Assignment 2

Data cleaning

Load data and find freature present in both approved and rejected data.

In [1]:
```
import pandas as pd

approved_csv = pd.read_csv("LoanStats3a.csv", skiprows=1)
rejected_csv = pd.read_csv("RejectStatsA.csv", skiprows=1)

### Mapping ###
# 'loan_amnt' - 'Amount Requested'
# 'issue_d' - 'Application Date'
# 'purpose', 'title' - 'Loan Title'
# '' - 'Risk_Score' (no match)
# 'dti' - 'Debt-To-Income Ratio'
# 'zip_code' - 'Zip Code'
# 'addr_state' - 'State'
# 'policy_code' - 'Policy Code'
# 'emp_length' - 'Employment Length'
approved_raw_data = approved_csv[['loan_amnt','issue_d', 'purpose', 'dt
i', 'zip_code',
                                  'addr_state', 'policy_code','emp_lengt
h']]
rejected_raw_data = rejected_csv[['Amount Requested','Application Date',
 'Loan Title',
                                  'Debt-To-Income Ratio', 'Zip Code', 'S
tate',
                                  'Policy Code', 'Employment Length']]
rejected_raw_data.columns = ['loan_amnt','issue_d', 'purpose', 'dti', 'z
ip_code',
                             'addr_state', 'policy_code', 'emp_length']
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pa
ckages/IPython/core/interactiveshell.py:2785: DtypeWarning: Columns (0,
47) have mixed types. Specify dtype option on import or set low_memory=
False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Count the number of entries avaliable in each set, and display a small sample.

In [2]:
```
print ("The dataset contain {:d} approved data.".format(approved_raw_dat
a.shape[0]))
print ("The dataset contain {:d} rejected data.".format(rejected_raw_dat
a.shape[0]))

print ("\n")
print ("Sample from approved data:")
print (approved_raw_data.sample(5))
print ("\n")
print ("Sample from rejected data:")
print (rejected_raw_data.sample(5))
```

```
The dataset contain 42538 approved data.
The dataset contain 755491 rejected data.


Sample from approved data:
       loan_amnt   issue_d             purpose    dti zip_code addr_sta
te  \
31404     6250.0  Mar-2010  debt_consolidation  18.04    342xx         
FL  
25328     5000.0  Sep-2010  debt_consolidation  16.78    010xx         
MA  
11346     3050.0  Jul-2011  debt_consolidation  16.65    799xx         
TX  
21971    11500.0  Dec-2010  debt_consolidation  15.32    923xx         
CA  
32074     8000.0  Feb-2010      small_business   1.65    770xx         
TX  

       policy_code emp_length  
31404          1.0  10+ years  
25328          1.0    8 years  
11346          1.0   < 1 year  
21971          1.0    5 years  
32074          1.0     1 year  


Sample from rejected data:
        loan_amnt     issue_d             purpose     dti zip_code addr
_state  \
3670      25000.0  2007-12-02           2Elisaiah   34.2%    300xx     
    GA  
748504    25000.0  2012-12-26         credit_card  10.18%    761xx     
    TX  
557466    25000.0  2012-07-07  debt_consolidation  19.41%    785xx     
    TX  
336713     3000.0  2011-09-03                 car   51.8%    352xx     
    AL  
83793     10000.0  2009-12-01                          0%    412xx     
    KY  

        policy_code emp_length  
3670              0    5 years  
748504            0   < 1 year  
557466            0   < 1 year  
336713            0   < 1 year  
83793             0  10+ years  
```

Count how may entries have missing value for both set.

In [3]:
```
nan_count = pd.DataFrame({"approved": approved_raw_data.isnull().sum(),
                          "approved %": approved_raw_data.isnull().sum()
 / len(approved_raw_data),
                          "rejected": rejected_raw_data.isnull().sum(),
                          "rejected %": rejected_raw_data.isnull().sum()
 / len(rejected_raw_data)})
print (nan_count)
```

