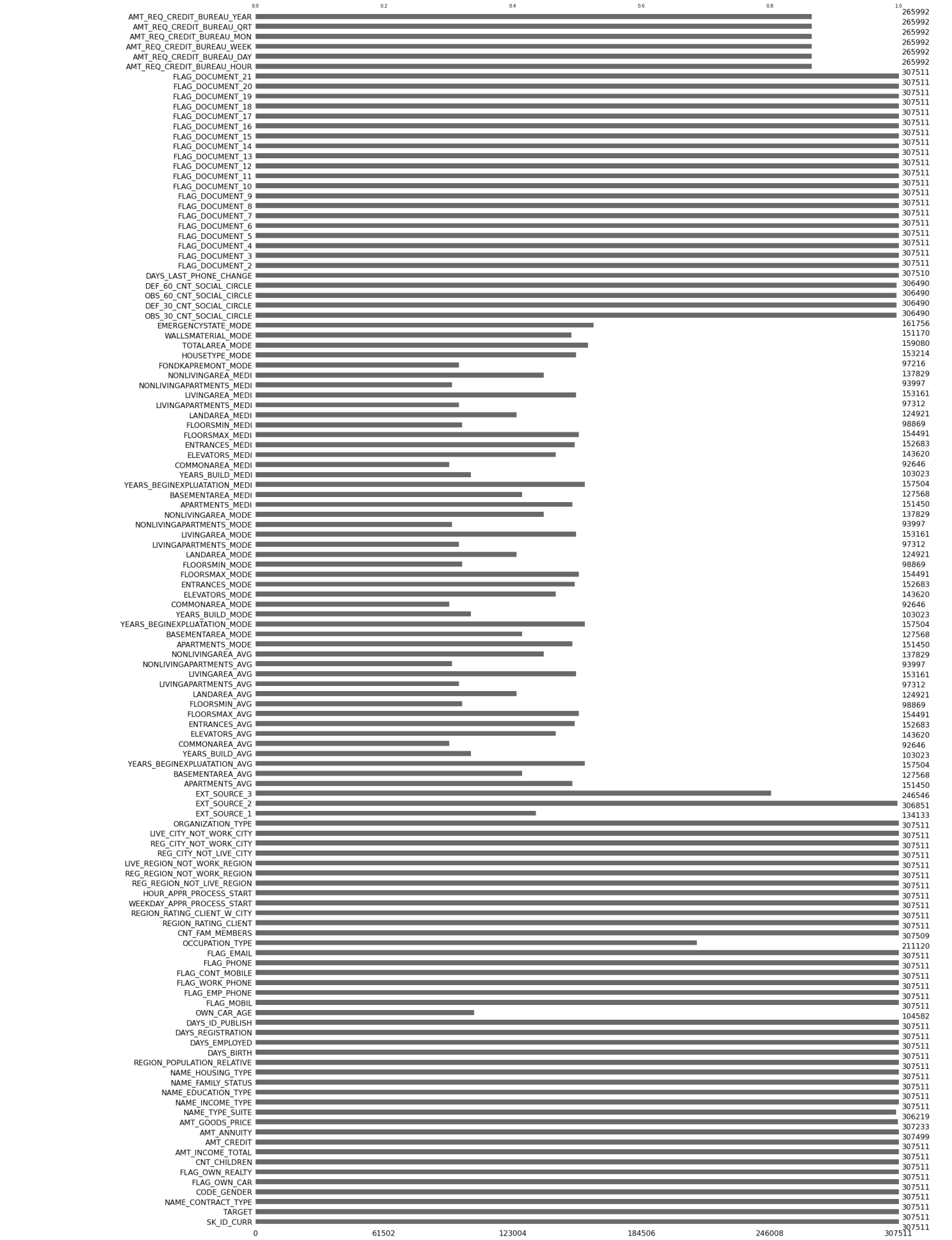
bar charts. With this library, it is possible to observe where the missing values have occurred and to check the correlation of the columns containing the missing with the target column.

In [25]:
```
df = datasets['application_train'].copy()
```

In [26]:
```
import missingno as msno
# msno.matrix(df)
```

In [27]:
```
msno.bar(datasets['application_train'])
```

Out[27]:
```
<AxesSubplot:>
```


0.0
0.2
0.4
0.6
0.8
1.0
AMT_REQ_CREDIT_BUREAU_YEAR 265992
AMT_REQ_CREDIT_BUREAU_QRT 265992
AMT_REQ_CREDIT_BUREAU_MON 265992
AMT_REQ_CREDIT_BUREAU_WEEK 265992
AMT_REQ_CREDIT_BUREAU_DAY 265992
AMT_REQ_CREDIT_BUREAU_HOUR 265992
FLAG_DOCUMENT_21 307511
FLAG_DOCUMENT_20 307511
FLAG_DOCUMENT_19 307511
FLAG_DOCUMENT_18 307511
FLAG_DOCUMENT_17 307511
FLAG_DOCUMENT_16 307511
FLAG_DOCUMENT_15 307511
FLAG_DOCUMENT_14 307511
FLAG_DOCUMENT_13 307511
FLAG_DOCUMENT_12 307511
FLAG_DOCUMENT_11 307511
FLAG_DOCUMENT_10 307511
FLAG_DOCUMENT_9 307511
FLAG_DOCUMENT_8 307511
FLAG_DOCUMENT_7 307511
FLAG_DOCUMENT_6 307511
FLAG_DOCUMENT_5 307511
FLAG_DOCUMENT_4 307511
FLAG_DOCUMENT_3 307511
FLAG_DOCUMENT_2 307511
DAYS_LAST_PHONE_CHANGE 307510
DEF_60_CNT_SOCIAL_CIRCLE 306490
OBS_60_CNT_SOCIAL_CIRCLE 306490
DEF_30_CNT_SOCIAL_CIRCLE 306490
OBS_30_CNT_SOCIAL_CIRCLE 306490
EMERGENCYSTATE_MODE 161756
WALLSMATERIAL_MODE 151170
TOTALAREA_MODE 159080
HOUSETYPE_MODE 153214
FONDKAPREMONT_MODE 97216
NONLIVINGAREA_MEDI 137829
NONLIVINGAPARTMENTS_MEDI 93997
LIVINGAREA_MEDI 153161
LIVINGAPARTMENTS_MEDI 97312
LANDAREA_MEDI 124921
FLOORSMIN_MEDI 98869
FLOORSMAX_MEDI 154491
ENTRANCES_MEDI 152683
ELEVATORS_MEDI 143620
COMMONAREA_MEDI 92646
YEARS_BUILD_MEDI 103023
YEARS_BEGINEXPLUATATION_MEDI 157504
BASEMENTAREA_MEDI 127568
APARTMENTS_MEDI 151450
NONLIVINGAREA_MODE 137829
NONLIVINGAPARTMENTS_MODE 93997
LIVINGAREA_MODE 153161
LIVINGAPARTMENTS_MODE 97312
LANDAREA_MODE 124921
FLOORSMIN_MODE 98869
FLOORSMAX_MODE 154491
ENTRANCES_MODE 152683
ELEVATORS_MODE 143620
COMMONAREA_MODE 92646
YEARS_BUILD_MODE 103023
YEARS_BEGINEXPLUATATION_MODE 157504
BASEMENTAREA_MODE 127568
APARTMENTS_MODE 151450
NONLIVINGAREA_AVG 137829
NONLIVINGAPARTMENTS_AVG 93997
LIVINGAREA_AVG 153161
LIVINGAPARTMENTS_AVG 97312
LANDAREA_AVG 124921
FLOORSMIN_AVG 98869
FLOORSMAX_AVG 154491
ENTRANCES_AVG 152683
ELEVATORS_AVG 143620
COMMONAREA_AVG 92646
YEARS_BUILD_AVG 103023
YEARS_BEGINEXPLUATATION_AVG 157504
BASEMENTAREA_AVG 127568
APARTMENTS_AVG 151450
EXT_SOURCE_3 246546
EXT_SOURCE_2 306851
EXT_SOURCE_1 134133
ORGANIZATION_TYPE 307511
LIVE_CITY_NOT_WORK_CITY 307511
REG_CITY_NOT_WORK_CITY 307511
REG_CITY_NOT_LIVE_CITY 307511
LIVE_REGION_NOT_WORK_REGION 307511
REG_REGION_NOT_WORK_REGION 307511
REG_REGION_NOT_LIVE_REGION 307511
HOUR_APPR_PROCESS_START 307511
WEEKDAY_APPR_PROCESS_START 307511
REGION_RATING_CLIENT_W_CITY 307511
REGION_RATING_CLIENT 307511
CNT_FAM_MEMBERS 307509
OCCUPATION_TYPE 211120
FLAG_EMAIL 307511
FLAG_PHONE 307511
FLAG_CONT_MOBILE 307511
FLAG_WORK_PHONE 307511
FLAG_EMP_PHONE 307511
FLAG_MOBIL 307511
OWN_CAR_AGE 104582
DAYS_ID_PUBLISH 307511
DAYS_REGISTRATION 307511
DAYS_EMPLOYED 307511
DAYS_BIRTH 307511
REGION_POPULATION_RELATIVE 307511
NAME_HOUSING_TYPE 307511
NAME_FAMILY_STATUS 307511
NAME_EDUCATION_TYPE 307511
NAME_INCOME_TYPE 307511
NAME_TYPE_SUITE 306219
AMT_GOODS_PRICE 307233
AMT_ANNUITY 307499
AMT_CREDIT 307511
AMT_INCOME_TOTAL 307511
CNT_CHILDREN 307511
FLAG_OWN_REALTY 307511
FLAG_OWN_CAR 307511
CODE_GENDER 307511
NAME_CONTRACT_TYPE 307511
TARGET 307511
SK_ID_CURR 307511
0
61502
123004
184506
246008
307511

> The above graph showing the number of missing values with respect to the columns.

Visualization Through Correlation Matrix

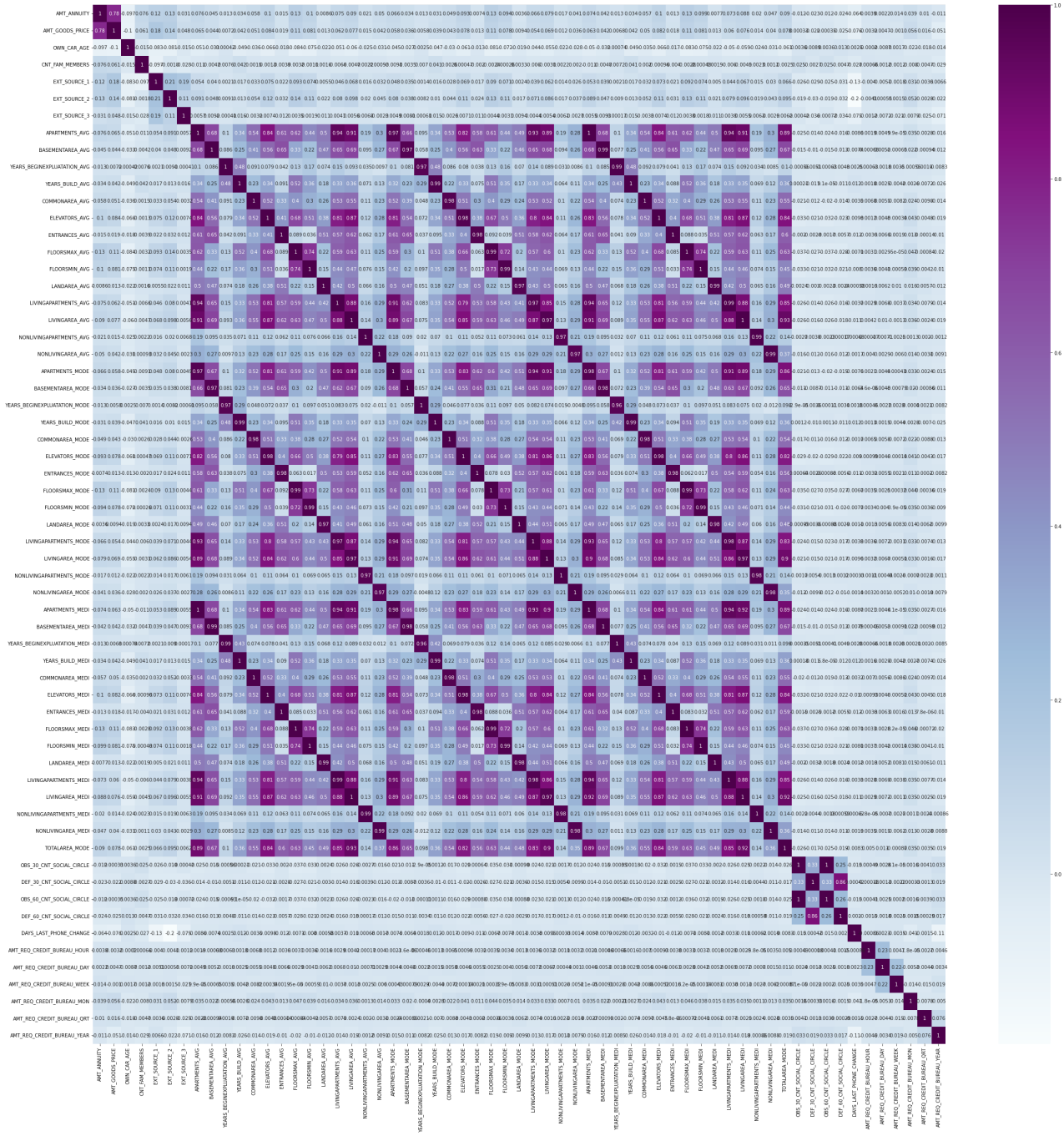
> A **Correlation Matrix** is used to examine the relationship between multiple variables at the same time. When we do this calculation we get a table containing the correlation coefficients between each variable and the others. Now, the coefficient show us both the strength of the relationship and its direction (positive or negative correlations). In Python, a correlation matrix can be created using the Python packages Pandas and NumPy, for instance.

- If we have a big data set, and we have an intention to explore patterns.
- For use in other statistical methods. For instance, correlation matrices can be used as data when conducting exploratory factor analysis, confirmatory factor analysis, structural equation models.
- Correlation matrices can also be used as a diagnostic when checking assumptions for e.g. regression analysis.

In [28]:
```
appp_50 = datasets['application_train'][datasets['application_train'].columns[da
```

In [29]:
```
plt.figure(figsize=(40,40))
sns.heatmap(appp_50.corr(),annot=True,cmap='BuPu')
```

Out[29]:
```
<AxesSubplot:>
```



Dropping columns with high correlation

In [30]:
```
datasets['application_train'] = datasets['application_train'].drop(['TOTALAREA_M
                                                                    'LIVINGAREA_
                                                                    'LIVINGAPART
                                                                   ,'APARTMENTS_MEDI','ELEVATORS_MEDI'
```

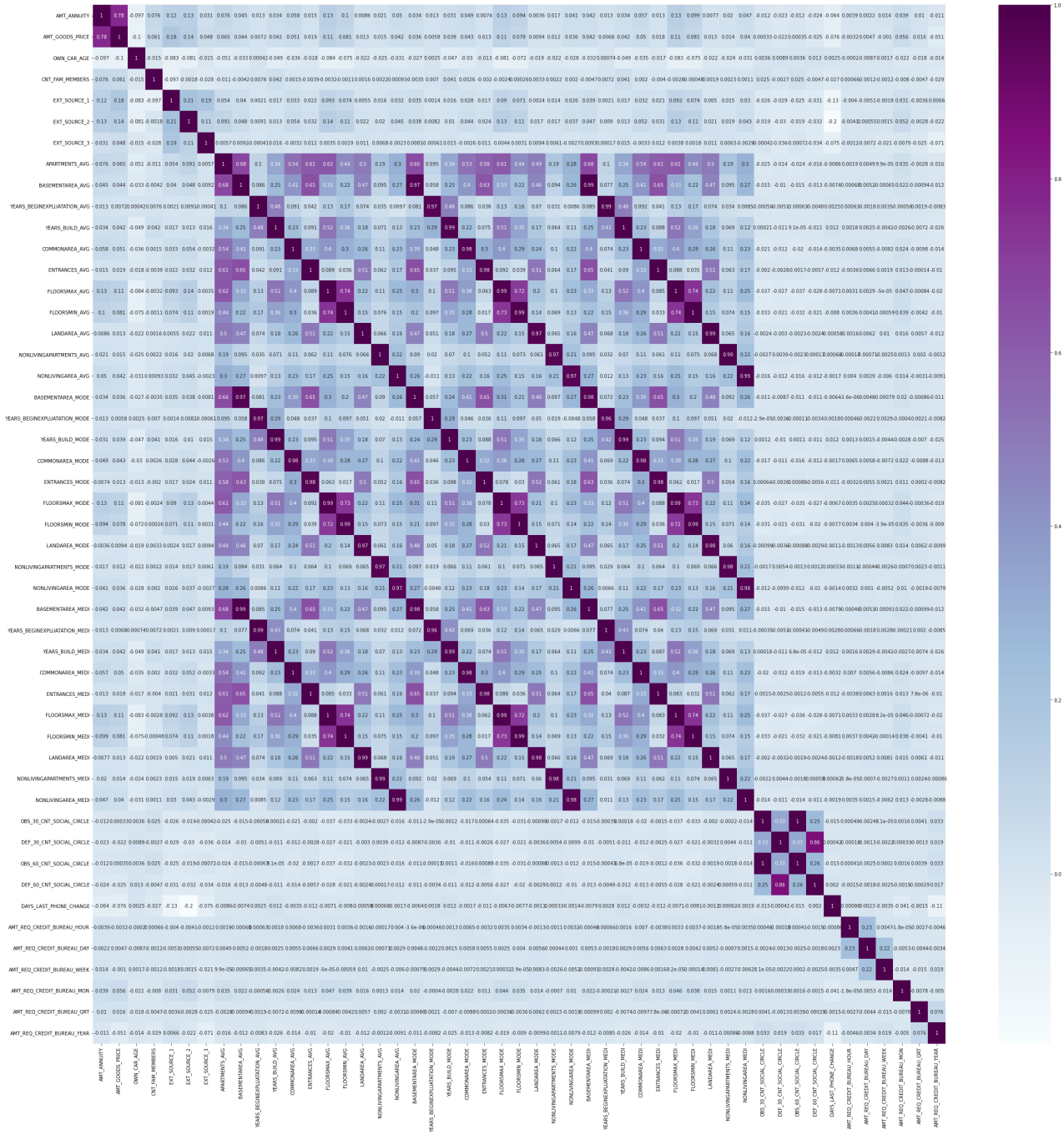
In [ ]:
```
```

In [31]:
```
appp_50A = datasets['application_train'][datasets['application_train'].columns[d
```

In [32]:
```
plt.figure(figsize=(40,40))
```

```
sns.heatmap(appp_50A.corr(),annot=True, cmap = 'BuPu')
```

Out[32]: <AxesSubplot:>



In [33]:

```
datasets['application_train'] = datasets['application_train'].drop(['NONLIVINGAR
                                                                    'FLOORSMAX_M
                                                                    'BASEMENTARE
                                                                'YEARS_BUILD_MODE','YEARS_BEGINEXPLU
                                                                    'ENTRANCES_M
                                                                    'REG_CITY_NO
                                                                'YEARS_BUILD_
                                                                    'NONLIVIN
                                                                'LIVE_CITY_N
                                                                'CNT_CHILDREN

                                                                ],axis = 1)
```

Conclusions of Correlation Matrix

- After we see that there are many independent or feature variable which are related to each other, we will deal with them in a way that we drop the repetitive features, i.e. delete one of those columns which are highly correlated with each other! Thus, as we can see above, I have removed all those columns that ae related to each other, just leaving out one!

In [34]:
```
percent = (datasets["application_train"].isnull().sum()/datasets["application_tr
sum_missing = datasets["application_train"].isna().sum().sort_values(ascending =
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys
missing_application_train_data.head(20)
```

Out[34]:

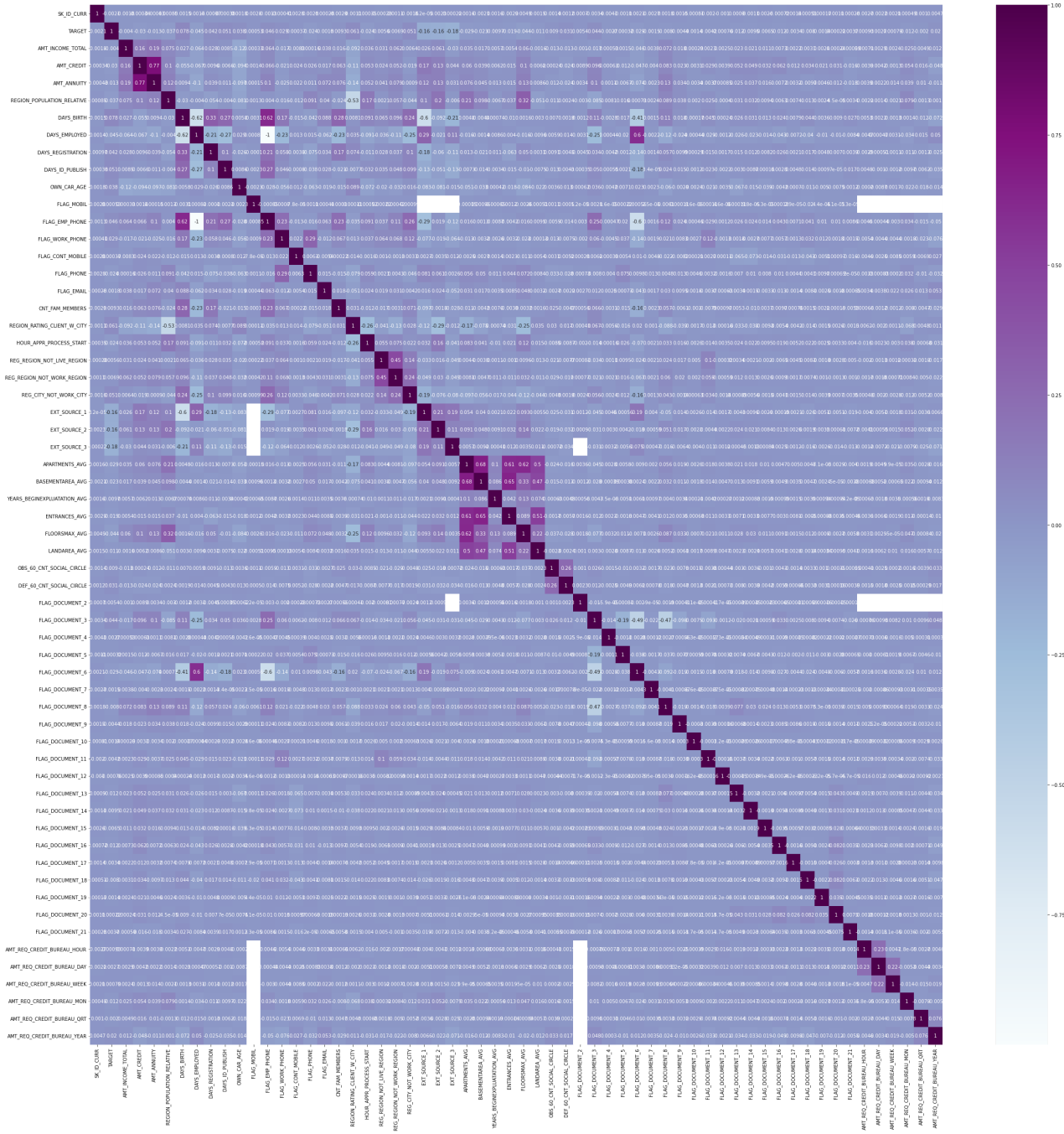
	Percent	Train Missing Count
FONDKAPREMONT_MODE	68.39	210295
OWN_CAR_AGE	65.99	202929
LANDAREA_AVG	59.38	182590
BASEMENTAREA_AVG	58.52	179943
EXT_SOURCE_1	56.38	173378
WALLSMATERIAL_MODE	50.84	156341
APARTMENTS_AVG	50.75	156061
ENTRANCES_AVG	50.35	154828
HOUSETYPE_MODE	50.18	154297
FLOORSMAX_AVG	49.76	153020
YEARS_BEGINEXPLUATATION_AVG	48.78	150007
EMERGENCYSTATE_MODE	47.40	145755
OCCUPATION_TYPE	31.35	96391
EXT_SOURCE_3	19.83	60965
AMT_REQ_CREDIT_BUREAU_HOUR	13.50	41519
AMT_REQ_CREDIT_BUREAU_DAY	13.50	41519
AMT_REQ_CREDIT_BUREAU_WEEK	13.50	41519
AMT_REQ_CREDIT_BUREAU_MON	13.50	41519
AMT_REQ_CREDIT_BUREAU_QRT	13.50	41519
AMT_REQ_CREDIT_BUREAU_YEAR	13.50	41519

In [35]:
```
datasets['application_train'].shape
```

Out[35]:
```
(307511, 76)
```

In [36]:
```
plt.figure(figsize=(40,40))
sns.heatmap(datasets['application_train'].corr(),annot=True, cmap='BuPu')
```

Out[36]:
```
<AxesSubplot:>
```



Visual Exploratory Data Analysis

In [37]:
```
datasets['application_train'].shape
```

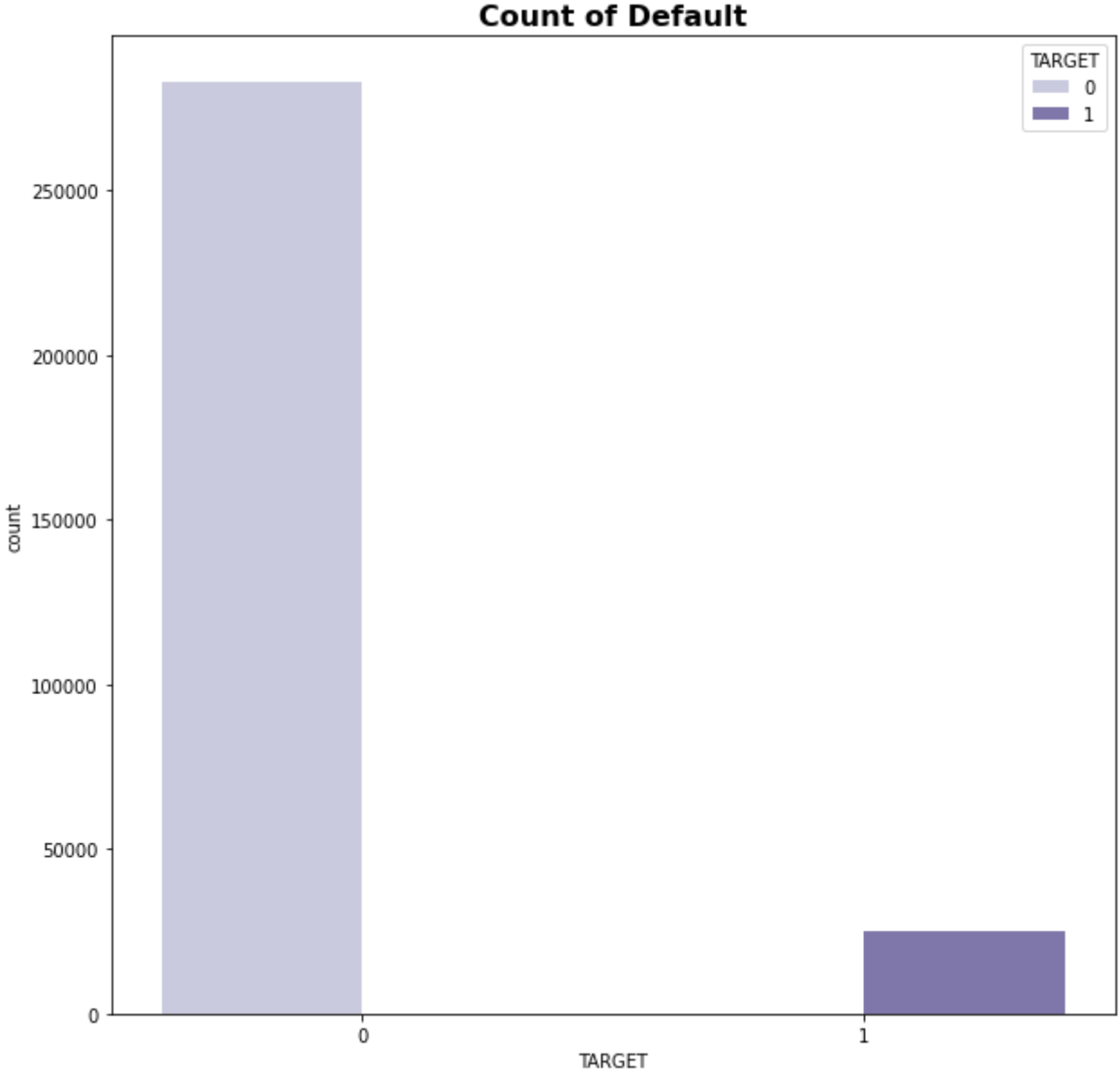
Out[37]:
```
(307511, 76)
```

Idea of Target Variables(Count)

In [38]:
```
plt.figure(figsize=(10,10))
plt.title("Count of Default", fontweight = 'bold', fontsize = 16)
```

```
sns.countplot(x ='TARGET',data=datasets['application_train'], hue='TARGET',palet
```

Out[38]:
```
<AxesSubplot:title={'center':'Count of Default'}, xlabel='TARGET', ylabel='coun
t'>
```

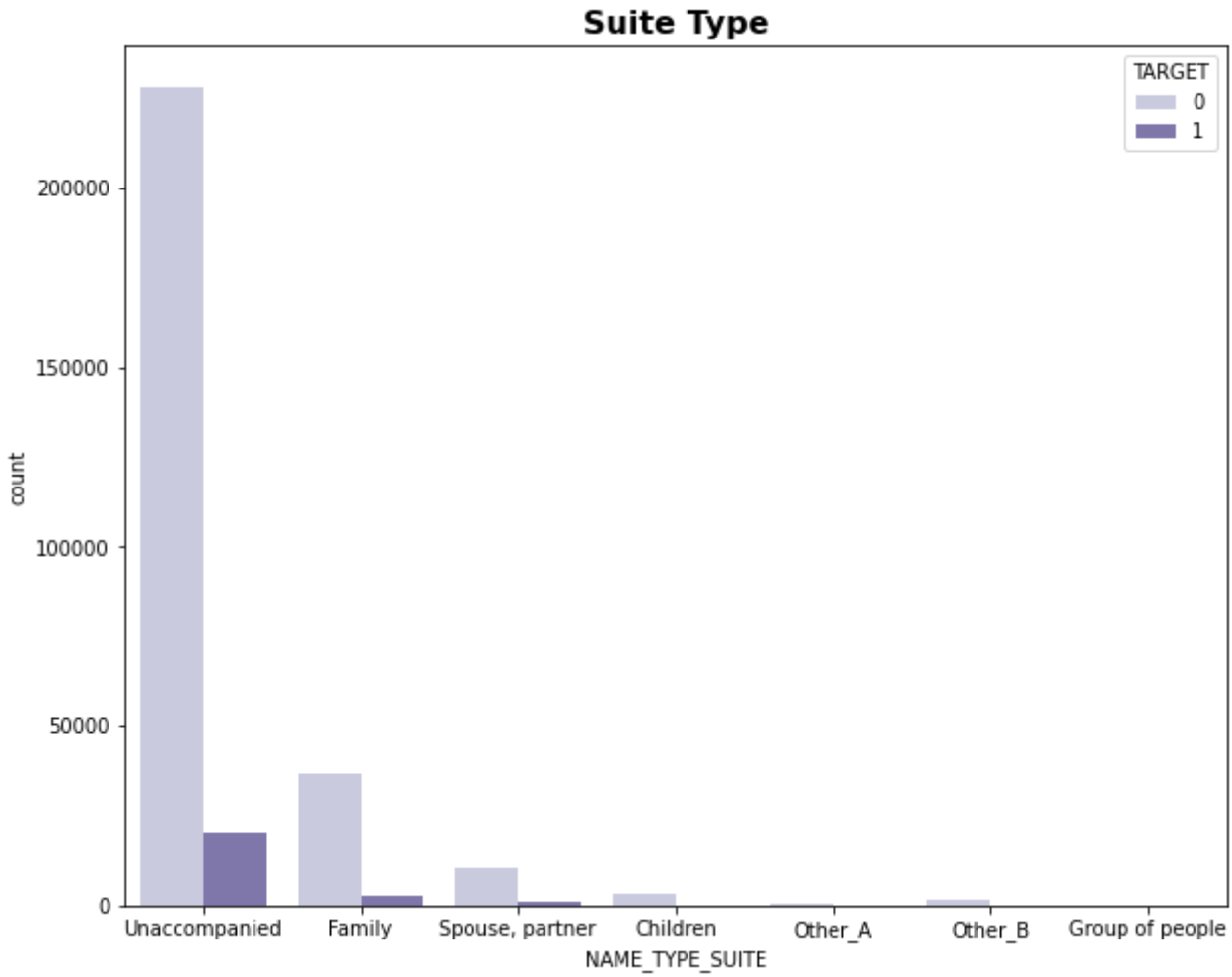


> People who repaid the loan were almost 10 times than the defaulters.

Visualization Through Suite Type with respect to Target variable

In [39]:
```
plt.figure(figsize=(10,8))
plt.title("Suite Type", fontweight = 'bold', fontsize = 16)
sns.countplot(datasets['application_train']['NAME_TYPE_SUITE'], hue=datasets['ap
```

Out[39]:
```
<AxesSubplot:title={'center':'Suite Type'}, xlabel='NAME_TYPE_SUITE', ylabel='co
unt'>
```

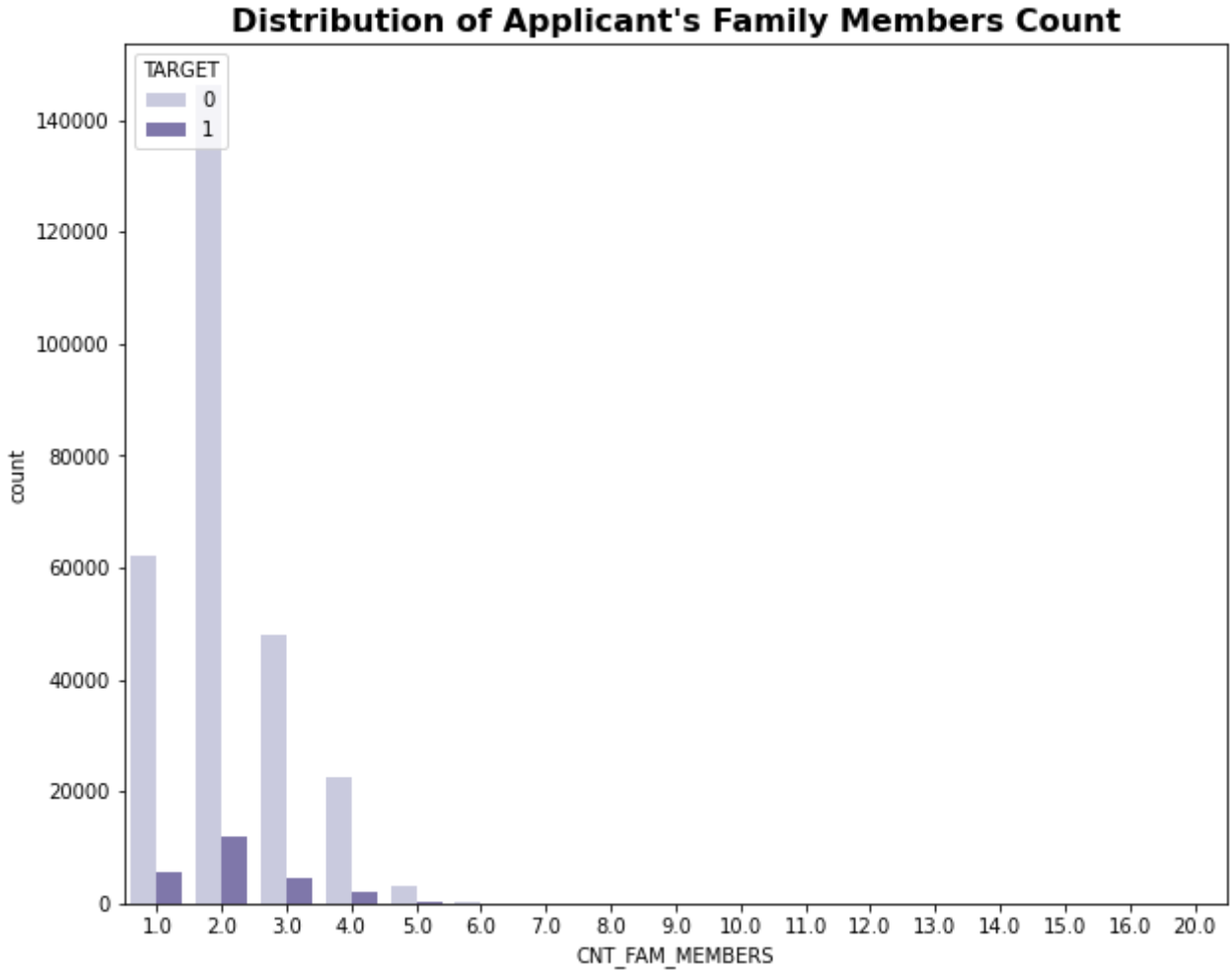


- Above, we have plotted a graph which describes the loan applicant's family members.
- As we can see above, there are many people who are unaccompanied followed by Family.

Visualization Through Distribution of Applicant's Family Members Count with respect to Target variable

In [40]:
```
plt.figure(figsize=(10,8))
plt.title("Distribution of Applicant's Family Members Count", fontweight = 'bold
sns.countplot(datasets['application_train']['CNT_FAM_MEMBERS'],hue=datasets['app
```

Out[40]:
```
<AxesSubplot:title={'center':"Distribution of Applicant's Family Members Coun
t"}, xlabel='CNT_FAM_MEMBERS', ylabel='count'>
```



- Above, we have plotted a graph which describes the loan applicant's family members.
- As we can see above, there are hardly any people who have a count of family members greater than 5.

Correlation of columns with respect to Target Variable

In [41]:
```
datasets['application_train'].corrwith(datasets['application_train']["TARGET"]).
```

Out[41]:
```
TARGET                         1.000000
DAYS_BIRTH                     0.078239
REGION_RATING_CLIENT_W_CITY    0.060893
DAYS_ID_PUBLISH                0.051457
REG_CITY_NOT_WORK_CITY         0.050994
FLAG_EMP_PHONE                 0.045982
FLAG_DOCUMENT_3                0.044346
DAYS_REGISTRATION              0.041975
OWN_CAR_AGE                    0.037612
DEF_60_CNT_SOCIAL_CIRCLE       0.031276
FLAG_WORK_PHONE                0.028524
AMT_REQ_CREDIT_BUREAU_YEAR     0.019930
CNT_FAM_MEMBERS                0.009308
OBS_60_CNT_SOCIAL_CIRCLE       0.009022
REG_REGION_NOT_WORK_REGION     0.006942
REG_REGION_NOT_LIVE_REGION     0.005576
FLAG_DOCUMENT_2                0.005417
```

```
FLAG_DOCUMENT_21               0.003709
AMT_REQ_CREDIT_BUREAU_DAY      0.002704
AMT_REQ_CREDIT_BUREAU_HOUR     0.000930
AMT_REQ_CREDIT_BUREAU_WEEK     0.000788
FLAG_MOBIL                     0.000534
FLAG_CONT_MOBILE               0.000370
FLAG_DOCUMENT_20               0.000215
FLAG_DOCUMENT_5               -0.000316
FLAG_DOCUMENT_12              -0.000756
FLAG_DOCUMENT_19              -0.001358
FLAG_DOCUMENT_10              -0.001414
FLAG_DOCUMENT_7               -0.001520
FLAG_EMAIL                    -0.001758
AMT_REQ_CREDIT_BUREAU_QRT     -0.002022
SK_ID_CURR                    -0.002108
FLAG_DOCUMENT_4               -0.002672
FLAG_DOCUMENT_17              -0.003378
AMT_INCOME_TOTAL              -0.003982
FLAG_DOCUMENT_11              -0.004229
FLAG_DOCUMENT_9               -0.004352
FLAG_DOCUMENT_15              -0.006536
FLAG_DOCUMENT_18              -0.007952
FLAG_DOCUMENT_8               -0.008040
FLAG_DOCUMENT_14              -0.009464
YEARS_BEGINEXPLUATATION_AVG   -0.009728
LANDAREA_AVG                  -0.010885
FLAG_DOCUMENT_13              -0.011583
FLAG_DOCUMENT_16              -0.011615
AMT_REQ_CREDIT_BUREAU_MON     -0.012462
AMT_ANNUITY                   -0.012817
ENTRANCES_AVG                 -0.019172
BASEMENTAREA_AVG              -0.022746
FLAG_PHONE                    -0.023806
HOUR_APPR_PROCESS_START       -0.024166
FLAG_DOCUMENT_6               -0.028602
APARTMENTS_AVG                -0.029498
AMT_CREDIT                    -0.030369
REGION_POPULATION_RELATIVE    -0.037227
FLOORSMAX_AVG                 -0.044003
DAYS_EMPLOYED                 -0.044932
EXT_SOURCE_1                  -0.155317
EXT_SOURCE_2                  -0.160472
EXT_SOURCE_3                  -0.178919
dtype: float64
```

> As we can see EXT_SOURCE_1, EXT_SOURCE_2 and EXT_SOURCE_3 are most negatively correlated.

In [42]:
```
target_corr = pd.DataFrame(datasets['application_train'].corrwith(datasets['appl
```

In [43]:
```
target_corr = target_corr.rename(columns={'index':'Column_name',0:'corr_with_tar
```

In [44]:
```
target_corr.head()
```

Out[44]:

	Column_name	corr_with_target
0	TARGET	1.000000
1	DAYS_BIRTH	0.078239
2	REGION_RATING_CLIENT_W_CITY	0.060893
3	DAYS_ID_PUBLISH	0.051457
4	REG_CITY_NOT_WORK_CITY	0.050994

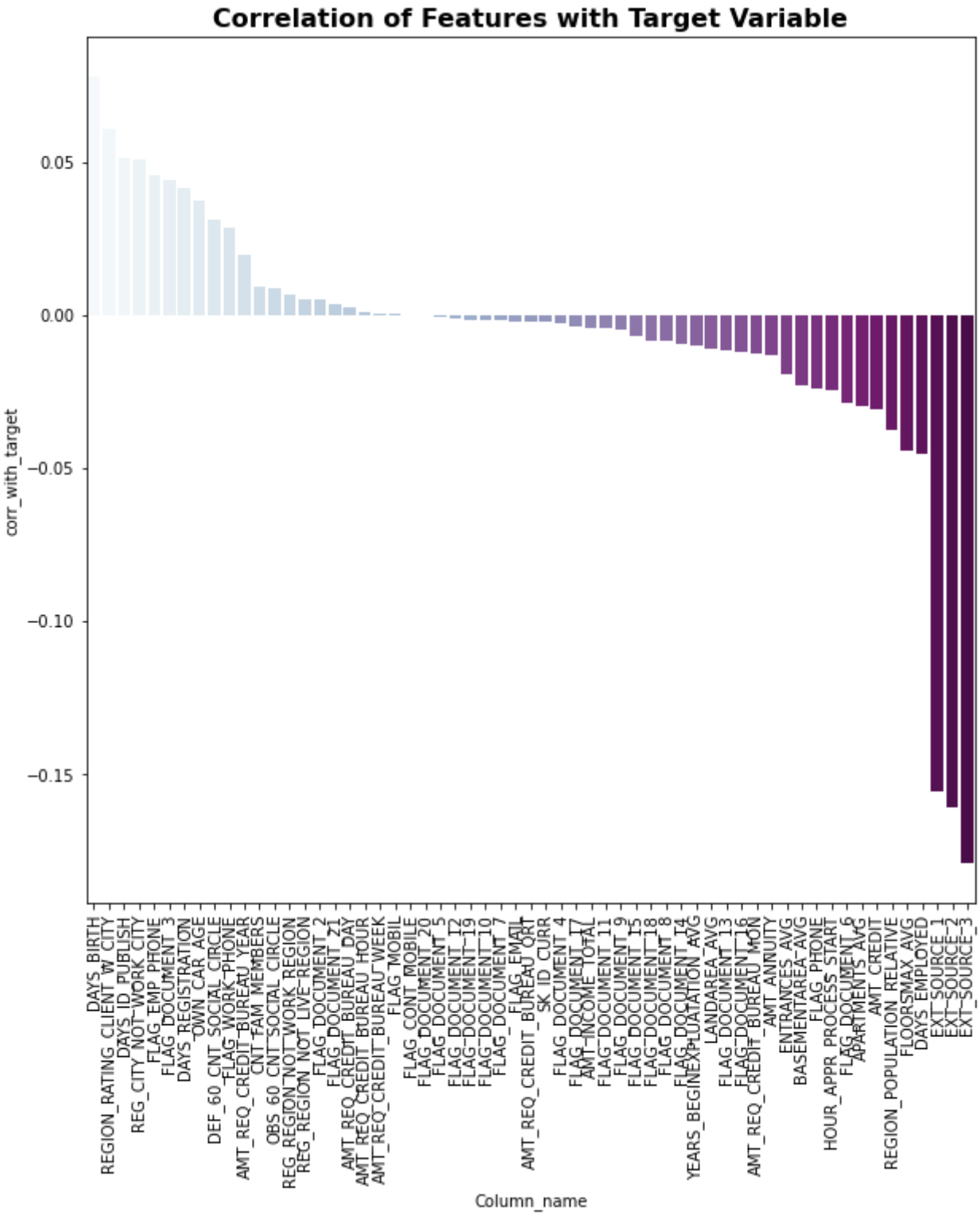
In [45]:
```
target_corr.tail()
```

Out[45]:

	Column_name	corr_with_target
55	FLOORSMAX_AVG	-0.044003
56	DAYS_EMPLOYED	-0.044932
57	EXT_SOURCE_1	-0.155317
58	EXT_SOURCE_2	-0.160472
59	EXT_SOURCE_3	-0.178919

Visualization Through Correlation with TARGET Variable

In [46]:
```
plt.figure(figsize=(10,10))
sns.barplot(data=target_corr,x=target_corr['Column_name'][1:], y= target_corr['c
plt.xticks(rotation=90)
plt.title('Correlation of Features with Target Variable', fontweight = 'bold', f
plt.show()
```



> Above, we have plotted the graph of correlation of all the features with the target variable. As it can be seen above, there are **approximately 30%** features which are not at all correlated with the target variable. We can drop them as I do not believe that they will influence our predictions of the target variable!

> Removing more columns on the basis of correlation with target variable.

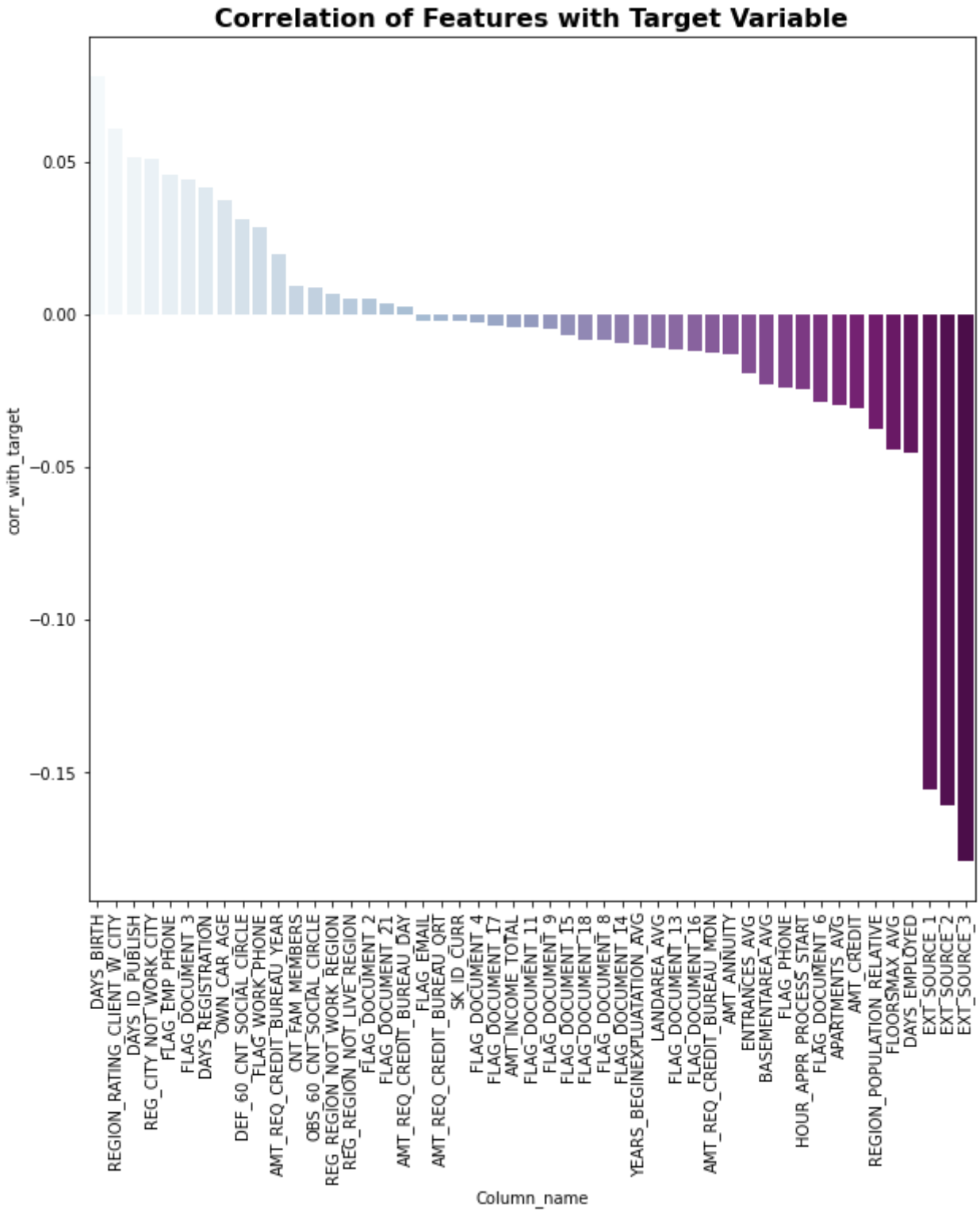
In [47]:

```
datasets['application_train'] = datasets['application_train'].drop(['AMT_REQ_CRE
'AMT_REQ_CREDIT_BUREAU_WEEK',
'FLAG_MOBIL',
'FLAG_CONT_MOBILE',
'FLAG_DOCUMENT_20',
'FLAG_DOCUMENT_5',
'FLAG_DOCUMENT_12',
'FLAG_DOCUMENT_19',
'FLAG_DOCUMENT_10',
'FLAG_DOCUMENT_7'],axis=1)
```

In [48]:
```
target_corr1 = pd.DataFrame(datasets['application_train'].corrwith(datasets['app
```

In [49]:
```
target_corr1 = target_corr1.rename(columns={'index':'Column_name',0:'corr_with_t
```

In [50]:
```
plt.figure(figsize=(10,10))
sns.barplot(data=target_corr1,x=target_corr1['Column_name'][1:], y= target_corr1
plt.xticks(rotation=90)
plt.title('Correlation of Features with Target Variable', fontweight = 'bold', f
plt.show()
```



In [52]:
```
datasets['application_train'].corrwith(datasets['application_train']["TARGET"]).
```

Out[52]:
```
TARGET                          1.000000
DAYS_BIRTH                      0.078239
REGION_RATING_CLIENT_W_CITY     0.060893
DAYS_ID_PUBLISH                 0.051457
REG_CITY_NOT_WORK_CITY          0.050994
FLAG_EMP_PHONE                  0.045982
FLAG_DOCUMENT_3                 0.044346
DAYS_REGISTRATION               0.041975
OWN_CAR_AGE                     0.037612
```

```
DEF_60_CNT_SOCIAL_CIRCLE       0.031276
FLAG_WORK_PHONE                0.028524
AMT_REQ_CREDIT_BUREAU_YEAR     0.019930
CNT_FAM_MEMBERS                0.009308
OBS_60_CNT_SOCIAL_CIRCLE       0.009022
REG_REGION_NOT_WORK_REGION     0.006942
REG_REGION_NOT_LIVE_REGION     0.005576
FLAG_DOCUMENT_2                0.005417
FLAG_DOCUMENT_21               0.003709
AMT_REQ_CREDIT_BUREAU_DAY      0.002704
FLAG_EMAIL                    -0.001758
AMT_REQ_CREDIT_BUREAU_QRT     -0.002022
SK_ID_CURR                    -0.002108
FLAG_DOCUMENT_4               -0.002672
FLAG_DOCUMENT_17              -0.003378
AMT_INCOME_TOTAL              -0.003982
FLAG_DOCUMENT_11              -0.004229
FLAG_DOCUMENT_9               -0.004352
FLAG_DOCUMENT_15              -0.006536
FLAG_DOCUMENT_18              -0.007952
FLAG_DOCUMENT_8               -0.008040
FLAG_DOCUMENT_14              -0.009464
YEARS_BEGINEXPLUATATION_AVG   -0.009728
LANDAREA_AVG                  -0.010885
FLAG_DOCUMENT_13              -0.011583
FLAG_DOCUMENT_16              -0.011615
AMT_REQ_CREDIT_BUREAU_MON     -0.012462
AMT_ANNUITY                   -0.012817
ENTRANCES_AVG                 -0.019172
BASEMENTAREA_AVG              -0.022746
FLAG_PHONE                    -0.023806
HOUR_APPR_PROCESS_START       -0.024166
FLAG_DOCUMENT_6               -0.028602
APARTMENTS_AVG                -0.029498
AMT_CREDIT                    -0.030369
REGION_POPULATION_RELATIVE    -0.037227
FLOORSMAX_AVG                 -0.044003
DAYS_EMPLOYED                 -0.044932
EXT_SOURCE_1                  -0.155317
EXT_SOURCE_2                  -0.160472
EXT_SOURCE_3                  -0.178919
dtype: float64
```

In [53]:

```
datasets['application_train'] = datasets['application_train'].drop(['REG_REGION_
'REG_REGION_NOT_LIVE_REGION',
'FLAG_DOCUMENT_2',
'FLAG_DOCUMENT_21',
'AMT_REQ_CREDIT_BUREAU_DAY',
'FLAG_EMAIL',
'AMT_REQ_CREDIT_BUREAU_QRT',
'FLAG_DOCUMENT_4',
'FLAG_DOCUMENT_17',
'FLAG_DOCUMENT_11',
'FLAG_DOCUMENT_9',
'FLAG_DOCUMENT_15',
'FLAG_DOCUMENT_18',
'FLAG_DOCUMENT_8'],axis=1)
```

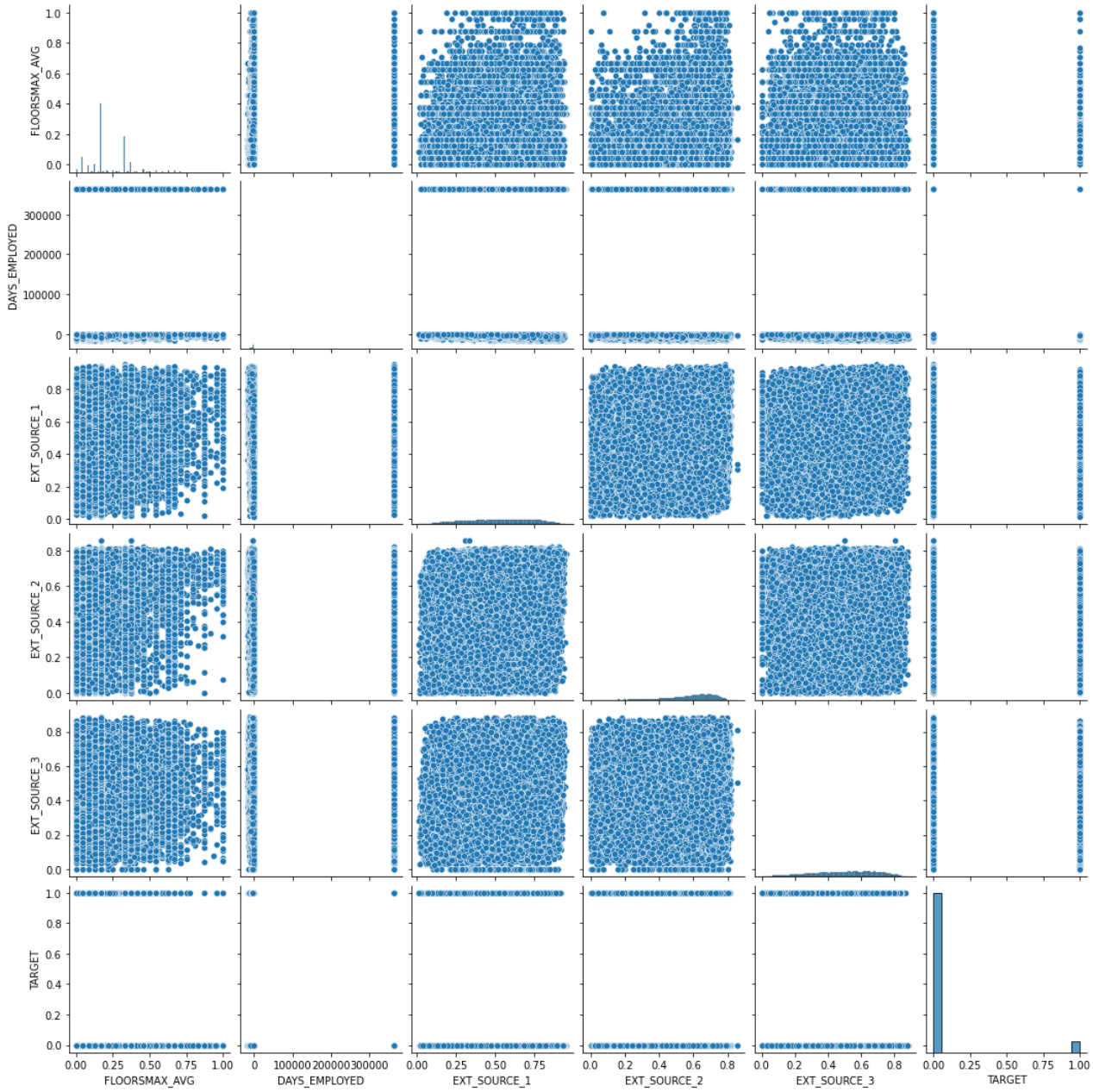
Top 5 Correlated features with Target Variable

In [54]:
```
df = pd.DataFrame(datasets['application_train'].corrwith(datasets['application_t
df1 = datasets['application_train'][['FLOORSMAX_AVG','DAYS_EMPLOYED','EXT_SOURCE
df1.dropna(inplace=True)
df1.head()
```

Out[54]:

	FLOORSMAX_AVG	DAYS_EMPLOYED	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	TARG
0	0.0833	-637	0.083037	0.262949	0.139376	
12	0.1667	-2717	0.464831	0.715042	0.176653	
25	0.1667	-3494	0.561948	0.651406	0.461482	
50	0.4083	-1176	0.656225	0.450850	0.479449	
51	0.4583	-6977	0.311510	0.713355	0.309275	

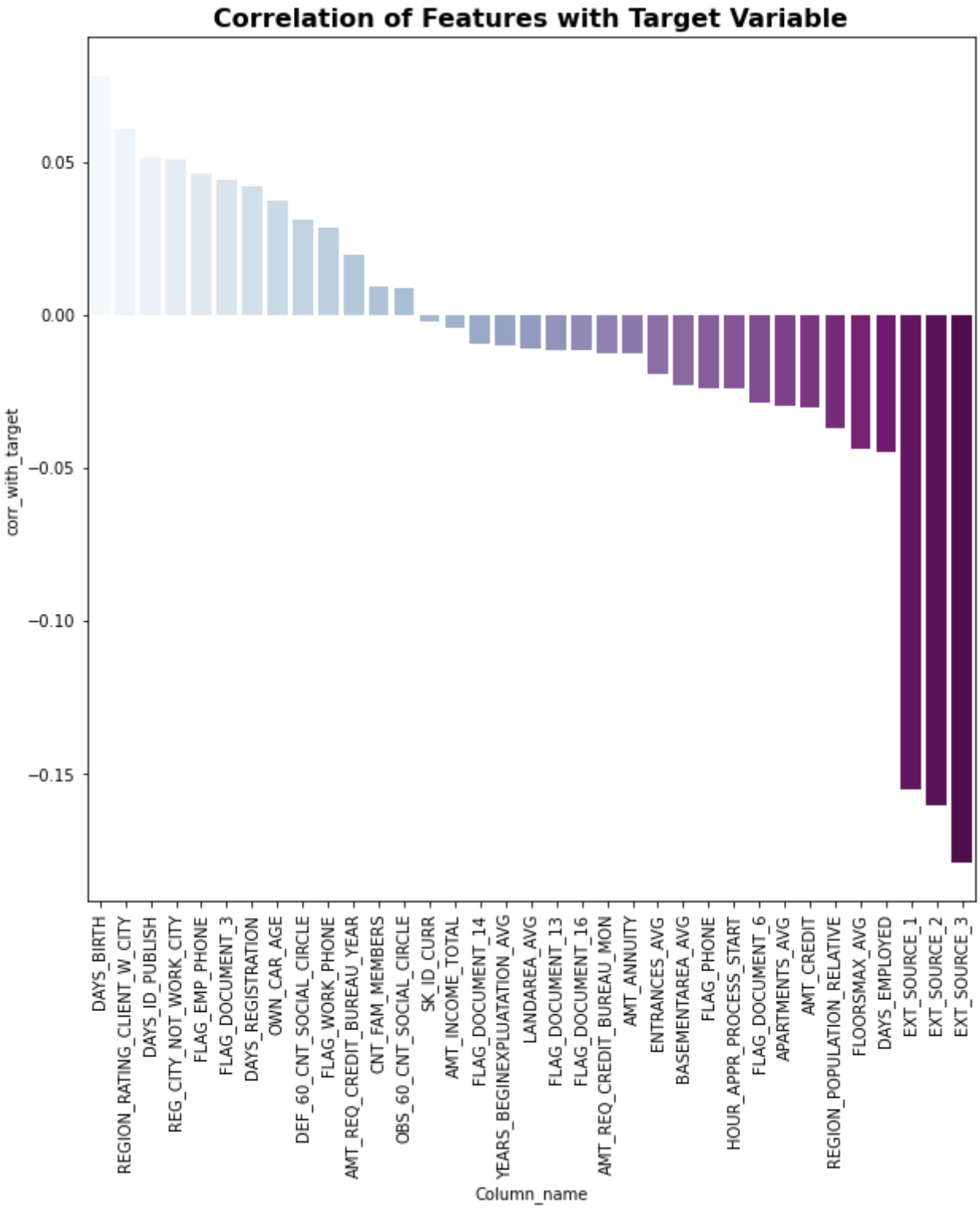
In [55]:
```
sns.pairplot(df1,palette='Purples')
plt.show()
```



In [56]:
```
target_corr2 = pd.DataFrame(datasets['application_train'].corrwith(datasets['app
```

In [57]:
```
target_corr2 = target_corr2.rename(columns={'index':'Column_name', 0:'corr_with_
```

In [58]:
```
plt.figure(figsize=(10,10))
sns.barplot(data=target_corr2,x=target_corr2['Column_name'][1:], y= target_corr2
plt.xticks(rotation=90)
plt.title('Correlation of Features with Target Variable', fontweight = 'bold', f
plt.show()
```



Conclusion from the Correlation Matrix

> After we have dropped the unimportant or unrelated columns, we have plotted the correlation with the TARGET variable! This looks so much better than before. We can see that the remaning variables are quite correlated with the TARGET Variable!

In [59]:
```
datasets['application_train'].info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 52 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   SK_ID_CURR                   307511 non-null  int64  
 1   TARGET                       307511 non-null  int64  
 2   NAME_CONTRACT_TYPE           307511 non-null  object 
 3   CODE_GENDER                  307511 non-null  object 
 4   FLAG_OWN_CAR                 307511 non-null  object 
 5   FLAG_OWN_REALTY              307511 non-null  object 
 6   AMT_INCOME_TOTAL             307511 non-null  float64
 7   AMT_CREDIT                   307511 non-null  float64
 8   AMT_ANNUITY                  307499 non-null  float64
 9   NAME_TYPE_SUITE              306219 non-null  object 
 10  NAME_INCOME_TYPE             307511 non-null  object 
 11  NAME_EDUCATION_TYPE          307511 non-null  object 
 12  NAME_FAMILY_STATUS           307511 non-null  object 
 13  NAME_HOUSING_TYPE            307511 non-null  object 
 14  REGION_POPULATION_RELATIVE   307511 non-null  float64
 15  DAYS_BIRTH                   307511 non-null  int64  
 16  DAYS_EMPLOYED                307511 non-null  int64  
 17  DAYS_REGISTRATION            307511 non-null  float64
 18  DAYS_ID_PUBLISH              307511 non-null  int64  
 19  OWN_CAR_AGE                  104582 non-null  float64
 20  FLAG_EMP_PHONE               307511 non-null  int64  
 21  FLAG_WORK_PHONE              307511 non-null  int64  
 22  FLAG_PHONE                   307511 non-null  int64  
 23  OCCUPATION_TYPE              211120 non-null  object 
 24  CNT_FAM_MEMBERS              307509 non-null  float64
 25  REGION_RATING_CLIENT_W_CITY  307511 non-null  int64  
 26  WEEKDAY_APPR_PROCESS_START   307511 non-null  object 
 27  HOUR_APPR_PROCESS_START      307511 non-null  int64  
 28  REG_CITY_NOT_WORK_CITY       307511 non-null  int64  
 29  ORGANIZATION_TYPE            307511 non-null  object 
 30  EXT_SOURCE_1                 134133 non-null  float64
 31  EXT_SOURCE_2                 306851 non-null  float64
 32  EXT_SOURCE_3                 246546 non-null  float64
 33  APARTMENTS_AVG               151450 non-null  float64
 34  BASEMENTAREA_AVG             127568 non-null  float64
 35  YEARS_BEGINEXPLUATATION_AVG  157504 non-null  float64
 36  ENTRANCES_AVG                152683 non-null  float64
 37  FLOORSMAX_AVG                154491 non-null  float64
 38  LANDAREA_AVG                 124921 non-null  float64
 39  FONDKAPREMONT_MODE           97216 non-null   object 
 40  HOUSETYPE_MODE               153214 non-null  object 
 41  WALLSMATERIAL_MODE           151170 non-null  object 
 42  EMERGENCYSTATE_MODE          161756 non-null  object 
 43  OBS_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 44  DEF_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 45  FLAG_DOCUMENT_3              307511 non-null  int64  
 46  FLAG_DOCUMENT_6              307511 non-null  int64  
 47  FLAG_DOCUMENT_13             307511 non-null  int64  
 48  FLAG_DOCUMENT_14             307511 non-null  int64  
 49  FLAG_DOCUMENT_16             307511 non-null  int64  
 50  AMT_REQ_CREDIT_BUREAU_MON    265992 non-null  float64
 51  AMT_REQ_CREDIT_BUREAU_YEAR   265992 non-null  float64
dtypes: float64(20), int64(16), object(16)
memory usage: 122.0+ MB
```

Visualization Through Correlation Matrix

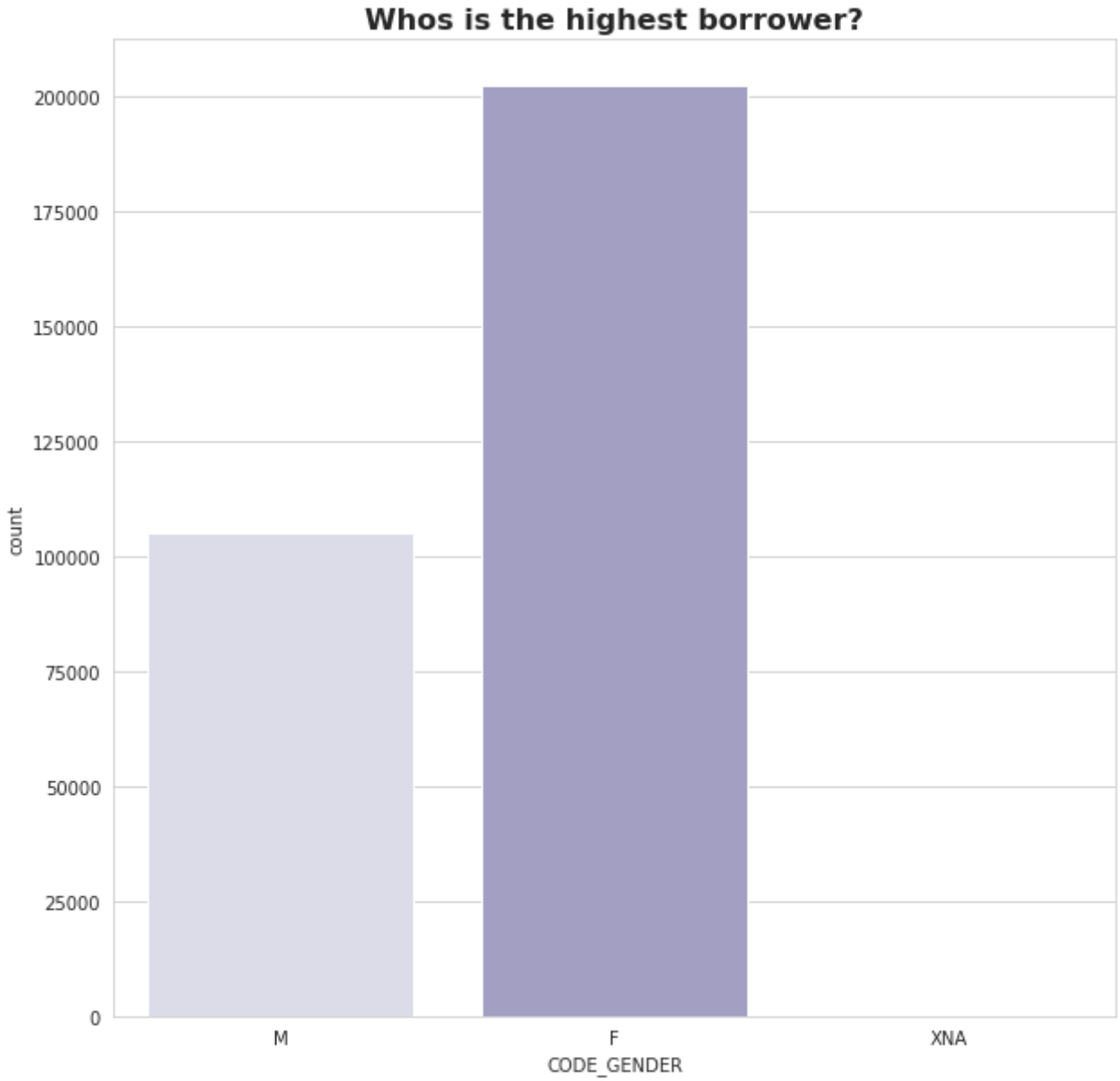
The **info()** function is used to print a concise summary of a DataFrame. This method prints information about a DataFrame including the index dtype and column dtypes, non-null values and memory usage.

we see that we have a lot of float, numeric, int and object data type values in the dataset!

