Yang, Steven. Assignment 2

February 19, 2022

1 Assignment 2

1.1 Steven H. Yang, Minerva University

1.1.1 Lending Club Data

The Lending Club has made an anonymized set of data available for anyone to study here or if you do not have a lending club account the data is also available on Kaggle here. Descriptions of the columns can be found here under 'LoanStats' and 'RejectStats'.

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 fun them.

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

Detailed Instructions Tools Jupyter Notebook, pandas, sklearn

1.1.2 Variables:

Below table are the columns that exist in both data set.

Accepted	Rejected
emp_length	Employment Length
policy_code	Policy Code
zip_code	Zip Code
dti	Debt-To-Income Ratio
addr_state	State
loan_amnt	Amount Requested
title	Loan Title

Risk score is not present in the accepted data set but I use FICO credit score's mean of high/low values. As credit score's purpose is to tell how much an individual is reliable, I think this is the great way to have risk score on accepted data.

As policy code strictly means accepted/rejected, I drop that. Using State address is more relavant that having zip code as it's more intuitive.

Thus the final variables I use are: - Employment Length - Loan Amount - Risk Score - DTI - State

[1]:
```python
import numpy as np
import pandas as pd
pd.set_option('max_rows', 50)
import matplotlib.pyplot as plt
%matplotlib inline

# Define Data Frames
accepted = pd.read_csv('accepted_data.csv')
rejected = pd.read_csv('rejected_data.csv')
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/IPython/core/interactiveshell.py:3444: DtypeWarning: Columns
(0,19,49,59,118,129,130,131,134,135,136,139,145,146,147) have mixed
types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

[2]:
```python
# To see if accepted data is loaded.
accepted.head()
```

[2]:
```
         id  member_id  loan_amnt  funded_amnt  funded_amnt_inv        term  \
0  68407277        NaN     3600.0       3600.0           3600.0   36 months
1  68355089        NaN    24700.0      24700.0          24700.0   36 months
2  68341763        NaN    20000.0      20000.0          20000.0   60 months
3  66310712        NaN    35000.0      35000.0          35000.0   60 months
4  68476807        NaN    10400.0      10400.0          10400.0   60 months

   int_rate  installment grade sub_grade  … hardship_payoff_balance_amount  \
0     13.99       123.03     C        C4  …                            NaN
1     11.99       820.28     C        C1  …                            NaN
2     10.78       432.66     B        B4  …                            NaN
3     14.85       829.90     C        C5  …                            NaN
4     22.45       289.91     F        F1  …                            NaN

  hardship_last_payment_amount disbursement_method  debt_settlement_flag  \
0                          NaN                Cash                     N
1                          NaN                Cash                     N
2                          NaN                Cash                     N
3                          NaN                Cash                     N
4                          NaN                Cash                     N

  debt_settlement_flag_date settlement_status settlement_date  \
0                       NaN               NaN             NaN
1                       NaN               NaN             NaN
2                       NaN               NaN             NaN
3                       NaN               NaN             NaN
4                       NaN               NaN             NaN
```

```
  settlement_amount settlement_percentage settlement_term
0               NaN                   NaN             NaN
1               NaN                   NaN             NaN
2               NaN                   NaN             NaN
3               NaN                   NaN             NaN
4               NaN                   NaN             NaN

[5 rows x 151 columns]
```

[3]:
```
# To see if rejected data is loaded.
rejected.head()
```

[3]:
```
   Amount Requested Application Date                         Loan Title  \
0            1000.0       2007-05-26  Wedding Covered but No Honeymoon
1            1000.0       2007-05-26                 Consolidating Debt
2           11000.0       2007-05-27        Want to consolidate my debt
3            6000.0       2007-05-27                            waksman
4            1500.0       2007-05-27                             mdrigo

   Risk_Score Debt-To-Income Ratio Zip Code State Employment Length  \
0       693.0                  10%    481xx    NM           4 years
1       703.0                  10%    010xx    MA          < 1 year
2       715.0                  10%    212xx    MD            1 year
3       698.0               38.64%    017xx    MA          < 1 year
4       509.0                9.43%    209xx    MD          < 1 year

   Policy Code
0          0.0
1          0.0
2          0.0
3          0.0
4          0.0
```

[4]:
```
# These are the columns that I'll use
final_columns = ["emp_length", "loan_amnt", "risk_score", "dti", "addr_state"]
```

[5]:
```
# New column for risk score in accepted data frame
accepted["risk_score"] = accepted[['fico_range_high', 'fico_range_low']].
 ↪mean(axis=1)

# Rename Columns in Rejected Data Frame
rejected = rejected.rename(columns={'Amount Requested': 'loan_amnt',␣
 ↪'Risk_Score':'risk_score', 'Debt-To-Income Ratio': 'dti', \
   'Zip Code':'zip_code', 'State':'addr_state', 'Employment Length':
 ↪'emp_length', 'Policy Code':'policy_code'})
```

```
# Drop all rows in N/A
rejected = rejected[final_columns].dropna()
accepted = accepted[final_columns].dropna()
```

[6]:
```
#Create New column says accepted or not
accepted["accepted"] = 1
accepted.head()
```

[6]:

	emp_length	loan_amnt	risk_score	dti	addr_state	accepted
0	10+ years	3600.0	677.0	5.91	PA	1
1	10+ years	24700.0	717.0	16.06	SD	1
2	10+ years	20000.0	697.0	10.78	IL	1
3	10+ years	35000.0	787.0	17.06	NJ	1
4	3 years	10400.0	697.0	25.37	PA	1

[7]:
```
#Create New column says accepted or not
rejected["accepted"] = 0
rejected.head()
```

[7]:

	emp_length	loan_amnt	risk_score	dti	addr_state	accepted
0	4 years	1000.0	693.0	10%	NM	0
1	< 1 year	1000.0	703.0	10%	MA	0
2	1 year	11000.0	715.0	10%	MD	0
3	< 1 year	6000.0	698.0	38.64%	MA	0
4	< 1 year	1500.0	509.0	9.43%	MD	0

[8]:
```
# Combine two data in one data frame
data = pd.concat([accepted, rejected], ignore_index=True)
data.head()
```

[8]:

	emp_length	loan_amnt	risk_score	dti	addr_state	accepted
0	10+ years	3600.0	677.0	5.91	PA	1
1	10+ years	24700.0	717.0	16.06	SD	1
2	10+ years	20000.0	697.0	10.78	IL	1
3	10+ years	35000.0	787.0	17.06	NJ	1
4	3 years	10400.0	697.0	25.37	PA	1

Data Cleaning and Transformation All rows with missing data are removed because they potentially affect the model in falwed way. This is a safe choice as the dataset is huge enough and missing some data should not affect the whole model.

Some data includes string variable and they were all converted to the float variables except states.

As combining data, I added the accepted column to differentiate accepted or rejected loans.

Model Selection Here, I try multiple models and will choose the one has the best accuracy among the other models.

I tried: Logistic Regression, LDA, and LogisticRegressionCV.

As LDA performs the best based on the following results, I use LDA as a final model.

1.1.3 Training Method

To avoid the overfitting, I splitted data into training and testing data. I tried different threshold to find the best ratio to split. According to the report, having 40% of test data and 60% of training data shows the best accuracy and I chose to do so. However, I could not test every threshold but did 0.05 discretely. To imporve the model, trying more thrsehold should benefit.

[9]:
```
numbers = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '.']
# Clean string variables to float variables
def str_to_float(x):
    if type(x) == str:
        num = ""
        for i in x:
            if i in numbers:
                num += i
        return float(num)
    else:
        return x

assert (str_to_float("13.35!") == 13.35)

# Change str columns to float columns
data['emp_length'] = data['emp_length'].map(lambda x: str_to_float(x))
data['dti'] = data['dti'].map(lambda x: str_to_float(x))
```

[10]:
```
# Y as a resultant vector
Y = data["accepted"]
# Other variables used for prediction to be X
X = pd.concat([data[['loan_amnt', 'risk_score', 'dti', 'emp_length']], pd.
 ↪get_dummies(data['addr_state'])], axis = 1)
X
```

[10]:
```
          loan_amnt  risk_score    dti  emp_length  AK  AL  AR  AZ  CA  CO  \
0            3600.0       677.0   5.91        10.0   0   0   0   0   0   0
1           24700.0       717.0  16.06        10.0   0   0   0   0   0   0
2           20000.0       697.0  10.78        10.0   0   0   0   0   0   0
3           35000.0       787.0  17.06        10.0   0   0   0   0   0   0
4           10400.0       697.0  25.37         3.0   0   0   0   0   0   0
…                 …           …      …           …  ..  ..  ..  ..  ..  ..
11106219    30000.0       681.0  55.15         1.0   0   0   1   0   0   0
11106220     1000.0       531.0  31.31         1.0   0   0   0   0   0   0
11106221    10000.0       590.0  41.26         1.0   0   0   0   0   0   0
11106222     1200.0       686.0  10.26         1.0   0   0   0   0   1   0
11106223    15000.0       684.0  10.58         1.0   0   0   0   0   0   0

          …  SD  TN  TX  UT  VA  VT  WA  WI  WV  WY
```

```
0         …  0  0  0  0  0  0  0  0  0  0
1         …  1  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
…        …  .. .. .. .. .. .. .. .. .. ..
11106219  …  0  0  0  0  0  0  0  0  0  0
11106220  …  0  0  1  0  0  0  0  0  0  0
11106221  …  0  0  0  0  0  0  0  0  0  0
11106222  …  0  0  0  0  0  0  0  0  0  0
11106223  …  0  0  0  0  0  0  0  0  0  0

[11106224 rows x 55 columns]
```

[11]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
# Test different thresholds to optimize the accuracy
thresholds = [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65,␣
↪0.7, 0.75, 0.8, 0.85, 0.9, 0.95]
threshold = 0
max_accuracy = 0

# Generate Train/Test sets
(train_inputs, test_inputs, train_classes, test_classes) = train_test_split(X,␣
↪Y, test_size=0.25, shuffle=True)
(train_clf_inputs, val_inputs, train_clf_classes, val_classes) =␣
↪train_test_split(train_inputs, train_classes, \
    test_size=0.5, shuffle=True)

clf = LogisticRegression(max_iter=400)
clf.fit(train_clf_inputs, train_clf_classes)

for i in thresholds:
    # Try different Thresholds to see what's the optimal threshold
    predictions = np.where(clf.predict_proba(val_inputs)[:,1] > i, 1, 0)
    report = classification_report(val_classes, predictions, output_dict=True)
    if report['accuracy'] > max_accuracy:
        threshold = i
        max_accuracy = report['accuracy']
print("Optimal threshold", threshold)
print("Max Accuracy", max_accuracy)
```

```
Optimal threshold 0.4
Max Accuracy 0.8836364186423756
```

[12]:

```
# Detailed Report of the optimal threshold model
(train_inputs, test_inputs, train_classes, test_classes) = train_test_split(X,␣
 ↪Y, test_size=0.25, shuffle=True)
clf = LogisticRegression(max_iter=400)
clf.fit(train_inputs, train_classes)
predictions = np.where(clf.predict_proba(test_inputs)[:,1] > 0.4, 1, 0)
report = classification_report(test_classes, predictions)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.90      0.94      0.92   2247250
           1       0.68      0.55      0.61    529306

    accuracy                           0.87   2776556
   macro avg       0.79      0.75      0.76   2776556
weighted avg       0.86      0.87      0.86   2776556
```

[13]:

```
# Using a Linear Discriminant Analysis model and comparing accuracy with the␣
 ↪Logistic Regression model
lda = LinearDiscriminantAnalysis()
lda.fit(train_inputs, train_classes)
predictions = lda.predict(test_inputs)
print(classification_report(test_classes, predictions))
```

```
              precision    recall  f1-score   support

           0       0.90      0.96      0.93   2247250
           1       0.75      0.56      0.65    529306

    accuracy                           0.88   2776556
   macro avg       0.83      0.76      0.79   2776556
weighted avg       0.87      0.88      0.88   2776556
```

The report shows that the LDA model has a .88 accuracy. Thefore, I expect that the model is 88% accurate for unseen data.

[140]:

```
# This function will get the input and find the maximum loan that can␣
 ↪potentially be accepted.
def maximum_loan(x):
    loan = 5000
    x["loan_amnt"] = loan
    x = np.array(x).reshape(1, -1)
    prediction = lda.predict(x)
    if prediction == 1:
        while prediction == 1:
```

```python
            # Keep increase the loan amount and try
            loan += 500
            x[0, 0] = loan
            prediction = lda.predict(x)
        return loan - 500
    elif prediction == 0:
        while prediction == 0 and loan >= 0:
            # Keep decrease the loan amount and try
            loan -= 500
            x[0, 0] = loan
            prediction = lda.predict(x)
        return loan + 500
```

[142]:
```python
#Testing on some data
customer = test_inputs.sample(1)
print(customer)
print(customer.index)
print("Maximum Loan Predicted for a Sampled Customer:", maximum_loan(customer))
```

```
         loan_amnt  risk_score   dti  emp_length  AK  AL  AR  AZ  CA  CO  …  \
1619796    19475.0       677.0  10.8         5.0   0   0   0   0   0   0  …

         SD  TN  TX  UT  VA  VT  WA  WI  WV  WY
1619796   0   0   0   0   0   0   1   0   0   0

[1 rows x 55 columns]
Int64Index([1619796], dtype='int64')
Largest Loan that will be successfully funded: 23000
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
```

```
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
```

```
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
```

```
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LinearDiscriminantAnalysis was fitted with feature names
  warnings.warn(
```

[14]:
```
#Function that computes the largest loan a person can take that will be␣
 ↪successfully funded
def maximum_loan_clf(x):
    loan = 5000
    x["loan_amnt"] = loan
    x = np.array(x).reshape(1, -1)
    prediction = clf.predict(x)
    if prediction == 1:
        while prediction == 1:
```

```
            #Continue increasing the loan amount until it reaches the decision␣
 ↪boundary
            loan += 500
            x[0, 0] = loan
            prediction = clf.predict(x)
        return loan - 500
    elif prediction == 0:
        while prediction == 0 and loan >= 0:
            #Continue decreasing the loan amount until it reaches the decision␣
 ↪boundary or gets to 0
            loan -= 500
            x[0, 0] = loan
            prediction = clf.predict(x)
        return loan + 500
```

[15]:
```
#Testing on some data
customer = test_inputs.sample(1)
print(customer)
print(customer.index)
print("Maximum Loan Predicted for a Sampled Customer:",␣
 ↪maximum_loan_clf(customer))
```

```
         loan_amnt  risk_score    dti  emp_length  AK  AL  AR  AZ  CA  CO  \
6968635    10000.0       487.0  24.96         1.0   0   0   0   0   0   0

         …  SD  TN  TX  UT  VA  VT  WA  WI  WV  WY
6968635  …   0   0   0   0   1   0   0   0   0   0

[1 rows x 55 columns]
Int64Index([6968635], dtype='int64')
Maximum Loan Predicted for a Sampled Customer: 0
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/base.py:450: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
```

1.1.4 Based on the test result above:

One individual who makes $19475.0 can maximum of $23,000 of loan as maximum.

[ ]:
```
## Bonus Trial

from sklearn.linear_model import LogisticRegressionCV
# randomize and split to reduce overusage of data but preserving data␣
↪characteristics
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.95,␣
↪random_state = 123)
# initiate classification model with cross-validation parameters to allow for␣
↪best parameter choice
cla = LogisticRegressionCV(max_iter=400)
```

```
# fit split training data to model.
cla.fit(X_train, Y_train)
# classification report to guage the accuracy on unseen data
print(classification_report(Y_test[:1000000], cla.predict(X_test.iloc[:
 ↪1000000])))
```

```
              precision    recall  f1-score   support

           0       0.90      0.96      0.93    809352
           1       0.77      0.54      0.63    190648

    accuracy                           0.88   1000000
   macro avg       0.83      0.75      0.78   1000000
weighted avg       0.87      0.88      0.87   1000000
```

```
[ ]:
```