H2O_automl_lending_club

January 22, 2019

1 Lending Club Analysis Using AutoML

In [1]:
```
import h2o
from h2o.automl import H2OAutoML
import random, os, sys
from datetime import datetime
import pandas as pd
import logging
import csv
import optparse
import time
import json
from distutils.util import strtobool
import psutil
import numpy as np
```

In [2]:
```
target='bad_loan'
min_mem_size=6
run_time=333
```

In [3]:
```
pct_memory=0.5
virtual_memory=psutil.virtual_memory()
min_mem_size=int(round(int(pct_memory*virtual_memory.available)/1073741824,0))
print(min_mem_size)
```

```
3
```

In [4]:
```
# 65535 Highest port no
port_no=random.randint(5555,55555)

#  h2o.init(strict_version_check=False,min_mem_size_GB=min_mem_size,port=port_no) # star
try:
  h2o.init(strict_version_check=False,min_mem_size_GB=min_mem_size,port=port_no) # start
except:
  logging.critical('h2o.init')
  h2o.download_all_logs(dirname=logs_path, filename=logfile)
  h2o.cluster().shutdown()
  sys.exit(2)
```

```
Checking whether there is an H2O instance running at http://localhost:18763... not found.
Attempting to start a local H2O server...
  Java Version: openjdk version "1.8.0_121"; OpenJDK Runtime Environment (Zulu 8.20.0.5-macosx)
  Starting server from /Users/bear/anaconda/lib/python3.6/site-packages/h2o/backend/bin/h2o.jar
  Ice root: /var/folders/lh/42j8mfjx069d1bkc2wlf2pw40000gn/T/tmpa93uia3k
  JVM stdout: /var/folders/lh/42j8mfjx069d1bkc2wlf2pw40000gn/T/tmpa93uia3k/h2o_bear_started_from
  JVM stderr: /var/folders/lh/42j8mfjx069d1bkc2wlf2pw40000gn/T/tmpa93uia3k/h2o_bear_started_from
  Server is running at http://127.0.0.1:18763
Connecting to H2O server at http://127.0.0.1:18763... successful.
```

```
--------------------------  ----------------------------------------
H2O cluster uptime:         01 secs
H2O cluster timezone:       America/New_York
H2O data parsing timezone:  UTC
H2O cluster version:        3.22.1.1
H2O cluster version age:    25 days
H2O cluster name:           H2O_from_python_bear_aiiuox
H2O cluster total nodes:    1
H2O cluster free memory:    3.556 Gb
H2O cluster total cores:    8
H2O cluster allowed cores:  8
H2O cluster status:         accepting new members, healthy
H2O connection url:         http://127.0.0.1:18763
H2O connection proxy:
H2O internal security:      False
H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
Python version:             3.6.5 final
--------------------------  ----------------------------------------
```

1.1 Import data and Manage Data Types

This exploration of H2O will use a version of the Lending Club Loan Data that can be found on Kaggle. This data consists of 15 variables:

	Column Name	Description
1	loan_amnt	Requested loan amount (US dollars)
2	term	Loan term length (months)
3	int_rate	Recommended interest rate
4	emp_length	Employment length (years)
5	home_ownership	Housing status
6	annual_inc	Annual income (US dollars)
7	purpose	Purpose for the loan
8	addr_state	State of residence
9	dti	Debt to income ratio
10	delinq_2yrs	Number of delinquencies in the past 2 years
11	revol_util	Percent of revolving credit line utilized

	Column Name	Description
12	total_acc	Number of active accounts
13	bad_loan	Bad loan indicator
14	longest_credit_length	Age of oldest active account
15	verification_status	Income verification status

```
In [5]: # https://s3-us-west-2.amazonaws.com/h2o-tutorials/data/topics/data/automl/loan.csv
        df = h2o.import_file(path = 'data/loan.csv')
        #train["bad_loan"] = train["bad_loan"].asfactor()
```

```
Parse progress: || 100%
```

```
In [6]: df.describe()
```

```
Rows:163987
Cols:15
```

1.2 Train Models Using H2O's AutoML

```
In [7]: def get_independent_variables(df, targ):
            C = [name for name in df.columns if name != targ]
            # determine column types
            ints, reals, enums = [], [], []
            for key, val in df.types.items():
                if key in C:
                    if val == 'enum':
                        enums.append(key)
                    elif val == 'int':
                        ints.append(key)
                    else:
                        reals.append(key)
            x=ints+enums+reals
            return x
```

```
In [8]: X=get_independent_variables(df, target)
        print(X)
```

```
['loan_amnt', 'emp_length', 'delinq_2yrs', 'total_acc', 'longest_credit_length', 'term', 'home_o
```

```
In [9]: # Set target and predictor variables
        y = target
```

1.3 Regression

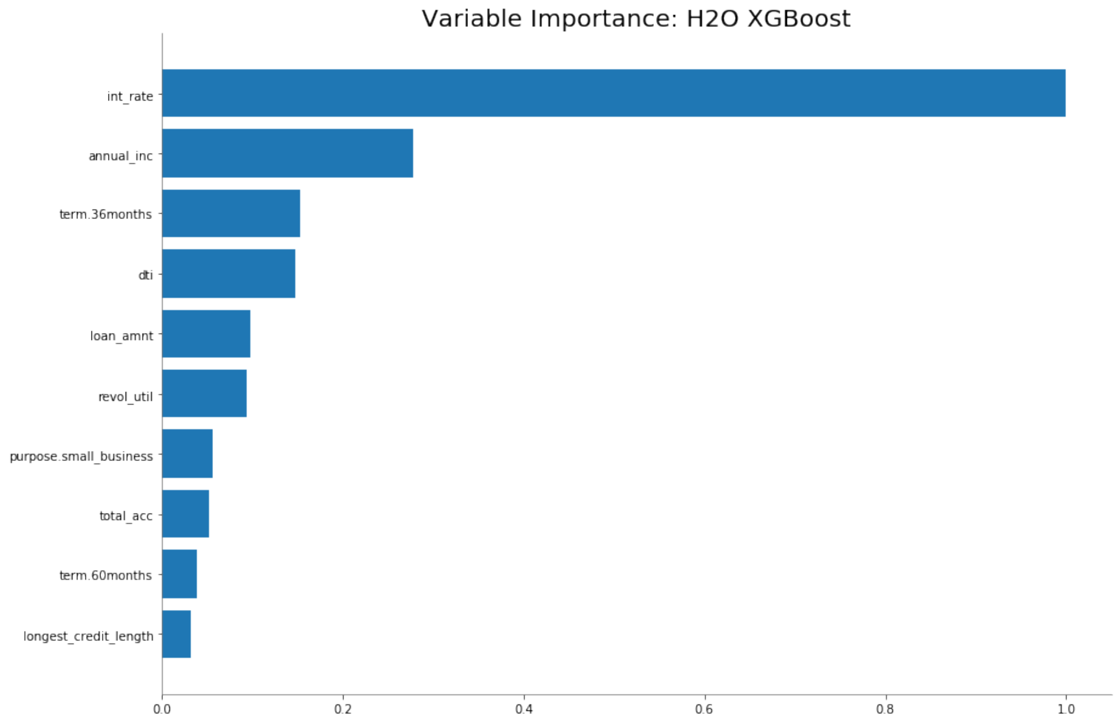
```
In [10]: # Set up AutoML

         aml = H2OAutoML(max_runtime_secs=run_time,exclude_algos = ['DeepLearning'])

In [11]: model_start_time = time.time()

         try:
           aml.train(x=X,y=y,training_frame=df)  # Change training_frame=train
         except Exception as e:
           logging.critical('aml.train')
           h2o.download_all_logs(dirname=logs_path, filename=logfile)
           h2o.cluster().shutdown()
           sys.exit(4)
```

```
AutoML progress: || 100%
```

```
In [12]: meta_data={}
         meta_data['model_execution_time'] = {"regression":(time.time() - model_start_time)}

In [13]: meta_data

Out[13]: {'model_execution_time': {'regression': 344.60554695129395}}

In [14]: print(aml.leaderboard)
```

1.4 Examine the Best Model

```
In [15]: best_model = h2o.get_model(aml.leaderboard[2,'model_id'])

In [16]: best_model.algo

Out[16]: 'xgboost'

In [17]: import matplotlib.pyplot as plt
         %matplotlib inline

         import warnings
         import matplotlib.cbook
         warnings.filterwarnings("ignore", category = matplotlib.cbook.mplDeprecation)

In [18]: best_model.varimp_plot()
```



1.5 Classification

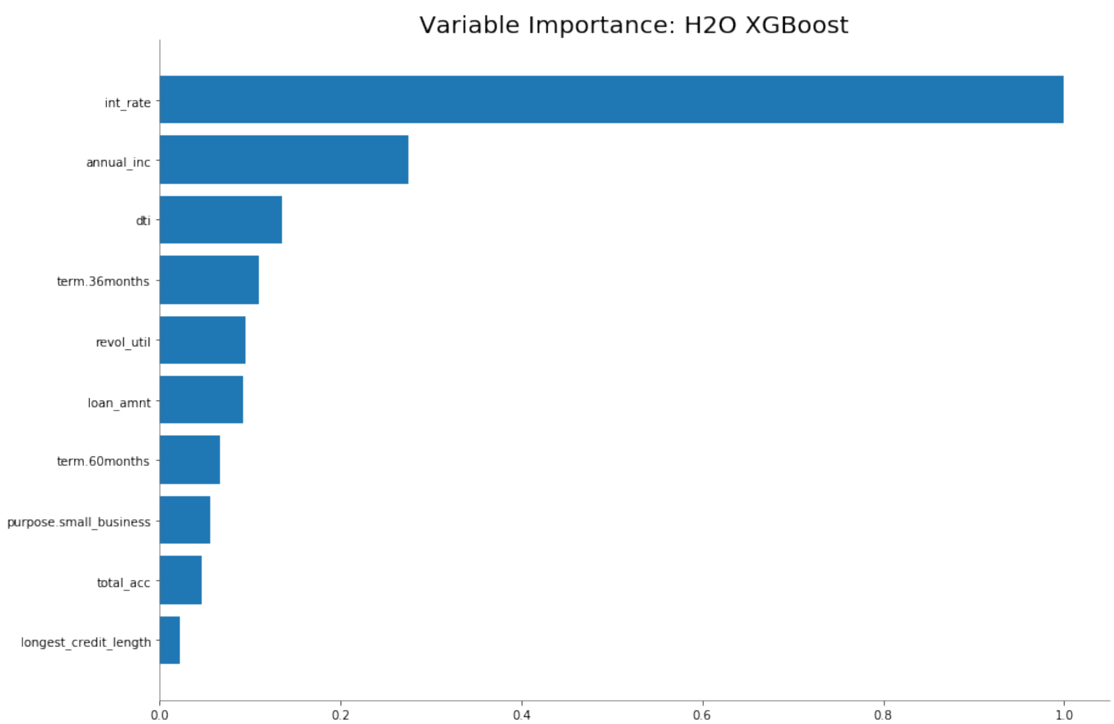
```
In [19]: df[y] = df[y].asfactor()

In [20]: df.describe()
```

```
Rows:163987
Cols:15
```

```
In [21]: # Set up AutoML

         aml = H2OAutoML(max_runtime_secs=run_time)

In [22]: model_start_time = time.time()

         try:
           aml.train(x=X,y=y,training_frame=df)  # Change training_frame=train
         except Exception as e:
           logging.critical('aml.train')
           h2o.download_all_logs(dirname=logs_path, filename=logfile)
           h2o.cluster().shutdown()
           sys.exit(4)
```

```
AutoML progress: || 100%
```

```
In [23]: d=meta_data['model_execution_time']
         d['classification']=(time.time() - model_start_time)
         meta_data['model_execution_time'] = d

In [24]: meta_data

Out[24]: {'model_execution_time': {'classification': 349.32825684547424,
           'regression': 344.60554695129395}}

In [25]: print(aml.leaderboard)

In [26]: best_model = h2o.get_model(aml.leaderboard[2,'model_id'])

In [27]: best_model.algo

Out[27]: 'xgboost'

In [28]: best_model.varimp_plot()
```




```
In [29]: print(best_model.auc(train = True))
```

```
0.7196111306763887
```

```
In [30]: print(best_model.logloss(train = True))
```

```
0.42907008247166123
```

1.6 Perform Feature Engineering

The goal of this section is to improve upon these predictors through a number of feature engineering steps. In particular, we will perform three feature engineering tasks:

1. **Separating Typical from Extreme Loan Amount**
2. **Converting Term to a 0/1 Indicator**
3. **Creating Missing Value Indicator for Employment Length**
4. **Combining Categories in Home Ownership**
5. **Separating Typical from Extreme Annual Income**
6. **Creating Target Encoding for Loan Purpose**
7. **Creating Target Encoding for State of Residence**
8. **Separating Typical from Extreme Debt to Income Ratio**
9. **Separating Typical from Extreme Number of Delinquencies in the Past 2 Years**
10. **Separating Typical from Extreme Revolving Credit Line Utilized**
11. **Separating Typical from Extreme Number of Credit Lines**
12. **Separating Typical from Extreme Longest Credit Length**
13. **Converting Income Verification Status to a 0/1 Indicator**

```
In [31]: x_orig = df.col_names
         x_orig.remove(y)
         x_orig.remove("int_rate")

         x_trans = x_orig.copy()
```

1.6.1 Cross Validation and Target Encoding

Some of the engineered features will use cross-validated mean target encoding of categorical predictors since one hot encodings can lead to overfitting of infrequent categories.

To achieve this goal, we will first create soft partitions using H2OFrame's `kfold_column` function, then calculate summary statistics using H2O's `group_by` function, and finally join these engineered features using H2OFrame's `merge`.

```
In [32]: cv_nfolds = 5
         cv_seed = 2307
```

```
In [33]: train=df
```

```
In [34]: train["cv_fold"] = train.kfold_column(n_folds = cv_nfolds, seed = cv_seed)
```

```
In [35]: train["cv_fold"].table()
```

```
Out[35]:
```

```
In [36]: def logit(p):
             return np.log(p) - np.log(1 - p)
```

```
In [37]: def mean_target(data, x, y = "bad_loan"):
             grouped_data = data[[x, y]].group_by([x])
             stats = grouped_data.count(na = "ignore").mean(na = "ignore")
             return stats.get_frame().as_data_frame()
```

```
In [38]: def mean_target_encoding(data, x, y = "bad_loan", fold_column = "cv_fold", prior_mean =
             """
             Creates target encoding for binary target
             data (H2OFrame) : data set
             x (string) : categorical predictor column name
             y (string) : binary target column name
             fold_column (string) : cross-validation fold column name
             prior_mean (float) : proportion of 1s in the target column
             prior_count (positive number) : weight to give to prior_mean
             """
             grouped_data = data[[x, fold_column, y]].group_by([x, fold_column])
             grouped_data.sum(na = "ignore").count(na = "ignore")
             df = grouped_data.get_frame().as_data_frame()
             df_list = []
             nfold = int(data[fold_column].max()) + 1
             for j in range(0, nfold):
                 te_x = "te_{}".format(x)
                 sum_y = "sum_{}".format(y)
                 oof = df.loc[df[fold_column] != j, [x, sum_y, "nrow"]]
                 stats = oof.groupby([x]).sum()
                 stats[x] = stats.index
                 stats[fold_column] = j
                 p = (stats[sum_y] + (prior_count * prior_mean)) / (stats["nrow"] + prior_count)
                 stats[te_x] = logit(p)
                 df_list.append(stats[[x, fold_column, te_x]])
             return h2o.H2OFrame(pd.concat(df_list))
```

1.6.2 Separating Typical from Extreme Loan Amount

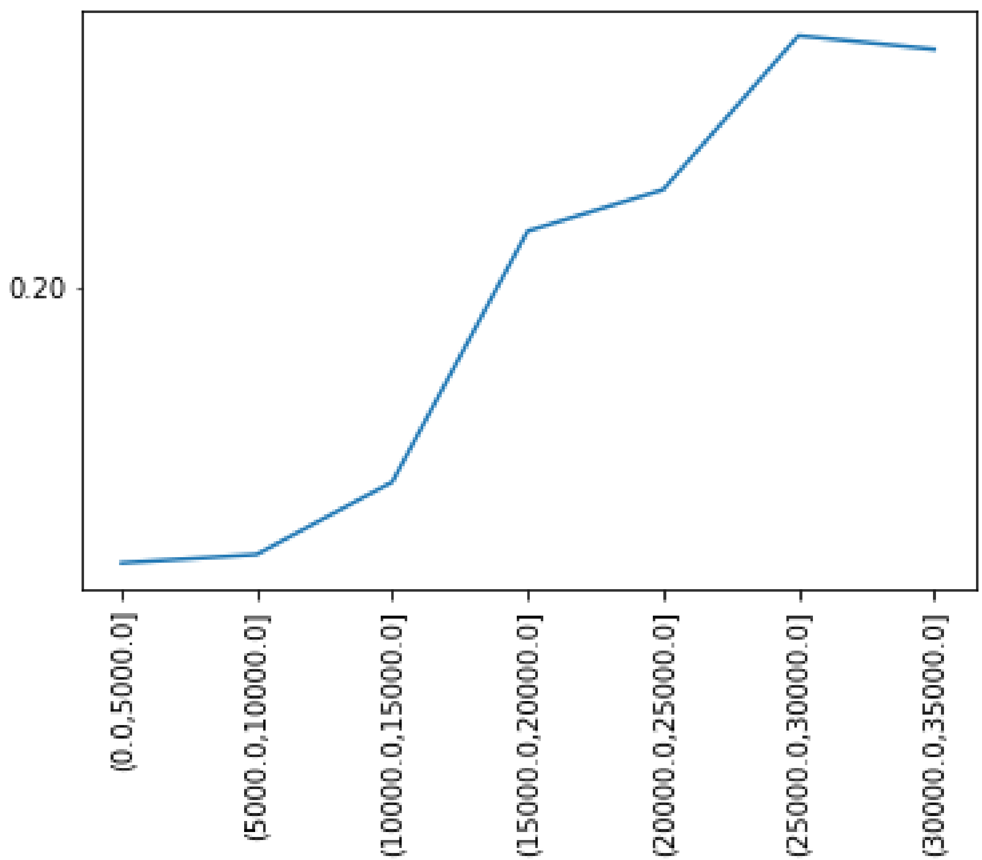
After binning `loan_amt` using H2OFrame's `cut` function and looking at the fraction of bad loans on a logit scale, we see that the chance of a bad loan roughly increases linearly in loan amount from \$5,000 to \$30,000 and is relatively flat below $5,000 and above $30,000. To reflect this finding in the modeling, we will replace the original `loan_amnt` measure with two derived measures:

$$
\begin{aligned}
loan\_amnt\_core &= \max(5000, \min(loan\_amnt, 30000)) && (1)\\
loan\_amnt\_diff &= loan\_amnt - loan\_amnt\_core && (2)
\end{aligned}
$$

```
In [39]: train["loan_amnt"].quantile([0, 0.05, 0.25, 0.5, 0.75, 0.95, 1])
```

```
Out[39]:
```

```
In [40]: breaks = np.linspace(0, 35000, 8).tolist()
         train["loan_amnt_cat"] = train["loan_amnt"].cut(breaks = breaks)

In [41]: df = mean_target(train, "loan_amnt_cat")

In [42]: plt.xticks(rotation = 90)
         plt.yscale("logit")
         plt.plot(df["loan_amnt_cat"], df["mean_bad_loan"])

Out[42]: [<matplotlib.lines.Line2D at 0x116e25a58>]
```




```
In [43]: df

Out[43]:        loan_amnt_cat   nrow  mean_bad_loan
         0        (0.0,5000.0]  25785       0.163234
         1    (5000.0,10000.0]  50384       0.164278
         2  (10000.0,15000.0]  35552       0.173436
         3  (15000.0,20000.0]  24659       0.208281
```

```
4  (20000.0,25000.0]  14447      0.214508
5  (25000.0,30000.0]   6913      0.238825
6  (30000.0,35000.0]   6247      0.236594
```

```
In [44]: x_trans.remove("loan_amnt")
         x_trans.append("loan_amnt_core")
         x_trans.append("loan_amnt_delta")

         train["loan_amnt_core"] = h2o.H2OFrame.ifelse(train["loan_amnt"] <= 5000, 5000, train["
         train["loan_amnt_core"] = h2o.H2OFrame.ifelse(train["loan_amnt_core"] <= 30000, train["

         train["loan_amnt_delta"] = train["loan_amnt"] - train["loan_amnt_core"]
```

1.6.3 Converting Term to a 0/1 Indicator

Given that term of the loans are either 3 or 5 years, we will create a simplifed `term_36month` binary indicator that is 1 when the terms of the loan is for 5 years and 0 for loans with a term of 3 years.

```
In [45]: train["term"].table()
```

```
Out[45]:
```

```
In [46]: x_trans.remove("term")
         x_trans.append("term_60months")

         train["term_60months"] = train["term"] == "60 months"
```

```
In [47]: train["term_60months"].table()
```

```
Out[47]:
```

1.6.4 Creating Missing Value Indicator for Employment Length

The most interesting characteristic about employment length is whether or not it is missing. The divide between those with missing values for employment length to those who have a recorded employment length is 26.3% bad loans to 18.0% bad loans respectively. Interestingly, there doesn't appear to be any differences in bad loans across employment lengths.

```
In [48]: train["emp_length"].summary()
```

```
In [49]: x_trans.append("emp_length_missing")

         train["emp_length_missing"] = train["emp_length"] == None
```

```
In [50]: mean_target_encoding(train, "emp_length_missing")
```

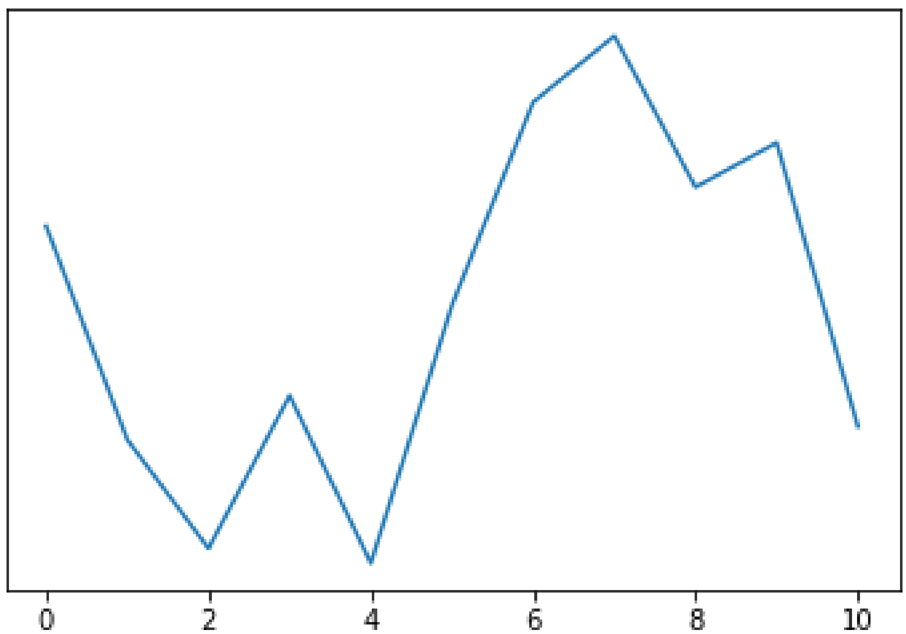
```
Parse progress: || 100%
```

```
Out[50]:
```

```
In [51]: df = mean_target(train, "emp_length")
```

```
In [52]: plt.yscale("logit")
         plt.plot(df["emp_length"], df["mean_bad_loan"])
```

```
Out[52]: [<matplotlib.lines.Line2D at 0x116f278d0>]
```




```
In [53]: df
```

```
Out[53]:     emp_length   nrow  mean_bad_loan
         0          NaN      0       0.262922
         1          0.0  14248       0.184307
         2          1.0  11414       0.177238
         3          2.0  15766       0.173728
         4          3.0  13611       0.178679
         5          4.0  11024       0.173258
         6          5.0  12347       0.181664
         7          6.0  10000       0.188500
         8          7.0   9079       0.190770
         9          8.0   7424       0.185614
         10         9.0   6087       0.187120
         11        10.0  47183       0.177670
```

1.6.5 Combining Categories in Home Ownership

Although there are 6 recorded categories within home ownership, only three had over 200 observations: OWN, MORTGAGE, and RENT. The remaining three are so infrequent we will combine them {ANY, NONE, OTHER} with RENT to form an enlarged OTHER category. This new `home_ownership_3cat` variable will have values in {MORTGAGE, OTHER, OWN}.

```
In [54]: mean_target(train, "home_ownership")
```

```
Out[54]:   home_ownership   nrow  mean_bad_loan
         0            ANY      1       0.000000
         1       MORTGAGE  79714       0.164137
         2           NONE     30       0.233333
         3          OTHER    156       0.224359
         4            OWN  13560       0.188348
         5           RENT  70526       0.203273
```

```
In [55]: lvls = ["OTHER", "MORTGAGE", "OTHER", "OTHER", "OWN", "OTHER"]
         train["home_ownership_3cat"] = train["home_ownership"].set_levels(lvls).ascharacter().a
```

```
In [56]: train[["home_ownership", "home_ownership_3cat"]].table()
```

```
Out[56]:
```

```
In [57]: mean_target(train, "home_ownership_3cat")
```

```
Out[57]:   home_ownership_3cat   nrow  mean_bad_loan
         0            MORTGAGE  79714       0.164137
         1               OTHER  70713       0.203329
         2                 OWN  13560       0.188348
```

```
In [58]: x_trans.remove("home_ownership")
         x_trans.append("home_ownership_3cat")
```

1.6.6 Separating Typical from Extreme Annual Income

Looking at the occurance of bad loans on a logit scale reveal that the chance of a bad loan roughly decreases linearly in annual income from \$10,000 to \$105,000 and is relatively flat above \$105,000. To reflect this finding in the modeling, we will replace the original `annual_inc` measure with two derived measures:

$$annual\_inc\_core = \max(10000, \min(annual\_inc, 105000)) \tag{3}$$
$$annual\_inc\_diff = annual\_inc - annual\_inc\_core \tag{4}$$

```
In [59]: train["annual_inc"].quantile([0, 0.05, 0.25, 0.5, 0.75, 0.95, 1])
```

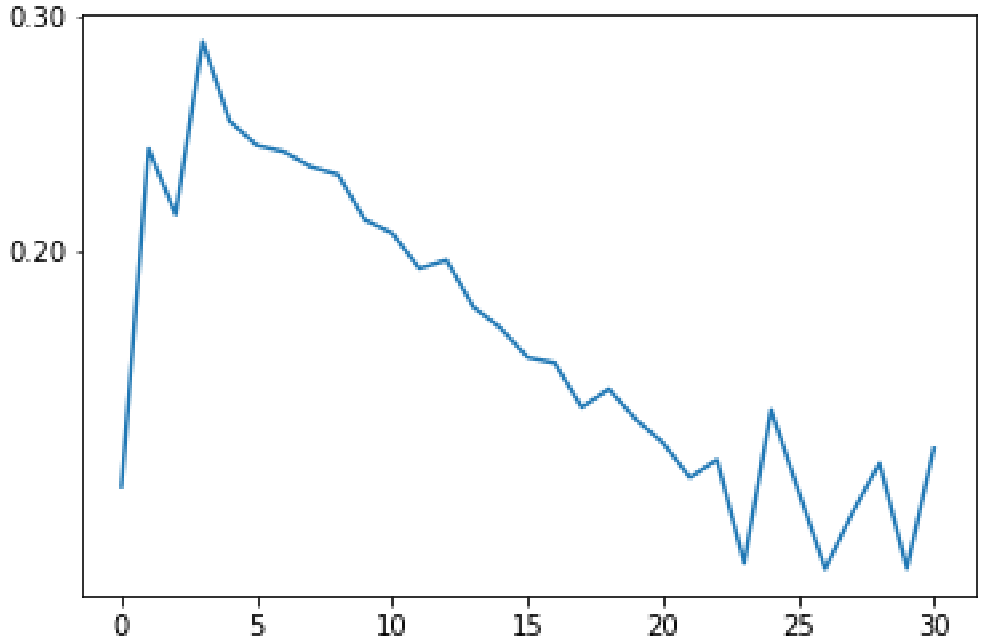
```
Out[59]:
```

```
In [60]: breaks = np.linspace(0, 150000, 31).tolist()
         train["annual_inc_cat"] = train["annual_inc"].cut(breaks = breaks)
```

```
In [61]: df = mean_target(train, "annual_inc_cat")
```

In [62]:
```
plt.yscale("logit")
plt.plot(df["annual_inc_cat"].index, df["mean_bad_loan"])
```

Out[62]: [<matplotlib.lines.Line2D at 0x116f78a20>]



In [63]:
```
df[0:6]
```

Out[63]:

	annual_inc_cat	nrow	mean_bad_loan
0	NaN	0	0.126695
1	(0.0,5000.0]	25	0.240000
2	(5000.0,10000.0]	192	0.213542
3	(10000.0,15000.0]	868	0.288018
4	(15000.0,20000.0]	1847	0.251760
5	(20000.0,25000.0]	3975	0.241509

In [64]:
```
df[20:31]
```

Out[64]:

	annual_inc_cat	nrow	mean_bad_loan
20	(95000.0,100000.0]	5473	0.138315
21	(100000.0,105000.0]	2860	0.129021
22	(105000.0,110000.0]	3275	0.133740
23	(110000.0,115000.0]	1908	0.108491
24	(115000.0,120000.0]	3009	0.147557
25	(120000.0,125000.0]	1741	0.125790
26	(125000.0,130000.0]	1642	0.107186
27	(130000.0,135000.0]	974	0.120123

```
28  (135000.0,140000.0]  1235        0.132794
29  (140000.0,145000.0]   746        0.107239
30  (145000.0,150000.0]  1492        0.136729
```

```
In [65]: x_trans.remove("annual_inc")
         x_trans.append("annual_inc_core")
         x_trans.append("annual_inc_delta")

         train["annual_inc_core"] = h2o.H2OFrame.ifelse(train["annual_inc"] <= 10000, 10000, tra
         train["annual_inc_core"] = h2o.H2OFrame.ifelse(train["annual_inc_core"] <= 105000,
                                                         train["annual_inc_core"], 105000)

         train["annual_inc_delta"] = train["annual_inc"] - train["annual_inc_core"]
```

1.6.7 Creating Target Encoding for Loan Purpose

Given that there is a high concentration of loans for debt consolidation (56.87%), a sizable number for credit card (18.78%), and the remaining 24.35% loans are spread amongst 12 other purposes, we will use mean target encoding to avoid overfitting of the later group.

```
In [66]: tbl = train["purpose"].table().as_data_frame()
         tbl["Percent"] = np.round((100 * tbl["Count"]/train.nrows), 2)
         tbl = tbl.sort_values(by = "Count", ascending = 0)
         tbl = tbl.reset_index(drop = True)
         print(tbl)
```

```
               purpose  Count  Percent
0   debt_consolidation  93261    56.87
1          credit_card  30792    18.78
2                other  10492     6.40
3     home_improvement   9872     6.02
4       major_purchase   4686     2.86
5       small_business   3841     2.34
6                  car   2842     1.73
7              medical   2029     1.24
8              wedding   1751     1.07
9               moving   1464     0.89
10               house   1245     0.76
11            vacation   1096     0.67
12         educational    418     0.25
13    renewable_energy    198     0.12
```

```
In [67]: df = mean_target(train, "purpose")
```

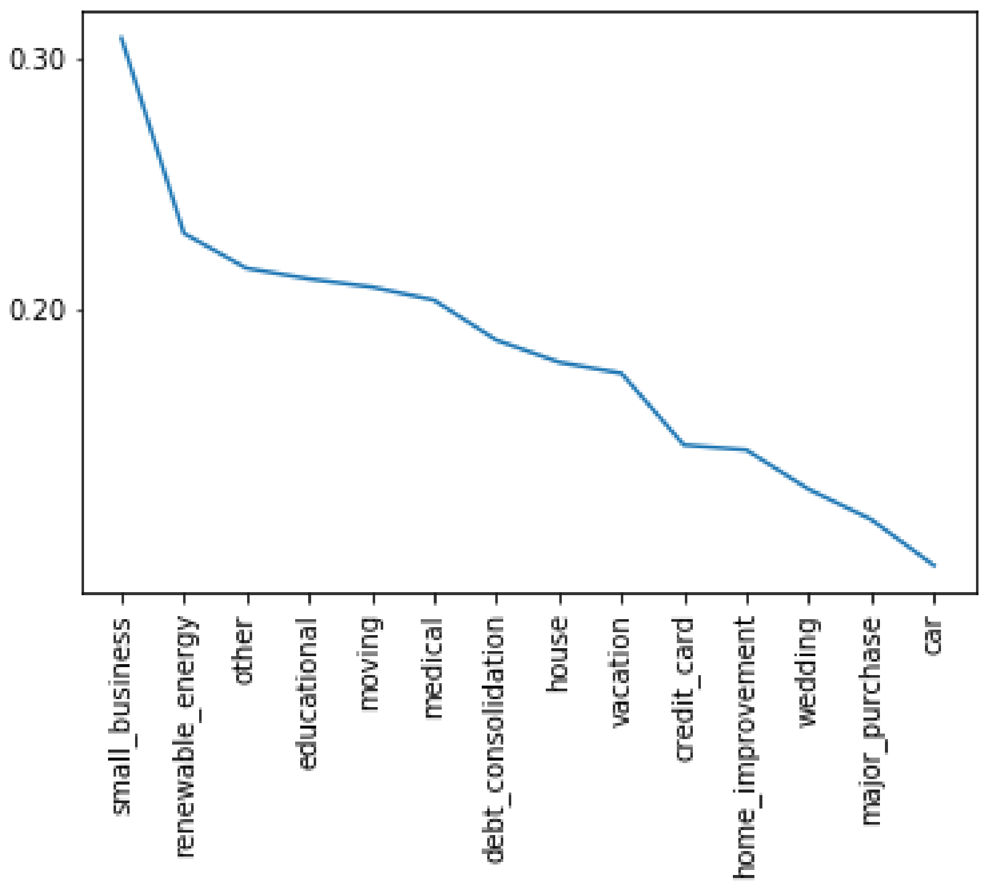
```
In [68]: df = df.sort_values(by = "mean_bad_loan", ascending = 0)
         df = df.reset_index(drop = True)
         df
```

Out[68]:

	purpose	nrow	mean_bad_loan
0	small_business	3841	0.309034
1	renewable_energy	198	0.227273
2	other	10492	0.214354
3	educational	418	0.210526
4	moving	1464	0.207650
5	medical	2029	0.203056
6	debt_consolidation	93261	0.189479
7	house	1245	0.182329
8	vacation	1096	0.178832
9	credit_card	30792	0.157281
10	home_improvement	9872	0.155895
11	wedding	1751	0.145060
12	major_purchase	4686	0.137217
13	car	2842	0.125968

In [69]:
```
plt.xticks(rotation = 90)
plt.yscale("logit")
plt.plot(df["purpose"], df["mean_bad_loan"])
```

Out[69]: [<matplotlib.lines.Line2D at 0x1170d27b8>]



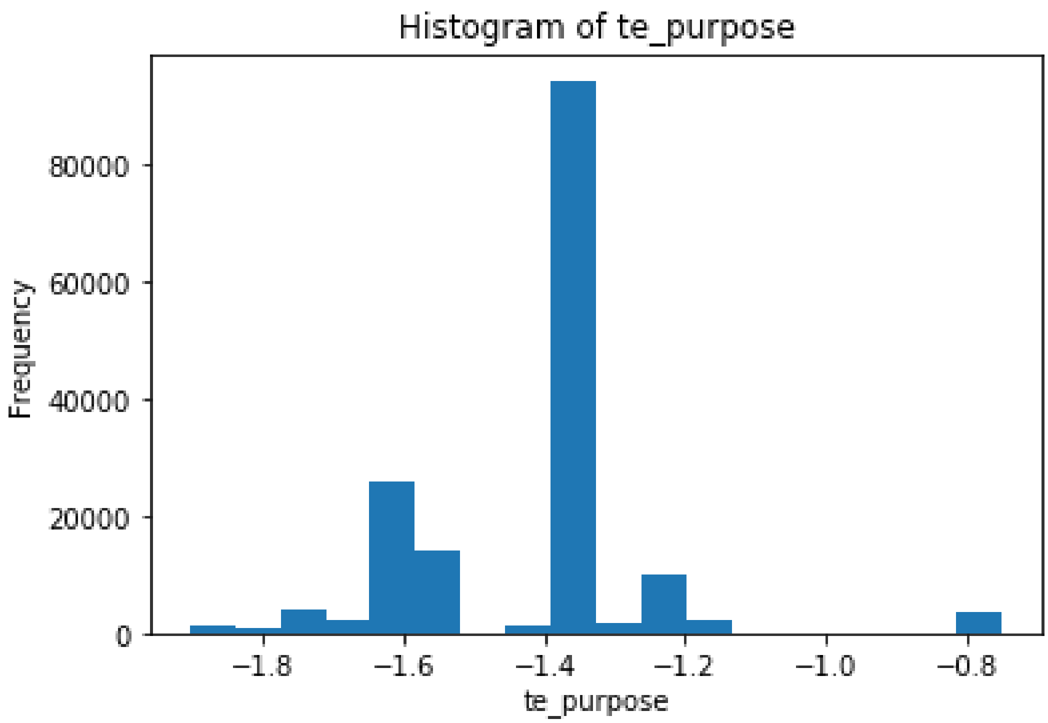
```
In [70]: te_purpose = mean_target_encoding(train, "purpose")
```

```
Parse progress: || 100%
```

```
In [71]: train = train.merge(te_purpose, all_x = True)
```

```
In [72]: x_trans.remove("purpose")
         x_trans.append("te_purpose")
```

```
In [73]: train["te_purpose"].hist()
```



1.6.8 Creating Target Encoding for State of Residence

We will also use a mean target encoding for state of residence for a reason similar to that for purpose.

```
In [74]: tbl = train["addr_state"].table().as_data_frame()
         tbl["Percent"] = np.round((100 * tbl["Count"]/train.nrows), 2)
         tbl = tbl.sort_values(by = "Count", ascending = 0)
         tbl = tbl.reset_index(drop = True)
         print(tbl[0:5])
```

```
  addr_state  Count  Percent
0         CA  28702    17.50
1         NY  14285     8.71
2         TX  12128     7.40
3         FL  11396     6.95
4         NJ   6457     3.94
```

```
In [75]: df = mean_target(train, "addr_state")
```

```
In [76]: df = df.sort_values(by = "mean_bad_loan", ascending = 0)
         df = df.reset_index(drop = True)
         print(df[0:5])
```

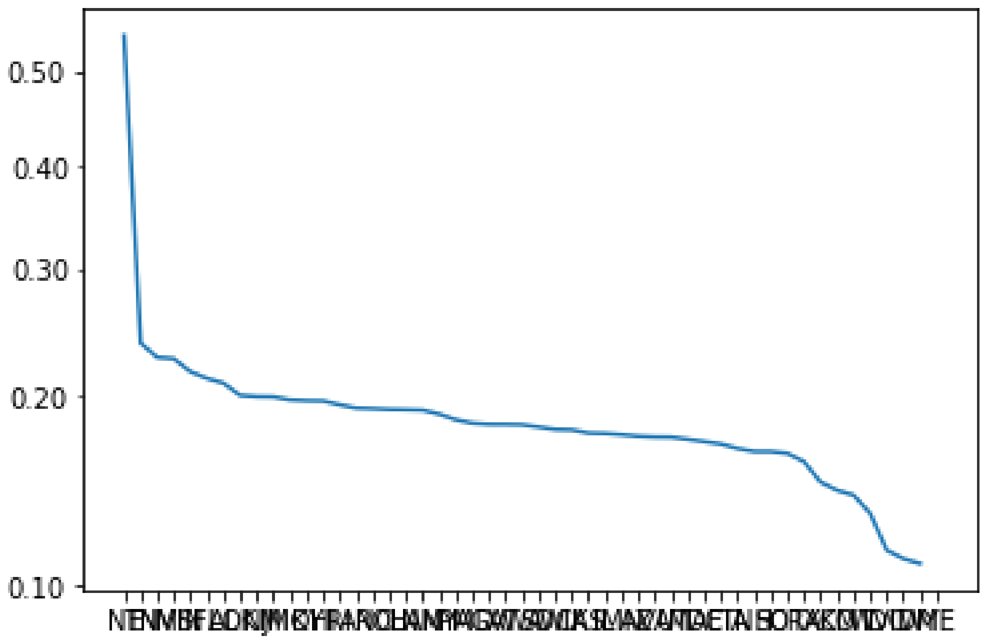
```
  addr_state  nrow  mean_bad_loan
0         NE    13       0.538462
1         TN  1327       0.238885
2         NV  2387       0.227901
3         MS   163       0.226994
4         IN  1463       0.217362
```

```
In [77]: print(df[45:50])
```

```
   addr_state  nrow  mean_bad_loan
45         WV   714       0.131653
46         DC   584       0.114726
47         ID     9       0.111111
48         WY   376       0.109043
49         ME     3       0.000000
```

```
In [78]: plt.yscale("logit")
         plt.plot(df["addr_state"], df["mean_bad_loan"])
```

```
Out[78]: [<matplotlib.lines.Line2D at 0x1172aca58>]
```




```
In [79]: te_addr_state = mean_target_encoding(train, "addr_state", prior_count = 30)
```

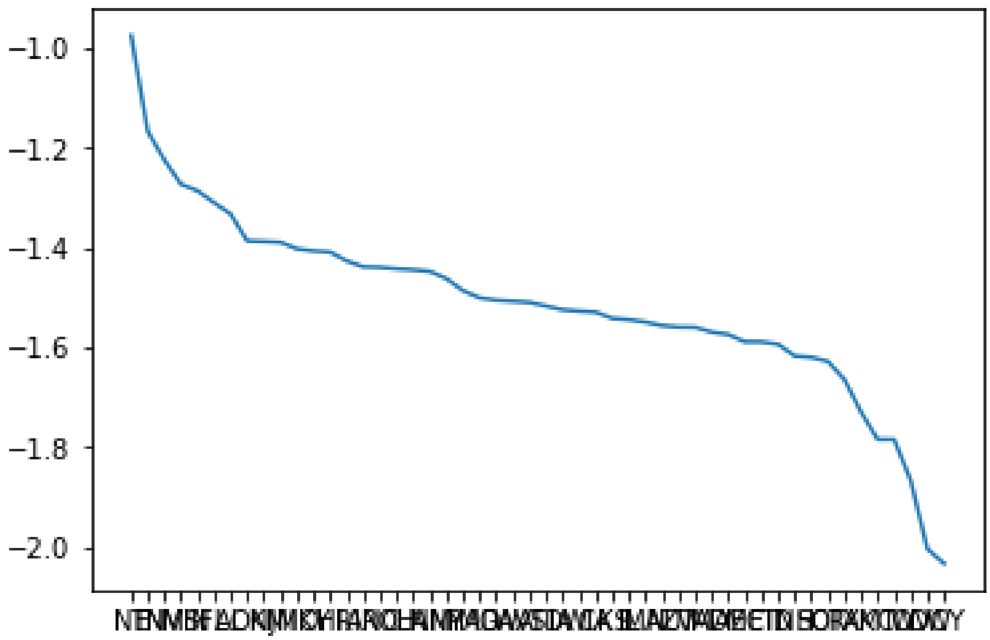
```
Parse progress: || 100%
```

```
In [80]: grouped_data = te_addr_state[["addr_state", "te_addr_state"]].group_by(["addr_state"])
         df = grouped_data.count(na = "ignore").mean(na = "ignore").get_frame().as_data_frame()
```

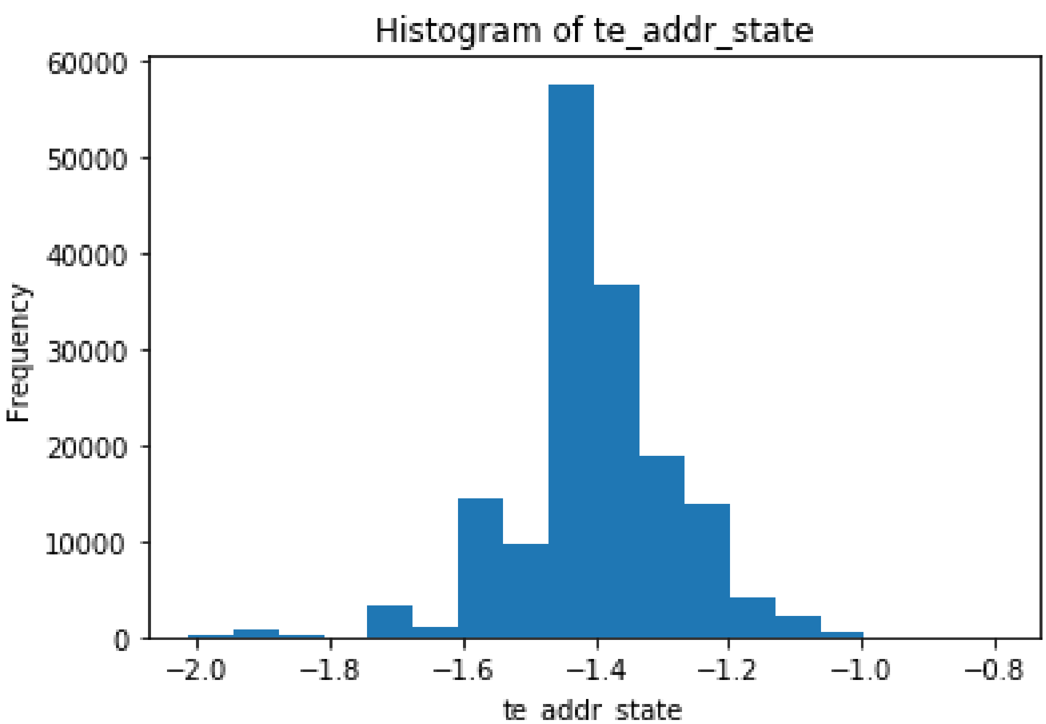
```
In [81]: df = df.sort_values(by = "mean_te_addr_state", ascending = 0)
         df = df.reset_index(drop = True)
```

```
In [82]: plt.plot(df["addr_state"], df["mean_te_addr_state"])
```

```
Out[82]: [<matplotlib.lines.Line2D at 0x11737fc88>]
```




```
In [83]: train = train.merge(te_addr_state, all_x = True)

In [84]: x_trans.remove("addr_state")
         x_trans.append("te_addr_state")

In [85]: train["te_addr_state"].hist()
```

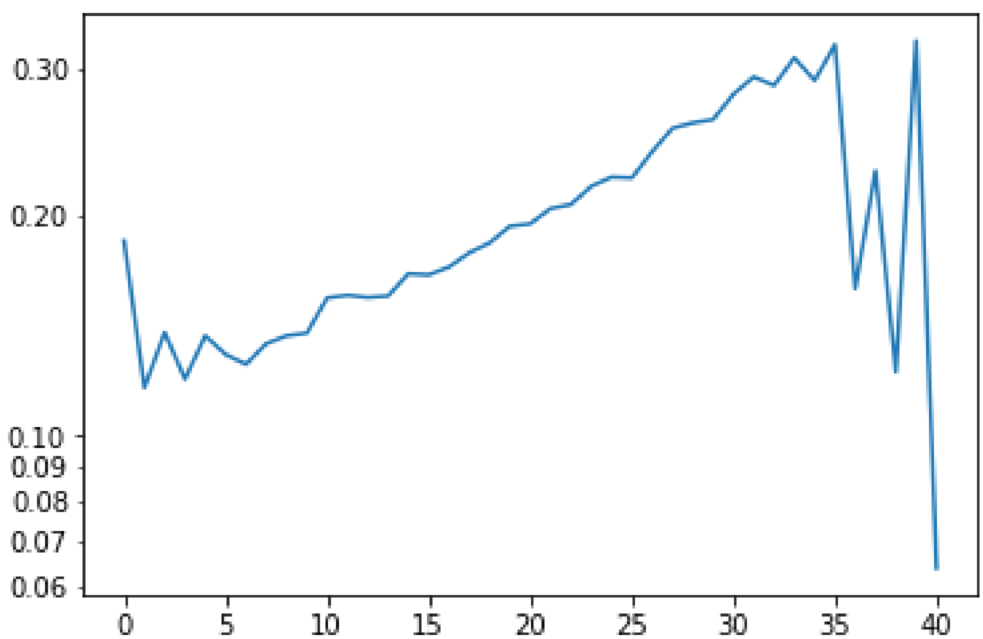


1.6.9 Separating Typical from Extreme Debt to Income Ratio

Looking at the occurance of bad loans on a logit scale reveal that the chance of a bad loan roughly increases linearly in debt-to-income from 5% to 30% and is highly volatile outside of that range due to small numbers of observations. To reflect this finding in the modeling, we will replace the original `dti` measure with two derived measures:

$$dti\_core = \max(5, \min(dti, 30)) \tag{5}$$
$$dti\_diff = dti - dti\_core \tag{6}$$

```
In [86]: train["dti"].quantile([0, 0.05, 0.25, 0.5, 0.75, 0.95, 1])

Out[86]:

In [87]: breaks = np.linspace(0, 40, 41).tolist()
         train["dti_cat"] = train["dti"].cut(breaks = breaks)

In [88]: df = mean_target(train, "dti_cat")

In [89]: plt.yscale("logit")
         plt.plot(df["dti_cat"].index, df["mean_bad_loan"])

Out[89]: [<matplotlib.lines.Line2D at 0x1174968d0>]
```



```
In [90]: df[30:41]
```

```
Out[90]:        dti_cat  nrow  mean_bad_loan
         30  (29.0,30.0]  2329       0.280378
         31  (30.0,31.0]  1496       0.293449
         32  (31.0,32.0]  1282       0.287051
         33  (32.0,33.0]  1051       0.308278
         34  (33.0,34.0]  1025       0.290732
         35  (34.0,35.0]   750       0.318667
         36  (35.0,36.0]    75       0.160000
         37  (36.0,37.0]    66       0.227273
         38  (37.0,38.0]    65       0.123077
         39  (38.0,39.0]    59       0.322034
         40  (39.0,40.0]    47       0.063830
```

```
In [91]: x_trans.remove("dti")
         x_trans.append("dti_core")
         x_trans.append("dti_delta")

         train["dti_core"] = h2o.H2OFrame.ifelse(train["dti"] <= 5, 5, train["dti"])
         train["dti_core"] = h2o.H2OFrame.ifelse(train["dti_core"] <= 30, train["dti_core"], 30)

         train["dti_delta"] = train["dti"] - train["dti_core"]
```

1.6.10 Separating Typical from Extreme Number of Delinquencies in the Past 2 Years

The chance of a bad loan seems to max out at 3 delinquent payments in the past two years. To reflect this finding in the modeling, we will replace the original `delinq_2yrs` measure with two derived measures:

$$delinq\_2yrs\_core = \min(delinq\_2yrs, 3) \tag{7}$$

$$delinq\_2yrs\_diff = delinq\_2yrs - delinq\_2yrs\_core \tag{8}$$

```
In [92]: train["delinq_2yrs"].quantile([0, 0.05, 0.25, 0.5, 0.75, 0.95, 1])
```

```
Out[92]:
```

```
In [93]: breaks = np.linspace(0, 5, 6).tolist()
         train["delinq_2yrs_cat"] = train["delinq_2yrs"].cut(breaks = breaks)
```

```
In [94]: mean_target(train, "delinq_2yrs_cat")
```

```
Out[94]:   delinq_2yrs_cat   nrow  mean_bad_loan
         0             NaN      0       0.181037
         1       (0.0,1.0]  17158       0.189299
         2       (1.0,2.0]   4635       0.201510
         3       (2.0,3.0]   1488       0.221774
         4       (3.0,4.0]    579       0.215889
         5       (4.0,5.0]    310       0.216129
```

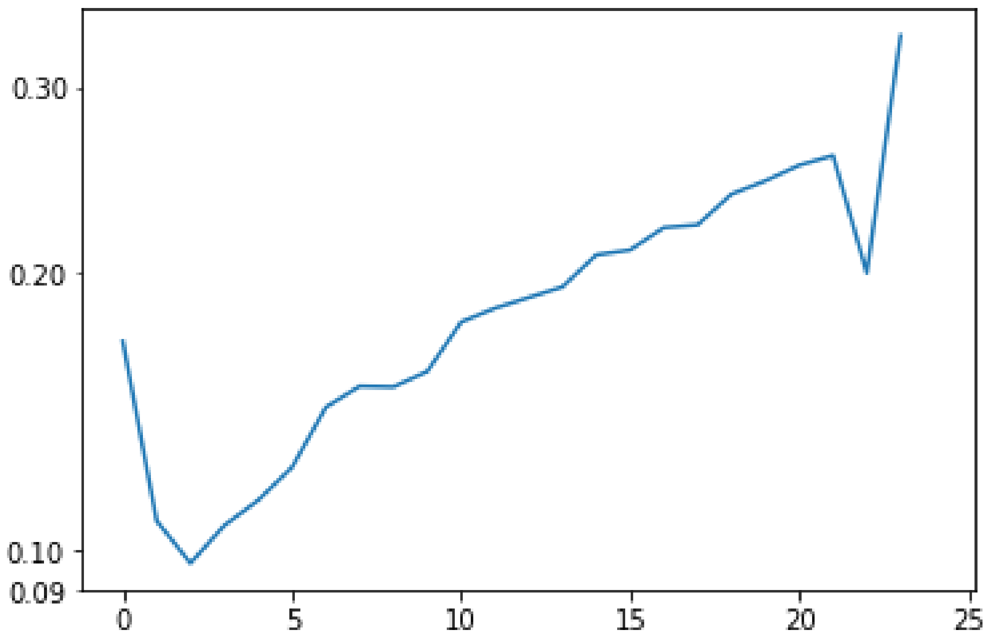
```
In [95]: x_trans.remove("delinq_2yrs")
         x_trans.append("delinq_2yrs_core")
         x_trans.append("delinq_2yrs_delta")

         train["delinq_2yrs_core"] = h2o.H2OFrame.ifelse(train["delinq_2yrs"] <= 3, train["delin

         train["delinq_2yrs_delta"] = train["delinq_2yrs"] - train["delinq_2yrs_core"]
```

1.6.11 Separating Typical from Extreme Revolving Credit Line Utilized

The relationship between credit line utilized is somewhat interesting. There appears to be a higher rate for a bad loan when 0% of the credit lines are utilized, then it drops down slightly and roughly increases linearly in credit line utilized up to 100%. To reflect this finding in the modeling, we will replace the original `revol_util` measure with three derived measures:

$$
\begin{aligned}
revol\_util\_0 &= I(revol\_util == 0) && (9)\\
revol\_util\_core &= \max(5, \min(revol\_util, 30)) && (10)\\
revol\_util\_diff &= revol\_util - revol\_util\_core && (11)
\end{aligned}
$$

```
In [96]: train["revol_util"].quantile([0, 0.05, 0.25, 0.5, 0.75, 0.95, 1])

Out[96]:

In [97]: breaks = np.linspace(0, 120, 25).tolist()
         train["revol_util_cat"] = train["revol_util"].cut(breaks = breaks)

In [98]: df = mean_target(train, "revol_util_cat")

In [99]: plt.yscale("logit")
         plt.plot(df["revol_util_cat"].index, df["mean_bad_loan"])

Out[99]: [<matplotlib.lines.Line2D at 0x116b71518>]
```



In [100]: df[20:25]

Out[100]:

	revol_util_cat	nrow	mean_bad_loan
20	(95.0,100.0]	4316	0.255097
21	(100.0,105.0]	238	0.260504
22	(105.0,110.0]	35	0.200000
23	(110.0,115.0]	9	0.333333
24	(115.0,120.0]	3	0.000000

In [101]:
```
x_trans.remove("revol_util")
x_trans.append("revol_util_0")
x_trans.append("revol_util_core")
x_trans.append("revol_util_delta")

train["revol_util_0"] = train["revol_util"] == 0

train["revol_util_core"] = h2o.H2OFrame.ifelse(train["revol_util"] <= 100, train["revo

train["revol_util_delta"] = train["revol_util"] - train["revol_util_core"]
```

1.6.12 Separating Typical from Extreme Number of Credit Lines

Looking at the occurance of bad loans on a logit scale reveal that the chance of a bad loan roughly decreases linearly in number of lines of credit up to about 50. To reflect this finding in the modeling, we will replace the original `total_acc` measure with two derived measures:

$$
\begin{aligned}
total\_acc\_core &= \min(total\_acc, 50) \quad (12)\\
total\_acc\_diff &= total\_acc - total\_acc\_core \quad (13)
\end{aligned}
$$

In [102]:
```
train["total_acc"].quantile([0, 0.05, 0.25, 0.5, 0.75, 0.95, 1])
```

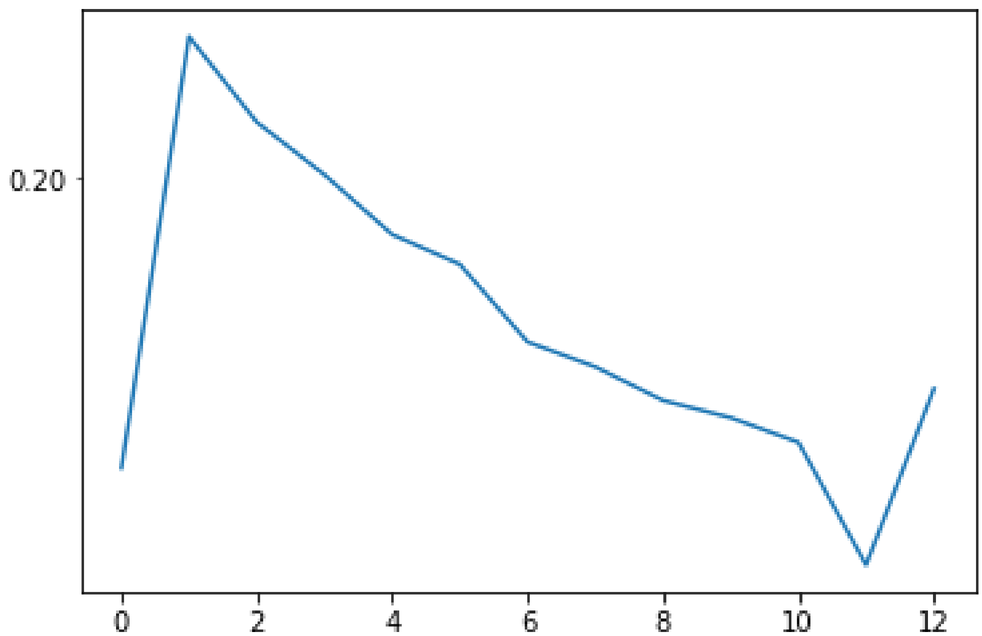
Out[102]:

In [103]:
```
breaks = np.linspace(0, 60, 13).tolist()
train["total_acc_cat"] = train["total_acc"].cut(breaks = breaks)
```

In [104]:
```
df = mean_target(train, "total_acc_cat")
```

In [105]:
```
plt.yscale("logit")
plt.plot(df["total_acc_cat"].index, df["mean_bad_loan"])
```

Out[105]:
```
[<matplotlib.lines.Line2D at 0x1164b8cf8>]
```



In [106]:
```
(train["total_acc"] == None).table()
```

Out[106]:

In [107]:
```
df[0:3]
```

Out[107]:

	total_acc_cat	nrow	mean_bad_loan
0	NaN	0	0.153788
1	(0.0,5.0]	2309	0.226072
2	(5.0,10.0]	12717	0.210034

```
In [108]: df[8:13]
```

```
Out[108]:    total_acc_cat   nrow  mean_bad_loan
         8    (35.0,40.0]  11251       0.163719
         9    (40.0,45.0]   7174       0.161137
         10   (45.0,50.0]   4203       0.157507
         11   (50.0,55.0]   2350       0.140426
         12   (55.0,60.0]   1287       0.165501
```

```
In [109]: x_trans.remove("total_acc")
          x_trans.append("total_acc_core")
          x_trans.append("total_acc_delta")

          train["total_acc_core"] = h2o.H2OFrame.ifelse(train["total_acc"] <= 50, train["total_a

          train["total_acc_delta"] = train["total_acc"] - train["total_acc_core"]
```

1.6.13 Separating Typical from Extreme Longest Credit Length

Looking at the occurance of bad loans on a logit scale reveal that the chance of a bad loan roughly decreases linearly in longest credit length from 3 to 20 years and is highly volatile outside of that range due to small numbers of observations. To reflect this finding in the modeling, we will replace the original `longest_credit_length` measure with two derived measures:

$$longest\_credit\_length\_core = \max(3, \min(longest\_credit\_length, 20)) \quad (14)$$
$$longest\_credit\_length\_diff = longest\_credit\_length - longest\_credit\_length\_core \quad (15)$$

```
In [110]: train["longest_credit_length"].quantile([0, 0.05, 0.25, 0.5, 0.75, 0.95, 1])
```

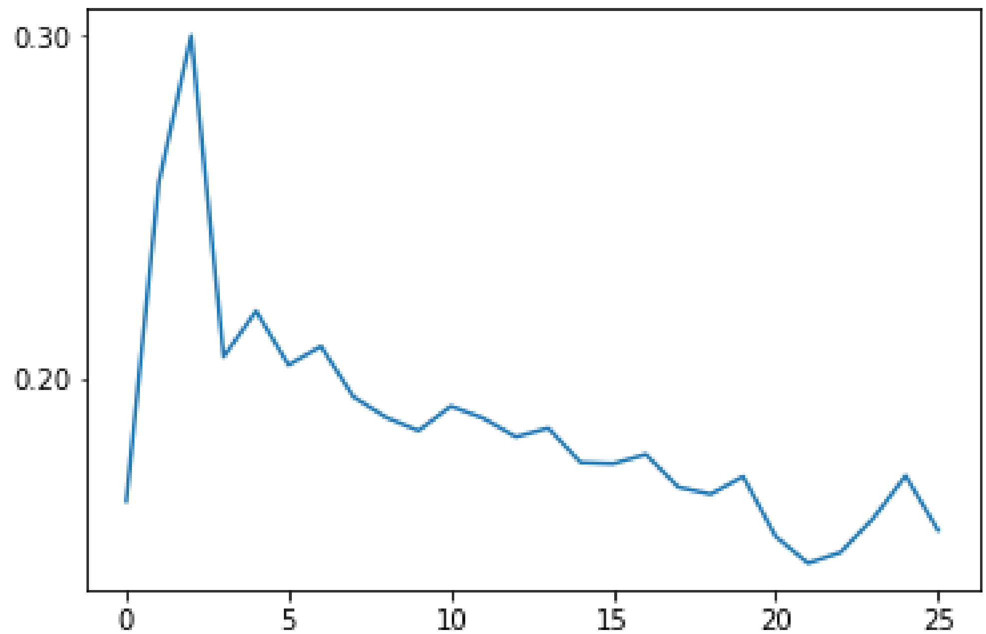
```
Out[110]:
```

```
In [111]: breaks = np.linspace(0, 25, 26).tolist()
          train["longest_credit_length_cat"] = train["longest_credit_length"].cut(breaks = break
```

```
In [112]: df = mean_target(train, "longest_credit_length_cat")
```

```
In [113]: plt.yscale("logit")
          plt.plot(df["longest_credit_length_cat"].index, df["mean_bad_loan"])
```

```
Out[113]: [<matplotlib.lines.Line2D at 0x1174895f8>]
```



In [114]:
```
df[0:4]
```

Out[114]:

	longest_credit_length_cat	nrow	mean_bad_loan
0	NaN	0	0.171094
1	(0.0,1.0]	67	0.253731
2	(1.0,2.0]	100	0.300000
3	(2.0,3.0]	914	0.205689

In [115]:
```
df[20:26]
```

Out[115]:

	longest_credit_length_cat	nrow	mean_bad_loan
20	(19.0,20.0]	4682	0.163392
21	(20.0,21.0]	3892	0.157760
22	(21.0,22.0]	3350	0.160000
23	(22.0,23.0]	3092	0.167206
24	(23.0,24.0]	2856	0.176821
25	(24.0,25.0]	2471	0.164711

In [116]:
```
x_trans.remove("longest_credit_length")
x_trans.append("longest_credit_length_core")
x_trans.append("longest_credit_length_delta")

train["longest_credit_length_core"] = h2o.H2OFrame.ifelse(train["longest_credit_length
                                                          3, train["longest_credit_len
train["longest_credit_length_core"] = h2o.H2OFrame.ifelse(train["longest_credit_length
                                                          train["longest_credit_length

train["longest_credit_length_delta"] = train["longest_credit_length"] - train["longest
```

1.6.14 Converting Income Verification Status to a 0/1 Indicator

Given that incomes are either verified or not verified, we will create a simplifed `verified` binary indicator that is 1 when income has been verified.

In [117]:
```
train["verification_status"].table()
```

Out[117]:

In [118]:
```
x_trans.remove("verification_status")
x_trans.append("verified")

train["verified"] = train["verification_status"] == "verified"
```

In [119]:
```
train["verified"].table()
```

Out[119]:

1.7 Train Models Using Transformed Data

In [120]:
```
print(aml.leaderboard)
```

In [121]:
```
print(x_trans)
```

```
['emp_length', 'loan_amnt_core', 'loan_amnt_delta', 'term_60months', 'emp_length_missing', 'home
```

In [122]:
```
# Set up AutoML

aml = H2OAutoML(max_runtime_secs=run_time)
```

In [123]:
```
model_start_time = time.time()

try:
  aml.train(x = x_trans, y = y, training_frame = train)  # Change training_frame=train
except Exception as e:
  logging.critical('aml.train')
  h2o.download_all_logs(dirname=logs_path, filename=logfile)
  h2o.cluster().shutdown()
  sys.exit(4)


meta_data['model_execution_time'] = {"classification":(time.time() - model_start_time)
```

```
AutoML progress: || 100%
```

```
In [124]: d=meta_data['model_execution_time']
          d['classification_trans']=(time.time() - model_start_time)
          meta_data['model_execution_time'] = d
```

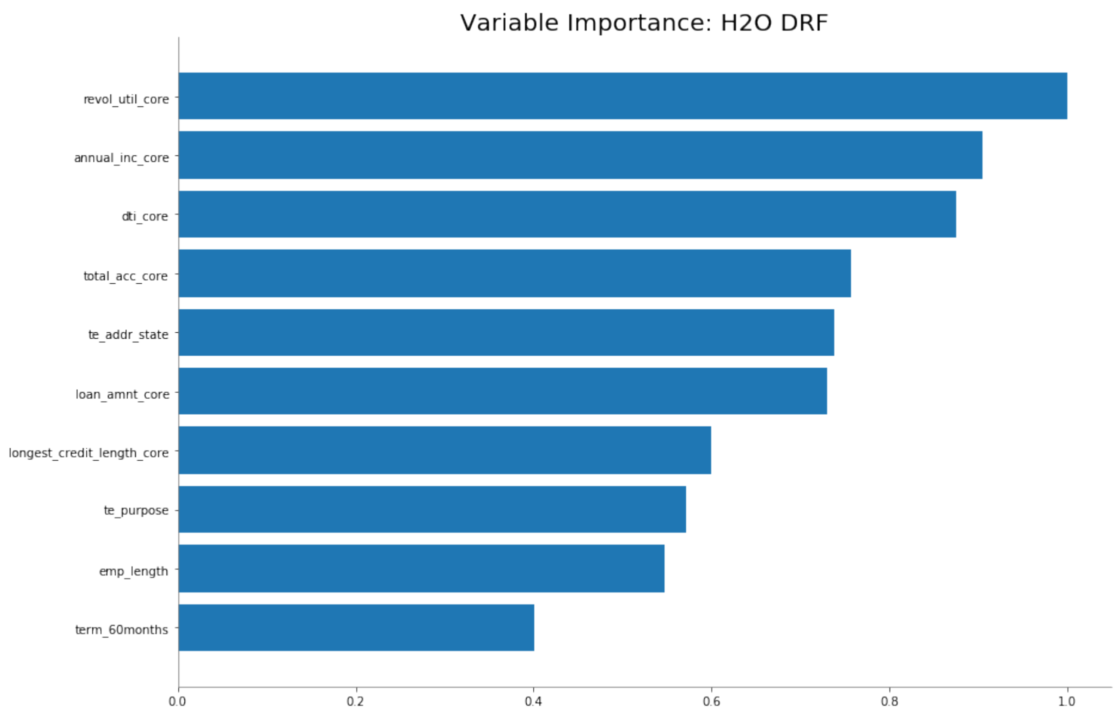
```
In [125]: meta_data
```

```
Out[125]: {'model_execution_time': {'classification': 348.23875093460083,
            'classification_trans': 348.24420404434204}}
```

```
In [127]: print(aml.leaderboard)
```

```
In [128]: best_model_trans = h2o.get_model(aml.leaderboard[3,'model_id'])
```

```
In [129]: best_model_trans.varimp_plot()
```



1.8 Shutdown H2O Cluster

```
In [ ]: h2o.cluster().shutdown()
```