Who's the highest borrower

In [60]:
```
sns.set_style('whitegrid')
plt.figure(figsize = (10,10))
plt.title("Whos is the highest borrower?", fontweight = 'bold', fontsize = 16)
sns.countplot(x='CODE_GENDER',data=datasets['application_train'], palette = 'Pur
```

Out[60]:
```
<AxesSubplot:title={'center':'Whos is the highest borrower?'}, xlabel='CODE_GEND
ER', ylabel='count'>
```



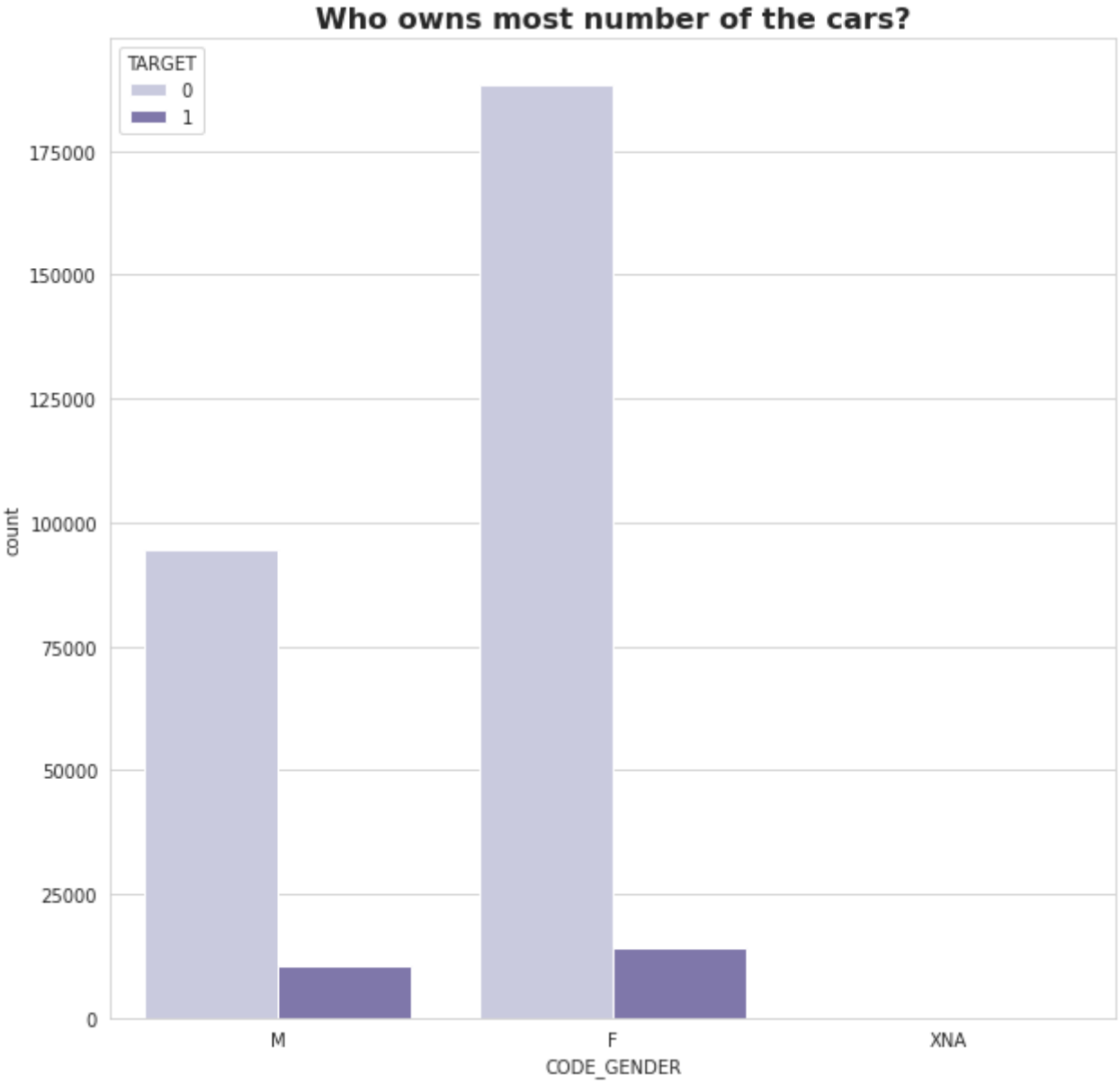
Insights:

Basically, there are more number of females than the number of males in the dataset! Thus one may also conclude that FEMALES take more loans than the MALES! But it might be incorrect to assume that!

Who's the highest borrower

In [61]:
```
plt.figure(figsize=(10,10))
plt.title("Who owns most number of the cars?", fontweight = 'bold', fontsize = 1
sns.countplot(x='CODE_GENDER',data= datasets['application_train'], hue='TARGET',
```

Out[61]:
```
<AxesSubplot:title={'center':'Who owns most number of the cars?'}, xlabel='CODE_
GENDER', ylabel='count'>
```

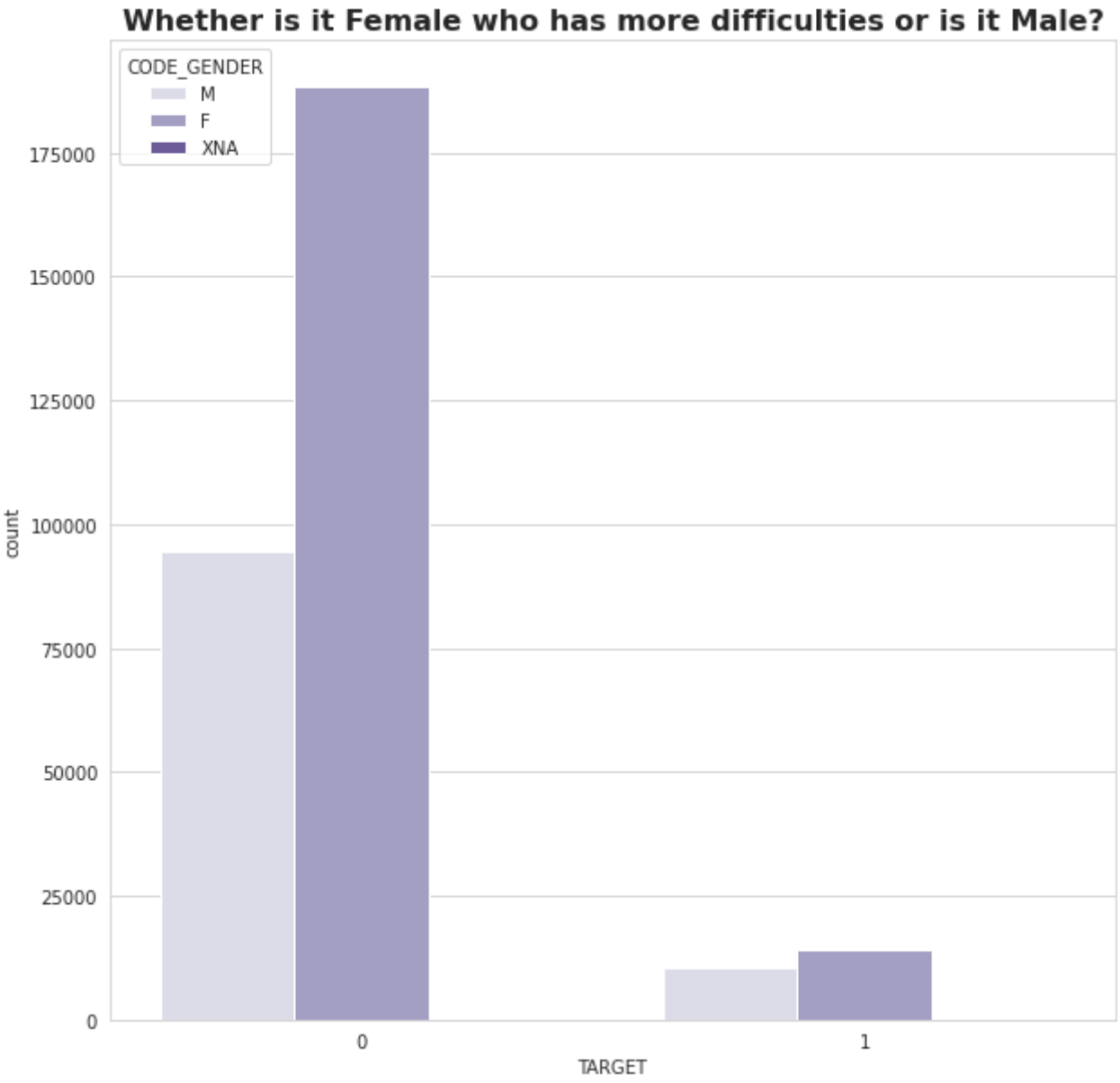


Conclusions:

> In this plot we have drawn a graph whther or not person i.e man or woman defaults a loan. However, the count of females is higher than the count of males, the graph says something different. The number of females owning a car is more than the number of males owning a car and percent wise they default less.

In [62]:
```
plt.figure(figsize=(10,10))
plt.title("Whether is it Female who has more difficulties or is it Male?", fontw
sns.countplot(x='TARGET',hue='CODE_GENDER',data=datasets['application_train'], p
```

Out[62]:
```
<AxesSubplot:title={'center':'Whether is it Female who has more difficulties or
is it Male?'}, xlabel='TARGET', ylabel='count'>
```

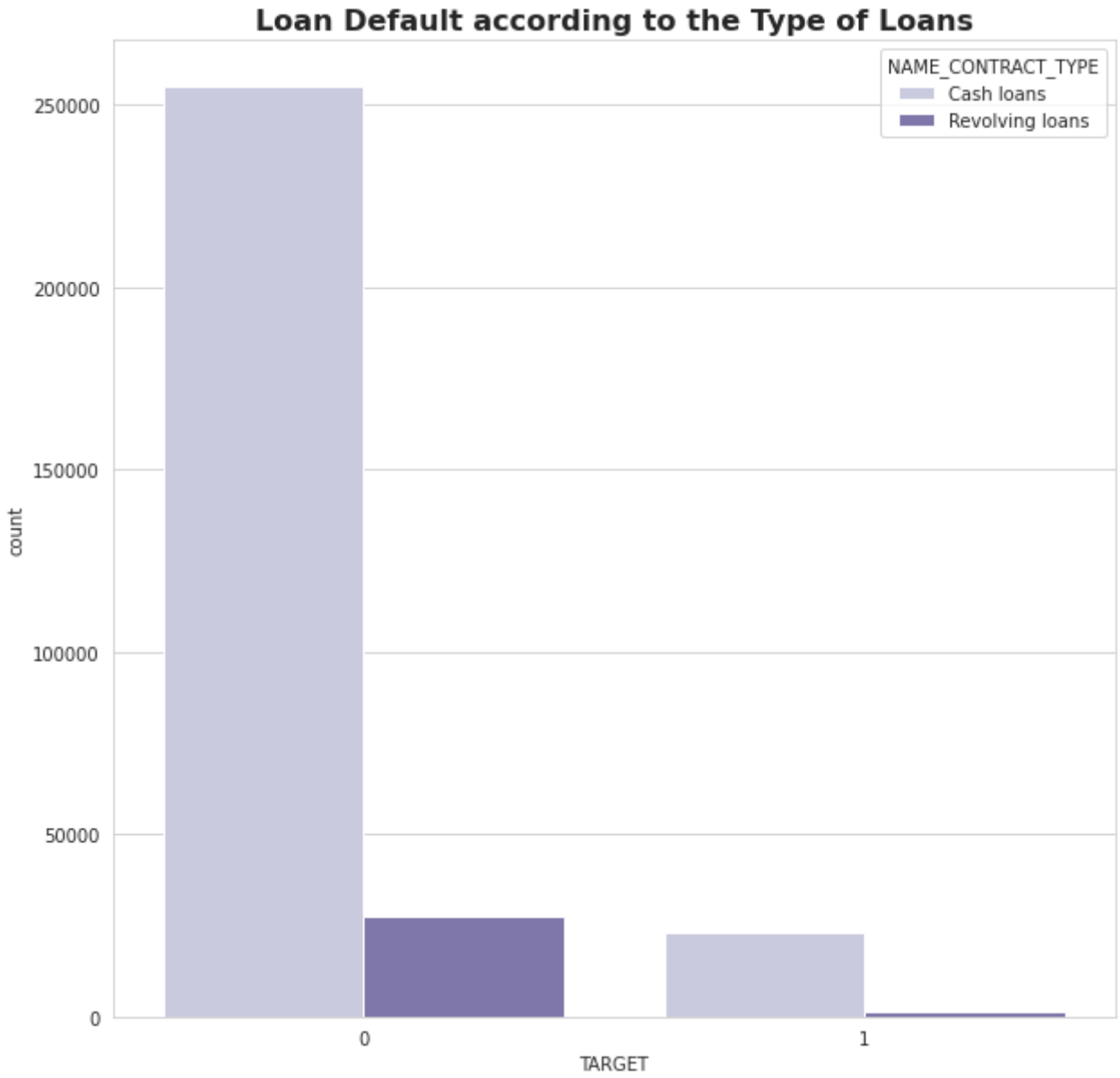


Insights:

It appears that Females have much more of difficulty in gettng the loans as compared to the males.

In [63]:
```
plt.figure(figsize=(10,10))
plt.title("Loan Default according to the Type of Loans",fontweight='bold' , font
sns.countplot(x='TARGET',hue='NAME_CONTRACT_TYPE',data=datasets['application_tra
```

Out[63]:
```
<AxesSubplot:title={'center':'Loan Default according to the Type of Loans'}, xla
bel='TARGET', ylabel='count'>
```

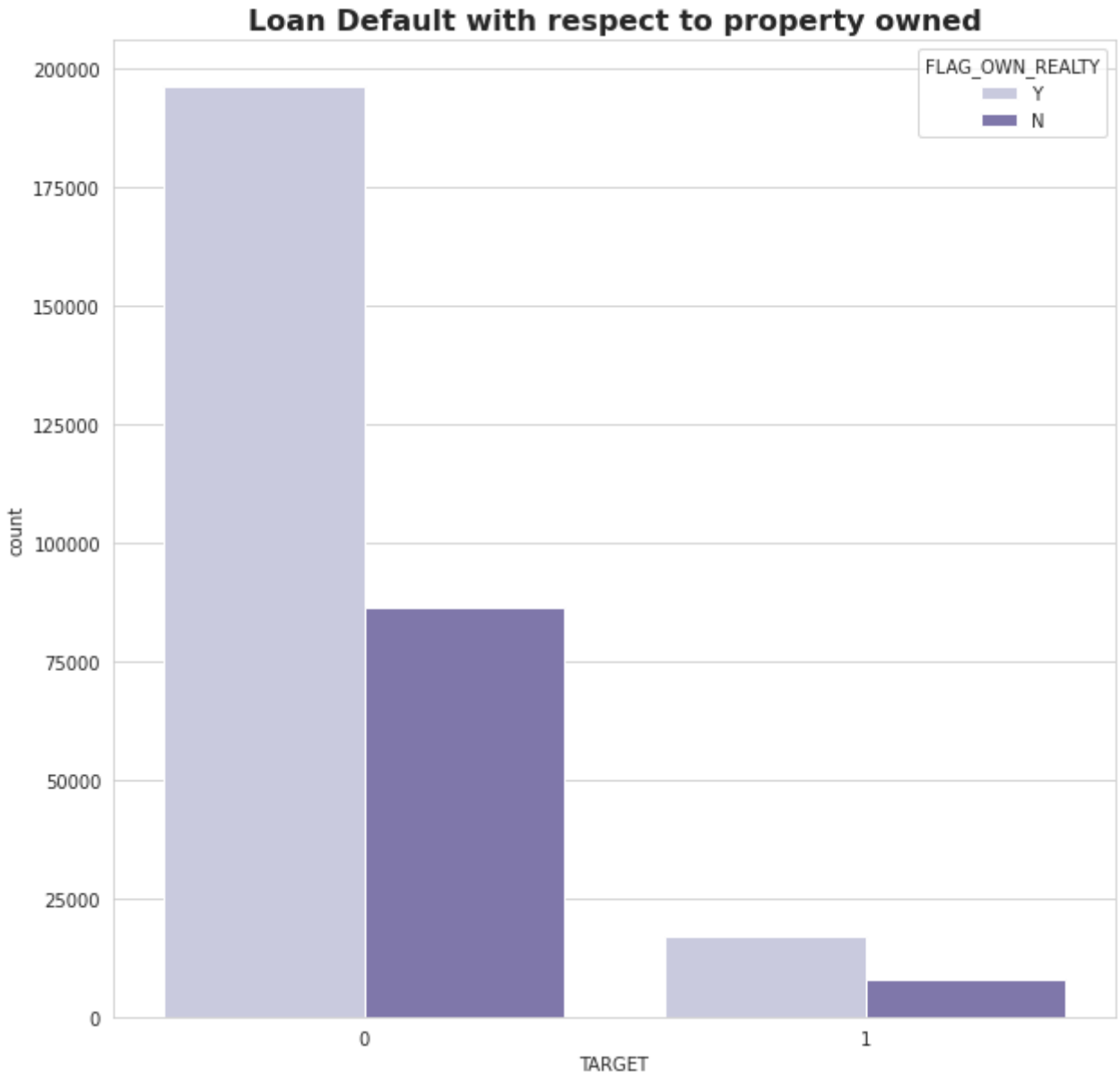


Insights:

It appears that cash loans are majority in defaulting the laons.

In [64]:
```
plt.figure(figsize=(10,10))
plt.title("Loan Default with respect to property owned",fontweight='bold' , font
sns.countplot(x='TARGET',hue='FLAG_OWN_REALTY',data=datasets['application_train'
```

Out[64]:
```
<AxesSubplot:title={'center':'Loan Default with respect to property owned'}, xla
bel='TARGET', ylabel='count'>
```

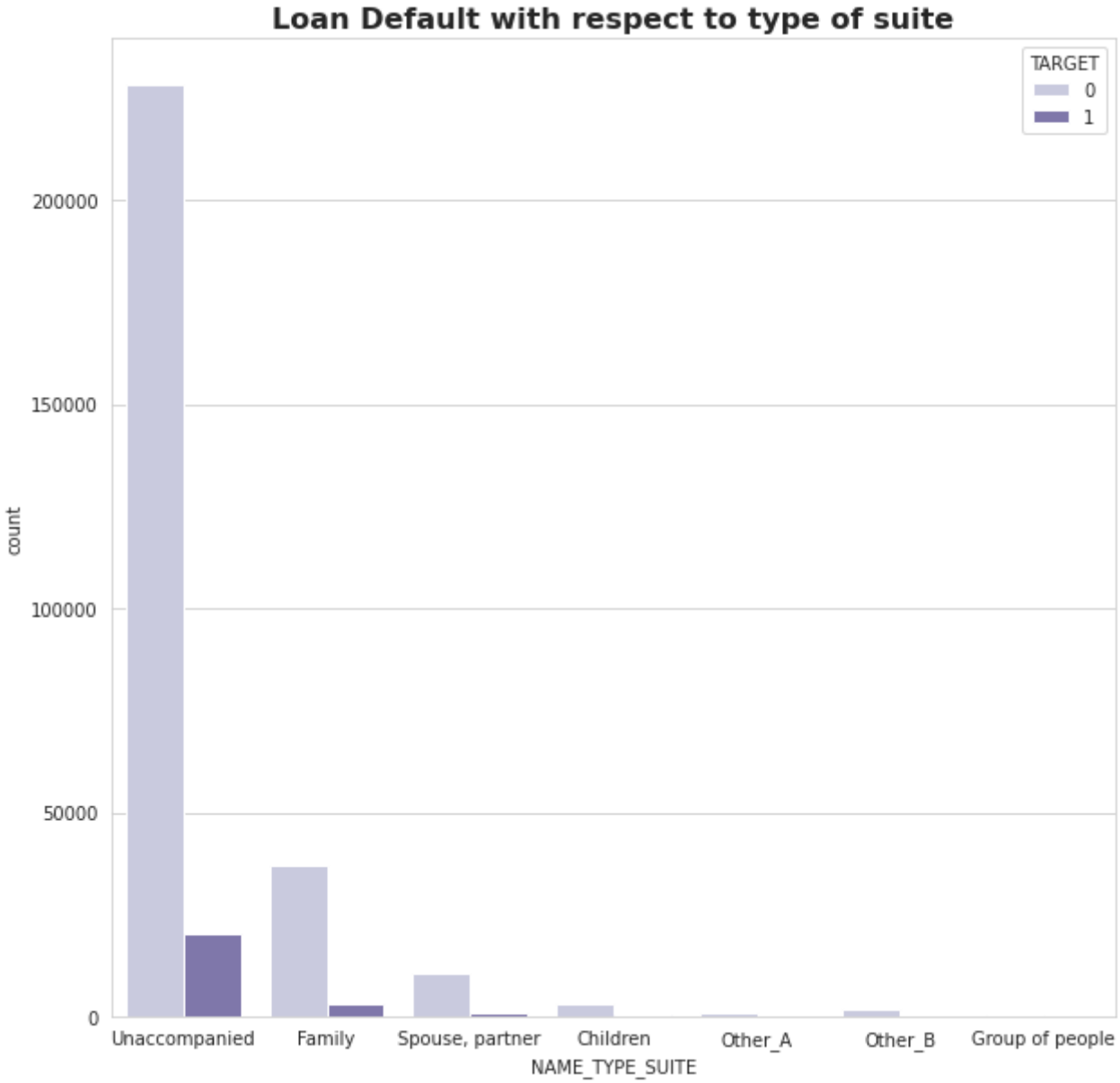


Insights:

It appears that there is not much difference in the percent of defaults with respect to the property owned.

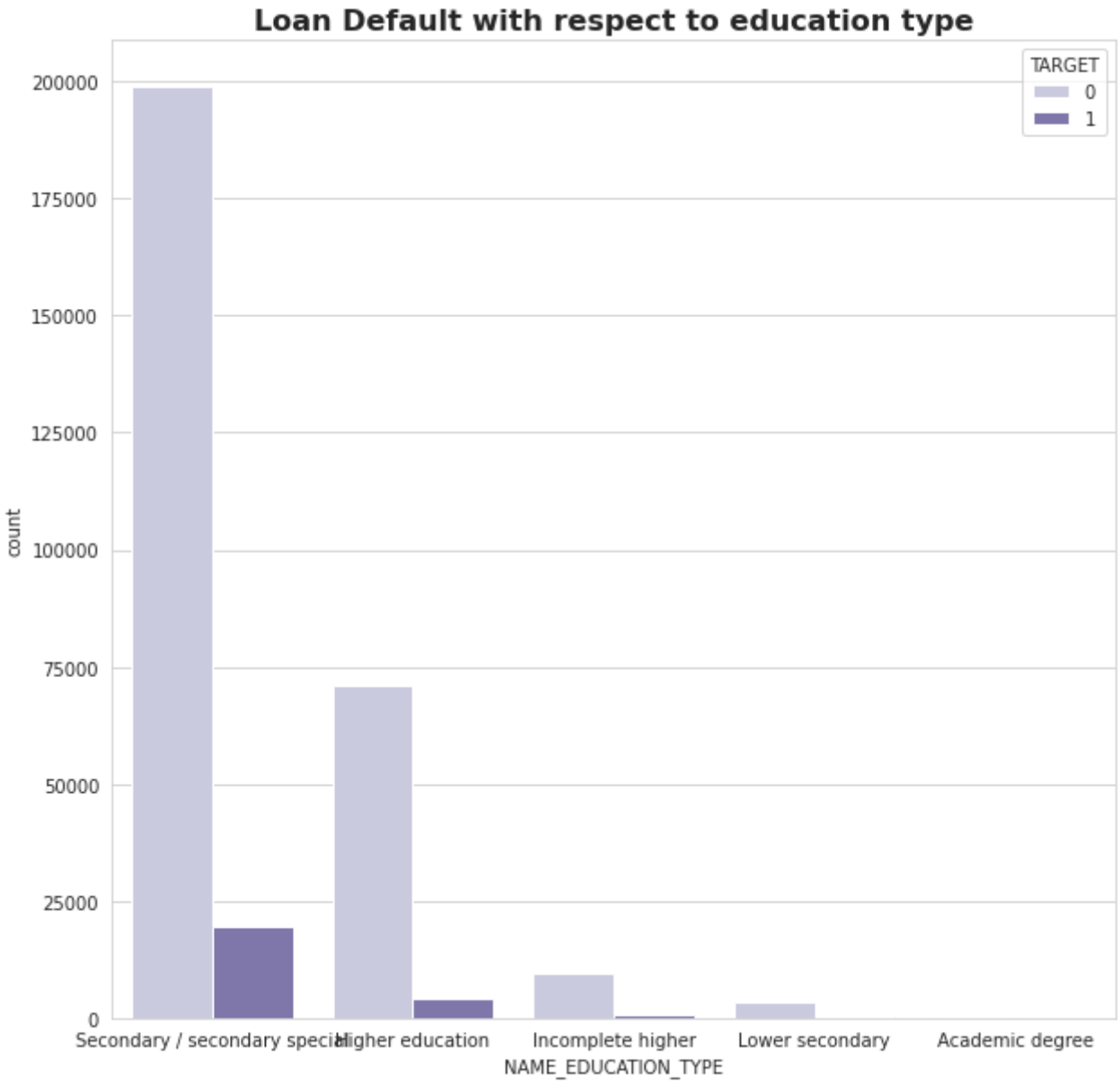
In [65]:
```
plt.figure(figsize=(10,10))
plt.title("Loan Default with respect to type of suite",fontweight='bold' , fonts
sns.countplot(x='NAME_TYPE_SUITE',hue='TARGET',data=datasets['application_train'
```

Out[65]:
```
<AxesSubplot:title={'center':'Loan Default with respect to type of suite'}, xlab
el='NAME_TYPE_SUITE', ylabel='count'>
```



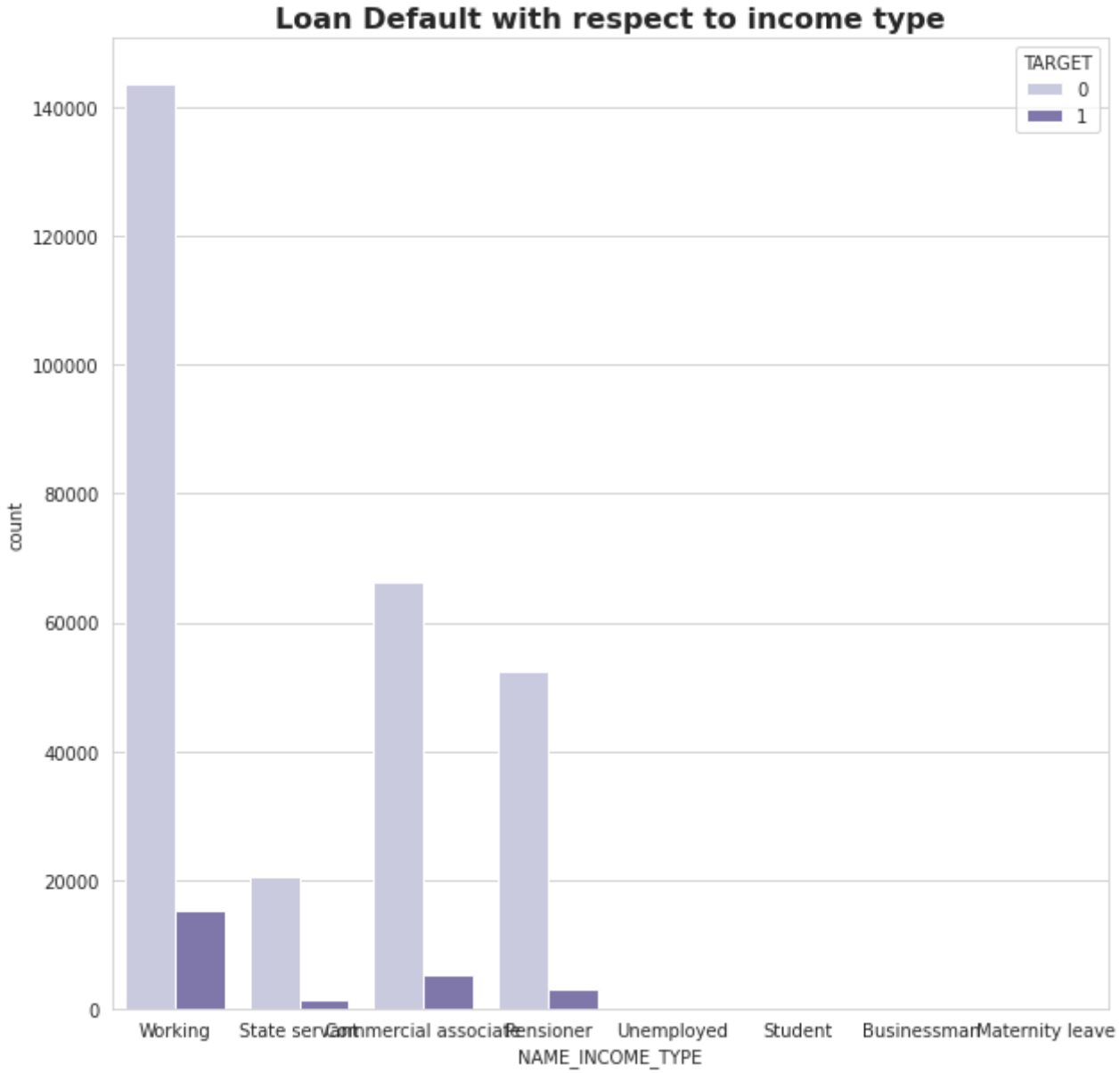
In [66]:
```
plt.figure(figsize=(10,10))
plt.title("Loan Default with respect to education type",fontweight='bold' , font
sns.countplot(x='NAME_EDUCATION_TYPE',hue='TARGET',data=datasets['application_tr
```

Out[66]:
```
<AxesSubplot:title={'center':'Loan Default with respect to education type'}, xla
bel='NAME_EDUCATION_TYPE', ylabel='count'>
```



In [67]:
```
plt.figure(figsize=(10,10))
plt.title("Loan Default with respect to income type",fontweight='bold' , fontsiz
sns.countplot(x='NAME_INCOME_TYPE',hue='TARGET',data=datasets['application_train
```

Out[67]:
```
<AxesSubplot:title={'center':'Loan Default with respect to income type'}, xlabel
='NAME_INCOME_TYPE', ylabel='count'>
```



In [68]:
```
datasets['application_train'].info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 52 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   SK_ID_CURR                   307511 non-null  int64
 1   TARGET                       307511 non-null  int64
 2   NAME_CONTRACT_TYPE           307511 non-null  object
 3   CODE_GENDER                  307511 non-null  object
 4   FLAG_OWN_CAR                 307511 non-null  object
 5   FLAG_OWN_REALTY              307511 non-null  object
 6   AMT_INCOME_TOTAL             307511 non-null  float64
 7   AMT_CREDIT                   307511 non-null  float64
 8   AMT_ANNUITY                  307499 non-null  float64
 9   NAME_TYPE_SUITE              306219 non-null  object
 10  NAME_INCOME_TYPE             307511 non-null  object
 11  NAME_EDUCATION_TYPE          307511 non-null  object
 12  NAME_FAMILY_STATUS           307511 non-null  object
 13  NAME_HOUSING_TYPE            307511 non-null  object
```

```
 14  REGION_POPULATION_RELATIVE   307511 non-null  float64
 15  DAYS_BIRTH                   307511 non-null  int64
 16  DAYS_EMPLOYED                307511 non-null  int64
 17  DAYS_REGISTRATION            307511 non-null  float64
 18  DAYS_ID_PUBLISH              307511 non-null  int64
 19  OWN_CAR_AGE                  104582 non-null  float64
 20  FLAG_EMP_PHONE               307511 non-null  int64
 21  FLAG_WORK_PHONE              307511 non-null  int64
 22  FLAG_PHONE                   307511 non-null  int64
 23  OCCUPATION_TYPE              211120 non-null  object
 24  CNT_FAM_MEMBERS              307509 non-null  float64
 25  REGION_RATING_CLIENT_W_CITY  307511 non-null  int64
 26  WEEKDAY_APPR_PROCESS_START   307511 non-null  object
 27  HOUR_APPR_PROCESS_START      307511 non-null  int64
 28  REG_CITY_NOT_WORK_CITY       307511 non-null  int64
 29  ORGANIZATION_TYPE            307511 non-null  object
 30  EXT_SOURCE_1                 134133 non-null  float64
 31  EXT_SOURCE_2                 306851 non-null  float64
 32  EXT_SOURCE_3                 246546 non-null  float64
 33  APARTMENTS_AVG               151450 non-null  float64
 34  BASEMENTAREA_AVG             127568 non-null  float64
 35  YEARS_BEGINEXPLUATATION_AVG  157504 non-null  float64
 36  ENTRANCES_AVG                152683 non-null  float64
 37  FLOORSMAX_AVG                154491 non-null  float64
 38  LANDAREA_AVG                 124921 non-null  float64
 39  FONDKAPREMONT_MODE           97216 non-null   object
 40  HOUSETYPE_MODE               153214 non-null  object
 41  WALLSMATERIAL_MODE           151170 non-null  object
 42  EMERGENCYSTATE_MODE          161756 non-null  object
 43  OBS_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 44  DEF_60_CNT_SOCIAL_CIRCLE     306490 non-null  float64
 45  FLAG_DOCUMENT_3              307511 non-null  int64
 46  FLAG_DOCUMENT_6              307511 non-null  int64
 47  FLAG_DOCUMENT_13             307511 non-null  int64
 48  FLAG_DOCUMENT_14             307511 non-null  int64
 49  FLAG_DOCUMENT_16             307511 non-null  int64
 50  AMT_REQ_CREDIT_BUREAU_MON    265992 non-null  float64
 51  AMT_REQ_CREDIT_BUREAU_YEAR   265992 non-null  float64
dtypes: float64(20), int64(16), object(16)
memory usage: 122.0+ MB
```

Seperating categorical and numerical data for Visualization

It appears that there is not much difference in the percent of defaults with respect to the property owned.

In [69]:
```
categorical_df= datasets['application_train'].select_dtypes(include='object')
categorical_df
```

Out[69]:

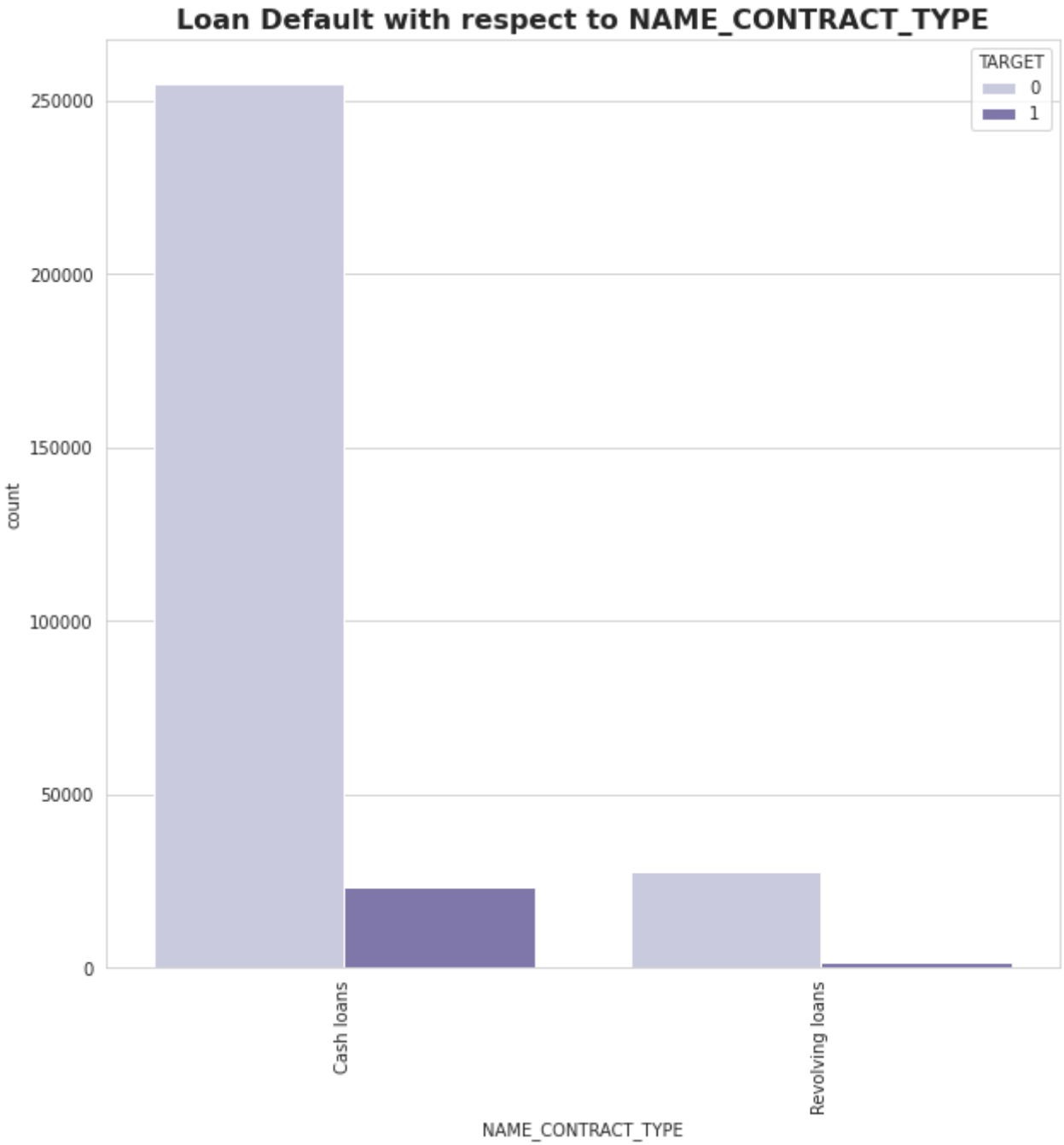
	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	NAME_T
0	Cash loans	M	N	Y	Una
1	Cash loans	F	N	N	

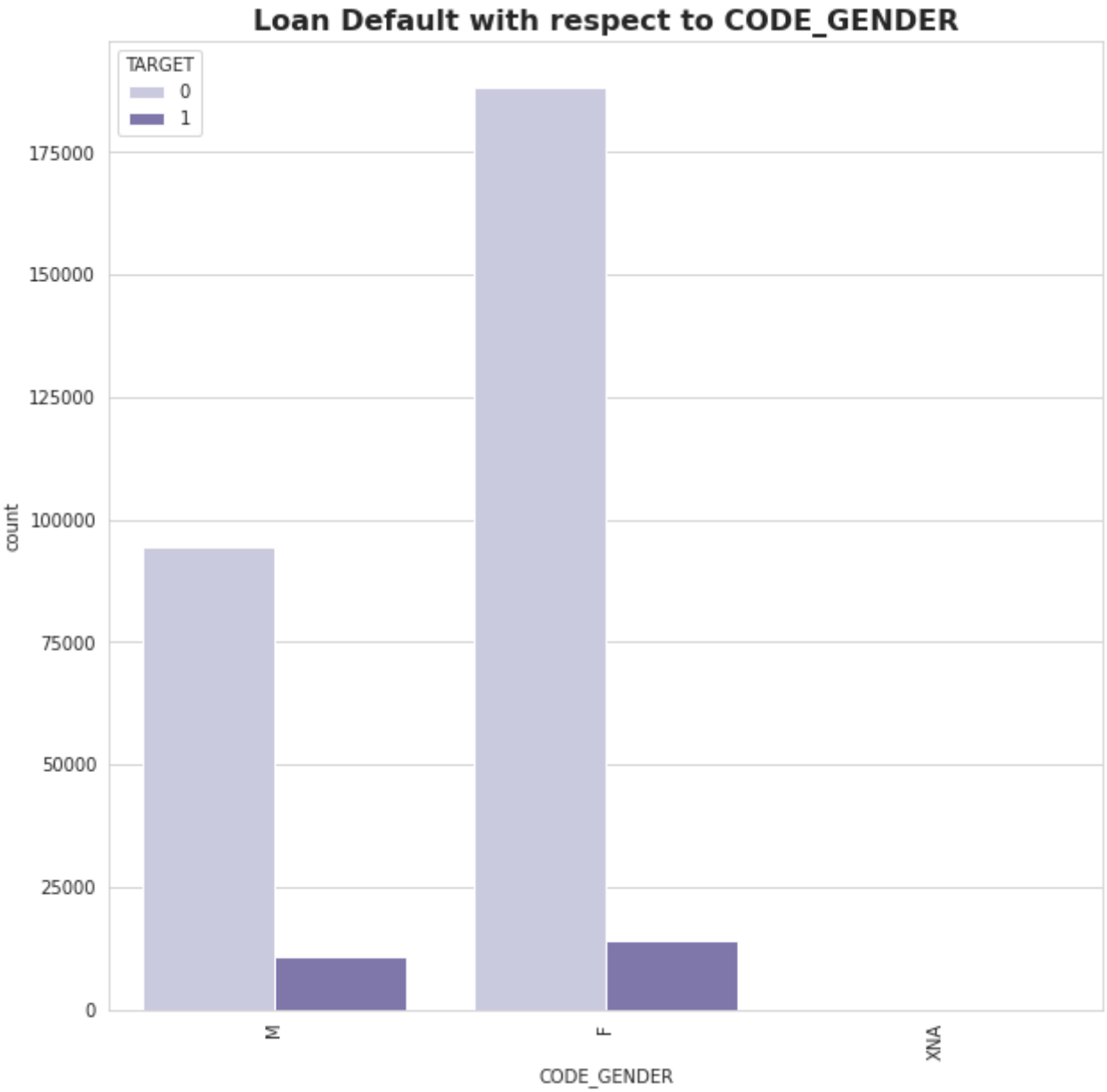
	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	NAME_T
2	Revolving loans	M	Y	Y	Una
3	Cash loans	F	N	Y	Una
4	Cash loans	M	N	Y	Una
...	...	...	...	...	
307506	Cash loans	M	N	N	Una
307507	Cash loans	F	N	Y	Una
307508	Cash loans	F	N	Y	Una
307509	Cash loans	F	N	Y	Una
307510	Cash loans	F	N	N	Una

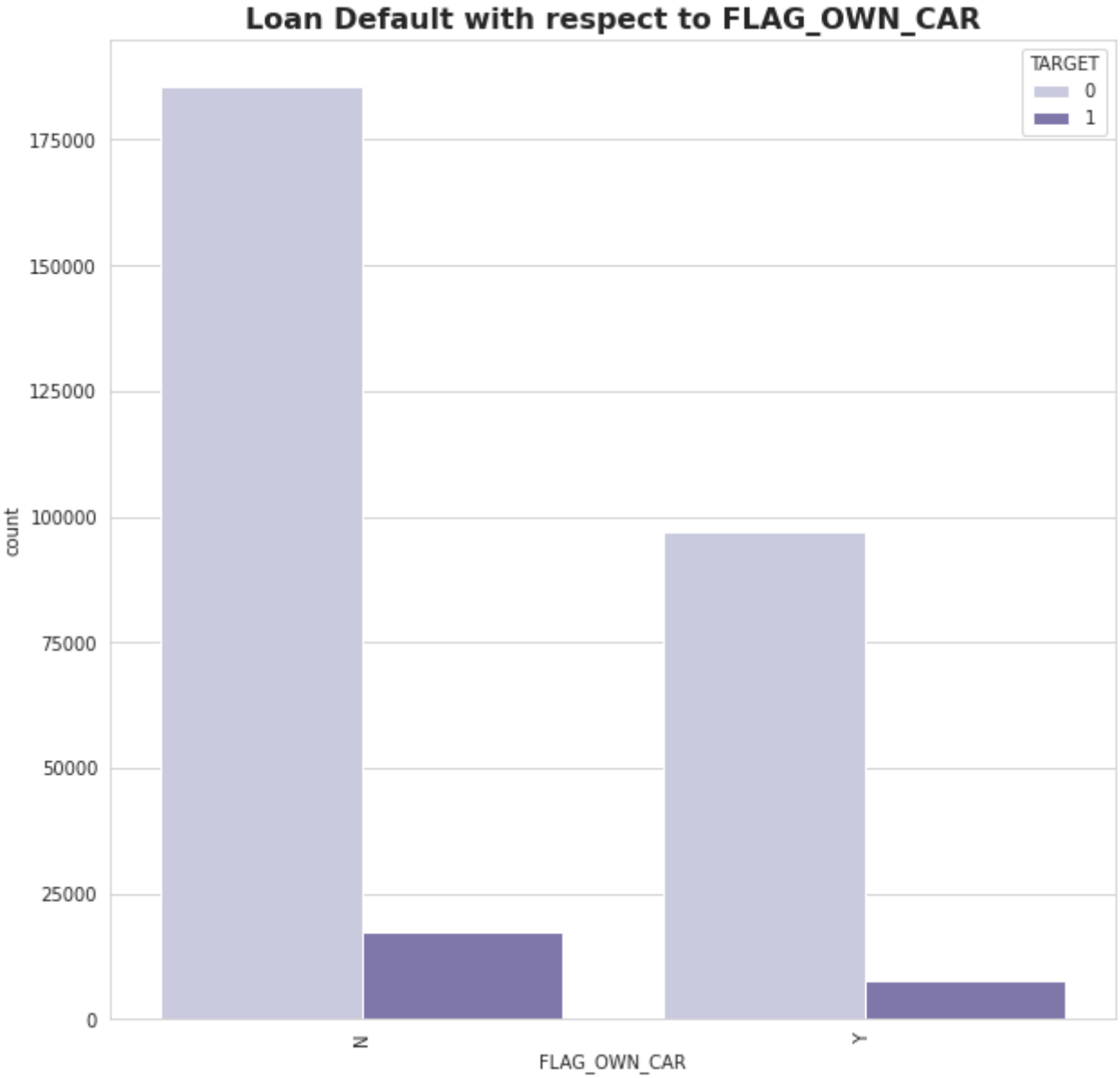
307511 rows × 16 columns

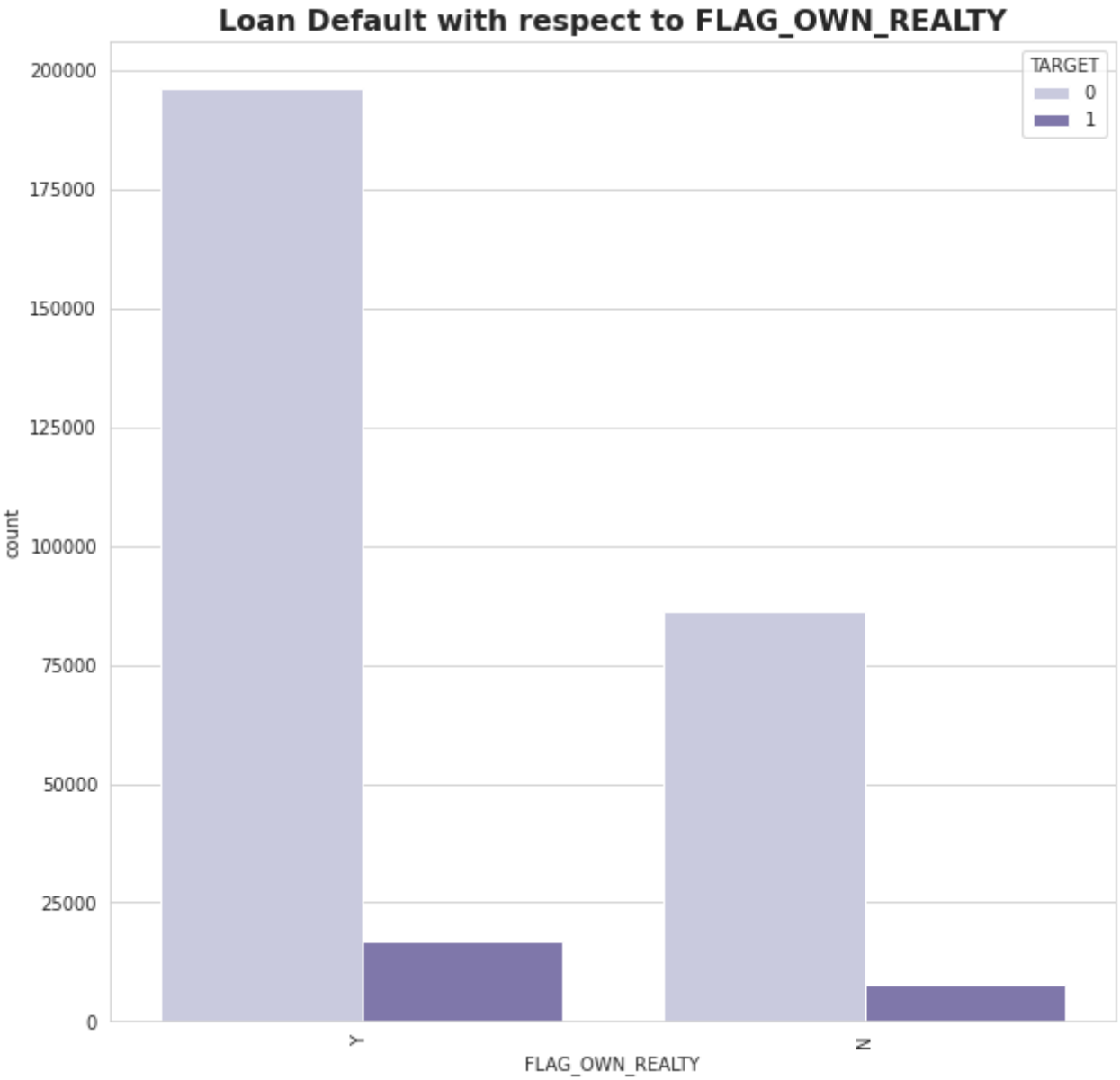
In [70]:

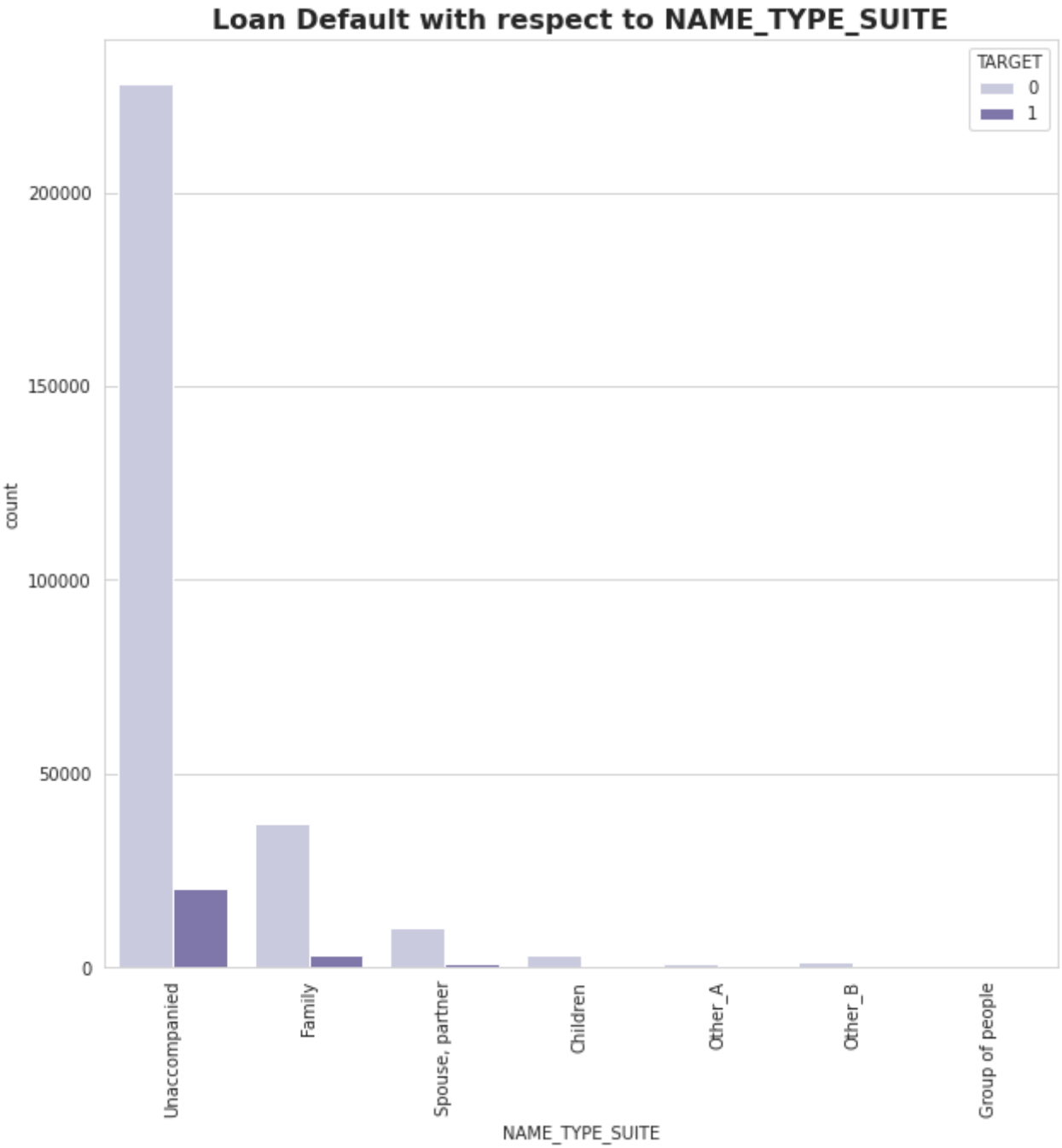
```
for cat in categorical_df.columns.values[:11]:
        plt.figure(figsize=(10,10))
        title= "Loan Default with respect to "+ str(cat)
        plt.title(title,fontweight='bold' , fontsize =16)
        sns.countplot(x=categorical_df[cat],hue='TARGET',data=datasets['applicat
        plt.xticks(rotation=90)
```

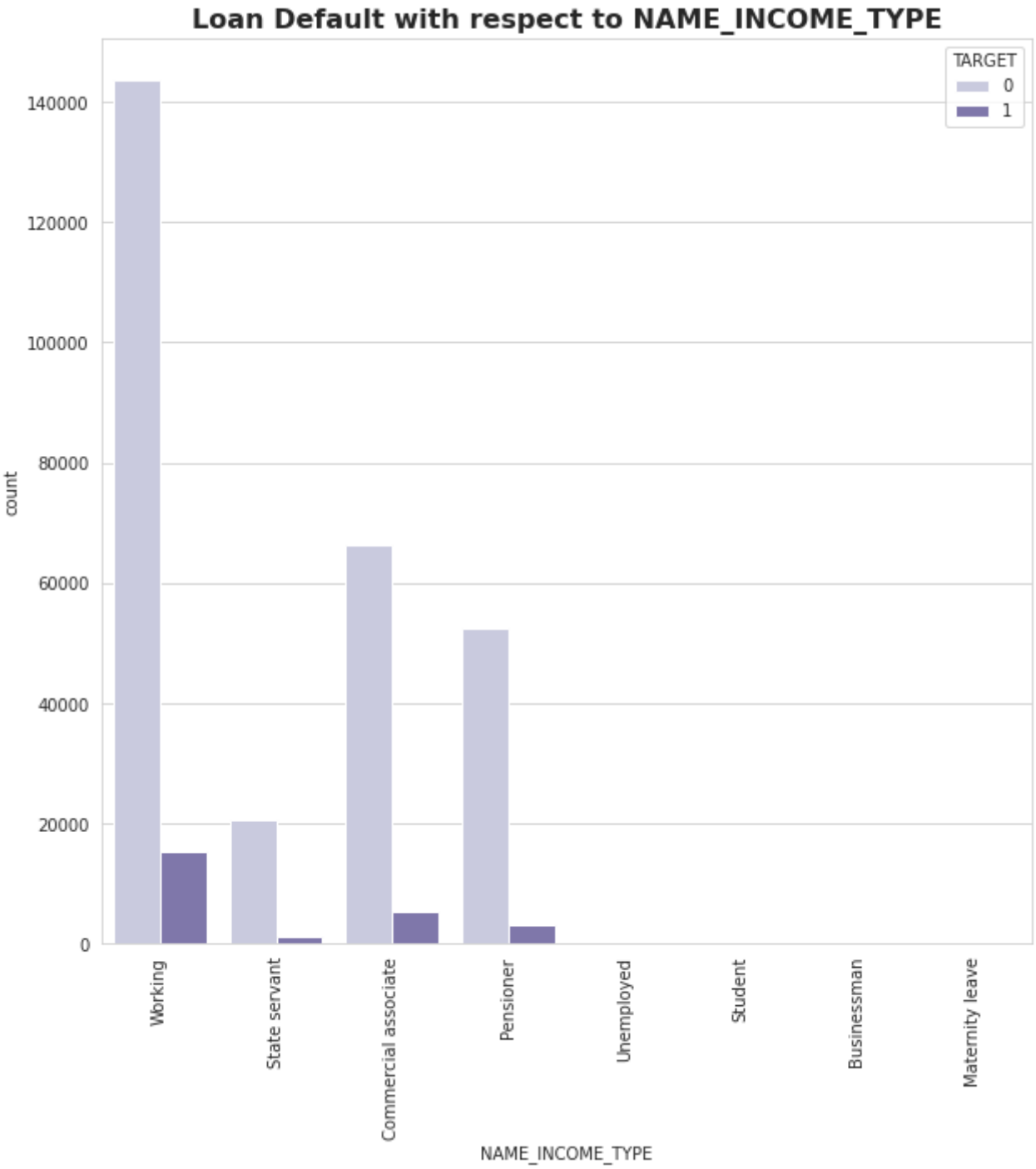

Loan Default with respect to NAME_CONTRACT_TYPE
TARGET
0
1
250000
200000
150000
100000
50000
0
count
Cash loans
Revolving loans
NAME_CONTRACT_TYPE

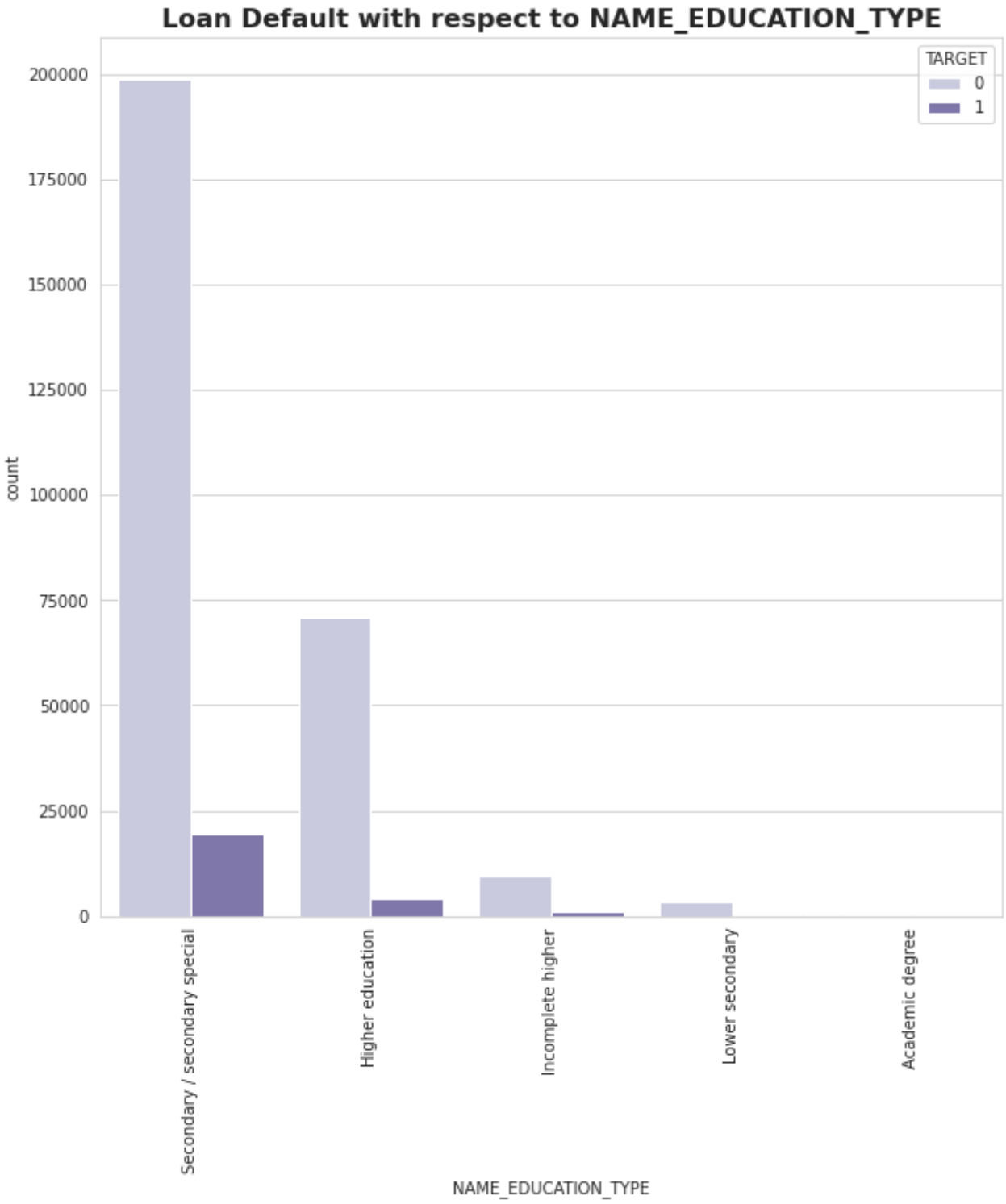

Loan Default with respect to CODE_GENDER
TARGET
0
1
175000
150000
125000
100000
75000
50000
25000
0
count
M
F
XNA
CODE_GENDER

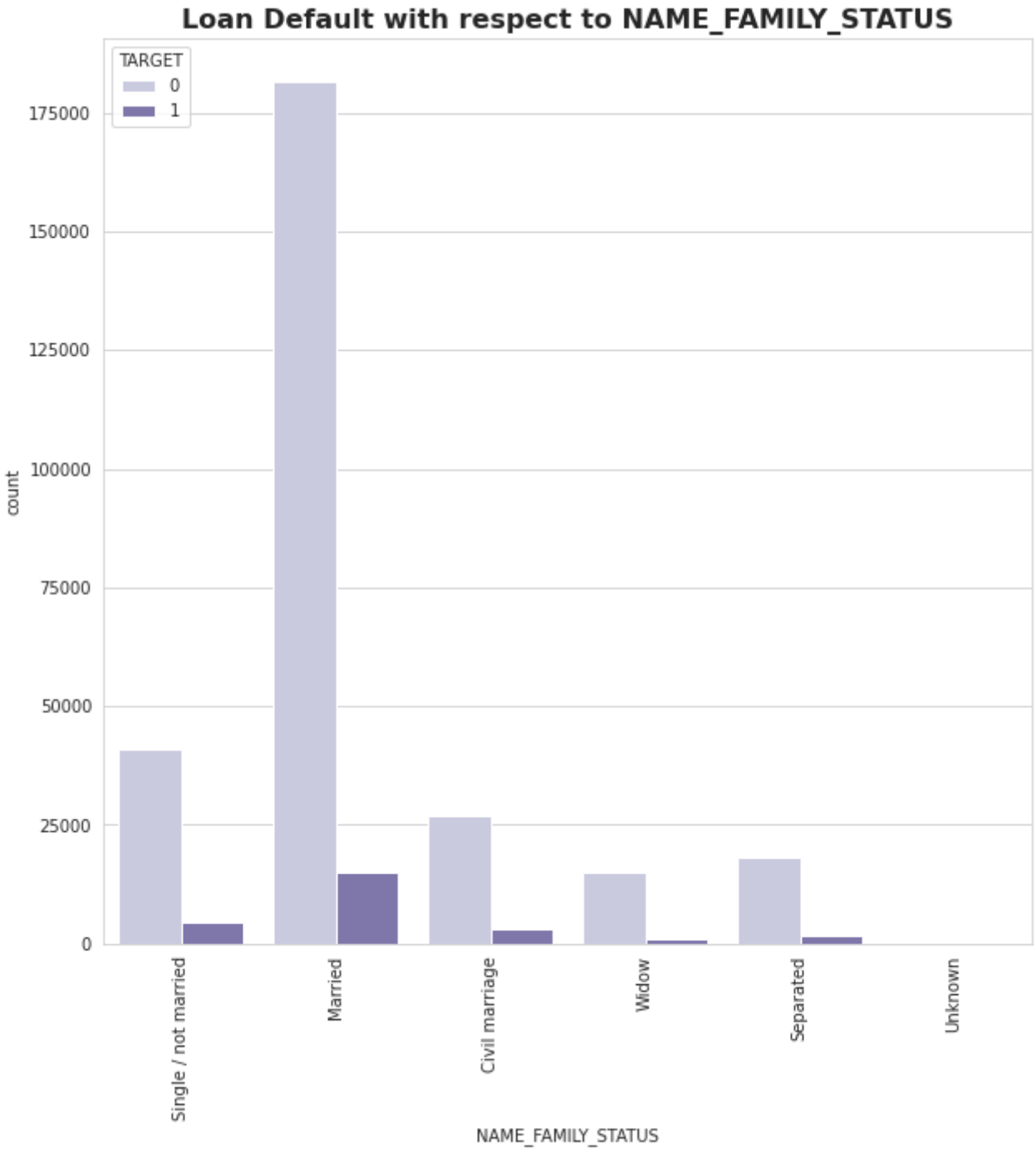

Loan Default with respect to FLAG_OWN_CAR
TARGET
0
1
175000
150000
125000
100000
75000
50000
25000
0
count
N
Y
FLAG_OWN_CAR

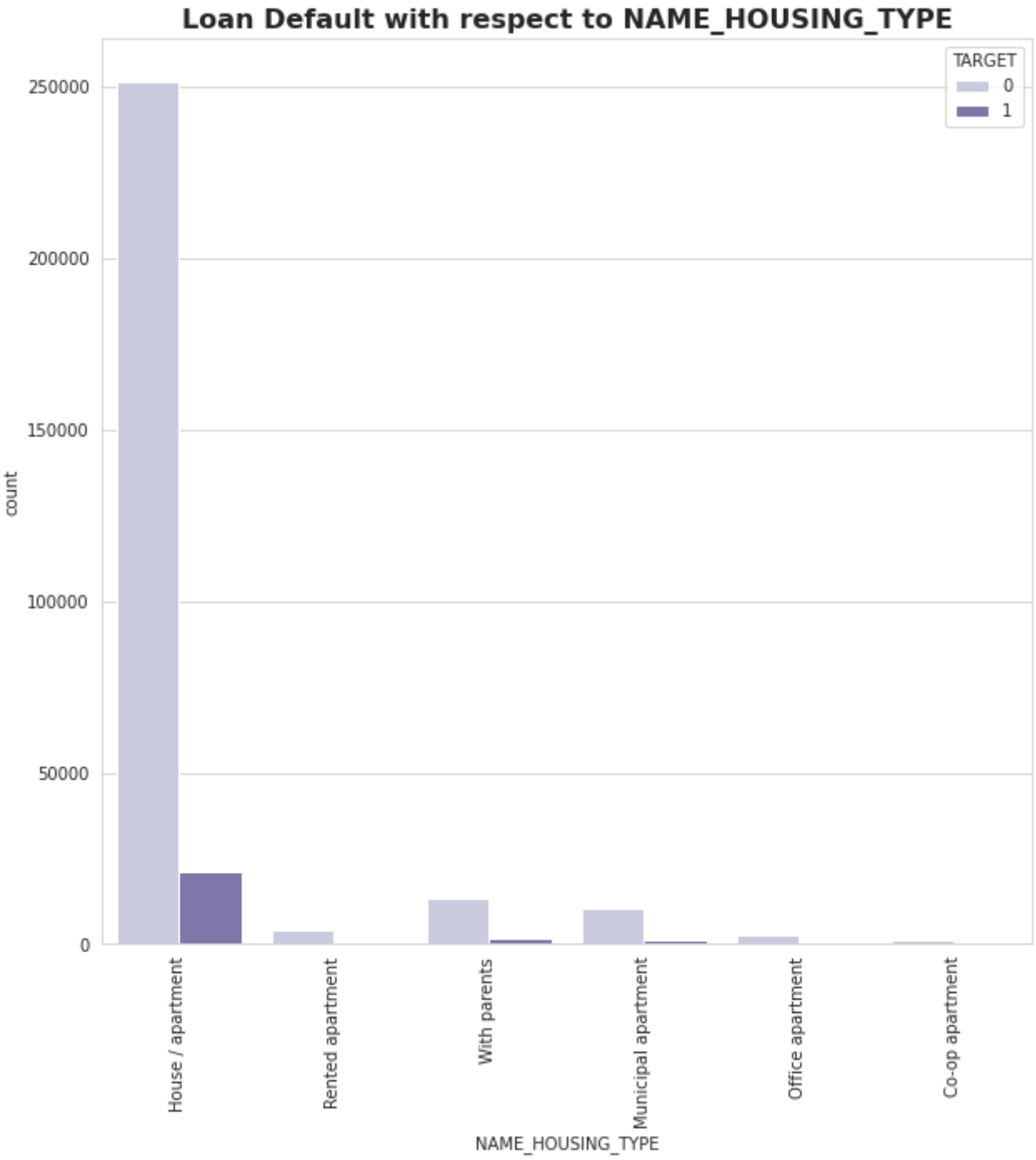

Loan Default with respect to FLAG_OWN_REALTY
TARGET
0
1
count
200000
175000
150000
125000
100000
75000
50000
25000
0
Y
N
FLAG_OWN_REALTY

