Group 28: HCDR

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

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 [ ]:
```
!pip install kaggle
```

```
Collecting kaggle
  Downloading kaggle-1.5.12.tar.gz (58 kB)
     |████████████████████████████████| 58 kB 1.1 MB/s eta 0:00:01
Requirement already satisfied: six>=1.10 in /opt/anaconda3/lib/python3.9/site-pa
ckages (from kaggle) (1.16.0)
```

```
Requirement already satisfied: certifi in /opt/anaconda3/lib/python3.9/site-pack
ages (from kaggle) (2021.10.8)
Requirement already satisfied: python-dateutil in /opt/anaconda3/lib/python3.9/s
ite-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /opt/anaconda3/lib/python3.9/site-pac
kages (from kaggle) (2.26.0)
Requirement already satisfied: tqdm in /opt/anaconda3/lib/python3.9/site-package
s (from kaggle) (4.62.3)
Requirement already satisfied: python-slugify in /opt/anaconda3/lib/python3.9/si
te-packages (from kaggle) (5.0.2)
Requirement already satisfied: urllib3 in /opt/anaconda3/lib/python3.9/site-pack
ages (from kaggle) (1.26.7)
Requirement already satisfied: text-unidecode>=1.3 in /opt/anaconda3/lib/python
3.9/site-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.9/site
-packages (from requests->kaggle) (3.2)
Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/anaconda3/lib/p
ython3.9/site-packages (from requests->kaggle) (2.0.4)
Building wheels for collected packages: kaggle
  Building wheel for kaggle (setup.py) ... done
  Created wheel for kaggle: filename=kaggle-1.5.12-py3-none-any.whl size=73051 s
ha256=e8823672e2cfae3d940929ac269cec3918ccb84a29b4bd3ede4eacef91dfd6cd
  Stored in directory: /Users/pranayreddydasari/Library/Caches/pip/wheels/ac/b2/
c3/fa4706d469b5879105991d1c8be9a3c2ef329ba9fe2ce5085e
Successfully built kaggle
Installing collected packages: kaggle
Successfully installed kaggle-1.5.12
```

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

```
/root/shared/Masters/Spring_22/AML/I526_AML_Student/Changes/Unit_Project_Home_Cr
edit_Default_Risk/HCDR_Phase_1_baseline_submission
```

In [ ]:
```
!mkdir .kaggle
```

In [ ]:
```
!cp kaggle.json ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json
```

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

```
name                                  size  creationDate
----------------------------------  -----  -------------------
bureau.csv                          162MB  2019-12-11 02:55:35
application_train.csv               158MB  2019-12-11 02:55:35
bureau_balance.csv                  358MB  2019-12-11 02:55:35
credit_card_balance.csv             405MB  2019-12-11 02:55:35
POS_CASH_balance.csv                375MB  2019-12-11 02:55:35
application_test.csv                 25MB  2019-12-11 02:55:35
installments_payments.csv           690MB  2019-12-11 02:55:35
previous_application.csv            386MB  2019-12-11 02:55:35
sample_submission.csv               524KB  2019-12-11 02:55:35
HomeCredit_columns_description.csv   37KB  2019-12-11 02:55:35
```

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

```
/root/shared/Masters/Spring_22/AML/I526_AML_Student/Changes/Unit_Project_Home_Cr
edit_Default_Risk/HCDR_Phase_1_baseline_submission
mkdir: cannot create directory ‘home-credit-default-risk-data’: File exists
```

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

```
total 3326068
-rw-rw-r-- 1 root root     37383 Dec 11  2019 HomeCredit_columns_description.csv
```

```
-rw-rw-r-- 1 root root 392703158 Dec 11  2019 POS_CASH_balance.csv
-rw-rw-r-- 1 root root  26567651 Dec 11  2019 application_test.csv
-rw-rw-r-- 1 root root 166133370 Dec 11  2019 application_train.csv
-rw-rw-r-- 1 root root 170016717 Dec 11  2019 bureau.csv
-rw-rw-r-- 1 root root 375592889 Dec 11  2019 bureau_balance.csv
-rw-rw-r-- 1 root root 424582605 Dec 11  2019 credit_card_balance.csv
-rw-r--r-- 1 root root 721616255 Apr 10 21:27 home-credit-default-risk.zip
-rw-rw-r-- 1 root root 723118349 Dec 11  2019 installments_payments.csv
-rw-rw-r-- 1 root root 404973293 Dec 11  2019 previous_application.csv
-rw-rw-r-- 1 root root    536202 Dec 11  2019 sample_submission.csv
```

In [ ]:

```
! kaggle competitions download home-credit-default-risk -p $DATA_DIR
```

```
Downloading home-credit-default-risk.zip to home-credit-default-risk-data
100%|████████████████████████████████████████| 688M/688M [04:19<00:00, 3.03MB/s]
100%|████████████████████████████████████████| 688M/688M [04:19<00:00, 2.78MB/s]
```

Importing Libraries

In [122…

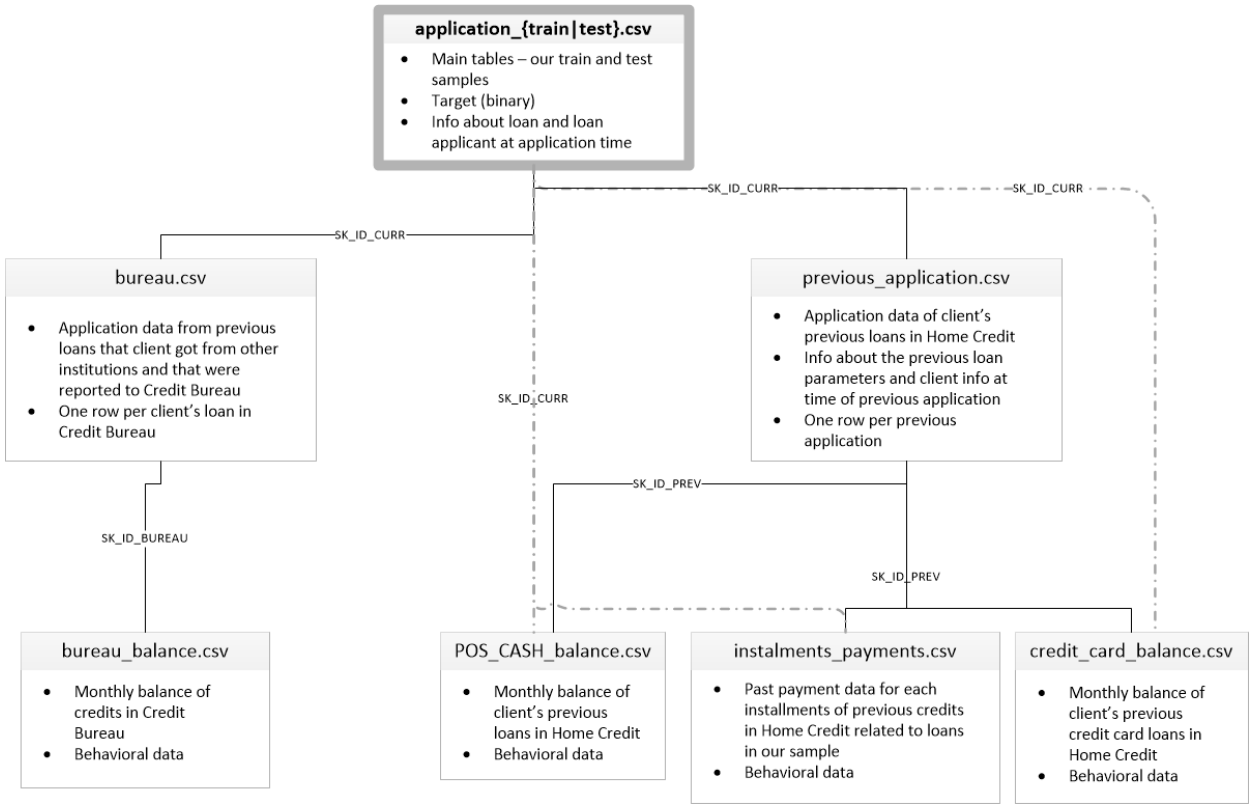
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import zipfile
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection 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
#warnings.filterwarnings('ignore')

from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.utils import resample
import json
from time import time, ctime
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score, log_loss
from sklearn.metrics import plot_roc_curve, plot_confusion_matrix, plot_precisio
from sklearn.model_selection import cross_validate
```

Data files overview

Data Dictionary

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

`HomeCredit_columns_description.csv`



Application train

In [123…

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.7/dist-packages
(0.90)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages
(from xgboost) (1.4.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from xgboost) (1.21.6)
```

In [124…

```
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.svm import SVC
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 matplotlib.pyplot as plt
```

```
from sklearn.metrics import roc_auc_score, make_scorer, roc_curve, ConfusionMatr
from xgboost import XGBClassifier
from sklearn.feature_selection import RFE
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings('ignore')
```

In [125…

```
# This function is used to reduce memory of a pandas dataframe
#Because of huge size of data kernel dies everytime. So came up with idea of red
# Reference: https://www.kaggle.com/gemartin/load-data-reduce-memory-usage
def reduce_mem_usage(df,name):
    """ iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe {} is {:.2f} MB'.format(name,start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object and col_type.name != 'category' and 'datetime' not
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).m
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int1
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int3
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int6
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.floa
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.fl
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)

    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem)

    return df
```

In [126…

```
# This function is used to plot countplot for all object datatypes from the data
def graph_objects(frame, hue=None):

    if hue is not None:
        hue = hue

    df = frame.dtypes
    df.index.name = 'columns'
    df = pd.DataFrame(df, columns=['dtype'])
    df = df.reset_index()
```

```
    df = df[df['dtype'] == 'object']
    list_objects = df['columns'].tolist()

    for obj in list_objects:
        plt.figure(figsize=(11,5))
        plot = sns.countplot(obj, data=frame, hue=hue)
        plt.xticks(rotation=90)
```

Datasets

- **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.

Loading Data from csv files

In [127…

```
def load_data(in_path, name):
    df = reduce_mem_usage(pd.read_csv(in_path),name)
    return df

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

In [128…

```
%%time
DATA_DIR = ""
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
```

```
Memory usage of dataframe application_train is 286.23 MB
Memory usage after optimization is: 92.38 MB
Decreased by 67.7%
Memory usage of dataframe application_test is 45.00 MB
Memory usage after optimization is: 14.60 MB
Decreased by 67.6%
Memory usage of dataframe bureau is 222.62 MB
Memory usage after optimization is: 112.95 MB
Decreased by 49.3%
Memory usage of dataframe bureau_balance is 511.20 MB
Memory usage after optimization is: 298.20 MB
Decreased by 41.7%
Memory usage of dataframe credit_card_balance is 488.80 MB
Memory usage after optimization is: 209.87 MB
Decreased by 57.1%
Memory usage of dataframe installments_payments is 350.42 MB
Memory usage after optimization is: 131.41 MB
Decreased by 62.5%
Memory usage of dataframe previous_application is 356.06 MB
Memory usage after optimization is: 235.77 MB
Decreased by 33.8%
Memory usage of dataframe POS_CASH_balance is 477.59 MB
Memory usage after optimization is: 194.02 MB
Decreased by 59.4%
CPU times: user 41.5 s, sys: 4.09 s, total: 45.6 s
Wall time: 56.8 s
```

In [129…

```
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          : [ 22,334,820, 3]
dataset credit_card_balance     : [  2,785,583, 23]
dataset installments_payments   : [  5,741,226, 8]
dataset previous_application    : [  1,261,346, 37]
dataset POS_CASH_balance        : [  7,824,771, 8]
```

Exploratory Data Analysis

EDA for Application Train

Summary of Application train

In [ ]:

```
datasets["application_train"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float16(61), float32(4), int16(2), int32(2), int8(37), object(16)
memory usage: 92.4+ MB
```

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

Out[ ]:

	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
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990259e+05	27
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	144
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	16
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 [ ]:
```
datasets["application_train"].describe(include='all') #look at all categorical a
```

Out[ ]:

	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
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

Missing data for application train

In [ ]:
```
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(10)
```

Out[ ]:

	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

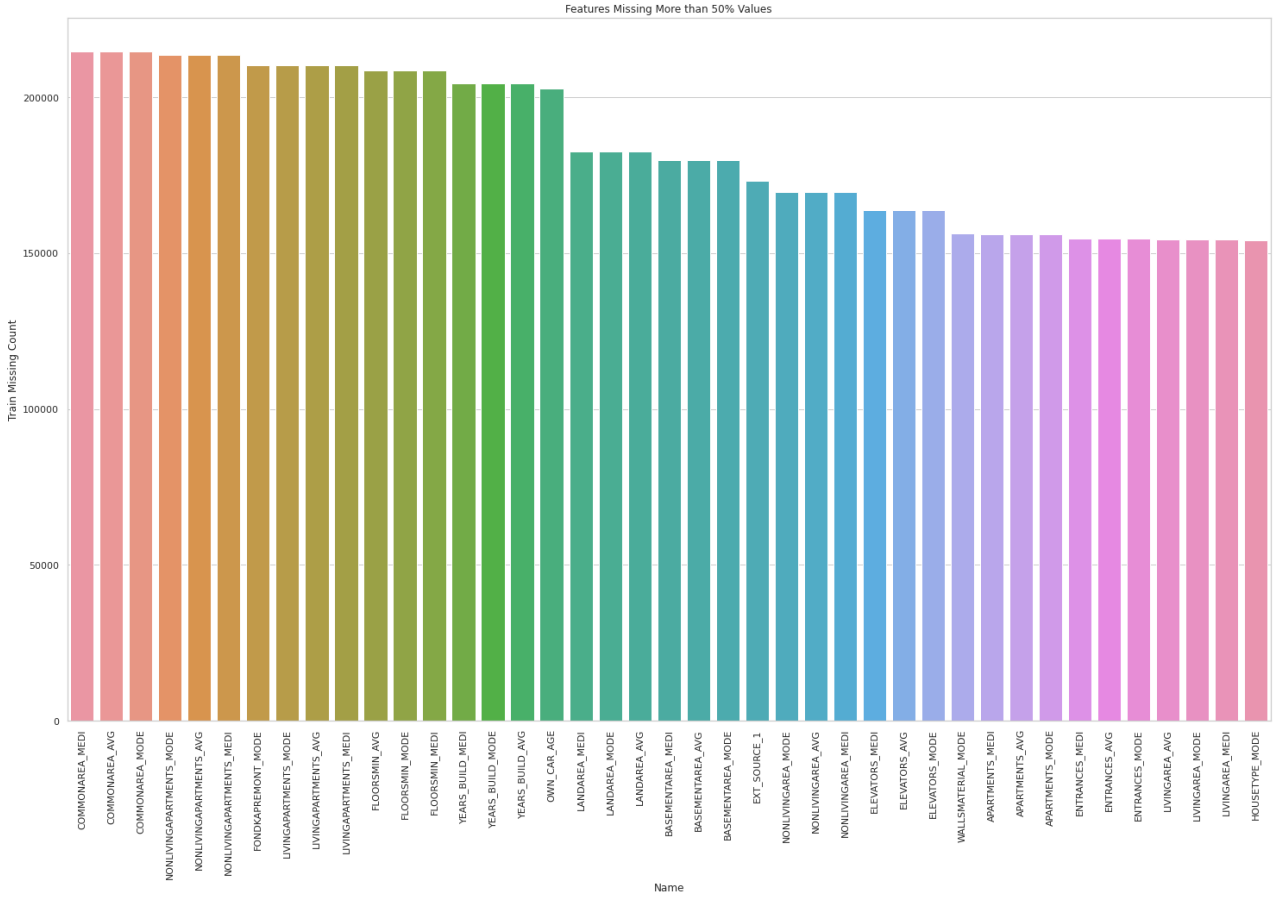
In [ ]:
```
print(f'{missing_application_train_data[missing_application_train_data["Train Mi
 attributes out of {datasets["application_train"].shape[1]} attributes has missi
```

```
67 attributes out of 122 attributes has missing values
```

In [ ]:
```
missing_application_train_data = pd.DataFrame(missing_application_train_data[mis
```

In [ ]:
```
missing_application_train_data.columns.name = 'Name'
missing_application_train_data['Name']=missing_application_train_data.index
```

In [ ]:
```
sns.set(style="whitegrid", color_codes=True,rc={'figure.figsize':(25,15)})
sns.barplot(x = 'Name', y = 'Train Missing Count', data=missing_application_trai
plt.xticks(rotation = 90)
plt.show()
```



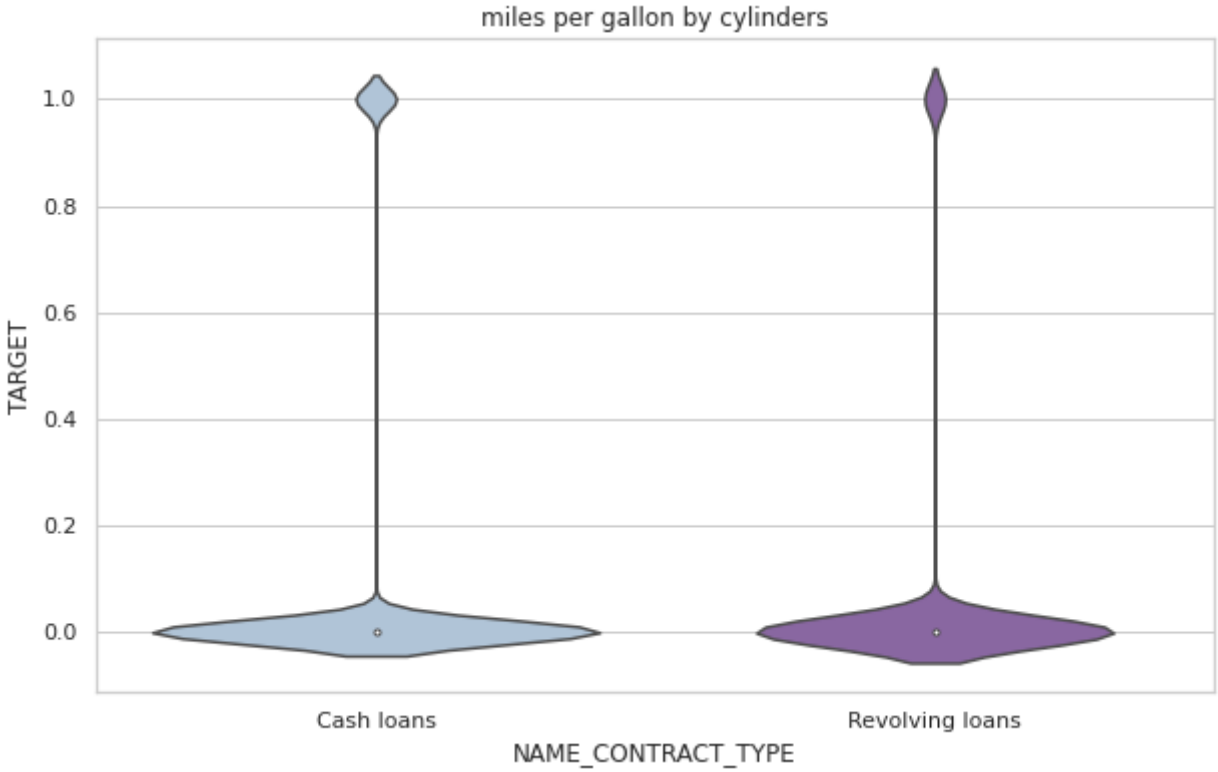
Observations

- The descriptive statistics reveal anomalies in the data for Days Birth, Days Employed, Days Registration, Days Id Publish which is a negative value and is not expected.
- Own car age has a max of 91.
- There are redundant features linked to living space and realty that may be completely wiped out during the feature education process to minimize multicollinearity difficulties.

Distribution of NAME_CONTRACT_TYPE attribute with TARGET class

In [ ]:

```
plt.figure(figsize = (10,6))
plt.title('miles per gallon by cylinders')
sns.violinplot(data=datasets['application_train'], x='NAME_CONTRACT_TYPE',y='TAR
plt.show()
```



Column Types

In [ ]:
```
datasets['application_train'].dtypes.value_counts()
```

Out[ ]:
```
float16    61
int8       37
object     16
float32     4
int32       2
int16       2
dtype: int64
```

Observations:

We could see that there are 16 categorical attributes in application train file and the remaining are numerical attributes

In [ ]:
```
datasets['application_train'].select_dtypes('object').nunique()
```

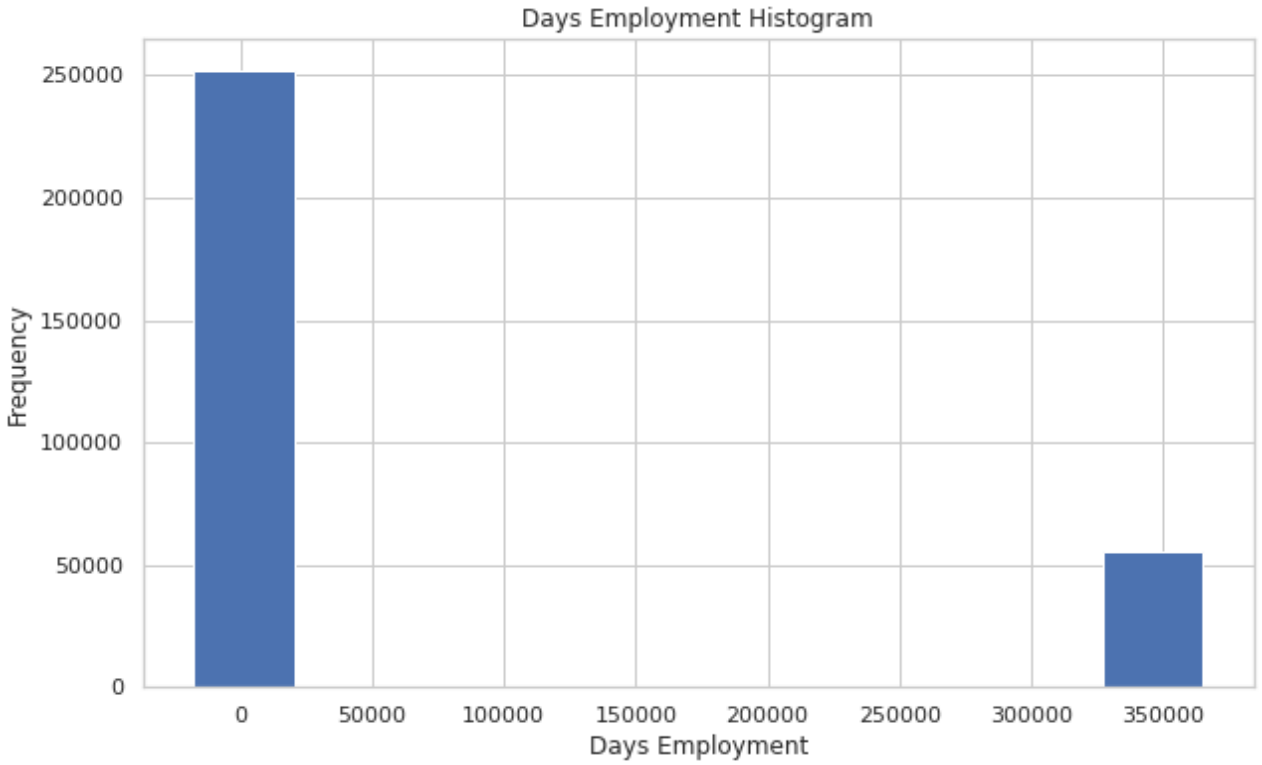
Out[ ]:
```
NAME_CONTRACT_TYPE             2
CODE_GENDER                    3
FLAG_OWN_CAR                   2
FLAG_OWN_REALTY                2
NAME_TYPE_SUITE                7
NAME_INCOME_TYPE               8
NAME_EDUCATION_TYPE            5
NAME_FAMILY_STATUS             6
NAME_HOUSING_TYPE              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
```

Observations

Above output depicts the number of unique values in categorical attributes in application train data.

In [ ]:
```
plt.figure(figsize = (10,6))
datasets['application_train']['DAYS_EMPLOYED'].plot.hist(title='Days Employment
plt.xlabel('Days Employment');
```



Anamolies in Application train of DAYS_EMPLOYED attribute

In [ ]:
```
datasets['application_train']['DAYS_EMPLOYED'].max()
```

Out[ ]:
```
365243
```

In [ ]:
```
max_days_employed = datasets['application_train']['DAYS_EMPLOYED'].max()

outliers=datasets['application_train'][datasets['application_train']['DAYS_EMPLO

non_outliers=datasets['application_train'][datasets['application_train']['DAYS_E

print('The non-anomalies default on %0.2f%% of loans' %(100*non_outliers['TARGET
print('The anomalies default on %0.2f%% of loans' % (100 * outliers['TARGET'].me
print('There are %d anomalous days of employment' % len(outliers))
```

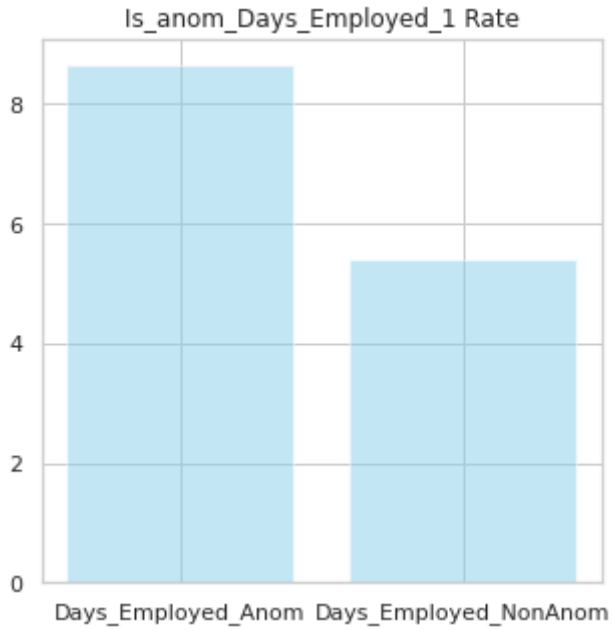
```
The non-anomalies default on 8.66% of loans
The anomalies default on 5.40% of loans
There are 55374 anomalous days of employment
```

In [ ]:
```
indexs = ['Days_Employed_Anom','Days_Employed_NonAnom']
column = ['Target_mean']
anamolies_data = [100*non_outliers['TARGET'].mean(),100 * outliers['TARGET'].mea
Is_anom_Days_Employed = pd.DataFrame(anamolies_data, columns=column, index=index
Is_anom_Days_Employed
```

Out[ ]:

	Target_mean
Days_Employed_Anom	8.659975
Days_Employed_NonAnom	5.399646

In [ ]:
```
plt.figure(figsize=(5,5))
plt.bar(indexs, Is_anom_Days_Employed['Target_mean'],color='skyblue',alpha=0.5)
plt.title("Is_anom_Days_Employed_1 Rate")
plt.show()
```



Observation:

- Distribution of Outliers and non outliers of DAYS_EMPLOYED attribute in application train data and we could see that anamolies had higher number of employement days.
- Data is not logical as number of days employed would show a steady source of income and could be a useful feature for predicting risk.

Distribution of the target column

In [ ]:
```
sns.catplot(data= datasets["application_train"], x = 'TARGET', kind='count').set
plt.show()
```

Observations

Above graph shows that majority of people would be able to repay the loan.

Correlation with the target column

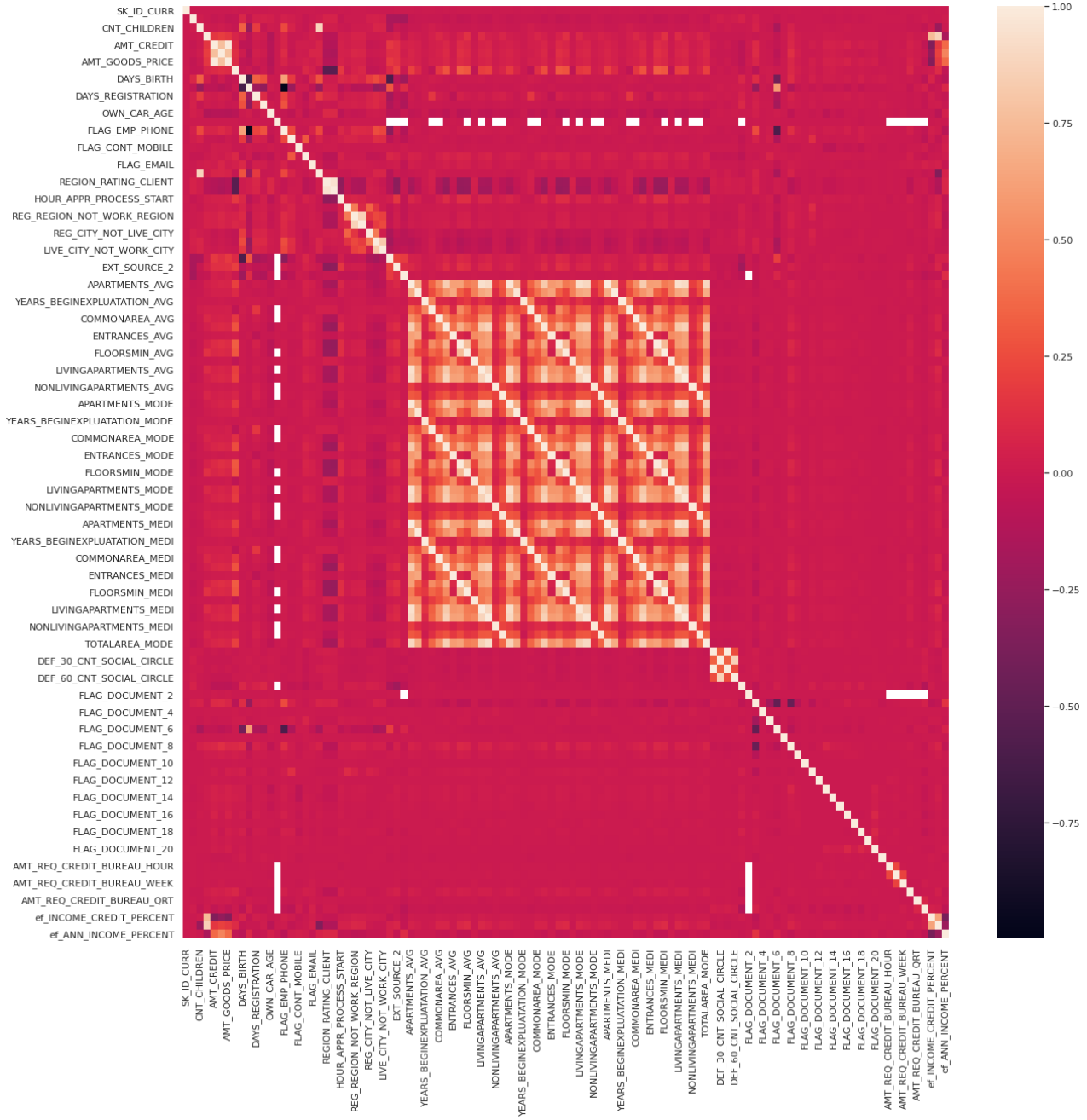
In [ ]:
```
correlations = datasets["application_train"].corr()['TARGET'].sort_values()
print('Most Positive Correlations:\n', correlations.tail(10))
print('\nMost Negative Correlations:\n', correlations.head(10))
```

```
Most Positive Correlations:
 FLAG_DOCUMENT_3                0.044346
REG_CITY_NOT_LIVE_CITY          0.044395
FLAG_EMP_PHONE                  0.045982
REG_CITY_NOT_WORK_CITY          0.050994
DAYS_ID_PUBLISH                 0.051457
DAYS_LAST_PHONE_CHANGE          0.055219
REGION_RATING_CLIENT            0.058899
REGION_RATING_CLIENT_W_CITY     0.060893
DAYS_BIRTH                      0.078239
TARGET                          1.000000
Name: TARGET, dtype: float64

Most Negative Correlations:
 EXT_SOURCE_3                 -0.178918
EXT_SOURCE_2                  -0.160471
EXT_SOURCE_1                  -0.155318
DAYS_EMPLOYED                 -0.044932
FLOORSMAX_AVG                 -0.044005
FLOORSMAX_MEDI                -0.043769
FLOORSMAX_MODE                -0.043228
AMT_GOODS_PRICE               -0.039645
REGION_POPULATION_RELATIVE    -0.037231
ELEVATORS_AVG                 -0.034199
Name: TARGET, dtype: float64
```

Observations

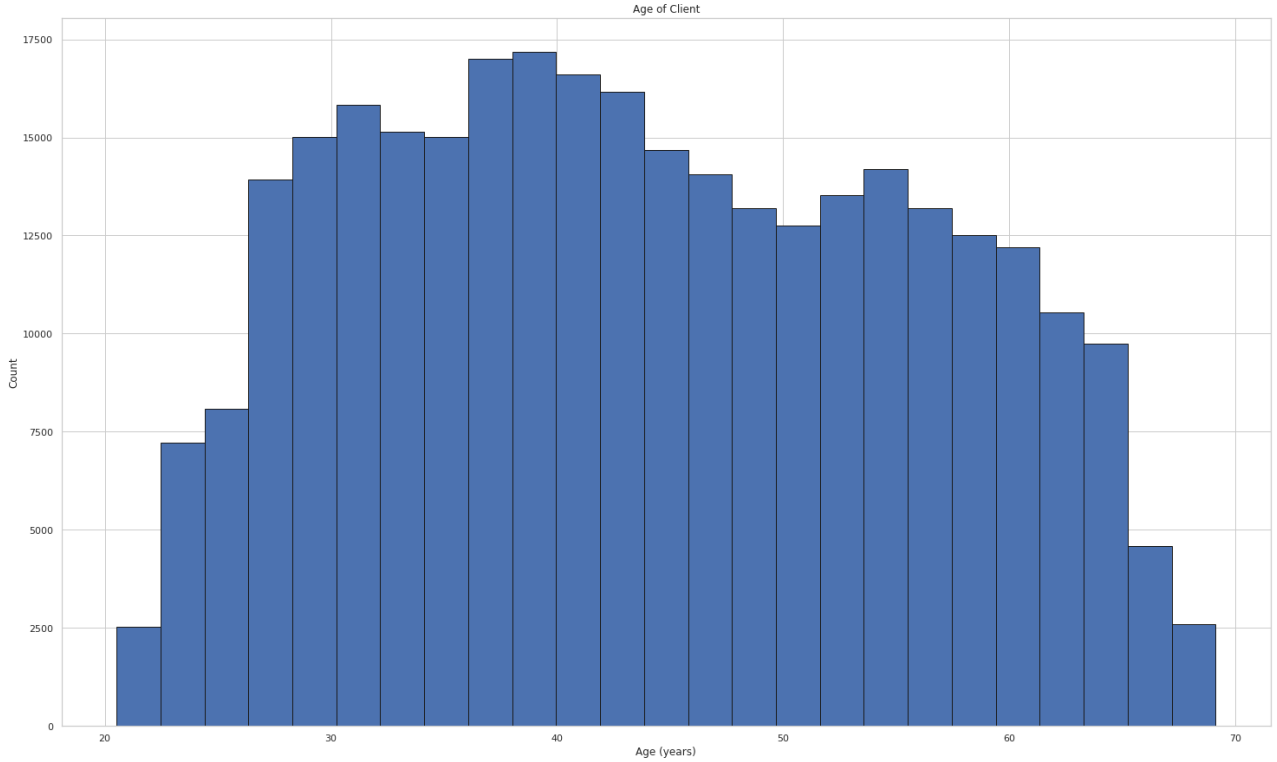
- Above are the top 10 positively and neagtively correlated features with Target attribute which could be important features for prediction and could be useful during feature engineering.
- Below is the heatmap depicted for visual representation of correlation of attributes.

In [ ]:
```
corrs = datasets['application_train'].corr()
plt.figure(figsize=(20,20))
sns.heatmap(datasets['application_train'].corr())
plt.show()
```



Applicants Age

In [ ]:
```
plt.hist(datasets["application_train"]['DAYS_BIRTH'] / -365, edgecolor = 'k', bi
plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('Count');
```



Observations

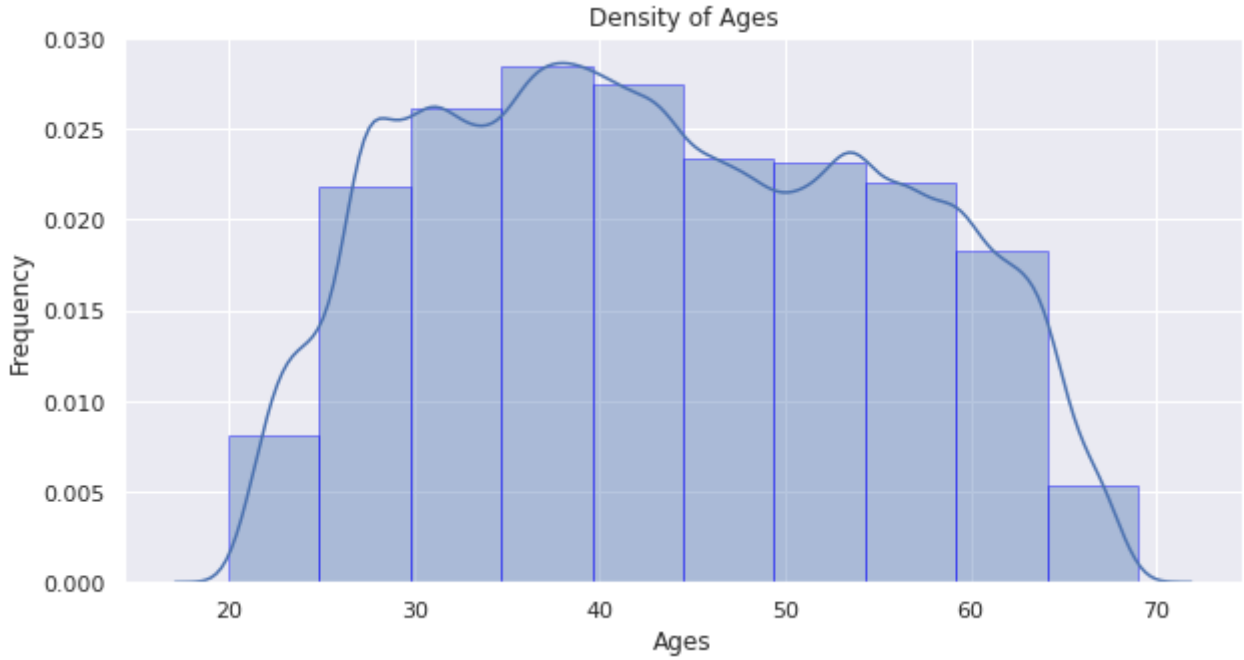
Above histogram shows that people in between the age of 35 to 40 have majorly applied for loan and this could be an useful factor for prediction.

In [ ]:

```
plt.figure(figsize=(10,8))

datasets["application_train"]

sns.kdeplot(datasets["application_train"].loc[datasets["application_train"]['TAR

sns.kdeplot(datasets["application_train"].loc[datasets["application_train"]['TAR

plt.xlabel('Age(years)');
plt.ylabel('Density');
plt.title('Distribution of Ages');
```

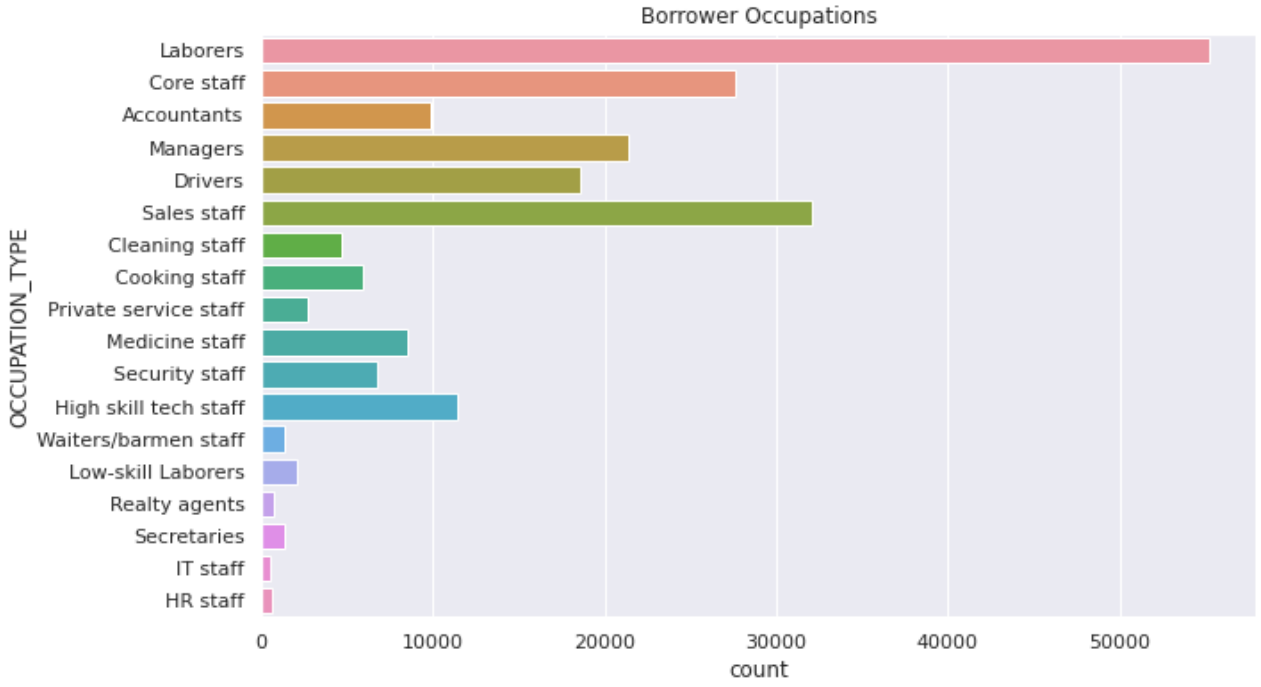
In [ ]:
```
ages = [int(-x/365) for x in datasets["application_train"].DAYS_BIRTH]
sns.set(rc={'figure.figsize':(10,5)})
sns.distplot(ages, hist=True,
             bins=10,hist_kws={'edgecolor':'blue'})
plt.title('Density of Ages')
plt.xlabel('Ages')
plt.ylabel('Frequency')
```

Out[ ]:
```
Text(0, 0.5, 'Frequency')
```



Applicants occupations

In [ ]:
```
plt.subplots(figsize =(10, 6))
sns.countplot(data = datasets["application_train"], y = 'OCCUPATION_TYPE')
plt.title('Borrower Occupations')
plt.show()
```



Observations

Above countplot shows that the most of the applicants are Laborers and this could be an intresting area to research in future phase.

EDA for external sources

Performing EDA on external sources attributes to check whether these attributes would be helpful or not.

In [ ]:
```
ext_data=datasets["application_train"][['TARGET','EXT_SOURCE_1','EXT_SOURCE_2','
ext_data_corrs=ext_data.corr()
ext_data_corrs
```

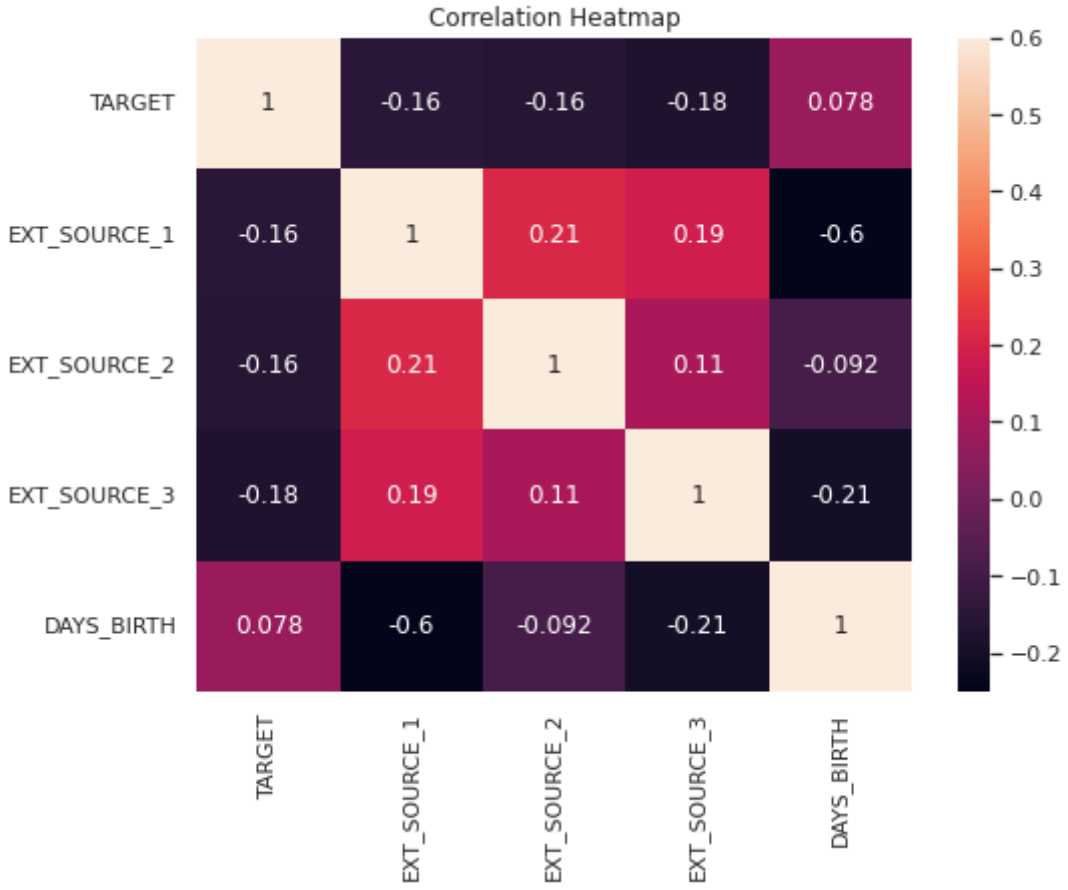
Out[ ]:

	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
TARGET	1.000000	-0.155318	-0.160471	-0.178918	0.078239
EXT_SOURCE_1	-0.155318	1.000000	0.213984	0.186842	-0.600611
EXT_SOURCE_2	-0.160471	0.213984	1.000000	0.109166	-0.091996
EXT_SOURCE_3	-0.178918	0.186842	0.109166	1.000000	-0.205477
DAYS_BIRTH	0.078239	-0.600611	-0.091996	-0.205477	1.000000

Observation

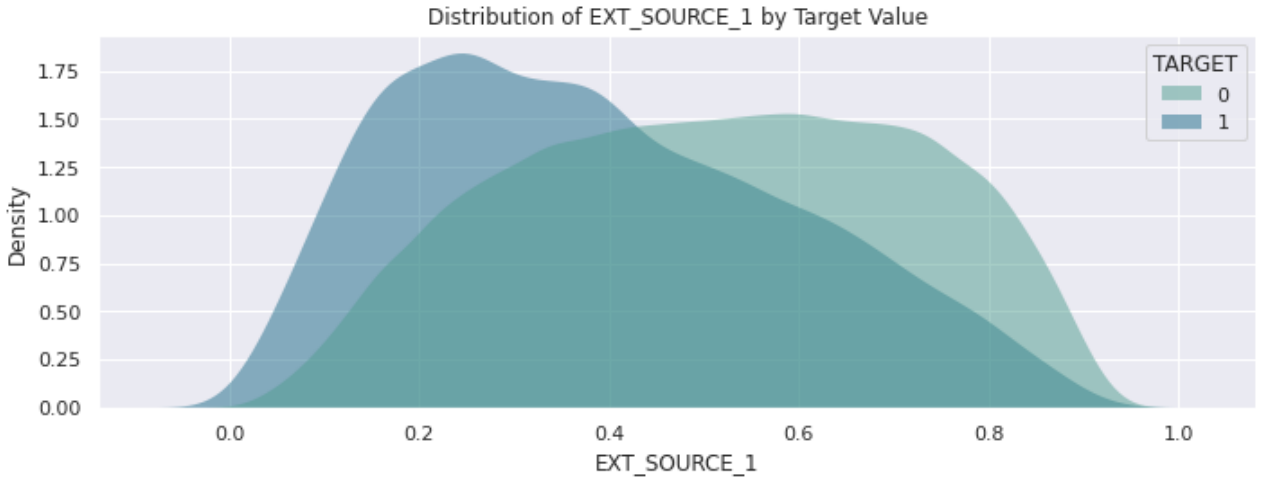
Based on the above correlation matrix, we can clearly state that External Source 1,2,3 and Days_Birth atrributes are not highly correlated with Target Attribute as the high value is 0.6. So, we donot drop these features in preprocessing since these may be useful in predicting the target.

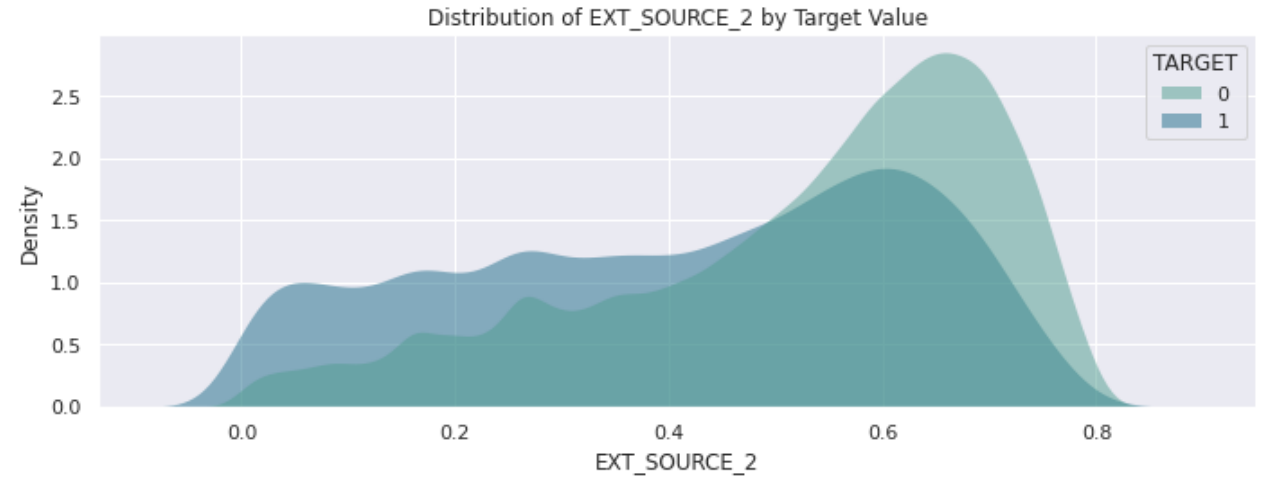
In [ ]:
```
plt.figure(figsize=(8,6))

sns.heatmap(ext_data_corrs,  vmin=-0.25, annot=True, vmax=0.6)
plt.title('Correlation Heatmap');
```

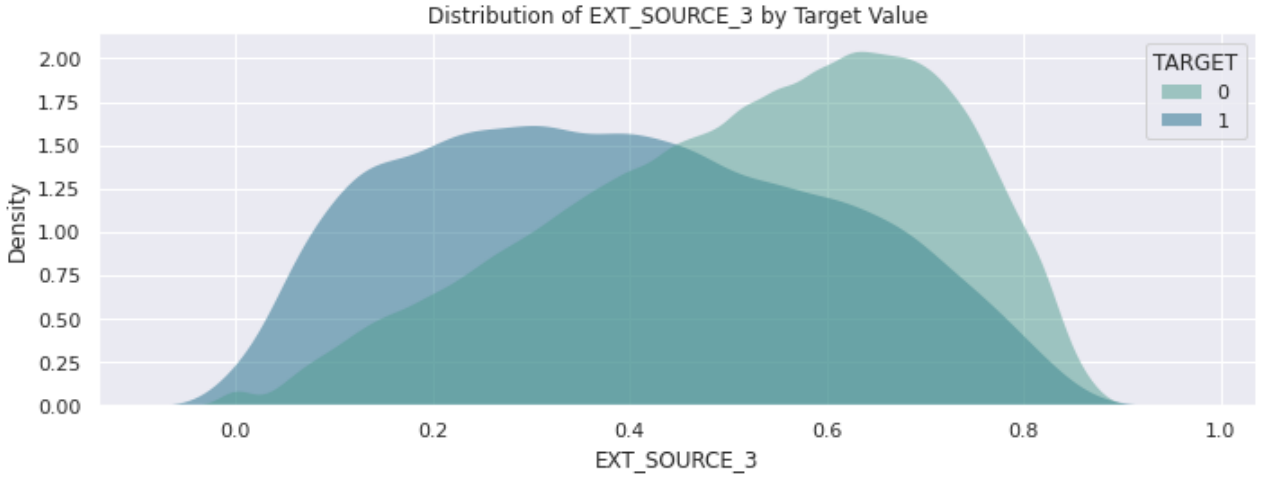


In [ ]:

```
plt.figure(figsize=(10,12))

# iterate through the sources
for i, source in enumerate(['EXT_SOURCE_1','EXT_SOURCE_2','EXT_SOURCE_3']):
    plt.subplot(3,1,i+1)

    sns.kdeplot(data=datasets["application_train"], x=source, hue="TARGET",fill=
   alpha=.5, linewidth=0)