```
             approved  approved %  rejected  rejected %
loan_amnt           3    0.000071         0    0.000000
issue_d             3    0.000071         0    0.000000
purpose             3    0.000071        14    0.000019
dti                 3    0.000071         0    0.000000
zip_code            3    0.000071        22    0.000029
addr_state          3    0.000071        21    0.000028
policy_code         3    0.000071         0    0.000000
emp_length       1115    0.026212      8130    0.010761
```

In general, the dataset is relatively complet, with most of the missing value occur in the 'emp_length' feature. However, it is still less than 3% of missing value for the approved set and about 1% for the rejected set. This means it is relatively safe to simply drop the entries with NaN value with low possibility of introducing a significant bias.

In [4]:
```
approved_raw_data = approved_raw_data.dropna()
rejected_raw_data = rejected_raw_data.dropna()
```

Data processing

Feature 'issue_d' and 'dti' are coded in different format in the approved and rejected set. We will need to standardize the format.

In [5]:
```
import copy

# deep copy raw data into intermediate processing set 0
approved_data_0 = copy.deepcopy(approved_raw_data)
rejected_data_0 = copy.deepcopy(rejected_raw_data)
```

In [6]:
```
from datetime import datetime
import numpy as np

def date_string_to_datetime(row):
    try:    # for approved dataset
        return datetime.strptime(row['issue_d'], "%b-%Y")
    except:    # for rejected dataset
        return datetime.strptime(row['issue_d'], "%Y-%m-%d")

def extract_year_lambda(row):
    return date_string_to_datetime(row).year

def extract_month_lambda(row):
    return date_string_to_datetime(row).month

def dti_percent_to_decimal(row):
    return np.float64(row['dti'].strip("%"))


approved_data_0['issue_y'] = approved_data_0.apply(extract_year_lambda,
axis=1)
approved_data_0['issue_m'] = approved_data_0.apply(extract_month_lambda,
 axis=1)
approved_data_0 = approved_data_0.drop(columns=['issue_d'])

rejected_data_0['issue_y'] = rejected_data_0.apply(extract_year_lambda,
axis=1)
rejected_data_0['issue_m'] = rejected_data_0.apply(extract_month_lambda,
 axis=1)
rejected_data_0['dti'] = rejected_data_0.apply(dti_percent_to_decimal, a
xis=1)
rejected_data_0 = rejected_data_0.drop(columns=['issue_d'])
```

In [7]:
```python
print ("Sample from approved data:")
print (approved_data_0.sample(5))
print ("\n")
print ("Sample from rejected data:")
print (rejected_data_0.sample(5))
```

```
Sample from approved data:
       loan_amnt             purpose    dti zip_code addr_state  policy
_code  \
34453    25000.0             wedding   9.96    070xx         NJ
  1.0
39199    15000.0  debt_consolidation  15.10    275xx         NC
  1.0
40935    15000.0  debt_consolidation  20.44    334xx         FL
  1.0
28389    14000.0  debt_consolidation  20.98    750xx         TX
  1.0
17704     2400.0      major_purchase  18.47    749xx         OK
  1.0

      emp_length  issue_y  issue_m
34453    4 years     2009       11
39199   < 1 year     2008        3
40935    8 years     2009       10
28389    2 years     2010        7
17704  10+ years     2011        3


Sample from rejected data:
        loan_amnt                  purpose    dti zip_code addr_state
\
414756     3000.0                    other   6.79    583xx         MN

209994    25000.0           small_business   2.67    786xx         TX

390181     3000.0                      car   0.00    300xx         GA

503258    25000.0  Debt Consolidation Loan  14.76    303xx         GA

156202    25000.0       debt_consolidation  83.47    127xx         NY


        policy_code emp_length  issue_y  issue_m
414756            0   < 1 year     2011       12
209994            0   < 1 year     2011        1
390181            0   < 1 year     2011       11
503258            0    9 years     2012        5
156202            0   < 1 year     2010        9
```

Feature 'purpose' is a string, containing discription of the intened use of the loan. We need to find a way to represent that data in a machine-friendly way.

In [8]:
```
# in the approved set, the purpose data is well organized label
print ("'purpose' feature count in approved dataset:")
print (approved_data_0[['purpose', 'policy_code']].groupby('purpose').co
unt())
print ("\n")

# in the rejected set, the purpose data is messy text string
print ("Sample of 'purpose' feature count in rejected dataset:")
print (rejected_data_0[['purpose', 'policy_code']].groupby('purpose').co
unt().sample(10))
```

```
'purpose' feature count in approved dataset:
                    policy_code
purpose
car                        1563
credit_card                5344
debt_consolidation        19363
educational                 413
home_improvement           3099
house                       412
major_purchase             2238
medical                     726
moving                      603
other                      4259
renewable_energy             98
small_business             1946
vacation                    368
wedding                     991


Sample of 'purpose' feature count in rejected dataset:
                                                      policy_code
purpose
Tradition pay-off                                               1
Business Supplies                                               1
Loan for student loand and paying bills                         1
BE ABLE TO BE BACK ON TRACK.............                        1
For College                                                     2
terrylou                                                        1
Cadillac 96                                                     1
Need Quick-In Escrow                                            1
Replace Privacy Fence and Update Kitchen                        1
looking to expand                                               1
```

To clean the 'purpose' feature data in the rejected set, we will use KNN to cluster the text strings in rejected set to the labels in approved set.

In [9]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
import numpy as np

np.random.seed(5)

# extract 'purpose' feature data for both sets
rejected_purpose_text = list(rejected_data_0['purpose'])
approved_purpose_label = list(approved_data_0.groupby('purpose').groups.
keys())

# build knn cluster model using all text in the rejected set
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(rejected_purpose_text)
true_k = 14
knn_cluster_model = KMeans(n_clusters=true_k, init='k-means++', max_iter
=100, n_init=1)
knn_cluster_model.fit(X)

# print a summary of the clusters
print ("Top terms in each cluster:")
order_centroids = knn_cluster_model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range (true_k):
    print ("Cluster {:d}: {:s}, {:s}, {:s}".format(* [i] + [terms[ind] f
or ind in order_centroids[i, :3]] ))

# map the approved set label to the cluster model using predict methord
Y = vectorizer.transform(approved_purpose_label)
prediction = knn_cluster_model.predict(Y)
label_mapping = pd.DataFrame({'original_label': approved_purpose_label,
'predicted_cluster': prediction,
                              'predicted_cluster_top_label': [terms[orde
r_centroids[c][0]] for c in prediction]})
print ("\n")
print ("Predicted mapping between approved set 'purpose' label and rejec
ted set 'purpose' text text")
print (label_mapping)
unmatched_cluster = list(set(range(14)) - set(prediction))
print ("Unmatched clusters:", unmatched_cluster)
print ("With the top label in cluster being:", [terms[order_centroids[c]
[0]] for c in unmatched_cluster])
```

```
Top terms in each cluster:
Cluster 0: moving, loan, expenses
Cluster 1: debt_consolidation, factoring, facelift
Cluster 2: home_improvement, loan, zzzzzgirl
Cluster 3: credit_card, loan, zzzzzgirl
Cluster 4: loan, consolidation, debt
Cluster 5: major_purchase, loan, zzzzzgirl
Cluster 6: medical, expenses, loan
Cluster 7: car, loan, need
Cluster 8: small_business, loan, zzzzzgirl
Cluster 9: house, loan, buy
Cluster 10: wedding, loan, expenses
Cluster 11: vacation, loan, dream
Cluster 12: consolidate, debt, credit
Cluster 13: bills, pay, medical


Predicted mapping between approved set 'purpose' label and rejected set
'purpose' text text
        original_label  predicted_cluster predicted_cluster_top_label
0                  car                  7                         car
1          credit_card                  3                 credit_card
2   debt_consolidation                  1          debt_consolidation
3          educational                  4                        loan
4     home_improvement                  2            home_improvement
5                house                  9                       house
6       major_purchase                  5              major_purchase
7              medical                  6                     medical
8               moving                  0                      moving
9                other                  4                        loan
10    renewable_energy                  4                        loan
11      small_business                  8              small_business
12            vacation                 11                    vacation
13             wedding                 10                     wedding
Unmatched clusters: [12, 13]
With the top label in cluster being: ['consolidate', 'bills']
```

By applying KNN clustering on the rejected set, we are able to map most of the label in the approved set to the cluster of the rejected set. The exceptions we have are for the 'educational', 'other', and 'renewable_energy' label. However, as 'educational' and 'renewable_energy' are relatively small group (413 and 98 entries compare to the over 40k dataset) and 'other' simply means other, we can group these labels together as 'other'. For the unmatched cluster 12 and 13, its top keyword being 'consolidate' and 'bills' also fit to the description of 'other'.