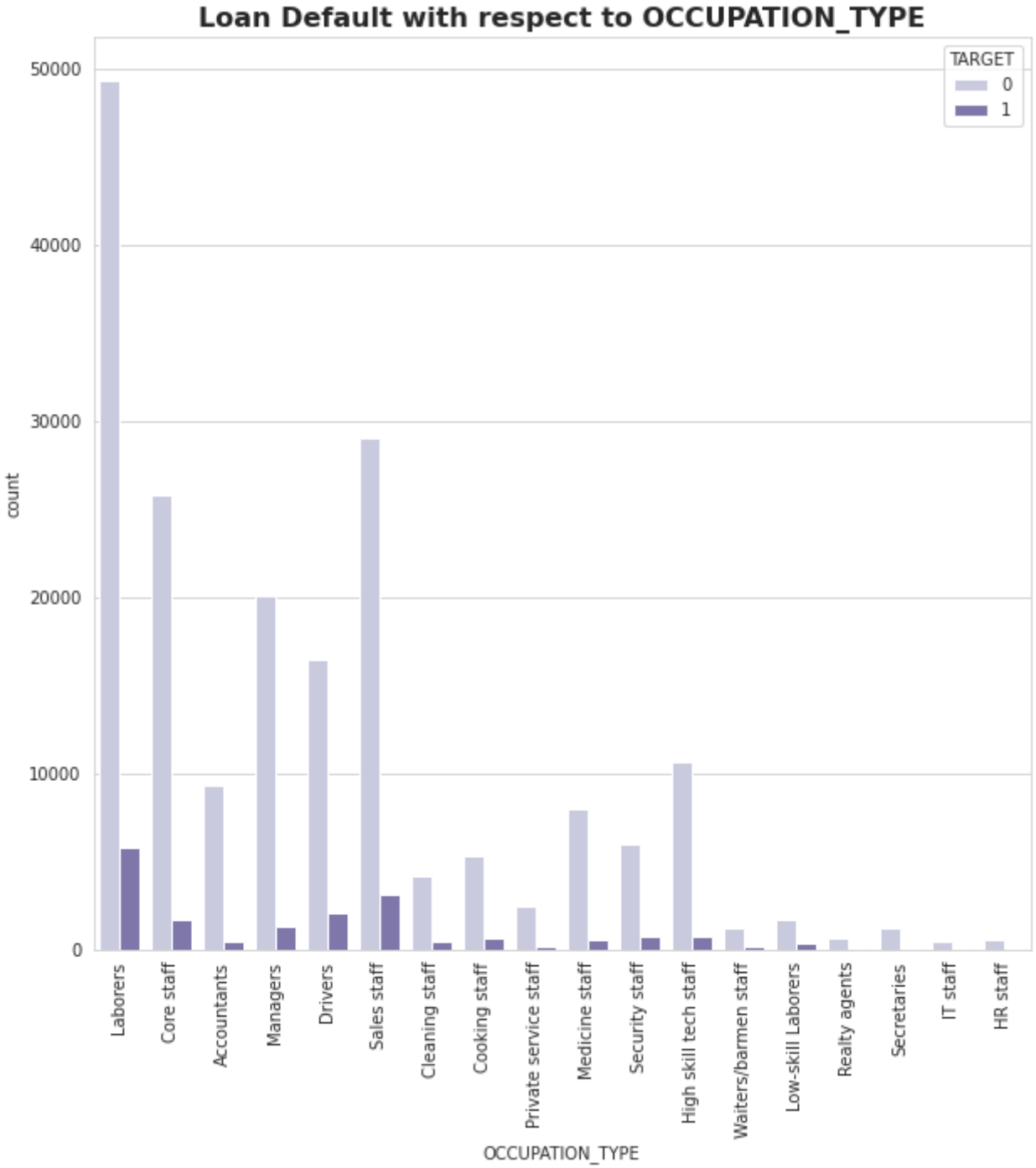

Loan Default with respect to NAME_TYPE_SUITE
TARGET
0
1
200000
150000
100000
50000
0
count
Unaccompanied
Family
Spouse, partner
Children
Other_A
Other_B
Group of people
NAME_TYPE_SUITE

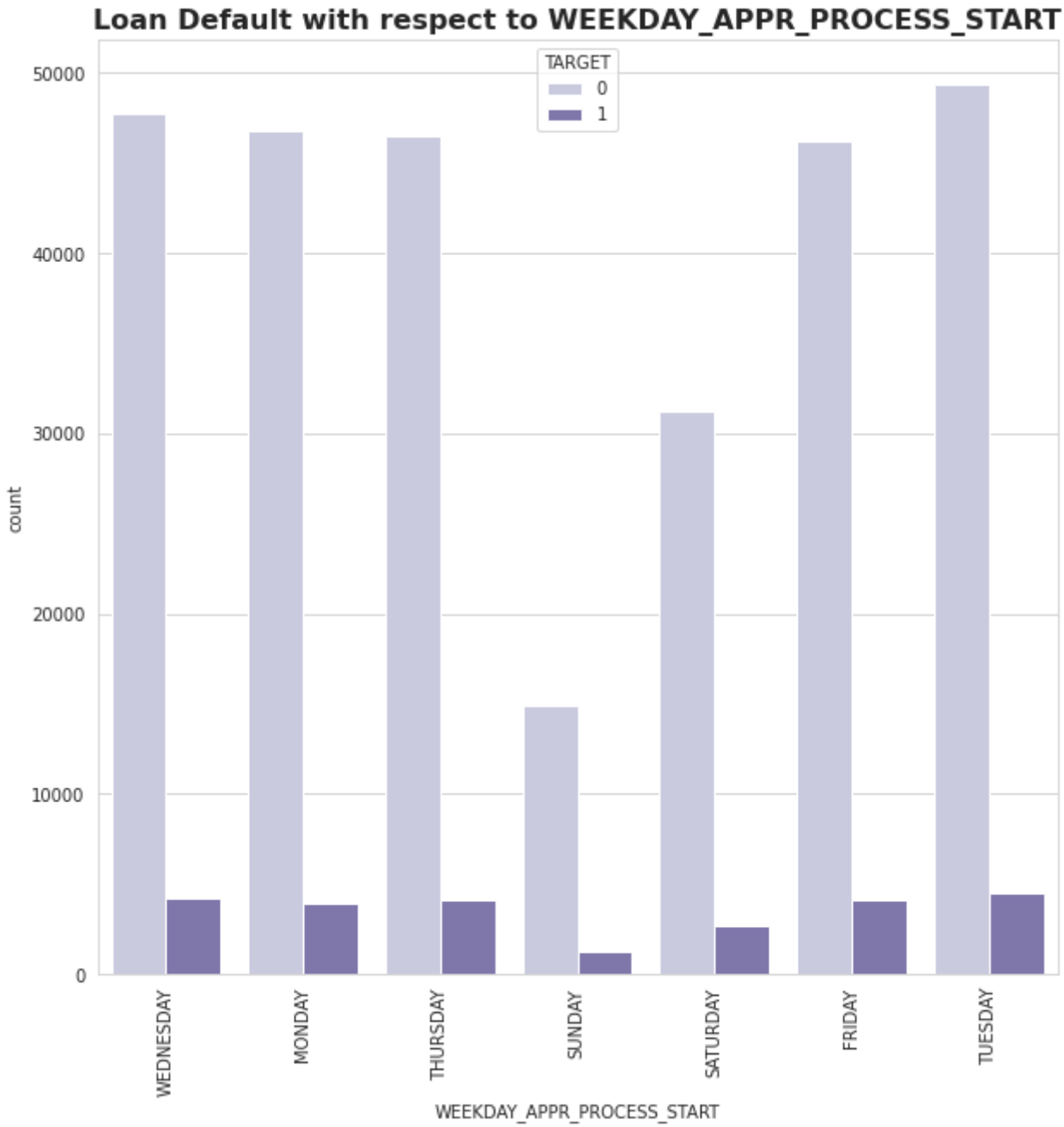

Loan Default with respect to NAME_INCOME_TYPE
TARGET
0
1
count
140000
120000
100000
80000
60000
40000
20000
0
Working
State servant
Commercial associate
Pensioner
Unemployed
Student
Businessman
Maternity leave
NAME_INCOME_TYPE


Loan Default with respect to NAME_EDUCATION_TYPE
TARGET
0
1
count
200000
175000
150000
125000
100000
75000
50000
25000
0
Secondary / secondary special
Higher education
Incomplete higher
Lower secondary
Academic degree
NAME_EDUCATION_TYPE


Loan Default with respect to NAME_FAMILY_STATUS
TARGET
0
1
count
175000
150000
125000
100000
75000
50000
25000
0
Single / not married
Married
Civil marriage
Widow
Separated
Unknown
NAME_FAMILY_STATUS


Loan Default with respect to NAME_HOUSING_TYPE
TARGET
0
1
250000
200000
150000
100000
50000
0
count
House / apartment
Rented apartment
With parents
Municipal apartment
Office apartment
Co-op apartment
NAME_HOUSING_TYPE


Loan Default with respect to OCCUPATION_TYPE
TARGET
0
1
count
50000
40000
30000
20000
10000
0
Laborers
Core staff
Accountants
Managers
Drivers
Sales staff
Cleaning staff
Cooking staff
Private service staff
Medicine staff
Security staff
High skill tech staff
Waiters/barmen staff
Low-skill Laborers
Realty agents
Secretaries
IT staff
HR staff
OCCUPATION_TYPE



In [71]:
```
integer_df1= datasets['application_train'].select_dtypes(include='int64')
integer_df1
float_df1 = datasets['application_train'].select_dtypes(include='float64')
```

In [72]:
```
integer_df1
```

Out[72]:

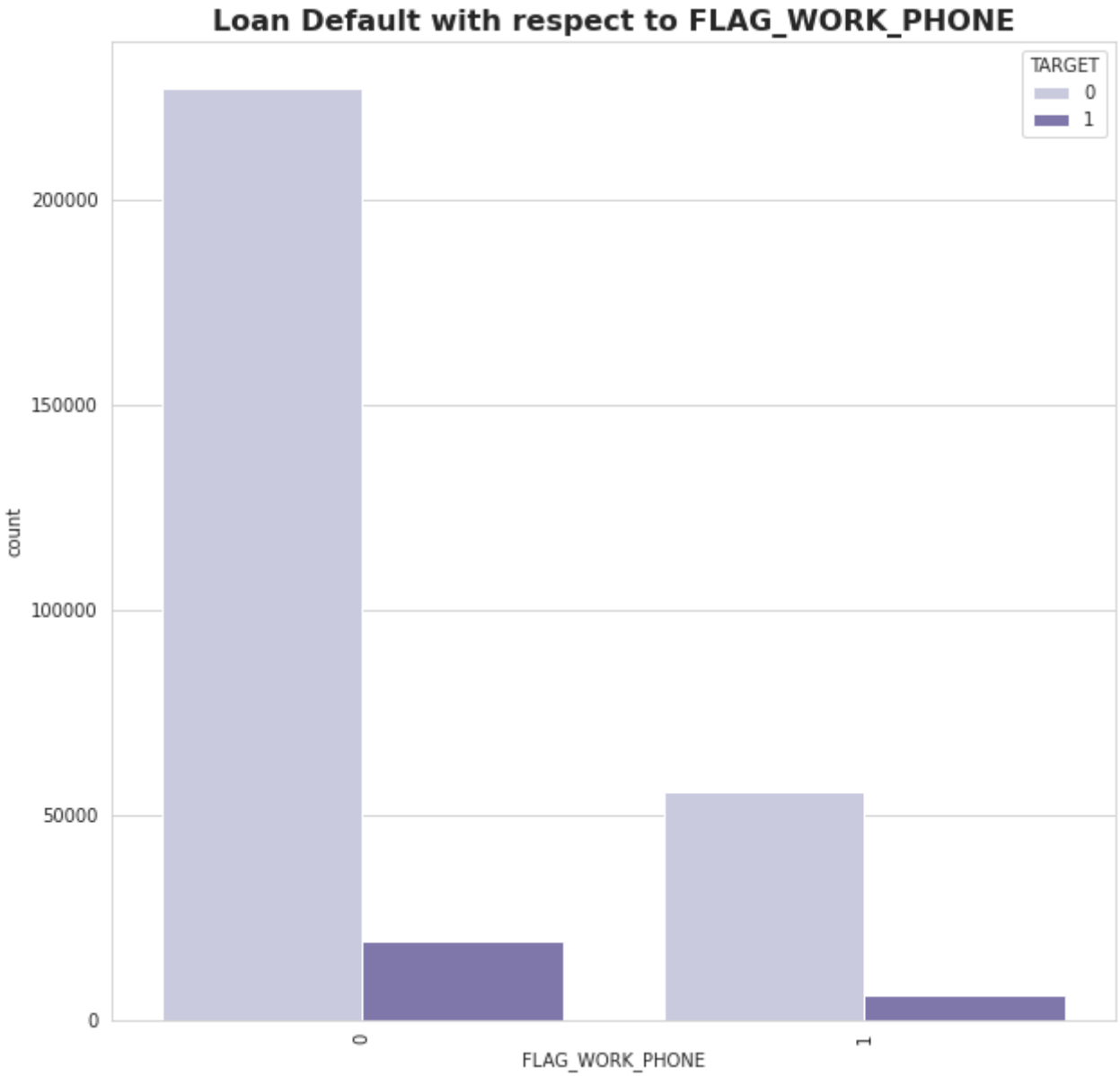
	SK_ID_CURR	TARGET	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_ID_PUBLISH	FLAG_EMP_P
0	100002	1	-9461	-637	-2120	
1	100003	0	-16765	-1188	-291	
2	100004	0	-19046	-225	-2531	
3	100006	0	-19005	-3039	-2437	
4	100007	0	-19932	-3038	-3458	

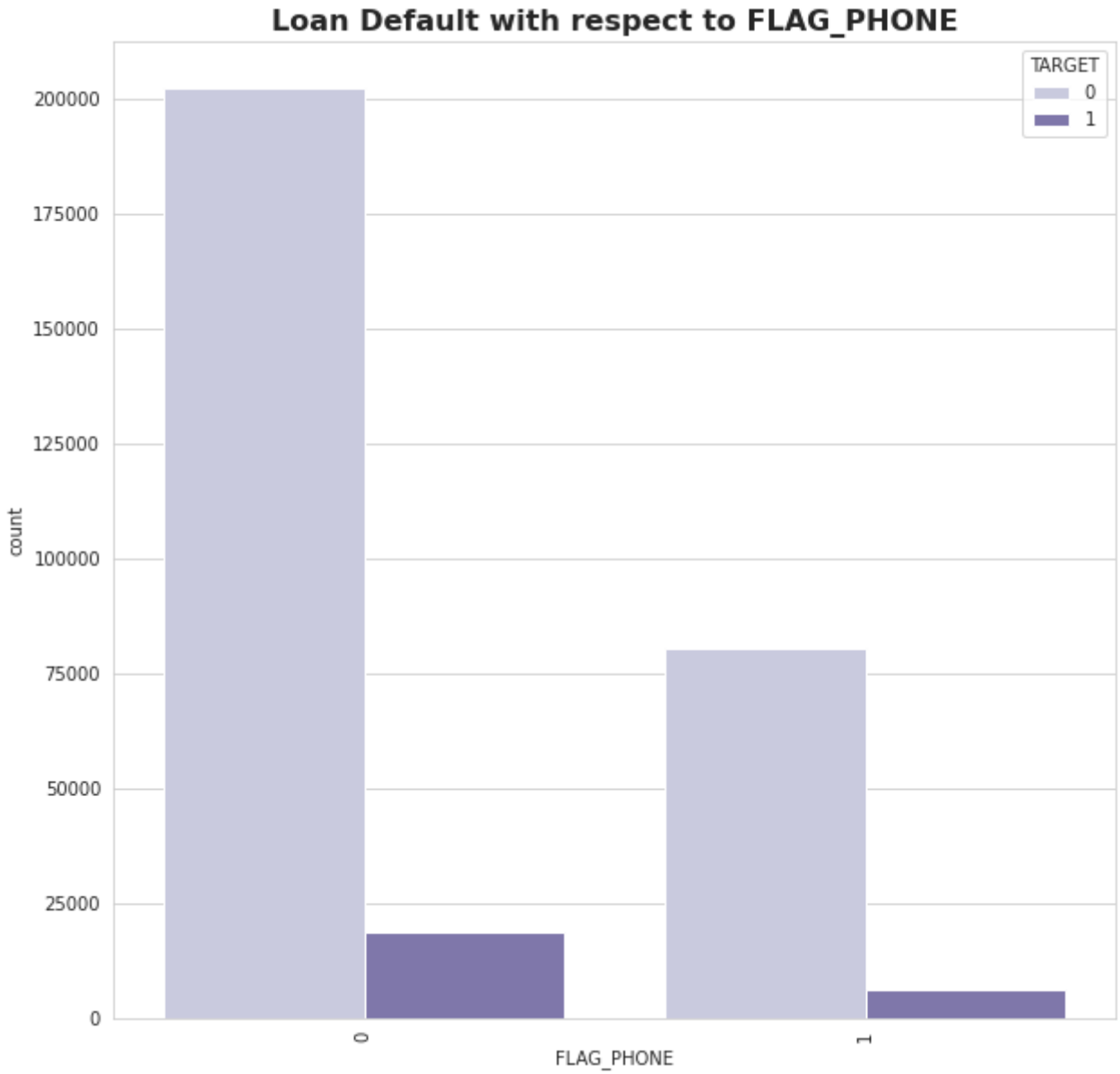
	SK_ID_CURR	TARGET	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_ID_PUBLISH	FLAG_EMP_P
...	...	...	...	...	...	
307506	456251	0	-9327	-236	-1982	
307507	456252	0	-20775	365243	-4090	
307508	456253	0	-14966	-7921	-5150	
307509	456254	1	-11961	-4786	-931	
307510	456255	0	-16856	-1262	-410	

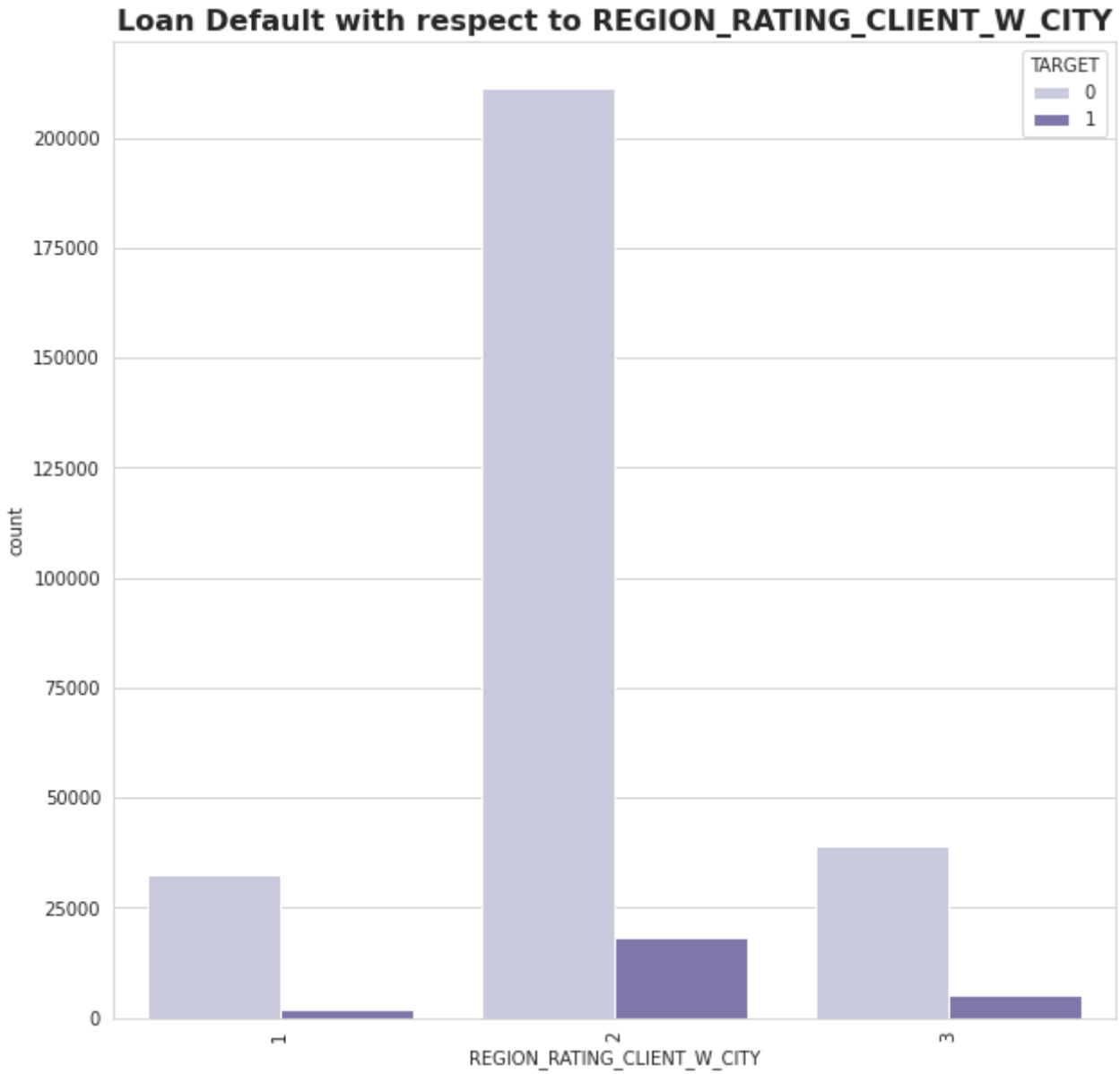
307511 rows × 16 columns

In [73]:

```
for integer in integer_df1.columns.values[6:9]:
        plt.figure(figsize=(10,10))
        title= "Loan Default with respect to "+ str(integer)
        plt.title(title,fontweight='bold' , fontsize =16)
        sns.countplot(x=integer_df1[integer],hue='TARGET',data=datasets['applica
        plt.xticks(rotation=90)
```


Loan Default with respect to FLAG_WORK_PHONE
TARGET
0
1
200000
150000
100000
50000
0
count
0
1
FLAG_WORK_PHONE

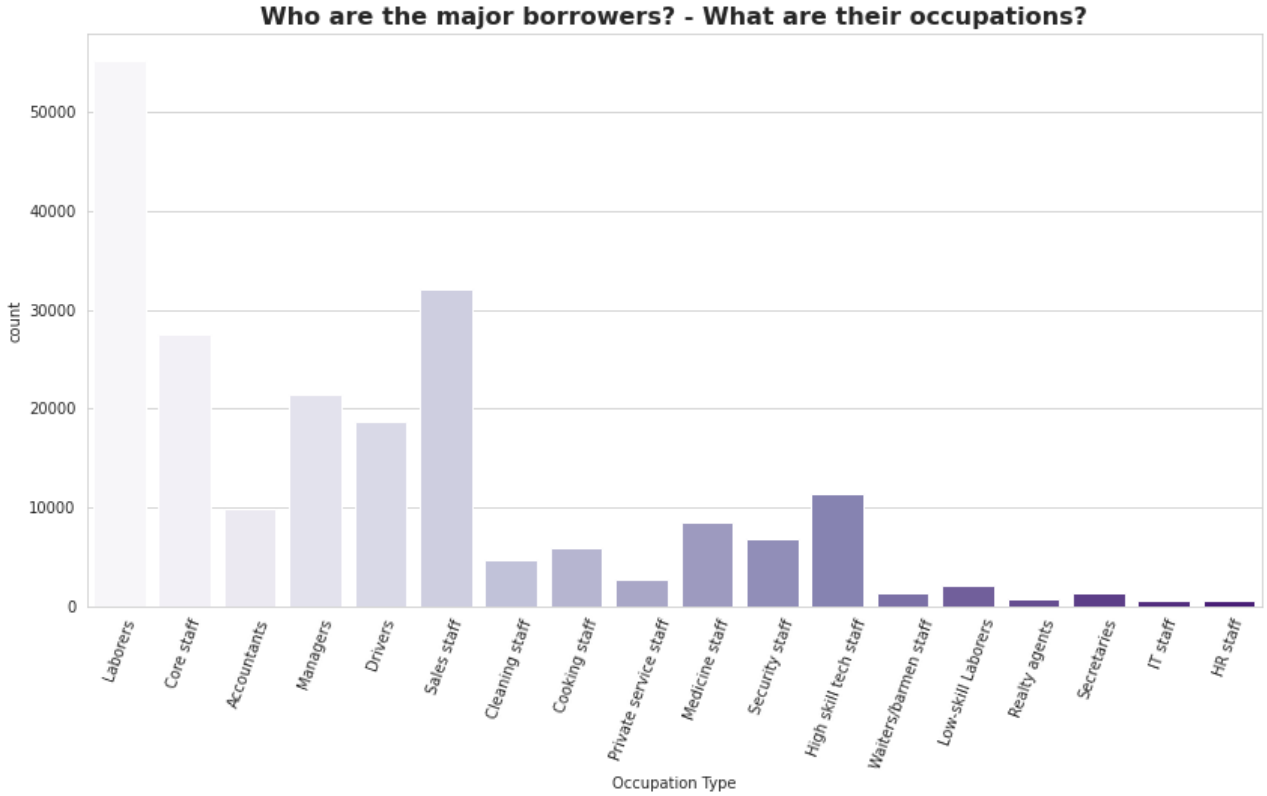

Loan Default with respect to FLAG_PHONE
TARGET
0
1
200000
175000
150000
125000
100000
75000
50000
25000
0
count
0
1
FLAG_PHONE



Who are major borrowers and what are their occupations

In [74]:

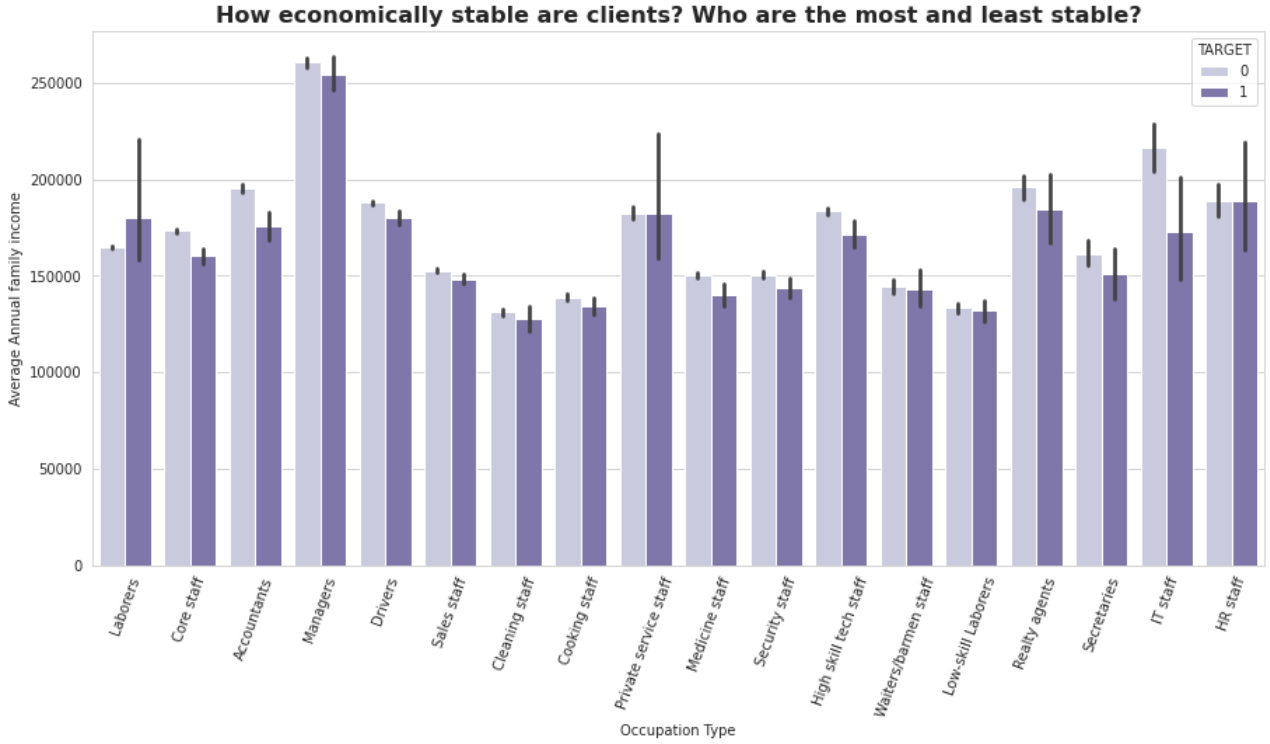
```
fig, ax = plt.subplots(figsize=(14,7))
sns.countplot(x='OCCUPATION_TYPE',data=datasets['application_train'], palette =
plt.xlabel("Occupation Type")
plt.title("Who are the major borrowers? - What are their occupations?", fontweig
plt.xticks(rotation=70)
plt.show()
```



How economically stable are clients? Who are the most and least stable?

In [75]:
```
fig, ax = plt.subplots(figsize=(15,7))
sns.barplot(x='OCCUPATION_TYPE',y='AMT_INCOME_TOTAL',hue='TARGET',data=datasets[
plt.xticks(rotation=70)
plt.xlabel("Occupation Type")
plt.ylabel("Average Annual family income")
plt.title("How economically stable are clients? Who are the most and least stabl
```

Out[75]:
```
Text(0.5, 1.0, 'How economically stable are clients? Who are the most and least
stable?')
```



Value Counts in Categorical Features

In [76]:
```
datasets['application_train'].select_dtypes('object').apply(pd.Series.nunique, a
```

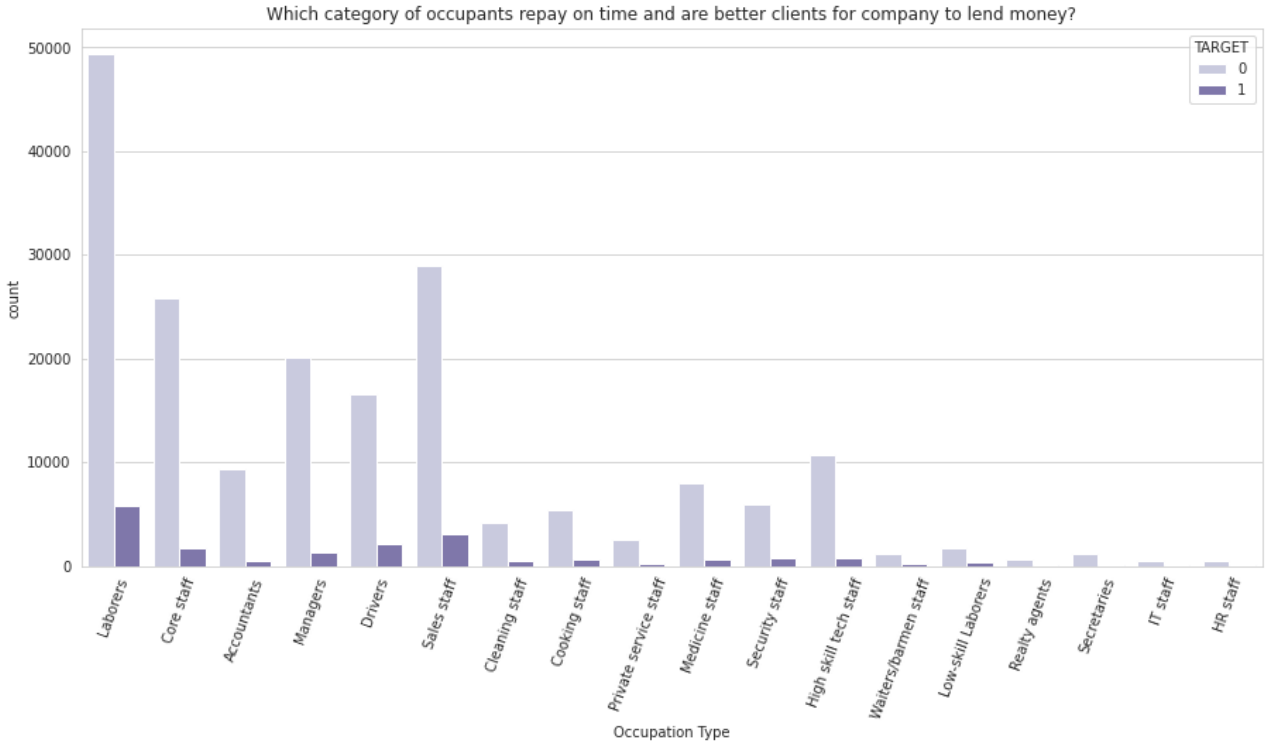
Out[76]:
```
NAME_CONTRACT_TYPE              2
CODE_GENDER                     3
FLAG_OWN_CAR                    2
FLAG_OWN_REALTY                 2
NAME_TYPE_SUITE                 7
NAME_INCOME_TYPE                8
NAME_EDUCATION_TYPE             5
NAME_FAMILY_STATUS              6
NAME_HOUSING_TYPE               6
OCCUPATION_TYPE                18
WEEKDAY_APPR_PROCESS_START      7
ORGANIZATION_TYPE              58
FONDKAPREMONT_MODE              4
HOUSETYPE_MODE                  3
WALLSMATERIAL_MODE              7
EMERGENCYSTATE_MODE             2
dtype: int64
```

Which category of occupants repay on time and are better clients for company to lend money?'

In [77]:
```
fig, ax = plt.subplots(figsize=(15,7))
sns.countplot(x='OCCUPATION_TYPE',hue='TARGET',data=datasets['application_train'
plt.xticks(rotation=70)
plt.xlabel("Occupation Type")
plt.title('Which category of occupants repay on time and are better clients for
```

```
Text(0.5, 1.0, 'Which category of occupants repay on time and are better clients
```

Out[77]:
```
for company to lend money?')
```



Preprocessing

Data preprocessing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we preprocess our data before feeding it into our model. The concepts that I will cover for this section of the project are-

- Handling Null Values
- Standardization
- Handling Categorical Variables
- One-Hot Encoding

Importing the necessary modules

In [78]:
```
from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.compose import ColumnTransformer
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.model_selection import train_test_split  # sklearn.cross_validation
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from time import time
```

In [79]:
```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import log_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
```

Splitting the data into train and test Data

> We are going to split the data into train and test Data so that we can perform a check on our model afterwards.

In [80]:
```
from sklearn.model_selection import train_test_split
```

In [81]:
```
datasets['application_train'].head()
```

Out[81]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
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	

5 rows × 52 columns

In [82]:
```
X = datasets['application_train'].drop(['TARGET'],axis=1)
y = datasets['application_train']['TARGET']
```

In [83]:
```
y = pd.DataFrame(y, columns = ["TARGET"])
y
```

Out[83]:

	TARGET
0	1
1	0
2	0
3	0
4	0
...	...
307506	0
307507	0
307508	0

	TARGET
307509	1
307510	0

307511 rows × 1 columns

In [84]:
```
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.2,random_stat
```

Seperating numerical and categorical variables to fit in Pipeline

In [85]:
```
integer_df= X.select_dtypes(include='int64')
float_df = X.select_dtypes(include='float64')
numerical_df = list(pd.concat([integer_df,float_df], axis=1))
categorical_df = list(X.select_dtypes(include='object'))
print(categorical_df)
print('                                    ')
print(numerical_df)
```

```
['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_T
YPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NA
ME_HOUSING_TYPE', 'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION
_TYPE', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCY
STATE_MODE']

['SK_ID_CURR', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_ID_PUBLISH', 'FLAG_EMP_PHON
E', 'FLAG_WORK_PHONE', 'FLAG_PHONE', 'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_P
ROCESS_START', 'REG_CITY_NOT_WORK_CITY', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_6',
'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_16', 'AMT_INCOME_TOTAL',
'AMT_CREDIT', 'AMT_ANNUITY', 'REGION_POPULATION_RELATIVE', 'DAYS_REGISTRATION',
'OWN_CAR_AGE', 'CNT_FAM_MEMBERS', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_
3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'ENTRAN
CES_AVG', 'FLOORSMAX_AVG', 'LANDAREA_AVG', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_C
NT_SOCIAL_CIRCLE', 'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_YEAR']
```

Build processing pipelines

In this part of the project the focus is on constructing the pipeline. Since the data has both numerical and categorical features, it is required to create two pipelines (one for each category of data) because they require different transformations. After finishing that, the two pipelines should be unified to produce one full pipeline that performs transformation on all the dataset.

In [86]:
```
num_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer(strategy = 'median'))
])
cat_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

In [87]:
```
data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, numerical_df),
    ("cat_pipeline", cat_pipeline, categorical_df)], remainder = 'drop')

X_train_transformed = data_pipeline.fit_transform(X_train)
```

In [88]:
```
column_names = numerical_df  + \
               list(data_pipeline.transformers_[1][1].named_steps["ohe"].get_fea

pd.DataFrame(X_train_transformed, columns=column_names).head()
```

Out[88]:

	SK_ID_CURR	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_ID_PUBLISH	FLAG_EMP_PHONE	FLAG_W(
0	-0.341075	-1.676136	2.132885	-0.937673	-2.132831	
1	-1.619000	1.173499	-0.452704	-0.269054	0.468860	
2	-1.003644	-1.644952	-0.467140	-1.335928	0.468860	
3	0.912957	1.223484	-0.469780	0.160347	0.468860	
4	0.925258	0.055244	-0.476786	-1.330627	0.468860	

5 rows × 175 columns

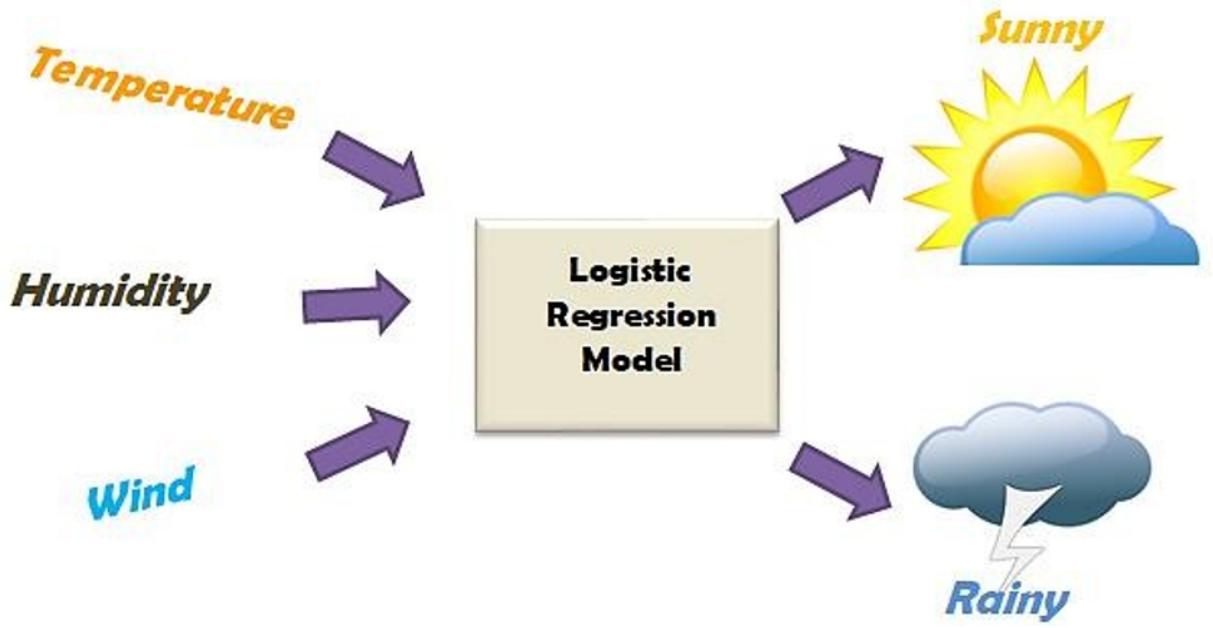
Modeling and Feature Engineering

Now that we have explored the data, cleaned it, preprocessed it and added a new feature to it, we can start the modeling part of the project by applying Machine Learning algorithms. In this section, you will have a baseline logistic regression model and grid searches on different models. In the end, you will find out which parameters are the best for each algorithm and you will be able to compare the performance of the models with the baseline model.

Baseline Logistic Regression

In [89]:
```
from IPython.display import Image
Image(url= "https://prwatech.in/blog/wp-content/uploads/2020/02/logi1.png", widt
```

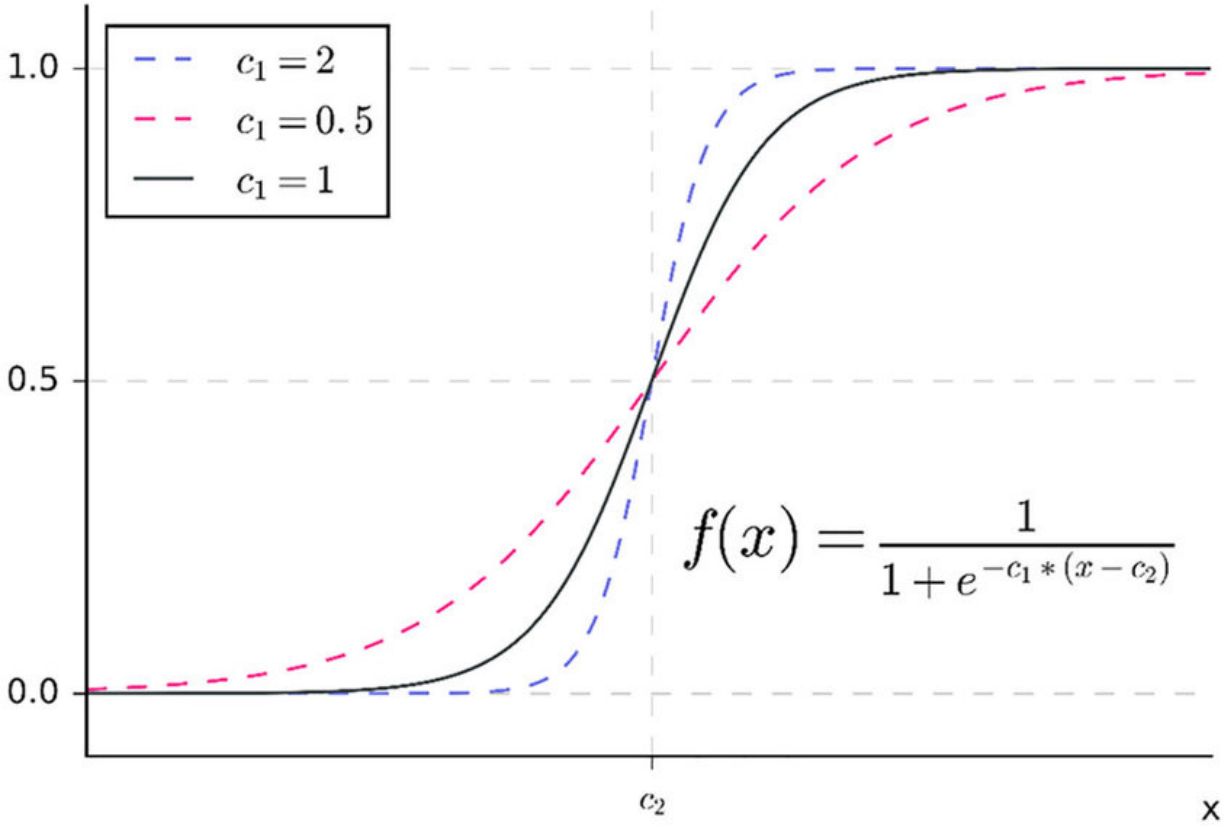
Out[89]:



Logistic regression (LR) is a statistical method similar to linear regression since LR finds an equation that predicts an outcome for a binary variable, Y, from one or more response variables, X. However, unlike linear regression the response variables can be categorical or continuous, as the model does not strictly require continuous data. To predict group membership, LR uses the log odds ratio rather than probabilities and an iterative maximum likelihood method rather than a least squares to fit the final model. This means the researcher has more freedom when using LR and the method may be more appropriate for nonnormally distributed data or when the samples have unequal covariance matrices. Logistic regression assumes independence among variables, which is not always met in morphoscopic datasets.

In [90]:
```
from IPython.display import Image

Image(url= "https://www.researchgate.net/publication/327512672/figure/fig2/AS:66
```

Out[90]:



Fitting and Storing Results

In [91]:

```
clf_pipe = make_pipeline(data_pipeline, LogisticRegression())
# clf_pipe.fit(X_train, y_train)

# train_acc = clf_pipe.score(X_train, y_train)
# # validAcc  = clf_pipe.score(X_valid, y_valid)
# testAcc   = clf_pipe.score(X_test, y_test)

# print(train_acc,testAcc)

start = time()
clf_pipe.fit(X_train, y_train)
train_time = np.round(time() - start, 4)

trainAcc  = clf_pipe.score(X_train, y_train)
validAcc  = clf_pipe.score(X_test, y_test)
start = time()
testAcc   = clf_pipe.score(X_test, y_test)
test_time = np.round(time() - start, 4)

#del experimentLog
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc"
                                               "Train Time(s)",  "Test Time(s)",
experimentLog.loc[len(experimentLog)] =[f"Baseline 1 LogReg", "HCDR",
                                        f"{trainAcc*100:8.2f}%", f"{validAcc*100
                                        train_time, test_time,
                                        "Baseline 1 LogReg pipeline with Cat+Num


experimentLog
```

Out[91]:

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Description
0	Baseline 1 LogReg	HCDR	91.91%	91.94%	91.94%	17.1005	0.4038	Baseline 1 LogReg pipeline with Cat+Num features

In [92]:
```
y_pred = clf_pipe.predict(X_test)
```

In [93]:
```
res = clf_pipe.predict_proba(X_train)
res
```

Out[93]:
```
array([[0.95176856, 0.04823144],
       [0.8854846 , 0.1145154 ],
       [0.95997397, 0.04002603],
       ...,
       [0.96730581, 0.03269419],
       [0.94754299, 0.05245701],
       [0.87877635, 0.12122365]])
```

> Prediction probability of Class 0 and Class 1

Validation Accuracy

In [94]:
```
print('Validation set accuracy score: ' + str(accuracy_score(y_test,y_pred)))
```

```
Validation set accuracy score: 0.919434824317513
```

Log Loss

In [95]:
```
Image(url= "https://miro.medium.com/max/1192/1*wilGXrItaMAJmZNl6RJq9Q.png", widt
```

Out[95]:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

ROC-AUC Score

In [96]:
```
roc_auc_score(y_test, clf_pipe.predict_proba(X_test)[:, 1])
```

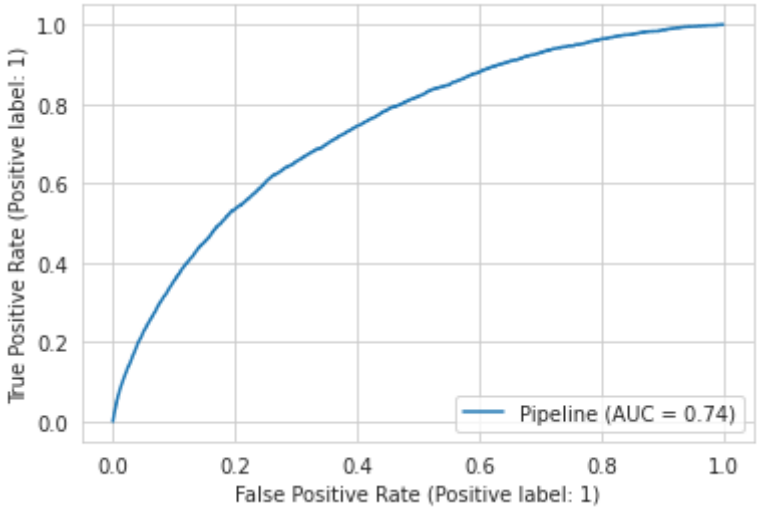
Out[96]:
```
0.7417386986188286
```

ROC Curve

In [97]:
```
import matplotlib.pyplot as plt
from sklearn import metrics
```

In [98]:
```
metrics.plot_roc_curve(clf_pipe, X_test, y_test)
```

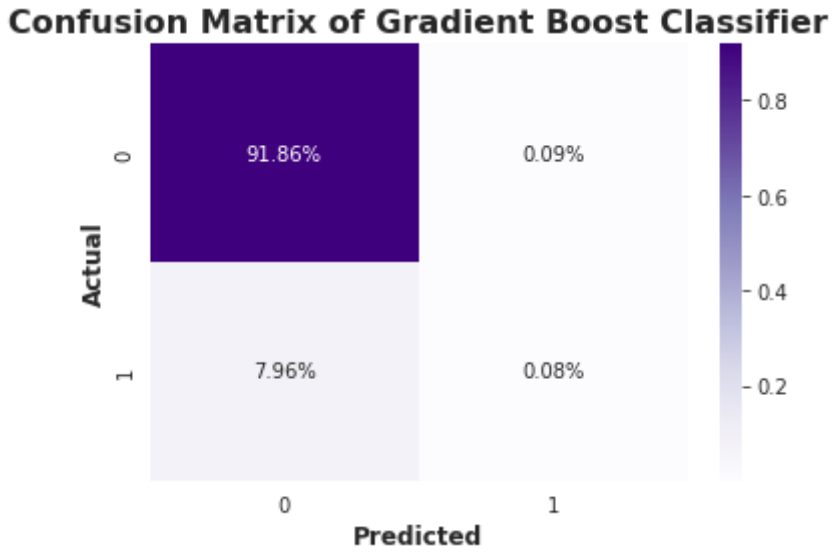
Out[98]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f42db3f0af0>
```



Confusion Matrix

In [99]:
```
from sklearn.metrics import accuracy_score, recall_score, plot_roc_curve, confus

cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Purples')
plt.title('Confusion Matrix of Gradient Boost Classifier', fontweight='bold', fo
plt.xlabel('Predicted', fontweight='bold', fontsize=12)
plt.ylabel('Actual', fontweight='bold', fontsize=12)
```

Out[99]:
```
Text(33.0, 0.5, 'Actual')
```



Baseline Random Forest

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

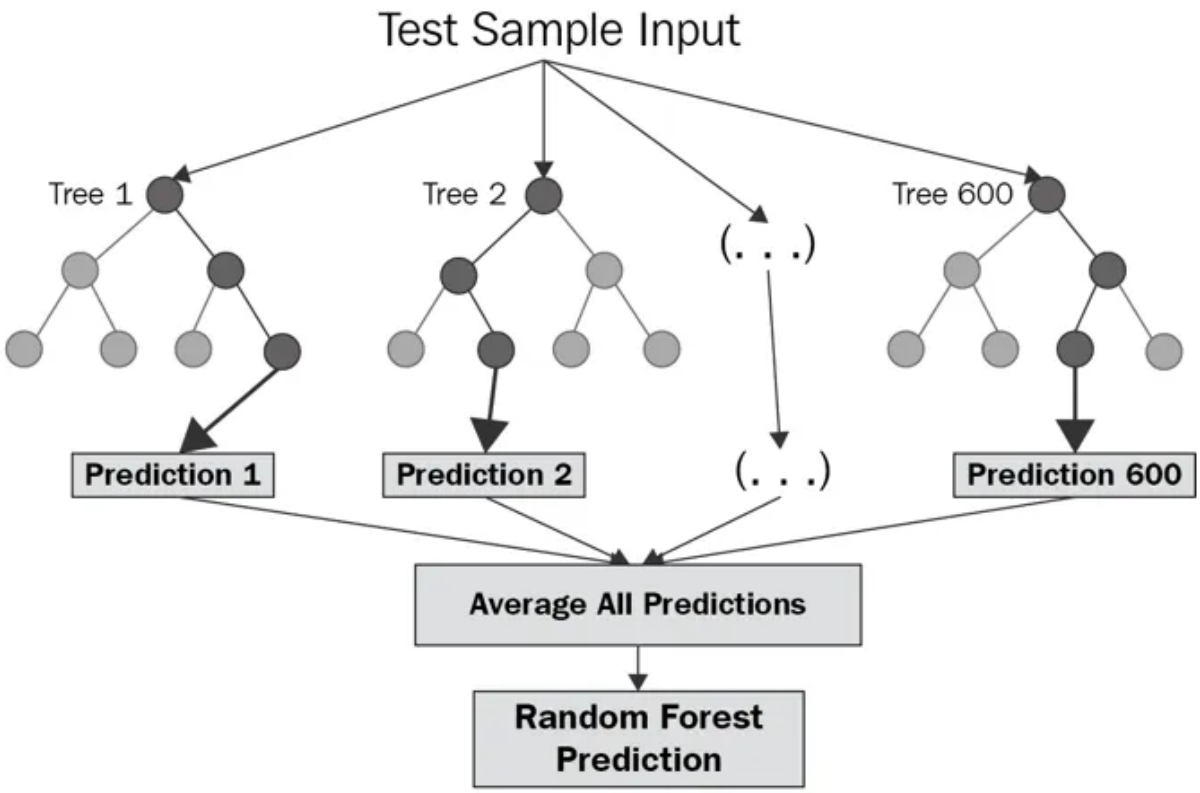
Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

In [100…
```
Image(url= "https://cdn.corporatefinanceinstitute.com/assets/random-forest.png",
```

Out[100…



In [101…
```
Image(url= "https://miro.medium.com/max/1400/1*cBaV_bLVW-gT6PkG5ve26A.png", widt
```

Out[101…

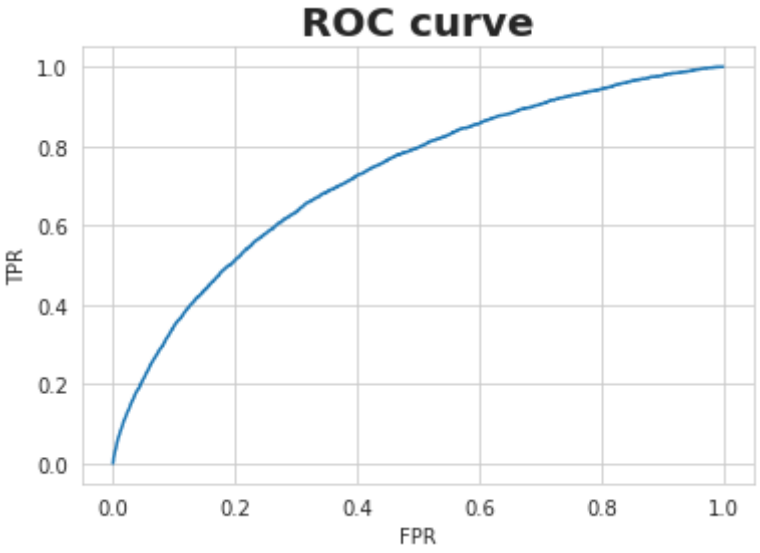
$$F(x) = \frac{1}{J}\sum_{j=1}^{J} c_{j_{full}} + \sum_{k=1}^{K}(\frac{1}{J}\sum_{j=1}^{J} contribution_j(x,k))$$

Fitting and Storing Results

In [102…
```
# def model_rf(X_train):
```

```
RF = RandomForestClassifier(random_state = 42,n_estimators=20, criterion='gini',

data_pipeline_rf = make_pipeline(data_pipeline, RF)
start = time()
data_pipeline_rf.fit(X_train, y_train)
train_time = np.round(time() - start, 4)
train_acc = data_pipeline_rf.score(X_train, y_train)
validAcc  = data_pipeline_rf.score(X_test, y_test)
start = time()
testAcc  = data_pipeline_rf.score(X_test, y_test)
test_time = np.round(time() - start, 4)

predictions = data_pipeline_rf.predict_proba(X_test)
print ("ROC_AUC_Score",roc_auc_score(y_test, predictions[:,1]))

fpr, tpr, _ = roc_curve(y_test, predictions[:,1])

plt.clf()
plt.plot(fpr, tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve',fontweight='bold',fontsize=20)
plt.show()
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc"
                                               "Train Time(s)",  "Test Time(s)",
experimentLog.loc[len(experimentLog)] =[f"Baseline 1 RandomForest", "HCDR",
                                        f"{trainAcc*100:8.2f}%", f"{validAcc*100
                                        train_time, test_time,
                                        "Baseline 1 RandomForest pipeline with C

experimentLog
```

```
ROC_AUC_Score 0.7253912369656725
```



Out[102…

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Description
0	Baseline 1 LogReg	HCDR	91.91%	91.94%	91.94%	17.1005	0.4038	Baseline 1 LogReg pipeline with Cat+Num features

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Description
1	Baseline 1 RandomForest	HCDR	91.91%	91.95%	91.95%	10.7311	0.4899	Baseline 1 RandomForest pipeline with Cat+Num ...

In [103…
```
y_pred_rf = data_pipeline_rf.predict(X_test)
```

In [104…
```
res = data_pipeline_rf.predict_proba(X_train)
res
```

Out[104…
```
array([[0.94402268, 0.05597732],
       [0.88961809, 0.11038191],
       [0.93682042, 0.06317958],
       ...,
       [0.95657847, 0.04342153],
       [0.91790688, 0.08209312],
       [0.89067702, 0.10932298]])
```

Validation Accuracy

In [105…
```
print('Validation set accuracy score: ' + str(accuracy_score(y_test,y_pred_rf
                                                             )))
```

```
Validation set accuracy score: 0.9195323805342829
```

Log Loss

In [106…
```
log_loss(y_test,y_pred_rf)
```

Out[106…
```
2.7792531157571663
```

ROC-AUC Score

In [107…
```
roc_auc_score(y_test, clf_pipe.predict_proba(X_test)[:, 1])
```

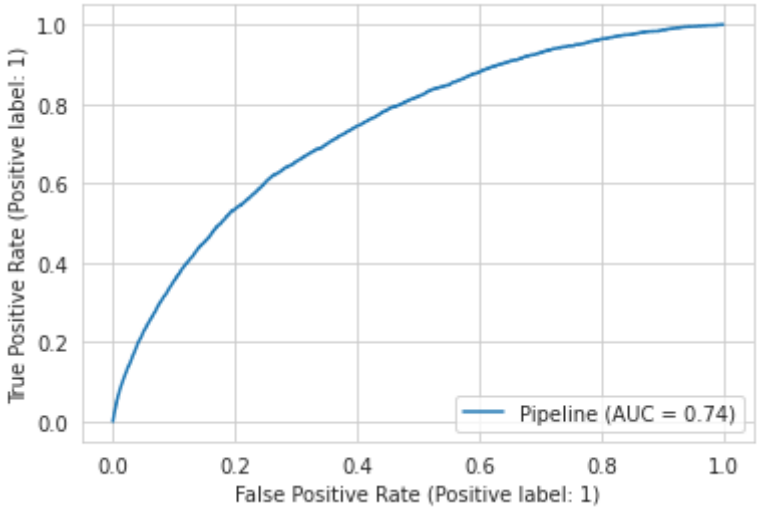
Out[107…
```
0.7417386986188286
```

ROC Curve

In [108…
```
import matplotlib.pyplot as plt
from sklearn import metrics
```

In [109…
```
metrics.plot_roc_curve(clf_pipe, X_test, y_test)
```

Out[109…
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f42dd21b1f0>
```



In [110…
```
X_train
```

Out[110…

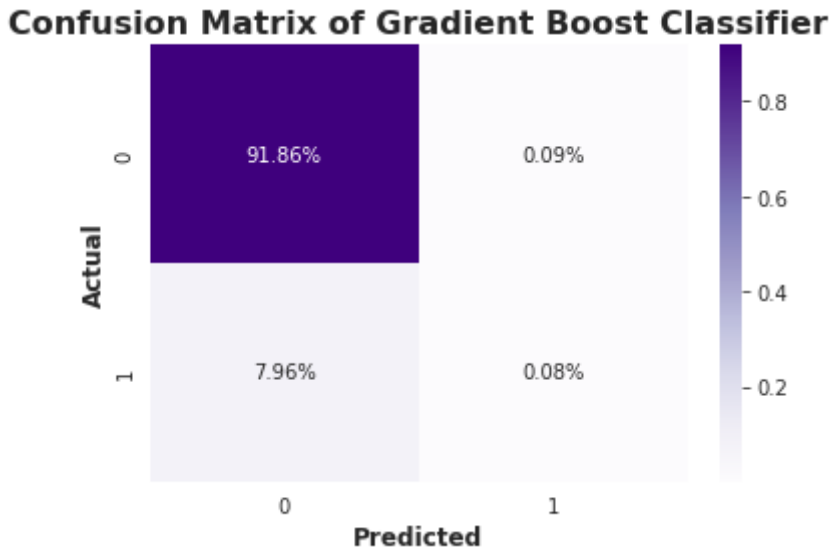
	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REA
123473	243191	Cash loans	F	Y	
10118	111778	Cash loans	M	N	
64716	175057	Cash loans	M	Y	
234940	372147	Cash loans	M	Y	
236051	373412	Cash loans	M	N	
...	...	...	...	...	
119879	239009	Cash loans	F	N	
259178	399937	Cash loans	F	N	
131932	253016	Cash loans	M	Y	
146867	270275	Revolving loans	M	Y	
121958	241394	Cash loans	M	Y	

246008 rows × 51 columns

Confusion Matrix

In [111…
```
from sklearn.metrics import accuracy_score, recall_score, plot_roc_curve, confus

cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Purples')
plt.title('Confusion Matrix of Gradient Boost Classifier', fontweight='bold', fo
plt.xlabel('Predicted', fontweight='bold', fontsize=12)
plt.ylabel('Actual', fontweight='bold', fontsize=12)
```

Out[111…
```
Text(33.0, 0.5, 'Actual')
```



Phase 2

Working on Bureau data

In [112…
```
datasets['bureau'].head()
```

Out[112…

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DA
0	215354	5714462	Closed	currency 1	-497	
1	215354	5714463	Active	currency 1	-208	
2	215354	5714464	Active	currency 1	-203	
3	215354	5714465	Active	currency 1	-203	
4	215354	5714466	Active	currency 1	-629	

In [113…
```
datasets['bureau']['CREDIT_ACTIVE'].value_counts()
```

Out[113…
```
Closed      1079273
Active       630607
Sold           6527
Bad debt         21
Name: CREDIT_ACTIVE, dtype: int64
```

Merging of Bureau with Bureau Balance

> **on SK_ID_BUREAU**

In [114…
```
df_bureau1 = pd.merge(left = datasets['bureau'], right = datasets['bureau_balanc
df_bureau1.shape
```

Out[114…
```
(24179741, 19)
```

In [115…
```
datasets['application_train'].head()
```

Out[115…

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
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	

5 rows × 52 columns

> **Making a test dataframe which contains only target with SK_ID_CURR so that we can check the correlation with the merged bureau column so that we can drop irrelevant columns.**

In [116…
```
df_test = datasets['application_train'][['SK_ID_CURR','TARGET']]
df_test
```

Out[116…

	SK_ID_CURR	TARGET
0	100002	1
1	100003	0
2	100004	0
3	100006	0
4	100007	0
…	…	…
307506	456251	0
307507	456252	0
307508	456253	0
307509	456254	1
307510	456255	0

307511 rows × 2 columns

In [117…
```
df_mergewithtarget  = pd.merge(left = df_bureau1, right = df_test,
                               how='left', left_on='SK_ID_CURR', right_on='SK_
```

In [118…
```
df_mergewithtarget.shape
```

Out[118… (24179741, 20)

In [119…
```
!pip install missingno
```

```
Requirement already satisfied: missingno in /geode2/home/u060/nmalpani/Carbonat
e/miniconda3/lib/python3.9/site-packages (0.5.0)
Requirement already satisfied: numpy in /geode2/home/u060/nmalpani/Carbonate/min
iconda3/lib/python3.9/site-packages (from missingno) (1.21.4)
Requirement already satisfied: matplotlib in /geode2/home/u060/nmalpani/Carbonat
e/miniconda3/lib/python3.9/site-packages (from missingno) (3.5.0)
Requirement already satisfied: seaborn in /geode2/home/u060/nmalpani/Carbonate/m
iniconda3/lib/python3.9/site-packages (from missingno) (0.11.2)
Requirement already satisfied: scipy in /geode2/home/u060/nmalpani/Carbonate/min
iconda3/lib/python3.9/site-packages (from missingno) (1.7.3)
Requirement already satisfied: pillow>=6.2.0 in /geode2/home/u060/nmalpani/Carbo
nate/miniconda3/lib/python3.9/site-packages (from matplotlib->missingno) (8.4.0)
Requirement already satisfied: python-dateutil>=2.7 in /geode2/home/u060/nmalpan
i/Carbonate/miniconda3/lib/python3.9/site-packages (from matplotlib->missingno)
(2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in /geode2/home/u060/nmalpani/C
arbonate/miniconda3/lib/python3.9/site-packages (from matplotlib->missingno) (4.
28.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /geode2/home/u060/nmalpani/C
arbonate/miniconda3/lib/python3.9/site-packages (from matplotlib->missingno) (1.
3.2)
Requirement already satisfied: cycler>=0.10 in /geode2/home/u060/nmalpani/Carbon
ate/miniconda3/lib/python3.9/site-packages (from matplotlib->missingno) (0.11.0)
Requirement already satisfied: packaging>=20.0 in /geode2/home/u060/nmalpani/Car
bonate/miniconda3/lib/python3.9/site-packages (from matplotlib->missingno) (21.
3)
Requirement already satisfied: setuptools-scm>=4 in /geode2/home/u060/nmalpani/C
arbonate/miniconda3/lib/python3.9/site-packages (from matplotlib->missingno) (6.
3.2)
Requirement already satisfied: pyparsing>=2.2.1 in /geode2/home/u060/nmalpani/Ca
rbonate/miniconda3/lib/python3.9/site-packages (from matplotlib->missingno) (3.
0.6)
Requirement already satisfied: pandas>=0.23 in /geode2/home/u060/nmalpani/Carbon
ate/miniconda3/lib/python3.9/site-packages (from seaborn->missingno) (1.3.4)
Requirement already satisfied: pytz>=2017.3 in /geode2/home/u060/nmalpani/Carbon
ate/miniconda3/lib/python3.9/site-packages (from pandas>=0.23->seaborn->missingn
o) (2021.3)
Requirement already satisfied: six>=1.5 in /geode2/home/u060/nmalpani/Carbonate/
miniconda3/lib/python3.9/site-packages (from python-dateutil>=2.7->matplotlib->m
issingno) (1.16.0)
Requirement already satisfied: tomli>=1.0.0 in /geode2/home/u060/nmalpani/Carbon
ate/miniconda3/lib/python3.9/site-packages (from setuptools-scm>=4->matplotlib->
missingno) (1.2.2)
Requirement already satisfied: setuptools in /geode2/home/u060/nmalpani/Carbonat
e/miniconda3/lib/python3.9/site-packages (from setuptools-scm>=4->matplotlib->mi
ssingno) (52.0.0.post20210125)
```

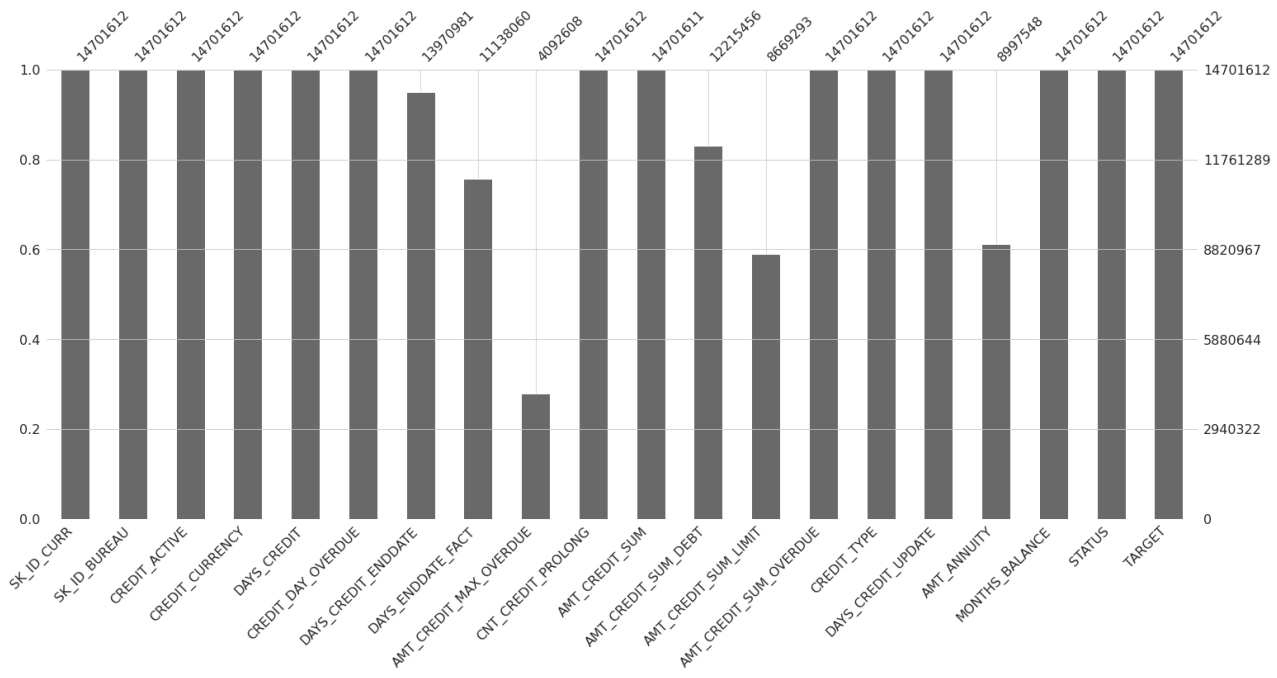
In [120…
```
import missingno as msno
```

In [121…
```
df_mergewithtarget1 = df_mergewithtarget.dropna(subset=['TARGET'])
```

EDA and preprocessing on Merged Bureau Data

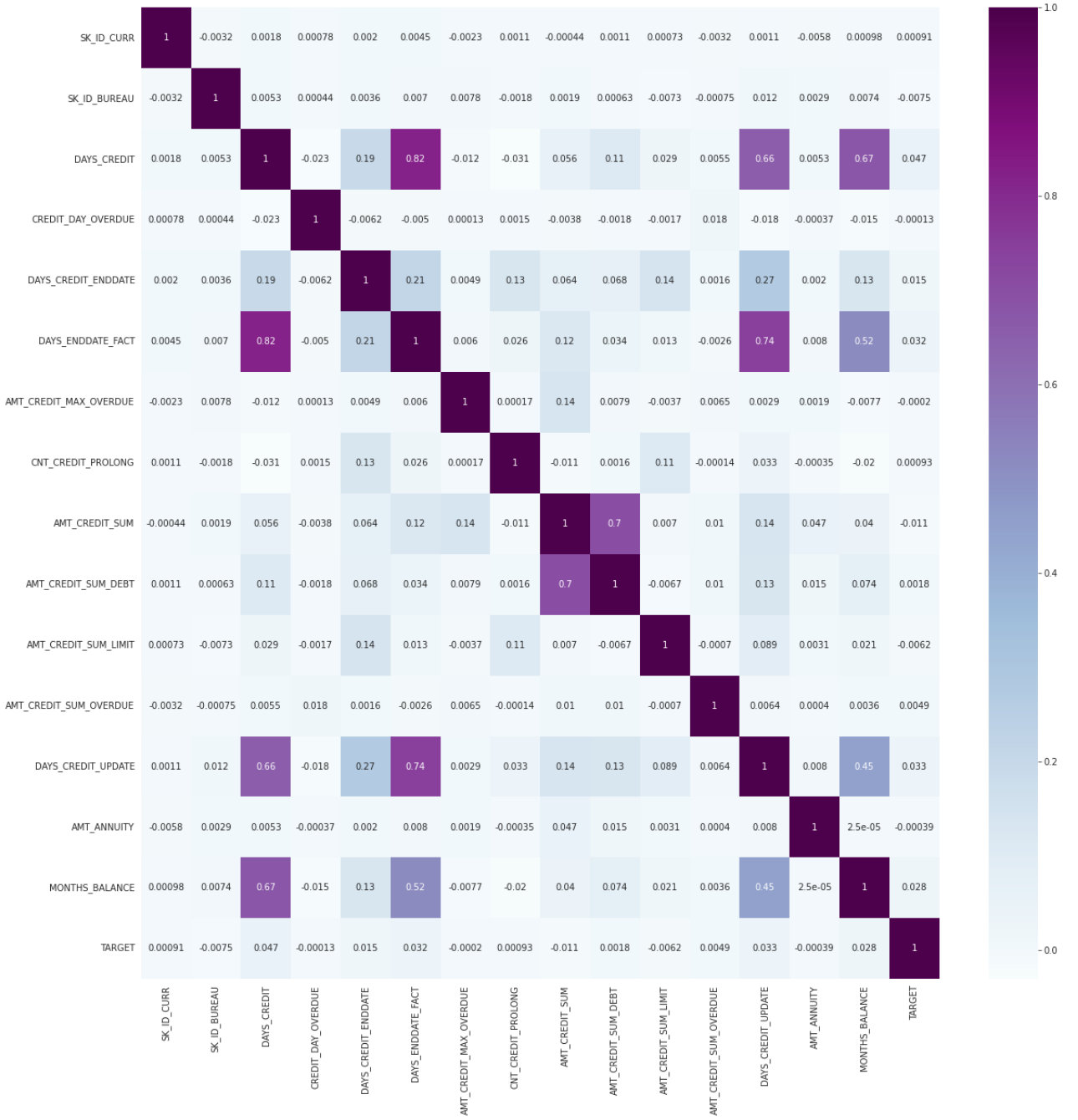
In [122…
```
msno.bar(df_mergewithtarget1)
```

Out[122…
```
<AxesSubplot:>
```



In [123…
```
plt.figure(figsize = (20,20))
sns.heatmap(df_mergewithtarget1.corr(),annot=True, cmap='BuPu')
```

Out[123…
```
<AxesSubplot:>
```



In [124…
```
df_mergewithtarget1.corrwith(df_mergewithtarget1["TARGET"]).sort_values(ascendin
```

Out[124…
```
TARGET                       1.000000
DAYS_CREDIT                  0.047350
DAYS_CREDIT_UPDATE           0.033169
DAYS_ENDDATE_FACT            0.031562
MONTHS_BALANCE               0.027773
DAYS_CREDIT_ENDDATE          0.014881
AMT_CREDIT_SUM_OVERDUE       0.004875
AMT_CREDIT_SUM_DEBT          0.001840
CNT_CREDIT_PROLONG           0.000929
SK_ID_CURR                   0.000909
CREDIT_DAY_OVERDUE          -0.000128
AMT_CREDIT_MAX_OVERDUE      -0.000204
AMT_ANNUITY                 -0.000389
AMT_CREDIT_SUM_LIMIT        -0.006195
SK_ID_BUREAU                -0.007517
```