    plt.title('Distribution of %s by Target Value' % source)
    plt.xlabel('%s' %source);
    plt.ylabel('Density');
plt.tight_layout(h_pad=2.5)
```







EXT_SOURCE_1, EXT_SOURCE_2, EXT_SOURCE_3 are distributed almost equally with target variables.

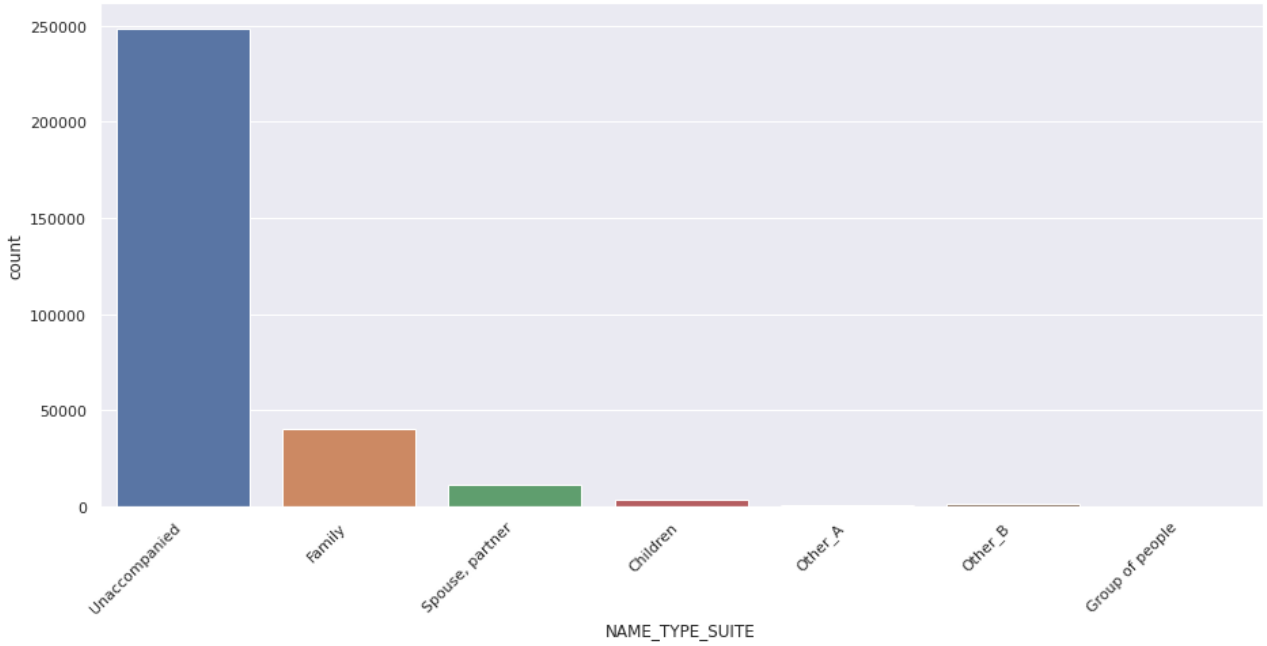
Categorical Variables

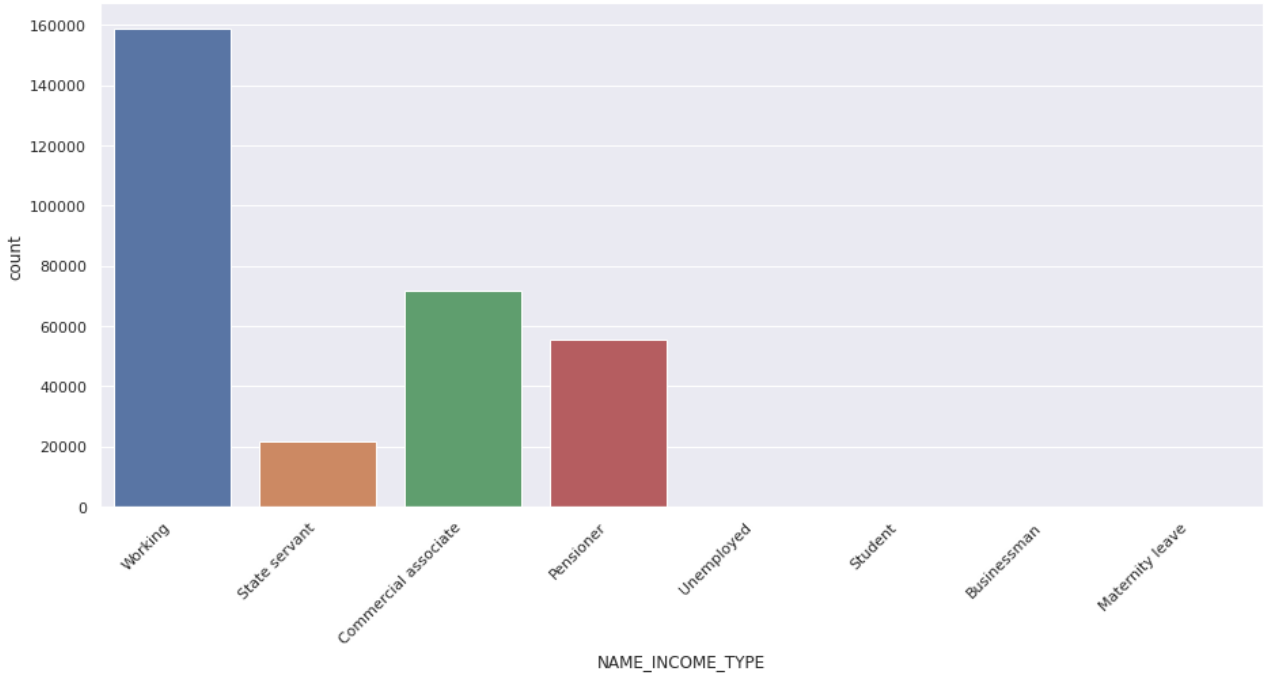
In [ ]:

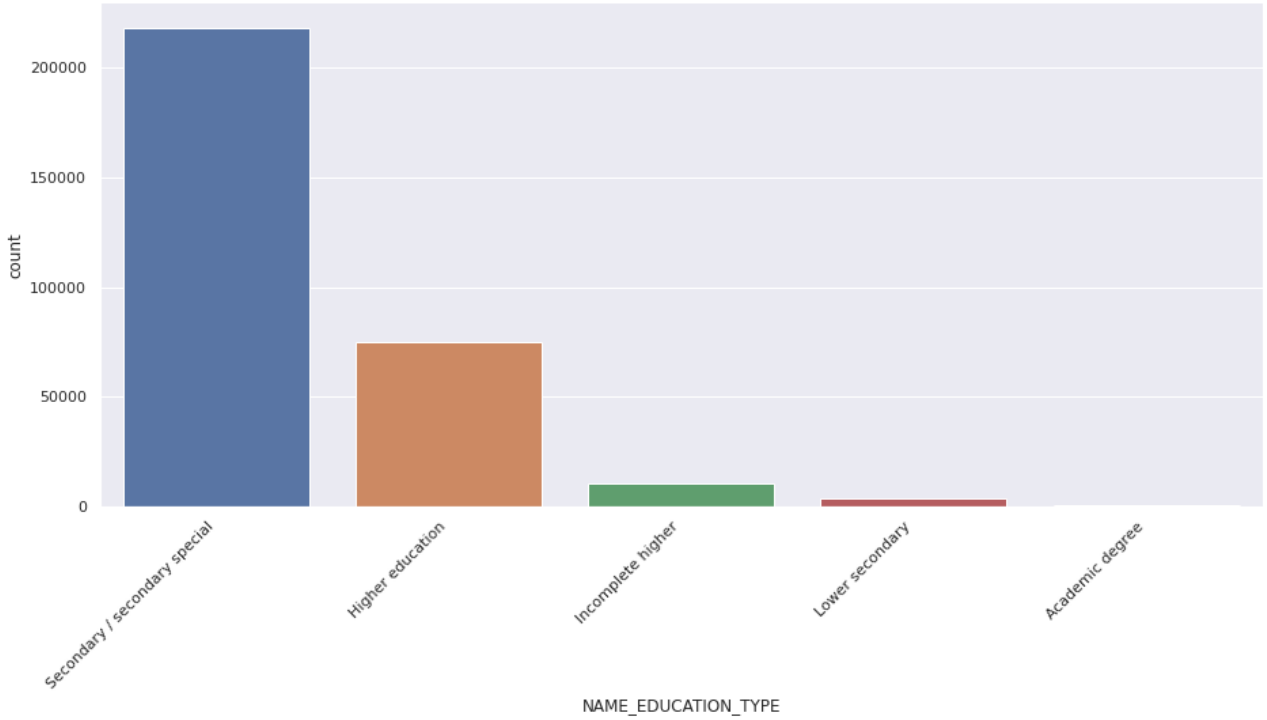
```
i=1
for col in datasets["application_train"].columns:
    if datasets["application_train"][col].dtype == 'object':
        print("object column %s have %s unique values"%(str(col),datasets["appli
        if datasets["application_train"][col].nunique() <= 3:
```

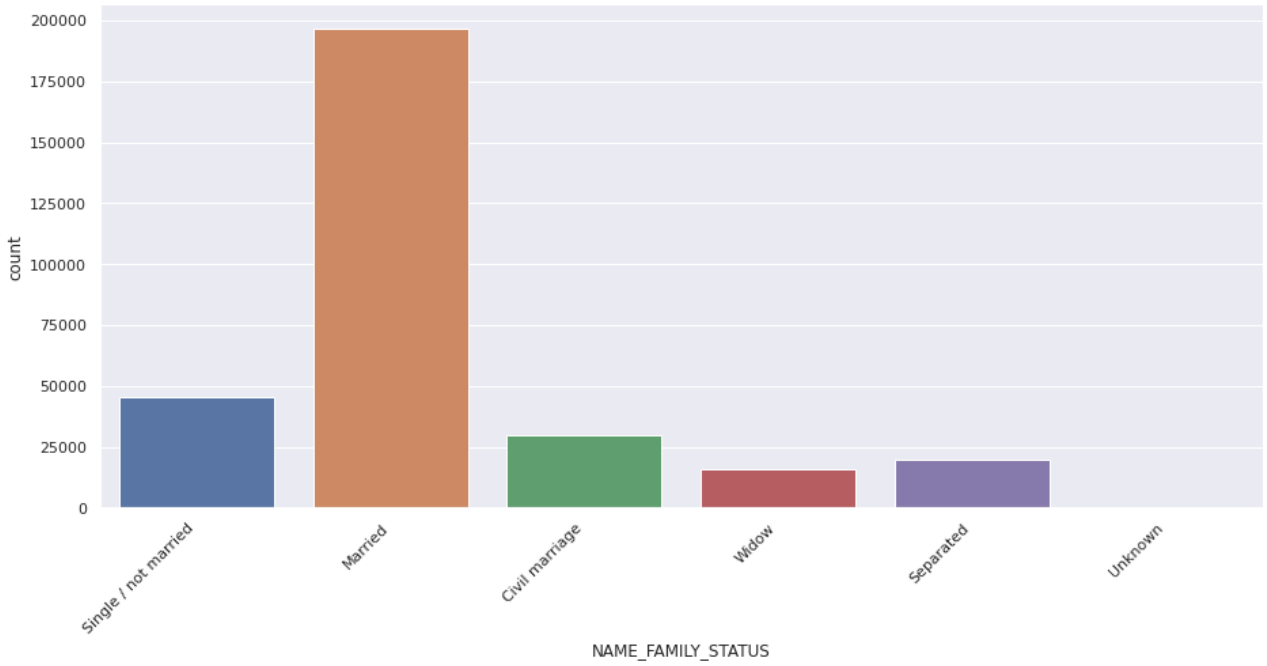
```
            print(datasets["application_train"][col].value_counts())
            print('-------------')
        else:
            ax = sns.catplot(x=col, kind="count", data=datasets["application_tra
            for axes in ax.axes.flat:
                axes.set_xticklabels(axes.get_xticklabels(),rotation = 45,horizo
```

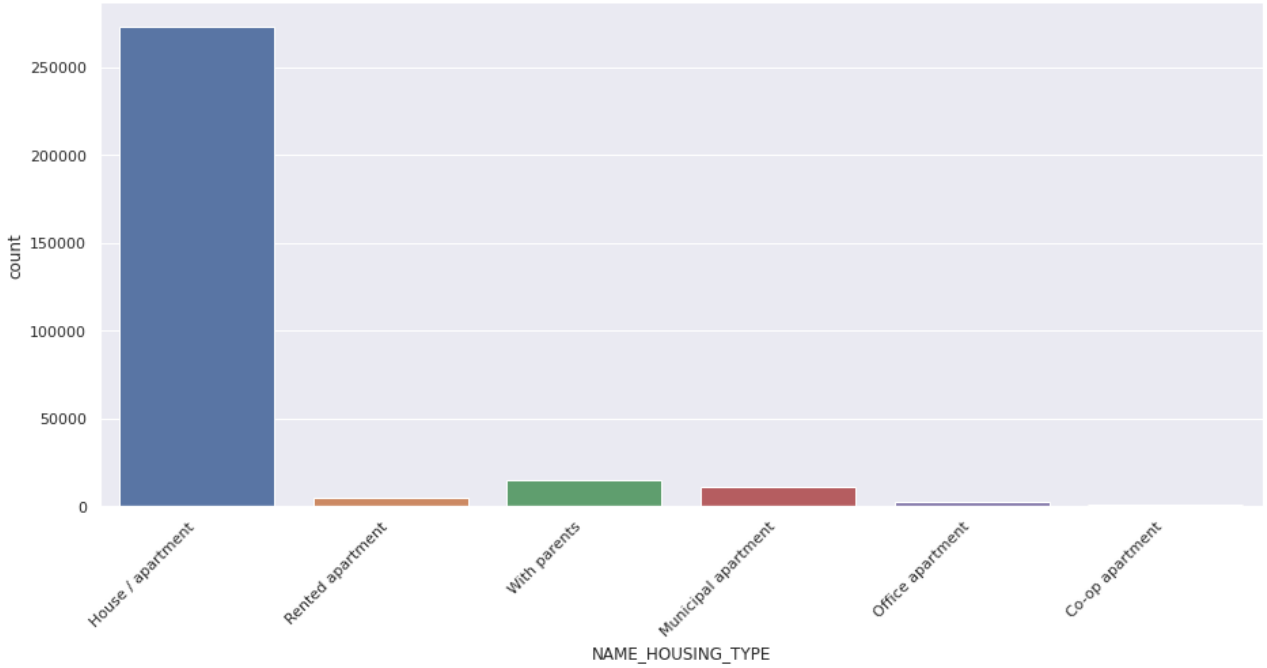
```
object column NAME_CONTRACT_TYPE have 2 unique values
Cash loans         278232
Revolving loans     29279
Name: NAME_CONTRACT_TYPE, dtype: int64
-------------
object column CODE_GENDER have 3 unique values
F      202448
M      105059
XNA         4
Name: CODE_GENDER, dtype: int64
-------------
object column FLAG_OWN_CAR have 2 unique values
N    202924
Y    104587
Name: FLAG_OWN_CAR, dtype: int64
-------------
object column FLAG_OWN_REALTY have 2 unique values
Y    213312
N     94199
Name: FLAG_OWN_REALTY, dtype: int64
-------------
object column NAME_TYPE_SUITE have 7 unique values
object column NAME_INCOME_TYPE have 8 unique values
object column NAME_EDUCATION_TYPE have 5 unique values
object column NAME_FAMILY_STATUS have 6 unique values
object column NAME_HOUSING_TYPE have 6 unique values
object column OCCUPATION_TYPE have 18 unique values
object column WEEKDAY_APPR_PROCESS_START have 7 unique values
object column ORGANIZATION_TYPE have 58 unique values
object column FONDKAPREMONT_MODE have 4 unique values
object column HOUSETYPE_MODE have 3 unique values
block of flats      150503
specific housing      1499
terraced house        1212
Name: HOUSETYPE_MODE, dtype: int64
-------------
object column WALLSMATERIAL_MODE have 7 unique values
object column EMERGENCYSTATE_MODE have 2 unique values
No     159428
Yes      2328
Name: EMERGENCYSTATE_MODE, dtype: int64
-------------
```

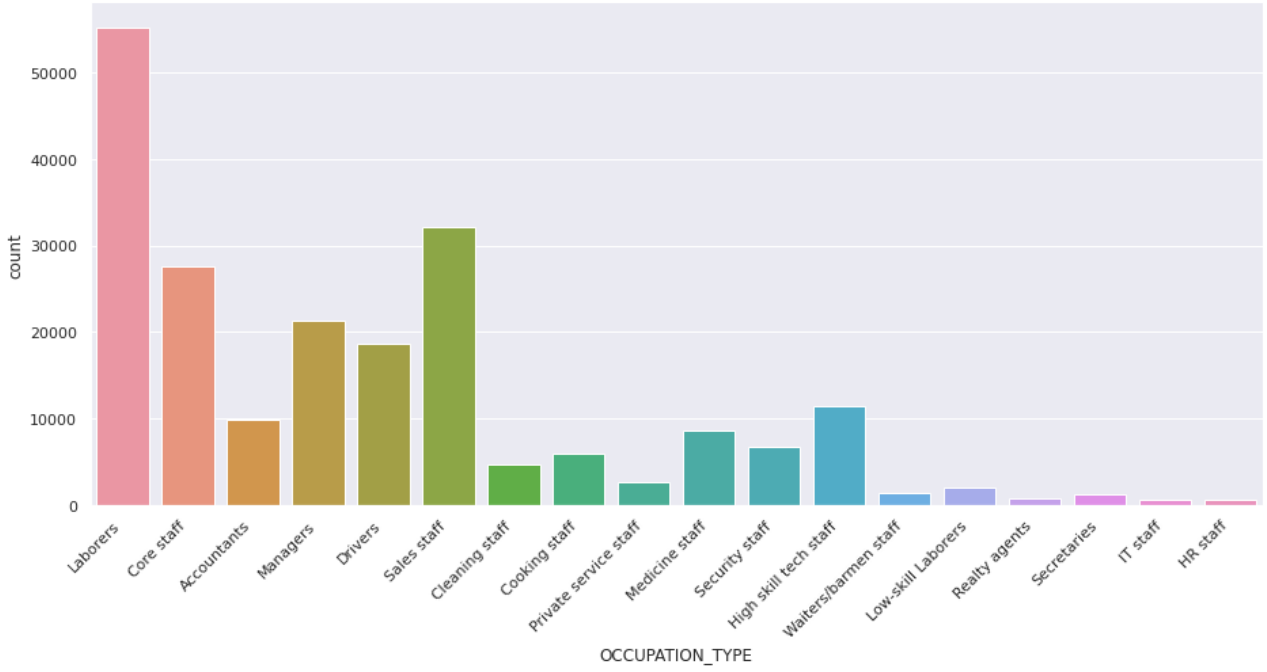

250000
200000
150000
100000
50000
0
count
Unaccompanied
Family
Spouse, partner
Children
Other_A
Other_B
Group of people
NAME_TYPE_SUITE

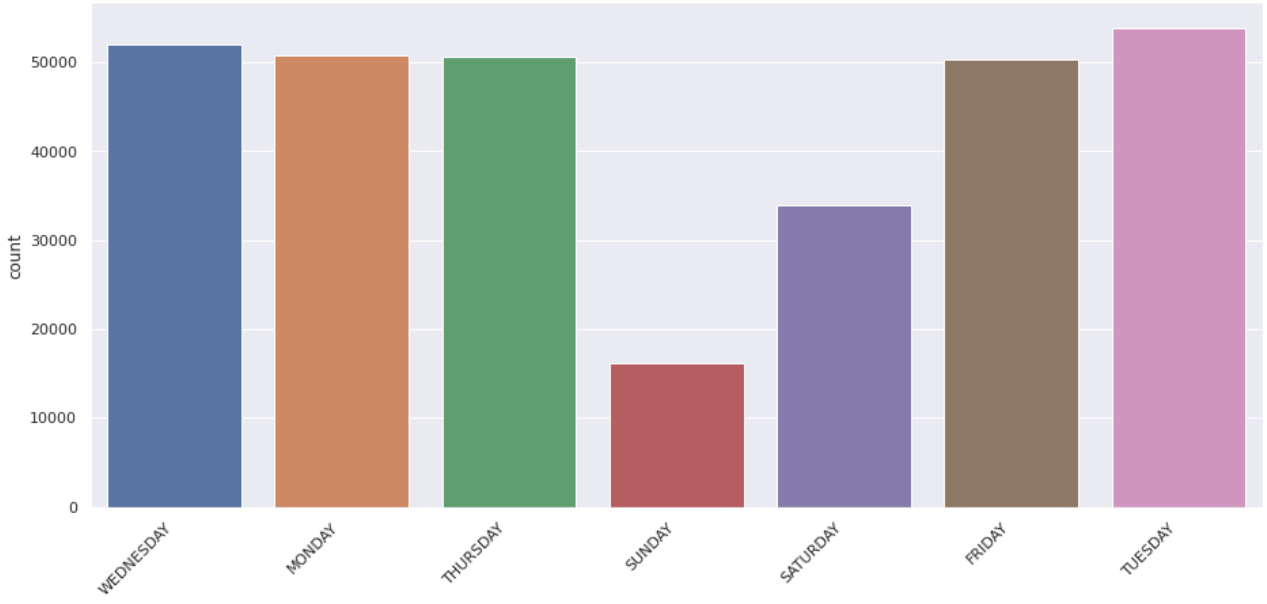

160000
140000
120000
100000
80000
60000
40000
20000
0
count
Working
State servant
Commercial associate
Pensioner
Unemployed
Student
Businessman
Maternity leave
NAME_INCOME_TYPE

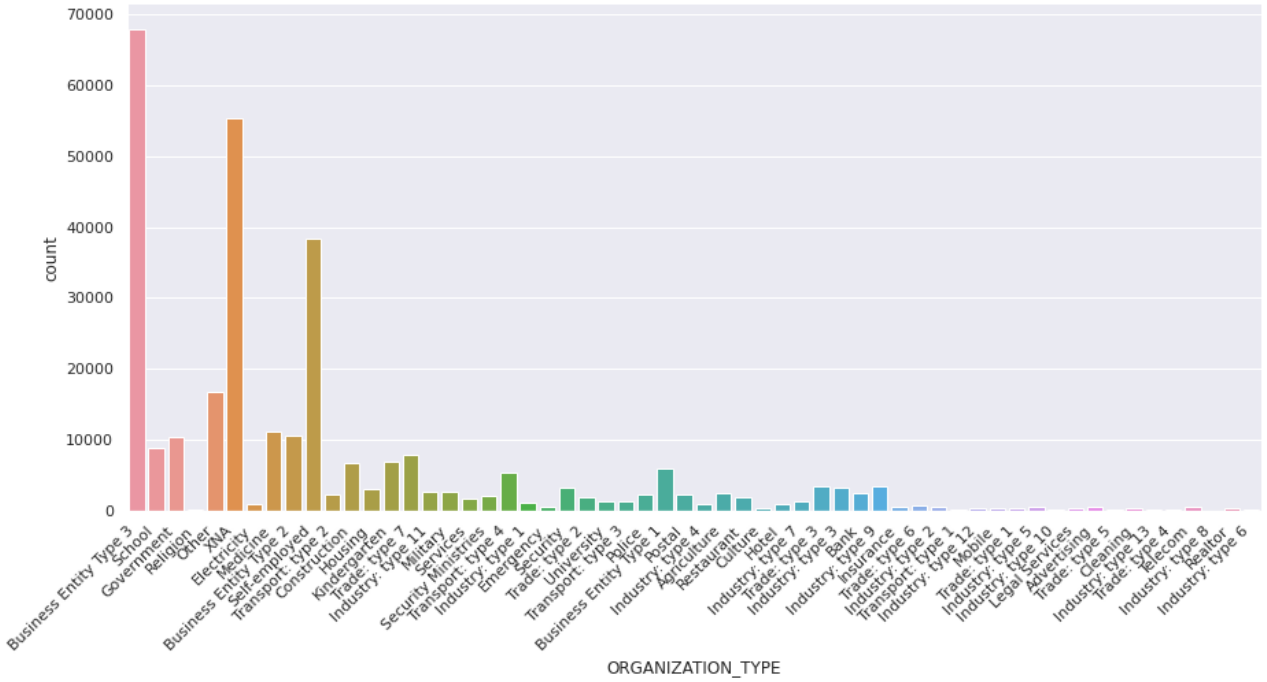

200000
150000
100000
50000
0
count
Secondary / secondary special
Higher education
Incomplete higher
Lower secondary
Academic degree
NAME_EDUCATION_TYPE

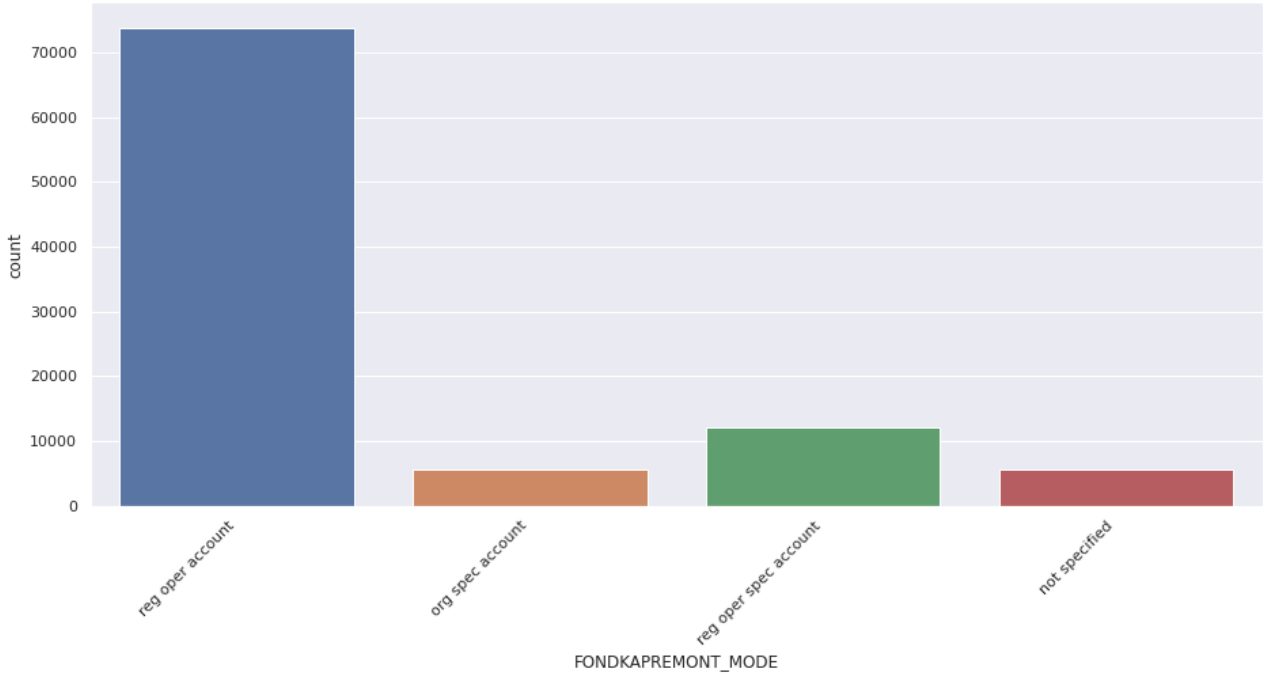

200000
175000
150000
125000
100000
75000
50000
25000
0
count
Single / not married
Married
Civil marriage
Widow
Separated
Unknown
NAME_FAMILY_STATUS

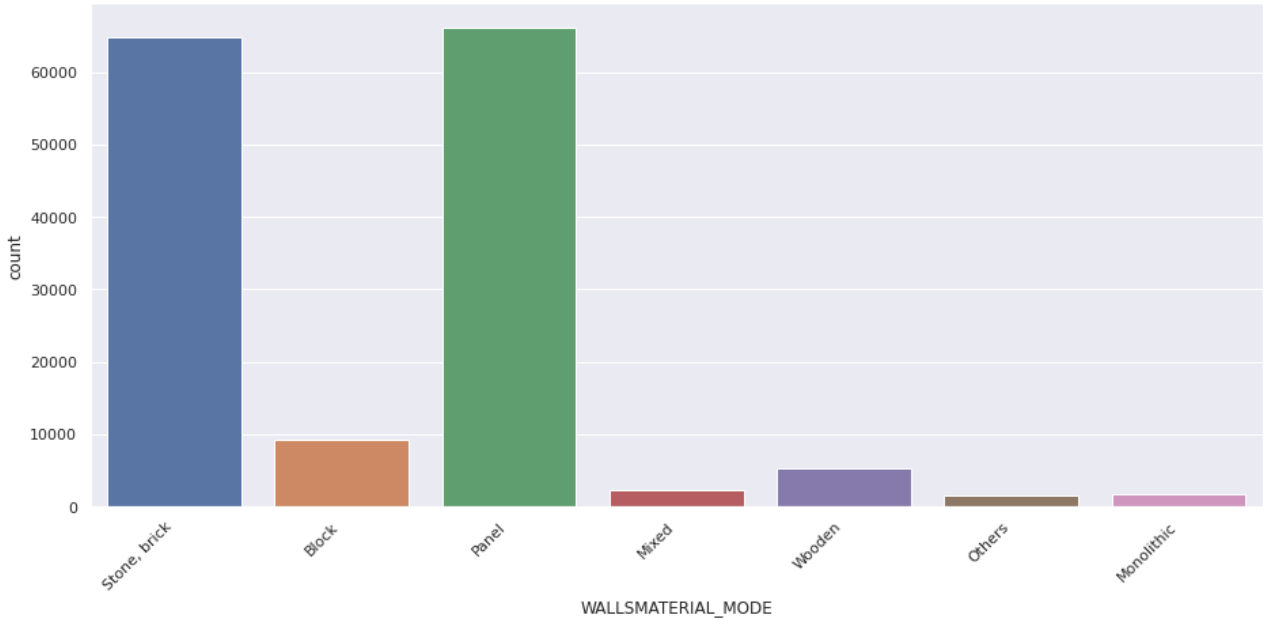

250000
200000
150000
100000
50000
0
count
House / apartment
Rented apartment
With parents
Municipal apartment
Office apartment
Co-op apartment
NAME_HOUSING_TYPE


50000
40000
30000
20000
10000
0
count
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


count
50000
40000
30000
20000
10000
0
WEDNESDAY
MONDAY
THURSDAY
SUNDAY
SATURDAY
FRIDAY
TUESDAY
WEEKDAY_APPR_PROCESS_START

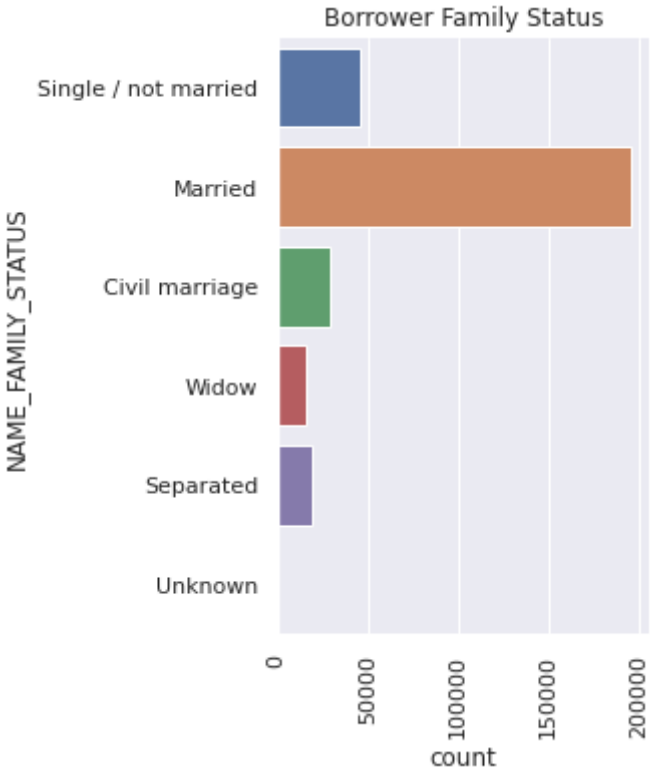

count
70000
60000
50000
40000
30000
20000
10000
0
Business Entity Type 3
School
Government
Religion
Other
XNA
Electricity
Medicine
Business Entity Type 2
Self-employed
Transport: type 2
Construction
Housing
Kindergarten
Trade: type 7
Industry: type 11
Military
Services
Security Ministries
Transport: type 4
Industry: type 1
Emergency
Security
Trade: type 2
University
Transport: type 3
Police
Business Entity Type 1
Industry: type 4
Postal
Agriculture
Restaurant
Culture
Hotel
Industry: type 7
Trade: type 3
Industry: type 3
Bank
Industry: type 9
Insurance
Trade: type 6
Industry: type 2
Transport: type 1
Industry: type 12
Mobile
Trade: type 1
Industry: type 5
Industry: type 10
Legal Services
Advertising
Trade: type 5
Cleaning
Industry: type 13
Trade: type 4
Telecom
Industry: type 8
Realtor
Industry: type 6
ORGANIZATION_TYPE



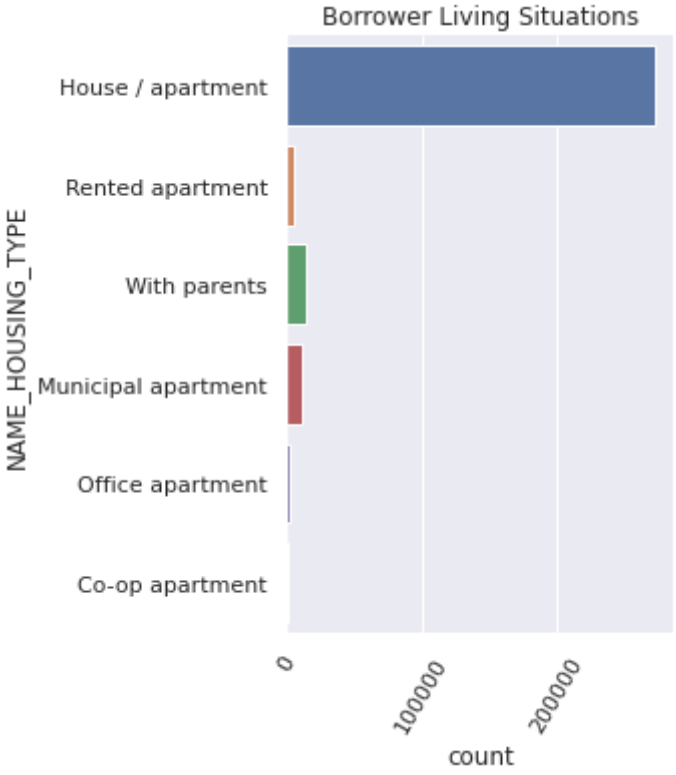


In [ ]:

```
sns.catplot(data =  datasets["application_train"], y = 'NAME_FAMILY_STATUS',kind
plt.xticks(rotation = 90)
plt.title('Borrower Family Status')
plt.show()
```



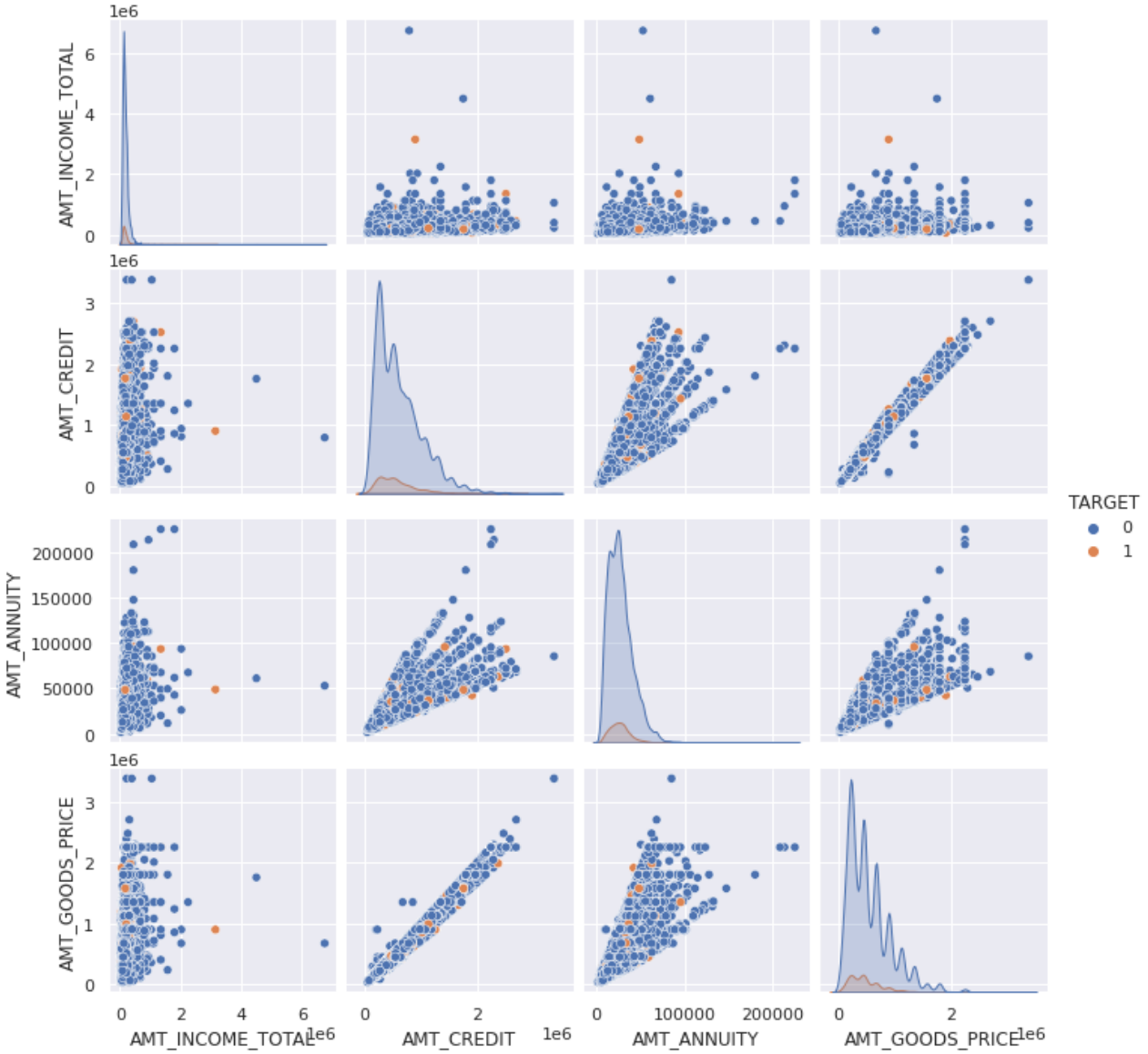
In [ ]:
```
sns.catplot(data= datasets["application_train"], y = 'NAME_HOUSING_TYPE', kind='
plt.xticks(rotation = 60)
plt.title('Borrower Living Situations')
plt.show()
```



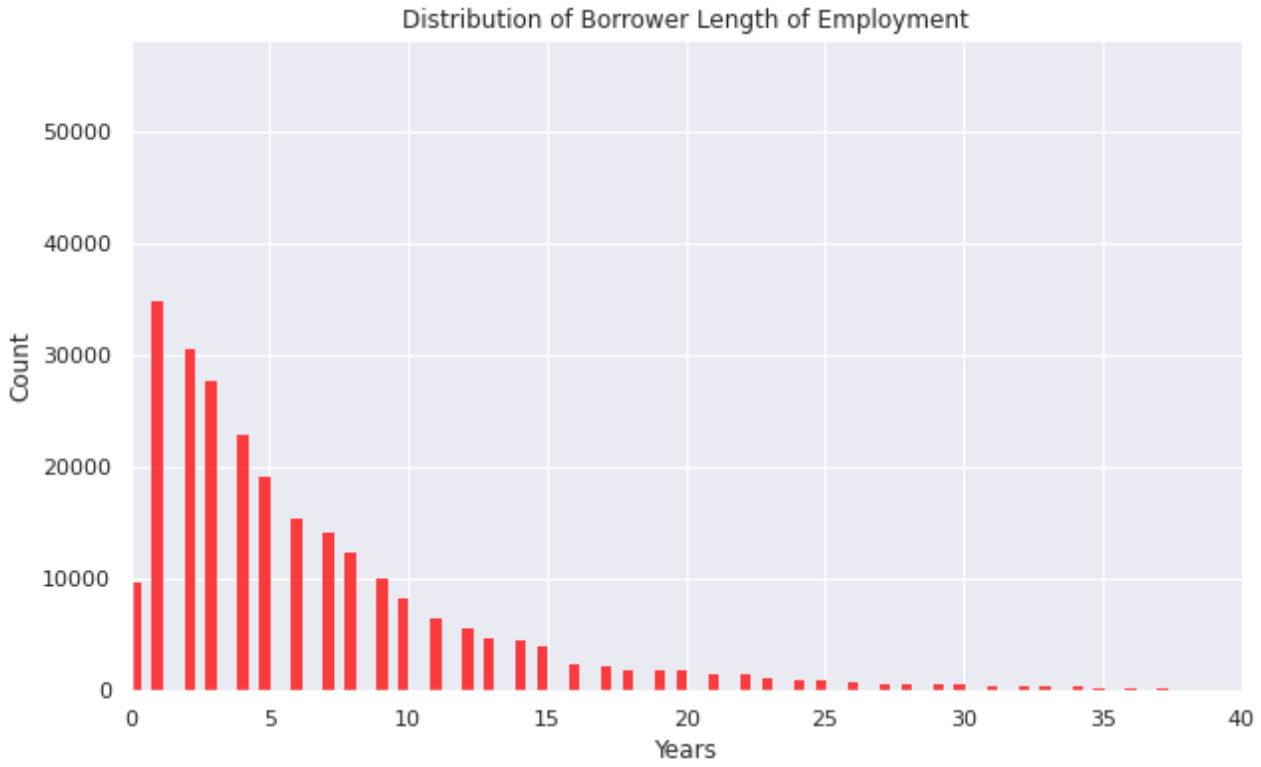
In [ ]:
```
train30k = datasets["application_train"].sample(frac = 0.10, replace = False, ra
```

In [ ]:
```
sns.pairplot(train30k[['TARGET','AMT_INCOME_TOTAL',
                       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE']], hue='TA
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0xfffefd982700>
```



In [ ]:
```
# Employment distribution
X = round(abs(datasets["application_train"]['DAYS_EMPLOYED'] / (365)))
plt.subplots(figsize =(10, 6))
sns.histplot(data= datasets["application_train"], x = X, color = 'red')
plt.title('Distribution of Borrower Length of Employment')
plt.xlabel('Years')
plt.xlim(0, 40)
plt.show()
```



The majority of borrowers less than 15 years of employment experience.

EDA for Bureau

Summary of Bureau

In [ ]:
```
datasets["bureau"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_CURR              int32  
 1   SK_ID_BUREAU            int32  
 2   CREDIT_ACTIVE           object 
 3   CREDIT_CURRENCY         object 
 4   DAYS_CREDIT             int16  
 5   CREDIT_DAY_OVERDUE      int16  
 6   DAYS_CREDIT_ENDDATE     float16
 7   DAYS_ENDDATE_FACT       float16
 8   AMT_CREDIT_MAX_OVERDUE  float32
 9   CNT_CREDIT_PROLONG      int8   
 10  AMT_CREDIT_SUM          float32
 11  AMT_CREDIT_SUM_DEBT     float32
 12  AMT_CREDIT_SUM_LIMIT    float32
 13  AMT_CREDIT_SUM_OVERDUE  float32
 14  CREDIT_TYPE             object 
 15  DAYS_CREDIT_UPDATE      int32  
 16  AMT_ANNUITY             float32
dtypes: float16(2), float32(6), int16(2), int32(3), int8(1), object(3)
memory usage: 112.9+ MB
```

In [ ]:
```
datasets["bureau"].shape
```

Out[ ]:
```
(1716428, 17)
```

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

Out[ ]:

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDI
count	1.716428e+06	1.716428e+06	1.716428e+06	1.716428e+06	1610
mean	2.782149e+05	5.924434e+06	-1.142108e+03	8.181666e-01	
std	1.029386e+05	5.322657e+05	7.951649e+02	3.654443e+01	
min	1.000010e+05	5.000000e+06	-2.922000e+03	0.000000e+00	-42
25%	1.888668e+05	5.463954e+06	-1.666000e+03	0.000000e+00	-1
50%	2.780550e+05	5.926304e+06	-9.870000e+02	0.000000e+00	-3
75%	3.674260e+05	6.385681e+06	-4.740000e+02	0.000000e+00	
max	4.562550e+05	6.843457e+06	0.000000e+00	2.792000e+03	31

In [ ]:
```
datasets["bureau"].describe(include='all') #look at all categorical and numerica
```

Out[ ]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CRE
count	1.716428e+06	1.716428e+06	1716428	1716428	1.716428e+06	
unique	NaN	NaN	4	4	NaN	
top	NaN	NaN	Closed	currency 1	NaN	
freq	NaN	NaN	1079273	1715020	NaN	
mean	2.782149e+05	5.924434e+06	NaN	NaN	-1.142108e+03	
std	1.029386e+05	5.322657e+05	NaN	NaN	7.951649e+02	
min	1.000010e+05	5.000000e+06	NaN	NaN	-2.922000e+03	
25%	1.888668e+05	5.463954e+06	NaN	NaN	-1.666000e+03	
50%	2.780550e+05	5.926304e+06	NaN	NaN	-9.870000e+02	
75%	3.674260e+05	6.385681e+06	NaN	NaN	-4.740000e+02	
max	4.562550e+05	6.843457e+06	NaN	NaN	0.000000e+00	

Missing data for bureau

In [ ]:
```
percent_bureau = (datasets["bureau"].isnull().sum()/datasets["bureau"].isnull().
sum_missing_bureau = datasets["bureau"].isna().sum().sort_values(ascending = Fal
missing_data_bureau  = pd.concat([percent_bureau, sum_missing_bureau], axis=1, k
missing_data_bureau.head(10)
```

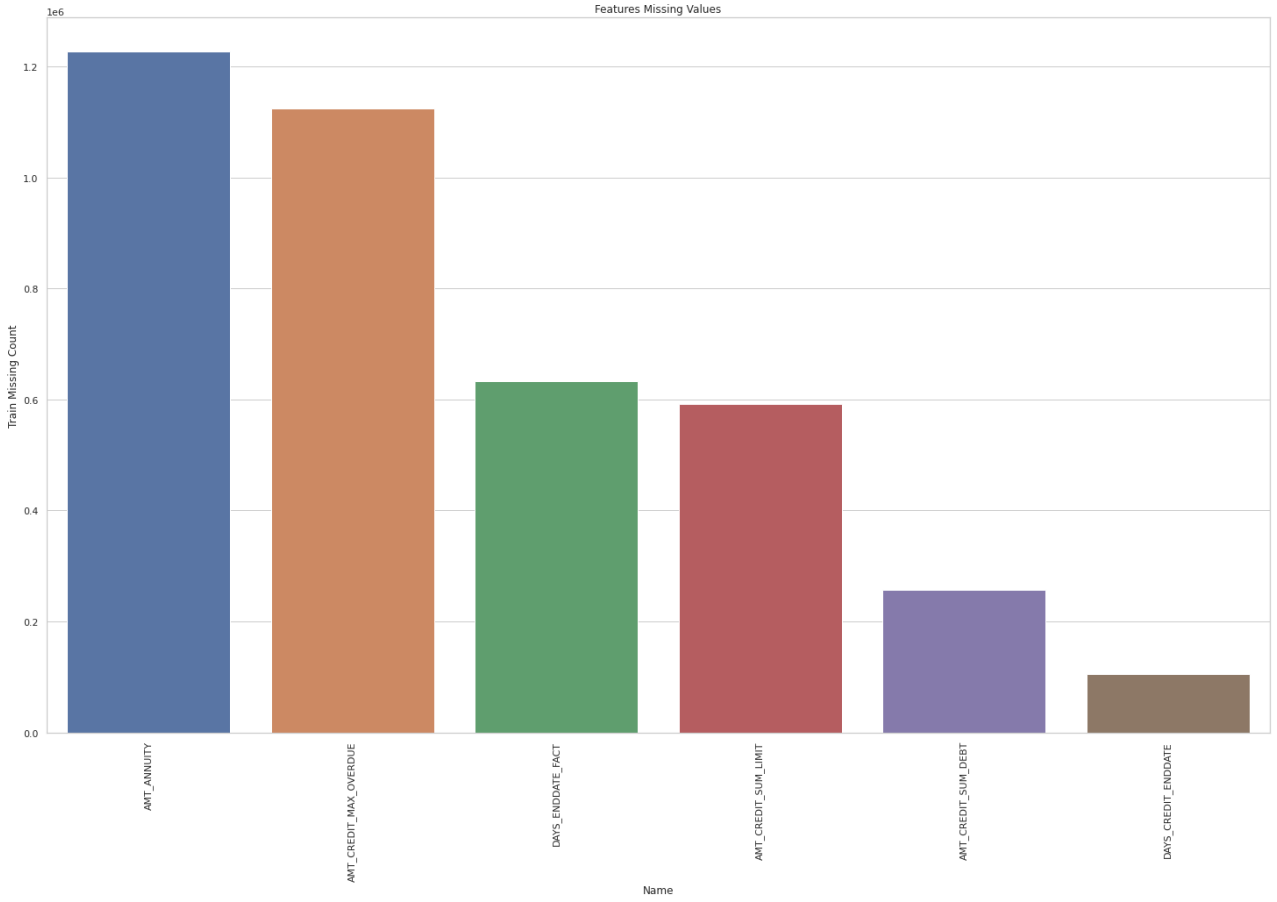
Out[ ]:

	Percent	Train Missing Count
AMT_ANNUITY	71.47	1226791
AMT_CREDIT_MAX_OVERDUE	65.51	1124488
DAYS_ENDDATE_FACT	36.92	633653
AMT_CREDIT_SUM_LIMIT	34.48	591780
AMT_CREDIT_SUM_DEBT	15.01	257669
DAYS_CREDIT_ENDDATE	6.15	105553
AMT_CREDIT_SUM	0.00	13
CREDIT_ACTIVE	0.00	0
CREDIT_CURRENCY	0.00	0
DAYS_CREDIT	0.00	0

In [ ]:
```
missing_data_bureau = pd.DataFrame(
missing_data_bureau[missing_data_bureau['Percent']>0.0])
```

In [ ]:
```
missing_data_bureau.columns.name = 'Name'
missing_data_bureau['Name']=missing_data_bureau.index
```

In [ ]:
```
sns.set(style="whitegrid", color_codes=True,rc={'figure.figsize':(25,15)})
sns.barplot(x = 'Name', y = 'Train Missing Count', data=missing_data_bureau).set
plt.xticks(rotation = 90)
plt.show()
```

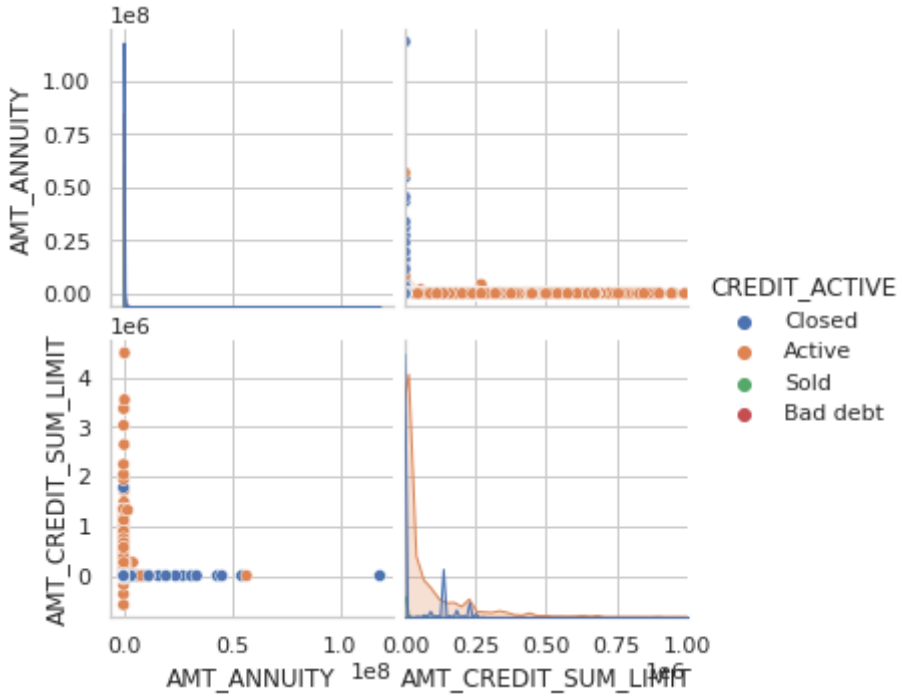


Bureau Missing Values

In [ ]:

```
plt.figure(figsize=(12,15))
ax = sns.pairplot(datasets["bureau"], vars = ['AMT_ANNUITY','AMT_CREDIT_SUM_LIMI
plt.xlim((0,1000000))
plt.show()
```

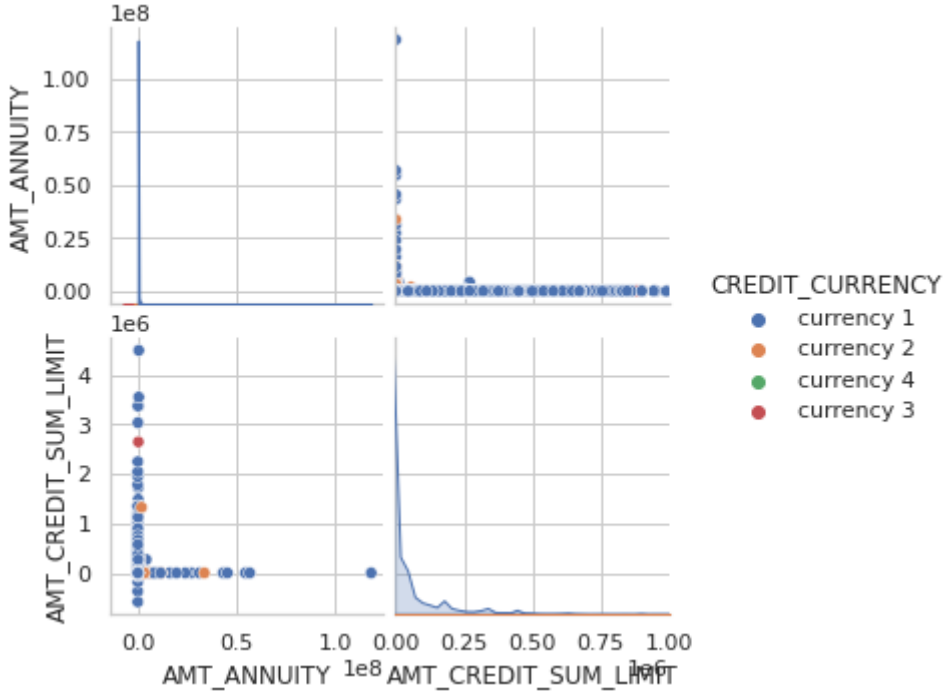
```
<Figure size 864x1080 with 0 Axes>
```



- AMT_ANNUITY and AMT_CREDIT_SUM_LIMIT do not have signficate replationshp.
- Accounting to customers CREDIT_ACTIVE, active is much common than any other options.

In [ ]:
```
#Visualizing the relationships between AMT_ANNUITY,AMT_CREDIT_SUM_LIMIT Vs. CRED
plt.figure(figsize=(12,5))
ax = sns.pairplot(datasets['bureau'], vars = ['AMT_ANNUITY','AMT_CREDIT_SUM_LIMI
plt.xlim((0,1000000))
plt.show()
```

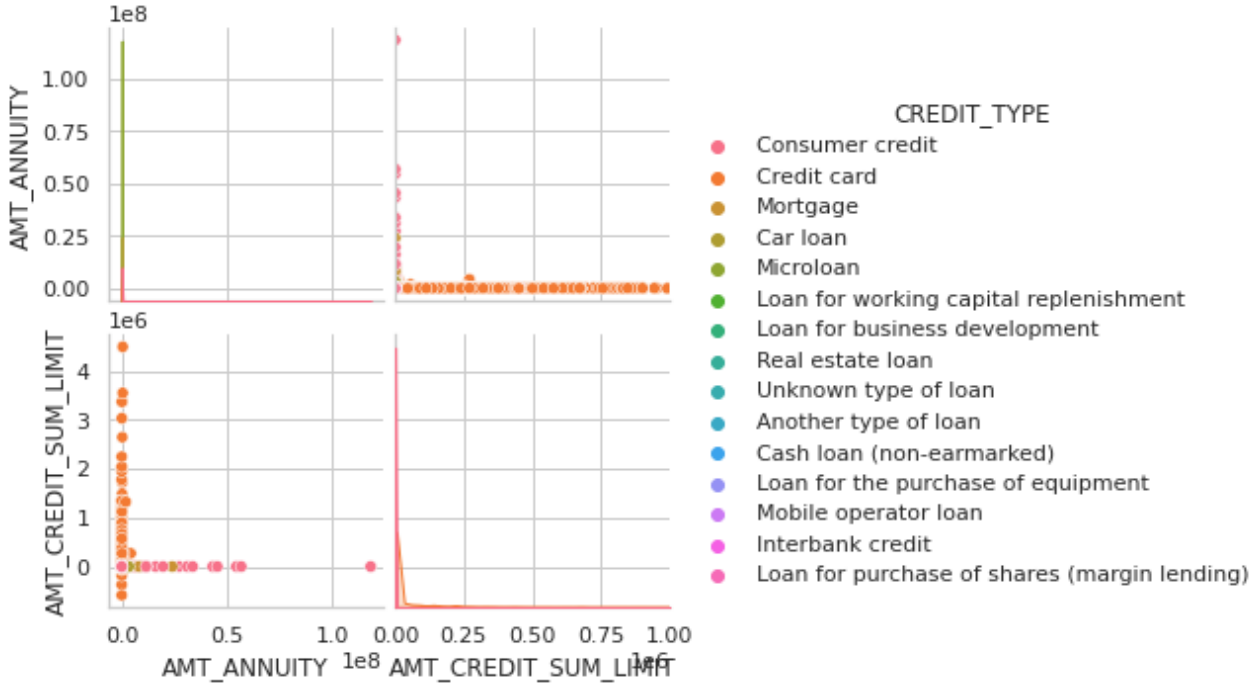
```
<Figure size 864x360 with 0 Axes>
```



- AMT_ANNUITY and AMT_CREDIT_SUM_LIMIT do not have signficate replationshp.
- Accounting customers CREDIT_CURRENCY, currency 1 is much common than any other options.

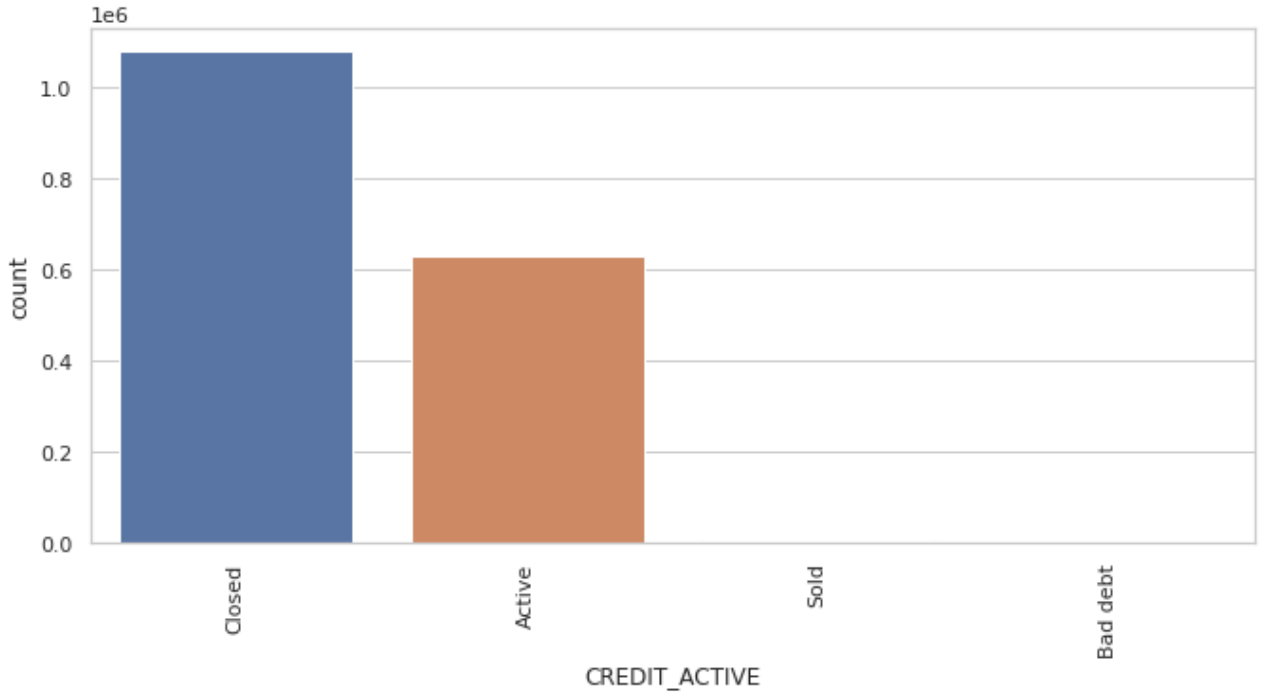
In [ ]:
```
#Visualizing the relationships between AMT_ANNUITY,AMT_CREDIT_SUM_LIMIT Vs. CRED
plt.figure(figsize=(12,5))
ax = sns.pairplot(datasets['bureau'], vars = ['AMT_ANNUITY','AMT_CREDIT_SUM_LIMI
plt.xlim((0,1000000))
plt.show()
```

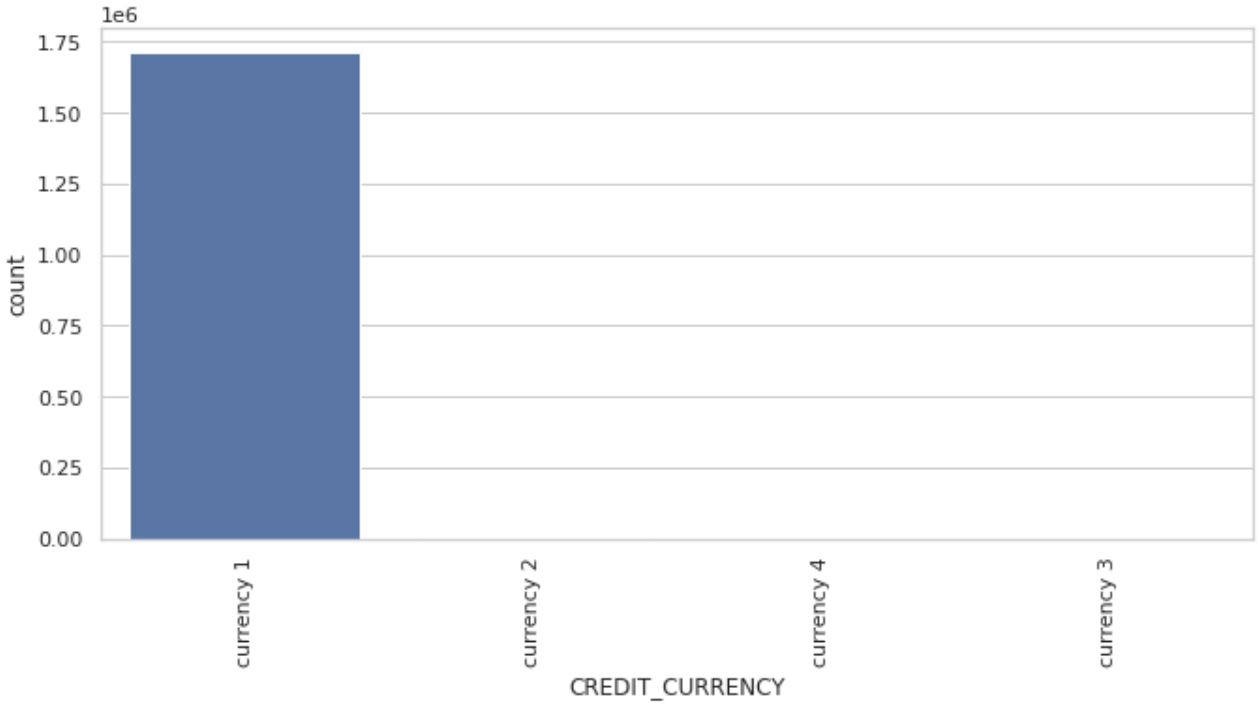
```
<Figure size 864x360 with 0 Axes>
```

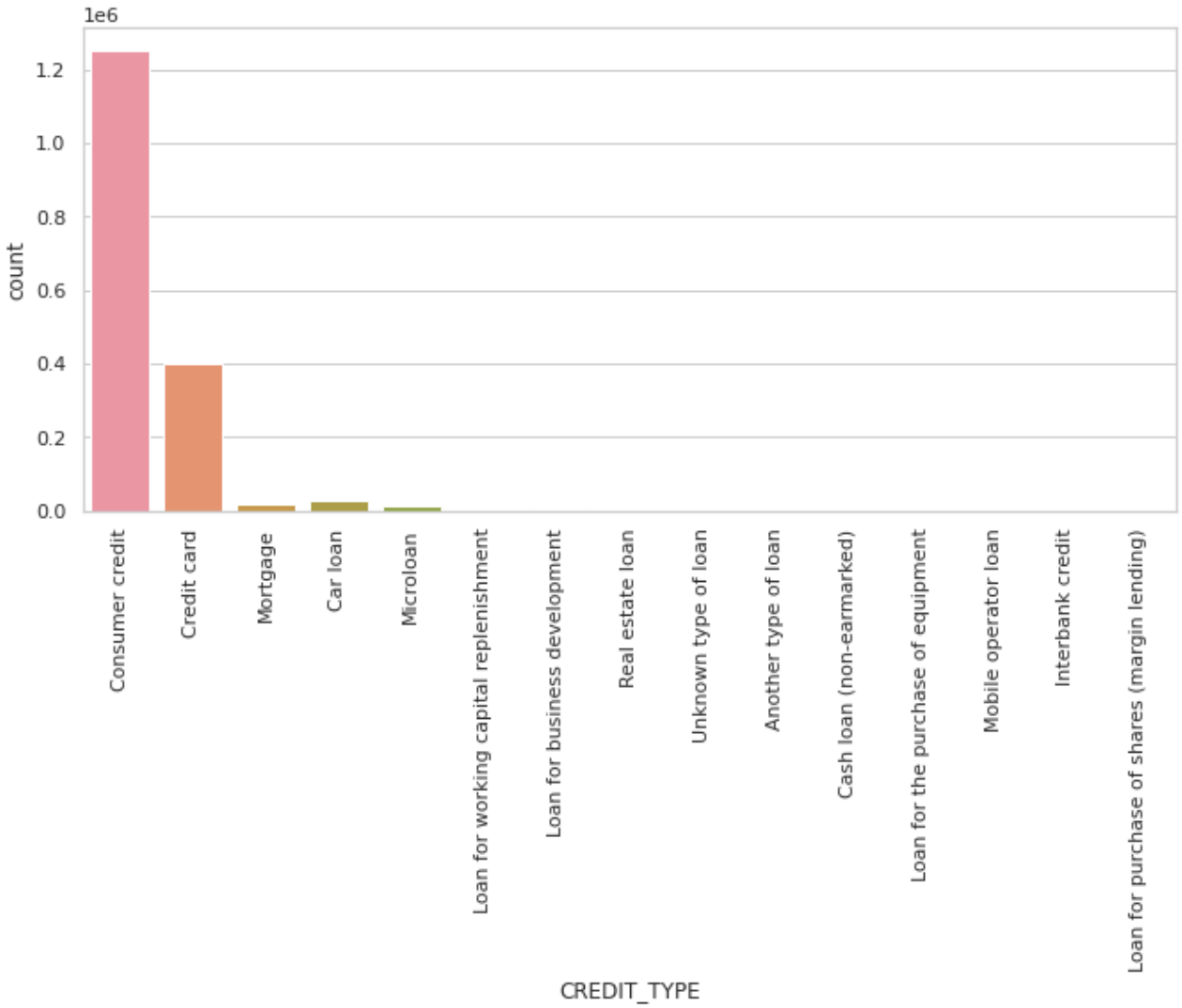


- AMT_ANNUITY and AMT_CREDIT_SUM_LIMIT do not have signficate replationshp.
- Accounting customers CREDIT_TYPE, credit card is much common than any other options.

In [ ]:
```
graph_objects(datasets['bureau'])
```

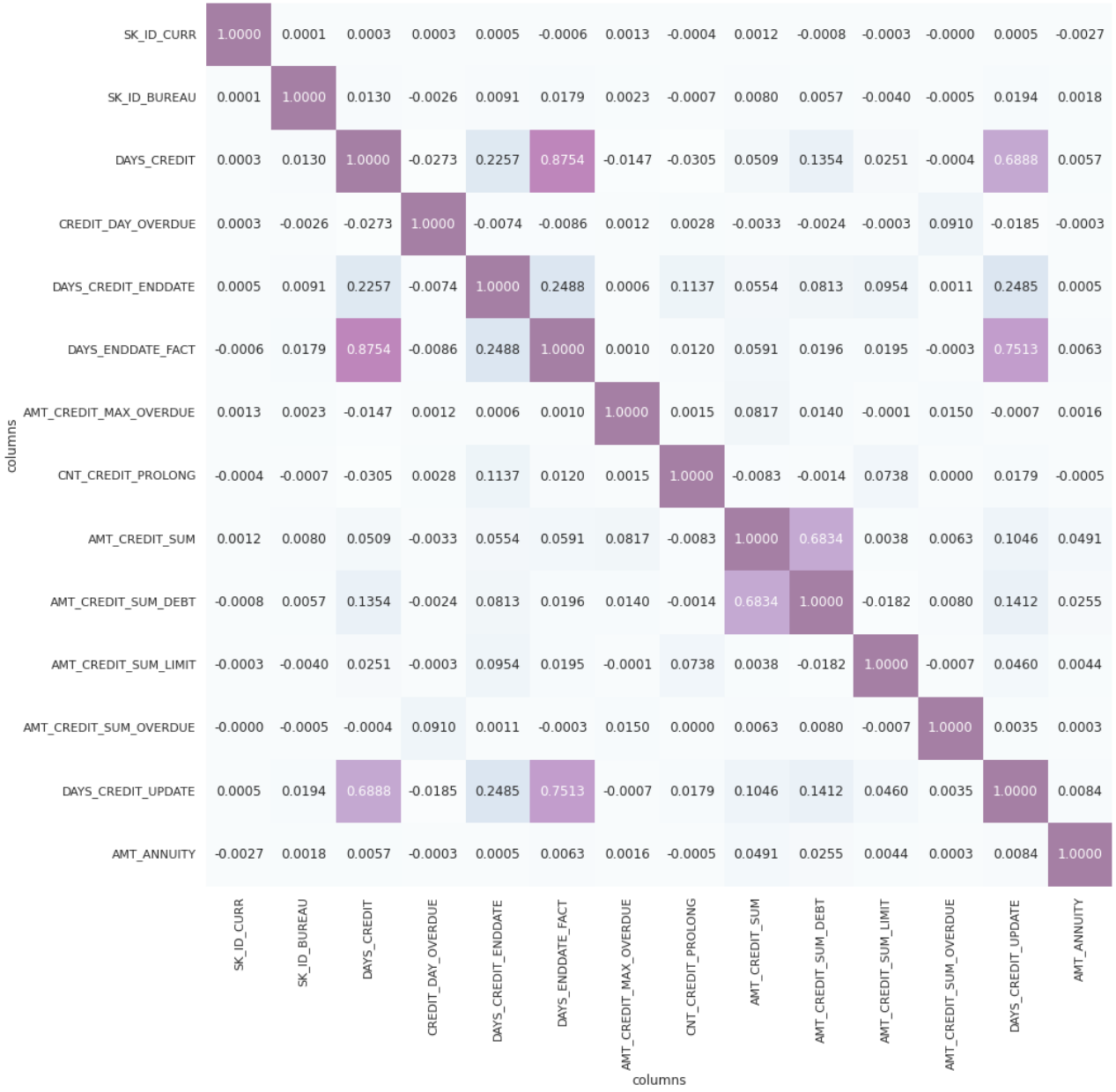






Correlation

In [ ]:
```
corrs = datasets['bureau'].corr()
plt.figure(figsize=(15,15))
sns.heatmap(datasets['bureau'].corr(), annot = True, cmap = "BuPu", alpha = 0.5,
plt.show()

annot = True   #gives you the numbers instead of just the colors.
cmap = "BuPu"  #is a blue/purple color scheme that I like to use
alpha = 0.5    #tones down the intensity of the colors
fmt = ".4f"    #formats the numbers to four decimal places
cbar = False   #turns off the scale bar on the right
```



Column Types

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

Out[ ]:
```
float32    6
int32      3
object     3
int16      2
float16    2
```