In [10]:
```
import copy

# deep copy intermediate processing set 0 into intermediate processing s
et 1
approved_data_1 = copy.deepcopy(approved_data_0)
rejected_data_1 = copy.deepcopy(rejected_data_0)
```

In [11]:
```
import swifter

def text_to_cluster_code(row):
    text = vectorizer.transform([row['purpose']])
    cluster_code = knn_cluster_model.predict(text)[0]
    #cluster_code = cluster_code[0]
    if cluster_code is 12 or cluster_code is 13:
        cluster_code = 4
    return cluster_code

approved_data_1['purpose'] = approved_data_1.swifter.apply(text_to_cluster_code, axis=1)
rejected_data_1['purpose'] = rejected_data_1.swifter.apply(text_to_cluster_code, axis=1)
```

```
Pandas Apply: 100%|██████████| 41423/41423 [00:22<00:00, 1829.03it/s]
Pandas Apply: 100%|██████████| 747325/747325 [07:31<00:00, 1654.33it/s]
```

In [12]:
```
print ("Sample from approved data:")
print (approved_data_1.sample(5))
print ("\n")
print ("Sample from rejected data:")
print (rejected_data_1.sample(5))
```

```
Sample from approved data:
       loan_amnt  purpose    dti zip_code addr_state  policy_code emp_l
ength  \
37835     5650.0        1   6.18    927xx         CA          1.0  10+
years
31779     5000.0        1   8.93    107xx         NY          1.0  10+
years
23927    20000.0        2  11.09    189xx         PA          1.0    5
years
3904      6000.0       11  10.98    606xx         IL          1.0     1
year
17841     6000.0        8  13.06    487xx         MI          1.0     1
year

       issue_y  issue_m
37835     2009        1
31779     2010        3
23927     2010       11
3904      2011       11
17841     2011        4


Sample from rejected data:
        loan_amnt  purpose    dti zip_code addr_state  policy_code emp_
length  \
518432    15000.0        1  14.00    280xx         NC            0    8
years
141699    30000.0        1  39.45    154xx         PA            0  10+
years
495626    25000.0        5   5.78    212xx         MD            0    <
1 year
629279     5000.0        1   7.87    549xx         WI            0    <
1 year
737255    25000.0        4  67.79    553xx         MN            0    <
1 year

        issue_y  issue_m
518432     2012        5
141699     2010        7
495626     2012        4
629279     2012        9
737255     2012       12
```

Feature 'emp_length' now is a categorical data, however, as the nature of the data is numerical, we will convert the it into numerical data. Feature 'purpose', 'addr_state', 'issue_y', and 'issue_m' are categorical data, and we will represent them using dummy variables. Feature 'zip_code' is between a categorical and numerical data, and is very difficuly to use directly as a feature without more pre-processing. We will drop it for simplicity purpose. In addition, we will normalize the data

In [35]:
```
import copy

# deep copy intermediate processing set 1 into intermediate processing s
et 2
approved_data_2 = copy.deepcopy(approved_data_1)
rejected_data_2 = copy.deepcopy(rejected_data_1)

# at this point, the remaining job is only to clean up, we can join the
 two sets
# policy_code=1 represent approved, policy_code=0 represent rejected
data = pd.concat([approved_data_2, rejected_data_2])
```

In [36]:
```
from sklearn import preprocessing

# Convert originally categorical data employment length to numerical dat
a
data['emp_length'].replace(to_replace='[^0-9]+', value='', inplace=True,
                                regex=True)
data['emp_length'] = data['emp_length'].astype(int)

# generate dummy variables for 'purpose', 'addr_state', 'issue_y', and
 'issue_m'
data = pd.get_dummies(data, columns=['purpose', 'addr_state', 'issue_y',
 'issue_m'])

# drop 'zip_code' feature
data = data.drop(columns=['zip_code'])

# normalize the data
column_name = list(data.columns.values)
min_max_scaler = preprocessing.MinMaxScaler()
scaled = min_max_scaler.fit_transform(data.values)
data = pd.DataFrame(scaled)
data.columns = column_name

print (data.sample(5))
```

```
        loan_amnt           dti  policy_code  emp_length  purpose_0  \
701750   0.003930  4.865997e-07          0.0    0.000000        0.0
288324   0.002501  2.105999e-07          0.0    0.000000        0.0
726844   0.001072  2.111999e-07          0.0    0.000000        0.0
18120    0.003215  3.799998e-08          1.0    0.111111        0.0
598842   0.010593  3.499998e-07          0.0    0.000000        0.0

        purpose_1  purpose_2  purpose_3  purpose_4  purpose_5     ...
    \
701750        1.0        0.0        0.0        0.0        0.0     ...

288324        0.0        1.0        0.0        0.0        0.0     ...

726844        0.0        0.0        0.0        1.0        0.0     ...

18120         0.0        0.0        0.0        1.0        0.0     ...

598842        1.0        0.0        0.0        0.0        0.0     ...


        issue_m_3  issue_m_4  issue_m_5  issue_m_6  issue_m_7  issue_m_
8  \
701750        0.0        0.0        0.0        0.0        0.0        0.
0
288324        0.0        1.0        0.0        0.0        0.0        0.
0
726844        0.0        0.0        0.0        0.0        0.0        0.
0
18120         1.0        0.0        0.0        0.0        0.0        0.
0
598842        0.0        0.0        0.0        0.0        1.0        0.
0

        issue_m_9  issue_m_10  issue_m_11  issue_m_12
701750        0.0         1.0         0.0         0.0
288324        0.0         0.0         0.0         0.0
726844        0.0         0.0         1.0         0.0
18120         0.0         0.0         0.0         0.0
598842        0.0         0.0         0.0         0.0

[5 rows x 87 columns]
```