```
AMT_CREDIT_SUM           -0.010691
dtype: float64
```

In [125…
```
df_bureau1 = df_bureau1.drop_duplicates()
```

In [126…
```
df_bureau1['CREDIT_ACTIVE'].value_counts()
```

Out[126…
```
Closed      18511853
Active       5545692
Sold          122096
Bad debt         100
Name: CREDIT_ACTIVE, dtype: int64
```

In [127…
```
df_bureau1['AMT_CREDIT_SUM_LIMIT'].isnull().sum()
```

Out[127…
```
10376144
```

In [128…
```
df_bureau1['AMT_CREDIT_SUM_LIMIT'] = df_bureau1['AMT_CREDIT_SUM_LIMIT'].replace(
```

In [129…
```
df_bureau1
```

Out[129…

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CR
0	380361	5715448	Active	currency 1	-820	
1	380361	5715448	Active	currency 1	-820	
2	380361	5715448	Active	currency 1	-820	
3	380361	5715448	Active	currency 1	-820	
4	380361	5715448	Active	currency 1	-820	
...	...	...	...	...	...	
24179736	407724	5053758	Closed	currency 1	-2423	
24179737	407724	5053758	Closed	currency 1	-2423	
24179738	407724	5053758	Closed	currency 1	-2423	
24179739	407724	5053758	Closed	currency 1	-2423	
24179740	407724	5053758	Closed	currency 1	-2423	

24179741 rows × 19 columns

In [130…
```
df_bureau1['AMT_CREDIT_SUM_LIMIT'].value_counts()
```

Out[130…
```
0.000         23537973
135000.000       14315
```

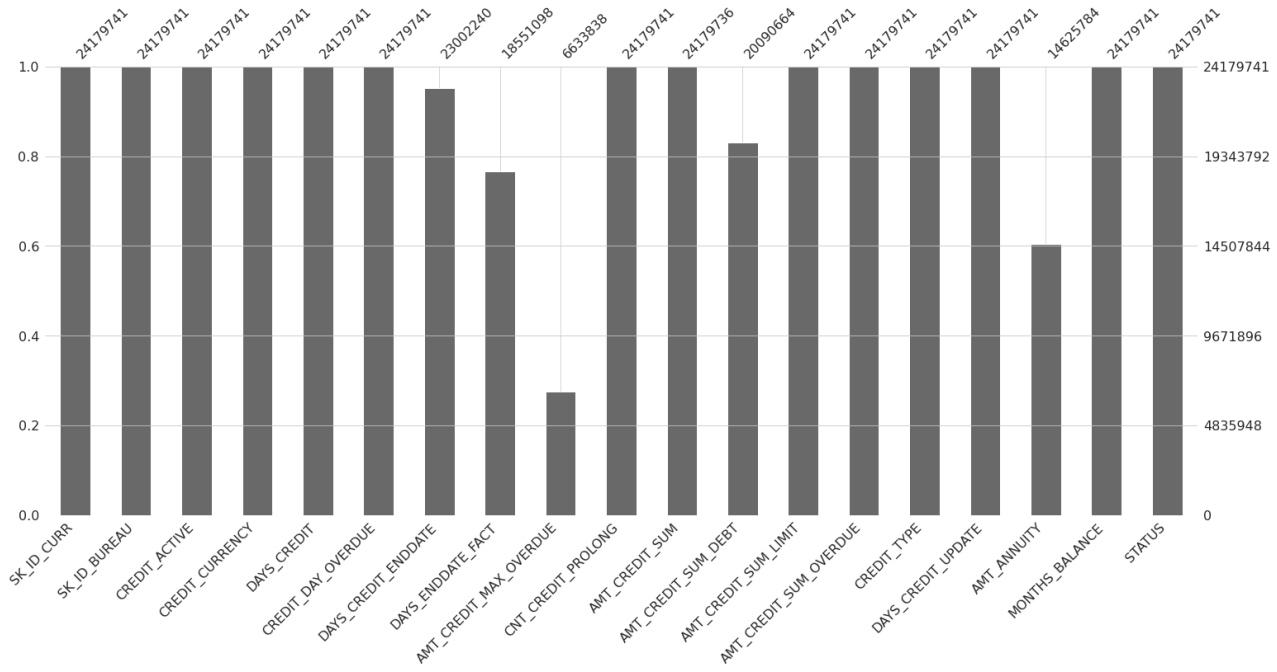
```
13500.000        12765
45000.000        11447
90000.000         8734
                 ...
62694.360            1
8651.295             1
21375.000            1
14805.000            1
2220.030             1
Name: AMT_CREDIT_SUM_LIMIT, Length: 22040, dtype: int64
```

In [131…

```
msno.bar(df_bureau1)
```

Out[131…

```
<AxesSubplot:>
```



In [132…

```
df_bureau1 = df_bureau1.groupby('SK_ID_CURR').median()
df_bureau1
```

Out[132…

SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DA
100001	5896632.0	-879.0	0.0	-492.0	
100002	6158905.0	-1043.0	0.0	-911.0	
100005	6735200.0	-373.0	0.0	-128.0	
100010	5576631.0	-1939.5	0.0	-119.5	
100013	5922082.0	-2005.0	0.0	-909.0	
...	...	...	...	...	
456247	6618241.0	-1757.0	0.0	-1402.0	
456250	6817237.0	-824.0	0.0	1797.0	
456253	6098498.0	-919.0	0.0	-189.0	

SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DA
456254	6669849.0	-1104.0	0.0	-859.0	
456255	5126331.0	-1021.0	0.0	279.0	

134542 rows × 14 columns

In [133…
```
df_bureau1.reset_index(inplace=True)
```

In [134…
```
df_bureau_final = df_bureau1[['SK_ID_CURR','DAYS_CREDIT','DAYS_ENDDATE_FACT','AM
                              'MONTHS_BALANCE',
                                'AMT_CREDIT_SUM_LIMIT']]
```

Merging of processed Bureau Data with Application Train Data

> **Joining on SK_ID_CURR**

In [135…
```
df_finalm_bureau_train =  pd.merge(left = df_bureau_final, right = datasets['app
                                   how='inner', left_on='SK_ID_CURR', right_on='SK
```

In [136…
```
df_finalm_bureau_train
```

Out[136…

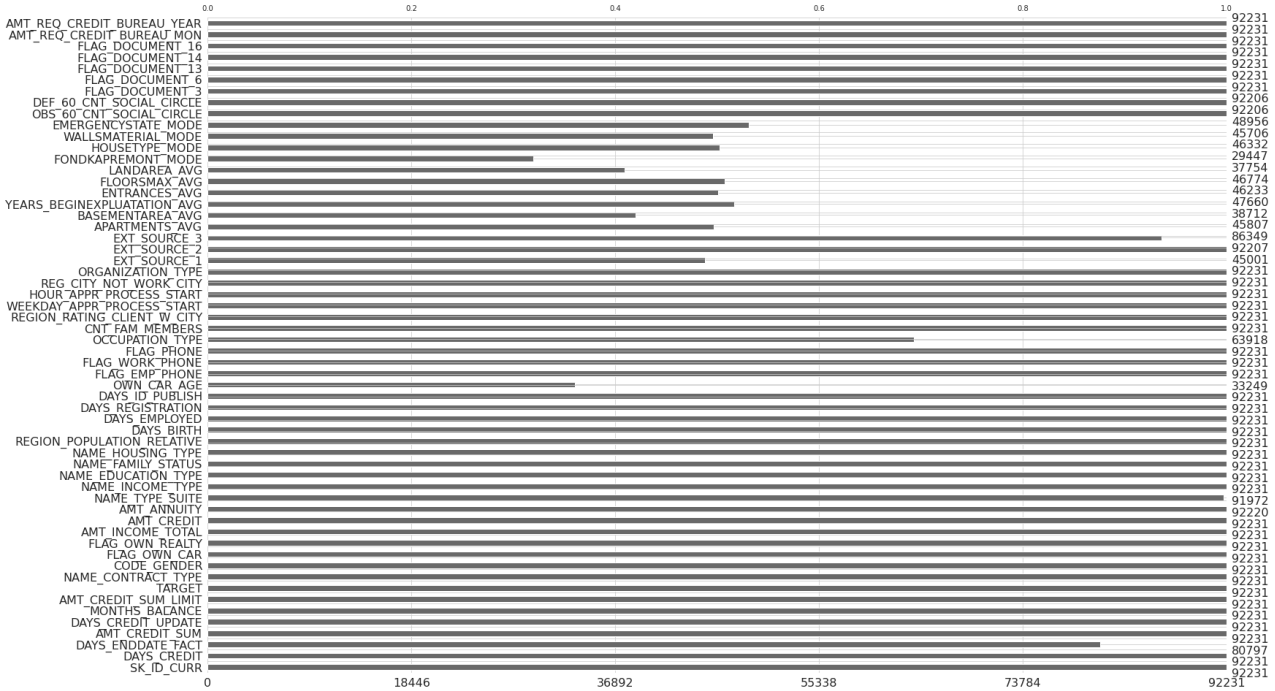
	SK_ID_CURR	DAYS_CREDIT	DAYS_ENDDATE_FACT	AMT_CREDIT_SUM	DAYS_CREDIT_UPI
0	100002	-1043.0	-967.0	40761.0	-7
1	100010	-1939.5	-1138.0	495000.0	-5
2	100019	-495.0	NaN	360000.0	
3	100032	-1169.5	-662.0	331875.0	-5
4	100033	-195.0	NaN	675000.0	-1
…	…	…	…	…	
92226	456244	-2138.0	-1438.0	315000.0	-13
92227	456247	-1757.0	-1402.0	143667.0	-14
92228	456253	-919.0	-794.0	675000.0	-1
92229	456254	-1104.0	-859.0	45000.0	-4
92230	456255	-1021.0	-958.0	436032.0	-7

92231 rows × 58 columns

In [137…
```
msno.bar(df_finalm_bureau_train)
```

```
<AxesSubplot:>
```

Out[137…



> **Storing the data to csv file so that if kernel crashes we can continue the work with this data and save time.**

In [138…

```
df_finalm_bureau_train.to_csv('bureau_files_merged.csv',index=False)
```

Working on POS Cash Balance

In [139…

```
datasets['POS_CASH_balance'].head()
```

Out[139…

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

In [140…

```
datasets['POS_CASH_balance'].shape
```

Out[140…

```
(10001358, 8)
```

EDA and Preprocessing on POS_CASH_BALANCE

> **to remove irrelavent data using the correlation criterion and domain**

knowledge

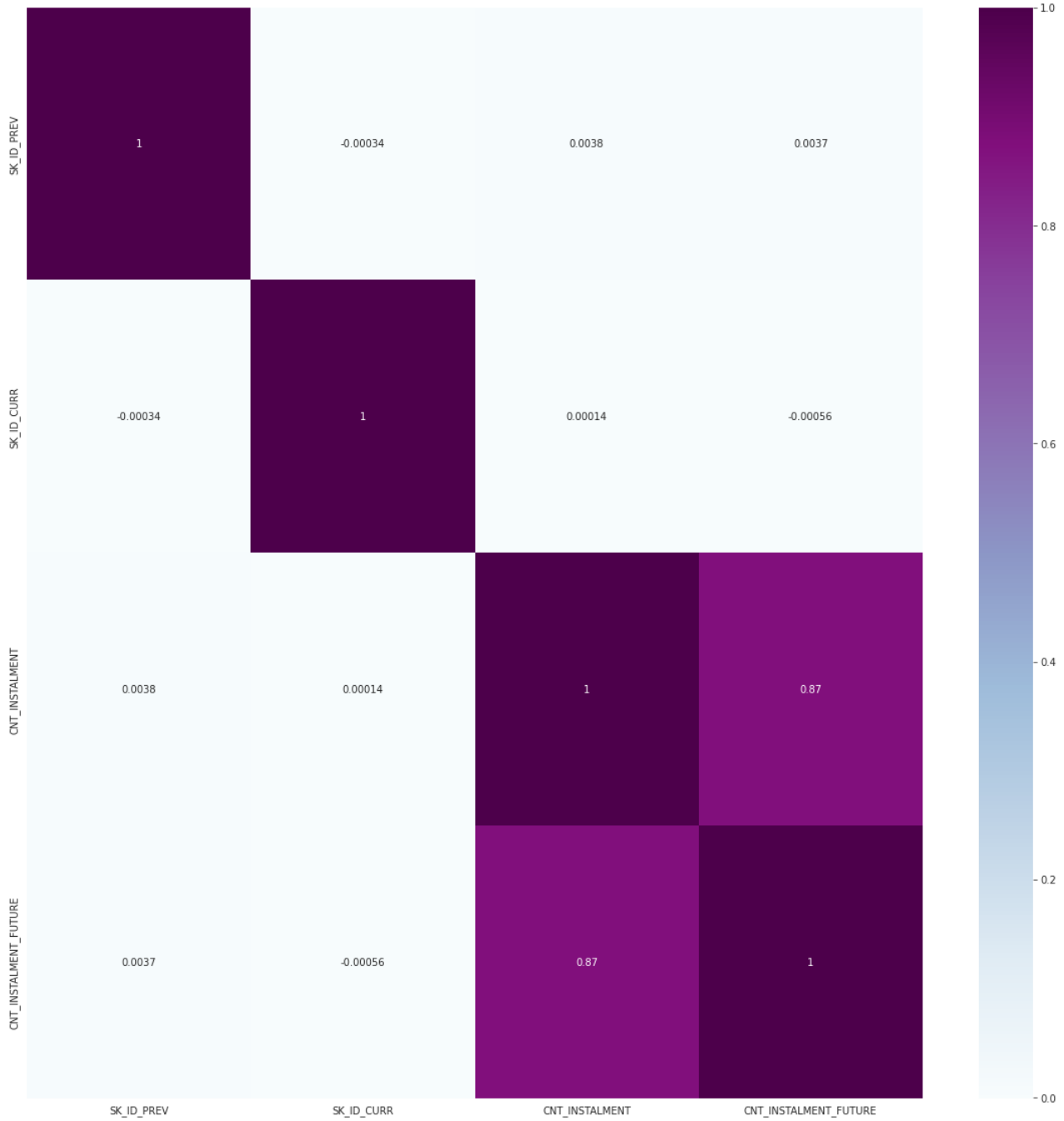
In [141…
```
datasets['POS_CASH_balance']['NAME_CONTRACT_STATUS'].value_counts()
```

Out[141…
```
Active                   9151119
Completed                 744883
Signed                     87260
Demand                      7065
Returned to the store       5461
Approved                    4917
Amortized debt               636
Canceled                      15
XNA                            2
Name: NAME_CONTRACT_STATUS, dtype: int64
```

In [142…
```
df_POS = datasets['POS_CASH_balance'][['SK_ID_PREV','SK_ID_CURR','CNT_INSTALMENT
```

In [143…
```
plt.figure(figsize = (20,20))
sns.heatmap(df_POS.corr(),annot=True, cmap='BuPu')
```

Out[143…
```
<AxesSubplot:>
```



In [144…
```
df_POS = df_POS.drop(['CNT_INSTALMENT'],axis=1)
```

Merging of POS CASH with Application Train on SK_ID_CURR

In [145…
```
df_POS_atrain = pd.merge(left=datasets['application_train'], right=df_POS, how='
                         right_on=['SK_ID_CURR'])
```

In [146…
```
df_POS_atrain.head()
```

Out[146…

SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
0	100002	1	Cash loans	M	N	
1	100002	1	Cash loans	M	N	
2	100002	1	Cash loans	M	N	
3	100002	1	Cash loans	M	N	
4	100002	1	Cash loans	M	N	

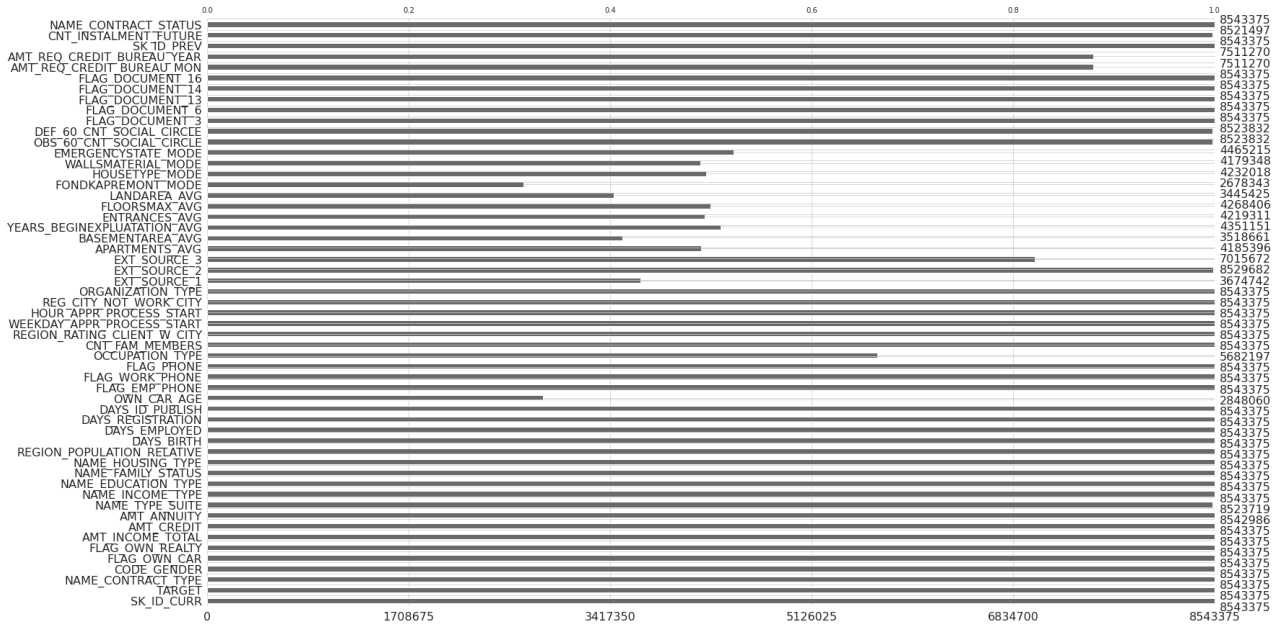
5 rows × 55 columns

In [147…
```
df_POS_atrain.shape
```

Out[147…
```
(8543375, 55)
```

In [148…
```
msno.bar(df_POS_atrain)
```

Out[148…
```
<AxesSubplot:>
```



Working on Credit Card Balance

In [149…
```
datasets['credit_card_balance'].head()
```

Out[149…

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL
0	2562384	378907	-6	56.970	135000
1	2582071	363914	-1	63975.555	45000
2	1740877	371185	-7	31815.225	450000
3	1389973	337855	-4	236572.110	225000
4	1891521	126868	-1	453919.455	450000

5 rows × 23 columns

> **Dropping of columns with no or close to none significance with Target.**

In [150…
```
df_ccbal = datasets['credit_card_balance'].drop(['SK_DPD','SK_DPD_DEF','MONTHS_B
```

In [151…
```
df_ccbal.head()
```

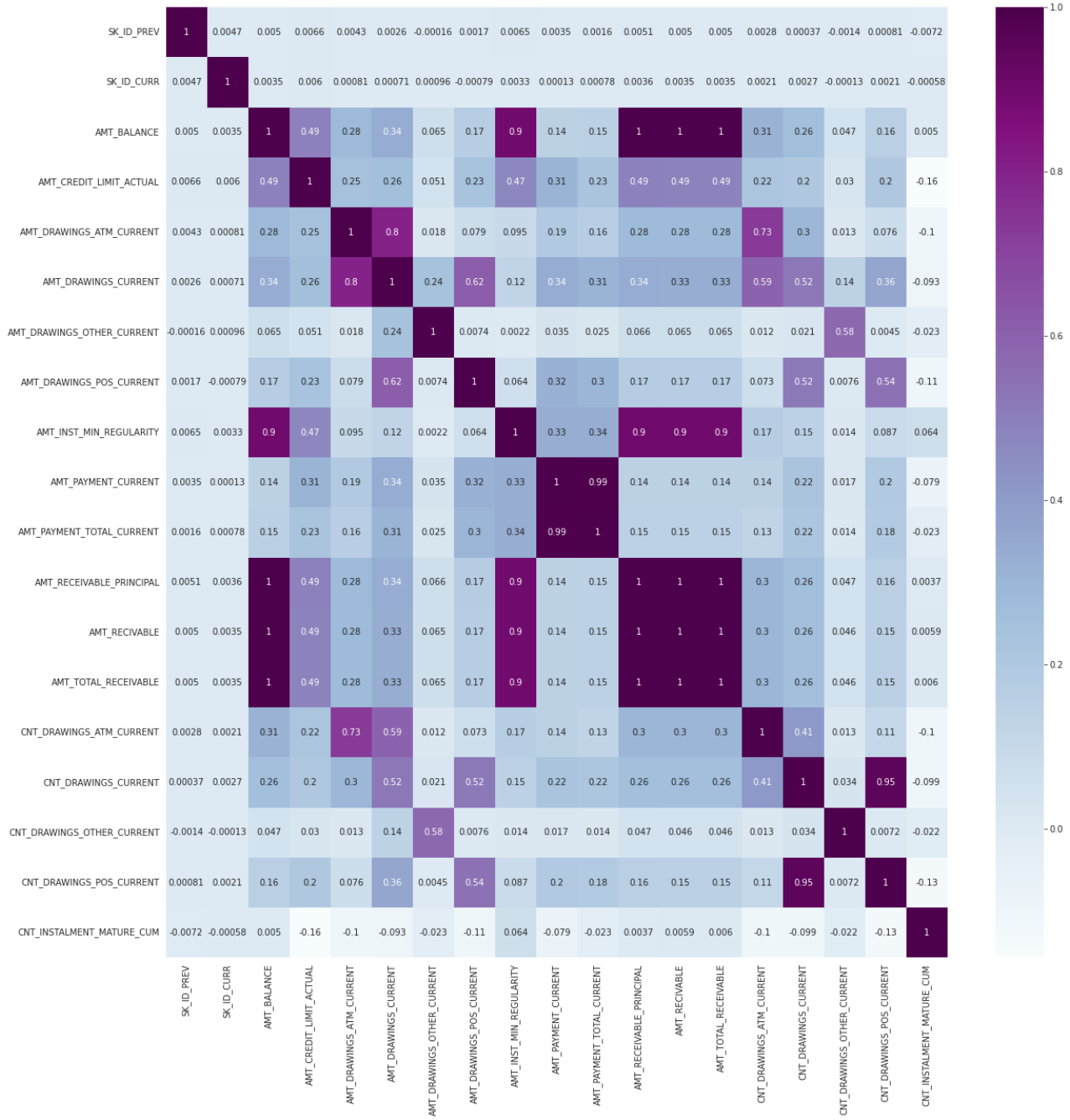
Out[151…

	SK_ID_PREV	SK_ID_CURR	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM
0	2562384	378907	56.970	135000	
1	2582071	363914	63975.555	45000	
2	1740877	371185	31815.225	450000	
3	1389973	337855	236572.110	225000	
4	1891521	126868	453919.455	450000	

EDA and Preprocessing of Credit Card Balance

In [152…
```
plt.figure(figsize = (20,20))
sns.heatmap(df_ccbal.corr(),annot=True, cmap='BuPu')
```

Out[152…
```
<AxesSubplot:>
```



In [153…
```
df_ccbal = df_ccbal.drop(['AMT_INST_MIN_REGULARITY','AMT_RECEIVABLE_PRINCIPAL','
                          'AMT_PAYMENT_TOTAL_CURRENT','AMT_DRAWINGS_ATM_CURRENT',
```

In [154…
```
df_ccbal.head()
```

Out[154…

	SK_ID_PREV	SK_ID_CURR	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_CUR
0	2562384	378907	56.970	135000	
1	2582071	363914	63975.555	45000	2
2	1740877	371185	31815.225	450000	
3	1389973	337855	236572.110	225000	2
4	1891521	126868	453919.455	450000	1

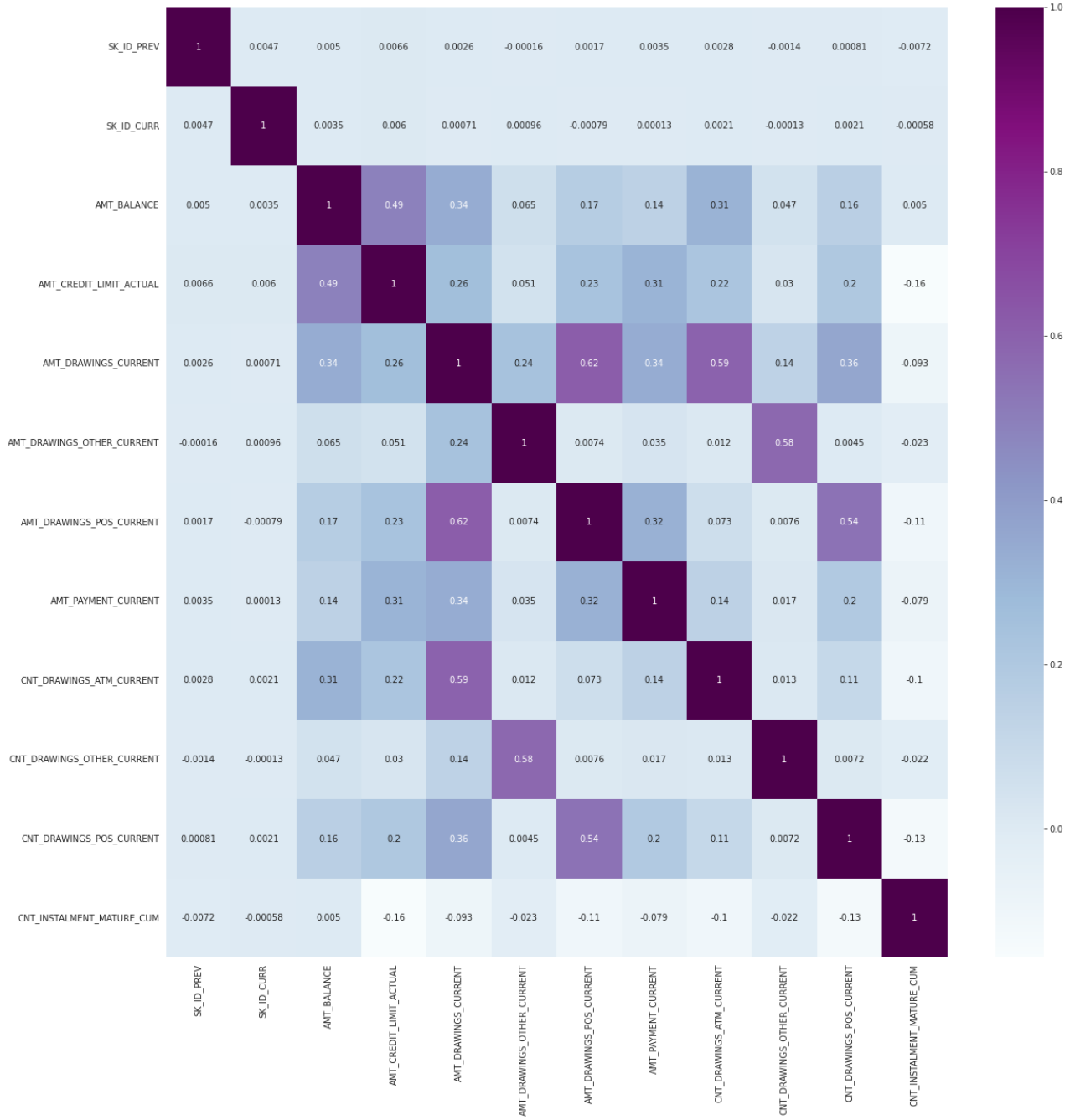
> **Grouping by SK_ID_CURR and SK_ID_PREV by mean so that we can restore as much information as we can while merging or joining**

In [155…
```
df_ccbal1=df_ccbal.groupby(['SK_ID_CURR','SK_ID_PREV']).mean()
```

In [156…
```
df_ccbal1 =  df_ccbal1.reset_index()
```

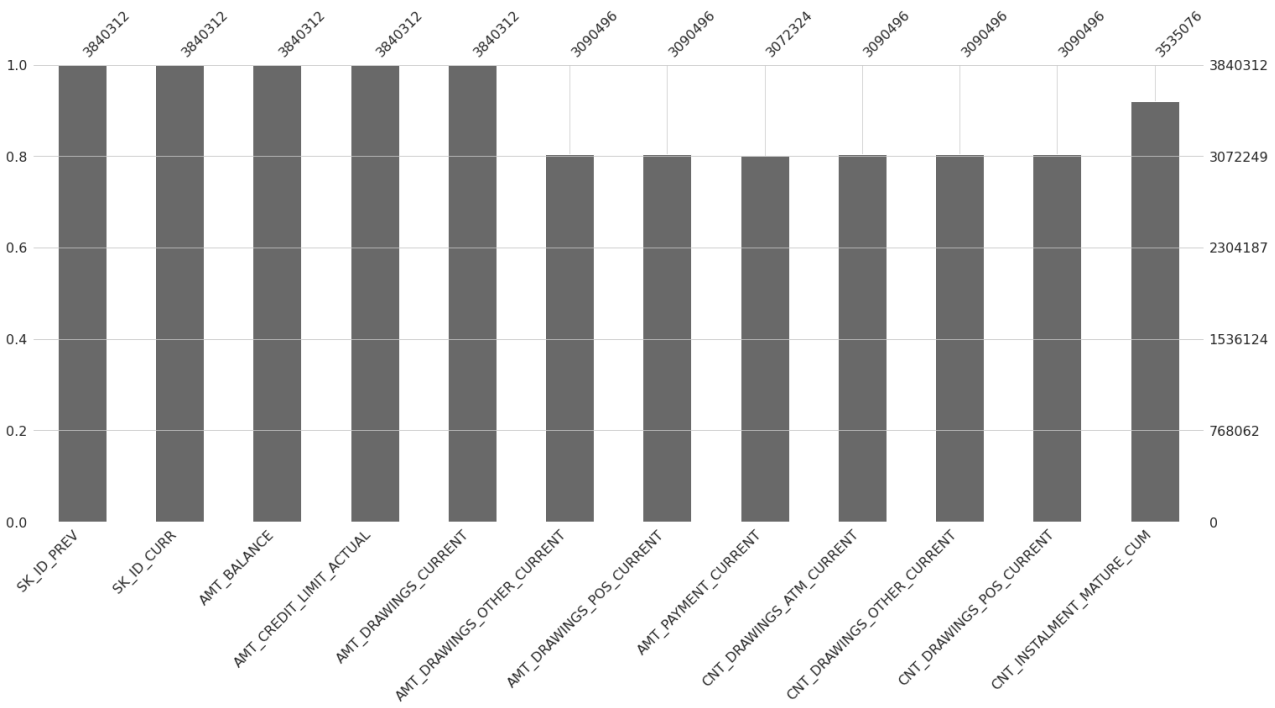
In [157…
```
plt.figure(figsize = (20,20))
sns.heatmap(df_ccbal.corr(),annot=True, cmap='BuPu')
```

Out[157…
```
<AxesSubplot:>
```



In [158…
```
msno.bar(df_ccbal)
```

Out[158…
```
<AxesSubplot:>
```



> **We will work on Installments Payments first and then join everything with Application train after joining installments payments and Credit Card Balance.**

Working on Installments Payments

In [159…
```
datasets['installments_payments'].head()
```

Out[159…

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_IN
0	1054186	161674	1.0	6	
1	1330831	151639	0.0	34	
2	2085231	193053	2.0	1	
3	2452527	199697	1.0	3	
4	2714724	167756	1.0	2	

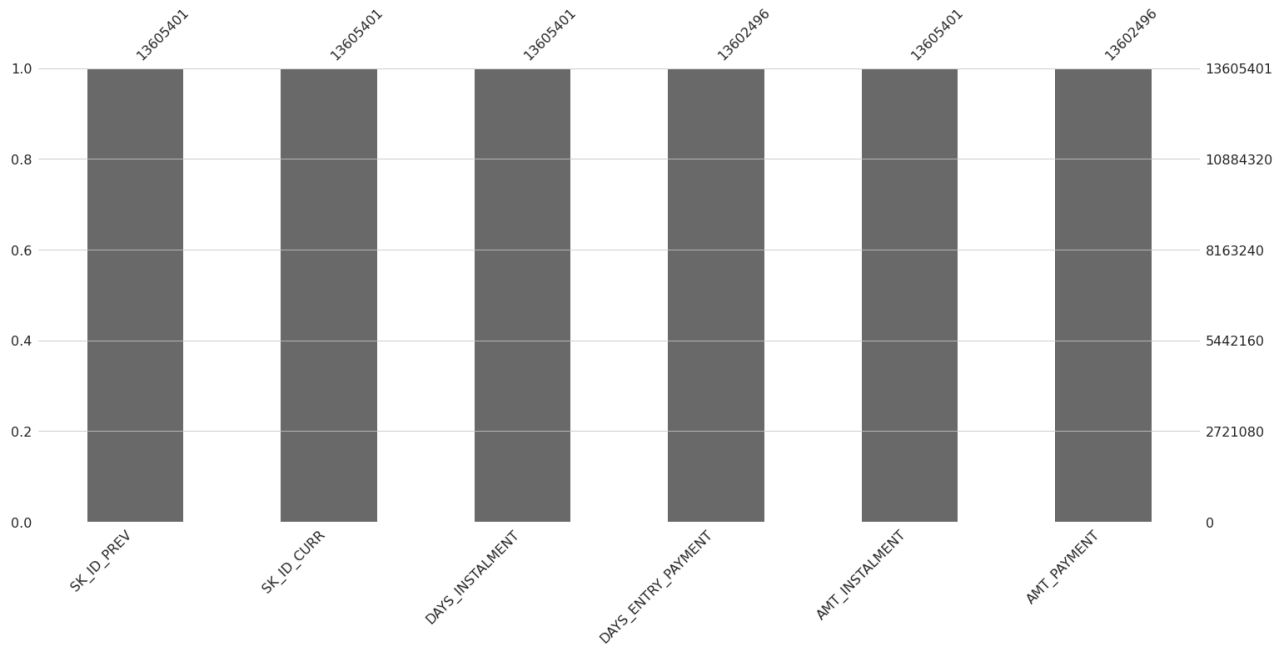
EDA and Preprocessing on Installments Payments

In [160…
```
df_installment = datasets['installments_payments'].drop(['NUM_INSTALMENT_VERSION
```

In [161…
```
msno.bar(df_installment)
```

```
<AxesSubplot:>
```

Out[161…



In [162…

```
df_installment.head()
```

Out[162…

	SK_ID_PREV	SK_ID_CURR	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT	AI
0	1054186	161674	-1180.0	-1187.0	6948.360	
1	1330831	151639	-2156.0	-2156.0	1716.525	
2	2085231	193053	-63.0	-63.0	25425.000	
3	2452527	199697	-2418.0	-2426.0	24350.130	
4	2714724	167756	-1383.0	-1366.0	2165.040	

> **Grouping by SK_ID_CURR and SK_ID_PREV by mean so that we can restore as much information as we can while merging or joining**

In [163…

```
df_installment1=df_installment.groupby(['SK_ID_CURR','SK_ID_PREV']).mean()
```

In [164…

```
df_installment1 = df_installment1.reset_index()
```

In [165…

```
df_installment1.head()
```

Out[165…

	SK_ID_CURR	SK_ID_PREV	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT	AI
0	100001	1369693	-1664.0	-1679.500000	7312.725000	
1	100001	1851984	-2886.0	-2882.333333	3981.675000	
2	100002	1038818	-295.0	-315.421053	11559.247105	
3	100003	1810518	-626.0	-630.428571	164425.332857	1(

	SK_ID_CURR	SK_ID_PREV	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT	AI
4	100003	2396755	-2145.0	-2151.750000	6731.115000	

In [166…
```
df_ccbal1['SK_ID_PREV'].value_counts()
```

Out[166…
```
1489396    1
1645533    1
2835518    1
2116495    1
2487050    1
          ..
2088402    1
2663364    1
2002588    1
1642103    1
1794451    1
Name: SK_ID_PREV, Length: 104307, dtype: int64
```

Joining Credit Card Balance and installments Payments on SK_ID_PREV and SK_ID_CURR, grouping by both tables by these both the ids by mean and then inner join on PREV only

In [167…
```
df_ccbal_installment1 = pd.merge(left=df_installment1, right=df_ccbal1, how='inn
                                 right_on=['SK_ID_PREV'])
```

In [168…
```
df_ccbal_installment1.head()
```

Out[168…

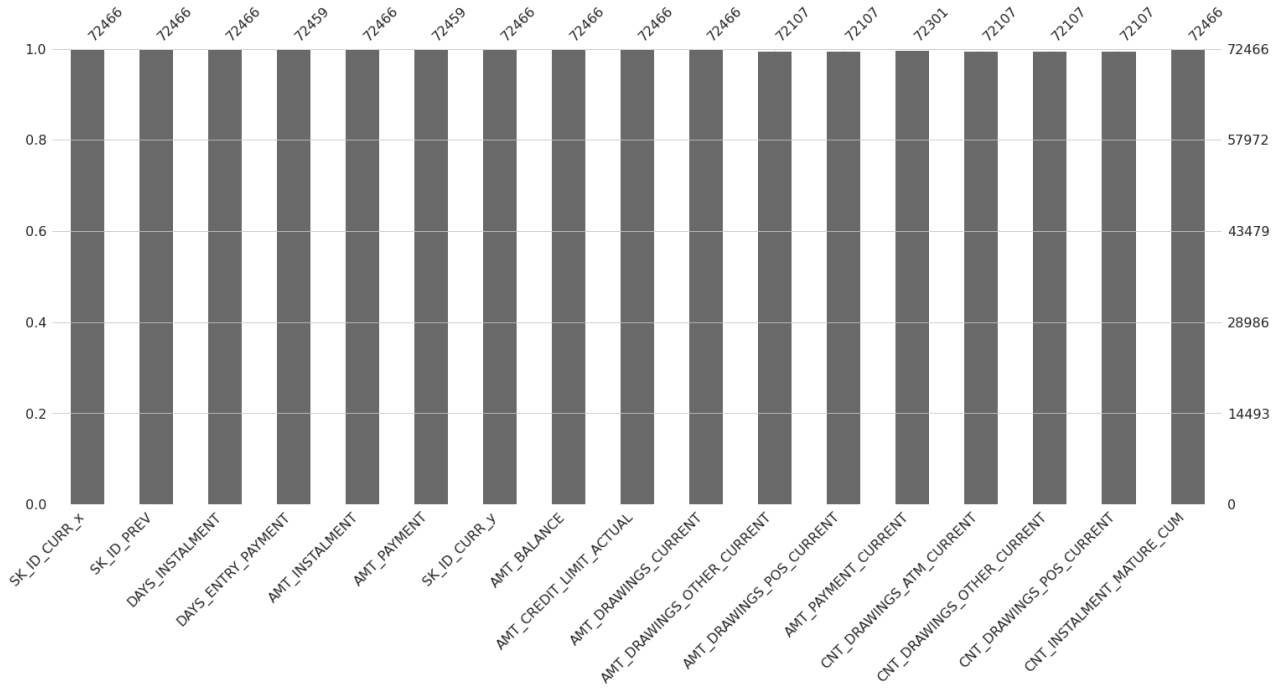
	SK_ID_CURR_x	SK_ID_PREV	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT
0	100011	1843384	-1139.157895	-1142.434211	4723.027697
1	100013	2038692	-1562.657895	-1565.903509	6107.987368
2	100028	1914954	-700.892857	-702.559524	4215.151071
3	100042	2137382	-1474.926230	-1479.500000	5205.409672
4	100043	1557583	-663.547619	-666.785714	17736.775714

EDA and Preprocessing on the merged file (credit card balance and installment payments)

> **We are doing this even after merging each individual table is already processed because there might be some columns that are now insignificant or higly correlated with each other.**

In [169…
```
msno.bar(df_ccbal_installment1)
```

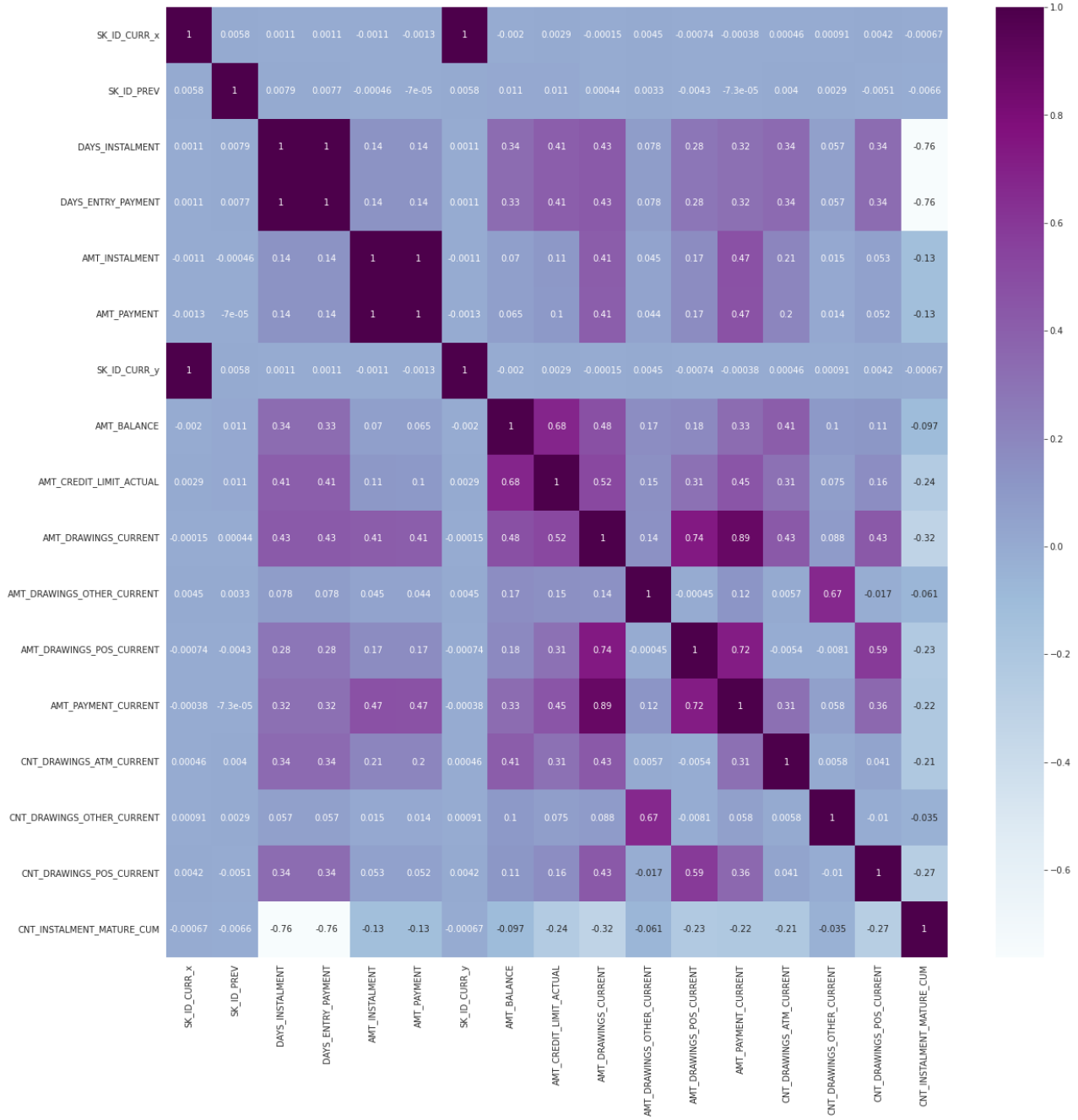
Out[169…
```
<AxesSubplot:>
```



In [170…
```
df_ccbal_installment1.to_csv('df_ccbal_installment.csv',index=False)
```

In [171…
```
plt.figure(figsize = (20,20))
sns.heatmap(df_ccbal_installment1.corr(),annot=True, cmap='BuPu')
```

Out[171…
```
<AxesSubplot:>
```



In [172…
```
df_ccbal_installment1 = df_ccbal_installment1.drop(['SK_ID_CURR_y','AMT_DRAWINGS
                                                    'AMT_PAYMENT','DAYS_ENTRY_PAY
```

In [173…
```
df_ccbal_installment1.rename(columns={'SK_ID_CURR_x':'SK_ID_CURR'},inplace=True)
```

In [174…
```
df_ccbal_installment1.head()
```

Out[174…

	SK_ID_CURR	SK_ID_PREV	DAYS_INSTALMENT	AMT_INSTALMENT	AMT_BALANCE	AMT_CRED
0	100011	1843384	-1139.157895	4723.027697	54482.111149	
1	100013	2038692	-1562.657895	6107.987368	18159.919219	
2	100028	1914954	-700.892857	4215.151071	8085.058163	

	SK_ID_CURR	SK_ID_PREV	DAYS_INSTALMENT	AMT_INSTALMENT	AMT_BALANCE	AMT_CRED
3	100042	2137382	-1474.926230	5205.409672	33356.183036	
4	100043	1557583	-663.547619	17736.775714	208572.600000	

Working on Previous Application Data

In [175…
```
df_prev_app =  datasets['previous_application']
```

In [176…
```
df_prev_app.head()
```

Out[176…

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_
0	2030495	271877	Consumer loans	1730.430	17145.0	
1	2802425	108129	Cash loans	25188.615	607500.0	6
2	2523466	122040	Cash loans	15060.735	112500.0	1
3	2819243	176158	Cash loans	47041.335	450000.0	4
4	1784265	202054	Cash loans	31924.395	337500.0	4

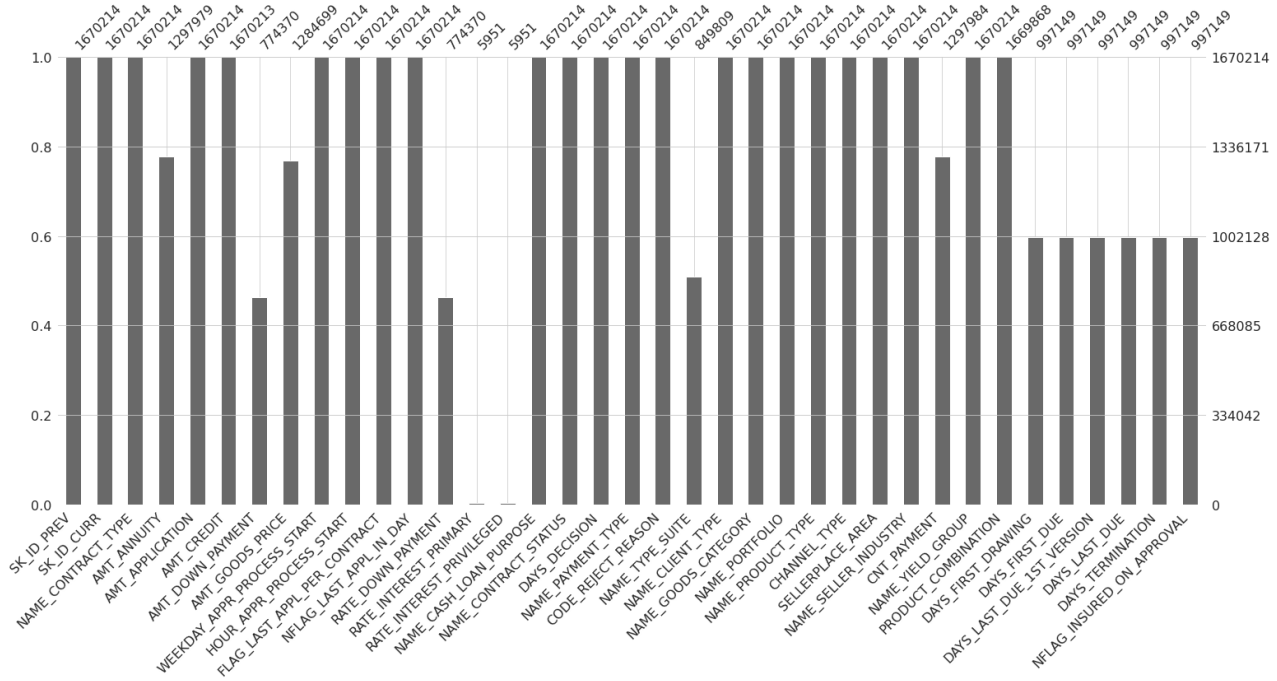
5 rows × 37 columns

EDA and Preprocessing of Previous Application data

> **removing columns with high correlation and containing large percentage of NULL values**

In [177…
```
msno.bar(df_prev_app)
```

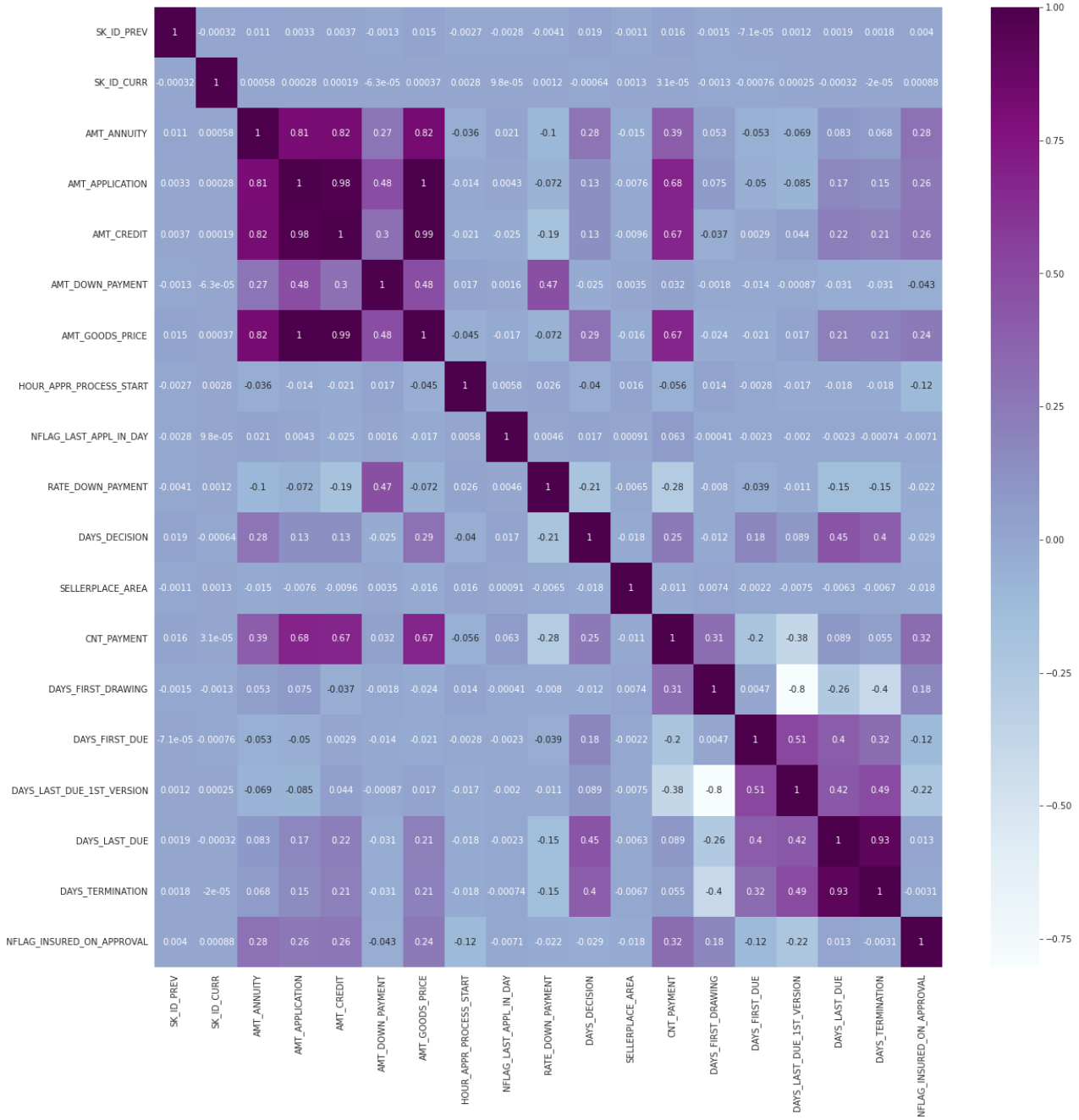
Out[177…
```
<AxesSubplot:>
```



In [178…
```
df_prev_app = df_prev_app.drop(['RATE_INTEREST_PRIVILEGED','RATE_INTEREST_PRIMAR
```

In [179…
```
plt.figure(figsize = (20,20))
sns.heatmap(df_prev_app.corr(),annot=True, cmap='BuPu')
```

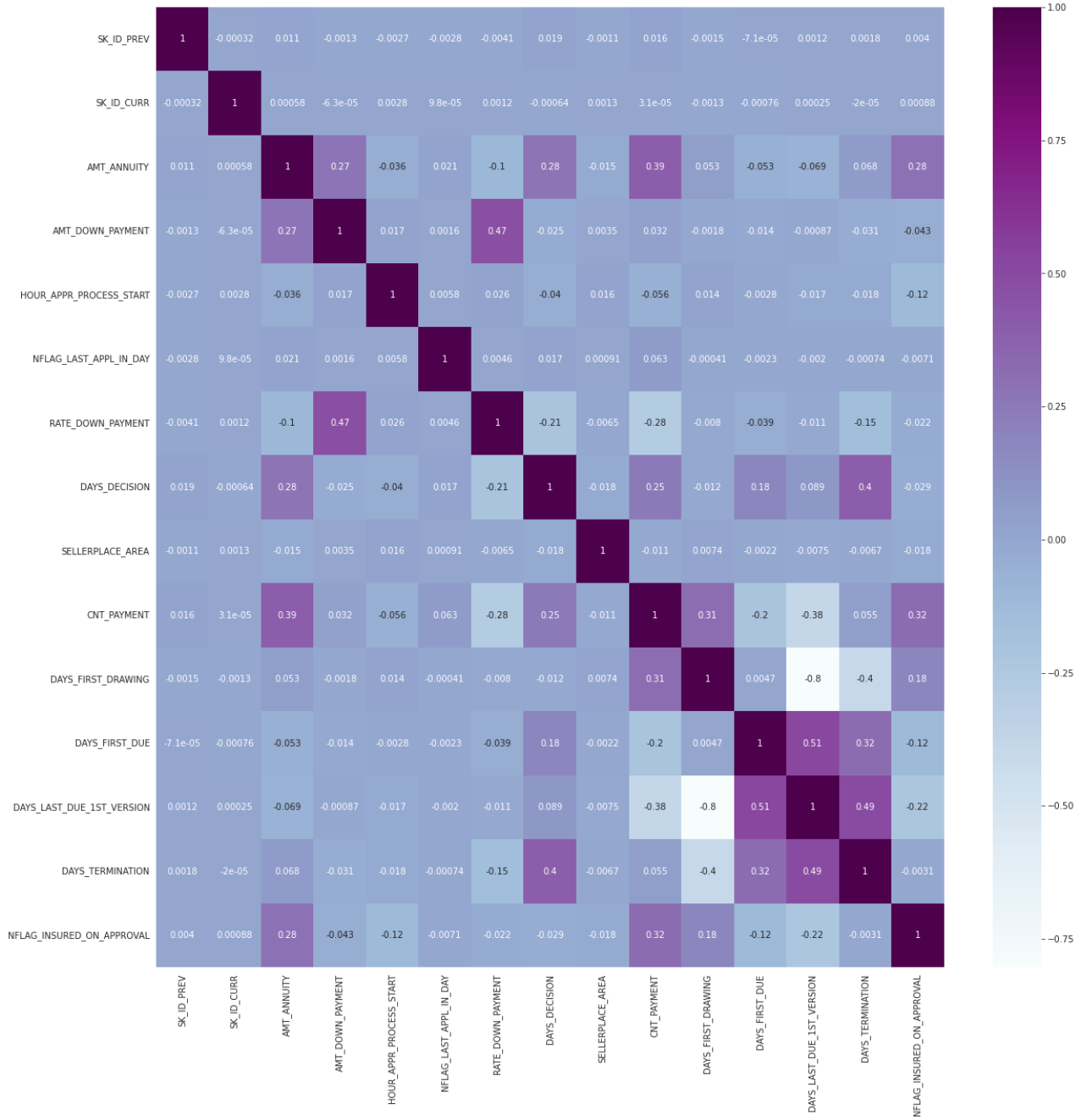
Out[179…
```
<AxesSubplot:>
```



In [180…
```
df_prev_app = df_prev_app.drop(['AMT_APPLICATION','AMT_CREDIT','AMT_GOODS_PRICE'
```

In [181…
```
plt.figure(figsize = (20,20))
sns.heatmap(df_prev_app.corr(),annot=True, cmap='BuPu')
```

Out[181…
```
<AxesSubplot:>
```



In [182…

```
df_prev_app = df_prev_app.drop(['WEEKDAY_APPR_PROCESS_START','HOUR_APPR_PROCESS_
                                'NAME_SELLER_INDUSTRY','NAME_CASH_LOAN_PURPOSE',
       'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE',
       'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE',
       'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY',
       'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION','DAYS_LAST_DUE_1
       'FLAG_LAST_APPL_PER_CONTRACT','NFLAG_INSURED_ON_APPROVAL',
      'NFLAG_LAST_APPL_IN_DAY','NAME_CONTRACT_TYPE','DAYS_FIRST_DRAWING'],axis=1)
```

In [183…

```
df_prev_app.head()
```

Out[183…

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_DOWN_PAYMENT	RATE_DOWN_PAYMENT	DA
0	2030495	271877	1730.430	0.0	0.0	

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_DOWN_PAYMENT	RATE_DOWN_PAYMENT	DA
1	2802425	108129	25188.615	NaN	NaN	
2	2523466	122040	15060.735	NaN	NaN	
3	2819243	176158	47041.335	NaN	NaN	
4	1784265	202054	31924.395	NaN	NaN	

In [184…
```
df_prev_app = df_prev_app.drop(['AMT_DOWN_PAYMENT','RATE_DOWN_PAYMENT'],axis=1)
```

> **Grouping by SK_ID_CURR and SK_ID_PREV by mean so that we can restore as much information as we can while merging or joining**

In [185…
```
df_prev_app = df_prev_app.groupby(['SK_ID_PREV','SK_ID_CURR']).mean()
```

In [186…
```
df_prev_app = df_prev_app.reset_index()
```

In [187…
```
df_prev_app.head()
```

Out[187…

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	DAYS_FIRST_DUE	DAYS_TERMINATION
0	1000001	158271	6404.310	-268.0	-233.0
1	1000002	101962	6264.000	-1600.0	-1501.0
2	1000003	252457	4951.350	-94.0	365243.0
3	1000004	260094	3391.110	-862.0	-672.0
4	1000005	176456	14713.605	-1688.0	-1415.0

In [188…
```
df_prev_app.shape
```

Out[188…
```
(1670214, 5)
```

Merging of Previous Application and previously merged dataframe (Credit card balance + Installments Payments)

In [189…
```
df_prev_ccbalinst = pd.merge(left=df_prev_app, right=df_ccbal_installment1, how=
                             right_on=['SK_ID_PREV','SK_ID_CURR'])
```

In [190…
```
df_prev_ccbalinst.shape
```

Out[190…
```
(62271, 15)
```

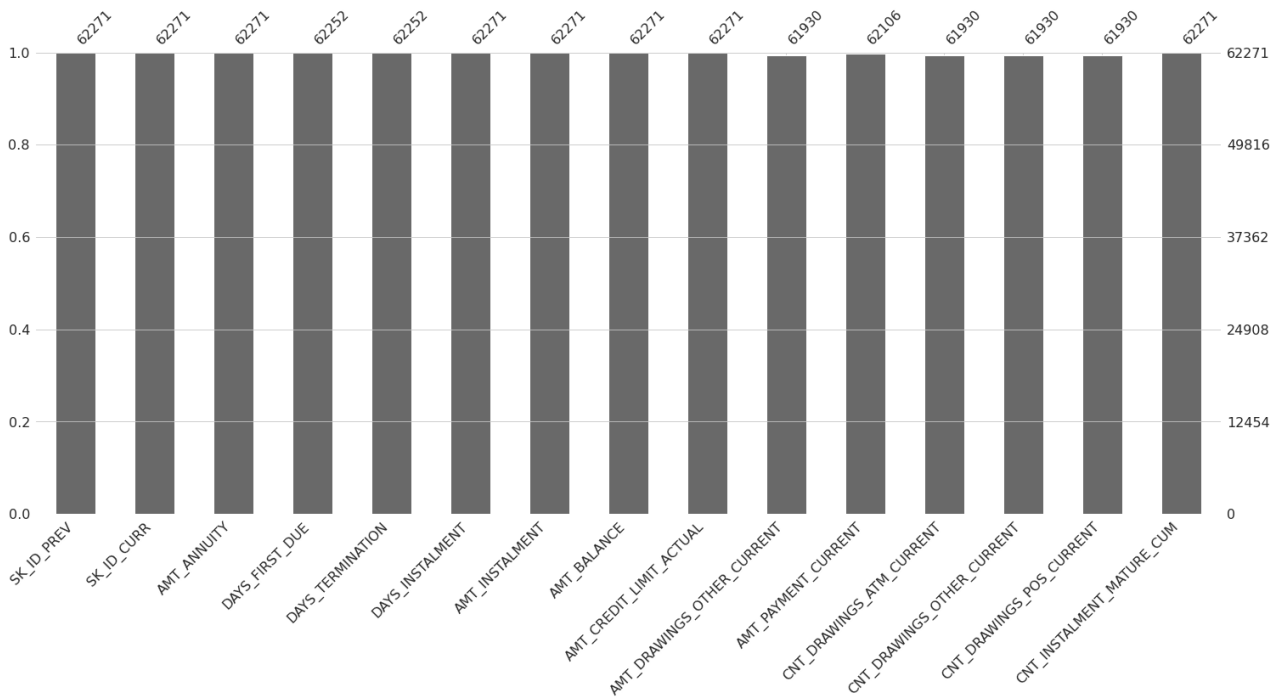
EDA and Preprocessing of Merged three Datasets (Previous

Application + Credit card balance + Installments Payments)

> **We are doing this even after merging each individual table is already processed because there might be some columns that are now insignificant or higly correlated with each other.**

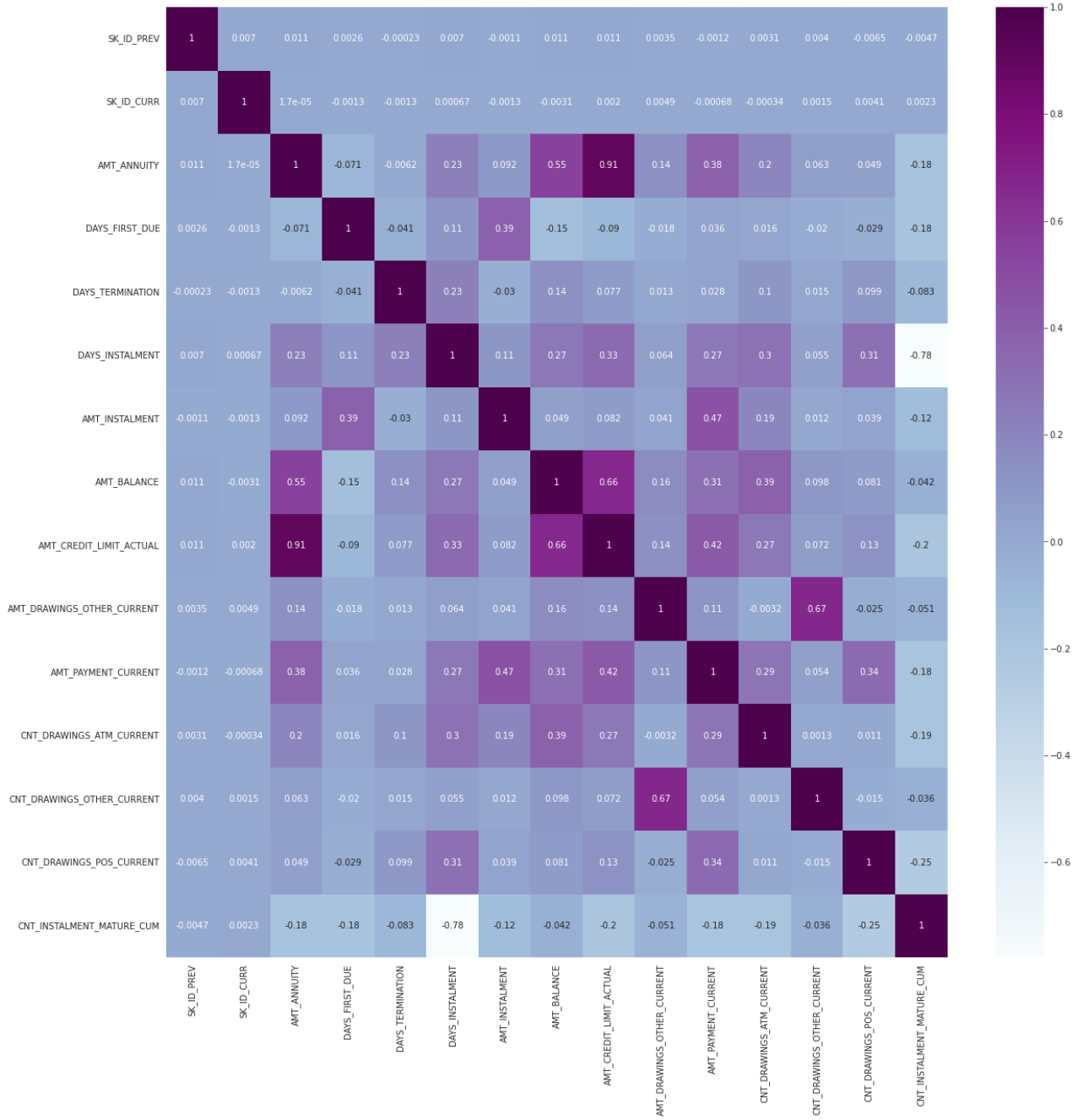
In [191…
```
msno.bar(df_prev_ccbalinst)
```

Out[191…
```
<AxesSubplot:>
```



In [192…
```
plt.figure(figsize = (20,20))
sns.heatmap(df_prev_ccbalinst.corr(),annot=True, cmap='BuPu')
```

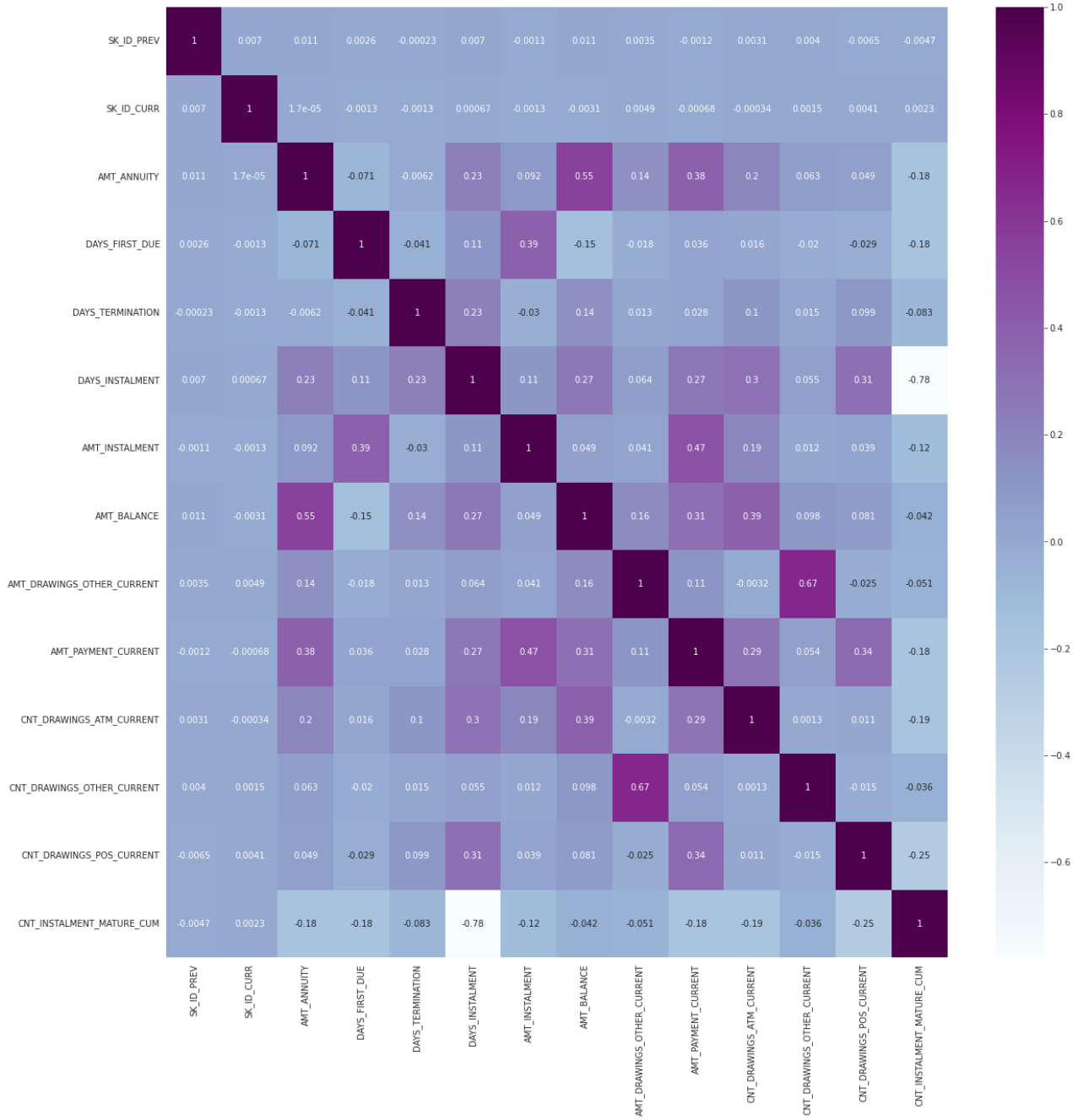
Out[192…
```
<AxesSubplot:>
```



In [193…
```
df_prev_ccbalinst = df_prev_ccbalinst.drop(['AMT_CREDIT_LIMIT_ACTUAL'],axis=1)
```

In [194…
```
plt.figure(figsize = (20,20))
sns.heatmap(df_prev_ccbalinst.corr(),annot=True, cmap='BuPu')
```

Out[194…
```
<AxesSubplot:>
```



df_prev_ccbalinst: Merged DataFrame(Previous Application + Credit card balance + Installments Payments)

Merging of df_PREV_CCBALINST with already merged Application_train on Bureau Data

> **Left joining on SK_ID_CURR**

In [195…

```
df_train_prev_ccbalinstal = pd.merge(left=df_finalm_bureau_train, right=df_prev_
                                     left_on=['SK_ID_CURR'],
                                    right_on=['SK_ID_CURR'])
```

In [196…
```
df_train_prev_ccbalinstal.shape
```

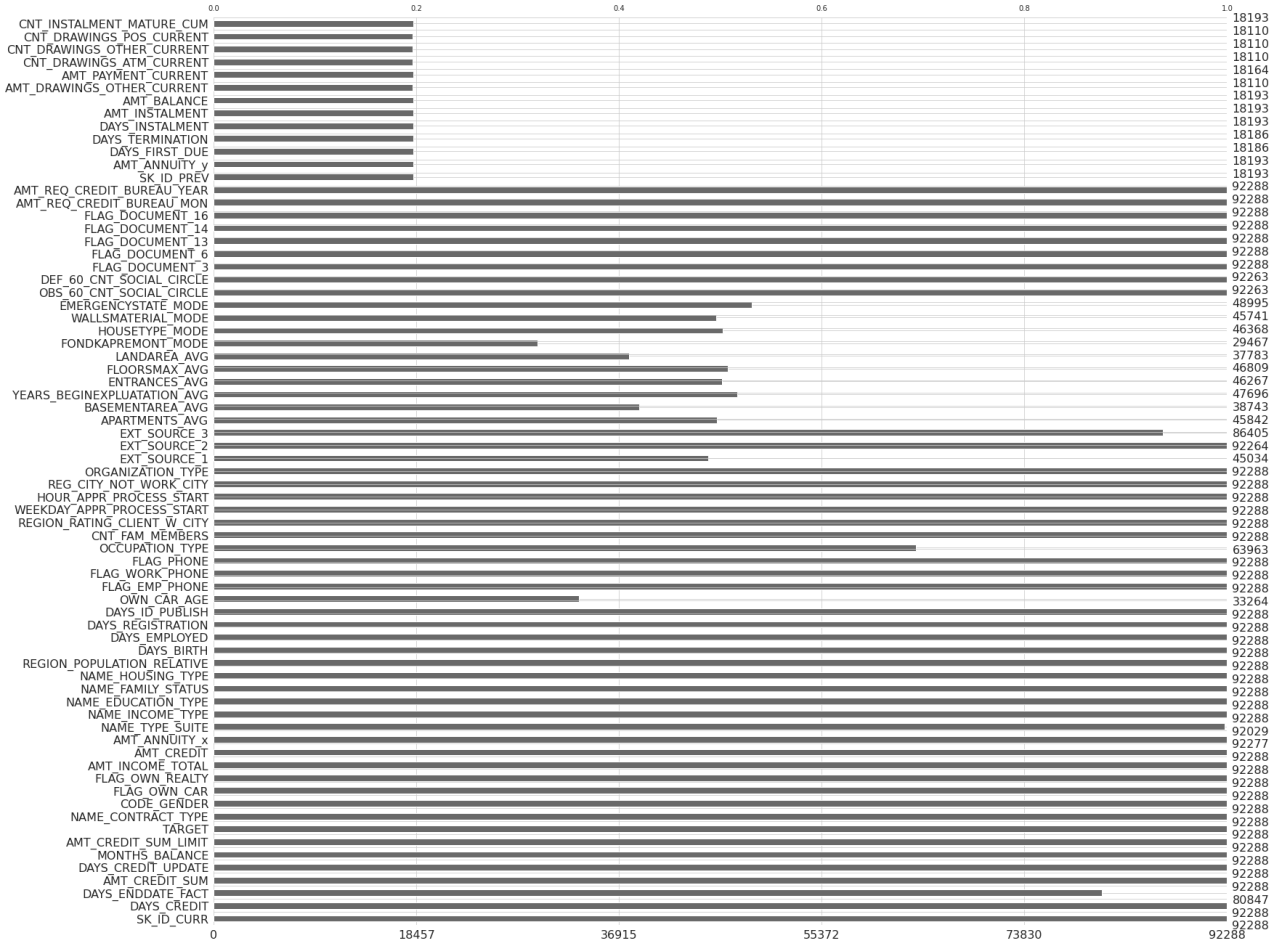
Out[196…
```
(92288, 71)
```

EDA and Preprocessing of Final Merged Data

> **We are doing this even after merging each individual table is already processed because there might be some columns that are now insignificant or higly correlated with each other.**

In [197…
```
msno.bar(df_train_prev_ccbalinstal)
```

Out[197…
```
<AxesSubplot:>
```



In [198…
```
df_train_prev_ccbalinstal.to_csv('final_dataframe.csv',index= False)
```

In [199…
```
df_final_merged = df_train_prev_ccbalinstal
```

In [200…
```
df_final_merged.head()
```

Out[200…

	SK_ID_CURR	DAYS_CREDIT	DAYS_ENDDATE_FACT	AMT_CREDIT_SUM	DAYS_CREDIT_UPDATE
0	100002	-1043.0	-967.0	40761.0	-758.0
1	100010	-1939.5	-1138.0	495000.0	-578.0
2	100019	-495.0	NaN	360000.0	-26.5
3	100032	-1169.5	-662.0	331875.0	-520.5
4	100033	-195.0	NaN	675000.0	-169.0

5 rows × 71 columns

In [201…

```
df_train_prev_ccbalinstal.head()
```

Out[201…

	SK_ID_CURR	DAYS_CREDIT	DAYS_ENDDATE_FACT	AMT_CREDIT_SUM	DAYS_CREDIT_UPDATE
0	100002	-1043.0	-967.0	40761.0	-758.0
1	100010	-1939.5	-1138.0	495000.0	-578.0
2	100019	-495.0	NaN	360000.0	-26.5
3	100032	-1169.5	-662.0	331875.0	-520.5
4	100033	-195.0	NaN	675000.0	-169.0

5 rows × 71 columns

Feature Engineering

> **Feature engineering an important part of machine-learning as we try to modify/create (i.e., engineer) new features from our existing dataset that might be meaningful in predicting the TARGET.**

> **Correlation of features with Target Variable in order to engineer new fatures**

In [202…

```
df_final_merged.corrwith(df_final_merged["TARGET"]).sort_values(ascending = Fals
```

Out[202…

```
TARGET                        1.000000
CNT_DRAWINGS_ATM_CURRENT      0.108122
DAYS_INSTALMENT               0.082245
DAYS_CREDIT                   0.077823
DAYS_BIRTH                    0.076995
MONTHS_BALANCE                0.069535
DAYS_CREDIT_UPDATE            0.060649
REGION_RATING_CLIENT_W_CITY   0.058259
REG_CITY_NOT_WORK_CITY        0.056437
DAYS_ENDDATE_FACT             0.052098
AMT_BALANCE                   0.050537
FLAG_EMP_PHONE                0.049907
DAYS_ID_PUBLISH               0.044734
DAYS_REGISTRATION             0.043915
```

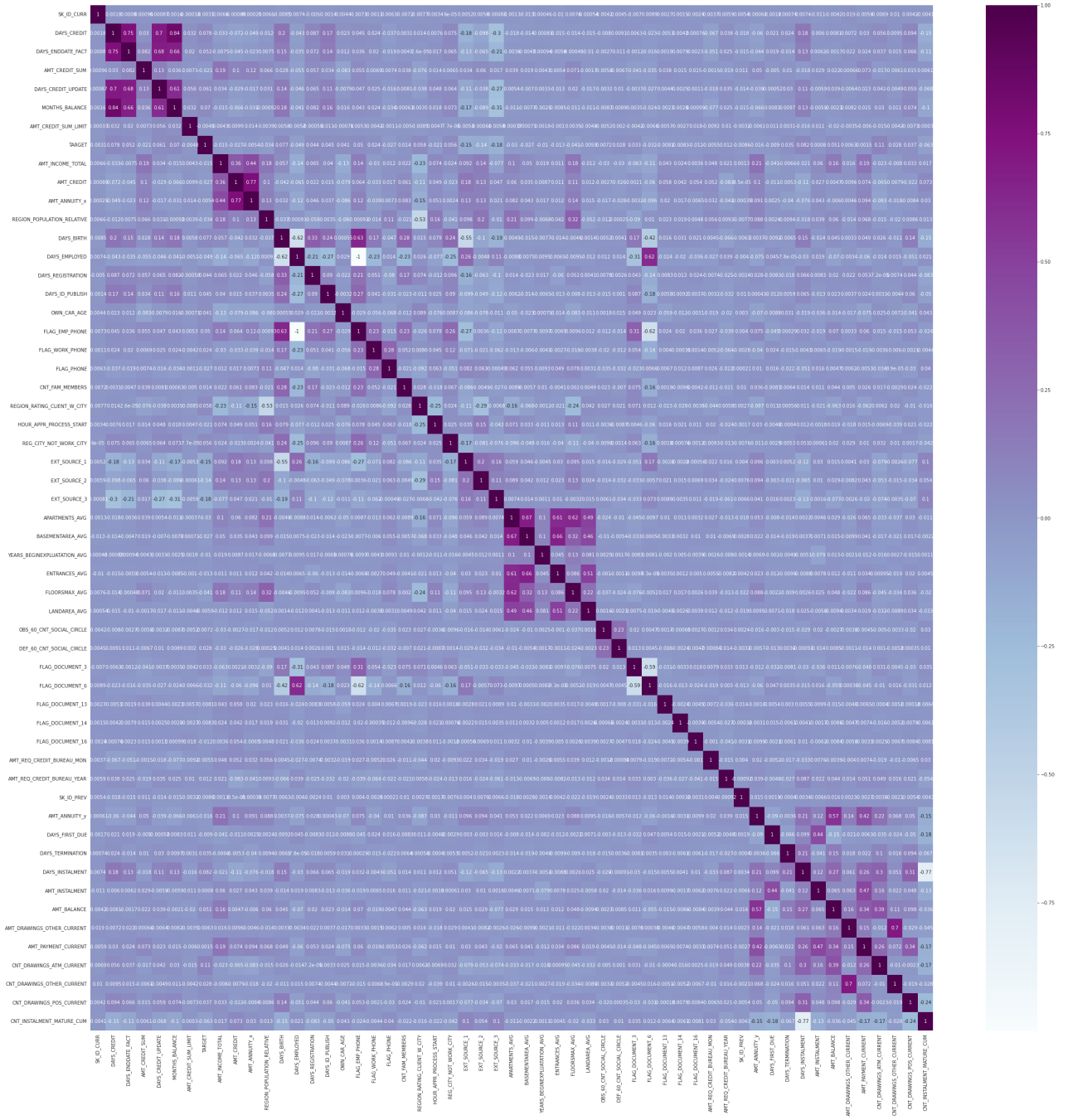
```
OWN_CAR_AGE                    0.041009
CNT_DRAWINGS_POS_CURRENT       0.037029
DAYS_TERMINATION               0.034801
FLAG_DOCUMENT_3                0.033429
DEF_60_CNT_SOCIAL_CIRCLE       0.028113
CNT_DRAWINGS_OTHER_CURRENT     0.027778
FLAG_WORK_PHONE                0.024296
CNT_FAM_MEMBERS                0.014143
AMT_REQ_CREDIT_BUREAU_YEAR     0.011893
OBS_60_CNT_SOCIAL_CIRCLE       0.007209
AMT_DRAWINGS_OTHER_CURRENT     0.006293
AMT_PAYMENT_CURRENT            0.001518
AMT_INSTALMENT                 0.000804
SK_ID_CURR                    -0.003079
AMT_CREDIT_SUM_LIMIT          -0.004759
AMT_ANNUITY_x                 -0.005442
AMT_REQ_CREDIT_BUREAU_MON     -0.005491
LANDAREA_AVG                  -0.005879
FLAG_DOCUMENT_13              -0.008082
FLAG_DOCUMENT_14              -0.008331
SK_ID_PREV                    -0.008585
DAYS_FIRST_DUE                -0.008960
YEARS_BEGINEXPLUATATION_AVG   -0.010040
FLAG_DOCUMENT_16              -0.011845
ENTRANCES_AVG                 -0.013182
AMT_INCOME_TOTAL              -0.015043
AMT_ANNUITY_y                 -0.016477
HOUR_APPR_PROCESS_START       -0.020789
AMT_CREDIT_SUM                -0.021473
AMT_CREDIT                    -0.026572
FLAG_PHONE                    -0.027116
BASEMENTAREA_AVG              -0.027292
APARTMENTS_AVG                -0.029746
FLAG_DOCUMENT_6               -0.032099
REGION_POPULATION_RELATIVE    -0.034377
FLOORSMAX_AVG                 -0.041341
DAYS_EMPLOYED                 -0.048916
CNT_INSTALMENT_MATURE_CUM     -0.063045
EXT_SOURCE_2                  -0.136431
EXT_SOURCE_1                  -0.148456
EXT_SOURCE_3                  -0.175782
dtype: float64
```

In [203…

```
plt.figure(figsize = (40,40))
sns.heatmap(df_final_merged.corr(),annot=True, cmap='BuPu')
```

Out[203…

```
<AxesSubplot:>
```



Expert knowledge features

> Often, experts have domain knowledge about what combination of existing features have strong explanatory/predictive power. In this case we are looking at the following features

Feature 1

In [204…

```
df_final_merged['DAYS_EMPLOYED_PCT'] = df_final_merged['DAYS_EMPLOYED'] / df_fin
```

> **df_final_merged['DAYS_EMPLOYED_PCT'] =**

> **df_final_merged['DAYS_EMPLOYED'] / df_final_merged['DAYS_BIRTH']**

- Percentage of days employed - How long a person has been employed as a percentage of his life is a stronger predictor of his ability to keep paying off his loans.
- We have taken this parameter because the total number of days a person has been employed in his life time will impact in his credit paying ability and since this is highly correlated to the target variable we have taken this feature to create new feature out of it.

Feature 2

In [205…
```
df_final_merged['CREDIT_INCOME_PCT'] = df_final_merged['AMT_CREDIT'] / df_final_
```

> df_final_merged['CREDIT_INCOME_PCT'] = df_final_merged['AMT_CREDIT'] / df_final_merged['AMT_INCOME_TOTAL']

- Available credit as a percentage of income - If a person has a very large amount of credit available as a percentage of income, this can impact his ability to pay off the loans
- If a person has a very high credit limit with respect to its earning then this will effect much in its loan paying capability and he or she will or maybe most of the times will not be able to pay back.

Feature 3

In [206…
```
df_final_merged['ANNUITY_INCOME_PCT'] = df_final_merged['AMT_ANNUITY_x'] / df_fi
```

> df_final_merged['ANNUITY_INCOME_PCT'] = df_final_merged['AMT_ANNUITY_x'] / df_final_merged['AMT_INCOME_TOTAL']

- Annuity as a percentage of income - If a person receives an annuity, this is a more stable source of income thus if it is higher, you are less likely to default.
- We have taken this feature because of the reason mentioned in the previous line and since it is highly correlated with the target variable.

Feature 4

In [207…
```
df_final_merged['CREDIT_TERM_PCT'] = df_final_merged['AMT_ANNUITY_x'] / df_final
```

> df_final_merged['CREDIT_TERM_PCT'] = df_final_merged['AMT_ANNUITY_x'] / df_final_merged['AMT_CREDIT']

- Annuity as a percentage of available credit - If a person receives an annuity, this is more stable source of income thus if it is a high percentage compared to his/her credit availability then the person is more likely be able to pay off his debts.

Feature 5

In [208…

```
df_final_merged['AMT_BALANCE_PCT'] = df_final_merged['AMT_BALANCE'] / df_final_m
```

> df_final_merged['AMT_BALANCE_PCT'] = df_final_merged['AMT_BALANCE'] / df_final_merged['DAYS_CREDIT']

- Amount Balance as the percentage - The remaining balance in the account is very much an important feature as this tells much about a person's ability to pay back it's debt.
- This is the reason that we took into accoun this feature also because of it's correlation with the target feature.

Feature 6

In [209…

```
df_final_merged['AVG_INCOME_EXT_PCT'] = (df_final_merged['EXT_SOURCE_1']+df_fina
                                         +df_final_merged['EXT_SOURCE_3'])/3
```

> df_final_merged['AVG_INCOME_EXT_PCT'] = (df_final_merged['EXT_SOURCE_1']+df_final_merged['EXT_SOURCE_2'] +df_final_merged['EXT_SOURCE_3'])/3

- Average of three External Sources of Income - Since these three features are the highest among all the other features that are correlated with the target variable, we took the average of all the three incomes and created a new feature that tells us how they are effecting the prediction of target varible.
- This feature is almost 13% correlated.

Feature 7

In [210…

```
df_final_merged['AVG_TOTALINCOME_PCT'] = df_final_merged['AMT_INCOME_TOTAL']/df_
```

> df_final_merged['AVG_TOTALINCOME_PCT'] = df_final_merged['AMT_INCOME_TOTAL']/df_final_merged['AVG_INCOME_EXT_PCT']

- Total Income avergae as percentage - This is the best feature that we generated out of all the seven newly made features. This shows the total income as a percent of the person's earning with respect to it's external income.
- We engineered this using the Feature 6 that we created above and this is almost 22% negatively correlated with the get variable.

In [211…
```
# Training dataset
# df_final_merged['DAYS_EMPLOYED_PCT'] = df_final_merged['DAYS_EMPLOYED'] / df_f
# df_final_merged['CREDIT_INCOME_PCT'] = df_final_merged['AMT_CREDIT'] / df_fina
# df_final_merged['ANNUITY_INCOME_PCT'] = df_final_merged['AMT_ANNUITY_x'] / df_
# df_final_merged['CREDIT_TERM_PCT'] = df_final_merged['AMT_ANNUITY_x'] / df_fin
# df_final_merged['AMT_BALANCE_PCT'] = df_final_merged['AMT_BALANCE'] / df_final
# df_final_merged['AVG_INCOME_EXT_PCT'] = (df_final_merged['EXT_SOURCE_1']+df_fi
#                                          +df_final_merged['EXT_SOURCE_3'])/
# df_final_merged['AVG_TOTALINCOME_PCT'] = df_final_merged['AMT_INCOME_TOTAL']/d
```

In [212…
```
df_final_merged.columns
```

Out[212…
```
Index(['SK_ID_CURR', 'DAYS_CREDIT', 'DAYS_ENDDATE_FACT', 'AMT_CREDIT_SUM',
       'DAYS_CREDIT_UPDATE', 'MONTHS_BALANCE', 'AMT_CREDIT_SUM_LIMIT',
       'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
       'FLAG_OWN_REALTY', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY_x',
       'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE',
       'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH',
       'OWN_CAR_AGE', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_PHONE',
       'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT_W_CITY',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'REG_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE', 'EXT_SOURCE_1',
       'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG',
       'YEARS_BEGINEXPLUATATION_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG',
       'LANDAREA_AVG', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE',
       'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE', 'OBS_60_CNT_SOCIAL_CIRCLE',
       'DEF_60_CNT_SOCIAL_CIRCLE', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_6',
       'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_16',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'SK_ID_PREV',
       'AMT_ANNUITY_y', 'DAYS_FIRST_DUE', 'DAYS_TERMINATION',
       'DAYS_INSTALMENT', 'AMT_INSTALMENT', 'AMT_BALANCE',
       'AMT_DRAWINGS_OTHER_CURRENT', 'AMT_PAYMENT_CURRENT',
       'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_OTHER_CURRENT',
       'CNT_DRAWINGS_POS_CURRENT', 'CNT_INSTALMENT_MATURE_CUM',
       'DAYS_EMPLOYED_PCT', 'CREDIT_INCOME_PCT', 'ANNUITY_INCOME_PCT',
       'CREDIT_TERM_PCT', 'AMT_BALANCE_PCT', 'AVG_INCOME_EXT_PCT',
       'AVG_TOTALINCOME_PCT'],
      dtype='object')
```

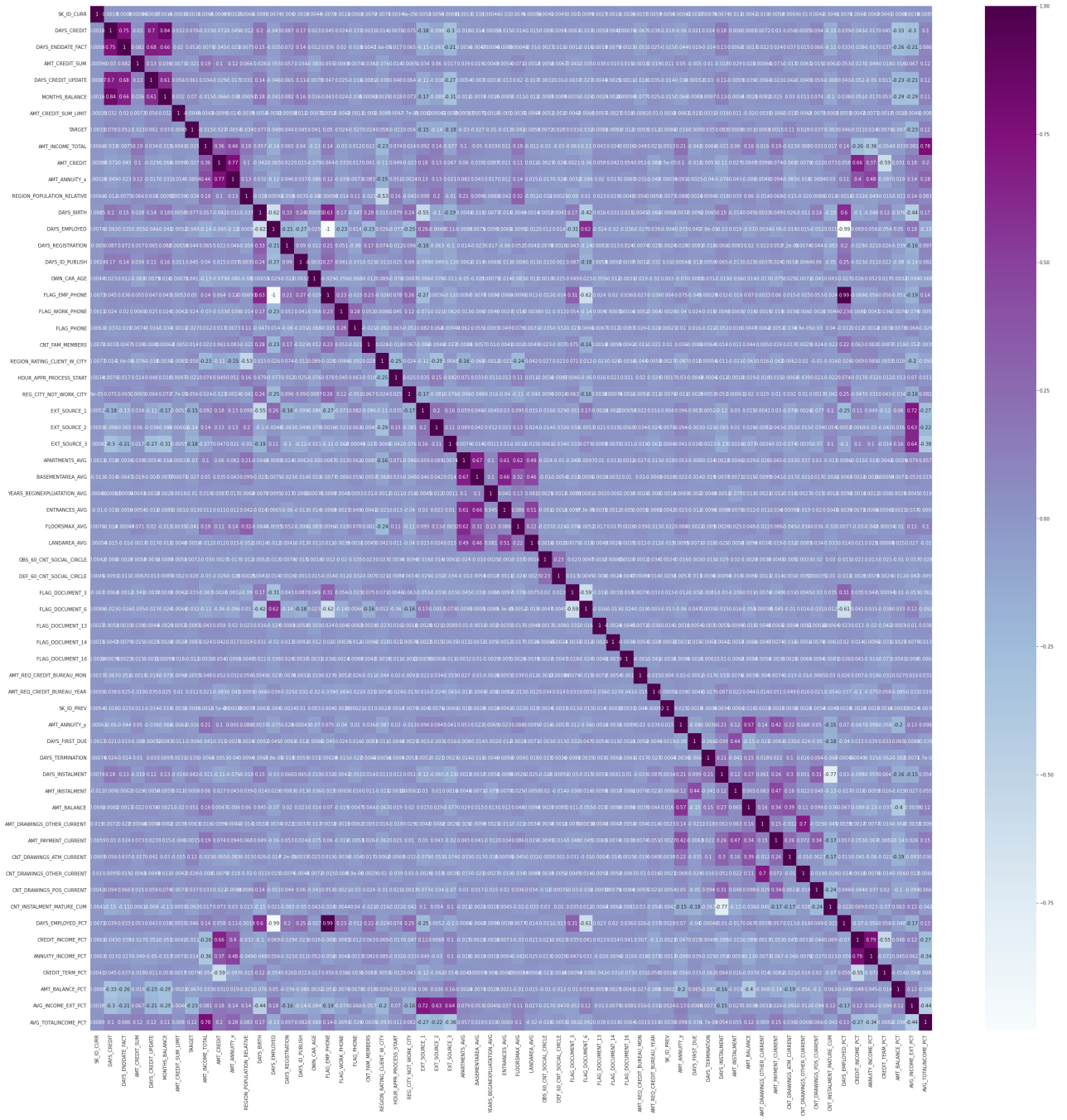
In [213…
```
df_final_merged.corrwith(df_final_merged["TARGET"]).sort_values(ascending = Fals
```

Out[213…
```
TARGET                       1.000000
AVG_TOTALINCOME_PCT          0.123221
CNT_DRAWINGS_ATM_CURRENT     0.108122
DAYS_INSTALMENT              0.082245
DAYS_CREDIT                  0.077823
                               ...
AMT_BALANCE_PCT             -0.066617
EXT_SOURCE_2                -0.136431
EXT_SOURCE_1                -0.148456
EXT_SOURCE_3                -0.175782
AVG_INCOME_EXT_PCT          -0.225926
Length: 62, dtype: float64
```

In [214…
```
plt.figure(figsize = (40,40))
sns.heatmap(df_final_merged.corr(),annot=True, cmap='BuPu')
```

Out [214...

```
<AxesSubplot:>
```



In [247...

```
plot_df = df_final_merged[['TARGET','DAYS_EMPLOYED_PCT', 'CREDIT_INCOME_PCT', 'A
       'CREDIT_TERM_PCT', 'AMT_BALANCE_PCT', 'AVG_INCOME_EXT_PCT',
       'AVG_TOTALINCOME_PCT']]
plot_df.head()
```

Out [247...

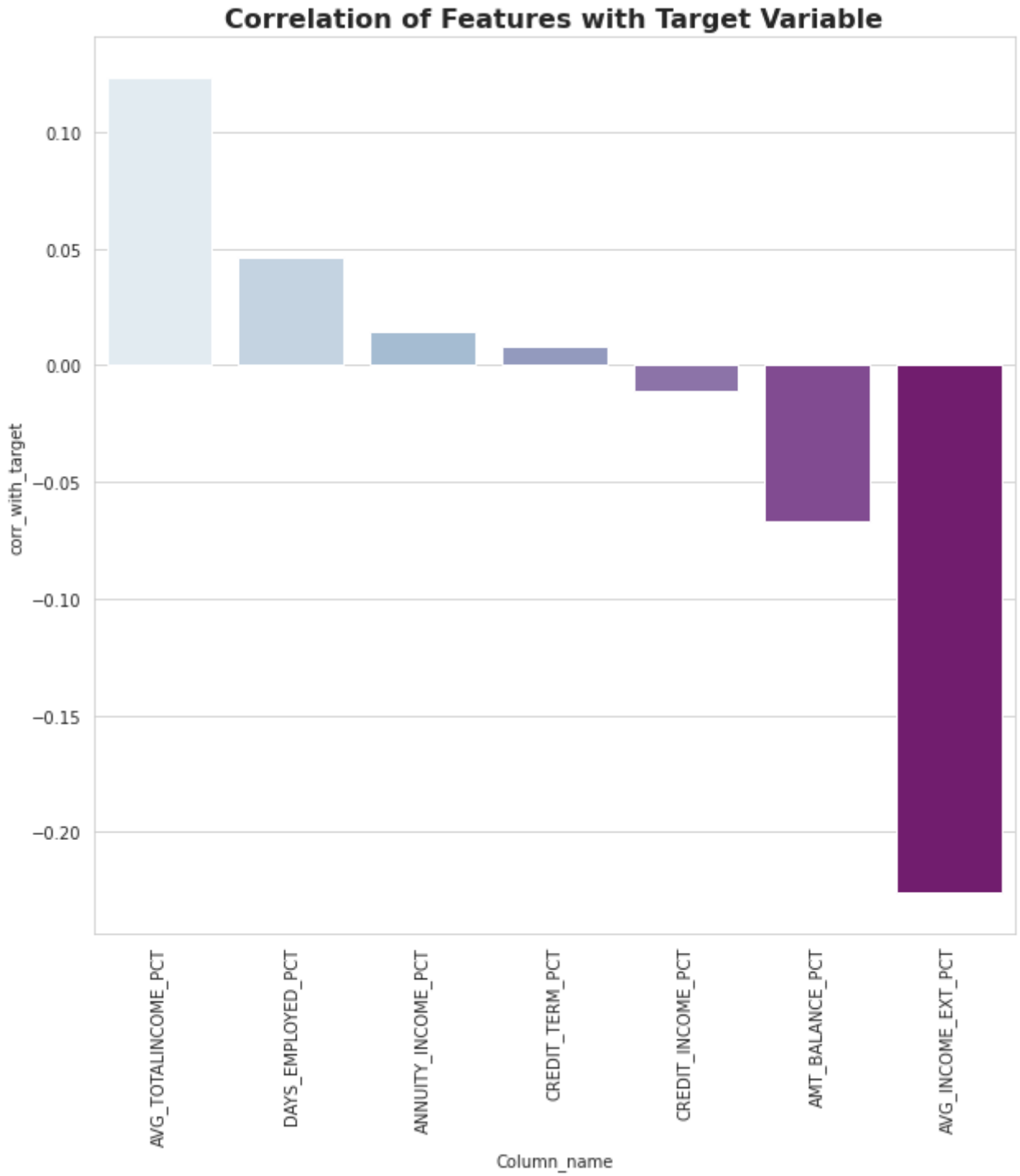
	TARGET	DAYS_EMPLOYED_PCT	CREDIT_INCOME_PCT	ANNUITY_INCOME_PCT	CREDIT_TERM_
0	1	0.067329	2.007889	0.121978	0.06
1	0	0.023820	4.250000	0.116875	0.02
2	0	0.132562	1.903314	0.128000	0.06
3	0	0.077376	2.906880	0.211800	0.07

	TARGET	DAYS_EMPLOYED_PCT	CREDIT_INCOME_PCT	ANNUITY_INCOME_PCT	CREDIT_TERM_
4	0	0.179708	2.929000	0.213617	0.07

In [248…
```
feature_plot = pd.DataFrame(plot_df.corrwith(plot_df["TARGET"]).sort_values(asce
```

In [249…
```
feature_plot = feature_plot.rename(columns={'index':'Column_name',0:'corr_with_t
```

In [250…
```
plt.figure(figsize=(10,10))
sns.barplot(data=feature_plot,x=feature_plot['Column_name'][1:], y= feature_plot
plt.xticks(rotation=90)
plt.title('Correlation of Features with Target Variable', fontweight = 'bold', f
plt.show()
```



Hyper-Parameter Tuning

In [215…
```
df_final_merged.head()
```

Out[215…

	SK_ID_CURR	DAYS_CREDIT	DAYS_ENDDATE_FACT	AMT_CREDIT_SUM	DAYS_CREDIT_UPDATE
0	100002	-1043.0	-967.0	40761.0	-758.0
1	100010	-1939.5	-1138.0	495000.0	-578.0
2	100019	-495.0	NaN	360000.0	-26.5
3	100032	-1169.5	-662.0	331875.0	-520.5

	SK_ID_CURR	DAYS_CREDIT	DAYS_ENDDATE_FACT	AMT_CREDIT_SUM	DAYS_CREDIT_UPDATE
4	100033	-195.0	NaN	675000.0	-169.0

5 rows × 78 columns

Seperating the dataframe in features and Target Varible

In [216…
```
X_merged = df_final_merged.drop(['TARGET'],axis=1)
y_merged = df_final_merged['TARGET']
```

In [217…
```
y_merged = pd.DataFrame(y_merged, columns = ["TARGET"])
y_merged
```

Out[217…

	TARGET
0	1
1	0
2	0
3	0
4	0
...	...
92283	0
92284	0
92285	0
92286	1
92287	0

92288 rows × 1 columns

Splitting the Data into Train and Test Data

In [218…
```
X_train_merged,X_test_merged,y_train_merged,y_test_merged = train_test_split(X_m
```

Seperating numerical and categorical variables to fit in Pipeline

In [219…
```
integer_df_merged= X_merged.select_dtypes(include='int64')
float_df_merged = X_merged.select_dtypes(include='float64')
numerical_df_merged = list(pd.concat([integer_df_merged,float_df_merged], axis=1
categorical_df_merged = list(X_merged.select_dtypes(include='object'))
print(categorical_df_merged)
print('                                  ')
print(numerical_df_merged)
```

```
['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_T
YPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NA
ME_HOUSING_TYPE', 'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION
_TYPE', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCY
STATE_MODE']

['SK_ID_CURR', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_ID_PUBLISH', 'FLAG_EMP_PHON
E', 'FLAG_WORK_PHONE', 'FLAG_PHONE', 'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_P
ROCESS_START', 'REG_CITY_NOT_WORK_CITY', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_6',
'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_16', 'DAYS_CREDIT', 'DAYS
_ENDDATE_FACT', 'AMT_CREDIT_SUM', 'DAYS_CREDIT_UPDATE', 'MONTHS_BALANCE', 'AMT_C
REDIT_SUM_LIMIT', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY_x', 'REGION_POP
ULATION_RELATIVE', 'DAYS_REGISTRATION', 'OWN_CAR_AGE', 'CNT_FAM_MEMBERS', 'EXT_S
OURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG',
'YEARS_BEGINEXPLUATATION_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'LANDAREA_AVG',
'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'AMT_REQ_CREDIT_BUREAU_M
ON', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'SK_ID_PREV', 'AMT_ANNUITY_y', 'DAYS_FIRST_DU
E', 'DAYS_TERMINATION', 'DAYS_INSTALMENT', 'AMT_INSTALMENT', 'AMT_BALANCE', 'AMT
_DRAWINGS_OTHER_CURRENT', 'AMT_PAYMENT_CURRENT', 'CNT_DRAWINGS_ATM_CURRENT', 'CN
T_DRAWINGS_OTHER_CURRENT', 'CNT_DRAWINGS_POS_CURRENT', 'CNT_INSTALMENT_MATURE_CU
M', 'DAYS_EMPLOYED_PCT', 'CREDIT_INCOME_PCT', 'ANNUITY_INCOME_PCT', 'CREDIT_TERM
_PCT', 'AMT_BALANCE_PCT', 'AVG_INCOME_EXT_PCT', 'AVG_TOTALINCOME_PCT']
```

Build processing pipelines

In this part of the project the focus is on constructing the pipeline. Since the data has both numerical and categorical features, it is required to create two pipelines (one for each category of data) because they require different transformations. After finishing that, the two pipelines should be unified to produce one full pipeline that performs transformation on all the dataset. After this we will perform hyperparmeter tuning using GridSearchCV and taking different parameters for training the model. We used Logistic Regression and Random forest algorithms.

In [220…

```
num_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer(strategy = 'median'))
])
cat_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

In [221…

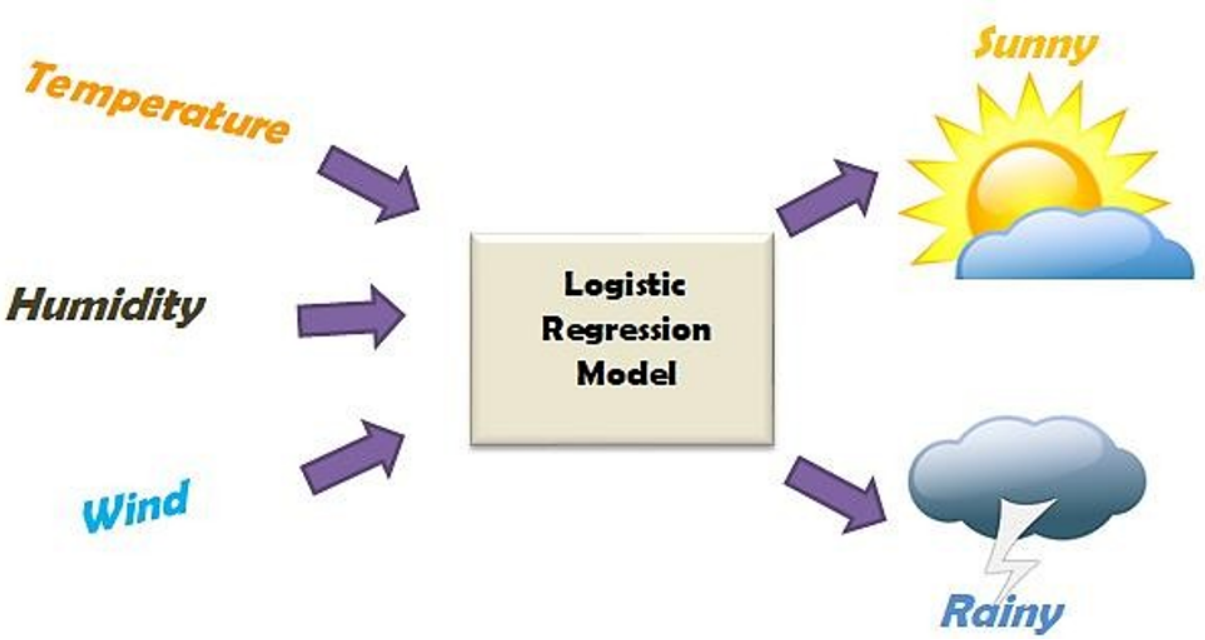
```
data_pipeline_merged = ColumnTransformer([
    ("num_pipeline", num_pipeline, numerical_df_merged),
    ("cat_pipeline", cat_pipeline, categorical_df_merged)])

model = Pipeline(steps = [('data_pipeline_merge',data_pipeline_merged),
                          ('classifier', LogisticRegression(solver='lbfgs',penalt
```

Modelling, Fitting and Storing Results

Logistic Regression with GridSearchCV

In [222…
```
from IPython.display import Image
Image(url= "https://prwatech.in/blog/wp-content/uploads/2020/02/logi1.png", widt
```

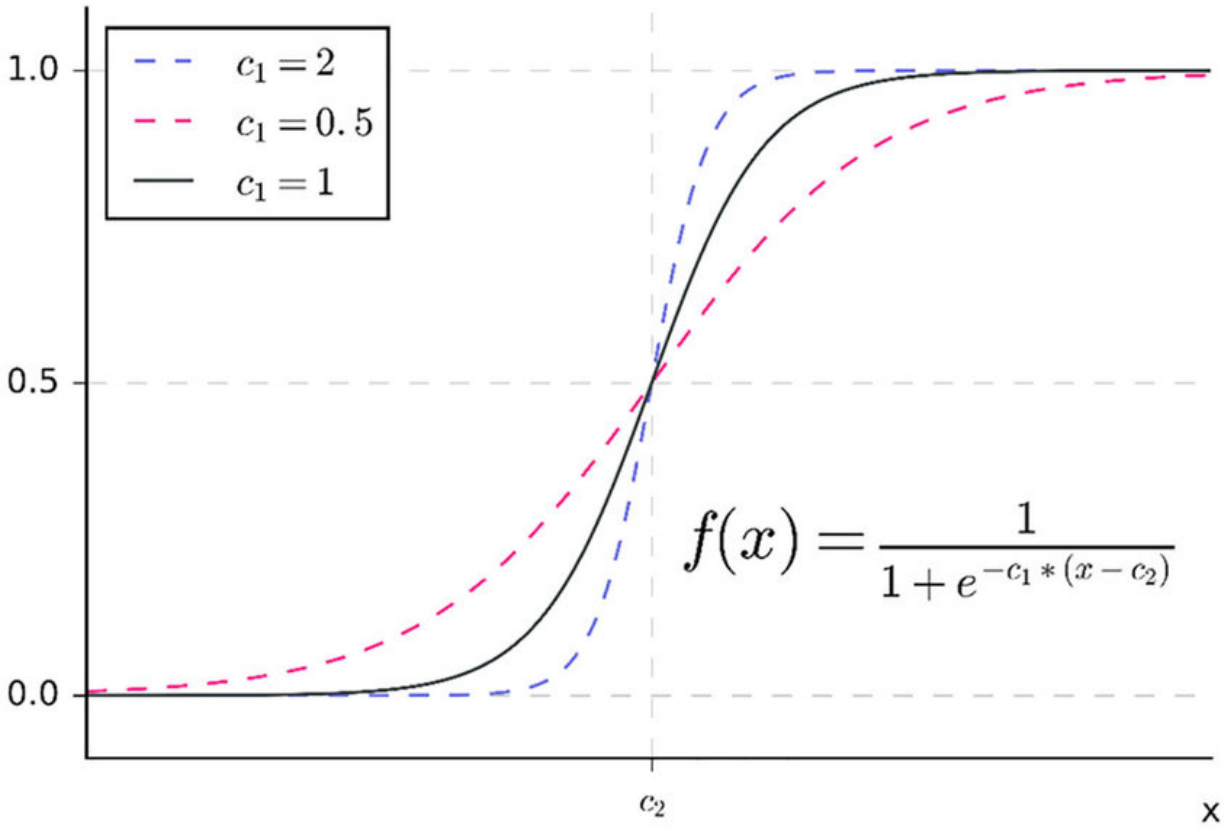
Out[222…



Logistic regression (LR) is a statistical method similar to linear regression since LR finds an equation that predicts an outcome for a binary variable, Y, from one or more response variables, X. However, unlike linear regression the response variables can be categorical or continuous, as the model does not strictly require continuous data. To predict group membership, LR uses the log odds ratio rather than probabilities and an iterative maximum likelihood method rather than a least squares to fit the final model. This means the researcher has more freedom when using LR and the method may be more appropriate for nonnormally distributed data or when the samples have unequal covariance matrices. Logistic regression assumes independence among variables, which is not always met in morphoscopic datasets.

In [223…
```
from IPython.display import Image

Image(url= "https://www.researchgate.net/publication/327512672/figure/fig2/AS:66
```

Out[223…



Prediction probability of Class 0 and Class 1

In [224…

```
param_grid = {
    'classifier__C': [0.1,1,10,100]
}
gs = GridSearchCV(model, param_grid, cv=5, n_jobs = -1, verbose = 2, refit = Tru
start = time()

gs.fit(X_train_merged,y_train_merged)

trainAcc  = gs.score(X_train_merged, y_train_merged)
validAcc  = gs.score(X_test_merged, y_test_merged)

start = time()
testAcc  = gs.score(X_test_merged, y_test_merged)
test_time = np.round(time() - start, 4)

#del experimentLog
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc"
                                               "Train Time(s)",  "Test Time(s)",
experimentLog.loc[len(experimentLog)] =[f"GridSearchCV LogReg", "HCDR",
                                        f"{trainAcc*100:8.2f}%", f"{validAcc*100
                                        train_time, test_time,
                                        "GridSearchCV LogReg pipeline with Cat+N


experimentLog
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

Out[224…

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Description

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Description
0	Baseline 1 LogReg	HCDR	91.91%	91.94%	91.94%	17.1005	0.4038	Baseline 1 LogReg pipeline with Cat+Num features
1	Baseline 1 RandomForest	HCDR	91.91%	91.95%	91.95%	10.7311	0.4899	Baseline 1 RandomForest pipeline with Cat+Num ...
2	GridSearchCV LogReg	HCDR	91.79%	91.99%	91.99%	10.7311	0.2651	GridSearchCV LogReg pipeline with Cat+Num feat...

In [225…
```
y_pred_merged = gs.predict(X_test_merged)
```

In [226…
```
res_merged = gs.predict_proba(X_train_merged)
res_merged
```

Out[226…
```
array([[0.98358155, 0.01641845],
       [0.98011809, 0.01988191],
       [0.97328193, 0.02671807],
       ...,
       [0.96764864, 0.03235136],
       [0.99244162, 0.00755838],
       [0.96137727, 0.03862273]])
```

> Prediction probability of Class 0 and Class 1

Validation Accuracy

In [227…
```
print('Validation set accuracy score: ' + str(accuracy_score(y_test_merged,y_pre
```

```
Validation set accuracy score: 0.9198721421605808
```

Log Loss

In [228…
```
Image(url= "https://miro.medium.com/max/1192/1*wilGXrItaMAJmZNl6RJq9Q.png", widt
```

Out[228…

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

In [229…

```
log_loss(y_test_merged,y_pred_merged)
```

Out[229…
```
2.7675196811129577
```

ROC-AUC Curve

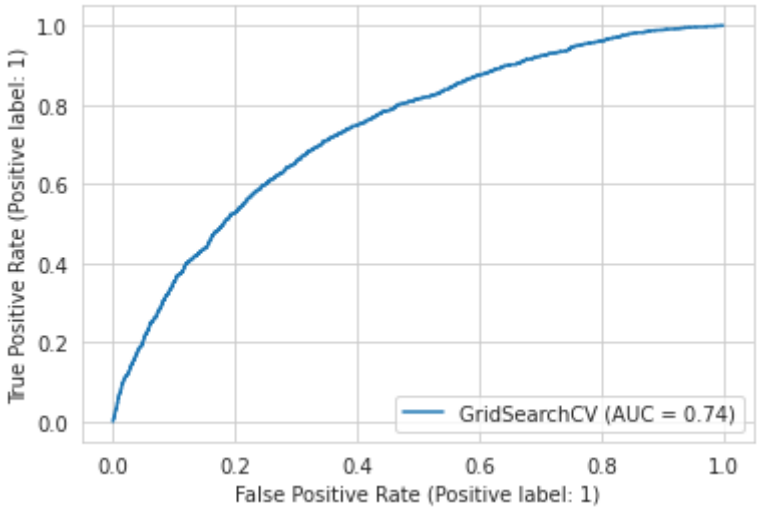
In [230…
```
roc_auc_score(y_test_merged, gs.predict_proba(X_test_merged)[:, 1])
```

Out[230…
```
0.738818586860982
```

Curve

In [231…
```
metrics.plot_roc_curve(gs, X_test_merged, y_test_merged)
```

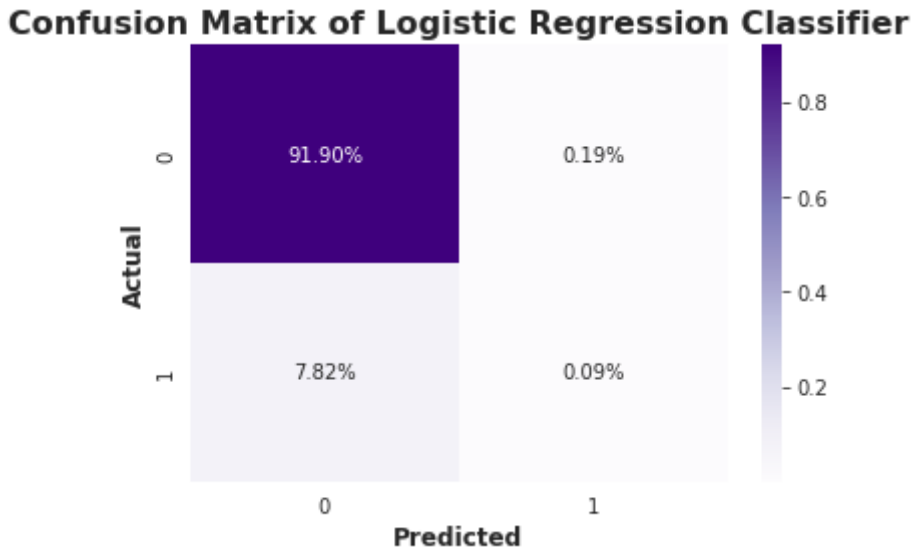
Out[231…
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f42740f2a00>
```



Confusion Matrix

In [232…
```
from sklearn.metrics import accuracy_score, recall_score, plot_roc_curve, confus

cf_matrix = confusion_matrix(y_test_merged, y_pred_merged)
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Purples')
plt.title('Confusion Matrix of Logistic Regression Classifier', fontweight='bold
plt.xlabel('Predicted', fontweight='bold', fontsize=12)
plt.ylabel('Actual', fontweight='bold', fontsize=12)
```

Out[232…
```
Text(33.0, 0.5, 'Actual')
```



Random Forest Classifier with GridSearchCV

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

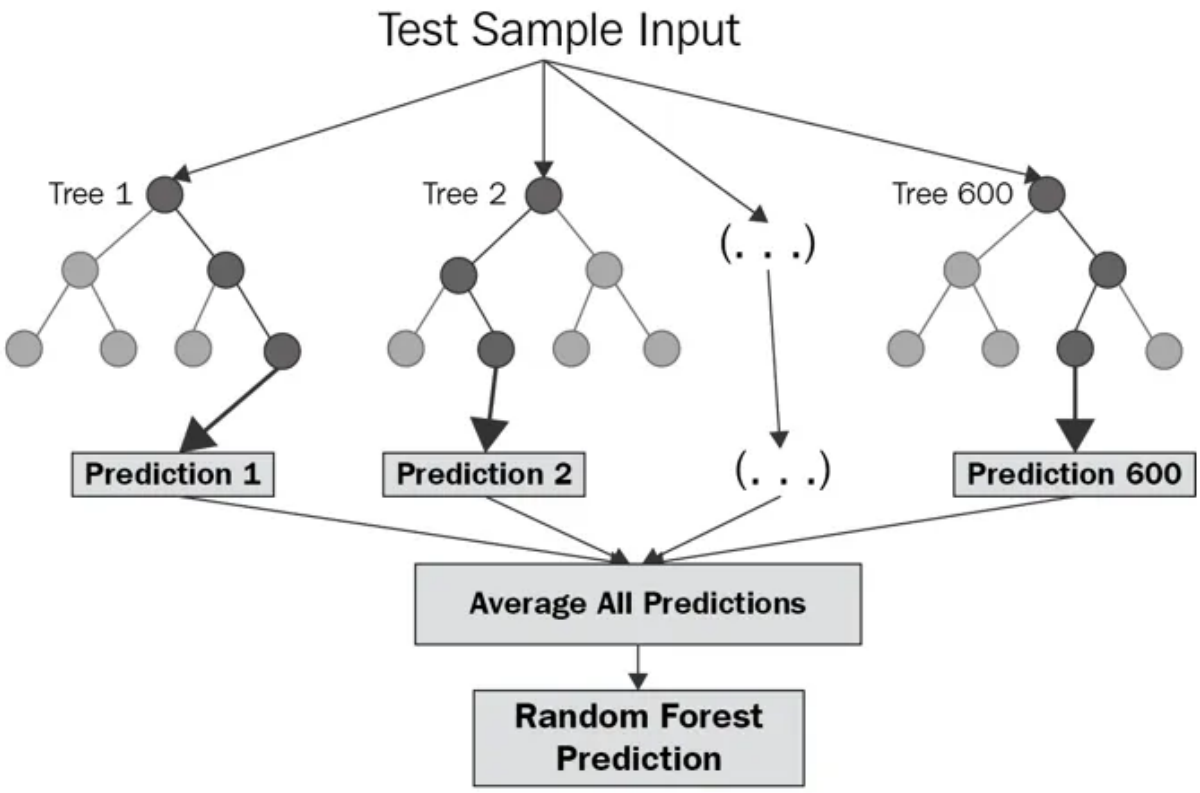
Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

In [233…

```
Image(url= "https://cdn.corporatefinanceinstitute.com/assets/random-forest.png",
```

Out[233…



In [234…
```
Image(url= "https://miro.medium.com/max/1400/1*cBaV_bLVW-gT6PkG5ve26A.png", widt
```

Out[234…

$$F(x) = \frac{1}{J}\sum_{j=1}^{J} c_{j_{full}} + \sum_{k=1}^{K}\left(\frac{1}{J}\sum_{j=1}^{J} contribution_j(x,k)\right)$$

In [235…
```
data_pipeline_merged = ColumnTransformer([
    ("num_pipeline", num_pipeline, numerical_df_merged),
    ("cat_pipeline", cat_pipeline, categorical_df_merged)])

model_rf = Pipeline(steps=[('data_pipeline_merged',data_pipeline_merged),
                           ('classifier', RandomForestClassifier())])
```

In [236…
```
param_grid = {
   'classifier__n_estimators': [100,200],
#     'classifier__max_depth':[4,5,6,7],
    'classifier__criterion':['gini','entropy']
}
gs_rf = GridSearchCV(model_rf,param_grid, cv=5, n_jobs = -1, verbose = 2, refit
start = time()

gs_rf.fit(X_train_merged,y_train_merged)

trainAcc  = gs_rf.score(X_train_merged, y_train_merged)
validAcc  = gs_rf.score(X_test_merged, y_test_merged)
```

```
start = time()
testAcc  = gs_rf.score(X_test_merged, y_test_merged)
test_time = np.round(time() - start, 4)

#del experimentLog
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc"
                                               "Train Time(s)",  "Test Time(s)",
experimentLog.loc[len(experimentLog)] =[f"GridSearchCV RandomForestClassifier",
                                        f"{trainAcc*100:8.2f}%", f"{validAcc*100
                                        train_time, test_time,
                                        "GridSearchCV RandomForestClassifier pip


experimentLog
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

Out [236…

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Desc
0	Baseline 1 LogReg	HCDR	91.91%	91.94%	91.94%	17.1005	0.4038	Baseline 1 l pipeline with Ca f
1	Baseline 1 RandomForest	HCDR	91.91%	91.95%	91.95%	10.7311	0.4899	Ba RandomForest p with Cat+
2	GridSearchCV LogReg	HCDR	91.79%	91.99%	91.99%	10.7311	0.2651	GridSearchCV l pipeline with Ca
3	GridSearchCV RandomForestClassifier	HCDR	100.00%	92.09%	92.09%	10.7311	1.2215	GridSe RandomForestCl pipel

In [237…

```
clf_pipe_merged_rf = make_pipeline(data_pipeline_merged, RandomForestClassifier(
# clf_pipe.fit(X_train, y_train)

# train_acc = clf_pipe.score(X_train, y_train)
# # validAcc  = clf_pipe.score(X_valid, y_valid)
# testAcc  = clf_pipe.score(X_test, y_test)

# print(train_acc,testAcc)

start = time()
clf_pipe_merged_rf.fit(X_train_merged, y_train_merged)
trainAcc  = clf_pipe_merged_rf.score(X_train_merged, y_train_merged)
validAcc  = clf_pipe_merged_rf.score(X_test_merged, y_test_merged)

start = time()
testAcc  = clf_pipe_merged_rf.score(X_test_merged, y_test_merged)
test_time = np.round(time() - start, 4)

#del experimentLog
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc"
                                               "Train Time(s)",  "Test Time(s)",
experimentLog.loc[len(experimentLog)] =[f"GridSearchCV RandomForestClassifier",
                                        f"{trainAcc*100:8.2f}%", f"{validAcc*100
                                        train_time, test_time,
```

```
                                                "GridSearchCV RandomForestClassifier pip

experimentLog
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 64 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  74 out of 100 | elapsed:    1.9s remaining:    0.7s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    2.1s finished
[Parallel(n_jobs=64)]: Using backend ThreadingBackend with 64 concurrent worker
s.
[Parallel(n_jobs=64)]: Done  74 out of 100 | elapsed:    0.2s remaining:    0.1s
[Parallel(n_jobs=64)]: Done 100 out of 100 | elapsed:    0.2s finished
[Parallel(n_jobs=64)]: Using backend ThreadingBackend with 64 concurrent worker
s.
[Parallel(n_jobs=64)]: Done  74 out of 100 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=64)]: Done 100 out of 100 | elapsed:    0.1s finished
[Parallel(n_jobs=64)]: Using backend ThreadingBackend with 64 concurrent worker
s.
[Parallel(n_jobs=64)]: Done  74 out of 100 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=64)]: Done 100 out of 100 | elapsed:    0.1s finished
```

Out[237…

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Desc
0	Baseline 1 LogReg	HCDR	91.91%	91.94%	91.94%	17.1005	0.4038	Baseline 1 l pipeline with Ca f
1	Baseline 1 RandomForest	HCDR	91.91%	91.95%	91.95%	10.7311	0.4899	Ba RandomForest p with Cat+
2	GridSearchCV LogReg	HCDR	91.79%	91.99%	91.99%	10.7311	0.2651	GridSearchCV l pipeline with Ca
3	GridSearchCV RandomForestClassifier	HCDR	100.00%	92.09%	92.09%	10.7311	1.2215	GridSe RandomForestCl pipel
4	GridSearchCV RandomForestClassifier	HCDR	100.00%	92.09%	92.09%	10.7311	0.2967	GridSe RandomForestCl pipel

In [238…

```
y_pred_merged_rf = gs_rf.predict(X_test_merged)
```

In [239…

```
res_merged_rf = gs_rf.predict_proba(X_train_merged)
res_merged_rf
```

Out[239…

```
array([[0.99 , 0.01 ],
       [0.99 , 0.01 ],
       [0.975, 0.025],
       ...,
       [0.97 , 0.03 ],
       [0.995, 0.005],
       [0.99 , 0.01 ]])
```

Validation Accuracy

In [240…
```
print('Validation set accuracy score: ' + str(accuracy_score(y_test_merged,y_pre
```

```
Validation set accuracy score: 0.9209015061220067
```

Log Loss

In [241…
```
log_loss(y_test_merged,y_pred_merged_rf)
```

Out[241…
```
2.7319651932262223
```

ROC-AUC Score

In [242…
```
roc_auc_score(y_test_merged, gs_rf.predict_proba(X_test_merged)[:, 1])
```

Out[242…
```
0.7153041501828019
```

ROC Curve

In [243…
```
metrics.plot_roc_curve(gs_rf, X_test_merged, y_test_merged)
```

Out[243…
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f42de792f70>
```

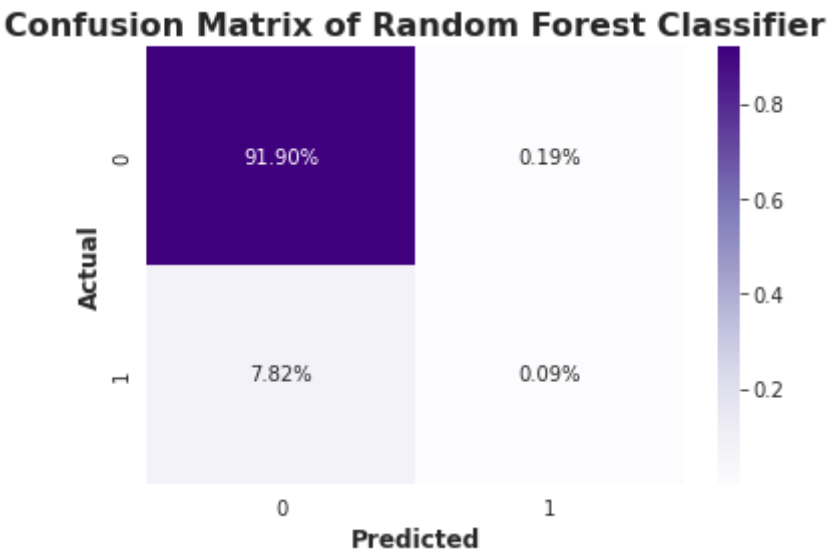
Confusion Matrix

In [244…
```
from sklearn.metrics import accuracy_score, recall_score, plot_roc_curve, confus

cf_matrix = confusion_matrix(y_test_merged, y_pred_merged)
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Purples')
plt.title('Confusion Matrix of Random Forest Classifier', fontweight='bold', fon
```

```
plt.xlabel('Predicted', fontweight='bold', fontsize=12)
plt.ylabel('Actual', fontweight='bold', fontsize=12)
```

Out [244…

```
Text(33.0, 0.5, 'Actual')
```



Preparing Test Data

In [245…

```
datasets['application_test'] = datasets['application_test'].drop(['REG_REGION_NO
'FLAG_DOCUMENT_2',
'FLAG_DOCUMENT_21',
'AMT_REQ_CREDIT_BUREAU_DAY',
'FLAG_EMAIL',
'AMT_REQ_CREDIT_BUREAU_QRT',
'FLAG_DOCUMENT_4',
'FLAG_DOCUMENT_17',
'FLAG_DOCUMENT_11',
'FLAG_DOCUMENT_9',
'FLAG_DOCUMENT_15',
'FLAG_DOCUMENT_18',
'FLAG_DOCUMENT_8','TOTALAREA_MODE','ELEVATORS_AVG','LIVINGAPARTMENTS_AVG',
                                                                    'LIVINGAREA_
                                                                    'LIVINGAPART
                                                ,'APARTMENTS_MEDI','ELEVATORS_MEDI'
'NONLIVINGAREA_MEDI','NONLIVINGAPARTMENTS_MEDI','FLOORSMIN_MEDI',
                                                                    'FLOORSMAX_M
                                                                    'BASEMENTARE
                                            'YEARS_BUILD_MODE','YEARS_BEGINEXPLU
                                                                    'ENTRANCES_M
                                                                    'REG_CITY_NO
                                                                  'YEARS_BUILD_
                                                                       'NONLIVIN
                                                                    'LIVE_CITY_N
                                                                  'CNT_CHILDREN

'AMT_REQ_CREDIT_BUREAU_HOUR',
'AMT_REQ_CREDIT_BUREAU_WEEK',
'FLAG_MOBIL',
'FLAG_CONT_MOBILE',
'FLAG_DOCUMENT_20',
'FLAG_DOCUMENT_5',
```

```
'FLAG_DOCUMENT_12',
'FLAG_DOCUMENT_19',
'FLAG_DOCUMENT_10',
'FLAG_DOCUMENT_7'],axis=1)
```

Missing Value Percent in Test Data

In [237…

```
percent = (datasets["application_test"].isnull().sum()/datasets["application_tes
sum_missing = datasets["application_test"].isna().sum().sort_values(ascending =
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys
missing_application_train_data.head(20)
```

Out[237…

	Percent	Test Missing Count
FONDKAPREMONT_MODE	67.28	32797
OWN_CAR_AGE	66.29	32312
LANDAREA_AVG	57.96	28254
BASEMENTAREA_AVG	56.71	27641
WALLSMATERIAL_MODE	49.02	23893
APARTMENTS_AVG	49.01	23887
HOUSETYPE_MODE	48.46	23619
ENTRANCES_AVG	48.37	23579
FLOORSMAX_AVG	47.84	23321
YEARS_BEGINEXPLUATATION_AVG	46.89	22856
EMERGENCYSTATE_MODE	45.56	22209
EXT_SOURCE_1	42.12	20532
OCCUPATION_TYPE	32.01	15605
EXT_SOURCE_3	17.78	8668
AMT_REQ_CREDIT_BUREAU_MON	12.41	6049
AMT_REQ_CREDIT_BUREAU_YEAR	12.41	6049
NAME_TYPE_SUITE	1.87	911
OBS_60_CNT_SOCIAL_CIRCLE	0.06	29
DEF_60_CNT_SOCIAL_CIRCLE	0.06	29
AMT_ANNUITY	0.05	24

Test Data Exploration

In [238…

```
datasets['application_test'].shape
```

Out[238…

```
(48744, 51)
```

In [239…
```
datasets['application_test'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Data columns (total 51 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   SK_ID_CURR                   48744 non-null  int64  
 1   NAME_CONTRACT_TYPE           48744 non-null  object 
 2   CODE_GENDER                  48744 non-null  object 
 3   FLAG_OWN_CAR                 48744 non-null  object 
 4   FLAG_OWN_REALTY              48744 non-null  object 
 5   AMT_INCOME_TOTAL             48744 non-null  float64
 6   AMT_CREDIT                   48744 non-null  float64
 7   AMT_ANNUITY                  48720 non-null  float64
 8   NAME_TYPE_SUITE              47833 non-null  object 
 9   NAME_INCOME_TYPE             48744 non-null  object 
 10  NAME_EDUCATION_TYPE          48744 non-null  object 
 11  NAME_FAMILY_STATUS           48744 non-null  object 
 12  NAME_HOUSING_TYPE            48744 non-null  object 
 13  REGION_POPULATION_RELATIVE   48744 non-null  float64
 14  DAYS_BIRTH                   48744 non-null  int64  
 15  DAYS_EMPLOYED                48744 non-null  int64  
 16  DAYS_REGISTRATION            48744 non-null  float64
 17  DAYS_ID_PUBLISH              48744 non-null  int64  
 18  OWN_CAR_AGE                  16432 non-null  float64
 19  FLAG_EMP_PHONE               48744 non-null  int64  
 20  FLAG_WORK_PHONE              48744 non-null  int64  
 21  FLAG_PHONE                   48744 non-null  int64  
 22  OCCUPATION_TYPE              33139 non-null  object 
 23  CNT_FAM_MEMBERS              48744 non-null  float64
 24  REGION_RATING_CLIENT_W_CITY  48744 non-null  int64  
 25  WEEKDAY_APPR_PROCESS_START   48744 non-null  object 
 26  HOUR_APPR_PROCESS_START      48744 non-null  int64  
 27  REG_CITY_NOT_WORK_CITY       48744 non-null  int64  
 28  ORGANIZATION_TYPE            48744 non-null  object 
 29  EXT_SOURCE_1                 28212 non-null  float64
 30  EXT_SOURCE_2                 48736 non-null  float64
 31  EXT_SOURCE_3                 40076 non-null  float64
 32  APARTMENTS_AVG               24857 non-null  float64
 33  BASEMENTAREA_AVG             21103 non-null  float64
 34  YEARS_BEGINEXPLUATATION_AVG  25888 non-null  float64
 35  ENTRANCES_AVG                25165 non-null  float64
 36  FLOORSMAX_AVG                25423 non-null  float64
 37  LANDAREA_AVG                 20490 non-null  float64
 38  FONDKAPREMONT_MODE           15947 non-null  object 
 39  HOUSETYPE_MODE               25125 non-null  object 
 40  WALLSMATERIAL_MODE           24851 non-null  object 
 41  EMERGENCYSTATE_MODE          26535 non-null  object 
 42  OBS_60_CNT_SOCIAL_CIRCLE     48715 non-null  float64
 43  DEF_60_CNT_SOCIAL_CIRCLE     48715 non-null  float64
 44  FLAG_DOCUMENT_3              48744 non-null  int64  
 45  FLAG_DOCUMENT_6              48744 non-null  int64  
 46  FLAG_DOCUMENT_13             48744 non-null  int64  
 47  FLAG_DOCUMENT_14             48744 non-null  int64  
 48  FLAG_DOCUMENT_16             48744 non-null  int64  
 49  AMT_REQ_CREDIT_BUREAU_MON    42695 non-null  float64
 50  AMT_REQ_CREDIT_BUREAU_YEAR   42695 non-null  float64
dtypes: float64(20), int64(15), object(16)
memory usage: 19.0+ MB
```