```
int8       1
dtype: int64
```

In [ ]:
```
datasets['bureau'].select_dtypes('object').nunique()
```

Out[ ]:
```
columns
CREDIT_ACTIVE       4
CREDIT_CURRENCY     4
CREDIT_TYPE        15
dtype: int64
```

In [ ]:
```
for col in datasets['bureau']:
    if datasets['bureau'][col].dtypes == 'object':
        print(col)
```

```
CREDIT_ACTIVE
CREDIT_CURRENCY
CREDIT_TYPE
```

EDA for Bureau Balance

Summary of Bureau Balance

In [ ]:
```
datasets["bureau_balance"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Data columns (total 3 columns):
 #   Column          Dtype
---  ------          -----
 0   SK_ID_BUREAU    int32
 1   MONTHS_BALANCE  int8
 2   STATUS          object
dtypes: int32(1), int8(1), object(1)
memory usage: 338.5+ MB
```

In [ ]:
```
datasets["bureau_balance"].shape
```

Out[ ]:
```
(27299925, 3)
```

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

Out[ ]:

	SK_ID_BUREAU	MONTHS_BALANCE
count	2.729992e+07	2.729992e+07
mean	6.036297e+06	-3.074169e+01
std	4.923489e+05	2.386451e+01
min	5.001709e+06	-9.600000e+01
25%	5.730933e+06	-4.600000e+01
50%	6.070821e+06	-2.500000e+01
75%	6.431951e+06	-1.100000e+01

	SK_ID_BUREAU	MONTHS_BALANCE
max	6.842888e+06	0.000000e+00

In [ ]:
```
datasets["bureau_balance"].describe(include='all') #look at all categorical and
```

Out[ ]:

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
count	2.729992e+07	2.729992e+07	27299925
unique	NaN	NaN	8
top	NaN	NaN	C
freq	NaN	NaN	13646993
mean	6.036297e+06	-3.074169e+01	NaN
std	4.923489e+05	2.386451e+01	NaN
min	5.001709e+06	-9.600000e+01	NaN
25%	5.730933e+06	-4.600000e+01	NaN
50%	6.070821e+06	-2.500000e+01	NaN
75%	6.431951e+06	-1.100000e+01	NaN
max	6.842888e+06	0.000000e+00	NaN

Missing data for bureau

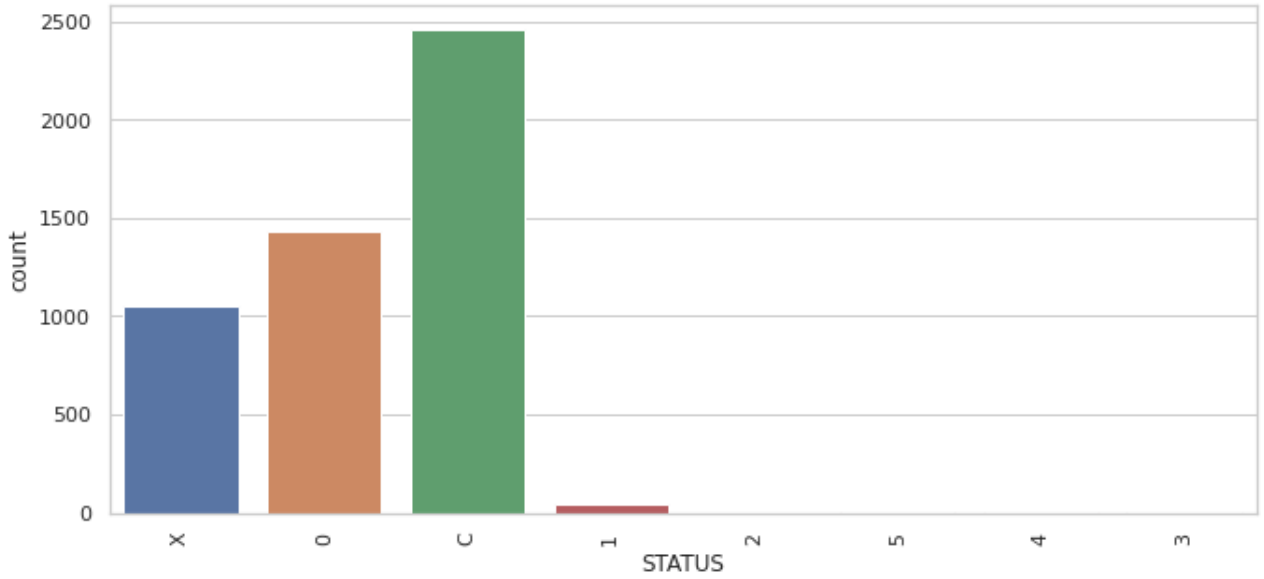
In [ ]:
```
percent_bureau_balance = (datasets["bureau_balance"].isnull().sum()/datasets["bu
sum_missing_bureau_balance = datasets["bureau_balance"].isna().sum().sort_values
missing_data_bureau_balance  = pd.concat([percent_bureau_balance, sum_missing_bu
missing_data_bureau_balance.head(10)
```

Out[ ]:

	Percent	Train Missing Count
SK_ID_BUREAU	0.0	0
MONTHS_BALANCE	0.0	0
STATUS	0.0	0

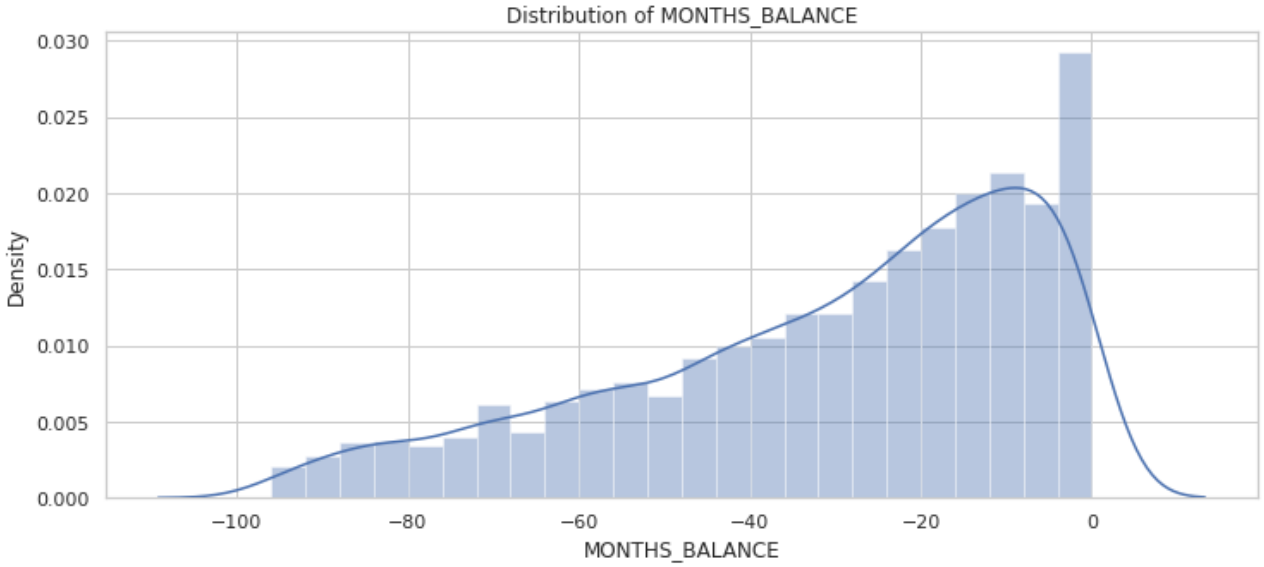
In [ ]:
```
bureau_bal_5K = datasets["bureau_balance"].sample(n=5000, random_state=1)
```

In [ ]:
```
graph_objects(bureau_bal_5K)
```



In [ ]:

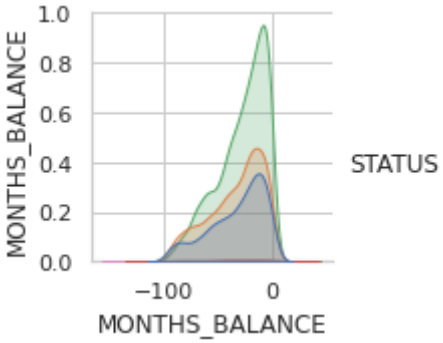
```
#Show MONTHS_BALANCE Distribution in bureau balance Data
plt.figure(figsize=(12,5))
plt.title("Distribution of MONTHS_BALANCE")
ax = sns.distplot(bureau_bal_5K.MONTHS_BALANCE.dropna())
plt.show()
```



In [ ]:

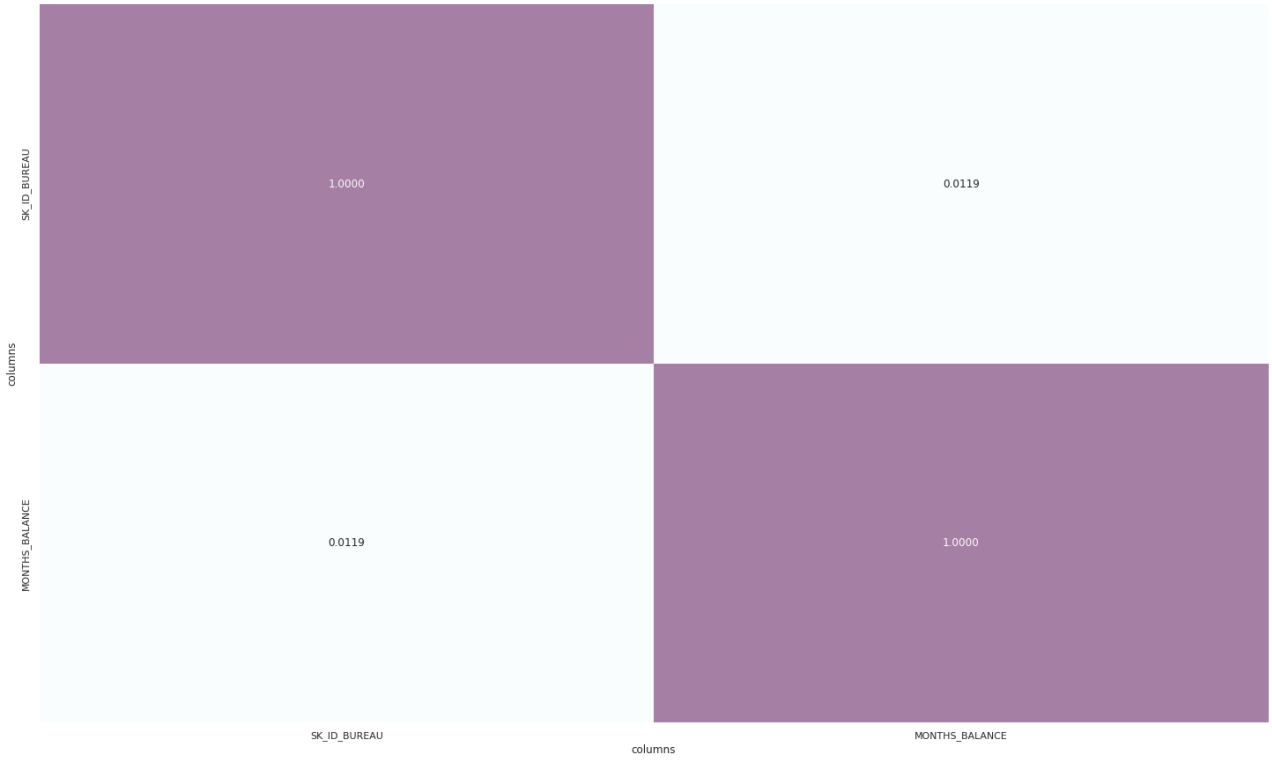
```
plt.figure(figsize=(100,15))
ax = sns.pairplot(bureau_bal_5K, vars = ['MONTHS_BALANCE'],hue='STATUS');
plt.show()
```

```
<Figure size 7200x1080 with 0 Axes>
```



Correlation

In [ ]:
```
corrs = datasets['bureau_balance'].corr()
sns.heatmap(datasets['bureau_balance'].corr(), annot = True, cmap = "BuPu", alph
plt.show()

annot = True   #gives you the numbers instead of just the colors.
cmap = "BuPu"  #is a blue/purple color scheme that I like to use
alpha = 0.5    #tones down the intensity of the colors
fmt = ".4f"    #formats the numbers to four decimal places
cbar = False   #turns off the scale bar on the right
```



Column Types

In [ ]:
```
datasets['bureau_balance'].dtypes.value_counts()
```

Out[ ]:
```
int32     1
int8      1
object    1
dtype: int64
```

In [ ]:
```
datasets['bureau_balance'].select_dtypes('object').nunique()
```

Out[ ]:
```
columns
STATUS    8
dtype: int64
```

In [ ]:
```
for col in datasets['bureau_balance']:
    if datasets['bureau_balance'][col].dtypes == 'object':
        print(col)
```

```
STATUS
```

EDA for Credit Card Balance

Summary of Credit Card Balance

In [ ]:
```
datasets["credit_card_balance"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column                       Dtype  
---  ------                       -----  
 0   SK_ID_PREV                   int32  
 1   SK_ID_CURR                   int32  
 2   MONTHS_BALANCE               int8   
 3   AMT_BALANCE                  float32
 4   AMT_CREDIT_LIMIT_ACTUAL      int32  
 5   AMT_DRAWINGS_ATM_CURRENT     float32
 6   AMT_DRAWINGS_CURRENT         float32
 7   AMT_DRAWINGS_OTHER_CURRENT   float32
 8   AMT_DRAWINGS_POS_CURRENT     float32
 9   AMT_INST_MIN_REGULARITY      float32
 10  AMT_PAYMENT_CURRENT          float32
 11  AMT_PAYMENT_TOTAL_CURRENT    float32
 12  AMT_RECEIVABLE_PRINCIPAL     float32
 13  AMT_RECIVABLE                float32
 14  AMT_TOTAL_RECEIVABLE         float32
 15  CNT_DRAWINGS_ATM_CURRENT     float16
 16  CNT_DRAWINGS_CURRENT         int16  
 17  CNT_DRAWINGS_OTHER_CURRENT   float16
 18  CNT_DRAWINGS_POS_CURRENT     float16
 19  CNT_INSTALMENT_MATURE_CUM    float16
 20  NAME_CONTRACT_STATUS         object 
 21  SK_DPD                       int16  
 22  SK_DPD_DEF                   int16  
dtypes: float16(4), float32(11), int16(3), int32(3), int8(1), object(1)
memory usage: 289.3+ MB
```

In [ ]:
```
datasets["credit_card_balance"].shape
```

Out[ ]:
```
(3840312, 23)
```

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

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACT

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACT
count	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.840312
mean	1.904504e+06	2.783242e+05	-3.452192e+01	5.830016e+04	1.538080
std	5.364695e+05	1.027045e+05	2.666775e+01	1.063070e+05	1.651457
min	1.000018e+06	1.000060e+05	-9.600000e+01	-4.202502e+05	0.000000
25%	1.434385e+06	1.895170e+05	-5.500000e+01	0.000000e+00	4.500000
50%	1.897122e+06	2.783960e+05	-2.800000e+01	0.000000e+00	1.125000
75%	2.369328e+06	3.675800e+05	-1.100000e+01	8.904669e+04	1.800000
max	2.843496e+06	4.562500e+05	-1.000000e+00	1.505902e+06	1.350000

8 rows × 22 columns

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

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_AC
count	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.84031
unique	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	
mean	1.904504e+06	2.783242e+05	-3.452192e+01	5.830016e+04	1.53808
std	5.364695e+05	1.027045e+05	2.666775e+01	1.063070e+05	1.65145
min	1.000018e+06	1.000060e+05	-9.600000e+01	-4.202502e+05	0.00000
25%	1.434385e+06	1.895170e+05	-5.500000e+01	0.000000e+00	4.50000
50%	1.897122e+06	2.783960e+05	-2.800000e+01	0.000000e+00	1.12500
75%	2.369328e+06	3.675800e+05	-1.100000e+01	8.904669e+04	1.80000
max	2.843496e+06	4.562500e+05	-1.000000e+00	1.505902e+06	1.35000

11 rows × 23 columns

Missing data for credit_card_balance

In [ ]:
```
percent_credit_card_balance = (datasets["credit_card_balance"].isnull().sum()/da
sum_missing_credit_card_balance = datasets["credit_card_balance"].isna().sum().s
missing_data_credit_card_balance  = pd.concat([percent_credit_card_balance, sum_
missing_data_credit_card_balance.head(10)
```

Out[ ]:

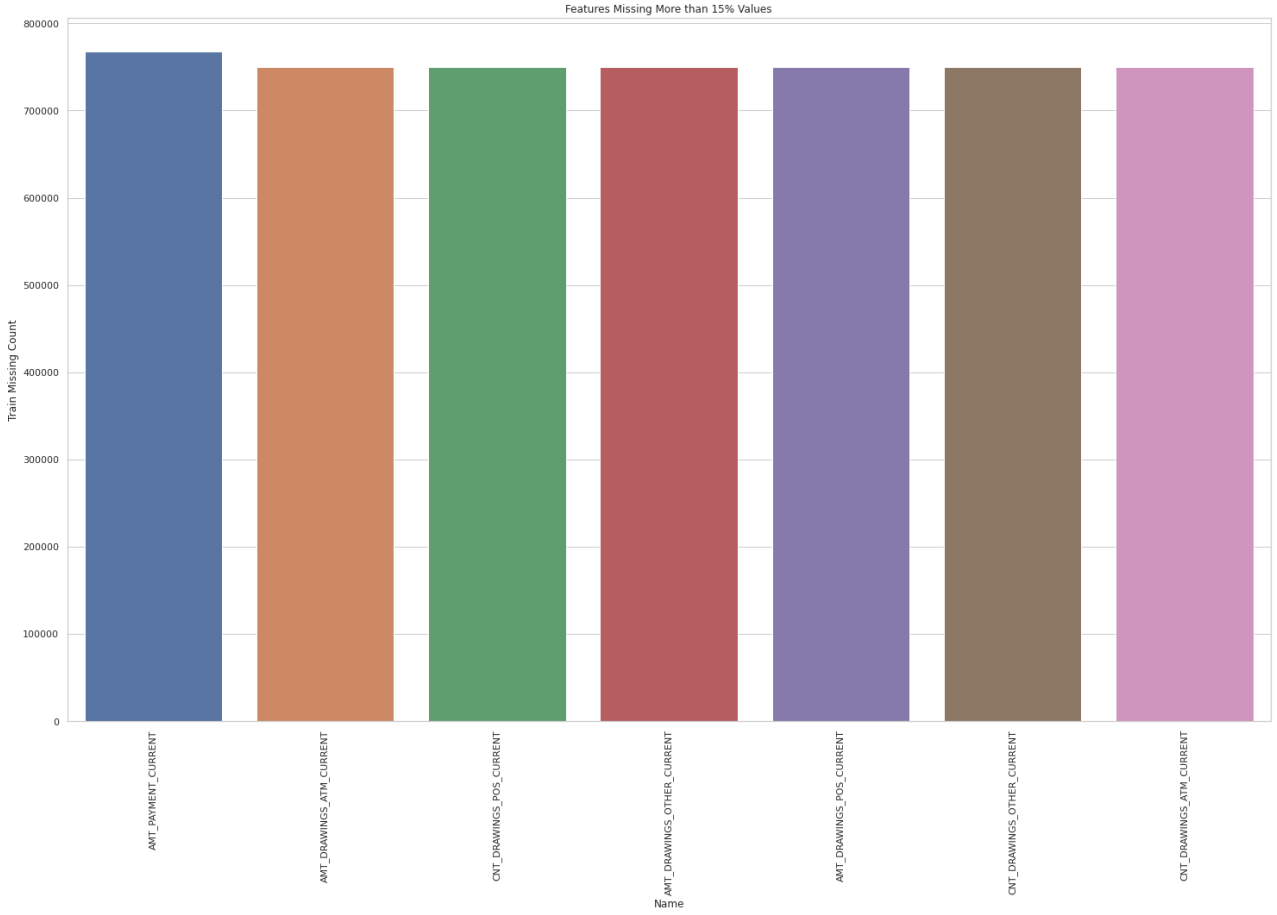
	Percent	Train Missing Count

	Percent	Train Missing Count
AMT_PAYMENT_CURRENT	20.00	767988
AMT_DRAWINGS_ATM_CURRENT	19.52	749816
CNT_DRAWINGS_POS_CURRENT	19.52	749816
AMT_DRAWINGS_OTHER_CURRENT	19.52	749816
AMT_DRAWINGS_POS_CURRENT	19.52	749816
CNT_DRAWINGS_OTHER_CURRENT	19.52	749816
CNT_DRAWINGS_ATM_CURRENT	19.52	749816
CNT_INSTALMENT_MATURE_CUM	7.95	305236
AMT_INST_MIN_REGULARITY	7.95	305236
SK_ID_PREV	0.00	0

In [ ]:
```
missing_data_credit_card_balance = pd.DataFrame(
missing_data_credit_card_balance[missing_data_credit_card_balance['Percent']>15.
```

In [ ]:
```
missing_data_credit_card_balance.columns.name = 'Name'
missing_data_credit_card_balance['Name']=missing_data_credit_card_balance.index
```

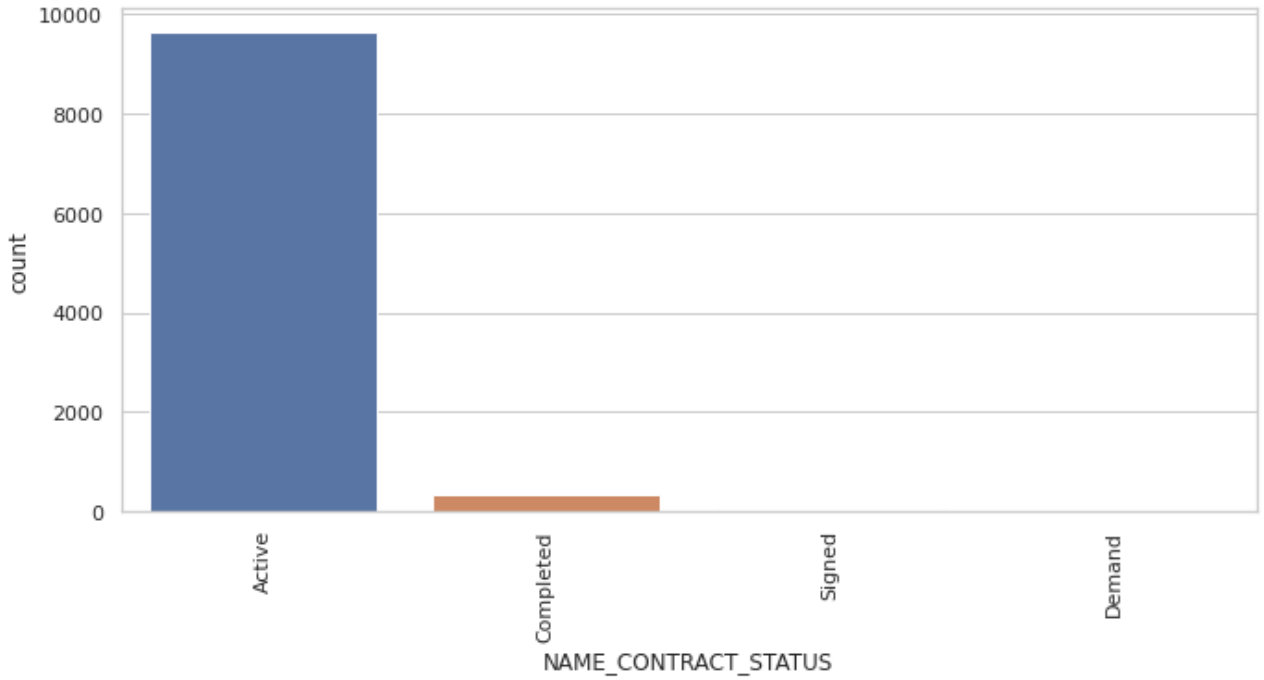
In [ ]:
```
sns.set(style="whitegrid", color_codes=True,rc={'figure.figsize':(25,15)})
sns.barplot(x = 'Name', y = 'Train Missing Count', data=missing_data_credit_card
plt.xticks(rotation = 90)
plt.show()
```



In [ ]:
```
credit_card_bal_10K = datasets["credit_card_balance"].sample(n=10000, random_sta
```

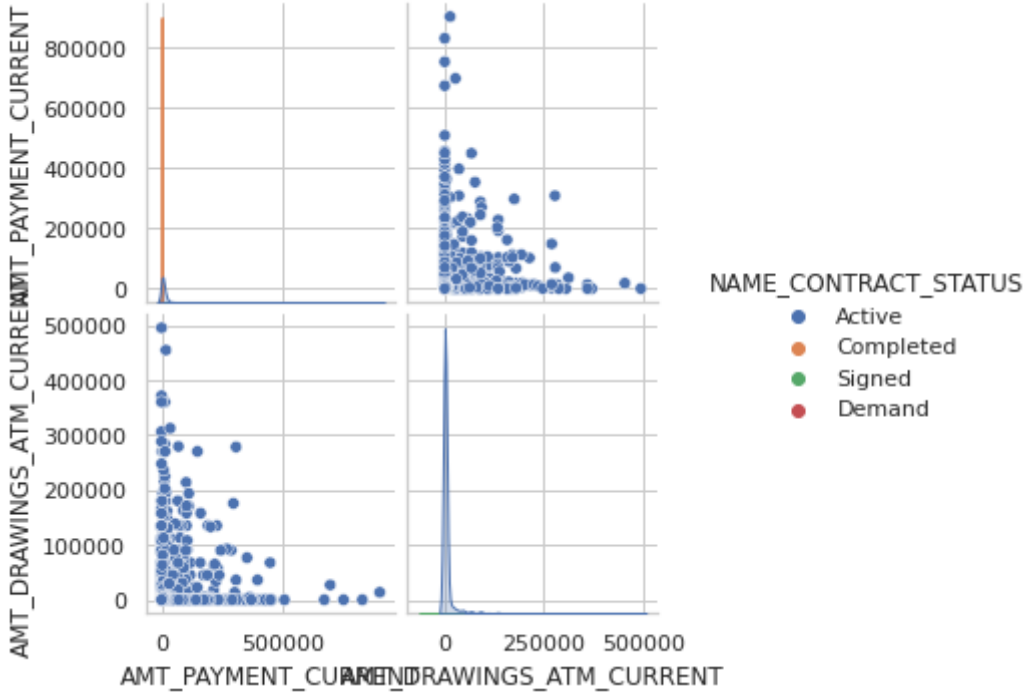
Plotting Count Plots for Categorical Attributes

In [ ]:
```
graph_objects(credit_card_bal_10K)
```

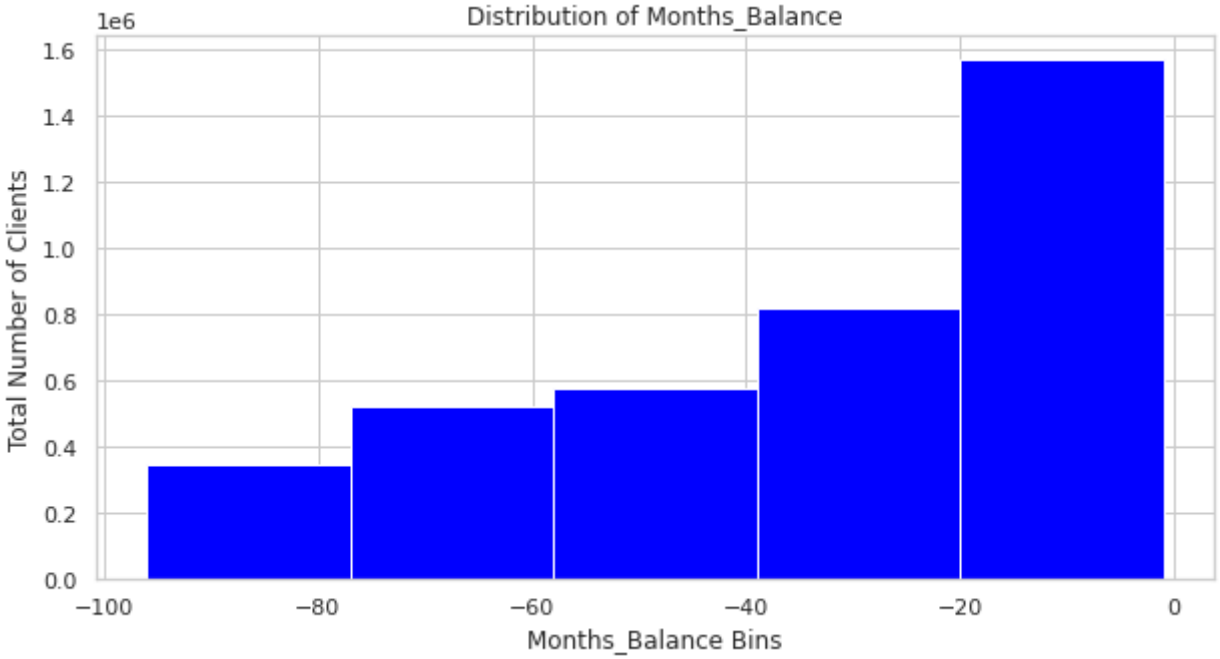


In [ ]:
```
#Visualizing the relationships between AMT_PAYMENT_CURRENT and AMT_DRAWINGS_ATM_
plt.figure(figsize=(20,20))
ax = sns.pairplot(credit_card_bal_10K, vars = ['AMT_PAYMENT_CURRENT','AMT_DRAWIN
plt.show()
```

```
<Figure size 1440x1440 with 0 Axes>
```



In [ ]:
```
plt.figure(figsize=(10,5))
plt.hist(datasets['credit_card_balance'][['MONTHS_BALANCE']].values, bins=5,colo
plt.title('Distribution of Months_Balance')
plt.xlabel('Months_Balance Bins')
plt.ylabel('Total Number of Clients')
plt.show()
```



Correlation

In [ ]:
```
corrs = datasets['credit_card_balance'].corr()
sns.heatmap(datasets['credit_card_balance'].corr(), annot = True, cmap = "BuPu",
plt.show()

annot = True   #gives you the numbers instead of just the colors.
cmap = "BuPu"  #is a blue/purple color scheme that I like to use
alpha = 0.5    #tones down the intensity of the colors
fmt = ".4f"    #formats the numbers to four decimal places
cbar = False   #turns off the scale bar on the right
```

Column Types

In [ ]:
```
datasets['credit_card_balance'].dtypes.value_counts()
```

Out[ ]:
```
float32    11
float16     4
int32       3
int16       3
int8        1
object      1
dtype: int64
```

In [ ]:
```
datasets['credit_card_balance'].select_dtypes('object').nunique()
```

Out[ ]:
```
columns
NAME_CONTRACT_STATUS    7
dtype: int64
```

EDA for Installments Balance

Summary of Installments Balance

In [ ]:
```
datasets["installments_payments"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_PREV              int32  
 1   SK_ID_CURR              int32  
 2   NUM_INSTALMENT_VERSION  float16
 3   NUM_INSTALMENT_NUMBER   int16  
 4   DAYS_INSTALMENT         float16
 5   DAYS_ENTRY_PAYMENT      float16
 6   AMT_INSTALMENT          float32
 7   AMT_PAYMENT             float32
dtypes: float16(3), float32(2), int16(1), int32(2)
memory usage: 311.4 MB
```

In [ ]:
```
datasets["installments_payments"].shape
```

Out[ ]:
```
(13605401, 8)
```

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

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	D
count	1.360540e+07	1.360540e+07	13605401.0	1.360540e+07	
mean	1.903365e+06	2.784449e+05	NaN	1.887090e+01	
std	5.362029e+05	1.027183e+05	0.0	2.666407e+01	
min	1.000001e+06	1.000010e+05	0.0	1.000000e+00	
25%	1.434191e+06	1.896390e+05	0.0	4.000000e+00	
50%	1.896520e+06	2.786850e+05	1.0	8.000000e+00	
75%	2.369094e+06	3.675300e+05	1.0	1.900000e+01	
max	2.843499e+06	4.562550e+05	178.0	2.770000e+02	

In [ ]:
```
datasets["installments_payments"].describe(include='all') #look at all categoric
```

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	D
count	1.360540e+07	1.360540e+07	13605401.0	1.360540e+07	
mean	1.903365e+06	2.784449e+05	NaN	1.887090e+01	
std	5.362029e+05	1.027183e+05	0.0	2.666407e+01	
min	1.000001e+06	1.000010e+05	0.0	1.000000e+00	
25%	1.434191e+06	1.896390e+05	0.0	4.000000e+00	

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	D
50%	1.896520e+06	2.786850e+05	1.0	8.000000e+00	
75%	2.369094e+06	3.675300e+05	1.0	1.900000e+01	
max	2.843499e+06	4.562550e+05	178.0	2.770000e+02	

Missing data for Installments payments

In [ ]:
```
percent_installments_payments = (datasets["installments_payments"].isnull().sum(
sum_missing_installments_payments = datasets["installments_payments"].isna().sum
missing_data_installments_payments  = pd.concat([percent_installments_payments,
missing_data_installments_payments.head(10)
```

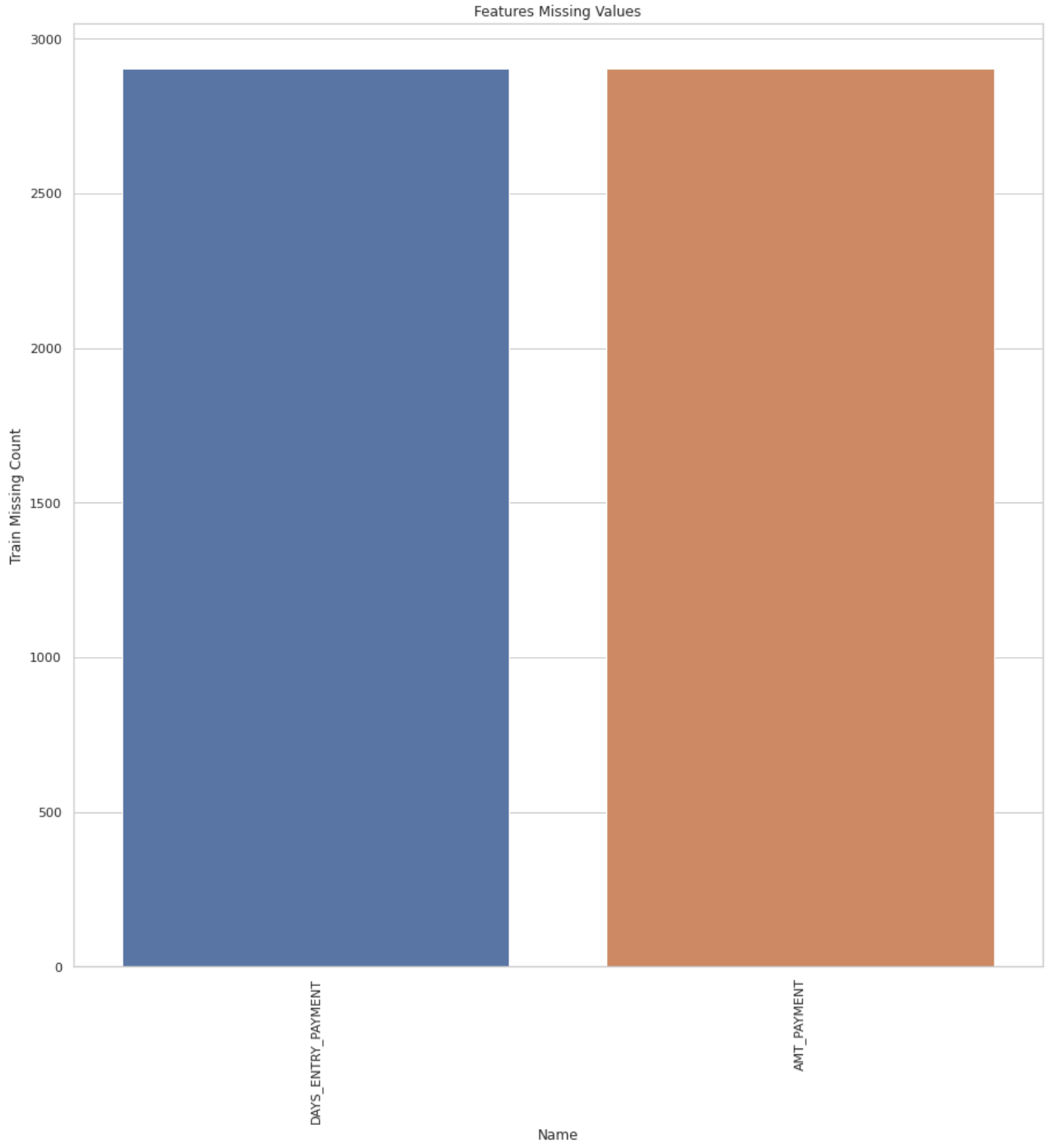
Out[ ]:

	Percent	Train Missing Count
DAYS_ENTRY_PAYMENT	0.02	2905
AMT_PAYMENT	0.02	2905
SK_ID_PREV	0.00	0
SK_ID_CURR	0.00	0
NUM_INSTALMENT_VERSION	0.00	0
NUM_INSTALMENT_NUMBER	0.00	0
DAYS_INSTALMENT	0.00	0
AMT_INSTALMENT	0.00	0

In [ ]:
```
missing_data_installments_payments = pd.DataFrame(
missing_data_installments_payments[missing_data_installments_payments['Percent']
```

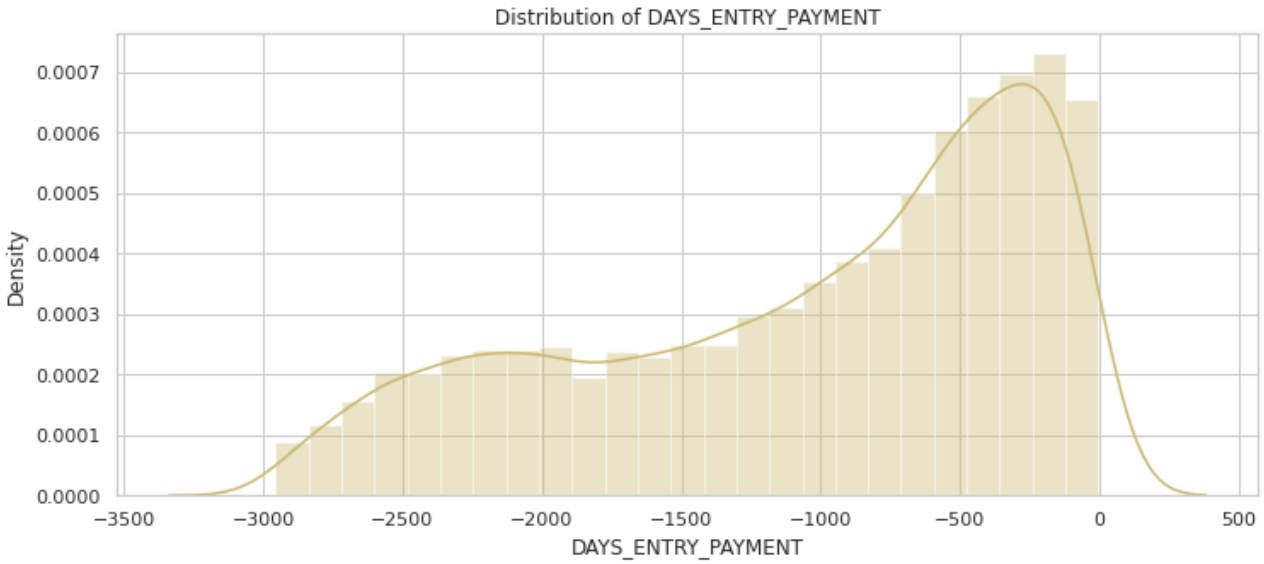
In [ ]:
```
missing_data_installments_payments.columns.name = 'Name'
missing_data_installments_payments['Name']=missing_data_installments_payments.in
```

In [ ]:
```
sns.set(style="whitegrid", color_codes=True,rc={'figure.figsize':(15,15)})
sns.barplot(x = 'Name', y = 'Train Missing Count', data=missing_data_installment
plt.xticks(rotation = 90)
plt.show()
```

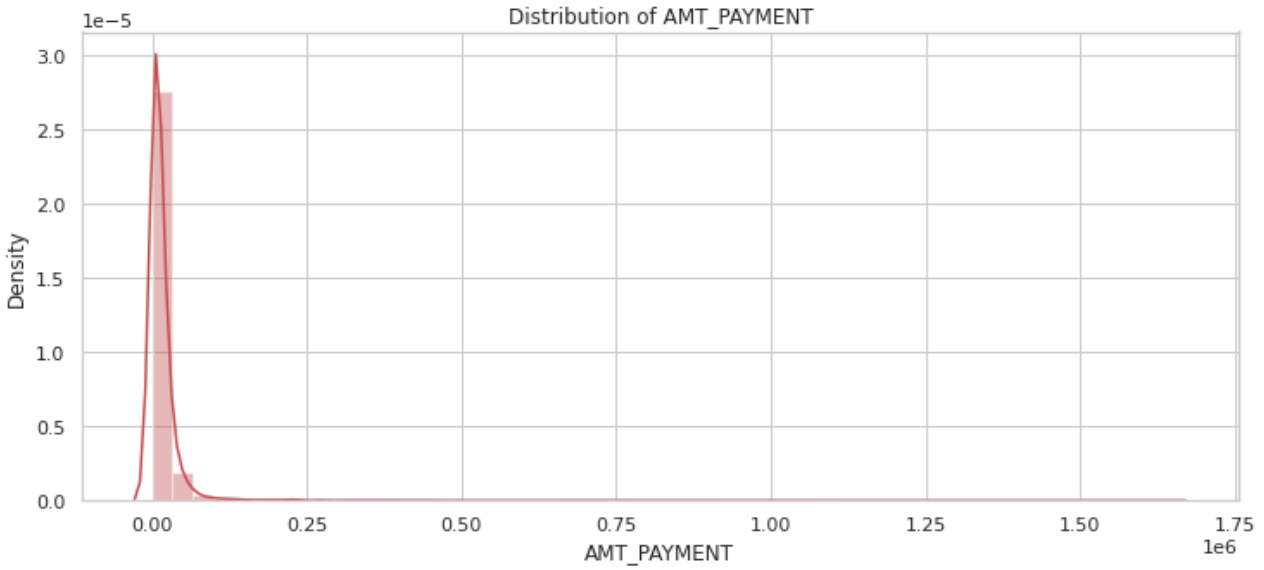


In [ ]:
```
installments_payments_10K = datasets["installments_payments"].sample(n=10000, ra
```

In [ ]:
```
#Show DAYS_ENTRY_PAYMENT Distribution in installements payments Data
plt.figure(figsize=(12,5))
plt.title("Distribution of DAYS_ENTRY_PAYMENT")
ax = sns.distplot(installments_payments_10K.DAYS_ENTRY_PAYMENT.dropna(),color="y
plt.show()
```

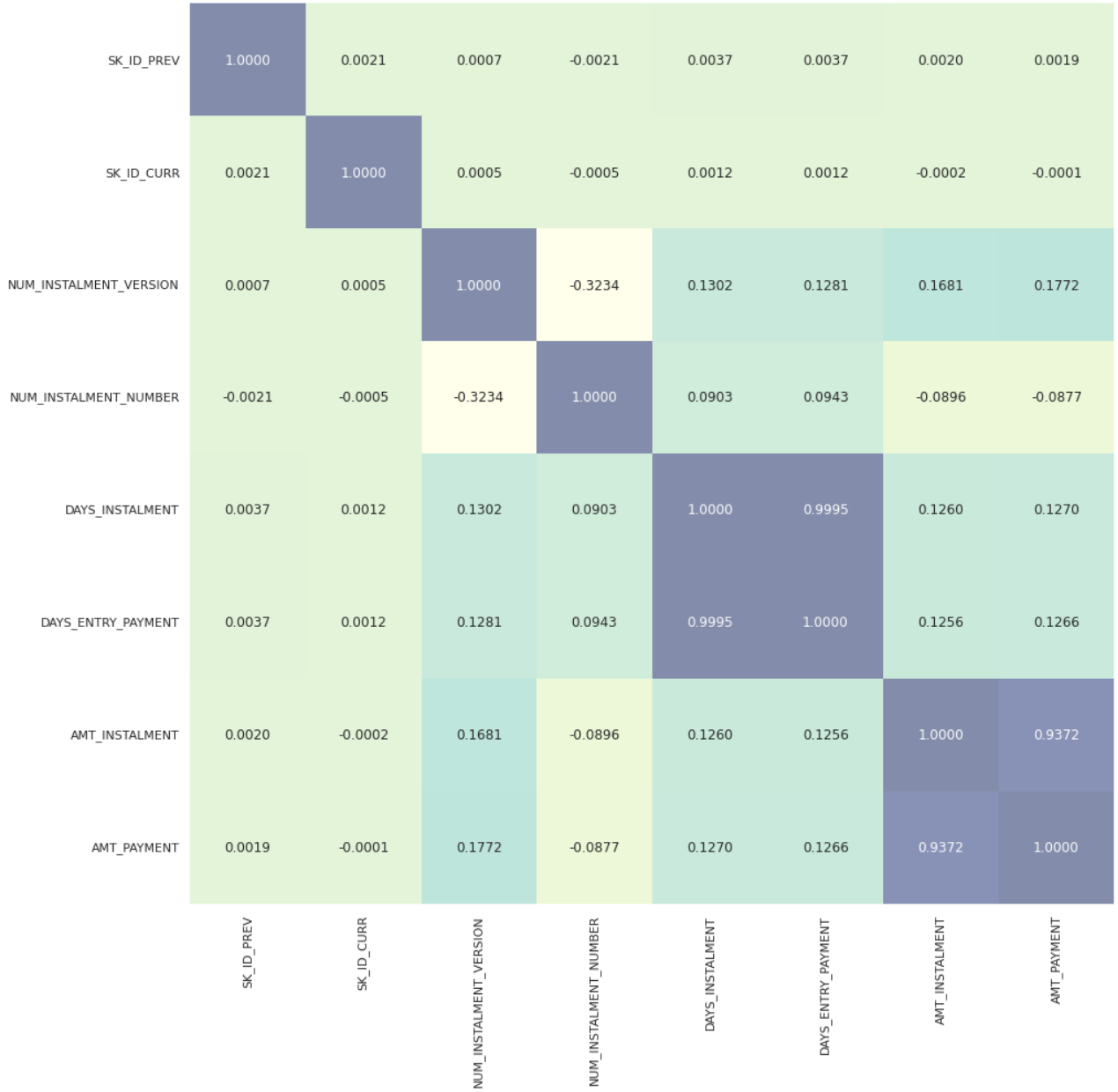


In [ ]:
```
#Show AMT_PAYMENT Distribution in installements payments Data
plt.figure(figsize=(12,5))
plt.title("Distribution of AMT_PAYMENT")
ax = sns.distplot(installments_payments_10K.AMT_PAYMENT.dropna(),color="r")
plt.show()
```



Correlation

In [ ]:
```
corrs = datasets['installments_payments'].corr()
sns.heatmap(datasets['installments_payments'].corr(), annot = True, cmap="YlGnBu
plt.show()

annot = True   #gives you the numbers instead of just the colors.
cmap = "YlGnBu"  #is a yellow/green/blue color scheme that I like to use
alpha = 0.5    #tones down the intensity of the colors
fmt = ".4f"    #formats the numbers to four decimal places
cbar = False   #turns off the scale bar on the right
```



Column Types

In [ ]:
```
datasets['installments_payments'].dtypes.value_counts()
```

Out[ ]:
```
float16    3
int32      2
float32    2
int16      1
dtype: int64
```

In [ ]:
```
datasets['installments_payments'].select_dtypes('object').nunique()
```

Out[ ]:
```
Series([], dtype: float64)
```

EDA for Previous Application

Summary of Previous Application

In [ ]:
```
datasets["previous_application"].info()
```

```
<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  int32  
 1   SK_ID_CURR                   1670214 non-null  int32  
 2   NAME_CONTRACT_TYPE           1670214 non-null  object 
 3   AMT_ANNUITY                  1297979 non-null  float32
 4   AMT_APPLICATION              1670214 non-null  float32
 5   AMT_CREDIT                   1670213 non-null  float32
 6   AMT_DOWN_PAYMENT             774370 non-null   float32
 7   AMT_GOODS_PRICE              1284699 non-null  float32
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object 
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int8   
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object 
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int8   
 12  RATE_DOWN_PAYMENT            774370 non-null   float16
 13  RATE_INTEREST_PRIMARY        5951 non-null     float16
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float16
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object 
 16  NAME_CONTRACT_STATUS         1670214 non-null  object 
 17  DAYS_DECISION                1670214 non-null  int16  
 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  int32  
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object 
 28  CNT_PAYMENT                  1297984 non-null  float16
 29  NAME_YIELD_GROUP             1670214 non-null  object 
 30  PRODUCT_COMBINATION          1669868 non-null  object 
 31  DAYS_FIRST_DRAWING           997149 non-null   float32
 32  DAYS_FIRST_DUE               997149 non-null   float32
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float32
 34  DAYS_LAST_DUE                997149 non-null   float32
 35  DAYS_TERMINATION             997149 non-null   float32
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float16
dtypes: float16(5), float32(10), int16(1), int32(3), int8(2), object(16)
memory usage: 309.0+ MB
```

In [ ]:
```
datasets["previous_application"].shape
```

Out[ ]:
```
(1670214, 37)
```

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

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DO\
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DO
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	
25%	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	
75%	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	
max	2.845382e+06	4.562550e+05	4.180582e+05	6.905160e+06	6.905160e+06	

8 rows × 21 columns

In [ ]:
```
datasets["previous_application"].describe(include='all') #look at all categorica
```

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION
count	1.670214e+06	1.670214e+06	1670214	1.297979e+06	1.670214e+06
unique	NaN	NaN	4	NaN	NaN
top	NaN	NaN	Cash loans	NaN	NaN
freq	NaN	NaN	747553	NaN	NaN
mean	1.923089e+06	2.783572e+05	NaN	1.595512e+04	1.752339e+05
std	5.325980e+05	1.028148e+05	NaN	1.478214e+04	2.927798e+05
min	1.000001e+06	1.000010e+05	NaN	0.000000e+00	0.000000e+00
25%	1.461857e+06	1.893290e+05	NaN	6.321780e+03	1.872000e+04
50%	1.923110e+06	2.787145e+05	NaN	1.125000e+04	7.104600e+04
75%	2.384280e+06	3.675140e+05	NaN	2.065842e+04	1.803600e+05
max	2.845382e+06	4.562550e+05	NaN	4.180582e+05	6.905160e+06

11 rows × 37 columns

Missing data for Previous Application

In [ ]:
```
percent_previous_application = (datasets["previous_application"].isnull().sum()/
sum_missing_previous_application = datasets["previous_application"].isna().sum()
missing_previous_application  = pd.concat([percent_previous_application, sum_mis
missing_previous_application.head(10)
```

Out[ ]:

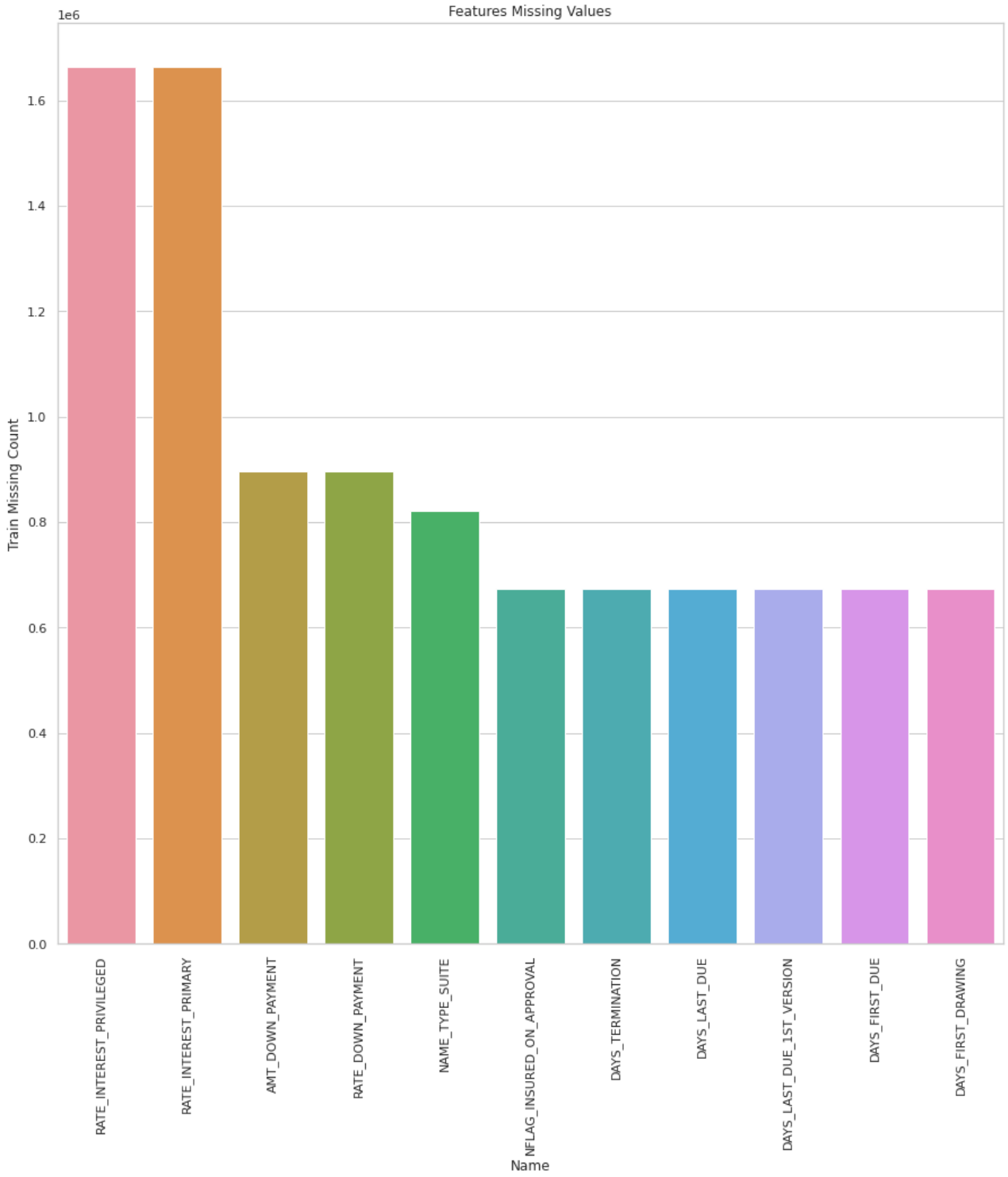
	Percent	Train Missing Count
RATE_INTEREST_PRIVILEGED	99.64	1664263
RATE_INTEREST_PRIMARY	99.64	1664263
AMT_DOWN_PAYMENT	53.64	895844

	Percent	Train Missing Count
RATE_DOWN_PAYMENT	53.64	895844
NAME_TYPE_SUITE	49.12	820405
NFLAG_INSURED_ON_APPROVAL	40.30	673065
DAYS_TERMINATION	40.30	673065
DAYS_LAST_DUE	40.30	673065
DAYS_LAST_DUE_1ST_VERSION	40.30	673065
DAYS_FIRST_DUE	40.30	673065

In [ ]:
```
missing_previous_application = pd.DataFrame(
missing_previous_application[missing_previous_application['Percent']>40.0])
```

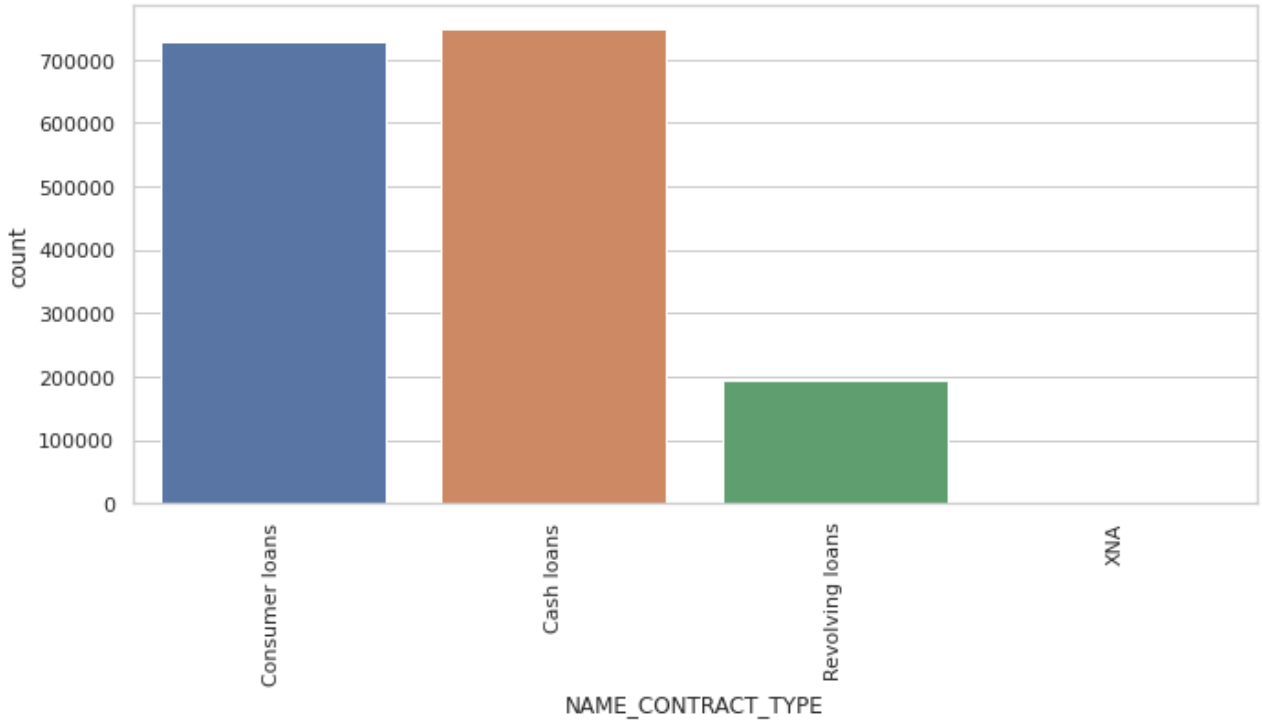
In [ ]:
```
missing_previous_application.columns.name = 'Name'
missing_previous_application['Name']=missing_previous_application.index
```

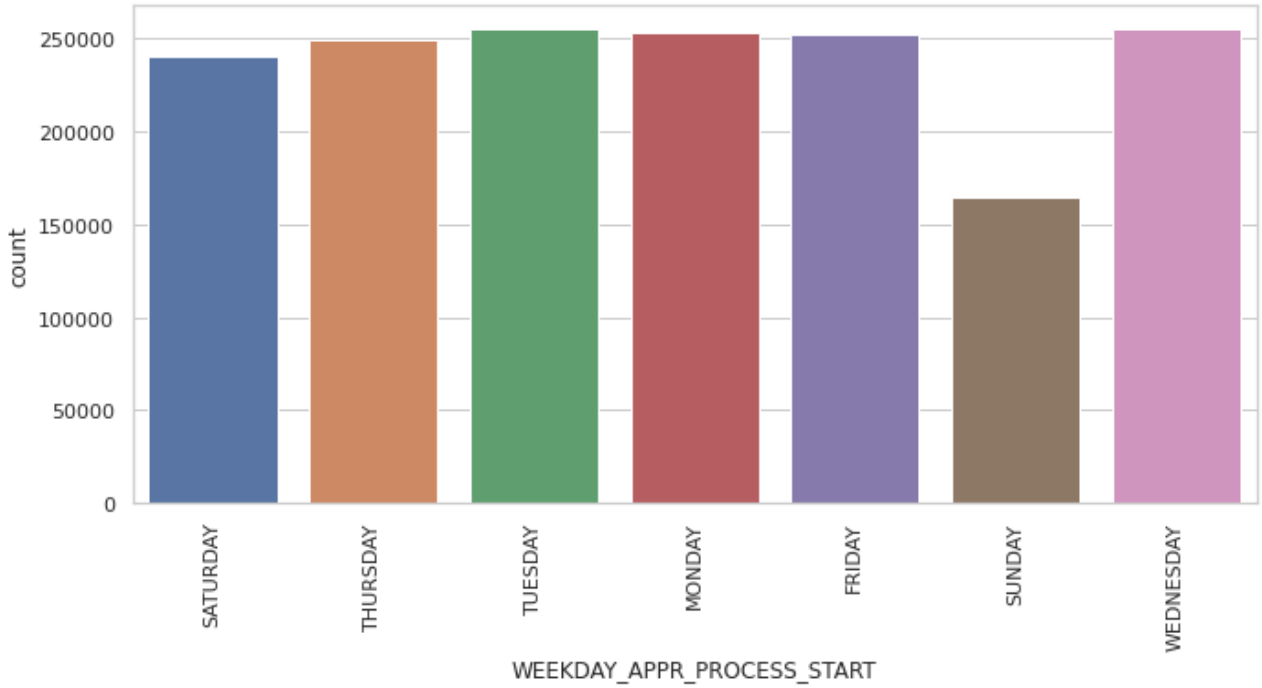
In [ ]:
```
sns.set(style="whitegrid", color_codes=True,rc={'figure.figsize':(15,15)})
sns.barplot(x = 'Name', y = 'Train Missing Count', data=missing_previous_applica
plt.xticks(rotation = 90)
plt.show()
```

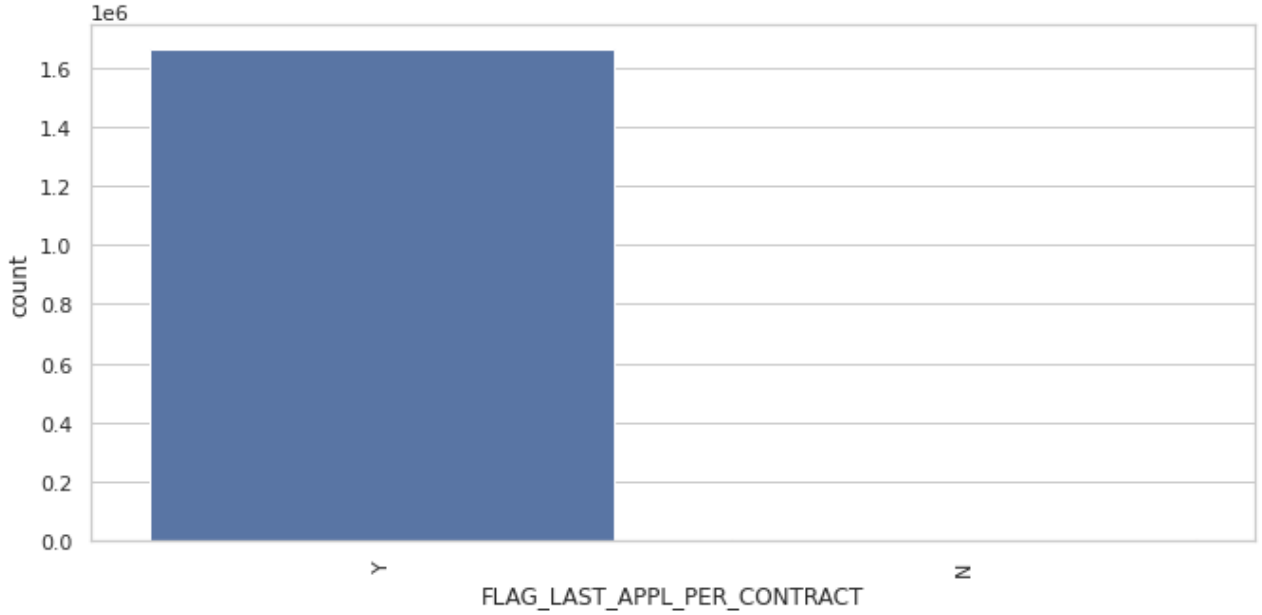


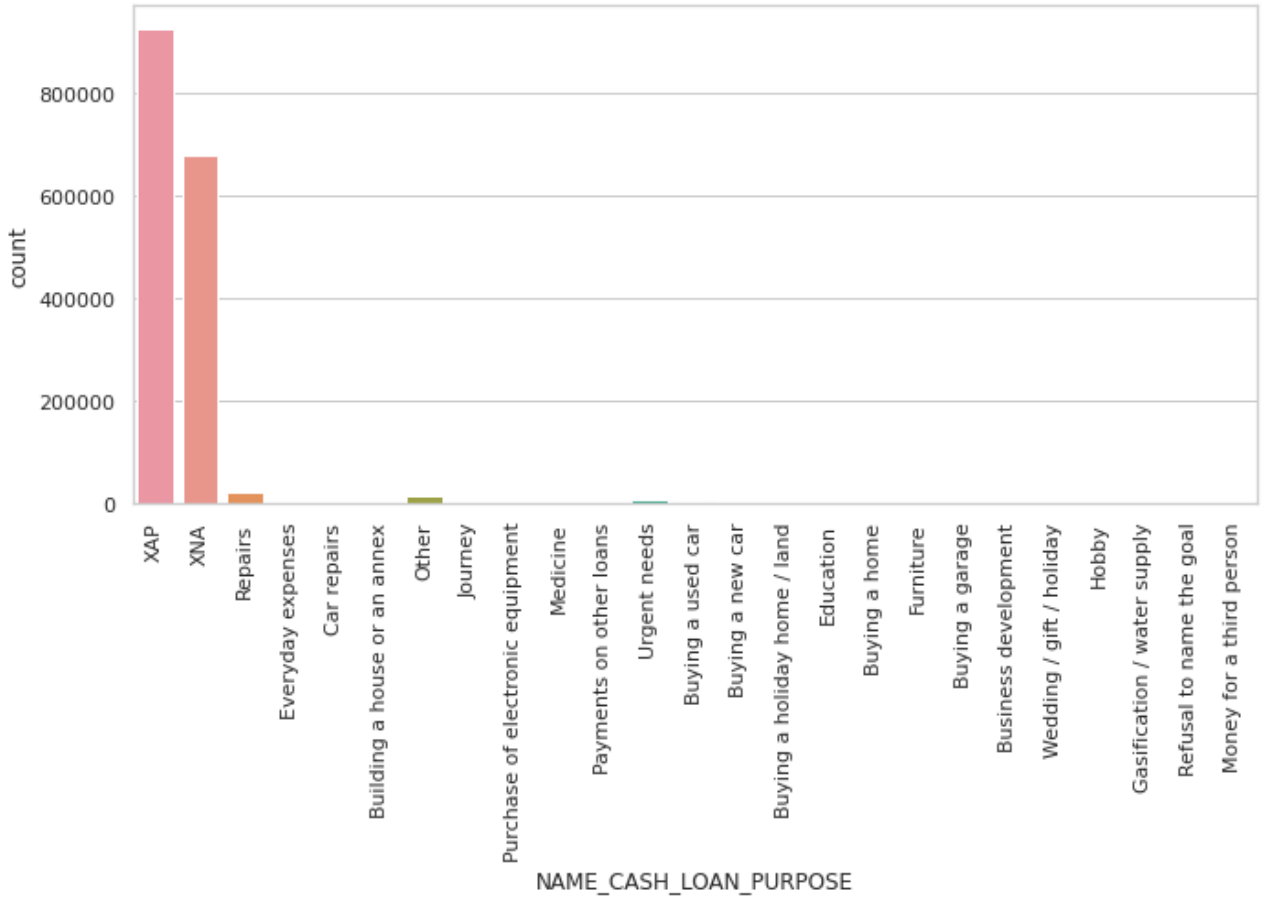
In [ ]:
```
previous_application_10K = datasets["previous_application"].sample(n=10000, rand
```

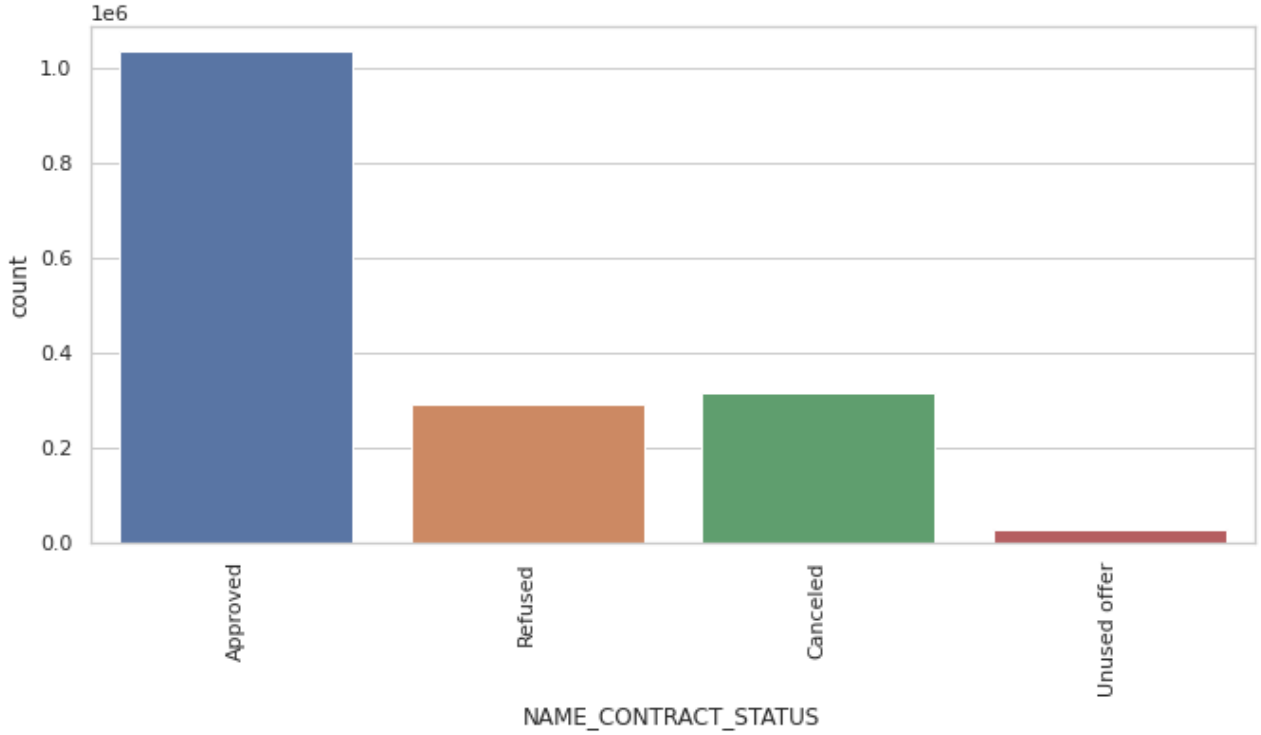
In [ ]:
```
graph_objects(datasets["previous_application"])
```

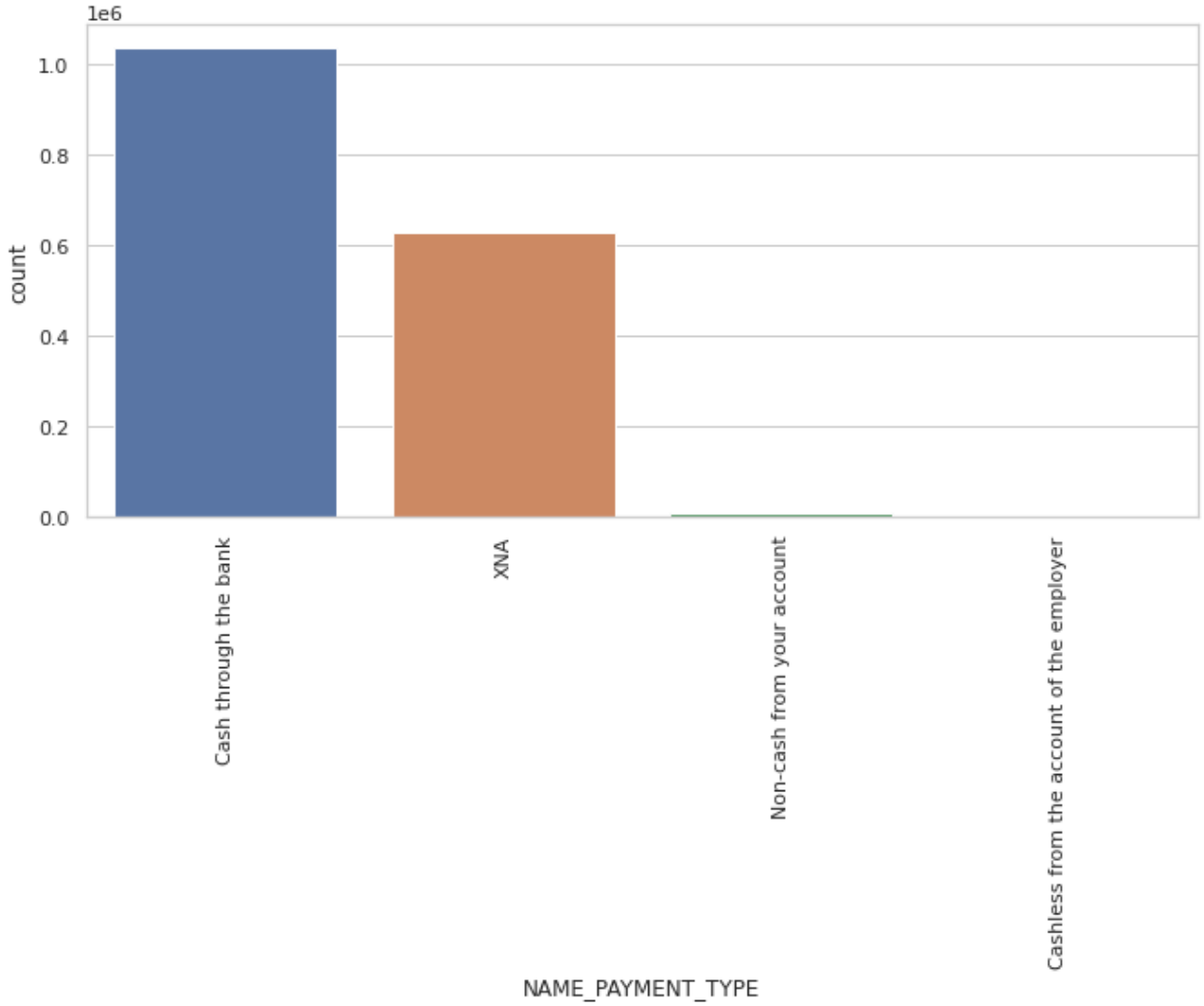

700000
600000
500000
400000
300000
200000
100000
0
count
Consumer loans
Cash loans
Revolving loans
XNA
NAME_CONTRACT_TYPE

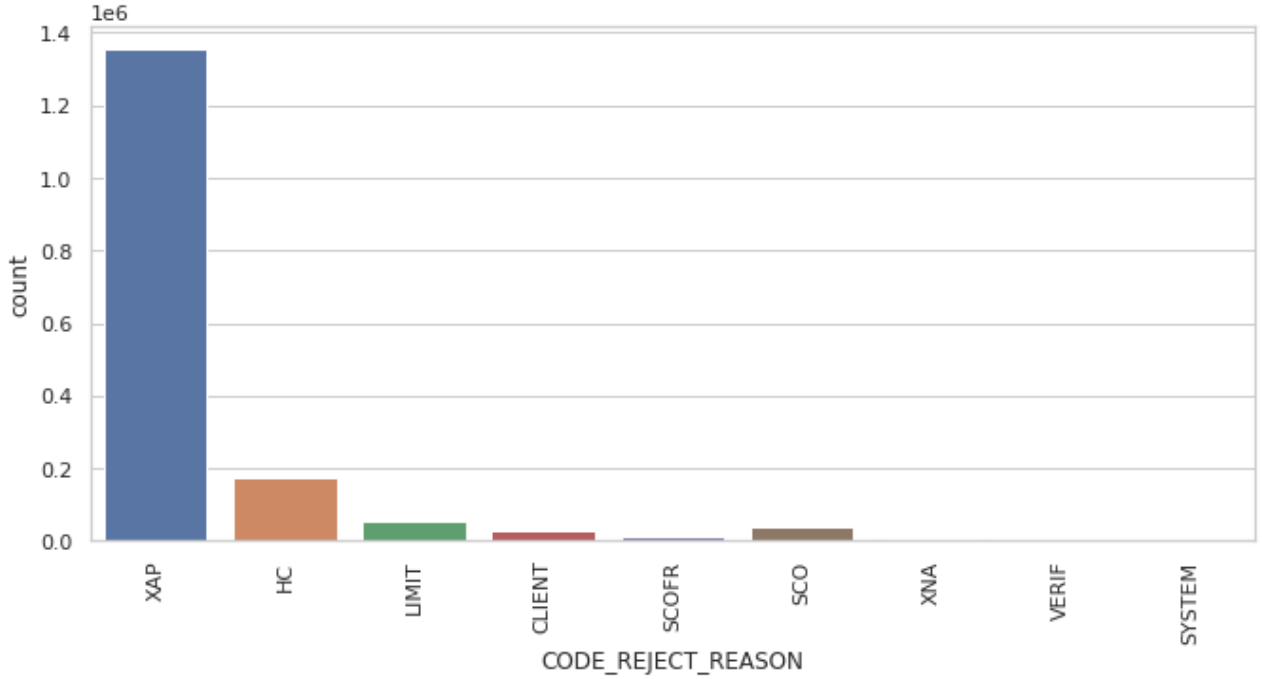

250000
200000
150000
100000
50000
0
count
SATURDAY
THURSDAY
TUESDAY
MONDAY
FRIDAY
SUNDAY
WEDNESDAY
WEEKDAY_APPR_PROCESS_START

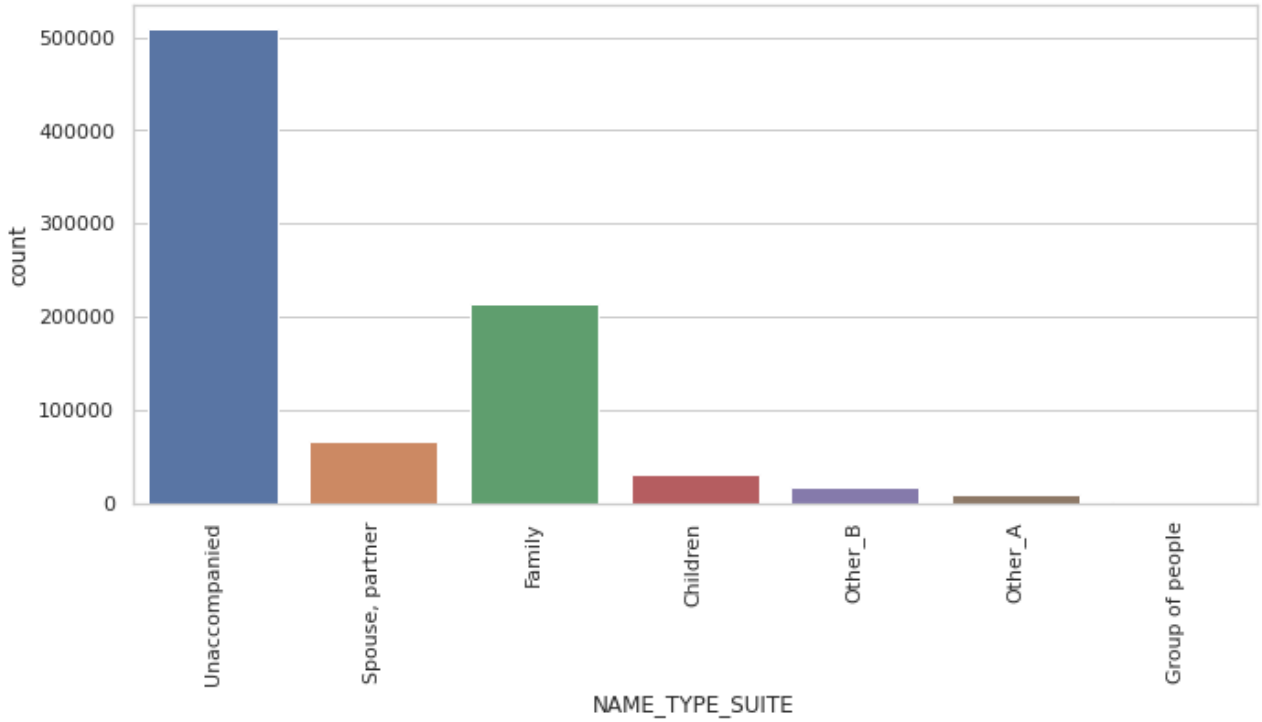

1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
count
Y
N
FLAG_LAST_APPL_PER_CONTRACT

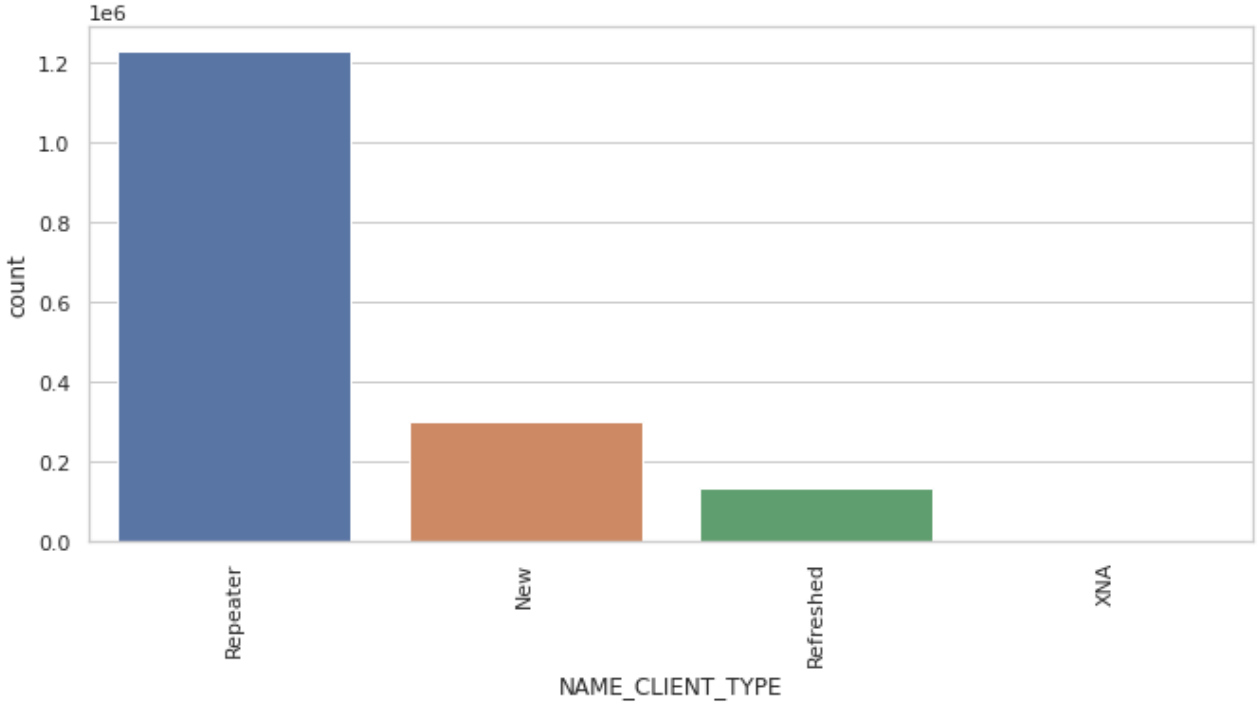

800000
600000
400000
200000
0
count
XAP
XNA
Repairs
Everyday expenses
Car repairs
Building a house or an annex
Other
Journey
Purchase of electronic equipment
Medicine
Payments on other loans
Urgent needs
Buying a used car
Buying a new car
Buying a holiday home / land
Education
Buying a home
Furniture
Buying a garage
Business development
Wedding / gift / holiday
Hobby
Gasification / water supply
Refusal to name the goal
Money for a third person
NAME_CASH_LOAN_PURPOSE

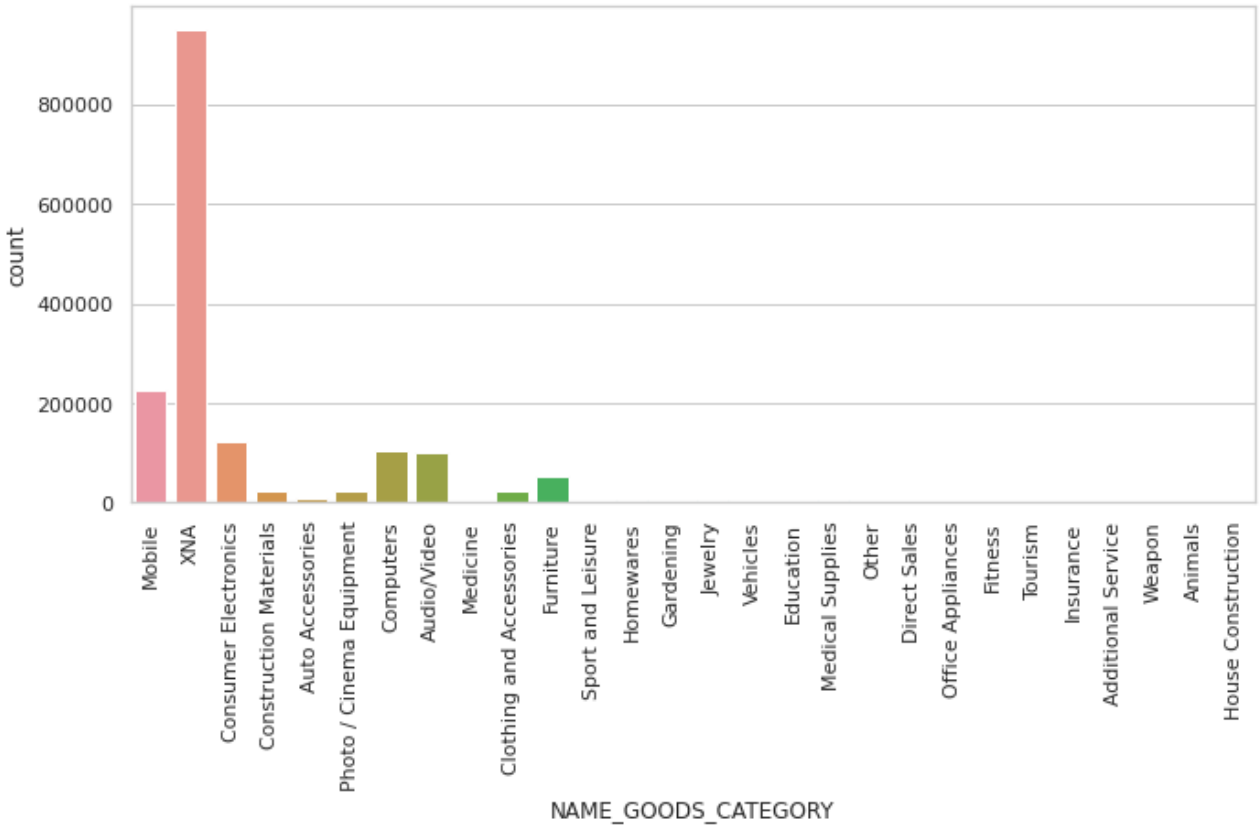

1e6
1.0
0.8
0.6
0.4
0.2
0.0
count
Approved
Refused
Canceled
Unused offer
NAME_CONTRACT_STATUS

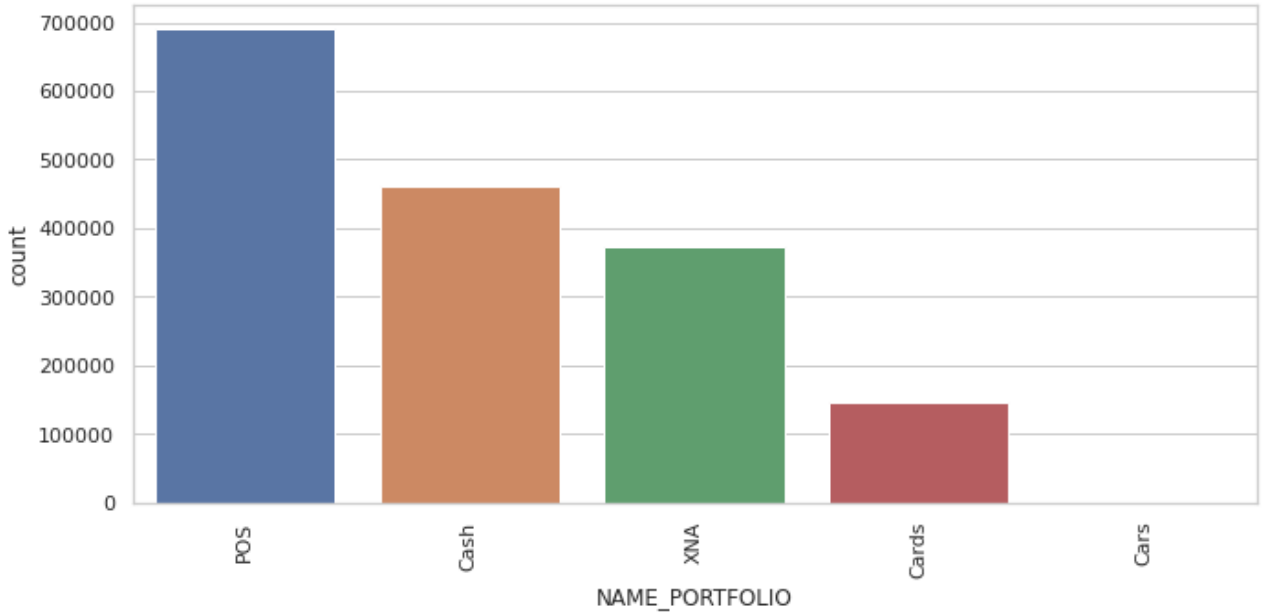

1e6
1.0
0.8
0.6
0.4
0.2
0.0
count
Cash through the bank
XNA
Non-cash from your account
Cashless from the account of the employer
NAME_PAYMENT_TYPE

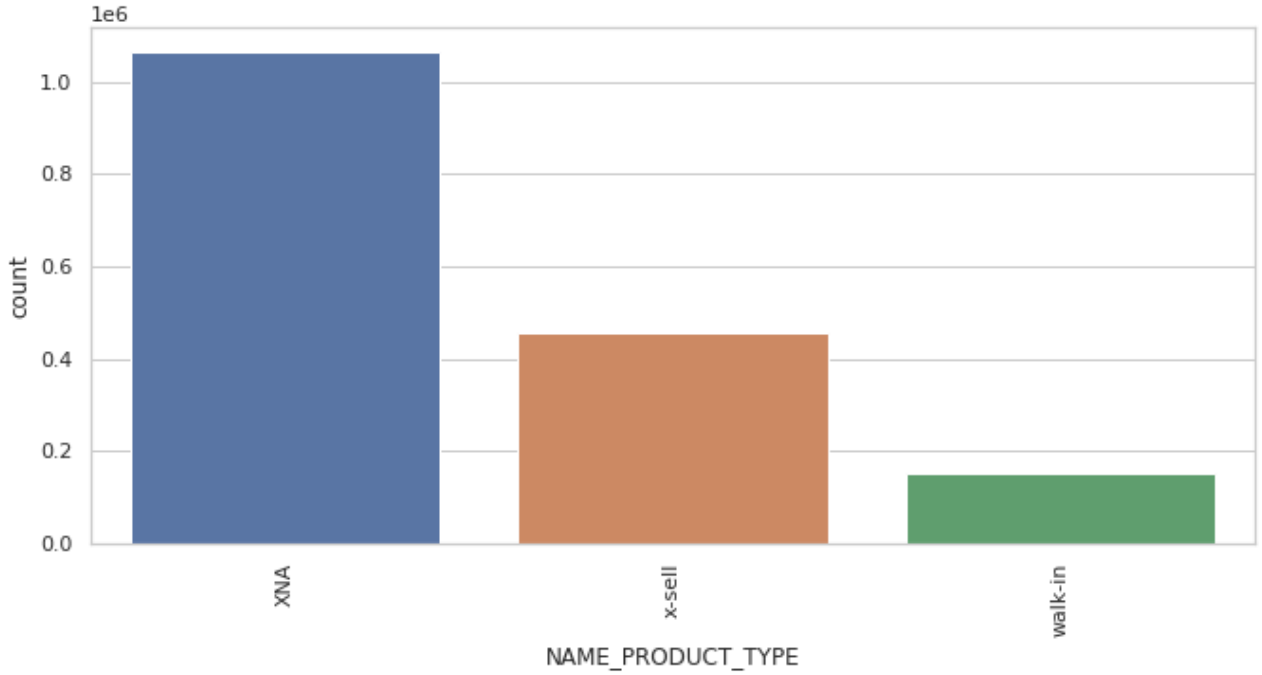

1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
count
XAP
HC
LIMIT
CLIENT
SCOFR
SCO
XNA
VERIF
SYSTEM
CODE_REJECT_REASON

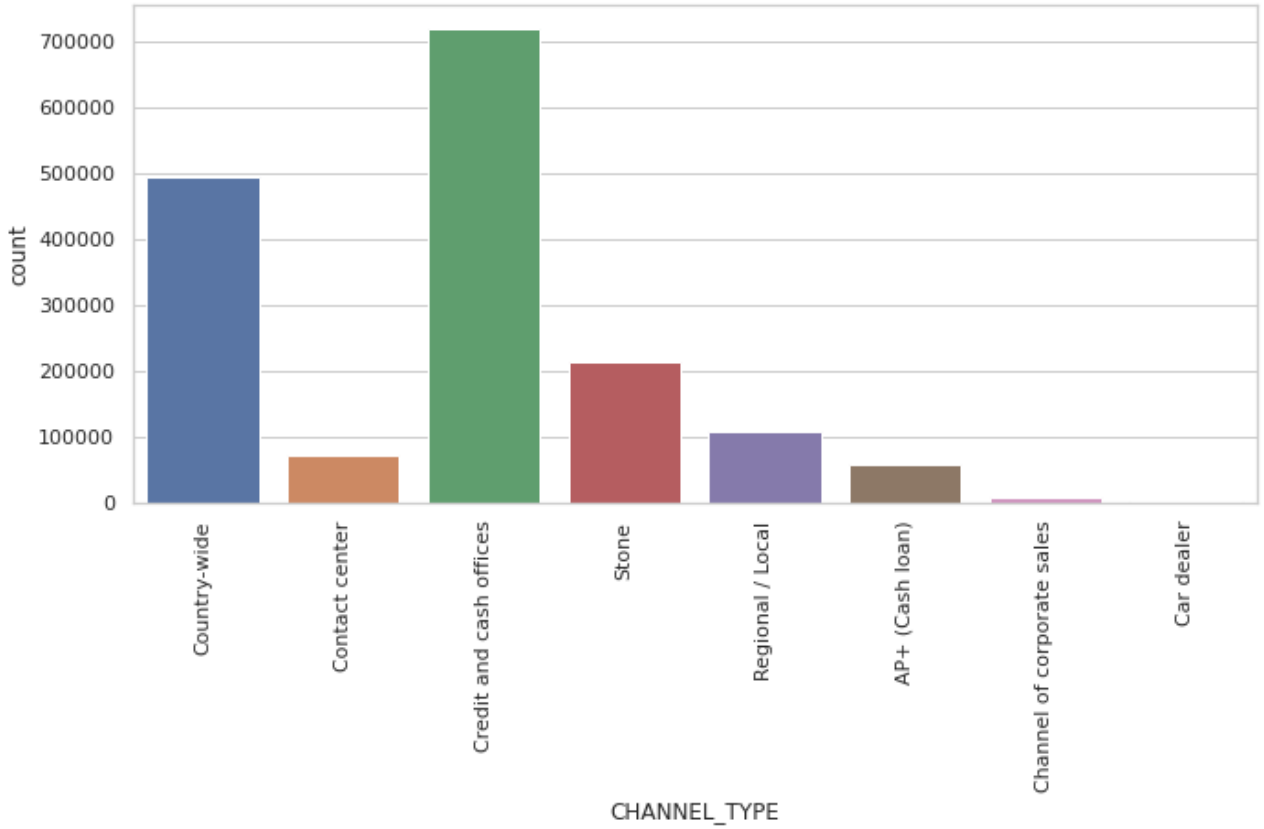

500000
400000
300000
200000
100000
0
count
Unaccompanied
Spouse, partner
Family
Children
Other_B
Other_A
Group of people
NAME_TYPE_SUITE

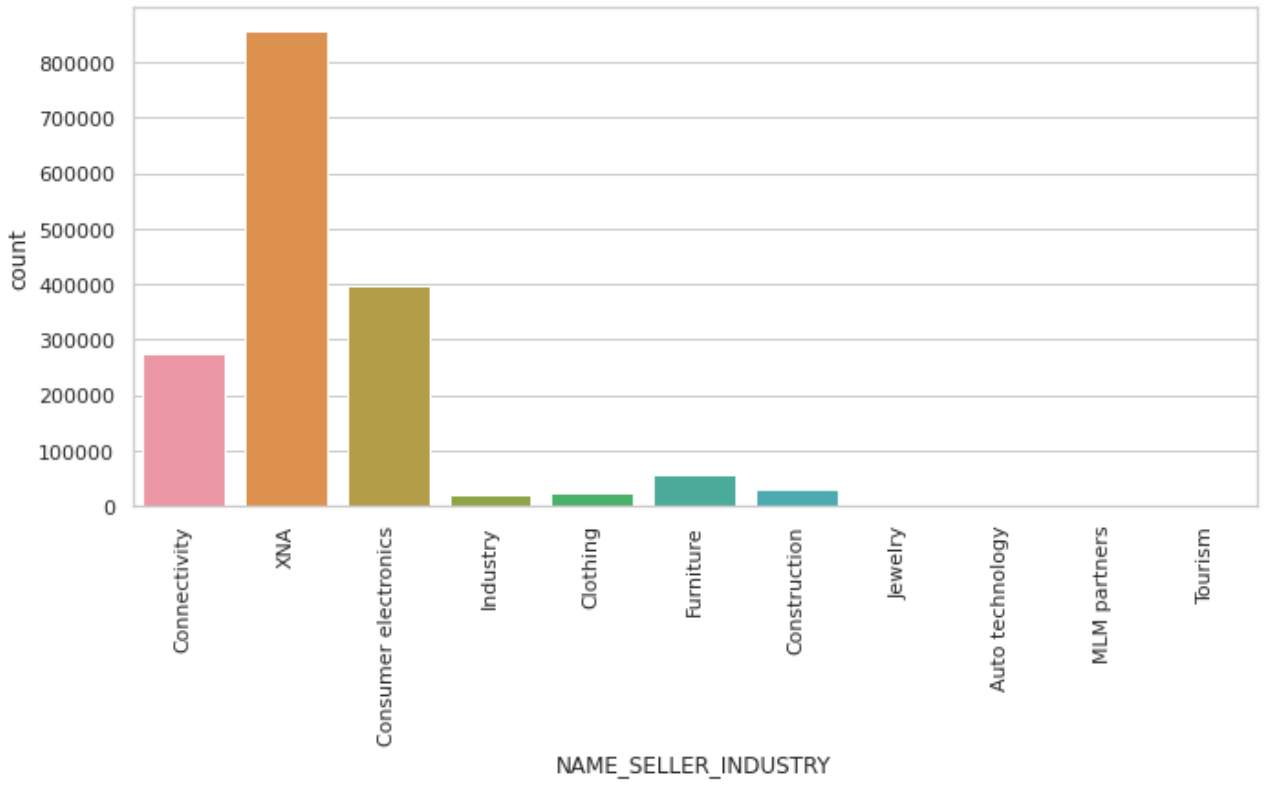

1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
count
Repeater
New
Refreshed
XNA
NAME_CLIENT_TYPE

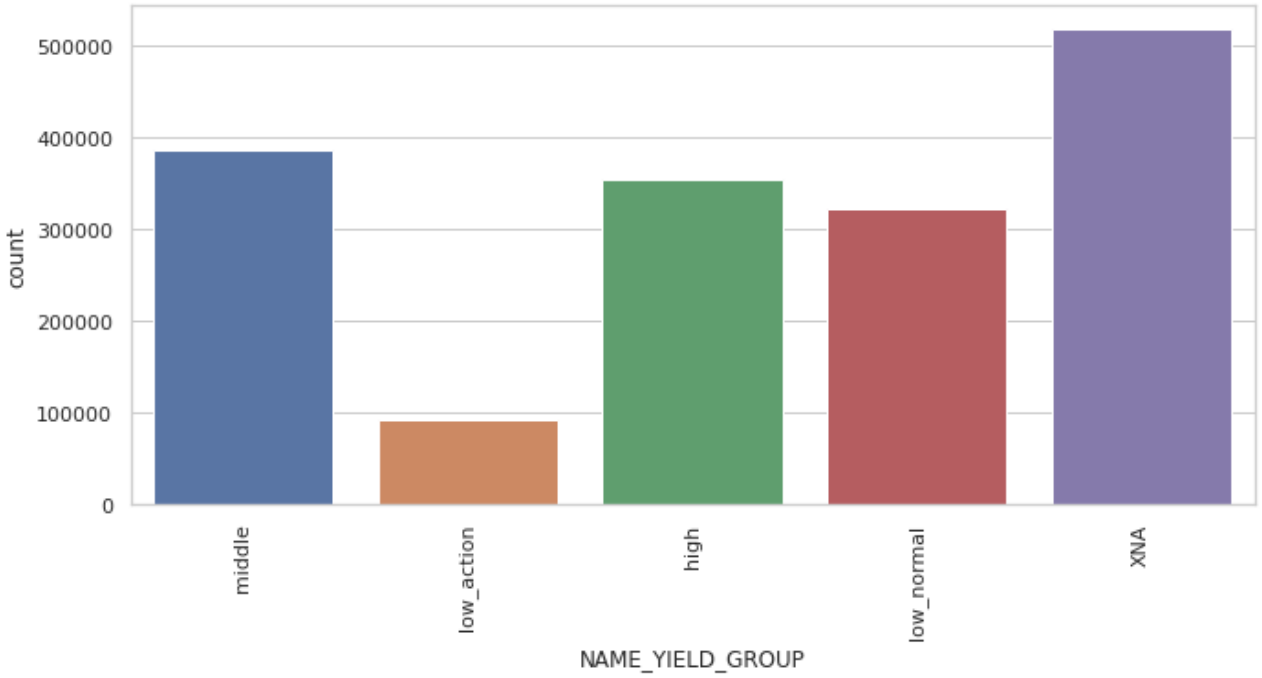

800000
600000
400000
200000
0
count
Mobile
XNA
Consumer Electronics
Construction Materials
Auto Accessories
Photo / Cinema Equipment
Computers
Audio/Video
Medicine
Clothing and Accessories
Furniture
Sport and Leisure
Homewares
Gardening
Jewelry
Vehicles
Education
Medical Supplies
Other
Direct Sales
Office Appliances
Fitness
Tourism
Insurance
Additional Service
Weapon
Animals
House Construction
NAME_GOODS_CATEGORY

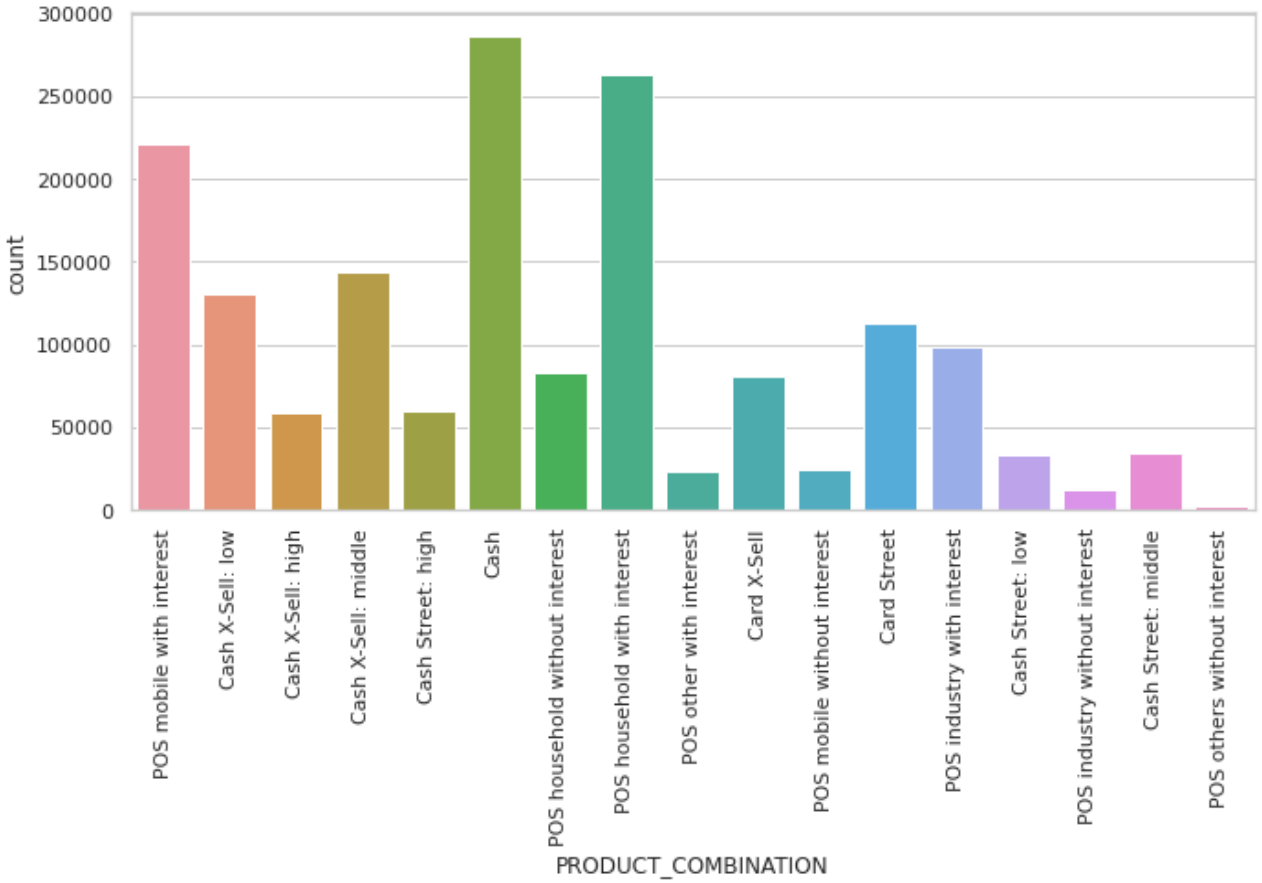

700000
600000
500000
400000
300000
200000
100000
0
count
POS
Cash
XNA
Cards
Cars
NAME_PORTFOLIO


1e6
1.0
0.8
0.6
0.4
0.2
0.0
count
XNA
x-sell
walk-in
NAME_PRODUCT_TYPE


count
700000
600000
500000
400000
300000
200000
100000
0
Country-wide
Contact center
Credit and cash offices
Stone
Regional / Local
AP+ (Cash loan)
Channel of corporate sales
Car dealer
CHANNEL_TYPE

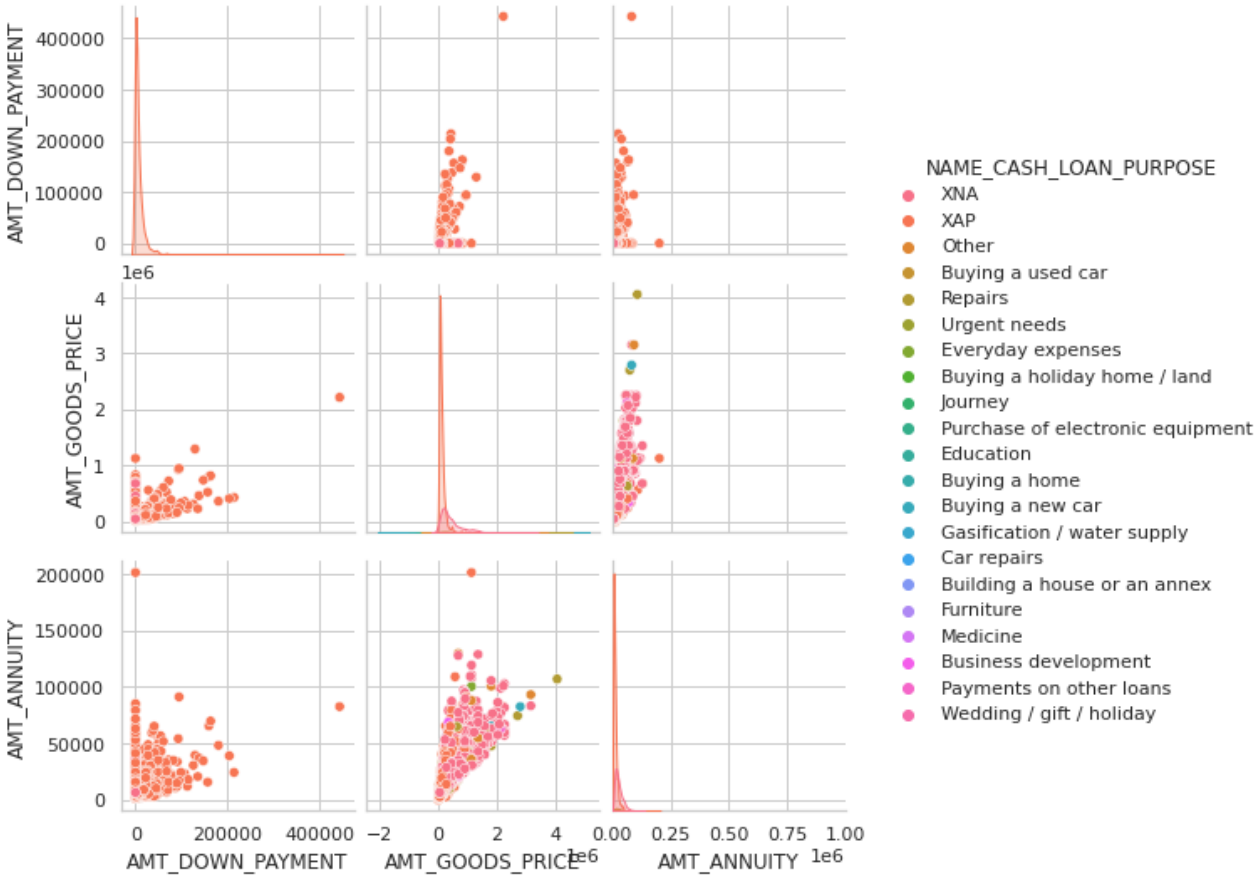

count
800000
700000
600000
500000
400000
300000
200000
100000
0
Connectivity
XNA
Consumer electronics
Industry
Clothing
Furniture
Construction
Jewelry
Auto technology
MLM partners
Tourism
NAME_SELLER_INDUSTRY





In [ ]:
```
#Visualizing the relationships between AMT_DOWN_PAYMENT, AMT_GOODS_PRICE, AMT_AN
plt.figure(figsize=(12,5))
ax = sns.pairplot(previous_application_10K, vars = ['AMT_DOWN_PAYMENT','AMT_GOOD
plt.xlim((0,1000000))
plt.show()
```

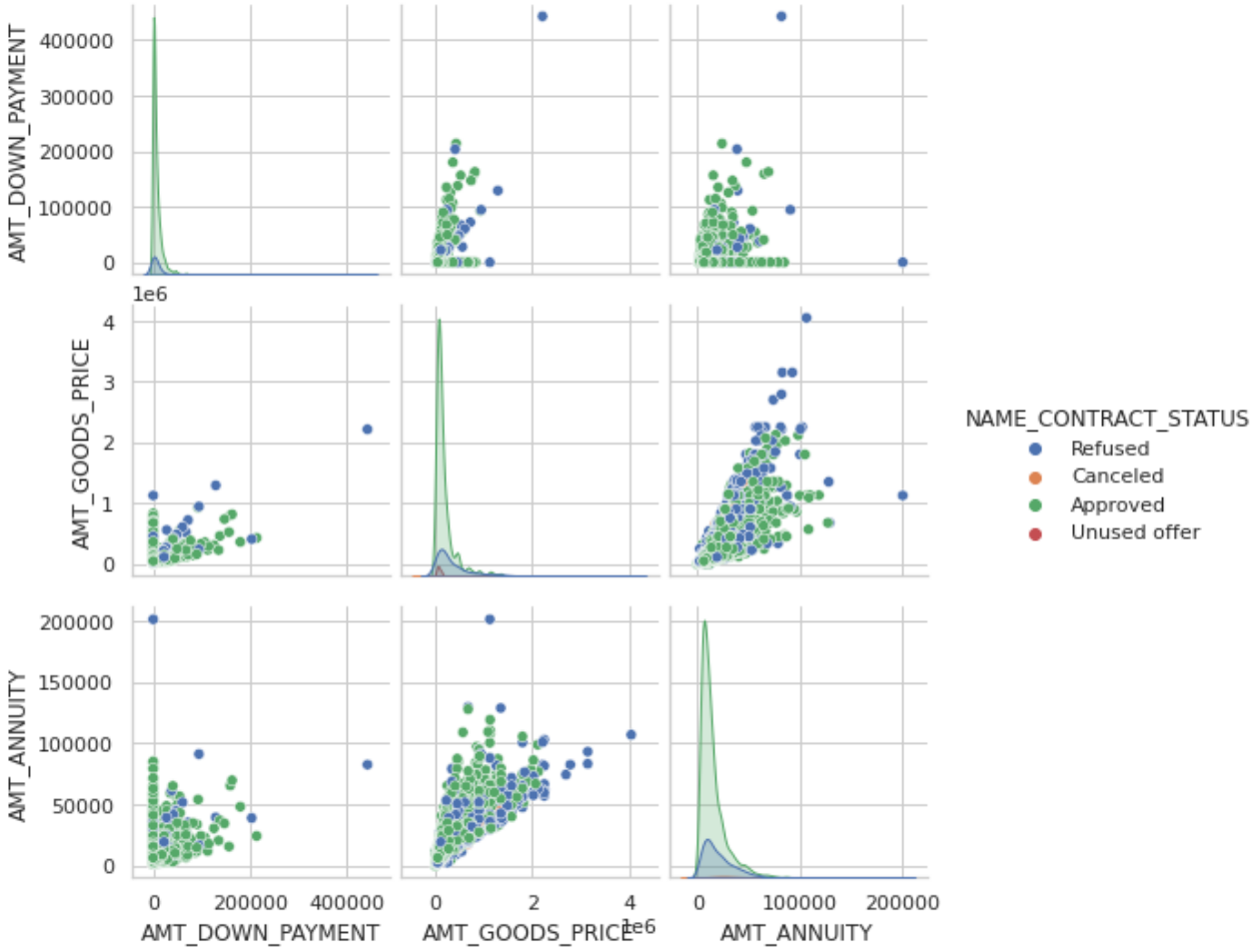
```
<Figure size 864x360 with 0 Axes>
```



- AMT_ANNUITY and AMT_DOWN_PAYMENT are highly corrleated with a postive linear slope.
- AMT_ANNUITY and AMT_GOODS_PRICE are highly corrleated with a postive linear slope.
- AMT_GOODS_PRICE and AMT_DOWN_PAYMENT are highly corrleated with a postive linear slope.
- Accounting customers NAME_CASH_LOAN_PURPOSE, XNA is much common than any other options.

In [ ]:
```
#Visualizing the relationships between AMT_DOWN_PAYMENT, AMT_GOODS_PRICE, AMT_AN
plt.figure(figsize=(12,5))
ax = sns.pairplot(previous_application_10K, vars = ['AMT_DOWN_PAYMENT','AMT_GOOD
plt.show()
```

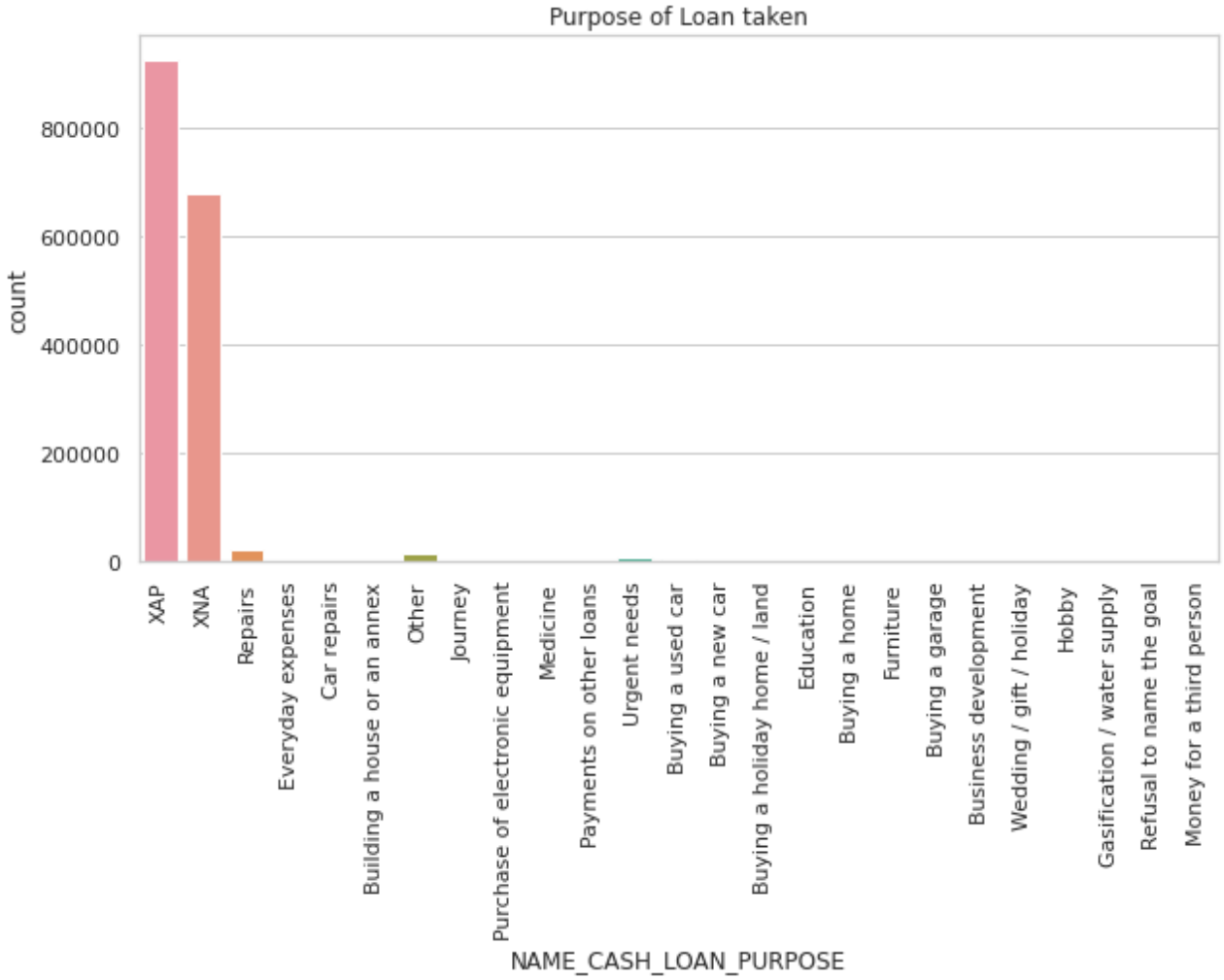
```
<Figure size 864x360 with 0 Axes>
```



- AMT_ANNUITY and AMT_DOWN_PAYMENT are highly corrleated with a postive linear slope.
- AMT_ANNUITY and AMT_GOODS_PRICE are highly corrleated with a postive linear slope.
- AMT_GOODS_PRICE and AMT_DOWN_PAYMENT are highly corrleated with a postive linear slope.
- Accounting customers NAME_CONTRACT_STATUS, approved is much common than any other options.

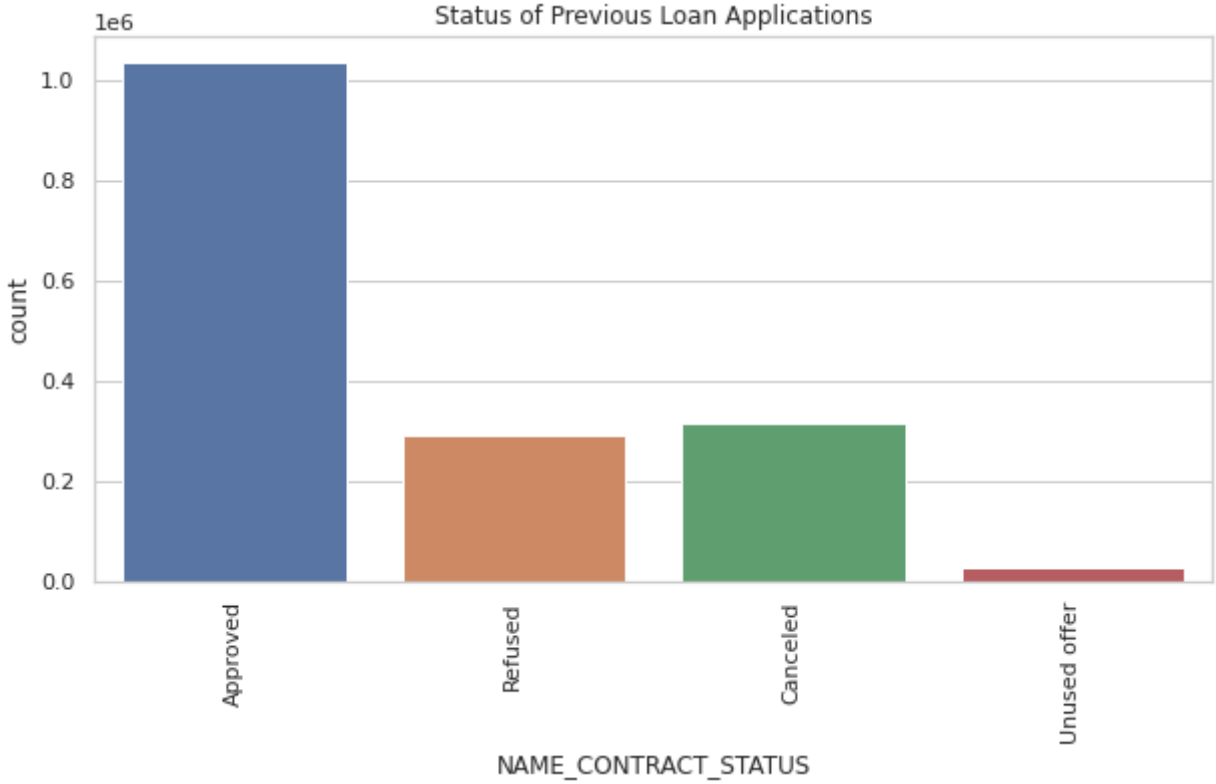
In [ ]:

```
plt.figure(figsize=(10,5))
sns.countplot(x='NAME_CASH_LOAN_PURPOSE', data=datasets["previous_application"])
plt.title('Purpose of Loan taken');
plt.xticks(rotation=90);
plt.show()
```



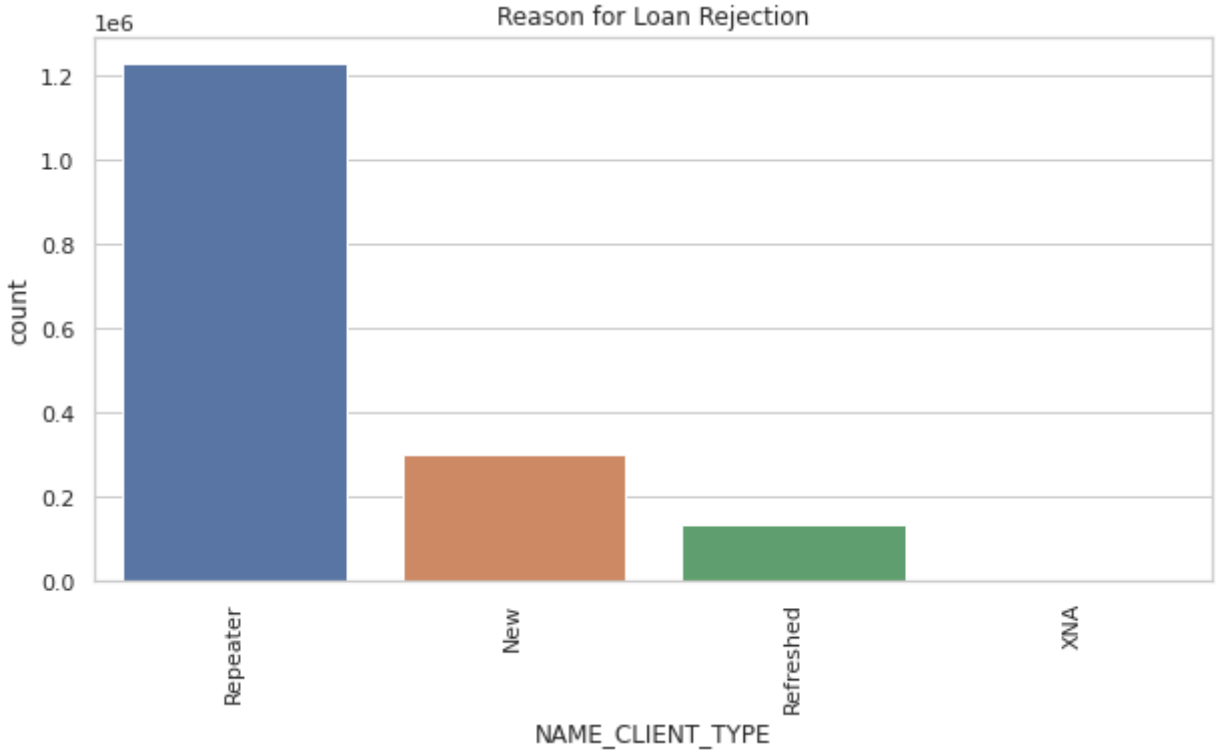
In [ ]:

```
plt.figure(figsize=(10,5))
sns.countplot(x='NAME_CONTRACT_STATUS', data=datasets["previous_application"]);
plt.title('Status of Previous Loan Applications');
plt.xticks(rotation=90);
plt.show()
```



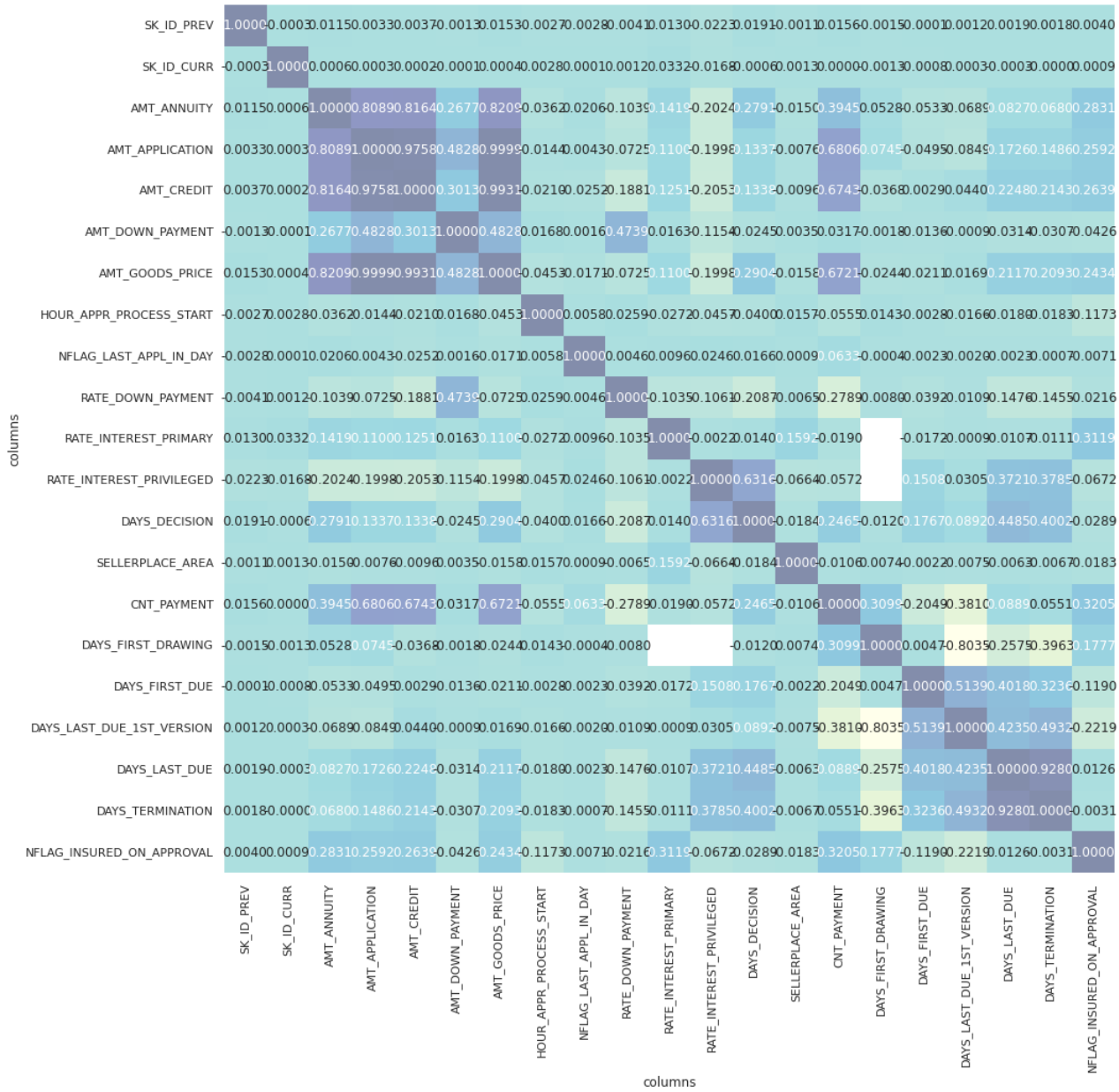
In [ ]:

```
plt.figure(figsize=(10,5))
sns.countplot(x='NAME_CLIENT_TYPE', data=datasets["previous_application"]);
plt.title('Reason for Loan Rejection');
plt.xticks(rotation=90);
plt.show()
```



Correlation

In [ ]:
```
corrs = datasets['previous_application'].corr()
sns.heatmap(datasets['previous_application'].corr(), annot = True, cmap="YlGnBu"
plt.show()

annot = True   #gives you the numbers instead of just the colors.
cmap = "YlGnBu"  #is a yellow/green/blue color scheme that I like to use
alpha = 0.5    #tones down the intensity of the colors
fmt = ".4f"    #formats the numbers to four decimal places
cbar = False   #turns off the scale bar on the right
```



Column Types

In [ ]:
```
datasets['previous_application'].dtypes.value_counts()
```

Out[ ]:
```
object     16
float32    10
float16     5
int32       3
int8        2
```

```
int16       1
dtype: int64
```

In [ ]:
```
datasets['previous_application'].select_dtypes('object').nunique()
```

Out[ ]:
```
columns
NAME_CONTRACT_TYPE              4
WEEKDAY_APPR_PROCESS_START      7
FLAG_LAST_APPL_PER_CONTRACT     2
NAME_CASH_LOAN_PURPOSE         25
NAME_CONTRACT_STATUS            4
NAME_PAYMENT_TYPE               4
CODE_REJECT_REASON              9
NAME_TYPE_SUITE                 7
NAME_CLIENT_TYPE                4
NAME_GOODS_CATEGORY            28
NAME_PORTFOLIO                  5
NAME_PRODUCT_TYPE               3
CHANNEL_TYPE                    8
NAME_SELLER_INDUSTRY           11
NAME_YIELD_GROUP                5
PRODUCT_COMBINATION            17
dtype: int64
```

EDA for POS_CASH_balance

Summary of POS_CASH_balance

In [ ]:
```
datasets["POS_CASH_balance"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10001358 entries, 0 to 10001357
Data columns (total 8 columns):
 #   Column                 Dtype
---  ------                 -----
 0   SK_ID_PREV             int32
 1   SK_ID_CURR             int32
 2   MONTHS_BALANCE         int8
 3   CNT_INSTALMENT         float16
 4   CNT_INSTALMENT_FUTURE  float16
 5   NAME_CONTRACT_STATUS   object
 6   SK_DPD                 int16
 7   SK_DPD_DEF             int16
dtypes: float16(2), int16(2), int32(2), int8(1), object(1)
memory usage: 238.5+ MB
```

In [ ]:
```
datasets["POS_CASH_balance"].shape
```

Out[ ]:
```
(10001358, 8)
```

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

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FU
count	1.000136e+07	1.000136e+07	1.000136e+07	9975287.0	997
mean	1.903217e+06	2.784039e+05	-3.501259e+01	NaN	

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FU
std	5.358465e+05	1.027637e+05	2.606657e+01	0.0	
min	1.000001e+06	1.000010e+05	-9.600000e+01	1.0	
25%	1.434405e+06	1.895500e+05	-5.400000e+01	10.0	
50%	1.896565e+06	2.786540e+05	-2.800000e+01	12.0	
75%	2.368963e+06	3.674290e+05	-1.300000e+01	24.0	
max	2.843499e+06	4.562550e+05	-1.000000e+00	92.0	

In [ ]:
```
datasets["POS_CASH_balance"].describe(include='all') #look at all categorical an
```

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_I
count	1.000136e+07	1.000136e+07	1.000136e+07	9975287.0	99
unique	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	
mean	1.903217e+06	2.784039e+05	-3.501259e+01	NaN	
std	5.358465e+05	1.027637e+05	2.606657e+01	0.0	
min	1.000001e+06	1.000010e+05	-9.600000e+01	1.0	
25%	1.434405e+06	1.895500e+05	-5.400000e+01	10.0	
50%	1.896565e+06	2.786540e+05	-2.800000e+01	12.0	
75%	2.368963e+06	3.674290e+05	-1.300000e+01	24.0	
max	2.843499e+06	4.562550e+05	-1.000000e+00	92.0	

Missing data for POS_CASH_balance

In [ ]:
```
percent_POS_CASH_balance = (datasets["POS_CASH_balance"].isnull().sum()/datasets
sum_missing_POS_CASH_balance = datasets["POS_CASH_balance"].isna().sum().sort_va
missing_POS_CASH_balance  = pd.concat([percent_POS_CASH_balance, sum_missing_POS
missing_POS_CASH_balance.head(10)
```

Out[ ]:

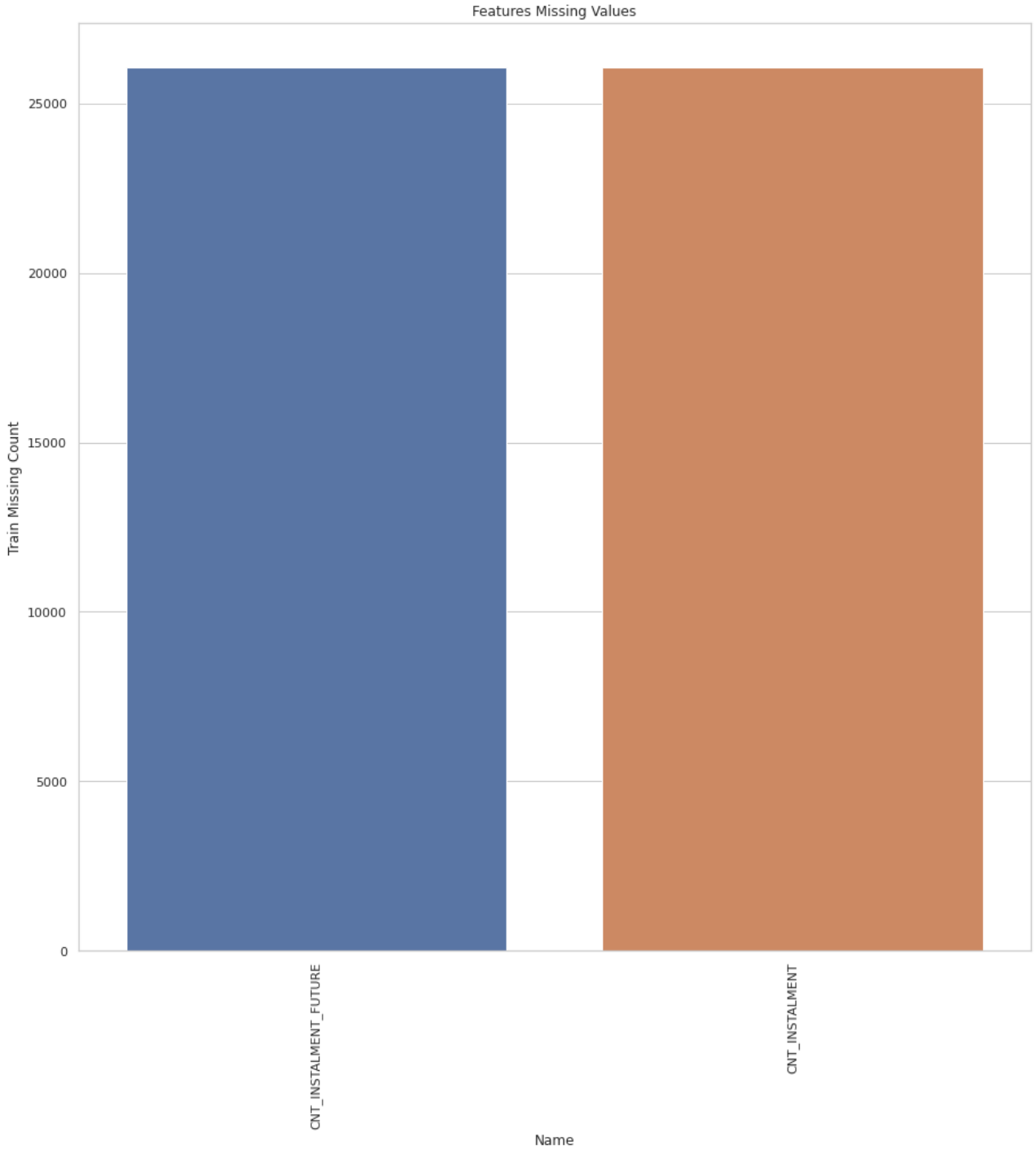
	Percent	Train Missing Count
CNT_INSTALMENT_FUTURE	0.26	26087
CNT_INSTALMENT	0.26	26071
SK_ID_PREV	0.00	0
SK_ID_CURR	0.00	0
MONTHS_BALANCE	0.00	0

	Percent	Train Missing Count
NAME_CONTRACT_STATUS	0.00	0
SK_DPD	0.00	0
SK_DPD_DEF	0.00	0

In [ ]:
```
missing_POS_CASH_balance = pd.DataFrame(
missing_POS_CASH_balance[missing_POS_CASH_balance['Percent']>0.0])
```

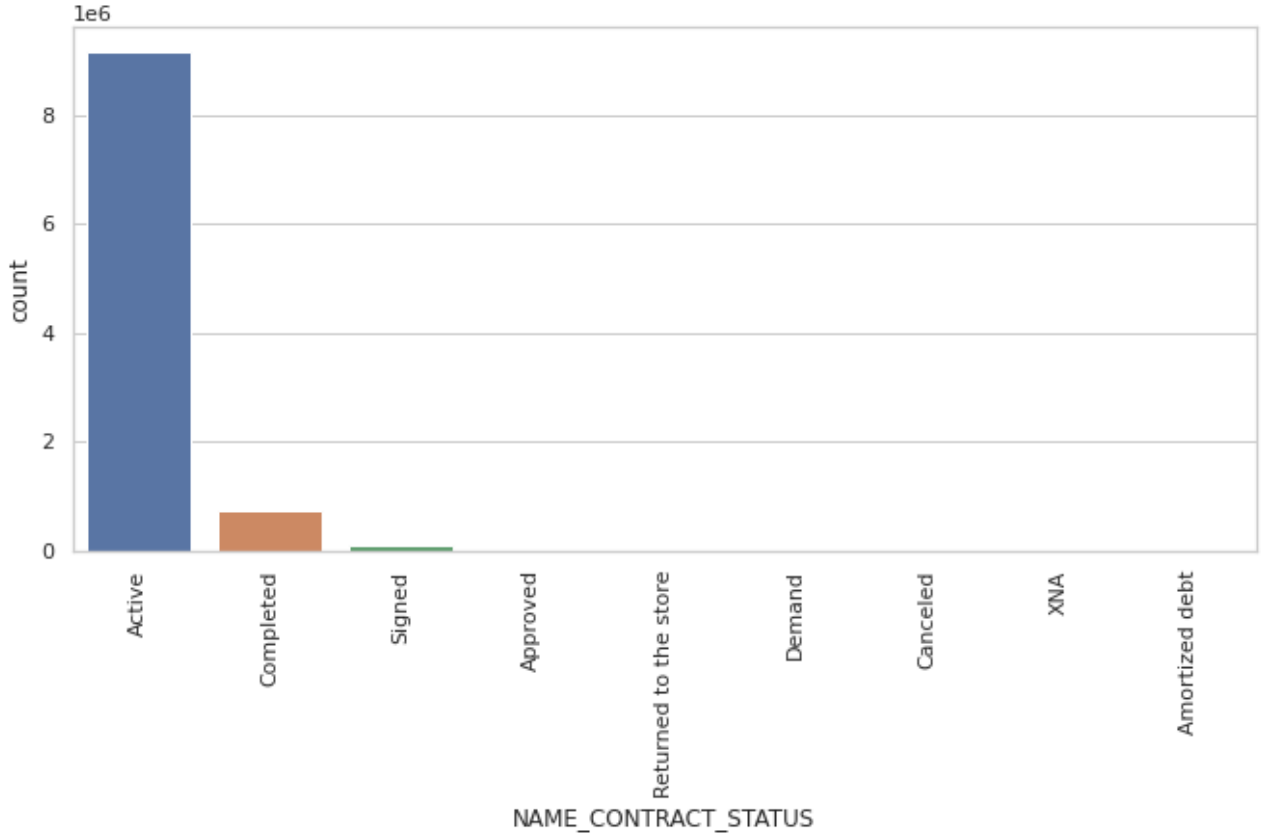
In [ ]:
```
missing_POS_CASH_balance.columns.name = 'Name'
missing_POS_CASH_balance['Name']=missing_POS_CASH_balance.index
```

In [ ]:
```
sns.set(style="whitegrid", color_codes=True,rc={'figure.figsize':(15,15)})
sns.barplot(x = 'Name', y = 'Train Missing Count', data=missing_POS_CASH_balance
plt.xticks(rotation = 90)
plt.show()
```



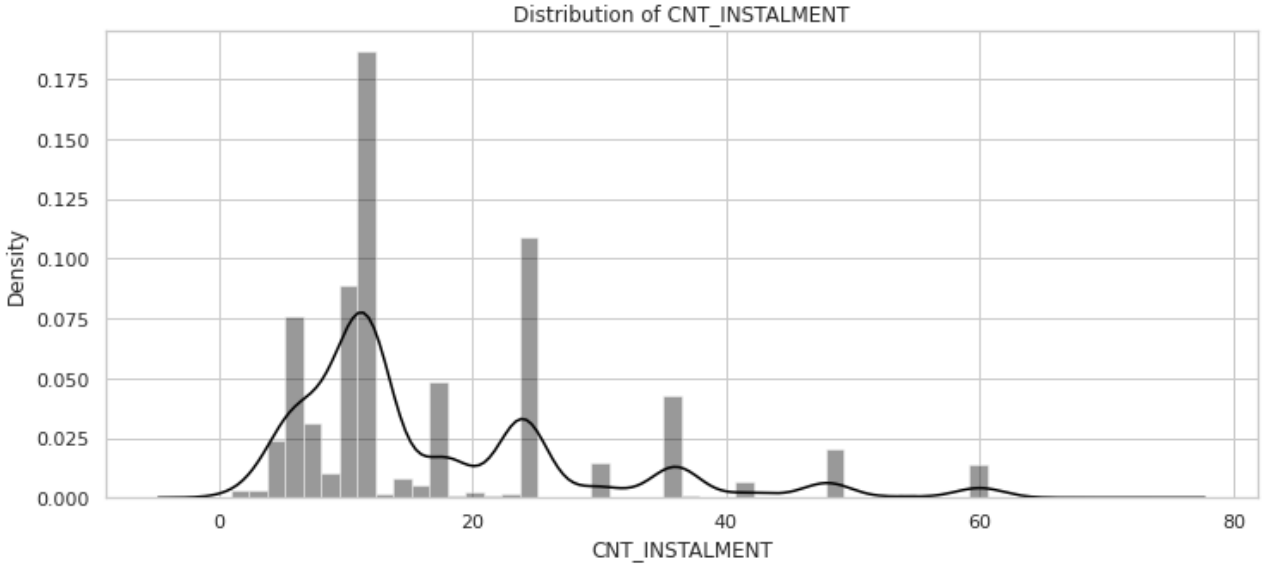
In [ ]:
```
POS_CASH_balance_10K = datasets["POS_CASH_balance"].sample(n=10000, random_state
```

In [ ]:
```
graph_objects(datasets["POS_CASH_balance"])
```



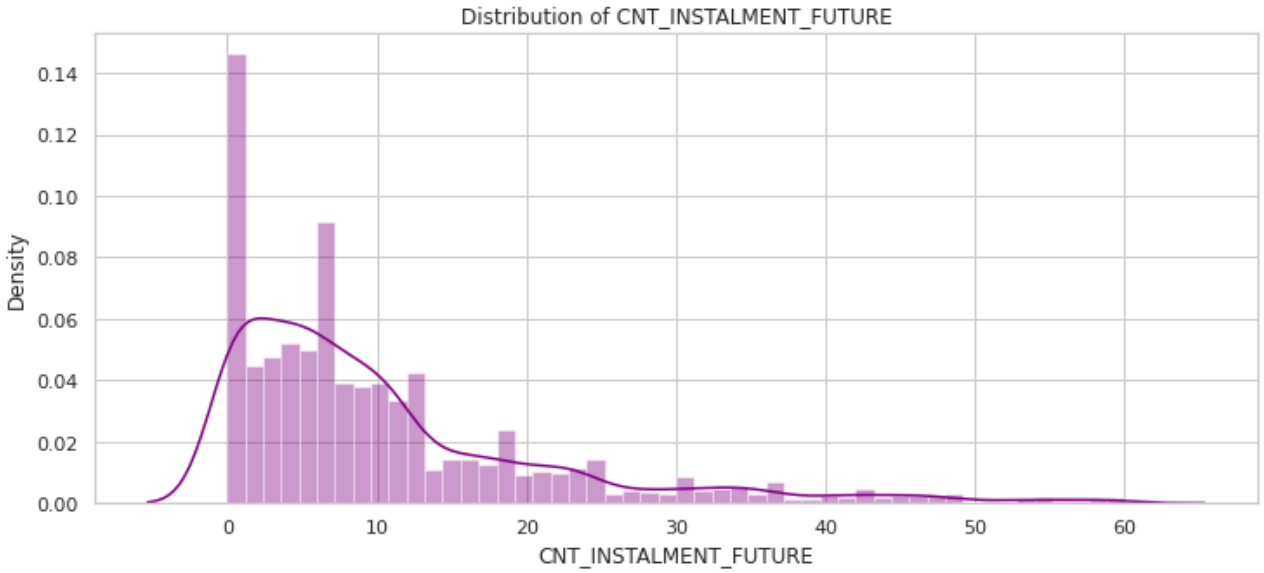
Most of Contracts in POS_CASH_BALANCE data are in active status

In [ ]:
```
#Show  Distribution of CNT_INSTALNENT in POS_CASH data
plt.figure(figsize=(12,5))
plt.title("Distribution of CNT_INSTALMENT")
ax = sns.distplot(POS_CASH_balance_10K.CNT_INSTALMENT.dropna(),color='black')
plt.show()
```



In [ ]:
```
#Show Distribution of CNT_INSTALNENT_FURURE in POS_CASH data
plt.figure(figsize=(12,5))
plt.title("Distribution of CNT_INSTALMENT_FUTURE")
```

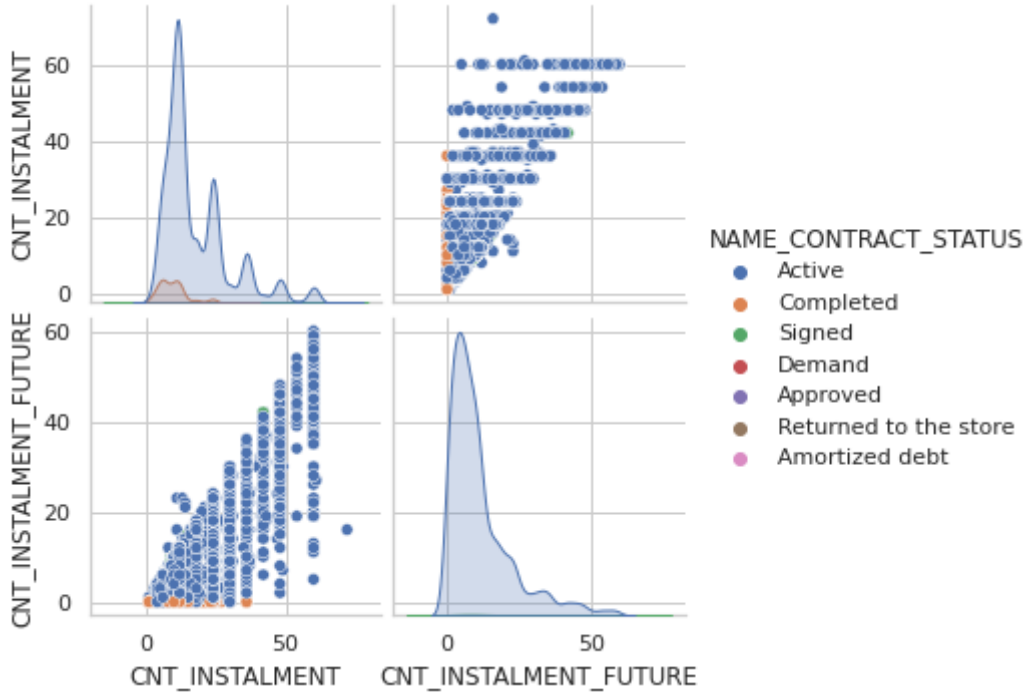
```
ax = sns.distplot(POS_CASH_balance_10K.CNT_INSTALMENT_FUTURE.dropna(),color='pur
plt.show()
```



In [ ]:

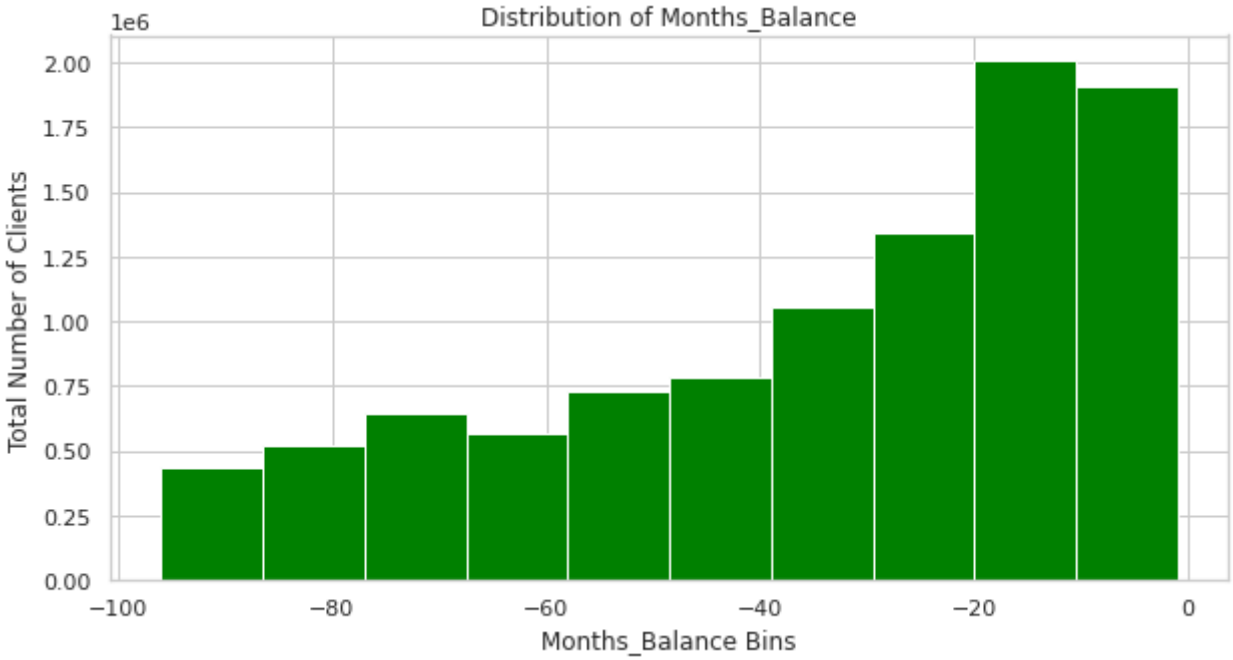
```
#Visualizing the relationships between CNT_INSTALMENT and CNT_INSTALMENT_FUTRUE
plt.figure(figsize=(12,5))
ax= sns.pairplot(POS_CASH_balance_10K, vars = ['CNT_INSTALMENT', 'CNT_INSTALMENT
plt.show()
```

```
<Figure size 864x360 with 0 Axes>
```

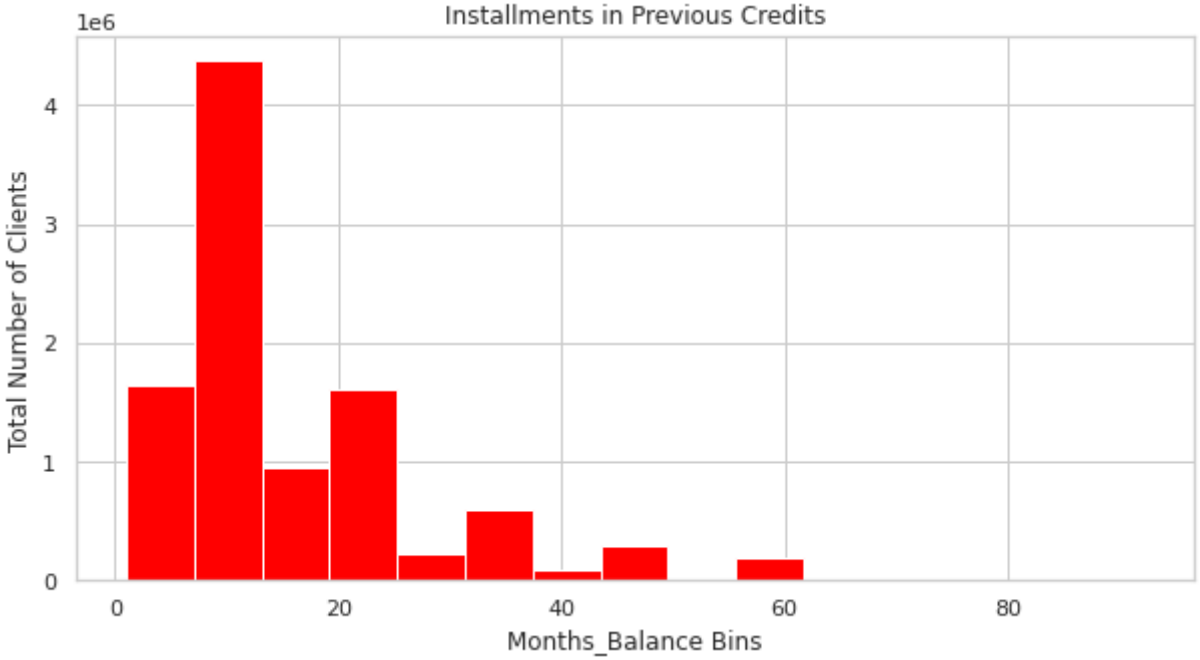


In [ ]:

```
plt.figure(figsize=(10,5))
plt.hist(datasets['POS_CASH_balance'][['MONTHS_BALANCE']].values, bins=10,color=
plt.title('Distribution of Months_Balance')
plt.xlabel('Months_Balance Bins')
plt.ylabel('Total Number of Clients')
plt.show()
```



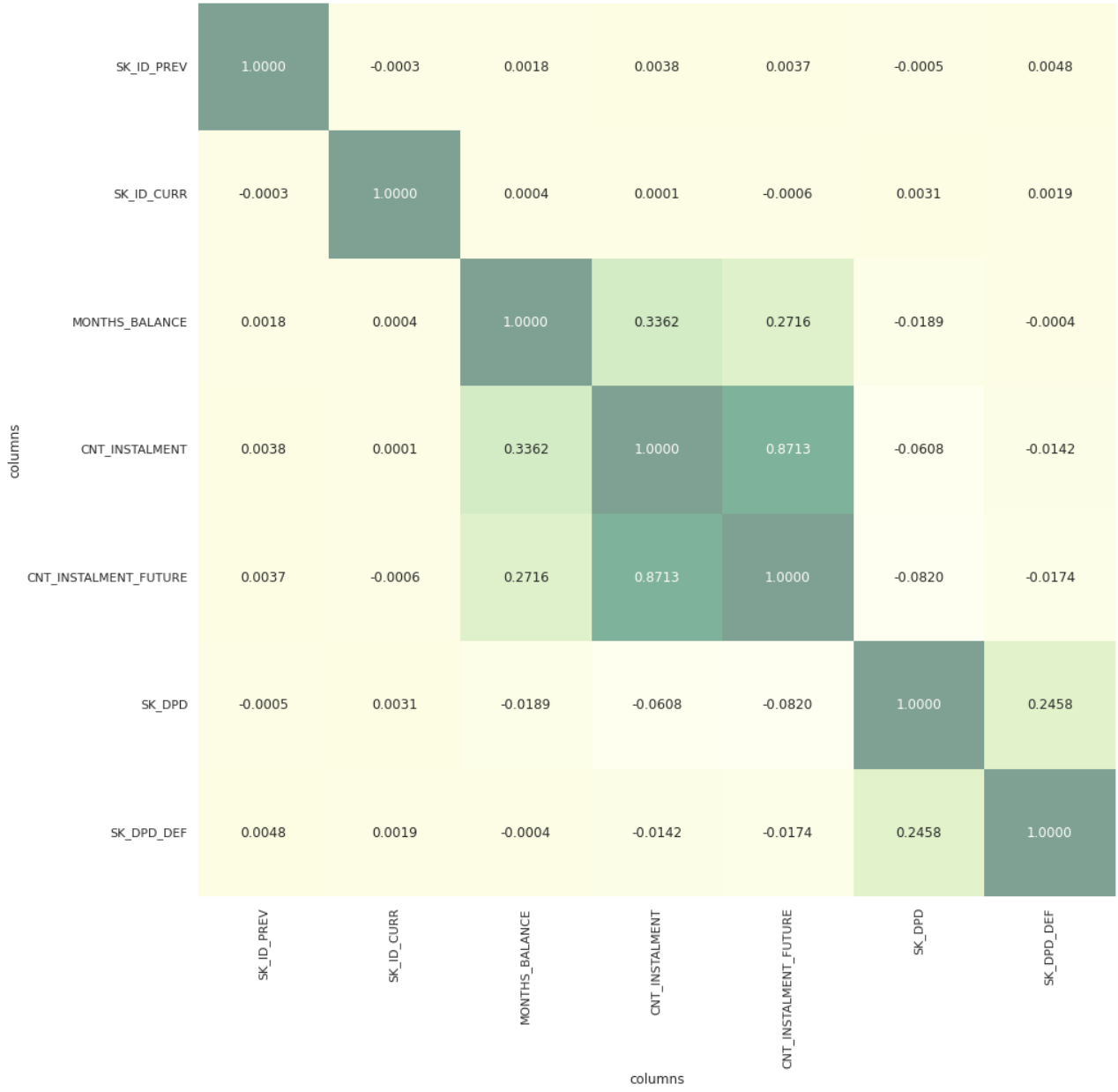
In [ ]:
```
plt.figure(figsize=(10,5))
plt.hist(datasets['POS_CASH_balance'][['CNT_INSTALMENT']].values, bins=15,color=
plt.title('Installments in Previous Credits')
plt.xlabel('Months_Balance Bins')
plt.ylabel('Total Number of Clients')
plt.show()
```



Correlation

In [ ]:
```
corrs = datasets['POS_CASH_balance'].corr()
sns.heatmap(datasets['POS_CASH_balance'].corr(), annot = True, cmap="YlGn", alph
plt.show()

annot = True   #gives you the numbers instead of just the colors.
```

```
cmap = "YlGn"  #is a yellow/green color scheme that I like to use
alpha = 0.5    #tones down the intensity of the colors
fmt = ".4f"    #formats the numbers to four decimal places
cbar = False   #turns off the scale bar on the right
```



Column Types

In [ ]:
```
datasets['POS_CASH_balance'].dtypes.value_counts()
```

Out[ ]:
```
int32      2
float16    2
int16      2
int8       1
object     1
dtype: int64
```

In [ ]:
```
datasets['POS_CASH_balance'].select_dtypes('object').nunique()
```

Out[ ]:
```
columns
```

```
NAME_CONTRACT_STATUS    9
dtype: int64
```

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

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

In [ ]:
```
def get_unique_values(df):
    unique_column_values = list()
    for col in df.columns:
        if len(df[col].unique())==1:
            unique_column_values.append(col)
    return unique_column_values
```

In [ ]:
```
for x in datasets:
    features_list = get_unique_values(datasets[x])
    print(f'{x} has {len(features_list)} fetaures which has only one unique valu
```

```
application_train has 0 fetaures which has only one unique value
application_test has 11 fetaures which has only one unique value
bureau has 0 fetaures which has only one unique value
bureau_balance has 0 fetaures which has only one unique value
credit_card_balance has 0 fetaures which has only one unique value
installments_payments has 0 fetaures which has only one unique value
previous_application has 0 fetaures which has only one unique value
POS_CASH_balance has 0 fetaures which has only one unique value
```

Dataset questions

Unique record for each SK_ID_CURR

In [130…
```
len(datasets["application_train"]["SK_ID_CURR"].unique()) == datasets["applicati
```

Out[130…
```
True
```

In [131…
```
np.intersect1d(datasets["application_train"]["SK_ID_CURR"], datasets["applicatio
```

Out[131…
```
array([], dtype=int32)
```

In [132…
```
datasets["application_test"].shape
```

Out[132…
```
(48744, 121)
```

In [133…
```
datasets["application_train"].shape
```

Out[133…
```
(307511, 122)
```

previous applications for the submission file

The persons in the kaggle submission file have had previous applications in the `previous_application.csv` . 47,800 out 48,744 people have had previous appications.

In [134…
```
appsDF = datasets["previous_application"]
appsDF.shape
```

Out[134… (1261346, 37)

In [135…
```
len(np.intersect1d(datasets["previous_application"]["SK_ID_CURR"], datasets["app
```

Out[135… 45383

In [136…
```
print(f"There are  {appsDF.shape[0]:,} previous applications")
```

There are 1,261,346 previous applications

In [137…
```
# How many entries are there for each month?
prevAppCounts = appsDF['SK_ID_CURR'].value_counts(dropna=False)
```

In [138…
```
len(prevAppCounts[prevAppCounts >40])  #more that 40 previous applications
```

Out[138… 30

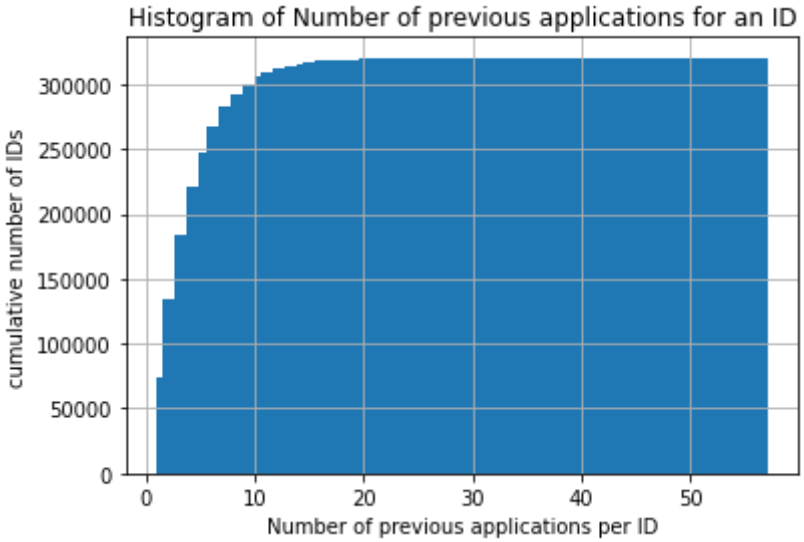
Histogram of Number of previous applications for an ID

In [139…
```
sum(appsDF['SK_ID_CURR'].value_counts()==1)
```

Out[139… 73708

In [140…
```
plt.hist(appsDF['SK_ID_CURR'].value_counts(), cumulative =True, bins = 100);
plt.grid()
plt.ylabel('cumulative number of IDs')
plt.xlabel('Number of previous applications per ID')
plt.title('Histogram of Number of previous applications for an ID')
```

Out[140… Text(0.5, 1.0, 'Histogram of Number of previous applications for an ID')



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

* Low = <5 claims (22%)
* Medium = 10 to 39 claims (58%)
* High = 40 or more claims (20%)

In [141…

```
apps_all = appsDF['SK_ID_CURR'].nunique()
apps_5plus = appsDF['SK_ID_CURR'].value_counts()>=5
apps_40plus = appsDF['SK_ID_CURR'].value_counts()>=40
print('Percentage with 10 or more previous apps:', np.round(100.*(sum(apps_5plus
print('Percentage with 40 or more previous apps:', np.round(100.*(sum(apps_40plu
```

```
Percentage with 10 or more previous apps: 31.28082
Percentage with 40 or more previous apps: 0.01061
```

Joining secondary tables with the primary table

In the case of the HCDR competition (and many other machine learning problems that involve multiple tables in 3NF or not) we need to join these datasets (denormalize) when using a machine learning pipeline. Joining the secondary tables with the primary table will lead to lots of new features about each loan application; these features will tend to be aggregate type features or meta data about the loan or its application. How can we do this when using Machine Learning Pipelines?

Joining `previous_application` with `application_x`

We refer to the `application_train` data (and also `application_test` data also) as the **primary table** and the other files as the **secondary tables** (e.g., `previous_application` dataset). All tables can be joined using the primary key `SK_ID_PREV`.

Let's assume we wish to generate a feature based on previous application attempts. In this case, possible features here could be:

- A simple feature could be the number of previous applications.
- Other summary features of original features such as `AMT_APPLICATION`, `AMT_CREDIT` could be based on average, min, max, median, etc.

To build such features, we need to join the `application_train` data (and also `application_test` data also) with the 'previous_application' dataset (and the other available datasets).

When joining this data in the context of pipelines, different strategies come to mind with various tradeoffs:

1. Preprocess each of the non-application data sets, thereby generating many new (derived) features, and then joining (aka merge) the results with the `application_train` data (the labeled dataset) and with the `application_test` data (the unlabeled submission dataset) prior to processing the data (in a train, valid, test partition) via your machine learning pipeline. [This approach is recommended for this HCDR competition. WHY?]

- Do the joins as part of the transformation steps. [Not recommended here. WHY?]. How can this be done? Will it work?
  - This would be necessary if we had dataset wide features such as IDF (inverse document frequency) which depend on the entire subset of data as opposed to a single loan application (e.g., a feature about the relative amount applied for such as the percentile of the loan amount being applied for).

I want you to think about this section and build on this.

Roadmap for secondary table processing

1. Transform all the secondary tables to features that can be joined into the main table the application table (labeled and unlabeled)
   - 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_payments',
   - 'previous_application', 'POS_CASH_balance'

- Merge the transformed secondary tables with the primary tables (i.e., the `application_train` data (the labeled dataset) and with the `application_test` data (the unlabeled submission dataset)), thereby leading to X_train, y_train, X_valid, etc.
- Proceed with the learning pipeline using X_train, y_train, X_valid, etc.
- Generate a submission file using the learnt model

Multiple condition expressions in Pandas

So far, both our boolean selections have involved a single condition. You can, of course, have as many conditions as you would like. To do so, you will need to combine your boolean expressions using the three logical operators and, or and not.

Use &, | , ~ Although Python uses the syntax and, or, and not, these will not work when testing multiple conditions with pandas. The details of why are explained here.

You must use the following operators with pandas:

- & for and
- | for or
- ~ for not

In [142…
```
appsDF[0:50][(appsDF["SK_ID_CURR"]==175704)]
```

Out[142…

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_
6	2315218	175704	Cash loans	NaN	0.0	

1 rows × 37 columns

In [143…
```
appsDF[0:50][(appsDF["SK_ID_CURR"]==175704) & ~(appsDF["AMT_CREDIT"]==1.0)]
```

Out[143…

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_
6	2315218	175704	Cash loans	NaN	0.0	

1 rows × 37 columns

Missing values in prevApps

In [144…
```
appsDF.isna().sum()
```

Out[144…
```
SK_ID_PREV                        0
SK_ID_CURR                        0
NAME_CONTRACT_TYPE                0
AMT_ANNUITY                  279992
AMT_APPLICATION                   0
AMT_CREDIT                        1
AMT_DOWN_PAYMENT             672399
AMT_GOODS_PRICE              289572
WEEKDAY_APPR_PROCESS_START        1
HOUR_APPR_PROCESS_START           1
FLAG_LAST_APPL_PER_CONTRACT       1
NFLAG_LAST_APPL_IN_DAY            1
RATE_DOWN_PAYMENT            672399
RATE_INTEREST_PRIMARY       1256858
RATE_INTEREST_PRIVILEGED    1256858
NAME_CASH_LOAN_PURPOSE            1
NAME_CONTRACT_STATUS              1
DAYS_DECISION                     1
NAME_PAYMENT_TYPE                 1
CODE_REJECT_REASON                1
```

```
NAME_TYPE_SUITE                619343
NAME_CLIENT_TYPE                    1
NAME_GOODS_CATEGORY                 1
NAME_PORTFOLIO                      1
NAME_PRODUCT_TYPE                   1
CHANNEL_TYPE                        1
SELLERPLACE_AREA                    1
NAME_SELLER_INDUSTRY                1
CNT_PAYMENT                    279990
NAME_YIELD_GROUP                    1
PRODUCT_COMBINATION               266
DAYS_FIRST_DRAWING             506146
DAYS_FIRST_DUE                 506146
DAYS_LAST_DUE_1ST_VERSION      506146
DAYS_LAST_DUE                  506146
DAYS_TERMINATION               506146
NFLAG_INSURED_ON_APPROVAL      506146
dtype: int64
```

In [145…

```
appsDF.columns
```

Out[145…

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

feature engineering for prevApp table

In [146…

```
features = ['AMT_ANNUITY', 'AMT_APPLICATION']
agg_op_features = {}
cols =list()
agg_func_list=["mean", "min", "max"]

for f in features: #build agg dictionary
    agg_op_features[f] = agg_func_list
    cols.append(f"{f}_{func}" for func in agg_func_list)

print(f"{appsDF[features].describe()}")

result = appsDF.groupby(["SK_ID_CURR"]).agg(agg_op_features)
result.columns = ["_".join(x) for x in result.columns.ravel()]
#result.columns.droplevel() #drop 1 of the header row but keep the feature name
result = result.reset_index(level=["SK_ID_CURR"])
result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - result['AMT_AP
print(f"result.shape: {result.shape}")
result[0:10]
```

```
         AMT_ANNUITY  AMT_APPLICATION
count  981354.000000     1.261346e+06
mean    15884.875977     1.741193e+05
```

```
std      14737.920898     2.918359e+05
min          0.000000     0.000000e+00
25%       6299.685059     1.900462e+04
50%      11250.000000     7.084348e+04
75%      20518.875000     1.800000e+05
max     418058.156250     6.905160e+06
result.shape: (320452, 8)
```

Out[146…

	SK_ID_CURR	AMT_ANNUITY_mean	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_APPLICATIO
0	100001	3951.000000	3951.000000	3951.000000	24835
1	100002	9251.775391	9251.775391	9251.775391	179055
2	100003	56553.988281	6737.310059	98356.992188	435436
3	100004	5357.250000	5357.250000	5357.250000	24282
4	100005	NaN	NaN	NaN	0
5	100006	26682.007812	13500.000000	39954.511719	289069
6	100007	10198.808594	1834.290039	16509.599609	140136
7	100008	15839.696289	8019.089844	25309.574219	15570
8	100009	8417.955078	7435.845215	8996.759766	64092
9	100011	18303.195312	9000.000000	31295.250000	202732

In [147…

```
result.isna().sum()
```

Out[147…

```
SK_ID_CURR                  0
AMT_ANNUITY_mean         4586
AMT_ANNUITY_min          4586
AMT_ANNUITY_max          4586
AMT_APPLICATION_mean        0
AMT_APPLICATION_min         0
AMT_APPLICATION_max         0
range_AMT_APPLICATION       0
dtype: int64
```

Feature Engineering for All Tables

Choosing Highly correlated features from all input datasets

In [148…

```
def correlation_files_target(df_name):
    A = datasets["application_train"].copy()
    B = datasets[df_name].copy()
    correlation_matrix =  pd.concat([A.TARGET, B], axis=1).corr().filter(B.colum
    return correlation_matrix
```

In [149…

```
def correlation_matrix(datasets):
    for ds_name in datasets:
        correlation_matrix = correlation_files_target(ds_name)
        print(f"Correlation of the {ds_name} against the Target is :\n{correlati
        print()
        print("---"*20)
```

In [150…

```
dataset_names = list(datasets.keys())[2:]
correlation_matrix(dataset_names)
```

```
Correlation of the bureau against the Target is :
DAYS_CREDIT_UPDATE         0.002159
DAYS_CREDIT_ENDDATE        0.002048
SK_ID_BUREAU               0.001550
DAYS_CREDIT                0.001443
AMT_CREDIT_SUM             0.000218
DAYS_ENDDATE_FACT          0.000204
AMT_ANNUITY                0.000189
AMT_CREDIT_MAX_OVERDUE    -0.000389
CNT_CREDIT_PROLONG        -0.000495
AMT_CREDIT_SUM_LIMIT      -0.000558
AMT_CREDIT_SUM_DEBT       -0.000946
SK_ID_CURR                -0.001070
AMT_CREDIT_SUM_OVERDUE    -0.001464
CREDIT_DAY_OVERDUE        -0.001815
Name: TARGET, dtype: float64

------------------------------------------------------------
Correlation of the bureau_balance against the Target is :
SK_ID_BUREAU       0.001223
MONTHS_BALANCE    -0.005262
Name: TARGET, dtype: float64

------------------------------------------------------------
Correlation of the credit_card_balance against the Target is :
CNT_DRAWINGS_ATM_CURRENT      0.001908
AMT_DRAWINGS_ATM_CURRENT      0.001520
AMT_INST_MIN_REGULARITY       0.001435
SK_ID_CURR                    0.001086
AMT_CREDIT_LIMIT_ACTUAL       0.000515
AMT_BALANCE                   0.000448
SK_ID_PREV                    0.000446
AMT_RECIVABLE                 0.000412
AMT_TOTAL_RECEIVABLE          0.000407
AMT_RECEIVABLE_PRINCIPAL      0.000383
SK_DPD                        0.000092
SK_DPD_DEF                   -0.000201
CNT_INSTALMENT_MATURE_CUM    -0.000342
MONTHS_BALANCE               -0.000768
AMT_PAYMENT_CURRENT          -0.001129
AMT_PAYMENT_TOTAL_CURRENT    -0.001395
AMT_DRAWINGS_CURRENT         -0.001419
CNT_DRAWINGS_CURRENT         -0.001764
CNT_DRAWINGS_OTHER_CURRENT   -0.001833
CNT_DRAWINGS_POS_CURRENT     -0.002387
AMT_DRAWINGS_OTHER_CURRENT   -0.002672
AMT_DRAWINGS_POS_CURRENT     -0.003518
Name: TARGET, dtype: float64

------------------------------------------------------------
Correlation of the installments_payments against the Target is :
SK_ID_PREV                 0.002891
NUM_INSTALMENT_VERSION     0.002511
NUM_INSTALMENT_NUMBER      0.000626
SK_ID_CURR                -0.000781
AMT_PAYMENT               -0.003512
DAYS_INSTALMENT           -0.003956
AMT_INSTALMENT            -0.003972
DAYS_ENTRY_PAYMENT        -0.004046
```

```
Name: TARGET, dtype: float64

------------------------------------------------------------
Correlation of the previous_application against the Target is :
AMT_DOWN_PAYMENT              0.002496
CNT_PAYMENT                   0.002341
DAYS_LAST_DUE_1ST_VERSION     0.001908
AMT_CREDIT                    0.001833
AMT_APPLICATION               0.001689
AMT_GOODS_PRICE               0.001676
SK_ID_CURR                    0.001107
NFLAG_INSURED_ON_APPROVAL     0.000879
RATE_DOWN_PAYMENT             0.000850
RATE_INTEREST_PRIMARY         0.000548
SK_ID_PREV                    0.000362
DAYS_DECISION                -0.000482
AMT_ANNUITY                  -0.000492
DAYS_FIRST_DUE               -0.000943
SELLERPLACE_AREA             -0.000954
DAYS_TERMINATION             -0.001072
NFLAG_LAST_APPL_IN_DAY       -0.001256
DAYS_FIRST_DRAWING           -0.001293
DAYS_LAST_DUE                -0.001940
HOUR_APPR_PROCESS_START      -0.002285
RATE_INTEREST_PRIVILEGED     -0.026419
Name: TARGET, dtype: float64

------------------------------------------------------------
Correlation of the POS_CASH_balance against the Target is :
CNT_INSTALMENT_FUTURE     0.002811
MONTHS_BALANCE            0.002775
SK_ID_PREV                0.002164
CNT_INSTALMENT            0.001434
SK_DPD                    0.000050
SK_ID_CURR               -0.000136
SK_DPD_DEF               -0.001362
Name: TARGET, dtype: float64

------------------------------------------------------------
```

feature transformer for prevApp table

In [151…

```
# Create aggregate features (via pipeline)
class prevAppsFeaturesAggregater(BaseEstimator, TransformerMixin):
    def __init__(self, features=None): # no *args or **kargs
        self.features = features
        self.agg_op_features = {}
        for f in features:
            #self.agg_op_features[f] = {f"{f}_{func}":func for func in ["min", "
            self.agg_op_features[f] = ["min","max","mean"]

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        #from IPython.core.debugger import Pdb as pdb;    pdb().set_trace() #bre
        result = X.groupby(["SK_ID_CURR"]).agg(self.agg_op_features)
        #result.columns = result.columns.droplevel()
        result.columns = ["_".join(x) for x in result.columns.ravel()]
        result = result.reset_index(level=["SK_ID_CURR"])
        result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - result
```

```
        return result # return dataframe with the join key "SK_ID_CURR"


from sklearn.pipeline import make_pipeline
def test_driver_prevAppsFeaturesAggregater(df, features):
    print(f"df.shape: {df.shape}\n")
    print(f"df[{features}][0:5]: \n{df[features][0:5]}")
    test_pipeline = make_pipeline(prevAppsFeaturesAggregater(features))
    return(test_pipeline.fit_transform(df))

features = ['AMT_ANNUITY', 'AMT_APPLICATION']
features = ['AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE',
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CNT_PAYMENT',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION']
features = ['AMT_ANNUITY', 'AMT_APPLICATION']
res = test_driver_prevAppsFeaturesAggregater(appsDF, features)
print(f"HELLO")
print(f"Test driver: \n{res[0:10]}")
print(f"input[features][0:10]: \n{appsDF[0:10]}")


# QUESTION, should we lower case df['OCCUPATION_TYPE'] as Sales staff != 'Sales
```

```
df.shape: (1261346, 37)

df[['AMT_ANNUITY', 'AMT_APPLICATION']][0:5]:
    AMT_ANNUITY  AMT_APPLICATION
0   1730.430054          17145.0
1  25188.615234         607500.0
2  15060.735352         112500.0
3  47041.335938         450000.0
4  31924.394531         337500.0
HELLO
Test driver:
   SK_ID_CURR  AMT_ANNUITY_min  AMT_ANNUITY_max  AMT_ANNUITY_mean  \
0      100001      3951.000000      3951.000000       3951.000000
1      100002      9251.775391      9251.775391       9251.775391
2      100003      6737.310059     98356.992188      56553.988281
3      100004      5357.250000      5357.250000       5357.250000
4      100005              NaN              NaN               NaN
5      100006     13500.000000     39954.511719      26682.007812
6      100007      1834.290039     16509.599609      10198.808594
7      100008      8019.089844     25309.574219      15839.696289
8      100009      7435.845215      8996.759766       8417.955078
9      100011      9000.000000     31295.250000      18303.195312

   AMT_APPLICATION_min  AMT_APPLICATION_max  AMT_APPLICATION_mean  \
0              24835.5              24835.5          24835.500000
1             179055.0             179055.0         179055.000000
2              68809.5             900000.0         435436.500000
3              24282.0              24282.0          24282.000000
4                  0.0                  0.0              0.000000
5                  0.0             675000.0         289069.500000
6              17176.5             247500.0         140136.296875
7                  0.0             450000.0         155701.796875
8              46876.5              98239.5          64092.000000
9                  0.0             675000.0         202732.875000
```

```
   range_AMT_APPLICATION
0                    0.0
1                    0.0
2               831190.5
3                    0.0
4                    0.0
5               675000.0
6               230323.5
7               450000.0
8                51363.0
9               675000.0
input[features][0:10]:
   SK_ID_PREV  SK_ID_CURR NAME_CONTRACT_TYPE   AMT_ANNUITY  AMT_APPLICATION  \
0     2030495      271877     Consumer loans   1730.430054          17145.0
1     2802425      108129         Cash loans  25188.615234         607500.0
2     2523466      122040         Cash loans  15060.735352         112500.0
3     2819243      176158         Cash loans  47041.335938         450000.0
4     1784265      202054         Cash loans  31924.394531         337500.0
5     1383531      199383         Cash loans  23703.929688         315000.0
6     2315218      175704         Cash loans           NaN              0.0
7     1656711      296299         Cash loans           NaN              0.0
8     2367563      342292         Cash loans           NaN              0.0
9     2579447      334349         Cash loans           NaN              0.0

   AMT_CREDIT  AMT_DOWN_PAYMENT  AMT_GOODS_PRICE WEEKDAY_APPR_PROCESS_START  \
0     17145.0               0.0          17145.0                  SATURDAY
1    679671.0               NaN         607500.0                  THURSDAY
2    136444.5               NaN         112500.0                   TUESDAY
3    470790.0               NaN         450000.0                    MONDAY
4    404055.0               NaN         337500.0                  THURSDAY
5    340573.5               NaN         315000.0                  SATURDAY
6         0.0               NaN              NaN                   TUESDAY
7         0.0               NaN              NaN                    MONDAY
8         0.0               NaN              NaN                    MONDAY
9         0.0               NaN              NaN                  SATURDAY

   HOUR_APPR_PROCESS_START  ... NAME_SELLER_INDUSTRY  CNT_PAYMENT  \
0                     15.0  ...         Connectivity         12.0
1                     11.0  ...                  XNA         36.0
2                     11.0  ...                  XNA         12.0
3                      7.0  ...                  XNA         12.0
4                      9.0  ...                  XNA         24.0
5                      8.0  ...                  XNA         18.0
6                     11.0  ...                  XNA          NaN
7                      7.0  ...                  XNA          NaN
8                     15.0  ...                  XNA          NaN
9                     15.0  ...                  XNA          NaN

   NAME_YIELD_GROUP       PRODUCT_COMBINATION  DAYS_FIRST_DRAWING  \
0            middle  POS mobile with interest            365243.0
1       low_action          Cash X-Sell: low            365243.0
2              high         Cash X-Sell: high            365243.0
3            middle       Cash X-Sell: middle            365243.0
4              high         Cash Street: high                 NaN
5       low_normal          Cash X-Sell: low            365243.0
6               XNA                     Cash                 NaN
7               XNA                     Cash                 NaN
8               XNA                     Cash                 NaN
9               XNA                     Cash                 NaN

  DAYS_FIRST_DUE DAYS_LAST_DUE_1ST_VERSION  DAYS_LAST_DUE DAYS_TERMINATION  \
0          -42.0                    300.0          -42.0            -37.0
1         -134.0                    916.0       365243.0         365243.0
2         -271.0                     59.0       365243.0         365243.0
3         -482.0                   -152.0         -182.0           -177.0
```

```
4            NaN                    NaN           NaN            NaN
5         -654.0                 -144.0        -144.0         -137.0
6            NaN                    NaN           NaN            NaN
7            NaN                    NaN           NaN            NaN
8            NaN                    NaN           NaN            NaN
9            NaN                    NaN           NaN            NaN

  NFLAG_INSURED_ON_APPROVAL
0                       0.0
1                       1.0
2                       1.0
3                       1.0
4                       NaN
5                       1.0
6                       NaN
7                       NaN
8                       NaN
9                       NaN

[10 rows x 37 columns]
```

In [152…

```
agg_funcs = ['min', 'max', 'mean', 'count', 'sum']

prevApps = datasets['previous_application']
prevApps_features = ['AMT_ANNUITY', 'AMT_APPLICATION']

bureau = datasets['bureau']
bureau_features = ['AMT_ANNUITY', 'AMT_CREDIT_SUM']

bureau_bal = datasets['bureau_balance']
bureau_bal_features = ['MONTHS_BALANCE']

cc_bal = datasets['credit_card_balance']
cc_bal_features = ['MONTHS_BALANCE', 'AMT_BALANCE', 'CNT_INSTALMENT_MATURE_CUM']

installments_pmnts = datasets['installments_payments']
installments_pmnts_features = ['AMT_INSTALMENT', 'AMT_PAYMENT']
```

Feature Aggregator

In [153…

```
# Pipelines
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import make_pipeline, Pipeline, FeatureUnion
from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncoder


class FeaturesAggregator(BaseEstimator, TransformerMixin):
    def __init__(self, file_name=None, features=None, funcs=None, primary_id=Non
        self.file_name = file_name
        self.features = features
        self.primary_id = primary_id
        self.funcs = funcs
        self.agg_op_features = {}
        for f in self.features:
            temp = {f"{file_name}_{f}_{func}":func for func in self.funcs}
            self.agg_op_features[f]=[(k, v) for k, v in temp.items()]
        print(self.agg_op_features)
```

```
    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        #from IPython.core.debugger import Pdb as pdb;    pdb().set_trace() #bre
        result = X.groupby([self.primary_id]).agg(self.agg_op_features)
        result.columns = result.columns.droplevel()
        result = result.reset_index(level=[self.primary_id])
        return result
```

In [154…

```
class engineer_features(BaseEstimator, TransformerMixin):
    def __init__(self, features=None):
        self

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):

# FROM APPLICATION
        # ADD INCOME CREDIT PERCENTAGE
        X['ef_INCOME_CREDIT_PERCENT'] = (
            X.AMT_INCOME_TOTAL / X.AMT_CREDIT).replace(np.inf, 0)

        # ADD INCOME PER FAMILY MEMBER
        X['ef_FAM_MEMBER_INCOME'] = (
            X.AMT_INCOME_TOTAL / X.CNT_FAM_MEMBERS).replace(np.inf, 0)

        # ADD ANNUITY AS PERCENTAGE OF ANNUAL INCOME
        X['ef_ANN_INCOME_PERCENT'] = (
            X.AMT_ANNUITY / X.AMT_INCOME_TOTAL).replace(np.inf, 0)

# FROM MERGED PREVIOUS APPLICATION
        # ADD PREVIOUS APPLICATION RANGE
#        X['ef_prevApps_AMT_APPLICATION_RANGE'] = (
#            X.prevApps_AMT_ANNUITY_max - X.prevApps_AMT_ANNUITY_min).replace(np

# FROM MERGED BUREAU
        # ADD BUREAU CREDIT RANGE
#        X['ef_bureau_AMT_CREDIT_RANGE'] = (
#            X.bureau_AMT_CREDIT_SUM_max - X.bureau_AMT_CREDIT_SUM_min).replace(

        return X
```

Prepare Datasets

In [155…

```
appsTrainDF = datasets['application_train']
appsTestDF = datasets['application_test']
prevAppsDF = datasets["previous_application"] #prev app
bureauDF = datasets["bureau"] #bureau

bureaubalDF = datasets['bureau_balance']
ccbalDF = datasets["credit_card_balance"] #credit card
installmentspaymentsDF = datasets["installments_payments"] #installments payment
pos_cash_bal_DF = datasets["POS_CASH_balance"] #pos_cash_balance
```

Teritary Datasets

The tertiary datasets or tables refer to bureau_balance, POS_CASH_balance, instalments_payments, credit_card_balance

In [156…

```
tertiary_datasets = ['bureau_balance', 'POS_CASH_balance', 'instalments_payments
                     'credit_card_balance']
```

Third Tier Custom - Domain Knowledge based features

Any domain based features that will aid in a better model have been included here. In the table credit card balance the payment difference can be value to predict risk.

In [157…

```
# Difference between the monthly amount paid - the expected monthly amount
ccbalDF['payment_diff_curr_pay'] = ccbalDF['AMT_PAYMENT_TOTAL_CURRENT'] - ccbalD
ccbalDF['payment_diff_min_pay'] = ccbalDF['AMT_PAYMENT_TOTAL_CURRENT'] - ccbalDF
 # Difference between the monthly amount paid - the minimum monthly amount
```

Third Tier Datasets Numerical feature Aggregation

In [158…

```
 #function to get the numerical features
def get_numattribs(ds_name):
    num_attribs=(datasets[ds_name].select_dtypes(exclude=['object']).columns.tol
    print()
    print('Numerical attributes for',ds_name,' : ',num_attribs)
    print()
    return num_attribs
```

Feature Aggregation for tertiary datasets

In [159…

```
agg_funcs = ['min', 'max']

primary_id1 = "SK_ID_PREV"
primary_id2 = "SK_ID_BUREAU"


posBal_features = ['MONTHS_BALANCE','CNT_INSTALMENT','CNT_INSTALMENT_FUTURE']
instalPay_features = ['DAYS_INSTALMENT','AMT_INSTALMENT']
ccBal_features = ['AMT_BALANCE','AMT_DRAWINGS_CURRENT','payment_diff_curr_pay','
burBal_features = ['MONTHS_BALANCE']


prevApps_features = ['AMT_APPLICATION','AMT_CREDIT','AMT_ANNUITY'] # NO MISSING
bureau_features = ['AMT_CREDIT_SUM']

cc_features_pipeline = Pipeline([

    ('credit_card_num_aggregator', FeaturesAggregator('credit_card_balance',ccBa
    ])

installment_features_pipeline = Pipeline([
    ('installment_num_aggregator', FeaturesAggregator('installments_payments',in
    ])
```

```
POS_CASH_balance_pipeline = Pipeline([
    ('POS_CASH_balance', FeaturesAggregator('POS_CASH_balance' ,posBal_features
    ])

bureau_balance_feature_pipeline = Pipeline([
    ('bureau_balance', FeaturesAggregator('bureau_balance' ,burBal_features , ag
    ])
```

```
{'AMT_BALANCE': [('credit_card_balance_AMT_BALANCE_min', 'min'), ('credit_card_b
alance_AMT_BALANCE_max', 'max')], 'AMT_DRAWINGS_CURRENT': [('credit_card_balance
_AMT_DRAWINGS_CURRENT_min', 'min'), ('credit_card_balance_AMT_DRAWINGS_CURRENT_m
ax', 'max')], 'payment_diff_curr_pay': [('credit_card_balance_payment_diff_curr_
pay_min', 'min'), ('credit_card_balance_payment_diff_curr_pay_max', 'max')], 'pa
yment_diff_min_pay': [('credit_card_balance_payment_diff_min_pay_min', 'min'),
('credit_card_balance_payment_diff_min_pay_max', 'max')]}
{'DAYS_INSTALMENT': [('installments_payments_DAYS_INSTALMENT_min', 'min'), ('ins
tallments_payments_DAYS_INSTALMENT_max', 'max')], 'AMT_INSTALMENT': [('installme
nts_payments_AMT_INSTALMENT_min', 'min'), ('installments_payments_AMT_INSTALMENT
_max', 'max')]}
{'MONTHS_BALANCE': [('POS_CASH_balance_MONTHS_BALANCE_min', 'min'), ('POS_CASH_b
alance_MONTHS_BALANCE_max', 'max')], 'CNT_INSTALMENT': [('POS_CASH_balance_CNT_I
NSTALMENT_min', 'min'), ('POS_CASH_balance_CNT_INSTALMENT_max', 'max')], 'CNT_IN
STALMENT_FUTURE': [('POS_CASH_balance_CNT_INSTALMENT_FUTURE_min', 'min'), ('POS_
CASH_balance_CNT_INSTALMENT_FUTURE_max', 'max')]}
{'MONTHS_BALANCE': [('bureau_balance_MONTHS_BALANCE_min', 'min'), ('bureau_balan
ce_MONTHS_BALANCE_max', 'max')]}
```

Merge Tertiary level data with secondary level data

In [160…

```
bureaubal_aggregated = bureau_balance_feature_pipeline.fit_transform(bureaubalDF
ccblance_aggregated = cc_features_pipeline.fit_transform(ccbalDF)
installments_pmnts_aggregated = installment_features_pipeline.fit_transform(inst
pos_cash_bal_aggregated = POS_CASH_balance_pipeline.fit_transform(pos_cash_bal_D
```

Merging the aggregated features for pos_cash_bal , installments_pmnts , credit card balance with Previous application

In [161…

```
prevApps_ThirdTierMerge = True

posBal_join_feature = 'SK_ID_PREV'
prevApps_join_feature = 'SK_ID_CURR'
bureau_join_feature = 'SK_ID_CURR'
instalPay_join_feature = 'SK_ID_PREV'
ccBal_join_feature = 'SK_ID_PREV'
burBal_join_feature = 'SK_ID_BUREAU'


if prevApps_ThirdTierMerge:
  # Merge Datasets
    prevAppsDF = prevAppsDF.merge(pos_cash_bal_aggregated, how='left', on=posBal
    prevAppsDF = prevAppsDF.merge(installments_pmnts_aggregated, how='left', on=
    prevAppsDF = prevAppsDF.merge(ccblance_aggregated, how='left', on=ccBal_join
    prevApps_features.extend(installments_pmnts_aggregated.columns[1:])
    prevApps_features.extend(ccblance_aggregated.columns[1:])
    prevApps_features.extend(pos_cash_bal_aggregated.columns[1:])
    reduce_mem_usage(prevAppsDF,'previous_application')
```

```
Memory usage of dataframe previous_application is 312.76 MB
```

```
Memory usage after optimization is: 310.35 MB
Decreased by 0.8%
```

Merging the aggregated features the dataset Bureau Balance with Bureau as per the data model.

In [162…

```
bureau_ThirdTierMerge = True

if bureau_ThirdTierMerge:
    bureauDF = bureauDF.merge(bureaubal_aggregated, how='left', on=burBal_join_f
    bureau_features.extend(bureaubal_aggregated.columns[1:])
    reduce_mem_usage(bureauDF,'bureau')
```

```
Memory usage of dataframe bureau is 132.59 MB
Memory usage after optimization is: 132.59 MB
Decreased by 0.0%
```

Secondary Datasets

Second Tier datasets Numerical feature aggregation

In [163…

```
agg_funcs = ['count', 'max', 'min', 'sum']
primary_id1 = "SK_ID_CURR"


prevApps_feature_pipeline = Pipeline([
    ('prevApps', FeaturesAggregator('prevApps' ,prevApps_features , agg_funcs, p
    ])

bureau_feature_pipeline = Pipeline([
    ('bureau', FeaturesAggregator('bureau' ,bureau_features , agg_funcs, primary
    ])
```

```
{'AMT_APPLICATION': [('prevApps_AMT_APPLICATION_count', 'count'), ('prevApps_AMT
_APPLICATION_max', 'max'), ('prevApps_AMT_APPLICATION_min', 'min'), ('prevApps_A
MT_APPLICATION_sum', 'sum')], 'AMT_CREDIT': [('prevApps_AMT_CREDIT_count', 'coun
t'), ('prevApps_AMT_CREDIT_max', 'max'), ('prevApps_AMT_CREDIT_min', 'min'), ('p
revApps_AMT_CREDIT_sum', 'sum')], 'AMT_ANNUITY': [('prevApps_AMT_ANNUITY_count',
'count'), ('prevApps_AMT_ANNUITY_max', 'max'), ('prevApps_AMT_ANNUITY_min', 'mi
n'), ('prevApps_AMT_ANNUITY_sum', 'sum')], 'installments_payments_DAYS_INSTALMEN
T_min': [('prevApps_installments_payments_DAYS_INSTALMENT_min_count', 'count'),
('prevApps_installments_payments_DAYS_INSTALMENT_min_max', 'max'), ('prevApps_in
stallments_payments_DAYS_INSTALMENT_min_min', 'min'), ('prevApps_installments_pa
yments_DAYS_INSTALMENT_min_sum', 'sum')], 'installments_payments_DAYS_INSTALMENT
_max': [('prevApps_installments_payments_DAYS_INSTALMENT_max_count', 'count'),
('prevApps_installments_payments_DAYS_INSTALMENT_max_max', 'max'), ('prevApps_in
stallments_payments_DAYS_INSTALMENT_max_min', 'min'), ('prevApps_installments_pa
yments_DAYS_INSTALMENT_max_sum', 'sum')], 'installments_payments_AMT_INSTALMENT_
min': [('prevApps_installments_payments_AMT_INSTALMENT_min_count', 'count'), ('p
revApps_installments_payments_AMT_INSTALMENT_min_max', 'max'), ('prevApps_instal
lments_payments_AMT_INSTALMENT_min_min', 'min'), ('prevApps_installments_payment
s_AMT_INSTALMENT_min_sum', 'sum')], 'installments_payments_AMT_INSTALMENT_max':
[('prevApps_installments_payments_AMT_INSTALMENT_max_count', 'count'), ('prevApp
s_installments_payments_AMT_INSTALMENT_max_max', 'max'), ('prevApps_installments
_payments_AMT_INSTALMENT_max_min', 'min'), ('prevApps_installments_payments_AMT_
INSTALMENT_max_sum', 'sum')], 'credit_card_balance_AMT_BALANCE_min': [('prevApps
_credit_card_balance_AMT_BALANCE_min_count', 'count'), ('prevApps_credit_card_ba
lance_AMT_BALANCE_min_max', 'max'), ('prevApps_credit_card_balance_AMT_BALANCE_m
in_min', 'min'), ('prevApps_credit_card_balance_AMT_BALANCE_min_sum', 'sum')],
'credit_card_balance_AMT_BALANCE_max': [('prevApps_credit_card_balance_AMT_BALAN
```

```
CE_max_count', 'count'), ('prevApps_credit_card_balance_AMT_BALANCE_max_max', 'm
ax'), ('prevApps_credit_card_balance_AMT_BALANCE_max_min', 'min'), ('prevApps_cr
edit_card_balance_AMT_BALANCE_max_sum', 'sum')], 'credit_card_balance_AMT_DRAWIN
GS_CURRENT_min': [('prevApps_credit_card_balance_AMT_DRAWINGS_CURRENT_min_coun
t', 'count'), ('prevApps_credit_card_balance_AMT_DRAWINGS_CURRENT_min_max', 'ma
x'), ('prevApps_credit_card_balance_AMT_DRAWINGS_CURRENT_min_min', 'min'), ('pre
vApps_credit_card_balance_AMT_DRAWINGS_CURRENT_min_sum', 'sum')], 'credit_card_b
alance_AMT_DRAWINGS_CURRENT_max': [('prevApps_credit_card_balance_AMT_DRAWINGS_C
URRENT_max_count', 'count'), ('prevApps_credit_card_balance_AMT_DRAWINGS_CURRENT
_max_max', 'max'), ('prevApps_credit_card_balance_AMT_DRAWINGS_CURRENT_max_min',
'min'), ('prevApps_credit_card_balance_AMT_DRAWINGS_CURRENT_max_sum', 'sum')],
'credit_card_balance_payment_diff_curr_pay_min': [('prevApps_credit_card_balance
_payment_diff_curr_pay_min_count', 'count'), ('prevApps_credit_card_balance_paym
ent_diff_curr_pay_min_max', 'max'), ('prevApps_credit_card_balance_payment_diff_
curr_pay_min_min', 'min'), ('prevApps_credit_card_balance_payment_diff_curr_pay_
min_sum', 'sum')], 'credit_card_balance_payment_diff_curr_pay_max': [('prevApps_
credit_card_balance_payment_diff_curr_pay_max_count', 'count'), ('prevApps_credi
t_card_balance_payment_diff_curr_pay_max_max', 'max'), ('prevApps_credit_card_ba
lance_payment_diff_curr_pay_max_min', 'min'), ('prevApps_credit_card_balance_pay
ment_diff_curr_pay_max_sum', 'sum')], 'credit_card_balance_payment_diff_min_pay_
min': [('prevApps_credit_card_balance_payment_diff_min_pay_min_count', 'count'),
('prevApps_credit_card_balance_payment_diff_min_pay_min_max', 'max'), ('prevApps
_credit_card_balance_payment_diff_min_pay_min_min', 'min'), ('prevApps_credit_ca
rd_balance_payment_diff_min_pay_min_sum', 'sum')], 'credit_card_balance_payment_
diff_min_pay_max': [('prevApps_credit_card_balance_payment_diff_min_pay_max_coun
t', 'count'), ('prevApps_credit_card_balance_payment_diff_min_pay_max_max', 'ma
x'), ('prevApps_credit_card_balance_payment_diff_min_pay_max_min', 'min'), ('pre
vApps_credit_card_balance_payment_diff_min_pay_max_sum', 'sum')], 'POS_CASH_bala
nce_MONTHS_BALANCE_min': [('prevApps_POS_CASH_balance_MONTHS_BALANCE_min_count',
'count'), ('prevApps_POS_CASH_balance_MONTHS_BALANCE_min_max', 'max'), ('prevApp
s_POS_CASH_balance_MONTHS_BALANCE_min_min', 'min'), ('prevApps_POS_CASH_balance_
MONTHS_BALANCE_min_sum', 'sum')], 'POS_CASH_balance_MONTHS_BALANCE_max': [('prev
Apps_POS_CASH_balance_MONTHS_BALANCE_max_count', 'count'), ('prevApps_POS_CASH_b
alance_MONTHS_BALANCE_max_max', 'max'), ('prevApps_POS_CASH_balance_MONTHS_BALAN
CE_max_min', 'min'), ('prevApps_POS_CASH_balance_MONTHS_BALANCE_max_sum', 'su
m')], 'POS_CASH_balance_CNT_INSTALMENT_min': [('prevApps_POS_CASH_balance_CNT_IN
STALMENT_min_count', 'count'), ('prevApps_POS_CASH_balance_CNT_INSTALMENT_min_ma
x', 'max'), ('prevApps_POS_CASH_balance_CNT_INSTALMENT_min_min', 'min'), ('prevA
pps_POS_CASH_balance_CNT_INSTALMENT_min_sum', 'sum')], 'POS_CASH_balance_CNT_INS
TALMENT_max': [('prevApps_POS_CASH_balance_CNT_INSTALMENT_max_count', 'count'),
('prevApps_POS_CASH_balance_CNT_INSTALMENT_max_max', 'max'), ('prevApps_POS_CASH
_balance_CNT_INSTALMENT_max_min', 'min'), ('prevApps_POS_CASH_balance_CNT_INSTAL
MENT_max_sum', 'sum')], 'POS_CASH_balance_CNT_INSTALMENT_FUTURE_min': [('prevApp
s_POS_CASH_balance_CNT_INSTALMENT_FUTURE_min_count', 'count'), ('prevApps_POS_CA
SH_balance_CNT_INSTALMENT_FUTURE_min_max', 'max'), ('prevApps_POS_CASH_balance_C
NT_INSTALMENT_FUTURE_min_min', 'min'), ('prevApps_POS_CASH_balance_CNT_INSTALMEN
T_FUTURE_min_sum', 'sum')], 'POS_CASH_balance_CNT_INSTALMENT_FUTURE_max': [('pre
vApps_POS_CASH_balance_CNT_INSTALMENT_FUTURE_max_count', 'count'), ('prevApps_PO
S_CASH_balance_CNT_INSTALMENT_FUTURE_max_max', 'max'), ('prevApps_POS_CASH_balan
ce_CNT_INSTALMENT_FUTURE_max_min', 'min'), ('prevApps_POS_CASH_balance_CNT_INSTA
LMENT_FUTURE_max_sum', 'sum')]}
{'AMT_CREDIT_SUM': [('bureau_AMT_CREDIT_SUM_count', 'count'), ('bureau_AMT_CREDI
T_SUM_max', 'max'), ('bureau_AMT_CREDIT_SUM_min', 'min'), ('bureau_AMT_CREDIT_SU
M_sum', 'sum')], 'bureau_balance_MONTHS_BALANCE_min': [('bureau_bureau_balance_M
ONTHS_BALANCE_min_count', 'count'), ('bureau_bureau_balance_MONTHS_BALANCE_min_m
ax', 'max'), ('bureau_bureau_balance_MONTHS_BALANCE_min_min', 'min'), ('bureau_b
ureau_balance_MONTHS_BALANCE_min_sum', 'sum')], 'bureau_balance_MONTHS_BALANCE_m
ax': [('bureau_bureau_balance_MONTHS_BALANCE_max_count', 'count'), ('bureau_bure
au_balance_MONTHS_BALANCE_max_max', 'max'), ('bureau_bureau_balance_MONTHS_BALAN
CE_max_min', 'min'), ('bureau_bureau_balance_MONTHS_BALANCE_max_sum', 'sum')]}
```

In [164…

```
prevApps_aggregated = prevApps_feature_pipeline.fit_transform(prevAppsDF)
bureau_aggregated = bureau_feature_pipeline.fit_transform(bureauDF)
```

In [165…
```
prevApps_aggregated.shape
```

Out[165… (320452, 85)

Second Tier Custom - Domain Knowledge based features

In [166…
```
prevApps_aggregated['prevApps_AMT_APPLICATION_avg'] = (
   prevApps_aggregated['prevApps_AMT_APPLICATION_sum'] / prevApps_aggregated['pr

prevApps_aggregated['prevApps_AMT_APPLICATION_range'] = (
   prevApps_aggregated['prevApps_AMT_APPLICATION_max'] - prevApps_aggregated['pr


bureau_aggregated['bureau_AMT_CREDIT_SUM_avg'] = (
   bureau_aggregated['bureau_AMT_CREDIT_SUM_sum'] / bureau_aggregated['bureau_AM

bureau_aggregated['bureau_AMT_APPLICATION_range'] = (
   bureau_aggregated['bureau_AMT_CREDIT_SUM_max'] - bureau_aggregated['bureau_AM
```

Primary Datasets

Merge Aggregated Dataset With Tier 1 Tables - Train and Test

Prior to merging with the Primary data, we will be dropping columns with more than 50% missing values because they are not reliable parameters.

In [167…
```
appsTrainDF = datasets["application_train"]
X_kaggle_test = datasets["application_test"]

df_missing = pd.DataFrame(np.round((appsTrainDF.isna().sum()/ appsTrainDF.shape[
df_missing_50_cols = df_missing[df_missing.Percent >= 50].index

# Drop
appsTrainDF.drop(columns=df_missing_50_cols, inplace=True)
X_kaggle_test.drop(columns=df_missing_50_cols, inplace=True)
appsTrainDF.shape
```

Out[167… (307511, 81)

In [168…
```
X_kaggle_test.shape
```

Out[168… (48744, 80)

Merging Secondary level data with Application Train&Test Data

In [169…
```
merge_all_data = True

if merge_all_data:
# 1. Join/Merge in prevApps Data
    appsTrainDF = appsTrainDF.merge(prevApps_aggregated, how='left', on='SK_ID_C
```

```
    X_kaggle_test = X_kaggle_test.merge(prevApps_aggregated, how='left', on='SK_

# 2. Join/Merge in bureau Data
    appsTrainDF = appsTrainDF.merge(bureau_aggregated, how='left', on="SK_ID_CUR
    X_kaggle_test = X_kaggle_test.merge(bureau_aggregated, how='left', on="SK_ID
    reduce_mem_usage(appsTrainDF,'application_train')
    reduce_mem_usage(X_kaggle_test,'application_test')
```

```
Memory usage of dataframe application_train is 193.85 MB
Memory usage after optimization is: 149.27 MB
Decreased by 23.0%
Memory usage of dataframe application_test is 30.68 MB
Memory usage after optimization is: 23.61 MB
Decreased by 23.0%
```

In [170…
```
appsTrainDF.shape
```

Out[170… (307511, 181)

In [171…
```
X_kaggle_test.shape
```

Out[171… (48744, 180)

Custom - Domain Knowledge based Features

In [172…
```
# Training dataset
appsTrainDF['CREDIT_INCOME_PCT'] = appsTrainDF['AMT_CREDIT'] / appsTrainDF['AMT_
appsTrainDF['ANNUITY_INCOME_PCT'] = appsTrainDF['AMT_ANNUITY'] / appsTrainDF['AM
appsTrainDF['CREDIT_TERM'] = appsTrainDF['AMT_ANNUITY'] / appsTrainDF['AMT_CREDI

# Test dataset
X_kaggle_test['DAYS_EMPLOYED_PCT'] = X_kaggle_test['DAYS_EMPLOYED'] / X_kaggle_t
X_kaggle_test['CREDIT_INCOME_PCT'] = X_kaggle_test['AMT_CREDIT'] / X_kaggle_test
X_kaggle_test['ANNUITY_INCOME_PCT'] = X_kaggle_test['AMT_ANNUITY'] / X_kaggle_te
X_kaggle_test['CREDIT_TERM'] = X_kaggle_test['AMT_ANNUITY'] / X_kaggle_test['AMT
```

Handle remaining missing values and null values

Fill NA values with 0, Execute Fillna(0)

In [173…
```
appsTrainDF[prevApps_aggregated.columns] = appsTrainDF[prevApps_aggregated.colum
X_kaggle_test[prevApps_aggregated.columns] = X_kaggle_test[prevApps_aggregated.c

appsTrainDF[bureau_aggregated.columns] = appsTrainDF[bureau_aggregated.columns].
X_kaggle_test[bureau_aggregated.columns] = X_kaggle_test[bureau_aggregated.colum
```

Total Numeric features in Application Train data

In [174…
```
appsTrainDF.select_dtypes(exclude=['object']).columns
```

Out[174…
```
Index(['SK_ID_CURR', 'TARGET', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
```

```
       ...
       'bureau_bureau_balance_MONTHS_BALANCE_min_sum',
       'bureau_bureau_balance_MONTHS_BALANCE_max_count',
       'bureau_bureau_balance_MONTHS_BALANCE_max_max',
       'bureau_bureau_balance_MONTHS_BALANCE_max_min',
       'bureau_bureau_balance_MONTHS_BALANCE_max_sum',
       'bureau_AMT_CREDIT_SUM_avg', 'bureau_AMT_APPLICATION_range',
       'CREDIT_INCOME_PCT', 'ANNUITY_INCOME_PCT', 'CREDIT_TERM'],
      dtype='object', length=171)
```

Total Categorical features in the application train data.

In [175…
```
appsTrainDF.select_dtypes(include=['object']).columns
```

Out[175…
```
Index(['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',
       'EMERGENCYSTATE_MODE'],
      dtype='object')
```

Deductions from the list of dtypes of the appsTrainDF

- There 172 numerical features.
- There are 13 categorical features.
- The categorical features dont show up in the final correlation matrix of the appsTrainDF.

In [176…
```
appsTrainDF.dtypes.value_counts()
```

Out[176…
```
float16    80
float32    50
int8       37
object     13
int32       2
int16       2
dtype: int64
```

In [177…
```
correlation_with_all_features = appsTrainDF.corr()
correlation_with_all_features["TARGET"].sort_values()
```

Out[177…
```
EXT_SOURCE_3                 -0.178918
EXT_SOURCE_2                 -0.160471
DAYS_EMPLOYED                -0.044932
FLOORSMAX_AVG                -0.044005
FLOORSMAX_MEDI               -0.043769
                                ...
DAYS_LAST_PHONE_CHANGE        0.055219
REGION_RATING_CLIENT          0.058899
REGION_RATING_CLIENT_W_CITY   0.060893
DAYS_BIRTH                    0.078239
TARGET                        1.000000
Name: TARGET, Length: 171, dtype: float64
```

In [178…
```
len(correlation_with_all_features.index)
```

Out[178…
```
171
```

In [179…

```
# set this value to choose the number of positive and negative correlated featur
print("    Total correlation of all the features.    " )

print("---"*15)
print(f"Top 15 negative correlated features")
print()
print(correlation_with_all_features.TARGET.sort_values(ascending = True).head(15
print()
print()
print(f"Top 15 positive correlated features")
print()
print(correlation_with_all_features.TARGET.sort_values(ascending = True).tail(15
```

```
    Total correlation of all the features.
---------------------------------------------
Top 15 negative correlated features

EXT_SOURCE_3                                               -0.178918
EXT_SOURCE_2                                               -0.160471
DAYS_EMPLOYED                                              -0.044932
FLOORSMAX_AVG                                              -0.044005
FLOORSMAX_MEDI                                             -0.043769
FLOORSMAX_MODE                                             -0.043228
AMT_GOODS_PRICE                                            -0.039645
REGION_POPULATION_RELATIVE                                 -0.037231
TOTALAREA_MODE                                             -0.032596
AMT_CREDIT                                                 -0.030369
FLAG_DOCUMENT_6                                            -0.028602
prevApps_POS_CASH_balance_CNT_INSTALMENT_min_count         -0.027760
prevApps_POS_CASH_balance_CNT_INSTALMENT_max_count         -0.027760
prevApps_POS_CASH_balance_CNT_INSTALMENT_FUTURE_min_count  -0.027759
prevApps_POS_CASH_balance_CNT_INSTALMENT_FUTURE_max_count  -0.027759
Name: TARGET, dtype: float64


Top 15 positive correlated features

prevApps_credit_card_balance_AMT_BALANCE_max_max    0.036072
prevApps_credit_card_balance_AMT_BALANCE_max_min    0.036129
prevApps_POS_CASH_balance_MONTHS_BALANCE_min_min    0.037533
prevApps_POS_CASH_balance_MONTHS_BALANCE_max_min    0.038676
DAYS_REGISTRATION                                   0.041975
FLAG_DOCUMENT_3                                     0.044346
REG_CITY_NOT_LIVE_CITY                              0.044395
FLAG_EMP_PHONE                                      0.045982
REG_CITY_NOT_WORK_CITY                              0.050994
DAYS_ID_PUBLISH                                     0.051457
DAYS_LAST_PHONE_CHANGE                              0.055219
REGION_RATING_CLIENT                                0.058899
REGION_RATING_CLIENT_W_CITY                         0.060893
DAYS_BIRTH                                          0.078239
TARGET                                              1.000000
Name: TARGET, dtype: float64
```

In [180…

```
tf_apps_train_final = list()

featureslist1 = correlation_with_all_features.TARGET.sort_values(ascending = Tru
featureslist2 = correlation_with_all_features.TARGET.sort_values(ascending = Tru
tf_apps_train_final = featureslist1 + featureslist2
```

In [181…

```
tf_apps_train_final
```

Out[181…
```
['EXT_SOURCE_3',
 'EXT_SOURCE_2',
 'DAYS_EMPLOYED',
 'FLOORSMAX_AVG',
 'FLOORSMAX_MEDI',
 'FLOORSMAX_MODE',
 'AMT_GOODS_PRICE',
 'REGION_POPULATION_RELATIVE',
 'TOTALAREA_MODE',
 'AMT_CREDIT',
 'FLAG_DOCUMENT_6',
 'prevApps_POS_CASH_balance_CNT_INSTALMENT_min_count',
 'prevApps_POS_CASH_balance_CNT_INSTALMENT_max_count',
 'prevApps_POS_CASH_balance_CNT_INSTALMENT_FUTURE_min_count',
 'prevApps_POS_CASH_balance_CNT_INSTALMENT_FUTURE_max_count',
 'prevApps_credit_card_balance_AMT_BALANCE_max_max',
 'prevApps_credit_card_balance_AMT_BALANCE_max_min',
 'prevApps_POS_CASH_balance_MONTHS_BALANCE_min_min',
 'prevApps_POS_CASH_balance_MONTHS_BALANCE_max_min',
 'DAYS_REGISTRATION',
 'FLAG_DOCUMENT_3',
 'REG_CITY_NOT_LIVE_CITY',
 'FLAG_EMP_PHONE',
 'REG_CITY_NOT_WORK_CITY',
 'DAYS_ID_PUBLISH',
 'DAYS_LAST_PHONE_CHANGE',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'DAYS_BIRTH',
 'TARGET']
```

In [182…
```
for idx in tf_apps_train_final:
    print(f"{idx:50}  {appsTrainDF[idx].dtypes}")
```

```
EXT_SOURCE_3                                       float16
EXT_SOURCE_2                                       float16
DAYS_EMPLOYED                                      int32
FLOORSMAX_AVG                                      float16
FLOORSMAX_MEDI                                     float16
FLOORSMAX_MODE                                     float16
AMT_GOODS_PRICE                                    float32
REGION_POPULATION_RELATIVE                         float16
TOTALAREA_MODE                                     float16
AMT_CREDIT                                         float32
FLAG_DOCUMENT_6                                    int8
prevApps_POS_CASH_balance_CNT_INSTALMENT_min_count  float16
prevApps_POS_CASH_balance_CNT_INSTALMENT_max_count  float16
prevApps_POS_CASH_balance_CNT_INSTALMENT_FUTURE_min_count  float16
prevApps_POS_CASH_balance_CNT_INSTALMENT_FUTURE_max_count  float16
prevApps_credit_card_balance_AMT_BALANCE_max_max    float32
prevApps_credit_card_balance_AMT_BALANCE_max_min    float32
prevApps_POS_CASH_balance_MONTHS_BALANCE_min_min    float16
prevApps_POS_CASH_balance_MONTHS_BALANCE_max_min    float16
DAYS_REGISTRATION                                  float16
FLAG_DOCUMENT_3                                    int8
REG_CITY_NOT_LIVE_CITY                             int8
FLAG_EMP_PHONE                                     int8
REG_CITY_NOT_WORK_CITY                             int8
DAYS_ID_PUBLISH                                    int16
DAYS_LAST_PHONE_CHANGE                             float16
REGION_RATING_CLIENT                               int8
REGION_RATING_CLIENT_W_CITY                        int8
```

```
DAYS_BIRTH                                    int16
TARGET                                        int8
```

In [183…

```
modeling_num_attrib = []
modeling_cat_attrib = []

for idx in tf_apps_train_final:
    if appsTrainDF[idx].dtypes not in ['object']:
        modeling_num_attrib.append(idx)
    else:
        modeling_cat_attrib.append(idx)

print(len(modeling_num_attrib))
print(len(modeling_cat_attrib))
```

```
30
0
```

In [184…

```
# Convert categorical features to numerical approximations (via pipeline)
class ClaimAttributesAdder(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        charlson_idx_dt = {'0': 0, '1-2': 2, '3-4': 4, '5+': 6}
        los_dt = {'1 day': 1, '2 days': 2, '3 days': 3, '4 days': 4, '5 days': 5
          '1- 2 weeks': 11, '2- 4 weeks': 21, '4- 8 weeks': 42, '26+ weeks': 180
        X['PayDelay'] = X['PayDelay'].apply(lambda x: int(x) if x != '162+' else
        X['DSFS'] = X['DSFS'].apply(lambda x: None if pd.isnull(x) else int(x[0]
        X['CharlsonIndex'] = X['CharlsonIndex'].apply(lambda x: charlson_idx_dt[
        X['LengthOfStay'] = X['LengthOfStay'].apply(lambda x: None if pd.isnull(
        return X
```

In [185…

```
from sklearn.base import BaseEstimator, TransformerMixin
import re

# Creates the following date features
# But could do so much more with these features
#    E.g.,
#      extract the domain address of the homepage and OneHotEncode it
#
# ['release_month','release_day','release_year', 'release_dayofweek','release_qu
class prep_OCCUPATION_TYPE(BaseEstimator, TransformerMixin):
    def __init__(self, features="OCCUPATION_TYPE"): # no *args or **kargs
        self.features = features
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        df = pd.DataFrame(X, columns=self.features)
        #from IPython.core.debugger import Pdb as pdb;    pdb().set_trace() #bre
        df['OCCUPATION_TYPE'] = df['OCCUPATION_TYPE'].apply(lambda x: 1. if x in
        #df.drop(self.features, axis=1, inplace=True)
        return np.array(df.values)  #return a Numpy Array to observe the pipelin


from sklearn.pipeline import make_pipeline
features = ["OCCUPATION_TYPE"]
def test_driver_prep_OCCUPATION_TYPE():
    print(f"X_train.shape: {X_train.shape}\n")
```

```
    print(f"X_train['name'][0:5]: \n{X_train[features][0:5]}")
    test_pipeline = make_pipeline(prep_OCCUPATION_TYPE(features))
    return(test_pipeline.fit_transform(X_train))

#x = test_driver_prep_OCCUPATION_TYPE()
#print(f"Test driver: \n{test_driver_prep_OCCUPATION_TYPE()[0:10, :]}")
#print(f"X_train['name'][0:10]: \n{X_train[features][0:10]}")


# QUESTION, should we lower case df['OCCUPATION_TYPE'] as Sales staff != 'Sales
```

Pipeline

In [186…
```
train_dataset=appsTrainDF
class_labels = ["No Default","Default"]
```

HCDR Preprocessing

Column Selector

In [187…
```
# Create a class to select numerical or categorical columns since Scikit-Learn d
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

Numerical Attributes

In [188…
```
num_attribs=train_dataset.select_dtypes(exclude=['object']).columns.tolist()
```

In [189…
```
num_attribs.remove('TARGET')
num_attribs.remove('SK_ID_CURR')
```

In [190…
```
len(num_attribs)
```

Out[190… 169

Numerical Pipeline definition

In [191…
```
num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('imputer', SimpleImputer(strategy='mean')),
        ('std_scaler', StandardScaler()),
    ])
```

OHE when previously unseen unique values in the test/validation set

Train, validation and Test sets (and the leakage problem we have mentioned previously):

Let's look at a small usecase to tell us how to deal with this:

- The OneHotEncoder is fitted to the training set, which means that for each unique value present in the training set, for each feature, a new column is created. Let's say we have 39 columns after the encoding up from 30 (before preprocessing).
- The output is a numpy array (when the option sparse=False is used), which has the disadvantage of losing all the information about the original column names and values.
- When we try to transform the test set, after having fitted the encoder to the training set, we obtain a `ValueError`. This is because the there are new, previously unseen unique values in the test set and the encoder doesn't know how to handle these values. In order to use both the transformed training and test sets in machine learning algorithms, we need them to have the same number of columns.

This last problem can be solved by using the option handle_unknown='ignore'of the OneHotEncoder, which, as the name suggests, will ignore previously unseen values when transforming the test set.

Here is a example that in action:

```
# Identify the categorical features we wish to consider.
cat_attribs = ['CODE_GENDER',
'FLAG_OWN_REALTY','FLAG_OWN_CAR','NAME_CONTRACT_TYPE',

'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_TYPE']

# Notice handle_unknown="ignore" in OHE which ignore values from the
validation/test that
# do NOT occur in the training set
cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

In [192…
```
cat_attribs=train_dataset.select_dtypes(include=['object']).columns.tolist()
```

In [193…
```
len(cat_attribs)
```

Out[193… 13

Categorical Pipeline definition

In [194…
```
# Notice handle_unknown="ignore" in OHE which ignore values from the validation/
# do NOT occur in the training set
cat_pipeline = Pipeline([
```

```
        ('selector', DataFrameSelector(cat_attribs)),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        #('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

Create Data Preparation Pipeline

With Feature union, combine numerical and categorical Pipeline together to prepare for Data pipeline

In [195…

```
data_prep_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline),
    ])
```

In [196…

```
selected_features = num_attribs + cat_attribs
tot_features = f"{len(selected_features)}:   Num:{len(num_attribs)},    Cat:{len
#Total Feature selected for processing
tot_features
```

Out[196… '182:   Num:169,   Cat:13'

Baseline model with Imbalanced Dataset

Create Train and Test Datasets

In [197…

```
# Split Sample to feed the pipeline and it will result in a new dataset that is
splits = 3

# Train Test split percentage
subsample_rate = 0.3

finaldf = np.array_split(train_dataset, splits)
X_train = finaldf[0][selected_features]
y_train = finaldf[0]['TARGET']
X_kaggle_test= X_kaggle_test[selected_features]

## split part of data
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, stratify=y
                                                    test_size=subsample_rate, ra

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,stratify=

print(f"X train           shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")
print(f"X X_kaggle_test   shape: {X_kaggle_test.shape}")
```

```
X train           shape: (60989, 182)
X validation      shape: (10763, 182)
```

```
X test            shape: (30752, 182)
X X_kaggle_test   shape: (48744, 182)
```

In [198…

```
def pct(x):
    return round(100*x,3)
```

In [199…

```
del experimentLog
```

In [200…

```
try:
    experimentLog
except NameError:
    experimentLog = pd.DataFrame(columns=["exp_name",
                                          "Train Acc",
                                          "Valid Acc",
                                          "Test  Acc",
                                          "Train AUC",
                                          "Valid AUC",
                                          "Test  AUC",
                                          "Train F1 Score",
                                          "Valid F1 Score",
                                          "Test F1 Score",
                                          "Description"
                                         ])
```

In [201…

```
# roc curve, precision recall curve for each model
fprs, tprs, precisions, recalls, names, scores, cvscores, accuracy, cnfmatrix =
features_list, final_best_clf,results = {}, {},[]
```

In [202…

```
def precision_recall_cust(model,X_train,y_train,X_test, y_test,X_valid, y_valid,
    # plot precision_recall Test
    precision, recall, threshold = precision_recall_curve(y_test,model.predict_p
    precisions.append(precision)
    recalls.append(recall)

    # plot combined Precision Recall curve for train, valid, test
    show_train_precision = plot_precision_recall_curve(model, X_train, y_train,
    show_test_precision = plot_precision_recall_curve(model, X_test, y_test, nam
    show_valid_precision = plot_precision_recall_curve(model, X_valid, y_valid,
    show_valid_precision.ax_.set_title ("Precision Recall Curve Comparison - " +
    plt.legend(bbox_to_anchor=(1.04,1), loc="upper left", borderaxespad=0)
    plt.show()
    return precisions,recalls
```

In [203…

```
def confusion_matrix_def(model,X_train,y_train,X_test, y_test, X_valid, y_valid,
    #Prediction
    preds_test = model.predict(X_test)
    preds_train = model.predict(X_train)
    preds_valid = model.predict(X_valid)

    cm_train = confusion_matrix(y_train, preds_train).astype(np.float32)
    #print(cm_train)
    cm_train /= cm_train.sum(axis=1)[:, np.newaxis]
```

```
    cm_test = confusion_matrix(y_test, preds_test).astype(np.float32)
    #print(cm_test)
    cm_test /= cm_test.sum(axis=1)[:, np.newaxis]

    cm_valid = confusion_matrix(y_valid, preds_valid).astype(np.float32)
    cm_valid /= cm_valid.sum(axis=1)[:, np.newaxis]

    plt.figure(figsize=(16, 4))
    #plt.subplots(1,3,figsize=(12,4))

    plt.subplot(131)
    g = sns.heatmap(cm_train, vmin=0, vmax=1, annot=True, cmap="Reds")
    plt.xlabel("Predicted", fontsize=14)
    plt.ylabel("True", fontsize=14)
    g.set(xticklabels=class_labels, yticklabels=class_labels)
    plt.title("Train", fontsize=14)

    plt.subplot(132)
    g = sns.heatmap(cm_valid, vmin=0, vmax=1, annot=True, cmap="Reds")
    plt.xlabel("Predicted", fontsize=14)
    plt.ylabel("True", fontsize=14)
    g.set(xticklabels=class_labels, yticklabels=class_labels)
    plt.title("Validation set", fontsize=14);

    plt.subplot(133)
    g = sns.heatmap(cm_test, vmin=0, vmax=1, annot=True, cmap="Reds")
    plt.xlabel("Predicted", fontsize=14)
    plt.ylabel("True", fontsize=14)
    g.set(xticklabels=class_labels, yticklabels=class_labels)
    plt.title("Test", fontsize=14);
    cnfmatrix.append(cm_test)

    return cnfmatrix
```

In [204…

```
def roc_curve_cust(model,X_train,y_train,X_test, y_test,X_valid, y_valid,fprs,tp
    fpr, tpr, threshold = roc_curve(y_test, model.predict_proba(X_test)[:, 1])
    fprs.append(fpr)
    tprs.append(tpr)
    # plot combined ROC curve for train, valid, test
    show_train_roc = plot_roc_curve(model, X_train, y_train, name="TrainRocAuc")
    show_test_roc = plot_roc_curve(model, X_test, y_test, name="TestRocAuc", ax=
    show_valid_roc = plot_roc_curve(model, X_valid, y_valid, name="ValidRocAuc",
    show_valid_roc.ax_.set_title ("ROC Curve Comparison - " + name)
    plt.legend(bbox_to_anchor=(1.04,1), loc="upper left", borderaxespad=0)
    plt.show()
    return fprs,tprs
```

In [205…

```
metrics = {'accuracy': make_scorer(accuracy_score),
           'roc_auc': 'roc_auc',
           'f1': make_scorer(f1_score)
          }
```

Define pipeline

In [206…

```
%%time
```

```
np.random.seed(42)
full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("linear", LogisticRegression())
    ])
```

```
CPU times: user 247 µs, sys: 19 µs, total: 266 µs
Wall time: 273 µs
```

Perform cross-fold validation and Train the model

Split the training data to 3 fold to perform Crossfold validation

In [207…

```
cvSplits = ShuffleSplit(test_size=0.3, random_state=0)
```

In [ ]:

```
X_train.head(5)
```

Out[ ]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE
40832	1	117000.0	157500.0	10656.0	157500.0
36820	0	166500.0	900000.0	39775.5	900000.0
81804	1	90000.0	495000.0	23944.5	495000.0
35092	0	112500.0	508495.5	20295.0	454500.0
57197	0	135000.0	400500.0	19606.5	400500.0

5 rows × 183 columns

In [ ]:

```
start = time()
model = full_pipeline_with_predictor.fit(X_train, y_train)
np.random.seed(42)

# Set up cross validation scores
logit_scores = cross_validate(model, X_train, y_train,cv=3,scoring=metrics, retu
train_time = np.round(time() - start, 4)

# Time and score valid predictions
start = time()
logit_score_valid  = full_pipeline_with_predictor.score(X_valid, y_valid)
valid_time = np.round(time() - start, 4)

# Time and score test predictions
start = time()
logit_score_test  = full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time() - start, 4)
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:76
```

```
3: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:76
3: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:76
3: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n
  n_iter_i = _check_optimize_result(
```

Calculate Metrics

In [ ]:

```
exp_name = f"Baseline_{len(selected_features)}_features"
experimentLog.loc[len(experimentLog)] = [f"{exp_name}"] + list(np.round(
               [logit_scores['train_accuracy'].mean(),
                logit_scores['test_accuracy'].mean(),
                accuracy_score(y_test, model.predict(X_test)),
                logit_scores['train_roc_auc'].mean(),
                logit_scores['test_roc_auc'].mean(),
                roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]),
                logit_scores['train_f1'].mean(),
                logit_scores['test_f1'].mean(),
                f1_score(y_test, model.predict(X_test)) ],4)) \
                +  [f"Imbalanced Logistic reg features {tot_features} with 20% t
experimentLog
```

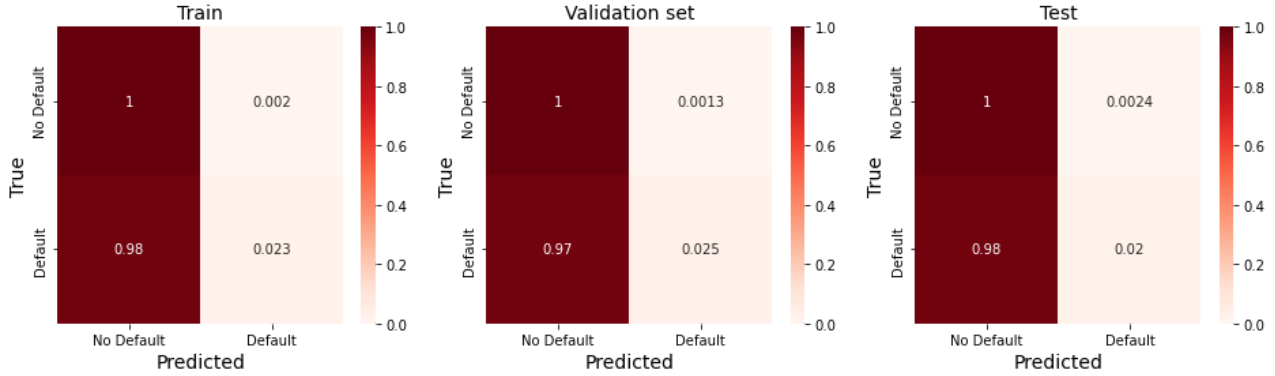
Out[ ]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score	Valid F1 Score	Test F1 Score	De
0	Baseline_183_features	0.919	0.9184	0.9183	0.7695	0.7544	0.7529	0.0478	0.0449	0.039	Im Lo

Confusion matrix

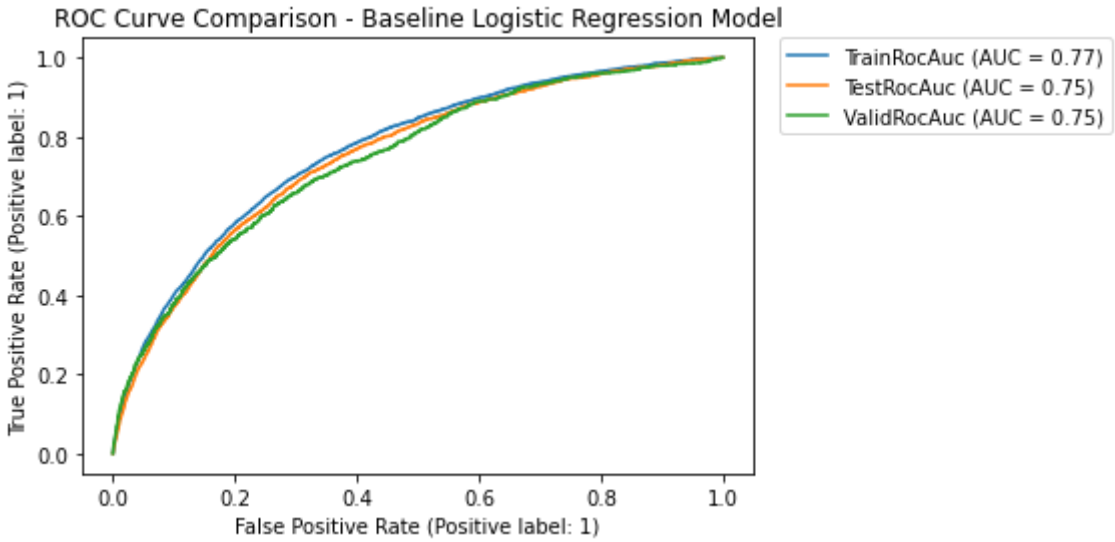
In [ ]:
```
# Create confusion matrix for baseline model
confusion_matrix_def(model,X_train,y_train,X_test,y_test,X_valid, y_valid,cnfmat
```

Out[ ]:
```
[array([[0.9975934 , 0.00240657],
        [0.9795673 , 0.02043269]], dtype=float32)]
```



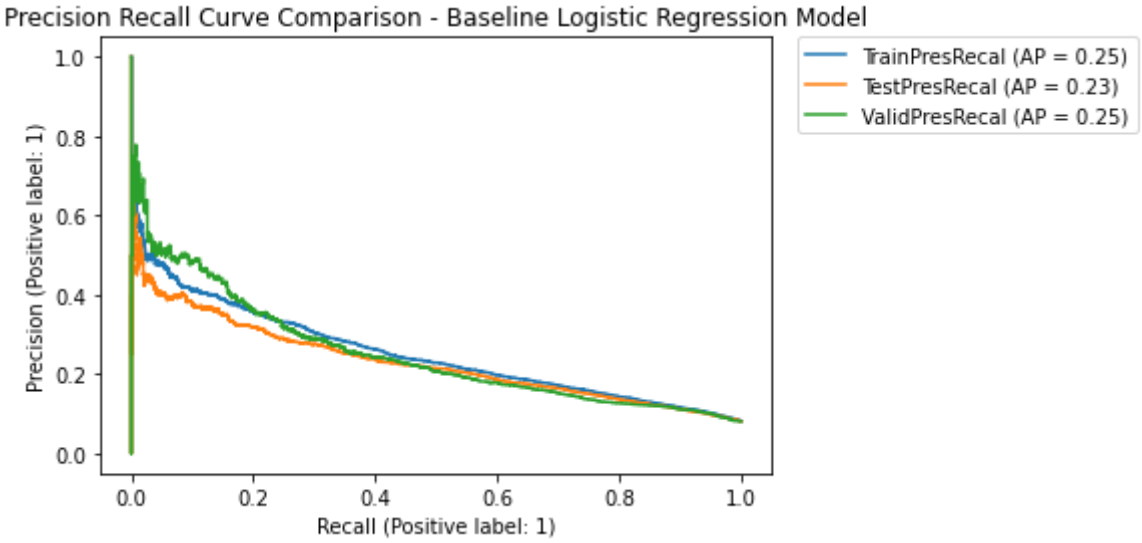
AUC (Area under ROC curve)

In [ ]:
```
_,_=roc_curve_cust(model,X_train,y_train,X_test, y_test,X_valid, y_valid,fprs,tp
```



Precision Recall Curve

In [ ]:
```
_,_=precision_recall_cust(model,X_train,y_train,X_test, y_test,X_valid, y_valid,
```



Baseline Model - With sampled data

To get a baseline, we will use some of the features after being preprocessed through the pipeline. The baseline model is a logistic regression model. Since 'No default and Default' target records are not balanced in trainging set, we are going to resample the minority class("Default with target value 1") to balance the input dataset

In [ ]:
```
print(f"X train           shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")
print(f"X X_kaggle_test   shape: {X_kaggle_test.shape}")
```

```
X train           shape: (60989, 183)
X validation      shape: (10763, 183)
X test            shape: (30752, 183)
X X_kaggle_test   shape: (48744, 183)
```

In [ ]:
```
# Bincount shows the imbalanced data in Target default and no default class
np.bincount(y_train)
```

Out[ ]:
```
array([56038,  4951])
```

Resample Minority class

Resampling should be performed only in the train dataset, to avoid overfitting and data leakage.

In [ ]:
```
# concatenate our training data back together
train_data = pd.concat([X_train, y_train], axis=1)
train_data.head()
```

Out[ ]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE
40832	1	117000.0	157500.0	10656.0	157500.0
36820	0	166500.0	900000.0	39775.5	900000.0

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE
81804	1	90000.0	495000.0	23944.5	495000.0
35092	0	112500.0	508495.5	20295.0	454500.0
57197	0	135000.0	400500.0	19606.5	400500.0

5 rows × 184 columns

After resampling, both default and non-default classes are balanced

In [ ]:
```
# separate minority and majority classes
no_default_data = train_data[train_data.TARGET==0]
default_data = train_data[train_data.TARGET==1]

# sample minority
default_sampled_data = resample(default_data,
                           replace=True, # sample with replacement
                           n_samples=len(no_default_data), # match number in majo
                           random_state=42) # reproducible

# combine majority and upsampled minority
train_data = pd.concat([no_default_data, default_sampled_data])

train_data.TARGET.value_counts()
```

Out[ ]:
```
0    56038
1    56038
Name: TARGET, dtype: int64
```

In [ ]:
```
y_train = train_data['TARGET']
X_train = train_data[selected_features]
```

Create a Pipeline with Baseline Model

In [ ]:
```
%%time
full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("linear", LogisticRegression())
    ])
```

```
CPU times: user 337 µs, sys: 123 µs, total: 460 µs
Wall time: 484 µs
```

Create crossfold validation splits

Split the training data to 5 fold to perform Crossfold validation

In [ ]:
```
cvSplits = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
```

Baseline Prediction

In [ ]:

```
model = full_pipeline_with_predictor.fit(X_train, y_train)
np.random.seed(42)

# Set up cross validation scores
logit_scores = cross_validate(model, X_train, y_train,cv=3,scoring=metrics, retu
train_time = np.round(time() - start, 4)

# Time and score test predictions
logit_score_valid  = full_pipeline_with_predictor.score(X_valid, y_valid)
valid_time = np.round(time() - start, 4)

# Time and score test predictions
logit_score_test  = full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time() - start, 4)
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:76
3: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:76
3: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:76
3: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n
  n_iter_i = _check_optimize_result(
```

Baseline metrics

Accuracy, AUC score, F1 Score and Log loss used for measuring the baseline model

In [ ]:

```
exp_name = f"Baseline_{len(selected_features)}_features"
experimentLog.loc[len(experimentLog)] = [f"{exp_name}"] + list(np.round(
               [logit_scores['train_accuracy'].mean(),
                logit_scores['test_accuracy'].mean(),
                accuracy_score(y_test, model.predict(X_test)),
                logit_scores['train_roc_auc'].mean(),
                logit_scores['test_roc_auc'].mean(),
                roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]),
```

```
                logit_scores['train_f1'].mean(),
                logit_scores['test_f1'].mean(),
                f1_score(y_test, model.predict(X_test)) ],4)) \
                + [f"Balanced Logistic reg features {tot_features} with 20% trai
experimentLog
```

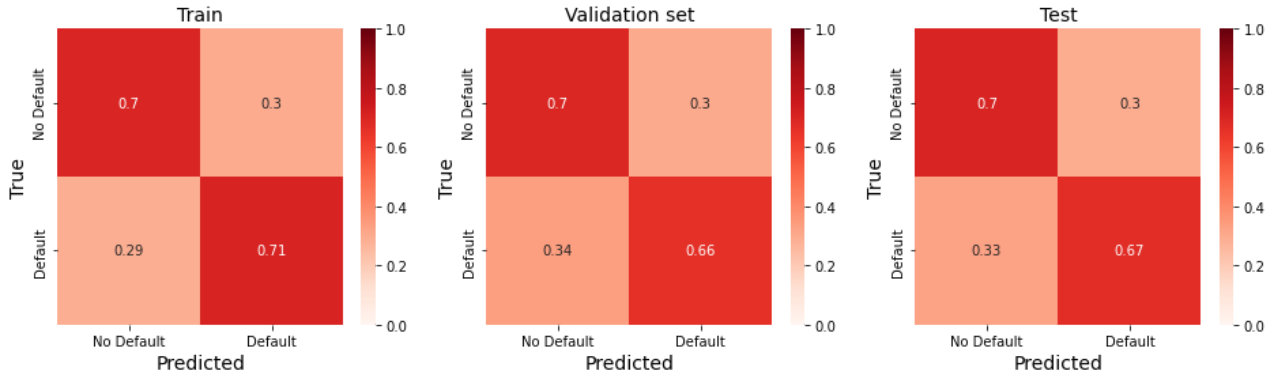
Out[ ]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score	Valid F1 Score	Test F1 Score
0	Baseline_183_features	0.9190	0.9184	0.9183	0.7695	0.7544	0.7529	0.0478	0.0449	0.0390
1	Baseline_183_features	0.7032	0.7002	0.7022	0.7703	0.7672	0.7511	0.7038	0.7007	0.2683

Confusion matrix

In [ ]:

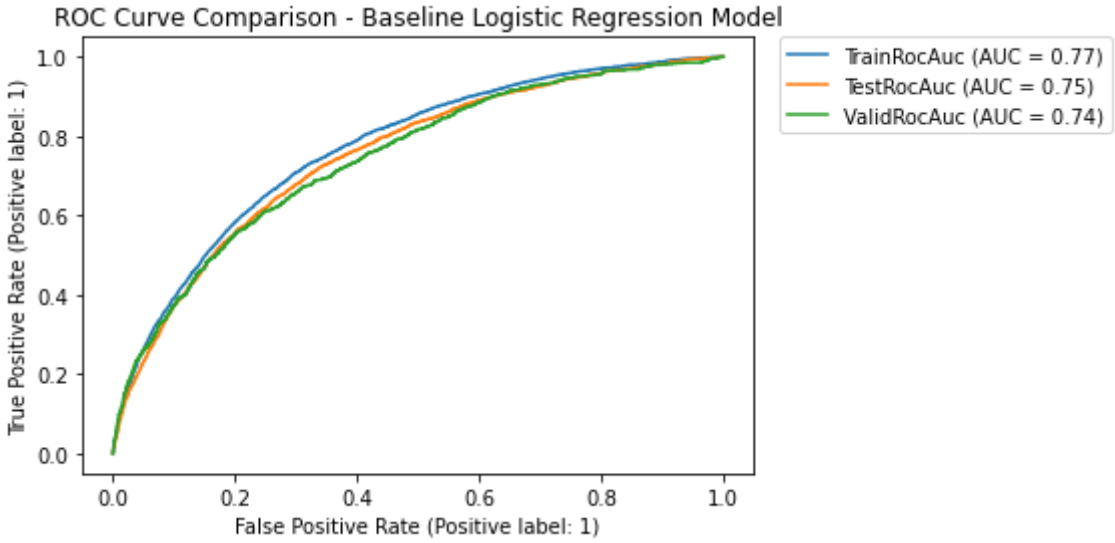
```
# Create confusion matrix for baseline model
_=confusion_matrix_def(model,X_train,y_train,X_test,y_test,X_valid, y_valid,cnfm
```



AUC (Area under ROC curve)

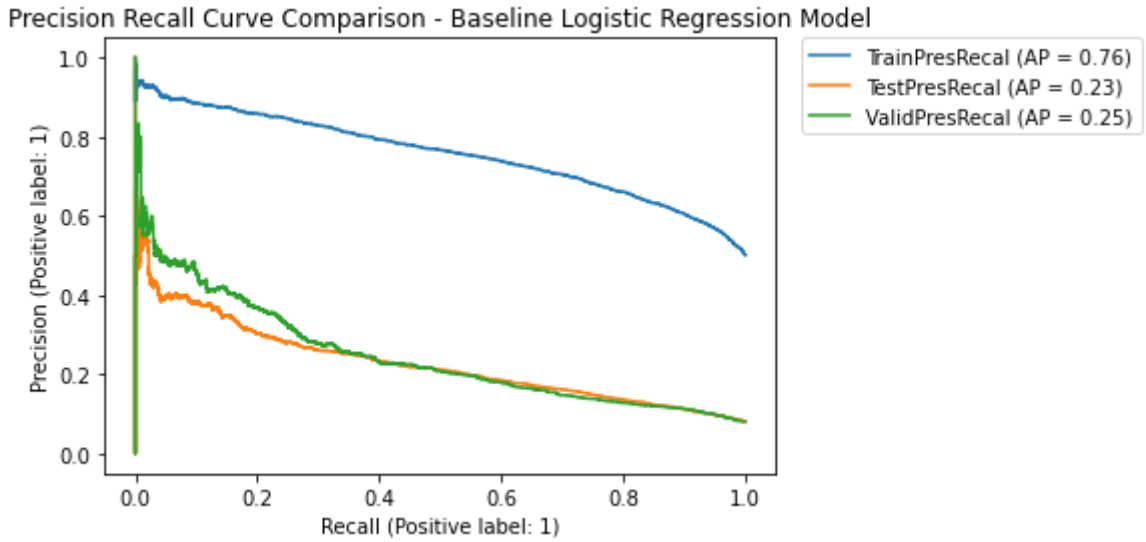
In [ ]:

```
_,_=roc_curve_cust(model,X_train,y_train,X_test, y_test,X_valid, y_valid,fprs,tp
```



Precision Recall Curve

In [ ]:
```
_,_=precision_recall_cust(model,X_train,y_train,X_test, y_test,X_valid, y_valid,
```



Tune Basline model with grid search

Various Classification algorithms were used to compare with the best model. Following metrics were used to find the best model

- Cross fold Train Accuracy
- Test Accuracy
- Train ROC_AUC_Score
- Test ROC_AUC_Score
- Train F1_Score
- Test F1_Score
- Confusion matrix

Classifiers

In [ ]:
```
classifiers = [
        [('Logistic Regression', LogisticRegression(solver='saga',random_state=4
        [('Support Vector', SVC(random_state=42,probability=True))],
        [('XGBoost', XGBClassifier(random_state=42))],
        [('RandomForest', RandomForestClassifier(random_state=42))]
    ]
```

Hyper-parameters for all models specified above

In [ ]:
```
# Arrange grid search parameters for each classifier
params_grid = {
        'Logistic Regression': {
            'penalty': ('l1', 'l2'),
            'tol': (0.0001, 0.00001),
            'C': (10, 1, 0.1, 0.01),
        }
    ,
        'Support Vector' : {
            'kernel': ('rbf','poly'),
            'degree': (4, 5),
            'C': ( 0.0001, 0.001),   #Low C - allow for misclassification
            'gamma':(0.01,0.1,1)  #Low gamma - high variance and low bias
        }
    ,
        'XGBoost':  {
            'max_depth': [3,5], # Lower helps with overfitting
            'n_estimators':[200,300],
            'learning_rate': [0.01,0.1],
            'colsample_bytree' : [0.2],
        },                      #small numbers reduces accuracy but runs faster

        'RandomForest':  {
            'max_depth': [5,10],
            'max_features': [15,20],
            'min_samples_split': [5, 10],
            'min_samples_leaf': [3, 5],
            'bootstrap': [True],
            'n_estimators':[100]},
    }
```

In [ ]:
```
# Set feature selection settings
# Features removed each step
feature_selection_steps=10
# Number of features used
features_used=len(selected_features)
```

In [ ]:
```
results.append(logit_scores['train_accuracy'])
names = ['Baseline LR']
def ConductGridSearch(in_classifiers,cnfmatrix,fprs,tprs,precisions,recalls,cvSp
    for (name, classifier) in in_classifiers:
        # Print classifier and parameters
        print('****** START', name,'*****')
        parameters = params_grid[name]
        print("Parameters:")
```

```
for p in sorted(parameters.keys()):
    print("\t"+str(p)+": "+ str(parameters[p]))

full_pipeline_with_predictor = Pipeline([
("preparation", data_prep_pipeline),
("predictor", classifier)
])

# Execute the grid search
params = {}
for p in parameters.keys():
    pipe_key = 'predictor__'+str(p)
    params[pipe_key] = parameters[p]
grid_search = GridSearchCV(full_pipeline_with_predictor, params, cv=cvSp
                           n_jobs=-1,verbose=1)
grid_search.fit(X_train, y_train)
# Best estimator score
best_train = pct(grid_search.best_score_)

# Best train scores
print("Cross validation with best estimator")
best_train_scores = cross_validate(grid_search.best_estimator_, X_train,
                                   return_train_score=True, n_jobs=-1)

#get all scores
best_train_accuracy = np.round(best_train_scores['train_accuracy'].mean(
best_train_f1 = np.round(best_train_scores['train_f1'].mean(),4)
best_train_roc_auc = np.round(best_train_scores['train_roc_auc'].mean(),

valid_time = np.round(best_train_scores['score_time'].mean(),4)
best_valid_accuracy = np.round(best_train_scores['test_accuracy'].mean()
best_valid_f1 = np.round(best_train_scores['test_f1'].mean(),4)
best_valid_roc_auc = np.round(best_train_scores['test_roc_auc'].mean(),4

#append all results
results.append(best_train_scores['train_accuracy'])
names.append(name)

#test and Prediction with whole data
# Best estimator fitting time
print("Fit and Prediction with best estimator")
start = time()
model = grid_search.best_estimator_.fit(X_train, y_train)
train_time = round(time() - start, 4)

# Best estimator prediction time
start = time()
y_test_pred = model.predict(X_test)
test_time = round(time() - start, 4)
scores.append(roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]))
accuracy.append(accuracy_score(y_test, y_test_pred))

# Create confusion matrix for the best model
cnfmatrix = confusion_matrix_def(model,X_train,y_train,X_test,y_test,X_v

# Create AUC ROC curve
fprs,tprs = roc_curve_cust(model,X_train,y_train,X_test, y_test,X_valid,

#Create Precision recall curve
precisions,recalls = precision_recall_cust(model,X_train,y_train,X_test,
```

```
        #Best Model
        final_best_clf[name]=pd.DataFrame([{'label': grid_search.best_estimator_
                                 'predictor': grid_search.best_estimator_.
        #Feature importance
        feature_name = num_attribs + list(grid_search.best_estimator_.named_step
        feature_list = feature_name
        #features_list[name]=pd.DataFrame({'feature_name': feature_list,
        #                              'feature_importance': grid_search.best_est
        # Collect the best parameters found by the grid search
        print("Best Parameters:")
        best_parameters = grid_search.best_estimator_.get_params()
        param_dump = []
        for param_name in sorted(params.keys()):
            param_dump.append((param_name, best_parameters[param_name]))
            print("\t"+str(param_name)+": " + str(best_parameters[param_name]))
        print("****** FINISH",name," *****")
        print("")

        # Record the results
        exp_name = name
        experimentLog.loc[len(experimentLog)] = [f"{exp_name}"] + list(np.round(
                [best_train_accuracy,
                best_valid_accuracy,
                accuracy_score(y_test, y_test_pred),
                best_train_roc_auc,
                best_valid_roc_auc,
                roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]),
                best_train_f1,
                best_valid_f1,
                f1_score(y_test, y_test_pred)
                ],4)) \
                + [json.dumps(param_dump)]
```

Logistic Regression Model

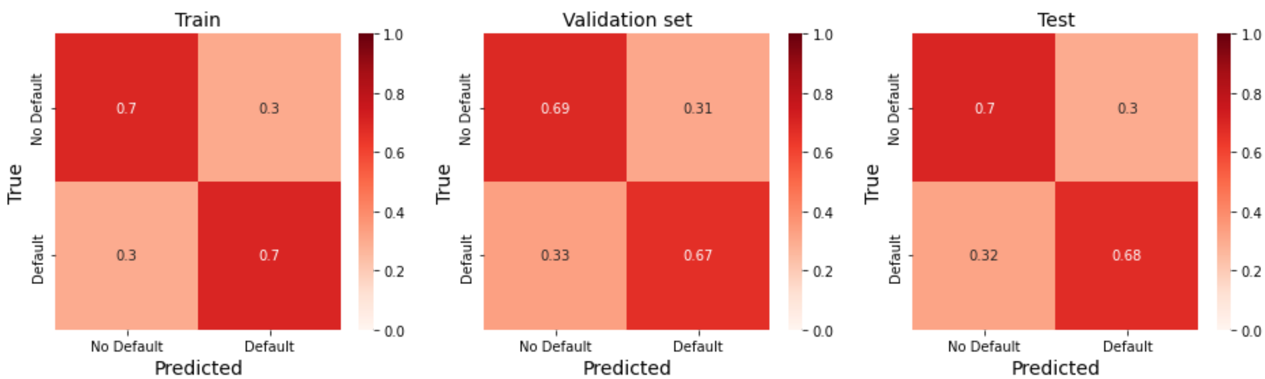
In [ ]:

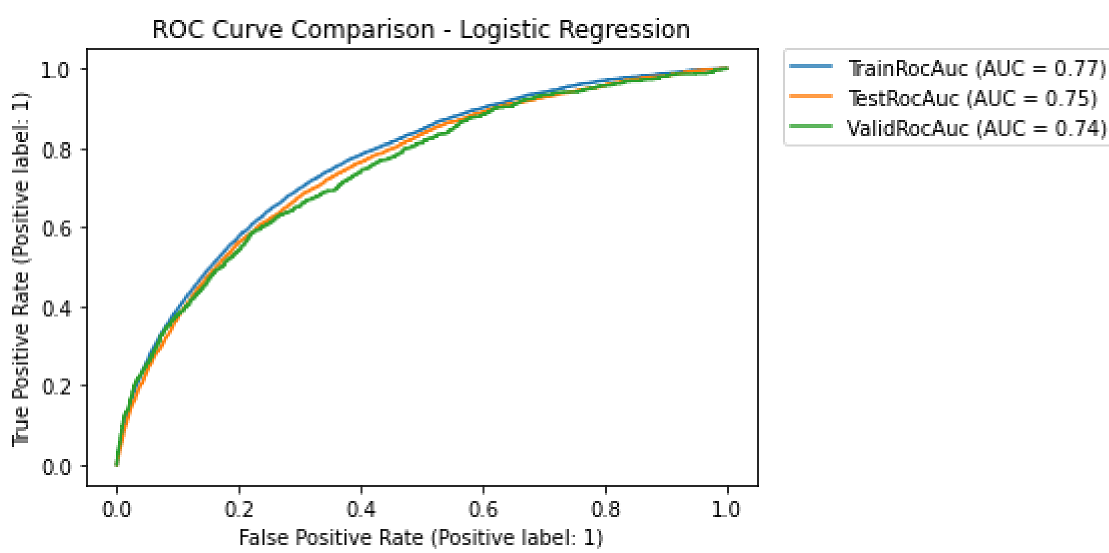
```
ConductGridSearch(classifiers[0],cnfmatrix,fprs,tprs,precisions,recalls,cvSplits
```

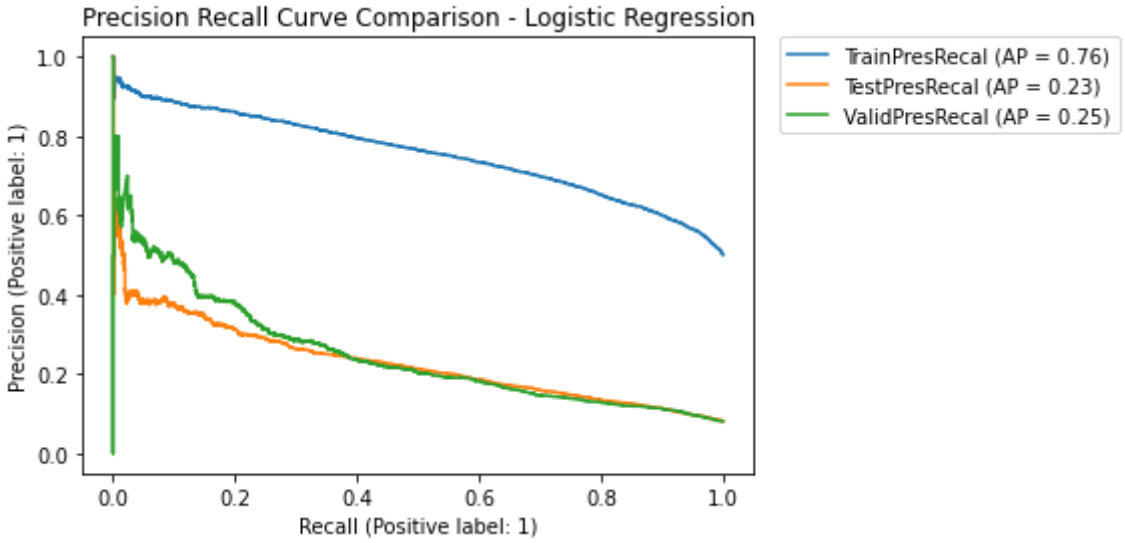
```
****** START Logistic Regression *****
Parameters:
        C: (10, 1, 0.1, 0.01)
        penalty: ('l1', 'l2')
        tol: (0.0001, 1e-05)
Fitting 2 folds for each of 16 candidates, totalling 32 fits
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
```

```
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
Cross validation with best estimator
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:328: Con
vergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
Fit and Prediction with best estimator
```





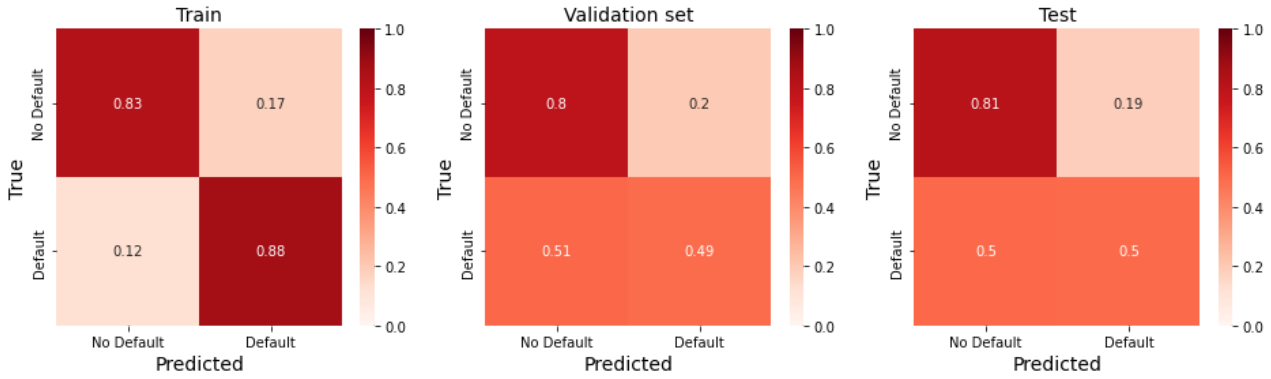


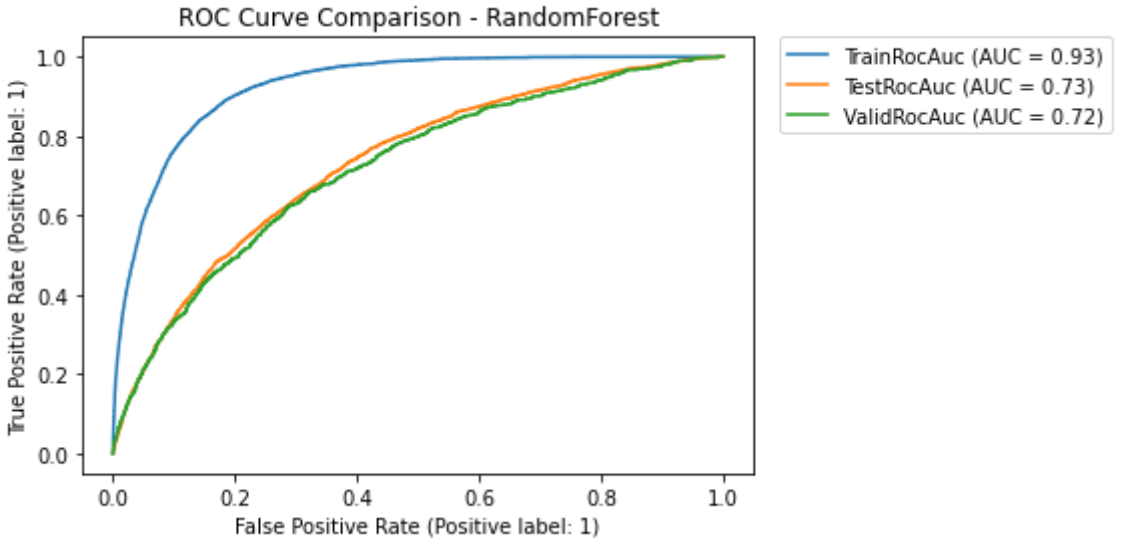

```
Best Parameters:
        predictor__C: 10
        predictor__penalty: 12
        predictor__tol: 0.0001
****** FINISH Logistic Regression  *****
```

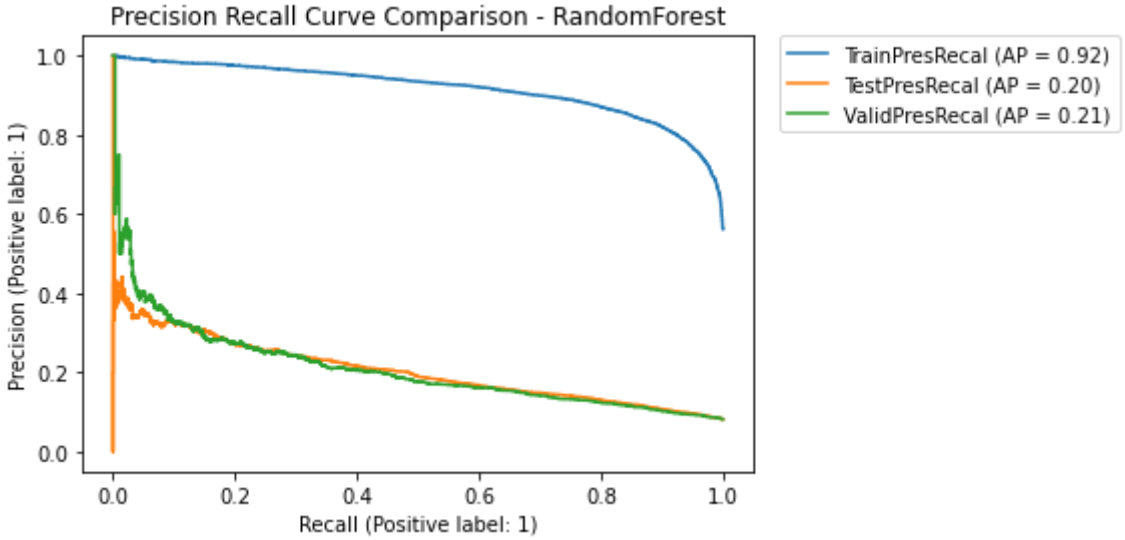
RandomForest

In [ ]:
```
ConductGridSearch(classifiers[3],cnfmatrix,fprs,tprs,precisions,recalls,cvSplits
```

```
****** START RandomForest *****
Parameters:
        bootstrap: [True]
        max_depth: [5, 10]
        max_features: [15, 20]
        min_samples_leaf: [3, 5]
        min_samples_split: [5, 10]
        n_estimators: [100]
Fitting 2 folds for each of 16 candidates, totalling 32 fits
Cross validation with best estimator
Fit and Prediction with best estimator
```








```
Best Parameters:
        predictor__bootstrap: True
        predictor__max_depth: 10
        predictor__max_features: 20
        predictor__min_samples_leaf: 3
        predictor__min_samples_split: 5
        predictor__n_estimators: 100
****** FINISH RandomForest  *****
```

SVM(Support Vector Machines)

It was taking infinite amount of time to execute.

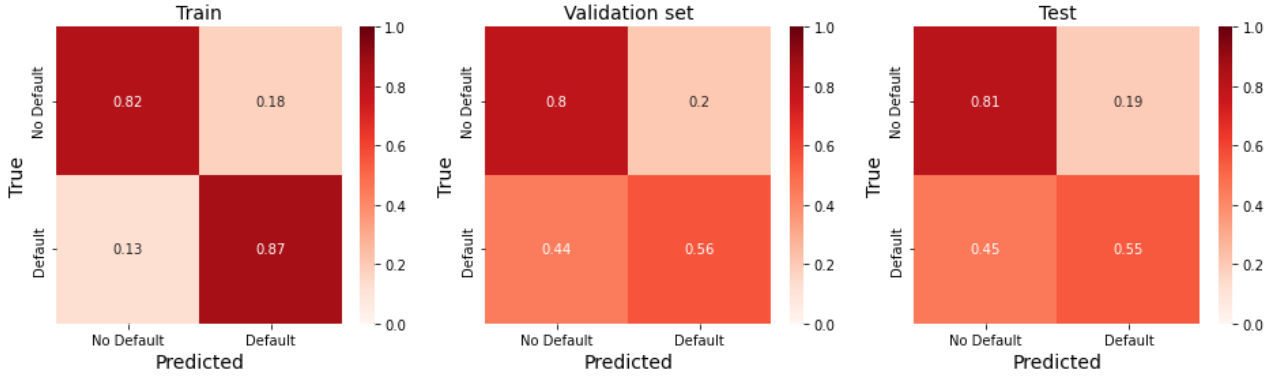
In [ ]:
```
ConductGridSearch(classifiers[1],cnfmatrix,fprs,tprs,precisions,recalls,cvSplits
```

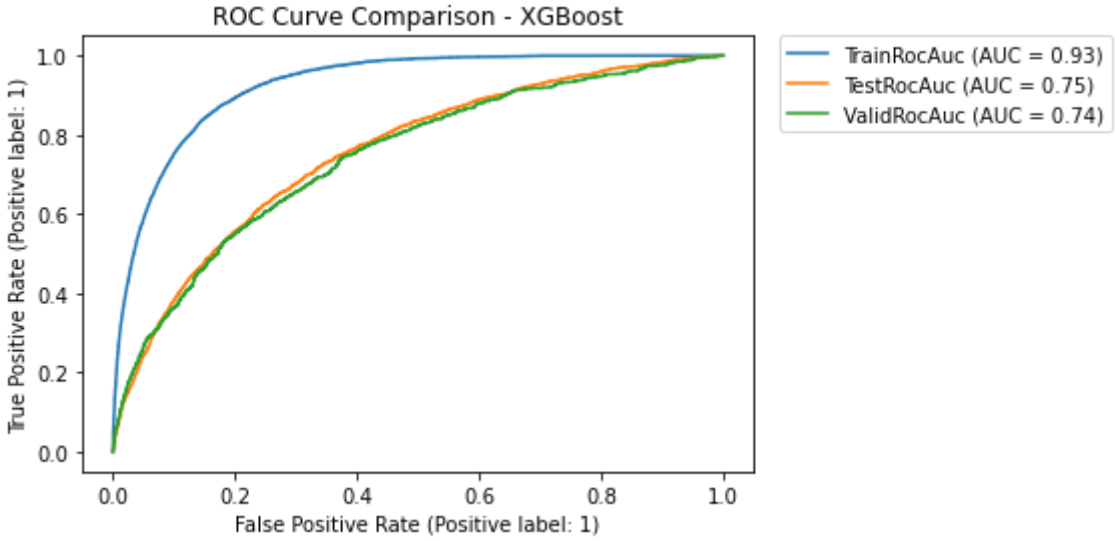
XGBoost

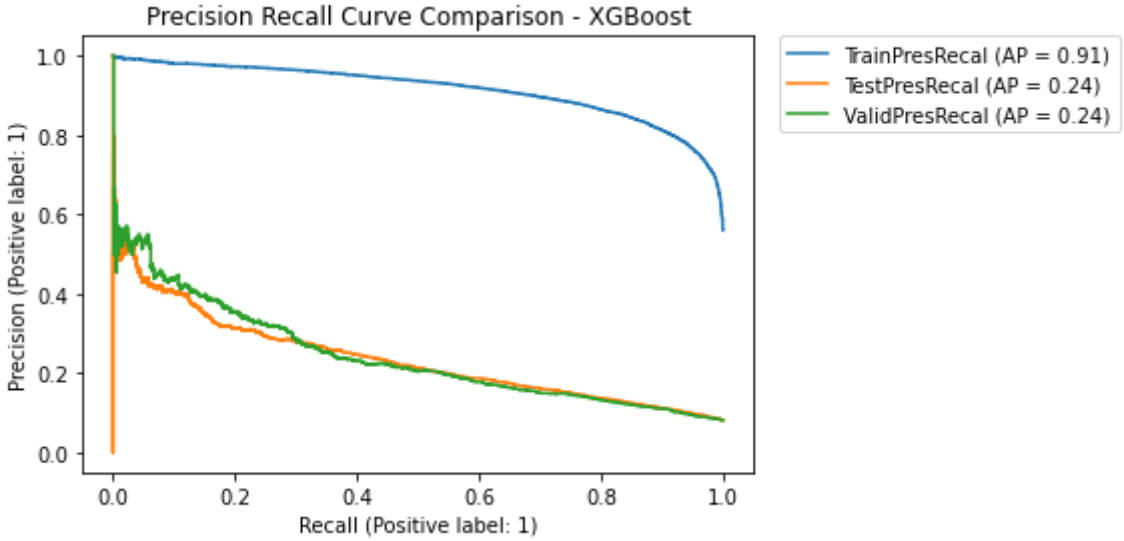
In [ ]:
```
ConductGridSearch(classifiers[2],cnfmatrix,fprs,tprs,precisions,recalls,cvSplits
```

```
****** START XGBoost *****
Parameters:
```

```
        colsample_bytree: [0.2]
        learning_rate: [0.01, 0.1]
        max_depth: [3, 5]
        n_estimators: [200, 300]
Fitting 2 folds for each of 8 candidates, totalling 16 fits
Cross validation with best estimator
Fit and Prediction with best estimator
```





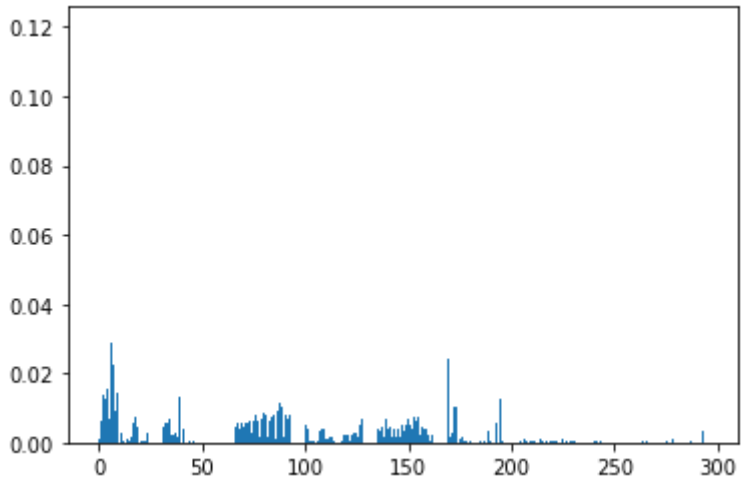



```
Best Parameters:
        predictor__colsample_bytree: 0.2
        predictor__learning_rate: 0.1
        predictor__max_depth: 5
        predictor__n_estimators: 300
****** FINISH XGBoost  *****
```

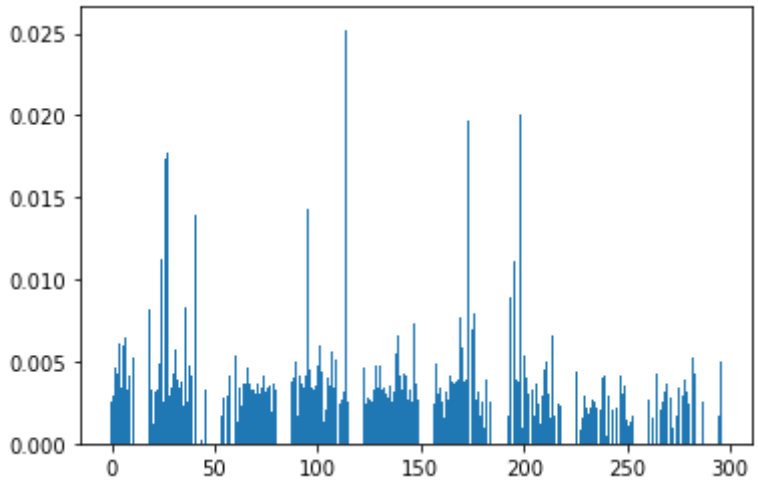
Model Validation

Feature Importance based on all Models

In [ ]:
```
# define the model
classifier = RandomForestClassifier(random_state=42,max_depth= 10,max_features=2
                                    min_samples_leaf=3,bootstrap=True,n_estimators=10
# fit the model
full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("predictor", classifier)
        ])
full_pipeline_with_predictor.fit(X_train, y_train)
importance=full_pipeline_with_predictor.steps[1][1].feature_importances_
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```

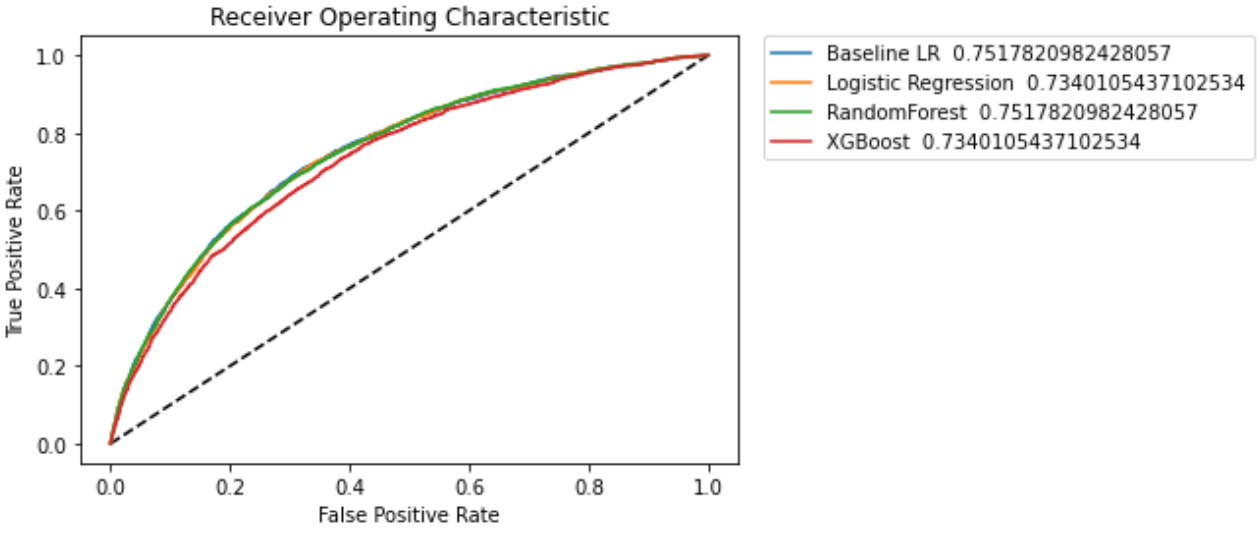


In [ ]:
```
# define the model
classifier = XGBClassifier(random_state=42,colsample_bytree= 0.2,learning_rate=
# fit the model
full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("predictor", classifier)
        ])
full_pipeline_with_predictor.fit(X_train, y_train)
importance=full_pipeline_with_predictor.steps[1][1].feature_importances_
# get importance
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```



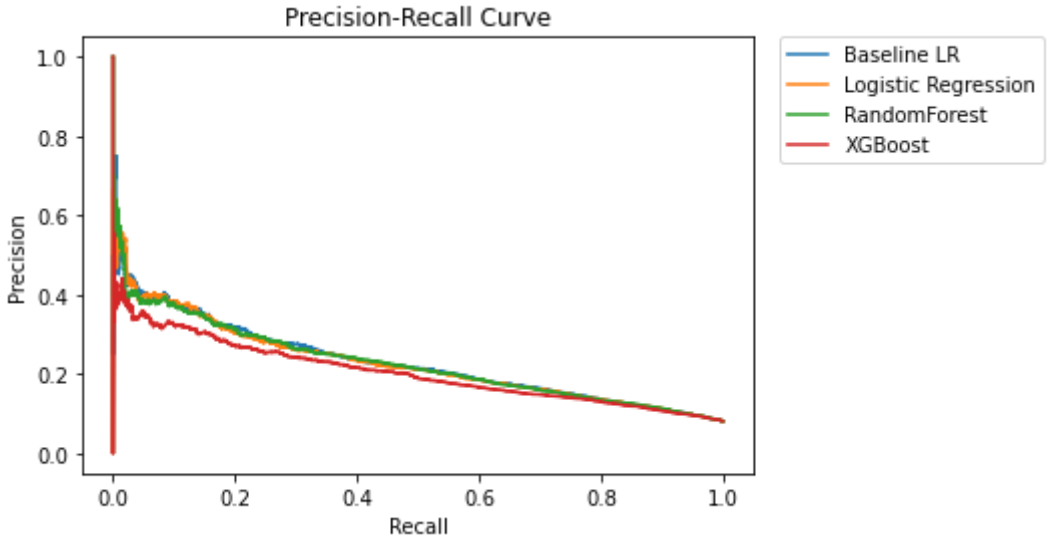
AUC(Area Under ROC Curve)

In [ ]:
```
# roc curve fpr, tpr  for all classifiers
plt.plot([0,1],[0,1], 'k--')
for i in range(len(names)):
    plt.plot(fprs[i],tprs[i],label = names[i] + '   ' + str(scores[i]))
plt.legend(bbox_to_anchor=(1.04,1), loc="upper left", borderaxespad=0)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title('Receiver Operating Characteristic')
plt.show()
```



Precision Recall Curve

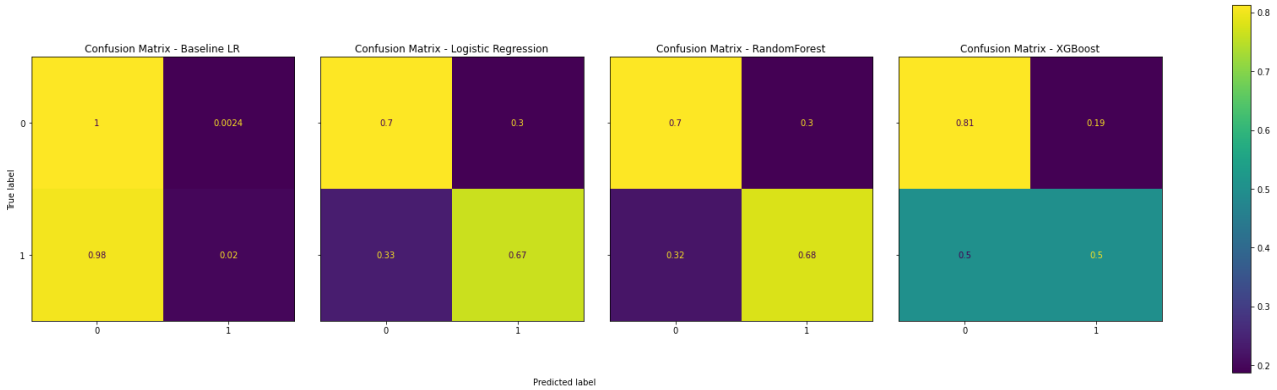
In [ ]:
```
# precision recall curve  for all classifiers
for i in range(len(names)):
    plt.plot(recalls[i],precisions[i],label = names[i])
plt.legend(bbox_to_anchor=(1.04,1), loc="upper left", borderaxespad=0)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title('Precision-Recall Curve')
plt.show()
```



Confusion Matrix

In [ ]:

```
# plot confusion matrix for all classifiers
f, axes = plt.subplots(1, len(names), figsize=(30, 8), sharey='row')
for i in range(len(names)):
    disp = ConfusionMatrixDisplay(cnfmatrix[i], display_labels=['0', '1'])
    disp.plot(ax=axes[i], xticks_rotation=0)
    disp.ax_.set_title("Confusion Matrix - " + names[i])
    disp.im_.colorbar.remove()
    disp.ax_.set_xlabel('')
    if i!=0:
        disp.ax_.set_ylabel('')

f.text(0.4, 0.1, 'Predicted label', ha='left')
plt.subplots_adjust(wspace=0.10, hspace=0.1)

f.colorbar(disp.im_, ax=axes)
plt.show()
```



Final Results

In [ ]:

```
pd.set_option('display.max_colwidth', None)
experimentLog
```

Out[ ]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score	Valid F1 Score	Test F1 Score
0	Baseline_183_features	0.9190	0.9184	0.9183	0.7695	0.7544	0.7529	0.0478	0.0449	0.0390
1	Baseline_183_features	0.7032	0.7002	0.7022	0.7703	0.7672	0.7511	0.7038	0.7007	0.2683
2	Logistic Regression	0.7005	0.6980	0.6984	0.7673	0.7647	0.7518	0.7011	0.6984	0.2672
3	RandomForest	0.8590	0.8240	0.7877	0.9340	0.9034	0.7340	0.8618	0.8282	0.2758
4	Logistic Regression	0.7005	0.6980	0.6984	0.7673	0.7647	0.7518	0.7011	0.6984	0.2672
5	RandomForest	0.8590	0.8240	0.7877	0.9340	0.9034	0.7340	0.8618	0.8282	0.2758
6	XGBoost	0.8600	0.8229	0.7865	0.9360	0.9026	0.7537	0.8626	0.8271	0.2931

Kaggle submission

Best Pipeline for submission

In [ ]:
```
%%time
np.random.seed(42)
model_selection = ['Logistic Regression','XGBoost','RandomForest']
print("Classifier with parameters")
final_estimators = []
for i,clf in enumerate(model_selection):
    model = final_best_clf[clf]['predictor'][0]
    print(i+1, " :",model)
    final_estimators.append((clf,make_pipeline(data_prep_pipeline,
```

```
                        RFE(estimator=model,n_features_to_select=features_used,
                        model)))
```

```
Classifier with parameters
1   : LogisticRegression(C=10, random_state=42, solver='saga')
2   : XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.2,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.1, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=5, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=300,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=42,
              reg_alpha=0, reg_lambda=1, ...)
3   : RandomForestClassifier(max_depth=10, max_features=20, min_samples_leaf=3,
                         min_samples_split=5, random_state=42)
CPU times: user 65.9 ms, sys: 1.88 ms, total: 67.8 ms
Wall time: 66.1 ms
```

Voting Classifier to predict best results based on best Classifier Probability

In [ ]:
```
voting_classifier = Pipeline([("clf", VotingClassifier(estimators=final_estimato

final_X_train = finaldf[0][selected_features]
final_y_train = finaldf[0]['TARGET']
voting_classifier.fit(final_X_train, final_y_train)
start = time()
train_time = round(time() - start, 4)
print("Voting Score:{0}".format(voting_classifier.score(final_X_train, final_y_t
```

```
Voting Score:0.9202275033169437
```

Submission File Prep

For each SK_ID_CURR in the test set, you must predict a probability for the TARGET variable. The file should contain a header and have the following format:

```
SK_ID_CURR,TARGET
100001,0.1
100005,0.9
100013,0.2
etc.
```

In [ ]:
```
test_class_scores = voting_classifier.predict_proba(X_kaggle_test)[:, 1]
```

In [ ]:
```
test_class_scores[0:10]
```

Out[ ]:
```
array([0.06906573, 0.18275018, 0.02448831, 0.03824977, 0.14961267,
       0.03413199, 0.03290953, 0.06676515, 0.02625002, 0.0886295 ])
```

In [ ]:
```
# Submission dataframe
submit_df = datasets["application_test"][['SK_ID_CURR']]
submit_df['TARGET'] = test_class_scores
```

```
submit_df.head()
```

Out[ ]:

	SK_ID_CURR	TARGET
0	100001	0.069066
1	100005	0.182750
2	100013	0.024488
3	100028	0.038250
4	100038	0.149613

In [ ]:
```
submit_df.to_csv("submission.csv",index=False)
```

XGBoosT best pipeline for submission

In [ ]:
```
model = final_best_clf[model_selection[2]]['predictor'][0]
XG_Pipeline = Pipeline([
                ("preparation", data_prep_pipeline),
                ('RFE', RFE(estimator=model,n_features_to_select=features_used,
                ('XGB', model)])
```

In [ ]:
```
XG_Pipeline.fit(final_X_train, final_y_train)
class_scores = XG_Pipeline.predict_proba(X_kaggle_test)[:, 1]
class_scores[0:10]
```

Out[ ]:
```
array([0.091756  , 0.11039003, 0.03413013, 0.06663625, 0.13032818,
       0.05869292, 0.04606297, 0.06919562, 0.03744675, 0.11461059])
```

In [ ]:
```
# Submission dataframe
submit_df_1 = datasets["application_test"][['SK_ID_CURR']]
submit_df_1['TARGET'] = class_scores

submit_df_1.head()
```

Out[ ]:

	SK_ID_CURR	TARGET
0	100001	0.091756
1	100005	0.110390
2	100013	0.034130
3	100028	0.066636
4	100038	0.130328

In [ ]:
```
submit_df_1.to_csv("submission1.csv",index=False)
```

Deep Learning

Deep Learning Model Pipeline & Workflow

Deep learning is a sub field of machine learning. Deep learning is about learning from past data using artificial neural networks with multiple hidden layers (2 or more hidden layers). Deep neural networks uncrumple complex representation of data step-by-step, layer-by-layer (hence multiple hidden layers) into a clear representation of the data. Artificial neural networks having one hidden layer apart from input and output layer is called as multi-layer perceptron (MLP) network.

Deep Learning Pipeline Model workflow

Imports

In [397…

```
import torch
import tensorflow as tf
import torch.nn as nn
import torch.nn.functional as func
import torch.optim as optim
from torch.utils.data import DataLoader

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
#from keras.layers.normalization import BatchNormalization

import copy
from datetime import datetime
import pickle
import time
import torch
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as func
import torch.optim as optim
from torch.optim import lr_scheduler

# Metrics
from sklearn.metrics import auc
from tensorflow.keras.callbacks import TensorBoard
from torch.utils.tensorboard import SummaryWriter
# writer logs the events to runs/ directory
writer = SummaryWriter("runs/")
```

Single layer Neural Network

Data Preparation Transform data using data pipeline and converted into Tensor for neural network pipeline.

In [398…

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

```
cuda:0
```

In [399…
```
full_X_train = data_prep_pipeline.fit_transform(X_train)
full_X_test = data_prep_pipeline.transform(X_test)

full_X_train_gpu = torch.FloatTensor(full_X_train).cuda()
full_X_test_gpu = torch.FloatTensor(full_X_test).cuda()

y_train_gpu =  torch.FloatTensor(y_train.to_numpy()).cuda()
y_test_gpu = torch.FloatTensor(y_test.to_numpy()).cuda()
```

In [400…
```
full_X_test_gpu.shape,full_X_train_gpu.shape
```

Out[400… `(torch.Size([92254, 295]), torch.Size([182968, 295]))`

In [401…
```
results = pd.DataFrame(columns=["ExpID",
               "Train Acc", "Val Acc", "Test Acc", "p-value",
               "Train AUC", "Val AUC", "Test AUC",
               "Train f1", "Val f1", "Test f1",
               "Train logloss", "Val logloss", "Test logloss",
               "Train Time(s)", "Val Time(s)", "Test Time(s)",
               "Experiment description",
               "Top 10 Features"])
```

One layer : Linear and Sigmoid Activate Function

Signmoid layer is used to create the probability of prediction.

In [402…
```
D_in = full_X_train_gpu.shape[1]
D_hidden1 = 20
D_hidden2 = 10
D_out= 1
model1 = torch.nn.Sequential(
    torch.nn.Linear(D_in, D_out),
    nn.Sigmoid())
```

In [403…
```
learning_rate = 0.01
optimizer = torch.optim.Adam(model1.parameters(), lr=learning_rate)
model1 = model1.cuda()
```

In [404…
```
def return_report(y, y_prob):
  _, y_pred = torch.max(y_prob, dim = 1)
  y_pred = y_pred.cpu().numpy()
  acc = accuracy_score(y, y_pred)
  roc_auc = roc_auc_score(y, y_prob.cpu().detach().numpy())

  return_list = ([round(acc,4), round(roc_auc, 4)])

  return return_list
```

In [405…

```
def print_report(y, y_prob):
  _, y_pred = torch.max(y_prob, dim = 1)
  y_pred = y_pred.cpu().numpy()
  acc = accuracy_score(y, y_pred)
  roc_auc = roc_auc_score(y, y_prob.cpu().detach().numpy())

  print(f'Accuracy : {round(acc,4)} ; ROC_AUC : {round(roc_auc, 4)}')
```

Train Neural Network

In [406…

```
epochs = 500
y_train_gpu = y_train_gpu.reshape(-1, 1)
print('Train data : ')
model1.train()
writer = SummaryWriter()

for i in range(epochs):


  y_train_pred_prob = model1(full_X_train_gpu)

  loss = func.binary_cross_entropy(y_train_pred_prob, y_train_gpu)
  optimizer.zero_grad()
  loss.backward()
  writer.add_scalar("Loss/train", loss, i)
  optimizer.step()

  if i % 50 == 49:
    print(f"Epoch {i + 1}:")
    print_report(y_train, y_train_pred_prob)
```

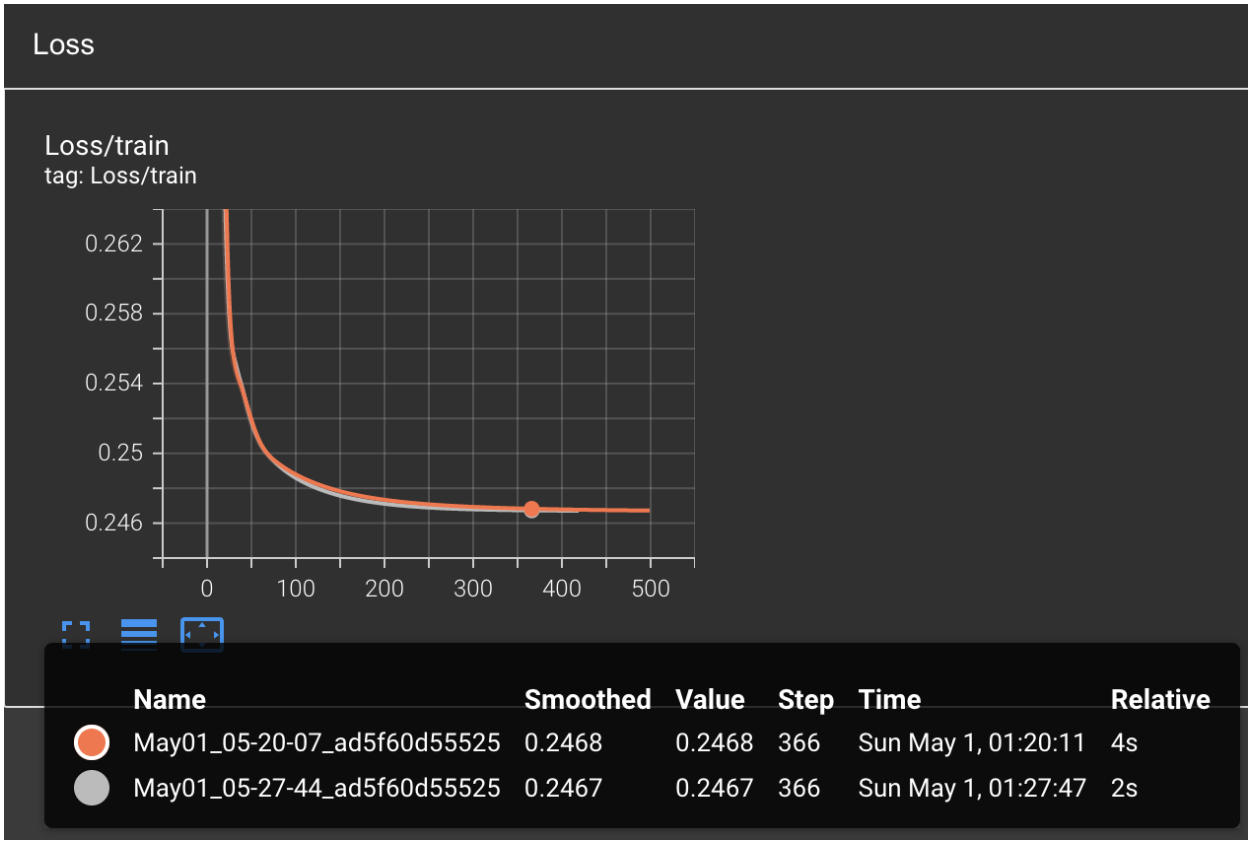
```
Train data :
Epoch 50:
Accuracy : 0.9193 ; ROC_AUC : 0.7406
Epoch 100:
Accuracy : 0.9193 ; ROC_AUC : 0.7503
Epoch 150:
Accuracy : 0.9193 ; ROC_AUC : 0.7541
Epoch 200:
Accuracy : 0.9193 ; ROC_AUC : 0.7556
Epoch 250:
Accuracy : 0.9193 ; ROC_AUC : 0.7564
Epoch 300:
Accuracy : 0.9193 ; ROC_AUC : 0.7567
Epoch 350:
Accuracy : 0.9193 ; ROC_AUC : 0.7569
Epoch 400:
Accuracy : 0.9193 ; ROC_AUC : 0.757
Epoch 450:
Accuracy : 0.9193 ; ROC_AUC : 0.757
Epoch 500:
Accuracy : 0.9193 ; ROC_AUC : 0.757
```

In [ ]:

```
%load_ext tensorboard
```

In [ ]:

```
%tensorboard --logdir=runs
```



Evaluation of Neural Network model

In [285…

```
model1.eval()
y_test_gpu = y_test_gpu.reshape(-1, 1)
with torch.no_grad():
    print(full_X_test_gpu.shape)
    y_test_pred_prob=model1(full_X_test_gpu)
    print('-' * 50)
    print('Test data : ')
    print_report(y_test, y_test_pred_prob)
    print('-' * 50)
```

```
torch.Size([92254, 295])
--------------------------------------------------
Test data :
Accuracy : 0.9193 ; ROC_AUC : 0.7558
--------------------------------------------------
```

Multi Layer NN model with custom Hinge & CXE Loss

Model Definition

The model contain one linear layer, one hidden layer with Relu function and sigmoid function for probability prediction.

In [409…
```
## Model using hidden layers
class SVMNNmodel(nn.Module):
  def __init__(self, input_features , hidden1 = 80, hidden2 = 80, output_feature
    super(SVMNNmodel, self).__init__()
    self.f_connected1 = nn.Linear(input_features, hidden1)
    self.out = nn.Linear(hidden1, output_features)

  def forward(self, x):
    h_relu = torch.relu(self.f_connected1(x))
    y_target_pred = torch.sigmoid(self.out(h_relu))
    return y_target_pred
```

In [410…
```
class SVMLoss(nn.Module):
  def __init__(self):
    super(SVMLoss,self).__init__()
  def forward(self,outputs,labels,model2):
     C = 0.10
    # data_loss = torch.mean(torch.clamp(1 - outputs.squeeze().t() == labels,min
     data_loss = torch.mean(torch.clamp(1 - outputs.squeeze(),min=0))
     weight = model2.out.weight.squeeze()
     reg_loss = weight.t() @ weight
     reg_loss = reg_loss + ( model2.out.bias.squeeze()**2)
     hinge = data_loss +( C*reg_loss/2)
     return (hinge)
```

In [411…
```
class Converttensor(Dataset):
    def __init__(self, feature, label, mode ='train', transforms=None):
        """
        Initialize data set as a list of IDs corresponding to each item of data

        :param feature: x - numpy array
        :param label: y - numpy array
        """

        self.x = feature
        self.y = label


    def __len__(self):
        """
        Return the length of data set using list of IDs

        :return: number of samples in data set
        """
        return (self.x.shape[0])

    def __getitem__(self, index):
        """
        Generate one item of data set.

        :param index: index of item in IDs list

        :return: image and label and bouding box params
        """
        x = self.x[index,:]
        y_target = self.y[index]
```

```
        x = torch.FloatTensor(x)
        y_target_arr = np.array(y_target)
        return x, y_target_arr
```

In [412…

```
fprs_net_train, tprs_net_train, fprs_net_valid, tprs_net_valid = [], [], [], []
roc_auc_net_train = 0.0
roc_auc_net_valid = 0.0
num_epochs=25
batch_size=256
CASE_NAME = "NN"
```

In [413…

```
splits = 1

# Train Test split percentage
subsample_rate = 0.3

finaldf = np.array_split(train_dataset, splits)
X_train = finaldf[0][selected_features]
y_train = finaldf[0]['TARGET']
final_X_kaggle_test = X_kaggle_test
## split part of data
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, stratify=y
                                                    test_size=subsample_rate, ra

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,stratify=

nn_X_train = data_prep_pipeline.fit_transform(X_train)
nn_X_valid = data_prep_pipeline.transform(X_valid)
nn_X_test = data_prep_pipeline.transform(X_test)
nn_X_kaggle_test = data_prep_pipeline.transform(final_X_kaggle_test)
full_X_kaggle_gpu = torch.FloatTensor(nn_X_kaggle_test).to(device)
nn_y_train = np.array(y_train)
nn_y_valid = np.array(y_valid)

in_feature_cnt = nn_X_train.shape[1]
out_feature_cnt = 1

print(f"X train           shape: {nn_X_train.shape}")
print(f"X validation      shape: {nn_X_valid.shape}")
print(f"X test            shape: {nn_X_test.shape}")
print(f"X kaggle_test     shape: {nn_X_kaggle_test.shape}")
print("Feature count           : ",in_feature_cnt)
```

```
X train           shape: (182968, 295)
X validation      shape: (32289, 295)
X test            shape: (92254, 295)
X kaggle_test     shape: (48744, 295)
Feature count           :  295
```

In [414…

```
nn_dataset = {'train': nn_X_train, 'val': nn_X_valid}
dataset_sizes = {x_type : len(nn_dataset[x_type]) for x_type in ['train','val']}
```

In [415…

```
dataset_sizes
```

Out[415… {'train': 182968, 'val': 32289}

In [416…
```
nn_dataset['train'] = Converttensor(nn_dataset['train'] ,nn_y_train, mode='train
```

In [417…
```
nn_dataset['val'] = Converttensor(nn_dataset['val'] ,nn_y_valid, mode='validatio
```

In [418…
```
nn_dataset
```

Out[418…
```
{'train': <__main__.Converttensor at 0x7ff4dd52d1d0>,
 'val': <__main__.Converttensor at 0x7ff4dd4fdbd0>}
```

In [419…
```
dataloaders = {x_type: torch.utils.data.DataLoader(nn_dataset[x_type], batch_siz
               for x_type in ['train', 'val']}
```

Train Model

In [420…
```
nn_model = SVMNNmodel(input_features = in_feature_cnt, output_features= 1).to(de
```

In [421…
```
try:
       convergence
       epoch_offset=convergence.epoch.iloc[-1]+1
except NameError:
        convergence=pd.DataFrame(columns=['epoch','phase','roc_auc','accuracy','
        epoch_offset=0
```

In [422…
```
def train(optimizer_cxe,optimizer_hinge,criteron,scheduler_cxe,scheduler_hinge,n

    global roc_auc_train
    global roc_auc_valid

    fac_cel=torch.tensor(w_cel)

    start = time.time()

    best_model_wts = copy.deepcopy(nn_model.state_dict())
    best_acc = 0.0

    # Store results to easier collect stats
    nn_y_pred = {x: np.zeros((dataset_sizes[x],1)) for x in ['train', 'val']}
    write= SummaryWriter()

    for epoch in range(num_epochs):

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            t0=time.time()
            # Reset to zero to be save

            nn_y_pred[phase].fill(0)
            if phase == 'train':
                nn_model.train()  # Set model to training mode
            else:
```

```
            nn_model.eval()   # Set model to evaluate mode

        running_loss = 0.0
        running_corrects = 0
        running_hinge = 0.0
        running_cxe = 0.0

        # Iterate over data.
        ix=0
        for inputs, targets in dataloaders[phase]:
            n_batch = len(targets)

            #nn_y_pred[phase][ix:ix+n_batch,:] = targets.detach().numpy().re

            inputs = inputs.to(device)
            targets = targets.to(device).float()

            # zero the parameter gradients
            optimizer_hinge.zero_grad()
            optimizer_cxe.zero_grad()

            # forward
            # track history if only in train
            with torch.set_grad_enabled(phase == 'train'):
                output_target = nn_model.forward(inputs)
                preds = torch.where((output_target > .5), 1, 0)
                #print(output_target.squeeze(),targets)
                ix += n_batch
                loss_cxe = func.binary_cross_entropy(output_target.squeeze()
                loss_hinge = criteron.forward(output_target.squeeze(), targe

                # backward + optimize only if in training phase
                if phase == 'train':
                    loss_hinge.backward()
                    optimizer_hinge.step()
                    #loss_cxe.backward()
                    optimizer_cxe.step()

            # statistics
            running_hinge += loss_hinge.item() * inputs.size(0)
            running_corrects += 1*(preds == targets.data.int()).sum().item()
            running_cxe += loss_cxe.item() * inputs.size(0)

            writer.add_scalar("Loss/train", running_cxe, epoch)

        if phase == 'train':
            scheduler_hinge.step()
            scheduler_cxe.step()

        epoch_cxe = running_cxe / dataset_sizes[phase]
        epoch_hinge = running_hinge / dataset_sizes[phase]
        epoch_acc = running_corrects / dataset_sizes[phase]

        epoch_roc_auc = 0.0
        if (phase == 'train'):
            ## Calculate 'false_positive_rate' and 'True_positive_rate' of t

            nn_fprs_train, nn_tpr_train, nn_thresholds = roc_curve(targets.d
            fprs_net_train.append(nn_fprs_train)
            tprs_net_train.append(nn_tpr_train)
```

```
                roc_auc_train = round(auc(nn_fprs_train, nn_tpr_train), 4)
                epoch_roc_auc = roc_auc_train

            elif (phase == 'val'):
                ## Calculate 'false_positive_rate' and 'True_positive_rate' of v
                nn_fpr_valid, nn_tpr_valid, thresholds = roc_curve(targets.detac
                fprs_net_valid.append(nn_fpr_valid)
                tprs_net_valid.append(nn_tpr_valid)
                roc_auc_valid = round(auc(nn_fpr_valid, nn_tpr_valid), 4)
                epoch_roc_auc = roc_auc_valid

            dt=time.time() - t0
            fmt='{:6s} ROC_AUC: {:.4f} Acc: {:.4f} CXE: {:.4f} Hinge: {:.4f}  DT
            out_list=[phase, epoch_roc_auc, epoch_acc, epoch_cxe, epoch_hinge] +
            out_str=fmt.format(*out_list)
            if phase=='train':
                epoch_str='Epoch {}/{} '.format(epoch, num_epochs)
                out_str=epoch_str + out_str
            else:
                out_str = ' '*len(epoch_str) + out_str
            print(out_str)

            if (phase == 'val') and epoch == num_epochs-1:
                 plt.plot(nn_fprs_train, nn_tpr_train, color='blue')
                 plt.plot(nn_fpr_valid, nn_tpr_valid, color='orange')
                 plt.xlim([0.0,1.0])
                 plt.ylim([0.0,1.0])
                 plt.xlabel('False Positive Rate')
                 plt.ylabel('True Positive Rate')
                 plt.title(f'ROC Curve Comparison')
                 plt.legend([f'TrainRocAuc (AUC = {roc_auc_train})', f'TestRocAu
                 plt.show()

            convergence.loc[len(convergence)] = [epoch+epoch_offset,phase,
                        epoch_roc_auc, epoch_acc, epoch_cxe, epoch_hinge]

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(nn_model.state_dict())

    time_elapsed = time.time() - start
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    nn_model.load_state_dict(best_model_wts)
```

Execute Model

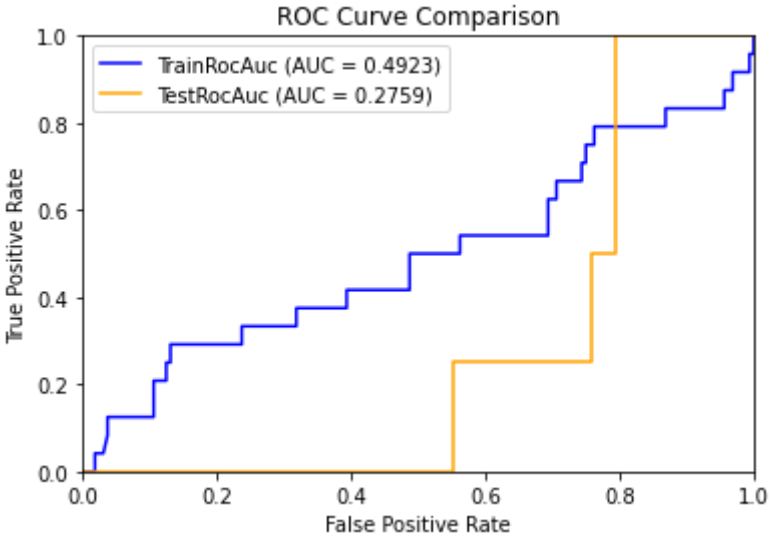
In [424…

```
learning_rate = 0.0001
optimizer_cxe = optim.Adam(nn_model.parameters(), lr=0.0001)
optimizer_hinge = torch.optim.SGD(nn_model.parameters(), lr=learning_rate,moment
nn_model = nn_model.cuda()
scheduler_cxe = lr_scheduler.StepLR(optimizer_cxe, step_size=10, gamma=0.1)
scheduler_hinge= lr_scheduler.StepLR(optimizer_hinge, step_size=10, gamma=0.1)
criteron = SVMLoss()
```

```
train(optimizer_cxe,optimizer_hinge,criteron,scheduler_cxe,scheduler_hinge,num_e

t0=time.time()
date_time = datetime.now().strftime("--%Y-%m-%d-%H-%M-%S-%f")
pickle.dump(nn_model,open(DATA_DIR + '/' + CASE_NAME + date_time + '.p','wb'))
print('Pickled in {:.2f} sec'.format(time.time()-t0))
```

```
Epoch 0/25 train  ROC_AUC: 0.3807 Acc: 20.6600 CXE: 6.4272 Hinge: 0.0062  DT=9.4
           val    ROC_AUC: 1.0000 Acc: 20.6624 CXE: 6.8345 Hinge: 0.0045  DT=0.9
Epoch 1/25 train  ROC_AUC: 0.5266 Acc: 20.6588 CXE: 7.0525 Hinge: 0.0037  DT=5.2
           val    ROC_AUC: 0.4224 Acc: 20.6417 CXE: 7.2803 Hinge: 0.0031  DT=0.7
Epoch 2/25 train  ROC_AUC: 0.5099 Acc: 20.6596 CXE: 7.4293 Hinge: 0.0027  DT=5.2
           val    ROC_AUC: 0.6935 Acc: 20.6555 CXE: 7.5849 Hinge: 0.0024  DT=0.7
Epoch 3/25 train  ROC_AUC: 0.5669 Acc: 20.6588 CXE: 7.6981 Hinge: 0.0021  DT=5.2
           val    ROC_AUC: 0.0444 Acc: 20.6486 CXE: 7.8398 Hinge: 0.0019  DT=0.7
Epoch 4/25 train  ROC_AUC: 0.6509 Acc: 20.6576 CXE: 7.9193 Hinge: 0.0018  DT=5.2
           val    ROC_AUC: 0.4677 Acc: 20.6555 CXE: 8.0420 Hinge: 0.0016  DT=0.7
Epoch 5/25 train  ROC_AUC: 0.4603 Acc: 20.6611 CXE: 8.1222 Hinge: 0.0015  DT=5.2
           val    ROC_AUC: 0.9062 Acc: 20.6624 CXE: 8.2142 Hinge: 0.0014  DT=0.6
Epoch 6/25 train  ROC_AUC: 0.4503 Acc: 20.6604 CXE: 8.2990 Hinge: 0.0013  DT=5.2
           val    ROC_AUC: nan Acc: 20.6693 CXE: 8.3802 Hinge: 0.0012  DT=0.7
Epoch 7/25 train  ROC_AUC: 0.4221 Acc: 20.6596 CXE: 8.4617 Hinge: 0.0011  DT=5.2
           val    ROC_AUC: 0.3929 Acc: 20.6348 CXE: 8.6156 Hinge: 0.0011  DT=0.7
Epoch 8/25 train  ROC_AUC: 0.6176 Acc: 20.6600 CXE: 8.6161 Hinge: 0.0010  DT=5.1
           val    ROC_AUC: 0.3778 Acc: 20.6486 CXE: 8.6942 Hinge: 0.0009  DT=0.7
Epoch 9/25 train  ROC_AUC: 0.4791 Acc: 20.6580 CXE: 8.7664 Hinge: 0.0009  DT=5.2
           val    ROC_AUC: 0.6643 Acc: 20.6348 CXE: 8.8792 Hinge: 0.0008  DT=0.7
Epoch 10/25 train  ROC_AUC: 0.4790 Acc: 20.6588 CXE: 8.8409 Hinge: 0.0008  DT=5.
2
            val    ROC_AUC: 0.3710 Acc: 20.6555 CXE: 8.8528 Hinge: 0.0008  DT=0.
7
Epoch 11/25 train  ROC_AUC: 0.5155 Acc: 20.6580 CXE: 8.8568 Hinge: 0.0008  DT=5.
1
            val    ROC_AUC: 0.2253 Acc: 20.6279 CXE: 8.8821 Hinge: 0.0008  DT=0.
7
Epoch 12/25 train  ROC_AUC: 0.5636 Acc: 20.6580 CXE: 8.8777 Hinge: 0.0008  DT=5.
2
            val    ROC_AUC: 0.7188 Acc: 20.6624 CXE: 8.8931 Hinge: 0.0008  DT=0.
7
Epoch 13/25 train  ROC_AUC: 0.5342 Acc: 20.6604 CXE: 8.8975 Hinge: 0.0008  DT=5.
1
            val    ROC_AUC: 0.9375 Acc: 20.6624 CXE: 8.9145 Hinge: 0.0008  DT=0.
7
Epoch 14/25 train  ROC_AUC: 0.4688 Acc: 20.6560 CXE: 8.9199 Hinge: 0.0008  DT=5.
2
            val    ROC_AUC: 0.7069 Acc: 20.6417 CXE: 8.9378 Hinge: 0.0008  DT=0.
7
Epoch 15/25 train  ROC_AUC: 0.4593 Acc: 20.6592 CXE: 8.9421 Hinge: 0.0008  DT=5.
1
            val    ROC_AUC: 0.2786 Acc: 20.6348 CXE: 8.9690 Hinge: 0.0007  DT=0.
7
Epoch 16/25 train  ROC_AUC: 0.4592 Acc: 20.6596 CXE: 8.9639 Hinge: 0.0007  DT=5.
1
            val    ROC_AUC: 0.0312 Acc: 20.6624 CXE: 9.0103 Hinge: 0.0007  DT=0.
7
Epoch 17/25 train  ROC_AUC: 0.6415 Acc: 20.6607 CXE: 8.9876 Hinge: 0.0007  DT=5.
1
            val    ROC_AUC: 0.5645 Acc: 20.6555 CXE: 8.9971 Hinge: 0.0007  DT=0.
7
Epoch 18/25 train  ROC_AUC: 0.4921 Acc: 20.6619 CXE: 9.0071 Hinge: 0.0007  DT=5.
1
            val    ROC_AUC: 0.5312 Acc: 20.6624 CXE: 9.0345 Hinge: 0.0007  DT=0.
7
```

```
Epoch 19/25 train  ROC_AUC: 0.6957 Acc: 20.6611 CXE: 9.0352 Hinge: 0.0007  DT=5.
2
            val    ROC_AUC: 0.4839 Acc: 20.6555 CXE: 9.0547 Hinge: 0.0007  DT=0.
7
Epoch 20/25 train  ROC_AUC: 0.4840 Acc: 20.6604 CXE: 9.0395 Hinge: 0.0007  DT=5.
2
            val    ROC_AUC: 0.8750 Acc: 20.6624 CXE: 9.0568 Hinge: 0.0007  DT=0.
7
Epoch 21/25 train  ROC_AUC: 0.5197 Acc: 20.6611 CXE: 9.0485 Hinge: 0.0007  DT=5.
1
            val    ROC_AUC: 0.1875 Acc: 20.6624 CXE: 9.0546 Hinge: 0.0007  DT=0.
6
Epoch 22/25 train  ROC_AUC: 0.5905 Acc: 20.6576 CXE: 9.0466 Hinge: 0.0007  DT=5.
1
            val    ROC_AUC: 0.5862 Acc: 20.6417 CXE: 9.0611 Hinge: 0.0007  DT=0.
7
Epoch 23/25 train  ROC_AUC: 0.4290 Acc: 20.6600 CXE: 9.0489 Hinge: 0.0007  DT=5.
2
            val    ROC_AUC: 0.3707 Acc: 20.6417 CXE: 9.0663 Hinge: 0.0007  DT=0.
6
Epoch 24/25 train  ROC_AUC: 0.4923 Acc: 20.6560 CXE: 9.0547 Hinge: 0.0007  DT=5.
1
            val    ROC_AUC: 0.2759 Acc: 20.6417 CXE: 9.0734 Hinge: 0.0007  DT=0.
7
```

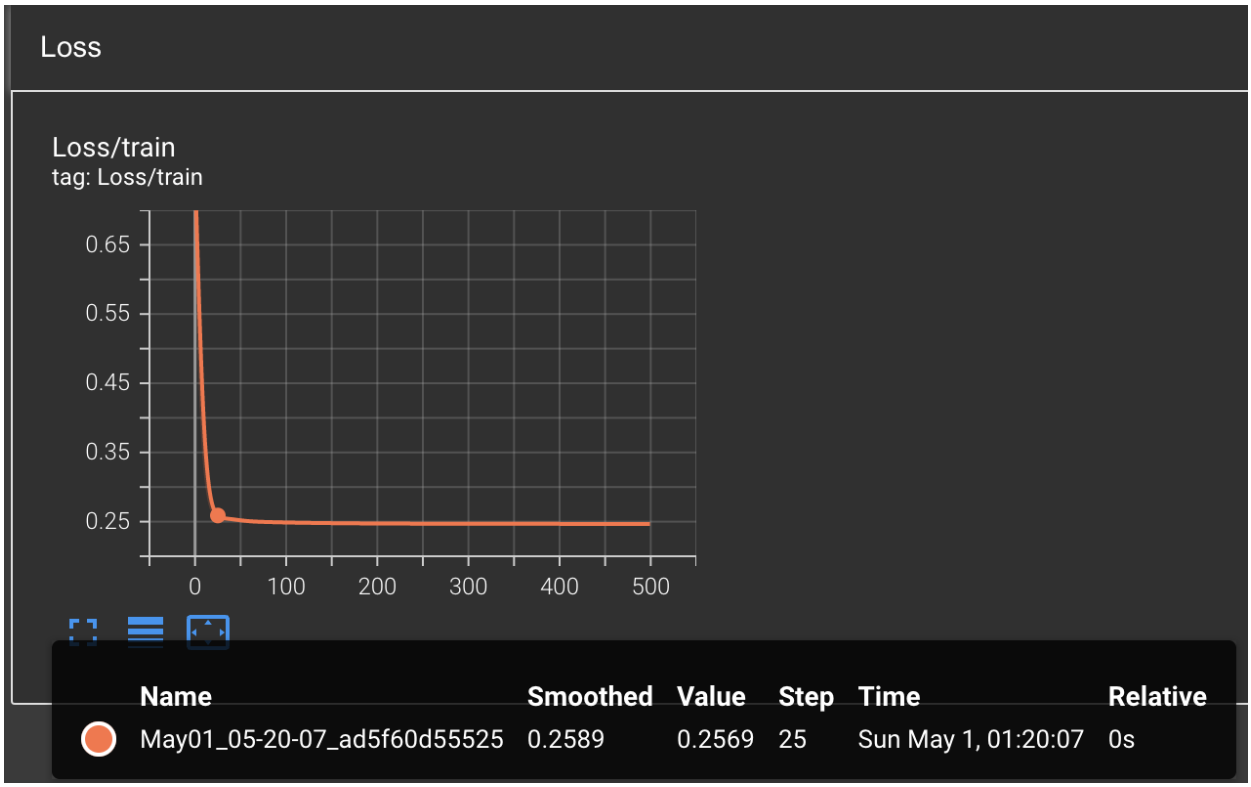



```
Training complete in 2m 31s
Best val Acc: 20.669330
Pickled in 0.00 sec
```

In [ ]:
```
%load_ext tensorboard
```

In [ ]:
```
%tensorboard --logdir=runs
```

Multi layer Neural Network Model results below:



In [427…
```
convergence.head(5)
```

Out[427…

	epoch	phase	roc_auc	accuracy	CXE	Hinge
0	0	train	0.4601	30.593132	0.809022	0.458004
1	0	train	0.5259	20.198315	1.011704	0.369346
2	0	train	0.5931	20.228558	1.263604	0.284973
3	1	train	0.3506	25.554807	2.573320	0.154764
4	2	train	0.6365	25.930993	2.520688	0.158814

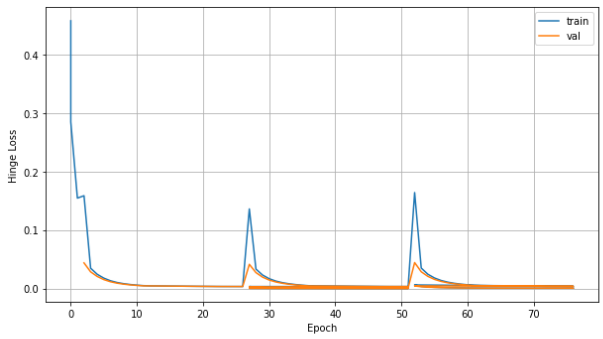
Plot Convergence

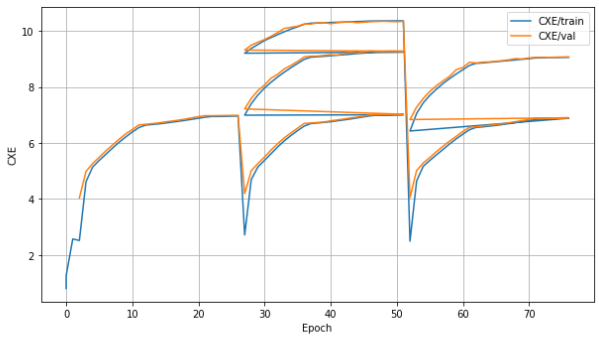
In [428…
```
from scipy.stats import iqr

def plot_convergence(figsize=(22,12)):
  conv = {phase : convergence[convergence.phase==phase]
          for phase in ['train','val']}

  fig,axes = plt.subplots(2,2,figsize=figsize)
  cols = {'train' : 'tab:blue', 'val' : 'tab:orange'}

  # Loss
  ax=axes[0,0]
  for phase in ['train','val']:
    ax.plot(conv[phase].epoch,conv[phase].Hinge,label=phase,c=cols[phase])
  ax.set_xlabel('Epoch')
  ax.set_ylabel('Hinge Loss')
  ax.legend()
```

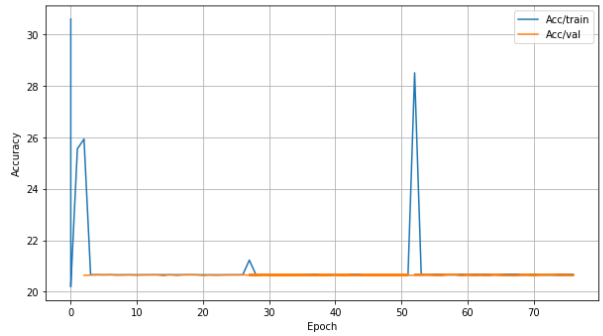
```
ax.grid()

# CXE
ax=axes[0,1]
for phase in ['train','val']:
  ax.plot(conv[phase].epoch,conv[phase].CXE,label='CXE/'+phase,c=cols[phase])
ax.set_xlabel('Epoch')
ax.set_ylabel('CXE')
ax.legend()
ax.grid()

# Accuracy
ax=axes[1,0]
for phase in ['train','val']:
  ax.plot(conv[phase].epoch,conv[phase].accuracy,label='Acc/'+phase,c=cols[pha
ax.set_xlabel('Epoch')
ax.set_ylabel('Accuracy')
ax.legend()
ax.grid()

# Plot ROC_AUC Curve of train and test
ax=axes[1,1]
for i in range(num_epochs):
    plt.plot(fprs_net_train[i],tprs_net_train[i], color='blue')
    plt.plot(fprs_net_valid[i],tprs_net_valid[i], color='orange')
ax.set_xlim([0.0,1.0])
ax.set_ylim([0.0,1.0])
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.set_title(f'ROC Curve Comparison')
ax.legend([f'TrainRocAuc (AUC = {roc_auc_train})', f'TestRocAuc (AUC = {roc_au
ax.grid()
```

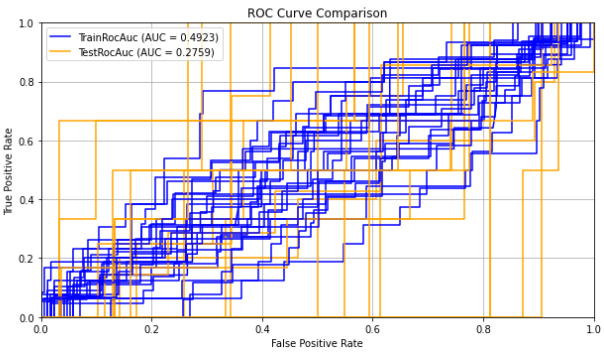
In [429…

```
plot_convergence(figsize=(22,12))
```







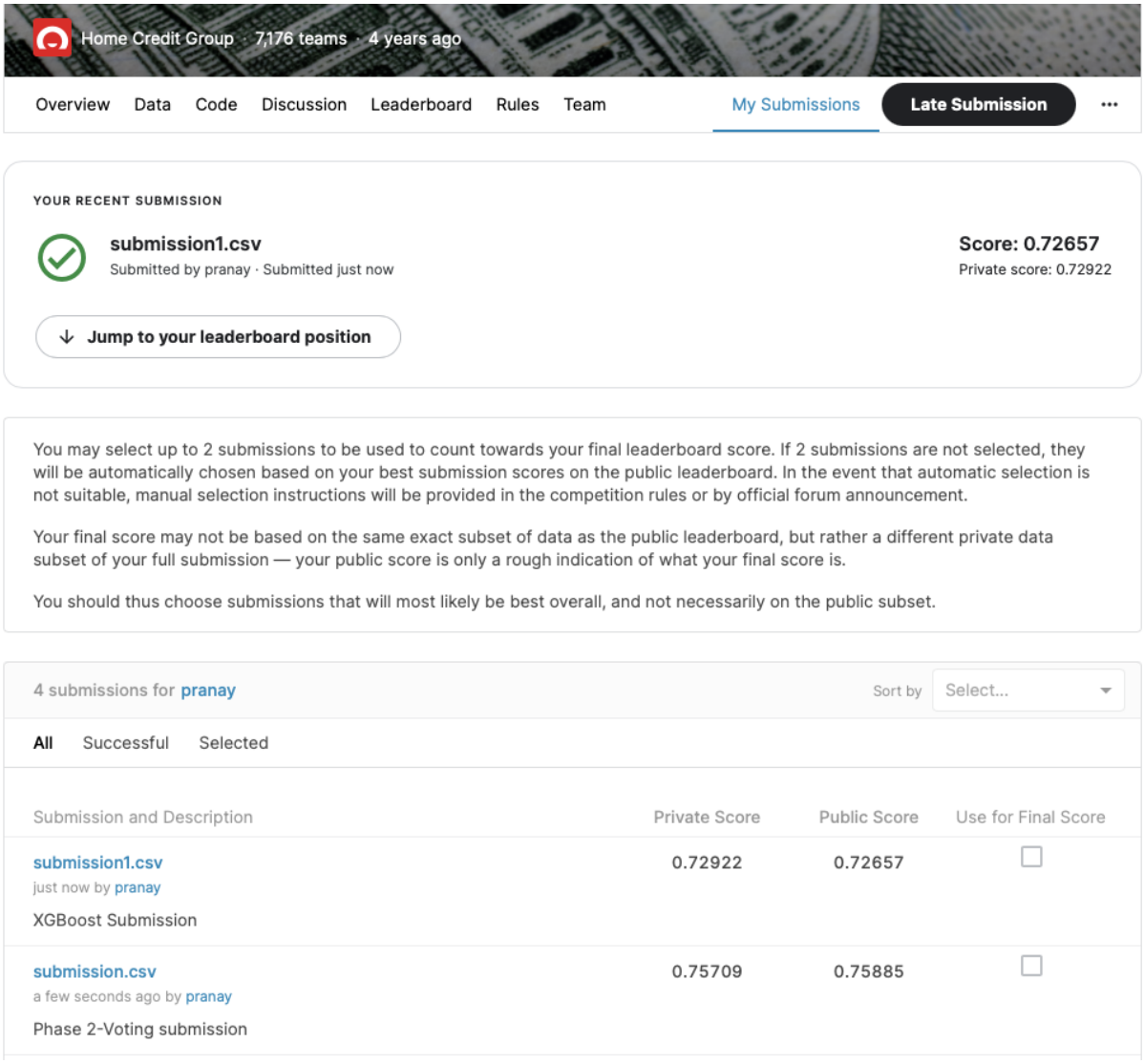


Kaggle submission via the command line API

Submitted Via website

In [ ]:
```
! kaggle competitions submit -c home-credit-default-risk -f submission.csv -m "P
```

In [ ]:
```
! kaggle competitions submit -c home-credit-default-risk -f submission1.csv -m "
```



Kaggle Submission for Neural Networks Model.

In [263…
```
nn_model.eval()
test_class_scores = nn_model(full_X_kaggle_gpu)
print(test_class_scores[0:10])
```

```
tensor([[0.9958],
        [0.9988],
        [0.9915],
        [0.9950],
        [0.9964],
        [1.0000],
```

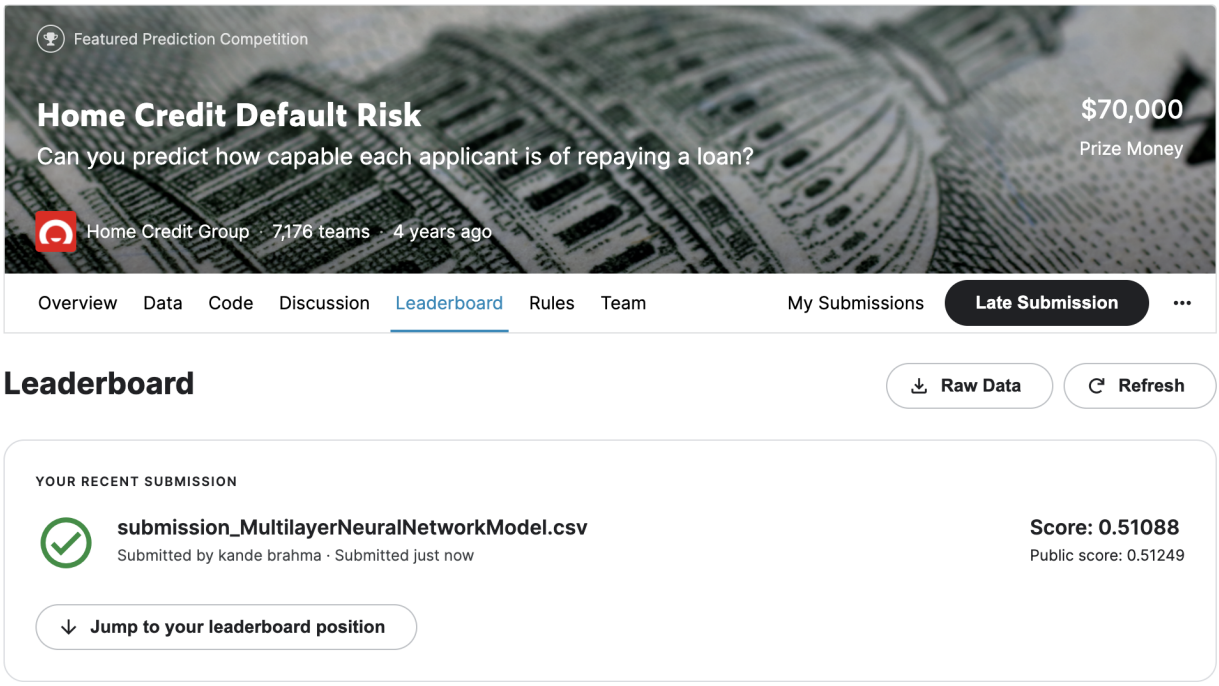
```
        [0.9926],
        [0.9970],
        [0.9966],
        [0.9997]], device='cuda:0', grad_fn=<SliceBackward0>)
```

In [264…

```
#For each SK_ID_CURR in the test set, you must predict a probability for the TAR
fs_type = "MultilayerNeuralNetworkModel"
submit_df = datasets["application_test"][['SK_ID_CURR']]
submit_df['TARGET'] = test_class_scores.detach().cpu().numpy()
print(submit_df.head(2))
submit_df.to_csv(f'/content/submission_{fs_type}.csv',index=False)
```

```
   SK_ID_CURR    TARGET
0      100001  0.995847
1      100005  0.998813
```

Kaggle Submission Score for Multilayer Neural Network Model



Write-up

Home Credit Default Risk

Team Members(Group 28):

Sno	***Name**	***Email***
1.	Vamsee Krishna Sai Naramsetty	vnarams@iu.edu
2.	Harishanker Brahma Kande	hkande@iu.edu

Sno	*Name	*Email*
3.	Pranay Reddy Dasari	pdasari@iu.edu
4.	Jaswanth Kumar Ranga	jranga@iu.edu

Overview

The course project is based on the Kaggle Competition on Home Credit Default Risk (HCDR). This project's purpose is to anticipate if a client will repay a loan. Home Credit uses a number of alternative data—including telco and transactional information—to estimate their clients' repayment ability in order to ensure that people who struggle to secure loans owing to weak or non-existent credit histories have a pleasant loan experience.

Abstract

The purpose of this project is to create a machine learning model/deep learning model that can predict consumer behavior during loan repayment.

In this phase our goal is to build a multi-layer neural network model in Pytorch and use Tensorboard to visualize real-time training results.This phase focused on building high performance Neural Networks and monitoring error generalization with early stopping technique and evaluating the model performance by monitoring through loss functions such as CXE and Hinge Loss.We did built 2 models, First model contains one linear layer with Relu function for

probability prediction and the second model contain one linear layer, one hidden layer with Relu function and sigmoid function for probability prediction. Using Tensorboard we visualize the CXE loss for training data for each epoch.

Our results in this phase for multi layer neural network model the AUC scores are 0.588 for train data and 0.5172 for test data. For single layer model the AUC score is 0.7558 for test data. For our submission in kaggle we received a public score of 0.512 and private score of 0.510.

Project Description

Home Credit is an international non-bank financial institution, which primarily focuses on lending people money regardless of their credit history. Home credit groups aim to provide positive borrowing experience to customers, who do not bank on traditional sources for pertaining loans. Hence, Home Credit Group published a dataset on Kaggle website with the objective of identifying and solving unfair loan rejection.

The purpose of this project is to create a machine learning model that can predict consumer behavior during loan repayment. Our task in this phase is to create a pipeline to build a baseline machine learning model using Logistic Regression algorithm. The resultant model will be evaluated with various performance metrics in order to build a better model. Companies can be able to rely on the output of this model to identify if loan is at risk to default. The new model built would help companies to avoid losses and make significant profits and 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.

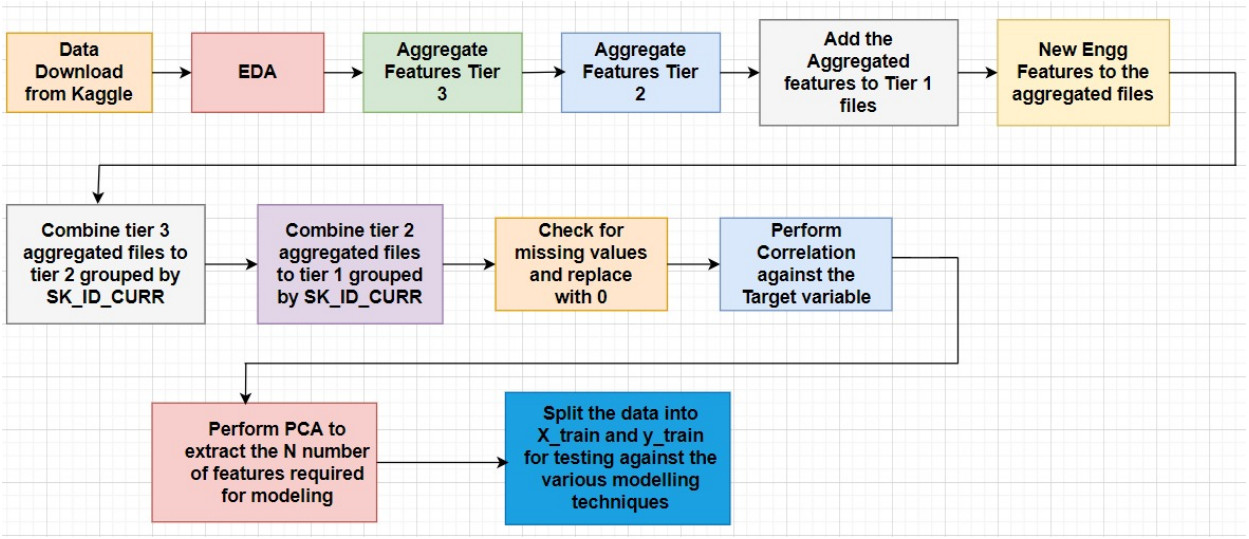
The results of our machine learning pipelines will be measured using the follwing metrics;

- Confusion Matrix
- Accuracy Score
- Precision
- Recall
- F1 score
- AUC (Area Under ROC Curve)

The pipeline results will be compared and ranked using the appropriate measurements and the most efficient pipeline will be submitted to the HCDR Kaggle Competition.

Workflow

For this project, we are following the proposed workflow as mentioned below.



Data Description

Overview The full dataset consists of 7 tables. There is 1 primary table and 6 secondary tables.

Primary Tables

- application_train

  This Primary table includes the application information for each loan application at Home Credit in one row.
  This row includes the target variable of whether or not the loan was repaid. We use this field as the basis to determine the feature importance. The target variable is binary in nature based since this is a classification problem.
  * ‘1' - client with payment difficulties: he/she had late payment more than N days on at least one of the first M installments of the loan in our sample
  * '0' - all other cases
  The number of columns are 122. The number of data entries are 307,511.
- application_test

  This table includes the application information for each loan application at Home Credit in one row. The features are the same as the train data but exclude the target variable
  The number of columns are 121. The number of data entries are 48,744.

Secondary Tables

- Bureau

  This table contains data points for all client's previous credits provided by other financial institutions that were reported to the Credit Bureau. There is one row for each previous credit, meaning a

many-to-one relationship with the primary table. We could join it with primary table by using current application ID, SK_ID_CURR.
The number of columns are 17.The number of data entries are 1,716,428.

- Bureau Balance

  This dataset has the monthly balance history of every previous credit reported to the Credit Bureau. There is one row for each monthly balance, meaning a many-to-one relationship with the Bureau table. We could join it with bureau table by using bureau's ID, SK_ID_BUREAU.
  The number of columns are 3. The number of data entries are 27,299,925

- Previous Application

  This table contains records for all previous applications for Home Credit loans of clients who have loans in our sample. There is one row for each previous application related to loans in our data sample. , meaning a many-to-one relationship with the primary table. We could join it with primary table by using current application ID, SK_ID_CURR.
  There are four types of contracts:
  a. Consumer loan(POS – Credit limit given to buy consumer goods)
  b. Cash loan(Client is given cash)
  c. Revolving loan(Credit)
  d. XNA (Contract type without values)
  The number of columns are 37. The number of data entries are 1,670,214

- POS CASH Balance

  This table includes a monthly balance snapshot of a previous point of sale or cash loan that the customer has at Home Credit. There is one row for each monthly balance, meaning a many-to-one relationship with the Previous Application table. We would join it with Previous Application table by using previous application ID, SK_ID_PREV, then join it with primary table by using current application ID, SK_ID_CURR.
  The number of columns are 8. The number of data entries are 10,001,358.

- Credit Card Balance

  This table includes a monthly balance snapshot of previous credit cards the customer has with Home Credit. There is one row for each previous monthly balance, meaning a many-to-one relationship with the Previous Application table.We could join it with Previous Application table by using previous application ID, SK_ID_PREV, then join it with primary table by using current application ID, SK_ID_CURR.
  The number of columns are 23. The number of data entries are 3,840,312

- Installments Payments

  This table includes previous repayments made or not made by the customer on credits issued by Home Credit. There is one row for each

payment or missed payment, meaning a many-to-one relationship with the Previous Application table. We would join it with Previous Application table by using previous application ID, SK_ID_PREV, then join it with primary table by using current application ID, SK_ID_CURR.
  The number of columns are 8 . The number of data entries are 13,605,401

EDA(Exploratory Data Analysis)

Exploratory data analysis is important to this project because it helps to understand the data and it allows us to get closer to the certainty that the future results will be valid, accurately interpreted, and applicable to the proposed solution.

In the phase-1 of our project eda helped us to look at the summary statistics on each table and focussing on missing data, Outliers and aggregate functions such as mean, median etc and visual representation of features for better understanding of the data.

For identifying missing data we made use of categorical and numerical features. Specific features have been visualized based on their correlation values. The highly correlated features were used to plot the density to evaluate the distributions in comparison to the target variable. We used different plots such as countplot, heatmap, densityplot, catplot etc for visualizing our analysis.

Key Observations:

- In terms of correlation of features with target variable there are no highly correlated features which makes it interesting for feature engineering phase.

Feature Engineering and transformers

Feature Engineering is important because it is directly reflected in the quality of the machine learning model, This is because in this phase new features are created, transformed, dropped based on aggregator functions such as max, min, mean, sum and count etc.

- Including Custom domain knowledge based features
- Creating engineered aggregated features
- Experimental modelling of the data
- Validating Manual OHE
- Merging all datasets
- Drop Columns with Missing Values

Domain knowledge-based features, which assist increase a model's accuracy, are an important aspect of any feature engineering process. The first step was to determine which of these were applicable to each dataset. Credit card balance after payment based on due amount, application amount average, credit average, and other new custom features were among them. Available

credit as a percentage of income, Annuity as a percentage of income, Annuity as a percentage of available credit are all examples of percentages.

The next stage was to find the numerical characteristics and aggregate them into mean, minimum, and maximum values. During the engineering phase, an effort was made to use label encoding for unique values greater than 5. However, to reduce the amount of code required to perform the same functionality, a design choice was taken to apply OHE at the pipeline level for specified highly correlated variables on the final merged dataset.

Extensive feature engineering was carried out by experimenting with several modeling techniques using main, secondary, and tertiary tables before settling on an efficient strategy that used the least amount of memory. For Tier 3 tables bureau balance, credit card installment, installment payments, and point of sale systems cash balance, the first attempt entailed developing engineered and aggregated features. This was then combined with Tier 2 tables, such as prev application balance with credit card installment, installment payments, and point of sale systems cash balance, as well as aggregated features, to create prev application balance. Along with the core dataset application train, a flattened view comprising all of the aforementioned tables was integrated. As a result, there were a lot of redundant features that took up a lot of memory.

Attempt 2 involved creating custom and aggregated features for tier 3 tables and merging with tier 2 tables based on the primary key provided, which was later “extended” to the tier1 tables based on the additional aggregated columns. This approach created less duplicates, was optimized and occupied less memory by using a garbage collector after each merge.

A train dataframe was created by merging the Tier3, Tier2, and Tier1 datasets. There were extra precautions made to verify that no columns had more than 50% of the data missing. The characteristics were engineered and included in the model with modest divides to assist test the model, however the accuracy was low. However, for Random forest and XGBoost, employing these combined features in conjunction with acceptable splits throughout the training face resulted in improved accuracy and reduced the risk of overfitting. Label encoding for unique categorical values in all categorical fields, not just a few, will be the focus of future research and trials.

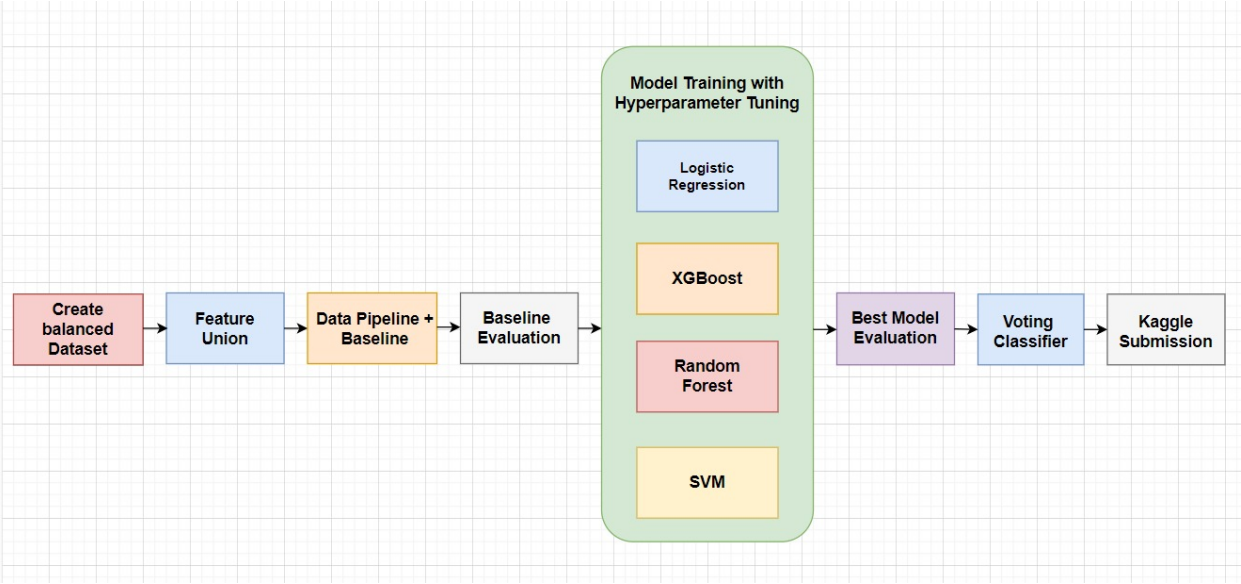
Pipelines

Phase-1

Logistic regression model is used as a baseline Model, since it's easy to implement yet provides great efficiency. Training a logistic regression model doesn't require high computation power. We also tuned the regularization, tolerance, and C hyper parameters for the Logistic regression model and compared the results with the baseline model. We used 5 fold cross fold validation with hyperparameters to tune the model and apply GridSearchCV function in Sklearn.

Below is the workflow for the model pipeline.

Phase-2



In Phase 1, we used the Logistic regression model as the baseline model since it didn't take a lot of computing resources and was simple to execute. We also used customized logistic models with a balanced dataset to increase the predictiveness of our model. In phase 2, we did look at different classification models to see if we can improve our forecast. Our main focus is on boosting algorithms, which are believed to be extremely efficient and relatively fast. The modeling workflow for phase 2 is depicted in the diagram below. We used XGBoost, RandomForest, and SVM in our research.

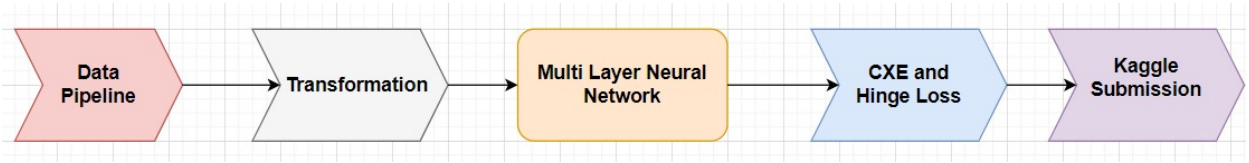
Below is the reason for choosing the mentioned models.

- XGBoost is one of the quickest implementations of gradient boosted trees. XGBoost is designed to handle missing values internally. This is helpful because there are many, many hyperparameters to tune.
- Random Forest is a tree-based machine learning algorithm that combines the output of multiple decision trees for making decisions. For each tree only a random subsample of the available features is selected for building the tree. Random Forest uses decision trees, which are more prone to overfitting.
- SVM performs similar to logistic regression when linear separation and performs well with non-linear boundaries depending on the kernel used. SVM is susceptible to overfitting/training issues depending on the kernel. A more complex kernel would overfit the model.

Boosting algorithms can overfit if the number of trees is very large. We did two submission in Kaggle, one using Voting Classifier and the other one with best classifier i.e. XGBoost. A Voting Classifier is a machine learning model that trains on an ensemble of various models and predicts an output based on their highest probability of chosen class as the output. We have chosen soft voting instead of hard voting since the soft voting predicts based on the average of all models.

Phase-3

Below is the workflow for the multi layer neural network model pipeline.



Below is the pipeline for the Multi layer neural network model.

- Multi layer neural network model has been choosen as it might provide results with high accuracy on the given data, If we are able to identify the accurate no of hidden layers it would help to improve the AUC score on the test data.

Hyperparameters Used

Below are the hyperparameters we used for training different models:

```
# Arrange grid search parameters for each classifier
params_grid = {
        'Logistic Regression': {
            'penalty': ('l1', 'l2'),
            'tol': (0.0001, 0.00001),
            'C': (10, 1, 0.1, 0.01),
        }
    ,
        'Support Vector' : {
            'kernel': ('rbf','poly'),
            'degree': (4, 5),
            'C': ( 0.0001, 0.001),   #Low C - allow for
misclassification
            'gamma':(0.01,0.1,1)  #Low gamma - high variance and
low bias
        }
    ,
        'XGBoost':  {
            'max_depth': [3,5], # Lower helps with overfitting
            'n_estimators':[200,300],
            'learning_rate': [0.01,0.1],
            'colsample_bytree' : [0.2],
        },                      #small numbers reduces accuracy but
runs faster

        'RandomForest':  {
            'max_depth': [5,10],
            'max_features': [15,20],
            'min_samples_split': [5, 10],
            'min_samples_leaf': [3, 5],
            'bootstrap': [True],
```

```
                'n_estimators':[100]},
        }
```

Best Parameters for All models

Logistic Regression

```
Best Parameters:
        predictor__C: 10
        predictor__penalty: l2
        predictor__tol: 0.0001
****** FINISH Logistic Regression  *****
```

Random Forest

```
Best Parameters:
        predictor__bootstrap: True
        predictor__max_depth: 10
        predictor__max_features: 20
        predictor__min_samples_leaf: 3
        predictor__min_samples_split: 5
        predictor__n_estimators: 100
****** FINISH RandomForest  *****
```

XGBoost Classifier

```
Best Parameters:
        predictor__colsample_bytree: 0.2
        predictor__learning_rate: 0.1
        predictor__max_depth: 5
        predictor__n_estimators: 300
****** FINISH XGBoost  *****
```

Experimental results

Traditional Models Below is the resulting table for the results on the given dataset.

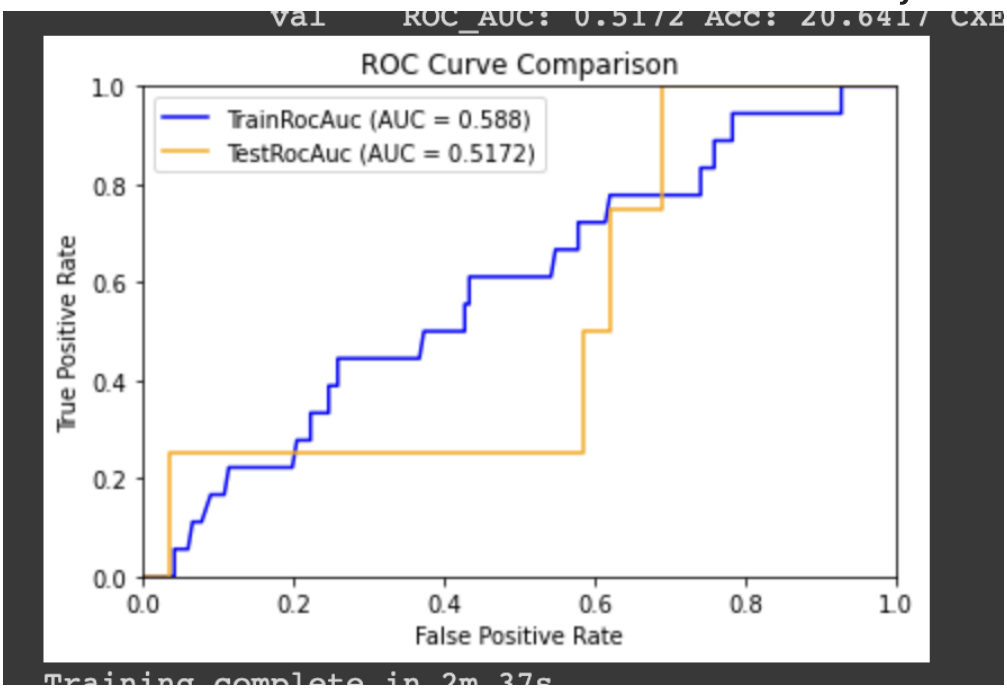
	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score	Valid F1 Score	Test F1 Score	Description
0	Baseline_183_features	0.9190	0.9184	0.9183	0.7695	0.7544	0.7529	0.0478	0.0449	0.0390	Imbalanced Logistic reg features 183: Num:170, Cat:13 with 20% training data
1	Baseline_183_features	0.7032	0.7002	0.7022	0.7703	0.7672	0.7511	0.7038	0.7007	0.2683	Balanced Logistic reg features 183: Num:170, Cat:13 with 20% training data
2	Logistic Regression	0.7005	0.6980	0.6984	0.7673	0.7647	0.7518	0.7011	0.6984	0.2672	[["predictor__C", 10], ["predictor__penalty", "l2"], ["predictor__tol", 0.0001]]
3	RandomForest	0.8590	0.8240	0.7877	0.9340	0.9034	0.7340	0.8618	0.8282	0.2758	[["predictor__bootstrap", true], ["predictor__max_depth", 10], ["predictor__max_features", 20], ["predictor__min_samples_leaf", 3], ["predictor__min_samples_split", 5], ["predictor__n_estimators", 100]]
4	Logistic Regression	0.7005	0.6980	0.6984	0.7673	0.7647	0.7518	0.7011	0.6984	0.2672	[["predictor__C", 10], ["predictor__penalty", "l2"], ["predictor__tol", 0.0001]]
5	RandomForest	0.8590	0.8240	0.7877	0.9340	0.9034	0.7340	0.8618	0.8282	0.2758	[["predictor__bootstrap", true], ["predictor__max_depth", 10], ["predictor__max_features", 20], ["predictor__min_samples_leaf", 3], ["predictor__min_samples_split", 5], ["predictor__n_estimators", 100]]
6	XGBoost	0.8600	0.8229	0.7865	0.9360	0.9026	0.7537	0.8626	0.8271	0.2931	[["predictor__colsample_bytree", 0.2], ["predictor__learning_rate", 0.1], ["predictor__max_depth", 5], ["predictor__n_estimators", 300]]

Deep Learning

Single Layer Neural Network

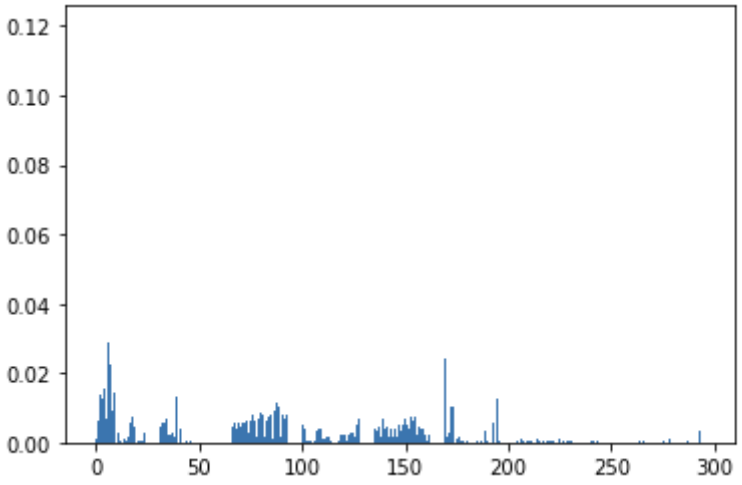
```
torch.Size([92254, 295])
-------------------------------------------------
Test data :
Accuracy : 0.9193 ; ROC_AUC : 0.7558
-------------------------------------------------
```

Multi Layer Neural Network

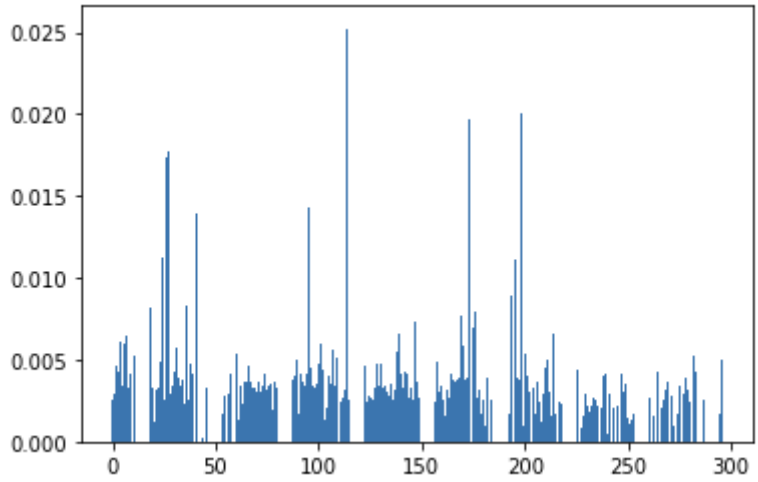


Feature Importance

Random Forest



XGBoost Classifier



Leakage Problem:

Proper measures has been taken in order to reduce the leakage problems by making use of Cross validation folds while training the model and by allocating specific amount of data to the validation set in training data due to which we feel we did take appropriate measures to handle the leakage problem during all the phases especially while splitting the data we are making sure to drop the target variable and then try to predict the model. Also we did make sure to perform OHE for categorical attributes and standard scaler for numerical attributes along with these we made use of imputing techniques such as most frequent one for categorical attributes and mean for numerical attributes, Therefore considering all these steps we do feel we handled the data leakage problem throughout the project.

Discussion of Results

Based on the models discussed above, XGBoost stood out as the best predictive model using the top 183 features with 75.37% ROC score and followed by Logistic regression and the worst performance by Multi layer neural network with 59.34% AUC score.

* Logistic Regression : This model was chosen as the baseline model trained with both balanced and imbalanced dataset with feature engineering. The training accuracy for this model 70.05% and test accuracy as 69.84%. A 75.18% ROC score resulted with best parameters for this model.

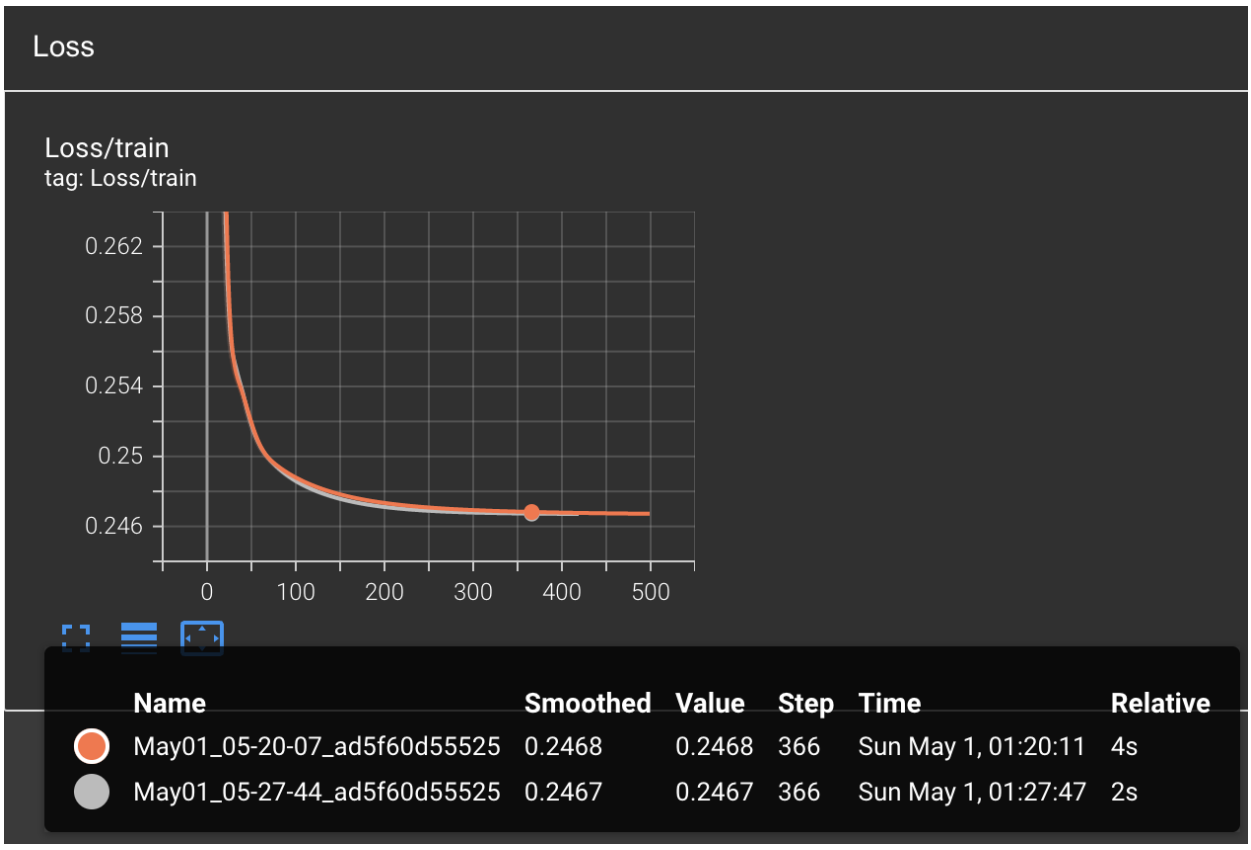
* XGBoost : By far this model resulted in the best model. Both in terms of timing and accuracy for the selected features and balanced dataset. The accuracy of the training and test are 86.00% and test 78.65%. Test ROC under the curve is 75.37%.

* Random Forest : On our last decision tree models, Random Forest produced training accuracy of 85.90% and test accuracy of 78.77%. Test ROC score came out as 73.40%.

* Multi layer Neural Network: By far this is the model which has been underperforming when compared to traditional models as it is resulting in 51.72% AUC score and for single layer it is resulting in 74.8% AUC score, Multi Layer neural network is underperforming due to lack of selecting best features and identifying the accurate no of hidden layers could be the possible reasons.

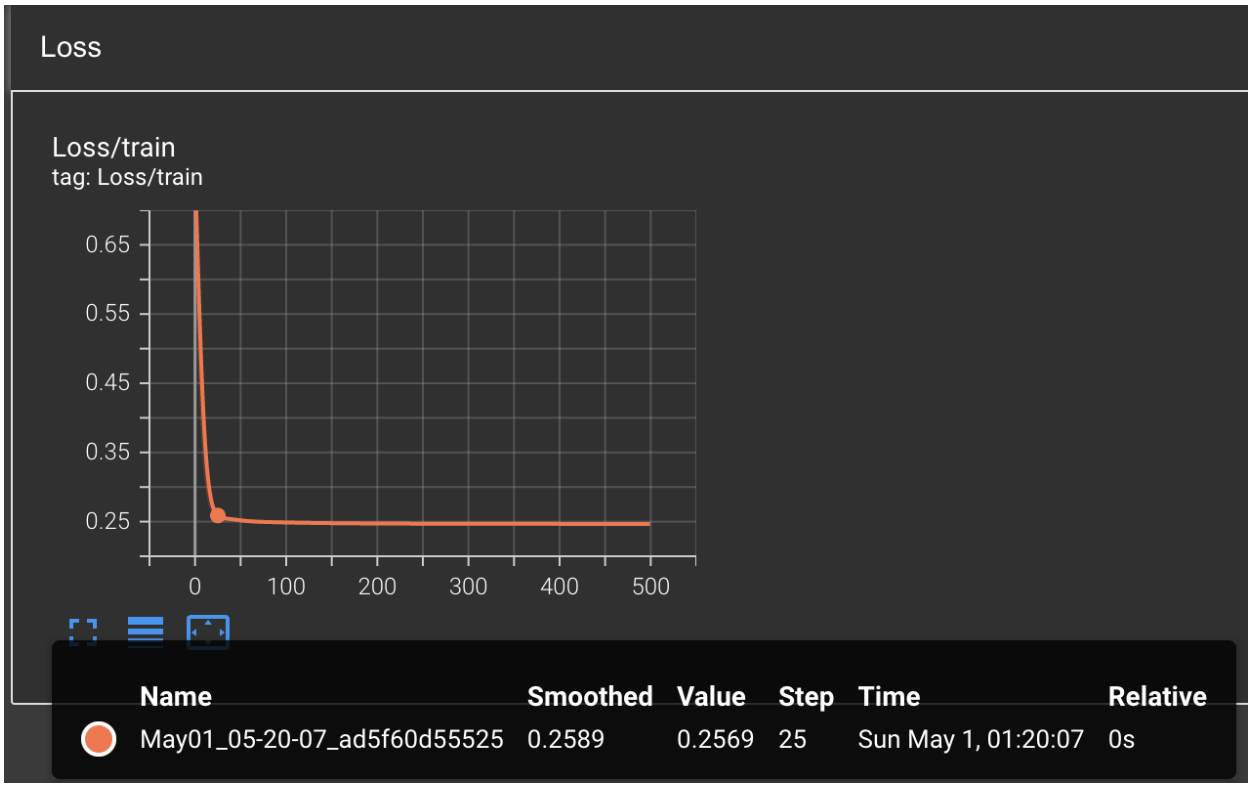
TensorBoard Results

Single Layer Neural Network Model results:



We can clearly notice that from the tensor board results that the loss for the train data has been converging at 300(approx) epoch out of 500 epochs.

Multi layer Neural Network Model results below:



We can clearly notice that from the tensor board results that the gradual decrease in loss for the train data during intial epochs. Thereafter the loss has been converged.

Problems faced

The problem encountered apart from the accuracy of the model include:

- SVM model was unable to run on the cpu as it is running forever and resulting in kernel crash.
- An unstable platform for running Machine Learning Models and collaboration.
- Long running models and system crash was the one of the biggest challenge we faced during training the model.
- Resampling techniques didn't produce good results with ensemble models.
- Identifying the accurate no of hidden layers for multi layer neural network model.

Conclusion

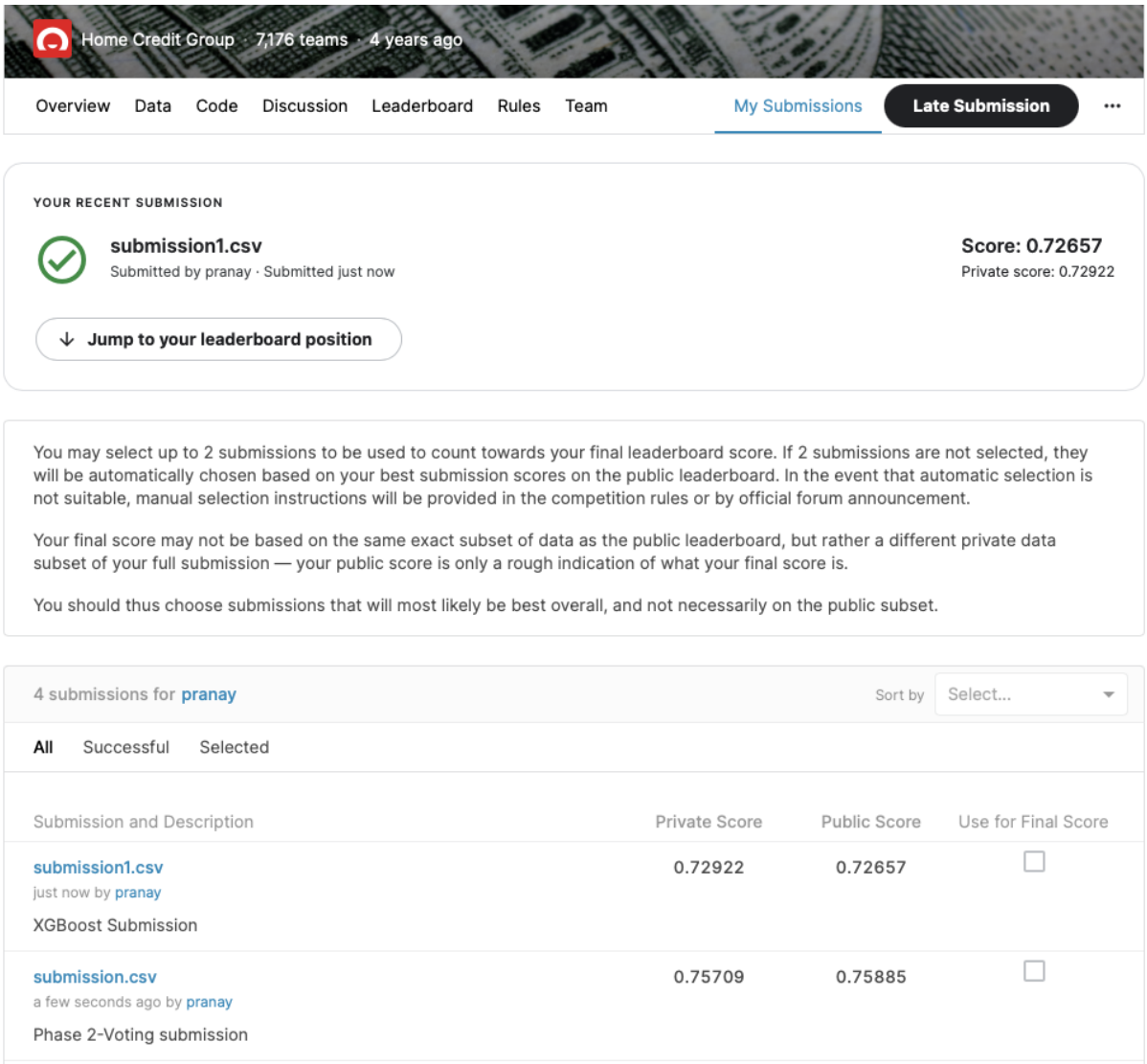
Our implementation using ML models to predict if an applicant will be able to repay a loan was successful. Extending from the phase-1's simple baseline model, data modelling with feature aggregation, feature engineering, and using various data preprocessing pipeline both increased & reduced efficiency of models. Models used for prediction were Logistic Regression , ensemble

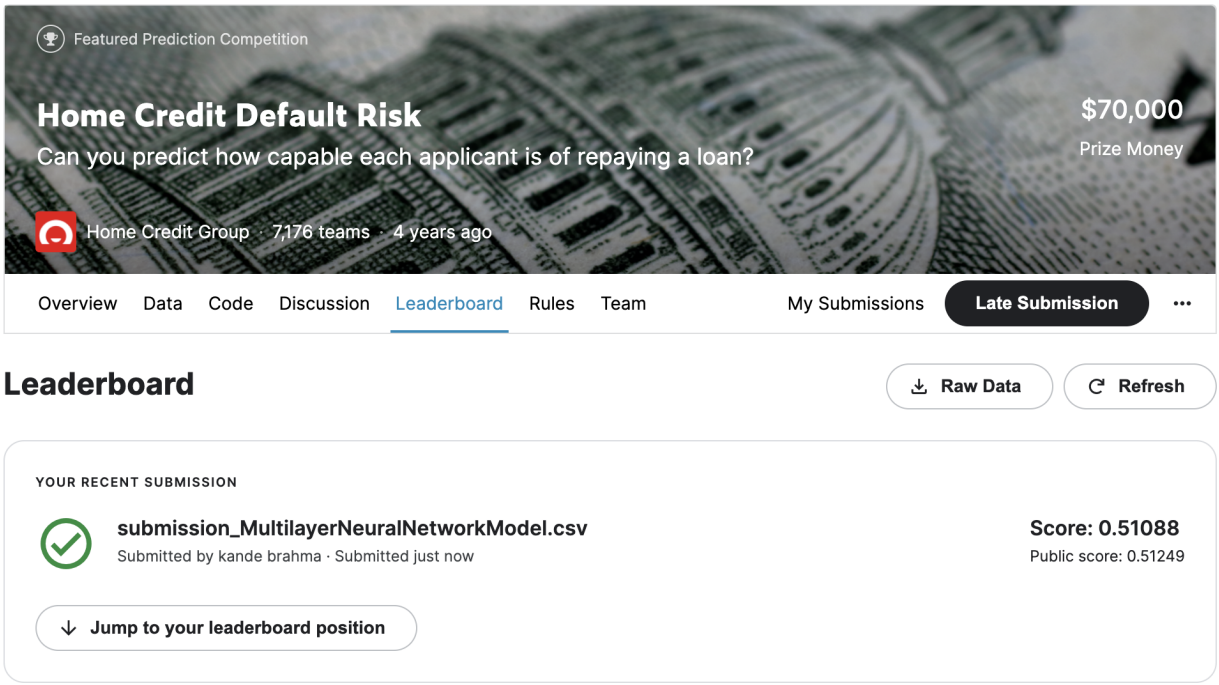
model approaches using gradient boosting, Xgboost, Random forest and SVM. In the current phase we did try to implement Multi layer neural network model using Pytorch.

Our best performing model was XGBoost with the best AUC score of 75.37%, The lowest performing model is Multi layer neural network model with 51.72 % , Our best score in Kaggle submission for XGBoost submission is 0.72922 private and 0.72657 for public and for voting classifier the score is 0.75709 private and 0.75885 for public, However we did believe that Multi layer neural network model would result in higher AUC score, However it has been underperforming compared to traditional models and the AUC score is 0.510 private and 0.512 for public.

Kaggle Submission

Please provide a screenshot of your best kaggle submission for traditional & Multi layer neural network model.





References

Some of the material in this notebook has been adopted from following

https://www.kaggle.com/competitions/home-credit-default-risk/data

https://www.kaggle.com/gemartin/load-data-reduce-memory-usage

https://towardsdatascience.com/a-machine-learning-approach-to-credit-risk-assessment-ba8eda1cd11f

https://medium.com/@dipti.rohan.pawar/correlation-statistical-analysis-9471411f0431

https://scikit-learn.org/stable/modules/generated/sklearn.utils.resample.html

https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction/notebook

https://juhiramzai.medium.com/introduction-to-credit-risk-modeling-e589d6914f57

https://scikit-learn.org/stable/modules/generated/sklearn.utils.resample.html

https://www.analyticsvidhya.com/blog/2020/10/7-feature-engineering-techniques-machine-learning/

https://www.analyticsvidhya.com/blog/2020/03/google-colab-machine-learning-deep-learning/

https://stackify.com/python-garbage-collection/

https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/

https://towardsdatascience.com/5-smote-techniques-for-oversampling-your-imbalance-data-b8155bdbe2b5

https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/

https://towardsdatascience.com/data-leakage-in-machine-learning-