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
```

```
# 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Warning: Looks like you're using an outdated API Version, please consider upd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 credit_card_balance.csv.zip to home-credit-default-risk
 75% 73.0M/96.7M [00:01<00:00, 50.9MB/s]
100% 96.7M/96.7M [00:01<00:00, 68.9MB/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, 33.6MB/s]
Downloading application_train.csv.zip to home-credit-default-risk
 47% 17.0M/36.1M [00:01<00:01, 19.3MB/s]
100% 36.1M/36.1M [00:01<00:00, 33.9MB/s]
Downloading POS_CASH_balance.csv.zip to home-credit-default-risk
 90% 98.0M/109M [00:02<00:00, 34.6MB/s]
100% 109M/109M [00:02<00:00, 52.3MB/s]
Downloading previous_application.csv.zip to home-credit-default-risk
 85% 65.0M/76.3M [00:01<00:00, 20.9MB/s]
100% 76.3M/76.3M [00:01<00:00, 56.2MB/s]
Downloading installments_payments.csv.zip to home-credit-default-risk
 93% 253M/271M [00:06<00:00, 19.3MB/s]
100% 271M/271M [00:06<00:00, 43.8MB/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, 168MB/s]
Downloading bureau.csv.zip to home-credit-default-risk
 90% 33.0M/36.8M [00:01<00:00, 19.9MB/s]
100% 36.8M/36.8M [00:01<00:00, 31.4MB/s]
Downloading application_test.csv.zip to home-credit-default-risk
 86% 5.00M/5.81M [00:00<00:00, 15.0MB/s]
100% 5.81M/5.81M [00:00<00:00, 16.7MB/s]
Downloading bureau_balance.csv.zip to home-credit-default-risk
 58% 33.0M/56.8M [00:01<00:01, 17.6MB/s]
100% 56.8M/56.8M [00:01<00:00, 45.2MB/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()
```

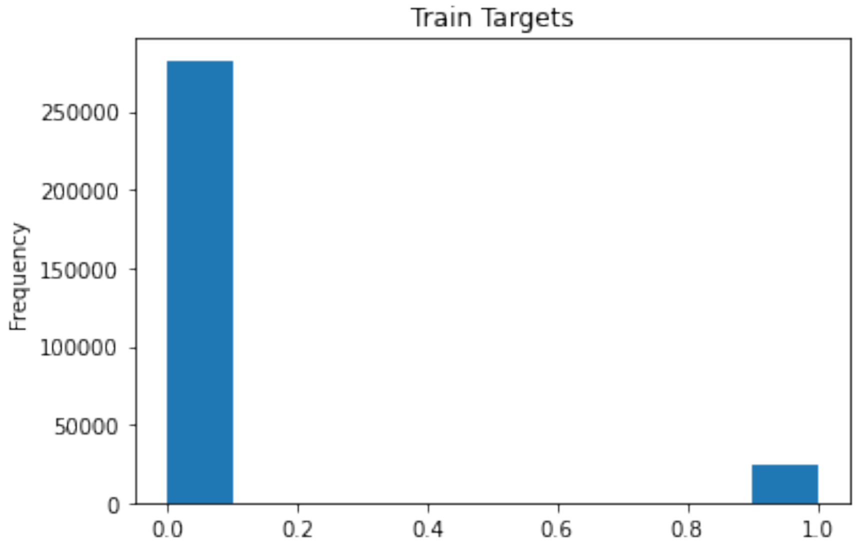
```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance
```

Analyze Application data

Since our application data is easily accessible, we should examine it first

```
datasets["application_train"]['TARGET'].astype(int).plot.hist(title="Train Targets
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7b00106650>
```



Our dataset is really heavily unbalanced. This will make it harder to visually see relationships between data and targets

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

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

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

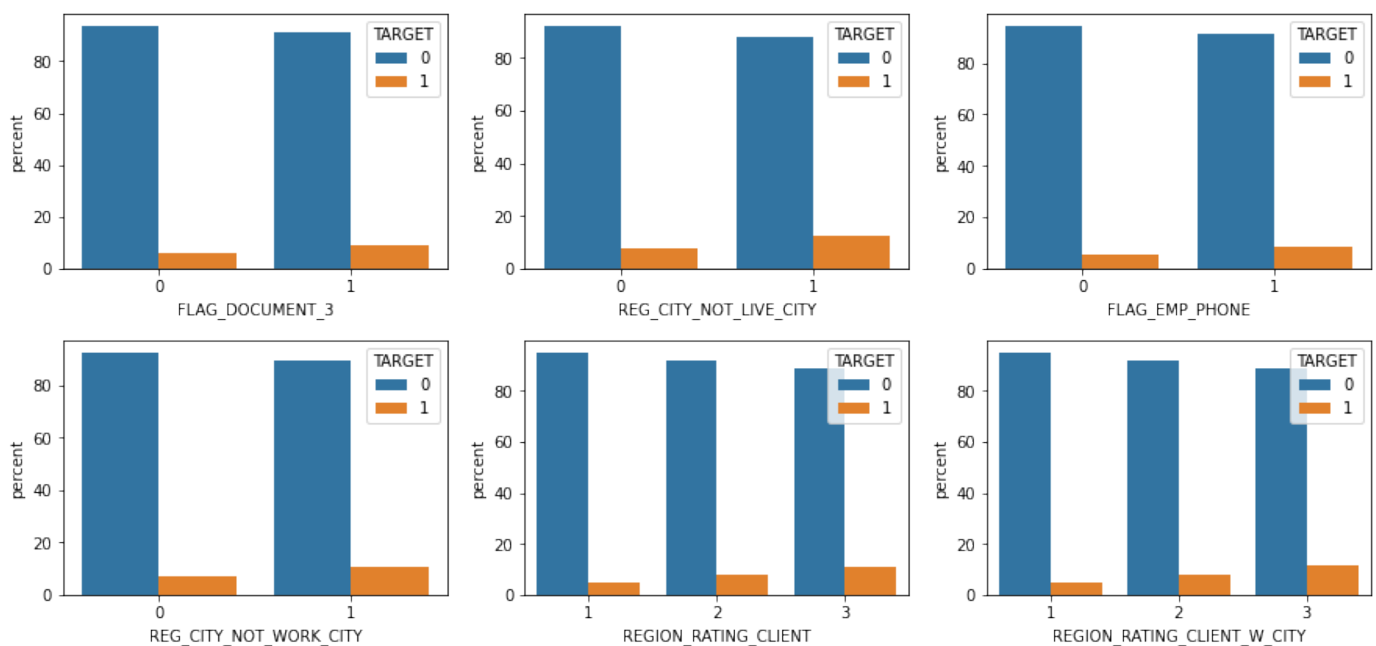
```
all_cat_features = [
  "NAME_CONTRACT_TYPE", "CODE_GENDER", "FLAG_OWN_CAR", "NAME_TYPE_SUITE",
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "NAME_FAMILY_STATUS", "NAME_HOUSING_T
  "FLAG_MOBIL", "FLAG_EMP_PHONE", "FLAG_WORK_PHONE", "FLAG_CONT_MOBILE",
  "FLAG_PHONE", "FLAG_EMAIL", "OCCUPATION_TYPE", "REGION_RATING_CLIENT", "REGION_R
  "WEEKDAY_APPR_PROCESS_START", "REG_REGION_NOT_LIVE_REGION", "REG_REGION_NOT_WORK
  "LIVE_REGION_NOT_WORK_REGION", "REG_CITY_NOT_LIVE_CITY", "REG_CITY_NOT_WORK_CITY
  "LIVE_CITY_NOT_WORK_CITY", "ORGANIZATION_TYPE", "FONDKAPREMONT_MODE", "HOUSETYPE
  "WALLSMATERIAL_MODE", "EMERGENCYSTATE_MODE", "FLAG_DOCUMENT_2", "FLAG_DOCUMENT_3
  "FLAG_DOCUMENT_4", "FLAG_DOCUMENT_5", "FLAG_DOCUMENT_6", "FLAG_DOCUMENT_7", "FLA
  "FLAG_DOCUMENT_9", "FLAG_DOCUMENT_10", "FLAG_DOCUMENT_11", "FLAG_DOCUMENT_12", "
  "FLAG_DOCUMENT_14", "FLAG_DOCUMENT_15", "FLAG_DOCUMENT_16", "FLAG_DOCUMENT_17",
  "FLAG_DOCUMENT_18", "FLAG_DOCUMENT_19", "FLAG_DOCUMENT_20", "FLAG_DOCUMENT_21",
  "HOUR_APPR_PROCESS_START"
]
all_num_features = [
  "CNT_CHILDREN", "AMT_INCOME_TOTAL", "AMT_CREDIT", "AMT_ANNUITY", "REGION_POPULAT
  "DAYS_BIRTH", "DAYS_EMPLOYED", "DAYS_REGISTRATION", "OWN_CAR_AGE", "CNT_FAM_MEMB
  "EXT_SOURCE_1", "EXT_SOURCE_2", "EXT_SOURCE_3", "APARTMENTS_AVG", "AMT_GOODS_PRI
  "BASEMENTAREA_AVG", "YEARS_BEGINEXPLUATATION_AVG", "YEARS_BUILD_AVG", "COMMONARE
  "ELEVATORS_AVG", "ENTRANCES_AVG", "FLOORSMAX_AVG", "FLOORSMIN_AVG", "LANDAREA_AV
  "LIVINGAPARTMENTS_AVG", "LIVINGAREA_AVG", "NONLIVINGAPARTMENTS_AVG", "NONLIVINGA
  "APARTMENTS_MODE", "BASEMENTAREA_MODE", "YEARS_BEGINEXPLUATATION_MODE", "YEARS_B
  "COMMONAREA_MODE", "ELEVATORS_MODE", "ENTRANCES_MODE", "FLOORSMAX_MODE", "FLOORS
  "LANDAREA_MODE", "LIVINGAPARTMENTS_MODE", "LIVINGAREA_MODE", "NONLIVINGAPARTMENT
  "NONLIVINGAREA_MODE", "APARTMENTS_MEDI", "BASEMENTAREA_MEDI", "YEARS_BEGINEXPLUA
  "YEARS_BUILD_MEDI", "COMMONAREA_MEDI", "ELEVATORS_MEDI", "ENTRANCES_MEDI", "FLOO
  "FLOORSMIN_MEDI", "LANDAREA_MEDI", "LIVINGAPARTMENTS_MEDI", "LIVINGAREA_MEDI",
  "NONLIVINGAPARTMENTS_MEDI", "NONLIVINGAREA_MEDI", "TOTALAREA_MODE", "OBS_30_CNT_
  "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",
]
```

Analysis of Categorical Features

Let's see if there's any interesting visual correlations between the categorical features and the target. NOTE: the below graphs are in percentages, not in absolute terms. This makes it much easier to see the difference in percentages as opposed to the absolute differences

Let's take a look at our most correlated features first.

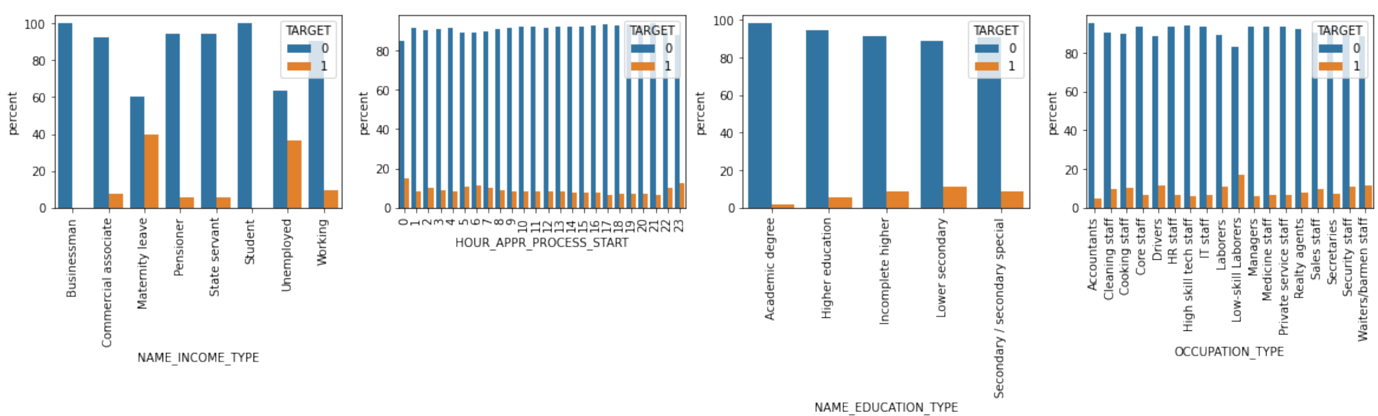
```
import seaborn as sns
import matplotlib.pyplot as plt

cat_corr_features1 = [
  "FLAG_DOCUMENT_3", "REG_CITY_NOT_LIVE_CITY", "FLAG_EMP_PHONE",
]

cat_corr_features2 = [
  "REG_CITY_NOT_WORK_CITY", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
]

app_train_df = datasets['application_train']
fig = plt.figure(figsize=(15,3))
for idx, cat in enumerate(cat_corr_features1):
  df = app_train_df.groupby(cat)['TARGET'].value_counts(normalize=True).mul(100).r
  ax = fig.add_subplot(int("13{}".format(idx+1)))
  sns.barplot(x=cat,y='percent',hue='TARGET',data=df,ax=ax)

fig = plt.figure(figsize=(15,3))
for idx, cat in enumerate(cat_corr_features2):
  df = app_train_df.groupby(cat)['TARGET'].value_counts(normalize=True).mul(100).r
  ax = fig.add_subplot(int("13{}".format(idx+1)))
  sns.barplot(x=cat,y='percent',hue='TARGET',data=df,ax=ax)
```



The most obvious splits here seem to be `REGION_RATING_CLIENT` and `REG_CITY_NOT_WORK_CITY`, just from visual inspection. However, it appears that there are no obvious splits that can be made from just visually inspecting the histogram, even on our most correlated fields.

Let's look at some more interesting fields

```
cat_corr_features_i = [
  "NAME_INCOME_TYPE", "HOUR_APPR_PROCESS_START", "NAME_EDUCATION_TYPE", "OCCUPATIO
]

app_train_df = datasets['application_train']
fig = plt.figure(figsize=(20,3))
for idx, cat in enumerate(cat_corr_features_i):
  df = app_train_df.groupby(cat)['TARGET'].value_counts(normalize=True).mul(100).r
  ax = fig.add_subplot(int("14{}".format(idx+1)))
  plt.xticks(rotation=90)
  sns.barplot(x=cat,y='percent',hue='TARGET',data=df,ax=ax)
```



We see a number of interesting things here.

According to `NAME_INCOME_TYPE`, Businessmen and Students almost never have problems with repayment, while those on maternity leave or unemployed are more likely to have difficulty repaying the loan.

According to `HOUR_APPR_PROCESS_START`, Those that file extremely late/early in the day have more issues repaying as well. This trend goes down throughout the afternoon, but picks up again at around 11pm. There appears to be a noticeable dip at around 6pm, perhaps this is because people that work a 9-5 would normally start their application after work hours.

`NAME_EDUCATION_TYPE` also shows some clear trends. The more educated the person applying for the loan, the more difficulty they are expected to have repaying it. This is interesting, as according to `NAME_INCOME_TYPE`, Students almost never have problem repaying.

Lastly, `OCCUPATION_TYPE` shows some trends. The most likely to have problems in repayment are Low-skill Laborers and the least likely are Accountants. Drivers, Laborers, and Waiters/barmen staff also tend to have more problems with repayment than average.

Missing Values

```
df = datasets['application_train'][all_cat_features]
percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending = Fals
sum_missing = df.isna().sum().sort_values(ascending = False)
missing_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Missin
missing_data.head(8)
```

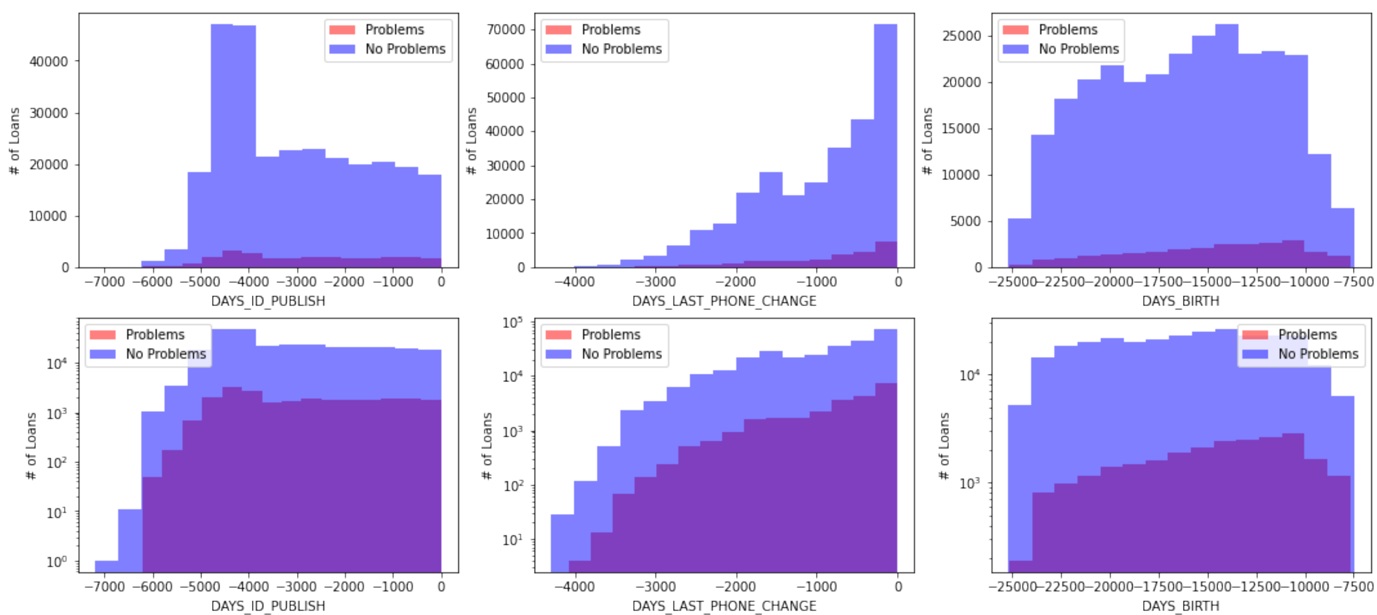
	Percent	Missing Count
FONDKAPREMONT_MODE	68.39	210295
WALLSMATERIAL_MODE	50.84	156341
HOUSETYPE_MODE	50.18	154297
EMERGENCYSTATE_MODE	47.40	145755
OCCUPATION_TYPE	31.35	96391
NAME_TYPE_SUITE	0.42	1292
HOUR_APPR_PROCESS_START	0.00	0
FLAG_EMAIL	0.00	0

Only a few features are missing, and all of them don't seem very interesting except for `OCCUPATION_TYPE`. However, since unemployment wasn't listed, this is most-likely why.

Analysis of Numerical Features

Now that we've looked at our categorical features, let's turn our attention to the numerical features. Let's take a look at the most correlated features first.

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

  ax.hist(df[df["TARGET"]==1][y], bins=15, alpha=0.5, color="red", 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()

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

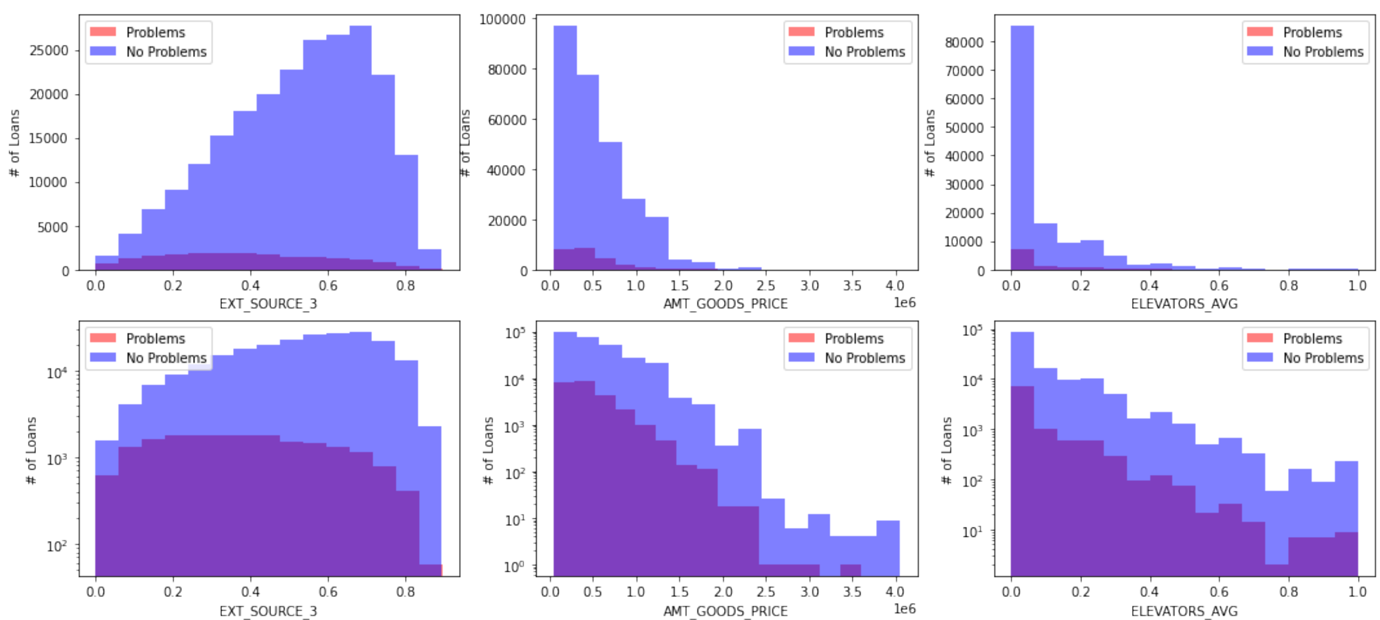


NOTE: these are the linear and log graphs for 3 different features. The log graph is directly below the linear one and is to help reading and interpreting the data

There are some interesing features here. It seems that the more recently the client changed their phone the more likely that there will be problems with repayment.

The age of the client also has some interesting features. The most problems seem to happen when the client is around 11,000 days old, or around 30 years old. The older or younger they are, the more capable they are of being able to repay the loan.

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

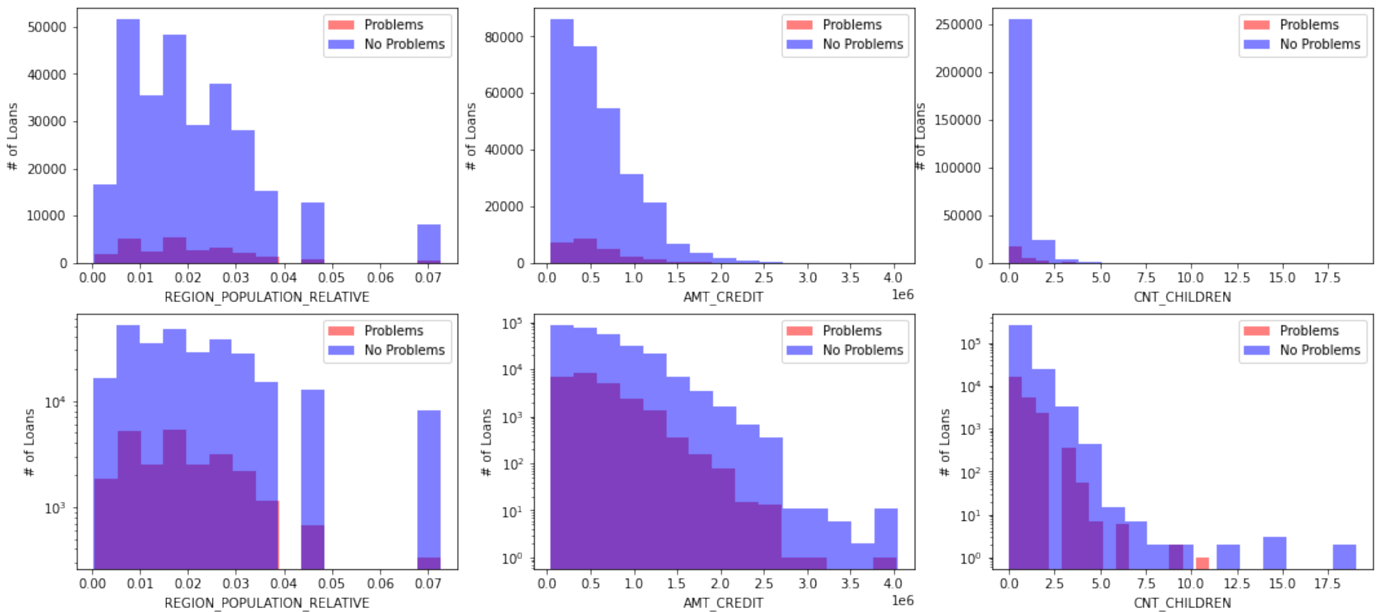


`EXT_SOURCE_3` was the most negatively-corrolated data field. It's data from an external source with no specific labels, so it's not very interesting in terms of making intuitive connections.

`AMT_GOODS_PRICE` is very negatively corrolated with repayment problems. The description on this is vague, but perhaps this refers to the collateral on the loan.

Interestingly, according to `ELEVATORS_AVG`, the more elevators that the building where the client lives has, the more likely they are to have problems repaying the loan. The number of floors is also highly correlated with the target, so it probably is related to rent price or neighborhood quality.

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



It seems that the higher the population where the client lives, the more successfully they can repay the loan. This might be due to the fact that it's more expensive to live in more populus areas.

We see that the more a person want to take as credit on a loan, the less problems they tend to have repaying it.

Also, the more children the client has, the harder it is for them to repay the loan without issues.

Missing Values

```
df = datasets['application_train'][all_num_features]
percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending = Fals
sum_missing = df.isna().sum().sort_values(ascending = False)
missing_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Missin
missing_data.head(54)
```

	Percent	Missing Count
COMMONAREA_MODE	69.87	214865
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865
NONLIVINGAPARTMENTS_MODE	69.43	213514
NONLIVINGAPARTMENTS_AVG	69.43	213514
NONLIVINGAPARTMENTS_MEDI	69.43	213514
LIVINGAPARTMENTS_AVG	68.35	210199
LIVINGAPARTMENTS_MODE	68.35	210199
LIVINGAPARTMENTS_MEDI	68.35	210199
FLOORSMIN_MODE	67.85	208642
FLOORSMIN_AVG	67.85	208642
FLOORSMIN_MEDI	67.85	208642
YEARS_BUILD_MEDI	66.50	204488
YEARS_BUILD_MODE	66.50	204488
YEARS_BUILD_AVG	66.50	204488
OWN_CAR_AGE	65.99	202929
LANDAREA_MEDI	59.38	182590
LANDAREA_MODE	59.38	182590
LANDAREA_AVG	59.38	182590
BASEMENTAREA_AVG	58.52	179943
BASEMENTAREA_MODE	58.52	179943
BASEMENTAREA_MEDI	58.52	179943

EXT_SOURCE_1	56.38	173378
NONLIVINGAREA_MODE	55.18	169682
NONLIVINGAREA_AVG	55.18	169682
NONLIVINGAREA_MEDI	55.18	169682
ELEVATORS_MEDI	53.30	163891
ELEVATORS_MODE	53.30	163891
ELEVATORS_AVG	53.30	163891
APARTMENTS_MEDI	50.75	156061
APARTMENTS_AVG	50.75	156061

Unfortunately, a lot of interesting features are missing in half or more of our data. Ideally, we'd like to avoid depending on this data.

`EXT_SOURCE_1` is highly corrolated with the target, as was information about the floors the client's house/apartment, but since it's missing in around half the rows, there might be a large bias to this type of data.

`EXT_SOURCE_3` is also missing in around 20% of the rows, which is pretty bad.

Pipeline 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())
])
```

Fit Baseline

```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_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)
np.random.seed(42)

cv30Splits = ShuffleSplit(n_splits = 30, 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()
train_time = np.round(time() - start, 4)

# 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, "Untuned LogisticRegression"]
```

```
results
```

ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description

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

Analyze Secondary Data

In order to improve our model, we should look at all the other data we have available. To do that, we need to aggregate data from sources besides the application files

```
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)
```

Credit Card Balances

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

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

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

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

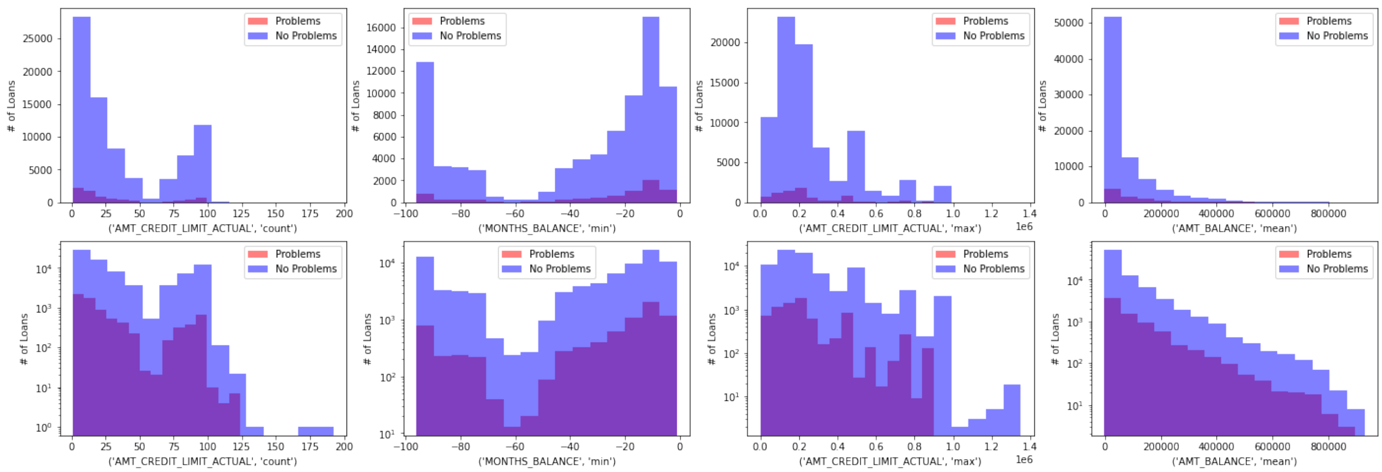
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_PREV              int64  
 1   SK_ID_CURR              int64  
 2   NUM_INSTALMENT_VERSION  float64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

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

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

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

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

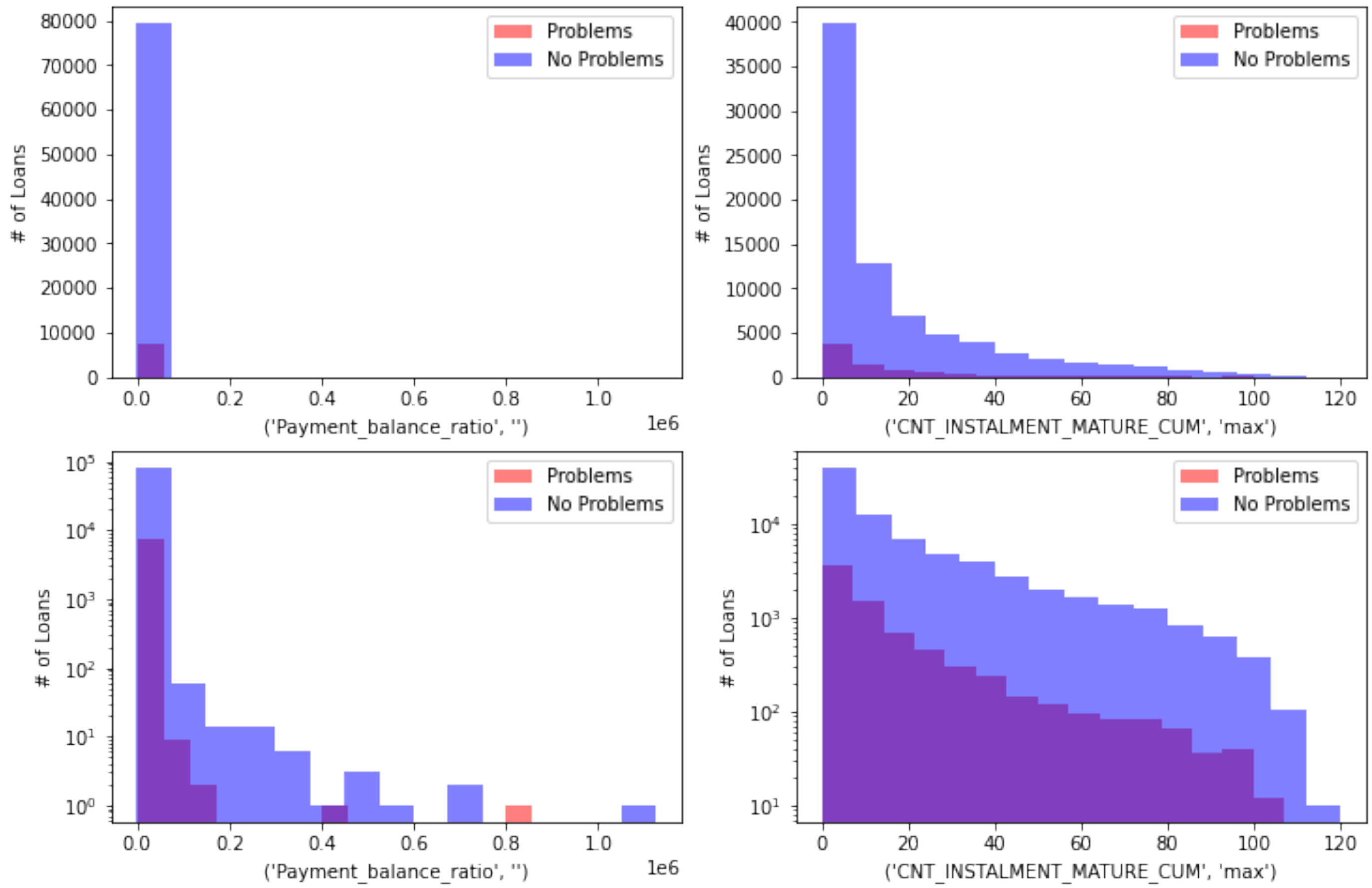
```
fig, axs = plt.subplots(2, 4, figsize=(24, 8))
num_hist(temp_app, ('AMT_CREDIT_LIMIT_ACTUAL', 'count'), axs[0,0])
num_hist(temp_app, ('MONTHS_BALANCE', 'min'), axs[0,1])
num_hist(temp_app, ('AMT_CREDIT_LIMIT_ACTUAL', 'max'), axs[0,2])
num_hist(temp_app, ('AMT_BALANCE', 'mean'), axs[0,3])
# log graphs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



It seems that clients with the highest credit limit have very few problems with repayment, which should be expected. The number of credit card balances seems to have a sort of 'U' shape. The few people at the extreme high end have no problems paying the loan, however.

It's hard to tell, but `AMT_BALANCE` might have some correlation with the target as well

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



It looks like `Payment_balance_ratio` is all concentrated around 0.0, so it hardly has any predictive power.

`CNT_INSTALMENT_MATURE_CUM` seems to have pretty good correlation, which makes sense. The more installments the client pays on their credit card, the less likely they are to have problems with a loan.

Correlations

```
temp_app[list(CCB_df.columns) + ["TARGET"]].corr()["TARGET"]
```

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

These are some strong correlation values. `AMT_BALANCE` refers to the balance on the credit card, so it makes sense that the higher the balance, the harder it will be to repay a loan. The closer to the time of application as the balance is due `MONTHS_BALANCE`, the harder it is to repay as well.

Missing Values

```
missing_vals(temp_app[CCB_df.columns])
```

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

Unfortunately, our highly correlated features have a lot of missing values. We might not be able to use these, since it only has predictive power in ~28% of the dataset.

Previous Applications

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

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

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

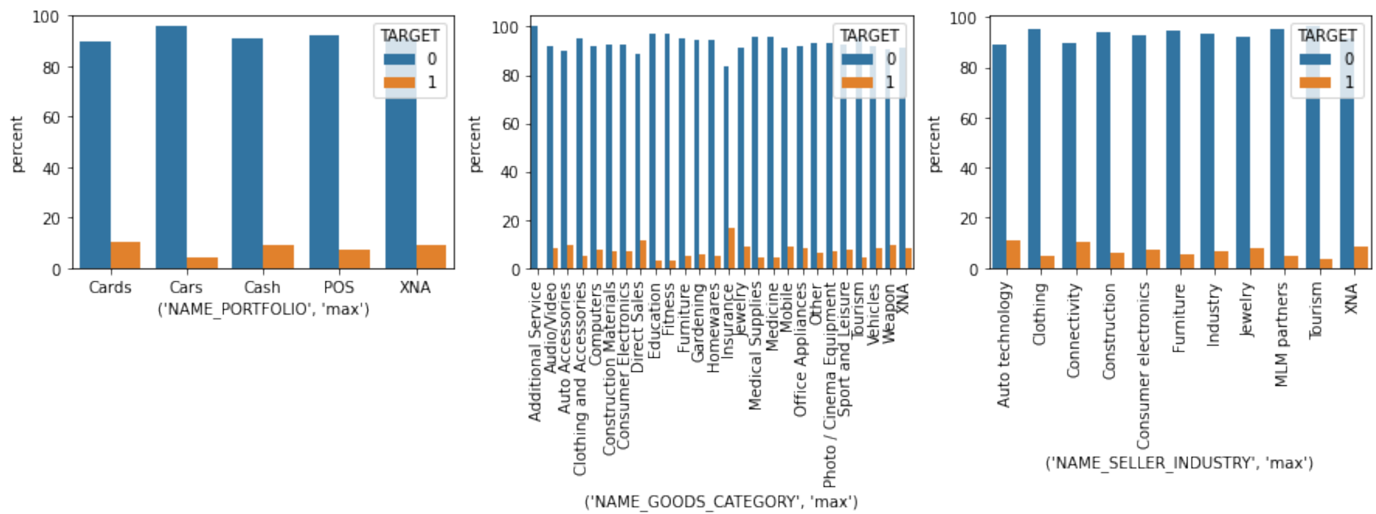
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_PREV              int64
 1   SK_ID_CURR              int64
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER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

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

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

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

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

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

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

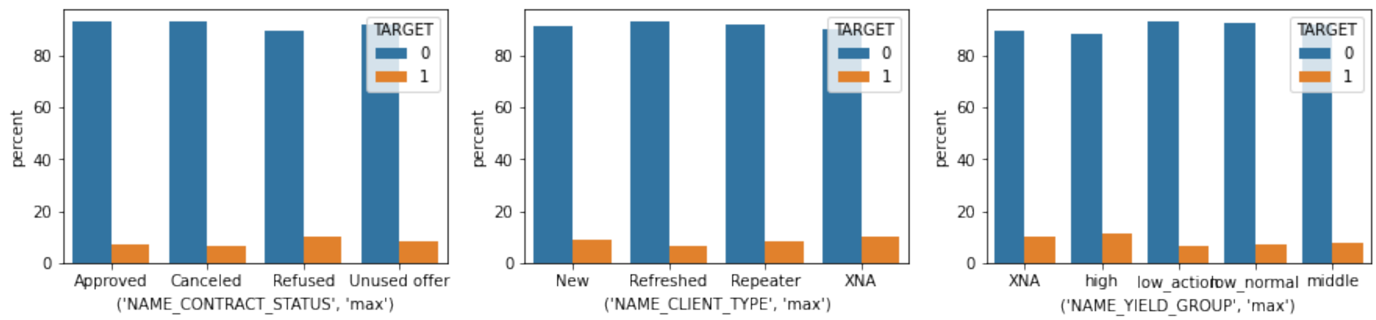


There's a lot of interesting points here. The `NAME_GOODS_CATEGORY` seems to really matter. 'Education', 'Fitness', and 'Tourism' have the least amount of repayment problems, while 'Insurance', 'Direct Sales', and 'Weapon' have some of the highest rates of problems.

In `NAME_PRODUCT_TYPE`, walk-ins have higher rates of problems than the other two, which could be interesting.

In `NAME_PORTFOLIO`, Car loans seem to have the least amount of problems, which is interesting. Cards have the highest rate

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

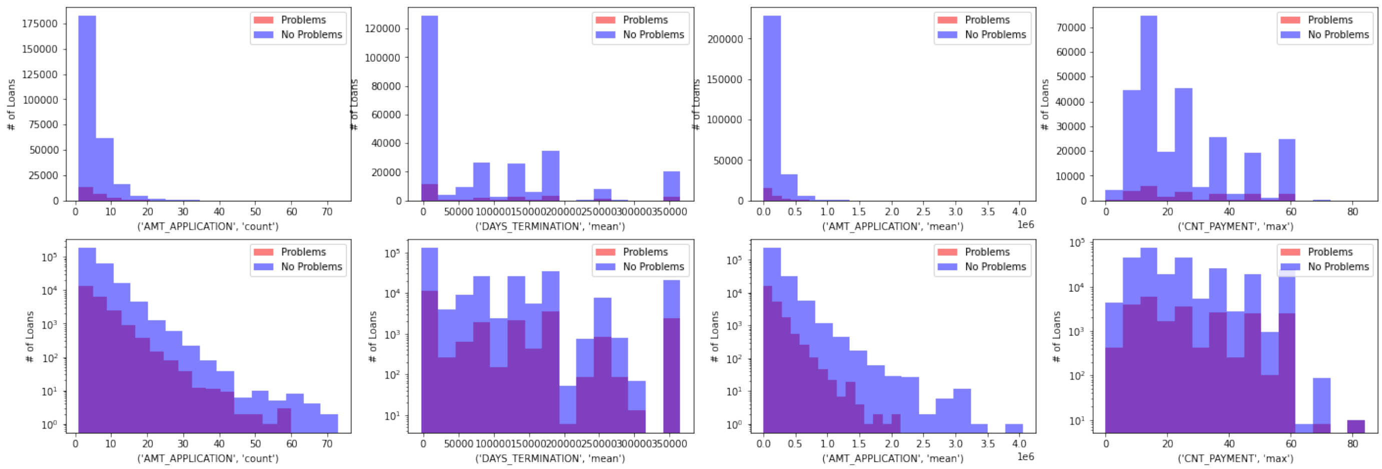


Although there's not a whole lot of variance in general, a few things stick out here. In `NAME_CONTRACT_STATUS`, the Refused clients have a higher probability of having payment issues.

In `NAME_CLIENT_TYPE`, 'Refreshed' clients have a lower probability of having problems.

In `NAME_YIELD_GROUP`, the low-yield clients have the fewest amount of problems, and 'XNA' and 'high' yield groups have the most amount of problems.

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



Although it's hard to see, it looks like clients with more applications and that request higher amounts tend to have more issues with repayment, although the difference seems small

Correlations

```
temp_app[list(PA_df.columns) + ["TARGET"]].corr()["TARGET"]
```

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

The correlations aren't great, but it's really the categories that look promising here

Missing Values

```
missing_vals(temp_app[PA_df.columns])
```

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

There's not a lot of missing data here, so we're safe to use these in our model without worrying too much about it

POS Cash Balance

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

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

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

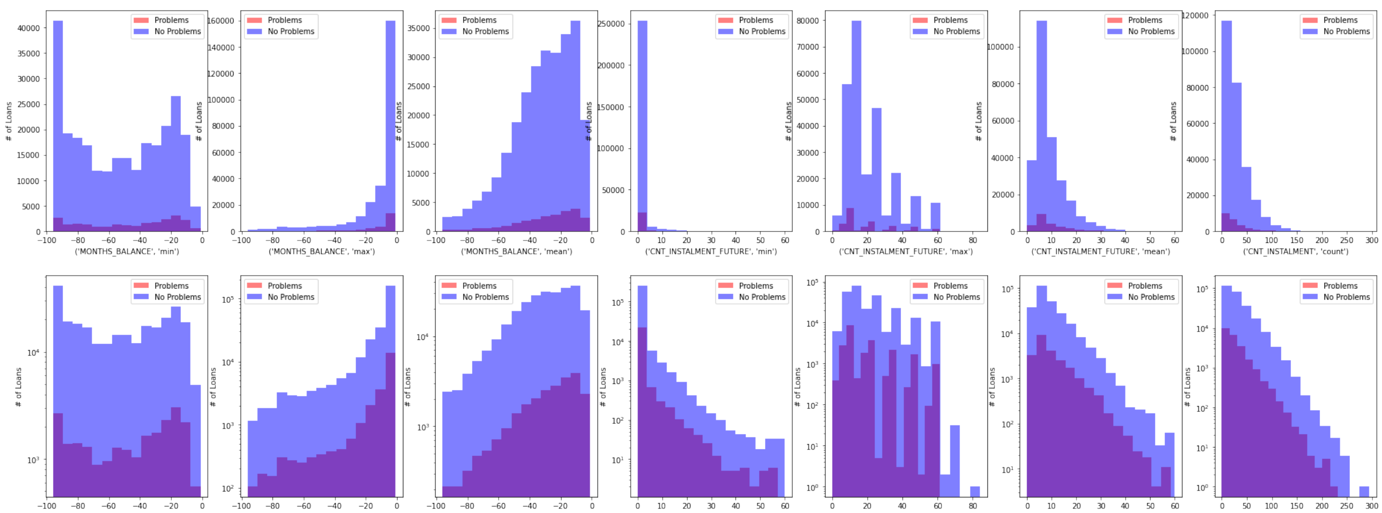
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_PREV              int64
 1   SK_ID_CURR              int64
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER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

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

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

})
```

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

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



MONTHS_BALANCE, min is very vaguely "U" shaped, though the differenece in problems to no problems is uniform throughout. Most of these graphs don't look like the difference between problems and no problems change very much, though CNT_INSTALMENT_FUTURE, max is very sporadic.

Correlations

```
temp_app[list(PCB_df.columns) + ["TARGET"]].corr()["TARGET"]
```

```
(CNT_INSTALMENT, count)          -0.035802
(CNT_INSTALMENT_FUTURE, max)      0.013324
(CNT_INSTALMENT_FUTURE, min)      0.019010
(CNT_INSTALMENT_FUTURE, mean)     0.027827
(MONTHS_BALANCE, max)            -0.004321
(MONTHS_BALANCE, min)             0.055307
(MONTHS_BALANCE, mean)            0.034543
(SK_DPD, max)                     0.004763
(SK_DPD, min)                     0.005444
(SK_DPD, mean)                    0.005436
TARGET                            1.000000
Name: TARGET, dtype: float64
```

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

Missing Values

```
missing_vals(temp_app[PCB_df.columns])
```

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

There are a relatively low amount of misssing values here.

Instalments Payments

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

```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NUM_INSTALMENT_VERSION',
       'NUM_INSTALMENT_NUMBER', 'DAYS_INSTALMENT', 'DAYS_ENTRY_PAYMENT',
       'AMT_INSTALMENT', 'AMT_PAYMENT'],
      dtype='object')
```

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

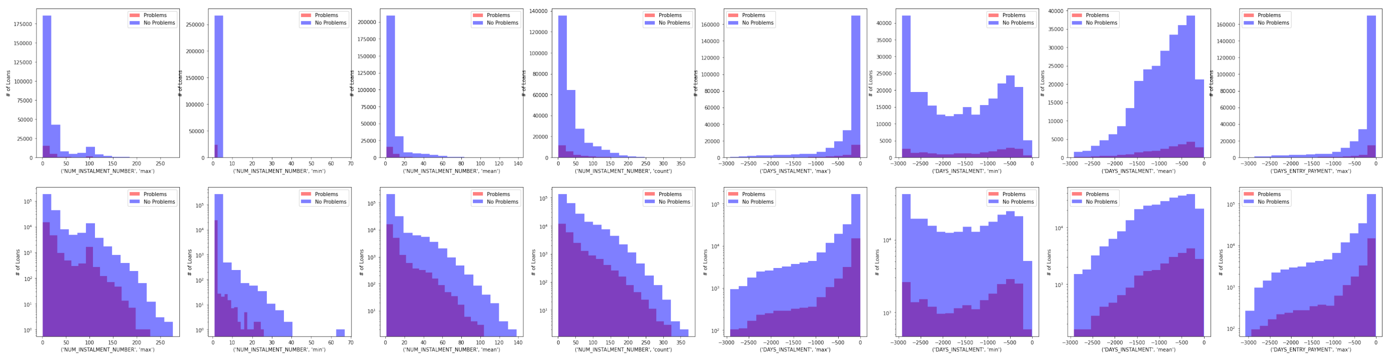
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_PREV              int64  
 1   SK_ID_CURR              int64  
 2   NUM_INSTALMENT_VERSION  float64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

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

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

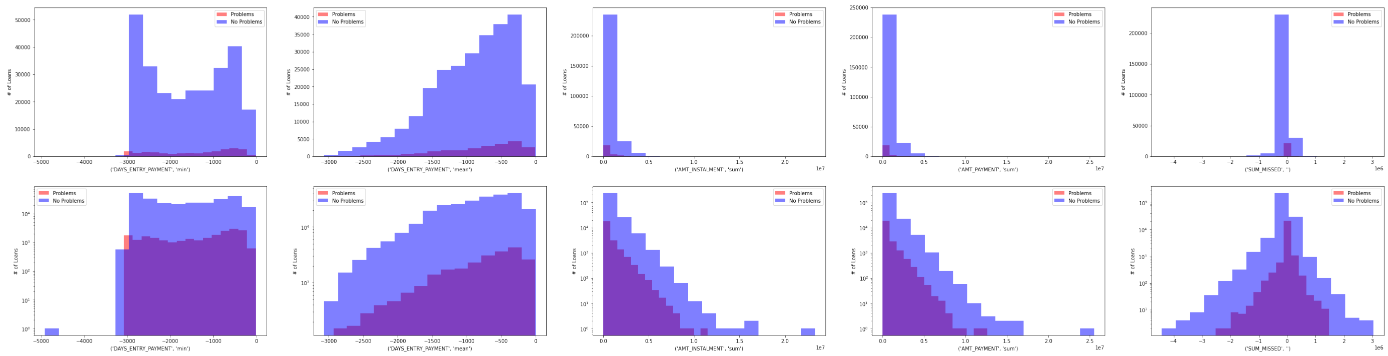
```
temp_app = datasets['application_train'].merge(IP_df, how='left', on='SK_ID_CURR')
```

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

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



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

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

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



These are also not very interesting.

Correlations

```
temp_app[list(IP_df.columns) + ["TARGET"]].corr()["TARGET"]
```

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

Missing Values

```
missing_vals(temp_app[IP_df.columns])
```

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

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

Bureau

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

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

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

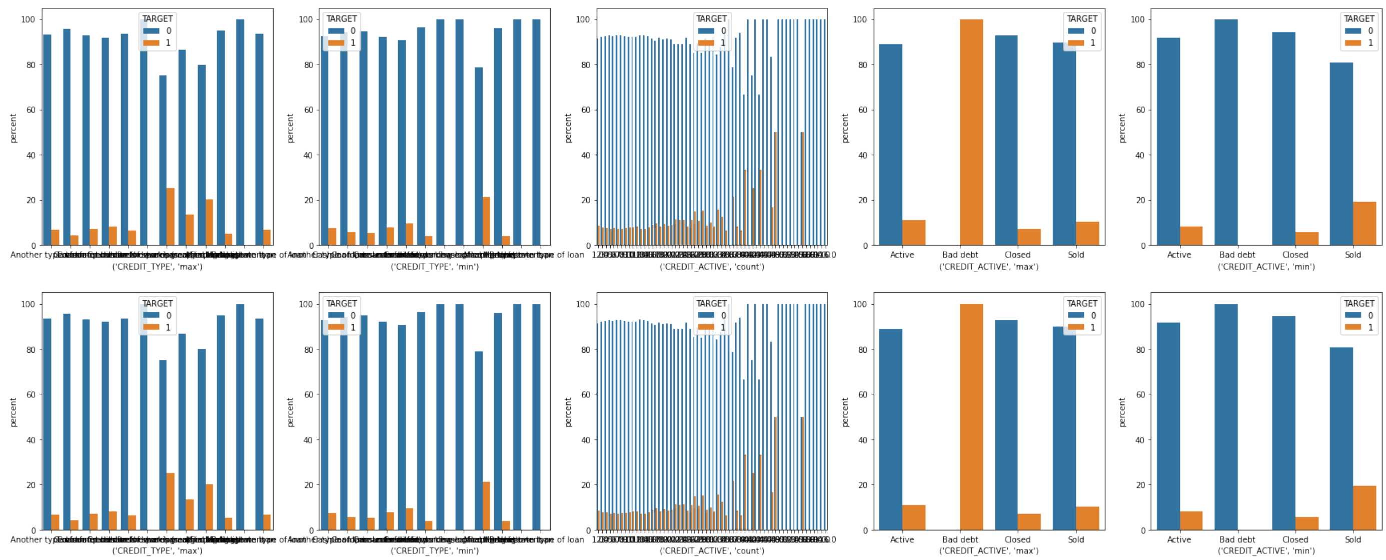
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_PREV              int64
 1   SK_ID_CURR              int64
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER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

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

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

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

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

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

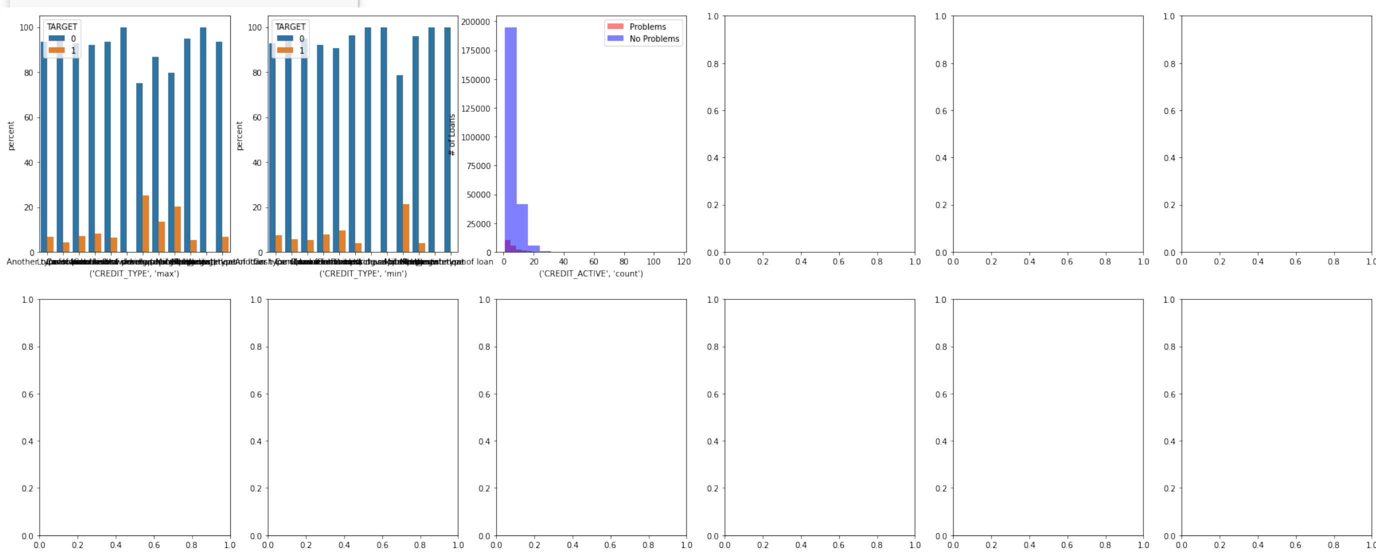


All of the days_credit and credit_type graphs look fairly promising, as parts of the graphs show higher percent chances of problems over no problems compared to others.

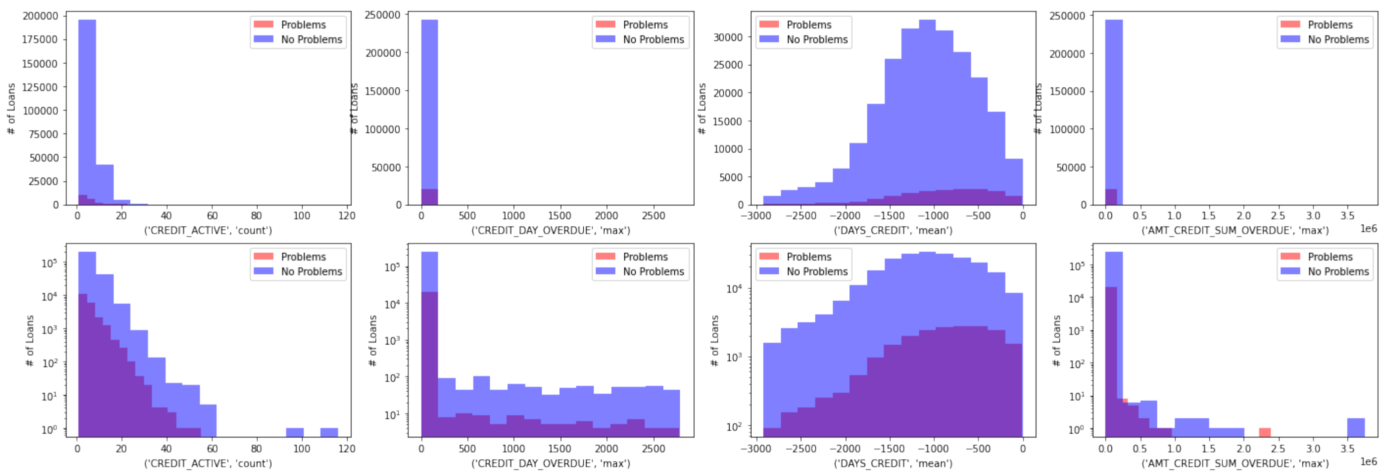
```
fig, axs = plt.subplots(2, 6, figsize=(30,12))
for i in range(6):
  if B_df.columns[5+i][0] == 'CREDIT_ACTIVE' or B_df.columns[i][0] == 'CREDIT_TYPE
    cat_bar(temp_app, B_df.columns[i], axs[0,i])
  else:
    num_hist(temp_app, B_df.columns[i], axs[0,i])

# log graphs
for i in range(6):
  if B_df.columns[5+i][0] == 'CREDIT_ACTIVE' or B_df.columns[i][0] == 'CREDIT_TYPE
    cat_bar(temp_app, B_df.columns[i], axs[1,i])
  else:
    num_hist(temp_app, B_df.columns[i], axs[1,i], True)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-205-23562bb8705f> in <module>()
      1 fig, axs = plt.subplots(2, 6, figsize=(30,12))
      2 for i in range(6):
----> 3   if B_df.columns[5+i][0] == 'CREDIT_ACTIVE' or B_df.columns[i][0] ==
'CREDIT_TYPE':
      4     cat_bar(temp_app, B_df.columns[i], axs[0,i])
      5   else:

/usr/local/lib/python3.7/dist-packages/pandas/core/indexes/multi.py in
__getitem__(self, key)
   1903             retval = []
   1904             for lev, level_codes in zip(self.levels, self.codes):
-> 1905                 if level_codes[key] == -1:
   1906                     retval.append(np.nan)
   1907                 else:

IndexError: index 8 is out of bounds for axis 0 with size 8
```

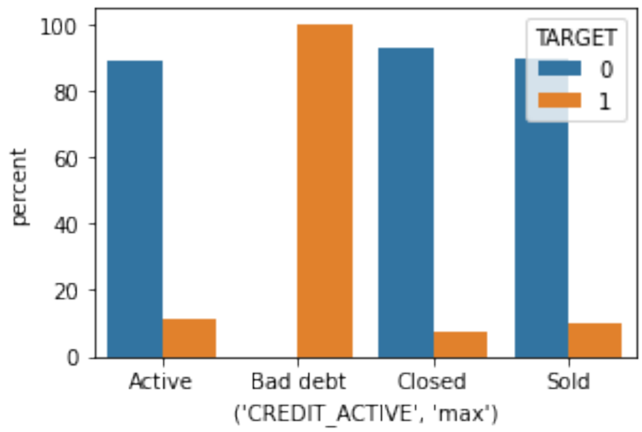
SEARCH STACK OVERFLOW

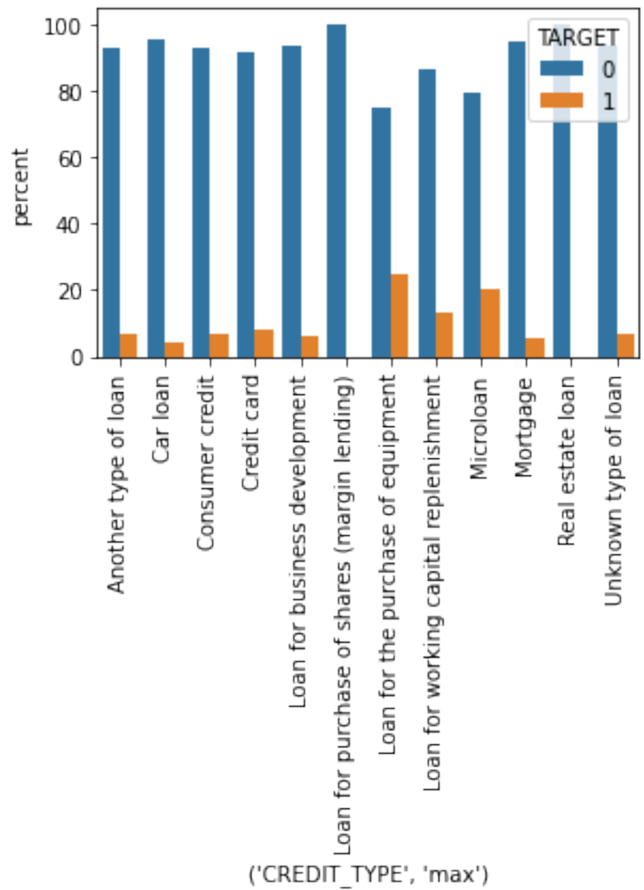


```
fig, axs = plt.subplots(2, 4, figsize=(24, 8))
num_hist(temp_app, ('CREDIT_ACTIVE', 'count'), axs[0,0])
num_hist(temp_app, ('CREDIT_DAY_OVERDUE', 'max'), axs[0,1])
num_hist(temp_app, ('DAYS_CREDIT', 'mean'), axs[0,2])
num_hist(temp_app, ('AMT_CREDIT_SUM_OVERDUE', 'max'), axs[0,3])
# log graphs
num_hist(temp_app,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



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





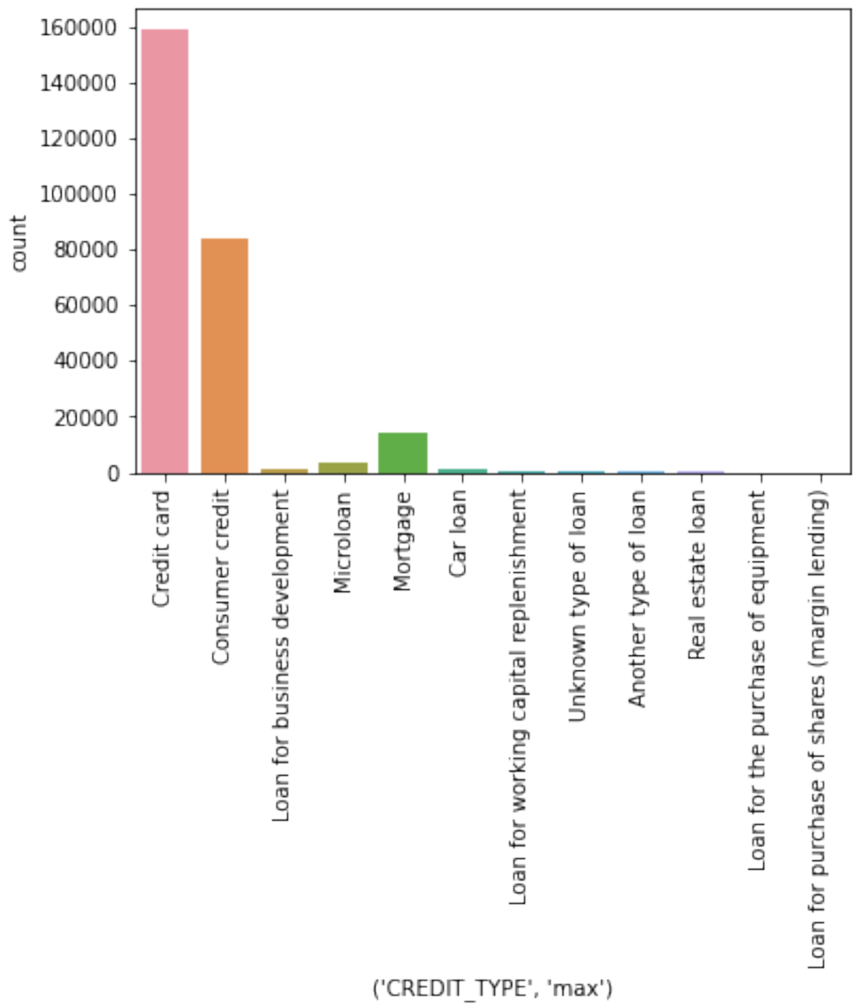
There are some clear trends here. In `CREDIT_ACTIVE`, the 'Closed' portaion had less problems than other groups. Also, everyone in the 'Bad debt' group had issues.

In `CREDIT_TYPE`, client who used their loan to purchase shares or real estate had no problems repaying, while those that used it for equipment, 'working capital replenishment', or a microloan all had significantly more issues repaying.

However, since these difference are so apparent, we should look at the number of samples for the interesting groups.

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

<matplotlib.axes._subplots.AxesSubplot at 0x7f1fcbbebc90>




```
len(temp_app.loc[temp_app[('CREDIT_ACTIVE', 'max')] == 'Bad debt'])
```

1

And, no surprise, but there's only one person in the 'Bad debt' category, rendering it almost useless.

In `CREDIT_TYPE`, it seems that the groups that were interesting to us didn't have a lot of samples, except 'Microloan', which is the 4th most popular category, but by a wide margin. This feature is probably still interesting to us, since the most popular type of loan is for 'Credit card' and that has a noticeably higher difficulty rate than 'Consumer credit' and 'Mortgage', the 2nd and 3rd most popular types, respectively.

Maybe we can group these into 'Credit card', 'Consumer credit', 'Mortgage', 'Other difficult', 'Other', where 'Other difficult' includes Microloans and others where clients had noticeably harder times repaying the loan

Correlations

```
temp_app[list(B_df.columns) + ["TARGET"]].corr()["TARGET"]
```

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

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

Missing Values

```
missing_vals(temp_app[B_df.columns])
```

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

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

Other Datasets

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

```
new_cat_features = [("CREDIT_ACTIVE", "max"), ("NAME_CONTRACT_STATUS", "max"), ("N
new_num_features = [("AMT_APPLICATION", "sum"), ("AMT_APPLICATION", "count"), ("CN
```

Combine Secondaries Into Application

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

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

```
train
```

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

307511 rows × 145 columns

Here we can see the top 20 most correlated features, and we can now see that several of the things added from the secondary datasets have been included.

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

```
Most Positive Correlations:
 FLAG_EMP_PHONE                 0.045982
REG_CITY_NOT_WORK_CITY         0.050994
DAYS_ID_PUBLISH                0.051457
DAYS_LAST_PHONE_CHANGE         0.055218
REGION_RATING_CLIENT           0.058899
REGION_RATING_CLIENT_W_CITY    0.060893
(DAYS_CREDIT, min)             0.075248
DAYS_BIRTH                     0.078239
(AMT_BALANCE, mean)            0.087177
TARGET                         1.000000
Name: TARGET, dtype: float64

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

Secondary + Primary Data Base Pipeline

Now, let's put together a complete pipeline that includes our secondary data as well

The features used come from the features used in the first pipeline as well as any that had either high correlation or looked useful in graph form from our secondary data.

Features Used

```
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",('CREDIT_TYPE', 'min'),('CREDIT_TYPE', 'max'),('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_RATING_CLIENT_W_CITY' ,
('DAYS_CREDIT', 'min')      ,
'DAYS_BIRTH'    ,
('AMT_BALANCE', 'mean')  ,
('DAYS_CREDIT', 'mean') ,
('AMT_CREDIT_LIMIT_ACTUAL', 'count'),
'DAYS_EMPLOYED',
('AMT_BALANCE', 'mean'),
('MONTHS_BALANCE', 'min'),
('AMT_CREDIT_LIMIT_ACTUAL', 'count'),
]
```

Pipeline

```
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

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())
])
```

Fit Second Pipeline

```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_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)
np.random.seed(42)

cv30Splits = ShuffleSplit(n_splits = 30, 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()
train_time = np.round(time() - start, 4)

# 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 w/ All Data", roc,pct(logit_score_train), np.round(pct
                   train_time, test_time, "Untuned 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	178.7383	0.3048	Untuned LogisticRegression

This is a very slight improvement over our first pipeline, and places us 5563 on the public leaderboards.