Applied Machine Learning Final Project Home Credit Default Risk

Group-06-HCDR

Durga Sai Sailesh Chodabattula, Manideep Varma Penumatsa, Sahith Reddy Peddireddy, Madhumitha Reddy Muduganti.

Team

Durga Sai Sailesh Chodabattula
dchodaba@iu.edu
Responsibilities: Feature Engineering, Pipelines, Training Models.

Manideep Varma Penumatsa
mpenumat@iu.edu
Responsibilities: Data Preprocessing, Pipelines, Testing.

Madhumitha Reddy Muduganti
mamudu@iu.edu
Responsibilities: Exploratory Data Analysis, Feature Engineering.

Sahith Reddy Peddireddy
speddir@iu.edu
Responsibilities: Exploratory Data Analysis, Pipelines.

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

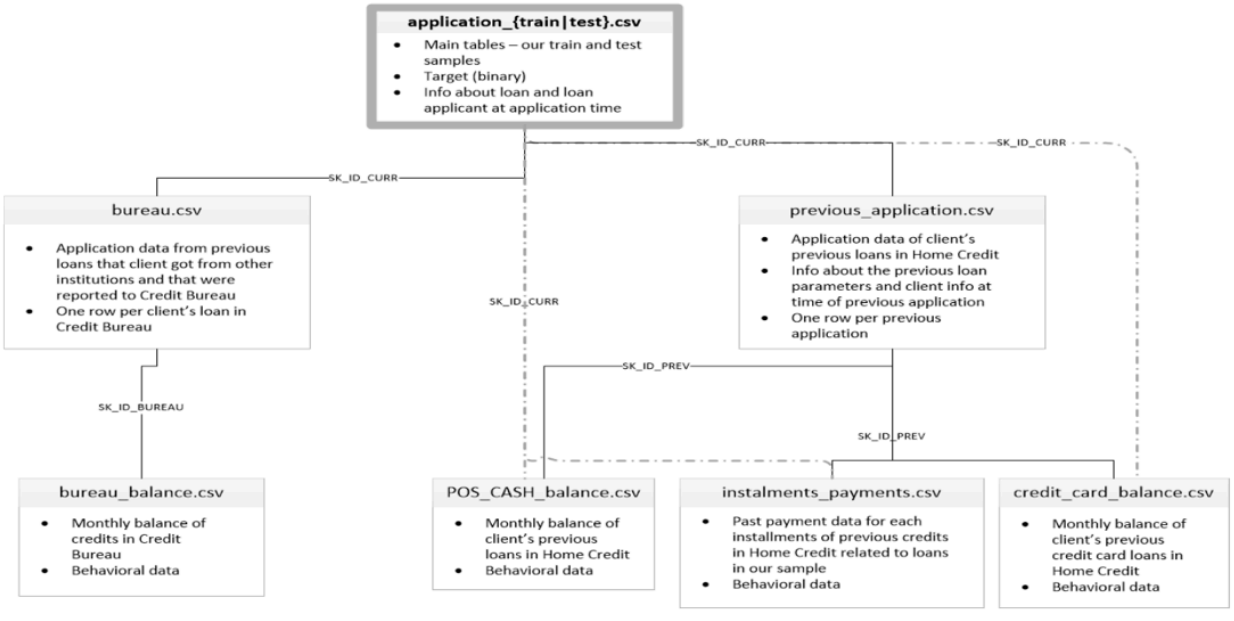
Absract

We will be working on the Home Credit Default Risk problem for our Final Project. Due to the lack of credit history, many people struggle to get loans. This project aims to ensure that the clients who can pay back their loans are not rejected. It will also enable Home Credit to give its clients a better repayment experience. To make sure that these individuals have a positive experience with their loans, Home Credit uses various data sources such as transactional information, etc. to predict their repayment abilities We propose that we evaluate a few Machine learning models, such as logistic regression, XGBClassifier, and Neural Network, and determine the best model for the prediction.

Since our project aim is to make loans available to customers with no credit histories to that aim, we'll examine the data while keeping in mind that many of the people who take out loans have little to no credit history. Machine Learning Algorithms For this project we plan to use Naive Bayes, Logistic Regression, XGBClassifier/ LGBM, Deep learning.

- Phase 1 Results:
- - We used Logistic Regression as Baseline model which resulted in ROCAUC score 0.7394 and test accuracy of 91.7%.
- Phase 2 Results:
- - We added 11 features and were able to increase our ROC_AUC score from 0.7403 to 0.745 using Logistic Regression.
- - As a part of exploring more models to improve the ROC_AUC, conducted the experiment analysis on Untuned LGBM.
- - We trained and tuned an LGBM model, which was very strong and achieve final Kaggle public score of 0.75073

Data Description:



Application{train|test}.csv contains static data for all applications, Individual loans are being used as the data. A single loan is referenced by a unique loan ID in the Both Train and Test datasets. An individual could have multiple loans, each of these would be distinctive from others in the dataset. Additionally, the data is broken down to other datasets, such as previous applications for persons who presently have loans, credit bureau information about other credits sanctioned for the individual, credit card history etc. These applications would exist as independent rows and would be linked to existing loans via a unique application ID. These applications would have their own row to serve as a reference point. Other datasets, such as POS CASH BALANCE and CREDIT CARD BALANCE, contain monthly data on prior POS or cash loans, as well as credit cards of the individuals with home credit. This can be a useful insight for the prediction on default probability. The other datasets can contain 0, 1, or more documents that correspond to the SK_ID_CURR, which will have to be accounted for in our feature engineering.

Preprocessing:

- We plan to use imputer to deal with the missing data. For numerical missing values we can fill mean or median values and for categorical data we can use most frequent label to fill in the missing values.
- We plan to scale our data using standardScaler for numerical data and OneHotEncoder for categorical data. And then pass the data to model and then evaluated for the best model.

In [ ]:
```
# Download Data
```

In [ ]:
```
For Google Colab. If not on Colab, make sure kaggle.json is it the rig
rom google.colab import files

upload kaggle.json
ploaded = files.upload()
```

Choose Files no files selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

In [ ]:
```
!pip install -q kaggle
!ls
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/kaggle.json
```

```
kaggle.json  sample_data
```

In [ ]:
```
!kaggle competitions download home-credit-default-risk -p "home-credit
```

```
Warning: Your Kaggle API key is readable by other users on this syste
m! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading home-credit-default-risk.zip to home-credit-default-risk
100% 687M/688M [00:06<00:00, 148MB/s]
100% 688M/688M [00:06<00:00, 111MB/s]
```

Loading Datasets

In [ ]:
```
import numpy as np
import pandas as pd
import os
import zipfile
import warnings
warnings.filterwarnings('ignore')

def load_data(in_path, name):
    df = pd.read_csv(in_path)
    print("DataSet:",f'{name}')
    print(f"shape is {df.shape}")
    print(df.info())
    display(df.head(3))
    return df

datasets={}
DATA_DIR = "home-credit-default-risk"

ds_names = ("application_train", "application_test", "bureau","bureau_
            "previous_application","POS_CASH_balance")

for ds_name in ds_names:
    datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.c
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,},
```

```
DataSet: application_train
shape is (307511, 122)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
None
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_C
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	

3 rows × 122 columns

```
DataSet: application_test
```

In [ ]:
```
datasets.keys()
```

Out[8]:
```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau
_balance', 'credit_card_balance', 'installments_payments', 'previous_
application', 'POS_CASH_balance'])
```

Analyse Data

In [ ]:
```
datasets["application_train"]['TARGET'].astype(int).plot.hist(title="T
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2eff785490>
```

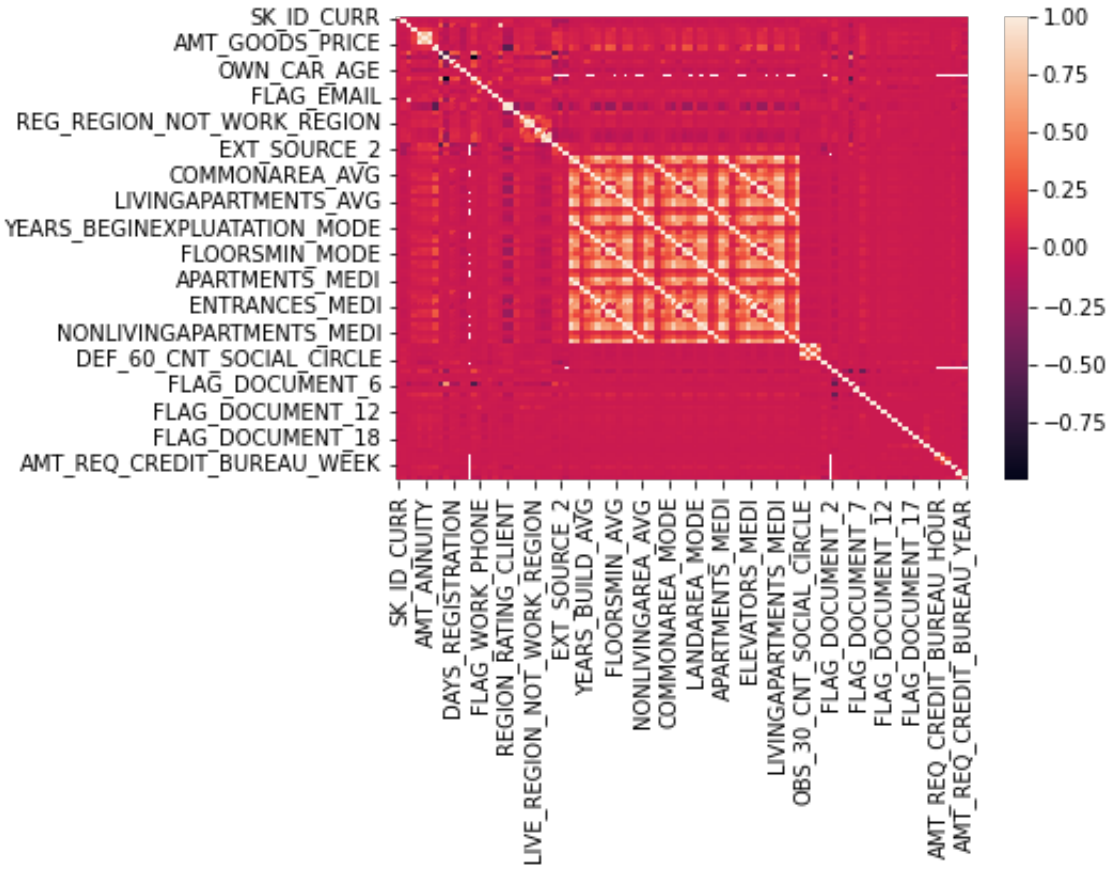
Our Data IS heavily unbalanced. It is very hard to trace relations between the data.

Lets see the correlations

In [ ]:
```
correl = datasets["application_train"].corr()['TARGET'].sort_values()
print('10 Most +ve Correlations:\n', correl.tail(10))
print('\n10 Most -ve Correlations:\n', correl.head(10))
```

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

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

In [ ]:
```
import seaborn as sns
sns.heatmap(datasets['application_train'].corr());
```



Let's segerate numerical and categorical features and analyse

The list was created after looking at the data type of the data set.

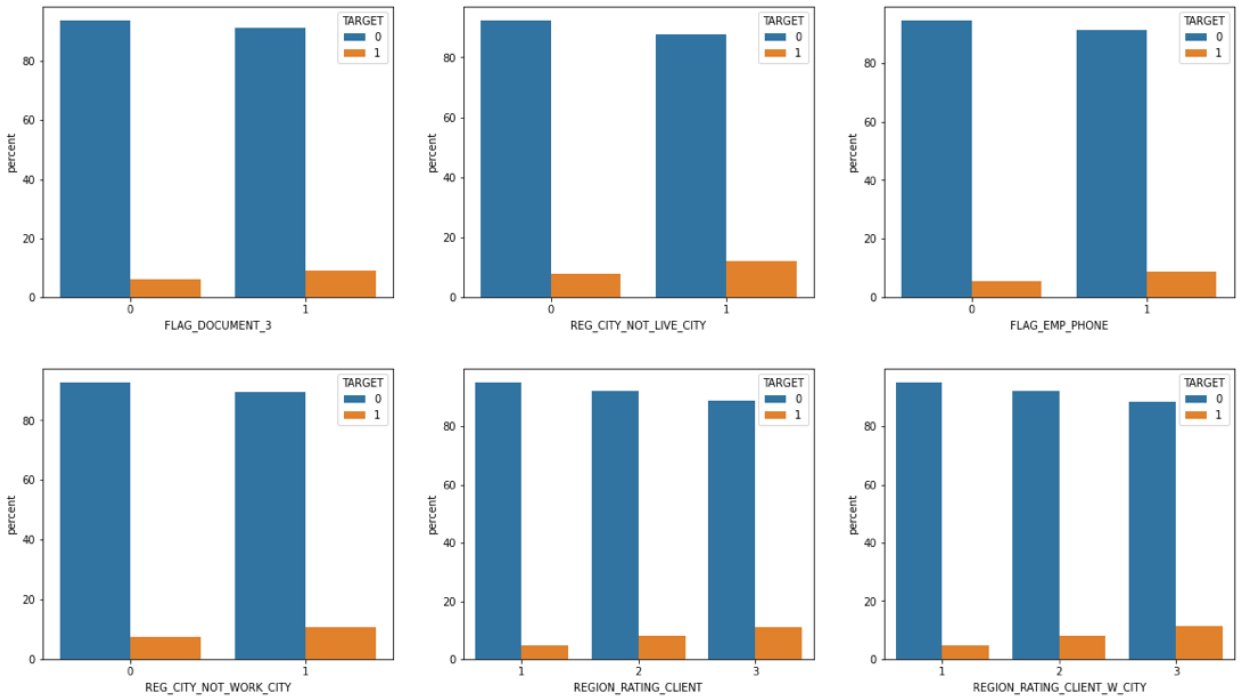
In [ ]:
```
allCatFeatures = [
  "NAME_CONTRACT_TYPE", "CODE_GENDER", "FLAG_OWN_CAR", "NAME_TYPE_SUIT
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "NAME_FAMILY_STATUS", "NA
  "FLAG_MOBIL", "FLAG_EMP_PHONE", "FLAG_WORK_PHONE", "FLAG_CONT_MOBILE
  "FLAG_PHONE", "FLAG_EMAIL", "OCCUPATION_TYPE", "REGION_RATING_CLIENT
  "WEEKDAY_APPR_PROCESS_START", "REG_REGION_NOT_LIVE_REGION", "REG_REG
  "LIVE_REGION_NOT_WORK_REGION", "REG_CITY_NOT_LIVE_CITY", "REG_CITY_N
  "LIVE_CITY_NOT_WORK_CITY", "ORGANIZATION_TYPE", "FONDKAPREMONT_MODE"
  "WALLSMATERIAL_MODE", "EMERGENCYSTATE_MODE", "FLAG_DOCUMENT_2", "FLA
  "FLAG_DOCUMENT_4", "FLAG_DOCUMENT_5", "FLAG_DOCUMENT_6", "FLAG_DOCUM
  "FLAG_DOCUMENT_9", "FLAG_DOCUMENT_10", "FLAG_DOCUMENT_11", "FLAG_DOC
  "FLAG_DOCUMENT_14", "FLAG_DOCUMENT_15", "FLAG_DOCUMENT_16", "FLAG_DO
  "FLAG_DOCUMENT_18", "FLAG_DOCUMENT_19", "FLAG_DOCUMENT_20", "FLAG_DO
  "HOUR_APPR_PROCESS_START"
]
allNumFeatures = [
  "CNT_CHILDREN", "AMT_INCOME_TOTAL", "AMT_CREDIT", "AMT_ANNUITY", "RE
  "DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_REGISTRATION", "OWN_CAR_AGE", "
  "EXT_SOURCE_1", "EXT_SOURCE_2", "EXT_SOURCE_3", "APARTMENTS_AVG", "A
  "BASEMENTAREA_AVG", "YEARS_BEGINEXPLUATATION_AVG", "YEARS_BUILD_AVG"
  "ELEVATORS_AVG", "ENTRANCES_AVG", "FLOORSMAX_AVG", "FLOORSMIN_AVG",
  "LIVINGAPARTMENTS_AVG", "LIVINGAREA_AVG", "NONLIVINGAPARTMENTS_AVG",
  "APARTMENTS_MODE", "BASEMENTAREA_MODE", "YEARS_BEGINEXPLUATATION_MOD
  "COMMONAREA_MODE", "ELEVATORS_MODE", "ENTRANCES_MODE", "FLOORSMAX_MO
  "LANDAREA_MODE", "LIVINGAPARTMENTS_MODE", "LIVINGAREA_MODE", "NONLIV
  "NONLIVINGAREA_MODE", "APARTMENTS_MEDI", "BASEMENTAREA_MEDI", "YEARS
  "YEARS_BUILD_MEDI", "COMMONAREA_MEDI", "ELEVATORS_MEDI", "ENTRANCES_
  "FLOORSMIN_MEDI", "LANDAREA_MEDI", "LIVINGAPARTMENTS_MEDI", "LIVINGA
  "NONLIVINGAPARTMENTS_MEDI", "NONLIVINGAREA_MEDI", "TOTALAREA_MODE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "OBS_60_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_
  "DAYS_LAST_PHONE_CHANGE", "AMT_REQ_CREDIT_BUREAU_HOUR", "AMT_REQ_CRE
  "AMT_REQ_CREDIT_BUREAU_WEEK", "AMT_REQ_CREDIT_BUREAU_MON", "AMT_REQ_
  "AMT_REQ_CREDIT_BUREAU_YEAR", "DAYS_ID_PUBLISH",
]
```

Analysis of categorical features

Let's check if there's an appealing relationship between category and goals. The following graph depicts a percentage rather than an absolute value. This makes the percentage difference, rather than the absolute difference, simple to notice.

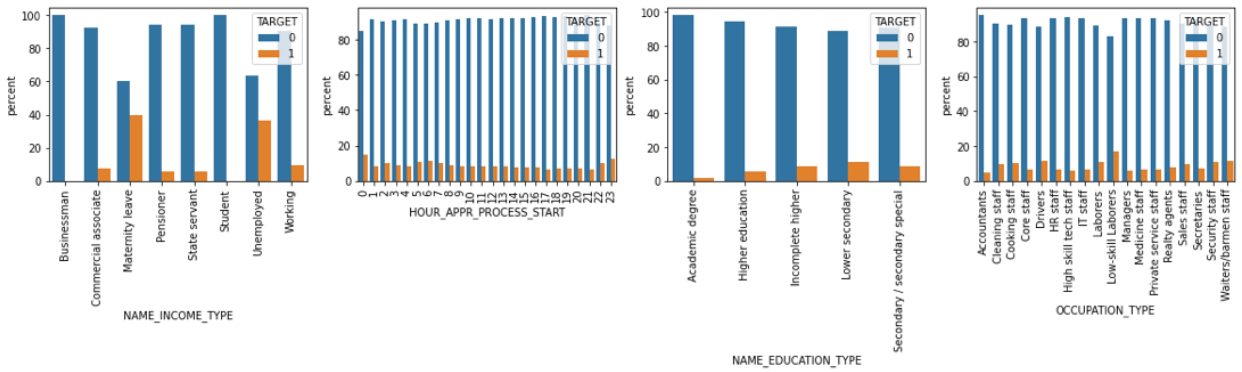
Between Most Correlated features.

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

catCorr1 = [
  "FLAG_DOCUMENT_3", "REG_CITY_NOT_LIVE_CITY", "FLAG_EMP_PHONE",
]

catCorr2 = [
  "REG_CITY_NOT_WORK_CITY", "REGION_RATING_CLIENT", "REGION_RATING_CLI
]

app_train_df = datasets['application_train']
fig = plt.figure(figsize=(20,5))
for idx, cat in enumerate(catCorr1):
  df = app_train_df.groupby(cat)['TARGET'].value_counts(normalize=True
  ax = fig.add_subplot(int("13{}".format(idx+1)))
  sns.barplot(x=cat,y='percent',hue='TARGET',data=df,ax=ax)

fig = plt.figure(figsize=(20,5))
for idx, cat in enumerate(catCorr2):
  df = app_train_df.groupby(cat)['TARGET'].value_counts(normalize=True
  ax = fig.add_subplot(int("13{}".format(idx+1)))
  sns.barplot(x=cat,y='percent',hue='TARGET',data=df,ax=ax)
```



The interesting and most obvious splits here seem to be REGION_RATING_CLIENT and REG_CITY_NOT_WORK_CITY. In on our most correlated fields there is nothing much for us to infer.

In [ ]:
```
at_corr_features_i = [
"NAME_INCOME_TYPE", "HOUR_APPR_PROCESS_START", "NAME_EDUCATION_TYPE",


pp_train_df = datasets['application_train']
ig = plt.figure(figsize=(20,3))
or idx, cat in enumerate(cat_corr_features_i):
df = app_train_df.groupby(cat)['TARGET'].value_counts(normalize=True).
ax = fig.add_subplot(int("14{}".format(idx+1)))
plt.xticks(rotation=90)
sns.barplot(x=cat,y='percent',hue='TARGET',data=df,ax=ax)
```



Observations:

- According to `NAME_INCOME_TYPE` , Businessmen and Students almost never have problems with repayment,but those on maternity leave or unemployed are more likely to have difficulty in repaying the loan.
- According to `HOUR_APPR_PROCESS_START` , Those that file late or early in the day have more difficulty repaying. This pattern decreases over the afternoon, but resumes after 11 p.m. There appears to be a substantial drop about 6 p.m., which could be due to persons who work a 9-5 starting their application after work hours.
- `NAME_EDUCATION_TYPE` also shows some clear trends. The more educated a borrower is, the more difficult it is expected for them to repay the debt.
- This is interesting, as according to `NAME_INCOME_TYPE` , Students almost never have problem repaying.
- `OCCUPATION_TYPE` shows some trends. Low-skill laborers are the most likely to experience repayment issues, while Accountants are the least likely. Drivers, laborers, and waiters/bartenders have a higher rate of default than the general population.

Missing Values.

In [ ]:
```
df = datasets['application_train'][allCatFeatures]
percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(asce
sum_missing = df.isna().sum().sort_values(ascending = False)
missing_data  = pd.concat([percent, sum_missing], axis=1, keys=['Perce
missing_data.head(8)
```

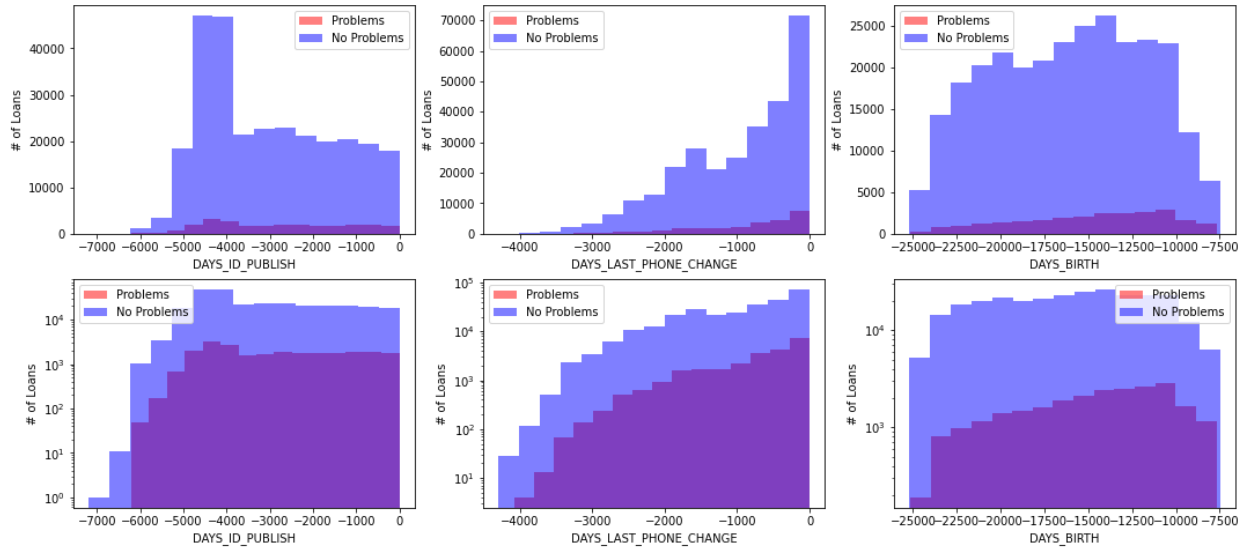
Out[20]:

	Percent	Missing Count
FONDKAPREMONT_MODE	68.39	210295
WALLSMATERIAL_MODE	50.84	156341
HOUSETYPE_MODE	50.18	154297
EMERGENCYSTATE_MODE	47.40	145755
OCCUPATION_TYPE	31.35	96391
NAME_TYPE_SUITE	0.42	1292
FLAG_DOCUMENT_10	0.00	0
FLAG_DOCUMENT_2	0.00	0

Only a few features are lacking, and except for OCCUPATION TYPE, none of them appear to be really interesting. This is most likely why, considering unemployment was not reported.

Analysis of Numerical Features

In [ ]:
```
def target_hist(y, ax, log=False):
  if log: ax.set_yscale('log')
  df = datasets['application_train']

  ax.hist(df[df["TARGET"]==1][y], bins=15, alpha=0.5, color="red", lab
  ax.hist(df[df["TARGET"]==0][y], bins=15, alpha=0.5, color="blue", la

  ax.set_xlabel(y)
  ax.set_ylabel("# of Loans")
  ax.legend()

fig, axs = plt.subplots(2, 3, figsize=(18, 8))
target_hist("DAYS_ID_PUBLISH", axs[0,0])
target_hist("DAYS_LAST_PHONE_CHANGE", axs[0,1])
target_hist("DAYS_BIRTH", axs[0,2])
# log graphs
target_hist("DAYS_ID_PUBLISH", axs[1,0], True)
target_hist("DAYS_LAST_PHONE_CHANGE", axs[1,1], True)
target_hist("DAYS_BIRTH", axs[1,2], True)
```

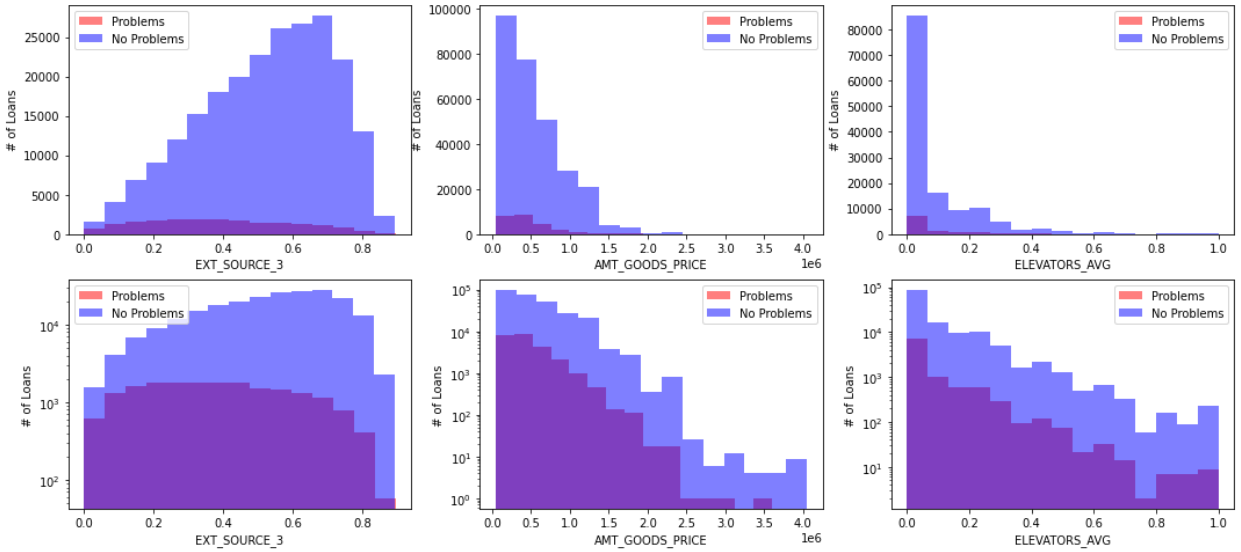


NOTE: For three separate features, these are the linear and log graphs. The log graph is located just beneath the linear graph and is used to aid in the reading and interpretation of data.

There are several intriguing characteristics to be found here. It appears that the more recently a customer replaced their phone, the more probable there will be payback issues.

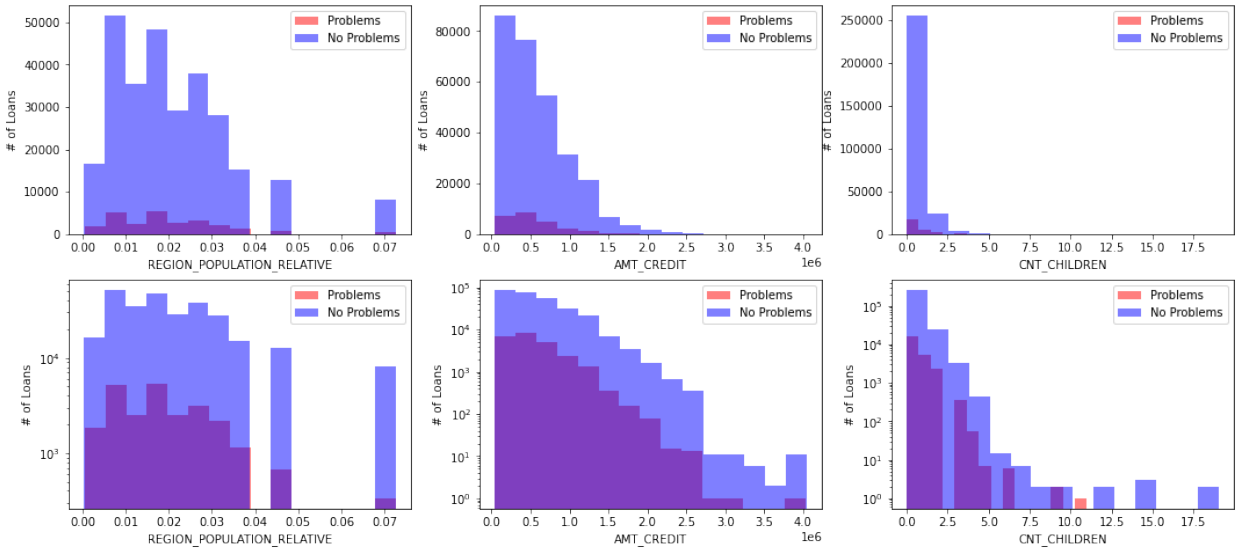
The client's age has some unique characteristics as well. When the client is roughly 11,000 days old, or around 30 years old, the main difficulties appear to occur. The more capable they are of repaying the loan, the older or younger they are.

In [ ]:
```
fig, axs = plt.subplots(2, 3, figsize=(18, 8))
target_hist("EXT_SOURCE_3", axs[0,0])
target_hist("AMT_GOODS_PRICE", axs[0,1])
target_hist("ELEVATORS_AVG", axs[0,2])
# log
target_hist("EXT_SOURCE_3", axs[1,0], True)
target_hist("AMT_GOODS_PRICE", axs[1,1], True)
target_hist("ELEVATORS_AVG", axs[1,2], True)
```



- The data field 'EXT SOURCE 3' has the strongest negative correlation. It's data from an external source with no specific labels, so drawing intuitive connections isn't really fascinating.
- 'AMT GOODS PRICE' is strongly associated with repayment issues. Although the description is ambiguous, it's possible that this relates to the loan's collateral.
- Interestingly, the more elevators in the building where the client lives, the more likely they are to have trouble repaying the loan, according to 'ELEVATORS AVG'. The target is also substantially connected with the number of floors, indicating that it is most likely related to rent pricing or neighborhood quality.

In [ ]:
```
fig, axs = plt.subplots(2, 3, figsize=(18, 8))
target_hist("REGION_POPULATION_RELATIVE", axs[0,0])
target_hist("AMT_CREDIT", axs[0,1])
target_hist("CNT_CHILDREN", axs[0,2])
# log graphs
target_hist("REGION_POPULATION_RELATIVE", axs[1,0], True)
target_hist("AMT_CREDIT", axs[1,1], True)
target_hist("CNT_CHILDREN", axs[1,2], True)
```



- The larger the population in which the customer lives, the better he seems to be able to repay the loan. This may be because it is more expensive to live in a populated area.
- You can see that the more people you want to borrow, the less repayment problems you have.
- The more children a customer has, the harder it is to repay the loan without problems.

Missing Values

In [ ]:
```
df = datasets['application_train'][allNumFeatures]
percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(asce
sum_missing = df.isna().sum().sort_values(ascending = False)
missing_data  = pd.concat([percent, sum_missing], axis=1, keys=['Perce
missing_data.head(54)
```

Out[24]:

	Percent	Missing Count
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865
COMMONAREA_MODE	69.87	214865
NONLIVINGAPARTMENTS_AVG	69.43	213514
NONLIVINGAPARTMENTS_MEDI	69.43	213514

NONLIVINGAPARTMENTS_MODE	69.43	213514
LIVINGAPARTMENTS_MEDI	68.35	210199
LIVINGAPARTMENTS_AVG	68.35	210199
LIVINGAPARTMENTS_MODE	68.35	210199
FLOORSMIN_MEDI	67.85	208642
FLOORSMIN_AVG	67.85	208642
FLOORSMIN_MODE	67.85	208642
YEARS_BUILD_MEDI	66.50	204488
YEARS_BUILD_AVG	66.50	204488
YEARS_BUILD_MODE	66.50	204488
OWN_CAR_AGE	65.99	202929
LANDAREA_AVG	59.38	182590
LANDAREA_MODE	59.38	182590
LANDAREA_MEDI	59.38	182590
BASEMENTAREA_MODE	58.52	179943
BASEMENTAREA_MEDI	58.52	179943
BASEMENTAREA_AVG	58.52	179943
EXT_SOURCE_1	56.38	173378
NONLIVINGAREA_MEDI	55.18	169682
NONLIVINGAREA_AVG	55.18	169682
NONLIVINGAREA_MODE	55.18	169682
ELEVATORS_AVG	53.30	163891
ELEVATORS_MEDI	53.30	163891
ELEVATORS_MODE	53.30	163891
APARTMENTS_MEDI	50.75	156061
APARTMENTS_AVG	50.75	156061
APARTMENTS_MODE	50.75	156061
ENTRANCES_MODE	50.35	154828
ENTRANCES_AVG	50.35	154828
ENTRANCES_MEDI	50.35	154828
LIVINGAREA_AVG	50.19	154350
LIVINGAREA_MODE	50.19	154350
LIVINGAREA_MEDI	50.19	154350
FLOORSMAX_MODE	49.76	153020

FLOORSMAX_MEDI	49.76	153020
FLOORSMAX_AVG	49.76	153020
YEARS_BEGINEXPLUATATION_MEDI	48.78	150007
YEARS_BEGINEXPLUATATION_MODE	48.78	150007
YEARS_BEGINEXPLUATATION_AVG	48.78	150007
TOTALAREA_MODE	48.27	148431
EXT_SOURCE_3	19.83	60965
AMT_REQ_CREDIT_BUREAU_HOUR	13.50	41519
AMT_REQ_CREDIT_BUREAU_WEEK	13.50	41519
AMT_REQ_CREDIT_BUREAU_YEAR	13.50	41519
AMT_REQ_CREDIT_BUREAU_QRT	13.50	41519
AMT_REQ_CREDIT_BUREAU_MON	13.50	41519
AMT_REQ_CREDIT_BUREAU_DAY	13.50	41519
OBS_30_CNT_SOCIAL_CIRCLE	0.33	1021
DEF_30_CNT_SOCIAL_CIRCLE	0.33	1021

- Unfortunately, more than half of the data lacks many interesting features. Ideally, you want to avoid relying on this data.
- `EXT_SOURCE_1` , like the information about the floor of the customer's house, is highly correlated with the destination, but is missing in about half of the rows, so this type of data can be heavily biased.
- `EXT_SOURCE_3` is also missing about 20% of the lines, which is bad for analysis.

Analyse Secondary Datasets

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

ome generalized functions to make the analysis easy on us
 cat_bar(df, x, ax):
f2 = df.groupby(x)['TARGET'].value_counts(normalize=True).mul(100).ren
ns.barplot(x=x,y='percent',hue='TARGET',data=df2,ax=ax)

 num_hist(df, y, ax, log=False):
f log: ax.set_yscale('log')

x.hist(df[df["TARGET"]==1][y], bins=25, alpha=0.5, color="red", label=
x.hist(df[df["TARGET"]==0][y], bins=25, alpha=0.5, color="green", labe

x.set_xlabel(y)
x.set_ylabel("No. of Loans")
x.legend()

 missing_vals(df):
ercent = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascen
um_missing = df.isna().sum().sort_values(ascending = False)
issing_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percen
eturn missing_data.head(10)
```

- For every other dataset, we groupped the data by SK_ID_CURR and performed some aggr fucntions and then merged onto the application train data.
- Also, analysed the dataset for the trends or insights.
- The process for repeated for all the secondary datasets.

Credit Card Balances

This dataset is month-to-month credit card balances, with one row being one month

In [ ]:
```
datasets['credit_card_balance'].columns
```

Out[27]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'MONTHS_BALANCE', 'AMT_BALANCE',
       'AMT_CREDIT_LIMIT_ACTUAL', 'AMT_DRAWINGS_ATM_CURRENT',
       'AMT_DRAWINGS_CURRENT', 'AMT_DRAWINGS_OTHER_CURRENT',
       'AMT_DRAWINGS_POS_CURRENT', 'AMT_INST_MIN_REGULARITY',
       'AMT_PAYMENT_CURRENT', 'AMT_PAYMENT_TOTAL_CURRENT',
       'AMT_RECEIVABLE_PRINCIPAL', 'AMT_RECIVABLE', 'AMT_TOTAL_RECEIV
ABLE',
       'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_CURRENT',
       'CNT_DRAWINGS_OTHER_CURRENT', 'CNT_DRAWINGS_POS_CURRENT',
       'CNT_INSTALMENT_MATURE_CUM', 'NAME_CONTRACT_STATUS', 'SK_DPD',
       'SK_DPD_DEF'],
      dtype='object')
```

In [ ]:
```
datasets['credit_card_balance'].info()
datasets['credit_card_balance'].describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column                      Dtype  
---  ------                      -----  
 0   SK_ID_PREV                  int64  
 1   SK_ID_CURR                  int64  
 2   MONTHS_BALANCE              int64  
 3   AMT_BALANCE                 float64
 4   AMT_CREDIT_LIMIT_ACTUAL     int64  
 5   AMT_DRAWINGS_ATM_CURRENT    float64
 6   AMT_DRAWINGS_CURRENT        float64
 7   AMT_DRAWINGS_OTHER_CURRENT  float64
 8   AMT_DRAWINGS_POS_CURRENT    float64
 9   AMT_INST_MIN_REGULARITY     float64
 10  AMT_PAYMENT_CURRENT         float64
 11  AMT_PAYMENT_TOTAL_CURRENT   float64
 12  AMT_RECEIVABLE_PRINCIPAL    float64
 13  AMT_RECIVABLE               float64
 14  AMT_TOTAL_RECEIVABLE        float64
 15  CNT_DRAWINGS_ATM_CURRENT    float64
 16  CNT_DRAWINGS_CURRENT        int64  
 17  CNT_DRAWINGS_OTHER_CURRENT  float64
 18  CNT_DRAWINGS_POS_CURRENT    float64
 19  CNT_INSTALMENT_MATURE_CUM   float64
 20  NAME_CONTRACT_STATUS        object 
 21  SK_DPD                      int64  
 22  SK_DPD_DEF                  int64  
dtypes: float64(15), int64(7), object(1)
memory usage: 673.9+ MB
```

Out[28]:

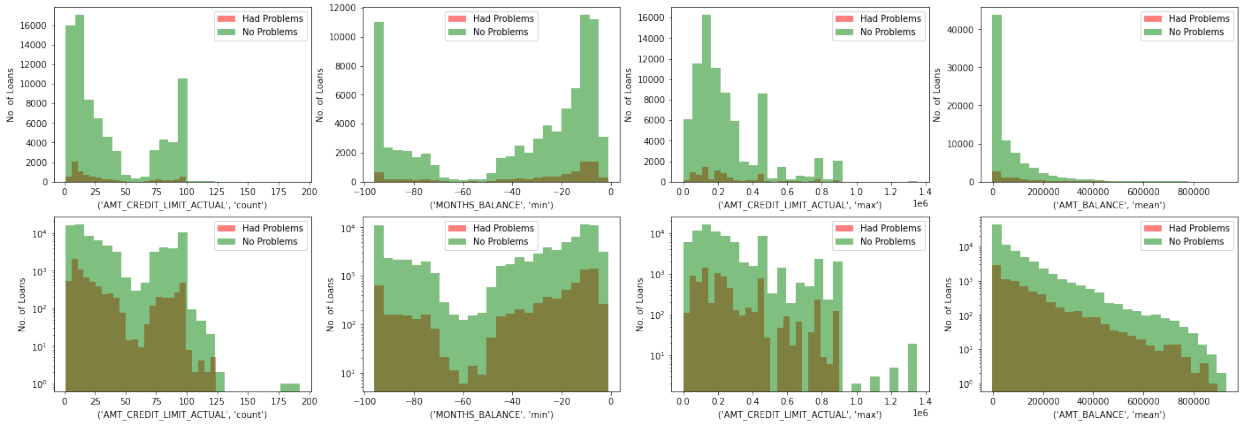
	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_A
count	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.8403
mean	1.904504e+06	2.783242e+05	-3.452192e+01	5.830016e+04	1.5380
std	5.364695e+05	1.027045e+05	2.666775e+01	1.063070e+05	1.6514
min	1.000018e+06	1.000060e+05	-9.600000e+01	-4.202502e+05	0.0000
25%	1.434385e+06	1.895170e+05	-5.500000e+01	0.000000e+00	4.5000
50%	1.897122e+06	2.783960e+05	-2.800000e+01	0.000000e+00	1.1250
75%	2.369328e+06	3.675800e+05	-1.100000e+01	8.904669e+04	1.8000
max	2.843496e+06	4.562500e+05	-1.000000e+00	1.505902e+06	1.3500

8 rows × 22 columns

In [ ]:
```
CCB_df = datasets['credit_card_balance'].groupby('SK_ID_CURR').agg({
    "AMT_BALANCE":["mean"],
    "MONTHS_BALANCE": ["min"],
    "AMT_CREDIT_LIMIT_ACTUAL": ["max", "count"],
    "CNT_INSTALMENT_MATURE_CUM": ["max"],
    "AMT_PAYMENT_CURRENT": ["mean"]
})

CCB_df[("AMT_PAYMENT_CURRENT", "mean")].fillna(value=0, inplace=True)
CCB_df['Payment_balance_ratio'] = CCB_df[("AMT_PAYMENT_CURRENT", "mean
```

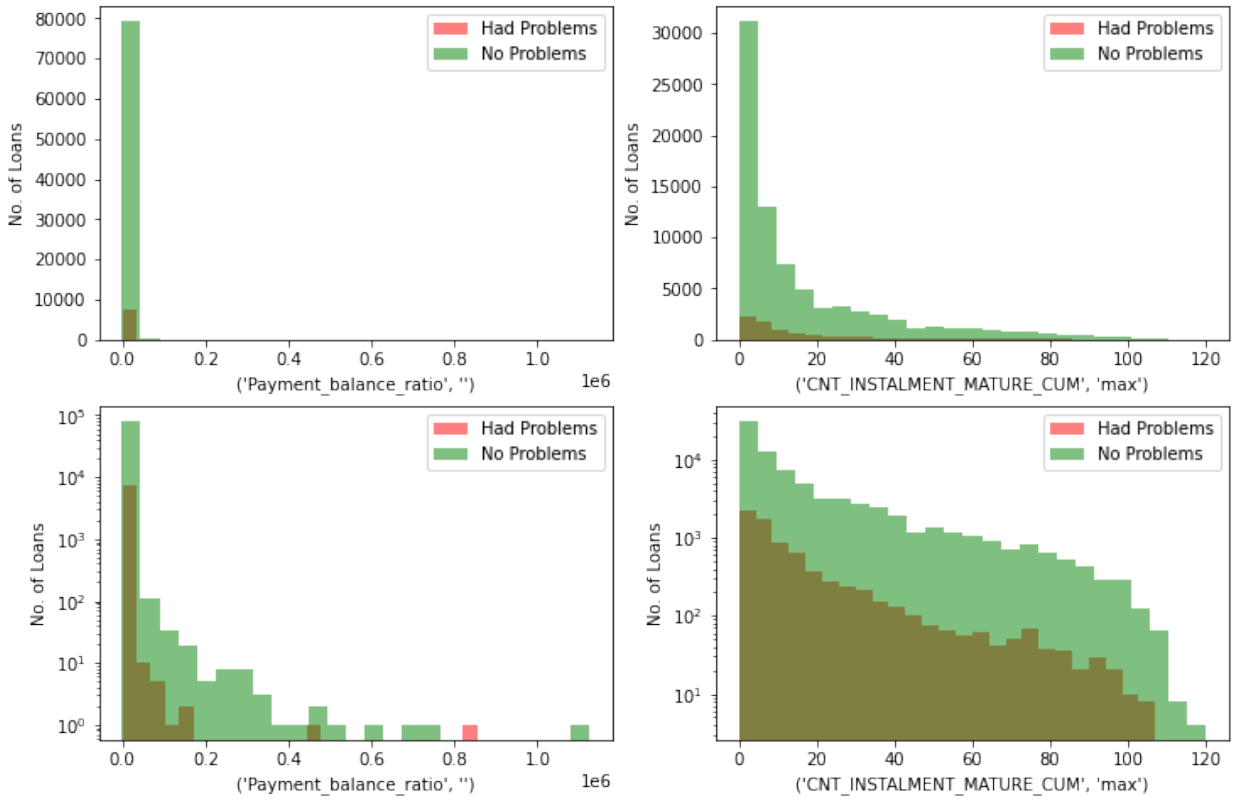
In [ ]:
```
tmp = datasets['application_train'].merge(CCB_df, how='left', on='SK_I
```

In [ ]:
```
fig, axs = plt.subplots(2, 4, figsize=(24, 8))
num_hist(tmp, ('AMT_CREDIT_LIMIT_ACTUAL', 'count'), axs[0,0])
num_hist(tmp, ('MONTHS_BALANCE', 'min'), axs[0,1])
num_hist(tmp, ('AMT_CREDIT_LIMIT_ACTUAL', 'max'), axs[0,2])
num_hist(tmp, ('AMT_BALANCE', 'mean'), axs[0,3])
# log graphs
num_hist(tmp, ('AMT_CREDIT_LIMIT_ACTUAL', 'count'), axs[1,0], True)
num_hist(tmp, ('MONTHS_BALANCE', 'min'), axs[1,1], True)
num_hist(tmp, ('AMT_CREDIT_LIMIT_ACTUAL', 'max'), axs[1,2], True)
num_hist(tmp, ('AMT_BALANCE', 'mean'), axs[1,3], True)
```



- It appears that customers with the highest credit limits have very few refund issues, which is to be expected. The number of credit card equalizations appears to be in the shape of a "U". However, those few people who are in extremely high positions have no problem paying the advance.
- It's hard to say, but `AMT_BALANCE` may also have something to do with the target.

In [ ]:
```
fig, axs = plt.subplots(2, 2, figsize=(12, 8))
num_hist(tmp, ("Payment_balance_ratio", ''), axs[0,0])
num_hist(tmp, ('CNT_INSTALMENT_MATURE_CUM', 'max'), axs[0,1])
# log graphs
num_hist(tmp, ("Payment_balance_ratio", ''), axs[1,0], True)
num_hist(tmp, ('CNT_INSTALMENT_MATURE_CUM', 'max'), axs[1,1], True)
```



- The `Payment_balance_ratio` all seem to be concentrated around 0.0, so there is little predictive power.
- `CNT_INSTALMENT_MATURE_CUM` seems to have a pretty good correlation. This makes sense. The more installments a client pays to a credit card, the less likely it is that the loan will have problems.

Correlations

In [ ]:
```
tmp[list(CCB_df.columns) + ["TARGET"]].corr()["TARGET"]
```

Out[33]:
```
(AMT_BALANCE, mean)                 0.087177
(MONTHS_BALANCE, min)               0.061359
(AMT_CREDIT_LIMIT_ACTUAL, max)     -0.011679
(AMT_CREDIT_LIMIT_ACTUAL, count)   -0.060481
(CNT_INSTALMENT_MATURE_CUM, max)   -0.017568
(AMT_PAYMENT_CURRENT, mean)         0.028224
(Payment_balance_ratio, )           0.000719
TARGET                              1.000000
Name: TARGET, dtype: float64
```

- These are some strong correlation values. `AMT_BALANCE` refers to your credit card balance. Therefore, it makes sense that the higher the balance, the harder it is to repay the loan.
- Since the balance is `MONTHS_BALANCE`, the closer you are to the application time, the more difficult it will be to repay.

Missing Values

In [ ]:
```
missing_vals(tmp[CCB_df.columns])
```

Out[36]:

	Percent	Missing Count
(AMT_BALANCE, mean)	71.74	220606
(MONTHS_BALANCE, min)	71.74	220606
(AMT_CREDIT_LIMIT_ACTUAL, max)	71.74	220606
(AMT_CREDIT_LIMIT_ACTUAL, count)	71.74	220606
(CNT_INSTALMENT_MATURE_CUM, max)	71.74	220606
(AMT_PAYMENT_CURRENT, mean)	71.74	220606
(Payment_balance_ratio,)	71.74	220606

Previous Applications

In [ ]:
```
datasets['previous_application'].columns
```

Out[37]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY
',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOOD
S_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_VER
SION',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVA
L'],
      dtype='object')
```

In [ ]:
```
datasets['previous_application'].info()
datasets['previous_application'].describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 16  NAME_CONTRACT_STATUS         1670214 non-null  object
 17  DAYS_DECISION                1670214 non-null  int64
 18  NAME_PAYMENT_TYPE            1670214 non-null  object
 19  CODE_REJECT_REASON           1670214 non-null  object
 20  NAME_TYPE_SUITE              849809 non-null   object
 21  NAME_CLIENT_TYPE             1670214 non-null  object
 22  NAME_GOODS_CATEGORY          1670214 non-null  object
 23  NAME_PORTFOLIO               1670214 non-null  object
```

```
 24  NAME_PRODUCT_TYPE             1670214 non-null  object
 25  CHANNEL_TYPE                  1670214 non-null  object
 26  SELLERPLACE_AREA              1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY          1670214 non-null  object
 28  CNT_PAYMENT                   1297984 non-null  float64
 29  NAME_YIELD_GROUP              1670214 non-null  object
 30  PRODUCT_COMBINATION           1669868 non-null  object
 31  DAYS_FIRST_DRAWING            997149 non-null   float64
 32  DAYS_FIRST_DUE                997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION     997149 non-null   float64
 34  DAYS_LAST_DUE                 997149 non-null   float64
 35  DAYS_TERMINATION              997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL     997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
```

Out[38]:

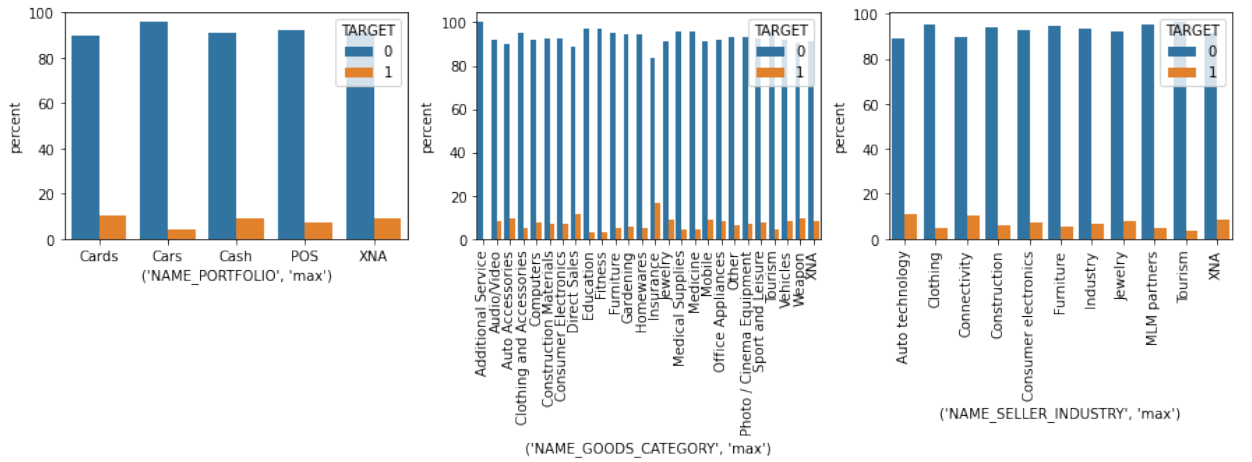
	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DC
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	
min	1.000001e+06	1.000010e+05	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	4.180581e+05	6.905160e+06	6.905160e+06	

8 rows × 21 columns

In [ ]:
```
PA_df = datasets['previous_application'].groupby('SK_ID_CURR').agg({
    "AMT_APPLICATION": ["mean", "count"],
    "NAME_CONTRACT_STATUS": "max",
    "NAME_CLIENT_TYPE": "max",
    "NAME_YIELD_GROUP": "max",
    "DAYS_TERMINATION":"mean",

    "NAME_PORTFOLIO": "max",
    "NAME_GOODS_CATEGORY": "max",
    "NAME_SELLER_INDUSTRY": "max",
    "CNT_PAYMENT": "max",
})
```

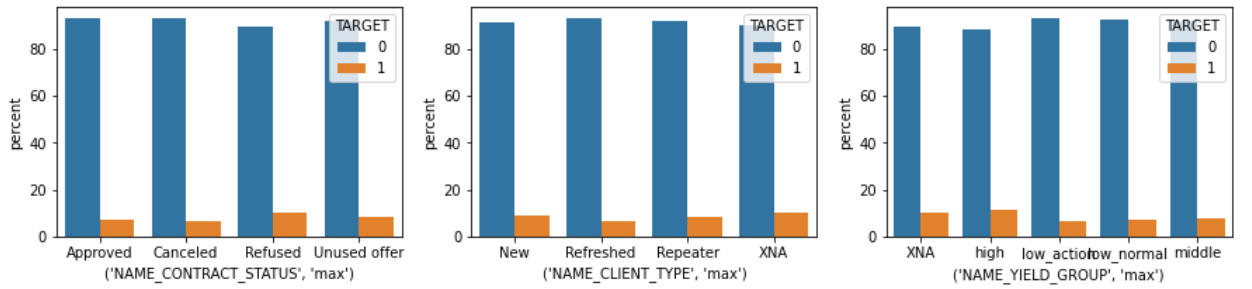
In [ ]:
```
tmp = datasets['application_train'].merge(PA_df, how='left', on='SK_ID
```

In [ ]:
```
fig, axs = plt.subplots(1, 3, figsize=(15, 3))
cat_bar(tmp, ('NAME_PORTFOLIO', 'max'), axs[0])
cat_bar(tmp, ('NAME_GOODS_CATEGORY', 'max'), axs[1])
cat_bar(tmp, ('NAME_SELLER_INDUSTRY', 'max'), axs[2])

for tick in axs[1].get_xticklabels():
  tick.set_rotation(90)
for tick in axs[2].get_xticklabels():
  tick.set_rotation(90)
```



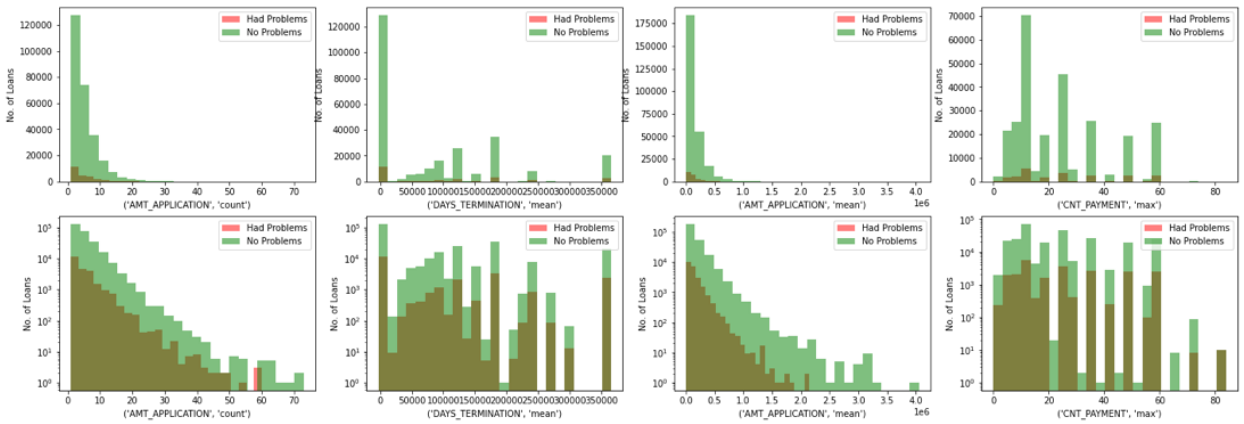
- The NAME_GOODS_CATEGORY , 'Education', 'Fitness', and 'Tourism' have the least amount of repayment problems, while 'Insurance', 'Direct Sales', and 'Weapon' have some of the highest rates of problems.
- NAME_PRODUCT_TYPE , walk-ins have higher rates of problems than the other two.
- NAME_PORTFOLIO , Car loans seem to have the least amount of problems

In [ ]:
```
fig, axs = plt.subplots(1, 3, figsize=(15, 3))
cat_bar(tmp, ('NAME_CONTRACT_STATUS', 'max'), axs[0])
cat_bar(tmp, ('NAME_CLIENT_TYPE', 'max'), axs[1])
cat_bar(tmp, ('NAME_YIELD_GROUP', 'max'), axs[2])
```



- NAME_CONTRACT_STATUS , the Refused clients have a higher probability of having payment issues.
- NAME_CLIENT_TYPE , 'Refreshed' clients have a lower probability of having problems.
- NAME_YIELD_GROUP , the low-yield clients have the fewest amount of problems, and 'XNA' and 'high' yield groups have the most amount of problems.

In [ ]:
```
fig, axs = plt.subplots(2, 4, figsize=(24, 8))
num_hist(tmp, ('AMT_APPLICATION', 'count'), axs[0,0])
num_hist(tmp, ('DAYS_TERMINATION', 'mean'), axs[0,1])
num_hist(tmp, ('AMT_APPLICATION', 'mean'), axs[0,2])
num_hist(tmp, ('CNT_PAYMENT', 'max'), axs[0,3])
# log graphs
num_hist(tmp, ('AMT_APPLICATION', 'count'), axs[1,0], True)
num_hist(tmp, ('DAYS_TERMINATION', 'mean'), axs[1,1], True)
num_hist(tmp, ('AMT_APPLICATION', 'mean'), axs[1,2], True)
num_hist(tmp, ('CNT_PAYMENT', 'max'), axs[1,3], True)
```



It's hard to see, but clients with a large number of applications and demanding higher amounts tend to have more repayment issues.

Correlations

In [ ]:
```
tmp[list(PA_df.columns) + ["TARGET"]].corr()["TARGET"]
```

Out[44]:
```
(AMT_APPLICATION, mean)    -0.021803
(AMT_APPLICATION, count)    0.019762
(DAYS_TERMINATION, mean)    0.025795
(CNT_PAYMENT, max)          0.029439
TARGET                      1.000000
Name: TARGET, dtype: float64
```

Missing Values

In [ ]:
```
missing_vals(tmp[PA_df.columns])
```

Out[45]:

	Percent	Missing Count
(DAYS_TERMINATION, mean)	5.77	17751
(CNT_PAYMENT, max)	5.49	16869
(AMT_APPLICATION, mean)	5.35	16454
(AMT_APPLICATION, count)	5.35	16454
(NAME_CONTRACT_STATUS, max)	5.35	16454
(NAME_CLIENT_TYPE, max)	5.35	16454
(NAME_YIELD_GROUP, max)	5.35	16454
(NAME_PORTFOLIO, max)	5.35	16454
(NAME_GOODS_CATEGORY, max)	5.35	16454
(NAME_SELLER_INDUSTRY, max)	5.35	16454

There's not a lot of missing data here.

Pos Cash Balance

In [ ]:
```
datasets['POS_CASH_balance'].columns
```

Out[47]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'MONTHS_BALANCE', 'CNT_INSTALMENT'
,
       'CNT_INSTALMENT_FUTURE', 'NAME_CONTRACT_STATUS', 'SK_DPD',
       'SK_DPD_DEF'],
      dtype='object')
```

In [ ]:
```
datasets['installments_payments'].info()
datasets['installments_payments'].describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_PREV              int64  
 1   SK_ID_CURR              int64  
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   int64  
 4   DAYS_INSTALMENT         float64
 5   DAYS_ENTRY_PAYMENT      float64
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
```

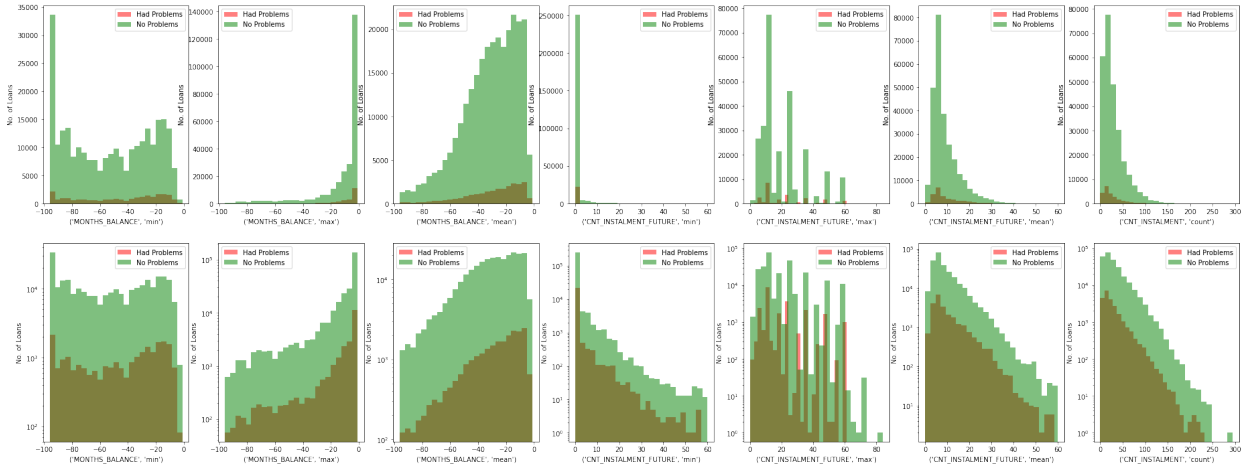
Out[48]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER
count	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07
mean	1.903365e+06	2.784449e+05	8.566373e-01	1.887090e+01
std	5.362029e+05	1.027183e+05	1.035216e+00	2.666407e+01
min	1.000001e+06	1.000010e+05	0.000000e+00	1.000000e+00
25%	1.434191e+06	1.896390e+05	0.000000e+00	4.000000e+00
50%	1.896520e+06	2.786850e+05	1.000000e+00	8.000000e+00
75%	2.369094e+06	3.675300e+05	1.000000e+00	1.900000e+01
max	2.843499e+06	4.562550e+05	1.780000e+02	2.770000e+02

In [ ]:
```
PCB_df = datasets['POS_CASH_balance'].groupby('SK_ID_CURR').agg({
    "CNT_INSTALMENT": ["count"],
    "CNT_INSTALMENT_FUTURE":["max", "min", "mean"],
    "MONTHS_BALANCE":["max","min", "mean"],

})
```

In [ ]:
```
tmp = datasets['application_train'].merge(PCB_df, how='left', on='SK_I
```

In [ ]:
```
fig, axs = plt.subplots(2, 7, figsize=(32, 12))
num_hist(tmp, ("MONTHS_BALANCE", 'min'), axs[0,0])
num_hist(tmp, ("MONTHS_BALANCE", 'max'), axs[0,1])
num_hist(tmp, ("MONTHS_BALANCE", 'mean'), axs[0,2])
num_hist(tmp, ("CNT_INSTALMENT_FUTURE", 'min'), axs[0,3])
num_hist(tmp, ("CNT_INSTALMENT_FUTURE", 'max'), axs[0,4])
num_hist(tmp, ("CNT_INSTALMENT_FUTURE", 'mean'), axs[0,5])
num_hist(tmp, ('CNT_INSTALMENT', 'count'), axs[0,6])
# log graphs
num_hist(tmp, ("MONTHS_BALANCE", 'min'), axs[1,0], True)
num_hist(tmp, ("MONTHS_BALANCE", 'max'), axs[1,1], True)
num_hist(tmp, ("MONTHS_BALANCE", 'mean'), axs[1,2], True)
num_hist(tmp, ("CNT_INSTALMENT_FUTURE", 'min'), axs[1,3], True)
num_hist(tmp, ("CNT_INSTALMENT_FUTURE", 'max'), axs[1,4], True)
num_hist(tmp, ("CNT_INSTALMENT_FUTURE", 'mean'), axs[1,5], True)
num_hist(tmp, ('CNT_INSTALMENT', 'count'), axs[1,6], True)
```



- MONTHS_BALANCE, min are very vaguely "U" shaped, but the difference between problematic and problem-free is uniform throughout. CNT_INSTALMENT_FUTURE, max are very sporadic, but most of these graphs don't look like a difference in the problem and the problem doesn't change much.

Correlations

In [ ]:
```
tmp[list(PCB_df.columns) + ["TARGET"]].corr()["TARGET"]
```

Out[52]:
```
(CNT_INSTALMENT, count)          -0.035802
(CNT_INSTALMENT_FUTURE, max)      0.013324
(CNT_INSTALMENT_FUTURE, min)      0.019010
(CNT_INSTALMENT_FUTURE, mean)     0.027827
(MONTHS_BALANCE, max)            -0.004321
(MONTHS_BALANCE, min)             0.055307
(MONTHS_BALANCE, mean)            0.034543
TARGET                            1.000000
Name: TARGET, dtype: float64
```

- As we can see, the best feature from these is (MONTHS_BALANCE, min), though it is not as high of a correlation as in other datasets.

Missing Values

In [ ]:
```
missing_vals(tmp[PCB_df.columns])
```

Out[53]:

	Percent	Missing Count
(CNT_INSTALMENT_FUTURE, max)	5.88	18091
(CNT_INSTALMENT_FUTURE, min)	5.88	18091
(CNT_INSTALMENT_FUTURE, mean)	5.88	18091
(CNT_INSTALMENT, count)	5.88	18067
(MONTHS_BALANCE, max)	5.88	18067
(MONTHS_BALANCE, min)	5.88	18067
(MONTHS_BALANCE, mean)	5.88	18067

There are a relatively low amount of misssing values here.

Installments Payments

In [ ]:
```
datasets['installments_payments'].columns
```

Out[54]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NUM_INSTALMENT_VERSION',
       'NUM_INSTALMENT_NUMBER', 'DAYS_INSTALMENT', 'DAYS_ENTRY_PAYMEN
T',
       'AMT_INSTALMENT', 'AMT_PAYMENT'],
      dtype='object')
```

In [ ]:
```
datasets['installments_payments'].info()
datasets['installments_payments'].describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_PREV              int64  
 1   SK_ID_CURR              int64  
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   int64  
 4   DAYS_INSTALMENT         float64
 5   DAYS_ENTRY_PAYMENT      float64
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
```

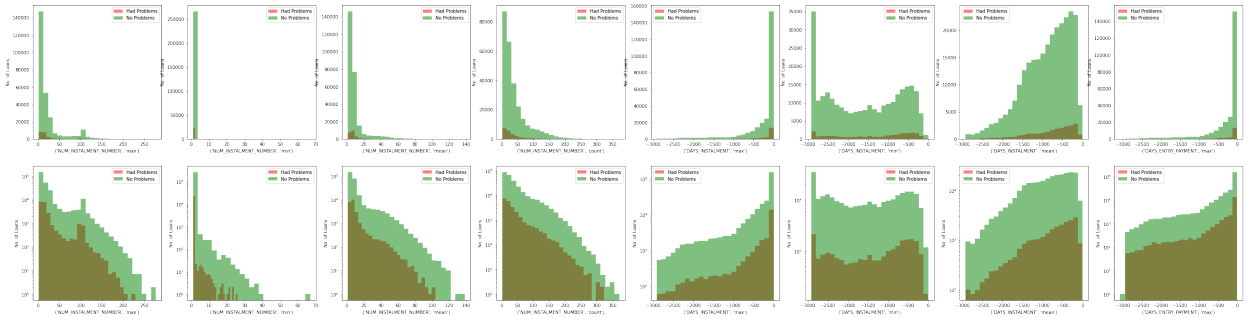
Out[55]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER
count	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07
mean	1.903365e+06	2.784449e+05	8.566373e-01	1.887090e+01
std	5.362029e+05	1.027183e+05	1.035216e+00	2.666407e+01
min	1.000001e+06	1.000010e+05	0.000000e+00	1.000000e+00
25%	1.434191e+06	1.896390e+05	0.000000e+00	4.000000e+00
50%	1.896520e+06	2.786850e+05	1.000000e+00	8.000000e+00
75%	2.369094e+06	3.675300e+05	1.000000e+00	1.900000e+01
max	2.843499e+06	4.562550e+05	1.780000e+02	2.770000e+02

In [ ]:
```
IP_df = datasets['installments_payments'].groupby('SK_ID_CURR').agg({
    "NUM_INSTALMENT_NUMBER":["max","min", "mean", "count"],
    "DAYS_INSTALMENT":["max","min", "mean"],
    "DAYS_ENTRY_PAYMENT":["max","min", "mean"],
    "AMT_INSTALMENT":["sum"],
    "AMT_PAYMENT":["sum"]
})
IP_df["SUM_MISSED"] = IP_df[("AMT_INSTALMENT", "sum")] - IP_df[("AMT_P
```

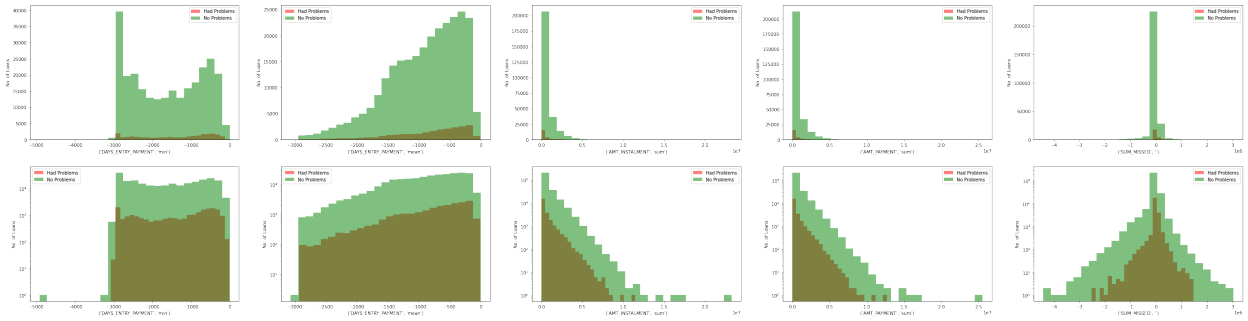
In [ ]:
```
tmp = datasets['application_train'].merge(IP_df, how='left', on='SK_ID
```

In [ ]:
```
fig, axs = plt.subplots(2, 8, figsize=(48,12))
for i in range(8):
  num_hist(tmp, IP_df.columns[i], axs[0,i])

# log graphs
for i in range(8):
  num_hist(tmp, IP_df.columns[i], axs[1,i], True)
```



None of these look very interesting. They all have uniform differences.

In [ ]:
```
fig, axs = plt.subplots(2, 5, figsize=(48,12))
for i in range(5):
  num_hist(tmp, IP_df.columns[8+i], axs[0,i])

# log graphs
for i in range(5):
  num_hist(tmp, IP_df.columns[8+i], axs[1,i], True)
```



These are also not very interesting.

In [ ]:
```
tmp[list(IP_df.columns) + ["TARGET"]].corr()["TARGET"]
```

Out[61]:
```
(NUM_INSTALMENT_NUMBER, max)      0.006304
(NUM_INSTALMENT_NUMBER, min)     -0.002334
(NUM_INSTALMENT_NUMBER, mean)    -0.009537
(NUM_INSTALMENT_NUMBER, count)   -0.021096
(DAYS_INSTALMENT, max)           -0.003231
(DAYS_INSTALMENT, min)            0.058648
(DAYS_INSTALMENT, mean)           0.043509
(DAYS_ENTRY_PAYMENT, max)        -0.002298
(DAYS_ENTRY_PAYMENT, min)         0.058794
(DAYS_ENTRY_PAYMENT, mean)        0.043992
(AMT_INSTALMENT, sum)            -0.019811
(AMT_PAYMENT, sum)               -0.024375
(SUM_MISSED, )                    0.027932
TARGET                            1.000000
Name: TARGET, dtype: float64
```

Missing Values

In [ ]:
```
missing_vals(tmp[IP_df.columns])
```

Out[62]:

	Percent	Missing Count
(DAYS_ENTRY_PAYMENT, max)	5.16	15876
(DAYS_ENTRY_PAYMENT, min)	5.16	15876
(DAYS_ENTRY_PAYMENT, mean)	5.16	15876
(NUM_INSTALMENT_NUMBER, max)	5.16	15868
(NUM_INSTALMENT_NUMBER, min)	5.16	15868
(NUM_INSTALMENT_NUMBER, mean)	5.16	15868
(NUM_INSTALMENT_NUMBER, count)	5.16	15868
(DAYS_INSTALMENT, max)	5.16	15868
(DAYS_INSTALMENT, min)	5.16	15868
(DAYS_INSTALMENT, mean)	5.16	15868

These all have relatively low missing counts, so the correlation is probably fairly accurate.

Bureau

In [ ]:
```
datasets['bureau'].columns
```

Out[63]:
```
Index(['SK_ID_CURR', 'SK_ID_BUREAU', 'CREDIT_ACTIVE', 'CREDIT_CURRENCY',
       'DAYS_CREDIT', 'CREDIT_DAY_OVERDUE', 'DAYS_CREDIT_ENDDATE',
       'DAYS_ENDDATE_FACT', 'AMT_CREDIT_MAX_OVERDUE', 'CNT_CREDIT_PROLONG',
       'AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT',
       'AMT_CREDIT_SUM_OVERDUE', 'CREDIT_TYPE', 'DAYS_CREDIT_UPDATE',
       'AMT_ANNUITY'],
      dtype='object')
```

In [ ]:
```
datasets['installments_payments'].info()
datasets['installments_payments'].describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_PREV              int64  
 1   SK_ID_CURR              int64  
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   int64  
 4   DAYS_INSTALMENT         float64
 5   DAYS_ENTRY_PAYMENT      float64
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
```

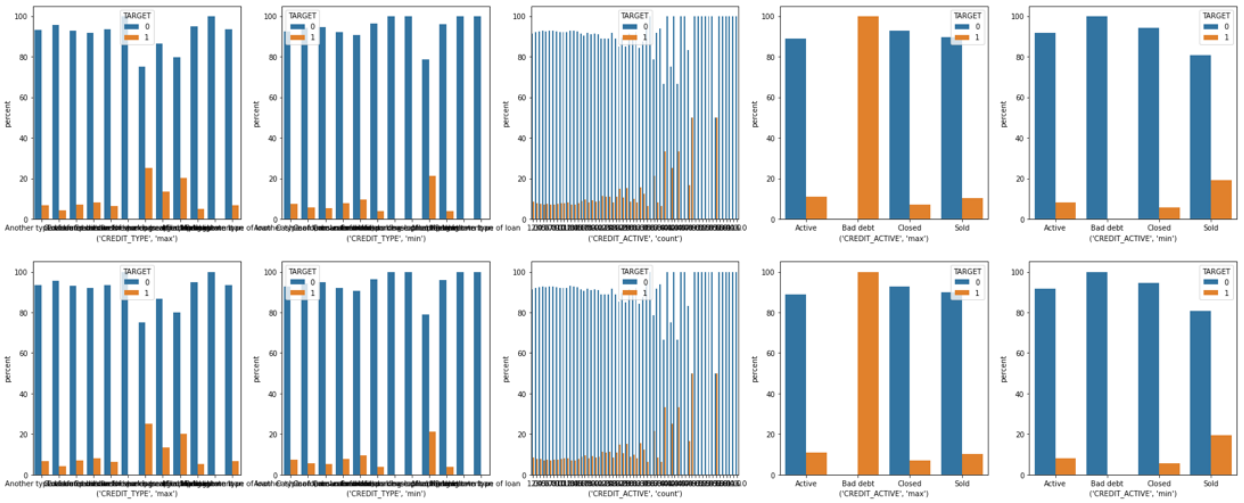
Out[64]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER
count	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07
mean	1.903365e+06	2.784449e+05	8.566373e-01	1.887090e+01
std	5.362029e+05	1.027183e+05	1.035216e+00	2.666407e+01
min	1.000001e+06	1.000010e+05	0.000000e+00	1.000000e+00
25%	1.434191e+06	1.896390e+05	0.000000e+00	4.000000e+00
50%	1.896520e+06	2.786850e+05	1.000000e+00	8.000000e+00
75%	2.369094e+06	3.675300e+05	1.000000e+00	1.900000e+01
max	2.843499e+06	4.562550e+05	1.780000e+02	2.770000e+02

In [ ]:
```
B_df = datasets['bureau'].groupby('SK_ID_CURR').agg({
    "CREDIT_TYPE":["max", "min"],
    "CREDIT_ACTIVE":["count", "max", "min"],
    "DAYS_CREDIT":["max", "min", "mean"],
    "CREDIT_DAY_OVERDUE":["max"],
    "AMT_CREDIT_SUM":["max"],
    "AMT_CREDIT_SUM_OVERDUE":["max"],
})
```

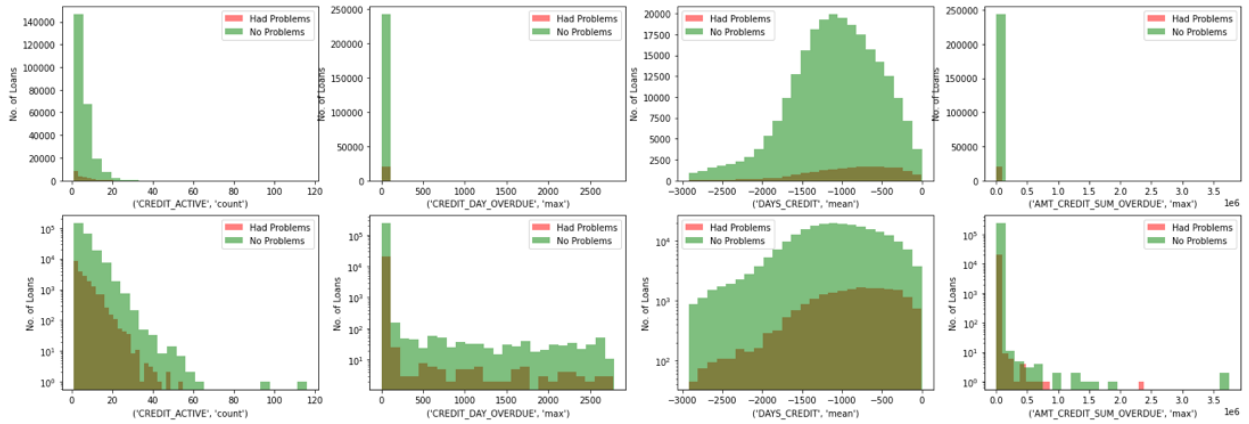
In [ ]:
```
tmp = datasets['application_train'].merge(B_df, how='left', on='SK_ID_
```

In [ ]:
```
fig, axs = plt.subplots(2, 5, figsize=(30,12))
for i in range(5):
  if B_df.columns[i][0] == 'CREDIT_ACTIVE' or B_df.columns[i][0] == 'C
    cat_bar(tmp, B_df.columns[i], axs[0,i])
  else:
    num_hist(tmp, B_df.columns[i], axs[0,i])

# log graphs
for i in range(5):
  if B_df.columns[i][0] == 'CREDIT_ACTIVE' or B_df.columns[i][0] == 'C
    cat_bar(tmp, B_df.columns[i], axs[1,i])
  else:
    num_hist(tmp, B_df.columns[i], axs[1,i], True)
```

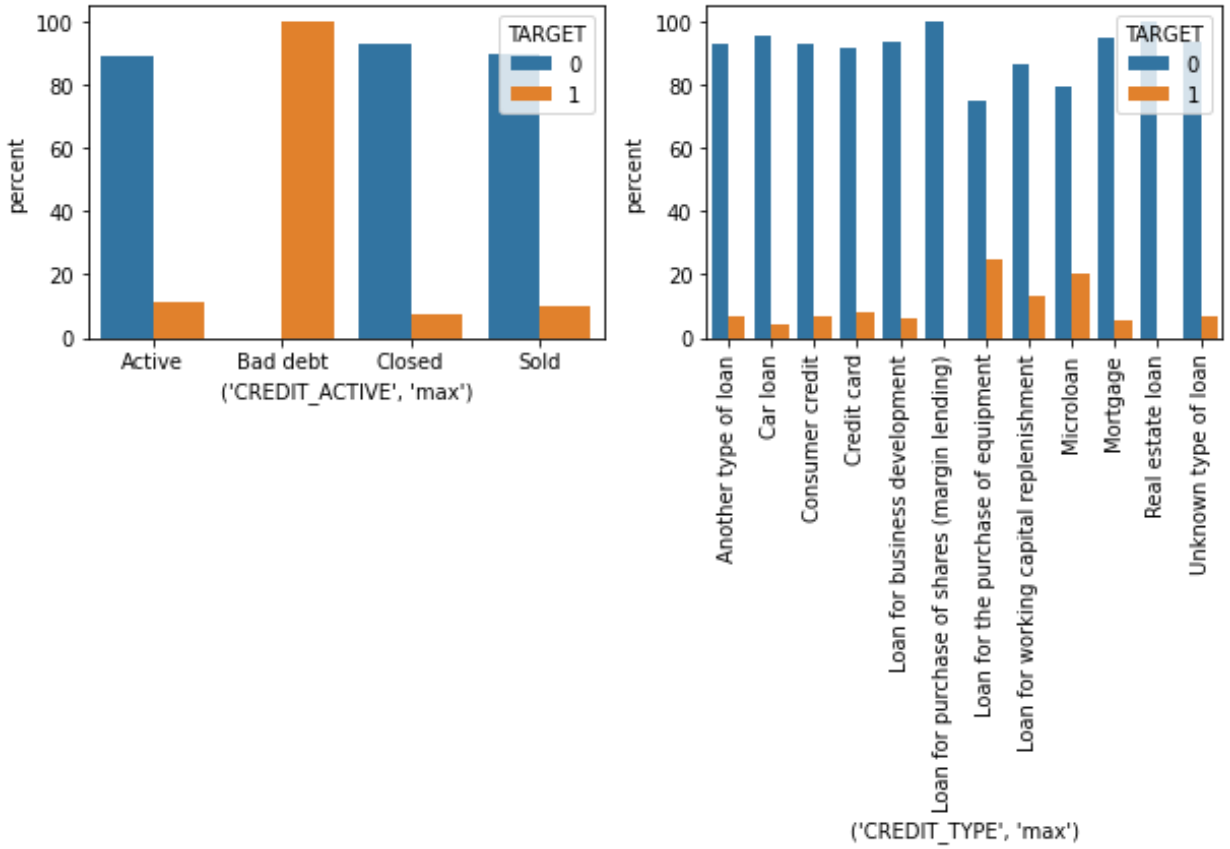


- All days_credit and credit_type graphs look pretty promising, as some of the graphs show that they are more likely to have problems than other graphs.

In [ ]:
```
fig, axs = plt.subplots(2, 4, figsize=(24, 8))
num_hist(tmp, ('CREDIT_ACTIVE', 'count'), axs[0,0])
num_hist(tmp, ('CREDIT_DAY_OVERDUE', 'max'), axs[0,1])
num_hist(tmp, ('DAYS_CREDIT', 'mean'), axs[0,2])
num_hist(tmp, ('AMT_CREDIT_SUM_OVERDUE', 'max'), axs[0,3])
# log graphs
num_hist(tmp, ('CREDIT_ACTIVE', 'count'), axs[1,0], True)
num_hist(tmp, ('CREDIT_DAY_OVERDUE', 'max'), axs[1,1], True)
num_hist(tmp, ('DAYS_CREDIT', 'mean'), axs[1,2], True)
num_hist(tmp, ('AMT_CREDIT_SUM_OVERDUE', 'max'), axs[1,3], True)
```



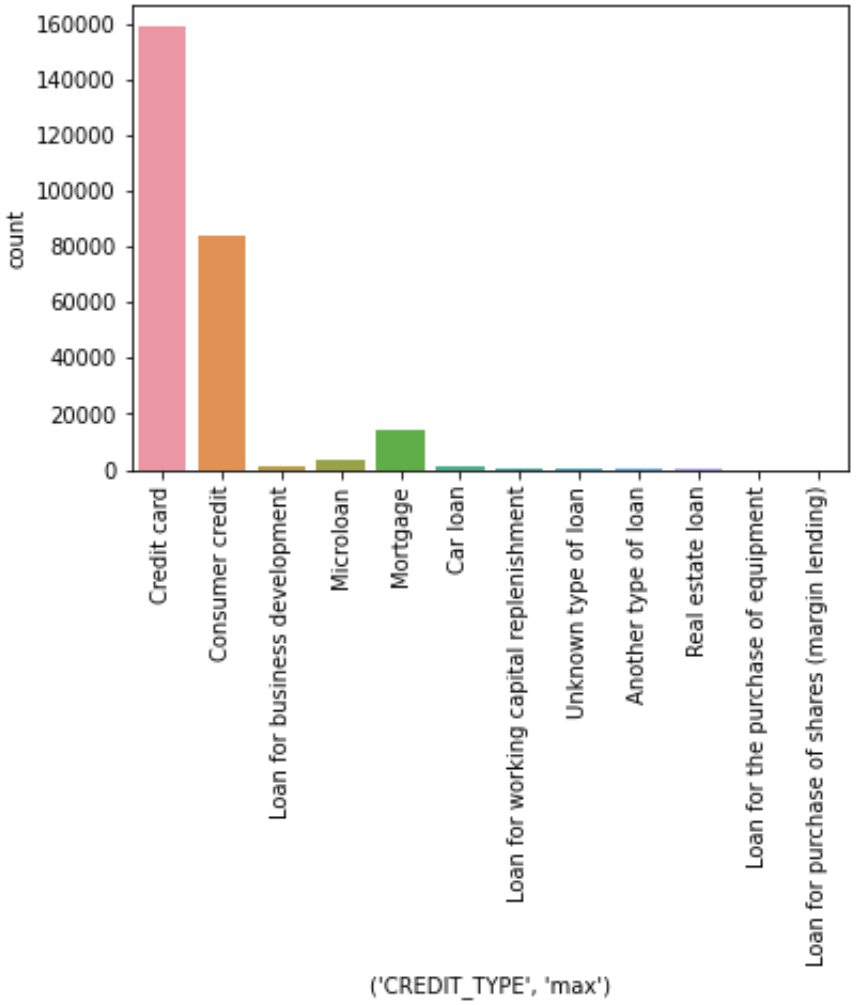
In [ ]:
```
fig, axs = plt.subplots(1, 2, figsize=(10, 3))
cat_bar(tmp, ('CREDIT_ACTIVE', 'max'), axs[0])
plt.xticks(rotation=90)
cat_bar(tmp, ('CREDIT_TYPE', 'max'), axs[1])
```



- There are some clear trends here. In "CREDIT_ACTIVE", the "CLOSED" part had fewer problems than the other groups. Also, everyone in the "bad debt" group had problems.
- In `CREDIT_TYPE` , the customer who used his loan to buy shares or real estate had no problems with repayment, while those who used it for equipment, or "working capital replenishment" ', or the small loans they all had bigger problems with repaying.

In [ ]:
```
plt.xticks(rotation=90)
sns.countplot(tmp['CREDIT_TYPE', 'max'])
```

Out[70]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2eeabf35d0>
```



In [ ]:
```
len(tmp.loc[tmp[('CREDIT_ACTIVE', 'max')] == 'Bad debt'])
```

Out[71]:
```
1
```

- In `CREDIT_TYPE` , except for the 4th most popular category," Microloan ", it seems that the groups we were interested in didn't have many samples, but there are significant differences. The most popular type of loan is the "credit card", which is significantly more difficult than the second and third most popular types "consumer credit" and "mortgage" respectively. This feature is probably still interesting.
- Turn these into "credit card", "consumer credit", "mortgage", "other difficult", "other", "other difficult" You may be able to group them. Includes microloans, etc., where the client had a significant difficulty in repayment of the loan

Correlations

In [ ]:
```
tmp[list(B_df.columns) + ["TARGET"]].corr()["TARGET"]
```

Out[72]:
```
(CREDIT_ACTIVE, count)           0.004056
(DAYS_CREDIT, max)               0.049782
(DAYS_CREDIT, min)               0.075248
(DAYS_CREDIT, mean)              0.089729
(CREDIT_DAY_OVERDUE, max)        0.005493
(AMT_CREDIT_SUM, max)           -0.019737
(AMT_CREDIT_SUM_OVERDUE, max)    0.010614
TARGET                           1.000000
Name: TARGET, dtype: float64
```

As we can see from the correlations, `DAYS_CREDIT` is quite significant.

Missing Values

In [ ]:
```
missing_vals(tmp[B_df.columns])
```

Out[73]:

	Percent	Missing Count
(AMT_CREDIT_SUM, max)	14.32	44021
(CREDIT_TYPE, max)	14.31	44020
(CREDIT_TYPE, min)	14.31	44020
(CREDIT_ACTIVE, count)	14.31	44020
(CREDIT_ACTIVE, max)	14.31	44020
(CREDIT_ACTIVE, min)	14.31	44020
(DAYS_CREDIT, max)	14.31	44020
(DAYS_CREDIT, min)	14.31	44020
(DAYS_CREDIT, mean)	14.31	44020
(CREDIT_DAY_OVERDUE, max)	14.31	44020

There's a decent amount of data missing, but since we have ~85% of the data, it should still perform pretty well for most data

Other Datasets

These are all of the aggregate datasets we want to get from each secondary file.

In [ ]:
```
new_cat_features = [("CREDIT_ACTIVE", "max"), ("NAME_CONTRACT_STATUS",
new_num_features = [("AMT_APPLICATION", "sum"), ("AMT_APPLICATION", "c
```

Combine Secondaries Into Application

In [ ]:

```
train = datasets['application_train'].merge(PA_df, how='left', on='SK_
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')

test = datasets['application_test'].merge(PA_df, how='left', on='SK_ID
test = test.merge(PCB_df, how='left', on='SK_ID_CURR')
test = test.merge(CCB_df, how='left', on='SK_ID_CURR')
test = test.merge(IP_df, how='left', on='SK_ID_CURR')
test = test.merge(B_df, how='left', on='SK_ID_CURR')
```

In [ ]:
```
train
```

Out[76]:

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

307511 rows × 170 columns

In [ ]:
```
correl = train.corr()['TARGET'].sort_values()
print('10 Most +ve Correlations:\n', correl.tail(10))
print('\n10 Most -ve Correlations:\n', correl.head(10))
```

```
10 Most +ve Correlations:
 (DAYS_INSTALMENT, min)          0.058648
(DAYS_ENTRY_PAYMENT, min)       0.058794
REGION_RATING_CLIENT            0.058899
REGION_RATING_CLIENT_W_CITY     0.060893
(MONTHS_BALANCE, min)           0.061359
(DAYS_CREDIT, min)              0.075248
DAYS_BIRTH                      0.078239
(AMT_BALANCE, mean)             0.087177
(DAYS_CREDIT, mean)             0.089729
TARGET                          1.000000
Name: TARGET, dtype: float64

10 Most -ve Correlations:
 EXT_SOURCE_3                      -0.178919
EXT_SOURCE_2                      -0.160472
EXT_SOURCE_1                      -0.155317
(AMT_CREDIT_LIMIT_ACTUAL, count)  -0.060481
DAYS_EMPLOYED                     -0.044932
FLOORSMAX_AVG                     -0.044003
FLOORSMAX_MEDI                    -0.043768
FLOORSMAX_MODE                    -0.043226
AMT_GOODS_PRICE                   -0.039645
REGION_POPULATION_RELATIVE        -0.037227
Name: TARGET, dtype: float64
```

BasePipeline with complete data

Features Used

In [ ]:
```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE"
]

num_features = [
"EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU
'REGION_RATING_CLIENT_W_CITY' ,
'DAYS_BIRTH'    ,
'DAYS_EMPLOYED',
]
```

Pipeline

In [ ]:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB


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

num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='most_frequent')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

full_pipeline1 = Pipeline([
  ('preprocessing', preprocess_pipeline),
  ('bayes',GaussianNB())
])

full_pipeline2 = Pipeline([
  ('preprocessing', preprocess_pipeline),
  ('linear', LogisticRegression())
])
```

FitPipeline

In [ ]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from time import time
X_train = train.loc[:, train.columns != "TARGET"]
y_train = train['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,

start = time()
full_pipeline1.fit(X_train, y_train)
np.random.seed(42)

splits = ShuffleSplit(n_splits = 30, test_size = .3, random_state = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full

logit_score_train = logit_scores.mean()
train_time = np.round(time() - start, 4)

# Time and score test predictions
start = time()
logit_score_test  = full_pipeline1.score(X_test, y_test)
roc = roc_auc_score(y_test, full_pipeline1.predict_proba(X_test)[:, 1]
test_time = np.round(time() - start, 4)
```

In [ ]:
```
# initialize results table
results = pd.DataFrame(columns=["ExpID", "ROC AUC Score", "Cross fold
```

In [ ]:
```
results.loc[1] = ["Baseline Bayes w/ All Data", roc,logit_score_train,
                  train_time, test_time, "Untuned Bayes"]
```

In [ ]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from time import time
X_train = train.loc[:, train.columns != "TARGET"]
y_train = train['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,

start = time()
full_pipeline2.fit(X_train, y_train)
np.random.seed(42)

splits = ShuffleSplit(n_splits = 30, test_size = .3, random_state = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full

logit_score_train = logit_scores.mean()
train_time = np.round(time() - start, 4)

# Time and score test predictions
start = time()
logit_score_test  = full_pipeline2.score(X_test, y_test)
roc = roc_auc_score(y_test, full_pipeline2.predict_proba(X_test)[:, 1]
test_time = np.round(time() - start, 4)
```

In [ ]:
```
results.loc[2] = ["Baseline Linear reg w/ All Data", roc,logit_score_t
                  train_time, test_time, "Untuned LogisticRegression"
```

In [ ]:
```
results
```

Out[96]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
1	Baseline Bayes w/ All Data	0.734758	0.919634	0.917	291.1163	0.3749	Untuned Bayes
2	Baseline Linear reg w/ All Data	0.734758	0.919634	0.917	326.3181	0.4088	Untuned LogisticRegression

upgrade lightgbm

!pip install --upgrade lightgbm !pip install --upgrade xgboost

In [ ]:
```
import numpy as np
import pandas as pd
import os
import zipfile
import warnings
warnings.filterwarnings('ignore')

def load_data(in_path, name):
    df = pd.read_csv(in_path)
    print("DataSet:",f'{name}')
    print(f"shape is {df.shape}")
    print(df.info())
    display(df.head(3))
    return df

datasets={}
DATA_DIR = "home-credit-default-risk"

ds_names = ("application_train", "application_test", "bureau","bureau_
            "previous_application","POS_CASH_balance")

for ds_name in ds_names:
    datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.c
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,},
```

```
DataSet: application_train
shape is (307511, 122)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
None
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_C
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	

3 rows × 122 columns

```
DataSet: application_test
```

In [ ]:
```
datasets.keys()
```

Out[98]:
```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau
_balance', 'credit_card_balance', 'installments_payments', 'previous_
application', 'POS_CASH_balance'])
```

New Application Features

To evaluate all possible ways of merging our application data to create new data, we take a brute-force approach. This consumes a large amount of RAM and cannot be repeated without clearing all of it. As a result, the code has been commented out and only specific samples have been chosen.

In [ ]:
```
all_num_features = [
  "CNT_CHILDREN", "AMT_INCOME_TOTAL", "AMT_CREDIT", "AMT_ANNUITY", "RE
  "DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_REGISTRATION", "CNT_FAM_MEMBERS
  "EXT_SOURCE_2", "EXT_SOURCE_3","AMT_GOODS_PRICE", "OBS_30_CNT_SOCIAL
  "DEF_30_CNT_SOCIAL_CIRCLE", "OBS_60_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_
  "DAYS_LAST_PHONE_CHANGE", "AMT_REQ_CREDIT_BUREAU_HOUR", "AMT_REQ_CRE
  "AMT_REQ_CREDIT_BUREAU_WEEK", "AMT_REQ_CREDIT_BUREAU_MON", "AMT_REQ_
  "AMT_REQ_CREDIT_BUREAU_YEAR", "DAYS_ID_PUBLISH",
]
```

Creating new features

In [ ]:
```
test_df = pd.DataFrame()
corrs = {}

# # this is an example for addition. Due to memory constraints, we can
 # one operation (+,-,/,*) per run
for key1 in all_num_features:
    for key2 in all_num_features:
         if key1 != key2:
            test_df[key1+"+"+key2] = datasets["application_train"][key
            corrs[key1+"+"+key2] = test_df[key1+"+"+key2].corr(dataset

test_df
```

Out[8]:

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

307511 rows × 552 columns

In [ ]:
```
corrs
```

```
 'REGION_POPULATION_RELATIVE+CNT_CHILDREN': 0.01847975953337787,
 'REGION_POPULATION_RELATIVE+CNT_FAM_MEMBERS': 0.008744594693338159,
 'REGION_POPULATION_RELATIVE+DAYS_BIRTH': 0.07823919765682923,
 'REGION_POPULATION_RELATIVE+DAYS_EMPLOYED': -0.04493166631988388,
 'REGION_POPULATION_RELATIVE+DAYS_ID_PUBLISH': 0.05145683337084685,
 'REGION_POPULATION_RELATIVE+DAYS_LAST_PHONE_CHANGE': 0.055217901393361156,
 'REGION_POPULATION_RELATIVE+DAYS_REGISTRATION': 0.041974725542145194,
 'REGION_POPULATION_RELATIVE+DEF_30_CNT_SOCIAL_CIRCLE': 0.031086239273997618,
 'REGION_POPULATION_RELATIVE+DEF_60_CNT_SOCIAL_CIRCLE': 0.029845959501545727,
 'REGION_POPULATION_RELATIVE+EXT_SOURCE_2': -0.1604647541352782,
 'REGION_POPULATION_RELATIVE+EXT_SOURCE_3': -0.18093058397229986,
 'REGION_POPULATION_RELATIVE+OBS_30_CNT_SOCIAL_CIRCLE': 0.008918919840672064,
 'REGION_POPULATION_RELATIVE+OBS_60_CNT_SOCIAL_CIRCLE': 0.008808479493836008}
```

Analysis of new features

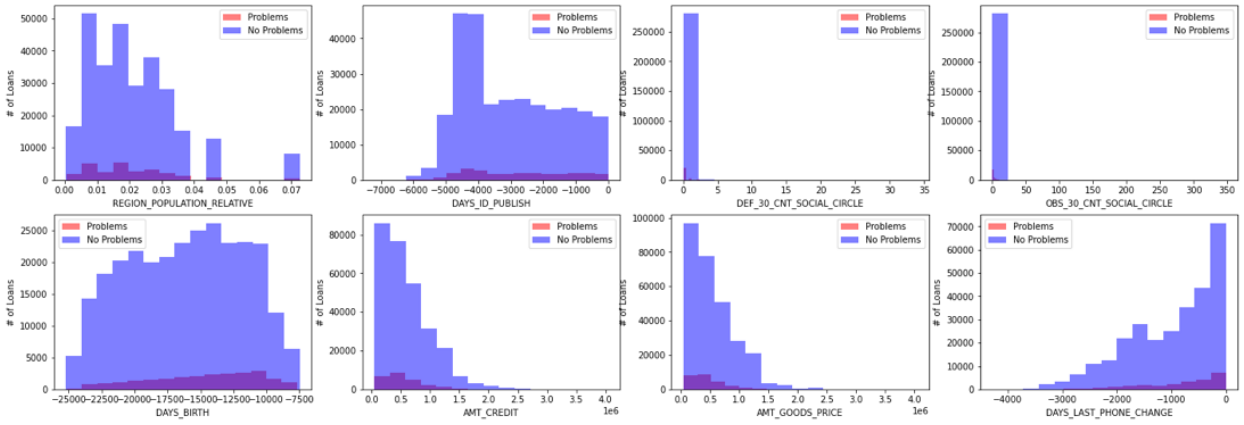
Just looking at the correlation values produced some intriguing results out of all of the features we built. But now is the time to look over our new data for anomalies or red flags.

In [ ]:

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

# some generalized functions to make the analysis easy on us
def cat_bar(df, x, ax):
  df2 = df.groupby(x)['TARGET'].value_counts(normalize=True).mul(100).
  sns.barplot(x=x,y='percent',hue='TARGET',data=df2,ax=ax)

def num_hist(df, y, ax, log=False):
  if log: ax.set_yscale('log')

  ax.hist(df[df["TARGET"]==1][y], bins=15, alpha=0.5, color="red", lab
  ax.hist(df[df["TARGET"]==0][y], bins=15, alpha=0.5, color="blue", la

  ax.set_xlabel(y)
  ax.set_ylabel("# of Loans")
  ax.legend()

def missing_vals(df):
  percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(as
  sum_missing = df.isna().sum().sort_values(ascending = False)
  missing_data  = pd.concat([percent, sum_missing], axis=1, keys=['Per
  return missing_data.head(10)
```

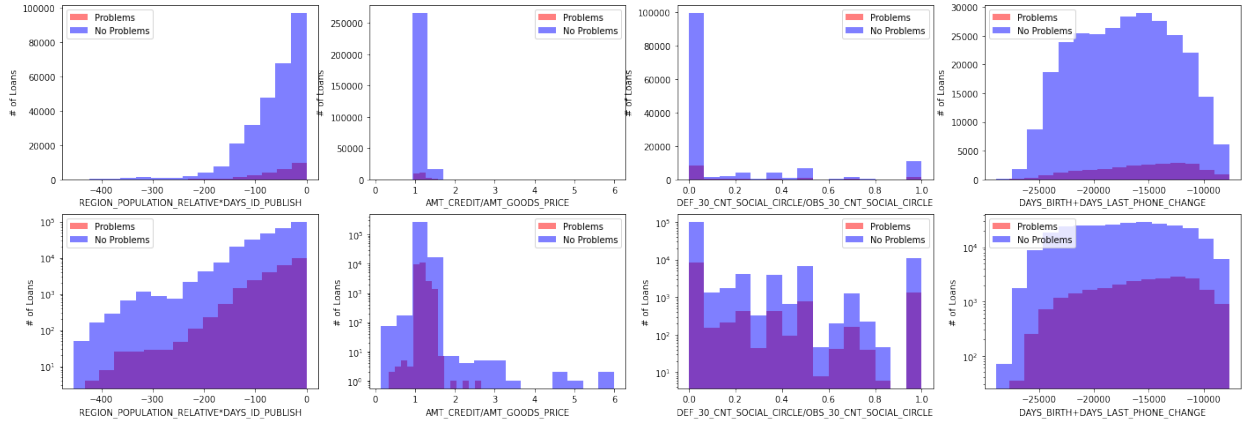
In [ ]:
```
temp_app = pd.DataFrame()
app_df = datasets['application_train']
temp_app["TARGET"] = app_df['TARGET']

temp_app["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = app_df['REGIO
temp_app["AMT_CREDIT/AMT_GOODS_PRICE"] = app_df['AMT_CREDIT'] / app_df
temp_app["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = app_df
temp_app["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = app_df['DAYS_BIRTH'] +
```

In [ ]:
```
fig, axs = plt.subplots(2, 4, figsize=(24, 8))
num_hist(app_df, "REGION_POPULATION_RELATIVE", axs[0,0])
num_hist(app_df, "DAYS_ID_PUBLISH", axs[0,1])
num_hist(app_df, "DEF_30_CNT_SOCIAL_CIRCLE", axs[0,2])
num_hist(app_df, "OBS_30_CNT_SOCIAL_CIRCLE", axs[0,3])

num_hist(app_df, "DAYS_BIRTH", axs[1,0])
num_hist(app_df, "AMT_CREDIT", axs[1,1])
num_hist(app_df, "AMT_GOODS_PRICE", axs[1,2])
num_hist(app_df, "DAYS_LAST_PHONE_CHANGE", axs[1,3])
```



The original features, which are graphed below, were used to construct the new features. It's possible that you'll want to revisit these graphs in the future.

In [ ]:
```
fig, axs = plt.subplots(2, 4, figsize=(24, 8))
num_hist(temp_app, "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", axs[0
num_hist(temp_app, "AMT_CREDIT/AMT_GOODS_PRICE", axs[0,1])
num_hist(temp_app, "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"
num_hist(temp_app, "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE", axs[0,3])
# log graphs
num_hist(temp_app, "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", axs[1
num_hist(temp_app, "AMT_CREDIT/AMT_GOODS_PRICE", axs[1,1], True)
num_hist(temp_app, "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"
num_hist(temp_app, "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE", axs[1,3], True
```



These connections are pretty intriguing.

REGION_POPULATION_RELATIVE and DAYS_ID_PUBLISH have graphs with one high point in the middle, thanks to the graphs above this set. However, there is a clear trend in REGION_POPULATION_RELATION*DAYS_ID_PUBLISH: the further to the right, the more difficulty the client has with payback.

DEF_30_CNT_SOCIAL_CIRCLE and OBS_30_CNT_SOCIAL_CIRCLE both contained a substantial number of the samples in one area, making it difficult to discern points apart, but DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE did a much better job of spreading those points out.

Baseline Model for just Application Data

Let's create a basic model with just applicaiton data

In [ ]:
```
results = pd.DataFrame(columns=["ExpID", "ROC AUC Score", "Cross fold
```

In [ ]:
```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU
]
```

In [ ]:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='most_frequent')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

full_pipeline = Pipeline([
  ('preprocessing', preprocess_pipeline),
  ('linear', LogisticRegression())
])
```

In [ ]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from time import time
X_train = datasets["application_train"].loc[:, datasets['application_t
y_train = datasets["application_train"]['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,

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

cv10Splits = ShuffleSplit(n_splits = 10, test_size = .3, random_state
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full
logit_score_train = logit_scores.mean()

# Time and score test predictions
start = time()
logit_score_test  = full_pipeline.score(X_test, y_test)
roc = roc_auc_score(y_test, full_pipeline.predict_proba(X_test)[:, 1])
test_time = np.round(time() - start, 4)
```

In [ ]:
```
results.loc[0] = ["Baseline", roc,logit_score_train, np.round(logit_sc
                  train_time, test_time, "LogisticRegression"]
results
```

Out[18]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	0.91996	0.917	9.6565	0.3952	LogisticRegression

Baseline Model with our new features

In [ ]:
```
# create new features
app_df = datasets['application_train']

app_df["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = app_df['REGION_
app_df["AMT_CREDIT/AMT_GOODS_PRICE"] = app_df['AMT_CREDIT'] / app_df['
app_df["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = app_df['
app_df["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = app_df['DAYS_BIRTH'] + a
app_df["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = app_df['
app_df["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = app_df['AMT_GOODS_PRICE'] +
app_df["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = app_df['REGION_
```

In [ ]:

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO
]
```

In [ ]:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='most_frequent')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

full_pipeline = Pipeline([
  ('preprocessing', preprocess_pipeline),
  ('linear', LogisticRegression())
])
```

In [ ]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from time import time
X_train = datasets["application_train"].loc[:, datasets['application_t
y_train = datasets["application_train"]['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,

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

cv10Splits = ShuffleSplit(n_splits = 10, test_size = .3, random_state
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full
logit_score_train = logit_scores.mean()

# Time and score test predictions
start = time()
logit_score_test  = full_pipeline.score(X_test, y_test)
roc = roc_auc_score(y_test, full_pipeline.predict_proba(X_test)[:, 1])
test_time = np.round(time() - start, 4)
```

In [ ]:
```
results.loc[1] = ["Baseline", roc,pct(logit_score_train), np.round(pct
                  train_time, test_time, "LogisticRegression + new Ap
results
```

Out[23]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	0.91996	0.917	9.6565	0.3952	LogisticRegression
1	Baseline	0.739299	92.00000	91.700	8.6859	0.3853	LogisticRegression + new Application features

Our additional features did not increase our accuracy, but they did improve the ROC AUC score. This is a win for us because the AUC score is the most essential statistic.

New Features for Other Datasets

In [ ]:

```
def missing_vals(df):
  percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(as
  sum_missing = df.isna().sum().sort_values(ascending = False)
  missing_data  = pd.concat([percent, sum_missing], axis=1, keys=['Per
  return missing_data.head(10)
```

Previous Applications

In [ ]:

```
PA_df = datasets['previous_application'].groupby('SK_ID_CURR').agg({
    "AMT_APPLICATION": "mean",
    "CNT_PAYMENT": "max",
    "DAYS_TERMINATION":"mean",

    "NAME_PORTFOLIO": "max",
    "NAME_GOODS_CATEGORY": "max",
    "NAME_SELLER_INDUSTRY": "max",
})
```

Previous PCB

In [ ]:

```
PCB_df_copy = datasets['POS_CASH_balance'].groupby('SK_ID_PREV').agg({
    "CNT_INSTALMENT": "count",
    "CNT_INSTALMENT_FUTURE": "mean",
    "MONTHS_BALANCE": "min",
})
```

In [ ]:

```
POS_to_PA_df =  datasets['previous_application'].merge(PCB_df_copy, ho
```

In [ ]:

```
PCB_df_temp = POS_to_PA_df.groupby('SK_ID_CURR').agg({
    "CNT_INSTALMENT": "count",
    "CNT_INSTALMENT_FUTURE": "mean",
    "MONTHS_BALANCE": "min",
})
PCB_df_temp=PCB_df_temp.rename({"CNT_INSTALMENT":"PREV_CNT_INSTALMENT"
```

In [ ]:

```
PA_df = pd.concat([PA_df, PCB_df_temp], axis=1)
```

EDA:

In [ ]:
```
temp_app = datasets['application_train'].merge(PCB_df_temp, how='left'
```

In [ ]:
```
print(temp_app[["PREV_CNT_INSTALMENT", "PREV_CNT_INSTALMENT_FUTURE","P

missing_vals(temp_app[["PREV_CNT_INSTALMENT", "PREV_CNT_INSTALMENT_FUT
```

```
PREV_CNT_INSTALMENT          -0.038646
PREV_CNT_INSTALMENT_FUTURE    0.032835
PREV_PCB_MONTHS_BALANCE       0.050945
TARGET                        1.000000
Name: TARGET, dtype: float64
```

Out[31]:

	Percent	Missing Count
PREV_CNT_INSTALMENT_FUTURE	6.69	20576
PREV_PCB_MONTHS_BALANCE	6.68	20544
PREV_CNT_INSTALMENT	5.35	16454

Previous IP

In [ ]:
```
IP_df_copy = datasets['installments_payments'].groupby('SK_ID_PREV').a
    "AMT_INSTALMENT": "sum",
    "AMT_PAYMENT": "sum",
    "DAYS_INSTALMENT": "min",
    "DAYS_ENTRY_PAYMENT": "min",
})
IP_df_copy["SUM_MISSED"] = IP_df_copy["AMT_INSTALMENT"] - IP_df_copy["
```

In [ ]:
```
IP_to_PA_df =  datasets['previous_application'].merge(IP_df_copy, how=
```

In [ ]:
```
IP_df_temp = IP_to_PA_df.groupby('SK_ID_CURR').agg({
    "AMT_INSTALMENT": "sum",
    "AMT_PAYMENT": "sum",
    "DAYS_INSTALMENT": "min",
    "DAYS_ENTRY_PAYMENT": "min"
})
IP_df_temp = IP_df_temp.rename({"AMT_INSTALMENT":"PREV_AMT_INSTALMENT"
```

In [ ]:
```
PA_df = pd.concat([PA_df, IP_df_temp], axis=1)
```

EDA:

In [ ]:
```
temp_app = datasets['application_train'].merge(IP_df_temp, how='left',
```

In [ ]:
```
print(temp_app[["PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT", "PREV_DAYS_I

missing_vals(temp_app[["PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT", "PREV
```

```
PREV_AMT_INSTALMENT       -0.018711
PREV_AMT_PAYMENT          -0.023428
PREV_DAYS_INSTALMENT       0.053545
PREV_DAYS_ENTRY_PAYMENT    0.053701
TARGET                     1.000000
Name: TARGET, dtype: float64
```

Out[37]:

	Percent	Missing Count
PREV_DAYS_ENTRY_PAYMENT	5.89	18113
PREV_DAYS_INSTALMENT	5.89	18105
PREV_AMT_INSTALMENT	5.35	16454
PREV_AMT_PAYMENT	5.35	16454

Previous CCB

In [ ]:
```
CCB_df_copy = datasets['credit_card_balance'].groupby('SK_ID_PREV').ag
    "AMT_BALANCE": "mean",
    "MONTHS_BALANCE": "min",
    "AMT_CREDIT_LIMIT_ACTUAL": "count",
})
```

In [ ]:
```
CCB_to_PA_df =  datasets['previous_application'].merge(CCB_df_copy, ho
```

In [ ]:
```
CCB_df_temp = CCB_to_PA_df.groupby('SK_ID_CURR').agg({
    "AMT_BALANCE": "mean",
    "MONTHS_BALANCE": "min",
    "AMT_CREDIT_LIMIT_ACTUAL": "count"
})
CCB_df_temp = CCB_df_temp.rename({"AMT_BALANCE":"PREV_AMT_BALANCE", "M
```

In [ ]:
```
PA_df = pd.concat([PA_df, CCB_df_temp], axis=1)
```

EDA:

In [ ]:
```
temp_app = datasets['application_train'].merge(CCB_df_temp, how='left'
```

In [ ]:
```
print(temp_app[["PREV_AMT_BALANCE", "PREV_CCB_MONTHS_BALANCE", "PREV_A

missing_vals(temp_app[["PREV_AMT_BALANCE", "PREV_CCB_MONTHS_BALANCE",
```

```
PREV_AMT_BALANCE                0.086693
PREV_CCB_MONTHS_BALANCE         0.049798
PREV_AMT_CREDIT_LIMIT_ACTUAL    0.018769
TARGET                          1.000000
Name: TARGET, dtype: float64
```

Out[43]:

	Percent	Missing Count
PREV_AMT_BALANCE	74.66	229577
PREV_CCB_MONTHS_BALANCE	74.66	229577
PREV_AMT_CREDIT_LIMIT_ACTUAL	5.35	16454

POS Cash Balances

In [ ]:
```
PCB_df = datasets['POS_CASH_balance'].groupby('SK_ID_CURR').agg({
    "CNT_INSTALMENT": "count",
    "CNT_INSTALMENT_FUTURE": "mean",
    "MONTHS_BALANCE": "min",
})
```

In [ ]:
```
comb_app = datasets['application_train'].merge(PCB_df, how='left', on=
```

In [ ]:
```
new_features = ["CNT_INSTALMENT", "CNT_INSTALMENT_FUTURE", "MONTHS_BAL
```

In [ ]:
```
test_df = pd.DataFrame()
corrs = {}

# this is an example for addition. Due to memory constraints, we can c
# one operation (+,-,/,*) per run
for key1 in all_num_features:
  for key2 in new_features:
    if key1 != key2:
      test_df[key1+"+"+key2] = comb_app[key1] + comb_app[key2]
      corrs[key1+"+"+key2] = test_df[key1+"+"+key2].corr(comb_app['TAR

test_df
```

Out[47]:

	CNT_CHILDREN+CNT_INSTALMENT	CNT_CHILDREN+CNT_INSTALMENT_FUTURE	CNT_(
0	19.0	15.000000	
1	28.0	5.785714	
2	4.0	2.250000	
3	20.0	8.650000	
4	66.0	8.969697	
...	...	...	
307506	8.0	4.375000	
307507	7.0	3.000000	
307508	17.0	2.000000	
307509	20.0	10.350000	
307510	71.0	15.140845	

307511 rows × 72 columns

In [ ]:
```
corrs
```

Out[48]:
```
{'AMT_ANNUITY+CNT_INSTALMENT': -0.008515917780832676,
 'AMT_ANNUITY+CNT_INSTALMENT_FUTURE': -0.008440793501912184,
 'AMT_ANNUITY+MONTHS_BALANCE': -0.008338714976260993,
 'AMT_CREDIT+CNT_INSTALMENT': -0.02730806085465315,
 'AMT_CREDIT+CNT_INSTALMENT_FUTURE': -0.027297097637611626,
 'AMT_CREDIT+MONTHS_BALANCE': -0.02730186511138258,
 'AMT_GOODS_PRICE+CNT_INSTALMENT': -0.03656542567387583,
 'AMT_GOODS_PRICE+CNT_INSTALMENT_FUTURE': -0.03655459975717435,
 'AMT_GOODS_PRICE+MONTHS_BALANCE': -0.03655872795487446,
 'AMT_INCOME_TOTAL+CNT_INSTALMENT': -0.0017020454766113427,
 'AMT_INCOME_TOTAL+CNT_INSTALMENT_FUTURE': -0.0016998058664761252,
 'AMT_INCOME_TOTAL+MONTHS_BALANCE': -0.0016916258330551884,
 'AMT_REQ_CREDIT_BUREAU_DAY+CNT_INSTALMENT': -0.033312974162881134,
 'AMT_REQ_CREDIT_BUREAU_DAY+CNT_INSTALMENT_FUTURE': 0.030706893892791
```

```
954,
 'AMT_REQ_CREDIT_BUREAU_DAY+MONTHS_BALANCE': 0.05194559120209246,
 'AMT_REQ_CREDIT_BUREAU_HOUR+CNT_INSTALMENT': -0.03332075980975123,
 'AMT_REQ_CREDIT_BUREAU_HOUR+CNT_INSTALMENT_FUTURE': 0.03067969225352
2555,
 'AMT_REQ_CREDIT_BUREAU_HOUR+MONTHS_BALANCE': 0.051939087105673784,
 'AMT_REQ_CREDIT_BUREAU_MON+CNT_INSTALMENT': -0.03374586729326836,
 'AMT_REQ_CREDIT_BUREAU_MON+CNT_INSTALMENT_FUTURE': 0.028455230401056
457,
 'AMT_REQ_CREDIT_BUREAU_MON+MONTHS_BALANCE': 0.05156907972925628,
 'AMT_REQ_CREDIT_BUREAU_QRT+CNT_INSTALMENT': -0.033385847584635926,
 'AMT_REQ_CREDIT_BUREAU_QRT+CNT_INSTALMENT_FUTURE': 0.029831973328827
02,
 'AMT_REQ_CREDIT_BUREAU_QRT+MONTHS_BALANCE': 0.051806007339173475,
 'AMT_REQ_CREDIT_BUREAU_WEEK+CNT_INSTALMENT': -0.03331528005020553,
 'AMT_REQ_CREDIT_BUREAU_WEEK+CNT_INSTALMENT_FUTURE': 0.03066428636133
782,
 'AMT_REQ_CREDIT_BUREAU_WEEK+MONTHS_BALANCE': 0.05193790339911501,
 'AMT_REQ_CREDIT_BUREAU_YEAR+CNT_INSTALMENT': -0.031088282729590088,
 'AMT_REQ_CREDIT_BUREAU_YEAR+CNT_INSTALMENT_FUTURE': 0.03198714792694
153,
 'AMT_REQ_CREDIT_BUREAU_YEAR+MONTHS_BALANCE': 0.05336017047333875,
 'CNT_CHILDREN+CNT_INSTALMENT': -0.03524935394976259,
 'CNT_CHILDREN+CNT_INSTALMENT_FUTURE': 0.030002495418029195,
 'CNT_CHILDREN+MONTHS_BALANCE': 0.055763072528013914,
 'CNT_FAM_MEMBERS+CNT_INSTALMENT': -0.03542846357737571,
 'CNT_FAM_MEMBERS+CNT_INSTALMENT_FUTURE': 0.029024574485010658,
 'CNT_FAM_MEMBERS+MONTHS_BALANCE': 0.05561658780911766,
 'DAYS_BIRTH+CNT_INSTALMENT': 0.08085915360319715,
 'DAYS_BIRTH+CNT_INSTALMENT_FUTURE': 0.08106136077166506,
 'DAYS_BIRTH+MONTHS_BALANCE': 0.08129650399774752,
 'DAYS_EMPLOYED+CNT_INSTALMENT': -0.04646180530996699,
 'DAYS_EMPLOYED+CNT_INSTALMENT_FUTURE': -0.0464331111404331,
 'DAYS_EMPLOYED+MONTHS_BALANCE': -0.04644492452513106,
 'DAYS_ID_PUBLISH+CNT_INSTALMENT': 0.05165098618789158,
 'DAYS_ID_PUBLISH+CNT_INSTALMENT_FUTURE': 0.052253730213610676,
 'DAYS_ID_PUBLISH+MONTHS_BALANCE': 0.053113380191010674,
 'DAYS_LAST_PHONE_CHANGE+CNT_INSTALMENT': 0.059379978630379464,
 'DAYS_LAST_PHONE_CHANGE+CNT_INSTALMENT_FUTURE': 0.0602853608974413,
 'DAYS_LAST_PHONE_CHANGE+MONTHS_BALANCE': 0.061167900828065745,
 'DAYS_REGISTRATION+CNT_INSTALMENT': 0.04369756229696534,
 'DAYS_REGISTRATION+CNT_INSTALMENT_FUTURE': 0.04398189946693391,
 'DAYS_REGISTRATION+MONTHS_BALANCE': 0.044366778647806256,
 'DEF_30_CNT_SOCIAL_CIRCLE+CNT_INSTALMENT': -0.035193654156639935,
 'DEF_30_CNT_SOCIAL_CIRCLE+CNT_INSTALMENT_FUTURE': 0.0299792741961926
,
 'DEF_30_CNT_SOCIAL_CIRCLE+MONTHS_BALANCE': 0.05592841681866118,
 'DEF_60_CNT_SOCIAL_CIRCLE+CNT_INSTALMENT': -0.035322887141405426,
 'DEF_60_CNT_SOCIAL_CIRCLE+CNT_INSTALMENT_FUTURE': 0.0295563156983649
56,
 'DEF_60_CNT_SOCIAL_CIRCLE+MONTHS_BALANCE': 0.05582499657476691,
 'EXT_SOURCE_2+CNT_INSTALMENT': -0.0370728444619872,
 'EXT_SOURCE_2+CNT_INSTALMENT_FUTURE': 0.023343608696319334,
```

```
 'EXT_SOURCE_2+MONTHS_BALANCE': 0.054430592826276145,
 'EXT_SOURCE_3+CNT_INSTALMENT': -0.033608846915619855,
 'EXT_SOURCE_3+CNT_INSTALMENT_FUTURE': 0.02720758777102746,
 'EXT_SOURCE_3+MONTHS_BALANCE': 0.05055271484092452,
 'OBS_30_CNT_SOCIAL_CIRCLE+CNT_INSTALMENT': -0.034564699484788276,
 'OBS_30_CNT_SOCIAL_CIRCLE+CNT_INSTALMENT_FUTURE': 0.028935055323351704,
 'OBS_30_CNT_SOCIAL_CIRCLE+MONTHS_BALANCE': 0.056209195735033815,
 'OBS_60_CNT_SOCIAL_CIRCLE+CNT_INSTALMENT': -0.03458808595242933,
 'OBS_60_CNT_SOCIAL_CIRCLE+CNT_INSTALMENT_FUTURE': 0.028895753470910383,
 'OBS_60_CNT_SOCIAL_CIRCLE+MONTHS_BALANCE': 0.05619616761983011,
 'REGION_POPULATION_RELATIVE+CNT_INSTALMENT': -0.03582173040988288,
 'REGION_POPULATION_RELATIVE+CNT_INSTALMENT_FUTURE': 0.027752817896447856,
 'REGION_POPULATION_RELATIVE+MONTHS_BALANCE': 0.055291985520445903}
```

Instalment Payments

In [ ]:
```
IP_df = datasets['installments_payments'].groupby('SK_ID_CURR').agg({
    "AMT_INSTALMENT": "sum",
    "AMT_PAYMENT": "sum",
    "DAYS_INSTALMENT": "min",
    "DAYS_ENTRY_PAYMENT": "min",
})
IP_df["SUM_MISSED"] = IP_df["AMT_INSTALMENT"] - IP_df["AMT_PAYMENT"]
```

In [ ]:
```
comb_app = datasets['application_train'].merge(IP_df, how='left', on='
```

In [ ]:
```
new_features = ["AMT_INSTALMENT", "AMT_PAYMENT", "DAYS_INSTALMENT", "D
```

In [ ]:
```
test_df = pd.DataFrame()
corrs = {}

# this is an example for addition. Due to memory constraints, we can c
# one operation (+,-,/,*) per run
for key1 in all_num_features:
  for key2 in new_features:
    if key1 != key2:
      test_df[key1+"-"+key2] = comb_app[key1] - comb_app[key2]
      corrs[key1+"-"+key2] = test_df[key1+"-"+key2].corr(comb_app['TAR

test_df
```

Out[52]:

	CNT_CHILDREN-AMT_INSTALMENT	CNT_CHILDREN-AMT_PAYMENT	CNT_CHILDREN-DAYS_INSTALMENT	CNT_CHILDREN-DAYS_ENTRY_PAYMENT	CN
0	-219625.695	-219625.695	565.0	587.0	
1	-1618864.650	-1618864.650	2310.0	2324.0	
2	-21288.465	-21288.465	784.0	795.0	
3	-1007153.415	-1007153.415	545.0	575.0	
4	-835985.340	-806127.975	2326.0	2318.0	
...	...	...	...	...	
307506	-52450.470	-52450.470	210.0	237.0	
307507	-60419.205	-60419.205	2466.0	2470.0	
307508	-61595.910	-57622.815	2915.0	2915.0	
307509	-194556.825	-194556.825	291.0	317.0	
307510	-3068388.810	-3525819.975	960.0	972.0	

307511 rows × 120 columns

In [ ]:
```
corrs
```

Bureau

In [ ]:
```
B_df = datasets['bureau'].groupby('SK_ID_CURR').agg({
    "CREDIT_TYPE": "min",
    "CREDIT_ACTIVE": "max",
    "DAYS_CREDIT": "mean",
    "AMT_CREDIT_SUM": "max",
})
```

In [ ]:
```
comb_app = datasets['application_train'].merge(B_df, how='left', on='S
```

In [ ]:
```
new_features = ["DAYS_CREDIT", "AMT_CREDIT_SUM"]
```

In [ ]:
```
test_df = pd.DataFrame()
corrs = {}

# this is an example for addition. Due to memory constraints, we can c
# one operation (+,-,/,*) per run
for key1 in all_num_features:
  for key2 in new_features:
    if key1 != key2:
      test_df[key1+"*"+key2] = comb_app[key1] * comb_app[key2]
      corrs[key1+"*"+key2] = test_df[key1+"*"+key2].corr(comb_app['TAR

test_df
```

Out[57]:

	CNT_CHILDREN*DAYS_CREDIT	CNT_CHILDREN*AMT_CREDIT_SUM	AMT_INCOME_TOTA
0	-0.0	0.0	
1	-0.0	0.0	
2	-0.0	0.0	
3	NaN	NaN	
4	-0.0	0.0	
...	...	...	
307506	NaN	NaN	
307507	NaN	NaN	
307508	-0.0	0.0	
307509	-0.0	0.0	
307510	-0.0	0.0	

307511 rows × 48 columns

In [ ]:
```
corrs
```

Out[58]:
```
{'AMT_ANNUITY*AMT_CREDIT_SUM': -0.016500855416944602,
 'AMT_ANNUITY*DAYS_CREDIT': 0.06366637507783798,
 'AMT_CREDIT*AMT_CREDIT_SUM': -0.020905415248609192,
 'AMT_CREDIT*DAYS_CREDIT': 0.06440242758103888,
 'AMT_GOODS_PRICE*AMT_CREDIT_SUM': -0.02227517007935924,
 'AMT_GOODS_PRICE*DAYS_CREDIT': 0.06960594936255725,
 'AMT_INCOME_TOTAL*AMT_CREDIT_SUM': -0.013364529671712626,
 'AMT_INCOME_TOTAL*DAYS_CREDIT': 0.051039812692558986,
 'AMT_REQ_CREDIT_BUREAU_DAY*AMT_CREDIT_SUM': -0.0003910004430030086,
 'AMT_REQ_CREDIT_BUREAU_DAY*DAYS_CREDIT': 0.0008101843356715257,
 'AMT_REQ_CREDIT_BUREAU_HOUR*AMT_CREDIT_SUM': -0.0009122680340084818,
 'AMT_REQ_CREDIT_BUREAU_HOUR*DAYS_CREDIT': 0.003012818168872427,
 'AMT_REQ_CREDIT_BUREAU_MON*AMT_CREDIT_SUM': -0.008367377934013686,
 'AMT_REQ_CREDIT_BUREAU_MON*DAYS_CREDIT': 0.020320771422730508,
 'AMT_REQ_CREDIT_BUREAU_QRT*AMT_CREDIT_SUM': -0.002262514401953606,
 'AMT_REQ_CREDIT_BUREAU_QRT*DAYS_CREDIT': 0.01603539595100403,
 'AMT_REQ_CREDIT_BUREAU_WEEK*AMT_CREDIT_SUM': -0.0038776888732083973,
 'AMT_REQ_CREDIT_BUREAU_WEEK*DAYS_CREDIT': 0.006953461442298675,
 'AMT_REQ_CREDIT_BUREAU_YEAR*AMT_CREDIT_SUM': -0.002714678441150 5058,
 'AMT_REQ_CREDIT_BUREAU_YEAR*DAYS_CREDIT': 0.02855886766556511,
 'CNT_CHILDREN*AMT_CREDIT_SUM': -0.009881917379479495,
 'CNT_CHILDREN*DAYS_CREDIT': 0.007871761218397406,
 'CNT_FAM_MEMBERS*AMT_CREDIT_SUM': -0.019123371064246997,
 'CNT_FAM_MEMBERS*DAYS_CREDIT': 0.06114151245982924,
 'DAYS_BIRTH*AMT_CREDIT_SUM': 0.025527547343085155,
 'DAYS_BIRTH*DAYS_CREDIT': -0.10053345766189184,
 'DAYS_EMPLOYED*AMT_CREDIT_SUM': -0.020276800638660445,
 'DAYS_EMPLOYED*DAYS_CREDIT': 0.04843904021467812,
 'DAYS_ID_PUBLISH*AMT_CREDIT_SUM': 0.023672160478421866,
 'DAYS_ID_PUBLISH*DAYS_CREDIT': -0.08024677015412683,
 'DAYS_LAST_PHONE_CHANGE*AMT_CREDIT_SUM': 0.021997870802034646,
 'DAYS_LAST_PHONE_CHANGE*DAYS_CREDIT': -0.07020737978717018,
 'DAYS_REGISTRATION*AMT_CREDIT_SUM': 0.018602627681851606,
 'DAYS_REGISTRATION*DAYS_CREDIT': -0.07190126984906592,
 'DEF_30_CNT_SOCIAL_CIRCLE*AMT_CREDIT_SUM': 0.006882630327600968,
 'DEF_30_CNT_SOCIAL_CIRCLE*DAYS_CREDIT': -0.009964622033870822,
 'DEF_60_CNT_SOCIAL_CIRCLE*AMT_CREDIT_SUM': 0.006329668934520675,
 'DEF_60_CNT_SOCIAL_CIRCLE*DAYS_CREDIT': -0.011357319583921662,
 'EXT_SOURCE_2*AMT_CREDIT_SUM': -0.0353267667010062 8,
 'EXT_SOURCE_2*DAYS_CREDIT': 0.13692178961644447,
 'EXT_SOURCE_3*AMT_CREDIT_SUM': -0.04115361239033155,
 'EXT_SOURCE_3*DAYS_CREDIT': 0.133669857839854,
 'OBS_30_CNT_SOCIAL_CIRCLE*AMT_CREDIT_SUM': -0.0029326259041254155,
 'OBS_30_CNT_SOCIAL_CIRCLE*DAYS_CREDIT': 0.01527504955558998,
 'OBS_60_CNT_SOCIAL_CIRCLE*AMT_CREDIT_SUM': -0.0029982582074333628,
 'OBS_60_CNT_SOCIAL_CIRCLE*DAYS_CREDIT': 0.015317304227677828,
 'REGION_POPULATION_RELATIVE*AMT_CREDIT_SUM': -0.019574014053956163,
 'REGION_POPULATION_RELATIVE*DAYS_CREDIT': 0.07094202515755103}
```

Bureau Balance dataset

In [ ]:
```
BB_df = datasets['bureau_balance'].groupby('SK_ID_BUREAU').agg({
    "MONTHS_BALANCE": "min",
    "STATUS": ["max", "min", "count"]
})
```

In [ ]:
```
temp = pd.DataFrame({"MONTHS_BALANCE_MIN": BB_df["MONTHS_BALANCE"]["mi
BB_df = temp
```

In [ ]:
```
BB_to_B_df =  datasets['bureau'].merge(BB_df, how='left', on='SK_ID_BU
BB_to_B_df = BB_to_B_df.dropna(subset = ["STATUS_MAX","STATUS_MIN"])
B_df_temp = BB_to_B_df.groupby('SK_ID_CURR').agg({
    "MONTHS_BALANCE_MIN": "min",
    "STATUS_MIN": "min",
    "STATUS_MAX" : 'max',
    "STATUS_COUNT": "count"
})
```

In [ ]:
```
B_df = pd.concat([B_df, B_df_temp], axis=1)
```

Credit Card Balances

In [ ]:
```
temp_app = datasets['application_train'].merge(B_df_temp, how='left',
```

In [ ]:
```
CCB_df = datasets['credit_card_balance'].groupby('SK_ID_CURR').agg({
    "AMT_BALANCE": "mean",
    "MONTHS_BALANCE": "min",
    "AMT_CREDIT_LIMIT_ACTUAL": "count",
})
```

In [ ]:
```
comb_app = datasets['application_train'].merge(CCB_df, how='left', on=
```

In [ ]:
```
new_features = ["AMT_BALANCE", "MONTHS_BALANCE", "AMT_CREDIT_LIMIT_ACT
```

In [ ]:
```
test_df = pd.DataFrame()
corrs = {}

# this is an example for addition. Due to memory constraints, we can c
# one operation (+,-,/,*) per run
for key1 in all_num_features:
  for key2 in new_features:
    if key1 != key2:
      test_df[key1+"+"+key2] = comb_app[key1] + comb_app[key2]
      corrs[key1+"+"+key2] = test_df[key1+"+"+key2].corr(comb_app['TAR

test_df
```

Out[67]:

	CNT_CHILDREN+AMT_BALANCE	CNT_CHILDREN+MONTHS_BALANCE	CNT_CHILDREN+
0	NaN	NaN	
1	NaN	NaN	
2	NaN	NaN	
3	0.0	-6.0	
4	NaN	NaN	
...	...	...	
307506	NaN	NaN	
307507	NaN	NaN	
307508	NaN	NaN	
307509	NaN	NaN	
307510	NaN	NaN	

307511 rows × 72 columns

In [ ]:
```
corrs
```

Out[68]:
```
{'AMT_ANNUITY+AMT_BALANCE': 0.08371049424218174,
 'AMT_ANNUITY+AMT_CREDIT_LIMIT_ACTUAL': -0.025063558915545698,
 'AMT_ANNUITY+MONTHS_BALANCE': -0.024771390825012576,
 'AMT_CREDIT+AMT_BALANCE': -0.009802925077219877,
 'AMT_CREDIT+AMT_CREDIT_LIMIT_ACTUAL': -0.03503374158665453,
 'AMT_CREDIT+MONTHS_BALANCE': -0.03502380865670211,
 'AMT_GOODS_PRICE+AMT_BALANCE': -0.015940860495845664,
 'AMT_GOODS_PRICE+AMT_CREDIT_LIMIT_ACTUAL': -0.04367573383904863,
 'AMT_GOODS_PRICE+MONTHS_BALANCE': -0.04366514956971031,
 'AMT_INCOME_TOTAL+AMT_BALANCE': 0.048450928829204605,
 'AMT_INCOME_TOTAL+AMT_CREDIT_LIMIT_ACTUAL': -0.017685313734971734,
 'AMT_INCOME_TOTAL+MONTHS_BALANCE': -0.0176466574023929,
 'AMT_REQ_CREDIT_BUREAU_DAY+AMT_BALANCE': 0.08599405922754702,
 'AMT_REQ_CREDIT_BUREAU_DAY+AMT_CREDIT_LIMIT_ACTUAL': -0.058841283458
```

```
255025,
 'AMT_REQ_CREDIT_BUREAU_DAY+MONTHS_BALANCE': 0.05975592643099151,
 'AMT_REQ_CREDIT_BUREAU_HOUR+AMT_BALANCE': 0.08599405852655541,
 'AMT_REQ_CREDIT_BUREAU_HOUR+AMT_CREDIT_LIMIT_ACTUAL': -0.05884224667
265547,
 'AMT_REQ_CREDIT_BUREAU_HOUR+MONTHS_BALANCE': 0.0597551572033678,
 'AMT_REQ_CREDIT_BUREAU_MON+AMT_BALANCE': 0.08599386643022511,
 'AMT_REQ_CREDIT_BUREAU_MON+AMT_CREDIT_LIMIT_ACTUAL': -0.059332965879
652956,
 'AMT_REQ_CREDIT_BUREAU_MON+MONTHS_BALANCE': 0.059208949114249725,
 'AMT_REQ_CREDIT_BUREAU_QRT+AMT_BALANCE': 0.08599401691771945,
 'AMT_REQ_CREDIT_BUREAU_QRT+AMT_CREDIT_LIMIT_ACTUAL': -0.059024443743
87306,
 'AMT_REQ_CREDIT_BUREAU_QRT+MONTHS_BALANCE': 0.05950026831492447,
 'AMT_REQ_CREDIT_BUREAU_WEEK+AMT_BALANCE': 0.08599405902845389,
 'AMT_REQ_CREDIT_BUREAU_WEEK+AMT_CREDIT_LIMIT_ACTUAL': -0.05884545267
34776,
 'AMT_REQ_CREDIT_BUREAU_WEEK+MONTHS_BALANCE': 0.059749492776338245,
 'AMT_REQ_CREDIT_BUREAU_YEAR+AMT_BALANCE': 0.0859943408372072,
 'AMT_REQ_CREDIT_BUREAU_YEAR+AMT_CREDIT_LIMIT_ACTUAL': -0.05786657602
9764576,
 'AMT_REQ_CREDIT_BUREAU_YEAR+MONTHS_BALANCE': 0.060524196891697656,
 'CNT_CHILDREN+AMT_BALANCE': 0.0871773200560069,
 'CNT_CHILDREN+AMT_CREDIT_LIMIT_ACTUAL': -0.06015646852519142,
 'CNT_CHILDREN+MONTHS_BALANCE': 0.06165421978175474,
 'CNT_FAM_MEMBERS+AMT_BALANCE': 0.08717729242917416,
 'CNT_FAM_MEMBERS+AMT_CREDIT_LIMIT_ACTUAL': -0.06020014822737341,
 'CNT_FAM_MEMBERS+MONTHS_BALANCE': 0.061595942635955345,
 'DAYS_BIRTH+AMT_BALANCE': 0.08930195615011875,
 'DAYS_BIRTH+AMT_CREDIT_LIMIT_ACTUAL': 0.0667169352971932,
 'DAYS_BIRTH+MONTHS_BALANCE': 0.06756082765439705,
 'DAYS_EMPLOYED+AMT_BALANCE': 0.028350672268488895,
 'DAYS_EMPLOYED+AMT_CREDIT_LIMIT_ACTUAL': -0.03737298956325402,
 'DAYS_EMPLOYED+MONTHS_BALANCE': -0.03734235500487899,
 'DAYS_ID_PUBLISH+AMT_BALANCE': 0.08775816565874107,
 'DAYS_ID_PUBLISH+AMT_CREDIT_LIMIT_ACTUAL': 0.04463289768094621,
 'DAYS_ID_PUBLISH+MONTHS_BALANCE': 0.047153570595337126,
 'DAYS_LAST_PHONE_CHANGE+AMT_BALANCE': 0.0877219703586256,
 'DAYS_LAST_PHONE_CHANGE+AMT_CREDIT_LIMIT_ACTUAL': 0.0685854134867341
4,
 'DAYS_LAST_PHONE_CHANGE+MONTHS_BALANCE': 0.07165195246297092,
 'DAYS_REGISTRATION+AMT_BALANCE': 0.08826016587247477,
 'DAYS_REGISTRATION+AMT_CREDIT_LIMIT_ACTUAL': 0.037620262466671044,
 'DAYS_REGISTRATION+MONTHS_BALANCE': 0.038737361009939145,
 'DEF_30_CNT_SOCIAL_CIRCLE+AMT_BALANCE': 0.08719248874954717,
 'DEF_30_CNT_SOCIAL_CIRCLE+AMT_CREDIT_LIMIT_ACTUAL': -0.0601999286287
3247,
 'DEF_30_CNT_SOCIAL_CIRCLE+MONTHS_BALANCE': 0.06176236184358404,
 'DEF_60_CNT_SOCIAL_CIRCLE+AMT_BALANCE': 0.08719247567028857,
 'DEF_60_CNT_SOCIAL_CIRCLE+AMT_CREDIT_LIMIT_ACTUAL': -0.0602570386877
3687,
 'DEF_60_CNT_SOCIAL_CIRCLE+MONTHS_BALANCE': 0.06170878301466095,
 'EXT_SOURCE_2+AMT_BALANCE': 0.08716875220628945,
```

```
 'EXT_SOURCE_2+AMT_CREDIT_LIMIT_ACTUAL': -0.06137410860418245,
 'EXT_SOURCE_2+MONTHS_BALANCE': 0.06051594622496881,
 'EXT_SOURCE_3+AMT_BALANCE': 0.08641605147897537,
 'EXT_SOURCE_3+AMT_CREDIT_LIMIT_ACTUAL': -0.0600618322062925,
 'EXT_SOURCE_3+MONTHS_BALANCE': 0.05902099627625955,
 'OBS_30_CNT_SOCIAL_CIRCLE+AMT_BALANCE': 0.0871927080878983,
 'OBS_30_CNT_SOCIAL_CIRCLE+AMT_CREDIT_LIMIT_ACTUAL': -0.0592043434279
0032,
 'OBS_30_CNT_SOCIAL_CIRCLE+MONTHS_BALANCE': 0.062469375703352675,
 'OBS_60_CNT_SOCIAL_CIRCLE+AMT_BALANCE': 0.08719270006725602,
 'OBS_60_CNT_SOCIAL_CIRCLE+AMT_CREDIT_LIMIT_ACTUAL': -0.0592287616511
3408,
 'OBS_60_CNT_SOCIAL_CIRCLE+MONTHS_BALANCE': 0.06244991391179811,
 'REGION_POPULATION_RELATIVE+AMT_BALANCE': 0.08717721556856706,
 'REGION_POPULATION_RELATIVE+AMT_CREDIT_LIMIT_ACTUAL': -0.06049488034
111936,
 'REGION_POPULATION_RELATIVE+MONTHS_BALANCE': 0.06134455443658401}
```

In [ ]:
```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')
```

In [ ]:
```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_
]
```

In [ ]:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='most_frequent')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

full_pipeline = Pipeline([
  ('preprocessing', preprocess_pipeline),
  ('linear', LogisticRegression())
])
```

In [ ]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from time import time
X_train = train.loc[:, train.columns != "TARGET"]
y_train = train['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,

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

cv10Splits = ShuffleSplit(n_splits = 10, test_size = .3, random_state
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full
logit_score_train = logit_scores.mean()

# Time and score test predictions
start = time()
logit_score_test  = full_pipeline.score(X_test, y_test)
roc = roc_auc_score(y_test, full_pipeline.predict_proba(X_test)[:, 1])
test_time = np.round(time() - start, 4)
```

In [ ]:
```
results.loc[2] = ["Baseline", roc,logit_score_train, np.round(logit_sc
                  train_time, test_time, "LogisticRegression + other
results
```

Out[74]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	0.919960	0.917	9.6565	0.3952	LogisticRegression
1	Baseline	0.739299	92.000000	91.700	8.6859	0.3853	LogisticRegression + new Application features
2	Baseline	0.740311	0.919982	0.917	20.9558	0.5814	LogisticRegression + other datasets

This doesn't really improve our ROC_AUC score very much

Baseline with new feature for all data

In [ ]:
```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_PO
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DE
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + tra
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DE
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + tr
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_PO

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_C
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONT
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CR
```

In [ ]:
```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT
]
```

In [ ]:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='most_frequent')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

full_pipeline = Pipeline([
  ('preprocessing', preprocess_pipeline),
  ('linear', LogisticRegression())
])
```

In [ ]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from time import time
X_train = train.loc[:, train.columns != "TARGET"]
y_train = train['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,

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

cv10Splits = ShuffleSplit(n_splits = 10, test_size = .3, random_state
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full
logit_score_train = logit_scores.mean()

# Time and score test predictions
start = time()
logit_score_test  = full_pipeline.score(X_test, y_test)
roc = roc_auc_score(y_test, full_pipeline.predict_proba(X_test)[:, 1])
test_time = np.round(time() - start, 4)
```

In [ ]:
```
results.loc[3] = ["Baseline", roc,logit_score_train, np.round(logit_sc
                  train_time, test_time, "LogisticRegression + other
results
```

Out[79]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	0.919960	0.917	9.6565	0.3952	LogisticRegression
1	Baseline	0.739299	92.000000	91.700	8.6859	0.3853	LogisticRegression + new Application features
2	Baseline	0.740311	0.919982	0.917	20.9558	0.5814	LogisticRegression + other datasets
3	Baseline	0.745049	92.000000	91.700	23.2227	0.6177	LogisticRegression + other datasets + new feat...

In [ ]:
```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_PO
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DE
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + tra
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DE
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + tr
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_PO

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_C
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONT
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CR
```

In [ ]:
```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG"
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT
]
```

In [ ]:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='most_frequent')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

full_pipeline = Pipeline([
  ('preprocessing', preprocess_pipeline),
  ('linear', LogisticRegression())
])
```

In [ ]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from time import time
X_train = train.loc[:, train.columns != "TARGET"]
y_train = train['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,

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

cv10Splits = ShuffleSplit(n_splits = 10, test_size = .3, random_state
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full
logit_score_train = logit_scores.mean()

# Time and score test predictions
start = time()
logit_score_test  = full_pipeline.score(X_test, y_test)
roc = roc_auc_score(y_test, full_pipeline.predict_proba(X_test)[:, 1])
test_time = np.round(time() - start, 4)
```

In [ ]:
```
results.loc[4] = ["Baseline", roc,pct(logit_score_train), np.round(pct
                  train_time, test_time, "LogisticRegression + even m
results
```

Out[84]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	0.919960	0.917	9.6565	0.3952	LogisticRegression
1	Baseline	0.739299	92.000000	91.700	8.6859	0.3853	LogisticRegression + new Application features
2	Baseline	0.740311	0.919982	0.917	20.9558	0.5814	LogisticRegression + other datasets
3	Baseline	0.745049	92.000000	91.700	23.2227	0.6177	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92.000000	91.700	17.7821	0.6412	LogisticRegression + even more data

Tweaking Imputers

Perhaps we might employ a constant strategy with the categorical imputer. Rather than putting NaN data to the most common category, perhaps we could create a new category for all of this information. This would deal with particular categories, such as employment data types, where unemployed clients appeared to be tagged as NaN and shouldn't be grouped with other categories.

In [ ]:
```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_PO
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DE
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + tra
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DE
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + tr
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_PO

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_C
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONT
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CR
```

In [ ]:

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT
]
```

In [ ]:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

full_pipeline = Pipeline([
  ('preprocessing', preprocess_pipeline),
  ('linear', LogisticRegression())
])
```

In [ ]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from time import time
X_train = train.loc[:, train.columns != "TARGET"]
y_train = train['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,

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

cv10Splits = ShuffleSplit(n_splits = 10, test_size = .3, random_state
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full
logit_score_train = logit_scores.mean()

# Time and score test predictions
start = time()
logit_score_test  = full_pipeline.score(X_test, y_test)
roc = roc_auc_score(y_test, full_pipeline.predict_proba(X_test)[:, 1])
test_time = np.round(time() - start, 4)
```

In [ ]:
```
results.loc[5] = ["Baseline", roc,logit_score_train, np.round(logit_sc
                  train_time, test_time, "LogisticRegression w/ Const
results
```

Out[89]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	0.919960	0.917	9.6565	0.3952	LogisticRegression
1	Baseline	0.739299	92.000000	91.700	8.6859	0.3853	LogisticRegression + new Application features
2	Baseline	0.740311	0.919982	0.917	20.9558	0.5814	LogisticRegression + other datasets
3	Baseline	0.745049	92.000000	91.700	23.2227	0.6177	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92.000000	91.700	17.7821	0.6412	LogisticRegression + even more data
5	Baseline	0.747196	92.000000	91.700	13.4003	0.9545	LogisticRegression w/ Constant Imputer

We're comparing this test to experiment 3 because it doesn't include the "even more data" from the prior test, and it performs better. In the future, we should keep employing this adjustment to the imputer.

Untuned LGBM

Let's train an LGBM Classifier on the data from application plus the data from our datasets and new engineered features

In [ ]:
```
from lightgbm import LGBMClassifier
```

In [ ]:
```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_PO
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DE
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + tra
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DE
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + tr
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_PO

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_C
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONT
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CR
```

In [ ]:

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT
]
```

In [ ]:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

full_pipeline = Pipeline([
  ('preprocessing', preprocess_pipeline),
  ('predictor', LGBMClassifier())
])
```

In [ ]:

```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from time import time
X_train = train.loc[:, train.columns != "TARGET"]
y_train = train['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,

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

cv10Splits = ShuffleSplit(n_splits = 10, test_size = .3, random_state
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full
logit_score_train = logit_scores.mean()

# Time and score test predictions
start = time()
logit_score_test  = full_pipeline.score(X_test, y_test)
roc = roc_auc_score(y_test, full_pipeline.predict_proba(X_test)[:, 1])
test_time = np.round(time() - start, 4)
```

In [ ]:
```
results.loc[6] = ["LGBM", roc,logit_score_train, np.round(logit_score_
                  train_time, test_time, "Untuned LGBM"]
results
```

Out[111]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	0.919960	0.917	9.6565	0.3952	LogisticRegression
1	Baseline	0.739299	92.000000	91.700	8.6859	0.3853	LogisticRegression + new Application features
2	Baseline	0.740311	0.919982	0.917	20.9558	0.5814	LogisticRegression + other datasets
3	Baseline	0.745049	92.000000	91.700	23.2227	0.6177	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92.000000	91.700	17.7821	0.6412	LogisticRegression + even more data
5	Baseline	0.747196	92.000000	91.700	13.4003	0.9545	LogisticRegression w/ Constant Imputer
6	LGBM	0.764011	0.920048	0.918	28.4584	2.2212	Untuned LGBM
7	LGBM	0.763808	92.000000	91.800	58.8361	1.2368	Untuned LGBM + aggregated datasets
8	LGBM	0.764011	92.000000	91.800	28.4584	2.2212	LGBM tuned

As you can see, it outperforms all other models we've seen so far. We still need to fine-tune it, and we can add more data if necessary.

Adding New Dataset Features

These characteristics result from combining the Bureau dataset with the 'bureau_balance' dataset and utilizing SK_ID_PREV to obtain additional data for the previous application dataset.

In [ ]:

```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_PO
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DE
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + tra
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DE
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + tr
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_PO

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_C
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONT
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CR
```

In [ ]:

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG"
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

In [ ]:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

full_pipeline = Pipeline([
  ('preprocessing', preprocess_pipeline),
  ('predictor', LGBMClassifier())
])
```

In [ ]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from time import time
X_train = train.loc[:, train.columns != "TARGET"]
y_train = train['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,

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

cv10Splits = ShuffleSplit(n_splits = 10, test_size = .3, random_state
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full
logit_score_train = logit_scores.mean()

# Time and score test predictions
start = time()
logit_score_test  = full_pipeline.score(X_test, y_test)
roc = roc_auc_score(y_test, full_pipeline.predict_proba(X_test)[:, 1])
test_time = np.round(time() - start, 4)
```

In [ ]:
```
results.loc[7] = ["LGBM", roc,pct(logit_score_train), np.round(pct(log
                  train_time, test_time, "Untuned LGBM + aggregated d
results
```

Out[100]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	0.919960	0.917	9.6565	0.3952	LogisticRegression
1	Baseline	0.739299	92.000000	91.700	8.6859	0.3853	LogisticRegression + new Application features
2	Baseline	0.740311	0.919982	0.917	20.9558	0.5814	LogisticRegression + other datasets
3	Baseline	0.745049	92.000000	91.700	23.2227	0.6177	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92.000000	91.700	17.7821	0.6412	LogisticRegression + even more data
5	Baseline	0.747196	92.000000	91.700	13.4003	0.9545	LogisticRegression w/ Constant Imputer
6	LGBM	0.755898	92.000000	91.700	55.4857	0.9255	Untuned LGBM
7	LGBM	0.763808	92.000000	91.800	58.8361	1.2368	Untuned LGBM + aggregated datasets

These new additions unquestionably improve our model. Now we must use Grid Search to optimize it.

Grid Search for LGBM

We checked all other hyperparameters before evaluating n_estimators because Grid Search takes a long time when testing for greater values of n_estimators.

In [ ]:
```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_PO
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DE
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + tra
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DE
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + tr
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_PO

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_C
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONT
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CR
```

In [ ]:

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG"
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

In [ ]:

```
m sklearn.base import BaseEstimator, TransformerMixin
m sklearn.pipeline import Pipeline
m sklearn.pipeline import FeatureUnion
m sklearn.preprocessing import StandardScaler
m sklearn.impute import SimpleImputer
m sklearn.preprocessing import OneHotEncoder
m sklearn.linear_model import LogisticRegression
m sklearn.model_selection import GridSearchCV
m lightgbm import LGBMClassifier

ustom layer to get columns we want from DataFrame
ss DataFrameSelector(BaseEstimator, TransformerMixin):
ef __init__(self, attribute_names):
 self.attribute_names = attribute_names
ef fit(self, X, y=None):
 return self
ef transform(self, X):
 return X[self.attribute_names].values
 pct(x):
 return round(100*x,1)
_pipeline = Pipeline([
'selector', DataFrameSelector(num_features)),
'imputer', SimpleImputer(strategy='median')),
'std_scaler', StandardScaler()),


_pipeline = Pipeline([
'selector', DataFrameSelector(cat_features)),
'imputer', SimpleImputer(strategy='constant')),
'ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),


process_pipeline = FeatureUnion(transformer_list=[
"num_pipeline", num_pipeline),
"cat_pipeline", cat_pipeline),


l_pipeline_with_predictor = Pipeline([
"preprocessing", preprocess_pipeline),
"predictor", LGBMClassifier(colsample_bytree=0.1, max_depth=5, n_estim


xecute the grid search
ams = {
 'predictor__colsample_bytree': [0.0, 0.1, 0.2, 0.5],
 'predictor__max_depth': [-1, 3, 5, 10],
 'predictor__min_split_gain': [0.0, 0.5, 1],
 'predictor__num_leaves': [10, 20, 31, 42],


d_search = GridSearchCV(full_pipeline_with_predictor, params, scoring=
                        n_jobs=-1, verbose=True)
```

Tuned LGBM

Let's combine the extra data from the "More Data" experiment with an LGBM model that has been optimized for hyperparameters.

In [ ]:
```
from lightgbm import LGBMClassifier
```

In [ ]:
```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_PO
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DE
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + tra
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DE
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + tr
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_PO

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_C
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONT
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CR
```

In [ ]:

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG"
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

In [ ]:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

full_pipeline = Pipeline([
  ('preprocessing', preprocess_pipeline),
  ('predictor', LGBMClassifier(colsample_bytree=0.5, max_depth=10, n_e
])
```

In [ ]:
```
learn.model_selection import ShuffleSplit
learn.model_selection import cross_val_score
learn.model_selection import train_test_split
learn.metrics import roc_auc_score
me import time
 = train.loc[:, train.columns != "TARGET"]
 = train['TARGET']
, X_test, y_train, y_test = train_test_split(X_train, y_train, test_siz

 time()
peline.fit(X_train, y_train)
ime = np.round(time() - start, 4)

its = ShuffleSplit(n_splits = 10, test_size = .3, random_state = 0)
cores = cross_val_score(X=X_train,y = y_train, estimator = full_pipelin
core_train = logit_scores.mean()

and score test predictions
 time()
core_test  = full_pipeline.score(X_test, y_test)
oc_auc_score(y_test, full_pipeline.predict_proba(X_test)[:, 1])
me = np.round(time() - start, 4)
```

In [ ]:
```
results.loc[8] = ["LGBM", roc,logit_score_train, np.round(logit_score_
                  train_time, test_time, "LGBM tuned"]
results
```

Out[109]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	0.919960	0.917	9.6565	0.3952	LogisticRegression
1	Baseline	0.739299	92.000000	91.700	8.6859	0.3853	LogisticRegression + new Application features
2	Baseline	0.740311	0.919982	0.917	20.9558	0.5814	LogisticRegression + other datasets
3	Baseline	0.745049	92.000000	91.700	23.2227	0.6177	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92.000000	91.700	17.7821	0.6412	LogisticRegression + even more data
5	Baseline	0.747196	92.000000	91.700	13.4003	0.9545	LogisticRegression w/ Constant Imputer
6	LGBM	0.755898	92.000000	91.700	55.4857	0.9255	Untuned LGBM
7	LGBM	0.763808	92.000000	91.800	58.8361	1.2368	Untuned LGBM + aggregated datasets
8	LGBM	0.764011	92.000000	91.800	28.4584	2.2212	LGBM tuned

In [ ]:
```
# from matplotlib import pyplot as plt
model = full_pipeline.steps[1][1]
features = list(full_pipeline.steps[0][1].transformer_list[1][1].steps

pd.DataFrame({'Value':model.feature_importances_,'Feature':features}).
```

Out[110]:

	Value	Feature
60	0	OCCUPATION_TYPE_Secretaries
59	0	OCCUPATION_TYPE_Sales staff
58	0	OCCUPATION_TYPE_Realty agents
52	0	OCCUPATION_TYPE_IT staff
89	0	CREDIT_TYPE_Car loan
91	0	CREDIT_TYPE_Consumer credit
131	0	DAYS_BIRTH
112	0	STATUS_MIN_X
129	0	ELEVATORS_AVG
128	0	REGION_POPULATION_RELATIVE
127	0	AMT_GOODS_PRICE
126	0	FLOORSMAX_AVG
117	0	STATUS_MAX_3
115	0	STATUS_MAX_1
114	0	STATUS_MAX_0
110	0	STATUS_MIN_5
92	0	CREDIT_TYPE_Credit card
109	0	STATUS_MIN_4
108	0	STATUS_MIN_3
102	0	CREDIT_ACTIVE_Closed
100	0	CREDIT_ACTIVE_Active
97	0	CREDIT_TYPE_Real estate loan
96	0	CREDIT_TYPE_Mortgage
94	0	CREDIT_TYPE_Loan for working capital replenish...
172	0	STATUS_COUNT

Interestingly, some of our finest correlation data, such as FLOORSMIN_AVG and ELEVATORS_AVG, are completely ignored by our model. This could be because they weren't beneficial because so much data was missing for so many samples. This could possibly be due to its strong correlation with other similar traits, such as TOTALAREA MODE.

Regrettably, it appears that CREDIT_ACTIVE was virtually completely useless.

In [ ]:
```
pd.DataFrame({'Value':model.feature_importances_,'Feature':features}).
```

Out[114]:

	Value	Feature
39	605	HOUR_APPR_PROCESS_START_18
0	604	FLAG_DOCUMENT_3_0
1	580	FLAG_DOCUMENT_3_1
9	544	NAME_INCOME_TYPE_Commercial associate
15	533	NAME_INCOME_TYPE_Working
2	518	REGION_RATING_CLIENT_1
42	490	HOUR_APPR_PROCESS_START_21
4	486	REGION_RATING_CLIENT_3
25	483	HOUR_APPR_PROCESS_START_4
22	482	HOUR_APPR_PROCESS_START_1
24	460	HOUR_APPR_PROCESS_START_3
10	450	NAME_INCOME_TYPE_Maternity leave
19	429	NAME_EDUCATION_TYPE_Lower secondary
37	426	HOUR_APPR_PROCESS_START_16
26	407	HOUR_APPR_PROCESS_START_5
8	373	NAME_INCOME_TYPE_Businessman
34	373	HOUR_APPR_PROCESS_START_13
20	370	NAME_EDUCATION_TYPE_Secondary / secondary special
5	346	REGION_RATING_CLIENT_W_CITY_1
30	336	HOUR_APPR_PROCESS_START_9
36	331	HOUR_APPR_PROCESS_START_15
7	329	REGION_RATING_CLIENT_W_CITY_3
35	327	HOUR_APPR_PROCESS_START_14
41	318	HOUR_APPR_PROCESS_START_20
12	302	NAME_INCOME_TYPE_State servant
13	255	NAME_INCOME_TYPE_Student

28	249	HOUR_APPR_PROCESS_START_7
43	233	HOUR_APPR_PROCESS_START_22
29	224	HOUR_APPR_PROCESS_START_8
27	212	HOUR_APPR_PROCESS_START_6
16	202	NAME_EDUCATION_TYPE_Academic degree
40	200	HOUR_APPR_PROCESS_START_19
14	191	NAME_INCOME_TYPE_Unemployed
48	189	OCCUPATION_TYPE_Core staff
44	186	HOUR_APPR_PROCESS_START_23
45	169	OCCUPATION_TYPE_Accountants
31	165	HOUR_APPR_PROCESS_START_10
49	142	OCCUPATION_TYPE_Drivers
33	120	HOUR_APPR_PROCESS_START_12
38	115	HOUR_APPR_PROCESS_START_17
46	114	OCCUPATION_TYPE_Cleaning staff
32	104	HOUR_APPR_PROCESS_START_11
11	89	NAME_INCOME_TYPE_Pensioner
3	65	REGION_RATING_CLIENT_2
6	62	REGION_RATING_CLIENT_W_CITY_2
21	45	HOUR_APPR_PROCESS_START_0
23	38	HOUR_APPR_PROCESS_START_2
99	34	CREDIT_TYPE_missing_value
50	31	OCCUPATION_TYPE_HR staff
130	30	DAYS_LAST_PHONE_CHANGE

The most useful information is categorical. It appears that HOUR_APPR_PROCESS_START and FLAG_DOCUMENT_3 were very useful. Since FLAG DOCUMENT 4 is in the data set, it's intriguing that the flags for the other documents were considerably less useful.

As far as we can tell, there are no significant numerical features.

In [ ]:
```
imp_df = pd.DataFrame({'Value':model.feature_importances_,'Feature':fe
print("Unimportant data points: {}%".format(round(sum(imp_df["Value"]
```

```
Unimportant data points: 35.84%
```

Overall, it appears that our model did not utilise roughly 34% of our features. While some of the numerical features, such as OCCUPATION_TYPE, had data points in both the most useful and least useful features due to the large number of possible values, categorical features, such as OCCUPATION_TYPE, had data points in both the most useful and least useful features due to the large number of possible values.

Buliding Deeplearning Model

In [ ]:
```
import numpy as np
import pandas as pd
import os
import zipfile
import warnings
warnings.filterwarnings('ignore')

def load_data(in_path, name):
    df = pd.read_csv(in_path)
    print(f"{name}: shape is {df.shape}")
    print(df.info())
    display(df.head(3))
    return df

datasets={}  # lets store the datasets in a dictionary so we can keep
DATA_DIR = "home-credit-default-risk" # folder where unzipped files ar

ds_names = ("application_train", "application_test", "bureau","bureau_
            "previous_application","POS_CASH_balance")

for ds_name in ds_names:
    datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.c
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,},
```

dtypes: float64(2), int64(3), object(1)
memory usage: 610.4+ MB
None

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTU
0	1803195	182943	-31	48.0	
1	1715348	367990	-33	36.0	
2	1784872	397406	-32	12.0	

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

Load In Datasets

In [2]:
```
## Previous Applications
```

In [3]:
```
PA_df = datasets['previous_application'].groupby('SK_ID_CURR').agg({
    "AMT_APPLICATION": "mean",
    "CNT_PAYMENT": "max",
    "DAYS_TERMINATION":"mean",

    "NAME_PORTFOLIO": "max",
    "NAME_GOODS_CATEGORY": "max",
    "NAME_SELLER_INDUSTRY": "max",
})
```

In [4]:
```
PCB_df_copy = datasets['POS_CASH_balance'].groupby('SK_ID_PREV').agg({
    "CNT_INSTALMENT": "count",
    "CNT_INSTALMENT_FUTURE": "mean",
    "MONTHS_BALANCE": "min",
})
POS_to_PA_df =  datasets['previous_application'].merge(PCB_df_copy, ho
PCB_df_temp = POS_to_PA_df.groupby('SK_ID_CURR').agg({
    "CNT_INSTALMENT": "count",
    "CNT_INSTALMENT_FUTURE": "mean",
    "MONTHS_BALANCE": "min",
})
PCB_df_temp=PCB_df_temp.rename({"CNT_INSTALMENT":"PREV_CNT_INSTALMENT"
PA_df = pd.concat([PA_df, PCB_df_temp], axis=1)
```

In [5]:
```
IP_df_copy = datasets['installments_payments'].groupby('SK_ID_PREV').a
    "AMT_INSTALMENT": "sum",
    "AMT_PAYMENT": "sum",
    "DAYS_INSTALMENT": "min",
    "DAYS_ENTRY_PAYMENT": "min",
})
IP_df_copy["SUM_MISSED"] = IP_df_copy["AMT_INSTALMENT"] - IP_df_copy["
IP_to_PA_df =  datasets['previous_application'].merge(IP_df_copy, how=
IP_df_temp = IP_to_PA_df.groupby('SK_ID_CURR').agg({
    "AMT_INSTALMENT": "sum",
    "AMT_PAYMENT": "sum",
    "DAYS_INSTALMENT": "min",
    "DAYS_ENTRY_PAYMENT": "min"
})
IP_df_temp = IP_df_temp.rename({"AMT_INSTALMENT":"PREV_AMT_INSTALMENT"
PA_df = pd.concat([PA_df, IP_df_temp], axis=1)
```

In [6]:
```
CCB_df_copy = datasets['credit_card_balance'].groupby('SK_ID_PREV').ag
    "AMT_BALANCE": "mean",
    "MONTHS_BALANCE": "min",
    "AMT_CREDIT_LIMIT_ACTUAL": "count",
})
CCB_to_PA_df =  datasets['previous_application'].merge(CCB_df_copy, ho
CCB_df_temp = CCB_to_PA_df.groupby('SK_ID_CURR').agg({
    "AMT_BALANCE": "mean",
    "MONTHS_BALANCE": "min",
    "AMT_CREDIT_LIMIT_ACTUAL": "count"
})
CCB_df_temp = CCB_df_temp.rename({"AMT_BALANCE":"PREV_AMT_BALANCE", "M
PA_df = pd.concat([PA_df, CCB_df_temp], axis=1)
```

POS Cash Balances

In [7]:
```
PCB_df = datasets['POS_CASH_balance'].groupby('SK_ID_CURR').agg({
    "CNT_INSTALMENT": "count",
    "CNT_INSTALMENT_FUTURE": "mean",
    "MONTHS_BALANCE": "min",
})
```

Instalment Payments

In [8]:
```
IP_df = datasets['installments_payments'].groupby('SK_ID_CURR').agg({
    "AMT_INSTALMENT": "sum",
    "AMT_PAYMENT": "sum",
    "DAYS_INSTALMENT": "min",
    "DAYS_ENTRY_PAYMENT": "min",
})
IP_df["SUM_MISSED"] = IP_df["AMT_INSTALMENT"] - IP_df["AMT_PAYMENT"]
```

Bureau

In [9]:
```
B_df = datasets['bureau'].groupby('SK_ID_CURR').agg({
    "CREDIT_TYPE": "min",
    "CREDIT_ACTIVE": "max",
    "DAYS_CREDIT": "mean",
    "AMT_CREDIT_SUM": "max",
})
```

In [11]:
```
_df = datasets['bureau_balance'].groupby('SK_ID_BUREAU').agg({
  "MONTHS_BALANCE": "min",
  "STATUS": ["max", "min", "count"]

mp = pd.DataFrame({"MONTHS_BALANCE_MIN": BB_df["MONTHS_BALANCE"]["min"
_df = temp
_to_B_df =  datasets['bureau'].merge(BB_df, how='left', on='SK_ID_BURE
_to_B_df = BB_to_B_df.dropna(subset = ["STATUS_MAX","STATUS_MIN"])
df_temp = BB_to_B_df.groupby('SK_ID_CURR').agg({
  "MONTHS_BALANCE_MIN": "min",
  "STATUS_MIN": "min",
  "STATUS_MAX" : 'max',
  "STATUS_COUNT": "count"

df = pd.concat([B_df, B_df_temp], axis=1)
```

In [12]:
```
CCB_df = datasets['credit_card_balance'].groupby('SK_ID_CURR').agg({
    "AMT_BALANCE": "mean",
    "MONTHS_BALANCE": "min",
    "AMT_CREDIT_LIMIT_ACTUAL": "count",
})
```

In [13]:
```
# initialize results table
results = pd.DataFrame(columns=["ExpID", "ROC AUC Score", "Cross fold
```

Deep Learning Model

In [14]:
```
ort torch
ort torch.nn as nn
ort numpy as np
ort torch.optim as optim
m sklearn.model_selection import train_test_split
m time import time

 = datasets["application_train"]
in_x = app.loc[:, app.columns != "TARGET"]
in_y = app["TARGET"]
in_x, test_x, train_y, test_y = train_test_split(train_x, train_y, tes
```

In [15]:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import roc_auc_score

# preprocess data
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU
]

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='most_frequent')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

In [16]:
```
# to tensors
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

In [17]:
```
# globals
batch_size = 64
num_epochs = 100
num_in = train_x.shape[1]
num_layer_1 = 20
num_output = 2
```

In [18]:
```
reate data loaders
in_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_tensor
a_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_siz
```

In [19]:
```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, num_layer_1),
        nn.ReLU(),
        nn.Linear(num_layer_1, num_output)
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.SGD(model.parameters(), lr=0.01)
loss_fn = nn.BCELoss()
```

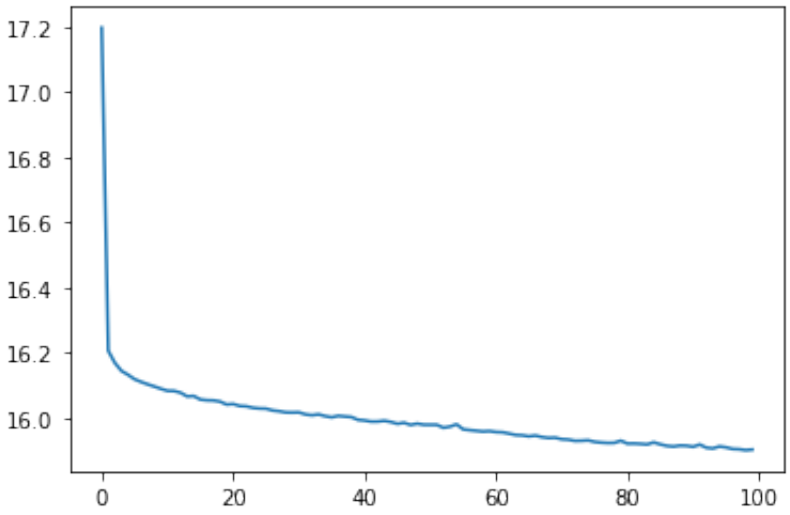
- For our data, we constructed a Neural Network model. We only used data from application train and application test when we first built it to ensure that it worked. Before entering our data into our model, we used our sklearn pipeline to preprocess it. Our architecture was also quite simple.
-

In [20]:
```
from time import time

losses = []
test_losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs

start = time()
for epoch in range(epochs):
  running_loss = 0.0
  running_auc = 0.0
  num_train_auc = 0
  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

    running_loss += loss.detach()
    try:
      running_auc += roc_auc_score(labels, pred.detach().numpy())
      num_train_auc += 1
    except: pass

  losses.append(running_loss/batch_size)
  roc_scores.append(running_auc/num_train_auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
train_time = time() - start
```

- The number of traits we had was estimated to be around 78. Before moving to our hidden layer, which we arbitrarily determined to be 20 nodes large, we employed a ReLU activation function. For our output, we used softmax.

In [21]:
```
import matplotlib.pyplot as plt
plt.plot(range(epochs), losses, label="train loss")
```

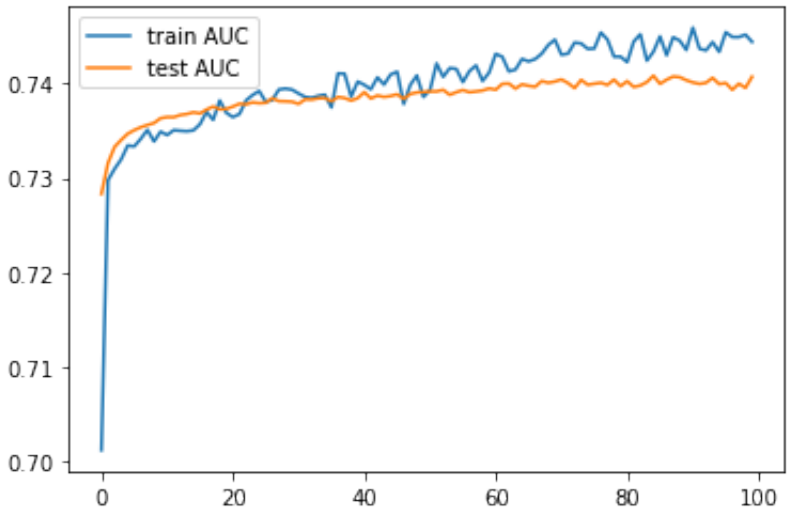
Out[21]:
```
[<matplotlib.lines.Line2D at 0x7facc8996050>]
```



- We trained this model for 100 epochs and a batch size of 64. Here is the loss over time:

In [22]:
```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

Out[22]:
```
<matplotlib.legend.Legend at 0x7facc9261dd0>
```



- And here is the AUC scores over time:

In [ ]:
```
from sklearn.metrics import roc_auc_score

start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start
```

In [24]:
```
ults.loc[0] = ["Deep Learning", roc, "--", "--",
               train_time, test_time, "Deep Learning w/ Application Da

ults
```

Out[24]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Deep Learning	0.740733	--	--	889.931723	0.034056	Deep Learning w/ Application Data

Adding All Features

In [26]:
```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_PO
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DE
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + tra
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DE
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + tr
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_PO

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_C
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONT
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CR
```

In [27]:
```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG"
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

In [28]:
```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y,
```

In [29]:
```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])


scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

In [30]:
```
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

In [31]:
```
batch_size = 64
num_epochs = 100
num_in = train_x.shape[1]
num_layer_1 = 20
num_output = 2
```

In [32]:
```
train_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_ten
data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_
```

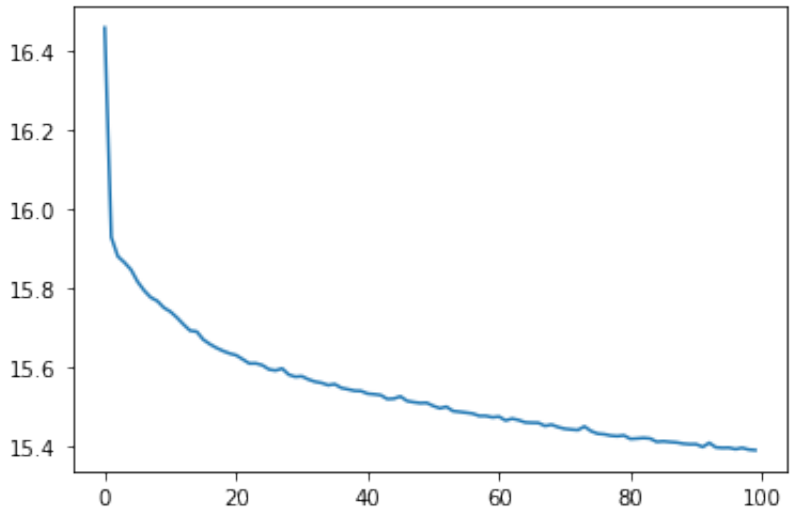
In [33]:
```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, num_layer_1),
        nn.ReLU(),
        nn.Linear(num_layer_1, num_output)
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.SGD(model.parameters(), lr=0.01)
loss_fn = nn.BCELoss()
```

In [34]:

```
from time import time

losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs

start = time()
for epoch in range(epochs):
  running_loss = 0.0
  running_auc = 0.0
  num_train_auc = 0
  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

    running_loss += loss.detach()
    try:
      running_auc += roc_auc_score(labels, pred.detach().numpy())
      num_train_auc += 1
    except: pass

  losses.append(running_loss/batch_size)
  roc_scores.append(running_auc/num_train_auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
train_time = time() - start
```

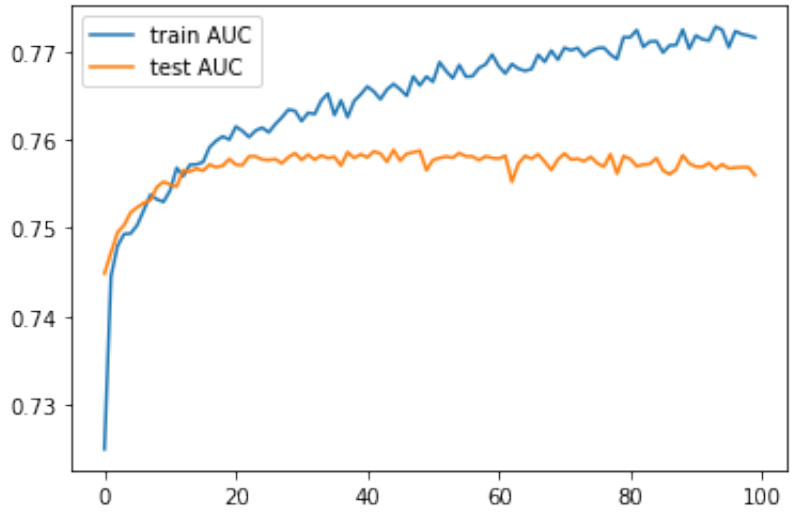
In [35]:
```
import matplotlib.pyplot as plt

plt.plot(range(epochs), losses)
```

Out[35]:
```
[<matplotlib.lines.Line2D at 0x7facc9992a50>]
```



In [36]:
```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

Out[36]:
```
<matplotlib.legend.Legend at 0x7facc9f8eed0>
```



In [37]:
```
from sklearn.metrics import roc_auc_score

start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start
```

In [38]:
```
results.loc[1] = ["Deep Learning", roc, "--", "--",
                  train_time, test_time, "Deep Learning w/ all other

results
```

Out[38]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Deep Learning	0.740733	--	--	889.931723	0.034056	Deep Learning w/ Application Data
1	Deep Learning	0.755966	--	--	893.774806	0.038644	Deep Learning w/ all other data

In [ ]:
```

```

Adam Optimizer

In [39]:
```
ain = datasets['application_train']
ain = train.merge(PA_df, how='left', on='SK_ID_CURR')
ain = train.merge(PCB_df, how='left', on='SK_ID_CURR')
ain = train.merge(IP_df, how='left', on='SK_ID_CURR')
ain = train.merge(B_df, how='left', on='SK_ID_CURR')
ain = train.merge(CCB_df, how='left', on='SK_ID_CURR')

ain["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_POPUL
ain["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GO
ain["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_3
ain["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train[
ain["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_3
ain["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train
ain["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPUL

ain["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHAN
ain["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_
ain["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHO
ain["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDI
```

In [40]:
```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG"
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

In [41]:
```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y,
```

In [42]:
```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])


scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

In [43]:
```
# to tensors
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

In [44]:
```
# create data loaders
train_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_ten
data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_
```

In [45]:
```
# globals
# note: realistically we can only get 20 epochs before overfitting
batch_size = 64
num_epochs = 20
num_layer_1 = 20
num_in = train_x.shape[1]
num_output = 2
```

In [46]:
```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, num_layer_1),
        nn.ReLU(),
        nn.Linear(num_layer_1, num_output),
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.Adam(model.parameters(), lr=0.0001, betas=(0.9, 0.999))
loss_fn = nn.BCELoss()
```

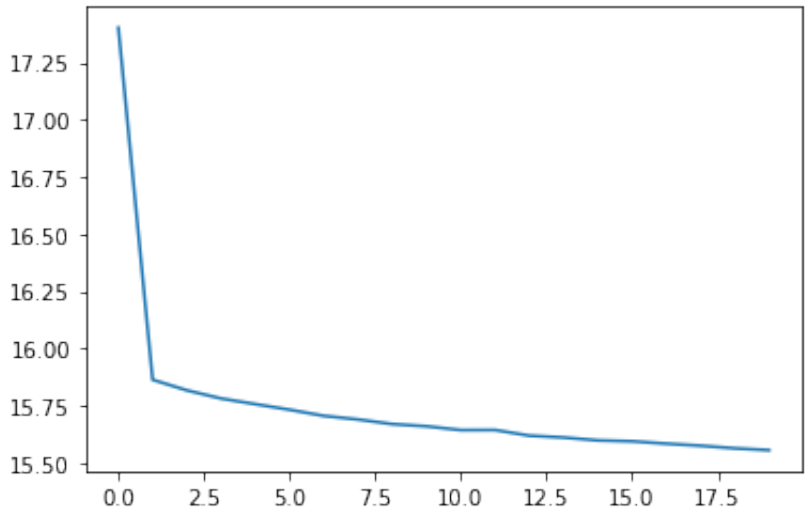
In [47]:
```
from time import time

losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs

start = time()
for epoch in range(epochs):
  running_loss = 0.0
  running_auc = 0.0
  num_train_auc = 0
  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

    running_loss += loss.detach()
    try:
      running_auc += roc_auc_score(labels, pred.detach().numpy())
      num_train_auc += 1
    except: pass

  losses.append(running_loss/batch_size)
  roc_scores.append(running_auc/num_train_auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
train_time = time() - start
```

- As our optimizer, we chose Stochastic Gradient Descent and Binary Cross-Entropy as our loss function, which is defined as:
-

$$l_n = -w_n \left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n) \right]$$

In [48]:
```
import matplotlib.pyplot as plt

plt.plot(range(epochs), losses)
```

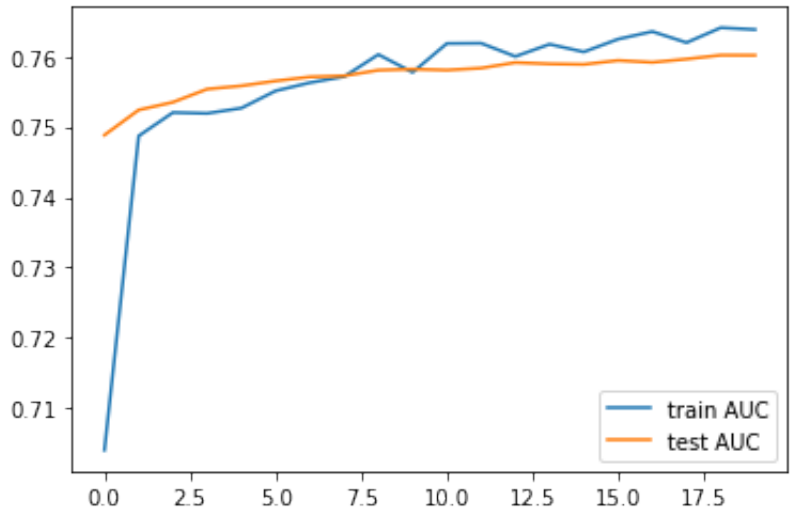
Out[48]:
```
[<matplotlib.lines.Line2D at 0x7facc8286250>]
```



We achieved a final ROC_AUC of 0.739, which is actually better than our baseline model for the same data, which was 0.734 for our test dataset. We then added the rest of the features and switched the optimizer to Adam, since it's usually the better choice. For all other models, we have 173 input features. We chose to run it for 20 epochs after some experimenting.

In [49]:
```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

Out[49]:
```
<matplotlib.legend.Legend at 0x7facc83fcbd0>
```



Type *Markdown* and LaTeX: α^2

In [50]:
```
from sklearn.metrics import roc_auc_score

start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start

acc = np.sum(np.round(preds) == test_y) / len(test_y)
```

In [51]:
```
results.loc[2] = ["Deep Learning", roc, "--", acc,
                  train_time, test_time, "Adam optimizer"]

results
```

Out[51]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Deep Learning	0.740733	--	--	889.931723	0.034056	Deep Learning w/ Application Data
1	Deep Learning	0.755966	--	--	893.774806	0.038644	Deep Learning w/ all other data
2	Deep Learning	0.760161	--	0.917489	197.976691	0.040780	Adam optimizer

Kaggle Submission

In [52]:
```
test = datasets['application_test']
test = test.merge(PA_df, how='left', on='SK_ID_CURR')
test = test.merge(PCB_df, how='left', on='SK_ID_CURR')
test = test.merge(IP_df, how='left', on='SK_ID_CURR')
test = test.merge(B_df, how='left', on='SK_ID_CURR')
test = test.merge(CCB_df, how='left', on='SK_ID_CURR')

test["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = test['REGION_POPU
test["AMT_CREDIT/AMT_GOODS_PRICE"] = test['AMT_CREDIT'] / test['AMT_GO
test["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = test['DEF_
test["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = test['DAYS_BIRTH'] + test[
test["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = test['DEF_
test["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = test['AMT_GOODS_PRICE'] + test
test["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = test['REGION_POPU

test["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = test["DAYS_LAST_PHONE_CHA
test["DAYS_BIRTH+MONTHS_BALANCE"] = test["DAYS_BIRTH"] + test["MONTHS_
test["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = test["DAYS_LAST_PH
test["DAYS_BIRTH*DAYS_CREDIT"] = test["DAYS_BIRTH"] * test["DAYS_CREDI
```

In [53]:
```
# convert test to tensor
test_numpy = scaler.transform(test)
test_tensor = torch.from_numpy(test_numpy).float()

preds = model(test_tensor)[:, 0].detach().numpy()
submit_df = test[['SK_ID_CURR']]
submit_df['TARGET'] = preds

submit_df.to_csv("submission.csv",index=False)

submit_df.head()
```

Out[53]:

	SK_ID_CURR	TARGET
0	100001	0.037238
1	100005	0.169940
2	100013	0.023183
3	100028	0.041304
4	100038	0.172269

In [54]:
```
! kaggle competitions submit -c home-credit-default-risk -f submissior
```

```
Warning: Your Kaggle API key is readable by other users on this syste
m! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
100% 878k/878k [00:00<00:00, 1.45MB/s]
Successfully submitted to Home Credit Default Risk
```

In [55]:
```
# More Layers
```

In [56]:
```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_PO
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DE
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + tra
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DE
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + tr
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_PO

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_C
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONT
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CR
```

In [57]:
```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG"
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

In [58]:
```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y,
```

In [59]:
```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])


scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

In [60]:
```
# to tensors
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

In [61]:
```
# globals
# note: realistically we can only get 20 epochs before overfitting
batch_size = 64
num_epochs = 20
num_in = train_x.shape[1]
num_output = 2
```

In [62]:
```
# create data loaders
train_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_ten
data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_
```

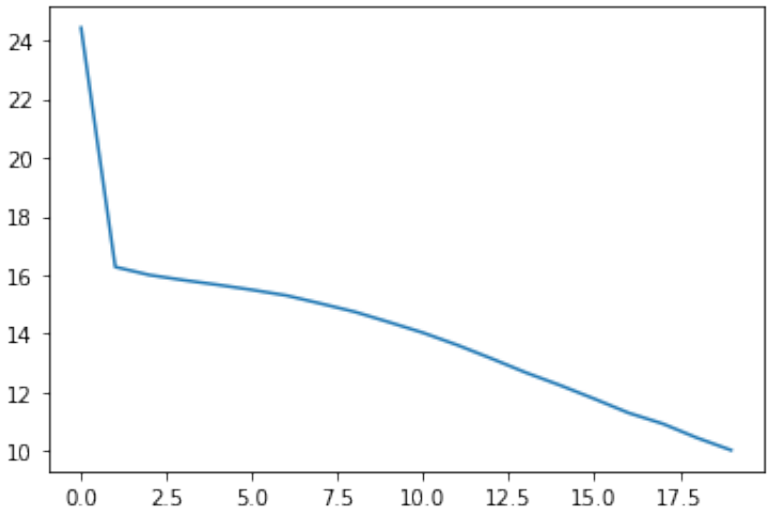
In [63]:
```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, 256),
        nn.ReLU(),
        nn.BatchNorm1d(256),
        nn.Linear(256, 512),
        nn.ReLU(),
        nn.BatchNorm1d(512),
        nn.Linear(512, 256),
        nn.ReLU(),
        nn.BatchNorm1d(256),
        nn.Linear(256, 128),
        nn.ReLU(),
        nn.BatchNorm1d(128),
        nn.Linear(128, 64),
        nn.ReLU(),
        nn.BatchNorm1d(64),
        nn.Linear(64, 32),
        nn.ReLU(),
        nn.BatchNorm1d(32),
        nn.Dropout(0.1),
        nn.Linear(32, num_output)
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.Adam(model.parameters(), lr=0.0001)
loss_fn = nn.BCELoss()
```

In [64]:

```
from time import time

losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs

start = time()
for epoch in range(epochs):
  running_loss = 0.0
  running_auc = 0.0
  num_train_auc = 0
  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

    running_loss += loss.detach()
    try:
      running_auc += roc_auc_score(labels, pred.detach().numpy())
      num_train_auc += 1
    except: pass

  losses.append(running_loss/batch_size)
  roc_scores.append(running_auc/num_train_auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
train_time = time() - start
```

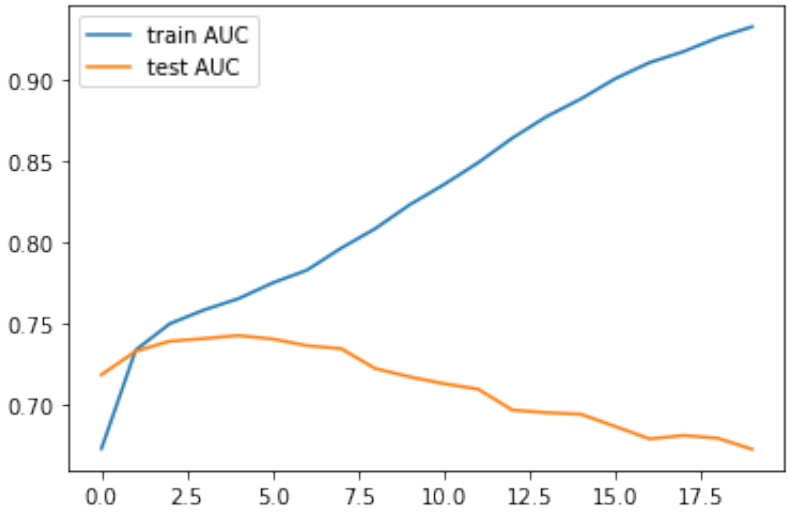
In [65]:
```
import matplotlib.pyplot as plt

plt.plot(range(epochs), losses)
```

Out[65]:
```
[<matplotlib.lines.Line2D at 0x7facc8305510>]
```



In [66]:
```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

Out[66]:
```
<matplotlib.legend.Legend at 0x7facc93dc0d0>
```



- Generally, we found that after 20 epochs, the model overfits. We determined whether a model was overfit or not based on the difference between test and train AUC. This model had a test ROC of 0.758 and after submitting to Kaggle we had an ROC score of 0.750, which was our best of all neural networks we considered in this phase.
- We thought that increasing the size of our network would help. We tried several layers with different parameters, but the AUC always look roughly like above graph.

In [67]:
```
start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start

acc = np.sum(np.round(preds) == test_y) / len(test_y)
```

In [68]:
```
results.loc[3] = ["Deep Learning", roc, "--", acc,
                  train_time, test_time, "More layers"]

results
```

Out[68]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Deep Learning	0.740733	--	--	889.931723	0.034056	Deep Learning w/ Application Data
1	Deep Learning	0.755966	--	--	893.774806	0.038644	Deep Learning w/ all other data
2	Deep Learning	0.760161	--	0.917489	197.976691	0.040780	Adam optimizer
3	Deep Learning	0.673333	--	0.90511	827.497962	0.861070	More layers

K-Fold Training

In [69]:
```
ain = datasets['application_train']
ain = train.merge(PA_df, how='left', on='SK_ID_CURR')
ain = train.merge(PCB_df, how='left', on='SK_ID_CURR')
ain = train.merge(IP_df, how='left', on='SK_ID_CURR')
ain = train.merge(B_df, how='left', on='SK_ID_CURR')
ain = train.merge(CCB_df, how='left', on='SK_ID_CURR')

ain["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_POPUL
ain["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GO
ain["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_3
ain["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train
ain["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_3
ain["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train
ain["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPUL

ain["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHAN
ain["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_
ain["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHO
ain["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDI
```

In [70]:
```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_C
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START"
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACT

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PU
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG"
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOO

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_IN
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

In [71]:
```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y,
```

In [72]:
```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import KFold

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])


scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

In [73]:
```
# to tensors
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

In [74]:
```
# globals
batch_size = 128
num_epochs = 20
num_in = train_x.shape[1]
num_output = 2

kfold = KFold(n_splits=5, shuffle=True)
indexes_gen = kfold.split(train_x_tensor)
```

In [75]:
```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, 256),
        nn.ReLU(),
        nn.Dropout(0.2),
        nn.BatchNorm1d(256),
        nn.Linear(256, 512),
        nn.ReLU(),
        nn.BatchNorm1d(512),
        nn.Linear(512, 256),
        nn.ReLU(),
        nn.BatchNorm1d(256),
        nn.Linear(256, 128),
        nn.ReLU(),
        nn.Dropout(0.2),
        nn.BatchNorm1d(128),
        nn.Linear(128, 64),
        nn.ReLU(),
        nn.BatchNorm1d(64),
        nn.Linear(64, 32),
        nn.ReLU(),
        nn.BatchNorm1d(32),
        nn.Dropout(0.2),
        nn.Linear(32, num_output)
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.Adam(model.parameters(), lr=0.0001)
loss_fn = nn.BCELoss()
```

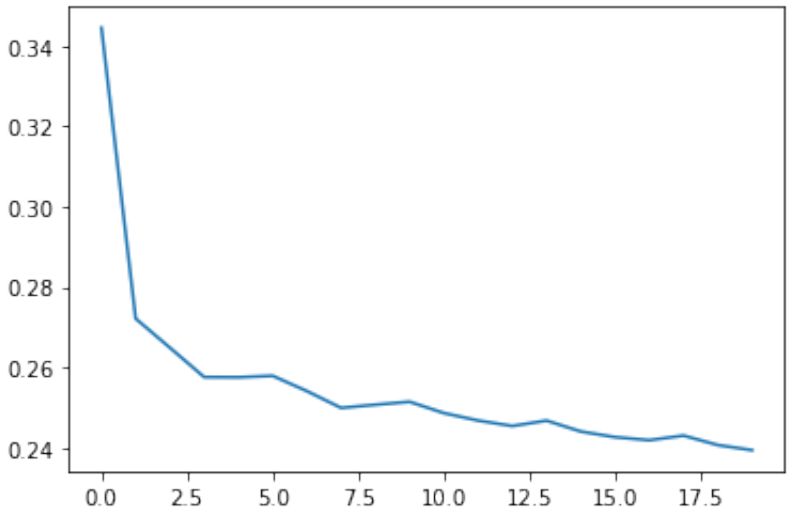
- The architecture for our largest model taht we used.

In [76]:
```
data_loaders = []
test_idxs = []
for train_idx, test_idx in kfold.split(train_x_tensor):
  dataset = torch.utils.data.TensorDataset(train_x_tensor[train_idx],
  data_loaders.append(torch.utils.data.DataLoader(dataset, batch_size=
  test_idxs.append(test_idx)
```

In [77]:
```
from time import time

losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs
curr_fold = 0

start = time()
for epoch in range(epochs):
  # get fold
  if curr_fold+1 >= len(data_loaders):
    curr_fold = 0
  data_loader = data_loaders[curr_fold]
  test_idx = test_idxs[curr_fold]

  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

  # get test-train scores
  preds = model(train_x_tensor[test_idx])[:, 0].detach()
  loss = loss_fn(preds, train_y_tensor[test_idx]).detach()
  try:
    auc = roc_auc_score(train_y_tensor[test_idx], preds.detach().numpy
  except:
    auc = 0

  losses.append(loss)
  roc_scores.append(auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
  curr_fold += 1
train_time = time() - start
```

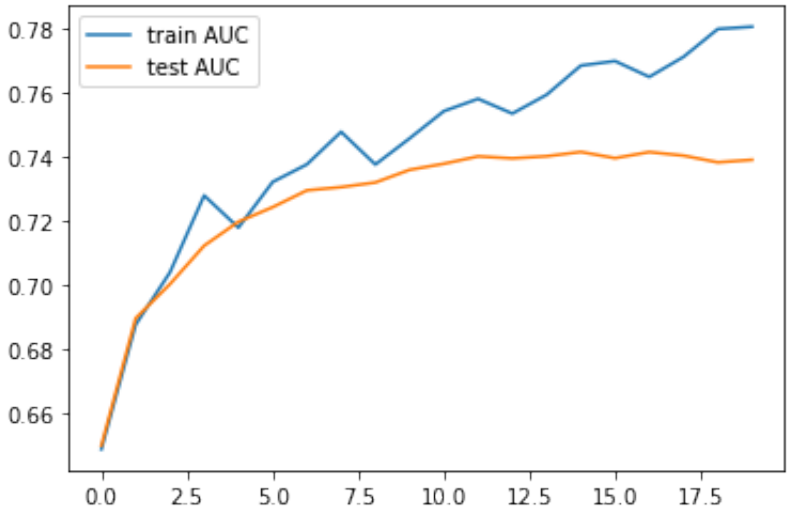
In [78]:
```
import matplotlib.pyplot as plt

plt.plot(range(epochs), losses)
```

Out[78]:
```
[<matplotlib.lines.Line2D at 0x7facc9406a10>]
```



In [79]:
```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

Out[79]:
```
<matplotlib.legend.Legend at 0x7facc9946290>
```



In [80]:
```
start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start

acc = np.sum(np.round(preds) == test_y) / len(test_y)
```

In [81]:
```
results.loc[4] = ["Deep Learning", roc, "--", acc,
                  train_time, test_time, "K-Fold training"]

results
```

Out[81]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Deep Learning	0.740733	--	--	889.931723	0.034056	Deep Learning w/ Application Data
1	Deep Learning	0.755966	--	--	893.774806	0.038644	Deep Learning w/ all other data
2	Deep Learning	0.760161	--	0.917489	197.976691	0.040780	Adam optimizer
3	Deep Learning	0.673333	--	0.90511	827.497962	0.861070	More layers
4	Deep Learning	0.739573	--	0.919028	470.775910	0.874678	K-Fold training

- It clearly overfits far too rapidly. We tried adding Dropout layers and BatchNorm layers, changing the learning rates, and so on, but we couldn't get the test AUC to be better than 0.75. We included K-Fold training as a last resort, in the hopes of reducing overfitting.
- Despite the fact that it appeared to work, our test AUC never exceeded 0.75. There are a lot of Dropout layers in this model. Finally, we chose to return to our original model, which outperformed all of our previous models.
- We experimented with different layer sizes after discovering that a single hidden layer produced the best results. However, a layer size of 64 worked poorly, so we went down to a layer size of 10. but it seemed identical to our base of 20 neurons. In the end, we stuck with our initial model of 1 hidden layer with 20 neurons.

Other Layer Sizes

In [82]:
```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_PO
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DE
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + tra
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DE
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + tr
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_PO

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_C
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONT
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CR
```

In [83]:
```
at_features = [
 "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CI
 "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START",
 "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
 "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

 "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACTI

 "STATUS_MIN", "STATUS_MAX"


um_features = [
 "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
 "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
 "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PUB
 "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG",
 "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
 "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

 "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_P
 "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
 "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
 "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
 "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOOD

 "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INS
 "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_A

 "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
 "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT"

 "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
 "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
 "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
 "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
 "MONTHS_BALANCE_MIN","STATUS_COUNT"
```

In [84]:
```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y,
```

In [85]:
```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

# custom layer to get columns we want from DataFrame
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
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])


scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

In [86]:
```
# to tensors
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

In [87]:
```
# globals
batch_size = 64
num_epochs = 20
num_in = train_x.shape[1]
num_output = 2
```

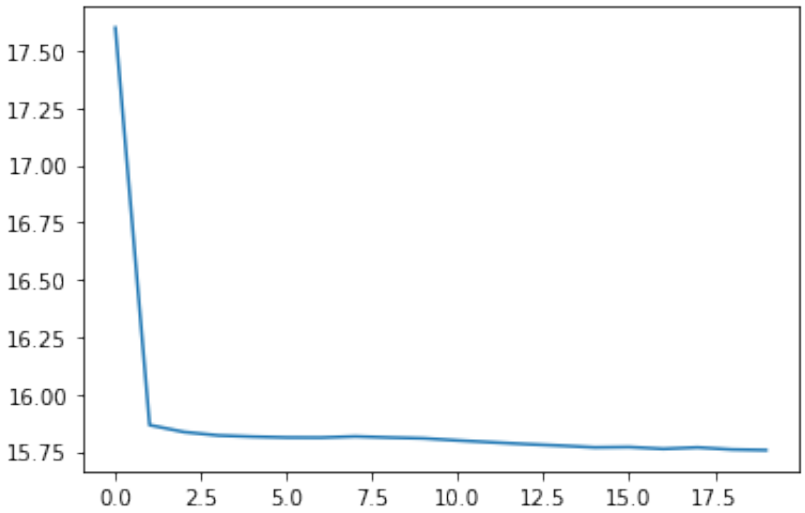
In [88]:
```
# create data loaders
train_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_ten
data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_
```

In [89]:
```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, 20),
        nn.ReLU(),
        nn.Linear(20, 20),
        nn.ReLU(),
        nn.Linear(20, num_output),
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.Adam(model.parameters(), lr=0.0001, betas=(0.9, 0.9))
loss_fn = nn.BCELoss()
```

In [90]:
```
from time import time

losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs

start = time()
for epoch in range(epochs):
  running_loss = 0.0
  running_auc = 0.0
  num_train_auc = 0
  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

    running_loss += loss.detach()
    try:
      running_auc += roc_auc_score(labels, pred.detach().numpy())
      num_train_auc += 1
    except: pass

  losses.append(running_loss/batch_size)
  roc_scores.append(running_auc/num_train_auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
train_time = time() - start
```

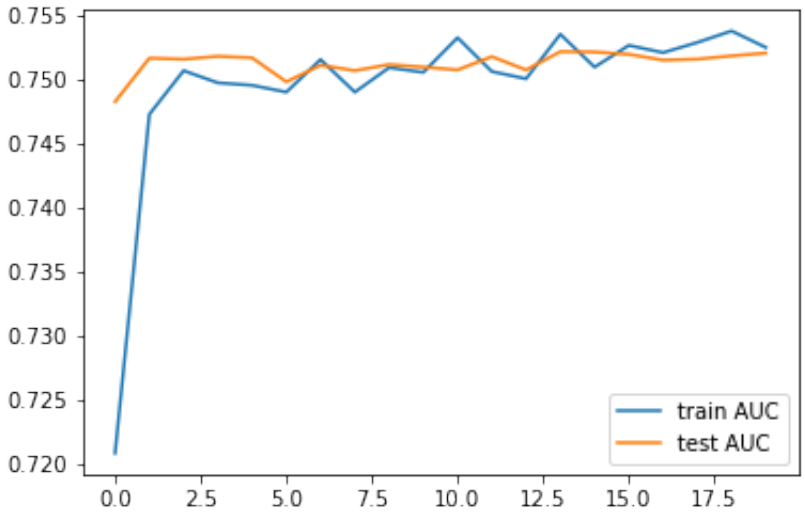
In [91]:
```
import matplotlib.pyplot as plt

plt.plot(range(epochs), losses)
```

Out[91]:
```
[<matplotlib.lines.Line2D at 0x7facc7ba2390>]
```



In [92]:
```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

Out[92]:
```
<matplotlib.legend.Legend at 0x7facc7af9f90>
```



In [93]:
```
start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start

acc = np.sum(np.round(preds) == test_y) / len(test_y)
```

In [94]:
```
results.loc[5] = ["Deep Learning", roc, "--", acc,
                  train_time, test_time, "Modifying Layer Sizes"]

results
```

Out[94]:

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Deep Learning	0.740733	--	--	889.931723	0.034056	Deep Learning w/ Application Data
1	Deep Learning	0.755966	--	--	893.774806	0.038644	Deep Learning w/ all other data
2	Deep Learning	0.760161	--	0.917489	197.976691	0.040780	Adam optimizer
3	Deep Learning	0.673333	--	0.90511	827.497962	0.861070	More layers
4	Deep Learning	0.739573	--	0.919028	470.775910	0.874678	K-Fold training
5	Deep Learning	0.752036	--	0.917294	220.617161	0.040833	Modifying Layer Sizes

Leakage

- The most common sources of leakage are while fitting data. We don't want our pipelines to match the test data when we're fitting them. Then, when it comes to training data, we want to avoid training on the test dataset.
- Changes in class distributions to match would be a more modest leakage. We didn't do this because we separated the datasets with sklearn's train test split.
- We don't have access to the actual labels for the real test dataset in our Kaggle competition, so it's nearly difficult to leak here, but we had to make sure we didn't leak anything from the test dataset we separated from our train dataset. We were very careful with our programming, and and our results were pretty realistic (train ROC > test > Kaggle), we are sure that we didn’t make some kind of mistake with our data.

In [94]:

In [ ]:

Test Data and Kaggle Submission

In [95]:
```
test = datasets['application_test']
test = test.merge(PA_df, how='left', on='SK_ID_CURR')
test = test.merge(PCB_df, how='left', on='SK_ID_CURR')
test = test.merge(IP_df, how='left', on='SK_ID_CURR')
test = test.merge(B_df, how='left', on='SK_ID_CURR')
test = test.merge(CCB_df, how='left', on='SK_ID_CURR')

test["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = test['REGION_POPU
test["AMT_CREDIT/AMT_GOODS_PRICE"] = test['AMT_CREDIT'] / test['AMT_GO
test["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = test['DEF_
test["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = test['DAYS_BIRTH'] + test[
test["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = test['DEF_
test["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = test['AMT_GOODS_PRICE'] + test
test["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = test['REGION_POPU

test["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = test["DAYS_LAST_PHONE_CHA
test["DAYS_BIRTH+MONTHS_BALANCE"] = test["DAYS_BIRTH"] + test["MONTHS_
test["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = test["DAYS_LAST_PH
test["DAYS_BIRTH*DAYS_CREDIT"] = test["DAYS_BIRTH"] * test["DAYS_CREDI
```

In [96]:
```
# convert test to tensor
test_numpy = scaler.transform(test)
test_tensor = torch.from_numpy(test_numpy).float()

preds = model(test_tensor)[:, 0].detach().numpy()
submit_df = test[['SK_ID_CURR']]
submit_df['TARGET'] = preds

submit_df.to_csv("submission.csv",index=False)

submit_df.head()
```

Out[96]:

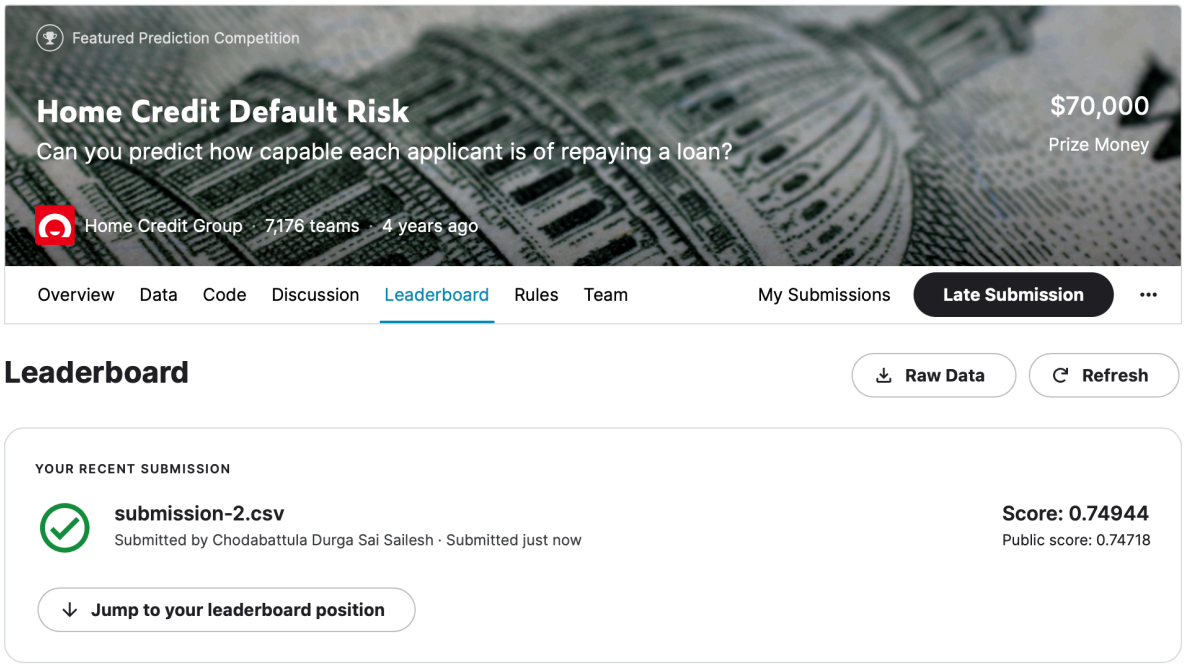
	SK_ID_CURR	TARGET
0	100001	0.054992
1	100005	0.241191
2	100013	0.026251
3	100028	0.029597
4	100038	0.178483

In [97]:
```
!pip install -q kaggle
#!ls
!mkdir ~\.kaggle
!copy kaggle.json ~\.kaggle\kaggle.json
```

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

In [98]:
```
! kaggle competitions submit -c home-credit-default-risk -f submissior
```

```
Warning: Your Kaggle API key is readable by other users on this syste
m! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
100% 877k/877k [00:00<00:00, 1.53MB/s]
Successfully submitted to Home Credit Default Risk
```



Conclusion

- Intially, We decided to use both Bayes and Linear regression as our baseline for the project, and then later decide which model is best but after the results, scores, we decided to go with the Linear regression as it has a concrete accuracy and score to proceed with.
- In phase 2 we have introduced new features into the data set and also combined all the data sets which resulted in improved ROC AUC score from 0.734212 to 0.745048.
- After that we tried to make chage to the imputer which lowered the training and testing time and also there was very slight imporvement in the ROC AUC score of 0.747196
- Then, we used LGBM (untuned) which resulted in improved ROC AUc score of 0.755799 and a slight increase in accuracy from 91.7 to 91.8.
- Then we tuned the LGBM without losing much of the AUC ROC score and the accuracy remained the same.
- For Phase 3, We were disappointed with the results of Part 3. After fine-tuning the architecture of our model, we expected to improve our LGBM scores, but instead found that each adjustment had no effect or worsened the model. One of the most significant changes we made was when we switched from SGD to Adam as our optimizer. Our results remained the same, but we saw a 5x increase in speed.
- We then added layers after that update. This refers to the experiment "More Layers." As you can see, we did a lot worse than they did. We did, however, try a number of other things. As illustrated in the train/test AUC graphs, as the network size was increased, it appeared to overfit significantly more quickly. We believe that a wider network was not the solution. We tried using Dropout or BatchNorm layers to reduce overfitting, but they didn't boost performance over 0.74. Given that our initial model had a score of 0.75, it was evident that our initial model was simply superior.

- ref: https://www.kaggle.com/competitions/home-credit-default-risk (https://www.kaggle.com/competitions/home-credit-default-risk)
- Multiple notebooks and stackoverflow were referred to complete the project

Future Tasks

- Sometimes it's better to be simple, but I think the experiments at this stage show that. No matter what I tried, a single hidden layer of about 20 neurons gave the best results. There may have been more room for experimentation here, but I think We've run out of almost every opportunity to improve performance. For what I left in the table, I probably had more opportunities to use feature engineering to improve the performance of my model than to optimize it.

In [ ]: