Assignment 2: Lending Club Data

Huey Ning Lok

February 12, 2019

The Lending Club is a platform which allows the crowdfunding of various loans. Various investors are able to browse the profiles of people applying for loans and decide whether or not to help fund them. The Lending club has made an anonymized set of data available for anyone to study at: https://www.lendingclub.com/info/download-data.action

In this assignment, I will build a model that predicts the largest loan amount that will be successfully funded for given individual. This model can then be used to advise applicants on how much they could apply for.

1 Data Cleaning and Preprocessing

1.1 Importing Libraries and Importing Data

The data imported is the 2015 Approved Loan and 2015 Declined Loan csv data from the lending club website.

```
In [ ]: #import libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [202]: #import loan data
          loan = pd.read_csv("LoanStats3d.csv",skiprows=1)
          #column names
          print(list(loan))
          #peak at the df
          loan.head()
```

```
/anaconda3/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2785:
DtypeWarning: Columns (0,19,55) have mixed types. Specify dtype option on import or
set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)


['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate',
'installment', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownership',
'annual_inc', 'verification_status', 'issue_d', 'loan_status', 'pymnt_plan', 'url',
'desc', 'purpose', 'title', 'zip_code', 'addr_state', 'dti', 'delinq_2yrs',
'earliest_cr_line', 'inq_last_6mths', 'mths_since_last_delinq',
'mths_since_last_record', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util',
'total_acc', 'initial_list_status', 'out_prncp', 'out_prncp_inv', 'total_pymnt',
'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int', 'total_rec_late_fee',
'recoveries', 'collection_recovery_fee', 'last_pymnt_d', 'last_pymnt_amnt',
```

```
'next_pymnt_d', 'last_credit_pull_d', 'collections_12_mths_ex_med',
'mths_since_last_major_derog', 'policy_code', 'application_type', 'annual_inc_joint',
'dti_joint', 'verification_status_joint', 'acc_now_delinq', 'tot_coll_amt',
'tot_cur_bal', 'open_acc_6m', 'open_act_il', 'open_il_12m', 'open_il_24m',
'mths_since_rcnt_il', 'total_bal_il', 'il_util', 'open_rv_12m', 'open_rv_24m',
'max_bal_bc', 'all_util', 'total_rev_hi_lim', 'inq_fi', 'total_cu_tl', 'inq_last_12m',
'acc_open_past_24mths', 'avg_cur_bal', 'bc_open_to_buy', 'bc_util',
'chargeoff_within_12_mths', 'delinq_amnt', 'mo_sin_old_il_acct',
'mo_sin_old_rev_tl_op', 'mo_sin_rcnt_rev_tl_op', 'mo_sin_rcnt_tl', 'mort_acc',
'mths_since_recent_bc', 'mths_since_recent_bc_dlq', 'mths_since_recent_inq',
'mths_since_recent_revol_delinq', 'num_accts_ever_120_pd', 'num_actv_bc_tl',
'num_actv_rev_tl', 'num_bc_sats', 'num_bc_tl', 'num_il_tl', 'num_op_rev_tl',
'num_rev_accts', 'num_rev_tl_bal_gt_0', 'num_sats', 'num_tl_120dpd_2m',
'num_tl_30dpd', 'num_tl_90g_dpd_24m', 'num_tl_op_past_12m', 'pct_tl_nvr_dlq',
'percent_bc_gt_75', 'pub_rec_bankruptcies', 'tax_liens', 'tot_hi_cred_lim',
'total_bal_ex_mort', 'total_bc_limit', 'total_il_high_credit_limit',
'revol_bal_joint', 'sec_app_earliest_cr_line', 'sec_app_inq_last_6mths',
'sec_app_mort_acc', 'sec_app_open_acc', 'sec_app_revol_util', 'sec_app_open_act_il',
'sec_app_num_rev_accts', 'sec_app_chargeoff_within_12_mths',
'sec_app_collections_12_mths_ex_med', 'sec_app_mths_since_last_major_derog',
'hardship_flag', 'hardship_type', 'hardship_reason', 'hardship_status',
'deferral_term', 'hardship_amount', 'hardship_start_date', 'hardship_end_date',
'payment_plan_start_date', 'hardship_length', 'hardship_dpd', 'hardship_loan_status',
'orig_projected_additional_accrued_interest', 'hardship_payoff_balance_amount',
'hardship_last_payment_amount', 'disbursement_method', 'debt_settlement_flag',
'debt_settlement_flag_date', 'settlement_status', 'settlement_date',
'settlement_amount', 'settlement_percentage', 'settlement_term']
```

```
Out[202]:     id  member_id  loan_amnt  funded_amnt  funded_amnt_inv        term  \
          0  NaN        NaN    24700.0      24700.0          24700.0   36 months
          1  NaN        NaN    17925.0      17925.0          17925.0   60 months
          2  NaN        NaN     9000.0       9000.0           9000.0   36 months
          3  NaN        NaN    11550.0      11550.0          11550.0   60 months
          4  NaN        NaN    12000.0      12000.0          12000.0   60 months

            int_rate  installment grade sub_grade       ...         \
          0   11.99%       820.28     C        C1       ...
          1   17.27%       448.09     D        D3       ...
          2    8.49%       284.07     B        B1       ...
          3   16.59%       284.51     D        D2       ...
          4    9.80%       253.79     B        B3       ...

            hardship_payoff_balance_amount hardship_last_payment_amount  \
          0                            NaN                          NaN
          1                            NaN                          NaN
          2                            NaN                          NaN
          3                            NaN                          NaN
          4                            NaN                          NaN

            disbursement_method  debt_settlement_flag debt_settlement_flag_date  \
          0                Cash                     N                       NaN
          1                Cash                     N                       NaN
```

```
2              Cash                  N                    NaN
3              Cash                  N                    NaN
4              Cash                  N                    NaN

  settlement_status settlement_date settlement_amount  settlement_percentage  \
0               NaN             NaN               NaN                    NaN
1               NaN             NaN               NaN                    NaN
2               NaN             NaN               NaN                    NaN
3               NaN             NaN               NaN                    NaN
4               NaN             NaN               NaN                    NaN

  settlement_term
0             NaN
1             NaN
2             NaN
3             NaN
4             NaN

[5 rows x 145 columns]
```

In [203]:
```python
#import reject data
reject = pd.read_csv("RejectStatsD.csv",skiprows=1)
#column names
print(list(reject))
#peak at the df
reject.head()
```

```
['Amount Requested', 'Application Date', 'Loan Title', 'Risk_Score', 'Debt-To-Income
Ratio', 'Zip Code', 'State', 'Employment Length', 'Policy Code']
```

Out[203]:
```
   Amount Requested Application Date          Loan Title  Risk_Score  \
0           30000.0       2015-01-01  debt_consolidation       681.0
1            5000.0       2015-01-01  debt_consolidation       648.0
2           10000.0       2015-01-01  Debt consolidation       721.0
3           10000.0       2015-01-01      major_purchase       659.0
4            5000.0       2015-01-01  debt_consolidation       501.0

  Debt-To-Income Ratio Zip Code State Employment Length  Policy Code
0               35.65%    958xx    CA          < 1 year            0
1               10.62%    945xx    CA          < 1 year            0
2               10.02%    750xx    TX           7 years            0
3               19.05%    853xx    AZ          < 1 year            0
4               10.73%    475xx    IN          < 1 year            0
```

1.2 Choosing which columns to keep

I can only keep columns that exist in both the approved and rejected loan datasets. The table below shows the column names that exist in both the loan data and reject data that share the same definition (as defined by the data dictionary on the lending data website).

Loan Data	Reject Data	Definition
loan_amnt	Amount Requested	The total amount requested by the borrower
title	Loan Title	The loan title provided by the borrower
dti	Debt-To-Income Ratio	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.
zip_code	Zip Code	The first 3 numbers of the zip code provided by the borrower in the loan application.
addr_state	State	The state provided by the borrower in the loan application
emp_length	Employment Length	Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
policy_code	Policy Code	publicly available policy_code=1; new products not publicly available policy_code=2

In [204]:
```
#Drop all columns that do not exist in both dataframes.
loan =
loan[['loan_amnt','title','dti','zip_code','addr_state','emp_length','policy_code']]
loan['Status'] = 1
print(loan.shape)
loan.head()
```

```
(421097, 8)
```

Out[204]:
```
   loan_amnt                    title    dti zip_code addr_state emp_length  \
0    24700.0                 Business  16.06    577xx         SD  10+ years
1    17925.0       Debt consolidation  27.78    432xx         OH  10+ years
2     9000.0       Debt consolidation   8.43    346xx         FL    8 years
3    11550.0  Credit card refinancing  21.07    436xx         OH    5 years
4    12000.0       Debt consolidation  23.84    660xx         KS  10+ years

   policy_code  Status
0          1.0       1
1          1.0       1
2          1.0       1
3          1.0       1
4          1.0       1
```

In [205]:
```
reject = reject[['Amount Requested','Loan Title','Debt-To-Income Ratio','Zip
Code','State','Employment Length','Policy Code']]
reject['Status'] = 0
print(reject.shape)
reject.head()
```

```
(2859379, 8)
```

Out[205]:
```
   Amount Requested           Loan Title Debt-To-Income Ratio Zip Code State  \
0           30000.0   debt_consolidation               35.65%    958xx    CA
```

```
1          5000.0  debt_consolidation                 10.62%    945xx    CA
2         10000.0  Debt consolidation                 10.02%    750xx    TX
3         10000.0      major_purchase                 19.05%    853xx    AZ
4          5000.0  debt_consolidation                 10.73%    475xx    IN

  Employment Length  Policy Code  Status
0          < 1 year            0       0
1          < 1 year            0       0
2           7 years            0       0
3          < 1 year            0       0
4          < 1 year            0       0
```

In [206]:
```python
#rename columns in loan data so that it is the same as reject data
loan.columns = list(reject)
data = pd.concat([loan,reject])
print(data.shape)
data.head()
```

```
(3280476, 8)
```

Out[206]:
```
   Amount Requested              Loan Title Debt-To-Income Ratio Zip Code  \
0           24700.0                Business                16.06    577xx
1           17925.0      Debt consolidation                27.78    432xx
2            9000.0      Debt consolidation                 8.43    346xx
3           11550.0  Credit card refinancing               21.07    436xx
4           12000.0      Debt consolidation                23.84    660xx

  State Employment Length  Policy Code  Status
0    SD         10+ years          1.0       1
1    OH         10+ years          1.0       1
2    FL           8 years          1.0       1
3    OH           5 years          1.0       1
4    KS         10+ years          1.0       1
```

In [207]:
```python
data.info(verbose=True, null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3280476 entries, 0 to 2859378
Data columns (total 8 columns):
Amount Requested        3280474 non-null float64
Loan Title              3280342 non-null object
Debt-To-Income Ratio    3280472 non-null object
Zip Code                3280474 non-null object
State                   3280474 non-null object
Employment Length       3182914 non-null object
Policy Code             3280474 non-null float64
Status                  3280476 non-null int64
dtypes: float64(2), int64(1), object(5)
memory usage: 225.3+ MB
```

In [208]:
```python
#drop NaN values from data
data.dropna(inplace=True)
data.info(verbose=True, null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3182789 entries, 0 to 2859378
Data columns (total 8 columns):
Amount Requested        3182789 non-null float64
Loan Title              3182789 non-null object
Debt-To-Income Ratio    3182789 non-null object
Zip Code                3182789 non-null object
State                   3182789 non-null object
Employment Length       3182789 non-null object
Policy Code             3182789 non-null float64
Status                  3182789 non-null int64
dtypes: float64(2), int64(1), object(5)
memory usage: 218.5+ MB
```

In [209]:
```python
#convert debt-to-income ratio and employment length to numeric data types
data['Debt-To-Income Ratio'] = data['Debt-To-Income
Ratio'].astype(str).str.extract('(\d+)').astype(float)
data['Employment Length'] = data['Employment Length'].str.extract('(\d+)').astype(int)

print(data.info(verbose=True, null_counts=True))
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3182789 entries, 0 to 2859378
Data columns (total 8 columns):
Amount Requested        3182789 non-null float64
Loan Title              3182789 non-null object
Debt-To-Income Ratio    3182789 non-null float64
Zip Code                3182789 non-null object
State                   3182789 non-null object
Employment Length       3182789 non-null int64
Policy Code             3182789 non-null float64
Status                  3182789 non-null int64
dtypes: float64(3), int64(2), object(3)
memory usage: 218.5+ MB
None
```

Out[209]:

	Amount Requested	Loan Title	Debt-To-Income Ratio	Zip Code
0	24700.0	Business	16.0	577xx
1	17925.0	Debt consolidation	27.0	432xx
2	9000.0	Debt consolidation	8.0	346xx
3	11550.0	Credit card refinancing	21.0	436xx
4	12000.0	Debt consolidation	23.0	660xx

	State	Employment Length	Policy Code	Status
0	SD	10	1.0	1
1	OH	10	1.0	1
2	FL	8	1.0	1
3	OH	5	1.0	1
4	KS	10	1.0	1

1.3 Closer Analysis for In-depth Preprocessing

In [210]:
```
#basic stats for numerical columns
data.describe(include = [np.number])
```

Out[210]:

	Amount Requested	Debt-To-Income Ratio	Employment Length
count	3.182789e+06	3.182789e+06	3.182789e+06
mean	1.435574e+04	1.071487e+02	2.113442e+00
std	1.872363e+04	2.039714e+03	2.519112e+00
min	1.000000e+03	0.000000e+00	1.000000e+00
25%	5.000000e+03	7.000000e+00	1.000000e+00
50%	1.000000e+04	1.800000e+01	1.000000e+00
75%	2.000000e+04	3.200000e+01	1.000000e+00
max	4.500000e+05	1.019525e+06	1.000000e+01

	Policy Code	Status
count	3.182789e+06	3.182789e+06
mean	1.302741e-01	1.247814e-01
std	3.446672e-01	3.304710e-01
min	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00
50%	0.000000e+00	0.000000e+00
75%	0.000000e+00	0.000000e+00
max	2.000000e+00	1.000000e+00

In [211]:
```
#basic stats for categorical columns
data.describe(include = ['O'])
```

Out[211]:

	Loan Title	Zip Code	State
count	3182789	3182789	3182789
unique	55	998	49
top	debt_consolidation	112xx	CA
freq	1257311	33264	400190

1.4 Observations

- The Policy Code column appears to contain zeros, although the data dictionary only includes definitions for policy codes of ones and twos.
  - Since more than half the dataset contains policy code values of 0, and there is no way to intuitively predict what policy code == 0 could stand for, I will drop the policy code column.
- There are 55 unique Loan Title values, 998 unique Zip Code values, and 49 unique State values.
  - Due to the high amount of unique Zip Code values, I will drop the Zip Code column as it may be quite messy to one-hot-encode 998 unique values, and Zip Code information can also be indirectly inferred from State values.
  - The Loan Title values will be analyzed and manually sorted into general, representative categories.
  - The State values will be kept as they are.
- From the summary stats tables aboves, it looks like there is a huge outlier in the Debt-To-Income Ratio category - the max value is 10^5 times bigger than the 75% percentile.[1]
  - While the outlier is erroneous, it is also feasible; there could be an individual out there with very high debt and very low income. Excluding the outlier could then lead to a biased dataset. On the other hand, the outlier may also be a result of human error - in which case the resulting model would not be representative of the data. I have decided to keep the outlier since no robust tests were conducted to determine whether it can be considered a legitimate outlier, and it is possible for an individual to have a very high debt-to-income ratio.

1.4.1 Drop Policy Code Column

In [212]:
```python
#Checking the number of rows with policy code values == 0
print(f"Number of rows where policy code == 0: {sum(data['Policy Code'] == 0)}")

#drop Policy Code column
data.drop(columns=['Policy Code'], inplace=True)

#peak at the df
data.head()
```

```
Number of rows where policy code == 0: 2776895
```

Out[212]:

	Amount Requested	Loan Title	Debt-To-Income Ratio	Zip Code \
0	24700.0	Business	16.0	577xx
1	17925.0	Debt consolidation	27.0	432xx
2	9000.0	Debt consolidation	8.0	346xx
3	11550.0	Credit card refinancing	21.0	436xx

[1] **#descriptivestats:** By observing a summary of the descriptive stats of the dataframe, I was able to manually detect irregular discrepancies in the data. In this case, the policy code contains 0 as a minimum (which is not included in the data dictionary), and the Debt-To-Income Ratio has a stark outlier - this is inferred by observing the max value (1.02e+06), which is a few orders of magnitude bigger than the 75% percentile (3.2e+01). Although the descriptive stats isn't a robust method of analyzing data irregularities, any stark irregularities can be intuitively singled out.

```
4          12000.0      Debt consolidation                    23.0    660xx

  State  Employment Length  Status
0    SD                 10       1
1    OH                 10       1
2    FL                  8       1
3    OH                  5       1
4    KS                 10       1
```

1.4.2 Drop Zip Code Column

In [213]:
```
#Drop the Zip code column
data.drop(columns=['Zip Code'], inplace=True)

#peak at the df
data.head()
```

Out[213]:
```
   Amount Requested                Loan Title  Debt-To-Income Ratio State  \
0           24700.0                  Business                  16.0    SD
1           17925.0        Debt consolidation                  27.0    OH
2            9000.0        Debt consolidation                   8.0    FL
3           11550.0   Credit card refinancing                  21.0    OH
4           12000.0        Debt consolidation                  23.0    KS

   Employment Length  Status
0                 10       1
1                 10       1
2                  8       1
3                  5       1
4                 10       1
```

1.4.3 Sort and One-Hot-Encode Loan Title Column

In [214]:
```
#Get value counts of loan title column
loan_titles_value_counts = data['Loan Title'].value_counts()
print(loan_titles_value_counts)
```

```
debt_consolidation                         1257311
other                                       339305
Debt consolidation                          315246
credit_card                                 299620
home_improvement                            145450
Credit card refinancing                     128477
car                                         108601
moving                                       86701
Business Loan                                83657
medical                                      80339
major_purchase                               80119
small_business                               66971
house                                        38949
vacation                                     35370
Home improvement                             34953
Other                                        27938
Major purchase                               11378
```

```
Business                                        11082
renewable_energy                                 6949
Medical expenses                                 5831
Car financing                                    5675
Moving and relocation                            3882
Vacation                                         3457
Business Line Of Credit                          2536
Home buying                                      2485
Green loan                                        478
Credit Card/Auto Repair                             1
freeup                                              1
Pay off Lowes Card                                  1
SAVE                                                1
thad31                                              1
new kitchen for momma!                              1
Prescription Drug and Medical Costs                 1
althea9621                                          1
Student Loan                                        1
dougie03                                            1
10 months away from being an RN                     1
New Baby and New House (CC Consolidate)             1
Business Advertising Loan                           1
loan                                                1
considerate                                         1
Trying to come back to reality!                     1
Simple Loan Until Contract Is Completed             1
Consolidation Loan                                  1
Need a decent rate on car financing                 1
educational                                         1
Consolidate debt                                    1
Small Business Expansion                            1
DebtC                                               1
Auto Financing                                      1
odymeds                                             1
new day                                             1
Paying off higher interest cards & auto             1
Learning and training                               1
smmoore2                                            1
Name: Loan Title, dtype: int64
```

From the value count above, we can see the most frequently used loan titles (any loan title with value > 1). There are some duplicates among these loan titles (e.g. debt_consolidation & Debt consolidation; home_improvement & Home Improvement), and titles which belong to the same general category (e.g. renewable_energy & Green loan; Home improvement & Home Buying).

General categories will be created and the duplicates as well as titles with similar meaning will be sorted into the categories as follows:

General Category	Loan Titles Included
Debt Consolidation	debt_consolidation, Debt consolidation,
Credit Card	credit_card, Credit card refinancing
House	Home improvement, home_improvement, house, Home buying
Car	car, Car financing

General Category	Loan Titles Included
Major Purchase	major_purchase, Major purchase
Moving	Moving and relocation, moving
Business	Business Loan, Business, small_business, Business Line Of Credit
Medical	medical, Medical expenses
Vacation	vacation, Vacation
Green Loan	renewable_energy, Green loan
Other	other, Other, all loan titles with value count == 1

In [215]:
```
rare_loan_titles = data['Loan
Title'].isin(loan_titles_value_counts.index[loan_titles_value_counts == 1])
print(f"There are {sum(rare_loan_titles)} loan titles with a value count of 1.\n")

#change value_count == 1 loan titles to "Other"
data.loc[rare_loan_titles, 'Loan Title'] = "Other"

print(data['Loan Title'].value_counts())
```

```
There are 29 loan titles with a value count of 1.

debt_consolidation         1257311
other                       339305
Debt consolidation          315246
credit_card                 299620
home_improvement            145450
Credit card refinancing     128477
car                         108601
moving                       86701
Business Loan                83657
medical                      80339
major_purchase               80119
small_business               66971
house                        38949
vacation                     35370
Home improvement             34953
Other                        27967
Major purchase               11378
Business                     11082
renewable_energy              6949
Medical expenses              5831
Car financing                 5675
Moving and relocation         3882
Vacation                      3457
Business Line Of Credit       2536
Home buying                   2485
Green loan                     478
Name: Loan Title, dtype: int64
```

In [217]:
```
#Sort loan titles as according to the General Category dictionary above

#loan titles dictionary
loan_titles_dct = {
                     'Debt consolidation':'Debt Consolidation',
                     'debt_consolidation':'Debt Consolidation',
                     'credit_card':'Credit Card',
                     'Credit card refinancing':'Credit Card',
                     'Home improvement':'House',
```

```
                          'Home improvement':'House',
                          'home_improvement':'House',
                          'house':'House',
                          'Home buying':'House',
                          'car':'Car',
                          'Car financing':'Car',
                          'major_purchase':'Major Purchase',
                          'Major purchase':'Major Purchase',
                          'Moving and relocation':'Moving',
                          'moving':'Moving',
                          'Business Loan':'Business',
                          'small_business':'Business',
                          'Business':'Business',
                          'Business Line Of Credit':'Business',
                          'medical':'Medical',
                          'Medical expenses':'Medical',
                          'vacation':'Vacation',
                          'Vacation':'Vacation',
                          'renewable_energy':'Green Loan',
                          'Green loan':'Green Loan',
                          'other':'Other',
                          'Other':'Other'
                      }


data['Loan Title'] = data['Loan Title'].map(loan_titles_dct)

print(data['Loan Title'].value_counts())
```

```
Debt Consolidation    1572557
Credit Card            428097
Other                  367272
House                  221837
Business               164246
Car                    114276
Major Purchase          91497
Moving                  90583
Medical                 86170
Vacation                38827
Green Loan               7427
Name: Loan Title, dtype: int64
```

In [222]:
```
#Check for NaN values
print(f"Presence of NaN values: {data['Loan Title'].isnull().values.any()}")
```

```
Presence of NaN values: False
```

In [236]:
```
#One-hot-encode the Loan Title column
from sklearn import preprocessing

mlb = preprocessing.MultiLabelBinarizer()

loan_titles_df = pd.DataFrame(mlb.fit_transform(data['Loan Title'].str.split(',
')),columns=mlb.classes_, index=data.index)
print(loan_titles_df.shape)
loan_titles_df.head()
```

```
(3182789, 11)
```

Out[236]:
```
   Business  Car  Credit Card  Debt Consolidation  Green Loan  House  \
0         1    0            0                   0           0      0
```

```
1          0    0            0                   1           0        0
2          0    0            0                   1           0        0
3          0    0            1                   0           0        0
4          0    0            0                   1           0        0

   Major Purchase  Medical  Moving  Other  Vacation
0               0        0       0      0         0
1               0        0       0      0         0
2               0        0       0      0         0
3               0        0       0      0         0
4               0        0       0      0         0
```

1.4.4 One-Hot-Encode State Column

In [232]:
```
#Observe and analyze State values
print(data['State'].value_counts())
```

```
CA    400190
TX    283460
NY    246080
FL    241847
GA    117447
IL    116277
PA    115775
OH    113283
NJ     99495
NC     97956
VA     88273
MI     85254
MD     68073
TN     66279
AZ     66238
MA     64275
WA     58178
IN     57982
CO     56771
AL     55329
MO     55078
SC     50255
LA     49288
MN     43130
WI     42167
CT     39999
NV     39596
KY     39349
OK     35139
OR     32533
AR     31347
MS     29836
KS     26914
UT     21207
NM     18750
HI     18515
WV     17398
NH     13273
RI     12846
```

```
DE      10471
NE       8723
MT       8206
AK       7313
SD       6743
DC       6595
VT       6448
WY       6252
ME       3689
ND       3267
Name: State, dtype: int64
```

In [238]:
```
#One-hot-encode the State column

mlb = preprocessing.MultiLabelBinarizer()

state_df = pd.DataFrame(mlb.fit_transform(data['State'].str.split(',
')),columns=mlb.classes_, index=data.index)
print(state_df.shape)
state_df.head()
```

```
(3182789, 49)
```

Out[238]:

```
   AK  AL  AR  AZ  CA  CO  CT  DC  DE  FL ...  SD  TN  TX  UT  VA  VT  WA  WI  \
0   0   0   0   0   0   0   0   0   0   0 ...   1   0   0   0   0   0   0   0
1   0   0   0   0   0   0   0   0   0   0 ...   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   1 ...   0   0   0   0   0   0   0   0
3   0   0   0   0   0   0   0   0   0   0 ...   0   0   0   0   0   0   0   0
4   0   0   0   0   0   0   0   0   0   0 ...   0   0   0   0   0   0   0   0

   WV  WY
0   0   0
1   0   0
2   0   0
3   0   0
4   0   0

[5 rows x 49 columns]
```

1.5 Create the Full Dataset

In [265]:
```
#combining everything back together:
full_data = pd.concat([data,loan_titles_df,state_df], axis = 1)
full_data.head()
```

Out[265]:

```
   Amount Requested          Loan Title  Debt-To-Income Ratio State  \
0           24700.0            Business                  16.0    SD
1           17925.0  Debt Consolidation                  27.0    OH
2            9000.0  Debt Consolidation                   8.0    FL
3           11550.0         Credit Card                  21.0    OH
4           12000.0  Debt Consolidation                  23.0    KS

   Employment Length  Status  Business  Car  Credit Card  Debt Consolidation  \
```

```
0                    10         1           1     0               0                        0
1                    10         1           0     0               0                        1
2                     8         1           0     0               0                        1
3                     5         1           0     0               1                        0
4                    10         1           0     0               0                        1

   ...  SD  TN  TX  UT  VA  VT  WA  WI  WV  WY
0 ...   1   0   0   0   0   0   0   0   0   0
1 ...   0   0   0   0   0   0   0   0   0   0
2 ...   0   0   0   0   0   0   0   0   0   0
3 ...   0   0   0   0   0   0   0   0   0   0
4 ...   0   0   0   0   0   0   0   0   0   0

[5 rows x 66 columns]
```

In [361]: full_data.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3182789 entries, 0 to 2859378
Data columns (total 66 columns):
Amount Requested        float64
Loan Title              object
Debt-To-Income Ratio    float64
State                   object
Employment Length       int64
Status                  int64
Business                int64
Car                     int64
Credit Card             int64
Debt Consolidation      int64
Green Loan              int64
House                   int64
Major Purchase          int64
Medical                 int64
Moving                  int64
Other                   int64
Vacation                int64
AK                      int64
AL                      int64
AR                      int64
AZ                      int64
CA                      int64
CO                      int64
CT                      int64
DC                      int64
DE                      int64
FL                      int64
GA                      int64
HI                      int64
IL                      int64
IN                      int64
KS                      int64
KY                      int64
LA                      int64
MA                      int64
MD                      int64
```

```
ME                       int64
MI                       int64
MN                       int64
MO                       int64
MS                       int64
MT                       int64
NC                       int64
ND                       int64
NE                       int64
NH                       int64
NJ                       int64
NM                       int64
NV                       int64
NY                       int64
OH                       int64
OK                       int64
OR                       int64
PA                       int64
RI                       int64
SC                       int64
SD                       int64
TN                       int64
TX                       int64
UT                       int64
VA                       int64
VT                       int64
WA                       int64
WI                       int64
WV                       int64
WY                       int64
dtypes: float64(2), int64(62), object(2)
memory usage: 1.6+ GB
```

2 Part 2: Modeling & Evaluation

Since the goal is to predict the largest loan amount that will be successfully funded for a given individual, I will be training a logistic regression model to predict the probability that a given individual would have their requested amount approved based on the following independent variables:

- Amount Requested
- Debt-To-Income Ratio
- Employment Length
- Loan Title (one-hot-encoded)
- State (one-hot-encoded)

The response variables will be the status of the loan:

- Status
  - 1: Approved
  - 0: Rejected

2.1 Balancing the Response Class

The response class (Status of Loan) has a rather unbalanced approval to rejection class ratio (~0.14). To balance the ratio of approval to rejection (so that the model is properly trained on predicting either class), I will randomly sample from the rejection set in order to cut it down to the length of the approval set. Lastly, I combine both sets back together.

```
In [372]: from sklearn.model_selection import train_test_split

          #check ratio of approval/rejection class in full_data
          full_data_approvals = full_data[full_data['Status']==1]
          full_data_rejections = full_data[full_data['Status']==0]

          print(f'Number of approvals in original data: {len(full_data_approvals)}')
          print(f'Number of rejections in original data: {len(full_data_rejections)}')
          print(f'Ratio of approval/rejection:
          {len(full_data_approvals)/len(full_data_rejections)}\n')

          #remove some rejection data to obtain a dataset with balanced predictor classes
          full_data_rejection_sample = full_data_rejections.sample(n = len(full_data_approvals))
          full_data_balanced = pd.concat([full_data_rejection_sample,full_data_approvals])

          print(f'Number of approvals in balanced data:
          {len(full_data_balanced[full_data_balanced["Status"]==1])}')
          print(f'Number of rejections in balanced data:
          {len(full_data_balanced[full_data_balanced["Status"]==0])}')
          print(f'Ratio of approval/rejection: {len(full_data_balanced[full_data_balanced["Status"]
          ]==1])/len(full_data_balanced[full_data_balanced["Status"]==0])}')
```

```
Number of approvals in original data: 397153
Number of rejections in original data: 2785636
Ratio of approval/rejection: 0.1425717502214934

Number of approvals in balanced data: 397153
Number of rejections in balanced data: 397153
Ratio of approval/rejection: 1.0
```

2.2 Preventing Overfit: Splitting Data & Cross-Validation

The balanced dataset is split into training and test sets to prevent overfitting. Cross-validation will be conducted on the training set to finetune the hyperparameters of the model. It is important that the cross-validation process never sees the testing dataset in order to avoid information leakage and overfitting.

```
In [376]: #separate dataset into independent variables (X) and dependent variable (y)
          X = full_data_balanced.drop(columns=['Status','Loan Title','State'])
          y = full_data_balanced['Status']

          #splitting the data
          #choose a small training size to speed up the model fitting process
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, stratify = y)
          print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(476583, 63) (317723, 63) (476583,) (317723,)
```

2.3 Cross-validation

In the cross-validation process, only the training data is used. The LogisticRegressionCV function is used for cross-validation since it is more efficient than using the brute force parameter search provided by the GridSearchCV function.[2]

The LogisticRegressionCV function is mainly used to optimize for C, i.e. the inverse of regularization parameter values. When an integer is provided for the C parameter, e.g. Cs = 10, then a grid of Cs values are chosen in a logarithmic scale between 1e-4 and 1e4. [3]

I have chosen to perform cross-validation on:

- a logistic regression model using a liblinear solver with l1 penalty
- a logistic regression model using a liblinear solver with l2 penalty
- a logistic regression model using a lbfgs solver (lbfgs solver only supports l2 penalty, which is the default)

```
In [419]: from sklearn.linear_model import LogisticRegression,LogisticRegressionCV
          from sklearn.metrics import classification_report

          logr_lib_l1 = LogisticRegressionCV(Cs=10,solver='liblinear',penalty='l1').fit(X_train,
          y_train)
          logr_lib_l2 = LogisticRegressionCV(Cs=10,solver='liblinear',penalty='l2').fit(X_train,
          y_train)
          logr_lbfgs = LogisticRegressionCV(Cs=10,solver='lbfgs').fit(X_train, y_train)
```

```
In [420]: logr_models = [logr_lib_l1,logr_lib_l2,logr_lbfgs]

          print('Logistic Regression CV results using different solvers, optimizing for C.\n')

          for model in logr_models:
              print(f'Logistic Regression CV results using {str(model)}.\n')
              print(f'Array of C used for cross-validation: \n{model.Cs_}\n')
              print(f'Array of C that maps to the best scores across every class: \n{model.C_}\n')
              print(f'Grid of scores obtained during cross-validating each fold:
          \n{model.scores_[1]}\n')
              print(f'Mean accuracy score: \n{np.mean(model.scores_[1])}\n')
              print(f'Classification report:\n{classification_report(y_train,
          model.predict(X_train))}')
```

[2] https://scikit-learn.org/stable/modules/grid_search.html#multimetric-grid-search

[3] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html#sklearn.linear_model.LogisticRegressionCV

```
Logistic Regression CV results using different solvers, optimizing for C.

Logistic Regression CV results using LogisticRegressionCV(Cs=10, class_weight=None,
cv=None, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
           refit=True, scoring=None, solver='liblinear', tol=0.0001,
           verbose=0).

Array of C used for cross-validation:
[1.00000000e-04 7.74263683e-04 5.99484250e-03 4.64158883e-02
 3.59381366e-01 2.78255940e+00 2.15443469e+01 1.66810054e+02
 1.29154967e+03 1.00000000e+04]

Array of C that maps to the best scores across every class:
[2.7825594]

Grid of scores obtained during cross-validating each fold:
[[0.81104984 0.81417834 0.82075638 0.82226083 0.82255039 0.82251892
  0.82248115 0.82248744 0.82248744 0.82248744]
 [0.80908467 0.81206841 0.81825621 0.81957812 0.81961589 0.8197292
  0.81974179 0.81974808 0.81973549 0.81974808]
 [0.81011583 0.81342692 0.81963364 0.82093667 0.82104369 0.82106887
  0.82104998 0.82104369 0.82104369 0.82104369]]

Mean accuracy score:
0.819032358927715

Classification report:
             precision    recall  f1-score   support

          0       0.78      0.90      0.83    238291
          1       0.88      0.74      0.81    238292

avg / total       0.83      0.82      0.82    476583

Logistic Regression CV results using LogisticRegressionCV(Cs=10, class_weight=None,
cv=None, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
           refit=True, scoring=None, solver='liblinear', tol=0.0001,
           verbose=0).

Array of C used for cross-validation:
[1.00000000e-04 7.74263683e-04 5.99484250e-03 4.64158883e-02
 3.59381366e-01 2.78255940e+00 2.15443469e+01 1.66810054e+02
 1.29154967e+03 1.00000000e+04]

Array of C that maps to the best scores across every class:
[0.35938137]

Grid of scores obtained during cross-validating each fold:
[[0.81181151 0.81644446 0.8212096  0.82113407 0.82226712 0.82219159
  0.82225454 0.82226083 0.82226712 0.82227971]
 [0.81007296 0.81479407 0.81768968 0.8181492  0.81819956 0.81819956
  0.81819956 0.81934521 0.81820585 0.81821215]
 [0.81090268 0.81626589 0.81264006 0.82076671 0.82091149 0.82077301
  0.82076671 0.81962734 0.81972806 0.82076671]]
```

```
Mean accuracy score:
0.8186112343554276

Classification report:
             precision    recall  f1-score   support

          0       0.77      0.90      0.83    238291
          1       0.89      0.74      0.80    238292

avg / total       0.83      0.82      0.82    476583

Logistic Regression CV results using LogisticRegressionCV(Cs=10, class_weight=None,
cv=None, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
           refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0).

Array of C used for cross-validation:
[1.00000000e-04 7.74263683e-04 5.99484250e-03 4.64158883e-02
 3.59381366e-01 2.78255940e+00 2.15443469e+01 1.66810054e+02
 1.29154967e+03 1.00000000e+04]

Array of C that maps to the best scores across every class:
[2.7825594]

Grid of scores obtained during cross-validating each fold:
[[0.8039934  0.8039934  0.8039934  0.8039934  0.8039934  0.8039997
  0.8039934  0.8039934  0.8039934  0.8039934 ]
 [0.80166938 0.80166938 0.80166938 0.80166938 0.80166938 0.80166938
  0.80166938 0.80166938 0.80166938 0.80166938]
 [0.80292711 0.80292711 0.80292711 0.80292711 0.80292711 0.80292711
  0.80292711 0.80292711 0.80292711 0.80292711]]

Mean accuracy score:
0.802863507370328

Classification report:
             precision    recall  f1-score   support

          0       0.76      0.90      0.82    238291
          1       0.87      0.71      0.78    238292

avg / total       0.81      0.80      0.80    476583
```

2.4 Feature-scaling Models

I feature-scaled the independent variables to lie between 0 and 1 using a MinMaxScaler. Feature-scaling may improve the model if "noisy" variables had an unjustly high weight, while important variables had comparatively low weights. However, it may also decrease model accuracy if the "noisy" variables were already weighted low and the important variables were already weighted high in the original data; in such a case, feature-scaling would only emphasize the noise by giving it equal weightage as the important predictors.

In [413]:
```
from sklearn import preprocessing

#try with feature scaling
min_max_scaler = preprocessing.MinMaxScaler()
X_train_scaled = min_max_scaler.fit_transform(X_train)

logr_lib_l1_scaled =
LogisticRegressionCV(Cs=10,solver='liblinear',penalty='l1').fit(X_train_scaled, y_train)
logr_lib_l2_scaled =
LogisticRegressionCV(Cs=10,solver='liblinear',penalty='l2').fit(X_train_scaled, y_train)
```

In [418]:
```
logr_scaled_models = [logr_lib_l1_scaled,logr_lib_l2_scaled]

print('Logistic Regression CV results using liblinear solver with scaled features,
optimizing for C.\n')

for model in logr_scaled_models:
    print(f'Logistic Regression CV results using {str(model)}.\n')
    print(f'Array of C used for cross-validation: \n{model.Cs_}\n')
    print(f'Array of C that maps to the best scores across every class: \n{model.C_}\n')
    print(f'Grid of scores obtained during cross-validating each fold:
\n{model.scores_[1]}\n')
    print(f'Mean accuracy score: \n{np.mean(model.scores_[1])}\n')
    print(f'Classification report:\n{classification_report(y_train,
model.predict(X_train_scaled))}')
```

```
Logistic Regression CV results using liblinear solver with scaled features, optimizing
for C.

Logistic Regression CV results using LogisticRegressionCV(Cs=10, class_weight=None,
cv=None, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
           refit=True, scoring=None, solver='liblinear', tol=0.0001,
           verbose=0).

Array of C used for cross-validation:
[1.00000000e-04 7.74263683e-04 5.99484250e-03 4.64158883e-02
 3.59381366e-01 2.78255940e+00 2.15443469e+01 1.66810054e+02
 1.29154967e+03 1.00000000e+04]

Array of C that maps to the best scores across every class:
[2.7825594]

Grid of scores obtained during cross-validating each fold:
[[0.78129446 0.81081064 0.81713689 0.82349461 0.82197757 0.82376528
  0.8231358  0.82256927 0.82250003 0.82248115]
 [0.77944241 0.80902802 0.81520323 0.82001246 0.82058529 0.82164282
  0.82015095 0.81974808 0.81975438 0.81974179]
 [0.78140501 0.81015359 0.81630366 0.82128919 0.82176759 0.82293214
  0.8218872  0.82113811 0.8210311  0.82104369]]
```

```
Mean accuracy score:
0.8157808796758563

Classification report:
             precision    recall  f1-score   support

          0       0.78      0.91      0.84    238291
          1       0.89      0.74      0.81    238292

avg / total       0.83      0.82      0.82    476583

Logistic Regression CV results using LogisticRegressionCV(Cs=10, class_weight=None,
cv=None, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
           refit=True, scoring=None, solver='liblinear', tol=0.0001,
           verbose=0).

Array of C used for cross-validation:
[1.00000000e-04 7.74263683e-04 5.99484250e-03 4.64158883e-02
 3.59381366e-01 2.78255940e+00 2.15443469e+01 1.66810054e+02
 1.29154967e+03 1.00000000e+04]

Array of C that maps to the best scores across every class:
[10000.]

Grid of scores obtained during cross-validating each fold:
[[0.81154083 0.81285644 0.82007654 0.82100188 0.82124108 0.82137327
  0.82202163 0.82245597 0.82317987 0.82300361]
 [0.81008555 0.80911615 0.81792259 0.81952776 0.81982362 0.81996211
  0.82040274 0.82065453 0.8208182  0.82097557]
 [0.812596   0.8135717  0.82113811 0.82152209 0.82160393 0.82172983
  0.82220823 0.82233413 0.82207604 0.82217046]]

Mean accuracy score:
0.8192996817342216

Classification report:
             precision    recall  f1-score   support

          0       0.78      0.91      0.84    238291
          1       0.89      0.74      0.81    238292

avg / total       0.83      0.82      0.82    476583
```

2.5 Best Hyperparameters

From the results, it appears that the logistic regression model using a liblinear solver with l2 penalty, with feature-scaling, has the highest accuracy, precision, and recall:

- accuracy: 0.819299
- precision: 0.83
- recall: 0.82

Without feature scaling, the logistic regression model using a liblinear solver with l1 penalty has the best performance:

- accuracy: 0.819032
- precision: 0.83
- recall: 0.82

Since the accuracy is only off by a negligable amount, and feature-scaling takes up additional resources, I will use the logistic regression model using a liblinear solver with l1 penalty to predict the results of the test data.

2.6 Model Evaluation

The chosen model is applied on the testing set to measure its performance. The accuracy, precision, and recall scores shown below suggest that the model is performing quite well.

```
In [428]: y_pred = logr_lib_l1.predict(X_test)
          print(f"Accuracy score: {logr_lib_l1.score(X_test,y_test)}\n")
          print(f"Classification report: \n{classification_report(y_test, y_pred)}")
```

```
Accuracy score: 0.8206236249815091

Classification report:
             precision    recall  f1-score   support

          0       0.78      0.90      0.83    158862
          1       0.88      0.74      0.80    158861

avg / total       0.83      0.82      0.82    317723
```

2.7 Predicting Highest Loan Amount for Approval

The predicted probabilities of being approved or rejected by the lending club based on the independent variables of amount requested, debt-to-income ratio, employment length, loan title (one-hot-encoded), and state (one-hotencoded) can also be obtained from the logistic regression model. By setting a boundary of 0.95 approval as a safe threshold for approval, we can advise customers on the highest loan amount they could potentially borrow by inputting their data into the model, and tweaking the amount requested until the 0.95 approval boundary is achieved.[4]

The dataframe below displays a small snapshot of individual features that resulted in >= 0.95 probability of approval.

In [497]:
```
y_proba_pred = logr_lib_l1.predict_proba(X_test)
X_test_df = pd.DataFrame(X_test).reset_index()
y_proba_pred_df = pd.DataFrame(y_proba_pred).reset_index()

predictions_df = pd.concat([X_test_df,y_proba_pred_df],axis = 1)
predictions_df.drop(columns=['index'], inplace=True)
predictions_df_high_approval = predictions_df[predictions_df[1] >= 0.95]
predictions_df_high_approval.head()
```

Out[497]:

	Amount Requested	Debt-To-Income Ratio	Employment Length	Business	Car
3	12000.0	12.0	10	0	0
19	9000.0	18.0	10	0	0
21	10000.0	14.0	10	0	0
24	25000.0	4.0	10	0	0
25	12000.0	4.0	10	0	0

	Credit Card	Debt Consolidation	Green Loan	House	Major Purchase
3	0	0	0	0	1
19	0	1	0	0	0
21	0	1	0	0	0
24	0	0	0	1	0
25	0	0	0	0	0

	...	TX	UT	VA	VT	WA	WI	WV	WY	0	1
3	...	0	0	0	0	0	0	0	0	0.037430	0.962570
19	...	0	0	1	0	0	0	0	0	0.016704	0.983296
21	...	0	0	0	0	0	0	0	0	0.013274	0.986726
24	...	0	0	0	0	0	0	0	0	0.024434	0.975566
25	...	0	0	0	0	1	0	0	0	0.039848	0.960152

[5 rows x 65 columns]

[4] **#probability:** I used the best logistic regression model out of all the trained models to predict the probability of loan approval on the unseen test set. The probability can be set at a specific threshold (in this case, 0.95), and the various features can then be manipulated to obtain the probability threshold that satisfies our decision boundary. If we had a higher tolerance for loan rejection, we could set the probability threshold to a lower level, e.g. 0.75. Probability is thus used as a way to gauge and predict the highest loan amount that we would safely advise to a customer.

2.7.1 Example Customer

The predictive model is tested on an example customer, Bob, who has a Debt-To-Income Ratio of 3, Employment Length of 7 years, is taking a loan for Debt Consolidation, and lives in California.

While testing the model, I experienced the following behavior:

- for a given set of customer stats, the model plateaus at a fixed probability of approval, insensitive to large amount changes.
- increasing the amount requested increased the probability of the loan being approved.

Some reasons for this behavior are conjectured as follows: - Other variables might play a much more significant role of predicting loan approval or rejection, and so changes in the amount requested do not do much to increase or decrease the customer's chances of getting a loan. - A higher amount requested could also correlate with better customer stats, and so the model might have ended up learning that with a given set of "approvable" stats, a higher requested amount means a higher rate of approval.

As opposed to changing the amount requested, a variable which seems to be extremely responsive in terms of affecting a customer's loan status is the employment length. A higher employment length starkly increases the probability of loan approval and vice versa. This makes intuitive sense since a person with longer employment length would most likely be able to pay off the loan since they have a constant, stable stream of income.

Both predictive behaviors for Bob are demonstrated below.

```
In [680]: bob = {
            'Amount Requested':'10000','Debt-To-Income Ratio':3,'Employment Length':7,
            'Business':0,'Car':0,'Credit Card':0,'Debt Consolidation':1,
            'Green Loan':0,'House':0,'Major
Purchase':0,'Medical':0,'Moving':0,'Other':0,'Vacation':0,
            'AK':0,'AL':0, 'AR':0, 'AZ':0,'CA':1,'CO':0,'CT':0,'DC':0,'DE':0,
            'FL':0,'GA':0,'HI':0,'IL':0,'IN':0,'KS':0,'KY':0,'LA':0,'MA':0,'MD':0,
            'ME':0,'MI':0,'MN':0,'MO':0,'MS':0,'MT':0,'NC':0,'ND':0,'NE':0,'NH':0,
            'NJ':0,'NM':0,'NV':0,'NY':0,'OH':0,'OK':0,'OR':0,'PA':0,'RI':0,'SC':0,
            'SD':0,'TN':0,'TX':0,'UT':0,'VA':0,'VT':0,'WA':0,'WI':0,'WV':0,'WY':0
            }

bob_df = pd.DataFrame(bob, index=[0])

print("With Debt-To-Income Ratio of 3.0, Employment Length 7, Debt Consolidation and
CA:\n")
for i in range(10000,60000,10000):
    bob_df['Amount Requested'] = i
    bob_pred = logr_lib_l1.predict_proba(bob_df)
    print(f"Amount Requested: {i}. Probability of approval: {bob_pred[0][1]}\n")

print("With Amount Requested of 10000, Debt-To-Income Ratio of 3.0, Debt Consolidation
and CA:\n")
for i in range(0,11,2):
    bob_df['Employment Length'] = i
    bob_pred = logr_lib_l1.predict_proba(bob_df)
    print(f"Employment Length: {i}. Probability of approval: {bob_pred[0][1]}\n")
```

```
With Debt-To-Income Ratio of 3.0, Employment Length 7, Debt Consolidation and CA.:

Amount Requested: 10000. Probability of approval: 0.9432501811402284
```

```
Amount Requested: 20000. Probability of approval: 0.9444063842278166

Amount Requested: 30000. Probability of approval: 0.9455403912338444

Amount Requested: 40000. Probability of approval: 0.9466525732485206

Amount Requested: 50000. Probability of approval: 0.9477432973968143

With Amount Requested of 10000, Debt-To-Income Ratio of 3.0, Debt Consolidation and
CA.:

Employment Length: 0. Probability of approval: 0.27943664927698353

Employment Length: 2. Probability of approval: 0.5377687181673515

Employment Length: 4. Probability of approval: 0.777297113653875

Employment Length: 6. Probability of approval: 0.9128232861071167

Employment Length: 8. Probability of approval: 0.9691483057945611

Employment Length: 10. Probability of approval: 0.9895002327730532
```

2.8 Data Visualizations - Feature values which lead to higher probability of approval

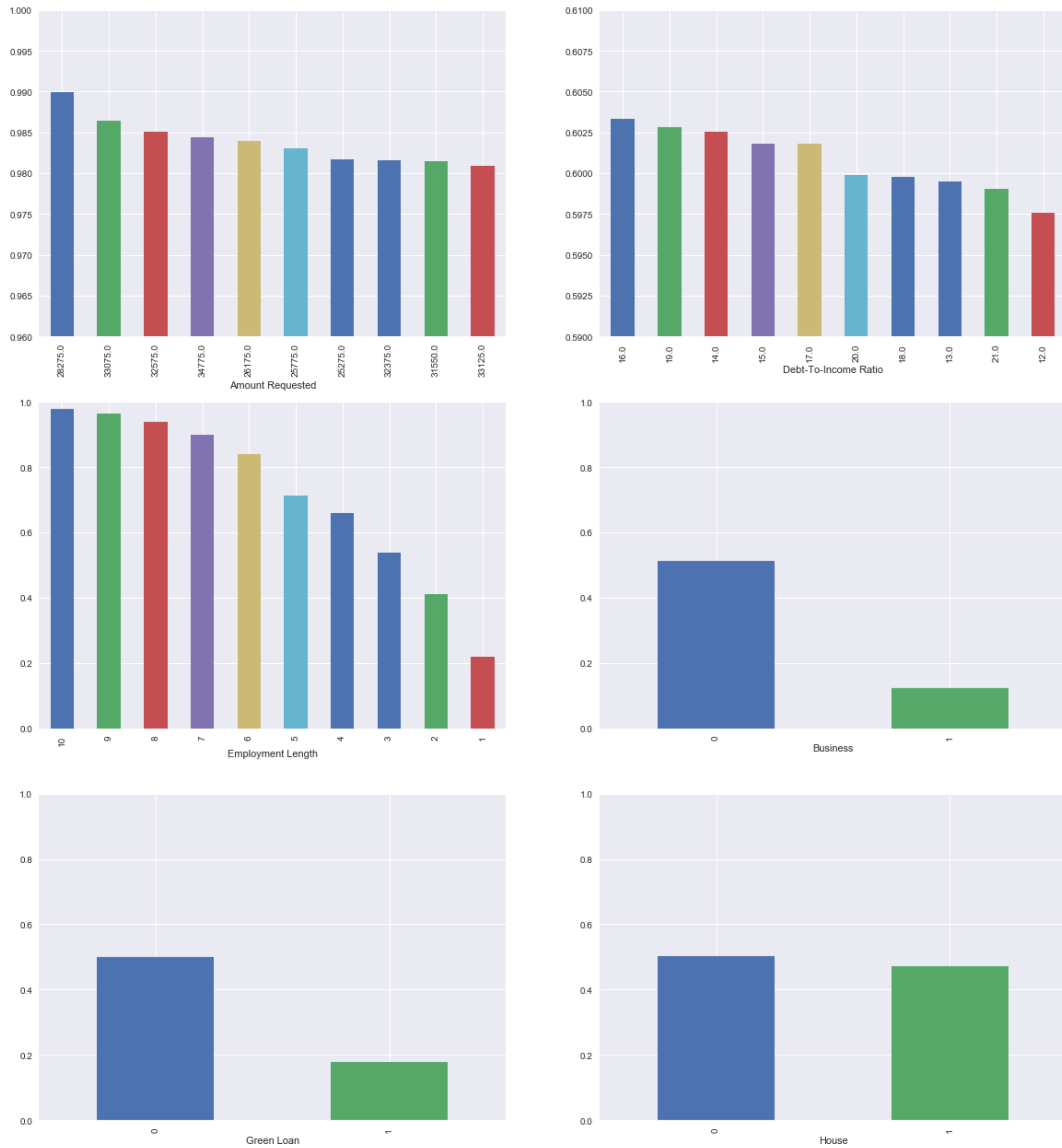
Below are some bar plots visualizing various feature values, ordered by the top 10 highest mean probabilities of approval. While most of the value changes don't appear to lead to significant changes in the probability, there are some categories - most noticeably "Employment Length" - that display a relatively clear relationship between the feature value and the mean probability of being approved. For "Employment Length", we can see that as the "Employment Length" decreases, the probability of being accepted for a loan also decreases.

Only a few feature categories were plotted since this is more for interest and observation vs a rigorous form of analysis; the multidimensional nature of the problem means that plotting single features against the probability will not provide robust results. Also, do note that the x-axis is not in order, and the y-axis has a trimmed scale in order to show the minute differences between the data. For best results, the trained model should be used directly.

```
In [581]: import seaborn as sns
          sns.set()

          plt.figure(figsize=(20,30))

          plt.subplot(4,2,1)
          predictions_df[1].groupby(predictions_df['Amount
          Requested']).mean().nlargest(10).plot.bar(ylim=(0.96,1))

          plt.subplot(4,2,2)
          predictions_df[1].groupby(predictions_df['Debt-To-Income
          Ratio']).mean().nlargest(10).plot.bar(ylim=(0.59,0.61))

          plt.subplot(4,2,3)
          predictions_df[1].groupby(predictions_df['Employment
          Length']).mean().nlargest(10).plot.bar(ylim=(0,1))

          plt.subplot(4,2,4)
```

```
predictions_df[1].groupby(predictions_df['Business']).mean().nlargest(10).plot.bar(ylim=
(0,1))

plt.subplot(4,2,5)
predictions_df[1].groupby(predictions_df['Green
Loan']).mean().nlargest(10).plot.bar(ylim=(0,1))

plt.subplot(4,2,6)
predictions_df[1].groupby(predictions_df['House']).mean().nlargest(10).plot.bar(ylim=(0,
1))
```

Out[581]: <matplotlib.axes._subplots.AxesSubplot at 0x1571b1668>



3 Summary

In this report, I built a logistic regression model that can predict the probability of a loan being approved or rejected based on various features provided by the customer.

3.1 Variables included in the model

The independent variables included in the model were: Amount Requested, Debt-To-Income Ratio, Employment Length, Loan Title (one-hot-encoded), State (one-hot-encoded). The response variables were the status of the loan, with 1 signifying an approved loan and 0 signifying a rejected loan.

3.2 Cleaning and transformations on the data

The following steps were taken to clean and transform the data for training and testing:

- Dropping columns that were not present in both the approval and rejection data.
- Dropping columns with a high amount of irregular values (more than half).
- Dropping rows with NaNs.
- Converting variables to the appropriate data types.
- Dropping categorical variables with a high number of discrete categories (in this case, 998 categories for unique zip codes).
- One-hot-encoding categorical variables that were kept (loan titles, states).
- Balancing the response variable class so that the ratio of approval to rejections is 1.0 in the entire dataset, and the same balanced ratio is transferred into the training and test sets.
- Feature scaling the independent variables to see whether that improves the model's performance.

3.3 The type of model used and model settings

A logistic regression model was used to predict the probability of getting a loan approval since logistic regression is suitable for modeling binary dependent variables. Cross-validation was conducted to finetune the best hyperparameters for the model. The parameters tuned were the Cs, i.e. the inverse of regularization parameter values; the normalization penalty, i.e. L1 or L2; and the type of solver, e.g. liblinear, lbfgs (limited memory Broyden–Fletcher–Goldfarb–Shann algorithm). The final model settings chosen - based on the model's performance on the training data - are C: 2.78, solver: liblinear, penalty: L1.

With feature-scaled variables, the best performing model had the following parameters: C: 10000, solver: liblinear, penalty: L2. The two best performing models (with feature-scaling vs without feature-scaling) only differered by a miniscule amount in terms of accuracy score (0.81929 vs 0.81903 respectively), and had the same average precision and recall scores of 0.83 and 0.82. Since feature-scaling would require additional computational resource, I have decided to use the model that did not require feature-scaling (C: 2.78, liblinear, L1 penalty).[5]

[5]**#regression:** The bulk of this assignment hinges on the development and training of a logistic regression model to predict the probability of loan approval for a given customer based on various independent variables. I provided motivation for the use of logistic regression (modeling binary dependent variables), used cross-validation to finetune various hyperparameters of the model, and trained the model on both feature-scaled and non-feature-scaled data to obtain the model with the best performance to be used on unseen test data. The chosen model is then used to predict

3.4 Training method used, and techniques to avoid overfitting the data

To train the model, the data was split into a training and test set with 0.4 test size and 0.6 training size, i.e. 317723 data points for the test set, and 476583 for the training set. By only performing cross-validation on the training set, overfitting is prevented since the test set remains unseen by the model until the final performance testing stage. The LogisticRegressionCV from the sklearn library is used for cross-validation since is it more efficient than using a brute force search over the parameter space, e.g. GridSearchCV.[6] The LogisticRegressionCV function is used with different solvers and optimizes for C. The best performing model is then chosen to predict results for the test data.

3.5 Estimate of how well the model will perform on unseen data

The model has a relatively good performance on the test data, returning an accuracy of 0.82, average precision of 0.83, and average recall of 0.82. As such, I infer that the model would show a similar performance on a different set of unseen data.

3.6 Assumptions

- The model was only trained on 2015 data, so the performance might be subpar when used on data with a stark difference in time frame.
- The model might be even more accurate with less features (some of these features could just be noise), but this was not tested.
- Some significant features may have been left out during the data cleaning process. As such, it is difficult to determine whether there were confounding features that could be affecting the data. That being said, the model does appear to work well as a predictive model.
- The loan titles were manually categorized by my personal judgement. While I believe that I have categorized them reasonably, I could be unaware that some categories whould be more appropriately categorized together/separated.

the probability of approval for a manually constructed customer.

[6] https://scikit-learn.org/stable/modules/grid_search.html#grid-search

4 Appendix

Code in Jupyter Notebook format is available online at:

https://github.com/hueyning/cs156-ml/blob/master/assignment-2-lending-data/assignment-2-lending-data.ipynb