In [240…
```
datasets['application_test'].describe()
```

Out[240…

	SK_ID_CURR	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	REGION_POPULATION
count	48744.000000	4.874400e+04	4.874400e+04	48720.000000	48
mean	277796.676350	1.784318e+05	5.167404e+05	29426.240209	
std	103169.547296	1.015226e+05	3.653970e+05	16016.368315	
min	100001.000000	2.694150e+04	4.500000e+04	2295.000000	
25%	188557.750000	1.125000e+05	2.606400e+05	17973.000000	
50%	277549.000000	1.575000e+05	4.500000e+05	26199.000000	
75%	367555.500000	2.250000e+05	6.750000e+05	37390.500000	
max	456250.000000	4.410000e+06	2.245500e+06	180576.000000	

8 rows × 35 columns

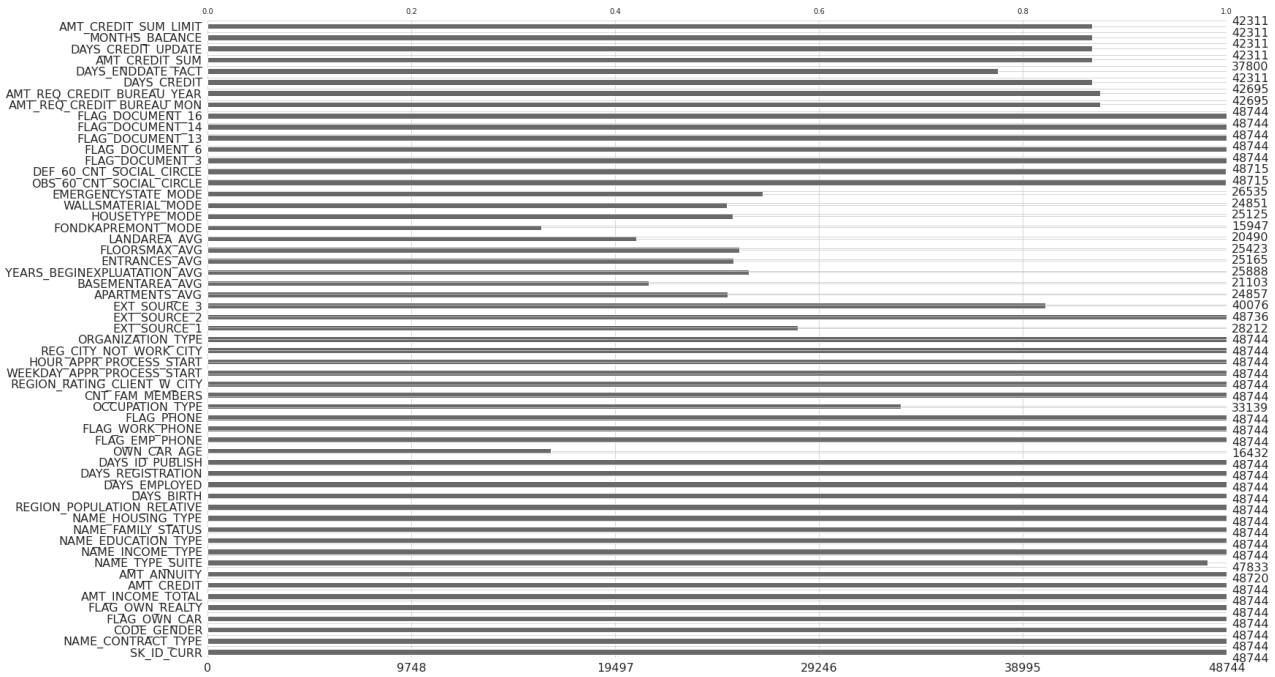
In [241…
```
df_finalm_bureau_test =  pd.merge(left = datasets['application_test'], right = d
                                  how='left', left_on='SK_ID_CURR', right_on='SK_
```

In [242…
```
df_finalm_bureau_test.shape
```

Out[242…
```
(48744, 57)
```

In [243…
```
msno.bar(df_finalm_bureau_test)
```

Out[243…
```
<AxesSubplot:>
```



In [244…
```
df_final_test = pd.merge(left=df_finalm_bureau_test, right=df_prev_ccbalinst, ho
```

```
                                left_on='SK_ID_CURR', right_on='SK_ID_CURR')
```

In [245…
```
df_final_test.shape
```

Out[245…
```
(48797, 70)
```

In [247…
```
df_final_test['DAYS_EMPLOYED_PCT'] = df_final_test['DAYS_EMPLOYED'] / df_final_t
df_final_test['CREDIT_INCOME_PCT'] = df_final_test['AMT_CREDIT'] / df_final_test
df_final_test['ANNUITY_INCOME_PCT'] = df_final_test['AMT_ANNUITY_x'] / df_final_
df_final_test['CREDIT_TERM_PCT'] = df_final_test['AMT_ANNUITY_x'] / df_final_tes
df_final_test['AMT_BALANCE_PCT'] = df_final_test['AMT_BALANCE'] / df_final_test[
df_final_test['AVG_INCOME_EXT_PCT'] = (df_final_test['EXT_SOURCE_1']+df_final_te
                                       +df_final_test['EXT_SOURCE_3'])/3
df_final_test['AVG_TOTALINCOME_PCT'] = df_final_test['AMT_INCOME_TOTAL']/df_fina
```

In [246…
```
df_final_merged.shape
```

Out[246…
```
(92288, 78)
```

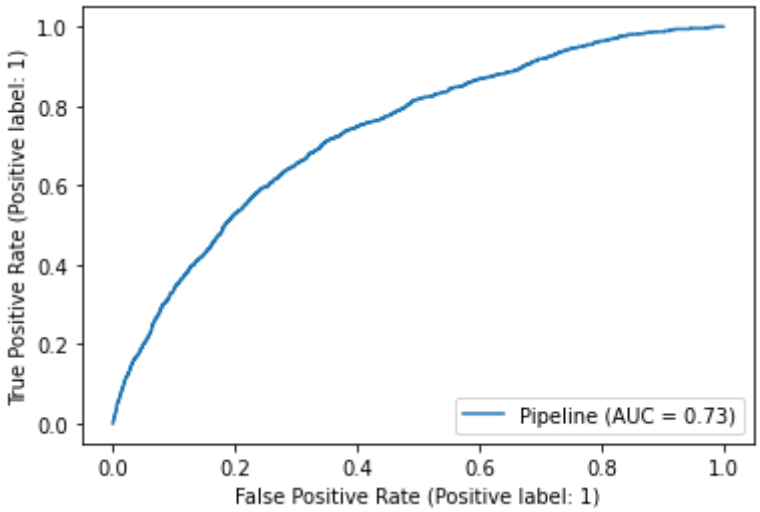
In [248…
```
df_final_test.shape
```

Out[248…
```
(48797, 77)
```

Datasets Questions

In [268…
```
metrics.plot_roc_curve(clf_pipe_merged, X_test_merged, y_test_merged)
```

Out[268…
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7fdf2f921430>
```



In [294…
```
datasets.keys()
```

Out[294…
```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance',
'credit_card_balance', 'installments_payments', 'previous_application', 'POS_CAS
H_balance'])
```

In [295…
```
datasets["application_train"].head()
```

Out[295…

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
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	

5 rows × 52 columns

In [296…
```
len(datasets["application_train"]["SK_ID_CURR"].unique()) == datasets["applicati
```

Out[296…
```
True
```

In [297…
```
np.intersect1d(datasets["application_train"]["SK_ID_CURR"], datasets["applicatio
```

Out[297…
```
array([], dtype=int64)
```

In [298…
```
datasets["application_test"].shape
```

Out[298…
```
(48744, 51)
```

In [299…
```
datasets["application_train"].shape
```

Out[299…
```
(307511, 52)
```

Kaggle Submission Phase1

In [302…
```
pred= clf_pipe.predict(datasets["application_test"])
pred_df = pd.DataFrame(pred)
pred_df.shape
```

Out[302…
```
(48744, 1)
```

In [303…
```
pred_df.value_counts()
```

Out[303…
```
0    48649
1       95
dtype: int64
```

In [304…
```
pred_proba = clf_pipe.predict_proba(datasets["application_test"])
pred_proba_df = pd.DataFrame(pred_proba)
```

pred_proba_df

In [305…
```
pred_proba_df
```

Out[305…

	0	1
0	0.939066	0.060934
1	0.766156	0.233844
2	0.947208	0.052792
3	0.950071	0.049929
4	0.857568	0.142432
...	...	...
48739	0.970325	0.029675
48740	0.946596	0.053404
48741	0.941403	0.058597
48742	0.928029	0.071971
48743	0.904105	0.095895

48744 rows × 2 columns

In [306…
```
class_1_proba = pred_proba_df[[1]]
```

In [307…
```
datasets['application_test'].head()
```

Out[307…

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY
0	100001	Cash loans	F	N	Y
1	100005	Cash loans	M	N	Y
2	100013	Cash loans	M	Y	Y
3	100028	Cash loans	F	N	Y
4	100038	Cash loans	M	Y	N

5 rows × 51 columns

In [308…
```
final = pd.DataFrame()
```

In [309…
```
final['SK_ID_CURR'] = datasets['application_test']['SK_ID_CURR']
final['TARGET'] = class_1_proba[1]
```

In [310…
```

```

Out[310…

	SK_ID_CURR	TARGET
0	100001	0.060934
1	100005	0.233844
2	100013	0.052792
3	100028	0.049929
4	100038	0.142432
...	...	...
48739	456221	0.029675
48740	456222	0.053404
48741	456223	0.058597
48742	456224	0.071971
48743	456250	0.095895

48744 rows × 2 columns

In [311…
```
final = final.set_index('SK_ID_CURR')
```

In [312…
```
final.to_csv('submission.csv')
```

Kaggle Submission Phase2

In [271…
```
pred2= gs.predict(df_final_test)
pred_df2 = pd.DataFrame(pred2)
pred_df2.shape
```

Out[271…
```
(48797, 1)
```

In [272…
```
pred_df2.value_counts()
```

Out[272…
```
0    48740
1       57
dtype: int64
```

In [259…
```
pred_proba_part2 = gs_rf.predict_proba(df_final_test)
pred_proba_df = pd.DataFrame(pred_proba_part2)
```

In [273…
```
pred_proba_part3 = gs.predict_proba(df_final_test)
pred_proba_df_log = pd.DataFrame(pred_proba_part3)
```

In [260…
```
pred_proba_df
```

Out[260…

	0	1
0	0.95	0.05
1	0.90	0.10
2	0.95	0.05
3	0.95	0.05
4	0.84	0.16
…	…	…
48792	0.96	0.04
48793	0.86	0.14
48794	0.93	0.07
48795	0.94	0.06
48796	0.77	0.23

48797 rows × 2 columns

In [274…
```
pred_proba_df_log
```

Out[274…

	0	1
0	0.937469	0.062531
1	0.826831	0.173169
2	0.960347	0.039653
3	0.965731	0.034269
4	0.805910	0.194090
…	…	…
48792	0.973858	0.026142
48793	0.941609	0.058391
48794	0.967976	0.032024
48795	0.942264	0.057736
48796	0.878175	0.121825

48797 rows × 2 columns

In [261…
```
class_1_proba_df = pred_proba_df[[1]]
```

In [275…
```
class_1_proba_df_log = pred_proba_df_log[[1]]
```

In [276…
```
df_final_test.head()
```

Out[276…

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY
0	100001	Cash loans	F	N	Y
1	100005	Cash loans	M	N	Y
2	100013	Cash loans	M	Y	Y
3	100028	Cash loans	F	N	Y
4	100038	Cash loans	M	Y	N

5 rows × 77 columns

In [263…
```
finalp2 = pd.DataFrame()
```

In [277…
```
finalp3 = pd.DataFrame()
```

In [264…
```
finalp2['SK_ID_CURR'] = df_final_test['SK_ID_CURR']
finalp2['TARGET'] = class_1_proba_df[1]
```

In [278…
```
finalp3['SK_ID_CURR'] = df_final_test['SK_ID_CURR']
finalp3['TARGET'] = class_1_proba_df_log[1]
```

In [265…
```
finalp2
```

Out[265…

	SK_ID_CURR	TARGET
0	100001	0.05
1	100005	0.10
2	100013	0.05
3	100028	0.05
4	100038	0.16
…	…	…
48792	456221	0.04
48793	456222	0.14
48794	456223	0.07
48795	456224	0.06
48796	456250	0.23

48797 rows × 2 columns

In [279…
```
finalp3
```

Out[279…

	SK_ID_CURR	TARGET

	SK_ID_CURR	TARGET
0	100001	0.062531
1	100005	0.173169
2	100013	0.039653
3	100028	0.034269
4	100038	0.194090
...	...	...
48792	456221	0.026142
48793	456222	0.058391
48794	456223	0.032024
48795	456224	0.057736
48796	456250	0.121825

48797 rows × 2 columns

In [266…

```
finalp2 = finalp2.drop_duplicates(subset='SK_ID_CURR', keep="first")
finalp2
```

Out[266…

	SK_ID_CURR	TARGET
0	100001	0.05
1	100005	0.10
2	100013	0.05
3	100028	0.05
4	100038	0.16
...	...	...
48792	456221	0.04
48793	456222	0.14
48794	456223	0.07
48795	456224	0.06
48796	456250	0.23

48744 rows × 2 columns

In [280…

```
finalp3 = finalp3.drop_duplicates(subset='SK_ID_CURR', keep="first")
finalp3
```

Out[280…

	SK_ID_CURR	TARGET
0	100001	0.062531
1	100005	0.173169

	SK_ID_CURR	TARGET
2	100013	0.039653
3	100028	0.034269
4	100038	0.194090
...	...	...
48792	456221	0.026142
48793	456222	0.058391
48794	456223	0.032024
48795	456224	0.057736
48796	456250	0.121825

48744 rows × 2 columns

In [267…
```
finalp2 = finalp2.set_index('SK_ID_CURR')
```

In [281…
```
finalp3 = finalp3.set_index('SK_ID_CURR')
```

In [268…
```
finalp2.to_csv('submission.csv')
```

In [282…
```
finalp3.to_csv('submission.csv')
```

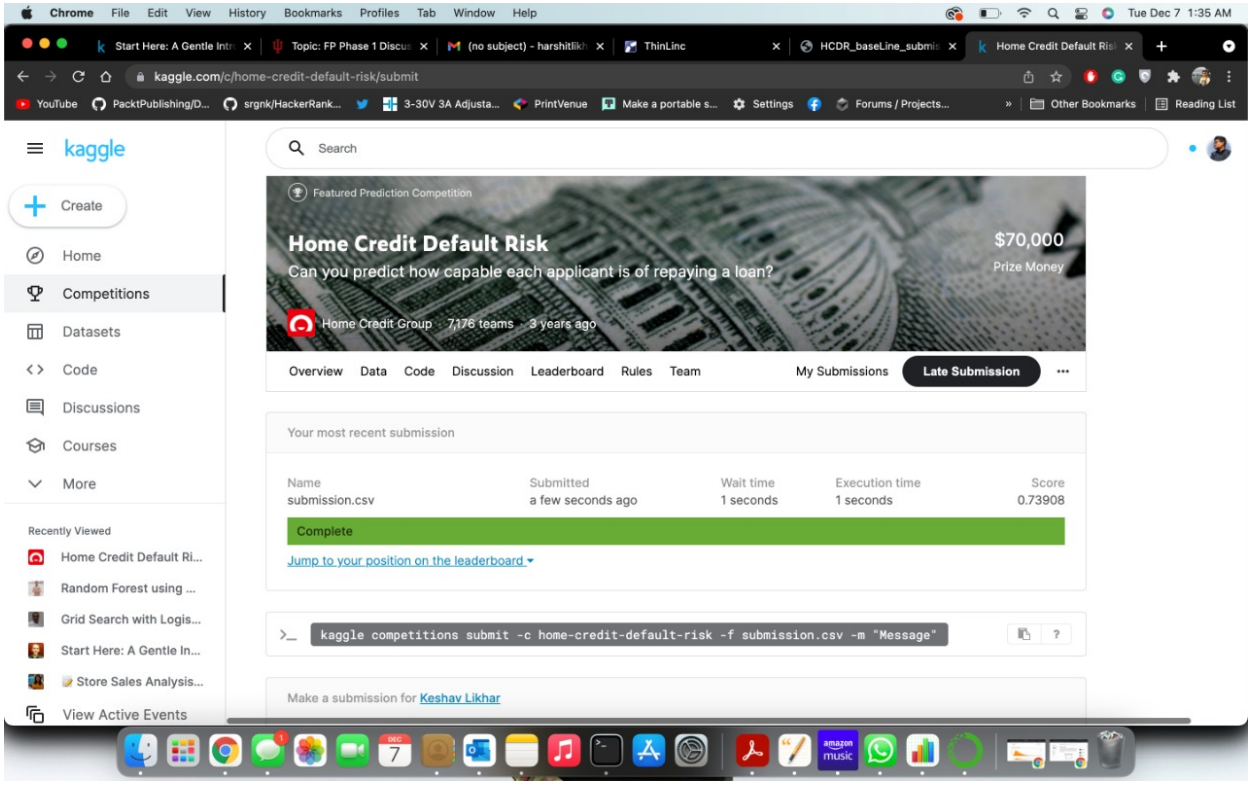
Kaggle submission via the command line API

In [284…
```
! kaggle competitions submit -c home-credit-default-risk -f submission.csv -m "b
```

```
/bin/bash: kaggle: command not found
```

report submission

Click on this link



Write Up Phase 1

Phase 1 - Final Project HCDR EDA + Baseline

Home Credit Default Risk Kaggle Competition

PROJECT MEMBERS:

ADITI MULYE - adimulye@iu.edu

KESHAV LIKHAR - klikhar@iu.edu

NIKUNJ MALPANI - nmalpani@iu.edu

PRASHASTI KARLEKAR - prkarl@iu.edu

PROJECT ABSTRACT

The course project is based on the Home Credit Default Risk (HCDR) (https://www.kaggle.com/c/home-credit-default-risk/). The objective of Home Credit Default Risk project is to correctly offer loans to individuals who can pay back and turn away those who cannot. In other words, the goal is to predict whether or not a client will repay a loan. In this phase of the project, we will perform analysis and modelling of the Home Credit default risk data currently hosted on Kaggle. The objective of this phase of the project is to perform exploratory

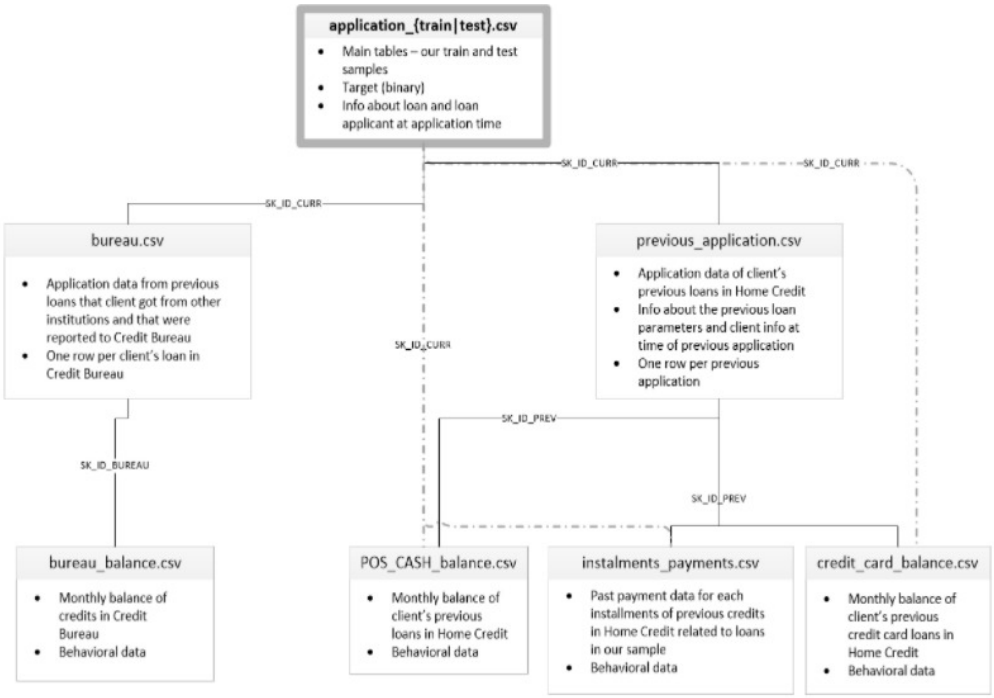
data analysis on the data, which includes describing the data, calculating the summary statistics on the data to summarize its main characteristics, visualizing the results, finding missing items count for all the input variables , etc. This step tells us what the data can tell us before we start modeling the data and to perform initial investigations on data so as to discover patterns, to spot irregularities and biases, and to check assumptions with help of summary statistics. The EDA process also involved dropping the columns with the most missing value counts. Subsequently, we built correlation matrix to find and remove highly correlated variables which would render our model inefficient. To avoid any inaccuracy, we Split the dataframe into train and test subsets. Next, we create pipelines to separately impute numerical and categorical values.We use one-hot encoding to transform the categorical variable values. Finally, we have used Logistic Regression and Random Forest to predict our target variable according to our input variables.

PROJECT DESCRIPTION

1. Data description The home-credit data is currently available on Kaggle for predicting whether or not a client will repay a loan or have difficulty, which is a critical business need. In this dataset we have a total of 122 columns. There are a lot of columns which have missing values in the dataset. As we can already see that there are so many missing values in the top 10 rows itself. We'll have to figure out a way to deal with this! There are 7 different sources of data:

- application_train/application_test: the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating 0: the loan was repaid or 1: the loan was not repaid.
- bureau: data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- bureau_balance: monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length. previous_application: previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- POS_CASH_BALANCE: monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- installments_payment: payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment. This diagram shows how

all of the data is related:

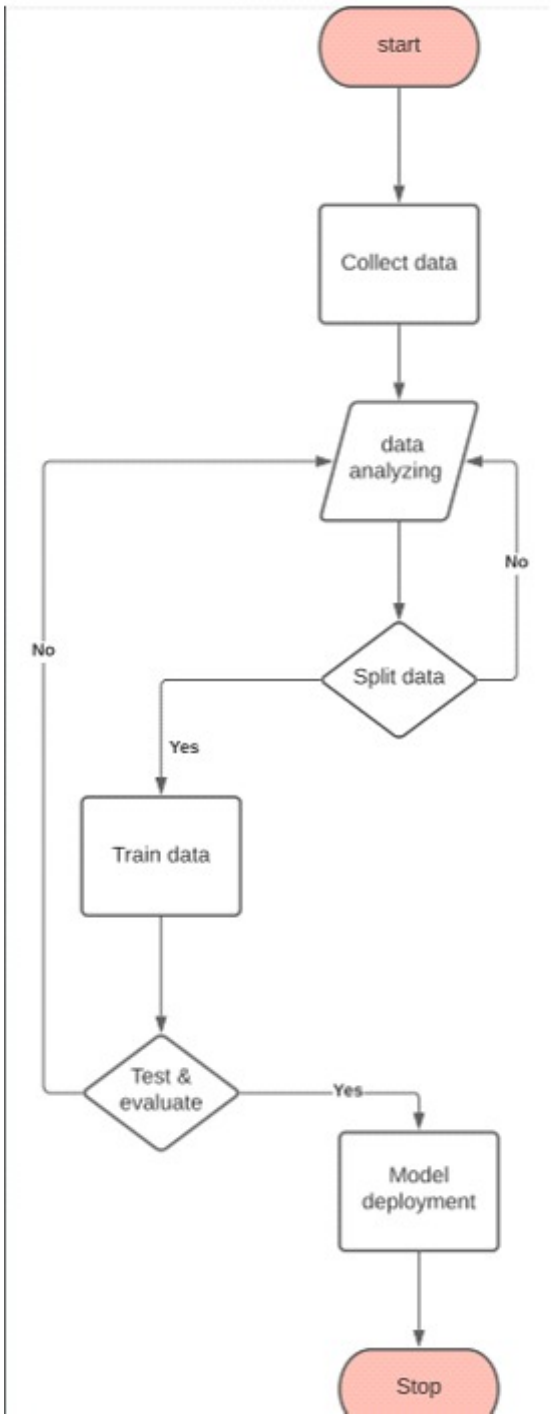


Task to be tackled

The task at hand for this phase of the project is to

- Explore the data
- Figure out and mitigate possible errors in the data that could render our model inaccurate
- Find missing values and highly correlated features
- Standardize the data
- Impute the data
- Split the data into training and test subsets
- Handling the categorical and numerical variables separately
- Build baseline models using Logistic Regression and Random Forest
- Calculate and validate the results

Provide diagrams to aid understanding the workflow



EXPLORATORY DATA ANALYSIS

Before deriving insights from the data, we need to know the data which gives a better idea of the problem at hand and the irregularities in the data are exposed with further analysis which need to be corrected before building the model. Here, we look at application_train.csv dataset.

Steps performed in this phase of project:

- Read and examine the dataset -
- In this step, firstly we import all the packages and modules. We import our data and load it in a dataframe.

- Determine the shape of the data- It is important to know the dimensionality of the dataset before processing the data. We use shape property of the dataframe to determine the number of columns and rows of our dataset. datasets['application_train'].shape

```
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
```

Identify the types of data available - Before proceeding for the further analysis of the data, we determine the datatypes of our variables. This uncovers several issues that can be encountered later, for example not converting the datatypes of the variables which seem to be correct but essentially need to be represented as another datatype.
datasets['application_train'].info(verbose=True) info() prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.

[159]:
```
datasets["application_train"].info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 122 columns):
 #    Column                       Dtype
---   ------                       -----
 0    SK_ID_CURR                   int64
 1    TARGET                       int64
 2    NAME_CONTRACT_TYPE           object
 3    CODE_GENDER                  object
 4    FLAG_OWN_CAR                 object
 5    FLAG_OWN_REALTY              object
 6    CNT_CHILDREN                 int64
 7    AMT_INCOME_TOTAL             float64
 8    AMT_CREDIT                   float64
 9    AMT_ANNUITY                  float64
 10   AMT_GOODS_PRICE              float64
 11   NAME_TYPE_SUITE              object
 12   NAME_INCOME_TYPE             object
 13   NAME_EDUCATION_TYPE          object
 14   NAME_FAMILY_STATUS           object
 15   NAME_HOUSING_TYPE            object
 16   REGION_POPULATION_RELATIVE   float64
 17   DAYS_BIRTH                   int64
 18   DAYS_EMPLOYED                int64
 19   DAYS_REGISTRATION            float64
 20   DAYS_ID_PUBLISH              int64
 21   OWN_CAR_AGE                  float64
 22   FLAG_MOBIL                   int64
 23   FLAG_EMP_PHONE               int64
 24   FLAG_WORK_PHONE              int64
 25   FLAG_CONT_MOBILE             int64
```

Evaluate basic statistical information about the data To get an idea of the average value of the data in the data set, we measure the central tendencies of our dataset. datasets["application_train"].describe(include='all') describe() prints descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution.

```
[164]: datasets["application_train"].describe(include='all') #look at all categorical and numerical
```

[164]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	...	FLAG_DOCUMENT
count	307511.000000	307511.000000	307511	307511	307511	307511	307511.000000	3.075110e+05	3.075110e+05	307499.000000	...	307511.00
unique	NaN	NaN	2	3	2	2	NaN	NaN	NaN	NaN	...	
top	NaN	NaN	Cash loans	F	N	Y	NaN	NaN	NaN	NaN	...	
freq	NaN	NaN	278232	202448	202924	213312	NaN	NaN	NaN	NaN	...	
mean	278180.518577	0.080729	NaN	NaN	NaN	NaN	0.417052	1.687979e+05	5.990260e+05	27108.573909	...	0.00
std	102790.175348	0.272419	NaN	NaN	NaN	NaN	0.722121	2.371231e+05	4.024908e+05	14493.737315	...	0.08
min	100002.000000	0.000000	NaN	NaN	NaN	NaN	0.000000	2.565000e+04	4.500000e+04	1615.500000	...	0.00
25%	189145.500000	0.000000	NaN	NaN	NaN	NaN	0.000000	1.125000e+05	2.700000e+05	16524.000000	...	0.00
50%	278202.000000	0.000000	NaN	NaN	NaN	NaN	0.000000	1.471500e+05	5.135310e+05	24903.000000	...	0.00
75%	367142.500000	0.000000	NaN	NaN	NaN	NaN	1.000000	2.025000e+05	8.086500e+05	34596.000000	...	0.00
max	456255.000000	1.000000	NaN	NaN	NaN	NaN	19.000000	1.170000e+08	4.050000e+06	258025.500000	...	1.00

11 rows × 122 columns

Finding number of numerical variables and categorical variables It is important to deal with numerical and categorical variables differently. Categorical features have a lot to say about the dataset thus they should be converted to numerical to make it into a machine-readable format.

6.3. Seperating numerical and categorical variables to fit in Pipeline

```
integer_df= X.select_dtypes(include='int64')
float_df = X.select_dtypes(include='float64')
numerical_df = list(pd.concat([integer_df,float_df], axis=1))
categorical_df = list(X.select_dtypes(include='object'))
print(categorical_df)
print('                              ')
print(numerical_df)
```

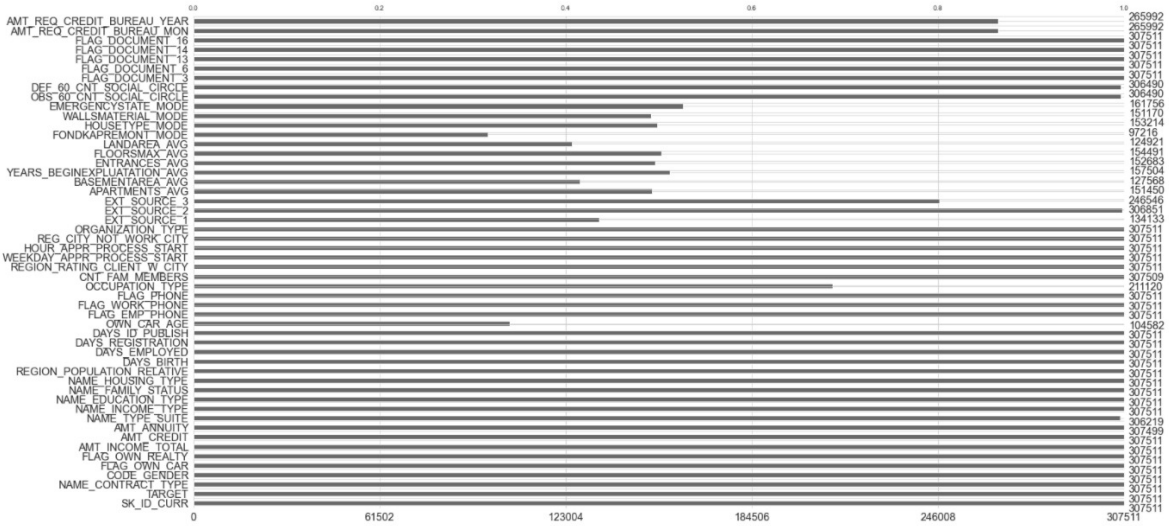
```
['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME
_HOUSING_TYPE', 'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYPE', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTAT
E_MODE']

['SK_ID_CURR', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_ID_PUBLISH', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_PHONE', 'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_PROC
ESS_START', 'REG_CITY_NOT_WORK_CITY', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_16', 'AMT_INCOME_TOTAL', 'AMT_C
REDIT', 'AMT_ANNUITY', 'REGION_POPULATION_RELATIVE', 'DAYS_REGISTRATION', 'OWN_CAR_AGE', 'CNT_FAM_MEMBERS', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APART
MENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'LANDAREA_AVG', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CI
RCLE', 'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_YEAR']
```

Check and determine missing data

Before we can use data with missing data fields, we need to transform these fields to be used for analysis and modelling. It is very important to handle missing data either by deleting or through imputation (handling the missing values with some estimation).

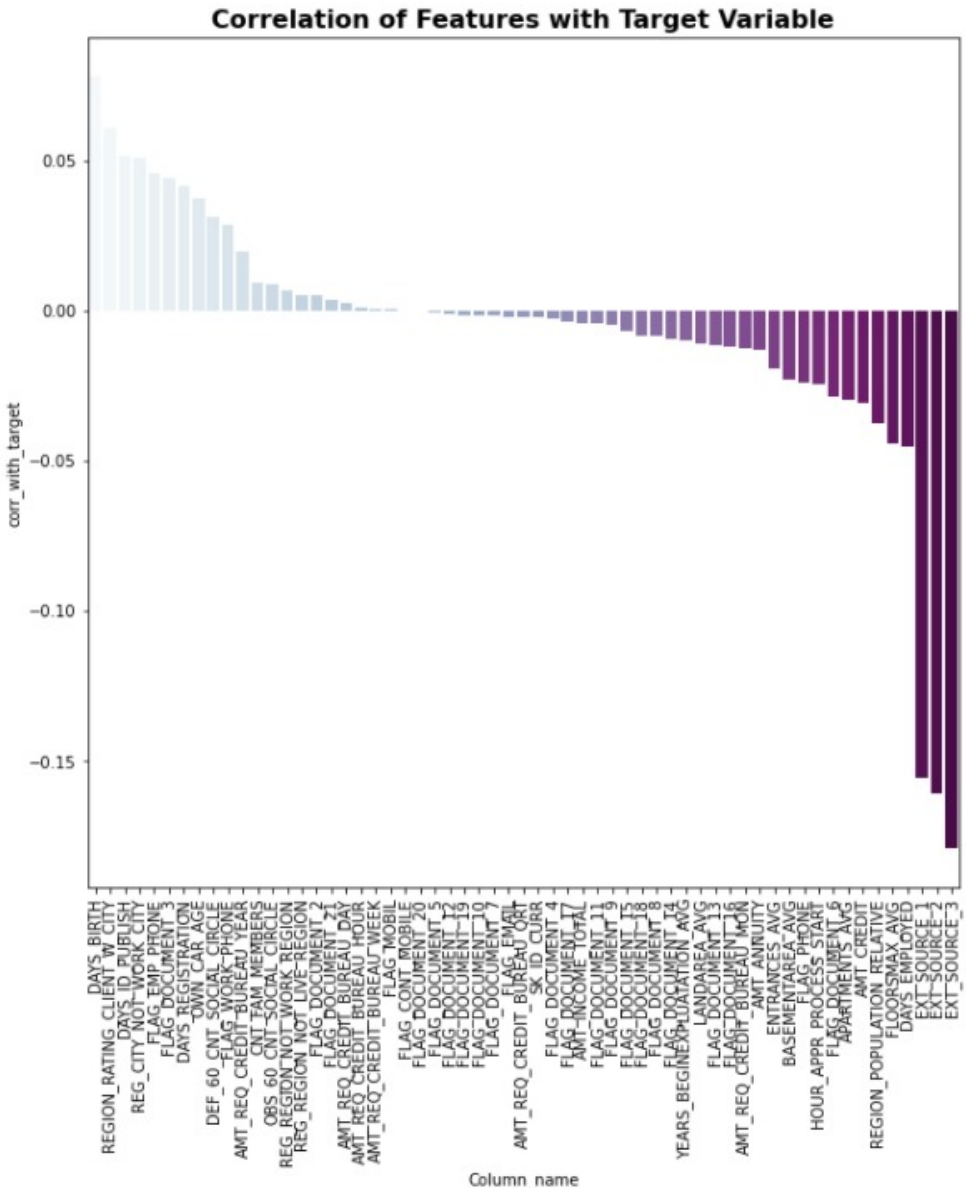
	Percent	Train Missing Count
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865
COMMONAREA_MODE	69.87	214865
NONLIVINGAPARTMENTS_MODE	69.43	213514
NONLIVINGAPARTMENTS_AVG	69.43	213514
NONLIVINGAPARTMENTS_MEDI	69.43	213514
FONDKAPREMONT_MODE	68.39	210295
LIVINGAPARTMENTS_MODE	68.35	210199
LIVINGAPARTMENTS_AVG	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

We determine the number of missing data in every column and sort the values in descending order according to the missing count to determine which features have the most missing values. We transform our dataframe to include only the columns which have more than 50 percent non-missing values. Next up, we impute the remaining missing data with median for numerical variables and the most frequent variable for the categorical variables.

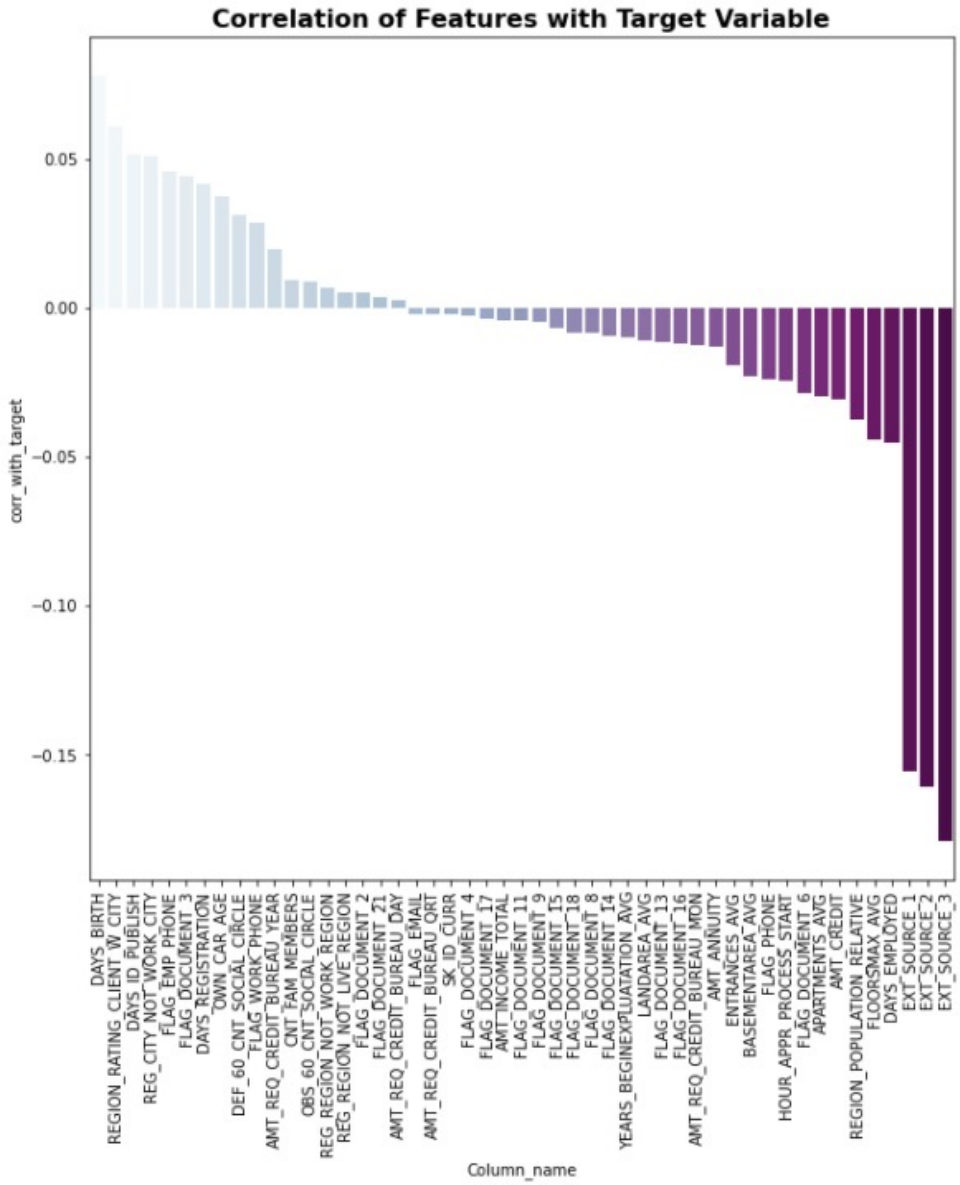


Visualization through correlation matrix

A Correlation Matrix is used to examine the relationship between multiple variables at the same time. When we do this calculation we get a table containing the correlation coefficients between each variable and the others. Now, the coefficient show us both the strength of the relationship and its direction (positive or negative correlations). In Python, a correlation matrix can be created using the Python packages Pandas and NumPy, for instance. If we have a big data set, and we have an intention to explore patterns. For use in other statistical methods. For instance, correlation matrices can be used as data when conducting exploratory factor analysis, confirmatory factor analysis, structural equation models. Correlation matrices can also be used as a diagnostic when checking assumptions for e.g. regression analysis.



Above, we have plotted the graph of correlation of all the features with the target variable. As it can be seen above, there are approximately 30% features which are not at all correlated with the target variable. We can drop them as we do not believe that they will influence our predictions of the target variable! Next, we drop the irrelevant variables which are highly correlated with one another.



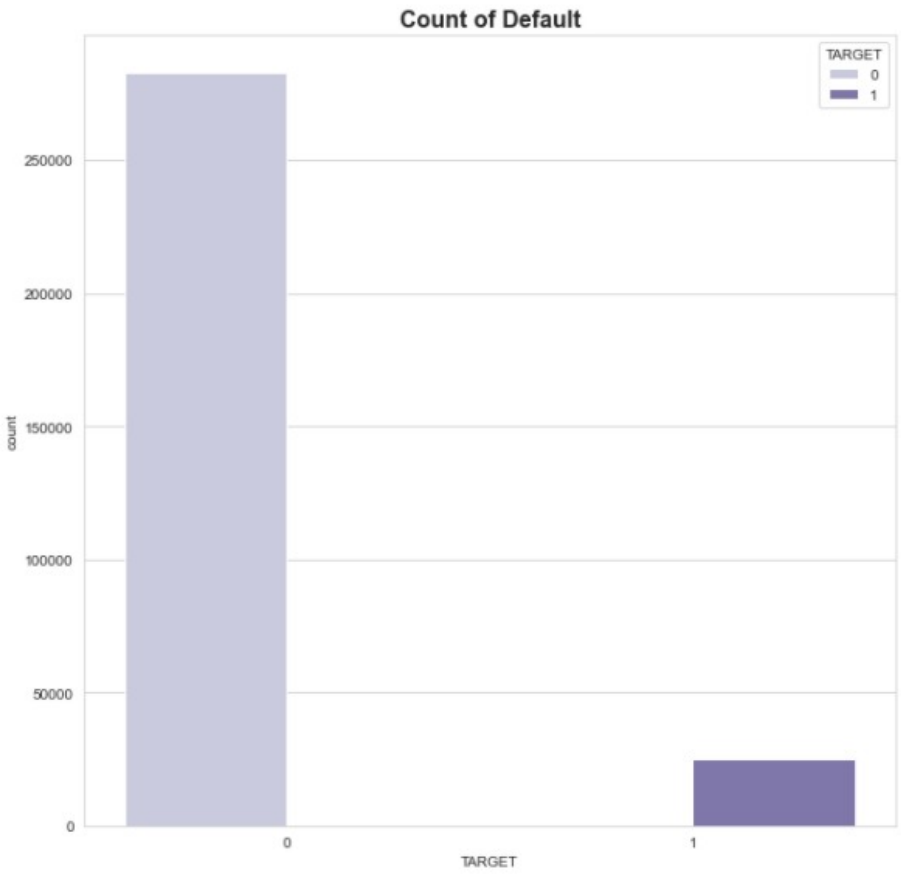
Conclusions of the Correlation Matrix After we see that there are many independent or feature variable which are related to each other, we will deal with them in a way that we drop the repetitive features, i.e. delete one of those columns which are highly correlated with each other! Thus, as we can see above, I have removed all those columns that ae related to each other, just leaving out one!

VISUAL EXPLORATORY DATA ANALYSIS

1. In this phase, we derive meaningful insights from the data by answering reasonable questions that we found that need to be answered from the earlier steps. How is the distribution of target labels? - Did most people return on time ? According to the description of the data -"1 indicates 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 indicates all other cases".

[66]:
```
plt.figure(figsize=(10,10))
plt.title("Count of Default", fontweight = 'bold', fontsize = 16)
sns.countplot(x ='TARGET',data=datasets['application_train'], hue='TARGET',palette="Purples")
```

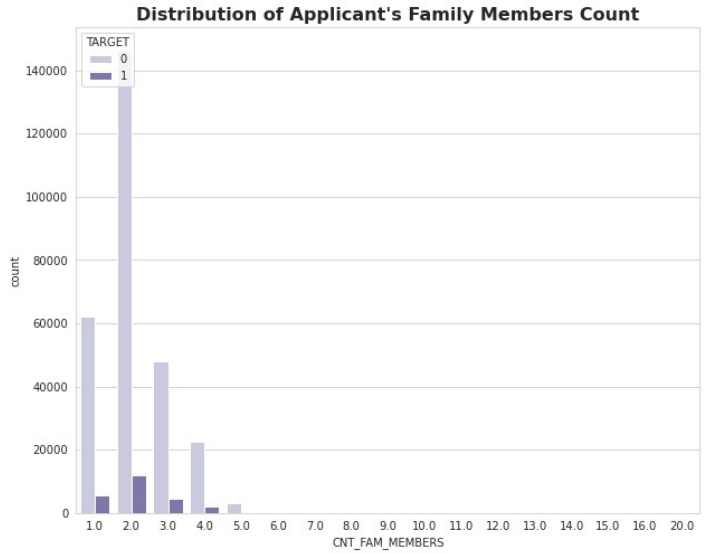
[66]: <AxesSubplot:title={'center':'Count of Default'}, xlabel='TARGET', ylabel='count'>



1. How is the distribution of Applicant's Family Members Count?

[183]:
```
plt.figure(figsize=(10,8))
plt.title("Distribution of Applicant's Family Members Count", fontweight = 'bold', fontsize = 16)
sns.countplot(datasets['application_train']['CNT_FAM_MEMBERS'],hue=datasets['application_train']['TARGET'], palette = 'Purples')
```

[183]: <AxesSubplot:title={'center':"Distribution of Applicant's Family Members Count"}, xlabel='CNT_FAM_MEMBERS', ylabel='count'>

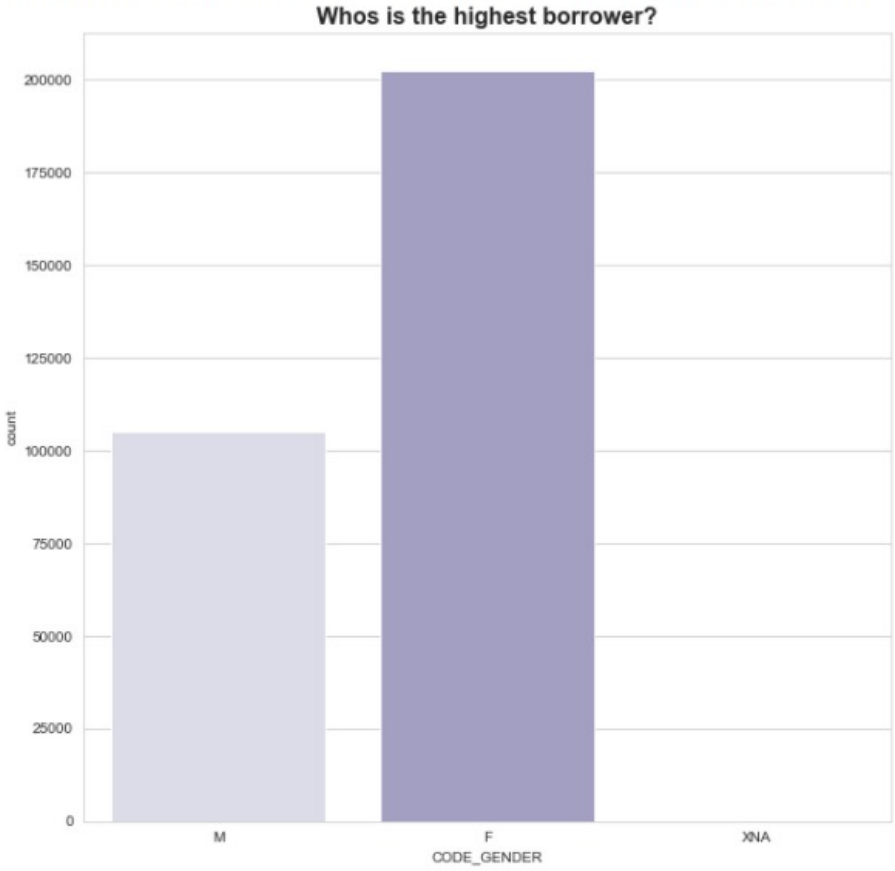


Above, we have plotted a graph which describes the loan applicant's family members. As we can see above, there are hardly any people who have a count of family members greater than 5.

Who is the highest borrower ?

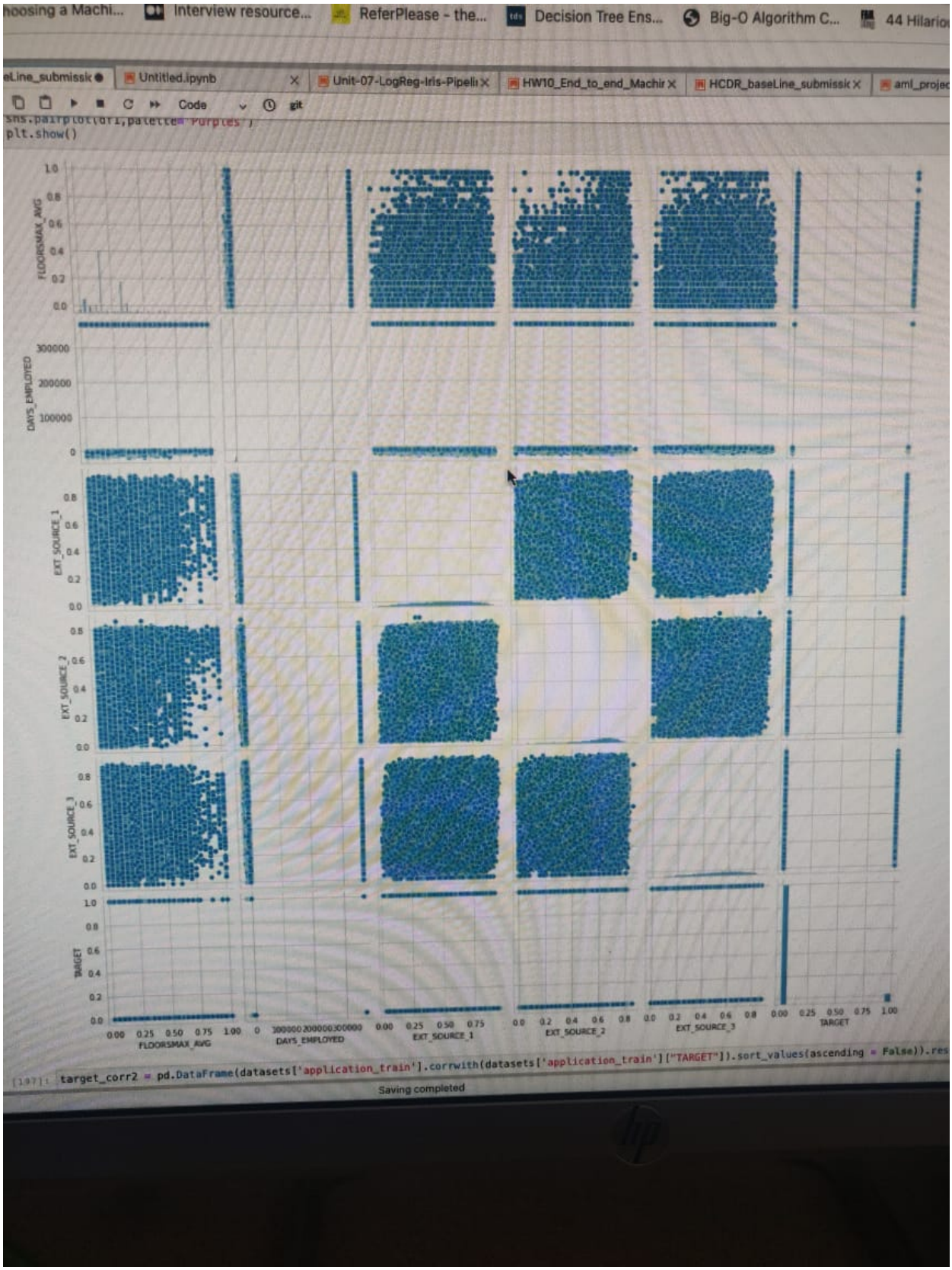
[65]:
```
sns.set_style('whitegrid')
plt.figure(figsize = (10,10))
plt.title("Whos is the highest borrower?", fontweight = 'bold', fontsize = 16)
sns.countplot(x='CODE_GENDER',data=datasets['application_train'], palette = 'Purples')
```

[65]:
```
<AxesSubplot:title={'center':'Whos is the highest borrower?'}, xlabel='CODE_GENDER', ylabel='count'>
```



As we can see Females are the highest borrowers.

PAIR BASED VISUALIZATION



Pairplot showing correlation with target variable for highly correlated variable with target.

MODELING PIPELINES

In this section of the project, the following steps have been performed: Standardizing the data - The numerical variables have been standardized. Handling missing values - Following the good

practice of imputing the missing values in the dataset, we impute the numerical as well as categorical variables as follows. All the work has been performed in pipelines. The numerical variables have been imputed with the median and the categorical variables have been imputed with the most frequent variable.

Handling categorical variables We have used one of the most common and efficient way to handle this transformation, ONE-HOT ENCODING. After the transformation, each column of the resulting data set corresponds to one unique value of each original feature. We want to implement the one-hot encoding to the data set, in such a way that the resulting sets are suitable to use in machine learning algorithms.

[98]:
```
num_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer(strategy = 'median'))
])
cat_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

[100]:
```
data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, numerical_df),
    ("cat_pipeline", cat_pipeline, categorical_df)], remainder = 'drop')

X_train_transformed = data_pipeline.fit_transform(X_train)
```

[101]:
```
column_names = numerical_df  + \
               list(data_pipeline.transformers_[1][1].named_steps["ohe"].get_feature_names(categorical_df))

pd.DataFrame(X_train_transformed, columns=column_names).head()
```

Model training : Now that we have explored the data, cleaned it, preprocessed it and added a new feature to it, we can start the modeling part of the project by applying Machine Learning algorithms. In this section, wel have a baseline logistic regression model and a random forest model. In the end, we will be comparing the performance of the models with the baseline models.

Baseline Logistic Regression

Logistic Regression is a powerful algorithm for classification problems that fit models for categorical data, especially for binary classification problems. Since our target (dependent) variable is categorical, using logistic regression can directly predict the probability that a customer is creditworthy (able to meet a financial obligation in a timely manner) or not, using a number of predictors.

Loss function used (data loss and regularization parts) in latex -

$$\text{Cost}(h_\Theta(x), Y(\text{actual})) = -\log(h_\Theta(x)) \text{ if } y=1$$

$$-\log(1 - h_\Theta(x)) \text{ if } y=0$$

Accuracy and AUC/ROC - Accuracy: Accuracy represents the number of correctly classified data instances over the total number of data instances. Below is the accuracy and the AUC/RUC of our logistic regression model.

[96]:
```
print('Validation set accuracy score: ' + str(accuracy_score(y_test,y_pred)))
```

```
Validation set accuracy score: 0.919337268100743
```

[97]:
```
log_loss(y_test,y_pred)
```

[97]:
```
2.7859928665299676
```

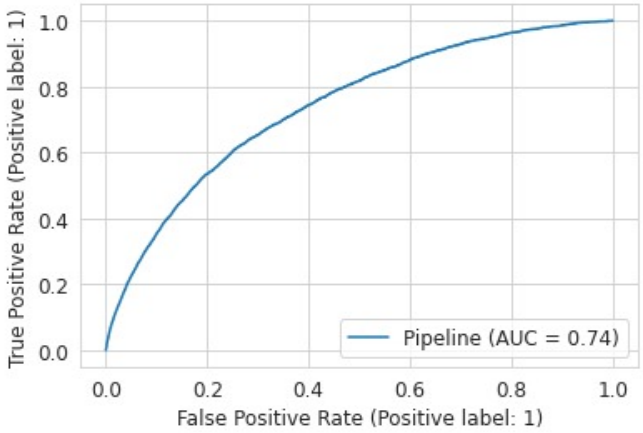
[98]:
```
roc_auc_score(y_test, clf_pipe.predict_proba(X_test)[:, 1])
```

[98]:
```
0.7417601109275497
```

[99]:
```
import matplotlib.pyplot as plt
from sklearn import metrics
```

[100]:
```
metrics.plot_roc_curve(clf_pipe, X_test, y_test)
```

[100]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x4ae8926810>
```



Baseline Random Forest

Random Forest is one of the most popular algorithms which assembles a large number of decision trees from the training dataset. Each decision tree represents a class prediction. This method collects the votes from these decision trees and the class with the most votes is considered as the final class. Random forest outperforms linear models because it can catch non-linear relationships between the object and the features.

```
RF = RandomForestClassifier(random_state = 42,n_estimators=20, criterion='gini', max_depth=6)

data_pipeline_rf = make_pipeline(data_pipeline, RF)
start = time()
data_pipeline_rf.fit(X_train, y_train)
train_time = np.round(time() - start, 4)
train_acc = data_pipeline_rf.score(X_train, y_train)
validAcc  = data_pipeline_rf.score(X_test, y_test)
start = time()
testAcc  = data_pipeline_rf.score(X_test, y_test)
test_time = np.round(time() - start, 4)

predictions = data_pipeline_rf.predict_proba(X_test)
print ("ROC_AUC_Score",roc_auc_score(y_test, predictions[:,1]))

fpr, tpr, _ = roc_curve(y_test, predictions[:,1])

plt.clf()
plt.plot(fpr, tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve',fontweight='bold',fontsize=20)
plt.show()
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc", "ValidAcc", "TestAcc",
                                               "Train Time(s)",  "Test Time(s)", "Description",])
experimentLog.loc[len(experimentLog)] =[f"Baseline 1 RandomForest", "HCDR",
                                        f"{trainAcc*100:8.2f}%", f"{validAcc*100:8.2f}%", f"{testAcc*100:8.2f}%",
                                        train_time, test_time,
                                        "Baseline 1 RandomForest pipeline with Cat+Num features"]

experimentLog
```

```
ROC_AUC_Score 0.7253912369656725
```

Loss function used (data loss and regularization parts) in latex -

$$\text{Cost}(h_\Theta(x), Y(\text{actual})) = -\log(h_\Theta(x)) \text{ if } y=1$$

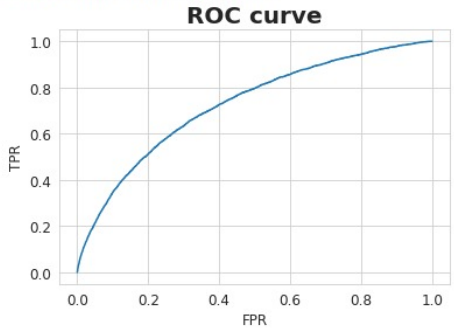
$$-\log(1- h_\Theta(x)) \text{ if } y=0$$

Accuracy and AUC/ROC - Accuracy: Accuracy represents the number of correctly classified data instances over the total number of data instances. ROC / AUC: An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: True Positive Rate False Positive Rate AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example.

Below is the accuracy and the AUC/RUC of our random forest model.

```
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc", "ValidAcc", "TestAcc",
                                               "Train Time(s)",  "Test Time(s)", "Description",])
experimentLog.loc[len(experimentLog)] =[f"Baseline 1 RandomForest", "HCDR",
                                        f"{trainAcc*100:8.2f}%", f"{validAcc*100:8.2f}%", f"{testAcc*100:8.2f}%",
                                        train_time, test_time,
                                        "Baseline 1 RandomForest pipeline with Cat+Num features"]

experimentLog
```

```
ROC_AUC_Score 0.7253912369656725
```



[123]:

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Description
0	Baseline 1 LogReg	HCDR	91.92%	91.93%	91.93%	92.2735	2.1092	Baseline 1 LogReg pipeline with Cat+Num features
1	Baseline 1 RandomForest	HCDR	91.92%	91.95%	91.95%	42.8980	2.6084	Baseline 1 RandomForest pipeline with Cat+Num ...
2	Baseline 1 RandomForest	HCDR	91.92%	91.95%	91.95%	43.5266	2.5553	Baseline 1 RandomForest pipeline with Cat+Num ...

Number of experiments conducted We have selected 2 models for prediction : Logistic Regression and Random Forest Classifier

Write Up Phase 2

FP Phase 2 - Final Project HCDR - feature engineering + hyperparameter tuning

Home Credit Default Risk Kaggle Competition

PROJECT MEMBERS:

ADITI MULYE - adimulye@iu.edu

KESHAV LIKHAR - klikhar@iu.edu

NIKUNJ MALPANI - nmalpani@iu.edu

PRASHASTI KARLEKAR - prkarl@iu.edu

Aditi Mulye
adimulye@iu.edu

Keshav Likhar
klikhar@iu.edu

Nikunj Malpani
nmalpani@iu.edu

Prashasti Karlekar
prkarl@iu.edu

PROJECT ABSTRACT

In this phase of the project, our aim is to build on our previous observations from Phase 1 and improve the overall accuracy of our model. The model built in Phase 1 resulted in an accuracy of ~73% on Kaggle and considering the pitfalls of not incorporating additional techniques in our model, we attempt to improve the quality of our dataset and consequently our model by performing Feature Engineering, merging all the datasets available in Home Credit Default Risk dataset, and performing hyper-parameter tuning using GridSearchCV. The first part of the Phase 2 deals with merging all the datasets available to us on the basis of primary key/ combination of keys in the datasets. To check on the progress of the quality of our dataset, we perform exploratory data analysis like heatmap , correlation matrix and missing number visualization on each dataset to better understand the important features in the individual dataset and removing the insignificant features on the basis of missing value counts and highly correlated features. The second part of the Phase 2 uses the merged dataset to build additional features in our dataset which we consider can improve the predictive power of our model. These domain knowledge features prove to be successful in attempting to increase the quality of the model as they are highly correlated with the target variable. The third part of Phase 2 revolves around building pipelines and tuning the hyperparameters with GridSearchCV and incorporate the most significant in our model. We then predict on our final dataset using Logistic Regression and Random Forest algorithms thereby improving the accuracy to 92.1% which was 91.85 before and on Kaggle ~74%.

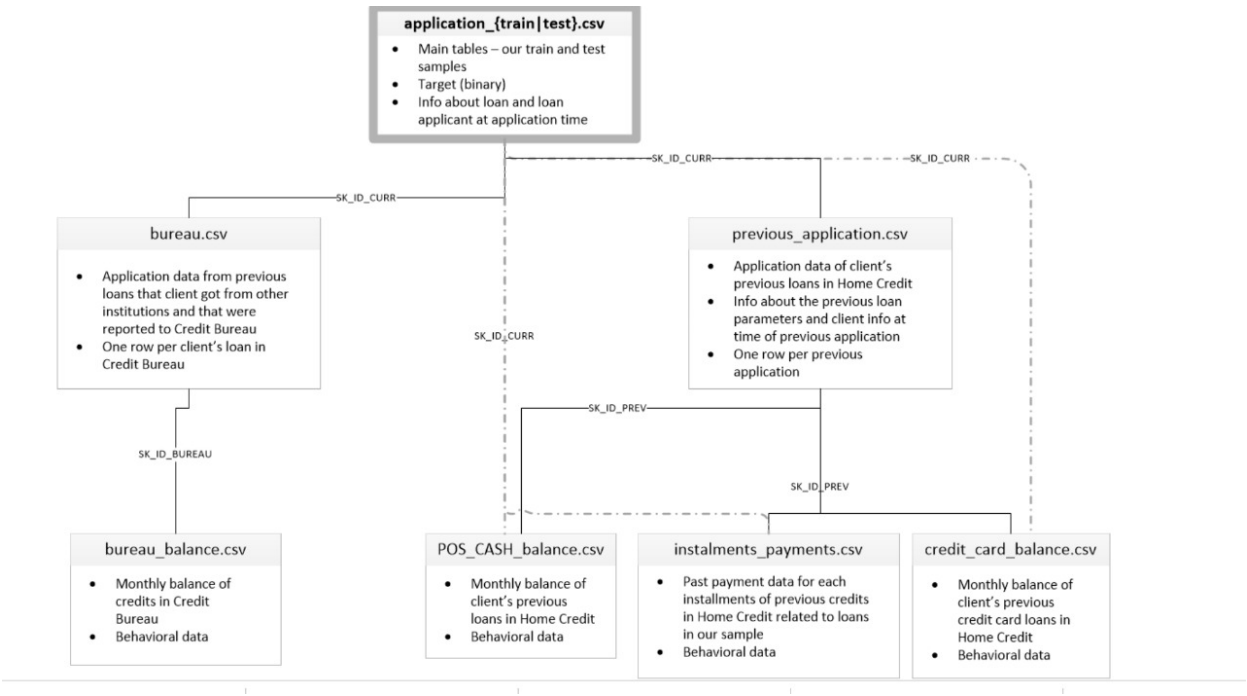
PROJECT DESCRIPTION

1. Data description The home-credit data is currently available on Kaggle for predicting whether or not a client will repay a loan or have difficulty, which is a critical business need. In this dataset we have a total of 122 columns. There are a lot of columns which have missing values in the dataset. As we can already see that there are so many missing values in the top 10 rows itself. We'll have to figure out a way to deal with this! There are 7 different sources of data:

- application_train/application_test: the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the

feature SK_ID_CURR. The training application data comes with the TARGET indicating 0: the loan was repaid or 1: the loan was not repaid.

- bureau: data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- bureau_balance: monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length. previous_application: previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- POS_CASH_BALANCE: monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- installments_payment: payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment. This diagram shows how all of the data is related:

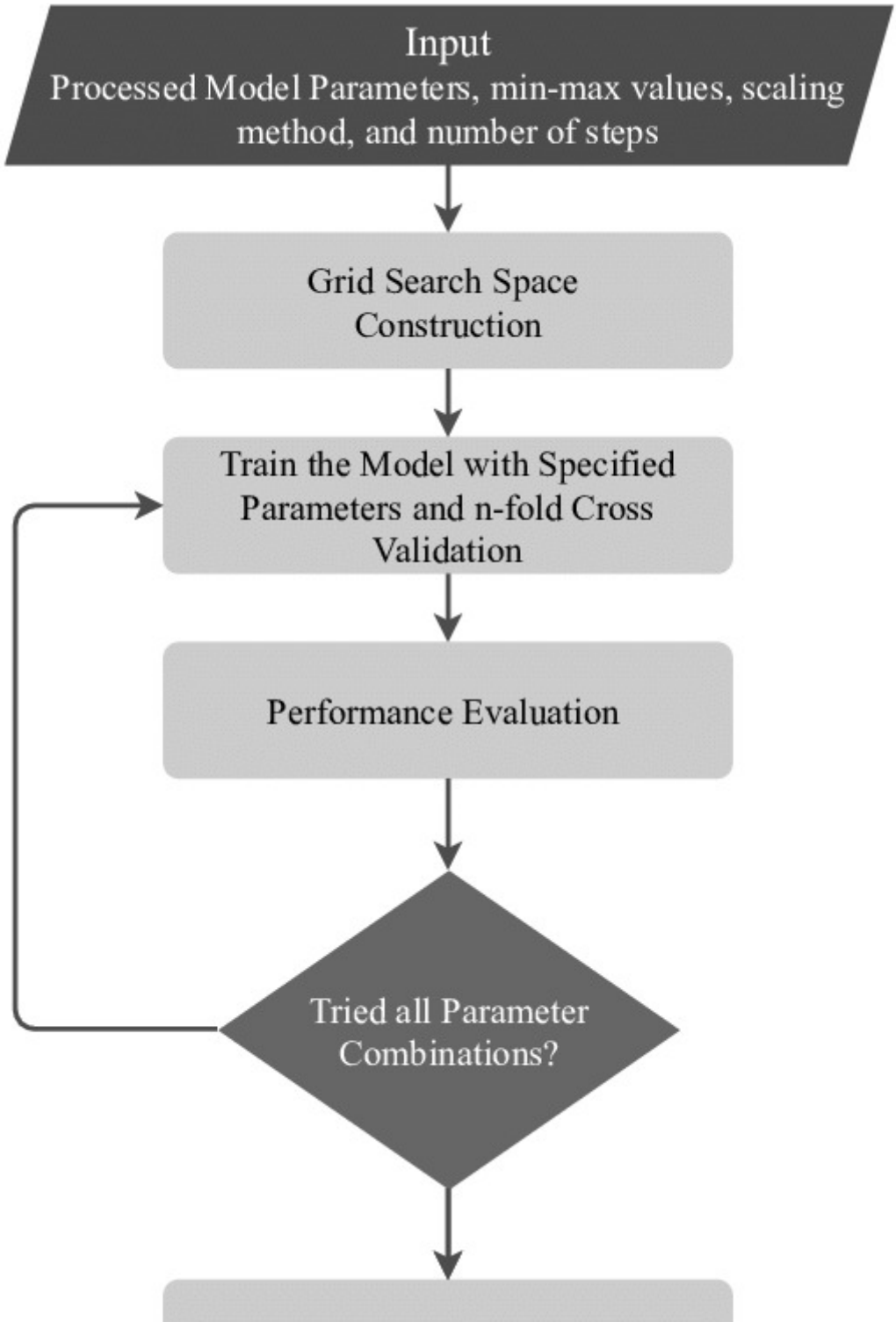


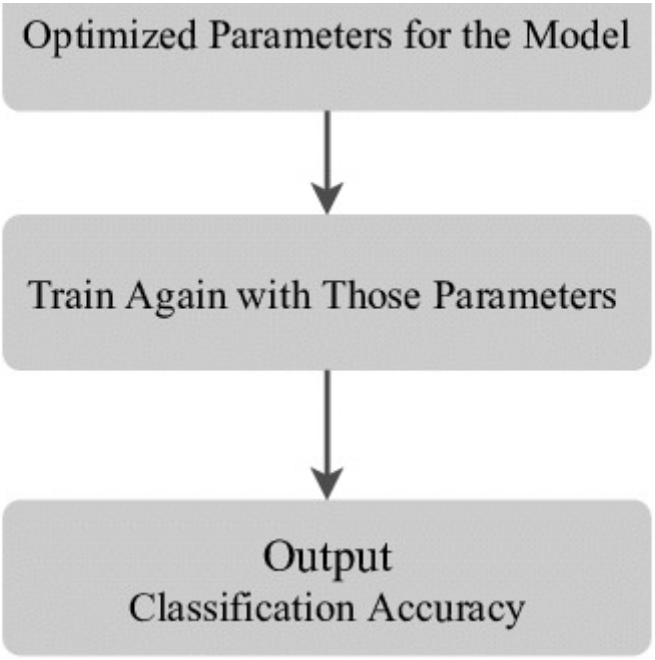
Task to be tackled

The task at hand for this phase of the project is to

- Join the datasets
- Perform EDA on other datasets, i.e. excluding application_train and the merged datasets
- Find missing values and highly correlated features in the merged data

- Figure out and mitigate possible errors in the merged data that could render our model inaccurate
- Add domain knowledge features that could improve the performance of the model
- Show the impact of the newly added features with the target variable
- Perform hyper-parameter tuning with GridSearchCV and find the most significant hyperparameters
- Show the impact of selecting these hyperparameters
- Build on the models previously considered in Phase 1 i.e. Logistic Regression and Random Forest
- Calculate and validate the results





Feature Engineering

The better the features that we prepare and choose from the dataset, the better the results. And this is why we perform feature engineering. We need great features that describe the structures inherent in the data. The main goal of Feature Engineering is to get the best results from our algorithms.

The following is the initial Feature Engineering performed on the dataset in Phase 1 :

- Standardize the data
- Impute the data
- Split the data into training and test subsets
- Handling the categorical and numerical variables separately

In this phase, in order to improve the accuracy of our model and to utilize all the features and data from all the datasets, we first merge the datasets together.

Lets look at each dataset individually and merge. More about features engineered are explained in detail below in the notebook.

Using Bureau Dataset

This dataset has 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. Here, one of the interesting features is 'CREDIT_ACTIVE' which shows the data about the status of the credits.

```
In [112]: datasets['bureau']['CREDIT_ACTIVE'].value_counts()

Out[112]: Closed      1079273
          Active       630607
          Sold           6527
          Bad debt         21
          Name: CREDIT_ACTIVE, dtype: int64
```

Merging of Bureau with Bureau Balance

We merge Bureau abd Bureau Balance datasets on on SK_ID_BUREAU.

```
In [115]: df_test = datasets['application_train'][['SK_ID_CURR','TARGET']]
          df_test
```

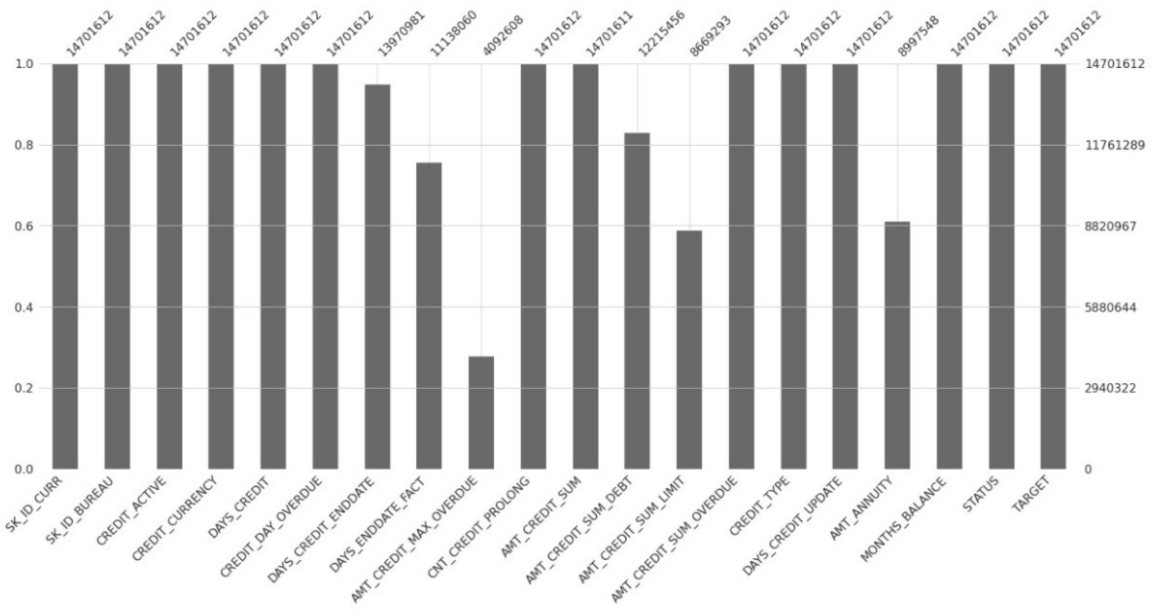
Out[115]:

	SK_ID_CURR	TARGET
0	100002	1
1	100003	0
2	100004	0
3	100006	0
4	100007	0
...	...	...
307506	456251	0
307507	456252	0
307508	456253	0
307509	456254	1
307510	456255	0

307511 rows × 2 columns

```
In [116]: df_mergewithtarget  = pd.merge(left = df_bureau1, right = df_test,
                                         how='left', left_on='SK_ID_CURR', right_on='SK_ID_CURR')
```

Here, we are creating a test dataframe which contains only target with SK_ID_CURR so that we can check the correlation with the merged bureau column so that we can drop irrelevant columns. Since this dataset has not been preprocessed, we perform EDA on this merged dataset by dropping missing values and columns which are highly correlated with the target column from our merged dataset.



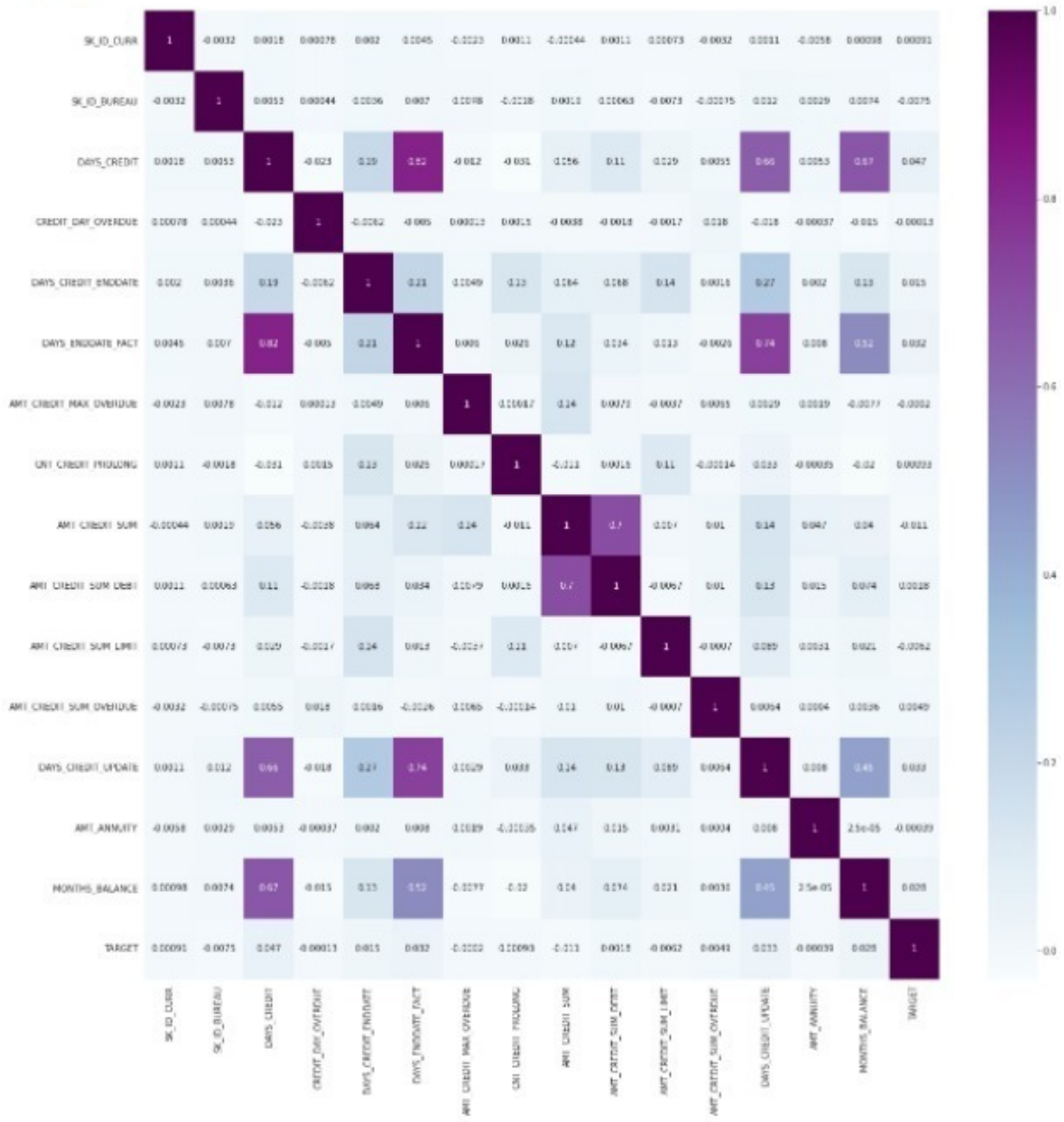
From the missing no graph, we see that 'AMT_CREDIT_SUM_LIMIT' has a lot of missing values, so we impute this column by replacing null values with 0.

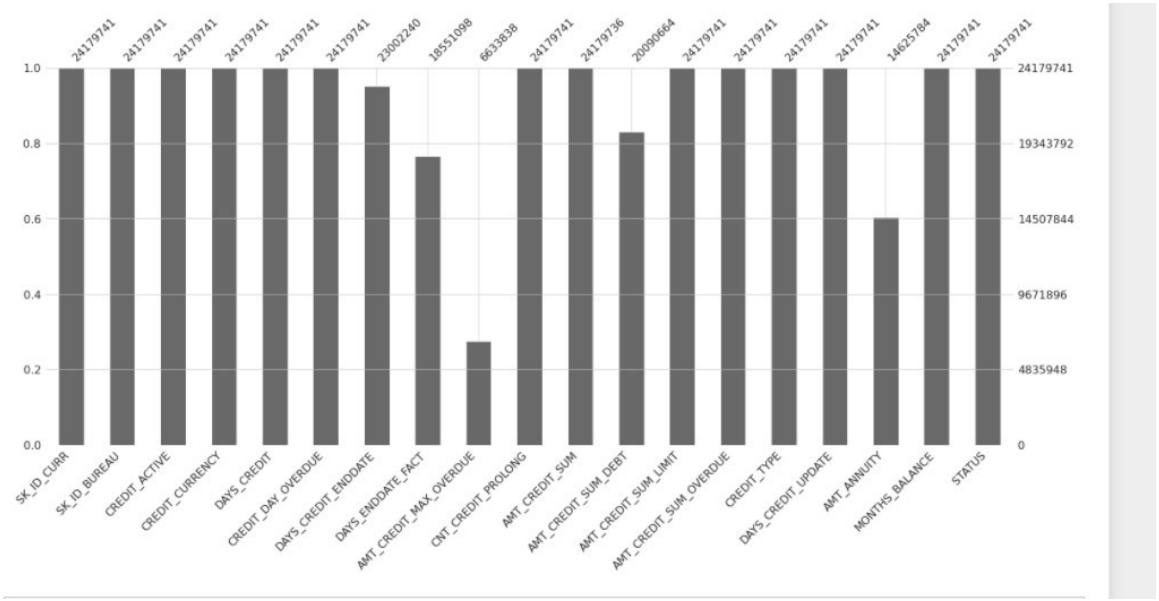
The merged dataset of bureau and bureau balance only contains the features that are important to the final dataset and hence, we perform feature selection to choose only the most important

features of the dataset.

```
In [122]: plt.figure(figsize = (20,20))
          sns.heatmap(df_mergewithtarget1.corr(),annot=True, cmap='BuPu')
```

```
Out[122]: <AxesSubplot:>
```

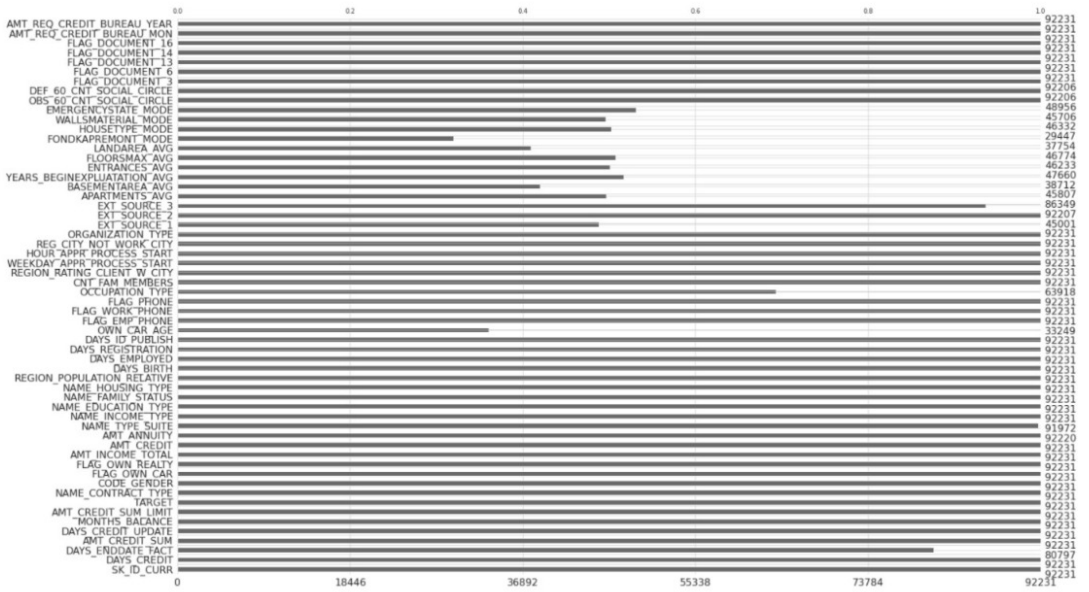




Merging of processed Bureau Data with Application Train Data

We merge our df_bureau_final with application_train on SK_ID_CURR.

```
In [136]: msno.bar(df_finalm_bureau_train)
Out[136]: <AxesSubplot:>
```



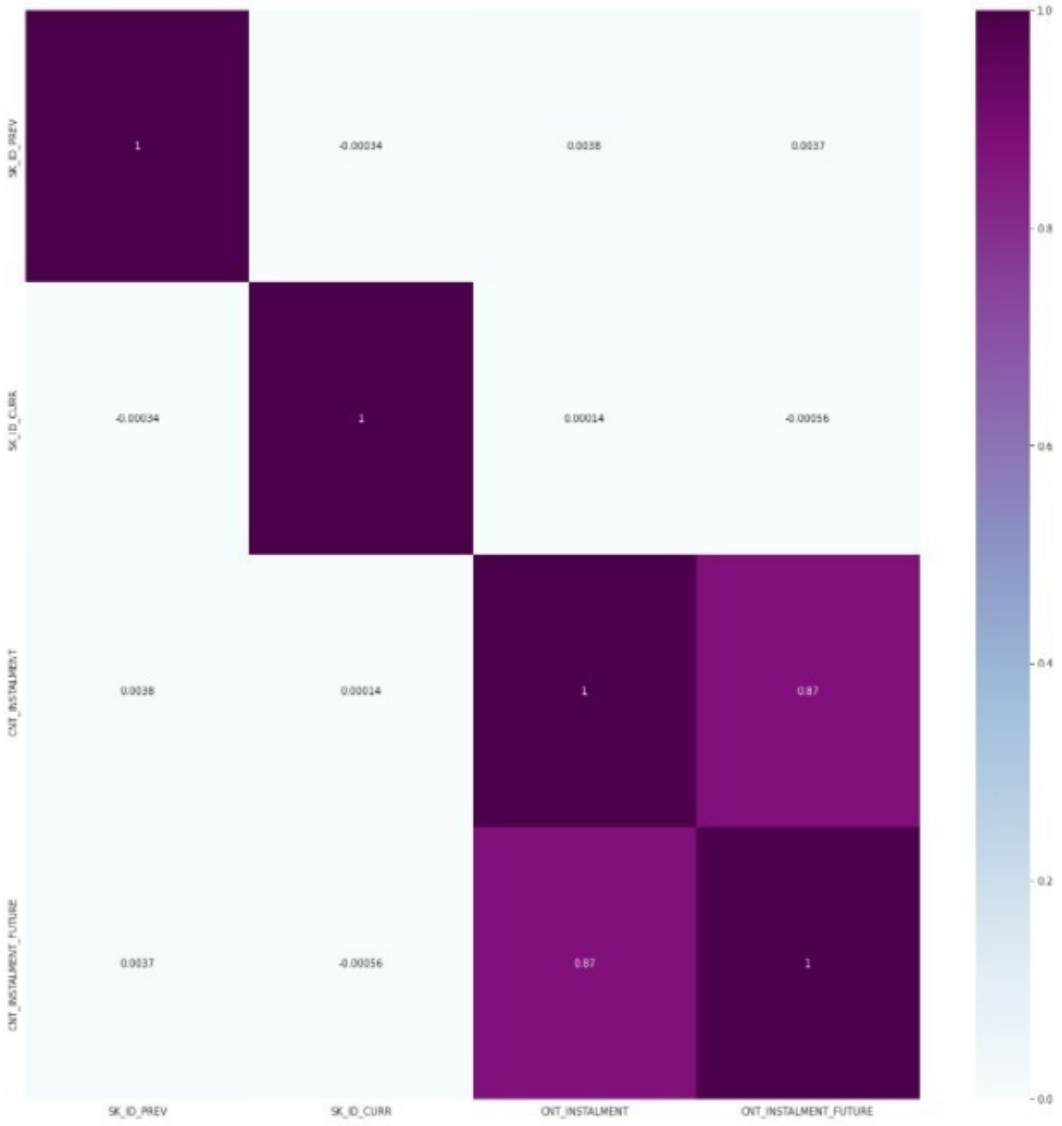
Working on POS Cash Balance

This dataset consists of monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.

We perform EDA on this dataset by finding out the missing values count and removing the irrelevant features as derived from the correlation matrix.

In [142]:
```
plt.figure(figsize = (20,20))
sns.heatmap(df_POS.corr(),annot=True, cmap='BuPu')
```

Out[142]: <AxesSubplot:>



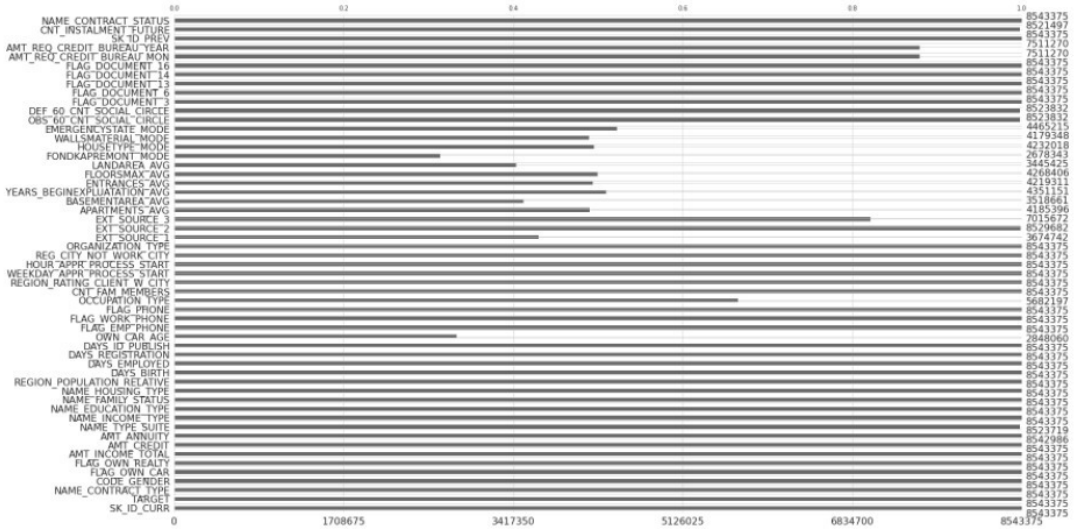
In [143]:
```
df_POS = df_POS.drop(['CNT_INSTALMENT'],axis=1)
```

Merging of POS CASH with Application Train on SK_ID_CURR

We merge our df_POS with application_train on SK_ID_CURR

```
In [147]: msno.bar(df_POS_atrain)
```

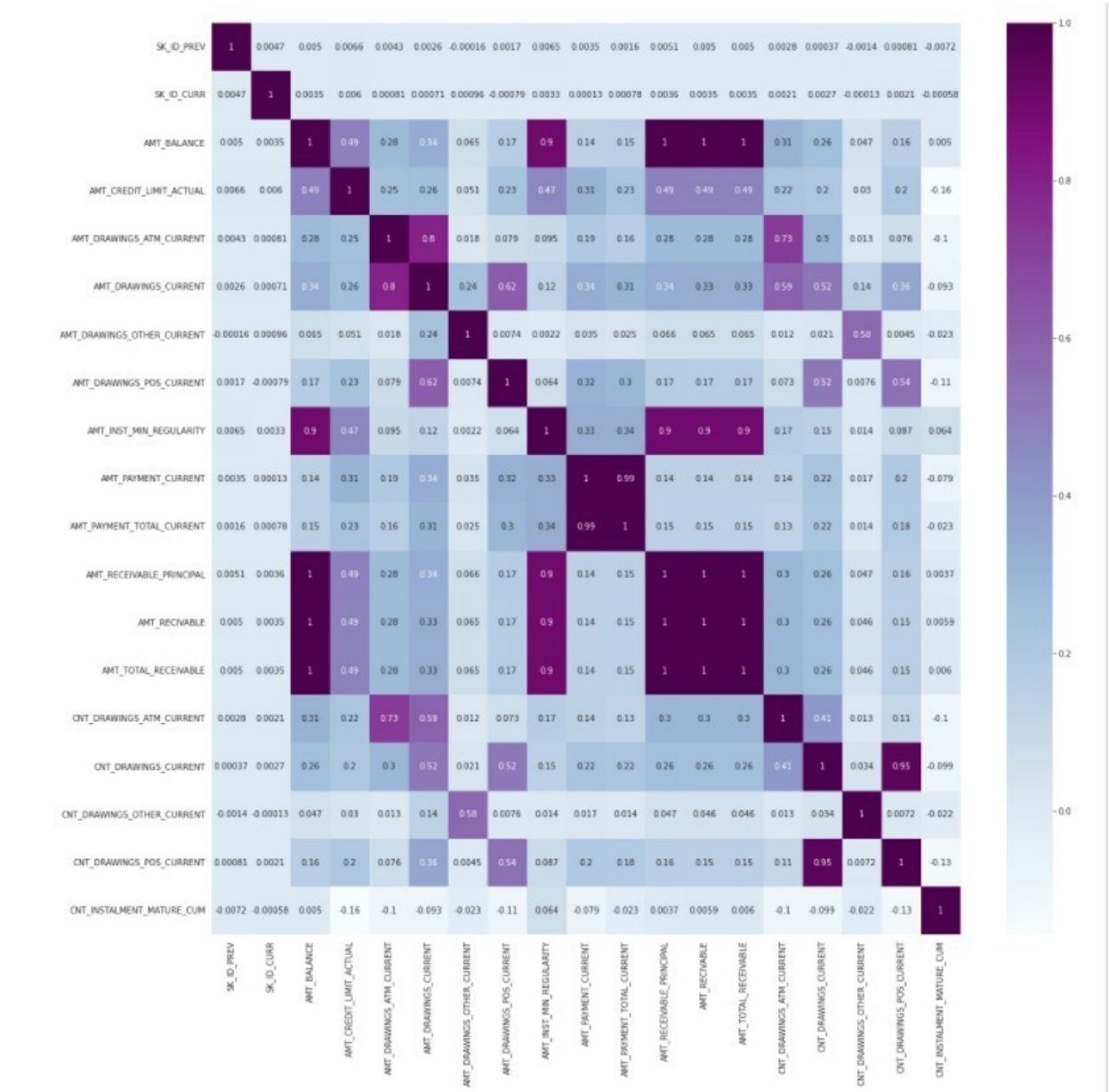
```
Out[147]: <AxesSubplot:>
```



Using Credit Card Balance

This dataset contains the monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.

We perform EDA on this dataset by finding out the missing values count and removing the irrelevant features which have no or insignificant correlation witht the target variable.




```
152]: df_ccbal = df_ccbal.drop(['AMT_INST_MIN_REGULARITY','AMT_RECEIVABLE_PRINCIPAL','AMT_RECIVABLE','AMT_TOTAL_RECEIVABLE',
                                'AMT_PAYMENT_TOTAL_CURRENT','AMT_DRAWINGS_ATM_CURRENT','CNT_DRAWINGS_CURRENT'],axis=1)
```

We also group by SK_ID_CURR and SK_ID_PREV by mean so that we can restore as much information as we can while merging or joining this dataset.

Next, we will work on Installments Payments first and then join everything with Application train after joining installments payments and Credit Card Balance.

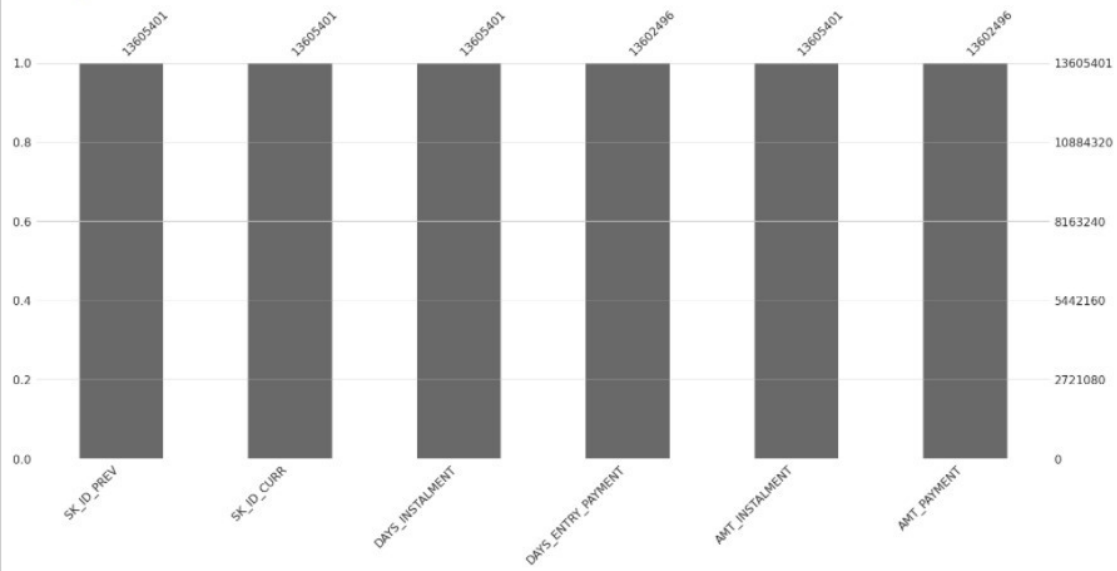
Using Installments Payments

This dataset contains the data of payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

We perform EDA on this dataset by finding out the missing values count and removing the irrelevant features which have no or insignificant correlation witht the target variable.

In [160]:
```
msno.bar(df_installment)
```

Out[160]:



In [161]:
```
df_installment.head()
```

Out[161]:

	SK_ID_PREV	SK_ID_CURR	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT	AMT_PAYMENT
0	1054186	161674	-1180.0	-1187.0	6948.360	6948.360
1	1330831	151639	-2156.0	-2156.0	1716.525	1716.525
2	2085231	193053	-63.0	-63.0	25425.000	25425.000

We group by SK_ID_CURR and SK_ID_PREV by mean so that we can restore as much information as we can while merging or joining the datasets.

In [164]:
```
df_installment1.head()
```

Out[164]:

	SK_ID_CURR	SK_ID_PREV	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT	AMT_PAYMENT
0	100001	1369693	-1664.0	-1679.500000	7312.725000	7312.725000
1	100001	1851984	-2886.0	-2882.333333	3981.675000	3981.675000
2	100002	1038818	-295.0	-315.421053	11559.247105	11559.247105
3	100003	1810518	-626.0	-630.428571	164425.332857	164425.332857
4	100003	2396755	-2145.0	-2151.750000	6731.115000	6731.115000

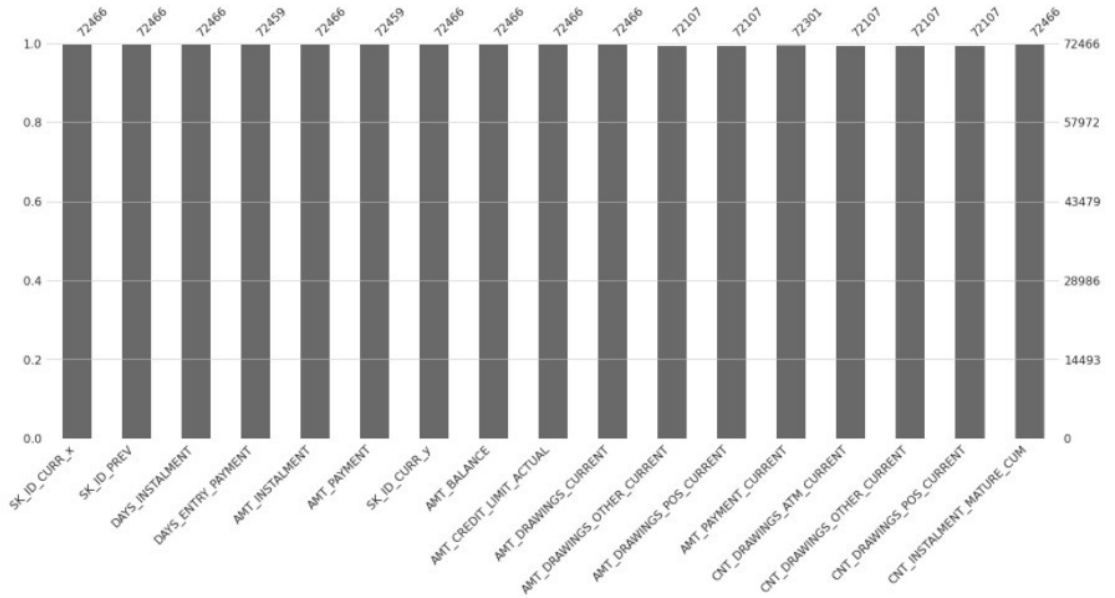
In [165]:
```
df_ccbal1['SK_ID_PREV'].value_counts()
```

Out[165]:
```
1489396    1
1645533    1
2835518    1
2116495    1
2487050    1
          ..
2088402    1
2663364    1
2002588    1
1642103    1
1794451    1
Name: SK_ID_PREV, Length: 104307, dtype: int64
```

Joining Credit Card Balance and installments Payments on SK_ID_PREV and SK_ID_CURR, grouping by both tables by these both the ids by mean and then inner join on PREV only

We then perform EDA and preprocess the data in the merged dataset of credit card balance and installment payments. We are doing this even after merging each individual table is already processed because there might be some columns that are now insignificant or higly correlated with each other.

```
In [168]: msno.bar(df_ccbal_installment1)
Out[168]: <AxesSubplot:>
```

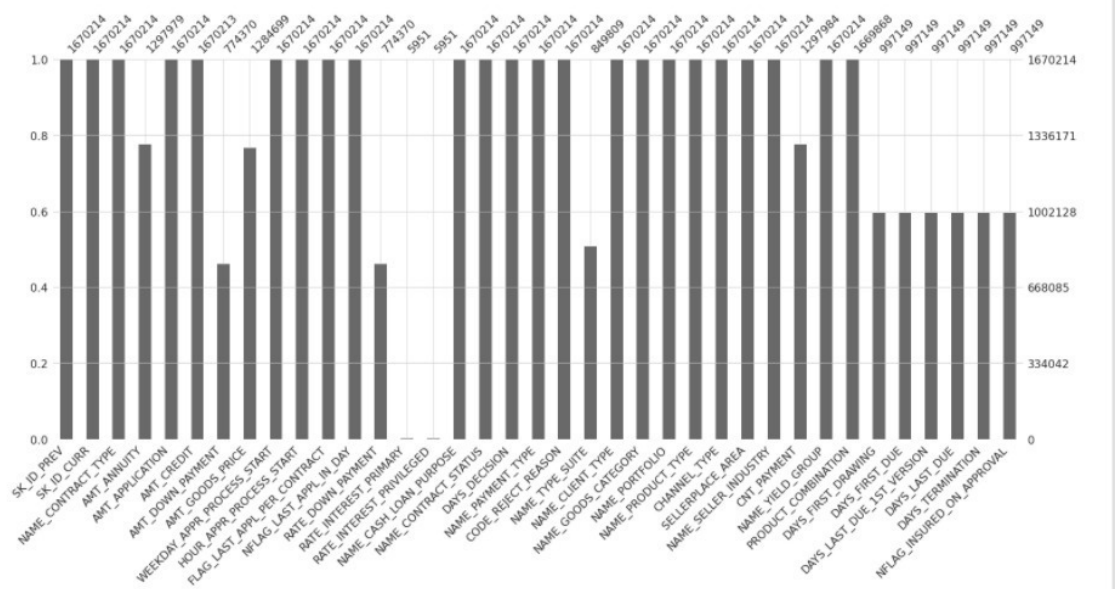


Using Previous Application Data

This dataset contains the data of previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.

We remove the columns with high correlation and containing large percentage of NULL values while performing EDA and preprocessing this dataset.

```
In [176]: msno.bar(df_prev_app)
Out[176]: <AxesSubplot:>
```



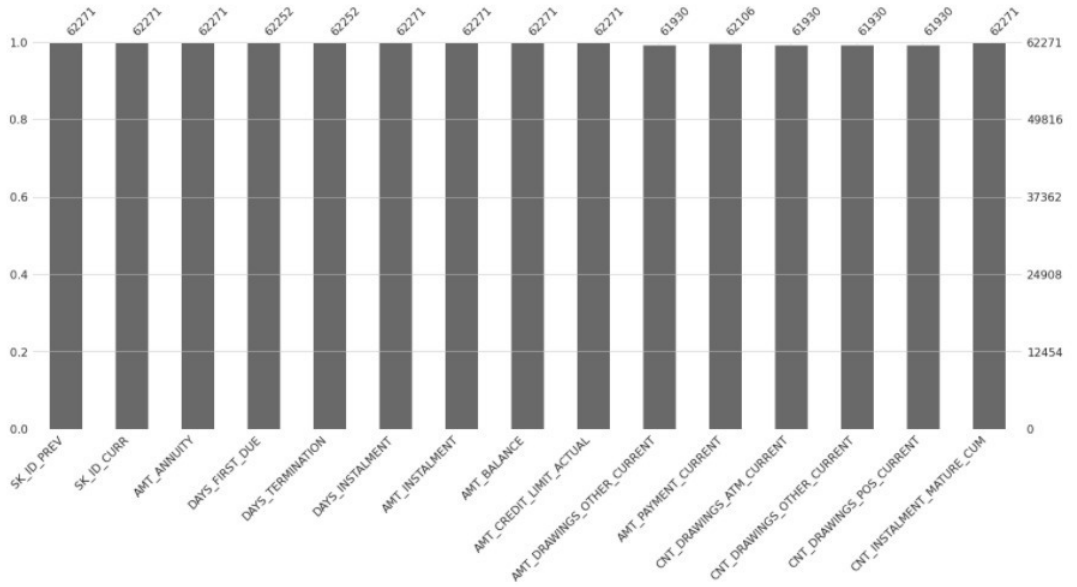

```
In [177]: df_prev_app = df_prev_app.drop(['RATE_INTEREST_PRIVILEGED','RATE_INTEREST_PRIMARY'],axis=1)
```

We group by SK_ID_CURR and SK_ID_PREV by mean so that we can restore as much information as we can while merging or joining

Joining Previous Application and previously merged dataframe (Credit card balance + Installments Payments)

After merging all the preprocessed daasets, we perform EDA and preprocess this merged dataset as well because there might be some columns that are now insignificant or higly correlated with each other.

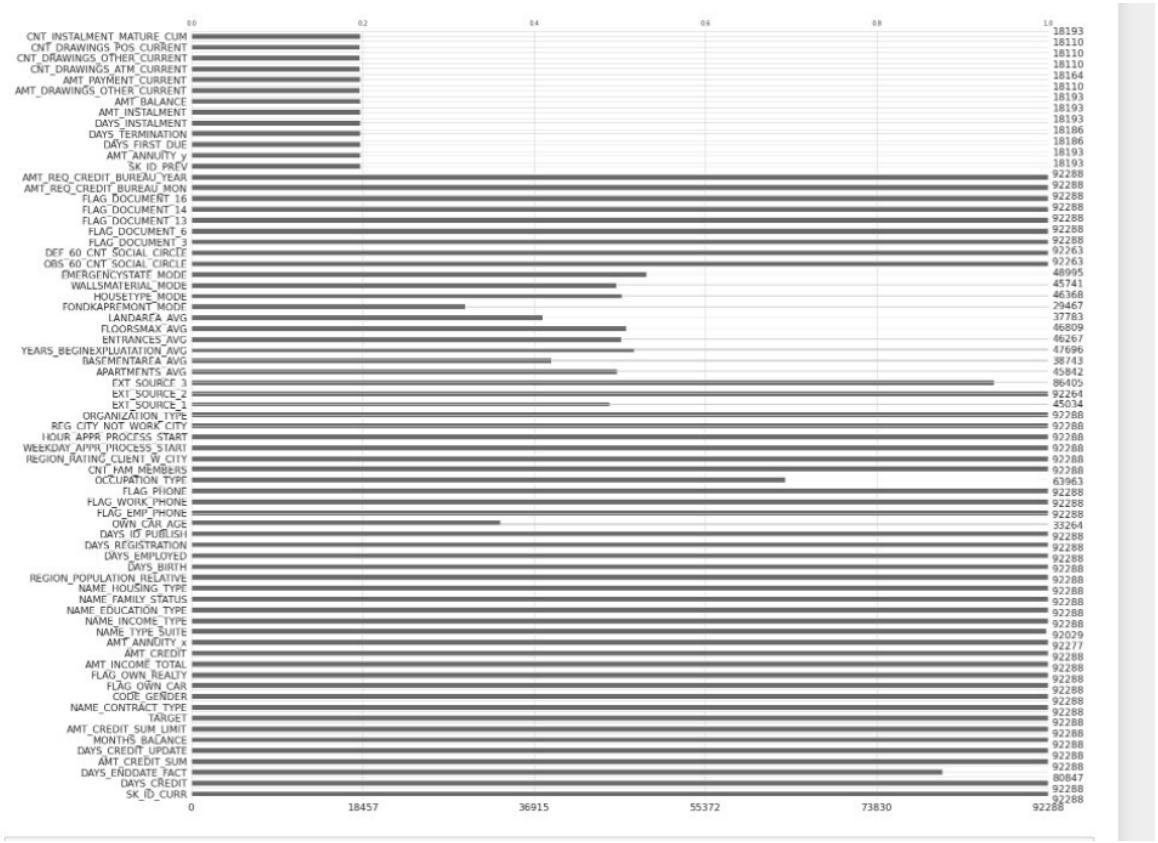
```
In [190]: msno.bar(df_prev_ccbalinst)
Out[190]: <AxesSubplot:>
```



Merging all the pre-processed merged datasets

We merge df_PREV_CCBALINST with already merged Application_train on Bureau Data. We left join the datasets on SK_ID_CURR.

We then perform EDA on the final merged dataset because because there might be some columns that are now insignificant or higly correlated with each other.



Additional features added to the final dataset

While performing EDA and with our domain knowledge, we found out some existing features whose combination can have strong explanatory/predictive power.

Here is the description of the additional features :

Feature 1 : 'DAYS_EMPLOYED_PCT' :

Percentage of days employed - How long a person has been employed as a percentage of his life is a stronger predictor of his ability to keep paying off his loans.

We have taken this parameter because the total number of days a person has been employed in his life time will impact in his credit paying ability and since this is highly correlated to the target variable we have taken this feature to create new feature out of it.

Feature 1

In [ ]:
```
df_final_merged['DAYS_EMPLOYED_PCT'] = df_final_merged['DAYS_EMPLOYED'] / df_final_merged['DAYS_BIRTH']
```

df_final_merged['DAYS_EMPLOYED_PCT'] = df_final_merged['DAYS_EMPLOYED'] / df_final_merged['DAYS_BIRTH']

Feature 2 : 'CREDIT_INCOME_PCT' :

Available credit as a percentage of income - If a person has a very large amount of credit available as a percentage of income, this can impact his ability to pay off the loans If a person

has a very high credit limit with respect to its earning then this will effect much in its loan paying capability and he or she will or maybe most of the times will not be able to pay back.

Feature 2

```
In [285]: df_final_merged['CREDIT_INCOME_PCT'] = df_final_merged['AMT_CREDIT'] / df_final_merged['AMT_INCOME_TOTAL']
```

```
df_final_merged['CREDIT_INCOME_PCT'] = df_final_merged['AMT_CREDIT'] / df_final_merged['AMT_INCOME_TOTAL']
```

Feature 3 : 'ANNUITY_INCOME_PCT' :

Annuity as a percentage of income - If a person receives an annuity, this is a more stable source of income thus if it is higher, you are less likely to default.

We have taken this feature because of the reason mentioned in the previous line and since it is highly correlated with the target variable.

Feature 3

```
In [ ]: df_final_merged['ANNUITY_INCOME_PCT'] = df_final_merged['AMT_ANNUITY_x'] / df_final_merged['AMT_INCOME_TOTAL']
```

```
df_final_merged['ANNUITY_INCOME_PCT'] = df_final_merged['AMT_ANNUITY_x'] / df_final_merged['AMT_INCOME_TOTAL']
```

Feature 4 : 'CREDIT_TERM_PCT' :

Annuity as a percentage of available credit - If a person receives an annuity, this is more stable source of income thus if it is a high percentage compared to his/her credit availability then the person is more likely be able to pay off his debts.

Feature 5 : 'AMT_BALANCE_PCT' :

Amount Balance as the percentage - The remaining balance in the account is very much an important feature as this tells much about a person's ability to pay back it's debt. This is the reason that we took into accoun this feature also because of it's correlation with the target feature.

Feature 6 : 'AVG_INCOME_EXT_PCT' :

Average of three External Sources of Income - Since these three features are the highest among all the other features that are correlated with the target variable, we took the average of all the three incomes and created a new feature that tells us how they are effecting the prediction of target varible. This feature is almost 13% correlated.

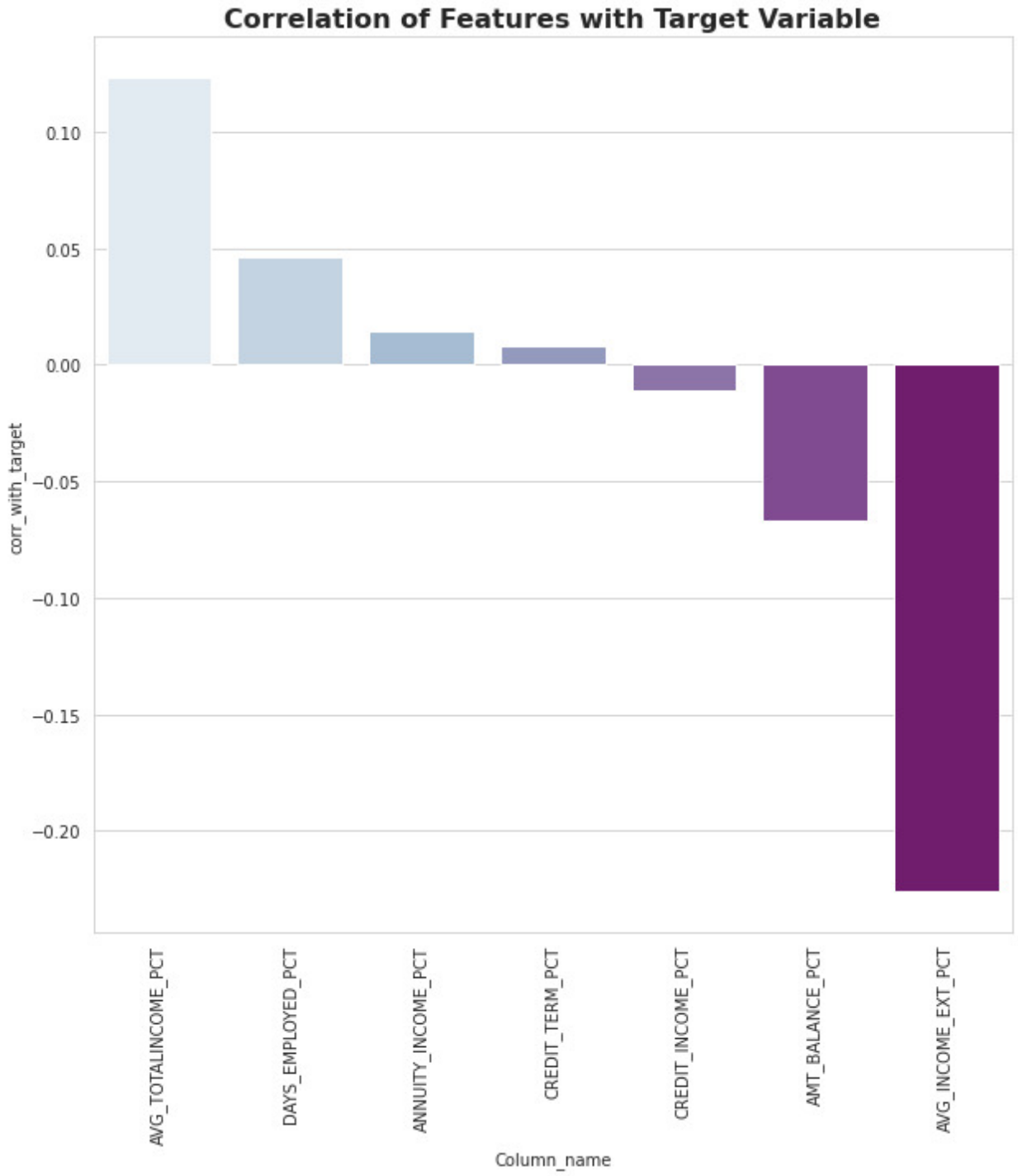
Feature 7: 'AVG_TOTALINCOME_PCT' :

Total Income avergae as percentage - This is the best feature that we generated out of all the seven newly made features. This shows the total income as a percent of the person's earning

with respect to it's external income. We engineered this using the Feature 6 that we created above and this is almost 22% negatively correlated with the get variable.

After adding these additional features, we analyze the expert features and find that AVG_TOTALINCOME_PCT and DAYS_EMPLOYED_PCT ranks highly as measured by correlation in relation to TARGET.

```
In [288]: df_final_merged.corrwith(df_final_merged["TARGET"]).sort_values(ascending = False)

Out[288]: TARGET                      1.000000
          AVG_TOTALINCOME_PCT         0.123221
          CNT_DRAWINGS_ATM_CURRENT    0.108122
          DAYS_INSTALMENT             0.082245
          DAYS_CREDIT                 0.077823
                                          ...
          AMT_BALANCE_PCT            -0.066617
          EXT_SOURCE_2               -0.136431
          EXT_SOURCE_1               -0.148456
          EXT_SOURCE_3               -0.175782
          AVG_INCOME_EXT_PCT         -0.225926
          Length: 62, dtype: float64
```



Hyperparameter Tuning

Hyperparameters contain the data that govern the training process itself. Hyperparameter tuning works by running multiple trials in a single training job. In this part of the phase, we fit and store our result with GridSearchCV.

Modeling Pipelines

Modeling Pipelines (HCDR)

- 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
- Loss function used (data loss and regularization parts) in latex
- Number of experiments conducted
- Experiment table with the following details per experiment: ----- Baseline experiment ----- Any additional experiments ----- Final model tuned ----- 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:

In this section of the project, the following steps have been performed: Standardizing the data - The numerical variables have been standardized. Handling missing values - Following the good practice of imputing the missing values in the dataset, we impute the numerical as well as categorical variables as follows. All the work has been performed in pipelines. The numerical variables have been imputed with the median and the categorical variables have been imputed with the most frequent variable.

Handling categorical variables

We have used one of the most common and efficient way to handle this transformation, ONE-HOT ENCODING. After the transformation, each column of the resulting data set corresponds to one unique value of each original feature. We want to implement the one-hot encoding to the data set, in such a way that the resulting sets are suitable to use in machine learning algorithms.

In [210]:
```
num_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer(strategy = 'median'))
])
cat_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

Model training

Now that we have explored the data, preprocessed it, and performed feature engineering along with hyper parameter tuning, we can start the modeling part of the project by applying Machine Learning algorithms. In this section, we have our upgraded logistic regression model and a random forest model with GridSearchCV. In the end, we will be comparing the performance of the models.

Logistic Regression

Logistic Regression is a powerful algorithm for classification problems that fit models for categorical data, especially for binary classification problems. Since our target (dependent)

variable is categorical, using logistic regression can directly predict the probability that a customer is creditworthy (able to meet a financial obligation in a timely manner) or not, using a number of predictors.

Logistic Regression with GridSearchCV

In [211]:
```
data_pipeline_merged = ColumnTransformer([
    ("num_pipeline", num_pipeline, numerical_df_merged),
    ("cat_pipeline", cat_pipeline, categorical_df_merged)])

model = Pipeline(steps = [('data_pipeline_merge',data_pipeline_merged),
                          ('classifier', LogisticRegression(solver='lbfgs',penalty='l2'))])
```

In [212]:
```
param_grid = {
    'classifier__C': [0.1,1,10,100]
}
gs = GridSearchCV(model, param_grid, cv=5, n_jobs = -1, verbose = 2, refit = True)
start = time()

gs.fit(X_train_merged,y_train_merged)

trainAcc  = gs.score(X_train_merged, y_train_merged)
validAcc  = gs.score(X_test_merged, y_test_merged)

start = time()
testAcc  = gs.score(X_test_merged, y_test_merged)
test_time = np.round(time() - start, 4)

#del experimentLog
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc", "ValidAcc", "TestAcc",
                                               "Train Time(s)",  "Test Time(s)", "Description",])
experimentLog.loc[len(experimentLog)] =[f"GridSearchCV LogReg", "HCDR",
                                        f"{trainAcc*100:8.2f}%", f"{validAcc*100:8.2f}%", f"{testAcc*100:8.2f}%",
                                        train_time, test_time,
                                        "GridSearchCV LogReg pipeline with Cat+Num features"]

experimentLog
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

Out[212]:

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Description
0	Baseline 1 LogReg	HCDR	91.91%	91.94%	91.94%	20.1773	0.5564	Baseline 1 LogReg pipeline with Cat+Num features
1	Baseline 1 RandomForest	HCDR	91.91%	91.95%	91.95%	13.4482	0.7798	Baseline 1 RandomForest pipeline with Cat+Num ...
2	GridSearchCV LogReg	HCDR	91.79%	91.99%	91.99%	13.4482	0.2749	GridSearchCV LogReg pipeline with Cat+Num feat...

Validation Accuracy

The *validation accuracy* of Logistic Regresion Model was 0.9198721421605808

Log Loss

Loss function used (data loss and regularization parts) in latex -

The *log loss* of the model was 2.7675196811129577

ROC-AUC Curve

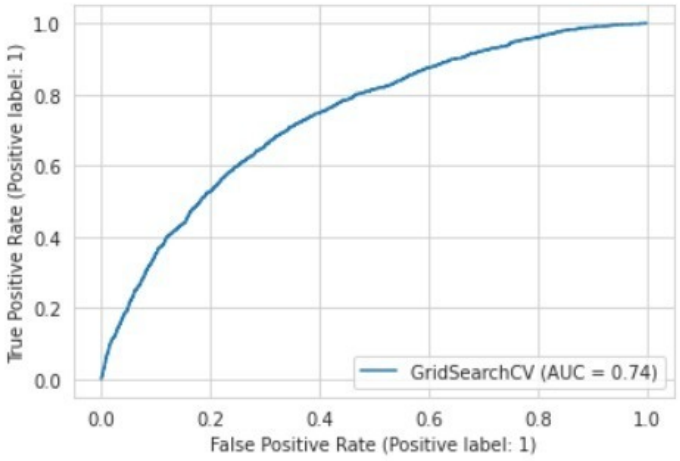
Accuracy: Accuracy represents the number of correctly classified data instances over the total number of data instances. Below is the accuracy and the AUC/RUC of our logistic regression model.

The *ROC-AUC* Curve of the model was 0.738818586860982

ROC-AUC curve (Logistic Regression)

```
In [223]: metrics.plot_roc_curve(gs, X_test_merged, y_test_merged)
```

```
Out[223]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f3dcee770d0>
```



Confusion Matrix of Logistic Regression Classifier

Random forest Classifier with GridSearchCV

Random Forest is one of the most popular algorithms which assembles a large number of decision trees from the training dataset. Each decision tree represents a class prediction. This

method collects the votes from these decision trees and the class with the most votes is considered as the final class. Random forest outperforms linear models because it can catch non-linear relationships between the object and the features.

In [226]:
```
data_pipeline_merged = ColumnTransformer([
    ("num_pipeline", num_pipeline, numerical_df_merged),
    ("cat_pipeline", cat_pipeline, categorical_df_merged)])

model_rf = Pipeline(steps=[('data_pipeline_merged',data_pipeline_merged),
                           ('classifier', RandomForestClassifier())])
```

Experiment Table with all the details of the experiment:

[236]:
```
param_grid = {
   'classifier__n_estimators': [100,200],
#     'classifier__max_depth':[4,5,6,7],
    'classifier__criterion':['gini','entropy']
}
gs_rf = GridSearchCV(model_rf,param_grid, cv=5, n_jobs = -1, verbose = 2, refit = True)
start = time()

gs_rf.fit(X_train_merged,y_train_merged)

trainAcc  = gs_rf.score(X_train_merged, y_train_merged)
validAcc  = gs_rf.score(X_test_merged, y_test_merged)

start = time()
testAcc  = gs_rf.score(X_test_merged, y_test_merged)
test_time = np.round(time() - start, 4)

#del experimentLog
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc", "ValidAcc", "TestAcc",
                                               "Train Time(s)",  "Test Time(s)", "Description",])
experimentLog.loc[len(experimentLog)] =[f"GridSearchCV RandomForestClassifier", "HCDR",
                                        f"{trainAcc*100:8.2f}%", f"{validAcc*100:8.2f}%", f"{testAcc*100:8.2f}%",
                                        train_time, test_time,
                                        "GridSearchCV RandomForestClassifier pipeline with Cat+Num features"]

experimentLog
```

Validation Accuracy

Accuracy represents the number of correctly classified data instances over the total number of data instances. ROC / AUC: An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: True Positive Rate False Positive Rate AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example.

The *validation accuracy* of Random Forest Model was 0.9209556831726081

Log Loss

Loss function used (data loss and regularization parts) in latex - The *log loss* of the model was 2.730093984189766

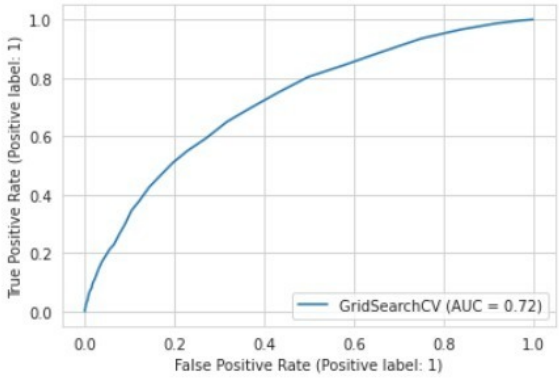
ROC-AUC Curve

The *ROC-AUC* Curve of the model was 0.7241955610236327

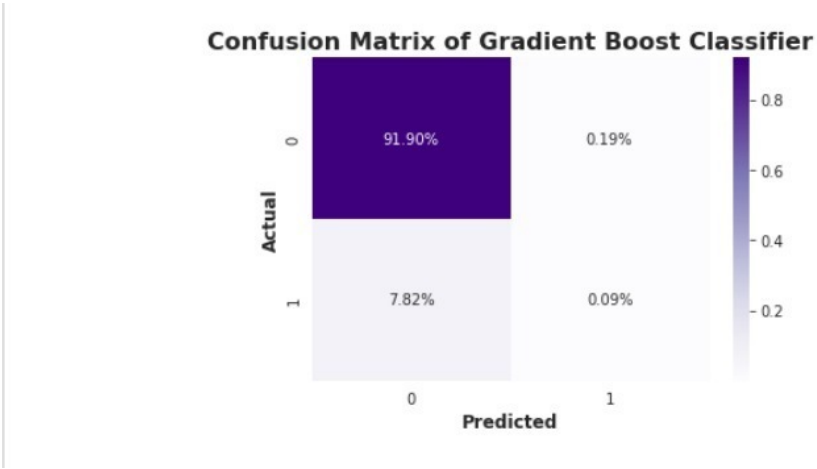
ROC-AUC Curve (Random Forest)

In [234]:
```
metrics.plot_roc_curve(gs_rf, X_test_merged, y_test_merged)
```

Out[234]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f3dcf2d0400>
```



Confusion Matrix of Random Forest



Conclusion and Future Scope

Initially, we tried merging all the different datasets and created several new columns to understand the relation with target column and gain fruitful insights from the dataset. On analysis, we found out that out of the new columns, a few of them were highly correlated with the target variable, thus helping us improve the model predictions. We also tried using synthetic sampling techniques like SMOTE and ADASYN to check if there was any change in the model performance.

Finally, we created a pipeline to perform Logistic Regression and Random Forest Classifier, using Hyperparameter Tuning to gather the best features Successively, we ran our model on the best parameters and found out an increase in the model accuracy and submitted the file on Kaggle.

For the Phase II of this Project, we built Logistic Regression and Random Forest Model, performed Hyperparameter Tuning using GridSearchCV and achieved an accuracy of over 92.1% with Random Forest and 92% with Logistic Regression. Initially, we tried merging all the different datasets and created several new columns to understand the relation with target column and gain fruitful insights from the dataset. On analysis, we found out that out of the new columns, a few of them were highly correlated with the target variable, thus helping us improve the model predictions. We also tried using synthetic sampling techniques like SMOTE and ADASYN to check if there was any change in the model performance. Finally, we created a pipeline to perform Logistic Regression and Random Forest Classifier, using Hyperparameter Tuning to gather the best features Successively, we ran our model on the best parameters and found out an increase in the model accuracy and submitted the file on Kaggle.

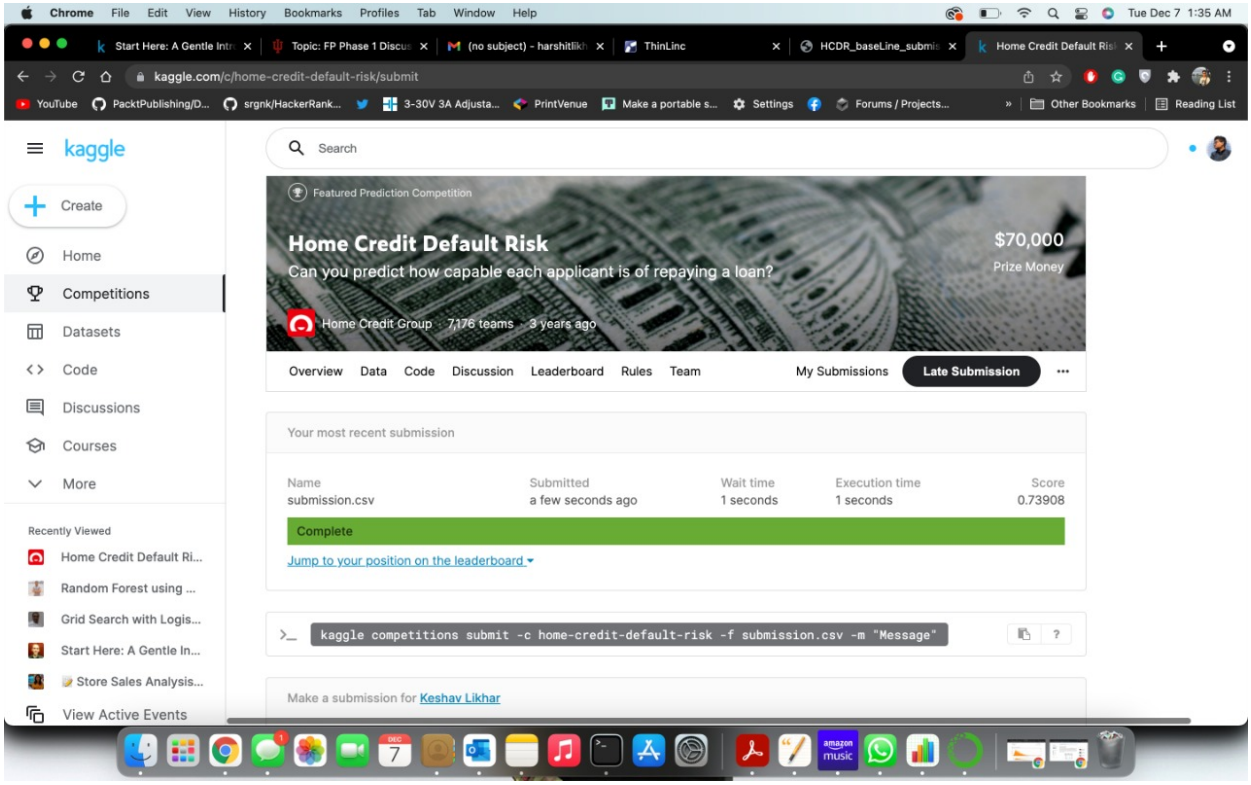
The next steps in our project would be to improve our model performance through implementation of Neural Networks.

)ut[237]:

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Description
0	Baseline 1 LogReg	HCDR	91.91%	91.94%	91.94%	17.1005	0.4038	Baseline 1 LogReg pipeline with Cat+Num features
1	Baseline 1 RandomForest	HCDR	91.91%	91.95%	91.95%	10.7311	0.4899	Baseline 1 RandomForest pipeline with Cat+Num ...
2	GridSearchCV LogReg	HCDR	91.79%	91.99%	91.99%	10.7311	0.2651	GridSearchCV LogReg pipeline with Cat+Num feat...
3	GridSearchCV RandomForestClassifier	HCDR	100.00%	92.09%	92.09%	10.7311	1.2215	GridSearchCV RandomForestClassifier pipeline w...
4	GridSearchCV RandomForestClassifier	HCDR	100.00%	92.09%	92.09%	10.7311	0.2967	GridSearchCV RandomForestClassifier pipeline w...

Kaggle Submission

Please provide a screenshot of your best kaggle submission.
The screenshot should show the different details of the submission and not just the score.



Phase 3 (Neural Net)

In [2]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
```

In [3]:
```
import torchvision
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter("runs/")
```

In [4]:
```
from sklearn.model_selection import train_test_split
```

In [5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [6]:
```
import pandas as pd
```

In [7]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [8]:
```
def corr_target(df,cor):
    cor_matrix = df.corr()['TARGET'].sort_values(key=abs,ascending=False).reset_
```

```
    cor_matrix.columns = ['col_name','Correlation']
    column_after_corr_filter = cor_matrix[abs(cor_matrix['Correlation'])>cor]
    return column_after_corr_filter
```

Data preparation and preprocessing

In [218]:
```
num_pipeline1 = Pipeline([
    ('imputer', SimpleImputer(strategy = 'median'))
])
cat_pipeline1 = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

In [220]:
```
data_pipeline1 = ColumnTransformer([
    ("num_pipeline", num_pipeline1, numerical_df1),
    ("cat_pipeline", cat_pipeline1, categorical_df1)], remainder = 'drop')

phase3_df = data_pipeline1.fit_transform(datasets['application_train'])
```

In [ ]:
```
phase3_df = data_pipeline1.fit_transform(datasets['application_train'])

column_names1 = numerical_df1  + \
                list(data_pipeline1.transformers_[1][1].named_steps["ohe"].get_feature_names(categorical_df1))

pd.DataFrame(phase3_df, columns=column_names1).head()
```

In [221]:
```
phase3_df
```

Out[221]:
```
array([[ 1.00002e+05,  1.00000e+00, -9.46100e+03, ...,  0.00000e+00,
         1.00000e+00,  0.00000e+00],
       [ 1.00003e+05,  0.00000e+00, -1.67650e+04, ...,  0.00000e+00,
         1.00000e+00,  0.00000e+00],
       [ 1.00004e+05,  0.00000e+00, -1.90460e+04, ...,  0.00000e+00,
         1.00000e+00,  0.00000e+00],
       ...,
       [ 4.56253e+05,  0.00000e+00, -1.49660e+04, ...,  0.00000e+00,
         1.00000e+00,  0.00000e+00],
       [ 4.56254e+05,  1.00000e+00, -1.19610e+04, ...,  0.00000e+00,
         1.00000e+00,  0.00000e+00],
       [ 4.56255e+05,  0.00000e+00, -1.68560e+04, ...,  0.00000e+00,
```

In [222]:
```
column_names1 = numerical_df1  + \
                list(data_pipeline1.transformers_[1][1].named_steps["ohe"].get_feature_names(categorical_df1))

pd.DataFrame(phase3_df, columns=column_names1).head()
```

Out[222]:

	SK_ID_CURR	TARGET	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_ID_PUBLISH	FLAG_EMP_PHONE	FLAG_WORK_PHONE	FLAG_PHONE	REGION_RATING_CLIE
0	100002.0	1.0	-9461.0	-637.0	-2120.0	1.0	0.0	1.0	
1	100003.0	0.0	-16765.0	-1188.0	-291.0	1.0	0.0	1.0	
2	100004.0	0.0	-19046.0	-225.0	-2531.0	1.0	1.0	1.0	
3	100006.0	0.0	-19005.0	-3039.0	-2437.0	1.0	0.0	0.0	
4	100007.0	0.0	-19932.0	-3038.0	-3458.0	1.0	0.0	0.0	

5 rows × 176 columns

In [224]:
```
integer_df_merged1= df_final_merged1.select_dtypes(include='int64')
float_df_merged1 = df_final_merged1.select_dtypes(include='float64')
numerical_df_merged1 = list(pd.concat([integer_df_merged1,float_df_merged1], axis=1))
categorical_df_merged1 = list(df_final_merged1.select_dtypes(include='object'))
print(categorical_df_merged1)
print('                                ')
print(numerical_df_merged1)
```

```
['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAM
E_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START', 'ORGA
NIZATION_TYPE', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE']

['SK_ID_CURR', 'TARGET', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_ID_PUBLISH', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLA
G_PHONE', 'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START', 'REG_CITY_NOT_WORK_CITY', 'FLAG_DOCUMENT_3', 'FL
AG_DOCUMENT_6', 'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_16', 'DAYS_CREDIT', 'DAYS_ENDDATE_FACT', 'AMT
_CREDIT_SUM', 'DAYS_CREDIT_UPDATE', 'MONTHS_BALANCE', 'AMT_CREDIT_SUM_LIMIT', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT
_ANNUITY_x', 'REGION_POPULATION_RELATIVE', 'DAYS_REGISTRATION', 'OWN_CAR_AGE', 'CNT_FAM_MEMBERS', 'EXT_SOURCE_1', 'E
XT_SOURCE_2', 'EXT_SOURCE_3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'ENTRANCES_AVG',
'FLOORSMAX_AVG', 'LANDAREA_AVG', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'AMT_REQ_CREDIT_BUREAU_MON
', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'SK_ID_PREV', 'AMT_ANNUITY_y', 'DAYS_FIRST_DUE', 'DAYS_TERMINATION', 'DAYS_INSTALME
NT', 'AMT_INSTALMENT', 'AMT_BALANCE', 'AMT_DRAWINGS_OTHER_CURRENT', 'AMT_PAYMENT_CURRENT', 'CNT_DRAWINGS_ATM_CURRENT
', 'CNT_DRAWINGS_OTHER_CURRENT', 'CNT_DRAWINGS_POS_CURRENT', 'CNT_INSTALMENT_MATURE_CUM', 'DAYS_EMPLOYED_PCT', 'CRED
IT_INCOME_PCT', 'ANNUITY_INCOME_PCT', 'CREDIT_TERM_PCT', 'AMT_BALANCE_PCT', 'AVG_INCOME_EXT_PCT', 'AVG_TOTALINCOME_P
CT']
```

In [225]:
```
data_pipeline_merged1 = ColumnTransformer([
    ("num_pipeline", num_pipeline1, numerical_df_merged1),
    ("cat_pipeline", cat_pipeline1, categorical_df_merged1)])

phase3_df_merged = data_pipeline_merged1.fit_transform(df_final_merged1)
```

In [229]:
```
column_names_merged1 = numerical_df_merged1  + \
              list(data_pipeline_merged1.transformers_[1][1].named_steps["ohe"].get_feature_names(categorical_df_mer

abc3 = pd.DataFrame(phase3_df_merged, columns=column_names_merged1)
abc3.shape
```

Out[229]: (92288, 197)

In [230]:
```
abc3.to_csv('df_for_phase3.csv',index=False)
```

Working on Test Data (Preparing)

In [244]:
```
df_final_test1 = df_final_test.copy()
```

In [245]:
```
integer_df_merged2= df_final_test1.select_dtypes(include='int64')
float_df_merged2 = df_final_test1.select_dtypes(include='float64')
numerical_df_merged2 = list(pd.concat([integer_df_merged2,float_df_merged2], axis=1))
categorical_df_merged2 = list(df_final_test1.select_dtypes(include='object'))
print(categorical_df_merged2)
print('                                  ')
print(numerical_df_merged2)
```

```
['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAM
E_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START', 'ORGA
NIZATION_TYPE', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE']

['SK_ID_CURR', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_ID_PUBLISH', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_PHONE',
'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START', 'REG_CITY_NOT_WORK_CITY', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMEN
T_6', 'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_16', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY_x',
'REGION_POPULATION_RELATIVE', 'DAYS_REGISTRATION', 'OWN_CAR_AGE', 'CNT_FAM_MEMBERS', 'EXT_SOURCE_1', 'EXT_SOURCE_2',
'EXT_SOURCE_3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG
', 'LANDAREA_AVG', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CRE
DIT_BUREAU_YEAR', 'DAYS_CREDIT', 'DAYS_ENDDATE_FACT', 'AMT_CREDIT_SUM', 'DAYS_CREDIT_UPDATE', 'MONTHS_BALANCE', 'AMT
_CREDIT_SUM_LIMIT', 'SK_ID_PREV', 'AMT_ANNUITY_y', 'DAYS_FIRST_DUE', 'DAYS_TERMINATION', 'DAYS_INSTALMENT', 'AMT_INS
TALMENT', 'AMT_BALANCE', 'AMT_DRAWINGS_OTHER_CURRENT', 'AMT_PAYMENT_CURRENT', 'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWI
NGS_OTHER_CURRENT', 'CNT_DRAWINGS_POS_CURRENT', 'CNT_INSTALMENT_MATURE_CUM', 'DAYS_EMPLOYED_PCT', 'CREDIT_INCOME_PCT
', 'ANNUITY_INCOME_PCT', 'CREDIT_TERM_PCT', 'AMT_BALANCE_PCT', 'AVG_INCOME_EXT_PCT', 'AVG_TOTALINCOME_PCT']
```

In [246]:
```
data_pipeline_merged2 = ColumnTransformer([
    ("num_pipeline", num_pipeline1, numerical_df_merged2),
    ("cat_pipeline", cat_pipeline1, categorical_df_merged2)])

phase3_df_merged_test = data_pipeline_merged2.fit_transform(df_final_test1)
```

In [247]:
```
column_names_merged2 = numerical_df_merged2  + \
              list(data_pipeline_merged2.transformers_[1][1].named_steps["ohe"].get_feature_names(categorical_df_mer

abc3_test = pd.DataFrame(phase3_df_merged_test, columns=column_names_merged2)
abc3_test.shape
```

Out[247]: (48797, 198)

In [252]:
```
abc3_test.head()
```

Out[252]:

	SK_ID_CURR	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_ID_PUBLISH	FLAG_EMP_PHONE	FLAG_WORK_PHONE	FLAG_PHONE	REGION_RATING_CLIENT_W_CIT
0	100001.0	-19241.0	-2329.0	-812.0	1.0	0.0	0.0	2.
1	100005.0	-18064.0	-4469.0	-1623.0	1.0	0.0	0.0	2.
2	100013.0	-20038.0	-4458.0	-3503.0	1.0	0.0	0.0	2.
3	100028.0	-13976.0	-1866.0	-4208.0	1.0	0.0	1.0	2.
4	100038.0	-13040.0	-2191.0	-4262.0	1.0	1.0	0.0	2.

5 rows × 198 columns

In [249]:
```
abc3_test.to_csv('dftest_for_phase3.csv',index=False)
```

In [248]:
```
abc3.shape
```

Out[248]: (92288, 197)

In [9]:
```
train_df = pd.read_csv('df_for_phase3.csv')
train_df = train_df.iloc[:,0:]
```

In [10]:

```
train_df.corr()['TARGET'].sort_values(key=abs,ascending=False).reset_index()
```

Out[10]:

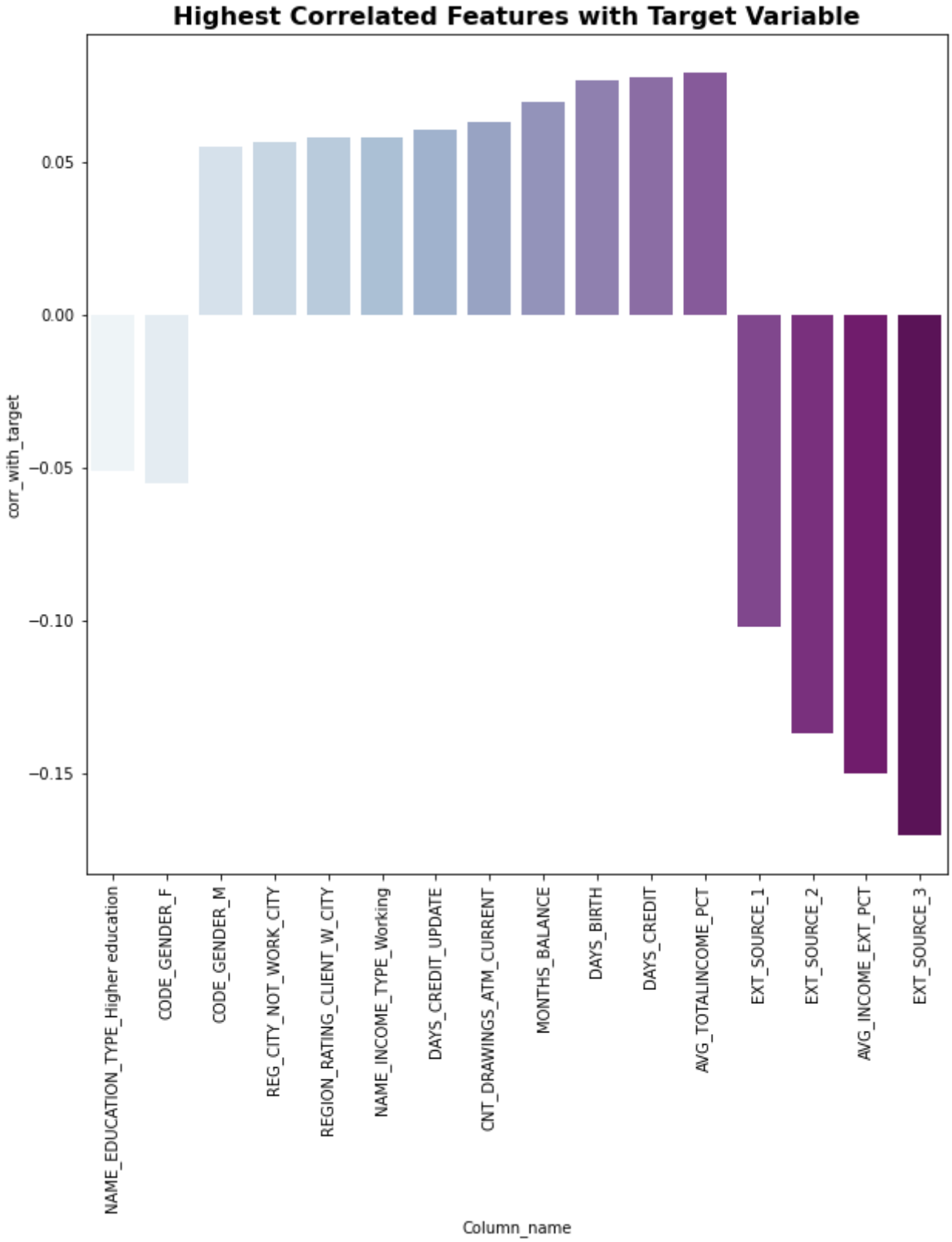
	index	TARGET
0	TARGET	1.000000
1	EXT_SOURCE_3	-0.170023
2	AVG_INCOME_EXT_PCT	-0.149985
3	EXT_SOURCE_2	-0.136448
4	EXT_SOURCE_1	-0.101719
...	...	...
192	ORGANIZATION_TYPE_Services	0.000325
193	HOUSETYPE_MODE_specific housing	0.000253
194	NAME_EDUCATION_TYPE_Incomplete higher	-0.000072
195	ORGANIZATION_TYPE_Telecom	-0.000061
196	NAME_HOUSING_TYPE_Co-op apartment	-0.000004

197 rows × 2 columns

In [11]:
```
target_corr3 = pd.DataFrame(train_df.corrwith(train_df["TARGET"]).sort_values(ke
```

In [12]:
```
target_corr3 = target_corr3.rename(columns={'index':'Column_name',0:'corr_with_t
```

In [13]:
```
plt.figure(figsize=(10,10))
sns.barplot(data=target_corr2,x=target_corr2['Column_name'][180:196], y= target_
plt.xticks(rotation=90)
plt.title('Highest Correlated Features with Target Variable', fontweight = 'bold
plt.show()
```



In [14]:
```
train_df_cols = corr_target(train_df,0)
```

In [15]:
```
train_df = train_df[train_df_cols['col_name']]
```

In [16]:
```
train_df.shape
```

Out[16]:
```
(92288, 197)
```

In [17]:
```
X = train_df.drop('TARGET',axis=1).values
y = train_df['TARGET'].values
```

In [18]:
```
y1 = y.astype('int32')
y1
```

Out[18]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int32)
```

In [19]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y1,test_size=0.2,random_state
```

In [20]:
```
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.tensor(y_train, dtype=torch.long, device=device)
y_test = torch.tensor(y_test, dtype=torch.long, device=device)
```

In [28]:
```
class ANN_model(nn.Module):
    def __init__(self,input_features=196, hidden1=64, hidden2=64,hidden3=32,hidd
        super().__init__()
        self.f_connected1 = nn.Linear(input_features,hidden1)
        self.f_connected2 = nn.Linear(hidden1,hidden2)
        self.f_connected3 = nn.Linear(hidden2,hidden3)
        self.f_connected4 = nn.Linear(hidden3,hidden4)
        self.f_connected5 = nn.Linear(hidden4,hidden5)
        self.out = nn.Linear(hidden5,out_features)
    def forward(self,x):
        x = F.leaky_relu(self.f_connected1(x))
        x = F.leaky_relu(self.f_connected2(x))
        x = F.leaky_relu(self.f_connected3(x))
        x = F.leaky_relu(self.f_connected4(x))
        x = F.leaky_relu(self.f_connected5(x))
        x = self.out(x)
        return x
```

In [29]:
```
# class ANN_model(nn.Module):
#     def __init__(self,input_features=10
#                  , hidden1=64, hidden2=32,hidden3=16,out_features=2):
#         super().__init__()
#         self.f_connected1 = nn.Linear(input_features,hidden1)
#         self.f_connected2 = nn.Linear(hidden1,hidden2)
#         self.f_connected3 = nn.Linear(hidden2,hidden3)
#         self.out = nn.Linear(hidden3,out_features)
#     def forward(self,x):
#         x = F.leaky_relu(self.f_connected1(x))
#         x = F.leaky_relu(self.f_connected2(x))
#         x = F.leaky_relu(self.f_connected3(x))
#         x = self.out(x)
#         return x
```

In [30]:

```
torch.manual_seed(20)
model= ANN_model().to(device)
```

In [31]:
```
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(),lr=0.001)
```

In [32]:
```
running_correct = 0
```

In [33]:
```
size_ = y_train.shape[0]
```

In [45]:
```
import time
import numpy as np
```

In [65]:
```
epochs = 1000
final_losses = []
start = time.time()
for i in range(epochs):
    i += 1
    y_pred = model.forward(X_train)
    loss = loss_function(y_pred,y_train)
    final_losses.append(loss.item())
    _,predicted = torch.max(y_pred,1)
    running_correct = (predicted==y_train).sum().item()
    if i%100 == 1:
        print("Epoch Number: {} and the loss: {} accuracy : {}".format(i,loss.it
        writer.add_scalar('Training loss',loss.item(),i)
        writer.add_scalar('Accuracy',running_correct/size_,i)
        train_acc = running_correct/size_
        running_correct = 0
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
train_time = np.round(time.time()-start,2)
```

```
Epoch Number: 1 and the loss: 0.26456618309020996 accuracy : 0.9182852498984153
Epoch Number: 101 and the loss: 0.2651815414428711 accuracy : 0.9183394284166329
Epoch Number: 201 and the loss: 0.26538515090942383 accuracy : 0.918312339157524
1
Epoch Number: 301 and the loss: 0.26524993777275085 accuracy : 0.918393606934850
3
Epoch Number: 401 and the loss: 0.2639715373516083 accuracy : 0.918420696193959
Epoch Number: 501 and the loss: 0.2640071213245392 accuracy : 0.9185019639712854
Epoch Number: 601 and the loss: 0.2632846236228943 accuracy : 0.918488419341731
Epoch Number: 701 and the loss: 0.2630040943622589 accuracy : 0.9185561424895029
Epoch Number: 801 and the loss: 0.2632701098918915 accuracy : 0.9185696871190573
Epoch Number: 901 and the loss: 0.262630432844162 accuracy : 0.9186374102668292
```

In [47]:
```
writer.add_graph(model,X_train)
writer.close()
```

In [48]:
```
# plt.plot(range(epochs),final_losses)
```

```
# plt.ylabel('Loss')
# plt.xlabel('Epoch')
```

In [49]:
```
predictions = []
probs = []
with torch.no_grad():
    for i,data in enumerate(X_test):
        y_pred = model(data)
#         print(F.softmax(y_pred))
        predictions.append(y_pred.argmax().item())
#         print(y_pred.argmax().item())
```

In [50]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,predictions)
cm
```

Out[50]:
```
array([[16980,    17],
       [ 1454,     7]])
```

In [51]:
```
from sklearn.metrics import accuracy_score, roc_auc_score
score = accuracy_score(y_test,predictions)
score
```

In [74]:
```
try: experimentLog3
except : experimentLog3 = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc
                                                "Kaggle Score", "Train Time(s)",
experimentLog3.loc[len(experimentLog3)] =[f"MLP Pytorch", "HCDR",
                                          f"{train_acc*100:8.2f}%", f"{score*100:8
                                          f"{score*100:8.2f}%",
                                          f"{roc:8.2f}",
                                            f"{0.71}",
                                          train_time, test_time,
                                          "Neural net Implementation"]
experimentLog3
```

Out[74]:

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	roc_auc_score	Kaggle Score	Train Time(s)	Test Time(s)	D
0	MLP Pytorch	HCDR	91.86%	92.03%	92.03%	0.73	0.71	85.73	9.62	Impl

Preparing the test Data

In [53]:
```
test_df = pd.read_csv('dftest_for_phase3.csv')
test_df = test_df.iloc[:,0:]
```

In [54]:
```
test_cols = list(train_df.columns)
test_cols.remove('TARGET')
```

In [55]:

```
test_cols
```

Out[55]:

```
['EXT_SOURCE_3',
 'AVG_INCOME_EXT_PCT',
 'EXT_SOURCE_2',
 'EXT_SOURCE_1',
 'AVG_TOTALINCOME_PCT',
 'DAYS_CREDIT',
 'DAYS_BIRTH',
 'MONTHS_BALANCE',
 'CNT_DRAWINGS_ATM_CURRENT',
 'DAYS_CREDIT_UPDATE',
 'NAME_INCOME_TYPE_Working',
 'REGION_RATING_CLIENT_W_CITY',
 'REG_CITY_NOT_WORK_CITY',
 'CODE_GENDER_F',
 'CODE_GENDER_M',
 'NAME_EDUCATION_TYPE_Higher education',
 'FLAG_EMP_PHONE',
 'NAME_INCOME_TYPE_Pensioner',
 'ORGANIZATION_TYPE_XNA',
 'DAYS_ENDDATE_FACT',
 'DAYS_EMPLOYED',
 'DAYS_EMPLOYED_PCT',
 'NAME_EDUCATION_TYPE_Secondary / secondary special',
 'DAYS_ID_PUBLISH',
 'DAYS_REGISTRATION',
 'AMT_BALANCE_PCT',
 'FLOORSMAX_AVG',
 'AMT_BALANCE',
 'REGION_POPULATION_RELATIVE',
 'FLAG_DOCUMENT_3',
 'FLAG_DOCUMENT_6',
 'OCCUPATION_TYPE_Low-skill Laborers',
 'CNT_DRAWINGS_POS_CURRENT',
 'NAME_HOUSING_TYPE_With parents',
 'NAME_HOUSING_TYPE_House / apartment',
 'ORGANIZATION_TYPE_Self-employed',
 'DEF_60_CNT_SOCIAL_CIRCLE',
 'APARTMENTS_AVG',
 'OCCUPATION_TYPE_Drivers',
 'FLAG_PHONE',
 'AMT_CREDIT',
 'ORGANIZATION_TYPE_Business Entity Type 3',
 'FLAG_WORK_PHONE',
 'OCCUPATION_TYPE_Accountants',
 'DAYS_INSTALMENT',
 'NAME_HOUSING_TYPE_Rented apartment',
 'NAME_FAMILY_STATUS_Single / not married',
 'AMT_CREDIT_SUM',
 'FONDKAPREMONT_MODE_reg oper account',
 'HOUR_APPR_PROCESS_START',
 'BASEMENTAREA_AVG',
 'OCCUPATION_TYPE_Sales staff',
 'NAME_INCOME_TYPE_State servant',
 'OWN_CAR_AGE',
 'NAME_FAMILY_STATUS_Widow',
 'CNT_DRAWINGS_OTHER_CURRENT',
 'NAME_FAMILY_STATUS_Married',
 'OCCUPATION_TYPE_Core staff',
```

```
'FLAG_OWN_CAR_N',
'FLAG_OWN_CAR_Y',
'ORGANIZATION_TYPE_Construction',
'NAME_FAMILY_STATUS_Civil marriage',
'OCCUPATION_TYPE_Managers',
'AMT_INCOME_TOTAL',
'CNT_INSTALMENT_MATURE_CUM',
'NAME_EDUCATION_TYPE_Lower secondary',
'OCCUPATION_TYPE_Cleaning staff',
'OCCUPATION_TYPE_High skill tech staff',
'ANNUITY_INCOME_PCT',
'ORGANIZATION_TYPE_Restaurant',
'WALLSMATERIAL_MODE_Panel',
'CNT_FAM_MEMBERS',
'FONDKAPREMONT_MODE_reg oper spec account',
'OCCUPATION_TYPE_Waiters/barmen staff',
'ORGANIZATION_TYPE_Trade: type 3',
'OCCUPATION_TYPE_Security staff',
'FONDKAPREMONT_MODE_org spec account',
'ENTRANCES_AVG',
'ORGANIZATION_TYPE_Transport: type 3',
'FLAG_OWN_REALTY_N',
'FLAG_OWN_REALTY_Y',
'AMT_REQ_CREDIT_BUREAU_YEAR',
'FLAG_DOCUMENT_16',
'ORGANIZATION_TYPE_School',
'NAME_INCOME_TYPE_Commercial associate',
'ORGANIZATION_TYPE_Bank',
'OCCUPATION_TYPE_Cooking staff',
'NAME_CONTRACT_TYPE_Cash loans',
'NAME_CONTRACT_TYPE_Revolving loans',
'CREDIT_INCOME_PCT',
'ORGANIZATION_TYPE_Military',
'AMT_PAYMENT_CURRENT',
'WALLSMATERIAL_MODE_Stone, brick',
'ORGANIZATION_TYPE_Government',
'WALLSMATERIAL_MODE_Monolithic',
'LANDAREA_AVG',
'ORGANIZATION_TYPE_Industry: type 12',
'FONDKAPREMONT_MODE_not specified',
'ORGANIZATION_TYPE_Security',
'ORGANIZATION_TYPE_Industry: type 9',
'FLAG_DOCUMENT_14',
'AMT_DRAWINGS_OTHER_CURRENT',
'ORGANIZATION_TYPE_Industry: type 3',
'WALLSMATERIAL_MODE_Block',
'FLAG_DOCUMENT_13',
'ORGANIZATION_TYPE_Industry: type 1',
'CREDIT_TERM_PCT',
'WEEKDAY_APPR_PROCESS_START_TUESDAY',
'ORGANIZATION_TYPE_Trade: type 2',
'OCCUPATION_TYPE_Medicine staff',
'ORGANIZATION_TYPE_University',
'ORGANIZATION_TYPE_Industry: type 4',
'AMT_INSTALMENT',
'OBS_60_CNT_SOCIAL_CIRCLE',
'ORGANIZATION_TYPE_Culture',
'WEEKDAY_APPR_PROCESS_START_MONDAY',
'ORGANIZATION_TYPE_Police',
'NAME_TYPE_SUITE_Unaccompanied',
```

```
'ORGANIZATION_TYPE_Hotel',
'WEEKDAY_APPR_PROCESS_START_FRIDAY',
'ORGANIZATION_TYPE_Security Ministries',
'ORGANIZATION_TYPE_Medicine',
'NAME_TYPE_SUITE_Family',
'ORGANIZATION_TYPE_Cleaning',
'OCCUPATION_TYPE_Laborers',
'ORGANIZATION_TYPE_Trade: type 6',
'OCCUPATION_TYPE_Private service staff',
'AMT_REQ_CREDIT_BUREAU_MON',
'ORGANIZATION_TYPE_Business Entity Type 2',
'NAME_TYPE_SUITE_Children',
'AMT_ANNUITY_x',
'ORGANIZATION_TYPE_Trade: type 7',
'ORGANIZATION_TYPE_Agriculture',
'ORGANIZATION_TYPE_Trade: type 4',
'ORGANIZATION_TYPE_Industry: type 2',
'YEARS_BEGINEXPLUATATION_AVG',
'AMT_CREDIT_SUM_LIMIT',
'ORGANIZATION_TYPE_Legal Services',
'ORGANIZATION_TYPE_Business Entity Type 1',
'WALLSMATERIAL_MODE_Wooden',
'NAME_HOUSING_TYPE_Office apartment',
'ORGANIZATION_TYPE_Industry: type 7',
'WEEKDAY_APPR_PROCESS_START_SATURDAY',
'NAME_EDUCATION_TYPE_Academic degree',
'ORGANIZATION_TYPE_Trade: type 5',
'ORGANIZATION_TYPE_Transport: type 1',
'ORGANIZATION_TYPE_Industry: type 13',
'SK_ID_PREV',
'ORGANIZATION_TYPE_Electricity',
'OCCUPATION_TYPE_IT staff',
'ORGANIZATION_TYPE_Transport: type 4',
'OCCUPATION_TYPE_HR staff',
'ORGANIZATION_TYPE_Insurance',
'SK_ID_CURR',
'WALLSMATERIAL_MODE_Others',
'ORGANIZATION_TYPE_Trade: type 1',
'ORGANIZATION_TYPE_Other',
'NAME_TYPE_SUITE_Other_B',
'WEEKDAY_APPR_PROCESS_START_WEDNESDAY',
'ORGANIZATION_TYPE_Religion',
'NAME_TYPE_SUITE_Other_A',
'NAME_INCOME_TYPE_Student',
'WALLSMATERIAL_MODE_Mixed',
'NAME_FAMILY_STATUS_Separated',
'DAYS_TERMINATION',
'ORGANIZATION_TYPE_Housing',
'OCCUPATION_TYPE_Realty agents',
'ORGANIZATION_TYPE_Kindergarten',
'ORGANIZATION_TYPE_Postal',
'ORGANIZATION_TYPE_Realtor',
'ORGANIZATION_TYPE_Industry: type 8',
'NAME_HOUSING_TYPE_Municipal apartment',
'OCCUPATION_TYPE_Secretaries',
'AMT_ANNUITY_y',
'ORGANIZATION_TYPE_Industry: type 6',
'ORGANIZATION_TYPE_Transport: type 2',
'ORGANIZATION_TYPE_Industry: type 10',
'DAYS_FIRST_DUE',
```

```
'WEEKDAY_APPR_PROCESS_START_SUNDAY',
'NAME_TYPE_SUITE_Group of people',
'EMERGENCYSTATE_MODE_No',
'EMERGENCYSTATE_MODE_Yes',
'ORGANIZATION_TYPE_Industry: type 11',
'NAME_TYPE_SUITE_Spouse, partner',
'ORGANIZATION_TYPE_Industry: type 5',
'ORGANIZATION_TYPE_Mobile',
'HOUSETYPE_MODE_terraced house',
'ORGANIZATION_TYPE_Emergency',
'HOUSETYPE_MODE_block of flats',
'ORGANIZATION_TYPE_Advertising',
'WEEKDAY_APPR_PROCESS_START_THURSDAY',
'ORGANIZATION_TYPE_Services',
'HOUSETYPE_MODE_specific housing',
'NAME_EDUCATION_TYPE_Incomplete higher',
'ORGANIZATION_TYPE_Telecom',
'NAME_HOUSING_TYPE_Co-op apartment']
```

In [56]:
```
testing = torch.FloatTensor(test_df[test_cols].values)
```

In [57]:
```
predictions = []
probs = []
start1 = time.time()
with torch.no_grad():
    for i,data in enumerate(testing):
        y_pred = model(data)
#         print(F.softmax(y_pred)[1].item())
        probs.append(F.softmax(y_pred)[1].item())
        predictions.append(y_pred.argmax().item())
test_time = np.round(time.time()-start1,2)
#         print(y_pred.argmax().item())
```

```
<ipython-input-57-5c87b321f4ff>:8: UserWarning: Implicit dimension choice for so
ftmax has been deprecated. Change the call to include dim=X as an argument.
  probs.append(F.softmax(y_pred)[1].item())
```

Write-up Phase3

FP Phase 3 - Final Project HCDR - Pytorch/Deep Learning

PROJECT MEMBERS:

ADITI MULYE - adimulye@iu.edu

KESHAV LIKHAR - klikhar@iu.edu

NIKUNJ MALPANI - nmalpani@iu.edu

PRASHASTI KARLEKAR - prkarl@iu.edu

Aditi Mulye
adimulye@iu.edu

Keshav Likhar
klikhar@iu.edu

Nikunj Malpani
nmalpani@iu.edu

Prashasti Karlekar
prkarl@iu.edu

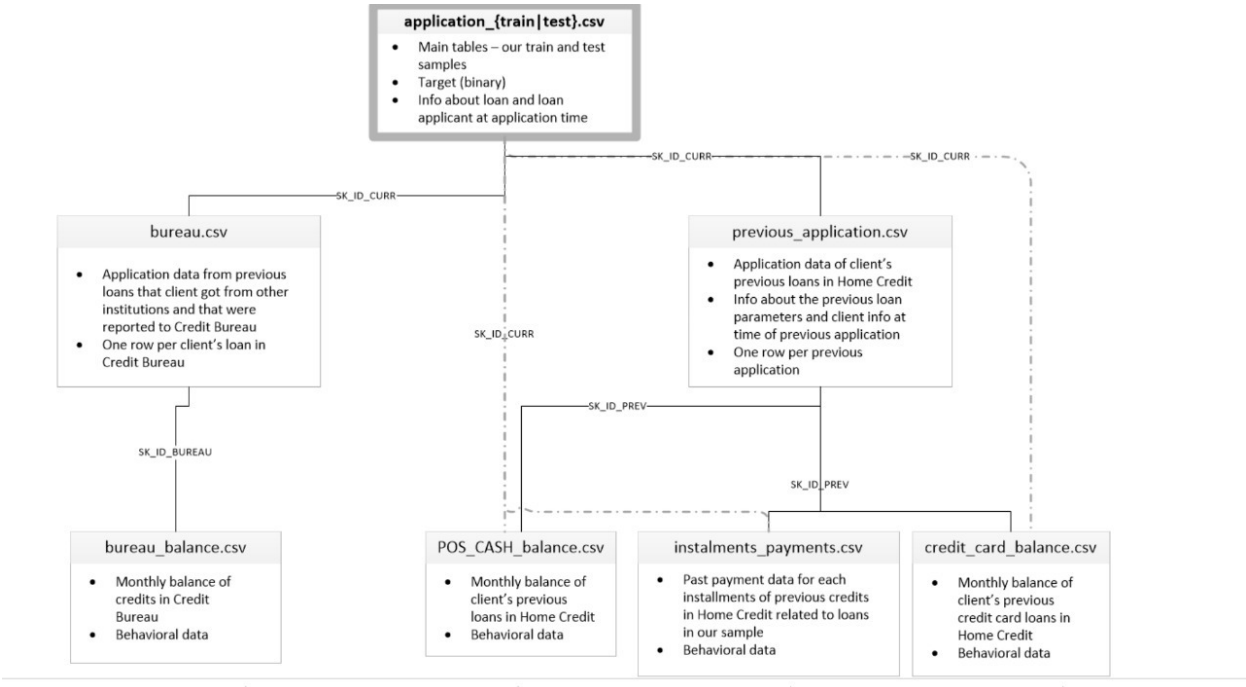
ABSTRACT

In this phase of the project, our aim is to extend and build on our previous observations from Phase 1 and Phase 2 with the implementation of a deep learning model. In the previous phases, we had performed exploratory data analysis and pre-processing of the dataset, along with Feature Engineering and Hyperparameter tuning. We have chosen a multi-layer perceptron (MLP) model, which is the best choice when it comes to modeling business problems. Also, we employed Pytorch for implementing Neural Network for our model using only the top 26 features that we found out in Phase 2. These top features were found out on the basis of their importance and correlation with the target variable. We have used One-hot Encoder and Simple Imputer on our selected features before passing them through the Neural network. Finally, we created the neural network using 1000 epochs with 4 hidden layers and Rectified Linear Unit (ReLu) as activation function. Furthermore, we have used TensorBoard to visualize important metrics and highlight potential issues.

- We used Pytorch to build a MLP model and built an Artificial Neural Network with 5 hidden layers consisting of 128, 64, 32, 10,5 neurons in each layer respectively with an output layer having 2 neurons
- The activation function that we used was the leaky relu activation function which is an upgrade on the relu activation function having more resilience to the vanishing gradient problem
- The loss function that we used for this model is the cross entropy loss and the optimizer (after experimenting different optimizers) we used for modeling is the Adam optimizer
- The learning rate (after trying different learning rates) that we decided for this particular model was 0.001 and a dropout layer with dropout rate of 0.5 was added in the network to regularize the overfitting of model
- Lastly, the neural network model was trained for 1000 epochs and a softmax function was applied on the final predictions to get the final answer in the form of probabilities

PROJECT DESCRIPTION

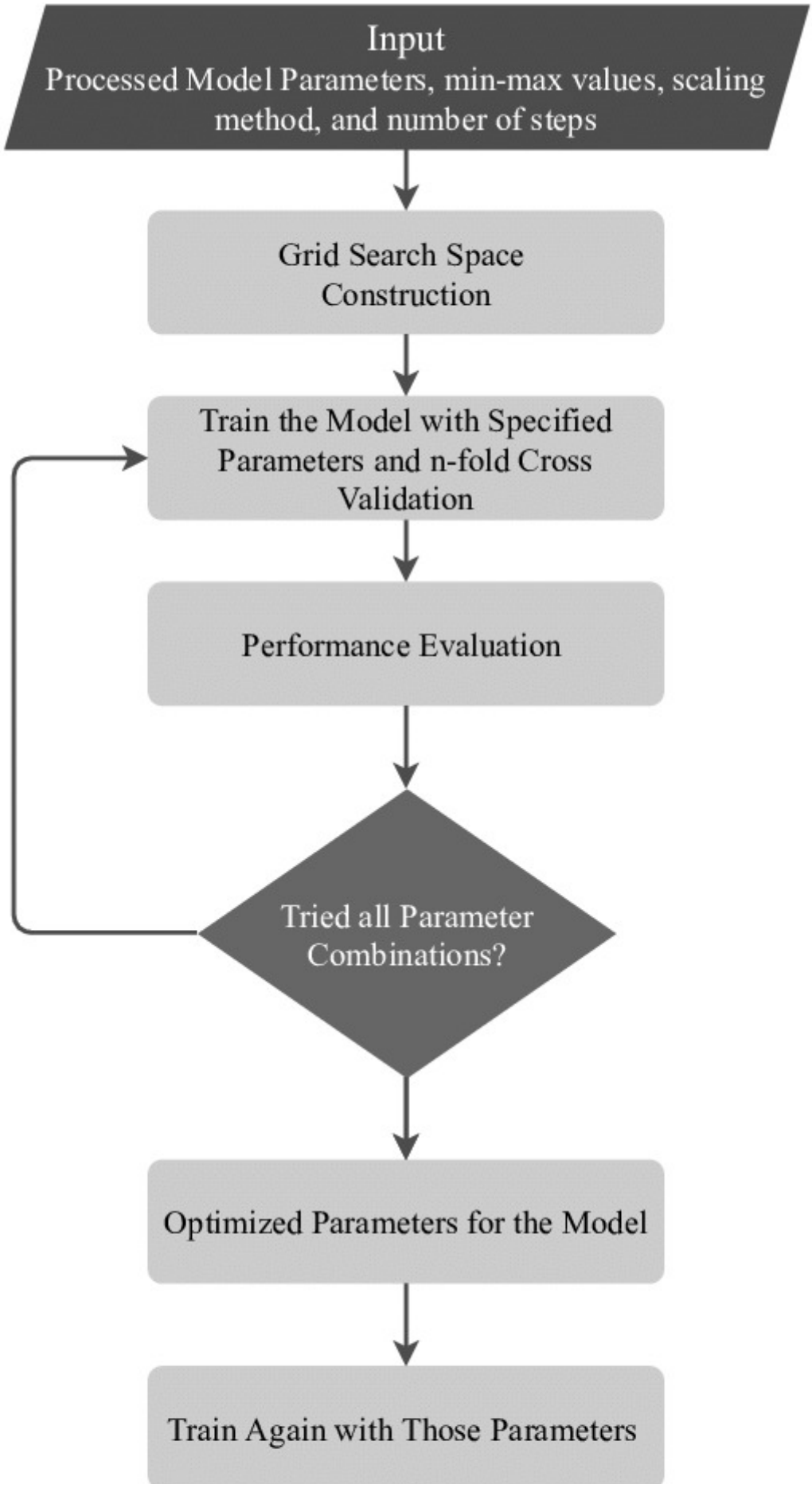
PROJECT DESCRIPTION Data description The home-credit data is currently available on Kaggle for predicting whether or not a client will repay a loan or have difficulty, which is a critical business need. In this dataset we have a total of 122 columns. There are a lot of columns which have missing values in the dataset. As we can already see that there are so many missing values in the top 10 rows itself. We'll have to figure out a way to deal with this! There are 7 different sources of data: application_train/application_test: the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating 0: the loan was repaid or 1: the loan was not repaid. bureau: data concerning client's previous credits from other financial institutions. Each previous credit has its own row in the bureau, but one loan in the application data can have multiple previous credits. bureau_balance: monthly data about the previous credits in the bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length. previous_application: previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV. POS_CASH_BALANCE: monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows. credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows. installments_payment: payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment. This diagram shows how all of the data is related.

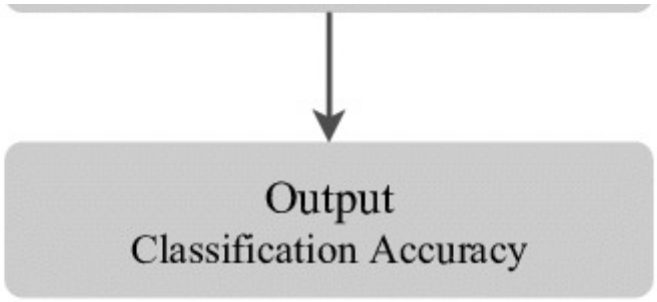


Tasks to be tackled:

- Implement Neural Network (NN) model using PyTorch

- Visualizing the training model in TensorBoard
- Finding and mitigating the leakage in the model pipelines





Neural Network/PyTorch (HCDR)

Deep learning is concerned with automatically generating representations from raw data in order to complete a task. Filters are improved during training in the ones versus zeros example by iteratively examining pairs of samples and target labels. Deep learning is achieved by a neural network's ability to consume input and extract usable representations from instances. The implementation of a deep neural network (DNN) necessitates careful network building. The following are the main steps in building the network: Defining the neural network's architecture, as well as its input and output The training sets are used to teach the network to imitate the intended problem or task. Before considering it for final usage, test the network's performance.

A typical neural network is made up of many simple, connected processors called neurons, each of which generates a series of real-valued activations. Sensors detecting the environment activate input neurons, whereas weighted connections from previously active neurons stimulate additional neurons. With the help of unsupervised learning, deep learning models become more practical to some extent. Multiple layers are added to a neural network in deep learning procedures, though these layers might be repeated. In order to discuss the subject, most deep learning algorithms rely on the four types of architectures listed below: I convolutional neural networks, which are standard neural networks with shared weights that have been extended across space; (ii) recurrent neural networks, which are standard neural networks with edges that feed into the next time step instead of the next layer in the same time step. (iii) recursive neural networks, which are hierarchical networks in which the input sequence has no time dimension but inputs must be processed hierarchically as in a tree; and (iv)The multilayer perceptron provides a nonlinear mapping between an input vector and an output vector. The major use cases of MLP are pattern classification, recognition, prediction and approximation. The MLP model is the best choice when it comes to modeling business problems, and thus, we have used MLP for our model.

The architecture of the neural network model

Following the selection of the model, it should be instanced with certain parameters (number of inputs, outputs, and hidden layers). The number of neurons that make up the input. There are no guidelines for making a decision. Even though it is challenging and time consuming, we employ trial and error until we obtain what we want. Finally, we arrive at the optimal number of input neurons, which is 64. The number of layers that are hidden. Most of the time, one, two, or even

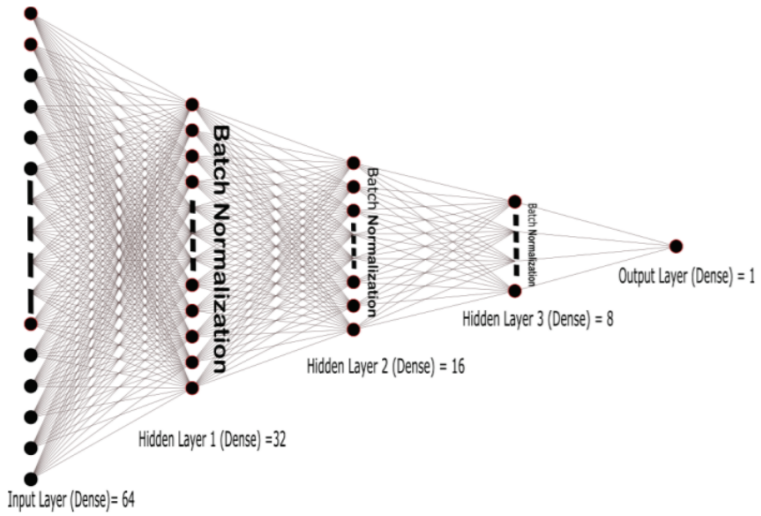
three hidden layers are sufficient, and increasing the number of layers raises the risk of overfitting. Because the number of hidden layers is difficult to estimate, it is necessary to train networks with varying numbers of hidden layers and compare them. We settle on three hidden layers as an optimal number solution at the end of our loop.

The number of neurons. There are no clear guidelines for determining the optimal number of hidden neurons. However, we must always keep in mind that a large number is resource-intensive, and a small number may not reflect the full range of input data. Finally, we chose to preserve the network that performed well on the testing set, so we set the number of hiding neurons to 56, spread among three hidden layers, each with half the number of the previous layer.

The number of output neurons. The number of output neurons is always determined by the problem; in multiclass problems, numerous neurons may be used. However, because we are dealing with a binary classification problem (whether the client is having trouble repaying the loan or not),(we just employed one output neuron. NEEDS TO BE UPDATED)

In neural networks, many variations of structures are available, selecting the appropriate form and associated parameters, can be more of an art than a Science.

> The architecture of our neural network model



$$Output = Activiation\,(Dot(input, kernel) + bias)$$

where, Input represents the input data

Kernel represents the weight data

The dot represents a scalar product, which is an algebraic operation that takes two equal-length sequences of numbers and returns a single number, the operation applies a scalar product of all

input and its corresponding weights.

Bias represents a biased value that is used to optimize the model.

The input layer contains 64 neurons, and each of the hidden levels contains half as many neurons as the previous layer, resulting in a pyramid structure.

What is PyTorch and Why are we using it

PyTorch is an open-source machine learning Python library used for deep learning implementations like computer vision (using TorchVision) and natural language processing. It was developed by Facebook's AI research lab (FAIR) in 2016 and has since been adopted across the fields of data science and ML.

PyTorch makes machine learning intuitive for those already familiar with Python and has great features like OOP support and dynamic computation graphs.

Along with building deep neural networks, PyTorch is also great for complicated mathematical computations because of its GPU acceleration. This feature allows PyTorch to use your computer's GPU to massively speed up computations.

The neural network model parameters

For a given neural network, the inputs are multiplied by the weights in a neuron and summed together. This value is then transformed via an activation function. We used Leaky ReLU, and we use it for input and hiding layers. The ReLU stands for a rectified linear unit, and it's an activation function that can overcome the vanishing gradient problem, allowing models to learn faster and perform better. It's the most commonly used activation function in multilayer neural networks. Mathematically, it is defined as y = max (0, x). we opted to use this function in the input and the hidden layer because it considered a non-saturated function. Which means that the limit of the function tends to the infinity.

Training the neural network

The process of adapting a neural network to make predictions from data is known as training. The most difficult aspect of network construction is the training phase. The training procedure typically begins with the collection of training instances into two big matrices. The inputs are in one matrix, and the outputs are in the other. The network parameters are then configured. Finally, the network is built and trained to forecast the output based on the input. Our neural network model is summarized in the next graph.

Leakage (HCDR)

-- Go through your Pipeline and check if there is any leakage. A method based on Artificial Neural Networks (ANN) is suggested for detecting and diagnosing multiple leaks in a pipeline using only two observations to recognize the flow pattern. The ANN-based system is trained,

tested, and validated using a nonlinear mathematical model of the pipeline. To account for system dynamics, this system was trained with tapped delays. Early results show that the method is effective in detecting and diagnosing many issues at the same time.

The Cardinal Sins of Machine Learning are:-

1. Blindly increasing the number of epochs when the model is not converging

There are times during model training when the loss-epoch graph keeps bouncing about and does not appear to converge, regardless of the number of epochs. There is no silver bullet because there are several core issues to look into: faulty training examples, missing truths, changing data distributions, and a learning rate that is too high. Bad training instances involving a combination of aberrant data and inaccurate labeling are the most typical one I've encountered.

1. In multiclass classification, not prioritizing specific per-class metrics accuracy

Instead of tracking overall classification accuracy for multiclass prediction problems, it is often more advantageous to focus the accuracy of specific classes and iteratively improve the model class by class. Focus on boosting the recall of specific classes (such as foreign transactions) based on business needs, for example, when classifying different types of fraudulent transactions.

1. Overinterpretation of Results

Every claim should be based on facts. You can hypothesize on the universal applicability of your method in discussions clearly indicating the speculation, but to actually claim this you have to provide either experimental or theoretical evidence.

1. Ignoring prediction bias

The disparity between the averages of predictions and labels in a dataset is known as prediction bias. Prediction bias can be used to detect model flaws early on. A large nonzero prediction bias suggests that the model has a flaw. In the topic of ad CTR, there's an intriguing Facebook paper. The bias is commonly used to compare prediction buckets.

1. Calling it a success just on model accuracy numbers

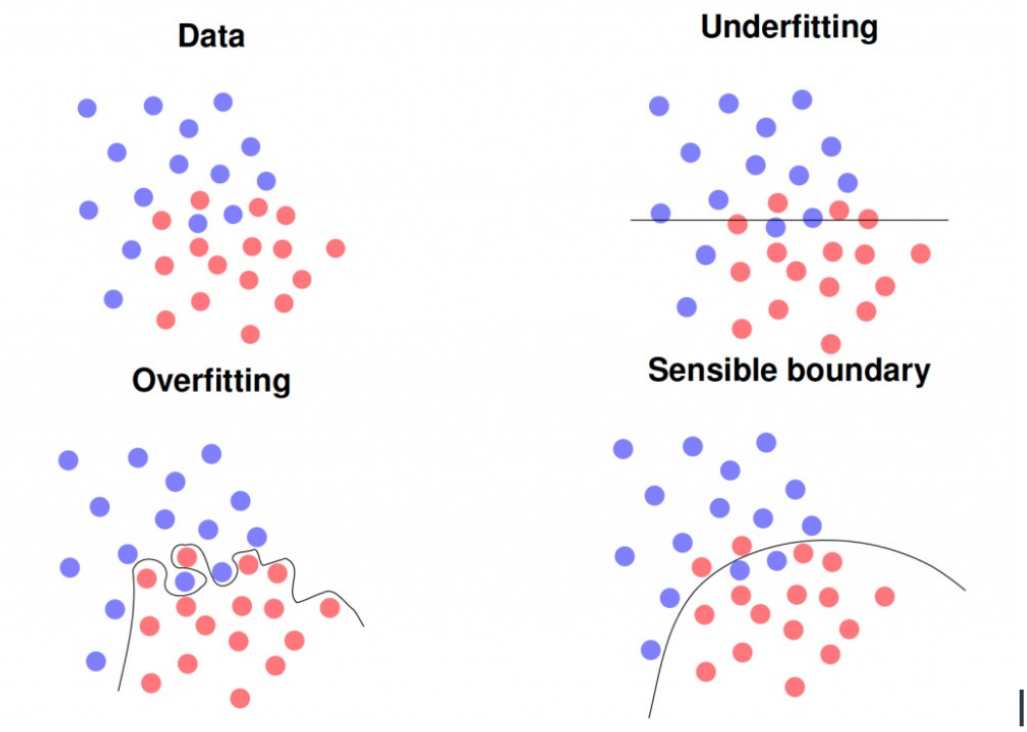
Accuracy of 95% means 95 of 100 predictions were correct. Accuracy is a flawed metric with a class imbalance in the dataset. Instead investigate deeply into metrics, such as precision/recall and how it correlates to overall user metrics such as spam detection, tumor classification, etc.

1. Not understanding the impact of regularization lambda

Lambda is a key parameter in striking the balance between simplicity and training-data fit. High lambda → simple model → possibly underfitting. Low lambda → complex model → potential overfitting your data (won't be able to generalize to new data). The ideal value of lambda is one that generalizes well to previously unseen data: data-dependent and requires analysis.

1. Not paying attention to initiation value in neural networks

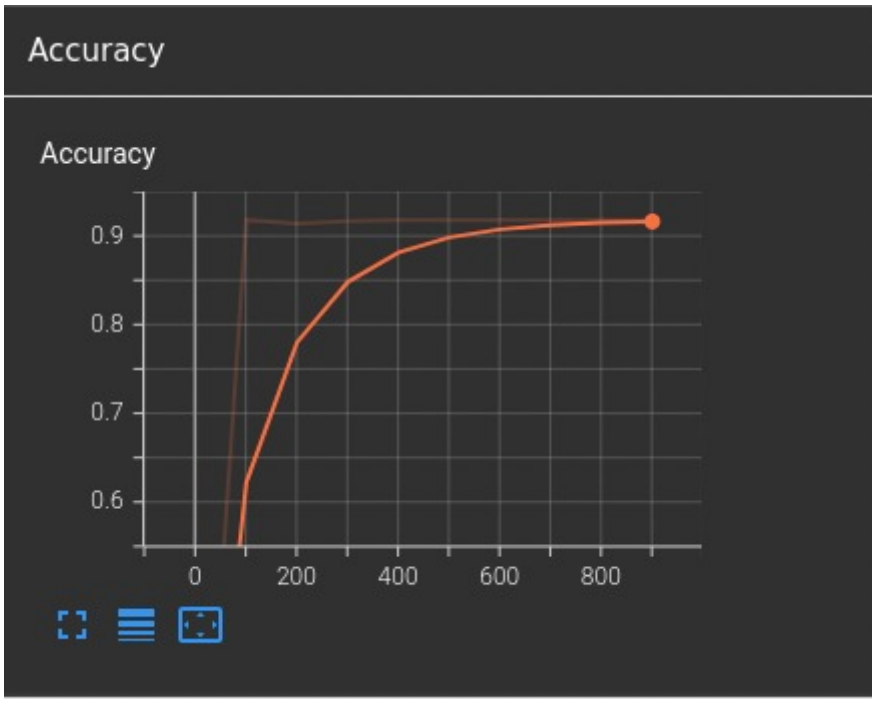
With simple classifiers such as the nearest neighbor, this immediately leads to a 100% recognition rate for most problems. In more sophisticated and deep models, it may not be 100%, but 98–99% accuracy. Hence, you should always scrutinize your experimental setup if you achieve such high recognition ratesin your first shot. If you go to new data, however, your model will completely break and you may even produce results that are worse than random guessing, i.e. lower accuracies than 1/K where K is the number of classes, e.g. less than 50% in a two-class problem. In the same line, you can also easily overfit your model by increasing the number of parameters such that it completely memorizes the training data set. Another variant is using a too-small training set that is not representative of your application. All these models are likely to break on new data, i.e. when employed in a real application scenario.

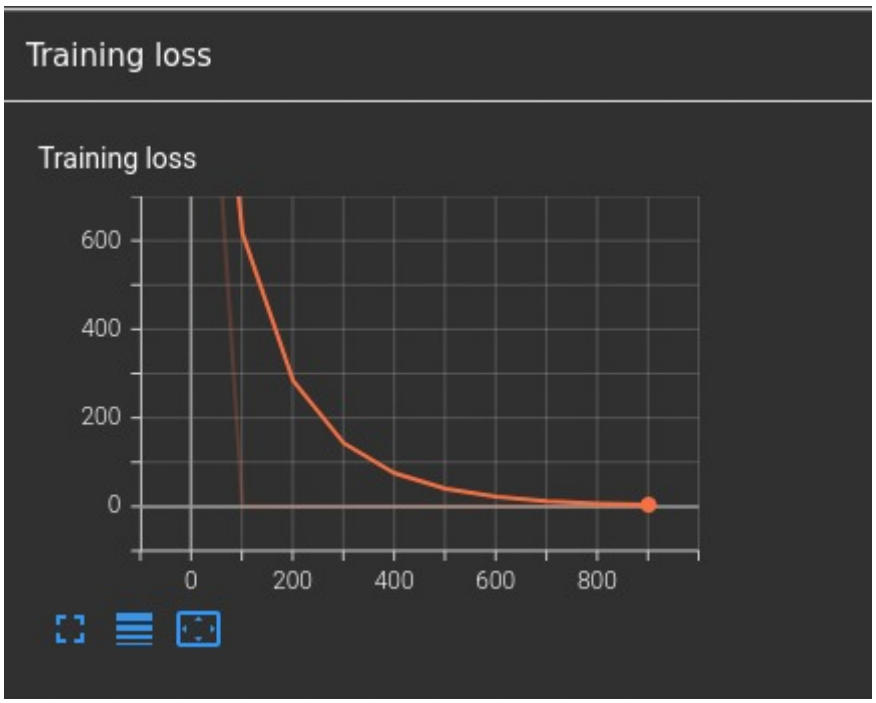


1. Assuming wrong labels always need to be fixed When wrong labels are detected, it is tempting to jump in and get them fixed. It is important to first analyze misclassified examples for the root cause. Oftentimes, errors due to incorrect labels may be a very small percentage. There might be a bigger opportunity to better train for specific data slices that might be the predominant root cause.

Results and Discussions

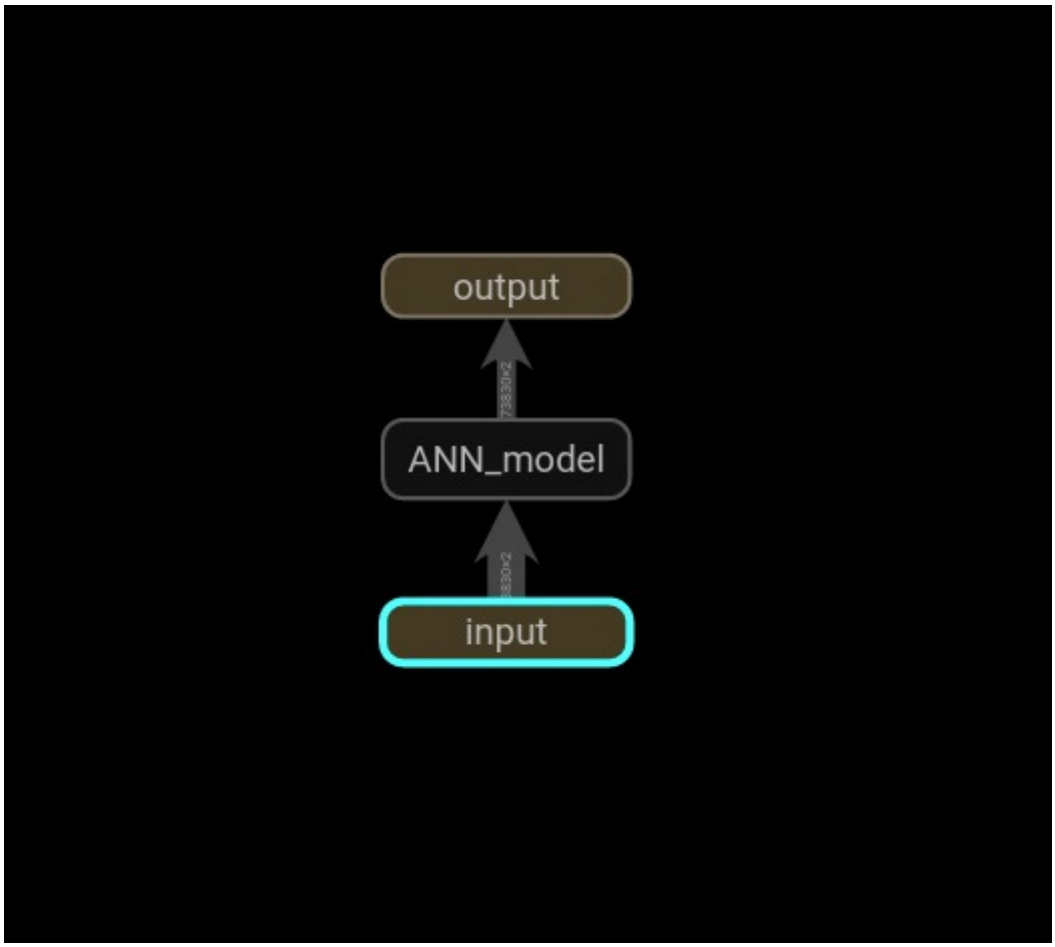
Below are the screenshots of our results we obtained in the results. Also we have shown our model that has been implemented on Tensorboard.

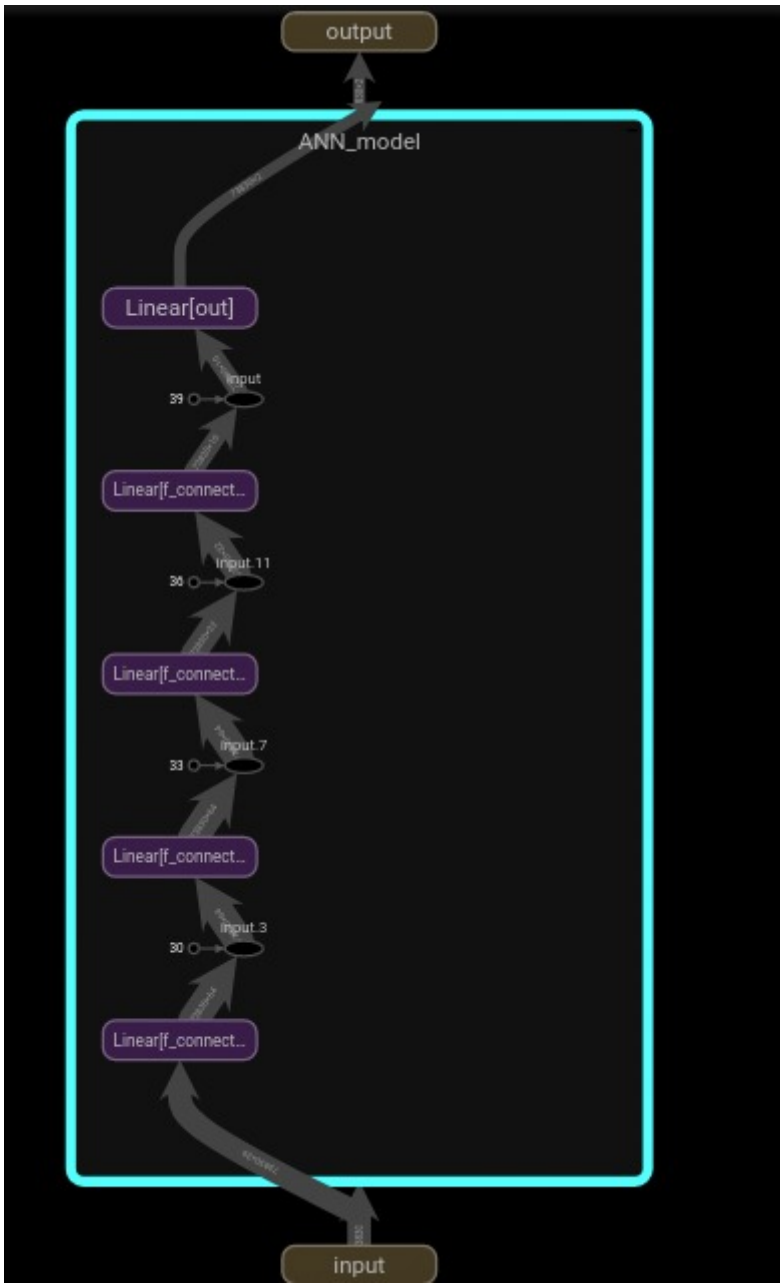

Accuracy
Accuracy
0.9
0.8
0.7
0.6
0
200
400
600
800

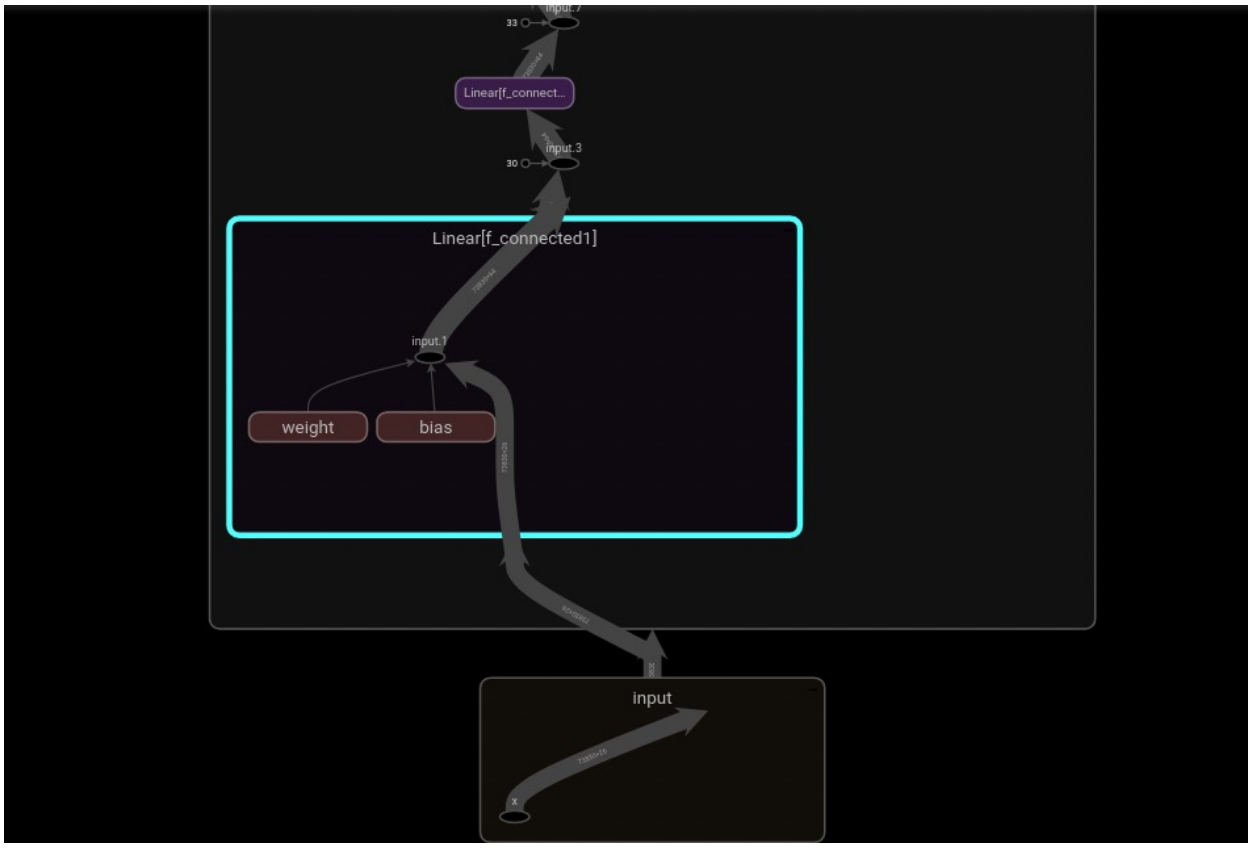

Training loss
Training loss
600
400
200
0
0
200
400
600
800

In [764]:
```
epochs = 1000
final_losses = []
for i in range(epochs):
    i += 1
    y_pred = model.forward(X_train)
    loss = loss_function(y_pred,y_train)
    final_losses.append(loss.item())
    _,predicted = torch.max(y_pred,1)
    running_correct = (predicted==y_train).sum().item()
    if i%100 == 1:
        print("Epoch Number: {} and the loss: {} accuracy : {}".format(i,loss.item(),running_correct/size_))
        writer.add_scalar('Training loss',loss.item(),i)
        writer.add_scalar('Accuracy',running_correct/size_,i)
        running_correct = 0
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
Epoch Number: 1 and the loss: 29.919469833374023 accuracy : 0.9179737234186645
Epoch Number: 101 and the loss: 0.5853087902069092 accuracy : 0.9179737234186645
Epoch Number: 201 and the loss: 0.5251537561416626 accuracy : 0.9179737234186645
Epoch Number: 301 and the loss: 0.4766022264957428 accuracy : 0.9179737234186645
Epoch Number: 401 and the loss: 0.4375353157520294 accuracy : 0.9179737234186645
Epoch Number: 501 and the loss: 0.40618953108787537 accuracy : 0.9179737234186645
Epoch Number: 601 and the loss: 0.3810862600803375 accuracy : 0.9179737234186645
Epoch Number: 701 and the loss: 0.3610040545463562 accuracy : 0.9179737234186645
Epoch Number: 801 and the loss: 0.3449486196041107 accuracy : 0.9179737234186645
Epoch Number: 901 and the loss: 0.33211925625801086 accuracy : 0.9179737234186645
```




output
ANN_model
Linear[out]
input
39
Linear[f_connect...
input.11
36
Linear[f_connect...
input.7
33
Linear[f_connect...
input.3
30
Linear[f_connect...
input



	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	roc_auc_score	Kaggle Score	Train Time(s)	Test Time(s)	Description
0	MLP Pytorch	HCDR	91.86%	92.03%	92.03%	0.73	0.71	85.73	9.62	Neural net Implementation

Comparisons

In phase 1, we mainly considered Logistic Regression and Random Forest Classifier as our candidate models, where our models obtained accuracy of 91.95 and 91.99 respectively.

In phase 1, we were able to obtain Accuracy of 0.7108 on Kaggle.

Furthermore,In the Phase 2, Feature Engineering was done, we derived 6 new features from the given set of datasets out of which we selected all 6 for our model training. Including these Features helped increase the Test accuracy of our models from 91.99% to 92.09% as it can be seen from the results. We also used GridSearchCV for Hyperparameter tuning for both logistic regression and Random forest Classifier. We were able to increase our Accuracy from 0.7108 to 0.7309 as our kaggle Accuracy in the phase 2.

In the final Phase, we implemented a multi layer perceptron neural network using pytorch library. We used 26 input features for our neural network. The train and test accuracy was almost similar. However, there was a noticeable change in the kaggle submission accuracy. We were able to obtain Accuracy of 0.7108 as our kaggle Accuracy.

Conclusion

For the Phase II of this Project, we built Logistic Regression and Random Forest Model, performed Hyperparameter Tuning using GridSearchCV and achieved an accuracy of over 92.1% with Random Forest and 92% with Logistic Regression.

Finally for the Phase III of the project, we implemented Neural Network using PyTorch Library.

We used around 26 highly important features for our model prediction with Leaky Rectified Linear unit as our Activation Function.

We also tried using synthetic sampling techniques like SMOTE and ADASYN to check if there was any change in the model performance.

Successively, we ran our model on the best parameters and found out an increase in the model accuracy and submitted the file on Kaggle.

We used Confusion Matrix, Log Loss, Accuracy, ROC Curve and CXE to evaluate the model performance.

Kaggle Submission Phase3

In [58]:
```
sub = pd.DataFrame()
```

In [59]:
```
test_df.head()
```

Out[59]:

	SK_ID_CURR	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_ID_PUBLISH	FLAG_EMP_PHONE	FLAG_W(
0	100001.0	-19241.0	-2329.0	-812.0	1.0	
1	100005.0	-18064.0	-4469.0	-1623.0	1.0	
2	100013.0	-20038.0	-4458.0	-3503.0	1.0	
3	100028.0	-13976.0	-1866.0	-4208.0	1.0	
4	100038.0	-13040.0	-2191.0	-4262.0	1.0	

5 rows × 198 columns

In [777…
```
sub['SK_ID_CURR']= test_df['SK_ID_CURR'].astype(int)
sub['TARGET'] = probs
sub.set_index('SK_ID_CURR')
```

Out[777…

SK_ID_CURR	TARGET
100001	0.178551
100005	0.178551
100013	0.178551

SK_ID_CURR	TARGET
100028	0.178551
100038	0.178551
...	...
456221	0.178551
456222	0.178551
456223	0.178551
456224	0.178551
456250	0.178551

48797 rows × 1 columns

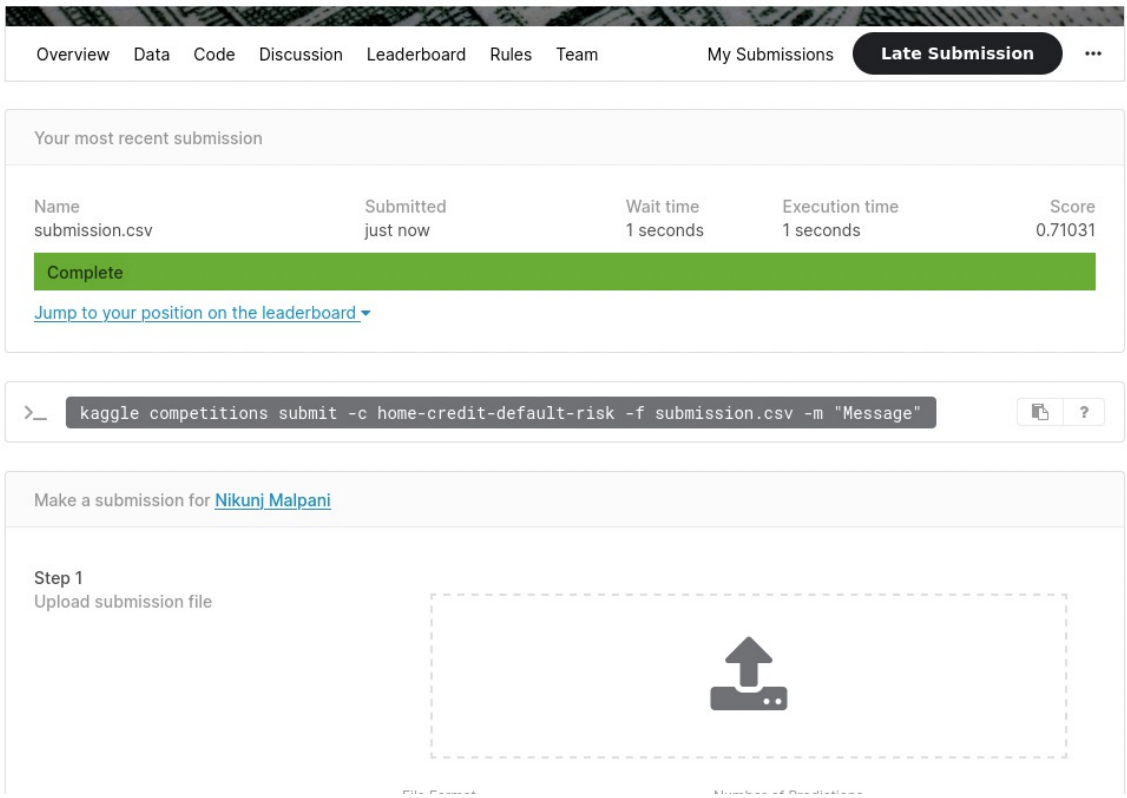
In [778…
```
sub = sub.drop_duplicates(subset='SK_ID_CURR', keep="first")
```

In [779…
```
sub.to_csv('submission.csv',index=False)
```

Kaggle Submission

In [76]:
```
from IPython.display import Image
Image(filename='WhatsApp Image 2021-12-13 at 4.08.43 PM.jpeg')
```

Out[76]:



References

https://github.iu.edu/jshanah/I526_AML_Student/tree/master/Labs/Labs-09-Pipelines-Column_Transformer_2020_10

https://github.iu.edu/jshanah/I526_AML_Student/tree/master/Labs/Labs-10.1-Decision-Trees

https://machinelearningmastery.com/random-forest-ensemble-in-python/

https://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics

https://builtin.com/data-science/random-forest-algorithm

https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

https://github.com/rakshithvasudev/Home-Credit-Default-Risk

https://randlow.github.io/posts/machine-learning/kaggle-home-loan-credit-risk-feat-eng-p3/

In [ ]: