Download Data

```
# For Google Colab. If not on Colab, make sure kaggle.json is it the right locatio
from google.colab import files

# upload kaggle.json
uploaded = 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
```

```
# move kaggle.json to the right location
!pip install -q kaggle
!ls
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/kaggle.json
```

```
kaggle.json  sample_data
```

```
# upgrade lightgbm
!pip install --upgrade lightgbm
!pip install --upgrade xgboost
```

```
Requirement already satisfied: lightgbm in /usr/local/lib/python3.7/dist-pack
Collecting lightgbm
  Downloading lightgbm-3.3.1-py3-none-manylinux1_x86_64.whl (2.0 MB)
     |████████████████████████████████| 2.0 MB 4.2 MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: wheel in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scikit-learn!=0.22.0 in /usr/local/lib/python3
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3
Installing collected packages: lightgbm
  Attempting uninstall: lightgbm
    Found existing installation: lightgbm 2.2.3
    Uninstalling lightgbm-2.2.3:
      Successfully uninstalled lightgbm-2.2.3
Successfully installed lightgbm-3.3.1
Requirement already satisfied: xgboost in /usr/local/lib/python3.7/dist-packa
Collecting xgboost
  Downloading xgboost-1.5.1-py3-none-manylinux2014_x86_64.whl (173.5 MB)
     |████████████████████████████████| 173.5 MB 9.6 kB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-package
Installing collected packages: xgboost
  Attempting uninstall: xgboost
    Found existing installation: xgboost 0.90
    Uninstalling xgboost-0.90:
      Successfully uninstalled xgboost-0.90
Successfully installed xgboost-1.5.1
```

```
# download our dataset using the Kaggle api
!kaggle competitions download home-credit-default-risk -p "home-credit-default-ris
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fi
Warning: Looks like you're using an outdated API Version, please consider upd
Downloading installments_payments.csv.zip to home-credit-default-risk
 94% 256M/271M [00:03<00:00, 111MB/s]
100% 271M/271M [00:03<00:00, 74.1MB/s]
Downloading POS_CASH_balance.csv.zip to home-credit-default-risk
 85% 92.0M/109M [00:00<00:00, 95.1MB/s]
100% 109M/109M [00:00<00:00, 122MB/s]
Downloading sample_submission.csv to home-credit-default-risk
  0% 0.00/524k [00:00<?, ?B/s]
100% 524k/524k [00:00<00:00, 167MB/s]
Downloading application_test.csv.zip to home-credit-default-risk
  0% 0.00/5.81M [00:00<?, ?B/s]
100% 5.81M/5.81M [00:00<00:00, 53.7MB/s]
Downloading previous_application.csv.zip to home-credit-default-risk
 81% 62.0M/76.3M [00:00<00:00, 98.8MB/s]
100% 76.3M/76.3M [00:00<00:00, 123MB/s]
Downloading HomeCredit_columns_description.csv to home-credit-default-risk
  0% 0.00/36.5k [00:00<?, ?B/s]
100% 36.5k/36.5k [00:00<00:00, 69.0MB/s]
Downloading credit_card_balance.csv.zip to home-credit-default-risk
 82% 79.0M/96.7M [00:00<00:00, 144MB/s]
100% 96.7M/96.7M [00:00<00:00, 152MB/s]
Downloading bureau.csv.zip to home-credit-default-risk
 84% 31.0M/36.8M [00:00<00:00, 85.9MB/s]
100% 36.8M/36.8M [00:00<00:00, 103MB/s]
Downloading application_train.csv.zip to home-credit-default-risk
 91% 33.0M/36.1M [00:00<00:00, 57.0MB/s]
100% 36.1M/36.1M [00:00<00:00, 73.5MB/s]
Downloading bureau_balance.csv.zip to home-credit-default-risk
 90% 51.0M/56.8M [00:00<00:00, 95.6MB/s]
100% 56.8M/56.8M [00:00<00:00, 105MB/s]
```

```
import os
import zipfile
import numpy as np
import pandas as pd

zip_ref = zipfile.ZipFile('home-credit-default-risk/application_train.csv.zip', 'r
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/application_test.csv.zip', 'r'
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/bureau_balance.csv.zip', 'r')
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/bureau.csv.zip', 'r')
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/credit_card_balance.csv.zip',
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/installments_payments.csv.zip'
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/POS_CASH_balance.csv.zip', 'r'
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/previous_application.csv.zip',
zip_ref.extractall('datasets')
zip_ref.close()
```

Load datasets from files

```
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 track of the
DATA_DIR = "datasets" # folder where unzipped files are

ds_names = ("application_train", "application_test", "bureau","bureau_balance","cr
            "previous_application","POS_CASH_balance")

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

```
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_OW
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

```
application_test: shape is (48744, 121)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
None
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALT
0	100001	Cash loans	F	N	
1	100005	Cash loans	M	N	
2	100013	Cash loans	M	Y	

3 rows × 121 columns

```
bureau: shape is (1716428, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
```

```
 #   Column                 Dtype
---  ------                 -----
 0   SK_ID_CURR             int64
 1   SK_ID_BUREAU           int64
 2   CREDIT_ACTIVE          object
 3   CREDIT_CURRENCY        object
 4   DAYS_CREDIT            int64
 5   CREDIT_DAY_OVERDUE     int64
 6   DAYS_CREDIT_ENDDATE    float64
 7   DAYS_ENDDATE_FACT      float64
 8   AMT_CREDIT_MAX_OVERDUE float64
 9   CNT_CREDIT_PROLONG     int64
 10  AMT_CREDIT_SUM         float64
 11  AMT_CREDIT_SUM_DEBT    float64
 12  AMT_CREDIT_SUM_LIMIT   float64
 13  AMT_CREDIT_SUM_OVERDUE float64
 14  CREDIT_TYPE            object
 15  DAYS_CREDIT_UPDATE     int64
```

```
# datasets that we have
datasets.keys()
```

```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance
```

New Application Features

We use a brute-force approach to consider all ways of combining our application data to create new data. This uses a lot of RAM and can't be run twice without clearing all RAM. For that reason the code is commented out and only specific examples are picked out.

```
all_num_features = [
  "CNT_CHILDREN", "AMT_INCOME_TOTAL", "AMT_CREDIT", "AMT_ANNUITY", "REGION_POPULAT
  "DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_REGISTRATION", "CNT_FAM_MEMBERS",
  "EXT_SOURCE_2", "EXT_SOURCE_3","AMT_GOODS_PRICE", "OBS_30_CNT_SOCIAL_CIRCLE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "OBS_60_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCL
  "DAYS_LAST_PHONE_CHANGE", "AMT_REQ_CREDIT_BUREAU_HOUR", "AMT_REQ_CREDIT_BUREAU_D
  "AMT_REQ_CREDIT_BUREAU_WEEK", "AMT_REQ_CREDIT_BUREAU_MON", "AMT_REQ_CREDIT_BUREA
  "AMT_REQ_CREDIT_BUREAU_YEAR", "DAYS_ID_PUBLISH",
]
```

Creating new features

```
# test_df = pd.DataFrame()
# corrs = {}

# # this is an example for addition. Due to memory constraints, we can only run
# # 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"][key1] + datasets["a
#       corrs[key1+"+"+key2] = test_df[key1+"+"+key2].corr(datasets["application_t

# test_df
```

	CNT_CHILDREN+AMT_INCOME_TOTAL	CNT_CHILDREN+AMT_CREDIT	CNT_CHILDREN+A
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

```
# corrs
```

```
{'AMT_ANNUITY+AMT_CREDIT': -0.029992788388881832,
 'AMT_ANNUITY+AMT_GOODS_PRICE': -0.03895368748304597,
 'AMT_ANNUITY+AMT_INCOME_TOTAL': -0.004700974848291929,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_DAY': -0.012692007797941576,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_HOUR': -0.012692023119781126,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_MON': -0.012692790257676228,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_QRT': -0.012692133281444523,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_WEEK': -0.012692014912021786,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_YEAR': -0.012689457376392555,
 'AMT_ANNUITY+CNT_CHILDREN': -0.012815592203985579,
```

```
'AMT_ANNUITY+CNT_FAM_MEMBERS': -0.012815340626177356,
'AMT_ANNUITY+DAYS_BIRTH': 0.010259714626087352,
'AMT_ANNUITY+DAYS_EMPLOYED': -0.0465059268859311,
'AMT_ANNUITY+DAYS_ID_PUBLISH': -0.007408521914076275,
'AMT_ANNUITY+DAYS_LAST_PHONE_CHANGE': -0.009686006998367894,
'AMT_ANNUITY+DAYS_REGISTRATION': -0.002517930400554016 4,
'AMT_ANNUITY+DEF_30_CNT_SOCIAL_CIRCLE': -0.012743848657843827,
'AMT_ANNUITY+DEF_60_CNT_SOCIAL_CIRCLE': -0.012744059596424216,
'AMT_ANNUITY+EXT_SOURCE_2': -0.012712060407309574,
'AMT_ANNUITY+EXT_SOURCE_3': -0.012706869694946091,
'AMT_ANNUITY+OBS_30_CNT_SOCIAL_CIRCLE': -0.012743346444190775,
'AMT_ANNUITY+OBS_60_CNT_SOCIAL_CIRCLE': -0.012743376871114948,
'AMT_ANNUITY+REGION_POPULATION_RELATIVE': -0.012816595582606848,
'AMT_CREDIT+AMT_ANNUITY': -0.029992788388881832,
'AMT_CREDIT+AMT_GOODS_PRICE': -0.03493261272149829,
'AMT_CREDIT+AMT_INCOME_TOTAL': -0.02643204209374059,
'AMT_CREDIT+AMT_REQ_CREDIT_BUREAU_DAY': -0.0288747327789067,
'AMT_CREDIT+AMT_REQ_CREDIT_BUREAU_HOUR': -0.02887473338296208,
'AMT_CREDIT+AMT_REQ_CREDIT_BUREAU_MON': -0.02887475819453305 7,
'AMT_CREDIT+AMT_REQ_CREDIT_BUREAU_QRT': -0.028874736616401865,
'AMT_CREDIT+AMT_REQ_CREDIT_BUREAU_WEEK': -0.028874733172638032,
'AMT_CREDIT+AMT_REQ_CREDIT_BUREAU_YEAR': -0.0288746479811228,
'AMT_CREDIT+CNT_CHILDREN': -0.030369251920242327,
'AMT_CREDIT+CNT_FAM_MEMBERS': -0.030369123231339105,
'AMT_CREDIT+DAYS_BIRTH': -0.029537001927404407,
'AMT_CREDIT+DAYS_EMPLOYED': -0.04447529458409413,
'AMT_CREDIT+DAYS_ID_PUBLISH': -0.030176839869587593,
'AMT_CREDIT+DAYS_LAST_PHONE_CHANGE': -0.03026117587557106,
'AMT_CREDIT+DAYS_REGISTRATION': -0.02999821743969418,
'AMT_CREDIT+DEF_30_CNT_SOCIAL_CIRCLE': -0.030435606304934394,
'AMT_CREDIT+DEF_60_CNT_SOCIAL_CIRCLE': -0.030435613874735944,
'AMT_CREDIT+EXT_SOURCE_2': -0.03033211758222435,
'AMT_CREDIT+EXT_SOURCE_3': -0.02892169585835199,
'AMT_CREDIT+OBS_30_CNT_SOCIAL_CIRCLE': -0.03043558687776561,
'AMT_CREDIT+OBS_60_CNT_SOCIAL_CIRCLE': -0.03043558799135641,
'AMT_CREDIT+REGION_POPULATION_RELATIVE': -0.030369287636623277,
'AMT_GOODS_PRICE+AMT_ANNUITY': -0.03895368748304597,
'AMT_GOODS_PRICE+AMT_CREDIT': -0.03493261272149829,
'AMT_GOODS_PRICE+AMT_INCOME_TOTAL': -0.033175981854863694,
'AMT_GOODS_PRICE+AMT_REQ_CREDIT_BUREAU_DAY': -0.0381313523782449,
'AMT_GOODS_PRICE+AMT_REQ_CREDIT_BUREAU_HOUR': -0.0381313530531617,
'AMT_GOODS_PRICE+AMT_REQ_CREDIT_BUREAU_MON': -0.03813137871458215,
'AMT_GOODS_PRICE+AMT_REQ_CREDIT_BUREAU_QRT': -0.038131356264196586,
'AMT_GOODS_PRICE+AMT_REQ_CREDIT_BUREAU_WEEK': -0.03813135281653225,
'AMT_GOODS_PRICE+AMT_REQ_CREDIT_BUREAU_YEAR': -0.03813126270067336,
'AMT_GOODS_PRICE+CNT_CHILDREN': -0.0396452437417633 5,
'AMT_GOODS_PRICE+CNT_FAM_MEMBERS': -0.03964525224084931,
'AMT_GOODS_PRICE+DAYS_BIRTH': -0.0387412488869335 9,
'AMT_GOODS_PRICE+DAYS_EMPLOYED': -0.054284369724802196
```

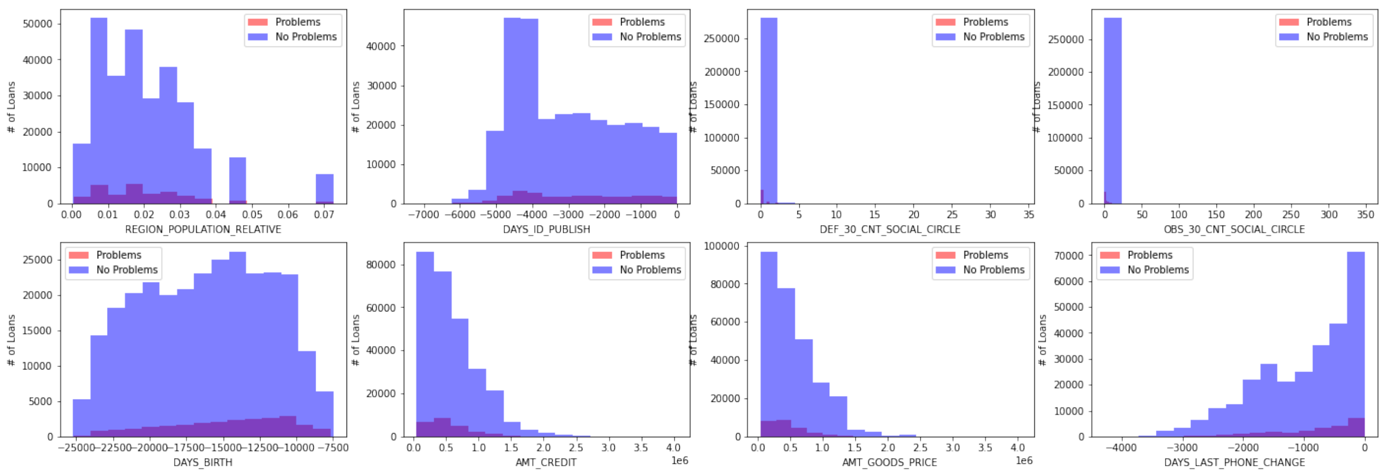
Analysis of new features

Of all the features we created, there were some interesting results just looking at the correlation values. But now we should visually inspect our new data for anomilies or red flags.

```
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).rename('perc
  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", label="Problems
  ax.hist(df[df["TARGET"]==0][y], bins=15, alpha=0.5, color="blue", label="No Prob

  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(ascending = Fa
  sum_missing = df.isna().sum().sort_values(ascending = False)
  missing_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Miss
  return missing_data.head(10)
```

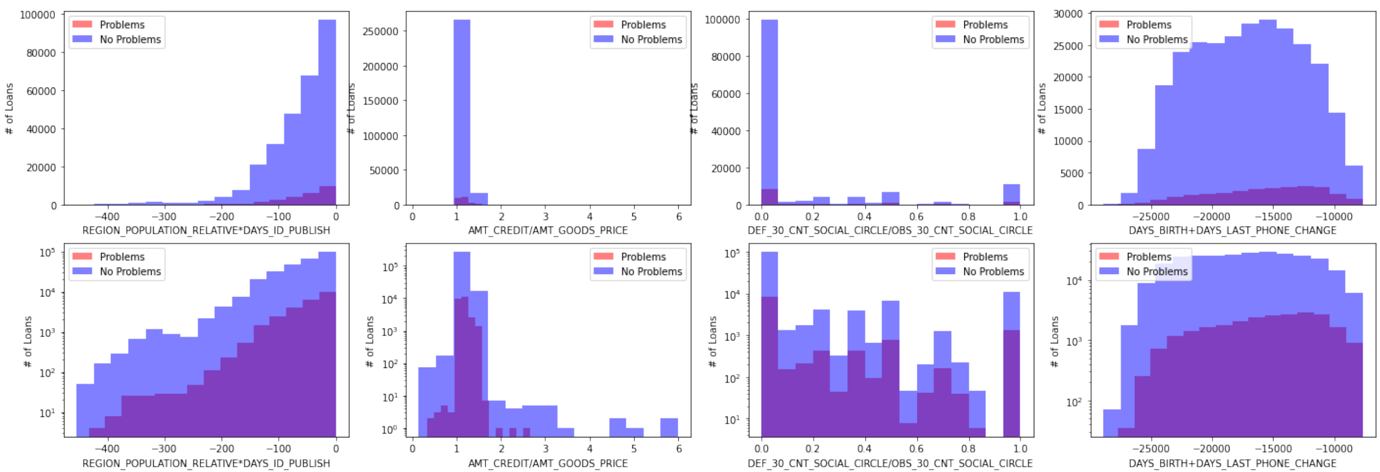
```
temp_app = pd.DataFrame()
app_df = datasets['application_train']
temp_app["TARGET"] = app_df['TARGET']

temp_app["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = app_df['REGION_POPULATION_
temp_app["AMT_CREDIT/AMT_GOODS_PRICE"] = app_df['AMT_CREDIT'] / app_df['AMT_GOODS_
temp_app["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = app_df['DEF_30_CNT_
temp_app["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = app_df['DAYS_BIRTH'] + app_df['DAY
```

```
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])
```



Above are the original features used to create the new features, which are graphed below. It might be useful to come back and reference these graphs

```
fig, axs = plt.subplots(2, 4, figsize=(24, 8))
num_hist(temp_app, "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", axs[0,0])
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", axs[0,2])
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[1,0], True)
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", axs[1,2],
num_hist(temp_app, "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE", axs[1,3], True)
```



These relationships are quite interesting.

By referencing the graphs above this set, `REGION_POPULATION_RELATIVE` and `DAYS_ID_PUBLISH` have graphs with one high point around the middle. However, `REGION_POPULATION_RELATION*DAYS_ID_PUBLISH` has a clear trend that, the further to the right, the more problems the client has with repayment.

`DEF_30_CNT_SOCIAL_CIRCLE` and `OBS_30_CNT_SOCIAL_CIRCLE` both had an extremely large portion of the samples in one region and it was really hard to even pick points apart, but `DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE` does a much better job of spreading those points apart.

Baseline Model for just Application Data

Let's create a basic model with just applicaiton data

```
results = pd.DataFrame(columns=["ExpID", "ROC AUC Score", "Cross fold train accura
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "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_PUBLISH"
]
```

```
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())
])
```

```
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_train'].colum
y_train = datasets["application_train"]['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.

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 = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full_pipeline, c
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)
```

```
results.loc[0] = ["Baseline", roc,pct(logit_score_train), np.round(pct(logit_score
                  train_time, test_time, "LogisticRegression"]
results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.398	LogisticRegression

Of course, this score isn't very good. In the Kaggle competition, it would place us 5773 out of 7176 entries. However, it's a good baseline to evaluate our future models on

Baseline Model with our new features

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

app_df["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = app_df['REGION_POPULATION_R
app_df["AMT_CREDIT/AMT_GOODS_PRICE"] = app_df['AMT_CREDIT'] / app_df['AMT_GOODS_PR
app_df["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = app_df['DEF_30_CNT_S
app_df["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = app_df['DAYS_BIRTH'] + app_df['DAYS_
app_df["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = app_df['DEF_30_CNT_S
app_df["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = app_df['AMT_GOODS_PRICE'] + app_df['DAYS_
app_df["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = app_df['REGION_POPULATION_R
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "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_PUBLISH",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_PRICE",
  "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_GOODS_PRICE"
]
```

```
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())
])
```

```
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_train'].colum
y_train = datasets["application_train"]['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.

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 = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full_pipeline, c
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)
```

```
results.loc[1] = ["Baseline", roc,pct(logit_score_train), np.round(pct(logit_score_
                   train_time, test_time, "LogisticRegression + new Application fe
results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
							LogisticRegression

We can see that our new features didn't improve our accuracy, but the ROC AUC score was improved. Since the AUC score is the most important metric to us, this is a success

New Features for Other Datasets

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

Previous Applications

```
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

```
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, how='left', on
```

```
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","CNT_INSTAL
```

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

EDA:

```
temp_app = datasets['application_train'].merge(PCB_df_temp, how='left', on='SK_ID_
```

```
print(temp_app[["PREV_CNT_INSTALMENT", "PREV_CNT_INSTALMENT_FUTURE","PREV_PCB_MONT

missing_vals(temp_app[["PREV_CNT_INSTALMENT", "PREV_CNT_INSTALMENT_FUTURE","PREV_P
```

```
PREV_CNT_INSTALMENT          -0.038646
PREV_CNT_INSTALMENT_FUTURE    0.032835
PREV_PCB_MONTHS_BALANCE       0.050945
TARGET                        1.000000
Name: TARGET, dtype: float64
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-45-00434491af16> in <module>()
      1 print(temp_app[["PREV_CNT_INSTALMENT",
"PREV_CNT_INSTALMENT_FUTURE","PREV_PCB_MONTHS_BALANCE", "TARGET"]].corr()
["TARGET"])
      2
----> 3 missing_vals(temp_app[["PREV_CNT_INSTALMENT",
"PREV_CNT_INSTALMENT_FUTURE","PREV_PCB_MONTHS_BALANCE" ]])

NameError: name 'missing_vals' is not defined
```

SEARCH STACK OVERFLOW

Previous IP

```
IP_df_copy = datasets['installments_payments'].groupby('SK_ID_PREV').agg({
    "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["AMT_PAYMENT"
```

```
IP_to_PA_df =  datasets['previous_application'].merge(IP_df_copy, how='left', on='
```

```
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", "AMT_PAYME
```

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

EDA:

```
temp_app = datasets['application_train'].merge(IP_df_temp, how='left', on='SK_ID_C
```

```
print(temp_app[["PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT", "PREV_DAYS_INSTALMENT",

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

```
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
```

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

Previous CCB

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

```
CCB_to_PA_df =  datasets['previous_application'].merge(CCB_df_copy, how='l
```

```
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", "MONTH
```

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

EDA:

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

```
print(temp_app[["PREV_AMT_BALANCE", "PREV_CCB_MONTHS_BALANCE", "PREV_AMT_C

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

```
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
```

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

POS Cash Balances

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

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

```
new_features = ["CNT_INSTALMENT", "CNT_INSTALMENT_FUTURE", "MONTHS_BALANCE"]
```

```
test_df = pd.DataFrame()
corrs = {}

# this is an example for addition. Due to memory constraints, we can only run
# 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['TARGET'])

test_df
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-1206db314d59> in <module>()
      4 # this is an example for addition. Due to memory constraints, we can
only run
      5 # one operation (+,-,/,*) per run
----> 6 for key1 in all_num_features:
      7   for key2 in new_features:
      8     if key1 != key2:

NameError: name 'all_num_features' is not defined
```

SEARCH STACK OVERFLOW

```
corrs
```

```
{'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': -0.036554599757174 35,
 'AMT_GOODS_PRICE+MONTHS_BALANCE': -0.03655872795487446,
 'AMT_INCOME_TOTAL+CNT_INSTALMENT': -0.001702045476611342 7,
 'AMT_INCOME_TOTAL+CNT_INSTALMENT_FUTURE': -0.0016998058664761252,
 'AMT_INCOME_TOTAL+MONTHS_BALANCE': -0.0016916258330551884,
 'AMT_REQ_CREDIT_BUREAU_DAY+CNT_INSTALMENT': -0.03331297416288113 4,
```

'AMT_REQ_CREDIT_BUREAU_DAY+CNT_INSTALMENT_FUTURE': 0.030706893892791954,
'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': 0.030679692253522555,
'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': 0.028455230401056457,
'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': 0.02983197332882702,
'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': 0.03066428636133782,
'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': 0.03198714792694153,
'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': 0.029556315698364956,
'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': 0.054430592836276145

Instalment Payments

```
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"]
```

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

```
new_features = ["AMT_INSTALMENT", "AMT_PAYMENT", "DAYS_INSTALMENT", "DAYS_ENTRY_PA
```

```
test_df = pd.DataFrame()
corrs = {}

# this is an example for addition. Due to memory constraints, we can only run
# 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['TARGET'])

test_df
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-42-7a33d43a116d> in <module>()
      4 # this is an example for addition. Due to memory constraints, we can
only run
      5 # one operation (+,-,/,*) per run
----> 6 for key1 in all_num_features:
      7   for key2 in new_features:
      8     if key1 != key2:

NameError: name 'all_num_features' is not defined
```

SEARCH STACK OVERFLOW

```
corrs
```

```
{'AMT_ANNUITY-AMT_INSTALMENT': 0.019716211234084097,
 'AMT_ANNUITY-AMT_PAYMENT': 0.02429603376430387,
 'AMT_ANNUITY-DAYS_ENTRY_PAYMENT': -0.012659572423444008,
```

```
'AMT_ANNUITY-DAYS_INSTALMENT': -0.012653013349201491,
'AMT_ANNUITY-SUM_MISSED': -0.028478592241464488,
'AMT_CREDIT-AMT_INSTALMENT': 0.007315135880388606,
'AMT_CREDIT-AMT_PAYMENT': 0.012321352050999787,
'AMT_CREDIT-DAYS_ENTRY_PAYMENT': -0.027842341138886666,
'AMT_CREDIT-DAYS_INSTALMENT': -0.027835645523273583,
'AMT_CREDIT-SUM_MISSED': -0.03587613896728746,
'AMT_GOODS_PRICE-AMT_INSTALMENT': 0.00478790296137049,
'AMT_GOODS_PRICE-AMT_PAYMENT': 0.009923263711473665,
'AMT_GOODS_PRICE-DAYS_ENTRY_PAYMENT': -0.03711095933545598,
'AMT_GOODS_PRICE-DAYS_INSTALMENT': -0.03710582679088353,
'AMT_GOODS_PRICE-SUM_MISSED': -0.04435465676736081,
'AMT_INCOME_TOTAL-AMT_INSTALMENT': 0.019063287531481916,
'AMT_INCOME_TOTAL-AMT_PAYMENT': 0.02363384844208317,
'AMT_INCOME_TOTAL-DAYS_ENTRY_PAYMENT': -0.0020267002947920083,
'AMT_INCOME_TOTAL-DAYS_INSTALMENT': -0.002030989100720451,
'AMT_INCOME_TOTAL-SUM_MISSED': -0.017770966157930865,
'AMT_REQ_CREDIT_BUREAU_DAY-AMT_INSTALMENT': 0.017315610898029198,
'AMT_REQ_CREDIT_BUREAU_DAY-AMT_PAYMENT': 0.021285836708001018,
'AMT_REQ_CREDIT_BUREAU_DAY-DAYS_ENTRY_PAYMENT': -0.05470595421666537,
'AMT_REQ_CREDIT_BUREAU_DAY-DAYS_INSTALMENT': -0.05456828746242355,
'AMT_REQ_CREDIT_BUREAU_DAY-SUM_MISSED': -0.024522303668901926,
'AMT_REQ_CREDIT_BUREAU_HOUR-AMT_INSTALMENT': 0.017315610619511132,
'AMT_REQ_CREDIT_BUREAU_HOUR-AMT_PAYMENT': 0.021285836440672112,
'AMT_REQ_CREDIT_BUREAU_HOUR-DAYS_ENTRY_PAYMENT': -0.054706246509442794,
'AMT_REQ_CREDIT_BUREAU_HOUR-DAYS_INSTALMENT': -0.05456857929271193,
'AMT_REQ_CREDIT_BUREAU_HOUR-SUM_MISSED': -0.024522305111187956,
'AMT_REQ_CREDIT_BUREAU_MON-AMT_INSTALMENT': 0.01731559772226145,
'AMT_REQ_CREDIT_BUREAU_MON-AMT_PAYMENT': 0.021285824234618875,
'AMT_REQ_CREDIT_BUREAU_MON-DAYS_ENTRY_PAYMENT': -0.054717141017268815,
'AMT_REQ_CREDIT_BUREAU_MON-DAYS_INSTALMENT': -0.05457946632706627,
'AMT_REQ_CREDIT_BUREAU_MON-SUM_MISSED': -0.024522375154846933,
'AMT_REQ_CREDIT_BUREAU_QRT-AMT_INSTALMENT': 0.017315607872508583,
'AMT_REQ_CREDIT_BUREAU_QRT-AMT_PAYMENT': 0.021285834018328134,
'AMT_REQ_CREDIT_BUREAU_QRT-DAYS_ENTRY_PAYMENT': -0.05470976525385729,
'AMT_REQ_CREDIT_BUREAU_QRT-DAYS_INSTALMENT': -0.05457209688898062,
'AMT_REQ_CREDIT_BUREAU_QRT-SUM_MISSED': -0.02452231938189437,
'AMT_REQ_CREDIT_BUREAU_WEEK-AMT_INSTALMENT': 0.0173156107230773,
'AMT_REQ_CREDIT_BUREAU_WEEK-AMT_PAYMENT': 0.02128583654486823,
'AMT_REQ_CREDIT_BUREAU_WEEK-DAYS_ENTRY_PAYMENT': -0.0547062245744872,
'AMT_REQ_CREDIT_BUREAU_WEEK-DAYS_INSTALMENT': -0.054568558860913545,
'AMT_REQ_CREDIT_BUREAU_WEEK-SUM_MISSED': -0.024522304730514744,
'AMT_REQ_CREDIT_BUREAU_YEAR-AMT_INSTALMENT': 0.017315660076260713,
'AMT_REQ_CREDIT_BUREAU_YEAR-AMT_PAYMENT': 0.02128588707385238,
'AMT_REQ_CREDIT_BUREAU_YEAR-DAYS_ENTRY_PAYMENT': -0.05465513115981941,
'AMT_REQ_CREDIT_BUREAU_YEAR-DAYS_INSTALMENT': -0.05451757865155801,
'AMT_REQ_CREDIT_BUREAU_YEAR-SUM_MISSED': -0.02452210656144546,
'CNT_CHILDREN-AMT_INSTALMENT': 0.01981094212071926,
'CNT_CHILDREN-AMT_PAYMENT': 0.024375349619034332,
'CNT_CHILDREN-DAYS_ENTRY_PAYMENT': -0.05877938147997689,
'CNT_CHILDREN-DAYS_INSTALMENT': -0.058633590030850155,
'CNT_CHILDREN-SUM_MISSED': -0.027932291372243904,
'CNT_FAM_MEMBERS-AMT_INSTALMENT': 0.019810936758714887,
'CNT_FAM_MEMBERS-AMT_PAYMENT': 0.02437534463853047,
'CNT_FAM_MEMBERS-DAYS_ENTRY_PAYMENT': -0.05878405326893657
```

```
 'CNT_FAM_MEMBERS-DAYS_INSTALMENT': -0.05863824708027986,
```

Bureau

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

```
comb_app = datasets['application_train'].merge(B_df, how='left', on='SK_ID_CURR')
```

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

```
test_df = pd.DataFrame()
corrs = {}

# this is an example for addition. Due to memory constraints, we can only run
# 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['TARGET'])

test_df
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-36-8de6b2044659> in <module>()
      4 # this is an example for addition. Due to memory constraints, we can
only run
      5 # one operation (+,-,/,*) per run
----> 6 for key1 in all_num_features:
      7   for key2 in new_features:
      8     if key1 != key2:

NameError: name 'all_num_features' is not defined
```

SEARCH STACK OVERFLOW

```
corrs
```

Bureau Balance dataset

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

```
temp = pd.DataFrame({"MONTHS_BALANCE_MIN": BB_df["MONTHS_BALANCE"]["min"],  "STATU
BB_df = temp
```

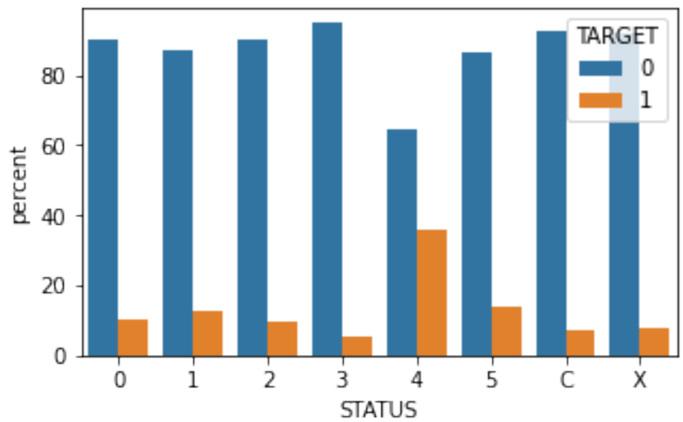
```
BB_to_B_df =  datasets['bureau'].merge(BB_df, how='left', on='SK_ID_BUREAU')
BB_to_B_df = BB_to_B_df.dropna(subset = ["STATUS_MAX","STATUS_MIN"])
B_df_temp = BB_to_B_df.groupby('SK_ID_CURR').agg({
    "MONTHS_BALANCE_MIN": "min",
    "STATUS_MIN": "min",
    "STATUS_MAX" : 'max',
    "STATUS_COUNT":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

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

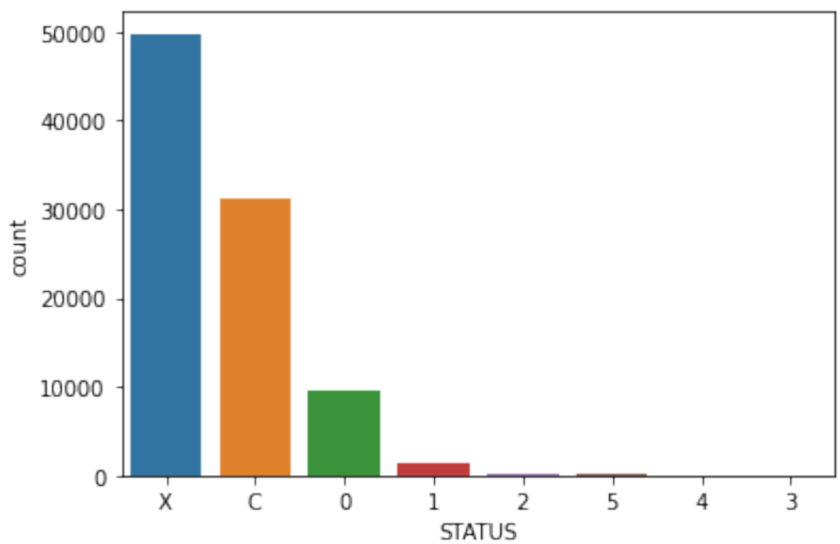
EDA for Bureau Balance

```
temp_app = datasets['application_train'].merge(B_df_temp, how='left', on='SK_ID_CU
```

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

def cat_bar(df, x, ax):
  df2 = df.groupby(x)['TARGET'].value_counts(normalize=True).mul(100).rename('perc
  sns.barplot(x=x,y='percent',hue='TARGET',data=df2,ax=ax)

fig, axs = plt.subplots(1, 1, figsize=(5, 3))
cat_bar(temp_app, 'STATUS', axs)
plt.show()
sns.countplot(temp_app["STATUS"])
```




```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa501a8ee50>
```



It looks like `STATUS` is pretty interesting, especially for values like 4 and 3. Unfortunately, there's not a lot of data for it

Now, let's take a look at our new feature, `BUREAU_MONTHS_BALANCE`

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

print(temp_app[["BUREAU_MONTHS_BALANCE", "TARGET"]].corr()["TARGET"])

missing_vals(temp_app[["BUREAU_MONTHS_BALANCE", "TARGET"]])
```

```
BUREAU_MONTHS_BALANCE    0.076424
TARGET                   1.000000
Name: TARGET, dtype: float64
```

	Percent	Missing Count
BUREAU_MONTHS_BALANCE	70.01	215280
TARGET	0.00	0

`BUREAU_MONTHS_BALANCE` seems to be relatively strongly correlated with our target, but 70% of the data is missing.

Credit Card Balances

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

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

```
new_features = ["AMT_BALANCE", "MONTHS_BALANCE", "AMT_CREDIT_LIMIT_ACTUAL"]
```

```
test_df = pd.DataFrame()
corrs = {}

# this is an example for addition. Due to memory constraints, we can only run
# 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['TARGET'])

test_df
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-41-1206db314d59> in <module>()
      4 # this is an example for addition. Due to memory constraints, we can
only run
      5 # one operation (+,-,/,*) per run
----> 6 for key1 in all_num_features:
      7   for key2 in new_features:
      8     if key1 != key2:

NameError: name 'all_num_features' is not defined
```

SEARCH STACK OVERFLOW

```
corrs
```

```
{'AMT_ANNUITY+AMT_BALANCE': 0.08371049424218176,
 'AMT_ANNUITY+AMT_CREDIT_LIMIT_ACTUAL': -0.025063558915545698,
 'AMT_ANNUITY+MONTHS_BALANCE': -0.024771390825012576,
 'AMT_CREDIT+AMT_BALANCE': -0.009802925077219879,
 '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': 0.04845092882920461,
 'AMT_INCOME_TOTAL+AMT_CREDIT_LIMIT_ACTUAL': -0.017685313734971734,
 'AMT_INCOME_TOTAL+MONTHS_BALANCE': -0.0176466574023929,
 'AMT_REQ_CREDIT_BUREAU_DAY+AMT_BALANCE': 0.08599405922754703,
 'AMT_REQ_CREDIT_BUREAU_DAY+AMT_CREDIT_LIMIT_ACTUAL': -0.05884128345825502 5
 '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': -0.0588422466726554 7
 'AMT_REQ_CREDIT_BUREAU_HOUR+MONTHS_BALANCE': 0.0597551572033678,
 'AMT_REQ_CREDIT_BUREAU_MON+AMT_BALANCE': 0.0859938643022511,
```

```
'AMT_REQ_CREDIT_BUREAU_MON+AMT_CREDIT_LIMIT_ACTUAL': -0.0593329658796529564
'AMT_REQ_CREDIT_BUREAU_MON+MONTHS_BALANCE': 0.05920894911424972 5,
'AMT_REQ_CREDIT_BUREAU_QRT+AMT_BALANCE': 0.08599401691771937,
'AMT_REQ_CREDIT_BUREAU_QRT+AMT_CREDIT_LIMIT_ACTUAL': -0.05902444374387306,
'AMT_REQ_CREDIT_BUREAU_QRT+MONTHS_BALANCE': 0.05950026831492447,
'AMT_REQ_CREDIT_BUREAU_WEEK+AMT_BALANCE': 0.0859940590284539,
'AMT_REQ_CREDIT_BUREAU_WEEK+AMT_CREDIT_LIMIT_ACTUAL': -0.058845452673477 6,
'AMT_REQ_CREDIT_BUREAU_WEEK+MONTHS_BALANCE': 0.059749492776338245,
'AMT_REQ_CREDIT_BUREAU_YEAR+AMT_BALANCE': 0.08599434083720722,
'AMT_REQ_CREDIT_BUREAU_YEAR+AMT_CREDIT_LIMIT_ACTUAL': -0.05786657602976457
'AMT_REQ_CREDIT_BUREAU_YEAR+MONTHS_BALANCE': 0.060524196891697656,
'CNT_CHILDREN+AMT_BALANCE': 0.08717732005600688,
'CNT_CHILDREN+AMT_CREDIT_LIMIT_ACTUAL': -0.06015646852519142,
'CNT_CHILDREN+MONTHS_BALANCE': 0.06165421978175474,
'CNT_FAM_MEMBERS+AMT_BALANCE': 0.08717729242917417,
'CNT_FAM_MEMBERS+AMT_CREDIT_LIMIT_ACTUAL': -0.06020014822737341,
'CNT_FAM_MEMBERS+MONTHS_BALANCE': 0.061595942635955345,
'DAYS_BIRTH+AMT_BALANCE': 0.08930195615011871,
'DAYS_BIRTH+AMT_CREDIT_LIMIT_ACTUAL': 0.0667169352971932,
'DAYS_BIRTH+MONTHS_BALANCE': 0.06756082765439705,
'DAYS_EMPLOYED+AMT_BALANCE': 0.0283506722684889,
'DAYS_EMPLOYED+AMT_CREDIT_LIMIT_ACTUAL': -0.0373729895632540 2,
'DAYS_EMPLOYED+MONTHS_BALANCE': -0.03734235500487899,
'DAYS_ID_PUBLISH+AMT_BALANCE': 0.08775816565874109,
'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': 0.06858541348673414,
'DAYS_LAST_PHONE_CHANGE+MONTHS_BALANCE': 0.07165195246297092,
'DAYS_REGISTRATION+AMT_BALANCE': 0.0882601658724748,
'DAYS_REGISTRATION+AMT_CREDIT_LIMIT_ACTUAL': 0.03762026246667104 4,
'DAYS_REGISTRATION+MONTHS_BALANCE': 0.038737361009939145,
'DEF_30_CNT_SOCIAL_CIRCLE+AMT_BALANCE': 0.08719248874954717,
'DEF_30_CNT_SOCIAL_CIRCLE+AMT_CREDIT_LIMIT_ACTUAL': -0.06019992862873247,
'DEF_30_CNT_SOCIAL_CIRCLE+MONTHS_BALANCE': 0.06176236184358404,
'DEF_60_CNT_SOCIAL_CIRCLE+AMT_BALANCE': 0.08719247567028855,
'DEF_60_CNT_SOCIAL_CIRCLE+AMT_CREDIT_LIMIT_ACTUAL': -0.0602570386877368 7,
'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
```


Baseline for all data

```
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')
```

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

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACTIVE"
]

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_PUBLISH",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL"
]
```

```
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())
])
```

```
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, test_size=0.

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 = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full_pipeline, c
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)
```

```
results.loc[2] = ["Baseline", roc,pct(logit_score_train), np.round(pct(logit_score
                  train_time, test_time, "LogisticRegression + other datasets"]
results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
1	Baseline	0.739299	92.0	91.7	8.7200	0.4097	LogisticRegression + new Application features
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	+ other datasets

This doesn't really improve our ROC_AUC score very much

Baseline with new feature for all data

```
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_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

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

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACTIVE",
]

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_PUBLISH",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_PRICE",
  "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_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

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

```
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())
])
```

```
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, test_size=0.

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 = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full_pipeline, c
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)
```

```
results.loc[3] = ["Baseline", roc,pct(logit_score_train), np.round(pct(logit_score
                   train_time, test_time, "LogisticRegression + other datasets + n
results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
1	Baseline	0.739299	92.0	91.7	8.7200	0.4097	LogisticRegression + new Application features
							LogisticRegression

The new features that we created really seem to help our AUC score

LogisticRegression

More Data

Let's look at how adding even more data affects our model

```
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_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "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_ACTIVE",
]

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_PUBLISH",
  "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_PRICE",
  "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_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

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

```
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())
])
```

```
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, test_size=0.

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 = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full_pipeline, c
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)
```

```
results.loc[4] = ["Baseline", roc,pct(logit_score_train), np.round(pct(logit_score
                   train_time, test_time, "LogisticRegression + even more data"]
results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
1	Baseline	0.739299	92.0	91.7	8.7200	0.4097	LogisticRegression + new Application features
2	Baseline	0.740311	92.0	91.7	10.4090	0.5733	LogisticRegression + other datasets
3	Baseline	0.745049	92.0	91.7	13.2626	0.5951	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92.0	91.7	15.3700	0.6871	LogisticRegression + even more data

Tweaking Imputers

Perhaps we should be using the categorical imputer with a constant strategy. Instead of assigning NaN data with the most frequent category, maybe we should instead create a new category for all of this data. This would deal with certain categories, like employment data types, where it seemed that unemployed clients were labeled as NaN and shouldn't be grouped in with other categories.

```
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_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

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

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACTIVE",
]

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_PUBLISH",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_PRICE",
  "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_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

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

```
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())
])
```

```
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, test_size=0.

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 = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full_pipeline, c
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)
```

```
results.loc[5] = ["Baseline", roc,pct(logit_score_train), np.round(pct(logit_score
                   train_time, test_time, "LogisticRegression w/ Constant Imputer"
results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
1	Baseline	0.739299	92.0	91.7	8.7200	0.4097	LogisticRegression + new Application features
2	Baseline	0.740311	92.0	91.7	10.4090	0.5733	LogisticRegression + other datasets
3	Baseline	0.745049	92.0	91.7	13.2626	0.5951	LogisticRegression + other datasets +
5	Baseline	0.747196	92.0	91.7	12.2126	0.5966	LogisticRegression w/ Constant

Since this test doesn't include the "even more data" from the previous test, we are comparing it with experiment 3, and it performs better. We should continue using this change to the imputer in the future

Untuned LGBM

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

```
from lightgbm import LGBMClassifier
```

```
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_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

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

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACTIVE",
]

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_PUBLISH",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_PRICE",
  "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_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

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

```
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())
])
```

```
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, test_size=0.

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 = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full_pipeline, c
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)
```

```
results.loc[6] = ["LGBM", roc,pct(logit_score_train), np.round(pct(logit_score_tes
                   train_time, test_time, "Untuned LGBM"]
results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
1	Baseline	0.739299	92.0	91.7	8.7200	0.4097	LogisticRegression + new Application features
2	Baseline	0.740311	92.0	91.7	10.4090	0.5733	LogisticRegression + other datasets
3	Baseline	0.745049	92.0	91.7	13.2626	0.5951	LogisticRegression + other datasets + new feat...

As you can see, it performs the best of any models we have yet. We still need to tune it, and can add more data

5	Baseline	0.747196	92.0	91.7	12.2126	0.5966	w/ Constant

Adding New Dataset Features

These features are from aggregating the Bureau dataset by the `bureau_balance` dataset and from using SK_ID_PREV to collect more data for the previous application dataset.

```
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_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "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_ACTIVE",

  "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_PUBLISH",
  "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_PRICE",
  "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_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

  "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"
]
```

```
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())
])
```

```
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, test_size=0.

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 = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full_pipeline, c
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)
```

```
results.loc[7] = ["LGBM", roc,pct(logit_score_train), np.round(pct(logit_score_tes
                   train_time, test_time, "Untuned LGBM + aggregated datasets"]
results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
1	Baseline	0.739299	92.0	91.7	8.7200	0.4097	LogisticRegression + new Application features
2	Baseline	0.740311	92.0	91.7	10.4090	0.5733	LogisticRegression + other datasets
3	Baseline	0.745049	92.0	91.7	13.2626	0.5951	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92.0	91.7	15.3700	0.6871	LogisticRegression + even more data
5	Baseline	0.747196	92.0	91.7	12.2126	0.5966	LogisticRegression w/ Constant

These new features clearly make our model better. Now, we need to do optimize it using Grid Search

Untuned LGBM +

Grid Search for LGBM

Since Grid Search takes a while when testing for larger values of n_estimators, we tested all other hyperparameters before testing n_estimators

```
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_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "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_ACTIVE",

  "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_PUBLISH",
  "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_PRICE",
  "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_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

  "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"
]
```

```
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.model_selection import GridSearchCV
from lightgbm import LGBMClassifier
```

```
# 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_with_predictor = Pipeline([
  ("preprocessing", preprocess_pipeline),
  ("predictor", LGBMClassifier(colsample_bytree=0.1, max_depth=5, n_estimators=100
])

# Execute the grid search
params = {
    '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],
}

grid_search = GridSearchCV(full_pipeline_with_predictor, params, scoring='roc_auc'
                           n_jobs=-1, verbose=True)
```

```
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, test_size=
grid_search.fit(X_train, y_train)

best_train = pct(grid_search.best_score_)
print("best train score: {}".format(best_train))

print("Best Parameters:")
best_parameters = grid_search.best_estimator_.get_params()
param_dump = []
for param_name in sorted(params.keys()):
  param_dump.append((param_name, best_parameters[param_name]))
  print("\t"+str(param_name)+": " + str(best_parameters[param_name]))
```

```
Fitting 3 folds for each of 192 candidates, totalling 576 fits
best train score: 76.3
Best Parameters:
	predictor__colsample_bytree: 0.5
	predictor__max_depth: 10
	predictor__min_split_gain: 1
	predictor__num_leaves: 31
```

Tuned LGBM

Let's add the extra data that we got from the "More Data" experiment to an LGBM model with tuned hyperparameters

```
from lightgbm import LGBMClassifier
```

```
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_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "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_ACTIVE",

  "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_PUBLISH",
  "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_PRICE",
  "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_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

  "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"
]
```

```
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_estimators=50
])
```

```
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, test_size=0.

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 = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full_pipeline, c
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)
```

```
results.loc[8] = ["LGBM", roc,pct(logit_score_train), np.round(pct(logit_score_tes
                  train_time, test_time, "LGBM tuned"]
results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92	91.7	8.83960	0.398000	LogisticRegression
1	Baseline	0.739299	92	91.7	8.72000	0.409700	LogisticRegression + new Application features
2	Baseline	0.740311	92	91.7	10.40900	0.573300	LogisticRegression + other datasets
3	Baseline	0.745049	92	91.7	13.26260	0.595100	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92	91.7	15.37000	0.687100	LogisticRegression + even more data
5	Baseline	0.747196	92	91.7	12.21260	0.596600	LogisticRegression w/ Constant Imputer
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
8	LGBM	0.763879	92	91.7	42.14050	2.552300	LGBM tuned

Feature Importances and Analysis

Let's take a close look at our feature importances

```
# from matplotlib import pyplot as plt
model = full_pipeline.steps[1][1]
features = list(full_pipeline.steps[0][1].transformer_list[1][1].steps[2][1].get_f

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

	Value	Feature
61	0	OCCUPATION_TYPE_Security staff
60	0	OCCUPATION_TYPE_Secretaries
59	0	OCCUPATION_TYPE_Sales staff
58	0	OCCUPATION_TYPE_Realty agents
89	0	CREDIT_TYPE_Car loan
91	0	CREDIT_TYPE_Consumer credit
132	0	DAYS_ID_PUBLISH
114	0	STATUS_MAX_0
131	0	DAYS_BIRTH
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
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 best correlation data is left as totally unused by our model, such as `FLOORSMIN_AVG` and `ELEVATORS_AVG`. This may be because they weren't useful since this data was missing for a large amount of samples. This also might be because it's highly correlated with other similar features, like `TOTALAREA_MODE`.

Unfortunately, it looks like `CREDIT_ACTIVE` was almost a completely useless feature.

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

	Value	Feature
1	614	FLAG_DOCUMENT_3_1
39	596	HOUR_APPR_PROCESS_START_18
0	545	FLAG_DOCUMENT_3_0
15	521	NAME_INCOME_TYPE_Working
42	514	HOUR_APPR_PROCESS_START_21
2	513	REGION_RATING_CLIENT_1
30	499	HOUR_APPR_PROCESS_START_9
9	495	NAME_INCOME_TYPE_Commercial associate
19	445	NAME_EDUCATION_TYPE_Lower secondary
25	442	HOUR_APPR_PROCESS_START_4
22	433	HOUR_APPR_PROCESS_START_1
4	432	REGION_RATING_CLIENT_3
24	418	HOUR_APPR_PROCESS_START_3
10	411	NAME_INCOME_TYPE_Maternity leave
26	406	HOUR_APPR_PROCESS_START_5
37	401	HOUR_APPR_PROCESS_START_16
34	390	HOUR_APPR_PROCESS_START_13
5	383	REGION_RATING_CLIENT_W_CITY_1
8	375	NAME_INCOME_TYPE_Businessman
20	370	NAME_EDUCATION_TYPE_Secondary / secondary special
36	364	HOUR_APPR_PROCESS_START_15
35	363	HOUR_APPR_PROCESS_START_14

41	354	HOUR_APPR_PROCESS_START_20
12	322	NAME_INCOME_TYPE_State servant
7	282	REGION_RATING_CLIENT_W_CITY_3
13	277	NAME_INCOME_TYPE_Student
43	272	HOUR_APPR_PROCESS_START_22
28	267	HOUR_APPR_PROCESS_START_7
29	234	HOUR_APPR_PROCESS_START_8
16	214	NAME_EDUCATION_TYPE_Academic degree
44	202	HOUR_APPR_PROCESS_START_23

The best pieces of data are all categorical. Specifically, it looks like `HOUR_APPR_PROCESS_START` and `FLAG_DOCUMENT_3` were very informative. It's interesting that the flags for the other documents were far less informative, since `FLAG_DOCUMENT_4` is in the data set.

There aren't any important numerical features, at least as far as we can see.

```
imp_df = pd.DataFrame({'Value':model.feature_importances_,'Feature':features})
print("Unimportant data points: {}%".format(round(sum(imp_df["Value"] == 0) / len(
```

```
Unimportant data points: 34.68%
```

Overall, it looks like around 34% of our features weren't used by our model. While a few of them mentioned above were numerical features, some categorical features, like `OCCUPATION_TYPE`, had data points in both the most useful and least useful features, since there were a lot of possible values.

Test Data and Kaggle Submission

[ ] ↳ *3 cells hidden*

Untuned XGBoost

Now that we've taken a deep dive into our LGBM model, let's explore what XGBoost has to offer

```
from xgboost import XGBClassifier
```

```
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_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "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_ACTIVE",

  "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_PUBLISH",
  "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_PRICE",
  "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_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

  "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"
]
```

```
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', XGBClassifier())
])
```

```
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, test_size=0.

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 = 0)
logit_scores = cross_val_score(X=X_train,y = y_train, estimator = full_pipeline, c
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)
```

```
[22:38:14] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the de
[22:42:02] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the de
[22:44:38] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the de
[22:47:14] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the de
[22:49:47] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the de
[22:52:20] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the de
[22:54:57] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the de
[22:57:33] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the de
[23:00:09] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the de
[23:02:42] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the de
[23:05:16] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the de
```

```
results.loc[9] = ["XGBoost", roc,pct(logit_score_train), np.round(pct(logit_score_
                  train_time, test_time, "Untuned XGBoost"]
results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
1	Baseline	0.739299	92.0	91.7	8.7200	0.4097	LogisticRegression + new Application features
2	Baseline	0.740311	92.0	91.7	10.4090	0.5733	LogisticRegression + other datasets
3	Baseline	0.745049	92.0	91.7	13.2626	0.5951	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92.0	91.7	15.3700	0.6871	LogisticRegression + even more data
5	Baseline	0.747196	92.0	91.7	12.2126	0.5966	LogisticRegression w/ Constant datasets

Deep Learning Model

For simplicity, we will train this model on application data for 100 epochs

```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
app = datasets["application_train"]
train_x = app.loc[:, app.columns != "TARGET"]
train_y = app["TARGET"]
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y, test_size=0.
```

```
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
```

```
# preprocess data
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "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_PUBLISH"
]

# 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)
```

```
# 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()
```

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

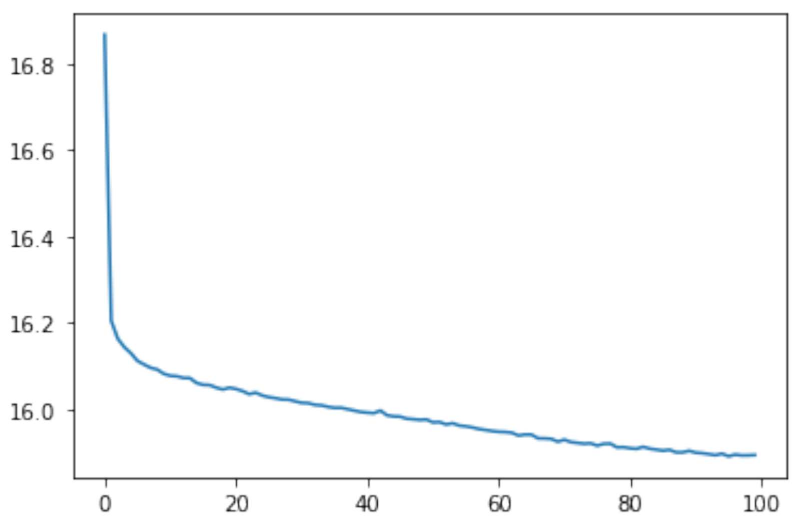
```
# create data loaders
train_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_tensor)
data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffl
```

```
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()
```

```
losses = []
epochs = num_epochs
start = time()
for epoch in range(epochs):
  running_loss = 0.0
  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

  losses.append(running_loss/batch_size)
train_time = time() - start
```

```
import matplotlib.pyplot as plt

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

```
[<matplotlib.lines.Line2D at 0x7f46db9d5ed0>]
```




```
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
```

```
results.loc[10] = ["Deep Learning", roc, "--", "--",
                   train_time, test_time, "Deep Learning w/ Application Data"]

results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92	91.7	8.83960	0.398000	LogisticRegression
1	Baseline	0.739299	92	91.7	8.72000	0.409700	LogisticRegression + new Application features
2	Baseline	0.740311	92	91.7	10.40900	0.573300	LogisticRegression + other datasets
3	Baseline	0.745049	92	91.7	13.26260	0.595100	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92	91.7	15.37000	0.687100	LogisticRegression + even more data
5	Baseline	0.747196	92	91.7	12.21260	0.596600	LogisticRegression w/ Constant Imputer
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
7	LGBM	0.763666	92	91.8	15.14810	1.240400	Untuned LGBM + aggregated datasets
8	LGBM	0.765221	80.7	79.9	42.38470	2.595000	LGBM tuned
9	XGBoost	0.756850	91.9	91.7	230.20260	1.238900	Untuned XGBoost
10	Deep Learning	0.739001	--	--	397.18878	0.050515	Deep Learning w/ Application Data