Modeling

As this is a classification problem, we will use a logistics regression model to predict the.

In [39]:
```
from sklearn.model_selection import train_test_split

Y = data['policy_code']
data = data.drop(columns=['policy_code'])
X = data.values

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.
25, random_state=10)
```

In [51]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# fit training data into a logistic classification model
logistic_predict_model = LogisticRegression()
logistic_predict_model.fit(X_train, Y_train)

# predict on the test set data
Y_predicted = logistic_predict_model.predict(X_test)
print ("The accuracy score is {:.3f}".format(accuracy_score(Y_test, Y_pr
edicted)))
print ("\n")
print ("Classification report:")
print(classification_report(Y_test, Y_predicted))
```

```
The accuracy score is 0.952


Classification report:
             precision    recall  f1-score   support

        0.0       0.96      0.99      0.97    186823
        1.0       0.60      0.24      0.34     10364

avg / total       0.94      0.95      0.94    197187
```

In [52]:
```python
print('Coefficients:',logistic_predict_model.coef_)
```

```
Coefficients: [[-2.58051647e+01 -3.54684879e-01  3.09539972e+00  3.2724
6782e-01
   1.14629195e+00  9.84096151e-01  1.65901588e+00 -1.76462904e+00
   7.72351852e-01  2.88352646e-01 -4.47197072e-01  8.80353344e-01
   4.69619091e-01  1.00298267e+00  2.33867661e-01 -4.65560058e+00
  -4.47003530e+00 -1.53562231e-01 -4.12771229e-01 -5.67588253e-01
   1.92936011e-01  3.63120395e-01  1.94059221e-01  2.46126351e-01
   1.16385432e+00 -1.30317199e-01  9.10420443e-02  3.11294771e-02
  -9.98486858e-02 -3.65153215e-01  4.65879725e-01 -3.87831414e-03
  -5.62932220e-01  5.42397008e-02 -5.24328787e-01 -2.52061395e-01
   4.23295145e-01  1.72678318e-01 -7.81866387e-01 -3.41373437e-01
   2.12479710e-01 -7.66000833e-02 -1.39390539e-01 -1.85593633e-01
  -8.98732936e-03 -6.31446986e-01 -6.52292513e-01 -1.65434666e-01
   2.96391522e-01 -2.97814470e-01  1.77753169e-01  2.55873211e-01
  -2.10861142e-01 -3.30239462e-01  2.93625517e-01 -1.59577206e-01
   4.49882534e-02 -2.26050111e-01 -1.88028899e-01  6.42011192e-02
  -6.51706211e-02  3.04513569e-02  1.83800988e-01 -6.18882104e-01
   2.37913079e-01 -2.16026615e-01 -3.94116171e-01 -6.92869542e-03
   1.21652439e+00  1.05640760e+00  9.95355022e-01  2.61158352e-01
   4.64244236e-01 -7.56697356e+00 -4.59246339e-01 -4.31691302e-01
  -5.59164453e-01 -4.47478179e-01  1.21997589e-01 -2.39644519e-01
  -2.50420161e-01 -2.19795049e-01 -2.20250913e-01 -2.64192117e-01
  -3.51086354e-01 -2.52312166e-01]]
```